Intel486™ Microprocessor Family Data Sheet Addendum: SL Enhanced Intel486™ Microprocessor Family

November 1993

Order Number: 241696-002

SL Enhanced Intel486 Microprocessor Family

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

PAGE

CONTENTS

PAGE

1.0 INTRODUCTION

Intel is enhancing its entire Intel486™ microprocessor family with the energy-efficient technology driving today's highly integrated Intel486 SL CPUs. The SL Enhanced Intel486 microprocessor family enables built-in power management while maintaining full compatibility with existing Intel architecture processors. The SL Enhanced Intel486 CPU family allows system designers to design desktop systems that exceed the Environmental Protection Agency's (EPA) Energy Star guidelines, without sacrificing performance. Bringing SL Technology to the entire Intel486 CPU line increases notebook system design flexibility and increases the battery life of all high-performance Intel486 CPU-based notebooks. SL technology allows system designers to differentiate their power management schemes with a variety of energy-efficient or battery life-preserving features.

SL Enhanced Intel486 CPU-based systems can maximize both energy efficiency and performance. SL Technology features like **Stop Clock** and **Auto HALT** power down allow CPU standby modes, saving systems up to an additional 4W over previous Intel486 CPUs while providing virtually instantaneous recovery to a full-on state. **Stop Clock** and **Auto Idle** power down also allow active power management of the CPU so systems can save energy even when in use. SL Enhanced OverDrive™ processors provide end-user performance upgradability to SL Enhanced Intel486 CPUs while maintaining their energy efficiency.

SL Enhanced Intel486 CPUs enable power management features to be built into hardware, allowing energy efficiency that is transparent to application and O/S software. **Stop Clock**, **Auto HALT** power down, and **Auto Idle** power down allow software transparent control over CPU power management. Equally important is the capability of the processor to manage system power consumption. The SL Enhanced Intel486 CPU incorporates the same **System Management Mode (SMM)** found in the SL CPU family. A non-maskable **System Management Interrupt (SMI)**, a corresponding **Resume (RSM)** instruction and a new memory space for system management code are the basis of Intel's System Management Mode. Intel's SMM ensures seamless power control of the processor core, system logic, main memory and one or more peripheral devices transparent to any application or operating system.

SL Enhanced Intel486 CPUs are available in a full range of speeds (25 MHz to 66 MHz), packages (PGA, SQFP, PQFP), and voltages (5V, 3.3V) to meet any system design requirement.

This Data Book Addendum highlights the features of Intel's SL Enhanced Intel486 microprocessors and should be used in conjunction with the latest version of the existing Intel486 microprocessor documents, including the following:

- *Intel486™ DX Microprocessor Data Book*, Order No. 240440
- *Intel486™ SX Microprocessor-Intel487™ SX Co-Processor Data Sheet*, Order No. 240950
- *Intel486™ Family of Microprocessors Low Power Version* Data Sheet, Order No. 241199
- *Intel486™ Microprocessor Family Programmer's Reference Manual*, Order No. 240486
- *Intel486™ Microprocessor Hardware Reference Manual*, Order No. 240552
- *Intel486™ DX2 Microprocessor Data Book*, Order No. 241245
- *Pentium™ OverDrive™ Processor Data Sheet*, Order No. 290436
- 1993 Packaging Handbook, Order No. 240800

End-user upgradability for the SL Enhanced Intel486 CPU product family is specified in the OverDrive Processor socket section (Section 9) of this document.

1.1 SL Enhanced Intel486 Microprocessor Features

- *Intel's System Management Mode* — A unique Intel architecture operating mode with its dedicated special purpose interrupt and address space which can be used to implement intelligent power management and other enhanced functions in a manner which is completely transparent to the operating system and applications software.
- *I/O Restart* — An I/O instruction interrupted by a System Management Interrupt (SMI#) can automatically be re-started during the execution of an RSM instruction.
- *Stop Clock* — Provides a clock control mechanism for the SL Enhanced Intel486 CPUs. This mechanism provides two low-power states: a fast wake-up Stop Grant State (~20–55 mA) and a Stop Clock state with **CLK frequency at 0-MHz (~100 μA–200 μA)**.
- *Auto HALT Power Down* — After the execution of a HALT instruction, the CPU issues a normal HALT bus cycle and the clock input to the SL Enhanced Intel486 CPU core is automatically stopped, causing the CPU to enter the Auto HALT Power Down state (~20 mA–55 mA).
- *Auto Idle Power Down* — This function which only applies to Intel486 DX2 CPUs, allows the CPU to reduce its core clock rate to half of its original frequency, without affecting performance. Auto Idle Power Down provides an average power savings of 10%.

- *Upgrade Power Down Mode* — When an OverDrive processor upgrade is installed, this feature detects the presence of the upgrade, then powers down the core, and three-states all outputs of the original CPU, so the original CPU consumes very low current.
- *Package Options* — The SL Enhanced Intel486 CPUs are available in 168 lead PGA, 196 lead PQFP, and 208 lead SQFP packages. The SQFP package provides a 40% volume savings over PQFP, making it an ideal solution for mobile systems.
- *3.3 Volt Operation* — The SL Enhanced Intel486 CPUs are available with either a 3.3V supply voltage or a 5V supply voltage. The 3.3V supply voltage provides a 50% power saving over a 5V supply voltage.
- *Clocking Options* — The SL Enhanced Intel486 CPUs are available with either a 1X clock input or a 2X clock input. The 1X clock option, in which the phases of the core clock are provided by an internal Phase Lock Loop circuit, provides a simpler system design with better I/O-timing performance. The 1X clock option, is required for DX2 clock-doubled systems and OverDrive Processor upgradability. The 2X clock option, in which both phases of the core clock are provided by the system, allows portable designs to use existing Intel486 "LP" CPU clock control mechanisms. The 2X clock option is intended as a short-term solution for portable applications.

All of the SL Enhanced Intel486 CPUs are based on an **Intel486 CPU** core which consists of a 32-bit integer processing unit, an 8 Kbyte cache, and a memory management unit. This ensures full binary compatibility with the 8086, 8088, 80186, 80286, Intel386™ SX, Intel386 DX, and all versions of Intel486 microprocessors. All of the Intel486 microprocessors described in this document offer the following features:

- *Full 32-bit RISC integer processor* — The Intel486 performs a complete set of arithmetic and logical operations on 8-, 16-, and 32-bit data types using a full-width ALU and eight general purpose registers.
- *Single Cycle Execution* — Many instructions execute in a single clock cycle.
- *Instruction Pipelining* — The fetching, decoding, execution, and address translation of instructions is overlapped within the Intel486 microprocessor.
- *On-Chip Floating Point Unit* — (All except Intel486 SX CPU) Supports 32-, 64-, and 80-bit formats specified in IEEE standard 754 are supported. The unit is binary compatible with the 8087, 80287, Intel387™SX and DX, and Intel487™ SX math coprocessors.
- *On-Chip Cache with Cache Consistency Support* — An 8 Kbyte internal write-through cache is used for both data and instructions. Cache hits provide zero wait-state access times for data within the cache. Bus activity is tracked to detect alterations in the memory which the internal cache represents. The internal cache can be invalidated or flushed so that an external cache controller can maintain cache consistency in multi-processor environments.
- *External Cache Control* — Write-back and flush controls for an external L2 cache are provided so the processor can maintain cache consistency in multiprocessor environments.
- *On-Chip Memory Management Unit* — Address management and memory space protection mechanisms maintain the integrity of memory in a multitasking and virtual memory environment. Both segmentation and paging are supported.
- *Separate 32-bit Address and Data Paths* — Up to four gigabytes of physical memory can be addressed directly.
- *Burst Cycles* — Burst transfers allow a new double word to be read from memory on each clock cycle. This capability is especially useful for instruction prefetch and for filling the internal cache.
- *Write Buffers* — The processor contains four write buffers to enhance the performance of consecutive writes to memory. The CPU can continue internal operations after a write, without waiting for the write to be executed on the external bus.
- *Bus Backoff* — If another bus master needs control of the bus during a CPU initiated bus cycle, the Intel486 CPU will float its bus signals, then restart the cycle when the bus becomes available again.
- *Instruction Restart* — Programs can continue execution following an exception generated by an unsuccessful attempt to access memory. This feature is important for supporting demand-paged virtual memory applications.
- *Dynamic Bus Sizing* — External controllers can dynamically alter the effective width of the data bus. Bus widths of 8, 16, or 32 bits can be used.
- *Boundary Scan (JTAG)* — Boundary Scan provides for in-circuit testing of components on printed circuit boards. The Intel Boundary Scan implementation conforms with the IEEE Standard Test Access Port and Boundary Scan Architecture. Not all products are offered with the boundary scan feature.

1.2 The SL Enhanced Intel486 CPU Product Family

Table 1-1 shows the SL Enhanced Intel486 CPUs and existing Intel486 microprocessors available by Clock Mode, Supply Voltage, Maximum Frequency, and Package. An individual product will have either a 1X clock or a 2X clock, but not both. Likewise, an individual product will have either a 5V supply voltage or a 3.3V supply voltage, but not both. Please contact Intel for the latest product availability and specifications.

Table 1-1. Product Options

Existing Intel486 CPUs	SL Enhanced Intel486 CPUs	Vcc	CPU FREQUENCY						168 PGA	196 PQFP	208 SQFP
1X CLOCK											
Intel486 SX CPU	SL Enhanced Intel486 SX CPU	3.3V		25	33						X
		5V		25	33				X	X	
Intel486 DX CPU	SL Enhanced Intel486 DX CPU	3.3V			33						X
		5V			33				X	X	
							50		X		
Intel486 DX2 CPU	SL Enhanced Intel486 DX2 CPU	3.3V				40	50				X
		5V					50	66	X		
2X CLOCK											
Low Power Intel486 SX CPU	SL Enhanced Intel486 SX CPU	3.3V		25	33						X
		5V		25	33					X	
Low Power Intel486 DX CPU	SL Enhanced Intel486 DX CPU	3.3V			33						X
		5V			33					X	

2.0 PIN DESCRIPTION

2.1 Pin Assignments

The following figures show the pin assignments of each package type for the SL Enhanced Intel486 CPU product family. Tables are provided showing the pin differences between the existing Intel486 CPU products and the SL Enhanced Intel486 CPU products.

NOTE:
NC pins should ALWAYS remain unconnected.

196 Lead PQFP — Plastic Quad Flat Pack
- — Package Diagram
- — Pin Assignment Difference Table
- — Pin Assignment Table in numerical order

168 Lead PGA — Pin Grid Array
- — Package Diagram
- — Pin Assignment Difference Table
- — Pin Cross Reference by Pin Name

208 Lead SQFP — Quad Flat Pack
- — Package Diagram
- — Pin Assignment Difference Table
- — Pin Assignment Table in numerical order

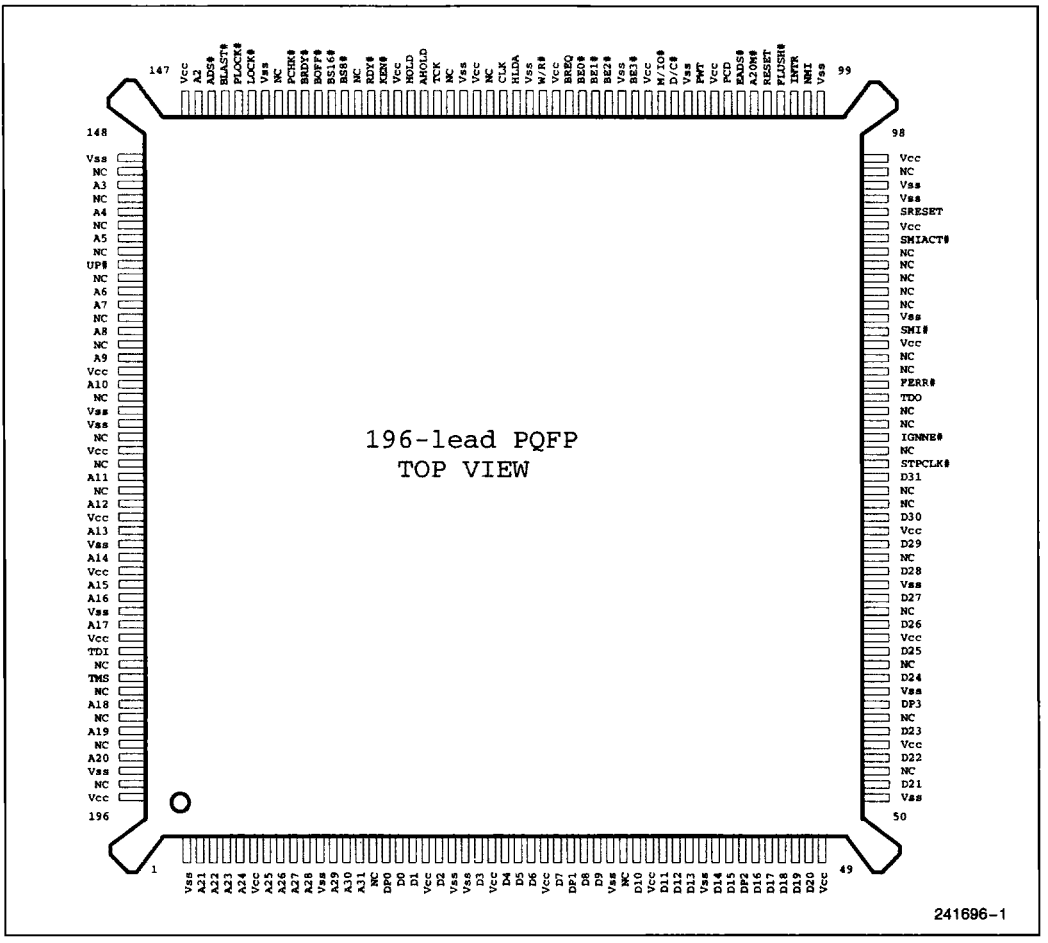


Figure 2-1. Package Diagram (196 Lead PQFP Package)

Table 2-1. Pinout Differences for 196 Lead PQFP Package

Pin #	Intel486 SX CPU	Low Power Intel486 SX CPU	SL Enhanced Intel486 SX CPU	Intel486 DX CPU	Low Power Intel486 DX CPU	SL Enhanced Intel486 DX CPU
75	NC	NC	STPCLK#	NC	NC	STPCLK#
77	NC	NC	NC	IGNNE#	IGNNE#	IGNNE#
81	NC	NC	NC	FERR#	FERR#	FERR#
85	NC	NC	SMI#	NC	NC	SMI#
92	NC	NC	SMIACT#	NC	NC	SMIACT#
94	NC	NC	SRESET	NC	NC	SRESET
127	NC	CLKSEL	NC	NC	CLKSEL	NC

Table 2-2. Pin Assignments for 196 Lead PQFP Package

Pin #	Description	Pin #	Description	Pin #	Description	Pin #	Description
1	V_{SS}	38	D12	75	STPCLK#	112	V_{CC}
2	A21	39	D13	76	NC	113	BE3#
3	A22	40	V_{SS}	77	IGNNE#	114	V_{SS}
4	A23	41	D14	78	NC	115	BE2#
5	A24	42	D15	79	NC	116	BE1#
6	V_{CC}	43	DP2	80	TDO	117	BE0#
7	A25	44	D16	81	FERR#	118	BREQ
8	A26	45	D17	82	NC	119	V_{CC}
9	A27	46	D18	83	NC	120	W/R#
10	A28	47	D19	84	V_{CC}	121	V_{SS}
11	V_{SS}	48	D20	85	SMI#	122	HLDA
12	A29	49	V_{CC}	86	V_{SS}	123	CLK
13	A30	50	V_{SS}	87	NC	124	NC
14	A31	51	D21	88	NC	125	V_{CC}
15	NC	52	NC	89	NC	126	V_{SS}
16	DP0	53	D22	90	NC	127	NC
17	D0	54	V_{CC}	91	NC	128	TCK
18	D1	55	D23	92	SMIACT#	129	AHOLD
19	V_{CC}	56	NC	93	V_{CC}	130	HOLD
20	D2	57	DP3	94	SRESET	131	V_{CC}
21	V_{SS}	58	V_{SS}	95	V_{SS}	132	KEN#
22	V_{SS}	59	D24	96	V_{SS}	133	RDY#
23	D3	60	NC	97	NC	134	NC
24	V_{CC}	61	D25	98	V_{CC}	135	BS8#
25	D4	62	V_{CC}	99	V_{SS}	136	BS16#
26	D5	63	D26	100	NMI	137	BOFF#
27	D6	64	NC	101	INTR	138	BRDY#
28	V_{CC}	65	D27	102	FLUSH#	139	PCHK#
29	D7	66	V_{SS}	103	RESET	140	NC
30	DP1	67	D28	104	A20M#	141	V_{SS}
31	D8	68	NC	105	EADS#	142	LOCK#
32	D9	69	D29	106	PCD	143	PLOCK#
33	V_{SS}	70	V_{CC}	107	V_{CC}	144	BLAST#
34	NC	71	D30	108	PWT	145	ADS#
35	D10	72	NC	109	V_{SS}	146	A2
36	V_{CC}	73	NC	110	D/C#	147	V_{CC}
37	D11	74	D31	111	M/IO#	148	V_{SS}

Table 2-2. Pin Assignments for 196 Lead PQFP Package (Continued)

Pin #	Description	Pin #	Description	Pin #	Description	Pin #	Description
149	NC	161	A8	173	NC	185	TDI
150	A3	162	NC	174	A12	186	NC
151	NC	163	A9	175	V_{CC}	187	TMS
152	A4	164	V_{CC}	176	A13	188	NC
153	NC	165	A10	177	V_{SS}	189	A18
154	A5	166	NC	178	A14	190	NC
155	NC	167	V_{SS}	179	V_{CC}	191	A19
156	UP#	168	V_{SS}	180	A15	192	NC
157	NC	169	NC	181	A16	193	A20
158	A6	170	V_{CC}	182	V_{SS}	194	V_{SS}
159	A7	171	NC	183	A17	195	NC
160	NC	172	A11	184	V_{CC}	196	V_{CC}

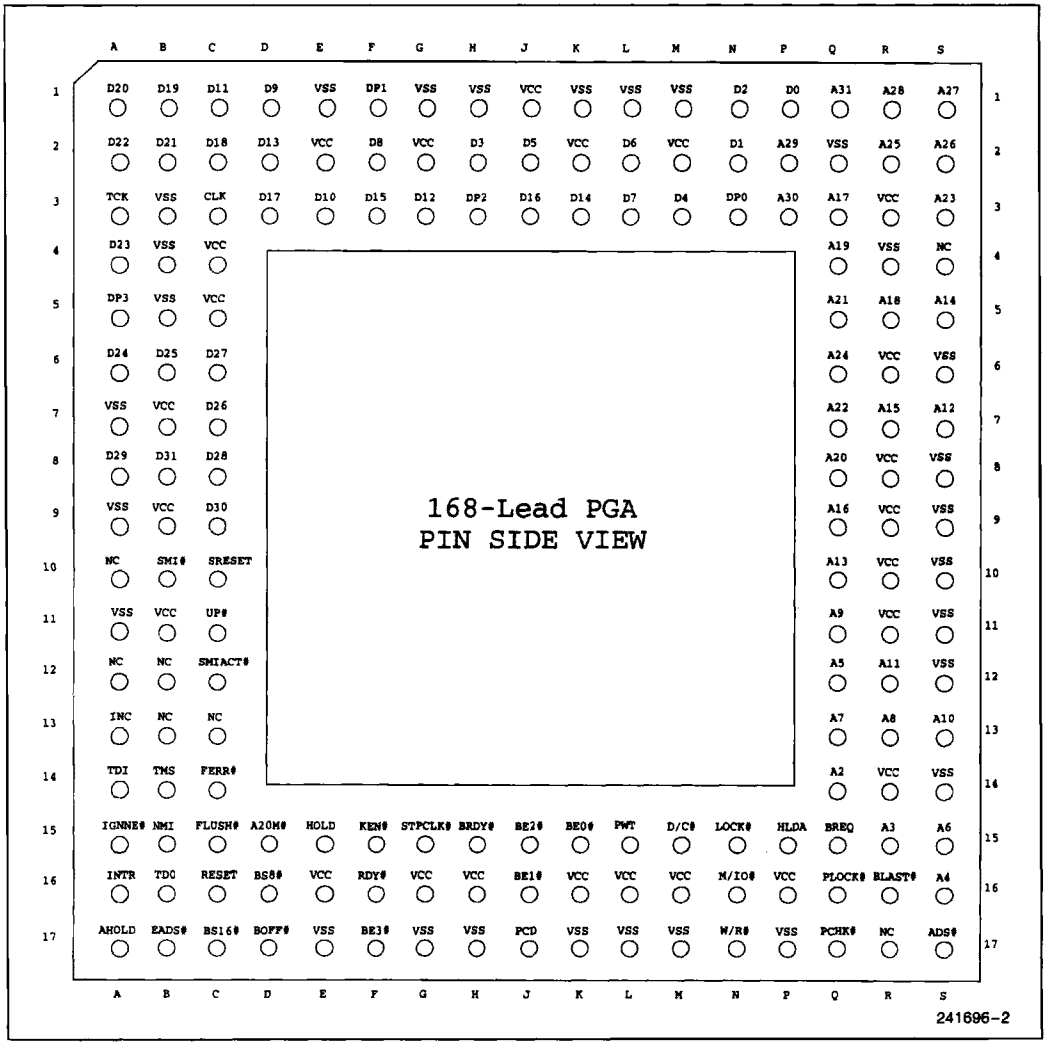


Figure 2-2. Package Diagram for 168 Lead PGA Package

Table 2-3. Pinout Differences for 168 Lead PGA Package

Pin	Intel486 SX CPU	SL Enhanced Intel486 SX CPU	Intel486 DX CPU	SL Enhanced Intel486 DX CPU	Intel486 DX2 CPU	SL Enhanced Intel486 DX2 CPU
A3	NC	NC	NC(5)	TCK	TCK	TCK
A10(1)	NC	NC	NC	NC	NC	NC
A13	NC	INC(2)	NC	INC(2)	NC	INC(2)
A14	NC	NC	NC(5)	TDI	TDI	TDI
A15	NMI	NMI	IGNNE#	IGNNE#	IGNNE#	IGNNE#
B10	NC	SMI#	NC	SMI#	NC	SMI#
B14	NC	NC	NC(5)	TMS	TMS	TMS
B15	NC	NC	NMI	NMI	NMI	NMI
B16	NC	NC	NC(5)	TDO	TDO	TDO
C10	NC	SRESET	NC	SRESET	NC	SRESET
C11	NC	UP#(3)	NC	UP#(3)	UP#	UP#
C12	NC	SMIACT#	NC	SMIACT#	NC	SMIACT#
C14	NC	NC	FERR#	FERR#	FERR#	FERR#
G15	NC	STPCLK#	NC	STPCLK#	NC	STPCLK#
J1	V_{CC}	V_{CC}(4)	V_{CC}	V_{CC}(4)	V_{CC}	V_{CC}(4)

NOTES:
1. Pin A10 is defined as NC. For compatibility with future CPUs, this pin should not be connected.
2. The **INC** pin is defined to be an *internal no-connect*. This means that the pin is not internally connected and may be used for the routing of external signals.
3. The SL Enhanced Intel486 SX and DX CPUs now offer Upgrade Power Down mode through the UP# pin. This is the new power down mechanism that should be used when designing in an OverDrive processor.
4. This pin is a V_{CC} pin. For compatibility with future 3.3V CPUs that will have 5V safe input buffers, this pin should be connected to a V_{CC} trace, not to the V_{CC} plane. For a mixed voltage system, where the CPU inputs are to be driven by 5V logic, this pin should be connected to 5V to bias the input buffers (the 3.3V CPU will require all of the normal V_{CC} pins to be connected to 3.3V). For systems that have all 3.3V logic, this pin should be connected to 3.3V.
5. For the Intel486 DX CPU 50 MHz version, pins A3, A14, B14 and B16 are boundary scan pins.

Table 2-4. Pin Cross Reference for 168 Lead PGA Package

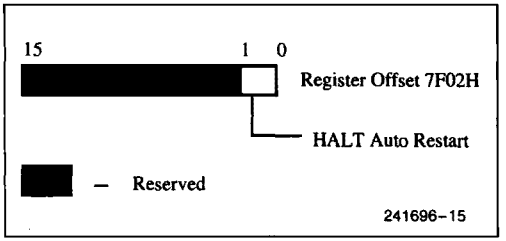
Address		Data		Control		NC	V_{CC}	V_{SS}
A2	Q14	D0	P1	A20M#	D15	A10(1)	B7	A7
A3	R15	D1	N2	ADS#	S17	A13(2)	B9	A9
A4	S16	D2	N1	AHOLD	A17	B12	B11	A11
A5	Q12	D3	H2	BE0#	K15	B13	C4	B3
A6	S15	D4	M3	BE1#	J16	C13	C5	B4
A7	Q13	D5	J2	BE2#	J15	R17	E2	B5
A8	R13	D6	L2	BE3#	F17	S4	E16	E1
A9	Q11	D7	L3	BLAST#	R16		G2	E17
A10	S13	D8	F2	BOFF#	D17		G16	G1
A11	R12	D9	D1	BRDY#	H15		H16	G17
A12	S7	D10	E3	BREQ	Q15		J1	H1
A13	Q10	D11	C1	BS8#	D16		K2	H17
A14	S5	D12	G3	BS16#	C17		K16	K1
A15	R7	D13	D2	CLK	C3		L16	K17
A16	Q9	D14	K3	D/C#	M15		M2	L1
A17	Q3	D15	F3	DP0	N3		M16	L17
A18	R5	D16	J3	DP1	F1		P16	M1
A19	Q4	D17	D3	DP2	H3		R3	M17
A20	Q8	D18	C2	DP3	A5		R6	P17
A21	Q5	D19	B1	EADS#	B17		R8	Q2
A22	Q7	D20	A1	FERR#	C14		R9	R4
A23	S3	D21	B2	FLUSH#	C15		R10	S6
A24	Q6	D22	A2	HLDA	P15			S8

Table 2-4. Pin Cross Reference for 168 Lead PGA Package (Continued)

Address		Data		Control		NC	V_{CC}	V_{SS}
A25	R2	D23	A4	HOLD	E15		R14	S9
A26	S2	D24	A6	IGNNE#	A15			S10
A27	S1	D25	B6	INTR	A16			S11
A28	R1	D26	C7	KEN#	F15			S12
A29	P2	D27	C6	LOCK#	N15			S14
A30	P3	D28	C8	M/IO#	N16			
A31	Q1	D29	A8	NMI	B15			
		D30	C9	PCD	J17			
		D31	B8	PCHK#	Q17			
				PWT	L15			
				PLOCK#	Q16			
				RDY#	F16			
				RESET	C16			
				SMI#	B10			
				SMIACT#	C12			
				UP#	C11(3)			
				W/R#	N17			
				STPCLK#	G15			
				SRESET	C10			
				TCK	A3(4)			
				TDI	A14(4)			
				TDO	B16(4)			
				TMS	B14(4)			

NOTES:
1. Pin A10. See note 1 for Table 2-3.
2. Pin A13. See note 2 for Table 2-3.
3. Pin C11. See note 3 for Table 2-3.
4. Boundary Scan pins are not included on the Intel486 SX CPU in PGA.

4.5.2 AUTO HALT RESTART

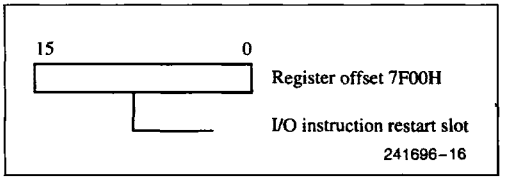


Value of Bit 0 at Entry	Value of Bit 0 at Exit	Comments
0	0	Returns to next instruction in interrupted program.
0	1	Unpredictable
1	0	Returns to next instruction after HALT
1	1	Returns to HALT state

The Auto HALT Restart slot at register offset (word location) 7F02H in SMRAM indicates to the SMM handler that the SMI interrupted the CPU during a HALT state (bit 0 of slot 7F02H is set to 1 if the previous instruction was a HALT). If the SMI did not interrupt the CPU in a HALT state, then the SMI micro code will set bit 0 of the Auto HALT Restart slot to a value of 0. If the previous instruction was a HALT, the SMM handler can choose to either set or reset bit 0. If this bit is set to 1, the RSM micro code execution will force the processor to re-enter the HALT state. If this bit is set to 0 when the RSM instruction is executed, the processor will continue execution with the instruction just after the interrupted HALT instruction. Note that 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.

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.

4.5.3 I/O INSTRUCTION RESTART



Value at Entry	Value at Exit	Comments
00H	00H	Do not restart trapped I/O instruction
00H	0FFH	Restart trapped I/O instruction

The I/O instruction restart slot (register offset 7F00H in SMRAM) gives the SMM handler the option of causing the RSM instruction to automatically re-execute the interrupted I/O instruction. When the RSM instruction is executed, if the I/O instruction restart slot contains the value 0FFH, then the CPU will automatically re-execute the I/O instruction that the SMI trapped. If the I/O instruction restart slot contains the value 00H when the RSM instruction is executed, then the CPU will 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 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 SMI# that originated on a non-I/O instruction boundary.

If the system executes back-to-back SMI requests, the second SMM handler must not set the I/O instruction restart slot (see Section 4.6.6).

4.5.4 SMM BASE RELOCATION

The SL Enhanced Intel486 CPU family provides a new control register, SMBASE. The address space used as SMRAM can be modified by changing the SMBASE register before exiting an SMI handler routine. SMBASE can be changed to any 32K aligned value (values that are not 32K aligned will cause the CPU to enter the shutdown state when executing the RSM instruction). SMBASE is set to the default value of 30000H on RESET, but is not changed on SRESET. If the SMBASE register is changed during an SMM handler, all following SMI# requests will initiate a state save at the new SMBASE.

Table 2-6. Pin Assignment for 208 Lead SQFP Package

Pin#	Description	Pin#	Description	Pin#	Description	Pin#	Description
1	V_{SS}	53	V_{SS}	105	V_{SS}	157	V_{SS}
2	V_{CC}	54	V_{CC}	106	V_{CC}	158	A24
3	V_{CC}(1)	55	V_{SS}	107	V_{SS}	159	A23
4	PCHK#	56	V_{CC}	108	D16	160	A22
5	BRDY#	57	V_{SS}	109	DP2	161	A21
6	BOFF#	58	SRESET	110	V_{SS}	162	V_{CC}
7	BS16#	59	SMIACT#	111	V_{CC}	163	V_{CC}
8	BS8#	60	V_{CC}	112	D15	164	A20
9	V_{CC}	61	V_{SS}	113	D14	165	A19
10	V_{SS}	62	V_{CC}	114	V_{CC}	166	A18
11	NC	63	NC	115	V_{SS}	167	TMS
12	RDY#	64	NC	116	D13	168	TDI
13	KEN#	65	SMI#	117	D12	169	V_{CC}
14	V_{CC}	66	FERR#	118	D11	170	V_{SS}
15	V_{SS}	67	NC	119	D10	171	A17
16	HOLD	68	TDO	120	V_{SS}	172	V_{CC}
17	AHOLD	69	V_{CC}	121	V_{CC}	173	A16
18	TCK	70	NC	122	V_{SS}	174	A15
19	V_{CC}	71	NC	123	D9	175	V_{SS}
20	V_{CC}	72	IGNNE#	124	D8	176	V_{CC}
21	V_{SS}	73	STPCLK#	125	DP1	177	A14
22	V_{CC}	74	D31	126	D7	178	A13
23	V_{CC}	75	D30	127	NC	179	V_{CC}
24	CLK	76	V_{SS}	128	V_{CC}	180	A12
25	V_{CC}	77	V_{CC}	129	D6	181	V_{SS}
26	HLDA	78	D29	130	D5	182	A11
27	W/R#	79	D28	131	V_{CC}	183	V_{CC}
28	V_{SS}	80	V_{CC}	132	V_{SS}	184	V_{SS}
29	V_{CC}	81	V_{SS}	133	V_{CC}	185	V_{CC}
30	BREQ	82	V_{CC}	134	V_{CC}	186	A10
31	BE0#	83	D27	135	V_{SS}	187	A9
32	BE1#	84	D26	136	V_{CC}	188	V_{CC}
33	BE2#	85	D25	137	V_{CC}	189	V_{SS}
34	BE3#	86	V_{CC}	138	V_{SS}	190	A8
35	V_{CC}	87	D24	139	V_{CC}	191	V_{CC}
36	V_{SS}	88	V_{SS}	140	D4	192	A7
37	M/IO#	89	V_{CC}	141	D3	193	A6
38	V_{CC}	90	DP3	142	D2	194	UP#
39	D/C#	91	D23	143	D1	195	A5
40	PWT	92	D22	144	D0	196	A4
41	PCD	93	D21	145	DP0	197	A3
42	V_{CC}	94	V_{SS}	146	V_{SS}	198	V_{CC}
43	V_{SS}	95	V_{CC}	147	A31	199	V_{SS}
44	V_{CC}	96	NC	148	A30	200	V_{CC}
45	V_{CC}	97	V_{SS}	149	A29	201	V_{SS}
46	EADS#	98	V_{CC}	150	V_{CC}	202	A2
47	A20M#	99	D20	151	A28	203	ADS#
48	RESET	100	D19	152	A27	204	BLAST#
49	FLUSH#	101	D18	153	A26	205	V_{CC}
50	INTR	102	V_{CC}	154	A25	206	PLOCK#
51	NMI	103	D17	155	V_{CC}	207	LOCK#
52	V_{SS}	104	V_{SS}	156	V_{SS}	208	V_{SS}

NOTE:
1. This pin is a V_{CC} pin. For compatibility with future 3.3V CPUs that will have 5V safe input buffers, this pin should be connected to a V_{CC} trace, not to the V_{CC} plane. For a mixed voltage system, where the CPU inputs are to be driven by 5V logic, this pin should be connected to 5V to bias the input buffers (the 3.3V CPU will require all of the normal V_{CC} pins to be connected to 3.3V). For systems that have all 3.3V logic, this pin should be connected to 3.3V.

