## Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 133 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers + Peripheral Control Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- Non volatile Program and Data Memories
  - 128K Bytes of In-System Reprogrammable Flash
    Endurance: 10,000 Write/Erase Cycles
  - Optional Boot Code Section with Independent Lock Bits
    Selectable Boot Size: 1K Bytes, 2K Bytes, 4K Bytes or 8K Bytes
    In-System Programming by On-Chip Boot Program (CAN, UART)
    True Read-While-Write Operation
  - 4K Bytes EEPROM (Endurance: 100,000 Write/Erase Cycles)
  - 4K Bytes Internal SRAM
  - Up to 64K Bytes Optional External Memory Space
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Programming Flash (Hardware ISP), EEPROM, Lock & Fuse Bits
  - Extensive On-chip Debug Support
- CAN Controller 2.0A & 2.0B
  - 15 Full Message Objects with Separate Identifier Tags and Masks
  - Transmit, Receive, Automatic Reply and Frame Buffer Receive Modes
  - 1Mbits/s Maximum Transfer Rate at 8 MHz
  - Time stamping, TTC & Listening Mode (Spying or Autobaud)
- Peripheral Features
  - Programmable Watchdog Timer with On-chip Oscillator
  - 8-bit Synchronous Timer/Counter-0
    10-bit Prescaler
    External Event Counter
    Output Compare or 8-bit PWM Output
  - 8-bit Asynchronous Timer/Counter-2
    10-bit Prescaler
    External Event Counter
    Output Compare or 8-Bit PWM Output
    32Khz Oscillator for RTC Operation
  - Dual 16-bit Synchronous Timer/Counters-1 & 3
    10-bit Prescaler
    Input Capture with Noise Canceler
    External Event Counter
    3-Output Compare or 16-Bit PWM Output
    Output Compare Modulation
  - 8-channel, 10-bit SAR ADC
    8 Single-ended channels
    7 Differential Channels
    2 Differential Channels With Programmable Gain at 1x, 10x, or 200x
  - On-chip Analog Comparator
  - Byte-oriented Two-wire Serial Interface
  - Dual Programmable Serial USART
  - Master/Slave SPI Serial Interface
    Programming Flash (Hardware ISP)
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - 8 External Interrupt Sources
  - 5 Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down & Standby
  - Software Selectable Clock Frequency
  - Global Pull-up Disable
- I/O and Packages
  - 53 Programmable I/O Lines
  - 64-lead TQFP and 64-lead QFN
- Operating Voltages
  - 2.7 - 5.5V
- Operating temperature
  - Industrial (-40°C to +85°C)
- Maximum Frequency
  - 8 MHz at 2.7V - Industrial range
  - 16 MHz at 4.5V - Industrial range

# 8-bit AVR® Microcontroller with 128K Bytes of ISP Flash and CAN Controller

## AT90CAN128

Rev. 4250E–CAN–12/04

## Description

The AT90CAN128 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the AT90CAN128 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

The AVR core combines a rich instruction set with 32 general purpose working registers. All 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The AT90CAN128 provides the following features: 128K bytes of In-System Programmable Flash with Read-While-Write capabilities, 4K bytes EEPROM, 4K bytes SRAM, 53 general purpose I/O lines, 32 general purpose working registers, a CAN controller, Real Time Counter (RTC), four flexible Timer/Counters with compare modes and PWM, 2 USARTs, a byte oriented Two-wire Serial Interface, an 8-channel 10-bit ADC with optional differential input stage with programmable gain, a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, IEEE std. 1149.1 compliant JTAG test interface, also used for accessing the On-chip Debug system and programming and five software selectable power saving modes.

The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI/CAN ports and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or Hardware Reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except Asynchronous Timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the Crystal/Resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption.

The device is manufactured using Atmel's high-density nonvolatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The boot program can use any interface to download the application program in the application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel AT90CAN128 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The AT90CAN128 AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits.

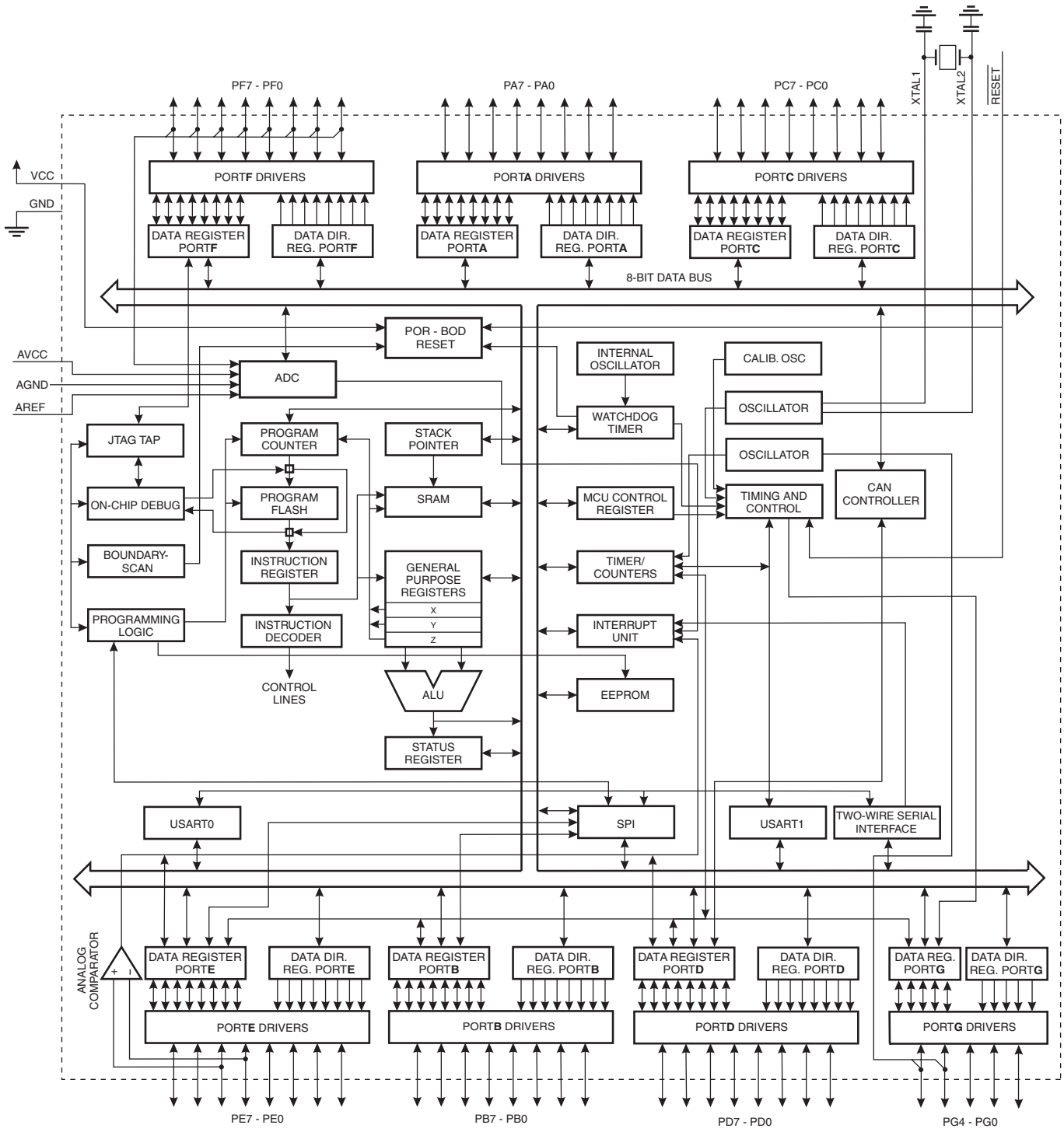
Applications that use the ATmega128 AVR microcontroller can be made compatible to use the AT90CAN128, refer to Application Note AVR 096, on the Atmel web site.

## Disclaimer

Typical values contained in this datasheet are based on simulations and characterization of other AVR microcontrollers manufactured on the same process technology. Min and Max values will be available after the device is characterized.

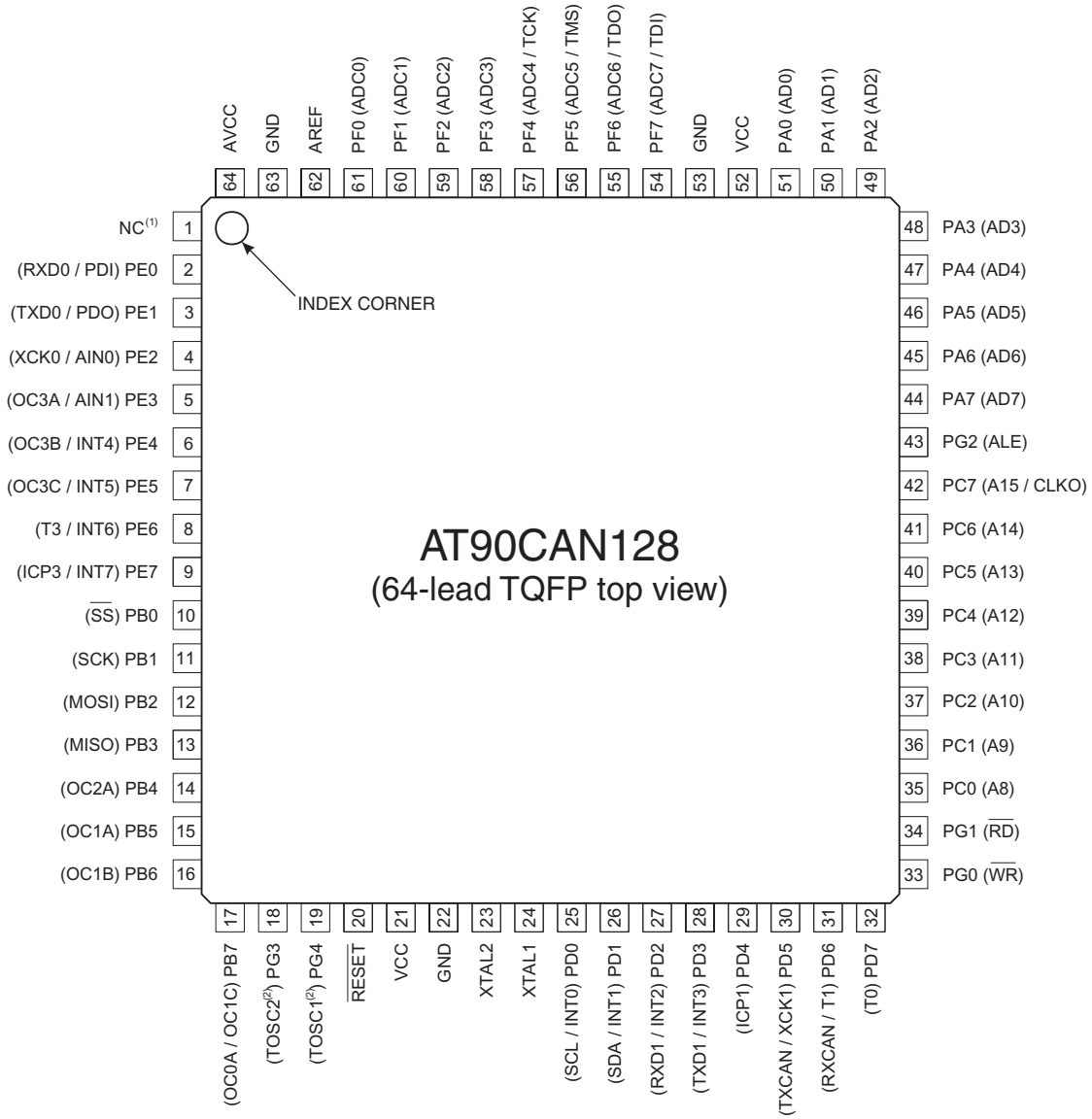
## Block Diagram

**Figure 1.** Block Diagram



## Pin Configurations

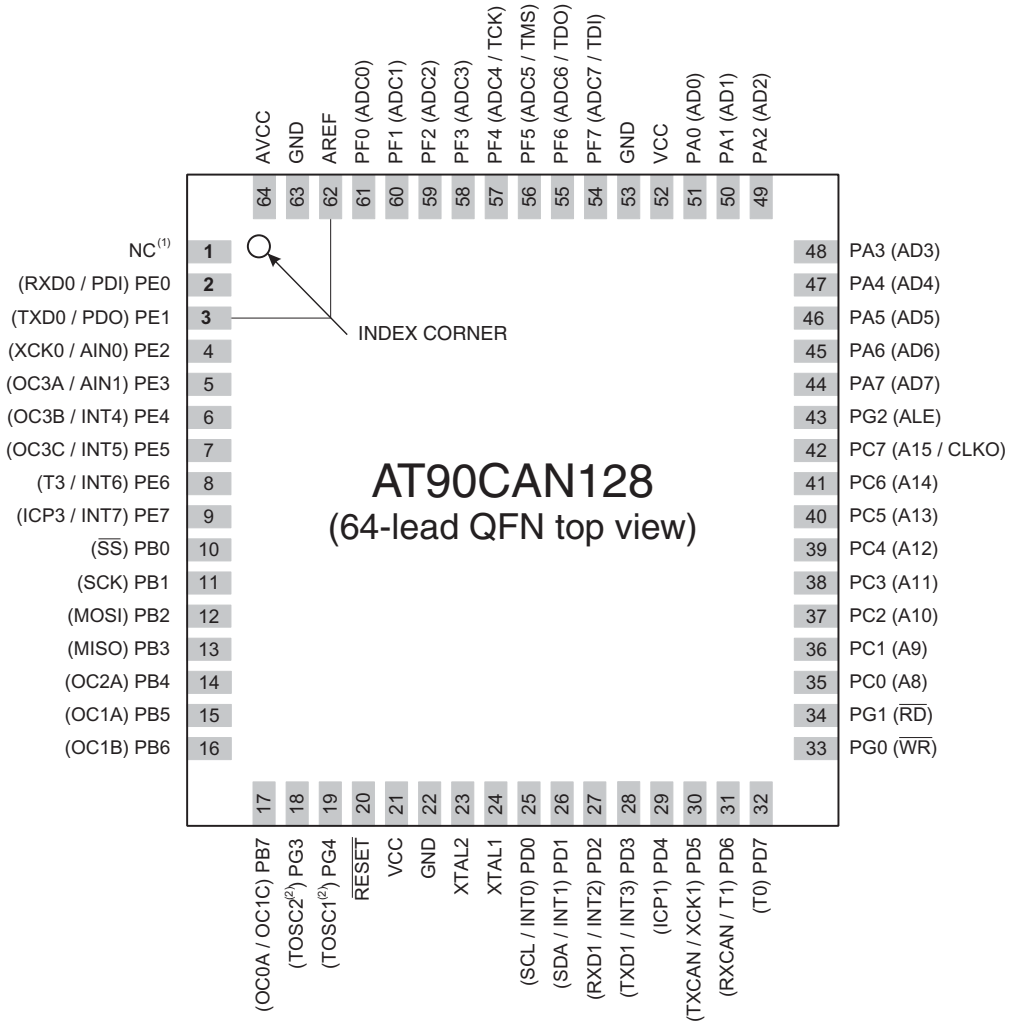
**Figure 2.** Pinout AT90CAN128- TQFP




(1) NC = Do not connect (May be used in future devices)

(2) Timer2 Oscillator

**Figure 3.** Pinout AT90CAN128- QFN




(1) NC = Do not connect (May be used in future devices)

(2) Timer2 Oscillator

## Pin Descriptions

**VCC** Digital supply voltage.

**GND** Ground.

**Port A (PA7..PA0)** Port A is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port A output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port A pins that are externally pulled low will source current if the pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port A also serves the functions of various special features of the AT90CAN128 as listed on page 69.

**Port B (PB7..PB0)** Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B also serves the functions of various special features of the AT90CAN128 as listed on page 71.

**Port C (PC7..PC0)** Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port C also serves the functions of special features of the AT90CAN128 as listed on page 73.

**Port D (PD7..PD0)** Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port D also serves the functions of various special features of the AT90CAN128 as listed on page 75.

**Port E (PE7..PE0)** Port E is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port E output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port E pins that are externally pulled low will source current if the pull-up resistors are activated. The Port E pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port E also serves the functions of various special features of the AT90CAN128 as listed on page 78.

**Port F (PF7..PF0)** Port F serves as the analog inputs to the A/D Converter.

Port F also serves as an 8-bit bi-directional I/O port, if the A/D Converter is not used. Port pins can provide internal pull-up resistors (selected for each bit). The Port F output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port F pins that are externally pulled low will source current if the pull-up

resistors are activated. The Port F pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port F also serves the functions of the JTAG interface. If the JTAG interface is enabled, the pull-up resistors on pins PF7(TDI), PF5(TMS), and PF4(TCK) will be activated even if a reset occurs.

**Port G (PG4..PG0)** Port G is a 5-bit I/O port with internal pull-up resistors (selected for each bit). The Port G output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port G pins that are externally pulled low will source current if the pull-up resistors are activated. The Port G pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port G also serves the functions of various special features of the AT90CAN128 as listed on page 83.

**$\overline{\text{RESET}}$** Reset input. A low level on this pin for longer than the minimum pulse length will generate a reset. The minimum pulse length is given in caracteristics. Shorter pulses are not guaranteed to generate a reset. The I/O ports of the AVR are immediately reset to their initial state even if the clock is not running. The clock is needed to reset the rest of the AT90CAN128.

**XTAL1** Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

**XTAL2** Output from the inverting Oscillator amplifier.

**AVCC** AVCC is the supply voltage pin for the A/D Converter on Port F. It should be externally connected to $V_{CC}$, even if the ADC is not used. If the ADC is used, it should be connected to $V_{CC}$ through a low-pass filter.

**AREF** This is the analog reference pin for the A/D Converter.

## About Code Examples

This documentation contains simple code examples that briefly show how to use various parts of the device. These code examples assume that the part specific header file is included before compilation. Be aware that not all C compiler vendors include bit definitions in the header files and interrupt handling in C is compiler dependent. Please confirm with the C compiler documentation for more details.

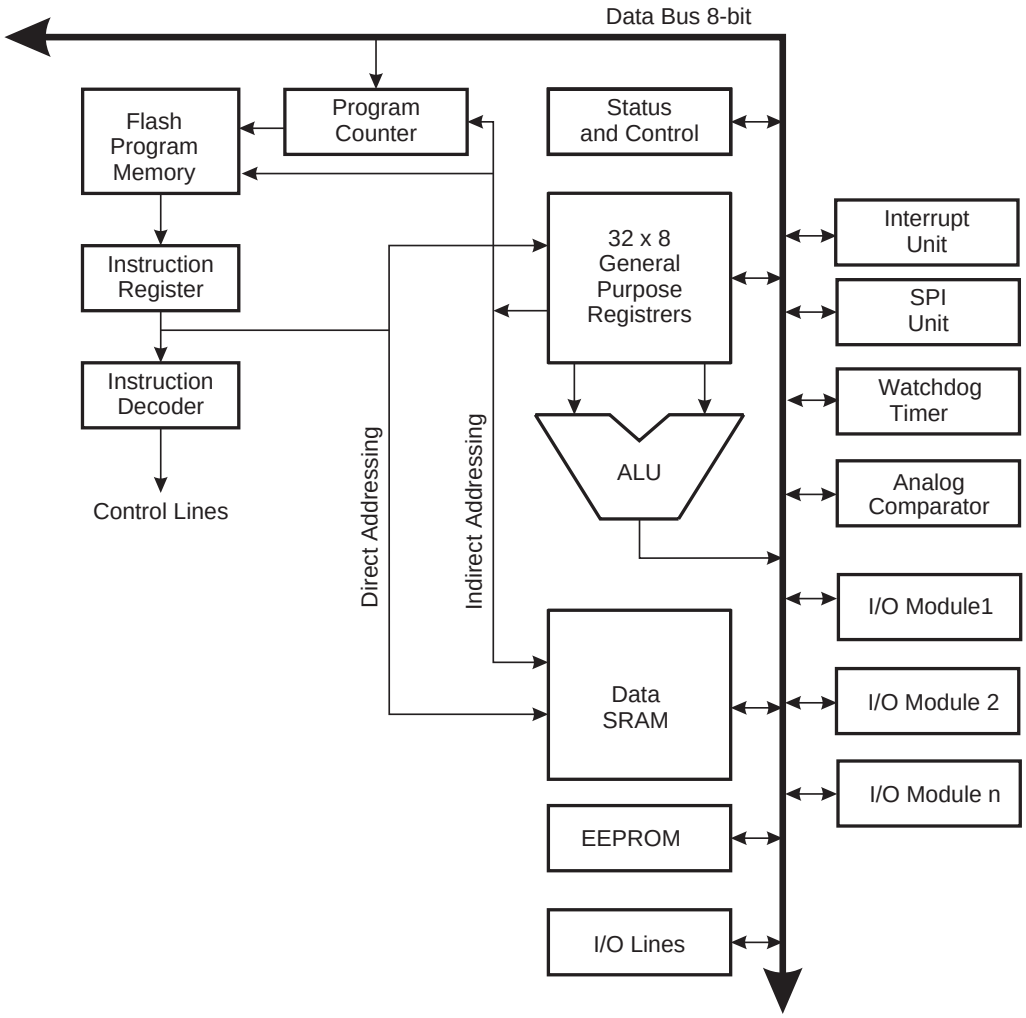
# AVR CPU Core

## Introduction

This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

## Architectural Overview

**Figure 4.** Block Diagram of the AVR Architecture




In order to maximize performance and parallelism, the AVR uses a Harvard architecture – with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-System Reprogrammable Flash memory.

The fast-access Register File contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the Register File,

the operation is executed, and the result is stored back in the Register File – in one clock cycle.

Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing – enabling efficient address calculations. One of the these address pointers can also be used as an address pointer for look up tables in Flash program memory. These added function registers are the 16-bit X-, Y-, and Z-register, described later in this section.

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status Register is updated to reflect information about the result of the operation.

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction.

Program Flash memory space is divided in two sections, the Boot Program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM (Store Program Memory) instruction that writes into the Application Flash memory section must reside in the Boot Program section.

During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently the Stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the Reset routine (before subroutines or interrupts are executed). The Stack Pointer (SP) is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture.

The memory spaces in the AVR architecture are all linear and regular memory maps.

A flexible interrupt module has its control registers in the I/O space with an additional Global Interrupt Enable bit in the Status Register. All interrupts have a separate Interrupt Vector in the Interrupt Vector table. The interrupts have priority in accordance with their Interrupt Vector position. The lower the Interrupt Vector address, the higher is the priority.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, SPI, and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register File, 0x20 - 0x5F. In addition, the AT90CAN128 has Extended I/O space from 0x60 - 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

## ALU – Arithmetic Logic Unit

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See the “Instruction Set” section for a detailed description.

## Status Register

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU operations, as specified in the Instruction Set Reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

The AVR Status Register – SREG – is defined as:

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | I | T | H | S | V | N | Z | C | SREG |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – I: Global Interrupt Enable**

The Global Interrupt Enable bit must be set to enabled the interrupts. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

- **Bit 6 – T: Bit Copy Storage**

The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

- **Bit 5 – H: Half Carry Flag**

The Half Carry Flag H indicates a Half Carry in some arithmetic operations. Half Carry Is useful in BCD arithmetic. See the “Instruction Set Description” for detailed information.

- **Bit 4 – S: Sign Bit, S = N $\oplus$ V**

The S-bit is always an exclusive or between the negative flag N and the Two’s Complement Overflow Flag V. See the “Instruction Set Description” for detailed information.

- **Bit 3 – V: Two’s Complement Overflow Flag**

The Two’s Complement Overflow Flag V supports two’s complement arithmetics. See the “Instruction Set Description” for detailed information.

- **Bit 2 – N: Negative Flag**

The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

- **Bit 1 – Z: Zero Flag**

The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

- **Bit 0 – C: Carry Flag**

The Carry Flag C indicates a carry in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

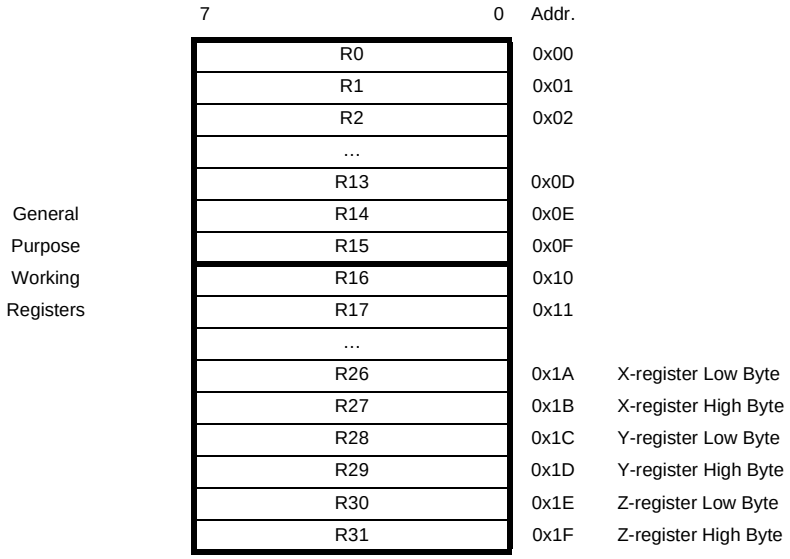
## General Purpose Register File

The Register File is optimized for the AVR Enhanced RISC instruction set. In order to achieve the required performance and flexibility, the following input/output schemes are supported by the Register File:

- One 8-bit output operand and one 8-bit result input
- Two 8-bit output operands and one 8-bit result input
- Two 8-bit output operands and one 16-bit result input
- One 16-bit output operand and one 16-bit result input

Figure 5 shows the structure of the 32 general purpose working registers in the CPU.

**Figure 5.** AVR CPU General Purpose Working Registers



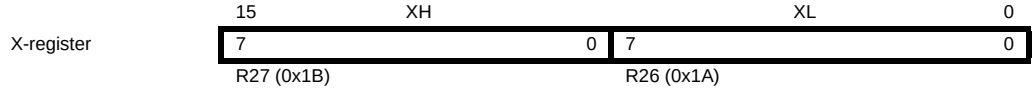

Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions.

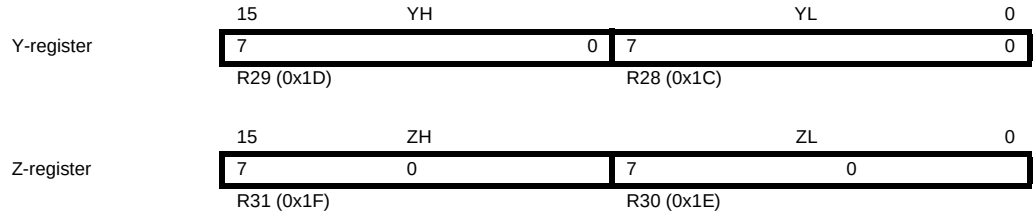
As shown in Figure 5, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user Data Space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y- and Z-pointer registers can be set to index any register in the file.

### The X-register, Y-register, and Z-register

The registers R26..R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the data space. The three indirect address registers X, Y, and Z are defined as described in Figure 6.

**Figure 6.** The X-, Y-, and Z-registers






In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the instruction set reference for details).

### RAM Page Z Select Register – RAMPZ

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | – | – | – | **RAMPZ0** | **RAMPZ** |
| Read/Write | R | R | R | R | R | R | R | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bits 7..2 – Res: Reserved Bits**

These are reserved bits and will always read as zero. When writing to this address location, write these bits to zero for compatibility with future devices.

- **Bit 1 – RAMPZ0: Extended RAM Page Z-pointer**

The RAMPZ Register is normally used to select which 64K RAM Page is accessed by the Z-pointer. As the AT90CAN128 does not support more than 64K of SRAM memory, this register is used only to select which page in the program memory is accessed when the ELPM/SPM instruction is used. The different settings of the RAMPZ0 bit have the following effects:

RAMPZ0 = 0: Program memory address 0x0000 - 0x7FFF (lower 64K bytes) is accessed by ELPM/SPM

RAMPZ0 = 1: Program memory address 0x8000 - 0xFFFF (higher 64K bytes) is accessed by ELPM/SPM

Note that LPM is not affected by the RAMPZ setting.

## Stack Pointer

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack Pointer Register always points to the top of the Stack. Note that the Stack is implemented as growing from higher memory locations to lower memory locations. This implies that a Stack PUSH command decreases the Stack Pointer.

The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. This Stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The Stack Pointer must be set to point above 0x60. The Stack Pointer is decremented by one when data is pushed onto the Stack with the PUSH instruction, and it is decremented by two when the return address is pushed onto the Stack with subroutine call or interrupt. The Stack Pointer is incremented by one when data is popped from the Stack with the POP instruction, and it is incremented by two when data is popped from the Stack with return from subroutine RET or return from interrupt RETI.

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some

implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | SP15 | SP14 | SP13 | SP12 | SP11 | SP10 | SP9 | SP8 | SPH |
| | SP7 | SP6 | SP5 | SP4 | SP3 | SP2 | SP1 | SP0 | SPL |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

## Instruction Execution Timing

This section describes the general access timing concepts for instruction execution. The AVR CPU is driven by the CPU clock $clk_{CPU}$, directly generated from the selected clock source for the chip. No internal clock division is used.

Figure 7 shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access Register File concept. This is the basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

**Figure 7.** The Parallel Instruction Fetches and Instruction Executions

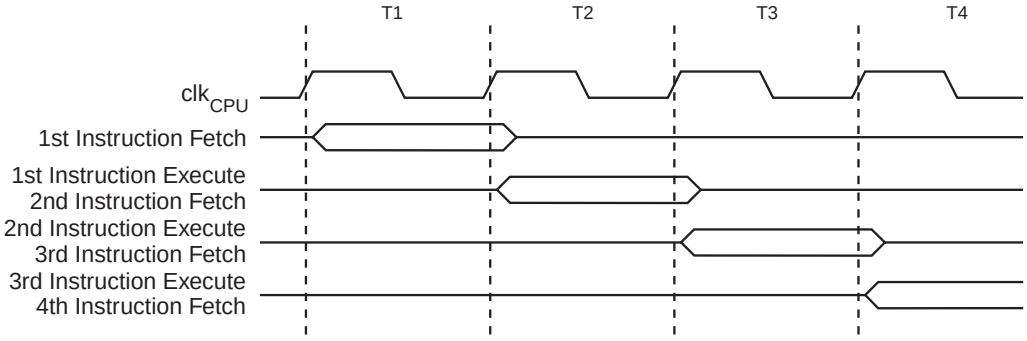
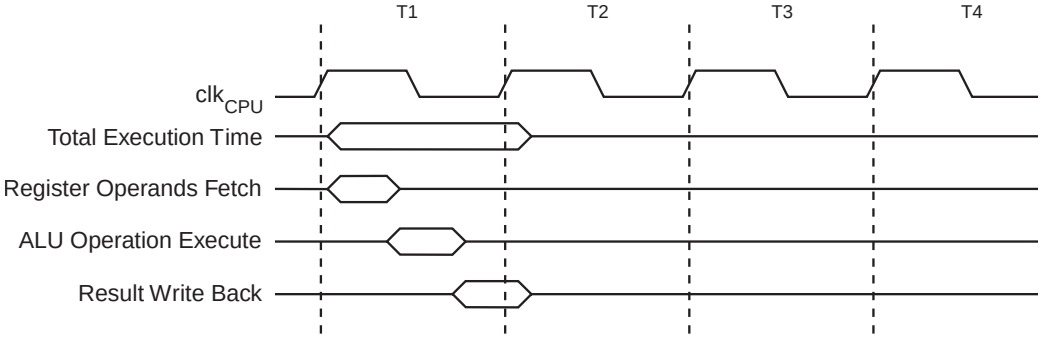



Figure 8 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

**Figure 8.** Single Cycle ALU Operation




## Reset and Interrupt Handling

The AVR provides several different interrupt sources. These interrupts and the separate Reset Vector each have a separate program vector in the program memory space. All interrupts are assigned individual enable bits which must be written logic one together

with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt. Depending on the Program Counter value, interrupts may be automatically disabled when Boot Lock bits BLB02 or BLB12 are programmed. This feature improves software security. See the section “Memory Programming” on page 325 for details.

The lowest addresses in the program memory space are by default defined as the Reset and Interrupt Vectors. The complete list of vectors is shown in “Interrupts” on page 56. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request 0. The Interrupt Vectors can be moved to the start of the Boot Flash section by setting the IVSEL bit in the MCU Control Register (MCUCR). Refer to “Interrupts” on page 56 for more information. The Reset Vector can also be moved to the start of the Boot Flash section by programming the BOOTRST Fuse, see “Boot Loader Support – Read-While-Write Self-Programming” on page 311.

**Interrupt Behavior**

When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed.

There are basically two types of interrupts. The first type is triggered by an event that sets the interrupt flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding interrupt flag. Interrupt flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the interrupt flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the Global Interrupt Enable bit is cleared, the corresponding interrupt flag(s) will be set and remembered until the Global Interrupt Enable bit is set, and will then be executed by order of priority.

The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have interrupt flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.

When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.

Note that the Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction. The following example shows how this can be used to avoid interrupts during the timed EEPROM write sequence.

Assembly Code Example

```
in  r16, SREG      ; store SREG value
cli                ; disable interrupts during timed sequence
sbi EECR, EEMWE    ; start EEPROM write
sbi EECR, EEWE
out SREG, r16      ; restore SREG value (I-bit)
```

C Code Example

```
char cSREG;
cSREG = SREG;        /* store SREG value */
/* disable interrupts during timed sequence */
_CLI();
EECR |= (1<<EEMWE); /* start EEPROM write */
EECR |= (1<<EEWE);
SREG = cSREG;        /* restore SREG value (I-bit) */
```

When using the SEI instruction to enable interrupts, the instruction following SEI will be executed before any pending interrupts, as shown in this example.

Assembly Code Example

```
sei   ; set Global Interrupt Enable
sleep ; enter sleep, waiting for interrupt
; note: will enter sleep before any pending
; interrupt(s)
```

C Code Example

```
_SEI(); /* set Global Interrupt Enable */
_SLEEP(); /* enter sleep, waiting for interrupt */
/* note: will enter sleep before any pending interrupt(s) */
```

**Interrupt Response Time**

The interrupt execution response for all the enabled AVR interrupts is four clock cycles minimum. After four clock cycles the program vector address for the actual interrupt handling routine is executed. During this four clock cycle period, the Program Counter is pushed onto the Stack. The vector is normally a jump to the interrupt routine, and this jump takes three clock cycles. If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served. If an interrupt occurs when the MCU is in sleep mode, the interrupt execution response time is increased by four clock cycles. This increase comes in addition to the start-up time from the selected sleep mode.

A return from an interrupt handling routine takes four clock cycles. During these four clock cycles, the Program Counter (two bytes) is popped back from the Stack, the Stack Pointer is incremented by two, and the I-bit in SREG is set.

## Memories

This section describes the different memories in the AT90CAN128. The AVR architecture has two main memory spaces, the Data Memory and the Program Memory space. In addition, the AT90CAN128 features an EEPROM Memory for data storage. All three memory spaces are linear and regular.

### In-System Reprogrammable Flash Program Memory

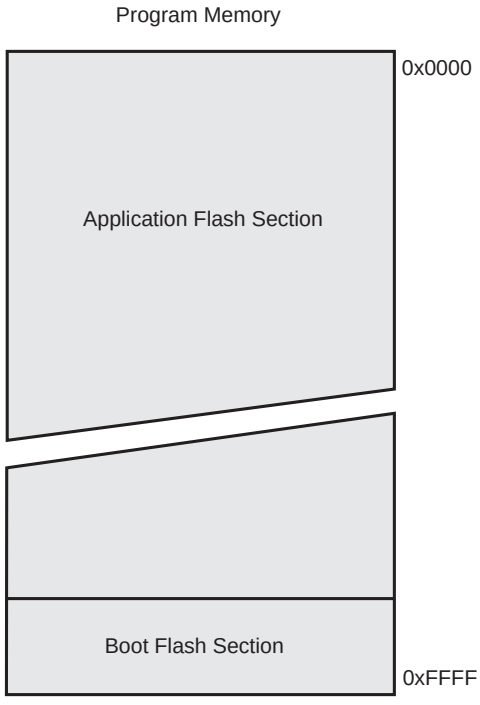
The AT90CAN128 contains 128K bytes On-chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 64K x 16. For software security, the Flash Program memory space is divided into two sections, Boot Program section and Application Program section.

The Flash memory has an endurance of at least 10,000 write/erase cycles. The AT90CAN128 Program Counter (PC) is 16 bits wide, thus addressing the 64K program memory locations. The operation of Boot Program section and associated Boot Lock bits for software protection are described in detail in “Boot Loader Support – Read-While-Write Self-Programming” on page 311. “Memory Programming” on page 325 contains a detailed description on Flash data serial downloading using the SPI pins or the JTAG interface.

Constant tables can be allocated within the entire program memory address space (see the LPM – Load Program Memory and ELPM – Extended Load Program Memory instruction description).

Timing diagrams for instruction fetch and execution are presented in “Instruction Execution Timing” on page 13.

**Figure 9.** Program Memory Map



## SRAM Data Memory

Figure 10 shows how the AT90CAN128 SRAM Memory is organized.

The AT90CAN128 is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in the Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

The lower 1,280 data memory locations address both the Register File, the I/O memory, Extended I/O memory, and the internal data SRAM. The first 32 locations address the Register File, the next 64 location the standard I/O memory, then 160 locations of Extended I/O memory, and the next 4096 locations address the internal data SRAM.

An optional external data SRAM can be used with the AT90CAN128. This SRAM will occupy an area in the remaining address locations in the 64K address space. This area starts at the address following the internal SRAM. The Register file, I/O, Extended I/O and Internal SRAM occupies the lowest 4352 bytes, so when using 64 KB (65,536 bytes) of External Memory, 61,184 bytes of External Memory are available. See “External Memory Interface” on page 24 for details on how to take advantage of the external memory map.

### SRAM Data Access

When the addresses accessing the SRAM memory space exceeds the internal data memory locations, the external data SRAM is accessed using the same instructions as for the internal data memory access. When the internal data memories are accessed, the read and write strobe pins (PG0 and PG1) are inactive during the whole access cycle. External SRAM operation is enabled by setting the SRE bit in the XMCRA Register.

Accessing external SRAM takes one additional clock cycle per byte compared to access of the internal SRAM. This means that the commands LD, ST, LDS, STS, LDD, STD, PUSH, and POP take one additional clock cycle. If the Stack is placed in external SRAM, interrupts, subroutine calls and returns take three clock cycles extra because the two-byte program counter is pushed and popped, and external memory access does not take advantage of the internal pipe-line memory access. When external SRAM interface is used with wait-state, one-byte external access takes two, three, or four additional clock cycles for one, two, and three wait-states respectively. Interrupts, subroutine calls and returns will need five, seven, or nine clock cycles more than specified in the instruction set manual for one, two, and three wait-states.

The five different addressing modes for the data memory cover: Direct, Indirect with Displacement, Indirect, Indirect with Pre-decrement, and Indirect with Post-increment. In the Register File, registers R26 to R31 feature the indirect addressing pointer registers.

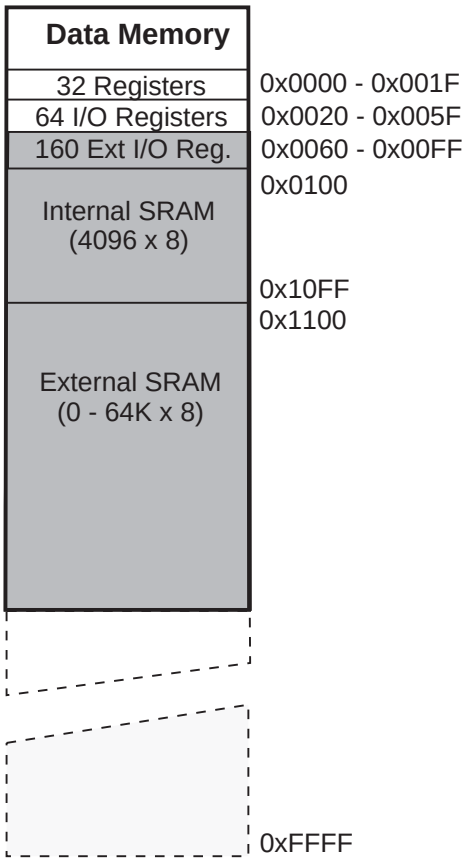
The direct addressing reaches the entire data space.

The Indirect with Displacement mode reaches 63 address locations from the base address given by the Y- or Z-register.

When using register indirect addressing modes with automatic pre-decrement and post-increment, the address registers X, Y, and Z are decremented or incremented.

The 32 general purpose working registers, 64 I/O Registers, 160 Extended I/O Registers, and the 1,024 bytes of internal data SRAM in the AT90CAN128 are all accessible through all these addressing modes. The Register File is described in “General Purpose Register File” on page 11.

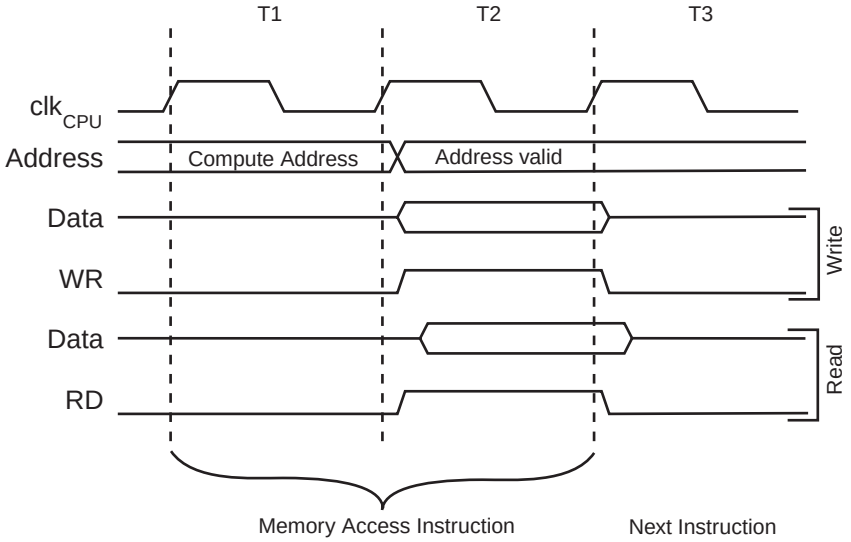
**Figure 10.** Data Memory Map




**SRAM Data Access Times**

This section describes the general access timing concepts for internal memory access. The internal data SRAM access is performed in two $clk_{CPU}$ cycles as described in Figure 11.

**Figure 11.** On-chip Data SRAM Access Cycles



## EEPROM Data Memory

The AT90CAN128 contains 4-Kbytes of data EEPROM memory. It is organized as a separate data space, in which single bytes can be read and written. The EEPROM has an endurance of at least 100,000 write/erase cycles. The access between the EEPROM and the CPU is described in the following, specifying the EEPROM Address Registers, the EEPROM Data Register, and the EEPROM Control Register.

For a detailed description of SPI, JTAG and Parallel data downloading to the EEPROM, see “SPI Serial Programming Overview” on page 337, “JTAG Programming Overview” on page 342, and “Parallel Programming Overview” on page 329 respectively.

### EEPROM Read/Write Access

The EEPROM Access Registers are accessible in the I/O space.

The write access time for the EEPROM is given in Table 1. A self-timing function, however, lets the user software detect when the next byte can be written. If the user code contains instructions that write the EEPROM, some precautions must be taken. In heavily filtered power supplies, $V_{CC}$ is likely to rise or fall slowly on power-up/down. This causes the device for some period of time to run at a voltage lower than specified as minimum for the clock frequency used. See “Preventing EEPROM Corruption” on page 23 for details on how to avoid problems in these situations.

In order to prevent unintentional EEPROM writes, a specific write procedure must be followed. Refer to the description of the EEPROM Control Register for details on this.

When the EEPROM is read, the CPU is halted for four clock cycles before the next instruction is executed. When the EEPROM is written, the CPU is halted for two clock cycles before the next instruction is executed.

### The EEPROM Address Registers – EEARH and EEARL

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | **EEAR11** | **EEAR10** | **EEAR9** | **EEAR8** | **EEARH** |
| | **EEAR7** | **EEAR6** | **EEAR5** | **EEAR4** | **EEAR3** | **EEAR2** | **EEAR1** | **EEAR0** | EEARL |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Read/Write | R | R | R | R | R/W | R/W | R/W | R/W | |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | X | X | X | X | |
| | X | X | X | X | X | X | X | X | |

- **Bits 15..12 – Reserved Bits**

These bits are reserved bits in the AT90CAN128 and will always read as zero.

- **Bits 11..0 – EEAR11..0: EEPROM Address**

The EEPROM Address Registers – EEARH and EEARL specify the EEPROM address in the 4-Kbytes EEPROM space. The EEPROM data bytes are addressed linearly between 0 and 4,095. The initial value of EEAR is undefined. A proper value must be written before the EEPROM may be accessed.

### The EEPROM Data Register – EEDR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **EEDR7** | **EEDR6** | **EEDR5** | **EEDR4** | **EEDR3** | **EEDR2** | **EEDR1** | **EEDR0** | **EEDR** |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bits 7..0 – EEDR7.0: EEPROM Data**

For the EEPROM write operation, the EEDR Register contains the data to be written to the EEPROM in the address given by the EEAR Register. For the EEPROM read operation, the EEDR contains the data read out from the EEPROM at the address given by EEAR.

**The EEPROM Control Register – EECR**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | EERIE | EEMWE | EEWE | EERE | EECR |
| Read/Write | R | R | R | R | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | X | 0 | |

- **Bits 7..4 – Reserved Bits**

These bits are reserved bits in the AT90CAN128 and will always read as zero.

- **Bit 3 – EERIE: EEPROM Ready Interrupt Enable**

Writing EERIE to one enables the EEPROM Ready Interrupt if the I bit in SREG is set. Writing EERIE to zero disables the interrupt. The EEPROM Ready interrupt generates a constant interrupt when EEWE is cleared.

- **Bit 2 – EEMWE: EEPROM Master Write Enable**

The EEMWE bit determines whether setting EEWE to one causes the EEPROM to be written. When EEMWE is set, setting EEWE within four clock cycles will write data to the EEPROM at the selected address If EEMWE is zero, setting EEWE will have no effect. When EEMWE has been written to one by software, hardware clears the bit to zero after four clock cycles. See the description of the EEWE bit for an EEPROM write procedure.

- **Bit 1 – EEWE: EEPROM Write Enable**

The EEPROM Write Enable Signal EEWE is the write strobe to the EEPROM. When address and data are correctly set up, the EEWE bit must be written to one to write the value into the EEPROM. The EEMWE bit must be written to one before a logical one is written to EEWE, otherwise no EEPROM write takes place. The following procedure should be followed when writing the EEPROM (the order of steps 3 and 4 is not essential):

1. Wait until EEWE becomes zero.
2. Wait until SPMEN (Store Program Memory Enable) in SPMCSR (Store Program Memory Control and Status Register) becomes zero.
3. Write new EEPROM address to EEAR (optional).
4. Write new EEPROM data to EEDR (optional).
5. Write a logical one to the EEMWE bit while writing a zero to EEWE in EECR.
6. Within four clock cycles after setting EEMWE, write a logical one to EEWE.

The EEPROM can not be programmed during a CPU write to the Flash memory. The software must check that the Flash programming is completed before initiating a new EEPROM write. Step 2 is only relevant if the software contains a Boot Loader allowing the CPU to program the Flash. If the Flash is never being updated by the CPU, step 2 can be omitted. See “Boot Loader Support – Read-While-Write Self-Programming” on page 311 for details about Boot programming.

**Caution:** An interrupt between step 5 and step 6 will make the write cycle fail, since the EEPROM Master Write Enable will time-out. If an interrupt routine accessing the

EEPROM is interrupting another EEPROM access, the EEAR or EEDR Register will be modified, causing the interrupted EEPROM access to fail. It is recommended to have the Global Interrupt Flag cleared during all the steps to avoid these problems.

When the write access time has elapsed, the EEWE bit is cleared by hardware. The user software can poll this bit and wait for a zero before writing the next byte. When EEWE has been set, the CPU is halted for two cycles before the next instruction is executed.

- **Bit 0 – EERE: EEPROM Read Enable**

The EEPROM Read Enable Signal EERE is the read strobe to the EEPROM. When the correct address is set up in the EEAR Register, the EERE bit must be written to a logic one to trigger the EEPROM read. The EEPROM read access takes one instruction, and the requested data is available immediately. When the EEPROM is read, the CPU is halted for four cycles before the next instruction is executed.

The user should poll the EEWE bit before starting the read operation. If a write operation is in progress, it is neither possible to read the EEPROM, nor to change the EEAR Register.

The calibrated Oscillator is used to time the EEPROM accesses. Table 1 lists the typical programming time for EEPROM access from the CPU.

**Table 1.** EEPROM Programming Time.

| Symbol | Number of Calibrated RC Oscillator Cycles | Typ Programming Time |
|---|---|---|
| EEPROM write (from CPU) | 67 584 | 8.5 ms |

Note: 1. Uses 1 MHz clock, independent of CKSEL Fuse settings.

The following code examples show one assembly and one C function for writing to the EEPROM. The examples assume that interrupts are controlled (e.g. by disabling interrupts globally) so that no interrupts will occur during execution of these functions. The examples also assume that no Flash Boot Loader is present in the software. If such code is present, the EEPROM write function must also wait for any ongoing SPM command to finish.

Assembly Code Example

```
EEPROM_write:
  ; Wait for completion of previous write
  sbic EECR,EEWE
  rjmp EEPROM_write
  ; Set up address (r18:r17) in address register
  out  EEARH, r18
  out  EEARL, r17
  ; Write data (r16) to data register
  out  EEDR,r16
  ; Write logical one to EEMWE
  sbi  EECR,EEMWE
  ; Start eeprom write by setting EEWE
  sbi  EECR,EEWE
  ret
```

C Code Example

```
void EEPROM_write (unsigned int uiAddress, unsigned char ucData)
{
  /* Wait for completion of previous write */
  while(EECR & (1<<EEWE))
    ;
  /* Set up address and data registers */
  EEAR = uiAddress;
  EEDR = ucData;
  /* Write logical one to EEMWE */
  EECR |= (1<<EEMWE);
  /* Start eeprom write by setting EEWE */
  EECR |= (1<<EEWE);
}
```

The next code examples show assembly and C functions for reading the EEPROM. The examples assume that interrupts are controlled so that no interrupts will occur during execution of these functions.

Assembly Code Example

```
EEPROM_read:
  ; Wait for completion of previous write
  sbic EECR,EEWE
  rjmp EEPROM_read
  ; Set up address (r18:r17) in address register
  out  EEARH, r18
  out  EEARL, r17
  ; Start eeprom read by writing EERE
  sbi  EECR,EERE
  ; Read data from data register
  in   r16,EEDR
  ret
```

C Code Example

```
unsigned char EEPROM_read(unsigned int uiAddress)
{
  /* Wait for completion of previous write */
  while(EECR & (1<<EEWE))
    ;
  /* Set up address register */
  EEAR = uiAddress;
  /* Start eeprom read by writing EERE */
  EECR |= (1<<EERE);
  /* Return data from data register */
  return EEDR;
}
```

**Preventing EEPROM Corruption**

During periods of low $V_{CC,}$ the EEPROM data can be corrupted because the supply voltage is too low for the CPU and the EEPROM to operate properly. These issues are the same as for board level systems using EEPROM, and the same design solutions should be applied.

An EEPROM data corruption can be caused by two situations when the voltage is too low. First, a regular write sequence to the EEPROM requires a minimum voltage to operate correctly. Secondly, the CPU itself can execute instructions incorrectly, if the supply voltage is too low.

EEPROM data corruption can easily be avoided by following this design recommendation:

Keep the AVR RESET active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal Brown-out Detector (BOD). If the detection level of the internal BOD does not match the needed detection level, an external low $V_{CC}$ reset Protection circuit can be used. If a reset occurs while a write operation is in progress, the write operation will be completed provided that the power supply voltage is sufficient.

## I/O Memory

The I/O space definition of the AT90CAN128 is shown in “Register Summary” on page 394.

All AT90CAN128 I/Os and peripherals are placed in the I/O space. All I/O locations may be accessed by the LD/LDS/LDD and ST/STS/STD instructions, transferring data between the 32 general purpose working registers and the I/O space. I/O registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions. Refer to the instruction set section for more details. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The AT90CAN128 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.

Some of the status flags are cleared by writing a logical one to them. Note that, unlike most other AVR’s, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such status flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.

The I/O and peripherals control registers are explained in later sections.

## External Memory Interface

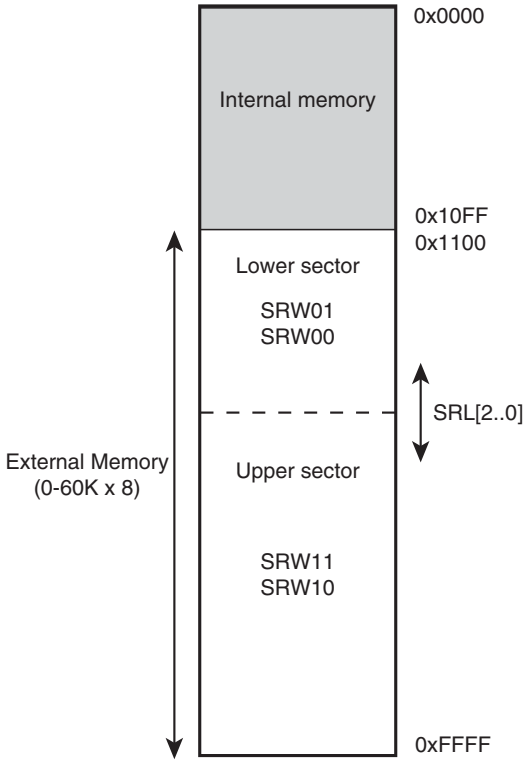
With all the features the External Memory Interface provides, it is well suited to operate as an interface to memory devices such as External SRAM and Flash, and peripherals such as LCD-display, A/D, and D/A. The main features are:

- Four different wait-state settings (including no wait-state).
- Independent wait-state setting for different extErnal Memory sectors (configurable sector size).
- The number of bits dedicated to address high byte is selectable.
- Bus keepers on data lines to minimize current consumption (optional).

### Overview

When the eXternal MEMory (XMEM) is enabled, address space outside the internal SRAM becomes available using the dedicated External Memory pins (see Figure 2 on page 4, Table 29 on page 69, Table 35 on page 73, and Table 47 on page 83). The memory configuration is shown in Figure 12.

**Figure 12.** External Memory with Sector Select




**Using the External Memory Interface**

The interface consists of:

- AD7:0: Multiplexed low-order address bus and data bus.
- A15:8: High-order address bus (configurable number of bits).
- ALE: Address latch enable.
- $\overline{RD}$: Read strobe.
- $\overline{WR}$: Write strobe.

The control bits for the External Memory Interface are located in two registers, the External Memory Control Register A – XMCRA, and the External Memory Control Register B – XMCRB.

When the XMEM interface is enabled, the XMEM interface will override the setting in the data direction registers that corresponds to the ports dedicated to the XMEM interface. For details about the port override, see the alternate functions in section “I/O-Ports” on page 61. The XMEM interface will auto-detect whether an access is internal or external. If the access is external, the XMEM interface will output address, data, and the control signals on the ports according to Figure 14 (this figure shows the wave forms without wait-states). When ALE goes from high-to-low, there is a valid address on AD7:0. ALE is low during a data transfer. When the XMEM interface is enabled, also an internal access will cause activity on address, data and ALE ports, but the $\overline{RD}$ and $\overline{WR}$ strobes will not toggle during internal access. When the External Memory Interface is disabled, the normal pin and data direction settings are used. Note that when the XMEM interface is disabled, the address space above the internal SRAM boundary is not mapped into the internal SRAM. Figure 13 illustrates how to connect an external SRAM to the AVR using an octal latch (typically “74 x 573” or equivalent) which is transparent when G is high.

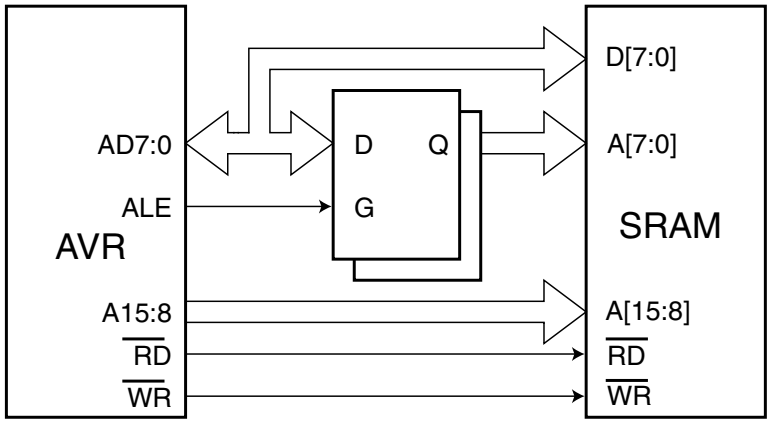
### Address Latch Requirements

Due to the high-speed operation of the XRAM interface, the address latch must be selected with care for system frequencies above 8 MHz @ 4V and 4 MHz @ 2.7V. When operating at conditions above these frequencies, the typical old style 74HC series latch becomes inadequate. The External Memory Interface is designed in compliance to the 74AHC series latch. However, most latches can be used as long they comply with the main timing parameters. The main parameters for the address latch are:

- D to Q propagation delay ($t_{PD}$).
- Data setup time before G low ($t_{SU}$).
- Data (address) hold time after G low ($_{TH}$).

The External Memory Interface is designed to guaranty minimum address hold time after G is asserted low of $t_h$ = 5 ns. Refer to $t_{LAXX_LD}$/$t_{LLAXX_ST}$ in “Memory Programming” Tables 142 through Tables 149. The D-to-Q propagation delay ($t_{PD}$) must be taken into consideration when calculating the access time requirement of the external component. The data setup time before G low ($t_{SU}$) must not exceed address valid to ALE low ($t_{AVLLC}$) minus PCB wiring delay (dependent on the capacitive load).

**Figure 13.** External SRAM Connected to the AVR




### Pull-up and Bus-keeper

The pull-ups on the AD7:0 ports may be activated if the corresponding Port register is written to one. To reduce power consumption in sleep mode, it is recommended to disable the pull-ups by writing the Port register to zero before entering sleep.

The XMEM interface also provides a bus-keeper on the AD7:0 lines. The bus-keeper can be disabled and enabled in software as described in “External Memory Control Register B – XMCRB” on page 30. When enabled, the bus-keeper will ensure a defined logic level (zero or one) on the AD7:0 bus when these lines would otherwise be tri-stated by the XMEM interface.

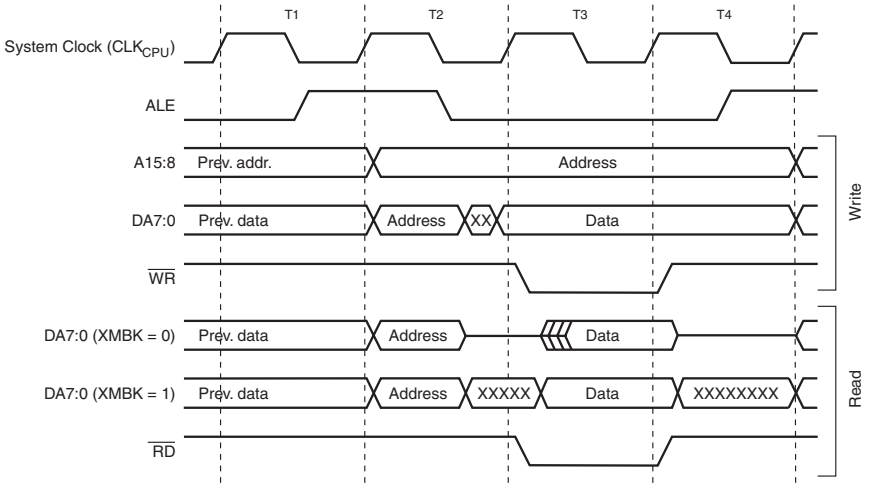
### Timing

External Memory devices have different timing requirements. To meet these requirements, the AT90CAN128 XMEM interface provides four different wait-states as shown in Table 3. It is important to consider the timing specification of the External Memory device before selecting the wait-state. The most important parameters are the access time for the external memory compared to the set-up requirement of the AT90CAN128. The access time for the External Memory is defined to be the time from receiving the chip select/address until the data of this address actually is driven on the bus. The access time cannot exceed the time from the ALE pulse must be asserted low until data is stable during a read sequence (See $t_{LLRL}$+ $t_{RLRH}$ - $t_{DVRH}$ in Tables 142 through Tables 149). The different wait-states are set up in software. As an additional feature, it is possible to divide the external memory space in two sectors with individual wait-state

settings. This makes it possible to connect two different memory devices with different timing requirements to the same XMEM interface. For XMEM interface timing details, please refer to Tables 142 through Tables 149 and Figure 173 to Figure 176 in the "External Data Memory Characteristics" on page 365.

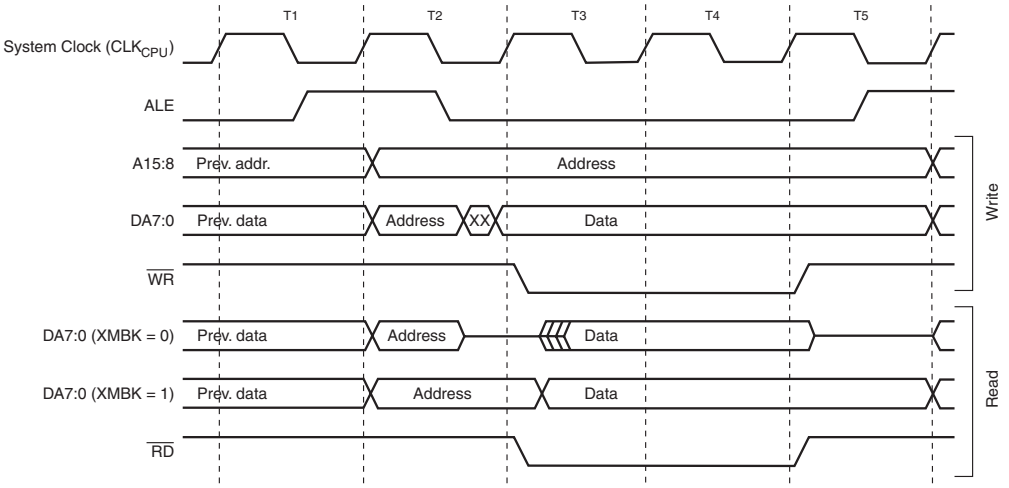
Note that the XMEM interface is asynchronous and that the waveforms in the following figures are related to the internal system clock. The skew between the internal and external clock (XTAL1) is not guarantied (varies between devices temperature, and supply voltage). Consequently, the XMEM interface is not suited for synchronous operation.

**Figure 14.** External Data Memory Cycles no Wait-state (SRWn1=0 and SRWn0=0)(1)



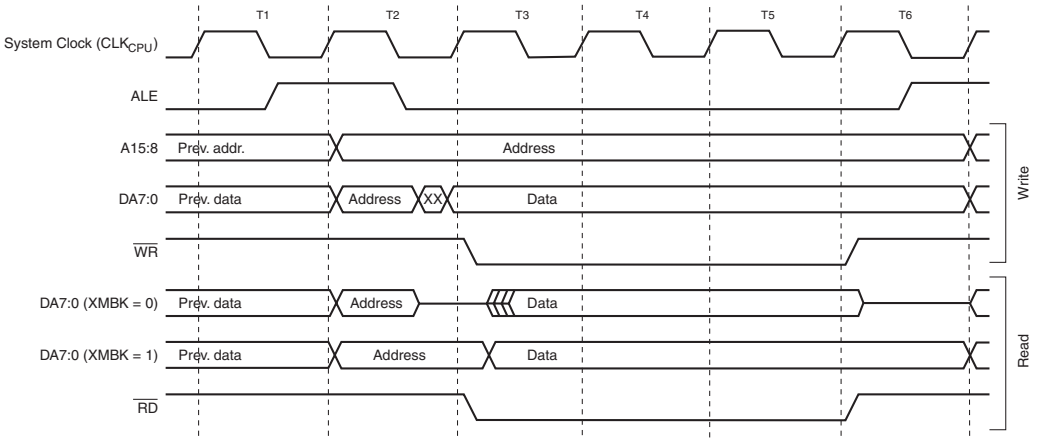

Note: 1. SRWn1 = SRW11 (upper sector) or SRW01 (lower sector), SRWn0 = SRW10 (upper sector) or SRW00 (lower sector). The ALE pulse in period T4 is only present if the next instruction accesses the RAM (internal or external).

**Figure 15.** External Data Memory Cycles with SRWn1 = 0 and SRWn0 = 1(1)



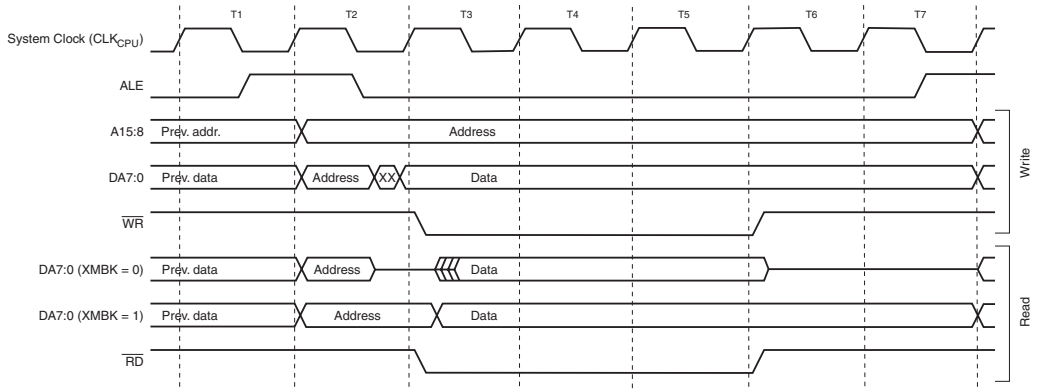

Note: 1. SRWn1 = SRW11 (upper sector) or SRW01 (lower sector), SRWn0 = SRW10 (upper sector) or SRW00 (lower sector).
The ALE pulse in period T5 is only present if the next instruction accesses the RAM (internal or external).

**Figure 16.** External Data Memory Cycles with SRWn1 = 1 and SRWn0 = 0(1)




Note: 1. SRWn1 = SRW11 (upper sector) or SRW01 (lower sector), SRWn0 = SRW10 (upper sector) or SRW00 (lower sector).
The ALE pulse in period T6 is only present if the next instruction accesses the RAM (internal or external).

**Figure 17.** External Data Memory Cycles with SRWn1 = 1 and SRWn0 = 1(1)




Note: 1. SRWn1 = SRW11 (upper sector) or SRW01 (lower sector), SRWn0 = SRW10 (upper sector) or SRW00 (lower sector).
The ALE pulse in period T7 is only present if the next instruction accesses the RAM (internal or external).

## XMEM Register Description

### External Memory Control Register A – XMCRA

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **SRE** | **SRL2** | **SRL1** | **SRL0** | **SRW11** | **SRW10** | **SRW01** | **SRW00** | **XMCRA** |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – SRE: External SRAM/XMEM Enable**

Writing SRE to one enables the External Memory Interface.The pin functions AD7:0, A15:8, ALE, $\overline{WR}$, and $\overline{RD}$ are activated as the alternate pin functions. The SRE bit over-

rides any pin direction settings in the respective data direction registers. Writing SRE to zero, disables the External Memory Interface and the normal pin and data direction settings are used. Note that when the XMEM interface is disabled, the address space above the internal SRAM boundary is not mapped into the internal SRAM.

- **Bit 6..4 – SRL2, SRL1, SRL0: Wait-state Sector Limit**

It is possible to configure different wait-states for different External Memory addresses. The external memory address space can be divided in two sectors that have separate wait-state bits. The SRL2, SRL1, and SRL0 bits select the split of the sectors, see Table 2 and Figure 12. By default, the SRL2, SRL1, and SRL0 bits are set to zero and the entire external memory address space is treated as one sector. When the entire SRAM address space is configured as one sector, the wait-states are configured by the SRW11 and SRW10 bits.

**Table 2.** Sector limits with different settings of SRL2..0

| SRL2 | SRL1 | SRL0 | Sector Limits |
|---|---|---|---|
| 0 | 0 | 0 | Lower sector = N/A<br>Upper sector = 0x1100 - 0xFFFF |
| 0 | 0 | 1 | Lower sector = 0x1100 - 0x1FFF<br>Upper sector = 0x2000 - 0xFFFF |
| 0 | 1 | 0 | Lower sector = 0x1100 - 0x3FFF<br>Upper sector = 0x4000 - 0xFFFF |
| 0 | 1 | 1 | Lower sector = 0x1100 - 0x5FFF<br>Upper sector = 0x6000 - 0xFFFF |
| 1 | 0 | 0 | Lower sector = 0x1100 - 0x7FFF<br>Upper sector = 0x8000 - 0xFFFF |
| 1 | 0 | 1 | Lower sector = 0x1100 - 0x9FFF<br>Upper sector = 0xA000 - 0xFFFF |
| 1 | 1 | 0 | Lower sector = 0x1100 - 0xBFFF<br>Upper sector = 0xC000 - 0xFFFF |
| 1 | 1 | 1 | Lower sector = 0x1100 - 0xDFFF<br>Upper sector = 0xE000 - 0xFFFF |

- **Bit 3..2 – SRW11, SRW10: Wait-state Select Bits for Upper Sector**

The SRW11 and SRW10 bits control the number of wait-states for the upper sector of the external memory address space, see Table 3.

- **Bit 1..0 – SRW01, SRW00: Wait-state Select Bits for Lower Sector**

The SRW01 and SRW00 bits control the number of wait-states for the lower sector of the external memory address space, see Table 3.

**Table 3.** Wait States(1)

| SRWn1 | SRWn0 | Wait States |
|---|---|---|
| 0 | 0 | No wait-states |
| 0 | 1 | Wait one cycle during read/write strobe |
| 1 | 0 | Wait two cycles during read/write strobe |
| 1 | 1 | Wait two cycles during read/write and wait one cycle before driving out new address |

Note: 1. n = 0 or 1 (lower/upper sector).
For further details of the timing and wait-states of the External Memory Interface, see Figures 14 through Figures 17 for how the setting of the SRW bits affects the timing.

### External Memory Control Register B – XMCRB

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | XMBK | – | – | – | – | XMM2 | XMM1 | XMM0 | XMCRB |
| Read/Write | R/W | R | R | R | R | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7– XMBK: External Memory Bus-keeper Enable**

Writing XMBK to one enables the bus keeper on the AD7:0 lines. When the bus keeper is enabled, it will ensure a defined logic level (zero or one) on AD7:0 when they would otherwise be tri-stated. Writing XMBK to zero disables the bus keeper. XMBK is not qualified with SRE, so even if the XMEM interface is disabled, the bus keepers are still activated as long as XMBK is one.

- **Bit 6..4 – Reserved Bits**

These are reserved bits and will always read as zero. When writing to this address location, write these bits to zero for compatibility with future devices.

- **Bit 2..0 – XMM2, XMM1, XMM0: External Memory High Mask**

When the External Memory is enabled, all Port C pins are default used for the high address byte. If the full 60KB address space is not required to access the External Memory, some, or all, Port C pins can be released for normal Port Pin function as described in Table 4. As described in “Using all 64KB Locations of External Memory” on page 31, it is possible to use the XMMn bits to access all 64KB locations of the External Memory.

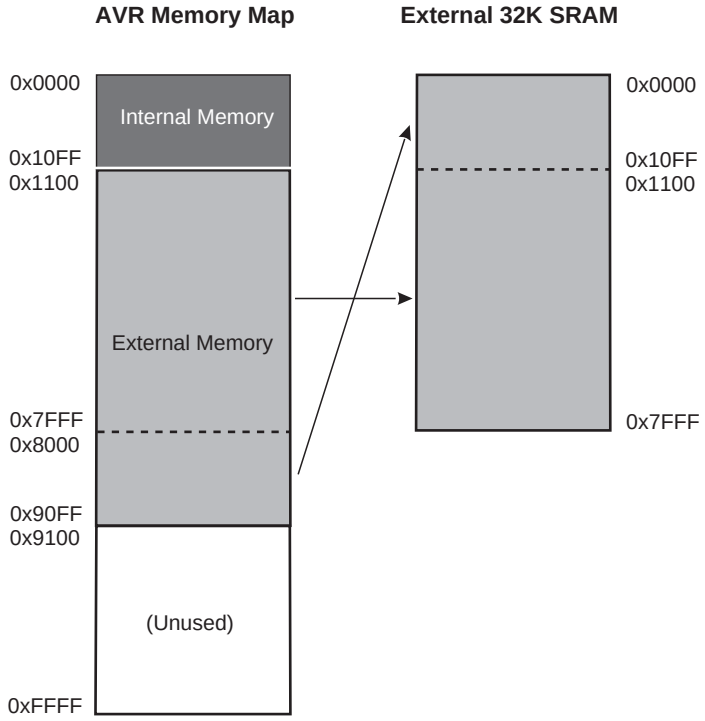
**Table 4.** Port C Pins Released as Normal Port Pins when the External Memory is Enabled

| XMM2 | XMM1 | XMM0 | # Bits for External Memory Address | Released Port Pins |
|---|---|---|---|---|
| 0 | 0 | 0 | 8 (Full 60 KB space) | None |
| 0 | 0 | 1 | 7 | PC7 |
| 0 | 1 | 0 | 6 | PC7 .. PC6 |
| 0 | 1 | 1 | 5 | PC7 .. PC5 |
| 1 | 0 | 0 | 4 | PC7 .. PC4 |
| 1 | 0 | 1 | 3 | PC7 .. PC3 |
| 1 | 1 | 0 | 2 | PC7 .. PC2 |
| 1 | 1 | 1 | No Address high bits | Full Port C |

### Using all Locations of External Memory Smaller than 64 KB

Since the external memory is mapped after the internal memory as shown in Figure 12, the external memory is not addressed when addressing the first 4,352 bytes of data space. It may appear that the first 4,352 bytes of the external memory are inaccessible (external memory addresses 0x0000 to 0x10FF). However, when connecting an external memory smaller than 64 KB, for example 32 KB, these locations are easily accessed simply by addressing from address 0x8000 to 0x90FF. Since the External Memory Address bit A15 is not connected to the external memory, addresses 0x8000 to 0x90FF will appear as addresses 0x0000 to 0x10FF for the external memory. Addressing above address 0x90FF is not recommended, since this will address an external memory location that is already accessed by another (lower) address. To the Application software, the external 32 KB memory will appear as one linear 32 KB address space from 0x1100 to 0x90FF. This is illustrated in Figure 18.

**Figure 18.** Address Map with 32 KB External Memory




### Using all 64KB Locations of External Memory

Since the External Memory is mapped after the Internal Memory as shown in Figure 12, only 60KB of External Memory is available by default (address space 0x0000 to 0x10FF is reserved for internal memory). However, it is possible to take advantage of the entire External Memory by masking the higher address bits to zero. This can be done by using the XMMn bits and control by software the most significant bits of the address. By setting Port C to output 0x00, and releasing the most significant bits for normal Port Pin operation, the Memory Interface will address 0x0000 - 0x1FFF. See the following code examples.

Assembly Code Example[1]

```
; OFFSET is defined to 0x2000 to ensure
; external memory access
; Configure Port C (address high byte) to
; output 0x00 when the pins are released
; for normal Port Pin operation
ldi   r16, 0xFF
out   DDRC, r16
ldi   r16, 0x00
out   PORTC, r16
; release PC7:5
ldi   r16, (1<<XMM1)|(1<<XMM0)
sts   XMCRB, r16
; write 0xAA to address 0x0001 of external
; memory
ldi   r16, 0xaa
sts   0x0001+OFFSET, r16
; re-enable PC7:5 for external memory
ldi   r16, (0<<XMM1)|(0<<XMM0)
sts   XMCRB, r16
; store 0x55 to address (OFFSET + 1) of
; external memory
ldi   r16, 0x55
sts   0x0001+OFFSET, r16
```

C Code Example[1]

```
#define OFFSET 0x2000

void XRAM_example(void)
{
unsigned char *p = (unsigned char *) (OFFSET + 1);

DDRC = 0xFF;
PORTC = 0x00;

XMCRB = (1<<XMM1) | (1<<XMM0);

*p = 0xaa;

XMCRB = 0x00;

*p = 0x55;
}
```

Note: 1. The example code assumes that the part specific header file is included.

Care must be exercised using this option as most of the memory is masked away.

## General Purpose I/O Registers

The AT90CAN128 contains three General Purpose I/O Registers. These registers can be used for storing any information, and they are particularly useful for storing global variables and status flags.

The General Purpose I/O Register 0, within the address range 0x00 - 0x1F, is directly bit-accessible using the SBI, CBI, SBIS, and SBIC instructions.

### General Purpose I/O Register 2 – GPIOR2

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | GPIOR07 | GPIOR06 | GPIOR05 | GPIOR04 | GPIOR03 | GPIOR02 | GPIOR01 | GPIOR00 | GPIOR2 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### General Purpose I/O Register 1 – GPIOR1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | GPIOR17 | GPIOR16 | GPIOR15 | GPIOR14 | GPIOR13 | GPIOR12 | GPIOR11 | GPIOR10 | GPIOR1 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### General Purpose I/O Register 0 – GPIOR0

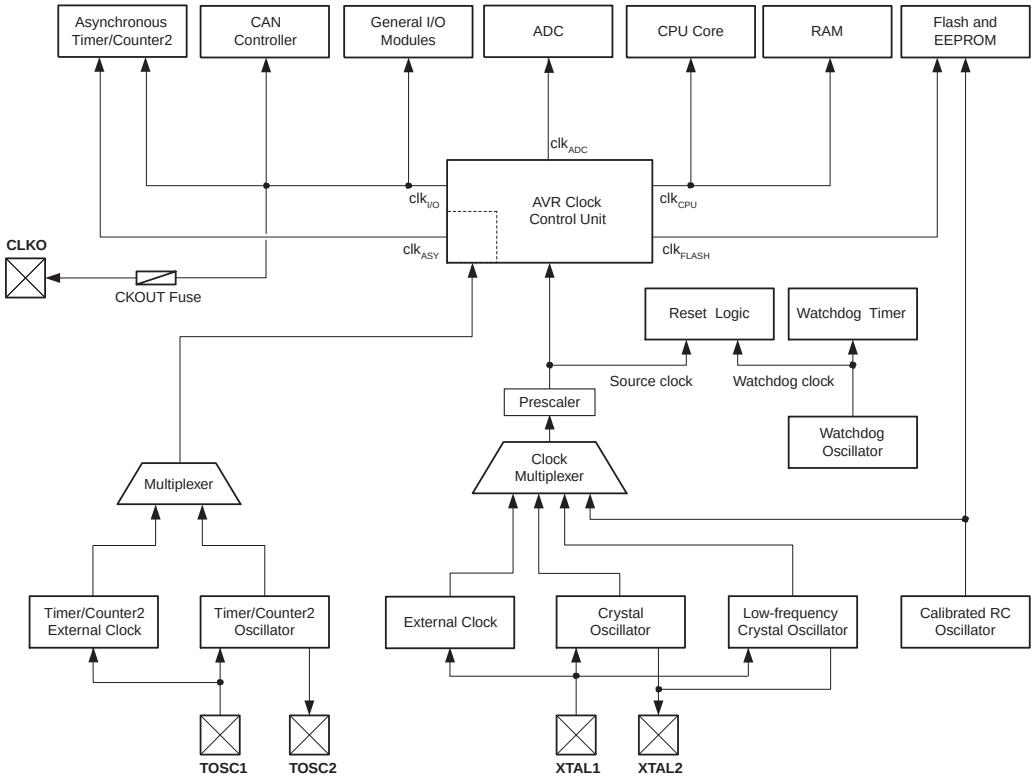
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | GPIOR27 | GPIOR26 | GPIOR25 | GPIOR24 | GPIOR23 | GPIOR22 | GPIOR21 | GPIOR20 | GPIOR0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

# System Clock

## Clock Systems and their Distribution

Figure 19 presents the principal clock systems in the AVR and their distribution. All of the clocks need not be active at a given time. In order to reduce power consumption, the clocks to unused modules can be halted by using different sleep modes, as described in “Power Management and Sleep Modes” on page 43. The clock systems are detailed below.

**Figure 19.** Clock Distribution




**CPU Clock – $clk_{CPU}$**

The CPU clock is routed to parts of the system concerned with operation of the AVR core. Examples of such modules are the General Purpose Register File, the Status Register and the data memory holding the Stack Pointer. Halting the CPU clock inhibits the core from performing general operations and calculations.

**I/O Clock – $clk_{I/O}$**

The I/O clock is used by the majority of the I/O modules, like Timer/Counters, SPI, CAN, USART. The I/O clock is also used by the External Interrupt module, but note that some external interrupts are detected by asynchronous logic, allowing such interrupts to be detected even if the I/O clock is halted. Also note that address recognition in the TWI module is carried out asynchronously when $clk_{I/O}$ is halted, enabling TWI address reception in all sleep modes.

**Flash Clock – $clk_{FLASH}$**

The Flash clock controls operation of the Flash interface. The Flash clock is usually active simultaneously with the CPU clock.

**Asynchronous Timer Clock – $clk_{ASY}$**

The Asynchronous Timer clock allows the Asynchronous Timer/Counter to be clocked directly from an external clock or an external 32 kHz clock crystal. The dedicated clock

domain allows using this Timer/Counter as a real-time counter even when the device is in sleep mode.

**ADC Clock – $clk_{ADC}$**

The ADC is provided with a dedicated clock domain. This allows halting the CPU and I/O clocks in order to reduce noise generated by digital circuitry. This gives more accurate ADC conversion results.

## Clock Sources

The device has the following clock source options, selectable by Flash Fuse bits as shown below. The clock from the selected source is input to the AVR clock generator, and routed to the appropriate modules.

**Table 5.** Device Clocking Options Select(1)

| Device Clocking Option | CKSEL3..0 |
|---|---|
| External Crystal/Ceramic Resonator | 1111 - 1000 |
| External Low-frequency Crystal | 0111 - 0100 |
| Calibrated Internal RC Oscillator | 0010 |
| External Clock | 0000 |
| Reserved | 0011, 0001 |

Note: 1. For all fuses “1” means unprogrammed while “0” means programmed.

The various choices for each clocking option is given in the following sections. When the CPU wakes up from Power-down or Power-save, the selected clock source is used to time the start-up, ensuring stable Oscillator operation before instruction execution starts. When the CPU starts from reset, there is an additional delay allowing the power to reach a stable level before starting normal operation. The Watchdog Oscillator is used for timing this real-time part of the start-up time. The number of WDT Oscillator cycles used for each time-out is shown in Table 6. The frequency of the Watchdog Oscillator is voltage dependent as shown in “AT90CAN128 Typical Characteristics” on page 374.

**Table 6.** Number of Watchdog Oscillator Cycles

| Typ Time-out ($V_{CC}$ = 5.0V) | Typ Time-out ($V_{CC}$ = 3.0V) | Number of Cycles |
|---|---|---|
| 4.1 ms | 4.3 ms | 4K (4,096) |
| 65 ms | 69 ms | 64K (65,536) |

## Default Clock Source

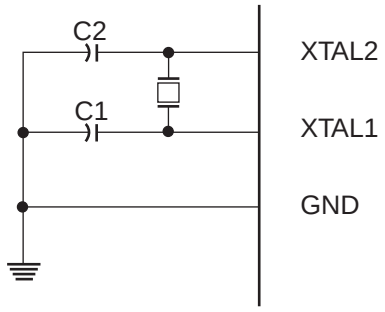
The device is shipped with CKSEL = “0010”, SUT = “10”, and CKDIV8 programmed. The default clock source setting is the Internal RC Oscillator with longest start-up time and an initial system clock prescaling of 8. This default setting ensures that all users can make their desired clock source setting using an In-System or Parallel programmer.

## Crystal Oscillator

XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier which can be configured for use as an On-chip Oscillator, as shown in Figure 20. Either a quartz crystal or a ceramic resonator may be used.

C1 and C2 should always be equal for both crystals and resonators. The optimal value of the capacitors depends on the crystal or resonator in use, the amount of stray capacitance, and the electromagnetic noise of the environment. Some initial guidelines for choosing capacitors for use with crystals are given in Table 7. For ceramic resonators, the capacitor values given by the manufacturer should be used. For more information on how to choose capacitors and other details on Oscillator operation, refer to the Multipurpose Oscillator Application Note.

**Figure 20.** Crystal Oscillator Connections




The Oscillator can operate in three different modes, each optimized for a specific frequency range. The operating mode is selected by the fuses CKSEL3..1 as shown in Table 7.

**Table 7.** Crystal Oscillator Operating Modes

| CKSEL3..1 | Frequency Range (MHz) | Recommended Range for Capacitors C1 and C2 for Use with Crystals (pF) |
|---|---|---|
| 100[1] | 0.4 - 0.9 | 12 - 22 |
| 101 | 0.9 - 3.0 | 12 - 22 |
| 110 | 3.0 - 8.0 | 12 - 22 |
| 111 | 8.0 - 16.0 | 12 - 22 |

Notes: 1. This option should not be used with crystals, only with ceramic resonators.

The CKSEL0 Fuse together with the SUT1..0 Fuses select the start-up times as shown in Table 8.

**Table 8.** Start-up Times for the Oscillator Clock Selection

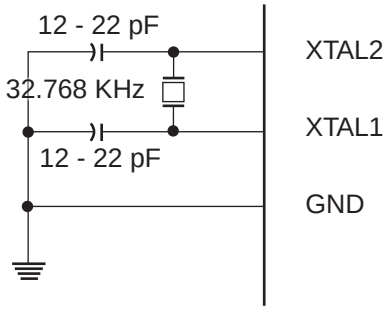
| CKSEL0 | SUT1..0 | Start-up Time from Power-down and Power-save | Additional Delay from Reset ($V_{CC}$ = 5.0V) | Recommended Usage |
|---|---|---|---|---|
| 0 | 00 | 258 CK(1) | 14CK + 4.1 ms | Ceramic resonator, fast rising power |
| 0 | 01 | 258 CK(1) | 14CK + 65 ms | Ceramic resonator, slowly rising power |
| 0 | 10 | 1K CK(2) | 14CK | Ceramic resonator, BOD enabled |
| 0 | 11 | 1K CK(2) | 14CK + 4.1 ms | Ceramic resonator, fast rising power |
| 1 | 00 | 1K CK(2) | 14CK + 65 ms | Ceramic resonator, slowly rising power |
| 1 | 01 | 16K CK | 14CK | Crystal Oscillator, BOD enabled |
| 1 | 10 | 16K CK | 14CK + 4.1 ms | Crystal Oscillator, fast rising power |
| 1 | 11 | 16K CK | 14CK + 65 ms | Crystal Oscillator, slowly rising power |

Notes: 1. These options should only be used when not operating close to the maximum frequency of the device, and only if frequency stability at start-up is not important for the application. These options are not suitable for crystals.
2. These options are intended for use with ceramic resonators and will ensure frequency stability at start-up. They can also be used with crystals when not operating close to the maximum frequency of the device, and if frequency stability at start-up is not important for the application.

## Low-frequency Crystal Oscillator

To use a 32.768 kHz watch crystal as the clock source for the device, the low-frequency crystal Oscillator must be selected by setting the CKSEL Fuses to “0100”, “0101”, “0110”, or “0111”. The crystal should be connected as shown in Figure 21.

**Figure 21.** Low-frequency Crystal Oscillator Connections




12-22 pF capacitors may be necessary if the parasitic impedance (pads, wires & PCB) is very low.

When this Oscillator is selected, start-up times are determined by the SUT1..0 fuses as shown in Table 9 and CKSEL1..0 fuses as shown in Table 10.

**Table 9.** Start-up Times for the Low-frequency Crystal Oscillator Clock Selection

| SUT1..0 | Additional Delay from Reset ($V_{CC}$ = 5.0V) | Recommended Usage |
|---|---|---|
| 00 | 14CK | Fast rising power or BOD enabled |
| 01 | 14CK + 4.1 ms | Slowly rising power |
| 10 | 14CK + 65 ms | Stable frequency at start-up |
| 11 | Reserved | |

**Table 10.** Start-up Times for the Low-frequency Crystal Oscillator Clock Selection

| CKSEL3..0 | Start-up Time from Power-down and Power-save | Recommended Usage |
|---|---|---|
| 0100(1) | 1K CK | |
| 0101 | 32K CK | Stable frequency at start-up |
| 0110(1) | 1K CK | |
| 0111 | 32K CK | Stable frequency at start-up |

Note: 1. These options should only be used if frequency stability at start-up is not important for the application

## Calibrated Internal RC Oscillator

The calibrated internal RC Oscillator provides a fixed 8.0 MHz clock. The frequency is nominal value at 3V and 25°C. If 8 MHz frequency exceeds the specification of the device (depends on $V_{CC}$), the CKDIV8 Fuse must be programmed in order to divide the internal frequency by 8 during start-up. The device is shipped with the CKDIV8 Fuse programmed. See “System Clock Prescaler” on page 41 for more details. This clock may be selected as the system clock by programming the CKSEL Fuses as shown in Table 11. If selected, it will operate with no external components. During reset, hardware loads the calibration byte into the OSCCAL Register and thereby automatically calibrates the RC Oscillator. At 5V and 25°C, this calibration gives a frequency within ± 10% of the nominal frequency. Using calibration methods as described in application notes available at www.atmel.com/avr it is possible to achieve ± 2% accuracy at any given VCC and temperature. When this Oscillator is used as the chip clock, the Watchdog Oscillator will still be used for the Watchdog Timer and for the Reset Time-out. For more information on the pre-programmed calibration value, see the section “Calibration Byte” on page 328.

**Table 11.** Internal Calibrated RC Oscillator Operating Modes(1)

| CKSEL3..0 | Nominal Frequency |
|---|---|
| 0010 | 8.0 MHz |

Note: 1. The device is shipped with this option selected.

When this Oscillator is selected, start-up times are determined by the SUT Fuses as shown in Table 12.

**Table 12.** Start-up times for the internal calibrated RC Oscillator clock selection

| SUT1..0 | Start-up Time from Power-down and Power-save | Additional Delay from Reset ($V_{CC}$ = 5.0V) | Recommended Usage |
|---|---|---|---|
| 00 | 6 CK | 14CK | BOD enabled |
| 01 | 6 CK | 14CK + 4.1 ms | Fast rising power |
| 10(1) | 6 CK | 14CK + 65 ms | Slowly rising power |
| 11 | Reserved | | |

Note: 1. The device is shipped with this option selected.

### Oscillator Calibration Register – OSCCAL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | CAL6 | CAL5 | CAL4 | CAL3 | CAL2 | CAL1 | CAL0 | OSCCAL |
| Read/Write | R | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | <----- ------ | | Device Specific Calibration Value | | | ------ -----> | | |

- **Bit 7 – Reserved Bit**

This bit is reserved for future use.

- **Bits 6..0 – CAL6..0: Oscillator Calibration Value**

Writing the calibration byte to this address will trim the internal Oscillator to remove process variations from the Oscillator frequency. This is done automatically during Chip Reset. When OSCCAL is zero, the lowest available frequency is chosen. Writing non-zero values to this register will increase the frequency of the internal Oscillator. Writing 0x7F to the register gives the highest available frequency. The calibrated Oscillator is used to time EEPROM and Flash access. If EEPROM or Flash is written, do not calibrate to more than 10% above the nominal frequency. Otherwise, the EEPROM or Flash write may fail. Note that the Oscillator is intended for calibration to 8.0 MHz. Tuning to other values is not guaranteed, as indicated in Table 13.

**Table 13.** Internal RC Oscillator Frequency Range.

| OSCCAL Value | Min Frequency in Percentage of Nominal Frequency | Max Frequency in Percentage of Nominal Frequency |
|---|---|---|
| 0x00 | 50% | 100% |
| 0x3F | 75% | 150% |
| 0x7F | 100% | 200% |

## External Clock

To drive the device from an external clock source, XTAL1 should be driven as shown in Figure 22. To run the device on an external clock, the CKSEL Fuses must be programmed to “0000”.

**Figure 22.** External Clock Drive Configuration

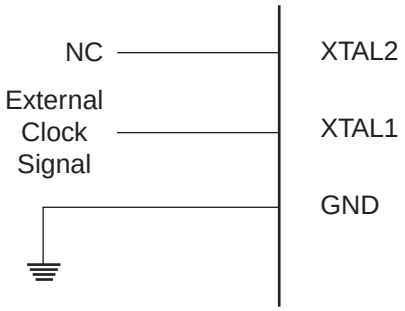



**Table 14.** External Clock Frequency

| CKSEL3..0 | Frequency Range |
|---|---|
| 0000 | 0 - 16 MHz |

When this clock source is selected, start-up times are determined by the SUT Fuses as shown in Table 15.

**Table 15.** Start-up Times for the External Clock Selection

| SUT1..0 | Start-up Time from Power-down and Power-save | Additional Delay from Reset ($V_{CC}$ = 5.0V) | Recommended Usage |
|---|---|---|---|
| 00 | 6 CK | 14CK | BOD enabled |
| 01 | 6 CK | 14CK + 4.1 ms | Fast rising power |
| 10 | 6 CK | 14CK + 65 ms | Slowly rising power |
| 11 | Reserved | | |

When applying an external clock, it is required to avoid sudden changes in the applied clock frequency to ensure stable operation of the MCU. A variation in frequency of more than 2% from one clock cycle to the next can lead to unpredictable behavior. It is required to ensure that the MCU is kept in Reset during such changes in the clock frequency.

Note that the System Clock Prescaler can be used to implement run-time changes of the internal clock frequency while still ensuring stable operation. Refer to “System Clock Prescaler” on page 41 for details.

## Clock Output Buffer

When the CKOUT Fuse is programmed, the system Clock will be output on CLKO. This mode is suitable when chip clock is used to drive other circuits on the system. The clock will be output also during reset and the normal operation of I/O pin will be overridden when the fuse is programmed. Any clock source, including internal RC Oscillator, can be selected when CLKO serves as clock output. If the System Clock Prescaler is used, it is the divided system clock that is output (CKOUT Fuse programmed).

## Timer/Counter2 Oscillator

For AVR microcontrollers with Timer/Counter2 Oscillator pins (TOSC1 and TOSC2), the crystal is connected directly between the pins. The Oscillator is optimized for use with a 32.768 kHz watch crystal. 12-22 pF capacitors may be necessary if the parasitic impedance (pads, wires & PCB) is very low.

AT90CAN128 share the Timer/Counter2 Oscillator Pins (TOSC1 and TOSC2) with PG4 and PG3. This means that both PG4 and PG3 can only be used when the Timer/Counter2 Oscillator is not enable.

Applying an external clock source to TOSC1 can be done in asynchronous operation if EXTCLK in the ASSR Register is written to logic one. See “Asynchronous operation of the Timer/Counter2” on page 154 for further description on selecting external clock as input instead of a 32 kHz crystal. In this configuration, PG4 cannot be used but PG3 is available.

## System Clock Prescaler

The AT90CAN128 system clock can be divided by setting the Clock Prescaler Register – CLKPR. This feature can be used to decrease power consumption when the requirement for processing power is low. This can be used with all clock source options, and it will affect the clock frequency of the CPU and all synchronous peripherals. $clk_{I/O}$, $clk_{ADC}$, $clk_{CPU}$, and $clk_{FLASH}$ are divided by a factor as shown in Table 16.

### Clock Prescaler Register – CLKPR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **CLKPCE** | – | – | – | **CLKPS3** | **CLKPS2** | **CLKPS1** | **CLKPS0** | **CLKPR** |
| Read/Write | R/W | R | R | R | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | See Bit Description | | | | |

- **Bit 7 – CLKPCE: Clock Prescaler Change Enable**

The CLKPCE bit must be written to logic one to enable change of the CLKPS bits. The CLKPCE bit is only updated when the other bits in CLKPR are simultaneously written to zero. CLKPCE is cleared by hardware four cycles after it is written or when CLKPS bits are written. Rewriting the CLKPCE bit within this time-out period does neither extend the time-out period, nor clear the CLKPCE bit.

- **Bit 6..0 – Reserved Bits**

These bits are reserved for future use.

- **Bits 3..0 – CLKPS3..0: Clock Prescaler Select Bits 3 - 0**

These bits define the division factor between the selected clock source and the internal system clock. These bits can be written run-time to vary the clock frequency to suit the application requirements. As the divider divides the master clock input to the MCU, the speed of all synchronous peripherals is reduced when a division factor is used. The division factors are given in Table 16.

To avoid unintentional changes of clock frequency, a special write procedure must be followed to change the CLKPS bits:

1. Write the Clock Prescaler Change Enable (CLKPCE) bit to one and all other bits in CLKPR to zero.
2. Within four cycles, write the desired value to CLKPS while writing a zero to CLKPCE.

Interrupts must be disabled when changing prescaler setting to make sure the write procedure is not interrupted.

The CKDIV8 Fuse determines the initial value of the CLKPS bits. If CKDIV8 is unprogrammed, the CLKPS bits will be reset to “0000”. If CKDIV8 is programmed, CLKPS bits are reset to “0011”, giving a division factor of 8 at start up. This feature should be used if the selected clock source has a higher frequency than the maximum frequency of the device at the present operating conditions. Note that any value can be written to the

CLKPS bits regardless of the CKDIV8 Fuse setting. The Application software must ensure that a sufficient division factor is chosen if the selected clock source has a higher frequency than the maximum frequency of the device at the present operating conditions. The device is shipped with the CKDIV8 Fuse programmed.

**Table 16.** Clock Prescaler Select

| CLKPS3 | CLKPS2 | CLKPS1 | CLKPS0 | Clock Division Factor |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 2 |
| 0 | 0 | 1 | 0 | 4 |
| 0 | 0 | 1 | 1 | 8 |
| 0 | 1 | 0 | 0 | 16 |
| 0 | 1 | 0 | 1 | 32 |
| 0 | 1 | 1 | 0 | 64 |
| 0 | 1 | 1 | 1 | 128 |
| 1 | 0 | 0 | 0 | 256 |
| 1 | 0 | 0 | 1 | Reserved |
| 1 | 0 | 1 | 0 | Reserved |
| 1 | 0 | 1 | 1 | Reserved |
| 1 | 1 | 0 | 0 | Reserved |
| 1 | 1 | 0 | 1 | Reserved |
| 1 | 1 | 1 | 0 | Reserved |
| 1 | 1 | 1 | 1 | Reserved |

Note: When the system clock is divided, Timer/Counter0 can be used with Asynchronous clock only. The frequency of the asynchronous clock must be lower than 1/4th of the frequency of the scaled down Source clock. Otherwise, interrupts may be lost, and accessing the Timer/Counter0 registers may fail.

## Power Management and Sleep Modes

Sleep modes enable the application to shut down unused modules in the MCU, thereby saving power. The AVR provides various sleep modes allowing the user to tailor the power consumption to the application's requirements.

To enter any of the five sleep modes, the SE bit in SMCR must be written to logic one and a SLEEP instruction must be executed. The SM2, SM1, and SM0 bits in the SMCR Register select which sleep mode (Idle, ADC Noise Reduction, Power-down, Power-save, or Standby) will be activated by the SLEEP instruction. See Table 17 for a summary. If an enabled interrupt occurs while the MCU is in a sleep mode, the MCU wakes up. The MCU is then halted for four cycles in addition to the start-up time, executes the interrupt routine, and resumes execution from the instruction following SLEEP. The contents of the register file and SRAM are unaltered when the device wakes up from sleep. If a reset occurs during sleep mode, the MCU wakes up and executes from the Reset Vector.

Figure 19 on page 34 presents the different clock systems in the AT90CAN128, and their distribution. The figure is helpful in selecting an appropriate sleep mode.

**Sleep Mode Control Register – SMCR**

The Sleep Mode Control Register contains control bits for power management.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | SM2 | SM1 | SM0 | SE | SMCR |
| Read/Write | R | R | R | R | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7..4 – Reserved Bits**

These bits are reserved for future use.

- **Bits 3..1 – SM2..0: Sleep Mode Select Bits 2, 1, and 0**

These bits select between the five available sleep modes as shown in Table 17.

**Table 17.** Sleep Mode Select

| SM2 | SM1 | SM0 | Sleep Mode |
|---|---|---|---|
| 0 | 0 | 0 | Idle |
| 0 | 0 | 1 | ADC Noise Reduction |
| 0 | 1 | 0 | Power-down |
| 0 | 1 | 1 | Power-save |
| 1 | 0 | 0 | Reserved |
| 1 | 0 | 1 | Reserved |
| 1 | 1 | 0 | Standby(1) |
| 1 | 1 | 1 | Reserved |

Note: 1. Standby mode is only recommended for use with external crystals or resonators.

- **Bit 1 – SE: Sleep Enable**

The SE bit must be written to logic one to make the MCU enter the sleep mode when the SLEEP instruction is executed. To avoid the MCU entering the sleep mode unless it is the programmer's purpose, it is recommended to write the Sleep Enable (SE) bit to one

just before the execution of the SLEEP instruction and to clear it immediately after waking up.

### Idle Mode

When the SM2..0 bits are written to 000, the SLEEP instruction makes the MCU enter Idle mode, stopping the CPU but allowing SPI, CAN, USART, Analog Comparator, ADC, Two-wire Serial Interface, Timer/Counters, Watchdog, and the interrupt system to continue operating. This sleep mode basically halts $clk_{CPU}$ and $clk_{FLASH}$, while allowing the other clocks to run.

Idle mode enables the MCU to wake up from external triggered interrupts as well as internal ones like the Timer Overflow and USART Transmit Complete interrupts. If wake-up from the Analog Comparator interrupt is not required, the Analog Comparator can be powered down by setting the ACD bit in the Analog Comparator Control and Status Register – ACSR. This will reduce power consumption in Idle mode. If the ADC is enabled, a conversion starts automatically when this mode is entered.

### ADC Noise Reduction Mode

When the SM2..0 bits are written to 001, the SLEEP instruction makes the MCU enter ADC Noise Reduction mode, stopping the CPU but allowing the ADC, the External Interrupts, the Two-wire Serial Interface address watch, Timer/Counter2, CAN and the Watchdog to continue operating (if enabled). This sleep mode basically halts $clk_{I/O}$, $clk_{CPU}$, and $clk_{FLASH}$, while allowing the other clocks to run.

This improves the noise environment for the ADC, enabling higher resolution measurements. If the ADC is enabled, a conversion starts automatically when this mode is entered. Apart from the ADC Conversion Complete interrupt, only an External Reset, a Watchdog Reset, a Brown-out Reset, a Two-wire Serial Interface address match interrupt, a Timer/Counter2 interrupt, an SPM/EEPROM ready interrupt, an External Level Interrupt on INT7:4, or an External Interrupt on INT3:0 can wake up the MCU from ADC Noise Reduction mode.

### Power-down Mode

When the SM2..0 bits are written to 010, the SLEEP instruction makes the MCU enter Power-down mode. In this mode, the External Oscillator is stopped, while the External Interrupts, the Two-wire Serial Interface address watch, and the Watchdog continue operating (if enabled). Only an External Reset, a Watchdog Reset, a Brown-out Reset, a Two-wire Serial Interface address match interrupt, an External Level Interrupt on INT7:4, or an External Interrupt on INT3:0 can wake up the MCU. This sleep mode basically halts all generated clocks, allowing operation of asynchronous modules only.

Note that if a level triggered interrupt is used for wake-up from Power-down mode, the changed level must be held for some time to wake up the MCU. Refer to “External Interrupts” on page 88 for details.

When waking up from Power-down mode, there is a delay from the wake-up condition occurs until the wake-up becomes effective. This allows the clock to restart and become stable after having been stopped. The wake-up period is defined by the same CKSEL fuses that define the Reset Time-out period, as described in “Clock Sources” on page 35.

### Power-save Mode

When the SM2..0 bits are written to 011, the SLEEP instruction makes the MCU enter Power-save mode. This mode is identical to Power-down, with one exception:

If Timer/Counter2 is clocked asynchronously, i.e., the AS2 bit in ASSR is set, Timer/Counter2 will run during sleep. The device can wake up from either Timer Overflow or Output Compare event from Timer/Counter2 if the corresponding

Timer/Counter2 interrupt enable bits are set in TIMSK2, and the global interrupt enable bit in SREG is set.

If the Asynchronous Timer is **NOT** clocked asynchronously, Power-down mode is recommended instead of Power-save mode because the contents of the registers in the asynchronous timer should be considered undefined after wake-up in Power-save mode if AS2 is 0.

This sleep mode basically halts all clocks except $clk_{ASY}$, allowing operation only of asynchronous modules, including Timer/Counter2 if clocked asynchronously.

## Standby Mode

When the SM2..0 bits are 110 and an External Crystal/Resonator clock option is selected, the SLEEP instruction makes the MCU enter Standby mode. This mode is identical to Power-down with the exception that the Oscillator is kept running. From Standby mode, the device wakes up in 6 clock cycles.

**Table 18.** Active Clock Domains and Wake-up Sources in the Different Sleep Modes.

| | Active Clock Domains | | | | | Oscillators | | Wake-up Sources | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Sleep Mode** | **$clk_{CPU}$** | **$clk_{FLASH}$** | **$clk_{IO}$** | **$clk_{ADC}$** | **$clk_{ASY}$** | **Main Clock Source Enabled** | **Timer Osc Enabled** | **INT7:0** | **TWI Address Match** | **Timer2** | **SPM/ EEPROM Ready** | **ADC** | **Other I/O** |
| Idle | | | X | X | X | X | X(2) | X | X | X | X | X | X |
| ADC Noise Reduction | | | | X | X | X | X(2) | X(3) | X | X(2) | X | X | |
| Power-down | | | | | | | | X(3) | X | | | | |
| Power-save | | | | | X(2) | | X(2) | X(3) | X | X(2) | | | |
| Standby(1) | | | | | | X | | X(3) | X | | | | |

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If AS2 bit in ASSR is set.
3. Only INT3:0 or level interrupt INT7:4.

## Minimizing Power Consumption

There are several issues to consider when trying to minimize the power consumption in an AVR controlled system. In general, sleep modes should be used as much as possible, and the sleep mode should be selected so that as few as possible of the device's functions are operating. All functions not needed should be disabled. In particular, the following modules may need special consideration when trying to achieve the lowest possible power consumption.

### Analog to Digital Converter

If enabled, the ADC will be enabled in all sleep modes. To save power, the ADC should be disabled before entering any sleep mode. When the ADC is turned off and on again, the next conversion will be an extended conversion. Refer to "Analog to Digital Converter - ADC" on page 265 for details on ADC operation.

### Analog Comparator

When entering Idle mode, the Analog Comparator should be disabled if not used. When entering ADC Noise Reduction mode, the Analog Comparator should be disabled. In other sleep modes, the Analog Comparator is automatically disabled. However, if the Analog Comparator is set up to use the Internal Voltage Reference as input, the Analog Comparator should be disabled in all sleep modes. Otherwise, the Internal Voltage Reference will be enabled, independent of sleep mode. Refer to "Analog Comparator" on page 262 for details on how to configure the Analog Comparator.

**Brown-out Detector**

If the Brown-out Detector is not needed by the application, this module should be turned off. If the Brown-out Detector is enabled by the BODLEVEL Fuses, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to “Brown-out Detection” on page 50 for details on how to configure the Brown-out Detector.

**Internal Voltage Reference**

The Internal Voltage Reference will be enabled when needed by the Brown-out Detection, the Analog Comparator or the ADC. If these modules are disabled as described in the sections above, the internal voltage reference will be disabled and it will not be consuming power. When turned on again, the user must allow the reference to start up before the output is used. If the reference is kept on in sleep mode, the output can be used immediately. Refer to “Internal Voltage Reference” on page 52 for details on the start-up time.

**Watchdog Timer**

If the Watchdog Timer is not needed in the application, the module should be turned off. If the Watchdog Timer is enabled, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to “Watchdog Timer” on page 53 for details on how to configure the Watchdog Timer.

**Port Pins**

When entering a sleep mode, all port pins should be configured to use minimum power. The most important is then to ensure that no pins drive resistive loads. In sleep modes where both the I/O clock ($clk_{I/O}$) and the ADC clock ($clk_{ADC}$) are stopped, the input buffers of the device will be disabled. This ensures that no power is consumed by the input logic when not needed. In some cases, the input logic is needed for detecting wake-up conditions, and it will then be enabled. Refer to the section “Digital Input Enable and Sleep Modes” on page 65 for details on which pins are enabled. If the input buffer is enabled and the input signal is left floating or have an analog signal level close to $V_{CC}/2$, the input buffer will use excessive power.

For analog input pins, the digital input buffer should be disabled at all times. An analog signal level close to $V_{CC}/2$ on an input pin can cause significant current even in active mode. Digital input buffers can be disabled by writing to the Digital Input Disable Registers (DIDR1 and DIDR0). Refer to “Digital Input Disable Register 1 – DIDR1” on page 264 and “Digital Input Disable Register 0 – DIDR0” on page 283 for details.

**JTAG Interface and On-chip Debug System**

If the On-chip debug system is enabled by OCDEN Fuse and the chip enter sleep mode, the main clock source is enabled, and hence, always consumes power. In the deeper sleep modes, this will contribute significantly to the total current consumption. There are three alternative ways to avoid this:

- Disable OCDEN Fuse.
- Disable JTAGEN Fuse.
- Write one to the JTD bit in MCUCSR.

The TDO pin is left floating when the JTAG interface is enabled while the JTAG TAP controller is not shifting data. If the hardware connected to the TDO pin does not pull up the logic level, power consumption will increase. Note that the TDI pin for the next device in the scan chain contains a pull-up that avoids this problem. Writing the JTD bit in the MCUCSR register to one or leaving the JTAG fuse unprogrammed disables the JTAG interface.

## System Control and Reset

### Resetting the AVR

During reset, all I/O Registers are set to their initial values, and the program starts execution from the Reset Vector. The instruction placed at the Reset Vector must be a JMP – Absolute Jump – instruction to the reset handling routine. If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the Boot section or vice versa. The circuit diagram in Figure 23 shows the reset logic. Table 19 defines the electrical parameters of the reset circuitry.

The I/O ports of the AVR are immediately reset to their initial state when a reset source goes active. This does not require any clock source to be running.

After all reset sources have gone inactive, a delay counter is invoked, stretching the internal reset. This allows the power to reach a stable level before normal operation starts. The time-out period of the delay counter is defined by the user through the SUT and CKSEL Fuses. The different selections for the delay period are presented in “Clock Sources” on page 35.

### Reset Sources

The AT90CAN128 has five sources of reset:

- Power-on Reset. The MCU is reset when the supply voltage is below the Power-on Reset threshold ($V_{POT}$).
- External Reset. The MCU is reset when a low level is present on the $\overline{\text{RESET}}$ pin for longer than the minimum pulse length.
- Watchdog Reset. The MCU is reset when the Watchdog Timer period expires and the Watchdog is enabled.
- Brown-out Reset. The MCU is reset when the supply voltage $V_{CC}$ is below the Brown-out Reset threshold ($V_{BOT}$) and the Brown-out Detector is enabled.
- JTAG AVR Reset. The MCU is reset as long as there is a logic one in the Reset Register, one of the scan chains of the JTAG system. Refer to the section “Boundary-scan IEEE 1149.1 (JTAG)” on page 290 for details.

**Figure 23.** Reset Logic

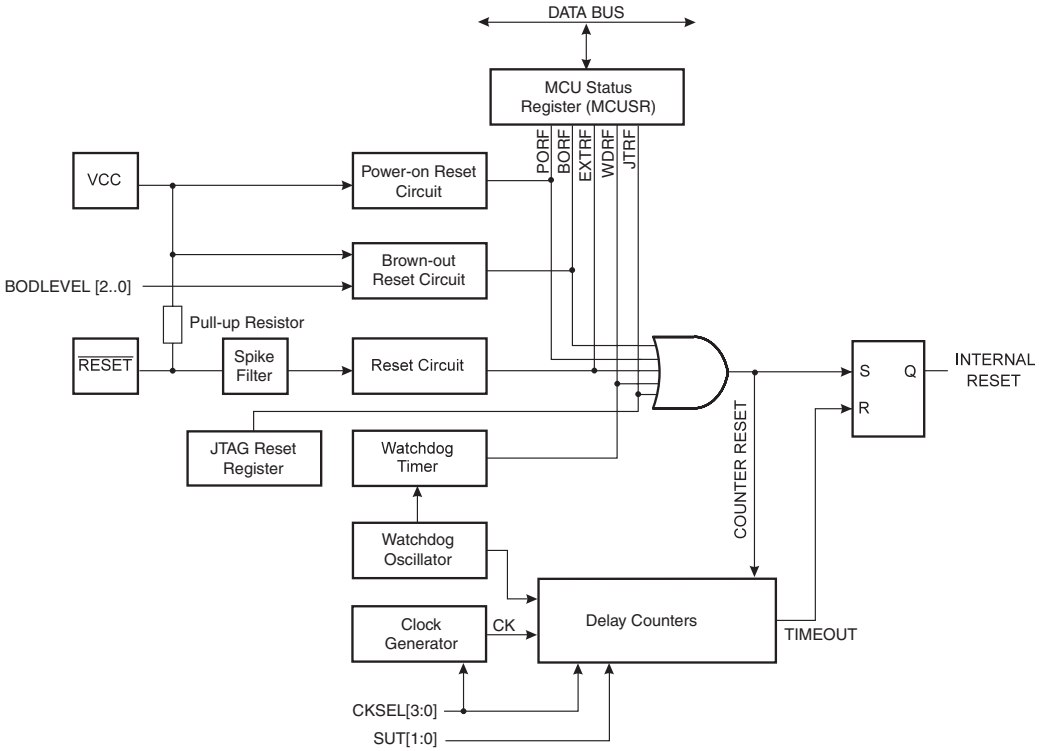



**Table 19.** Reset Characteristics

| Symbol | Parameter | Condition | Min. | Typ. | Max. | Units |
|---|---|---|---|---|---|---|
| $V_{POT}$ | Power-on Reset Threshold Voltage (rising) | | | 1.4 | 2.3 | V |
| | Power-on Reset Threshold Voltage (falling)[1] | | | 1.3 | 2.3 | V |
| $V_{RST}$ | $\overline{\text{RESET}}$ Pin Threshold Voltage | | 0.2 $V_{CC}$ | | 0.85 $V_{CC}$ | V |
| $t_{RST}$ | Minimum pulse width on $\overline{\text{RESET}}$ Pin | Vcc = 5 V, temperature = 25 °C | | 400 | | ns |

Notes: 1. The Power-on Reset will not work unless the supply voltage has been below $V_{POT}$ (falling)

### Power-on Reset

A Power-on Reset (POR) pulse is generated by an On-chip detection circuit. The detection level is defined in Table 19. The POR is activated whenever $V_{CC}$ is below the detection level. The POR circuit can be used to trigger the start-up Reset, as well as to detect a failure in supply voltage.

A Power-on Reset (POR) circuit ensures that the device is reset from Power-on. Reaching the Power-on Reset threshold voltage invokes the delay counter, which determines how long the device is kept in RESET after $V_{CC}$ rise. The RESET signal is activated again, without any delay, when $V_{CC}$ decreases below the detection level.

**Figure 24.** MCU Start-up, $\overline{\text{RESET}}$ Tied to $V_{CC}$

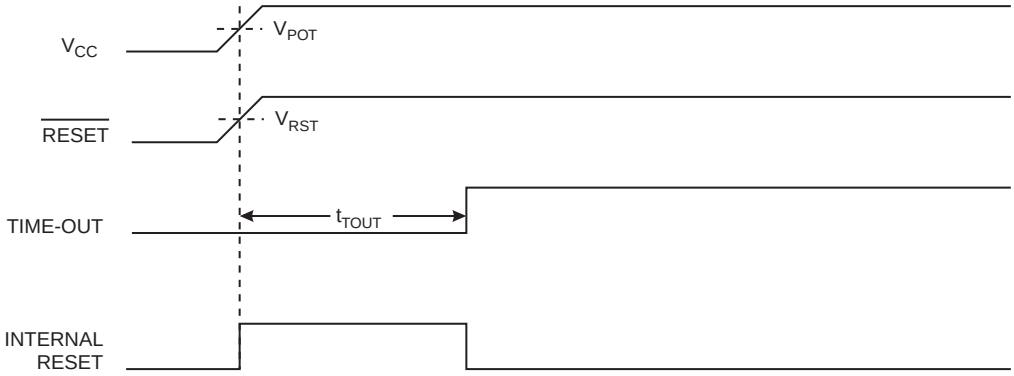
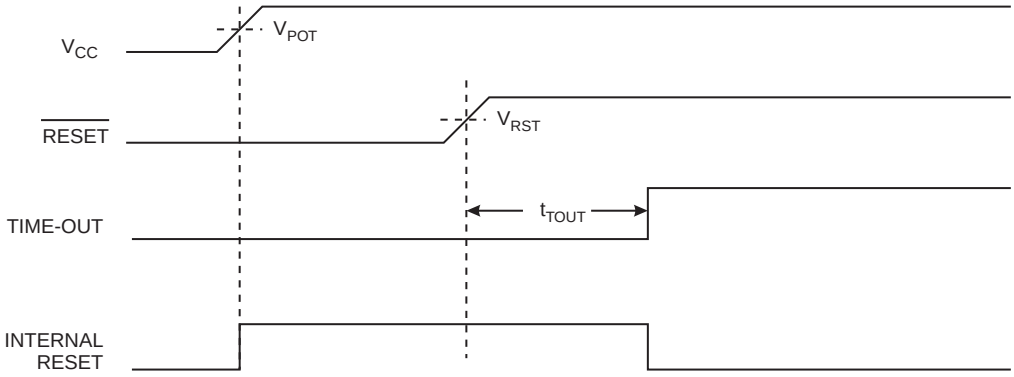


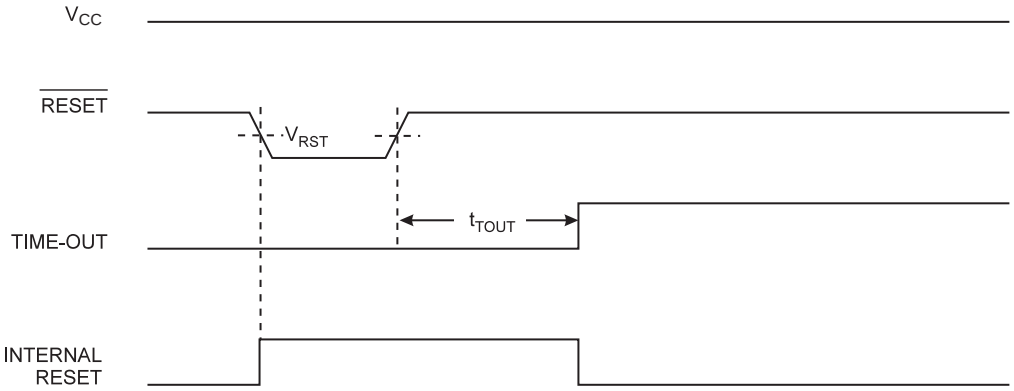

**Figure 25.** MCU Start-up, $\overline{\text{RESET}}$ Extended Externally




### External Reset

An External Reset is generated by a low level on the $\overline{\text{RESET}}$ pin. Reset pulses longer than the minimum pulse width (see Table 19) will generate a reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a reset. When the applied signal reaches the Reset Threshold Voltage – $V_{RST}$ – on its positive edge, the delay counter starts the MCU after the Time-out period – $t_{TOUT}$ – has expired.

**Figure 26.** External Reset During Operation




**Brown-out Detection**

AT90CAN128 has an On-chip Brown-out Detection (BOD) circuit for monitoring the $V_{CC}$ level during operation by comparing it to a fixed trigger level. The trigger level for the BOD can be selected by the BODLEVEL Fuses. The trigger level has a hysteresis to ensure spike free Brown-out Detection. The hysteresis on the detection level should be interpreted as $V_{BOT+} = V_{BOT} + V_{HYST}/2$ and $V_{BOT-} = V_{BOT} - V_{HYST}/2$.

**Table 20.** BODLEVEL Fuse Coding(1)

| BODLEVEL 2..0 Fuses | Min $V_{BOT}$ | Typ $V_{BOT}$ | Max $V_{BOT}$ | Units |
|---|---|---|---|---|
| 111 | BOD Disabled | | | |
| 110 | | 4.1 | | V |
| 101 | | 4.0 | | V |
| 100 | | 3.9 | | V |
| 011 | | 3.8 | | V |
| 010 | | 2.7 | | V |
| 001 | | 2.6 | | V |
| 000 | | 2.5 | | V |

Notes: 1. $V_{BOT}$ may be below nominal minimum operating voltage for some devices. For devices where this is the case, the device is tested down to $V_{CC} = V_{BOT}$ during the production test. This guarantees that a Brown-Out Reset will occur before $V_{CC}$ drops to a voltage where correct operation of the microcontroller is no longer guaranteed. The test is performed using BODLEVEL = 010 for Low Operating Voltage and BODLEVEL = 101 for High Operating Voltage .

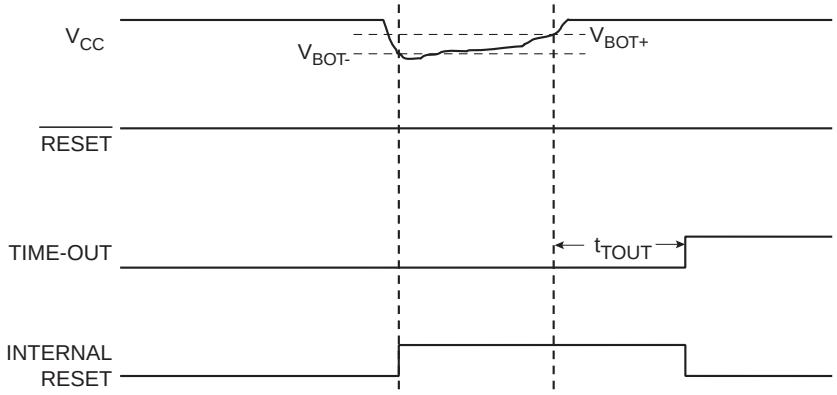
**Table 21.** Brown-out Characteristics

| Symbol | Parameter | Min. | Typ. | Max. | Units |
|---|---|---|---|---|---|
| $V_{HYST}$ | Brown-out Detector Hysteresis | | 70 | | mV |
| $t_{BOD}$ | Min Pulse Width on Brown-out Reset | | 2 | | µs |

When the BOD is enabled, and $V_{CC}$ decreases to a value below the trigger level ($V_{BOT-}$ in Figure 27), the Brown-out Reset is immediately activated. When $V_{CC}$ increases above the trigger level ($V_{BOT+}$ in Figure 27), the delay counter starts the MCU after the Time-out period $t_{TOUT}$ has expired.

The BOD circuit will only detect a drop in $V_{CC}$ if the voltage stays below the trigger level for longer than $t_{BOD}$ given in Table 19.

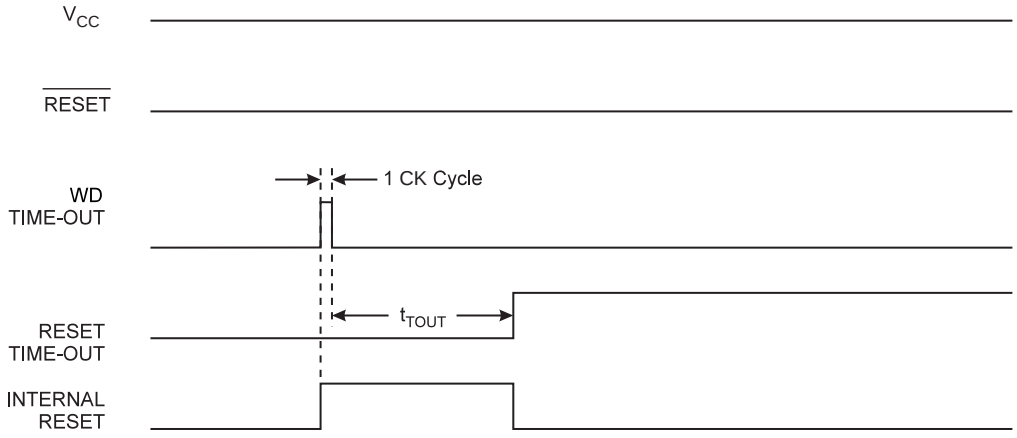
**Figure 27.** Brown-out Reset During Operation




## Watchdog Reset

When the Watchdog times out, it will generate a short reset pulse of one CK cycle duration. On the falling edge of this pulse, the delay timer starts counting the Time-out period $t_{TOUT}$. Refer to page 53 for details on operation of the Watchdog Timer.

**Figure 28.** Watchdog Reset During Operation




## MCU Status Register – MCUSR

The MCU Status Register provides information on which reset source caused an MCU reset.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | JTRF | WDRF | BORF | EXTRF | PORF | MCUSR |
| Read/Write | R | R | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | See Bit Description | | | | | |

- **Bit 7..5 – Reserved Bits**

These bits are reserved for future use.

- **Bit 4 – JTRF: JTAG Reset Flag**

This bit is set if a reset is being caused by a logic one in the JTAG Reset Register selected by the JTAG instruction AVR_RESET. This bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

- **Bit 3 – WDRF: Watchdog Reset Flag**

This bit is set if a Watchdog Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

- **Bit 2 – BORF: Brown-out Reset Flag**

This bit is set if a Brown-out Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

- **Bit 1 – EXTRF: External Reset Flag**

This bit is set if an External Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

- **Bit 0 – PORF: Power-on Reset Flag**

This bit is set if a Power-on Reset occurs. The bit is reset only by writing a logic zero to the flag.

To make use of the Reset flags to identify a reset condition, the user should read and then reset the MCUSR as early as possible in the program. If the register is cleared before another reset occurs, the source of the reset can be found by examining the reset flags.

## Internal Voltage Reference

AT90CAN128 features an internal bandgap reference. This reference is used for Brown-out Detection, and it can be used as an input to the Analog Comparator or the ADC.

### Voltage Reference Enable Signals and Start-up Time

The voltage reference has a start-up time that may influence the way it should be used. The start-up time is given in Table 22. To save power, the reference is not always turned on. The reference is on during the following situations:

1. When the BOD is enabled (by programming the BODLEVEL [2..0] Fuse).
2. When the bandgap reference is connected to the Analog Comparator (by setting the ACBG bit in ACSR).
3. When the ADC is enabled.

Thus, when the BOD is not enabled, after setting the ACBG bit or enabling the ADC, the user must always allow the reference to start up before the output from the Analog Comparator or ADC is used. To reduce power consumption in Power-down mode, the user can avoid the three conditions above to ensure that the reference is turned off before entering Power-down mode.

### Voltage Reference Characteristics

**Table 22.** Internal Voltage Reference Characteristics

| Symbol | Parameter | Condition | Min. | Typ. | Max. | Units |
|---|---|---|---|---|---|---|
| $V_{BG}$ | Bandgap reference voltage | | 1.0 | 1.1 | 1.2 | V |
| $t_{BG}$ | Bandgap reference start-up time | | | 40 | 70 | µs |
| $I_{BG}$ | Bandgap reference current consumption | | | 15 | | µA |

## Watchdog Timer

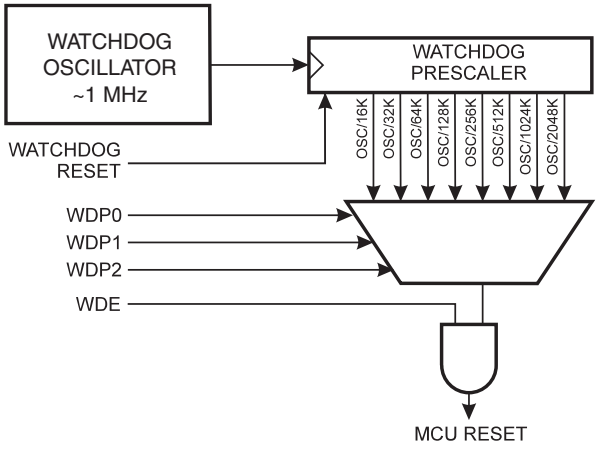
The Watchdog Timer is clocked from a separate On-chip Oscillator which runs at 1 MHz. This is the typical value at $V_{CC}$ = 5V. See characterization data for typical values at other $V_{CC}$ levels. By controlling the Watchdog Timer prescaler, the Watchdog Reset interval can be adjusted as shown in Table 24 on page 54. The WDR – Watchdog Reset – instruction resets the Watchdog Timer. The Watchdog Timer is also reset when it is disabled and when a Chip Reset occurs. Eight different clock cycle periods can be selected to determine the reset period. If the reset period expires without another Watchdog Reset, the AT90CAN128 resets and executes from the Reset Vector. For timing details on the Watchdog Reset, refer to Table 24 on page 54.

To prevent unintentional disabling of the Watchdog or unintentional change of time-out period, two different safety levels are selected by the fuse WDTON as shown in Table 23. Refer to “Timed Sequences for Changing the Configuration of the Watchdog Timer” on page 55 for details.

**Table 23.** WDT Configuration as a Function of the Fuse Settings of WDTON

| WDTON | Safety Level | WDT Initial State | How to Disable the WDT | How to Change Time-out |
|---|---|---|---|---|
| Unprogrammed | 1 | Disabled | Timed sequence | Timed sequence |
| Programmed | 2 | Enabled | Always enabled | Timed sequence |

**Figure 29.** Watchdog Timer



### Watchdog Timer Control Register – WDTCR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | WDCE | WDE | WDP2 | WDP1 | WDP0 | WDTCR |
| Read/Write | R | R | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bits 7..5 – Reserved Bits**

These bits are reserved bits for future use.

- **Bit 4 – WDCE: Watchdog Change Enable**

This bit must be set when the WDE bit is written to logic zero. Otherwise, the Watchdog will not be disabled. Once written to one, hardware will clear this bit after four clock cycles. Refer to the description of the WDE bit for a Watchdog disable procedure. This bit must also be set when changing the prescaler bits. See “Timed Sequences for Changing the Configuration of the Watchdog Timer” on page 55

- **Bit 3 – WDE: Watchdog Enable**

When the WDE is written to logic one, the Watchdog Timer is enabled, and if the WDE is written to logic zero, the Watchdog Timer function is disabled. WDE can only be cleared if the WDCE bit has logic level one. To disable an enabled Watchdog Timer, the following procedure must be followed:

1. In the same operation, write a logic one to WDCE and WDE. A logic one must be written to WDE even though it is set to one before the disable operation starts.
2. Within the next four clock cycles, write a logic 0 to WDE. This disables the Watchdog.

In safety level 2, it is not possible to disable the Watchdog Timer, even with the algorithm described above. See “Timed Sequences for Changing the Configuration of the Watchdog Timer” on page 55

- **Bits 2..0 – WDP2, WDP1, WDP0: Watchdog Timer Prescaler 2, 1, and 0**

The WDP2, WDP1, and WDP0 bits determine the Watchdog Timer prescaling when the Watchdog Timer is enabled. The different prescaling values and their corresponding Timeout Periods are shown in Table 24.

**Table 24.** Watchdog Timer Prescale Select

| WDP2 | WDP1 | WDP0 | Number of WDT Oscillator Cycles | Typical Time-out at $V_{CC}$ = 3.0V | Typical Time-out at $V_{CC}$ = 5.0V |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 16K cycles | 17.1 ms | 16.3 ms |
| 0 | 0 | 1 | 32K cycles | 34.3 ms | 32.5 ms |
| 0 | 1 | 0 | 64K cycles | 68.5 ms | 65 ms |
| 0 | 1 | 1 | 128K cycles | 0.14 s | 0.13 s |
| 1 | 0 | 0 | 256K cycles | 0.27 s | 0.26 s |
| 1 | 0 | 1 | 512K cycles | 0.55 s | 0.52 s |
| 1 | 1 | 0 | 1,024K cycles | 1.1 s | 1.0 s |
| 1 | 1 | 1 | 2,048K cycles | 2.2 s | 2.1 s |

The following code example shows one assembly and one C function for turning off the WDT. The example assumes that interrupts are controlled (e.g. by disabling interrupts globally) so that no interrupts will occur during execution of these functions.

Assembly Code Example(1)

```
WDT_off:
  ; Write logical one to WDCE and WDE
  ldi  r16, (1<<WDCE)|(1<<WDE)
  sts  WDTCR, r16
  ; Turn off WDT
  ldi  r16, (0<<WDE)
  sts  WDTCR, r16
  ret
```

C Code Example(1)

```
void WDT_off(void)
{
  /* Write logical one to WDCE and WDE */
  WDTCR = (1<<WDCE) | (1<<WDE);
  /* Turn off WDT */
  WDTCR = 0x00;
}
```

Note: 1. The example code assumes that the part specific header file is included.

## Timed Sequences for Changing the Configuration of the Watchdog Timer

The sequence for changing configuration differs slightly between the two safety levels. Separate procedures are described for each level.

### Safety Level 1

In this mode, the Watchdog Timer is initially disabled, but can be enabled by writing the WDE bit to 1 without any restriction. A timed sequence is needed when changing the Watchdog Time-out period or disabling an enabled Watchdog Timer. To disable an enabled Watchdog Timer, and/or changing the Watchdog Time-out, the following procedure must be followed:

1. In the same operation, write a logic one to WDCE and WDE. A logic one must be written to WDE regardless of the previous value of the WDE bit.
2. Within the next four clock cycles, in the same operation, write the WDE and WDP bits as desired, but with the WDCE bit cleared.

### Safety Level 2

In this mode, the Watchdog Timer is always enabled, and the WDE bit will always read as one. A timed sequence is needed when changing the Watchdog Time-out period. To change the Watchdog Time-out, the following procedure must be followed:

1. In the same operation, write a logical one to WDCE and WDE. Even though the WDE always is set, the WDE must be written to one to start the timed sequence.
2. Within the next four clock cycles, in the same operation, write the WDP bits as desired, but with the WDCE bit cleared. The value written to the WDE bit is irrelevant.

# Interrupts

This section describes the specifics of the interrupt handling as performed in AT90CAN128. For a general explanation of the AVR interrupt handling, refer to “Reset and Interrupt Handling” on page 13.

## Interrupt Vectors in AT90CAN128

**Table 25.** Reset and Interrupt Vectors

| Vector No. | Program Address(2) | Source | Interrupt Definition |
|---|---|---|---|
| 1 | 0x0000(1) | RESET | External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset |
| 2 | 0x0002 | INT0 | External Interrupt Request 0 |
| 3 | 0x0004 | INT1 | External Interrupt Request 1 |
| 4 | 0x0006 | INT2 | External Interrupt Request 2 |
| 5 | 0x0008 | INT3 | External Interrupt Request 3 |
| 6 | 0x000A | INT4 | External Interrupt Request 4 |
| 7 | 0x000C | INT5 | External Interrupt Request 5 |
| 8 | 0x000E | INT6 | External Interrupt Request 6 |
| 9 | 0x0010 | INT7 | External Interrupt Request 7 |
| 10 | 0x0012 | TIMER2 COMP | Timer/Counter2 Compare Match |
| 11 | 0x0014 | TIMER2 OVF | Timer/Counter2 Overflow |
| 12 | 0x0016 | TIMER1 CAPT | Timer/Counter1 Capture Event |
| 13 | 0x0018 | TIMER1 COMPA | Timer/Counter1 Compare Match A |
| 14 | 0x001A | TIMER1 COMPB | Timer/Counter1 Compare Match B |
| 15 | 0x001C | TIMER1 COMPC | Timer/Counter1 Compare Match C |
| 16 | 0x001E | TIMER1 OVF | Timer/Counter1 Overflow |
| 17 | 0x0020 | TIMER0 COMP | Timer/Counter0 Compare Match |
| 18 | 0x0022 | TIMER0 OVF | Timer/Counter0 Overflow |
| 19 | 0x0024 | CANIT | CAN Transfer Complete or Error |
| 20 | 0x0026 | OVRIT | CAN Timer Overrun |
| 21 | 0x0028 | SPI, STC | SPI Serial Transfer Complete |
| 22 | 0x002A | USART0, RX | USART0, Rx Complete |
| 23 | 0x002C | USART0, UDRE | USART0 Data Register Empty |
| 24 | 0x002E | USART0, TX | USART0, Tx Complete |
| 25 | 0x0030 | ANALOG COMP | Analog Comparator |
| 26 | 0x0032 | ADC | ADC Conversion Complete |
| 27 | 0x0034 | EE READY | EEPROM Ready |
| 28 | 0x0036 | TIMER3 CAPT | Timer/Counter3 Capture Event |
| 29 | 0x0038 | TIMER3 COMPA | Timer/Counter3 Compare Match A |

**Table 25.** Reset and Interrupt Vectors (Continued)

| Vector No. | Program Address(2) | Source | Interrupt Definition |
|---|---|---|---|
| 30 | 0x003A | TIMER3 COMPB | Timer/Counter3 Compare Match B |
| 31 | 0x003C | TIMER3 COMPC | Timer/Counter3 Compare Match C |
| 32 | 0x003E | TIMER3 OVF | Timer/Counter3 Overflow |
| 33 | 0x0040 | USART1, RX | USART1, Rx Complete |
| 34 | 0x0042 | USART1, UDRE | USART1 Data Register Empty |
| 35 | 0x0044 | USART1, TX | USART1, Tx Complete |
| 36 | 0x0046 | TWI | Two-wire Serial Interface |
| 37 | 0x0048 | SPM READY | Store Program Memory Ready |

Notes: 1. When the BOOTRST Fuse is programmed, the device will jump to the Boot Loader address at reset, see “Boot Loader Support – Read-While-Write Self-Programming” on page 311.
2. When the IVSEL bit in MCUCR is set, Interrupt Vectors will be moved to the start of the Boot Flash Section. The address of each Interrupt Vector will then be the address in this table added to the start address of the Boot Flash Section.

