Power Management 24

The embedded Pentium® processor family implements Intel's System Management Mode (SMM) architecture. This chapter describes the hardware interface to SMM and Clock Control.

24.1 Power Management Features

- System Management Interrupt can be delivered through the SMI# signal or through the local APIC using the SMI# message, which enhances the SMI interface, and provides for SMI delivery in APIC-based Pentium processor dual processing systems.
- In dual processing systems, SMIACT# from the bus master (MRM) behaves differently than in uniprocessor systems. If the LRM processor is the processor in SMM mode, SMIACT# will be inactive and remain so until that processor becomes the MRM.
- The Pentium processor is capable of supporting an SMM I/O instruction restart. This feature is automatically disabled following RESET. To enable the I/O instruction restart feature, set bit 9 of the TR12 register to "1".
- The Pentium processor default SMM revision identifier has a value of 2 when the SMM I/O instruction restart feature is enabled.
- SMI# is NOT recognized by the processor in the shutdown state.

24.2 System Management Interrupt Processing

The system interrupts the normal program execution and invokes SMM by generating a System Management Interrupt (SMI#) to the processor. The processor will service the SMI# by executing the following sequence. See Figure 24-1.

1. Wait for all pending bus cycles to complete and EWBE# to go active.
2. The processor asserts the SMIACT# signal while in SMM indicating to the system that it should enable the SMRAM.
3. The processor saves its state (context) to SMRAM, starting at address location SMBASE + 0FFFFH, proceeding downward in a stack-like fashion.
4. The processor switches to the System Management Mode processor environment (a pseudo-real mode).
5. The processor will then jump to the absolute address of SMBASE + 8000H in SMRAM to execute the SMI handler. This SMI handler performs the system management activities.
6. The SMI handler will then execute the RSM instruction which restores the processor's context from SMRAM, deasserts the SMIACT# signal, and then returns control to the previously interrupted program execution.

Note: The default SMBASE value following RESET is 30000H.

Figure 24-1. Basic SMI# Interrupt Service

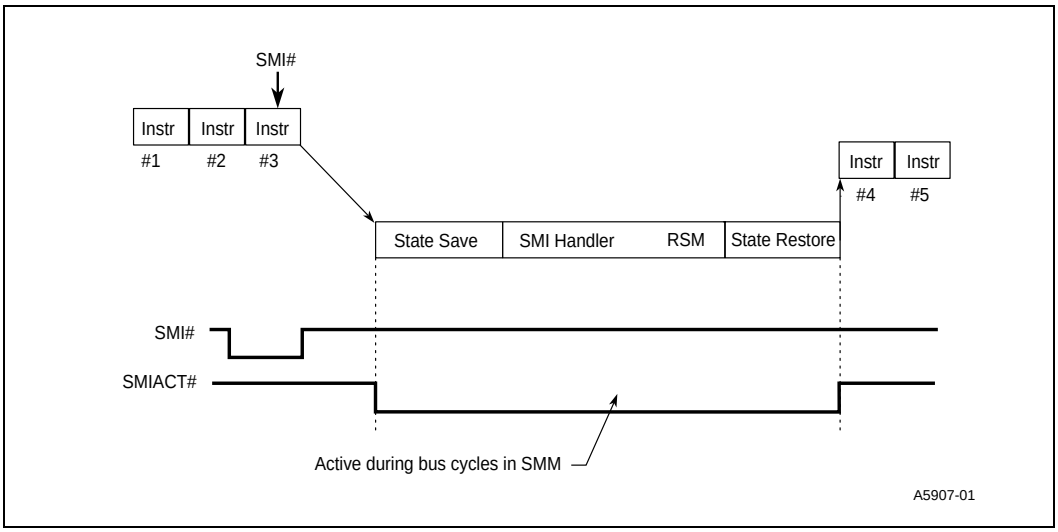
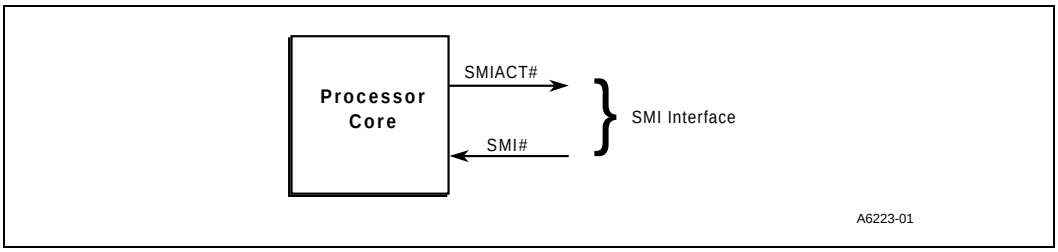


Figure 24-2 describes the System Management Interrupt hardware interface which consists of the SMI# interrupt request input and the SMIACT# output used by the system to decode the SMRAM.

Figure 24-2. Basic SMI# Hardware Interface



24.2.1 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, t28 and t29, must be met in order to guarantee recognition on 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.