2.2 Quick Pin Reference

Table 2-7. Pin Descriptions

Symbol	Type	Name and Function
CLK	I	**CLocK** provides the fundamental timing and the internal operating frequency for the CPU. All external timing parameters are specified with respect to the rising edge of CLK. For the 1X clock mode the CLK frequency is the same as the frequency of the CPU. For the DX2 clock doubled products the CLK frequency is half the frequency of the CPU.
RESET	I	The **RESET** input forces the CPU to begin execution at a known state. Reset is asynchronous, but must meet setup and hold times t20 and t21 for recognition in any specific clock. The CPU cnnot begin execution of instructions until at least 1 ms after V_{CC} and CLK have reached their proper AC and DC specifications. However, for soft resets, RESET should remain active for at least 15 CLK periods. The RESET pin should remain active during this time to ensure proper CPU operation. RESET is active HIGH. RESET sets the SMBASE descriptor to a default address of 30000H. If the system uses SMBASE relocation, then the SRESET pin should be used for soft resets.
SRESET	I	The SRESET pin duplicates all the functionality of the RESET pin with the following two exceptions: 1. The SMBASE register will retain its previous value. 2. If UP# (I) is asserted, SRESET will not have an effect on the host microprocessor. For soft resets, SRESET should remain active for at least 15 CLK periods. SRESET is active HIGH. SRESET is asynchronous but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
SMI#	I	The **System Management Interrupt** input is used to invoke the System Management Mode (SMM). SMI# is a falling edge triggered signal which forces the CPU into SMM at the completion of the current instruction. SMI# is recognized on an instruction boundary and at each iteration for repeat string instructions. SMI# does not break LOCKed bus cycles and cannot interrupt a currently executing SMM. The CPU will latch the falling edge of one pending SMI# signal while the CPU is executing an existing SMI. The nested SMI will not be recognized until after the execution of a Resume (RSM) instruction.
SMIACT#	O	The **System Management Interrupt ACTive** is an active low output, indicating that the processor is operating in SMM. It is asserted when the CPU begins to execute the SMI state save sequence and will remain active LOW until the processor executes the last state restore cycle out of SMRAM.
STPCLK#	I	The **SToP CLocK request** input signal indicates a request has been made to turn off the CLK input. When the CPU recognizes a STPCLK#, the processor will stop execution on the next instruction boundary, unless superseded by a higher priority interrupt, empty all internal pipelines and the write buffers and generate a Stop Grant acknowledge bus cycle. STPCLK# is active LOW and is provided with an internal pull-up resistor. STPCLK# is an asynchronous signal, but must remain active until the processor issues the Stop Grant bus cycle. STPCLK# may be de-asserted at any time after the processor has issued the Stop Grant bus cycle. Note that STPCLK# should NOT be de-asserted before the CPU has issued the Stop Grant bus cycle.
UP#	I	The **Upgrade Present** input detects the presence of the upgrade processor, then powers down the core, and three-states all outputs of the original CPU, so that the original CPU consumes very low zero current. UP# is active LOW and sampled at all times, including after power-up and during reset.

Table 2-8. Output Pins

Name	Active Level	When Floated
BREQ	HIGH	
HLDA	HIGH	
BE3#–BE0#	LOW	Bus Hold
PWT, PCD	HIGH	Bus Hold
W/R#, M/IO#, D/C#	N/A	Bus Hold
LOCK#	LOW	Bus Hold
PLOCK#	LOW	Bus Hold
ADS#	LOW	Bus Hold
BLAST#	LOW	Bus Hold
PCHK#	LOW	
FERR#*	LOW	
A3–A2	N/A	Bus, Address Hold
SMIACT#	LOW	

Table 2-9. Input/Output Pins

Name	Active Level	When Floated
D31–D0	N/A	Bus Hold
DP3–DP0	HIGH	Bus Hold
A31–A4	N/A	Bus, Address Hold

NOTES:
All input/output signals are floated when UP# is asserted.
* Present on the Intel486 DX and DX2 CPUs only.

Table 2-10. Test Pins

Name	Input or Output	Sampled/Driven On
TCK	Input	N/A
TDI	Input	Rising Edge of TCK
TDO	Output	Falling Edge of TCK
TMS	Input	Rising Edge of TCK

Table 2-11. Input Pins

Name	Active Level	Synchronous/ Asynchronous	Internal Pull-up/ Pull-Down
CLK, CLK2			
RESET	HIGH	Asynchronous	
SRESET	HIGH	Asynchronous	Pull-down
HOLD	HIGH	Synchronous	
AHOLD	HIGH	Synchronous	Pull-down
EADS#	LOW	Synchronous	Pull-up
BOFF#	LOW	Synchronous	Pull-up
FLUSH#	LOW	Asynchronous	Pull-up
A20M#	LOW	Asynchronous	Pull-up
BS16#, BS8#	LOW	Synchronous	Pull-up
KEN#	LOW	Synchronous	Pull-up
RDY#	LOW	Synchronous	
BRDY#	LOW	Synchronous	Pull-up
INTR	HIGH	Asynchronous	
NMI	HIGH	Asynchronous	
IGNNE#*	LOW	Asynchronous	Pull-up
SMI#	LOW	Asynchronous	Pull-up
STPCLK#	LOW	Asynchronous	Pull-up
UP#	LOW		Pull-up
TCK	HIGH		Pull-up
TDI	HIGH		Pull-up
TMS	HIGH		Pull-up

* Present on the Intel486 DX and DX2 CPUs only.

3.0 CLOCK CONTROL

3.1 Clock Generation

The standard Intel486 SX and Intel486 DX CPUs are driven by a 1X clock as opposed to the Intel386 CPUs which use a 2X clock input. The SL Enhanced Intel486 CPUs are available with both the 1X and 2X clocking options. The 1X CLK input frequency is the same as the internal frequency of the CPU. The 1X clock allows simpler system design by cutting in half the clock speed required in the external system. The CLK2 input frequency for 2X CLK CPUs is twice the frequency of the CPU. **Clock control for CPUs with the 2X clock option is described in section 7.3. The remainder of Section 3 discusses clock control for 1X clock CPUs.**

The 1X clock relies on an internal Phase 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 with respect to the rising edge of CLK. The PLL requires a constant frequency CLK input (to within 0.1%), and therefore the CLK input cannot be changed dynamically.

On processors which use an internal clock doubling (DX2), CLK provides the fundamental timing reference for the bus interface unit. The CLK input is doubled internally so that the CPU core will operate at twice the CLK input frequency, hence twice the bus frequency. The internal clock doubler enhances all operations operating out of the internal cache and/or not blocked by external bus accesses. This mode also uses a Phase Lock Loop (PLL) and therefore the CLK input must be maintained at a constant frequency.

3.2 Stop Clock

The SL Enhanced Intel486 CPU provides an interrupt mechanism, STPCLK#, that allows system hardware to control the power consumption of the CPU by stopping the internal clock (output of the PLL) to the CPU core in a controlled manner. This low-power state is called the Stop Grant state. In addition, the STPCLK# interrupt allows the system to change the input frequency within the specified range or completely stop the CLK input frequency (input to the PLL). If the CLK input is completely stopped, the CPU enters into the Stop Clock state—the lowest power state.

STPCLK# is active LOW and is provided with an internal pull-up resistor. STPCLK# is an asynchronous signal, but must remain active until the processor issues the Stop Grant but cycle. STPCLK# may be de-asserted at any time after the processor has issued the Stop Grant bus cycle. Note that STPCLK# should NOT be de-asserted before the CPU has issued the Stop Grant bus cycle. When the CPU enters the Stop Grant state, the internal pull-up resistor is disabled so that the CPU power consumption is reduced. The STPCLK# input must be driven high (not floated) in order to exit the Stop Grant state. STPCLK# must be deasserted for a minimum of 5 clocks after RDY# or BRDY# is returned active for the Stop Grant bus cycle before being asserted again (see Figure 3-1).

There are two targets for the low-power mode supply current:

~20 mA–55 mA in the **Stop Grant state** (fast wake-up, frequency- and voltage-dependent).
and
~100 μA–200 μA in the full **Stop Clock state** (slow wake-up, voltage-dependent).

These are explained in further detail in Sections 3.5.2 and 3.5.3.

When the CPU recognizes a STPCLK# interrupt, the processor will stop execution on the next instruction boundary (unless superseded by a higher priority interrupt), stop the pre-fetch unit, empty all internal pipelines and the write buffers, generate a Stop Grant bus cycle, and then stop the internal clock. At this point the CPU is in the Stop Grant state.

The CPU cannot respond to a 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 STPCLK# will tell the CPU that it can return to program execution at the instruction following the interrupted instruction.

Unlike the normal interrupts, INTR and NMI, the STPCLK# interrupt does not initiate interrupt acknowledge cycles or interrupt table reads. Among external interrupts, STPCLK#'s order of priority is shown as follows:

External Events in Order of Priority

1) RESET/SRESET
2) FLUSH#
3) SMI#
4) NMI
5) INTR
6) STPCLK#

STPCLK# will be recognized while in an interrupt service routine or an SMM handler.

3.3 Stop Grant Bus Cycle

A special Stop Grant bus cycle will be driven to the bus after the CPU recognizes the STPCLK# interrupt. The definition of this bus cycle is the same as the HALT cycle definition for the standard Intel486 microprocessor, with the exception that the Stop Grant bus cycle drives the value 0000 0010H on the address pins. The system hardware must acknowledge this cycle by returning RDY# or BRDY#. **The CPU will not enter the Stop Grant state until either RDY# or BRDY# has been returned.**

The Stop Grant bus cycle is defined as follows:

M/IO# = 0, D/C# = 0, W/R# = 1, Address Bus = 0000 0010H (A_4 = 1), BE3#-BE0# = 1011, Data bus = undefined

The latency between a STPCLK# request and the Stop Grant bus cycle is dependent on the current instruction, the amount of data in the CPU write buffers, and the system memory performance.

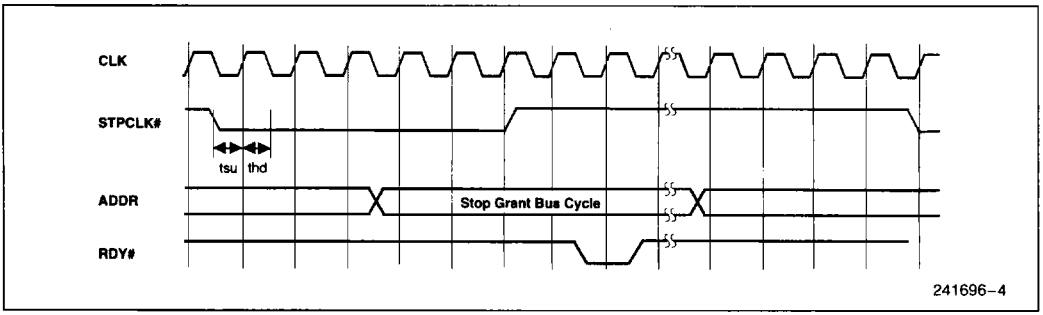


Figure 3-1. Stop Clock Protocol

3.4 Pin State During Stop Grant

During the Stop Grant state, most output and input/output signals of the microprocessor will maintain their previous condition (the level they held when entering the Stop Grant state). The data and data parity signals will be three-stated. In response to HOLD being driven active during the Stop Grant state (when the CLK input is running), the CPU will generate HLDA and three-state all output and input/output signals that are three-stated during the HOLD/HLDA state. After HOLD is de-asserted all signals will return to their prior state before the HOLD/HLDA sequence.

Table 3-1. 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
BE3#-BE0#	O	Previous State
DP3-DP0	I/O	Floated
W/R#, D/C#, M/IO#	O	Previous State
ADS#	O	Inactive
LOCK#, PLOCK#	O	Inactive
BREQ	O	Previous State
HLDA	O	As per HOLD
BLAST#	O	Previous State
FERR#	O	Previous State
PWT/PCD	O	Previous State
PCHK#	O	Previous State
SMIACT#	O	Previous State

In order to achieve the lowest possible power consumption during the Stop Grant state, the system designer must ensure the input signals with pull-up resistors are not driven LOW and the input signals with pull-down resistors are not driven HIGH. (See Table 2-10 in the Quick Pin Reference section for signals with internal pull-up and pull-down resistors).

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 with future processors, data pins must be driven low to achieve the lowest possible power consumption.

3.5 Clock Control State Diagram

The following state descriptions and diagram show the state transitions during a Stop Clock cycle for 1X clock mode (including the Intel486 DX2 CPU clock mode). Refer to Figure 3-2 for the Stop Clock state diagram.

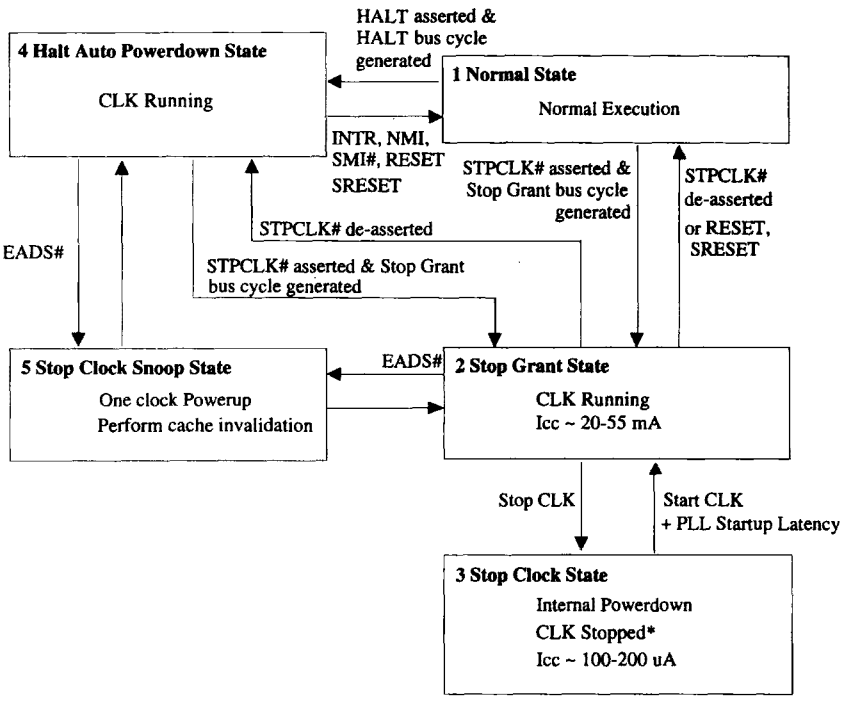
3.5.1 NORMAL STATE

This is the normal operating state of the CPU.

3.5.2 STOP GRANT STATE

The **Stop Grant state** (~20–55 mA) provides a fast wake-up state that can be entered by simply asserting the external STPCLK# interrupt pin. Once the Stop Grant bus cycle has been placed on the bus, and either RDY# or BRDY# is returned, the CPU is in this ~20–55 mA state (depending on the CLK input frequency). The CPU returns to the normal execution state 10–20 clock periods after STPCLK# has been de-asserted.

While in the Stop Grant state, the pull-up resistors on STPCLK# and UP# are disabled internally. The system must continue to drive these inputs to the state



*The system can change the input frequency within the specified range or completely stop the CLK input frequency (input to PLL).

Figure 3-2. Stop Clock State Machine - 1X Clock Mode

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.

A RESET or SRESET will bring the CPU from the Stop Grant state to the Normal state. The CPU will recognize the inputs required for cache invalidation's (HOLD, AHOLD, BOFF# and EADS#) as explained later in this section. The CPU will not recognize any other inputs while in the Stop Grant state. Input signals to the CPU will not be recognized until 1 CLK after STPCLK# is de-asserted (see Figure 3-3).

While in the Stop Grant state, the CPU will not recognize transitions on the interrupt signals (SMI#, NMI, and INTR). Driving an active edge on either SMI# or NMI will 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 de-asserted, the corresponding interrupt will be serviced. The SL Enhanced Intel486 CPU product family requires INTR to be held active until the CPU issues an interrupt acknowledge cycle in order to guarantee recognition. This condition also applies to the existing Intel486 CPUs (see Figure 3-3).

When the CPU is in the Stop Grant state, the system is allowed to stop or change the CLK input. If the CPU clock input frequency is changed, the CPU will not return to the Stop Grant state until the CLK input has been running at a constant frequency for the time period necessary for the PLL to stabilize. This constant frequency must be within the specified operating frequency range of the CPU.

When the CLK input to the CPU is stopped (or changed), current members of the SL Enhanced Intel486 CPU family require the CLK input to be held at a constant frequency for a minimum of 1 ms before de-asserting STPCLK#. This 1 ms time period is necessary so that the PLL can stabilize, and it must be met before the CPU will return to the Stop Grant state. Future versions of the SL Enhanced Intel486 CPU family may require substantially less than 1 ms for the PLL to stabilize. In order to take advantage of a shorter PLL stabilization period, a system design should provide for a programmable time period between restarting the CPU clock and de-asserting STPCLK#.

The CPU will generate a Stop Grant bus cycle only when entering that 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 will not generate a Stop Grant bus cycle.

3.5.3 STOP CLOCK STATE

Stop Clock state (~ 100–200 μA) 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 (with the exception of INTR, NMI and SMI) before the CPU has returned to the Stop Grant state will result in unpredictable behavior. If INTR is driven active while the CLK input is stopped, and held active until the CPU issues an interrupt acknowledge bus cycle, it will be serviced in the normal manner. The system design must ensure the CPU is in the correct state prior to asserting cache invalidation or interrupt signals to the CPU.

The CPU will return to the ~20-55 mA Stop Grant state after the CLK input has been running at a constant frequency for a period of time equal to the PLL startup latency (see section 3.5.2). The CLK input can be restarted to any frequency between the minimum and maximum frequency listed in the A.C. timing specifications.

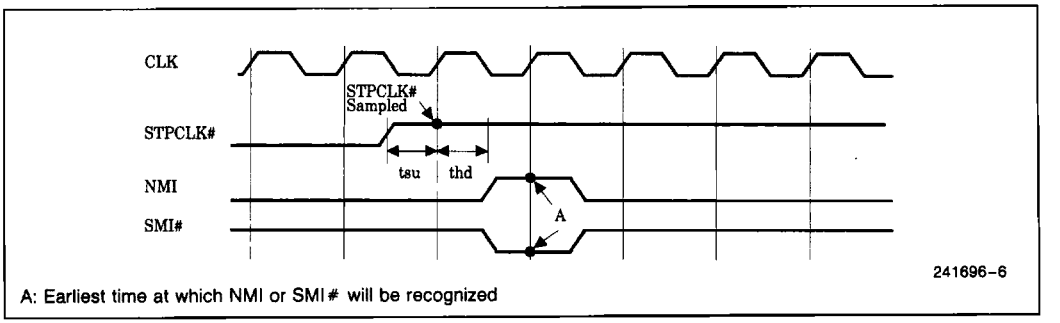


Figure 3-3. Recognition of Inputs When Exiting Stop Grant State

3.5.4 AUTO HALT POWER DOWN STATE

The execution of a HALT instruction will also cause the CPU to automatically enter a ~20–55 mA state, called the **Auto HALT Power Down state**. The CPU will issue a normal HALT bus cycle when entering this state. The CPU will transition to the Normal state on the occurrence of INTR, NMI, SMI#, RESET, or SRESET.

The system can generate a STPCLK# while the CPU is in the Auto HALT Power Down state. The CPU will generate a Stop Grant bus cycle when it enters the Stop Grant State from the HALT state. When the system de-asserts the STPCLK# interrupt, the CPU will return execution to the HALT state. The CPU will generate a new HALT bus cycle when it re-enters the HALT state from the Stop Grant state.

3.5.5 STOP CLOCK SNOOP STATE (CACHE INVALIDATIONS)

When the CPU is in the Stop Grant state or the Auto HALT Power Down state, the CPU will recognize HOLD, AHOLD, BOFF# and EADS# for cache invalidation. When the system asserts HOLD, AHOLD, or BOFF#, the CPU will float the bus accordingly. When the system then asserts EADS#, the CPU will transparently enter the **Stop Clock Snoop state** and will power up for 1 full core clock in order to perform the required cache snoop cycle. It will then re-freeze the clock to the CPU core and return to the previous state. The CPU does not generate a bus cycle when it returns to the previous state.

A FLUSH# event during the Stop Grant state or the Auto HALT Power Down state will be latched and acted upon by asserting the internal FLUSH# signal for one clock upon re-entering the Normal state.

3.5.6 AUTO IDLE POWER DOWN STATE

When the chip is known to be truly idle and waiting for a ready from a memory or I/O bus cycle read, the Intel486 DX2 CPU will reduce its core clock rate to half of its original frequency without affecting performance. When any ready is asserted, the part will return to clocking the core at the doubled DX2 clock. This functionality is transparent to software and external hardware. This feature applies only to those members of the SL Enhanced Intel486 CPU family that have a doubled core clock rate. Auto Idle Power Down provides an average power savings of 10%.

3.6 Supply Current Model for Stop Clock Modes and Transitions

The following diagram illustrates the effect of different Stop Clock state transitions on the supply current.

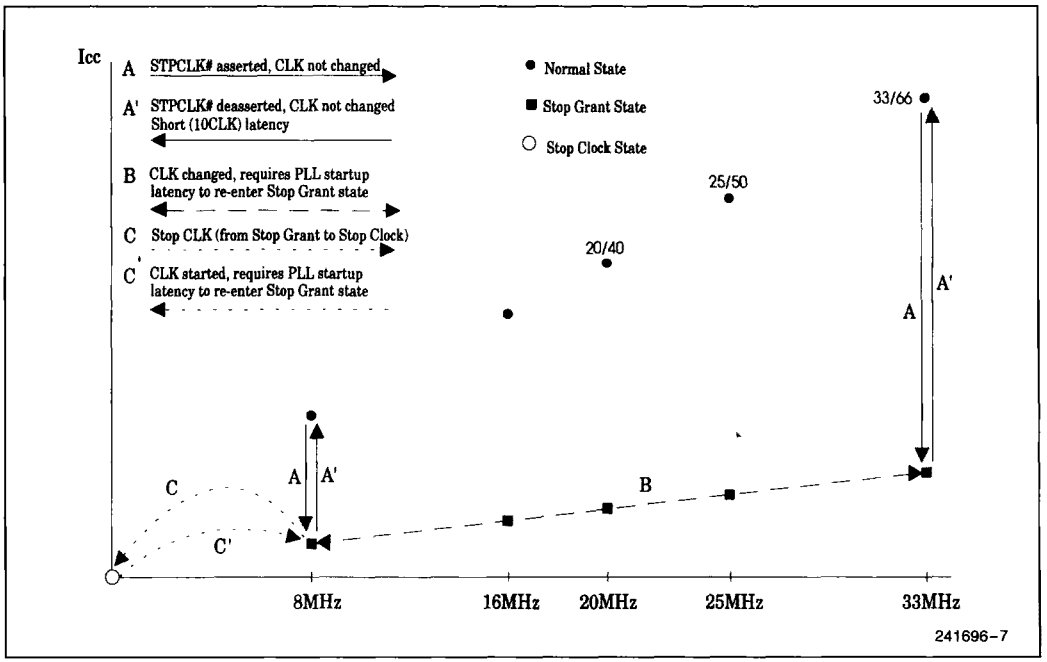


Figure 3-4. Supply Current Model for Stop Clock Modes and Transitions

4.0 INTEL'S SYSTEM MANAGEMENT MODE ARCHITECTURE

4.1 SMM Overview

The SL Enhanced Intel486 microprocessor supports four modes: **Real, Virtual-86, Protected and System Management Mode (SMM)**. As an operating mode, SMM has its distinct processor environment, interface and hardware/software features.

SMM provides the system designer with a means of adding new software controlled features to their computer products which 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:

1. A System Management Interrupt (SMI#) hardware interface.
2. A dedicated and secure memory space (SMRAM) for SMI handler code and CPU state (context) data with a status signal for the system to decode access to that memory space, SMIACT#.
3. Resume (RSM) instruction, for exiting the System Management Mode.
4. Special Features such as I/O Restart, for transparent power management of I/O peripherals, and Auto HALT Restart.

4.2 Terminology

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

SMM: System Management Mode. This is the operating environment that the processor (system) enters when the System Management Interrupt is being serviced.

SMI: System Management Interrupt. This is part of the SMM interface. When SMI# is asserted (SMI# pin asserted low) it causes the processor to invoke SMM. **The SMI# pin is the only means of entering SMM.**

SMM handler: System Management Mode handler. This is the code that will be executed when the processor is in SMM. An example application that this code might implement is a power management control or a system control function.

RSM: Resume instruction. This instruction is used by the SMM handler to exit the SMM and return to the interrupted OS or Application process.

SMRAM: This is the physical memory dedicated to SMM. The SMM handler code and related data reside in this memory. This memory is also used by the processor to store its context before executing the SMM handler. The operating system and applications do not have access to this memory 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.

4.3 System Management Interrupt Processing

The system interrupts the normal program execution and invokes SMM by generating a System Management Interrupt (SMI#) to the CPU. The CPU will service the SMI# by executing the following sequence.

1. The CPU asserts the SMIACT# signal, indicating to the system that it should enable the SMRAM.
2. The CPU saves its state (context) to SMRAM, starting at address location 3FFFFH, proceeding downward in a stack-like fashion.
3. The CPU switches to the System Management Mode processor environment (a pseudo-real mode).
4. The CPU will then jump to the absolute address of 38000H in SMRAM to execute the SMI handler. This SMI handler performs the system management activities.
5. The SMI handler will then execute the RSM instruction which restores the CPU's context from SMRAM, de-asserts the SMIACT# signal, and then returns control to the previously interrupted program execution.

The System Management Interrupt hardware interface consists of the SMI# interrupt request input and the SMIACT# output used by the system to decode the SMRAM.

4.3.1 SYSTEM MANAGEMENT INTERRUPT (SMI#)

SMI# is a falling-edge triggered, non-maskable interrupt request signal. SMI# is an asynchronous signal, but setup and hold times, t20 and t21, must be met in order to guarantee recognition in a specific clock. The SMI# input need not remain active until the interrupt is actually serviced. The SMI# input only needs to remain active for a single clock if the required setup and hold times are met. SMI# will also work correctly if it is held active for an arbitrary number of clocks.

The SMI# input must be held inactive for at least four clocks after it is asserted to reset the edge triggered logic.

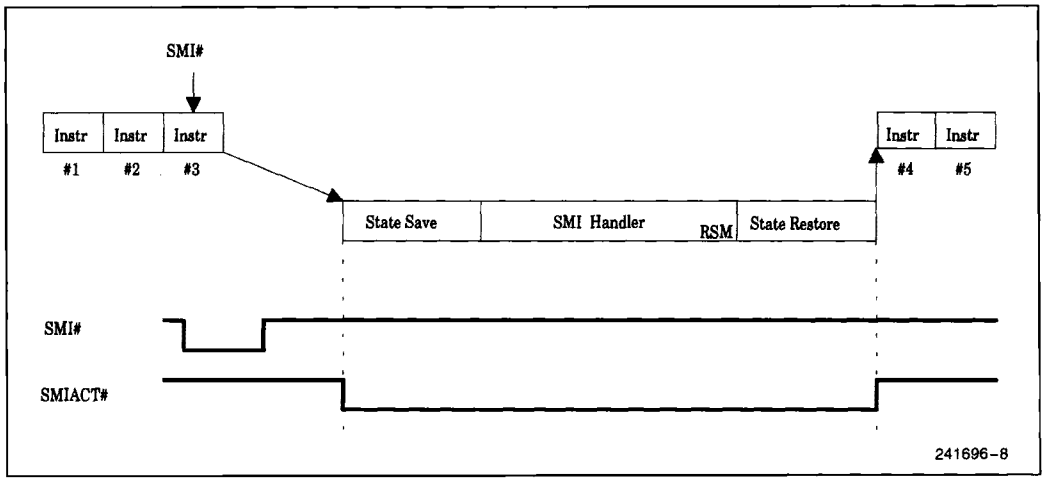


Figure 4-1. Basic SMI# Interrupt Service

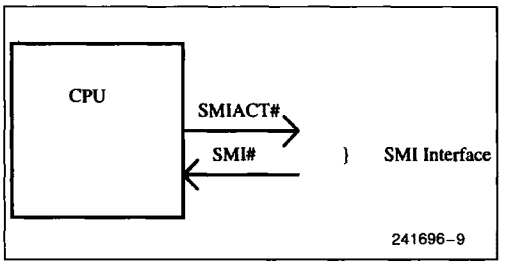


Figure 4-2. Basic SMI# Hardware Interface

SMI#, like NMI, is not affected by the IF bit in the EFLAGS register and is recognized on an instruction boundary. An SMI# will not break locked bus cycles. The SMI# has a higher priority than NMI and is not masked during an NMI.

After the SMI# interrupt is recognized, the SMI# signal will be masked internally until the RSM instruction is executed and the interrupt service routine is complete. Masking the SMI# prevents recursive SMI# calls. If another SMI# occurs while the SMI# is masked, the pending SMI# will be 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 SMM handlers. Only one SMI# can be pending while SMI# is masked.

The SMI# signal is synchronized internally and must be asserted at least three (3) CLK periods prior to asserting the RDY# signal in order to guarantee recognition on a specific instruction boundary. This is important for servicing an I/O trap with an SMI# handler. In order for SMI# to be recognized with respect to SRESET, SMI# should not be asserted until two (2) clocks after SRESET becomes inactive.

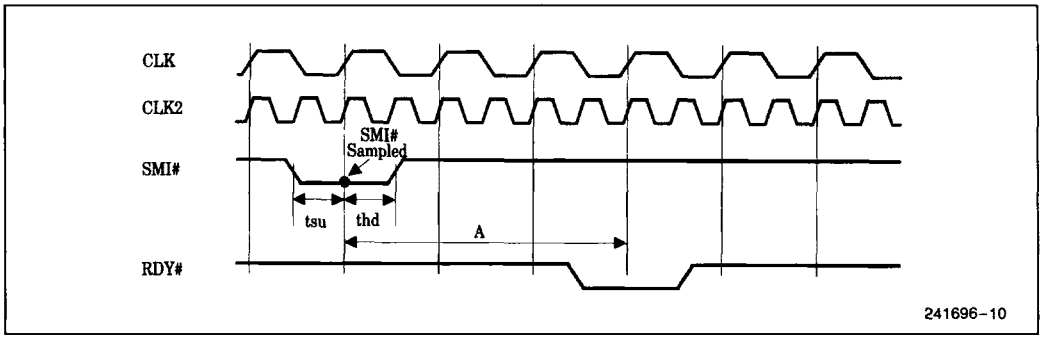


Figure 4-3. SMI# Timing for Servicing an I/O Trap

4.3.2 SMI ACTIVE (SMIACT#)

SMIACT# indicates that the CPU is operating in System Management Mode. 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.