Table 26 shows reset and Interrupt Vectors placement for the various combinations of BOOTRST and IVSEL settings. If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the Boot section or vice versa.

**Table 26.** Reset and Interrupt Vectors Placement(Note:)

| BOOTRST | IVSEL | Reset Address | Interrupt Vectors Start Address |
|---|---|---|---|
| 1 | 0 | 0x0000 | 0x0002 |
| 1 | 1 | 0x0000 | Boot Reset Address + 0x0002 |
| 0 | 0 | Boot Reset Address | 0x0002 |
| 0 | 1 | Boot Reset Address | Boot Reset Address + 0x0002 |

Note: The Boot Reset Address is shown in Table 119 on page 323. For the BOOTRST Fuse “1” means unprogrammed while “0” means programmed.

The most typical and general program setup for the Reset and Interrupt Vector Addresses in AT90CAN128 is:

```
;AddressLabels Code             Comments
0x0000         jmp     RESET    ; Reset Handler
0x0002         jmp     EXT_INT0 ; IRQ0 Handler
0x0004         jmp     EXT_INT1 ; IRQ1 Handler
0x0006         jmp     EXT_INT2 ; IRQ2 Handler
0x0008         jmp     EXT_INT3 ; IRQ3 Handler
0x000A         jmp     EXT_INT4 ; IRQ4 Handler
0x000C         jmp     EXT_INT5 ; IRQ5 Handler
0x000E         jmp     EXT_INT6 ; IRQ6 Handler
0x0010         jmp     EXT_INT7 ; IRQ7 Handler
0x0012         jmp     TIM2_COMP ; Timer2 Compare Handler
0x0014         jmp     TIM2_OVF  ; Timer2 Overflow Handler
```

```
0x0016          jmp     TIM1_CAPT  ; Timer1 Capture Handler
0x0018          jmp     TIM1_COMPA ; Timer1 CompareA Handler
0x001A          jmp     TIM1_COMPB ; Timer1 CompareB Handler
0x001C          jmp     TIM1_OVF   ; Timer1 CompareC Handler
0x001E          jmp     TIM1_OVF   ; Timer1 Overflow Handler
0x0020          jmp     TIM0_COMP  ; Timer0 Compare Handler
0x0022          jmp     TIM0_OVF   ; Timer0 Overflow Handler
0x0024          jmp     CAN_IT     ; CAN Handler
0x0026          jmp     CTIM_OVF   ; CAN Timer Overflow Handler
0x0028          jmp     SPI_STC    ; SPI Transfer Complete Handler
0x002A          jmp     USART0_RXC ; USART0 RX Complete Handler
0x002C          jmp     USART0_DRE ; USART0,UDR Empty Handler
0x002E          jmp     USART0_TXC ; USART0 TX Complete Handler
0x0030          jmp     ANA_COMP   ; Analog Comparator Handler
0x0032          jmp     ADC        ; ADC Conversion Complete Handler
0x0034          jmp     EE_RDY     ; EEPROM Ready Handler
0x0036          jmp     TIM3_CAPT  ; Timer3 Capture Handler
0x0038          jmp     TIM3_COMPA ; Timer3 CompareA Handler
0x003A          jmp     TIM3_COMPB ; Timer3 CompareB Handler
0x003C          jmp     TIM3_COMPC ; Timer3 CompareC Handler
0x003E          jmp     TIM3_OVF   ; Timer3 Overflow Handler
0x0040          jmp     USART1_RXC ; USART1 RX Complete Handler
0x0042          jmp     USART1_DRE ; USART1,UDR Empty Handler
0x0044          jmp     USART1_TXC ; USART1 TX Complete Handler
0x0046          jmp     TWI        ; TWI Interrupt Handler
0x0048          jmp     SPM_RDY    ; SPM Ready Handler
;
0x0049   RESET: ldi     r16, high(RAMEND); Main program start
0x004A          out     SPH,r16          ;Set Stack Pointer to top of RAM
0x004B          ldi     r16, low(RAMEND)
0x004C          out     SPL,r16
0x004D          sei                      ; Enable interrupts
0x004E          <instr>  xxx
  ...      ...     ...    ...
```

When the BOOTRST Fuse is unprogrammed, the Boot section size set to 8K bytes and the IVSEL bit in the MCUCR Register is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
;Address  Labels  Code                    Comments
0x0000   RESET:  ldi     r16,high(RAMEND) ; Main program start
0x0001           out     SPH,r16          ; Set Stack Pointer to top of RAM
0x0002           ldi     r16,low(RAMEND)
0x0003           out     SPL,r16
0x0004           sei                      ; Enable interrupts
0x0005           <instr>  xxx
;
.org 0xF002
0xF002           jmp     EXT_INT0         ; IRQ0 Handler
0xF004           jmp     PCINT0           ; PCINT0 Handler
```

```
...             ...     ...             ;
0xF00C          jmp     SPM_RDY         ; Store Program Memory Ready Handler
```

When the BOOTRST Fuse is programmed and the Boot section size set to 8K bytes, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
;Address  Labels  Code                  Comments
.org 0x0002
0x0002           jmp     EXT_INT0       ; IRQ0 Handler
0x0004           jmp     PCINT0         ; PCINT0 Handler
...              ...     ...            ;
0x002C           jmp     SPM_RDY        ; Store Program Memory Ready Handler
;
.org 0xF000
0xF000   RESET:  ldi     r16,high(RAMEND) ; Main program start
0xF001           out     SPH,r16        ; Set Stack Pointer to top of RAM
0xF002           ldi     r16,low(RAMEND)
0xF003           out     SPL,r16
0xF004           sei                    ; Enable interrupts
0xF005           <instr>  xxx
```

When the BOOTRST Fuse is programmed, the Boot section size set to 8K bytes and the IVSEL bit in the MCUCR Register is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
;Address  Labels  Code                  Comments
;
.org 0xF000
0xF000           jmp     RESET          ; Reset handler
0xF002           jmp     EXT_INT0       ; IRQ0 Handler
0xF004           jmp     PCINT0         ; PCINT0 Handler
...              ...     ...            ;
0xF044           jmp     SPM_RDY        ; Store Program Memory Ready Handler
;
0xF046   RESET:  ldi     r16,high(RAMEND) ; Main program start
0xF047           out     SPH,r16        ; Set Stack Pointer to top of RAM
0xF048           ldi     r16,low(RAMEND)
0xF049           out     SPL,r16
0xF04A           sei                    ; Enable interrupts
0xF04B           <instr>  xxx
```

## Moving Interrupts Between Application and Boot Space

The General Interrupt Control Register controls the placement of the Interrupt Vector table.

### MCU Control Register – MCUCR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | JTD | – | – | PUD | – | – | IVSEL | IVCE | MCUCR |
| Read/Write | R/W | R | R | R/W | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 1 – IVSEL: Interrupt Vector Select**

When the IVSEL bit is cleared (zero), the Interrupt Vectors are placed at the start of the Flash memory. When this bit is set (one), the Interrupt Vectors are moved to the beginning of the Boot Loader section of the Flash. The actual address of the start of the Boot Flash Section is determined by the BOOTSZ Fuses. Refer to the section “Boot Loader Support – Read-While-Write Self-Programming” on page 311 for details. To avoid unintentional changes of Interrupt Vector tables, a special write procedure must be followed to change the IVSEL bit:

1. Write the Interrupt Vector Change Enable (IVCE) bit to one.
2. Within four cycles, write the desired value to IVSEL while writing a zero to IVCE.

Interrupts will automatically be disabled while this sequence is executed. Interrupts are disabled in the cycle IVCE is set, and they remain disabled until after the instruction following the write to IVSEL. If IVSEL is not written, interrupts remain disabled for four cycles. The I-bit in the Status Register is unaffected by the automatic disabling.

Note: If Interrupt Vectors are placed in the Boot Loader section and Boot Lock bit BLB02 is programmed, interrupts are disabled while executing from the Application section. If Interrupt Vectors are placed in the Application section and Boot Lock bit BLB12 is programed, interrupts are disabled while executing from the Boot Loader section. Refer to the section “Boot Loader Support – Read-While-Write Self-Programming” on page 311 for details on Boot Lock bits.

- **Bit 0 – IVCE: Interrupt Vector Change Enable**

The IVCE bit must be written to logic one to enable change of the IVSEL bit. IVCE is cleared by hardware four cycles after it is written or when IVSEL is written. Setting the IVCE bit will disable interrupts, as explained in the IVSEL description above. See Code Example below.

Assembly Code Example

```
Move_interrupts:
  ; Enable change of Interrupt Vectors
  ldi  r16, (1<<IVCE)
  out  MCUCR, r16
  ; Move interrupts to Boot Flash section
  ldi  r16, (1<<IVSEL)
  out  MCUCR, r16
  ret
```

C Code Example

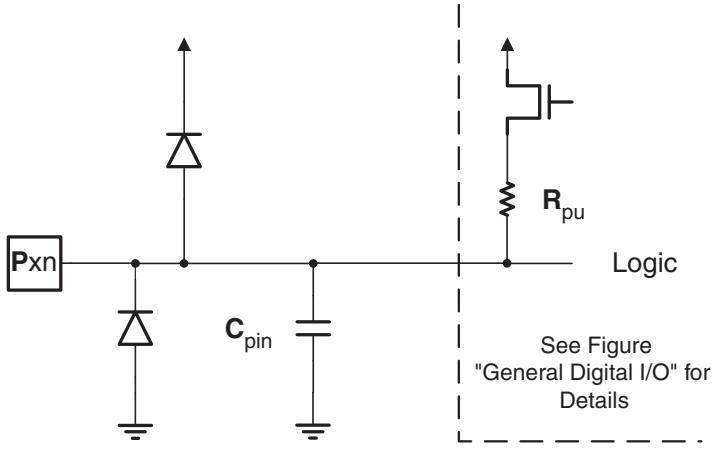
```
void Move_interrupts(void)
{
  /* Enable change of Interrupt Vectors */
  MCUCR = (1<<IVCE);
  /* Move interrupts to Boot Flash section */
  MCUCR = (1<<IVSEL);
}
```

# I/O-Ports

## Introduction

All AVR ports have true Read-Modify-Write functionality when used as general digital I/O ports. This means that the direction of one port pin can be changed without unintentionally changing the direction of any other pin with the SBI and CBI instructions. The same applies when changing drive value (if configured as output) or enabling/disabling of pull-up resistors (if configured as input). Each output buffer has symmetrical drive characteristics with both high sink and source capability. All port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance. All I/O pins have protection diodes to both $V_{CC}$ and Ground as indicated in Figure 30. Refer to “Electrical Characteristics(1)” on page 355 for a complete list of parameters.

**Figure 30.** I/O Pin Equivalent Schematic




All registers and bit references in this section are written in general form. A lower case “x” represents the numbering letter for the port, and a lower case “n” represents the bit number. However, when using the register or bit defines in a program, the precise form must be used. For example, PORTB3 for bit no. 3 in Port B, here documented generally as PORTxn. The physical I/O Registers and bit locations are listed in “Register Description for I/O-Ports”.

Three I/O memory address locations are allocated for each port, one each for the Data Register – PORTx, Data Direction Register – DDRx, and the Port Input Pins – PINx. The Port Input Pins I/O location is read only, while the Data Register and the Data Direction Register are read/write. However, writing a logic one to a bit in the PINx Register, will result in a toggle in the corresponding bit in the Data Register. In addition, the Pull-up Disable – PUD bit in MCUCR disables the pull-up function for all pins in all ports when set.

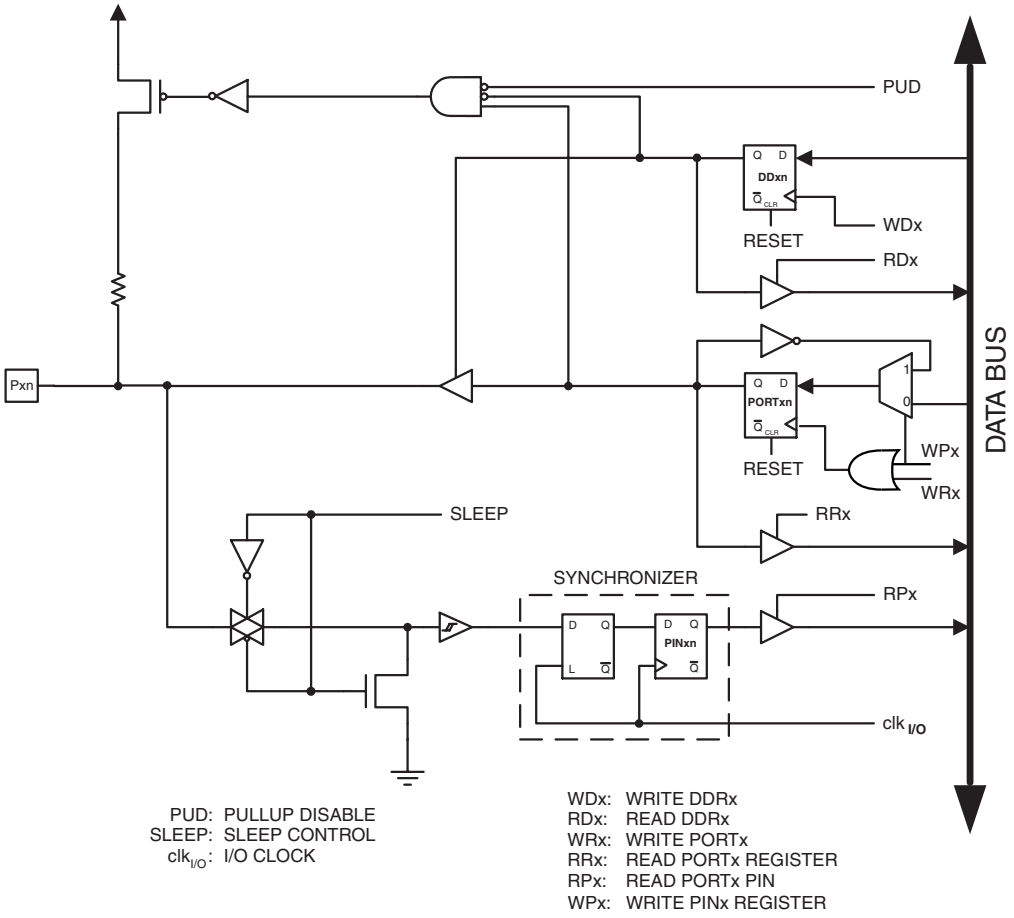
Using the I/O port as General Digital I/O is described in “Ports as General Digital I/O”. Most port pins are multiplexed with alternate functions for the peripheral features on the device. How each alternate function interferes with the port pin is described in “Alternate Port Functions” on page 67. Refer to the individual module sections for a full description of the alternate functions.

Note that enabling the alternate function of some of the port pins does not affect the use of the other pins in the port as general digital I/O.

## Ports as General Digital I/O

The ports are bi-directional I/O ports with optional internal pull-ups. Figure 31 shows a functional description of one I/O-port pin, here generically called Pxn.

**Figure 31.** General Digital I/O(1)




Note: 1. WRx, WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

### Configuring the Pin

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. As shown in "Register Description for I/O-Ports" on page 85, the DDxn bits are accessed at the DDRx I/O address, the PORTxn bits at the PORTx I/O address, and the PINxn bits at the PINx I/O address.

The DDxn bit in the DDRx Register selects the direction of this pin. If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.

If PORTxn is written logic one when the pin is configured as an input pin, the pull-up resistor is activated. To switch the pull-up resistor off, PORTxn has to be written logic zero or the pin has to be configured as an output pin

The port pins are tri-stated when reset condition becomes active, even if no clocks are running.

If PORTxn is written logic one when the pin is configured as an output pin, the port pin is driven high (one). If PORTxn is written logic zero when the pin is configured as an output pin, the port pin is driven low (zero).

**Toggling the Pin**

Writing a logic one to PINxn toggles the value of PORTxn, independent on the value of DDRxn. Note that the SBI instruction can be used to toggle one single bit in a port.

**Switching Between Input and Output**

When switching between tri-state ({DDxn, PORTxn} = 0b00) and output high ({DDxn, PORTxn} = 0b11), an intermediate state with either pull-up enabled {DDxn, PORTxn} = 0b01) or output low ({DDxn, PORTxn} = 0b10) must occur. Normally, the pull-up enabled state is fully acceptable, as a high-impedant environment will not notice the difference between a strong high driver and a pull-up. If this is not the case, the PUD bit in the MCUCR Register can be set to disable all pull-ups in all ports.

Switching between input with pull-up and output low generates the same problem. The user must use either the tri-state ({DDxn, PORTxn} = 0b00) or the output high state ({DDxn, PORTxn} = 0b11) as an intermediate step.

Table 27 summarizes the control signals for the pin value.

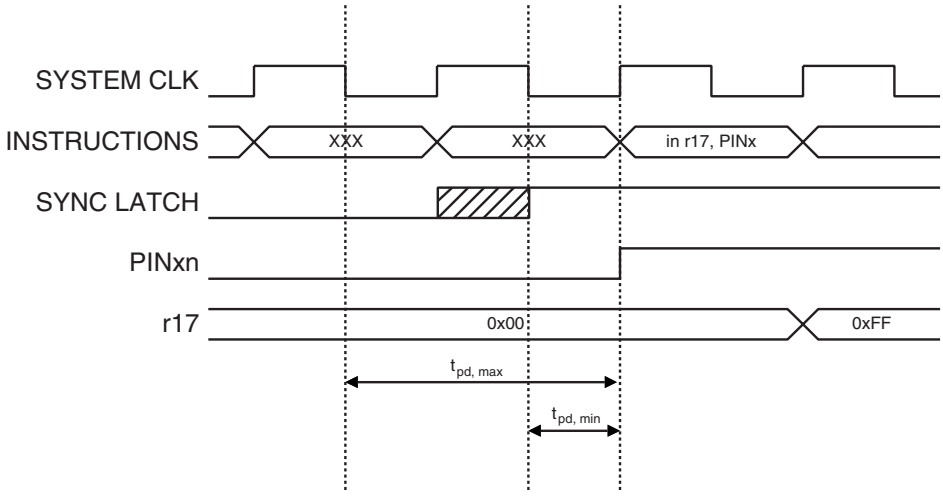
**Table 27.** Port Pin Configurations

| DDxn | PORTxn | PUD (in MCUCR) | I/O | Pull-up | Comment |
|---|---|---|---|---|---|
| 0 | 0 | X | Input | No | Default configuration after Reset. Tri-state (Hi-Z) |
| 0 | 1 | 0 | Input | Yes | Pxn will source current if ext. pulled low. |
| 0 | 1 | 1 | Input | No | Tri-state (Hi-Z) |
| 1 | 0 | X | Output | No | Output Low (Sink) |
| 1 | 1 | X | Output | No | Output High (Source) |

**Reading the Pin Value**

Independent of the setting of Data Direction bit DDxn, the port pin can be read through the PINxn Register bit. As shown in Figure 31, the PINxn Register bit and the preceding latch constitute a synchronizer. This is needed to avoid metastability if the physical pin changes value near the edge of the internal clock, but it also introduces a delay. Figure 32 shows a timing diagram of the synchronization when reading an externally applied pin value. The maximum and minimum propagation delays are denoted $t_{pd,max}$ and $t_{pd,min}$ respectively.

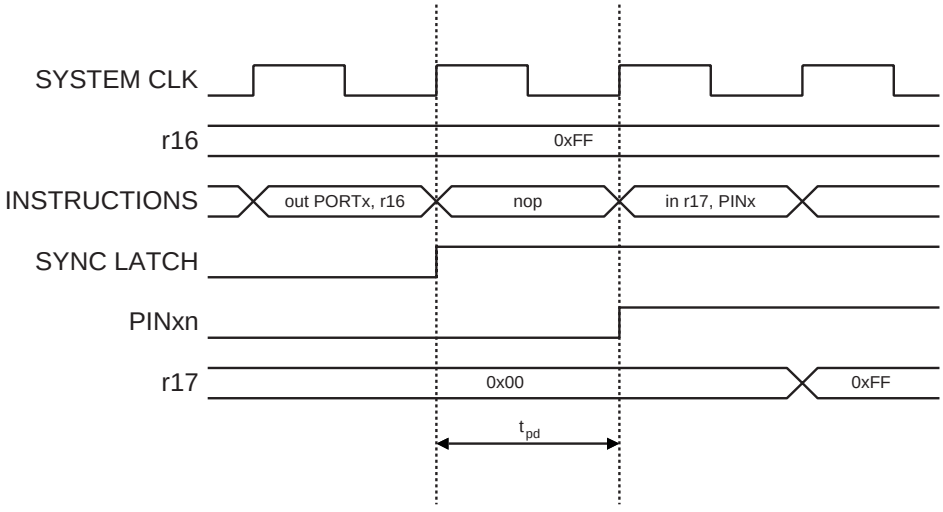
**Figure 32.** Synchronization when Reading an Externally Applied Pin value




Consider the clock period starting shortly after the first falling edge of the system clock. The latch is closed when the clock is low, and goes transparent when the clock is high, as indicated by the shaded region of the “SYNC LATCH” signal. The signal value is latched when the system clock goes low. It is clocked into the PINxn Register at the succeeding positive clock edge. As indicated by the two arrows $t_{pd,max}$ and $t_{pd,min}$, a single signal transition on the pin will be delayed between ½ and 1½ system clock period depending upon the time of assertion.

When reading back a software assigned pin value, a nop instruction must be inserted as indicated in Figure 33. The out instruction sets the “SYNC LATCH” signal at the positive edge of the clock. In this case, the delay $t_{pd}$ through the synchronizer is 1 system clock period.

**Figure 33.** Synchronization when Reading a Software Assigned Pin Value



The following code example shows how to set port B pins 0 and 1 high, 2 and 3 low, and define the port pins from 4 to 7 as input with pull-ups assigned to port pins 6 and 7. The resulting pin values are read back again, but as previously discussed, a nop instruction is included to be able to read back the value recently assigned to some of the pins.

Assembly Code Example[1]

```
...
; Define pull-ups and set outputs high
; Define directions for port pins
ldi   r16, (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0)
ldi   r17, (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0)
out   PORTB, r16
out   DDRB, r17
; Insert nop for synchronization
nop
; Read port pins
in    r16, PINB
...
```

C Code Example

```
unsigned char i;
...
/* Define pull-ups and set outputs high */
/* Define directions for port pins */
PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);
DDRB = (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0);
/* Insert nop for synchronization*/
_NOP();
/* Read port pins */
i = PINB;
...
```

Note: 1. For the assembly program, two temporary registers are used to minimize the time from pull-ups are set on pins 0, 1, 6, and 7, until the direction bits are correctly set, defining bit 2 and 3 as low and redefining bits 0 and 1 as strong high drivers.

### Digital Input Enable and Sleep Modes

As shown in Figure 31, the digital input signal can be clamped to ground at the input of the schmitt-trigger. The signal denoted SLEEP in the figure, is set by the MCU Sleep Controller in Power-down mode, Power-save mode, and Standby mode to avoid high power consumption if some input signals are left floating, or have an analog signal level close to $V_{CC}/2$.

SLEEP is overridden for port pins enabled as external interrupt pins. If the external interrupt request is not enabled, SLEEP is active also for these pins. SLEEP is also overridden by various other alternate functions as described in “Alternate Port Functions” on page 67.

If a logic high level (“one”) is present on an Asynchronous External Interrupt pin configured as “Interrupt on Rising Edge, Falling Edge, or Any Logic Change on Pin” while the external interrupt is not enabled, the corresponding External Interrupt Flag will be set when resuming from the above mentioned sleep modes, as the clamping in these sleep modes produces the requested logic change.

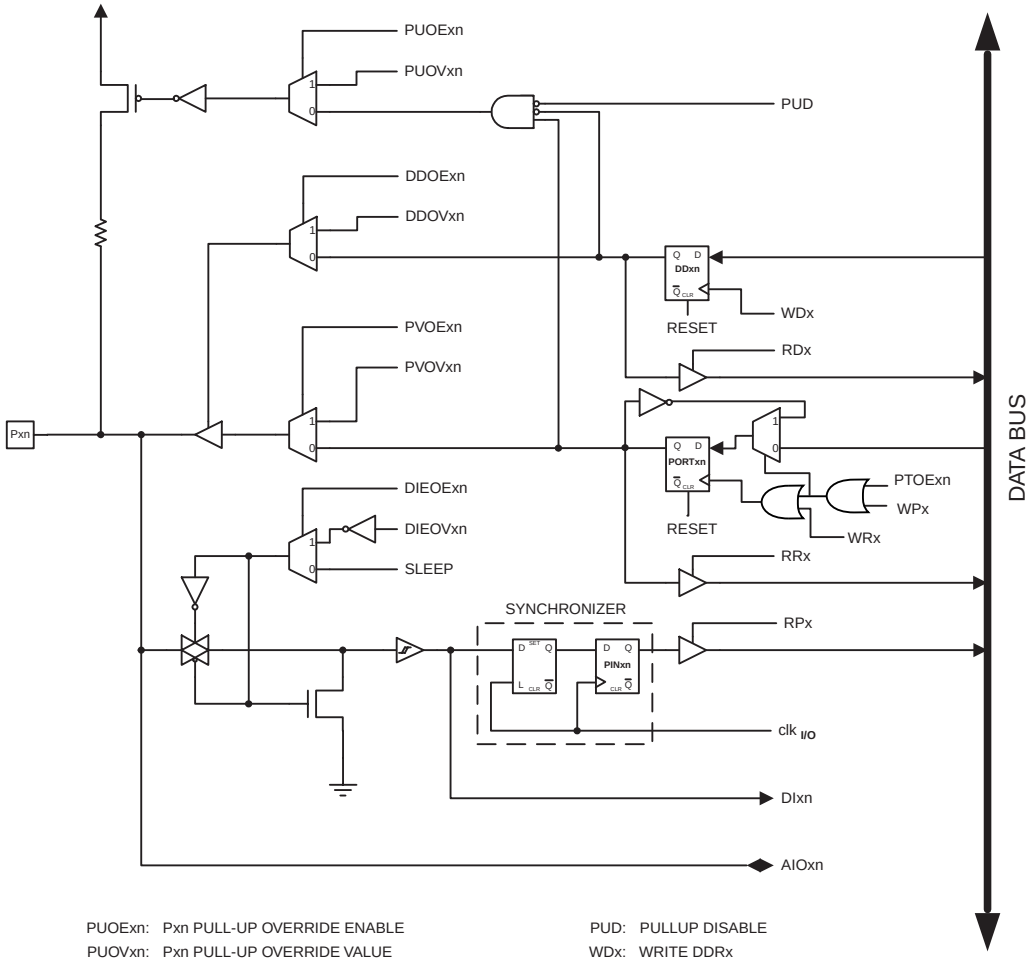
**Unconnected Pins**

If some pins are unused, it is recommended to ensure that these pins have a defined level. Even though most of the digital inputs are disabled in the deep sleep modes as described above, floating inputs should be avoided to reduce current consumption in all other modes where the digital inputs are enabled (Reset, Active mode and Idle mode). The simplest method to ensure a defined level of an unused pin, is to enable the internal pull-up. In this case, the pull-up will be disabled during reset. If low power consumption during reset is important, it is recommended to use an external pull-up or pull-down. Connecting unused pins directly to $V_{CC}$ or GND is not recommended, since this may cause excessive currents if the pin is accidentally configured as an output.

## Alternate Port Functions

Most port pins have alternate functions in addition to being general digital I/Os. Figure 34 shows how the port pin control signals from the simplified Figure 31 can be overridden by alternate functions. The overriding signals may not be present in all port pins, but the figure serves as a generic description applicable to all port pins in the AVR microcontroller family.

**Figure 34.** Alternate Port Functions(1)




| | | | |
|---|---|---|---|
| PUOExn: | Pxn PULL-UP OVERRIDE ENABLE | PUD: | PULLUP DISABLE |
| PUOVxn: | Pxn PULL-UP OVERRIDE VALUE | WDx: | WRITE DDRx |
| DDOExn: | Pxn DATA DIRECTION OVERRIDE ENABLE | RDx: | READ DDRx |
| DDOVxn: | Pxn DATA DIRECTION OVERRIDE VALUE | RRx: | READ PORTx REGISTER |
| PVOExn: | Pxn PORT VALUE OVERRIDE ENABLE | WRx: | WRITE PORTx |
| PVOVxn: | Pxn PORT VALUE OVERRIDE VALUE | RPx: | READ PORTx PIN |
| DIEOExn: | Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE | WPx: | WRITE PINx |
| DIEOVxn: | Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE | $clk_{I/O}$: | I/O CLOCK |
| SLEEP: | SLEEP CONTROL | DIxn: | DIGITAL INPUT PIN n ON PORTx |
| PTOExn: | Pxn, PORT TOGGLE OVERRIDE ENABLE | AIOxn: | ANALOG INPUT/OUTPUT PIN n ON PORTx |

Note: 1. WRx, WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports. All other signals are unique for each pin.

Table 28 summarizes the function of the overriding signals. The pin and port indexes from Figure 34 are not shown in the succeeding tables. The overriding signals are generated internally in the modules having the alternate function.

**Table 28.** Generic Description of Overriding Signals for Alternate Functions

| Signal Name | Full Name | Description |
|---|---|---|
| PUOE | Pull-up Override Enable | If this signal is set, the pull-up enable is controlled by the PUOV signal. If this signal is cleared, the pull-up is enabled when {DDxn, PORTxn, PUD} = 0b010. |
| PUOV | Pull-up Override Value | If PUOE is set, the pull-up is enabled/disabled when PUOV is set/cleared, regardless of the setting of the DDxn, PORTxn, and PUD Register bits. |
| DDOE | Data Direction Override Enable | If this signal is set, the Output Driver Enable is controlled by the DDOV signal. If this signal is cleared, the Output driver is enabled by the DDxn Register bit. |
| DDOV | Data Direction Override Value | If DDOE is set, the Output Driver is enabled/disabled when DDOV is set/cleared, regardless of the setting of the DDxn Register bit. |
| PVOE | Port Value Override Enable | If this signal is set and the Output Driver is enabled, the port value is controlled by the PVOV signal. If PVOE is cleared, and the Output Driver is enabled, the port Value is controlled by the PORTxn Register bit. |
| PVOV | Port Value Override Value | If PVOE is set, the port value is set to PVOV, regardless of the setting of the PORTxn Register bit. |
| PTOE | Port Toggle Override Enable | If PTOE is set, the PORTxn Register bit is inverted. |
| DIEOE | Digital Input Enable Override Enable | If this bit is set, the Digital Input Enable is controlled by the DIEOV signal. If this signal is cleared, the Digital Input Enable is determined by MCU state (Normal mode, sleep mode). |
| DIEOV | Digital Input Enable Override Value | If DIEOE is set, the Digital Input is enabled/disabled when DIEOV is set/cleared, regardless of the MCU state (Normal mode, sleep mode). |
| DI | Digital Input | This is the Digital Input to alternate functions. In the figure, the signal is connected to the output of the schmitt trigger but before the synchronizer. Unless the Digital Input is used as a clock source, the module with the alternate function will use its own synchronizer. |
| AIO | Analog Input/Output | This is the Analog Input/output to/from alternate functions. The signal is connected directly to the pad, and can be used bi-directionally. |

The following subsections shortly describe the alternate functions for each port, and relate the overriding signals to the alternate function. Refer to the alternate function description for further details.

**MCU Control Register – MCUCR**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | JTD | – | – | PUD | – | – | IVSEL | IVCE | MCUCR |
| Read/Write | R/W | R | R | R/W | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 4 – PUD: Pull-up Disable**

When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01). See “Configuring the Pin” for more details about this feature.

**Alternate Functions of Port A**

The Port A has an alternate function as the address low byte and data lines for the External Memory Interface.

The Port A pins with alternate functions are shown in Table 29.

**Table 29.** Port A Pins Alternate Functions

| Port Pin | Alternate Function |
|---|---|
| PA7 | AD7 (External memory interface address and data bit 7) |
| PA6 | AD6 (External memory interface address and data bit 6) |
| PA5 | AD5 (External memory interface address and data bit 5) |
| PA4 | AD4 (External memory interface address and data bit 4) |
| PA3 | AD3 (External memory interface address and data bit 3) |
| PA2 | AD2 (External memory interface address and data bit 2) |
| PA1 | AD1 (External memory interface address and data bit 1) |
| PA0 | AD0 (External memory interface address and data bit 0) |

The alternate pin configuration is as follows:

- **AD7 – Port A, Bit 7**

AD7, External memory interface address 7 and Data 7.

- **AD6 – Port A, Bit 6**

AD6, External memory interface address 6 and Data 6.

- **AD5 – Port A, Bit 5**

AD5, External memory interface address 5 and Data 5.

- **AD4 – Port A, Bit 4**

AD4, External memory interface address 4 and Data 4.

- **AD3 – Port A, Bit 3**

AD3, External memory interface address 3 and Data 3.

- **AD2 – Port A, Bit 2**

AD2, External memory interface address 2 and Data 2.

- **AD1 – Port A, Bit 1**

AD1, External memory interface address 1 and Data 1.

- **AD0 – Port A, Bit 0**

AD0, External memory interface address 0 and Data 0.

Table 30 and Table 31 relates the alternate functions of Port A to the overriding signals shown in Figure 34 on page 67.

**Table 30.** Overriding Signals for Alternate Functions in PA7..PA4

| Signal Name | PA7/AD7 | PA6/AD6 | PA5/AD5 | PA4/AD4 |
|---|---|---|---|---|
| PUOE | SRE • (ADA + $\overline{WR}$) | SRE • (ADA + $\overline{WR}$) | SRE • (ADA + $\overline{WR}$) | SRE • (ADA + $\overline{WR}$) |
| PUOV | 0 | 0 | 0 | 0 |
| DDOE | SRE | SRE | SRE | SRE |
| DDOV | $\overline{WR}$ + ADA | $\overline{WR}$ + ADA | $\overline{WR}$ + ADA | $\overline{WR}$ + ADA |
| PVOE | SRE | SRE | SRE | SRE |
| PVOV | A7 • ADA + $\overline{D7}$ OUTPUT • $\overline{WR}$ | A6 • ADA + $\overline{D6}$ OUTPUT • $\overline{WR}$ | A5 • ADA + $\overline{D5}$ OUTPUT • $\overline{WR}$ | A4 • ADA + $\overline{D4}$ OUTPUT • $\overline{WR}$ |
| PTOE | 0 | 0 | 0 | 0 |
| DIEOE | 0 | 0 | 0 | 0 |
| DIEOV | 0 | 0 | 0 | 0 |
| DI | D7 INPUT | D6 INPUT | D5 INPUT | D4 INPUT |
| AIO | – | – | – | – |

Note: 1. ADA is short for ADdress Active and represents the time when address is output. See “External Memory Interface” on page 24 for details.

**Table 31.** Overriding Signals for Alternate Functions in PA3..PA0

| Signal Name | PA3/AD3 | PA2/AD2 | PA1/AD1 | PA0/AD0 |
|---|---|---|---|---|
| PUOE | SRE • (ADA + $\overline{WR}$) | SRE • (ADA + $\overline{WR}$) | SRE • (ADA + $\overline{WR}$) | SRE • (ADA + $\overline{WR}$) |
| PUOV | 0 | 0 | 0 | 0 |
| DDOE | SRE | SRE | SRE | SRE |
| DDOV | $\overline{WR}$ + ADA | $\overline{WR}$ + ADA | $\overline{WR}$ + ADA | $\overline{WR}$ + ADA |
| PVOE | SRE | SRE | SRE | SRE |
| PVOV | A3 • ADA + $\overline{D3}$ OUTPUT • $\overline{WR}$ | A2 • ADA + $\overline{D2}$ OUTPUT • $\overline{WR}$ | A1 • ADA + $\overline{D1}$ OUTPUT • $\overline{WR}$ | A0 • ADA + $\overline{D0}$ OUTPUT • $\overline{WR}$ |
| PTOE | 0 | 0 | 0 | 0 |
| DIEOE | 0 | 0 | 0 | 0 |
| DIEOV | 0 | 0 | 0 | 0 |
| DI | D3 INPUT | D2 INPUT | D1 INPUT | D0 INPUT |
| AIO | – | – | – | – |

Note: 1. ADA is short for ADdress Active and represents the time when address is output. See “External Memory Interface” on page 24 for details.

**Alternate Functions of Port B**

The Port B pins with alternate functions are shown in Table 32.

**Table 32.** Port B Pins Alternate Functions

| Port Pin | Alternate Functions |
|---|---|
| PB7 | OC0A/OC1C (Output Compare and PWM Output A for Timer/Counter0 or Output Compare and PWM Output C for Timer/Counter1) |
| PB6 | OC1B (Output Compare and PWM Output B for Timer/Counter1) |
| PB5 | OC1A (Output Compare and PWM Output A for Timer/Counter1) |
| PB4 | OC2A (Output Compare and PWM Output A for Timer/Counter2 ) |
| PB3 | MISO (SPI Bus Master Input/Slave Output) |
| PB2 | MOSI (SPI Bus Master Output/Slave Input) |
| PB1 | SCK (SPI Bus Serial Clock) |
| PB0 | $\overline{SS}$ (SPI Slave Select input) |

The alternate pin configuration is as follows:

- **OC0A/OC1C, Bit 7**

OC0A, Output Compare Match A output. The PB7 pin can serve as an external output for the Timer/Counter0 Output Compare A. The pin has to be configured as an output (DDB7 set “one”) to serve this function. The OC0A pin is also the output pin for the PWM mode timer function.

OC1C, Output Compare Match C output. The PB7 pin can serve as an external output for the Timer/Counter1 Output Compare C. The pin has to be configured as an output (DDB7 set “one”) to serve this function. The OC1C pin is also the output pin for the PWM mode timer function.

- **OC1B, Bit 6**

OC1B, Output Compare Match B output. The PB6 pin can serve as an external output for the Timer/Counter1 Output Compare B. The pin has to be configured as an output (DDB6 set “one”) to serve this function. The OC1B pin is also the output pin for the PWM mode timer function.

- **OC1A, Bit 5**

OC1A, Output Compare Match A output. The PB5 pin can serve as an external output for the Timer/Counter1 Output Compare A. The pin has to be configured as an output (DDB5 set “one”) to serve this function. The OC1A pin is also the output pin for the PWM mode timer function.

- **OC2A, Bit 4**

OC2A, Output Compare Match A output. The PB4 pin can serve as an external output for the Timer/Counter2 Output Compare A. The pin has to be configured as an output (DDB4 set “one”) to serve this function. The OC2A pin is also the output pin for the PWM mode timer function.

- **MISO – Port B, Bit 3**

MISO, Master Data input, Slave Data output pin for SPI channel. When the SPI is enabled as a master, this pin is configured as an input regardless of the setting of DDB3. When the SPI is enabled as a slave, the data direction of this pin is controlled by

DDB3. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB3 bit.

- **MOSI – Port B, Bit 2**

MOSI, SPI Master Data output, Slave Data input for SPI channel. When the SPI is enabled as a slave, this pin is configured as an input regardless of the setting of DDB2. When the SPI is enabled as a master, the data direction of this pin is controlled by DDB2. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB2 bit.

- **SCK – Port B, Bit 1**

SCK, Master Clock output, Slave Clock input pin for SPI channel. When the SPI is enabled as a slave, this pin is configured as an input regardless of the setting of DDB1. When the SPI is enabled as a master, the data direction of this pin is controlled by DDB1. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB1 bit.

- **$\overline{SS}$ – Port B, Bit 0**

$\overline{SS}$, Slave Port Select input. When the SPI is enabled as a slave, this pin is configured as an input regardless of the setting of DDB0. As a slave, the SPI is activated when this pin is driven low. When the SPI is enabled as a master, the data direction of this pin is controlled by DDB0. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB0 bit.

Table 33 and Table 34 relate the alternate functions of Port B to the overriding signals shown in Figure 34 on page 67. SPI MSTR INPUT and SPI SLAVE OUTPUT constitute the MISO signal, while MOSI is divided into SPI MSTR OUTPUT and SPI SLAVE INPUT.

Table 33 and Table 34 relates the alternate functions of Port B to the overriding signals shown in Figure 34 on page 67.

**Table 33.** Overriding Signals for Alternate Functions in PB7..PB4

| Signal Name | PB7/OC0A/OC1C | PB6/OC1B | PB5/OC1A | PB4/OC2A |
|---|---|---|---|---|
| PUOE | 0 | 0 | 0 | 0 |
| PUOV | 0 | 0 | 0 | 0 |
| DDOE | 0 | 0 | 0 | 0 |
| DDOV | 0 | 0 | 0 | 0 |
| PVOE | OC0A/OC1C ENABLE(1) | OC1B ENABLE | OC1A ENABLE | OC2A ENABLE |
| PVOV | OC0A/OC1C(1) | OC1B | OC1A | OC2A |
| PTOE | 0 | 0 | 0 | 0 |
| DIEOE | 0 | 0 | 0 | 0 |
| DIEOV | 0 | 0 | 0 | 0 |
| DI | – | – | – | – |
| AIO | – | – | – | – |

Note: 1. See “Output Compare Modulator - OCM” on page 160 for details.

**Table 34.** Overriding Signals for Alternate Functions in PB3..PB0

| Signal Name | PB3/MISO | PB2/MOSI | PB1/SCK | PB0/$\overline{SS}$ |
|---|---|---|---|---|
| PUOE | SPE • MSTR | SPE • $\overline{MSTR}$ | SPE • $\overline{MSTR}$ | SPE • $\overline{MSTR}$ |
| PUOV | PORTB3 • $\overline{PUD}$ | PORTB2 • $\overline{PUD}$ | PORTB1 • $\overline{PUD}$ | PORTB0 • $\overline{PUD}$ |
| DDOE | SPE • MSTR | SPE • $\overline{MSTR}$ | SPE • $\overline{MSTR}$ | SPE • $\overline{MSTR}$ |
| DDOV | 0 | 0 | 0 | 0 |
| PVOE | SPE • $\overline{MSTR}$ | SPE • MSTR | SPE • MSTR | 0 |
| PVOV | SPI SLAVE OUTPUT | SPI MASTER OUTPUT | SCK OUTPUT | 0 |
| PTOE | 0 | 0 | 0 | 0 |
| DIEOE | 0 | 0 | 0 | 0 |
| DIEOV | 0 | 0 | 0 | 0 |
| DI | SPI MASTER INPUT | SPI SLAVE INPUT • $\overline{RESET}$ | SCK INPUT | SPI $\overline{SS}$ |
| AIO | – | – | – | – |

**Alternate Functions of Port C**

The Port C has an alternate function as the address high byte for the External Memory Interface.

The Port C pins with alternate functions are shown in Table 35.

**Table 35.** Port C Pins Alternate Functions

| Port Pin | Alternate Function |
|---|---|
| PC7 | A15/CLKO (External memory interface address 15 or Divided System Clock) |
| PC6 | A14 (External memory interface address 14) |
| PC5 | A13 (External memory interface address 13) |
| PC4 | A12 (External memory interface address 12) |
| PC3 | A11 (External memory interface address 11) |
| PC2 | A10 (External memory interface address 10) |
| PC1 | A9 (External memory interface address 9) |
| PC0 | A8 (External memory interface address 8) |

The alternate pin configuration is as follows:

- **A15/CLKO – Port C, Bit 7**

A15, External memory interface address 15.

CLKO, Divided System Clock: The divided system clock can be output on the PC7 pin. The divided system clock will be output if the CKOUT Fuse is programmed, regardless of the PORTC7 and DDC7 settings. It will also be output during reset.

- **A14 – Port C, Bit 6**

A14, External memory interface address 14.

- **A13 – Port C, Bit 5**

A13, External memory interface address 13.

- **A12 – Port C, Bit 4**

A12, External memory interface address 12.

- **A11 – Port C, Bit 3**

A11, External memory interface address 11.

- **A10 – Port C, Bit 2**

A10, External memory interface address 10.

- **A9 – Port C, Bit 1**

A9, External memory interface address 9.

- **A8 – Port C, Bit 0**

A8, External memory interface address 8.

Table 36 and Table 37 relate the alternate functions of Port C to the overriding signals shown in Figure 34 on page 67.

**Table 36.** Overriding Signals for Alternate Functions in PC7..PC4

| Signal Name | PC7/A15 | PC6/A14 | PC5/A13 | PC4/A12 |
|---|---|---|---|---|
| PUOE | SRE • (XMM<1) | SRE • (XMM<2) | SRE • (XMM<3) | SRE • (XMM<4) |
| PUOV | 0 | 0 | 0 | 0 |
| DDOE | CKOUT(1) + (SRE • (XMM<1)) | SRE • (XMM<2) | SRE • (XMM<3) | SRE • (XMM<4) |
| DDOV | 1 | 1 | 1 | 1 |
| PVOE | CKOUT(1) + (SRE • (XMM<1)) | SRE • (XMM<2) | SRE • (XMM<3) | SRE • (XMM<4) |
| PVOV | (A15 • $\overline{\text{CKOUT}}$(1)) + (CLKO • CKOUT(1)) | A14 | A13 | A12 |
| PTOE | 0 | 0 | 0 | 0 |
| DIEOE | 0 | 0 | 0 | 0 |
| DIEOV | 0 | 0 | 0 | 0 |
| DI | – | – | – | – |
| AIO | – | – | – | – |

Note: 1. CKOUT is one if the CKOUT Fuse is programmed

**Table 37.** Overriding Signals for Alternate Functions in PC3..PC0

| Signal Name | PC3/A11 | PC2/A10 | PC1/A9 | PC0/A8 |
|---|---|---|---|---|
| PUOE | SRE • (XMM<5) | SRE • (XMM<6) | SRE • (XMM<7) | SRE • (XMM<7) |
| PUOV | 0 | 0 | 0 | 0 |
| DDOE | SRE • (XMM<5) | SRE • (XMM<6) | SRE • (XMM<7) | SRE • (XMM<7) |
| DDOV | 1 | 1 | 1 | 1 |
| PVOE | SRE • (XMM<5) | SRE • (XMM<6) | SRE • (XMM<7) | SRE • (XMM<7) |
| PVOV | A11 | A10 | A9 | A8 |
| PTOE | 0 | 0 | 0 | 0 |
| DIEOE | 0 | 0 | 0 | 0 |
| DIEOV | 0 | 0 | 0 | 0 |
| DI | – | – | – | – |
| AIO | – | – | – | – |

## Alternate Functions of Port D

The Port D pins with alternate functions are shown in Table 38.

**Table 38.** Port D Pins Alternate Functions

| Port Pin | Alternate Function |
|---|---|
| PD7 | T0 (Timer/Counter0 Clock Input) |
| PD6 | RXCAN/T1 (CAN Receive Pin or Timer/Counter1 Clock Input) |
| PD5 | TXCAN/XCK1 (CAN Transmit Pin or USART1 External Clock Input/Output) |
| PD4 | ICP1 (Timer/Counter1 Input Capture Trigger) |
| PD3 | INT3/TXD1 (External Interrupt3 Input or UART1 Transmit Pin) |
| PD2 | INT2/RXD1 (External Interrupt2 Input or UART1 Receive Pin) |
| PD1 | INT1/SDA (External Interrupt1 Input or TWI Serial DAta) |
| PD0 | INT0/SCL (External Interrupt0 Input or TWI Serial CLock) |

The alternate pin configuration is as follows:

- **T0/CLKO – Port D, Bit 7**

T0, Timer/Counter0 counter source.

- **RXCAN/T1 – Port D, Bit 6**

RXCAN, CAN Receive Data (Data input pin for the CAN). When the CAN controller is enabled this pin is configured as an input regardless of the value of DDD6. When the CAN forces this pin to be an input, the pull-up can still be controlled by the PORTD6 bit.

T1, Timer/Counter1 counter source.

- **TXCAN/XCK1 – Port D, Bit 5**

TXCAN, CAN Transmit Data (Data output pin for the CAN). When the CAN is enabled, this pin is configured as an output regardless of the value of DDD5.

XCK1, USART1 External clock. The Data Direction Register (DDD5) controls whether the clock is output (DDD5 set) or input (DDD45 cleared). The XCK1 pin is active only when the USART1 operates in Synchronous mode.

- **ICP1 – Port D, Bit 4**

ICP1, Input Capture Pin1. The PD4 pin can act as an input capture pin for Timer/Counter1.

- **INT3/TXD1 – Port D, Bit 3**

INT3, External Interrupt source 3. The PD3 pin can serve as an external interrupt source to the MCU.

TXD1, Transmit Data (Data output pin for the USART1). When the USART1 Transmitter is enabled, this pin is configured as an output regardless of the value of DDD3.

- **INT2/RXD1 – Port D, Bit 2**

INT2, External Interrupt source 2. The PD2 pin can serve as an External Interrupt source to the MCU.

RXD1, Receive Data (Data input pin for the USART1). When the USART1 receiver is enabled this pin is configured as an input regardless of the value of DDD2. When the USART forces this pin to be an input, the pull-up can still be controlled by the PORTD2 bit.

- **INT1/SDA – Port D, Bit 1**

INT1, External Interrupt source 1. The PD1 pin can serve as an external interrupt source to the MCU.

SDA, Two-wire Serial Interface Data. When the TWEN bit in TWCR is set (one) to enable the Two-wire Serial Interface, pin PD1 is disconnected from the port and becomes the Serial Data I/O pin for the Two-wire Serial Interface. In this mode, there is a spike filter on the pin to suppress spikes shorter than 50 ns on the input signal, and the pin is driven by an open drain driver with slew-rate limitation.

- **INT0/SCL – Port D, Bit 0**

INT0, External Interrupt source 0. The PD0 pin can serve as an external interrupt source to the MCU.

SCL, Two-wire Serial Interface Clock: When the TWEN bit in TWCR is set (one) to enable the Two-wire Serial Interface, pin PD0 is disconnected from the port and becomes the Serial Clock I/O pin for the Two-wire Serial Interface. In this mode, there is a spike filter on the pin to suppress spikes shorter than 50 ns on the input signal, and the pin is driven by an open drain driver with slew-rate limitation.

Table 39 and Table 40 relates the alternate functions of Port D to the overriding signals shown in Figure 34 on page 67.

**Table 39.** Overriding Signals for Alternate Functions PD7..PD4

| Signal Name | PD7/T0 | PD6/T1/RXCAN | PD5/XCK1/TXCAN | PD4/ICP1 |
|---|---|---|---|---|
| PUOE | 0 | RXCANEN | TXCANEN + | 0 |
| PUOV | 0 | PORTD6 • $\overline{PUD}$ | 0 | 0 |
| DDOE | 0 | RXCANEN | TXCANEN | 0 |
| DDOV | 0 | 0 | 1 | 0 |
| PVOE | 0 | 0 | TXCANEN + UMSEL1 | 0 |
| PVOV | 0 | 0 | (XCK1 OUTPUT • UMSEL1 • $\overline{TXCANEN}$) + (TXCAN • TXCANEN) | 0 |
| PTOE | 0 | 0 | 0 | 0 |
| DIEOE | 0 | 0 | 0 | 0 |
| DIEOV | 0 | 0 | 0 | 0 |
| DI | T0 INPUT | T1 INPUT/RXCAN | XCK1 INPUT | ICP1 INPUT |
| AIO | – | – | – | – |

**Table 40.** Overriding Signals for Alternate Functions in PD3..PD0[1]

| Signal Name | PD3/INT3/TXD1 | PD2/INT2/RXD1 | PD1/INT1/SDA | PD0/INT0/SCL |
|---|---|---|---|---|
| PUOE | TXEN1 | RXEN1 | TWEN | TWEN |
| PUOV | 0 | PORTD2 • $\overline{PUD}$ | PORTD1 • $\overline{PUD}$ | PORTD0 • $\overline{PUD}$ |
| DDOE | TXEN1 | RXEN1 | 0 | 0 |
| DDOV | 1 | 0 | 0 | 0 |
| PVOE | TXEN1 | 0 | TWEN | TWEN |
| PVOV | TXD1 | 0 | SDA_OUT | SCL_OUT |
| PTOE | 0 | 0 | 0 | 0 |
| DIEOE | INT3 ENABLE | INT2 ENABLE | INT1 ENABLE | INT0 ENABLE |
| DIEOV | INT3 ENABLE | INT2 ENABLE | INT1 ENABLE | INT0 ENABLE |
| DI | INT3 INPUT | INT2 INPUT/RXD1 | INT1 INPUT | INT0 INPUT |
| AIO | – | – | SDA INPUT | SCL INPUT |

Note: 1. When enabled, the Two-wire Serial Interface enables Slew-Rate controls on the output pins PD0 and PD1. This is not shown in this table. In addition, spike filters are connected between the AIO outputs shown in the port figure and the digital logic of the TWI module.

**Alternate Functions of Port E**

The Port E pins with alternate functions are shown in Table 41.

**Table 41.** Port E Pins Alternate Functions

| Port Pin | Alternate Function |
|---|---|
| PE7 | INT7/ICP3 (External Interrupt 7 Input or Timer/Counter3 Input Capture Trigger) |
| PE6 | INT6/ T3 (External Interrupt 6 Input or Timer/Counter3 Clock Input) |
| PE5 | INT5/OC3C (External Interrupt 5 Input or Output Compare and PWM Output C for Timer/Counter3) |
| PE4 | INT4/OC3B (External Interrupt4 Input or Output Compare and PWM Output B for Timer/Counter3) |
| PE3 | AIN1/OC3A (Analog Comparator Negative Input or Output Compare and PWM Output A for Timer/Counter3) |
| PE2 | AIN0/XCK0 (Analog Comparator Positive Input or USART0 external clock input/output) |
| PE1 | PDO/TXD0 (Programming Data Output or UART0 Transmit Pin) |
| PE0 | PDI/RXD0 (Programming Data Input or UART0 Receive Pin) |

The alternate pin configuration is as follows:

- **PCINT7/ICP3 – Port E, Bit 7**

INT7, External Interrupt source 7. The PE7 pin can serve as an external interrupt source.

ICP3, Input Capture Pin3: The PE7 pin can act as an input capture pin for Timer/Counter3.

- **INT6/T3 – Port E, Bit 6**

INT6, External Interrupt source 6. The PE6 pin can serve as an external interrupt source.

T3, Timer/Counter3 counter source.

- **INT5/OC3C – Port E, Bit 5**

INT5, External Interrupt source 5. The PE5 pin can serve as an External Interrupt source.

OC3C, Output Compare Match C output. The PE5 pin can serve as an External output for the Timer/Counter3 Output Compare C. The pin has to be configured as an output (DDE5 set “one”) to serve this function. The OC3C pin is also the output pin for the PWM mode timer function.

- **INT4/OC3B – Port E, Bit 4**

INT4, External Interrupt source 4. The PE4 pin can serve as an External Interrupt source.

OC3B, Output Compare Match B output. The PE4 pin can serve as an External output for the Timer/Counter3 Output Compare B. The pin has to be configured as an output (DDE4 set (one)) to serve this function. The OC3B pin is also the output pin for the PWM mode timer function.

- **AIN1/OC3A – Port E, Bit 3**

AIN1 – Analog Comparator Negative input. This pin is directly connected to the negative input of the Analog Comparator.

OC3A, Output Compare Match A output. The PE3 pin can serve as an External output for the Timer/Counter3 Output Compare A. The pin has to be configured as an output (DDE3 set “one”) to serve this function. The OC3A pin is also the output pin for the PWM mode timer function.

- **AIN0/XCK0 – Port E, Bit 2**

AIN0 – Analog Comparator Positive input. This pin is directly connected to the positive input of the Analog Comparator.

XCK0, USART0 External clock. The Data Direction Register (DDE2) controls whether the clock is output (DDE2 set) or input (DDE2 cleared). The XCK0 pin is active only when the USART0 operates in Synchronous mode.

- **PDO/TXD0 – Port E, Bit 1**

PDO, SPI Serial Programming Data Output. During Serial Program Downloading, this pin is used as data output line for the AT90CAN128.

TXD0, UART0 Transmit pin.

- **PDI/RXD0 – Port E, Bit 0**

PDI, SPI Serial Programming Data Input. During Serial Program Downloading, this pin is used as data input line for the AT90CAN128.

RXD0, USART0 Receive Pin. Receive Data (Data input pin for the USART0). When the USART0 receiver is enabled this pin is configured as an input regardless of the value of DDRE0. When the USART0 forces this pin to be an input, a logical one in PORTE0 will turn on the internal pull-up.

Table 42 and Table 43 relates the alternate functions of Port E to the overriding signals shown in Figure 34 on page 67.

**Table 42.** Overriding Signals for Alternate Functions PE7..PE4

| Signal Name | PE7/INT7/ICP3 | PE6/INT6/T3 | PE5/INT5/OC3C | PE4/INT4/OC3B |
|---|---|---|---|---|
| PUOE | 0 | 0 | 0 | 0 |
| PUOV | 0 | 0 | 0 | 0 |
| DDOE | 0 | 0 | 0 | 0 |
| DDOV | 0 | 0 | 0 | 0 |
| PVOE | 0 | 0 | OC3C ENABLE | OC3B ENABLE |
| PVOV | 0 | 0 | OC3C | OC3B |
| PTOE | 0 | 0 | 0 | 0 |
| DIEOE | INT7 ENABLE | INT6 ENABLE | INT5 ENABLE | INT4 ENABLE |
| DIEOV | INT7 ENABLE | INT6 ENABLE | INT5 ENABLE | INT4 ENABLE |
| DI | INT7 INPUT /ICP3 INPUT | INT6 INPUT /T3 INPUT | INT5 INPUT | INT4 INPUT |
| AIO | – | – | – | – |

**Table 43.** Overriding Signals for Alternate Functions in PE3..PE0

| Signal Name | PE3/AIN1/OC3A | PE2/AIN0/XCK0 | PE1/PDO/TXD0 | PE0/PDI/RXD0 |
|---|---|---|---|---|
| PUOE | 0 | 0 | TXEN0 | RXEN0 |
| PUOV | 0 | 0 | 0 | PORTE0 • $\overline{PUD}$ |
| DDOE | 0 | 0 | TXEN0 | RXEN0 |
| DDOV | 0 | 0 | 1 | 0 |
| PVOE | OC3A ENABLE | UMSEL0 | TXEN0 | 0 |
| PVOV | OC3A | XCK0 OUTPUT | TXD0 | 0 |
| PTOE | 0 | 0 | 0 | 0 |
| DIEOE | AIN1D(1) | AIN0D(1) | 0 | 0 |
| DIEOV | 0 | 0 | 0 | 0 |
| DI | 0 | XCK0 INPUT | – | RXD0 |
| AIO | AIN1 INPUT | AIN0 INPUT | – | – |

Note: 1. AIN0D and AIN1D is described in “Digital Input Disable Register 1 – DIDR1” on page 264.

**Alternate Functions of Port F**

The Port F has an alternate function as analog input for the ADC as shown in Table 44. If some Port F pins are configured as outputs, it is essential that these do not switch when a conversion is in progress. This might corrupt the result of the conversion. If the JTAG interface is enabled, the pull-up resistors on pins PF7 (TDI), PF5 (TMS) and PF4 (TCK) will be activated even if a reset occurs.

**Table 44.** Port F Pins Alternate Functions

| Port Pin | Alternate Function |
|---|---|
| PF7 | ADC7/TDI (ADC input channel 7 or JTAG Data Input) |
| PF6 | ADC6/TDO (ADC input channel 6 or JTAG Data Output) |
| PF5 | ADC5/TMS (ADC input channel 5 or JTAG mode Select) |
| PF4 | ADC4/TCK (ADC input channel 4 or JTAG ClocK) |
| PF3 | ADC3 (ADC input channel 3) |
| PF2 | ADC2 (ADC input channel 2) |
| PF1 | ADC1 (ADC input channel 1) |
| PF0 | ADC0 (ADC input channel 0) |

The alternate pin configuration is as follows:

- **TDI, ADC7 – Port F, Bit 7**

ADC7, Analog to Digital Converter, input channel 7.

TDI, JTAG Test Data In. Serial input data to be shifted in to the Instruction Register or Data Register (scan chains). When the JTAG interface is enabled, this pin can not be used as an I/O pin.

- **TCK, ADC6 – Port F, Bit 6**

ADC6, Analog to Digital Converter, input channel 6.

TDO, JTAG Test Data Out. Serial output data from Instruction Register or Data Register. When the JTAG interface is enabled, this pin can not be used as an I/O pin.

- **TMS, ADC5 – Port F, Bit 5**

ADC5, Analog to Digital Converter, input channel 5.

TMS, JTAG Test mode Select. This pin is used for navigating through the TAP-controller state machine. When the JTAG interface is enabled, this pin can not be used as an I/O pin.

- **TDO, ADC4 – Port F, Bit 4**

ADC4, Analog to Digital Converter, input channel 4.

TCK, JTAG Test Clock. JTAG operation is synchronous to TCK. When the JTAG interface is enabled, this pin can not be used as an I/O pin.

- **ADC3 – Port F, Bit 3**

ADC3, Analog to Digital Converter, input channel 3.

- **ADC2 – Port F, Bit 2**

ADC2, Analog to Digital Converter, input channel 2.

- **ADC1 – Port F, Bit 1**

ADC1, Analog to Digital Converter, input channel 1.

- **ADC0 – Port F, Bit 0**

ADC0, Analog to Digital Converter, input channel 0.

Table 45 and Table 46 relates the alternate functions of Port F to the overriding signals shown in Figure 34 on page 67.

**Table 45.** Overriding Signals for Alternate Functions in PF7..PF4

| Signal Name | PF7/ADC7/TDI | PF6/ADC6/TDO | PF5/ADC5/TMS | PF4/ADC4/TCK |
|---|---|---|---|---|
| PUOE | JTAGEN | JTAGEN | JTAGEN | JTAGEN |
| PUOV | JTAGEN | JTAGEN | JTAGEN | JTAGEN |
| DDOE | JTAGEN | JTAGEN | JTAGEN | JTAGEN |
| DDOV | 0 | SHIFT_IR + SHIFT_DR | 0 | 0 |
| PVOE | JTAGEN | JTAGEN | JTAGEN | JTAGEN |
| PVOV | 0 | TDO | 0 | 0 |
| PTOE | 0 | 0 | 0 | 0 |
| DIEOE | JTAGEN + ADC7D | JTAGEN + ADC6D | JTAGEN + ADC5D | JTAGEN + ADC4D |
| DIEOV | JTAGEN | 0 | JTAGEN | JTAGEN |
| DI | TDI | – | TMS | TCK |
| AIO | ADC7 INPUT | ADC6 INPUT | ADC5 INPUT | ADC4 INPUT |

**Table 46.** Overriding Signals for Alternate Functions in PF3..PF0

| Signal Name | PF3/ADC3 | PF2/ADC2 | PF1/ADC1 | PF0/ADC0 |
|---|---|---|---|---|
| PUOE | 0 | 0 | 0 | 0 |
| PUOV | 0 | 0 | 0 | 0 |
| DDOE | 0 | 0 | 0 | 0 |
| DDOV | 0 | 0 | 0 | 0 |
| PVOE | 0 | 0 | 0 | 0 |
| PVOV | 0 | 0 | 0 | 0 |
| PTOE | 0 | 0 | 0 | 0 |
| DIEOE | ADC3D | ADC2D | ADC1D | ADC0D |
| DIEOV | 0 | 0 | 0 | 0 |
| DI | – | – | – | – |
| AIO | ADC3 INPUT | ADC2 INPUT | ADC1 INPUT | ADC0 INPUT |

**Alternate Functions of Port G**

The alternate pin configuration is as follows:

**Table 47.** Port G Pins Alternate Functions

| Port Pin | Alternate Function |
|---|---|
| PG4 | TOSC1 (RTC Oscillator Timer/Counter2) |
| PG3 | TOSC2 (RTC Oscillator Timer/Counter2) |
| PG2 | ALE (Address Latch Enable to external memory) |
| PG1 | $\overline{RD}$ (Read strobe to external memory) |
| PG0 | $\overline{WR}$ (Write strobe to external memory) |

The alternate pin configuration is as follows:

- **TOSC1 – Port G, Bit 4**

TOSC2, Timer/Counter2 Oscillator pin 1. When the AS2 bit in ASSR is set (one) to enable asynchronous clocking of Timer/Counter2, pin PG4 is disconnected from the port, and becomes the input of the inverting Oscillator amplifier. In this mode, a Crystal Oscillator is connected to this pin, and the pin can not be used as an I/O pin.

- **TOSC2 – Port G, Bit 3**

TOSC2, Timer/Counter2 Oscillator pin 2. When the AS2 bit in ASSR is set (one) to enable asynchronous clocking of Timer/Counter2, pin PG3 is disconnected from the port, and becomes the inverting output of the Oscillator amplifier. In this mode, a Crystal Oscillator is connected to this pin, and the pin can not be used as an I/O pin.

- **ALE – Port G, Bit 2**

ALE is the external data memory Address Latch Enable signal.

- **$\overline{RD}$ – Port G, Bit 1**

$\overline{RD}$ is the external data memory read control strobe.

- **$\overline{WR}$ – Port G, Bit 0**

$\overline{WR}$ is the external data memory write control strobe.

Table 47 and Table 48 relates the alternate functions of Port G to the overriding signals shown in Figure 34 on page 67.

**Table 48.** Overriding Signals for Alternate Function in PG4

| Signal Name | - | - | - | PG4/TOSC1 |
|---|---|---|---|---|
| PUOE | | | | AS2 |
| PUOV | | | | 0 |
| DDOE | | | | AS2 |
| DDOV | | | | 0 |
| PVOE | | | | 0 |
| PVOV | | | | 0 |
| PTOE | | | | 0 |
| DIEOE | | | | AS2 |
| DIEOV | | | | EXCLK |
| DI | | | | – |
| AIO | | | | T/C2 OSC INPUT |

**Table 49.** Overriding Signals for Alternate Functions in PG3:0

| Signal Name | PG3/TOSC2 | PG2/ALE | PG1/$\overline{RD}$ | PG0/$\overline{WR}$ |
|---|---|---|---|---|
| PUOE | AS2 • $\overline{EXCLK}$ | SRE | SRE | SRE |
| PUOV | 0 | 0 | 0 | 0 |
| DDOE | AS2 • $\overline{EXCLK}$ | SRE | SRE | SRE |
| DDOV | 0 | 1 | 1 | 1 |
| PVOE | 0 | SRE | SRE | SRE |
| PVOV | 0 | ALE | RD | WR |
| PTOE | 0 | 0 | 0 | 0 |
| DIEOE | AS2 | 0 | 0 | 0 |
| DIEOV | 0 | 0 | 0 | 0 |
| DI | – | – | – | – |
| AIO | T/C2 OSC OUTPUT | – | – | – |

## Register Description for I/O-Ports

### Port A Data Register – PORTA

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PORTA7 | PORTA6 | PORTA5 | PORTA4 | PORTA3 | PORTA2 | PORTA1 | PORTA0 | PORTA |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port A Data Direction Register – DDRA

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | DDA7 | DDA6 | DDA5 | DDA4 | DDA3 | DDA2 | DDA1 | DDA0 | DDRA |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port A Input Pins Address – PINA

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PINA7 | PINA6 | PINA5 | PINA4 | PINA3 | PINA2 | PINA1 | PINA0 | PINA |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | |

### Port B Data Register – PORTB

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PORTB7 | PORTB6 | PORTB5 | PORTB4 | PORTB3 | PORTB2 | PORTB1 | PORTB0 | PORTB |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port B Data Direction Register – DDRB

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | DDB7 | DDB6 | DDB5 | DDB4 | DDB3 | DDB2 | DDB1 | DDB0 | DDRB |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port B Input Pins Address – PINB

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PINB7 | PINB6 | PINB5 | PINB4 | PINB3 | PINB2 | PINB1 | PINB0 | PINB |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | |

### Port C Data Register – PORTC

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PORTC7 | PORTC6 | PORTC5 | PORTC4 | PORTC3 | PORTC2 | PORTC1 | PORTC0 | PORTC |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port C Data Direction Register – DDRC

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | DDC7 | DDC6 | DDC5 | DDC4 | DDC3 | DDC2 | DDC1 | DDC0 | DDRC |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port C Input Pins Address – PINC

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PINC7 | PINC6 | PINC5 | PINC4 | PINC3 | PINC2 | PINC1 | PINC0 | PINC |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | |

### Port D Data Register – PORTD

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PORTD7 | PORTD6 | PORTD5 | PORTD4 | PORTD3 | PORTD2 | PORTD1 | PORTD0 | PORTD |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port D Data Direction Register – DDRD

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | DDD7 | DDD6 | DDD5 | DDD4 | DDD3 | DDD2 | DDD1 | DDD0 | DDRD |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port D Input Pins Address – PIND

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PIND7 | PIND6 | PIND5 | PIND4 | PIND3 | PIND2 | PIND1 | PIND0 | PIND |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | |

### Port E Data Register – PORTE

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PORTE7 | PORTE6 | PORTE5 | PORTE4 | PORTE3 | PORTE2 | PORTE1 | PORTE0 | PORTE |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port E Data Direction Register – DDRE

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | DDE7 | DDE6 | DDE5 | DDE4 | DDE3 | DDE2 | DDE1 | DDE0 | DDRE |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port E Input Pins Address – PINE

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PINE7 | PINE6 | PINE5 | PINE4 | PINE3 | PINE2 | PINE1 | PINE0 | PINE |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | |

### Port F Data Register – PORTF

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PORTF7 | PORTF6 | PORTF5 | PORTF4 | PORTF3 | PORTF2 | PORTF1 | PORTF0 | PORTF |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port F Data Direction Register – DDRF

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | DDF7 | DDF6 | DDF5 | DDF4 | DDF3 | DDF2 | DDF1 | DDF0 | DDRF |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port F Input Pins Address – PINF

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **PINF7** | **PINF6** | **PINF5** | **PINF4** | **PINF3** | **PINF2** | **PINF1** | **PINF0** | **PINF** |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | |

### Port G Data Register – PORTG

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **–** | **–** | **–** | **PORTG4** | **PORTG3** | **PORTG2** | **PORTG1** | **PORTG0** | **PORTG** |
| Read/Write | R | R | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port G Data Direction Register – DDRG

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **–** | **–** | **–** | **DDG4** | **DDG3** | **DDG2** | **DDG1** | **DDG0** | **DDRG** |
| Read/Write | R | R | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port G Input Pins Address – PING

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **–** | **–** | **–** | **PING4** | **PING3** | **PING2** | **PING1** | **PING0** | **PING** |
| Read/Write | R | R | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | N/A | N/A | N/A | N/A | N/A | |

## External Interrupts

The External Interrupts are triggered by the INT7:0 pins. Observe that, if enabled, the interrupts will trigger even if the INT7:0 pins are configured as outputs. This feature provides a way of generating a software interrupt. The External Interrupts can be triggered by a falling or rising edge or a low level. This is set up as indicated in the specification for the External Interrupt Control Registers – EICRA (INT3:0) and EICRB (INT7:4). When the external interrupt is enabled and is configured as level triggered, the interrupt will trigger as long as the pin is held low. Note that recognition of falling or rising edge interrupts on INT7:4 requires the presence of an I/O clock, described in “Clock Systems and their Distribution” on page 34. Low level interrupts and the edge interrupt on INT3:0 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than Idle mode. The I/O clock is halted in all sleep modes except Idle mode.

Note that if a level triggered interrupt is used for wake-up from Power-down mode, the changed level must be held for some time to wake up the MCU. This makes the MCU less sensitive to noise. The changed level is sampled twice by the Watchdog Oscillator clock. The period of the Watchdog Oscillator is 1 µs (nominal) at 5.0V and 25°C. The frequency of the Watchdog Oscillator is voltage dependent as shown in the “Electrical Characteristics(1)” on page 355. The MCU will wake up if the input has the required level during this sampling or if it is held until the end of the start-up time. The start-up time is defined by the SUT fuses as described in “System Clock” on page 34. If the level is sampled twice by the Watchdog Oscillator clock but disappears before the end of the start-up time, the MCU will still wake up, but no interrupt will be generated. The required level must be held long enough for the MCU to complete the wake up to trigger the level interrupt.

### External Interrupt Control Register A – EICRA

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **ISC31** | **ISC30** | **ISC21** | **ISC20** | **ISC11** | **ISC10** | **ISC01** | **ISC00** | **EICRA** |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bits 7..0 – ISC31, ISC30 – ISC01, ISC00: External Interrupt 3 - 0 Sense Control Bits**

The External Interrupts 3 - 0 are activated by the external pins INT3:0 if the SREG I-flag and the corresponding interrupt mask in the EIMSK is set. The level and edges on the external pins that activate the interrupts are defined in Table 50. Edges on INT3..INT0 are registered asynchronously. Pulses on INT3:0 pins wider than the minimum pulse width given in Table 51 will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt. If enabled, a level triggered interrupt will generate an interrupt request as long as the pin is held low. When changing the ISCn bit, an interrupt can occur. Therefore, it is recommended to first disable INTn by clearing its Interrupt Enable bit in the EIMSK Register. Then, the ISCn bit can be changed. Finally, the INTn interrupt flag should be cleared by writing a logical one to its Interrupt Flag bit (INTFn) in the EIFR Register before the interrupt is re-enabled.

**Table 50.** Interrupt Sense Control(1)

| ISCn1 | ISCn0 | Description |
|---|---|---|
| 0 | 0 | The low level of INTn generates an interrupt request. |
| 0 | 1 | Reserved |
| 1 | 0 | The falling edge of INTn generates asynchronously an interrupt request. |
| 1 | 1 | The rising edge of INTn generates asynchronously an interrupt request. |

Note: 1. n = 3, 2, 1or 0.
When changing the ISCn1/ISCn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.

**Table 51.** Asynchronous External Interrupt Characteristics

| Symbol | Parameter | Condition | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| $t_{INT}$ | Minimum pulse width for asynchronous external interrupt | | | 50 | | ns |

### External Interrupt Control Register B – EICRB

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | ISC71 | ISC70 | ISC61 | ISC60 | ISC51 | ISC50 | ISC41 | ISC40 | EICRB |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bits 7..0 – ISC71, ISC70 - ISC41, ISC40: External Interrupt 7 - 4 Sense Control Bits**

The External Interrupts 7 - 4 are activated by the external pins INT7:4 if the SREG I-flag and the corresponding interrupt mask in the EIMSK is set. The level and edges on the external pins that activate the interrupts are defined in Table 52. The value on the INT7:4 pins are sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. Observe that CPU clock frequency can be lower than the XTAL frequency if the XTAL divider is enabled. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt. If enabled, a level triggered interrupt will generate an interrupt request as long as the pin is held low.

**Table 52.** Interrupt Sense Control(1)

| ISCn1 | ISCn0 | Description |
|---|---|---|
| 0 | 0 | The low level of INTn generates an interrupt request. |
| 0 | 1 | Any logical change on INTn generates an interrupt request |
| 1 | 0 | The falling edge between two samples of INTn generates an interrupt request. |
| 1 | 1 | The rising edge between two samples of INTn generates an interrupt request. |

Note: 1. n = 7, 6, 5 or 4.
When changing the ISCn1/ISCn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.

### External Interrupt Mask Register – EIMSK

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | INT7 | INT6 | INT5 | INT4 | INT3 | INT2 | INT1 | IINT0 | EIMSK |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bits 7..0 – INT7 – INT0: External Interrupt Request 7 - 0 Enable**

When an INT7 – INT0 bit is written to one and the I-bit in the Status Register (SREG) is set (one), the corresponding external pin interrupt is enabled. The Interrupt Sense Control bits in the External Interrupt Control Registers – EICRA and EICRB – defines whether the external interrupt is activated on rising or falling edge or level sensed. Activity on any of these pins will trigger an interrupt request even if the pin is enabled as an output. This provides a way of generating a software interrupt.

### External Interrupt Flag Register – EIFR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | INTF7 | INTF6 | INTF5 | INTF4 | INTF3 | INTF2 | INTF1 | IINTF0 | EIFR |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bits 7..0 – INTF7 - INTF0: External Interrupt Flags 7 - 0**

When an edge or logic change on the INT7:0 pin triggers an interrupt request, INTF7:0 becomes set (one). If the I-bit in SREG and the corresponding interrupt enable bit, INT7:0 in EIMSK, are set (one), the MCU will jump to the interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. These flags are always cleared when INT7:0 are configured as level interrupt. Note that when entering sleep mode with the INT3:0 interrupts disabled, the input buffers on these pins will be disabled. This may cause a logic change in internal signals which will set the INTF3:0 flags. See “Digital Input Enable and Sleep Modes” on page 65 for more information.

## Timer/Counter3/1/0 Prescalers

Timer/Counter3, Timer/Counter1 and Timer/Counter0 share the same prescaler module, but the Timer/Counters can have different prescaler settings. The description below applies to both Timer/Counter3, Timer/Counter1 and Timer/Counter0.

### Overview

Most bit references in this section are written in general form. A lower case “n” replaces the Timer/Counter number.

#### Internal Clock Source

The Timer/Counter can be clocked directly by the system clock (by setting the CSn2:0 = 1). This provides the fastest operation, with a maximum Timer/Counter clock frequency equal to system clock frequency ($f_{CLK_I/O}$). Alternatively, one of four taps from the prescaler can be used as a clock source. The prescaled clock has a frequency of either $f_{CLK_I/O}/8$, $f_{CLK_I/O}/64$, $f_{CLK_I/O}/256$, or $f_{CLK_I/O}/1024$.

#### Prescaler Reset

The prescaler is free running, i.e., operates independently of the Clock Select logic of the Timer/Counter, and it is shared by Timer/Counter3, Timer/Counter1 and Timer/Counter0. Since the prescaler is not affected by the Timer/Counter’s clock select, the state of the prescaler will have implications for situations where a prescaled clock is used. One example of prescaling artifacts occurs when the timer is enabled and clocked by the prescaler (6 > CSn2:0 > 1). The number of system clock cycles from when the timer is enabled to the first count occurs can be from 1 to N+1 system clock cycles, where N equals the prescaler divisor (8, 64, 256, or 1024).

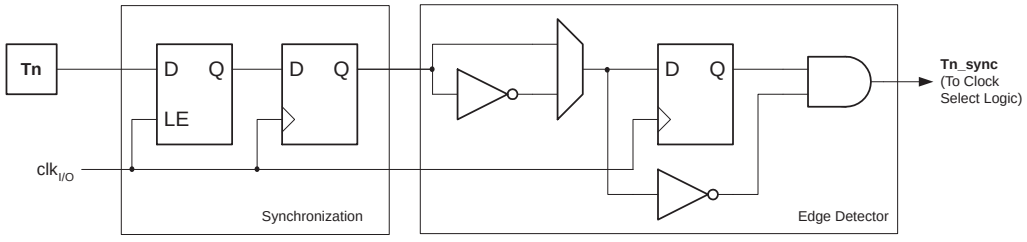
It is possible to use the prescaler reset for synchronizing the Timer/Counter to program execution. However, care must be taken if the other Timer/Counter that shares the same prescaler also uses prescaling. A prescaler reset will affect the prescaler period for all Timer/Counters it is connected to.

#### External Clock Source

An external clock source applied to the T3/T1/T0 pin can be used as Timer/Counter clock ($clk_{T3}/clk_{T1}/clk_{T0}$). The T3/T1/T0 pin is sampled once every system clock cycle by the pin synchronization logic. The synchronized (sampled) signal is then passed through the edge detector. Figure 35 shows a functional equivalent block diagram of the T3/T1/T0 synchronization and edge detector logic. The registers are clocked at the positive edge of the internal system clock ($clk_{I/O}$). The latch is transparent in the high period of the internal system clock.

The edge detector generates one $clk_{T3}/clk_{T1}/clk_{T0}$ pulse for each positive (CSn2:0 = 7) or negative (CSn2:0 = 6) edge it detects.

**Figure 35.** T3/T1/T0 Pin Sampling



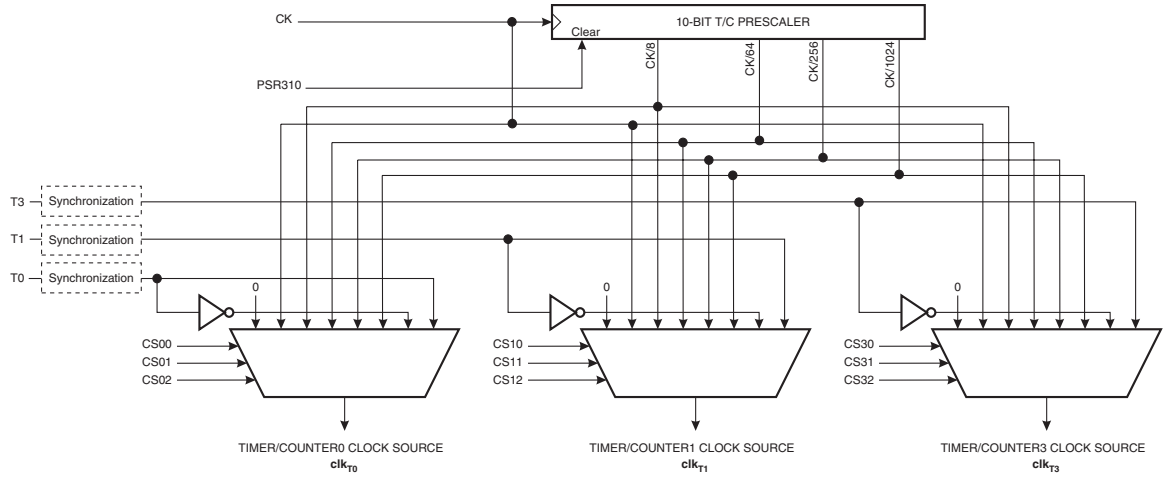

The synchronization and edge detector logic introduces a delay of 2.5 to 3.5 system clock cycles from an edge has been applied to the T3/T1/T0 pin to the counter is updated.

Enabling and disabling of the clock input must be done when T3/T1/T0 has been stable for at least one system clock cycle, otherwise it is a risk that a false Timer/Counter clock pulse is generated.

Each half period of the external clock applied must be longer than one system clock cycle to ensure correct sampling. The external clock must be guaranteed to have less than half the system clock frequency ($f_{ExtClk} < f_{clk_I/O}/2$) given a 50/50 % duty cycle. Since the edge detector uses sampling, the maximum frequency of an external clock it can detect is half the sampling frequency (Nyquist sampling theorem). However, due to variation of the system clock frequency and duty cycle caused by Oscillator source (crystal, resonator, and capacitors) tolerances, it is recommended that maximum frequency of an external clock source is less than $f_{clk_I/O}/2.5$.

An external clock source can not be prescaled.

**Figure 36.** Prescaler for Timer/Counter3, Timer/Counter1 and Timer/Counter0[1]




Note: 1. The synchronization logic on the input pins (T0/T1/T3) is shown in Figure 35.

## Timer/Counter0/1/3 Prescalers Register Description

### General Timer/Counter Control Register – GTCCR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **TSM** | **–** | **–** | **–** | **–** | **–** | **PSR2** | **PSR310** | **GTCCR** |
| Read/Write | R | R | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – TSM: Timer/Counter Synchronization Mode**

Writing the TSM bit to one activates the Timer/Counter Synchronization mode. In this mode, the value that is written to the PSR2 and PSR310 bits is kept, hence keeping the corresponding prescaler reset signals asserted. This ensures that the corresponding Timer/Counters are halted and can be configured to the same value without the risk of one of them advancing during configuration. When the TSM bit is written to zero, the PSR2 and PSR310 bits are cleared by hardware, and the Timer/Counters start counting simultaneously.

- **Bit 0 – PSR310: Prescaler Reset Timer/Counter3, Timer/Counter1 and Timer/Counter0**

When this bit is one, Timer/Counter3, Timer/Counter1 and Timer/Counter0 prescaler will be Reset. This bit is normally cleared immediately by hardware, except if the TSM bit is set. Note that Timer/Counter3, Timer/Counter1 and Timer/Counter0 share the same prescaler and a reset of this prescaler will affect these three timers.

# 8-bit Timer/Counter0 with PWM

Timer/Counter0 is a general purpose, single channel, 8-bit Timer/Counter module. The main features are:

## Features

- **Single Channel Counter**
- **Clear Timer on Compare Match (Auto Reload)**
- **Glitch-free, Phase Correct Pulse Width Modulator (PWM)**
- **Frequency Generator**
- **External Event Counter**
- **10-bit Clock Prescaler**
- **Overflow and Compare Match Interrupt Sources (TOV0 and OCF0A)**

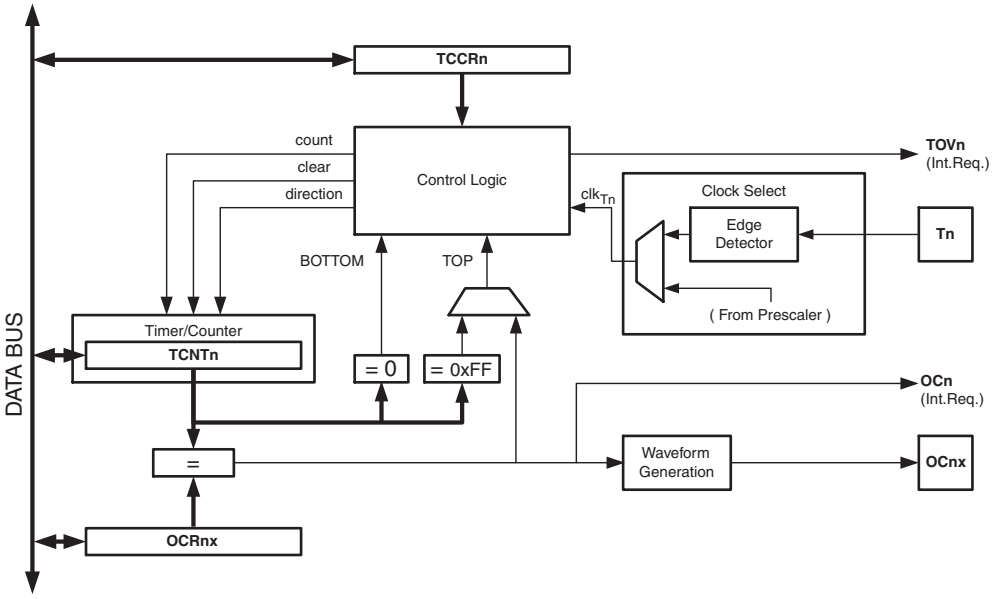
## Overview

Many register and bit references in this section are written in general form.

- A lower case “n” replaces the Timer/Counter number, in this case 0. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT0 for accessing Timer/Counter0 counter value and so on.
- A lower case “x” replaces the Output Compare unit channel, in this case A. However, when using the register or bit defines in a program, the precise form must be used, i.e., OCR0A for accessing Timer/Counter0 output compare channel A value and so on.

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 37. For the actual placement of I/O pins, refer to “Pinout AT90CAN128- TQFP” on page 4. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the “8-bit Timer/Counter Register Description” on page 104.

**Figure 37.** 8-bit Timer/Counter Block Diagram




## Registers

The Timer/Counter (TCNT0) and Output Compare Register (OCR0A) are 8-bit registers. Interrupt request (abbreviated to Int.Req. in the figure) signals are all visible in the Timer

Interrupt Flag Register (TIFR0). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK0). TIFR0 and TIMSK0 are not shown in the figure.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T0 pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the Clock Select logic is referred to as the timer clock ($clk_{T0}$).

The double buffered Output Compare Register (OCR0A) is compared with the Timer/Counter value at all times. The result of the compare can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pin (OC0A). See “Output Compare Unit” on page 96 for details. The compare match event will also set the Compare Flag (OCF0A) which can be used to generate an Output Compare interrupt request.

**Definitions**

The definitions in Table 53 are also used extensively throughout the document.

**Table 53.** Definitions

| | |
|---|---|
| BOTTOM | The counter reaches the BOTTOM when it becomes 0x00. |
| MAX | The counter reaches its MAXimum when it becomes 0xFF (decimal 255). |
| TOP | The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be the fixed value 0xFF (MAX) or the value stored in the OCR0A Register. The assignment is dependent on the mode of operation. |

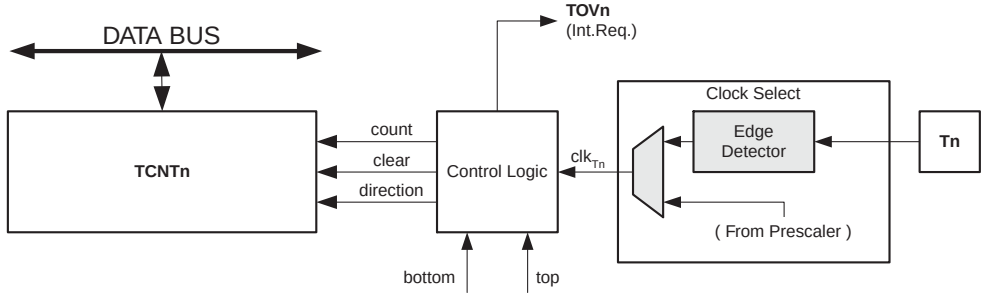
## Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the Clock Select (CS02:0) bits located in the Timer/Counter Control Register (TCCR0A). For details on clock sources and prescaler, see “Timer/Counter3/1/0 Prescalers” on page 91.

## Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 38 shows a block diagram of the counter and its surroundings.

**Figure 38.** Counter Unit Block Diagram




Signal description (internal signals):

**count** Increment or decrement TCNT0 by 1.

**direction** Select between increment and decrement.

**clear** Clear TCNT0 (set all bits to zero).

**$clk_{Tn}$** Timer/Counter clock, referred to as $clk_{T0}$ in the following.

**top** Signalize that TCNT0 has reached maximum value.

**bottom** Signalize that TCNT0 has reached minimum value (zero).

Depending of the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock ($clk_{T0}$). $clk_{T0}$ can be generated from an external or internal clock source, selected by the Clock Select bits (CS02:0). When no clock source is selected (CS02:0 = 0) the timer is stopped. However, the TCNT0 value can be accessed by the CPU, regardless of whether $clk_{T0}$ is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the WGM01 and WGM00 bits located in the Timer/Counter Control Register (TCCR0A). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare output OC0A. For more details about advanced counting sequences and waveform generation, see “Modes of Operation” on page 99.

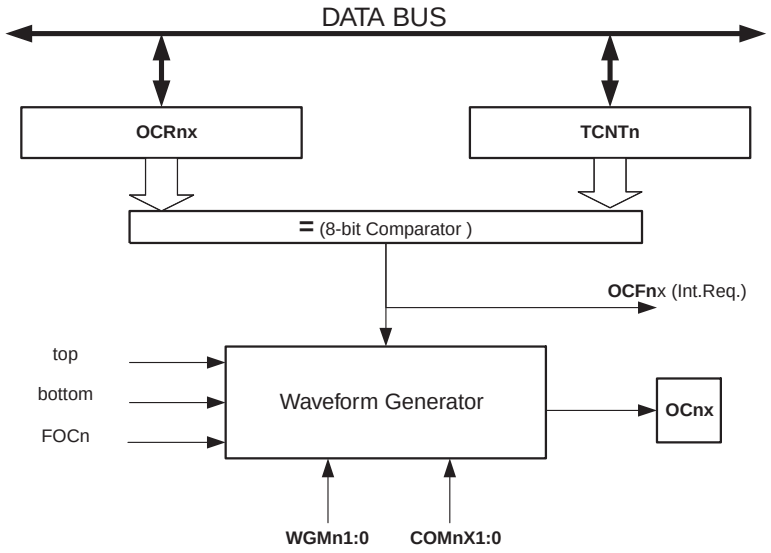
The Timer/Counter Overflow Flag (TOV0) is set according to the mode of operation selected by the WGM01:0 bits. TOV0 can be used for generating a CPU interrupt.

## Output Compare Unit

The 8-bit comparator continuously compares TCNT0 with the Output Compare Register (OCR0A). Whenever TCNT0 equals OCR0A, the comparator signals a match. A match will set the Output Compare Flag (OCF0A) at the next timer clock cycle. If enabled (OCIE0A = 1 and Global Interrupt Flag in SREG is set), the Output Compare Flag generates an Output Compare interrupt. The OCF0A flag is automatically cleared when the interrupt is executed. Alternatively, the OCF0A flag can be cleared by software by writing a logical one to its I/O bit location. The Waveform Generator uses the match signal to generate an output according to operating mode set by the WGM01:0 bits and Compare Output mode (COM0A1:0) bits. The max and bottom signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation (See “Modes of Operation” on page 99 ).

Figure 39 shows a block diagram of the Output Compare unit.

**Figure 39.** Output Compare Unit, Block Diagram



The OCR0A Register is double buffered when using any of the Pulse Width Modulation (PWM) modes. For the normal and Clear Timer on Compare (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR0A Compare Register to either top or bottom of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR0A Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR0A Buffer Register, and if double buffering is disabled the CPU will access the OCR0A directly.

**Force Output Compare**

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the Force Output Compare (FOC0A) bit. Forcing compare match will not set the OCF0A flag or reload/clear the timer, but the OC0A pin will be updated as if a real compare match had occurred (the COM0A1:0 bits settings define whether the OC0A pin is set, cleared or toggled).

**Compare Match Blocking by TCNT0 Write**

All CPU write operations to the TCNT0 Register will block any compare match that occur in the next timer clock cycle, even when the timer is stopped. This feature allows OCR0A to be initialized to the same value as TCNT0 without triggering an interrupt when the Timer/Counter clock is enabled.

**Using the Output Compare Unit**

Since writing TCNT0 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT0 when using the Output Compare channel, independently of whether the Timer/Counter is running or not. If the value written to TCNT0 equals the OCR0A value, the compare match will be missed, resulting in incorrect waveform generation. Similarly, do not write the TCNT0 value equal to BOTTOM when the counter is downcounting.

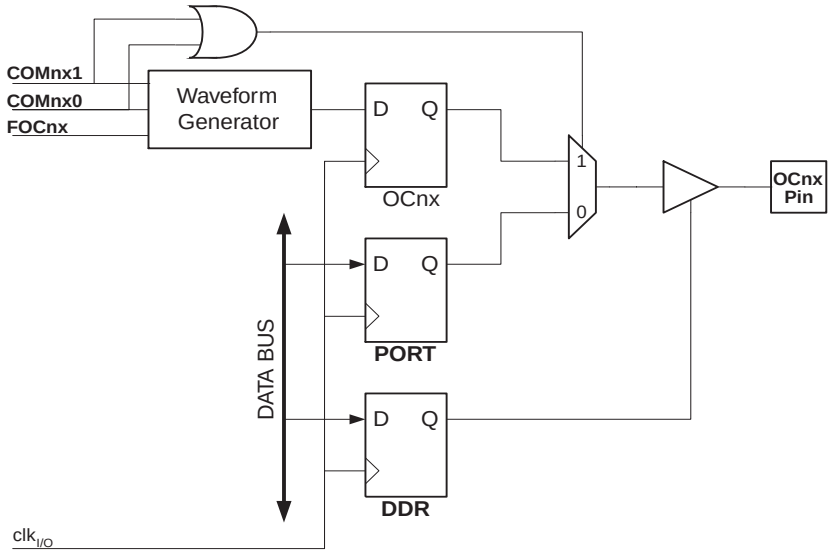
The setup of the OC0A should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC0A value is to use the Force Output Compare (FOC0A) strobe bits in Normal mode. The OC0A Register keeps its value even when changing between Waveform Generation modes.

Be aware that the COM0A1:0 bits are not double buffered together with the compare value. Changing the COM0A1:0 bits will take effect immediately.

## Compare Match Output Unit

The Compare Output mode (COM0A1:0) bits have two functions. The Waveform Generator uses the COM0A1:0 bits for defining the Output Compare (OC0A) state at the next compare match. Also, the COM0A1:0 bits control the OC0A pin output source. Figure 40 shows a simplified schematic of the logic affected by the COM0A1:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O port control registers (DDR and PORT) that are affected by the COM0A1:0 bits are shown. When referring to the OC0A state, the reference is for the internal OC0A Register, not the OC0A pin. If a system reset occur, the OC0A Register is reset to “0”.

**Figure 40.** Compare Match Output Unit, Schematic




### Compare Output Function

The general I/O port function is overridden by the Output Compare (OC0A) from the Waveform Generator if either of the COM0A1:0 bits are set. However, the OC0A pin direction (input or output) is still controlled by the Data Direction Register (DDR) for the port pin. The Data Direction Register bit for the OC0A pin (DDR_OC0A) must be set as output before the OC0A value is visible on the pin. The port override function is independent of the Waveform Generation mode.

The design of the Output Compare pin logic allows initialization of the OC0A state before the output is enabled. Note that some COM0A1:0 bit settings are reserved for certain modes of operation. See “8-bit Timer/Counter Register Description” on page 104

### Compare Output Mode and Waveform Generation

The Waveform Generator uses the COM0A1:0 bits differently in Normal, CTC, and PWM modes. For all modes, setting the COM0A1:0 = 0 tells the Waveform Generator that no action on the OC0A Register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 55 on page 105. For fast PWM mode, refer to Table 56 on page 105, and for phase correct PWM refer to Table 57 on page 106.

A change of the COM0A1:0 bits state will have effect at the first compare match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOC0A strobe bits.

## Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the Waveform Generation mode (WGM01:0) and Compare Output mode (COM0A1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM0A1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM0A1:0 bits control whether the output should be set, cleared, or toggled at a compare match (See “Compare Match Output Unit” on page 98 ).

For detailed timing information refer to Figure 44, Figure 45, Figure 46 and Figure 47 in “Timer/Counter Timing Diagrams” on page 102.

### Normal Mode

The simplest mode of operation is the Normal mode (WGM01:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 8-bit value (TOP = 0xFF) and then restarts from the bottom (0x00). In normal operation the Timer/Counter Overflow Flag (TOV0) will be set in the same timer clock cycle as the TCNT0 becomes zero. The TOV0 flag in this case behaves like a ninth bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV0 flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

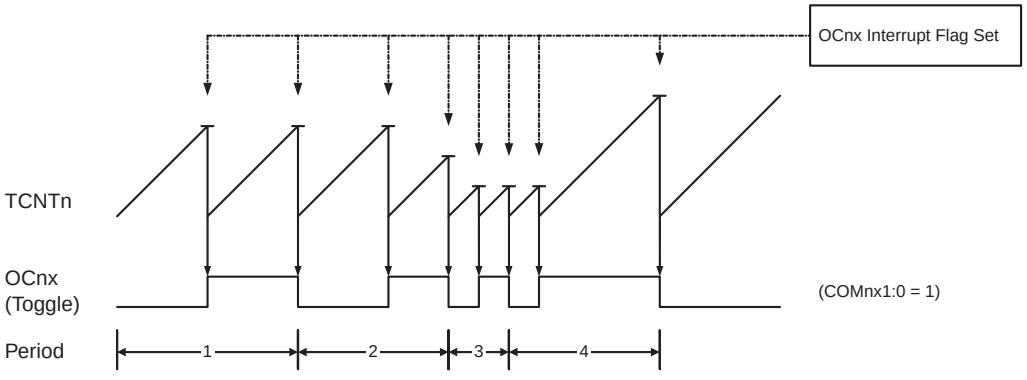
The Output Compare unit can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

### Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGM01:0 = 2), the OCR0A Register is used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT0) matches the OCR0A. The OCR0A defines the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 41. The counter value (TCNT0) increases until a compare match occurs between TCNT0 and OCR0A, and then counter (TCNT0) is cleared.

**Figure 41.** CTC Mode, Timing Diagram




An interrupt can be generated each time the counter value reaches the TOP value by using the OCF0A flag. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written

to OCR0A is lower than the current value of TCNT0, the counter will miss the compare match. The counter will then have to count to its maximum value (0xFF) and wrap around starting at 0x00 before the compare match can occur.

For generating a waveform output in CTC mode, the OC0A output can be set to toggle its logical level on each compare match by setting the Compare Output mode bits to toggle mode (COM0A1:0 = 1). The OC0A value will not be visible on the port pin unless the data direction for the pin is set to output. The waveform generated will have a maximum frequency of $f_{OC0A} = f_{clk_I/O}/2$ when OCR0A is set to zero (0x00). The waveform frequency is defined by the following equation:

$$f_{OCnx} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRnx)}$$

The N variable represents the prescale factor (1, 8, 64, 256, or 1024).

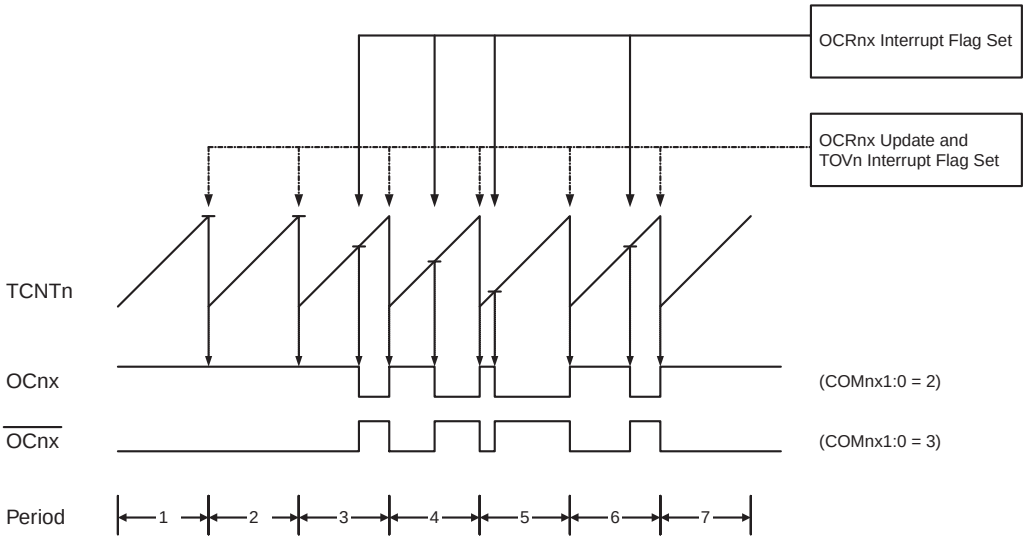
As for the Normal mode of operation, the TOV0 flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

**Fast PWM Mode**

The fast Pulse Width Modulation or fast PWM mode (WGM01:0 = 3) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to MAX then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC0A) is cleared on the compare match between TCNT0 and OCR0A, and set at BOTTOM. In inverting Compare Output mode, the output is set on compare match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct PWM mode that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), and therefore reduces total system cost.

In fast PWM mode, the counter is incremented until the counter value matches the MAX value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 42. The TCNT0 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT0 slopes represent compare matches between OCR0A and TCNT0.

**Figure 42.** Fast PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV0) is set each time the counter reaches MAX. If the interrupt is enabled, the interrupt handler routine can be used for updating the compare value.

In fast PWM mode, the compare unit allows generation of PWM waveforms on the OC0A pin. Setting the COM0A1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM0A1:0 to three (See Table 56 on page 105). The actual OC0A value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by setting (or clearing) the OC0A Register at the compare match between OCR0A and TCNT0, and clearing (or setting) the OC0A Register at the timer clock cycle the counter is cleared (changes from MAX to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{clk_I/O}}{N \cdot 256}$$

The N variable represents the prescale factor (1, 8, 64, 256, or 1024).

The extreme values for the OCR0A Register represents special cases when generating a PWM waveform output in the fast PWM mode. If the OCR0A is set equal to BOTTOM, the output will be a narrow spike for each MAX+1 timer clock cycle. Setting the OCR0A equal to MAX will result in a constantly high or low output (depending on the polarity of the output set by the COM0A1:0 bits.)

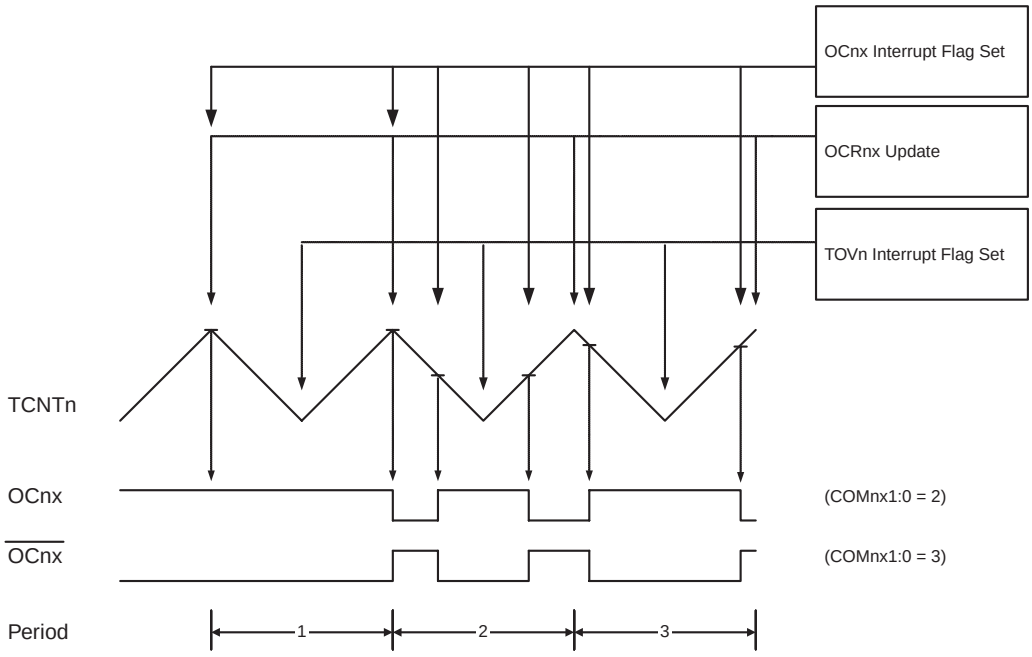
A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC0A to toggle its logical level on each compare match (COM0A1:0 = 1). The waveform generated will have a maximum frequency of $f_{OC0A} = f_{clk_I/O}/2$ when OCR0A is set to zero. This feature is similar to the OC0A toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

**Phase Correct PWM Mode**

The phase correct PWM mode (WGM01:0 = 1) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to MAX and then from MAX to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC0A) is cleared on the compare match between TCNT0 and OCR0A while upcounting, and set on the compare match while downcounting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

The PWM resolution for the phase correct PWM mode is fixed to eight bits. In phase correct PWM mode the counter is incremented until the counter value matches MAX. When the counter reaches MAX, it changes the count direction. The TCNT0 value will be equal to MAX for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 43. The TCNT0 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT0 slopes represent compare matches between OCR0A and TCNT0.

**Figure 43.** Phase Correct PWM Mode, Timing Diagram




The Timer/Counter Overflow Flag (TOV0) is set each time the counter reaches BOTTOM. The interrupt flag can be used to generate an interrupt each time the counter reaches the BOTTOM value.

In phase correct PWM mode, the compare unit allows generation of PWM waveforms on the OC0A pin. Setting the COM0A1:0 bits to two will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM0A1:0 to three (See Table 57 on page 106). The actual OC0A value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by clearing (or setting) the OC0A Register at the compare match between OCR0A and TCNT0 when the counter increments, and setting (or clearing) the OC0A Register at compare match between OCR0A and TCNT0 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnxPCPWM} = \frac{f_{clk_I/O}}{N \cdot 510}$$

The N variable represents the prescale factor (1, 8, 64, 256, or 1024).

The extreme values for the OCR0A Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR0A is set equal to BOTTOM, the output will be continuously low and if set equal to MAX the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

## Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock ($clk_{T0}$) is therefore shown as a clock enable signal in the following figures. The figures include information on when interrupt flags are set. Figure 44 contains timing data for basic Timer/Counter operation. The figure shows the count sequence close to the MAX value in all modes other than phase correct PWM mode.

**Figure 44.** Timer/Counter Timing Diagram, no Prescaling

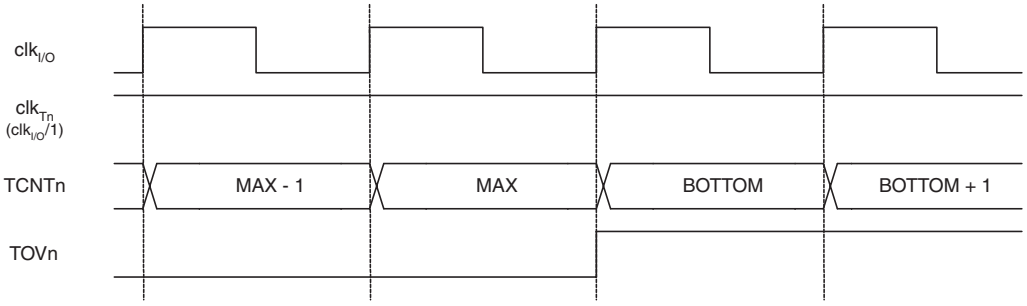



Figure 45 shows the same timing data, but with the prescaler enabled.

**Figure 45.** Timer/Counter Timing Diagram, with Prescaler ($f_{clk_I/O}/8$)

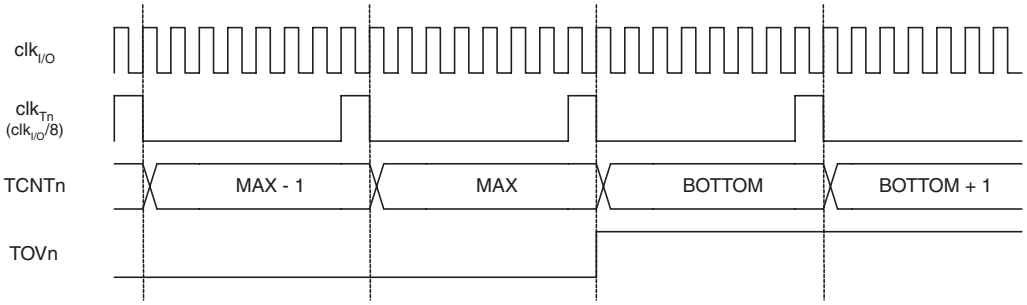



Figure 46 shows the setting of OCF0A in all modes except CTC mode.

**Figure 46.** Timer/Counter Timing Diagram, Setting of OCF0A, with Prescaler ($f_{clk_I/O}/8$)

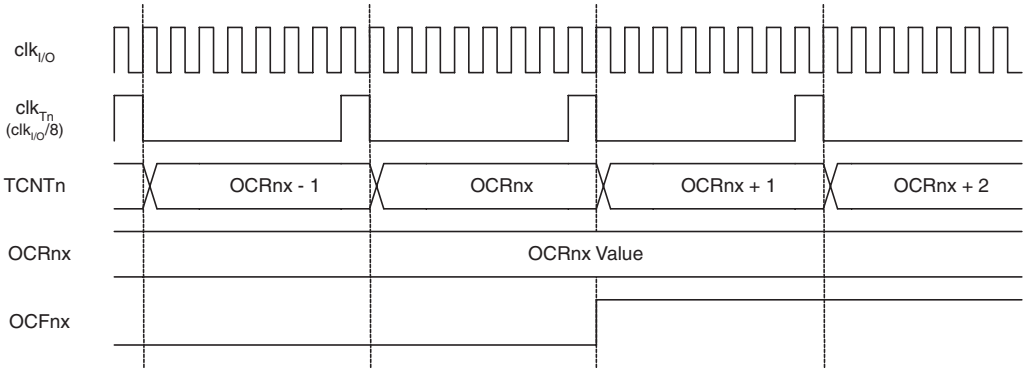
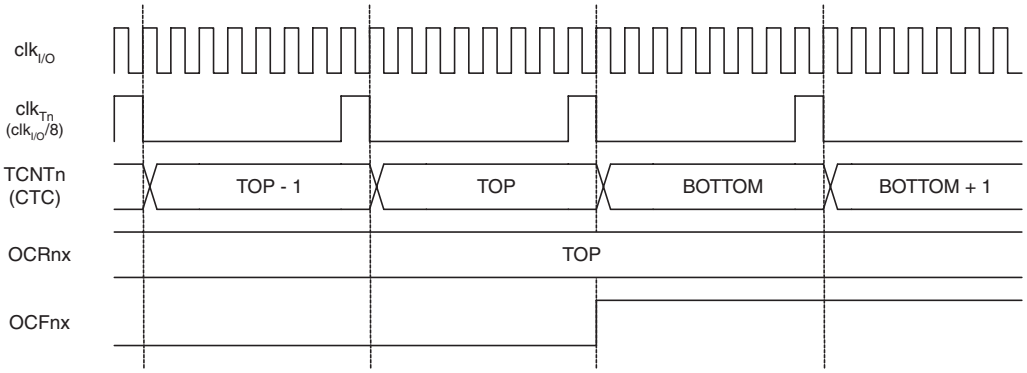


Figure 47 shows the setting of OCF0A and the clearing of TCNT0 in CTC mode.

**Figure 47.** Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{clk_I/O}/8$)




## 8-bit Timer/Counter Register Description

### Timer/Counter0 Control Register A – TCCR0A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | FOC0A | WGM00 | COM0A1 | COM0A0 | WGM01 | CS02 | CS01 | CS00 | TCCR0 |
| Read/Write | W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – FOC0A: Force Output Compare A**

The FOC0A bit is only active when the WGM00 bit specifies a non-PWM mode. However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR0 is written when operating in PWM mode. When writing a logical one to the FOC0A bit, an immediate compare match is forced on the Waveform Generation unit. The OC0A output is changed according to its COM0A1:0 bits setting. Note that the FOC0A bit is implemented as a strobe. Therefore it is the value present in the COM0A1:0 bits that determines the effect of the forced compare.

A FOC0A strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR0A as TOP.

The FOC0A bit is always read as zero.

- **Bit 6, 3 – WGM01:0: Waveform Generation Mode**

These bits control the counting sequence of the counter, the source for the maximum (TOP) counter value, and what type of waveform generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode, Clear Timer on Compare match (CTC) mode, and two types of Pulse Width Modulation (PWM) modes. See Table 54 and “Modes of Operation” on page 99.

**Table 54.** Waveform Generation Mode Bit Description[1]

| Mode | WGM01 (CTC0) | WGM00 (PWM0) | Timer/Counter Mode of Operation | TOP | Update of OCR0A at | TOV0 Flag Set on |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | Normal | 0xFF | Immediate | MAX |
| 1 | 0 | 1 | PWM, Phase Correct | 0xFF | TOP | BOTTOM |
| 2 | 1 | 0 | CTC | OCR0A | Immediate | MAX |
| 3 | 1 | 1 | Fast PWM | 0xFF | TOP | MAX |

Note: 1. The CTC0 and PWM0 bit definition names are now obsolete. Use the WGM01:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

- **Bit 5:4 – COM01:0: Compare Match Output Mode**

These bits control the Output Compare pin (OC0A) behavior. If one or both of the COM0A1:0 bits are set, the OC0A output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OC0A pin must be set in order to enable the output driver.

When OC0A is connected to the pin, the function of the COM0A1:0 bits depends on the WGM01:0 bit setting. Table 55 shows the COM0A1:0 bit functionality when the WGM01:0 bits are set to a normal or CTC mode (non-PWM).

**Table 55.** Compare Output Mode, non-PWM Mode

| COM0A1 | COM0A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC0A disconnected. |
| 0 | 1 | Toggle OC0A on compare match |
| 1 | 0 | Clear OC0A on compare match |
| 1 | 1 | Set OC0A on compare match |

Table 56 shows the COM0A1:0 bit functionality when the WGM01:0 bits are set to fast PWM mode.

**Table 56.** Compare Output Mode, Fast PWM Mode[1]

| COM0A1 | COM0A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC0A disconnected. |
| 0 | 1 | Reserved |
| 1 | 0 | Clear OC0A on compare match.<br>Set OC0A at TOP |
| 1 | 1 | Set OC0A on compare match.<br>Clear OC0A at TOP |

Note: 1. A special case occurs when OCR0A equals TOP and COM0A1 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See “Fast PWM Mode” on page 100 for more details.

Table 57 shows the COM0A1:0 bit functionality when the WGM01:0 bits are set to phase correct PWM mode.

**Table 57.** Compare Output Mode, Phase Correct PWM Mode(1)

| COM0A1 | COM0A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC0A disconnected. |
| 0 | 1 | Reserved |
| 1 | 0 | Clear OC0A on compare match when up-counting.<br>Set OC0A on compare match when downcounting. |
| 1 | 1 | Set OC0A on compare match when up-counting.<br>Clear OC0A on compare match when downcounting. |

Note: 1. A special case occurs when OCR0A equals TOP and COM0A1 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See “Phase Correct PWM Mode” on page 101 for more details.

- **Bit 2:0 – CS02:0: Clock Select**

The three Clock Select bits select the clock source to be used by the Timer/Counter.

**Table 58.** Clock Select Bit Description

| CS02 | CS01 | CS00 | Description |
|---|---|---|---|
| 0 | 0 | 0 | No clock source (Timer/Counter stopped) |
| 0 | 0 | 1 | $clk_{I/O}$/(No prescaling) |
| 0 | 1 | 0 | $clk_{I/O}$/8 (From prescaler) |
| 0 | 1 | 1 | $clk_{I/O}$/64 (From prescaler) |
| 1 | 0 | 0 | $clk_{I/O}$/256 (From prescaler) |
| 1 | 0 | 1 | $clk_{I/O}$/1024 (From prescaler) |
| 1 | 1 | 0 | External clock source on T0 pin. Clock on falling edge. |
| 1 | 1 | 1 | External clock source on T0 pin. Clock on rising edge. |

If external pin modes are used for the Timer/Counter0, transitions on the T0 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

**Timer/Counter0 Register – TCNT0**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TCNT0[7:0] | | | | | | | | TCNT0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT0 Register blocks (removes) the compare match on the following timer clock. Modifying the counter (TCNT0) while the counter is running, introduces a risk of missing a compare match between TCNT0 and the OCR0A Register.

### Output Compare Register A – OCR0A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | OCR0A[7:0] | | | | | | | | OCR0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

The Output Compare Register A contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC0A pin.

### Timer/Counter0 Interrupt Mask Register – TIMSK0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | – | – | OCIE0A | TOIE0 | TIMSK0 |
| Read/Write | R | R | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7..2 – Reserved Bits**

These are reserved bits for future use.

- **Bit 1 – OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable**

When the OCIE0A bit is written to one, and the I-bit in the Status Register is set (one), the Timer/Counter0 Compare Match A interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter0 occurs, i.e., when the OCF0A bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR0.

- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

When the TOIE0 bit is written to one, and the I-bit in the Status Register is set (one), the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR0.

### Timer/Counter0 Interrupt Flag Register – TIFR0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | – | – | OCF0A | TOV0 | TIFR0 |
| Read/Write | R | R | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 1 – OCF0A: Output Compare Flag 0 A**

The OCF0A bit is set (one) when a compare match occurs between the Timer/Counter0 and the data in OCR0A – Output Compare Register0. OCF0A is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0A is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE0A (Timer/Counter0 Compare match Interrupt Enable), and OCF0A are set (one), the Timer/Counter0 Compare match Interrupt is executed.

- **Bit 0 – TOV0: Timer/Counter0 Overflow Flag**

The bit TOV0 is set (one) when an overflow occurs in Timer/Counter0. TOV0 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV0 is cleared by writing a logic one to the flag. When the SREG I-bit, TOIE0 (Timer/Counter0 Overflow Interrupt Enable), and TOV0 are set (one), the Timer/Counter0 Overflow interrupt is executed. In phase correct PWM mode, this bit is set when Timer/Counter0 changes counting direction at 0x00.

# 16-bit Timer/Counter (Timer/Counter1 and Timer/Counter3)

The 16-bit Timer/Counter unit allows accurate program execution timing (event management), wave generation, and signal timing measurement. The main features are:

## Features

- **True 16-bit Design (i.e., Allows 16-bit PWM)**
- **Three independent Output Compare Units**
- **Double Buffered Output Compare Registers**
- **One Input Capture Unit**
- **Input Capture Noise Canceler**
- **Clear Timer on Compare Match (Auto Reload)**
- **Glitch-free, Phase Correct Pulse Width Modulator (PWM)**
- **Variable PWM Period**
- **Frequency Generator**
- **External Event Counter**
- **Four independent interrupt Sources (TOV1, OCF1A, OCF1B, and ICF1 for Timer/Counter1 - TOV3, OCF3A, OCF3B, and ICF3 for Timer/Counter3)**

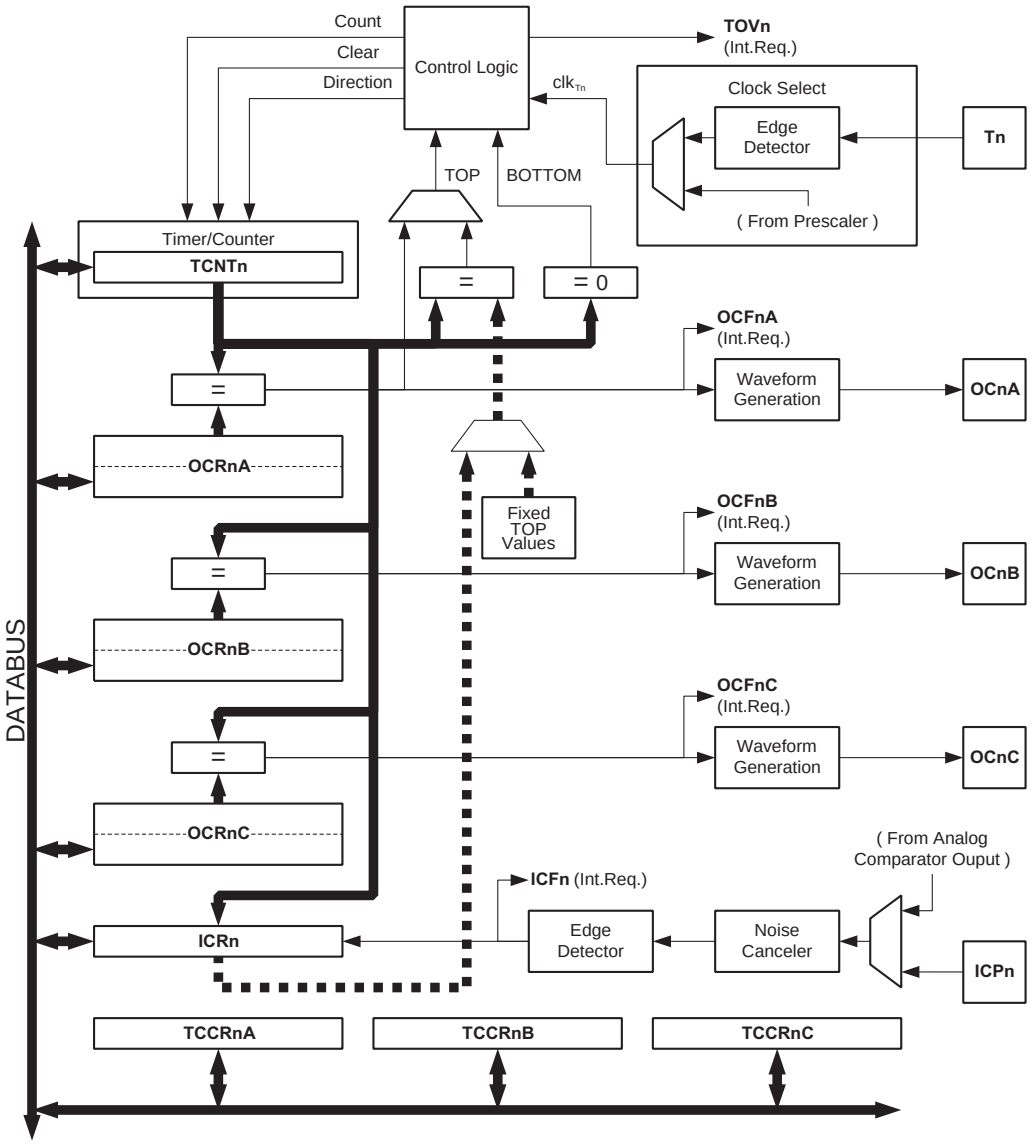
## Overview

Many register and bit references in this section are written in general form.

- A lower case “n” replaces the Timer/Counter number, in this case 1 or 3. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT1 for accessing Timer/Counter1 counter value and so on.
- A lower case “x” replaces the Output Compare unit channel, in this case A, B or C. However, when using the register or bit defines in a program, the precise form must be used, i.e., OCRnA for accessing Timer/Countern output compare channel A value and so on.

A simplified block diagram of the 16-bit Timer/Counter is shown in Figure 48. For the actual placement of I/O pins, refer to “Pinout AT90CAN128- TQFP” on page 4. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the “16-bit Timer/Counter Register Description” on page 131.

**Figure 48.** 16-bit Timer/Counter Block Diagram[1]




Note: 1. Refer to Figure 2 on page 4, Table 32 on page 71, and Table 41 on page 78 for Timer/Counter1 and 3 pin placement and description.

**Registers**

The Timer/Counter (TCNTn), Output Compare Registers (OCRnx), and Input Capture Register (ICRn) are all 16-bit registers. Special procedures must be followed when accessing the 16-bit registers. These procedures are described in the section “Accessing 16-bit Registers” on page 111. The Timer/Counter Control Registers (TCCRnx) are 8-bit registers and have no CPU access restrictions. Interrupt requests (abbreviated to Int.Req. in the figure) signals are all visible in the Timer Interrupt Flag Register (TIFRn). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSKn). TIFRn and TIMSKn are not shown in the figure.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the Tn pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is

inactive when no clock source is selected. The output from the Clock Select logic is referred to as the timer clock ($clk_{Tn}$).

The double buffered Output Compare Registers (OCRnx) are compared with the Timer/Counter value at all time. The result of the compare can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pin (OCnx). See “Output Compare Units” on page 118 . The compare match event will also set the Compare Match Flag (OCFnx) which can be used to generate an Output Compare interrupt request.

The Input Capture Register can capture the Timer/Counter value at a given external (edge triggered) event on either the Input Capture pin (ICPn) or on the Analog Comparator pins (See “Analog Comparator” on page 262 ) The Input Capture unit includes a digital filtering unit (Noise Canceler) for reducing the chance of capturing noise spikes.

The TOP value, or maximum Timer/Counter value, can in some modes of operation be defined by either the OCRnA Register, the ICRn Register, or by a set of fixed values. When using OCRnA as TOP value in a PWM mode, the OCRnA Register can not be used for generating a PWM output. However, the TOP value will in this case be double buffered allowing the TOP value to be changed in run time. If a fixed TOP value is required, the ICRn Register can be used as an alternative, freeing the OCRnA to be used as PWM output.

**Definitions**

The following definitions are used extensively throughout the section:

**Table 59.** Definitions

| | |
|---|---|
| BOTTOM | The counter reaches the *BOTTOM* when it becomes 0x0000. |
| MAX | The counter reaches its *MAX*imum when it becomes 0xFFFF (decimal 65,535). |
| TOP | The counter reaches the *TOP* when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be one of the fixed values: 0x00FF, 0x01FF, or 0x03FF, or to the value stored in the OCRnA or ICRn Register. The assignment is dependent of the mode of operation. |

**Compatibility**

The 16-bit Timer/Counter has been updated and improved from previous versions of the 16-bit AVR Timer/Counter. This 16-bit Timer/Counter is fully compatible with the earlier version regarding:

- All 16-bit Timer/Counter related I/O Register address locations, including Timer Interrupt Registers.
- Bit locations inside all 16-bit Timer/Counter Registers, including Timer Interrupt Registers.
- Interrupt Vectors.

The following control bits have changed name, but have same functionality and register location:

- PWMn0 is changed to WGMn0.
- PWMn1 is changed to WGMn1.
- CTCn is changed to WGMn2.

The following registers are added to the 16-bit Timer/Counter:

- Timer/Counter Control Register C (TCCRnC).
- Output Compare Register C, OCRnCH and OCRnCL, combined OCRnC.

The 16-bit Timer/Counter has improvements that will affect the compatibility in some special cases.

The following bits are added to the 16-bit Timer/Counter Control Registers:

- COMnC1:0 are added to TCCRnA.
- FOCnA, FOCnB and FOCnC are added to TCCRnC.
- WGMn3 is added to TCCRnB.

Interrupt flag and mask bits for output compare unit C are added.

The 16-bit Timer/Counter has improvements that will affect the compatibility in some special cases.

## Accessing 16-bit Registers

The TCNTn, OCRnx, and ICRn are 16-bit registers that can be accessed by the AVR CPU via the 8-bit data bus. The 16-bit register must be byte accessed using two read or write operations. Each 16-bit timer has a single 8-bit register for temporary storing of the high byte of the 16-bit access. The same temporary register is shared between all 16-bit registers within each 16-bit timer. Accessing the low byte triggers the 16-bit read or write operation. When the low byte of a 16-bit register is written by the CPU, the high byte stored in the temporary register, and the low byte written are both copied into the 16-bit register in the same clock cycle. When the low byte of a 16-bit register is read by the CPU, the high byte of the 16-bit register is copied into the temporary register in the same clock cycle as the low byte is read.

Not all 16-bit accesses uses the temporary register for the high byte. Reading the OCRnx 16-bit registers does not involve using the temporary register.

To do a 16-bit write, the high byte must be written before the low byte. For a 16-bit read, the low byte must be read before the high byte.

**Code Examples**

The following code examples show how to access the 16-bit timer registers assuming that no interrupts updates the temporary register. The same principle can be used directly for accessing the OCRnx and ICRn Registers. Note that when using “C”, the compiler handles the 16-bit access.

Assembly Code Examples(1)

```
...
; Set TCNTn to 0x01FF
ldi r17,0x01
ldi r16,0xFF
sts TCNTnH,r17
sts TCNTnL,r16
; Read TCNTn into r17:r16
lds r16,TCNTnL
lds r17,TCNTnH
...
```

C Code Examples(1)

```
unsigned int i;
...
/* Set TCNTn to 0x01FF */
TCNTn = 0x1FF;
/* Read TCNTn into i */
i = TCNTn;
...
```

Note: 1. The example code assumes that the part specific header file is included.

The assembly code example returns the TCNTn value in the r17:r16 register pair.

It is important to notice that accessing 16-bit registers are atomic operations. If an interrupt occurs between the two instructions accessing the 16-bit register, and the interrupt code updates the temporary register by accessing the same or any other of the 16-bit timer registers, then the result of the access outside the interrupt will be corrupted. Therefore, when both the main code and the interrupt code update the temporary register, the main code must disable the interrupts during the 16-bit access.

The following code examples show how to do an atomic read of the TCNTn Register contents. Reading any of the OCRnx or ICRn Registers can be done by using the same principle.

Assembly Code Example(1)

```
TIM16_ReadTCNTn:
  ; Save global interrupt flag
  in  r18,SREG
  ; Disable interrupts
  cli
  ; Read TCNTn into r17:r16
  lds r16,TCNTnL
  lds r17,TCNTnH
  ; Restore global interrupt flag
  out SREG,r18
  ret
```

C Code Example(1)

```
unsigned int TIM16_ReadTCNTn( void )
{
  unsigned char sreg;
  unsigned int i;
  /* Save global interrupt flag */
  sreg = SREG;
  /* Disable interrupts */
  _CLI();
  /* Read TCNTn into i */
  i = TCNTn;
  /* Restore global interrupt flag */
  SREG = sreg;
  return i;
}
```

Note: 1. The example code assumes that the part specific header file is included.
For I/O Registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

The assembly code example returns the TCNTn value in the r17:r16 register pair.

The following code examples show how to do an atomic write of the TCNTn Register contents. Writing any of the OCRnx or ICRn Registers can be done by using the same principle.

Assembly Code Example[1]

```
TIM16_WriteTCNTn:
  ; Save global interrupt flag
  in  r18,SREG
  ; Disable interrupts
  cli
  ; Set TCNTn to r17:r16
  sts TCNTnH,r17
  sts TCNTnL,r16
  ; Restore global interrupt flag
  out SREG,r18
  ret
```

C Code Example[1]

