PRELIMINARY

Am5$_X$86™ Microprocessor Family

Advanced Micro Devices

DISTINCTIVE CHARACTERISTICS

- **High-Performance Design**
  - — Industry-standard write-back cache support
  - — Frequent instructions execute in one clock
  - — 105.6-million bytes/second burst bus at 33 MHz
  - — Flexible write-through and write-back address control
  - — Advanced 0.35-µ CMOS-process technology
  - — Dynamic bus sizing for 8-, 16-, and 32-bit buses
  - — Supports “soft reset” capability
- **High On-Chip Integration**
  - — 16-Kbyte unified code and data cache
  - — Floating-point unit
  - — Paged, virtual memory management
- **Enhanced System and Power Management**
  - — Stop clock control for reduced power consumption
  - — Industry-standard two-pin System Management Interrupt ($\overline{SMI}$) for power management independent of processor operating mode and operating system
  - — Static design with Auto Halt power-down support
  - — Wide range of chipsets supporting SMM available to allow product differentiation
- **Complete 32-Bit Architecture**
  - — Address and data buses
  - — All registers
  - — 8-, 16-, and 32-bit data types
- **Standard Features**
  - — 3-V core with 5-V tolerant I/O
  - — Available in a 133-MHz version
  - — Binary compatible with all Am486®DX, Am486DX2, and Am486DX4 microprocessors
  - — Wide range of chipsets and support available through the AMD FusionPC℠ Program
- **168-pin PGA package or 208-pin SQFP package**
- **IEEE 1149.1 JTAG Boundary-Scan Compatibility**
- **Supports Environmental Protection Agency's Energy Star program**
  - — 3-V operation reduces power consumption up to 40%
  - — Energy management capability provides excellent base for energy-efficient design
  - — Works with a variety of energy-efficient, power-managed devices

GENERAL DESCRIPTION

The Am5$_X$86™ microprocessor is an addition to the AMD microprocessor product family. The new processor enhances system performance by raising the microprocessor operating frequency to the highest levels allowed by current manufacturing technology, while maintaining complete compatibility with the standard Am486 processor architecture and Microsoft® Windows®. The CPUs incorporate write-back cache, flexible clock control, and enhanced SMM. Table 1 shows available processors in the Am5$_X$86 microprocessor family.

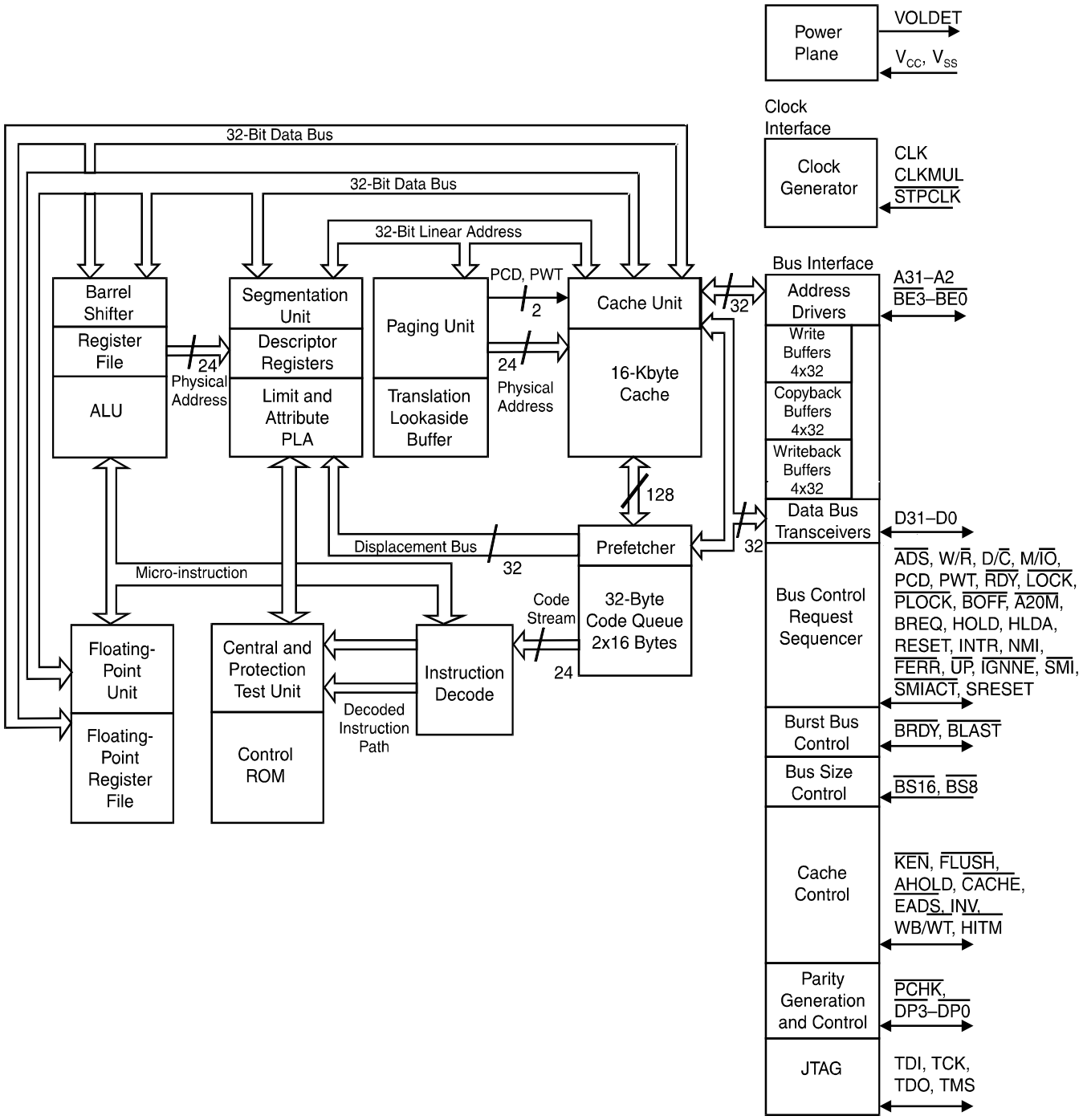
Table 1. Clocking Options

Operating Frequency	Input Clock	Available Package
133 MHz	33 MHz	168-pin PGA
133 MHz	33 MHz	208-pin SQFP

The Am5$_X$86 microprocessor family allows write-back configuration through software and cacheable access control. On-chip cache lines are configurable as either write-through or write-back. The CPU clock control feature permits the CPU clock to be stopped under controlled conditions, allowing reduced power consumption during system inactivity. The SMM function is implemented with an industry standard two-pin interface.

Publication # **19751** Rev**: C** Amendment**/0**
Issue Date: **March 1996**

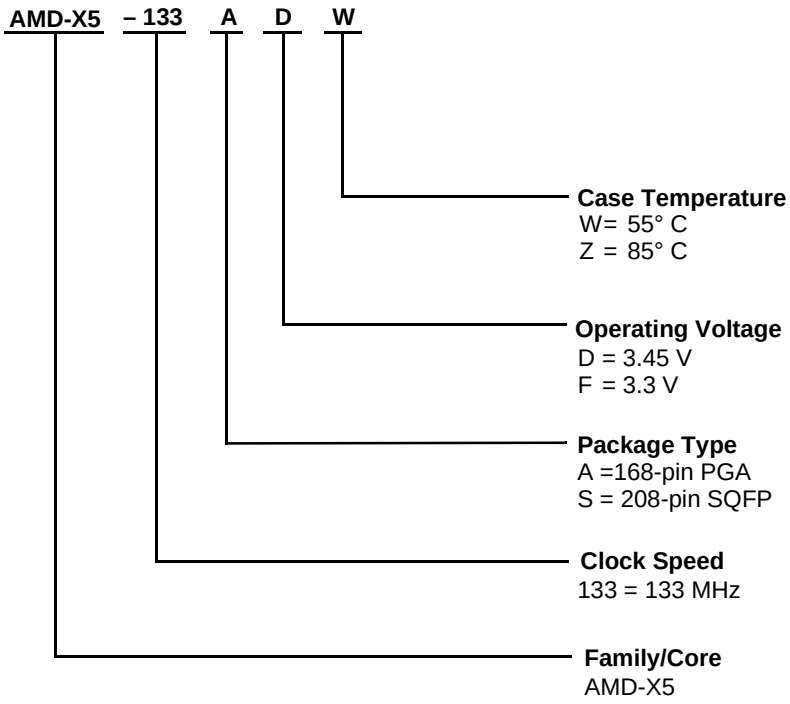
BLOCK DIAGRAM



ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of the elements below.



Valid Combinations

OPN	Package Type	Operating Voltage	Case Temperature (Max)
AMD-X5-133ADW	PGA	3.45 V	55° C
AMD-X5-133ADZ	PGA	3.45 V	85° C
AMD-X5-133SFZ	SQFP	3.3 V	85° C
AMD-X5-133SDZ	SQFP	3.45 V	85° C

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations and to check on newly released combinations.

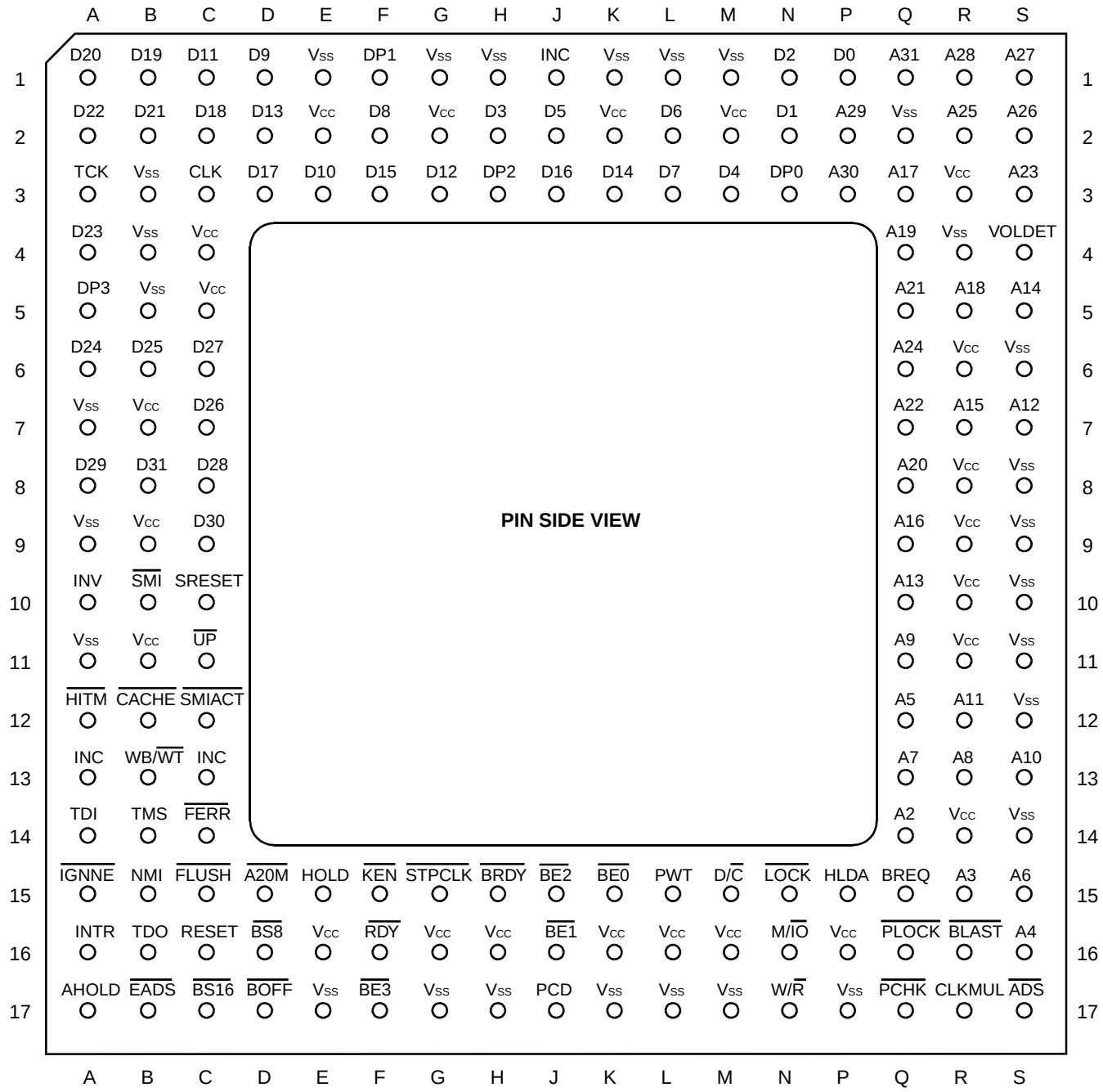
Table of Contents

LIST OF FIGURES

LIST OF TABLES

1 CONNECTION DIAGRAMS AND PIN DESIGNATIONS

1.1 168-pin PGA (Pin Grid Array) Package



1.2 168-pin PGA Designations (Functional Grouping)

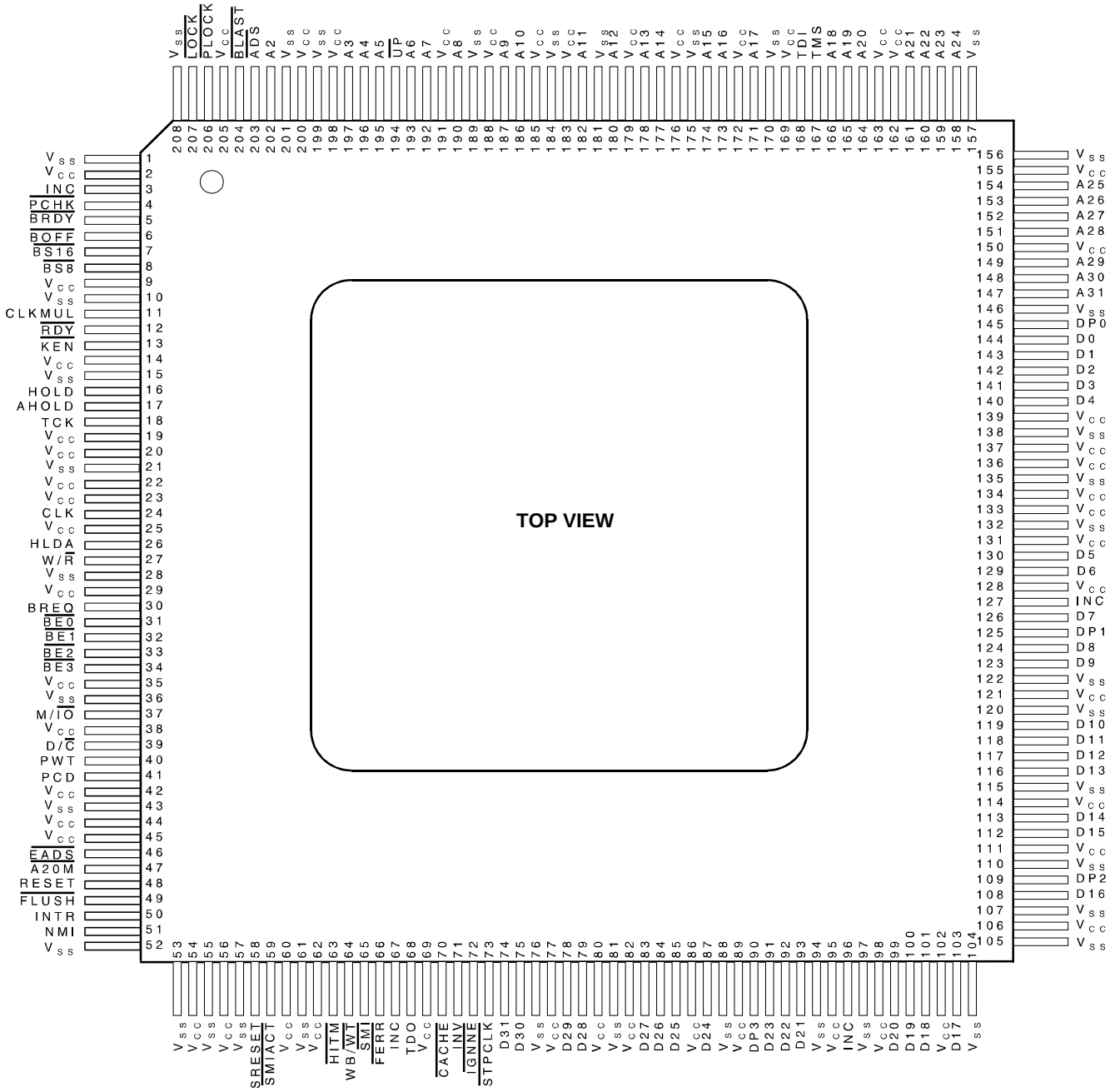
Address		Data		Control		Test		INC	V_{cc}	V_{ss}
Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin No.	Pin No.	Pin No.
A2	Q-14	D0	P-1	$\overline{A20M}$	D-15	TCK	A-3	A-13	B-7	A-7
A3	R-15	D1	N-2	$\overline{ADS}$	S-17	TDI	A-14	C-13	B-9	A-9
A4	S-16	D2	N-1	AHOLD	A-17	TDO	B-16	J-1	B-11	A-11
A5	Q-12	D3	H-2	$\overline{BE0}$	K-15	TMS	B-14		C-4	B-3
A6	S-15	D4	M-3	$\overline{BE1}$	J-16				C-5	B-4
A7	Q-13	D5	J-2	$\overline{BE2}$	J-15				E-2	B-5
A8	R-13	D6	L-2	$\overline{BE3}$	F-17				E-16	E-1
A9	Q-11	D7	L-3	$\overline{BLAST}$	R-16				G-2	E-17
A10	S-13	D8	F-2	$\overline{BOFF}$	D-17				G-16	G-1
A11	R-12	D9	D-1	$\overline{BRDY}$	H-15				H-16	G-17
A12	S-7	D10	E-3	BREQ	Q-15				K-2	H-1
A13	Q-10	D11	C-1	$\overline{BS8}$	D-16				K-16	H-17
A14	S-5	D12	G-3	$\overline{BS16}$	C-17				L-16	K-1
A15	R-7	D13	D-2	$\overline{CACHE}$	B-12				M-2	K-17
A16	Q-9	D14	K-3	CLK	C-3				M-16	L-1
A17	Q-3	D15	F-3	CLKMUL	R-17				P-16	L-17
A18	R-5	D16	J-3	D/$\overline{C}$	M-15				R-3	M-1
A19	Q-4	D17	D-3	DP0	N-3				R-6	M-17
A20	Q-8	D18	C-2	DP1	F-1				R-8	P-17
A21	Q-5	D19	B-1	DP2	H-3				R-9	Q-2
A22	Q-7	D20	A-1	DP3	A-5				R-10	R-4
A23	S-3	D21	B-2	$\overline{EADS}$	B-17				R-11	S-6
A24	Q-6	D22	A-2	$\overline{FERR}$	C-14				R-14	S-8
A25	R-2	D23	A-4	$\overline{FLUSH}$	C-15					S-9
A26	S-2	D24	A-6	$\overline{HITM}$	A-12					S-10
A27	S-1	D25	B-6	HLDA	P-15					S-11
A28	R-1	D26	C-7	HOLD	E-15					S-12
A29	P-2	D27	C-6	$\overline{IGNNE}$	A-15					S-14
A30	P-3	D28	C-8	INTR	A-16					
A31	Q-1	D29	A-8	INV	A-10					
		D30	C-9	$\overline{KEN}$	F-15					
		D31	B-8	$\overline{LOCK}$	N-15					
				M/$\overline{IO}$	N-16					
				NMI	B-15					
				PCD	J-17					
				$\overline{PCHK}$	Q-17					
				$\overline{PLOCK}$	Q-16					
				PWT	L-15					
				$\overline{RDY}$	F-16					
				RESET	C-16					
				$\overline{SMI}$	B-10					
				$\overline{SMIACT}$	C-12					
				SRESET	C-10					
				$\overline{STPCLK}$	G-15					
				$\overline{UP}$	C-11					
				VOLDET	S-4					
				WB/$\overline{WT}$	B-13					
				W/$\overline{R}$	N-17					

Notes:

1. VOLDET is connected internally to V_{SS}.

2. INC = Internal No Connect

1.3 208-pin SQFP (Shrink Quad Flat Pack) Package

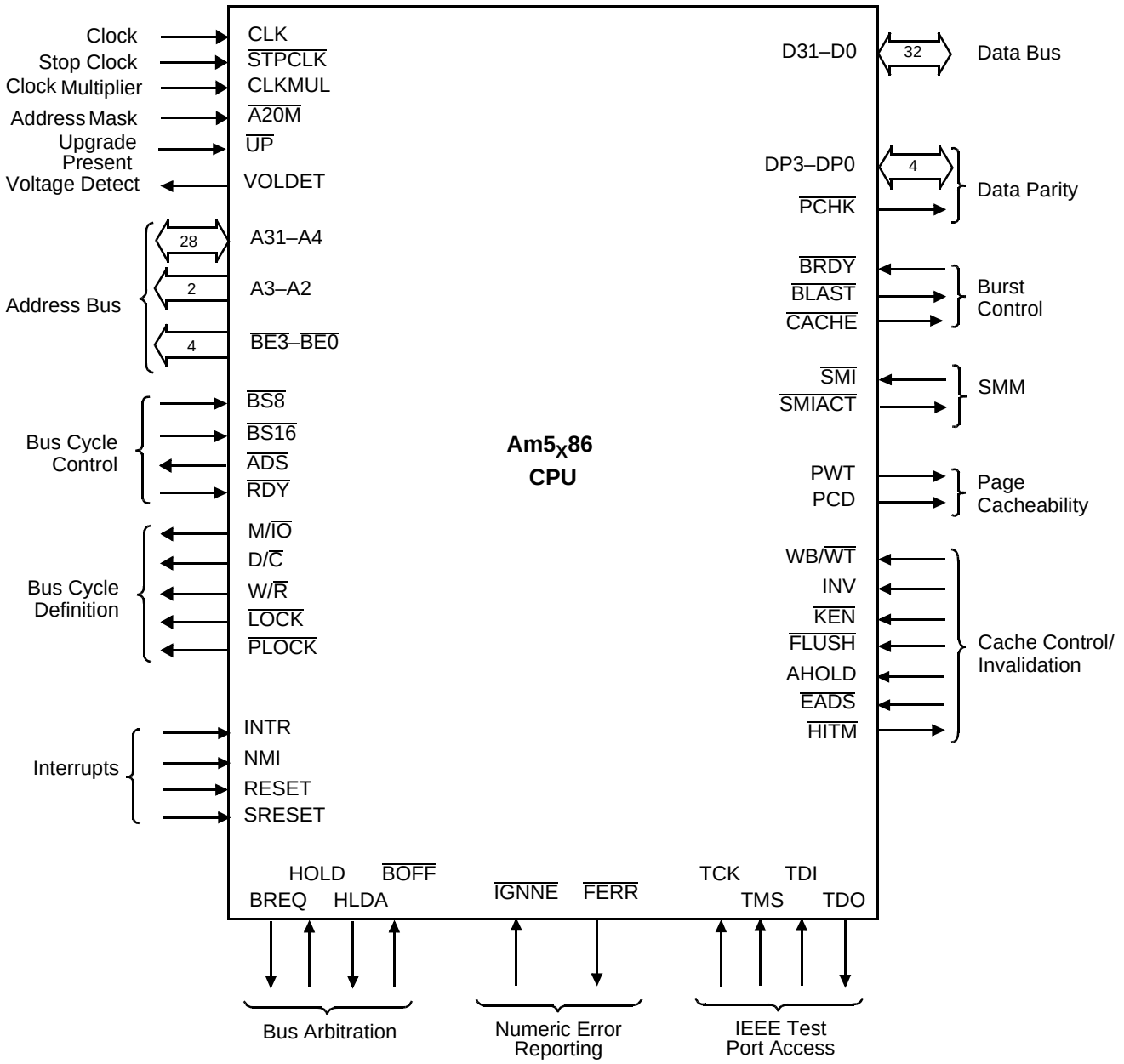


1.4 208-pin SQFP Designations (Functional Grouping)

Address		Data		Control		Test		INC	V_{cc}	V_{ss}
Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin No.	Pin No.	Pin No.
A2	202	D0	144	$\overline{A20M}$	47	TCK	18	3	2	1
A3	197	D1	143	$\overline{ADS}$	203	TDI	168	67	9	10
A4	196	D2	142	AHOLD	17	TDO	68	96	14	15
A5	195	D3	141	$\overline{BE0}$	31	TMS	167	127	19	21
A6	193	D4	140	$\overline{BE1}$	32				20	28
A7	192	D5	130	$\overline{BE2}$	33				22	36
A8	190	D6	129	$\overline{BE3}$	34				23	43
A9	187	D7	126	$\overline{BLAST}$	204				25	52
A10	186	D8	124	$\overline{BOFF}$	6				29	53
A11	182	D9	123	$\overline{BRDY}$	5				35	55
A12	180	D10	119	BREQ	30				38	57
A13	178	D11	118	$\overline{BS8}$	8				42	61
A14	177	D12	117	$\overline{BS16}$	7				44	76
A15	174	D13	116	$\overline{CACHE}$	70				45	81
A16	173	D14	113	CLK	24				54	88
A17	171	D15	112	CLKMUL	11				56	94
A18	166	D16	108	D/$\overline{C}$	39				60	97
A19	165	D17	103	DP0	145				62	104
A20	164	D18	101	DP1	125				69	105
A21	161	D19	100	DP2	109				77	107
A22	160	D20	99	DP3	90				80	110
A23	159	D21	93	$\overline{EADS}$	46				82	115
A24	158	D22	92	$\overline{FERR}$	66				86	120
A25	154	D23	91	$\overline{FLUSH}$	49				89	122
A26	153	D24	87	$\overline{HITM}$	63				95	132
A27	152	D25	85	HLDA	26				98	135
A28	151	D26	84	HOLD	16				102	138
A29	149	D27	83	$\overline{IGNNE}$	72				106	146
A30	148	D28	79	INTR	50				111	156
A31	147	D29	78	INV	71				114	157
		D30	75	$\overline{KEN}$	13				121	170
		D31	74	$\overline{LOCK}$	207				128	175
				M/$\overline{IO}$	37				131	181
				NMI	51				133	184
				PCD	41				134	189
				$\overline{PCHK}$	4				136	199
				$\overline{PLOCK}$	206				137	201
				PWT	40				139	208
				$\overline{RDY}$	12				150	
				RESET	48				155	
				$\overline{SMI}$	65				162	
				SRESET	58				163	
				$\overline{STPCLK}$	73				169	
				$\overline{SMIACT}$	59				172	
				$\overline{UP}$	194				176	
				WB/$\overline{WT}$	64				179	
				W/$\overline{R}$	27				183	
									185	
									188	
									191	
									198	
									200	
									205	

Note:
I*NC = Internal No Connect*

2 LOGIC SYMBOL



3 PIN DESCRIPTIONS

The $Am5_X86$ microprocessor provides the complete interface support offered by the Enhanced Am486 microprocessor family products. The CLKMUL pin settings have changed to accommodate the higher operating speed selection.

$\overline{A20M}$

Address Bit 20 Mask (Active Low; Input)

A Low signal on the $\overline{A20M}$ pin causes the microprocessor to mask address line A20 before performing a lookup to the internal cache, or driving a memory cycle on the bus. Asserting $\overline{A20M}$ causes the processor to wrap the address at 1 Mbyte, emulating Real mode operation. The signal is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition during a specific clock. During normal operation, $\overline{A20M}$ should be sampled High at the falling edge of RESET.

A31–A4/A3–A2

Address Lines (Inputs/Outputs)/(Outputs)

Pins A31–A2 define a physical area in memory or indicate an input/output (I/O) device. Address lines A31–A4 drive addresses into the microprocessor to perform cache line invalidations. Input signals must meet setup and hold times t_{22} and t_{23}. A31–A2 are not driven during bus or address hold.

$\overline{ADS}$

Address Status (Active Low; Output)

A Low output from this pin indicates that a valid bus cycle definition and address are available on the cycle definition lines and address bus. $\overline{ADS}$ is driven active by the same clock as the addresses. $\overline{ADS}$ is active Low and is not driven during bus hold.

AHOLD

Address Hold (Active High; Input)

The external system may assert AHOLD to perform a cache snoop. In response to the assertion of AHOLD, the microprocessor stops driving the address bus A31–A2 in the next clock. The data bus remains active and data can be transferred for previously issued read or write bus cycles during address hold. AHOLD is recognized even during RESET and $\overline{LOCK}$. The earliest that AHOLD can be deasserted is two clock cycles after $\overline{EADS}$ is asserted to start a cache snoop. If $\overline{HITM}$ is activated due to a cache snoop, the microprocessor completes the current bus activity and then asserts $\overline{ADS}$ and drives the address bus while AHOLD is active. This starts the write-back of the modified line that was the target of the snoop.

$\overline{BE3}$–$\overline{BE0}$

Byte Enable (Active Low; Outputs)

The byte enable pins indicate which bytes are enabled and active during read or write cycles. During the first cache fill cycle, however, an external system should ignore these signals and assume that all bytes are active.

- $\overline{BE3}$ for D31–D24
- $\overline{BE2}$ for D23–D16
- $\overline{BE1}$ for D15–D8
- $\overline{BE0}$ for D7–D0

$\overline{BE3}$–$\overline{BE0}$ are active Low and are not driven during bus hold.

$\overline{BLAST}$

Burst Last (Active Low; Output)

Burst Last goes Low to tell the CPU that the next $\overline{BRDY}$ signal completes the burst bus cycle. $\overline{BLAST}$ is active for both burst and non-burst cycles. $\overline{BLAST}$ is active Low and is not driven during a bus hold.

$\overline{BOFF}$

Back Off (Active Low; Input)

This input signal forces the microprocessor to float all pins normally floated during hold, but HLDA is not asserted in response to $\overline{BOFF}$. $\overline{BOFF}$ has higher priority than $\overline{RDY}$ or $\overline{BRDY}$; if both are returned in the same clock, $\overline{BOFF}$ takes effect. The microprocessor remains in bus hold until $\overline{BOFF}$ goes High. If a bus cycle is in progress when BOFF is asserted, the cycle restarts. $\overline{BOFF}$ must meet setup and hold times t_{18} and t_{19} for proper operation. $\overline{BOFF}$ has an internal weak pull-up.

$\overline{BRDY}$

Burst Ready Input (Active Low; Input)

The $\overline{BRDY}$ signal performs the same function during a burst cycle that $\overline{RDY}$ performs during a non-burst cycle. $\overline{BRDY}$ indicates that the external system has presented valid data in response to a read, or that the external system has accepted data in response to a write. $\overline{BRDY}$ is ignored when the bus is idle and at the end of the first clock in a bus cycle. $\overline{BRDY}$ is sampled in the second and subsequent clocks of a burst cycle. The data presented on the data bus is strobed into the microprocessor when $\overline{BRDY}$ is sampled active. If $\overline{RDY}$ is returned simultaneously with $\overline{BRDY}$, $\overline{BRDY}$ is ignored and the cycle is converted to a non-burst cycle. $\overline{BRDY}$ is active Low and has a small pull-up resistor, and must satisfy the setup and hold times t_{16} and t_{17}.

BREQ

Internal Cycle Pending (Active High; Output)

BREQ indicates that the microprocessor has generated a bus request internally, whether or not the microprocessor is driving the bus. BREQ is active High and is floated only during Tri-state Test mode (see $\overline{FLUSH}$).

$\overline{BS8}$/$\overline{BS16}$

Bus Size 8 (Active Low; Input)/
Bus Size 16 (Active Low; Input)

The $\overline{BS8}$ and $\overline{BS16}$ signals allow the processor to operate with 8-bit and 16-bit I/O devices by running multiple bus cycles to respond to data requests: four for 8-bit devices, and two for 16-bit devices. The bus sizing pins are sampled every clock. The microprocessor samples the pins every clock before $\overline{RDY}$ to determine the appropriate bus size for the requesting device. The signals are active Low input with internal pull-up resistors, and must satisfy setup and hold times t_{14} and t_{15} for correct operation. Bus sizing is not permitted during copy-back or write-back operation. $\overline{BS8}$ and $\overline{BS16}$ are ignored during copy-back or write-back cycles.

$\overline{CACHE}$

Internal Cacheability (Active Low; Output)

In Write-through mode, this signal always floats. In Write-back mode for processor-initiated cycles, a Low output on this pin indicates that the current read cycle is cacheable, or that the current cycle is a burst write-back or copy-back cycle. If the $\overline{CACHE}$ signal is driven High during a read, the processor will not cache the data even if the $\overline{KEN}$ pin signal is asserted. If the processor determines that the data is cacheable, $\overline{CACHE}$ goes active when $\overline{ADS}$ is asserted and remains in that state until the next $\overline{RDY}$ or $\overline{BRDY}$ is asserted. $\overline{CACHE}$ floats in response to a $\overline{BOFF}$ or HOLD request.

CLK

Clock (Input)

The CLK input provides the basic microprocessor timing signal. The CLKMUL input selects the multiplier value used to generate the internal operating frequency for the Am5$_X$86 microprocessor family. All external timing parameters are specified with respect to the rising edge of CLK. The clock signal passes through an internal Phase-Lock Loop (PLL).

CLKMUL

Clock Multiplier (Input)

The microprocessor samples the CLKMUL input signal at RESET to determine the design operating frequency. An internal pull-up resistor connects to V_{CC}, which selects Clock-tripled mode if the input is High or left floating. For Clock-quadrupled mode, the input must be pulled Low. For 133-MHz processors, this input must always be connected to V_{SS} to ensure correct operation.

