3.0 IBM 6x86MX MICROPROCESSOR BUS INTERFACE

The signals used in the IBM 6x86MX CPU bus interface are described in this chapter. Figure 3-1 shows the signal directions and the major signal groupings. A description of each signal and their reference to the text are provided in Table 3-1 (Page 3-2).

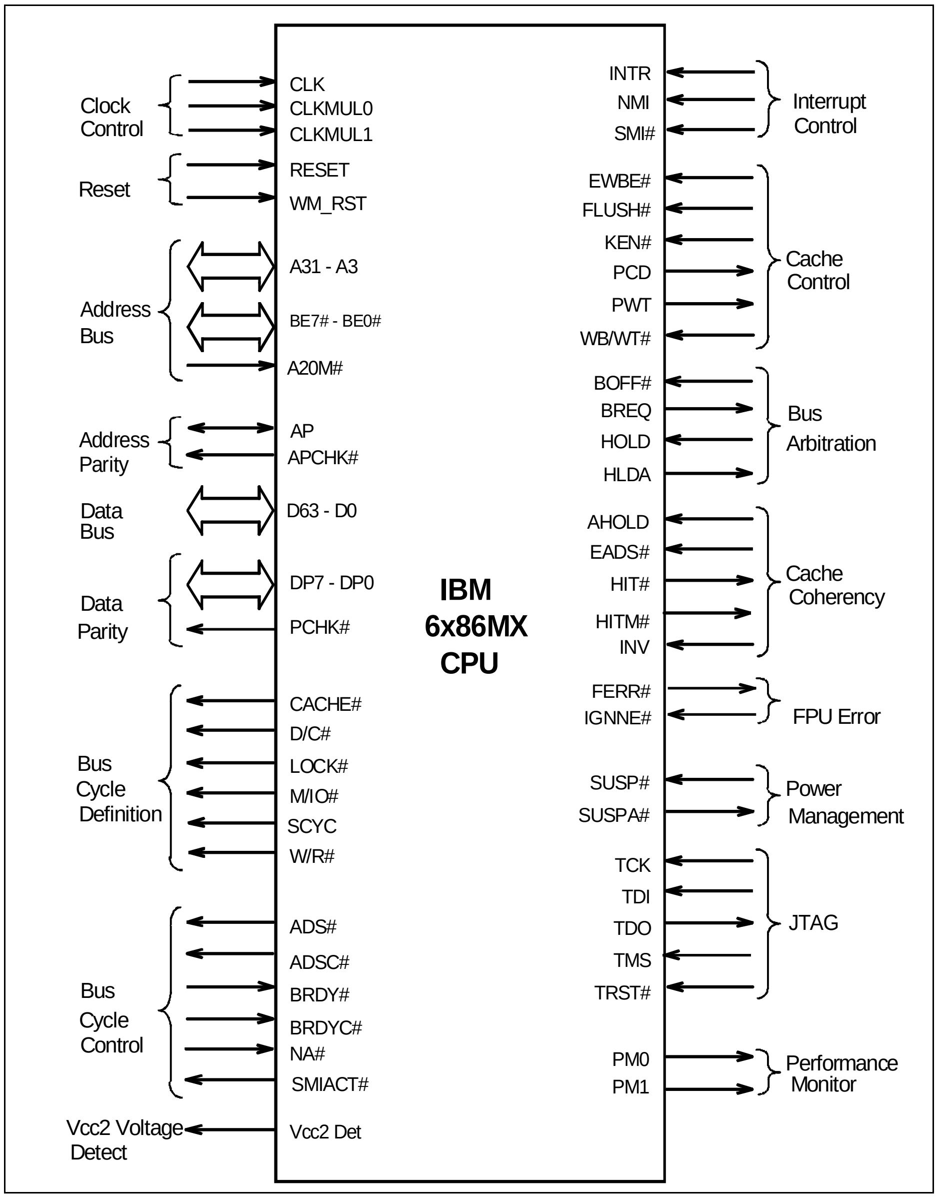


Figure 3-1. IBM 6x86MX CPU Functional Signal Groupings

3.1 Signal Description Table

The Signal Summary Table (Table 3-1) describes the signals in their active state unless otherwise mentioned. Signals containing slashes (/) have logic levels defined as “1/0.” For example the signal W/R#, is defined as write when W/R#=1, and as read when W/R#=0. Signals ending with a “#” character are active low.

Table 3-1. IBM 6x86MX CPU Signals Sorted by Signal Name

Signal Name	Description	I/O	Reference
A20M#	**A20 Mask** causes the CPU to mask (force to 0) the A20 address bit when driving the external address bus or performing an internal cache access. A20M# is provided to emulate the 1 MByte address wrap-around that occurs on the 8086. Snoop addressing is not affected.	Input	Page 3-9
A31-A3	The **Address Bus**, in conjunction with the Byte Enable signals (BE7#-BE0#), provides addresses for physical memory and external I/O devices. During cache inquiry cycles, A31-A5 are used as inputs to perform cache line invalidations.	3-state I/O	Page 3-9
ADS#	**Address Strobe** begins a memory/I/O cycle and indicates the address bus (A31-A3, BE7#-BE0#) and bus cycle definition signals (CACHE#, D/C#, LOCK#, M/IO#, PCD, PWT, SCYC, W/R#) are valid.	Output	Page 3-13
ADSC#	**Cache Address Strobe** performs the same function as ADS#.	Output	Page 3-13
AHOLD	**Address Hold** allows another bus master access to the IBM 6x86MX CPU address bus for a cache inquiry cycle. In response to the assertion of AHOLD, the CPU floats AP and A31-A3 in the following clock cycle.	Input	Page 3-18
AP	**Address Parity** is the even parity output signal for address lines A31-A5 (A4 and A3 are excluded). During cache inquiry cycles, AP is the even-parity input to the CPU, and is sampled with EADS# to produce correct parity check status on the APCHK# output.	3-state I/O	Page 3-10
APCHK#	**Address Parity Check Status** is asserted during a cache inquiry cycle if an address bus parity error has been detected. APCHK# is valid two clocks after EADS# is sampled active. APCHK# will remain asserted for one clock cycle if a parity error is detected.	Output	Page 3-10
BE7#-BE0#	The **Byte Enables**, in conjunction with the address lines, determine the active data bytes transferred during a memory or I/O bus cycle.	3-state I/O	Page 3-9
BOFF#	**Back-Off** forces the IBM 6x86MX CPU to abort the current bus cycle and relinquish control of the CPU local bus during the next clock cycle. The IBM 6x86MX CPU enters the bus hold state and remains in this state until BOFF# is negated.	Input	Page 3-16
BRDY#	**Burst Ready** indicates that the current transfer within a burst cycle, or the current single transfer cycle, can be terminated. The IBM 6x86MX CPU samples BRDY# in the second and subsequent clocks of a bus cycle. BRDY# is active during address hold states.	Input	Page 3-13
BRDYC#	**Cache Burst Ready** performs the same function as BRDY# and is logically ORed with BRDY# within the IBM 6x86MX CPU.	Input	Page 3-13

Table 3-1. IBM 6x86MX CPU Signals Sorted by Signal Name (Continued)

Signal Name	Description	I/O	Reference
BREQ	**Bus Request** is asserted by the IBM 6x86MX CPU when an internal bus cycle is pending. The IBM 6x86MX CPU always asserts BREQ, along with ADS#, during the first clock of a bus cycle. If a bus cycle is pending, BREQ is asserted during the bus hold and address hold states. If no additional bus cycles are pending, BREQ is negated prior to termination of the current cycle.	Output	Page 3-16
CACHE#	**Cacheability Status** indicates that a read bus cycle is a potentially cacheable cycle; or that a write bus cycle is a cache line write-back or line replacement burst cycle. If CACHE# is asserted for a read cycle and KEN# is asserted by the system, the read cycle becomes a cache line fill burst cycle.	Output	Page 3-11
CLK	**Clock** provides the fundamental timing for the IBM 6x86MX CPU. The frequency of the IBM 6x86MX CPU input clock determines the operating frequency of the CPU's bus. External timing is defined referenced to the rising edge of CLK.	Input	Page 3-7

CLKMUL1-
CLKMUL0
	The **Clock Multiplier** inputs are sampled during RESET to determine the IBM 6x86MX CPU core operating frequency.
	If = 00 core/bus ratio is 2.5
	If = 01 core/bus ratio is 3.0
	If = 10 core/bus ratio is 2.0 (default)
	If = 11 core/bus ratio is 3.5
		Input	Page 3-7

D63-D0	**Data Bus** signals are three-state, bi-directional signals which provide the data path between the IBM 6x86MX CPU and external memory and I/O devices. The data bus is only driven while a write cycle is active (state=T2). The data bus is floated when DHOLD is asserted.	3-state I/O	Page 3-10
D/C#	**Data/Control Status**. If high, indicates that the current bus cycle is an I/O or memory data access cycle. If low, indicates a code fetch or special bus cycle such as a halt, prefetch, or interrupt acknowledge bus cycle. D/C# is driven valid in the same clock as ADS# is asserted.	Output	Page 3-11
DP7-DP0	**Data Parity** signals provide parity for the data bus, one data parity bit per data byte. Even parity is driven on DP7-DP0 for all data write cycles. DP7-DP0 are read by the IBM 6x86MX CPU during read cycles to check for even parity. The data parity bus is only driven while a write cycle is active (state=T2).	3-state I/O	Page 3-10
EADS#	**External Address Strobe** indicates that a valid cache inquiry address is being driven on the IBM 6x86MX CPU address bus (A31-A5) and AP. The state of INV at the time EADS# is sampled active determines the final state of the cache line. A cache inquiry cycle using EADS# may be run while the IBM 6x86MX CPU is in the address hold or bus hold state.	Input	Page 3-18
EWBE#	**External Write Buffer Empty** indicates that there are no pending write cycles in the external system. EWBE# is sampled only during I/O and memory write cycles. If EWBE# is negated, the IBM 6x86MX CPU delays all subsequent writes to on-chip cache lines in the "exclusive" or "modified" state until EWBE# is asserted.	Input	Page 3-15

Table 3-1. IBM 6x86MX CPU Signals Sorted by Signal Name (Continued)

Signal Name	Description	I/O	Reference
FERR#	**FPU Error Status** indicates an unmasked floating point error has occurred. FERR# is asserted during execution of the FPU instruction that caused the error. FERR# does not float during bus hold states.	Output	Page 3-19
FLUSH#	**Cache Flush** forces the IBM 6x86MX CPU to flush the cache. External interrupts and additional FLUSH# assertions are ignored during the flush. Cache inquiry cycles are permitted during the flush.	Input	Page 3-15
HIT#	**Cache Hit** indicates that the current cache inquiry address has been found in the cache (modified, exclusive or shared states). HIT# is valid two clocks after EADS# is sampled active, and remains valid until the next cache inquiry cycle.	Output	Page 3-18
HITM#	**Cache Hit Modified Data** indicates that the current cache inquiry address has been found in the cache and dirty data exists in the cache line (modified state). The IBM 6x86MX CPU does not accept additional cache inquiry cycles while HITM# is asserted. HITM# is valid two clocks after EADS#.	Output	Page 3-18
HLDA	**Hold Acknowledge** indicates that the IBM 6x86MX CPU has responded to the HOLD input and relinquished control of the local bus. The IBM 6x86MX CPU continues to operate during bus hold as long as the on-chip cache can satisfy bus requests.	Output	Page 3-17
HOLD	**Hold Request** indicates that another bus master has requested control of the CPU's local bus.	Input	Page 3-16
IGNNE#	**Ignore Numeric Error** forces the IBM 6x86MX CPU to ignore any pending unmasked FPU errors and allows continued execution of floating point instructions.	Input	Page 3-19
INTR	**Maskable Interrupt** forces the processor to suspend execution of the current instruction stream and begin execution of an interrupt service routine. The INTR input can be masked (ignored) through the IF bit in the Flags Register.	Input	Page 3-14
INV	**Invalidate Request** is sampled with EADS# to determine the final state of the cache line in the case of a cache inquiry hit. An asserted INV directs the processor to change the state of the cache line to "invalid". A negated INV directs the processor to change the state of the cache line to "shared."	Input	Page 3-18
KEN#	**Cache Enable** allows the data being returned during the current cycle to be placed in the CPU's cache. When the IBM 6x86MX CPU is performing a cacheable code fetch or memory data read cycle (CACHE# asserted), and KEN# is sampled asserted, the cycle is transformed into a 32-byte cache line fill. KEN# is sampled with the first asserted BRDY# or NA# for the cycle.	Input	Page 3-15
LOCK#	**Lock Status** indicates that other system bus masters are denied access to the local bus. The IBM 6x86MX CPU does not enter the bus hold state in response to HOLD while LOCK# is asserted.	Output	Page 3-11
M/IO#	**Memory/IO Status**. If high, indicates that the current bus cycle is a memory cycle (read or write). If low, indicates that the current bus cycle is an I/O cycle (read or write, interrupt acknowledge, or special cycle).	Output	Page 3-11

Table 3-1. IBM 6x86MX CPU Signals Sorted by Signal Name (Continued)

Signal Name	Description	I/O	Reference
NA#	**Next Address** requests the next pending bus cycle address and cycle definition information. If either the current or next bus cycle is a locked cycle, a line replacement, a write-back cycle, or if there is no pending bus cycle, the IBM 6x86MX CPU does not start a pipelined bus cycle regardless of the state of NA#.	Input	Page 3-13
NMI	**Non-Maskable Interrupt Request** forces the processor to suspend execution of the current instruction stream and begin execution of an NMI interrupt service routine.	Input	Page 3-14
PCD	**Page Cache Disable** reflects the state of the PCD page attribute bit in the page table entry or the directory table entry. If paging is disabled, or for cycles that are not paged, the PCD pin is driven low. PCD is masked by the cache disable (CD) bit in CR0, and floats during bus hold states.	Output	Page 3-15
PCHK#	**Data Parity Check** indicates that a data bus parity error has occurred during a read operation. PCHK# is only valid during the second clock immediately after read data is returned to the IBM 6x86MX CPU (BRDY# asserted) and is inactive otherwise. Parity errors signaled by a logic low on PCHK# have no effect on processor execution.	Output	Page 3-10
PM0-PM1	**Performance Monitor** indicate an at least one overflow or event occurred in the associated Performance Monitor Register (0-1).	Output	Page 3-39
PWT	**Page Write-Through** reflects the state of the PWT page attribute bit in the page table entry or the directory table entry. PWT pin is negated during cycles that are not paged, or if paging is disabled. PWT takes priority over WB/WT#.	Output	Page 3-15
RESET	**Reset** suspends all operations in progress and places the IBM 6x86MX CPU into a reset state. Reset forces the CPU to begin executing in a known state. All data in the on-chip caches is invalidated.	Input	Page 3-7
SCYC	**Split Locked Cycle** indicates that the current bus cycle is part of a misaligned locked transfer. SCYC is defined for locked cycles only. A misaligned transfer is defined as any transfer that crosses an 8-byte boundary.	Output	Page 3-11
SMI#	**SMM Interrupt** forces the processor to save the CPU state to the top of SMM memory and to begin execution of the SMI service routine at the beginning of the defined SMM memory space. An SMI is a higher-priority interrupt than an NMI.	Input	Page 3-14
SMIACT#	**SMM Interrupt Active** indicates that the processor is operating in System Management Mode. SMIACT# does not float during bus hold states.	Output	Page 3-13
SUSP#	**Suspend Request** requests that the CPU enter suspend mode. SUSP# is ignored following RESET and is enabled by setting the SUSP bit in CCR2.	Input	Page 3-19
SUSPA#	**Suspend Acknowledge** indicates that the IBM 6x86MX CPU has entered low-power suspend mode. SUSPA# floats following RESET and is enabled by setting the SUSP bit in CCR2.	Output	Page 3-19
TCK	**Test Clock** (JTAG) is the clock input used by the IBM 6x86MX CPU's boundary scan (JTAG) test logic.	Input	Page 3-22
TDI	**Test Data In** (JTAG) is the serial data input used by the IBM 6x86MX CPU's boundary scan (JTAG) test logic.	Input	Page 3-22

Table 3-1. IBM 6x86MX CPU Signals Sorted by Signal Name (Continued)

Signal Name	Description	I/O	Reference
TDO	**Test Data Out** (JTAG) is the serial data output used by the IBM 6x86MX CPU's boundary scan (JTAG) test logic.	Output	Page 3-22
TMS	**Test Mode Select** (JTAG) is the control input used by the IBM 6x86MX CPU's boundary scan (JTAG) test logic.	Input	Page 3-22
TRST#	**Test Mode Reset** (JTAG) initializes the IBM 6x86MX CPU's boundary scan (JTAG) test logic.	Input	Page 3-22
VCC2DET	**Vcc2 Detect** is always driven low by the CPU to indicate that the IBM 6x86MX requires two different Vcc voltages.	Output	
WB/WT#	**Write-Back/Write-Through** is sampled during cache line fills to define the cache line write policy. If high, the cache line write policy is write-back. If low, the cache line write policy is write-through. (PWT forces write-through policy when PWT=1.)	Input	Page 3-16
WM_RST	**Warm Reset** forces the IBM 6x86MX CPU to complete the current instruction and then places the IBM 6x86MX CPU in a known state. Once WM_RST is sampled active by the CPU, the reset sequence begins on the next instruction boundary. WM_RST does not change the state of the configuration registers, the on-chip cache, the write buffers and the FPU registers. WM_RST is sampled during reset.	Input	Page 3-9
W/R#	**Write/Read Status**. If high, indicates that the current memory, or I/O bus cycle is a write cycle. If low, indicates that the current bus cycle is a read cycle.	Output	Page 3-11

3.2 Signal Descriptions

The following paragraphs provide additional information about the IBM 6x86MX CPU signals. For ease of this discussion, the signals are divided into 16 functional groups as illustrated in Figure 3-1 (Page 3-1).

3.2.1 Clock Control

The **Clock Input** (CLK) signal, supplied by the system, is the timing reference used by the IBM 6x86MX CPU bus interface. All external timing parameters are defined with respect to the CLK rising edge. The CLK signal enters the IBM 6x86MX CPU where it is multiplied to produce the IBM 6x86MX CPU internal clock signal. During power on, the CLK signal must be running even if CLK does not meet AC specifications.

The **Clock Multiplier** (CLKMUL1, CLMUL0) inputs are sampled during RESET to determine the CPU's core operating frequency (Table 3-2).

Table 3-2. Clock Control

CLKMUL1	CLKMUL0	CORE TO BUS CLOCK RATIO
0	0	2.5
0	1	3.0
1	0	2.0 (Default)
1	1	3.5

The CLKMUL pins have internal pull-up and pull down resistors to define the default ratio. Therefore the default setting indicates which mode the CPU will operate in if the CLKMUL are not driven and left floating.

3.2.2 Reset Control

The IBM 6x86MX CPU output signals are initialized to their reset states during the CPU reset sequence, as shown in Table 3-4 (Page 3-8). The signal states given in Table 3-4 assume that HOLD, AHOLD, and BOFF# are negated.

Asserting **RESET** suspends all operations in progress and places the IBM 6x86MX CPU in a reset state. RESET is an asynchronous signal but must meet specified setup and hold times to guarantee recognition at a particular clock edge.

On system power-up, RESET must be held asserted for at least 1 msec after Vcc and CLK have reached specified DC and AC limits. This delay allows the CPU's clock circuit to stabilize and guarantees proper completion of the reset sequence.

During normal operation, RESET must be asserted for at least 15 CLK periods in order to guarantee the proper reset sequence is executed. When RESET negates (on its falling edge), the pins listed in Table 3-3 determine if certain IBM 6x86MX CPU functions are enabled.

Table 3-3. Pins Sampled During RESET

SIGNAL NAME	DESCRIPTION
FLUSH#	If = 0, three-state test mode enabled.
WM_RST	If = 1, built-in self test initiated.

Table 3-4. Signal States During RESET

SIGNAL LINE	STATE	SIGNAL LINE	STATE
A20M#	Ignored	IGNNE#	Ignored
A31-A3	Undefined until first ADS#	INTR	Ignored
ADS#	1	INV	Ignored
ADSC#	1	KEN#	Ignored
AHOLD	Recognized	LOCK#	1
AP	Undefined until first ADS#	M/IO#	Undefined until first ADS#
APCHK#	1	NA#	Ignored
BE7#-BE0#	Undefined until first ADS#	NMI	Ignored
BOFF#	Recognized	PCD	Undefined until first ADS#
BRDY#	Ignored	PCHK#	1
BRDYC#	Ignored	PWT	Undefined until first ADS#
BREQ	0	RESET	1
CACHE#	Undefined until first ADS#	SCYC	Undefined until first ADS#
D(63-0)	Float	SMI#	Ignored
D/C#	Undefined until first ADS#	SMIACT#	1
DHOLD	Ignored	SUSP#	Ignored
DP(7-0)	Float	SUSPA#	Float
EADS#	Ignored	TCK	Recognized
EWBE#	Ignored	TDI	Recognized
FERR#	1	TDO	Responds to TCK, TDI, TMS, TRST#
FLUSH#	Initiates three-state test mode	TMS	Recognized
HIT#	1	TRST#	Recognized
HITM#	1	W/R#	Undefined until first ADS#
HLDA	Responds to HOLD	WB/WT#	Ignored
HOLD	Recognized	WM_RST	Initiates self-test

Warm Reset (WM_RST) allows the IBM 6x86MX CPU to complete the current instruction and then places the IBM 6x86MX CPU in a known state. WM_RST is an asynchronous signal, but must meet specified setup and hold times in order to guarantee recognition at a particular CLK edge. Once WM_RST is sampled active by the CPU, the reset sequence begins on the next instruction boundary.

WM_RST differs from RESET in that the contents of the on-chip cache, the write buffers, the configuration registers and the floating point registers contents remain unchanged.