```
void TIM16_WriteTCNTn( unsigned int i )
{
  unsigned char sreg;
  unsigned int i;
  /* Save global interrupt flag */
  sreg = SREG;
  /* Disable interrupts */
  _CLI();
  /* Set TCNTn to i */
  TCNTn = i;
  /* Restore global interrupt flag */
  SREG = sreg;
}
```

Note: 1. The example code assumes that the part specific header file is included.

The assembly code example requires that the r17:r16 register pair contains the value to be written to TCNTn.

**Reusing the Temporary High Byte Register**

If writing to more than one 16-bit register where the high byte is the same for all registers written, then the high byte only needs to be written once. However, note that the same rule of atomic operation described previously also applies in this case.

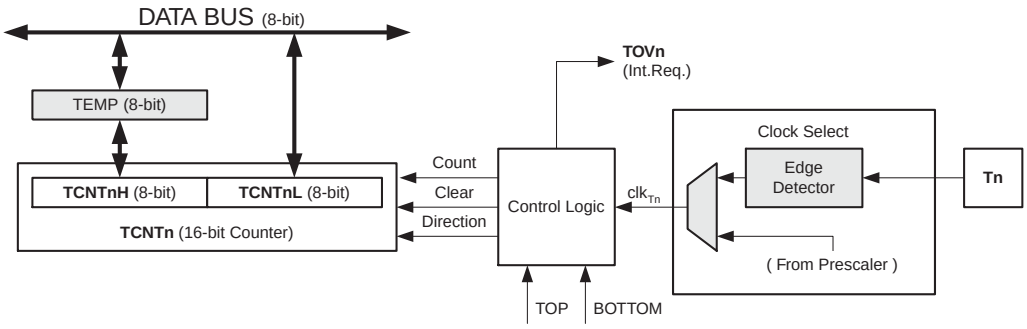
## Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the Clock Select (CSn2:0) bits located in the Timer/Counter control Register B (TCCRnB). For details on clock sources and prescaler, see “Timer/Counter3/1/0 Prescalers” on page 91.

## Counter Unit

The main part of the 16-bit Timer/Counter is the programmable 16-bit bi-directional counter unit. Figure 49 shows a block diagram of the counter and its surroundings.

**Figure 49.** Counter Unit Block Diagram




Signal description (internal signals):

| | |
|---|---|
| **Count** | Increment or decrement TCNTn by 1. |
| **Direction** | Select between increment and decrement. |
| **Clear** | Clear TCNTn (set all bits to zero). |
| **$clk_{Tn}$** | Timer/Counter clock. |
| **TOP** | Signalize that TCNTn has reached maximum value. |
| **BOTTOM** | Signalize that TCNTn has reached minimum value (zero). |

The 16-bit counter is mapped into two 8-bit I/O memory locations: Counter High (TCNTnH) containing the upper eight bits of the counter, and Counter Low (TCNTnL) containing the lower eight bits. The TCNTnH Register can only be indirectly accessed by the CPU. When the CPU does an access to the TCNTnH I/O location, the CPU accesses the high byte temporary register (TEMP). The temporary register is updated with the TCNTnH value when the TCNTnL is read, and TCNTnH is updated with the temporary register value when TCNTnL is written. This allows the CPU to read or write the entire 16-bit counter value within one clock cycle via the 8-bit data bus. It is important to notice that there are special cases of writing to the TCNTn Register when the counter is counting that will give unpredictable results. The special cases are described in the sections where they are of importance.

Depending on the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock ($clk_{Tn}$). The $clk_{Tn}$ can be generated from an external or internal clock source, selected by the Clock Select bits (CSn2:0). When no clock source is selected (CSn2:0 = 0) the timer is stopped. However, the TCNTn value can be accessed by the CPU, independent of whether $clk_{Tn}$ is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the Waveform Generation mode bits (WGMn3:0) located in the Timer/Counter Control Registers A and B (TCCRnA and TCCRnB). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare outputs OCnx. For more details

about advanced counting sequences and waveform generation, see “Modes of Operation” on page 121.

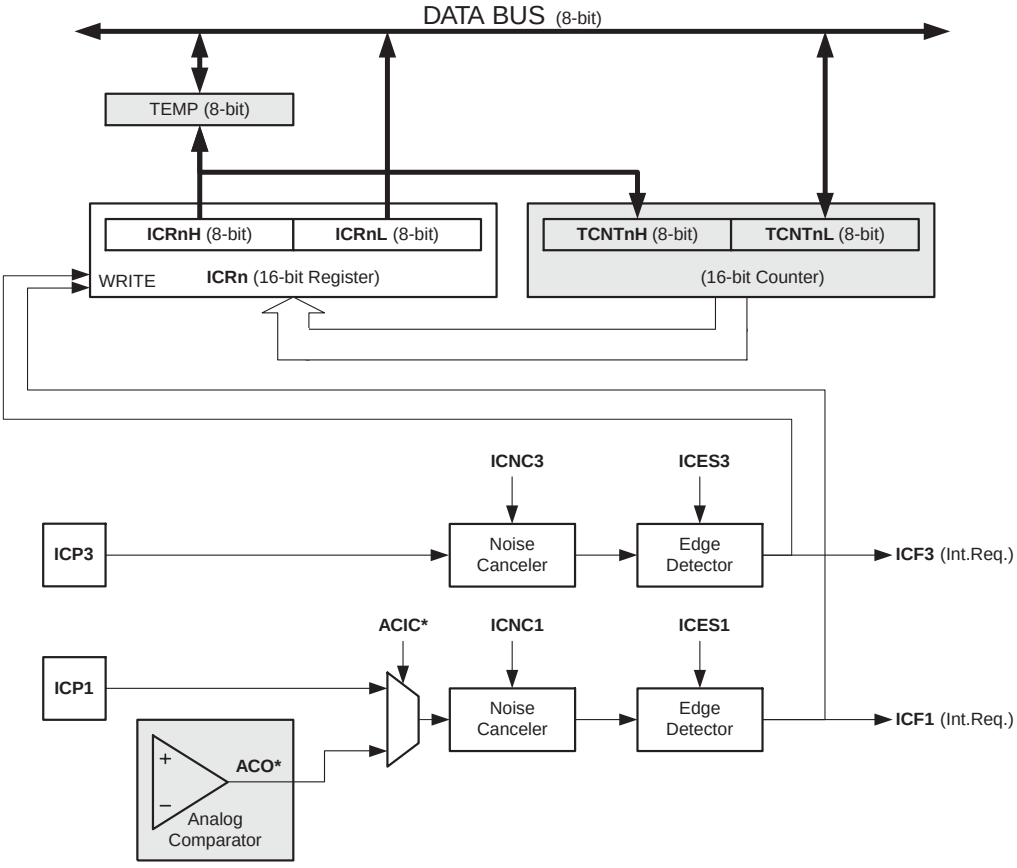
The Timer/Counter Overflow Flag (TOVn) is set according to the mode of operation selected by the WGMn3:0 bits. TOVn can be used for generating a CPU interrupt.

## Input Capture Unit

The Timer/Counter incorporates an Input Capture unit that can capture external events and give them a time-stamp indicating time of occurrence. The external signal indicating an event, or multiple events, can be applied via the ICPn pin or alternatively, via the analog-comparator unit. The time-stamps can then be used to calculate frequency, duty-cycle, and other features of the signal applied. Alternatively the time-stamps can be used for creating a log of the events.

The Input Capture unit is illustrated by the block diagram shown in Figure 50. The elements of the block diagram that are not directly a part of the Input Capture unit are gray shaded.

**Figure 50.** Input Capture Unit Block Diagram




Note: The Analog Comparator Output (ACO) can only trigger the Timer/Counter1 IC Unit– not Timer/Counter3.

When a change of the logic level (an event) occurs on the Input Capture pin (ICPn), alternatively on the Analog Comparator output (ACO), and this change confirms to the setting of the edge detector, a capture will be triggered. When a capture is triggered, the 16-bit value of the counter (TCNTn) is written to the Input Capture Register (ICRn). The Input Capture Flag (ICFn) is set at the same system clock as the TCNTn value is copied

into ICRn Register. If enabled (ICIEn = 1), the Input Capture Flag generates an Input Capture interrupt. The ICFn flag is automatically cleared when the interrupt is executed. Alternatively the ICFn flag can be cleared by software by writing a logical one to its I/O bit location.

Reading the 16-bit value in the Input Capture Register (ICRn) is done by first reading the low byte (ICRnL) and then the high byte (ICRnH). When the low byte is read the high byte is copied into the high byte temporary register (TEMP). When the CPU reads the ICRnH I/O location it will access the TEMP Register.

The ICRn Register can only be written when using a Waveform Generation mode that utilizes the ICRn Register for defining the counter's TOP value. In these cases the Waveform Generation mode (WGMn3:0) bits must be set before the TOP value can be written to the ICRn Register. When writing the ICRn Register the high byte must be written to the ICRnH I/O location before the low byte is written to ICRnL.

For more information on how to access the 16-bit registers refer to "Accessing 16-bit Registers" on page 111.

**Input Capture Trigger Source**

The main trigger source for the Input Capture unit is the Input Capture pin (ICPn). Only Timer/Counter1 can alternatively use the Analog Comparator output as trigger source for the Input Capture unit. The Analog Comparator is selected as trigger source by setting the Analog Comparator Input Capture (ACIC) bit in the Analog Comparator Control and Status Register (ACSR). Be aware that changing trigger source can trigger a capture. The Input Capture Flag must therefore be cleared after the change.

Both the Input Capture pin (ICPn) and the Analog Comparator output (ACO) inputs are sampled using the same technique as for the Tn pin (Figure 35 on page 91). The edge detector is also identical. However, when the noise canceler is enabled, additional logic is inserted before the edge detector, which increases the delay by four system clock cycles. Note that the input of the noise canceler and edge detector is always enabled unless the Timer/Counter is set in a Waveform Generation mode that uses ICRn to define TOP.

An Input Capture can be triggered by software by controlling the port of the ICPn pin.

**Noise Canceler**

The noise canceler improves noise immunity by using a simple digital filtering scheme. The noise canceler input is monitored over four samples, and all four must be equal for changing the output that in turn is used by the edge detector.

The noise canceler is enabled by setting the Input Capture Noise Canceler (ICNCn) bit in Timer/Counter Control Register B (TCCRnB). When enabled the noise canceler introduces additional four system clock cycles of delay from a change applied to the input, to the update of the ICRn Register. The noise canceler uses the system clock and is therefore not affected by the prescaler.

**Using the Input Capture Unit**

The main challenge when using the Input Capture unit is to assign enough processor capacity for handling the incoming events. The time between two events is critical. If the processor has not read the captured value in the ICRn Register before the next event occurs, the ICRn will be overwritten with a new value. In this case the result of the capture will be incorrect.

When using the Input Capture interrupt, the ICRn Register should be read as early in the interrupt handler routine as possible. Even though the Input Capture interrupt has relatively high priority, the maximum interrupt response time is dependent on the maximum number of clock cycles it takes to handle any of the other interrupt requests.

Using the Input Capture unit in any mode of operation when the TOP value (resolution) is actively changed during operation, is not recommended.

Measurement of an external signal's duty cycle requires that the trigger edge is changed after each capture. Changing the edge sensing must be done as early as possible after the ICRn Register has been read. After a change of the edge, the Input Capture Flag (ICFn) must be cleared by software (writing a logical one to the I/O bit location). For measuring frequency only, the clearing of the ICFn flag is not required (if an interrupt handler is used).

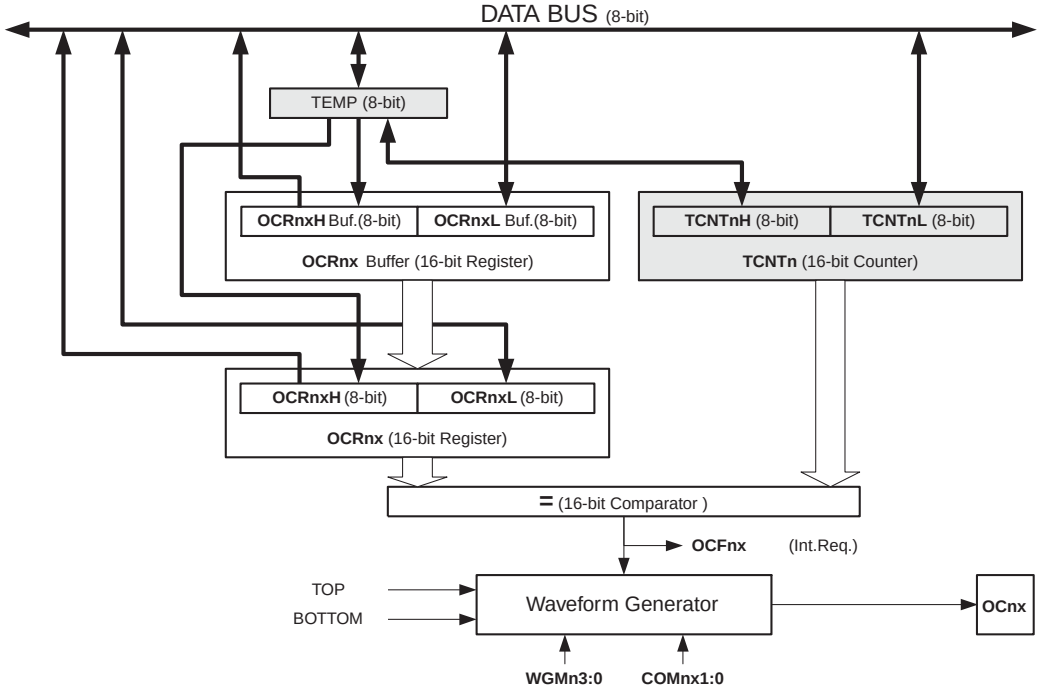
## Output Compare Units

The 16-bit comparator continuously compares TCNTn with the Output Compare Register (OCRnx). If TCNT equals OCRnx the comparator signals a match. A match will set the Output Compare Flag (OCFnx) at the next timer clock cycle. If enabled (OCIEnx = 1), the Output Compare Flag generates an Output Compare interrupt. The OCFnx flag is automatically cleared when the interrupt is executed. Alternatively the OCFnx flag can be cleared by software by writing a logical one to its I/O bit location. The Waveform Generator uses the match signal to generate an output according to operating mode set by the Waveform Generation mode (WGMn3:0) bits and Compare Output mode (COMnx1:0) bits. The TOP and BOTTOM signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation (See "Modes of Operation" on page 121 )

A special feature of Output Compare unit A allows it to define the Timer/Counter TOP value (i.e., counter resolution). In addition to the counter resolution, the TOP value defines the period time for waveforms generated by the Waveform Generator.

Figure 51 shows a block diagram of the Output Compare unit. The elements of the block diagram that are not directly a part of the Output Compare unit are gray shaded.

**Figure 51.** Output Compare Unit, Block Diagram



The OCRnx Register is double buffered when using any of the twelve Pulse Width Modulation (PWM) modes. For the Normal and Clear Timer on Compare (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCRnx Compare Register to either TOP or BOTTOM of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCRnx Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCRnx Buffer Register, and if double buffering is disabled the CPU will access the OCRnx directly. The content of the OCRnx (Buffer or Compare) Register is only changed by a write operation (the Timer/Counter does not update this register automatically as the TCNT1 and ICRn Register). Therefore OCRnx is not read via the high byte temporary register (TEMP). However, it is a good practice to read the low byte first as when accessing other 16-bit registers. Writing the OCRnx Registers must be done via the TEMP Register since the compare of all 16 bits is done continuously. The high byte (OCRnxH) has to be written first. When the high byte I/O location is written by the CPU, the TEMP Register will be updated by the value written. Then when the low byte (OCRnxL) is written to the lower eight bits, the high byte will be copied into the upper 8-bits of either the OCRnx buffer or OCRnx Compare Register in the same system clock cycle.

For more information of how to access the 16-bit registers refer to “Accessing 16-bit Registers” on page 111.

**Force Output Compare**

In non-PWM Waveform Generation modes, the match output of the comparator can be forced by writing a one to the Force Output Compare (FOCnx) bit. Forcing compare match will not set the OCFnx flag or reload/clear the timer, but the OCnx pin will be updated as if a real compare match had occurred (the COMnx1:0 bits settings define whether the OCnx pin is set, cleared or toggled).

**Compare Match Blocking by TCNTn Write**

All CPU writes to the TCNTn Register will block any compare match that occurs in the next timer clock cycle, even when the timer is stopped. This feature allows OCRnx to be initialized to the same value as TCNTn without triggering an interrupt when the Timer/Counter clock is enabled.

**Using the Output Compare Unit**

Since writing TCNTn in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNTn when using any of the Output Compare channels, independent of whether the Timer/Counter is running or not. If the value written to TCNTn equals the OCRnx value, the compare match will be missed, resulting in incorrect waveform generation. Do not write the TCNTn equal to TOP in PWM modes with variable TOP values. The compare match for the TOP will be ignored and the counter will continue to 0xFFFF. Similarly, do not write the TCNTn value equal to BOTTOM when the counter is downcounting.

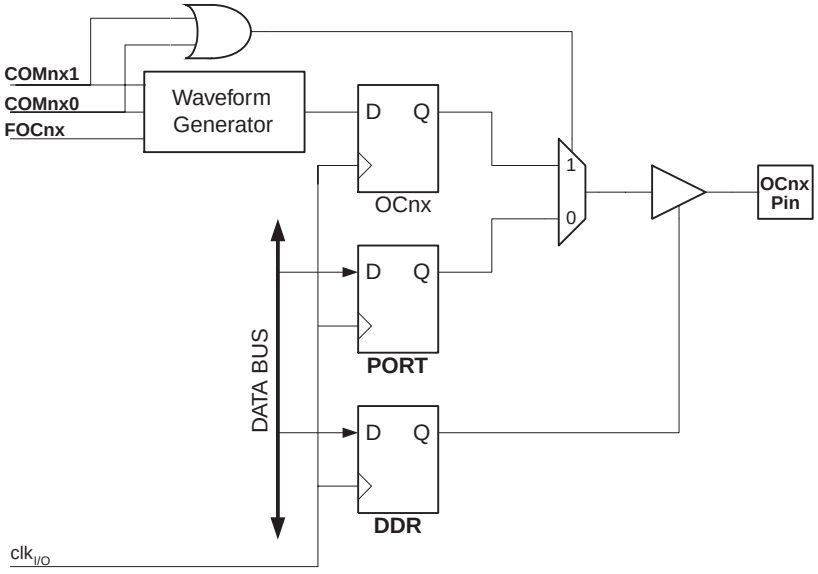
The setup of the OCnx should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OCnx value is to use the Force Output Compare (FOCnx) strobe bits in Normal mode. The OCnx Register keeps its value even when changing between Waveform Generation modes.

Be aware that the COMnx1:0 bits are not double buffered together with the compare value. Changing the COMnx1:0 bits will take effect immediately.

## Compare Match Output Unit

The Compare Output mode (COMnx1:0) bits have two functions. The Waveform Generator uses the COMnx1:0 bits for defining the Output Compare (OCnx) state at the next compare match. Secondly the COMnx1:0 bits control the OCnx pin output source. Figure 52 shows a simplified schematic of the logic affected by the COMnx1:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O port control registers (DDR and PORT) that are affected by the COMnx1:0 bits are shown. When referring to the OCnx state, the reference is for the internal OCnx Register, not the OCnx pin. If a system reset occur, the OCnx Register is reset to “0”.

**Figure 52.** Compare Match Output Unit, Schematic




### Compare Output Function

The general I/O port function is overridden by the Output Compare (OCnx) from the Waveform Generator if either of the COMnx1:0 bits are set. However, the OCnx pin direction (input or output) is still controlled by the Data Direction Register (DDR) for the port pin. The Data Direction Register bit for the OCnx pin (DDR_OCnx) must be set as output before the OCnx value is visible on the pin. The port override function is generally independent of the Waveform Generation mode, but there are some exceptions. Refer to Table 60, Table 61 and Table 62 for details.

The design of the Output Compare pin logic allows initialization of the OCnx state before the output is enabled. Note that some COMnx1:0 bit settings are reserved for certain modes of operation. See “16-bit Timer/Counter Register Description” on page 131

The COMnx1:0 bits have no effect on the Input Capture unit.

### Compare Output Mode and Waveform Generation

The Waveform Generator uses the COMnx1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COMnx1:0 = 0 tells the Waveform Generator that no action on the OCnx Register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 60 on page 131. For fast PWM mode refer to Table 61 on page 132, and for phase correct and phase and frequency correct PWM refer to Table 62 on page 132.

A change of the COMnx1:0 bits state will have effect at the first compare match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOCnx strobe bits.

## Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the Waveform Generation mode (WGMn3:0) and Compare Output mode (COMnx1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COMnx1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COMnx1:0 bits control whether the output should be set, cleared or toggle at a compare match (See “Compare Match Output Unit” on page 120 )

For detailed timing information refer to “Timer/Counter Timing Diagrams” on page 128.

### Normal Mode

The simplest mode of operation is the Normal mode (WGMn3:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 16-bit value (MAX = 0xFFFF) and then restarts from the BOTTOM (0x0000). In normal operation the Timer/Counter Overflow Flag (TOVn) will be set in the same timer clock cycle as the TCNTn becomes zero. The TOVn flag in this case behaves like a 17th bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOVn flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

The Input Capture unit is easy to use in Normal mode. However, observe that the maximum interval between the external events must not exceed the resolution of the counter. If the interval between events are too long, the timer overflow interrupt or the prescaler must be used to extend the resolution for the capture unit.

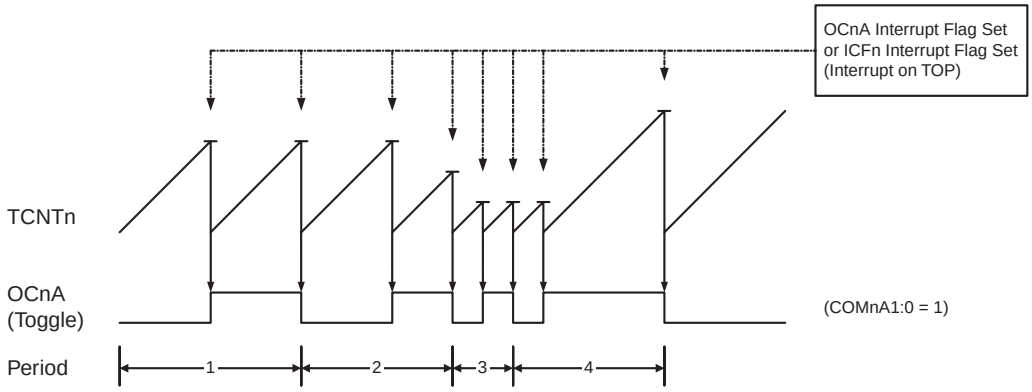
The Output Compare units can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

### Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGMn3:0 = 4 or 12), the OCRnA or ICRn Register are used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNTn) matches either the OCRnA (WGMn3:0 = 4) or the ICRn (WGMn3:0 = 12). The OCRnA or ICRn define the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 53. The counter value (TCNTn) increases until a compare match occurs with either OCRnA or ICRn, and then counter (TCNTn) is cleared.

**Figure 53.** CTC Mode, Timing Diagram




An interrupt can be generated at each time the counter value reaches the TOP value by either using the OCFnA or ICFn flag according to the register used to define the TOP value. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing the TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCRnA or ICRn is lower than the current value of TCNTn, the counter will miss the compare match. The counter will then have to count to its maximum value (0xFFFF) and wrap around starting at 0x0000 before the compare match can occur. In many cases this feature is not desirable. An alternative will then be to use the fast PWM mode using OCRnA for defining TOP (WGMn3:0 = 15) since the OCRnA then will be double buffered.

For generating a waveform output in CTC mode, the OCnA output can be set to toggle its logical level on each compare match by setting the Compare Output mode bits to toggle mode (COMnA1:0 = 1). The OCnA value will not be visible on the port pin unless the data direction for the pin is set to output (DDR_OCnA = 1). The waveform generated will have a maximum frequency of $f_{OCnA} = f_{clk_I/O}/2$ when OCRnA is set to zero (0x0000). The waveform frequency is defined by the following equation:

$$f_{OCnA} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRnA)}$$

The N variable represents the prescaler factor (1, 8, 64, 256, or 1024).

As for the Normal mode of operation, the TOVn flag is set in the same timer clock cycle that the counter counts from MAX to 0x0000.

**Fast PWM Mode**

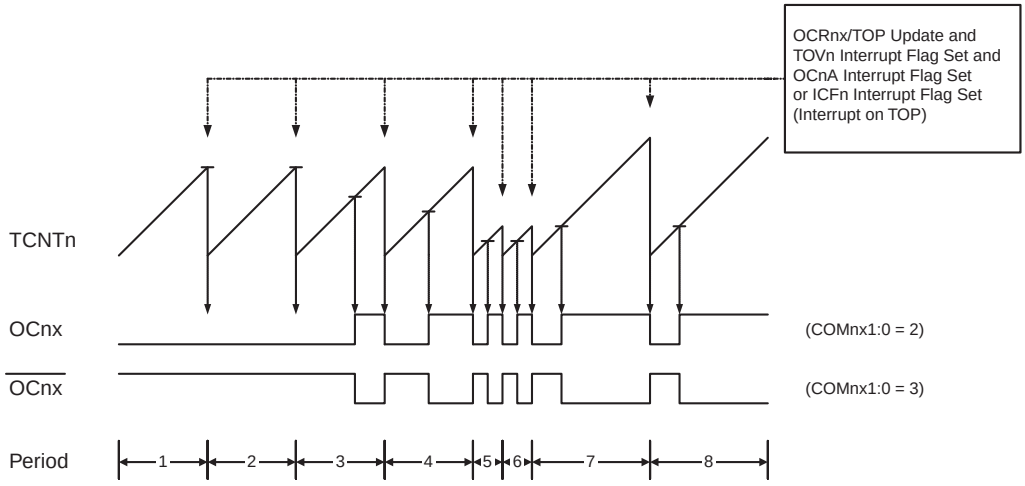
The fast Pulse Width Modulation or fast PWM mode (WGMn3:0 = 5, 6, 7, 14, or 15) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OCnx) is set on the compare match between TCNTn and OCRnx, and cleared at TOP. In inverting Compare Output mode output is cleared on compare match and set at TOP. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct and phase and frequency correct PWM modes that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), hence reduces total system cost.

The PWM resolution for fast PWM can be fixed to 8-, 9-, or 10-bit, or defined by either ICRn or OCRnA. The minimum resolution allowed is 2-bit (ICRn or OCRnA set to 0x0003), and the maximum resolution is 16-bit (ICRn or OCRnA set to MAX). The PWM resolution in bits can be calculated by using the following equation:

$$R_{FPWM} = \frac{\log(TOP+1)}{\log(2)}$$

In fast PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGMn3:0 = 5, 6, or 7), the value in ICRn (WGMn3:0 = 14), or the value in OCRnA (WGMn3:0 = 15). The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 54. The figure shows fast PWM mode when OCRnA or ICRn is used to define TOP. The TCNTn value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNTn slopes represent compare matches between OCRnx and TCNTn. The OCnx interrupt flag will be set when a compare match occurs.

**Figure 54.** Fast PWM Mode, Timing Diagram




The Timer/Counter Overflow Flag (TOVn) is set each time the counter reaches TOP. In addition the OCnA or ICFn flag is set at the same timer clock cycle as TOVn is set when either OCRnA or ICRn is used for defining the TOP value. If one of the interrupts are enabled, the interrupt handler routine can be used for updating the TOP and compare values.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNTn and the OCRnx. Note that when using fixed TOP values the unused bits are masked to zero when any of the OCRnx Registers are written.

The procedure for updating ICRn differs from updating OCRnA when used for defining the TOP value. The ICRn Register is not double buffered. This means that if ICRn is changed to a low value when the counter is running with none or a low prescaler value, there is a risk that the new ICRn value written is lower than the current value of TCNTn. The result will then be that the counter will miss the compare match at the TOP value.

The counter will then have to count to the MAX value (0xFFFF) and wrap around starting at 0x0000 before the compare match can occur. The OCRnA Register however, is double buffered. This feature allows the OCRnA I/O location to be written anytime. When the OCRnA I/O location is written the value written will be put into the OCRnA Buffer Register. The OCRnA Compare Register will then be updated with the value in the Buffer Register at the next timer clock cycle the TCNTn matches TOP. The update is done at the same timer clock cycle as the TCNTn is cleared and the TOVn flag is set.

Using the ICRn Register for defining TOP works well when using fixed TOP values. By using ICRn, the OCRnA Register is free to be used for generating a PWM output on OCnA. However, if the base PWM frequency is actively changed (by changing the TOP value), using the OCRnA as TOP is clearly a better choice due to its double buffer feature.

In fast PWM mode, the compare units allow generation of PWM waveforms on the OCnx pins. Setting the COMnx1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COMnx1:0 to three (see Table  on page 132). The actual OCnx value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OCnx). The PWM waveform is generated by setting (or clearing) the OCnx Register at the compare match between OCRnx and TCNTn, and clearing (or setting) the OCnx Register at the timer clock cycle the counter is cleared (changes from TOP to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{clk_I/O}}{N \cdot (1 + TOP)}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCRnx Register represents special cases when generating a PWM waveform output in the fast PWM mode. If the OCRnx is set equal to BOTTOM (0x0000) the output will be a narrow spike for each TOP+1 timer clock cycle. Setting the OCRnx equal to TOP will result in a constant high or low output (depending on the polarity of the output set by the COMnx1:0 bits.)

A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OCnA to toggle its logical level on each compare match (COMnA1:0 = 1). The waveform generated will have a maximum frequency of $f_{OCnA} = f_{clk_I/O}/2$ when OCRnA is set to zero (0x0000). This feature is similar to the OCnA toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

**Phase Correct PWM Mode**

The phase correct Pulse Width Modulation or phase correct PWM mode (WGMn3:0 = 1, 2, 3, 10, or 11) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is, like the phase and frequency correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OCnx) is cleared on the compare match between TCNTn and OCRnx while upcounting, and set on the compare match while downcounting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

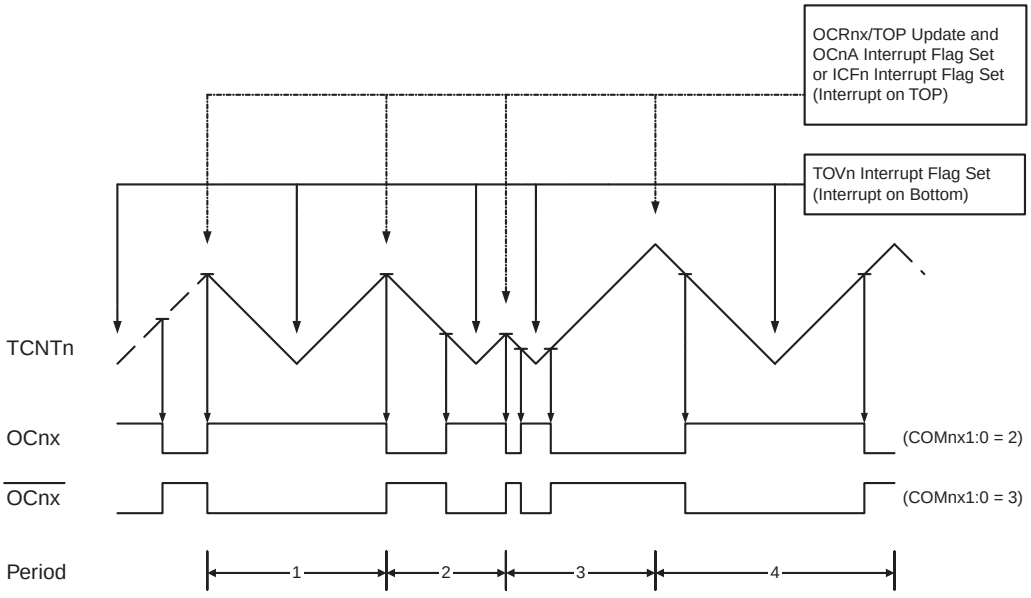
The PWM resolution for the phase correct PWM mode can be fixed to 8-, 9-, or 10-bit, or defined by either ICRn or OCRnA. The minimum resolution allowed is 2-bit (ICRn or

OCRnA set to 0x0003), and the maximum resolution is 16-bit (ICRn or OCRnA set to MAX). The PWM resolution in bits can be calculated by using the following equation:

$$R_{PCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In phase correct PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGMn3:0 = 1, 2, or 3), the value in ICRn (WGMn3:0 = 10), or the value in OCRnA (WGMn3:0 = 11). The counter has then reached the TOP and changes the count direction. The TCNTn value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 55. The figure shows phase correct PWM mode when OCRnA or ICRn is used to define TOP. The TCNTn value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNTn slopes represent compare matches between OCRnx and TCNTn. The OCnx interrupt flag will be set when a compare match occurs.

**Figure 55.** Phase Correct PWM Mode, Timing Diagram




The Timer/Counter Overflow Flag (TOVn) is set each time the counter reaches BOTTOM. When either OCRnA or ICRn is used for defining the TOP value, the OCnA or ICFn flag is set accordingly at the same timer clock cycle as the OCRnx Registers are updated with the double buffer value (at TOP). The interrupt flags can be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNTn and the OCRnx. Note that when using fixed TOP values, the unused bits are masked to zero when any of the OCRnx Registers are written. As the third period shown in Figure 55 illustrates, changing the TOP actively while the Timer/Counter is running in the phase correct mode can result in an unsymmetrical output. The reason for this can be found in the time of update of the OCRnx Register. Since the OCRnx update occurs at TOP, the PWM period starts and ends at TOP. This implies that the length of the fall-

ing slope is determined by the previous TOP value, while the length of the rising slope is determined by the new TOP value. When these two values differ the two slopes of the period will differ in length. The difference in length gives the unsymmetrical result on the output.

It is recommended to use the phase and frequency correct mode instead of the phase correct mode when changing the TOP value while the Timer/Counter is running. When using a static TOP value there are practically no differences between the two modes of operation.

In phase correct PWM mode, the compare units allow generation of PWM waveforms on the OCnx pins. Setting the COMnx1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COMnx1:0 to three (See Table on page 132). The actual OCnx value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OCnx). The PWM waveform is generated by setting (or clearing) the OCnx Register at the compare match between OCRnx and TCNTn when the counter increments, and clearing (or setting) the OCnx Register at compare match between OCRnx and TCNTn when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnxPCPWM} = \frac{f_{clk_I/O}}{2 \cdot N \cdot TOP}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCRnx Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCRnx is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

**Phase and Frequency Correct PWM Mode**

The phase and frequency correct Pulse Width Modulation, or phase and frequency correct PWM mode (WGMn3:0 = 8 or 9) provides a high resolution phase and frequency correct PWM waveform generation option. The phase and frequency correct PWM mode is, like the phase correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OCnx) is cleared on the compare match between TCNTn and OCRnx while upcounting, and set on the compare match while downcounting. In inverting Compare Output mode, the operation is inverted. The dual-slope operation gives a lower maximum operation frequency compared to the single-slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

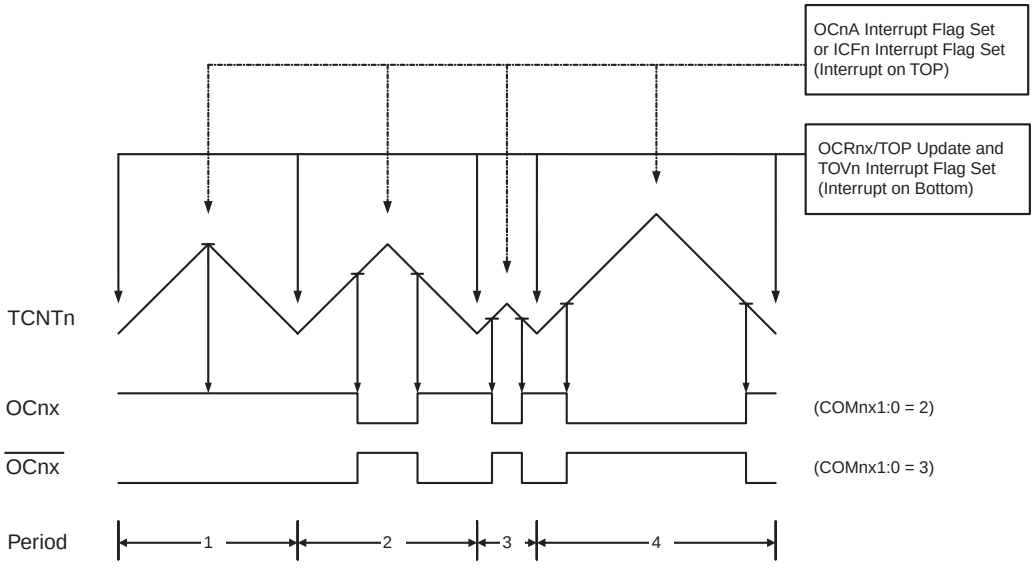
The main difference between the phase correct, and the phase and frequency correct PWM mode is the time the OCRnx Register is updated by the OCRnx Buffer Register, (see Figure 55 and Figure 56).

The PWM resolution for the phase and frequency correct PWM mode can be defined by either ICRn or OCRnA. The minimum resolution allowed is 2-bit (ICRn or OCRnA set to 0x0003), and the maximum resolution is 16-bit (ICRn or OCRnA set to MAX). The PWM resolution in bits can be calculated using the following equation:

$$R_{PFCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In phase and frequency correct PWM mode the counter is incremented until the counter value matches either the value in ICRn (WGMn3:0 = 8), or the value in OCRnA (WGMn3:0 = 9). The counter has then reached the TOP and changes the count direction. The TCNTn value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct and frequency correct PWM mode is shown on Figure 56. The figure shows phase and frequency correct PWM mode when OCRnA or ICRn is used to define TOP. The TCNTn value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNTn slopes represent compare matches between OCRnx and TCNTn. The OCnx interrupt flag will be set when a compare match occurs.

**Figure 56.** Phase and Frequency Correct PWM Mode, Timing Diagram




The Timer/Counter Overflow Flag (TOVn) is set at the same timer clock cycle as the OCRnx Registers are updated with the double buffer value (at BOTTOM). When either OCRnA or ICRn is used for defining the TOP value, the OCnA or ICFn flag set when TCNTn has reached TOP. The interrupt flags can then be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNTn and the OCRnx.

As Figure 56 shows the output generated is, in contrast to the phase correct mode, symmetrical in all periods. Since the OCRnx Registers are updated at BOTTOM, the length of the rising and the falling slopes will always be equal. This gives symmetrical output pulses and is therefore frequency correct.

Using the ICRn Register for defining TOP works well when using fixed TOP values. By using ICRn, the OCRnA Register is free to be used for generating a PWM output on OCnA. However, if the base PWM frequency is actively changed by changing the TOP value, using the OCRnA as TOP is clearly a better choice due to its double buffer feature.

In phase and frequency correct PWM mode, the compare units allow generation of PWM waveforms on the OCnx pins. Setting the COMnx1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COMnx1:0 to three (See Table  on page 132). The actual OCnx value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OCnx). The PWM waveform is generated by setting (or clearing) the OCnx Register at the compare match between OCRnx and TCNTn when the counter increments, and clearing (or setting) the OCnx Register at compare match between OCRnx and TCNTn when the counter decrements. The PWM frequency for the output when using phase and frequency correct PWM can be calculated by the following equation:

$$f_{OCnxPFCPWM} = \frac{f_{clk_I/O}}{2 \cdot N \cdot TOP}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCRnx Register represents special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCRnx is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be set to high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

## Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock ($clk_{Tn}$) is therefore shown as a clock enable signal in the following figures. The figures include information on when interrupt flags are set, and when the OCRnx Register is updated with the OCRnx buffer value (only for modes utilizing double buffering). Figure 57 shows a timing diagram for the setting of OCFnx.

**Figure 57.** Timer/Counter Timing Diagram, Setting of OCFnx, no Prescaling

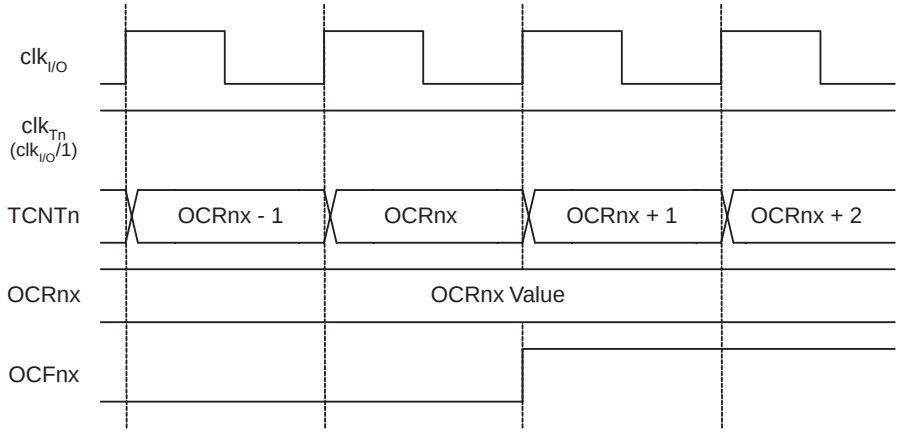



Figure 58 shows the same timing data, but with the prescaler enabled.

**Figure 58.** Timer/Counter Timing Diagram, Setting of OCFnx, with Prescaler ($f_{clk_I/O}/8$)

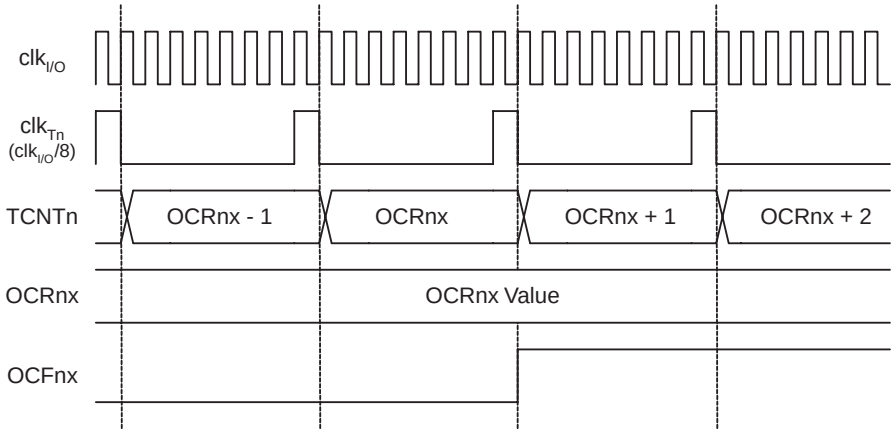



Figure 59 shows the count sequence close to TOP in various modes. When using phase and frequency correct PWM mode the OCRnx Register is updated at BOTTOM. The timing diagrams will be the same, but TOP should be replaced by BOTTOM, TOP-1 by BOTTOM+1 and so on. The same renaming applies for modes that set the TOVn flag at BOTTOM.

**Figure 59.** Timer/Counter Timing Diagram, no Prescaling

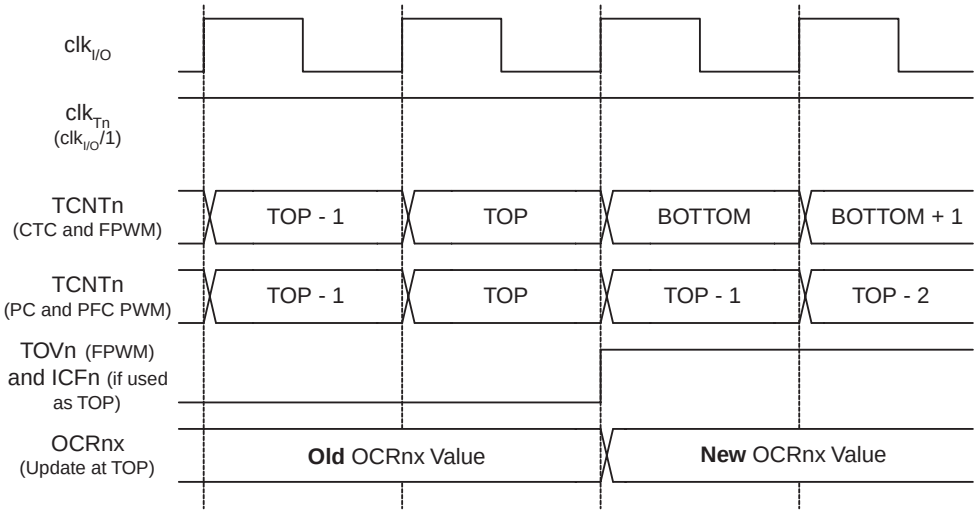
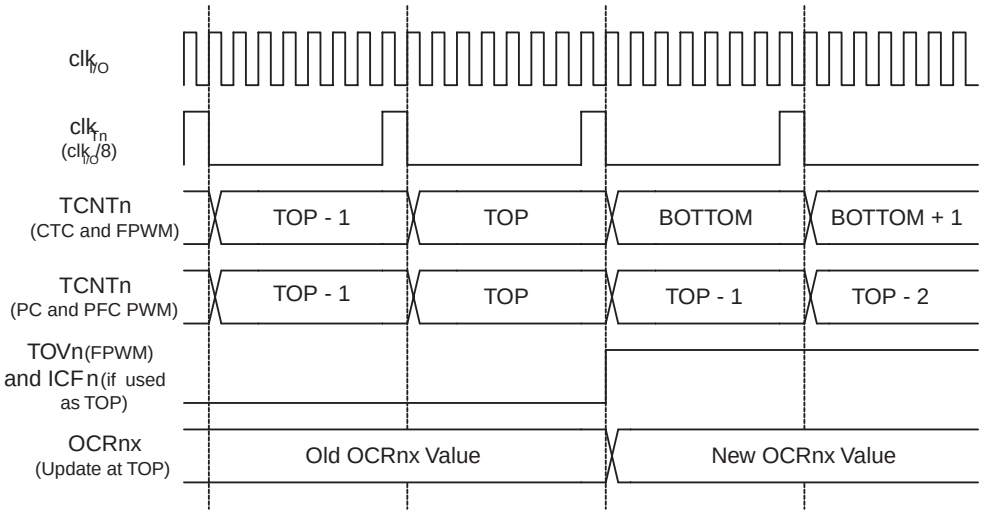



Figure 60 shows the same timing data, but with the prescaler enabled.

**Figure 60.** Timer/Counter Timing Diagram, with Prescaler ($f_{clk_I/O}/8$)



## 16-bit Timer/Counter Register Description

### Timer/Counter1 Control Register A – TCCR1A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | COM1A1 | COM1A0 | COM1B1 | COM1B0 | COM1C1 | COM1C0 | WGM11 | WGM10 | TCCR1A |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Timer/Counter3 Control Register A – TCCR3A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | COM3A1 | COM3A0 | COM3B1 | COM3B0 | COM3C1 | COM3C0 | WGM31 | WGM30 | TCCR3A |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7:6 – COMnA1:0: Compare Output Mode for Channel A**
- **Bit 5:4 – COMnB1:0: Compare Output Mode for Channel B**
- **Bit 3:2 – COMnC1:0: Compare Output Mode for Channel C**

The COMnA1:0, COMnB1:0 and COMnC1:0 control the Output Compare pins (OCnA, OCnB and OCnC respectively) behavior. If one or both of the COMnA1:0 bits are written to one, the OCnA output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COMnB1:0 bit are written to one, the OCnB output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COMnC1:0 bit are written to one, the OCnC output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OCnA, OCnB or OCnC pin must be set in order to enable the output driver.

When the OCnA, OCnB or OCnC is connected to the pin, the function of the COMnx1:0 bits is dependent of the WGMn3:0 bits setting. Table 60 shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to a Normal or a CTC mode (non-PWM).

**Table 60.** Compare Output Mode, non-PWM

| COMnA1/COMnB1/ COMnC1 | COMnA0/COMnB0/ COMnC0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OCnA/OCnB/OCnC disconnected. |
| 0 | 1 | Toggle OCnA/OCnB/OCnC on Compare Match. |
| 1 | 0 | Clear OCnA/OCnB/OCnC on Compare Match (Set output to low level). |
| 1 | 1 | Set OCnA/OCnB/OCnC on Compare Match (Set output to high level). |

Table 61 shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to the fast PWM mode.

**Table 61.** Compare Output Mode, Fast PWM(1)

| COMnA1/COMnB1/ COMnC1 | COMnA0/COMnB0/ COMnC0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OCnA/OCnB/OCnC disconnected. |
| 0 | 1 | WGMn3=0: Normal port operation, OCnA/OCnB/OCnC disconnected.<br>WGMn3=1: Toggle OCnA on Compare Match, OCnB/OCnC reserved. |
| 1 | 0 | Clear OCnA/OCnB/OCnC on Compare Match<br>Set OCnA/OCnB/OCnC at TOP |
| 1 | 1 | Set OCnA/OCnB/OCnC on Compare Match<br>Clear OCnA/OCnB/OCnC at TOP |

Note: 1. A special case occurs when OCRnA/OCRnB/OCRnC equals TOP and COMnA1/COMnB1/COMnC1 is set. In this case the compare match is ignored, but the set or clear is done at TOP. See “Fast PWM Mode” on page 122 for more details.

Table 62 shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to the phase correct or the phase and frequency correct, PWM mode.

**Table 62.** Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM(1)

| COMnA1/COMnB1/ COMnC1 | COMnA0/COMnB0/ COMnC0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OCnA/OCnB/OCnC disconnected. |
| 0 | 1 | WGMn3=0: Normal port operation, OCnA/OCnB/OCnC disconnected.<br>WGMn3=1: Toggle OCnA on Compare Match, OCnB/OCnC reserved. |
| 1 | 0 | Clear OCnA/OCnB/OCnC on Compare Match when up-counting.<br>Set OCnA/OCnB/OCnC on Compare Match when downcounting. |
| 1 | 1 | Set OCnA/OCnB/OCnC on Compare Match when up-counting.<br>Clear OCnA/OCnB/OCnC on Compare Match when downcounting. |

Note: 1. A special case occurs when OCnA/OCnB/OCnC equals TOP and COMnA1/COMnB1/COMnC1 is set. See “Phase Correct PWM Mode” on page 124 for more details.

- **Bit 1:0 – WGMn1:0: Waveform Generation Mode**

Combined with the WGMn3:2 bits found in the TCCRnB Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 63. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare

match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes. (See “Modes of Operation” on page 121 ).

**Table 63.** Waveform Generation Mode Bit Description(1)

| Mode | WGMn3 | WGMn2 (CTCn) | WGMn1 (PWMn1) | WGMn0 (PWMn0) | Timer/Counter Mode of Operation | TOP | Update of OCRnx at | TOVn Flag Set on |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | Normal | 0xFFFF | Immediate | MAX |
| 1 | 0 | 0 | 0 | 1 | PWM, Phase Correct, 8-bit | 0x00FF | TOP | BOTTOM |
| 2 | 0 | 0 | 1 | 0 | PWM, Phase Correct, 9-bit | 0x01FF | TOP | BOTTOM |
| 3 | 0 | 0 | 1 | 1 | PWM, Phase Correct, 10-bit | 0x03FF | TOP | BOTTOM |
| 4 | 0 | 1 | 0 | 0 | CTC | OCRnA | Immediate | MAX |
| 5 | 0 | 1 | 0 | 1 | Fast PWM, 8-bit | 0x00FF | TOP | TOP |
| 6 | 0 | 1 | 1 | 0 | Fast PWM, 9-bit | 0x01FF | TOP | TOP |
| 7 | 0 | 1 | 1 | 1 | Fast PWM, 10-bit | 0x03FF | TOP | TOP |
| 8 | 1 | 0 | 0 | 0 | PWM, Phase and Frequency Correct | ICRn | BOTTOM | BOTTOM |
| 9 | 1 | 0 | 0 | 1 | PWM, Phase and Frequency Correct | OCRnA | BOTTOM | BOTTOM |
| 10 | 1 | 0 | 1 | 0 | PWM, Phase Correct | ICRn | TOP | BOTTOM |
| 11 | 1 | 0 | 1 | 1 | PWM, Phase Correct | OCRnA | TOP | BOTTOM |
| 12 | 1 | 1 | 0 | 0 | CTC | ICRn | Immediate | MAX |
| 13 | 1 | 1 | 0 | 1 | (Reserved) | – | – | – |
| 14 | 1 | 1 | 1 | 0 | Fast PWM | ICRn | TOP | TOP |
| 15 | 1 | 1 | 1 | 1 | Fast PWM | OCRnA | TOP | TOP |

Note: 1. The CTCn and PWMn1:0 bit definition names are obsolete. Use the WGMn2:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

### Timer/Counter1 Control Register B – TCCR1B

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | ICNC1 | ICES1 | – | WGM13 | WGM12 | CS12 | CS11 | CS10 | TCCR1B |
| Read/Write | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Timer/Counter3 Control Register B – TCCR3B

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | ICNC3 | ICES3 | – | WGM33 | WGM32 | CS32 | CS31 | CS30 | TCCR3B |
| Read/Write | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – ICNCn: Input Capture Noise Canceler**

Setting this bit (to one) activates the Input Capture Noise Canceler. When the noise canceler is activated, the input from the Input Capture pin (ICPn) is filtered. The filter function requires four successive equal valued samples of the ICPn pin for changing its output. The Input Capture is therefore delayed by four Oscillator cycles when the noise canceler is enabled.

- **Bit 6 – ICESn: Input Capture Edge Select**

This bit selects which edge on the Input Capture pin (ICPn) that is used to trigger a capture event. When the ICESn bit is written to zero, a falling (negative) edge is used as trigger, and when the ICESn bit is written to one, a rising (positive) edge will trigger the capture.

When a capture is triggered according to the ICESn setting, the counter value is copied into the Input Capture Register (ICRn). The event will also set the Input Capture Flag (ICFn), and this can be used to cause an Input Capture Interrupt, if this interrupt is enabled.

When the ICRn is used as TOP value (see description of the WGMn3:0 bits located in the TCCRnA and the TCCRnB Register), the ICPn is disconnected and consequently the Input Capture function is disabled.

- **Bit 5 – Reserved Bit**

This bit is reserved for future use. For ensuring compatibility with future devices, this bit must be written to zero when TCCRnB is written.

- **Bit 4:3 – WGMn3:2: Waveform Generation Mode**

See TCCRnA Register description.

- **Bit 2:0 – CSn2:0: Clock Select**

The three Clock Select bits select the clock source to be used by the Timer/Counter, see Figure 57 and Figure 58.

**Table 64.** Clock Select Bit Description

| CSn2 | CSn1 | CSn0 | Description |
|---|---|---|---|
| 0 | 0 | 0 | No clock source (Timer/Counter stopped). |
| 0 | 0 | 1 | $clk_{I/O}$/1 (No prescaling) |
| 0 | 1 | 0 | $clk_{I/O}$/8 (From prescaler) |
| 0 | 1 | 1 | $clk_{I/O}$/64 (From prescaler) |
| 1 | 0 | 0 | $clk_{I/O}$/256 (From prescaler) |
| 1 | 0 | 1 | $clk_{I/O}$/1024 (From prescaler) |
| 1 | 1 | 0 | External clock source on Tn pin. Clock on falling edge. |
| 1 | 1 | 1 | External clock source on Tn pin. Clock on rising edge. |

If external pin modes are used for the Timer/Countern, transitions on the Tn pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

**Timer/Counter1 Control Register C – TCCR1C**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **FOC1A** | **FOC1B** | **FOC1C** | **–** | **–** | **–** | **–** | **–** | **TCCR1C** |
| Read/Write | R/W | R/W | R/W | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Timer/Counter3 Control Register C – TCCR3C

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | FOC3A | FOC3B | FOC3C | – | – | – | – | – | TCCR3C |
| Read/Write | R/W | R/W | R/W | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – FOCnA: Force Output Compare for Channel A**
- **Bit 6 – FOCnB: Force Output Compare for Channel B**
- **Bit 5 – FOCnC: Force Output Compare for Channel C**

The FOCnA/FOCnB/FOCnC bits are only active when the WGMn3:0 bits specifies a non-PWM mode. However, for ensuring compatibility with future devices, these bits must be set to zero when TCCRnA is written when operating in a PWM mode. When writing a logical one to the FOCnA/FOCnB/FOCnC bit, an immediate compare match is forced on the Waveform Generation unit. The OCnA/OCnB/OCnC output is changed according to its COMnx1:0 bits setting. Note that the FOCnA/FOCnB/FOCnC bits are implemented as strobes. Therefore it is the value present in the COMnx1:0 bits that determine the effect of the forced compare.

A FOCnA/FOCnB/FOCnC strobe will not generate any interrupt nor will it clear the timer in Clear Timer on Compare match (CTC) mode using OCRnA as TOP.

The FOCnA/FOCnB/FOCnC bits are always read as zero.

### Timer/Counter1 – TCNT1H and TCNT1L

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TCNT1[15:8] | | | | | | | | TCNT1H |
| | TCNT1[7:0] | | | | | | | | TCNT1L |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Timer/Counter3 – TCNT3H and TCNT3L

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TCNT3[15:8] | | | | | | | | TCNT3H |
| | TCNT3[7:0] | | | | | | | | TCNT3L |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

The two Timer/Counter I/O locations (TCNTnH and TCNTnL, combined TCNTn) give direct access, both for read and for write operations, to the Timer/Counter unit 16-bit counter. To ensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 111

Modifying the counter (TCNTn) while the counter is running introduces a risk of missing a compare match between TCNTn and one of the OCRnx Registers.

Writing to the TCNTn Register blocks (removes) the compare match on the following timer clock for all compare units.

**Output Compare Register A – OCR1AH and OCR1AL**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | OCR1A[15:8] | | | | | | | | OCR1AH |
| | OCR1A[7:0] | | | | | | | | OCR1AL |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Output Compare Register B – OCR1BH and OCR1BL**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | OCR1B[15:8] | | | | | | | | OCR1BH |
| | OCR1B[7:0] | | | | | | | | OCR1BL |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Output Compare Register C – OCR1CH and OCR1CL**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | OCR1C[15:8] | | | | | | | | OCR1CH |
| | OCR1C[7:0] | | | | | | | | OCR1CL |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Output Compare Register A – OCR3AH and OCR3AL**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | OCR3A[15:8] | | | | | | | | OCR3AH |
| | OCR3A[7:0] | | | | | | | | OCR3AL |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Output Compare Register B – OCR3BH and OCR3BL**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | OCR3B[15:8] | | | | | | | | OCR3BH |
| | OCR3B[7:0] | | | | | | | | OCR3BL |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Output Compare Register C – OCR3CH and OCR3CL**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | OCR3C[15:8] | | | | | | | | OCR3CH |
| | OCR3C[7:0] | | | | | | | | OCR3CL |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

The Output Compare Registers contain a 16-bit value that is continuously compared with the counter value (TCNTn). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OCnx pin.

The Output Compare Registers are 16-bit in size. To ensure that both the high and low bytes are written simultaneously when the CPU writes to these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 111

**Input Capture Register – ICR1H and ICR1L**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | ICR1[15:8] | | | | | | | | ICR1H |
| | ICR1[7:0] | | | | | | | | ICR1L |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Input Capture Register – ICR3H and ICR3L**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | ICR3[15:8] | | | | | | | | ICR3H |
| | ICR3[7:0] | | | | | | | | ICR3L |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

The Input Capture is updated with the counter (TCNTn) value each time an event occurs on the ICPn pin (or optionally on the Analog Comparator output for Timer/Counter1). The Input Capture can be used for defining the counter TOP value.

The Input Capture Register is 16-bit in size. To ensure that both the high and low bytes are read simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 111

**Timer/Counter1 Interrupt Mask Register – TIMSK1**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | ICIE1 | – | OCIE1C | OCIE1B | OCIE1A | TOIE1 | TIMSK1 |
| Read/Write | R | R | R/W | R | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Timer/Counter3 Interrupt Mask Register – TIMSK3**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | ICIE3 | – | OCIE3C | OCIE3B | OCIE3A | TOIE3 | TIMSK3 |
| Read/Write | R | R | R/W | R | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7..6 – Reserved Bits**

These bits are reserved for future use.

- **Bit 5 – ICIEn: Input Capture Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Input Capture interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 56 ) is executed when the ICFn flag, located in TIFRn, is set.

- **Bit 4 – Reserved Bit**

This bit is reserved for future use.

- **Bit 3 – OCIEnC: Output Compare C Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Output Compare C Match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 56 ) is executed when the OCFnC flag, located in TIFRn, is set.

- **Bit 2 – OCIEnB: Output Compare B Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Output Compare B Match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 56 ) is executed when the OCFnB flag, located in TIFRn, is set.

- **Bit 1 – OCIEnA: Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Output Compare A Match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 56 ) is executed when the OCFnA flag, located in TIFRn, is set.

- **Bit 0 – TOIEn: Timer/Counter Overflow Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Overflow interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 56 ) is executed when the TOVn flag, located in TIFRn, is set.

**Timer/Counter1 Interrupt Flag Register – TIFR1**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | ICF1 | – | OCF1C | OCF1B | OCF1A | TOV1 | TIFR1 |
| Read/Write | R | R | R/W | R | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Timer/Counter3 Interrupt Flag Register – TIFR3**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | ICF3 | – | OCF3C | OCF3B | OCF3A | TOV3 | TIFR3 |
| Read/Write | R | R | R/W | R | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7..6 – Reserved Bits**

These bits are reserved for future use.

- **Bit 5 – ICFn: Input Capture Flag**

This flag is set when a capture event occurs on the ICPn pin. When the Input Capture Register (ICRn) is set by the WGMn3:0 to be used as the TOP value, the ICFn flag is set when the counter reaches the TOP value.

ICFn is automatically cleared when the Input Capture Interrupt Vector is executed. Alternatively, ICFn can be cleared by writing a logic one to its bit location.

- **Bit 4 – Reserved Bit**

This bit is reserved for future use.

- **Bit 3 – OCFnC: Output Compare C Match Flag**

This flag is set in the timer clock cycle after the counter (TCNTn) value matches the Output Compare Register C (OCRnC).

Note that a Forced Output Compare (FOCnC) strobe will not set the OCFnC flag.

OCFnC is automatically cleared when the Output Compare Match C Interrupt Vector is executed. Alternatively, OCFnC can be cleared by writing a logic one to its bit location.

- **Bit 2 – OCFnB: Output Compare B Match Flag**

This flag is set in the timer clock cycle after the counter (TCNTn) value matches the Output Compare Register B (OCRnB).

Note that a Forced Output Compare (FOCnB) strobe will not set the OCFnB flag.

OCFnB is automatically cleared when the Output Compare Match B Interrupt Vector is executed. Alternatively, OCFnB can be cleared by writing a logic one to its bit location.

- **Bit 1 – OCFnA: Output Compare A Match Flag**

This flag is set in the timer clock cycle after the counter (TCNTn) value matches the Output Compare Register A (OCRnA).

Note that a Forced Output Compare (FOCnA) strobe will not set the OCFnA flag.

OCFnA is automatically cleared when the Output Compare Match A Interrupt Vector is executed. Alternatively, OCFnA can be cleared by writing a logic one to its bit location.

- **Bit 0 – TOVn: Timer/Counter Overflow Flag**

The setting of this flag is dependent of the WGMn3:0 bits setting. In Normal and CTC modes, the TOVn flag is set when the timer overflows. Refer to Table 63 on page 133 for the TOVn flag behavior when using another WGMn3:0 bit setting.

TOVn is automatically cleared when the Timer/Countern Overflow Interrupt Vector is executed. Alternatively, TOVn can be cleared by writing a logic one to its bit location.

# 8-bit Timer/Counter2 with PWM and Asynchronous Operation

Timer/Counter2 is a general purpose, single channel, 8-bit Timer/Counter module. The main features are:

## Features

- **Single Channel Counter**
- **Clear Timer on Compare Match (Auto Reload)**
- **Glitch-free, Phase Correct Pulse Width Modulator (PWM)**
- **Frequency Generator**
- **10-bit Clock Prescaler**
- **Overflow and Compare Match Interrupt Sources (TOV2 and OCF2A)**
- **Allows Clocking from External 32 kHz Watch Crystal Independent of the I/O Clock**

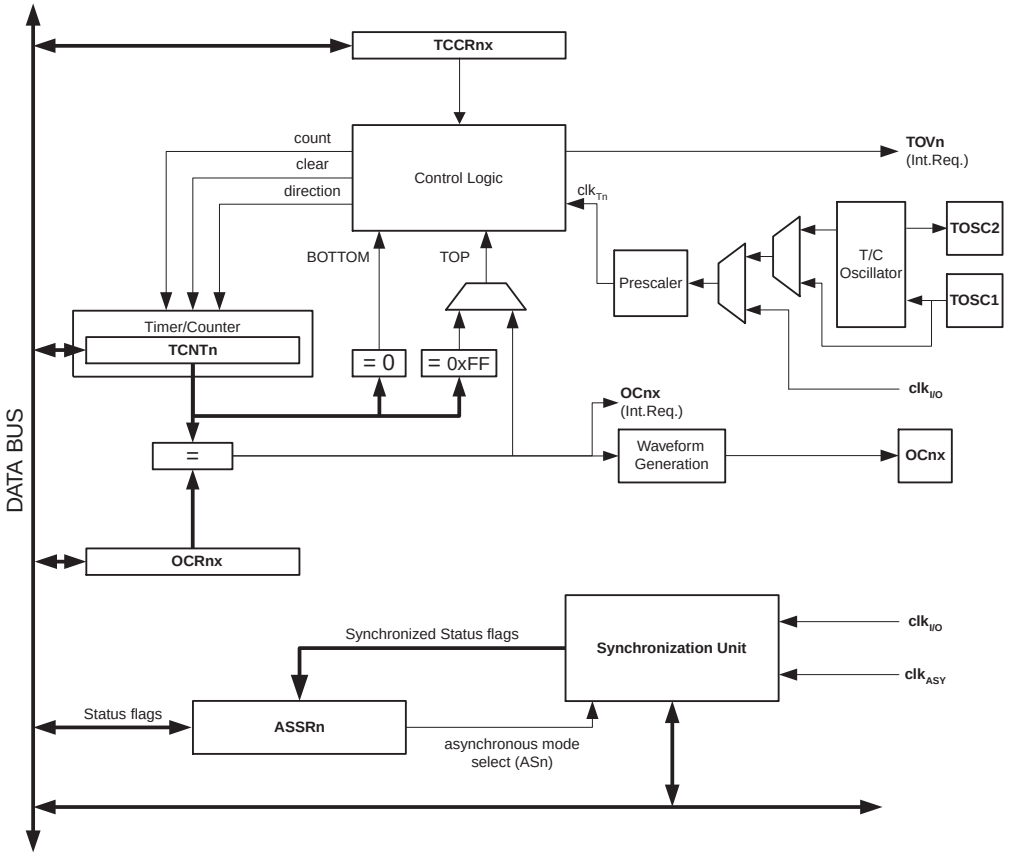
## Overview

Many register and bit references in this section are written in general form.

- A lower case “n” replaces the Timer/Counter number, in this case 2. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT2 for accessing Timer/Counter2 counter value and so on.
- A lower case “x” replaces the Output Compare unit channel, in this case A. However, when using the register or bit defines in a program, the precise form must be used, i.e., OCR2A for accessing Timer/Counter2 output compare channel A value and so on.

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 61. For the actual placement of I/O pins, refer to Figure 2 on page 4. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the “8-bit Timer/Counter Register Description” on page 151.

**Figure 61.** 8-bit Timer/Counter2 Block Diagram




The Timer/Counter (TCNT2) and Output Compare Register (OCR2A) are 8-bit registers. Interrupt request (shorten as Int.Req.) signals are all visible in the Timer Interrupt Flag Register (TIFR2). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK2). TIFR2 and TIMSK2 are not shown in the figure.

The Timer/Counter can be clocked internally, via the prescaler, or asynchronously clocked from the TOSC1/2 pins, as detailed later in this section. The asynchronous operation is controlled by the Asynchronous Status Register (ASSR). The Clock Select logic block controls which clock source the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the Clock Select logic is referred to as the timer clock ($clk_{T2}$).

The double buffered Output Compare Register (OCR2A) is compared with the Timer/Counter value at all times. The result of the compare can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pin (OC2A). See “Output Compare Unit” on page 143 for details. The compare match event will also set the compare flag (OCF2A) which can be used to generate an Output Compare interrupt request.

**Definitions**

The definitions in Table 65 are also used extensively throughout the section.

**Table 65.** Definitions

| | |
|---|---|
| BOTTOM | The counter reaches the BOTTOM when it becomes zero (0x00). |
| MAX | The counter reaches its MAXimum when it becomes 0xFF (decimal 255). |
| TOP | The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be the fixed value 0xFF (MAX) or the value stored in the OCR2A Register. The assignment is dependent on the mode of operation. |

## Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal synchronous or an external asynchronous clock source. The clock source is selected by the clock select logic which is controlled by the clock select (CS22:0) bits located in the Timer/Counter control register (TCCR2).The clock source $clk_{T2}$ is by default equal to the MCU clock, $clk_{I/O}$. When the AS2 bit in the ASSR Register is written to logic one, the clock source is taken from the Timer/Counter Oscillator connected to TOSC1 and TOSC2 or directly from TOSC1. For details on asynchronous operation, see “Asynchronous Status Register – ASSR” on page 154. For details on clock sources and prescaler, see “Timer/Counter2 Prescaler” on page 158.

## Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 62 shows a block diagram of the counter and its surrounding environment.

**Figure 62.** Counter Unit Block Diagram

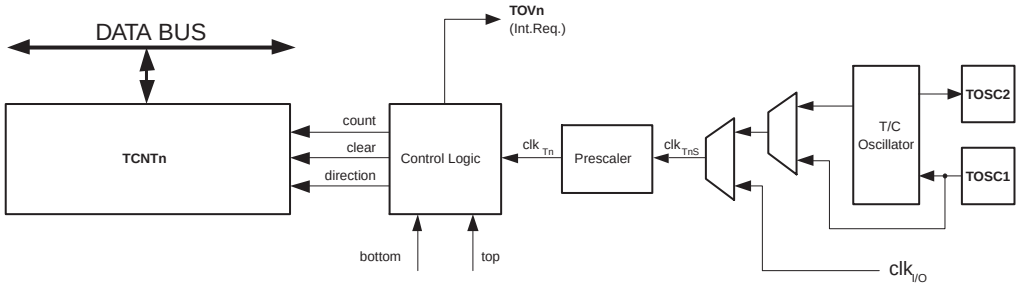



**Figure 63.**

Signal description (internal signals):

| | |
|---|---|
| **count** | Increment or decrement TCNT2 by 1. |
| **direction** | Selects between increment and decrement. |
| **clear** | Clear TCNT2 (set all bits to zero). |
| **$clk_{T2}$** | Timer/Counter clock. |
| **top** | Signalizes that TCNT2 has reached maximum value. |
| **bottom** | Signalizes that TCNT2 has reached minimum value (zero). |

Depending on the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock ($clk_{T2}$). $clk_{T2}$ can be generated from an external or internal clock source, selected by the Clock Select bits (CS22:0). When no clock source is selected (CS22:0 = 0) the timer is stopped. However, the TCNT2 value can be accessed

by the CPU, regardless of whether $clk_{T2}$ is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the WGM21 and WGM20 bits located in the Timer/Counter Control Register (TCCR2A). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare output OC2A. For more details about advanced counting sequences and waveform generation, see “Modes of Operation” on page 145.

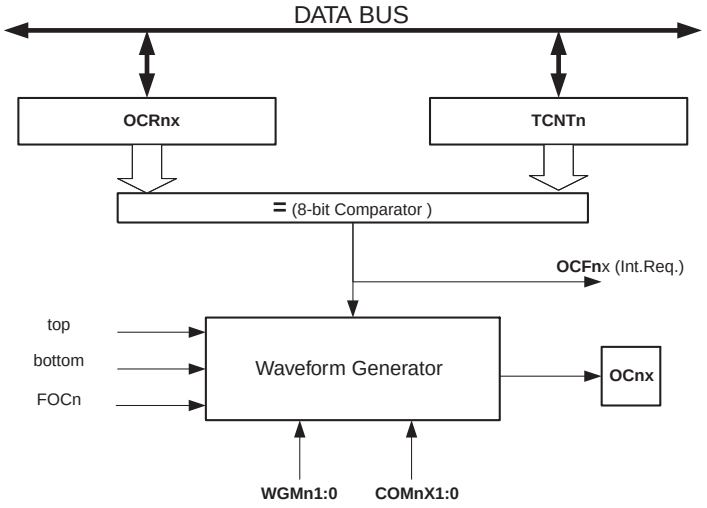
The Timer/Counter Overflow Flag (TOV2) is set according to the mode of operation selected by the WGM21:0 bits. TOV2 can be used for generating a CPU interrupt.

## Output Compare Unit

The 8-bit comparator continuously compares TCNT2 with the Output Compare Register (OCR2A). Whenever TCNT2 equals OCR2A, the comparator signals a match. A match will set the Output Compare Flag (OCF2A) at the next timer clock cycle. If enabled (OCIE2A = 1), the Output Compare Flag generates an Output Compare interrupt. The OCF2A flag is automatically cleared when the interrupt is executed. Alternatively, the OCF2A flag can be cleared by software by writing a logical one to its I/O bit location. The Waveform Generator uses the match signal to generate an output according to operating mode set by the WGM21:0 bits and Compare Output mode (COM2A1:0) bits. The max and bottom signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation (“Modes of Operation” on page 145).

Figure 64 shows a block diagram of the Output Compare unit.

**Figure 64.** Output Compare Unit, Block Diagram




The OCR2A Register is double buffered when using any of the Pulse Width Modulation (PWM) modes. For the Normal and Clear Timer on Compare (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR2A Compare Register to either top or bottom of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR2A Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR2A Buffer Register, and if double buffering is disabled the CPU will access the OCR2A directly.

**Force Output Compare**

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the Force Output Compare (FOC2A) bit. Forcing compare match will not set the OCF2A flag or reload/clear the timer, but the OC2A pin will be updated as if a real compare match had occurred (the COM2A1:0 bits settings define whether the OC2A pin is set, cleared or toggled).

**Compare Match Blocking by TCNT2 Write**

All CPU write operations to the TCNT2 Register will block any compare match that occurs in the next timer clock cycle, even when the timer is stopped. This feature allows OCR2A to be initialized to the same value as TCNT2 without triggering an interrupt when the Timer/Counter clock is enabled.

**Using the Output Compare Unit**

Since writing TCNT2 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT2 when using the Output Compare channel, independently of whether the Timer/Counter is running or not. If the value written to TCNT2 equals the OCR2A value, the compare match will be missed, resulting in incorrect waveform generation. Similarly, do not write the TCNT2 value equal to BOTTOM when the counter is downcounting.

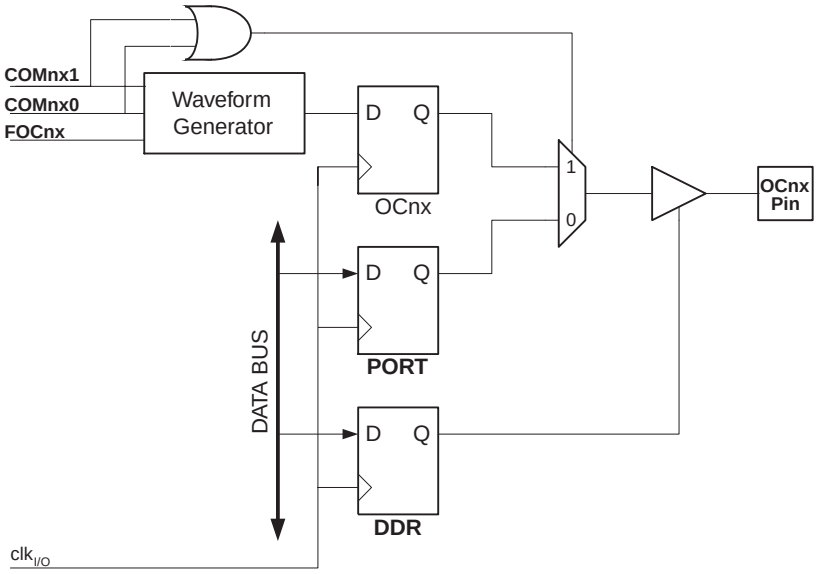
The setup of the OC2A should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC2A value is to use the Force Output Compare (FOC2A) strobe bit in Normal mode. The OC2A Register keeps its value even when changing between Waveform Generation modes.

Be aware that the COM2A1:0 bits are not double buffered together with the compare value. Changing the COM2A1:0 bits will take effect immediately.

## Compare Match Output Unit

The Compare Output mode (COM2A1:0) bits have two functions. The Waveform Generator uses the COM2A1:0 bits for defining the Output Compare (OC2A) state at the next compare match. Also, the COM2A1:0 bits control the OC2A pin output source. Figure 65 shows a simplified schematic of the logic affected by the COM2A1:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O port control registers (DDR and PORT) that are affected by the COM2A1:0 bits are shown. When referring to the OC2A state, the reference is for the internal OC2A Register, not the OC2A pin.

**Figure 65.** Compare Match Output Unit, Schematic




### Compare Output Function

The general I/O port function is overridden by the Output Compare (OC2A) from the Waveform Generator if either of the COM2A1:0 bits are set. However, the OC2A pin direction (input or output) is still controlled by the Data Direction Register (DDR) for the port pin. The Data Direction Register bit for the OC2A pin (DDR_OC2A) must be set as output before the OC2A value is visible on the pin. The port override function is independent of the Waveform Generation mode.

The design of the Output Compare pin logic allows initialization of the OC2A state before the output is enabled. Note that some COM2A1:0 bit settings are reserved for certain modes of operation. See “8-bit Timer/Counter Register Description” on page 151

### Compare Output Mode and Waveform Generation

The Waveform Generator uses the COM2A1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COM2A1:0 = 0 tells the Waveform Generator that no action on the OC2A Register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 67 on page 152. For fast PWM mode, refer to Table 68 on page 152, and for phase correct PWM refer to Table 69 on page 153.

A change of the COM2A1:0 bits state will have effect at the first compare match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOC2A strobe bits.

## Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the Waveform Generation mode (WGM21:0) and

Compare Output mode (COM2A1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM2A1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM2A1:0 bits control whether the output should be set, cleared, or toggled at a compare match (See “Compare Match Output Unit” on page 145 ).

For detailed timing information refer to “Timer/Counter Timing Diagrams” on page 150.

**Normal Mode**

The simplest mode of operation is the Normal mode (WGM21:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 8-bit value (TOP = 0xFF) and then restarts from the bottom (0x00). In normal operation the Timer/Counter Overflow Flag (TOV2) will be set in the same timer clock cycle as the TCNT2 becomes zero. The TOV2 flag in this case behaves like a ninth bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV2 flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

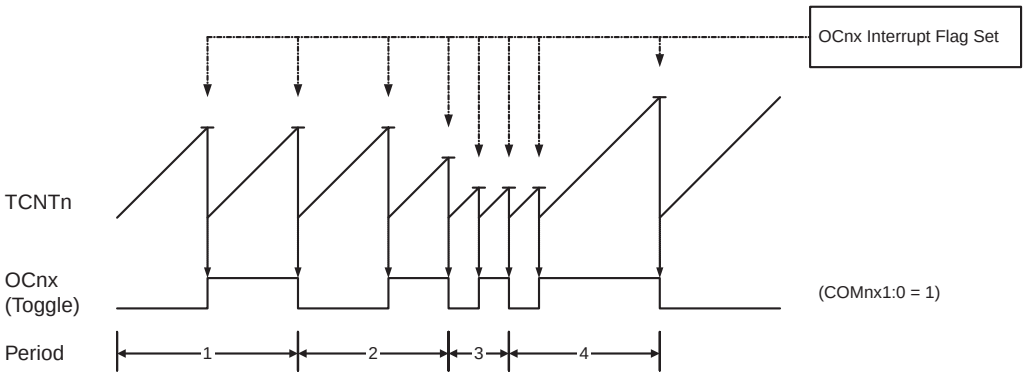
The Output Compare unit can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

**Clear Timer on Compare Match (CTC) Mode**

In Clear Timer on Compare or CTC mode (WGM21:0 = 2), the OCR2A Register is used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT2) matches the OCR2A. The OCR2A defines the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 66. The counter value (TCNT2) increases until a compare match occurs between TCNT2 and OCR2A, and then counter (TCNT2) is cleared.

**Figure 66.** CTC Mode, Timing Diagram




An interrupt can be generated each time the counter value reaches the TOP value by using the OCF2A flag. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing the TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCR2A is lower than the current value of TCNT2, the counter will miss the compare match. The counter will then have to count to its maximum value (0xFF) and wrap around starting at 0x00 before the compare match can occur.

For generating a waveform output in CTC mode, the OC2A output can be set to toggle its logical level on each compare match by setting the Compare Output mode bits to toggle mode (COM2A1:0 = 1). The OC2A value will not be visible on the port pin unless the data direction for the pin is set to output. The waveform generated will have a maximum frequency of $f_{OC2A} = f_{clk_I/O}/2$ when OCR2A is set to zero (0x00). The waveform frequency is defined by the following equation:

$$f_{OCnx} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRnx)}$$

The N variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

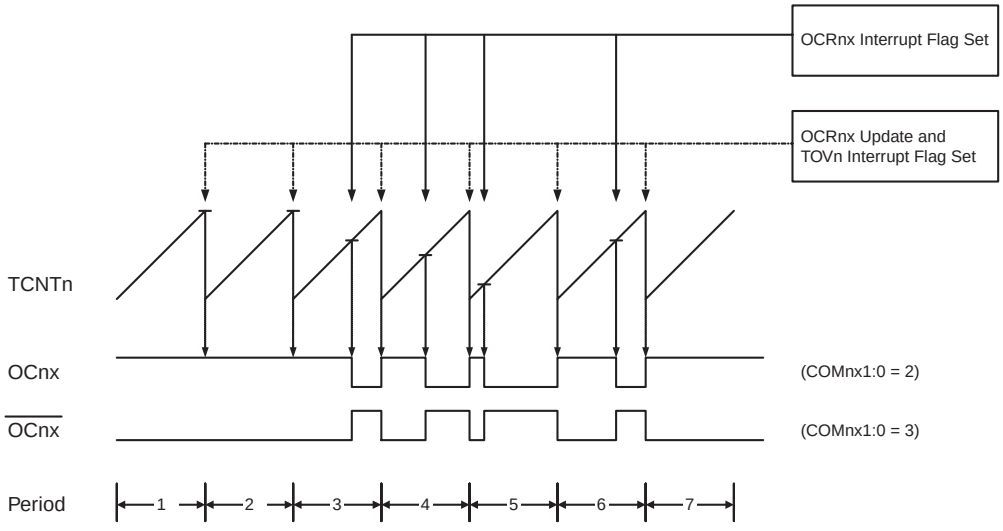
As for the Normal mode of operation, the TOV2 flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

**Fast PWM Mode**

The fast Pulse Width Modulation or fast PWM mode (WGM21:0 = 3) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to MAX then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC2A) is cleared on the compare match between TCNT2 and OCR2A, and set at BOTTOM. In inverting Compare Output mode, the output is set on compare match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct PWM mode that uses dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), and therefore reduces total system cost.

In fast PWM mode, the counter is incremented until the counter value matches the MAX value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 67. The TCNT2 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT2 slopes represent compare matches between OCR2A and TCNT2.

**Figure 67.** Fast PWM Mode, Timing Diagram




The Timer/Counter Overflow Flag (TOV2) is set each time the counter reaches MAX. If the interrupt is enabled, the interrupt handler routine can be used for updating the compare value.

In fast PWM mode, the compare unit allows generation of PWM waveforms on the OC2A pin. Setting the COM2A1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM2A1:0 to three (See Table 68 on page 152). The actual OC2A value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by setting (or clearing) the OC2A Register at the compare match between OCR2A and TCNT2, and clearing (or setting) the OC2A Register at the timer clock cycle the counter is cleared (changes from MAX to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{clk_I/O}}{N \cdot 256}$$

The N variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

The extreme values for the OCR2A Register represent special cases when generating a PWM waveform output in the fast PWM mode. If the OCR2A is set equal to BOTTOM, the output will be a narrow spike for each MAX+1 timer clock cycle. Setting the OCR2A equal to MAX will result in a constantly high or low output (depending on the polarity of the output set by the COM2A1:0 bits.)

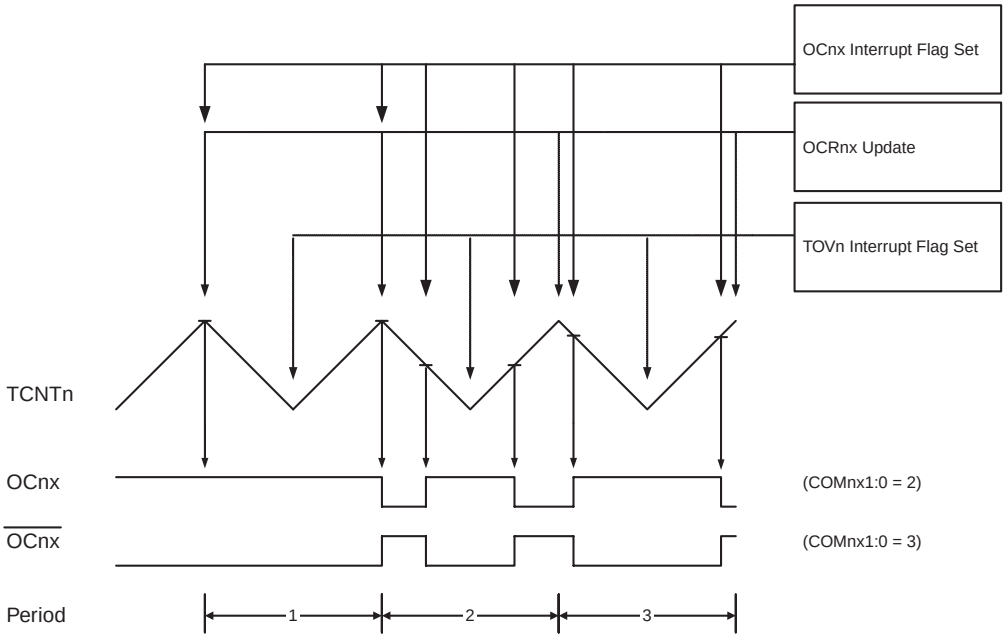
A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC2A to toggle its logical level on each compare match (COM2A1:0 = 1). The waveform generated will have a maximum frequency of $f_{oc2A} = f_{clk_I/O}/2$ when OCR2A is set to zero. This feature is similar to the OC2A toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

**Phase Correct PWM Mode**

The phase correct PWM mode (WGM21:0 = 1) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to MAX and then from MAX to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC2A) is cleared on the compare match between TCNT2 and OCR2A while upcounting, and set on the compare match while downcounting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

The PWM resolution for the phase correct PWM mode is fixed to eight bits. In phase correct PWM mode the counter is incremented until the counter value matches MAX. When the counter reaches MAX, it changes the count direction. The TCNT2 value will be equal to MAX for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 68. The TCNT2 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT2 slopes represent compare matches between OCR2A and TCNT2.

**Figure 68.** Phase Correct PWM Mode, Timing Diagram




The Timer/Counter Overflow Flag (TOV2) is set each time the counter reaches BOTTOM. The interrupt flag can be used to generate an interrupt each time the counter reaches the BOTTOM value.

In phase correct PWM mode, the compare unit allows generation of PWM waveforms on the OC2A pin. Setting the COM2A1:0 bits to two will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM2A1:0 to three (See Table 69 on page 153). The actual OC2A value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by clearing (or setting) the OC2A Register at the compare match between OCR2A and TCNT2 when the counter increments, and setting (or clearing) the OC2A Register at compare match between OCR2A and TCNT2 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnxPCPWM} = \frac{f_{clk_I/O}}{N \cdot 510}$$

The N variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

The extreme values for the OCR2A Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR2A is set equal to BOTTOM, the output will be continuously low and if set equal to MAX the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

## Timer/Counter Timing Diagrams

The following figures show the Timer/Counter in synchronous mode, and the timer clock ($clk_{T2}$) is therefore shown as a clock enable signal. In asynchronous mode, $clk_{I/O}$ should be replaced by the Timer/Counter Oscillator clock. The figures include information on when interrupt flags are set. Figure 69 contains timing data for basic Timer/Counter operation. The figure shows the count sequence close to the MAX value in all modes other than phase correct PWM mode.

**Figure 69.** Timer/Counter Timing Diagram, no Prescaling

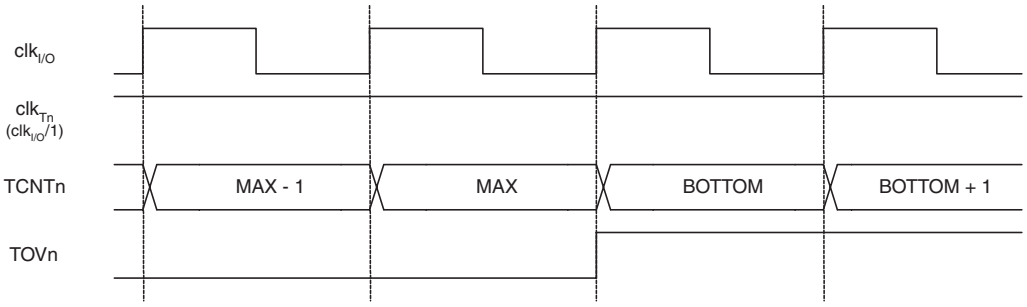



Figure 70 shows the same timing data, but with the prescaler enabled.

**Figure 70.** Timer/Counter Timing Diagram, with Prescaler ($f_{clk_I/O}/8$)

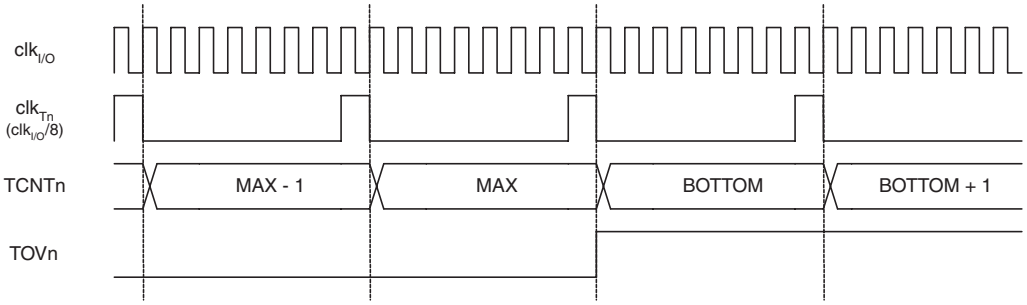



Figure 71 shows the setting of OCF2A in all modes except CTC mode.

**Figure 71.** Timer/Counter Timing Diagram, Setting of OCF2A, with Prescaler ($f_{clk_I/O}/8$)

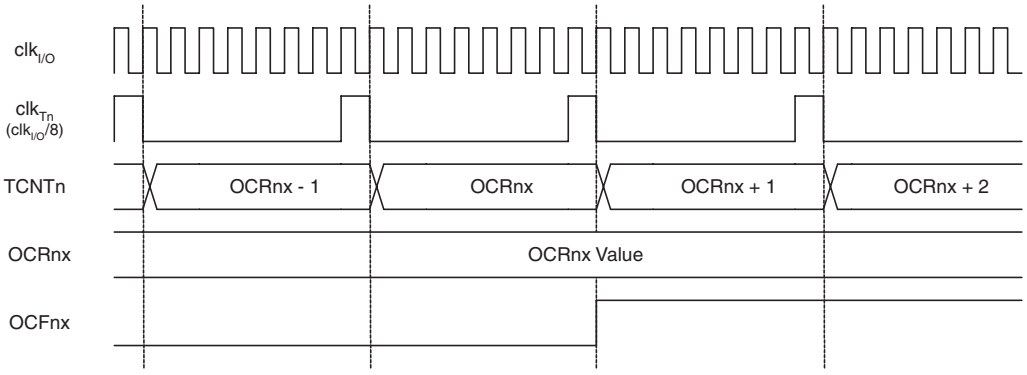
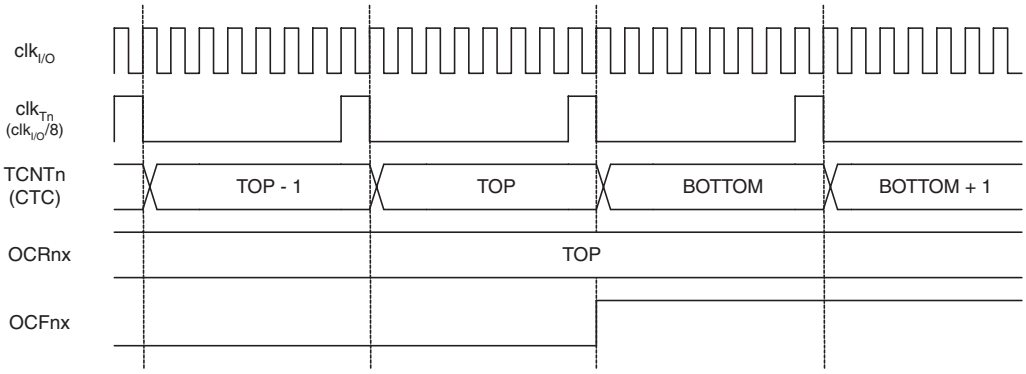


Figure 72 shows the setting of OCF2A and the clearing of TCNT2 in CTC mode.

**Figure 72.** Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{clk_I/O}/8$)




## 8-bit Timer/Counter Register Description

### Timer/Counter2 Control Register A– TCCR2A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | FOC2A | WGM20 | COM2A1 | COM2A0 | WGM21 | CS22 | CS21 | CS20 | TCCR2A |
| Read/Write | W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – FOC2A: Force Output Compare A**

The FOC2A bit is only active when the WGM bits specify a non-PWM mode. However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR2A is written when operating in PWM mode. When writing a logical one to the FOC2A bit, an immediate compare match is forced on the Waveform Generation unit. The OC2A output is changed according to its COM2A1:0 bits setting. Note that the FOC2A bit is implemented as a strobe. Therefore it is the value present in the COM2A1:0 bits that determines the effect of the forced compare.

A FOC2A strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR2A as TOP.

The FOC2A bit is always read as zero.

- **Bit 6, 3 – WGM21:0: Waveform Generation Mode**

These bits control the counting sequence of the counter, the source for the maximum (TOP) counter value, and what type of waveform generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode, Clear Timer on Compare match (CTC) mode, and two types of Pulse Width Modulation (PWM) modes. See Table 66 and “Modes of Operation” on page 145.

**Table 66.** Waveform Generation Mode Bit Description(1)

| Mode | WGM21 (CTC2) | WGM20 (PWM2) | Timer/Counter Mode of Operation | TOP | Update of OCR2A at | TOV2 Flag Set on |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | Normal | 0xFF | Immediate | MAX |
| 1 | 0 | 1 | PWM, Phase Correct | 0xFF | TOP | BOTTOM |
| 2 | 1 | 0 | CTC | OCR2A | Immediate | MAX |
| 3 | 1 | 1 | Fast PWM | 0xFF | TOP | MAX |

Note: 1. The CTC2 and PWM2 bit definition names are now obsolete. Use the WGM21:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

- **Bit 5:4 – COM2A1:0: Compare Match Output Mode A**

These bits control the Output Compare pin (OC2A) behavior. If one or both of the COM2A1:0 bits are set, the OC2A output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to OC2A pin must be set in order to enable the output driver.

When OC2A is connected to the pin, the function of the COM2A1:0 bits depends on the WGM21:0 bit setting. Table 67 shows the COM2A1:0 bit functionality when the WGM21:0 bits are set to a normal or CTC mode (non-PWM).

**Table 67.** Compare Output Mode, non-PWM Mode

| COM2A1 | COM2A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC2A disconnected. |
| 0 | 1 | Toggle OC2A on compare match. |
| 1 | 0 | Clear OC2A on compare match. |
| 1 | 1 | Set OC2A on compare match. |

Table 68 shows the COM2A1:0 bit functionality when the WGM21:0 bits are set to fast PWM mode.

**Table 68.** Compare Output Mode, Fast PWM Mode(1)

| COM2A1 | COM2A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC2A disconnected. |
| 0 | 1 | Reserved |
| 1 | 0 | Clear OC2A on compare match.<br>Set OC2A at TOP. |
| 1 | 1 | Set OC2A on compare match.<br>Clear OC2A at TOP. |

Note: 1. A special case occurs when OCR2A equals TOP and COM2A1 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See “Fast PWM Mode” on page 147 for more details.

Table 69 shows the COM21:0 bit functionality when the WGM21:0 bits are set to phase correct PWM mode.

**Table 69.** Compare Output Mode, Phase Correct PWM Mode(1)

| COM2A1 | COM2A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC2A disconnected. |
| 0 | 1 | Reserved |
| 1 | 0 | Clear OC2A on compare match when up-counting.<br>Set OC2A on compare match when downcounting. |
| 1 | 1 | Set OC2A on compare match when up-counting.<br>Clear OC2A on compare match when downcounting. |

Note: 1. A special case occurs when OCR2A equals TOP and COM2A1 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See “Phase Correct PWM Mode” on page 148 for more details.

- **Bit 2:0 – CS22:0: Clock Select**

The three Clock Select bits select the clock source to be used by the Timer/Counter, see Table 70.

**Table 70.** Clock Select Bit Description

| CS22 | CS21 | CS20 | Description |
|---|---|---|---|
| 0 | 0 | 0 | No clock source (Timer/Counter stopped). |
| 0 | 0 | 1 | $clk_{T2S}$/(No prescaling) |
| 0 | 1 | 0 | $clk_{T2S}$/8 (From prescaler) |
| 0 | 1 | 1 | $clk_{T2S}$/32 (From prescaler) |
| 1 | 0 | 0 | $clk_{T2S}$/64 (From prescaler) |
| 1 | 0 | 1 | $clk_{T2S}$/128 (From prescaler) |
| 1 | 1 | 0 | $clk_{T2S}$/256 (From prescaler) |
| 1 | 1 | 1 | $clk_{T2S}$/1024 (From prescaler) |

### Timer/Counter2 Register – TCNT2

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TCNT2[7:0] | | | | | | | | TCNT2 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT2 Register blocks (removes) the compare match on the following timer clock. Modifying the counter (TCNT2) while the counter is running, introduces a risk of missing a compare match between TCNT2 and the OCR2A Register.

### Output Compare Register A – OCR2A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | OCR2A[7:0] | | | | | | | | OCR2A |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

The Output Compare Register A contains an 8-bit value that is continuously compared with the counter value (TCNT2). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC2A pin.

## Asynchronous operation of the Timer/Counter2

### Asynchronous Status Register – ASSR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | EXCLK | AS2 | TCN2UB | OCR2UB | TCR2UB | ASSR |
| Read/Write | R | R | R | R/W | R/W | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7..5 – Reserved Bits**

These bits are reserved for future use.

- **Bit 4 – EXCLK: Enable External Clock Input**

When EXCLK is written to one, and asynchronous clock is selected, the external clock input buffer is enabled and an external clock can be input on Timer Oscillator 1 (TOSC1) pin instead of a 32 kHz crystal. Writing to EXCLK should be done before asynchronous operation is selected. Note that the crystal Oscillator will only run when this bit is zero.

- **Bit 3 – AS2: Asynchronous Timer/Counter2**

When AS2 is written to zero, Timer/Counter2 is clocked from the I/O clock, $clk_{I/O}$ and the crystal Oscillator connected to the Timer/Counter2 Oscillator (TOSC) does nor run. When AS2 is written to one, Timer/Counter2 is clocked from a crystal Oscillator connected to the Timer/Counter2 Oscillator (TOSC) or from external clock on TOSC1 depending on EXCLK setting. When the value of AS2 is changed, the contents of TCNT2, OCR2A, and TCCR2A might be corrupted.

- **Bit 2 – TCN2UB: Timer/Counter2 Update Busy**

When Timer/Counter2 operates asynchronously and TCNT2 is written, this bit becomes set. When TCNT2 has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that TCNT2 is ready to be updated with a new value.

- **Bit 1 – OCR2UB: Output Compare Register2 Update Busy**

When Timer/Counter2 operates asynchronously and OCR2A is written, this bit becomes set. When OCR2A has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that OCR2A is ready to be updated with a new value.

- **Bit 0 – TCR2UB: Timer/Counter Control Register2 Update Busy**

When Timer/Counter2 operates asynchronously and TCCR2A is written, this bit becomes set. When TCCR2A has been updated from the temporary storage register,

this bit is cleared by hardware. A logical zero in this bit indicates that TCCR2A is ready to be updated with a new value.

If a write is performed to any of the three Timer/Counter2 Registers while its update busy flag is set, the updated value might get corrupted and cause an unintentional interrupt to occur.

The mechanisms for reading TCNT2, OCR2A, and TCCR2A are different. When reading TCNT2, the actual timer value is read. When reading OCR2A or TCCR2A, the value in the temporary storage register is read.

**Asynchronous Operation of Timer/Counter2**

When Timer/Counter2 operates asynchronously, some considerations must be taken.

- Warning: When switching between asynchronous and synchronous clocking of Timer/Counter2, the timer registers TCNT2, OCR2A, and TCCR2A might be corrupted. A safe procedure for switching clock source is:
  1. Disable the Timer/Counter2 interrupts by clearing OCIE2A and TOIE2.
  2. Select clock source by setting AS2 and EXCLK as appropriate.
  3. Write new values to TCNT2, OCR2A, and TCCR2A.
  4. To switch to asynchronous operation: Wait for TCN2UB, OCR2UB, and TCR2UB.
  5. Clear the Timer/Counter2 interrupt flags.
  6. Enable interrupts, if needed.
- The Oscillator is optimized for use with a 32.768 kHz watch crystal. The CPU main clock frequency must be more than four times the Oscillator or external clock frequency.
- When writing to one of the registers TCNT2, OCR2A, or TCCR2A, the value is transferred to a temporary register, and latched after two positive edges on TOSC1. The user should not write a new value before the contents of the temporary register have been transferred to its destination. Each of the three mentioned registers have their individual temporary register, which means that e.g. writing to TCNT2 does not disturb an OCR2A write in progress. To detect that a transfer to the destination register has taken place, the Asynchronous Status Register – ASSR has been implemented.
- When entering Power-save or Extended Standby mode after having written to TCNT2, OCR2A, or TCCR2A, the user must wait until the written register has been updated if Timer/Counter2 is used to wake up the device. Otherwise, the MCU will enter sleep mode before the changes are effective. This is particularly important if the Output Compare2 interrupt is used to wake up the device, since the Output Compare function is disabled during writing to OCR2A or TCNT2. If the write cycle is not finished, and the MCU enters sleep mode before the OCR2UB bit returns to zero, the device will never receive a compare match interrupt, and the MCU will not wake up.
- If Timer/Counter2 is used to wake the device up from Power-save or Extended Standby mode, precautions must be taken if the user wants to re-enter one of these modes: The interrupt logic needs one TOSC1 cycle to be reset. If the time between wake-up and re-entering sleep mode is less than one TOSC1 cycle, the interrupt will not occur, and the device will fail to wake up. If the user is in doubt whether the time before re-entering Power-save mode is sufficient, the following algorithm can be used to ensure that one TOSC1 cycle has elapsed:
  1. Write a value to TCCR2A, TCNT2, or OCR2A.
  2. Wait until the corresponding Update Busy flag in ASSR returns to zero.

3. Enter Power-save or ADC Noise Reduction mode.

- When the asynchronous operation is selected, the 32.768 kHz Oscillator for Timer/Counter2 is always running, except in Power-down and Standby modes. After a Power-up Reset or wake-up from Power-down or Standby mode, the user should be aware of the fact that this Oscillator might take as long as one second to stabilize. The user is advised to wait for at least one second before using Timer/Counter2 after power-up or wake-up from Power-down or Standby mode. The contents of all Timer/Counter2 Registers must be considered lost after a wake-up from Power-down or Standby mode due to unstable clock signal upon start-up, no matter whether the Oscillator is in use or a clock signal is applied to the TOSC1 pin.
- Description of wake up from Power-save mode when the timer is clocked asynchronously: When the interrupt condition is met, the wake up process is started on the following cycle of the timer clock, that is, the timer is always advanced by at least one before the processor can read the counter value. After wake-up, the MCU is halted for four cycles, it executes the interrupt routine, and resumes execution from the instruction following SLEEP.
- Reading of the TCNT2 Register shortly after wake-up from Power-save may give an incorrect result. Since TCNT2 is clocked on the asynchronous TOSC clock, reading TCNT2 must be done through a register synchronized to the internal I/O clock domain. Synchronization takes place for every rising TOSC1 edge. When waking up from Power-save mode, and the I/O clock ($clk_{I/O}$) again becomes active, TCNT2 will read as the previous value (before entering sleep) until the next rising TOSC1 edge. The phase of the TOSC clock after waking up from Power-save mode is essentially unpredictable, as it depends on the wake-up time. The recommended procedure for reading TCNT2 is thus as follows:
  1. Write any value to either of the registers OCR2A or TCCR2A.
  2. Wait for the corresponding Update Busy Flag to be cleared.
  3. Read TCNT2.
- During asynchronous operation, the synchronization of the interrupt flags for the asynchronous timer takes 3 processor cycles plus one timer cycle. The timer is therefore advanced by at least one before the processor can read the timer value causing the setting of the interrupt flag. The Output Compare pin is changed on the timer clock and is not synchronized to the processor clock.

**Timer/Counter2 Interrupt Mask Register – TIMSK2**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | – | – | OCIE2A | TOIE2 | TIMSK2 |
| Read/Write | R | R | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7..2 – Reserved Bits**

These bits are reserved for future use.

- **Bit 1 – OCIE2A: Timer/Counter2 Output Compare Match A Interrupt Enable**

When the OCIE2A bit is written to one and the I-bit in the Status Register is set (one), the Timer/Counter2 Compare Match A interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter2 occurs, i.e., when the OCF2A bit is set in the Timer/Counter2 Interrupt Flag Register – TIFR2.

- **Bit 0 – TOIE2: Timer/Counter2 Overflow Interrupt Enable**

When the TOIE2 bit is written to one and the I-bit in the Status Register is set (one), the Timer/Counter2 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter2 occurs, i.e., when the TOV2 bit is set in the Timer/Counter2 Interrupt Flag Register – TIFR2.

**Timer/Counter2 Interrupt Flag Register – TIFR2**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | – | – | OCF2A | TOV2 | TIFR2 |
| Read/Write | R | R | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7..2 – Reserved Bits**

These bits are reserved for future use.

- **Bit 1 – OCF2A: Output Compare Flag 2 A**

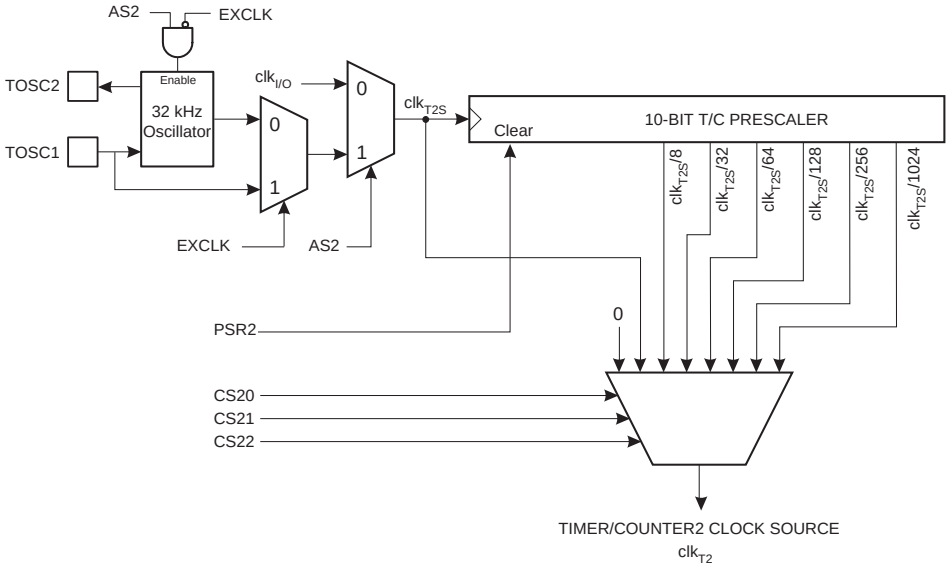
The OCF2A bit is set (one) when a compare match occurs between the Timer/Counter2 and the data in OCR2A – Output Compare Register2. OCF2A is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF2A is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE2 (Timer/Counter2 Compare match Interrupt Enable), and OCF2A are set (one), the Timer/Counter2 Compare match Interrupt is executed.

- **Bit 0 – TOV2: Timer/Counter2 Overflow Flag**

The TOV2 bit is set (one) when an overflow occurs in Timer/Counter2. TOV2 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV2 is cleared by writing a logic one to the flag. When the SREG I-bit, TOIE2A (Timer/Counter2 Overflow Interrupt Enable), and TOV2 are set (one), the Timer/Counter2 Overflow interrupt is executed. In PWM mode, this bit is set when Timer/Counter2 changes counting direction at 0x00.

## Timer/Counter2 Prescaler

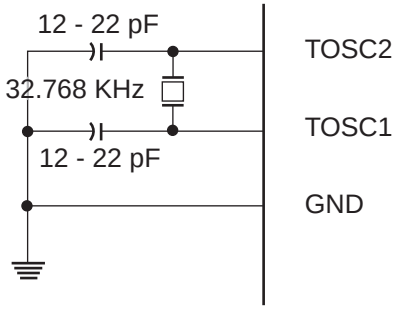
**Figure 73.** Prescaler for Timer/Counter2




The clock source for Timer/Counter2 is named $clk_{T2S}$. $clk_{T2S}$ is by default connected to the main system I/O clock $clk_{IO}$. By setting the AS2 bit in ASSR, Timer/Counter2 is asynchronously clocked from the TOSC oscillator or TOSC1 pin. This enables use of Timer/Counter2 as a Real Time Counter (RTC).

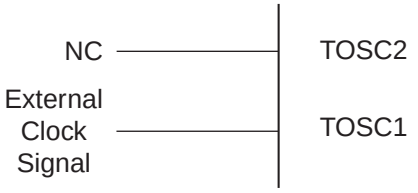
A crystal can then be connected between the TOSC1 and TOSC2 pins to serve as an independent clock source for Timer/Counter2. The Oscillator is optimized for use with a 32.768 kHz crystal. Setting AS2 and resetting EXCLK enables the TOSC oscillator.

**Figure 74.** Timer/Counter2 Crystal Oscillator Connections




A external clock can also be used using TOSC1 as input. Setting AS2 and EXCLK enables this configuration.

**Figure 75.** Timer/Counter2 External Clock Connections



For Timer/Counter2, the possible prescaled selections are: $clk_{T2S}/8$, $clk_{T2S}/32$, $clk_{T2S}/64$, $clk_{T2S}/128$, $clk_{T2S}/256$, and $clk_{T2S}/1024$. Additionally, $clk_{T2S}$ as well as 0 (stop) may be selected. Setting the PSR2 bit in GTCCR resets the prescaler. This allows the user to operate with a predictable prescaler.

### General Timer/Counter Control Register – GTCCR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **TSM** | **–** | **–** | **–** | **–** | **–** | **PSR2** | **PSR310** | **GTCCR** |
| Read/Write | R/W | R | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 1 – PSR2: Prescaler Reset Timer/Counter2**

When this bit is one, the Timer/Counter2 prescaler will be reset. This bit is normally cleared immediately by hardware. If the bit is written when Timer/Counter2 is operating in asynchronous mode, the bit will remain one until the prescaler has been reset. The bit will not be cleared by hardware if the TSM bit is set. Refer to the description of the “Bit 7 – TSM: Timer/Counter Synchronization Mode” on page 92 for a description of the Timer/Counter Synchronization mode.

# Output Compare Modulator - OCM

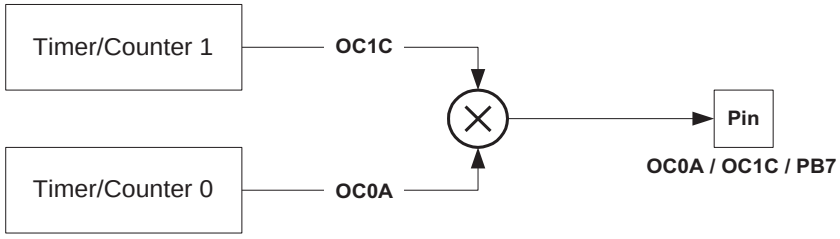
## Overview

Many register and bit references in this section are written in general form.

- A lower case “n” replaces the Timer/Counter number, in this case 0 and 1. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT0 for accessing Timer/Counter0 counter value and so on.
- A lower case “x” replaces the Output Compare unit channel, in this case A or C. However, when using the register or bit defines in a program, the precise form must be used, i.e., OCR0A for accessing Timer/Counter0 output compare channel A value and so on.

The Output Compare Modulator (OCM) allows generation of waveforms modulated with a carrier frequency. The modulator uses the outputs from the Output Compare Unit C of the 16-bit Timer/Counter1 and the Output Compare Unit of the 8-bit Timer/Counter0. For more details about these Timer/Counters see “16-bit Timer/Counter (Timer/Counter1 and Timer/Counter3)” on page 108 and “8-bit Timer/Counter0 with PWM” on page 94.

**Figure 76.** Output Compare Modulator, Block Diagram




When the modulator is enabled, the two output compare channels are modulated together as shown in the block diagram (Figure 76).

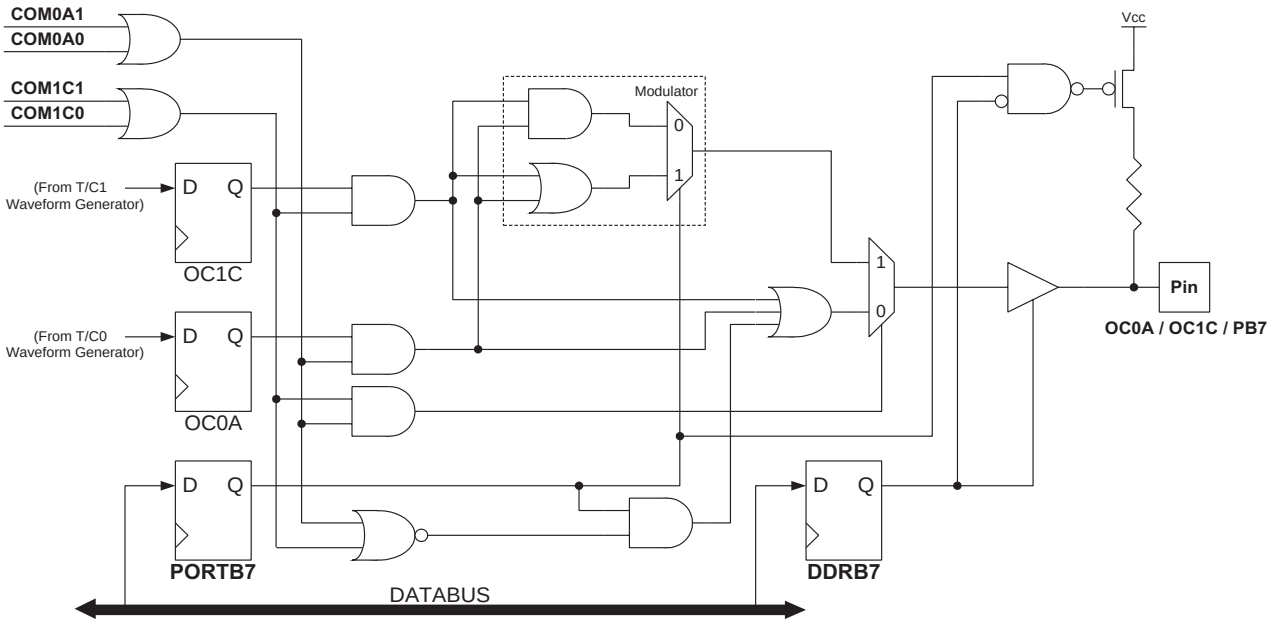
## Description

The Output Compare unit 1C and Output Compare unit 0A shares the PB7 port pin for output. The outputs of the Output Compare units (OC1C and OC0A) overrides the normal PORTB7 Register when one of them is enabled (i.e., when COMnx1:0 is not equal to zero). When both OC1C and OC0A are enabled at the same time, the modulator is automatically enabled.

When the modulator is enabled the type of modulation (logical AND or OR) can be selected by the PORTB7 Register. Note that the DDRB7 controls the direction of the port independent of the COMnx1:0 bit setting.

The functional equivalent schematic of the modulator is shown on Figure 77. The schematic includes part of the Timer/Counter units and the port B pin 7 output driver circuit.

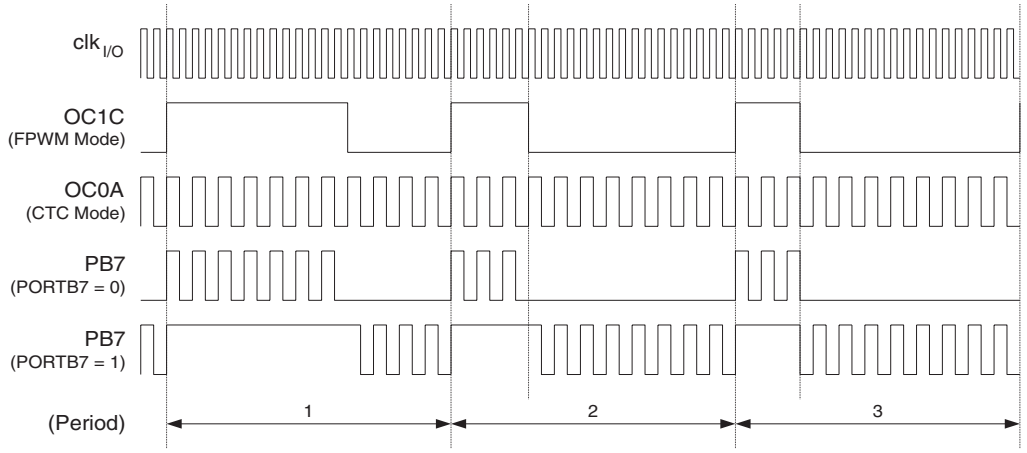
**Figure 77.** Output Compare Modulator, Schematic




**Timing Example**

Figure 78 illustrates the modulator in action. In this example the Timer/Counter1 is set to operate in fast PWM mode (non-inverted) and Timer/Counter0 uses CTC waveform mode with toggle Compare Output mode (COMnx1:0 = 1).

**Figure 78.** Output Compare Modulator, Timing Diagram




In this example, Timer/Counter0 provides the carrier, while the modulating signal is generated by the Output Compare unit C of the Timer/Counter1.

**Resolution of the PWM Signal**

The resolution of the PWM signal (OC1C) is reduced by the modulation. The reduction factor is equal to the number of system clock cycles of one period of the carrier (OC0A). In this example the resolution is reduced by a factor of two. The reason for the reduction is illustrated in Figure 78 at the second and third period of the PB7 output when PORTB7 equals zero. The period 2 high time is one cycle longer than the period 3 high time, but the result on the PB7 output is equal in both periods.

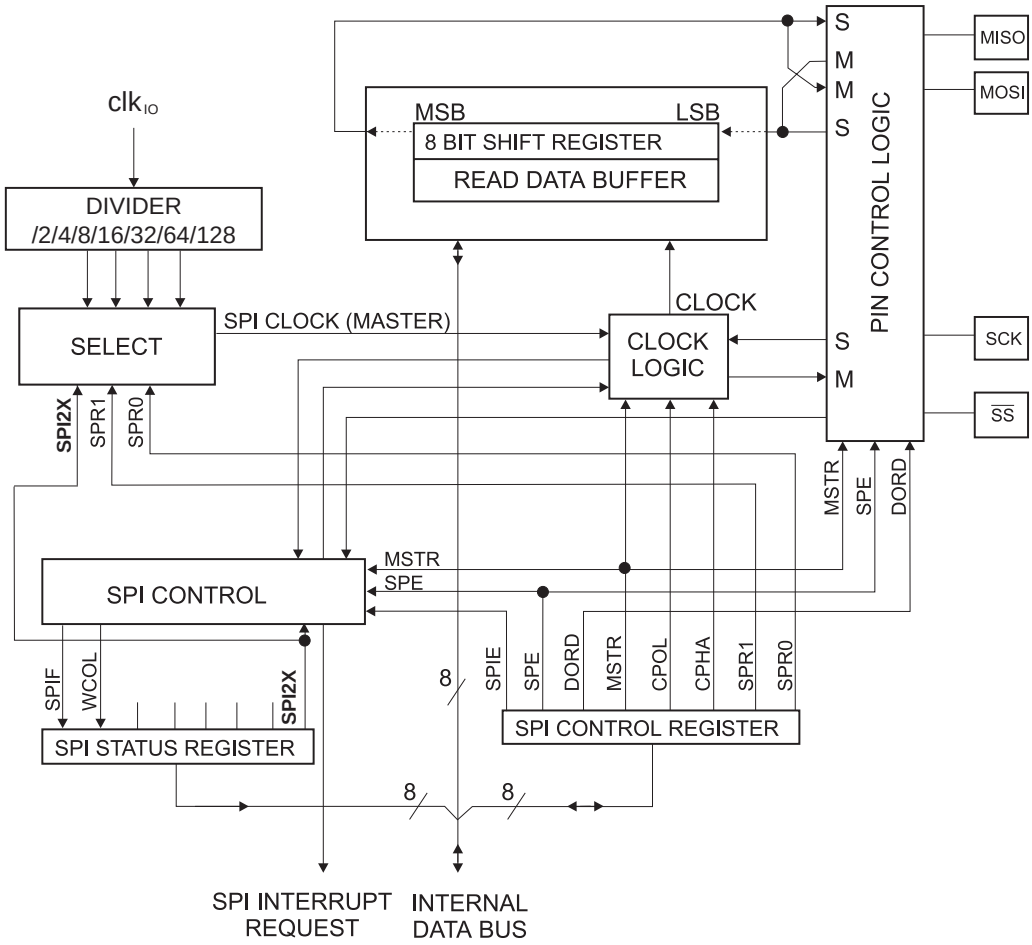
# Serial Peripheral Interface – SPI

The Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between the AT90CAN128 and peripheral devices or between several AVR devices. The AT90CAN128 SPI includes the following features:

## Features

- **Full-duplex, Three-wire Synchronous Data Transfer**
- **Master or Slave Operation**
- **LSB First or MSB First Data Transfer**
- **Seven Programmable Bit Rates**
- **End of Transmission Interrupt Flag**
- **Write Collision Flag Protection**
- **Wake-up from Idle Mode**
- **Double Speed (CK/2) Master SPI Mode**

**Figure 79.** SPI Block Diagram(1)




Note: 1. Refer to Figure 2 on page 4, and Table 32 on page 71 for SPI pin placement.

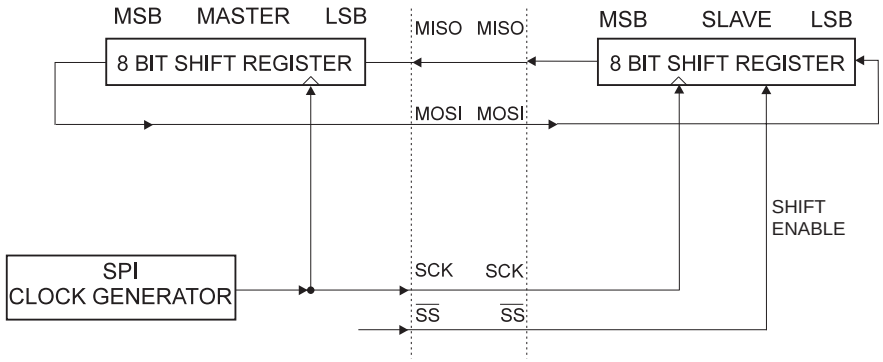
The interconnection between Master and Slave CPUs with SPI is shown in Figure 80. The system consists of two shift Registers, and a Master clock generator. The SPI Master initiates the communication cycle when pulling low the Slave Select $\overline{SS}$ pin of the desired Slave. Master and Slave prepare the data to be sent in their respective shift Registers, and the Master generates the required clock pulses on the SCK line to inter-

change data. Data is always shifted from Master to Slave on the Master Out – Slave In, MOSI, line, and from Slave to Master on the Master In – Slave Out, MISO, line. After each data packet, the Master will synchronize the Slave by pulling high the Slave Select, $\overline{SS}$, line.

When configured as a Master, the SPI interface has no automatic control of the $\overline{SS}$ line. This must be handled by user software before communication can start. When this is done, writing a byte to the SPI Data Register starts the SPI clock generator, and the hardware shifts the eight bits into the Slave. After shifting one byte, the SPI clock generator stops, setting the end of transmission flag (SPIF). If the SPI Interrupt Enable bit (SPIE) in the SPCR Register is set, an interrupt is requested. The Master may continue to shift the next byte by writing it into SPDR, or signal the end of packet by pulling high the Slave Select, $\overline{SS}$ line. The last incoming byte will be kept in the Buffer Register for later use.

When configured as a Slave, the SPI interface will remain sleeping with MISO tri-stated as long as the $\overline{SS}$ pin is driven high. In this state, software may update the contents of the SPI Data Register, SPDR, but the data will not be shifted out by incoming clock pulses on the SCK pin until the $\overline{SS}$ pin is driven low. As one byte has been completely shifted, the end of transmission flag, SPIF is set. If the SPI Interrupt Enable bit, SPIE, in the SPCR Register is set, an interrupt is requested. The Slave may continue to place new data to be sent into SPDR before reading the incoming data. The last incoming byte will be kept in the Buffer Register for later use.

**Figure 80.** SPI Master-slave Interconnection




The system is single buffered in the transmit direction and double buffered in the receive direction. This means that bytes to be transmitted cannot be written to the SPI Data Register before the entire shift cycle is completed. When receiving data, however, a received character must be read from the SPI Data Register before the next character has been completely shifted in. Otherwise, the first byte is lost.

In SPI Slave mode, the control logic will sample the incoming signal of the SCK pin. To ensure correct sampling of the clock signal, the frequency of the SPI clock should never exceed $f_{clkio}/4$.

When the SPI is enabled, the data direction of the MOSI, MISO, SCK, and $\overline{SS}$ pins is overridden according to Table 71. For more details on automatic port overrides, refer to “Alternate Port Functions” on page 67.

**Table 71.** SPI Pin Overrides[1]

| Pin | Direction, Master SPI | Direction, Slave SPI |
|---|---|---|
| MOSI | User Defined | Input |
| MISO | Input | User Defined |
| SCK | User Defined | Input |
| $\overline{SS}$ | User Defined | Input |

Note: 1. See “Alternate Functions of Port B” on page 71 for a detailed description of how to define the direction of the user defined SPI pins.

The following code examples show how to initialize the SPI as a Master and how to perform a simple transmission.

DDR_SPI in the examples must be replaced by the actual Data Direction Register controlling the SPI pins. DD_MOSI, DD_MISO and DD_SCK must be replaced by the actual data direction bits for these pins. E.g. if MOSI is placed on pin PB2, replace DD_MOSI with DDB2 and DDR_SPI with DDRB.

Assembly Code Example[1]

```
SPI_MasterInit:
  ; Set MOSI and SCK output, all others input
  ldi   r17,(1<<DD_MOSI)|(1<<DD_SCK)
  out   DDR_SPI,r17
  ; Enable SPI, Master, set clock rate fck/16
  ldi   r17,(1<<SPE)|(1<<MSTR)|(1<<SPR0)
  out   SPCR,r17
  ret

SPI_MasterTransmit:
  ; Start transmission of data (r16)
  out   SPDR,r16
Wait_Transmit:
  ; Wait for transmission complete
  in    r17,SPSR
  sbrs  r17,SPIF
  rjmp  Wait_Transmit
  ret
```

C Code Example[1]

```
void SPI_MasterInit(void)
{
  /* Set MOSI and SCK output, all others input */
  DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
  /* Enable SPI, Master, set clock rate fck/16 */
  SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}

void SPI_MasterTransmit(char cData)
{
  /* Start transmission */
  SPDR = cData;
  /* Wait for transmission complete */
  while(!(SPSR & (1<<SPIF)))
    ;
}
```

Note: 1. The example code assumes that the part specific header file is included.

The following code examples show how to initialize the SPI as a Slave and how to perform a simple reception.

Assembly Code Example[1]

```
SPI_SlaveInit:
  ; Set MISO output, all others input
  ldi   r17,(1<<DD_MISO)
  out   DDR_SPI,r17
  ; Enable SPI
  ldi   r17,(1<<SPE)
  out   SPCR,r17
  ret

SPI_SlaveReceive:
  ; Wait for reception complete
  sbis  SPSR,SPIF
  rjmp  SPI_SlaveReceive
  ; Read received data and return
  in    r16,SPDR
  ret
```

C Code Example[1]

```
void SPI_SlaveInit(void)
{
  /* Set MISO output, all others input */
  DDR_SPI = (1<<DD_MISO);
  /* Enable SPI */
  SPCR = (1<<SPE);
}

char SPI_SlaveReceive(void)
{
  /* Wait for reception complete */
  while(!(SPSR & (1<<SPIF)))
    ;
  /* Return data register */
  return SPDR;
}
```

Note: 1. The example code assumes that the part specific header file is included.

## $\overline{SS}$ Pin Functionality

**Slave Mode**

When the SPI is configured as a Slave, the Slave Select ($\overline{SS}$) pin is always input. When $\overline{SS}$ is held low, the SPI is activated, and MISO becomes an output if configured so by the user. All other pins are inputs. When $\overline{SS}$ is driven high, all pins are inputs, and the SPI is passive, which means that it will not receive incoming data. Note that the SPI logic will be reset once the $\overline{SS}$ pin is driven high.

The $\overline{SS}$ pin is useful for packet/byte synchronization to keep the slave bit counter synchronous with the master clock generator. When the $\overline{SS}$ pin is driven high, the SPI slave will immediately reset the send and receive logic, and drop any partially received data in the Shift Register.

**Master Mode**

When the SPI is configured as a Master (MSTR in SPCR is set), the user can determine the direction of the $\overline{SS}$ pin.

If $\overline{SS}$ is configured as an output, the pin is a general output pin which does not affect the SPI system. Typically, the pin will be driving the $\overline{SS}$ pin of the SPI Slave.

If $\overline{SS}$ is configured as an input, it must be held high to ensure Master SPI operation. If the $\overline{SS}$ pin is driven low by peripheral circuitry when the SPI is configured as a Master with the $\overline{SS}$ pin defined as an input, the SPI system interprets this as another master selecting the SPI as a slave and starting to send data to it. To avoid bus contention, the SPI system takes the following actions:

1. The MSTR bit in SPCR is cleared and the SPI system becomes a Slave. As a result of the SPI becoming a Slave, the MOSI and SCK pins become inputs.
2. The SPIF flag in SPSR is set, and if the SPI interrupt is enabled, and the I-bit in SREG is set, the interrupt routine will be executed.

Thus, when interrupt-driven SPI transmission is used in Master mode, and there exists a possibility that $\overline{SS}$ is driven low, the interrupt should always check that the MSTR bit is still set. If the MSTR bit has been cleared by a slave select, it must be set by the user to re-enable SPI Master mode.

**SPI Control Register – SPCR**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **SPIE** | **SPE** | **DORD** | **MSTR** | **CPOL** | **CPHA** | **SPR1** | **SPR0** | **SPCR** |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – SPIE: SPI Interrupt Enable**

This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR Register is set and the if the Global Interrupt Enable bit in SREG is set.

- **Bit 6 – SPE: SPI Enable**

When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

- **Bit 5 – DORD: Data Order**

When the DORD bit is written to one, the LSB of the data word is transmitted first.

When the DORD bit is written to zero, the MSB of the data word is transmitted first.

- **Bit 4 – MSTR: Master/Slave Select**

This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If $\overline{SS}$ is configured as an input and is driven low while MSTR is set, MSTR will be cleared, and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI Master mode.

- **Bit 3 – CPOL: Clock Polarity**

When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle. Refer to Figure 81 and Figure 82 for an example. The CPOL functionality is summarized below:

**Table 72.** CPOL Functionality

| CPOL | Leading Edge | Trailing Edge |
|---|---|---|
| 0 | Rising | Falling |
| 1 | Falling | Rising |

- **Bit 2 – CPHA: Clock Phase**

The settings of the Clock Phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK. Refer to Figure 81 and Figure 82 for an example. The CPOL functionality is summarized below:

**Table 73.** CPHA Functionality

| CPHA | Leading Edge | Trailing Edge |
|---|---|---|
| 0 | Sample | Setup |
| 1 | Setup | Sample |

- **Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0**

These two bits control the SCK rate of the device configured as a Master. SPR1 and SPR0 have no effect on the Slave. The relationship between SCK and the $clk_{IO}$ frequency $f_{clkio}$ is shown in the following table:

**Table 74.** Relationship Between SCK and the Oscillator Frequency

| SPI2X | SPR1 | SPR0 | SCK Frequency |
|---|---|---|---|
| 0 | 0 | 0 | $f_{clkio}/4$ |
| 0 | 0 | 1 | $f_{clkio}/16$ |
| 0 | 1 | 0 | $f_{clkio}/64$ |
| 0 | 1 | 1 | $f_{clkio}/128$ |
| 1 | 0 | 0 | $f_{clkio}/2$ |
| 1 | 0 | 1 | $f_{clkio}/8$ |
| 1 | 1 | 0 | $f_{clkio}/32$ |
| 1 | 1 | 1 | $f_{clkio}/64$ |

**SPI Status Register – SPSR**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | SPIF | WCOL | – | – | – | – | – | SPI2X | SPSR |
| Read/Write | R | R | R | R | R | R | R | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – SPIF: SPI Interrupt Flag**

When a serial transfer is complete, the SPIF flag is set. An interrupt is generated if SPIE in SPCR is set and global interrupts are enabled. If $\overline{SS}$ is an input and is driven low when the SPI is in Master mode, this will also set the SPIF flag. SPIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, the SPIF bit is cleared by first reading the SPI Status Register with SPIF set, then accessing the SPI Data Register (SPDR).

- **Bit 6 – WCOL: Write COLlision Flag**

The WCOL bit is set if the SPI Data Register (SPDR) is written during a data transfer. The WCOL bit (and the SPIF bit) are cleared by first reading the SPI Status Register with WCOL set, and then accessing the SPI Data Register.

- **Bit 5..1 – Res: Reserved Bits**

These bits are reserved bits in the AT90CAN128 and will always read as zero.

- **Bit 0 – SPI2X: Double SPI Speed Bit**

When this bit is written logic one the SPI speed (SCK Frequency) will be doubled when the SPI is in Master mode (see Table 74). This means that the minimum SCK period will be two CPU clock periods. When the SPI is configured as Slave, the SPI is only guaranteed to work at $f_{clkio}/4$ or lower.

The SPI interface on the AT90CAN128 is also used for program memory and EEPROM downloading or uploading. See page 337 for serial programming and verification.

**SPI Data Register – SPDR**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **SPD7** | **SPD6** | **SPD5** | **SPD4** | **SPD3** | **SPD2** | **SPD1** | **SPD0** | **SPDR** |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | X | X | X | X | X | X | X | X | Undefined |

- **Bits 7:0 - SPD7:0: SPI Data**

The SPI Data Register is a read/write register used for data transfer between the Register File and the SPI Shift Register. Writing to the register initiates data transmission. Reading the register causes the Shift Register Receive buffer to be read.

## Data Modes

There are four combinations of SCK phase and polarity with respect to serial data, which are determined by control bits CPHA and CPOL. The SPI data transfer formats are shown in Figure 81 and Figure 82. Data bits are shifted out and latched in on opposite edges of the SCK signal, ensuring sufficient time for data signals to stabilize. This is clearly seen by summarizing Table 72 and Table 73, as done below:

**Table 75.** CPOL Functionality

| | Leading Edge | Trailing eDge | SPI Mode |
|---|---|---|---|
| CPOL=0, CPHA=0 | Sample (Rising) | Setup (Falling) | 0 |
| CPOL=0, CPHA=1 | Setup (Rising) | Sample (Falling) | 1 |
| CPOL=1, CPHA=0 | Sample (Falling) | Setup (Rising) | 2 |
| CPOL=1, CPHA=1 | Setup (Falling) | Sample (Rising) | 3 |

**Figure 81.** SPI Transfer Format with CPHA = 0

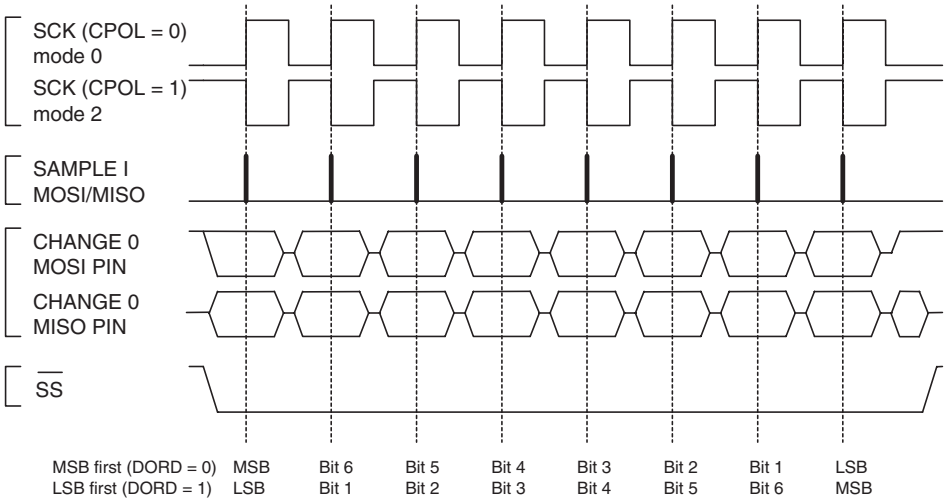
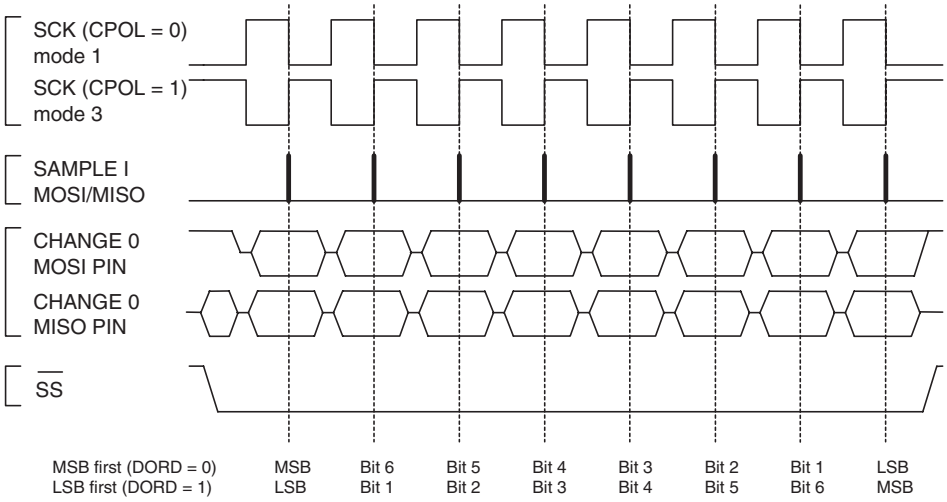



**Figure 82.** SPI Transfer Format with CPHA = 1



# USART (USART0 and USART1)

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device. The main features are:

## Features

- **Full Duplex Operation (Independent Serial Receive and Transmit Registers)**
- **Asynchronous or Synchronous Operation**
- **Master or Slave Clocked Synchronous Operation**
- **High Resolution Baud Rate Generator**
- **Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits**
- **Odd or Even Parity Generation and Parity Check Supported by Hardware**
- **Data OverRun Detection**
- **Framing Error Detection**
- **Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter**
- **Three Separate Interrupts on TX Complete, TX Data Register Empty and RX Complete**
- **Multi-processor Communication Mode**
- **Double Speed Asynchronous Communication Mode**

## Dual USART

The AT90CAN128 has two USART's, USART0 and USART1. The functionality for both USART's is described below. USART0 and USART1 have different I/O registers as shown in "Register Summary" on page 394.

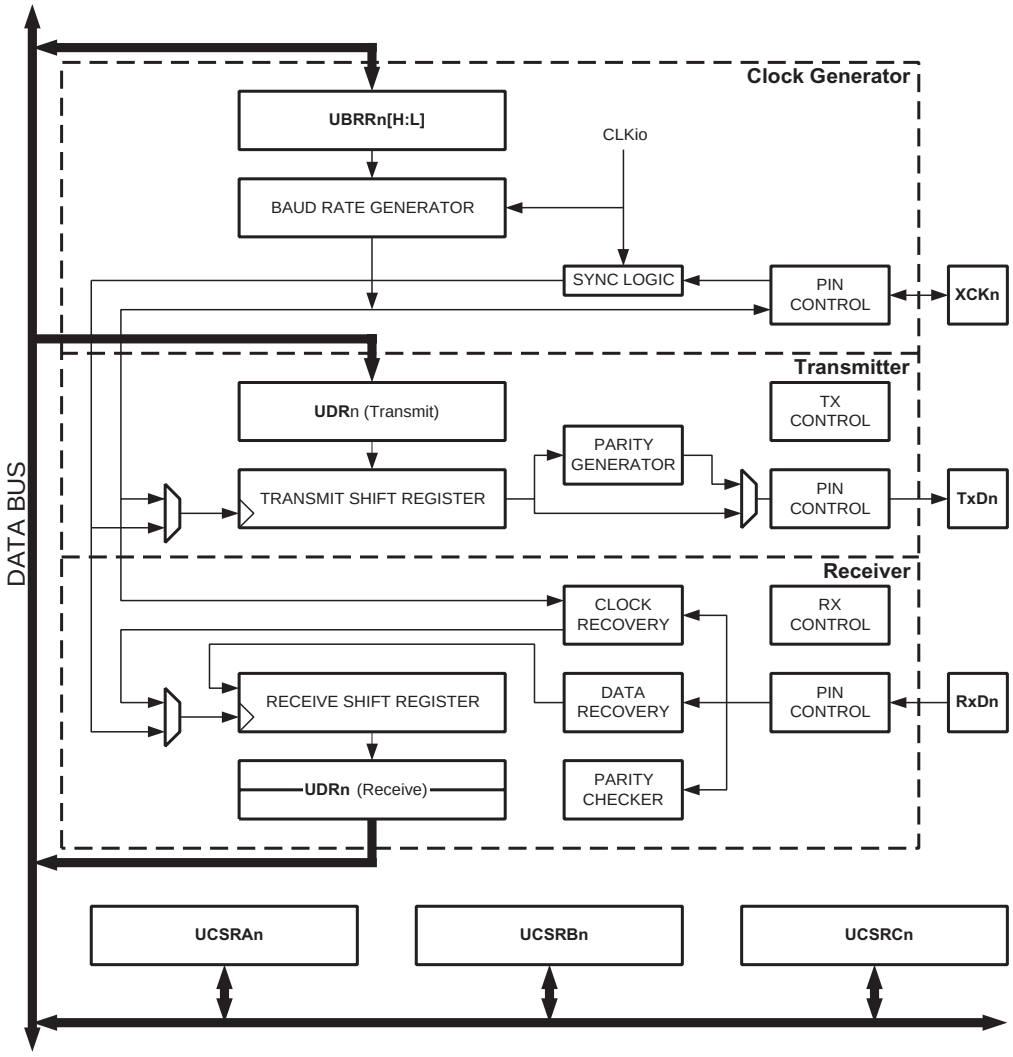
## Overview

Many register and bit references in this section are written in general form.

A lower case “n” replaces the USART number, in this case 0 and 1. However, when using the register or bit defines in a program, the precise form must be used, i.e., UDR0 for accessing USART0 I/O data value and so on.

A simplified block diagram of the USARTn Transmitter is shown in Figure 83. CPU accessible I/O Registers and I/O pins are shown in bold.

**Figure 83.** USARTn Block Diagram[1]




Note: 1. Refer to Figure 2 on page 4, Table 41 on page 78, and Table 36 on page 74 for USARTn pin placement.

The dashed boxes in the block diagram separate the three main parts of the USARTn (listed from the top): Clock Generator, Transmitter and Receiver. Control registers are shared by all units. The Clock Generation logic consists of synchronization logic for external clock input used by synchronous slave operation, and the baud rate generator. The XCKn (Transfer Clock) pin is only used by synchronous transfer mode. The Transmitter consists of a single write buffer, a serial Shift Register, Parity Generator and Control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without any delay between frames. The Receiver is the most complex part of the USARTn module due to its clock and data recovery units. The

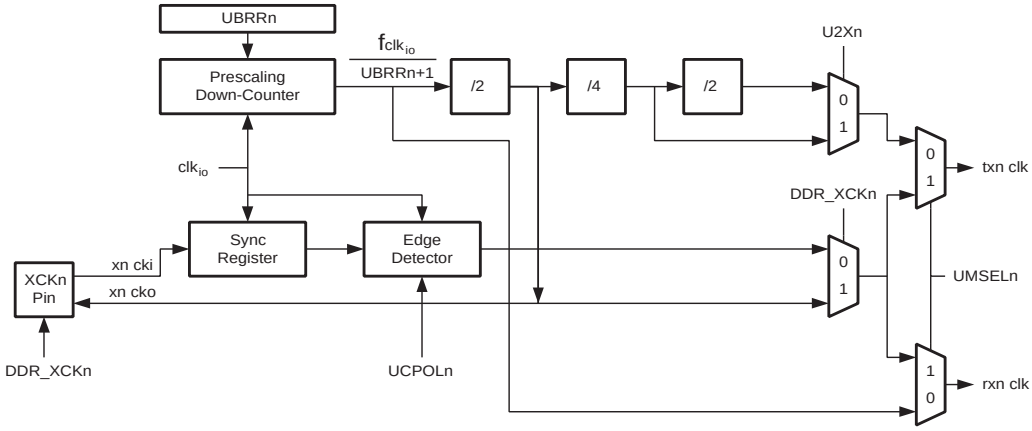
recovery units are used for asynchronous data reception. In addition to the recovery units, the Receiver includes a Parity Checker, Control logic, a Shift Register and a two level receive buffer (UDRn). The Receiver supports the same frame formats as the Transmitter, and can detect Frame Error, Data OverRun and Parity Errors.

## Clock Generation

The Clock Generation logic generates the base clock for the Transmitter and Receiver. The USARTn supports four modes of clock operation: Normal asynchronous, Double Speed asynchronous, Master synchronous and Slave synchronous mode. The UMSELn bit in USARTn Control and Status Register C (UCSRnC) selects between asynchronous and synchronous operation. Double Speed (asynchronous mode only) is controlled by the U2Xn found in the UCSRnA Register. When using synchronous mode (UMSELn = 1), the Data Direction Register for the XCKn pin (DDR_XCKn) controls whether the clock source is internal (Master mode) or external (Slave mode). The XCKn pin is only active when using synchronous mode.

Figure 84 shows a block diagram of the clock generation logic.

**Figure 84.** USARTn Clock Generation Logic, Block Diagram




Signal description:

**txn clk** Transmitter clock (Internal Signal).

**rxn clk** Receiver base clock (Internal Signal).

**xn cki** Input from XCK pin (internal Signal). Used for synchronous slave operation.

**xn cko** Clock output to XCK pin (Internal Signal). Used for synchronous master operation.

**$f_{clk_{io}}$** System I/O Clock frequency.

### Internal Clock Generation – Baud Rate Generator

Internal clock generation is used for the asynchronous and the synchronous master modes of operation. The description in this section refers to Figure 84.

The USARTn Baud Rate Register (UBRRn) and the down-counter connected to it function as a programmable prescaler or baud rate generator. The down-counter, running at system clock ($f_{clk_{io}}$), is loaded with the UBRRn value each time the counter has counted down to zero or when the UBRRnL Register is written. A clock is generated each time the counter reaches zero. This clock is the baud rate generator clock output (= $f_{clk_{io}}/(UBRRn+1)$). The Transmitter divides the baud rate generator clock output by 2, 8 or 16 depending on mode. The baud rate generator output is used directly by the Receiver's clock and data recovery units. However, the recovery units use a state

machine that uses 2, 8 or 16 states depending on mode set by the state of the UMSELn, U2Xn and DDR_XCKn bits.

Table 76 contains equations for calculating the baud rate (in bits per second) and for calculating the UBRRn value for each mode of operation using an internally generated clock source.

**Table 76.** Equations for Calculating Baud Rate Register Setting

| Operating Mode | Equation for Calculating Baud Rate[1] | Equation for Calculating UBRRn Value |
|---|---|---|
| Asynchronous Normal mode (U2Xn = 0) | $BAUD = \frac{f_{CLKio}}{16(\boldsymbol{UBRR}n + 1)}$ | $\boldsymbol{UBRR}n = \frac{f_{CLKio}}{16BAUD} - 1$ |
| Asynchronous Double Speed mode (U2Xn = 1) | $BAUD = \frac{f_{CLKio}}{8(\boldsymbol{UBRR}n + 1)}$ | $\boldsymbol{UBRR}n = \frac{f_{CLKio}}{8BAUD} - 1$ |
| Synchronous Master mode | $BAUD = \frac{f_{CLKio}}{2(\boldsymbol{UBRR}n + 1)}$ | $\boldsymbol{UBRR}n = \frac{f_{CLKio}}{2BAUD} - 1$ |

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps)

**BAUD** Baud rate (in bits per second, bps).

**$f_{clk_{io}}$** System I/O Clock frequency.

**UBRRn** Contents of the UBRRnH and UBRRnL Registers, (0-4095).

Some examples of UBRRn values for some system clock frequencies are found in Table 84 (see page 195).

**Double Speed Operation (U2X)**

The transfer rate can be doubled by setting the U2Xn bit in UCSRnA. Setting this bit only has effect for the asynchronous operation. Set this bit to zero when using synchronous operation.

Setting this bit will reduce the divisor of the baud rate divider from 16 to 8, effectively doubling the transfer rate for asynchronous communication. Note however that the Receiver will in this case only use half the number of samples (reduced from 16 to 8) for data sampling and clock recovery, and therefore a more accurate baud rate setting and system clock are required when this mode is used. For the Transmitter, there are no downsides.

**External Clock**

External clocking is used by the synchronous slave modes of operation. The description in this section refers to Figure 84 for details.

External clock input from the XCKn pin is sampled by a synchronization register to minimize the chance of meta-stability. The output from the synchronization register must then pass through an edge detector before it can be used by the Transmitter and Receiver. This process introduces a two CPU clock period delay and therefore the maximum external XCKn clock frequency is limited by the following equation:

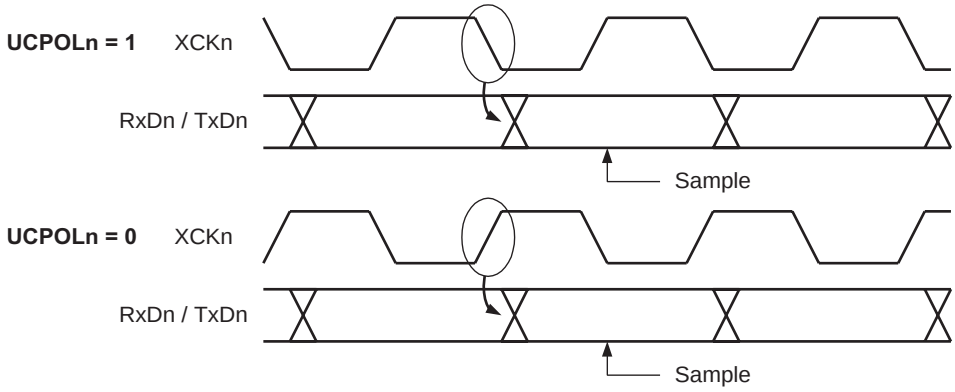
$$f_{XCKn} < \frac{f_{CLKio}}{4}$$

Note that $f_{clk_{io}}$ depends on the stability of the system clock source. It is therefore recommended to add some margin to avoid possible loss of data due to frequency variations.

### Synchronous Clock Operation

When synchronous mode is used (UMSELn = 1), the XCKn pin will be used as either clock input (Slave) or clock output (Master). The dependency between the clock edges and data sampling or data change is the same. The basic principle is that data input (on RxDn) is sampled at the opposite XCKn clock edge of the edge the data output (TxDn) is changed.

**Figure 85.** Synchronous Mode XCKn Timing.




The UCPOLn bit UCRSnC selects which XCKn clock edge is used for data sampling and which is used for data change. As Figure 85 shows, when UCPOLn is zero the data will be changed at rising XCKn edge and sampled at falling XCKn edge. If UCPOLn is set, the data will be changed at falling XCKn edge and sampled at rising XCKn edge.

## Serial Frame

A serial frame is defined to be one character of data bits with synchronization bits (start and stop bits), and optionally a parity bit for error checking.

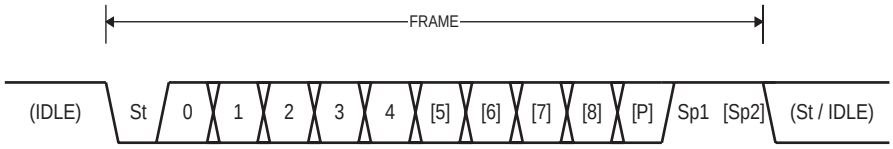
### Frame Formats

The USARTn accepts all 30 combinations of the following as valid frame formats:

- 1 start bit
- 5, 6, 7, 8, or 9 data bits
- no, even or odd parity bit
- 1 or 2 stop bits

A frame starts with the start bit followed by the least significant data bit. Then the next data bits, up to a total of nine, are succeeding, ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, it can be directly followed by a new frame, or the communication line can be set to an idle (high) state. Figure 86 illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

**Figure 86.** Frame Formats




**St** Start bit, always low.

**(n)** Data bits (0 to 8).

**P** Parity bit. Can be odd or even.

**Sp** Stop bit, always high.

**IDLE** No transfers on the communication line (RxDn or TxDn). An IDLE line must be high.

The frame format used by the USARTn is set by the UCSZn2:0, UPMn1:0 and USBSn bits in UCSRnB and UCSRnC. The Receiver and Transmitter use the same setting. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter.

The USARTn Character SiZe (UCSZn2:0) bits select the number of data bits in the frame. The USARTn Parity mode (UPMn1:0) bits enable and set the type of parity bit. The selection between one or two stop bits is done by the USARTn Stop Bit Select (USBSn) bit. The Receiver ignores the second stop bit. An FEn (Frame Error) will therefore only be detected in the cases where the first stop bit is zero.

### Parity Bit Calculation

The parity bit is calculated by doing an exclusive-or of all the data bits. If odd parity is used, the result of the exclusive or is inverted. The relation between the parity bit and data bits is as follows:

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

**$P_{even}$** Parity bit using even parity

**$P_{odd}$** Parity bit using odd parity

**$d_n$** Data bit n of the character

If used, the parity bit is located between the last data bit and first stop bit of a serial frame.

## USART Initialization

The USARTn has to be initialized before any communication can take place. The initialization process normally consists of setting the baud rate, setting frame format and enabling the Transmitter or the Receiver depending on the usage. For interrupt driven USARTn operation, the Global Interrupt Flag should be cleared (and interrupts globally disabled) when doing the initialization.

Before doing a re-initialization with changed baud rate or frame format, be sure that there are no ongoing transmissions during the period the registers are changed. The TXCn flag can be used to check that the Transmitter has completed all transfers, and the RXCn flag can be used to check that there are no unread data in the receive buffer. Note that the TXCn flag must be cleared before each transmission (before UDRn is written) if it is used for this purpose.

The following simple USART0 initialization code examples show one assembly and one C function that are equal in functionality. The examples assume asynchronous operation using polling (no interrupts enabled) and a fixed frame format. The baud rate is given as a function parameter. For the assembly code, the baud rate parameter is assumed to be stored in the r17:r16 Registers.

Assembly Code Example[1]