D31–D0

Data Lines (Inputs/Outputs)

Lines D31–D0 define the data bus. The signals must meet setup and hold times t_{22} and t_{23} for proper read operations. These pins are driven during the second and subsequent clocks of write cycles.

D/$\overline{C}$

Data/Control (Output)

This bus cycle definition pin distinguishes memory and I/O data cycles from control cycles. The control cycles are:

- Interrupt Acknowledge
- Halt/Special Cycle
- Code Read (instruction fetching)

DP3–DP0

Data Parity (Inputs/Outputs)

Data parity is generated on all write data cycles with the same timing as the data driven by the microprocessor. Even parity information must be driven back into the microprocessor on the data parity pins with the same timing as read information to ensure that the processor uses the correct parity check. The signals read on these pins do not affect program execution. Input signals must meet setup and hold times t_{22} and t_{23}. DP3–DP0 should be connected to V_{CC} through a pull-up resistor in systems not using parity. DP3–DP0 are active High and are driven during the second and subsequent clocks of write cycles.

$\overline{EADS}$

External Address Strobe (Active Low; Input)

This signal indicates that a valid external address has been driven on the address pins A31–A4 of the microprocessor to be used for a cache snoop. This signal is recognized while the processor is in hold (HLDA is driven active), while forced off the bus with the $\overline{BOFF}$ input, or while AHOLD is asserted. The microprocessor ignores $\overline{EADS}$ at all other times. $\overline{EADS}$ is not recognized if $\overline{HITM}$ is active, nor during the clock after $\overline{ADS}$, nor during the clock after a valid assertion of $\overline{EADS}$. Snoops to the on-chip cache must be completed before another snoop cycle is initiated. Table 2 describes $\overline{EADS}$ when first sampled. $\overline{EADS}$ can be asserted every other clock cycle as long as the hold remains active and $\overline{HITM}$ remains inactive. INV is sampled in the same clock period that $\overline{EADS}$ is asserted. $\overline{EADS}$ has an internal weak pull-up.

Table 2. $\overline{EADS}$ Sample Time

Trigger	$\overline{EADS}$ First Sampled
AHOLD	Second clock after AHOLD asserted
HOLD	First clock after HLDA asserted
$\overline{BOFF}$	Second clock after $\overline{BOFF}$ asserted

Note:
The triggering signal (AHOLD, HOLD, or $\overline{BOFF}$) must remain active for at least 1 clock after $\overline{EADS}$ to ensure proper operation.

$\overline{\text{FERR}}$

Floating-Point Error (Active Low; Output)

Driven active when a floating-point error occurs, $\overline{\text{FERR}}$ is similar to the $\overline{\text{ERROR}}$ pin on a 387 math coprocessor. $\overline{\text{FERR}}$ is included for compatibility with systems using DOS-type floating-point error reporting. $\overline{\text{FERR}}$ is active Low, and is not floated during bus hold, except during Tri-state Test mode (see $\overline{\text{FLUSH}}$).

$\overline{\text{FLUSH}}$

Cache Flush (Active Low; Input)

In Write-back mode, $\overline{\text{FLUSH}}$ forces the microprocessor to write-back all modified cache lines and invalidate its internal cache. The microprocessor generates two flush acknowledge special bus cycles to indicate completion of the write-back and invalidation. In Write-through mode, $\overline{\text{FLUSH}}$ invalidates the cache without issuing a special bus cycle. $\overline{\text{FLUSH}}$ is an active Low input that needs to be asserted only for one clock. $\overline{\text{FLUSH}}$ is asynchronous, but setup and hold times t_{20} and t_{21} must be met for recognition in any specific clock. Sampling $\overline{\text{FLUSH}}$ Low in the clock before the falling edge of RESET causes the microprocessor to enter Tri-state Test mode.

$\overline{\text{HITM}}$

Hit Modified Line (Active Low; Output)

In Write-back mode (WB/$\overline{\text{WT}}$=1 at RESET), $\overline{\text{HITM}}$ indicates that an external snoop cache tag comparison hit a modified line. When a snoop hits a modified line in the internal cache, the microprocessor asserts $\overline{\text{HITM}}$ two clocks after $\overline{\text{EADS}}$ is asserted. The $\overline{\text{HITM}}$ signal stays asserted (Low) until the last $\overline{\text{BRDY}}$ for the corresponding write-back cycle. At all other times, $\overline{\text{HITM}}$ is deasserted (High). During RESET, the $\overline{\text{HITM}}$ signal can be used to detect whether the CPU is operating in Write-back mode. In Write-back mode (WB/$\overline{\text{WT}}$=1 at RESET), $\overline{\text{HITM}}$ is deasserted (driven High) until the first snoop that hits a modified line. In Write-through mode, $\overline{\text{HITM}}$ floats at all times.

HLDA

Hold Acknowledge (Active High; Output)

The HLDA signal is activated in response to a hold request presented on the HOLD pin. HLDA indicates that the microprocessor has given the bus to another local bus master. HLDA is driven active in the same clock in which the microprocessor floats its bus. HLDA is driven inactive when leaving bus hold. HLDA is active High and remains driven during bus hold. HLDA is floated only during Tri-state Test mode (see $\overline{\text{FLUSH}}$).

HOLD

Bus Hold Request (Active High; Input)

HOLD gives control of the microprocessor bus to another bus master. In response to HOLD going active, the microprocessor floats most of its output and input/output pins. HLDA is asserted after completing the current bus cycle, burst cycle, or sequence of locked cycles. The microprocessor remains in this state until HOLD is deasserted. HOLD is active High and does not have an internal pull-down resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper operation.

$\overline{\text{IGNNE}}$

Ignore Numeric Error (Active Low; Input)

When this pin is asserted, the $Am5_X86$ microprocessor will ignore a numeric error and continue executing non-control floating-point instructions. When $\overline{\text{IGNNE}}$ is deasserted, the $Am5_X86$ microprocessor will freeze on a non-control floating-point instruction if a previous floating-point instruction caused an error. $\overline{\text{IGNNE}}$ has no effect when the NE bit in Control Register 0 is set. $\overline{\text{IGNNE}}$ is active Low and is provided with a small internal pullup resistor. $\overline{\text{IGNNE}}$ is asynchronous but must meet setup and hold times t_{20} and t_{21} to ensure recognition in any specific clock.

INTR

Maskable Interrupt (Active High; Input)

When asserted, this signal indicates that an external interrupt has been generated. If the internal interrupt flag is set in EFLAGS, active interrupt processing is initiated. The microprocessor generates two locked interrupt acknowledge bus cycles in response to the INTR pin going active. INTR must remain active until the interrupt acknowledges have been performed to ensure that the interrupt is recognized. INTR is active High and is not provided with an internal pull-down resistor. INTR is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.

INV

Invalidate (Active High; Input)

The external system asserts INV to invalidate the cache-line state when an external bus master proposes a write. It is sampled together with A31–A4 during the clock in which $\overline{\text{EADS}}$ is active. INV has an internal weak pull-up. INV is ignored in Write-through mode.

$\overline{\text{KEN}}$

Cache Enable (Active Low; Input)

$\overline{\text{KEN}}$ determines whether the current cycle is cacheable. When the microprocessor generates a cacheable cycle and $\overline{\text{KEN}}$ is active one clock before $\overline{\text{RDY}}$ or $\overline{\text{BRDY}}$ during the first transfer of the cycle, the cycle becomes a cache line fill cycle. Returning $\overline{\text{KEN}}$ active one clock before $\overline{\text{RDY}}$ during the last read in the cache line fill causes the line to be placed in the on-chip cache. $\overline{\text{KEN}}$ is active Low and is provided with a small internal pull-up resistor. $\overline{\text{KEN}}$ must satisfy setup and hold times t_{14} and t_{15} for proper operation.

$\overline{\text{LOCK}}$

Bus Lock (Active Low; Output)

A Low output on this pin indicates that the current bus cycle is locked. The microprocessor ignores HOLD when $\overline{\text{LOCK}}$ is asserted (although it does acknowledge AHOLD and $\overline{\text{BOFF}}$). $\overline{\text{LOCK}}$ goes active in the first clock of the first locked bus cycle and goes inactive after the last clock of the last locked bus cycle. The last locked cycle ends when $\overline{\text{RDY}}$ is returned. $\overline{\text{LOCK}}$ is active Low and is not driven during bus hold. Locked read cycles are not transformed into cache fill cycles if $\overline{\text{KEN}}$ is active.

M/$\overline{\text{IO}}$

Memory/Input-Output (Active High/Active Low; Output)

A High output indicates a memory cycle. A Low output indicates an I/O cycle.

NMI

Non-Maskable Interrupt (Active High; Input)

A High NMI input signal indicates that an external non-maskable interrupt has occurred. NMI is rising-edge sensitive. NMI must be held Low for at least four CLK periods before this rising edge. The NMI input does not have an internal pull-down resistor. The NMI input is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.

PCD

Page Cache Disable (Active High; Output)

This pin reflects the state of the PCD bit in the page table entry or page directory entry (programmable through the PCD bit in CR3). If paging is disabled, the CPU ignores the PCD bit and drives the PCD output Low. PCD has the same timing as the cycle definition pins (M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$, and W/$\overline{\text{R}}$). PCD is active High and is not driven during bus hold. PCD is masked by the Cache Disable bit (CD) in Control Register 0 (CR0).

$\overline{\text{PCHK}}$

Parity Status (Active Low; Output)

Parity status is driven on the $\overline{\text{PCHK}}$ pin the clock after $\overline{\text{RDY}}$ for read operations. The parity status reflects data sampled at the end of the previous clock. A Low $\overline{\text{PCHK}}$ indicates a parity error. Parity status is checked only for enabled bytes as is indicated by the byte enable and bus size signals. $\overline{\text{PCHK}}$ is valid only in the clock immediately after read data is returned to the microprocessor; at all other times $\overline{\text{PCHK}}$ is inactive High. $\overline{\text{PCHK}}$ is floated only during Tri-state Test mode (see $\overline{\text{FLUSH}}$).

$\overline{\text{PLOCK}}$

Pseudo-Lock (Active Low; Output)

In Write-back mode, the processor forces the output High and the signal is always read as inactive. In Write-through mode, $\overline{\text{PLOCK}}$ operates normally. When asserted, $\overline{\text{PLOCK}}$ indicates that the current bus transaction requires more than one bus cycle. Examples of such operations are segment table descriptor reads (8 bytes) and cache line fills (16 bytes). The microprocessor drives $\overline{\text{PLOCK}}$ active until the addresses for the last bus cycle of the transaction have been driven, whether or not $\overline{\text{RDY}}$ or $\overline{\text{BRDY}}$ is returned. $\overline{\text{PLOCK}}$ is a function of the $\overline{\text{BS8}}$, $\overline{\text{BS16}}$, and $\overline{\text{KEN}}$ inputs. $\overline{\text{PLOCK}}$ should be sampled on the clock when $\overline{\text{RDY}}$ is returned. $\overline{\text{PLOCK}}$ is active Low and is not driven during bus hold.

PWT

Page Write-Through (Active High; Output)

This pin reflects the state of the PWT bit in the page table entry or page directory entry (programmable through the PWT bit in CR3). If paging is disabled, the CPU ignores the PWT bit and drives the PWT output Low. PWT has the same timing as the cycle definition pins (M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$, and W/$\overline{\text{R}}$). PWT is active High and is not driven during bus hold.

RESET

Reset (Active High; Input)

RESET forces the microprocessor to initialize. The microprocessor cannot begin execution of instructions until at least 1 ms after V_{CC} and CLK have reached their proper DC and AC specifications. To ensure proper microprocessor operation, the RESET pin should remain active during this time. RESET is active High. RESET is asynchronous but must meet setup and hold times t_{20} and t_{21} to ensure recognition on any specific clock.

$\overline{\text{RDY}}$

Non-Burst Ready (Active Low; Input)

A Low input on this pin indicates that the current bus cycle is complete, that is, either the external system has presented valid data on the data pins in response to a read, or the external system has accepted data from the microprocessor in response to a write. $\overline{\text{RDY}}$ is ignored when the bus is idle and at the end of the bus cycle's first clock. $\overline{\text{RDY}}$ is active during address hold. Data can be returned to the processor while AHOLD is active. $\overline{\text{RDY}}$ is active Low and does not have an internal pull-up resistor. $\overline{\text{RDY}}$ must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.

$\overline{\text{SMI}}$

SMM Interrupt (Active Low; Input)

A Low signal on the $\overline{\text{SMI}}$ pin signals the processor to enter System Management mode (SMM). $\overline{\text{SMI}}$ is the highest level processor interrupt. The $\overline{\text{SMI}}$ signal is recognized on an instruction boundary, similar to the NMI and INTR signals. $\overline{\text{SMI}}$ is sampled on every rising clock edge. $\overline{\text{SMI}}$ is a falling-edge sensitive input. The $\overline{\text{SMI}}$ input has an internal pull-up resister. Recognition of $\overline{\text{SMI}}$ is guaranteed in a specific clock if it is asserted synchronously and meets the setup and hold times. If $\overline{\text{SMI}}$ is asserted asynchronously, it must go High for a minimum of two clocks before going Low, and it must remain Low

for at least two clocks to guarantee recognition. When the CPU recognizes $\overline{SMI}$, it enters SMM before executing the next instruction and saves internal registers in SMM space.

$\overline{SMIACT}$

SMM Interrupt Active (Active Low; Output)

$\overline{SMIACT}$ goes Low in response to $\overline{SMI}$. It indicates that the processor is operating under SMM control. $\overline{SMIACT}$ remains Low until the processor receives a RESET signal or executes the Resume Instruction (RSM) to leave SMM. This signal is always driven. It does not float during bus HOLD or $\overline{BOFF}$.

Note: *Do not use SRESET to exit from SMM. The system should block SRESET during SMM.*

SRESET

Soft Reset (Active High; Input)

The CPU samples SRESET on every rising clock edge. If SRESET is sampled active, the SRESET sequence begins on the next instruction boundary. SRESET resets the processor, but, unlike RESET, does not cause it to sample $\overline{UP}$ or WB/$\overline{WT}$, or affect the FPU, cache, CD and NW bits in CR0, and SMBASE. SRESET is asynchronous and must meet the same timing as RESET. The SRESET input has an internal pull-down resistor.

$\overline{STPCLK}$

Stop Clock (Active Low; Input)

A Low input signal indicates a request has been made to turn off the CLK input. When the CPU recognizes a $\overline{STPCLK}$, the processor:

- Stops execution on the next instruction boundary (unless superseded by a higher priority interrupt)
- Empties all internal pipelines and write buffers
- Generates a Stop Grant acknowledge bus cycle

$\overline{STPCLK}$ is active Low and has an internal pull-up resistor. $\overline{STPCLK}$ is asynchronous, but it must meet setup and hold times t_{20} and t_{21} to ensure recognition in any specific clock. $\overline{STPCLK}$ must remain active until the Stop Clock special bus cycle is issued and the system returns either $\overline{RDY}$ or $\overline{BRDY}$.

TCK

Test Clock (Input)

Test Clock provides the clocking function for the JTAG boundary scan feature. TCK clocks state information and data into the component on the rising edge of TCK on TMS and TDI, respectively. Data is clocked out of the component on the falling edge of TCK on TDO.

TDI

Test Data Input (Input)

TDI is the serial input that shifts JTAG instructions and data into the tested component. TDI is sampled on the rising edge of TCK during the SHIFT-IR and the SHIFT-DR TAP (Test Access Port) controller states. During all other TAP controller states, TDI is ignored. TDI uses an internal weak pull-up.

TDO

Test Data Output (Active High; Output)

TDO is the serial output that shifts JTAG instructions and data out of the component. TDO is driven on the falling edge of TCK during the SHIFT-IR and SHIFT-DR TAP controller states. Otherwise, TDO is tri-stated.

TMS

Test Mode Select (Active High; Input)

TMS is decoded by the JTAG TAP to select the operation of the test logic. TMS is sampled on the rising edge of TCK. To guarantee deterministic behavior of the TAP controller, the TMS pin has an internal pull-up resistor.

$\overline{UP}$

Write/Read (Input)

The processor samples the Upgrade Present ($\overline{UP}$) pin in the clock before the falling edge of RESET. If it is Low, the processor tri-states its outputs immediately. $\overline{UP}$ must remain asserted to keep the processor inactive. The pin uses an internal pull-up resistor.

VOLDET—(168-pin PGA package only)

Voltage Detect (Output)

VOLDET provides an external signal to allow the system to determine the CPU input power level (3 V or 5 V). For $Am5_X86$ processors, the pin ties internally to V_{SS}.

WB/$\overline{WT}$

Write-Back/Write-Through (Input)

If the processor samples WB/$\overline{WT}$ High at RESET, the processor is configured in Write-back mode and all subsequent cache line fills sample WB/$\overline{WT}$ on the same clock edge in which it finds either $\overline{RDY}$ or the first $\overline{BRDY}$ of a burst transfer to determine if the cache line is designated as Write-back mode or Write-through. If the signal is Low on the first $\overline{BRDY}$ or $\overline{RDY}$, the cache line is write-through. If the signal is High, the cache line is write-back. If WB/$\overline{WT}$ is sampled Low at RESET, all cache line fills are write-through. WB/$\overline{WT}$ has an internal weak pull-down.

W/$\overline{R}$

Write/Read (Output)

A High output indicates a write cycle. A Low output indicates a read cycle.

Note: *The $Am5_X86$ microprocessor does not use the V_{CC5} pin used by some 3-V, clock-tripled, 486-based processors. The corresponding pin on the $Am5_X86$ microprocessor is an Internal No Connect (INC).*

4 FUNCTIONAL DESCRIPTION

4.1 Overview

$Am5_X86$ microprocessors use a 32-bit architecture with on-chip memory management and cache memory units. The instruction set includes the complete 486 microprocessor instruction set along with extensions to serve the new extended applications. All software written for the 486 microprocessor and previous members of the x86 architectural family can run on the $Am5_X86$ microprocessor without modification.

The on-chip Memory Management Unit (MMU) is completely compatible with the 486 MMU. The MMU includes a segmentation unit and a paging unit. Segmentation allows management of the logical address space by providing easy data and code relocatibility and efficient sharing of global resources. The paging mechanism operates beneath segmentation and is transparent to the segmentation process. Paging is optional and can be disabled by system software. Each segment can be divided into one or more 4-Kbyte segments. To implement a virtual memory system, the $Am5_X86$ microprocessor supports full restartability for all page and segment faults.

4.2 Memory

Memory is organized into one or more variable length segments, each up to 4 Gbytes (2^{32} bytes). A segment can have attributes associated with it, including its location, size, type (i.e., stack, code, or data), and protection characteristics. Each task on a microprocessor can have a maximum of 16,381 segments, each up to 4 Gbytes. Thus, each task has a maximum of 64 Tbytes of virtual memory.

The segmentation unit provides four levels of protection for isolating and protecting applications and the operating system from each other. The hardware-enforced protection allows high-integrity system designs.

4.3 Modes of Operation

The $Am5_X86$ microprocessor has four modes of operation: Real Address mode (Real mode), Virtual 8086 Address mode (Virtual mode), Protected Address mode (Protected mode), and System Management mode (SMM).

4.3.1 Real Mode

In Real mode, the $Am5_X86$ microprocessor operates as a fast 8086. Real mode is required primarily to set up the processor for Protected mode operation.

4.3.2 Virtual Mode

In Virtual mode, the processor appears to be in Real mode, but can use the extended memory accessing of Protected mode.

4.3.3 Protected Mode

Protected mode provides access to the sophisticated memory management paging and privilege capabilities of the processor.

4.3.4 System Management Mode

SMM is a special operating mode described in detail in Section 7.

4.4 Cache Architecture

The $Am5_X86$ microprocessor family supports a superset architecture of the standard 486 cache implementation. This architectural enhancement improves not only CPU performance, but total system performance.

4.4.1 Write-Through Cache

The standard 486DX-type write-through cache architecture is characterized by the following:

- External read accesses are placed in the cache if they meet proper caching requirements.
- Subsequent reads to the data in the cache are made if the address is stored in the cache tag array.
- Write operations to a valid address in the cache are updated in the cache *and* to external memory. This data writing technique is called *write-through*.

The write-through cache implementation forces all writes to flow through to the external bus and back to main memory. Consequently, the write-through cache generates a large amount of bus traffic on the external data bus.

4.4.2 Write-Back Cache

The microprocessor write-back cache architecture is characterized by the following:

- External read accesses are placed in the cache if they meet proper caching requirements.
- Subsequent reads to the data in the cache are made if the address is stored in the cache tag array.
- Write operations to a valid address in the cache that is in the write-through (shared) state is updated in the cache and to external memory.
- Write operations to a valid address in the cache that is in the write-back (exclusive or modified) state is updated *only* in the cache. External memory is *not* updated at the time of the cache update.
- Modified data is written back to external memory when the modified cache line is being replaced with a new cache line (copy-back operation) or an external bus master has snooped a modified cache line (write-back).

The write-back cache feature significantly reduces the amount of bus traffic on the external bus; however, it also adds complexity to the system design to maintain memory coherency. The write-back cache requires en-

hanced system support because the cache may contain data that is not identical to data in main memory at the same address location.

4.5 Write-Back Cache Protocol

The Am5$_X$86 microprocessor family write-back cache coherency protocol reduces bus activity while maintaining data coherency in a multimaster environment. The cache coherency protocol offers the following advantages:

- No unnecessary bus traffic. The protocol dynamically identifies shared data to the granularity of a cache line. This dynamic identification ensures that the traffic on the external bus is the minimum necessary to ensure coherency.
- Software-transparent. Because the protocol gives the appearance of a single, unified memory, software does not have to maintain coherency or identify shared data. Application software developed for a system without a cache can run without modification. Software support is required only in the operating system to identify non-cacheable data regions.

The Am5$_X$86 microprocessor family implements a modified MESI protocol on systems with write-back cache support. MESI allows a cache line to exist in four states: modified, exclusive, shared, and invalid. The Am5$_X$86 microprocessor family allocates memory in the cache due to a read miss. Write allocation is not implemented. To maintain coherency between cache and main memory, the MESI protocol has the following characteristics:

- The system memory is always updated during a snoop when a modified line is hit.
- If a modified line is hit by another master during snooping, the master is forced off the bus and the snooped cache writes back the modified line to the system memory. After the snooped cache completes the write, the forced-off bus master restarts the access and reads the modified data from memory.

4.5.1 Cache Line Overview

To implement the Am5$_X$86 microprocessor cache coherency protocol, each tag entry is expanded to 2 bits: S1 and S0. Each tag entry is associated with a cache line. Table 3 shows the cache line organization.

Table 3. Cache Line Organization

Data Words (32 Bits)	Address Tag and Status
D0	Address Tag, S1, S0
D1	
D2	
D3	

4.5.2 Line Status and Line State

A cache line can occupy one of four legal states as indicated by bits S0 and S1. The line states are shown in Table 4. Each line in the cache is in one of these states. The state transition is induced either by the processor or during snooping from an external bus master.

Table 4. Legal Cache Line States

S1	S0	Line State
0	0	Invalid
0	1	Exclusive
1	0	Modified
1	1	Shared

4.5.2.1 Invalid

An invalid cache line does not contain valid data for any external memory location. An invalid line does not participate in the cache coherency protocol.

4.5.2.2 Exclusive

An exclusive line contains valid data for some external memory location. The data exactly matches the data in the external memory location.

4.5.2.3 Shared

A shared line contains valid data for an external memory location, the data is shared by another cache, and the shared data matches the data in the external memory exactly; or the cache line is in Write-through mode.

4.5.2.4 Modified

A modified line contains valid data for an external memory location. However, the data does not match the data in the external location because the processor has modified the data since it was loaded from the external memory. A cache that contains a modified line is responsible for ensuring that the data is properly maintained. This means that in the case of an external access to that line from another external bus master, the modified line is first written back to the external memory before the other external bus master can complete its access. Table 5 shows the MESI cache line states and the corresponding availability of data.

Table 5. MESI Cache Line Status

Situation	Modified	Exclusive	Shared	Invalid
Line valid?	Yes	Yes	Yes	No
External memory is...	out-of-date	valid	valid	status unknown
A write to this cache line...	does not go to the bus	does not go to the bus	goes to the bus and updates	goes directly to the bus

4.6 Cache Replacement Description

The cache line replacement algorithm uses the standard Am486 CPU pseudo LRU (Least-Recently Used) strategy. When a line must be placed in the internal cache, the microprocessor first checks to see if there is an invalid line available in the set. If no invalid line is available, the LRU algorithm replaces the least-recently used cache line in the four-way set with the new cache line. If the cache line for replacement is modified, the modified cache line is placed into the copy-back buffer for copying back to external memory, and the new cache line is placed into the cache. This copy-back ensures that the external memory is updated with the modified data upon replacement.

4.7 Memory Configuration

In computer systems, memory regions require specific caching and memory write methods. For example, some memory regions are non-cacheable while others are cacheable but are write-through. To allow maximum memory configuration, the microprocessor supports specific memory region requirements. All bus masters, such as DMA controllers, must reflect all data transfers on the microprocessor local bus so that the microprocessor can respond appropriately.

4.7.1 Cacheability

The $Am5_X86$ CPU caches data based on the state of the CD and NW bits in CR0, in conjunction with the $\overline{KEN}$ signal, at the time of a burst read access from memory. If the WB/$\overline{WT}$ signal is Low during the first $\overline{BRDY}$, $\overline{KEN}$ meets the standard setup and hold requirements and the four 32-bit doublewords are still placed in the cache. However, all cacheable accesses in this mode are considered write-through. When the WB/$\overline{WT}$ is High during the first $\overline{BRDY}$, the entire four 32-bit doubleword transfer is considered write-back.

Note: *The CD bit in CR0 enables (0) or disables (1) the internal cache. The NW bit in CR0 enables (0) or disables (1) write-through and snooping cycles. RESET sets CD and NW to 1. Unlike RESET, however, SRESET does not invalidate the cache nor does it modify the values of CD and NW in CR0.*

4.7.2 Write-Through/Write-Back

If the CPU is operating in Write-back mode (i.e., the WB/$\overline{WT}$ pin was sampled High at RESET), the WB/$\overline{WT}$ pin indicates whether an individual write access is executed as write-through or write-back. The $Am5_X86$ microprocessor does this on an access-by-access basis. Once the cache line is in the cache, the STATUS bit is tested each time the processor writes to the cache line or a tag compare results in a hit during Bus-watching mode. If the WB/$\overline{WT}$ signal is Low during the first $\overline{BRDY}$ of the cache line read access, the cache line is considered a write-through access. Therefore, all writes to this location in the cache are reflected on the external bus, even if the cache line is write protected.

4.8 Cache Functionality in Write-Back Mode

The description of cache functionality in Write-back mode is divided into two sections: processor-initiated cache functions and snooping actions.

4.8.1 Processor-Initiated Cache Functions and State Transitions

The microprocessor contains two new buffers for use with the MESI protocol support: the copy-back buffer and the write-back buffer. The processor uses the copy-back buffer for cache line replacement of modified lines. The write-back buffer is used when an external bus master hits a modified line in the cache during a snoop operation and the cache line is designated for write-back to main memory. Each buffer is four doublewords in size. Figure 1 shows a diagram of the state transitions induced by the local processor. When a read miss occurs, the line selected for replacement remains in the modified state until overwritten. A copy of the modified line is sent to the copy-back buffer to be written back after replacement. When reload has successfully completed, the line is set either to the exclusive or the shared state, depending on the state of PWT and WB/$\overline{WT}$ signals.

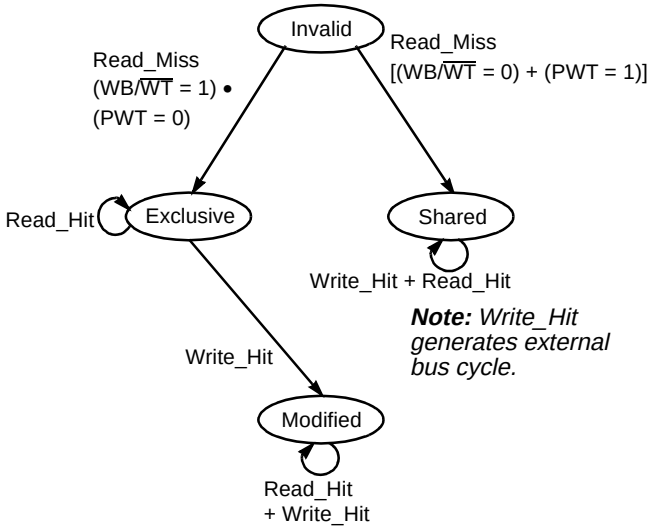


Figure 1. Processor-Induced Line Transitions in Write-Back Mode

If the PWT signal is 0, the external WB/$\overline{WT}$ signal determines the new state of the line. If the WB/$\overline{WT}$ signal was asserted to 1 during reload, the line transits to the exclusive state. If the WB/$\overline{WT}$ signal was 0, the line transits to the shared state. If the PWT signal is 1, it overrides the WB/$\overline{WT}$ signal, forcing the line into the shared state. Therefore, if paging is enabled, the software programmed PWT bit can override the hardware signal WB/$\overline{WT}$.

Until the line is reallocated, a write is the only processor action that can change the state of the line. If the write occurs to a line in the exclusive state, the data is simply written into the cache and the line state is changed to modified. The modified state indicates that the contents of the line require copy-back to the main memory before the line is reallocated.

If the write occurs to a line in the shared state, the cache performs a write of the data on the external bus to update the external memory. The line remains in the shared state until it is replaced with a new cache line or until it is flushed. In the modified state, the processor continues to write the line without any further external actions or state transitions.

If the PWT or PCD bits are changed for a specified memory location, the tag bits in the cache are assumed to be correct. To avoid memory inconsistencies with respect to cacheability and write status, a cache copy-back and invalidation should be invoked either by using the WBINVD instruction or asserting the $\overline{FLUSH}$ signal.

4.8.2 Snooping Actions and State Transitions

To maintain cache coherency, the CPU must allow snooping by the current bus master. The bus master initiates a snoop cycle to check whether an address is cached in the internal cache of the microprocessor. A snoop cycle differs from any other cycle in that it is initiated externally to the microprocessor, and the signal for beginning the cycle is $\overline{EADS}$ instead of $\overline{ADS}$. The address bus of the microprocessor is bidirectional to allow the address of the snoop to be driven by the system. A snoop access can begin during any hold state:

- While HOLD and HLDA are asserted
- While $\overline{BOFF}$ is asserted
- While AHOLD is asserted

In the clock in which $\overline{EADS}$ is asserted, the microprocessor samples the INV input to qualify the type of inquiry. INV specifies whether the line (if found) must be invalidated (i.e., the MESI status changes to Invalid or I). A line is invalidated if the snoop access was generated due to a write of another bus master. This is indicated by INV set to 1. In the case of a read, the line does not have to be invalidated, which is indicated by INV set to 0.

The core system logic can generate $\overline{EADS}$ by watching the $\overline{ADS}$ from the current bus master, and INV by watching the W/$\overline{R}$ signal. The microprocessor compares the address of the snoop request with addresses of lines in the cache and of any line in the copy-back buffer waiting to be transferred on the bus. It does not, however, compare with the address of write-miss data in the write buffers. Two clock cycles after sampling $\overline{EADS}$, the microprocessor drives the results of the snoop on the $\overline{HITM}$ pin. If $\overline{HITM}$ is active, the line was found in the modified state; if inactive, the line was in the exclusive or shared state, or was not found.

Figure 2 shows a diagram of the state transitions induced by snooping accesses.

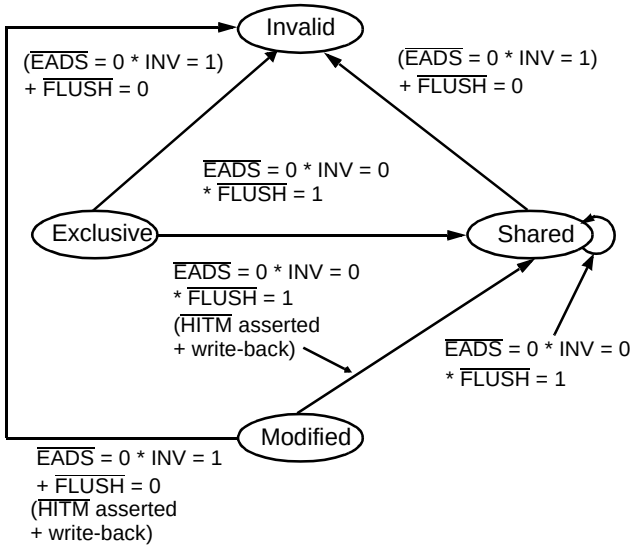


Figure 2. Snooping State Transitions

4.8.2.1 Difference between Snooping Access Cases

Snooping accesses are external accesses to the microprocessor. As described earlier, the snooping logic has a set of signals independent from the processor-related signals. Those signals are:

- $\overline{EADS}$
- INV
- $\overline{HITM}$

In addition to these signals, the address bus is required as an input. This is achieved by setting AHOLD, HOLD, or $\overline{BOFF}$ active.

Snooping can occur in parallel with a processor-initiated access that has already been started. The two accesses depend on each other only when a modified line is written back. In this case, the snoop requires the use of the cycle control signals and the data bus. The following sections describe the scenarios for the HOLD, AHOLD, and $\overline{BOFF}$ implementations.