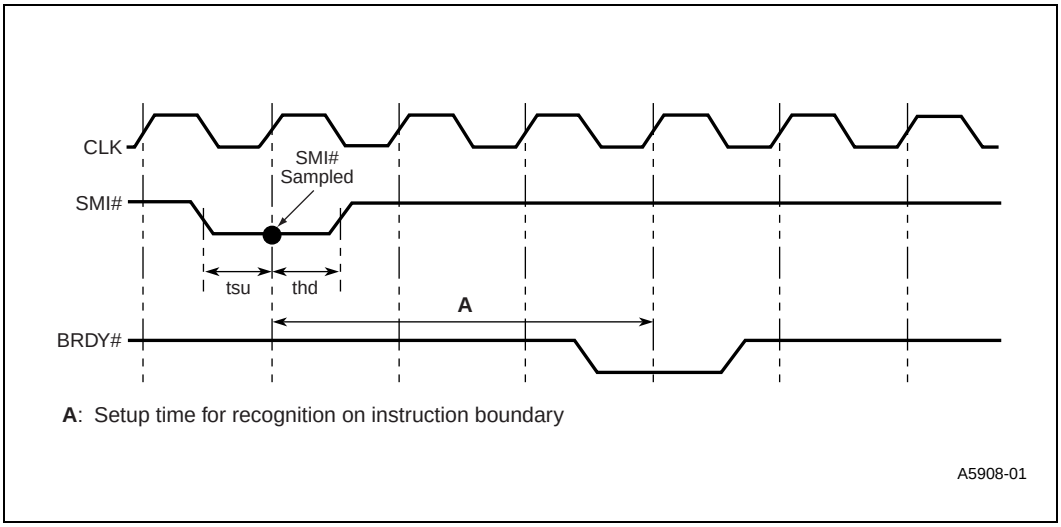
The SMI# signal is synchronized internally and must be asserted at least three CLK periods prior to asserting the BRDY# signal in order to guarantee recognition on a specific instruction boundary. See Figure 24-3.

The SMI# input must be held inactive for at least four clocks after it is asserted to reset the edge triggered logic. A subsequent SMI# might not be recognized if the SMI# input is not held inactive for at least four clocks after being asserted.

SMI#, like NMI, is not affected by the IF bit in the EFLAGS register and is recognized on an instruction boundary. 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.

Figure 24-3. SMI# Timing



24.2.1.1 SMI# Synchronization for I/O Instruction Restart

The SMI# signal is synchronized internally and must be asserted at least three CLK periods prior to asserting the BRDY# signal in order to guarantee recognition on a specific I/O instruction boundary. This is important for servicing an I/O trap with an SMI# handler. Due to the asynchronous nature of SMI# delivery with the APIC, it is impossible to synchronize the assertion of BRDY#. As a result, the SMM I/O instruction restart feature cannot be used when an SMI is delivered via the local APIC.

24.2.1.2 Dual Processing Considerations For SMI# Delivery

Although the SMM functions the same when the dual processor is inserted into Socket 7, the dual processor operation of the system must be carefully considered. Table 24-1 shows the four possible options for SMI# delivery depending on the SMM applications (mainly power management) the system has to support. There are implications to system design and the SMM handler. Note that for operation with the Dual processor and upgradability with a future upgrade processor, Option #3 is strongly recommended.

Table 24-1. Dual Processing SMI# Delivery Options

	SMI# Pins Tied Together	SMI# Pins NOT Tied Together
SMI# pins delivering SMI	Option #1 Both processors enter SMM.	Option #2 One processor enters SMM.
APIC delivering SMI	Option #3 One or Both processors enter SMM.	Option #4 One or Both processors enter SMM

Note: The I/O Instruction Restart Power Management feature should not be used when delivering the system management interrupt via the local APIC. Refer to the *Intel Architecture Software Developer's Manual, Volume 3* for additional details on I/O instruction restart.

Implications

1. SMI# pin delivery of SMI with the SMI# pins tied together: Any assertion of the SMI# pin will cause both the Primary and Dual processors to interrupt normal processing, enter SMM mode and start executing SMM code in their respective SMRAM spaces. In this case, using the I/O Instruction restart feature in Dual Processor mode will require additional system hardware (D/P# pin) and software (detection of which processor was the MRM when the SMI# pin was asserted) considerations.
2. SMI# pin delivery of SMI with the SMI# pins NOT tied together: Only the processor whose SMI# pin is asserted will handle SMM processing. It is possible that both the Primary and Dual processor will be doing SMM processing at the same time, especially if the I/O Instruction restart feature is being used. If I/O instruction restart is not supported, then it is possible to dedicate only one processor for SMM handling at any time.
3. APIC SMI# delivery of SMI with the SMI# pins tied together: This option is strongly recommended for operation with the Dual processor and upgradability with the Pentium OverDrive® processor. System Management Interrupts should be delivered via the APIC for DP systems, and may be delivered either via the APIC or the SMI# pin for turbo-upgraded systems. Either the Primary or Dual processor can be the assigned target for SMI# delivery and hence SMM handling. The SMM I/O instruction restart feature may be used in a uniprocessor system or in a system with a (with SMI# pin delivery of the interrupt), but the system must not use this feature when operating in dual processing mode (with APIC delivery of the interrupt).
4. APIC SMI# delivery of SMI with the SMI# pins NOT tied together: I/O Instruction Restart feature is not recommended when delivering SMI via the local APIC. Either the Primary or Dual processor can be the assigned target for SMI# delivery and hence SMM handling.

24.2.2 System Management Interrupt Via APIC

When SMI# is asserted (SMI# pin asserted low or APIC SMI# message) it causes the processor to invoke SMM.

24.2.3 SMI Active (SMIACT#)

SMIACT# indicates that the processor is operating in System Management Mode. The processor asserts SMIACT# in response to an SMI interrupt request on the SMI# pin or through the APIC message. SMIACT# is driven active for accesses only after the processor has completed all pending write cycles (including emptying the write buffers — EWBE# returned active by the system). SMIACT# will be asserted for all accesses in SMM beginning with the first access to SMRAM when the processor saves (writes) its state (or context) to SMRAM. SMIACT# is driven active for every access until the last access to SMRAM when the processor restores (reads) its state from SMRAM. The SMIACT# signal is used by the system logic to decode SMRAM.