Following completion of the internal reset sequence, normal processor execution begins even if WM_RST remains asserted. If RESET and WM_RST are asserted simultaneously, WM_RST is ignored and RESET takes priority. If WM_RST is asserted at the falling edge of RESET, built-in self test (BIST) is initiated.

3.2.3 Address Bus

The **Address Bus** (A31-A3) lines provide the physical memory and external I/O device addresses. A31-A5 are bi-directional signals used by the IBM 6x86MX CPU to drive addresses to both memory devices and I/O devices. During cache inquiry cycles the IBM 6x86MX CPU receives addresses from the system using signals A31-A5.

Using signals A31-A3, the IBM 6x86MX CPU can address a 4-GByte memory address space. Using signals A15-A3, the IBM 6x86MX CPU can address a 64-KByte I/O space through the processor's I/O ports. During I/O accesses, signals A31-A16 are driven low. A31-A3 float during bus hold and address hold states.

The **Byte Enable** (BE7#-BE0#) lines are bi-directional signals that define the valid data bytes within the 64-bit data bus. The correlation between the enable signals and data bytes is shown in Table 3-5.

Table 3-5. Byte Enable Signal to Data Bus Byte Correlation

BYTE ENABLE	CORRESPONDINGDA-TA BYTE
BE7#	D63-D56
BE6#	D55-D48
BE5#	D47-D40
BE4#	D39-D32
BE3#	D31-D24
BE2#	D23-D16
BE1#	D15-D8
BE0#	D7-D0

During a cache line fill, (burst read or “1+4” burst read) the IBM 6x86MX CPU expects data to be returned as if all data bytes are enabled, regardless of the state of the byte enables. BE7#-BE0# float during bus hold and byte enable hold states.

Address Bit 20 Mask (A20M#) is an active low input which causes the IBM 6x86MX CPU to mask (force low) physical address bit 20 when driving the external address bus or when performing an internal cache access. Asserting A20M# emulates the 1 MByte address wrap-around that occurs on the 8086. The A20 signal is never masked during write-back cycles, inquiry cycles, system management address space accesses or when paging is enabled, regardless of the state of the A20M# input.

3.2.4 Address Parity

Address Parity (AP) is a bi-directional signal which provides the parity associated with address lines A31-A5. (A4 and A3 are not included in the parity determination.) During IBM 6x86MX CPU generated bus cycles, while the address bus lines are driven, AP becomes an output supplying even address parity. During cache inquiry cycles, AP becomes an input and is sampled by EADS#. During cache inquiry cycles, even-parity must be placed on the AP line to guarantee an accurate result on the APCHK# (Address Parity Check Status) pin.

Address Parity Check Status (APCHK#) is driven active by the CPU when an address bus parity error has been detected for a cache inquiry cycle. APCHK# is asserted two clocks after EADS# is sampled asserted, and remains valid for one clock only. Address parity errors signaled by APCHK# have no effect on processor execution.

3.2.5 Data Bus

Data Bus (D63-D0) lines carry three-state, bi-directional signals between the IBM 6x86MX CPU and the system (i.e., external memory and I/O devices). The data bus transfers data to the IBM 6x86MX CPU during memory read, I/O read, and interrupt acknowledge cycles. Data is transferred from the IBM 6x86MX CPU during memory and I/O write cycles.

Data setup and hold times must be met for correct read cycle operation. The data bus is driven only while a write cycle is active.

3.2.6 Data Parity

The **Data Parity Bus** (DP7-DP0) provides and receives parity data for each of the eight data bus bytes (Table 3-6). The IBM 6x86MX CPU generates even parity on the bus during write cycles and accepts even parity from the system during read cycles. DP7-DP0 is driven only while a write cycle is active.

Table 3-6. Parity Bit to Data Byte Correlation

PARITY BIT	DATA BYTE
DP7	D63-D56
DP6	D55-D48
DP5	D47-D40
DP4	D39-D32
DP3	D31-D24
DP2	D23-D16
DP1	D15-D8
DP0	D7-D0

Parity Check (PCHK#) is asserted when a data bus parity error is detected. Parity is checked during code, memory and I/O reads, and the second interrupt acknowledge cycle. Parity is not checked during the first interrupt acknowledge cycle.

Parity is checked for only the active data bytes as determined by the active byte enable signals except during a cache line fill (burst read or “1+4” burst read). During a cache line fill, the IBM 6x86MX CPU assumes all data bytes are valid and parity is checked for all data bytes regardless of the state of the byte enables.

PCHK# is valid only during the second clock immediately after read data is returned to the IBM 6x86MX CPU (BRDY# asserted). At other times PCHK# is not active. Parity errors signaled by the assertion of PCHK# have no effect on processor execution.

3.2.7 Bus Cycle Definition

Each bus cycle is assigned a bus cycle type. The bus cycle types are defined by six three-state outputs: CACHE#, D/C#, LOCK#, M/IO#, SCYC, and W/R# as listed in Table 3-7 (Page 3-12).

These bus cycle definition signals are driven valid while ADS# is active. D/C#, M/IO#, W/R#, SCYC and CACHE# remain valid until the clock following the earliest of two signals: NA# asserted, or the last BRDY# for the cycle.

LOCK# continues asserted until after BRDY# is returned for the last locked bus cycle. The bus cycle definition signals float during bus hold states.

Cache Cycle Indicator (CACHE#) is an output that indicates that the current bus cycle is a potentially cacheable cycle (for a read), or indicates that the current bus cycle is a cache line write-back or line replacement burst cycle (for a write). If CACHE# is asserted for a read cycle and the KEN# input is returned active by the system, the read cycle becomes a cache line fill burst cycle.

Data/Control (D/C#) distinguishes between data and control operations. When high, this signal indicates that the current bus cycle is a data transfer to or from memory or I/O. When low, D/C# indicates that the current bus cycle involves a control function such as a halt, interrupt acknowledge or code fetch.

Bus Lock (LOCK#) is an active low output which, when asserted, indicates that other system bus masters are denied access to control of the CPU bus. The LOCK# signal may be explicitly activated during bus operations by including the LOCK prefix on certain instructions. LOCK# is also asserted during descriptor updates, page table accesses, interrupt acknowledge sequences and when executing the XCHG instruction. However, if the NO_LOCK bit in CCR1 is set, LOCK# is asserted only during page table accesses and interrupt acknowledge sequences. The IBM 6x86MX CPU does not enter the bus hold state in response to HOLD while the LOCK# output is active.

Memory/IO (M/IO#) distinguishes between memory and I/O operations. When high, this signal indicates that the current bus cycle is a memory read or memory write. When low, M/IO# indicates that the current bus cycle is an I/O read, I/O write, interrupt acknowledge cycle or special bus cycle.

Split Cycle (SCYC) is an active high output that indicates that the current bus cycle is part of a misaligned locked transfer. SCYC is defined for locked cycles only. A misaligned transfer is defined as any transfer that crosses an 8-byte boundary.

Write/Read (W/R#) distinguishes between write and read operations. When high, this signal indicates that the current bus cycle is a memory write, I/O write or a special bus cycle. When low, this signal indicates that the current cycle is a memory read, I/O read or interrupt acknowledge cycle.

Table 3-7. Bus Cycle Types

BUS CYCLE TYPE	M/IO#	D/C#	W/R#	CACHE#	LOCK#
Interrupt Acknowledge	0	0	0	1	0
Does not occur.	0	0	0	X	1
Does not occur.	0	0	1	X	0
Special Cycles: If BE(7-0)# = FEh: Shutdown If BE(7-0)# = FDh: Flush (INVD, WBINVD) If A4 = 0 and BE(7-0)# = FBh: Halt (HLT) If BE(7-0)# = F7h: Write-Back (WBINVD) If BE(7-0)# = EFh: Flush Acknowledge (FLUSH#) If A4 = 1 and BE(7-0)# = FBh: Stop Grant (SUSP#)	0	0	1	1	1
Does not occur.	0	1	X	X	0
I/O Data Read	0	1	0	1	1
I/O Data Write	0	1	1	1	1
Does not occur.	1	0	X	X	0
Cacheable Memory Code Read (Burst Cycle if KEN# Returned Active)	1	0	0	0	1
Non-cacheable Memory Code Read	1	0	0	1	1
Does not occur.	1	0	1	X	1
Locked Memory Data Read	1	1	0	1	0
Cacheable Memory Data Read (Burst Cycle if KEN# Returned Active)	1	1	0	0	1
Non-cacheable Memory Data Read	1	1	0	1	1
Locked Memory Write	1	1	1	1	0
Burst Memory Write (Writeback or Line Replacement)	1	1	1	0	1*
Single Transfer Memory Write	1	1	1	1	1

Note: X = Don't Care
*Note: LOCK# continues to be asserted during a write-back cycle that occurs following an aborted (BOFF# asserted) locked bus cycle.

3.2.8 Bus Cycle Control

The bus cycle control signals (ADS#, ADSC#, BRDY#, BRDYC#, NA#, and SMIACT#) indicate the beginning of a bus cycle and allow system hardware to control bus cycle termination timing and address pipelining.

Address Strobe (ADS#) is an active low output which indicates that the CPU has driven a valid address and bus cycle definition on the appropriate output pins. ADS# floats during bus hold states.

Cache Address Strobe (ADSC#) performs the same function as ADS#. ADSC# is used to interface directly to a secondary cache controller.

Burst Ready (BRDY#) is an active low input that is driven by the system to indicate that the current transfer within a burst cycle or the current single transfer bus cycle can be terminated. The CPU samples BRDY# in the second and subsequent clocks of a cycle. BRDY# is active during address hold states.

Cache Burst Ready (BRDYC#) performs the same function as BRDY# and is logically ORed with BRDY internally by the CPU. BRDYC# is used to interface directly to a secondary cache controller.

Next Address (NA#) is an active low input that is driven by the system to request the next pending bus cycle address and cycle definition information even though all data transfers for the current bus cycle are not complete. This new bus cycle is referred to as a “pipelined” cycle. If either the current or next bus cycle is a locked cycle, a line replacement, a write-back cycle or there is no pending bus cycle, the IBM 6x86MX CPU does not start a pipelined bus cycle regardless of the state of the NA# input.

System Management Mode Active (SMIACT#) behaves in one of two ways depending on which SMM mode is in effect.

In SL-Compatible Mode, SMIACT# is an active low output which indicates that the CPU is operating in System Management Mode. SMIACT# is asserted in response to the assertion of SMI# or due to execution of SMINT instruction. SMIACT# is also asserted during accesses to define SMM memory if SMAC bit CCR1 is set. The SMAC bit allows access to SMM memory while not in SMM mode and typically used for initialization purposes.

While in SL-compatible mode, when servicing an SMI# interrupt or SMINT instruction, SMIACT# remains asserted until a RSM instruction is executed. The RSM instruction causes the IBM 6x86MX CPU to exit SMM mode and negate the SMIACT# output. If a cache inquiry cycle occurs while SMIACT# is active, any resulting write-back cycle is issued with SMIACT# asserted. This occurs even thought the write-back cycle is intended for normal memory rather than SMM memory.

In Cyrix Enhanced Mode, SMIACT# does not indicate that the CPU is operating in system management mode. In Cyrix Enhanced Mode, SMIACT# is asserted for every SMM memory bus cycle and negated for every non-SMM memory cycle. In this mode SMIACT# follows the timing of MIO# and W/R#.

During RESET, the USE_SMI bit in CCR1 is cleared. While USE_SMI is zero, SMIACT# is always negated. SMIACT# does not float during bus hold states, except during Cyrix’s Enhanced SMM Operations.

3.2.9 Interrupt Control

The interrupt control signals (INTR, NMI, SMI#) allow the execution of the current instruction stream to be interrupted and suspended.

Maskable Interrupt Request (INTR) is an active high level-sensitive input which causes the processor to suspend execution of the current instruction stream and begin execution of an interrupt service routine. The INTR input can be masked (ignored) through the IF bit in the Flags Register.

When not masked, the IBM 6x86MX CPU responds to the INTR input by performing two locked interrupt acknowledge bus cycles. During the second interrupt acknowledge cycle, the IBM 6x86MX CPU reads the interrupt vector (an 8-bit value), from the data bus. The 8-bit interrupt vector indicates the interrupt level that caused generation of the INTR and is used by the CPU to determine the beginning address of the interrupt service routine. To assure recognition of the INTR request, INTR must remain active until the start of the first interrupt acknowledge cycle.

Non-Maskable Interrupt Request (NMI) is a rising edge sensitive input which causes the processor to suspend execution of the current instruction stream and begin execution of an NMI interrupt service routine. The NMI interrupt cannot be masked by the IF bit in the Flags Register. Asserting NMI causes an interrupt which internally supplies interrupt vector 2h to the CPU core. Therefore, external interrupt acknowledge cycles are not issued.

Once NMI processing has started, no additional NMIs are processed until an IRET instruction is executed, typically at the end of the NMI service routine. If NMI is re-asserted prior to execution of the IRET, one and only one NMI rising edge is stored and then processed after execution of the next IRET.

System Management Interrupt Request (SMI#) is an interrupt input with higher priority than the NMI input. Asserting SMI# forces the processor to save the CPU state to SMM memory and to begin execution of the SMI service routine.

SMI# behaves one of two ways depending on the IBM 6x86MX's SMM mode.

In SL-compatible mode SMI# is a falling edge sensitive input and is sampled on every rising edge of the processor input clock. Once SMI# servicing has started, no additional SMI# interrupts are processed until a RSM instruction is executed. If SMI# is reasserted prior to execution of a RSM instruction, one and only one SMI# falling edge is stored and then processed after execution of the next RSM.

In Cyrix enhanced SMM mode, SMI# is level sensitive, and nested SMI's are permitted under control of the SMI service routine. As a level sensitive input, software can process all SMI interrupts until all sources in the chipset have cleared. In enhanced mode, SMIACT# is asserted for every SMM memory bus cycle and negated for every non-SMM bus cycle.

In either mode, SMI# is ignored following reset and recognition is enabled by setting the USE_SMI bit in CCR1.

3.2.10 Cache Control

The cache control signals (EWBE#, FLUSH#, KEN#, PCD, PWT, WB/WT#) are used to indicate cache status and control caching activity.

External Write Buffer Empty (EWBE#) is an active low input driven by the system to indicate when there are no pending write cycles in the external system. The IBM 6x86MX CPU samples EWBE# during write cycles (I/O and memory) only. If EWBE# is not asserted, the processor delays all subsequent writes to on-chip cache lines in the “exclusive” or “modified” state until EWBE# is asserted. Regardless of the state of EWBE#, all writes to the on-chip cache are delayed until any previously issued external write cycle is complete. This ensures that external write cycles occur in program order and is referred to as “strong write ordering”. To enhance performance, “weak write ordering” may be allowed for specific address regions using the Address Region Registers (ARRs) and Region Control Registers (RCRs).

Cache Flush (FLUSH#) is a falling edge sensitive input that forces the processor to write-back all dirty data in the cache and then invalidate the entire cache contents. FLUSH# need only be asserted for a single clock but must meet specified setup and hold times to guarantee recognition at a particular clock edge.

Once FLUSH# is sampled active, the IBM 6x86MX CPU begins the cache flush sequence after completion of the current instruction. External interrupts and additional FLUSH# requests are ignored while the cache flush is in progress. However, cache inquiry cycles are permitted during the flush sequence. The IBM 6x86MX CPU issues a special flush acknowledge cycle to indicate completion of the flush sequence. If the processor is in a halt or shutdown state, FLUSH# is recognized and the IBM 6x86MX CPU returns to the halt or shutdown state following completion of the flush sequence. If FLUSH# is active at the falling edge of RESET, the processor enters three state test mode.

Cache Enable (KEN#) is an active low input which indicates that the data being returned during the current cycle is cacheable. When the IBM 6x86MX CPU is performing a cacheable code fetch or memory data read cycle and KEN# is sampled asserted, the cycle is transformed into a cache line fill (4 transfer burst cycle) or a “1+4” cache line fill. KEN# is sampled with the first asserted BRDY# or NA# for the cycle. I/O accesses, locked reads, system management memory accesses and interrupt acknowledge cycles are never cached.

Page Cache Disable (PCD) is an active high output that reflects the state of the PCD page attribute bit in the page table entry or the directory table entry. If paging is disabled or for cycles that are not paged, the PCD pin is driven low. PCD is masked by the cache disable (CD) bit in CR0 (driven high if CD=1) and floats during bus hold states.

Page Write Through (PWT) is an active high output that reflects the state of the PWT page attribute bit in the page table entry or the directory table entry. During non-paging cycles, and while paging is disabled the PWT pin is driven low. If PWT is asserted, PWT takes priority over the WB/WT# input. If PWT is asserted for either reads or writes, the cache line is saved in, or remains in, the shared (write-through) state. PWT floats during bus hold states.

The **Write-Back/Write-Through** (WB/WT#) input allows the system to define the write policy of the on-chip cache on a line-by-line basis. If WB/WT# is sampled high during a line fill cycle and PWT is low, the line is defined as write-back and is stored in the exclusive state. If WB/WT# is sampled high during a write to a write-through cache line (shared state) and PWT is low, the line is transitioned to write-back (exclusive state). If WB/WT# is sampled low or PWT is high, the line is defined as write-through and is stored in (line fill), or remains in (write), the shared state. Table 3-8 (Page 3-16) lists the effects of WB/WT# on the state of the cache line for various bus cycles.

Table 3-8. Effects of WB/WT# on Cache Line State

BUS CYCLE TYPE	PWT	WB/ WT#	WRITE POLICY	MESI STATE
Line Fill	0	0	Write-through	Shared
Line Fill	0	1	Write-back	Exclusive
Line Fill	1	x	Write-through	Shared
Memory Write (Note)	0	0	Write-through	Shared
Memory Write (Note)	0	1	Write-back	Exclusive
Memory Write (Note)	1	x	Write-through	Shared

Note: Only applies to memory writes to addresses that are currently valid in the cache.

3.2.11 Bus Arbitration

The bus arbitration signals (BOFF#, BREQ, HOLD, and HLDA) allow the IBM 6x86MX CPU to relinquish control of its local bus when requested by another bus master device. Once the processor has released its bus, the bus master device can then drive the local bus signals.

Back-Off (BOFF#) is an active low input that forces the IBM 6x86MX CPU to abort the current bus cycle and relinquish control of the CPU's local bus in the next clock. The IBM 6x86MX CPU responds to BOFF# by entering the bus hold state as listed in Table 3-9 (Page 3-17). The IBM 6x86MX CPU remains in bus hold until BOFF# is negated. Once BOFF# is negated, the IBM 6x86MX CPU restarts any aborted bus cycle in its entirety. Any data returned to the IBM 6x86MX CPU while BOFF# is asserted is ignored. If BOFF# is asserted in the same clock that ADS# is asserted, the IBM 6x86MX CPU may float ADS# while in the active low state.

Bus Request (BREQ) is an active high output asserted by the IBM 6x86MX CPU whenever a bus cycle is pending internally. The IBM 6x86MX CPU always asserts BREQ in the first clock of a bus cycle with ADS# as well as during bus hold and address hold states if a bus cycle is pending. If no additional bus cycles are pending, BREQ is negated prior to termination of the current cycle.

Bus Hold Request (HOLD) is an active high input used to indicate that another bus master requests control of the CPU's local bus. After recognizing the HOLD request and completing the current bus cycle or sequence of locked bus cycles, the IBM 6x86MX CPU responds by floating the local bus and asserting the hold acknowledge (HLDA) output. The bus remains granted to the requesting bus master until HOLD is negated. Once HOLD is sampled negated, the IBM 6x86MX CPU simultaneously drives the local bus and negates HLDA.

Hold Acknowledge (HLDA) is an active high output used to indicate that the IBM 6x86MX CPU has responded to the HOLD input and has relinquished control of its local bus. Table 3-9 (Page 3-17) lists the state of all the IBM 6x86MX CPU signals during a bus hold state. The IBM 6x86MX CPU continues to operate during bus hold states as long as the on-chip cache can satisfy bus requests. HLDA is asserted until HOLD is negated. Once HOLD is sampled negated, the IBM 6x86MX CPU simultaneously drives the local bus and negates HLDA.

Table 3-9. Signal States During Bus Hold

SIGNAL LINE	STATE
A20M#	Recognized internally
A31-A3	Float
ADS#	Float
ADSC#	Float
AHOLD	Ignored
AP	Float
APCHK#	Driven
BE7#-BE0#	Float
BOFF#	Recognized
BRDY#	Ignored
BRDYC#	Ignored
BREQ	Driven
CACHE#	Float
D/C#	Float
D63-D0	Float
DP7-DP0	Float
EADS#	Recognized
EWBE#	Recognized internally
FERR#	Driven
FLUSH#	Recognized
HIT#	Driven
HITM#	Driven
HLDA	Responds to HOLD
HOLD	Recognized
IGNNE#	Recognized internally
INTR	Recognized internally
INV	Recognized
KEN#	Ignored
LOCK#	Float
M/IO#	Float
NA#	Ignored
NMI	Recognized internally
PCD	Float
PCHK#	Driven
PWT	Float
RESET	Recognized
SCYC	Float
SMI#	Recognized
SMIACT#	Driven
SUSP#	Recognized
SUSPA#	Driven
TCK	Recognized
TDI	Recognized
TDO	Responds to TCK, TDI, TMS, TRST#
TMS	Recognized
TRST#	Recognized
W/R#	Float
WB/WT#	Ignored
WM_RST	Recognized

3.2.12 Cache Coherency

The cache coherency signals (AHOLD, EADS#, HIT#, HITM#, and INV) are used to initiate and monitor cache inquiry cycles. These signals are intended to be used to ensure cache coherency in a uni-processor environment only. Contact Cyrix for additional specifications on maintaining coherency in a multi-processor environment.

Address Hold Request (AHOLD) is an active high input which forces the IBM 6x86MX CPU to float A31-A3 and AP in the next clock cycle. While AHOLD is asserted, only the address bus is disabled. The current bus cycle remains active and can be completed in the normal fashion. The IBM 6x86MX CPU does not generate additional bus cycles while AHOLD is asserted except write-back cycles in response to a cache inquiry cycle.