4.8.2.2 HOLD Bus Arbitration Implementation

The HOLD/HLDA bus arbitration scheme is used primarily in systems where all memory transfers are seen by the microprocessor. The HOLD/HLDA bus arbitration scheme permits simple write-back cache design while maintaining a relatively high performing system. Figure 3 shows a typical system block diagram for HOLD/HLDA bus arbitration.

Note: *To maintain proper system timing, the HOLD signal must remain active for one clock cycle after $\overline{HITM}$ transitions active. Deassertion of HOLD in the same clock cycle as $\overline{HITM}$ assertion may lead to unpredictable processor behavior.*

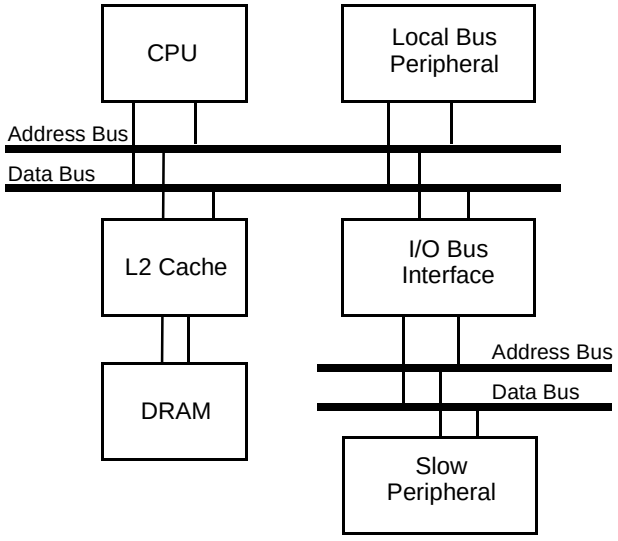


Figure 3. Typical System Block Diagram for HOLD/HLDA Bus Arbitration

4.8.2.2.1 Processor-Induced Bus Cycles

In the following scenarios, read accesses are assumed to be cache line fills. The cases also assume that the core system logic does not return $\overline{BRDY}$ or $\overline{RDY}$ until $\overline{HITM}$ is sampled. The addition of wait states follows the standard 486 bus protocol. For demonstration purposes, only the zero wait state approach is shown. Table 6 explains the key to switching waveforms.

Table 6. Key to Switching Waveforms

Waveform	Inputs	Outputs
	Must be steady	Will be steady
	May change from H to L	Will change from H to L
	May change from L to H	Will change from L to H
	Don't care; any change permitted	Changing; state unknown
	Does not apply	Center line is High-impedance "Off" state

4.8.2.2.2 External Read

***Scenario*:** The data resides in external memory (see Figure 4).

Step 1 The processor starts the external read access by asserting $\overline{ADS}$ = 0 and W/$\overline{R}$ = 0.

Step 2 WB/$\overline{WT}$ is sampled in the same cycle as $\overline{BRDY}$. If WB/$\overline{WT}$ = 1, the data resides in a write-back cacheable memory location.

Step 3 The processor completes its burst read and asserts $\overline{BLAST}$.

4.8.2.2.3 External Write

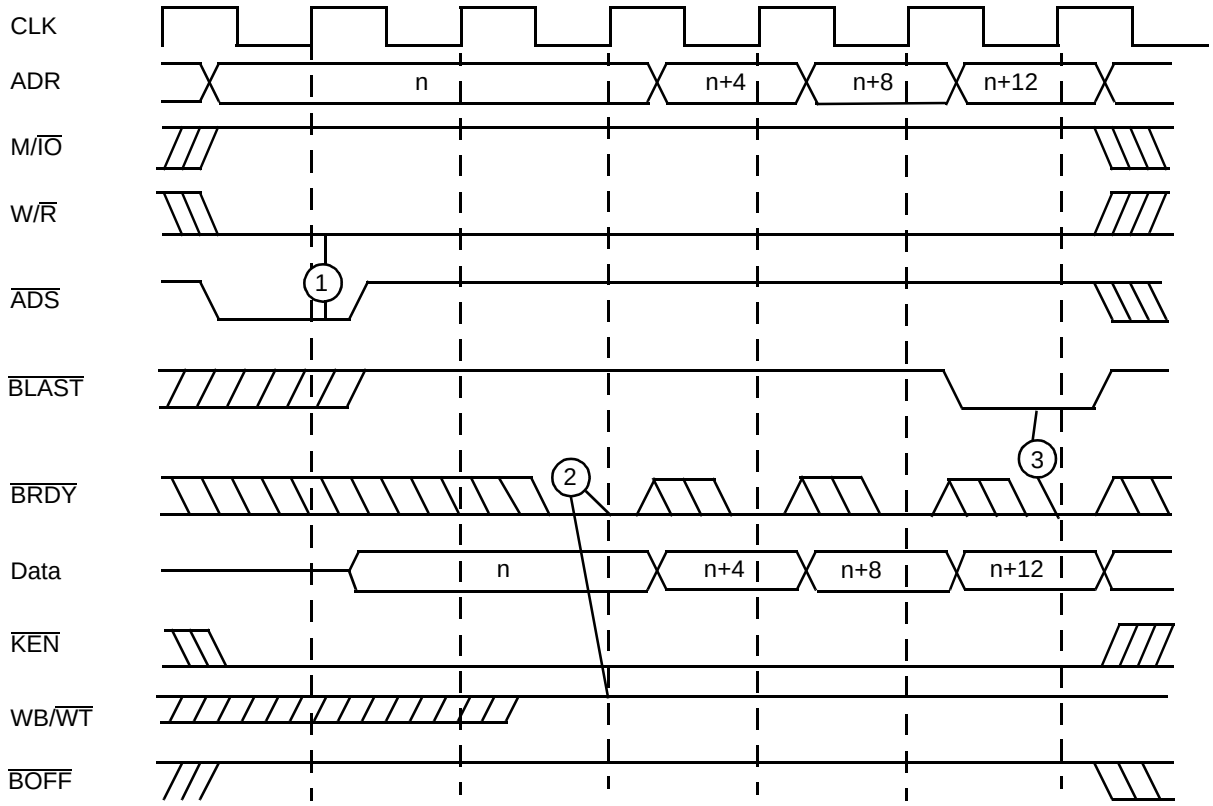
Scenario: The data is written to the external memory (see Figure 5).

Step 1 The processor starts the external write access by asserting $\overline{ADS}$ = 0 and W/$\overline{R}$ = 1.

Step 2 The processor completes its write to the core system logic.

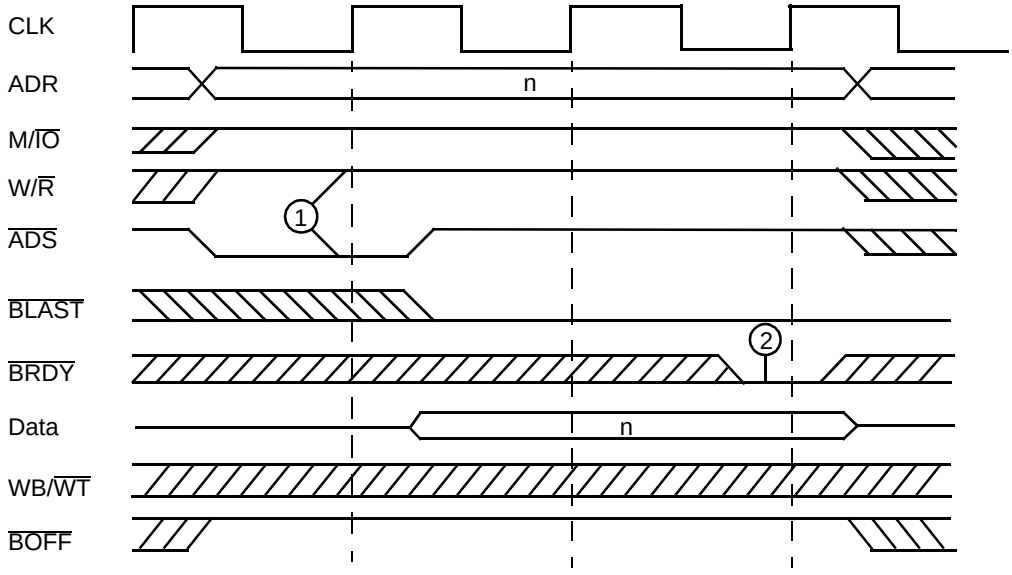
4.8.2.2.4 HOLD/HLDA External Access TIming

In systems with two or more bus masters, each bus master is equipped with individual HOLD and HLDA control signals. These signals are then centralized to the core system logic that controls individual bus masters, depending on bus request signals and the $\overline{HITM}$ signal.



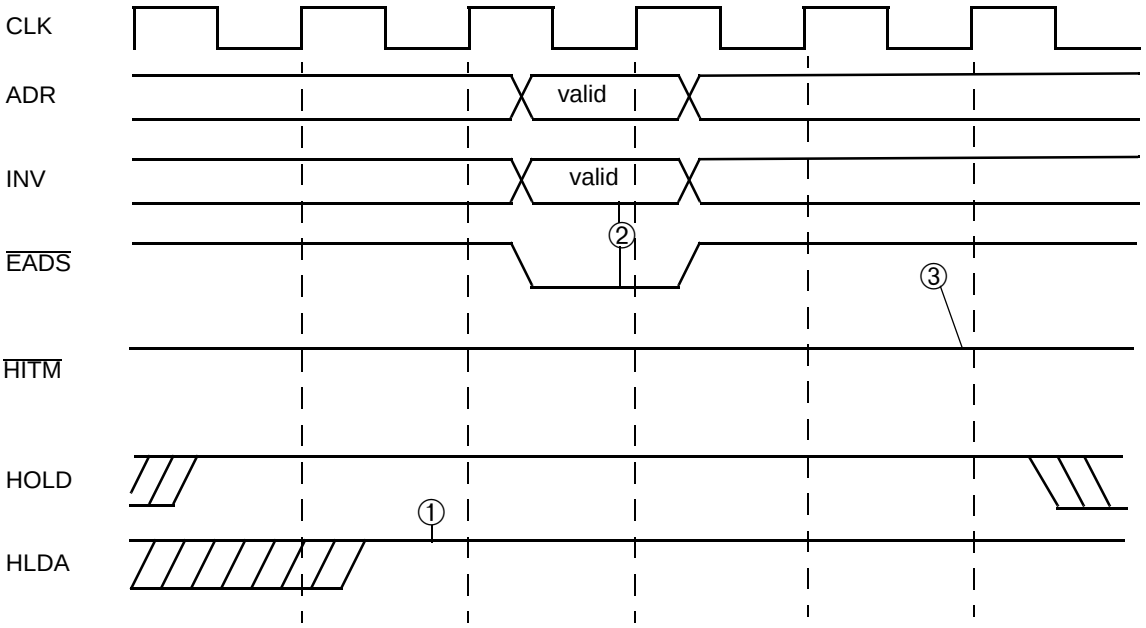
Note:
The circled numbers in this figure represent the steps in section 4.8.2.2.2.

Figure 4. External Read



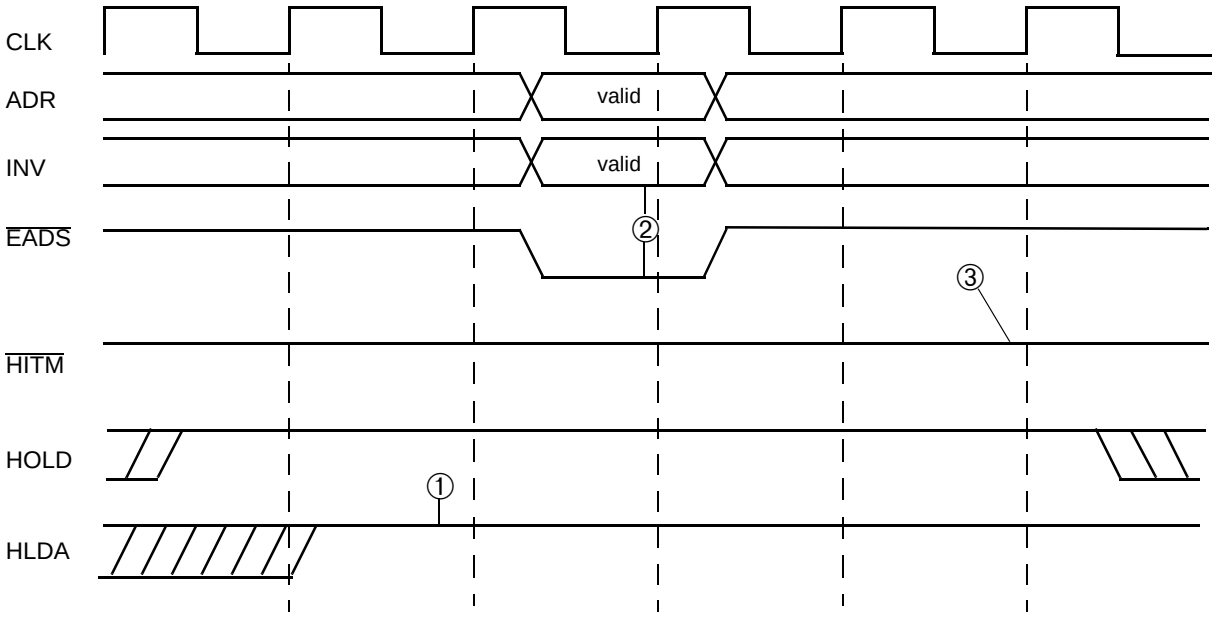
Note:
The circled numbers in this figure represent the steps in section 4.8.2.2.3.

Figure 5. External Write



Note:
The circled numbers in this figure represent the steps in section 4.8.3.1.

Figure 6. Snoop of On-Chip Cache That Does Not Hit a Line



Note:
The circled numbers in this figure represent the steps in section 4.8.3.2.

Figure 7. Snoop of On-Chip Cache That Hits a Non-modified Line

4.8.3 External Bus Master Snooping Actions

The following scenarios describe the snooping actions of an external bus master.

4.8.3.1 Snoop Miss

Scenario: A snoop of the on-chip cache does not hit a line, as shown in Figure 6.

Step 1 The microprocessor is placed in Snooping mode with HOLD. HLDA must be High for a minimum of one clock cycle before $\overline{EADS}$ assertion. In the fastest case, this means that HOLD was asserted one clock cycle before the HLDA response.

Step 2 $\overline{EADS}$ and INV are applied to the microprocessor. If INV is 0, a read access caused the snooping cycle. If INV is 1, a write access caused the snooping cycle.

Step 3 Two clock cycles after $\overline{EADS}$ is asserted, $\overline{HITM}$ becomes valid. Because the addressed line is not in the snooping cache, $\overline{HITM}$ is 1.

4.8.3.2 Snoop Hit to a Non-Modified Line

Scenario: The snoop of the on-chip cache hits a line, and the line is not modified (see Figure 7).

Step 1 The microprocessor is placed in Snooping mode with HOLD. HLDA must be High for a minimum of one clock cycle before $\overline{EADS}$ assertion. In the fastest case, this means that HOLD was asserted one clock cycle before the HLDA response.

Step 2 $\overline{EADS}$ and INV are applied to the microprocessor. If INV is 0, a read access caused the snooping cycle. If INV is 1, a write access caused the snooping cycle.

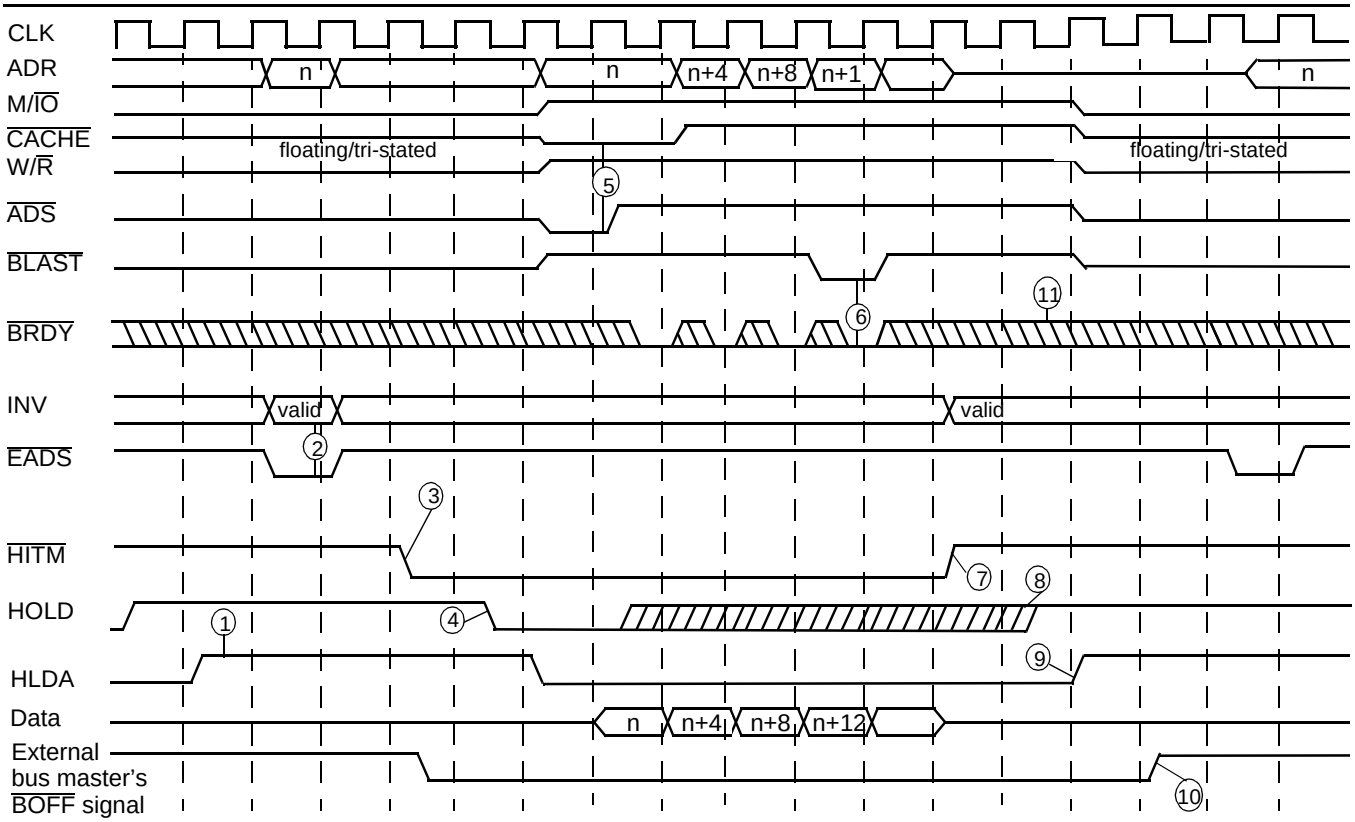
Step 3 Two clock cycles after $\overline{EADS}$ is asserted, $\overline{HITM}$ becomes valid. In this case, $\overline{HITM}$ is 1.

4.8.4 Write-Back Case

Scenario: Write-back accesses are always burst writes with a length of four 32-bit words. For burst writes, the burst always starts with the microprocessor line offset at 0. HOLD must be deasserted before the write-back can be performed (see Figure 8).

Step 1 HOLD places the microprocessor in Snooping mode. HLDA must be High for a minimum of one clock cycle before $\overline{EADS}$ assertion. In the fastest case, this means that HOLD asserts one clock cycle before the HLDA response.

Step 2 $\overline{EADS}$ and INV are asserted. If INV is 0, snooping is caused by a read access. If INV is 1, snooping is caused by a write access. $\overline{EADS}$ is not sampled again until after the modified line is written back to memory. It is detected again as early as in Step 11.



Note:
The circled numbers in this figure represent the steps in section 4.8.4.

Figure 8. Snoop That Hits a Modified Line (Write-Back)

Step 3 Two clock cycles after $\overline{EADS}$ is asserted, $\overline{HITM}$ becomes valid, and is 0 because the line is modified.

Step 4 In the next clock, the core system logic deasserts the HOLD signal in response to the $\overline{HITM}$ = 0 signal. The core system logic backs off the current bus master at the same time so that the microprocessor can access the bus. HOLD can be reasserted immediately after $\overline{ADS}$ is asserted for burst cycles.

Step 5 The snooping cache starts it's write-back of the modified line by asserting $\overline{ADS}$ = 0, $\overline{CACHE}$ = 0, and W/$\overline{R}$ = 1. The write access is a burst write. The number of clock cycles between deasserting HOLD to the snooping cache and first asserting $\overline{ADS}$ for the write-back cycles can vary. In this example, it is one clock cycle, which is the shortest possible time. Regardless of the number of clock cycles, the start of the write-back is seen by $\overline{ADS}$ going Low.

Step 6 The write-back access is finished when $\overline{BLAST}$ and $\overline{BRDY}$ both are 0.

Step 7 In the clock cycle after the final write-back access, the processor drives $\overline{HITM}$ back to 1.

Step 8 HOLD is sampled by the microprocessor.

Step 9 One cycle after sampling HOLD High, the microprocessor transitions HLDA transitions to 1, acknowledging the HOLD request.

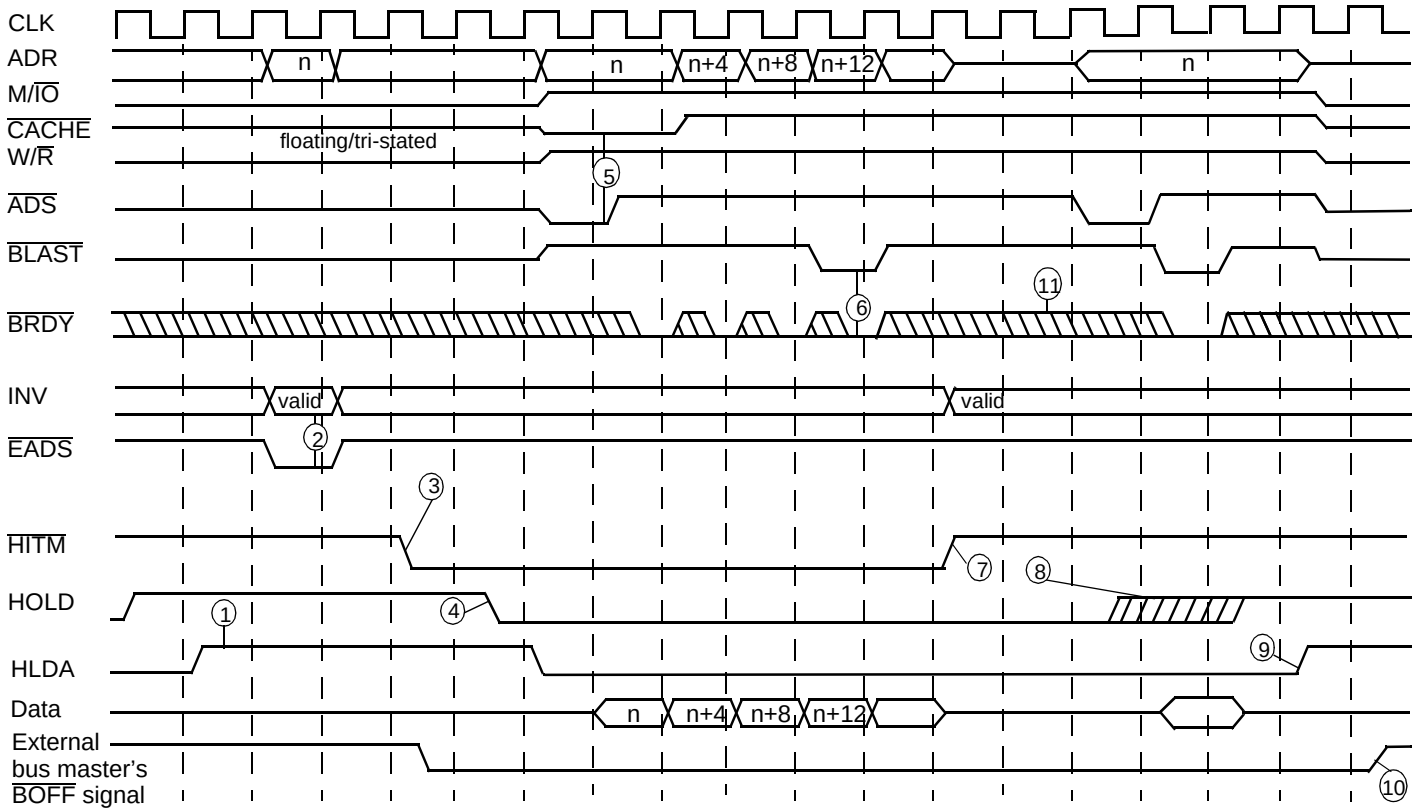
Step 10 The core system logic removes hold-off control to the external bus master. This allows the external bus master to immediately retry the aborted access. $\overline{ADS}$ is strobed Low, which generates $\overline{EADS}$ Low in the same clock cycle.

Step 11 The bus master restarts the aborted access. $\overline{EADS}$ and INV are applied to the microprocessor as before. This starts another snoop cycle.

The status of the addressed line is now either shared (INV = 0) or is changed to invalid (INV = 1).

4.8.5 Write-Back and Pending Access

Scenario: The following occurs when, in addition to the write-back operation, other bus accesses initiated by the processor associated with the snooped cache are pending. The microprocessor gives the write-back access priority. This implies that if HOLD is deasserted, the microprocessor first writes back the modified line (see Figure 9).



Note:
The circled numbers in this figure represent the steps in section 4.8.5.

Figure 9. Write-Back and Pending Access

Step 1 HOLD places the microprocessor in Snooping mode. HLDA must be High for a minimum of one clock cycle before $\overline{EADS}$ assertion. In the fastest case, this means that HOLD asserts one clock cycle before the HLDA response.

Step 2 $\overline{EADS}$ and INV are asserted. If INV is 0, snooping is caused by a read access. If INV is 1, snooping is caused by a write access. $\overline{EADS}$ is not sampled again until after the modified line is written back to memory. It is detected again as early as in Step 11.

Step 3 Two clock cycles after $\overline{EADS}$ is asserted, $\overline{HITM}$ becomes valid, and is 0 because the line is modified.

Step 4 In the next clock the core system logic deasserts the HOLD signal in response to the $\overline{HITM}$ = 0. The core system logic backs off the current bus master at the same time so that the microprocessor can access the bus. HOLD can be reasserted immediately after $\overline{ADS}$ is asserted for burst cycles.

Step 5 The snooping cache starts its write-back of the modified line by asserting $\overline{ADS}$ = 0, $\overline{CACHE}$ = 0, and W/$\overline{R}$ = 1. The write access is a burst write. The number of clock cycles between deasserting HOLD to the snooping cache and first asserting $\overline{ADS}$ for the write-back cycles can vary. In this example, it is one clock cycle, which is the shortest possible time. Regardless of the number of clock cycles, the start of the write-back is seen by $\overline{ADS}$ going Low.

Step 6 The write-back access is finished when $\overline{BLAST}$ and $\overline{BRDY}$ both are 0.

Step 7 In the clock cycle after the final write-back access, the processor drives $\overline{HITM}$ back to 1.

Step 8 HOLD is sampled by the microprocessor.

Step 9 A minimum of 1 clock cycle after the completion of the pending access, HLDA transitions to 1, acknowledging the HOLD request.

Step 10 The core system logic removes hold-off control to the external bus master. This allows the external bus master to immediately retry the aborted access. $\overline{ADS}$ is strobed Low, which generates $\overline{EADS}$ Low in the same clock cycle.

Step 11 The bus master restarts the aborted access. $\overline{EADS}$ and INV are applied to the microprocessor as before. This starts another snoop cycle.

The status of the addressed line is now either shared (INV = 0) or is changed to invalid (INV = 1).

4.8.5.1 HOLD/HLDA Write-Back Design Considerations

When designing a write-back cache system that uses HOLD/HLDA as the bus arbitration method, the following considerations must be observed to ensure proper operation (see Figure 10).

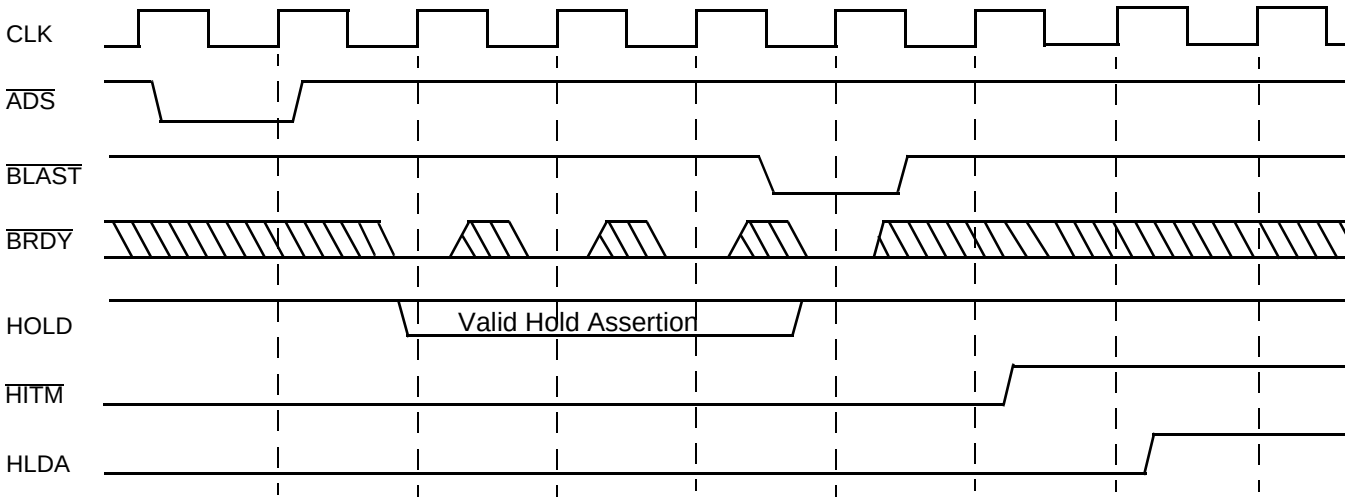


Figure 10. Valid HOLD Assertion During Write-Back

Step 1 During a snoop to the on-chip cache that hits a modified cache line, the HOLD signal cannot be deasserted to the microprocessor until the next clock cycle after $\overline{HITM}$ transitions active.

Step 2 After the write-back has commenced, the HOLD signal should be asserted no earlier than the next clock cycle after $\overline{ADS}$ goes active, and no later than in the final $\overline{BRDY}$ of the last write. Asserting HOLD later than the final $\overline{BRDY}$ may allow the microprocessor to permit a pending access to begin.

Step 3 If $\overline{RDY}$ is returned instead of $\overline{BRDY}$ during a write-back, the HOLD signal can be reasserted at any time starting one clock after $\overline{ADS}$ goes active in the first transfer up to the final transfer when $\overline{RDY}$ is asserted. Asserting $\overline{RDY}$ instead of $\overline{BRDY}$ will not break the write-back cycle if HOLD is asserted. The processor ignores HOLD until the final write cycle of the write-back.

4.8.5.2 AHOLD Bus Arbitration Implementation

The use of AHOLD as the control mechanism is often found in systems where an external second-level cache is closely coupled to the microprocessor. This tight coupling allows the microprocessor to operate with the least amount of stalling from external snooping of the on-chip cache. Additionally, snooping of the cache can be performed concurrently with an access by the microprocessor. This feature further improves the performance of the total system (see Figure 11).

Note: *To maintain proper system timing, the AHOLD signal must remain active for one clock cycle after $\overline{HITM}$ transitions active. Deassertion of AHOLD in the same clock cycle as $\overline{HITM}$ assertion may lead to unpredictable processor behavior.*

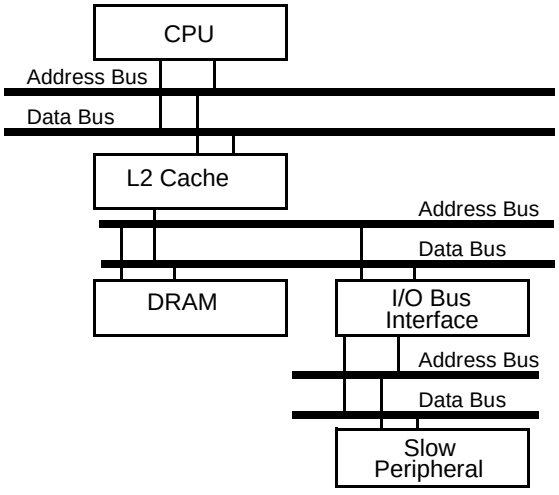


Figure 11. Closely Coupled Cache Block Diagram

The following sections describe the snooping scenarios for the AHOLD implementation.

4.8.5.3 Normal Write-Back

Scenario: This scenario assumes that a processor-initiated access has already started and that the external logic can finish that access even without the address being applied after the first clock cycle. Therefore, a snooping access with AHOLD can be done in parallel. In this case, the processor-initiated access is finished first, then the write-back is executed (see Figure 12).

The sequence is as follows:

Step 1 The processor initiates an external, simple, non-cacheable read access, strobing $\overline{ADS}$ = 0 and W/$\overline{R}$ = 0. The address is driven from the CPU.

Step 2 In the same cycle, AHOLD is asserted to indicate the start of snooping. The address bus floats and becomes an input in the next clock cycle.