The number of CLKs required to complete the SMM state save and restore is very dependent on system memory performance and the processor bus frequency.

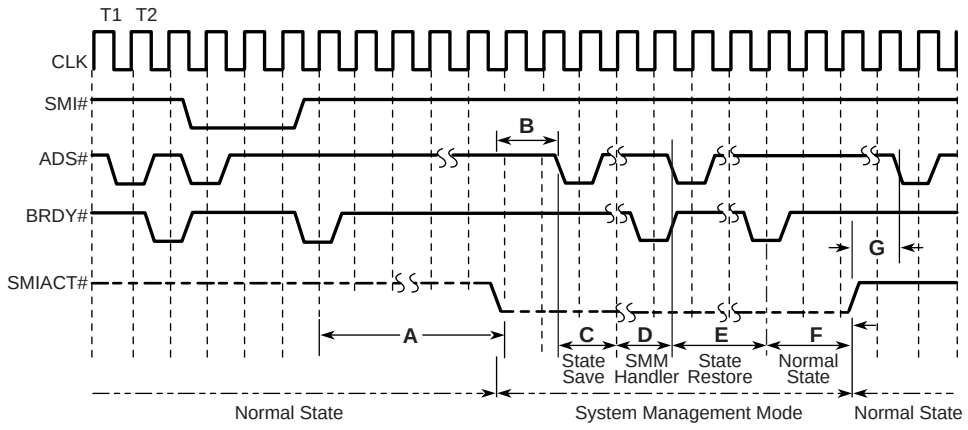
As shown in Figure 24-4, the approximate time required to enter an SMI handler routine for the Pentium processor (from the completion of the interrupted instruction) is given by:

Latency to beginning of SMI handler = A + B + C = ~184 CLKs

The approximate 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 = ~207 CLKs

Figure 24-4. SMIACT# Timing



A: Last BRDY# from non-SMM transfer to SMIACT# assertion: 2 CLKs minimum
B: SMIACT# assertion to first ADS# for SMM state save: 2 CLKs minimum
C: SMM state save (dependent on memory performance): Approximately 180 CLKs
D: SMM handler: User determined
E: SMM state restore (dependent on memory performance): Approximately 200 CLKs
F: Last RDY# from SMM transfer to deassertion of SMIACT#: 2 CLKs minimum
G: SMIACT# deassertion to first non-SMM ADS#: 5 CLKs minimum

A5909-01

24.2.3.1 Dual Processing Considerations for SMIACT#

When the processor is the only processor present, then it always drives the D/P# signal low. SMIACT# is asserted when the processor enters SMM and is deasserted only when the processor exits SMM.

When the Dual processor is also present, the D/P# signal toggles depending upon whether the Primary or Dual processor owns the bus (MRM). The SMIACT# pins may be tied together or be used separately to ensure SMRAM access by the correct processor.

Caution: If SMIACT# is used separately: the SMIACT# signal is only driven by the Primary or Dual processor when it is the MRM, so this signal must be qualified with the D/P# signal.

In a dual socket system, connecting the SMIACT# signals together on the Primary and Dual processor sockets is strongly recommended for dual processing operation.

In dual processing systems, SMIACT# may not remain low (e.g., may toggle) if both processors are not in SMM mode. he SMIACT# signal is asserted by either the Primary or Dual processor based on two conditions: the processor is in SMM mode and is the bus master (MRM). If one processor is executing in normal address space, the SMIACT# signal will go inactive when that processor is MRM. The LRM processor, even if in SMM mode, will not drive the SMIACT# signal low.

24.3 SMM — System Design Considerations

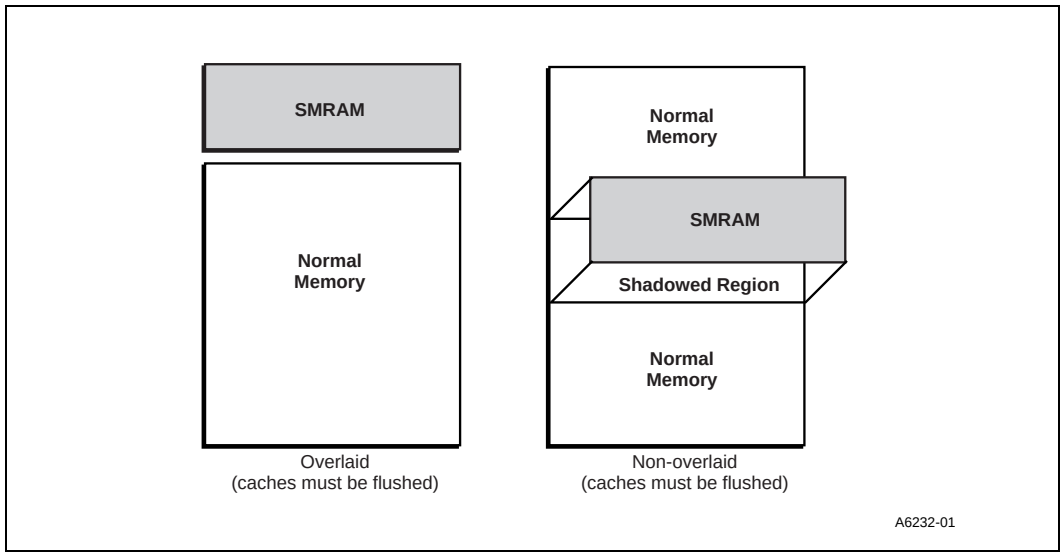
24.3.1 SMRAM Interface

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 SMBASE + 8000H to SMBASE + 0FFFFH 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 processor, 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.
5. Inquire cycles are permitted during SMM, but it is the responsibility of the system to ensure that any snoop writeback completes to the correct memory space, irrespective of the state of the SMIACT# pin. Specifically, if SMM is overlaid, and SMM space is non cacheable, then any snoop writeback cycle occurring during SMM must complete to system memory, even though SMIACT# will remain active.