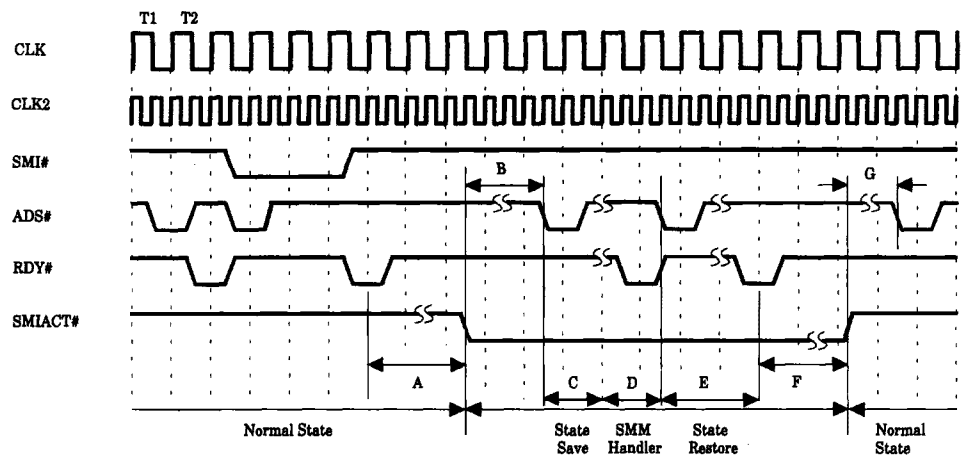
NOTE:
The number of CLKs required to complete the SMM state save and restore is very dependent on system memory performance. The values shown in Figure 4-4 assume 0 wait-state memory writes (2 CLK cycles), 2-1-1-1 burst read cycles, and 0 wait-state non-burst reads (2 CLK cycles). Additionally, it is assumed that the data read during the SMM state restore sequence is not cacheable.

As shown in Figure 4-4, the minimum time required to enter an SMI handler routine for the Intel486 DX CPU (from the completion of the interrupted instruction) is given by:

Latency to beginning of SMI handler = A + B + C = 306 CLKs

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

Latency to continue interrupted application = E + F + G = 327 CLKs



	Intel486 DX CPU	Intel486 DX2 CPU
A: Last RDY# from non-SMM transfer to SMIACT# assertion	2 CLK minimum	2 CLK minimum
B: SMIACT# assertion to first ADS# for SMM state save	40 CLK minimum	20 CLK minimum
C: SMM state save (dependent on memory performance)	Approximately 264 CLKs	Approximately 139 CLKs
D: SMM handler	User determined	User determined
E: SMM state restore (dependent on memory performance)	Approximately 303 CLKs	Approximately 236 CLKs
F: Last RDY# from SMM transfer to de-assertion of SMIACT#	4 CLK minimum	2 CLK minimum
G: SMIACT# de-assertion to first non-SMM ADS#	20 CLK minimum	20 CLK minimum

Figure 4-4. SMIACT# Timing

4.3.3 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.

The CPU asserts the SMIACT# output to indicate to the memory controller that it is operating in System Management Mode. The system logic should ensure that only the CPU has 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 from 38000H-3FFFFH as SMRAM area. The CPU will save its state to the state save area from 3FFFFH downward to 3FE00H. After saving its state the CPU will jump 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 will enable initialization of the SMRAM space (i.e., loading SMI handler) before executing the SMI handler during SMM. See Figure 4-5.

4.3.3.1 SMRAM State Save Map

When the SMI# is recognized on an instruction boundary, the CPU core first sets the 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 CS Base + [8000H + 7FFFH]. The default CS Base is 30000H, therefore the default state save area is at 3FFFFH. In this case, the CS Base can also be referred to as the SMBASE.

If the *SMBASE Relocation feature* is enabled, then the SMRAM addresses can change. The following formula is used to determine the relocated addresses where the context is saved. The context will reside at CS Base + [8000H + Register Offset], where the default initial CS Base is 30000H and the Register Offset is listed in the SMRAM state save map (Table 4-1). Reserved spaces will be used to accommodate new registers in future CPUs. The state save area starts at 7FFFH and continues downward in a stack-like fashion.

Some of the registers in the SMRAM state save area may be read and changed by the SMI handler, with the changed values restored to the processor registers by the RSM instruction. Some register images are read-only, and must not be modified (modifying these registers will result in unpredictable behavior). The values stored in the areas marked reserved may change in future CPUs. An SMM handler should not rely on any values stored in an area that is marked as reserved.

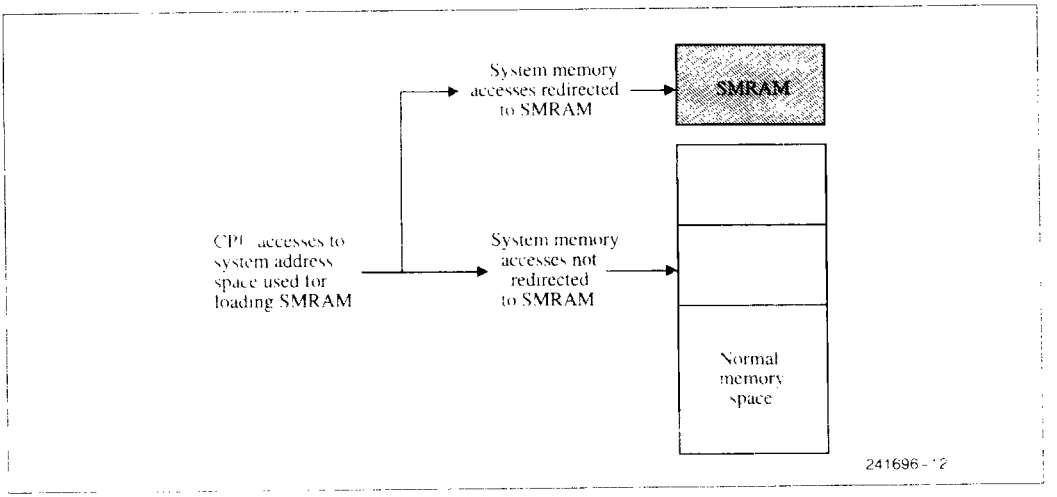


Figure 4-5. Redirecting System Memory Addresses to SMRAM

Table 4-1. SMRAM State Save Map

Register Offset	Register	Writeable?
7FFC	CR0	NO
7FF8	CR3	NO
7FF4	EFLAGS	YES
7FF0	EIP	YES
7FEC	EDI	YES
7EE8	ESI	YES
7FE4	EBP	YES
7FE0	ESP	YES
7FDC	EBX	YES
7FD8	EDX	YES
7FD4	ECX	YES
7FD0	EAX	YES
7FCC	DR6	NO
7FC8	DR7	NO
7FC4	TR*	NO
7FC0	LDTR*	NO
7FBC	GS*	NO
7FB8	FS*	NO
7FB4	DS*	NO
7FB0	SS*	NO
7FAC	CS*	NO
7FA8	ES*	NO
7FA7–7F98	Reserved	NO
7F94	IDT Base	NO
7F93–7F8C	Reserved	NO
7F88	GDT Base	NO
7F87–7F04	Reserved	NO
7F02	Auto HALT Restart Slot (Word)	YES
7F00	I/O Trap Restart Slot (Word)	YES
7EFC	SMM Revision Identifier (Dword)	NO
7EF8	SMBASE Slot (Dword)	YES
7EF7–7E00	Reserved	NO

NOTES:
*Upper two bytes are reserved
Modifying a value that is marked as not writeable will result in unpredictable behavior.
Words are stored in 2 consecutive bytes in memory with the low-order byte at the lowest address and the high-order byte in the high address.

The following registers are saved and restored (in areas of the state save that are marked reserved) but are not visible to the system software programmer:

CR1, CR2 and CR4, hidden descriptor registers for CS, DS, ES, FS, GS.

If an SMI request is issued for the purpose of powering down the CPU, the values of all reserved locations in the SMM state save must be saved to non-volatile memory.

The following registers are not automatically saved and restored by SMI# and RSM:

DR5–DR0, TR7–TR3, FPU registers: STn, FCS, FSW, tag word, FP instruction pointer, FP op code, and operand pointer.

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

4.3.4 EXIT FROM SMM

The **RSM** instruction, is only available to the SMI handler. The op code of the instruction is 0FAAH. Execution of this instruction while the CPU is executing outside of SMM will cause an invalid op-code error. The last instruction of the SMI handler will be the RSM instruction.

The RSM instruction restores the state save image from SMRAM back to the CPU, then returns control back to the interrupted program execution. There are three SMM features that can be enabled by writing to control "slots" in the SMRAM state save area.

Auto HALT Restart. It is possible for the SMI request to interrupt the HALT state. The 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.

I/O Trap Restart. If the SMI# interrupt was generated on an I/O access to a powered-down device, the SMI handler can tell the RSM instruction to re-execute that I/O instruction by setting the I/O Trap Restart slot.

SMBASE Relocation. The system can relocate the SMRAM by setting the SMBASE Relocation slot in the state save area. The RSM instruction will set the SMBASE in the processor based on the value in the SMBASE relocation slot. The SMBASE must be 32K aligned.

For further details on these SMM features, see Section 4.5.

If the processor detects invalid state information, it enters the shutdown state; this happens only in the following situations:

- The value stored in the SMBASE slot is not a 32 Kbyte-aligned address.
- A reserved bit of CR4 is set to 1.
- A combination of bits in CR0 is illegal; namely, (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 special bus cycle to indicate it has entered shutdown mode.

4.4 System Management Mode Programming Model

4.4.1 ENTERING SYSTEM MANAGEMENT MODE

SMM is one of the major operating modes, on a level with Protected mode, Real address mode or virtual-86 mode. Figure 4-6 shows how the processor can enter SMM from any of the three modes and then return.

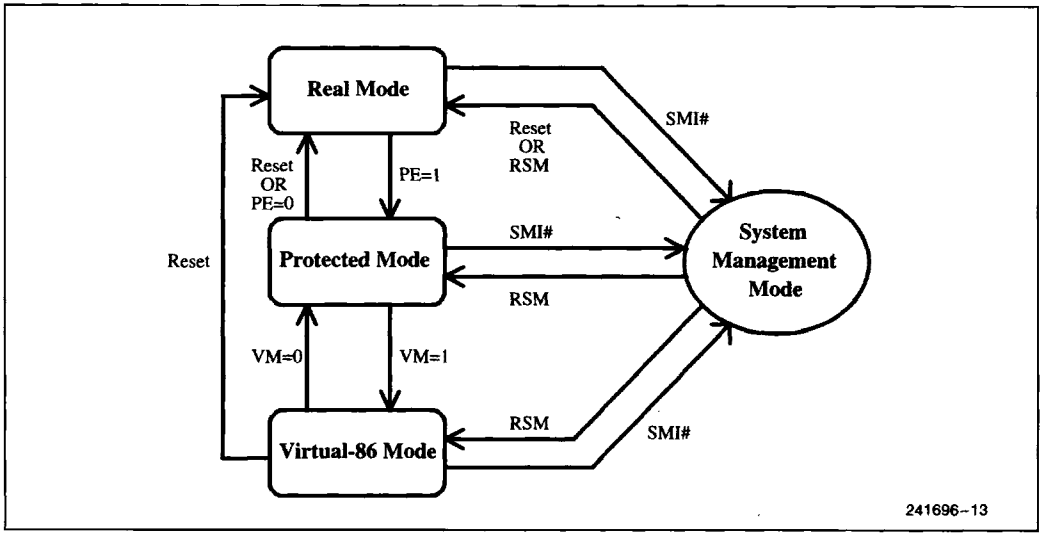


Figure 4-6. Transition to and from System Management Mode

The external signal SMI# causes the processor to switch to SMM. The RSM instruction exits SMM. SMM is transparent to applications programs and operating systems because of the following:

- The only way to enter SMM is via a type of non-maskable 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 in a part of SMRAM called the SMM context save space.
- All interrupts normally handled by the operating system or by applications are disabled upon entry into SMM.
- A special instruction, RSM, restores processor registers from the SMM context save space and returns control to the interrupted program.

SMM is similar to Real address mode in that there are no privilege levels or address mapping. An SMM program can execute all I/O and other system instructions and can address up to four Gbytes of memory.

4.4.2 PROCESSOR ENVIRONMENT

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

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 SMI# processing occurs, not a debug exception or external interrupt. Subsequent SMI# requests are not acknowledged while the processor is in SMM. The first SMI# interrupt request that occurs while the processor is in SMM is latched, and serviced when the processor exits SMM with the RSM instruction. Only one SMI# will be latched by the CPU while it is in SMM.

When the CPU invokes SMM, the CPU core registers are initialized as follows:

Table 4-2. SMM Initial CPU Core Register Settings

Register	Contents
General purpose registers	Unpredictable
EFLAGS	00000002H
EIP	00008000H
CS selector	3000H
CS base	SMM Base (default 30000H)
DS, ES, FS, GS, SS Selectors	0000H
DS, ES, FS, GS, SS Bases	000000000H
DS, ES, FS, GS, SS Limits	0FFFFFFFFH
CR0	Bits 0,2,3 & 31 cleared (PE, EM, TS & PG); others unmodified
DR6	Unpredictable
DR7	00000000H

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

1. Real mode style address calculation
2. Gbyte limit checking
3. IF flag is cleared
4. NMI is disabled
5. TF flag in EFLAGS is cleared; single step traps are disabled
6. DR7 is cleared; debug traps are disabled
7. The RSM instruction no longer generates an invalid op code error
8. Default 16-bit op code, register and stack use.

All bus arbitration (HOLD, AHOLD, BOFF#) inputs and bus sizing (BS8#, BS16#) inputs operate normally while the CPU is in SMM.

4.4.3 EXECUTING SYSTEM MANAGEMENT MODE HANDLER

The processor begins execution of the SMM handler at offset 8000H in the CS segment. The CS Base is initially 30000H. The CS Base can be changed, however, *using the SMM Base relocation feature.*

When the SMM 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 SMI handler should not use floating point unit instructions until the FPU is properly detected (within the 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. The limits and attributes are not modified.

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.

4.4.3.1 Exceptions and Interrupts within 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 is done to prevent a debug application from accidentally breaking into an SMM handler. This is necessary because the SMM handler operates from a distinct address space (SMRAM) and hence the debug trap will not represent the normal system memory space.

If an SMM handler wishes to use the debug trap feature of the processor to debug SMM handler code, it must first ensure that an SMM compliant debug handler is available. The SMM handler must also ensure DR0–DR3 is saved to be restored later. The debug registers DR0–DR3 and DR7 must then be initialized with the appropriate values.

If the processor wishes 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 upon 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 upon entry to the SMM handler. If an NMI request occurs during the SMM handler, it is latched and serviced after the processor exits SMM. Only one NMI request will be latched during the SMM 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 SMM handler requires the use of NMI interrupts, it should invoke a dummy interrupt service routine for the purpose of executing an IRET instruction. Once an IRET instruction is executed, NMI interrupt requests are serviced in the same "real mode" manner in which they are handled outside of SMM.

4.5 SMM Features

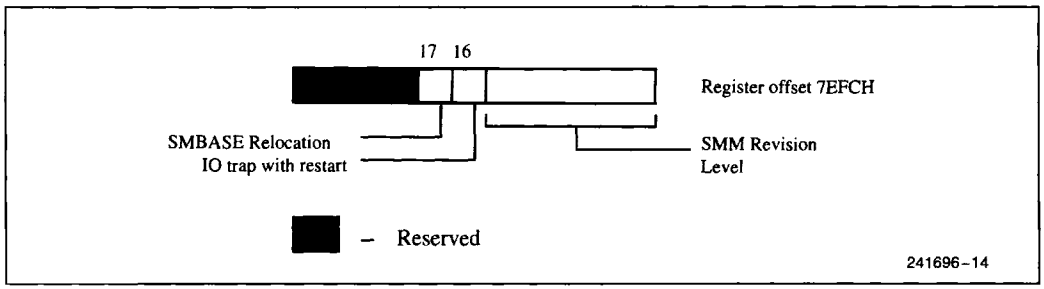
4.5.1 SMM REVISION IDENTIFIER

The SMM revision identifier is used to indicate the version of SMM and the SMM extensions that are supported by the processor. The SMM revision identifier is written during SMM entry and can be examined in SMRAM space at Register Offset 7EFCH. The lower word of the SMM revision identifier refers to the version of the base SMM architecture. The upper word of the SMM revision identifier refers to the extensions available.

Bit 16 of the SMM revision identifier is used to indicate to the SMM handler that this processor supports the SMM I/O trap extension. If this bit is high, then this processor supports the SMM I/O trap extension. If this bit is low, then this processor does not support I/O trapping using the I/O trap slot mechanism.

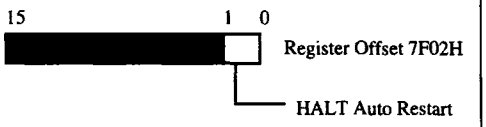
Bit 17 of this slot indicates whether the processor supports relocation of the SMM jump vector and the SMRAM base address.

All members of the SL Enhanced Intel486 CPU family support both the I/O Trap Restart and the SMBASE relocation features.



Bits	Value	Comments
16	0	Processor does not support I/O Trap Restart
16	1	Processor supports I/O Trap Restart
17	0	Processor does not support SMBASE Relocation
17	1	Processor supports SMBASE Relocation

4.5.2 AUTO HALT RESTART

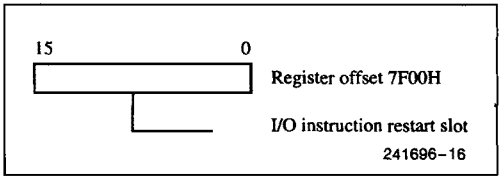


Value of Bit 0 at Entry	Value of Bit 0 at Exit	Comments
0	0	Returns to next instruction in interrupted program.
0	1	Unpredictable
1	0	Returns to next instruction after HALT
1	1	Returns to HALT state

The Auto HALT Restart slot at register offset (word location) 7F02H in SMRAM indicates to the SMM handler that the SMI interrupted the CPU during a HALT state (bit 0 of slot 7F02H is set to 1 if the previous instruction was a HALT). If the SMI did not interrupt the CPU in a HALT state, then the SMI micro code will set bit 0 of the Auto HALT Restart slot to a value of 0. If the previous instruction was a HALT, the SMM handler can choose to either set or reset bit 0. If this bit is set to 1, the RSM micro code execution will force the processor to re-enter the HALT state. If this bit is set to 0 when the RSM instruction is executed, the processor will continue execution with the instruction just after the interrupted HALT instruction. Note that 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.

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.

4.5.3 I/O INSTRUCTION RESTART



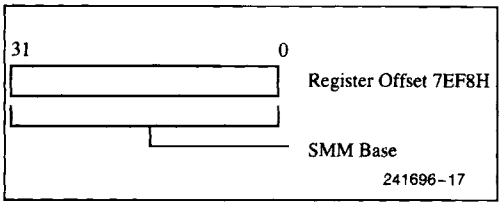
Value at Entry	Value at Exit	Comments
00H	00H	Do not restart trapped I/O instruction
00H	0FFH	Restart trapped I/O instruction

The I/O instruction restart slot (register offset 7F00H in SMRAM) gives the SMM handler the option of causing the RSM instruction to automatically re-execute the interrupted I/O instruction. When the RSM instruction is executed, if the I/O instruction restart slot contains the value 0FFH, then the CPU will automatically re-execute the I/O instruction that the SMI trapped. If the I/O instruction restart slot contains the value 00H when the RSM instruction is executed, then the CPU will 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 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 SMI# that originated on a non-I/O instruction boundary.

If the system executes back-to-back SMI requests, the second SMM handler must not set the I/O instruction restart slot (see Section 4.6.6).

4.5.4 SMM BASE RELOCATION

The SL Enhanced Intel486 CPU family provides a new control register, SMBASE. The address space used as SMRAM can be modified by changing the SMBASE register before exiting an SMI handler routine. SMBASE can be changed to any 32K aligned value (values that are not 32K aligned will cause the CPU to enter the shutdown state when executing the RSM instruction). SMBASE is set to the default value of 30000H on RESET, but is not changed on SRESET. If the SMBASE register is changed during an SMM handler, all following SMI# requests will initiate a state save at the new SMBASE.



The SMBASE slot in the SMM state save area is a feature used to indicate and change the SMI jump vector location and the 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. During the execution of the RSM instruction, the CPU will read this slot and initialize the CPU to use the new SMBASE during the next SMI. During an SMI, the CPU will do its context save to the new SMRAM area pointed to by the SMBASE, store the current SMBASE in the SMM Base slot (offset 7EF8H), and then start execution of the new jump vector based on the current SMBASE.

The SMBASE must be a 32 Kbyte aligned, 32-bit integer that indicates a base address for the SMRAM context save area and the SMI jump vector. For example when the processor first powers up, the minimum SMRAM area is from 38000H–3FFFFH. The default SMBASE is 30000H. Hence the starting address of the jump vector is calculated by:

SMBASE + 8000H

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

SMM Base + [8000H + 7FFFH]

Hence when this feature is enabled the SMRAM register map is addressed according to the above formulae.

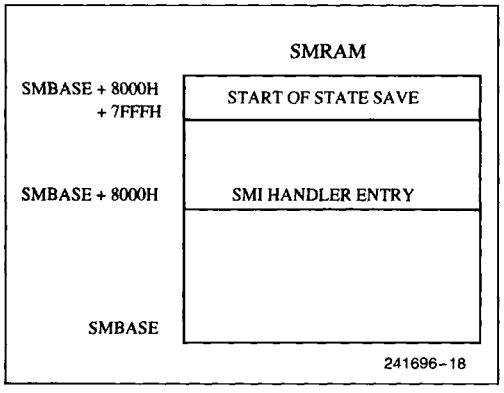


Figure 4-7. SMRAM Usage

To change the SMRAM base address and SMI jump vector location, the SMM handler should modify the SMBASE slot. Upon executing an RSM instruction, the processor will read the SMBASE slot and store it internally. Upon recognition of the next SMI request, the processor will use the new SMBASE slot for the SMRAM dump and SMI jump vector.

If the modified SMBASE slot does not contain a 32 KByte-aligned value, the RSM micro code will cause the CPU to enter the shutdown state.

4.6 SMM - System Design Considerations

4.6.1 SMRAM INTERFACE

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

1. A provision should be made to allow for initialization of SMRAM space during system boot up. This initialization of SMRAM space must happen before the first occurrence of an SMI# interrupt. Initializing the SMRAM space must include installation of an SMM handler, and may include installation of related data structures necessary for particular SMM applications. The memory controller providing the interface to the SMRAM should provide a means for the initialization code to manually open the SMRAM space.
2. A minimum initial SMRAM address space of 38000H-3FFFFH should be decoded by the memory controller.
3. Alternate bus masters (such as DMA controllers) should not be allowed to access SMRAM space. Only the CPU, either through SMI or during initialization, should be allowed access to SMRAM.
4. In order to implement a zero-volt suspend function, the system must have access to all of normal system memory from within an SMM handler routine. If the SMRAM is going to overlay normal system memory, there must be a method of accessing any system memory that is located underneath SMRAM (see Section A.2.3).

There are two potential schemes for locating the SMRAM, either overlaid to an address space on top of normal system memory, or placed in a distinct address space. See Figure 4-8. When SMRAM is overlaid on the top of normal system memory, the CPU output signal SMIACT# must be used to distinguish SMRAM from main system memory. Additionally, both the CPU internal cache and any second level caches must be empty before the first read of an SMM handler routine and before the first read of normal memory following an SMM handler routine. This is done by flushing the caches, and is required to maintain cache coherency. When the default SMRAM location

is used, SMRAM is overlaid on top of system main memory (at 38000H through 3FFFFH).

If SMRAM is located in its own distinct memory space, which can be completely decoded with only the CPU address signals, it is said to be non-overlaid. In this case, there are no new requirements for maintaining cache coherency.

4.6.2 CACHE FLUSHES

The CPU does not unconditionally flush its cache before entering SMM (this option is left to the system designer). If SMRAM is shadowed in a memory area that is visible to the application or operating system, it is necessary for the system to empty both the CPU cache and any second level cache before entering and after leaving SMM. That is, if SMRAM is in the same physical address location as the normal cacheable memory space, then an SMM read may HIT the cache which would contain normal memory space code/data. Likewise the normal read cycles after SMM may HIT the cache which may contain SMM code/data. In this case the cache should be empty before the first memory read cycle during SMM and before the first normal cycle after exiting SMM.

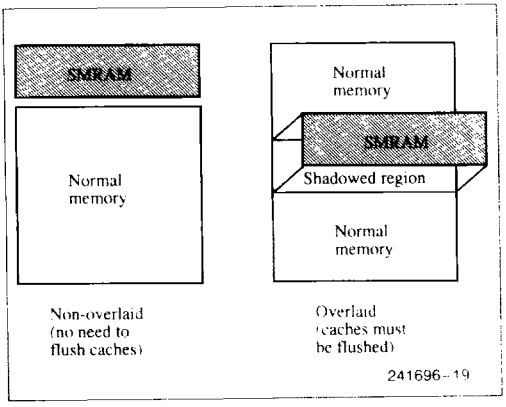


Figure 4-8. SMRAM Location

The FLUSH# and KEN# signals can be used to ensure cache coherency when switching between normal and SMM modes. Cache flushing during SMM entry is accomplished by asserting the FLUSH# pin when SMIACT# is driven active. Cache flushing during SMM exit is accomplished by asserting the FLUSH# pin after the SMIACT# pin is de-asserted (within 1 CLK). To guarantee this behavior, the constraints on setup and hold timings on the interaction of FLUSH# and SMIACT# as specified for a processor should be followed.

If the SMRAM area is overlaid over normal memory and if the system designer does not want to flush the caches upon leaving SMM then references to the SMRAM area should not be cached. It is the obligation of the system designer to ensure that the KEN# pin is sampled inactive during all references to the SMRAM area.

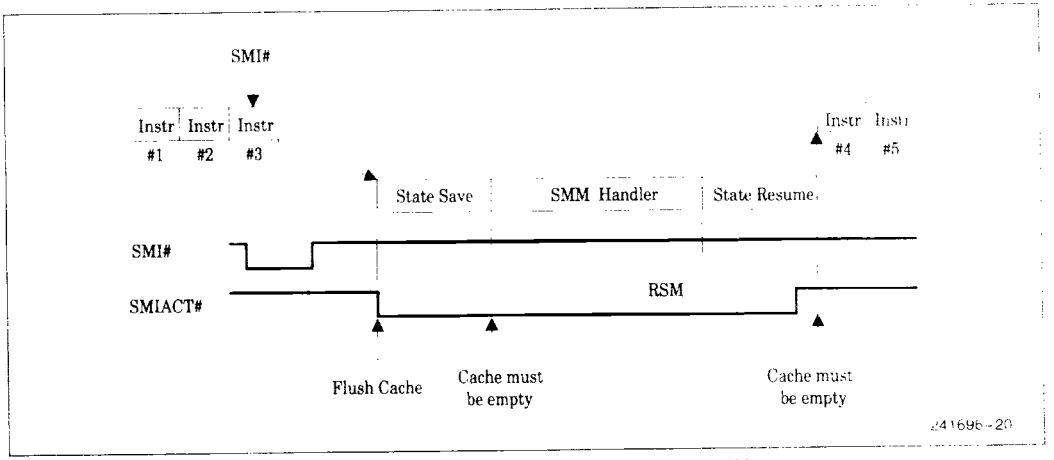


Figure 4-9. FLUSH# Mechanism During SMM

Two methods are suggested as shown in Figures 4-10 and 4-11.

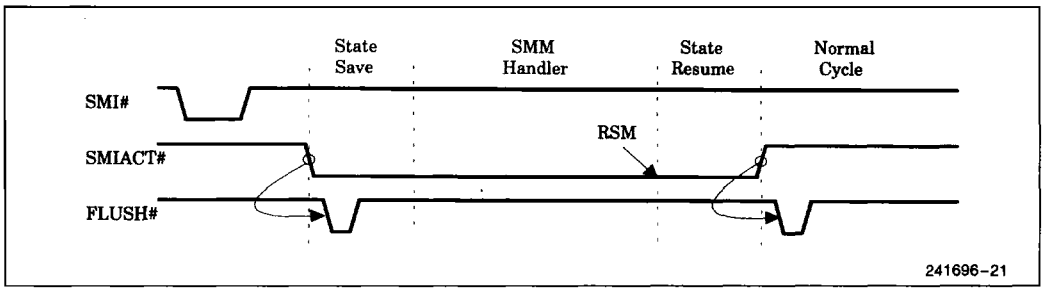


Figure 4-10. Cached SMM

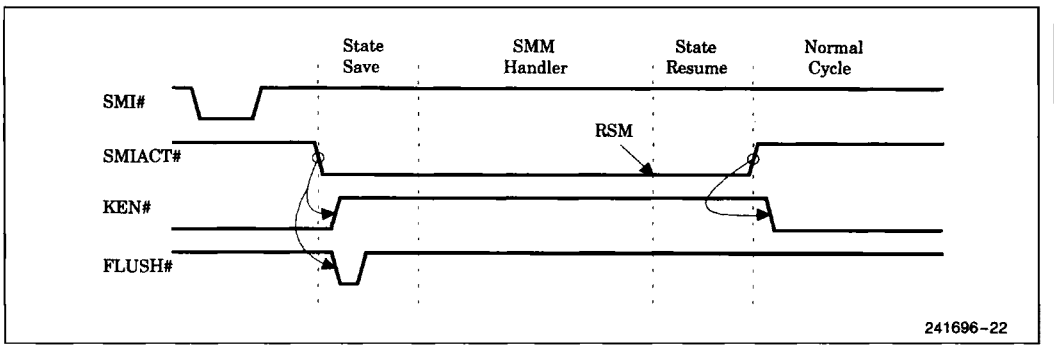


Figure 4-11. Non-cached SMM

4.6.3 A20M# PIN AND SMBASE RELOCATION

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 wrap around functionality of the original IBM PC. The A20M# pin on SL Enhanced Intel486 CPUs provides this function. When A20M# is active, all external bus cycles will drive A20 low, and all internal cache accesses will be performed with A20 low.

The A20M# pin is recognized while the CPU is in SMM. The functionality of the A20M# input must be recognized in two instances:

1. If the SMM handler needs to access system memory space above 1 Mbyte (for example, when saving memory to disk for a zero-volt suspend), the 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 A20M# is active upon entering SMM, the CPU will attempt to access SMRAM at the relocated address, but with A20 low. This could cause the system to crash, since there would be no valid SMM interrupt handler at the accessed location.

In order to account for these two situations, the system designer must ensure that A20M# is deasserted on entry to SMM. 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 A20M# whenever SMIACT# is active.

4.6.4 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 is used to switch the CPU from Protected mode to Real mode. If we look at the relative interrupt priorities, RESET has higher priority than 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 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 de-assertion of SMIACT# and the completion of the RSM micro code. 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 SMIACT# has been driven inactive.
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. In order to maintain software transparency, the system logic must latch any SRESET signals which 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.

4.6.5 SMM AND SECOND LEVEL WRITE BUFFERS

Before an SL Enhanced Intel486 CPU enters SMM, it empties its internal write buffers. This is necessary so that the data in the write buffers is written to normal memory space, not SMM space. Once the CPU is ready to begin writing an SMM state save to SMRAM, it asserts the SMIACT# signal. 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 SMIACT# along with the address for each word in the write buffers.

4.6.6 NESTED SMI# AND I/O RESTART

Special care must be taken when executing an SMM handler for the purpose of restarting an I/O instruction. When the CPU executes a RSM instruction with the I/O restart slot set, the restored EIP is modified to point to the instruction immediately preceding the SMI# request, so that the I/O instruction can be re-executed. If a new SMI# request is received while the CPU is executing an SMM handler, the CPU will service this 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 SMM handler, the RSM micro code will decrement the restored EIP again. EIP now points to an address different from the originally interrupted instruction, and the CPU will begin execution of the interrupted application code at an incorrect entry point.

To prevent this from occurring, the SMM handler routine must not set the I/O restart slot during the second of two consecutive SMM handlers.

4.7 SMM - Software Considerations

4.7.1 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 four Gbyte logical address space.

With operand-size override prefixes, the SMM handler can use jumps, calls, and returns, to transfer control to any location within the four Gbyte space. Note, however, the following restrictions:

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

4.7.2 EXCEPTION HANDLING

Upon entry into SMM, external interrupts that require handlers are disabled (the IF bit in the 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 SMM 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 SMM handler can prevent internal exceptions. When new exception vectors are initialized, internal exceptions can be serviced. The following are the restrictions:

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 (64 KBytes).
3. If exceptions or interrupts are allowed to occur, only the low order 16 bits of the return address (EIP) 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 could be to perform software adjustment of the return address on the stack).