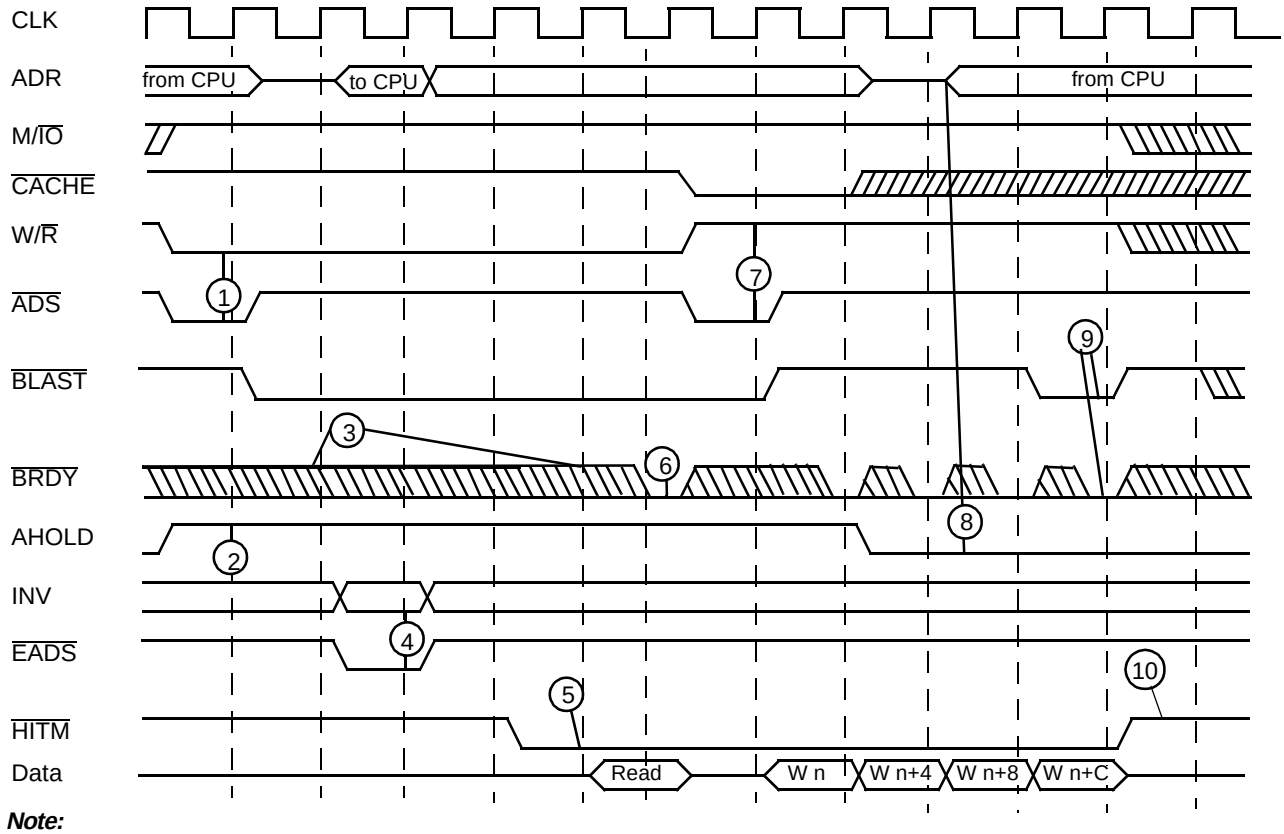
Step 3 During the next clock cycles, the $\overline{BRDY}$ or $\overline{RDY}$ signal is not strobed Low. Therefore, the processor-initiated access is not finished.

Step 4 Two clock cycles after AHOLD is asserted, the $\overline{EADS}$ signal is activated to start an actual snooping cycle, and INV is valid. If INV is 0, a read access caused the snooping cycle. If INV is 1, a write access caused the snooping cycle. Additional $\overline{EADS}$ are ignored due to the hit of a modified line. It is detected after $\overline{HITM}$ goes inactive.

Step 5 Two clock cycles after $\overline{EADS}$ is asserted, the snooping signal $\overline{HITM}$ becomes valid. The line is modified; therefore, $\overline{HITM}$ is 0.

Step 6 In this cycle, the processor-initiated access is finished.

Step 7 Two clock cycles after the end of the processor-initiated access, the cache immediately starts writing back the modified line. This is indicated by $\overline{ADS}$ = 0 and W/$\overline{R}$ = 1. Note that AHOLD is still active and the address bus is still an input. However, the write-back access can be executed without any address. This is because the corresponding address must have been on the bus when $\overline{EADS}$ was strobed. Therefore, in the case of the core system logic, the address for the write-back must be latched with $\overline{EADS}$ to be available later. This is required only if AHOLD is not removed if $\overline{HITM}$ becomes 0. Otherwise, the address of the write-back is put onto the address bus by the microprocessor.



Note:
The circled numbers in this figure represent the steps in section 4.8.5.3.

Figure 12. Snoop Hit Cycle with Write-Back

Step 8 As an example, AHOLD is now removed. In the next clock cycle, the current address of the write-back access is driven onto the address bus.

Step 9 The write-back access is finished when $\overline{\text{BLAST}}$ and $\overline{\text{BRDY}}$ both transition to 0.

Step 10 In the clock cycle after the final write-back access, the snooping cache drives $\overline{\text{HITM}}$ back to 1.

The status of the snooped and written-back line is now either shared (INV = 0) or is changed to invalid (INV = 1).

4.8.6 Reordering of Write-Backs (AHOLD) with $\overline{\text{BOFF}}$

As seen previously, the Bus Interface Unit (BIU) completes the processor-initiated access first if the snooping access occurs after the start of the processor-initiated access. If the $\overline{\text{HITM}}$ signal occurs one clock cycle before the $\overline{\text{ADS}}$ = 0 of the processor-initiated access, the write-back receives priority and is executed first.

However, if the snooping access is executed after the start of the processor-initiated access, there is a methodology to reorder the access order. The $\overline{\text{BOFF}}$ signal delays outstanding processor-initiated cycles so that a snoop write-back can occur immediately (see Figure 13).

Scenario: If there are outstanding processor-initiated cycles on the bus, asserting $\overline{\text{BOFF}}$ clears the bus pipeline. If a snoop causes $\overline{\text{HITM}}$ to be asserted, the first cycle issued by the microprocessor after deassertion of $\overline{\text{BOFF}}$ is the write-back cycle. After the write-back cycle, it reissues the aborted cycles. This translates into the following sequence:

Step 1 The processor starts a cacheable burst read cycle.

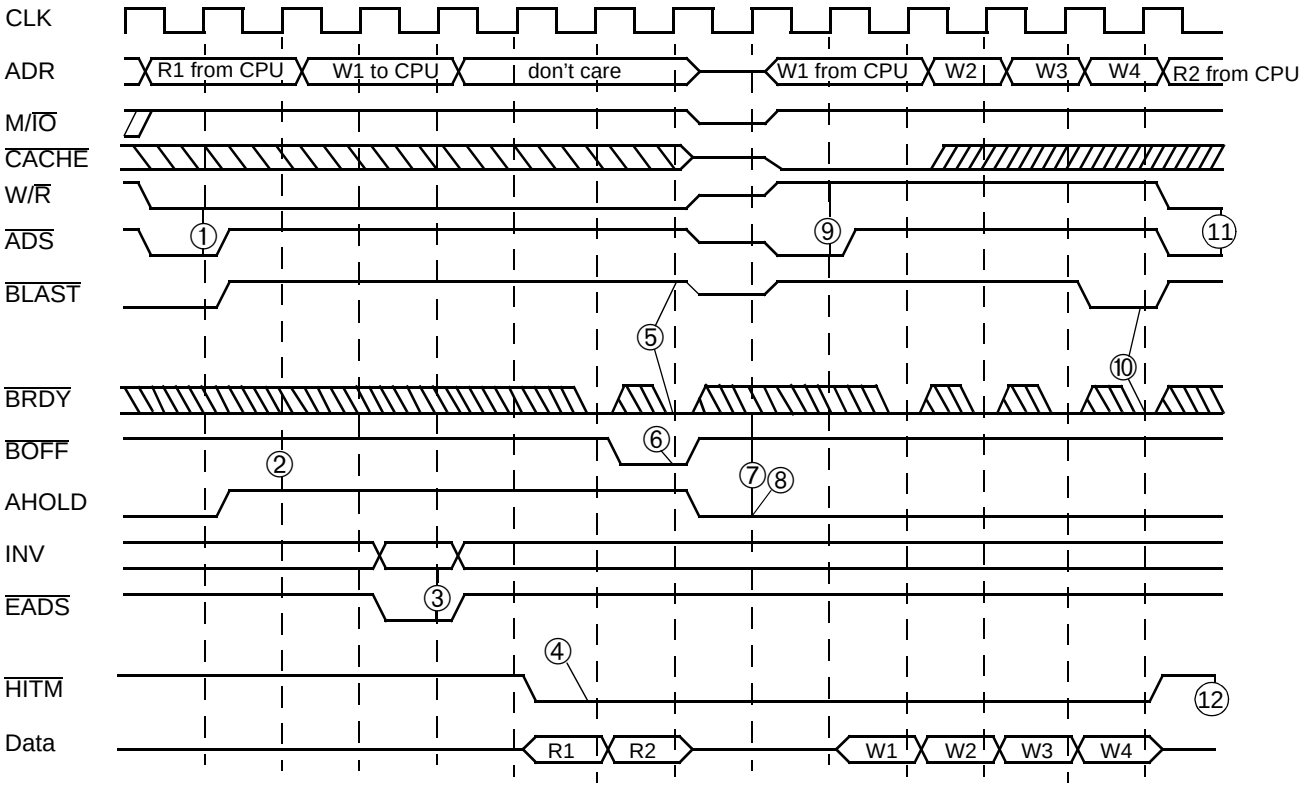
Step 2 One clock cycle later, AHOLD is asserted. This switches the address bus into an input one clock cycle after AHOLD is asserted.

Step 3 Two clock cycles after AHOLD is asserted, the $\overline{\text{EADS}}$ and INV signals are asserted to start the snooping cycle.

Step 4 Two clock cycles after $\overline{\text{EADS}}$ is asserted, $\overline{\text{HITM}}$ becomes valid. The line is modified, therefore $\overline{\text{HITM}}$ = 0.

Step 5 Note that the processor-initiated access is not completed because $\overline{\text{BLAST}}$ = 1.

Step 6 With $\overline{\text{HITM}}$ going Low, the core system logic asserts $\overline{\text{BOFF}}$ in the next clock cycle to the snooping processor to reorder the access. $\overline{\text{BOFF}}$ overrides $\overline{\text{BRDY}}$. Therefore, the partial read is not used. It is reread later.



Note:
The circled numbers in this figure represent the steps in section 4.8.6.

Figure 13. Cycle Reordering with $\overline{\text{BOFF}}$ (Write-Back)

Step 7 One clock cycle later $\overline{\text{BOFF}}$ is deasserted. The write-back access starts one clock cycle later because the $\overline{\text{BOFF}}$ has cleared the bus pipeline.

Step 8 AHOLD is deasserted. In the next clock cycle the address for the write-back is driven on the address bus.

Step 9 One cycle after $\overline{\text{BOFF}}$ is deasserted, the cache immediately starts writing back the modified line. This is indicated by $\overline{\text{ADS}}$ = 0 and W/$\overline{\text{R}}$ = 1.

Step 10 The write-back access is finished when $\overline{\text{BLAST}}$ and $\overline{\text{BRDY}}$ go active 0.

Step 11 The BIU restarts the aborted cache line fill with the previous read. This is indicated by $\overline{\text{ADS}}$ = 0 and W/$\overline{\text{R}}$ = 0.

Step 12 In the same clock cycle, the snooping cache drives $\overline{\text{HITM}}$ back to 1.

Step 13 The previous read is now reread.

4.8.7 Special Scenarios for AHOLD Snooping

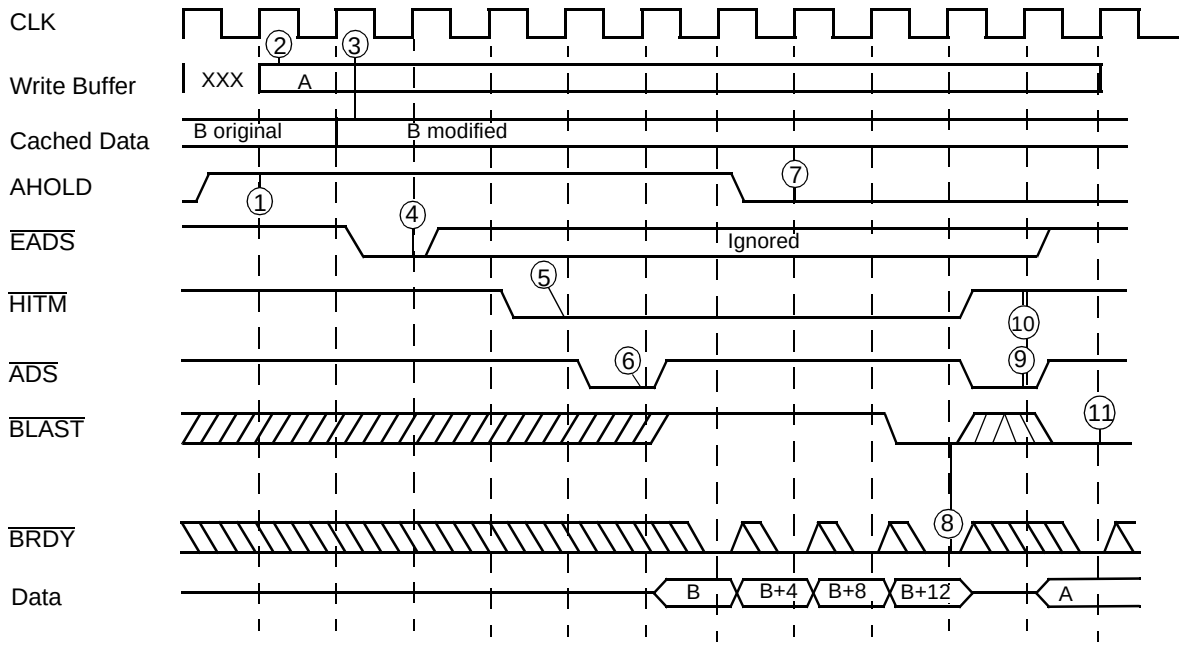
In addition to the previously described scenarios, there are special scenarios regarding the time of the $\overline{\text{EADS}}$ and AHOLD assertion. The final result depends on the time $\overline{\text{EADS}}$ and AHOLD are asserted relative to other processor-initiated operations.

4.8.7.1 Write Cycle Reordering due to Buffering

Scenario: The MESI cache protocol and the ability to perform and respond to snoop cycles guarantee that writes to the cache are logically equivalent to writes to memory. In particular, the order of read and write operations on cached data is the same as if the operations were on data in memory. Even non-cached memory read and write requests usually occur on the external bus in the same order that they were issued in the program. For example, when a write miss is followed by a read miss, the write data goes on the bus before the read request is put on the bus. However, the posting of writes in write buffers coupled with snooping cycles may cause the order of writes seen on the external bus to differ from the order they appear in the program. Consider the following example, which is illustrated in Figure 14. For simplicity, snooping signals that behave in their usual manner are not shown.

Step 1 AHOLD is asserted. No further processor-initiated accesses to the external bus can be started. No other access is in progress.

Step 2 The processor writes data A to the cache, resulting in a write miss. Therefore, the data is put into the write buffers, assuming they are not full. No external access can be started because AHOLD is still 1.



Note:
The circled numbers in this figure represent the steps in section 4.8.7.1.

Figure 14. Write Cycle Reordering Due to Buffering

Step 3 The next write of the processor hits the cache and the line is non-shared. Therefore, data B is written into the cache. The cache line transits to the modified state.

Step 4 In the same clock cycle, a snoop request to the same address where data B resides is started because $\overline{EADS}$ = 0. The snoop hits a modified line. $\overline{EADS}$ is ignored due to the hit of a modified line, but is detected again as early as in step 10.

Step 5 Two clock cycles after $\overline{EADS}$ asserts, $\overline{HITM}$ becomes valid.

Step 6 Because the processor-initiated access cannot be finished (AHOLD is still 1), the BIU gives priority to a write-back access that does not require the use of the address bus. Therefore, in the clock cycle, the cache starts the write-back sequence indicated by $\overline{ADS}$ = 0 and W/$\overline{R}$ = 0.

Step 7 During the write-back sequence, AHOLD is deasserted.

Step 8 The write-back access is finished when $\overline{BLAST}$ and $\overline{BRDY}$ transition to 0.

Step 9 After the last write-back access, the BIU starts writing data A from the write buffers. This is indicated by $\overline{ADS}$ = 0 and W/$\overline{R}$ = 0.

Step 10 In the same clock cycle, the snooping cache drives $\overline{HITM}$ back to 1.

Step 11 The write of data A is finished if $\overline{BRDY}$ transitions to 0 ($\overline{BLAST}$ = 0), because it is a single word.

The software write sequence was first data A and then data B. But on the external bus the data appear first as data B and then data A. The order of writes is changed. In most cases, it is unnecessary to strictly maintain the ordering of writes. However, some cases (for example, writing to hardware control registers) require writes to be observed externally in the same order as programmed. There are two options to ensure serialization of writes, both of which drive the cache to Write-through mode:

1. Set the PWT bit in the page table entries.
2. Drive the WB/$\overline{WT}$ signal Low when accessing these memory locations.

Option 1 is an operating-system-level solution not directly implemented by user-level code. Option 2, the hardware solution, is implemented at the system level.

4.8.7.2 BOFF Write-Back Arbitration Implementation

The use of BOFF to perform snooping of the on-chip cache is used in systems where more than one cacheable bus master resides on the microprocessor bus. The BOFF signal forces the microprocessor to relinquish the bus in the following clock cycle, regardless of the type of bus cycle it was performing at the time. Consequently, the use of BOFF as a bus arbitrator should be implemented with care to avoid system problems.

4.8.8 BOFF Design Considerations

The use of BOFF as a bus arbitration control mechanism is immediate. BOFF forces the microprocessor to abort an access in the following clock cycle after it is asserted. The following design issues must be considered.

4.8.8.1 Cache Line Fills

The microprocessor aborts a cache line fill during a burst read if BOFF is asserted during the access. Upon regaining the bus, the read access commences where it left off when BOFF was recognized. External buffers should take this cycle continuation into consideration if BOFF is allowed to abort burst read cycles.

4.8.8.2 Cache Line Copy-Backs

Similar to the burst read, the burst write also can be aborted at any time with the BOFF signal. Upon regaining access to the bus, the write continues from where it was aborted. External buffers and control logic should take into consideration the necessary control, if any, for burst write continuations.

4.8.8.3 Locked Accesses

Locked bus cycles occur in various forms. Locked accesses occur during read-modify-write operations, interrupt acknowledges, and page table updates. Although asserting BOFF during a locked cycle is permitted, extreme care should be taken to ensure data coherency for semaphore updates and proper data ordering.

4.8.9 BOFF During Write-Back

If BOFF is asserted during a write-back, the processor performing the write-back goes off the bus in the next clock cycle. If BOFF is released, the processor restarts that write-back access from the point at which it was aborted. The behavior is identical to the normal BOFF case that includes the abort and restart behavior.

4.8.10 Snooping Characteristics During a Cache Line Fill

The microprocessor takes responsibility for responding to snoop cycles for a cache line only during the time that the line is actually in the cache or in a copy-back buffer. There are times during the cache line fill cycle and during the cache replacement cycle when the line is “in transit” and snooping responsibility must be taken by other system components.

The following cases apply if snooping is invoked via AHOLD, and neither HOLD nor BOFF is asserted.

- System designers should consider the possibility that a snooping cycle may arrive at the same time as a cache line fill or replacement for the same address. If a snooping cycle arrives at the same time as a cache line fill with the same address, the CPU uses the cache line fill, but does not place it in the cache.
- If a snooping cycle occurs at the same time as a cache line fill with a different address, the cache line fill is placed into the cache unless EADS is recognized before the first BRDY but after ADS is asserted, or EADS is recognized on the last BRDY of the cache line fill. In these cases, the line is not placed into the cache.

4.8.11 Snooping Characteristics During a Copy-Back

If a copy-back is occurring because of a cache line replacement, the address being replaced can be matched by a snoop until assertion of the last BRDY of the copy-back. This is when the modified line resides in the copy-back buffer. An EADS as late as two clocks before the last BRDY can cause HITM to be asserted.

Figure 15 illustrates the microprocessor relinquishing responsibility of recognizing snoops for a line that is copied back. It shows the latest EADS assertion that can cause HITM assertion. HITM remains active for only one clock period in that example. HITM remains active through the last BRDY of the corresponding write-back; in that case, the write-back has already completed. This is the latest point where snooping can start, because two clock cycles later, the final BRDY of the write-back is applied.

If a snoop cycle hits the copy-back address after the first BRDY of the copy-back and ADS has been issued, the microprocessor asserts HITM. Keep in mind that the write-back was initiated due to a read miss and not due to a snoop to a modified line. In the second case, no snooping is recognized if a modified line is detected.

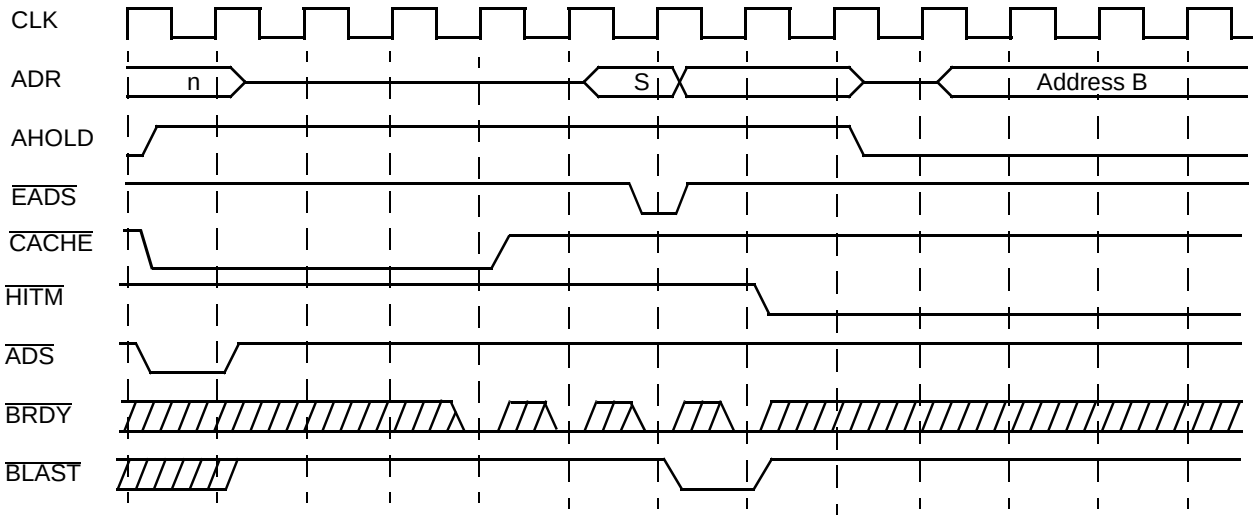


Figure 15. Latest Snooping of Copy-Back

4.9 Cache Invalidation and Flushing in Write-Back Mode

The Am5$_X$86 microprocessor family supports cache invalidation and flushing, much like the Am486 microprocessor Write-through mode. However, the addition of the write-back cache adds some complexity.

4.9.1 Cache Invalidation through Software

To invalidate the on-chip cache, the Am5$_X$86 microprocessor family uses the same instructions as the Am486 microprocessor family. The two invalidation instructions, INVD and WBINVD, while similar, are slightly different for use in the write-back environment.

The WBINVD instruction first performs a write-back of the modified data in the cache to external memory. Then it invalidates the cache, followed by two special bus cycles. The INVD instruction only invalidates the cache, regardless of whether modified data exists, and follows with a special bus cycle. The utmost care should be taken when executing the INVD instruction to ensure memory coherency. Otherwise, modified data may be invalidated prior to writing back to main memory. In Write-back mode, WBINVD requires a minimum of 4100 internal clocks to search the cache for modified data. Writing back modified data adds to this minimum time. WBINVD can only be stopped by a RESET.

Two special bus cycles follow the write-back of modified data upon execution of the WBINVD instruction: first the write-back, and then the flush special bus cycle. The INVD operates identically to the standard 486 microprocessor family in that the flush special bus cycle is generated when the on-chip cache is invalidated. Table 7 specifies the special bus cycle states for the instructions WBINVD and INVD.

Table 7. WBINVD/INVD Special Bus Cycles

A32–A2	M/$\overline{IO}$	D/$\overline{C}$	W/$\overline{R}$	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	Bus Cycle
0000 0000 h	0	0	1	0	1	1	1	Write-back[1]
0000 0000 h	0	0	1	1	1	0	1	Flush[1, 2]

Notes:
1. WBINVD generates first write-back, then flush.
2. INVD generates only flush.

4.9.2 Cache Invalidation through Hardware

The other mechanism for cache invalidation is the $\overline{FLUSH}$ pin. The $\overline{FLUSH}$ pin operates similarly to the WBINVD command, writing back modified cache lines to main memory. After the entire cache has copied back all the modified data, the microprocessor generates two special bus cycles. These special bus cycles signal to the external caches that the microprocessor on-chip cache has completed its copy-back and that the second level cache may begin its copy-back to memory, if so required.

Two flush acknowledge cycles are generated after the $\overline{FLUSH}$ pin is asserted and the modified data in the cache is written back. As with the WBINVD instruction, in Write-back mode, a flush requires a minimum of 4100 internal clocks to test the cache for modified data. Writing back modified data adds to this minimum time. The flush operation can only be stopped by a RESET. Table 8 shows the special flush bus cycle configuration.

Table 8. $\overline{\text{FLUSH}}$ Special Bus Cycles

A32–A2	M/$\overline{IO}$	D/$\overline{C}$	W/$\overline{R}$	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	Bus Cycle
0000 0001h	0	0	1	0	1	1	1	First Flush Acknowledge
0000 0001h	0	0	1	1	1	0	1	Second Flush Acknowledge

4.9.3 Snooping During Cache Flushing

As with snooping during normal operation, snooping is permitted during a cache flush, whether initiated by the $\overline{\text{FLUSH}}$ pin or WBINVD instruction. After completion of the snoop, and write-back, if needed, the microprocessor completes the copy-back of modified cache lines.

4.10 Burst Write

The Am5$_X$86 microprocessor improves system performance by implementing a burst write feature for cache line write-backs and copy-backs. Standard write operations are still supported. Burst writes are always four 32-bit words and start at the beginning of a cache line address of 0 for the starting access. The timing of the $\overline{\text{BLAST}}$ and $\overline{\text{BRDY}}$ signals is identical to the burst read. Figure 16 shows a burst write access. (See Figure 17 and Figure 18 for burst read and burst write access with $\overline{\text{BOFF}}$ asserted.) In addition to using $\overline{\text{BLAST}}$, the $\overline{\text{CACHE}}$ signal indicates burstable cycles.

$\overline{\text{CACHE}}$ is a cycle definition pin used when in Write-back mode ($\overline{\text{CACHE}}$ floats in Write-through mode). For processor-initiated cycles, the signal indicates:

- For a read cycle, the internal cacheability of the cycle
- For a write cycle, a burst write-back or copy-back, if $\overline{\text{KEN}}$ is asserted (for linefills).

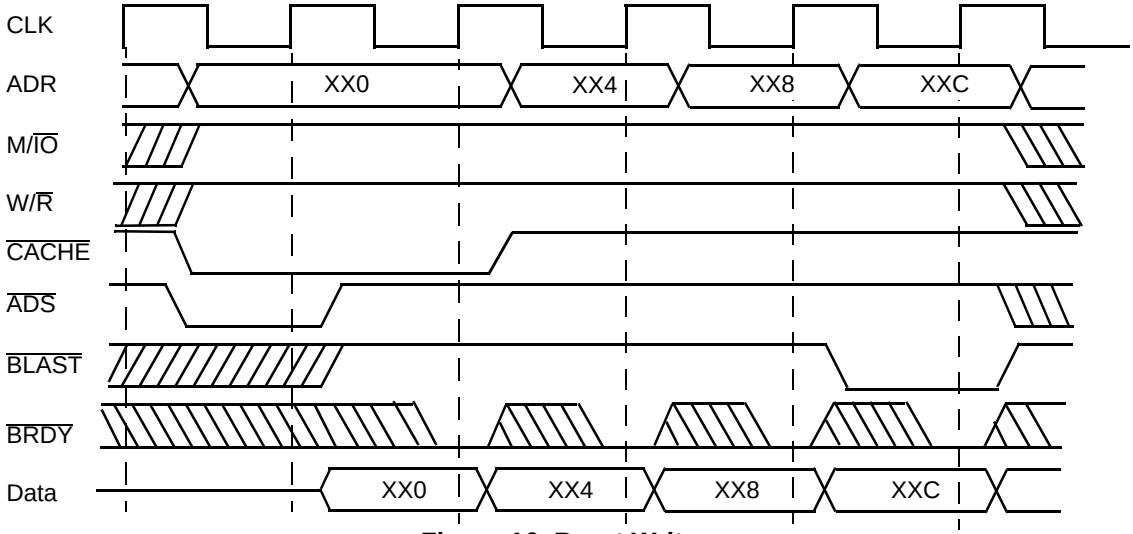


Figure 16. Burst Write

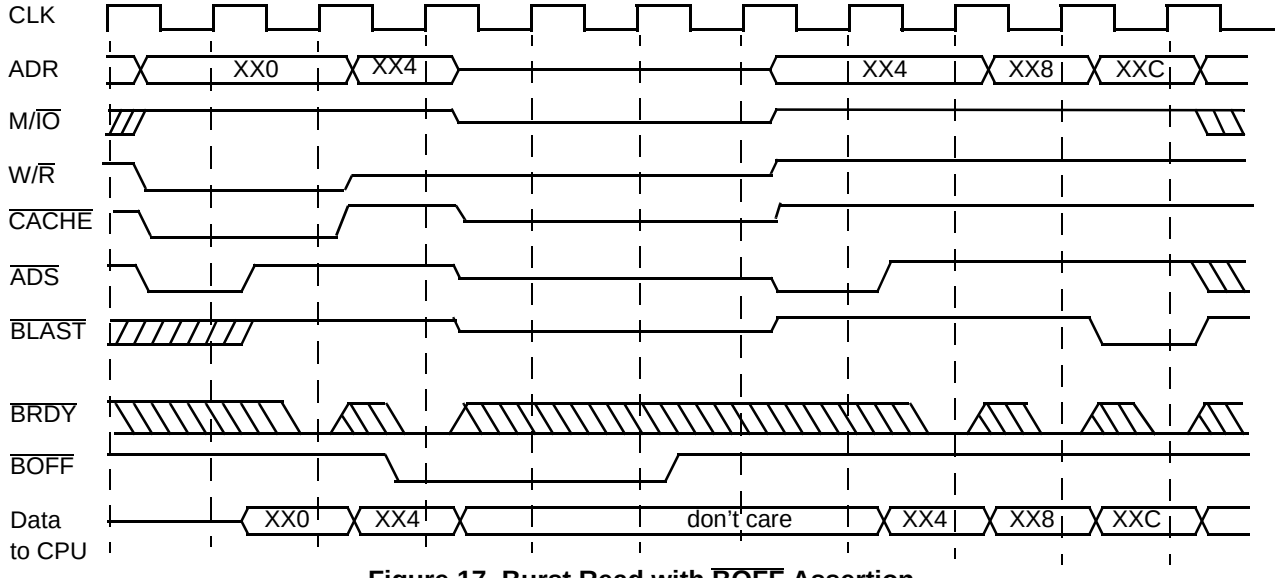


Figure 17. Burst Read with $\overline{\text{BOFF}}$ Assertion

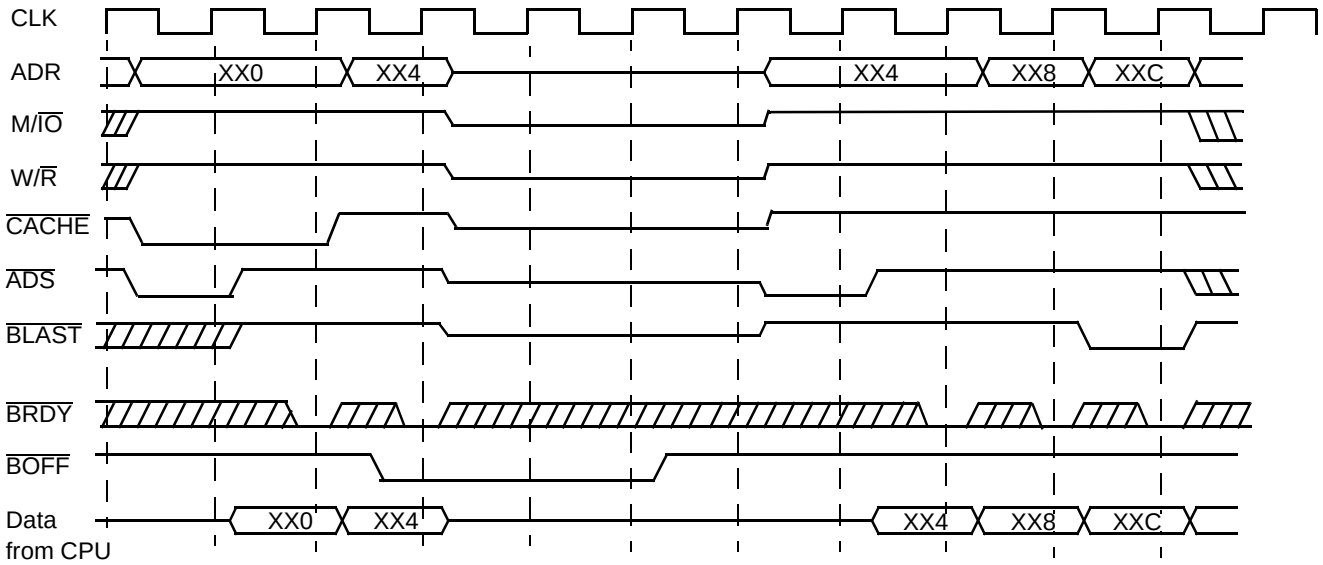


Figure 18. Burst Write with BOFF Assertion

CACHE is asserted for cacheable reads, cacheable code fetches, and write-backs/copy-backs. CACHE is deasserted for non-cacheable reads, translation lookaside buffer (TLB) replacements, locked cycles (except for write-back cycles generated by an external snoop operation that interrupts a locked read/modify/write sequence), I/O cycles, special cycles, and write-throughs. CACHE is driven to its valid level in the same clock as the assertion of ADS and remains valid until the next RDY or BRDY assertion. The CACHE output pin floats one clock after BOFF is asserted. Additionally, the signal floats when HLDA is asserted.