   If an inquire cycle occurs after assertion of SMI# to the processor, but before SMIACT# is returned, note that SMIACT# could be returned at any point during the snoop writeback cycle. Depending on the timing of SMI# and the inquire cycle, SMIACT# could change states during the writeback cycle. Again, it is the responsibility of the system, if it supports snooping during SMM, to ensure that the snoop writeback cycle completes to the correct memory space, irrespective of the state of the SMIACT# pin.
6. It should also be noted that upon entering SMM, the branch target buffer (BTB) is not flushed and thus it is possible to get a speculative prefetch to an address outside of SMRAM address space due to branch predictions based on code executed prior to entering SMM. If this occurs, the system must still return BRDY# for each code fetch cycle.

Figure 24-5. SMRAM Location



24.3.2 Cache Flushes

The processor does not unconditionally writeback and invalidate its cache before entering SMM (this option is left to the system designer). If the SMRAM is in an area that is cacheable and overlaid on top of normal memory that is visible to the application or operating system (default), then it is necessary for the system to flush both the processor cache and any second level cache upon entering SMM. This may be accomplished by asserting flush the same time as the request to enter SMM (i.e., cache flushing during SMM entry is accomplished by asserting the FLUSH# pin at the same time as the request to enter SMM through SMI#). The priorities of FLUSH# and SMI# are such that the FLUSH# will be serviced first. To guarantee this behavior, the constraints on setup and hold timings on the interaction of FLUSH# and SMI# as specified for a processor should be obeyed. When the default SMRAM location is used, SMRAM is overlaid on top of system main memory (at SMBASE + 8000H to SMBASE + 0FFFFH).

In a system where FLUSH# and SMI# pins are synchronous and setup/hold times are met, then the FLUSH# and SMI# pins may be asserted in the same clock. In asynchronous systems, the FLUSH# pin must be asserted at least one clock before the SMI# pin to guarantee that the FLUSH# pin is serviced first. Note that in systems that use the FLUSH# pin to write back and invalidate the cache contents before entering SMM, the processor prefetches at least one cache line in between the time the Flush Acknowledge special cycle is run and the recognition of SMI# and the driving of SMIACT# for SMRAM accesses. It is the obligation of the system to ensure that these lines are not cached by returning KEN# inactive.

If SMRAM is located in its own distinct memory space, which can be completely decoded with only the processor address signals, it is said to be non-overlaid. In this case, there is one new requirement for maintaining cache coherency. Refer to Table 24-2.

Table 24-2. Scenarios for Cache Flushes with Writeback Caches

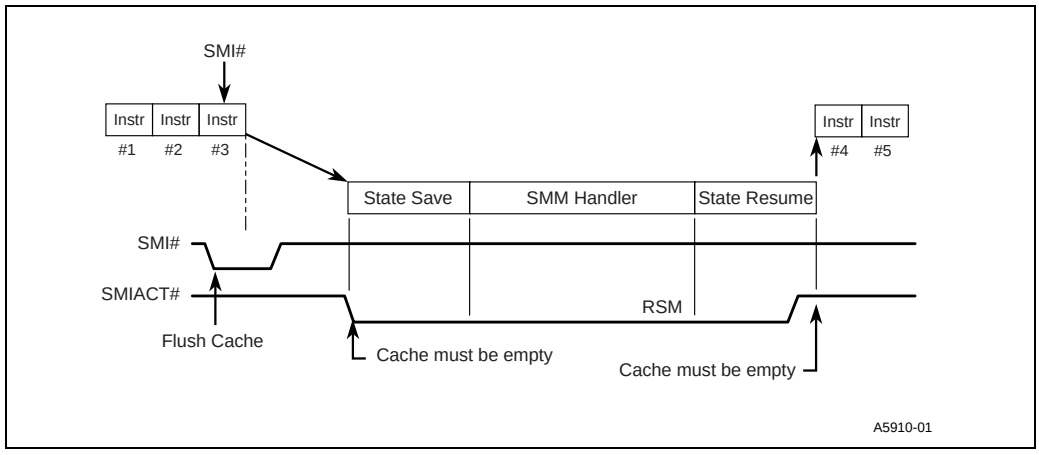
Is SMRAM overlapped with normal memory?	Is Normal Memory cacheable?	Is SMRAM cacheable?	Flush required during SMM entry?	Flush required during SMM exit?	Comments
No	No	No	No	No	
	No	WT	No	No	
	WT	No	No	No	
	WB	No	No†	No	†Snoop WBs must always go to normal memory space
	WT	WT	No	No	
	WB	WT	No†	No	†Snoop and Replacement WBs must go to normal memory space.
Yes	No	No	No	No	
	No	WT	No	Yes	
	WT	No	Yes	No	
	WB	No	Yes	No	
	WT	WT	Yes	Yes	
	WB	WT	Yes	Yes	

Note: Writeback cacheable SMRAM is not recommended. When flushing upon SMM exit, SMIACT# will be deasserted and may cause regular memory to be overwritten.