```
USART0_Init:
  ; Set baud rate
  sts   UBRR0H, r17
  sts   UBRR0L, r16
  ; Set frame format: 8data, no parity & 2 stop bits
  ldi   r16, (0<<UMSEL0)|(0<<UPM0)|(1<<USBS0)|(3<<UCSZ0)
  sts   UCSR0C, r16
  ; Enable receiver and transmitter
  ldi   r16, (1<<RXEN0)|(1<<TXEN0)
  sts   UCSR0B, r16
  ret
```

C Code Example[1]

```
void USART0_Init (unsigned int baud )
{
  /* Set baud rate */
  UBRR0H = (unsigned char) (baud>>8);
  UBRR0L = (unsigned char) baud;
  /* Set frame format: 8data, no parity & 2 stop bits */
  UCSR0C = (0<<UMSEL0) | (0<<UPM0) | (1<<USBS0) | (3<<UCSZ0);
  /* Enable receiver and transmitter */
  UCSR0B = (1<<RXEN0) | (1<<TXEN0);
}
```

Note: 1. The example code assumes that the part specific header file is included.

More advanced initialization routines can be made that include frame format as parameters, disable interrupts and so on. However, many applications use a fixed setting of the baud and control registers, and for these types of applications the initialization code can be placed directly in the main routine, or be combined with initialization code for other I/O modules.

## Data Transmission – USART Transmitter

The USARTn Transmitter is enabled by setting the Transmit Enable (TXENn) bit in the UCSRnB Register. When the Transmitter is enabled, the normal port operation of the TxDn pin is overridden by the USARTn and given the function as the Transmitter's serial output. The baud rate, mode of operation and frame format must be set up once before doing any transmissions. If synchronous operation is used, the clock on the XCKn pin will be overridden and used as transmission clock.

### Sending Frames with 5 to 8 Data Bit

A data transmission is initiated by loading the transmit buffer with the data to be transmitted. The CPU can load the transmit buffer by writing to the UDRn I/O location. The buffered data in the transmit buffer will be moved to the Shift Register when the Shift Register is ready to send a new frame. The Shift Register is loaded with new data if it is in idle state (no ongoing transmission) or immediately after the last stop bit of the previous frame is transmitted. When the Shift Register is loaded with new data, it will transfer one complete frame at the rate given by the Baud Register, U2Xn bit or by XCKn depending on mode of operation.

The following code examples show a simple USART0 transmit function based on polling of the Data Register Empty (UDRE0) flag. When using frames with less than eight bits, the most significant bits written to the UDR0 are ignored. The USART0 has to be initialized before the function can be used. For the assembly code, the data to be sent is assumed to be stored in Register R16.

Assembly Code Example[1]

```
USART0_Transmit:
  ; Wait for empty transmit buffer
  lds   r17, UCSR0A
  sbrs  r17, UDRE0
  rjmp  USART0_Transmit
  ; Put data (r16) into buffer, sends the data
  sts   UDR0, r16
  ret
```

C Code Example[1]

```
void USART0_Transmit (unsigned char data )
{
  /* Wait for empty transmit buffer */
  while ( ! ( UCSRA0 & (1<<UDRE0)))
        ;
  /* Put data into buffer, sends the data */
  UDR0 = data;
}
```

Note: 1. The example code assumes that the part specific header file is included.

The function simply waits for the transmit buffer to be empty by checking the UDRE0 flag, before loading it with new data to be transmitted. If the Data Register Empty interrupt is utilized, the interrupt routine writes the data into the buffer.

### Sending Frames with 9 Data Bit

If 9-bit characters are used (UCSZn = 7), the ninth bit must be written to the TXB8n bit in UCSRnB before the low byte of the character is written to UDRn. The following code examples show a transmit function that handles 9-bit characters. For the assembly code, the data to be sent is assumed to be stored in registers R17:R16.

Assembly Code Example[1][2]

```
USART0_Transmit:
  ; Wait for empty transmit buffer
  lds   r18, UCSR0A
  sbrs  r18, UDRE0
  rjmp  USART0_Transmit
  ; Copy 9th bit from r17-bit0 to TXB80 via T-bit of SREG
  lds   r18, UCSR0B
  bst   r17, 0
  bld   r18, TXB80
  sts   UCSR0B, r18
  ; Put LSB data (r16) into buffer, sends the data
  sts   UDR0, r16
  ret
```

C Code Example[1][2]

```
void USART0_Transmit (unsigned int data )
{
  /* Wait for empty transmit buffer */
  while ( !( UCSR0A & (1<<UDRE0)))
        ;
  /* Copy 9th bit to TXB8 */
  UCSR0B &= ~(1<<TXB80);
  if ( data & 0x0100 )
    UCSR0B |= (1<<TXB80);
  /* Put data into buffer, sends the data */
  UDR0 = data;
}
```

Notes:
1. These transmit functions are written to be general functions. They can be optimized if the contents of the UCSR0B is static. For example, only the TXB80 bit of the UCSRB0 Register is used after initialization.
2. The example code assumes that the part specific header file is included.
   sparc.

The ninth bit can be used for indicating an address frame when using multi processor communication mode or for other protocol handling as for example synchronization.

### Transmitter Flags and Interrupts

The USARTn Transmitter has two flags that indicate its state: USART Data Register Empty (UDREn) and Transmit Complete (TXCn). Both flags can be used for generating interrupts.

The Data Register Empty (UDREn) flag indicates whether the transmit buffer is ready to receive new data. This bit is set when the transmit buffer is empty, and cleared when the transmit buffer contains data to be transmitted that has not yet been moved into the Shift Register. For compatibility with future devices, always write this bit to zero when writing the UCSRnA Register.

When the Data Register Empty Interrupt Enable (UDRIEn) bit in UCSRBn is written to one, the USARTn Data Register Empty Interrupt will be executed as long as UDREn is set (provided that global interrupts are enabled). UDREn is cleared by writing UDRn. When interrupt-driven data transmission is used, the Data Register Empty interrupt routine must either write new data to UDRn in order to clear UDREn or disable the Data Register Empty interrupt, otherwise a new interrupt will occur once the interrupt routine terminates.

The Transmit Complete (TXCn) flag bit is set one when the entire frame in the Transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer. The TXCn flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXCn flag is useful in half-duplex communication interfaces (like the RS-485 standard), where a transmitting application must enter receive mode and free the communication bus immediately after completing the transmission.

When the Transmit Complete Interrupt Enable (TXCIEn) bit in UCSRnB is set, the USARTn Transmit Complete Interrupt will be executed when the TXCn flag becomes set (provided that global interrupts are enabled). When the transmit complete interrupt is used, the interrupt handling routine does not have to clear the TXCn flag, this is done automatically when the interrupt is executed.

### Parity Generator

The Parity Generator calculates the parity bit for the serial frame data. When parity bit is enabled (UPMn1 = 1), the transmitter control logic inserts the parity bit between the last data bit and the first stop bit of the frame that is sent.

### Disabling the Transmitter

The disabling of the Transmitter (setting the TXENn to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer override the TxDn pin.

## Data Reception – USART Receiver

The USARTn Receiver is enabled by writing the Receive Enable (RXENn) bit in the UCSRnB Register to one. When the Receiver is enabled, the normal pin operation of the RxDn pin is overridden by the USARTn and given the function as the Receiver's serial input. The baud rate, mode of operation and frame format must be set up once before any serial reception can be done. If synchronous operation is used, the clock on the XCKn pin will be used as transfer clock.

### Receiving Frames with 5 to 8 Data Bits

The Receiver starts data reception when it detects a valid start bit. Each bit that follows the start bit will be sampled at the baud rate or XCKn clock, and shifted into the Receive Shift Register until the first stop bit of a frame is received. A second stop bit will be ignored by the Receiver. When the first stop bit is received, i.e., a complete serial frame is present in the Receive Shift Register, the contents of the Shift Register will be moved into the receive buffer. The receive buffer can then be read by reading the UDRn I/O location.

The following code example shows a simple USART0 receive function based on polling of the Receive Complete (RXC0) flag. When using frames with less than eight bits the most significant bits of the data read from the UDR0 will be masked to zero. The USART0 has to be initialized before the function can be used.

Assembly Code Example[1]

```
USART0_Receive:
  ; Wait for data to be received
  lds   r18, UCSR0A
  sbrs  r18, RXC0
  rjmp  USART0_Receive
  ; Get and return received data from buffer
  lds   r16, UDR0
  ret
```

C Code Example[1]

```
unsigned char USART0_Receive (void )
{
  /* Wait for data to be received */
  while ( ! (UCSR0A & (1<<RXC0)))
        ;
  /* Get and return received data from buffer */
  return UDR0;
}
```

Note: 1. The example code assumes that the part specific header file is included.

The function simply waits for data to be present in the receive buffer by checking the RXC0 flag, before reading the buffer and returning the value.

**Receiving Frames with 9 Data Bits**

If 9-bit characters are used (UCSZn=7) the ninth bit must be read from the RXB8n bit in UCSRnB **before** reading the low bits from the UDRn. This rule applies to the FEn, DORn and UPEn Status Flags as well. Read status from UCSRnA, then data from UDRn. Reading the UDRn I/O location will change the state of the receive buffer FIFO and consequently the TXB8n, FEn, DORn and UPEn bits, which all are stored in the FIFO, will change.

The following code example shows a simple USART0 receive function that handles both nine bit characters and the status bits.

Assembly Code Example[1]

```
USART0_Receive:
  ; Wait for data to be received
  lds   r18, UCSR0A
  sbrs  r18, RXC0
  rjmp  USART0_Receive
  ; Get status and 9th bit, then data from buffer
  lds   r17, UCSR0B
  lds   r16, UDR0
  ; If error, return -1
  andi  r18, (1<<FE0) | (1<<DOR0) | (1<<UPE0)
  breq  USART0_ReceiveNoError
  ldi   r17, HIGH(-1)
  ldi   r16, LOW(-1)
USART0_ReceiveNoError:
  ; Filter the 9th bit, then return
  lsr   r17
  andi  r17, 0x01
  ret
```

C Code Example[1]

```
unsigned int USART0_Receive( void )
{
  unsigned char status, resh, resl;
  /* Wait for data to be received */
  while ( ! (UCSR0A & (1<<RXC0)))
        ;
  /* Get status and 9th bit, then data */
  /* from buffer */
  status = UCSR0A;
  resh = UCSR0B;
  resl = UDR0;
  /* If error, return -1 */
  if ( status & (1<<FE0)|(1<<DOR0)|(1<<UPE0) )
    return -1;
  /* Filter the 9th bit, then return */
  resh = (resh >> 1) & 0x01;
  return ((resh << 8) | resl);
}
```

Note: 1. The example code assumes that the part specific header file is included.

The receive function example reads all the I/O Registers into the Register File before any computation is done. This gives an optimal receive buffer utilization since the buffer location read will be free to accept new data as early as possible.

### Receive Complete Flag and Interrupt

The USARTn Receiver has one flag that indicates the Receiver state.

The Receive Complete (RXCn) flag indicates if there are unread data present in the receive buffer. This flag is one when unread data exist in the receive buffer, and zero when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled (RXENn = 0), the receive buffer will be flushed and consequently the RXCn bit will become zero.

When the Receive Complete Interrupt Enable (RXCIEn) in UCSRnB is set, the USARTn Receive Complete interrupt will be executed as long as the RXCn flag is set (provided that global interrupts are enabled). When interrupt-driven data reception is used, the receive complete routine must read the received data from UDRn in order to clear the RXCn flag, otherwise a new interrupt will occur once the interrupt routine terminates.

### Receiver Error Flags

The USARTn Receiver has three error flags: Frame Error (FEn), Data OverRun (DORn) and Parity Error (UPEn). All can be accessed by reading UCSRnA. Common for the error flags is that they are located in the receive buffer together with the frame for which they indicate the error status. Due to the buffering of the error flags, the UCSRnA must be read before the receive buffer (UDRn), since reading the UDRn I/O location changes the buffer read location. Another equality for the error flags is that they can not be altered by software doing a write to the flag location. However, all flags must be set to zero when the UCSRnA is written for upward compatibility of future USART implementations. None of the error flags can generate interrupts.

The Frame Error (FEn) flag indicates the state of the first stop bit of the next readable frame stored in the receive buffer. The FEn flag is zero when the stop bit was correctly read (as one), and the FEn flag will be one when the stop bit was incorrect (zero). This flag can be used for detecting out-of-sync conditions, detecting break conditions and protocol handling. The FEn flag is not affected by the setting of the USBSn bit in UCSRnC since the Receiver ignores all, except for the first, stop bits. For compatibility with future devices, always set this bit to zero when writing to UCSRnA.

The Data OverRun (DORn) flag indicates data loss due to a receiver buffer full condition. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. If the DORn flag is set there was one or more serial frame lost between the frame last read from UDRn, and the next frame read from UDRn. For compatibility with future devices, always write this bit to zero when writing to UCSRnA. The DORn flag is cleared when the frame received was successfully moved from the Shift Register to the receive buffer.

The Parity Error (UPEn) Flag indicates that the next frame in the receive buffer had a Parity Error when received. If Parity Check is not enabled the UPEn bit will always be read zero. For compatibility with future devices, always set this bit to zero when writing to UCSRnA. For more details see “Parity Bit Calculation” on page 176 and “Parity Checker” on page 183.

### Parity Checker

The Parity Checker is active when the high USARTn Parity mode (UPMn1) bit is set. Type of Parity Check to be performed (odd or even) is selected by the UPMn0 bit. When enabled, the Parity Checker calculates the parity of the data bits in incoming frames and compares the result with the parity bit from the serial frame. The result of the check is stored in the receive buffer together with the received data and stop bits. The Parity Error (UPEn) flag can then be read by software to check if the frame had a Parity Error.

The UPEn bit is set if the next character that can be read from the receive buffer had a Parity Error when received and the Parity Checking was enabled at that point (UPMn1 = 1). This bit is valid until the receive buffer (UDRn) is read.

### Disabling the Receiver

In contrast to the Transmitter, disabling of the Receiver will be immediate. Data from ongoing receptions will therefore be lost. When disabled (i.e., the RXENn is set to zero) the Receiver will no longer override the normal function of the RxDn port pin. The Receiver buffer FIFO will be flushed when the Receiver is disabled. Remaining data in the buffer will be lost

### Flushing the Receive Buffer

The receiver buffer FIFO will be flushed when the Receiver is disabled, i.e., the buffer will be emptied of its contents. Unread data will be lost. If the buffer has to be flushed during normal operation, due to for instance an error condition, read the UDRn I/O location until the RXCn flag is cleared.

The following code example shows how to flush the receive buffer.

Assembly Code Example[1]

```
USART0_Flush:
  lds   r16, UCSR0A
  sbrs  r16, RXC0
  ret
  lds   r16, UDR0
  rjmp  USART0_Flush
```

C Code Example[1]