The following steps describe the burst write sequence:

1. The access is started by asserting: ADS = 0, M/IO = 1, W/R = 1, CACHE = 0. The address offset always is 0, so the burst write always starts on a cache line boundary. CACHE transitions High (inactive) after the first BRDY.
2. In the second clock cycle, BLAST is 1 to indicate that the burst is not finished.
3. The burst write access is finished when BLAST is 0 and BRDY is 0.

When the RDY signal is returned instead of the BRDY signal, the $Am5_X86$ microprocessor halts the burst cycle and proceeds with the standard non-burst cycle.

4.10.1 Locked Accesses

Locked accesses of an $Am5_X86$ microprocessor occur for read-modify-write operations and interrupt acknowledge cycles. The timing is identical to the DX microprocessor, although the state transitions differ from the standard DX microprocessor. Unlike processor-initiated accesses, state transitions for locked accesses are seen by all processors in the system. Any locked read or write generates an external bus cycle, regardless of cache hit or miss. During locked cycles, the processor does not recognize a HOLD request, but it does recognize BOFF and AHOLD requests.

Locked read operations always read data from the external memory, regardless of whether the data is in the cache. In the event that the data is in the cache and unmodified, the cache line is invalidated and an external read operation is performed. The data from the external memory is used instead of the data in the cache, thus ensuring that the locked read is seen by all other bus masters. If a locked read occurs, the data is in the cache, and it is modified. The microprocessor first copies back the data to external memory, invalidates the cache line, and then performs a read operation to the same location, thus ensuring that the locked read is seen by all other bus masters. At no time is the data in the cache used directly by the microprocessor or a locked read operation before reading the data from external memory. Since locked cycles always begin with a locked read access, and locked read cycles always invalidate a cache line, a locked write cycle to a valid cache line, either modified or unmodified, does not occur.

4.10.2 Serialization

Locked accesses are totally serialized:

- All reads and writes in the write buffer that precede the locked access are issued on the bus before the first locked access is executed.
- No read or write after the last locked access is issued internally or on the bus until the final RDY or BRDY for all locked accesses.
- It is possible to get a locked read, write-back, locked write cycle.

4.10.3 $\overline{\text{PLOCK}}$ Operation in Write-Through Mode

As described in Section 3, $\overline{\text{PLOCK}}$ is only used in Write-through mode; the signal is driven inactive in Write-back mode. In Write-through mode, the processor drives $\overline{\text{PLOCK}}$ Low to indicate that the current bus transaction requires more than one bus cycle. The CPU continues to drive the signal Low until the transaction is completed, whether or not $\overline{\text{RDY}}$ or $\overline{\text{BRDY}}$ is returned. Refer to the pin description for additional information.

5 CLOCK CONTROL

5.1 Clock Generation

The Am5$_X$86 CPU is driven by a 1x clock that relies on phased-lock loop (PLL) to generate the two internal clock phases: phase one and phase two. The rising edge of CLK corresponds to the start of phase one (ph1). All external timing parameters are specified relative to the rising edge of CLK.

5.2 Stop Clock

The Am5$_X$86 CPU also provides an interrupt mechanism, $\overline{\text{STPCLK}}$, that allows system hardware to control the power consumption of the CPU by stopping the internal clock to the CPU core in a sequenced manner. The first low-power state is called the Stop Grant state. If the CLK input is completely stopped, the CPU enters into the Stop Clock state (the lowest power state). When the CPU recognizes a $\overline{\text{STPCLK}}$ interrupt, the processor:

- Stops execution on the next instruction boundary (unless superseded by a higher priority interrupt)
- Waits for completion of cache flush
- Stops the pre-fetch unit
- Empties all internal pipelines and write buffers
- Generates a Stop Grant bus cycle
- Stops the internal clock

At this point the CPU is in the Stop Grant state.

The CPU cannot respond to a $\overline{\text{STPCLK}}$ request from an HLDA state because it cannot empty the write buffers and, therefore, cannot generate a Stop Grant cycle. The rising edge of $\overline{\text{STPCLK}}$ signals the CPU to return to program execution at the instruction following the interrupted instruction. Unlike the normal interrupts (INTR and NMI), $\overline{\text{STPCLK}}$ does not initiate interrupt acknowledge cycles or interrupt table reads.

5.2.1 External Interrupts in Order of Priority

In Write-through mode, the priority order of external interrupts is:

1. RESET/SRESET
2. $\overline{\text{FLUSH}}$
3. $\overline{\text{SMI}}$
4. NMI
5. INTR
6. $\overline{\text{STPCLK}}$

In Write-back mode, the priority order of external interrupts is:

1. RESET
2. $\overline{\text{FLUSH}}$
3. SRESET
4. $\overline{\text{SMI}}$
5. NMI
6. INTR
7. $\overline{\text{STPCLK}}$

$\overline{\text{STPCLK}}$ is active Low and has an internal pull-up resistor. $\overline{\text{STPCLK}}$ is asynchronous, but setup and hold times must be met to ensure recognition in any specific clock. $\overline{\text{STPCLK}}$ must remain active until the Stop Grant special bus cycle is asserted and the system responds with either $\overline{\text{RDY}}$ or $\overline{\text{BRDY}}$. When the CPU enters the Stop Grant state, the internal pull-up resistor is disabled, reducing the CPU power consumption. The $\overline{\text{STPCLK}}$ input must be driven High (not floated) to exit the Stop Grant state. $\overline{\text{STPCLK}}$ must be deasserted for a minimum of five clocks after $\overline{\text{RDY}}$ or $\overline{\text{BRDY}}$ is returned active for the Stop Grant bus cycle before being asserted again. There are two regions for the Low-power mode supply current:

1. *Low Power:* Stop Grant state (fast wake-up, frequency- and voltage-dependent)
2. *Lowest Power:* Stop Clock state (slow wake-up, voltage-dependent)

5.3 Stop Grant Bus Cycle

The processor drives a special Stop Grant bus cycle to the bus after recognizing the $\overline{\text{STPCLK}}$ interrupt. This bus cycle is the same as the HALT cycle used by a standard Am486 microprocessor, with the exception that the Stop Grant bus cycle drives the value 0000 0010h on the address pins.

- M/$\overline{\text{IO}}$ = 0
- D/$\overline{\text{C}}$ = 0
- W/$\overline{\text{R}}$ =1
- Address Bus = 0000 0010h (A_4 = 1)
- $\overline{\text{BE3}}$–$\overline{\text{BE0}}$ = 1011
- Data bus = undefined

The system hardware must acknowledge this cycle by returning $\overline{\text{RDY}}$ or $\overline{\text{BRDY}}$, or the processor will not enter the Stop Grant state (see Figure 19). The latency between a $\overline{\text{STPCLK}}$ request and the Stop Grant bus cycle depends on the current instruction, the amount of data in the CPU write buffers, and the system memory performance.

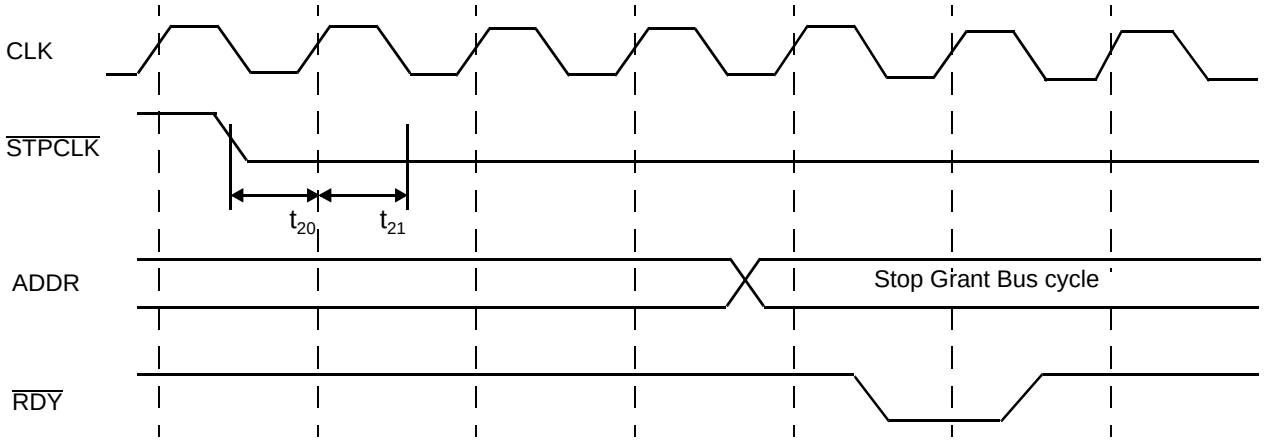


Figure 19. Entering Stop Grant State

5.4 Pin State During Stop Grant

Table 9 shows the pin states during Stop Grant Bus states. During the Stop Grant state, most output and input/output signals of the microprocessor maintain the level they held when entering the Stop Grant state. The data and data parity signals are tri-stated. In response to HOLD being driven active during the Stop Grant state (when the CLK input is running), the CPU generates HLDA and tri-states all output and input/output signals that are tri-stated during the HOLD/HLDA state. After HOLD is deasserted, all signals return to the same state they were before the HOLD/HLDA sequence.

Table 9. Pin State During Stop Grant Bus State

Signal	Type	State
A3–A2	O	Previous State
A31–A4	I/O	Previous State
D31–D0	I/O	Floated
$\overline{BE3}$–$\overline{BE0}$	O	Previous State
DP3–DP0	I/O	Floated
W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{CACHE}$	O	Previous State
$\overline{ADS}$	O	Inactive
$\overline{LOCK}$, $\overline{PLOCK}$	O	Inactive
BREQ	O	Previous State
HLDA	O	As per HOLD
$\overline{BLAST}$	O	Previous State
$\overline{FERR}$	O	Previous State
$\overline{PCHK}$	O	Previous State
$\overline{SMIACT}$	O	Previous State
$\overline{HITM}$	O	Previous State

To achieve the lowest possible power consumption during the Stop Grant state, the system designer must ensure that the input signals with pull-up resistors are not driven Low, and the input signals with pull-down resistors are not driven High.

All inputs except data bus pins must be driven to the power supply rails to ensure the lowest possible current consumption during Stop Grant or Stop Clock modes. For compatibility, data pins must be driven Low to achieve the lowest possible power consumption.

5.5 Clock Control State Diagram

Figure 20 shows the state transitions during a Stop Clock cycle.

5.5.1 Normal State

This is the normal operating state of the CPU. While in the normal state, the CLK input can be dynamically changed within the specified CLK period stability limits.

5.5.2 Stop Grant State

The Stop Grant state provides a low-power state that can be entered by simply asserting the external $\overline{STPCLK}$ interrupt pin. When the Stop Grant bus cycle has been placed on the bus, and either $\overline{RDY}$ or $\overline{BRDY}$ is returned, the CPU is in this state. The CPU returns to the normal execution state 10–20 clock cycles after $\overline{STPCLK}$ has been deasserted.

While in the Stop Grant state, the pull-up resistors on $\overline{STPCLK}$ and $\overline{UP}$ are disabled internally. The system must continue to drive these inputs to the state they were in immediately before the CPU entered the Stop Grant State. For minimum CPU power consumption, all other input pins should be driven to their inactive level while the CPU is in the Stop Grant state.

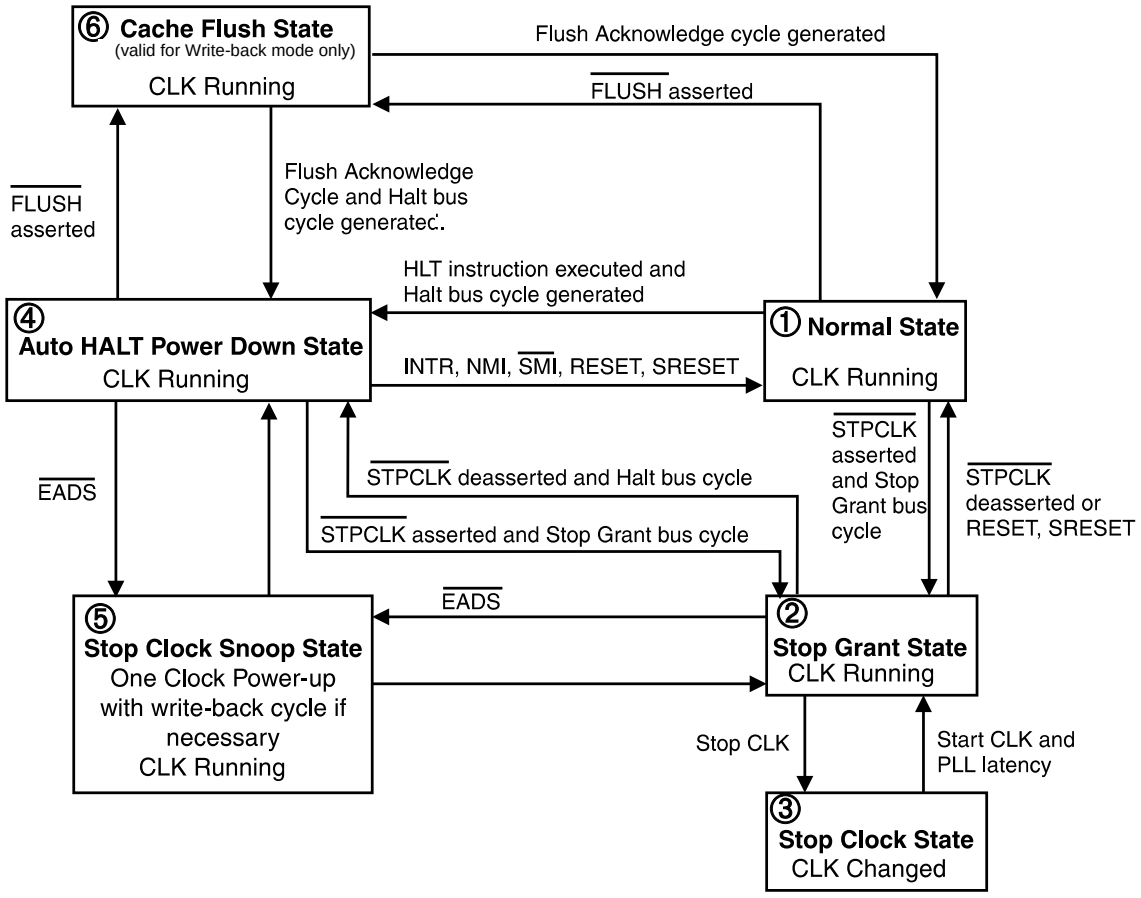
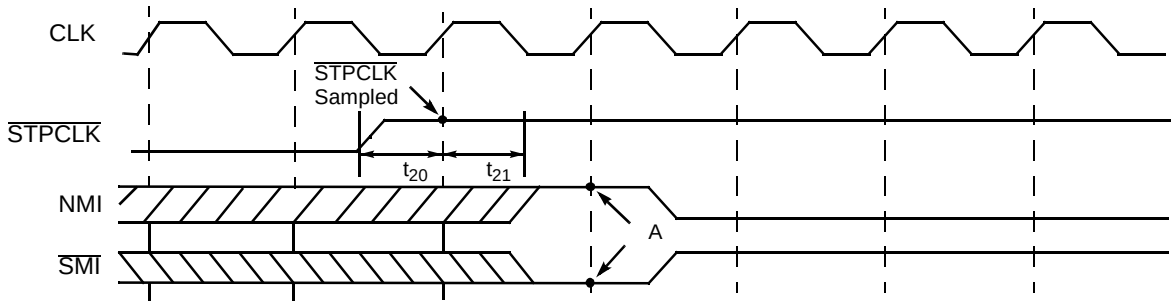


Figure 20. Stop Clock State Machine



Note: *A = Earliest time at which NMI or SMI is recognized.*

Figure 21. Recognition of Inputs when Exiting Stop Grant State

A RESET or SRESET brings the CPU from the Stop Grant state to the Normal state. The CPU recognizes the inputs required for cache invalidations (HOLD, AHOLD, BOFF, and EADS) as explained later. The CPU does not recognize any other inputs while in the Stop Grant state. Input signals to the CPU are not recognized until 1 clock after STPCLK is deasserted (see Figure 21).

While in the Stop Grant state, the CPU does not recognize transitions on the interrupt signals (SMI, NMI, and INTR). Driving an active edge on either SMI or NMI does not guarantee recognition and service of the interrupt request following exit from the Stop Grant state. However, if one of the interrupt signals (SMI, NMI, or INTR) is driven active while the CPU is in the Stop Grant state, and held active for at least one CLK after STPCLK is deasserted, the corresponding interrupt will be serviced.

The Am5$_X$86 CPU product family requires INTR to be held active until the CPU issues an interrupt acknowledge cycle to guarantee recognition. This condition also applies to the existing Am486 CPUs.

In the Stop Grant state, the system can stop or change the CLK input. When the clock stops, the CPU enters the Stop Clock state. The CPU returns to the Stop Grant state immediately when the CLK input is restarted. You must hold the $\overline{\text{STPCLK}}$ input Low until a stabilized frequency has been maintained for at least 1 ms to ensure that the PLL has had sufficient time to stabilize.

The CPU generates a Stop Grant bus cycle when entering the state from the Normal or the Auto HALT Power Down state. When the CPU enters the Stop Grant state from the Stop Clock state or the Stop Clock Snoop state, the CPU does not generate a Stop Grant bus cycle.

5.5.3 Stop Clock State

Stop Clock state is entered from the Stop Grant state by stopping the CLK input (either logic High or logic Low). None of the CPU input signals should change state while the CLK input is stopped. Any transition on an input signal (except INTR) before the CPU has returned to the Stop Grant state may result in unpredictable behavior. If INTR goes active while the CLK input is stopped, and stays active until the CPU issues an interrupt acknowledge bus cycle, it is serviced in the normal manner. System design must ensure the CPU is in the correct state prior to asserting cache invalidation or interrupt signals to the CPU.

5.5.4 Auto Halt Power Down State

A HALT instruction causes the CPU to enter the Auto HALT Power Down state. The CPU issues a normal HALT bus cycle, and only transitions to the Normal state when INTR, NMI, $\overline{\text{SMI}}$, RESET, or SRESET occurs.

The system can generate a $\overline{\text{STPCLK}}$ while the CPU is in the Auto HALT Power Down state. The CPU generates a Stop Grant bus cycle when it enters the Stop Grant state from the HALT state. When the system deasserts the $\overline{\text{STPCLK}}$ interrupt, the CPU returns execution to the HALT state. The CPU generates a new HALT bus cycle when it reenters the HALT state from the Stop Grant state.

5.5.5 Stop Clock Snoop State (Cache Invalidations)

When the CPU is in the Stop Grant state or the Auto HALT Power Down state, the CPU recognizes HOLD, AHOLD, $\overline{\text{BOFF}}$, and $\overline{\text{EADS}}$ for cache invalidation. When the system asserts HOLD, AHOLD, or $\overline{\text{BOFF}}$, the CPU floats the bus accordingly. When the system asserts $\overline{\text{EADS}}$, the CPU transparently enters Stop Clock Snoop state and powers up for one full clock to perform the required cache snoop cycle. If a modified line is snooped, a cache write-back occurs with $\overline{\text{HITM}}$ transitioning active until the completion of the write-back. It then powers down and returns to the previous state. The CPU does not generate a bus cycle when it returns to the previous state.

5.5.6 Cache Flush State

When configured in Write-back mode, the processor recognizes $\overline{\text{FLUSH}}$ for copying back modified cache lines to memory in the Auto Halt Power Down State or Normal State. Upon the completion of the cache flush, the processor returns to its prior state, and regenerates a special bus cycle, if necessary.

6 SRESET FUNCTION

The Am5$_X$86 microprocessor family supports a soft reset function through the SRESET pin. SRESET forces the processor to begin execution in a known state. The processor state after SRESET is the same as after RESET except that the internal caches, CD and NW in CR0, write buffers, SMBASE registers, and floating-point registers retain the values they had prior to SRESET, and cache snooping is allowed. The processor starts execution at physical address FFFFFFF0h. SRESET can be used to help performance for DOS extenders written for the 80286 processor. SRESET provides a method to switch from Protected to Real mode while maintaining the internal caches, CR0, and the FPU state. SRESET may not be used in place of RESET after power-up.

In Write-back mode, once SRESET is sampled active, the SRESET sequence begins on the next instruction boundary (unless $\overline{\text{FLUSH}}$ or RESET occur before that boundary). When started, the SRESET sequence continues to completion and then normal processor execution resumes, independent of the deassertion of SRESET. If a snoop hits a modified line during SRESET, a normal write-back cycle occurs. $\overline{\text{ADS}}$ is asserted to drive the bus cycles even if SRESET is not deasserted.

7 SYSTEM MANAGEMENT MODE

7.1 Overview

The Am5$_X$86 microprocessor supports four modes: Real, Virtual, Protected, and System Management mode (SMM). As an operating mode, SMM has a distinct processor environment, interface, and hardware/software features. SMM lets the system designer add new software-controlled features to the computer products that always operate transparent to the operating system (OS) and software applications. SMM is intended for use only by system firmware, not by applications software or general purpose systems software.

The SMM architectural extension consists of the following elements:

- System Management Interrupt ($\overline{\text{SMI}}$) hardware interface

- Dedicated and secure memory space (SMRAM) for $\overline{\text{SMI}}$ handler code and CPU state (context) data with a status signal for the system to decode access to that memory space, $\overline{\text{SMIACT}}$
- Resume (RSM) instruction, for exiting SMM
- Special features, such as I/O Restart and I/O instruction information, for transparent power management of I/O peripherals, and Auto HALT Restart

7.2 Terminology

The following terms are used throughout the discussion of System Management mode.

- **SMM:** System Management mode. The operating environment that the processor (system) enters when servicing a System Management Interrupt.
- **$\overline{\text{SMI}}$:** System Management Interrupt. This is the trigger mechanism for the SMM interface. When $\overline{\text{SMI}}$ is asserted ($\overline{\text{SMI}}$ pin asserted Low) it causes the processor to invoke SMM. The $\overline{\text{SMI}}$ pin is the only means of entering SMM.
- **$\overline{\text{SMI}}$ handler:** System Management mode handler. This is the code that is executed when the processor is in SMM. Example applications that this code might implement are a power management control or a system control function.
- **RSM:** Resume instruction. This instruction is used by the $\overline{\text{SMI}}$ handler to exit the SMM and return to the interrupted OS or application process.
- **SMRAM:** This is the physical memory dedicated to SMM. The $\overline{\text{SMI}}$ handler code and related data reside in this memory. The processor also uses this memory to store its context before executing the $\overline{\text{SMI}}$ handler. The operating system and applications should not have access to this memory space.
- **SMBASE:** This is a control register that contains the base address that defines the SMRAM space.
- **Context:** This term refers to the processor state. The SMM discussion refers to the context, or processor state, just before the processor invokes SMM. The context normally consists of the CPU registers that fully represent the processor state.
- **Context Switch:** A context switch is the process of either saving or restoring the context. The SMM discussion refers to the context switch as the process of saving/restoring the context while invoking/exiting SMM, respectively.
- **SMSAVE:** A mechanism that saves and restores all internal registers to and from SMRAM.

7.3 System Management Interrupt Processing

The system interrupts the normal program execution and invokes SMM by generating a System Management Interrupt ($\overline{\text{SMI}}$) to the CPU. The CPU services the $\overline{\text{SMI}}$ by executing the following sequence (see Figure 22).

1. The CPU asserts the $\overline{\text{SMIACT}}$ signal, instructing the system to enable the SMRAM.
2. The CPU saves its state (internal register) to SMRAM. It starts at the SMBASE relative address location (see Section 7.3.3), and proceeds downward in a stack-like fashion.
3. The CPU switches to the SMM processor environment (an external pseudo-real mode).
4. The CPU then jumps to the absolute address of SMBASE + 8000h in SMRAM to execute the $\overline{\text{SMI}}$ handler. This $\overline{\text{SMI}}$ handler performs the system management activities.

Note: *If the SMRAM shares the same physical address location with part of the system RAM, it is "overlaid" SMRAM. To preserve cache consistency and correct SMM operation in systems using overlaid SMRAM, the cache must be flushed via the $\overline{\text{FLUSH}}$ pin when entering SMM.*

5. The $\overline{\text{SMI}}$ handler then executes the RSM instruction which restores the CPU's context from SMRAM, deasserts the $\overline{\text{SMIACT}}$ signal, and then returns control to the previously interrupted program execution.

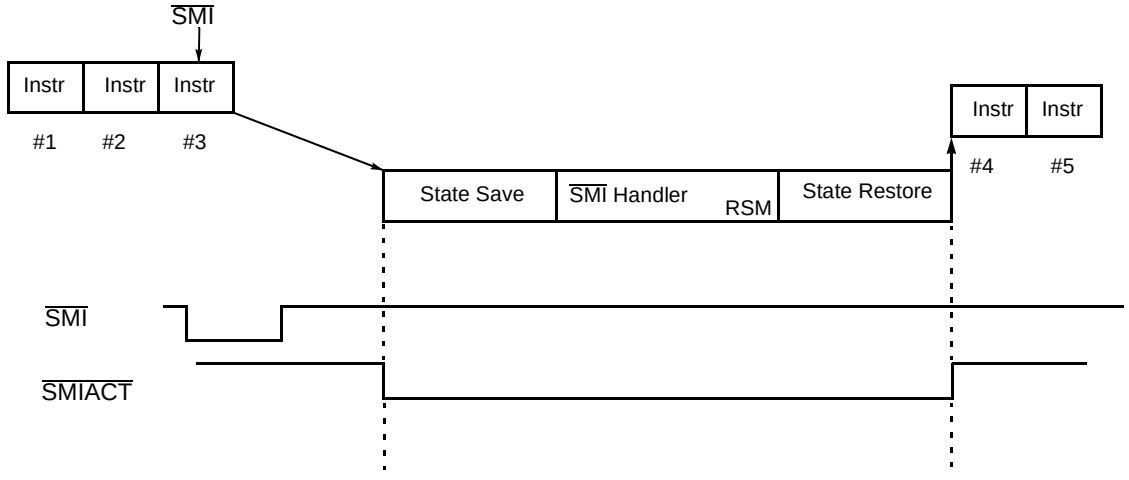


Figure 22. Basic $\overline{\text{SMI}}$ Interrupt Service

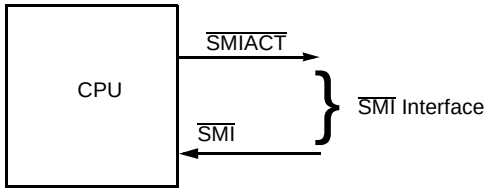


Figure 23. Basic SMI Hardware Interface

For uses such as fast enabling of external I/O devices, the SMSAVE mode permits the restarting of the I/O instructions and the HALT instruction. This is accomplished through I/O Trap Restart and Halt/Auto HALT Restart slots. Only I/O and HALT opcodes are restartable. Attempts to restart any other opcode may result in unpredictable behavior.

The System Management Interrupt hardware interface consists of the SMI request input and the SMIACT output used by the system to decode the SMRAM (see Figure 23).

7.3.1 System Management Interrupt Processing

SMI is a falling-edge-triggered, non-maskable interrupt request signal. SMI is an asynchronous signal, but setup and hold times must be met to guarantee recognition in a specific clock. The SMI input does not have to remain active until the interrupt is actually serviced. The SMI input needs to remain active for only a single clock if the required setup and hold times are met. SMI also works correctly if it is held active for an arbitrary number of clocks (see Figure 24).

The SMI input must be held inactive for at least four clocks after it is asserted to reset the edge-triggered logic. A subsequent SMI may not be recognized if the SMI input is not held inactive for at least four clocks after being asserted. SMI, like NMI, is not affected by the IF bit in the EFLAGS register and is recognized on an instruction boundary. SMI does not break locked bus cycles. SMI has a higher priority than NMI and is not masked during an NMI. After SMI is recognized, the SMI signal is masked internally until the RSM instruction is executed and the interrupt service routine is complete.

Masking SMI prevents recursive calls. If another SMI occurs while SMI is masked, the pending SMI is recognized and executed on the next instruction boundary after the current SMI completes. This instruction boundary occurs before execution of the next instruction in the interrupted application code, resulting in back-to-back SMI handlers. Only one SMI signal can be pending while SMI is masked. The SMI signal is synchronized internally and must be asserted at least three clock cycles prior to asserting the RDY signal to guarantee recognition on a specific instruction boundary. This is important for servicing an I/O trap with an SMI handler.

7.3.2 SMI Active (SMIACT)

SMIACT indicates that the CPU is operating in SMM. The CPU asserts SMIACT in response to an SMI interrupt request on the SMI pin. SMIACT is driven active after the CPU has completed all pending write cycles (including emptying the write buffers), and before the first access to SMRAM when the CPU saves (writes) its state (or context) to SMRAM. SMIACT remains active until the last access to SMRAM when the CPU restores (reads) its state from SMRAM. The SMIACT signal does not float in response to HOLD. The SMIACT signal is used by the system logic to decode SMRAM. The number of clocks required to complete the SMM state save and restore is dependent on system memory performance. The values shown in Figure 25 assume 0 wait-state memory writes (2 clock cycles), 2–1–1–1 burst read cycles, and 0 wait-state non-burst reads (two clock cycles). Additionally, it is assumed that the data read during the SMM state restore sequence is not cacheable. The minimum time required to enter a SMSAVE SMI handler routine for the CPU (from the completion of the interrupted instruction) is given by:

Latency to start of SMI handler = A + B + C = 161 clocks

and the minimum time required to return to the interrupted application (following the final SMM instruction before RSM) is given by:

Latency to continue application = E + F + G = 258 clocks

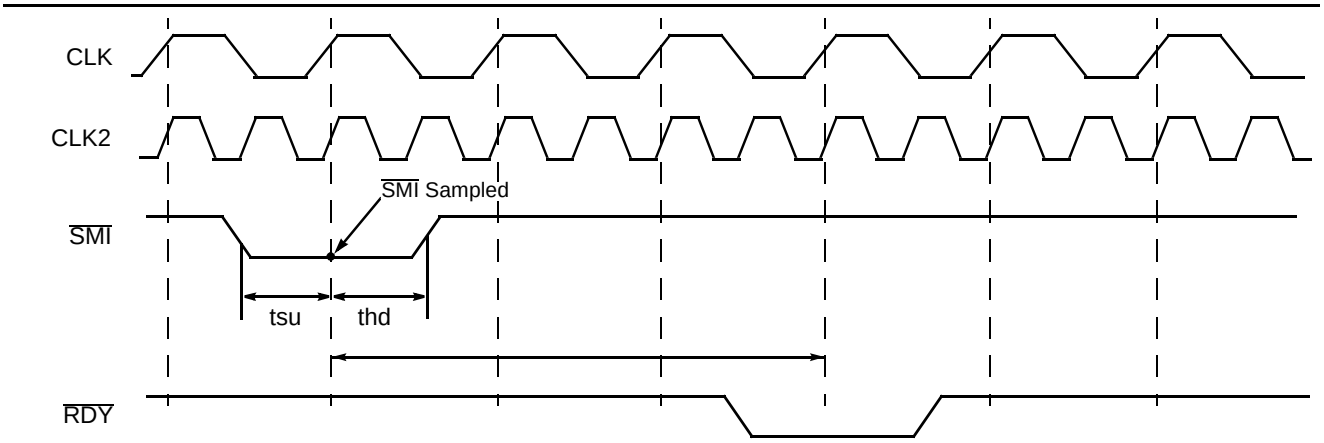
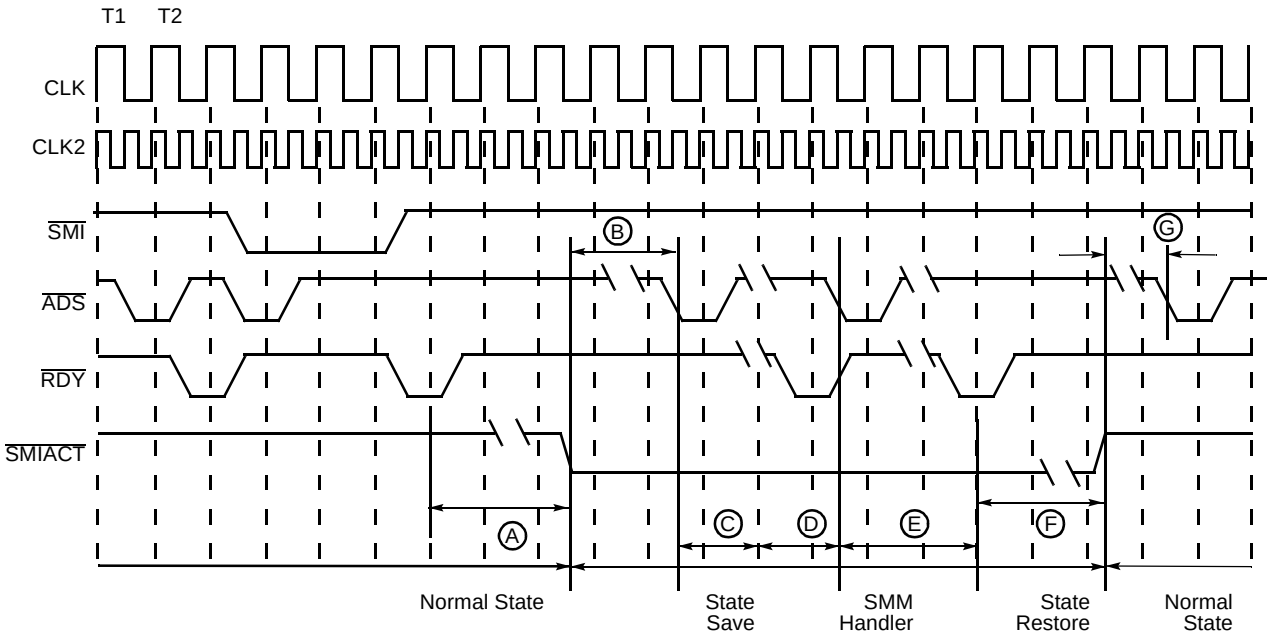


Figure 24. SMI Timing for Servicing an I/O Trap



	Clock-Tripled CPU	Clock-Quadrupled CPU
A: Last RDY from non-SMM transfer to SMIACT assertion	2 CLKs minimum	2 CLKs minimum
B: SMIACT assertion to first ADS for SMM state save	15 CLKs minimum	10 CLKs minimum
C: SMM state save (dependent on memory performance)	100 CLKs	70 CLKs
D: SMI handler	User-determined	User-determined
E: SMM state restore (dependent on memory performance)	180 CLKs	120 CLKs
F: Last RDY from SMM transfer to deassertion of SMIACT	2 CLKs minimum	2 CLKs minimum
G: SMIACT deassertion of first non-SMM ADS	20 CLKs minimum	20 CLKs minimum

Figure 25. SMIACT Timing

7.3.3 SMRAM

The CPU uses the SMRAM space for state save and state restore operations during an SMI. The SMI handler, which also resides in SMRAM, uses the SMRAM space to store code, data, and stacks. In addition, the SMI handler can use the SMRAM for system management information such as the system configuration, configuration of a powered-down device, and system designer-specific information.