External Address Strobe (EADS#) is an active low input used to indicate to the IBM 6x86MX CPU that a valid cache inquiry address is being driven on the IBM 6x86MX CPU address bus (A31-A5) and AP. The IBM 6x86MX CPU checks the on-chip cache for this address. If the address is present in the cache the HIT# signal is asserted. If the data associated with the inquiry address is “dirty” (modified state), the HITM# signal is also asserted. If dirty data exists, a write-back cycle is issued to update external memory with the dirty data. Additional cache inquiry cycles are ignored while HITM# is asserted.

The state of the INV pin at the time EADS# is sampled active determines the final state of the cache line. If INV is sampled high, the final state of the cache line is “invalid”. If INV is sampled low, the final state of the cache line is “shared”. A cache inquiry cycle using EADS# may be run while the IBM 6x86MX CPU is in either an address hold or bus hold state. The inquiry address must be driven by an external device.

Hit on Cache Line (HIT#) is an active low output used to indicate that the current cache inquiry address has been found in the cache (modified, exclusive or shared states). HIT# is valid two clocks after EADS# is sampled active, and remains valid until the next cache inquiry cycle.

Hit on Modified Data (HITM#) is an active low output used to indicate that the current cache inquiry address has been found in the cache and dirty data exists in the cache line (modified state). If HITM# is asserted, a write-back cycle is issued to update external memory. HITM# is valid two clocks after EADS# is sampled active, and remains asserted until two clocks after the last BRDY# of the write-back cycle is sampled active. The IBM 6x86MX CPU does not accept additional cache inquiry cycles while HITM# is asserted.

Invalidate Request (INV) is an active high input used to determine the final state of the cache line in the case of a cache inquiry hit. INV is sampled with EADS#. A logic one on INV directs the processor to change the state of the cache line to “invalid”. A logic zero on INV directs the processor to change the state of the cache line to “shared”.

3.2.13 FPU Error Interface

The FPU interface signals FERR# and IGNNE# are used to control error reporting for the on-chip floating point unit. These signals are typically used for a PC-compatible system implementation. For other applications, FPU errors are reported to the IBM 6x86MX CPU core through an internal interface.

Floating Point Error Status (FERR#) is an active low output asserted by the IBM 6x86MX CPU when an unmasked floating point error occurs. FERR# is asserted during execution of the FPU instruction that caused the error. FERR# does not float during bus hold states.

Ignore Numeric Error (IGNNE#) is an active low input which forces the IBM 6x86MX CPU to ignore any pending unmasked FPU errors and allows continued execution of floating point instructions. When IGNNE# is not asserted and an unmasked FPU error is pending, the IBM 6x86MX CPU only executes the following floating point instructions: FNCLEX, FNINIT, FNSAVE, FNSTCW, FNSTENV, and FNSTSW#. IGNNE# is ignored when the NE bit in CR0 is set to a 1.

3.2.14 Power Management Interface

The two power management signals (SUSP#, SUSPA#) allow the IBM 6x86MX CPU to enter and exit suspend mode. The IBM 6x86MX CPU also enters suspend mode as the result of executing a HALT instruction if the HALT bit is set in CCR2. Suspend mode circuitry forces the IBM 6x86MX CPU to consume minimal power while maintaining the entire internal CPU state.

Suspend Request (SUSP#) is an active low input which requests that the IBM 6x86MX CPU enter suspend mode. After recognition of an active SUSP# input, the IBM 6x86MX CPU completes execution of the current instruction, any pending decoded instructions and associated bus cycles, issues a stop grant bus cycle, and then asserts the SUSPA# output. SUSP# is ignored following RESET and is enabled by setting the SUSP bit in CCR2.

The **Suspend Acknowledge** (SUSPA#) output indicates that the IBM 6x86MX CPU has entered low-power suspend mode as the result of either assertion of SUSP# or execution of a HALT instruction. SUSPA# remains asserted until SUSP# is negated, or until an interrupt is serviced if suspend mode was entered via the HALT instruction. If SUSP# is asserted and then negated prior to SUSPA# assertion, SUSPA# may toggle state after SUSP# negates.

The IBM 6x86MX CPU accepts cache flush requests and cache inquiry cycles while SUSPA# is asserted. If FLUSH# is asserted, the CPU exits the low power state and services the flush request. After completion of all required write-back cycles, the CPU returns to the low power state. SUSPA# negates during the write-back cycles. Before issuing the write-back cycle, the CPU may execute several code fetches.

If AHOLD, BOFF# or HOLD is asserted while SUSPA# is asserted, the CPU exits the low power state in preparation for a cache inquiry cycle. After completion of any required write-back cycles resulting from the cache inquiry, the CPU returns to the low power state only if HOLD, BOFF# and AHOLD are negated. SUSPA# negates during the write-back cycle.

Table 3-10 (Page 3-21) lists the IBM 6x86MX CPU signal states for suspend mode when initiated by either SUSP# or the HALT instruction. SUSPA# is disabled (three-state) following RESET and is enabled by setting the SUSP bit in CCR2.

3.2.15 Performance Monitoring

The PM0 and PM1 pins are outputs that are associated with performance monitoring. These pins can be defined in two different ways.

If PM0, bit 9 in the Counter Event Control Register is set, the PM0 pin indicates an overflow has occurred; if reset, the PM0 pin indicates that a performance counter event has occurred. The PM1 pin operates in the same manner, but is controlled by PM1, bit 25.

The PM0 and PM1 pins indicate only that an event or overflow occurred at least once. More than one event or overflow can occur in the same CPU or external clock cycle.

Table 3-10. Signal States During Suspend Mode

SIGNAL LINE	SUSP# INITIATED/ HALT INITIATED	SIGNAL LINE	SUSP# INITIATED/ HALT INITIATED
A20M#	Ignored	INTR	Latched/Recognized
A31-A3	Driven	INV	Recognized
ADS#	1	KEN#	Ignored
ADSC#	1	LOCK#	1
AHOLD	Recognized	M/IO#	Driven
AP	Driven	NA#	Ignored
APCHK#	1	NMI	Latched/Recognized
BE7#-BE0#	Driven	PCD	Driven
BOFF#	Recognized	PCHK#	1
BRDY#	Ignored	PWT	Driven
BRDYC#	Ignored	RESET	Recognized
BREQ	0	SCYC	Driven
CACHE#	Driven	SMI#	Latched/Recognized
D/C#	Driven	SMIACT#	1
D63-D0	Float	SUSP#	0 / Recognized
DP7-DP0	Float	SUSPA#	0
EADS#	Recognized	TCK	Recognized
EWBE#	Ignored	TDI	Recognized
FERR#	1	TDO	Responds to TCK, TDI, TMS, TRST#
FLUSH#	Recognized	TMS	Recognized
HIT#	Driven	TRST#	Recognized
HITM#	1	W/R#	Driven
HLDA	Driven in response to HOLD	WB/WT#	Ignored
HOLD	Recognized	WM_RST	Latched/Recognized
IGNNE#	Ignored		

3.2.16 JTAG Interface

The IBM 6x86MX CPU can be tested using JTAG Interface (IEEE Std. 1149.1) boundary scan test logic. The IBM 6x86MX CPU pin state can be set according to serial data supplied to the chip. The IBM 6x86MX CPU pin state can also be recorded and supplied as serial data.

Test Clock (TCK) is the clock input used by the IBM 6x86MX CPU boundary scan (JTAG) test logic. The rising edge of TCK is used to clock control and data information into the IBM 6x86MX CPU using the TMS and TDI pins. The falling edge of TCK is used to clock data information out of the IBM 6x86MX CPU using the TDO pin.

Test Data Input (TDI) is the serial data input used by the IBM 6x86MX CPU boundary scan (JTAG) test logic. TDI is sampled on the rising edge of TCK.

Test Data Output (TDO) is the serial data output used by the IBM 6x86MX CPU boundary scan (JTAG) test logic. TDO is output on the falling edge of TCK.

Test Mode Select (TMS) is the control input used by the IBM 6x86MX CPU boundary scan (JTAG) test logic. TMS is sampled on the rising edge of TCK.

Test Reset (TRST#) is an active low input used to initialize the IBM 6x86MX CPU boundary scan (JTAG) test logic.

3.3 Functional Timing

3.3.1 Reset Timing

Figure 3-2 illustrates the required RESET timing for both a power-on reset and a reset that occurs during operation. The WM_RST and FLUSH# inputs are sampled at the falling edge of RESET to determine if the IBM 6x86MX CPU should enter built-in self-test, enable tree-state test mode or enable the scatter-gather interface pins, respectively. WM_RST and FLUSH# must be valid at least two clocks prior to the RESET falling edge.

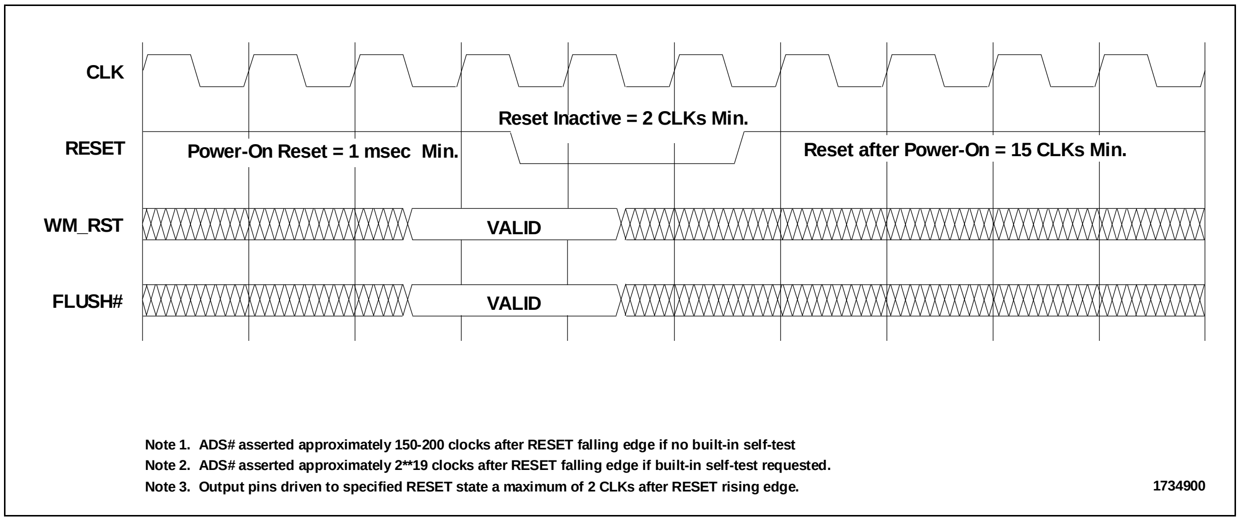

CLK
RESET
WM_RST
FLUSH#
Reset Inactive = 2 CLKs Min.
Power-On Reset = 1 msec Min.
Reset after Power-On = 15 CLKs Min.
VALID
VALID
Note 1. ADS# asserted approximately 150-200 clocks after RESET falling edge if no built-in self-test
Note 2. ADS# asserted approximately 2**19 clocks after RESET falling edge if built-in self-test requested.
Note 3. Output pins driven to specified RESET state a maximum of 2 CLKs after RESET rising edge.
1734900

Figure 3-2. RESET Timing

3.3.2 Bus State Definition

The IBM 6x86MX CPU bus controller supports non-pipelined and pipelined operation as well as single transfer and burst bus cycles. During each CLK period, the bus controller exists in one of six states as listed in Table 3-11. Each of bus state and its associated state transitions are illustrated in Figure 3-3, (Page 3-25) and listed in Table 3-12, (Page 3-26).

Table 3-11. IBM 6x86MX CPU Bus States

STATE	NAME	DESCRIPTION
Ti	Idle Clock	During Ti, no bus cycles are in progress. BOFF# and RESET force the bus to the idle state. The bus is always in the idle state while HLDA is active.
T1	First Bus Cycle Clock	During the first clock of a non-pipelined bus cycle, the bus enters the T1 state. ADS# is asserted during T1 along with valid address and bus cycle definition information.
T2	Second and Subsequent Bus Cycle Clock	During the second clock of a non-pipelined bus cycle, the bus enters the T2 state. The bus remains in the T2 state for subsequent clocks of the bus cycle as long as a pipelined cycle is not initiated. During T2, valid data is driven during write cycles and data is sampled during reads. BRDY# is also sampled during T2. The bus also enters the T2 state to complete bus cycles that were initiated as pipelined cycles but complete as the only outstanding bus cycle.
T12	First Pipelined Bus Cycle Clock	During the first clock of a pipelined cycle, the bus enters the T12 state. During T12, data is being transferred and BRDY# is sampled for the current cycle at the same time that ADS# is asserted and address/bus cycle definition information is driven for the next (pipelined) cycle.
T2P	Second and Subsequent Pipelined Bus Cycle Clock	During the second and subsequent clocks of a pipelined bus cycle where two cycles are outstanding, the bus enters the T2P state. During T2P, data is being transferred and BRDY# is sampled for the current cycle. However, valid address and bus cycle definition information continues to be driven for the next pipelined cycle.
Td	Dead Clock	The bus enters the Td state if a pipelined cycle was initiated that requires one idle clock to turn around the direction of the data bus. Td is required for a read followed immediately by a pipelined write, and for a write followed immediately by a pipelined read.

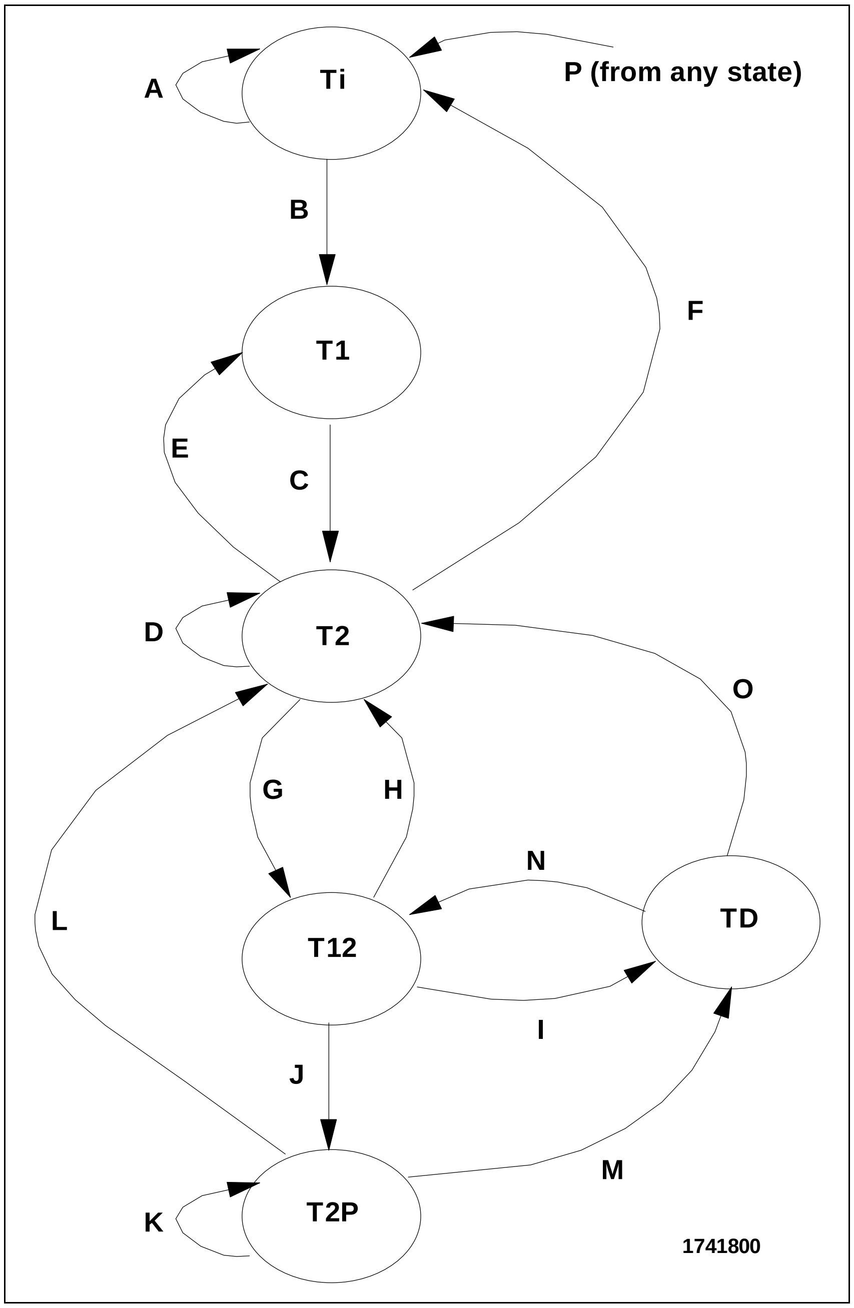


Figure 3-3. IBM 6x86MX CPU Bus State Diagram

Table 3-12. Bus State Transitions

TRANSITION	CURRENT STATE	NEXT STATE	EQUATION
A	Ti	Ti	No Bus Cycle Pending.
B	Ti	T1	New or Aborted Bus Cycle Pending.
C	T1	T2	Always.
D	T2	T2	Not Last BRDY# and No New Bus Cycle Pending, or Not Last BRDY# and New Bus Cycle Pending and NA# Negated.
E	T2	T1	Last BRDY# and New Bus Cycle Pending and HITM# Negated.
F	T2	Ti	Last BRDY# and No New Bus Cycle Pending, or Last BRDY# and HITM# Asserted.
G	T2	T12	Not Last BRDY# and New Bus Cycle Pending and NA# Sampled Asserted.
H	T12	T2	Last BRDY# and No Dead Clock Required.
I	T12	Td	Last BRDY# and Dead Clock Required.
J	T12	T2P	Not Last BRDY#.
K	T2P	T2P	Not Last BRDY#.
L	T2P	T2	Last BRDY# and No Dead Clock Required.
M	T2P	Td	Last BRDY# and Dead Clock Required.
N	Td	T12	New Bus Cycle Pending and NA# Sampled Asserted.
O	Td	T2	No New Bus Cycle Pending, or New Bus Cycle Pending and NA# Negated.
P	Any State	Ti	RESET Asserted, or BOFF# Asserted.

3.3.3 Non-Pipelined Bus Cycles

Non-pipelined bus operation may be used for all bus cycle types. The term “non-pipelined” refers to a mode of operation where the CPU allows only one outstanding bus cycle. In other words, the current bus cycle must complete before a second bus cycle is allowed to start.

3.3.3.1 Non-Pipelined Single Transfer Cycles

Single transfer read cycles occur during non-cacheable memory reads, I/O read cycles, and special cycles. A non-pipelined single transfer read cycle begins with address and bus cycle definition information driven on the bus during the first clock (T1 state) of the bus cycle. The CPU then monitors the BRDY# input at the end of the second clock (T2 state). If BRDY# is asserted, the CPU reads the appropriate data and data parity lines and terminates the bus cycle. If BRDY# is not active, the CPU continues to sample the BRDY# input at the end of each subsequent cycle (T2 states). Each of the additional clocks is referred to as a wait state.

The CPU uses the data parity inputs to check for even parity on the active data lines. If the CPU detects an error, the parity check output (PCHK#) asserts during the second clock following the termination of the read cycle.

Figure 3-4 (Page 3-28) illustrates the functional timing for two non-pipelined single-transfer read cycles. Cycle 2 is a potentially cacheable cycle as indicated by the CACHE# output. Because this cycle is potentially cacheable, the CPU samples the KEN# input at the same clock edge that BRDY# is asserted. If KEN# is negated, the cycle terminates as shown in the diagram. If KEN# is asserted, the CPU converts this cycle into a burst cycle as described in the next section. NA# must be negated for non-pipelined operation. Pipelined bus cycles are described later in this chapter.