4. The SMBASE Relocation feature affects the way the CPU will return from an interrupt or exception during an SMI handler.

4.7.3 HALT DURING SMM

HALT should not be executed during SMM, unless interrupts have been enabled (see section 4.7.2). Interrupts are disabled in SMM and INTR, NMI, and SMI# are the only events that take the CPU out of HALT.

4.7.4 RELOCATING SMRAM TO AN ADDRESS ABOVE ONE MEGABYTE

Within SMM (or Real mode), the segment base registers can only be updated 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 four bits to determine the segment base address). If SMRAM is relocated to an address above one megabyte, 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 16M, the DS and ES registers are still initialized to 0000 0000H. We can still access data in SMRAM by using 32-bit displacement registers:

```
mov esi,00FFxxxxH ;64K segment immediately
                  ;below 16M
mov ax,ds:[esi]
```

5.0 RESET AND INITIALIZATION

5.1 RESET

The RESET input must be used at power-up to initialize the CPU. The Reset input forces the CPU to begin execution at a known state. The microprocessor cannot begin execution of instructions until at least 1 ms after V_{CC} and CLK have reached their proper D.C. and A.C. specifications. The RESET pin should remain active during this time to ensure proper microprocessor operation. However, for warm boot-ups RESET should remain active for at least 15 CLK periods. RESET is active HIGH. RESET is asynchronous but must meet setup and hold times t20 and t21 for recognition in any specific clock. The RESET signal is the same as the standard Intel486 CPU RESET signal with the following exceptions.

RESET has a special function of resetting SMBASE to the default value of 30000H. If SMBASE relocation is not used, the RESET signal can be used as the only reset.

The microprocessor will be placed in the Power Down mode if UP# is sampled active at the falling edge of RESET.

5.2 SRESET

The SRESET (Soft RESET) input, has the same functions as RESET, but does not change the SMBASE, and UP# is not sampled on the falling edge of SRESET. If SMBASE relocation is used by the system, the soft resets should be handled using the SRESET input. The SRESET signal should not be used for the cold boot-up power-on reset.

The SRESET input pin is provided to save the status of SMBASE during i286 CPU-compatible mode change. SRESET leaves the status of the on-chip FPU and SMBASE intact while resetting other units including the on-chip cache. For compatibility with future generation Intel486 CPUs, the system should not rely on flushing the on-chip cache through the SRESET input pin. The FLUSH# input pin is provided for this purpose and should be used to flush the on-chip cache.

SRESET should not be used to initiate test modes (this behavior is subject to change in future versions).

5.3 Pin State During Reset

While in reset, the SL Enhanced Intel486 microprocessor bus is in the state shown in Figure 5-1, if the HOLD, AHOLD and BOFF# requests are inactive. Note that the address (A31-A2, BE3#-BE0#) and cycle definition (M/IO#, D/C#, W/R#) pins are undefined from the time reset is asserted up to the start of the first bus cycle. All defined pins (except FERR#) assume known values at the beginning of the first bus cycle. The first bus cycle is always a code fetch to address 0FFFFFFF0H. FERR# reflects the state of the ES (error summary status) bit in the floating point unit status word. The ES bit is initialized whenever the floating point unit state is initialized.

Figures 5-1 and 5-2 show the bus state for the SL Enhanced Intel486 DX and SX CPUs when UP# is not asserted and for the SL Enhanced Intel486 SX CPU when UP# is asserted.

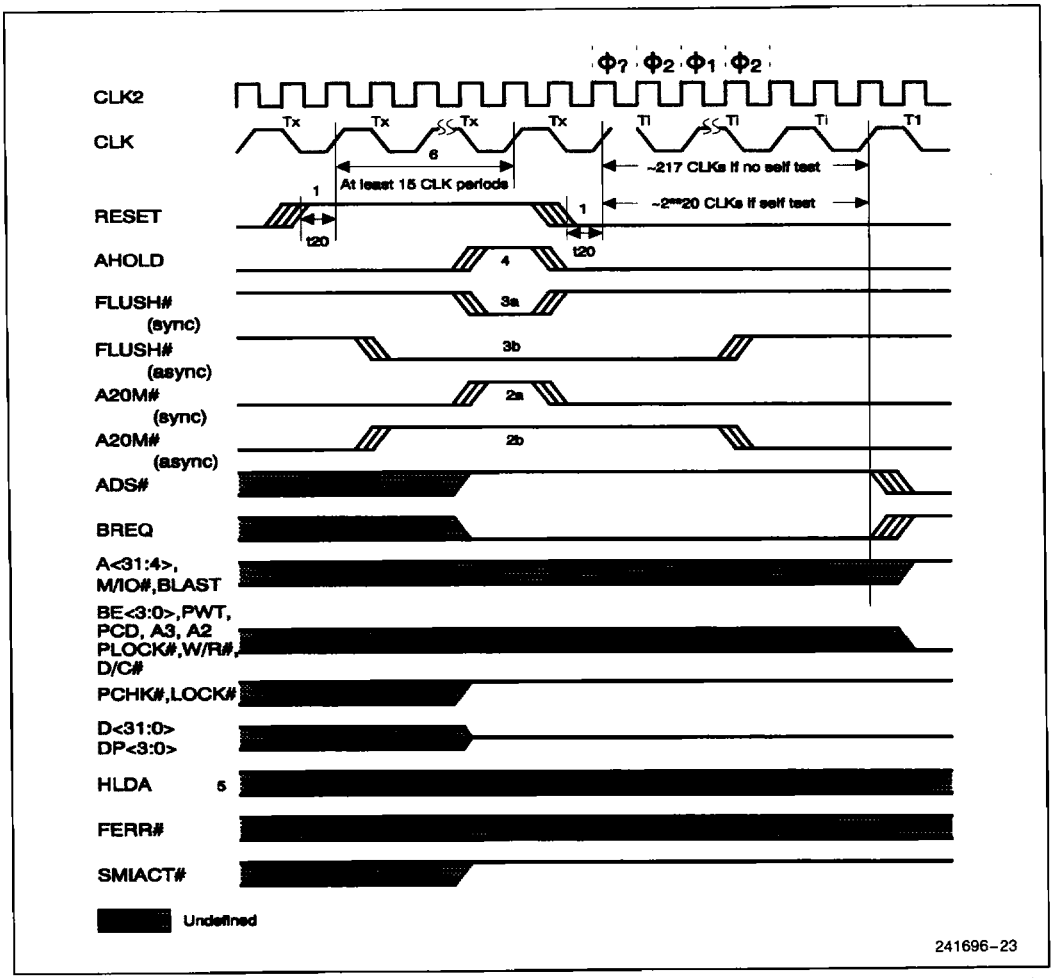
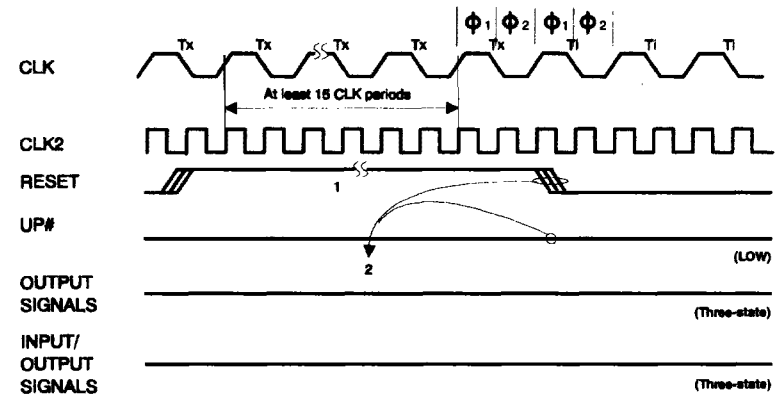


Figure 5-1. SL Enhanced Intel486™ DX CPU or SL Enhanced Intel486 SX CPU without UP# Asserted During SRESET or RESET

NOTES:
1. RESET is an asynchronous input. t_{20} must be met only to guarantee recognition on a specific clock edge.
2a. When A20M# is driven synchronously, it must be driven high (inactive) for the CLK edge prior to and the falling edge of RESET to ensure proper operation. A20M# setup and hold times must be met.
2b. When A20M# is driven asynchronously, it must be driven high (inactive) for two CLKs prior to and two CLKs after the falling edge of RESET to ensure proper operation.
3a. When FLUSH# is driven synchronously, it should be driven low (active) for the CLK edge prior to the falling edge of RESET to invoke the Three-State Output Test Mode. All outputs are guaranteed three-stated within 10 CLKs of RESET being deasserted. FLUSH# setup and hold times must be met.
3b. When FLUSH# is driven asynchronously, it must be driven low (active) for two CLKs prior to and two CLKs after the falling edge of RESET to invoke the Three-State Output Test Mode. All outputs are guaranteed three-stated within 10 CLKs of RESET being deasserted.
4. AHOLD should be driven high (active) for the CLK edge prior to the falling edge of RESET to invoke the Build-In-Self-Test (BIST). AHOLD setup and hold times must be met.
5. Hold is recognized normally during RESET.
6. 15 CLKs RESET pulse width for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.



241696-24

1. 15 CLKs reset pulse width for warm reset. Power-up resets require reset to be asserted for at least 1 ms after V_{CC} and CLK are stable.
2. When UP# is active, the falling edge of RESET will disable all the internal input buffers except UP#. RESET and the internal input buffers of CLK will also be disabled.

Figure 5-2. SL Enhanced Intel486™ SX CPU During SRESET or RESET with UP# Asserted

5.4 CPU Identification Codes

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

Table 5-1. CPU ID Codes

SL Enhanced Intel486™ CPU	Component ID (DH)	Revision ID (DL)
Intel486 SX CPU	04	2x
Intel486 DX CPU	04	1x
Intel486 DX2 CPU	04	3x

5.4.1 CPU ID INSTRUCTION

The SL Enhanced Intel486 microprocessor family implements a new 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. The actual state of the EFLAGS.ID bit is irrelevant and provides no significance to the hardware. This bit is reset to zero upon device reset (RESET or SRESET) for compatibility with existing Intel486 processor designs.

Table 5-2. CPUID Instruction Description

OP CODE	Instruction	CPU Core Clocks	EAX Input Value	Description
0F A2	CPUID	14	1	CPU Identification
		9	0 or >1	Intel string/ null registers

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] ← 1
  EBX[31:0] ← 756E6547
  ECX[31:0] ← 6C65746E
  EDX[31:0] ← 49656E69

The values in EBX, ECX, and EDX indicate an Intel microprocessor. When taken in the proper order, they decode to the string "GenuineIntel".

- When the parameter passed in EAX is one, the register values returned are as follows:

  EAX[3:0] ← Stepping ID*
  EAX[7:4] ← Model
  i486 DX CPU = 1
  i486 SX CPU = 2
  i486 DX2 CPU = 3
  EAX[11:8] ← Family
  i486 CPU = 4
  EAX[15:12] ← 0000
  EAX[31:16] ← RESERVED
  EBX[31:0] ← 00000000
  ECX[31:0] ← 00000000
  EDX[31:0] ← 00000001
  bit 0 = FPU

*Please contact Intel 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, the register values returned upon instruction execution are:

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

Flags Affected: No flags are affected.

Exceptions: None.

6.0 ELECTRICAL AND MECHANICAL SPECIFICATIONS

6.1 D.C. Specifications

6.1.1 3.3V D.C. CHARACTERISTICS

Table 6-1. 3.3V D.C. Specifications

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

Symbol	Parameter	Min	Typ	Max	Unit	Test Cond.
V_{IL}	Input LOW Voltage	−0.3		+0.8	V	
V_{IH}	Input HIGH Voltage	2.0		V_{CC}+0.3	V	Note 4
V_{IHC}	Input HIGH Voltage of CLK, CLK2	V_{CC}−0.6		V_{CC}+0.3	V	
V_{OL}	Output LOW Voltage I_{OL} = 2.0 mA I_{OL} = 100 uA			 0.4 0.2	 V V	
V_{OH}	Output HIGH Voltage I_{OH} = −2.0 mA I_{OH} = −100 uA	 2.4 V_{CC}−0.2			 V V	
I_{CCU}	UP# Active Supply Current		15	35	mA	Note 5
I_{LI}	Input Leakage Current	−15		15	µA	Note 1
I_{IH}	Input Leakage Current			200	µA	Note 2
I_{IL}	Input Leakage Current			−400	µA	Note 3
I_{LO}	Output Leakage Current	−15		15	µA	
C_{IN}	Input Capacitance			10	pF	Note 6
C_{OUT}	Output or I/O Capacitance			10	pF	Note 6
C_{CLK}	CLK Capacitance			6	pF	Note 6

NOTES:
1. This parameter is for inputs without pull-ups or pull downs and $0V < V_{IN} < V_{CC}$.
2. This parameter is for inputs with Pull downs and V_{IH} = 2.4V.
3. This parameter is for inputs with pull-ups and V_{IL} = 0.4V.
4. All inputs except CLK, CLK2.
5. When the CPU is in Stop Grant state, the I_{CCU} of the host CPU is less than 2 mA.
6. F_C = 1 MHz; Not 100% tested.

Table 6-2. 3.3V Preliminary I_{CC} Values for 1X Clock SL Enhanced Intel486 SX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
3.3V	I_{CC} Active	25 MHz	250 mA	315 mA
		33 MHz	300 mA	385 mA
	I_{CC} Stop Grant	25 MHz	20 mA	40 mA
		33 MHz	25 mA	50 mA
	I_{CC} Stop Clock(1)	0 MHz	100 μA	1 mA

Table 6-3. 3.3V Preliminary I_{CC} Values for 1X Clock SL Enhanced Intel486 DX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
3.3V	I_{CC} Active	33 MHz	330 mA	415 mA
	I_{CC} Stop Grant	33 MHz	25 mA	50 mA
	I_{CC} Stop Clock(1)	0 MHz	100 μA	1 mA

Table 6-4. 3.3V Preliminary I_{CC} Values for SL Enhanced Intel486 DX2 CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
3.3V	I_{CC} Active	40 MHz 50 MHz	375 mA 460 mA	450 mA 550 mA
	I_{CC} Stop Grant	40 MHz 50 MHz	20 mA 23 mA	40 mA 50 mA
	I_{CC} Stop Clock(1)	0 MHz	100 μA	1 mA

NOTE:
1. The V_{IH} and V_{IL} levels must be equal to V_{CC} and 0V, respectively, in order to meet the I_{CC} Stop Clock specifications.

6.1.2 5V D.C. CHARACTERISTICS

Table 6-5. 5V D.C. Specifications

Functional operating range: V_{CC} = 5V ± 5%; T_{CASE} = 0°C to +85°C

Symbol	Parameter	Min	Typical	Max	Unit	Test Condition
V_{IL}	Input LOW Voltage	−0.3		+0.8	V	
V_{IH}	Input HIGH Voltage	2.0		V_{CC}+0.3	V	
V_{OL}	Output LOW Voltage			0.45	V	Note 1
V_{OH}	Output HIGH Voltage	2.4			V	Note 2
I_{CCU}	UP# Active Supply Current		25	50	mA	Note 6
I_{LI}	Input Leakage Current	−15		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		15	μA	
C_{IN}	Input Capacitance PGA PQFP			 20 10	 pF pF	Note 7
C_{OUT}	Output or I/O Capacitance PGA PQFP			 20 10	 pF pF	Note 7
C_{CLK}	CLK Capacitance PGA PQFP			 20 6	 pF pF	Note 7

NOTES:
1. This parameter is measured at: Address, Data, BEn 4.0 mA; Definition, Control 5.0 mA
2. This parameter is measured at: Address, Data, BEn −1.0 mA; Definition, Control −0.9 mA
3. This parameter is for inputs without pull-ups or pull-downs and $0V \leq V_{IN} \leq V_{CC}$.
4. This parameter is for inputs with pull-downs and V_{IH} = 2.4V.
5. This parameter is for inputs with pull-ups and V_{IL} = 0.45V.
6. When the CPU is in Stop Grant state, the I_{CCU} of the host CPU is less than 2 mA.
7. F_C = 1 MHz; Not 100% tested.

Table 6-6. 5V Preliminary I_{CC} Values for 1X Clock SL Enhanced Intel486 SX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
5V	I_{CC} Active	25 MHz	430 mA	560 mA
		33 MHz	590 mA	685 mA
	I_{CC} Stop Grant	25 MHz	35 mA	65 mA
		33 MHz	40 mA	80 mA
	I_{CC} Stop Clock(1)	0 MHz	200 μA	2 mA

Table 6-7. 5V Preliminary I_{CC} Values for 1X Clock SL Enhanced Intel486 DX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
5V	I_{CC} Active	33 MHz	500 mA	630 mA
		50 MHz	775 mA	950 mA
	I_{CC} Stop Grant	33 MHz	40 mA	80 mA
		50 MHz	55 mA	100 mA
	I_{CC} Stop Clock(1)	0 mHz	200 μA	2 mA

Table 6-8. 5V Preliminary I_{CC} Values for SL Enhanced Intel486 DX2 CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
5V	I_{CC} Active	50 MHz	775 mA	950 mA
		66 MHz	975 mA	1200 mA
	I_{CC} Stop Grant	50 MHz	35 mA	70 mA
		66 MHz	45 mA	90 mA
	I_{CC} Stop Clock(1)	0 MHz	200 μA	2 mA

NOTE:
1. The V_{IH} and V_{IL} levels must be equal to V_{CC} and 0V, respectively, in order to meet the I_{CC} Stop Clock specifications.

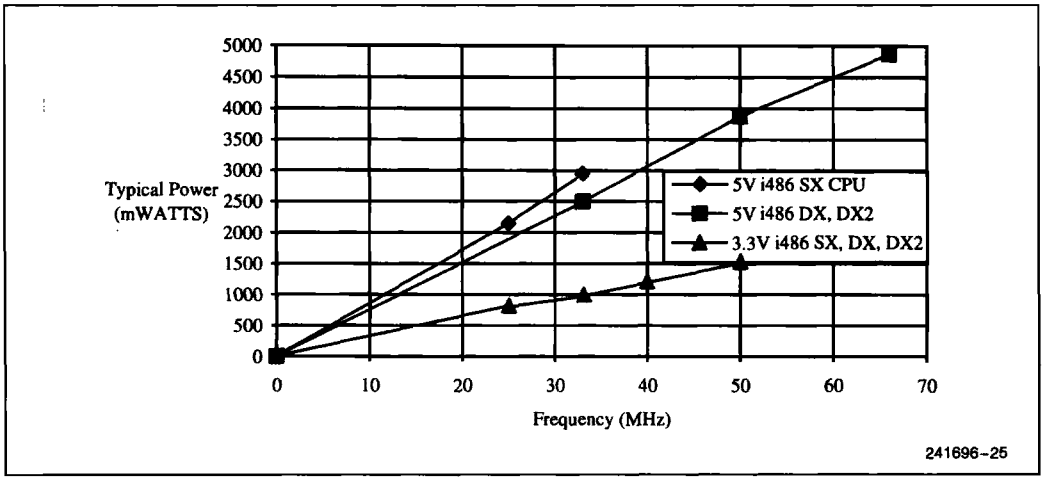


Figure 6-1. Frequency vs Power (Typ) in 1X CLK Mode for SL Enhanced Intel486 SX, DX and DX2 CPUs

6.2 A.C. Specifications for 1X CLK Option

6.2.1 5V A.C. CHARACTERISTICS

The A.C. specifications given in the tables of this section consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the rising edge of the input system clock (CLK) unless otherwise specified.

Table 6-9. 5V A.C. Characteristics (1X Clock) Frequency = 25 and 33 MHz (Preliminary Information)
Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Min	Max	Unit	Notes
	Frequency	8	**25**	8	**33**	MHz	Note 1
t_1	CLK Period	40	125	30	125	ns	
t_{1a}	CLK Period Stability		0.1		0.1	%	Adjacent clocks
t_2	CLK High Time	14		11		ns	at 2V
t_3	CLK Low Time	14		11		ns	at 0.8V
t_4	CLK Fall Time		4		3	ns	2V to 0.8V
t_5	CLK Rise Time		4		3	ns	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA, SMIACT#, **FERR#** * Valid Delay	3	19	3	16	ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		28		20	ns	Note 2
t_8	PCHK# Valid Delay	3	24	3	22	ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	24	3	20	ns	
t_9	BLAST#, PLOCK# Float Delay		28		20	ns	Note 2
t_{10}	D0–D31, DP0–DP3 Write Data Valid Delay	3	20	3	18	ns	

Table 6-9. 5V A.C. Characteristics (1X Clock) Frequency = 25 and 33 MHz (Preliminary Information) (Continued)

Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Min	Max	Unit	Notes
t_{11}	D0–D31, DP0–DP3 Write Data Float Delay		28		20	ns	Note 2
t_{12}	EADS# Setup Time	8		5		ns	
t_{13}	EADS# Hold Time	3		3		ns	
t_{14}	KEN#, BS16#, BS8# Setup Time	8		5		ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		3		ns	
t_{16}	RDY#, BRDY# Setup Time	8		5		ns	
t_{17}	RDY#, BRDY# Hold Time	3		3		ns	
t_{18}	HOLD, AHOLD Setup Time	10		6		ns	
t_{18a}	BOFF# Setup Time	10		8		ns	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		3		ns	
t_{20}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#*** Setup Time	10		5		ns	
t_{21}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#*** Hold ime	3		3		ns	
t_{22}	D0–D31, DP0–DP3, A4–A31 Read Setup Time	5		5		ns	
t_{23}	D0–D31, DP0–DP3, A4–A31 Read Hold Time	3		3		ns	

NOTES:
*Present only in the SL Enhanced Intel486 DX and DX2 CPUs.
1. 0 MHz operation is guaranteed when the STPCLK# and STOP GRANT bus cycle protocol is used.
2. Not 100% tested, guaranteed by design characterization.

Table 6-10. 5V A.C. Characteristics (1X Clock) Frequency = 50 MHz (Intel486 DX CPU) (Preliminary Information)

Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C; C_L = 0 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Unit	Notes
	Frequency	16	**50**	MHz	Note 1
t_1	CLK Period	20	62.5	ns	
t_{1a}	CLK Period Stability		0.1	%	Adjacent clocks
t_2	CLK High Time	7		ns	at 2V
t_3	CLK Low Time	7		ns	at 0.8V
t_4	CLK Fall Time		2	ns	2V to 0.8V
t_5	CLK Rise Time		2	ns	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA, SMIACT#, FERR# Valid Delay	3	12	ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		18	ns	Note 2
t_8	PCHK# Valid Delay	3	14	ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	12	ns	
t_9	BLAST#, PLOCK# Float Delay		18	ns	Note 2
t_{10}	D0–D31, DP0–DP3 Write Data Valid Delay	3	12	ns	
t_{11}	D0–D31, DP0–DP3 Write Data Float Delay		18	ns	Note 2
t_{12}	EADS# Setup Time	5		ns	
t_{13}	EADS# Hold Time	2		ns	
t_{14}	KEN#, BS16#, BS8# Setup Time	5		ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	2		ns	
t_{16}	RDY#, BRDY# Setup Time	5		ns	
t_{17}	RDY#, BRDY# Hold Time	2		ns	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	5		ns	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	2		ns	
t_{20}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, IGNNE# Setup Time	5		ns	
t_{21}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, IGNNE# Hold Time	2		ns	
t_{22}	D0–D31, DP0–DP3, A4–A31 Read Setup Time	4		ns	
t_{23}	D0–D31, DP0–DP3, A4–A31 Read Hold Time	2		ns	

NOTES:
1. 0 MHz operation is guaranteed when the STPCLK# and STOP GRANT bus cycle protocol is used.
2. Not 100% tested, guaranteed by design characterization.

The following specifications are different for existing Intel486 DX2 CPU products. A system board that will support all of the SL Enhanced Intel486 CPU products should be designed to the worst-case specifications of the 25 and 33 MHz local bus timings.

Table 6-11. A.C. Characteristics (1X Clock) Frequency = 50 and 66 MHz (Intel486 DX2 CPU) (Preliminary Information)

Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

		DX2-50		DX2-66		
Symbol	**Parameter**	**Min**	**Max**	**Min**	**Max**	**Unit**
	Frequency		**50**		**66**	MHz
	CLK Frequency	8	**25**	8	**33**	MHz
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA, SMIACT#, **FERR#** Valid Delay				14	ns
t_8	PCHK# Valid Delay				14	ns
t_{8a}	BLAST#, PLOCK# Valid Delay				14	ns
t_{10}	D0–D31, DP0–DP3 Write Data Valid Delay				14	ns
t_{18}	HOLD, AHOLD Setup Time	8				ns
t_{18a}	BOFF# Setup Time	8		7		ns
t_{20}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#** Setup Time	8				ns

NOTES:
1. 0 MHz operation is guaranteed when the STPCLK# and STOP GRANT bus cycle protocol is used.
2. Not 100% tested, guaranteed by design characterization.

6.2.2 3.3V A.C. CHARACTERISTICS

**Table 6-12. 3.3V A.C. Characteristics (1X Clock Option)
Frequency = 20 (Intel486 DX2-40 CPU), 25, (Intel486
DX2-50 and SX-25 CPUs) and 33 MHz. (Preliminary Information)**

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

Symbol	Parameter	Min	Max	Min	Max	Min	Max	Figure	Unit	Notes
	Frequency	8	**20**	8	**25**	8	**33**		MHz	Note 1
t_1	CLK Period	50	125	40	125	30	125		ns	
t_{1a}	CLK Period Stability		0.1		0.1		0.1		%	Adjacent clocks
t_2	CLK High Time	16		14		11			ns	at 2V
t_3	CLK Low Time	16		14		11			ns	at 0.8V
t_4	CLK Fall Time		6		4		3		ns	2V to 0.8V
t_5	CLK Rise Time		6		4		3		ns	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA, SMIACT#, **FERR#*** Valid Delay	3	23	3	19	3	16		ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		37		28		20		ns	Note 2
t_8	PCHK# Valid Delay	3	28	3	24	3	22		ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	28	3	24	3	20		ns	
t_9	BLAST#, PLOCK# Float Delay		37		28		20		ns	Note 2
t_{10}	D0–D31, DP0–DP3 Write Data Valid Delay	3	26	3	20	3	19		ns	
t_{11}	D0–D31, DP0–DP3 Write DAta Float Delay		37		28		20		ns	Note 2
t_{12}	EADS# Setup Time	10		8		6			ns	
t_{13}	EADS# Hold Time	3		3		3			ns	
t_{14}	KEN#, BS16#, BS8# Setup Time	10		8		6			ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		3		3			ns	
t_{16}	RDY#, BRDY# Setup Time	10		8		6			ns	
t_{17}	RDY#, BRDY# Hold Time	3		3		3			ns	
t_{18}	HOLD, AHOLD Setup Time	12		10		6			ns	
t_{18a}	BOFF# Setup Time	12		10		9			ns	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		3		3			ns	

2

Table 6-12. 3.3V A.C. Characteristics (1X Clock Option)
Frequency = 20, (Intel486 DX2-40 CPU), 25, (Intel486 DX2-50 and SX-25 CPUs) and 33 MHz. (Preliminary Information)

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

Symbol	Parameter	Min	Max	Min	Max	Min	Max	Figure	Unit	Notes
t_{20}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#*** Setup Time	12		10		6			ns	
t_{21}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#*** Hold Time	3		3		3			ns	
t_{22}	D0–D31, DP0–DP3, A4–A31 Read Setup Time	6		6		6			ns	
t_{23}	D0–D31, DP0–DP3, A4–A31 Read Hold Time	3		3		3			ns	

NOTES:
* Present only in the SL Enhanced Intel486 DX and DX2 CPUs.
1. 0 MHz operation is guaranteed when the STPCLK# and STOP GRANT bus cycle protocol is used.
2. Not 100% tested, guaranteed by design characterization.

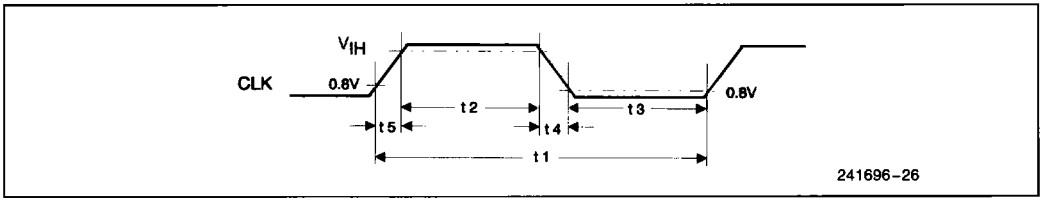


Figure 6-2. CLK Waveforms

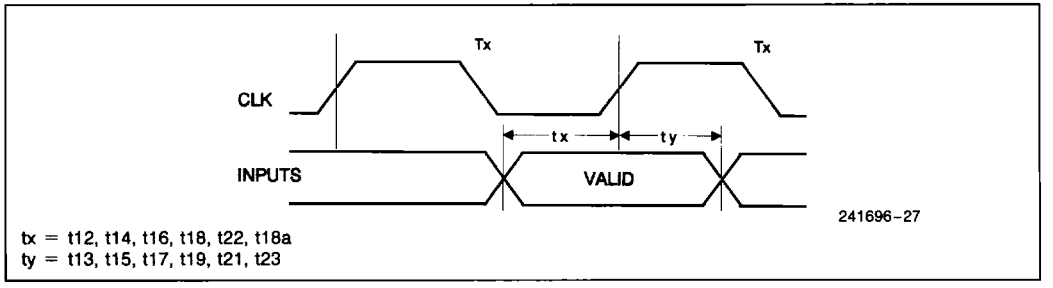


Figure 6-3. Input Setup and Hold Timing

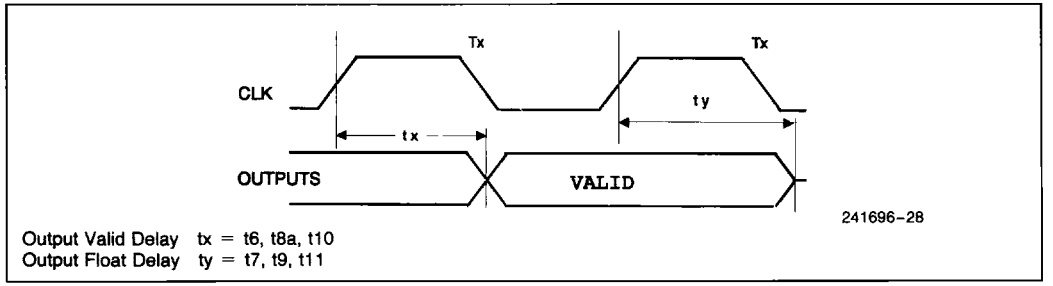


Figure 6-4. Output Valid and Float Delay Timing

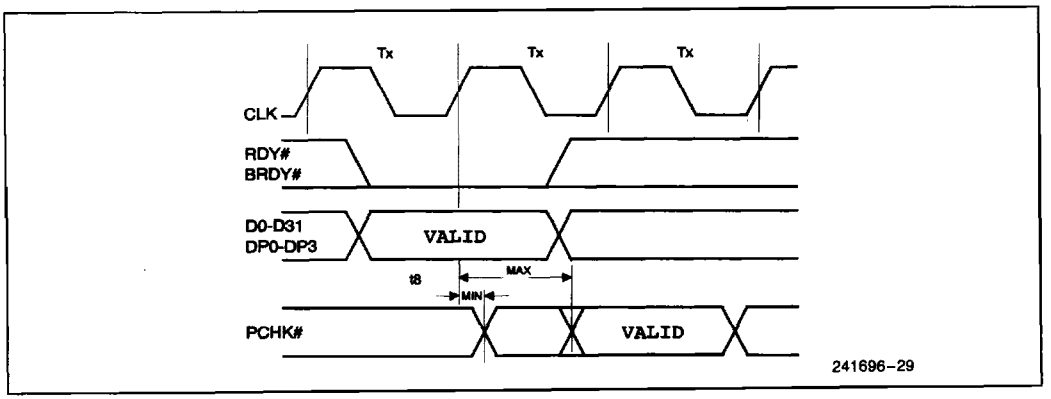


Figure 6-5. PCHK# Valid Delay Timing

2

6.3 Mechanical Data

6.3.1 PACKAGE MECHANICAL SPECIFICATIONS FOR THE 208 LEAD SQFP PACKAGE

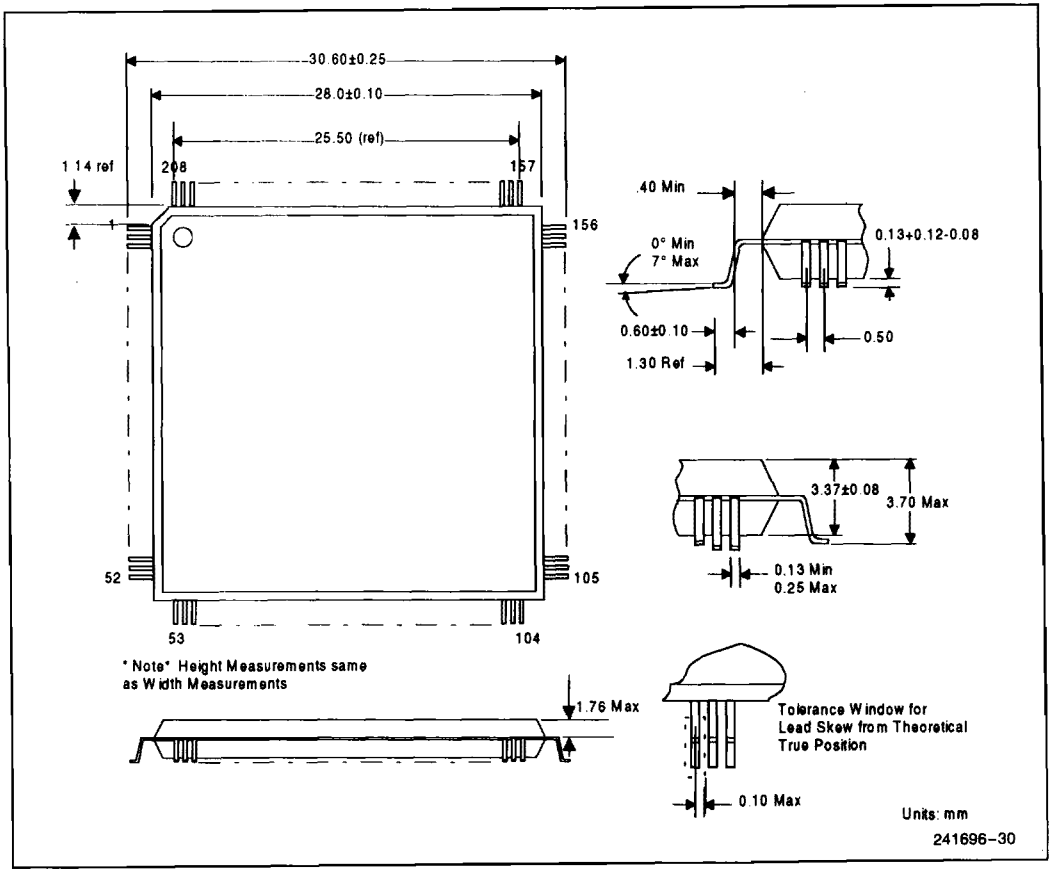


Figure 6-6. 208 Lead SQFP Package Dimensions

6.3.2 PACKAGE THERMAL SPECIFICATIONS

The SL Enhanced Intel486 microprocessors are specified for operation when T_C (the case temperature) is within the range of 0 °C − 85 °C. T_C may be measured in any environment to determine whether the Intel486 microprocessor is within the specified operating range. The case temperature, with and without heat sink should be measured using a 0.005″ diameter (AWG#36) thermocouple with a 90° angle adhesive bond at the center of the package top surface, opposite the pins. Figures 6-7 and 6-8 illustrate this methodology.