```
void USART0_Flush (void )
{
  unsigned char dummy;
  while (UCSR0A & (1<<RXC0) ) dummy = UDR0;
}
```

Note: 1. The example code assumes that the part specific header file is included.

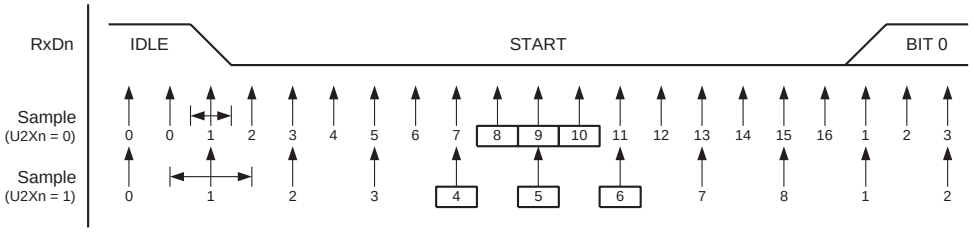
## Asynchronous Data Reception

The USARTn includes a clock recovery and a data recovery unit for handling asynchronous data reception. The clock recovery logic is used for synchronizing the internally generated baud rate clock to the incoming asynchronous serial frames at the RxDn pin. The data recovery logic samples and low pass filters each incoming bit, thereby improving the noise immunity of the Receiver. The asynchronous reception operational range depends on the accuracy of the internal baud rate clock, the rate of the incoming frames, and the frame size in number of bits.

### Asynchronous Clock Recovery

The clock recovery logic synchronizes internal clock to the incoming serial frames. Figure 87 illustrates the sampling process of the start bit of an incoming frame. The sample rate is 16 times the baud rate for Normal mode, and eight times the baud rate for Double Speed mode. The horizontal arrows illustrate the synchronization variation due to the sampling process. Note the larger time variation when using the Double Speed mode (U2Xn = 1) of operation. Samples denoted zero are samples done when the RxDn line is idle (i.e., no communication activity).

**Figure 87.** Start Bit Sampling

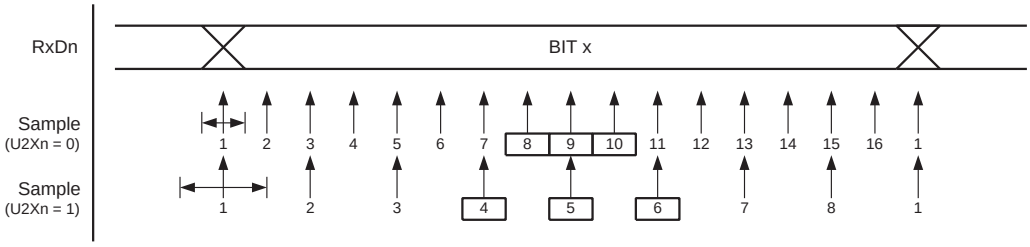



When the clock recovery logic detects a high (idle) to low (start) transition on the RxDn line, the start bit detection sequence is initiated. Let sample 1 denote the first zero-sample as shown in the figure. The clock recovery logic then uses samples 8, 9, and 10 for Normal mode, and samples 4, 5, and 6 for Double Speed mode (indicated with sample numbers inside boxes on the figure), to decide if a valid start bit is received. If two or more of these three samples have logical high levels (the majority wins), the start bit is rejected as a noise spike and the Receiver starts looking for the next high to low-transition. If however, a valid start bit is detected, the clock recovery logic is synchronized and the data recovery can begin. The synchronization process is repeated for each start bit.

**Asynchronous Data Recovery**

When the receiver clock is synchronized to the start bit, the data recovery can begin. The data recovery unit uses a state machine that has 16 states for each bit in Normal mode and eight states for each bit in Double Speed mode. Figure 88 shows the sampling of the data bits and the parity bit. Each of the samples is given a number that is equal to the state of the recovery unit.

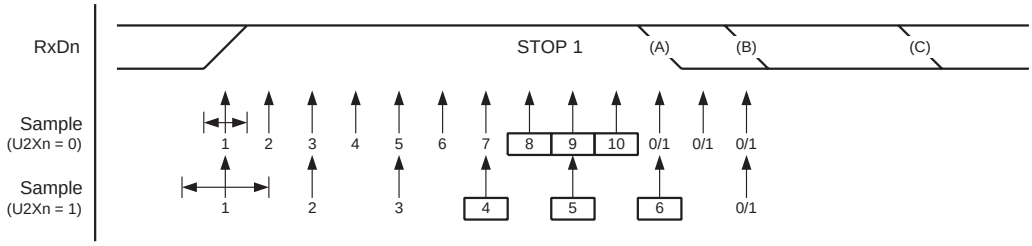
**Figure 88.** Sampling of Data and Parity Bit




The decision of the logic level of the received bit is taken by doing a majority voting of the logic value to the three samples in the center of the received bit. The center samples are emphasized on the figure by having the sample number inside boxes. The majority voting process is done as follows: If two or all three samples have high levels, the received bit is registered to be a logic 1. If two or all three samples have low levels, the received bit is registered to be a logic 0. This majority voting process acts as a low pass filter for the incoming signal on the RxDn pin. The recovery process is then repeated until a complete frame is received. Including the first stop bit. Note that the Receiver only uses the first stop bit of a frame.

Figure 89 shows the sampling of the stop bit and the earliest possible beginning of the start bit of the next frame.

**Figure 89.** Stop Bit Sampling and Next Start Bit Sampling




The same majority voting is done to the stop bit as done for the other bits in the frame. If the stop bit is registered to have a logic 0 value, the Frame Error (FEn) flag will be set.

A new high to low transition indicating the start bit of a new frame can come right after the last of the bits used for majority voting. For Normal Speed mode, the first low level sample can be at point marked (A) in Figure 89. For Double Speed mode the first low level must be delayed to (B). (C) marks a stop bit of full length. The early start bit detection influences the operational range of the Receiver.

### Asynchronous Operational Range

The operational range of the Receiver is dependent on the mismatch between the received bit rate and the internally generated baud rate. If the Transmitter is sending frames at too fast or too slow bit rates, or the internally generated baud rate of the Receiver does not have a similar (see Table 77) base frequency, the Receiver will not be able to synchronize the frames to the start bit.

The following equations can be used to calculate the ratio of the incoming data rate and internal receiver baud rate.

$$R_{slow} = \frac{(D+1)S}{S-1+D\cdot S+S_F} \qquad R_{fast} = \frac{(D+2)S}{(D+1)S+S_M}$$

**D** Sum of character size and parity size (D = 5 to 10 bit)

**S** Samples per bit. S = 16 for Normal Speed mode and S = 8 for Double Speed mode.

**$S_F$** First sample number used for majority voting. $S_F$ = 8 for normal speed and $S_F$ = 4 for Double Speed mode.

**$S_M$** Middle sample number used for majority voting. $S_M$ = 9 for normal speed and $S_M$ = 5 for Double Speed mode.

**$R_{slow}$** is the ratio of the slowest incoming data rate that can be accepted in relation to the receiver baud rate.

**$R_{fast}$** is the ratio of the fastest incoming data rate that can be accepted in relation to the receiver baud rate.

Table 77 and Table 78 list the maximum receiver baud rate error that can be tolerated. Note that Normal Speed mode has higher toleration of baud rate variations.

**Table 77.** Recommended Maximum Receiver Baud Rate Error for Normal Speed Mode (U2Xn = 0)

| D # (Data+Parity Bit) | $R_{slow}$ (%) | $R_{fast}$ (%) | Max Total Error (%) | Recommended Max Receiver Error (%) |
|---|---|---|---|---|
| 5 | 93.20 | 106.67 | +6.67/-6.8 | ± 3.0 |
| 6 | 94.12 | 105.79 | +5.79/-5.88 | ± 2.5 |
| 7 | 94.81 | 105.11 | +5.11/-5.19 | ± 2.0 |
| 8 | 95.36 | 104.58 | +4.58/-4.54 | ± 2.0 |
| 9 | 95.81 | 104.14 | +4.14/-4.19 | ± 1.5 |
| 10 | 96.17 | 103.78 | +3.78/-3.83 | ± 1.5 |

**Table 78.** Recommended Maximum Receiver Baud Rate Error for Double Speed Mode (U2Xn = 1)

| D # (Data+Parity Bit) | $R_{slow}$ (%) | $R_{fast}$ (%) | Max Total Error (%) | Recommended Max Receiver Error (%) |
|---|---|---|---|---|
| 5 | 94.12 | 105.66 | +5.66/-5.88 | ± 2.5 |
| 6 | 94.92 | 104.92 | +4.92/-5.08 | ± 2.0 |
| 7 | 95.52 | 104,35 | +4.35/-4.48 | ± 1.5 |
| 8 | 96.00 | 103.90 | +3.90/-4.00 | ± 1.5 |
| 9 | 96.39 | 103.53 | +3.53/-3.61 | ± 1.5 |
| 10 | 96.70 | 103.23 | +3.23/-3.30 | ± 1.0 |

The recommendations of the maximum receiver baud rate error was made under the assumption that the Receiver and Transmitter equally divides the maximum total error.

There are two possible sources for the receivers baud rate error. The Receiver's system clock (XTAL) will always have some minor instability over the supply voltage range and the temperature range. When using a crystal to generate the system clock, this is rarely a problem, but for a resonator the system clock may differ more than 2% depending of the resonators tolerance. The second source for the error is more controllable. The baud rate generator can not always do an exact division of the system frequency to get the baud rate wanted. In this case an UBRRn value that gives an acceptable low error can be used if possible.

## Multi-processor Communication Mode

Setting the Multi-processor Communication mode (MPCMn) bit in UCSRnA enables a filtering function of incoming frames received by the USARTn Receiver. Frames that do not contain address information will be ignored and not put into the receive buffer. This effectively reduces the number of incoming frames that has to be handled by the CPU, in a system with multiple MCUs that communicate via the same serial bus. The Transmitter is unaffected by the MPCMn setting, but has to be used differently when it is a part of a system utilizing the Multi-processor Communication mode.

### MPCM Protocol

If the Receiver is set up to receive frames that contain 5 to 8 data bits, then the first stop bit indicates if the frame contains data or address information. If the Receiver is set up for frames with nine data bits, then the ninth bit (RXB8n) is used for identifying address

and data frames. When the frame type bit (the first stop or the ninth bit) is one, the frame contains an address. When the frame type bit is zero the frame is a data frame.

The Multi-processor Communication mode enables several slave MCUs to receive data from a master MCU. This is done by first decoding an address frame to find out which MCU has been addressed. If a particular slave MCU has been addressed, it will receive the following data frames as normal, while the other slave MCUs will ignore the received frames until another address frame is received.

**Using MPCM**

For an MCU to act as a master MCU, it can use a 9-bit character frame format (UCSZn = 7). The ninth bit (TXB8n) must be set when an address frame (TXB8n = 1) or cleared when a data frame (TXBn = 0) is being transmitted. The slave MCUs must in this case be set to use a 9-bit character frame format.

The following procedure should be used to exchange data in Multi-processor Communication mode:

1. All Slave MCUs are in Multi-processor Communication mode (MPCMn in UCSRnA is set).
2. The Master MCU sends an address frame, and all slaves receive and read this frame. In the Slave MCUs, the RXCn flag in UCSRnA will be set as normal.
3. Each Slave MCU reads the UDRn Register and determines if it has been selected. If so, it clears the MPCMn bit in UCSRnA, otherwise it waits for the next address byte and keeps the MPCMn setting.
4. The addressed MCU will receive all data frames until a new address frame is received. The other Slave MCUs, which still have the MPCMn bit set, will ignore the data frames.
5. When the last data frame is received by the addressed MCU, the addressed MCU sets the MPCMn bit and waits for a new address frame from master. The process then repeats from 2.

Using any of the 5- to 8-bit character frame formats is possible, but impractical since the Receiver must change between using N and N+1 character frame formats. This makes full-duplex operation difficult since the Transmitter and Receiver use the same character size setting. If 5- to 8-bit character frames are used, the Transmitter must be set to use two stop bit (USBSn = 1) since the first stop bit is used for indicating the frame type.

## USART Register Description

### USART0 I/O Data Register – UDR0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | RXB0[7:0] | | | | | | | | UDR0 (Read) |
| | TXB0[7:0] | | | | | | | | UDR0 (Write) |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### USART1 I/O Data Register – UDR1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | RXB1[7:0] | | | | | | | | UDR1 (Read) |
| | TXB1[7:0] | | | | | | | | UDR1 (Write) |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7:0 – RxBn7:0: Receive Data Buffer** (read access)
- **Bit 7:0 – TxBn7:0: Transmit Data Buffer** (write access)

The USARTn Transmit Data Buffer Register and USARTn Receive Data Buffer Registers share the same I/O address referred to as USARTn Data Register or UDRn. The Transmit Data Buffer Register (TXBn) will be the destination for data written to the UDRn Register location. Reading the UDRn Register location will return the contents of the Receive Data Buffer Register (RXBn).

For 5-, 6-, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

The transmit buffer can only be written when the UDREn flag in the UCSRnA Register is set. Data written to UDRn when the UDREn flag is not set, will be ignored by the USARTn Transmitter. When data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the Transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TxDn pin.

The receive buffer consists of a two level FIFO. The FIFO will change its state whenever the receive buffer is accessed.

### USART0 Control and Status Register A – UCSR0A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | RXC0 | TXC0 | UDRE0 | FE0 | DOR0 | UPE0 | U2X0 | MPCM0 | UCSR0A |
| Read/Write | R | R/W | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |

### USART1 Control and Status Register A – UCSR1A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | RXC1 | TXC1 | UDRE1 | FE1 | DOR1 | UPE1 | U2X1 | MPCM1 | UCSR1A |
| Read/Write | R | R/W | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – RXCn: USARTn Receive Complete**

This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled, the receive buffer will be flushed and consequently the RXCn bit will become zero. The RXCn flag can be used to generate a Receive Complete interrupt (see description of the RXCIEn bit).

- **Bit 6 – TXCn: USARTn Transmit Complete**

This flag bit is set when the entire frame in the Transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer (UDRn). The TXCn flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXCn flag can generate a Transmit Complete interrupt (see description of the TXCIEn bit).

- **Bit 5 – UDREn: USARTn Data Register Empty**

The UDREn flag indicates if the transmit buffer (UDRn) is ready to receive new data. If UDREn is one, the buffer is empty, and therefore ready to be written. The UDREn flag can generate a Data Register Empty interrupt (see description of the UDRIEn bit).

UDREn is set after a reset to indicate that the Transmitter is ready.

- **Bit 4 – FEn: Frame Error**

This bit is set if the next character in the receive buffer had a Frame Error when received. I.e., when the first stop bit of the next character in the receive buffer is zero. This bit is valid until the receive buffer (UDRn) is read. The FEn bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSRnA.

- **Bit 3 – DORn: Data OverRun**

This bit is set if a Data OverRun condition is detected. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. This bit is valid until the receive buffer (UDRn) is read. Always set this bit to zero when writing to UCSRnA.

- **Bit 2 – UPEn: USARTn Parity Error**

This bit is set if the next character in the receive buffer had a Parity Error when received and the Parity Checking was enabled at that point (UPMn1 = 1). This bit is valid until the receive buffer (UDRn) is read. Always set this bit to zero when writing to UCSRnA.

- **Bit 1 – U2Xn: Double the USARTn Transmission Speed**

This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation.

Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8 effectively doubling the transfer rate for asynchronous communication.

- **Bit 0 – MPCMn: Multi-processor Communication Mode**

This bit enables the Multi-processor Communication mode. When the MPCMn bit is written to one, all the incoming frames received by the USARnT Receiver that do not contain address information will be ignored. The Transmitter is unaffected by the MPCMn setting. For more detailed information see “Multi-processor Communication Mode” on page 187.

**USART0 Control and Status Register B – UCSR0B**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **RXCIE0** | **TXCIE0** | **UDRIE0** | **RXEN0** | **TXEN0** | **UCSZ02** | **RXB80** | **TXB80** | **UCSR0B** |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### USART1 Control and Status Register B – UCSR1B

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | RXCIE1 | TXCIE1 | UDRIE1 | RXEN1 | TXEN1 | UCSZ12 | RXB81 | TXB81 | UCSR1B |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – RXCIEn: RX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the RXCn flag. A USARTn Receive Complete interrupt will be generated only if the RXCIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXCn bit in UCSRnA is set.

- **Bit 6 – TXCIEn: TX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the TXCn flag. A USARTn Transmit Complete interrupt will be generated only if the TXCIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXCn bit in UCSRnA is set.

- **Bit 5 – UDRIEn: USARTn Data Register Empty Interrupt Enable**

Writing this bit to one enables interrupt on the UDREn flag. A Data Register Empty interrupt will be generated only if the UDRIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDREn bit in UCSRnA is set.

- **Bit 4 – RXENn: Receiver Enable**

Writing this bit to one enables the USARTn Receiver. The Receiver will override normal port operation for the RxDn pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FEn, DORn, and UPEn Flags.

- **Bit 3 – TXENn: Transmitter Enable**

Writing this bit to one enables the USARTn Transmitter. The Transmitter will override normal port operation for the TxDn pin when enabled. The disabling of the Transmitter (writing TXENn to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer override the TxDn port.

- **Bit 2 – UCSZn2: Character Size**

The UCSZn2 bits combined with the UCSZn1:0 bit in UCSRnC sets the number of data bits (Character SiZe) in a frame the Receiver and Transmitter use.

- **Bit 1 – RXB8n: Receive Data Bit 8**

RXB8n is the ninth data bit of the received character when operating with serial frames with nine data bits. Must be read before reading the low bits from UDRn.

- **Bit 0 – TXB8n: Transmit Data Bit 8**

TXB8n is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. Must be written before writing the low bits to UDRn.

### USART0 Control and Status Register C – UCSR0C

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | UMSEL0 | UPM01 | UPM00 | USBS0 | UCSZ01 | UCSZ00 | UCPOL0 | UCSR0C |
| Read/Write | R | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Initial Value | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | |

### USART1 Control and Status Register C – UCSR1C

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | UMSEL1 | UPM11 | UPM10 | USBS1 | UCSZ11 | UCSZ10 | UCPO1L | UCSR1C |
| Read/Write | R | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | |

- **Bit 7 – Reserved Bit**

This bit is reserved for future use. For compatibility with future devices, these bit must be written to zero when UCSRnC is written.

- **Bit 6 – UMSELn: USARTn Mode Select**

This bit selects between asynchronous and synchronous mode of operation.

**Table 79.** UMSELn Bit Settings

| UMSELn | Mode |
|---|---|
| 0 | Asynchronous Operation |
| 1 | Synchronous Operation |

- **Bit 5:4 – UPMn1:0: Parity Mode**

These bits enable and set type of parity generation and check. If enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the UPMn0 setting. If a mismatch is detected, the UPEn Flag in UCSRnA will be set.

**Table 80.** UPMn Bits Settings

| UPMn1 | UPMn0 | Parity Mode |
|---|---|---|
| 0 | 0 | Disabled |
| 0 | 1 | Reserved |
| 1 | 0 | Enabled, Even Parity |
| 1 | 1 | Enabled, Odd Parity |

- **Bit 3 – USBSn: Stop Bit Select**

This bit selects the number of stop bits to be inserted by the Transmitter. The Receiver ignores this setting.

**Table 81.** USBSn Bit Settings

| USBSn | Stop Bit(s) |
|---|---|
| 0 | 1-bit |
| 1 | 2-bit |

- **Bit 2:1 – UCSZn1:0: Character Size**

The UCSZn1:0 bits combined with the UCSZn2 bit in UCSRnB sets the number of data bits (Character SiZe) in a frame the Receiver and Transmitter use.

**Table 82.** UCSZn Bits Settings

| UCSZn2 | UCSZn1 | UCSZn0 | Character Size |
|---|---|---|---|
| 0 | 0 | 0 | 5-bit |
| 0 | 0 | 1 | 6-bit |
| 0 | 1 | 0 | 7-bit |
| 0 | 1 | 1 | 8-bit |
| 1 | 0 | 0 | Reserved |
| 1 | 0 | 1 | Reserved |
| 1 | 1 | 0 | Reserved |
| 1 | 1 | 1 | 9-bit |

- **Bit 0 – UCPOLn: Clock Polarity**

This bit is used for synchronous mode only. Write this bit to zero when asynchronous mode is used. The UCPOLn bit sets the relationship between data output change and data input sample, and the synchronous clock (XCKn).

**Table 83.** UCPOLn Bit Settings

| UCPOLn | Transmitted Data Changed (Output of TxDn Pin) | Received Data Sampled (Input on RxDn Pin) |
|---|---|---|
| 0 | Rising XCK Edge | Falling XCK Edge |
| 1 | Falling XCK Edge | Rising XCK Edge |

### USART0 Baud Rate Registers – UBRR0L and UBRR0H

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | UBRR0[11:8] | | | | UBRR0H |
| | UBRR0[7:0] | | | | | | | | UBRR0L |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Read/Write | R | R | R | R | R/W | R/W | R/W | R/W | |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### USART1 Baud Rate Registers – UBRR1L and UBRR1H

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | UBRR1[11:8] | | | | UBRR1H |
| | UBRR1[7:0] | | | | | | | | UBRR1L |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Read/Write | R | R | R | R | R/W | R/W | R/W | R/W | |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 15:12 – Reserved Bits**

These bits are reserved for future use. For compatibility with future devices, these bit must be written to zero when UBRRnH is written.

- **Bit 11:0 – UBRRn11:0: USARTn Baud Rate Register**

This is a 12-bit register which contains the USARTn baud rate. The UBRRnH contains the four most significant bits, and the UBRRnL contains the eight least significant bits of the USARTn baud rate. Ongoing transmissions by the Transmitter and Receiver will be corrupted if the baud rate is changed. Writing UBRRnL will trigger an immediate update of the baud rate prescaler.

## Examples of Baud Rate Setting

For standard crystal, resonator and external oscillator frequencies, the most commonly used baud rates for asynchronous operation can be generated by using the UBRRn settings in Table 84 up to Table 87. UBRRn values which yield an actual baud rate differing less than 0.5% from the target baud rate, are bold in the table. Higher error ratings are acceptable, but the Receiver will have less noise resistance when the error ratings are high, especially for large serial frames (see “Asynchronous Operational Range” on page 186). The error values are calculated using the following equation:

$$\text{Error}[\%] = \left(1 - \frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}}\right) \bullet 100\%$$

**Table 84.** Examples of UBRRn Settings for Commonly Frequencies

| Baud Rate (bps) | $f_{clk_{io}}$ = 1.0000 MHz | | | | $f_{clk_{io}}$ = 1.8432 MHz | | | | $f_{clk_{io}}$ = 2.0000 MHz | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | |
| | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error |
| 2400 | 25 | 0.2% | 51 | 0.2% | 47 | 0.0% | 95 | 0.0% | 51 | 0.2% | 103 | 0.2% |
| 4800 | 12 | 0.2% | 25 | 0.2% | 23 | 0.0% | 47 | 0.0% | 25 | 0.2% | 51 | 0.2% |
| 9600 | ***6*** | ***-7.0%*** | 12 | 0.2% | 11 | 0.0% | 23 | 0.0% | 12 | 0.2% | 25 | 0.2% |
| 14.4k | ***3*** | ***8.5%*** | ***8*** | ***-3.5%*** | 7 | 0.0% | 15 | 0.0% | ***8*** | ***-3.5%*** | ***16*** | ***2.1%*** |
| 19.2k | ***2*** | ***8.5%*** | ***6*** | ***-7.0%*** | 5 | 0.0% | 11 | 0.0% | ***6*** | ***-7.0%*** | 12 | 0.2% |
| 28.8k | ***1*** | ***8.5%*** | ***3*** | ***8.5%*** | 3 | 0.0% | 7 | 0.0% | ***3*** | ***8.5%*** | ***8*** | ***-3.5%*** |
| 38.4k | ***1*** | ***-18.6%*** | ***2*** | ***8.5%*** | 2 | 0.0% | 5 | 0.0% | ***2*** | ***8.5%*** | ***6*** | ***-7.0%*** |
| 57.6k | ***0*** | ***8.5%*** | ***1*** | ***8.5%*** | 1 | 0.0% | 3 | 0.0% | ***1*** | ***8.5%*** | ***3*** | ***8.5%*** |
| 76.8k | – | – | ***1*** | ***-18.6%*** | ***1*** | ***-25.0%*** | 2 | 0.0% | ***1*** | ***-18.6%*** | ***2*** | ***8.5%*** |
| 115.2k | – | – | ***0*** | ***8.5%*** | 0 | 0.0% | 1 | 0.0% | ***0*** | ***8.5%*** | ***1*** | ***8.5%*** |
| 230.4k | – | – | – | – | – | – | 0 | 0.0% | – | – | – | – |
| 250k | – | – | – | – | – | – | – | – | – | – | – | – |
| 500k | – | – | – | – | – | – | – | – | – | – | – | – |
| 1M | – | – | – | – | – | – | – | – | – | – | – | – |
| Max. (1) | 62.5 kbps | | 125 kbps | | 115.2 kbps | | 230.4 Kbps | | 125 kpbs | | 250 kbps | |

1. UBRRn = 0, Error = 0.0%

**Table 85.** Examples of UBRRn Settings for Commonly Frequencies (Continued)

| Baud Rate (bps) | $f_{clk_{io}}$ = 3.6864 MHz | | | | $f_{clk_{io}}$ = 4.0000 MHz | | | | $f_{clk_{io}}$ = 7.3728 MHz | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | |
| | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error |
| 2400 | 95 | 0.0% | 191 | 0.0% | 103 | 0.2% | 207 | 0.2% | 191 | 0.0% | 383 | 0.0% |
| 4800 | 47 | 0.0% | 95 | 0.0% | 51 | 0.2% | 103 | 0.2% | 95 | 0.0% | 191 | 0.0% |
| 9600 | 23 | 0.0% | 47 | 0.0% | 25 | 0.2% | 51 | 0.2% | 47 | 0.0% | 95 | 0.0% |
| 14.4k | 15 | 0.0% | 31 | 0.0% | ***16*** | ***2.1%*** | ***34*** | ***-0.8%*** | 31 | 0.0% | 63 | 0.0% |
| 19.2k | 11 | 0.0% | 23 | 0.0% | 12 | 0.2% | 25 | 0.2% | 23 | 0.0% | 47 | 0.0% |
| 28.8k | 7 | 0.0% | 15 | 0.0% | ***8*** | ***-3.5%*** | ***16*** | ***2.1%*** | 15 | 0.0% | 31 | 0.0% |
| 38.4k | 5 | 0.0% | 11 | 0.0% | ***6*** | ***-7.0%*** | 12 | 0.2% | 11 | 0.0% | 23 | 0.0% |
| 57.6k | 3 | 0.0% | 7 | 0.0% | ***3*** | ***8.5%*** | ***8*** | ***-3.5%*** | 7 | 0.0% | 15 | 0.0% |
| 76.8k | 2 | 0.0% | 5 | 0.0% | ***2*** | ***8.5%*** | ***6*** | ***-7.0%*** | 5 | 0.0% | 11 | 0.0% |
| 115.2k | 1 | 0.0% | 3 | 0.0% | ***1*** | ***8.5%*** | ***3*** | ***8.5%*** | 3 | 0.0% | 7 | 0.0% |
| 230.4k | 0 | 0.0% | 1 | 0.0% | ***0*** | ***8.5%*** | ***1*** | ***8.5%*** | 1 | 0.0% | 3 | 0.0% |
| 250k | ***0*** | ***-7.8%*** | ***1*** | ***-7.8%*** | 0 | 0.0% | 1 | 0.0% | ***1*** | ***-7.8%*** | ***3*** | ***-7.8%*** |
| 500k | – | – | ***0*** | ***-7.8%*** | – | – | 0 | 0.0% | ***0*** | ***-7.8%*** | ***1*** | ***-7.8%*** |
| 1M | – | – | – | – | – | – | – | – | – | – | ***0*** | ***-7.8%*** |
| Max. (1) | 230.4 kbps | | 460.8 kbps | | 250 kbps | | 0.5 Mbps | | 460.8 kpbs | | 921.6 kbps | |

1. UBRRn = 0, Error = 0.0%

**Table 86.** Examples of UBRRn Settings for Commonly Frequencies (Continued)

| Baud Rate (bps) | $f_{clk_{io}}$ = 8.0000 MHz | | | | $f_{clk_{io}}$ = 10.000 MHz | | | | $f_{clk_{io}}$ = 11.0592 MHz | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | |
| | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error |
| 2400 | 207 | 0.2% | 416 | -0.1% | 259 | 0.2% | 520 | 0.0% | 287 | 0.0% | 575 | 0.0% |
| 4800 | 103 | 0.2% | 207 | 0.2% | 129 | 0.2% | 259 | 0.2% | 143 | 0.0% | 287 | 0.0% |
| 9600 | 51 | 0.2% | 103 | 0.2% | 64 | 0.2% | 129 | 0.2% | 71 | 0.0% | 143 | 0.0% |
| 14.4k | ***34*** | ***-0.8%*** | ***68*** | ***0.6%*** | ***42*** | ***0.9%*** | 86 | 0.2% | 47 | 0.0% | 95 | 0.0% |
| 19.2k | 25 | 0.2% | 51 | 0.2% | ***32*** | ***-1.4%*** | 64 | 0.2% | 35 | 0.0% | 71 | 0.0% |
| 28.8k | ***16*** | ***2.1%*** | ***34*** | ***-0.8%*** | ***21*** | ***-1.4%*** | ***42*** | ***0.9%*** | 23 | 0.0% | 47 | 0.0% |
| 38.4k | 12 | 0.2% | 25 | 0.2% | ***15*** | ***1.8%*** | ***32*** | ***-1.4%*** | 17 | 0.0% | 35 | 0.0% |
| 57.6k | ***8*** | ***-3.5%*** | ***16*** | ***2.1%*** | ***10*** | ***-1.5%*** | ***21*** | ***-1.4%*** | 11 | 0.0% | 23 | 0.0% |
| 76.8k | ***6*** | ***-7.0%*** | 12 | 0.2% | ***7*** | ***1.9%*** | ***15*** | ***1.8%*** | 8 | 0.0% | 17 | 0.0% |
| 115.2k | ***3*** | ***8.5%*** | ***8*** | ***-3.5%*** | ***4*** | ***9.6%*** | ***10*** | ***-1.5%*** | 5 | 0.0% | 11 | 0.0% |
| 230.4k | ***1*** | ***8.5%*** | ***3*** | ***8.5%*** | ***2*** | ***-16.8%*** | ***4*** | ***9.6%*** | 2 | 0.0% | 5 | 0.0% |
| 250k | 1 | 0.0% | 3 | 0.0% | ***2*** | ***-33.3%*** | 4 | 0.0% | ***2*** | ***-7.8%*** | ***5*** | ***-7.8%*** |
| 500k | 0 | 0.0% | 1 | 0.0% | – | – | ***2*** | ***-33.3%*** | – | – | ***2*** | ***-7.8%*** |
| 1M | – | – | 0 | 0.0% | – | – | – | – | – | – | – | – |
| Max. (1) | 0.5 Mbps | | 1 Mbps | | 625 kbps | | 1.25 Mbps | | 691.2 kbps | | 1.3824 Mbps | |

1. UBRRn = 0, Error = 0.0%

**Table 87.** Examples of UBRRn Settings for Commonly Frequencies (Continued)

| Baud Rate (bps) | $f_{clk_{io}}$ = 12.0000 MHz | | | | $f_{clk_{io}}$ = 14.7456 MHz | | | | $f_{clk_{io}}$ = 16.0000 MHz | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | |
| | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error |
| 2400 | 312 | -0.2% | 624 | 0.0% | 383 | 0.0% | 767 | 0.0% | 416 | -0.1% | 832 | 0.0% |
| 4800 | 155 | 0.2% | 312 | -0.2% | 191 | 0.0% | 383 | 0.0% | 207 | 0.2% | 416 | -0.1% |
| 9600 | 77 | 0.2% | 155 | 0.2% | 95 | 0.0% | 191 | 0.0% | 103 | 0.2% | 207 | 0.2% |
| 14.4k | 51 | 0.2% | 103 | 0.2% | 63 | 0.0% | 127 | 0.0% | ***68*** | ***0.6%*** | 138 | -0.1% |
| 19.2k | 38 | 0.2% | 77 | 0.2% | 47 | 0.0% | 95 | 0.0% | 51 | 0.2% | 103 | 0.2% |
| 28.8k | 25 | 0.2% | 51 | 0.2% | 31 | 0.0% | 63 | 0.0% | ***34*** | ***-0.8%*** | ***68*** | ***0.6%*** |
| 38.4k | ***19*** | ***-2.5%*** | 38 | 0.2% | 23 | 0.0% | 47 | 0.0% | 25 | 0.2% | 51 | 0.2% |
| 57.6k | 12 | 0.2% | 25 | 0.2% | 15 | 0.0% | 31 | 0.0% | ***16*** | ***2.1%*** | ***34*** | ***-0.8%*** |
| 76.8k | ***9*** | ***-2.7%*** | ***19*** | ***-2.5%*** | 11 | 0.0% | 23 | 0.0% | 12 | 0.2% | 25 | 0.2% |
| 115.2k | ***6*** | ***-8.9%*** | 12 | 0.2% | 7 | 0.0% | 15 | 0.0% | ***8*** | ***-3.5%*** | ***16*** | ***2.1%*** |
| 230.4k | ***2*** | ***11.3%*** | ***6*** | ***-8.9%*** | 3 | 0.0% | 7 | 0.0% | ***3*** | ***8.5%*** | ***8*** | ***-3.5%*** |
| 250k | 2 | 0.0% | 5 | 0.0% | ***3*** | ***-7.8%*** | ***6*** | ***5.3%*** | 3 | 0.0% | 7 | 0.0% |
| 500k | – | – | 2 | 0.0% | ***1*** | ***-7.8%*** | ***3*** | ***-7.8%*** | 1 | 0.0% | 3 | 0.0% |
| 1M | – | – | – | – | ***0*** | ***-7.8%*** | ***1*** | ***-7.8%*** | 0 | 0.0% | 1 | 0.0% |
| Max. (1) | 750 kbps | | 1.5 Mbps | | 921.6 kbps | | 1.8432 Mbps | | 1 Mbps | | 2 Mbps | |

1. UBRRn = 0, Error = 0.0%

# Two-wire Serial Interface

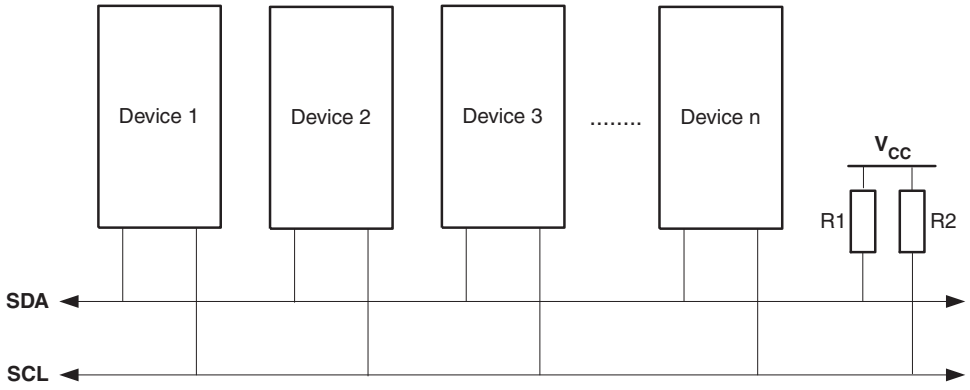
## Features

- **Simple yet Powerful and Flexible Communication Interface, only Two Bus Lines Needed**
- **Both Master and Slave Operation Supported**
- **Device can Operate as Transmitter or Receiver**
- **7-bit Address Space allows up to 128 Different Slave Addresses**
- **Multi-master Arbitration Support**
- **Up to 400 kHz Data Transfer Speed**
- **Slew-rate Limited Output Drivers**
- **Noise Suppression Circuitry Rejects Spikes on Bus Lines**
- **Fully Programmable Slave Address with General Call Support**
- **Address Recognition Causes Wake-up when AVR is in Sleep Mode**

## Two-wire Serial Interface Bus Definition

The Two-wire Serial Interface (TWI) is ideally suited for typical microcontroller applications. The TWI protocol allows the systems designer to interconnect up to 128 different devices using only two bi-directional bus lines, one for clock (SCL) and one for data (SDA). The only external hardware needed to implement the bus is a single pull-up resistor for each of the TWI bus lines. All devices connected to the bus have individual addresses, and mechanisms for resolving bus contention are inherent in the TWI protocol.

**Figure 90.** TWI Bus Interconnection




### TWI Terminology

The following definitions are frequently encountered in this section.

**Table 88.** TWI Terminology

| Term | Description |
|---|---|
| Master | The device that initiates and terminates a transmission. The master also generates the SCL clock |
| Slave | The device addressed by a master |
| Transmitter | The device placing data on the bus |
| Receiver | The device reading data from the bus |

### Electrical Interconnection

As depicted in Figure 90, both bus lines are connected to the positive supply voltage through pull-up resistors. The bus drivers of all TWI-compliant devices are open-drain or open-collector. This implements a wired-AND function which is essential to the operation of the interface. A low level on a TWI bus line is generated when one or more TWI devices output a zero. A high level is output when all TWI devices tri-state their outputs,

allowing the pull-up resistors to pull the line high. Note that all AVR devices connected to the TWI bus must be powered in order to allow any bus operation.

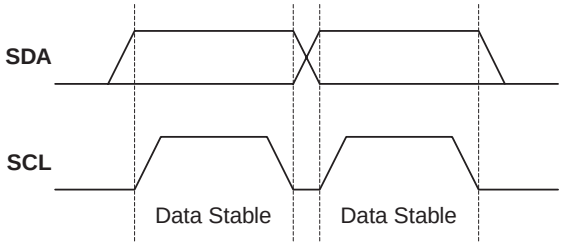
The number of devices that can be connected to the bus is only limited by the bus capacitance limit of 400 pF and the 7-bit slave address space. A detailed specification of the electrical characteristics of the TWI is given in “Two-wire Serial Interface Characteristics” on page 359. Two different sets of specifications are presented there, one relevant for bus speeds below 100 kHz, and one valid for bus speeds up to 400 kHz.

## Data Transfer and Frame Format

### Transferring Bits

Each data bit transferred on the TWI bus is accompanied by a pulse on the clock line. The level of the data line must be stable when the clock line is high. The only exception to this rule is for generating start and stop conditions.

**Figure 91.** Data Validity

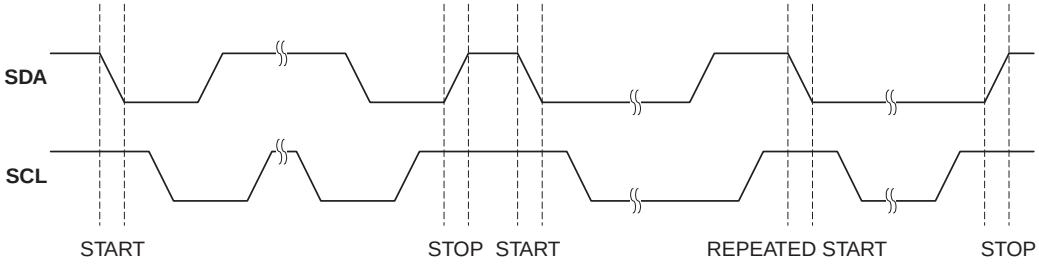



**Data Change**

### START and STOP Conditions

The master initiates and terminates a data transmission. The transmission is initiated when the master issues a START condition on the bus, and it is terminated when the master issues a STOP condition. Between a START and a STOP condition, the bus is considered busy, and no other master should try to seize control of the bus. A special case occurs when a new START condition is issued between a START and STOP condition. This is referred to as a REPEATED START condition, and is used when the master wishes to initiate a new transfer without relinquishing control of the bus. After a REPEATED START, the bus is considered busy until the next STOP. This is identical to the START behaviour, and therefore START is used to describe both START and REPEATED START for the remainder of this datasheet, unless otherwise noted. As depicted below, START and STOP conditions are signalled by changing the level of the SDA line when the SCL line is high.

**Figure 92.** START, REPEATED START and STOP Conditions



### Address Packet Format

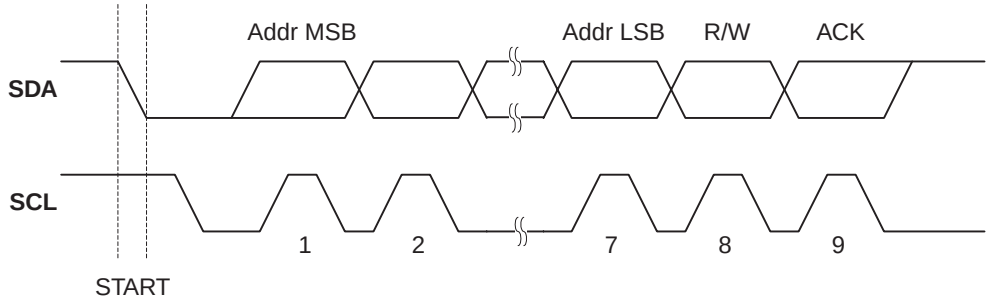
All address packets transmitted on the TWI bus are 9 bits long, consisting of 7 address bits, one READ/WRITE control bit and an acknowledge bit. If the READ/WRITE bit is set, a read operation is to be performed, otherwise a write operation should be performed. When a slave recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle. If the addressed slave is busy, or for some other reason can not service the master's request, the SDA line should be left high in the ACK clock cycle. The master can then transmit a STOP condition, or a REPEATED START condition to initiate a new transmission. An address packet consisting of a slave address and a READ or a WRITE bit is called SLA+R or SLA+W, respectively.

The MSB of the address byte is transmitted first. Slave addresses can freely be allocated by the designer, but the address 0000 000 is reserved for a general call.

When a general call is issued, all slaves should respond by pulling the SDA line low in the ACK cycle. A general call is used when a master wishes to transmit the same message to several slaves in the system. When the general call address followed by a Write bit is transmitted on the bus, all slaves set up to acknowledge the general call will pull the SDA line low in the ack cycle. The following data packets will then be received by all the slaves that acknowledged the general call. Note that transmitting the general call address followed by a Read bit is meaningless, as this would cause contention if several slaves started transmitting different data.

All addresses of the format 1111 xxx should be reserved for future purposes.

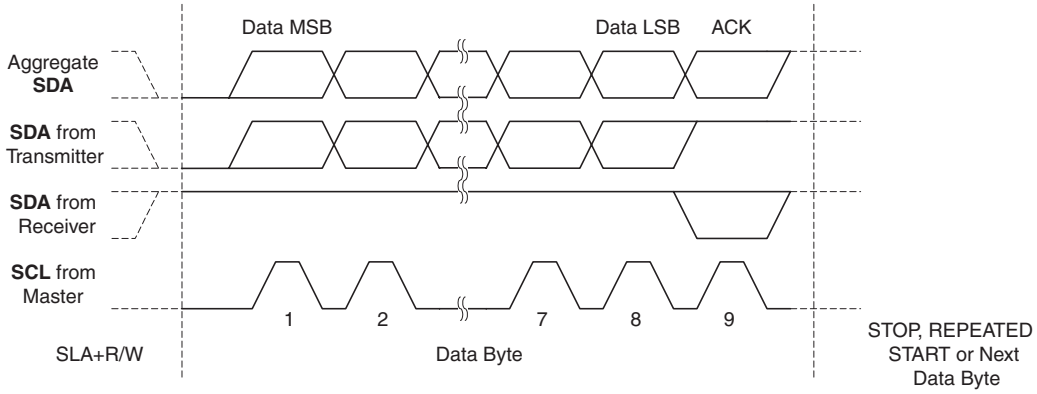
**Figure 93.** Address Packet Format




### Data Packet Format

All data packets transmitted on the TWI bus are 9 bits long, consisting of one data byte and an acknowledge bit. During a data transfer, the master generates the clock and the START and STOP conditions, while the receiver is responsible for acknowledging the reception. An Acknowledge (ACK) is signalled by the receiver pulling the SDA line low during the ninth SCL cycle. If the receiver leaves the SDA line high, a NACK is signalled. When the receiver has received the last byte, or for some reason cannot receive any more bytes, it should inform the transmitter by sending a NACK after the final byte. The MSB of the data byte is transmitted first.

**Figure 94.** Data Packet Format

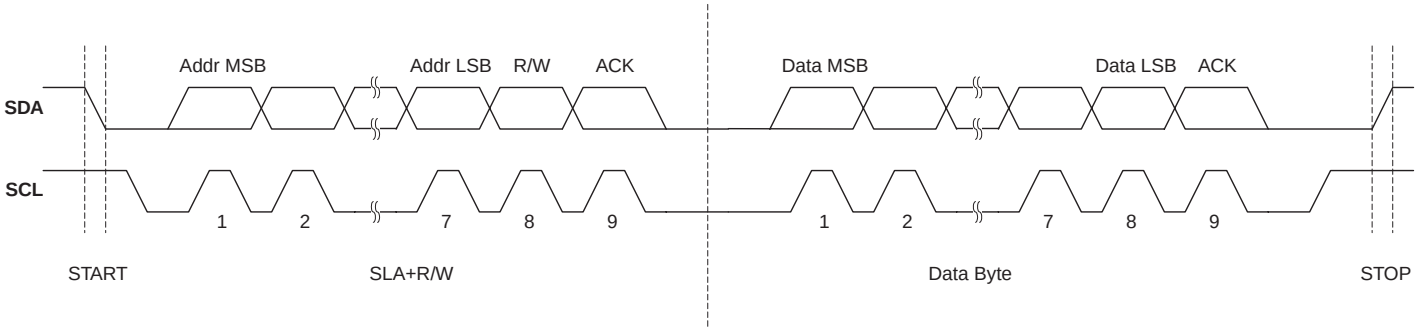



### Combining Address and Data Packets Into a Transmission

A transmission basically consists of a START condition, a SLA+R/W, one or more data packets and a STOP condition. An empty message, consisting of a START followed by a STOP condition, is illegal. Note that the Wired-ANDing of the SCL line can be used to implement handshaking between the master and the slave. The slave can extend the SCL low period by pulling the SCL line low. This is useful if the clock speed set up by the master is too fast for the slave, or the slave needs extra time for processing between the data transmissions. The slave extending the SCL low period will not affect the SCL high period, which is determined by the master. As a consequence, the slave can reduce the TWI data transfer speed by prolonging the SCL duty cycle.

Figure 95 shows a typical data transmission. Note that several data bytes can be transmitted between the SLA+R/W and the STOP condition, depending on the software protocol implemented by the application software.

**Figure 95.** Typical Data Transmission




### Multi-master Bus Systems, Arbitration and Synchronization

The TWI protocol allows bus systems with several masters. Special concerns have been taken in order to ensure that transmissions will proceed as normal, even if two or more masters initiate a transmission at the same time. Two problems arise in multi-master systems:

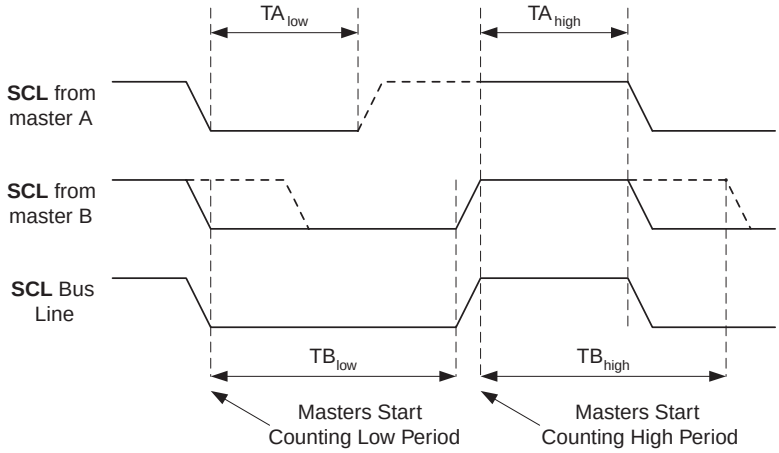
- An algorithm must be implemented allowing only one of the masters to complete the transmission. All other masters should cease transmission when they discover that they have lost the selection process. This selection process is called arbitration. When a contending master discovers that it has lost the arbitration process, it should immediately switch to slave mode to check whether it is being addressed by the winning master. The fact that multiple masters have started transmission at the

same time should not be detectable to the slaves, i.e., the data being transferred on the bus must not be corrupted.

- Different masters may use different SCL frequencies. A scheme must be devised to synchronize the serial clocks from all masters, in order to let the transmission proceed in a lockstep fashion. This will facilitate the arbitration process.

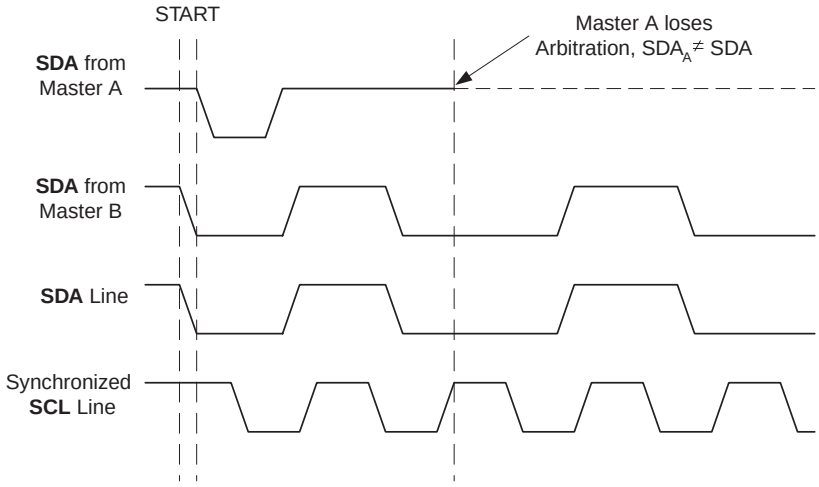
The wired-ANDing of the bus lines is used to solve both these problems. The serial clocks from all masters will be wired-ANDed, yielding a combined clock with a high period equal to the one from the master with the shortest high period. The low period of the combined clock is equal to the low period of the master with the longest low period. Note that all masters listen to the SCL line, effectively starting to count their SCL high and low time-out periods when the combined SCL line goes high or low, respectively.

**Figure 96.** SCL Synchronization between Multiple Masters




Arbitration is carried out by all masters continuously monitoring the SDA line after outputting data. If the value read from the SDA line does not match the value the master had output, it has lost the arbitration. Note that a master can only lose arbitration when it outputs a high SDA value while another master outputs a low value. The losing master should immediately go to slave mode, checking if it is being addressed by the winning master. The SDA line should be left high, but losing masters are allowed to generate a clock signal until the end of the current data or address packet. Arbitration will continue until only one master remains, and this may take many bits. If several masters are trying to address the same slave, arbitration will continue into the data packet.

**Figure 97.** Arbitration Between two Masters




Note that arbitration is not allowed between:

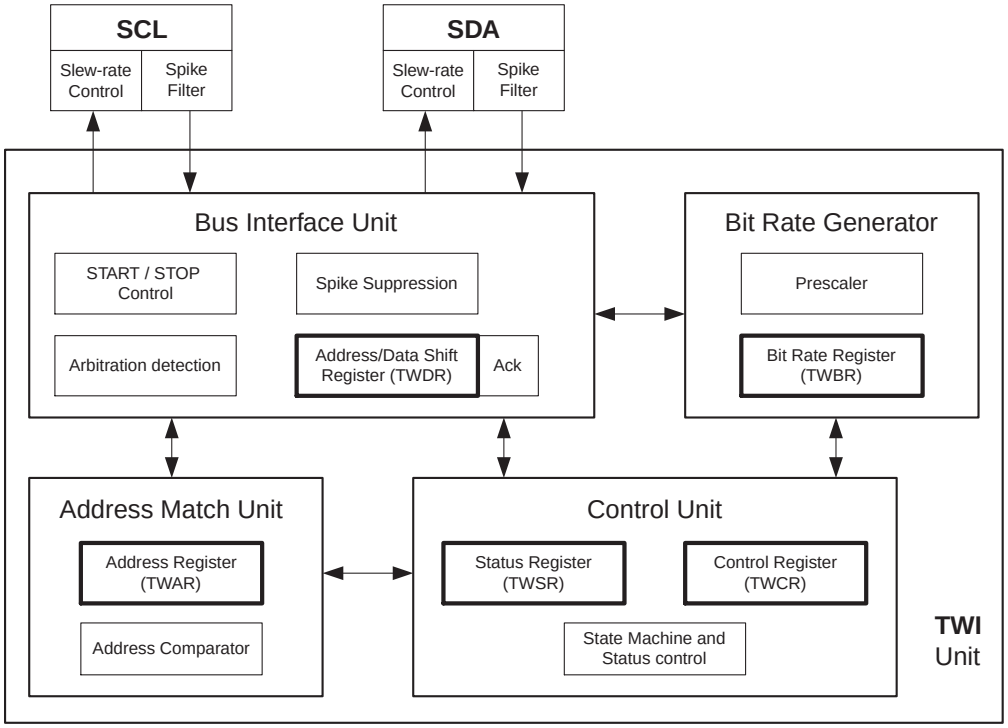
- A REPEATED START condition and a data bit
- A STOP condition and a data bit
- A REPEATED START and a STOP condition

It is the user software's responsibility to ensure that these illegal arbitration conditions never occur. This implies that in multi-master systems, all data transfers must use the same composition of SLA+R/W and data packets. In other words: All transmissions must contain the same number of data packets, otherwise the result of the arbitration is undefined.

## Overview of the TWI Module

The TWI module is comprised of several submodules, as shown in Figure 98. All registers drawn in a thick line are accessible through the AVR data bus.

**Figure 98.** Overview of the TWI Module




### Scl and SDA Pins

These pins interface the AVR TWI with the rest of the MCU system. The output drivers contain a slew-rate limiter in order to conform to the TWI specification. The input stages contain a spike suppression unit removing spikes shorter than 50 ns. Note that the internal pullups in the AVR pads can be enabled by setting the PORT bits corresponding to the SCL and SDA pins, as explained in the I/O Port section. The internal pull-ups can in some systems eliminate the need for external ones.

### Bit Rate Generator Unit

This unit controls the period of SCL when operating in a Master mode. The SCL period is controlled by settings in the TWI Bit Rate Register (TWBR) and the Prescaler bits in the TWI Status Register (TWSR). Slave operation does not depend on Bit Rate or Prescaler settings, but the CPU clock frequency in the slave must be at least 16 times higher than the SCL frequency. Note that slaves may prolong the SCL low period, thereby reducing the average TWI bus clock period. The SCL frequency is generated according to the following equation:

$$\text{SCL frequency} = \frac{\text{CLKio}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

- TWBR = Value of the TWI Bit Rate Register
- TWPS = Value of the prescaler bits in the TWI Status Register

Note: TWBR should be 10 or higher if the TWI operates in Master mode. If TWBR is lower than 10, the master may produce an incorrect output on SDA and SCL for the reminder of the byte. The problem occurs when operating the TWI in Master mode, sending Start + SLA + R/W to a slave (a slave does not need to be connected to the bus for the condition to happen).

**Bus Interface Unit**

This unit contains the Data and Address Shift Register (TWDR), a START/STOP Controller and Arbitration detection hardware. The TWDR contains the address or data bytes to be transmitted, or the address or data bytes received. In addition to the 8-bit TWDR, the Bus Interface Unit also contains a register containing the (N)ACK bit to be transmitted or received. This (N)ACK Register is not directly accessible by the application software. However, when receiving, it can be set or cleared by manipulating the TWI Control Register (TWCR). When in Transmitter mode, the value of the received (N)ACK bit can be determined by the value in the TWSR.

The START/STOP Controller is responsible for generation and detection of START, REPEATED START, and STOP conditions. The START/STOP controller is able to detect START and STOP conditions even when the AVR MCU is in one of the sleep modes, enabling the MCU to wake up if addressed by a master.

If the TWI has initiated a transmission as master, the Arbitration Detection hardware continuously monitors the transmission trying to determine if arbitration is in process. If the TWI has lost an arbitration, the Control Unit is informed. Correct action can then be taken and appropriate status codes generated.

**Address Match Unit**

The Address Match unit checks if received address bytes match the 7-bit address in the TWI Address Register (TWAR). If the TWI General Call Recognition Enable (TWGCE) bit in the TWAR is written to one, all incoming address bits will also be compared against the General Call address. Upon an address match, the Control Unit is informed, allowing correct action to be taken. The TWI may or may not acknowledge its address, depending on settings in the TWCR. The Address Match unit is able to compare addresses even when the AVR MCU is in sleep mode, enabling the MCU to wake up if addressed by a master. If another interrupt (e.g., INT0) occurs during TWI Power-down address match and wakes up the CPU, the TWI aborts operation and return to it's idle state. If this cause any problems, ensure that TWI Address Match is the only enabled interrupt when entering Power-down.

**Control Unit**

The Control unit monitors the TWI bus and generates responses corresponding to settings in the TWI Control Register (TWCR). When an event requiring the attention of the application occurs on the TWI bus, the TWI Interrupt Flag (TWINT) is asserted. In the next clock cycle, the TWI Status Register (TWSR) is updated with a status code identifying the event. The TWSR only contains relevant status information when the TWI Interrupt Flag is asserted. At all other times, the TWSR contains a special status code indicating that no relevant status information is available. As long as the TWINT flag is set, the SCL line is held low. This allows the application software to complete its tasks before allowing the TWI transmission to continue.

The TWINT flag is set in the following situations:

- After the TWI has transmitted a START/REPEATED START condition
- After the TWI has transmitted SLA+R/W
- After the TWI has transmitted an address byte
- After the TWI has lost arbitration
- After the TWI has been addressed by own slave address or general call
- After the TWI has received a data byte
- After a STOP or REPEATED START has been received while still addressed as a slave
- When a bus error has occurred due to an illegal START or STOP condition

## TWI Register Description

### TWI Bit Rate Register – TWBR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TWBR7 | TWBR6 | TWBR5 | TWBR4 | TWBR3 | TWBR2 | TWBR1 | TWBR0 | TWBR |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bits 7..0 – TWI Bit Rate Register**

TWBR selects the division factor for the bit rate generator. The bit rate generator is a frequency divider which generates the SCL clock frequency in the Master modes. See “Bit Rate Generator Unit” on page 205 for calculating bit rates.

### TWI Control Register – TWCR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | – | TWIE | TWCR |
| Read/Write | R/W | R/W | R/W | R/W | R | R/W | R | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

The TWCR is used to control the operation of the TWI. It is used to enable the TWI, to initiate a master access by applying a START condition to the bus, to generate a receiver acknowledge, to generate a stop condition, and to control halting of the bus while the data to be written to the bus are written to the TWDR. It also indicates a write collision if data is attempted written to TWDR while the register is inaccessible.

- **Bit 7 – TWINT: TWI Interrupt Flag**

This bit is set by hardware when the TWI has finished its current job and expects application software response. If the I-bit in SREG and TWIE in TWCR are set, the MCU will jump to the TWI interrupt vector. While the TWINT flag is set, the SCL low period is stretched. The TWINT flag must be cleared by software by writing a logic one to it. Note that this flag is not automatically cleared by hardware when executing the interrupt routine. Also note that clearing this flag starts the operation of the TWI, so all accesses to the TWI Address Register (TWAR), TWI Status Register (TWSR), and TWI Data Register (TWDR) must be complete before clearing this flag.

- **Bit 6 – TWEA: TWI Enable Acknowledge Bit**

The TWEA bit controls the generation of the ACK pulse. If the TWEA bit is written to one, the ACK pulse is generated on the TWI bus if the following conditions are met:

1. The device’s own slave address has been received.
2. A general call has been received, while the TWGCE bit in the TWAR is set.
3. A data byte has been received in Master Receiver or Slave Receiver mode.

By writing the TWEA bit to zero, the device can be virtually disconnected from the Two-wire Serial Bus temporarily. Address recognition can then be resumed by writing the TWEA bit to one again.

- **Bit 5 – TWSTA: TWI START Condition Bit**

The application writes the TWSTA bit to one when it desires to become a master on the Two-wire Serial Bus. The TWI hardware checks if the bus is available, and generates a START condition on the bus if it is free. However, if the bus is not free, the TWI waits until a STOP condition is detected, and then generates a new START condition to claim

the bus Master status. TWSTA must be cleared by software when the START condition has been transmitted.

- **Bit 4 – TWSTO: TWI STOP Condition Bit**

Writing the TWSTO bit to one in Master mode will generate a STOP condition on the Two-wire Serial Bus. When the STOP condition is executed on the bus, the TWSTO bit is cleared automatically. In slave mode, setting the TWSTO bit can be used to recover from an error condition. This will not generate a STOP condition, but the TWI returns to a well-defined unaddressed Slave mode and releases the SCL and SDA lines to a high impedance state.

- **Bit 3 – TWWC: TWI Write Collision Flag**

The TWWC bit is set when attempting to write to the TWI Data Register – TWDR when TWINT is low. This flag is cleared by writing the TWDR Register when TWINT is high.

- **Bit 2 – TWEN: TWI Enable Bit**

The TWEN bit enables TWI operation and activates the TWI interface. When TWEN is written to one, the TWI takes control over the I/O pins connected to the SCL and SDA pins, enabling the slew-rate limiters and spike filters. If this bit is written to zero, the TWI is switched off and all TWI transmissions are terminated, regardless of any ongoing operation.

- **Bit 1 – Reserved Bit**

This bit is reserved for future use. For compatibility with future devices, this must be written to zero when TWCR is written.

- **Bit 0 – TWIE: TWI Interrupt Enable**

When this bit is written to one, and the I-bit in SREG is set, the TWI interrupt request will be activated for as long as the TWINT flag is high.

**TWI Status Register – TWSR**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **TWS7** | **TWS6** | **TWS5** | **TWS4** | **TWS3** | **–** | **TWPS1** | **TWPS0** | **TWSR** |
| Read/Write | R | R | R | R | R | R | R/W | R/W | |
| Initial Value | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | |

- **Bits 7..3 – TWS: TWI Status**

These 5 bits reflect the status of the TWI logic and the Two-wire Serial Bus. The different status codes are described later in this section. Note that the value read from TWSR contains both the 5-bit status value and the 2-bit prescaler value. The application designer should mask the prescaler bits to zero when checking the Status bits. This makes status checking independent of prescaler setting. This approach is used in this datasheet, unless otherwise noted.

- **Bit 2 – Res: Reserved Bit**

This bit is reserved and will always read as zero.

- **Bits 1..0 – TWPS: TWI Prescaler Bits**

These bits can be read and written, and control the bit rate prescaler.

**Table 89.** TWI Bit Rate Prescaler

| TWPS1 | TWPS0 | Prescaler Value |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 4 |
| 1 | 0 | 16 |
| 1 | 1 | 64 |

To calculate bit rates, see “Bit Rate Generator Unit” on page 205. The value of TWPS1..0 is used in the equation.

### TWI Data Register – TWDR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TWD7 | TWD6 | TWD5 | TWD4 | TWD3 | TWD2 | TWD1 | TWD0 | TWDR |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |

In Transmit mode, TWDR contains the next byte to be transmitted. In receive mode, the TWDR contains the last byte received. It is writable while the TWI is not in the process of shifting a byte. This occurs when the TWI interrupt flag (TWINT) is set by hardware. Note that the Data Register cannot be initialized by the user before the first interrupt occurs. The data in TWDR remains stable as long as TWINT is set. While data is shifted out, data on the bus is simultaneously shifted in. TWDR always contains the last byte present on the bus, except after a wake up from a sleep mode by the TWI interrupt. In this case, the contents of TWDR is undefined. In the case of a lost bus arbitration, no data is lost in the transition from Master to Slave. Handling of the ACK bit is controlled automatically by the TWI logic, the CPU cannot access the ACK bit directly.

- **Bits 7..0 – TWD: TWI Data Register**

These eight bits constitute the next data byte to be transmitted, or the latest data byte received on the TWI Serial Bus.

### TWI (Slave) Address Register – TWAR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TWA6 | TWA5 | TWA4 | TWA3 | TWA2 | TWA1 | TWA0 | TWGCE | TWAR |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | |

- **Bits 7..1 – TWA: TWI (Slave) Address Register**

These seven bits constitute the slave address of the TWI unit. The TWAR should be loaded with the 7-bit slave address to which the TWI will respond when programmed as a slave transmitter or receiver, and not needed in the master modes. In multimaster systems, TWAR must be set in masters which can be addressed as slaves by other masters.

- **Bit 0 – TWGCE: TWI General Call Recognition Enable Bit**

TWGCE is used to enable recognition of the general call address (0x00). There is an associated address comparator that looks for the slave address (or general call address if enabled) in the received serial address. If a match is found, an interrupt request is generated. If set, this bit enables the recognition of a General Call given over the TWI Serial Bus.

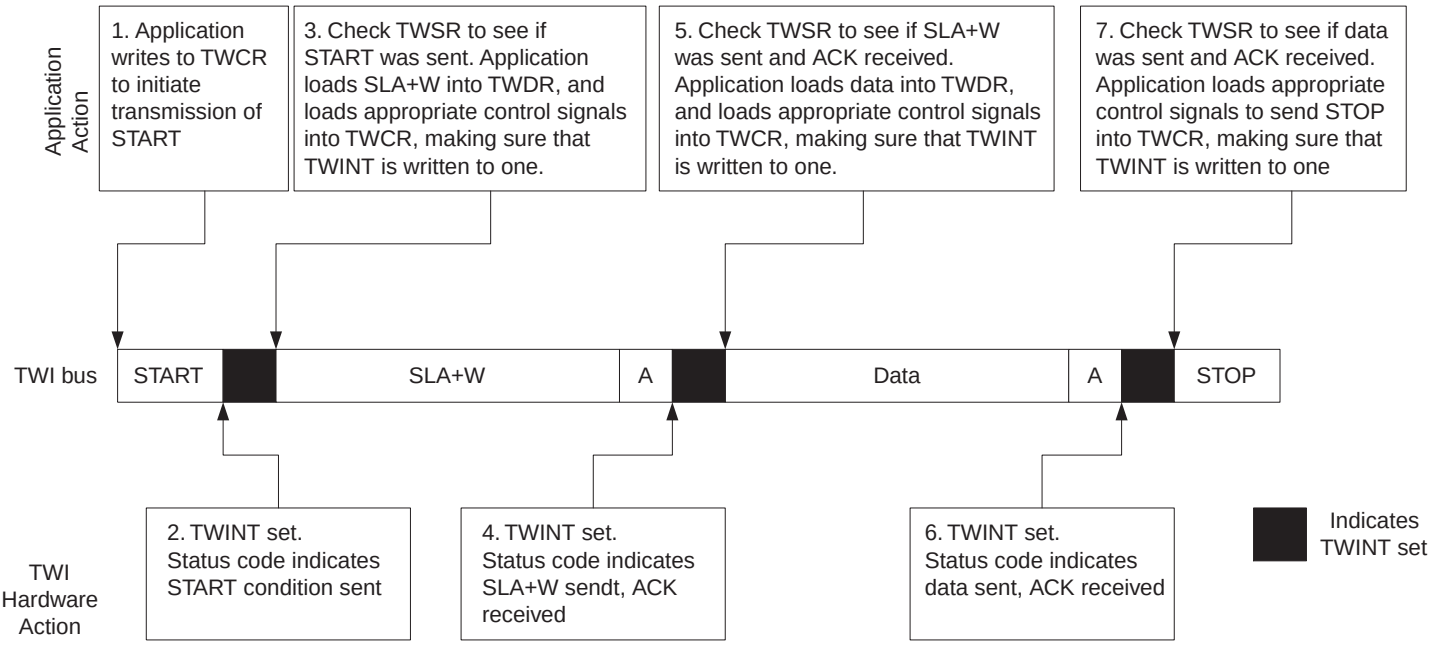
## Using the TWI

The AVR TWI is byte-oriented and interrupt based. Interrupts are issued after all bus events, like reception of a byte or transmission of a START condition. Because the TWI is interrupt-based, the application software is free to carry on other operations during a TWI byte transfer. Note that the TWI Interrupt Enable (TWIE) bit in TWCR together with the Global Interrupt Enable bit in SREG allow the application to decide whether or not assertion of the TWINT flag should generate an interrupt request. If the TWIE bit is cleared, the application must poll the TWINT flag in order to detect actions on the TWI bus.

When the TWINT flag is asserted, the TWI has finished an operation and awaits application response. In this case, the TWI Status Register (TWSR) contains a value indicating the current state of the TWI bus. The application software can then decide how the TWI should behave in the next TWI bus cycle by manipulating the TWCR and TWDR Registers.

Figure 99 is a simple example of how the application can interface to the TWI hardware. In this example, a master wishes to transmit a single data byte to a slave. This description is quite abstract, a more detailed explanation follows later in this section. A simple code example implementing the desired behaviour is also presented.

**Figure 99.** Interfacing the Application to the TWI in a Typical Transmission




1. The first step in a TWI transmission is to transmit a START condition. This is done by writing a specific value into TWCR, instructing the TWI hardware to transmit a START condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the START condition.
2. When the START condition has been transmitted, the TWINT flag in TWCR is set, and TWSR is updated with a status code indicating that the START condition has successfully been sent.

3. The application software should now examine the value of TWSR, to make sure that the START condition was successfully transmitted. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load SLA+W into TWDR. Remember that TWDR is used both for address and data. After TWDR has been loaded with the desired SLA+W, a specific value must be written to TWCR, instructing the TWI hardware to transmit the SLA+W present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the address packet.
4. When the address packet has been transmitted, the TWINT flag in TWCR is set, and TWSR is updated with a status code indicating that the address packet has successfully been sent. The status code will also reflect whether a slave acknowledged the packet or not.
5. The application software should now examine the value of TWSR, to make sure that the address packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load a data packet into TWDR. Subsequently, a specific value must be written to TWCR, instructing the TWI hardware to transmit the data packet present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the data packet.
6. When the data packet has been transmitted, the TWINT flag in TWCR is set, and TWSR is updated with a status code indicating that the data packet has successfully been sent. The status code will also reflect whether a slave acknowledged the packet or not.
7. The application software should now examine the value of TWSR, to make sure that the data packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must write a specific value to TWCR, instructing the TWI hardware to transmit a STOP condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the STOP condition. Note that TWINT is NOT set after a STOP condition has been sent.

Even though this example is simple, it shows the principles involved in all TWI transmissions. These can be summarized as follows:

- When the TWI has finished an operation and expects application response, the TWINT flag is set. The SCL line is pulled low until TWINT is cleared.
- When the TWINT flag is set, the user must update all TWI Registers with the value relevant for the next TWI bus cycle. As an example, TWDR must be loaded with the value to be transmitted in the next bus cycle.
- After all TWI Register updates and other pending application software tasks have been completed, TWCR is written. When writing TWCR, the TWINT bit should be

set. Writing a one to TWINT clears the flag. The TWI will then commence executing whatever operation was specified by the TWCR setting.

In the following an assembly and C implementation of the example is given. Note that the code below assumes that several definitions have been made for example by using include-files.

| | Assembly Code Example | C Example | Comments |
|---|---|---|---|
| 1 | `ldi r16, (1<<TWINT)\|(1<<TWSTA)\|`<br>`(1<<TWEN)`<br>`sts TWCR, r16` | `TWCR = (1<<TWINT)\|(1<<TWSTA)\|`<br>`(1<<TWEN)` | Send START condition |
| 2 | `wait1:`<br>`lds r16,TWCR`<br>`sbrs r16,TWINT`<br>`rjmp wait1` | `while (!(TWCR & (1<<TWINT)))`<br>`;` | Wait for TWINT flag set. This indicates that the START condition has been transmitted |
| 3 | `lds r16,TWSR`<br>`andi r16, 0xF8`<br>`cpi r16, START`<br>`brne ERROR` | `if ((TWSR & 0xF8) != START)`<br>`ERROR();` | Check value of TWI Status Register. Mask prescaler bits. If status different from START go to ERROR |
| | `ldi r16, SLA_W`<br>`sts TWDR, r16`<br>`ldi r16, (1<<TWINT) \| (1<<TWEN)`<br>`sts TWCR, r16` | `TWDR = SLA_W;`<br>`TWCR = (1<<TWINT) \| (1<<TWEN);` | Load SLA_W into TWDR Register. Clear TWINT bit in TWCR to start transmission of address |
| 4 | `wait2:`<br>`lds r16,TWCR`<br>`sbrs r16,TWINT`<br>`rjmp wait2` | `while (!(TWCR & (1<<TWINT)))`<br>`;` | Wait for TWINT flag set. This indicates that the SLA+W has been transmitted, and ACK/NACK has been received. |
| 5 | `lds r16,TWSR`<br>`andi r16, 0xF8`<br>`cpi r16, MT_SLA_ACK`<br>`brne ERROR` | `if ((TWSR & 0xF8) != MT_SLA_ACK)`<br>`ERROR();` | Check value of TWI Status Register. Mask prescaler bits. If status different from MT_SLA_ACK go to ERROR |
| | `ldi r16, DATA`<br>`sts TWDR, r16`<br>`ldi r16, (1<<TWINT) \| (1<<TWEN)`<br>`sts TWCR, r16` | `TWDR = DATA;`<br>`TWCR = (1<<TWINT) \| (1<<TWEN);` | Load DATA into TWDR Register. Clear TWINT bit in TWCR to start transmission of data |
| 6 | `wait3:`<br>`lds r16,TWCR`<br>`sbrs r16,TWINT`<br>`rjmp wait3` | `while (!(TWCR & (1<<TWINT)))`<br>`;` | Wait for TWINT flag set. This indicates that the DATA has been transmitted, and ACK/NACK has been received. |
| 7 | `lds r16,TWSR`<br>`andi r16, 0xF8`<br>`cpi r16, MT_DATA_ACK`<br>`brne ERROR` | `if ((TWSR & 0xF8) != MT_DATA_ACK)`<br>`ERROR();` | Check value of TWI Status Register. Mask prescaler bits. If status different from MT_DATA_ACK go to ERROR |
| | `ldi r16, (1<<TWINT)\|(1<<TWEN)\|`<br>`(1<<TWSTO)`<br>`sts TWCR, r16` | `TWCR = (1<<TWINT)\|(1<<TWEN)\|`<br>`(1<<TWSTO);` | Transmit STOP condition |

## Transmission Modes

The TWI can operate in one of four major modes. These are named Master Transmitter (MT), Master Receiver (MR), Slave Transmitter (ST) and Slave Receiver (SR). Several of these modes can be used in the same application. As an example, the TWI can use MT mode to write data into a TWI EEPROM, MR mode to read the data back from the EEPROM. If other masters are present in the system, some of these might transmit data to the TWI, and then SR mode would be used. It is the application software that decides which modes are legal.

The following sections describe each of these modes. Possible status codes are described along with figures detailing data transmission in each of the modes. These figures contain the following abbreviations:

**S**: START condition

**Rs**: REPEATED START condition

**R**: Read bit (high level at SDA)

**W**: Write bit (low level at SDA)

**A**: Acknowledge bit (low level at SDA)

$\overline{\textbf{A}}$: Not acknowledge bit (high level at SDA)

**Data**: 8-bit data byte

**P**: STOP condition

**SLA**: Slave Address

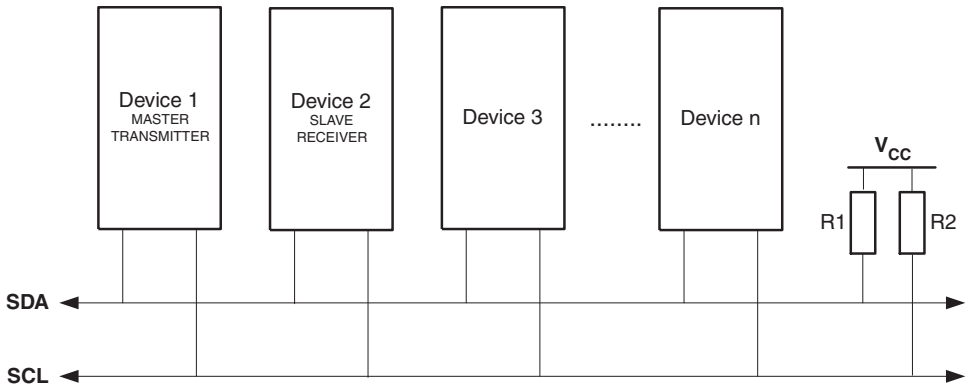
In Figure 101 to Figure 107, circles are used to indicate that the TWINT flag is set. The numbers in the circles show the status code held in TWSR, with the prescaler bits masked to zero. At these points, actions must be taken by the application to continue or complete the TWI transfer. The TWI transfer is suspended until the TWINT flag is cleared by software.

When the TWINT flag is set, the status code in TWSR is used to determine the appropriate software action. For each status code, the required software action and details of the following serial transfer are given in Table 90 to Table 93. Note that the prescaler bits are masked to zero in these tables.

### Master Transmitter Mode

In the Master Transmitter mode, a number of data bytes are transmitted to a slave receiver (see Figure 100). In order to enter a Master mode, a START condition must be transmitted. The format of the following address packet determines whether Master Transmitter or Master Receiver mode is to be entered. If SLA+W is transmitted, MT mode is entered, if SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

**Figure 100.** Data Transfer in Master Transmitter Mode




A START condition is sent by writing the following value to TWCR:

| TWCR | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | – | TWIE |
|---|---|---|---|---|---|---|---|---|
| value | 1 | X | 1 | 0 | X | 1 | 0 | X |

TWEN must be set to enable the Two-wire Serial Interface, TWSTA must be written to one to transmit a START condition and TWINT must be written to one to clear the TWINT flag. The TWI will then test the Two-wire Serial Bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT flag is set by hardware, and the status code in TWSR will be 0x08 (See Table 90). In order to enter MT mode, SLA+W must be transmitted. This is done by writing SLA+W to TWDR. Thereafter the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

| TWCR | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | – | TWIE |
|---|---|---|---|---|---|---|---|---|
| value | 1 | X | 0 | 0 | X | 1 | 0 | X |

When SLA+W have been transmitted and an acknowledgment bit has been received, TWINT is set again and a number of status codes in TWSR are possible. Possible status codes in Master mode are 0x18, 0x20, or 0x38. The appropriate action to be taken for each of these status codes is detailed in Table 90.

When SLA+W has been successfully transmitted, a data packet should be transmitted. This is done by writing the data byte to TWDR. TWDR must only be written when TWINT is high. If not, the access will be discarded, and the Write Collision bit (TWWC) will be set in the TWCR Register. After updating TWDR, the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

| TWCR | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | – | TWIE |
|---|---|---|---|---|---|---|---|---|
| value | 1 | X | 0 | 0 | X | 1 | 0 | X |

This scheme is repeated until the last byte has been sent and the transfer is ended by generating a STOP condition or a repeated START condition. A STOP condition is generated by writing the following value to TWCR:

| TWCR | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | – | TWIE |
|---|---|---|---|---|---|---|---|---|
| value | 1 | X | 0 | 1 | X | 1 | 0 | X |

A REPEATED START condition is generated by writing the following value to TWCR:

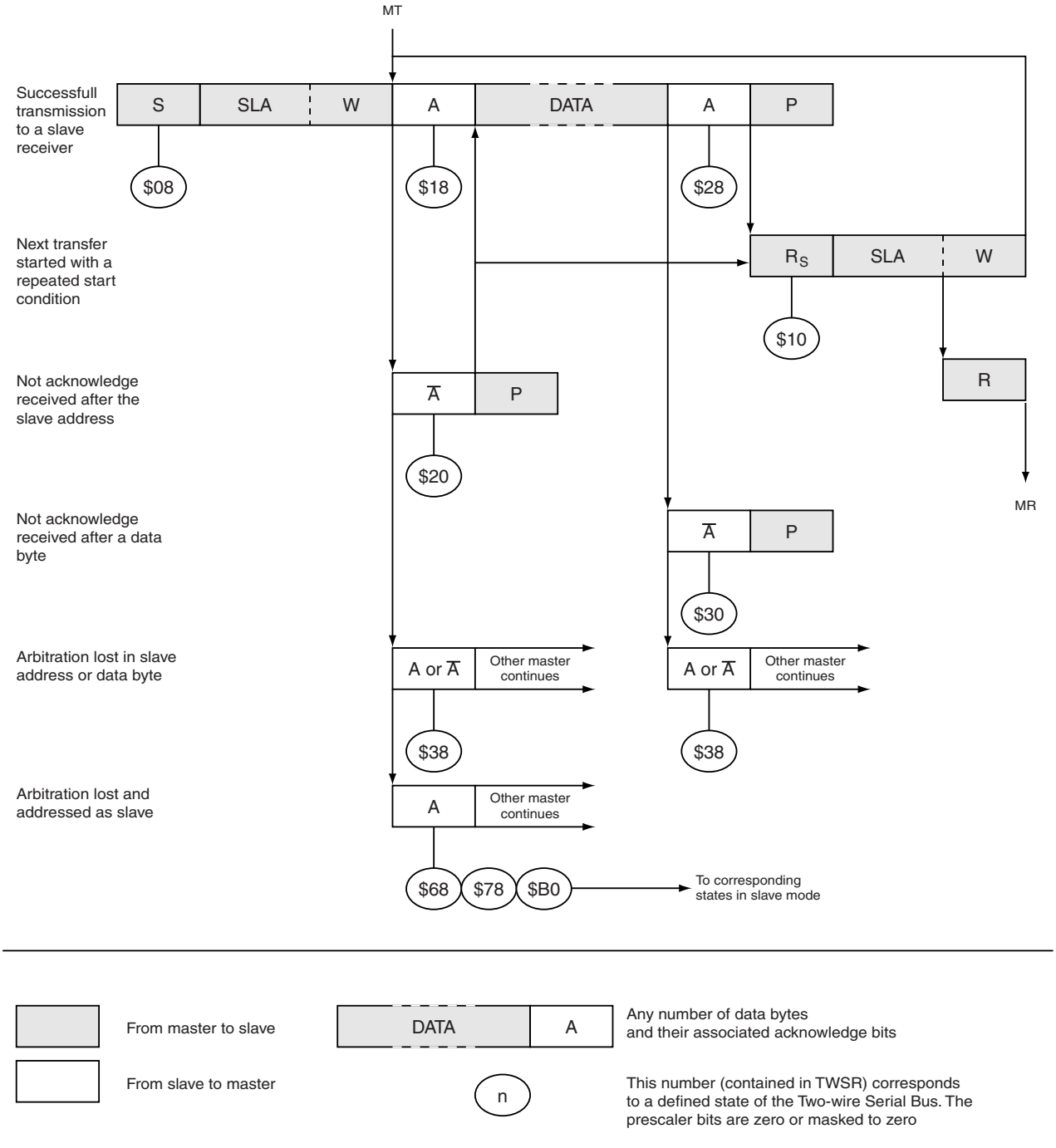
| TWCR | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | – | TWIE |
|---|---|---|---|---|---|---|---|---|
| value | 1 | X | 1 | 0 | X | 1 | 0 | X |

After a repeated START condition (state 0x10) the Two-wire Serial Interface can access the same slave again, or a new slave without transmitting a STOP condition. Repeated START enables the master to switch between slaves, Master Transmitter mode and Master Receiver mode without losing control of the bus.

**Table 90.** Status Codes for Master Transmitter Mode

| Status Code (TWSR) Prescaler Bits are 0 | Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware | Application Software Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWDR | To TWCR | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0x08 | A START condition has been transmitted | Load SLA+W | X | 0 | 1 | X | SLA+W will be transmitted;<br>ACK or NOT ACK will be received |
| 0x10 | A repeated START condition has been transmitted | Load SLA+W or | X | 0 | 1 | X | SLA+W will be transmitted;<br>ACK or NOT ACK will be received |
| | | Load SLA+R | X | 0 | 1 | X | SLA+R will be transmitted;<br>Logic will switch to master receiver mode |
| 0x18 | SLA+W has been transmitted; ACK has been received | Load data byte or | 0 | 0 | 1 | X | Data byte will be transmitted and ACK or NOT ACK will be received |
| | | No TWDR action or | 1 | 0 | 1 | X | Repeated START will be transmitted |
| | | No TWDR action or | 0 | 1 | 1 | X | STOP condition will be transmitted and TWSTO flag will be reset |
| | | No TWDR action | 1 | 1 | 1 | X | STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset |
| 0x20 | SLA+W has been transmitted; NOT ACK has been received | Load data byte or | 0 | 0 | 1 | X | Data byte will be transmitted and ACK or NOT ACK will be received |
| | | No TWDR action or | 1 | 0 | 1 | X | Repeated START will be transmitted |
| | | No TWDR action or | 0 | 1 | 1 | X | STOP condition will be transmitted and TWSTO flag will be reset |
| | | No TWDR action | 1 | 1 | 1 | X | STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset |
| 0x28 | Data byte has been transmitted; ACK has been received | Load data byte or | 0 | 0 | 1 | X | Data byte will be transmitted and ACK or NOT ACK will be received |
| | | No TWDR action or | 1 | 0 | 1 | X | Repeated START will be transmitted |
| | | No TWDR action or | 0 | 1 | 1 | X | STOP condition will be transmitted and TWSTO flag will be reset |
| | | No TWDR action | 1 | 1 | 1 | X | STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset |
| 0x30 | Data byte has been transmitted; NOT ACK has been received | Load data byte or | 0 | 0 | 1 | X | Data byte will be transmitted and ACK or NOT ACK will be received |
| | | No TWDR action or | 1 | 0 | 1 | X | Repeated START will be transmitted |
| | | No TWDR action or | 0 | 1 | 1 | X | STOP condition will be transmitted and TWSTO flag will be reset |
| | | No TWDR action | 1 | 1 | 1 | X | STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset |
| 0x38 | Arbitration lost in SLA+W or data bytes | No TWDR action or | 0 | 0 | 1 | X | Two-wire Serial Bus will be released and not addressed slave mode entered |
| | | No TWDR action | 1 | 0 | 1 | X | A START condition will be transmitted when the bus becomes free |

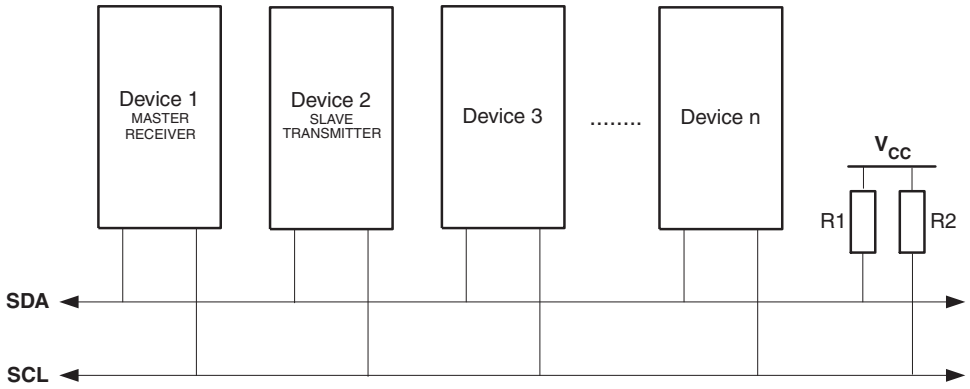
**Figure 101.** Formats and States in the Master Transmitter Mode



### Master Receiver Mode

In the Master Receiver Mode, a number of data bytes are received from a slave transmitter (see Figure 102). In order to enter a Master mode, a START condition must be transmitted. The format of the following address packet determines whether Master Transmitter or Master Receiver mode is to be entered. If SLA+W is transmitted, MT mode is entered, if SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

**Figure 102.** Data Transfer in Master Receiver Mode




A START condition is sent by writing the following value to TWCR:

| TWCR | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | – | TWIE |
|---|---|---|---|---|---|---|---|---|
| value | 1 | X | 1 | 0 | X | 1 | 0 | X |

TWEN must be written to one to enable the Two-wire Serial Interface, TWSTA must be written to one to transmit a START condition and TWINT must be set to clear the TWINT flag. The TWI will then test the Two-wire Serial Bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT flag is set by hardware, and the status code in TWSR will be 0x08 (See Table 90). In order to enter MR mode, SLA+R must be transmitted. This is done by writing SLA+R to TWDR. Thereafter the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

| TWCR | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | – | TWIE |
|---|---|---|---|---|---|---|---|---|
| value | 1 | X | 0 | 0 | X | 1 | 0 | X |

When SLA+R have been transmitted and an acknowledgment bit has been received, TWINT is set again and a number of status codes in TWSR are possible. Possible status codes in Master mode are 0x38, 0x40, or 0x48. The appropriate action to be taken for each of these status codes is detailed in Table 101. Received data can be read from the TWDR Register when the TWINT flag is set high by hardware. This scheme is repeated until the last byte has been received. After the last byte has been received, the MR should inform the ST by sending a NACK after the last received data byte. The transfer is ended by generating a STOP condition or a repeated START condition. A STOP condition is generated by writing the following value to TWCR:

| TWCR | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | – | TWIE |
|---|---|---|---|---|---|---|---|---|
| value | 1 | X | 0 | 1 | X | 1 | 0 | X |

A REPEATED START condition is generated by writing the following value to TWCR:

| TWCR | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | – | TWIE |
|---|---|---|---|---|---|---|---|---|
| value | 1 | X | 1 | 0 | X | 1 | 0 | X |

After a repeated START condition (state 0x10) the Two-wire Serial Interface can access the same slave again, or a new slave without transmitting a STOP condition. Repeated START enables the master to switch between slaves, Master Transmitter mode and Master Receiver mode without losing control over the bus.

**Figure 103.** Formats and States in the Master Receiver Mode

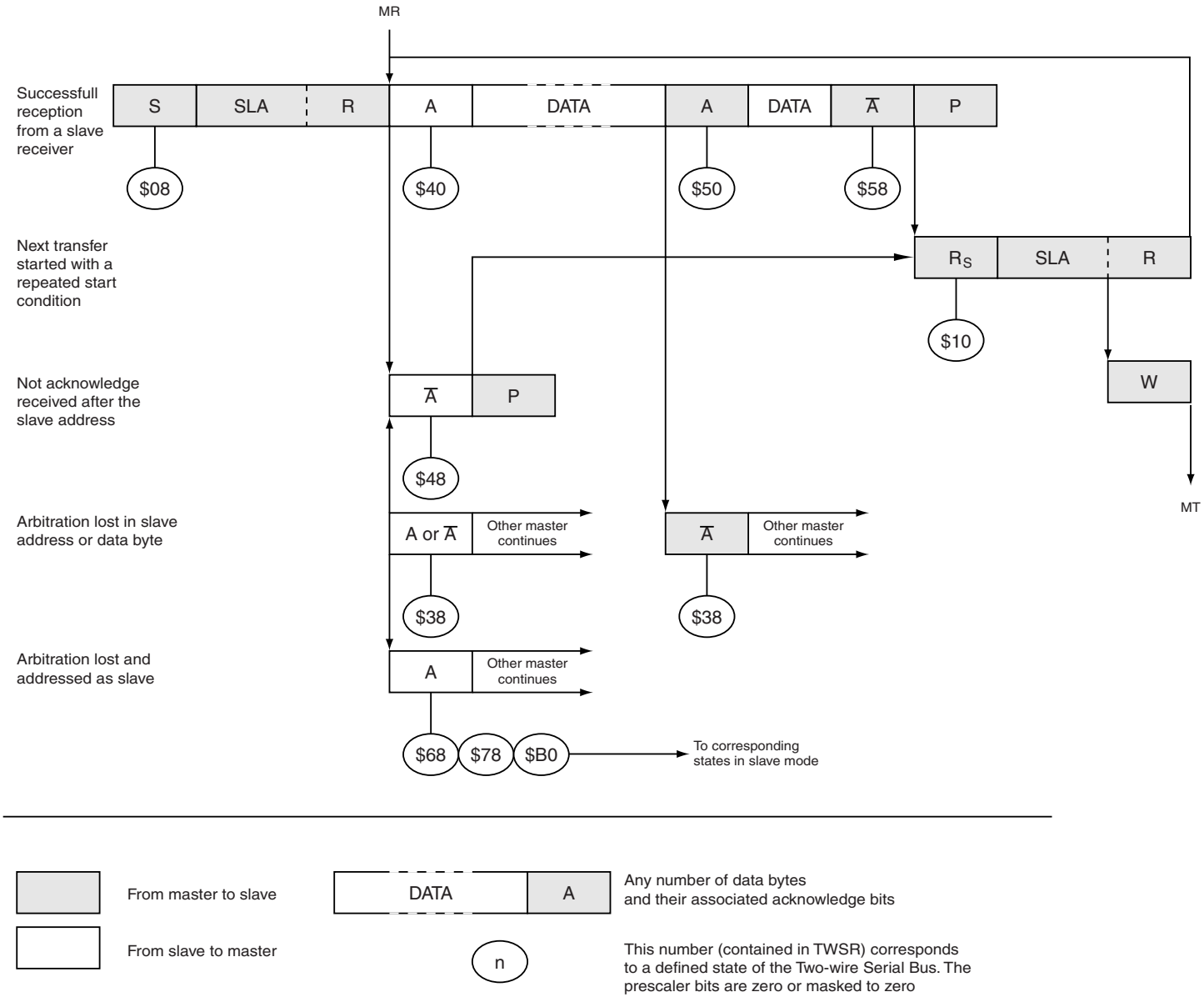


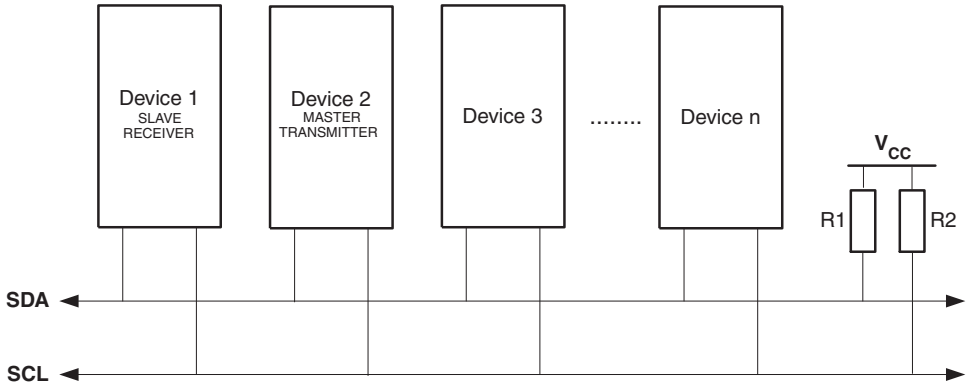
**Table 91.** Status Codes for Master Receiver Mode

| Status Code (TWSR) Prescaler Bits are 0 | Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware | Application Software Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWDR | To TWCR | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0x08 | A START condition has been transmitted | Load SLA+R | X | 0 | 1 | X | SLA+R will be transmitted<br>ACK or NOT ACK will be received |
| 0x10 | A repeated START condition has been transmitted | Load SLA+R or | X | 0 | 1 | X | SLA+R will be transmitted<br>ACK or NOT ACK will be received |
| | | Load SLA+W | X | 0 | 1 | X | SLA+W will be transmitted<br>Logic will switch to master transmitter mode |
| 0x38 | Arbitration lost in SLA+R or NOT ACK bit | No TWDR action or | 0 | 0 | 1 | X | Two-wire Serial Bus will be released and not addressed slave mode will be entered |
| | | No TWDR action | 1 | 0 | 1 | X | A START condition will be transmitted when the bus becomes free |
| 0x40 | SLA+R has been transmitted; ACK has been received | No TWDR action or | 0 | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | No TWDR action | 0 | 0 | 1 | 1 | Data byte will be received and ACK will be returned |
| 0x48 | SLA+R has been transmitted; NOT ACK has been received | No TWDR action or | 1 | 0 | 1 | X | Repeated START will be transmitted |
| | | No TWDR action or | 0 | 1 | 1 | X | STOP condition will be transmitted and TWSTO flag will be reset |
| | | No TWDR action | 1 | 1 | 1 | X | STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset |
| 0x50 | Data byte has been received; ACK has been returned | Read data byte or | 0 | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | Read data byte | 0 | 0 | 1 | 1 | Data byte will be received and ACK will be returned |
| 0x58 | Data byte has been received; NOT ACK has been returned | Read data byte or | 1 | 0 | 1 | X | Repeated START will be transmitted |
| | | Read data byte or | 0 | 1 | 1 | X | STOP condition will be transmitted and TWSTO flag will be reset |
| | | Read data byte | 1 | 1 | 1 | X | STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset |

**Slave Receiver Mode**

In the Slave Receiver mode, a number of data bytes are received from a master transmitter (see Figure 104). All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

**Figure 104.** Data Transfer in Slave Receiver Mode




To initiate the Slave Receiver mode, TWAR and TWCR must be initialized as follows:

| TWAR | TWA6 | TWA5 | TWA4 | TWA3 | TWA2 | TWA1 | TWA0 | TWGCE |
|---|---|---|---|---|---|---|---|---|
| value | Device's Own Slave Address | | | | | | | |

The upper seven bits are the address to which the Two-wire Serial Interface will respond when addressed by a master. If the LSB is set, the TWI will respond to the general call address (0x00), otherwise it will ignore the general call address.

| TWCR | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | – | TWIE |
|---|---|---|---|---|---|---|---|---|
| value | 0 | 1 | 0 | 0 | 0 | 1 | 0 | X |

TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgment of the device's own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is "0" (write), the TWI will operate in SR mode, otherwise ST mode is entered. After its own slave address and the write bit have been received, the TWINT flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in Table 92. The slave receiver mode may also be entered if arbitration is lost while the TWI is in the master mode (see states 0x68 and 0x78).

If the TWEA bit is reset during a transfer, the TWI will return a "Not Acknowledge" ("1") to SDA after the next received data byte. This can be used to indicate that the slave is not able to receive any more bytes. While TWEA is zero, the TWI does not acknowledge its own slave address. However, the Two-wire Serial Bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the Two-wire Serial Bus.

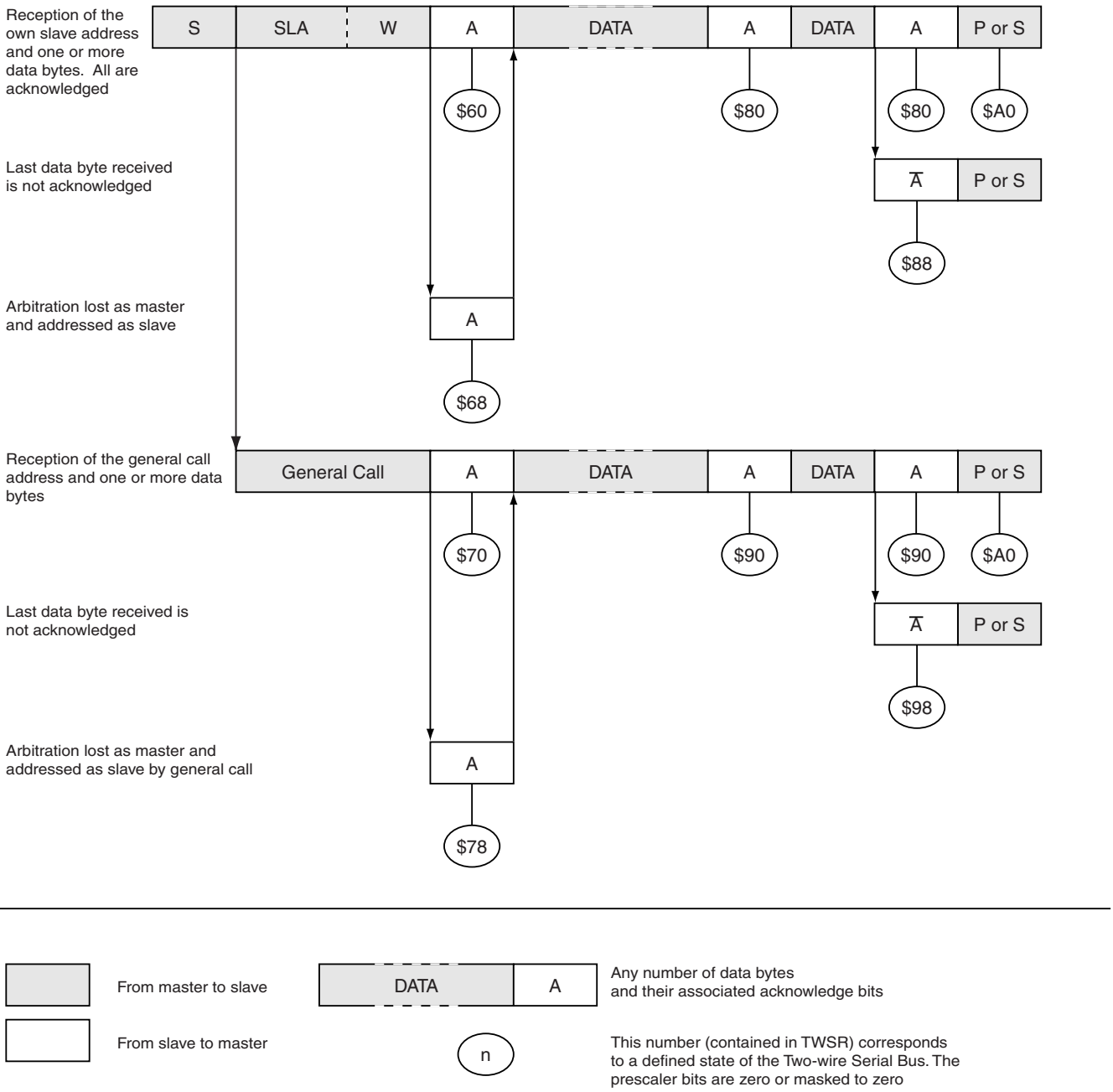
In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the Two-wire Serial Bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock low during the wake up and until the TWINT flag is cleared (by writing it to one). Further data reception will be carried out as normal, with the AVR clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

Note that the Two-wire Serial Interface Data Register – TWDR does not reflect the last byte present on the bus when waking up from these sleep modes.

**Table 92.** Status Codes for Slave Receiver Mode

| Status Code (TWSR) Prescaler Bits are 0 | Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware | Application Software Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWDR | To TWCR | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0x60 | Own SLA+W has been received; ACK has been returned | No TWDR action or | X | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | No TWDR action | X | 0 | 1 | 1 | Data byte will be received and ACK will be returned |
| 0x68 | Arbitration lost in SLA+R/W as master; own SLA+W has been received; ACK has been returned | No TWDR action or | X | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | No TWDR action | X | 0 | 1 | 1 | Data byte will be received and ACK will be returned |
| 0x70 | General call address has been received; ACK has been returned | No TWDR action or | X | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | No TWDR action | X | 0 | 1 | 1 | Data byte will be received and ACK will be returned |
| 0x78 | Arbitration lost in SLA+R/W as master; General call address has been received; ACK has been returned | No TWDR action or | X | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | No TWDR action | X | 0 | 1 | 1 | Data byte will be received and ACK will be returned |
| 0x80 | Previously addressed with own SLA+W; data has been received; ACK has been returned | Read data byte or | X | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | Read data byte | X | 0 | 1 | 1 | Data byte will be received and ACK will be returned |
| 0x88 | Previously addressed with own SLA+W; data has been received; NOT ACK has been returned | Read data byte or | 0 | 0 | 1 | 0 | Switched to the not addressed slave mode; no recognition of own SLA or GCA |
| | | Read data byte or | 0 | 0 | 1 | 1 | Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1” |
| | | Read data byte or | 1 | 0 | 1 | 0 | Switched to the not addressed slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free |
| | | Read data byte | 1 | 0 | 1 | 1 | Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free |
| 0x90 | Previously addressed with general call; data has been received; ACK has been returned | Read data byte or | X | 0 | 1 | 0 | Data byte will be received and NOT ACK will be returned |
| | | Read data byte | X | 0 | 1 | 1 | Data byte will be received and ACK will be returned |
| 0x98 | Previously addressed with general call; data has been received; NOT ACK has been returned | Read data byte or | 0 | 0 | 1 | 0 | Switched to the not addressed slave mode; no recognition of own SLA or GCA |
| | | Read data byte or | 0 | 0 | 1 | 1 | Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1” |
| | | Read data byte or | 1 | 0 | 1 | 0 | Switched to the not addressed slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free |
| | | Read data byte | 1 | 0 | 1 | 1 | Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free |
| 0xA0 | A STOP condition or repeated START condition has been received while still addressed as slave | Read data byte or | 0 | 0 | 1 | 0 | Switched to the not addressed slave mode; no recognition of own SLA or GCA |
| | | Read data byte or | 0 | 0 | 1 | 1 | Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1” |
| | | Read data byte or | 1 | 0 | 1 | 0 | Switched to the not addressed slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free |
| | | Read data byte | 1 | 0 | 1 | 1 | Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free |

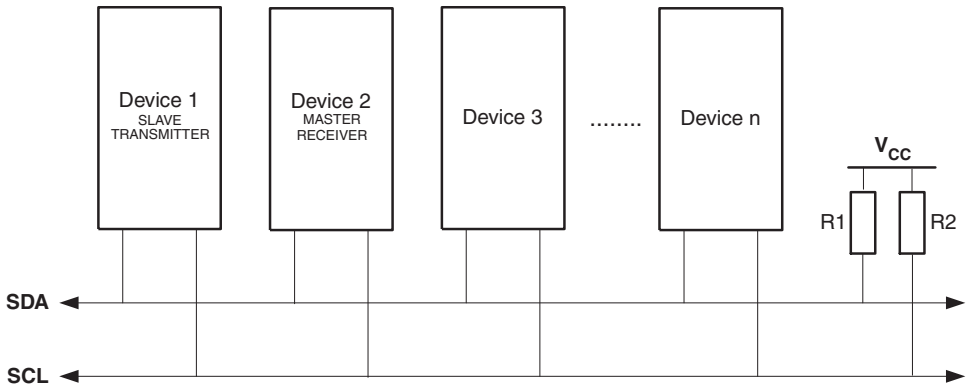
**Figure 105.** Formats and States in the Slave Receiver Mode




**Slave Transmitter Mode**

In the Slave Transmitter mode, a number of data bytes are transmitted to a master receiver (see Figure 106). All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

**Figure 106.** Data Transfer in Slave Transmitter Mode




To initiate the Slave Transmitter mode, TWAR and TWCR must be initialized as follows:

| TWAR | TWA6 | TWA5 | TWA4 | TWA3 | TWA2 | TWA1 | TWA0 | TWGCE |
|---|---|---|---|---|---|---|---|---|
| value | Device's Own Slave Address | | | | | | | |

The upper seven bits are the address to which the Two-wire Serial Interface will respond when addressed by a master. If the LSB is set, the TWI will respond to the general call address (0x00), otherwise it will ignore the general call address.

| TWCR | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | – | TWIE |
|---|---|---|---|---|---|---|---|---|
| value | 0 | 1 | 0 | 0 | 0 | 1 | 0 | X |

TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgment of the device's own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is "1" (read), the TWI will operate in ST mode, otherwise SR mode is entered. After its own slave address and the write bit have been received, the TWINT flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in Table 93. The Slave Transmitter mode may also be entered if arbitration is lost while the TWI is in the Master mode (see state 0xB0).

If the TWEA bit is written to zero during a transfer, the TWI will transmit the last byte of the transfer. State 0xC0 or state 0xC8 will be entered, depending on whether the master receiver transmits a NACK or ACK after the final byte. The TWI is switched to the not addressed slave mode, and will ignore the master if it continues the transfer. Thus the master receiver receives all "1" as serial data. State 0xC8 is entered if the master demands additional data bytes (by transmitting ACK), even though the slave has transmitted the last byte (TWEA zero and expecting NACK from the master).

While TWEA is zero, the TWI does not respond to its own slave address. However, the Two-wire Serial Bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the Two-wire Serial Bus.

In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the Two-wire Serial Bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock will low during the wake up and

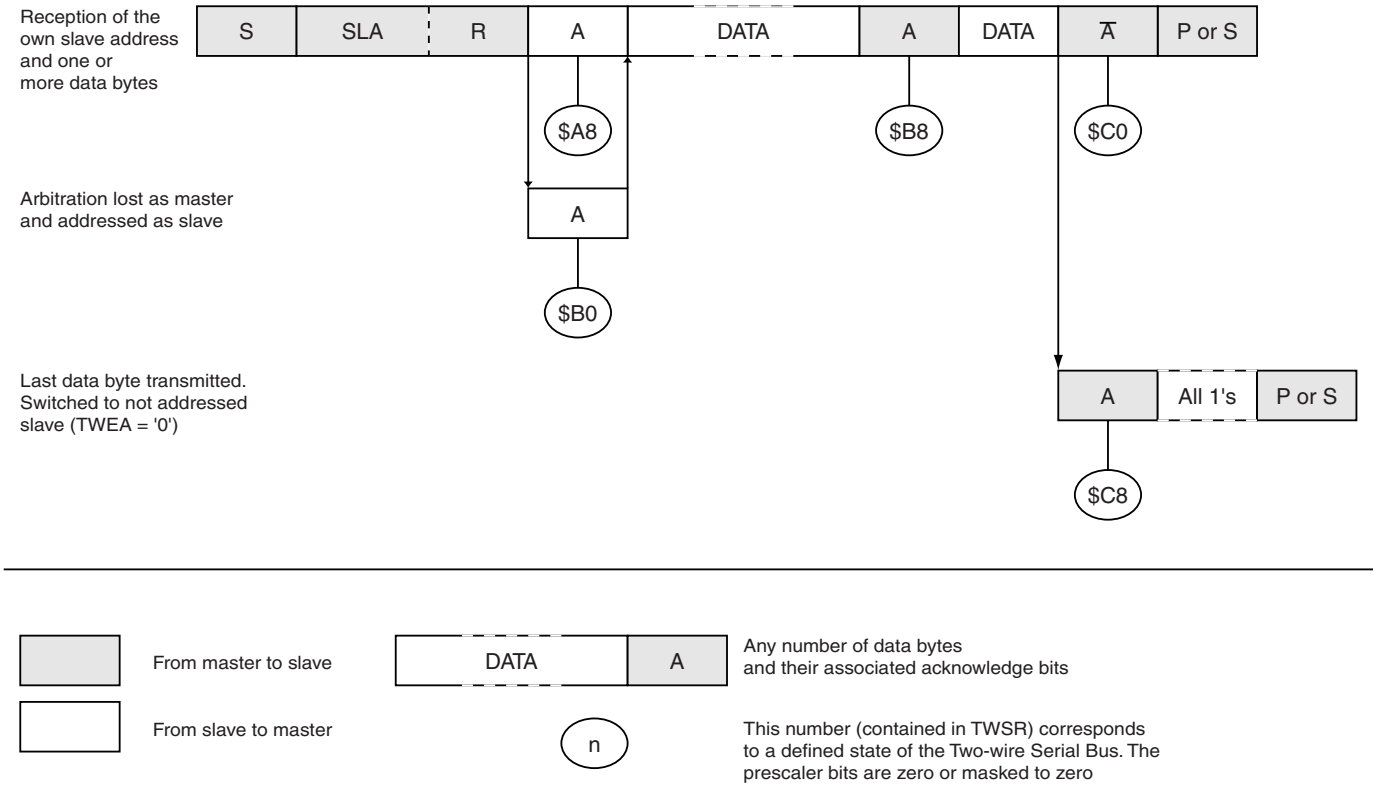
until the TWINT flag is cleared (by writing it to one). Further data transmission will be carried out as normal, with the AVR clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

Note that the Two-wire Serial Interface Data Register – TWDR does not reflect the last byte present on the bus when waking up from these sleep modes.

**Table 93.** Status Codes for Slave Transmitter Mode

| Status Code (TWSR) Prescaler Bits are 0 | Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware | Application Software Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWDR | To TWCR | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0xA8 | Own SLA+R has been received; ACK has been returned | Load data byte or | X | 0 | 1 | 0 | Last data byte will be transmitted and NOT ACK should be received |
| | | Load data byte | X | 0 | 1 | 1 | Data byte will be transmitted and ACK should be received |
| 0xB0 | Arbitration lost in SLA+R/W as master; own SLA+R has been received; ACK has been returned | Load data byte or | X | 0 | 1 | 0 | Last data byte will be transmitted and NOT ACK should be received |
| | | Load data byte | X | 0 | 1 | 1 | Data byte will be transmitted and ACK should be received |
| 0xB8 | Data byte in TWDR has been transmitted; ACK has been received | Load data byte or | X | 0 | 1 | 0 | Last data byte will be transmitted and NOT ACK should be received |
| | | Load data byte | X | 0 | 1 | 1 | Data byte will be transmitted and ACK should be received |
| 0xC0 | Data byte in TWDR has been transmitted; NOT ACK has been received | No TWDR action or | 0 | 0 | 1 | 0 | Switched to the not addressed slave mode; no recognition of own SLA or GCA |
| | | No TWDR action or | 0 | 0 | 1 | 1 | Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1” |
| | | No TWDR action or | 1 | 0 | 1 | 0 | Switched to the not addressed slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free |
| | | No TWDR action | 1 | 0 | 1 | 1 | Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free |
| 0xC8 | Last data byte in TWDR has been transmitted (TWEA = “0”); ACK has been received | No TWDR action or | 0 | 0 | 1 | 0 | Switched to the not addressed slave mode; no recognition of own SLA or GCA |
| | | No TWDR action or | 0 | 0 | 1 | 1 | Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1” |
| | | No TWDR action or | 1 | 0 | 1 | 0 | Switched to the not addressed slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free |
| | | No TWDR action | 1 | 0 | 1 | 1 | Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free |

**Figure 107.** Formats and States in the Slave Transmitter Mode




**Miscellaneous States**

There are two status codes that do not correspond to a defined TWI state, see Table 94.

Status 0xF8 indicates that no relevant information is available because the TWINT flag is not set. This occurs between other states, and when the TWI is not involved in a serial transfer.

Status 0x00 indicates that a bus error has occurred during a Two-wire Serial Bus transfer. A bus error occurs when a START or STOP condition occurs at an illegal position in the format frame. Examples of such illegal positions are during the serial transfer of an address byte, a data byte, or an acknowledge bit. When a bus error occurs, TWINT is set. To recover from a bus error, the TWSTO flag must set and TWINT must be cleared by writing a logic one to it. This causes the TWI to enter the not addressed slave mode and to clear the TWSTO flag (no other bits in TWCR are affected). The SDA and SCL lines are released, and no STOP condition is transmitted.

**Table 94.** Miscellaneous States

| Status Code (TWSR) Prescaler Bits are 0 | Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware | Application Software Response | | | | | Next Action Taken by TWI Hardware |
|---|---|---|---|---|---|---|---|
| | | To/from TWDR | To TWCR | | | | |
| | | | STA | STO | TWINT | TWEA | |
| 0xF8 | No relevant state information available; TWINT = “0” | No TWDR action | No TWCR action | | | | Wait or proceed current transfer |
| 0x00 | Bus error due to an illegal START or STOP condition | No TWDR action | 0 | 1 | 1 | X | Only the internal hardware is affected, no STOP condition is sent on the bus. In all cases, the bus is released and TWSTO is cleared. |

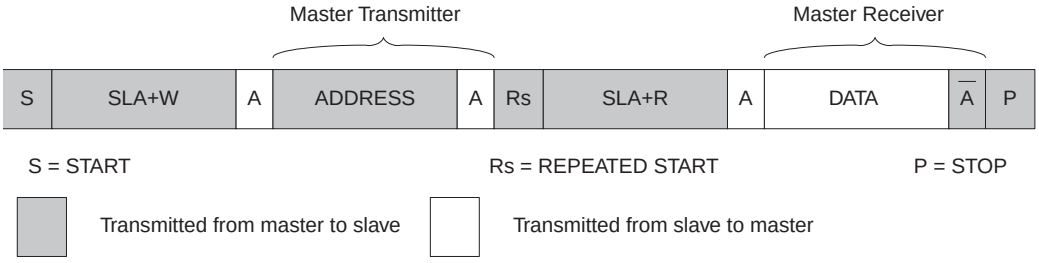
**Combining Several TWI Modes**

In some cases, several TWI modes must be combined in order to complete the desired action. Consider for example reading data from a serial EEPROM. Typically, such a transfer involves the following steps:

1. The transfer must be initiated
2. The EEPROM must be instructed what location should be read
3. The reading must be performed
4. The transfer must be finished

Note that data is transmitted both from master to slave and vice versa. The master must instruct the slave what location it wants to read, requiring the use of the MT mode. Subsequently, data must be read from the slave, implying the use of the MR mode. Thus, the transfer direction must be changed. The master must keep control of the bus during all these steps, and the steps should be carried out as an atomical operation. If this principle is violated in a multimaster system, another master can alter the data pointer in the EEPROM between steps 2 and 3, and the master will read the wrong data location. Such a change in transfer direction is accomplished by transmitting a REPEATED START between the transmission of the address byte and reception of the data. After a REPEATED START, the master keeps ownership of the bus. The following figure shows the flow in this transfer.

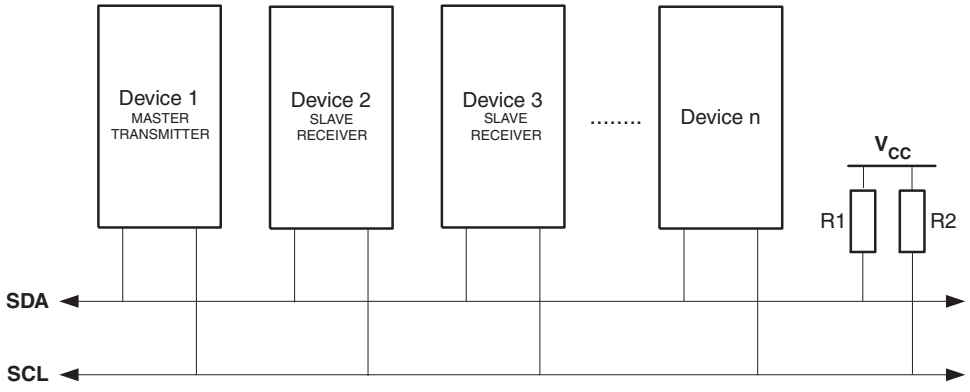
**Figure 108.** Combining Several TWI Modes to Access a Serial EEPROM




## Multi-master Systems and Arbitration

If multiple masters are connected to the same bus, transmissions may be initiated simultaneously by one or more of them. The TWI standard ensures that such situations are handled in such a way that one of the masters will be allowed to proceed with the transfer, and that no data will be lost in the process. An example of an arbitration situation is depicted below, where two masters are trying to transmit data to a slave receiver.

**Figure 109.** An Arbitration Example



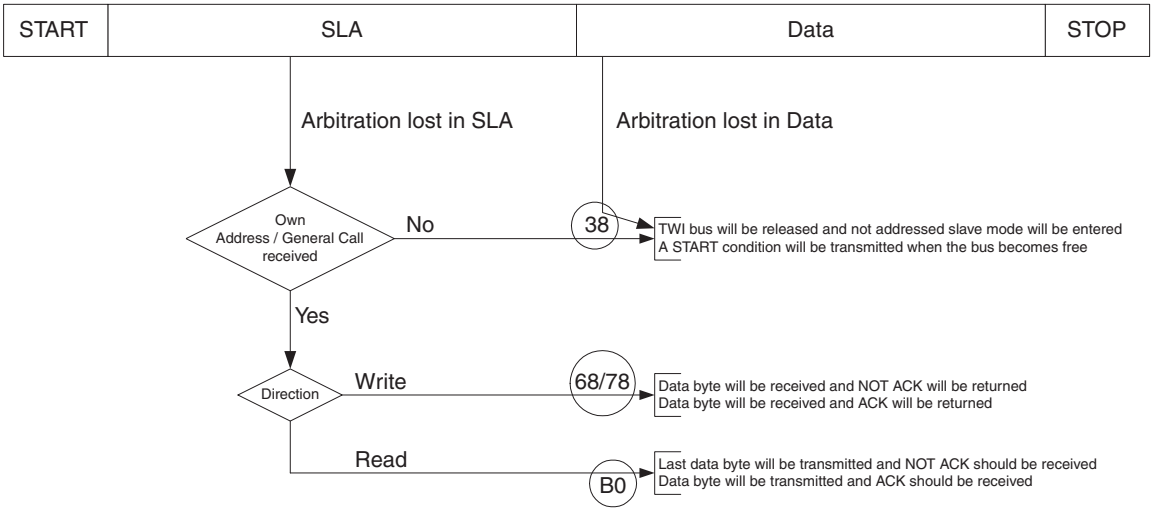

Several different scenarios may arise during arbitration, as described below:

- Two or more masters are performing identical communication with the same slave. In this case, neither the slave nor any of the masters will know about the bus contention.

- Two or more masters are accessing the same slave with different data or direction bit. In this case, arbitration will occur, either in the READ/WRITE bit or in the data bits. The masters trying to output a one on SDA while another master outputs a zero will lose the arbitration. Losing masters will switch to not addressed slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.
- Two or more masters are accessing different slaves. In this case, arbitration will occur in the SLA bits. Masters trying to output a one on SDA while another master outputs a zero will lose the arbitration. Masters losing arbitration in SLA will switch to slave mode to check if they are being addressed by the winning master. If addressed, they will switch to SR or ST mode, depending on the value of the READ/WRITE bit. If they are not being addressed, they will switch to not addressed slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.

This is summarized in Figure 110. Possible status values are given in circles.

**Figure 110.** Possible Status Codes Caused by Arbitration



## Controller Area Network - CAN

The Controller Area Network (CAN) protocol is a real-time, serial, broadcast protocol with a very high level of security. The AT90CAN128 CAN controller is fully compatible with the CAN Specification 2.0 Part A and Part B. It delivers the features required to implement the kernel of the CAN bus protocol according to the ISO/OSI Reference Model:

- The Data Link Layer
  - the Logical Link Control (LLC) sublayer
  - the Medium Access Control (MAC) sublayer
- The Physical Layer
  - the Physical Signalling (PLS) sublayer
  - not supported - the Physical Medium Attach (PMA)
  - not supported - the Medium Dependent Interface (MDI)

The CAN controller is able to handle all types of frames (Data, Remote, Error and Overload) and achieves a bitrate of 1 Mbit/s.

### Features

- Full Can Controller
- Fully Compliant with CAN Standard rev 2.0 A and rev 2.0 B
- 15 MOb (Message Object) with their own:
  - 11 bits of Identifier Tag (rev 2.0 A), 29 bits of Identifier Tag (rev 2.0 B)
  - 11 bits of Identifier Mask (rev 2.0 A), 29 bits of Identifier Mask (rev 2.0 B)
  - 8 Bytes Data Buffer (Static Allocation)
  - Tx, Rx, Frame Buffer or Automatic Reply Configuration
  - Time Stamping
- 1 Mbit/s Maximum Transfer Rate at 8 MHz
- TTC Timer
- Listening Mode (for Spying or Autobaud)

### CAN Protocol

The CAN protocol is an international standard defined in the ISO 11898 for high speed and ISO 11519-2 for low speed.

#### Principles

CAN is based on a broadcast communication mechanism. This broadcast communication is achieved by using a message oriented transmission protocol. These messages are identified by using a message identifier. Such a message identifier has to be unique within the whole network and it defines not only the content but also the priority of the message.

The priority at which a message is transmitted compared to another less urgent message is specified by the identifier of each message. The priorities are laid down during system design in the form of corresponding binary values and cannot be changed dynamically. The identifier with the lowest binary number has the highest priority.

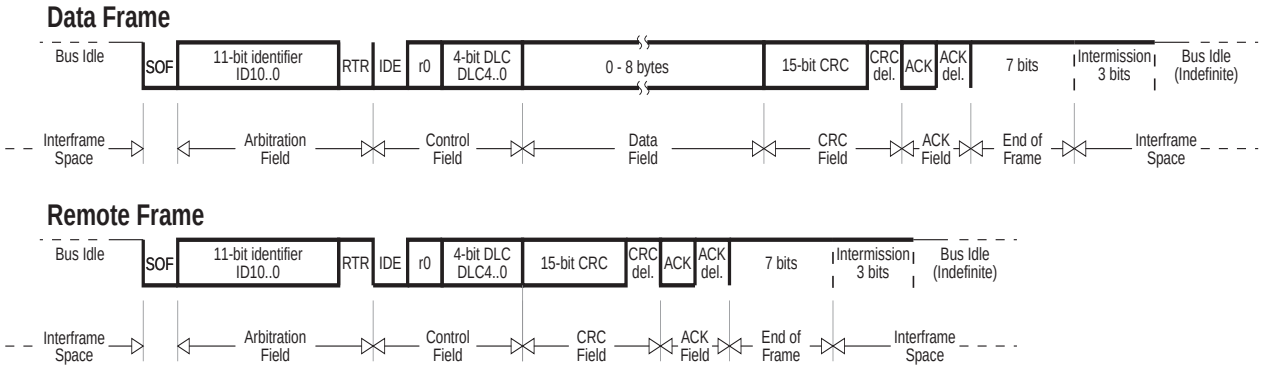
Bus access conflicts are resolved by bit-wise arbitration on the identifiers involved by each node observing the bus level bit for bit. This happens in accordance with the "wired and" mechanism, by which the dominant state overwrites the recessive state. The competition for bus allocation is lost by all nodes with recessive transmission and dominant observation. All the "losers" automatically become receivers of the message with the highest priority and do not re-attempt transmission until the bus is available again.

**Message Formats**

The CAN protocol supports two message frame formats, the only essential difference being in the length of the identifier. The CAN standard frame, also known as CAN 2.0 A, supports a length of 11 bits for the identifier, and the CAN extended frame, also known as CAN 2.0 B, supports a length of 29 bits for the identifier.

Can Standard Frame

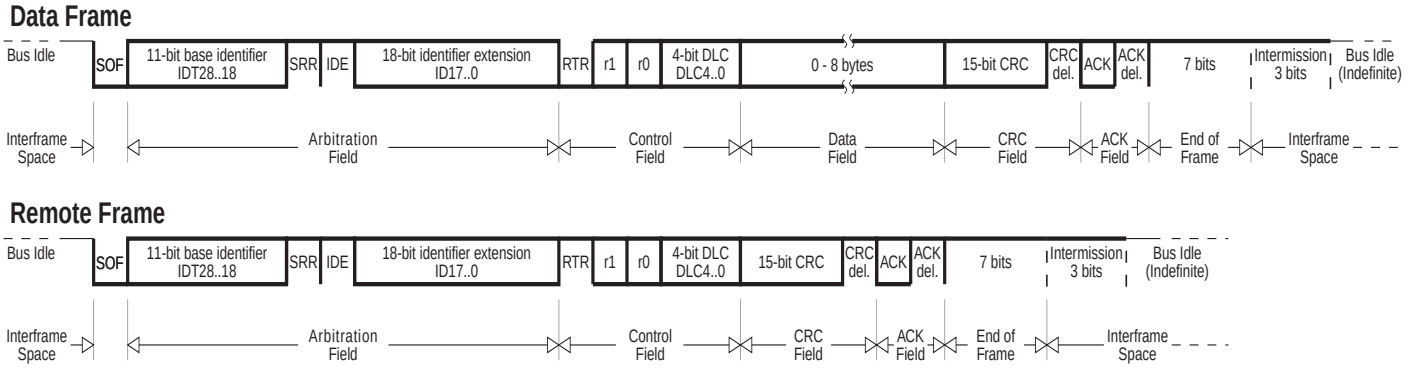
**Figure 111.** CAN Standard Frames




A message in the CAN standard frame format begins with the "Start Of Frame (SOF)", this is followed by the "Arbitration field" which consist of the identifier and the "Remote Transmission Request (RTR)" bit used to distinguish between the data frame and the data request frame called remote frame. The following "Control field" contains the "IDentifier Extension (IDE)" bit and the "Data Length Code (DLC)" used to indicate the number of following data bytes in the "Data field". In a remote frame, the DLC contains the number of requested data bytes. The "Data field" that follows can hold up to 8 data bytes. The frame integrity is guaranteed by the following "Cyclic Redundant Check (CRC)" sum. The "ACKnowledge (ACK) field" compromises the ACK slot and the ACK delimiter. The bit in the ACK slot is sent as a recessive bit and is overwritten as a dominant bit by the receivers which have at this time received the data correctly. Correct messages are acknowledged by the receivers regardless of the result of the acceptance test. The end of the message is indicated by "End Of Frame (EOF)". The "Intermission Frame Space (IFS)" is the minimum number of bits separating consecutive messages. If there is no following bus access by any node, the bus remains idle.

CAN Extended Frame

**Figure 112.** CAN Extended Frames



A message in the CAN extended frame format is likely the same as a message in CAN standard frame format. The difference is the length of the identifier used. The identifier is made up of the existing 11-bit identifier (base identifier) and an 18-bit extension (identifier extension). The distinction between CAN standard frame format and CAN extended frame format is made by using the IDE bit which is transmitted as dominant in case of a frame in CAN standard frame format, and transmitted as recessive in the other case.

Format Co-existence

As the two formats have to co-exist on one bus, it is laid down which message has higher priority on the bus in the case of bus access collision with different formats and the same identifier / base identifier: The message in CAN standard frame format always has priority over the message in extended format.

There are three different types of CAN modules available:

- 2.0A - Considers 29 bit ID as an error
- 2.0B Passive - Ignores 29 bit ID messages
- 2.0B Active - Handles both 11 and 29 bit ID Messages

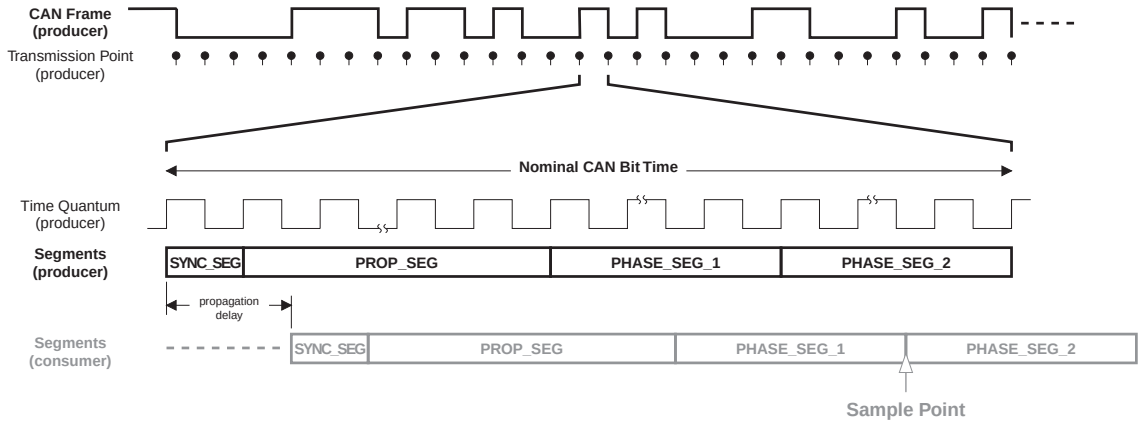
**CAN Bit Timing**

To ensure correct sampling up to the last bit, a CAN node needs to re-synchronize throughout the entire frame. This is done at the beginning of each message with the falling edge SOF and on each recessive to dominant edge.

Bit Construction

One CAN bit time is specified as four non-overlapping time segments. Each segment is constructed from an integer multiple of the Time Quantum. The Time Quantum or TQ is the smallest discrete timing resolution used by a CAN node.

**Figure 113.** CAN Bit Construction




Synchronization Segment

The first segment is used to synchronize the various bus nodes.

On transmission, at the start of this segment, the current bit level is output. If there is a bit state change between the previous bit and the current bit, then the bus state change is expected to occur within this segment by the receiving nodes.

Propagation Time Segment

This segment is used to compensate for signal delays across the network.

This is necessary to compensate for signal propagation delays on the bus line and through the transceivers of the bus nodes.

| | |
|---|---|
| Phase Segment 1 | Phase Segment 1 is used to compensate for edge phase errors.<br>This segment may be lengthened during re-synchronization. |
| Sample Point | The sample point is the point of time at which the bus level is read and interpreted as the value of the respective bit. Its location is at the end of Phase Segment 1 (between the two Phase Segments). |
| Phase Segment 2 | This segment is also used to compensate for edge phase errors.<br>This segment may be shortened during re-synchronization, but the length has to be at least as long as the Information Processing Time (IPT) and may not be more than the length of Phase Segment 1. |
| Information Processing Time | It is the time required for the logic to determine the bit level of a sampled bit.<br>The IPT begins at the sample point, is measured in TQ and is fixed at 2TQ for the Atmel CAN. Since Phase Segment 2 also begins at the sample point and is the last segment in the bit time, PS2 minimum shall not be less than the IPT. |
| Bit Lengthening | As a result of resynchronization, Phase Segment 1 may be lengthened or Phase Segment 2 may be shortened to compensate for oscillator tolerances. If, for example, the transmitter oscillator is slower than the receiver oscillator, the next falling edge used for resynchronization may be delayed. So Phase Segment 1 is lengthened in order to adjust the sample point and the end of the bit time. |
| Bit Shortening | If, on the other hand, the transmitter oscillator is faster than the receiver one, the next falling edge used for resynchronization may be too early. So Phase Segment 2 in bit N is shortened in order to adjust the sample point for bit N+1 and the end of the bit time |
| Synchronization Jump Width | The limit to the amount of lengthening or shortening of the Phase Segments is set by the Resynchronization Jump Width.<br>This segment may not be longer than Phase Segment 2. |
| Programming the Sample Point | Programming of the sample point allows "tuning" of the characteristics to suit the bus.<br>Early sampling allows more Time Quanta in the Phase Segment 2 so the Synchronization Jump Width can be programmed to its maximum. This maximum capacity to shorten or lengthen the bit time decreases the sensitivity to node oscillator tolerances, so that lower cost oscillators such as ceramic resonators may be used.<br>Late sampling allows more Time Quanta in the Propagation Time Segment which allows a poorer bus topology and maximum bus length. |
| Synchronization | Hard synchronization occurs on the recessive-to-dominant transition of the start bit. The bit time is restarted from that edge.<br>Re-synchronization occurs when a recessive-to-dominant edge doesn't occur within the Synchronization Segment in a message. |

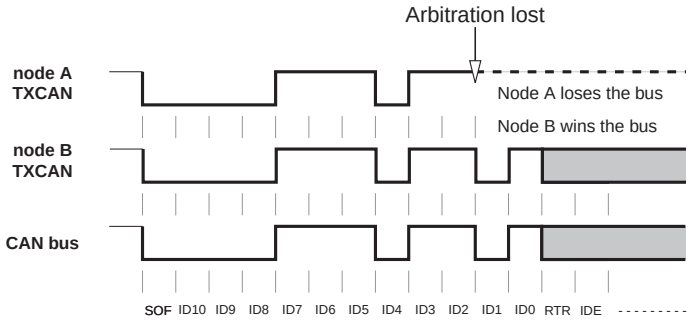
**Arbitration**

The CAN protocol handles bus accesses according to the concept called “Carrier Sense Multiple Access with Arbitration on Message Priority”.

During transmission, arbitration on the CAN bus can be lost to a competing device with a higher priority CAN Identifier. This arbitration concept avoids collisions of messages whose transmission was started by more than one node simultaneously and makes sure the most important message is sent first without time loss.

The bus access conflict is resolved during the arbitration field mostly over the identifier value. If a data frame and a remote frame with the same identifier are initiated at the same time, the data frame prevails over the remote frame (c.f. RTR bit).

**Figure 114.** Bus Arbitration




**Errors**

The CAN protocol signals any errors immediately as they occur. Three error detection mechanisms are implemented at the message level and two at the bit level:

Error at Message Level

- Cyclic Redundancy Check (CRC)
  The CRC safeguards the information in the frame by adding redundant check bits at the transmission end. At the receiver these bits are re-computed and tested against the received bits. If they do not agree there has been a CRC error.
- Frame Check
  This mechanism verifies the structure of the transmitted frame by checking the bit fields against the fixed format and the frame size. Errors detected by frame checks are designated "format errors".
- ACK Errors
  As already mentioned frames received are acknowledged by all receivers through positive acknowledgement. If no acknowledgement is received by the transmitter of the message an ACK error is indicated.

Error at Bit Level

- Monitoring
  The ability of the transmitter to detect errors is based on the monitoring of bus signals. Each node which transmits also observes the bus level and thus detects differences between the bit sent and the bit received. This permits reliable detection of global errors and errors local to the transmitter.
- Bit Stuffing
  The coding of the individual bits is tested at bit level. The bit representation used by CAN is "Non Return to Zero (NRZ)" coding, which guarantees maximum efficiency in bit coding. The synchronization edges are generated by means of bit stuffing.

Error Signalling

If one or more errors are discovered by at least one node using the above mechanisms, the current transmission is aborted by sending an "error flag". This prevents other nodes accepting the message and thus ensures the consistency of data throughout the network. After transmission of an erroneous message that has been aborted, the sender automatically re-attempts transmission.

## CAN Controller

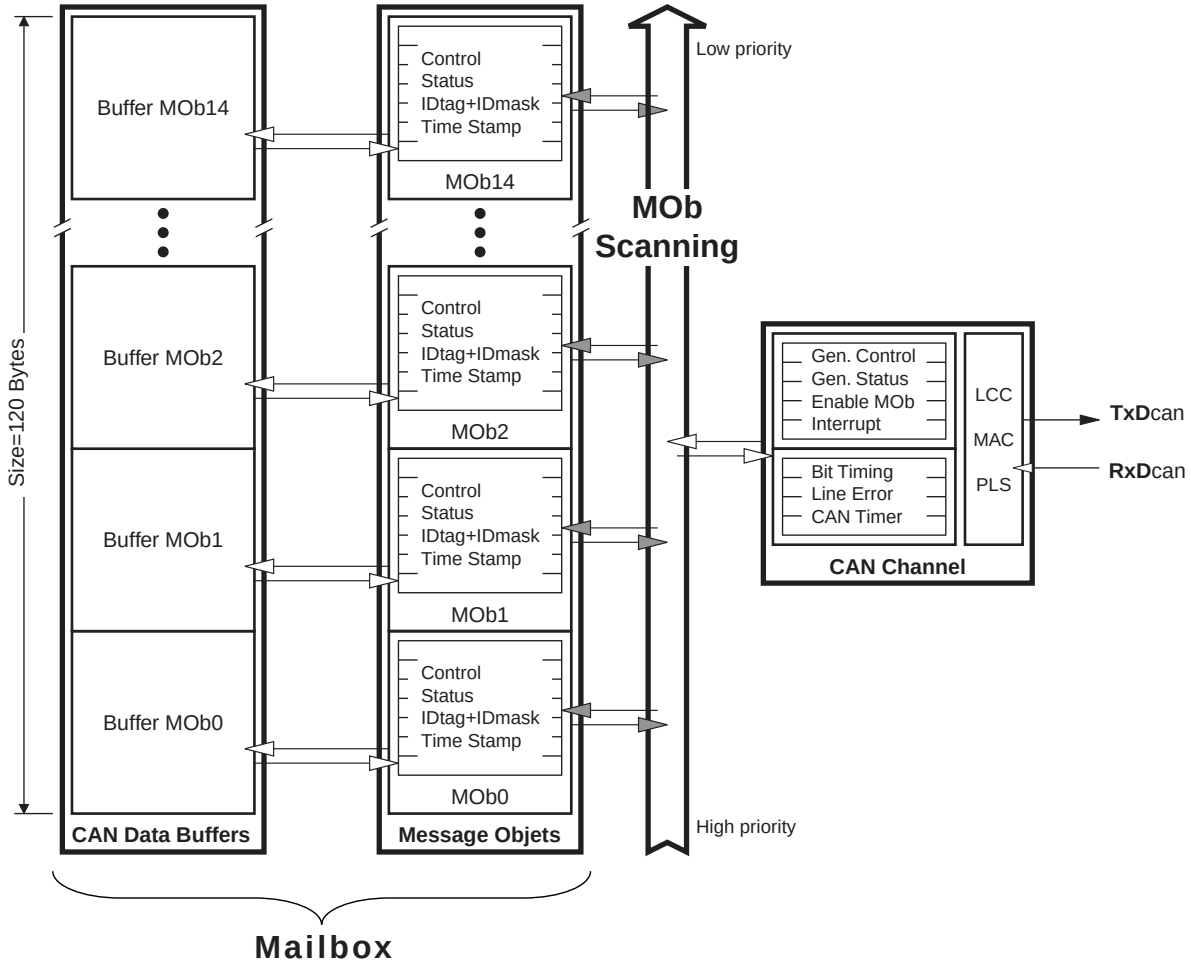
The CAN controller implemented into AT90CAN128 offers V2.0B Active.

This full-CAN controller provides the whole hardware for convenient acceptance filtering and message management. For each message to be transmitted or received this module contains one so called message object in which all information regarding the message (e.g. identifier, data bytes etc.) are stored.

During the initialization of the peripheral, the application defines which messages are to be sent and which are to be received. Only if the CAN controller receives a message whose identifier matches with one of the identifiers of the programmed (receive-) message objects the message is stored and the application is informed by interrupt. Another advantage is that incoming remote frames can be answered automatically by the full-CAN controller with the corresponding data frame. In this way, the CPU load is strongly reduced compared to a basic-CAN solution.

Using full-CAN controller, high baudrates and high bus loads with many messages can be handled.

**Figure 115.** CAN Controller Structure



## CAN Channel

### Configuration

The CAN channel can be in:

- **Enabled mode**

  In this mode:
  - the CAN channel (internal TXDCAN & RXDCAN) is enabled,
  - the input clock is enabled.

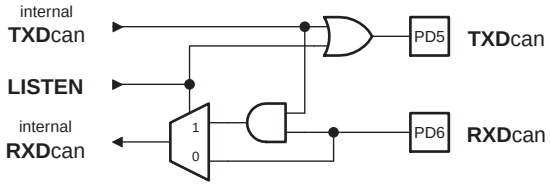
- **Standby mode**

  In standby mode:
  - the transmitter constantly provides a recessive level (on internal TXDCAN) and the receiver is disabled,
  - input clock is enabled,
  - the registers and pages remain accessible.

- **Listening mode**

  This mode is transparent for the CAN channel:
  - enables a hardware loop back, internal TXDCAN on internal RXDCAN
  - provides a recessive level on TXDCAN pin
  - does not disable RXDCAN
  - freezes TEC and REC error counters

**Figure 116.** Listening Mode




### Bit Timing

FSM's (Finite State Machine) of the CAN channel need to be synchronous to the time quantum. So, the input clock for bit timing is the clock used into CAN channel FSM's.

Field and segment abbreviations:

- BRP: Baud Rate Prescaler.
- TQ: Time Quantum (output of Baud Rate Prescaler).
- SYNS: SYNchronization Segment is 1 TQ long.
- PRS: PRopagation time Segment is programmable to be 1, 2, ..., 8 TQ long.
- PHS1: PHase Segment 1 is programmable to be 1, 2, ..., 8 TQ long.
- PHS2: PHase Segment 2 is programmable to be maximum of PHS1 and INFORMATION PROCESSING TIME.
- INFORMATION PROCESSING TIME is 2 TQ.
- SJW: (Re) Synchronization Jump Width is programmable to be minimum of PHS1 and 4.

The total number of TQ in a bit time has to be programmed at least from 8 to 25.

**Figure 117.** Sample and Transmission Point

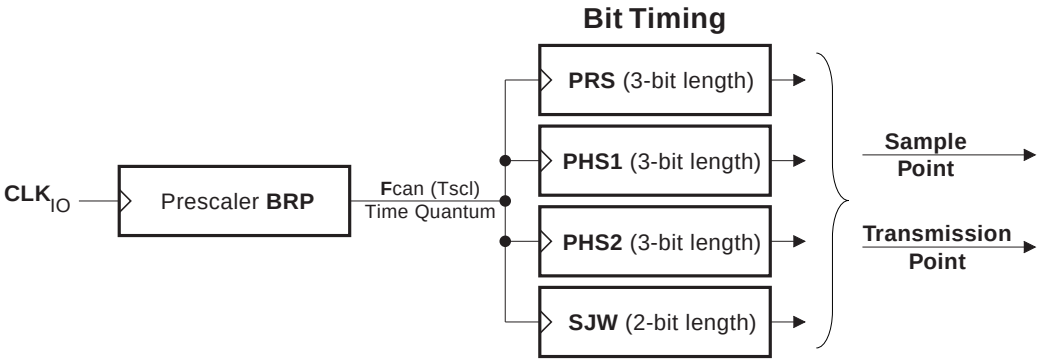
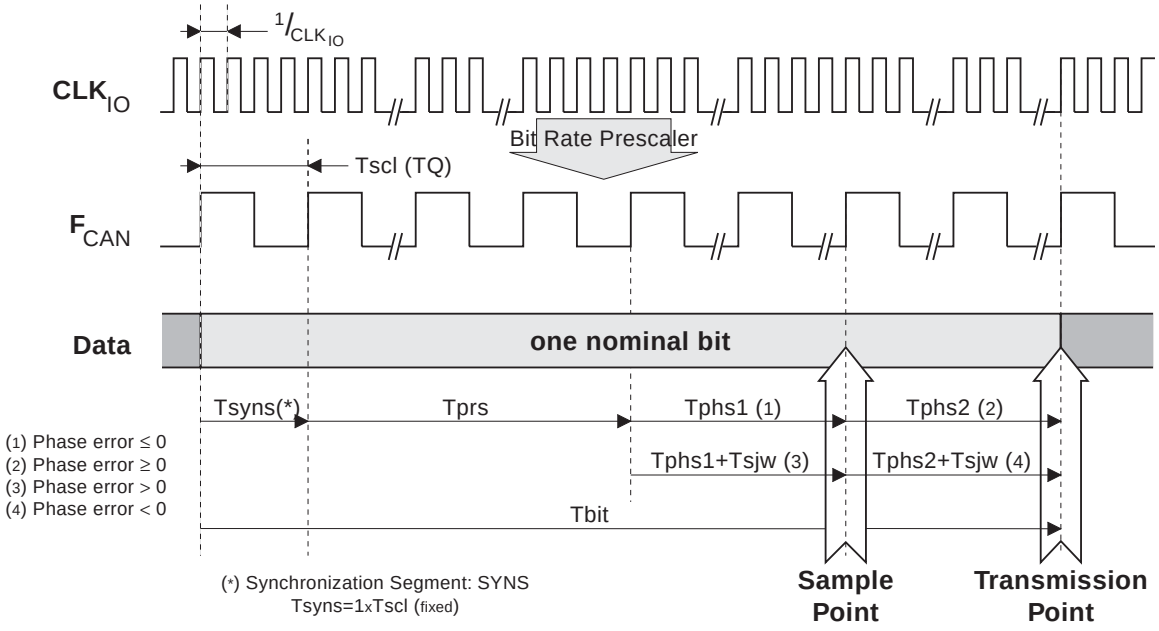



**Figure 118.** General Structure of a Bit Period




**Baud Rate**

The baud rate selection is made by $T_{bit}$ calculation:

$Tbit^{(1)} = Tsyns + Tprs + Tphs1 + Tphs2$

1. Tsyns = 1 x Tscl = (BRP[5..0]+ 1)/$clk_{IO}$ (= 1TQ)
2. Tprs = (1 to 8) x Tscl = (PRS[2..0]+ 1) x Tscl
3. Tphs1 = (1 to 8) x Tscl = (PHS1[2..0]+ 1) x Tscl
4. Tphs2 = (1 to 8) x Tscl = (PHS2[2..0]$^{(2)}$+ 1) x Tscl
5. Tsjw = (1 to 4) x Tscl = (SJW[1..0]+ 1) x Tscl

Notes: 1. The total number of Tscl (Time Quanta) in a bit time must be between 8 to 25.
2. PHS2[2..0] 2 is programmable to be maximum of PHS1[2..0] and 1.

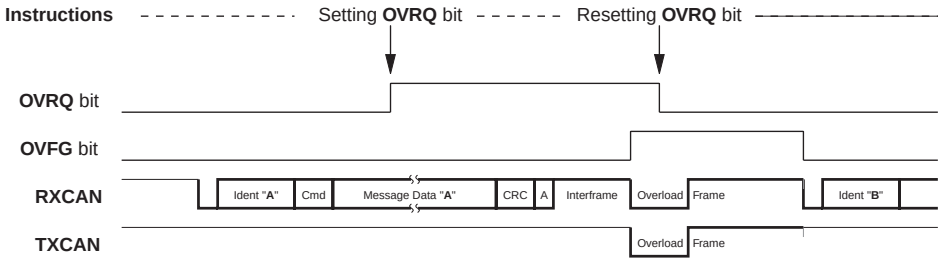
**Fault Confinement**

(c.f. Section “Error Management”).

### Overload Frame

An overload frame is sent by setting an overload request (OVRQ). After the next reception, the CAN channel sends an overload frame in accordance with the CAN specification. A status or flag is set (OVRF) as long as the overload frame is sent.

**Figure 119.** Overload Frame




## Message Objects

The MOb is a CAN frame descriptor. It contains all information to handle a CAN frame. This means that a MOb has been outlined to allow to describe a CAN message like an object. The set of MObs is the front end part of the “mailbox” where the messages to send and/or to receive are pre-defined as well as possible to decrease the work load of the software.

The MObs are numbered from 0 up to 14 (no MOb [15]). They are independent but priority is given to the lower one in case of multi matching. The operating modes are:

- Disabled mode
- Transmit mode
- Receive mode
- Automatic reply
- Frame buffer receive mode

### Operating Modes

Every MOb has its own fields to control the operating mode. There is **no** default mode after RESET. Before enabling the CAN peripheral, each MOb must be configured (ex: disabled mode - CONMOB=00).

**Table 95.** MOb Configuration

| MOb Configuration | | Reply Valid | RTR Tag | Operating Mode |
|---|---|---|---|---|
| 0 | 0 | x | x | Disabled |
| 0 | 1 | x | 0 | Tx Data Frame |
| | | x | 1 | Tx Remote Frame |
| 1 | 0 | x | 0 | Rx Data Frame |
| | | 0 | 1 | Rx Remote Frame |
| | | 1 | | Rx Remote Frame then, Tx Data Frame (reply) |
| 1 | 1 | x | x | Frame Buffer Receive Mode |

Disabled

In this mode, the MOb is “free”.

Tx Data & Remote Frame

1. Several fields must be initialized before sending:
   - Identifier tag (IDT)
   - Identifier extension (IDE)

- Remote transmission request (RTRTAG)
- Data length code (DLC)
- Reserved bit(s) tag (RBnTAG)
- Data bytes of message (MSG)

2. The MOb is ready to send a data or a remote frame when the MOb configuration is set (CONMOB).
3. Then, the CAN channel scans all the MObs in Tx configuration, finds the MOb having the highest priority and tries to send it.
4. When the transmission is completed the TXOK flag is set (interrupt).
5. All the parameters and data are available in the MOb until a new initialization.

Rx Data & Remote Frame

1. Several fields must be initialized before receiving:
   - Identifier tag (IDT)
   - Identifier mask (IDMSK)
   - Identifier extension (IDE)
   - Identifier extension mask (IDEMSK)
   - Remote transmission request (RTRTAG)
   - Remote transmission request mask (RTRMSK)
   - Data length code (DLC)
   - Reserved bit(s) tag (RBnTAG)
2. The MOb is ready to receive a data or a remote frame when the MOb configuration is set (CONMOB).
3. When a frame identifier is received on CAN network, the CAN channel scans all the MObs in receive mode, tries to find the MOb having the highest priority which is matching.
4. On a hit, the IDT, the IDE and the DLC of the matched MOb are updated from the incoming (frame) values.
5. Once the reception is completed, the data bytes of the received message are stored (not for remote frame) in the data buffer of the matched MOb and the RXOK flag is set (interrupt).
6. All the parameters and data are available in the MOb until a new initialization.

Automatic Reply

A reply (data frame) to a remote frame can be automatically sent after reception of the expected remote frame.

1. Several fields must be initialized before receiving the remote frame:
   - (c.f. Section “Rx Data & Remote Frame”)
2. When a remote frame matches, automatically the RTRTAG and the reply valid bit (RPLV) are reset. No flag (or interrupt) is set at this time. Since the CAN data buffer has not been used by the incoming remote frame, the MOb is then ready to be in transmit mode without any more setting. The IDT, the IDE, the other tags and the DLC of the received remote frame are used for the reply.
3. When the transmission of the reply is completed the TXOK flag is set (interrupt).
4. All the parameters and data are available in the MOb until a new initialization.

Frame Buffer Receive Mode

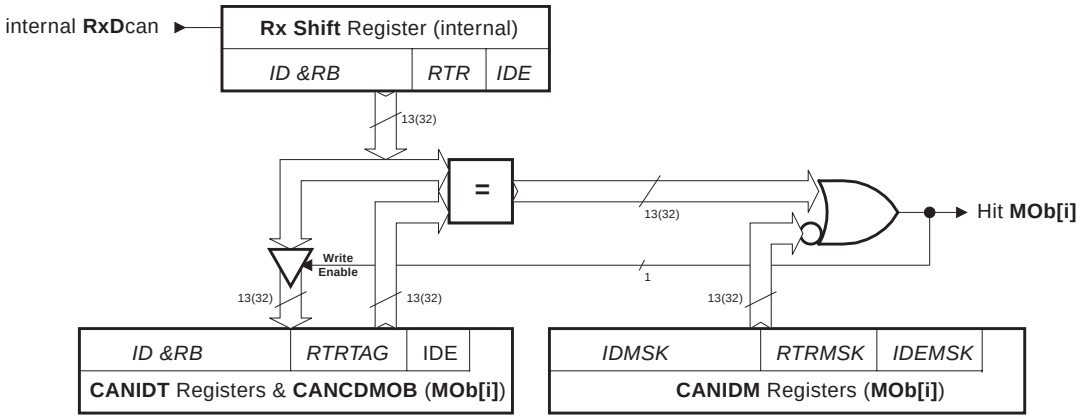
This mode is useful to receive multi frames. The priority between MObs offers a management for these incoming frames. One set MObs (including non-consecutive MObs) is created when the MObs are set in this mode. Due to the mode setting, only one set is possible. A frame buffer completed flag (or interrupt) - BXOK - will rise only when all the MObs of the set will have received their dedicated CAN frame.

1. MObs in frame buffer receive mode need to be initialized as MObs in standard receive mode.
2. The MObs are ready to receive data (or a remote) frames when their respective configurations are set (CONMOB).
3. When a frame identifier is received on CAN network, the CAN channel scans all the MObs in receive mode, tries to find the MOb having the highest priority which is matching.
4. On a hit, the IDT, the IDE and the DLC of the matched MOb are updated from the incoming (frame) values.
5. Once the reception is completed, the data bytes of the received message are stored (not for remote frame) in the data buffer of the matched MOb and the RXOK flag is set (interrupt).
6. When the reception in the last MOb of the set is completed, the frame buffer completed BXOK flag is set (interrupt). BXOK flag can be cleared only if all CONMOB fields of the set have been re-written before.
7. All the parameters and data are available in the MObs until a new initialization.

**Acceptance Filter**

Upon a reception hit (i.e., a good comparison between the ID + RTR + RBn + IDE received and an IDT+ RTRTAG + RBnTAG + IDE specified while taking the comparison mask into account) the IDT + RTRTAG + RBnTAG + IDE received are updated in the MOb (written over the registers).

**Figure 120.** Acceptance Filter Block Diagram




Note: Examples:

To accept only ID = 0x317 in part A.
- ID MSK = 111 1111 1111 b
- ID TAG = 011 0001 0111 b

To accept ID from 0x310 up to 0x317 in part A.
- ID MSK = 111 1111 1000 b
- ID TAG = 011 0001 0xxx b

**MOb Page**

Every MOb is mapped into a page to save place. The page number is the MOb number. This page number is set in CANPAGE register. The number 15 is reserved for factory tests.

CANHPMOB register gives the MOb having the highest priority in CANSIT registers. It is formatted to provide a direct entry for CANPAGE register. Because CANHPMOB codes CANSIT registers, it will be only updated if the corresponding enable bits (ENRX, ENTX, ENERR) are enabled (c.f. Figure 124).

### CAN Data Buffers

To preserve register allocation, the CAN data buffer is seen such as a FIFO (with address pointer accessible) into a MOb selection.This also allows to reduce the risks of un-controlled accesses.

There is one FIFO per MOb. This FIFO is accessed into a MOb page thanks to the CAN message register.

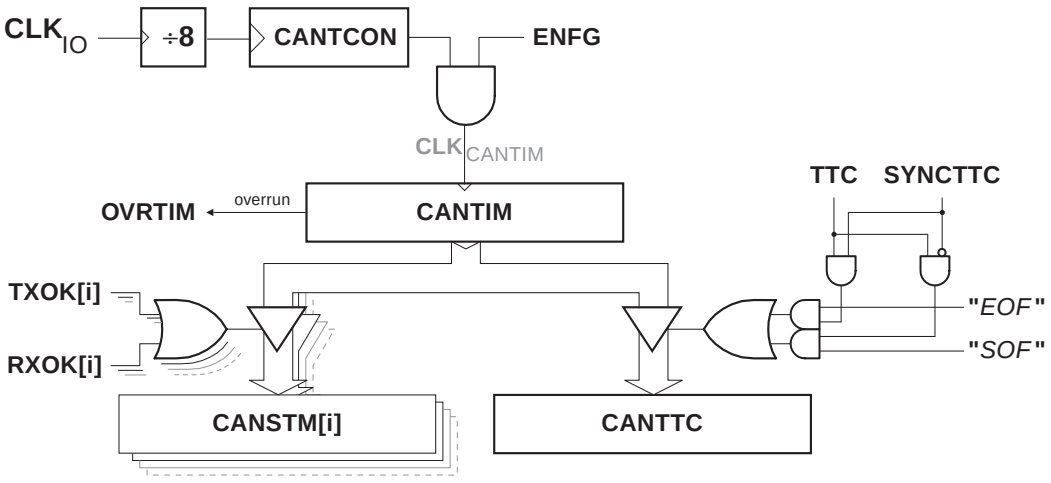
The data index (INDX) is the address pointer to the required data byte. The data byte can be read or write. The data index is automatically incremented after every access if the AINC* bit is reset. A roll-over is implemented, after data index=7 it is data index=0.

The first byte of a CAN frame is stored at the data index=0, the second one at the data index=1, ...

## CAN Timer

A programmable 16-bit timer is used for message stamping and time trigger communication (TTC).

**Figure 121.** CAN Timer Block Diagram




### Prescaler

An 8-bit prescaler is initialized by CANTCON register. It receives the $clk_{IO}$ divided by 8. It provides $CLK_{CANTIM}$ to the CAN Timer if the CAN controller is enabled.

$$CLK_{CANTIM} = CLK_{IO} \times 8 \times (CANTCON\ [7:0] + 1)$$

### 16-bit Timer

This timer starts counting from 0x0000 when the CAN controller is enabled (ENFG bit). When the timer rolls over from 0xFFFF to 0x0000, an interrupt is generated (OVRTIM).

### Time Triggering

Two synchronization modes are implemented for TTC (TTC bit):

- synchronization on Start of Frame (SYNCTTC=0),
- synchronization on End of Frame (SYNCTTC=1).

In TTC mode, a frame is sent once, even if an error occurs.

### Stamping Message

The capture of the timer value is done in the MOb which receives or sends the frame. All managed MOb are stamped, the stamping of a received (sent) frame occurs on RxOk (TXOK).

## Error Management

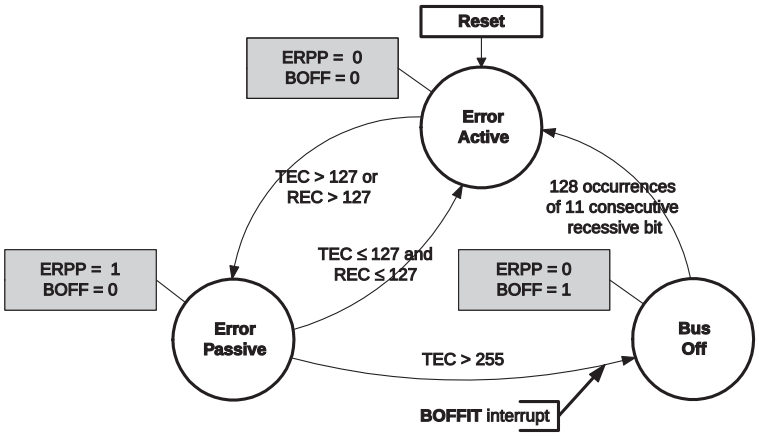
### Fault Confinement

The CAN channel may be in one of the three following states:

- Error active (default):
  The CAN channel takes part in bus communication and can send an active error frame when the CAN macro detects an error.
- Error passive:
  The CAN channel cannot send an active error frame. It takes part in bus communication, but when an error is detected, a passive error frame is sent. Also, after a transmission, an error passive unit will wait before initiating further transmission.
- Bus off:
  The CAN channel is not allowed to have any influence on the bus.

For fault confinement, a transmit error counter (TEC) and a receive error counter (REC) are implemented. BOFF and ERRP bits give the information of the state of the CAN channel. Setting BOFF to one may generate an interrupt.

**Figure 122.** Line Error Mode




Note: More than one REC/TEC change may apply during a given message transfer.

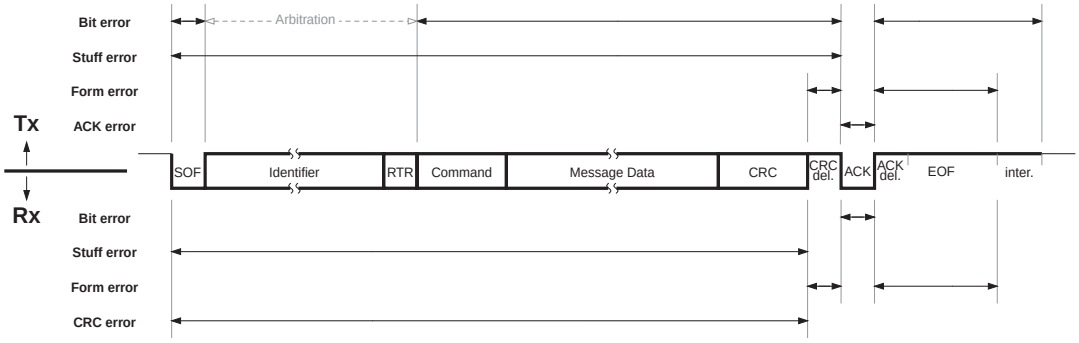
### Error Types

- **BERR**: Bit error. The bit value which is monitored is different from the bit value sent.
  Note:Exceptions:
  - Recessive bit sent monitored as dominant bit during the arbitration field and the acknowledge slot.
  - Detecting a dominant bit during the sending of an error frame.
- **SERR**: Stuff error. Detection of more than five consecutive bit with the same polarity.
- **CERR**: CRC error (Rx only). The receiver performs a CRC check on every destuffed received message from the start of frame up to the data field. If this checking does not match with the destuffed CRC field, an CRC error is set.
- **FERR**: Form error. The form error results from one (or more) violations of the fixed form of the following bit fields:
  – CRC delimiter
  – acknowledgement delimiter
  – end-of-frame
  – error delimiter

– overload delimiter

- **AERR**: Acknowledgment error (Tx only). No detection of the dominant bit in the acknowledge slot.

**Figure 123.** Error Detection Procedures in a Data Frame




**Error Setting**

The CAN channel can detect some errors on the CAN network.

- In transmission:
  The error is set at MOb level.
- In reception:
  - The identified has matched:
    The error is set at MOb level.
  - The identified has not or not yet matched:
    The error is set at general level.

After detecting an error, the CAN channel sends an error frame on network. If the CAN channel detects an error frame on network, it sends its own error frame.

## Interrupts

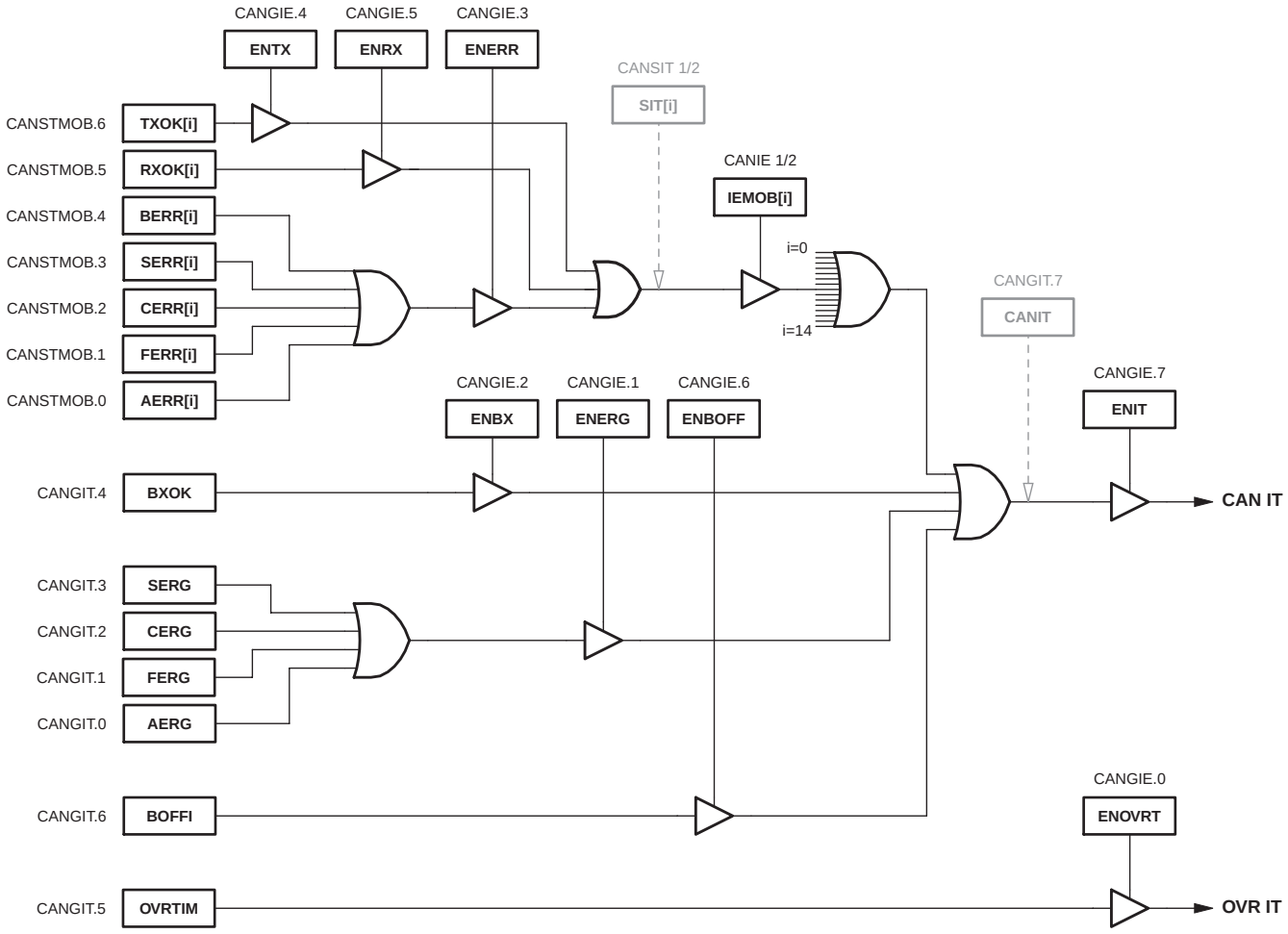
### Interrupt organization

The different interrupts are:

- Interrupt on receive completed OK,
- Interrupt on transmit completed OK,
- Interrupt on error (bit error, stuff error, crc error, form error, acknowledge error),
- Interrupt on frame buffer full,
- Interrupt on “Bus Off” setting,
- Interrupt on overrun of CAN timer.

The general interrupt enable is provided by ENIT bit and the specific interrupt enable for CAN timer overrun is provided by ENORVT bit.

**Figure 124.** CAN Controller Interrupt Structure



**Interrupt Behavior**

When an interrupt occurs, the corresponding bit is set in the CANSITn or CANGIT registers.

To acknowledge a MOb interrupt, the corresponding bits of CANSTMOB register (RXOK, TXOK,...) must be cleared by the software application. This operation needs a read-modify-write software routine.

To acknowledge a general interrupt, the corresponding bits of CANGIT register (BXOK, BOFFIT,...) must be cleared by the software application. This operation is made writing a logical one in these interrupt flags (writing a logical zero doesn't change the interrupt flag value).

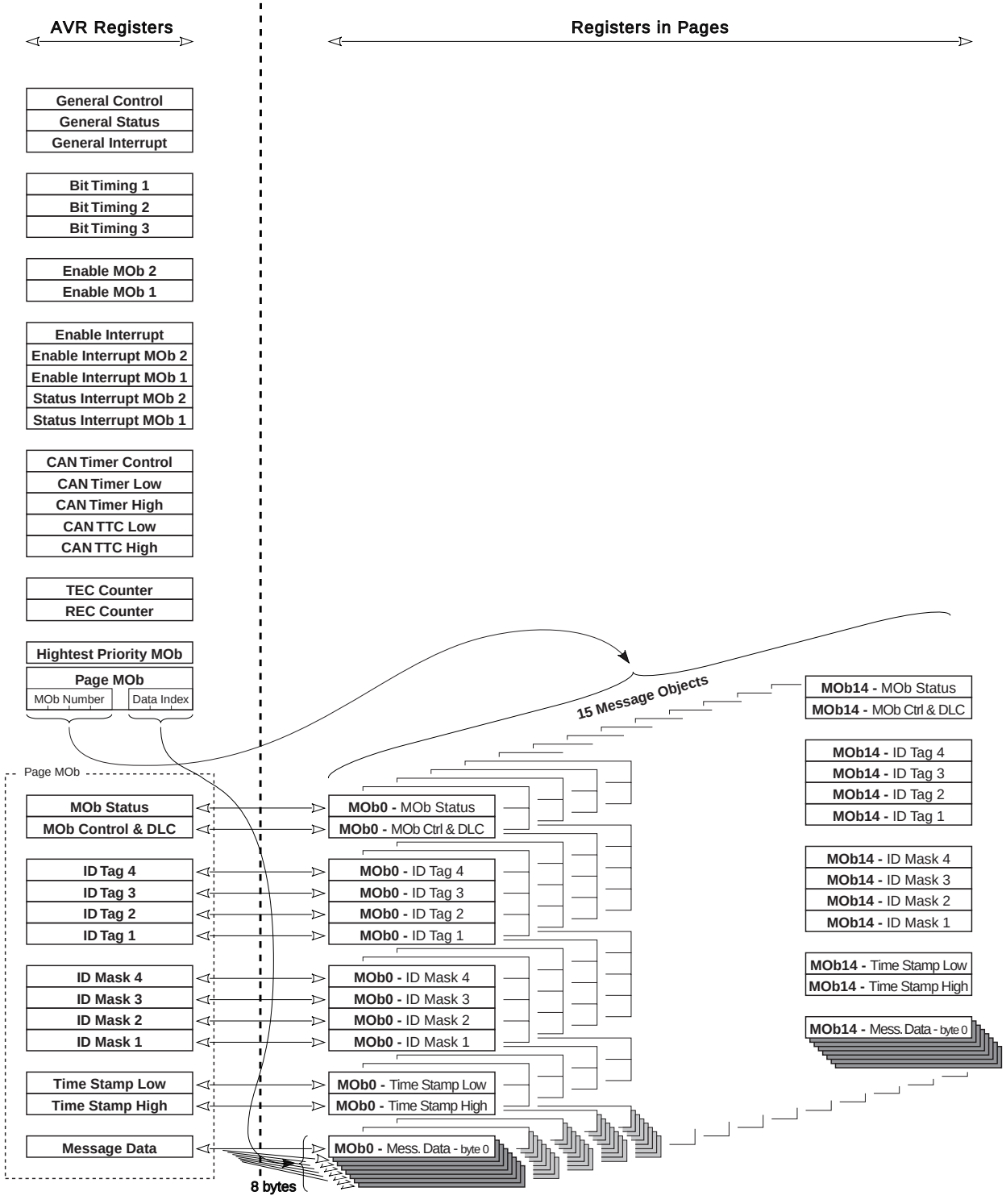
OVRTIM interrupt flag is reset as the other interrupt sources of CANGIT register and is also reset entering in its dedicated interrupt handler.

When the CAN node is in transmission and detects a Form Error in its frame, a bit Error will also be raised. Consequently, two consecutive interrupts can occur, both due to the same error.

When a MOb error occurs and is set in its own CANSTMOB register, no general error is set in CANGIT register.

## CAN Register Description

**Figure 125.** Registers Organization



## General CAN Registers

### CAN General Control Register - CANGCON

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | ABRQ | OVRQ | TTC | SYNTTC | LISTEN | TEST | ENA/STB | SWRES | CANGCON |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – ABRQ: Abort Request**

This is not an auto resettable bit.

  – 0 - no request.
  – 1 - abort request: a reset of CANEN1 and CANEN2 registers is done. The pending communications are immediately disabled and the on-going one will be normally terminated, setting the appropriate status flags.
  Note that CONCDMOB register remain unchanged.

- **Bit 6 – OVRQ: Overload Frame Request**

This is not an auto resettable bit.

  – 0 - no request.
  – 1 - overload frame request: send an overload frame after the next received frame.

The overload frame can be traced observing OVFG in CANGSTA register (c.f. Figure 119 on page 237).

- **Bit 5 – TTC: Time Trigger Communication**
  – 0 - no TTC.
  – 1- TTC mode.

- **Bit 4 – SYNTTC: Synchronization of TTC**

This bit is only used in TTC mode.

  – 0 - the TTC timer is caught on SOF.
  – 1 - the TTC timer is caught on the last bit of the EOF.

- **Bit 3 – LISTEN: Listening Mode**
  – 0 - no listening mode.
  – 1 - listening mode.

- **Bit 2 – TEST: Test Mode**
  – 0 - no test mode
  – 1 - test mode: intend for factory testing and **not** for customer use.

Note: CAN may malfunction if this bit is set.

- **Bit 1 – ENA/$\overline{\text{STB}}$: Enable / $\overline{\text{Standby}}$ Mode**

Because this bit is a command and is not immediately effective, the ENFG bit in CANGSTA register gives the true state of the chosen mode.

  – 0 - standby mode: the on-going communication is normally terminated and the CAN channel is frozen (the CONMOB bits of every MOb do not change). The transmitter constantly provides a recessive level. In this mode, the receiver is not enabled but all the registers and mailbox remain accessible from CPU.
  – 1 - enable mode: the CAN channel enters in enable mode once 11 recessive bits has been read.

- **Bit 0 – SWRES: Software Reset Request**

This auto resettable bit only resets the CAN controller.

– 0 - no reset
– 1 - reset: this reset is “ORed” with the hardware reset.

**CAN General Status Register - CANGSTA**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | - | OVFG | - | TXBSY | RXBSY | ENFG | BOFF | ERRP | CANGSTA |
| Read/Write | - | R | - | R | R | R | R | R | |
| Initial Value | - | 0 | - | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – Reserved Bit**

This bit is reserved for future use.

- **Bit 6 – OVFG: Overload Frame Flag**

This flag does not generate an interrupt.

– 0 - no overload frame.
– 1 - overload frame: set by hardware as long as the produced overload frame is sent.

- **Bit 5 – Reserved Bit**

This bit is reserved for future use.

- **Bit 4 – TXBSY: Transmitter Busy**

This flag does not generate an interrupt.

– 0 - transmitter not busy.
– 1 - transmitter busy: set by hardware as long as a frame (data, remote, overload or error frame) or an ACK field is sent. Also set when an inter frame space is sent.

- **Bit 3 – RXBSY: Receiver Busy**

This flag does not generate an interrupt.

– 0 - receiver not busy
– 1 - receiver busy: set by hardware as long as a frame is received or monitored.

- **Bit 2 – ENFG: Enable Flag**

This flag does not generate an interrupt.

– 0 - CAN controller disable: because an enable/disable command is not immediately effective, this status gives the true state of the chosen mode.
– 1 - CAN controller enable.

- **Bit 1 – BOFF: Bus Off Mode**

BOFF gives the information of the state of the CAN channel. Only entering in bus off mode generates the BOFFIT interrupt.

– 0 - no bus off mode.
– 1 - bus off mode.

- **Bit 0 – ERRP: Error Passive Mode**

ERRP gives the information of the state of the CAN channel. This flag does not generate an interrupt.

– 0 - no error passive mode.
– 1 - error passive mode.

**CAN General Interrupt Register - CANGIT**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | CANIT | BOFFIT | OVRTIM | BXOK | SERG | CERG | FERG | AERG | CANGIT |
| Read/Write | R | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – CANIT: General Interrupt Flag**

This is a read only bit.

– 0 - no interrupt.
– 1 - CAN interrupt: image of all the CAN controller interrupts except for OVRTIM interrupt. This bit can be used for polling method.

- **Bit 6 – BOFFIT: Bus Off Interrupt Flag**

Writing a logical one resets this interrupt flag. BOFFIT flag is only set when the CAN enters in bus off mode coming from error passive mode.

– 0 - no interrupt.
– 1 - bus off interrupt when the CAN enters in bus off mode.

- **Bit 5 – OVRTIM: Overrun CAN Timer**

Writing a logical one resets this interrupt flag. Entering in CAN timer overrun interrupt handler also reset this interrupt flag

– 0 - no interrupt.
– 1 - CAN timer overrun interrupt: set when the CAN timer switches from 0xFFFF to 0x0000.

- **Bit 4 – BXOK: Frame Buffer Receive Interrupt**

Writing a logical one resets this interrupt flag. BXOK flag can be cleared only if all CONMOB fields of the MOb's of the buffer have been re-written before.

– 0 - no interrupt.
– 1 - burst receive interrupt: set when the frame buffer receive is completed.

- **Bit 3 – SERG: Stuff Error General**

Writing a logical one resets this interrupt flag.

– 0 - no interrupt.
– 1 - stuff error interrupt: detection of more than five consecutive bits with the same polarity.

- **Bit 2 – CERG: CRC Error General**

Writing a logical one resets this interrupt flag.

– 0 - no interrupt.
– 1 - CRC error interrupt: the CRC check on destuffed message does not fit with the CRC field.

- **Bit 1 – FERG: Form Error General**

Writing a logical one resets this interrupt flag.

  – 0 - no interrupt.
  – 1 - form error interrupt: one or more violations of the fixed form in the CRC delimiter, acknowledgment delimiter or EOF.

- **Bit 0 – AERG: Acknowledgment Error General**

Writing a logical one resets this interrupt flag.

  – 0 - no interrupt.
  – 1 - acknowledgment error interrupt: no detection of the dominant bit in acknowledge slot.

**CAN General Interrupt Enable Register - CANGIE**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | ENIT | ENBOFF | ENRX | ENTX | ENERR | ENBX | ENERG | ENOVRT | CANGIE |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – ENIT: Enable all Interrupts (Except for CAN Timer Overrun Interrupt)**
  – 0 - interrupt disabled.
  – 1- CANIT interrupt enabled.

- **Bit 6 – ENBOFF: Enable Bus Off Interrupt**
  – 0 - interrupt disabled.
  – 1- bus off interrupt enabled.

- **Bit 5 – ENRX: Enable Receive Interrupt**
  – 0 - interrupt disabled.
  – 1- receive interrupt enabled.

- **Bit 4 – ENTX: Enable Transmit Interrupt**
  – 0 - interrupt disabled.
  – 1- transmit interrupt enabled.

- **Bit 3 – ENERR: Enable MOb Errors Interrupt**
  – 0 - interrupt disabled.
  – 1- MOb errors interrupt enabled.

- **Bit 2 – ENBX: Enable Frame Buffer Interrupt**
  – 0 - interrupt disabled.
  – 1- frame buffer interrupt enabled.

- **Bit 1 – ENERG: Enable General Errors Interrupt**
  – 0 - interrupt disabled.
  – 1- general errors interrupt enabled.

- **Bit 0 – ENOVRT: Enable CAN Timer Overrun Interrupt**
  – 0 - interrupt disabled.
  – 1- CAN timer interrupt overrun enabled.

### CAN Enable MOb Registers - CANEN2 and CANEN1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | ENMOB7 | ENMOB6 | ENMOB5 | ENMOB4 | ENMOB3 | ENMOB2 | ENMOB1 | ENMOB0 | CANEN2 |
| | - | ENMOB14 | ENMOB13 | ENMOB12 | ENMOB11 | ENMOB10 | ENMOB9 | ENMOB8 | CANEN1 |
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| Read/Write | R | R | R | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| Read/Write | - | R | R | R | R | R | R | R | |
| Initial Value | - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bits 14:0 - ENMOB14:0: Enable MOb**

This bit provides the availability of the MOb.
It is set to one when the MOb is enabled (i.e. CONMOB1:0 of CANCDMOB register). Once TXOK or RXOK is set to one (TXOK for automatic reply), the corresponding ENMOB is reset. ENMOB is also set to zero configuring the MOb in disabled mode, applying abortion or standby mode.

  – 0 - message object disabled: MOb available for a new transmission or reception.
  – 1 - message object enabled: MOb in use.

- **Bit 15 – Reserved Bit**

This bit is reserved for future use.

### CAN Enable Interrupt MOb Registers - CANIE2 and CANIE1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | IEMOB7 | IEMOB6 | IEMOB5 | IEMOB4 | IEMOB3 | IEMOB2 | IEMOB1 | IEMOB0 | CANIE2 |
| | - | IEMOB14 | IEMOB13 | IEMOB12 | IEMOB11 | IEMOB10 | IEMOB9 | IEMOB8 | CANIE1 |
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| Read/Write | - | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bits 14:0 - IEMOB14:0: Interrupt Enable by MOb**

  – 0 - interrupt disabled.
  – 1 - MOb interrupt enabled

Note: Example: CANIE2 = 0000 $1100_b$ : enable of interrupts on MOb 2 & 3.

- **Bit 15 – Reserved Bit**

This bit is reserved for future use. For compatibility with future devices, this must be written to zero when CANIE1 is written.

### CAN Status Interrupt MOb Registers - CANSIT2 and CANSIT1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | SIT7 | SIT6 | SIT5 | SIT4 | SIT3 | SIT2 | SIT1 | SIT0 | CANSIT2 |
| | - | SIT14 | SIT13 | SIT12 | SIT11 | SIT10 | SIT9 | SIT8 | CANSIT1 |
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| Read/Write | R | R | R | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| Read/Write | - | R | R | R | R | R | R | R | |
| Initial Value | - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bits 14:0 - SIT14:0: Status of Interrupt by MOb**
  - 0 - no interrupt.
  - 1- MOb interrupt.

Note: Example: CANSIT2 = 0010 $0001_b$ : MOb 0 & 5 interrupts.

- **Bit 15 – Reserved Bit**

This bit is reserved for future use.

**CAN Bit Timing Register 1 - CANBT1**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | - | BRP5 | BRP4 | BRP3 | BRP2 | BRP1 | BRP0 | - | CANBT1 |
| Read/Write | - | R/W | R/W | R/W | R/W | R/W | R/W | - | |
| Initial Value | - | 0 | 0 | 0 | 0 | 0 | 0 | - | |

- **Bit 7– Reserved Bit**

This bit is reserved for future use. For compatibility with future devices, this must be written to zero when CANBT1 is written.

- **Bit 6:1 – BRP5:0: Baud Rate Prescaler**

The period of the CAN controller system clock Tscl is programmable and determines the individual bit timing.

$$Tscl = \frac{BRP[5:0] + 1}{clk_{IO}\ frequency}$$

- **Bit 0 – Reserved Bit**

This bit is reserved for future use. For compatibility with future devices, this must be written to zero when CANBT1 is written.

**CAN Bit Timing Register 2 - CANBT2**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | - | SJW1 | SJW0 | - | PRS2 | PRS1 | PRS0 | - | CANBT2 |
| Read/Write | - | R/W | R/W | - | R/W | R/W | R/W | - | |
| Initial Value | - | 0 | 0 | - | 0 | 0 | 0 | - | |

- **Bit 7– Reserved Bit**

This bit is reserved for future use. For compatibility with future devices, this must be written to zero when CANBT2 is written.

- **Bit 6:5 – SJW1:0: Re-Synchronization Jump Width**

To compensate for phase shifts between clock oscillators of different bus controllers, the controller must re-synchronize on any relevant signal edge of the current transmission. The synchronization jump width defines the maximum number of clock cycles. A bit period may be shortened or lengthened by a re-synchronization.

$$Tsjw = Tscl \times (SJW\ [1:0] + 1)$$

- **Bit 4 – Reserved Bit**

This bit is reserved for future use. For compatibility with future devices, this must be written to zero when CANBT2 is written.

- **Bit 3:1 – PRS2:0: Propagation Time Segment**

This part of the bit time is used to compensate for the physical delay times within the network. It is twice the sum of the signal propagation time on the bus line, the input comparator delay and the output driver delay.

$$Tprs = Tscl \times (PRS[2:0] + 1)$$

- **Bit 0 – Reserved Bit**

This bit is reserved for future use. For compatibility with future devices, this must be written to zero when CANBT2 is written.