The processor implements writeback caches. Hence the performance hit due to flushing the cache for SMM execution can be more significant. Due to the writeback nature of the cache, flushing the cache has the following penalties:

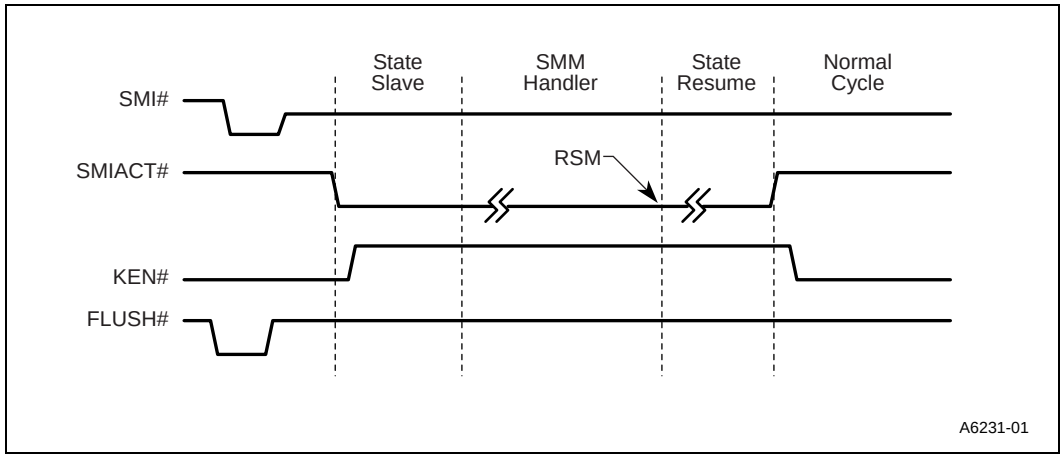
1. Before entry into SMM (when SMRAM is cacheable), the cache has to be flushed. Hence, all dirty lines need to be written back. This may cause a large number of bus cycles and increase SMM entry latency.
2. If the cache had to be flushed upon SMM exit, execution starts with cache miss 100%. The cache fill cycles reduce performance.

Figure 24-6. FLUSH# Mechanism During SMM with Overlay



The method suggested is shown in Figure 24-7.

Figure 24-7. Flush with Non-Cached SMM with Overlay



24.3.2.1 Dual Processing Considerations for Cache Flushes

Cache flushing during SMM exit is not possible while both the Primary and Dual processors are present due to the fact that it is not possible to clearly predict when the processor in SMM has exited. This is because the SMIACT# is not a static status indicator but only a bus cycle indicator for SMRAM accesses.

24.3.3 A20M# Signal

Systems based on the MS-DOS* operating system contain a feature that enables the processor 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 the processor 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 processor 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 processor 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 the above 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.

In addition to blocking the assertion of A20M# whenever SMIACT# is active, the system must also guarantee that A20M# is de-asserted at least one I/O clock prior to the assertion of SMIACT#. The processor may start the SMM state save as soon as SMIACT# is asserted. Processors faster than 200 MHz may not have enough time to recognize the de-assertion of A20M# before starting the SMM state save. As a result, this may cause the processor to start the first few cycles of the SMM state save with A20M# asserted. To avoid this, the system designer can use either of the following:

- When relocating the SMRAM above 1 Megabyte, ensure that the SMRAM does not coincide with any odd megabyte addresses. (Note that systems which use A20M# and SMM but do not relocate SMRAM above 1 Megabyte are not affected.)
- Use external logic to prevent the assertion of SMI to the processor until A20M# is de-asserted (and guarantee that A20M# remains de-asserted while in SMM). Note that the A20M# input must also meet setup and hold times in order to be recognized in a specific clock.

24.3.4 SMM and Second Level Write Buffers

Before the processor 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 processor is ready to begin writing an SMM state save to SMRAM, it asserts the SMIACT# signal for SMRAM references. SMIACT# may be driven active by the processor 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.

EWBE# can also be used to prevent the processor from asserting SMIACT# before write buffers are empty. The processor will wait for an active EWBE# before asserting SMIACT#.

24.4 Clock Control

24.4.1 Clock Generation

To understand the additional power management fears of the Pentium processor and how it manipulates the clock to conserve power, it is necessary to understand how the clock operates. The processor is capable of running internally at frequencies much higher than the bus speed via the various bus frequency settings. This allows simpler system design by lowering the clock speeds required in the external system. The high frequency internal clock relies on an internal Phase Lock Loop (PLL) to generate the two internal clock phases, “phase one” and “phase two.” Most external timing parameters are specified with respect to the rising edge of CLK. The PLL requires a constant frequency CLK input, and therefore the CLK input cannot be changed dynamically.

On the embedded Pentium processor, CLK provides the fundamental timing reference for the bus interface unit. The internal clock converter enhances all operations functioning out of the internal cache and/or operations not blocked by external bus accesses.

24.4.2 Stop Clock

The processor provides an interrupt mechanism, STPCLK#, that allows system hardware to control the power consumption of the processor by stopping the internal clock (output of the PLL) to the processor core in a controlled manner. This low-power state is called the Stop Grant state. The target for low-power mode supply current in the Stop Grant state is ~15% of normal I_{CC}.

When the processor 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, complete all outstanding writes, generate a Stop Grant bus cycle, and then stop the internal clock. At this point, the processor is in the Stop Grant state.

Note: If STPCLK# is asserted during RESET and continues to be held active after RESET is deasserted, the processor will execute one instruction before the STPCLK# interrupt is recognized. Execution of instructions will therefore stop on the second instruction boundary after the falling edge of RESET.

The processor cannot respond to a STPCLK# request from a HLDA state because it cannot generate a Stop Grant cycle.