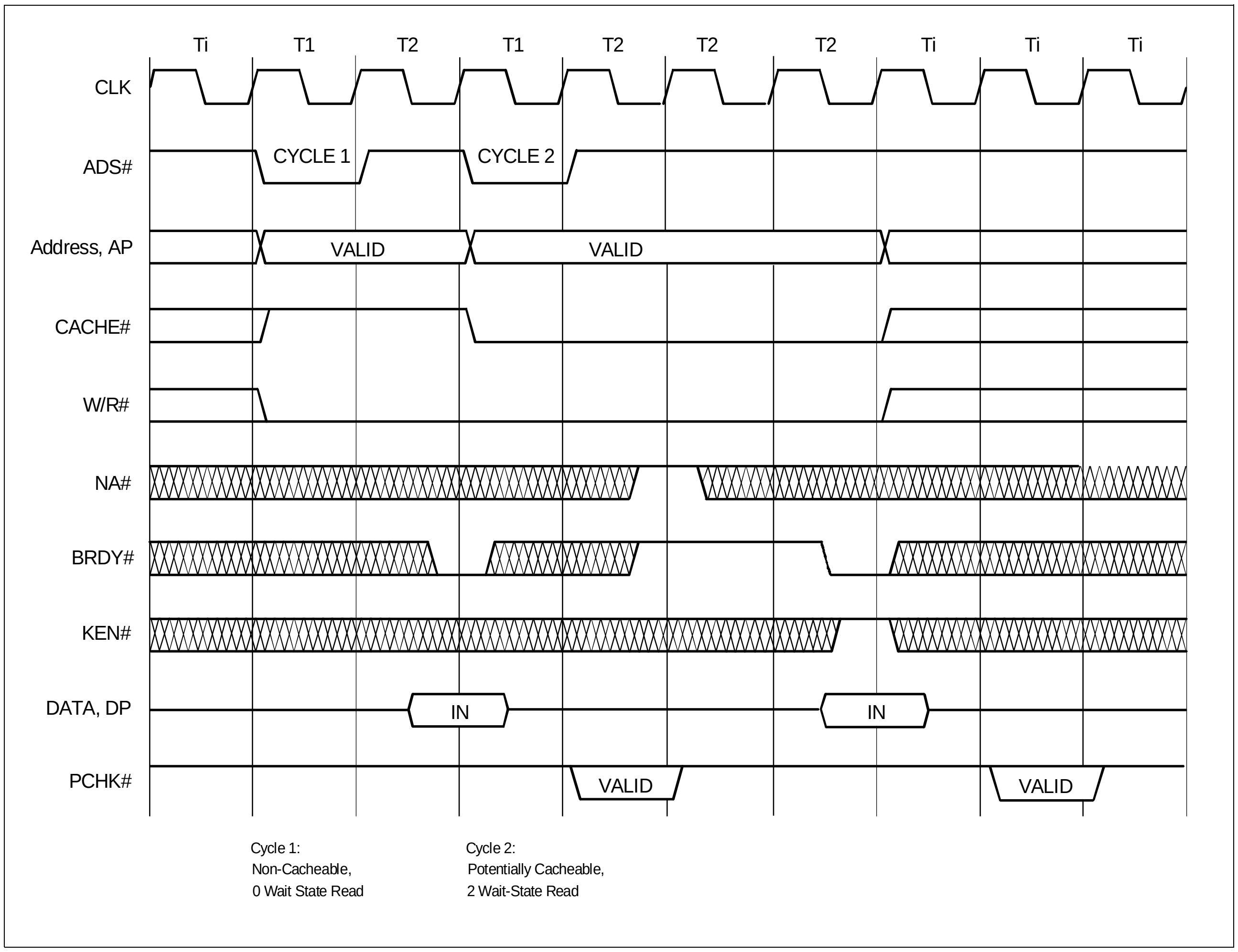


Figure 3-4. Non-Pipelined Single Transfer Read Cycles

Single transfer write cycles occur for writes that are neither line replacement nor write-back cycles. The functional timing of two non-pipelined single transfer write cycles is shown in Figure 3-5. During a write cycle, the data and data parity lines are outputs and are driven valid during the second clock (T2 state) of the bus cycle. Data and data parity remain valid during all wait states. If the write cycle is a write to a valid cache location in the “shared” state, the WB/WT# pin is sampled with BRDY#. If WB/WT# is sampled high, the cache line transitions from the “shared” to the “exclusive” state.

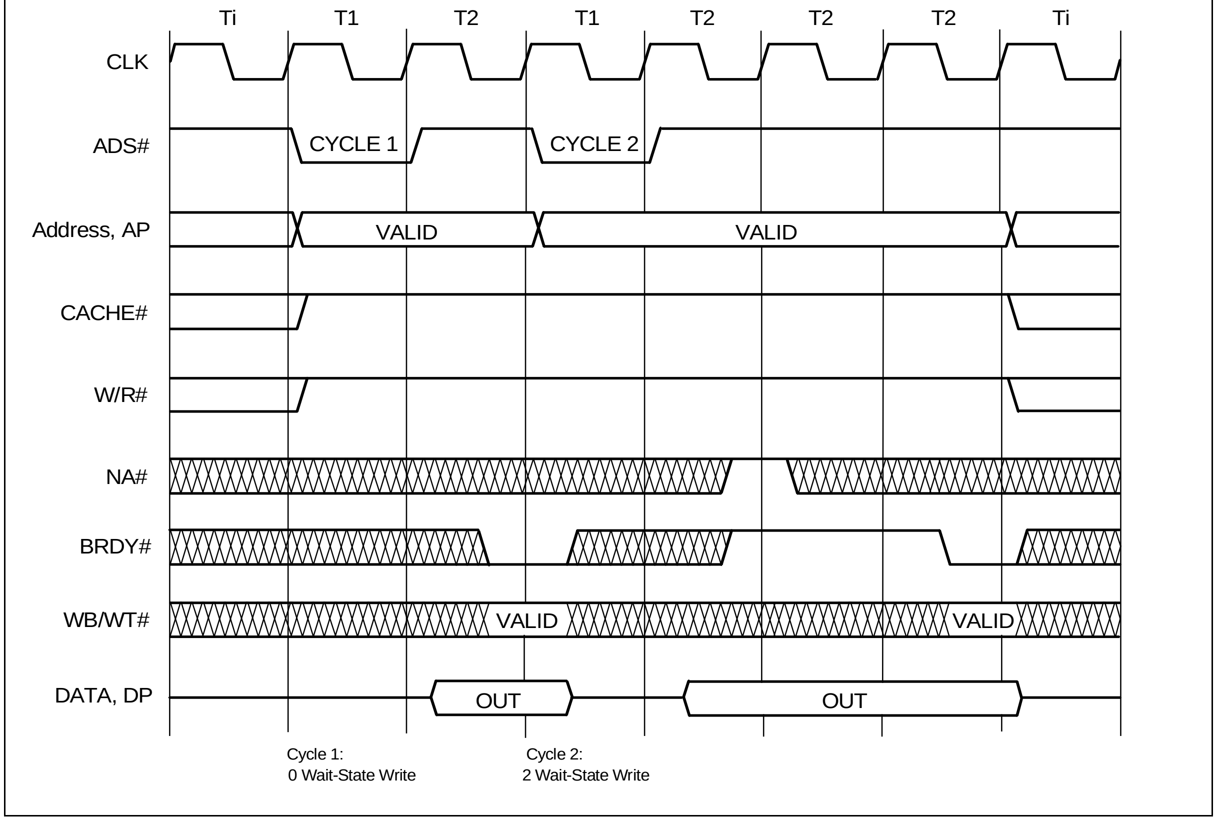


Figure 3-5. Non-Pipelined Single Transfer Write Cycles

3.3.3.2 Non-pipelined Burst Read Cycles

The IBM 6x86MX CPU uses burst read cycles to perform cache line fills. During a burst read cycle, four 64-bit data transfers occur to fill one of the CPU's 32-byte internal cache lines. A non-pipelined burst read cycle begins with address and bus cycle definition information driven on the bus during the first clock (T1 state) of the bus cycle. The CACHE# output is always active during a burst read cycle and is driven during the T1 clock.

The CPU then monitors the BRDY# input at the end of the second clock (T2 state). If BRDY# is asserted, the CPU reads the data and data parity and also checks the KEN# input. If KEN# is negated, the CPU terminates the bus cycle as a single transfer cycle. If KEN# is asserted, the CPU converts the cycle into a burst (cache line fill) by continuing to sample BRDY# at the end of each subsequent clock. BRDY# must be asserted a total of four times to complete the burst cycle.

WB/WT# is sampled at the same clock edge as KEN#. In conjunction with PWT and the on-chip configuration registers, WB/WT# determines the MESI state of the cache line for the current line fill.

Each time BRDY# is sampled asserted during the burst cycle, a data transfer occurs. The CPU reads the data and data parity busses and assigns the data to an internally generated burst address. Although the CPU internally generates the burst address sequence, only the first address of the burst is driven on the external address bus. System logic must predict the burst address sequence based on the first address. Wait states may be added to any transfer within a burst by delaying the assertion of BRDY# by the desired number of clocks.

The CPU checks even data parity for each of the four transfers within the burst. If the CPU detects an error, the parity check output (PCHK#) asserts during the second clock following the BRDY# assertion of the data transfer.

Figure 3-6 (Page 3-31) illustrates two non-pipelined burst read cycles. The cycles shown are the fastest possible burst sequences (2-1-1-1). NA# must be negated for non-pipelined operation as shown in the diagram. Pipelined bus cycles are described later in this chapter.

Figure 3-7 (Page 3-32) depicts a burst read cycle with wait states. A 3-2-2-2 burst read is shown.

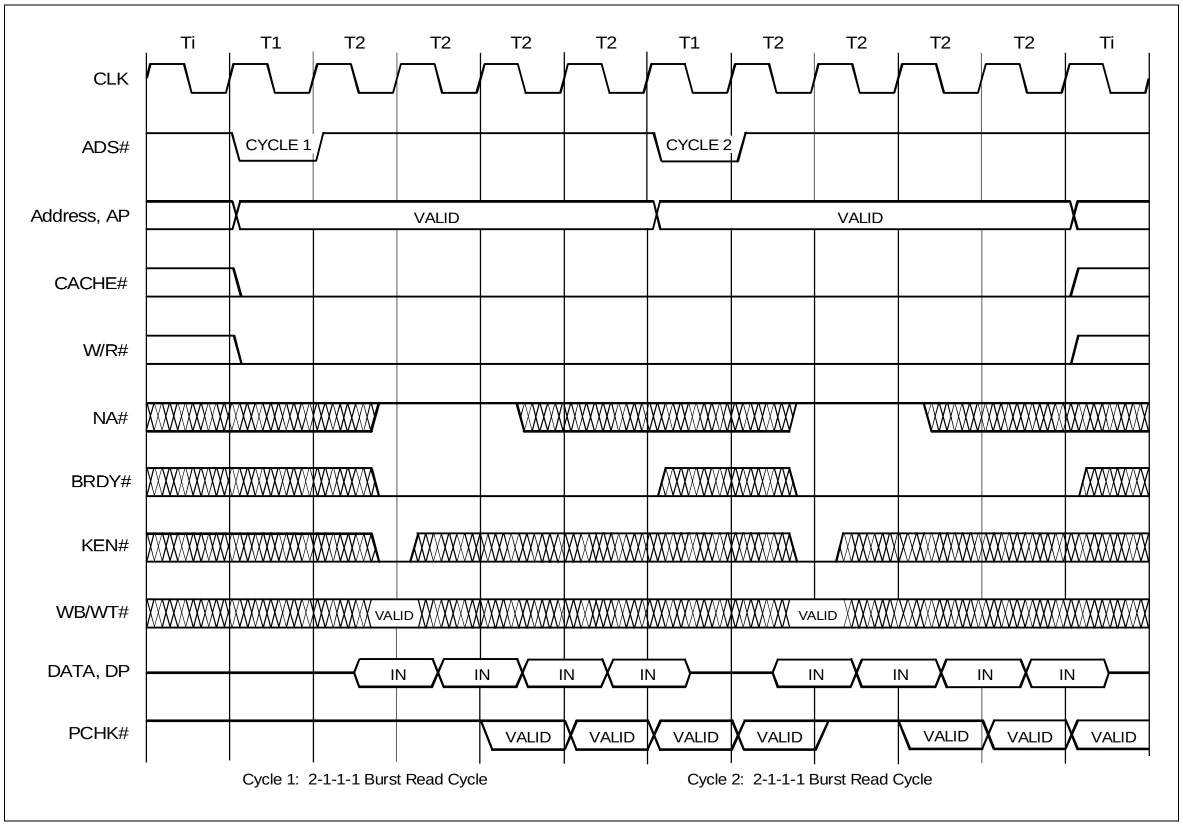


Figure 3-6. Non-Pipelined Burst Read Cycles

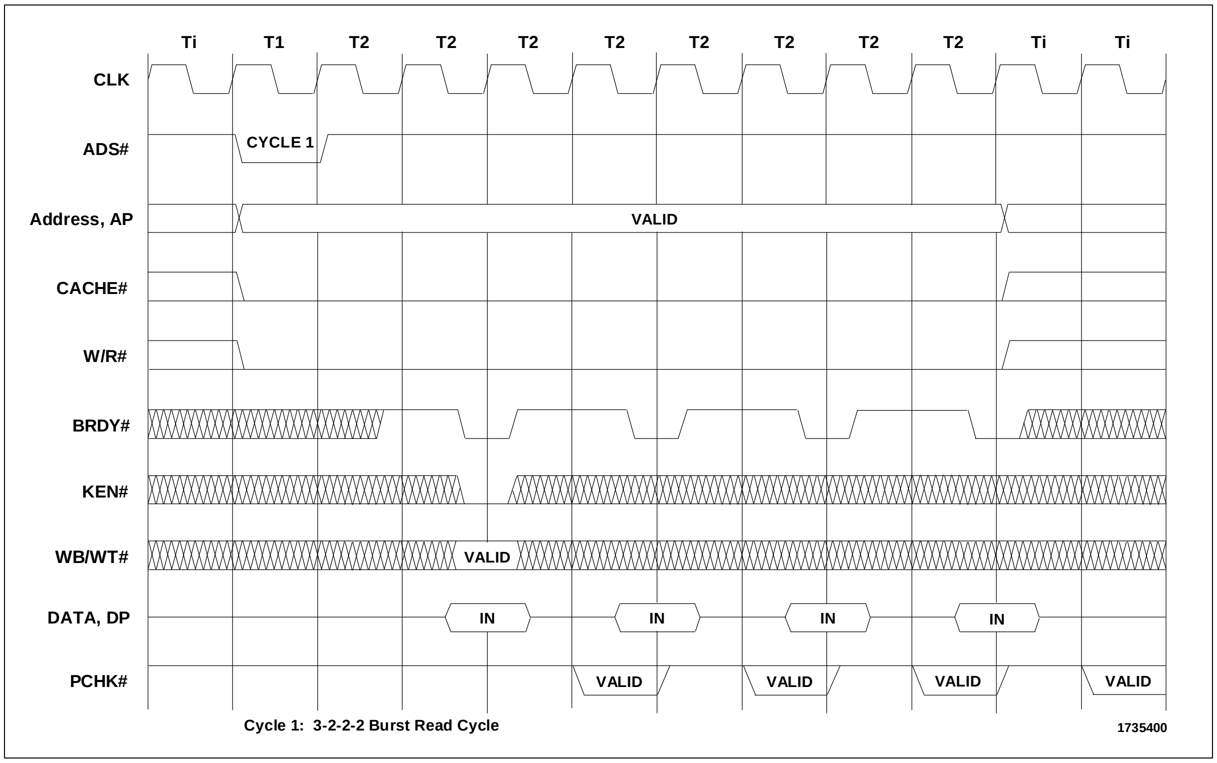


Figure 3-7. Burst Cycle with Wait States

Burst Cycle Address Sequence.

The IBM 6x86MX CPU provides two different address sequences for burst read cycles. The IBM 6x86MX CPU burst cycle address sequence modes are referred to as “1+4” and “linear”. After reset, the CPU default mode is “1+4”.

In “1+4” mode, the CPU performs a single transfer read cycle prior to the burst cycle, if the desired first address is (...xx8). During this single transfer read cycle, the CPU reads the critical data. In addition, the IBM 6x86MX CPU samples the state of KEN#. If KEN# is active, the CPU then performs the burst cycle with the address sequence shown in Table 3-13 (Page 3-33). The IBM 6x86MX CPU CACHE# output is not asserted during the single read cycle prior to the burst. Therefore, CACHE# must not be used to qualify the KEN# input to the processor. In addition, if KEN# is returned active for the “1” read cycle in the “1+4”, all data bytes supplied to the CPU must be valid. The CPU samples WB/WT# during the “1” read cycle, and does not resample WB/WT# during the following burst cycle. Figure 3-8 (Page 3-33) illustrates a “1+4” burst read cycle.

Table 3-13. “1+4” Burst Address Sequences

BURST CYCLE FIRST ADDRESS	SINGLE READ CYCLE PRIOR TO BURST	BURST CYCLE ADDRESS SEQUENCE
0	None	0-8-10-18
8	Address 8	0-8-10-18
10	None	10-18-0-8
18	Address 18	10-18-0-8

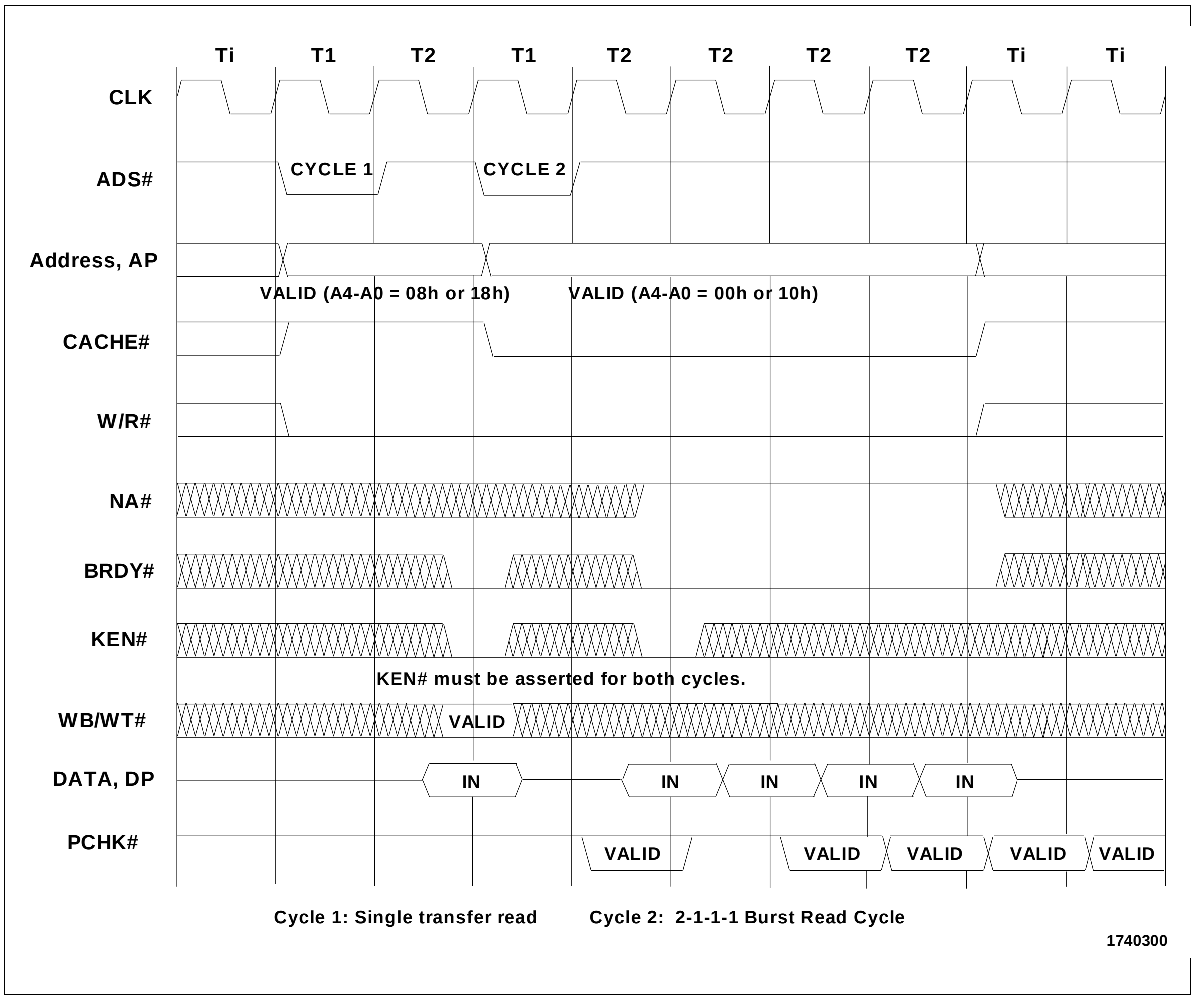

Ti
T1
T2
T1
T2
T2
T2
T2
Ti
Ti
CLK
ADS#
CYCLE 1
CYCLE 2
Address, AP
VALID (A4-A0 = 08h or 18h)
VALID (A4-A0 = 00h or 10h)
CACHE#
W/R#
NA#
BRDY#
KEN#
KEN# must be asserted for both cycles.
WB/WT#
VALID
DATA, DP
IN
IN
IN
IN
IN
PCHK#
VALID
VALID
VALID
VALID
VALID
Cycle 1: Single transfer read
Cycle 2: 2-1-1-1 Burst Read Cycle
1740300

Figure 3-8. “1+4” Burst Read Cycle

The address sequences for the IBM 6x86MX CPU's linear burst mode are shown in Table 3-14. Operating the CPU in linear burst mode minimizes processor bus activity resulting in higher system performance. Linear burst mode can be enabled through the IBM 6x86MX CPU CCR3 configuration register.

Table 3-14. Linear Burst Address Sequences

BURST CYCLE FIRST ADDRESS	BURST CYCLE ADDRESS SEQUENCE
0	0-8-10-18
8	8-10-18-0
10	10-18-0-8
18	18-0-8-10

3.3.3.3 Burst Write Cycles

Burst write cycles occur for line replacement and write-back cycles. Burst writes are similar to burst read cycles in that the CACHE# output is asserted and four 64-bit data transfers occur. Burst writes differ from burst reads in that the data and data parity lines are outputs rather than inputs. Also, KEN# and WB/WT# are not sampled during burst write cycles.

Data and data parity for the first data transfer are driven valid during the second clock (T2 state) of the bus cycle. Once BRDY# is sampled asserted for the first data transfer, valid data and data parity for the second transfer are driven during the next clock cycle. The same timing relationship between BRDY# and data applies for the third and fourth data transfers as well. Wait states may be added to any transfer within a burst by delaying the assertion of BRDY# by the required number of clocks.

As on burst read cycles, only the first address of a burst write cycle is driven on the external address bus. System logic must predict the remaining burst address sequence based on the first address. Burst write cycles always begin with a first address ending in 0 (signals A4-A0=0) and follow an ascending address sequence for the remaining transfers (0-8-10-18).

Figure 3-9 illustrates two non-pipelined burst write cycles. The cycles shown are the fastest possible burst sequences (2-1-1-1). As shown, an idle clock always exists between two back-to-back burst write cycles. Therefore, the second burst write cycle in a pair of back-to-back burst writes is always issued as a non-pipelined cycle regardless of the state of the NA# input.