**CAN Bit Timing Register 3 - CANBT3**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | - | PHS22 | PHS21 | PHS20 | PHS12 | PHS11 | PHS10 | SMP | CANBT3 |
| Read/Write | - | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7– Reserved Bit**

This bit is reserved for future use. For compatibility with future devices, this must be written to zero when CANBT3 is written.

- **Bit 6:4 – PHS22:0: Phase Segment 2**

This phase is used to compensate for phase edge errors. This segment may be shortened by the re-synchronization jump width. PHS2[2..0] shall be ≥1 and ≤PHS1[2..0] (c.f. Section “CAN Bit Timing” and Section “Baud Rate”).

$$Tphs2 = Tscl \times (PHS2[2:0] + 1)$$

- **Bit 3:1 – PHS12:0: Phase Segment 1**

This phase is used to compensate for phase edge errors. This segment may be lengthened by the re-synchronization jump width.

$$Tphs1 = Tscl \times (PHS1[2:0] + 1)$$

- **Bit 0 – SMP: Sample Point(s)**
  - 0 - once, at the sample point.
  - 1 - three times, the threefold sampling of the bus is the sample point and twice over a distance of a 1/2 period of the Tscl. The result corresponds to the majority decision of the three values.

**CAN Timer Control Register - CANTCON**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TPRSC7 | TPRSC6 | TPRSC5 | TPRSC4 | TPRSC3 | TPRSC2 | TRPSC1 | TPRSC0 | CANTCON |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7:0 – TPRSC7:0: CAN Timer Prescaler**

Prescaler for the CAN timer upper counter range 0 to 255. It provides the clock to the CAN timer if the CAN controller is enabled.

$$CLK_{CANTIM} = CLK_{IO} \times 8 \times (CANTCON[7:0] + 1)$$

### CAN Timer Registers - CANTIML and CANTIMH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | CANTIM7 | CANTIM6 | CANTIM5 | CANTIM4 | CANTIM3 | CANTIM2 | CANTIM1 | CANTIM0 | CANTIML |
| | CANTIM15 | CANTIM14 | CANTIM13 | CANTIM12 | CANTIM11 | CANTIM10 | CANTIM9 | CANTIM8 | CANTIMH |
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| Read/Write | R | R | R | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bits 15:0 - CANTIM15:0: CAN Timer Count**

CAN timer counter range 0 to 65,535.

### CAN TTC Timer Registers - CANTTCL and CANTTCH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TIMTTC7 | TIMTTC6 | TIMTTC5 | TIMTTC4 | TIMTTC3 | TIMTTC2 | TIMTTC1 | TIMTTC0 | CANTTCL |
| | TIMTTC15 | TIMTTC14 | TIMTTC13 | TIMTTC12 | TIMTTC11 | TIMTTC10 | TIMTTC9 | TIMTTC8 | CANTTCH |
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| Read/Write | R | R | R | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bits 15:0 - TIMTTC15:0: TTC Timer Count**

CAN TTC timer counter range 0 to 65,535.

### CAN Transmit Error Counter Register - CANTEC

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TEC7 | TEC6 | TEC5 | TEC4 | TEC3 | TEC2 | TEC1 | TEC0 | CANTEC |
| Read/Write | R | R | R | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7:0 – TEC7:0: Transmit Error Count**

CAN transmit error counter range 0 to 255.

### CAN Receive Error Counter Register - CANREC

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | REC7 | REC6 | REC5 | REC4 | REC3 | REC2 | REC1 | REC0 | CANREC |
| Read/Write | R | R | R | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7:0 – REC7:0: Receive Error Count**

CAN receive error counter range 0 to 255.

### CAN Highest Priority MOb Register - CANHPMOB

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | HPMOB3 | HPMOB2 | HPMOB1 | HPMOB0 | CGP3 | CGP2 | CGP1 | CGP0 | CANHPMOB |
| Read/Write | R | R | R | R | R/W | R/W | R/W | R/W | |
| Initial Value | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | |

- **Bit 7:4 – HPMOB3:0: Highest Priority MOb Number**

MOb having the highest priority in CANSIT registers.
If CANSIT = 0 (no MOb), the return value is 0xF.

- **Bit 3:0 – CGP3:0: CAN General Purpose Bits**

These bits can be pre-programmed to match with the wanted configuration of the CANPAGE register (i.e., AINC and INDX2:0 setting).

### CAN Page MOb Register - CANPAGE

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | MOBNB3 | MOBNB2 | MOBNB1 | MOBNB0 | AINC | INDX2 | INDX1 | INDX0 | CANPAGE |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7:4 – MOBNB3:0: MOb Number**

Selection of the MOb number, the available numbers are from 0 to 14.

- **Bit 3 – $\overline{\text{AINC}}$: Auto Increment of the FIFO CAN Data Buffer Index (Active Low)**
  - 0 - auto increment of the index (default value).
  - 1- no auto increment of the index.

- **Bit 2:0 – INDX2:0: FIFO CAN Data Buffer Index**

Byte location of the CAN data byte into the FIFO for the defined MOb.

## MOb Registers

The MOb registers has **no** initial (default) value after RESET.

### CAN MOb Status Register - CANSTMOB

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | DLCW | TXOK | RXOK | BERR | SERR | CERR | FERR | AERR | CANSTMOB |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | - | - | - | - | - | - | - | - | |

- **Bit 7 – DLCW: Data Length Code Warning**

The incoming message does not have the DLC expected. Whatever the frame type, the DLC field of the CANCDMOB register is updated by the received DLC.

- **Bit 6 – TXOK: Transmit OK**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

The communication enabled by transmission is completed. When the controller is ready to send a frame, if two or more message objects are enabled as producers, the lower MOb index (0 to 14) is supplied first.

- **Bit 5 – RXOK: Receive OK**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

The communication enabled by reception is completed. In the case of two or more message object reception hits, the lower MOb index (0 to 14) is updated first.

- **Bit 4 – BERR: Bit Error (Only in Transmission)**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

The bit value monitored is different from the bit value sent.

Exceptions: the monitored recessive bit sent as a dominant bit during the arbitration field and the acknowledge slot detecting a dominant bit during the sending of an error frame.

- **Bit 3 – SERR: Stuff Error**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

Detection of more than five consecutive bits with the same polarity. This flag can generate an interrupt.

- **Bit 2 – CERR: CRC Error**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

The receiver performs a CRC check on every de-stuffed received message from the start of frame up to the data field. If this checking does not match with the de-stuffed CRC field, a CRC error is set.

- **Bit 1 – FERR: Form Error**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

The form error results from one or more violations of the fixed form in the following bit fields:

  - CRC delimiter.
  - Acknowledgment delimiter.
  - EOF

- **Bit 0 – AERR: Acknowledgment Error**

This flag can generate an interrupt. It must be cleared using a read-modify-write software routine on the whole CANSTMOB register.

No detection of the dominant bit in the acknowledge slot.

**CAN MOb Control and DLC Register - CANCDMOB**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | CONMOB1 | CONMOB0 | RPLV | IDE | DLC3 | DLC2 | DLC1 | DLC0 | CANCDMOB |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | - | - | - | - | - | - | - | - | |

- **Bit 7:6 – CONMOB1:0: Configuration of Message Object**

These bits set the communication to be performed (**no** initial value after RESET).

– 00 - disable.
– 01 - enable transmission.
– 10 - enable reception.
– 11 - enable frame buffer reception

These bits are **not** cleared once the communication is performed. The user must re-write the configuration to enable a new communication.

- This operation is necessary to be able to reset the BXOK flag.
- This operation also set the corresponding bit in the CANEN registers.

- **Bit 5 – RPLV: Reply Valid**

Used in the automatic reply mode after receiving a remote frame.

– 0 - reply not ready.
– 1 - reply ready and valid.

- **Bit 4 – IDE: Identifier Extension**

IDE bit of the remote or data frame to send.
This bit is updated with the corresponding value of the remote or data frame received.

– 0 - CAN standard rev 2.0 A (identifiers length = 11 bits).
– 1 - CAN standard rev 2.0 B (identifiers length = 29 bits).

- **Bit 3:0 – DLC3:0: Data Length Code**

Number of Bytes in the data field of the message.

DLC field of the remote or data frame to send. The range of DLC is from 0 up to 8. If DLC field >8 then effective DLC=8.

This field is updated with the corresponding value of the remote or data frame received. If the expected DLC differs from the incoming DLC, a DLC warning appears in the CANSTMOB register.

**CAN Identifier Tag Registers - CANIDT1, CANIDT2, CANIDT3, and CANIDT4**

**V2.0 part A**

| Bit | 15/7 | 14/6 | 13/5 | 12/4 | 11/3 | 10/2 | 9/1 | 8/0 | |
|---|---|---|---|---|---|---|---|---|---|
| | - | - | - | - | - | RTRTAG | - | RB0TAG | CANIDT4 |
| | - | - | - | - | - | - | - | - | CANIDT3 |
| | IDT2 | IDT1 | IDT0 | - | - | - | - | - | CANIDT2 |
| | IDT10 | IDT9 | IDT8 | IDT7 | IDT6 | IDT5 | IDT4 | IDT3 | CANIDT1 |
| Bit | 31/23 | 30/22 | 29/21 | 28/20 | 27/19 | 26/18 | 25/17 | 24/16 | |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | - | - | - | - | - | - | - | - | |

**V2.0 part B**

| Bit | 15/7 | 14/6 | 13/5 | 12/4 | 11/3 | 10/2 | 9/1 | 8/0 | |
|---|---|---|---|---|---|---|---|---|---|
| | IDT4 | IDT3 | IDT2 | IDT1 | IDT0 | RTRTAG | RB1TAG | RB0TAG | CANIDT4 |
| | IDT12 | IDT11 | IDT10 | IDT9 | IDT8 | IDT7 | IDT6 | IDT5 | CANIDT3 |
| | IDT20 | IDT19 | IDT18 | IDT17 | IDT16 | IDT15 | IDT14 | IDT13 | CANIDT2 |
| | IDT28 | IDT27 | IDT26 | IDT25 | IDT24 | IDT23 | IDT22 | IDT21 | CANIDT1 |
| Bit | 31/23 | 30/22 | 29/21 | 28/20 | 27/19 | 26/18 | 25/17 | 24/16 | |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | - | - | - | - | - | - | - | - | |

V2.0 part A

- **Bit 31:21 – IDT10:0: Identifier Tag**

Identifier field of the remote or data frame to send.

This field is updated with the corresponding value of the remote or data frame received.

- **Bit 20:3 – Reserved Bits**

These bits are reserved for future use. For compatibility with future devices, they must be written to zero when CANIDTn are written.

When a remote or data frame is received, these bits do not operate in the comparison but they are updated with un-predicted values.

- **Bit 2 – RTRTAG: Remote Transmission Request Tag**

RTR bit of the remote or data frame to send.

This tag is updated with the corresponding value of the remote or data frame received.

- **Bit 1 – Reserved Bit**

This bit is reserved for future use. For compatibility with future devices, it must be written to zero when CANIDTn are written.

When a remote or data frame is received, this bit does not operate in the comparison but it is updated with un-predicted values.

- **Bit 0 – RB0TAG: Reserved Bit 0 Tag**

RB0 bit of the remote or data frame to send.

This tag is updated with the corresponding value of the remote or data frame received.

V2.0 part B

- **Bit 31:3 – IDT28:0: Identifier Tag**

Identifier field of the remote or data frame to send.

This field is updated with the corresponding value of the remote or data frame received.

- **Bit 2 – RTRTAG: Remote Transmission Request Tag**

RTR bit of the remote or data frame to send.

This tag is updated with the corresponding value of the remote or data frame received.

- **Bit 1 – RB1TAG: Reserved Bit 1 Tag**

RB1 bit of the remote or data frame to send.

This tag is updated with the corresponding value of the remote or data frame received.

- **Bit 0 – RB0TAG: Reserved Bit 0 Tag**

RB0 bit of the remote or data frame to send.

This tag is updated with the corresponding value of the remote or data frame received.

**CAN Identifier Mask Registers - CANIDM1, CANIDM2, CANIDM3, and CANIDM4**

**V2.0 part A**

| Bit | 15/7 | 14/6 | 13/5 | 12/4 | 11/3 | 10/2 | 9/1 | 8/0 | |
|---|---|---|---|---|---|---|---|---|---|
| | - | - | - | - | - | RTRMSK | - | IDEMSK | CANIDM4 |
| | - | - | - | - | - | - | - | - | CANIDM3 |
| | IDMSK2 | IDMSK1 | IDMSK0 | - | - | - | - | - | CANIDM2 |
| | IDMSK10 | IDMSK9 | IDMSK8 | IDMSK7 | IDMSK6 | IDMSK5 | IDMSK4 | IDMSK3 | CANIDM1 |
| Bit | 31/23 | 30/22 | 29/21 | 28/20 | 27/19 | 26/18 | 25/17 | 24/16 | |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | - | - | - | - | - | - | - | - | |

**V2.0 part B**

| Bit | 15/7 | 14/6 | 13/5 | 12/4 | 11/3 | 10/2 | 9/1 | 8/0 | |
|---|---|---|---|---|---|---|---|---|---|
| | IDMSK4 | IDMSK3 | IDMSK2 | IDMSK1 | IDMSK0 | RTRMSK | - | IDEMSK | CANIDM4 |
| | IDMSK12 | IDMSK11 | IDMSK10 | IDMSK9 | IDMSK8 | IDMSK7 | IDMSK6 | IDMSK5 | CANIDM3 |
| | IDMSK20 | IDMSK19 | IDMSK18 | IDMSK17 | IDMSK16 | IDMSK15 | IDMSK14 | IDMSK13 | CANIDM2 |
| | IDMSK28 | IDMSK27 | IDMSK26 | IDMSK25 | IDMSK24 | IDMSK23 | IDMSK22 | IDMSK21 | CANIDM1 |
| Bit | 31/23 | 30/22 | 29/21 | 28/20 | 27/19 | 26/18 | 25/17 | 24/16 | |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | - | - | - | - | - | - | - | - | |

V2.0 part A

- **Bit 31:21 – IDMSK10:0: Identifier Mask**
  - 0 - comparison true forced
  - 1 - bit comparison enabled.

- **Bit 20:3 – Reserved Bits**

These bits are reserved for future use. For compatibility with future devices, they must be written to zero when CANIDMn are written.

- **Bit 2 – RTRMSK: Remote Transmission Request Mask**
  - 0 - comparison true forced
  - 1 - bit comparison enabled.

- **Bit 1 – Reserved Bit**

This bit is reserved for future use. For compatibility with future devices, it must be written to zero when CANIDTn are written.

- **Bit 0 – IDEMSK: Identifier Extension Mask**
  - 0 - comparison true forced
  - 1 - bit comparison enabled.

V2.0 part B

- **Bit 31:3 – IDMSK28:0: Identifier Mask**
  - 0 - comparison true forced
  - 1 - bit comparison enabled.

- **Bit 2 – RTRMSK: Remote Transmission Request Mask**
  - 0 - comparison true forced
  - 1 - bit comparison enabled.

- **Bit 1 – Reserved Bit**

Writing zero in this bit is recommended.

- **Bit 0 – IDEMSK: Identifier Extension Mask**
  - 0 - comparison true forced
  - 1 - bit comparison enabled.

### CAN Time Stamp Registers - CANSTML and CANSTMH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TIMSTM7 | TIMSTM6 | TIMSTM5 | TIMSTM4 | TIMSTM3 | TIMSTM2 | TIMSTM1 | TIMSTM0 | CANSTML |
| | TIMSTM15 | TIMSTM14 | TIMSTM13 | TIMSTM12 | TIMSTM11 | TIMSTM10 | TIMSTM9 | TIMSTM8 | CANSTMH |
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| Read/Write | R | R | R | R | R | R | R | R | |
| Initial Value | - | - | - | - | - | - | - | - | |

- **Bits 15:0 - TIMSTM15:0: Time Stamp Count**

CAN time stamp counter range 0 to 65,535.

### CAN Data Message Register - CANMSG

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | MSG 7 | MSG 6 | MSG 5 | MSG 4 | MSG 3 | MSG 2 | MSG 1 | MSG 0 | CANMSG |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | - | - | - | - | - | - | - | - | |

- **Bit 7:0 – MSG7:0: Message Data**

This register contains the CAN data byte pointed at the page MOb register.

After writing in the page MOb register, this byte is equal to the specified message location of the pre-defined identifier + index. If auto-incrementation is used, at the end of the data register writing or reading cycle, the index is auto-incremented.
The range of the counting is 8 with no end of loop (0, 1,..., 7, 0,...).

## Examples of CAN Baud Rate Setting

The CAN bus requires very accurate timing especially for high baud rates. It is recommended to use only an external crystal for CAN operations.

(Refer to “Bit Timing” on page 235 for timing description and page 251 to page 252 for “CAN Bit Timing Registers”).

**Table 96.** Examples of CAN Baud Rate Settings for Commonly Frequencies

| $f_{clk_{io}}$ (MHz) | CAN Baud Rate (Kbps) | Description | | | Segments | | | | Registers | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Sampling Point | TQ (µs) | Tbit (TQ) | Tprs (TQ) | Tph1 (TQ) | Tph2 (TQ) | Tsjw (TQ) | CANBT1 | CANBT2 | CANBT3 |
| 16.000 | 1000 | 75 % | 0.0625 | 16 | 7 | 4 | 4 | 1 | **0x00** | 0x0C | 0x37 |
| | | | 0.125 | 8 | 3 | 2 | 2 | 1 | **0x02** | 0x04 | 0x13 |
| | 500 | 75 % | 0.125 | 16 | 7 | 4 | 4 | 1 | **0x02** | 0x0C | 0x37 |
| | | | 0.250 | 8 | 3 | 2 | 2 | 1 | **0x06** | 0x04 | 0x13 |
| | 250 | 75 % | 0.250 | 16 | 7 | 4 | 4 | 1 | **0x06** | 0x0C | 0x37 |
| | | | 0.500 | 8 | 3 | 2 | 2 | 1 | **0x0E** | 0x04 | 0x13 |
| | 200 | 75 % | 0.3125 | 16 | 7 | 4 | 4 | 1 | **0x08** | 0x0C | 0x37 |
| | | | 0.625 | 8 | 3 | 2 | 2 | 1 | **0x12** | 0x04 | 0x13 |
| | 125 | 75 % | 0.500 | 16 | 7 | 4 | 4 | 1 | **0x0E** | 0x0C | 0x37 |
| | | | 1.000 | 8 | 3 | 2 | 2 | 1 | **0x1E** | 0x04 | 0x13 |
| | 100 | 75 % | 0.625 | 16 | 7 | 4 | 4 | 1 | **0x12** | 0x0C | 0x37 |
| | | | 1.350 | 8 | 3 | 2 | 2 | 1 | **0x26** | 0x04 | 0x13 |
| 12.000 | 1000 | 75 % | **0.083333** | **12** | **5** | **3** | **3** | 1 | **0x00** | **0x08** | **0x25** |
| | | | | x | *- - - no data - - -* | | | | | | |
| | 500 | 75 % | **0.166666** | **12** | **5** | **3** | **3** | 1 | **0x02** | **0x08** | **0x25** |
| | | | 0.250 | 8 | 3 | 2 | 2 | 1 | **0x04** | 0x04 | 0x13 |
| | 250 | 75 % | 0.250 | 16 | 7 | 4 | 4 | 1 | **0x04** | 0x0C | 0x37 |
| | | | 0.500 | 8 | 3 | 2 | 2 | 1 | **0x0A** | 0x04 | 0x13 |
| | 200 | 75 % | **0.250** | **20** | **8** | **6** | **5** | 1 | **0x04** | **0x0E** | **0x4B** |
| | | | **0.416666** | **12** | **5** | **3** | **3** | 1 | **0x08** | **0x08** | **0x25** |
| | 125 | 75 % | 0.500 | 16 | 7 | 4 | 4 | 1 | **0x0A** | 0x0C | 0x37 |
| | | | 1.000 | 8 | 3 | 2 | 2 | 1 | **0x16** | 0x04 | 0x13 |
| | 100 | 75 % | **0.500** | **20** | **8** | **6** | **5** | 1 | **0x0A** | **0x0E** | **0x4B** |
| | | | **0.833333** | **12** | **5** | **3** | **3** | 1 | **0x12** | **0x08** | **0x25** |

**Table 96.** Examples of CAN Baud Rate Settings for Commonly Frequencies (Continued)

| $f_{clk_{io}}$ (MHz) | CAN Baud Rate (Kbps) | Description | | | Segments | | | | Registers | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Sampling Point | TQ (µs) | Tbit (TQ) | Tprs (TQ) | Tph1 (TQ) | Tph2 (TQ) | Tsjw (TQ) | CANBT1 | CANBT2 | CANBT3 |
| 8.000 | 1000 | 75 % | | x | *- - - no data- - -* | | | | | | |
| | | | 0.125 | 8 | 3 | 2 | 2 | 1 | **0x00** | 0x04 | 0x13 |
| | 500 | 75 % | 0.125 | 16 | 7 | 4 | 4 | 1 | **0x00** | 0x0C | 0x37 |
| | | | 0.250 | 8 | 3 | 2 | 2 | 1 | **0x02** | 0x04 | 0x13 |
| | 250 | 75 % | 0.250 | 16 | 7 | 4 | 4 | 1 | **0x02** | 0x0C | 0x37 |
| | | | 0.500 | 8 | 3 | 2 | 2 | 1 | **0x06** | 0x04 | 0x13 |
| | 200 | 75 % | **0.250** | **20** | **8** | **6** | **5** | 1 | **0x02** | **0x0E** | **0x4B** |
| | | | 0.625 | 8 | 3 | 2 | 2 | 1 | **0x08** | 0x04 | 0x13 |
| | 125 | 75 % | 0.500 | 16 | 7 | 4 | 4 | 1 | **0x06** | 0x0C | 0x37 |
| | | | 1.000 | 8 | 3 | 2 | 2 | 1 | **0x0E** | 0x04 | 0x13 |
| | 100 | 75 % | 0.625 | 16 | 7 | 4 | 4 | 1 | **0x08** | 0x0C | 0x37 |
| | | | 1.350 | 8 | 3 | 2 | 2 | 1 | **0x12** | 0x04 | 0x13 |
| 6.000 | 1000 | *- - - not applicable- - -* | | | | | | | | | |
| | 500 | 75 % | **0.166666** | **12** | **5** | **3** | **3** | 1 | **0x00** | **0x08** | **0x25** |
| | | | | x | *- - - no data- - -* | | | | | | |
| | 250 | 75 % | **0.333333** | **12** | **5** | **3** | **3** | 1 | **0x02** | **0x08** | **0x25** |
| | | | 0.500 | 8 | 3 | 2 | 2 | 1 | **0x04** | 0x04 | 0x13 |
| | 200 | **80 %** | **0.333333** | **15** | **7** | **4** | **3** | 1 | **0x02** | **0x0C** | **0x35** |
| | | | **0.500** | **10** | **4** | **3** | **2** | 1 | **0x04** | **0x06** | **0x23** |
| | 125 | 75 % | 0.500 | 16 | 7 | 4 | 4 | 1 | **0x04** | 0x0C | 0x37 |
| | | | 1.000 | 8 | 3 | 2 | 2 | 1 | **0x0A** | 0x04 | 0x13 |
| | 100 | 75 % | **0.500** | **20** | **8** | **6** | **5** | 1 | **0x04** | **0x0E** | **0x4B** |
| | | | **0.833333** | **12** | **5** | **3** | **3** | 1 | **0x08** | **0x08** | **0x25** |
| 4.000 | 1000 | *- - - not applicable- - -* | | | | | | | | | |
| | 500 | 75 % | | x | *- - - no data- - -* | | | | | | |
| | | | 0.250 | 8 | 3 | 2 | 2 | 1 | **0x00** | 0x04 | 0x13 |
| | 250 | 75 % | 0.250 | 16 | 7 | 4 | 4 | 1 | **0x00** | 0x0C | 0x37 |
| | | | 0.500 | 8 | 3 | 2 | 2 | 1 | **0x02** | 0x04 | 0x13 |
| | 200 | 75 % | **0.250** | **20** | **8** | **6** | **5** | 1 | **0x00** | **0x0E** | **0x4B** |
| | | | | x | *- - - no data- - -* | | | | | | |
| | 125 | 75 % | 0.500 | 16 | 7 | 4 | 4 | 1 | **0x02** | 0x0C | 0x37 |
| | | | 1.000 | 8 | 3 | 2 | 2 | 1 | **0x06** | 0x04 | 0x13 |
| | 100 | 75 % | **0.500** | **20** | **8** | **6** | **5** | 1 | **0x02** | **0x0E** | **0x4B** |
| | | | 1.350 | 8 | 3 | 2 | 2 | 1 | 0x08 | 0x04 | 0x13 |

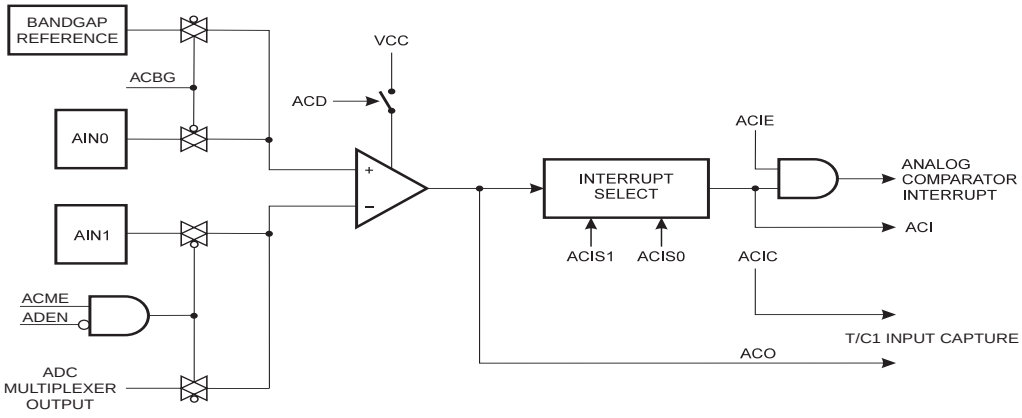
## Analog Comparator

The Analog Comparator compares the input values on the positive pin AIN0 and negative pin AIN1.

### Overview

When the voltage on the positive pin AIN0 is higher than the voltage on the negative pin AIN1, the Analog Comparator output, ACO, is set. The comparator's output can be set to trigger the Timer/Counter1 Input Capture function. In addition, the comparator can trigger a separate interrupt, exclusive to the Analog Comparator. The user can select Interrupt triggering on comparator output rise, fall or toggle. A block diagram of the comparator and its surrounding logic is shown in Figure 126.

**Figure 126.** Analog Comparator Block Diagram[1][2]




Notes: 1. ADC multiplexer output: see Table 98 on page 264.
2. Refer to Figure 2 on page 4 and Table 41 on page 78 for Analog Comparator pin placement.

## Analog Comparator Register Description

### ADC Control and Status Register B – ADCSRB

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **ADHSM** | **ACME** | **–** | **–** | **–** | **ADTS2** | **ADTS1** | **ADTS0** | **ADCSRB** |
| Read/Write | R/W | R/W | R | R | R | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 6 – ACME: Analog Comparator Multiplexer Enable**

When this bit is written logic one and the ADC is switched off (ADEN in ADCSRA is zero), the ADC multiplexer selects the negative input to the Analog Comparator. When this bit is written logic zero, AIN1 is applied to the negative input of the Analog Comparator. For a detailed description of this bit, see "Analog Comparator Multiplexed Input" on page 264.

### Analog Comparator Control and Status Register – ACSR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **ACD** | **ACBG** | **ACO** | **ACI** | **ACIE** | **ACIC** | **ACIS1** | **ACIS0** | **ACSR** |
| Read/Write | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | N/A | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – ACD: Analog Comparator Disable**

When this bit is written logic one, the power to the Analog Comparator is switched off. This bit can be set at any time to turn off the Analog Comparator. This will reduce power consumption in Active and Idle mode. When changing the ACD bit, the Analog Comparator Interrupt must be disabled by clearing the ACIE bit in ACSR. Otherwise an interrupt can occur when the bit is changed.

- **Bit 6 – ACBG: Analog Comparator Bandgap Select**

When this bit is set, a fixed bandgap reference voltage replaces the positive input to the Analog Comparator. When this bit is cleared, AIN0 is applied to the positive input of the Analog Comparator. See “Internal Voltage Reference” on page 52

- **Bit 5 – ACO: Analog Comparator Output**

The output of the Analog Comparator is synchronized and then directly connected to ACO. The synchronization introduces a delay of 1 - 2 clock cycles.

- **Bit 4 – ACI: Analog Comparator Interrupt Flag**

This bit is set by hardware when a comparator output event triggers the interrupt mode defined by ACIS1 and ACIS0. The Analog Comparator interrupt routine is executed if the ACIE bit is set and the I-bit in SREG is set. ACI is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ACI is cleared by writing a logic one to the flag.

- **Bit 3 – ACIE: Analog Comparator Interrupt Enable**

When the ACIE bit is written logic one and the I-bit in the Status Register is set, the Analog Comparator interrupt is activated. When written logic zero, the interrupt is disabled.

- **Bit 2 – ACIC: Analog Comparator Input Capture Enable**

When written logic one, this bit enables the input capture function in Timer/Counter1 to be triggered by the Analog Comparator. The comparator output is in this case directly connected to the input capture front-end logic, making the comparator utilize the noise canceler and edge select features of the Timer/Counter1 Input Capture interrupt. When written logic zero, no connection between the Analog Comparator and the input capture function exists. To make the comparator trigger the Timer/Counter1 Input Capture interrupt, the ICIE1 bit in the Timer Interrupt Mask Register (TIMSK1) must be set.

- **Bits 1, 0 – ACIS1, ACIS0: Analog Comparator Interrupt Mode Select**

These bits determine which comparator events that trigger the Analog Comparator interrupt. The different settings are shown in Table 97.

**Table 97.** ACIS1/ACIS0 Settings

| ACIS1 | ACIS0 | Interrupt Mode |
|---|---|---|
| 0 | 0 | Comparator Interrupt on Output Toggle. |
| 0 | 1 | Reserved |
| 1 | 0 | Comparator Interrupt on Falling Output Edge. |
| 1 | 1 | Comparator Interrupt on Rising Output Edge. |

When changing the ACIS1/ACIS0 bits, the Analog Comparator Interrupt must be disabled by clearing its Interrupt Enable bit in the ACSR Register. Otherwise an interrupt can occur when the bits are changed.

## Analog Comparator Multiplexed Input

It is possible to select any of the ADC7..0 pins to replace the negative input to the Analog Comparator. The ADC multiplexer is used to select this input, and consequently, the ADC must be switched off to utilize this feature. If the Analog Comparator Multiplexer Enable bit (ACME in ADCSRB) is set and the ADC is switched off (ADEN in ADCSRA is zero), MUX2..0 in ADMUX select the input pin to replace the negative input to the Analog Comparator, as shown in Table 98. If ACME is cleared or ADEN is set, AIN1 is applied to the negative input to the Analog Comparator.

**Table 98.** Analog Comparator Multiplexed Input

| ACME | ADEN | MUX2..0 | Analog Comparator Negative Input |
|---|---|---|---|
| 0 | x | xxx | AIN1 |
| 1 | 1 | xxx | AIN1 |
| 1 | 0 | 000 | ADC0 |
| 1 | 0 | 001 | ADC1 |
| 1 | 0 | 010 | ADC2 |
| 1 | 0 | 011 | ADC3 |
| 1 | 0 | 100 | ADC4 |
| 1 | 0 | 101 | ADC5 |
| 1 | 0 | 110 | ADC6 |
| 1 | 0 | 111 | ADC7 |

## Digital Input Disable Register 1 – DIDR1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | – | – | AIN1D | AIN0D | DIDR1 |
| Read/Write | R | R | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 1, 0 – AIN1D, AIN0D: AIN1, AIN0 Digital Input Disable**

When this bit is written logic one, the digital input buffer on the AIN1/0 pin is disabled. The corresponding PIN Register bit will always read as zero when this bit is set. When an analog signal is applied to the AIN1/0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.

# Analog to Digital Converter - ADC

## Features

- **10-bit Resolution**
- **0.5 LSB Integral Non-linearity**
- **± 2 LSB Absolute Accuracy**
- **65 - 260 µs Conversion Time**
- **Up to 15 kSPS at Maximum Resolution**
- **Eight Multiplexed Single Ended Input Channels**
- **Seven Differential input channels**
- **Optional Left Adjustment for ADC Result Readout**
- **0 - $V_{CC}$ ADC Input Voltage Range**
- **Selectable 2.56 V ADC Reference Voltage**
- **Free Running or Single Conversion Mode**
- **ADC Start Conversion by Auto Triggering on Interrupt Sources**
- **Interrupt on ADC Conversion Complete**
- **Sleep Mode Noise Canceler**

The AT90CAN128 features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel Analog Multiplexer which allows eight single-ended voltage inputs constructed from the pins of Port F. The single-ended voltage inputs refer to 0V (GND).

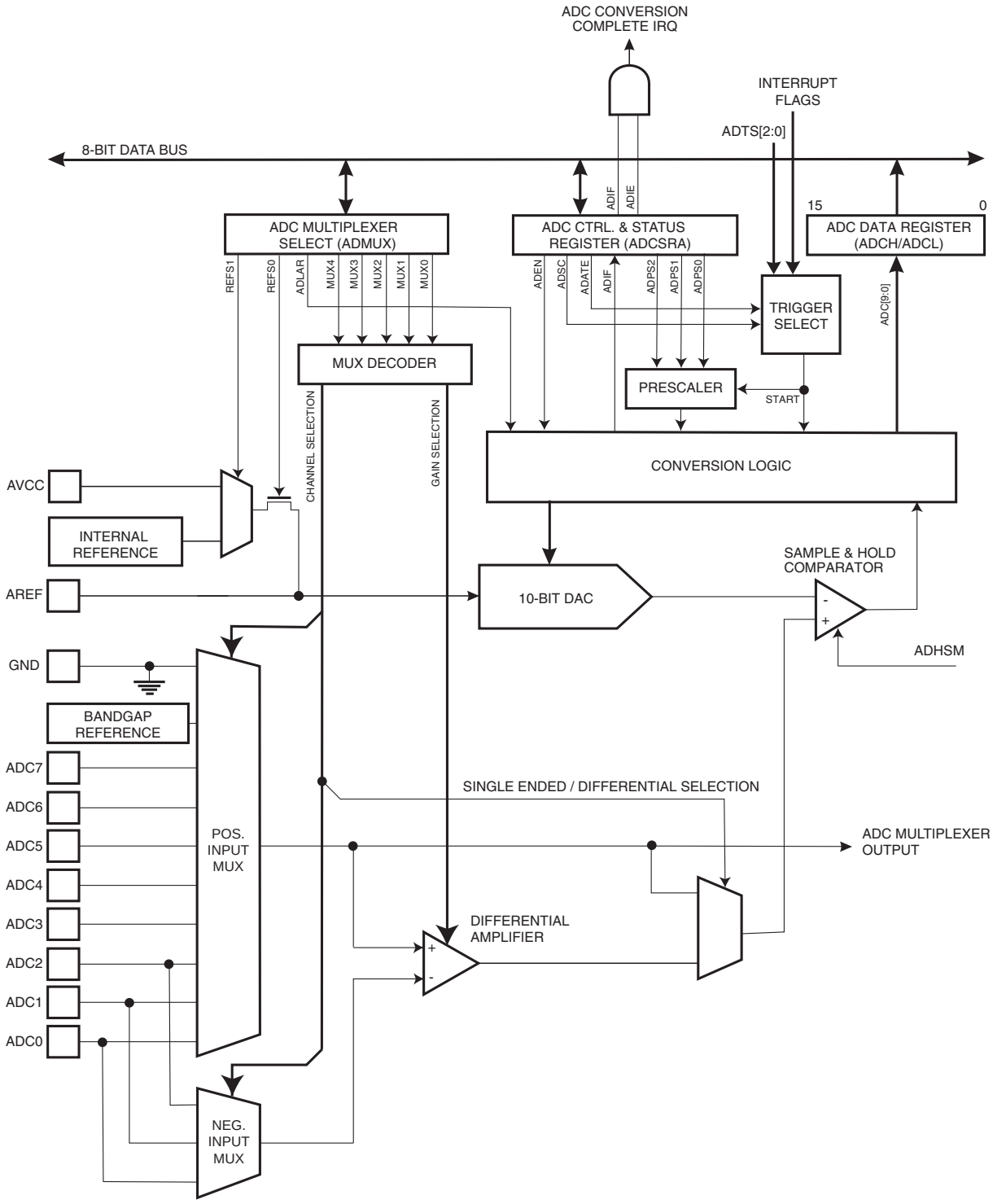
The device also supports 16 differential voltage input combinations. Two of the differential inputs (ADC1, ADC0 and ADC3, ADC2) are equipped with a programmable gain stage, providing amplification steps of 0 dB (1x), 20 dB (10x), or 46 dB (200x) on the differential input voltage before the A/D conversion. Seven differential analog input channels share a common negative terminal (ADC1), while any other ADC input can be selected as the positive input terminal. If 1x or 10x gain is used, 8-bit resolution can be expected. If 200x gain is used, 7-bit resolution can be expected.

The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown in Figure 127.

The ADC has a separate analog supply voltage pin, $AV_{CC}$. $AV_{CC}$ must not differ more than ± 0.3V from $V_{CC}$. See the paragraph “ADC Noise Canceler” on page 272 on how to connect this pin.

Internal reference voltages of nominally 2.56V or $AV_{CC}$ are provided On-chip. The voltage reference may be externally decoupled at the AREF pin by a capacitor for better noise performance.

**Figure 127.** Analog to Digital Converter Block Schematic



## Operation

The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Optionally, $AV_{CC}$ or an internal 2.56V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX Register. The internal voltage reference may thus be decoupled by an external capacitor at the AREF pin to improve noise immunity.

The analog input channel and differential gain are selected by writing to the MUX bits in ADMUX. Any of the ADC input pins, as well as GND and a fixed bandgap voltage reference, can be selected as single ended inputs to the ADC. A selection of ADC input pins can be selected as positive and negative inputs to the differential amplifier.

The ADC is enabled by setting the ADC Enable bit, ADEN in ADCSRA. Voltage reference and input channel selections will not go into effect until ADEN is set. The ADC does not consume power when ADEN is cleared, so it is recommended to switch off the ADC before entering power saving sleep modes.

The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADMUX.

If the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH, to ensure that the content of the Data Registers belongs to the same conversion. Once ADCL is read, ADC access to Data Registers is blocked. This means that if ADCL has been read, and a conversion completes before ADCH is read, neither register is updated and the result from the conversion is lost. When ADCH is read, ADC access to the ADCH and ADCL Registers is re-enabled.

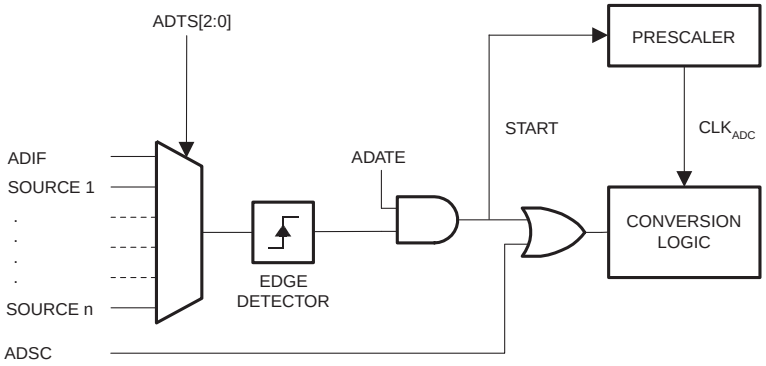
The ADC has its own interrupt which can be triggered when a conversion completes. The ADC access to the Data Registers is prohibited between reading of ADCH and ADCL, the interrupt will trigger even if the result is lost.

## Starting a Conversion

A single conversion is started by writing a logical one to the ADC Start Conversion bit, ADSC. This bit stays high as long as the conversion is in progress and will be cleared by hardware when the conversion is completed. If a different data channel is selected while a conversion is in progress, the ADC will finish the current conversion before performing the channel change.

Alternatively, a conversion can be triggered automatically by various sources. Auto Triggering is enabled by setting the ADC Auto Trigger Enable bit, ADATE in ADCSRA. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in ADCSRB (See description of the ADTS bits for a list of the trigger sources). When a positive edge occurs on the selected trigger signal, the ADC prescaler is reset and a conversion is started. This provides a method of starting conversions at fixed intervals. If the trigger signal is still set when the conversion completes, a new conversion will not be started. If another positive edge occurs on the trigger signal during conversion, the edge will be ignored. Note that an interrupt flag will be set even if the specific interrupt is disabled or the Global Interrupt Enable bit in SREG is cleared. A conversion can thus be triggered without causing an interrupt. However, the interrupt flag must be cleared in order to trigger a new conversion at the next interrupt event.

**Figure 128.** ADC Auto Trigger Logic

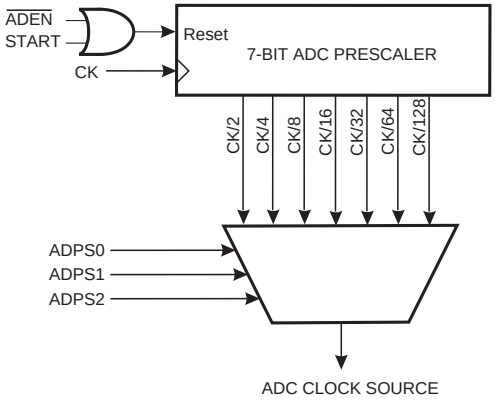



Using the ADC Interrupt Flag as a trigger source makes the ADC start a new conversion as soon as the ongoing conversion has finished. The ADC then operates in Free Running mode, constantly sampling and updating the ADC Data Register. The first conversion must be started by writing a logical one to the ADSC bit in ADCSRA. In this mode the ADC will perform successive conversions independently of whether the ADC Interrupt Flag, ADIF is cleared or not.

If Auto Triggering is enabled, single conversions can be started by writing ADSC in ADCSRA to one. ADSC can also be used to determine if a conversion is in progress. The ADSC bit will be read as one during a conversion, independently of how the conversion was started.

## Prescaling and Conversion Timing

**Figure 129.** ADC Prescaler




By default, the successive approximation circuitry requires an input clock frequency between 50 kHz and 200 kHz to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200 kHz to get a higher sample rate. Alternatively, setting the ADHSM bit in ADCSRB allows an increased ADC clock frequency at the expense of higher power consumption.

The ADC module contains a prescaler, which generates an acceptable ADC clock frequency from any CPU frequency above 100 kHz. The prescaling is set by the ADPS bits in ADCSRA. The prescaler starts counting from the moment the ADC is switched on by setting the ADEN bit in ADCSRA. The prescaler keeps running for as long as the ADEN bit is set, and is continuously reset when ADEN is low.

When initiating a single ended conversion by setting the ADSC bit in ADCSRA, the conversion starts at the following rising edge of the ADC clock cycle. See “Differential Channels” on page 270 for details on differential conversion timing.

A normal conversion takes 13 ADC clock cycles. The first conversion after the ADC is switched on (ADEN in ADCSRA is set) takes 25 ADC clock cycles in order to initialize the analog circuitry.

The actual sample-and-hold takes place 1.5 ADC clock cycles after the start of a normal conversion and 13.5 ADC clock cycles after the start of an first conversion. When a conversion is complete, the result is written to the ADC Data Registers, and ADIF is set. In Single Conversion mode, ADSC is cleared simultaneously. The software may then set ADSC again, and a new conversion will be initiated on the first rising ADC clock edge.

When Auto Triggering is used, the prescaler is reset when the trigger event occurs. This assures a fixed delay from the trigger event to the start of conversion. In this mode, the sample-and-hold takes place two ADC clock cycles after the rising edge on the trigger source signal. Three additional CPU clock cycles are used for synchronization logic.

In Free Running mode, a new conversion will be started immediately after the conversion completes, while ADSC remains high. For a summary of conversion times, see Table 99.

**Figure 130.** ADC Timing Diagram, First Conversion (Single Conversion Mode)

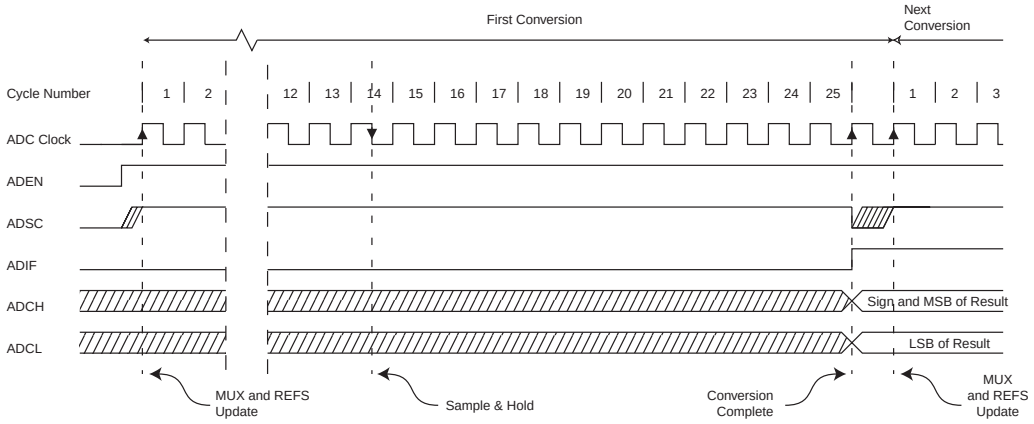



**Figure 131.** ADC Timing Diagram, Single Conversion

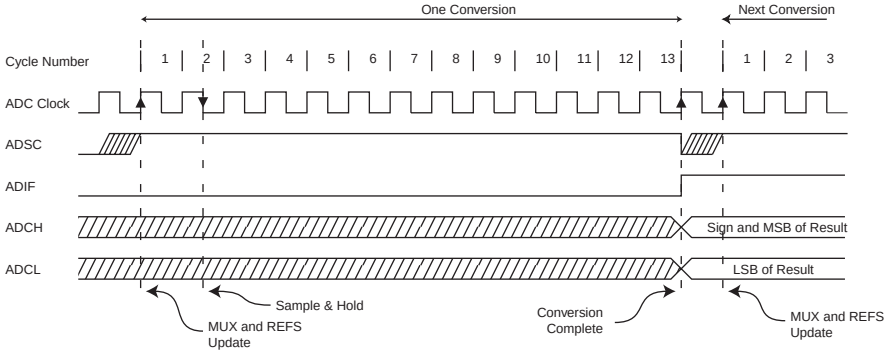


**Figure 132.** ADC Timing Diagram, Auto Triggered Conversion

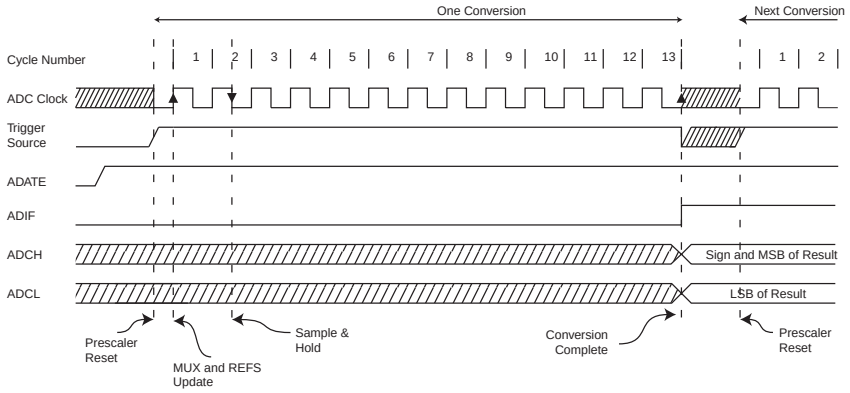



**Figure 133.** ADC Timing Diagram, Free Running Conversion

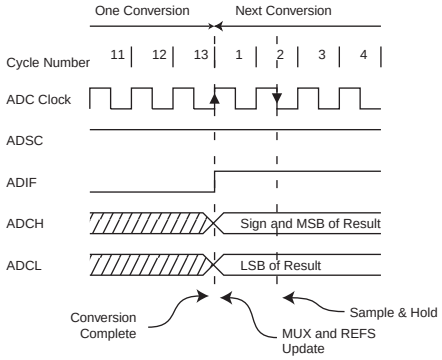



**Table 99.** ADC Conversion Time

| Condition | First Conversion | Normal Conversion, Single Ended | Auto Triggered Convertion |
|---|---|---|---|
| Sample & Hold (Cycles from Start of Convertion) | 14.5 | 1.5 | 2 |
| Conversion Time (Cycles) | 25 | 13 | 13.5 |

**Differential Channels**

When using differential channels, certain aspects of the conversion need to be taken into consideration.

Differential conversions are synchronized to the internal clock $CK_{ADC2}$ equal to half the ADC clock frequency. This synchronization is done automatically by the ADC interface in such a way that the sample-and-hold occurs at a specific phase of $CK_{ADC2}$. A conversion initiated by the user (i.e., all single conversions, and the first free running conversion) when $CK_{ADC2}$ is low will take the same amount of time as a single ended conversion (13 ADC clock cycles from the next prescaled clock cycle). A conversion initiated by the user when $CK_{ADC2}$ is high will take 14 ADC clock cycles due to the synchronization mechanism. In Free Running mode, a new conversion is initiated imme-

diately after the previous conversion completes, and since $CK_{ADC2}$ is high at this time, all automatically started (i.e., all but the first) Free Running conversions will take 14 ADC clock cycles.

If differential channels are used and conversions are started by Auto Triggering, the ADC must be switched off between conversions. When Auto Triggering is used, the ADC prescaler is reset before the conversion is started. Since the stage is dependent of a stable ADC clock prior to the conversion, this conversion will not be valid. By disabling and then re-enabling the ADC between each conversion (writing ADEN in ADCSRA to “0” then to “1”), only extended conversions are performed. The result from the extended conversions will be valid. See “Prescaling and Conversion Timing” on page 268 for timing details.

The gain stage is optimized for a bandwidth of 4 kHz at all gain settings. Higher frequencies may be subjected to non-linear amplification. An external low-pass filter should be used if the input signal contains higher frequency components than the gain stage bandwidth. Note that the ADC clock frequency is independent of the gain stage bandwidth limitation. E.g. the ADC clock period may be 6 µs, allowing a channel to be sampled at 12 kSPS, regardless of the bandwidth of this channel.

## Changing Channel or Reference Selection

The MUXn and REFS1:0 bits in the ADMUX Register are single buffered through a temporary register to which the CPU has random access. This ensures that the channels and reference selection only takes place at a safe point during the conversion. The channel and reference selection is continuously updated until a conversion is started. Once the conversion starts, the channel and reference selection is locked to ensure a sufficient sampling time for the ADC. Continuous updating resumes in the last ADC clock cycle before the conversion completes (ADIF in ADCSRA is set). Note that the conversion starts on the following rising ADC clock edge after ADSC is written. The user is thus advised not to write new channel or reference selection values to ADMUX until one ADC clock cycle after ADSC is written.

If Auto Triggering is used, the exact time of the triggering event can be indeterministic. Special care must be taken when updating the ADMUX Register, in order to control which conversion will be affected by the new settings.

If both ADATE and ADEN is written to one, an interrupt event can occur at any time. If the ADMUX Register is changed in this period, the user cannot tell if the next conversion is based on the old or the new settings. ADMUX can be safely updated in the following ways:

1. When ADATE or ADEN is cleared.
2. During conversion, minimum one ADC clock cycle after the trigger event.
3. After a conversion, before the interrupt flag used as trigger source is cleared.

When updating ADMUX in one of these conditions, the new settings will affect the next ADC conversion.

Special care should be taken when changing differential channels. Once a differential channel has been selected, the stage may take as much as 125 µs to stabilize to the new value. Thus conversions should not be started within the first 125 µs after selecting a new differential channel. Alternatively, conversion results obtained within this period should be discarded.

The same settling time should be observed for the first differential conversion after changing ADC reference (by changing the REFS1:0 bits in ADMUX).

The settling time and gain stage bandwidth is independent of the ADHSM bit setting.

### ADC Input Channels

When changing channel selections, the user should observe the following guidelines to ensure that the correct channel is selected:

- In Single Conversion mode, always select the channel before starting the conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the conversion to complete before changing the channel selection.
- In Free Running mode, always select the channel before starting the first conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the first conversion to complete, and then change the channel selection. Since the next conversion has already started automatically, the next result will reflect the previous channel selection. Subsequent conversions will reflect the new channel selection.

When switching to a differential gain channel, the first conversion result may have a poor accuracy due to the required settling time for the automatic offset cancellation circuitry. The user should preferably disregard the first conversion result.

### ADC Voltage Reference

The reference voltage for the ADC ($V_{REF}$) indicates the conversion range for the ADC. Single ended channels that exceed $V_{REF}$ will result in codes close to 0x3FF. $V_{REF}$ can be selected as either $AV_{CC}$, internal 2.56V reference, or external AREF pin.

$AV_{CC}$ is connected to the ADC through a passive switch. The internal 2.56V reference is generated from the internal bandgap reference ($V_{BG}$) through an internal amplifier. In either case, the external AREF pin is directly connected to the ADC, and the reference voltage can be made more immune to noise by connecting a capacitor between the AREF pin and ground. $V_{REF}$ can also be measured at the AREF pin with a high impedant voltmeter. Note that $V_{REF}$ is a high impedant source, and only a capacitive load should be connected in a system.

If the user has a fixed voltage source connected to the AREF pin, the user may not use the other reference voltage options in the application, as they will be shorted to the external voltage. If no external voltage is applied to the AREF pin, the user may switch between $AV_{CC}$ and 2.56V as reference selection. The first ADC conversion result after switching reference voltage source may be inaccurate, and the user is advised to discard this result.

If differential channels are used, the selected reference should not be closer to $AV_{CC}$ than indicated in Table 140 on page 363.

## ADC Noise Canceler

The ADC features a noise canceler that enables conversion during sleep mode to reduce noise induced from the CPU core and other I/O peripherals. The noise canceler can be used with ADC Noise Reduction and Idle mode. To make use of this feature, the following procedure should be used:

1. Make sure that the ADC is enabled and is not busy converting. Single Conversion mode must be selected and the ADC conversion complete interrupt must be enabled.
2. Enter ADC Noise Reduction mode (or Idle mode). The ADC will start a conversion once the CPU has been halted.
3. If no other interrupts occur before the ADC conversion completes, the ADC interrupt will wake up the CPU and execute the ADC Conversion Complete interrupt routine. If another interrupt wakes up the CPU before the ADC conversion is complete, that interrupt will be executed, and an ADC Conversion Complete interrupt request will be generated when the ADC conversion

completes. The CPU will remain in active mode until a new sleep command is executed.

Note that the ADC will not be automatically turned off when entering other sleep modes than Idle mode and ADC Noise Reduction mode. The user is advised to write zero to ADEN before entering such sleep modes to avoid excessive power consumption.

If the ADC is enabled in such sleep modes and the user wants to perform differential conversions, the user is advised to switch the ADC off and on after waking up from sleep to prompt an extended conversion to get a valid result.

### Analog Input Circuitry

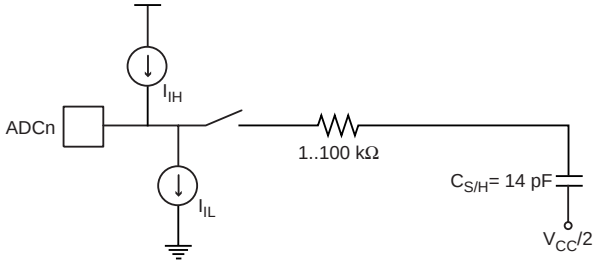
The analog input circuitry for single ended channels is illustrated in Figure 134. An analog source applied to ADCn is subjected to the pin capacitance and input leakage of that pin, regardless of whether that channel is selected as input for the ADC. When the channel is selected, the source must drive the S/H capacitor through the series resistance (combined resistance in the input path).

The ADC is optimized for analog signals with an output impedance of approximately 10 kΩ or less. If such a source is used, the sampling time will be negligible. If a source with higher impedance is used, the sampling time will depend on how long time the source needs to charge the S/H capacitor, with can vary widely. The user is recommended to only use low impedant sources with slowly varying signals, since this minimizes the required charge transfer to the S/H capacitor.

If differential gain channels are used, the input circuitry looks somewhat different, although source impedances of a few hundred kΩ or less is recommended.

Signal components higher than the Nyquist frequency ($f_{ADC}/2$) should not be present for either kind of channels, to avoid distortion from unpredictable signal convolution. The user is advised to remove high frequency components with a low-pass filter before applying the signals as inputs to the ADC.

**Figure 134.** Analog Input Circuitry

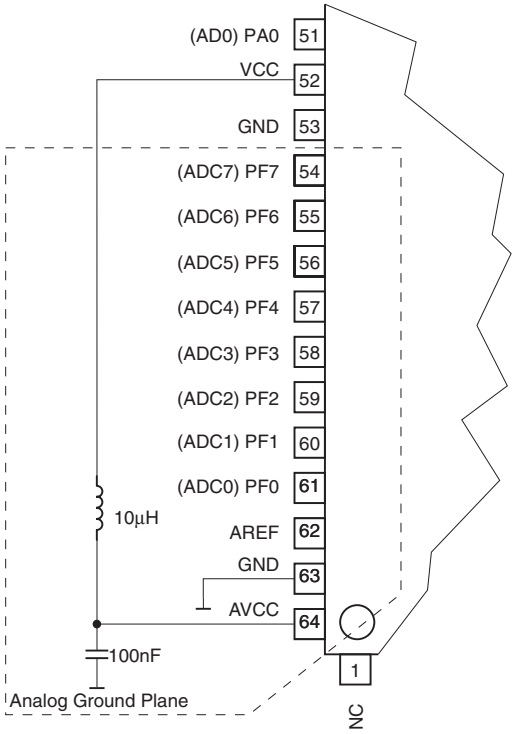



### Analog Noise Canceling Techniques

Digital circuitry inside and outside the device generates EMI which might affect the accuracy of analog measurements. If conversion accuracy is critical, the noise level can be reduced by applying the following techniques:

1. Keep analog signal paths as short as possible. Make sure analog tracks run over the analog ground plane, and keep them well away from high-speed switching digital tracks.
2. The $AV_{CC}$ pin on the device should be connected to the digital $V_{CC}$ supply voltage via an LC network as shown in Figure 135.
3. Use the ADC noise canceler function to reduce induced noise from the CPU.
4. If any ADC port pins are used as digital outputs, it is essential that these do not switch while a conversion is in progress.

**Figure 135.** ADC Power Connections




**Offset Compensation Schemes**

The gain stage has a built-in offset cancellation circuitry that nulls the offset of differential measurements as much as possible. The remaining offset in the analog path can be measured directly by selecting the same channel for both differential inputs. This offset residue can be then subtracted in software from the measurement results. Using this kind of software based offset correction, offset on any channel can be reduced below one LSB.

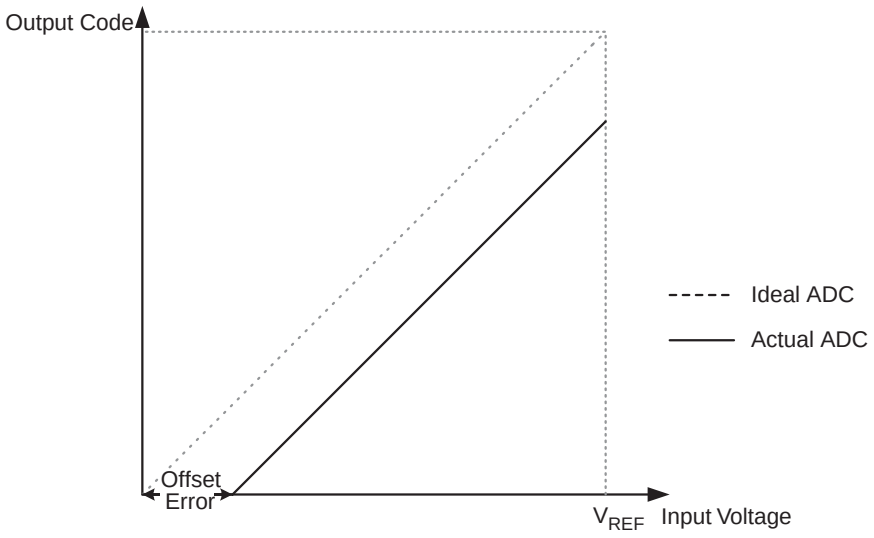
**ADC Accuracy Definitions**

An n-bit single-ended ADC converts a voltage linearly between GND and $V_{REF}$ in $2^n$ steps (LSBs). The lowest code is read as 0, and the highest code is read as $2^n$-1.

Several parameters describe the deviation from the ideal behavior:

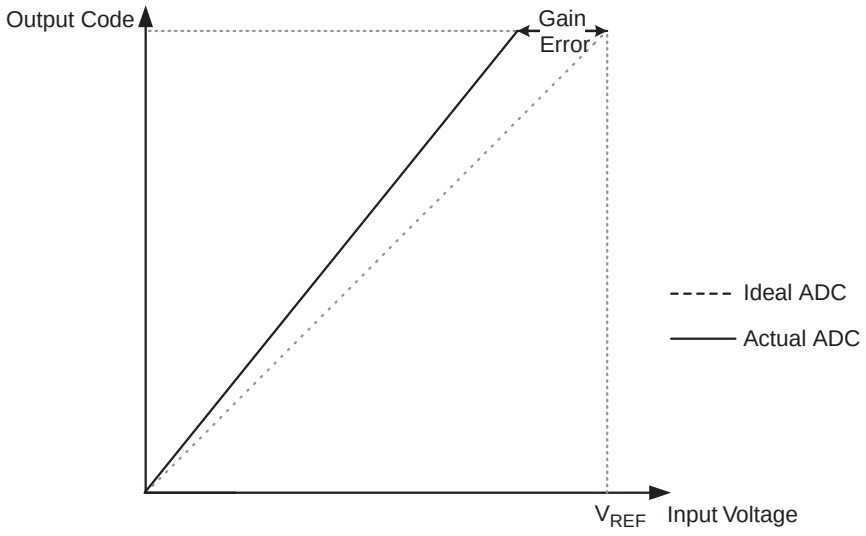
- Offset: The deviation of the first transition (0x000 to 0x001) compared to the ideal transition (at 0.5 LSB). Ideal value: 0 LSB.

**Figure 136.** Offset Error



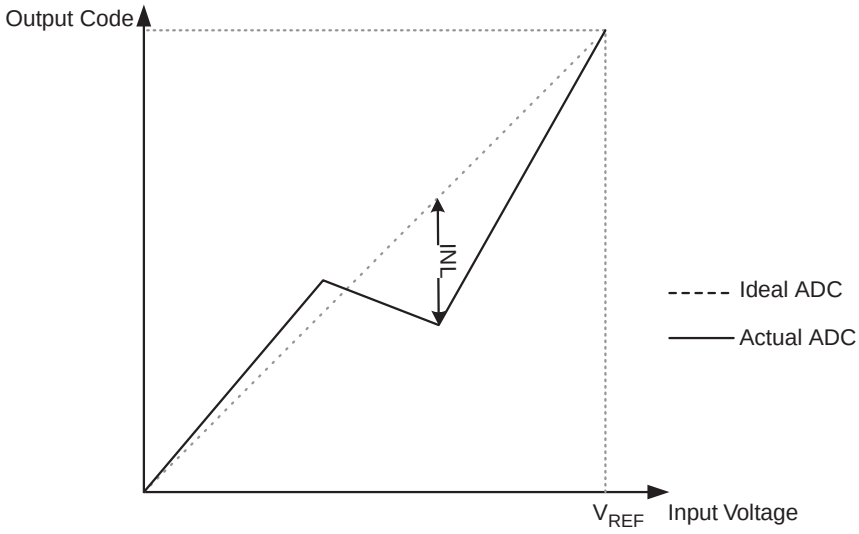

- Gain Error: After adjusting for offset, the Gain Error is found as the deviation of the last transition (0x3FE to 0x3FF) compared to the ideal transition (at 1.5 LSB below maximum). Ideal value: 0 LSB

**Figure 137.** Gain Error



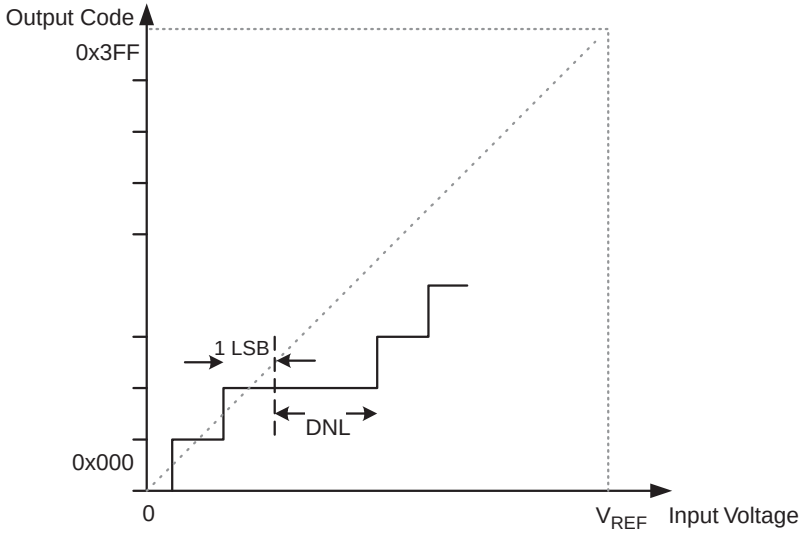
- Integral Non-linearity (INL): After adjusting for offset and gain error, the INL is the maximum deviation of an actual transition compared to an ideal transition for any code. Ideal value: 0 LSB.

**Figure 138.** Integral Non-linearity (INL)




- Differential Non-linearity (DNL): The maximum deviation of the actual code width (the interval between two adjacent transitions) from the ideal code width (1 LSB). Ideal value: 0 LSB.

**Figure 139.** Differential Non-linearity (DNL)




- Quantization Error: Due to the quantization of the input voltage into a finite number of codes, a range of input voltages (1 LSB wide) will code to the same value. Always ± 0.5 LSB.
- Absolute Accuracy: The maximum deviation of an actual (unadjusted) transition compared to an ideal transition for any code. This is the compound effect of offset, gain error, differential error, non-linearity, and quantization error. Ideal value: ± 0.5 LSB.

## ADC Conversion Result

After the conversion is complete (ADIF is high), the conversion result can be found in the ADC Result Registers (ADCL, ADCH).

For single ended conversion, the result is:

$$ADC = \frac{V_{IN} \cdot 1023}{V_{REF}}$$

where $V_{IN}$ is the voltage on the selected input pin and $V_{REF}$ the selected voltage reference (see Table 101 on page 279 and Table 102 on page 280). 0x000 represents analog ground, and 0x3FF represents the selected reference voltage minus one LSB.

If differential channels are used, the result is:

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot GAIN \cdot 512}{V_{REF}}$$

where $V_{POS}$ is the voltage on the positive input pin, $V_{NEG}$ the voltage on the negative input pin, GAIN the selected gain factor and $V_{REF}$ the selected voltage reference. The result is presented in two's complement form, from 0x200 (-512d) through 0x1FF (+511d). Note that if the user wants to perform a quick polarity check of the result, it is sufficient to read the MSB of the result (ADC9 in ADCH). If the bit is one, the result is negative, and if this bit is zero, the result is positive. Figure 140 shows the decoding of the differential input range.

Table 82 shows the resulting output codes if the differential input channel pair (ADCn - ADCm) is selected with a reference voltage of $V_{REF}$.

**Figure 140.** Differential Measurement Range

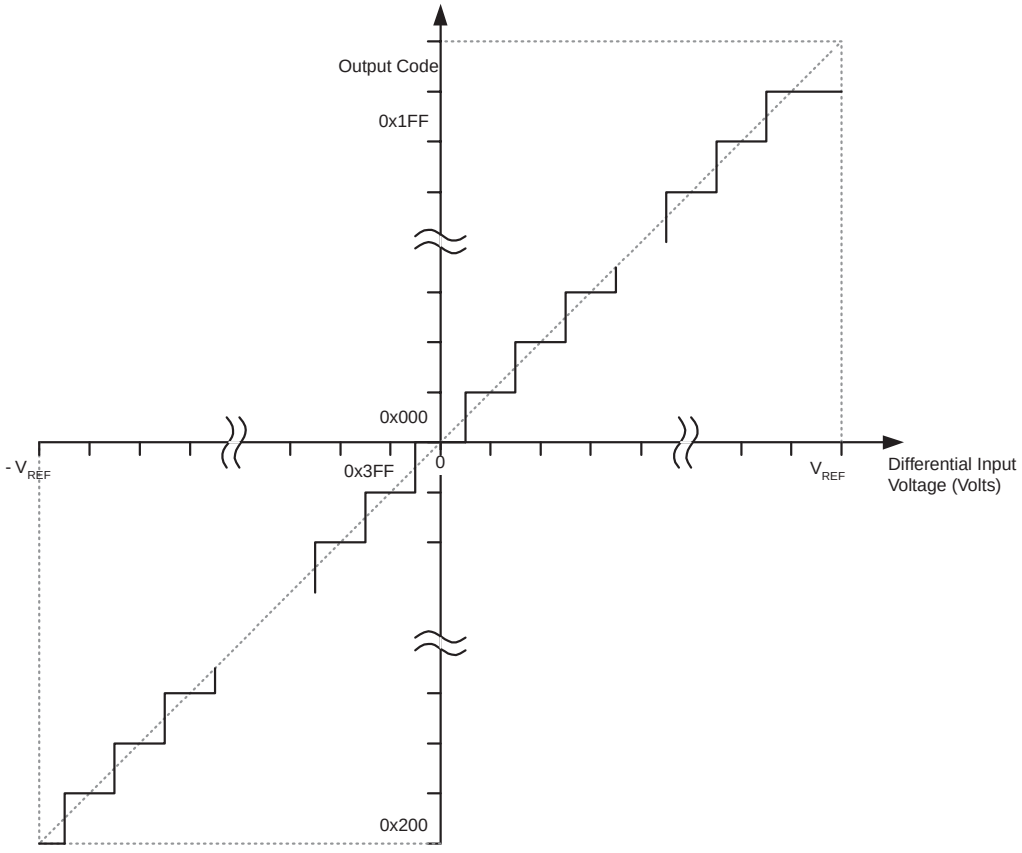


**Table 100.** Correlation Between Input Voltage and Output Codes

| $V_{ADCn}$ | Read code | Corresponding decimal value |
|---|---|---|
| $V_{ADCm} + V_{REF}/GAIN$ | 0x1FF | 511 |
| $V_{ADCm} + 0.999\ V_{REF}/GAIN$ | 0x1FF | 511 |
| $V_{ADCm} + 0.998\ V_{REF}/GAIN$ | 0x1FE | 510 |
| ... | ... | ... |
| $V_{ADCm} + 0.001\ V_{REF}/GAIN$ | 0x001 | 1 |
| $V_{ADCm}$ | 0x000 | 0 |
| $V_{ADCm} - 0.001\ V_{REF}/GAIN$ | 0x3FF | -1 |
| ... | ... | ... |
| $V_{ADCm} - 0.999\ V_{REF}/GAIN$ | 0x201 | -511 |
| $V_{ADCm} - V_{REF}/GAIN$ | 0x200 | -512 |

Example 1:

- ADMUX = 0xED (ADC3 - ADC2, 10x gain, 2.56V reference, left adjusted result)
- Voltage on ADC3 is 300 mV, voltage on ADC2 is 500 mV.
- ADCR = 512 * 10 * (300 - 500) / 2560 = -400 = 0x270
- ADCL will thus read 0x00, and ADCH will read 0x9C.
  Writing zero to ADLAR right adjusts the result: ADCL = 0x70, ADCH = 0x02.

Example 2:

- ADMUX = 0xFB (ADC3 - ADC2, 1x gain, 2.56V reference, left adjusted result)
- Voltage on ADC3 is 300 mV, voltage on ADC2 is 500 mV.
- ADCR = 512 * 1 * (300 - 500) / 2560 = -41 = 0x029.
- ADCL will thus read 0x40, and ADCH will read 0x0A.
  Writing zero to ADLAR right adjusts the result: ADCL = 0x00, ADCH = 0x29.

## ADC Register Description

### ADC Multiplexer Selection Register – ADMUX

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | REFS1 | REFS0 | ADLAR | MUX4 | MUX3 | MUX2 | MUX1 | MUX0 | ADMUX |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7:6 – REFS1:0: Reference Selection Bits**

These bits select the voltage reference for the ADC, as shown in Table 101. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

**Table 101.** Voltage Reference Selections for ADC

| REFS1 | REFS0 | Voltage Reference Selection |
|---|---|---|
| 0 | 0 | AREF, Internal Vref turned off |
| 0 | 1 | $AV_{CC}$ with external capacitor on AREF pin |
| 1 | 0 | Reserved |
| 1 | 1 | Internal 2.56V Voltage Reference with external capacitor on AREF pin |

- **Bit 5 – ADLAR: ADC Left Adjust Result**

The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately, regardless of any ongoing conversions. For a complete description of this bit, see “The ADC Data Register – ADCL and ADCH” on page 282.

- **Bits 4:0 – MUX4:0: Analog Channel Selection Bits**

The value of these bits selects which combination of analog inputs are connected to the ADC. These bits also select the gain for the differential channels. See Table 102 for details. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set).

**Table 102.** Input Channel and Gain Selections

| MUX4..0 | Single Ended Input | Positive Differential Input | Negative Differential Input | Gain |
|---|---|---|---|---|
| 00000 | ADC0 | **N/A** | | |
| 00001 | ADC1 | | | |
| 00010 | ADC2 | | | |
| 00011 | ADC3 | | | |
| 00100 | ADC4 | | | |
| 00101 | ADC5 | | | |
| 00110 | ADC6 | | | |
| 00111 | ADC7 | | | |
| 01000 | **N/A** | *(ADC0 / ADC0 / 10x)* | | |
| 01001 | | ADC1 | ADC0 | 10x |
| 01010 | | *(ADC0 / ADC0 / 200x)* | | |
| 01011 | | ADC1 | ADC0 | 200x |
| 01100 | | *(Reserved - ADC2 / ADC2 / 10x)* | | |
| 01101 | | ADC3 | ADC2 | 10x |
| 01110 | | *(ADC2 / ADC2 / 200x)* | | |
| 01111 | | ADC3 | ADC2 | 200x |
| 10000 | | ADC0 | ADC1 | 1x |
| 10001 | | *(ADC1 / ADC1 / 1x)* | | |
| 10010 | | ADC2 | ADC1 | 1x |
| 10011 | | ADC3 | ADC1 | 1x |
| 10100 | | ADC4 | ADC1 | 1x |
| 10101 | | ADC5 | ADC1 | 1x |
| 10110 | | ADC6 | ADC1 | 1x |
| 10111 | | ADC7 | ADC1 | 1x |
| 11000 | | ADC0 | ADC2 | 1x |
| 11001 | | ADC1 | ADC2 | 1x |
| 11010 | | *(ADC2 / ADC2 / 1x)* | | |
| 11011 | | ADC3 | ADC2 | 1x |
| 11100 | | ADC4 | ADC2 | 1x |
| 11101 | | ADC5 | ADC2 | 1x |
| 11110 | 1.1V ($V_{Band\ Gap}$) | **N/A** | | |
| 11111 | 0V (GND) | | | |

### ADC Control and Status Register A – ADCSRA

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | ADEN | ADSC | ADATE | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 | ADCSRA |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – ADEN: ADC Enable**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**

In Single Conversion mode, write this bit to one to start each conversion. In Free Running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

- **Bit 5 – ADATE: ADC Auto Trigger Enable**

When this bit is written to one, Auto Triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in ADCSRB.

- **Bit 4 – ADIF: ADC Interrupt Flag**

This bit is set when an ADC conversion completes and the Data Registers are updated. The ADC Conversion Complete Interrupt is executed if the ADIE bit and the I-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a Read-Modify-Write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

- **Bit 3 – ADIE: ADC Interrupt Enable**

When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

- **Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits**

These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

**Table 103.** ADC Prescaler Selections

| ADPS2 | ADPS1 | ADPS0 | Division Factor |
|---|---|---|---|
| 0 | 0 | 0 | 2 |
| 0 | 0 | 1 | 2 |
| 0 | 1 | 0 | 4 |
| 0 | 1 | 1 | 8 |
| 1 | 0 | 0 | 16 |
| 1 | 0 | 1 | 32 |
| 1 | 1 | 0 | 64 |
| 1 | 1 | 1 | 128 |

### The ADC Data Register – ADCL and ADCH

ADLAR = 0

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | – | – | ADC9 | ADC8 | ADCH |
| | ADC7 | ADC6 | ADC5 | ADC4 | ADC3 | ADC2 | ADC1 | ADC0 | ADCL |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Read/Write | R | R | R | R | R | R | R | R | |
| | R | R | R | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

ADLAR = 1

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | ADC9 | ADC8 | ADC7 | ADC6 | ADC5 | ADC4 | ADC3 | ADC2 | ADCH |
| | ADC1 | ADC0 | – | – | – | – | – | – | ADCL |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Read/Write | R | R | R | R | R | R | R | R | |
| | R | R | R | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

When an ADC conversion is complete, the result is found in these two registers. If differential channels are used, the result is presented in two's complement form.

When ADCL is read, the ADC Data Register is not updated until ADCH is read. Consequently, if the result is left adjusted and no more than 8-bit precision (7 bit + sign bit for differential input channels) is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH.

The ADLAR bit in ADMUX, and the MUXn bits in ADMUX affect the way the result is read from the registers. If ADLAR is set, the result is left adjusted. If ADLAR is cleared (default), the result is right adjusted.

- **ADC9:0: ADC Conversion Result**

These bits represent the result from the conversion, as detailed in “ADC Conversion Result” on page 277.

### ADC Control and Status Register B – ADCSRB

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | ADHSM | ACME | – | – | – | ADTS2 | ADTS1 | ADTS0 | ADCSRB |
| Read/Write | R/W | R/W | R | R | R | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – ADHSM: ADC High Speed Mode**

Writing this bit to one enables the ADC High Speed mode. This mode enables higher conversion rate at the expense of higher power consumption.

- **Bit 2:0 – ADTS2:0: ADC Auto Trigger Source**

If ADATE in ADCSRA is written to one, the value of these bits selects which source will trigger an ADC conversion. If ADATE is cleared, the ADTS2:0 settings will have no effect. A conversion will be triggered by the rising edge of the selected interrupt flag. Note that switching from a trigger source that is cleared to a trigger source that is set, will generate a positive edge on the trigger signal. If ADEN in ADCSRA is set, this will start a conversion. Switching to Free Running mode (ADTS[2:0]=0) will not cause a trigger event, even if the ADC Interrupt Flag is set.

**Table 104.** ADC Auto Trigger Source Selections

| ADTS2 | ADTS1 | ADTS0 | Trigger Source |
|---|---|---|---|
| 0 | 0 | 0 | Free Running mode |
| 0 | 0 | 1 | Analog Comparator |
| 0 | 1 | 0 | External Interrupt Request 0 |
| 0 | 1 | 1 | Timer/Counter0 Compare Match |
| 1 | 0 | 0 | Timer/Counter0 Overflow |
| 1 | 0 | 1 | Timer/Counter1 Compare Match B |
| 1 | 1 | 0 | Timer/Counter1 Overflow |
| 1 | 1 | 1 | Timer/Counter1 Capture Event |

### Digital Input Disable Register 0 – DIDR0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | ADC7D | ADC6D | ADC5D | ADC4D | ADC3D | ADC2D | ADC1D | ADC0D | DIDR0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7:0 – ADC7D..ADC0D: ADC7:0 Digital Input Disable**

When this bit is written logic one, the digital input buffer on the corresponding ADC pin is disabled. The corresponding PIN Register bit will always read as zero when this bit is set. When an analog signal is applied to the ADC7..0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.

# JTAG Interface and On-chip Debug System

## Features

- **JTAG (IEEE std. 1149.1 Compliant) Interface**
- **Boundary-scan Capabilities According to the IEEE std. 1149.1 (JTAG) Standard**
- **Debugger Access to:**
  - **All Internal Peripheral Units**
  - **Internal and External RAM**
  - **The Internal Register File**
  - **Program Counter**
  - **EEPROM and Flash Memories**
- **Extensive On-chip Debug Support for Break Conditions, Including**
  - **AVR Break Instruction**
  - **Break on Change of Program Memory Flow**
  - **Single Step Break**
  - **Program Memory Break Points on Single Address or Address Range**
  - **Data Memory Break Points on Single Address or Address Range**
- **Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface**
- **On-chip Debugging Supported by AVR Studio®**

## Overview

The AVR IEEE std. 1149.1 compliant JTAG interface can be used for:

- Testing PCBs by using the JTAG Boundary-scan capability
- Programming the non-volatile memories, Fuses and Lock bits
- On-chip debugging

A brief description is given in the following sections. Detailed descriptions for Programming via the JTAG interface, and using the Boundary-scan Chain can be found in the sections “JTAG Programming Overview” on page 342 and “Boundary-scan IEEE 1149.1 (JTAG)” on page 290**,** respectively. The On-chip Debug support is considered being private JTAG instructions, and distributed within ATMEL and to selected third party vendors only.

Figure 141 shows a block diagram of the JTAG interface and the On-chip Debug system. The TAP Controller is a state machine controlled by the TCK and TMS signals. The TAP Controller selects either the JTAG Instruction Register or one of several Data Registers as the scan chain (Shift Register) between the TDI – input and TDO – output. The Instruction Register holds JTAG instructions controlling the behavior of a Data Register.

The ID-Register (IDentifier Register), Bypass Register, and the Boundary-scan Chain are the Data Registers used for board-level testing. The JTAG Programming Interface (actually consisting of several physical and virtual Data Registers) is used for serial programming via the JTAG interface. The Internal Scan Chain and Break Point Scan Chain are used for On-chip debugging only.

## Test Access Port – TAP

The JTAG interface is accessed through four of the AVR’s pins. In JTAG terminology, these pins constitute the Test Access Port – TAP. These pins are:

- **TMS**: Test mode select. This pin is used for navigating through the TAP-controller state machine.
- **TCK**: Test Clock. JTAG operation is synchronous to TCK.
- **TDI**: Test Data In. Serial input data to be shifted in to the Instruction Register or Data Register (Scan Chains).
- **TDO**: Test Data Out. Serial output data from Instruction Register or Data Register (Scan Chains).

The IEEE std. 1149.1 also specifies an optional TAP signal; TRST – Test ReSeT – which is not provided.

When the JTAGEN fuse is unprogrammed, these four TAP pins are normal port pins and the TAP controller is in reset. When programmed and the JTD bit in MCUCR is cleared, the TAP input signals are internally pulled high and the JTAG is enabled for Boundary-scan and programming. In this case, the TAP output pin (TDO) is left floating in states where the JTAG TAP controller is not shifting data, and must therefore be con nected to a pull-up resistor or other hardware having pull-ups (for instance the TDI-input of the next device in the scan chain). The device is shipped with this fuse programmed.

For the On-chip Debug system, in addition to the JTAG interface pins, the $\overline{\text{RESET}}$ pin is monitored by the debugger to be able to detect external reset sources. The debugger can also pull the $\overline{\text{RESET}}$ pin low to reset the whole system, assuming only open collectors on the reset line are used in the application.

**Figure 141.** Block Diagram

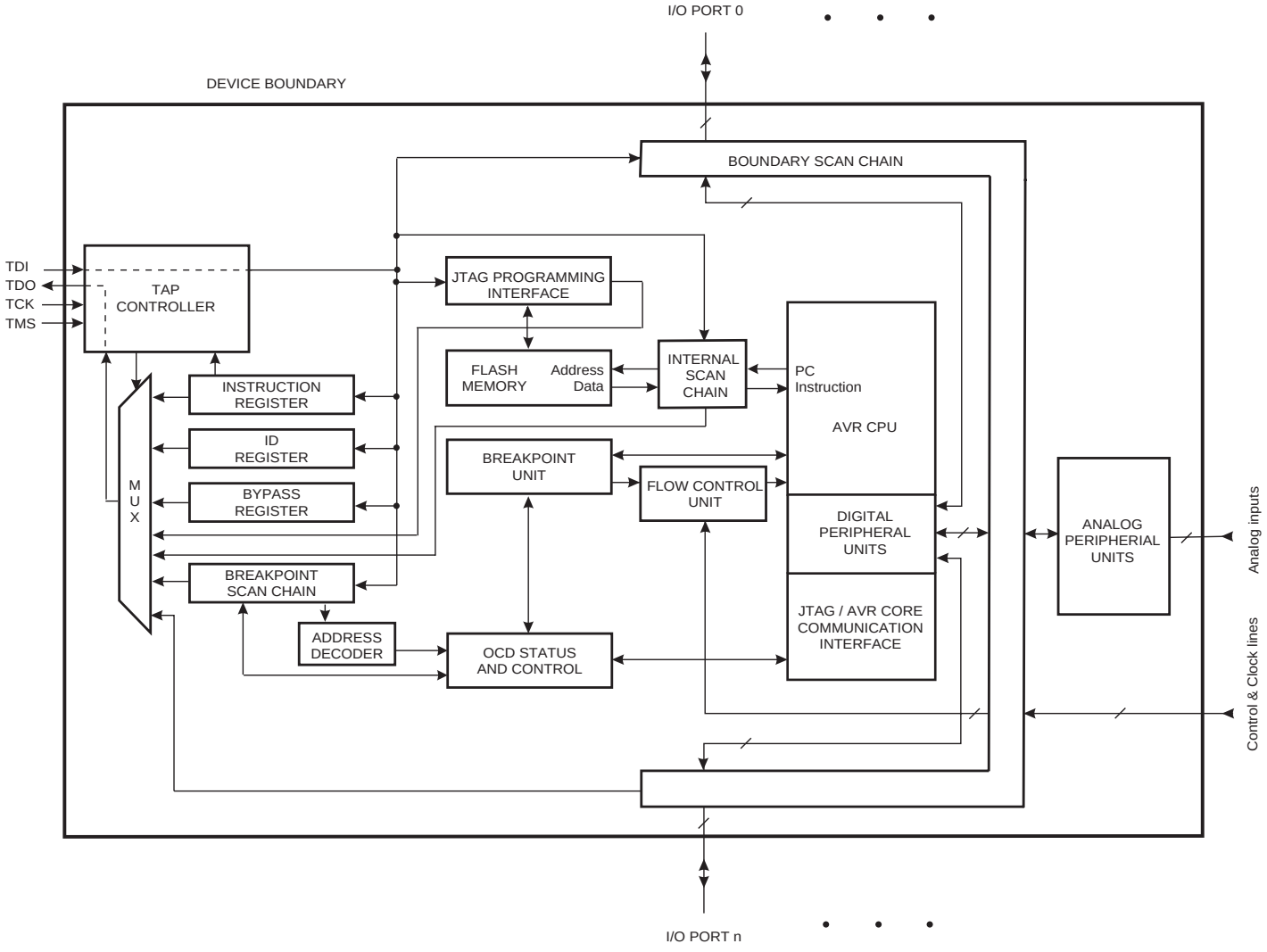
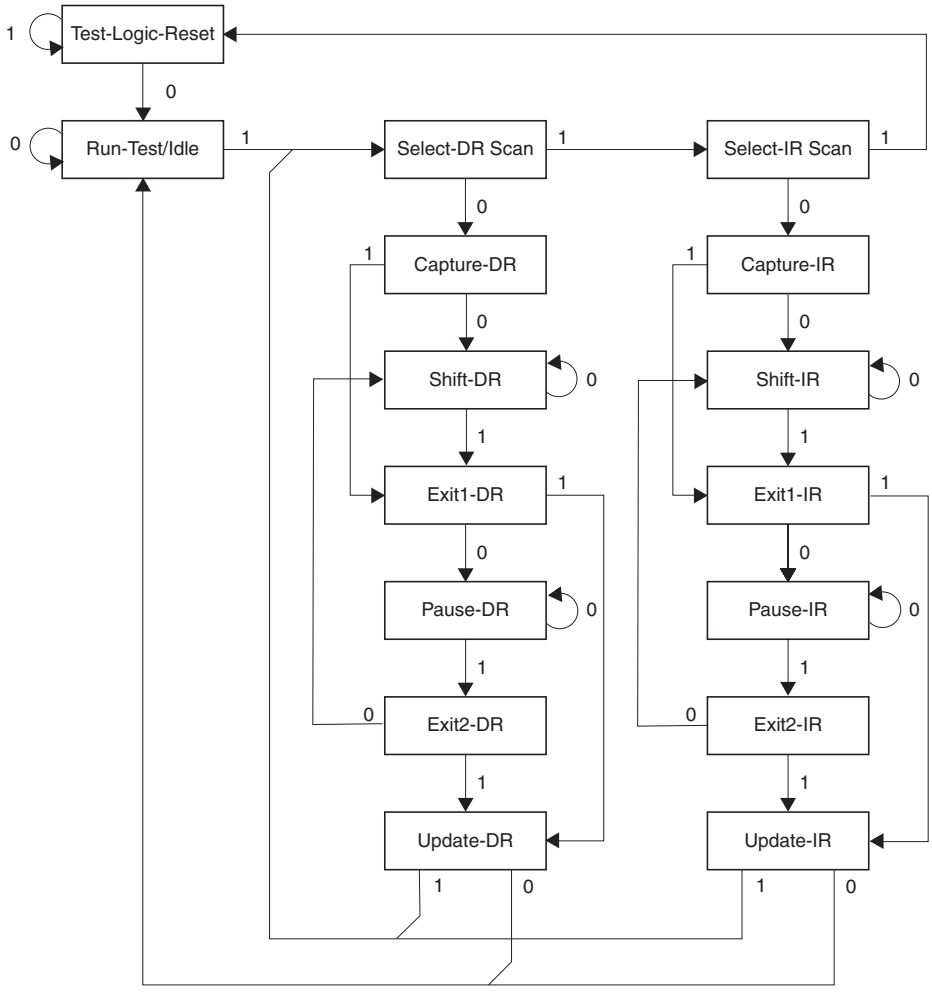


**Figure 142.** TAP Controller State Diagram




## TAP Controller

The TAP controller is a 16-state finite state machine that controls the operation of the Boundary-scan circuitry, JTAG programming circuitry, or On-chip Debug system. The state transitions depicted in Figure 142 depend on the signal present on TMS (shown adjacent to each state transition) at the time of the rising edge at TCK. The initial state after a Power-on Reset is Test-Logic-Reset.

As a definition in this document, the LSB is shifted in and out first for all Shift Registers.

Assuming Run-Test/Idle is the present state, a typical scenario for using the JTAG interface is:

- At the TMS input, apply the sequence 1, 1, 0, 0 at the rising edges of TCK to enter the Shift Instruction Register – Shift-IR state. While in this state, shift the four bits of the JTAG instructions into the JTAG Instruction Register from the TDI input at the rising edge of TCK. The TMS input must be held low during input of the 3 LSBs in order to remain in the Shift-IR state. The MSB of the instruction is shifted in when this state is left by setting TMS high. While the instruction is shifted in from the TDI pin, the captured IR-state 0x01 is shifted out on the TDO pin. The JTAG Instruction selects a particular Data Register as path between TDI and TDO and controls the circuitry surrounding the selected Data Register.

- Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. The instruction is latched onto the parallel output from the Shift Register path in the Update-IR state. The Exit-IR, Pause-IR, and Exit2-IR states are only used for navigating the state machine.
- At the TMS input, apply the sequence 1, 0, 0 at the rising edges of TCK to enter the Shift Data Register – Shift-DR state. While in this state, upload the selected data register (selected by the present JTAG instruction in the JTAG Instruction Register) from the TDI input at the rising edge of TCK. In order to remain in the Shift-DR state, the TMS input must be held low during input of all bits except the MSB. The MSB of the data is shifted in when this state is left by setting TMS high. While the data register is shifted in from the TDI pin, the parallel inputs to the data register captured in the Capture-DR state is shifted out on the TDO pin.
- Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. If the selected data register has a latched parallel-output, the latching takes place in the Update-DR state. The Exit-DR, Pause-DR, and Exit2-DR states are only used for navigating the state machine.

As shown in the state diagram, the Run-Test/Idle state need not be entered between selecting JTAG instruction and using data registers, and some JTAG instructions may select certain functions to be performed in the Run-Test/Idle, making it unsuitable as an Idle state.

Note: Independent of the initial state of the TAP Controller, the Test-Logic-Reset state can always be entered by holding TMS high for five TCK clock periods.

For detailed information on the JTAG specification, refer to the literature listed in “Bibliography” on page 289.

## Using the Boundary-scan Chain

A complete description of the Boundary-scan capabilities are given in the section “Boundary-scan IEEE 1149.1 (JTAG)” on page 290.

## Using the On-chip Debug System

As shown in Figure 141, the hardware support for On-chip Debugging consists mainly of

- A scan chain on the interface between the internal AVR CPU and the internal peripheral units.
- Break Point unit.
- Communication interface between the CPU and JTAG system.

All read or modify/write operations needed for implementing the Debugger are done by applying AVR instructions via the internal AVR CPU Scan Chain. The CPU sends the result to an I/O memory mapped location which is part of the communication interface between the CPU and the JTAG system.

The Break Point Unit implements Break on Change of Program Flow, Single Step Break, two Program Memory Break Points, and two combined Break Points. Together, the four Break Points can be configured as either:

- 4 single Program Memory Break Points.
- 3 single Program Memory Break Points + 1 single Data Memory Break Point.
- 2 single Program Memory Break Points + 2 single Data Memory Break Points.
- 2 single Program Memory Break Points + 1 Program Memory Break Point with mask (“range Break Point”).

- 2 single Program Memory Break Points + 1 Data Memory Break Point with mask (“range Break Point”).

A debugger, like the AVR Studio, may however use one or more of these resources for its internal purpose, leaving less flexibility to the end-user.

A list of the On-chip Debug specific JTAG instructions is given in “On-chip Debug Specific JTAG Instructions” on page 288.

The JTAGEN Fuse must be programmed to enable the JTAG Test Access Port. In addition, the OCDEN Fuse must be programmed and no Lock bits must be set for the On-chip debug system to work. As a security feature, the On-chip debug system is disabled when either of the LB1 or LB2 Lock bits are set. Otherwise, the On-chip debug system would have provided a back-door into a secured device.

The AVR Studio enables the user to fully control execution of programs on an AVR device with On-chip Debug capability, AVR In-Circuit Emulator, or the built-in AVR Instruction Set Simulator. AVR Studio® supports source level execution of Assembly programs assembled with Atmel Corporation’s AVR Assembler and C programs compiled with third party vendors’ compilers.

AVR Studio runs under Microsoft® Windows® 95/98/2000/NT/XP.

For a full description of the AVR Studio, please refer to the AVR Studio User Guide. Only highlights are presented in this document.

All necessary execution commands are available in AVR Studio, both on source level and on disassembly level. The user can execute the program, single step through the code either by tracing into or stepping over functions, step out of functions, place the cursor on a statement and execute until the statement is reached, stop the execution, and reset the execution target. In addition, the user can have an unlimited number of code Break Points (using the BREAK instruction) and up to two data memory Break Points, alternatively combined as a mask (range) Break Point.

## On-chip Debug Specific JTAG Instructions

The On-chip debug support is considered being private JTAG instructions, and distributed within ATMEL and to selected third party vendors only. Instruction opcodes are listed for reference.

**PRIVATE0 (0x8)** Private JTAG instruction for accessing On-chip debug system.

**PRIVATE1 (0x9)** Private JTAG instruction for accessing On-chip debug system.

**PRIVATE2 (0xA)** Private JTAG instruction for accessing On-chip debug system.

**PRIVATE3 (0xB)** Private JTAG instruction for accessing On-chip debug system.

## On-chip Debug Related Register in I/O Memory

**On-chip Debug Register – OCDR**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | IDRD/OCDR7 | OCDR6 | OCDR5 | OCDR4 | OCDR3 | OCDR2 | OCDR1 | OCDR0 | OCDR |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

The OCDR Register provides a communication channel from the running program in the microcontroller to the debugger. The CPU can transfer a byte to the debugger by writing to this location. At the same time, an internal flag; I/O Debug Register Dirty – IDRD – is set to indicate to the debugger that the register has been written. When the CPU reads the OCDR Register the 7 LSB will be from the OCDR Register, while the MSB is the IDRD bit. The debugger clears the IDRD bit when it has read the information.

In some AVR devices, this register is shared with a standard I/O location. In this case, the OCDR Register can only be accessed if the OCDEN Fuse is programmed, and the debugger enables access to the OCDR Register. In all other cases, the standard I/O location is accessed.

Refer to the debugger documentation for further information on how to use this register.

## Using the JTAG Programming Capabilities

Programming of AVR parts via JTAG is performed via the 4-pin JTAG port, TCK, TMS, TDI, and TDO. These are the only pins that need to be controlled/observed to perform JTAG programming (in addition to power pins). It is not required to apply 12V externally. The JTAGEN Fuse must be programmed and the JTD bit in the MCUCR Register must be cleared to enable the JTAG Test Access Port.

The JTAG programming capability supports:

- Flash programming and verifying.
- EEPROM programming and verifying.
- Fuse programming and verifying.
- Lock bit programming and verifying.

The Lock bit security is exactly as in parallel programming mode. If the Lock bits LB1 or LB2 are programmed, the OCDEN Fuse cannot be programmed unless first doing a chip erase. This is a security feature that ensures no back-door exists for reading out the content of a secured device.

The details on programming through the JTAG interface and programming specific JTAG instructions are given in the section “JTAG Programming Overview” on page 342.

## Bibliography

For more information about general Boundary-scan, the following literature can be consulted:


- IEEE: IEEE Std 1149.1-1990. IEEE Standard Test Access Port and Boundary-scan Architecture, IEEE, 1993.
- Colin Maunder: The Board Designers Guide to Testable Logic Circuits, Addison-Wesley, 1992.

# Boundary-scan IEEE 1149.1 (JTAG)

## Features

- **JTAG (IEEE std. 1149.1 compliant) Interface**
- **Boundary-scan Capabilities According to the JTAG Standard**
- **Full Scan of all Port Functions as well as Analog Circuitry having Off-chip Connections**
- **Supports the Optional IDCODE Instruction**
- **Additional Public AVR_RESET Instruction to Reset the AVR**

## System Overview

The Boundary-scan chain has the capability of driving and observing the logic levels on the digital I/O pins, as well as the boundary between digital and analog logic for analog circuitry having off-chip connections. At system level, all ICs having JTAG capabilities are connected serially by the TDI/TDO signals to form a long Shift Register. An external controller sets up the devices to drive values at their output pins, and observe the input values received from other devices. The controller compares the received data with the expected result. In this way, Boundary-scan provides a mechanism for testing interconnections and integrity of components on Printed Circuits Boards by using the four TAP signals only.

The four IEEE 1149.1 defined mandatory JTAG instructions IDCODE, BYPASS, SAMPLE/PRELOAD, and EXTEST, as well as the AVR specific public JTAG instruction AVR_RESET can be used for testing the Printed Circuit Board. Initial scanning of the data register path will show the ID-Code of the device, since IDCODE is the default JTAG instruction. It may be desirable to have the AVR device in reset during test mode. If not reset, inputs to the device may be determined by the scan operations, and the internal software may be in an undetermined state when exiting the test mode. Entering reset, the outputs of any port pin will instantly enter the high impedance state, making the HIGHZ instruction redundant. If needed, the BYPASS instruction can be issued to make the shortest possible scan chain through the device. The device can be set in the reset state either by pulling the external $\overline{\text{RESET}}$ pin low, or issuing the AVR_RESET instruction with appropriate setting of the Reset Data Register.

The EXTEST instruction is used for sampling external pins and loading output pins with data. The data from the output latch will be driven out on the pins as soon as the EXTEST instruction is loaded into the JTAG IR-Register. Therefore, the SAMPLE/PRELOAD should also be used for setting initial values to the scan ring, to avoid damaging the board when issuing the EXTEST instruction for the first time. SAMPLE/PRELOAD can also be used for taking a snapshot of the external pins during normal operation of the part.

The JTAGEN Fuse must be programmed and the JTD bit in the I/O Register MCUCR must be cleared to enable the JTAG Test Access Port.

When using the JTAG interface for Boundary-scan, using a JTAG TCK clock frequency higher than the internal chip frequency is possible. The chip clock is not required to run.

## Data Registers

The data registers relevant for Boundary-scan operations are:

- Bypass Register
- Device Identification Register
- Reset Register
- Boundary-scan Chain

### Bypass Register

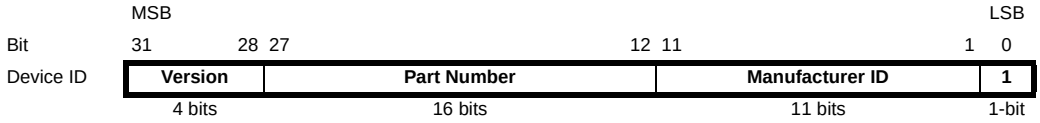
The Bypass Register consists of a single Shift Register stage. When the Bypass Register is selected as path between TDI and TDO, the register is reset to 0 when leaving the

Capture-DR controller state. The Bypass Register may be used to shorten the scan chain on a system when the other devices are to be tested.

**Device Identification Register**

Figure 143 shows the structure of the Device Identification Register.

**Figure 143.** The Format of the Device Identification Register




*Version*

Version is a 4-bit number identifying the revision of the component. The relevant version number is shown in Table 105.

**Table 105.** JTAG Version Numbers

| Version | JTAG Version Number (Hex) |
|---|---|
| AT90CAN128 revision A | 0x0 |
| | |

*Part Number*

The part number is a 16-bit code identifying the component. The JTAG Part Number for AT90CAN128 is listed in Table 106.

**Table 106.** AVR JTAG Part Number

| Part Number | JTAG Part Number (Hex) |
|---|---|
| AT90CAN128 | 0x9781 |

*Manufacturer ID*

The Manufacturer ID is a 11-bit code identifying the manufacturer. The JTAG manufacturer ID for ATMEL is listed in Table 107.

**Table 107.** Manufacturer ID

| Manufacturer | JTAG Manufactor ID (Hex) |
|---|---|
| ATMEL | 0x01F |

*Device ID*

The full Device ID is listed in Table 108 following the AT90CAN128 version.

**Table 108.** Device ID

| Version | JTAG Device ID (Hex) |
|---|---|
| AT90CAN128 revision A | 0x0978103F |
| | |

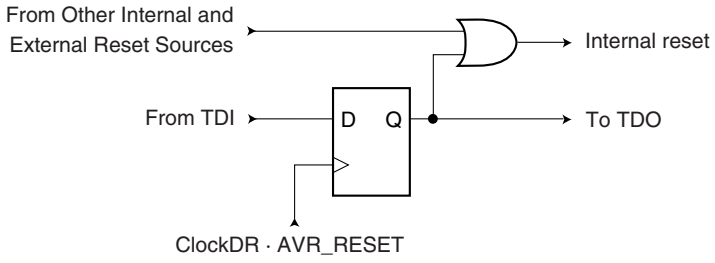
**Reset Register**

The Reset Register is a test data register used to reset the part. Since the AVR tri-states Port Pins when reset, the Reset Register can also replace the function of the unimplemented optional JTAG instruction HIGHZ.

A high value in the Reset Register corresponds to pulling the external Reset low. The part is reset as long as there is a high value present in the Reset Register. Depending on the fuse settings for the clock options, the part will remain reset for a reset time-out period (refer to “System Clock” on page 34) after releasing the Reset Register. The out-

put from this data register is not latched, so the reset will take place immediately, as shown in Figure 144.

**Figure 144.** Reset Register




**Boundary-scan Chain**

The Boundary-scan Chain has the capability of driving and observing the logic levels on the digital I/O pins, as well as the boundary between digital and analog logic for analog circuitry having off-chip connections.

See “Boundary-scan Chain” on page 294 for a complete description.

## Boundary-scan Specific JTAG Instructions

The instruction register is 4-bit wide, supporting up to 16 instructions. Listed below are the JTAG instructions useful for Boundary-scan operation. Note that the optional HIGHZ instruction is not implemented, but all outputs with tri-state capability can be set in high-impedant state by using the AVR_RESET instruction, since the initial state for all port pins is tri-state.

As a definition in this datasheet, the LSB is shifted in and out first for all Shift Registers.

The OPCODE for each instruction is shown behind the instruction name in hex format. The text describes which data register is selected as path between TDI and TDO for each instruction.

**EXTEST (0x0)**

Mandatory JTAG instruction for selecting the Boundary-scan Chain as data register for testing circuitry external to the AVR package. For port-pins, Pull-up Disable, Output Control, Output Data, and Input Data are all accessible in the scan chain. For Analog circuits having off-chip connections, the interface between the analog and the digital logic is in the scan chain. The contents of the latched outputs of the Boundary-scan chain is driven out as soon as the JTAG IR-Register is loaded with the EXTEST instruction.

The active states are:

- **Capture-DR**: Data on the external pins are sampled into the Boundary-scan Chain.
- **Shift-DR**: The Internal Scan Chain is shifted by the TCK input.
- **Update-DR**: Data from the scan chain is applied to output pins.

**IDCODE (0x1)**

Optional JTAG instruction selecting the 32 bit ID-Register as data register. The ID-Register consists of a version number, a device number and the manufacturer code chosen by JEDEC. This is the default instruction after power-up.

The active states are:

- **Capture-DR**: Data in the IDCODE Register is sampled into the Boundary-scan Chain.
- **Shift-DR**: The IDCODE scan chain is shifted by the TCK input.

**SAMPLE_PRELOAD (0x2)**

Mandatory JTAG instruction for pre-loading the output latches and taking a snap-shot of the input/output pins without affecting the system operation. However, the output latches are not connected to the pins. The Boundary-scan Chain is selected as data register.

The active states are:

- **Capture-DR**: Data on the external pins are sampled into the Boundary-scan Chain.
- **Shift-DR**: The Boundary-scan Chain is shifted by the TCK input.
- **Update-DR**: Data from the Boundary-scan chain is applied to the output latches. However, the output latches are not connected to the pins.

**AVR_RESET (0xC)**

The AVR specific public JTAG instruction for forcing the AVR device into the Reset mode or releasing the JTAG reset source. The TAP controller is not reset by this instruction. The one bit Reset Register is selected as data register.

Note that the reset will be active as long as there is a logic “one” in the Reset Chain.

The output from this chain is not latched.

The active states are:

- **Shift-DR**: The Reset Register is shifted by the TCK input.

**BYPASS (0xF)**

Mandatory JTAG instruction selecting the Bypass Register for data register.

The active states are:

- **Capture-DR**: Loads a logic “0” into the Bypass Register.
- **Shift-DR**: The Bypass Register cell between TDI and TDO is shifted.

## Boundary-scan Related Register in I/O Memory

**MCU Control Register – MCUCR**

The MCU Control Register contains control bits for general MCU functions.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | **JTD** | – | – | **PUD** | – | – | **IVSEL** | **IVCE** | **MCUCR** |
| Read/Write | R/W | R | R | R/W | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bits 7 – JTD: JTAG Interface Disable**

When this bit is zero, the JTAG interface is enabled if the JTAGEN Fuse is programmed. If this bit is one, the JTAG interface is disabled. In order to avoid unintentional disabling or enabling of the JTAG interface, a timed sequence must be followed when changing this bit: The application software must write this bit to the desired value twice within four cycles to change its value. Note that this bit must not be altered when using the On-chip Debug system.

If the JTAG interface is left unconnected to other JTAG circuitry, the JTD bit should be set to one. The reason for this is to avoid static current at the TDO pin in the JTAG interface.

### MCU Status Register – MCUSR

The MCU Status Register provides information on which reset source caused an MCU reset.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | JTRF | WDRF | BORF | EXTRF | PORF | MCUSR |
| Read/Write | R | R | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | See Bit Description | | | | | |

- **Bit 4 – JTRF: JTAG Reset Flag**

This bit is set if a reset is being caused by a logic one in the JTAG Reset Register selected by the JTAG instruction AVR_RESET. This bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

## Boundary-scan Chain

The Boundary-scan chain has the capability of driving and observing the logic levels on the digital I/O pins, as well as the boundary between digital and analog logic for analog circuitry having off-chip connection.

### Scanning the Digital Port Pins

Figure 145 shows the Boundary-scan Cell for a bi-directional port pin with pull-up function. The cell consists of a standard Boundary-scan cell for the Pull-up Enable – PUExn – function, and a bi-directional pin cell that combines the three signals Output Control – OCxn, Output Data – ODxn, and Input Data – IDxn, into only a two-stage Shift Register. The port and pin indexes are not used in the following description

The Boundary-scan logic is not included in the figures in the datasheet. Figure 146 shows a simple digital port pin as described in the section “I/O-Ports” on page 61. The Boundary-scan details from Figure 145 replaces the dashed box in Figure 146.

When no alternate port function is present, the Input Data – ID – corresponds to the PINxn Register value (but ID has no synchronizer), Output Data corresponds to the PORT Register, Output Control corresponds to the Data Direction – $\overline{\text{DD}}$ Register, and the Pull-up Enable – PUExn – corresponds to logic expression $\overline{\text{PUD}} \cdot \overline{\text{DDxn}} \cdot \text{PORTxn}$.

Digital alternate port functions are connected outside the dotted box in Figure 146 to make the scan chain read the actual pin value. For Analog function, there is a direct connection from the external pin to the analog circuit, and a scan chain is inserted on the interface between the digital logic and the analog circuitry.

**Figure 145.** Boundary-scan Cell for Bi-directional Port Pin with Pull-up Function.

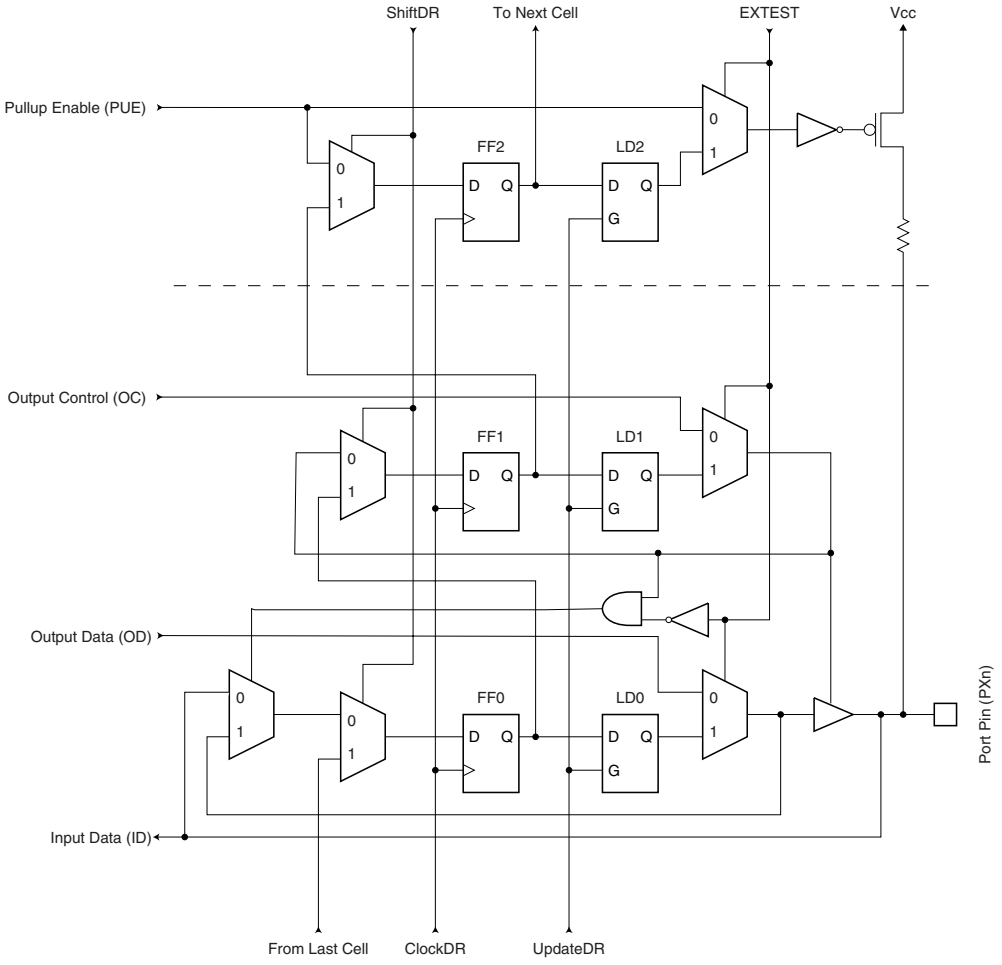
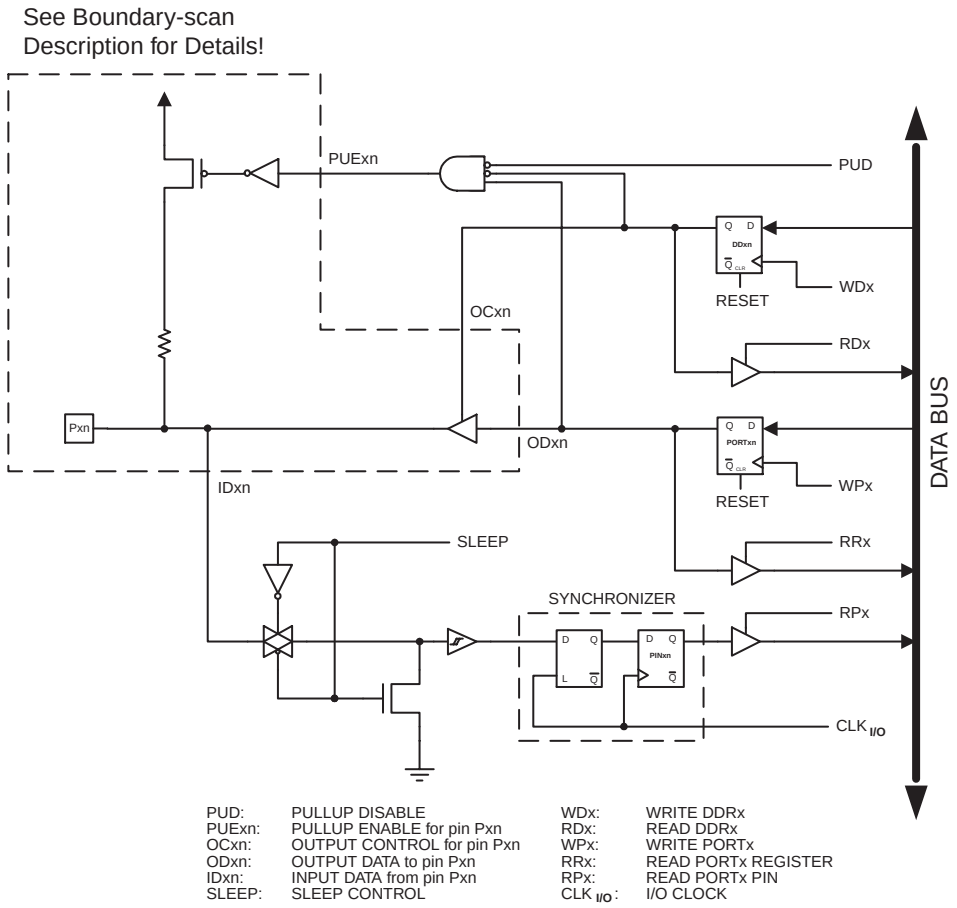


**Figure 146.** General Port Pin Schematic Diagram

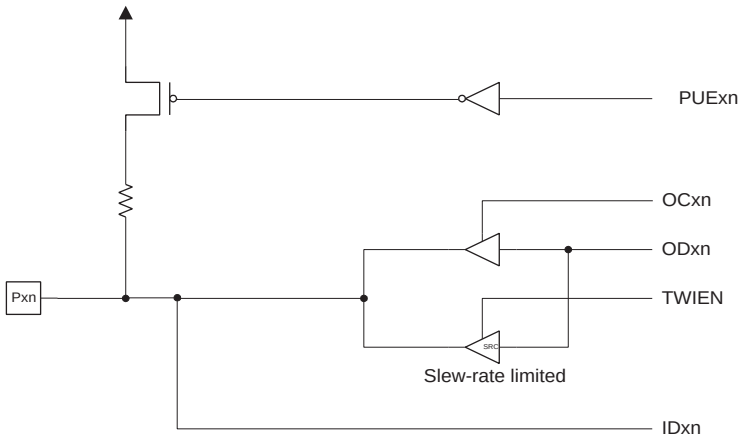



**Boundary-scan and the Two-wire Interface**

The two Two-wire Interface pins SCL and SDA have one additional control signal in the scan-chain; Two-wire Interface Enable – TWIEN. As shown in Figure 147, the TWIEN signal enables a tri-state buffer with slew-rate control in parallel with the ordinary digital port pins. A general scan cell as shown in Figure 151 is attached to the TWIEN signal.

Notes:
1. A separate scan chain for the 50 ns spike filter on the input is not provided. The ordinary scan support for digital port pins suffice for connectivity tests. The only reason for having TWIEN in the scan path, is to be able to disconnect the slew-rate control buffer when doing boundary-scan.
2. Make sure the OC and TWIEN signals are not asserted simultaneously, as this will lead to drive contention.

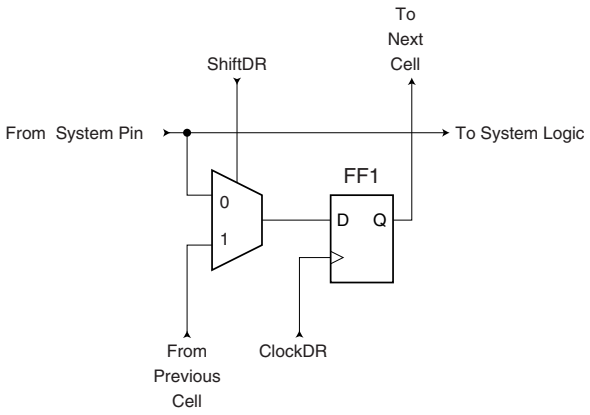
**Figure 147.** Additional Scan Signal for the Two-wire Interface




**Scanning the RESET Pin**

The RESET pin accepts 3V or 5V active low logic for standard reset operation, and 12V active high logic for High Voltage Parallel programming. An observe-only cell as shown in Figure 148 is inserted both for the 3V or 5V reset signal - RSTT, and the 12V reset signal - RSTHV.

**Figure 148.** Observe-only Cell for RESET pin




**Scanning the Clock Pins**

The AVR devices have many clock options selectable by fuses. These are: Internal RC Oscillator, External Clock, (High Frequency) Crystal Oscillator, Low-frequency Crystal Oscillator, and Ceramic Resonator.

Figure 149 shows how each oscillator with external connection is supported in the scan chain. The Enable signal is supported with a general Boundary-scan cell, while the Oscillator/clock output is attached to an observe-only cell. In addition to the main clock, the Timer2 Oscillator is scanned in the same way. The output from the internal RC Oscillator is not scanned, as this oscillator does not have external connections.

**Figure 149.** Boundary-scan Cells for Oscillators and Clock Options

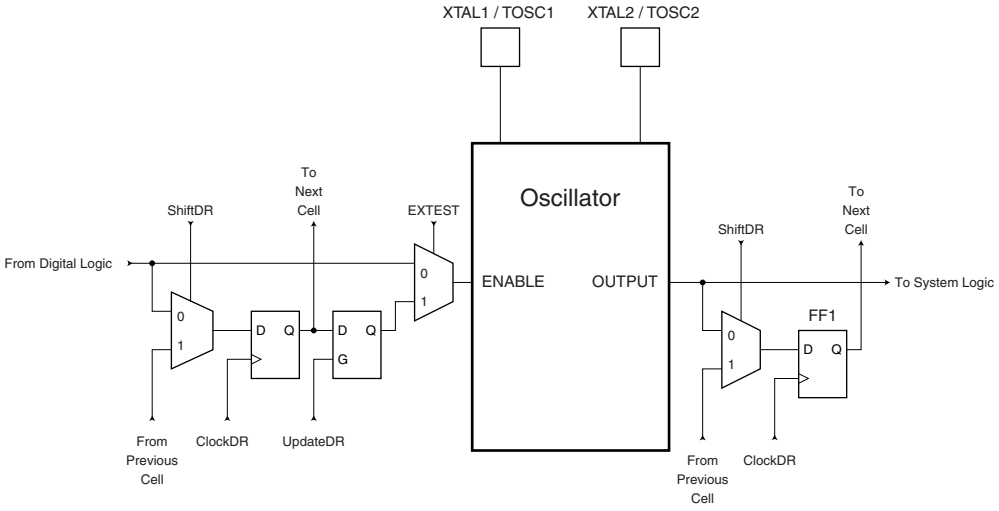



Table 109 summaries the scan registers for the external clock pin XTAL1, oscillators with XTAL1/XTAL2 connections as well as external Timer2 clock pin TOSC1 and 32kHz Timer2 Oscillator.

**Table 109.** Scan Signals for the Oscillators(1)(2)(3)

| Enable Signal | Scanned Clock Line | Clock Option | Scanned Clock Line when not Used |
|---|---|---|---|
| EXTCLKEN | EXTCLK (XTAL1) | External Main Clock | 0 |
| OSCON | OSCCK | External Crystal<br>External Ceramic Resonator | 1 |
| OSC32EN | OSC32CK | Low Freq. External Crystal | 1 |
| TOSKON | TOSCK | 32 kHz Timer2 Oscillator | 1 |

Notes:
1. Do not enable more than one clock source as clock at a time.
2. Scanning an Oscillator output gives unpredictable results as there is a frequency drift between the internal Oscillator and the JTAG TCK clock. If possible, scanning an external clock is preferred.
3. The main clock configuration is programmed by fuses. As a fuse is not changed run-time, the main clock configuration is considered fixed for a given application. The user is advised to scan the same clock option as to be used in the final system. The enable signals are supported in the scan chain because the system logic can disable clock options in sleep modes, thereby disconnecting the Oscillator pins from the scan path if not provided.

### Scanning the Analog Comparator

The relevant Comparator signals regarding Boundary-scan are shown in Figure 150. The Boundary-scan cell from Figure 151 is attached to each of these signals. The signals are described in Table 110.

The Comparator need not be used for pure connectivity testing, since all analog inputs are shared with a digital port pin as well.

**Figure 150.** Analog Comparator

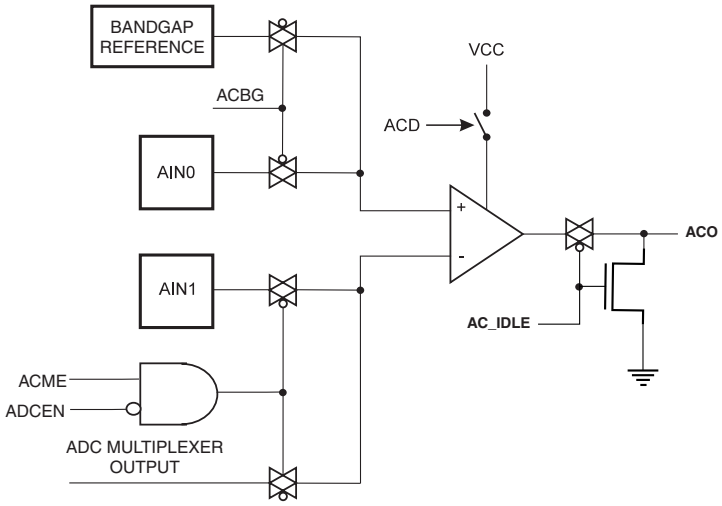



**Figure 151.** General Boundary-scan cell Used for Signals for Comparator and ADC

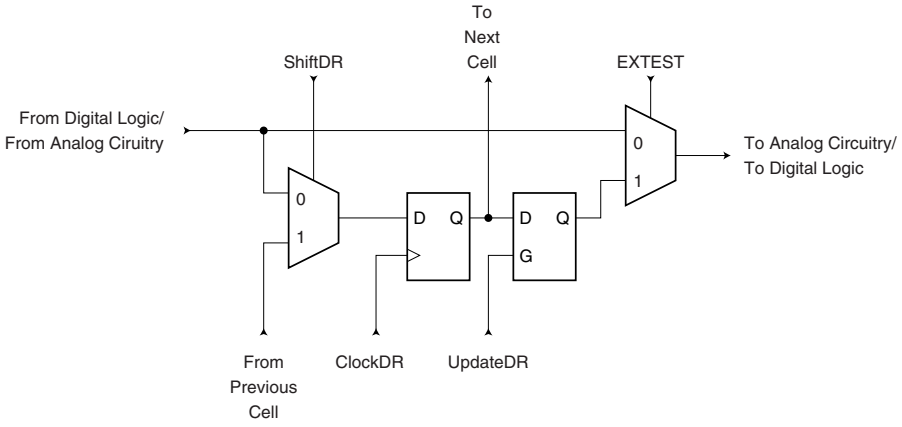


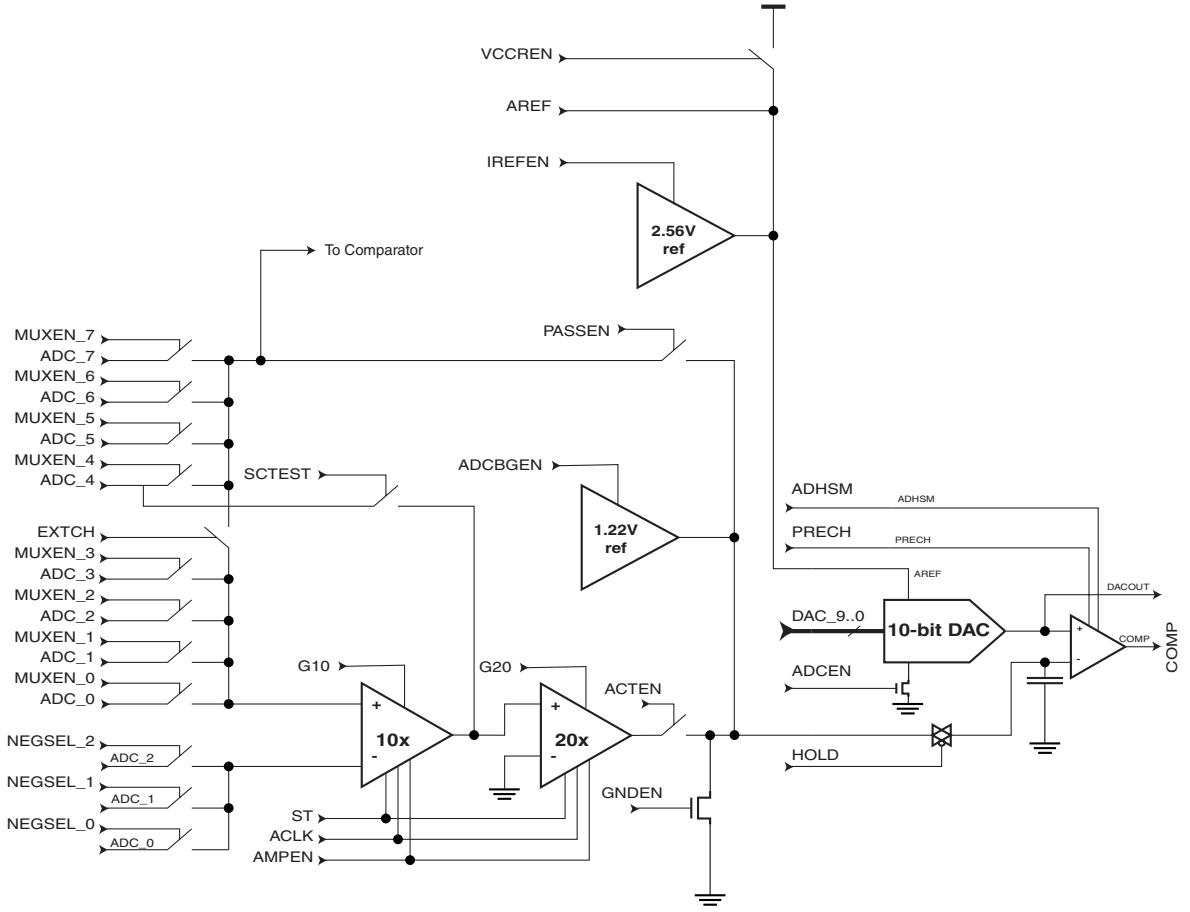

**Table 110.** Boundary-scan Signals for the Analog Comparator

| Signal Name | Direction as Seen from the Comparator | Description | Recommended Input when Not in Use | Output Values when Recommended Inputs are Used |
|---|---|---|---|---|
| AC_IDLE | input | Turns off Analog Comparator when true | 1 | Depends upon µC code being executed |
| ACO | output | Analog Comparator Output | Will become input to µC code being executed | 0 |
| ACME | input | Uses output signal from ADC mux when true | 0 | Depends upon µC code being executed |
| ACBG | input | Bandgap Reference enable | 0 | Depends upon µC code being executed |

**Scanning the ADC**

Figure 152 shows a block diagram of the ADC with all relevant control and observe signals. The Boundary-scan cell from Figure 151 is attached to each of these signals. The ADC need not be used for pure connectivity testing, since all analog inputs are shared with a digital port pin as well.

**Figure 152.** Analog to Digital Converter




The signals are described briefly in Table 111.

**Table 111.** Boundary-scan Signals for the ADC(1)

| Signal Name | Direction as Seen from the ADC | Description | Recommended Input when not in Use | Output Values when Recommended Inputs are Used, and CPU is not Using the ADC |
|---|---|---|---|---|
| COMP | Output | Comparator Output | 0 | 0 |
| ACLK | Input | Clock signal to gain stages implemented as Switch-cap filters | 0 | 0 |
| ACTEN | Input | Enable path from gain stages to the comparator | 0 | 0 |

**Table 111.** Boundary-scan Signals for the ADC[1] (Continued)

| Signal Name | Direction as Seen from the ADC | Description | Recommended Input when not in Use | Output Values when Recommended Inputs are Used, and CPU is not Using the ADC |
|---|---|---|---|---|
| ADHSM | Input | Increases speed of comparator at the sacrifice of higher power consumption | 0 | 0 |
| ADCBGEN | Input | Enable Band-gap reference as negative input to comparator | 0 | 0 |
| ADCEN | Input | Power-on signal to the ADC | 0 | 0 |
| AMPEN | Input | Power-on signal to the gain stages | 0 | 0 |
| DAC_9 | Input | Bit 9 of digital value to DAC | 1 | 1 |
| DAC_8 | Input | Bit 8 of digital value to DAC | 0 | 0 |
| DAC_7 | Input | Bit 7 of digital value to DAC | 0 | 0 |
| DAC_6 | Input | Bit 6 of digital value to DAC | 0 | 0 |
| DAC_5 | Input | Bit 5 of digital value to DAC | 0 | 0 |
| DAC_4 | Input | Bit 4 of digital value to DAC | 0 | 0 |
| DAC_3 | Input | Bit 3 of digital value to DAC | 0 | 0 |
| DAC_2 | Input | Bit 2 of digital value to DAC | 0 | 0 |
| DAC_1 | Input | Bit 1 of digital value to DAC | 0 | 0 |
| DAC_0 | Input | Bit 0 of digital value to DAC | 0 | 0 |
| EXTCH | Input | Connect ADC channels 0 - 3 to by-pass path around gain stages | 1 | 1 |
| G10 | Input | Enable 10x gain | 0 | 0 |
| G20 | Input | Enable 20x gain | 0 | 0 |
| GNDEN | Input | Ground the negative input to comparator when true | 0 | 0 |

**Table 111.** Boundary-scan Signals for the ADC[1] (Continued)

| Signal Name | Direction as Seen from the ADC | Description | Recommended Input when not in Use | Output Values when Recommended Inputs are Used, and CPU is not Using the ADC |
|---|---|---|---|---|
| HOLD | Input | Sample & Hold signal. Sample analog signal when low. Hold signal when high. If gain stages are used, this signal must go active when ACLK is high. | 1 | 1 |
| IREFEN | Input | Enables Band-gap reference as AREF signal to DAC | 0 | 0 |
| MUXEN_7 | Input | Input Mux bit 7 | 0 | 0 |
| MUXEN_6 | Input | Input Mux bit 6 | 0 | 0 |
| MUXEN_5 | Input | Input Mux bit 5 | 0 | 0 |
| MUXEN_4 | Input | Input Mux bit 4 | 0 | 0 |
| MUXEN_3 | Input | Input Mux bit 3 | 0 | 0 |
| MUXEN_2 | Input | Input Mux bit 2 | 0 | 0 |
| MUXEN_1 | Input | Input Mux bit 1 | 0 | 0 |
| MUXEN_0 | Input | Input Mux bit 0 | 1 | 1 |
| NEGSEL_2 | Input | Input Mux for negative input for differential signal, bit 2 | 0 | 0 |
| NEGSEL_1 | Input | Input Mux for negative input for differential signal, bit 1 | 0 | 0 |
| NEGSEL_0 | Input | Input Mux for negative input for differential signal, bit 0 | 0 | 0 |
| PASSEN | Input | Enable pass-gate of gain stages. | 1 | 1 |
| PRECH | Input | Precharge output latch of comparator. (Active low) | 1 | 1 |

**Table 111.** Boundary-scan Signals for the ADC[1] (Continued)

| Signal Name | Direction as Seen from the ADC | Description | Recommended Input when not in Use | Output Values when Recommended Inputs are Used, and CPU is not Using the ADC |
|---|---|---|---|---|
| SCTEST | Input | Switch-cap TEST enable. Output from x10 gain stage send out to Port Pin having ADC_4 | 0 | 0 |
| ST | Input | Output of gain stages will settle faster if this signal is high first two ACLK periods after AMPEN goes high. | 0 | 0 |
| VCCREN | Input | Selects Vcc as the ACC reference voltage. | 0 | 0 |

Note: 1. Incorrect setting of the switches in Figure 152 will make signal contention and may damage the part. There are several input choices to the S&H circuitry on the negative input of the output comparator in Figure 152. Make sure only one path is selected from either one ADC pin, Bandgap reference source, or Ground.

If the ADC is not to be used during scan, the recommended input values from Table 111 should be used. The user is recommended **not** to use the Differential Gain stages during scan. Switch-Cap based gain stages require fast operation and accurate timing which is difficult to obtain when used in a scan chain. Details concerning operations of the differential gain stage is therefore not provided. For the same reason, the ADC High Speed mode (ADHSM) bit does not make any sense during boundary-scan operation.

The AVR ADC is based on the analog circuitry shown in Figure 152 with a successive approximation algorithm implemented in the digital logic. When used in Boundary-scan, the problem is usually to ensure that an applied analog voltage is measured within some limits. This can easily be done without running a successive approximation algorithm: apply the lower limit on the digital DAC[9:0] lines, make sure the output from the comparator is low, then apply the upper limit on the digital DAC[9:0] lines, and verify the output from the comparator to be high.

The ADC need not be used for pure connectivity testing, since all analog inputs are shared with a digital port pin as well.

When using the ADC, remember the following

- The port pin for the ADC channel in use must be configured to be an input with pull-up disabled to avoid signal contention.
- In Normal mode, a dummy conversion (consisting of 10 comparisons) is performed when enabling the ADC. The user is advised to wait at least 200ns after enabling the ADC before controlling/observing any ADC signal, or perform a dummy conversion before using the first result.
- The DAC values must be stable at the midpoint value 0x200 when having the HOLD signal low (Sample mode).

As an example, consider the task of verifying a 1.5V ± 5% input signal at ADC channel 3 when the power supply is 5.0V and AREF is externally connected to $V_{CC}$.

The lower limit is: $\lceil 1024 \cdot 1{,}5V \cdot 0{,}95/5V \rceil = 291 = 0x123$
The upper limit is: $\lceil 1024 \cdot 1{,}5V \cdot 1{,}05/5V \rceil = 323 = 0x143$

The recommended values from Table 111 are used unless other values are given in the algorithm in Table 112. Only the DAC and port pin values of the Scan Chain are shown. The column “Actions” describes what JTAG instruction to be used before filling the Boundary-scan Register with the succeeding columns. The verification should be done on the data scanned out when scanning in the data on the same row in the table.

**Table 112.** Algorithm for Using the ADC

| Step | Actions | ADCEN | DAC | MUXEN | HOLD | PRECH | PA3. Data | PA3. Control | PA3. Pullup_ Enable |
|---|---|---|---|---|---|---|---|---|---|
| 1 | SAMPLE_ PRELOAD | 1 | 0x200 | 0x08 | 1 | 1 | 0 | 0 | 0 |
| 2 | EXTEST | 1 | 0x200 | 0x08 | 0 | 1 | 0 | 0 | 0 |
| 3 | | 1 | 0x200 | 0x08 | 1 | 1 | 0 | 0 | 0 |
| 4 | | 1 | 0x123 | 0x08 | 1 | 1 | 0 | 0 | 0 |
| 5 | | 1 | 0x123 | 0x08 | 1 | 0 | 0 | 0 | 0 |
| 6 | Verify the COMP bit scanned out to be 0 | 1 | 0x200 | 0x08 | 1 | 1 | 0 | 0 | 0 |
| 7 | | 1 | 0x200 | 0x08 | 0 | 1 | 0 | 0 | 0 |
| 8 | | 1 | 0x200 | 0x08 | 1 | 1 | 0 | 0 | 0 |
| 9 | | 1 | 0x143 | 0x08 | 1 | 1 | 0 | 0 | 0 |
| 10 | | 1 | 0x143 | 0x08 | 1 | 0 | 0 | 0 | 0 |
| 11 | Verify the COMP bit scanned out to be 1 | 1 | 0x200 | 0x08 | 1 | 1 | 0 | 0 | 0 |

Using this algorithm, the timing constraint on the HOLD signal constrains the TCK clock frequency. As the algorithm keeps HOLD high for five steps, the TCK clock frequency has to be at least five times the number of scan bits divided by the maximum hold time, $t_{hold,max}$

## AT90CAN128 Boundary-scan Order

Table 113 shows the Scan order between TDI and TDO when the Boundary-scan chain is selected as data path. Bit 0 is the LSB; the first bit scanned in, and the first bit scanned out. The scan order follows the pin-out order as far as possible. Therefore, the bits of Port A is scanned in the opposite bit order of the other ports. Exceptions from the rules are the Scan chains for the analog circuits, which constitute the most significant bits of the scan chain regardless of which physical pin they are connected to. In Figure 145, PXn. Data corresponds to FF0, PXn. Control corresponds to FF1, and PXn. Pullup_enable corresponds to FF2. Bit 2, 3, 4, and 5 of Port C is not in the scan chain, since these pins constitute the TAP pins when the JTAG is enabled.

**Table 113.** AT90CAN128 Boundary-scan Order

| Bit Number | Signal Name | Comment | Module |
|---|---|---|---|
| 200 | AC_IDLE | | Comparator |
| 199 | ACO | | |
| 198 | ACME | | |
| 197 | AINBG | | |
| 196 | COMP | | ADC |
| 195 | ACLK | | |
| 194 | ACTEN | | |
| 193 | ADHSM | | |
| 192 | ADCBGEN | | |
| 191 | ADCEN | | |
| 190 | AMPEN | | |
| 189 | DAC_9 | | |
| 188 | DAC_8 | | |
| 187 | DAC_7 | | |
| 186 | DAC_6 | | |
| 185 | DAC_5 | | |
| 184 | DAC_4 | | |
| 183 | DAC_3 | | |
| 182 | DAC_2 | | |
| 181 | DAC_1 | | |
| 180 | DAC_0 | | |
| 179 | EXTCH | | |
| 178 | G10 | | |
| 177 | G20 | | |
| 176 | GNDEN | | |
| 175 | HOLD | | |
| 174 | IREFEN | | |
| 173 | MUXEN_7 | | |
| 172 | MUXEN_6 | | |
| 171 | MUXEN_5 | | |
| 170 | MUXEN_4 | | |

**Table 113.** AT90CAN128 Boundary-scan Order (Continued)

| Bit Number | Signal Name | Comment | Module |
|---|---|---|---|
| 169 | MUXEN_3 | | ADC |
| 168 | MUXEN_2 | | |
| 167 | MUXEN_1 | | |
| 166 | MUXEN_0 | | |
| 165 | NEGSEL_2 | | |
| 164 | NEGSEL_1 | | |
| 163 | NEGSEL_0 | | |
| 162 | PASSEN | | |
| 161 | PRECH | | |
| 160 | SCTEST | | |
| 159 | ST | | |
| 158 | VCCREN | | |
| 157 | PE0.Data | | Port E |
| 156 | PE0.Control | | |
| 155 | PE0.Pullup_Enable | | |
| 154 | PE1.Data | | |
| 153 | PE1.Control | | |
| 152 | PE1.Pullup_Enable | | |
| 151 | PE2.Data | | |
| 150 | PE2.Control | | |
| 149 | PE2.Pullup_Enable | | |
| 148 | PE3.Data | | |
| 147 | PE3.Control | | |
| 146 | PE3.Pullup_Enable | | |
| 145 | PE4.Data | | |
| 144 | PE4.Control | | |
| 143 | PE4.Pullup_Enable | | |
| 142 | PE5.Data | | |
| 141 | PE5.Control | | |
| 140 | PE5.Pullup_Enable | | |
| 139 | PE6.Data | | |
| 138 | PE6.Control | | |
| 137 | PE6.Pullup_Enable | | |
| 136 | PE7.Data | | |
| 135 | PE7.Control | | |
| 134 | PE7.Pullup_Enable | | |
| 133 | PB0.Data | | Port B |
| 132 | PB0.Control | | |

**Table 113.** AT90CAN128 Boundary-scan Order (Continued)

| Bit Number | Signal Name | Comment | Module |
|---|---|---|---|
| 131 | PB0.Pullup_Enable | | Port B |
| 130 | PB1.Data | | |
| 129 | PB1.Control | | |
| 128 | PB1.Pullup_Enable | | |
| 127 | PB2.Data | | |
| 126 | PB2.Control | | |
| 125 | PB2.Pullup_Enable | | |
| 124 | PB3.Data | | |
| 123 | PB3.Control | | |
| 122 | PB3.Pullup_Enable | | |
| 121 | PB4.Data | | |
| 120 | PB4.Control | | |
| 119 | PB4.Pullup_Enable | | |
| 118 | PB5.Data | | |
| 117 | PB5.Control | | |
| 116 | PB5.Pullup_Enable | | |
| 115 | PB6.Data | | |
| 114 | PB6.Control | | |
| 113 | PB6.Pullup_Enable | | |
| 112 | PB7.Data | | |
| 111 | PB7.Control | | |
| 110 | PB7.Pullup_Enable | | |
| 109 | PG3.Data | | Port G |
| 108 | PG3.Control | | |
| 107 | PG3.Pullup_Enable | | |
| 106 | PG4.Data | | |
| 105 | PG4.Control | | |
| 104 | PG4.Pullup_Enable | | |
| 103 | - | (Private Signal) | |
| 102 | RSTT | (Observe Only) | RESET Logic |
| 101 | RSTHV | | |
| 100 | EXTCLKEN | | Oscillators |
| 99 | OSCON | | |
| 98 | OSC32EN | | |
| 97 | TOSKON | | |
| 96 | EXTCLK | (XTAL1) | |
| 95 | OSCCK | | |
| 94 | OSC32CK | | |

**Table 113.** AT90CAN128 Boundary-scan Order (Continued)

| Bit Number | Signal Name | Comment | Module |
|---|---|---|---|
| 93 | TOSK | | Oscillators |
| 92 | PD0.Data | | Port D |
| 91 | PD0.Control | | |
| 90 | PD0.Pullup_Enable | | |
| 89 | PD1.Data | | |
| 88 | PD1.Control | | |
| 87 | PD1.Pullup_Enable | | |
| 86 | PD2.Data | | |
| 85 | PD2.Control | | |
| 84 | PD2.Pullup_Enable | | |
| 83 | PD3.Data | | |
| 82 | PD3.Control | | |
| 81 | PD3.Pullup_Enable | | |
| 80 | PD4.Data | | |
| 79 | PD4.Control | | |
| 78 | PD4.Pullup_Enable | | |
| 77 | PD5.Data | | |
| 76 | PD5.Control | | |
| 75 | PD5.Pullup_Enable | | |
| 74 | PD6.Data | | |
| 73 | PD6.Control | | |
| 72 | PD6.Pullup_Enable | | |
| 71 | PD7.Data | | |
| 70 | PD7.Control | | |
| 69 | PD7.Pullup_Enable | | |
| 68 | PG0.Data | | Port G |
| 67 | PG0.Control | | |
| 66 | PG0.Pullup_Enable | | |
| 65 | PG1.Data | | |
| 64 | PG1.Control | | |
| 63 | PG1.Pullup_Enable | | |
| 62 | PC0.Data | | Port C |
| 61 | PC0.Control | | |
| 60 | PC0.Pullup_Enable | | |
| 59 | PC1.Data | | |
| 58 | PC1.Control | | |
| 57 | PC1.Pullup_Enable | | |
| 56 | PC2.Data | | |

**Table 113.** AT90CAN128 Boundary-scan Order (Continued)

| Bit Number | Signal Name | Comment | Module |
|---|---|---|---|
| 55 | PC2.Control | | Port C |
| 54 | PC2.Pullup_Enable | | |
| 53 | PC3.Data | | |
| 52 | PC3.Control | | |
| 51 | PC3.Pullup_Enable | | |
| 50 | PC4.Data | | |
| 49 | PC4.Control | | |
| 48 | PC4.Pullup_Enable | | |
| 47 | PC5.Data | | |
| 46 | PC5.Control | | |
| 45 | PC5.Pullup_Enable | | |
| 44 | PC6.Data | | |
| 43 | PC6.Control | | |
| 42 | PC6.Pullup_Enable | | |
| 41 | PC7.Data | | |
| 40 | PC7.Control | | |
| 39 | PC7.Pullup_Enable | | |
| 38 | PG2.Data | | Port G |
| 37 | PG2.Control | | |
| 36 | PG2.Pullup_Enable | | |
| 35 | PA7.Data | | Port A |
| 34 | PA7.Control | | |
| 33 | PA7.Pullup_Enable | | |
| 32 | PA6.Data | | |
| 31 | PA6.Control | | |
| 30 | PA6.Pullup_Enable | | |
| 29 | PA5.Data | | |
| 28 | PA5.Control | | |
| 27 | PA5.Pullup_Enable | | |
| 26 | PA4.Data | | |
| 25 | PA4.Control | | |
| 24 | PA4.Pullup_Enable | | |
| 23 | PA3.Data | | |
| 22 | PA3.Control | | |
| 21 | PA3.Pullup_Enable | | |
| 20 | PA2.Data | | |
| 19 | PA2.Control | | |
| 18 | PA2.Pullup_Enable | | |

**Table 113.** AT90CAN128 Boundary-scan Order (Continued)

| Bit Number | Signal Name | Comment | Module |
|---|---|---|---|
| 17 | PA1.Data | | Port A |
| 16 | PA1.Control | | |
| 15 | PA1.Pullup_Enable | | |
| 14 | PA0.Data | | |
| 13 | PA0.Control | | |
| 12 | PA0.Pullup_Enable | | |
| 11 | PF3.Data | | Port F |
| 10 | PF3.Control | | |
| 9 | PF3.Pullup_Enable | | |
| 8 | PF2.Data | | |
| 7 | PF2.Control | | |
| 6 | PF2.Pullup_Enable | | |
| 5 | PF1.Data | | |
| 4 | PF1.Control | | |
| 3 | PF1.Pullup_Enable | | |
| 2 | PF0.Data | | |
| 1 | PF0.Control | | |
| 0 | PF0.Pullup_Enable | | |

## Boundary-scan Description Language Files

Boundary-scan Description Language (BSDL) files describe Boundary-scan capable devices in a standard format used by automated test-generation software. The order and function of bits in the Boundary-scan Data Register are included in this description. A BSDL file for AT90CAN128 is available.

# Boot Loader Support – Read-While-Write Self-Programming

The Boot Loader Support provides a real Read-While-Write Self-Programming mechanism for downloading and uploading program code by the MCU itself. This feature allows flexible application software updates controlled by the MCU using a Flash-resident Boot Loader program. The Boot Loader program can use any available data interface and associated protocol to read code and write (program) that code into the Flash memory, or read the code from the program memory. The program code within the Boot Loader section has the capability to write into the entire Flash, including the Boot Loader memory. The Boot Loader can thus even modify itself, and it can also erase itself from the code if the feature is not needed anymore. The size of the Boot Loader memory is configurable with fuses and the Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

## Features

- **Read-While-Write Self-Programming**
- **Flexible Boot Memory Size**
- **High Security (Separate Boot Lock Bits for a Flexible Protection)**
- **Separate Fuse to Select Reset Vector**
- **Optimized Page(1) Size**
- **Code Efficient Algorithm**
- **Efficient Read-Modify-Write Support**

Note: 1. A page is a section in the Flash consisting of several bytes (see Table 131 on page 330) used during programming. The page organization does not affect normal operation.

## Application and Boot Loader Flash Sections

The Flash memory is organized in two main sections, the Application section and the Boot Loader section (see Figure 154). The size of the different sections is configured by the BOOTSZ Fuses as shown in Table 119 on page 323 and Figure 154. These two sections can have different level of protection since they have different sets of Lock bits.

### AS - Application Section

The Application section is the section of the Flash that is used for storing the application code. The protection level for the Application section can be selected by the application Boot Lock bits (BLB02 and BLB01 bits), see Table 115 on page 315. The Application section can never store any Boot Loader code since the SPM instruction is disabled when executed from the Application section.

### BLS – Boot Loader Section

While the Application section is used for storing the application code, the The Boot Loader software must be located in the BLS since the SPM instruction can initiate a programming when executing from the BLS only. The SPM instruction can access the entire Flash, including the BLS itself. The protection level for the Boot Loader section can be selected by the Boot Loader Lock bits (BLB12 and BLB11 bits), see Table 116 on page 315.

## Read-While-Write and No Read-While-Write Flash Sections

Whether the CPU supports Read-While-Write or if the CPU is halted during a Boot Loader software update is dependent on which address that is being programmed. In addition to the two sections that are configurable by the BOOTSZ Fuses as described above, the Flash is also divided into two fixed sections, the Read-While-Write (RWW) section and the No Read-While-Write (NRWW) section. The limit between the RWW- and NRWW sections is given in Table 120 on page 323 and Figure 154 on page 314. The main difference between the two sections is:

- When erasing or writing a page located inside the RWW section, the NRWW section can be read during the operation.
- When erasing or writing a page located inside the NRWW section, the CPU is halted during the entire operation.

Note that the user software can never read any code that is located inside the RWW section during a Boot Loader software operation. The syntax “Read-While-Write section” refers to which section that is being programmed (erased or written), not which section that actually is being read during a Boot Loader software update.

**RWW – Read-While-Write Section**

If a Boot Loader software update is programming a page inside the RWW section, it is possible to read code from the Flash, but only code that is located in the NRWW section. During an on-going programming, the software must ensure that the RWW section never is being read. If the user software is trying to read code that is located inside the RWW section (i.e., by a call/jmp/lpm or an interrupt) during programming, the software might end up in an unknown state. To avoid this, the interrupts should either be disabled or moved to the Boot Loader section. The Boot Loader section is always located in the NRWW section. The RWW Section Busy bit (RWWSB) in the Store Program Memory Control and Status Register (SPMCSR) will be read as logical one as long as the RWW section is blocked for reading. After a programming is completed, the RWWSB must be cleared by software before reading code located in the RWW section. See “Store Program Memory Control and Status Register – SPMCSR” on page 316 for details on how to clear RWWSB.

**NRWW – No Read-While-Write Section**

The code located in the NRWW section can be read when the Boot Loader software is updating a page in the RWW section. When the Boot Loader code updates the NRWW section, the CPU is halted during the entire Page Erase or Page Write operation.

**Table 114.** Read-While-Write Features

| Which Section does the Z-pointer Address During the Programming? | Which Section Can be Read During Programming? | Is the CPU Halted? | Read-While-Write Supported? |
|---|---|---|---|
| RWW Section | NRWW Section | No | Yes |
| NRWW Section | None | Yes | No |

**Figure 153.** Read-While-Write vs. No Read-While-Write

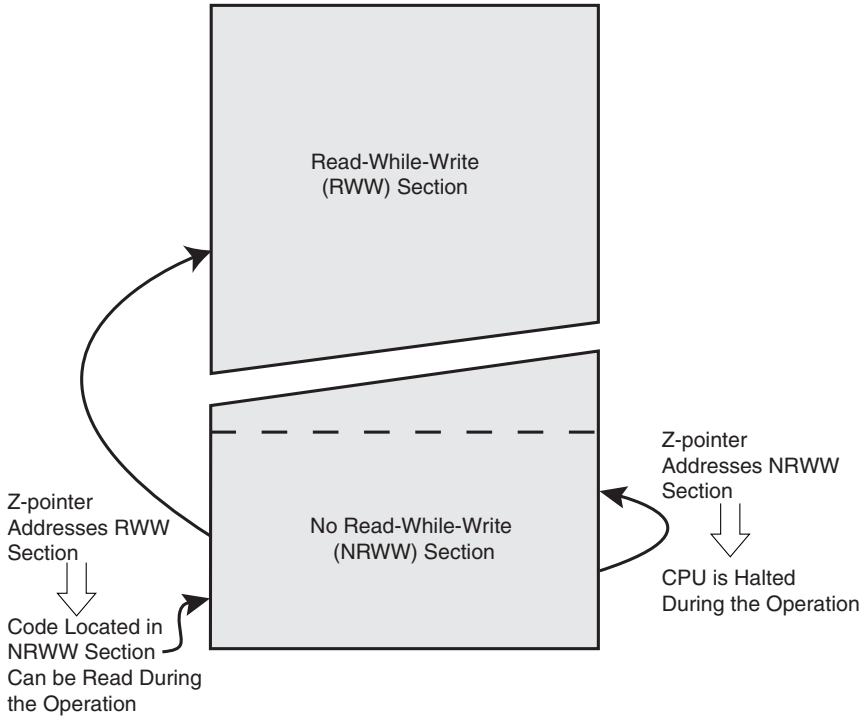
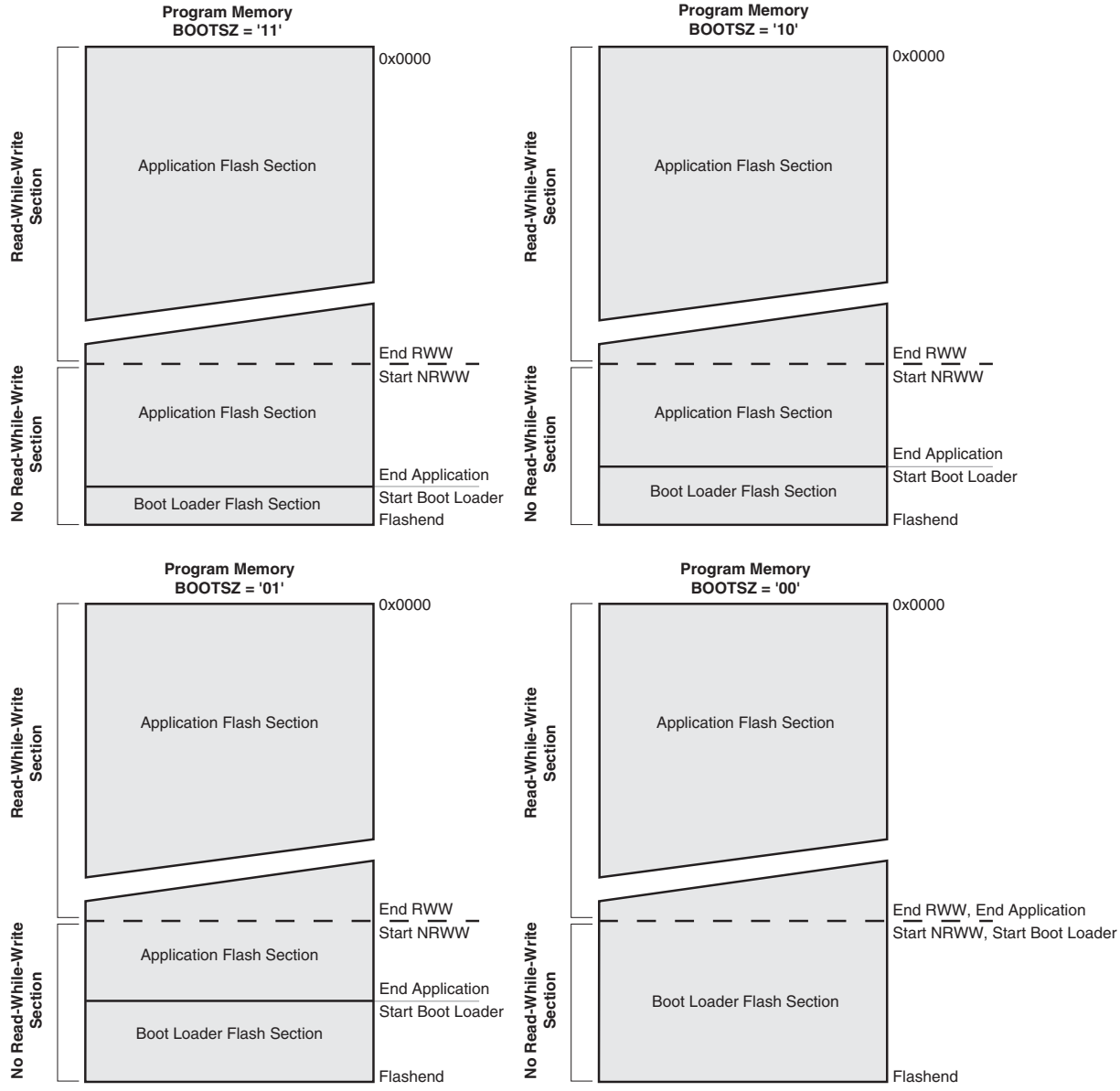


**Figure 154.** Memory Sections




Note: 1. The parameters in the figure above are given in Table 119 on page 323.

## Boot Loader Lock Bits

If no Boot Loader capability is needed, the entire Flash is available for application code. The Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

The user can select:

- To protect the entire Flash from a software update by the MCU.
- To protect only the Boot Loader Flash section from a software update by the MCU.
- To protect only the Application Flash section from a software update by the MCU.
- Allow software update in the entire Flash.

See Table 115 and Table 116 for further details. The Boot Lock bits can be set in software and in Serial or Parallel Programming mode, but they can be cleared by a Chip Erase command only. The general Write Lock (Lock Bit mode 2) does not control the programming of the Flash memory by SPM instruction. Similarly, the general Read/Write Lock (Lock Bit mode 1) does not control reading nor writing by LPM/SPM (Load Program Memory / Store Program Memory) instructions, if it is attempted.

**Table 115.** Boot Lock Bit0 Protection Modes (Application Section)(1)

| Lock Bit Mode | BLB02 | BLB01 | Protection |
|---|---|---|---|
| **1** | 1 | 1 | No restrictions for SPM or LPM accessing the Application section. |
| **2** | 1 | 0 | SPM is not allowed to write to the Application section. |
| **3** | 0 | 0 | SPM is not allowed to write to the Application section, and LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section. |
| **4** | 0 | 1 | LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section. |

Note: 1. “1” means unprogrammed, “0” means programmed

**Table 116.** Boot Lock Bit1 Protection Modes (Boot Loader Section)(1)

| Lock Bit Mode | BLB12 | BLB11 | Protection |
|---|---|---|---|
| **1** | 1 | 1 | No restrictions for SPM or LPM accessing the Boot Loader section. |
| **2** | 1 | 0 | SPM is not allowed to write to the Boot Loader section. |
| **3** | 0 | 0 | SPM is not allowed to write to the Boot Loader section, and LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section. |
| **4** | 0 | 1 | LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section. |

Note: 1. “1” means unprogrammed, “0” means programmed

## Entering the Boot Loader Program

Entering the Boot Loader takes place by a jump or call from the application program. This may be initiated by a trigger such as a command received via USART, or SPI interface. Alternatively, the Boot Reset Fuse can be programmed so that the Reset Vector is pointing to the Boot Flash start address after a reset. In this case, the Boot Loader is started after a reset. After the application code is loaded, the program can start executing the application code. Note that the fuses cannot be changed by the MCU itself. This means that once the Boot Reset Fuse is programmed, the Reset Vector will always

point to the Boot Loader Reset and the fuse can only be changed through the serial or parallel programming interface.

**Table 117.** Boot Reset Fuse(1)

| BOOTRST | Reset Address |
|---|---|
| 1 | Reset Vector = Application Reset (address 0x0000) |
| 0 | Reset Vector = Boot Loader Reset (see Table 119 on page 323) |

Note: 1. “1” means unprogrammed, “0” means programmed

### Store Program Memory Control and Status Register – SPMCSR

The Store Program Memory Control and Status Register contains the control bits needed to control the Boot Loader operations.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | SPMIE | RWWSB | – | RWWSRE | BLBSET | PGWRT | PGERS | SPMEN | SPMCSR |
| Read/Write | R/W | R | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – SPMIE: SPM Interrupt Enable**

When the SPMIE bit is written to one, and the I-bit in the Status Register is set (one), the SPM ready interrupt will be enabled. The SPM ready Interrupt will be executed as long as the SPMEN bit in the SPMCSR Register is cleared.

- **Bit 6 – RWWSB: Read-While-Write Section Busy**

When a Self-Programming (Page Erase or Page Write) operation to the RWW section is initiated, the RWWSB will be set (one) by hardware. When the RWWSB bit is set, the RWW section cannot be accessed. The RWWSB bit will be cleared if the RWWSRE bit is written to one after a Self-Programming operation is completed. Alternatively the RWWSB bit will automatically be cleared if a page load operation is initiated.

- **Bit 5 – Res: Reserved Bit**

This bit is a reserved bit in the AT90CAN128 and always read as zero.

- **Bit 4 – RWWSRE: Read-While-Write Section Read Enable**

When programming (Page Erase or Page Write) to the RWW section, the RWW section is blocked for reading (the RWWSB will be set by hardware). To re-enable the RWW section, the user software must wait until the programming is completed (SPMEN will be cleared). Then, if the RWWSRE bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles re-enables the RWW section. The RWW section cannot be re-enabled while the Flash is busy with a Page Erase or a Page Write (SPMEN is set). If the RWWSRE bit is written while the Flash is being loaded, the Flash load operation will abort and the data loaded will be lost.

- **Bit 3 – BLBSET: Boot Lock Bit Set**

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles sets Boot Lock bits, according to the data in R0. The data in R1 and the address in the Z-pointer/RAMPZ are ignored. The BLBSET bit will automatically be cleared upon completion of the Lock bit set, or if no SPM instruction is executed within four clock cycles.

An LPM instruction within three cycles after BLBSET and SPMEN are set in the SPMCSR Register, will read either the Lock bits or the Fuse bits (depending on Z0 in

the Z-pointer) into the destination register. See “Reading the Fuse and Lock Bits from Software” on page 320 for details.

- **Bit 2 – PGWRT: Page Write**

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles executes Page Write, with the data stored in the temporary buffer. The page address is taken from the high part of the Z-pointer and the low part of RAMPZ. The data in R1 and R0 are ignored. The PGWRT bit will auto-clear upon completion of a Page Write, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire Page Write operation if the NRWW section is addressed.

- **Bit 1 – PGERS: Page Erase**

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles executes Page Erase. The page address is taken from the high part of the Z-pointer and the low part of RAMPZ. The data in R1 and R0 are ignored. The PGERS bit will auto-clear upon completion of a Page Erase, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire Page Write operation if the NRWW section is addressed.

- **Bit 0 – SPMEN: Store Program Memory Enable**

This bit enables the SPM instruction for the next four clock cycles. If written to one together with either RWWSRE, BLBSET, PGWRT’ or PGERS, the following SPM instruction will have a special meaning, see description above. If only SPMEN is written, the following SPM instruction will store the value in R1:R0 in the temporary page buffer addressed by the Z-pointer/RAMPZ. The LSB of the Z-pointer is ignored. The SPMEN bit will auto-clear upon completion of an SPM instruction, or if no SPM instruction is executed within four clock cycles. During Page Erase and Page Write, the SPMEN bit remains high until the operation is completed.

Writing any other combination than “10001”, “01001”, “00101”, “00011” or “00001” in the lower five bits will have no effect.

## Addressing the Flash During Self-Programming

The Z-pointer together with RAMPZ are used to address the SPM commands. For details on how to use the RAMPZ, see “RAM Page Z Select Register – RAMPZ” on page 12.

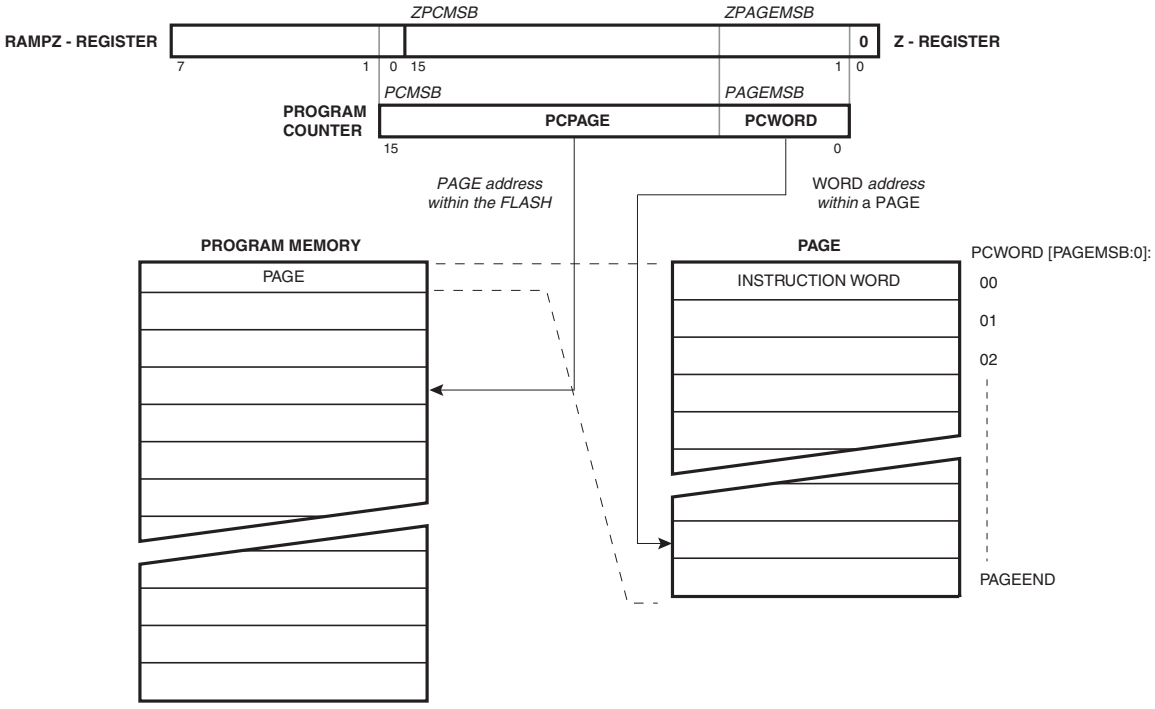
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| **ZH (R31)** | **Z15** | **Z14** | **Z13** | **Z12** | **Z11** | **Z10** | **Z9** | **Z8** |
| **ZL (R30)** | **Z7** | **Z6** | **Z5** | **Z4** | **Z3** | **Z2** | **Z1** | **Z0** |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Since the Flash is organized in pages (see Table 131 on page 330), the program counter can be treated as having two different sections. One section, consisting of the least significant bits, is addressing the words within a page, while the most significant bits are addressing the pages. This is shown in Figure 155. Note that the page erase and page write operations are addressed independently. Therefore it is of major importance that the Boot Loader software addresses the same page in both the page erase and page write operation. Once a programming operation is initiated, the address is latched and the Z-pointer/RAMPZ can be used for other operations.

The only SPM operation that does not use the Z-pointer/RAMPZ is setting the Boot Loader Lock bits. The content of the Z-pointer/RAMPZ is ignored and will have no effect on the operation. The (E)LPM instruction does also use the Z-pointer/RAMPZ to store

the address. Since this instruction addresses the Flash byte by byte, also the LSB (bit Z0) of the Z-pointer is used.

**Figure 155.** Addressing the Flash During SPM(1)




Note: 1. The different variables used in Figure 155 are listed in Table 121 on page 324.

## Self-Programming the Flash

The program memory is updated in a page by page fashion. Before programming a page with the data stored in the temporary page buffer, the page must be erased. The temporary page buffer is filled one word at a time using SPM and the buffer can be filled either before the Page Erase command or between a Page Erase and a Page Write operation:

**Alternative 1**: fill the buffer before a Page Erase

- Fill temporary page buffer
- Perform a Page Erase
- Perform a Page Write

**Alternative 2**: fill the buffer after Page Erase

- Perform a Page Erase
- Fill temporary page buffer
- Perform a Page Write

If only a part of the page needs to be changed, the rest of the page must be stored (for example in the temporary page buffer) before the erase, and then be rewritten. When using alternative 1, the Boot Loader provides an effective Read-Modify-Write feature which allows the user software to first read the page, do the necessary changes, and then write back the modified data. If alternative 2 is used, it is not possible to read the old data while loading since the page is already erased. The temporary page buffer can be accessed in a random sequence. It is essential that the page address used in both

the Page Erase and Page Write operation is addressing the same page. See “Simple Assembly Code Example for a Boot Loader” on page 321 for an assembly code example.

**Performing Page Erase by SPM**

To execute Page Erase, set up the address in the Z-pointer/RAMPZ, write “X0000011” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The data in R1 and R0 is ignored. The page address must be written to PCPAGE in the Z-register and RAMPZ. Other bits in the Z-pointer must be written zero during this operation.

- Page Erase to the RWW section: The NRWW section can be read during the page erase.
- Page Erase to the NRWW section: The CPU is halted during the operation.

**Filling the Temporary Buffer (Page Loading)**

To write an instruction word, set up the address in the Z-pointer/RAMPZ and data in R1:R0, write “00000001” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The content of PCWORD in the Z-register is used to address the data in the temporary buffer. The temporary buffer will auto-erase after a Page Write operation or by writing the RWWSRE bit in SPMCSR. It is also erased after a system reset. Note that it is not possible to write more than one time to each address without erasing the temporary buffer.

If the EEPROM is written in the middle of an SPM Page Load operation, all data loaded will be lost.

**Performing a Page Write**

To execute Page Write, set up the address in the Z-pointer/RAMPZ, write “X0000101” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The data in R1 and R0 is ignored. The page address must be written to PCPAGE. Other bits in the Z-pointer will be ignored during this operation.

- Page Write to the RWW section: The NRWW section can be read during the Page Write.
- Page Write to the NRWW section: The CPU is halted during the operation.

**Using the SPM Interrupt**

If the SPM interrupt is enabled, the SPM interrupt will generate a constant interrupt when the SPMEN bit in SPMCSR is cleared. This means that the interrupt can be used instead of polling the SPMCSR Register in software. When using the SPM interrupt, the Interrupt Vectors should be moved to the BLS section to avoid that an interrupt is accessing the RWW section when it is blocked for reading. How to move the interrupts is described in “Interrupts” on page 56.

**Consideration While Updating BLS**

Special care must be taken if the user allows the Boot Loader section to be updated by leaving Boot Lock bit11 unprogrammed. An accidental write to the Boot Loader itself can corrupt the entire Boot Loader, and further software updates might be impossible. If it is not necessary to change the Boot Loader software itself, it is recommended to program the Boot Lock bit11 to protect the Boot Loader software from any internal software changes.

**Prevent Reading the RWW Section During Self-Programming**

During Self-Programming (either Page Erase or Page Write), the RWW section is always blocked for reading. The user software itself must prevent that this section is addressed during the self programming operation. The RWWSB in the SPMCSR will be set as long as the RWW section is busy. During Self-Programming the Interrupt Vector table should be moved to the BLS as described in “Interrupts” on page 56, or the interrupts must be disabled. Before addressing the RWW section after the programming is completed, the user software must clear the RWWSB by writing the RWWSRE. See “Simple Assembly Code Example for a Boot Loader” on page 321 for an example.

**Setting the Boot Loader Lock Bits by SPM**

To set the Boot Loader Lock bits, write the desired data to R0, write “X0001001” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The only accessible Lock bits are the Boot Lock bits that may prevent the Application and Boot Loader section from any software update by the MCU.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **R0** | **1** | **1** | **BLB12** | **BLB11** | **BLB02** | **BLB01** | **1** | **1** |

See Table 115 and Table 116 for how the different settings of the Boot Loader bits affect the Flash access.

If bits 5..2 in R0 are cleared (zero), the corresponding Boot Lock bit will be programmed if an SPM instruction is executed within four cycles after BLBSET and SPMEN are set in SPMCSR. The Z-pointer is don’t care during this operation, but for future compatibility it is recommended to load the Z-pointer with 0x0001 (same as used for reading the Lock bits). For future compatibility it is also recommended to set bits 7, 6, 1, and 0 in R0 to “1” when writing the Lock bits. When programming the Lock bits the entire Flash can be read during the operation.

**EEPROM Write Prevents Writing to SPMCSR**

Note that an EEPROM write operation will block all software programming to Flash. Reading the Fuses and Lock bits from software will also be prevented during the EEPROM write operation. It is recommended that the user checks the status bit (EEWE) in the EECR Register and verifies that the bit is cleared before writing to the SPMCSR Register.

**Reading the Fuse and Lock Bits from Software**

It is possible to read both the Fuse and Lock bits from software. To read the Lock bits, load the Z-pointer with 0x0001 and set the BLBSET and SPMEN bits in SPMCSR. When an LPM instruction is executed within three CPU cycles after the BLBSET and SPMEN bits are set in SPMCSR, the value of the Lock bits will be loaded in the destination register. The BLBSET and SPMEN bits will auto-clear upon completion of reading the Lock bits or if no LPM instruction is executed within three CPU cycles or no SPM instruction is executed within four CPU cycles. When BLBSET and SPMEN are cleared, LPM will work as described in the Instruction set Manual.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Rd (Z=0x0001) | – | – | **BLB12** | **BLB11** | **BLB02** | **BLB01** | **LB2** | **LB1** |

The algorithm for reading the Fuse Low byte is similar to the one described above for reading the Lock bits. To read the Fuse Low byte, load the Z-pointer with 0x0000 and set the BLBSET and SPMEN bits in SPMCSR. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCSR, the value of the Fuse Low byte (FLB) will be loaded in the destination register as shown below. Refer to Table 126 on page 327 for a detailed description and mapping of the Fuse Low byte.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Rd (Z=0x0000) | **FLB7** | **FLB6** | **FLB5** | **FLB4** | **FLB3** | **FLB2** | **FLB1** | **FLB0** |

Similarly, when reading the Fuse High byte, load 0x0003 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCSR, the value of the Fuse High byte (FHB) will be loaded in the destination register as shown below. Refer to Table 125 on page 326 for detailed description and mapping of the Fuse High byte.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Rd (Z=0x0003) | **FHB7** | **FHB6** | **FHB5** | **FHB4** | **FHB3** | **FHB2** | **FHB1** | **FHB0** |

When reading the Extended Fuse byte, load 0x0002 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCSR, the value of the Extended Fuse byte (EFB) will be loaded in the destination register as shown below. Refer to Table 124 on page 326 for detailed description and mapping of the Extended Fuse byte.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Rd (Z=0x0002) | – | – | – | – | EFB3 | EFB2 | EFB1 | EFB0 |

Fuse and Lock bits that are programmed, will be read as zero. Fuse and Lock bits that are unprogrammed, will be read as one.

### Preventing Flash Corruption

During periods of low $V_{CC}$, the Flash program can be corrupted because the supply voltage is too low for the CPU and the Flash to operate properly. These issues are the same as for board level systems using the Flash, and the same design solutions should be applied.

A Flash program corruption can be caused by two situations when the voltage is too low.

- First, a regular write sequence to the Flash requires a minimum voltage to operate correctly.
- Secondly, the CPU itself can execute instructions incorrectly, if the supply voltage for executing instructions is too low.

Flash corruption can easily be avoided by following these design recommendations (one is sufficient):

1. If there is no need for a Boot Loader update in the system, program the Boot Loader Lock bits to prevent any Boot Loader software updates.
2. Keep the AVR RESET active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal Brown-out Detector (BOD) if the operating voltage matches the detection level. If not, an external low $V_{CC}$ reset protection circuit can be used. If a reset occurs while a write operation is in progress, the write operation will be completed provided that the power supply voltage is sufficient.
3. Keep the AVR core in Power-down sleep mode during periods of low $V_{CC}$. This will prevent the CPU from attempting to decode and execute instructions, effectively protecting the SPMCSR Register and thus the Flash from unintentional writes.

### Programming Time for Flash when Using SPM

The calibrated RC Oscillator is used to time Flash accesses. Table 118 shows the typical programming time for Flash accesses from the CPU.

**Table 118.** SPM Programming Time

| Symbol | Min Programming Time | Max Programming Time |
|---|---|---|
| Flash write (Page Erase, Page Write, and write Lock bits by SPM) | 3.7 ms | 4.5 ms |

### Simple Assembly Code Example for a Boot Loader