The ambient temperature can be calculated from θ_{JC} and θ_{JA} from the following equations.

$T_J = T_C + P * \theta_{JC}$
$T_A = T_J - P * \theta_{JA}$
$T_C = T_A + P * [\theta_{JA} - \theta_{JC}]$

Where T_J, T_A, T_C = Junction, Ambient and Case Temperature, respectively. 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 the following tables for a variety of packages and operating frequencies.

Refer to the OverDrive Processor socket section for OverDrive processor θ_{JA} and θ_{JC} values.

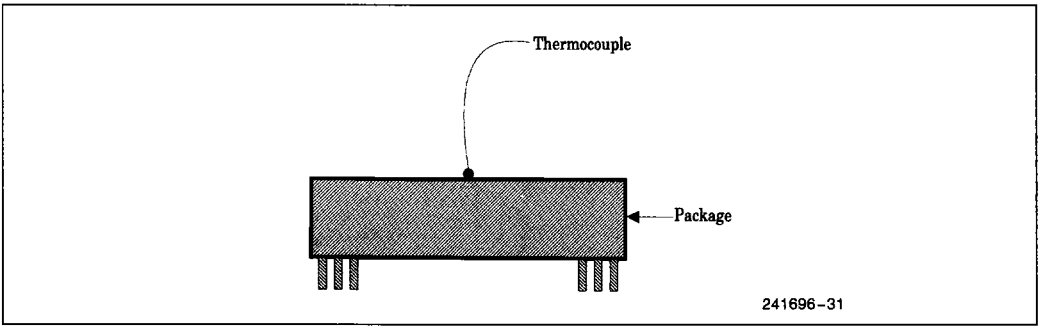


Figure 6-7. Case Temperature Measurement without Heat Sink (0.005″ Dia. Thermocouple on the Center of the Package Top Surface with a 90° Angle Adhesive Bond)

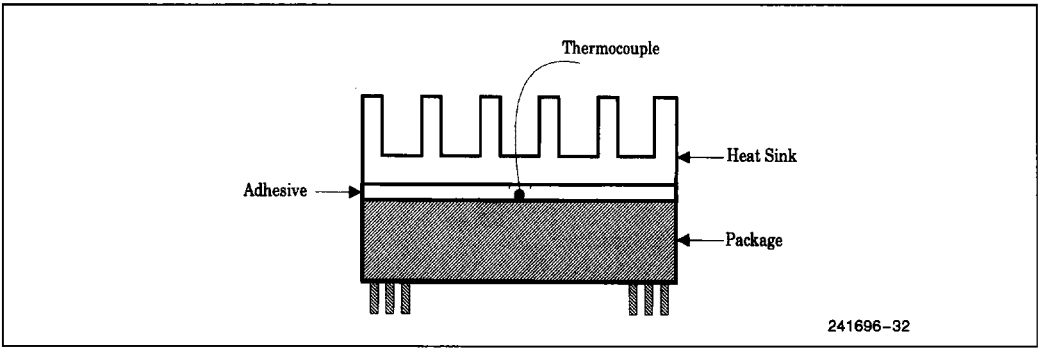


Figure 6-8. Case Temperature Measurement with Heat Sink (0.005″ Dia. Thermocouple on the Center of the Package Top Surface with a 90° Angle Adhesive Bond through a Hole Drilled at the Heat Sink Base)

Table 6-13. Thermal Resistance (°C/W) θ_{JC} and θ_{JA} for the SL Enhanced Intel486™ CPUs (for PGA)

	θ_{JC}	θ_{JA} vs. Airflow - ft/min. (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
With Heat Sink	1.5	13	8.0	6.0	5.0	4.5	4.25
Without Heat Sink	1.5	17	14.5	12.5	11.0	10.0	9.5

* 0.350" high omnidirectional heat sink.

Table 6-14. Thermal Resistance (°C/W) θ_{JC} and θ_{JA} for the SL Enhanced Intel486™ CPUs (for PQFP)

	θ_{JC}	θ_{JA} vs. Airflow - ft/min. (m/sec)			
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
With Heat Sink	3.5	17.0	10.5	8.5	8.0
Without Heat Sink	3.5	20.5	16.5	14.0	12.5

*0.350" high omnidirectional heat sink.

2

Table 6-15(a). Thermal Resistance (°C/W) θ_{JA} for the SL Enhanced Intel486™ CPUs (for SQFP)

	θ_{JA} vs. Airflow - ft/min. (m/sec)			
	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
Intel486 SX CPU Without Heat Sink	36.0	27.5	25.0	22.5
Intel486 DX CPU Without Heat Sink	25.0	17.5	15.0	13.0
Intel486 DX2 CPU Without Heat Sink	24.0	17.0	15.0	13.0

Table 6-15(b). Thermal Resistance (°C/W) θ_{JC} for the SL Enhanced Intel486 CPUs (for SQFP)

	θ_{JC} vs. Airflow - ft/min. (m/sec)			
	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
Intel486 SX CPU	4.0	7.5	8.0	8.5
Intel486 DX and DX2 CPUs	3.5	6.0	6.0	6.0

The following tables show maximum ambient temperatures of SL Enhanced Intel486 CPUs for each package and operating frequency. The maximum ambient temperatures listed below are not valid for the OverDrive Processor.

Table 6-16. Maximum Tambient for the 5V, 168-pin PGA SL Enhanced Intel486™ DX and DX2 CPUs

	Freq.	Airflow - ft/min. (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
Tambient °C With Heat Sink	33	49	65	71	74	76	76
	50	30	54	64	68	71	72
	66	16	46	58	64	67	69
Tambient °C Without Heat Sink	33	36	44	50	55	58	60
	50	11	23	33	40	45	47
	66	−8	7	19	28	34	37

Table 6-17. Maximum Tambient for the 5V, 168-pin PGA SL Enhanced Intel486™ SX CPU

	Freq.	Airflow - ft/min. (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
Tambient °C With Heat Sink	25	53	67	72	75	77	77
	33	46	63	70	73	75	76
Tambient °C Without Heat Sink	25	42	49	54	58	61	63
	33	32	40	47	52	56	58

Table 6-18. Maximum Tambient for the 5V, 196-lead PQFP SL Enhanced Intel486™ DX CPU

	Freq.	Airflow - ft/min. (m/sec)			
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
Tambient °C With Heat Sink	33	42	63	69	71
Tambient °C Without Heat Sink	33	31	44	52	57

Table 6-19. Maximum Tambient for the 5V, 196-lead PQFP SL Enhanced Intel486™ SX CPU

	Freq.	Airflow - ft/min. (m/sec)			
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
Tambient °C With Heat Sink	25	47	65	71	72
	33	39	61	68	70
Tambient °C Without Heat Sink	25	37	49	56	60
	33	27	40	49	54

Table 6-20. Maximum Tambient for the 3.3V, 208-lead SQFP SL Enhanced Intel486™ SX CPU

	Freq.	Airflow - ft/min. (m/sec)			
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
Tambient °C Without Heat Sink Intel486 SX CPU	25	51.1	64.0	67.0	70.0
	33	44.5	59.5	63.5	67.0

Table 6-21. Maximum Tambient for the 3.3V, 208-lead SQFP SL Enhanced Intel486™ DX CPU

	Freq.	Airflow - ft/min. (m/sec)			
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
Tambient °C Without Heat Sink Intel486 DX CPU	33	55.5	69.0	72.5	75.5

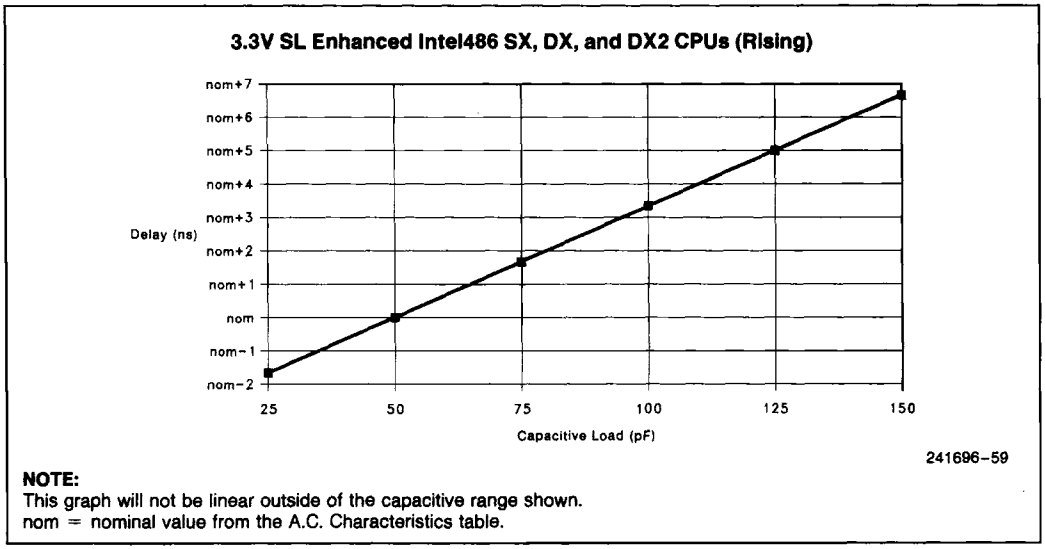
Table 6-22. Maximum Tambient for the 3.3V, 208-lead SQFP SL Enhanced Intel486™ DX2 CPU

	Freq.	Airflow - ft/min. (m/sec)			
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
Tambient °C Without Heat Sink Intel486 DX2 CPU	40	54.5	68.5	71.5	74.5
	50	48.0	65.0	68.5	72.0

For further details about thermal and mechanical package specifications and methodologies, refer to the 1994 Packaging Handbook (order number 240800).

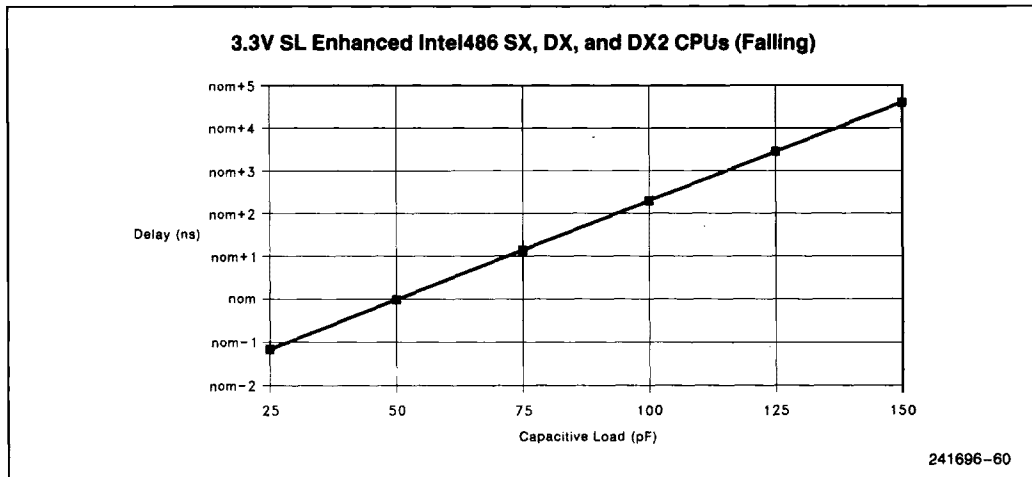
6.4 Capacitive Derating Information

The capacitive derating curves illustrate output delay versus capacitive load for 3.3V and 5V SL Enhanced Intel486 microprocessors. The derating curves show the delays for the rising and falling edges under worst-case conditions. Figures 6-9 and 6-10 apply to all 3.3V SL Enhanced Intel486 SX, DX, and DX2 CPUs. Figures 6-11 and 6-12 apply to 5V SL Enhanced Intel486 DX and DX2 CPUs. Figures 6-13 and 6-14 apply to 5V SL Enhanced Intel486 SX CPUs. The figures apply to all frequencies specified for each corresponding product.



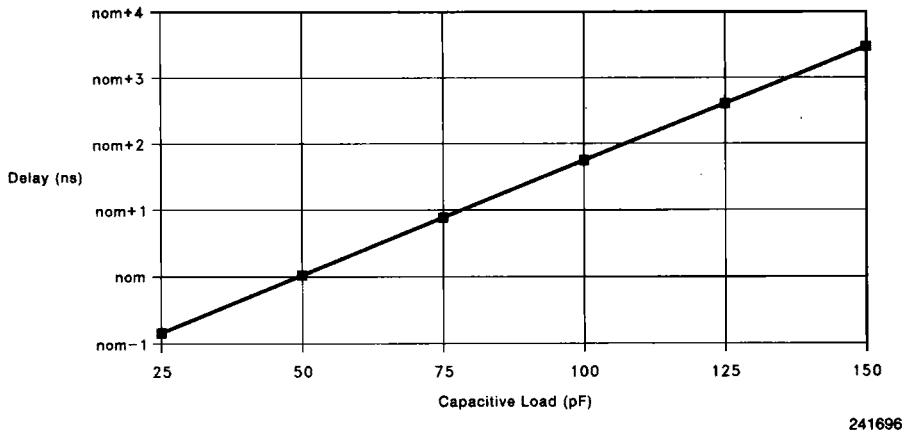
NOTE:
This graph will not be linear outside of the capacitive range shown.
nom = nominal value from the A.C. Characteristics table.

Figure 6-9. Typical Loading Delay versus Load Capacitance under Worst-Case Conditions for a Low-to-High Transition



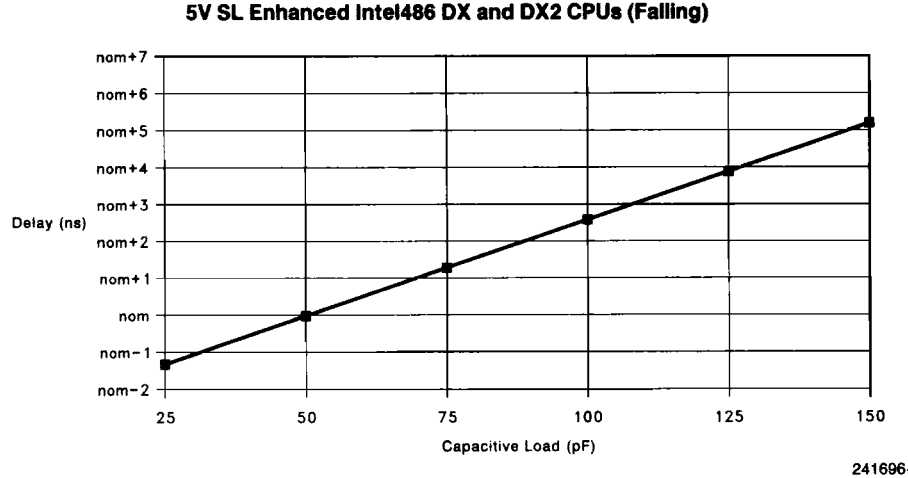
NOTE:
This graph will not be linear outside of the capacitive range shown.
nom = nominal value from the A.C. Characteristics table.

Figure 6-10. Typical Loading Delay versus Load Capacitance under Worst-Case Conditions for a High-to-Low Transition



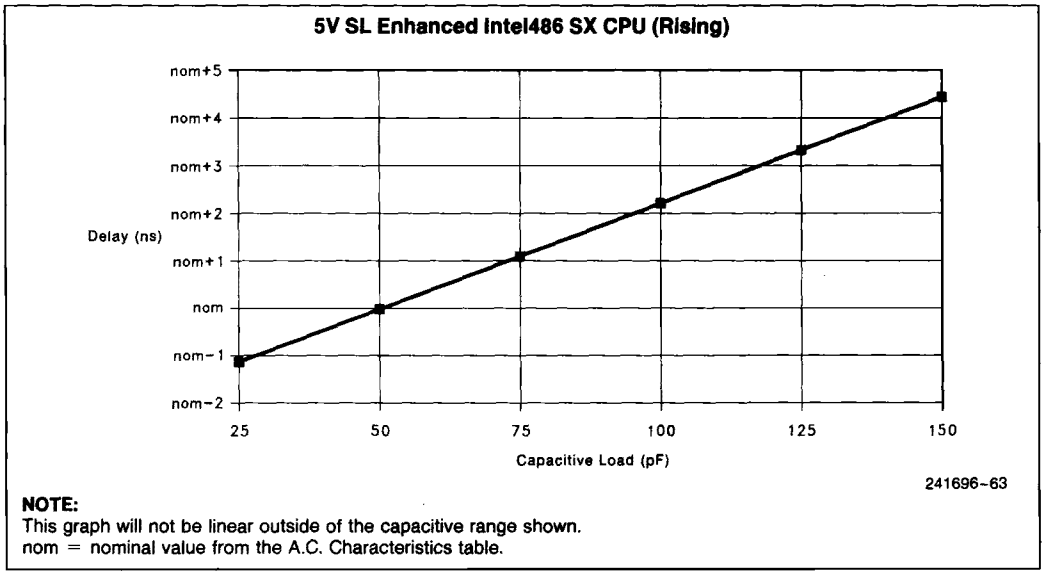
NOTE:
This graph will not be linear outside of the capacitive range shown.
nom = nominal value from the A.C. Characteristics table.

Figure 6-11. Typical Loading Delay versus Load Capacitance under Worst-Case Conditons for a Low-to-High Transition



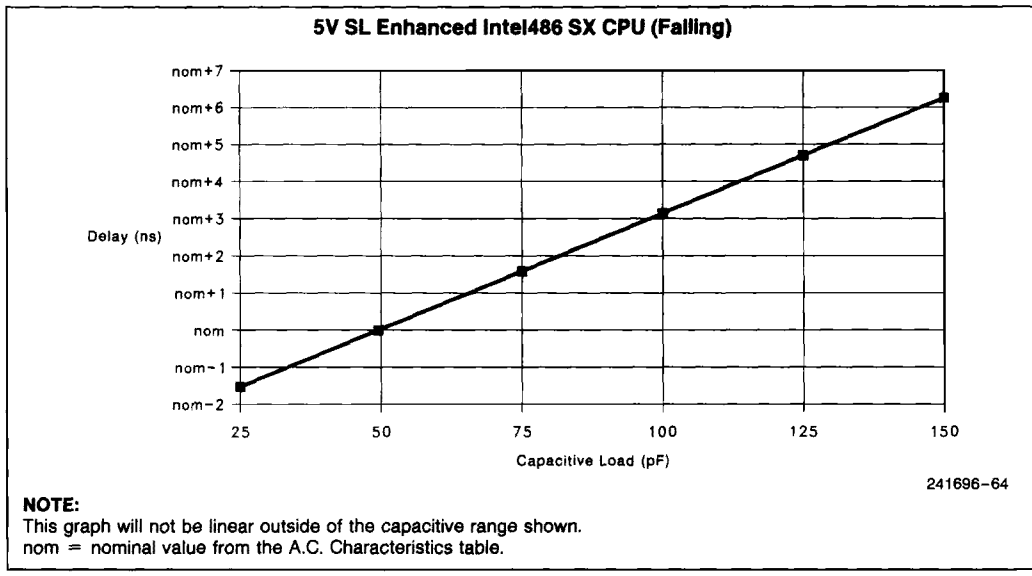
NOTE:
This graph will not be linear outside of the capacitive range shown.
nom = nominal value from the A.C. Characteristics table.

Figure 6-12. Typical Loading Delay versus Load Capacitance under Worst-Case Conditions for a High-to-Low Transition



NOTE:
This graph will not be linear outside of the capacitive range shown.
nom = nominal value from the A.C. Characteristics table.

Figure 6-13. Typical Loading Delay versus Load Capacitance under Worst-Case Conditions for a Low-to-High Transition



NOTE:
This graph will not be linear outside of the capacitive range shown.
nom = nominal value from the A.C. Characteristics table.

Figure 6-14. Typical Loading Delay versus Load Capacitance under Worst-Case Conditions for a High-to-Low Transition

7.0 2X CLOCK MODE

The SL Enhanced Intel486 CPU offers 2X clock mode for systems that rely on dynamic frequency scaling for CPU power management. This product is not intended for the desktop computer. This 2X clock CPU differs from the 1X clock CPU in the following ways:

Pin Assignment/Function: The 2X clock product has a CLK2 input, rather than the 1X clock product's CLK input. The CLK2 input must be synchronized to the system phase using the falling edge of RESET. (For reference, the pinout change from the existing Low Power Intel486 DX and SX CPUs is also shown. The CLKSEL pin is not used on the SL Enhanced Intel486 CPUs, as it is on the existing Low Power Intel486 DX and SX CPUs.)

Clock Control: The CLK2 input can be changed dynamically. The Stop Clock interrupt is handled in a different manner.

AC Specifications: In general, the AC specifications for the 2X clock device will have slightly longer setups, holds, and maximum valid delays. This is consistent with the existing Low Power Intel486 DX and SX CPU products.

Upgrades: There are no end user upgrade products planned for the 2X clock mode product. The UP# function is still provided for use by system designers that offer SX to DX CPU upgrade cards.

This section will explain the differences between the CPU with the 2X clock mode and the CPU with the 1X clock mode.

7.1 Pin Assignments

The SL Enhanced Intel486 CPU with the 2X clock option is available in the 196 Lead PQFP package. The pinout is identical to the SL Enhanced Intel486 CPU with the 1X clock option with the exception of the name of the clock input. The 1X clock input is called CLK and the 2X clock input is called CLK2.

Table 7-1 shows the change between the existing products and new products.

Table 7-1. Pinout Differences for the 2X Clock Mode (Low Power) CPUs (PQFP Package)

Pin	Low Power Intel486 SX CPU	SL Enhanced Intel486 SX CPU	Low Power Intel486 DX CPU	SL Enhanced Intel486 DX CPU
75	NC	STPCLK#	NC	STPCLK#
77	NC	NC	IGNNE#	IGNNE#
81	NC	NC	FERR#	FERR#
85	NC	SMI#	N C	SMI#
92	NC	SMIACT#	NC	SMIACT#
94	NC	SRESET	NC	SRESET
127	CLKSEL	NC	CLKSEL	NC

7.2 Quick Pin Reference

Table 7-2. Pin Descriptions

Symbol	Type	Name and Function
CLK2	I	**CLK2** provides the fundamental timing for the CPU. Both of the internal timing phases, phase-1 (ph1) and phase-2 (ph2), are provided by the external CLK2 input. All external timing parameters are specified with respect to the phase-1 rising edge of CLK2. For the 2X clock mode the CLK frequency is twice the frequency of the CPU.
RESET	I	The **RESET** input forces the CPU to begin execution at a known state. The CPU cannot begin execution of instructions until at least 1 ms after V_{CC} and CLK2 have reached their proper AC and DC specifications. However, for soft resets, RESET should remain active for at least 30 CLK2 periods (equal to 15 internal CPU CLK). The RESET pin should remain active during this time to insure proper CPU operation. RESET is active HIGH. Reset is asynchronous, but must meet setup and hold times t20, t20a and t21 for recognition in any specific clock. RESET sets the SMBASE descriptor to default address of 30000H. If the system uses SMBASE relocation, then the SRESET pin should be used for soft resets. For the 2X clock mode, the falling edge of RESET synchronizes the CPU internal clock phase. RESET must be used at power up and anytime the phase of the CPU clock must be re-synchronized to the system phase.
SRESET	I	The SRESET pin duplicates all the functionality of the RESET pin with the following two exceptions: 1. The SMBASE register will retain its previous value. 2. If UP# is asserted, SRESET will not have an effect on the host microprocessor. For soft resets, SRESET should remain active for at least 30 CLK2 periods (equal to 15 internal CPU CLK). SRESET is active HIGH. SRESET is asynchronous but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.

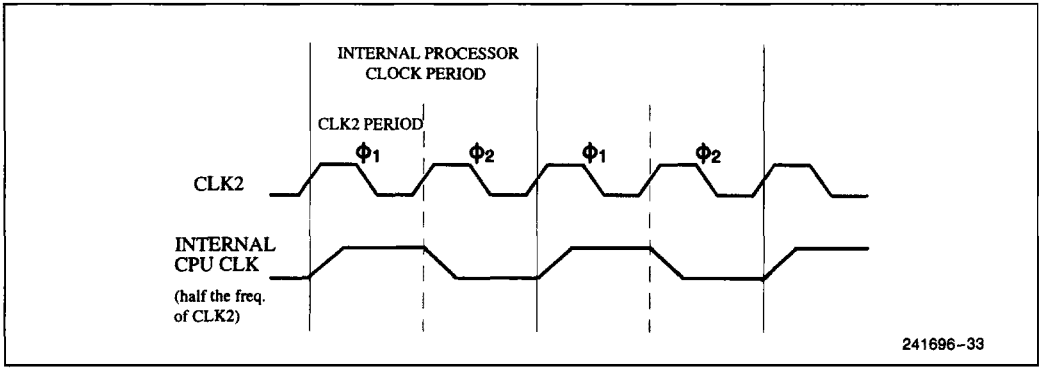


Figure 7-1. CLK2 Signal and Internal Processor Clock

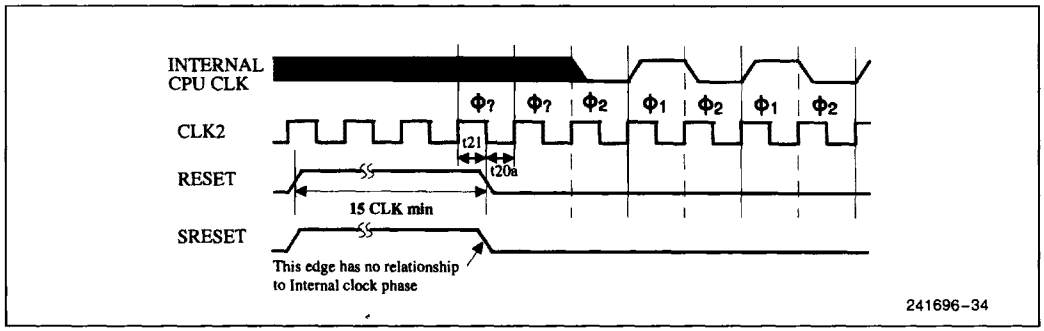


Figure 7-2. CLK2 and Internal Processor CLK vs. SRESET and RESET Timings

7.3 Clock Control

7.3.1 CLOCK GENERATION

The frequency of **CLK2** is twice the internal frequency of the CPU. The internal clock is comprised of two phases, "PH1" and "PH2". Each CLK2 period is a phase of the internal clock. Figure 7-1 illustrates the relationship between the CLK2 input and the internal phases. All set-up, hold, float-delay and valid delay timings are referenced to the rising edge of phase 1 of CLK2. Thus it is important to synchronize the external circuitry with the phase of the CLK2 input. The internal processor clock phase is determined at the falling edge of the RESET input. RESET must meet the specified setup and hold times to correctly synchronize the internal clock phase. See Figure 7-2.

7.3.2 STOP CLOCK

The CPU with the 2X clock option does not rely on an internal Phase Lock Loop to generate the internal phase clocks. Therefore, the frequency of the CLK2 input can be changed dynamically or "on-the-fly".

The 2X clock mode SL Enhanced Intel486 CPU provides an interrupt mechanism, STPCLK#, that places the CPU into a known state. Although the frequency of the CLK2 input can be dynamically changed between 0 MHz and the maximum operating frequency of the CPU, operation between 0 MHz and 8 MHz is not tested. Stopping the CLK2 input with the CPU in a known state requires use of the STPCLK# mechanism. When the CPU recognizes a STPCLK# request, the processor will stop execution on the next instruction boundary, stop the pre-fetcher, empty all internal pipelines and the write buffers, and then generate a Stop Grant bus cycle. At this point the CPU is in the Stop Grant state.

The rising edge of STPCLK# will tell the CPU that it can return to program execution at the instruction following the interrupted instruction.

Unlike the normal interrupts, INTR and NMI, the STPCLK# interrupt does not initiate interrupt acknowledge cycles or interrupt vector table reads.

STPCLK# is active LOW and is provided with an internal pull-up resistor. STPCLK# is an asynchronous signal, but must remain active until the processor issues the Stop Grant bus cycle. STPCLK# may be de-asserted at any time after the processor has issued the Stop Grant bus cycle. Note that STPCLK# should NOT be de-asserted before the CPU has issued the Stop Grant bus cycle. STPCLK# must be deasserted for a minimum of 5 clocks after RDY# is returned active for the Stop Grant bus cycle before being asserted again.

7.3.3 CLOCK CONTROL STATE DIAGRAM

The state diagram in Figure 7-3 shows the Stop Clock state transitions for the 2X clock mode.