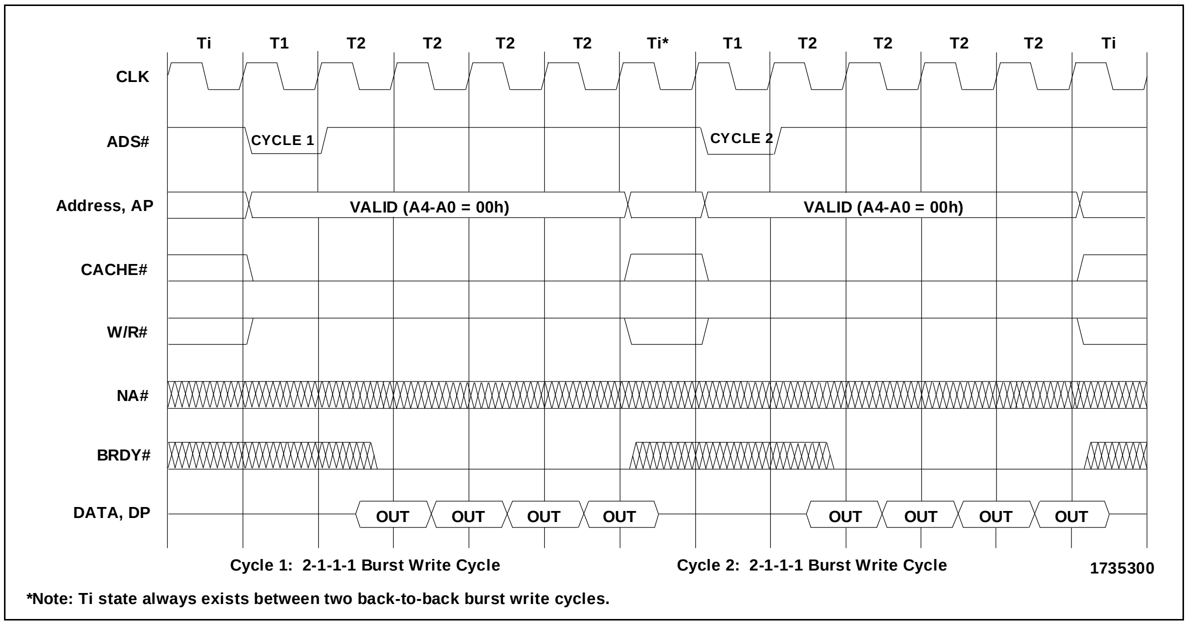


Figure 3-9. Non-Pipelined Burst Write Cycles

3.3.4 Pipelined Bus Cycles

Pipelined addressing is a mode of operation where the CPU allows up to two outstanding bus cycles at any given time. Using pipelined addressing, the address of the first bus cycle is driven on the bus. While the CPU waits for the data for the first cycle, the address for a second bus cycle is issued. Pipelined bus cycles occur for all cycle types except locked cycles and burst write cycles.

Pipelined cycles are initiated by asserting NA#. The CPU samples NA# at the end of each T2, T2P and Td state. KEN# and WB/WT# are sampled at either the same clock as NA# is active, or at the same clock as the first BRDY# for that cycle, whichever occurs first. The CPU issues the next address a minimum of two clocks after NA# is sampled asserted.

The CPU latches the state of the NA# pin internally. Therefore, even if a new bus cycle is not pending internally at the time NA# was sampled asserted, the CPU still issues a pipelined bus cycle if an internal bus request occurs prior to completion of the current bus cycle. Once NA# is sampled asserted, the state of NA# is ignored until the current bus cycle completes. If two cycles are outstanding and the second cycle is a read, the CPU samples KEN# and WB/WT# for the second cycle when NA# is sampled asserted.

Figure 3-10 and Figure 3-11 (Page 3-37) illustrate pipelined single transfer read cycles and pipelined burst read cycles, respectively.

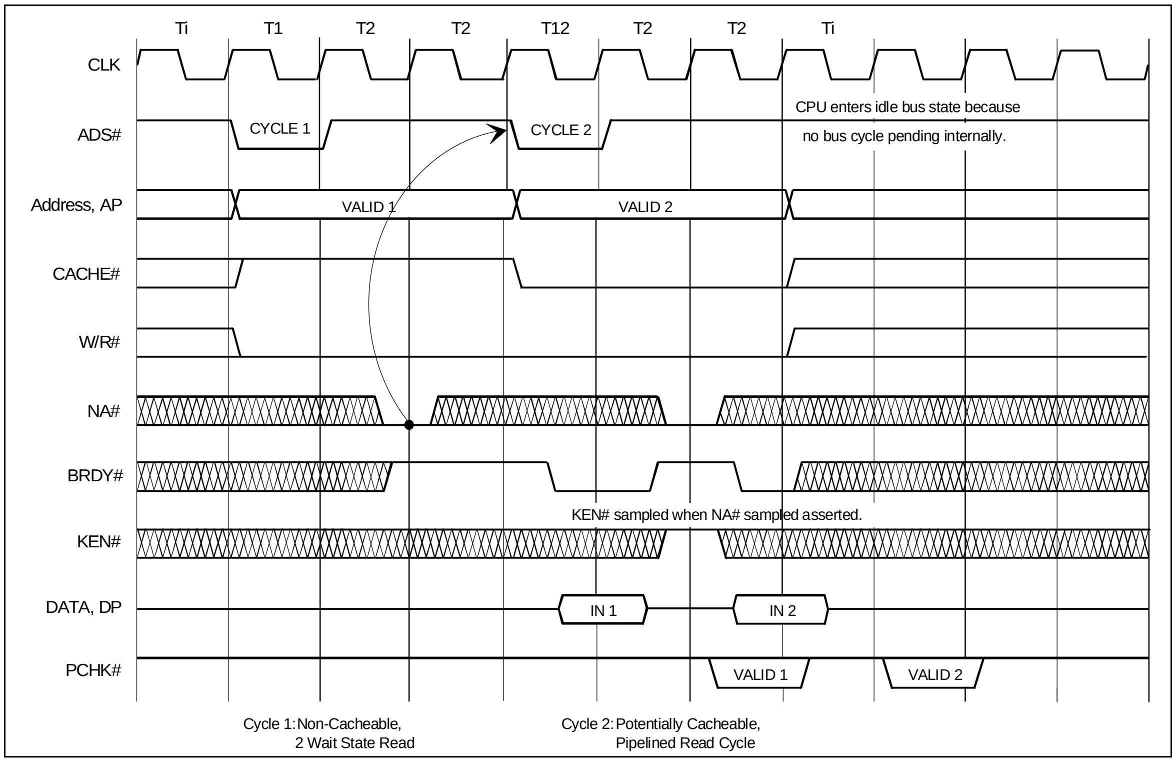

Ti
T1
T2
T2
T12
T2
T2
Ti
CLK
ADS#
CYCLE 1
CYCLE 2
CPU enters idle bus state because
no bus cycle pending internally.
Address, AP
VALID 1
VALID 2
CACHE#
W/R#
NA#
BRDY#
KEN# sampled when NA# sampled asserted.
KEN#
DATA, DP
IN 1
IN 2
PCHK#
VALID 1
VALID 2
Cycle 1: Non-Cacheable, 2 Wait State Read
Cycle 2: Potentially Cacheable, Pipelined Read Cycle

Figure 3-10. Pipelined Single Transfer Read Cycles

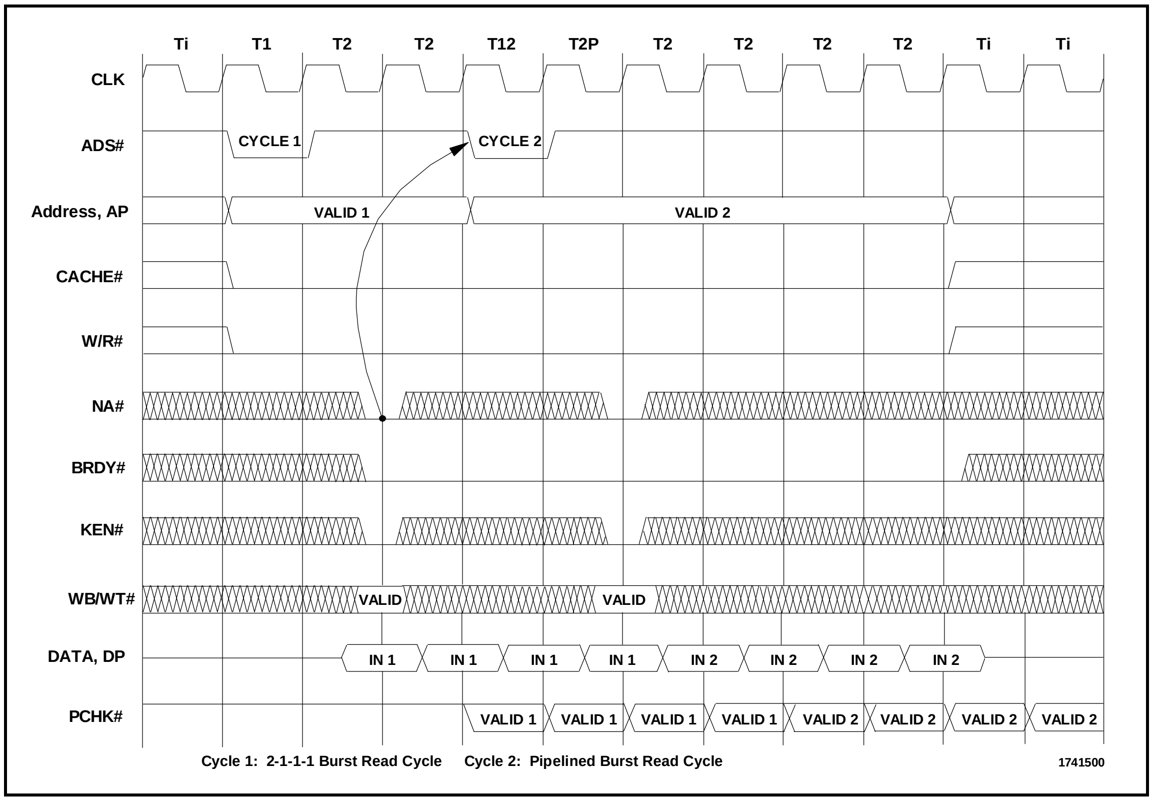


Figure 3-11. Pipelined Burst Read Cycles

3.3.4.1 Pipelined Back-to-Back Read/Write Cycles

Figure 3-12 depicts a read cycle followed by a pipelined write cycle. Under this condition, the data bus must change from an input for the read cycle to an output for the write cycle. In order to accomplish this transition without causing data bus contention, the CPU automatically inserts a "dead" (Td) clock cycle. During the Td state, the data bus floats. The CPU then drives the write data onto the bus in the following clock. The CPU also inserts a Td clock between a write cycle and a pipelined read cycle to allow the data bus to smoothly transition from an output to an input.

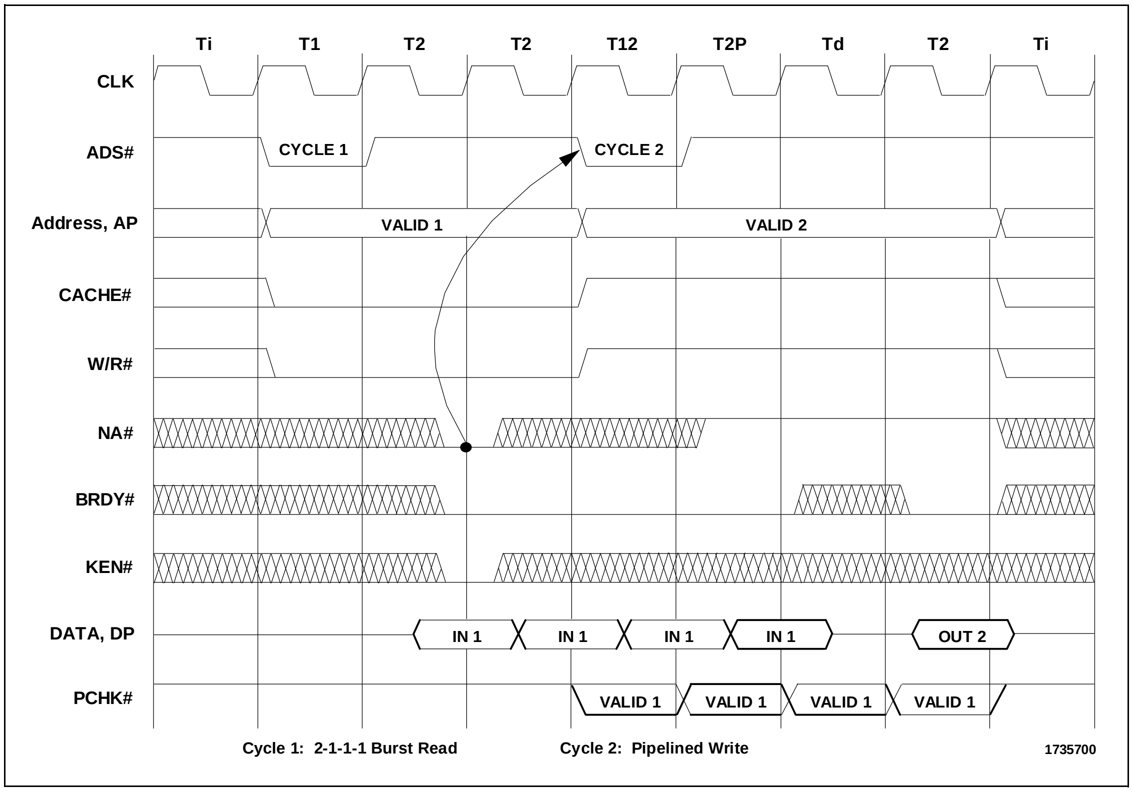


Figure 3-12. Read Cycle Followed by Pipelined Write Cycle

3.3.5 Interrupt Acknowledge Cycles

The CPU issues interrupt acknowledge bus cycles in response to an active INTR input. Interrupt acknowledge cycles are single transfer cycles and always occur in locked pairs as shown in Figure 3-13. The CPU reads the interrupt vector from the lower eight bits of the data bus at the completion of the second interrupt acknowledge cycle. Parity is not checked during the first interrupt acknowledge cycle.

M/IO#, D/C# and W/R# are always logic low during interrupt acknowledge cycles. Additionally, the address bus is driven with a value of 0000 0004h for the first interrupt acknowledge cycle and with a value of 0000 0000h for the second. A minimum of one idle clock always occurs between the two interrupt acknowledge cycles.

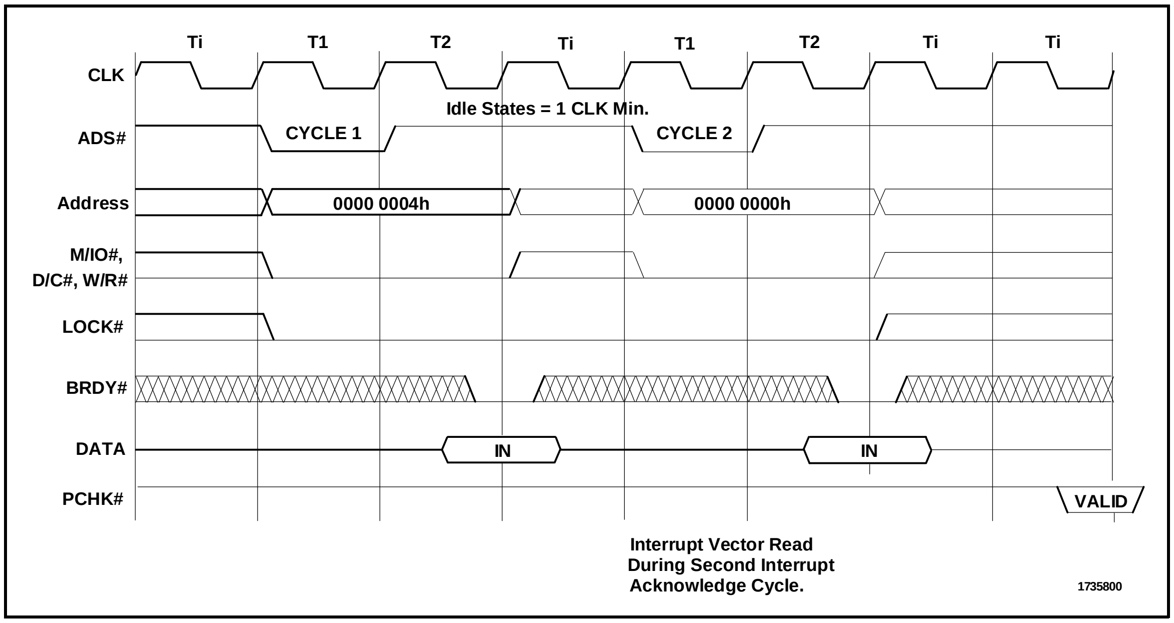


Figure 3-13. Interrupt Acknowledge Cycles

3.3.6 SMI# Interrupt Timing

The CPU samples the System Management Interrupt (SMI#) input at each clock edge. At the next appropriate instruction boundary, the CPU recognizes the SMI# and completes all pending write cycles. The CPU then asserts SMIACT# and begins saving the SMM header information to the SMM address space. SMIACT# remains asserted until after execution of a RSM instruction. Figure 3-14 illustrates the functional timing of the SMIACT# signal.

To facilitate using SMI# to power manage I/O peripherals, the IBM 6x86MX CPU implements a feature called I/O trapping. If the current bus cycle is an I/O cycle and SMI# is asserted a minimum of three clocks prior to BRDY#, the CPU immediately begins execution of the SMI service routine following completion of the I/O instruction. No additional instructions are executed prior to entering the SMI service routine. I/O trap timing requirements are shown in Figure 3-15 (Page 3-41).

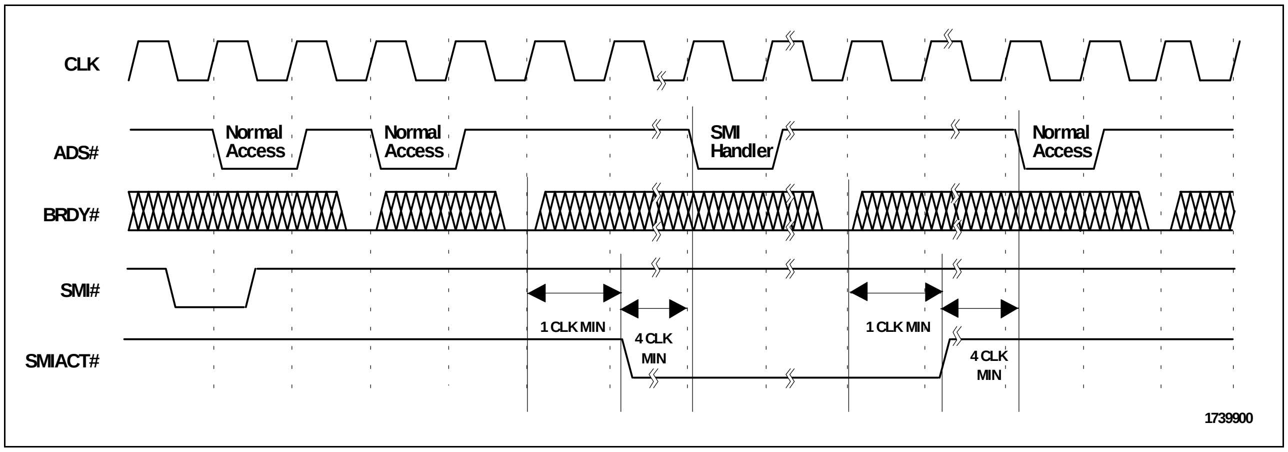


Figure 3-14. SMIACT# Timing in SL Compatible Mode

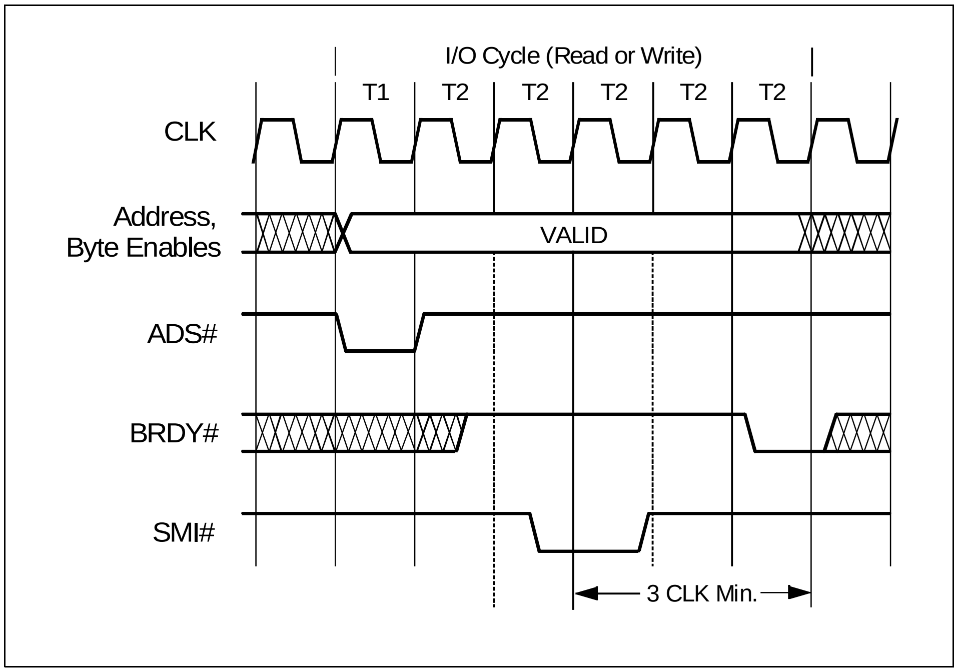


Figure 3-15. SMM I/O Trap Timing

3.3.7 Cache Control Timing

3.3.7.1 Invalidating the Cache Using FLUSH#

The FLUSH# input forces the CPU to write-back and invalidate the entire contents of the on-chip cache. FLUSH# is sampled at each clock edge, latched internally and then recognized internally at the next instruction boundary. Once FLUSH# is recognized, the CPU issues a series of burst write cycles to write-back any “modified” cache lines. The cache lines are invalidated as they are written back. Following completion of the write-back cycles, the CPU issues a flush acknowledge special bus cycle.