Note: *Access to SMRAM is through the CPU internal cache. To ensure cache consistency and correct operation, always assert the FLUSH pin in the same clock as SMI for systems using overlaid SMRAM.*

The CPU asserts SMIACT to indicate to the memory controller that it is operating in System Management mode. The system logic should ensure that only the CPU and SMI handler have access to this area. Alternate bus masters or DMA devices trying to access the SMRAM space when SMIACT is active should be directed to system RAM in the respective area. The system logic is minimally required to decode the physical memory address range 38000h–3FFFFh as SMRAM area. The CPU saves its state to the state save area from 3FFFFh downward to 3FE00h. After saving its state, the CPU jumps to the address location 38000h to begin executing the SMI handler. The system logic can choose to decode a larger area of SMRAM as needed. The size of this SMRAM can be between 32 Kbytes and 4 Gbytes.The system logic should provide a manual method for switching the SMRAM into system memory space when the CPU is not in SMM. This enables initialization of the SMRAM space (i.e., loading SMI handler) before executing the SMI handler during SMM (see Figure 26).

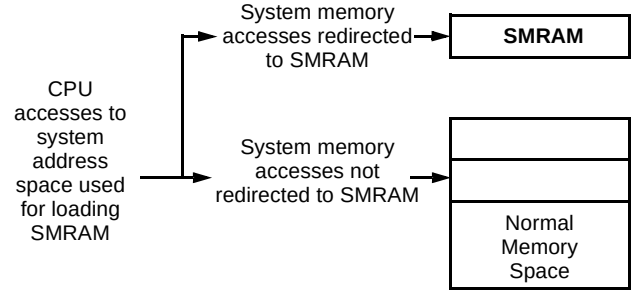


Figure 26. Redirecting System Memory Address to SMRAM

7.3.4 SMRAM State Save Map

When $\overline{SMI}$ is recognized on an instruction boundary, the CPU core first sets the $\overline{SMIACT}$ signal Low, indicating to the system logic that accesses are now being made to the system-defined SMRAM areas. The CPU then writes its state to the state save area in the SMRAM. The state save area starts at SMBASE + [8000h + 7FFFh]. The default CS Base is 30000h; therefore, the default state save area is at 3FFFFh. In this case, the CS Base is also referred to as the SMBASE.

Table 10. SMRAM State Save Map

Register Offset*	Register	Writable?
7FFCh	CRO	No
7FF8h	CR3	No
7FF4h	EFLAGS	Yes
7FF0h	EIP	Yes
7FECh	EDI	Yes
7FE8h	ESI	Yes
7FE4h	EBP	Yes
7FE0h	ESP	Yes
7FDCh	EBX	Yes
7FD8h	EDX	Yes
7FD4h	ECX	Yes
7FD0h	EAX	Yes
7FCCh	DR6	No
7FC8h	DR7	No
7FC4h	TR*	No
7FC0h	LDTR*	No
7FBCh	GS*	No
7FB8h	FS*	No
7FB4h	DS*	No
7FB0h	SS*	No
7FACh	CS*	No
7FA8h	ES*	No
7FA7h–7F98h	Reserved	No
7F94h	IDT Base	No
7F93h–7F8Ch	Reserved	No
7F88h	GDT Base	No
7F87h–7F08h	Reserved	No
7F04h	I/O Trap Word	No
7F02h	Halt Auto Restart	Yes
7F00h	I/O Trap Restart	Yes
7EFCh	SMM Revision Identifier	Yes
7EF8h	State Dump Base	Yes
7EF7h–7E00h	Reserved	No

Note:
**Upper 2 bytes are not modified.*

If the SMBASE relocation feature is enabled, the SMRAM addresses can change. The following formula is used to determine the relocated addresses where the context is saved: SMBASE + [8000h + Register Offset], where the default initial SMBASE is 30000h and the Register Offset is listed in Table 10. Reserved spaces are for new registers in future CPUs. Some registers in the SMRAM state save area may be read and changed by the $\overline{SMI}$ handler, with the changed values restored to the processor register by the RSM instruction. Some register images are read-only, and must not be modified. (Modifying these registers results in unpredictable behavior.) The values stored in the "reserved" areas may change in future CPUs. An $\overline{SMI}$ handler should not rely on values stored in a reserved area.

The following registers are written out during SMSAVE mode to the RESERVED memory locations (7FA7h–7F98h, 7F93h–7F8Ch, and 7F87h–7F08h), but are not visible to the system software programmer:

- DR3–DR0
- CR2
- CS, DS, ES, FS, GS, and SS hidden descriptor registers
- EIP_Previous
- GDT Attributes and Limits
- IDT Attributes and Limits
- LDT Attributes, Base, and Limits
- TSS Attributes, Base, and Limits

If an $\overline{SMI}$ request is issued to power down the CPU, the values of all reserved locations in the SMM state save area must be saved to non-volatile memory.

The following registers are not automatically saved and restored by $\overline{SMI}$ and RSM:

- TR7–TR3
- FPU registers:
  - — STn
  - — FCS
  - — FSW
  - — Tag Word
  - — FP instruction pointer
  - — FP opcode
  - — Operand pointer

Note: *You can save the FPU state by using an FSAVE or FNSAVE instruction.*

For all $\overline{SMI}$ requests except for power down suspend/resume, these registers do not have to be saved because their contents will not change. During a power down suspend/resume, however, a resume reset clears these registers back to their default values. In this case, the suspend $\overline{SMI}$ handler should read these registers directly to save them and restore them during the power up resume. Anytime the $\overline{SMI}$ handler changes these registers in the CPU, it must also save and restore them.

7.4 Entering System Management Mode

SMM is one of the major operating modes, along with Protected mode, Real mode, and Virtual mode. Figure 27 shows how the processor can enter SMM from any of the three modes and then return.

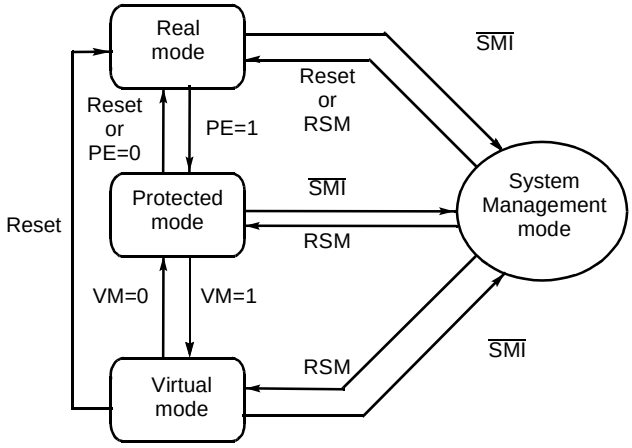


Figure 27. Transition to and from SMM

The external signal $\overline{SMI}$ causes the processor to switch to SMM. The RSM instruction exits SMM. SMM is transparent to applications, programs, and operating systems for the following reasons:

- The only way to enter SMM is via a type of nonmaskable interrupt triggered by an external signal
- The processor begins executing SMM code from a separate address space, referred to earlier as system management RAM (SMRAM)
- Upon entry into SMM, the processor saves the register state of the interrupted program (depending on the save mode) in a part of SMRAM called the SMM context save space
- All interrupts normally handled by the operating system or applications are disabled upon SMM entry
- A special instruction, RSM, restores processor registers from the SMM context save space and returns control to the interrupted program

Similar to Real mode, SMM has no privilege levels or address mapping. SMM programs can execute all I/O and other system instructions and can address up to 4 Gbytes of memory.

7.5 Exiting System Management Mode

The RSM instruction (opcode 0F AAh) leaves SMM and returns control to the interrupted program. The RSM instruction can be executed only in SMM. An attempt to execute the RSM instruction outside of SMM generates an invalid opcode exception. When the RSM instruction is executed and the processor detects invalid state information during the reloading of the save state, the processor enters the shutdown state. This occurs in the following situations:

- The value in the State Dump base field is not a 32-Kbyte aligned address
- A combination of bits in CR0 is illegal: (PG=1 and PE=0) or (NW=1 and CD=0)

In Shutdown mode, the processor stops executing instructions until an NMI interrupt is received or reset initialization is invoked. The processor generates a shutdown bus cycle.

Three SMM features can be enabled by writing to control slots in the SMRAM state save area:

1. **Auto HALT Restart**. It is possible for the $\overline{SMI}$ request to interrupt the HALT state. The $\overline{SMI}$ handler can tell the RSM instruction to return control to the HALT instruction or to return control to the instruction following the HALT instruction by appropriately setting the Auto HALT Restart slot. The default operation is to restart the HALT instruction.
2. **I/O Trap Restart**. If the $\overline{SMI}$ was generated on an I/O access to a powered-down device, the $\overline{SMI}$ handler can instruct the RSM instruction to re-execute that I/O instruction by setting the I/O Trap Restart slot.
3. **SMBASE Relocation**. The system can relocate the SMRAM by setting the SMBASE Relocation slot in the state save area. The RSM instruction sets SMBASE in the processor based on the value in the SMBASE relocation slot. The SMBASE must be aligned on 32-Kbyte boundaries.

A RESET also causes execution to exit from SMM.

7.6 Processor Environment

When an $\overline{SMI}$ signal is recognized on an instruction execution boundary, the processor waits for all stores to complete, including emptying the write buffers. The final write cycle is complete when the system returns $\overline{RDY}$ or $\overline{BRDY}$. The processor then drives $\overline{SMIACT}$ active, saves its register state to SMRAM space, and begins to execute the $\overline{SMI}$ handler.

$\overline{SMI}$ has greater priority than debug exceptions and external interrupts. This means that if more than one of these conditions occur at an instruction boundary, only the $\overline{SMI}$ processing occurs. Subsequent $\overline{SMI}$ requests are not acknowledged while the processor is in SMM. The first $\overline{SMI}$ request that occurs while the processor is in SMM is latched, and serviced when the processor exits SMM with the RSM instruction. Only one $\overline{SMI}$ signal is latched by the CPU while it is in SMM. When the CPU invokes SMM, the CPU core registers are initialized as indicated in Table 11.

Table 11. SMM Initial CPU Core Register Settings

Register	SMM Initial State
General Purpose Registers	Unmodified
EFLAGS	0000 0002h
CR0	Bits 0, 2, 3, and 31 cleared (PE, EM, TS, and PG); rest unmodified
DR6	Unpredictable state
DR7	0000 0400h
GDTR, LDTR, IDTR, TSSR	Unmodified
EIP	0000 8000h

***Note**:*
Interrupts from INT and NMI are disabled on SMM entry.

The following is a summary of the key features in the SMM environment:

- Real mode style address calculation
- 4-Gbyte limit checking
- IF flag is cleared
- NMI is disabled
- TF flag in EFLAGS is cleared; single step traps are disabled
- DR7 is cleared; debug traps are disabled
- The RSM instruction no longer generates an invalid opcode error
- Default 16-bit opcode, register, and stack use
- All bus arbitration (HOLD, AHOLD, $\overline{\text{BOFF}}$) inputs, and bus sizing ($\overline{\text{BS8}}$, $\overline{\text{BS16}}$) inputs operate normally while the CPU is in SMM

7.7 Executing System Management Mode Handler

The processor begins execution of the $\overline{\text{SMI}}$ handler at offset 8000h in the CS segment. The CS Base is initially 30000h, as shown in Table 12.

Table 12. Segment Register Initial States

Segment Register	Selector	Base	Attributes	Limit[1]
CS[2]	3000h	30000h	16-bit, expand up	4 Gbytes
DS	0000h	00000000h	16-bit, expand up	4 Gbytes
ES	0000h	00000000h	16-bit, expand up	4 Gbytes
FS	0000h	00000000h	16-bit, expand up	4 Gbytes
GS	0000h	00000000h	16-bit, expand up	4 Gbytes
SS	0000h	00000000h	16-bit, expand up	4 Gbytes

***Notes**:*

1. *The segment limit check is 4 Gbytes instead of the usual 64 Kbytes.*
2. *The Selector value for CS remains at 3000h even if the SMBASE is changed.*

The CS Base can be changed using the SMM Base relocation feature. When the $\overline{\text{SMI}}$ handler is invoked, the CPU's PE and PG bits in CR0 are reset to 0. The processor is in an environment similar to Real mode, but without the 64-Kbyte limit checking. However, the default operand size and the default address size are set to 16 bits. The EM bit is cleared so that no exceptions are generated. (If the SMM was entered from Protected mode, the Real mode interrupt and exception support is not available.) The $\overline{\text{SMI}}$ handler should not use floating-point unit instructions until the FPU is properly detected (within the $\overline{\text{SMI}}$ handler) and the exception support is initialized.

Because the segment bases (other than CS) are cleared to 0 and the segment limits are set to 4 Gbytes, the address space may be treated as a single flat 4-Gbyte linear space that is unsegmented. The CPU is still in Real mode and when a segment selector is loaded with a 16-bit value, that value is then shifted left by 4 bits and loaded into the segment base cache.

In SMM, the CPU can access or jump anywhere within the 4-Gbyte logical address space. The CPU can also indirectly access or perform a near jump anywhere within the 4-Gbyte logical address space.

7.7.1 Exceptions and Interrupts with System Management Mode

When the CPU enters SMM, it disables INTR interrupts, debug, and single step traps by clearing the EFLAGS, DR6, and DR7 registers. This prevents a debug application from accidentally breaking into an $\overline{SMI}$ handler. This is necessary because the $\overline{SMI}$ handler operates from a distinct address space (SMRAM) and the debug trap does not represent the normal system memory space.

For an $\overline{SMI}$ handler to use the debug trap feature of the processor to debug $\overline{SMI}$ handler code, it must first ensure that an SMM-compliant debug handler is available. The $\overline{SMI}$ handler must also ensure DR3–DR0 is saved to be restored later. The debug registers DR3–DR0 and DR7 must then be initialized with the appropriate values.

For the processor to use the single step feature of the processor, it must ensure that an SMM-compliant single step handler is available and then set the trap flag in the EFLAGS register. If the system design requires the processor to respond to hardware INTR requests while in SMM, it must ensure that an SMM-compliant interrupt handler is available, and then set the interrupt flag in the EFLAGS register (using the STI instruction). Software interrupts are not blocked on entry to SMM, and the system software designer must provide an SMM-compliant interrupt handler before attempting to execute any software interrupt instructions. Note that in SMM mode, the interrupt vector table has the same properties and location as the Real mode vector table.

NMI interrupts are blocked on entry to the $\overline{SMI}$ handler. If an NMI request occurs during the $\overline{SMI}$ handler, it is latched and serviced after the processor exits SMM. Only one NMI request is latched during the $\overline{SMI}$ handler. If an NMI request is pending when the processor executes the RSM instruction, the NMI is serviced before the next instruction of the interrupted code sequence.

Although NMI requests are blocked when the CPU enters SMM, they may be enabled through software by executing an IRET instruction. If the $\overline{SMI}$ handler requires the use of NMI interrupts, it should invoke a dummy interrupt service routine to execute an IRET instruction. When an IRET instruction is executed, NMI interrupt requests are serviced in the same Real mode manner in which they are handled outside of SMM.

7.7.2 SMM Revisions Identifier

The 32-bit SMM Revision Identifier specifies the version of SMM and the extensions that are available on the processor. The fields of the SMM Revision Identifiers and bit definitions are shown in Table 13 and Table 14. Bit 17 or 16 indicates whether the feature is supported (1=supported, 0=not supported). The processor always reads the SMM Revision Identifier at the time of a restore. The I/O Trap Extension and SMM Base Relocation bits are fixed. The processor writes these bits out at the time it performs a save state.

Note: *Changing the state of the reserved bits may result in unpredictable processor behavior.*

Table 13. SMM Revision Identifier

31–18	17	16	15–0
Reserved	**SMM Base Relocation**	**I/O Trap Extension**	**SMM Revision Level**
00000000000000	1	1	0000h

Table 14. SMM Revision Identifier Bit Definitions

Bit Name	Description	Default State	State at SMM Entry	State at SMM Exit	Notes
SMM Base Relocation	1=SMM Base Relocation Available 0=SMM Base Relocation Unavailable	1	1 0	1 0	No Change in State No Change in State
I/O Trap Extension	1=I/O Trapping Available 0=I/O Trapping Unavailable	1	1 0	1 0	No Change in State No Change in State

7.7.3 Auto HALT Restart

The Auto HALT Restart slot at register offset (word location) 7F02h in SMRAM indicates to the $\overline{SMI}$ handler that the $\overline{SMI}$ interrupted the CPU during a HALT state; bit 0 of slot 7F02h is set to 1 if the previous instruction was a HALT (see Figure 28). If the $\overline{SMI}$ did not interrupt the CPU in a HALT state, then the $\overline{SMI}$ microcode sets bit 0 of the Auto HALT Restart slot to 0. If the previous instruction was a HALT, the $\overline{SMI}$ handler can choose to either set or reset bit 0. If this bit is set to 1, the RSM microcode execution forces the processor to re-enter the HALT state. If this bit is set to 0 when the RSM instruction is executed, the processor continues execution with the instruction just after the interrupted HALT instruction. If the HALT instruction is restarted, the CPU will generate a memory access to fetch the HALT instruction (if it is not in the internal cache), and execute a HALT bus cycle.

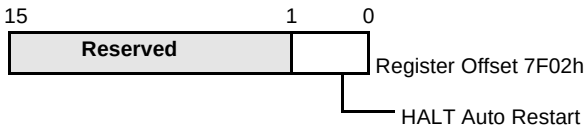


Figure 28. Auto HALT Restart Register Offset

Table 15 shows the possible restart configurations. If the interrupted instruction was not a HALT instruction (bit 0 is set to 0 in the Auto HALT Restart slot upon SMM entry), setting bit 0 to 1 will cause unpredictable behavior when the RSM instruction is executed.

Table 15. HALT Auto Restart Configuration

Value at Entry	Value at Exit	Processor Action on Exit
0	0	Return to next instruction in interrupted program
0	1	Unpredictable
1	0	Returns to instruction after HALT
1	1	Returns to interrupted HALT instruction

7.7.4 I/O Trap Restart

The I/O instruction restart slot (register offset 7F00h in SMRAM) gives the $\overline{SMI}$ handler the option of causing the RSM instruction to automatically re-execute the interrupted I/O instruction (see Figure 29).

.

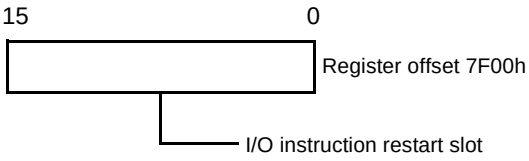


Figure 29. I/O Instruction Restart Register Offset

When the RSM instruction is executed — if the I/O instruction restart slot contains the value 0FFh — the CPU automatically re-executes the I/O instruction that the $\overline{SMI}$ signal trapped. If the I/O instruction restart slot contains the value 00h when the RSM instruction is executed, then the CPU does not re-execute the I/O instruction. The CPU automatically initializes the I/O instruction restart slot to 00h during SMM entry. The I/O instruction restart slot should be written only when the processor has generated an $\overline{SMI}$ on an I/O instruction boundary. Processor operation is unpredictable when the I/O instruction restart slot is set when the processor is servicing an $\overline{SMI}$ that originated on a non-I/O instruction boundary.

If the system executes back-to-back $\overline{SMI}$ requests, the second $\overline{SMI}$ handler must not set the I/O instruction restart slot. The second back-to-back $\overline{SMI}$ signal will not have the I/O Trap Word set.

7.7.5 I/O Trap Word

The I/O Trap Word contains the address of the I/O access that forced the external chipset to assert $\overline{SMI}$, whether it was a read or write access, and whether the instruction that caused the access to the I/O address was a valid I/O instruction. Table 16 shows the layout.

Table 16. I/O Trap Word Configuration

31–16	15–2	1	0
I/O Address	Reserved	Valid I/O Instruction	R/$\overline{W}$

Bits 31–16 contain the I/O address that was being accessed at the time $\overline{SMI}$ became active. Bits 15–2 are reserved.

If the instruction that caused the I/O trap to occur was a valid I/O instruction (IN, OUT, INS, OUTS, REP INS, or REP OUTS), the Valid I/O Instruction bit is set. If it was not a valid I/O instruction, the bit is saved as a 0. For REP instructions, the external chip set should return a valid $\overline{SMI}$ within the first access.

Bit 0 indicates whether the opcode that was accessing the I/O location was performing either a read (1) or a write (0) operation as indicated by the R/$\overline{W}$ bit.

If an $\overline{SMI}$ occurs and it does not trap an I/O instruction, the contents of the I/O address and R/$\overline{W}$ bit are unpredictable and should not be used.

7.7.6 SMM Base Relocation

The Am5$_X$86 CPU family provides a new control register, SMBASE. The SMRAM address space can be modified by changing the SMBASE register before exiting an $\overline{SMI}$ handler routine. SMBASE can be changed to any 32K-aligned value. (Values that are not 32K-aligned cause the CPU to enter the shutdown state when executing the RSM instruction.) SMBASE is set to the default value of 30000h on RESET. If SMBASE is changed by an $\overline{SMI}$ handler, all subsequent $\overline{SMI}$ requests initiate a state save at the new SMBASE.

The SMBASE slot in the SMM state save area indicates and changes the $\overline{SMI}$ jump vector location and SMRAM save area. When bit 17 of the SMM Revision Identifier is set, then this feature exists and the SMRAM base and consequently, the jump vector, are as indicated by the SMM Base slot (see Figure 30). During the execution of the RSM instruction, the CPU reads this slot and initializes the CPU to use the new SMBASE during the next $\overline{SMI}$. During an $\overline{SMI}$, the CPU does its context save to the new SMRAM area pointed to by the SMBASE, stores the current SMBASE in the SMM Base slot (offset 7EF8h), and then starts execution of the new jump vector based on the current SMBASE (see Figure 31).

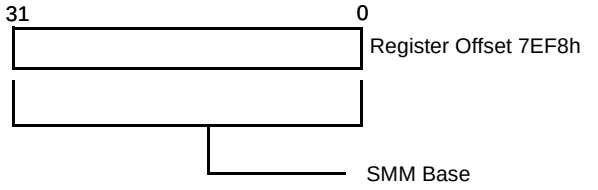


Figure 30. SMM Base Slot Offset

The SMBASE must be a 32-Kbyte aligned, 32-bit integer that indicates a base address for the SMRAM context save area and the $\overline{SMI}$ jump vector. For example, when the processor first powers up, the range for the SMRAM area is from 38000h–3FFFFh. The default value for SMBASE is 30000h.

As illustrated in Figure 31, the starting address of the jump vector is calculated by:

SMBASE + 8000h

The starting address for the SMRAM state save area is calculated by:

SMBASE + [8000h + 7FFFh]

When this feature is enabled, the SMRAM register map is addressed according to the above formula.

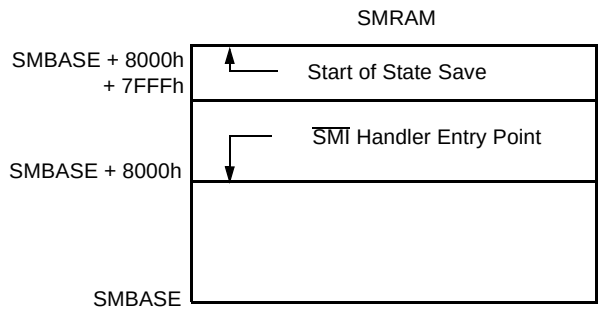


Figure 31. SRAM Usage

To change the SMRAM base address and $\overline{SMI}$ jump vector location, $\overline{SMI}$ handler modifies the SMBASE slot. Upon executing an RSM instruction, the processor reads the SMBASE slot and stores it internally. Upon recognition of the next $\overline{SMI}$ request, the processor uses the new SMBASE slot for the SMRAM dump and $\overline{SMI}$ jump vector. If the modified SMBASE slot does not contain a 32-Kbyte aligned value, the RSM microcode causes the CPU to enter the shutdown state.

7.8 SMM System Design Considerations

7.8.1 SMRAM Interface

The hardware designed to control the SMRAM space must follow these guidelines:

- Initialize SMRAM space during system boot up. Initialization must occur before the first $\overline{SMI}$ occurs. Initialization of SMRAM space must include installation of an $\overline{SMI}$ handler and may include installation of related data structures necessary for particular SMM applications. The memory controller interfacing SMRAM should provide a means for the initialization code to open the SMRAM space manually.
- The memory controller must decode a minimum initial SMRAM address space of 38000h–3FFFFh.
- Alternate bus masters (such as DMA controllers) must not be able to access SMRAM space. The system should allow only the CPU, either through $\overline{SMI}$ or during initialization, to access SMRAM.
- To implement a 0-V suspend function, the system must have access to all normal system memory from within an $\overline{SMI}$ handler routine. If the SMRAM overlays normal system memory (see Figure 32), there must be a method to access overlaid system memory independently.

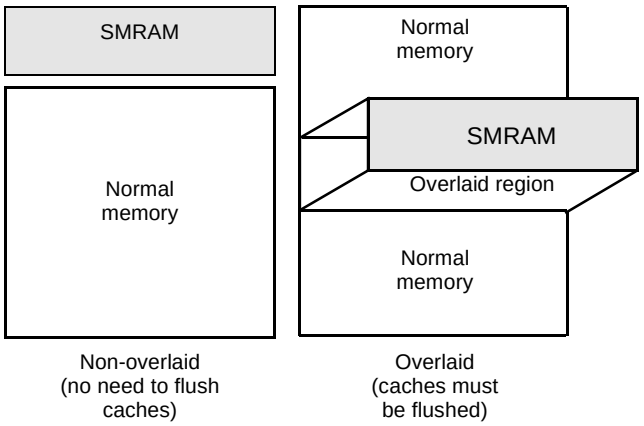


Figure 32. SMRAM Location

The recommended configuration is to use a separate (non-overlaid) physical address for SMRAM. This non-overlaid scheme prevents the CPU from improperly accessing the SMRAM or system RAM directly or through the cache. Figure 33 shows the relative SMM timing for non-overlaid SMRAM for systems configured in Write-through mode. For systems configured in Write-back mode, WB/$\overline{WT}$ must be driven Low (as shown in Figure 34) to force caching during SMM to be write-through. Alternately, caching can be disabled during SMM by deasserting $\overline{KEN}$ with $\overline{SMI}$ (as shown in Figure 35).

When the default SMRAM location is used, however, SMRAM is overlaid with system main memory (at 38000h–3FFFFh). For simplicity, system designers may want to use this default address, or they may select another overlaid address range. However, in this case the system control circuitry must use $\overline{SMIACT}$ to distinguish between SMRAM and main system memory, and must restrict SMRAM space access to the CPU only. To maintain cache coherency and to ensure proper system operation in systems configured in Write-through mode, the system must flush both the CPU internal cache and any second level caches in response to $\overline{SMIACT}$ going Low. A system that uses cache during SMM must flush the cache a second time in response to $\overline{SMIACT}$ going High (see Figure 36). If $\overline{KEN}$ is driven High when $\overline{FLUSH}$ is asserted, the cache is disabled and a second flush is not required (see Figure 37). If the system is configured in Write-back mode, the cache must be flushed when $\overline{SMI}$ is asserted and then disabled (see Figure 38).

7.8.2 Cache Flushes

The CPU does not unconditionally flush its cache before entering SMM. Therefore, the designer must ensure that, for systems using overlaid SMRAM, the cache is flushed upon SMM entry and SMM exit if caching is enabled.

Note: *A cache flush in a system configured in Write-back mode requires a minimum of 4100 internal clocks to test the cache for modified data, whether invoked by the $\overline{FLUSH}$ pin input or the WBINVD instruction, and therefore invokes a performance penalty. There is no flush penalty for systems configured in Write-through mode.*

If the flush at SMM entry is not done, the first SMM read could hit in a cache that contains normal memory space code/data instead of the required $\overline{SMI}$ handler, and the handler could not be executed. If the cache is not disabled and is not flushed at SMM exit, the normal read cycles after SMM may hit in a cache that may contain SMM code/data instead of the normal system memory contents.

In Write-through mode, assert the $\overline{FLUSH}$ signal in response to the assertion of $\overline{SMIACT}$ at SMM entry, and, if required because the cache is enabled, assert $\overline{FLUSH}$ again in response to the deassertion of $\overline{SMIACT}$ at SMM exit (see Figure 36 and Figure 37). For systems configured in Write-back mode, assert $\overline{FLUSH}$ with $\overline{SMI}$ (see Figure 38).

Reloading the state registers at the end of SMM restores cache functionality to its pre-SMM state.

7.8.3 $\overline{A20M}$ Pin

Systems based on the MS-DOS operating system contain a feature that enables the CPU address bit A20 to be forced to 0. This limits physical memory to a maximum of 1 Mbyte, and is provided to ensure compatibility with those programs that relied on the physical address wraparound functionality of the original IBM PC. The $\overline{A20M}$ pin on Am5$_X$86 CPUs provides this function. When $\overline{A20M}$ is active, all external bus cycles drive A20 Low, and all internal cache accesses are performed with A20 Low.

The $\overline{A20M}$ pin is recognized while the CPU is in SMM. The functionality of the $\overline{A20M}$ input must be recognized in two instances:

1. If the $\overline{SMI}$ handler needs to access system memory space above 1 Mbyte (for example, when saving memory to disk for a 0-V suspend), the $\overline{A20M}$ pin must be deasserted before the memory above 1 Mbyte is addressed.
2. If SMRAM has been relocated to address space above 1 Mbyte, and $\overline{A20M}$ is active upon entering SMM, the CPU attempts to access SMRAM at the relocated address, but with A20 Low. This could cause the system to crash, because there would be no valid SMM interrupt handler at the accessed location.

To account for these two situations, the system designer must ensure that $\overline{A20M}$ is deasserted on entry to SMM. $\overline{A20M}$ must be driven inactive before the first cycle of the SMM state save, and must be returned to its original level after the last cycle of the SMM state restore. This can be done by blocking the assertion of $\overline{A20M}$ when $\overline{SMIACT}$ is active.