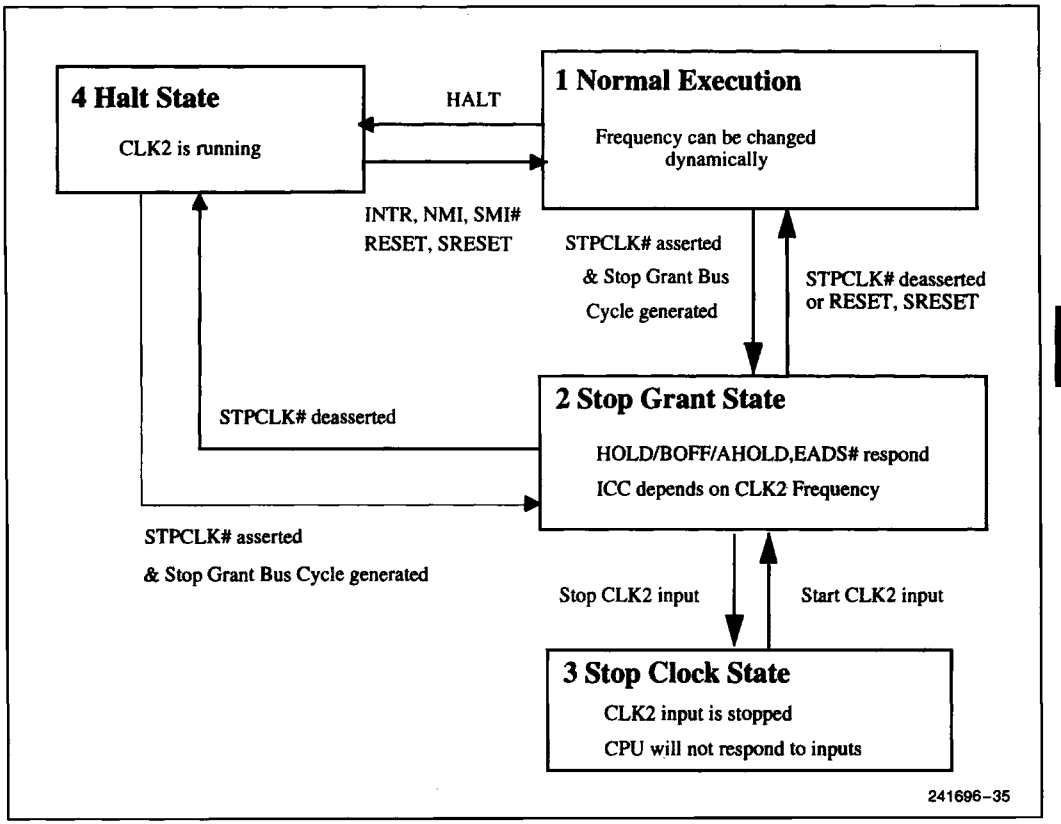


Figure 7-3. Stop Clock State Machine - 2X Clock Mode

7.3.3.1 Normal State

This is the Normal operating state of the CPU. During this state, the CLK2 input frequency can be changed dynamically or "on-the-fly" for power consumption control with no clock control latency. This capability provides a wide range of performance/power consumption options. Operation of the processor is tested between 8 MHz and the maximum operating frequency of the CPU. Operation below 8 MHz is guaranteed by design, though is not 100% tested. **Operation at 0 MHz is tested when the stop clock protocol (STPCLK#) is used.**

7.3.3.2 Stop Grant State

The CPU enters the **Stop Grant state** in response to a STPCLK# interrupt. The CPU will generate a Stop Grant bus cycle when it enters this state from the Normal state or the HALT state. The CPU will not generate a Stop Grant bus cycle when it enters the Stop Grant state from the Stop Clock state.

While in the **Stop Grant state**, the pull-up resistors on STPCLK# and 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.

During the Stop Grant state, the CPU will respond to HOLD, AHOLD and BOFF# normally and can perform cache invalidates. An active edge on either the SMI# or NMI interrupts will be latched and will be serviced after the rising edge of STPCLK#. An INTR request will be serviced after the CPU returns to the normal state as long as INTR is held active until the CPU issues an interrupt acknowledge bus cycle.

7.3.3.3 Stop Clock State

The CPU enters the **Stop Clock state** when the system stops the CLK2 input. The system can stop the CLK2 input on either a logic high or a logic low. The CLK2 input must be restarted in the same state as when it was stopped. In other words, any CLK2 input state can be stretched. (See Figure 7-4 for details). CPU operation at 0 MHz is tested only when the CPU is in the Stop Clock state.

In the Stop Clock state, the CPU does not respond to any stimulus. The CPU must re-enter the Stop Grant state (CLK2 input must be restarted) in order to perform any bus actions such as HOLD/HLDA cycles, invalidates (AHOLD/EADS# or FLUSH# cycles), and BOFF#. It is recommended that CLK2 be restarted 2 clocks before and continue until 2 clocks after the transition of the HOLD, AHOLD, EADS#, FLUSH#, or BOFF# signals.

The interrupt signals (SMI#, NMI, and INTR) will be recognized and serviced correctly if the input is held in the active state until the CPU returns to the Stop Grant state. The SL Enhanced Intel486 CPU Family requires that INTR be held active until the CPU issues an interrupt acknowledge cycle in order to guarantee recognition. This condition also applies to the existing Intel486 CPUs.

7.3.3.4 HALT State

The CPU enters the HALT state from the Normal state on a HLT instruction. The system can place the CPU into the Stop Grant state from the HALT state by asserting the STPCLK# input. The CPU will generate a Stop Grant bus cycle when it enters the Stop Grant state. If the CPU entered the Stop Grant state from the HALT state then it will return to the HALT state when the STPCLK# interrupt is de-asserted. When the CPU re-enters the HALT state it will generate a HALT bus cycle.

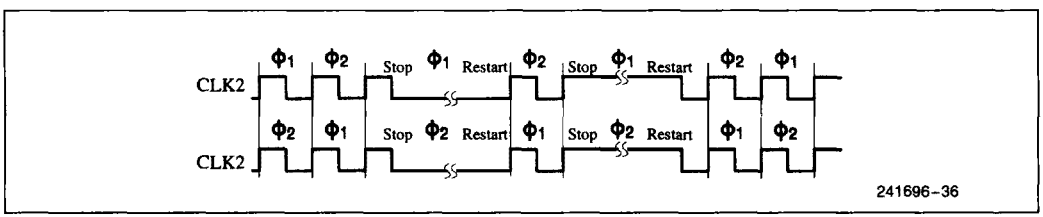


Figure 7-4. CLK2 Phase Coherence in CLK2 Stop and Restart

7.3.4 SUPPLY CURRENT MODEL FOR STOP CLOCK MODES AND TRANSITIONS

The following diagram illustrates the effect of different Stop Clock state transitions on the supply current.

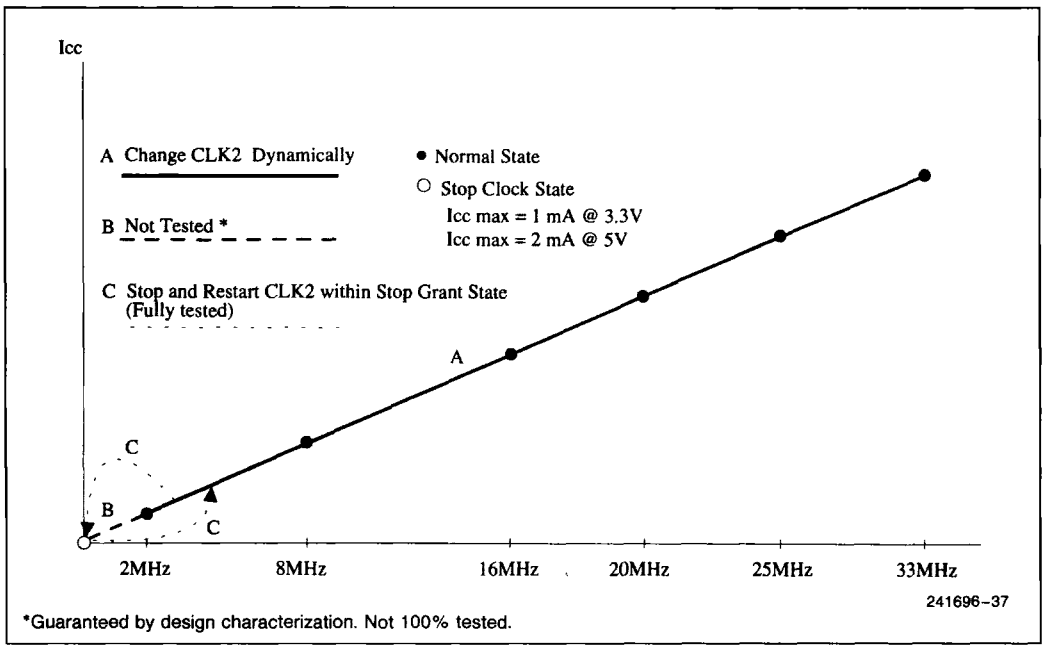


Figure 7-5. Supply Current Model for Stop Clock Modes and Transitions

Table 7-3. 3.3V Preliminary I_{CC} Values for 2X Clock SL Enhanced Intel486 SX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
3.3V	I_{CC} Active	25 MHz	250 mA	315 mA
		33 MHz	330 mA	415 mA
	I_{CC} Stop Clock	0 MHz	100 μA	1 mA

Table 7-4. 3.3V Preliminary I_{CC} Values for 2X Clock SL Enhanced Intel486 DX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
3.3V	I_{CC} Active	33 MHz	330 mA	415 mA
	I_{CC} Stop Clock	0 MHz	100 μA	1 mA

Table 7-5. 5V Preliminary I_{CC} Values for 2X Clock SL Enhanced Intel486™ SX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
5V	I_{CC} Active	25 MHz	430 mA	560 mA
		33 MHz	590 mA	685 mA
	I_{CC} Stop Clock	0 MHz	200 μA	2 mA

Table 7-6. 5V Preliminary I_{CC} Values for 2X Clock SL Enhanced Intel486 DX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
5V	I_{CC} Active	33 MHz	500 mA	630 mA
	I_{CC} Stop Clock	0 MHz	200 μA	2 mA

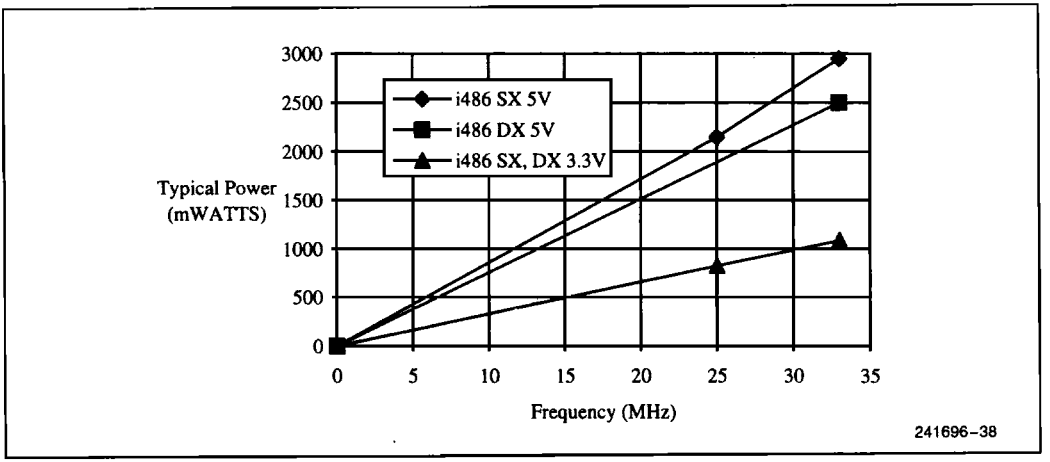


Figure 7-6. Frequency vs. Power (Typ) in 2X CLK Mode for 3.3V and 5V, SL Enhanced Intel486 SX and DX CPUs

7.4 D.C. Specifications for 2X Clock Option

For 2X clock D.C. specifications, refer to the 1X clock D.C. specifications (Section 6.1).

7.5 A.C. Specifications for 2X Clock Option

The A.C. specifications given in the tables of this section consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the rising edge of the phase 1 of the input system clock (CLK2), unless otherwise specified.

7.5.1 5V A.C. CHARACTERISTICS

Table 7-7. 5V A.C. Characteristics (2X Clock) Frequency = 25 and 33 MHz (Preliminary Information)
Functional operating range: V_{CC} = 5V ± 5%; T_{CASE} = 0 °C +85 °C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Min	Max	Unit	Notes
	Frequency		**25**		**33**	MHz	Note 1
	CLK2 Frequency		**50**		**66**	MHz	Note 1
t_1	CLK2 Period	20		15		ns	
t_2	CLK2 High Time	7		5		ns	at 2V
t_3	CLK2 Low Time	7		5		ns	at 0.8V
t_4	CLK2 Fall Time		2		2	ns	2V to 0.8V (2)
t_5	CLK2 Rise Time		2		2	ns	0.8V to 2V (2)
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA, SMIACT#, **FERR#*** Valid Delay	3	19	3	17	ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		28		21	ns	Note 2
t_8	PCHK# Valid Delay	3	24	3	23	ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	24	3	21	ns	
t_9	BLAST#, PLOCK# Float Delay		28		21	ns	Note 2
t_{10}	D0–D31, DP0–DP3 Write Data Valid Delay	3	20	3	19	ns	
t_{11}	D0–D31, DP0–DP3 Write Data Float Delay		28		21	ns	Note 2
t_{12}	EADS# Setup Time	9		6		ns	
t_{13}	EADS# Hold Time	4		4		ns	
t_{14}	KEN#, BS16#, BS8# Setup Time	9		6		ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	4		4		ns	
t_{16}	RDY#, BRDY# Setup Time	9		6		ns	
t_{17}	RDY#, BRDY# Hold Time	4		4		ns	
t_{18}	HOLD, AHOLD Setup Time	11		7		ns	
t_{18a}	BOFF# Setup Time	11		9		ns	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	4		4		ns	
t_{20}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#*** Setup Time	11		6		ns	
t_{20a}	RESET Falling Edge Setup Time	9		5		ns	
t_{21}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#*** Hold Time	4		4		ns	

2

Table 7-7. 5V A.C. Characteristics (2X Clock) Frequency = 25 and 33 MHz (Preliminary Information) (Continued)

Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Min	Max	Unit	Notes
t_{22}	D0–D31, DP0–DP3, A4–A31 Read Setup Time	6		6		ns	
t_{23}	D0–D31, DP0–DP3, A4–A31 Read Hold Time	4		4		ns	

NOTES:
* Present only in the SL Enhanced Intel486™ DX microprocessor.
1. 0 MHz operation is tested using the STPCLK# and STOP GRANT bus cycle protocol. Operation between 0 MHz < CLK2 < 8 MHz is guaranteed by design characterization, but is not 100% tested.
2. Not 100% tested, guaranteed by design characterization.

7.5.2 3.3V A.C. CHARACTERISTICS

Table 7-8. 3.3V A.C. Characteristics (2X Clock) Frequency = 25 and 33 MHz (Preliminary Information)

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

Symbol	Parameter	Min	Max	Min	Max	Unit	Notes
	Frequency		**25**		**33**	MHz	Note 1
	CLK2 Frequency		**50**		**66**	MHz	Note 1
t_1	CLK2 Period	20		15		ns	
t_2	CLK2 High Time	7		5		ns	at 2V
t_3	CLK2 Low Time	7		5		ns	at 0.8V
t_4	CLK2 Fall Time		2		2	ns	2V to 0.8V(2)
t_5	CLK2 Rise Time		2		2	ns	0.8V to 2V(2)
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA, SMIACT#, **FERR#*** Valid Delay	3	19	3	17	ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		28		21	ns	Note 2
t_8	PCHK# Valid Delay	3	24	3	23	ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	24	3	21	ns	
t_9	BLAST#, PLOCK# Float Delay		28		21	ns	Note 2
t_{10}	D0–D31, DP0–DP3 Write Data Valid Delay	3	20	3	19	ns	
t_{11}	D0–D31, DP0–DP3 Write Data Float Delay		28		21	ns	Note 2
t_{12}	EADS# Setup Time	9		6		ns	
t_{13}	EADS# Hold Time	4		4		ns	
t_{14}	KEN#, BS16#, BS8# Setup Time	9		6		ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	4		4		ns	
t_{16}	RDY#, BRDY# Setup Time	9		6		ns	
t_{17}	RDY#, BRDY# Hold Time	4		4		ns	
t_{18}	HOLD, AHOLD Setup Time	11		7		ns	
t_{18a}	BOFF# Setup Time	11		9		ns	

Table 7-8. 3.3V A.C. Characteristics (2X Clock) Frequency = 25 and 33 MHz (Preliminary Information) (Continued)

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

Symbol	Parameter	Min	Max	Min	Max	Unit	Notes
t_{19}	HOLD, AHOLD, BOFF# Hold Time	4		4		ns	
t_{20}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#** * Setup Time	11		6		ns	
t_{20a}	RESET Falling Egde Setup Time	9		5		ns	
t_{21}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#** * Hold Time	4		4		ns	
t_{22}	D0–D31, DP0–DP3, A4–A31 Read Setup Time	6		6		ns	
t_{23}	D0–D31, DP0–DP3, A4–A31 Read Hold Time	4		4		ns	

NOTES:
* Present only in the SL Enhanced Intel486™ DX Microprocessor
1. 0 MHz operation is tested using the STPCLK# and STOP GRANT bus cycle protocol. Operation between 0 MHz<CLK2<8 MHz is guaranteed by design characterization, but is not 100% tested.
2. Not 100% tested, guaranteed by design characterization.

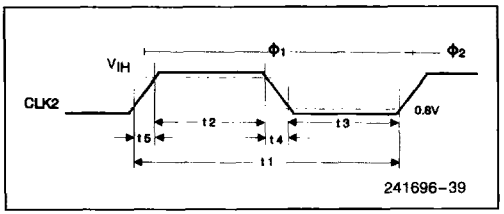
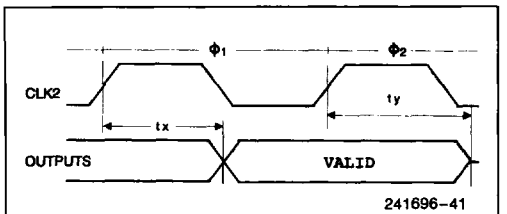
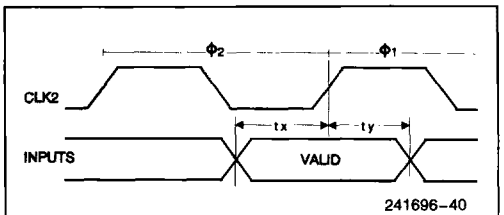


Figure 7-7. CLK2 Waveform



Output Valid Delay tx = t6, t8, t8a, t10
Output Float Delay ty = t7, t9, t11

Figure 7-9. Valid and Float Delay Timings



tx = t12, t14, t16, t18, t20, t20a, t22, t18a
ty = t13, t15, t17, t19, t21, t23

Figure 7-8. Setup and Hold Timings

8.0 TESTABILITY

8.1 Test Access Port

TCK
This Test Clock signal is an input to the CPU and provides the clocking function required by the JTAG Boundary scan feature. TCK is used to clock state information and data into the component on the rising edge of TCK on TMS and TDI, respectively. Data is clocked out of the CPU on the falling edge of TCK and TDO.

TDI
This Test Data Input signal is the serial input used to shift JTAG instruction and data into the CPU. TDI is sampled on the rising edge of TCK, during the SHIFT-IR and SHIFT-DR TAP controller states. During all other tap controller states, TDI is a "don't care".

TDO
This Test Data Output signal is the serial output used to shift JTAG instructions and the data out of the CPU. TDO is driven on the falling edge of TCK during the SHIFT-IR and SHIFT-DR TAP controller states. At all other times, TDO is driven to the high impedance state.

TMS
This Test Mode Select signal is decoded by the JTAG TAP (Test Access Port) 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, TMS is provided with an internal pull-up resistor.

Table 8-1. A.C. Specifications for the Test Access Port

Symbol	Parameter	Min	Max	Unit	Notes
t_{24}	TCK Frequency		8	MHz	Note 2
t_{25}	TCK Period	125		ns	
t_{26}	TCK High Time	40		ns	@ 2.0V
t_{27}	TCK Low Time	40		ns	@ 0.8V
t_{28}	TCK Rise Time		8	ns	Note 1
t_{29}	TCK Fall Time		8	ns	Note 1
t_{30}	TDI, TMS Setup Time	8		ns	Note 3
t_{31}	TDI, TMS Hold Time	10		ns	Note 3
t_{32}	TDO Valid Delay	3	30	ns	Note 3
t_{33}	TDO Float Delay		36	ns	Note 3
t_{34}	All outputs (Non-Test Valid Delay)	3	30	ns	Note 3
t_{35}	All Outputs (Non-Test Float Delay)		36	ns	Note 3
t_{36}	All Inputs (Non-Test) Setup Time	8		ns	Note 3
t_{37}	All Inputs (Non-Test) Hold Time	10		ns	Note 3

NOTES:
1. Rise/Fall Times are measured between 0.8V and 0.2V. Rise/Fall times can be relaxed by 1 ns per 10 ns increase in TCK period.
2. TCK period ≥ CLK period.
3. Parameter measured from TCK.

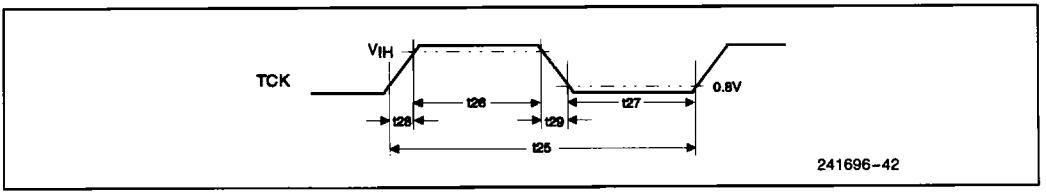


Figure 8-1. TCK Waveform

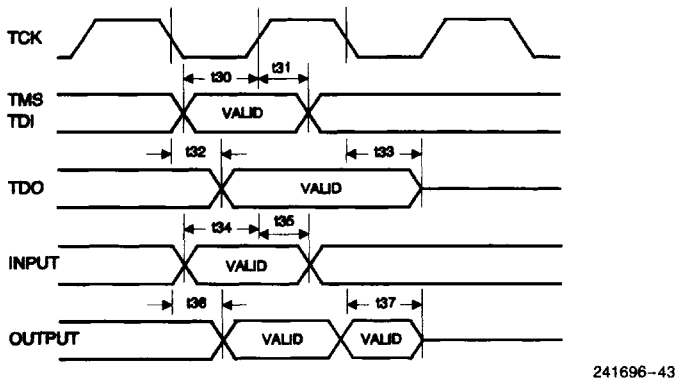


Figure 8-2. Test Signal Timing Diagram

8.2 Boundary Scan Component Identification Support

The IEEE standard 1149.1 includes an optional public instruction called "IDCODE" which is used to verify the correct parts are installed in the correct locations on a board. The SL Enhanced Intel486 CPUs support this instruction with a IDCODE value of 0010. When implemented IEEE requires that the register that is selected for the scan path be 32 bits long with four specific sub fields. These fields are:

- LSB (bit 0) must have a value of "1".
- Manufacturer Identity (bits 1–11). This is a compressed form of the JEDEC (pub 106-A) manufacturers code. Intel is identified by a hex value of 009.
- Part Number (bits 12–27). IEEE only requires that 2 different parts in the same package style with the TAP pins in the same location must have different values. Intel has sub divided this field further as indicated in the following tables. These sub fields are V_{CC}, Type, Family, and Model.
- Version (bits 28–31). IEEE recommends that this field should be assigned to identify the variant of a component type. Intel is using this field to identify major steppings of the parts.

Table 8-2. Boundary Scan Component Identification for Intel486™ CPUs (5V Versions)

Intel486 CPU Type	CMD Register	Version	V_{CC} 1=3 0=5	Intel Architecture Type	Intel486 CPU Family	Model	MFG ID Intel = 009H	1st bit	Boundary Scan ID (Hex)
DX2	0001	xxxx*	0	000001	0100	00101	00000001001	1	x0285013H
DX	0001	xxxx*	0	000001	0100	00001	00000001001	1	x0281013H
SX	0001	xxxx*	0	000001	0100	00010	00000001001	1	x0282013H
DX 2x CLK	1001	xxxx*	0	000001	0100	00001	00000001001	1	x0281013H
SX 2x CLK	1001	xxxx*	0	000001	0100	00010	00000001001	1	x0282013H

*Contact Intel for details.

Table 8-3. Boundary Scan Component Identification for Intel486™ CPUs (3.3V Version)

Intel486 CPU Type	CMD Register	Version	V_{CC} 1=3 0=5	Intel Architecture Type	Intel486 CPU Family	Model	MFG ID Intel = 009H	1st bit	Boundary Scan ID (Hex)
DX2	0001	xxxx*	1	000001	0100	00101	00000001001	1	x8285013H
DX	0001	xxxx*	1	000001	0100	00001	00000001001	1	x8281013H
SX	0001	xxxx*	1	000001	0100	00010	00000001001	1	x8282013H
DX 2x CLK	1001	xxxx*	1	000001	0100	00001	00000001001	1	x8281013H
SX 2x CLK	1001	xxxx*	1	000001	0100	00010	00000001001	1	x8282013H

*Contact Intel for details.

NOTES:
1. DX indicates the inclusion of a FPU, and SX indicates the exclusion of a FPU.
2. The distinction between 1x and 2x bus clocks is determined by reading the 4 bits loaded into the shift-register during the "Capture-IR" state. (labeled here as "cmd reg")

9.0 OverDrive™ PROCESSOR SOCKET FOR SL ENHANCED INTEL486 CPU-BASED SYSTEMS

This section provides OverDrive Processor socket specifications for systems based on the SL Enhanced Intel486 SX, DX and DX2 CPUs.

The OverDrive Processor socket for Intel486 DX2 CPU-based systems is a superset of the OverDrive Processor socket for Intel486 SX and DX CPU-based systems. Section 9.1 specifies the OverDrive Processor socket for Intel486 SX and DX CPU-based systems. Section 9.2 specifies the Pentium™ OverDrive Processor socket for 5V Intel486 DX2 CPU-based systems.

The OverDrive Processor socket specifications in this section are only valid for systems using 1X clock mode CPUs. There are no OverDrive Processor socket specifications compatible with 2X clock mode CPUs.

9.1 OverDrive Processor Socket for 5V SL Enhanced Intel486 SX and DX CPU-Based Systems

This section contains OverDrive Processor socket specifications for 5V systems based on the SL Enhanced Intel486 SX or DX CPUs. The OverDrive Processor socket specifications are also valid for 5V systems based on the existing Intel486 SX or DX CPUs.

9.1.1 OVERDRIVE PROCESSOR SOCKET CIRCUIT DESIGN

Figures 9-1 and 9-2 show the circuits which interface the original CPU with the OverDrive Processor socket. These circuits allow SL Enhanced Intel486 SX and DX (as well as Intel486 SX and DX) CPU-based systems to be upgraded with the OverDrive Processor.

9.1.1.1 OverDrive Processor Socket Circuit for SL Enhanced Intel486 DX CPU-Based Systems

The OverDrive Processor socket circuit for Intel486 DX CPU-based systems allows the Intel486 DX CPU to directly recognize when the OverDrive Processor socket is populated. When the UP# pin is driven active to the Intel486 DX CPU, the CPU three-states all of its output pins and enters power-down mode. This circuit is shown in Figure 9-1.

9.1.1.2 OverDrive Processor Socket Circuit for SL Enhanced Intel486 SX CPU-Based Systems

The OverDrive Processor socket circuit for Intel486 SX CPU-based systems allows the microprocessor to directly recognize when the OverDrive Processor socket is populated. When the UP# pin is driven active to the Intel486 SX CPU, the CPU three-states all of its output pins and enters power-down mode. This circuit is shown in Figure 9-2.

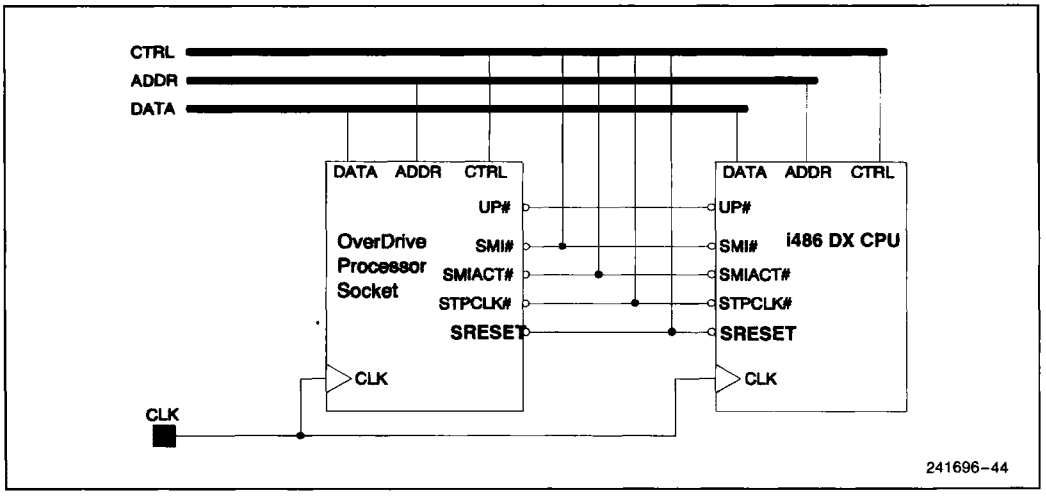


Figure 9-1. OverDrive™ Processor Socket Circuit for SL Enhanced Intel486™ DX CPU-Based Systems

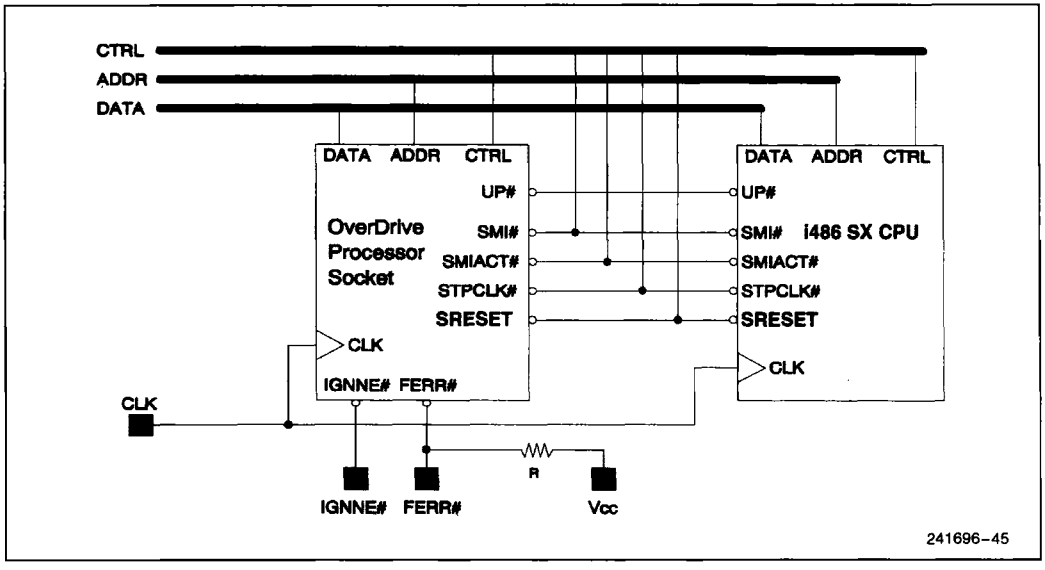


Figure 9-2. OverDrive™ Processor Socket Circuit for SL Enhanced Intel486™ SX CPU-Based Systems

9.1.2 SOCKET LAYOUT

The OverDrive Processors for Intel486 SX and DX CPU-based systems are supplied with an attached heat sink. Intel486 SX and DX microprocessor system designs must provide space for the heat sink on the OverDrive Processor.

The maximum and minimum dimensions for the 169 pin, PGA package with heat sink are shown in Table 9-1. The maximum height of the OverDrive Processor for 5V Intel486 SX and DX CPU-based systems from the pin stand-offs to the top of the heat sink, including the adhesive thickness, is 0.552 inches.