```
;- the routine writes one page of data from RAM to Flash
; the first data location in RAM is pointed to by the Y-pointer
; the first data location in Flash is pointed to by the Z-pointer
;- error handling is not included
;- the routine must be placed inside the Boot space
; (at least the Do_spm sub routine). Only code inside NRWW section can
; be read during Self-Programming (Page Erase and Page Write).
;- registers used: r0, r1, temp1 (r16), temp2 (r17), looplo (r24),
```

```
  ; loophi (r25), spmcsrval (r20)
  ; storing and restoring of registers is not included in the routine
  ; register usage can be optimized at the expense of code size
  ;- it is assumed that either the interrupt table is moved to the Boot
  ; loader section or that the interrupts are disabled.

.equ  PAGESIZEB = PAGESIZE*2      ;PAGESIZEB is page size in BYTES, not words
.org SMALLBOOTSTART

Write_page:
  ; Page Erase
  ldi   spmcsrval, (1<<PGERS) | (1<<SPMEN)
  call  Do_spm

  ; re-enable the RWW section
  ldi   spmcsrval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm

  ; transfer data from RAM to Flash page buffer
  ldi   looplo, low(PAGESIZEB)      ;init loop variable
  ldi   loophi, high(PAGESIZEB)     ;not required for PAGESIZEB<=256

Wrloop:
  ld    r0, Y+
  ld    r1, Y+
  ldi   spmcsrval, (1<<SPMEN)
  call  Do_spm
  adiw  ZH:ZL, 2
  sbiw  loophi:looplo, 2            ;use subi for PAGESIZEB<=256
  brne  Wrloop

  ; execute Page Write
  subi  ZL, low(PAGESIZEB)          ;restore pointer
  sbci  ZH, high(PAGESIZEB)         ;not required for PAGESIZEB<=256
  ldi   spmcsrval, (1<<PGWRT) | (1<<SPMEN)
  call  Do_spm

  ; re-enable the RWW section
  ldi   spmcsrval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm

  ; read back and check, optional
  ldi   looplo, low(PAGESIZEB)      ;init loop variable
  ldi   loophi, high(PAGESIZEB)     ;not required for PAGESIZEB<=256
  subi  YL, low(PAGESIZEB)          ;restore pointer
  sbci  YH, high(PAGESIZEB)

Rdloop:
  lpm   r0, Z+
  ld    r1, Y+
  cpse  r0, r1
  jmp   Error
  sbiw  loophi:looplo, 1            ;use subi for PAGESIZEB<=256
  brne  Rdloop

  ; return to RWW section
  ; verify that RWW section is safe to read

Return:
  in    temp1, SPMCSR
  sbrs  temp1, RWWSB       ; If RWWSB is set, the RWW section is not ready yet
  ret
  ; re-enable the RWW section
```

```
  ldi   spmcsrval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm
  rjmp  Return