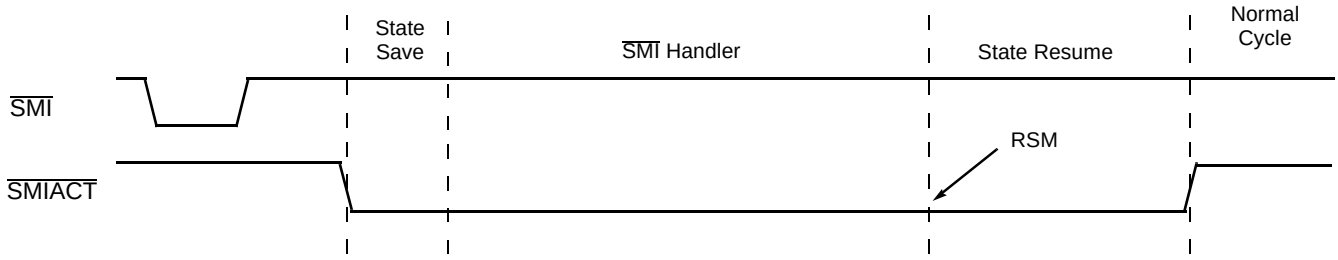
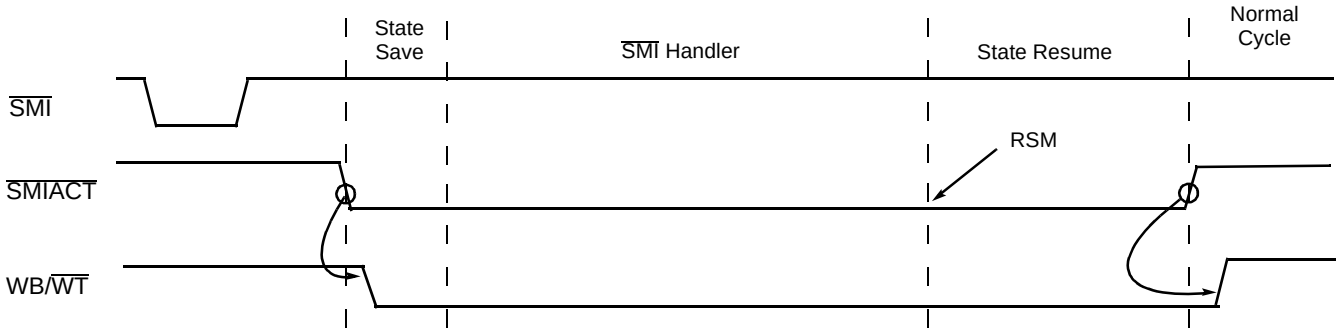


Figure 33. SMM Timing in Systems Using Non-Overlaid Memory Space and Write-Through Mode with Caching Enabled During SMM



Note:
For proper operation of systems configured in Write-back mode when caching during SMM is allowed, force WB/WT Low to force all caching to be write-through during SMM.

Figure 34. SMM Timing in Systems Using Non-Overlaid Memory Spaces and Write-Back Mode with Caching Enabled During SMM

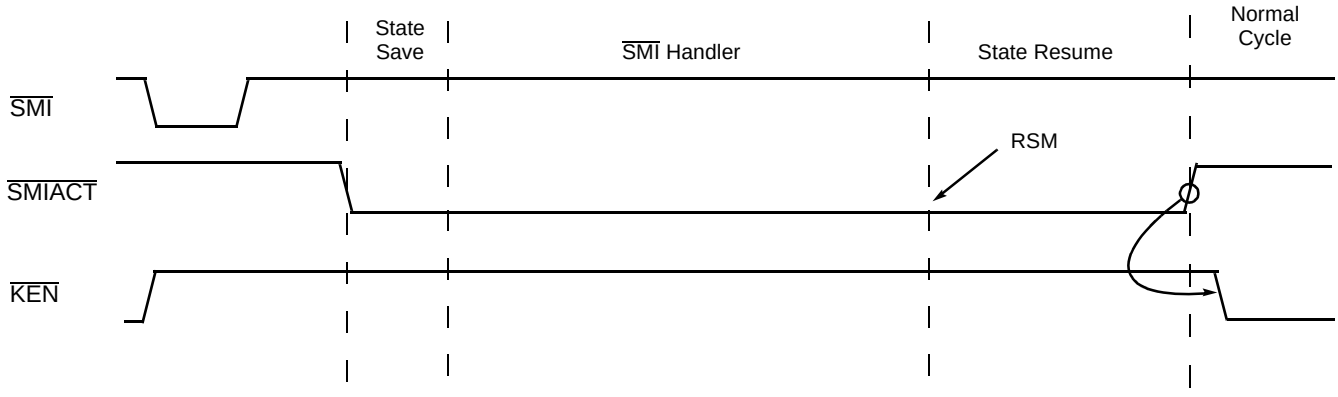


Figure 35. SMM Timing in Systems Using Non-Overlaid Memory Spaces and Write-Back Mode with Caching Disabled During SMM

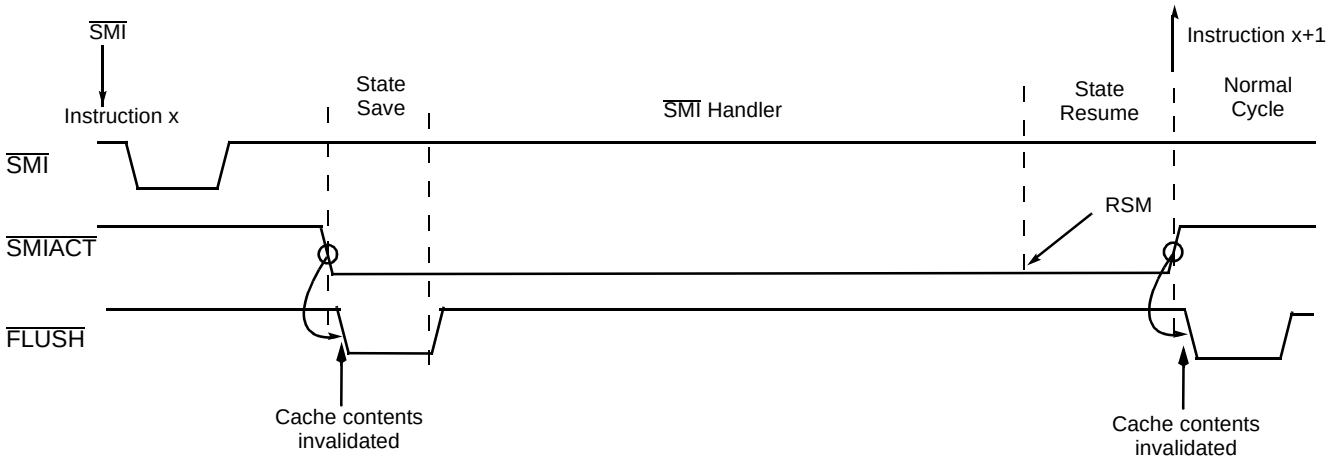


Figure 36. SMM Timing in Systems Using Overlaid Memory Space and Write-Through Mode with Caching Enabled During SMM

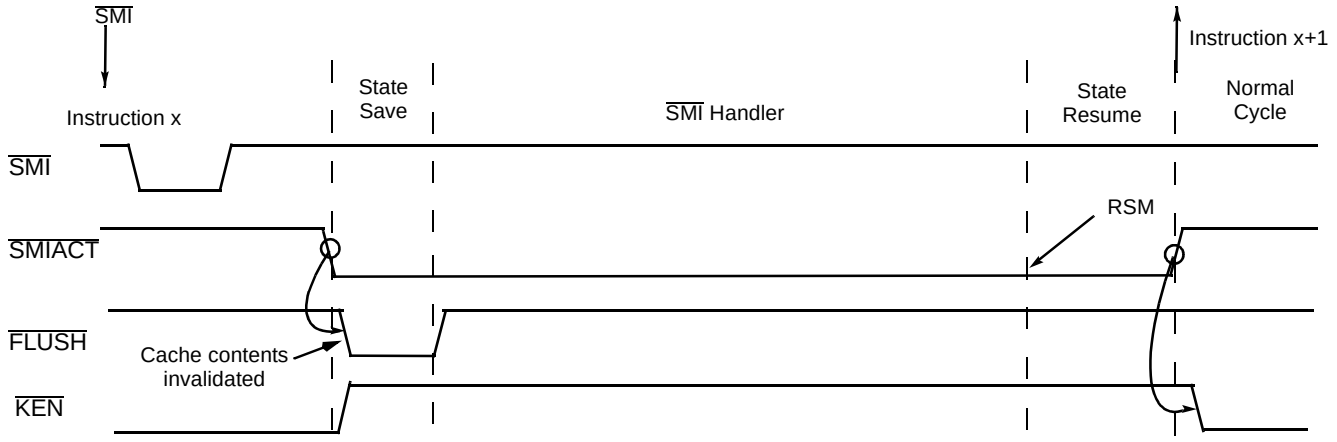


Figure 37. SMM Timing in Systems Using Overlaid Memory Spaces and Write-Through Mode with Caching Disabled During SMM

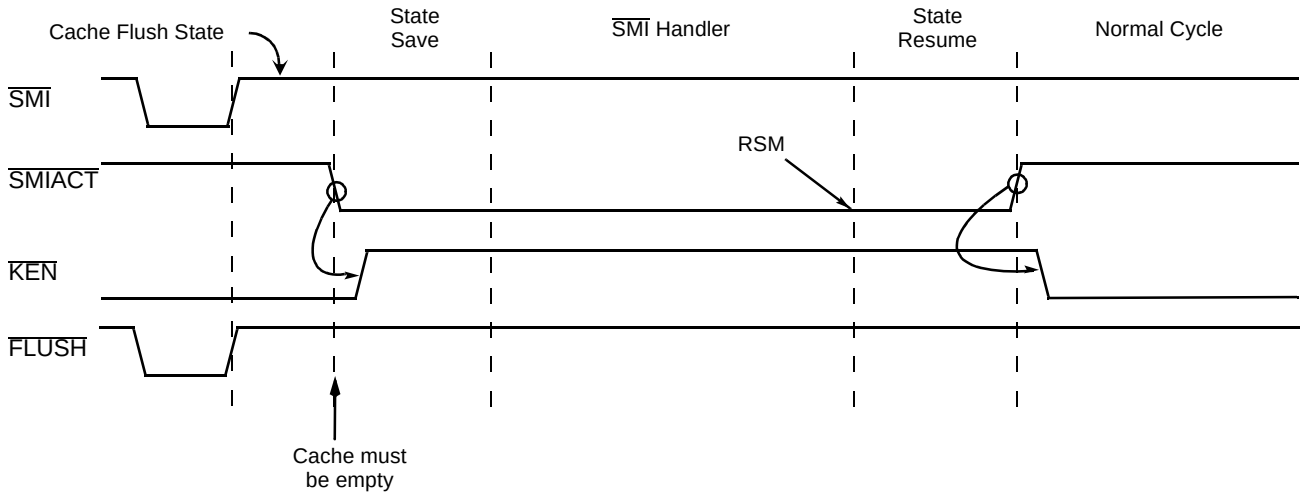


Figure 38. SMM Timing in Systems Using Overlaid Memory Spaces and Configured in Write-Back Mode

7.8.4 CPU Reset During SMM

The system designer should take into account the following restrictions while implementing the CPU Reset logic:

1. When running software written for the 80286 CPU, a CPU RESET switches the CPU from Protected mode to Real mode. RESET and SRESET have a higher priority than $\overline{SMI}$. When the CPU is in SMM, the SRESET to the CPU during SMM should be blocked until the CPU exits SMM. SRESET must be blocked beginning from the time when $\overline{SMI}$ is driven active. Care should be taken not to block the global system RESET, which may be necessary to recover from a system crash.
2. During execution of the RSM instruction to exit SMM, there is a small time window between the deassertion of $\overline{SMIACT}$ and the completion of the RSM microcode. If a Protected mode to Real mode SRESET is asserted during this window, it is possible that the SMRAM space will be violated. The system designer must guarantee that SRESET is blocked until at least 20 CPU clock cycles after $\overline{SMIACT}$ has been driven inactive or until the start of a bus cycle.
3. Any request for a CPU RESET for the purpose of switching the CPU from Protected mode to Real mode must be acknowledged after the CPU has exited SMM. To maintain software transparency, the system logic must latch any SRESET signals that are blocked during SMM.

For these reasons, the SRESET signal should be used for any soft resets, and the RESET signal should be used for all hard resets.

7.8.5 SMM and Second Level Write Buffers

Before the processor enters SMM, it empties its internal write buffers. This is to ensure that the data in the write buffers is written to normal memory space, not SMM space. When the CPU is ready to begin writing an SMM state save to SMRAM, it asserts $\overline{SMIACT}$. $\overline{SMIACT}$ may be driven active by the CPU before the system memory controller has had an opportunity to empty the second level write buffers.

To prevent the data from these second level write buffers from being written to the wrong location, the system memory controller needs to direct the memory write cycles to either SMM space or normal memory space. This can be accomplished by saving the status of $\overline{SMIACT}$ with the address for each word in the write buffers.

7.8.6 Nested $\overline{SMI}$ and I/O Restart

Special care must be taken when executing an $\overline{SMI}$ handler for the purpose of restarting an I/O instruction. When the CPU executes a Resume (RSM) instruction with the I/O restart slot set, the restored EIP is modified to point to the instruction immediately preceding the $\overline{SMI}$ request, so that the I/O instruction can be re-executed. If a new $\overline{SMI}$ request is received while the CPU is executing an $\overline{SMI}$ handler, the CPU services this $\overline{SMI}$ request before restarting the original I/O instruction. If the I/O restart slot is set when the CPU executes the RSM instruction for the second $\overline{SMI}$ handler, the RSM microcode decrements the restored EIP again. EIP then points to an address different from the originally interrupted instruction, and the CPU begins execution at an incorrect entry point. To prevent this from occurring, the $\overline{SMI}$ handler routine must not set the I/O restart slot during the second of two consecutive $\overline{SMI}$ handlers.

7.9 SMM Software Considerations

7.9.1 SMM Code Considerations

The default operand size and the default address size are 16 bits; however, operand-size override and address-size override prefixes can be used as needed to directly access data anywhere within the 4-Gbyte logical address space.

With operand-size override prefixes, the $\overline{SMI}$ handler can use jumps, calls, and returns to transfer a control to any location within the 4-Gbyte space. Note, however, the following restrictions:

1. Any control transfer that does not have an operand-size override prefix truncates EIP to 16 Low-order bits.
2. Due to the Real mode style of base-address formation, a long jump or call cannot transfer control segment with a base address of more than 20 bits (1 Mbyte).

7.9.2 Exception Handling

Upon entry into SMM, external interrupts that require handlers are disabled (the IF in EFLAGS is cleared). This is necessary because, while the processor is in SMM, it is running in a separate memory space. Consequently, the vectors stored in the interrupt descriptor table (IDT) for the prior mode are not applicable. Before allowing exception handling (or software interrupts), the SMM program must initialize new interrupt and exception vectors. The interrupt vector table for SMM has the same format as for Real mode. Until the interrupt vector table is correctly initialized, the $\overline{SMI}$ handler must not generate an exception (or software interrupt). Even though hardware interrupts are disabled, exceptions and software interrupts can still occur. Only a correctly written $\overline{SMI}$ handler can prevent internal exceptions. When new exception vectors are initialized, internal exceptions can be serviced. Restrictions are as follows:

1. Due to the Real mode style of base address formation, an interrupt or exception cannot transfer control to a segment with a base address of more than 20 bits.
2. An interrupt or exception cannot transfer control to a segment offset of more than 16 bits.
3. If exceptions or interrupts are allowed to occur, only the Low order 16 bits of the return address are

pushed onto the stack. If the offset of the interrupted procedure is greater than 64 Kbytes, it is not possible for the interrupt/exception handler to return control to that procedure. (One work-around is to perform software adjustment of the return address on the stack.)

4. The SMBASE Relocation feature affects the way the CPU returns from an interrupt or exception during an $\overline{SMI}$ handler.

Note: *The execution of an IRET instruction enables Non-Maskable Interrupt (NMI) processing.*

7.9.3 Halt during SMM

HALT should not be executed during SMM, unless interrupts have been enabled. Interrupts are disabled on entry to SMM. INTR and NMI are the only events that take the CPU out of HALT within SMM.

7.9.4 Relocating SMRAM to an Address above 1 Mbyte

Within SMM (or Real mode), the segment base registers can be updated only by changing the segment register. The segment registers contain only 16 bits, which allows only 20 bits to be used for a segment base address (the segment register is shifted left 4 bits to determine the segment base address). If SMRAM is relocated to an address above 1 Mbyte, the segment registers can no longer be initialized to point to SMRAM.

These areas can still be accessed by using address override prefixes to generate an offset to the correct address. For example, if the SMBASE has been relocated immediately below 16 Mbytes, the DS and ES registers are still initialized to 0000 0000h. Data in SMRAM can still be accessed by using 32-bit displacement registers

```
move esi,OOFFxxxxh      ;64K segment
                         immediately below 16M
move ax,ds:[esi]
```

8 TEST REGISTERS 4 AND 5 MODIFICATIONS

The Cache Test Registers for the $Am5_X86$ microprocessor are the same test registers (TR3, TR4, and TR5) provided in Am486 microprocessors. TR3 is the cache test data register. TR4, the cache test status register, and TR5, the cache test control register, operate together with TR3.

If WB/$\overline{WT}$ meets the necessary setup timing and is sampled Low on the falling edge of RESET, the processor is placed in Write-through mode and the test register function is identical to the Am486 microprocessors. If WB/$\overline{WT}$ meets the necessary setup timing and is sampled High on the falling edge of RESET, the processor is placed in Write-back mode and the test registers TR4 and TR5 are modified to support the added write-back cache functionality. Tables 17 and 18 show the individual bit functions of these registers. Sections 8.1 and 8.2 provide a detailed description of the field functions.

Note: *TR3 has the same functions in both Write-through and Write-back modes.These functions are identical to the TR3 register functions provided by Am486 microprocessors.*

8.1 TR4 Definition

This section includes a detailed description of the bit fields defined for TR4.

Note: *Bits listed in Table 17 as Reserved or Not used are not included in these descriptions.*

- *Tag (bits 31–12):* Read/Write, always available in Write-through mode. Available only when EXT=0 in TR5 in Write-back mode. For a cache write, this is the tag that specifies the address in memory. On a cache look-up, this is tag for the selected entry in the cache.

Table 17. Test Register TR4 Bit Descriptions

	31	30–29	28	27–26	25–24	23–22	21–20	19–16	15–12	11	10	9–7	6–3	2–0
EXT = 0	Tag									0	Valid	LRU	Valid (rd)	Not used
EXT = 1	Not used	STn	Rsvd.	ST3	ST2	ST1	ST0	Reserved	Not used		Valid	LRU	Valid (rd)	Not used

Notes:
1. The values of STn and ST3–ST0 are: 00 = Invalid; 01 = Exclusive; 10 = Modified; 11 = Shared.
2. During a cache look-up, bit 11 is read only and always 0. The bit is read/write otherwise.

Table 18. Test Register TR5 Bit Descriptions

	31–20	19	18–17	16	15–12	11–4	3–2	1–0
Write-Back	Not used	Ext	Set State	Reserved	Not used	Index	Entry	Control
Write-Through	Not used					Index	Entry	Control

Notes:
1. Bit 19 in TR5 is EXT. If EXT = 0, TR4 has the standard 486 processor definition for write-through cache.
2. The values of Set State are: 00 = Invalid; 01 = Exclusive; 10 = Modified; 11 = Shared.

- *STn (bits 30–29):* Read Only, available only in Write-back mode when Ext=1 in TR5. STn returns the status of the set (ST3, ST2, ST1, or ST0) specified by the TR5 Set State field (bits 18–17) during cache look-ups. Returned values are:
  - — 00 = invalid
  - — 01 = exclusive
  - — 10 = modified
  - — 11 = shared

- *ST3 (bits 27–26):* Read Only, available only in Write-back mode when Ext=1 in TR5. ST3 returns the status of Set 3 during cache look-ups. Returned values are:
  - — 00 = invalid
  - — 01 = exclusive
  - — 10 = modified
  - — 11 = shared

- *ST2 (bits 25–24):* Read Only, available only in Write-back mode when Ext=1 in TR5. ST2 returns the status of Set 2 during cache look-ups. Returned values are:
  - — 00 = invalid
  - — 01 = exclusive
  - — 10 = modified
  - — 11 = shared

- *ST1 (bits 23–22):* Read Only, available only in Write-back mode when Ext=1 in TR5. ST1 returns the status of Set 1 during cache look-ups. Returned values are:
  - — 00 = invalid
  - — 01 = exclusive
  - — 10 = modified
  - — 11 = shared

- *ST0 (bits 21–20):* Read Only, available only in Write-back mode when Ext=1 in TR5. ST0 returns the status of Set 0 during cache look-ups. Returned values are:
  - — 00 = invalid
  - — 01 = exclusive
  - — 10 = modified
  - — 11 = shared

- *Valid (bit 10):* Read/Write, independent of the Ext bit in TR5. This is the Valid bit for the accessed entry. On a cache look-up, Valid is a copy of one of the bits reported in bits 6–3. On a cache write in Write-through mode, Valid becomes the new Valid bit for the selected entry and set. In Write-back mode, writing to the Valid bit has no effect and is ignored; the Set State bit locations in TR5 are used to set the Valid bit for the selected entry and set.

- *LRU (bits 9–7):* Read Only, independent of the Ext bit in TR5. On a cache look-up, these are the three LRU bits of the accessed set. On a cache write, these bits are ignored; the LRU bits in the cache are updated by the pseudo-LRU cache replacement algorithm. Write operations to these locations have no effect on the device.

- *Valid (bits 6–3):* Read Only, independent of the Ext bit in TR5. On a cache look-up, these are the four Valid bits of the accessed set. In Write-back mode, these valid bits are set if a cache set is in the exclusive, modified, or shared state. Write operations to these locations have no effect on the device.

8.2 TR5 Definition

This section includes a detailed description of the bit fields in the TR5.

Note: *Bits listed in Table 18 as Reserved or Not Used are not included in the descriptions.*

- *Ext (bit 19):* Read/Write, available only in Write-back mode. Ext, or extension, determines which bit fields are defined for TR4: the address TAG field, or the STn and ST3–ST0 status bit fields. In Write-through mode, the Ext bit is not accessible. The following describes the two states of Ext:
  - — Ext = 0, bits 31–11 of TR4 contain the TAG address
  - — Ext = 1, bits 30–29 of TR4 contain STn, bits 27–20 contain ST3–ST0

- *Set State (bits 18–17):* Read/Write, available only in Write-back mode. The Set State field is used to change the MESI state of the set specified by the Index and Entry bits. The state is set by writing one of the following combinations to this field:
  - — 00 = invalid
  - — 01 = exclusive
  - — 10 = modified
  - — 11 = shared

- *Index (bits 11–4):* Read/Write, independent of write-through or Write-back mode. Index selects one of the 256 cache lines.

- *Entry (bits 3–2):* Read/Write, independent of write-through or Write-back mode. Entry selects between one of the four entries in the set addressed by the Set Select during a cache read or write. During cache fill buffer writes or cache read buffer reads, the value in the Entry field selects one of the four doublewords in a cache line.

- *Control (bits 1–0):* Read/Write, independent of Write-through or Write-back mode. The control bits determine which operation to perform. The following is a definition of the control operations:
  - — 00 = Write to cache fill buffer, or read from cache read buffer
  - — 01 = Perform cache write
  - — 10 = Perform cache read
  - — 11 = Flush the cache (mark all entries invalid)

8.3 Using TR4 and TR5 for Cache Testing

The following paragraphs provide examples of testing the cache using TR4 and TR5.

8.3.1 Example 1: Reading The Cache (Write-back Mode Only)

1. Disable caching by setting the CD bit in the CR0 register.
2. In TR5, load 0 into the Ext field (bit 19), the required index into the Index field (bits 10–4), the required entry value into the Entry field (bits 3–2), and 10 into the Control field (bits 1–0). Loading the values into TR5 triggers the cache read. The cache read loads the TR4 register with the TAG for the read entry, and the LRU and Valid bits for the entire set that was read. The cache read loads 128 data bits into the cache read buffer. The entire buffer can be read by placing each of the four binary combinations in the Entry field and setting the Control field in TR5 to 00 (binary). Read each doubleword from the cache read buffer through TR3.
3. Reading the Set State fields in TR4 during Write-back mode is accomplished by setting the Ext field in TR5 to 1 and rereading TR4.

8.3.2 Example 2: Writing The Cache

1) Disable the cache by setting the CD bit in the CR0 register.
2. In TR5, load 0 into the Ext field (bit 19), the required entry value into the Entry field (bits 3–2), and 00 into the Control field (bits 1–0).
3. Load the TR3 register with the data to write to the cache fill buffer. The cache fill buffer write is triggered by loading TR3.
4. Repeat steps 2 and 3 for the remaining three doublewords in the cache fill buffer.
5. In TR4, load the required values into TAG field (bits 31–11) and the Valid field (bit 10). In Write-back mode, the Valid bit is ignored since the Set State field in TR5 is used in place of the TR4 Valid bit. The other bits in TR4 (9:0) have no effect on the cache write.
6. In TR5, load 0 into the Ext field (bit 19), the required value into the Set State field (bits 18–17) (Write-back mode only), the required index into the Index field (bits 10–4), the required entry value into the Entry field (bits 3–2), and 01 into the Control field (bits 1–0). Loading the values into TR5 triggers the cache write. In Write-through mode, the Set State field is ignored, and the Valid bit (bit 10) in TR4 is used instead to define the state of the specified set.

8.3.3 Example 3: Flushing The Cache

The cache flush mechanism functions in the same way in Write-back and Write-through modes. Load 11 into the Control field (bits 1–0) of TR5. All other fields are ignored, except for Ext in Write-back mode. The cache flush is triggered by loading the value into TR5. All of the LRU bits, Valid bits, and Set State bits are cleared.

9 $Am5_X86$ CPU Functional Differences

Several important differences exist between $Am5_X86$ microprocessors and standard Am486DX microprocessors:

- The ID register contains a different version signature.
- The $\overline{EADS}$ function performs cache line write-backs of modified lines to memory in Write-back mode.
- A burst write feature is available for copy-backs. The $\overline{FLUSH}$ pin and WBINVD instruction copy back all modified data to external memory prior to issuing the special bus cycle or reset.

The $Am5_X86$ processor is functionally identical to the Enhanced Am486 processor except for the function of the CLKMUL pin (see Section 9.3) and the redefinition of TR4 and TR5 to access the 16-Kbyte cache (see Section 8).

9.1 Status after Reset

The RESET state is invoked either after power up or after the RESET signal is applied according to the standard Am486DX microprocessor specification.

9.2 Cache Status

After reset, the STATUS bits of all lines are set to 0. The LRU bits of each set are placed in a starting state.

9.3 CLKMUL Pin

For the standard Am486 processor, the Enhanced Am486 processor, and the $Am5_X86$ processor, if the CLKMUL pin is driven High at RESET, the processor uses a Clock-tripled mode.

To ensure correct operation of the 133-MHz $Am5_X86$ processor, always connect the CLKMUL input to V_{SS}.

10 Am5$_X$86 CPU IDENTIFICATION

The Am5$_X$86 microprocessor supports two standard methods for identifying the CPU in a system. The reported values are assigned based on the RESET status of the WB/$\overline{WT}$ pin input (Low = write-through; High = write-back).

10.1 DX Register at RESET

The DX register always contains a component identifier at the conclusion of RESET. The upper byte of DX (DH) contains 04 and the lower byte of DX (DL) contains a CPU type/stepping identifier (see Table 19).

Table 19. CPU ID Codes

CLKMUL Setting/Cache mode	Component ID (DH)	Revision ID (DL)
Write-through mode	04	Ex
Write-back mode	04	Fx

10.2 CPUID Instruction

The Am5$_X$86 microprocessor family implements the CPUID instruction that makes information available to software about the family, model and stepping of the microprocessor on which it is executing. Support of this instruction is indicated by the presence of a user-modifiable bit in position EFLAGS.21, referred to as the EFLAGS.ID bit. This bit is reset to zero at device reset (RESET or SRESET) for compatibility with existing processor designs.

10.2.1 CPUID Timing

CPUID execution timing depends on the selected EAX parameter values (see Table 20).

Table 20. CPUID Instruction Description

OP Code	Instruction	EAX Input Value	CPU Core Clocks	Description
0F A2	CPUID	0 1 >1	41 14 9	AMD string CPU ID Register null registers

10.2.2 CPUID Operation

The CPUID instruction requires the user to pass an input parameter to the CPU in the EAX register. The CPU response is returned to the user in registers EAX, EBX, ECX, and EDX.

When the parameter passed in EAX is zero, the register values returned upon instruction execution are:

EAX[31:0] ← 00000001h
EBX[31:0] ← 68747541h
ECX[31:0] ← 444D4163h
EDX[31:0] ← 69746E65h

The values in EBX, ECX, and EDX indicate an AMD microprocessor. When taken in the proper order:

- EBX (least significant bit to most significant bit)
- EDX (least significant bit to most significant bit)
- ECX (least significant bit to most significant bit)

they decode to

AuthenticAMD

When the parameter passed in EAX is 1, the register values returned are:

EAX[3:0] ← Stepping ID*
EAX[7:4] ← model:
Am5$_X$86 CPU:
Write-through mode = Eh
Write-back mode = Fh
EAX[11:8] ← Family
486 Instruction Set = 4h
EAX[15:12] ← 0000
EAX[31:16] ← RESERVED
EBX[31:0] ← 00000000h
ECX[31:0] ← 00000000h
EDX[31:0] ← 00000001h = all versions
The 1 in bit 0 indicates that the FPU is present

Note:
**Please contact AMD at (800) 222-9323 for stepping ID details.*

The value returned in EAX after CPUID instruction execution is identical to the value loaded into EDX upon device reset. Software must avoid any dependency upon the state of reserved processor bits.

When the parameter passed in EAX is greater than one, register values returned upon instruction execution are:

EAX[31:0] ← 00000000h
EBX[31:0] ← 00000000h
ECX[31:0] ← 00000000h
EDX[31:0] ← 00000000h

Flags affected*: No flags are affected.*
Exceptions*: None*

11 Electrical Data

The following sections describe recommended electrical connections for the $Am5_X86$ microprocessors and electrical specifications.

11.1 Power and Grounding

11.1.1 Power Connections

$Am5_X86$ microprocessors with 16 Kbytes of cache have modest power requirements. However, the high clock frequency output buffers can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean, on-chip power distribution at high frequency, 23 V_{CC} pins and 28 V_{SS} pins feed the microprocessor in the 168-pin PGA package. The 208-pin SQFP package includes 53 V_{CC} pins and 38 V_{SS} pins.

Power and ground connections must be made to all external V_{CC} and V_{SS} pins of the microprocessors. On a circuit board, all V_{CC} pins must connect to a V_{CC} plane. Likewise, all V_{SS} pins must connect to a common GND plane.

The $Am5_X86$ microprocessor family requires only 3.3 V as input power. Unlike other 3-V processors, the $Am5_X86$ microprocessor family does not require a V_{CC5} input of 5 V to indicate the presence of 5-V I/O devices on the system motherboard. For socket compatibility, this pin is INC, allowing the $Am5_X86$ CPU to operate in 3-V sockets in systems that use 5-V I/O.

11.1.2 Power Decoupling Recommendations

Liberal decoupling capacitance should be placed near the microprocessor. The microprocessor, driving its 32-bit parallel address and data buses at high frequencies, can cause transient power surges, particularly when driving large capacitive loads.

Low inductance capacitors and interconnects are recommended for best high-frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the microprocessor and the decoupling capacitors. Capacitors designed specifically for use with PGA packages are commercially available.

11.1.3 Other Connection Recommendations

For reliable operation, always connect unused inputs to an appropriate signal level. Active Low inputs should be connected to V_{CC} through a pull-up resistor. Pull-ups in the range of 20 KΩ are recommended. Active High inputs should be connected to GND.

ABSOLUTE MAXIMUM RATINGS

Case Temperature under Bias . . . – 65°C to +110°C
Storage Temperature . . . . . – 65°C to +150°C
Voltage on any pin
with respect to ground . . . . . – 0.5 V to V_{cc} +2.6 V
Supply voltage with
respect to V_{SS} . . . . . – 0.5 V to +4.6 V

Stresses above those listed under Absolute Maximum Ratings may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to Absolute Maximum Ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices

T_{CASE} . . . . . 0°C to 85°C
V_{CC} . . . . . 3.3 V ±0.3 V

Operating Ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over COMMERCIAL operating ranges

V_{CC} = 3.3 V ± 0.3 V; T_{CASE} = 0°C to + 85°C

Symbol	Parameter	Preliminary Info		Notes
		Min	Max	
V_{IL}	Input Low Voltage	– 0.3 V	+0.8 V	
V_{IH}	Input High Voltage	2.0 V	V_{CC} + 2.4 V	
V_{OL}	Output Low Voltage		0.45 V	Note 1
V_{OH}	Output High Voltage	2.4 V		Note 2
I_{CC}	Power Supply Current: 133 MHz		931 mA	Typical supply current: 825 mA @ 133 MHz. Inputs at rails, outputs unloaded.
$I_{CCSTOPGRANT}$ or $I_{CCAUTOHALT}$	Input Current in Stop Grant or Auto Halt mode 133 MHz		93 mA	Typical supply current for Stop Grant or Auto Halt mode: 50 mA @ 133 MHz.
$I_{CCSTPCLK}$	Input Current in Stop Clock mode		5 mA	Typical supply current in Stop Clock mode is 600 µA.
I_{LI}	Input Leakage Current		±15 µA	Note 3
I_{IH}	Input Leakage Current		200 µA	Note 4
I_{IL}	Input Leakage Current		– 400 µA	Note 5
I_{LO}	Output Leakage Current		±15 µA	
C_{IN}	Input Capacitance		10 pF	F_C = 1 MHz (Note 6)
C_O	I/O or Output Capacitance		14 pF	F_C = 1 MHz (Note 6)
C_{CLK}	CLK Capacitance		12 pF	F_C = 1 MHz (Note 6)

Notes:

1. *This parameter is measured at: Address, Data, $\overline{BE3}$–$\overline{BE0}$ = 4.0 mA; Definition, Control = 5.0 mA*
2. *This parameter is measured at: Address, Data, $\overline{BE3}$–$\overline{BE0}$ = –1.0 mA; Definition, Control = –0.9 mA*
3. *This parameter is for inputs without internal pull-ups or pull-downs and $0 \le V_{IN} \le V_{CC}$.*
4. *This parameter is for inputs with internal pull-downs and V_{IH} = 2.4 V.*
5. *This parameter is for inputs with internal pull-ups and V_{IL} = 0.45 V.*
6. *Not 100% tested.*

SWITCHING CHARACTERISTICS over COMMERCIAL operating ranges

The AC specifications, provided in the AC characteristics table, consist of output delays, input setup requirements, and input hold requirements. All AC specifications are relative to the rising edge of the CLK signal. AC specifications measurement is defined by Figure 39. All timings are referenced to 1.5 V unless otherwise specified. Am5$_X$86 microprocessor output delays are specified with minimum and maximum limits, measured as shown. The minimum microprocessor delay times are hold times provided to external circuitry. Input setup and hold times are specified as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct microprocessor operation.