The rising edge of STPCLK# will tell the processor 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 table reads. Among external interrupts, STPCLK# is the lowest priority.

24.4.2.1 STPCLK# Signal

STPCLK# is treated as a level triggered interrupt to the processor. This interrupt may be asserted asynchronously and is prioritized below all of the external interrupts. If asserted, the processor will recognize STPCLK# on the next instruction boundary, and then do the following:

1. Flush the instruction pipeline of any instructions waiting to be executed.
2. Wait for all pending bus cycles to complete and EWBE# to go active.

3. Drive a special bus cycle (Stop Grant bus cycle) to indicate that the clock is being stopped.
4. Enter low power mode.

STPCLK# is active low. To ensure STPCLK# recognition, the system must keep this signal active until the appropriate special cycle has been issued by the processor. To guarantee that every STPCLK# assertion, and subsequent deassertion and re-assertion, is recognized and thus will get a Stop Grant bus cycle response (which will also ensure that each deassertion of STPCLK# allows execution of at least one instruction), the system must meet the following requirements:

1. Hold STPCLK# active at least until the processor's Stop Grant cycle response has been completed by the system's BRDY# response.
2. STPCLK# must not be re-asserted until five clocks after the *last* of the following events:
   a. The processor's Stop Grant cycle has been completed by the system's BRDY# response.
   b. HITM# is deasserted. (This applies only if HITM# was asserted while waiting for one of the other two events listed here, *or* within two bus clocks of their completion.)
   c. EWBE# becomes active after it was sampled inactive at the last relevant BRDY#. A relevant BRDY# is one which ends *either* a stop-grant cycle **or** an external snoop writeback caused by HITM# being asserted as in case b) above.

Events b) and c) can in principle alternate indefinitely, continuing to delay STPCLK# deassertion recognition, if the system design allows that to happen.

Note that if a system is not relying on either a Stop Grant bus cycle response for every STPCLK# assertion, or for each deassertion of STPCLK# to allow execution of at least one instruction, these detailed requirements can be ignored. Though STPCLK# is asynchronous, setup and hold times may be met to ensure recognition on a specific clock.

The STPCLK# input must be driven high (not floated) in order to exit the Stop Grant state. Once STPCLK# is deasserted and the processor resumes execution, the processor is guaranteed to execute at least one instruction before STPCLK# is recognized again. To return to normal state, external hardware must deassert STPCLK#.

24.4.2.2 Dual Processing Considerations

The Primary and Dual processors may or may not tie their STPCLK# signals together. The decision is dependent on system specific processor power conservation needs. Connecting the STPCLK# signals on the Primary and Dual processors together is strongly recommended for operation with the Dual processor.

Tying the STPCLK# signals together causes both the Primary and Dual processors to eventually enter the Stop Grant state on assertion of STPCLK#. The system ceases processing until the STPCLK# signal is deasserted. In Dual Processor mode with the STPCLK# pins tied together, independent STPCLK# control of each processor is not possible. Both the Primary processor and Dual processor will go into the Stop Grant state independently, and will each generate a Stop Grant special bus cycle.

Note: In a dual processing system where STPCLK# is tied to both the primary and dual processors, the system expects to see two Stop Grant Bus Cycles after STPCLK# is asserted. FLUSH# should not be asserted between the time STPCLK# is asserted and the completion of the second Stop Grant

Bus Cycle. If FLUSH# is asserted during this interval, the system may not see the second Stop Grant Bus Cycle until after STPCLK# is deasserted.

Not tying the STPCLK# signals together gives the flexibility to control either or both the processors' power consumption based on the system performance required. External logic would be required to control this signal to each processor in a DP system.

24.4.3 Stop Grant Bus Cycle

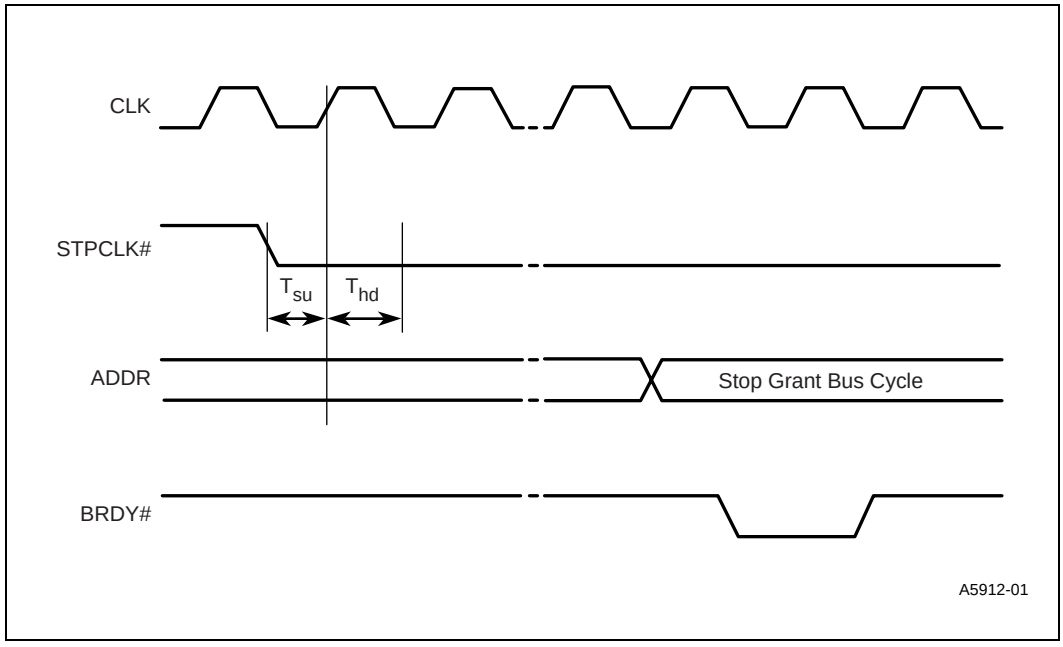
A special Stop Grant bus cycle will be driven to the bus after the processor recognizes the STPCLK# interrupt. The definition of this bus cycle is the same as the HALT cycle definition for the standard Intel486 microprocessor architecture, with the exception that the Stop Grant bus cycle drives the value 0000 0010H on the address pins. In a Dual Processor system, with both STPCLK# signals tied together, two stop grant cycles will occur in a row. The system hardware must acknowledge the Stop Grant cycle by returning BRDY#. The processor will not enter the Stop Grant state until BRDY# has been returned.