The latency between when FLUSH# occurs and when the cache invalidation actually completes varies depending on:

(1) the state of the processor when FLUSH# is asserted,
(2) the number of modified cache lines,
(3) the number of wait states inserted during the write-back cycles.

Figure 3-16 (Page 3-42) illustrates the sequence of events that occur on the bus in response to a FLUSH# request.

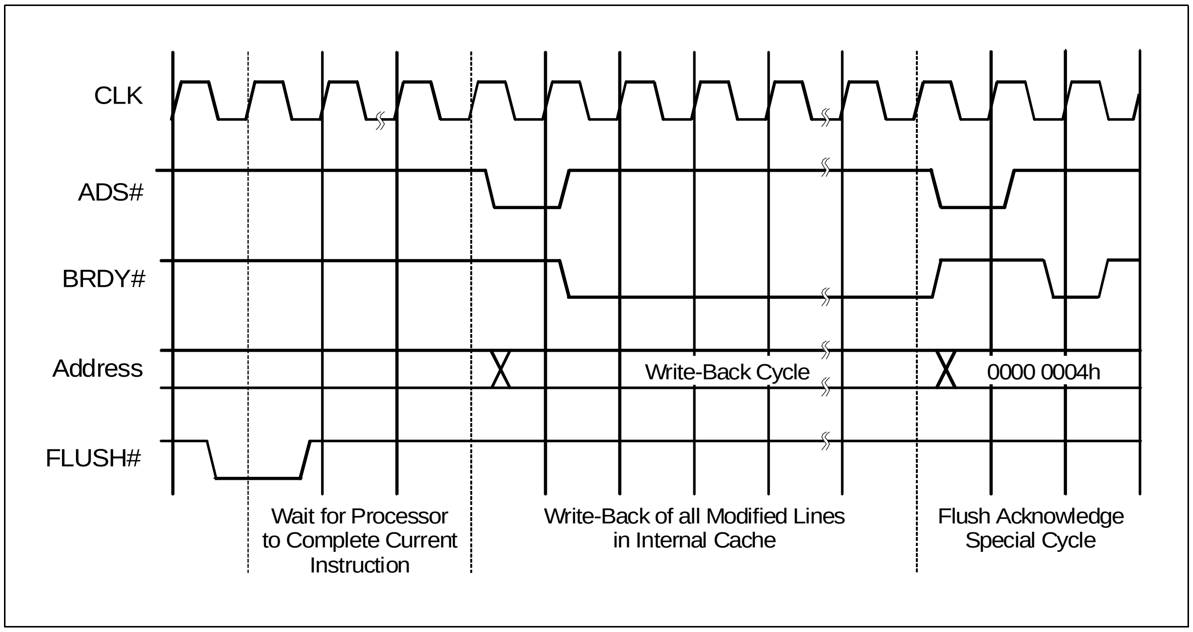

CLK
ADS#
BRDY#
Address
FLUSH#
Write-Back Cycle
0000 0004h
Wait for Processor to Complete Current Instruction
Write-Back of all Modified Lines in Internal Cache
Flush Acknowledge Special Cycle

Figure 3-16. Cache Invalidation Using FLUSH#

3.3.7.2 EWBE# Timing

During memory and I/O write cycles, the IBM 6x86MX CPU samples the external write buffer empty (EWBE#) input. If EWBE# is negated, the CPU does not write any data to “exclusive” or “modified” internal cache lines. After sampling EWBE# negated, the CPU continues to sample EWBE# at each clock edge until it asserts. Once EWBE# is asserted, all internal cache writes are allowed. Through use of this signal, the external system may enforce strong write ordering when external write buffers are used. EWBE# functional timing is shown in Figure 3-17.

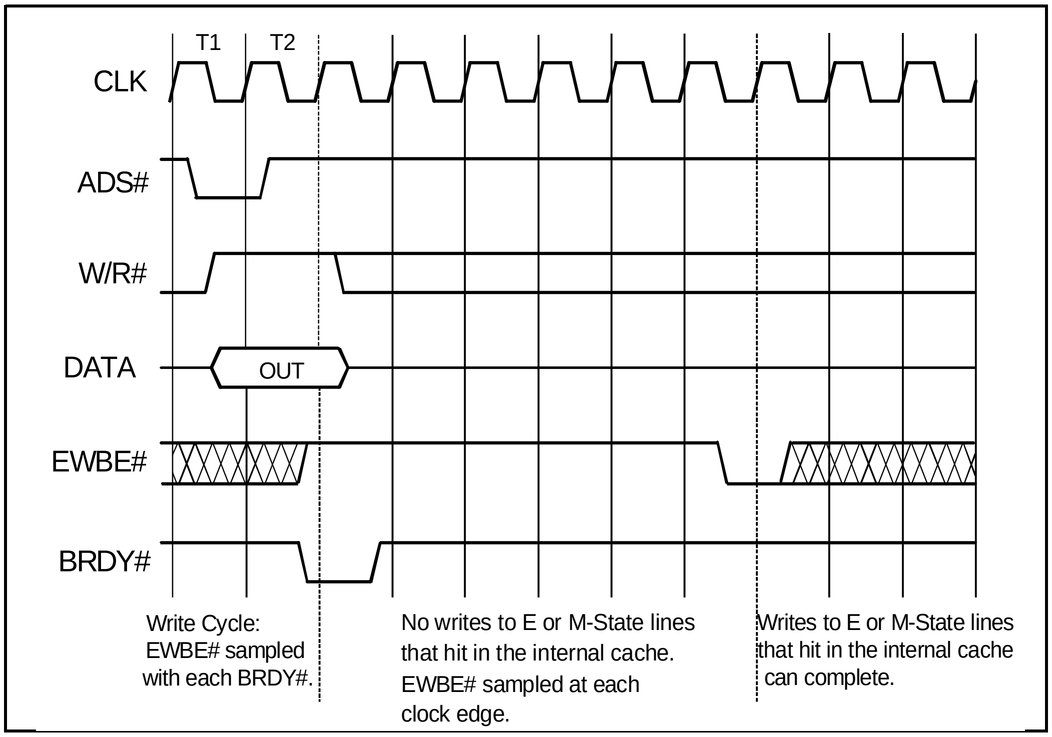

T1
T2
CLK
ADS#
W/R#
DATA
OUT
EWBE#
BRDY#
Write Cycle: EWBE# sampled with each BRDY#.
No writes to E or M-State lines that hit in the internal cache. EWBE# sampled at each clock edge.
Writes to E or M-State lines that hit in the internal cache can complete.

Figure 3-17. External Write Buffer Empty (EWBE#) Timing

3.3.8 Bus Arbitration

An external bus master can take control of the CPU's bus using either the HOLD/HLDA handshake signals or the back-off (BOFF#) input. Both mechanisms force the IBM 6x86MX CPU to enter the bus hold state.

3.3.8.1 HOLD and HLDA

Using the HOLD/HLDA handshake, an external bus master requests control of the CPU's bus by asserting the HOLD signal. In response to an active HOLD signal, the CPU completes all outstanding bus cycles, enters the bus hold state by floating the bus, and asserts the HLDA output. The CPU remains in the bus hold state until HOLD is negated. Figures 3-18 (this page), Figure 3-19 (Page 3-45) and Figure 3-20 (Page 3-46) illustrate the timing associated with requesting HOLD during an idle bus, during a non-pipelined bus cycle and during a pipelined bus cycle, respectively.

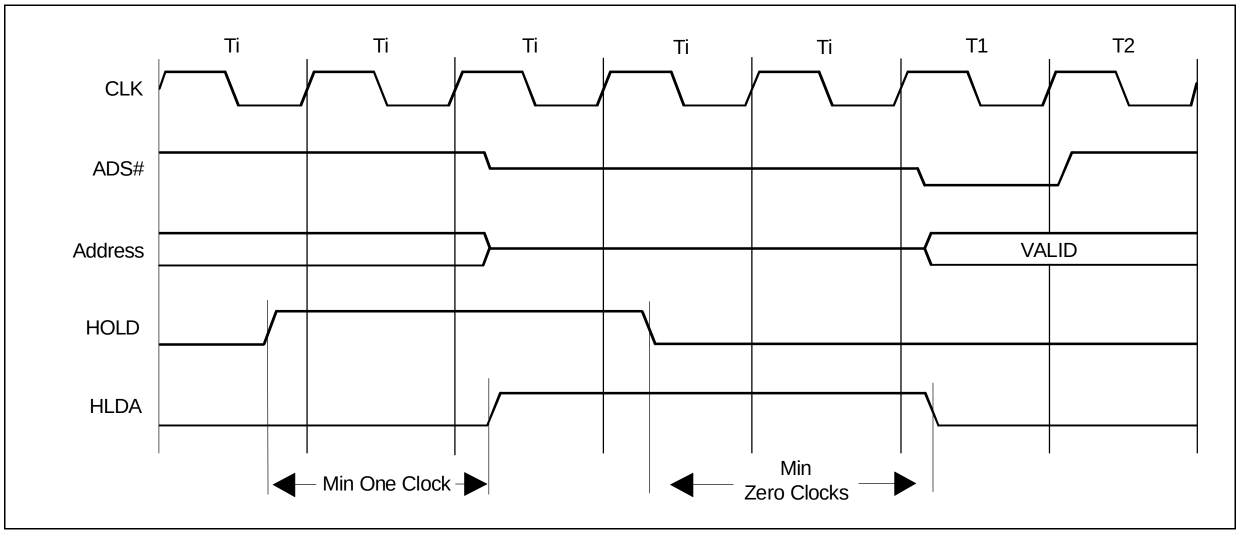


Figure 3-18. Requesting Hold from an Idle Bus

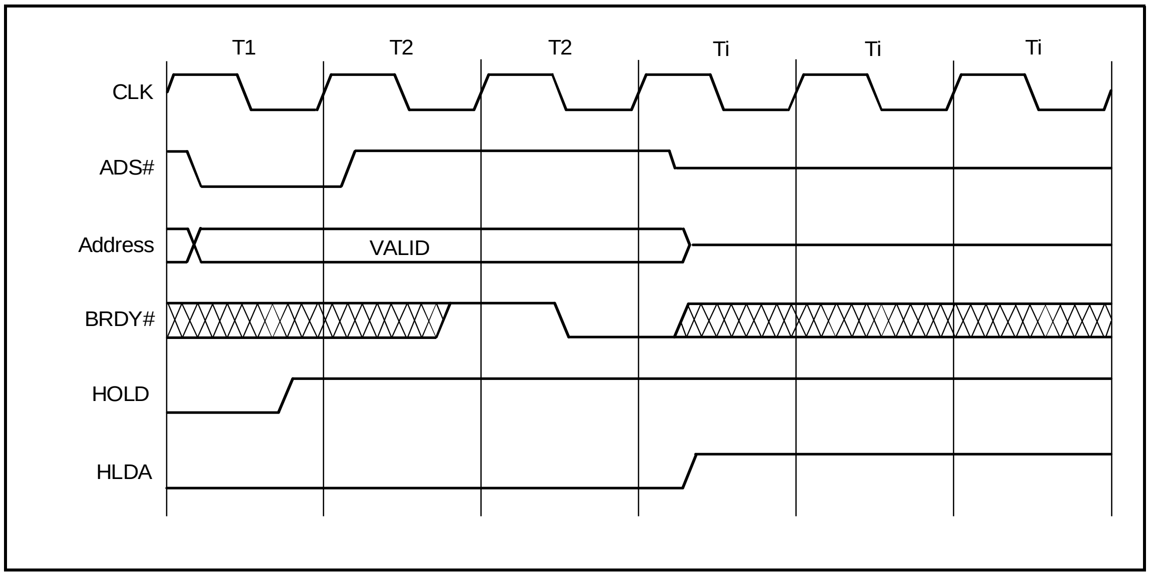


Figure 3-19. Requesting Hold During a Non-Pipelined Bus Cycle

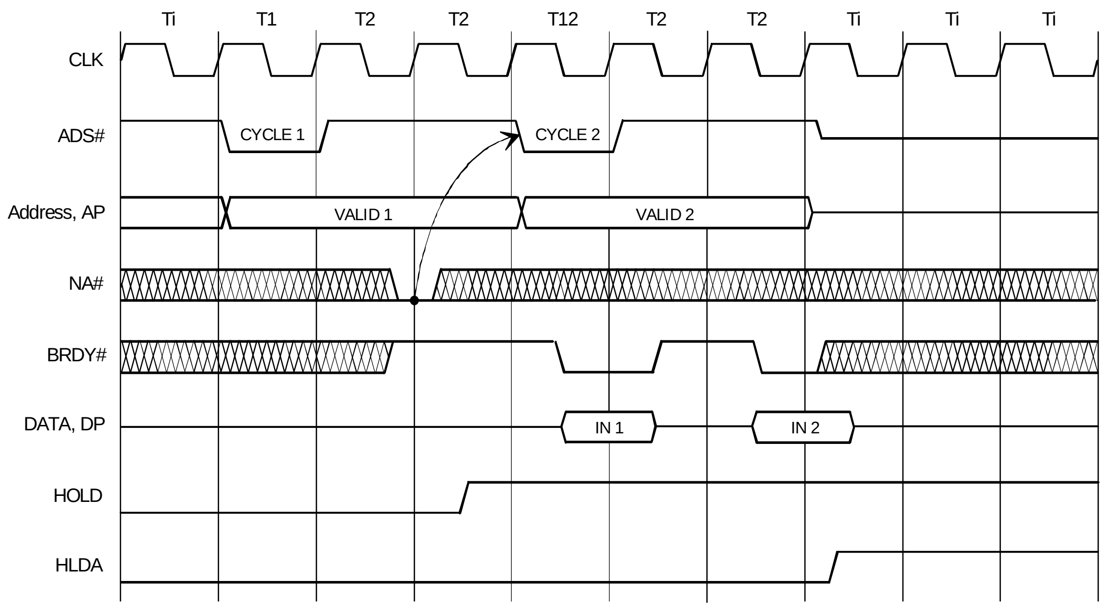


Figure 3-20. Requesting Hold During a Pipelined Bus Cycle

3.3.8.2 Back-Off Timing

An external bus master requests immediate control of the CPU's bus by asserting the back-off (BOFF#) input. The CPU samples BOFF# at each clock edge and responds by floating the bus in the next clock cycle as shown in Figure 3-21. The CPU remains in the bus hold state until BOFF# is negated.

If the assertion of BOFF# interrupts a bus cycle, the bus cycle is restarted in its entirety following the negation of BOFF#. If KEN# was sampled by the processor before the cycle was aborted, it must be returned with the same value during the restarted cycle. The state of WB/WT# may be changed during the restarted cycle.

If BOFF# and BRDY# are active at the same clock edge, the CPU ignores BRDY#. Any data returned to the CPU with the BRDY# is also ignored. If BOFF# interrupts a burst read cycle, the CPU does not cache any data returned prior to BOFF#. However, this data may be used for internal CPU execution.

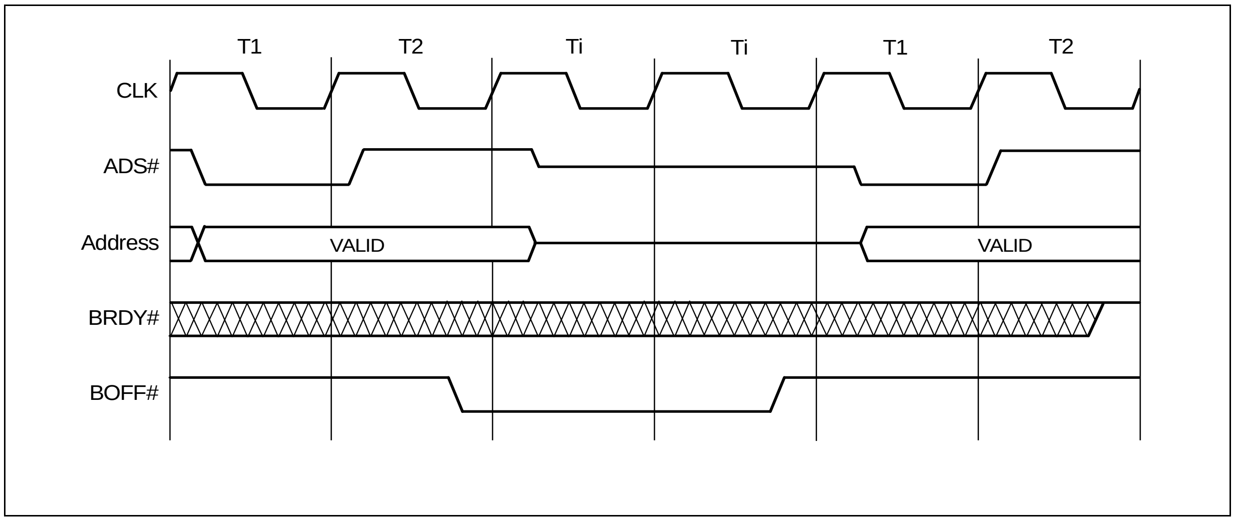


Figure 3-21. Back-Off Timing

3.3.9 Cache Inquiry Cycles

Cache inquiry cycles are issued by the system with the CPU in either a bus hold or address hold state. Bus hold is requested by asserting either HOLD or BOFF#, and address hold is requested by asserting AHOLD. The system initiates the cache inquiry cycle by asserting the EADS# input. The system must also drive the desired inquiry address on the address lines, and a valid state on the INV input.

In response to the cache inquiry cycle, the CPU checks to see if the specified address is present in the internal cache. If the address is present in the cache, the CPU checks the MESI state of the cache line. If the line is in the "exclusive" or "shared" state, the CPU asserts the HIT# output and changes the cache line state to "invalid" if the INV input was sampled logic high with EADS#.

If the line is in the "modified" state, the CPU asserts both HIT# and HITM#. The CPU then issues a bus cycle request to write the modified cache line to external memory. HITM# remains asserted until the write-back bus cycle completes. No additional cache inquiry cycles are accepted while HITM# is asserted. Write-back cycles always start at burst address 0. Once the write-back cycle has completed, the CPU changes the cache line state to "invalid" if the INV input was sampled logic high, or "shared" if the INV input was sampled low.

In addition to checking the cache, the CPU also snoops the internal line fill and cache write-back buffers in response to a cache inquiry cycle. The following sections describe the functional timing for cache inquiry cycles and the corresponding write-back cycles for the various types of inquiry cycles.

3.3.9.1 Inquiry Cycles Using HOLD/HLDA

Figure 3-22 illustrates an inquiry cycle where HOLD is used to force the CPU into a bus hold state. In this case, the system asserts HOLD and must wait for the CPU to respond with HLDA before issuing the cache inquiry cycle. To avoid address bus contention, EADS# should not be asserted until the second clock after HLDA as shown in the diagram. If the inquiry address hits on a modified cache line, HIT# and HITM# are asserted during the second clock following EADS#. Once HITM# asserts, the system must negate HOLD to allow the CPU to run the corresponding write-back cycle. The first cycle issued following negation of HLDA is the write-back bus cycle.

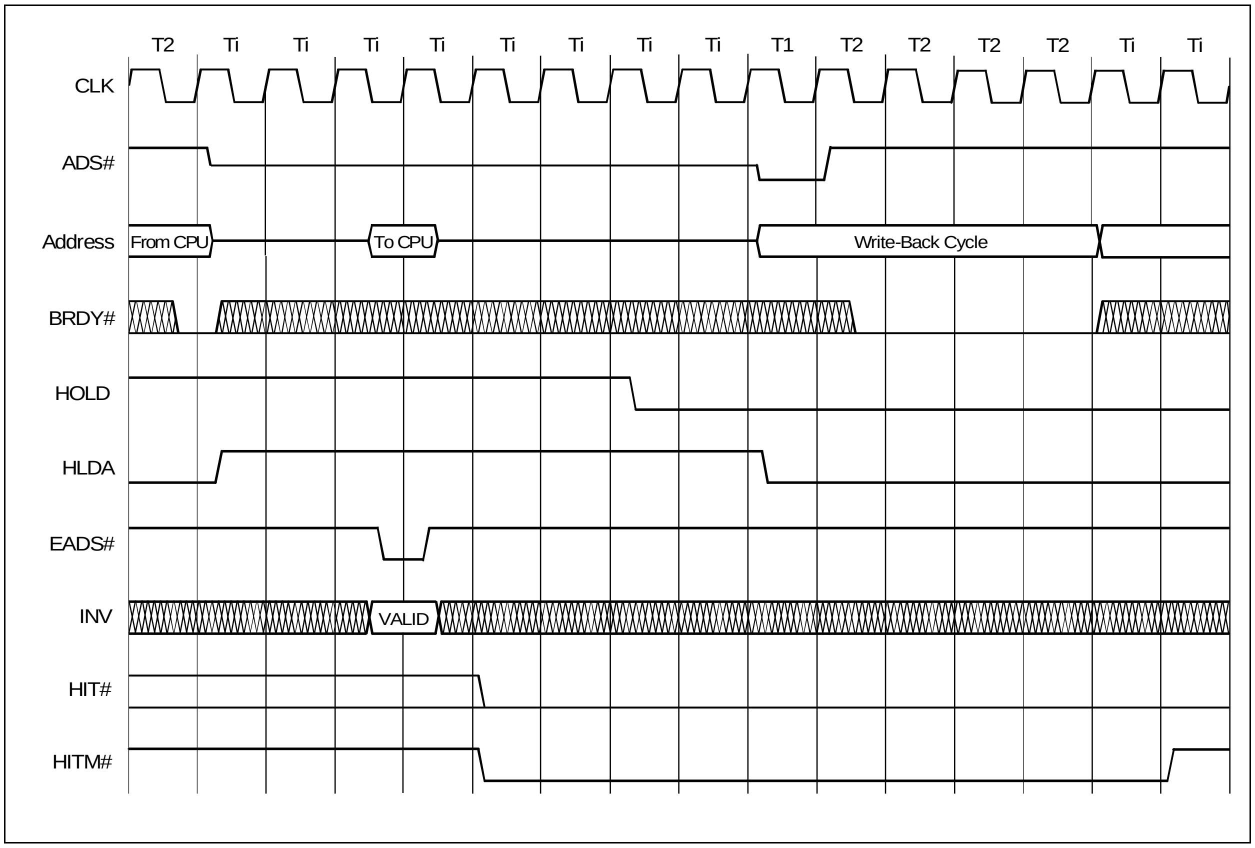


Figure 3-22. HOLD Inquiry Cycle that Hits on a Modified Line

3.3.9.2 Inquiry Cycles Using BOFF#

Figure 3-23 illustrates an inquiry cycle where BOFF# is used to force the CPU into a bus hold state. In this case, the system asserts BOFF# and the CPU immediately relinquishes control of the bus in the next clock. To avoid address bus contention, EADS# should not be asserted until the second clock edge after BOFF# as shown in the diagram. If the inquiry address hits on a modified cache line, HIT# and HITM# are asserted during the second clock following EADS#. Once HITM# asserts, the system must negate BOFF# to allow the CPU to run the corresponding write-back cycle. The first cycle issued following negation of BOFF# is the write-back bus cycle.