33-MHz bus (133-MHz operating frequency)

V_{CC} = 3.3 V ±0.3 V; T_{CASE} = 0°C to + 85°C; C_L = 50 pF unless otherwise specified

Symbol	Parameter	Preliminary Info		Unit	Figure	Notes
		Min	Max			
	Frequency	8	33	MHz		Note 2
t_1	CLK Period	30	125	ns	39	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks Notes 3 and 4
t_2	CLK High Time at 2 V	11		ns	39	Note 3
t_3	CLK Low Time at 0.8 V	11		ns	39	Note 3
t_4	CLK Fall Time (2 V–0.8 V)		3	ns	39	Note 3
t_5	CLK Rise Time (0.8 V–2 V)		3	ns	39	Note 3
t_6	A31–A2, PWT, PCD, $\overline{BE3}$–$\overline{BE0}$, M/$\overline{IO}$, D/$\overline{C}$, $\overline{CACHE}$, W/$\overline{R}$, $\overline{ADS}$, $\overline{LOCK}$, $\overline{FERR}$, BREQ, HLDA, $\overline{SMIACT}$, $\overline{HITM}$ Valid Delay	3	14	ns	40	Note 5
t_7	A31–A2, PWT, PCD, $\overline{BE3}$–$\overline{BE0}$, M/$\overline{IO}$, D/$\overline{C}$, $\overline{CACHE}$, W/$\overline{R}$, $\overline{ADS}$, $\overline{LOCK}$ Float Delay	3	20	ns	41	Note 3
t_8	$\overline{PCHK}$ Valid Delay	3	14	ns	42	
t_{8a}	$\overline{BLAST}$, $\overline{PLOCK}$, Valid Delay	3	14	ns	40	
t_9	$\overline{BLAST}$, $\overline{PLOCK}$, Float Delay	3	20	ns	41	Note 3
t_{10}	D31–D0, DP3–DP0 Write Data Valid Delay	3	14	ns	40	
t_{11}	D31–D0, DP3–DP0 Write Data Float Delay	3	20	ns	41	Note 3
t_{12}	$\overline{EADS}$, INV, WB/$\overline{WT}$ Setup Time	5		ns	43	
t_{13}	$\overline{EADS}$, INV, WB/$\overline{WT}$ Hold Time	3		ns	43	
t_{14}	$\overline{KEN}$, $\overline{BS16}$, $\overline{BS8}$ Setup Time	5		ns	43	
t_{15}	$\overline{KEN}$, $\overline{BS16}$, $\overline{BS8}$ Hold Time	3		ns	43	
t_{16}	$\overline{RDY}$, $\overline{BRDY}$ Setup Time	5		ns	44	
t_{17}	$\overline{RDY}$, $\overline{BRDY}$ Hold Time	3		ns	44	
t_{18}	HOLD, AHOLD Setup Time	6		ns	43	
t_{18a}	$\overline{BOFF}$ Setup Time	7		ns	43	
t_{19}	HOLD, AHOLD, $\overline{BOFF}$ Hold Time	3		ns	43	
t_{20}	RESET, $\overline{FLUSH}$, $\overline{A20M}$, NMI, INTR, $\overline{IGNNE}$, $\overline{STPCLK}$, SRESET, $\overline{SMI}$ Setup Time	5		ns	43	Note 5
t_{21}	RESET, $\overline{FLUSH}$, $\overline{A20M}$, NMI, INTR, $\overline{IGNNE}$, $\overline{STPCLK}$, SRESET, $\overline{SMI}$ Hold Time	3		ns	43	Note 5
t_{22}	D31–D0, DP3–DP0, A31–A4 Read Setup Time	5		ns	43, 44	
t_{23}	D32–D0, DP3–DP0, A31–A4 Read Hold Time	3		ns	43, 44	

Notes:

1. *Specifications assume C_L = 50 pF. I/O Buffer model must be used to determine delays due to loading (trace and component). First Order I/O buffer models for the processor are available.*
2. *0-MHz operation guaranteed during stop clock operation.*
3. *Not 100% tested. Guaranteed by design characterization.*
4. *For faster transitions (>0.1% between adjacent clocks), use the Stop Clock protocol to switch operating frequency.*
5. *All timings are referenced at 1.5 V (as illustrated in the listed figures) unless otherwise noted.*

$Am5_X86$ Microprocessor AC Characteristics for Boundary Scan Test Signals at 25 MHz

V_{CC} = 3.3 V ±0.3 V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified

Symbol	Parameter	Preliminary Info		Unit	Figure	Notes
		Min	Max			
t_{24}	TCK Frequency		25	MHz		1x Clock
t_{25}	TCK Period	40		ns	45, 46	Note 1
t_{26}	TCK High Time at 2 V	10		ns	45	
t_{27}	TCK Low Time at 0.8 V	10		ns	45	
t_{28}	TCK Rise Time (0.8 V–2 V)		4	ns	45	Note 2
t_{29}	TCK Fall Time (2 V–0.8 V)		4	ns	45	Note 2
t_{30}	TDI, TMS Setup Time	8		ns	46	Note 3
t_{31}	TDI, TMS Hold Time	7		ns	46	Note 3
t_{32}	TDO Valid Delay	3	25	ns	46	Note 3
t_{33}	TDO Float Delay		36	ns	46	Note 3
t_{34}	All Outputs (Non-Test) Valid Delay	3	25	ns	46	Note 3
t_{35}	All Outputs (Non-Test) Float Delay		30	ns	46	Note 3
t_{36}	All Inputs (Non-Test) Setup Delay	8		ns	46	Note 3
t_{37}	All Inputs (Non-Test) Hold Time	7		ns	46	Note 3

Notes:

1. *TCK period ≥ CLK period.*
2. *Rise/Fall times can be relaxed by 1 ns per 10-ns increase in TCK period.*
3. *Parameter measured from TCK.*

Key to Switching Waveforms

Waveform	Inputs	Outputs
	Must be steady	Will be steady
	May change from H to L	Will change from H to L
	May change from L to H	Will change from L to H
	Don't care; any change permitted	Changing; state unknown
	Does not apply	Center line is High-impedance "Off" state

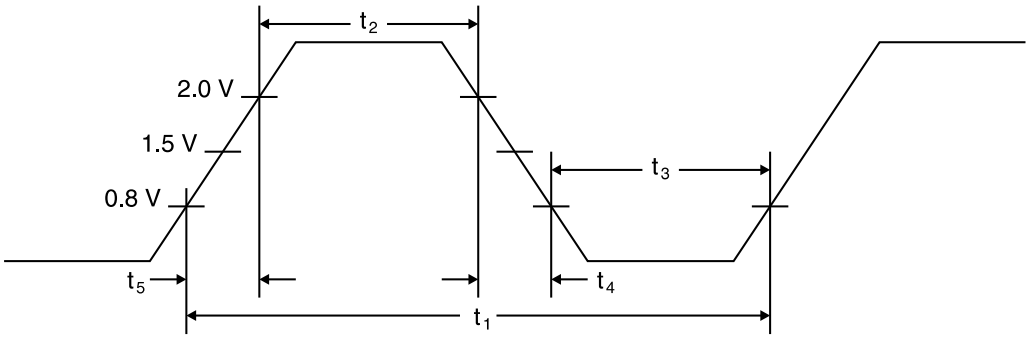


Figure 39. CLK Waveforms

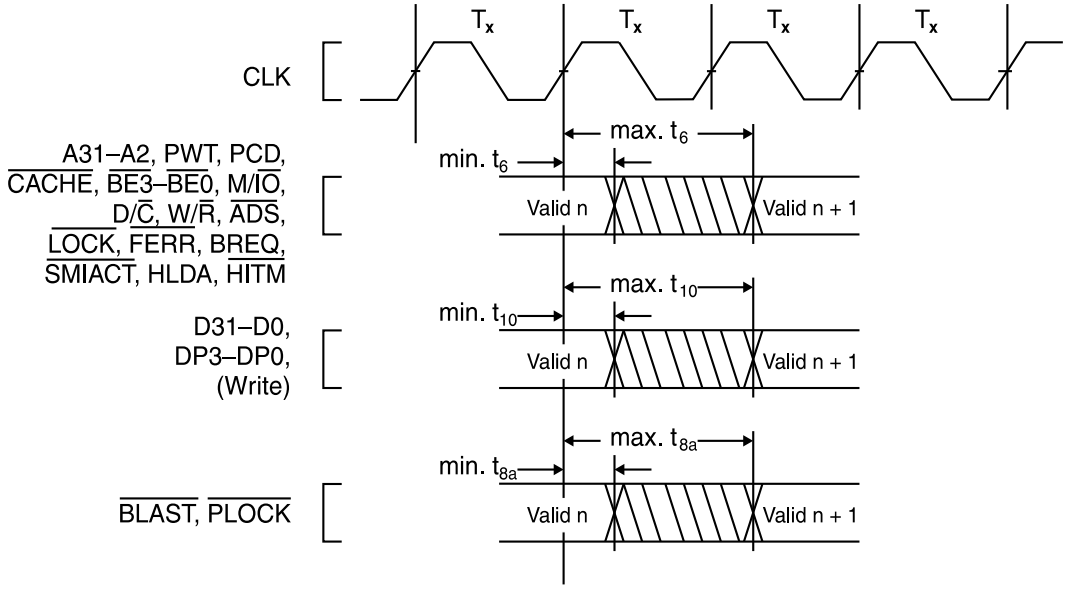


Figure 40. Output Valid Delay Timing

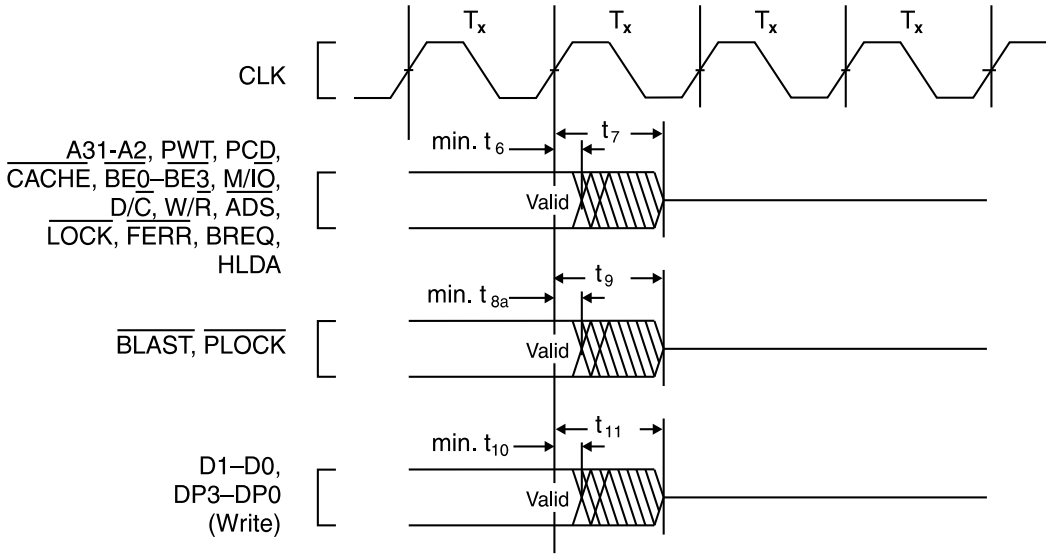


Figure 41. Maximum Float Delay Timing

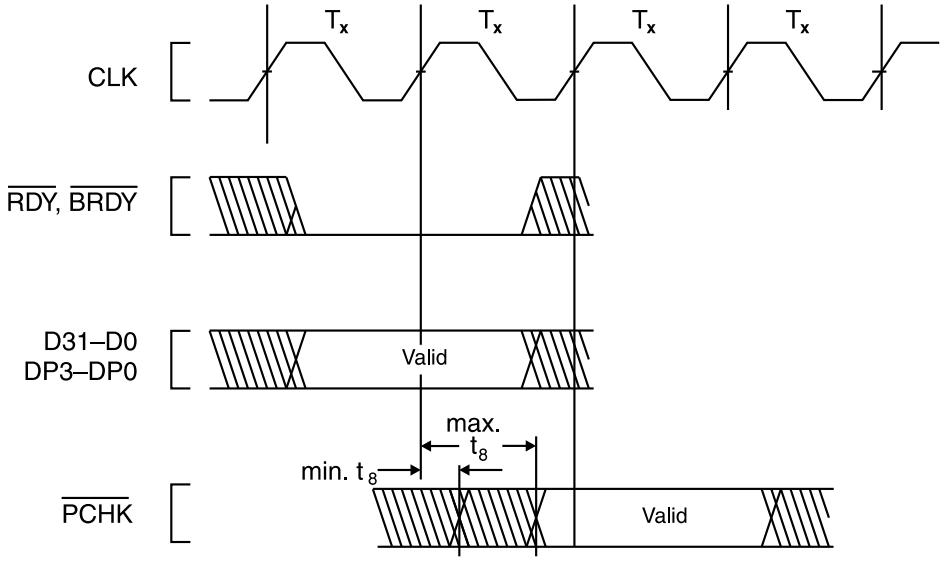


Figure 42. PCHK Valid Delay Timing

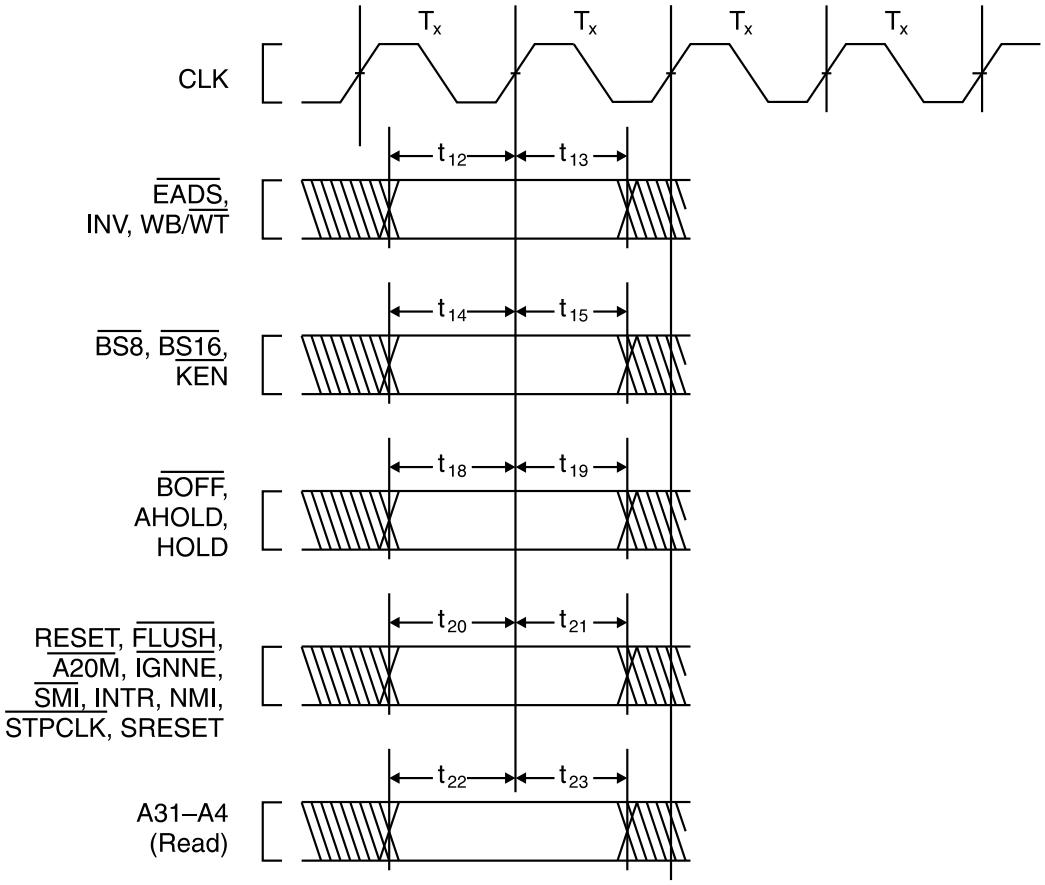


Figure 43. Input Setup and Hold Timing

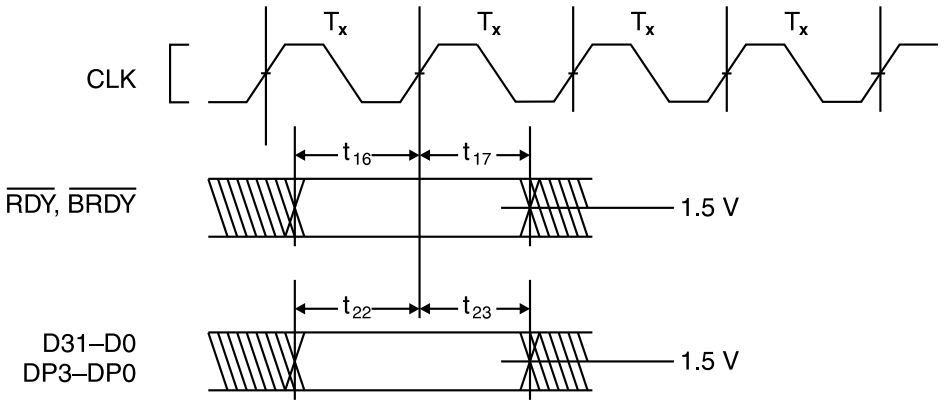


Figure 44. RDY and BRDY Input Setup and Hold Timing

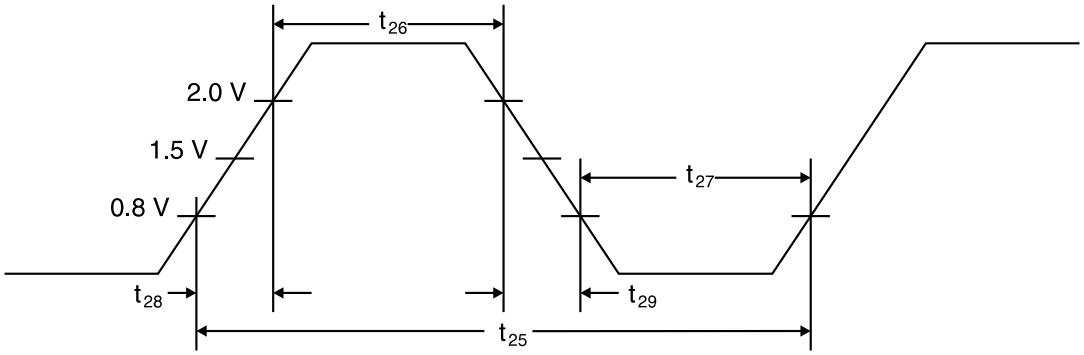


Figure 45. TCK Waveforms

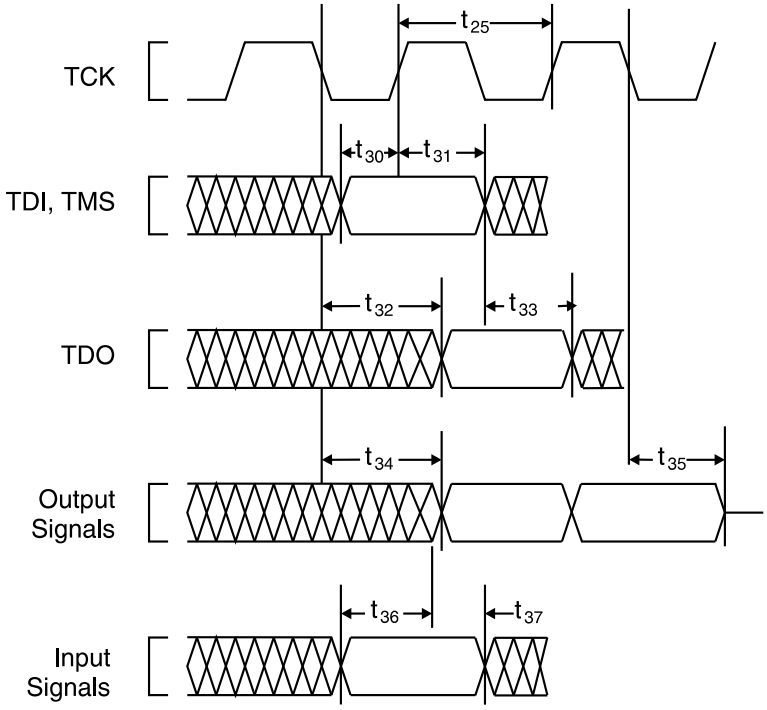


Figure 46. Test Signal Timing Diagram

12 PACKAGE THERMAL SPECIFICATIONS

The $Am5_X86$ microprocessor is specified for operation when T_{CASE} (the case temperature) is within the range of 0°C to +55°C or +85°C. T_{CASE} can be measured in any environment to determine whether the $Am5_X86$ microprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

The ambient temperature (T_A) is guaranteed if T_{CASE} is not violated. The ambient temperature can be calculated from θ_{JC} and θ_{JA} and from the following equations:

$T_J = T_{CASE} + (P \bullet \theta_{JC})$
$T_A = T_J - (P \bullet \theta_{JA})$
$T_{CASE} = T_A + (P \bullet [\theta_{JA} - \theta_{JC}])$

Where:

T_J, T_A, T_{CASE} = Junction, Ambient, and Case Temperature
θ_{JC}, θ_{JA} = Junction-to-Case and Junction-to-Ambient Thermal Resistance, respectively
P = Maximum Power Consumption

The values for θ_{JA} and θ_{JC} are given in Table 21 for the 1.75 sq. in., 168-pin, ceramic PGA. For the 208-pin SQFP plastic package, θ_{JA} = 14.0 and θ_{JC} = 1.5.

Table 22 shows the T_A allowable (without exceeding T_{CASE}) at various airflows and T_{CASE} values for the PGA package. Note that T_A is greatly improved by attaching a heat sink to the package. P (the maximum power consumption) is calculated by using a maximum I_{CC} value of 931 mA at 3.3 V. Table 23 shows the T_A allowable (without exceeding T_{CASE}) for the SQFP package using a maximum I_{CC} value of 931 mA at 3.3 V.

Table 21. Thermal Resistance (°C/W) θ_{JC} and θ_{JA} for the $Am5_X86$ CPU in 168-Pin PGA Package

Cooling Mechanism	θ_{JC}	θ_{JA} vs. Airflow-Linear ft/min. (m/s)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
No Heat Sink	1.5	16.5	14.0	12.0	10.5	9.5	9.0
Heat Sink*	2.0	12.0	7.0	5.0	4.0	3.5	3.25
Heat Sink* and fan	2.0	5.0	4.6	4.2	3.8	3.5	3.25

Note:
**0.350" high unidirectional heat sink (Al alloy 6063-T5, 40 mil fin width, 155 mil center-to-center fin spacing)*

Table 22. Maximum T_A at Various Airflows in °C

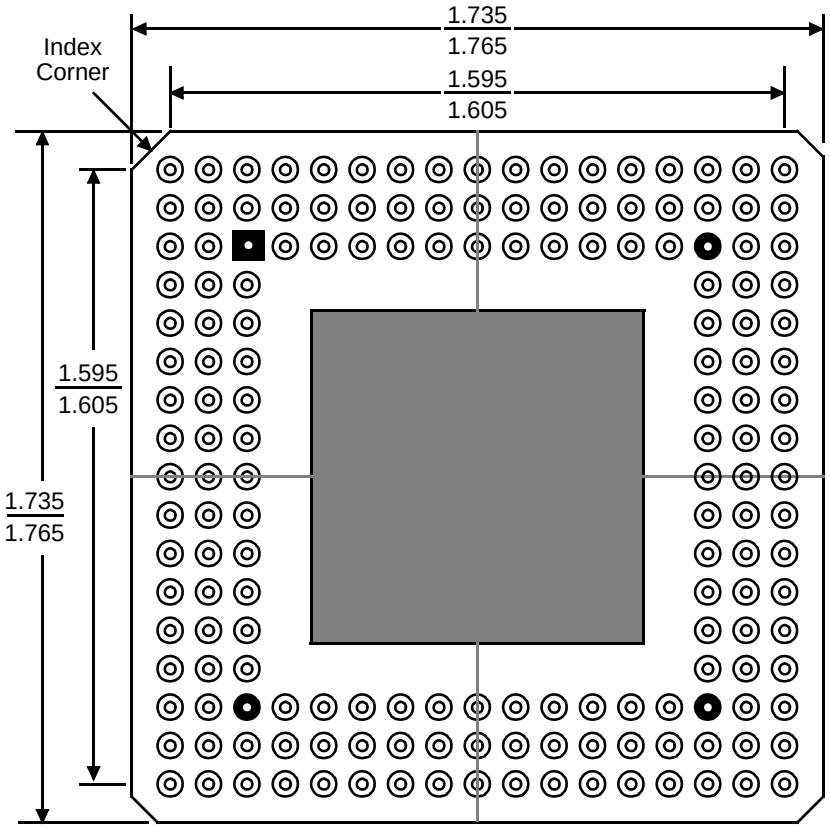
T_A by Cooling Type	T_{CASE}	Clock	Airflow-Linear ft/min. (m/sec)					
			0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
T_A without Heat Sink	55°C	133 MHz	8.9°C	16.6°C	22.7°C	27.3°C	30.4°C	32.0°C
T_A with Heat Sink	55°C	133 MHz	24.3°C	39.6°C	45.8°C	48.9°C	50.4°C	51.2°C
T_A with Heat Sink and fan	55°C	133 MHz	45.8°C	47.0°C	48.2°C	49.5°C	50.4°C	51.2°C
T_A without Heat Sink	85°C	133 MHz	38.9°C	46.6°C	52.7°C	57.3°C	60.4°C	62.0°C
T_A with Heat Sink	85°C	133 MHz	54.3°C	69.6°C	75.8°C	78.9°C	80.4°C	81.2°C
T_A with Heat Sink and fan	85°C	133 MHz	75.8°C	77.0°C	78.2°C	79.5°C	80.4°C	81.2°C

Table 23. Maximum T_A for SQFP Package by Clock Frequency

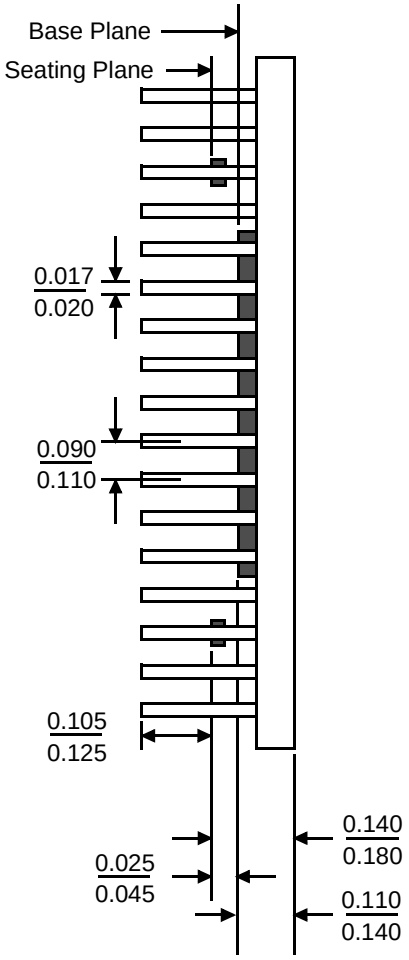
T_{CASE}	Clock	T_A
85°C	133 MHz	46.6°C

13 PHYSICAL DIMENSIONS

168-Pin PGA



Bottom View (Pins Facing Up)

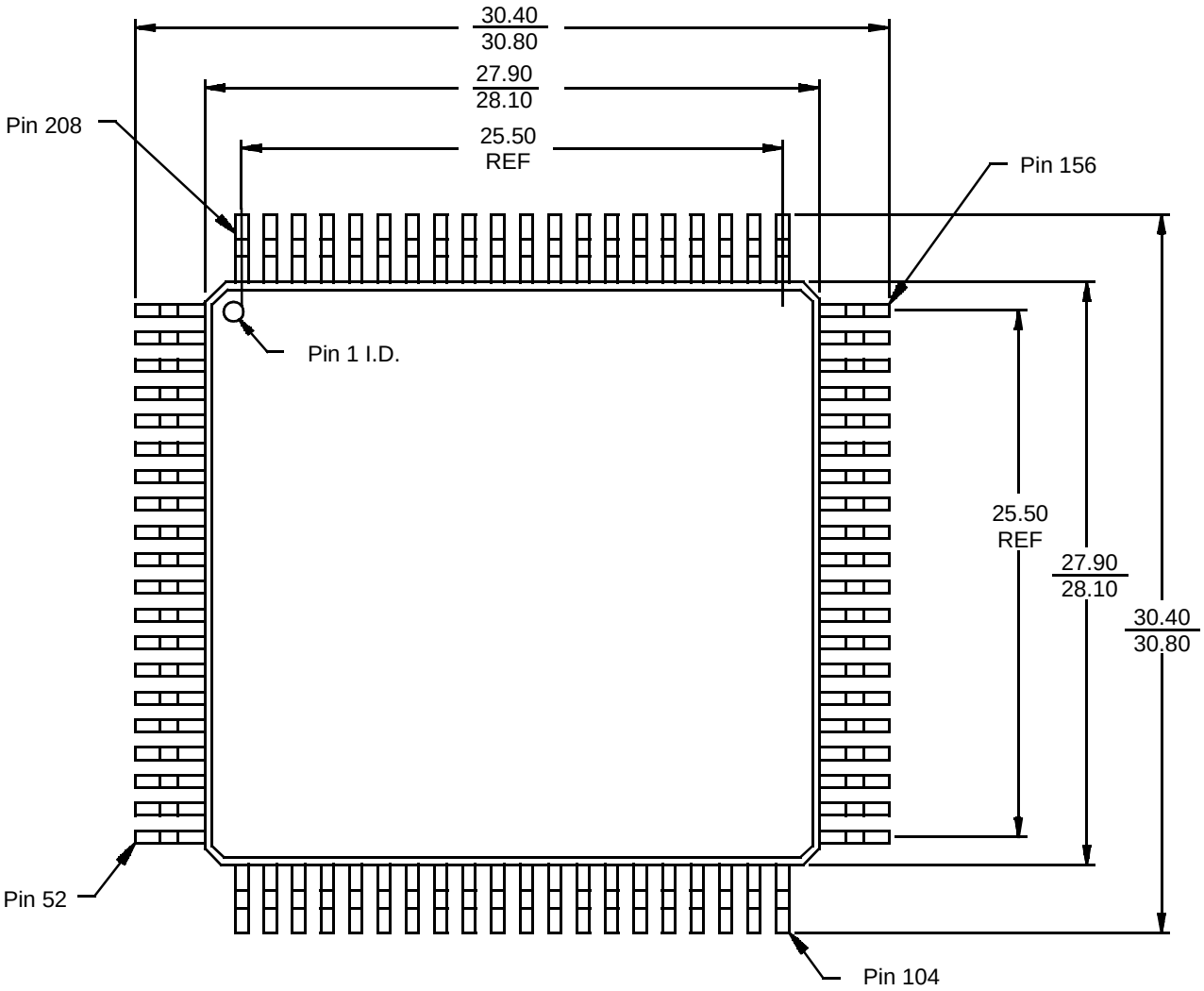


Side View

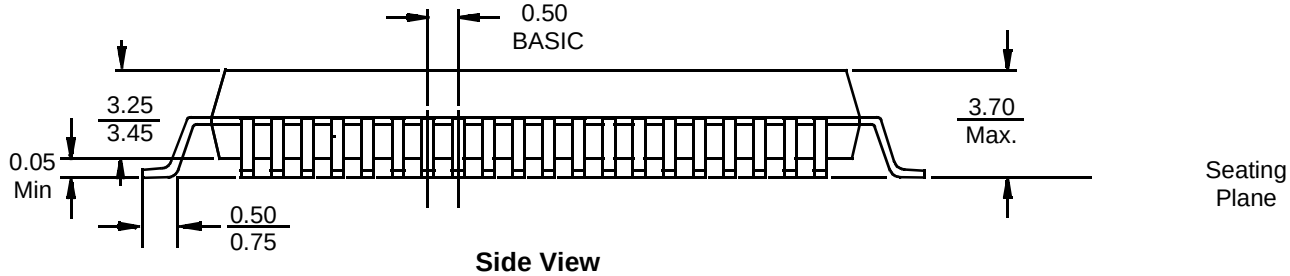
Notes:

1. *All measurements are in inches.*
2. *Not to scale. For reference only.*
3. *BSC is an ANSI standard for Basic Space Centering.*

208-Pin SQFP



Top View



Side View

Notes:

1. *All measurements are in millimeters unless otherwise noted.*
2. *Not to scale. For reference only.*

Trademarks

AMD, Am386, and Am486 are registered trademarks and $Am5_X86$ is a trademark of Advanced Micro Devices, Inc.
FusionPC is a service mark of Advanced Micro Devices, Inc.
Microsoft and Windows are registered trademarks of Microsoft Corp.

Product names used in this publication are for identification purposes only and may be trademarks of their respective companies.