The Stop Grant Bus cycle consists of the following signal states: M/IO# = 0, D/C# = 0, W/R# = 1, Address Bus = 0000 0010H (A4 = 1), BE7#–BE0# = 1111 1011, Data bus = undefined.

Note: When operating in dual processing mode, and the STPCLK# signals are tied together, both the Primary processor and Dual processor will go into the Stop Grant state independently, and will each generate a Stop Grant special bus cycle. The system must return BRDY# for both of the special bus cycles.

The latency between a STPCLK# request and the Stop Grant bus cycle is dependent on the current instruction, the amount of data in the processor write buffers, and the system memory performance. Refer to Figure 24-8.

Figure 24-8. Entering Stop Grant State



24.4.4 Pin State During Stop Grant

During the Stop Grant state, most output and input/output signals of the microprocessor will be held at their previous states (the level they held when entering the Stop Grant state). See Table 24-3. However, the data bus and data parity pins will be floated. In response to HOLD being driven active during the Stop Grant state (when the CLK input is running), the processor will generate HLDA and three-state all output and input/output signals that are three-stated during the HOLD/HLDA state. After HOLD is deasserted, all signals will return to their states prior to the HOLD/HLDA sequence.

Table 24-3. Pin State During Stop Grant Bus State

Signal	Type	State
A31–A3	I/O	Previous State
D63–D0	I/O	Floated
BE7#–BE0#	O	Previous State
DP7–DP0	I/O	Floated
W/R#, D/C#, M/IO#	O	Previous State
ADS#, ADSC#	O	Inactive
LOCK#	O	Inactive
BREQ	O	Previous State
HLDA	O	As per HOLD
FERR#	O	Previous State
PCHK#	O	Previous State
PWT, PCD	O	Previous State
SMIACT#	O	Previous State

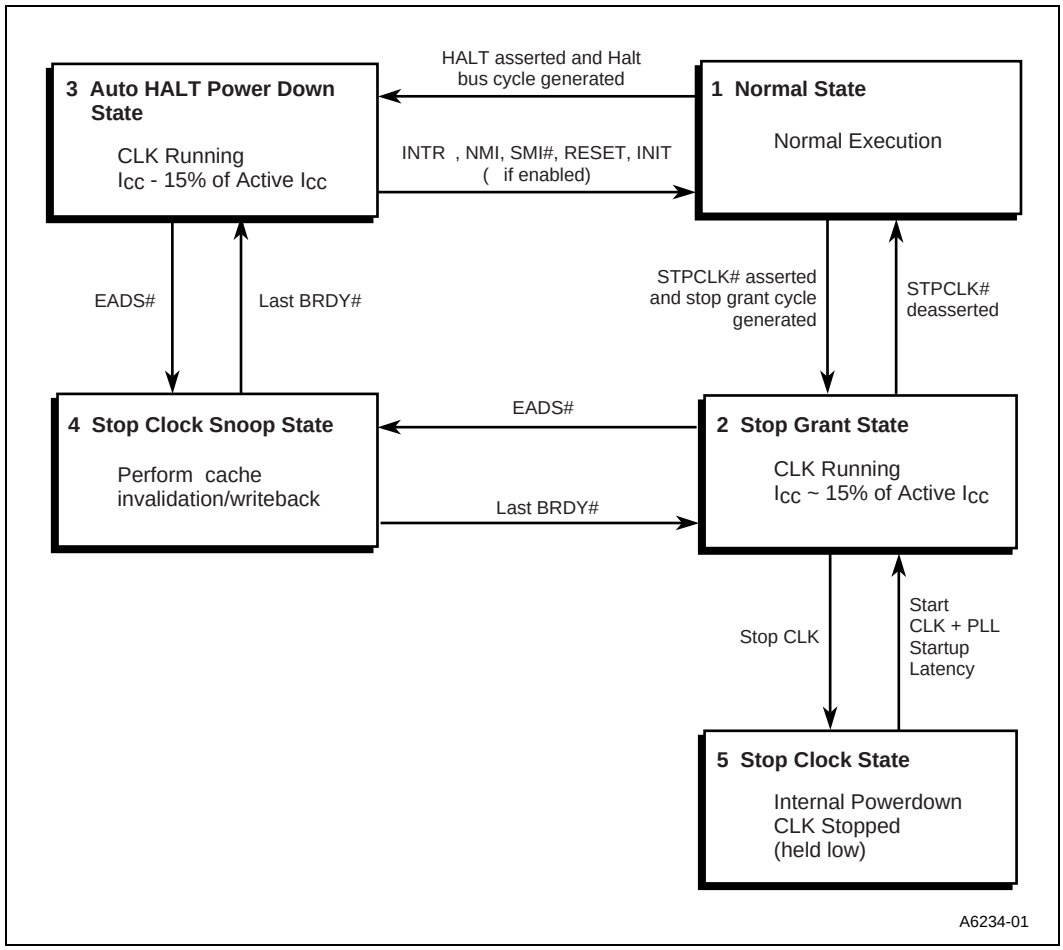
In order to achieve the lowest possible power consumption during the Stop Grant state, the system designer must ensure the input signals with pull-up resistors are not driven low and the input signals with pull-down resistors are not driven high.