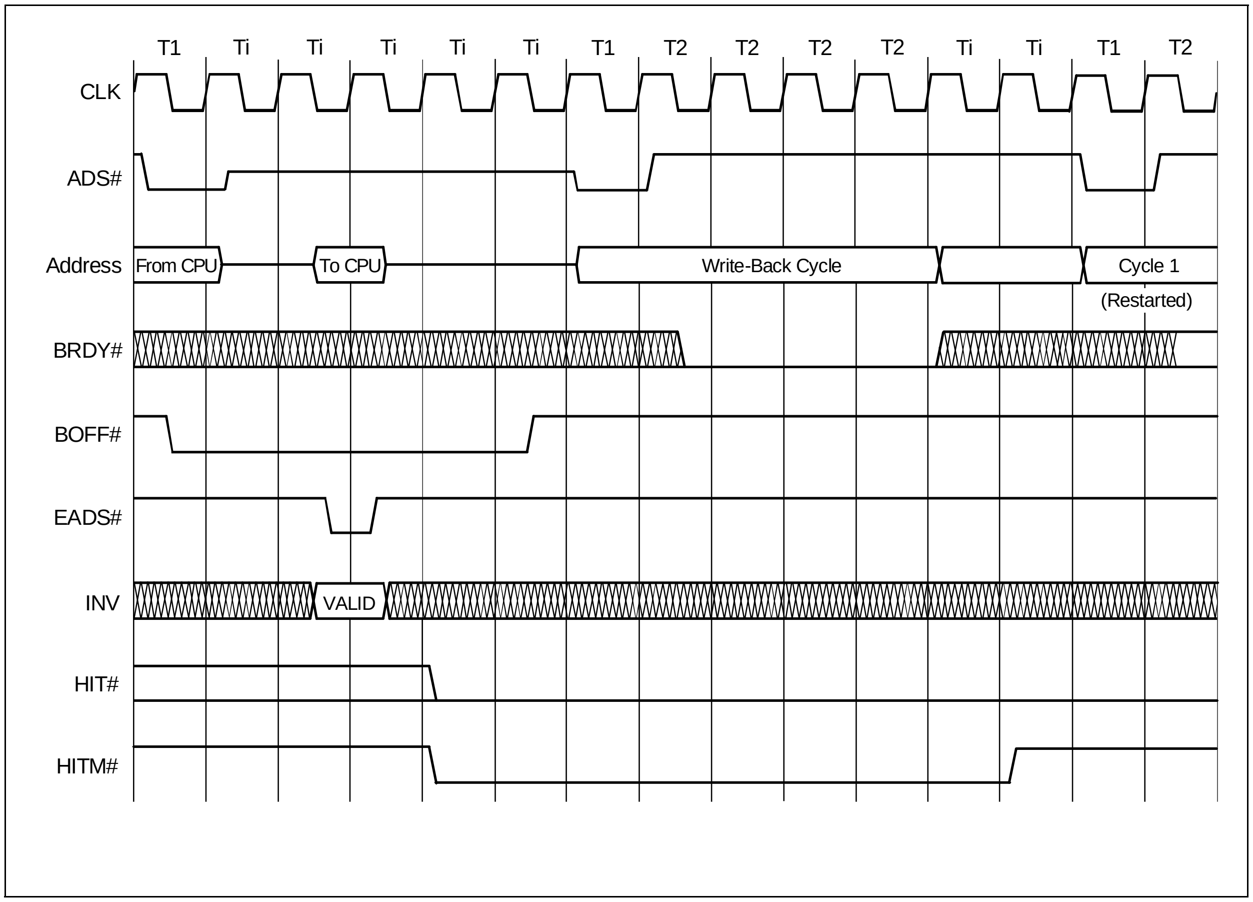
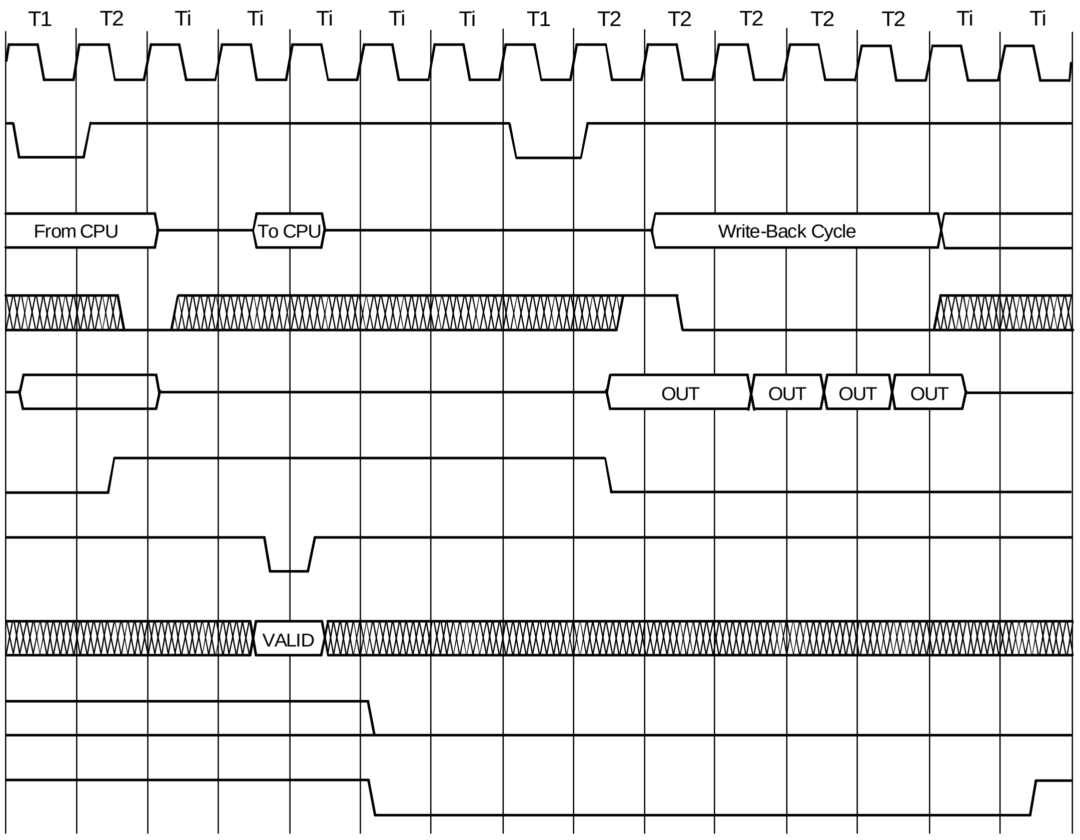


Figure 3-23. BOFF# Inquiry Cycle that Hits on a Modified Line

3.3.9.3 Inquiry Cycles Using AHOLD

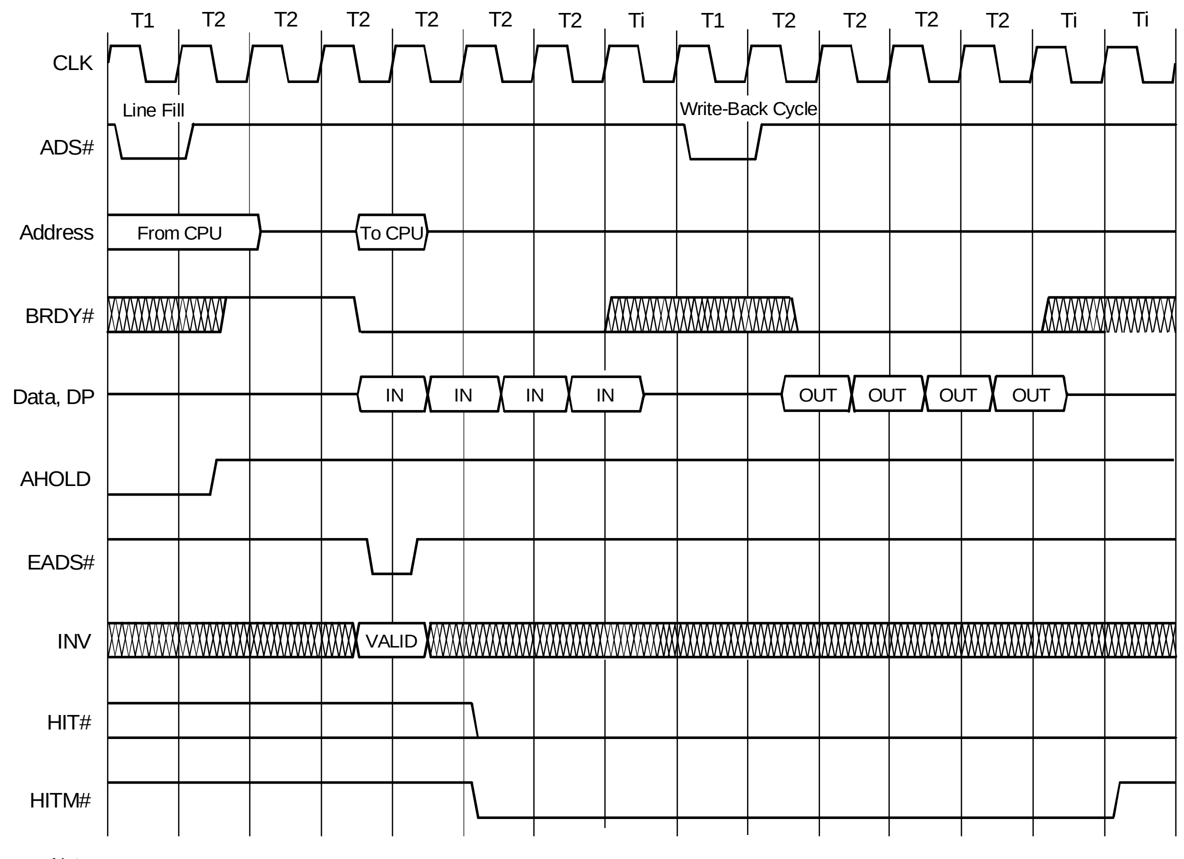
Figure 3-24 illustrates an inquiry cycle where AHOLD is used to force the CPU into an address hold state. In this case, the system asserts AHOLD and the CPU immediately floats the address bus in the next clock. To avoid address bus contention, EADS# should not be asserted until the second clock edge after AHOLD as shown in the diagram. If the inquiry address hits on a modified cache line, the CPU asserts HIT# and HITM# during the second clock following EADS#. The CPU then issues the write-back cycle even if AHOLD remains asserted. ADS# for the write-back cycle asserts two clocks after HITM# is asserted. To prevent the address bus and data bus from switching simultaneously, the system must adhere to the restrictions on negation of AHOLD as shown in Figure 3-24.



Restrictions on negating AHOLD:
1. During a write cycle, AHOLD should not be negated in the same clock that BRDY# is asserted.
2. During pipelined bus cycles, AHOLD should not be negated during the Td clock between a read cycle followed by a pipelined write cycle.
3. While HITM# is asserted, AHOLD should not be negated in the same clock that ADS# is asserted.

Figure 3-24. AHOLD Inquiry Cycle that Hits on a Modified Line

Figure 3-25 depicts an AHOLD inquiry cycle during a line fill. In this case, the write-back cycle occurs after the line fill is completed. At least one idle clock exists between the final BRDY# of the line fill and the ADS# for the write-back cycle. If the inquiry cycle hits on the address of the line fill that is in progress, the data from the line fill cycle is always used to complete the pending internal operation. However, the data is not placed in the cache if INV is sampled asserted with EADS#. The data is placed in the cache in a "shared" state if INV is sampled negated.



Note: If the inquiry cycle hits on the line fill in progress, the data from the line fill will be used to complete the pending internal operation. The line is not placed in the cache if INV is sampled asserted with EADS#. The line is placed in the cache in a "shared" state if INV is sampled negated with EADS#.

Figure 3-25. AHOLD Inquiry Cycle During a Line Fill

During cache inquiry cycles, the CPU performs address parity checking using A31-A5 and the AP signal. The CPU checks for even parity and asserts the APCHK# output if a parity error is detected. Figure 3-26 illustrates the functional timing of the APCHK# output.

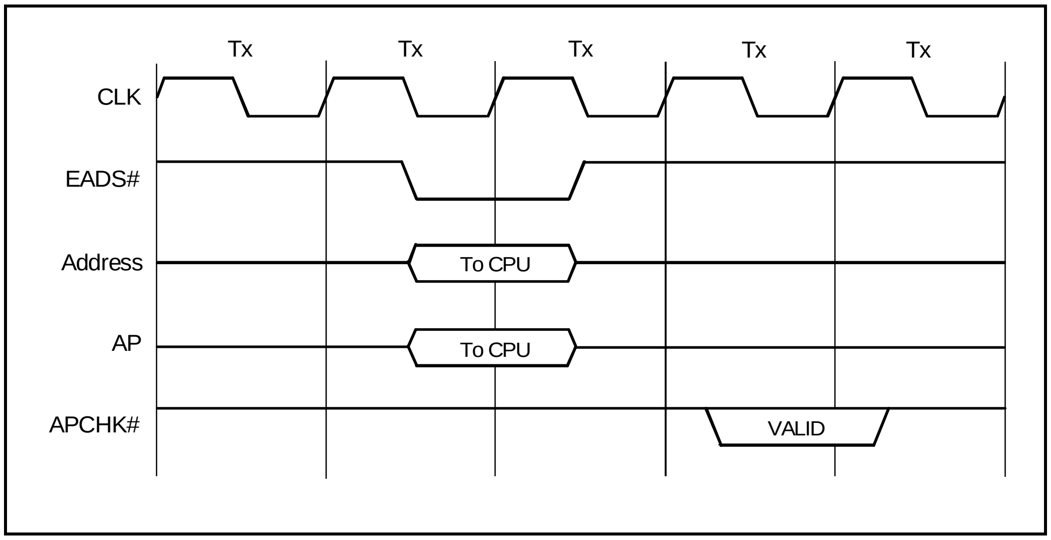


Figure 3-26. APCHK# Timing

3.3.10 Cache Inquiry Cycles During SMM Mode

It is assumed that while operating in SL-compatible mode SMM code and data are non-cacheable thereby precluding any inquiry cycles from hitting on cache lines containing modified SMM data. Therefore this section is only relevant while operating in Cyrix enhanced SMM mode.

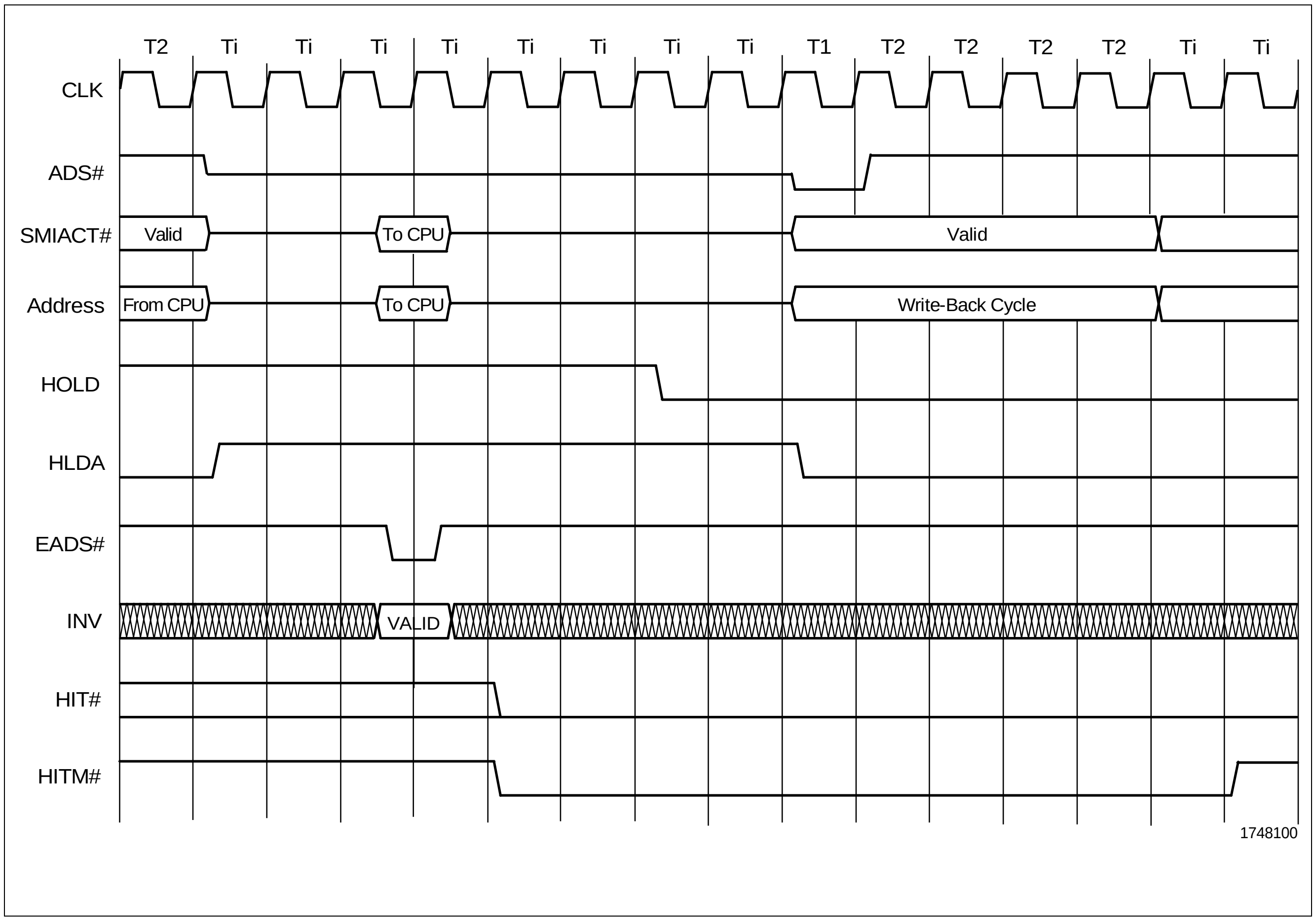


Figure 3-27. Hold Inquiry that Hits on a Modified Data Line

Cache inquiry cycles are issued by the system with the CPU in either a bus hold or address hold state. The SMIACT# pin is floated along with the other buses, and bus control signals as defined by the bus hold state. The SMIACT# pin follows the timing protocol shown in Figure 3-27 in regards to an inquiry during an address hold request. Bus hold is requested by asserting either HOLD or BOFF#, and address hold is requested by asserting AHOLD. The system initiates the cache inquiry cycle by asserting the EADS# input. The system must also drive the desired inquiry address on the address lines, and a valid state on the INV input.

In response to the cache inquiry cycle the CPU checks to see if the specified address is present in the internal cache. If the address is present in the cache, the CPU checks the MESI state of the cache line. If the line is in the “exclusive” or “shared” state, the CPU asserts the HIT# output and changes the cache line state to “invalid” if the INV input was sampled logic high with EADS#. If the line is in the “modified” state, the CPU asserts both HIT# and HITM#. The CPU then issues a bus cycle request to write the modified cache line to external memory. If the data to be written back is SMM data, the CPU asserts SMIACT# 1 cycle before asserting the ADS of the write back cycle. HITM# remains asserted until the write-back bus cycle completes. No additional cache inquiry cycles are accepted while HITM# is asserted. Write-back cycles always start at burst address 0. Once the write-back cycle has completed, the CPU changes the cache line state to “invalid” if the INV input was sampled logic high, or “shared” if the INV input was sampled low.

3.3.10.1 Inquiry Cycles Using BOFF, HOLD/HLDA

The system asserts HOLD or BOFF# to force the CPU into a bus hold state. The system must wait for the CPU to respond with HLDA before issuing the cache inquiry cycle, or in the case of BOFF# the CPU immediately relinquishes control to the bus in the next cycle. To avoid address bus contention, EADS# should not be asserted until the second clock edge after HLDA/BOFF#. If the inquiry address hits on a modified cache line, HIT# and HITM# are asserted during the second clock following EADS#. Once HITM# asserts, the system must negate HOLD/BOFF# to allow the CPU to run the corresponding write-back cycle. The first cycle issued following negation of HLDA/BOFF# is the write-back bus cycle. If this cycle is to SMM memory then SMIACT# is asserted, otherwise this cycle is run with SMIACT# high.

If SMIACT# was low prior to HLDA/BOFF# assertion and write-back cycle is intended for main memory then SMIACT# must be pulled high at least one clock prior to assertion of ADS# for the write-back cycle. See Figure 3-28 and Figure 3-29 (Page 3-57). If there is no write-back bus cycle to run and the next cycle to be run is to SMM memory then SMIACT# must be asserted at least 1 clock prior to assertion of ADS# as defined in Figure 3-30 (Page 3-58).