Table 9-1 OverDrive™ Processor, 169 pin, PGA Package Dimensions With Heat Sink Attached

DIMENSION (inches)	Minimum	Maximum
A. Heat Sink Width	1.520	1.550
B. PGA Package Width	1.735	1.765
C. Heat Sink Edge Gap	0.065	0.155
D. Heat Sink Height	0.312	0.360
E. Adhesive Thickness	0.008	0.012
F. Package Height from Stand-Offs	0.140	0.180
G. Total Height from Package Stand-Offs to Top of Heat Sink	0.460	0.552

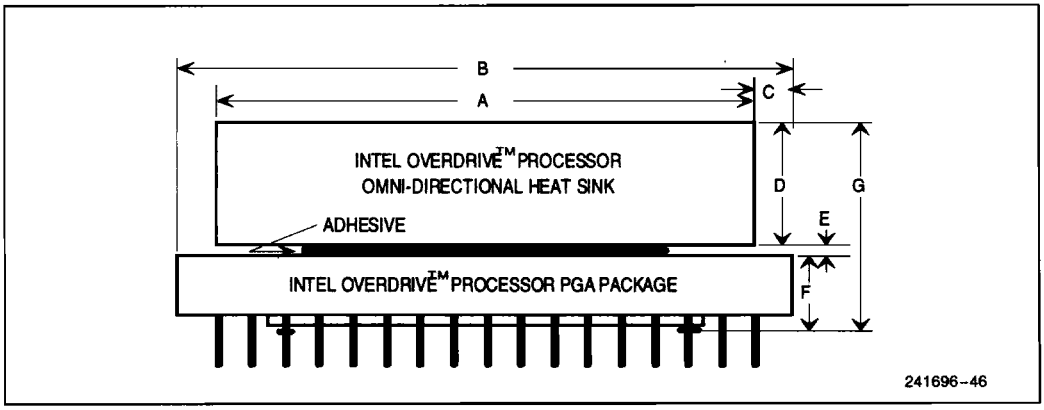


Figure 9-3. OverDrive™ Processor for 5V SL Enhanced Intel486™ SX and DX CPU-Based Systems (169 pin, PGA Package with Heat Sink)

9.1.3 THERMAL MANAGEMENT

The heat generated by the OverDrive Processor requires careful management of heat dissipation.

Thermal Equations: The method for calculating the heat dissipation for an OverDrive Processor (with a heat sink) or a standard CPU is the same except that the reference point for measuring the device temperature is different. The OverDrive Processor specifies T_{sink} (the temperature at the outside center base of the heat sink, not on the heat sink marking plate or cooling posts) and θ_{JS} (the thermal resistance from the silicon junction to the heat sink base). The relationships between temperature, thermal resistance and power are shown in the following equations:

$$T_{sink} = T_{ambient} + (P_{max} * \theta_{SA})$$

where,

$T_{ambient}$ = Ambient Temperature,
P_{max} = Power ($I_{CC} * V_{CC}$),
θ_{SA} = $\theta_{JA} - \theta_{JS}$.

The maximum I_{CC} values in Table 9-2 are the best known design estimates and should be used as the maximum specification.

Table 9-2. OverDrive™ Processor Maximum and Typical I_{CC} Values for 5V Intel486 SX and DX CPU-Based Sytems

System Frequency (MHz)	Typical I_{CC} (mA)	Maximum I_{CC} (mA)
25	TBD	1000
33	TBD	1250

The thermal resistance values for the OverDrive Processor (169 pin, PGA package with heat sink) are shown in Table 9-3. The maximum TA values shown in Table 9-4 were calculated using T_S = 85°C, V_{CC} = 5.3V, the maximum I_{CC} values shown in Table 9-2, and θ_{JS} and θ_{JA} values shown in Table 9-3.

Table 9-3. Thermal Resistance for the OverDrive Processor for 5V Intel486 SX and DX CPU-Based Systems

5V OverDrive Processor	θ_{JS}	Airflow (Ft/min., LFM)				
	1.9°C/W	0	200	400	600	800
θ_{JA} (°C/W)		12.0	8.6	6.6	5.1	4.6

Table 9-4. Maximum T_A for the OverDrive™ Processor for 5V Intel486 SX and DX CPU-Based Systems

5V OverDrive Processor	Airflow (Ft/min., LFM)					
	f_{CLK}(MHz)	0	200	400	600	800
T_A(°C)	25	31	49	60	68	70
	33	18	40	53	63	67

For further details about thermal and mechanical package specifications and methodologies, refer to the 1993 Packaging Handbook (order number 240800).

9.1.4 DESIGN CONSIDERATIONS

Timing Dependent Loops

The OverDrive Processor may internally execute instructions at multiple frequencies of the input clock. Thus software (or instruction based) timing loops will execute faster on the OverDrive Processor than on the Intel486 SX or DX CPU (at the same input clock frequency). Instructions such as NOP, LOOP, and JMP $+2, have been used by BIOS to implement timing loops, that are required, for example, to enforce recovery time between consecutive accesses for I/O devices. These instruction-based timing loop implementations may require modification for systems intended to be upgraded with the OverDrive Processor.

In order to avoid any incompatibilities, it is recommended that timing loops be implemented in hardware rather than in software. This provides transparency and also requires no change to BIOS or I/O device drivers in the future when moving to higher processor clock speeds. As an example, a timing loop may be implemented as follows: The software performs a dummy I/O instruction to an unused I/O port. The hardware for the bus controller logic recognizes this I/O instruc-

tion and delays the termination of the I/O cycle to the CPU by keeping RDY# or BRDY# deasserted for the appropriate amount of time.

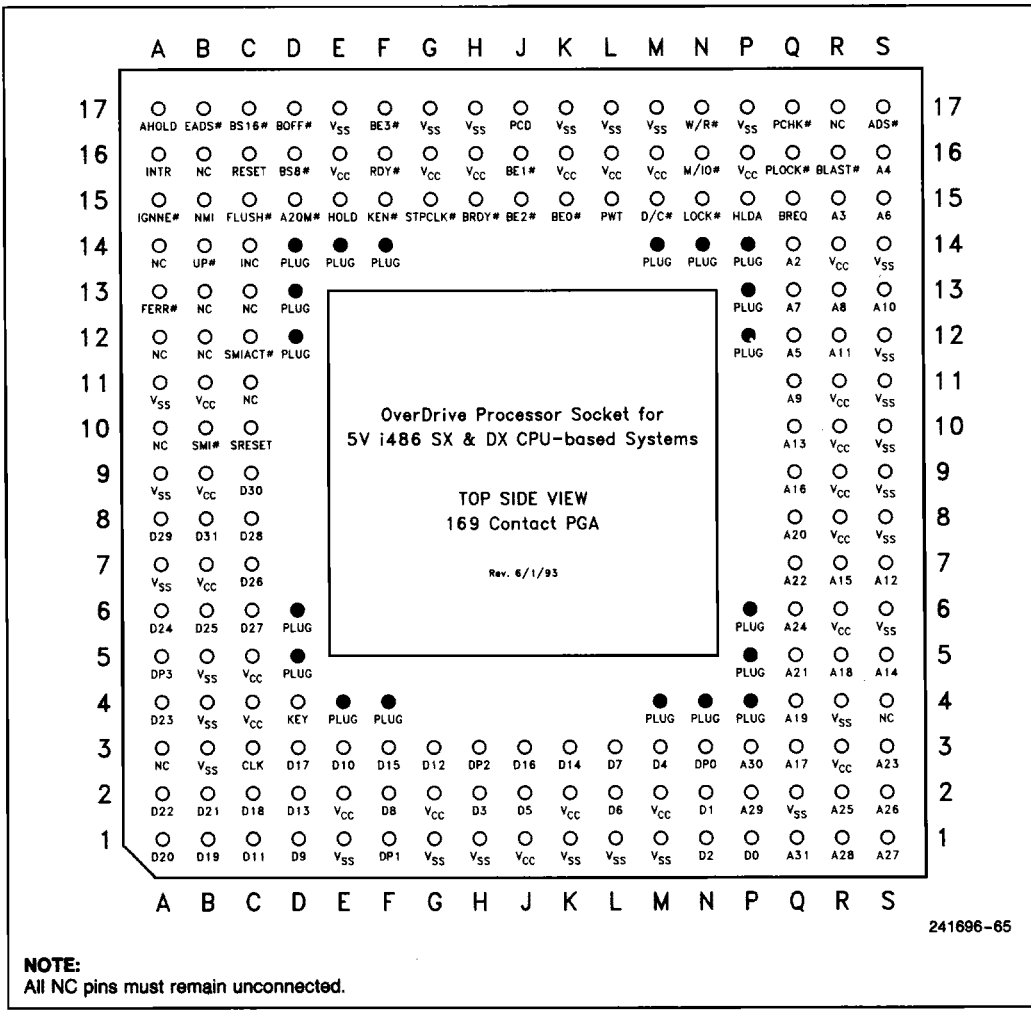
Boundary Scan

The OverDrive Processor does not support the Boundary Scan feature.

9.1.5 TESTABILITY

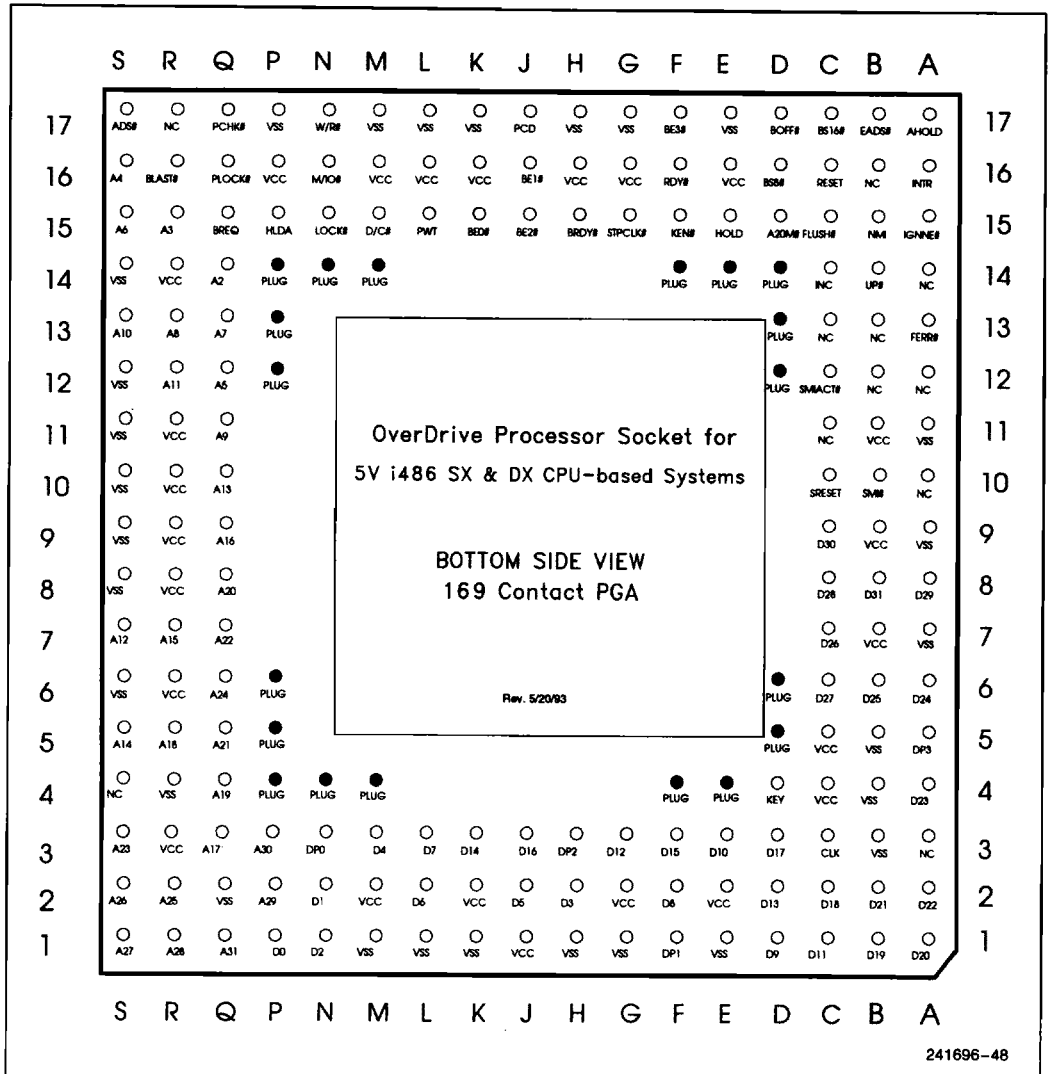
The electrical functionality of the OverDrive Processor socket can be verified by fully testing the system with a populated OverDrive Processor socket. We recommend the system be tested with all compatible OverDrive Processors to ensure there are no compatibility issues.

9.1.6 OVERDRIVE PROCESSOR SOCKET PINOUT FOR 5V INTEL486 SX AND DX CPU-BASED SYSTEMS



NOTE:
All NC pins must remain unconnected.

Figure 9-4. 5V OverDrive™ Processor Socket Pinout (169 contact PGA, Top Side View)



NOTE:
All NC pins must remain unconnected.

Figure 9-5. 5V OverDrive™ Processor Socket Pinout (169 contact PGA, Bottom Side View)

Table 9-5. 5V OverDrive™ Processor Socket Pin Cross Reference (169 contact PGA)

Address		Data		Control		Control		N/C[1]	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	PWT	L15	A3	B7	A7
A_3	R15	D_1	N2	ADS#	S17	PLOCK#	Q16	A10	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	RDY#	F16	A12	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	RESET	C16	A14	C4	B3
A_6	S15	D_4	M3	BE1#	J16	SMI#	B10	B12	C5	B4
A_7	Q13	D_5	J2	BE2#	J15	SMIACT#	C12	B13	E2	B5
A_8	R13	D_6	L2	BE3#	F17	SRESET	C10	B16	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16	STPCLK#	G15	C11	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17	UP#	B14	C13	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15	W/R#	N17	R17	H16	G17
A_{12}	S7	D_{10}	E3	BREQ	Q15			S4	J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16				K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17				K16	K1
A_{15}	R7	D_{13}	D2	CLK	C3				L16	K17
A_{16}	Q9	D_{14}	K3	D/C#	M15	**Position**		**INC[2]**	M2	L1
A_{17}	Q3	D_{15}	F3	DP0	N3					
A_{18}	R5	D_{16}	J3	DP1	F1	KEY	D4	C14	M16	L17
A_{19}	Q4	D_{17}	D3	DP2	H3	PLUG	D5		P16	M1
A_{20}	Q8	D_{18}	C2	DP3	A5	PLUG	D13		R3	M17
A_{21}	Q5	D_{19}	B1	EADS#	B17	PLUG	D14		R6	P17
A_{22}	Q7	D_{20}	A1	FERR#	A13	PLUG	E4		R8	Q2
A_{23}	S3	D_{21}	B2	FLUSH#	C15	PLUG	E14		R9	R4
A_{24}	Q6	D_{22}	A2	HLDA	P15	PLUG	N4		R10	S6
A_{25}	R2	D_{23}	A4	HOLD	E15	PLUG	N14		R11	S8
A_{26}	S2	D_{24}	A6	IGNNE#	A15	PLUG	P4		R14	S9
A_{27}	S1	D_{25}	B6	INTR	A16	PLUG	P5			S10
A_{28}	R1	D_{26}	C7	KEN#	F15	PLUG	P13			S11
A_{29}	P2	D_{27}	C6	LOCK#	N15	PLUG	P14			S12
A_{30}	P3	D_{28}	C8	M/IO#	N16					S14
A_{31}	Q1	D_{29}	A8	NMI	B15					
		D_{30}	C9	PCD	J17					
		D_{31}	B8	PCHK#	Q17					

NOTES:
The correct orientation of the OverDrive Processor for 5V Intel486™ SX and DX CPU-based systems is keyed by the 'KEY' pin (D4). Plugs must be provided in three other inner corner pin locations (D14, P4, and P14) and the next nearest pin locations in all corners as listed above.
1. All NC pins must remain unconnected.
2. The **INC** pin is defined to be an *internal no-connect*. This means that the pin is not internally connected and may be used for the routing of external signals.

9.1.7 D.C./A.C. CHARACTERISTICS

The OverDrive Processor is compatible with the maximum rating specification and the D.C./A.C. specifications of the Intel486 SX and DX CPUs. Refer to the thermal management section (Section 9.1.3) for the maximum power supply current and operating temperature specifications.

9.2 Pentium™ OverDrive Processor Socket for 5V SL Enhanced Intel486 DX2 CPU-Based Systems

This section contains the specification for the Pentium OverDrive Processor socket for systems based on the Intel486 DX2 CPU. These OverDrive Processor socket specifications are also valid for 5V systems based on the existing Intel486 DX2 CPU. Please note that the Pentium OverDrive Processor socket specifications presented in this document are preliminary and subject to change. For more updated information about the OverDrive Processor socket, please contact your local Intel representative.

9.2.1 PENTIUM OVERDRIVE PROCESSOR SOCKET OVERVIEW FOR 5V SL ENHANCED Intel486 DX2 CPU-BASED SYSTEMS

The OverDrive Processor socket specifies 237 contacts, which correspond to a standard 240 pin socket with one inside "KEY" contact, one outer "KEY" contact and four "orientation" contacts plugged on the outside corner. The inside "KEY" contact provides backward compatibility with the OverDrive Processor socket for Intel486 SX and DX CPU-based systems. The four contacts plugged on the outside corner and the outer "KEY" pin ensure proper orientation for the Pentium OverDrive Processor. Additionally, all 120 inner contacts of a standard 240 pin socket (the inner 11 rows minus the "KEY" pin) should be plugged to ensure proper alignment of the OverDrive Processor for Intel486 SX and DX CPU-based systems.

To support Intel's SMM of the SL Enhanced Intel486 processor, new pins have been added to the Pentium OverDrive Processor socket.

The **INC** Pin is defined to be an internal no-connect. This means that the pin is not internally connected and may be used for the routing of external signals. For example, pin D15 is defined to be the **FERR#** pin on the Intel486 DX CPU. By routing D15 and B14 together, the **FERR#** signal can be made to appear on both pins.

Note that the **SRESET** pin has been defined at location D11 for compatibility with the SL Enhanced Intel486 DX2 Processor. On the Pentium OverDrive processor, D11 is defined as an **INC** pin. The OverDrive processor signal **INIT** and the SL Enhanced Intel486 CPU signal **SRESET** may be electrically connected on the motherboard, provided the system can handle the functionality differences depending on which part is inserted into the socket. Specifically, **INIT** and **SRESET** are similar except that **SRESET** will flush the on-chip cache, while **INIT** will keep the cache contents intact. Systems should not depend on **SRESET** to flush the cache in future versions of the SL Enhanced Intel486 processor, as previously described in this document.

9.2.2 MECHANICAL DESIGN CONSIDERATIONS

The Pentium OverDrive Processor is designed to fit in a standard 240-lead (19 x 19) PGA socket with four corner pins removed. The Pentium OverDrive Processor uses an active heat sink, and therefore, requires vertical clearance to allow adequate air circulation.

The maximum and minimum dimensions of the Pentium OverDrive Processor package with a fan/heat sink are shown in Table 9-6. The fan/heat sink unit is divided into the size of the actual heat sink, and the required free space above the heat sink. The total height required for the Pentium OverDrive Processor from the motherboard will depend on the height of the PGA socket. The total external height given in Table 9-6 is only measured from the PGA pin stand-offs. Table 9-6 also details the minimum clearance needed around all four sides of the PGA package.

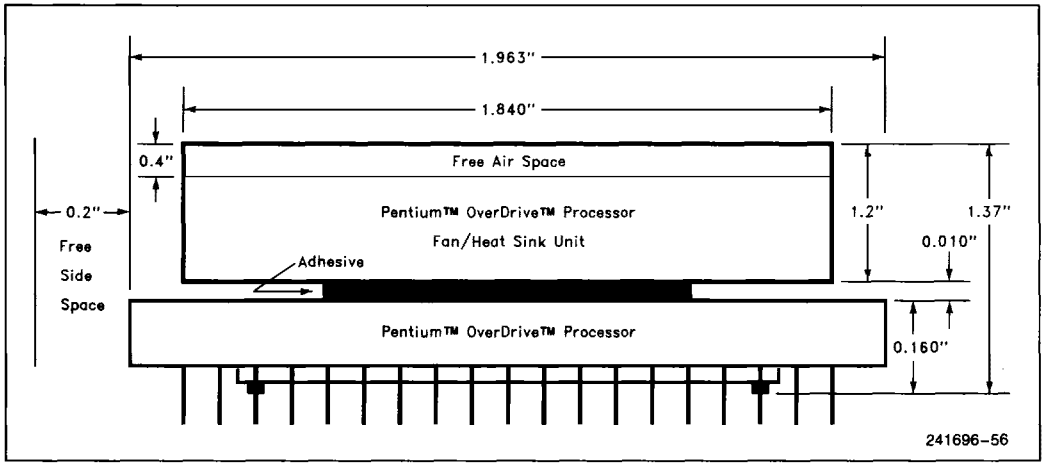


Figure 9-6. 236 pin, PGA Package with Active Heat Sink Attached

Table 9-6. Pentium™ OverDrive Processor, 236 Pin PGA Package Dimensions with Active Heat Sink Attached

Component	Length and Width (Inches)		Height (inches)	
	Minimum	Maximum	Minimum	Maximum
PGA Package	1.950	1.975	0.140	0.180
Adhesive	N/A	N/A	0.008	0.012
Heat Sink Unit	1.830	1.850	N/A	N/A
Heat Sink	N/A	N/A	0.790	0.810
Req'd Free Space	N/A	N/A	0.400	0.400
External Total	**1.950**	**1.975**	**1.338**	**1.402**
Space from Package	0.200	0.200	N/A	N/A

Since the Pentium OverDrive Processor dissipates more power than the Intel486 CPU family members, it requires a larger cooling capacity. To facilitate the task of cooling the Pentium OverDrive Processor, Intel will ship the product with a fan/heat sink. No external connections (i.e., power) will be required for the fan/heat sink. All the needed connections will be made through the pins of the processor. The amount of extra power needed for the fan is accounted for in the I_{CC} numbers of the processor (see Section 9.2.3). To ensure adequate air circulation, the additional clearance specified in Table 9-6 must be provided.

9.2.3 THERMAL DESIGN CONSIDERATIONS

The Pentium OverDrive Processor and system chassis have several unique design requirements due to the attached active heat sink. The following sections provide sample maximum system operating temperature calculations so systems may be designed to comply with the thermal requirements of the Pentium OverDrive Processor.

Thermal Calculations for a Hypothetical System:

The following equation can be used to calculate the maximum operating temperature of a system:

$T_A(in) = T_{SINK} - (Power * \theta_{SI})$

The parameters are defined as follows:

$T_A(in)$: The temperature of the air going **into** the fan/heat sink.

T_{SINK}: Temperature of heat sink base, as measured in the center.

Power: Dissipation in Watts $= V_{CC} * I_{CC}$

θ_{SI}: Heat Sink to Internal Temperature $[T_A(in)]$ Thermal Resistance

$T_A(out)$: The temperature of the air outside the system.

Since the Pentium OverDrive Processor uses an active heat sink, θ_{SI} (as shown in Table 9-7) is relatively constant, regardless of the airflow provided to the processor. Table 9-8 details the maximum current requirements of the Pentium OverDrive Processor. The maximum ambient temperature specification for the Pentium OverDrive Processor is 55°C for both 25 MHz and 33 MHz processors with the heat sink attached. Therefore, the internal temperature of the air (T_A (in) may not exceed 55°C under the worst case operating conditions specified for the system. This ensures that the value of T_{SINK} does not exceed 85°C.

Table 9-7. Thermal Resistance (°C/W) θ_{SI}

Processor Type	θ_{SI} – °C/W
Fan/Heat Sink	2.4

Table 9-8. Pentium™ OverDrive Processor Typical and Maximum I_{CC} Values

System Frequency (MHz)	Processor Typical I_{CC} (mA)	Processor Maximum I_{CC} (mA)
25	TBD	1900
33	TBD	2500

I_{CC} is dependent upon the V_{CC} level of the system, processor bus loading, software code sequences, and silicon process variations.

Maximum TA(in) is specified and can be verified by using the equation and parameters provided.

$T_A(in) = T_{SINK} - (Power * \theta_{SI})$

$T_A(in) = 85°C - ((2.5A * 5V) * 2.4 °C/W)$

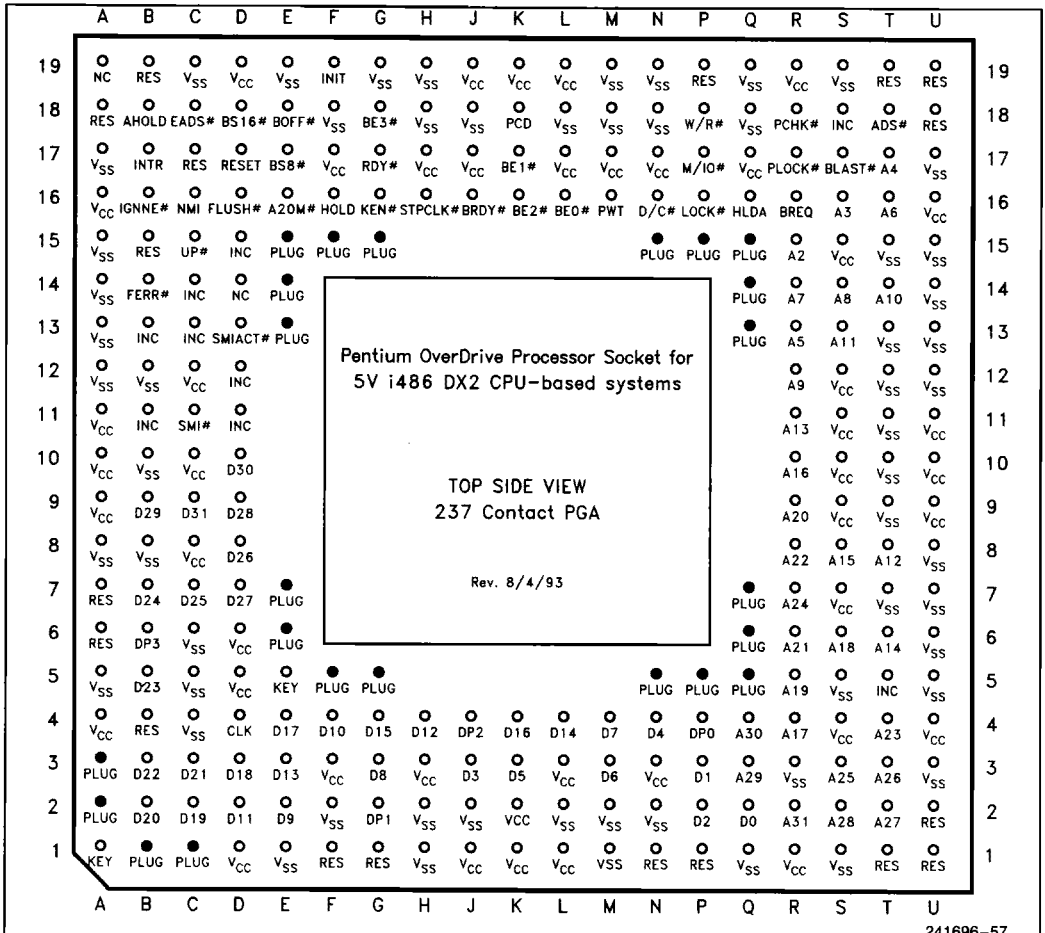
$T_A(in) = 85°C - (12.5 Watts * 2.4 °C/Watt)$

$T_A(in) = 85°C - 30°C$

$T_A(in) = 55°C$

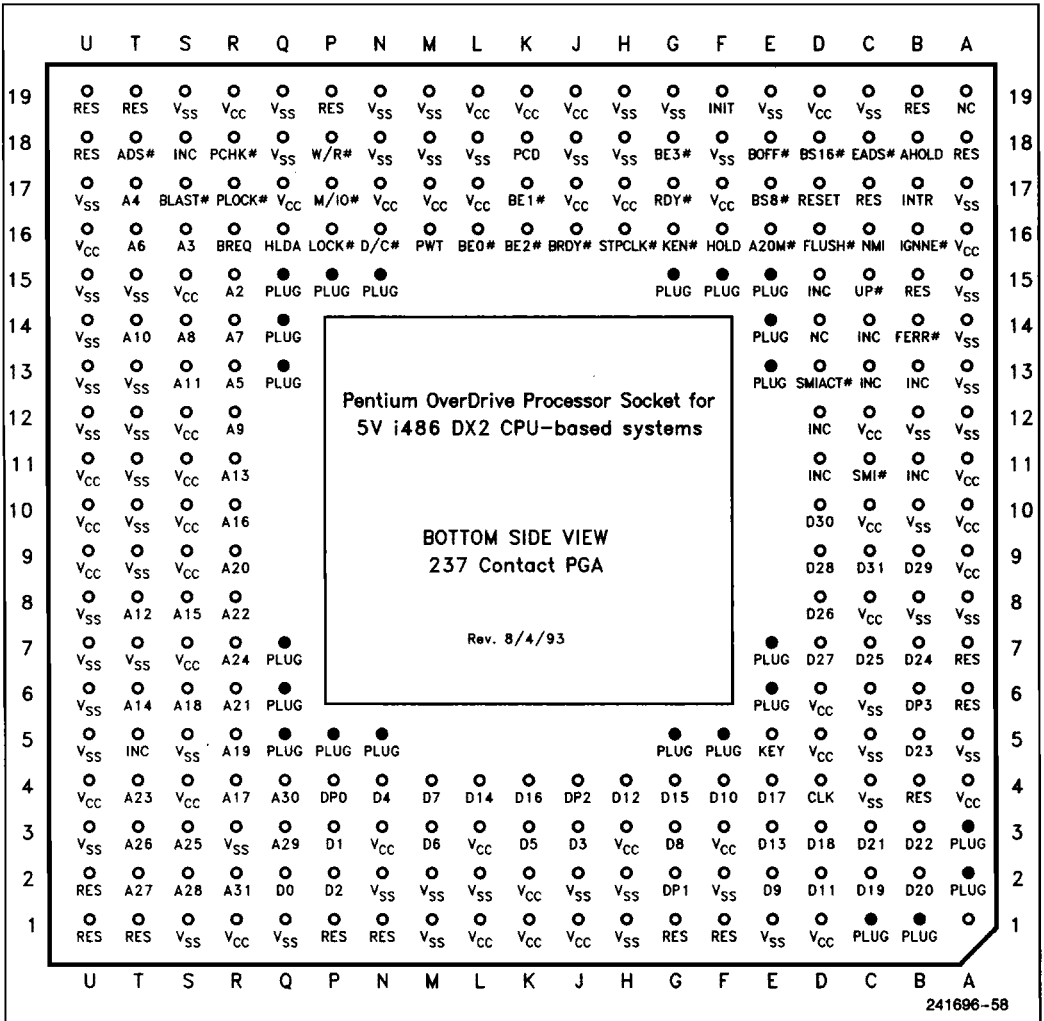
Assuming the internal system ambient TA(in) is within 5°C–10°C of TA(out), this would allow the maximum Ta(out) temperature to be approximately 45°C–50°C. It is the responsibility of the system designer to ensure TA(in) meets this specification by providing sufficient airflow around the Pentium OverDrive Processor to remove the heated air expelled by the fan/heat sink.

9.2.4 PENTIUM™ OVERDRIVE PROCESSOR SOCKET PINOUT FOR 5V SL ENHANCED INTEL486 DX2 CPU-BASED SYSTEMS



NOTE:
All NC and RES pins must remain unconnected.

Figure 9-7. Pentium™ OverDrive Processor Socket Pinout for 5V SL Enhanced Intel486 DX2 CPU-Based Systems (Top Side View)



NOTE:
All NC and RES pins must remain unconnected.