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. Data pins should be driven low to achieve the lowest power consumption. Pull down resistors or bus keepers are needed to minimize the leakage current.

24.4.5 Clock Control State Diagram

Figure 24-9 shows the state descriptions and the state transitions for the clock control architecture.

Figure 24-9. Stop Clock State Machine



A Flush State can be entered from states 1, 2 and 3 by asserting the FLUSH# input signal. The flush state is exited (e.g., the processor returns to the state from which it came) when the Flush Acknowledge Special Bus Cycle is issued by the processor.

24.4.5.1 Normal State — State 1

This is the normal operating state of the processor.

24.4.5.2 Stop Grant State — State 2

The Stop Grant state (~15% of normal state I_{CC}) 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 BRDY# is returned, the processor is in this state. The processor returns to the normal execution state in approximately 10 clock periods after STPCLK# has been deasserted.

For minimum processor power consumption, all other input pins should be driven to their inactive level while the processor is in the Stop Grant state. A RESET will bring the processor from the Stop Grant state to the normal state (note: unless STPCLK# is also deasserted, an active RESET will only bring the processor out of the Stop Grant state for a few cycles). The processor will recognize the inputs required for maintaining cache coherency (e.g., HOLD, AHOLD, BOFF#, and EADS# for cache invalidations and snoops) as explained later in this section. The processor will not recognize any other inputs while in the Stop Grant state. Input signals to the processor will not be recognized until 1 CLK after STPCLK# is deasserted.

While in the Stop Grant state, the processor will latch transitions on the external interrupt signals (SMI#, NMI, INTR, FLUSH#, R/S#, and INIT). All of these interrupts are taken after the deassertion of STPCLK# (e.g., upon re-entering the normal state). The Pentium processor requires INTR to be held active until the processor issues an interrupt acknowledge cycle in order to guarantee recognition.

The processor will generate a Stop Grant bus cycle only when entering that state from the normal state. When the processor enters the Stop Grant state from the Stop Clock Snoop state, the processor will not generate a Stop Grant bus cycle.

24.4.5.3 Auto Halt Powerdown State — State 3

The execution of a HLT instruction will also cause the Pentium processor to automatically enter the Auto HALT Power Down state where I_{CC} will be ~15% of I_{CC} in the Normal state. The processor will issue a normal HALT bus cycle when entering this state. The processor will transition to the normal state upon the occurrence of INTR, NMI, SMI#, RESET, or INIT.

A FLUSH# event during the Auto HALT power down state will be latched and acted upon while in this state.

STPCLK# is not recognized by the processor while in the Auto HALT Powerdown state. The system can generate a STPCLK# while the processor is in the Auto HALT Powerdown state, but the processor will only service this interrupt if the STPCLK# pin is still asserted when the Pentium returns to the normal state.

While in Auto HALT Powerdown state, the processor will only recognize the inputs required for maintaining cache coherency (e.g., HOLD, AHOLD, BOFF#, and EADS# for cache invalidations and snoops) as explained later in this section.

24.4.5.4 Stop Clock Snoop State (Cache Invalidations) — State 4

When the processor is in the Stop Grant state or the Auto HALT Powerdown state, the processor will recognize HOLD, AHOLD, BOFF# and EADS# for cache invalidation/writebacks. When the system asserts HOLD, AHOLD, or BOFF#, the processor will float the bus accordingly. When the system then asserts EADS#, the processor will transparently enter the Stop Clock Snoop state and perform the required cache snoop cycle. It will then re-freeze the clock to the processor core and return to the previous state. The processor does not generate the Stop Grant bus cycle or HALT special cycle when it returns to the previous state.

24.4.5.5 Stop Clock State — State 5

Stop Clock state (~ 1% of normal state I_{CC}) is entered from the Stop Grant state by stopping the CLK input. Note: the CLK must be held at a logic low while stopped. None of the processor input signals should change state while the CLK input is stopped. Any transition on an input signal (with the exception of INTR) before the processor 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 processor issues an interrupt acknowledge bus cycle, it will be serviced in the normal manner once the clock has been restarted. The system design must ensure the processor is in the correct state prior to asserting cache invalidation or interrupt signals to the processor.

While the processor is in the Stop Clock state, all pins with static pullups or pulldowns must be driven to their appropriate values as specified in the datasheet.

During the Stop Clock state the processor input frequency may be changed to any frequency between the minimum and maximum frequency listed in the AC timing specifications found in the datasheet. To exit out of the Stop Clock state, the CLK input must be restarted and remain at a constant frequency for a minimum of 1 ms. The PLL requires this amount of time to properly stabilize. After the PLL stabilizes, the processor will return to Stop Grant state and the STPCLK# signal may be deasserted to take the processor out of Stop Grant state and back to the Normal state.

In order to realize the maximum power reduction while in the Stop Clock state, PICCLK and TCK should also be stopped. These clock inputs have the same restarting restrictions as CLK. The local APIC cannot be used while in the Stop Clock state since it also uses the system clock, CLK.

Warning: The Stop Clock state feature cannot be used in dual processing or functional redundancy checking modes because there is no way to re-synchronize the internal clocks of the two processors.