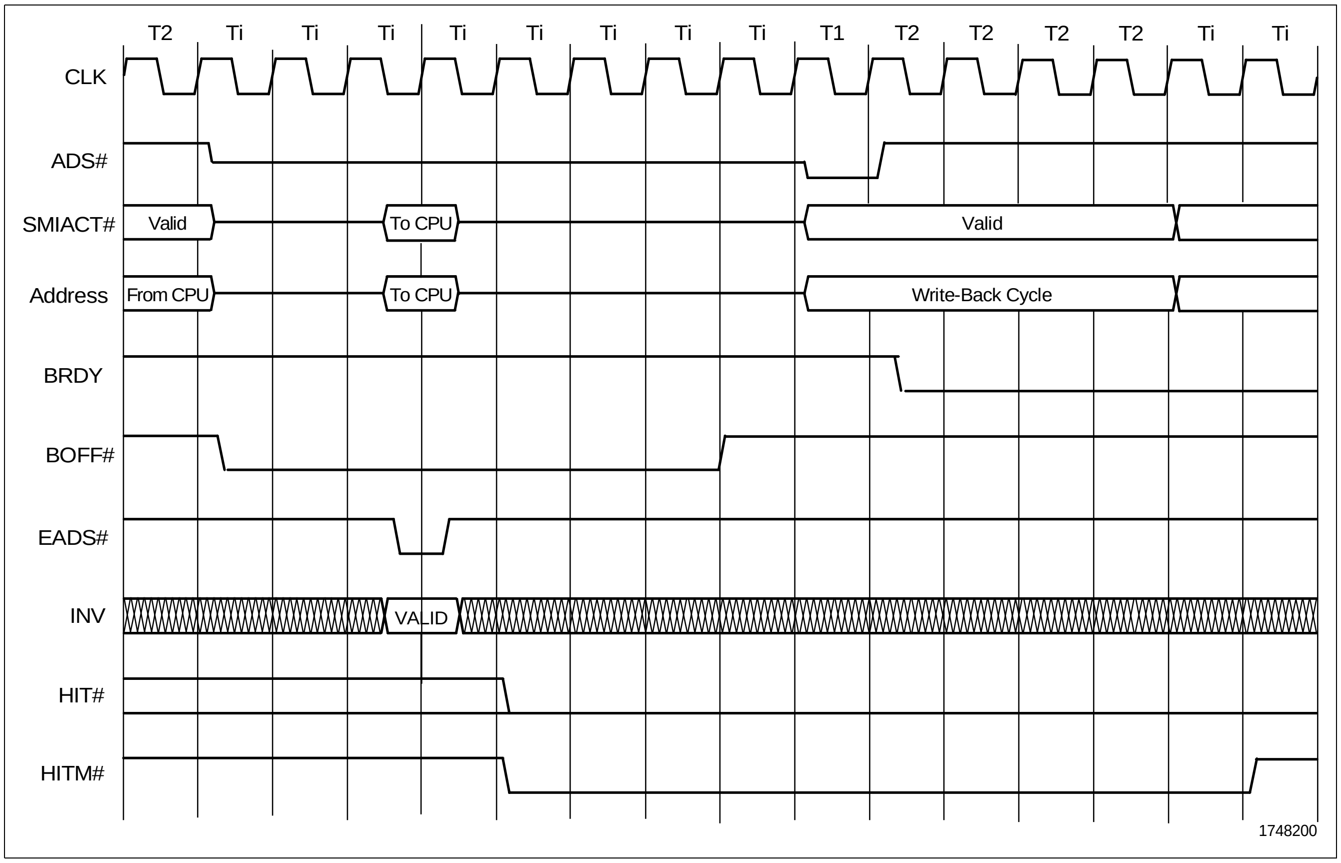


Figure 3-28. BOFF# Inquiry Cycle that Hits on a Modified Data Line

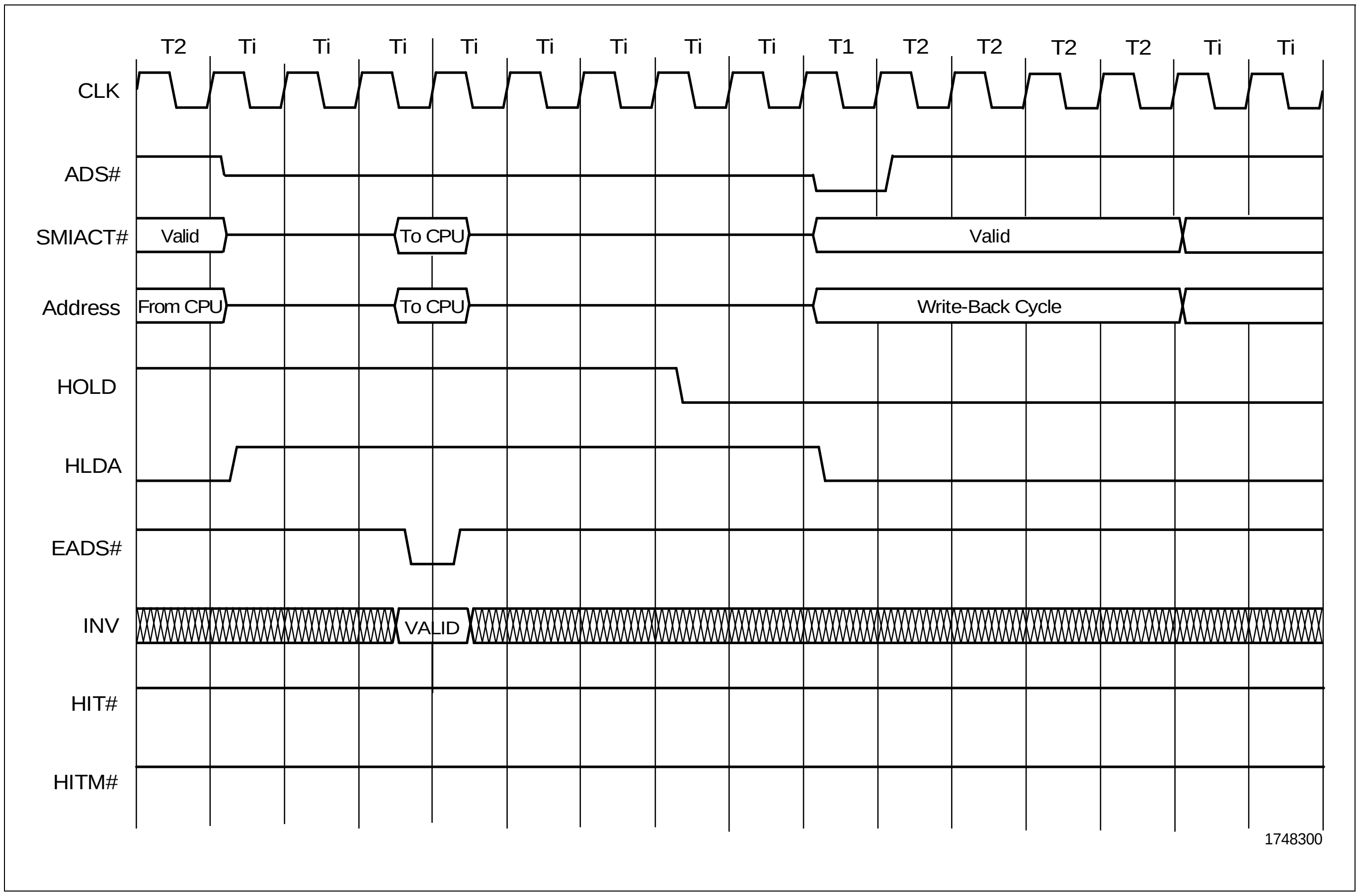


Figure 3-29. HOLD Inquiry Cycle that Misses the Cache While in SMM Mode

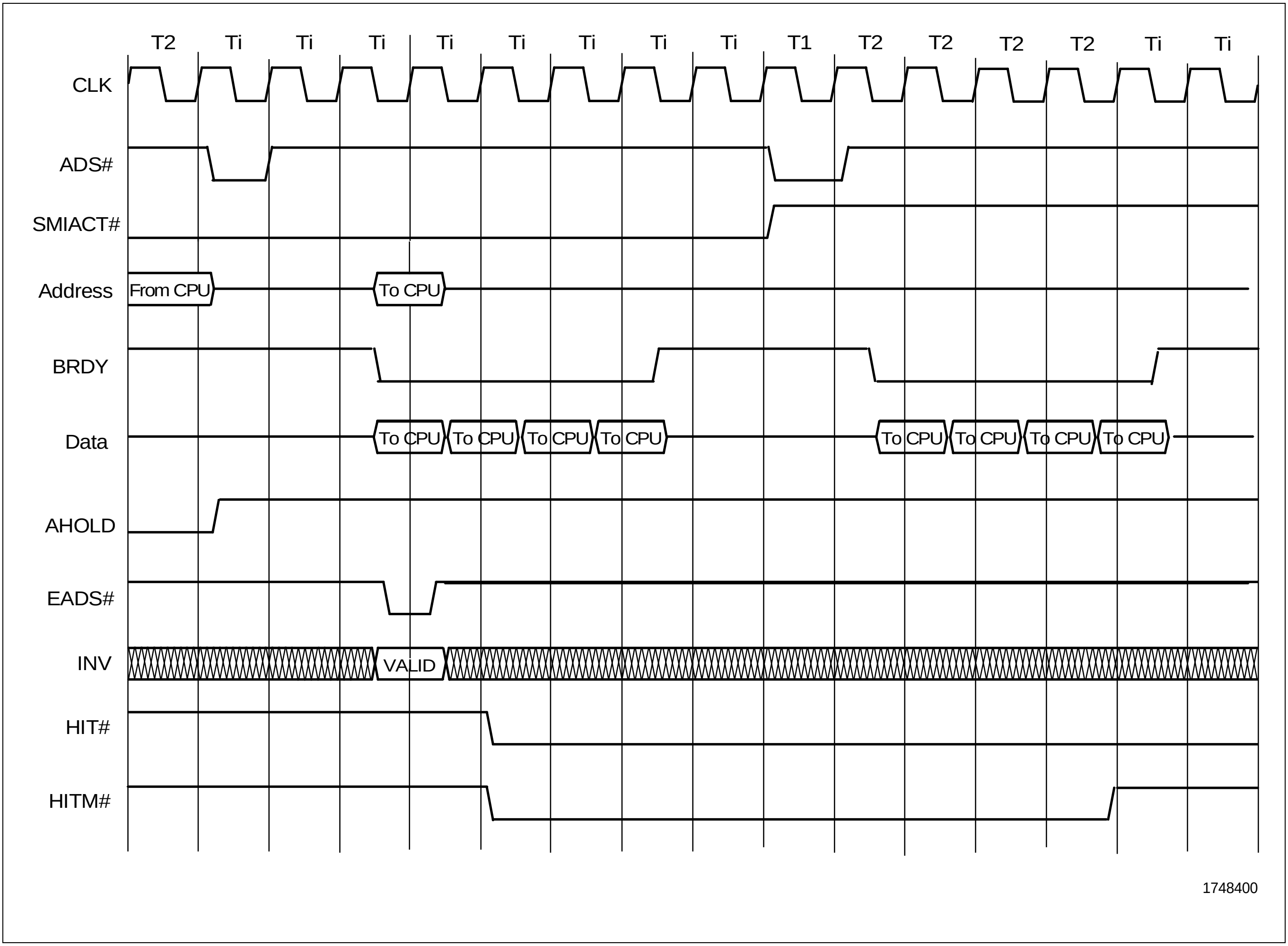


Figure 3-30. AHOLD Inquiry Cycle During a Line Fill from SMM Memory

3.3.10.2 Inquiry Cycles Using AHOLD

In this case, the system asserts AHOLD the CPU immediately floats the address bus in the next clock. To avoid address bus contention, EADS# should not be asserted until the second clock edge after AHOLD. If the inquiry address hits on a modified cache line the CPU asserts HIT# and HITM# during the second clock following EADS#. The CPU then issues the write-back cycle even if AHOLD remains asserted. If this cycle is to SMM memory then SMIACT# is asserted, otherwise this cycle is run with SMIACT# high. If SMIACT# was low prior to AHOLD assertion and write back cycle is intended for main memory then SMACT# must be pulled high at least one clock prior to assertion of ADS# for the write-back cycle. Likewise, if SMIACT# was high prior to AHOLD assertion and the write-back cycle is intended for SMM memory then SMIACT# must be pulled low at least one clock prior to assertion of ADS#. If there is no write-back bus cycle to run and the next cycle to be run is to SMM memory then SMIACT# must be asserted at least one clock prior to assertion of ADS#.

The following timing diagram depicts an AHOLD inquiry cycle during a line fill from SMM memory. In this case, the write-back cycle occurs after the line fill is completed, and one clock after SMIACT# is set to a logic high provided the write-back cycle is to main memory. For this case, if the write-back cycle is to SMM memory then the one clock setup time criterion for SMIACT# to ADS# is met and the write-back cycle can start immediately.

3.3.11 Power Management Interface

SUSP# Initiated Suspend Mode

The IBM 6x86MX CPU enters suspend mode when the SUSP# input is asserted and execution of the current instruction, any pending decoded instructions and associated bus cycles are completed. A stop grant bus cycle is then issued and the SUSPA# output is asserted. The CPU responds to SUSP# and asserts SUSPA# only if the SUSP bit is set in the CCR2 configuration register.

SUSP# is sampled (Figure 3-31) on the rising edge of CLK. SUSP# must meet specified setup and hold times to be recognized at a particular CLK edge. The time from assertion of SUSP# to activation of SUSPA# varies depending on which instructions were decoded prior to assertion of SUSP#. The minimum time from SUSP# sampled active to SUSPA# asserted is eight CLKs. As a maximum, the CPU may execute up to two instructions and associated bus cycles prior to asserting SUSPA#. The time required for the CPU to deactivate SUSPA# once SUSP# has been sampled inactive is five CLKs.

If the CPU is in a hold acknowledge state and SUSP# is asserted, the CPU may or may not enter suspend mode depending on the state of the CPU internal execution pipeline. If the CPU is in a SUSP# initiated suspend mode, one occurrence of NMI, INTR and SMI# is stored for execution once suspend mode is exited. The IBM 6x86MX CPU also recognizes and acknowledges the HOLD, AHOLD, BOFF# and FLUSH# signals while in suspend mode.

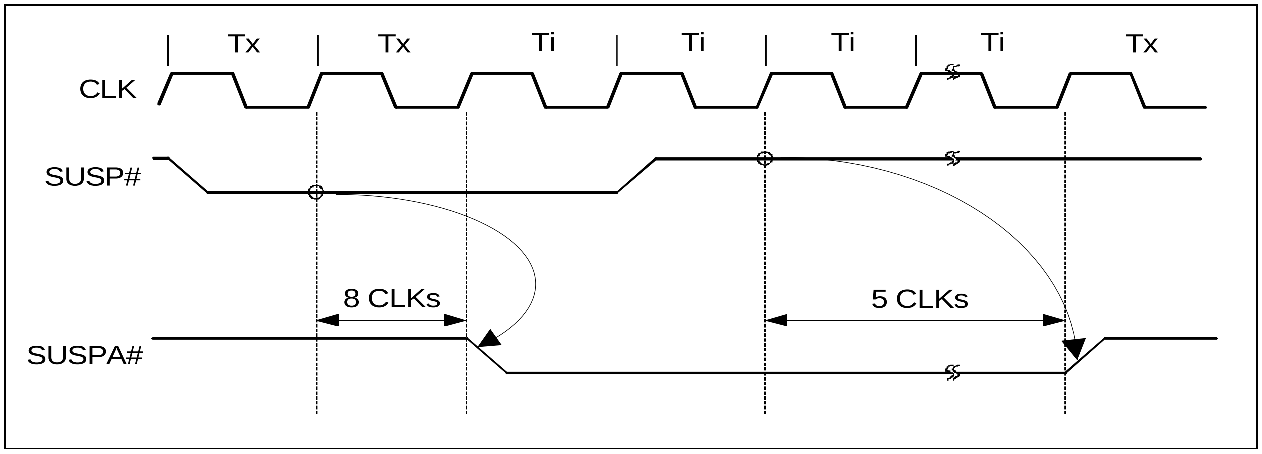


Figure 3-31. SUSP# Initiated Suspend Mode

HALT Initiated Suspend Mode

The CPU also enters suspend mode as a result of executing a HALT instruction if the HALT bit in CCR2 is set. The SUSPA# output is asserted no later than 40 CLKs following BRDY# sampled active for the HALT bus cycle as shown in Figure 3-32. Suspend mode is then exited upon recognition of an NMI, an unmasked INTR or an SMI#. SUSPA# is deactivated 10 CLKs after sampling of an active interrupt.

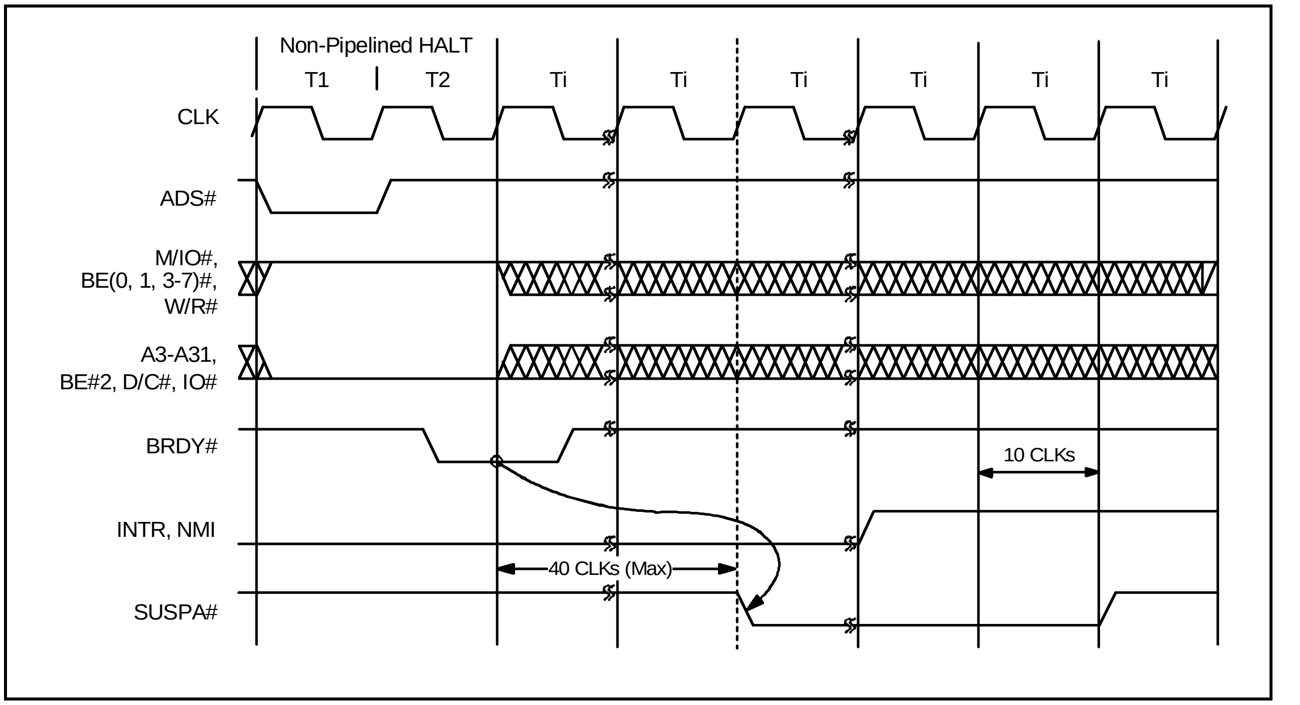


Figure 3-32. HALT Initiated Suspend Mode

Stopping the Input Clock

Once the CPU has entered suspend mode, the input clock (CLK) can be stopped and restarted without loss of any internal CPU data. The CLK input can be stopped at either a logic high or logic low state.

The CPU remains suspended until CLK is restarted and suspend mode is exited as described earlier. While the CLK is stopped, the CPU can no longer sample and respond to any input stimulus.

Figure 3-33 illustrates the recommended sequence for stopping the CLK using SUSP# to initiate suspend mode. CLK may be started prior to or following negation of the SUSP# input. The system must allow sufficient time for the CPU's internal PLL to lock to the desired frequency before exiting suspend mode.

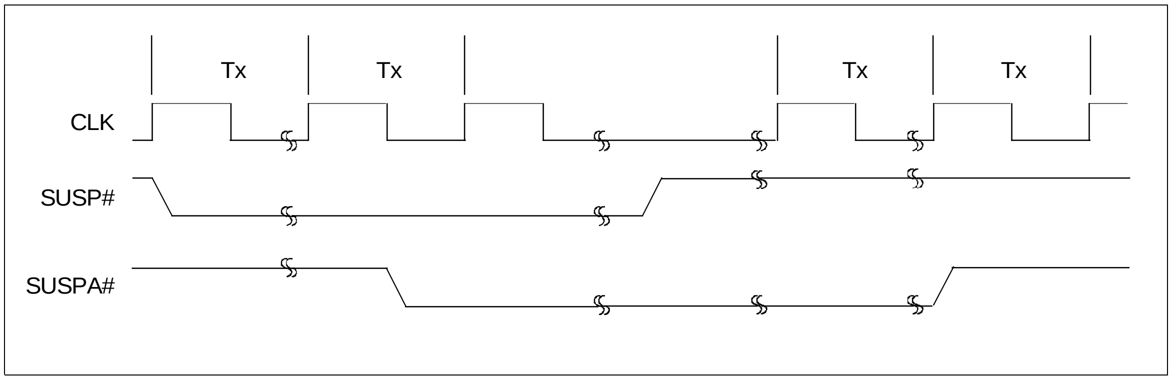


Figure 3-33. Stopping CLK During Suspend Mode