Do_spm:
  ; check for previous SPM complete
Wait_spm:
  in    temp1, SPMCSR
  sbrc  temp1, SPMEN
  rjmp  Wait_spm
  ; input: spmcsrval determines SPM action
  ; disable interrupts if enabled, store status
  in    temp2, SREG
  cli
  ; check that no EEPROM write access is present
Wait_ee:
  sbic  EECR, EEWE
  rjmp  Wait_ee
  ; SPM timed sequence
  out   SPMCSR, spmcsrval
  spm
  ; restore SREG (to enable interrupts if originally enabled)
  out   SREG, temp2
  ret
```

### AT90CAN128 Boot Loader Parameters

In Table 119 through Table 121, the parameters used in the description of the Self-Programming are given.

**Table 119.** Boot Size Configuration (Word Addresses)(1)

| BOOTSZ1 | BOOTSZ0 | Boot Size | Pages | Application Flash Section | Boot Loader Flash Section | End Application Section | Boot Reset Address (Start Boot Loader Section) |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 512 words | 4 | 0x0000 - 0xFDFF | 0xFE00 - 0xFFFF | 0xFDFF | 0xFE00 |
| 1 | 0 | 1024 words | 8 | 0x0000 - 0xFBFF | 0xFC00 - 0xFFFF | 0xFBFF | 0xFC00 |
| 0 | 1 | 2048 words | 16 | 0x0000 - 0xF7FF | 0xF800 - 0xFFFF | 0xF7FF | 0xF800 |
| 0 | 0 | 4096 words | 32 | 0x0000 - 0xEFFF | 0xF000 - 0xFFFF | 0xEFFF | 0xF000 |

Note: 1. The different BOOTSZ Fuse configurations are shown in Figure 154

**Table 120.** Read-While-Write Limit (Word Addresses)(1)

| Section | Pages | Address |
|---|---|---|
| Read-While-Write section (RWW) | 480 | 0x0000 - 0xEFFF |
| No Read-While-Write section (NRWW) | 32 | 0xF000 - 0xFFFF |

Note: 1. For details about these two section, see “NRWW – No Read-While-Write Section” on page 312 and “RWW – Read-While-Write Section” on page 312.

**Table 121.** Explanation of Different Variables Used in Figure 155 and the Mapping to the Z-Pointer/RAMPZ(3)

| Variable | | Corresponding Z-value | Description(2) |
|---|---|---|---|
| PCMSB | 15 | | Most significant bit in the program counter. (The program counter is 16 bits PC[15:0]) |
| PAGEMSB | 6 | | Most significant bit which is used to address the words within one page (128 words in a page requires 7 bits PC [6:0]). |
| ZPCMSB | | Z16(1) | Bit in Z-register that is mapped to PCMSB. Because Z0 is not used, the ZPCMSB equals PCMSB + 1. |
| ZPAGEMSB | | Z7 | Bit in Z-register that is mapped to PAGEMSB. Because Z0 is not used, the ZPAGEMSB equals PAGEMSB + 1. |
| PCPAGE | PC[15:7] | Z16(1):Z7 | Program counter page address: Page select, for Page Erase and Page Write. |
| PCWORD | PC[6:0] | Z7:Z1 | Program counter word address: Word select, for filling temporary buffer (must be zero during PAGE WRITE operation). |

Notes: 1. The Z-register is only 16 bits wide. Bit 16 is located in the RAMPZ register in the I/O map.
2. Z0: should be zero for all SPM commands, byte select for the (E)LPM instruction.
3. See “Addressing the Flash During Self-Programming” on page 317 for details about the use of Z-pointer/RAMPZ during self-programming.

# Memory Programming

## Program and Data Memory Lock Bits

The AT90CAN128 provides six Lock bits which can be left unprogrammed (“1”) or can be programmed (“0”) to obtain the additional features listed in Table 123. The Lock bits can only be erased to “1” with the Chip Erase command.

**Table 122.** Lock Bit Byte(1)

| Lock Bit Byte | Bit No | Description | Default Value |
|---|---|---|---|
| | 7 | – | 1 (unprogrammed) |
| | 6 | – | 1 (unprogrammed) |
| BLB12 | 5 | Boot Lock bit | 1 (unprogrammed) |
| BLB11 | 4 | Boot Lock bit | 1 (unprogrammed) |
| BLB02 | 3 | Boot Lock bit | 1 (unprogrammed) |
| BLB01 | 2 | Boot Lock bit | 1 (unprogrammed) |
| LB2 | 1 | Lock bit | 1 (unprogrammed) |
| LB1 | 0 | Lock bit | 1 (unprogrammed) |

Note: 1. “1” means unprogrammed, “0” means programmed

**Table 123.** Lock Bit Protection Modes(1)(2)

| Memory Lock Bits | | | Protection Type |
|---|---|---|---|
| **LB Mode** | **LB2** | **LB1** | |
| 1 | 1 | 1 | No memory lock features enabled. |
| 2 | 1 | 0 | Further programming of the Flash and EEPROM is disabled in Parallel and Serial Programming mode. The Fuse bits are locked in both Serial and Parallel Programming mode.(1) |
| 3 | 0 | 0 | Further programming and verification of the Flash and EEPROM is disabled in Parallel and Serial Programming mode. The Boot Lock bits and Fuse bits are locked in both Serial and Parallel Programming mode.(1) |
| **BLB0 Mode** | **BLB02** | **BLB01** | |
| 1 | 1 | 1 | No restrictions for SPM (Store Program Memory) or LPM (Load Program Memory) accessing the Application section. |
| 2 | 1 | 0 | SPM is not allowed to write to the Application section. |
| 3 | 0 | 0 | SPM is not allowed to write to the Application section, and LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section. |
| 4 | 0 | 1 | LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section. |
| **BLB1 Mode** | **BLB12** | **BLB11** | |
| 1 | 1 | 1 | No restrictions for SPM or LPM accessing the Boot Loader section. |

**Table 123.** Lock Bit Protection Modes[1][2] (Continued)

| Memory Lock Bits | | | Protection Type |
|---|---|---|---|
| 2 | 1 | 0 | SPM is not allowed to write to the Boot Loader section. |
| 3 | 0 | 0 | SPM is not allowed to write to the Boot Loader section, and LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section. |
| 4 | 0 | 1 | LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section. |

Notes: 1. Program the Fuse bits and Boot Lock bits before programming the LB1 and LB2.
2. "1" means unprogrammed, "0" means programmed

## Fuse Bits

The AT90CAN128 has three Fuse bytes. Table 124, Table 125 and Table 126 describe briefly the functionality of all the fuses and how they are mapped into the Fuse bytes. Note that the fuses are read as logical zero, "0", if they are programmed.

**Table 124.** Extended Fuse Byte

| Fuse Extended Byte | Bit No | Description | Default Value |
|---|---|---|---|
| – | 7 | – | 1 |
| – | 6 | – | 1 |
| – | 5 | – | 1 |
| – | 4 | – | 1 |
| BODLEVEL2[1] | 3 | Brown-out Detector trigger level | 1 (unprogrammed) |
| BODLEVEL1[1] | 2 | Brown-out Detector trigger level | 1 (unprogrammed) |
| BODLEVEL0[1] | 1 | Brown-out Detector trigger level | 1 (unprogrammed) |
| TA0SEL | 0 | (Reserved for factory tests) | 1 (unprogrammed) |

Notes: 1. See Table 20 on page 50 for BODLEVEL Fuse decoding.

**Table 125.** Fuse High Byte

| Fuse High Byte | Bit No | Description | Default Value |
|---|---|---|---|
| OCDEN[4] | 7 | Enable OCD | 1 (unprogrammed, OCD disabled) |
| JTAGEN[5] | 6 | Enable JTAG | 0 (programmed, JTAG enabled) |
| SPIEN[1] | 5 | Enable Serial Program and Data Downloading | 0 (programmed, SPI prog. enabled) |
| WDTON[3] | 4 | Watchdog Timer always on | 1 (unprogrammed) |
| EESAVE | 3 | EEPROM memory is preserved through the Chip Erase | 1 (unprogrammed, EEPROM not preserved) |

**Table 125.** Fuse High Byte (Continued)

| Fuse High Byte | Bit No | Description | Default Value |
|---|---|---|---|
| BOOTSZ1 | 2 | Select Boot Size (see Table 119 for details) | 0 (programmed)(2) |
| BOOTSZ0 | 1 | Select Boot Size (see Table 119 for details) | 0 (programmed)(2) |
| BOOTRST | 0 | Select Reset Vector | 1 (unprogrammed) |

Note:
1. The SPIEN Fuse is not accessible in serial programming mode.
2. The default value of BOOTSZ1..0 results in maximum Boot Size. See Table 119 on page 323 for details.
3. See “Watchdog Timer Control Register – WDTCR” on page 54 for details.
4. Never ship a product with the OCDEN Fuse programmed regardless of the setting of Lock bits and JTAGEN Fuse. A programmed OCDEN Fuse enables some parts of the clock system to be running in all sleep modes. This may increase the power consumption.
5. If the JTAG interface is left unconnected, the JTAGEN fuse should if possible be disabled. This to avoid static current at the TDO pin in the JTAG interface.

**Table 126.** Fuse Low Byte

| Fuse Low Byte | Bit No | Description | Default Value |
|---|---|---|---|
| CKDIV8(4) | 7 | Divide clock by 8 | 0 (programmed) |
| CKOUT(3) | 6 | Clock output | 1 (unprogrammed) |
| SUT1 | 5 | Select start-up time | 1 (unprogrammed)(1) |
| SUT0 | 4 | Select start-up time | 0 (programmed)(1) |
| CKSEL3 | 3 | Select Clock source | 0 (programmed)(2) |
| CKSEL2 | 2 | Select Clock source | 0 (programmed)(2) |
| CKSEL1 | 1 | Select Clock source | 1 (unprogrammed)(2) |
| CKSEL0 | 0 | Select Clock source | 0 (programmed)(2) |

Note:
1. The default value of SUT1..0 results in maximum start-up time for the default clock source. See Table 12 on page 39 for details.
2. The default setting of CKSEL3..0 results in internal RC Oscillator @ 8 MHz. See Table 5 on page 35 for details.
3. The CKOUT Fuse allow the system clock to be output on Port PC7. See “Clock Output Buffer” on page 40 for details.
4. See “System Clock Prescaler” on page 41 for details.

The status of the Fuse bits is not affected by Chip Erase. Note that the Fuse bits are locked if Lock bit1 (LB1) is programmed. Program the Fuse bits before programming the Lock bits.

### Latching of Fuses

The fuse values are latched when the device enters programming mode and changes of the fuse values will have no effect until the part leaves Programming mode. This does not apply to the EESAVE Fuse which will take effect once it is programmed. The fuses are also latched on Power-up in Normal mode.

## Signature Bytes

All Atmel microcontrollers have a three-byte signature code which identifies the device. This code can be read in both serial and parallel mode, also when the device is locked. The three bytes reside in a separate address space.

For the AT90CAN128 the signature bytes are:

1. 0x000: 0x1E (indicates manufactured by Atmel).
2. 0x001: 0x97 (indicates 128KB Flash memory).
3. 0x002: 0x81 (indicates AT90CAN128 device when 0x001 is 0x97).

## Calibration Byte

The AT90CAN128 has a byte calibration value for the internal RC Oscillator. This byte resides in the high byte of address 0x000 in the signature address space. During reset, this byte is automatically written into the OSCCAL Register to ensure correct frequency of the calibrated RC Oscillator.

## Parallel Programming Overview

This section describes how to parallel program and verify Flash Program memory, EEPROM Data memory, Memory Lock bits, and Fuse bits in the AT90CAN128. Pulses are assumed to be at least 250 ns unless otherwise noted.

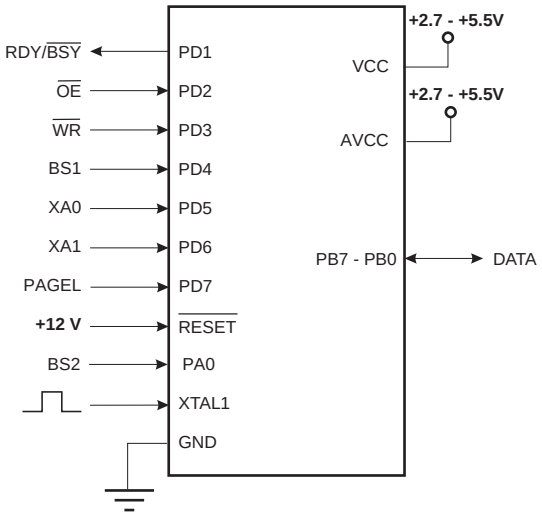
### Signal Names

In this section, some pins of the AT90CAN128 are referenced by signal names describing their functionality during parallel programming, see Figure 156 and Table 127. Pins not described in the following table are referenced by pin names.

The XA1/XA0 pins determine the action executed when the XTAL1 pin is given a positive pulse. The bit coding is shown in Table 129.

When pulsing $\overline{WR}$ or $\overline{OE}$, the command loaded determines the action executed. The different Commands are shown in Table 130.

**Figure 156.** Parallel Programming




### Pin Mapping

**Table 127.** Pin Name Mapping

| Signal Name in Programming Mode | Pin Name | I/O | Function |
|---|---|---|---|
| RDY/$\overline{BSY}$ | PD1 | O | 0: Device is busy programming,<br>1: Device is ready for new command. |
| $\overline{OE}$ | PD2 | I | Output Enable (Active low). |
| $\overline{WR}$ | PD3 | I | Write Pulse (Active low). |
| BS1 | PD4 | I | Byte Select 1<br>(“0” selects low byte, “1” selects high byte). |
| XA0 | PD5 | I | XTAL Action Bit 0 |
| XA1 | PD6 | I | XTAL Action Bit 1 |
| PAGEL | PD7 | I | Program Memory and EEPROM data Page Load. |
| BS2 | PA0 | I | Byte Select 2<br>(“0” selects low byte, “1” selects 2’nd high byte). |
| DATA | PB7-0 | I/O | Bi-directional Data bus (Output when $\overline{OE}$ is low). |

### Commands

**Table 128.** Pin Values Used to Enter Programming Mode

| Pin | Symbol | Value |
|---|---|---|
| PAGEL | Prog_enable[3] | 0 |
| XA1 | Prog_enable[2] | 0 |
| XA0 | Prog_enable[1] | 0 |
| BS1 | Prog_enable[0] | 0 |

**Table 129.** XA1 and XA0 Coding

| XA1 | XA0 | Action when XTAL1 is Pulsed |
|---|---|---|
| 0 | 0 | Load Flash or EEPROM Address (High or low address byte determined by BS1). |
| 0 | 1 | Load Data (High or Low data byte for Flash determined by BS1). |
| 1 | 0 | Load Command |
| 1 | 1 | No Action, Idle |

**Table 130.** Command Byte Bit Coding

| Command Byte | Command Executed |
|---|---|
| 1000 0000 | Chip Erase |
| 0100 0000 | Write Fuse bits |
| 0010 0000 | Write Lock bits |
| 0001 0000 | Write Flash |
| 0001 0001 | Write EEPROM |
| 0000 1000 | Read Signature bytes and Calibration byte |
| 0000 0100 | Read Fuse and Lock bits |
| 0000 0010 | Read Flash |
| 0000 0011 | Read EEPROM |

### Parameters

**Table 131.** No. of Words in a Page and No. of Pages in the Flash

| Flash Size | Page Size | PCWORD | No. of Pages | PCPAGE | PCMSB |
|---|---|---|---|---|---|
| 64K words (128K bytes) | 128 words | PC[6:0] | 512 | PC[15:7] | 15 |

**Table 132.** No. of Words in a Page and No. of Pages in the EEPROM

| EEPROM Size | Page Size | PCWORD | No. of Pages | PCPAGE | EEAMSB |
|---|---|---|---|---|---|
| 4K bytes | 8 bytes | EEA[2:0] | 512 | EEA[11:3] | 11 |

## Parallel Programming

### Enter Programming Mode

The following algorithm puts the device in parallel programming mode:

1. Apply 4.5 - 5.5V between $V_{CC}$ and GND.
2. Set $\overline{\text{RESET}}$ to “0” and toggle XTAL1 at least six times.
3. Set the Prog_enable pins listed in Table 128 on page 330 to “0000” and wait at least 100 ns.
4. Apply 11.5 - 12.5V to $\overline{\text{RESET}}$. Any activity on Prog_enable pins within 100 ns after +12V has been applied to $\overline{\text{RESET}}$, will cause the device to fail entering programming mode.
5. Wait at least 50 µs before sending a new command.

### Considerations for Efficient Programming

The loaded command and address are retained in the device during programming. For efficient programming, the following should be considered.

- The command needs only be loaded once when writing or reading multiple memory locations.
- Skip writing the data value 0xFF, that is the contents of the entire EEPROM (unless the EESAVE Fuse is programmed) and Flash after a Chip Erase.
- Address high byte needs only be loaded before programming or reading a new 256 word window in Flash or 256 byte EEPROM. This consideration also applies to Signature bytes reading.

### Chip Erase

The Chip Erase will erase the Flash and EEPROM(1) memories plus Lock bits. The Lock bits are not reset until the program memory has been completely erased. The Fuse bits are not changed. A Chip Erase must be performed before the Flash and/or EEPROM are reprogrammed.

Note: 1. The EEPRPOM memory is preserved during Chip Erase if the EESAVE Fuse is programmed.

Load Command “Chip Erase”

1. Set XA1, XA0 to “10”. This enables command loading.
2. Set BS1 to “0”.
3. Set DATA to “1000 0000”. This is the command for Chip Erase.
4. Give XTAL1 a positive pulse. This loads the command.
5. Give $\overline{\text{WR}}$ a negative pulse. This starts the Chip Erase. RDY/$\overline{\text{BSY}}$ goes low.
6. Wait until RDY/$\overline{\text{BSY}}$ goes high before loading a new command.

### Programming the Flash

The Flash is organized in pages, see Table 131 on page 330. When programming the Flash, the program data is latched into a page buffer. This allows one page of program data to be programmed simultaneously. The following procedure describes how to program the entire Flash memory:

A. Load Command “Write Flash”

1. Set XA1, XA0 to “10”. This enables command loading.
2. Set BS1 to “0”.
3. Set DATA to “0001 0000”. This is the command for Write Flash.
4. Give XTAL1 a positive pulse. This loads the command.

B. Load Address Low byte

1. Set XA1, XA0 to “00”. This enables address loading.
2. Set BS1 to “0”. This selects low address.
3. Set DATA = Address low byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the address low byte.

C. Load Data Low Byte

1. Set XA1, XA0 to “01”. This enables data loading.
2. Set DATA = Data low byte (0x00 - 0xFF).
3. Give XTAL1 a positive pulse. This loads the data byte.

D. Load Data High Byte

1. Set BS1 to “1”. This selects high data byte.
2. Set XA1, XA0 to “01”. This enables data loading.
3. Set DATA = Data high byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the data byte.

E. Latch Data

1. Set BS1 to “1”. This selects high data byte.
2. Give PAGEL a positive pulse. This latches the data bytes. (See Figure 158 for signal waveforms)

F. Repeat B through E until the entire buffer is filled or until all data within the page is loaded.

While the lower bits in the address are mapped to words within the page, the higher bits address the pages within the FLASH. This is illustrated in Figure 157 on page 333. Note that if less than eight bits are required to address words in the page (pagesize < 256), the most significant bit(s) in the address low byte are used to address the page when performing a Page Write.

G. Load Address High byte

1. Set XA1, XA0 to “00”. This enables address loading.
2. Set BS1 to “1”. This selects high address.
3. Set DATA = Address high byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the address high byte.

H. Program Page

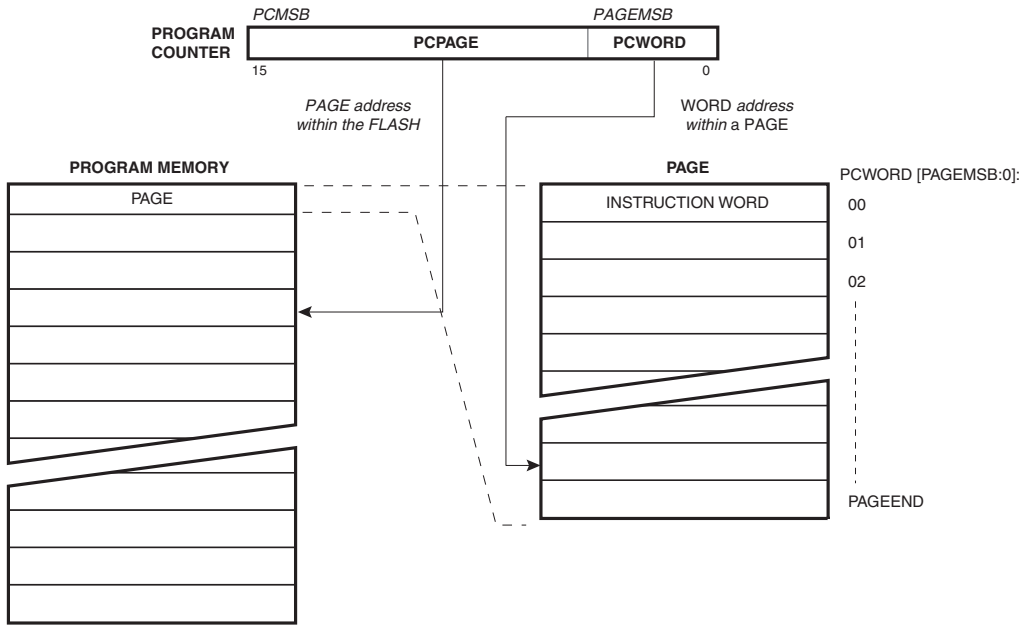
1. Give $\overline{WR}$ a negative pulse. This starts programming of the entire page of data. RDY/$\overline{BSY}$goes low.
2. Wait until RDY/$\overline{BSY}$ goes high (See Figure 158 for signal waveforms).

I. Repeat B through H until the entire Flash is programmed or until all data has been programmed.

J. End Page Programming

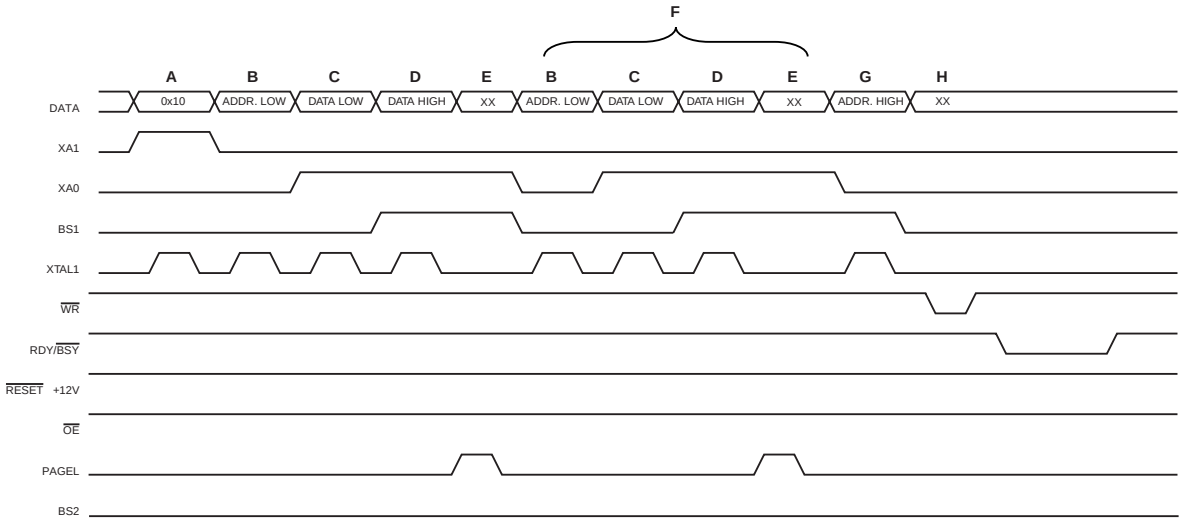
1. 1. Set XA1, XA0 to “10”. This enables command loading.
2. Set DATA to “0000 0000”. This is the command for No Operation.
3. Give XTAL1 a positive pulse. This loads the command, and the internal write signals are reset.

**Figure 157.** Addressing the Flash Which is Organized in Pages[1]




Note: 1. PCPAGE and PCWORD are listed in Table 131 on page 330.

**Figure 158.** Programming the Flash Waveforms[1]




Note: 1. "XX" is don't care. The letters refer to the programming description above.

**Programming the EEPROM**

The EEPROM is organized in pages, see Table 132 on page 330. When programming the EEPROM, the program data is latched into a page buffer. This allows one page of data to be programmed simultaneously. The programming algorithm for the EEPROM data memory is as follows (refer to "Programming the Flash" on page 331 for details on Command, Address and Data loading):

1. A: Load Command "0001 0001".
2. G: Load Address High Byte (0x00 - 0xFF).

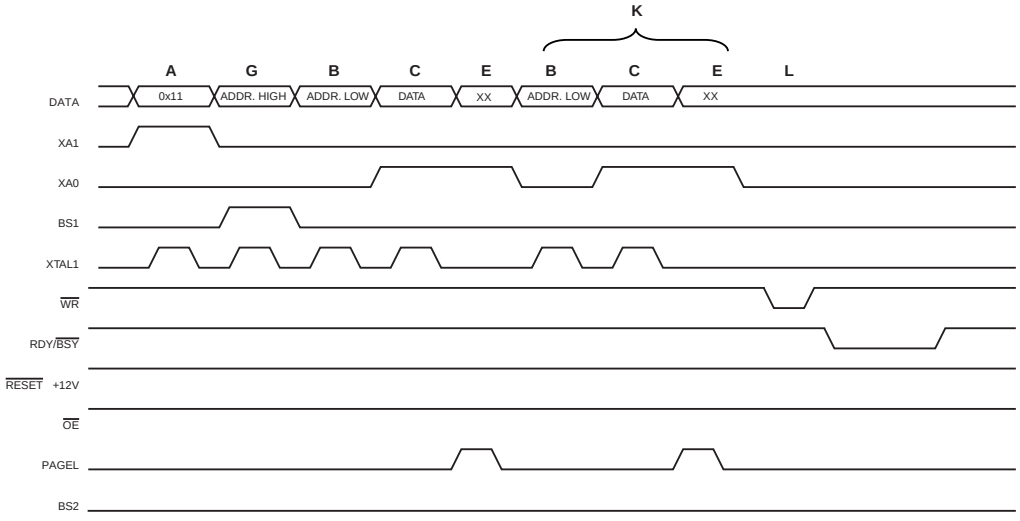
3. B: Load Address Low Byte (0x00 - 0xFF).
4. C: Load Data (0x00 - 0xFF).
5. E: Latch data (give PAGEL a positive pulse).

K: Repeat 3 through 5 until the entire buffer is filled.

L: Program EEPROM page

1. Set BS to “0”.
2. Give $\overline{WR}$ a negative pulse. This starts programming of the EEPROM page. RDY/$\overline{BSY}$ goes low.
3. Wait until to RDY/$\overline{BSY}$ goes high before programming the next page (See Figure 159 for signal waveforms).

**Figure 159.** Programming the EEPROM Waveforms




## Reading the Flash

The algorithm for reading the Flash memory is as follows (refer to “Programming the Flash” on page 331 for details on Command and Address loading):

1. A: Load Command “0000 0010”.
2. G: Load Address High Byte (0x00 - 0xFF).
3. B: Load Address Low Byte (0x00 - 0xFF).
4. Set $\overline{OE}$ to “0”, and BS1 to “0”. The Flash word low byte can now be read at DATA.
5. Set BS to “1”. The Flash word high byte can now be read at DATA.
6. Set $\overline{OE}$ to “1”.

**Reading the EEPROM**

The algorithm for reading the EEPROM memory is as follows (refer to “Programming the Flash” on page 331 for details on Command and Address loading):

1. A: Load Command “0000 0011”.
2. G: Load Address High Byte (0x00 - 0xFF).
3. B: Load Address Low Byte (0x00 - 0xFF).
4. Set $\overline{OE}$ to “0”, and BS1 to “0”. The EEPROM Data byte can now be read at DATA.
5. Set $\overline{OE}$ to “1”.

**Programming the Fuse Low Bits**

The algorithm for programming the Fuse Low bits is as follows (refer to “Programming the Flash” on page 331 for details on Command and Data loading):

1. A: Load Command “0100 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.

**Programming the Fuse High Bits**

The algorithm for programming the Fuse High bits is as follows (refer to “Programming the Flash” on page 331 for details on Command and Data loading):

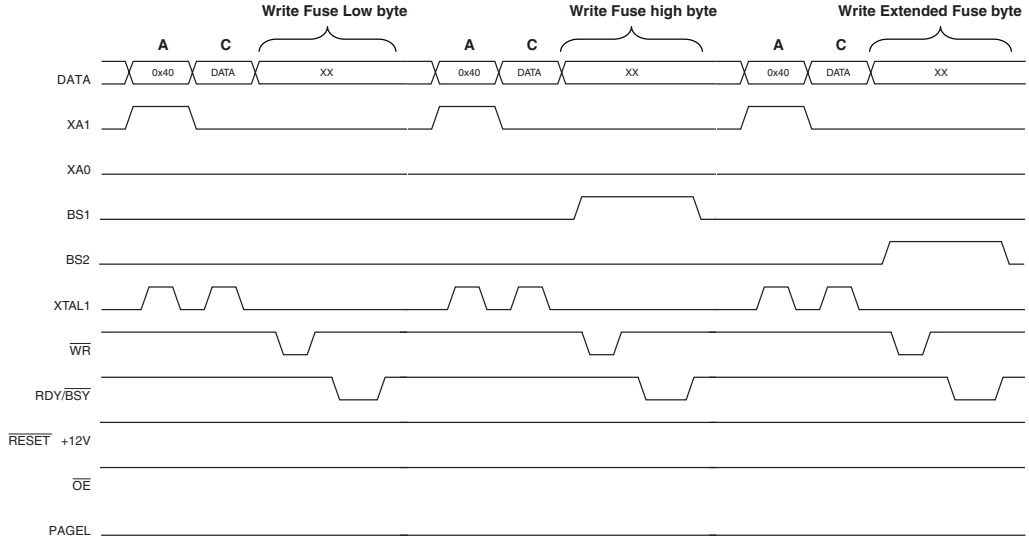
1. A: Load Command “0100 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Set BS1 to “1” and BS2 to “0”. This selects high data byte.
4. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.
5. Set BS1 to “0”. This selects low data byte.

**Programming the Extended Fuse Bits**

The algorithm for programming the Extended Fuse bits is as follows (refer to “Programming the Flash” on page 331 for details on Command and Data loading):

1. A: Load Command “0100 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Set BS1 to “0” and BS2 to “1”. This selects extended data byte.
4. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.
5. Set BS2 to “0”. This selects low data byte.

**Figure 160.** Programming the FUSES Waveforms



**Programming the Lock Bits**

The algorithm for programming the Lock bits is as follows (refer to “Programming the Flash” on page 331 for details on Command and Data loading):

1. A: Load Command “0010 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs the Lock bit. If LB mode 3 is programmed (LB1 and LB2 is programmed), it is not possible to program the Boot Lock bits by any External Programming mode.
3. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.

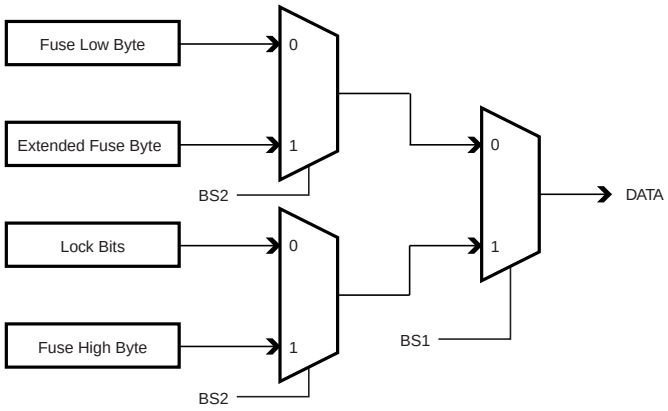
The Lock bits can only be cleared by executing Chip Erase.

**Reading the Fuse and Lock Bits**

The algorithm for reading the Fuse and Lock bits is as follows (refer to “Programming the Flash” on page 331 for details on Command loading):

1. A: Load Command “0000 0100”.
2. Set $\overline{OE}$ to “0”, BS2 to “0” and BS1 to “0”. The status of the Fuse Low bits can now be read at DATA (“0” means programmed).
3. Set $\overline{OE}$ to “0”, BS2 to “1” and BS1 to “1”. The status of the Fuse High bits can now be read at DATA (“0” means programmed).
4. Set OE to “0”, BS2 to “1”, and BS1 to “0”. The status fo the Extended Fuse bits can now be read at DATA (“0” means programmed).
5. Set $\overline{OE}$ to “0”, BS2 to “0” and BS1 to “1”. The status of the Lock bits can now be read at DATA (“0” means programmed).
6. Set $\overline{OE}$ to “1”.

**Figure 161.** Mapping Between BS1, BS2 and the Fuse and Lock Bits During Read




**Reading the Signature Bytes**

The algorithm for reading the Signature bytes is as follows (refer to “Programming the Flash” on page 331 for details on Command and Address loading):

1. A: Load Command “0000 1000”.
2. B: Load Address Low Byte (0x00 - 0x02).
3. Set $\overline{OE}$ to “0”, and BS to “0”. The selected Signature byte can now be read at DATA.
4. Set $\overline{OE}$ to “1”.

**Reading the Calibration Byte**

The algorithm for reading the Calibration byte is as follows (refer to “Programming the Flash” on page 331 for details on Command and Address loading):

1. A: Load Command “0000 1000”.
2. B: Load Address Low Byte, 0x00.
3. Set $\overline{OE}$ to “0”, and BS1 to “1”. The Calibration byte can now be read at DATA.
4. Set $\overline{OE}$ to “1”.

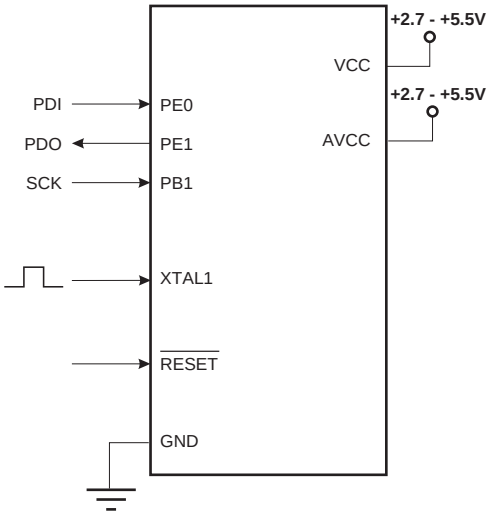
## SPI Serial Programming Overview

This section describes how to serial program and verify Flash Program memory, EEPROM Data memory, Memory Lock bits, and Fuse bits in the AT90CAN128.

### Signal Names

Both the Flash and EEPROM memory arrays can be programmed using the serial SPI bus while RESET is pulled to GND. The serial interface consists of pins SCK, MOSI (input) and MISO (output). After RESET is set low, the Programming Enable instruction needs to be executed first before program/erase operations can be executed. NOTE, in Table 133 on page 338, the pin mapping for SPI programming is listed. Not all parts use the SPI pins dedicated for the internal SPI interface. Note that throughout the description about Serial downloading, MOSI and MISO are used to describe the serial data in and serial data out respectively. For AT90CAN128 these pins are mapped to PDI (PE0) and PDO (PE1).

**Figure 162.** Serial Programming and Verify(1)




Notes: 1. If the device is clocked by the internal Oscillator, it is no need to connect a clock source to the XTAL1 pin.

When programming the EEPROM, an auto-erase cycle is built into the self-timed programming operation (in the Serial mode ONLY) and there is no need to first execute the Chip Erase instruction. The Chip Erase operation turns the content of every memory location in both the Program and EEPROM arrays into 0xFF.

**Depending on CKSEL Fuses, a valid clock must be present. The minimum low and high periods for the serial clock (SCK) input are defined as follows:**

**Low:** > 2 CPU clock cycles for $f_{ck}$ < 12 MHz, 3 CPU clock cycles for $f_{ck} \geq$ 12 MHz

**High:** > 2 CPU clock cycles for $f_{ck}$ < 12 MHz, 3 CPU clock cycles for $f_{ck} \geq$ 12 MHz

### Pin Mapping

**Table 133.** Pin Mapping Serial Programming

| Symbol | Pins | I/O | Description |
|---|---|---|---|
| MOSI (PDI) | PE0 | I | Serial Data in |
| MISO (PDO) | PE1 | O | Serial Data out |
| SCK | PB1 | I | Serial Clock |

### Parameters

The Flash parameters are given in Table 131 on page 330 and the EEPROM parameters in Table 132 on page 330.

## SPI Serial Programming

When writing serial data to the AT90CAN128, data is clocked on the rising edge of SCK. When reading data from the AT90CAN128, data is clocked on the falling edge of SCK.

To program and verify the AT90CAN128 in the serial programming mode, the following sequence is recommended (See four byte instruction formats in Table 135):

1. Power-up sequence:
   Apply power between $V_{CC}$ and GND while $\overline{\text{RESET}}$ and SCK are set to “0”. In some systems, the programmer can not guarantee that SCK is held low during power-up. In this case, $\overline{\text{RESET}}$ must be given a positive pulse of at least two CPU clock cycles duration after SCK has been set to “0”.
2. Wait for at least 20 ms and enable serial programming by sending the Programming Enable serial instruction to pin MOSI.
3. The serial programming instructions will not work if the communication is out of synchronization. When in sync. the second byte (0x53), will echo back when issuing the third byte of the Programming Enable instruction. Whether the echo is correct or not, all four bytes of the instruction must be transmitted. If the 0x53 did not echo back, give $\overline{\text{RESET}}$ a positive pulse and issue a new Programming Enable command.
4. The Flash is programmed one page at a time. The memory page is loaded one byte at a time by supplying the 7 LSB of the address and data together with the Load Program Memory Page instruction. To ensure correct loading of the page, the data low byte must be loaded before data high byte is applied for a given address. The Program Memory Page is stored by loading the Write Program Memory Page instruction with the 9 MSB of the address. If polling is not used, the user must wait at least $t_{WD_FLASH}$ before issuing the next page. (See Table 134.) Accessing the serial programming interface before the Flash write operation completes can result in incorrect programming.
5. The EEPROM array is programmed one byte at a time by supplying the address and data together with the appropriate Write instruction. An EEPROM memory location is first automatically erased before new data is written. If polling is not used, the user must wait at least $t_{WD_EEPROM}$ before issuing the next byte. (See Table 134.) In a chip erased device, no 0xFFs in the data file(s) need to be programmed.
6. Any memory location can be verified by using the Read instruction which returns the content at the selected address at serial output MISO.
7. At the end of the programming session, $\overline{\text{RESET}}$ can be set high to commence normal operation.
8. Power-off sequence (if needed):
   Set $\overline{\text{RESET}}$ to “1”.
   Turn $V_{CC}$ power off.

**Data Polling Flash**

When a page is being programmed into the Flash, reading an address location within the page being programmed will give the value 0xFF. At the time the device is ready for a new page, the programmed value will read correctly. This is used to determine when the next page can be written. Note that the entire page is written simultaneously and any address within the page can be used for polling. Data polling of the Flash will not work for the value 0xFF, so when programming this value, the user will have to wait for at least $t_{WD_FLASH}$ before programming the next page. As a chip-erased device contains 0xFF in all locations, programming of addresses that are meant to contain 0xFF, can be skipped. See Table 134 for $t_{WD_FLASH}$ value.

**Data Polling EEPROM**

When a new byte has been written and is being programmed into EEPROM, reading the address location being programmed will give the value 0xFF. At the time the device is ready for a new byte, the programmed value will read correctly. This is used to determine when the next byte can be written. This will not work for the value 0xFF, but the user should have the following in mind: As a chip-erased device contains 0xFF in all locations, programming of addresses that are meant to contain 0xFF, can be skipped. This does not apply if the EEPROM is re-programmed without chip erasing the device. In this case, data polling cannot be used for the value 0xFF, and the user will have to wait at least $t_{WD_EEPROM}$ before programming the next byte. See Table 134 for $t_{WD_EEPROM}$ value.

**Table 134.** Minimum Wait Delay Before Writing the Next Flash or EEPROM Location

| Symbol | Minimum Wait Delay |
|---|---|
| $t_{WD_FLASH}$ | 4.5 ms |
| $t_{WD_EEPROM}$ | 9.0 ms |
| $t_{WD_ERASE}$ | 9.0 ms |

**Figure 163.** Serial Programming Waveforms

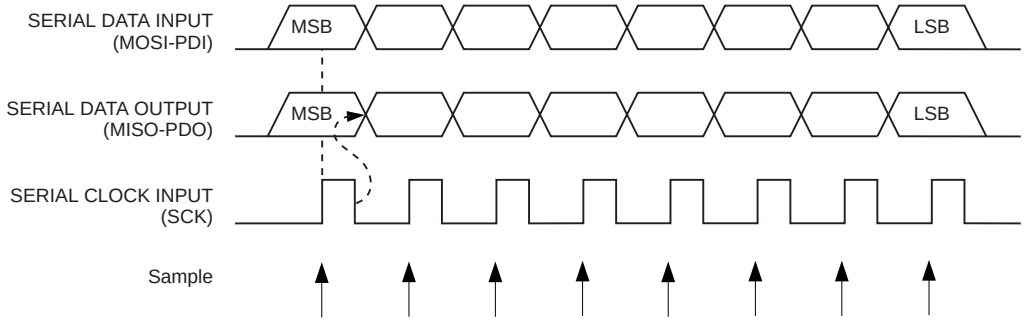


**Table 135.** Serial Programming Instruction Set

| Instruction | Instruction Format | | | | Operation |
|---|---|---|---|---|---|
| | Byte 1 | Byte 2 | Byte 3 | Byte4 | |
| Programming Enable | 1010 1100 | 0101 0011 | xxxx xxxx | xxxx xxxx | Enable Serial Programming after $\overline{\text{RESET}}$ goes low. |
| Chip Erase | 1010 1100 | 100x xxxx | xxxx xxxx | xxxx xxxx | Chip Erase EEPROM and Flash. |
| Read Program Memory | 0010 **H**000 | **aaaa aaaa** | **bbbb bbbb** | **oooo oooo** | Read **H** (high or low) data **o** from Program memory at word address **a**:**b**. |
| Load Program Memory Page | 0100 **H**000 | 000x xxxx | x**bbb bbbb** | **iiii iiii** | Write **H** (high or low) data **i** to Program Memory page at word address **b**. Data low byte must be loaded before Data high byte is applied within the same address. |
| Write Program Memory Page | 0100 1100 | **aaaa aaaa** | **b**xxx xxxx | xxxx xxxx | Write Program Memory Page at address **a**:**b**. |
| Read EEPROM Memory | 1010 0000 | 000x **aaaa** | **bbbb bbbb** | **oooo oooo** | Read data **o** from EEPROM memory at address **a**:**b**. |
| Write EEPROM Memory | 1100 0000 | 000x **aaaa** | **bbbb bbbb** | **iiii iiii** | Write data **i** to EEPROM memory at address **a**:**b**. |
| Load EEPROM Memory Page (page access) | 1100 0001 | 0000 0000 | 0000 0**bbb** | **iiii iiii** | Load data **i** to EEPROM memory page buffer. After data is loaded, program EEPROM page. |
| Write EEPROM Memory Page (page access) | 1100 0010 | 00xx **aaaa** | **bbbb b000** | xxxx xxxx | Write EEPROM page at address **a**:**b**. |
| Read Lock bits | 0101 1000 | 0000 0000 | xxxx xxxx | xx**oo oooo** | Read Lock bits.<br>“0” = programmed, “1” = unprogrammed.<br>See Table 122 on page 325 for details. |
| Write Lock bits | 1010 1100 | 111x xxxx | xxxx xxxx | 11**ii iiii** | Write Lock bits. Set bits = “0” to program Lock bits. See Table 122 on page 325 for details. |
| Read Signature Byte | 0011 0000 | 000x xxxx | xxxx xx**bb** | **oooo oooo** | Read Signature Byte **o** at address **b**. |
| Write Fuse Low bits | 1010 1100 | 1010 0000 | xxxx xxxx | **iiii iiii** | Set bits = “0” to program, “1” to unprogram. See Table 126 on page 327 for details. |
| Write Fuse High bits | 1010 1100 | 1010 1000 | xxxx xxxx | **iiii iiii** | Set bits = “0” to program, “1” to unprogram. See Table 125 on page 326 for details. |
| Write Extended Fuse Bits | 1010 1100 | 1010 0100 | xxxx xxxx | **xxxx iiii** | Set bits = “0” to program, “1” to unprogram. See Table 124 on page 326 for details. |
| Read Fuse Low bits | 0101 0000 | 0000 0000 | xxxx xxxx | **oooo oooo** | Read Fuse bits. “0” = programmed, “1” = unprogrammed. See Table 126 on page 327 for details. |
| Read Fuse High bits | 0101 1000 | 0000 1000 | xxxx xxxx | **oooo oooo** | Read Fuse High bits. “0” = pro-grammed, “1” = unprogrammed. See Table 125 on page 326 for details. |

**Table 135.** Serial Programming Instruction Set (Continued)

| Instruction | Instruction Format | | | | Operation |
|---|---|---|---|---|---|
| | Byte 1 | Byte 2 | Byte 3 | Byte4 | |
| Read Extended Fuse Bits | 0101 0000 | 0000 1000 | xxxx xxxx | oooo oooo | Read Extended Fuse bits. “0” = pro-grammed, “1” = unprogrammed. See Table 124 on page 326 for details. |
| Read Calibration Byte | 0011 1000 | 000x xxxx | 0000 0000 | oooo oooo | Read Calibration Byte |
| Poll RDY/$\overline{\text{BSY}}$ | 1111 0000 | 0000 0000 | xxxx xxxx | xxxx xxxo | If **o** = “1”, a programming operation is still busy. Wait until this bit returns to “0” before applying another command. |

Note: **a** = address high bits
**b** = address low bits
**H** = 0 - Low byte, 1 - High Byte
**o** = data out
**i** = data in
x = don’t care

## JTAG Programming Overview

Programming through the JTAG interface requires control of the four JTAG specific pins: TCK, TMS, TDI, and TDO. Control of the reset and clock pins is not required.

To be able to use the JTAG interface, the JTAGEN Fuse must be programmed. The device is default shipped with the fuse programmed. In addition, the JTD bit in MCUCR must be cleared. Alternatively, if the JTD bit is set, the external reset can be forced low. Then, the JTD bit will be cleared after two chip clocks, and the JTAG pins are available for programming. This provides a means of using the JTAG pins as normal port pins in Running mode while still allowing In-System Programming via the JTAG interface. Note that this technique can not be used when using the JTAG pins for Boundary-scan or On-chip Debug. In these cases the JTAG pins must be dedicated for this purpose.

During programming the clock frequency of the TCK Input must be less than the maximum frequency of the chip. The System Clock Prescaler can not be used to divide the TCK Clock Input into a sufficiently low frequency.

As a definition in this datasheet, the LSB is shifted in and out first of all Shift Registers.

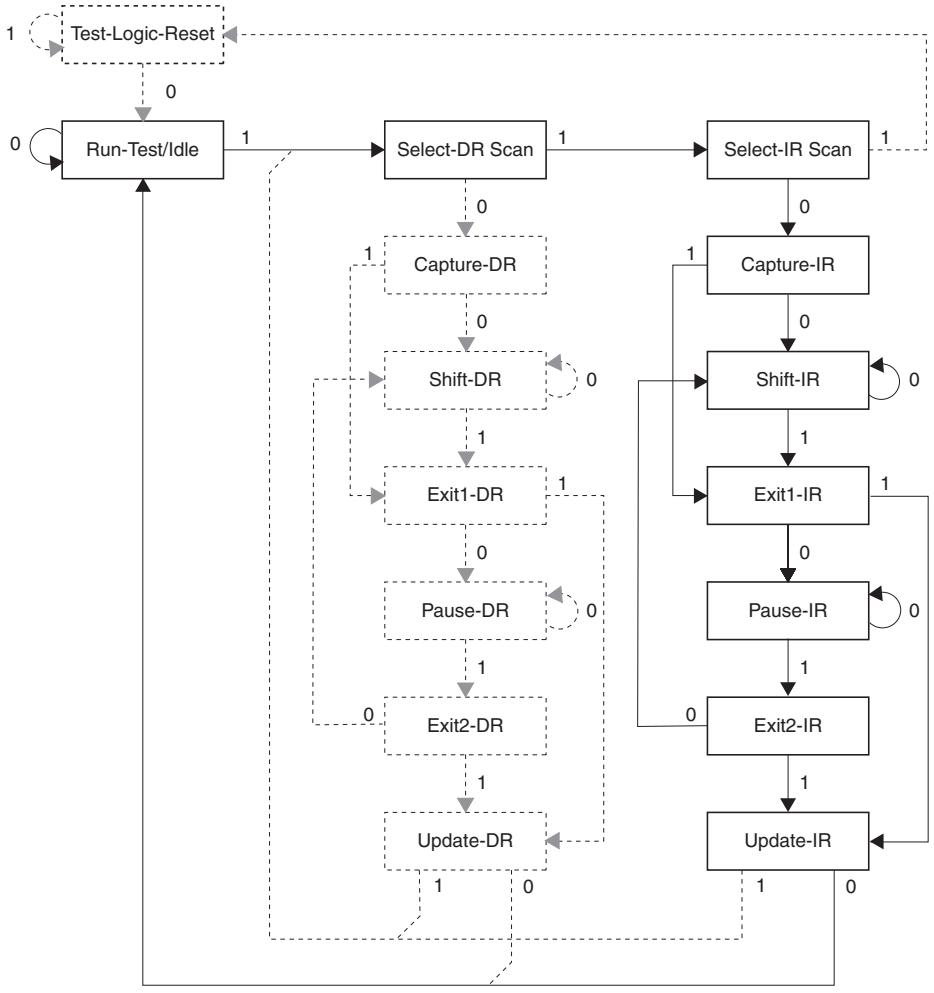
### Programming Specific JTAG Instructions

The instruction register is 4-bit wide, supporting up to 16 instructions. The JTAG instructions useful for programming are listed below.

The OPCODE for each instruction is shown behind the instruction name in hex format. The text describes which data register is selected as path between TDI and TDO for each instruction.

The Run-Test/Idle state of the TAP controller is used to generate internal clocks. It can also be used as an idle state between JTAG sequences. The state machine sequence for changing the instruction word is shown in Figure 164.

**Figure 164.** State Machine Sequence for Changing the Instruction Word




AVR_RESET (0xC)

The AVR specific public JTAG instruction for setting the AVR device in the Reset mode or taking the device out from the Reset mode. The TAP controller is not reset by this instruction. The one bit Reset Register is selected as data register. Note that the reset will be active as long as there is a logic “one” in the Reset Chain. The output from this chain is not latched.

The active states are:

- Shift-DR: The Reset Register is shifted by the TCK input.

PROG_ENABLE (0x4)

The AVR specific public JTAG instruction for enabling programming via the JTAG port. The 16-bit Programming Enable Register is selected as data register. The active states are the following:

- Shift-DR: The programming enable signature is shifted into the data register.
- Update-DR: The programming enable signature is compared to the correct value, and Programming mode is entered if the signature is valid.

PROG_COMMANDS (0x5)

The AVR specific public JTAG instruction for entering programming commands via the JTAG port. The 15-bit Programming Command Register is selected as data register. The active states are the following:

- Capture-DR: The result of the previous command is loaded into the data register.
- Shift-DR: The data register is shifted by the TCK input, shifting out the result of the previous command and shifting in the new command.
- Update-DR: The programming command is applied to the Flash inputs
- Run-Test/Idle: One clock cycle is generated, executing the applied command (not always required, see Table 136 below).

PROG_PAGELOAD (0x6)

The AVR specific public JTAG instruction to directly load the Flash data page via the JTAG port. An 8-bit Flash Data Byte Register is selected as the data register. This is physically the 8 LSBs of the Programming Command Register. The active states are the following:

- Shift-DR: The Flash Data Byte Register is shifted by the TCK input.
- Update-DR: The content of the Flash Data Byte Register is copied into a temporary register. A write sequence is initiated that within 11 TCK cycles loads the content of the temporary register into the Flash page buffer. The AVR automatically alternates between writing the low and the high byte for each new Update-DR state, starting with the low byte for the first Update-DR encountered after entering the PROG_PAGELOAD command. The Program Counter is pre-incremented before writing the low byte, except for the first written byte. This ensures that the first data is written to the address set up by PROG_COMMANDS, and loading the last location in the page buffer does not make the program counter increment into the next page.

PROG_PAGEREAD (0x7)

The AVR specific public JTAG instruction to directly capture the Flash content via the JTAG port. An 8-bit Flash Data Byte Register is selected as the data register. This is physically the 8 LSBs of the Programming Command Register. The active states are the following:

- Capture-DR: The content of the selected Flash byte is captured into the Flash Data Byte Register. The AVR automatically alternates between reading the low and the high byte for each new Capture-DR state, starting with the low byte for the first Capture-DR encountered after entering the PROG_PAGEREAD command. The Program Counter is post-incremented after reading each high byte, including the first read byte. This ensures that the first data is captured from the first address set up by PROG_COMMANDS, and reading the last location in the page makes the program counter increment into the next page.
- Shift-DR: The Flash Data Byte Register is shifted by the TCK input.

**Data Registers**

The data registers are selected by the JTAG instruction registers described in section “Programming Specific JTAG Instructions” on page 342. The data registers relevant for programming operations are:

- Reset Register
- Programming Enable Register
- Programming Command Register
- Flash Data Byte Register

Reset Register

The Reset Register is a Test Data Register used to reset the part during programming. It is required to reset the part before entering Programming mode.

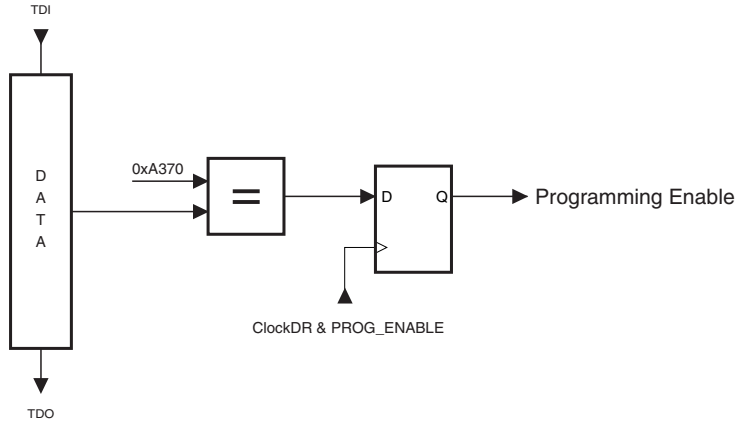
A high value in the Reset Register corresponds to pulling the external reset low. The part is reset as long as there is a high value present in the Reset Register. Depending on the Fuse settings for the clock options, the part will remain reset for a Reset Time-out

period (refer to “Clock Sources” on page 35) after releasing the Reset Register. The output from this data register is not latched, so the reset will take place immediately, as shown in Figure 144 on page 292.

Programming Enable Register

The Programming Enable Register is a 16-bit register. The contents of this register is compared to the programming enable signature, binary code 0b1010_0011_0111_0000. When the contents of the register is equal to the programming enable signature, programming via the JTAG port is enabled. The register is reset to 0 on Power-on Reset, and should always be reset when leaving Programming mode.

**Figure 165.** Programming Enable Register




Programming Command Register

The Programming Command Register is a 15-bit register. This register is used to serially shift in programming commands, and to serially shift out the result of the previous command, if any. The JTAG Programming Instruction Set is shown in Table 136. The state sequence when shifting in the programming commands is illustrated in Figure 167.

**Figure 166.** Programming Command Register

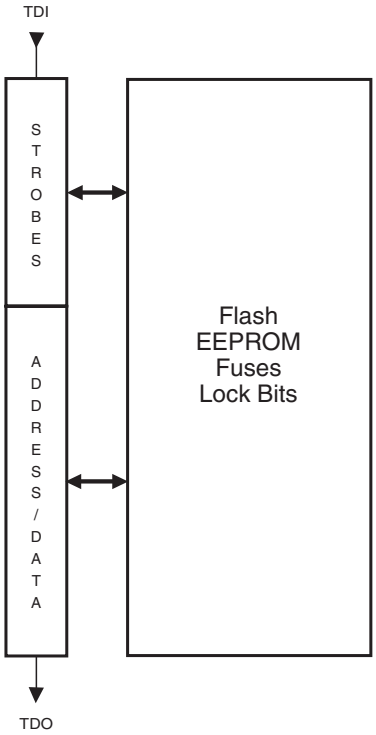


**Table 136.** JTAG Programming Instruction

Set **a** = address high bits, **b** = address low bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, x = don't care

| Instruction | TDI Sequence | TDO Sequence | Notes |
|---|---|---|---|
| 1a. Chip Erase | 0100011_10000000<br>0110001_10000000<br>0110011_10000000<br>0110011_10000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | |
| 1b. Poll for Chip Erase Complete | 0110011_10000000 | xxxxx**o**x_xxxxxxxx | (2) |
| 2a. Enter Flash Write | 0100011_00010000 | xxxxxxx_xxxxxxxx | |
| 2b. Load Address High Byte | 0000111_**aaaaaaaa** | xxxxxxx_xxxxxxxx | (9) |
| 2c. Load Address Low Byte | 0000011_**bbbbbbbb** | xxxxxxx_xxxxxxxx | |
| 2d. Load Data Low Byte | 0010011_**iiiiiiii** | xxxxxxx_xxxxxxxx | |
| 2e. Load Data High Byte | 0010111_**iiiiiiii** | xxxxxxx_xxxxxxxx | |
| 2f. Latch Data | 0110111_00000000<br>1110111_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 2g. Write Flash Page | 0110111_00000000<br>0110101_00000000<br>0110111_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 2h. Poll for Page Write Complete | 0110111_00000000 | xxxxx**o**x_xxxxxxxx | (2) |
| 3a. Enter Flash Read | 0100011_00000010 | xxxxxxx_xxxxxxxx | |
| 3b. Load Address High Byte | 0000111_**aaaaaaaa** | xxxxxxx_xxxxxxxx | (9) |
| 3c. Load Address Low Byte | 0000011_**bbbbbbbb** | xxxxxxx_xxxxxxxx | |
| 3d. Read Data Low and High Byte | 0110010_00000000<br>0110110_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo**<br>xxxxxxx_**oooooooo** | <br>Low byte<br>High byte |
| 4a. Enter EEPROM Write | 0100011_00010001 | xxxxxxx_xxxxxxxx | |
| 4b. Load Address High Byte | 0000111_**aaaaaaaa** | xxxxxxx_xxxxxxxx | (9) |
| 4c. Load Address Low Byte | 0000011_**bbbbbbbb** | xxxxxxx_xxxxxxxx | |
| 4d. Load Data Byte | 0010011_**iiiiiiii** | xxxxxxx_xxxxxxxx | |
| 4e. Latch Data | 0110111_00000000<br>1110111_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 4f. Write EEPROM Page | 0110011_00000000<br>0110001_00000000<br>0110011_00000000<br>0110011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 4g. Poll for Page Write Complete | 0110011_00000000 | xxxxx**o**x_xxxxxxxx | (2) |
| 5a. Enter EEPROM Read | 0100011_00000011 | xxxxxxx_xxxxxxxx | |
| 5b. Load Address High Byte | 0000111_**aaaaaaaa** | xxxxxxx_xxxxxxxx | (9) |

**Table 136.** JTAG Programming Instruction (Continued)
Set (Continued) **a** = address high bits, **b** = address low bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, x = don't care

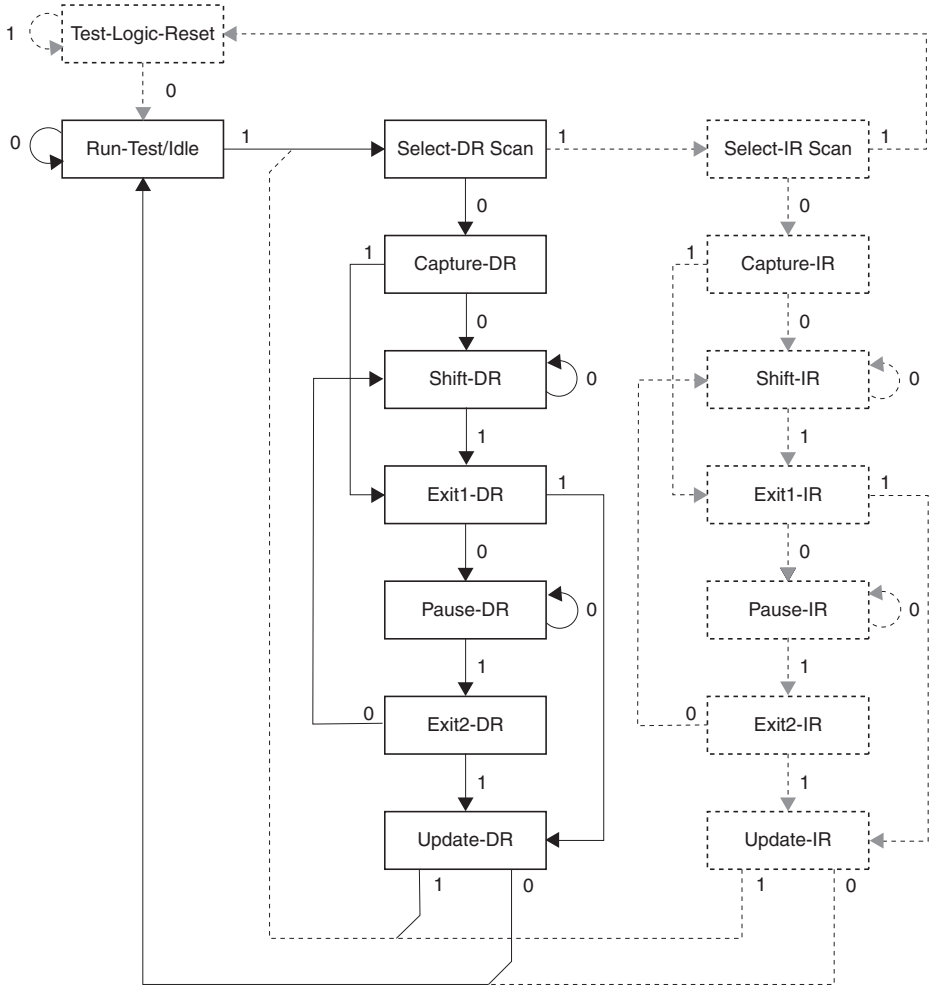
| Instruction | TDI Sequence | TDO Sequence | Notes |
|---|---|---|---|
| 5c. Load Address Low Byte | 0000011_**bbbbbbbb** | xxxxxxx_xxxxxxxx | |
| 5d. Read Data Byte | 0110011_**bbbbbbbb**<br>0110010_00000000<br>0110011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo** | |
| 6a. Enter Fuse Write | 0100011_01000000 | xxxxxxx_xxxxxxxx | |
| 6b. Load Data Low Byte(6) | 0010011_**iiiiiiii** | xxxxxxx_xxxxxxxx | (3) |
| 6c. Write Fuse Extended Byte | 0111011_00000000<br>0111001_00000000<br>0111011_00000000<br>0111011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 6d. Poll for Fuse Write Complete | 0110111_00000000 | xxxxx**o**x_xxxxxxxx | (2) |
| 6e. Load Data Low Byte(7) | 0010011_**iiiiiiii** | xxxxxxx_xxxxxxxx | (3) |
| 6f. Write Fuse High Byte | 0110111_00000000<br>0110101_00000000<br>0110111_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 6g. Poll for Fuse Write Complete | 0110111_00000000 | xxxxx**o**x_xxxxxxxx | (2) |
| 6h. Load Data Low Byte(7) | 0010011_**iiiiiiii** | xxxxxxx_xxxxxxxx | (3) |
| 6i. Write Fuse Low Byte | 0110011_00000000<br>0110001_00000000<br>0110011_00000000<br>0110011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 6j. Poll for Fuse Write Complete | 0110011_00000000 | xxxxx**o**x_xxxxxxxx | (2) |
| 7a. Enter Lock Bit Write | 0100011_00100000 | xxxxxxx_xxxxxxxx | |
| 7b. Load Data Byte(9) | 0010011_11**iiiiii** | xxxxxxx_xxxxxxxx | (4) |
| 7c. Write Lock Bits | 0110011_00000000<br>0110001_00000000<br>0110011_00000000<br>0110011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 7d. Poll for Lock Bit Write complete | 0110011_00000000 | xxxxx**o**x_xxxxxxxx | (2) |
| 8a. Enter Fuse/Lock Bit Read | 0100011_00000100 | xxxxxxx_xxxxxxxx | |
| 8b. Read Extended Fuse Byte(6) | 0111010_00000000<br>0111011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo** | |
| 8c. Read Fuse High Byte(7) | 0111110_00000000<br>0111111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo** | |
| 8d. Read Fuse Low Byte(8) | 0110010_00000000<br>0110011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo** | |
| 8e. Read Lock Bits(9) | 0110110_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xx**oooooo** | (5) |

**Table 136.** JTAG Programming Instruction (Continued)
Set (Continued) **a** = address high bits, **b** = address low bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, x = don't care

| Instruction | TDI Sequence | TDO Sequence | Notes |
|---|---|---|---|
| 8f. Read Fuses and Lock Bits | 0111010_00000000<br>0111110_00000000<br>0110010_00000000<br>0110110_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo**<br>xxxxxxx_**oooooooo**<br>xxxxxxx_**oooooooo**<br>xxxxxxx_**oooooooo** | (5)<br>Fuse Ext. byte<br>Fuse High byte<br>Fuse Low byte<br>Lock bits |
| 9a. Enter Signature Byte Read | 0100011_00001000 | xxxxxxx_xxxxxxxx | |
| 9b. Load Address Byte | 0000011_**bbbbbbbb** | xxxxxxx_xxxxxxxx | |
| 9c. Read Signature Byte | 0110010_00000000<br>0110011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo** | |
| 10a. Enter Calibration Byte Read | 0100011_00001000 | xxxxxxx_xxxxxxxx | |
| 10b. Load Address Byte | 0000011_**bbbbbbbb** | xxxxxxx_xxxxxxxx | |
| 10c. Read Calibration Byte | 0110110_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo** | |
| 11a. Load No Operation Command | 0100011_00000000<br>0110011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | |

Notes:
1. This command sequence is not required if the seven MSB are correctly set by the previous command sequence (which is normally the case).
2. Repeat until **o** = "1".
3. Set bits to "0" to program the corresponding Fuse, "1" to unprogram the Fuse.
4. Set bits to "0" to program the corresponding Lock bit, "1" to leave the Lock bit unchanged.
5. "0" = programmed, "1" = unprogrammed.
6. The bit mapping for Fuses Extended byte is listed in Table 124 on page 326
7. The bit mapping for Fuses High byte is listed in Table 125 on page 326
8. The bit mapping for Fuses Low byte is listed in Table 126 on page 327
9. The bit mapping for Lock bits byte is listed in Table 122 on page 325
10. Address bits exceeding PCMSB and EEAMSB (Table 131 and Table 132) are don't care
11. All TDI and TDO sequnces are represented by binary digits (0b...).

**Figure 167.** State Machine Sequence for Changing/Reading the Data Word




Flash Data Byte Register

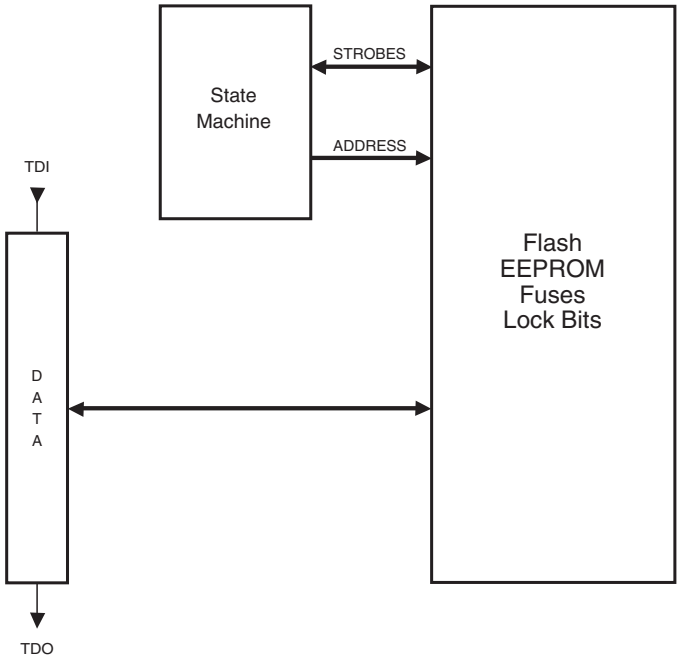
The Flash Data Byte Register provides an efficient way to load the entire Flash page buffer before executing Page Write, or to read out/verify the content of the Flash. A state machine sets up the control signals to the Flash and senses the strobe signals from the Flash, thus only the data words need to be shifted in/out.

The Flash Data Byte Register actually consists of the 8-bit scan chain and a 8-bit temporary register. During page load, the Update-DR state copies the content of the scan chain over to the temporary register and initiates a write sequence that within 11 TCK cycles loads the content of the temporary register into the Flash page buffer. The AVR automatically alternates between writing the low and the high byte for each new Update-DR state, starting with the low byte for the first Update-DR encountered after entering the PROG_PAGELOAD command. The Program Counter is pre-incremented before writing the low byte, except for the first written byte. This ensures that the first data is written to the address set up by PROG_COMMANDS, and loading the last location in the page buffer does not make the Program Counter increment into the next page.

During Page Read, the content of the selected Flash byte is captured into the Flash Data Byte Register during the Capture-DR state. The AVR automatically alternates between reading the low and the high byte for each new Capture-DR state, starting with the low byte for the first Capture-DR encountered after entering the PROG_PAGEREAD command. The Program Counter is post-incremented after reading each high byte,

including the first read byte. This ensures that the first data is captured from the first address set up by PROG_COMMANDS, and reading the last location in the page makes the program counter increment into the next page.

**Figure 168.** Flash Data Byte Register




The state machine controlling the Flash Data Byte Register is clocked by TCK. During normal operation in which eight bits are shifted for each Flash byte, the clock cycles needed to navigate through the TAP controller automatically feeds the state machine for the Flash Data Byte Register with sufficient number of clock pulses to complete its operation transparently for the user. However, if too few bits are shifted between each Update-DR state during page load, the TAP controller should stay in the Run-Test/Idle state for some TCK cycles to ensure that there are at least 11 TCK cycles between each Update-DR state.

**Programming Algorithm**

All references below of type “1a”, “1b”, and so on, refer to Table 136.

Entering Programming Mode

1. Enter JTAG instruction AVR_RESET and shift 1 in the Reset Register.
2. Enter instruction PROG_ENABLE and shift 0b1010_0011_0111_0000 in the Programming Enable Register.

Leaving Programming Mode

1. Enter JTAG instruction PROG_COMMANDS.
2. Disable all programming instructions by using no operation instruction 11a.
3. Enter instruction PROG_ENABLE and shift 0b0000_0000_0000_0000 in the programming Enable Register.
4. Enter JTAG instruction AVR_RESET and shift 0 in the Reset Register.

Performing Chip Erase

1. Enter JTAG instruction PROG_COMMANDS.
2. Start Chip Erase using programming instruction 1a.
3. Poll for Chip Erase complete using programming instruction 1b, or wait for $t_{WLRH_CE}$ (refer to Table 150 on page 372).

Programming the Flash

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash write using programming instruction 2a.
3. Load address High byte using programming instruction 2b.
4. Load address Low byte using programming instruction 2c.
5. Load data using programming instructions 2d, 2e and 2f.
6. Repeat steps 4 and 5 for all instruction words in the page.
7. Write the page using programming instruction 2g.
8. Poll for Flash write complete using programming instruction 2h, or wait for $t_{WLRH}$ (refer to ).
9. Repeat steps 3 to 7 until all data have been programmed.

A more efficient data transfer can be achieved using the PROG_PAGELOAD instruction:

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash write using programming instruction 2a.
3. Load the page address using programming instructions 2b and 2c. PCWORD (refer to Table 131 on page 330) is used to address within one page and must be written as 0.
4. Enter JTAG instruction PROG_PAGELOAD.
5. Load the entire page by shifting in all instruction words in the page byte-by-byte, starting with the LSB of the first instruction in the page and ending with the MSB of the last instruction in the page. Use Update-DR to copy the contents of the Flash Data Byte Register into the Flash page location and to auto-increment the Program Counter before each new word.
6. Enter JTAG instruction PROG_COMMANDS.
7. Write the page using programming instruction 2g.
8. Poll for Flash write complete using programming instruction 2h, or wait for $t_{WLRH}$ (refer to Table 150 on page 372).
9. Repeat steps 3 to 8 until all data have been programmed.

Reading the Flash

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash read using programming instruction 3a.
3. Load address using programming instructions 3b and 3c.
4. Read data using programming instruction 3d.
5. Repeat steps 3 and 4 until all data have been read.

A more efficient data transfer can be achieved using the PROG_PAGEREAD instruction:

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash read using programming instruction 3a.
3. Load the page address using programming instructions 3b and 3c. PCWORD (refer to Table 131 on page 330) is used to address within one page and must be written as 0.
4. Enter JTAG instruction PROG_PAGEREAD.
5. Read the entire page (or Flash) by shifting out all instruction words in the page (or Flash), starting with the LSB of the first instruction in the page (Flash) and ending with the MSB of the last instruction in the page (Flash). The Capture-DR state both captures the data from the Flash, and also auto-increments the pro-

gram counter after each word is read. Note that Capture-DR comes before the shift-DR state. Hence, the first byte which is shifted out contains valid data.

6. Enter JTAG instruction PROG_COMMANDS.
7. Repeat steps 3 to 6 until all data have been read.

Programming the EEPROM

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable EEPROM write using programming instruction 4a.
3. Load address High byte using programming instruction 4b.
4. Load address Low byte using programming instruction 4c.
5. Load data using programming instructions 4d and 4e.
6. Repeat steps 4 and 5 for all data bytes in the page.
7. Write the data using programming instruction 4f.
8. Poll for EEPROM write complete using programming instruction 4g, or wait for $t_{WLRH}$ (refer to Table 150 on page 372).
9. Repeat steps 3 to 8 until all data have been programmed.

Note that the PROG_PAGELOAD instruction can not be used when programming the EEPROM.

Reading the EEPROM

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable EEPROM read using programming instruction 5a.
3. Load address using programming instructions 5b and 5c.
4. Read data using programming instruction 5d.
5. Repeat steps 3 and 4 until all data have been read.

Note that the PROG_PAGEREAD instruction can not be used when reading the EEPROM.

Programming the Fuses

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Fuse write using programming instruction 6a.
3. Load data high byte using programming instructions 6b. A bit value of “0” will program the corresponding fuse, a “1” will unprogram the fuse.
4. Write Fuse High byte using programming instruction 6c.
5. Poll for Fuse write complete using programming instruction 6d, or wait for $t_{WLRH}$ (refer to Table 150 on page 372).
6. Load data low byte using programming instructions 6e. A “0” will program the fuse, a “1” will unprogram the fuse.
7. Write Fuse low byte using programming instruction 6f.
8. Poll for Fuse write complete using programming instruction 6g, or wait for $t_{WLRH}$ (refer to Table 150 on page 372).

Programming the Lock Bits

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Lock bit write using programming instruction 7a.
3. Load data using programming instructions 7b. A bit value of “0” will program the corresponding lock bit, a “1” will leave the lock bit unchanged.
4. Write Lock bits using programming instruction 7c.
5. Poll for Lock bit write complete using programming instruction 7d, or wait for $t_{WLRH}$ (refer to Table 150 on page 372).

Reading the Fuses and Lock Bits

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Fuse/Lock bit read using programming instruction 8a.
3. To read all Fuses and Lock bits, use programming instruction 8f.
   To only read Extended Fuse byte, use programming instruction 8b.
   To only read Fuse High byte, use programming instruction 8c.
   To only read Fuse Low byte, use programming instruction 8d.
   To only read Lock bits, use programming instruction 8e.

Reading the Signature Bytes

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Signature byte read using programming instruction 9a.
3. Load address 0x00 using programming instruction 9b.
4. Read first signature byte using programming instruction 9c.
5. Repeat steps 3 and 4 with address 0x01 and address 0x02 to read the second and third signature bytes, respectively.

Reading the Calibration Byte

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Calibration byte read using programming instruction 10a.
3. Load address 0x00 using programming instruction 10b.
4. Read the calibration byte using programming instruction 10c.

# Electrical Characteristics(1)

## Absolute Maximum Ratings*

| | |
|---|---|
| Industrial Operating Temperature .....– 40°C to +85°C<br><br>Storage Temperature .....– 65°C to +150°C<br><br>Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground .....– 0.5V to $V_{CC}$+0.5V<br><br>Voltage on $\overline{\text{RESET}}$ with respect to Ground....– 0.5V to +13.0V<br><br>Voltage on $V_{CC}$ with respect to Ground.....– 0.5V to 6.0V<br><br>DC Current per I/O Pin ..... 40.0 mA<br><br>DC Current $V_{CC}$ and GND Pins..... 200.0 mA | *NOTICE: Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability. |

Note: 1. Electrical Characteristics for this product have not yet been finalized. Please consider all values listed herein as preliminary and non-contractual.

## DC Characteristics

TA = -40°C to +85°C, $V_{CC}$ = 2.7V to 5.5V (unless otherwise noted)

| Symbol | Parameter | Condition | Min. | Typ. | Max. | Units |
|---|---|---|---|---|---|---|
| $V_{IL}$ | Input Low Voltage | Except XTAL1 and $\overline{\text{RESET}}$ pins | – 0.5 | | 0.2 Vcc (1) | V |
| $V_{IL1}$ | Input Low Voltage | XTAL1 pin - External Clock Selected | – 0.5 | | 0.1 Vcc (1) | V |
| $V_{IL2}$ | Input Low Voltage | $\overline{\text{RESET}}$ pin | – 0.5 | | 0.2 Vcc (1) | V |
| $V_{IH}$ | Input High Voltage | Except XTAL1 and $\overline{\text{RESET}}$ pins | 0.6 Vcc (2) | | Vcc + 0.5 | V |
| $V_{IH1}$ | Input High Voltage | XTAL1 pin - External Clock Selected | 0.7 Vcc (2) | | Vcc + 0.5 | V |
| $V_{IH2}$ | Input High Voltage | $\overline{\text{RESET}}$ pin | 0.85 Vcc (2) | | Vcc + 0.5 | V |
| $V_{OL}$ | Output Low Voltage (3) (Ports A, B, C, D, E, F, G) | $I_{OL}$ = 20 mA, $V_{CC}$ = 5V<br>$I_{OL}$ = 10 mA, $V_{CC}$ = 3V | | | 0.7<br>0.5 | V |
| $V_{OH}$ | Output High Voltage (4) (Ports A, B, C, D, E, F, G) | $I_{OH}$ = – 20 mA, $V_{CC}$ = 5V<br>$I_{OH}$ = – 10 mA, $V_{CC}$ = 3V | 4.2<br>2.4 | | | V |
| $I_{IL}$ | Input Leakage Current I/O Pin | $V_{CC}$ = 5.5V, pin low (absolute value) | | | 1.0 | µA |
| $I_{IH}$ | Input Leakage Current I/O Pin | $V_{CC}$ = 5.5V, pin high (absolute value) | | | 1.0 | µA |
| $R_{RST}$ | Reset Pull-up Resistor | | 30 | | 60 | kΩ |
| $R_{pu}$ | I/O Pin Pull-up Resistor | | 30 | | 60 | kΩ |
| $I_{CC}$ | Power Supply Current Active Mode | 8 MHz, $V_{CC}$ = 5V | | 20 | | mA |
| | | 16 MHz, $V_{CC}$ = 5V | | 32 | | mA |
| | | 4 MHz, $V_{CC}$ = 3V | | 5 | | mA |
| | | 8 MHz, $V_{CC}$ = 3V | | 9 | | mA |
| | Power Supply Current Idle Mode | 8 MHz, $V_{CC}$ = 5V | | 12 | | mA |
| | | 16 MHz, $V_{CC}$ = 5V | | 21 | | mA |
| | | 4 MHz, $V_{CC}$ = 3V | | 2 | | mA |
| | | 8 MHz, $V_{CC}$ = 3V | | 7.5 | | mA |
| | Power Supply Current Power-down Mode | WDT enabled, $V_{CC}$ = 5V | | 25 | | µA |
| | | WDT disabled, $V_{CC}$ = 5V | | 3.5 | | µA |
| | | WDT enabled, $V_{CC}$ = 3V | | 8 | | µA |
| | | WDT disabled, $V_{CC}$ = 3V | | 1 | | µA |
| $V_{ACIO}$ | Analog Comparator Input Offset Voltage | $V_{CC}$ = 5V<br>$V_{in}$ = $V_{CC}$/2 | 1.0 | 8.0 | 20 | mV |

TA = -40°C to +85°C, $V_{CC}$ = 2.7V to 5.5V (unless otherwise noted) (Continued)

| Symbol | Parameter | Condition | Min. | Typ. | Max. | Units |
|---|---|---|---|---|---|---|
| $I_{ACLK}$ | Analog Comparator Input Leakage Current | $V_{CC}$ = 5V<br>$V_{in}$ = $V_{CC}$/2 | – 50 | | 50 | nA |
| $t_{ACID}$ | Analog Comparator Propagation Delay Common Mode Vcc/2 | $V_{CC}$ = 2.7V | | 170 | | ns |
| | | $V_{CC}$ = 5.0V | | 180 | | ns |

Note:
1. “Max” means the highest value where the pin is guaranteed to be read as low
2. “Min” means the lowest value where the pin is guaranteed to be read as high
3. Although each I/O port can sink more than the test conditions (20 mA at $V_{CC}$ = 5V, 10 mA at $V_{CC}$ = 3V) under steady state conditions (non-transient), the following must be observed:
   TQFP and QFN Package:
   1] The sum of all IOL, for all ports, should not exceed 400 mA.
   2] The sum of all IOL, for ports A0 - A7, G2, C3 - C7 should not exceed 300 mA.
   3] The sum of all IOL, for ports C0 - C2, G0 - G1, D0 - D7, XTAL2 should not exceed 150 mA.
   4] The sum of all IOL, for ports B0 - B7, G3 - G4, E0 - E7 should not exceed 150 mA.
   5] The sum of all IOL, for ports F0 - F7, should not exceed 200 mA.
   If $I_{OL}$ exceeds the test condition, $V_{OL}$ may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.
4. Although each I/O port can source more than the test conditions (-20 mA at $V_{CC}$ = 5V, -10 mA at $V_{CC}$ = 3V) under steady state conditions (non-transient), the following must be observed:
   TQFP and QFN Package:
   1] The sum of all $I_{OH}$, for all ports, should not exceed -400 mA.
   2] The sum of all $I_{OH}$, for ports A0 - A7, G2, C3 - C7 should not exceed -300 mA.
   3] The sum of all $I_{OH}$, for ports C0 - C2, G0 - G1, D0 - D7, XTAL2 should not exceed 1-50 mA.
   4] The sum of all $I_{OH}$, for ports B0 - B7, G3 - G4, E0 - E7 should not exceed -150 mA.
   5] The sum of all $I_{OH}$, for ports F0 - F7, should not exceed -200 mA.
   If $I_{OH}$ exceeds the test condition, $V_{OH}$ may exceed the related specification. Pins are not guaranteed to source current greater than the listed test condition.

## External Clock Drive Characteristics

**Figure 169.** External Clock Drive Waveforms

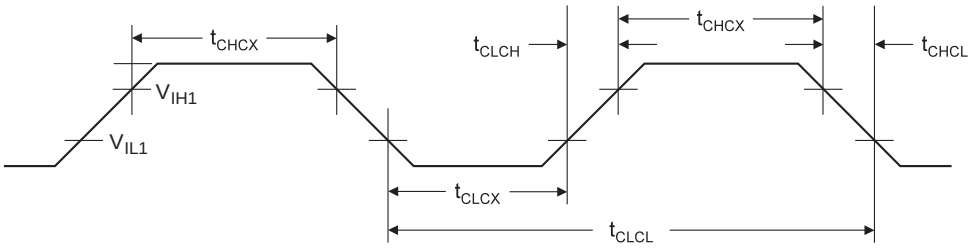



**Table 137.** External Clock Drive

| Symbol | Parameter | $V_{CC}$ = 2.7 - 5.5V | | $V_{CC}$ = 4.5 - 5.5V | | Units |
|---|---|---|---|---|---|---|
| | | Min. | Max. | Min. | Max. | |
| $1/t_{CLCL}$ | Oscillator Frequency | 0 | 8 | 0 | 16 | MHz |
| $t_{CLCL}$ | Clock Period | 125 | | 62.5 | | ns |
| $t_{CHCX}$ | High Time | 50 | | 25 | | ns |
| $t_{CLCX}$ | Low Time | 50 | | 25 | | ns |
| $t_{CLCH}$ | Rise Time | | 1.6 | | 0.5 | µs |
| $t_{CHCL}$ | Fall Time | | 1.6 | | 0.5 | µs |
| $\Delta t_{CLCL}$ | Change in period from one clock cycle to the next | | 2 | | 2 | % |

## Two-wire Serial Interface Characteristics

Table 138 describes the requirements for devices connected to the Two-wire Serial Bus. The AT90CAN128 Two-wire Serial Interface meets or exceeds these requirements under the noted conditions.

Timing symbols refer to Figure 170.

**Table 138.** Two-wire Serial Bus Requirements

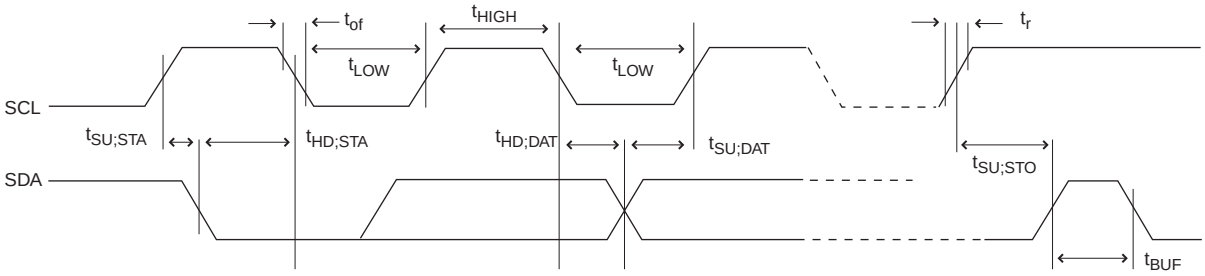
| Symbol | Parameter | Condition | Min | Max | Units |
|---|---|---|---|---|---|
| $V_{IL}$ | Input Low-voltage | | – 0.5 | 0.3 Vcc | V |
| $V_{IH}$ | Input High-voltage | | 0.7 Vcc | Vcc + 0.5 | V |
| $V_{hys}$ (1) | Hysteresis of Schmitt Trigger Inputs | | 0.05 Vcc (2) | – | V |
| $V_{OL}$ (1) | Output Low-voltage | 3 mA sink current | 0 | 0.4 | V |
| $t_r$ (1) | Rise Time for both SDA and SCL | | $20 + 0.1C_b$ (3)(2) | 300 | ns |
| $t_{of}$ (1) | Output Fall Time from $V_{IHmin}$ to $V_{ILmax}$ | 10 pF < $C_b$ < 400 pF (3) | $20 + 0.1C_b$ (3)(2) | 250 | ns |
| $t_{SP}$ (1) | Spikes Suppressed by Input Filter | | 0 | 50 (2) | ns |
| $I_i$ | Input Current each I/O Pin | 0.1 $V_{CC}$ < $V_i$ < 0.9 $V_{CC}$ | – 10 | 10 | µA |
| $C_i$ (1) | Capacitance for each I/O Pin | | – | 10 | pF |
| $f_{SCL}$ | SCL Clock Frequency | $f_{CK}$ (4) > max(16$f_{SCL}$, 250kHz) (5) | 0 | 400 | kHz |
| Rp | Value of Pull-up resistor | $f_{SCL}$ ≤ 100 kHz | $\frac{V_{CC} - 0,4V}{3mA}$ | $\frac{1000ns}{C_b}$ | Ω |
| | | $f_{SCL}$ > 100 kHz | $\frac{V_{CC} - 0,4V}{3mA}$ | $\frac{300ns}{C_b}$ | Ω |
| $t_{HD;STA}$ | Hold Time (repeated) START Condition | $f_{SCL}$ ≤ 100 kHz | 4.0 | – | µs |
| | | $f_{SCL}$ > 100 kHz | 0.6 | – | µs |
| $t_{LOW}$ | Low Period of the SCL Clock | $f_{SCL}$ ≤ 100 kHz (6) | 4.7 | – | µs |
| | | $f_{SCL}$ > 100 kHz (7) | 1.3 | – | µs |
| $t_{HIGH}$ | High period of the SCL clock | $f_{SCL}$ ≤ 100 kHz | 4.0 | – | µs |
| | | $f_{SCL}$ > 100 kHz | 0.6 | – | µs |
| $t_{SU;STA}$ | Set-up time for a repeated START condition | $f_{SCL}$ ≤ 100 kHz | 4.7 | – | µs |
| | | $f_{SCL}$ > 100 kHz | 0.6 | – | µs |
| $t_{HD;DAT}$ | Data hold time | $f_{SCL}$ ≤ 100 kHz | 0 | 3.45 | µs |
| | | $f_{SCL}$ > 100 kHz | 0 | 0.9 | µs |
| $t_{SU;DAT}$ | Data setup time | $f_{SCL}$ ≤ 100 kHz | 250 | – | ns |
| | | $f_{SCL}$ > 100 kHz | 100 | – | ns |

**Table 138.** Two-wire Serial Bus Requirements (Continued)

| Symbol | Parameter | Condition | Min | Max | Units |
|---|---|---|---|---|---|
| $t_{SU;STO}$ | Setup time for STOP condition | $f_{SCL} \leq 100$ kHz | 4.0 | – | µs |
| | | $f_{SCL} > 100$ kHz | 0.6 | – | µs |
| $t_{BUF}$ | Bus free time between a STOP and START condition | $f_{SCL} \leq 100$ kHz | 4.7 | – | µs |

Notes:
1. In AT90CAN128, this parameter is characterized and not 100% tested.
2. Required only for $f_{SCL} > 100$ kHz.
3. $C_b$ = capacitance of one bus line in pF.
4. $f_{CK}$ = CPU clock frequency
5. This requirement applies to all AT90CAN128 Two-wire Serial Interface operation. Other devices connected to the Two-wire Serial Bus need only obey the general $f_{SCL}$ requirement.
6. The actual low period generated by the AT90CAN128 Two-wire Serial Interface is ($1/f_{SCL} - 2/f_{CK}$), thus $f_{CK}$ must be greater than 6 MHz for the low time requirement to be strictly met at $f_{SCL} = 100$ kHz.
7. The actual low period generated by the AT90CAN128 Two-wire Serial Interface is ($1/f_{SCL} - 2/f_{CK}$), thus the low time requirement will not be strictly met for $f_{SCL} > 308$ kHz when $f_{CK} = 8$ MHz. Still, AT90CAN128 devices connected to the bus may communicate at full speed (400 kHz) with other AT90CAN128 devices, as well as any other device with a proper $t_{LOW}$ acceptance margin.

**Figure 170.** Two-wire Serial Bus Timing



## SPI Timing Characteristics

See Figure 171 and Figure 172 for details.

**Table 139.** SPI Timing Parameters

| | Description | Mode | Min. | Typ. | Max. | |
|---|---|---|---|---|---|---|
| 1 | SCK period | Master | | See Table 74 | | |
| 2 | SCK high/low | Master | | 50% duty cycle | | |
| 3 | Rise/Fall time | Master | | 3.6 | | |
| 4 | Setup | Master | | 10 | | |
| 5 | Hold | Master | | 10 | | |
| 6 | Out to SCK | Master | | 0.5 • $t_{sck}$ | | ns |
| 7 | SCK to out | Master | | 10 | | |
| 8 | SCK to out high | Master | | 10 | | |
| 9 | SS low to out | Slave | | 15 | | |
| 10 | SCK period | Slave | 4 • $t_{ck}$ | | | |
| 11 | SCK high/low (1) | Slave | 2 • $t_{ck}$ | | | |
| 12 | Rise/Fall time | Slave | | | 1.6 | µs |
| 13 | Setup | Slave | 10 | | | |
| 14 | Hold | Slave | $t_{ck}$ | | | |
| 15 | SCK to out | Slave | | 15 | | |
| 16 | SCK to $\overline{SS}$ high | Slave | 20 | | | ns |
| 17 | $\overline{SS}$ high to tri-state | Slave | | 10 | | |
| 18 | SS low to SCK | Slave | 2 • $t_{ck}$ | | | |

Notes: 1. In SPI Programming mode the minimum SCK high/low period is:
- 2 $t_{CLCL}$ for $f_{CK}$ < 12 MHz
- 3 $t_{CLCL}$ for $f_{CK}$ >12 MHz

**Figure 171.** SPI Interface Timing Requirements (Master Mode)

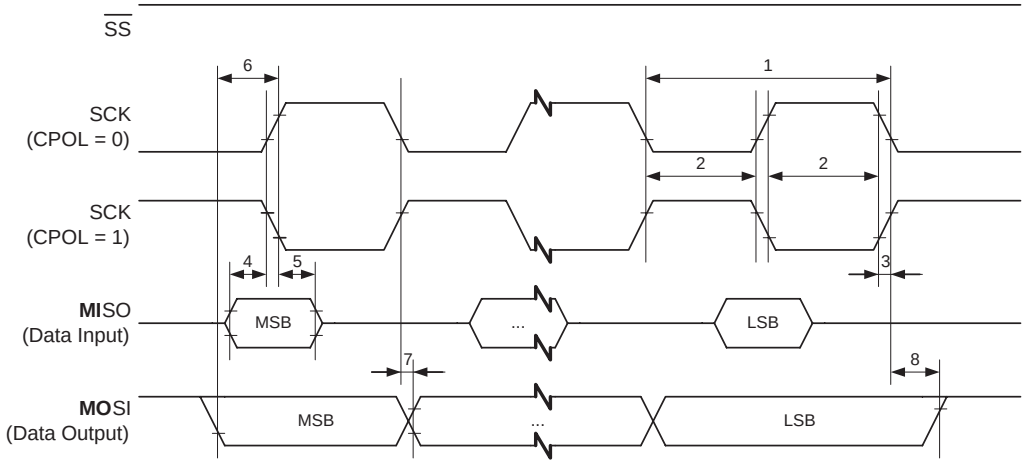
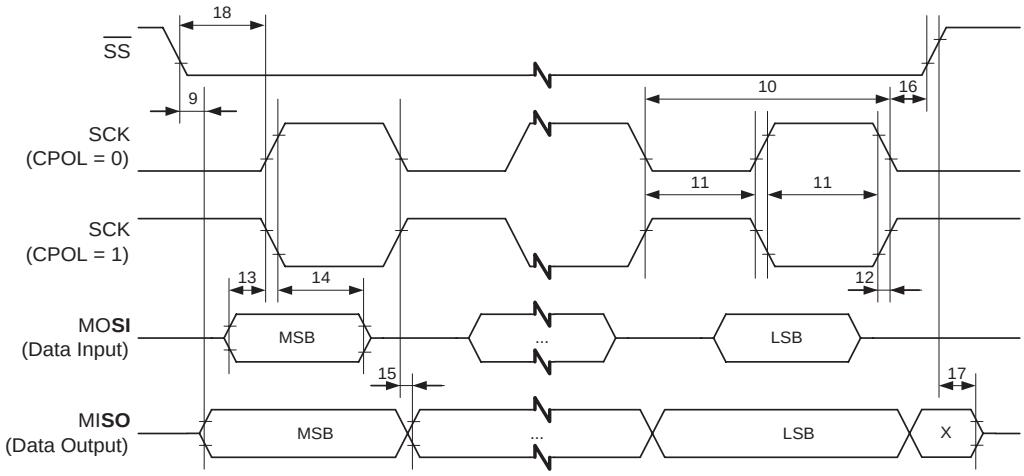


**Figure 172.** SPI Interface Timing Requirements (Slave Mode)




## CAN Physical Layer Characteristics

Only pads dedicated to the CAN communication belong to the physical layer.

**Table** : CAN Physical Layer Characteristics (1)

| | Parameter | Condition | Min. | Max. | Units |
|---|---|---|---|---|---|
| 1 | TxCAN output delay | Vcc=2.7 V<br>Load=20 pF<br>$V_{OL}/V_{OH}=V_{CC}/2$ | | 9 | ns |
| | | Vcc=4.5 V<br>Load=20 pF<br>$V_{OL}/V_{OH}=V_{CC}/2$ | | 5.3 | |
| 2 | RxCAN input delay | Vcc=2.7 V<br>$V_{IL}/V_{IH}=V_{CC}/2$ | | $9 + {}^{1}/ f_{CLK_{IO}}$(2) | |
| | | Vcc=4.5 V<br>$V_{IL}/V_{IH}=V_{CC}/2$ | | $7.2 + {}^{1}/ f_{CLK_{IO}}$(2) | |

Notes: 1. Characteristics for CAN physical layer have not yet been finalized.
2. Metastable immunity flip-flop.

## ADC Characteristics

**Table 140.** ADC Characteristics, Single Ended Channels

| Symbol | Parameter | Condition | Min(1) | Typ(1) | Max(1) | Units |
|---|---|---|---|---|---|---|
| | Resolution | Single Ended Conversion | | 10 | | Bits |
| | Absolute accuracy (Included INL, DNL, Quantization Error, Gain and Offset Error) | Single Ended Conversion<br>$V_{REF}$ = 4V, Vcc = 4V<br>ADC clock = 200 kHz | | 1.5 | | LSB |
| | | Single Ended Conversion<br>$V_{REF}$ = 4V, Vcc = 4V<br>ADC clock = 1 MHz | | | | LSB |
| | | Single Ended Conversion<br>$V_{REF}$ = 4V, Vcc = 4V<br>ADC clock = 200 kHz<br>Noise Reduction Mode | | 1.5 | | LSB |
| | | Single Ended Conversion<br>$V_{REF}$ = 4V, Vcc = 4V<br>ADC clock = 1 MHz<br>Noise Reduction Mode | | | | LSB |
| | Integral Non-linearity (INL) | Single Ended Conversion<br>$V_{REF}$ = 4V, Vcc = 4V<br>ADC clock = 200 kHz | | 0.5 | 1 | LSB |
| | Differential Non-linearity (DNL) | Single Ended Conversion<br>$V_{REF}$ = 4V, Vcc = 4V<br>ADC clock = 200 kHz | | 0.3 | 1 | LSB |
| | Gain Error | Single Ended Conversion<br>$V_{REF}$ = 4V, Vcc = 4V<br>ADC clock = 200 kHz | – 2 | 0 | + 2 | LSB |
| | Offset Error | Single Ended Conversion<br>$V_{REF}$ = 4V, Vcc = 4V<br>ADC clock = 200 kHz | – 2 | 1 | + 2 | LSB |
| | Clock Frequency | Free Running Conversion | 50 | | 1000 | kHz |
| | Conversion Time | Free Running Conversion | 65 | | 260 | µs |
| $AV_{CC}$ | Analog Supply Voltage | | $V_{CC}$ – 0.3 (2) | | $V_{CC}$ + 0.3 (3) | V |
| $V_{REF}$ | External Reference Voltage | | 2.0 | | $AV_{CC}$ | V |
| $V_{IN}$ | Input voltage | | GND | | $V_{REF}$ | V |
| | Input bandwidth | | | 38.5 | | kHz |
| $V_{INT}$ | Internal Voltage Reference | | 2.4 | 2.56 | 2.7 | V |
| $R_{REF}$ | Reference Input Resistance | | | 32 | | kΩ |
| $R_{AIN}$ | Analog Input Resistance | | | 100 | | MΩ |

Note: 1. Values are guidelines only.
2. Minimum for $AV_{CC}$ is 2.7 V.
3. Maximum for $AV_{CC}$ is 5.5 V

**Table 141.** ADC Characteristics, Differential Channels

| Symbol | Parameter | Condition | Min(1) | Typ(1) | Max(1) | Units |
|---|---|---|---|---|---|---|
| | Resolution | Differential Conversion<br>Gain = 1x or 10x | | 8 | | Bits |
| | | Differential Conversion<br>Gain = 200x | | 7 | | Bits |
| | Absolute accuracy | Gain = 1x , 10x or 200x<br>$V_{REF}$ = 4V, Vcc = 5V<br>ADC clock = 50 - 200 kHz | | 1 | | LSB |
| | Integral Non-linearity (INL)<br>(Accuracy after Calibration for Offset and Gain Error) | Gain = 1x , 10x or 200x<br>$V_{REF}$ = 4V, Vcc = 5V<br>ADC clock = 50 - 200 kHz | | 0.5 | 1 | LSB |
| | Gain Error | Gain = 1x , 10x or 200x | – 2 | 0 | + 2 | LSB |
| | Offset Error | Gain = 1x , 10x or 200x<br>$V_{REF}$ = 4V, Vcc = 5V<br>ADC clock = 50 - 200 kHz | – 1 | 0 | + 1 | LSB |
| | Clock Frequency | Free Running Conversion | 50 | | 200 | kHz |
| | Conversion Time | Free Running Conversion | 65 | | 260 | µs |
| $AV_{CC}$ | Analog Supply Voltage | | $V_{CC}$ – 0.3 (2) | | $V_{CC}$ + 0.3 (3) | V |
| $V_{REF}$ | External Reference Voltage | Differential Conversion | 2.0 | | $AV_{CC}$ - 0.5 | V |
| $V_{IN}$ | Input voltage | Differential Conversion | 0 | | $AV_{CC}$ | V |
| $V_{DIFF}$ | Input Differential Voltage | | $-V_{REF}$/Gain | | $+V_{REF}$/Gain | V |
| | ADC Convertion Output | | –511 | | 511 | LSB |
| | Input bandwidth | Differential Conversion | | 4 | | kHz |
| $V_{INT}$ | Internal Voltage Reference | | 2.4 | 2.56 | 2.7 | V |
| $R_{REF}$ | Reference Input Resistance | | | 32 | | kΩ |
| $R_{AIN}$ | Analog Input Resistance | | | 100 | | MΩ |

Note: 1. Values are guidelines only.
2. Minimum for $AV_{CC}$ is 2.7 V.
3. Maximum for $AV_{CC}$ is 5.5 V

## External Data Memory Characteristics

**Table 142.** External Data Memory Characteristics, $V_{CC}$ = 4.5 - 5.5 Volts, No Wait-state

| | | | 8 MHz Oscillator | | Variable Oscillator | | |
|---|---|---|---|---|---|---|---|
| | **Symbol** | **Parameter** | **Min.** | **Max.** | **Min.** | **Max.** | **Unit** |
| 0 | $1/t_{CLCL}$ | Oscillator Frequency | | | 0.0 | 16 | MHz |
| 1 | $t_{LHLL}$ | ALE Pulse Width | 115 | | $1.0\ t_{CLCL} - 10$ | | ns |
| 2 | $t_{AVLL}$ | Address Valid A to ALE Low | 57.5 | | $0.5\ t_{CLCL} - 5$ (1) | | ns |
| 3a | $t_{LLAX_ST}$ | Address Hold After ALE Low, write access | 5 | | 5 | | ns |
| 3b | $t_{LLAX_LD}$ | Address Hold after ALE Low, read access | 5 | | 5 | | ns |
| 4 | $t_{AVLLC}$ | Address Valid C to ALE Low | 57.5 | | $0.5\ t_{CLCL} - 5$ (1) | | ns |
| 5 | $t_{AVRL}$ | Address Valid to RD Low | 115 | | $1.0\ t_{CLCL} - 10$ | | ns |
| 6 | $t_{AVWL}$ | Address Valid to WR Low | 115 | | $1.0\ t_{CLCL} - 10$ | | ns |
| 7 | $t_{LLWL}$ | ALE Low to WR Low | 47.5 | 67.5 | $0.5\ t_{CLCL} - 15$ (2) | $0.5\ t_{CLCL} + 5$ (2) | ns |
| 8 | $t_{LLRL}$ | ALE Low to RD Low | 47.5 | 67.5 | $0.5\ t_{CLCL} - 15$ (2) | $0.5\ t_{CLCL} + 5$ (2) | ns |
| 9 | $t_{DVRH}$ | Data Setup to RD High | 40 | | 40 | | ns |
| 10 | $t_{RLDV}$ | Read Low to Data Valid | | 75 | | $1.0\ t_{CLCL} - 50$ | ns |
| 11 | $t_{RHDX}$ | Data Hold After RD High | 0 | | 0 | | ns |
| 12 | $t_{RLRH}$ | RD Pulse Width | 115 | | $1.0\ t_{CLCL} - 10$ | | ns |
| 13 | $t_{DVWL}$ | Data Setup to WR Low | 42.5 | | $0.5\ t_{CLCL} - 20$ (1) | | ns |
| 14 | $t_{WHDX}$ | Data Hold After WR High | 115 | | $1.0\ t_{CLCL} - 10$ | | ns |
| 15 | $t_{DVWH}$ | Data Valid to WR High | 125 | | $1.0\ t_{CLCL}$ | | ns |
| 16 | $t_{WLWH}$ | WR Pulse Width | 115 | | $1.0\ t_{CLCL} - 10$ | | ns |

Notes: 1. This assumes 50% clock duty cycle. The half period is actually the high time of the external clock, XTAL1.
2. This assumes 50% clock duty cycle. The half period is actually the low time of the external clock, XTAL1.

**Table 143.** External Data Memory Characteristics, $V_{CC}$ = 4.5 - 5.5 Volts, 1 Cycle Wait-state

| | | | 8 MHz Oscillator | | Variable Oscillator | | |
|---|---|---|---|---|---|---|---|
| | **Symbol** | **Parameter** | **Min.** | **Max.** | **Min.** | **Max.** | **Unit** |
| 0 | $1/t_{CLCL}$ | Oscillator Frequency | | | 0.0 | 16 | MHz |
| 10 | $t_{RLDV}$ | Read Low to Data Valid | | 200 | | $2.0\ t_{CLCL} - 50$ | ns |
| 12 | $t_{RLRH}$ | RD Pulse Width | 240 | | $2.0\ t_{CLCL} - 10$ | | ns |
| 15 | $t_{DVWH}$ | Data Valid to WR High | 240 | | $2.0\ t_{CLCL}$ | | ns |
| 16 | $t_{WLWH}$ | WR Pulse Width | 240 | | $2.0\ t_{CLCL} - 10$ | | ns |

**Table 144.** External Data Memory Characteristics, $V_{CC}$ = 4.5 - 5.5 Volts, SRWn1 = 1, SRWn0 = 0

| | | | 8 MHz Oscillator | | Variable Oscillator | | |
|---|---|---|---|---|---|---|---|
| | **Symbol** | **Parameter** | **Min.** | **Max.** | **Min.** | **Max.** | **Unit** |
| 0 | $1/t_{CLCL}$ | Oscillator Frequency | | | 0.0 | 16 | MHz |
| 10 | $t_{RLDV}$ | Read Low to Data Valid | | 325 | | $3.0\ t_{CLCL} - 50$ | ns |
| 12 | $t_{RLRH}$ | RD Pulse Width | 365 | | $3.0\ t_{CLCL} - 10$ | | ns |
| 15 | $t_{DVWH}$ | Data Valid to WR High | 375 | | $3.0\ t_{CLCL}$ | | ns |
| 16 | $t_{WLWH}$ | WR Pulse Width | 365 | | $3.0\ t_{CLCL} - 10$ | | ns |

**Table 145.** External Data Memory Characteristics, $V_{CC}$ = 4.5 - 5.5 Volts, SRWn1 = 1, SRWn0 = 0

| | | | 8 MHz Oscillator | | Variable Oscillator | | |
|---|---|---|---|---|---|---|---|
| | **Symbol** | **Parameter** | **Min.** | **Max.** | **Min.** | **Max.** | **Unit** |
| 0 | $1/t_{CLCL}$ | Oscillator Frequency | | | 0.0 | 16 | MHz |
| 10 | $t_{RLDV}$ | Read Low to Data Valid | | 200 | | $3.0\ t_{CLCL} - 50$ | ns |
| 12 | $t_{RLRH}$ | RD Pulse Width | 365 | | $3.0\ t_{CLCL} - 10$ | | ns |
| 14 | $t_{WHDX}$ | Data Hold After WR High | 240 | | $2.0\ t_{CLCL} - 10$ | | ns |
| 15 | $t_{DVWH}$ | Data Valid to WR High | 375 | | $3.0\ t_{CLCL}$ | | ns |
| 16 | $t_{WLWH}$ | WR Pulse Width | 365 | | $3.0\ t_{CLCL} - 10$ | | ns |

**Table 146.** External Data Memory Characteristics, $V_{CC}$ = 2.7 - 5.5 Volts, No Wait-state

| | | | 4 MHz Oscillator | | Variable Oscillator | | |
|---|---|---|---|---|---|---|---|
| | **Symbol** | **Parameter** | **Min.** | **Max.** | **Min.** | **Max.** | **Unit** |
| 0 | $1/t_{CLCL}$ | Oscillator Frequency | | | 0.0 | 16 | MHz |
| 1 | $t_{LHLL}$ | ALE Pulse Width | 235 | | $t_{CLCL}$ – 15 | | ns |
| 2 | $t_{AVLL}$ | Address Valid A to ALE Low | 115 | | 0.5 $t_{CLCL}$ – 10 (1) | | ns |
| 3a | $t_{LLAX_ST}$ | Address Hold After ALE Low, write access | 5 | | 5 | | ns |
| 3b | $t_{LLAX_LD}$ | Address Hold after ALE Low, read access | 5 | | 5 | | ns |
| 4 | $t_{AVLLC}$ | Address Valid C to ALE Low | 115 | | 0.5 $t_{CLCL}$ – 10 (1) | | ns |
| 5 | $t_{AVRL}$ | Address Valid to RD Low | 235 | | 1.0 $t_{CLCL}$ – 15 | | ns |
| 6 | $t_{AVWL}$ | Address Valid to WR Low | 235 | | 1.0 $t_{CLCL}$ – 15 | | ns |
| 7 | $t_{LLWL}$ | ALE Low to WR Low | 115 | 130 | 0.5 $t_{CLCL}$ – 10 (2) | 0.5 $t_{CLCL}$ + 5 (2) | ns |
| 8 | $t_{LLRL}$ | ALE Low to RD Low | 115 | 130 | 0.5 $t_{CLCL}$ – 10 (2) | 0.5 $t_{CLCL}$ + 5 (2) | ns |
| 9 | $t_{DVRH}$ | Data Setup to RD High | 45 | | 45 | | ns |
| 10 | $t_{RLDV}$ | Read Low to Data Valid | | 190 | | 1.0 $t_{CLCL}$ – 60 | ns |
| 11 | $t_{RHDX}$ | Data Hold After RD High | 0 | | 0 | | ns |
| 12 | $t_{RLRH}$ | RD Pulse Width | 235 | | 1.0 $t_{CLCL}$ – 15 | | ns |
| 13 | $t_{DVWL}$ | Data Setup to WR Low | 105 | | 0.5 $t_{CLCL}$ – 20 (1) | | ns |
| 14 | $t_{WHDX}$ | Data Hold After WR High | 235 | | 1.0 $t_{CLCL}$ – 15 | | ns |
| 15 | $t_{DVWH}$ | Data Valid to WR High | 250 | | 1.0 $t_{CLCL}$ | | ns |
| 16 | $t_{WLWH}$ | WR Pulse Width | 235 | | 1.0 $t_{CLCL}$ – 15 | | ns |

Notes: 1. This assumes 50% clock duty cycle. The half period is actually the high time of the external clock, XTAL1.
2. This assumes 50% clock duty cycle. The half period is actually the low time of the external clock, XTAL1.

**Table 147.** External Data Memory Characteristics, $V_{CC}$ = 2.7 - 5.5 Volts, SRWn1 = 0, SRWn0 = 1

| | | | 4 MHz Oscillator | | Variable Oscillator | | |
|---|---|---|---|---|---|---|---|
| | **Symbol** | **Parameter** | **Min.** | **Max.** | **Min.** | **Max.** | **Unit** |
| 0 | $1/t_{CLCL}$ | Oscillator Frequency | | | 0.0 | 8 | MHz |
| 10 | $t_{RLDV}$ | Read Low to Data Valid | | 440 | | $2.0\ t_{CLCL} - 60$ | ns |
| 12 | $t_{RLRH}$ | RD Pulse Width | 485 | | $2.0\ t_{CLCL} - 15$ | | ns |
| 15 | $t_{DVWH}$ | Data Valid to WR High | 500 | | $2.0\ t_{CLCL}$ | | ns |
| 16 | $t_{WLWH}$ | WR Pulse Width | 485 | | $2.0\ t_{CLCL} - 15$ | | ns |

**Table 148.** External Data Memory Characteristics, $V_{CC}$ = 2.7 - 5.5 Volts, SRWn1 = 1, SRWn0 = 0

| | | | 4 MHz Oscillator | | Variable Oscillator | | |
|---|---|---|---|---|---|---|---|
| | **Symbol** | **Parameter** | **Min.** | **Max.** | **Min.** | **Max.** | **Unit** |
| 0 | $1/t_{CLCL}$ | Oscillator Frequency | | | 0.0 | 8 | MHz |
| 10 | $t_{RLDV}$ | Read Low to Data Valid | | 690 | | $3.0\ t_{CLCL} - 60$ | ns |
| 12 | $t_{RLRH}$ | RD Pulse Width | 735 | | $3.0\ t_{CLCL} - 15$ | | ns |
| 15 | $t_{DVWH}$ | Data Valid to WR High | 750 | | $3.0\ t_{CLCL}$ | | ns |
| 16 | $t_{WLWH}$ | WR Pulse Width | 735 | | $3.0\ t_{CLCL} - 15$ | | ns |

**Table 149.** External Data Memory Characteristics, $V_{CC}$ = 2.7 - 5.5 Volts, SRWn1 = 1, SRWn0 = 1

| | | | 4 MHz Oscillator | | Variable Oscillator | | |
|---|---|---|---|---|---|---|---|
| | **Symbol** | **Parameter** | **Min.** | **Max.** | **Min.** | **Max.** | **Unit** |
| 0 | $1/t_{CLCL}$ | Oscillator Frequency | | | 0.0 | 8 | MHz |
| 10 | $t_{RLDV}$ | Read Low to Data Valid | | 690 | | $3.0\ t_{CLCL} - 60$ | ns |
| 12 | $t_{RLRH}$ | RD Pulse Width | 735 | | $3.0\ t_{CLCL} - 15$ | | ns |
| 14 | $t_{WHDX}$ | Data Hold After WR High | 485 | | $2.0\ t_{CLCL} - 15$ | | ns |
| 15 | $t_{DVWH}$ | Data Valid to WR High | 750 | | $3.0\ t_{CLCL}$ | | ns |
| 16 | $t_{WLWH}$ | WR Pulse Width | 735 | | $3.0\ t_{CLCL} - 15$ | | ns |

**Figure 173.** External Memory Timing (SRWn1 = 0, SRWn0 = 0)

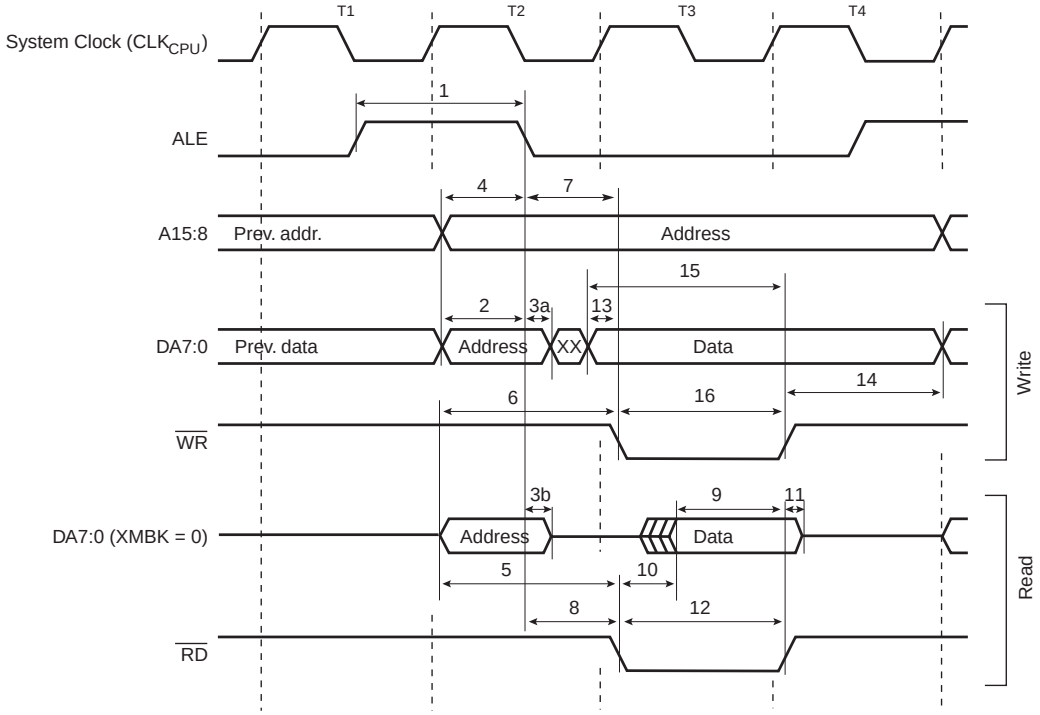



**Figure 174.** External Memory Timing (SRWn1 = 0, SRWn0 = 1)

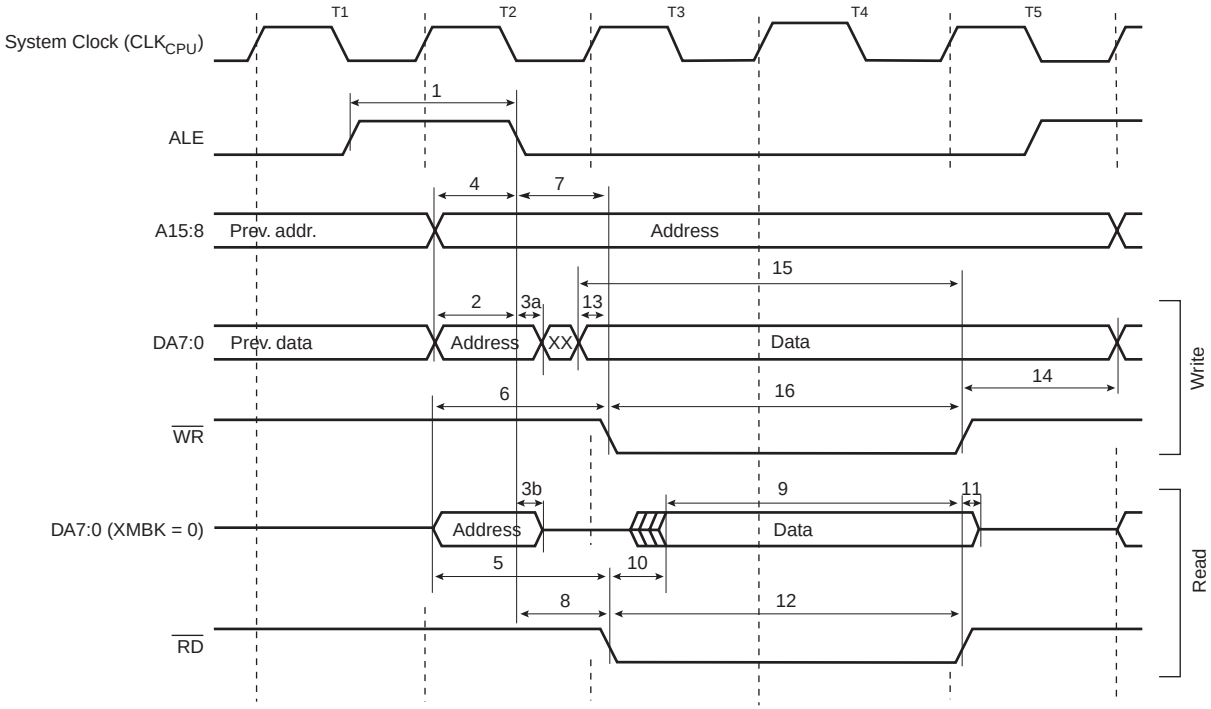


**Figure 175.** External Memory Timing (SRWn1 = 1, SRWn0 = 0)

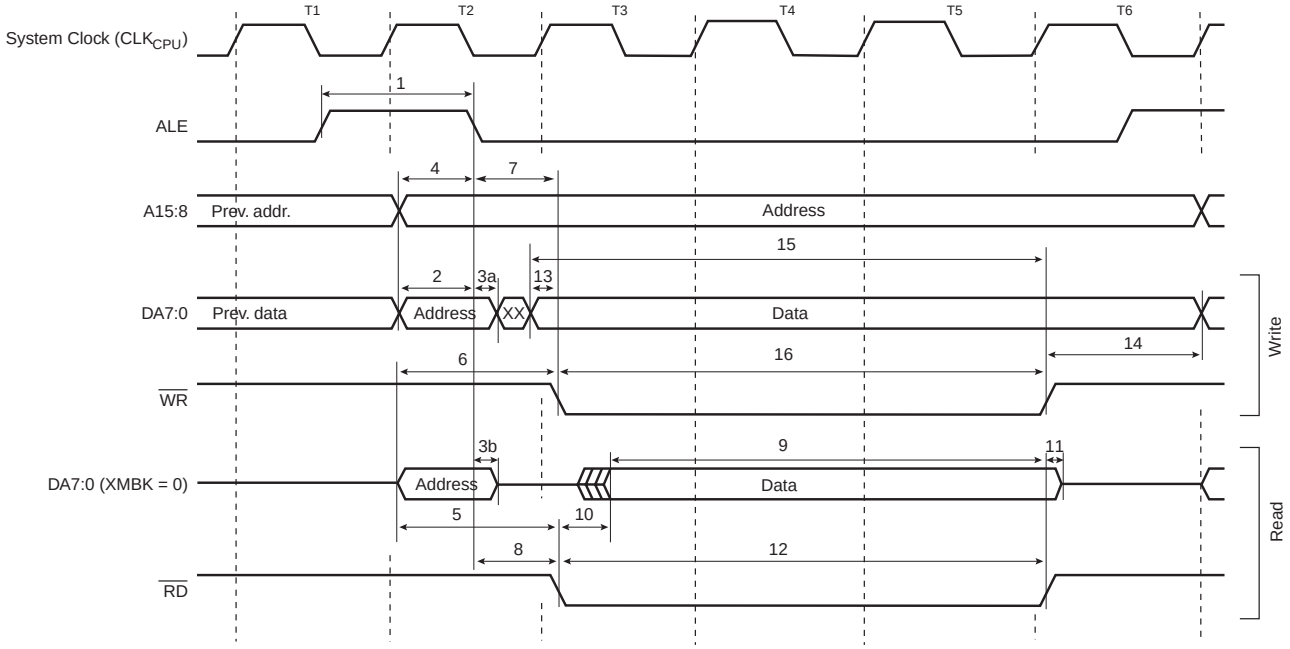
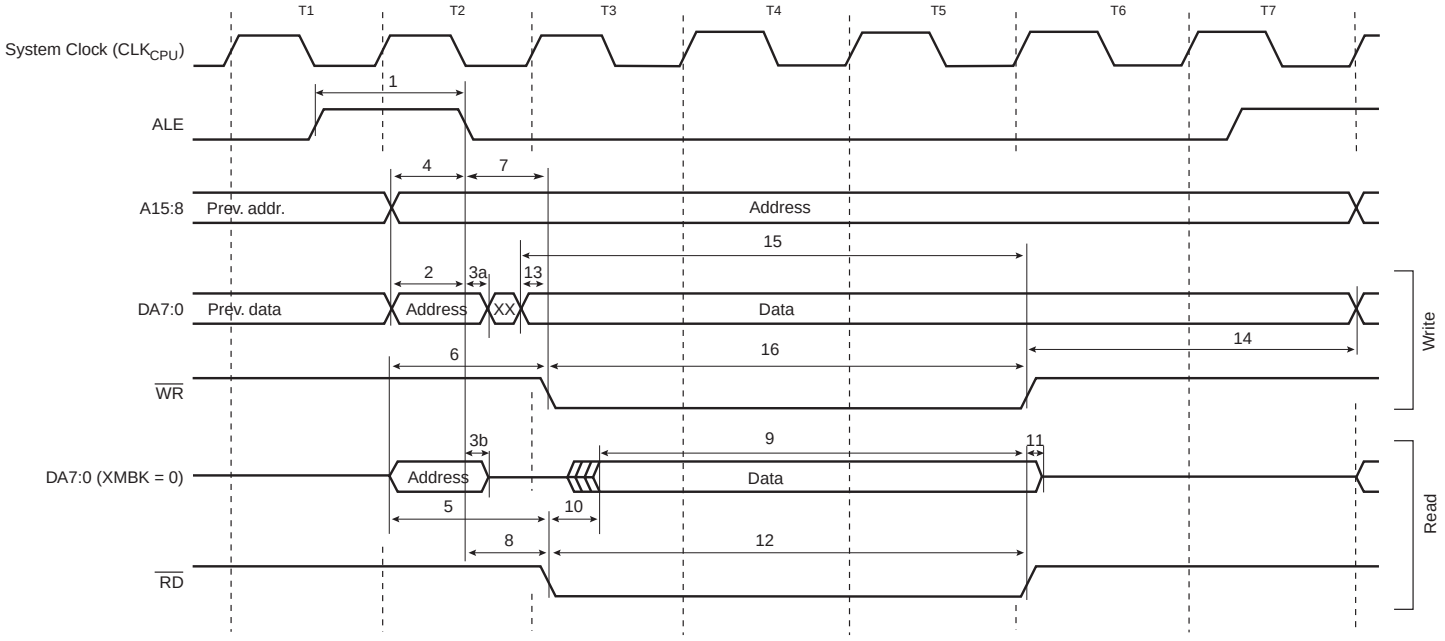



**Figure 176.** External Memory Timing (SRWn1 = 1, SRWn0 = 1)[1]




Note: 1. The ALE pulse in the last period (T4-T7) is only present if the next instruction accesses the RAM (internal or external).

## Parallel Programming Characteristics

**Figure 177.** Parallel Programming Timing, Including some General Timing Requirements

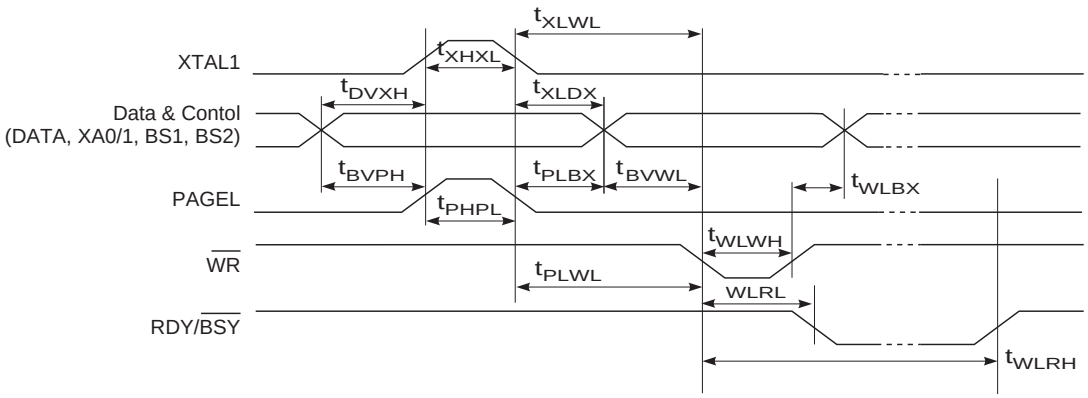
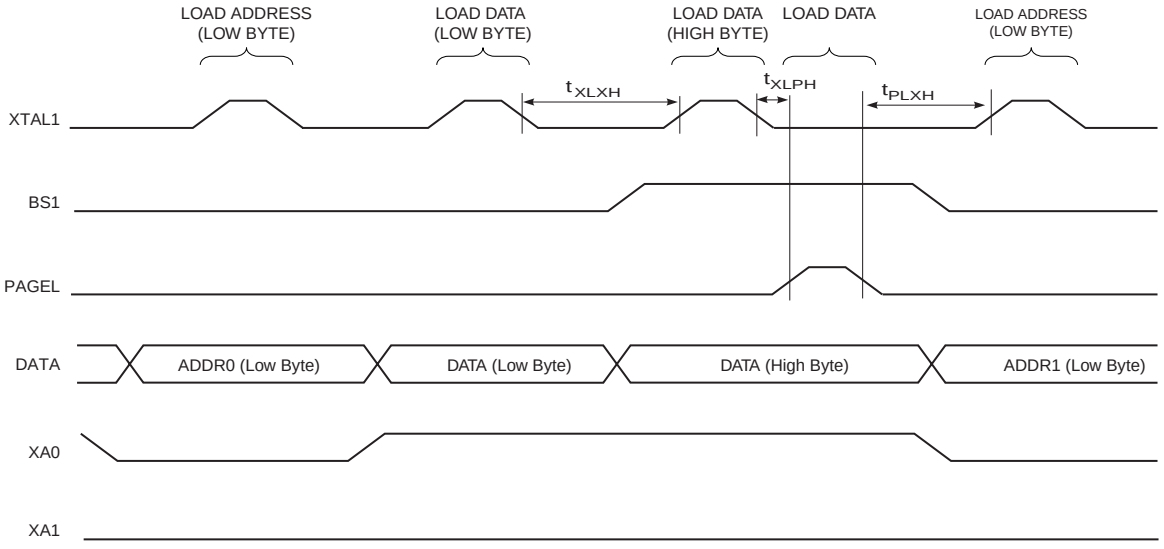
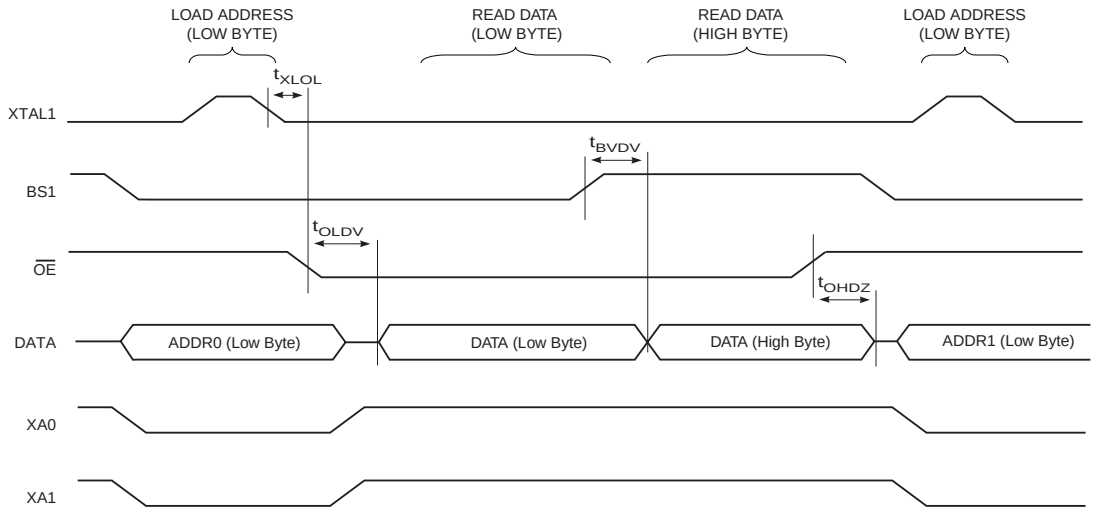



**Figure 178.** Parallel Programming Timing, Loading Sequence with Timing Requirements(1)




Note: 1. The timing requirements shown in Figure 177 (i.e., $t_{DVXH}$, $t_{XHXL}$, and $t_{XLDX}$) also apply to loading operation.

**Figure 179.** Parallel Programming Timing, Reading Sequence (within the Same Page) with Timing Requirements(1)




Note: 1. The timing requirements shown in Figure 177 (i.e., $t_{DVXH}$, $t_{XHXL}$, and $t_{XLDX}$) also apply to reading operation.

**Table 150.** Parallel Programming Characteristics, $V_{CC}$ = 5V ± 10%

| Symbol | Parameter | Min. | Typ. | Max. | Units |
|---|---|---|---|---|---|
| $V_{PP}$ | Programming Enable Voltage | 11.5 | | 12.5 | V |
| $I_{PP}$ | Programming Enable Current | | | 250 | µA |
| $t_{DVXH}$ | Data and Control Valid before XTAL1 High | 67 | | | ns |
| $t_{XLXH}$ | XTAL1 Low to XTAL1 High | 200 | | | ns |
| $t_{XHXL}$ | XTAL1 Pulse Width High | 150 | | | ns |
| $t_{XLDX}$ | Data and Control Hold after XTAL1 Low | 67 | | | ns |
| $t_{XLWL}$ | XTAL1 Low to $\overline{WR}$ Low | 0 | | | ns |
| $t_{XLPH}$ | XTAL1 Low to PAGEL high | 0 | | | ns |
| $t_{PLXH}$ | PAGEL low to XTAL1 high | 150 | | | ns |
| $t_{BVPH}$ | BS1 Valid before PAGEL High | 67 | | | ns |
| $t_{PHPL}$ | PAGEL Pulse Width High | 150 | | | ns |
| $t_{PLBX}$ | BS1 Hold after PAGEL Low | 67 | | | ns |
| $t_{WLBX}$ | BS2/1 Hold after $\overline{WR}$ Low | 67 | | | ns |
| $t_{PLWL}$ | PAGEL Low to $\overline{WR}$ Low | 67 | | | ns |
| $t_{BVWL}$ | BS1 Valid to $\overline{WR}$ Low | 67 | | | ns |
| $t_{WLWH}$ | $\overline{WR}$ Pulse Width Low | 150 | | | ns |
| $t_{WLRL}$ | $\overline{WR}$ Low to RDY/$\overline{BSY}$ Low | 0 | | 1 | µs |
| $t_{WLRH}$ | $\overline{WR}$ Low to RDY/$\overline{BSY}$ High(1) | 3.7 | | 5 | ms |
| $t_{WLRH_CE}$ | $\overline{WR}$ Low to RDY/$\overline{BSY}$ High for Chip Erase(2) | 7.5 | | 10 | ms |
| $t_{XLOL}$ | XTAL1 Low to $\overline{OE}$ Low | 0 | | | ns |

**Table 150.** Parallel Programming Characteristics, $V_{CC}$ = 5V ± 10% (Continued)

| Symbol | Parameter | Min. | Typ. | Max. | Units |
|---|---|---|---|---|---|
| $t_{BVDV}$ | BS1 Valid to DATA valid | 0 | | 250 | ns |
| $t_{OLDV}$ | $\overline{OE}$ Low to DATA Valid | | | 250 | ns |
| $t_{OHDZ}$ | $\overline{OE}$ High to DATA Tri-stated | | | 250 | ns |

Notes: 1. $t_{WLRH}$ is valid for the Write Flash, Write EEPROM, Write Fuse bits and Write Lock bits commands.
2. $t_{WLRH_CE}$ is valid for the Chip Erase command.

## AT90CAN128 Typical Characteristics

- The following charts show typical behavior. These figures are not tested during manufacturing. All current consumption measurements are performed with all I/O pins configured as inputs and with internal pull-ups enabled. A sine wave generator with rail-to-rail output is used as clock source.
- The power consumption in Power-down mode is independent of clock selection.
- The current consumption is a function of several factors such as: operating voltage, operating frequency, loading of I/O pins, switching rate of I/O pins, code executed and ambient temperature. The dominating factors are operating voltage and frequency.
- The current drawn from capacitive loaded pins may be estimated (for one pin) as $C_L$*$V_{CC}$*f where $C_L$ = load capacitance, $V_{CC}$ = operating voltage and f = average switching frequency of I/O pin.
- The parts are characterized at frequencies higher than test limits. Parts are not guaranteed to function properly at frequencies higher than the ordering code indicates.
- The difference between current consumption in Power-down mode with Watchdog Timer enabled and Power-down mode with Watchdog Timer disabled represents the differential current drawn by the Watchdog Timer.

### Active Supply Current

**Figure 180.** Active Supply Current vs. Frequency (0.1 - 1.0 MHz)

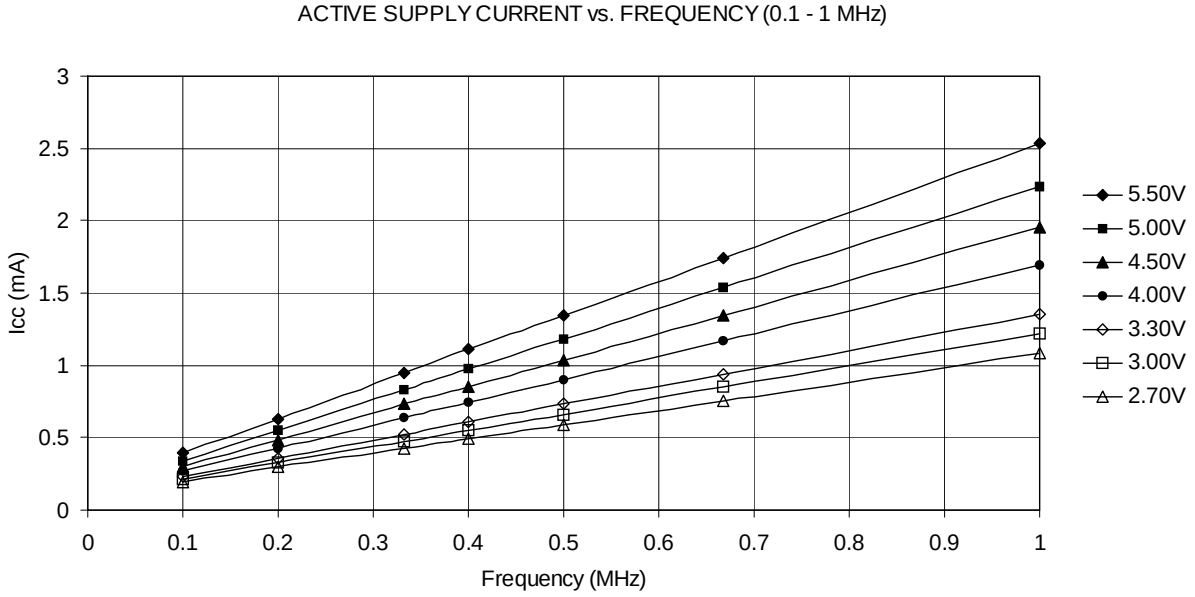


**Figure 181.** Active Supply Current vs. Frequency (1 - 16 MHz)

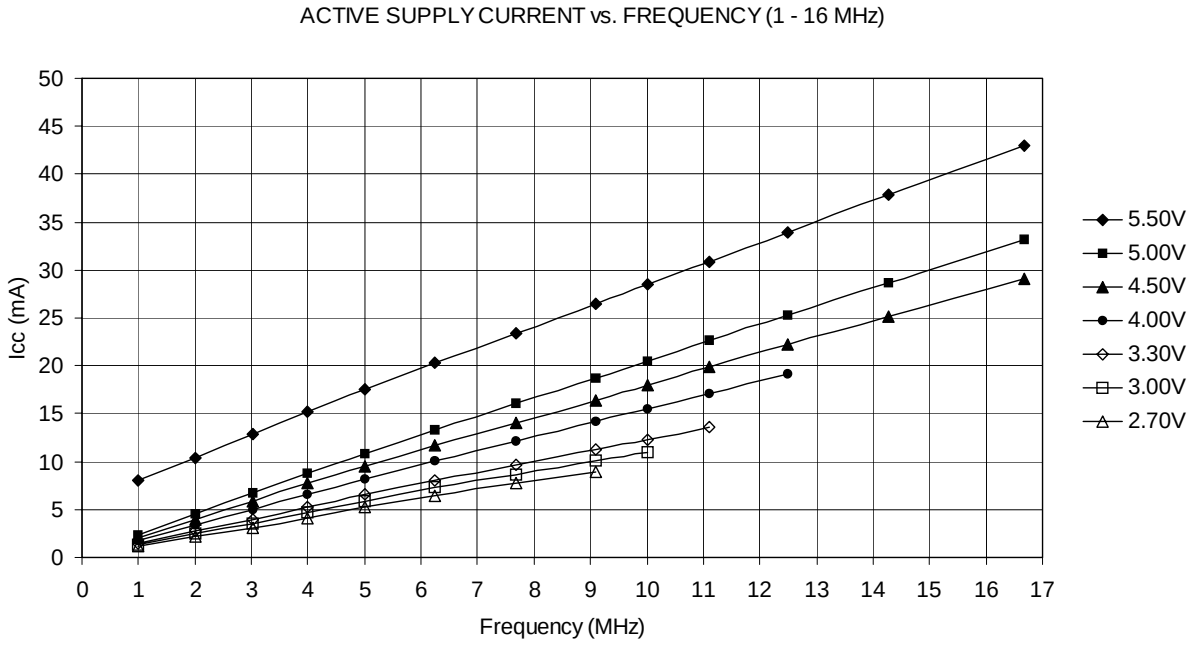



**Figure 182.** Active Supply Current vs. Vcc (Internal RC Oscillator 8 MHz)

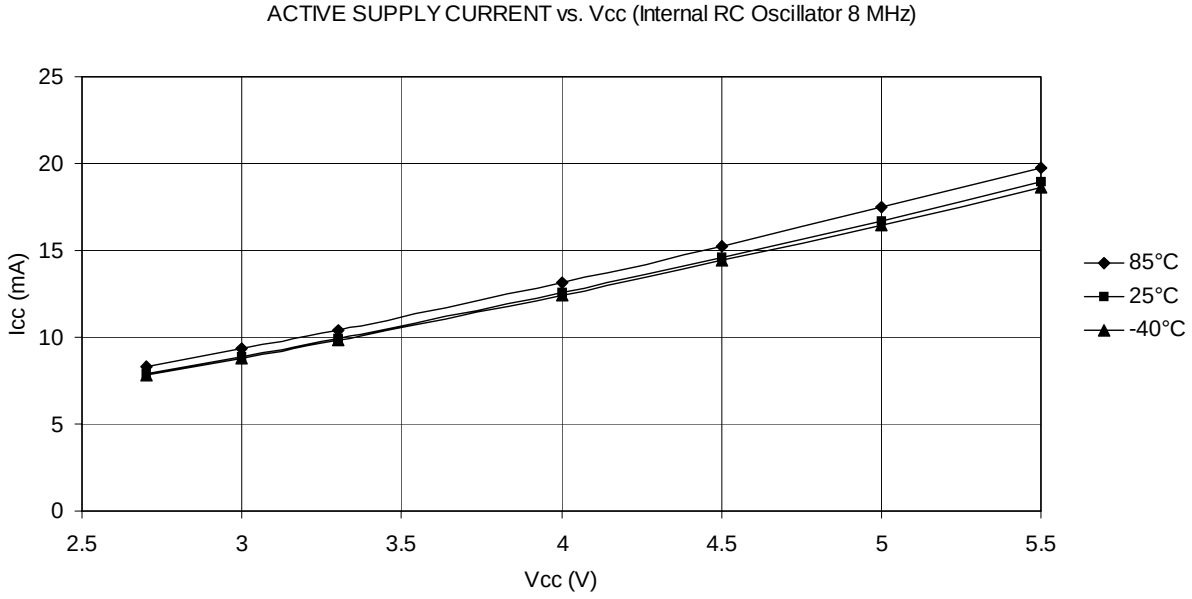


**Figure 183.** Active Supply Current vs. Vcc (Internal RC Oscillator 1 MHz)

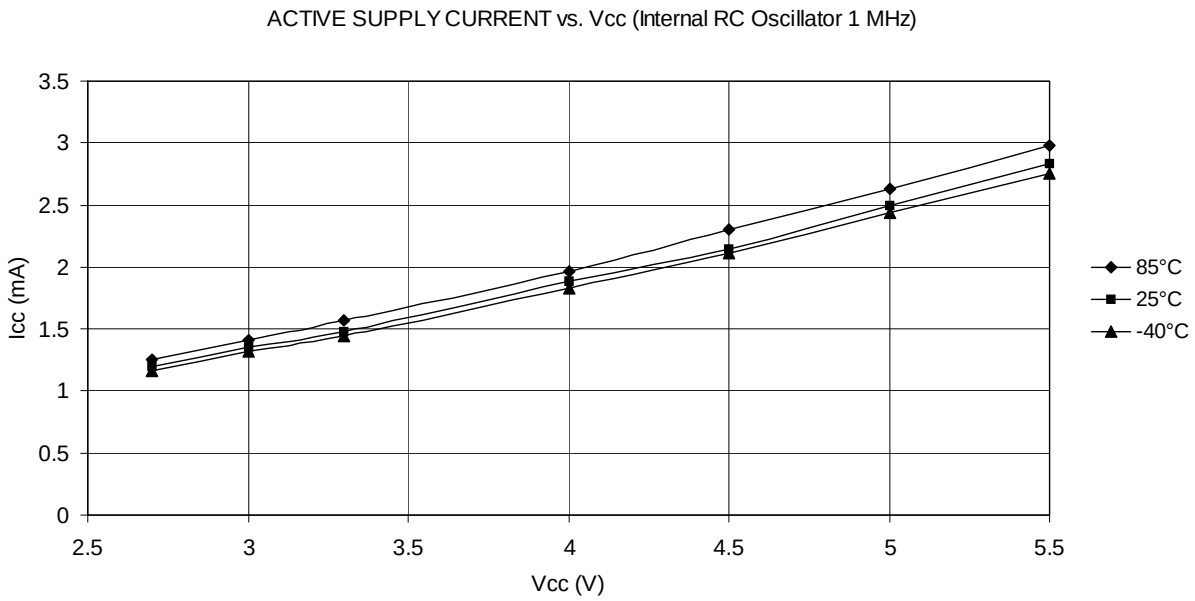
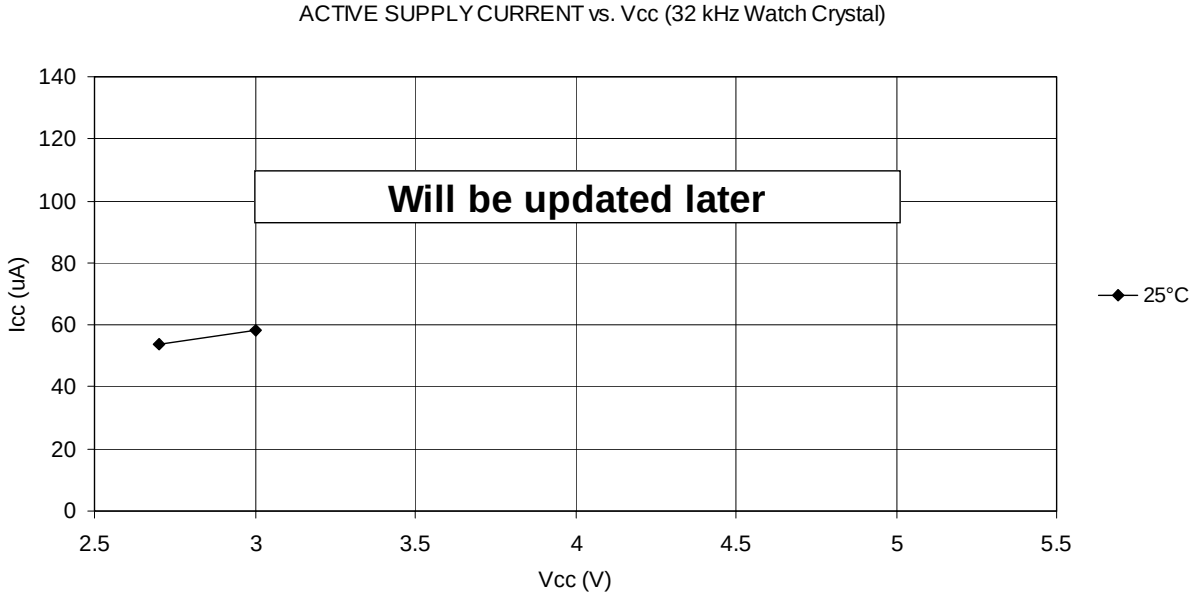



**Figure 184.** Active Supply Current vs. Vcc (32 kHz Watch Crystal)



## Idle Supply Current

**Figure 185.** Idle Supply Current vs. Frequency (0.1 - 1.0 MHz)

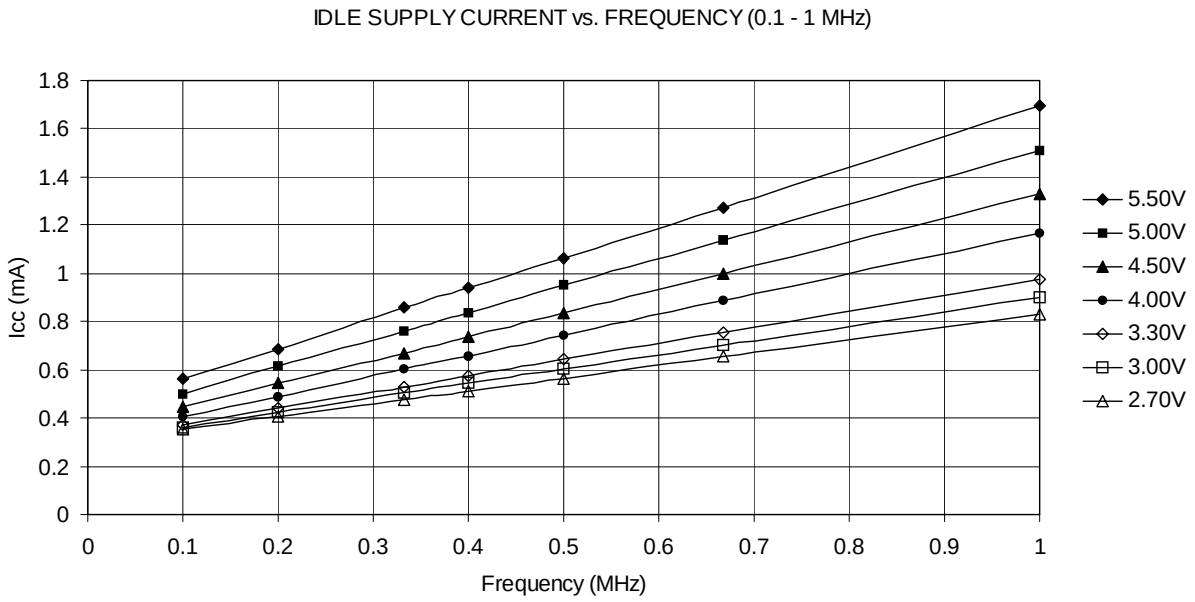



**Figure 186.** Idle Supply Current vs. Frequency (1 - 16 MHz)

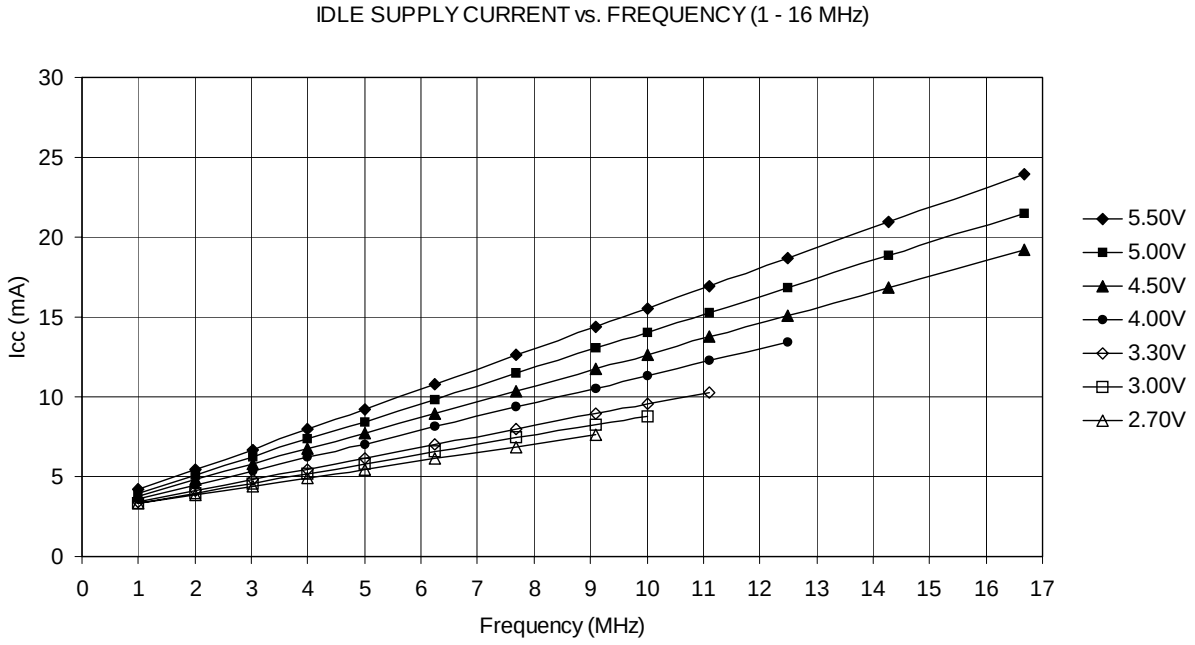


**Figure 187.** Idle Supply Current vs. Vcc (Internal RC Oscillator 8 MHz)

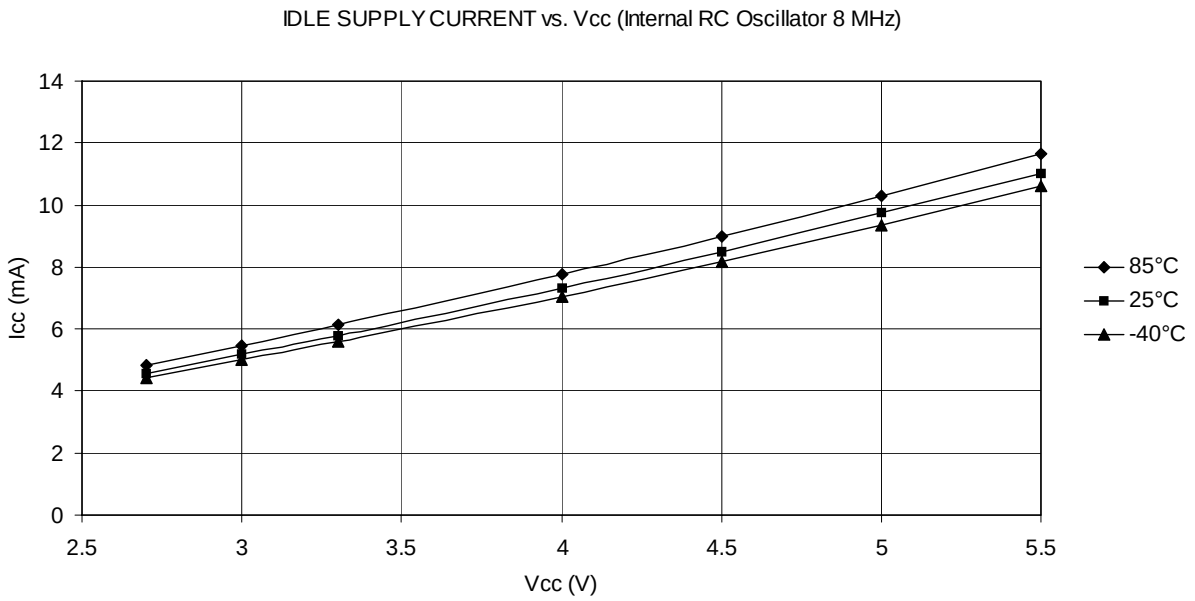



**Figure 188.** Idle Supply Current vs. Vcc (Internal RC Oscillator 1 MHz)

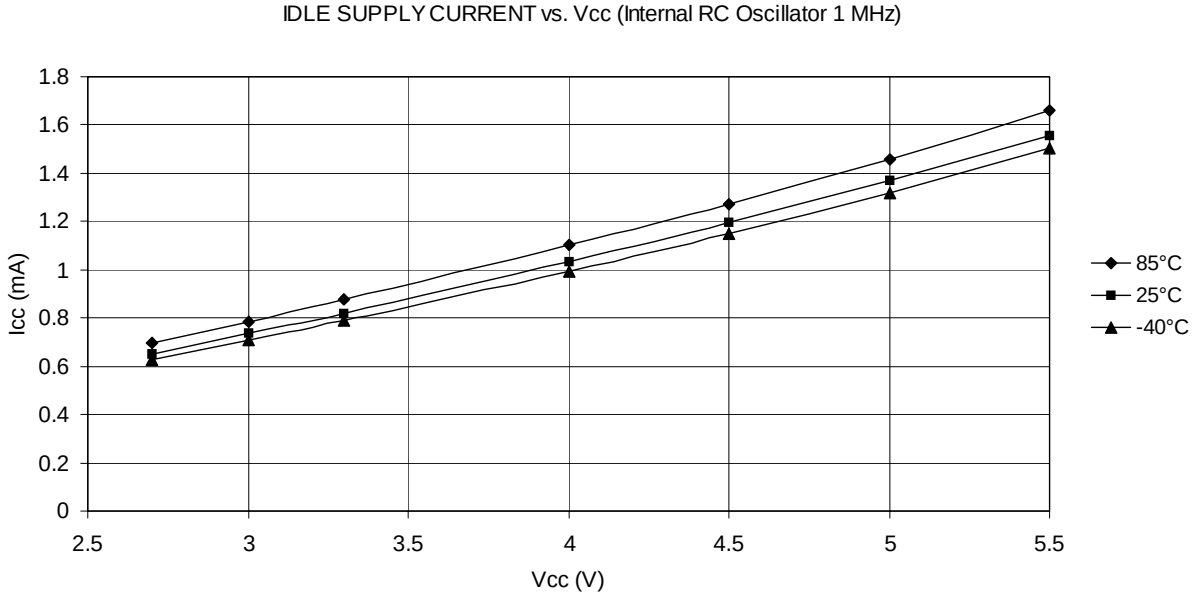
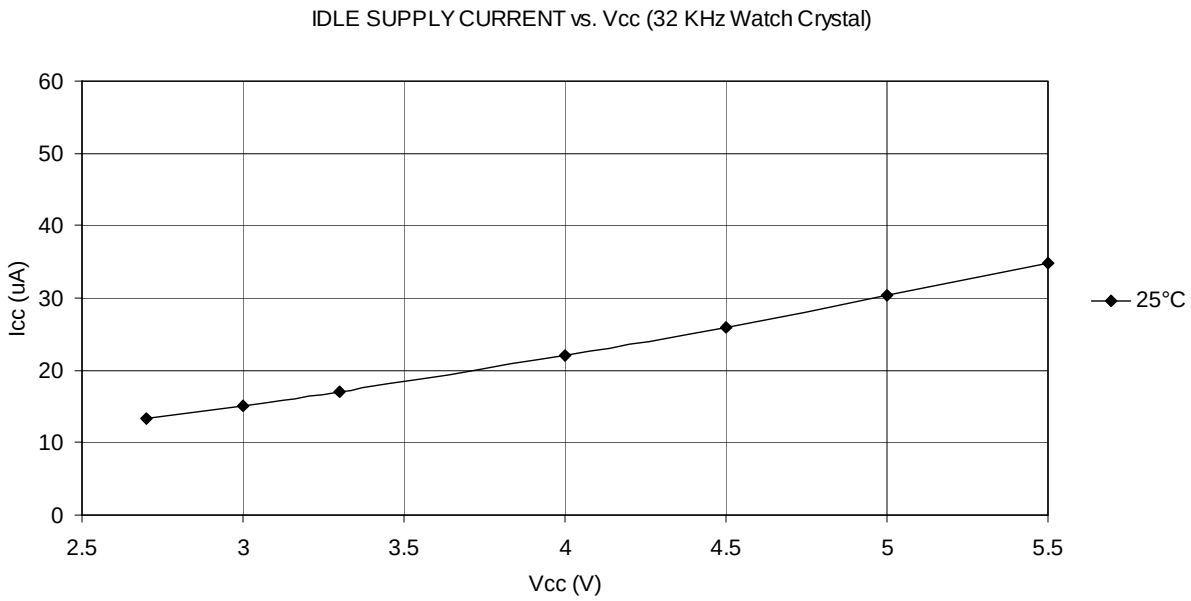


**Figure 189.** Idle Supply Current vs. Vcc (32 kHz Watch Crystal)




## Power-down Supply Current

**Figure 190.** Power-down Supply Current vs. Vcc (Watchdog Timer Disabled)

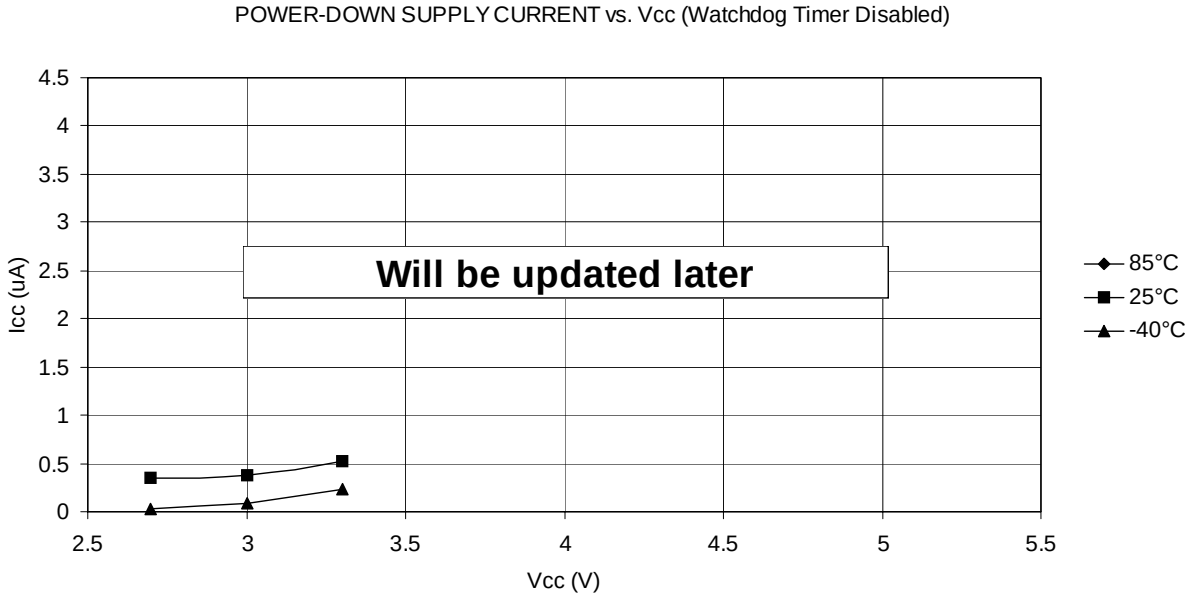
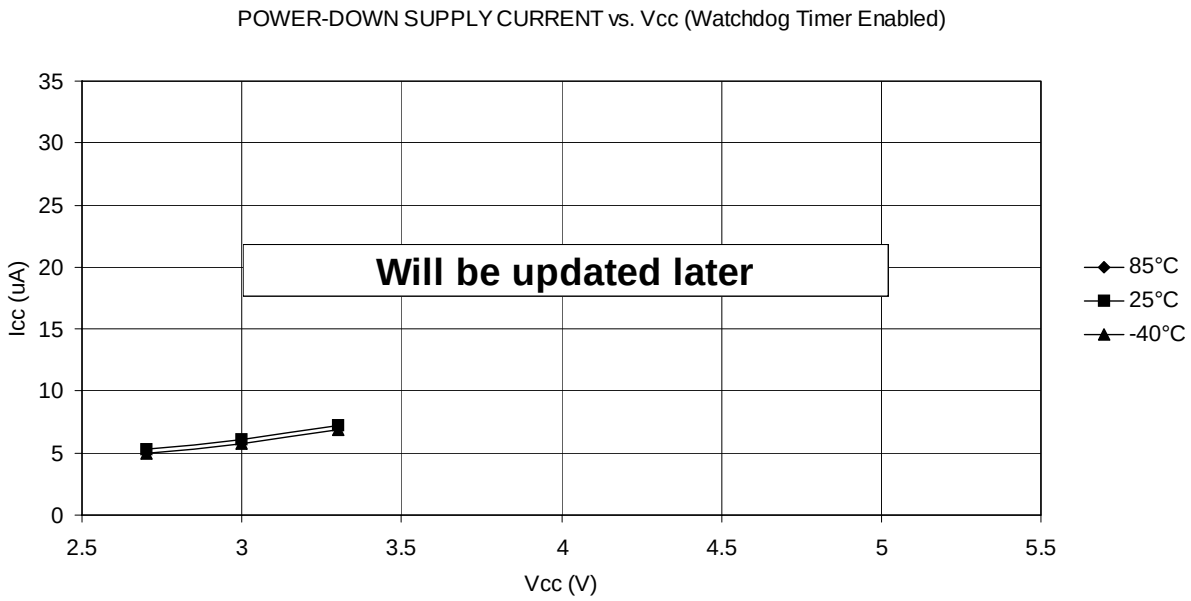
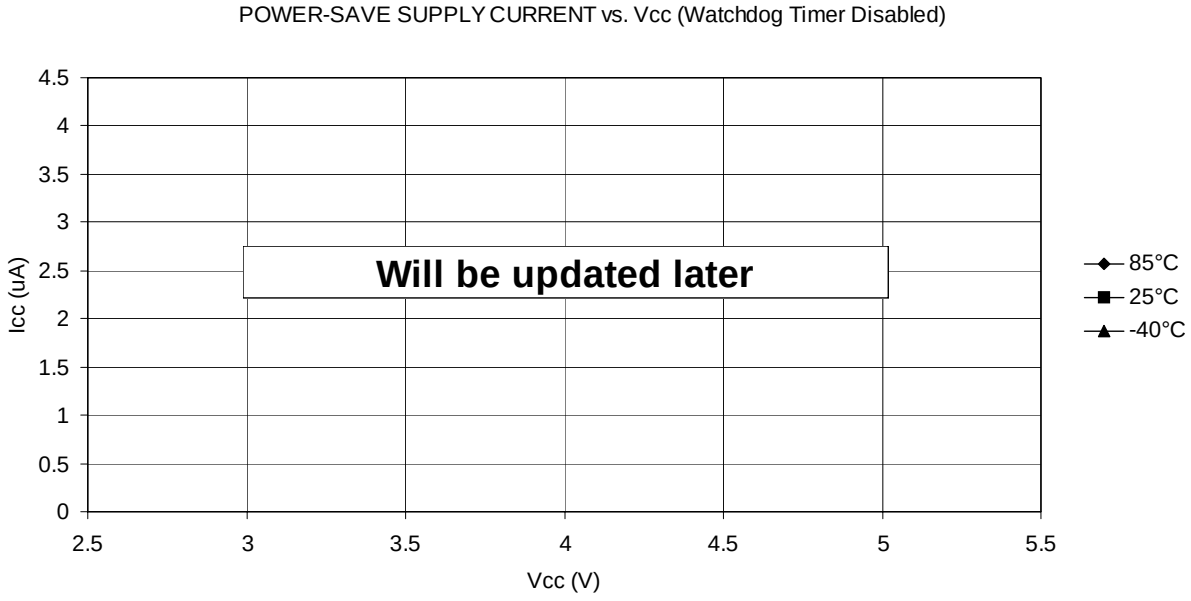


**Figure 191.** Power-down Supply Current vs. Vcc (Watchdog Timer Enabled)



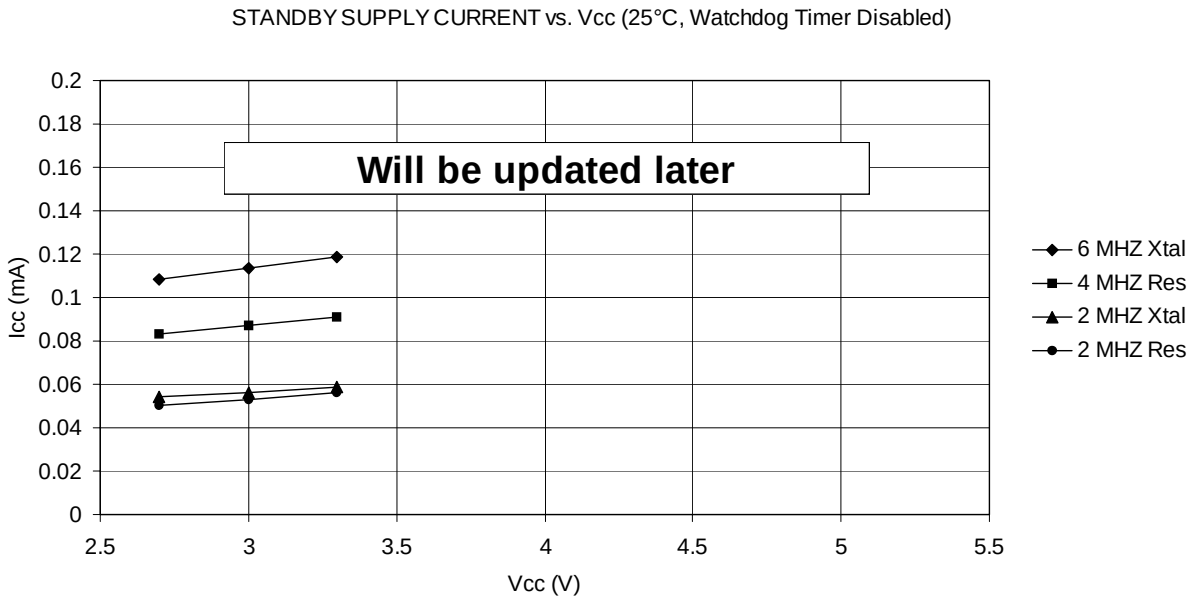

## Power-save Supply Current

**Figure 192.** Power-save Supply Current vs. Vcc (Watchdog Timer Disabled)



## Standby Supply Current

**Figure 193.** Power-save Supply Current vs. Vcc (25°C, Watchdog Timer Disabled)




## Pin Pull-up

**Figure 194.** I/O Pin Pull-up Resistor Current vs. Input Voltage (Vcc = 5V)

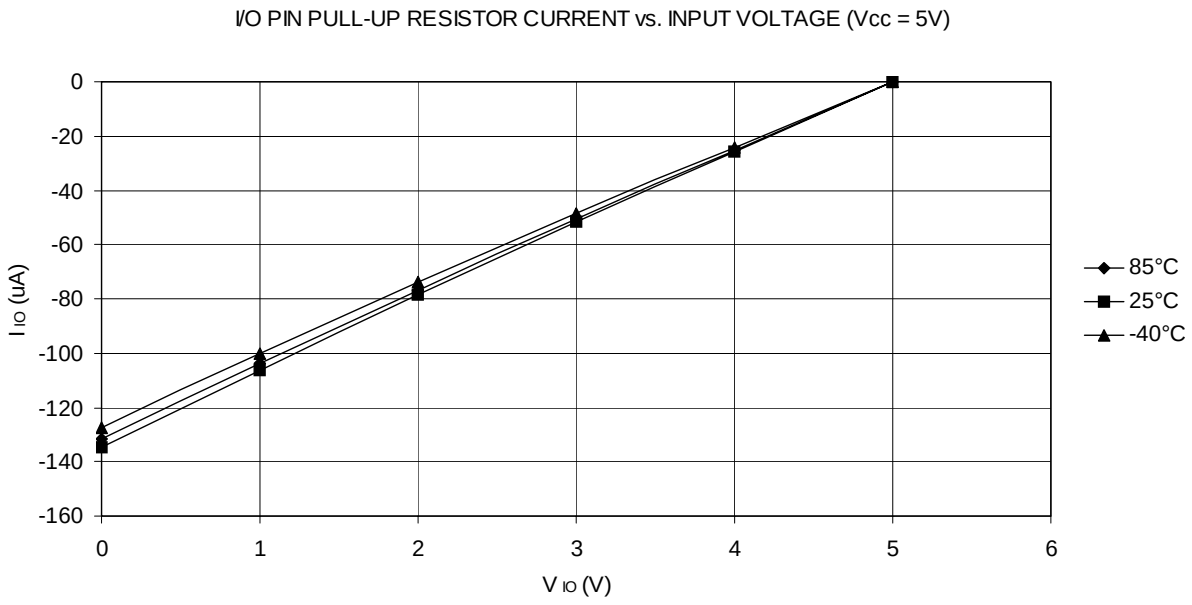


**Figure 195.** I/O Pin Pull-up Resistor Current vs. Input Voltage (Vcc = 2.7V)

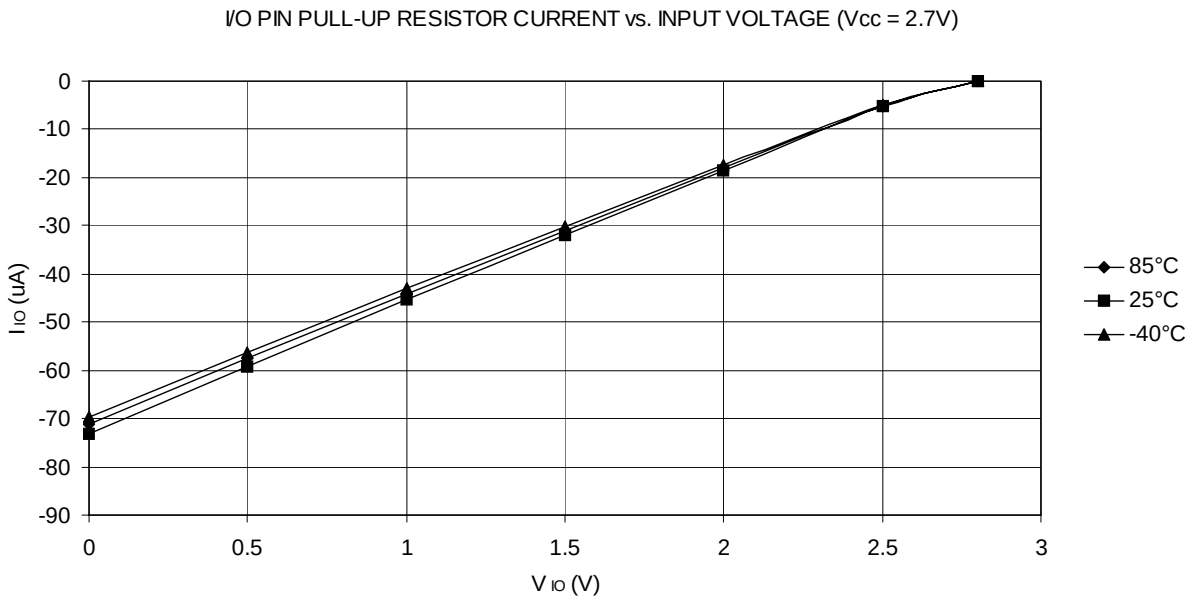



**Figure 196.** Reset Pull-up Resistor Current vs. Reset Pin Voltage (Vcc = 5V)

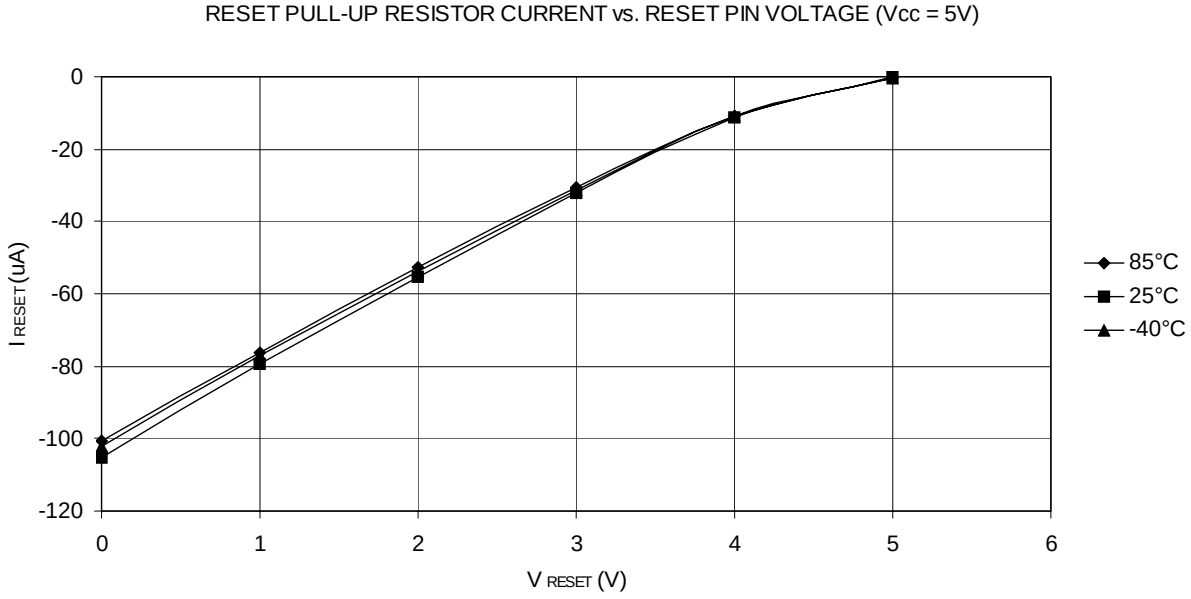
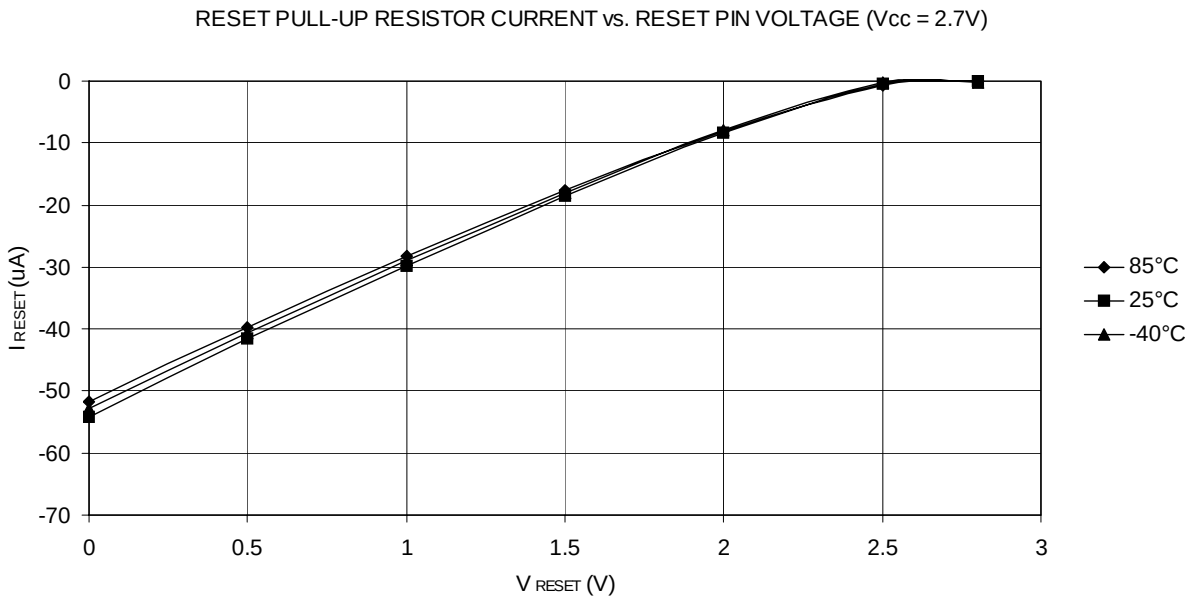


**Figure 197.** Reset Pull-up Resistor Current vs. Reset Pin Voltage (Vcc = 2.7V)




## Pin Driver Strength

**Figure 198.** I/O Pin Source Current vs. Output Voltage (Vcc = 5V)

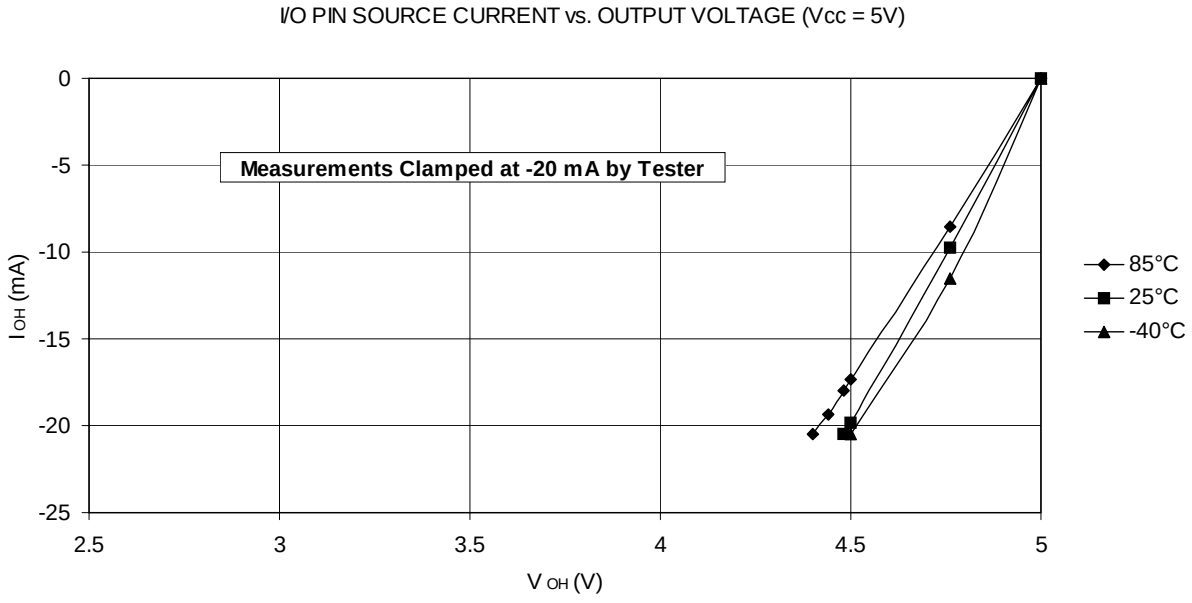


**Figure 199.** I/O Pin Source Current vs. Output Voltage (Vcc = 2.7V)

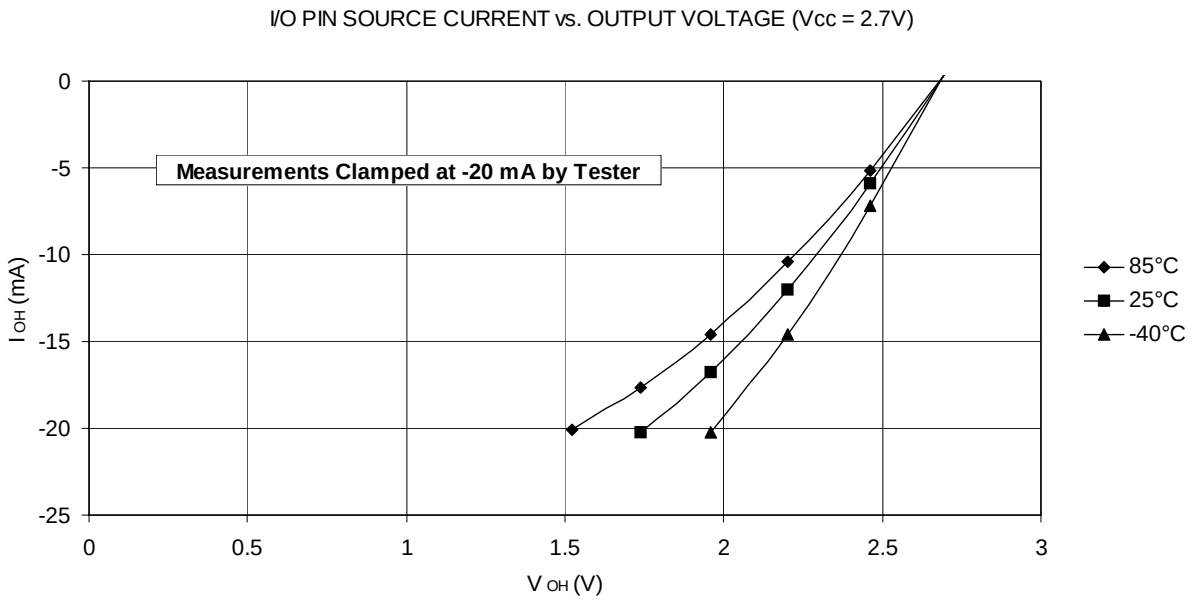



**Figure 200.** I/O Pin Sink Current vs. Output Voltage (Vcc = 5V)

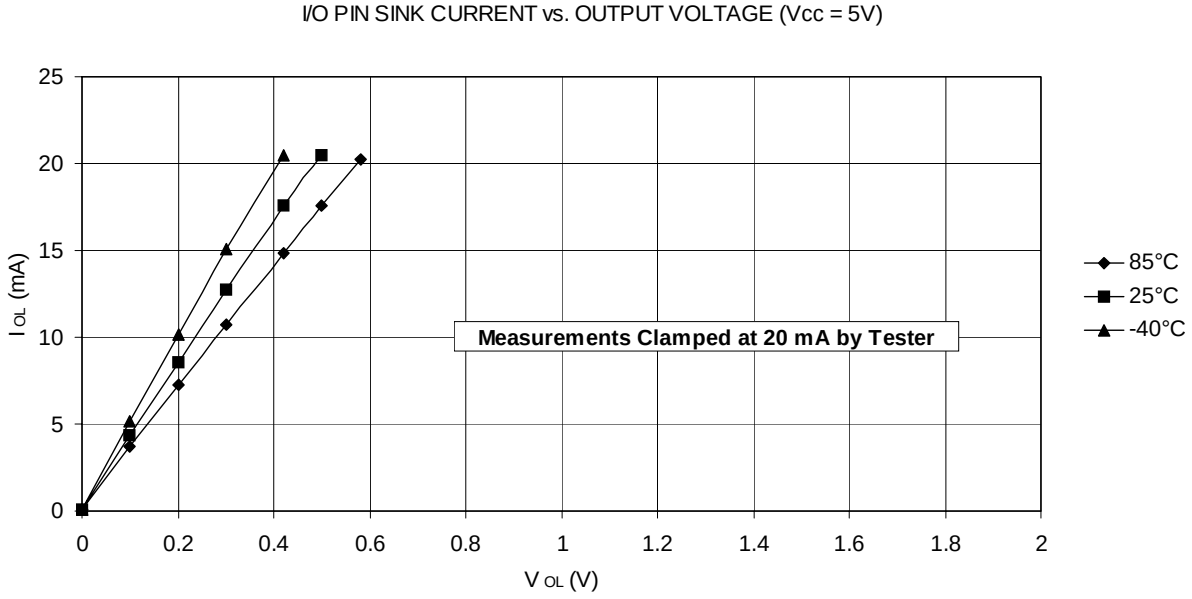
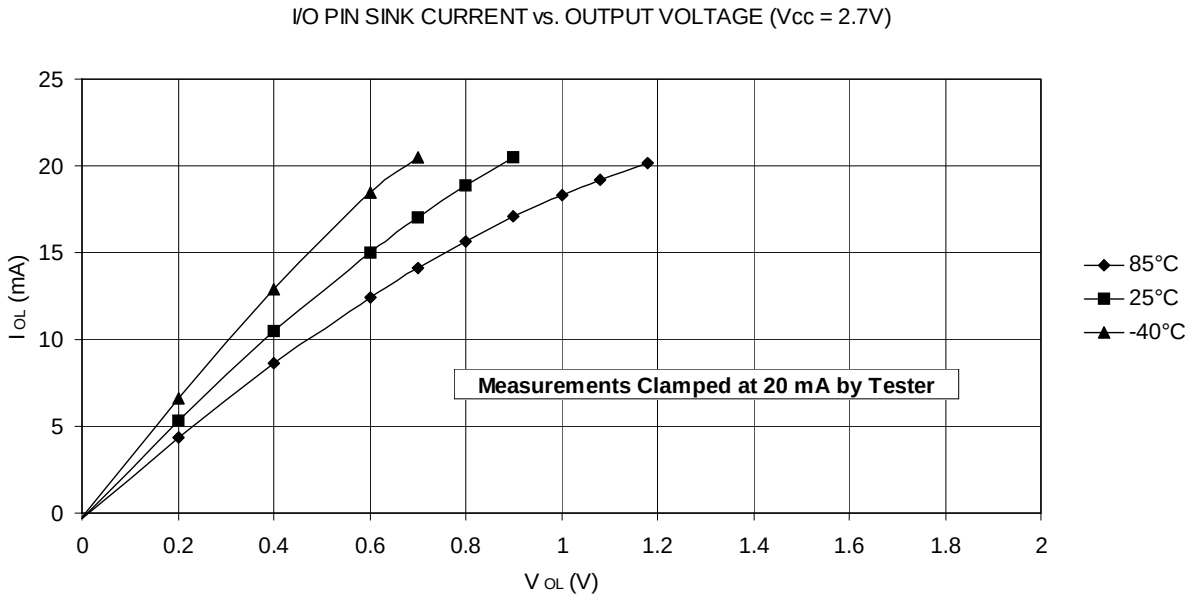


**Figure 201.** I/O Pin Sink Current vs. Output Voltage (Vcc = 2.7V)




## BOD Thresholds and Analog Comparator Offset

**Figure 202.** BOD Thresholds vs. Temperature (BOD level is 4.1V)

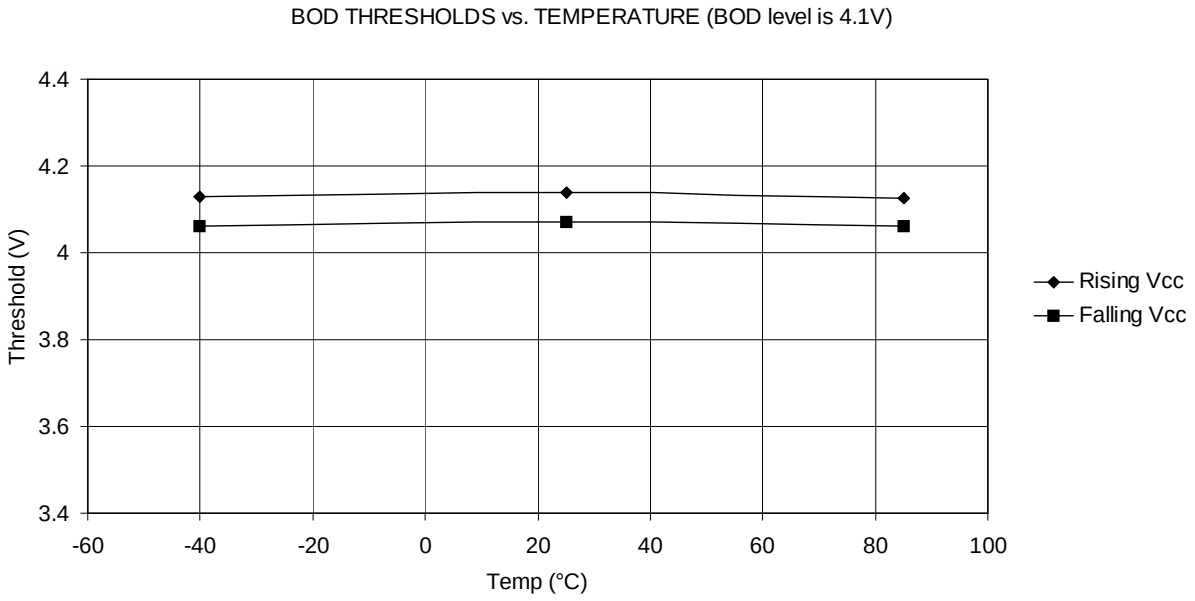


**Figure 203.** BOD Thresholds vs. Temperature (BOD level is 2.7V)

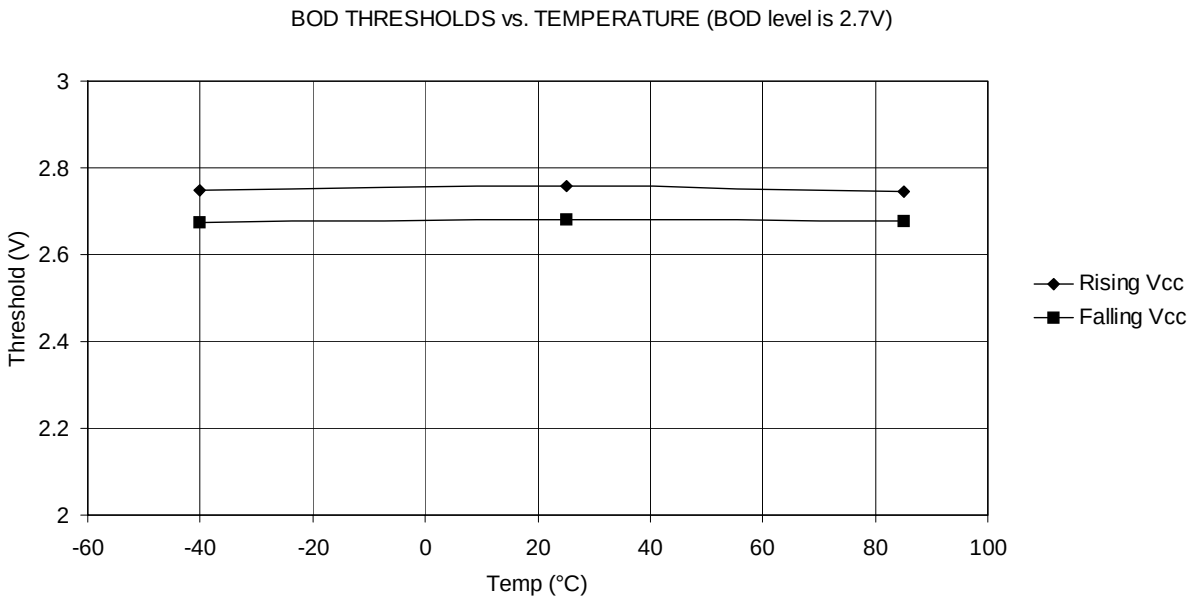



**Figure 204.** Bandgap Voltage vs. Operating Voltage

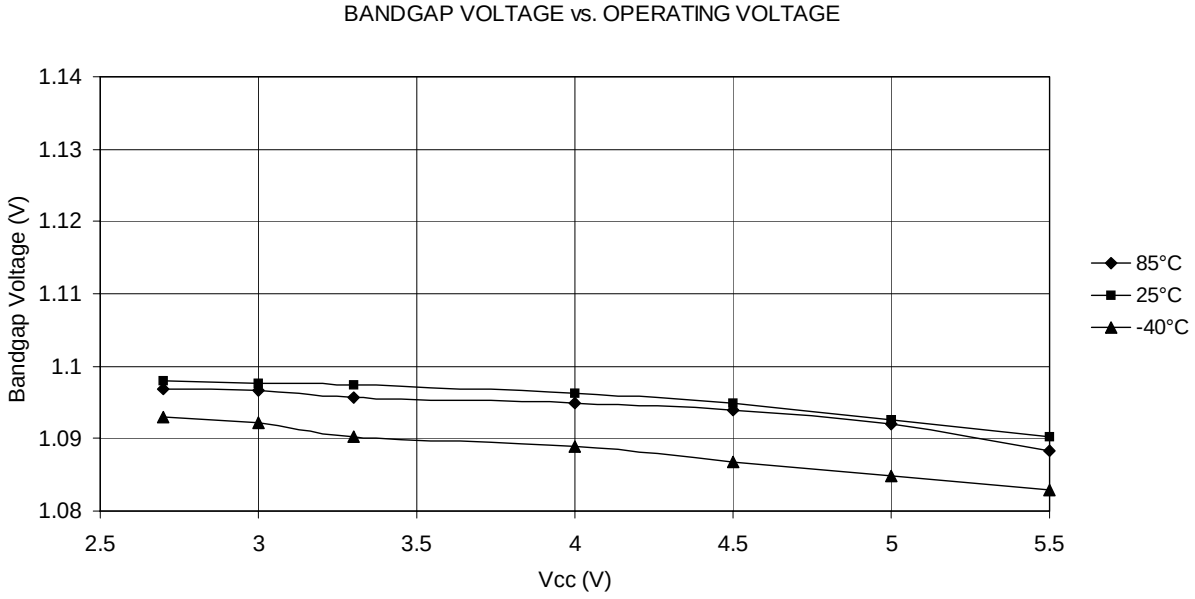
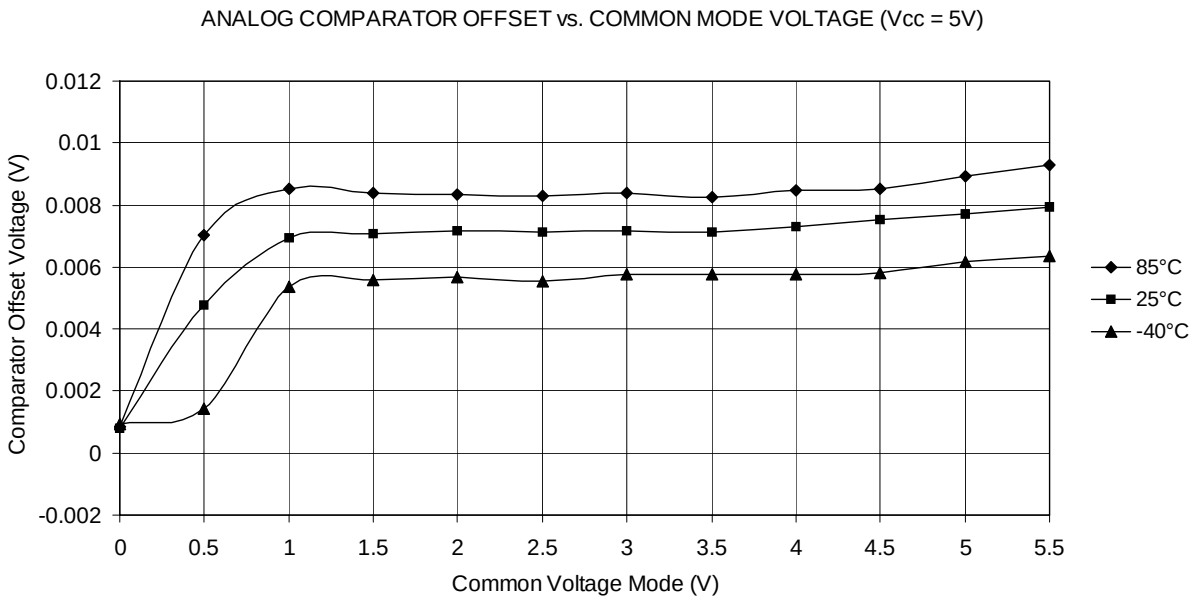


**Figure 205.** Analog Comparator Offset vs. Common Mode Voltage (Vcc = 5V)




## Internal Oscillator Speed

**Figure 206.** Watchdog Oscillator Frequency vs. Operating Voltage

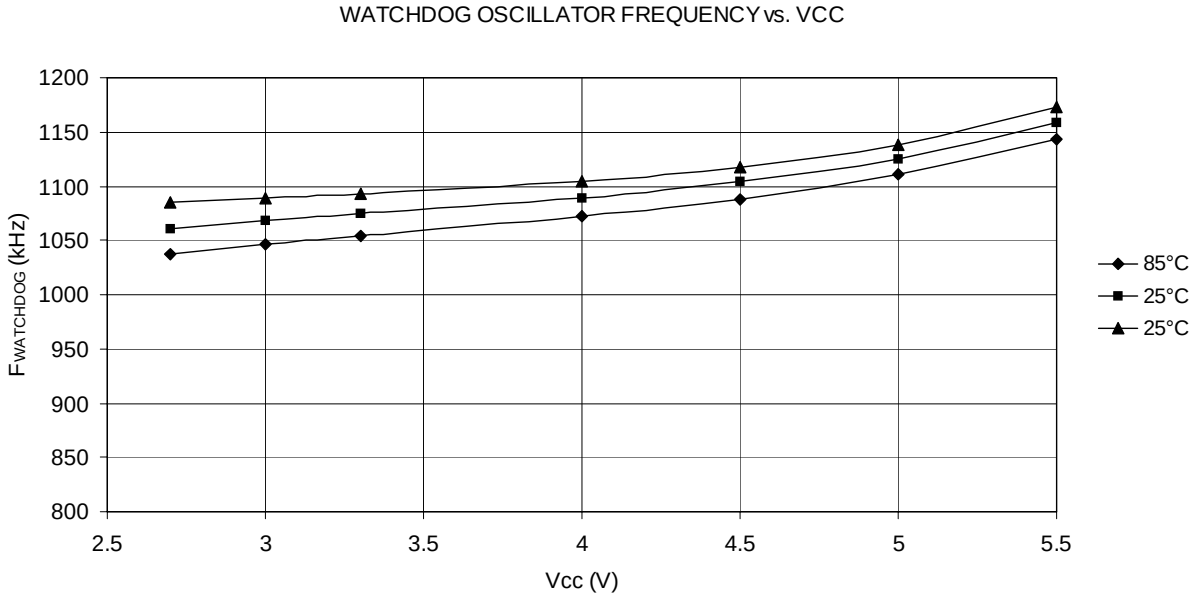


**Figure 207.** Calibrated 8 MHz RC Oscillator Frequency vs. Temperature

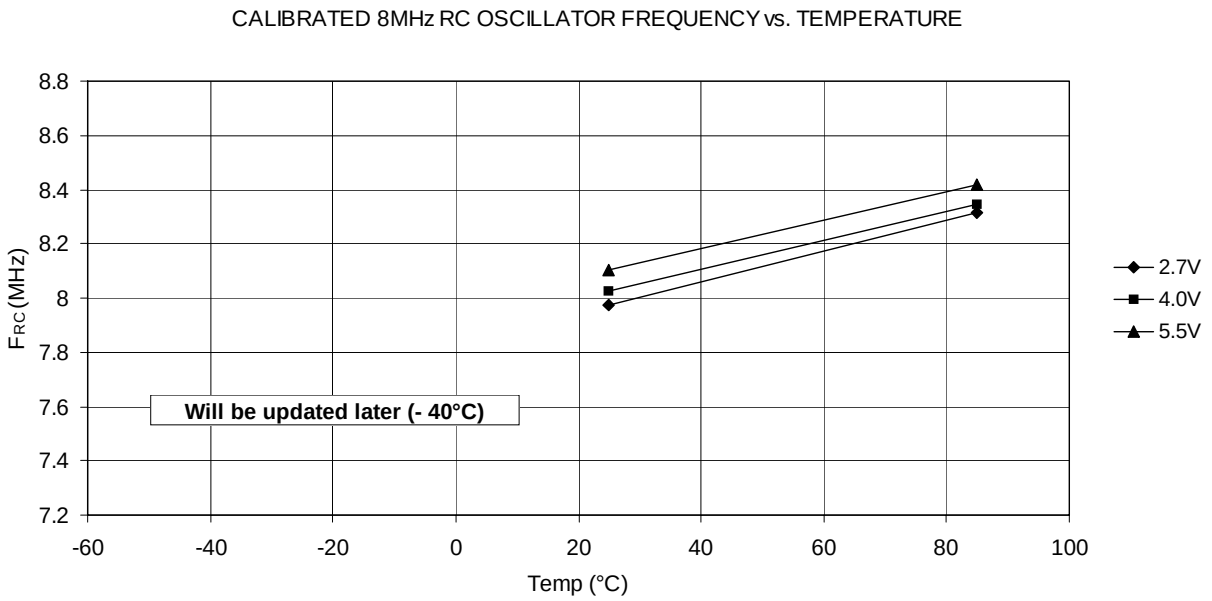



**Figure 208.** Calibrated 8 MHz RC Oscillator Frequency vs. Operating Voltage

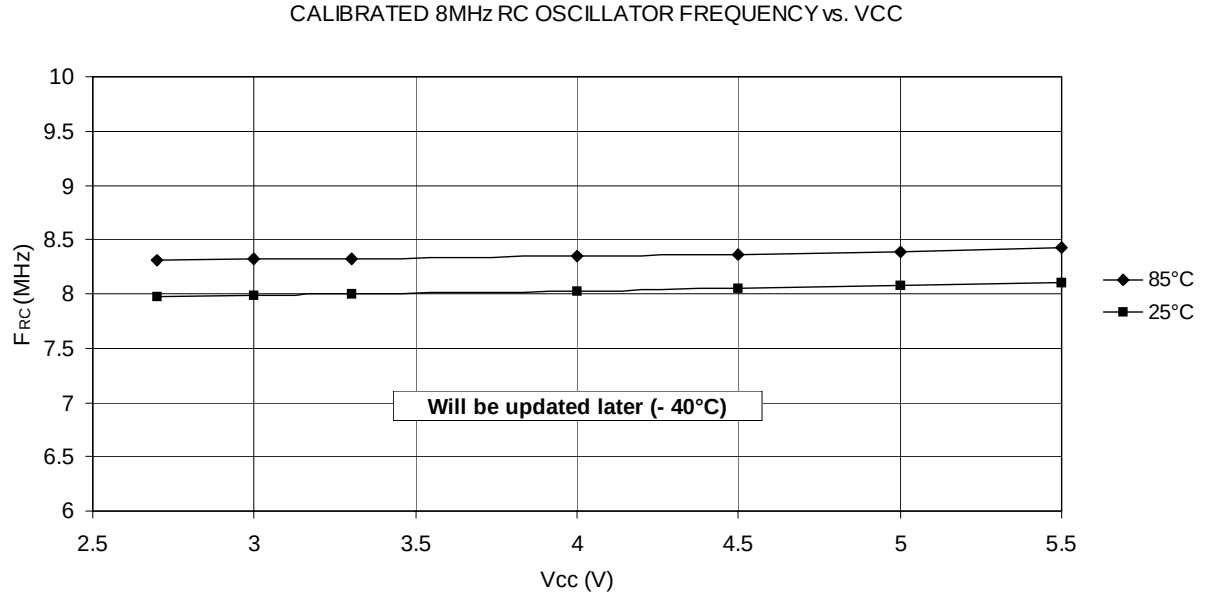
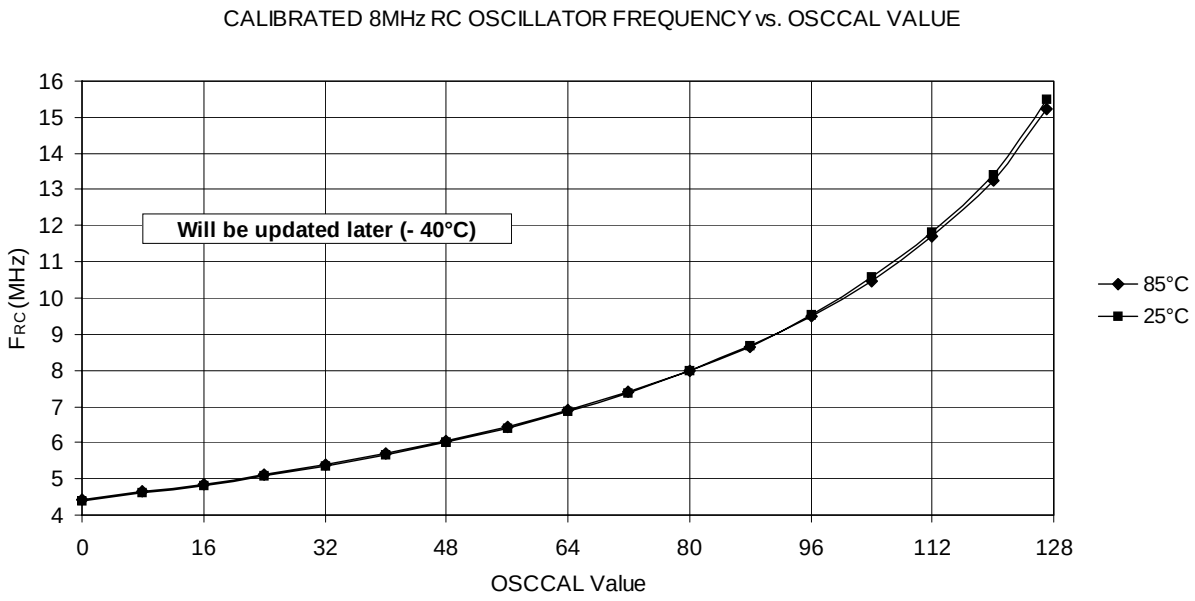


**Figure 209.** Calibrated 8 MHz RC Oscillator Frequency vs. OSCCAL Value




## Current Consumption of Peripheral Units

**Figure 210.** Brownout Detector Current vs. Operating Voltage

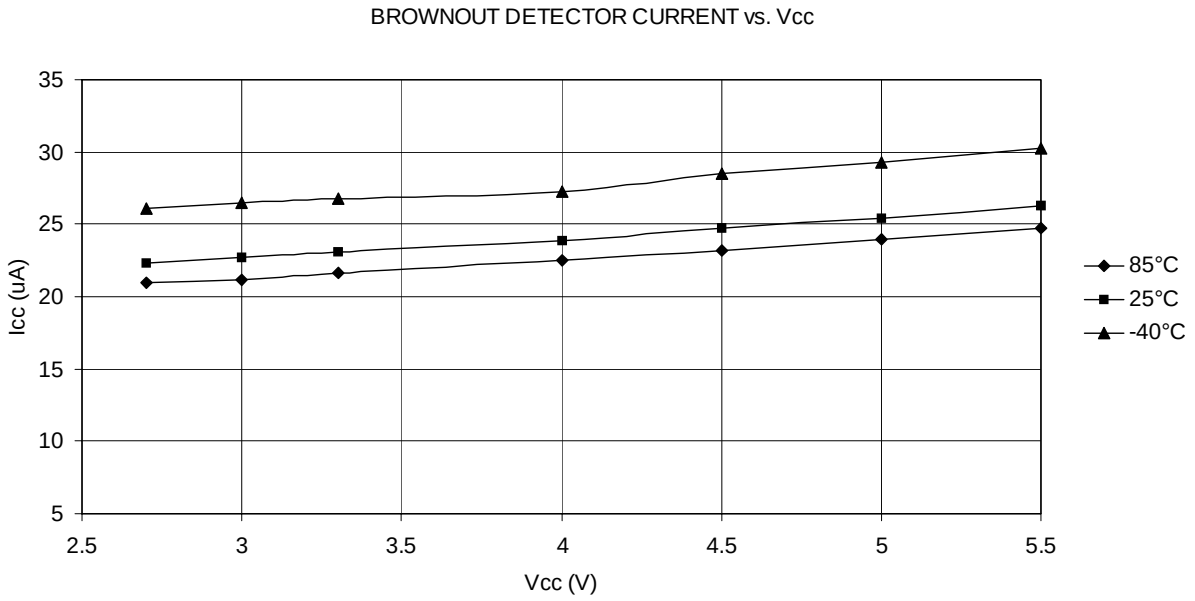


**Figure 211.** ADC Current vs. Operating Voltage (ADC at 1 MHz)

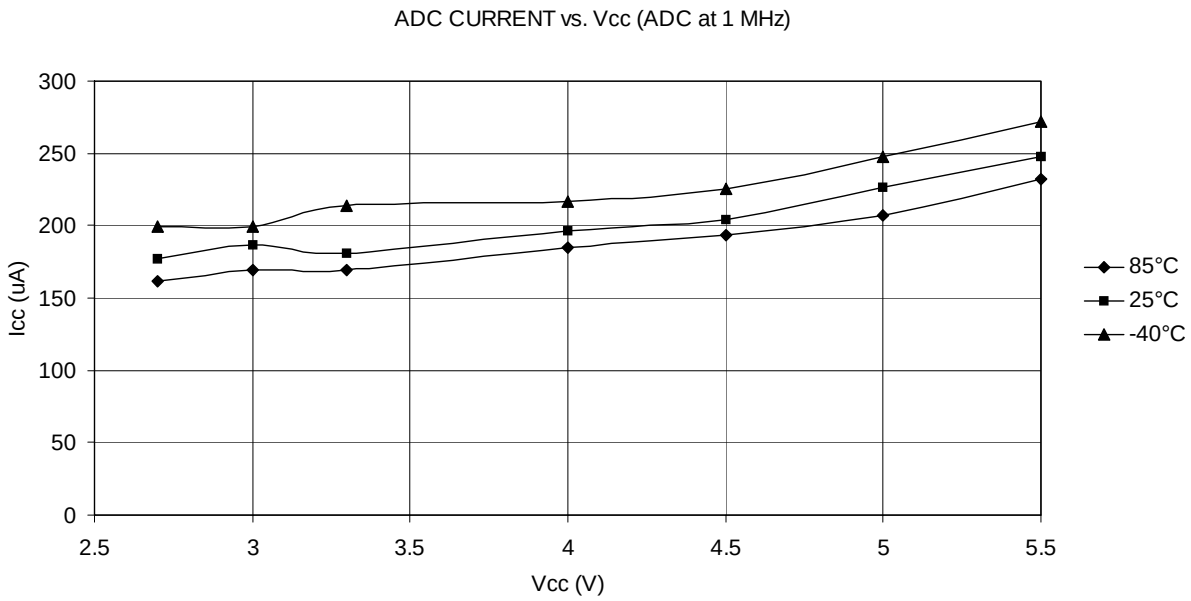



**Figure 212.** AREF External Reference Current vs. Operating Voltage

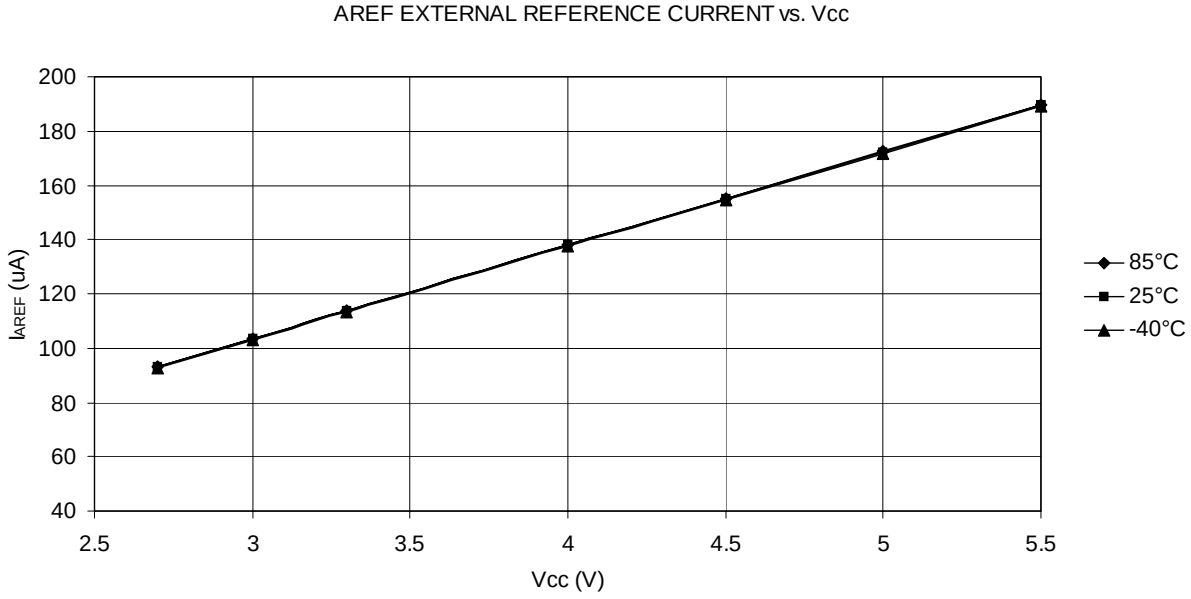


**Figure 213.** Analog Comparator Current vs. Operating Voltage

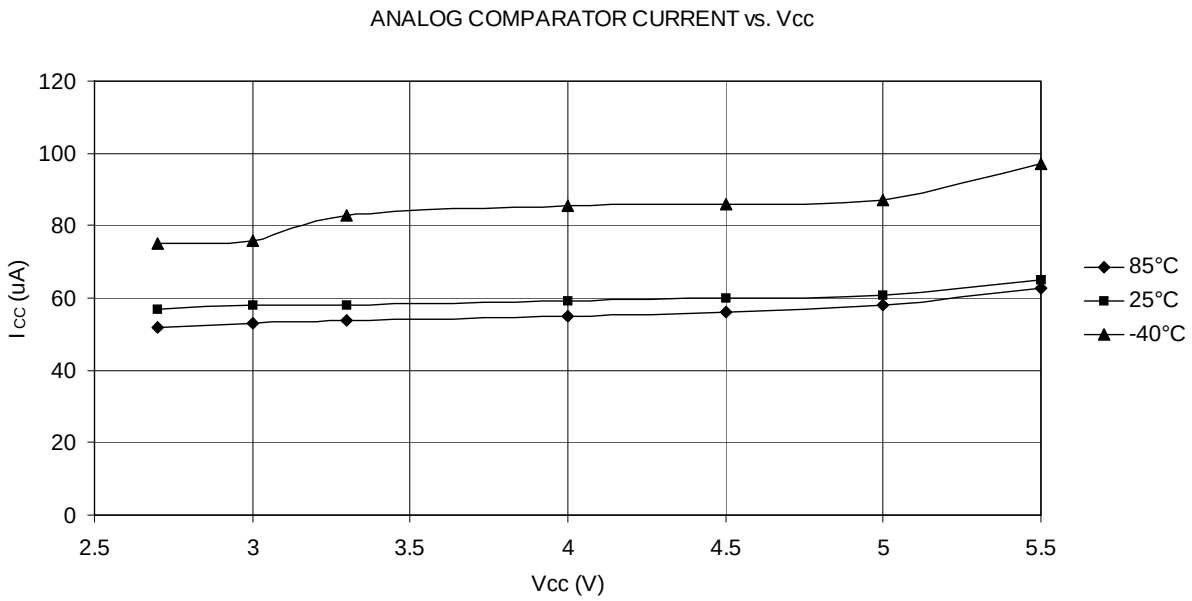
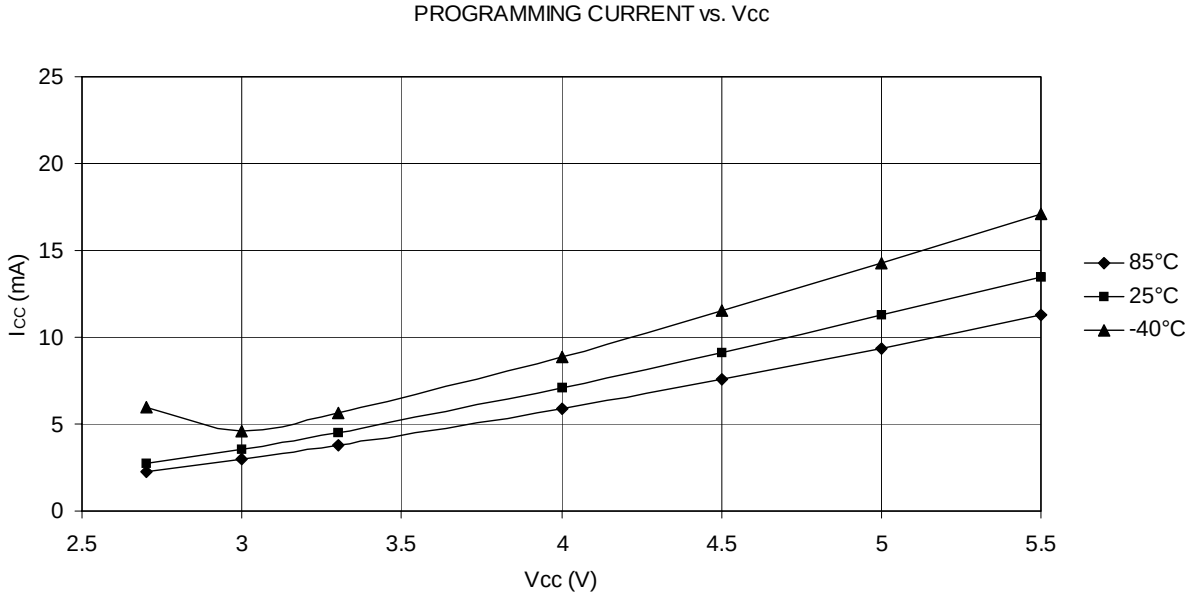



**Figure 214.** Programming Current vs. Operating Voltage



## Current Consumption in Reset and Reset Pulse Width

**Figure 215.** Reset Supply Current vs. Operating Voltage (0.1 - 1.0 MHz) (Excluding Current Through the Reset Pull-up)

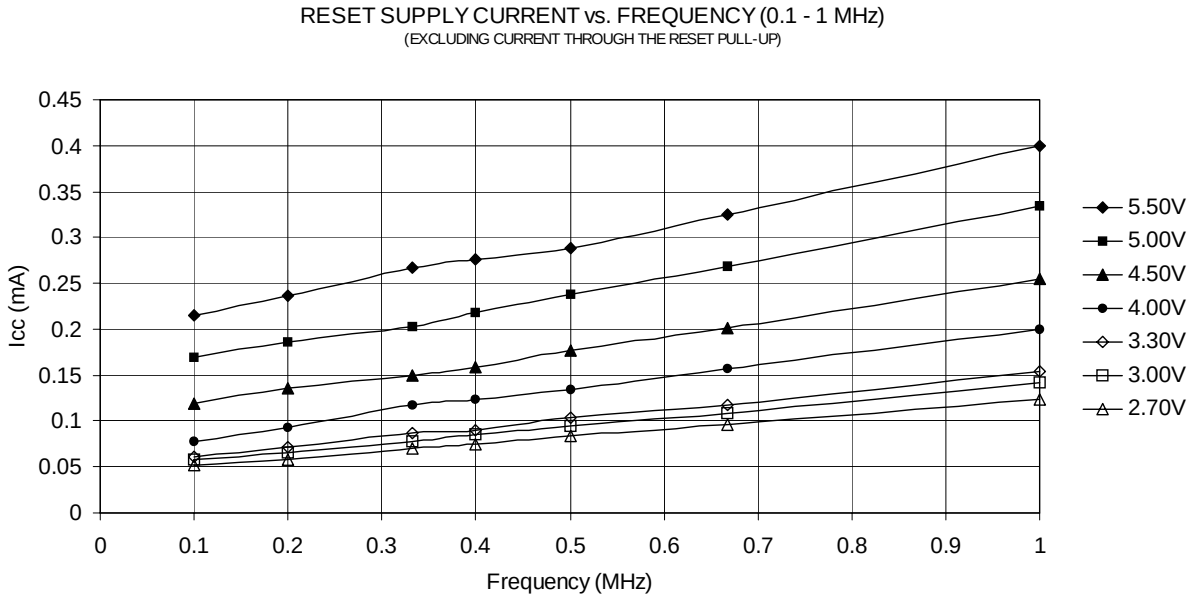



**Figure 216.** Reset Supply Current vs. Operating Voltage (1 - 16 MHz) (Excluding Current Through the Reset Pull-up)

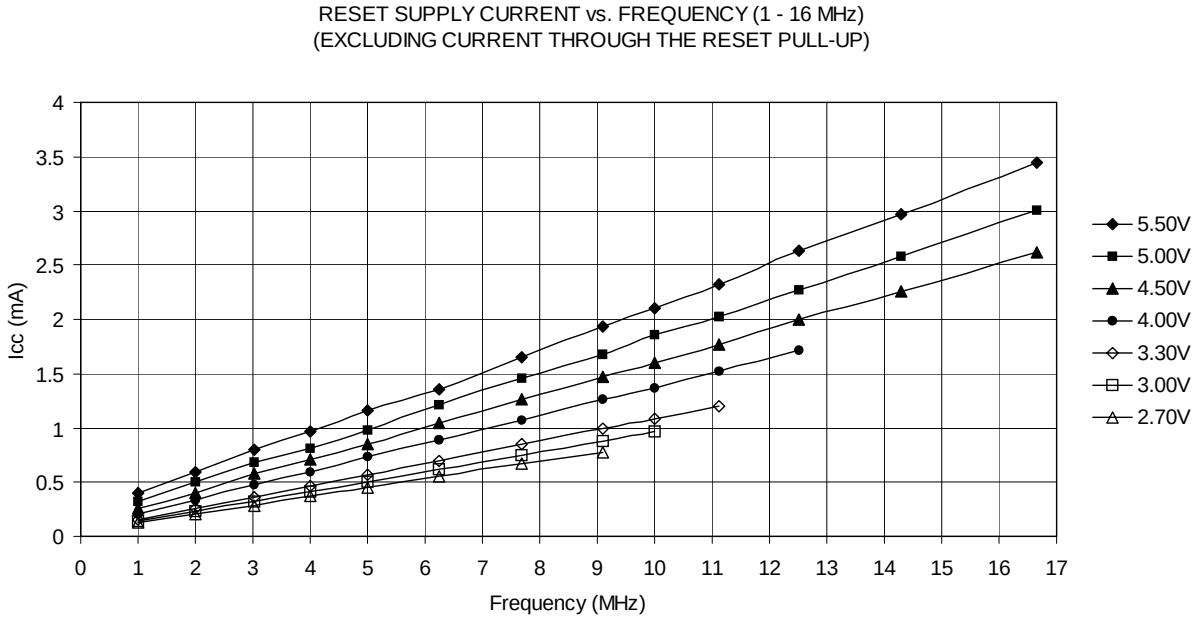
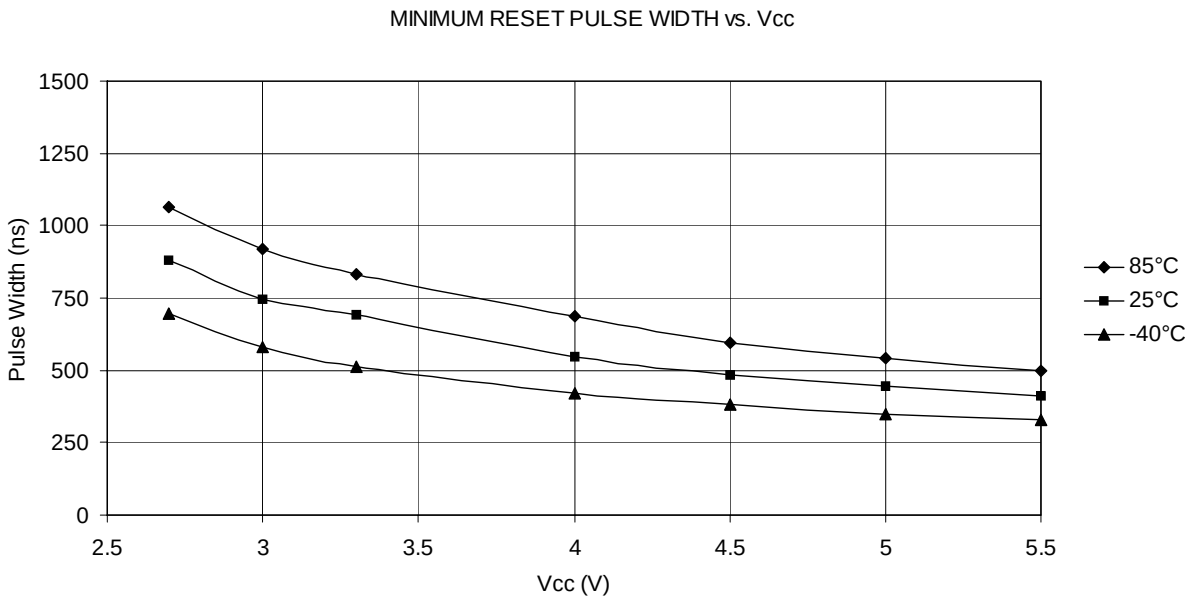


**Figure 217.** Minimum Reset Pulse Width vs. Operating Voltage



## Register Summary

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Page |
|---|---|---|---|---|---|---|---|---|---|---|
| (0xFF) | **Reserved** | | | | | | | | | |
| (0xFE) | **Reserved** | | | | | | | | | |
| (0xFD) | **Reserved** | | | | | | | | | |
| (0xFC) | **Reserved** | | | | | | | | | |
| (0xFB) | **Reserved** | | | | | | | | | |
| (0xFA) | **CANMSG** | MSG 7 | MSG 6 | MSG 5 | MSG 4 | MSG 3 | MSG 2 | MSG 1 | MSG 0 | page 259 |
| (0xF9) | **CANSTMH** | TIMSTM15 | TIMSTM14 | TIMSTM13 | TIMSTM12 | TIMSTM11 | TIMSTM10 | TIMSTM9 | TIMSTM8 | page 259 |
| (0xF8) | **CANSTML** | TIMSTM7 | TIMSTM6 | TIMSTM5 | TIMSTM4 | TIMSTM3 | TIMSTM2 | TIMSTM1 | TIMSTM0 | page 259 |
| (0xF7) | **CANIDM1** | IDMSK28 | IDMSK27 | IDMSK26 | IDMSK25 | IDMSK24 | IDMSK23 | IDMSK22 | IDMSK21 | page 258 |
| (0xF6) | **CANIDM2** | IDMSK20 | IDMSK19 | IDMSK18 | IDMSK17 | IDMSK16 | IDMSK15 | IDMSK14 | IDMSK13 | page 258 |
| (0xF5) | **CANIDM3** | IDMSK12 | IDMSK11 | IDMSK10 | IDMSK9 | IDMSK8 | IDMSK7 | IDMSK6 | IDMSK5 | page 258 |
| (0xF4) | **CANIDM4** | IDMSK4 | IDMSK3 | IDMSK2 | IDMSK1 | IDMSK0 | RTRMSK | – | IDEMSK | page 258 |
| (0xF3) | **CANIDT1** | IDT28 | IDT27 | IDT26 | IDT25 | IDT24 | IDT23 | IDT22 | IDT21 | page 257 |
| (0xF2) | **CANIDT2** | IDT20 | IDT19 | IDT18 | IDT17 | IDT16 | IDT15 | IDT14 | IDT13 | page 257 |
| (0xF1) | **CANIDT3** | IDT12 | IDT11 | IDT10 | IDT9 | IDT8 | IDT7 | IDT6 | IDT5 | page 257 |
| (0xF0) | **CANIDT4** | IDT4 | IDT3 | IDT2 | IDT1 | IDT0 | RTRTAG | RB1TAG | RB0TAG | page 257 |
| (0xEF) | **CANCDMOB** | CONMOB1 | CONMOB0 | RPLV | IDE | DLC3 | DLC2 | DLC1 | DLC0 | page 256 |
| (0xEE) | **CANSTMOB** | DLCW | TXOK | RXOK | BERR | SERR | CERR | FERR | AERR | page 254 |
| (0xED) | **CANPAGE** | MOBNB3 | MOBNB2 | MOBNB1 | MOBNB0 | AINC | INDX2 | INDX1 | INDX0 | page 254 |
| (0xEC) | **CANHPMOB** | HPMOB3 | HPMOB2 | HPMOB1 | HPMOB0 | CGP3 | CGP2 | CGP1 | CGP0 | page 254 |
| (0xEB) | **CANREC** | REC7 | REC6 | REC5 | REC4 | REC3 | REC2 | REC1 | REC0 | page 253 |
| (0xEA) | **CANTEC** | TEC7 | TEC6 | TEC5 | TEC4 | TEC3 | TEC2 | TEC1 | TEC0 | page 253 |
| (0xE9) | **CANTTCH** | TIMTTC15 | TIMTTC14 | TIMTTC13 | TIMTTC12 | TIMTTC11 | TIMTTC10 | TIMTTC9 | TIMTTC8 | page 253 |
| (0xE8) | **CANTTCL** | TIMTTC7 | TIMTTC6 | TIMTTC5 | TIMTTC4 | TIMTTC3 | TIMTTC2 | TIMTTC1 | TIMTTC0 | page 253 |
| (0xE7) | **CANTIMH** | CANTIM15 | CANTIM14 | CANTIM13 | CANTIM12 | CANTIM11 | CANTIM10 | CANTIM9 | CANTIM8 | page 253 |
| (0xE6) | **CANTIML** | CANTIM7 | CANTIM6 | CANTIM5 | CANTIM4 | CANTIM3 | CANTIM2 | CANTIM1 | CANTIM0 | page 253 |
| (0xE5) | **CANTCON** | TPRSC7 | TPRSC6 | TPRSC5 | TPRSC4 | TPRSC3 | TPRSC2 | TRPSC1 | TPRSC0 | page 252 |
| (0xE4) | **CANBT3** | – | PHS22 | PHS21 | PHS20 | PHS12 | PHS11 | PHS10 | SMP | page 252 |
| (0xE3) | **CANBT2** | – | SJW1 | SJW0 | – | PRS2 | PRS1 | PRS0 | – | page 251 |
| (0xE2) | **CANBT1** | – | BRP5 | BRP4 | BRP3 | BRP2 | BRP1 | BRP0 | – | page 251 |
| (0xE1) | **CANSIT1** | – | SIT14 | SIT13 | SIT12 | SIT11 | SIT10 | SIT9 | SIT8 | page 250 |
| (0xE0) | **CANSIT2** | SIT7 | SIT6 | SIT5 | SIT4 | SIT3 | SIT2 | SIT1 | SIT0 | page 250 |
| (0xDF) | **CANIE1** | – | IEMOB14 | IEMOB13 | IEMOB12 | IEMOB11 | IEMOB10 | IEMOB9 | IEMOB8 | page 250 |
| (0xDE) | **CANIE2** | IEMOB7 | IEMOB6 | IEMOB5 | IEMOB4 | IEMOB3 | IEMOB2 | IEMOB1 | IEMOB0 | page 250 |
| (0xDD) | **CANEN1** | – | ENMOB14 | ENMOB13 | ENMOB12 | ENMOB11 | ENMOB10 | ENMOB9 | ENMOB8 | page 250 |
| (0xDC) | **CANEN2** | ENMOB7 | ENMOB6 | ENMOB5 | ENMOB4 | ENMOB3 | ENMOB2 | ENMOB1 | ENMOB0 | page 250 |
| (0xDB) | **CANGIE** | ENIT | ENBOFF | ENRX | ENTX | ENERR | ENBX | ENERG | ENOVRT | page 249 |
| (0xDA) | **CANGIT** | CANIT | BOFFIT | OVRTIM | BXOK | SERG | CERG | FERG | AERG | page 248 |
| (0xD9) | **CANGSTA** | – | OVRG | – | TXBSY | RXBSY | ENFG | BOFF | ERRP | page 247 |
| (0xD8) | **CANGCON** | ABRQ | OVRQ | TTC | SYNTTC | LISTEN | TEST | ENA/STB | SWRES | page 246 |
| (0xD7) | **Reserved** | | | | | | | | | |
| (0xD6) | **Reserved** | | | | | | | | | |
| (0xD5) | **Reserved** | | | | | | | | | |
| (0xD4) | **Reserved** | | | | | | | | | |
| (0xD3) | **Reserved** | | | | | | | | | |
| (0xD2) | **Reserved** | | | | | | | | | |
| (0xD1) | **Reserved** | | | | | | | | | |
| (0xD0) | **Reserved** | | | | | | | | | |
| (0xCF) | **Reserved** | | | | | | | | | |
| (0xCE) | **UDR1** | UDR17 | UDR16 | UDR15 | UDR14 | UDR13 | UDR12 | UDR11 | UDR10 | page 189 |
| (0xCD) | **UBRR1H** | – | – | – | – | UBRR111 | UBRR110 | UBRR19 | UBRR18 | page 193 |
| (0xCC) | **UBRR1L** | UBRR17 | UBRR16 | UBRR15 | UBRR14 | UBRR13 | UBRR12 | UBRR11 | UBRR10 | page 193 |
| (0xCB) | **Reserved** | | | | | | | | | |
| (0xCA) | **UCSR1C** | – | UMSEL1 | UPM11 | UPM10 | USBS1 | UCSZ11 | UCSZ10 | UCPOL1 | page 192 |
| (0xC9) | **UCSR1B** | RXCIE1 | TXCIE1 | UDRIE1 | RXEN1 | TXEN1 | UCSZ12 | RXB81 | TXB81 | page 191 |
| (0xC8) | **UCSR1A** | RXC1 | TXC1 | UDRE1 | FE1 | DOR1 | UPE1 | U2X1 | MPCM1 | page 189 |
| (0xC7) | **Reserved** | | | | | | | | | |
| (0xC6) | **UDR0** | UDR07 | UDR06 | UDR05 | UDR04 | UDR03 | UDR02 | UDR01 | UDR00 | page 189 |
| (0xC5) | **UBRR0H** | – | – | – | – | UBRR011 | UBRR010 | UBRR09 | UBRR08 | page 193 |
| (0xC4) | **UBRR0L** | UBRR07 | UBRR06 | UBRR05 | UBRR04 | UBRR03 | UBRR02 | UBRR01 | UBRR00 | page 193 |
| (0xC3) | **Reserved** | | | | | | | | | |
| (0xC2) | **UCSR0C** | – | UMSEL0 | UPM01 | UPM00 | USBS0 | UCSZ01 | UCSZ00 | UCPOL0 | page 191 |
| (0xC1) | **UCSR0B** | RXCIE0 | TXCIE0 | UDRIE0 | RXEN0 | TXEN0 | UCSZ02 | RXB80 | TXB80 | page 190 |
| (0xC0) | **UCSR0A** | RXC0 | TXC0 | UDRE0 | FE0 | DOR0 | UPE0 | U2X0 | MPCM0 | page 189 |

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Page |
|---|---|---|---|---|---|---|---|---|---|---|
| (0xBF) | **Reserved** | | | | | | | | | |
| (0xBE) | **Reserved** | | | | | | | | | |
| (0xBD) | **Reserved** | | | | | | | | | |
| (0xBC) | **TWCR** | TWINT | TWEA | TWSTA | TWSTO | TWWC | TWEN | – | TWIE | page 207 |
| (0xBB) | **TWDR** | TWDR7 | TWDR6 | TWDR5 | TWDR4 | TWDR3 | TWDR2 | TWDR1 | TWDR0 | page 209 |
| (0xBA) | **TWAR** | TWAR6 | TWAR5 | TWAR4 | TWAR3 | TWAR2 | TWAR1 | TWAR0 | TWGCE | page 209 |
| (0xB9) | **TWSR** | TWS7 | TWS6 | TWS5 | TWS4 | TWS3 | – | TWPS1 | TWPS0 | page 208 |
| (0xB8) | **TWBR** | TWBR7 | TWBR6 | TWBR5 | TWBR4 | TWBR3 | TWBR2 | TWBR1 | TWBR0 | page 207 |
| (0xB7) | **Reserved** | | | | | | | | | |
| (0xB6) | **ASSR** | – | – | – | EXCLK | AS2 | TCN2UB | OCR2UB | TCR2UB | page 154 |
| (0xB5) | **Reserved** | | | | | | | | | |
| (0xB4) | **Reserved** | | | | | | | | | |
| (0xB3) | **OCR2A** | OCR2A7 | OCR2A6 | OCR2A5 | OCR2A4 | OCR2A3 | OCR2A2 | OCR2A1 | OCR2A0 | page 154 |
| (0xB2) | **TCNT2** | TCNT27 | TCNT26 | TCNT25 | TCNT24 | TCNT23 | TCNT22 | TCNT21 | TCNT20 | page 153 |
| (0xB1) | **Reserved** | | | | | | | | | |
| (0xB0) | **TCCR2A** | FOC2A | WGM20 | COM2A1 | COM2A0 | WGM21 | CS22 | CS21 | CS20 | page 159 |
| (0xAF) | **Reserved** | | | | | | | | | |
| (0xAE) | **Reserved** | | | | | | | | | |
| (0xAD) | **Reserved** | | | | | | | | | |
| (0xAC) | **Reserved** | | | | | | | | | |
| (0xAB) | **Reserved** | | | | | | | | | |
| (0xAA) | **Reserved** | | | | | | | | | |
| (0xA9) | **Reserved** | | | | | | | | | |
| (0xA8) | **Reserved** | | | | | | | | | |
| (0xA7) | **Reserved** | | | | | | | | | |
| (0xA6) | **Reserved** | | | | | | | | | |
| (0xA5) | **Reserved** | | | | | | | | | |
| (0xA4) | **Reserved** | | | | | | | | | |
| (0xA3) | **Reserved** | | | | | | | | | |
| (0xA2) | **Reserved** | | | | | | | | | |
| (0xA1) | **Reserved** | | | | | | | | | |
| (0xA0) | **Reserved** | | | | | | | | | |
| (0x9F) | **Reserved** | | | | | | | | | |
| (0x9E) | **Reserved** | | | | | | | | | |
| (0x9D) | **OCR3CH** | OCR3C15 | OCR3C14 | OCR3C13 | OCR3C12 | OCR3C11 | OCR3C10 | OCR3C9 | OCR3C8 | page 136 |
| (0x9C) | **OCR3CL** | OCR3C7 | OCR3C6 | OCR3C5 | OCR3C4 | OCR3C3 | OCR3C2 | OCR3C1 | OCR3C0 | page 136 |
| (0x9B) | **OCR3BH** | OCR3B15 | OCR3B14 | OCR3B13 | OCR3B12 | OCR3B11 | OCR3B10 | OCR3B9 | OCR3B8 | page 136 |
| (0x9A) | **OCR3BL** | OCR3B7 | OCR3B6 | OCR3B5 | OCR3B4 | OCR3B3 | OCR3B2 | OCR3B1 | OCR3B0 | page 136 |
| (0x99) | **OCR3AH** | OCR3A15 | OCR3A14 | OCR3A13 | OCR3A12 | OCR3A11 | OCR3A10 | OCR3A9 | OCR3A8 | page 136 |
| (0x98) | **OCR3AL** | OCR3A7 | OCR3A6 | OCR3A5 | OCR3A4 | OCR3A3 | OCR3A2 | OCR3A1 | OCR3A0 | page 136 |
| (0x97) | **ICR3H** | ICR315 | ICR314 | ICR313 | ICR312 | ICR311 | ICR310 | ICR39 | ICR38 | page 137 |
| (0x96) | **ICR3L** | ICR37 | ICR36 | ICR35 | ICR34 | ICR33 | ICR32 | ICR31 | ICR30 | page 137 |
| (0x95) | **TCNT3H** | TCNT315 | TCNT314 | TCNT313 | TCNT312 | TCNT311 | TCNT310 | TCNT39 | TCNT38 | page 135 |
| (0x94) | **TCNT3L** | TCNT37 | TCNT36 | TCNT35 | TCNT34 | TCNT33 | TCNT32 | TCNT31 | TCNT30 | page 135 |
| (0x93) | **Reserved** | | | | | | | | | |
| (0x92) | **TCCR3C** | FOC3A | FOC3B | FOC3C | – | – | – | – | | page 135 |
| (0x91) | **TCCR3B** | ICNC3 | ICES3 | – | WGM33 | WGM32 | CS32 | CS31 | CS30 | page 133 |
| (0x90) | **TCCR3A** | COM3A1 | COM3A0 | COM3B1 | COM3B0 | COM3C1 | COM3C0 | WGM31 | WGM30 | page 131 |
| (0x8F) | **Reserved** | | | | | | | | | |
| (0x8E) | **Reserved** | | | | | | | | | |
| (0x8D) | **OCR1CH** | OCR1C15 | OCR1C14 | OCR1C13 | OCR1C12 | OCR1C11 | OCR1C10 | OCR1C9 | OCR1C8 | page 136 |
| (0x8C) | **OCR1CL** | OCR1C7 | OCR1C6 | OCR1C5 | OCR1C4 | OCR1C3 | OCR1C2 | OCR1C1 | OCR1C0 | page 136 |
| (0x8B) | **OCR1BH** | OCR1B15 | OCR1B14 | OCR1B13 | OCR1B12 | OCR1B11 | OCR1B10 | OCR1B9 | OCR1B8 | page 136 |
| (0x8A) | **OCR1BL** | OCR1B7 | OCR1B6 | OCR1B5 | OCR1B4 | OCR1B3 | OCR1B2 | OCR1B1 | OCR1B0 | page 136 |
| (0x89) | **OCR1AH** | OCR1A15 | OCR1A14 | OCR1A13 | OCR1A12 | OCR1A11 | OCR1A10 | OCR1A9 | OCR1A8 | page 136 |
| (0x88) | **OCR1AL** | OCR1A7 | OCR1A6 | OCR1A5 | OCR1A4 | OCR1A3 | OCR1A2 | OCR1A1 | OCR1A0 | page 136 |
| (0x87) | **ICR1H** | ICR115 | ICR114 | ICR113 | ICR112 | ICR111 | ICR110 | ICR19 | ICR18 | page 136 |
| (0x86) | **ICR1L** | ICR17 | ICR16 | ICR15 | ICR14 | ICR13 | ICR12 | ICR11 | ICR10 | page 136 |
| (0x85) | **TCNT1H** | TCNT115 | TCNT114 | TCNT113 | TCNT112 | TCNT111 | TCNT110 | TCNT19 | TCNT18 | page 135 |
| (0x84) | **TCNT1L** | TCNT17 | TCNT16 | TCNT15 | TCNT14 | TCNT13 | TCNT12 | TCNT11 | TCNT10 | page 135 |
| (0x83) | **Reserved** | | | | | | | | | |
| (0x82) | **TCCR1C** | FOC1A | FOC1B | FOC1C | – | – | – | – | – | page 134 |
| (0x81) | **TCCR1B** | ICNC1 | ICES1 | – | WGM13 | WGM12 | CS12 | CS11 | CS10 | page 133 |
| (0x80) | **TCCR1A** | COM1A1 | COM1A0 | COM1B1 | COM1B0 | COM1C1 | COM1C0 | WGM11 | WGM10 | page 131 |
| (0x7F) | **DIDR1** | – | – | – | – | – | – | AIN1D | AIN0D | page 264 |
| (0x7E) | **DIDR0** | ADC7D | ADC6D | ADC5D | ADC4D | ADC3D | ADC2D | ADC1D | ADC0D | page 283 |

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Page |
|---|---|---|---|---|---|---|---|---|---|---|
| (0x7D) | **Reserved** | | | | | | | | | |
| (0x7C) | **ADMUX** | REFS1 | REFS0 | ADLAR | MUX4 | MUX3 | MUX2 | MUX1 | MUX0 | page 279 |
| (0x7B) | **ADCSRB** | ADHSM | ACME | – | – | – | ADTS2 | ADTS1 | ADTS0 | page 283, 262 |
| (0x7A) | **ADCSRA** | ADEN | ADSC | ADATE | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 | page 281 |
| (0x79) | **ADCH** | - / ADC9 | - / ADC8 | - / ADC7 | - / ADC6 | - / ADC5 | - / ADC4 | ADC9 / ADC3 | ADC8 / ADC2 | page 282 |
| (0x78) | **ADCL** | ADC7 / ADC1 | ADC6 / ADC0 | ADC5 / - | ADC4 / - | ADC3 / - | ADC2 / - | ADC1 / - | ADC0 / | page 282 |
| (0x77) | **Reserved** | | | | | | | | | |
| (0x76) | **Reserved** | | | | | | | | | |
| (0x75) | **XMCRB** | XMBK | – | – | – | – | XMM2 | XMM1 | XMM0 | page 30 |
| (0x74) | **XMCRA** | SRE | SRL2 | SRL1 | SRL0 | SRW11 | SRW10 | SRW01 | SRW00 | page 28 |
| (0x73) | **Reserved** | | | | | | | | | |
| (0x72) | **Reserved** | | | | | | | | | |
| (0x71) | **TIMSK3** | – | – | ICIE3 | – | OCIE3C | OCIE3B | OCIE3A | TOIE3 | page 137 |
| (0x70) | **TIMSK2** | – | – | – | – | – | – | OCIE2A | TOIE2 | page 156 |
| (0x6F) | **TIMSK1** | – | – | ICIE1 | – | OCIE1C | OCIE1B | OCIE1A | TOIE1 | page 137 |
| (0x6E) | **TIMSK0** | – | – | – | – | – | – | OCIE0A | TOIE0 | page 107 |
| (0x6D) | **Reserved** | | | | | | | | | |
| (0x6C) | **Reserved** | | | | | | | | | |
| (0x6B) | **Reserved** | | | | | | | | | |
| (0x6A) | **EICRB** | ISC71 | ISC70 | ISC61 | ISC60 | ISC51 | ISC50 | ISC41 | ISC40 | page 89 |
| (0x69) | **EICRA** | ISC31 | ISC30 | ISC21 | ISC20 | ISC11 | ISC10 | ISC01 | ISC00 | page 88 |
| (0x68) | **Reserved** | | | | | | | | | |
| (0x67) | **Reserved** | | | | | | | | | |
| (0x66) | **OSCCAL** | – | CAL6 | CAL5 | CAL4 | CAL3 | CAL2 | CAL1 | CAL0 | page 39 |
| (0x65) | **Reserved** | | | | | | | | | |
| (0x64) | **Reserved** | | | | | | | | | |
| (0x63) | **Reserved** | | | | | | | | | |
| (0x62) | **Reserved** | | | | | | | | | |
| (0x61) | **CLKPR** | CLKPCE | – | – | – | CLKPS3 | CLKPS2 | CLKPS1 | CLKPS0 | page 41 |
| (0x60) | **WDTCR** | – | – | – | WDCE | WDE | WDP2 | WDP1 | WDP0 | page 54 |
| 0x3F (0x5F) | **SREG** | I | T | H | S | V | N | Z | C | page 10 |
| 0x3E (0x5E) | **SPH** | SP15 | SP14 | SP13 | SP12 | SP11 | SP10 | SP9 | SP8 | page 12 |
| 0x3D (0x5D) | **SPL** | SP7 | SP6 | SP5 | SP4 | SP3 | SP2 | SP1 | SP0 | page 12 |
| 0x3C (0x5C) | **Reserved** | | | | | | | | | |
| 0x3B (0x5B) | **RAMPZ** | – | – | – | – | – | – | – | RAMPZ0 | page 12 |
| 0x3A (0x5A) | **Reserved** | | | | | | | | | |
| 0x39 (0x59) | **Reserved** | | | | | | | | | |
| 0x38 (0x58) | **Reserved** | | | | | | | | | |
| 0x37 (0x57) | **SPMCSR** | SPMIE | RWWSB | – | RWWSRE | BLBSET | PGWRT | PGERS | SPMEN | page 316 |
| 0x36 (0x56) | **Reserved** | – | – | – | – | – | – | – | – | |
| 0x35 (0x55) | **MCUCR** | JTD | – | – | PUD | – | – | IVSEL | IVCE | page 59, 68, 293 |
| 0x34 (0x54) | **MCUSR** | – | – | – | JTRF | WDRF | BORF | EXTRF | PORF | page 51, 294 |
| 0x33 (0x53) | **SMCR** | – | – | – | – | SM2 | SM1 | SM0 | SE | page 43 |
| 0x32 (0x52) | **Reserved** | | | | | | | | | |
| 0x31 (0x51) | **OCDR** | IDRD/OCDR7 | OCDR6 | OCDR5 | OCDR4 | OCDR3 | OCDR2 | OCDR1 | OCDR0 | page 288 |
| 0x30 (0x50) | **ACSR** | ACD | ACBG | ACO | ACI | ACIE | ACIC | ACIS1 | ACIS0 | page 262 |
| 0x2F (0x4F) | **Reserved** | | | | | | | | | |
| 0x2E (0x4E) | **SPDR** | SPD7 | SPD6 | SPD5 | SPD4 | SPD3 | SPD2 | SPD1 | SPD0 | page 169 |
| 0x2D (0x4D) | **SPSR** | SPIF | WCOL | – | – | – | – | – | SPI2X | page 168 |
| 0x2C (0x4C) | **SPCR** | SPIE | SPE | DORD | MSTR | CPOL | CPHA | SPR1 | SPR0 | page 167 |
| 0x2B (0x4B) | **GPIOR2** | GPIOR27 | GPIOR26 | GPIOR25 | GPIOR24 | GPIOR23 | GPIOR22 | GPIOR21 | GPIOR20 | page 33 |
| 0x2A (0x4A) | **GPIOR1** | GPIOR17 | GPIOR16 | GPIOR15 | GPIOR14 | GPIOR13 | GPIOR12 | GPIOR11 | GPIOR10 | page 33 |
| 0x29 (0x49) | **Reserved** | | | | | | | | | |
| 0x28 (0x48) | **Reserved** | | | | | | | | | |
| 0x27 (0x47) | **OCR0A** | OCR0A7 | OCR0A6 | OCR0A5 | OCR0A4 | OCR0A3 | OCR0A2 | OCR0A1 | OCR0A0 | page 107 |
| 0x26 (0x46) | **TCNT0** | TCNT07 | TCNT06 | TCNT05 | TCNT04 | TCNT03 | TCNT02 | TCNT01 | TCNT00 | page 106 |
| 0x25 (0x45) | **Reserved** | | | | | | | | | |
| 0x24 (0x44) | **TCCR0A** | FOC0A | WGM00 | COM0A1 | COM0A0 | WGM01 | CS02 | CS01 | CS00 | page 104 |
| 0x23 (0x43) | **GTCCR** | TSM | – | – | – | – | – | PSR2 | PSR310 | page 92, 159 |
| 0x22 (0x42) | **EEARH** | – | – | – | – | EEAR11 | EEAR10 | EEAR9 | EEAR8 | page 19 |
| 0x21 (0x41) | **EEARL** | EEAR7 | EEAR6 | EEAR5 | EEAR4 | EEAR3 | EEAR2 | EEAR1 | EEAR0 | page 19 |
| 0x20 (0x40) | **EEDR** | EEDR7 | EEDR6 | EEDR5 | EEDR4 | EEDR3 | EEDR2 | EEDR1 | EEDR0 | page 19 |
| 0x1F (0x3F) | **EECR** | – | – | – | – | EERIE | EEMWE | EEWE | EERE | page 20 |
| 0x1E (0x3E) | **GPIOR0** | GPIOR07 | GPIOR06 | GPIOR05 | GPIOR04 | GPIOR03 | GPIOR02 | GPIOR01 | GPIOR00 | page 33 |
| 0x1D (0x3D) | **EIMSK** | INT7 | INT6 | INT5 | INT4 | INT3 | INT2 | INT1 | INT0 | page 90 |
| 0x1C (0x3C) | **EIFR** | INTF7 | INTF6 | INTF5 | INTF4 | INTF3 | INTF2 | INTF1 | INTF0 | page 90 |

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Page |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x1B (0x3B) | **Reserved** | | | | | | | | | |
| 0x1A (0x3A) | **Reserved** | | | | | | | | | |
| 0x19 (0x39) | **Reserved** | | | | | | | | | |
| 0x18 (0x38) | **TIFR3** | – | – | ICF3 | – | OCF3C | OCF3B | OCF3A | TOV3 | page 138 |
| 0x17 (0x37) | **TIFR2** | – | – | – | – | – | – | OCF2A | TOV2 | page 157 |
| 0x16 (0x36) | **TIFR1** | – | – | ICF1 | – | OCF1C | OCF1B | OCF1A | TOV1 | page 138 |
| 0x15 (0x35) | **TIFR0** | – | – | – | – | – | – | OCF0A | TOV0 | page 107 |
| 0x14 (0x34) | **PORTG** | – | – | – | PORTG4 | PORTG3 | PORTG2 | PORTG1 | PORTG0 | page 87 |
| 0x13 (0x33) | **DDRG** | – | – | – | DDG4 | DDG3 | DDG2 | DDG1 | DDG0 | page 87 |
| 0x12 (0x32) | **PING** | – | – | – | PING4 | PING3 | PING2 | PING1 | PING0 | page 87 |
| 0x11 (0x31) | **PORTF** | PORTF7 | PORTF6 | PORTF5 | PORTF4 | PORTF3 | PORTF2 | PORTF1 | PORTF0 | page 86 |
| 0x10 (0x30) | **DDRF** | DDF7 | DDF6 | DDF5 | DDF4 | DDF3 | DDF2 | DDF1 | DDF0 | page 86 |
| 0x0F (0x2F) | **PINF** | PINF7 | PINF6 | PINF5 | PINF4 | PINF3 | PINF2 | PINF1 | PINF0 | page 87 |
| 0x0E (0x2E) | **PORTE** | PORTE7 | PORTE6 | PORTE5 | PORTE4 | PORTE3 | PORTE2 | PORTE1 | PORTE0 | page 86 |
| 0x0D (0x2D) | **DDRE** | DDE7 | DDE6 | DDE5 | DDE4 | DDE3 | DDE2 | DDE1 | DDE0 | page 86 |
| 0x0C (0x2C) | **PINE** | PINE7 | PINE6 | PINE5 | PINE4 | PINE3 | PINE2 | PINE1 | PINE0 | page 86 |
| 0x0B (0x2B) | **PORTD** | PORTD7 | PORTD6 | PORTD5 | PORTD4 | PORTD3 | PORTD2 | PORTD1 | PORTD0 | page 86 |
| 0x0A (0x2A) | **DDRD** | DDD7 | DDD6 | DDD5 | DDD4 | DDD3 | DDD2 | DDD1 | DDD0 | page 86 |
| 0x09 (0x29) | **PIND** | PIND7 | PIND6 | PIND5 | PIND4 | PIND3 | PIND2 | PIND1 | PIND0 | page 86 |
| 0x08 (0x28) | **PORTC** | PORTC7 | PORTC6 | PORTC5 | PORTC4 | PORTC3 | PORTC2 | PORTC1 | PORTC0 | page 85 |
| 0x07 (0x27) | **DDRC** | DDC7 | DDC6 | DDC5 | DDC4 | DDC3 | DDC2 | DDC1 | DDC0 | page 85 |
| 0x06 (0x26) | **PINC** | PINC7 | PINC6 | PINC5 | PINC4 | PINC3 | PINC2 | PINC1 | PINC0 | page 86 |
| 0x05 (0x25) | **PORTB** | PORTB7 | PORTB6 | PORTB5 | PORTB4 | PORTB3 | PORTB2 | PORTB1 | PORTB0 | page 85 |
| 0x04 (0x24) | **DDRB** | DDB7 | DDB6 | DDB5 | DDB4 | DDB3 | DDB2 | DDB1 | DDB0 | page 85 |
| 0x03 (0x23) | **PINB** | PINB7 | PINB6 | PINB5 | PINB4 | PINB3 | PINB2 | PINB1 | PINB0 | page 85 |
| 0x02 (0x22) | **PORTA** | PORTA7 | PORTA6 | PORTA5 | PORTA4 | PORTA3 | PORTA2 | PORTA1 | PORTA0 | page 85 |
| 0x01 (0x21) | **DDRA** | DDA7 | DDA6 | DDA5 | DDA4 | DDA3 | DDA2 | DDA1 | DDA0 | page 85 |
| 0x00 (0x20) | **PINA** | PINA7 | PINA6 | PINA5 | PINA4 | PINA3 | PINA2 | PINA1 | PINA0 | page 85 |

Note:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O Registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such status flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The AT90CAN128 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

# Instruction Set Summary

| Mnemonics | Operands | Description | Operation | Flags | #Clocks |
|---|---|---|---|---|---|
| **ARITHMETIC AND LOGIC INSTRUCTIONS** | | | | | |
| ADD | Rd, Rr | Add two Registers | Rd ← Rd + Rr | Z,C,N,V,H | 1 |
| ADC | Rd, Rr | Add with Carry two Registers | Rd ← Rd + Rr + C | Z,C,N,V,H | 1 |
| ADIW | Rdl,K | Add Immediate to Word | Rdh:Rdl ← Rdh:Rdl + K | Z,C,N,V,S | 2 |
| SUB | Rd, Rr | Subtract two Registers | Rd ← Rd - Rr | Z,C,N,V,H | 1 |
| SUBI | Rd, K | Subtract Constant from Register | Rd ← Rd - K | Z,C,N,V,H | 1 |
| SBC | Rd, Rr | Subtract with Carry two Registers | Rd ← Rd - Rr - C | Z,C,N,V,H | 1 |
| SBCI | Rd, K | Subtract with Carry Constant from Reg. | Rd ← Rd - K - C | Z,C,N,V,H | 1 |
| SBIW | Rdl,K | Subtract Immediate from Word | Rdh:Rdl ← Rdh:Rdl - K | Z,C,N,V,S | 2 |
| AND | Rd, Rr | Logical AND Registers | Rd ← Rd • Rr | Z,N,V | 1 |
| ANDI | Rd, K | Logical AND Register and Constant | Rd ← Rd • K | Z,N,V | 1 |
| OR | Rd, Rr | Logical OR Registers | Rd ← Rd v Rr | Z,N,V | 1 |
| ORI | Rd, K | Logical OR Register and Constant | Rd ← Rd v K | Z,N,V | 1 |
| EOR | Rd, Rr | Exclusive OR Registers | Rd ← Rd ⊕ Rr | Z,N,V | 1 |
| COM | Rd | One's Complement | Rd ← 0xFF – Rd | Z,C,N,V | 1 |
| NEG | Rd | Two's Complement | Rd ← 0x00 – Rd | Z,C,N,V,H | 1 |
| SBR | Rd,K | Set Bit(s) in Register | Rd ← Rd v K | Z,N,V | 1 |
| CBR | Rd,K | Clear Bit(s) in Register | Rd ← Rd • (0xFF - K) | Z,N,V | 1 |
| INC | Rd | Increment | Rd ← Rd + 1 | Z,N,V | 1 |
| DEC | Rd | Decrement | Rd ← Rd – 1 | Z,N,V | 1 |
| TST | Rd | Test for Zero or Minus | Rd ← Rd • Rd | Z,N,V | 1 |
| CLR | Rd | Clear Register | Rd ← Rd ⊕ Rd | Z,N,V | 1 |
| SER | Rd | Set Register | Rd ← 0xFF | None | 1 |
| MUL | Rd, Rr | Multiply Unsigned | R1:R0 ← Rd x Rr | Z,C | 2 |
| MULS | Rd, Rr | Multiply Signed | R1:R0 ← Rd x Rr | Z,C | 2 |
| MULSU | Rd, Rr | Multiply Signed with Unsigned | R1:R0 ← Rd x Rr | Z,C | 2 |
| FMUL | Rd, Rr | Fractional Multiply Unsigned | R1:R0 ← (Rd x Rr) << 1 | Z,C | 2 |
| FMULS | Rd, Rr | Fractional Multiply Signed | R1:R0 ← (Rd x Rr) << 1 | Z,C | 2 |
| FMULSU | Rd, Rr | Fractional Multiply Signed with Unsigned | R1:R0 ← (Rd x Rr) << 1 | Z,C | 2 |
| **BRANCH INSTRUCTIONS** | | | | | |
| RJMP | k | Relative Jump | PC ← PC + k + 1 | None | 2 |
| IJMP | | Indirect Jump to (Z) | PC ← Z | None | 2 |
| JMP | k | Direct Jump | PC ← k | None | 3 |
| RCALL | k | Relative Subroutine Call | PC ← PC + k + 1 | None | 3 |
| ICALL | | Indirect Call to (Z) | PC ← Z | None | 3 |
| CALL | k | Direct Subroutine Call | PC ← k | None | 4 |
| RET | | Subroutine Return | PC ← STACK | None | 4 |
| RETI | | Interrupt Return | PC ← STACK | I | 4 |
| CPSE | Rd,Rr | Compare, Skip if Equal | if (Rd = Rr) PC ← PC + 2 or 3 | None | 1/2/3 |
| CP | Rd,Rr | Compare | Rd – Rr | Z, N,V,C,H | 1 |
| CPC | Rd,Rr | Compare with Carry | Rd – Rr – C | Z, N,V,C,H | 1 |
| CPI | Rd,K | Compare Register with Immediate | Rd – K | Z, N,V,C,H | 1 |
| SBRC | Rr, b | Skip if Bit in Register Cleared | if (Rr(b)=0) PC ← PC + 2 or 3 | None | 1/2/3 |
| SBRS | Rr, b | Skip if Bit in Register is Set | if (Rr(b)=1) PC ← PC + 2 or 3 | None | 1/2/3 |
| SBIC | P, b | Skip if Bit in I/O Register Cleared | if (P(b)=0) PC ← PC + 2 or 3 | None | 1/2/3 |
| SBIS | P, b | Skip if Bit in I/O Register is Set | if (P(b)=1) PC ← PC + 2 or 3 | None | 1/2/3 |
| BRBS | s, k | Branch if Status Flag Set | if (SREG(s) = 1) then PC←PC+k + 1 | None | 1/2 |
| BRBC | s, k | Branch if Status Flag Cleared | if (SREG(s) = 0) then PC←PC+k + 1 | None | 1/2 |
| BREQ | k | Branch if Equal | if (Z = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRNE | k | Branch if Not Equal | if (Z = 0) then PC ← PC + k + 1 | None | 1/2 |
| BRCS | k | Branch if Carry Set | if (C = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRCC | k | Branch if Carry Cleared | if (C = 0) then PC ← PC + k + 1 | None | 1/2 |
| BRSH | k | Branch if Same or Higher | if (C = 0) then PC ← PC + k + 1 | None | 1/2 |
| BRLO | k | Branch if Lower | if (C = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRMI | k | Branch if Minus | if (N = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRPL | k | Branch if Plus | if (N = 0) then PC ← PC + k + 1 | None | 1/2 |
| BRGE | k | Branch if Greater or Equal, Signed | if (N ⊕ V= 0) then PC ← PC + k + 1 | None | 1/2 |
| BRLT | k | Branch if Less Than Zero, Signed | if (N ⊕ V= 1) then PC ← PC + k + 1 | None | 1/2 |
| BRHS | k | Branch if Half Carry Flag Set | if (H = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRHC | k | Branch if Half Carry Flag Cleared | if (H = 0) then PC ← PC + k + 1 | None | 1/2 |
| BRTS | k | Branch if T Flag Set | if (T = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRTC | k | Branch if T Flag Cleared | if (T = 0) then PC ← PC + k + 1 | None | 1/2 |
| BRVS | k | Branch if Overflow Flag is Set | if (V = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRVC | k | Branch if Overflow Flag is Cleared | if (V = 0) then PC ← PC + k + 1 | None | 1/2 |

| Mnemonics | Operands | Description | Operation | Flags | #Clocks |
|---|---|---|---|---|---|
| BRIE | k | Branch if Interrupt Enabled | if ( I = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRID | k | Branch if Interrupt Disabled | if ( I = 0) then PC ← PC + k + 1 | None | 1/2 |
| **BIT AND BIT-TEST INSTRUCTIONS** | | | | | |
| SBI | P,b | Set Bit in I/O Register | I/O(P,b) ← 1 | None | 2 |
| CBI | P,b | Clear Bit in I/O Register | I/O(P,b) ← 0 | None | 2 |
| LSL | Rd | Logical Shift Left | Rd(n+1) ← Rd(n), Rd(0) ← 0 | Z,C,N,V | 1 |
| LSR | Rd | Logical Shift Right | Rd(n) ← Rd(n+1), Rd(7) ← 0 | Z,C,N,V | 1 |
| ROL | Rd | Rotate Left Through Carry | Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7) | Z,C,N,V | 1 |
| ROR | Rd | Rotate Right Through Carry | Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0) | Z,C,N,V | 1 |
| ASR | Rd | Arithmetic Shift Right | Rd(n) ← Rd(n+1), n=0..6 | Z,C,N,V | 1 |
| SWAP | Rd | Swap Nibbles | Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0) | None | 1 |
| BSET | s | Flag Set | SREG(s) ← 1 | SREG(s) | 1 |
| BCLR | s | Flag Clear | SREG(s) ← 0 | SREG(s) | 1 |
| BST | Rr, b | Bit Store from Register to T | T ← Rr(b) | T | 1 |
| BLD | Rd, b | Bit load from T to Register | Rd(b) ← T | None | 1 |
| SEC | | Set Carry | C ← 1 | C | 1 |
| CLC | | Clear Carry | C ← 0 | C | 1 |
| SEN | | Set Negative Flag | N ← 1 | N | 1 |
| CLN | | Clear Negative Flag | N ← 0 | N | 1 |
| SEZ | | Set Zero Flag | Z ← 1 | Z | 1 |
| CLZ | | Clear Zero Flag | Z ← 0 | Z | 1 |
| SEI | | Global Interrupt Enable | I ← 1 | I | 1 |
| CLI | | Global Interrupt Disable | I ← 0 | I | 1 |
| SES | | Set Signed Test Flag | S ← 1 | S | 1 |
| CLS | | Clear Signed Test Flag | S ← 0 | S | 1 |
| SEV | | Set Twos Complement Overflow. | V ← 1 | V | 1 |
| CLV | | Clear Twos Complement Overflow | V ← 0 | V | 1 |
| SET | | Set T in SREG | T ← 1 | T | 1 |
| CLT | | Clear T in SREG | T ← 0 | T | 1 |
| SEH | | Set Half Carry Flag in SREG | H ← 1 | H | 1 |
| CLH | | Clear Half Carry Flag in SREG | H ← 0 | H | 1 |
| **DATA TRANSFER INSTRUCTIONS** | | | | | |
| MOV | Rd, Rr | Move Between Registers | Rd ← Rr | None | 1 |
| MOVW | Rd, Rr | Copy Register Word | Rd+1:Rd ← Rr+1:Rr | None | 1 |
| LDI | Rd, K | Load Immediate | Rd ← K | None | 1 |
| LD | Rd, X | Load Indirect | Rd ← (X) | None | 2 |
| LD | Rd, X+ | Load Indirect and Post-Inc. | Rd ← (X), X ← X + 1 | None | 2 |
| LD | Rd, - X | Load Indirect and Pre-Dec. | X ← X - 1, Rd ← (X) | None | 2 |
| LD | Rd, Y | Load Indirect | Rd ← (Y) | None | 2 |
| LD | Rd, Y+ | Load Indirect and Post-Inc. | Rd ← (Y), Y ← Y + 1 | None | 2 |
| LD | Rd, - Y | Load Indirect and Pre-Dec. | Y ← Y - 1, Rd ← (Y) | None | 2 |
| LDD | Rd,Y+q | Load Indirect with Displacement | Rd ← (Y + q) | None | 2 |
| LD | Rd, Z | Load Indirect | Rd ← (Z) | None | 2 |
| LD | Rd, Z+ | Load Indirect and Post-Inc. | Rd ← (Z), Z ← Z+1 | None | 2 |
| LD | Rd, -Z | Load Indirect and Pre-Dec. | Z ← Z - 1, Rd ← (Z) | None | 2 |
| LDD | Rd, Z+q | Load Indirect with Displacement | Rd ← (Z + q) | None | 2 |
| LDS | Rd, k | Load Direct from SRAM | Rd ← (k) | None | 2 |
| ST | X, Rr | Store Indirect | (X) ← Rr | None | 2 |
| ST | X+, Rr | Store Indirect and Post-Inc. | (X) ← Rr, X ← X + 1 | None | 2 |
| ST | - X, Rr | Store Indirect and Pre-Dec. | X ← X - 1, (X) ← Rr | None | 2 |
| ST | Y, Rr | Store Indirect | (Y) ← Rr | None | 2 |
| ST | Y+, Rr | Store Indirect and Post-Inc. | (Y) ← Rr, Y ← Y + 1 | None | 2 |
| ST | - Y, Rr | Store Indirect and Pre-Dec. | Y ← Y - 1, (Y) ← Rr | None | 2 |
| STD | Y+q,Rr | Store Indirect with Displacement | (Y + q) ← Rr | None | 2 |
| ST | Z, Rr | Store Indirect | (Z) ← Rr | None | 2 |
| ST | Z+, Rr | Store Indirect and Post-Inc. | (Z) ← Rr, Z ← Z + 1 | None | 2 |
| ST | -Z, Rr | Store Indirect and Pre-Dec. | Z ← Z - 1, (Z) ← Rr | None | 2 |
| STD | Z+q,Rr | Store Indirect with Displacement | (Z + q) ← Rr | None | 2 |
| STS | k, Rr | Store Direct to SRAM | (k) ← Rr | None | 2 |
| LPM | | Load Program Memory | R0 ← (Z) | None | 3 |
| LPM | Rd, Z | Load Program Memory | Rd ← (Z) | None | 3 |
| LPM | Rd, Z+ | Load Program Memory and Post-Inc | Rd ← (Z), Z ← Z+1 | None | 3 |
| ELPM | | Extended Load Program Memory | R0 ← (RAMPZ:Z) | None | 3 |
| ELPM | Rd, Z | Extended Load Program Memory | Rd ← (RAMPZ:Z) | None | 3 |
| ELPM | Rd, Z+ | Extended Load Program Memory and Post-Inc | Rd ← (RAMPZ:Z), RAMPZ:Z ← RAMPZ:Z+1 | None | 3 |
| SPM | | Store Program Memory | (Z) ← R1:R0 | None | - |

| Mnemonics | Operands | Description | Operation | Flags | #Clocks |
|---|---|---|---|---|---|
| IN | Rd, P | In Port | Rd ← P | None | 1 |
| OUT | P, Rr | Out Port | P ← Rr | None | 1 |
| PUSH | Rr | Push Register on Stack | STACK ← Rr | None | 2 |
| POP | Rd | Pop Register from Stack | Rd ← STACK | None | 2 |
| **MCU CONTROL INSTRUCTIONS** | | | | | |
| NOP | | No Operation | | None | 1 |
| SLEEP | | Sleep | (see specific descr. for Sleep function) | None | 1 |
| WDR | | Watchdog Reset | (see specific descr. for WDR/timer) | None | 1 |
| BREAK | | Break | For On-chip Debug Only | None | N/A |

## Ordering Information

| Ordering Code | Speed (MHz) | Power Supply (V) | Package | Operation Range | Product Marking |
|---|---|---|---|---|---|
| AT90CAN128-16AI | 16 | 2.7 - 5.5 | 64A | Industrial (-40° to +85°C) | AT90CAN128-IL |
| AT90CAN128-16MI | 16 | 2.7 - 5.5 | 64M1 | Industrial (-40° to +85°C) | AT90CAN128-IL |
| AT90CAN128-16AU | 16 | 2.7 - 5.5 | 64A | Industrial (-40° to +85°C) Green | AT90CAN128-UL |
| AT90CAN128-16MU | 16 | 2.7 - 5.5 | 64M1 | Industrial (-40° to +85°C) Green | AT90CAN128-UL |

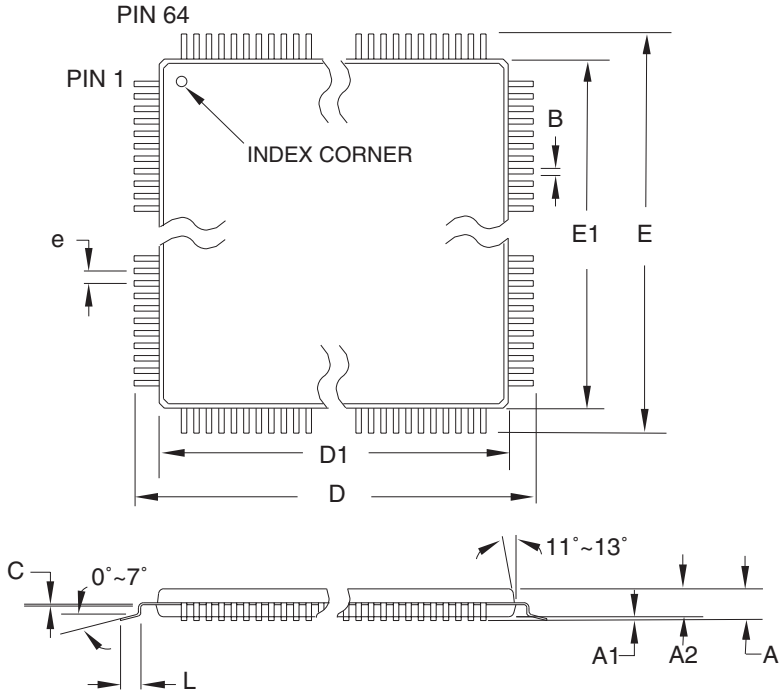
Note: This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.

## Packaging Information

| Package Type | |
|---|---|
| **64A** | 64-Lead, Thin (1.0 mm) Plastic Gull Wing Quad Flat Package (TQFP) |
| **64M1** | 64-Lead, Quad Flat No lead (QFN) |

## TQFP64

### 64 LEADS Thin Quad Flat Package

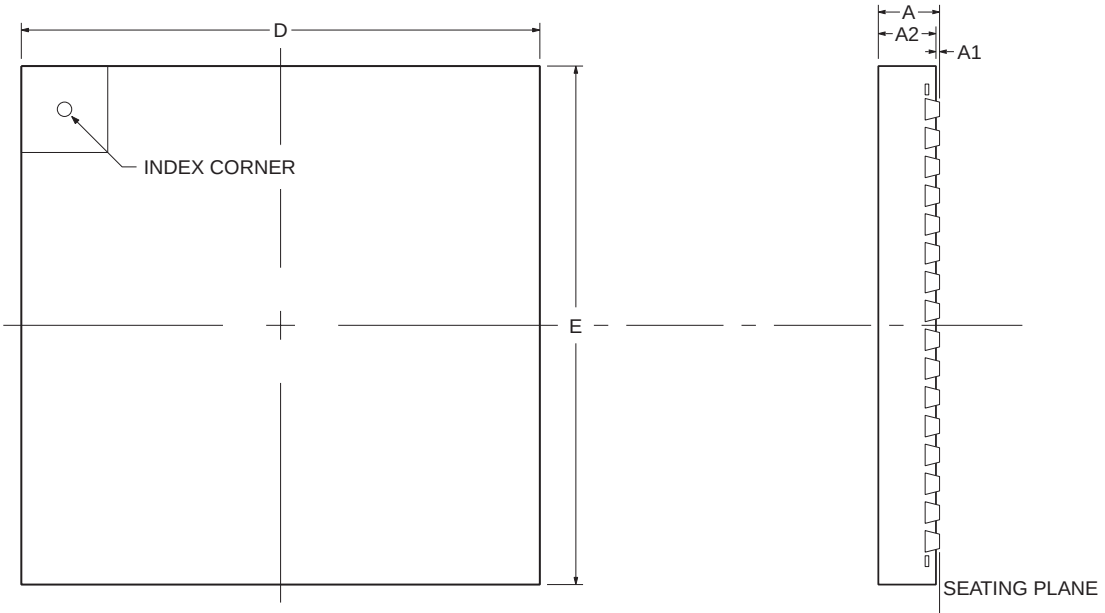


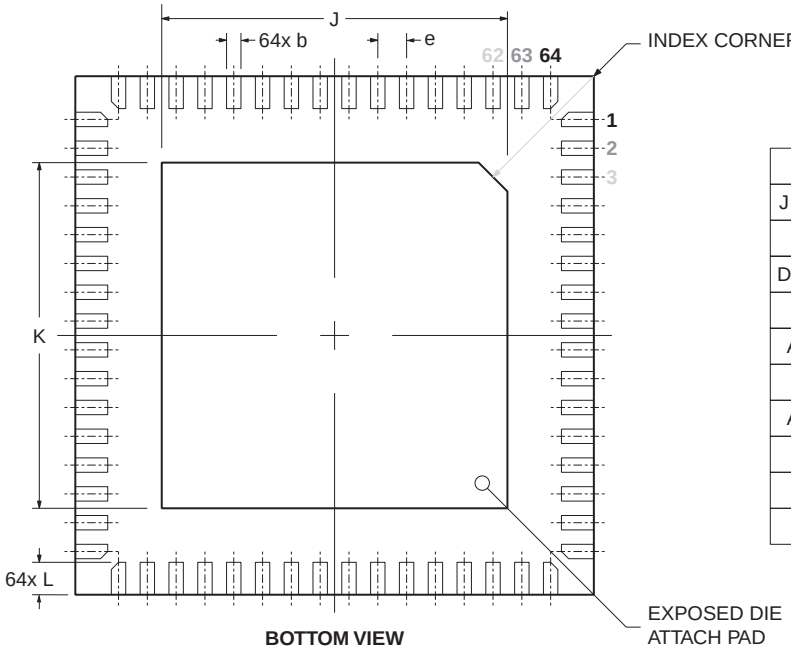

| | MM | | | INCH | | |
|---|---|---|---|---|---|---|
| SYMBOL | MIN | NOM | MAX | MIN | NOM | MAX |
| A | – | – | 1.20 | – | – | . 047 |
| A1 | 0.05 | – | 0.15 | . 002 | – | . 006 |
| A2 | 0.95 | 1.00 | 1.05 | . 037 | . 039 | . 041 |
| D | 15.75 | 16.00 | 16.25 | . 620 | . 630 | . 640 |
| D1 (2) | 13.90 | 14.00 | 14.10 | . 547 | . 551 | . 555 |
| E | 15.75 | 16.00 | 16.25 | . 620 | . 630 | . 640 |
| E1 (2) | 13.90 | 14.00 | 14.10 | . 547 | . 551 | . 555 |
| B | 0.30 | – | 0.45 | . 012 | – | . 018 |
| C | 0.09 | – | 0.20 | . 004 | – | . 008 |
| L | 0.45 | – | 0.75 | . 018 | – | . 030 |
| e | 0.80 TYP | | | . 0315 TYP | | |

Notes:
1. This package conforms to JEDEC reference MS-026, Variation AEB.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25 mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
3. Lead coplanarity is 0.10 mm maximum.

## QFN64

**64 LEADS Quad Flat No lead**




| | MM | | | INCH | | |
|---|---|---|---|---|---|---|
| | MIN | NOM | MAX | MIN | NOM | MAX |
| A | 0.80 | | 1.00 | . 031 | | . 039 |
| J / K | 6.47 | 6.57 | 6.67 | . 255 | . 259 | . 263 |
| | | | | | | |
| D / E | 9.00 BSC | | | . 354 BSC | | |
| | | | | | | |
| A1 | 0.00 | | 0.05 | . 000 | | . 002 |
| N | 64 | | | | | |
| A2 | 0.75 | | 1.00 | . 029 | | . 039 |
| e | 0.50 BSC | | | . 020 BSC | | |
| L | 0.40 | 0.45 | 0.50 | . 016 | . 018 | . 020 |
| b | 0.17 | 0.25 | 0.27 | . 007 | . 010 | . 011 |

Note: Compliant JEDEC MO-220

## Errata

The revision letter in this section refers to the revision of the AT90CAN128 device.

## Rev C

*Rev C (Part marked: M90CAN128 - I )*

- **Asynchronous Timer-2 wakes up without interrupt**
- **SPI programming timing**

**1. Asynchronous Timer-2 wakes up without interrupt**

The asynchronous timer can wake from sleep without giving interrupt. The error only occurs if the interrupt flag(s) is cleared by software less than 4 cycles before going to sleep and this clear is done exactly when it is supposed to be set (compare match or overflow). Only the interrupts flags are affected by the clear, not the signal witch is used to wake up the part.

**Problem Fix/Workaround**

No known workaround, try to lock the code to avoid such a timing.

**2. SPI programming timing**

When the fuse high byte or the extended fuse byte has been written, it is necessary to wait the end of the programming using “Poll RDY/$\overline{\text{BSY}}$” instruction. If this instruction is entered too speedily after the “Write Fuse” instruction, the fuse low byte is written instead of high fuse /extended fuse byte.

**Problem Fix/Workaround**

Wait sometime before applying the “Poll RDY/$\overline{\text{BSY}}$” instruction. For 8MHz system clock, waiting 1 µs is sufficient.

## Rev A & B

*- Rev A (Part marked: M128CAN11 - EL)*
*- Rev B (Part marked: 90CAN128 - EL)*

- **Sporadic CAN error frames**
- **Spike on TWI pins when TWI is enabled**
- **ADC differential gain error with x1 & x10 amplification**
- **Asynchronous Timer-2 wakes up without interrupt**
- **SPI programming timing**
- **IDCODE masks data from TDI input**

**6. Sporadic CAN error frames**

When BRP = 0 the CAN controller may desynchronize and send one error frame to ask for the retransmission of the incoming frame, even though it had no error.
This is likely to occur with BRP = 0 after long inter frame periods without synchronization (low bus load). The CAN macro can still properly synchronize on frames following the error.

**Problem Fix/Workaround**

Set BRP greater than 0 in CANBT1.

**5. Spike on TWI pins when TWI is enabled**

100 ns negative spike occurs on SDA and SCL pins when TWI is enabled.

**Problem Fix/Workaround**
No known workaround, enable AT90CAN128 TWI first versus the others nodes of the TWI network.

**4. ADC differential gain error with x1 & x10 amplification**
Gain error of - 4 lsb has been characterized on the part.

**Problem Fix/Workaround**
Software adjustment.

**3. Asynchronous Timer-2 wakes up without interrupt**
The asynchronous timer can wake from sleep without giving interrupt. The error only occurs if the interrupt flag(s) is cleared by software less than 4 cycles before going to sleep and this clear is done exactly when it is supposed to be set (compare match or overflow). Only the interrupts flags are affected by the clear, not the signal witch is used to wake up the part.

**Problem Fix/Workaround**
No known workaround, try to lock the code to avoid such a timing.

**2. SPI programming timing**
When the fuse high byte or the extended fuse byte has been written, it is necessary to wait the end of the programming using “Poll RDY/$\overline{\text{BSY}}$” instruction. If this instruction is entered too speedily after the “Write Fuse” instruction, the fuse low byte is written instead of high fuse /extended fuse byte.

**Problem Fix/Workaround**
Wait sometime before applying the “Poll RDY/$\overline{\text{BSY}}$” instruction. For 8MHz system clock, waiting 1 µs is sufficient.

**1. IDCODE masks data from TDI input**
The JTAG instruction IDCODE is not working correctly. Data to succeeding devices are replaced by all-ones during Update-DR.

**Problem Fix / Workaround**

– If AT90CAN128 is the only device in the scan chain, the problem is not visible.
– Select the Device ID Register of the AT90CAN128 by issuing the IDCODE instruction or by entering the Test-Logic-Reset state of the TAP controller to read out the contents of its Device ID Register and possibly data from succeeding devices of the scan chain. Issue the BYPASS instruction to the AT90CAN128 while reading the Device ID Registers of preceding devices of the boundary scan chain.
– If the Device IDs of all devices in the boundary scan chain must be captured simultaneously, the AT90CAN128 must be the first device in the chain.

# Datasheet Change Log for AT90CAN128

Please note that the referring page numbers in this section are referring to this document. The referring revision in this section are referring to the document revision.

## Changes from 4250D-07/04 to 4250E-12/04

1. Information on PHS2 segment of CAN bit timing (See “Bit Timing” on page 235 , see “Baud Rate” on page 236 and see “CAN Bit Timing Register 3 - CANBT3” on page 252).
2. Information on capacitors when using 32.768 KHz crystal on XTAL1 & 2 and TOSC1 & 2 pins.
3. Correction Table 135. "Serial Programming Instruction Set" on page 340.
4. Updated RESET, BOD & Bandgap characteristics in section “System Control and Reset” on page 47.
5. Added curves in section “AT90CAN128 Typical Characteristics” on page 374.
6. Updated characteristics in section “Electrical Characteristics(1)” on page 355.
7. Updated Errata device REV C.
8. Changed Ordering Information.

## Changes from 4250C-03/04 to 4250D-07/04

1. Updated Errata device REV A & B.

## Changes from 4250B-02/04 to 4250C-03/04

1. Changed part number to AT90CAN128.
2. Changed Ordering Information.

## Changes from 4250A-10/03 to 4250B-02/04

1. Modified Product Ordering Information.
2. Added Errata section.
3. Updated Boundary-scan IEEE 1149.1 (JTAG) chapter.
4. Updated assembler examples.

## Table of Contents


***Features*** ..... ***1***

***Description*** ..... ***2***

***Disclaimer*** ..... ***2***

Block Diagram ..... 3

***Pin Configurations*** ..... ***4***

Pin Descriptions ..... 6

About Code Examples ..... 7

***AVR CPU Core*** ..... ***8***

Introduction ..... 8

Architectural Overview ..... 8

ALU – Arithmetic Logic Unit ..... 9

Status Register ..... 10

General Purpose Register File ..... 11

Stack Pointer ..... 12

Instruction Execution Timing ..... 13

Reset and Interrupt Handling ..... 13

***Memories*** ..... ***16***

In-System Reprogrammable Flash Program Memory ..... 16

SRAM Data Memory ..... 17

EEPROM Data Memory ..... 19

I/O Memory ..... 24

External Memory Interface ..... 24

General Purpose I/O Registers ..... 33

***System Clock*** ..... ***34***

Clock Systems and their Distribution ..... 34

Clock Sources ..... 35

Default Clock Source ..... 35

Crystal Oscillator ..... 36

Low-frequency Crystal Oscillator ..... 37

Calibrated Internal RC Oscillator ..... 38

External Clock ..... 39

Clock Output Buffer ..... 40

Timer/Counter2 Oscillator ..... 40

System Clock Prescaler ..... 41

***Power Management and Sleep Modes*** ..... ***43***

Idle Mode ..... 44

ADC Noise Reduction Mode ..... 44

Power-down Mode ..... 44
Power-save Mode ..... 44
Standby Mode ..... 45
Minimizing Power Consumption ..... 45

***System Control and Reset ..... 47***
Internal Voltage Reference ..... 52
Watchdog Timer ..... 53
Timed Sequences for Changing the Configuration of the Watchdog Timer ..... 55

***Interrupts ..... 56***
Interrupt Vectors in AT90CAN128 ..... 56
Moving Interrupts Between Application and Boot Space ..... 59

***I/O-Ports ..... 61***
Introduction ..... 61
Ports as General Digital I/O ..... 62
Alternate Port Functions ..... 67
Register Description for I/O-Ports ..... 85

***External Interrupts ..... 88***

***Timer/Counter3/1/0 Prescalers ..... 91***
Overview ..... 91
Timer/Counter0/1/3 Prescalers Register Description ..... 92

***8-bit Timer/Counter0 with PWM ..... 94***
Features ..... 94
Overview ..... 94
Timer/Counter Clock Sources ..... 95
Counter Unit ..... 95
Output Compare Unit ..... 96
Compare Match Output Unit ..... 98
Modes of Operation ..... 99
Timer/Counter Timing Diagrams ..... 102
8-bit Timer/Counter Register Description ..... 104

***16-bit Timer/Counter (Timer/Counter1 and Timer/Counter3) ..... 108***
Features ..... 108
Overview ..... 108
Accessing 16-bit Registers ..... 111
Timer/Counter Clock Sources ..... 115
Counter Unit ..... 115
Input Capture Unit ..... 116
Output Compare Units ..... 118
Compare Match Output Unit ..... 120

Modes of Operation ..... 121
Timer/Counter Timing Diagrams..... 128
16-bit Timer/Counter Register Description ..... 131

***8-bit Timer/Counter2 with PWM and Asynchronous Operation ... 140***
Features..... 140
Overview..... 140
Timer/Counter Clock Sources..... 142
Counter Unit..... 142
Output Compare Unit..... 143
Compare Match Output Unit..... 145
Modes of Operation ..... 145
Timer/Counter Timing Diagrams..... 150
8-bit Timer/Counter Register Description ..... 151
Asynchronous operation of the Timer/Counter2..... 154
Timer/Counter2 Prescaler..... 158

***Output Compare Modulator - OCM ..... 160***
Overview..... 160
Description..... 160

***Serial Peripheral Interface – SPI..... 162***
Features..... 162
$\overline{SS}$ Pin Functionality..... 167
Data Modes ..... 170

***USART (USART0 and USART1)..... 171***
Features..... 171
Dual USART ..... 171
Overview..... 172
Clock Generation..... 173
Serial Frame ..... 175
USART Initialization..... 176
Data Transmission – USART Transmitter ..... 178
Data Reception – USART Receiver..... 180
Asynchronous Data Reception ..... 184
Multi-processor Communication Mode ..... 187
USART Register Description ..... 189
Examples of Baud Rate Setting..... 195

***Two-wire Serial Interface ..... 199***
Features..... 199
Two-wire Serial Interface Bus Definition..... 199
Data Transfer and Frame Format..... 200
Multi-master Bus Systems, Arbitration and Synchronization ..... 202
Overview of the TWI Module ..... 205

TWI Register Description ..... 207
Using the TWI ..... 211
Transmission Modes ..... 214
Multi-master Systems and Arbitration ..... 227

***Controller Area Network - CAN ..... 229***
Features ..... 229
CAN Protocol ..... 229
CAN Controller ..... 234
CAN Channel ..... 235
Message Objects ..... 237
CAN Timer ..... 240
Error Management ..... 241
Interrupts ..... 243
CAN Register Description ..... 245
General CAN Registers ..... 246
MOb Registers ..... 254
Examples of CAN Baud Rate Setting ..... 260

***Analog Comparator ..... 262***
Overview ..... 262
Analog Comparator Register Description ..... 262
Analog Comparator Multiplexed Input ..... 264

***Analog to Digital Converter - ADC ..... 265***
Features ..... 265
Operation ..... 267
Starting a Conversion ..... 267
Prescaling and Conversion Timing ..... 268
Changing Channel or Reference Selection ..... 271
ADC Noise Canceler ..... 272
ADC Conversion Result ..... 277
ADC Register Description ..... 279

***JTAG Interface and On-chip Debug System ..... 284***
Features ..... 284
Overview ..... 284
Test Access Port – TAP ..... 284
TAP Controller ..... 286
Using the Boundary-scan Chain ..... 287
Using the On-chip Debug System ..... 287
On-chip Debug Specific JTAG Instructions ..... 288
On-chip Debug Related Register in I/O Memory ..... 288
Using the JTAG Programming Capabilities ..... 289
Bibliography ..... 289

***Boundary-scan IEEE 1149.1 (JTAG) ..... 290***
Features..... 290
System Overview..... 290
Data Registers..... 290
Boundary-scan Specific JTAG Instructions ..... 292
Boundary-scan Related Register in I/O Memory ..... 293
Boundary-scan Chain ..... 294
AT90CAN128 Boundary-scan Order .....305
Boundary-scan Description Language Files..... 310

***Boot Loader Support – Read-While-Write Self-Programming..... 311***
Features..... 311
Application and Boot Loader Flash Sections..... 311
Read-While-Write and No Read-While-Write Flash Sections..... 311
Boot Loader Lock Bits..... 314
Entering the Boot Loader Program..... 315
Addressing the Flash During Self-Programming ..... 317
Self-Programming the Flash..... 318

***Memory Programming ..... 325***
Program and Data Memory Lock Bits..... 325
Fuse Bits..... 326
Signature Bytes ..... 328
Calibration Byte ..... 328
Parallel Programming Overview .....329
Parallel Programming ..... 331
SPI Serial Programming Overview .....337
SPI Serial Programming ..... 338
JTAG Programming Overview.....342

***Electrical Characteristics(1)..... 355***
Absolute Maximum Ratings*..... 355
DC Characteristics..... 356
External Clock Drive Characteristics .....358
Two-wire Serial Interface Characteristics .....359
SPI Timing Characteristics .....361
CAN Physical Layer Characteristics..... 362
ADC Characteristics ..... 363
External Data Memory Characteristics .....365
Parallel Programming Characteristics .....371

***AT90CAN128 Typical Characteristics..... 374***
Active Supply Current..... 374
Idle Supply Current..... 377
Power-down Supply Current..... 379
Power-save Supply Current..... 380

Standby Supply Current ..... 381
Pin Pull-up ..... 381
Pin Driver Strength ..... 383
BOD Thresholds and Analog Comparator Offset ..... 385
Internal Oscillator Speed ..... 387
Current Consumption of Peripheral Units ..... 389
Current Consumption in Reset and Reset Pulse Width ..... 392

***Register Summary ..... 394***

***Instruction Set Summary ..... 398***

***Ordering Information ..... 401***

***Packaging Information ..... 401***
TQFP64 ..... 402
QFN64 ..... 403

***Errata ..... 404***
Rev C ..... 404
Rev A & B ..... 404

***Datasheet Change Log for AT90CAN128 ..... 406***
Changes from 4250D-07/04 to 4250E-12/04 ..... 406
Changes from 4250C-03/04 to 4250D-07/04 ..... 406
Changes from 4250B-02/04 to 4250C-03/04 ..... 406
Changes from 4250A-10/03 to 4250B-02/04 ..... 406

***Table of Contents ..... 1***

## Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

## Regional Headquarters

***Europe***
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

***Asia***
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

***Japan***
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

## Atmel Operations

***Memory***
2325 Orchard Parkway
San Jose, CA 95131
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

***Microcontrollers***
2325 Orchard Parkway
San Jose, CA 95131
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

***ASIC/ASSP/Smart Cards***
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

***RF/Automotive***
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

***Biometrics/Imaging/Hi-Rel MPU/***
***High Speed Converters/RF Datacom***
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

***e-mail***
literature@atmel.com

***Web Site***
http://www.atmel.com

**Disclaimer:** Atmel Corporation makes no warranty for the use of its products, other than those expressly contained in the Company's standard warranty which is detailed in Atmel's Terms and Conditions located on the Company's web site. The Company assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. No licenses to patents or other intellectual property of Atmel are granted by the Company in connection with the sale of Atmel products, expressly or by implication. Atmel's products are not authorized for use as critical components in life support devices or systems.

© Atmel Corporation 2004. All rights reserved. Atmel®, logo and combinations thereof are registered trademarks, and Everywhere You Are(SM) are the trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.

Printed on recycled paper.

4250E–CAN–12/04 /xM