Figure 9-8. Pentium™ OverDrive Processor Socket Pinout for 5V SL Enhanced Intel486 DX2 CPU-Based Systems (Bottom Side View)

Table 9-9. 5V OverDrive Processor Socket Pin Cross Reference for Intel486 DX2 CPU-Based Systems

Address		Data		Control		Control		RES*	V_{CC}		V_{SS}	
A2	R15	D0	Q2	A20M#	E16	RDY#	G17	A6	A4	L1	A5	M18
A3	S16	D1	P3	ADS#	T18	RESET	D17	A7	A9	L3	A8	M19
A4	T17	D2	P2	AHOLD	B18	SMI#	C11	A18	A10	L17	A12	N2
A5	R13	D3	J3	BE0#	L16	SMIACT#	D13	A19	A11	L19	A13	N18
A6	T16	D4	N4	BE1#	K17	STPCLK#	H16	B4	A16	M17	A14	N19
A7	R14	D5	K3	BE2#	K16	UP#	C15	B15	C8	N3	A15	Q1
A8	S14	D6	M3	BE3#	G18	W/R#	P18	B19	C10	N17	A17	Q18
A9	R12	D7	M4	BLAST#	S17			C17	C12	Q17	B8	Q19
A10	T14	D8	G3	BOFF#	E18			F1	D1	R1	B10	R3
A11	S13	D9	E2	BRDY#	J16			G1	D5	R19	B12	S1
A12	T8	D10	F4	BREQ	R16			N1	D6	S4	C4	S5
A13	R11	D11	D2	BS8#	E17			P1	D19	S7	C5	S19
A14	T6	D12	H4	BS16#	D18			P19	F3	S9	C6	T7
A15	S8	D13	E3	CLK	D4			T1	F17	S10	C19	T9
A16	R10	D14	L4	D/C#	N16			T19	H3	S11	E1	T10
A17	R4	D15	G4	DP0	P4	**Position**		U1	H17	S12	E19	T11
A18	S6	D16	K4	DP1	G2	KEY	E5	U2	J1	S15	F2	T12
A19	R5	D17	E4	DP2	J4	KEY	A1	U18	J17	U4	F18	T13
A20	R9	D18	D3	DP3	B6	PLUG	A2	U19	J19	U9	G19	T15
A21	R6	D19	C2	EADS#	C18	PLUG	A3	**N/C***	K1	U10	H1	U3
A22	R8	D20	B2	FERR#	B14	PLUG	B1	A19	K2	U11	H2	U5
A23	T4	D21	C3	FLUSH#	D16	PLUG	C1	D14	K19	U16	H18	U6
A24	R7	D22	B3	HLDA	Q16	PLUG	E6	**INC**			H19	U7
A25	S3	D23	B5	HOLD	F16	PLUG	E14	B11			J2	U8
A26	T3	D24	B7	IGNNE#	B16	PLUG	E15	B13			J18	U12
A27	T2	D25	C7	INIT	F19	PLUG	F5	C13			L2	U13
A28	S2	D26	D8	INTR	B17	PLUG	F15	C14			L18	U14
A29	Q3	D27	D7	KEN#	G16	PLUG	P5	D11			M1	U15
A30	Q4	D28	D9	LOCK#	P16	PLUG	P15	D12			M2	U17
A31	R2	D29	B9	M/IO#	P17	PLUG	Q5	D15				
		D30	D10	NMI	C16	PLUG	Q6	S18				
		D31	C9	PCD	K18	PLUG	Q14	T5				
				PCHK#	R18	PLUG	Q15					
				PLOCK#	R17							
				PWT	M16							

* All RES pins are reserved for later use by Intel. To ensure proper operation of the microprocessor, all RES and N/C pins should be left unconnected. Please contact Intel for design information.

9.2.5 D.C./A.C. CHARACTERISTICS

The 5V Pentium OverDrive Processor is compatible with the A.C. specifications for the 5V SL Enhanced Intel486 DX2 CPU (maximum I_{CC} current values provided in Section 9.2.3).

10.0 REVISION HISTORY

Revision -002 of the SL Enhanced Intel486 Microprocessor Data Sheet Addendum contains many updates and improvements to the original version. The sections significantly revised since version -001 are:

Table 1-1	Added 3.3V 50 MHz version of the Intel486 DX2 CPU.
Table 2-3	Added note to explain that pins A3, A14, B14 and B16 are Boundary Scan pins for the 50 MHz version of the standard Intel486 DX CPU.
Table 2-4	Corrected PGA control pins, F17 and A10.
Table 2-5	Added new table to show pinout differences between the SL Enhanced Intel486 SX CPU and the SL Enhanced Intel486 DX and DX2 CPUs for pins 66 and 72.
Table 2-7, Sections 3.2 and 7.3.2	Added clarification for the STPCLK# signal with respect to the Stop Grant bus cycle.
Figure 3-1	Modified figure to show that STPCLK# must be deasserted for at least 5 clocks after RDY# becomes active.
Section 3.5.6	Added statement about Auto Idle Power Down.
Section 4.3.1	Added clarification about the SMI# specification.
Section 4.3.3.1 and Table 4-1	Added information about SMRAM state save map.
Sections 6.0 to 6.4	Changed T_{case} specification guarantee of 90°C at zero airflow to 85°C.
Table 6-1	Added max. D.C. spec. of 6 pF for C_{CLK}.
Table 6-2	Changed 3.3V Intel486 SX-33 CPU typical and maximum I_{CC} specifications.
Table 6-4	Added I_{CC} values for 3.3V Intel486 DX2 CPU.
Figure 6-1	Added 3.3V Intel486 DX2-50 CPU to graph.
Table 6-11	Added minimum CLK frequency specifications for 50 and 66 MHz versions of 5V Intel486 DX2 CPU.
Tables 6-15a and 6-15b	Changed θ_{JA} and θ_{JC} specifications for Intel486 DX and DX2 CPUs.
Tables 6-20, 6-21 and 6-22	Recalculated $T_{ambient}$ values for Intel486 SX, DX and DX2 CPUs based on T_{case} of 85°C, new θ_{JA}, θ_{JC} and I_{CC} specifications.
Section 6.4	Added capacitive derating information section.
Figures 9-4 and 9-5	Corrected pinouts of the OverDrive processor for 5V Intel486 SX and DX CPUs.
Section 9.2	Modified 5V OverDrive Processor Socket section to reflect new Pentium™ OverDrive Processor Socket specifications for SL Enhanced Intel486 DX2 CPU-based systems.
Section 10.0	Added revision history section to reflect changes since -001 version of publication.

APPENDIX A
SYSTEM DESIGN NOTES

A.1 SMM Environment Initialization

When the SL Enhanced Intel486 CPUs are operating in Real Mode, the physical address at which instructions and data are fetched is determined by the segment register and an offset (i.e., CS and IP for instructions). When a new value is loaded into a segment register, the new value is shifted to the left by four bits and stored in a segment base register that corresponds to that particular segment (CSBASE, DSBASE, ESBASE, etc.). It is the value stored in the segment base register that is actually used to generate a physical address. For example, the linear address to be used for fetching instructions is determined by adding the value contained in the CS segment base register with the value in the IP register.

When the CPU is in Protected Mode, the segment registers are used as selectors to a descriptor table. Each descriptor in a descriptor table contains information about the segment in use, including the segments BASE address (i.e., CSBASE), the limit (or size of the segment), as well as protection level, privileges, operand sizes, and the segment type. In Protected Mode, the linear address is determined by adding the base portion of the descriptor to the appropriate offset.

When in System Management Mode, the CPU operates in a pseudo-Real Mode, with address calculation performed in the Real Mode manner. However, the CPU adds the value in the segment base register with the value in the EIP register, rather than the IP register, so there are no limits as to the segment size. The physical address of an instruction is obtained by adding the value in CSBASE to the value in EIP.

When entering SMM, it may be necessary to initialize the segment registers to point to SMRAM (see section 4.4.2. for their value on SMM entry). If SMBASE has not been relocated, then the necessary segment registers can be initialized to point to SMRAM by using the value in the CS register, 3000H, which points to the SMRAM address space.

When an SMI# occurs after SMBASE has been modified, CSBASE is loaded with the new value of SMBASE. **However, the CS selector register still contains the value 3000H, not the value corresponding to the new SMBASE.**

In order to initialize the segment registers to point to the new SMRAM area, we need to read the value of SMBASE from the SMM state save in memory. Since the data segment registers are initialized to 0, we cannot use them to access the SMM state save area. However, we can perform a read relative to the CS register by using a CS override prefix to a normal memory read. Although CS still contains 3000H, CSBASE contains the value of SMBASE, and CSBASE is used for the address generation.

Once the value of SMBASE is obtained, it must be shifted to the right by four bits to get the appropriate value to be placed in the segment registers. The CS register itself can be initialized by executing a far jump instruction to an address within SMBASE, which causes CS to be reloaded with a value corresponding to SMBASE.

Figure A-1 describes one method of initializing the segment registers when SMBASE has been relocated. This method works if SMBASE is less than 1 Megabyte.

```
;read the value of SMBASE from the state save area

      mov           si,FEF8H              ;SMBASE slot in SMM state save area
      mov           eax,cs:[si]           ;copy SMBASE from SMBASE:FEF8H to eax

;scale the SMBASE value to a 16-bit quantity

      mov           cl,4
      ror           eax,cl                ;scaled value of SMBASE now in ax

;to load cs, we need to execute a far jump to an address that has been stored
;at memory location PTR_ADDR

;store the SMBASE value and an offset to a memory location that can be used as
;an indirect jump address

      mov           di,PTR_ADDR           ;PTR_ADDR is the location used to
                                          ;store the jump address
      mov           bx,OFFSET             ;OFFSET is the address where
                                          ;execution continues after the
                                          ;far jump

      mov           cs:[di],bx            ;store the offset for the far jump
      inc           di
      inc           di
      mov           cs:[di],ax            ;store the segment address for the
                                          ;far jump, which is SMBASE
      mov           bx,PTR_ADDR           ;bx now contains the address of the
                                          ;location holding the jump address

;initialize DS and ES with the correct address of SMBASE

      mov           ds,ax
      mov           es,ax

;execute a far jump instruction to load the CS register

      jmp           far [bx]              ;jump to address stored at memory
                                          ;location pointed to by bx

;CS now contains the correct value of SMBASE, and execution continues from the
;address SMBASE:OFFSET
```

Figure A-1. Initialization of Segment Registers Within SMM

A.2 Accessing SMRAM

A.2.1 LOADING SMRAM WITH AN INITIAL SMI HANDLER

Under normal conditions, the SMRAM address space should only be accessible by the CPU while it is in SMM mode. However, some provision must be made for providing the initial SMM interrupt handler routine.

Since System Management Mode must be transparent to all operating systems, the initial SMM handler must be loaded by the system BIOS. At some time during the power on sequence, the system BIOS will need to move the SMM handler routine from the BIOS ROM to the SMRAM. The system designer must provide a hardware mechanism that allows access to SMRAM while SMIACT# from the CPU is inactive. One method would be to provide an I/O port in the memory controller that forces memory cycles at a given address to be relocated to the SMRAM. Once the initial SMM handler has been loaded to SMRAM, the I/O port would be disabled to protect against accidental accesses to SMRAM.

The system BIOS must provide an SMM handler at the address 38000H. If the system designer has chosen to take advantage of the SMRAM relocation feature of the CPU, this handler must change the SMBASE register in the SMM state save. Next, the BIOS must move the full featured SMM handler to the new address. An SMI# must be generated in order to change the SMBASE register before the BIOS passes control to the operating system.

A.2.2 SMRAM HIDDEN FROM DMA AND BUS MASTERS

In a system that allows DMA or other devices to take control of the system bus, care must be taken to ensure that only the master CPU can access SMRAM. If an external bus master requests use of the system bus (by asserting HOLD or BOFF#) while the CPU is executing an SMM handler routine, the CPU would respond by passing control of the bus to the requesting device. The system memory controller must redirect any memory accesses that are not generated by the CPU to normal system memory as if SMIACT# was inactive.

DMA accesses to the SMRAM area must be redirected to the correct address space when the initialization routine is loading SMRAM, as well as when the CPU is in SMM.

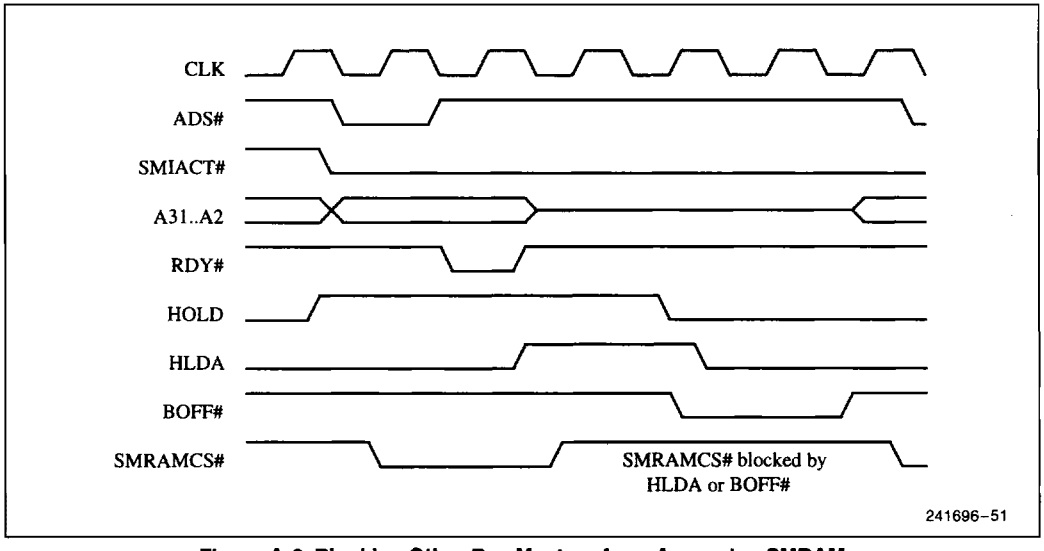


Figure A-2. Blocking Other Bus Masters from Accessing SMRAM

It is not recommended to block bus control requests when in SMM, since the increased bus access latency could cause compatibility issues with some software or expansion hardware.

A.2.3 ACCESSING SYSTEM MEMORY FROM WITHIN SMM

In order to enter a suspend state where power is removed from some or all of system memory, it is necessary for the CPU to have access to the entire system address space from within SMM. Access to system memory from within SMM requires that the memory controller decode both SMIACT# and the CPU address to determine accesses to SMRAM. Only those memory addresses that are defined as being SMRAM space would be directed to SMRAM. If SMRAM is located at an address that overlays normal system memory address space (see section 4.6.2.), the CPU must have a method of accessing both SMRAM (for code reads) and system memory simultaneously.

Ideally, a method of accessing system memory that is mapped underneath SMRAM would be provided by the system memory controller. The memory controller would provide a register that allows system memory at a given address to be remapped to a different address, which is not overlaid by SMRAM. When the SMM handler implements a suspend, it would first move all of system memory that is not underneath SMRAM to a non-volatile medium (such as a hard disk drive). Next, the SMRAM image would be transferred to the non-volatile medium. Finally, the memory underneath SMRAM would be accessed and copied to the non-volatile medium with a CPU read to the remap address space, which is redirected to the overlaid system memory (see Figure A-3).

If the memory controller does not provide a method of accessing overlaid system memory, it is possible to implement a software procedure to accomplish the same goal. However, the software method is quite complex, and a hardware method is preferred. A description of the software method follows.

The ability to access the system memory that is located in the address space under SMRAM requires a method of resuming from SMM to a predetermined address space. This can be accomplished with the following procedure.

When resuming from SMM, the CPU continues execution at the address contained in the CS and EIP slots within the SMM state save. However, the resume address cannot be changed by simply modifying the CS and EIP slots, since the CPU will use the CS descriptor to determine the actual resume address. The descriptor registers are stored in reserved slots in the SMM state save, and they cannot be directly modified.

By replacing the suspend state save with a previously obtained image of a state save that returns to a known location, the SMM suspend handler can force a return to a given address:

1. During initial system power up, execute an SMI# from a predetermined address (the address immediately preceding the address to which you later wish to resume). This can be accomplished by generating an SMI# in response to an I/O instruction or executing a halt instruction and using an SMI# to exit the halt state.

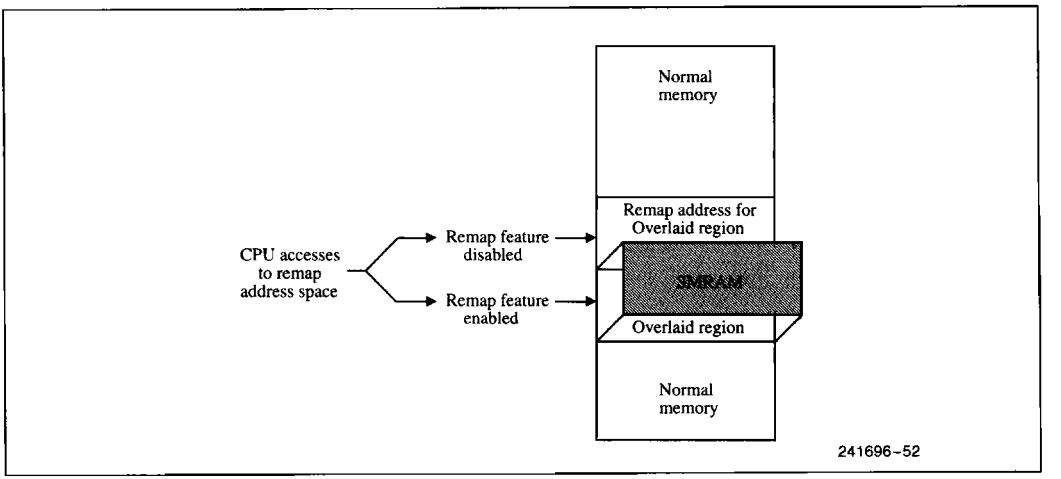


Figure A-3. Remapping Memory that is Overlaid by SMRAM

2. Save the state save from this SMM to a safe location (SMRAM).
3. When the system needs to resume to a given address from some other SMI#, the stored state save can be substituted for the state save generated from that particular SMM.

Now that we have the ability to resume from SMM to a predetermined address, we can access the entire system memory space from within SMM before executing a suspend:

1. During a suspend SMM, save all system memory except that which is located underneath SMRAM to a specified (and reserved) section of the hard disk. The ability to access system memory requires the memory controller to decode both SMIACT# and the CPU address, and direct a limited section (maybe 64 or 128K) of the CPU address space to SMRAM. All other CPU memory accesses should go to normal system memory.
2. Save the contents of the SMM state save to the hard disk.
3. Modify the SMM state save so that the RSM instruction will return to a predefined address, which is not in the application that was interrupted. The code at this address must contain the remainder of the suspend SMM handler. The predefined address can be anywhere in the CPU address space, since the contents of system memory have already been saved to disk.
4. Execute a RSM instruction, which exits SMM and returns control to a predetermined address (which must contain the rest of the SMM suspend handler).
5. Save the rest of system memory (that which is located underneath SMRAM) to the hard disk. This address space can now be accessed with normal move instructions, since we are no longer in SMM.
6. Save a flag (in CMOS memory) indicating that the next reset should cause a resume from suspend.
7. Powerdown the memory (and possibly the CPU).
8. When power is restored, the CPU is reset and begins execution of the POST in BIOS. Early in the POST, the system should check the status of the suspend flag.
9. Load a preliminary SMM handler to location 38000H and generate an SMI#. The SMM handler should read the SMBASE slot from the SMM state save that was stored to hard disk. SMBASE is then modified to point to the final SMRAM location and the system resumes from SMM back to the system BIOS.
10. Restore the contents of system memory located underneath SMRAM from the hard disk.
11. Generate a second SMI#, which executes an SMM handler at the original value of SMBASE (before the suspend SMM). The SMM handler restores the contents of the rest of system memory from the hard disk, and then restores the original SMM state save to the SMM state save area in SMRAM, discarding the most recent SMM state save.
12. Execute an RSM instruction, which returns execution to the application that was interrupted by the suspend request.

A.3 Interrupts and Exceptions During SMM Handler Routines

To ensure transparency to existing system software, the SMM handler should not depend on interrupt or exception handlers provided by the operating system. However, in some cases it may be necessary to service interrupts or exceptions while in System Management Mode. In these cases, SMM compliant interrupt and exception handlers, as well as an SMM compliant interrupt vector table, should be provided.

A.3.1 SMM COMPLIANT VECTOR TABLES

An SMI# interrupt request can be generated while code is running under any of the other three CPU operating modes (Real, Virtual-86, or Protected). When entering the SMM handler, the CPU enters a pseudo-real mode, and the beginning of the interrupt vector table must be located at the address 00000000H. Before allowing any interrupts or exceptions to occur, the SMM handler routine must provide a valid interrupt vector table. Any code that is executed before setting up an SMM compliant interrupt vector table must be written carefully to ensure that no exceptions are generated.

The system memory controller could relocate accesses to the SMM interrupt vector table to a location within SMRAM. In this case, when SMIACT# is active, all accesses to the lowest 1 Kbyte of the CPU address space would be redirected to SMRAM, which would contain an SMM compliant vector table that has already been initialized.

If the system memory controller does not redirect interrupt vector table reads to an address within SMRAM, there are three steps required to provide an SMM compliant interrupt vector table:

1. Save the contents of memory at address 00000000H to SMRAM
2. Provide vectors for any possible interrupts or execptions at the apropriate location in the vector table
3. Restore the original memory contents from SMRAM before exiting the SMM handler routine

A.3.2 INTERRUPTS AND SUBROUTINES WITH SMRAM RELOCATION

There is an additional issue that must be considered if interrupts or exceptions are to be executed within SMM and SMRAM has been relocated. Interrupt or subroutine calls from within SMM operate in a manner similar to Real Mode. When a subroutine is called or an interrupt is recognized, the 16-bit CS and IP registers are pushed onto the stack to provide a return address.

When SMRAM is relocated to an address space above 1M and an interrupt or subroutine call occurs, only 16 bits of the EIP register are pushed onto the stack. When returning from the subroutine or interrupt, the CPU will vector to a location where the upper 16 bits of EIP are zero. This can be avoided for subroutines by using an address size override before calling the subroutine. However, the issue remains for interrupts.

A.4 Floating Point Operation and SMM

A.4.1 THE NEED TO SAVE THE FPU ENVIRONMENT

When the CPU enters System Management Mode, the context information for the interrupted application is automatically saved to a specific state save address. When the SMM handler returns control to the interrupted application by executing the RSM instruction, the context information from the interrupted application is restored to the CPU by reading from the state save location. This mechanism allows the SMM handler routine to modify most of the CPU registers without the need to explicitly save them to memory. However, the registers in the CPU's Floating Point Unit (FPU) are not automatically saved when the CPU enters SMM. If the SMM handler needs to modify any of the registers in the FPU, or if the register data will be lost due to entering a power down state, the SMM handler must first explicitly save the FPU state as it existed in the interrupted application.

There are two instances in which an SMM handler routine must be aware of the Floating Point Unit (FPU):

1. When removing power from the CPU / FPU for the purpose of executing a suspend sequence.
2. When the SMM handler uses FPU instructions.

In both of these cases, the SMM handler must save the state of the FPU as it was left by the interrupted application.

The information stored by the FPU state save instructions (FSAVE, FNSAVE, FSTENV, and FNSTENV) is dependent on the operating mode of the CPU. The FPU state save instructions store the FPU state information in one of four formats: 16-bit Real Mode, 32-bit Real Mode, 16-bit Protected Mode, or 32-bit Protected Mode, depending on the CPU operating mode. The content of the information saved also varies slightly, depending on the CPU operating mode in which the save instruction was executed. For example, the 32-bit Protected Mode FNSAVE instruction saves the address of the last executed FPU instruction and its operands in the form of a segment selector and a 32-bit offset. In contrast, the 16-bit Real Mode FNSAVE instruction saves the address information in the form of a 20 bit physical address. Since the format with which the FPU state restore instructions (FRSTOR and FLDENV) recall the information is also dependent on the operating mode of the CPU, the save and restore instructions must be executed from the same CPU operating mode.

A.4.2 SAVING THE STATE OF THE FLOATING POINT UNIT

When an SMM handler routine needs to save the state of the Floating Point Unit, it must save all FPU state information necessary for the interrupted application to continue processing. This state information includes the contents of the Floating Point Unit stack, which requires use of the FNSAVE or FSAVE instruction (FSTENV does not save the contents of the FPU stack). If the last executed non-control Floating Point instruction caused an error (such as a divide by 0), the saved information must also include the address of the failing instruction and the addresses of any operands for that instruction. Without these addresses, it would be impossible for the FPU exception handler of the interrupted application to correct the error and restart the instruction.

The FNSAVE and FSAVE instructions differ in that FNSAVE does not wait for the FPU to check for an existing error condition before storing the FPU environment information. If there is an unmasked FPU exception condition pending, execution of the FSAVE instruction will force the CPU to wait until the error condition is cleared by the software exception handler. Because the CPU is in System Management Mode, the appropriate exception handler will not be available, and the FPU error would not be corrected in the manner expected by the interrupted application program. For this reason, the FNSAVE instruction should be used when saving the environment of the FPU within SMM.

Since the SMM handler does not know the CPU mode in which the interrupted application was executing (16 or 32 bit, Real or Protected), the SMM handler must execute the FNSAVE instruction in a mode in which

all FPU state information is stored. The 32-bit Protected Mode format of the FNSAVE instruction is a super set of all other formats of the FNSAVE instruction. Therefore, executing the 32-bit Protected Mode FNSAVE instruction ensures that all FPU state information will be saved.

Executing the FNSAVE instruction in 32-bit Protected Mode requires that the CPU be temporarily placed in Protected Mode. Rather than perform all of the setup details and overhead necessary to place the CPU into Protected Mode, including the initialization of all descriptors and descriptor tables, it is possible to temporarily place the CPU into Protected Mode for the purpose of executing only a few carefully written instructions. This can be accomplished by setting the PE bit in the CR0 register, and then executing a short jump to clear the instruction pipelines.

It is important to note that any instruction that modifies a segment register will cause the CPU to attempt to load a new descriptor from the descriptor table. (The occurrence of an interrupt or an exception would cause the CPU to load a new descriptor, so interrupts must be disabled during this sequence.) Since neither the descriptors nor the descriptor table have been initialized, this would cause the system to crash. Therefore, all segment registers that are to be used in the FPU state save instructions must be initialized before entering Protected Mode.

Figure A-4 gives an example of the code that can be used to place the CPU in Protected Mode and save the FPU state.

Note that the no wait form (FNSAVE) of the save instruction must be used. In the event that the previous FPU instruction caused a floating point error, we do not want to wait for this error to be serviced before executing the save instruction. Additionally, if the FSAVE instruction were used, the operand size override prefix would be incorrectly applied to the implicit WAIT instruction which precedes FSAVE, rather than to the save instruction itself.

Before exiting the SMM handler and returning to the interrupted application, the register contents of the Floating Point Unit must be returned to their previous values. This can be accomplished by executing the 32-bit Protected Mode format of the FRSTOR instruction. Figure A-5 gives an example code segment that can be used to restore the FPU to the state in which it was interrupted by the SMI request.


```
;first initialize the registers used to store the state save information

       mov             dx,SEGMENT          ;SEGMENT is the segment to be used by
                                           ;the save instruction,
       mov             ds,dx               ; normally it should point to SMRAM
       mov             si,OFFSET           ;OFFSET is the offset used in the save
                                           ;instruction

;set the PE bit in CR0

       mov             eax,cr0             ;read the old value of CR0
       or              eax,00000001H       ;set the PE bit
       mov             cr0, eax

;enter protected mode by executing a short jump to clear the prefetch queue

       jmp             protect
protect:

;we can now save the state of the FPU in the protected mode format

       db              66H                 ;use an operand size override prefix
                                           ;to set 32-bit format
       fnsave          [si]                ;FPU state saved to SEGMENT:OFFSET

;now return to real mode to continue with the SMM handler (no jump is
;required)

       mov             eax,cr0             ;clear the PE bit in CR0
       and             eax,0FFFFFFFEH
       mov             cr0,eax
```

Figure A-4. Saving the FPU State in 32-bit Protected Mode

```
;first initialize the registers used to recall the state save information

       mov          dx,SEGMENT          ;SEGMENT is the segment to be used by
                                        ;the restore instruction,
       mov          ds,dx               ; normally it should point to SMRAM
       mov          si,OFFSET           ;OFFSET is the offset used in the
                                        ;restore instruction

;set the PE bit in CR0

       mov          eax,cr0             ;read the old value of CR0
       or           eax,00000001H       ;set the PE bit
       mov          cr0, eax

;enter protected mode by executing a short jump to clear the prefetch queue

       jmp          protect
protect:

;we can now recall the state of the FPU from the previous FNSAVE instruction
;(in the protected mode format)

       db           66H                 ;use an operand size override prefix
                                        ;to set 32-bit format
       fnrstor      [si]                ;FPU state restored from
                                        ;SEGMENT:OFFSET

;now return to real mode to continue with the SMM handler (no jump is
;required)

       mov          eax,cr0             ;clear the PE bit in CR0
       and          eax,FFFFFFFEH
       mov          cr0,eax
```

Figure A-5. Restoring the FPU State from a 32-bit Protected Mode Save

Note that the no wait form (FNRSTOR) of the restore instruction must be used. If the FRSTOR instruction were used, the operand size override prefix would be incorrectly applied to the implicit WAIT instruction which precedes FRSTOR, rather than to the save instruction itself.

A.5 Support for Power Managed Peripherals

A.5.1 SHADOW REGISTERS

Before power is removed from any device, the state of that device must be saved in a protected memory space so that the device can be reinitialized to its previous state. If a peripheral contains a write only register, the value in that register can be recovered by providing shadow registers that are both readable and writeable. These shadow registers would be updated every time the peripheral registers are written, but they have no function other than tracking the data written to a particular register.

In addition to the write only registers in a system, there are several other registers that must be shadowed. Any device that requires registers to be programmed in a particular sequence must also have its registers shadowed. Examples in a typical personal computer include the programmable interrupt controller, the DMA controller, and the programmable timer/counter.

It is also possible to perform shadowing of some write only registers using SMM. Any time a write cycle is generated to a write only register, the system can generate an SMI#. The SMM handler can use the CPU state information saved in the SMM state save to save the data from the interrupted I/O cycle to a predetermined location in the SMRAM space.

The information contained in the SMM state save can be used (with the knowledge that the SMI# was in response to an I/O write instruction) to determine both the address and the data of the interrupted write instruction. The SMM handler can examine the OPCODEs of previous instructions by decrementing the IP (or EIP) register. Once the correct OPCODE is determined, it can be used with the values in the EAX and DX slots of the SMM state save to update the information in the memory used to shadow the I/O register. I/O write instructions occur in one of three forms: 1) a write to an address that is specified in the OPCODE; 2) a write to an address contained in the DX register; or 3) a string write to an address contained in the DX register.

The I/O write instructions have the following OPCODEs:

Table A-1. I/O Write Instruction OPCODEs

Instruction	OPCODE	Notes
OUT x,al	E6 x	x is the address of the I/O port
OUT x,ax	E7 x	x is the address of the I/O port
OUT x,eax	E7 x	x is the address of the I/O port
OUT dx,al	EE	
OUT dx,ax	EF	
OUT dx,eax	EF	
OUTSB	6E	
OUTSW	6F	
OUTSD	6F	

The SMM handler must know whether a particular I/O port is 16 or 32 bits in order to distinguish between 16 and 32 bit I/O write cycles.

The SMM handler can decrement the value of IP contained in the state save, and then examine the memory contents at that address. If the SMM handler knows that the last instruction was an I/O write instruction, and writes to I/O addresses 6EH, 6FH, 0EEH, and 0EFH will not cause an SMI#, it can use the SMM state save data for EAX and EDX to reconstruct the last instruction.

A.5.2 HANDLING INTERRUPTED I/O WRITE SEQUENCES

In a typical personal computer, there are several hardware devices that require the control registers for that device to be programmed in a particular order. For example, the interrupt controller, the DMA controller, the programmable timer/counter, the keyboard controller, and the real time clock all require a series of accesses to properly initialize the registers in that particular device. Some of these devices may require successive accesses to registers located at different addresses, while others may require several control registers to be programmed through write cycles to the same address.

If an SMI request interrupts an application that is in the process of initializing the registers in one of these devices, special care must be taken to ensure that the peripheral is returned to its original state when control is returned to the interrupted program. For some SMM handler events, it may be necessary to power down the device or change the state of a register within the device. In these cases, the SMM handler must return control to the interrupted application in such a way that the application can continue with the correct sequential access in the interrupted sequence.

To accomplish this, the SMM handler must restore the original values of all registers in the device, and restart the interrupted sequence so that the application may continue where it left off. This requires system hardware to shadow all registers that need to be accessed in the sequence, keep an index indicating which position in the sequence the register occupies, and keep a pointer so that SMM software knows to which register the last access was directed. This pointer would indicate the last register of each sequence that was programmed in the particular peripheral.

For example, programming the master interrupt controller requires a write to I/O port 20H (ICW1) followed by four write cycles to I/O port 21H (ICW2, ICW3, ICW4, and OCW1). If this sequence is interrupted by an SMI request, and the resulting SMM handler either modifies or powers down the interrupt controller, the SMM handler must return control to the interrupted application such that the following access to the interrupt controller would access the correct register in the sequence. System hardware must save the contents of each of the registers, as well as a pointer indicating which register was last written.

Before returning control to the interrupted application, the SMM handler must initialize ICW1-ICW4 and OCW1 to their previous values. It would then re-write the appropriate registers so that the first access by the application program would be to the location in the sequence following the last location it programmed before it was interrupted by the SMI request.

A similar procedure must be followed for each of the peripherals that require control registers to be initialized in a particular order.