Error Detection 22

The embedded Pentium processor incorporates a number of data integrity features that are focused on the detection and limited recovery of errors. The data integrity features provide capabilities for error detection of the internal devices and the external interface. The processor also provides the capability to obtain maximum levels of error detection by incorporating Functional Redundancy Checking (FRC) support. Error detecting circuits in the embedded Pentium processor do not limit the operating frequency of the chip.

The data integrity features can be categorized as (1) internal error detection, (2) error detection at the bus interface, and (3) FRC support.

22.1 Internal Error Detection

Detection of errors of a majority of the devices in the processor is accomplished by employing parity checking in the large memory arrays of the chip. The data and instruction caches (both storage and tag arrays), translation lookaside buffers, and microcode ROM are all parity protected. The following describes the parity checking employed in the major memory arrays in the processor (MESI status bits are not parity protected):

- Parity bit per byte in the data cache storage array.
- Parity bit per entry in the data cache tag array.
- Four Parity bits: One for each of the even upper, even lower, odd upper, odd lower bits of an instruction cache line.
- Parity bit per entry in the instruction cache tag array.
- Parity bit per entry in both the data and instruction TLBs storage arrays.
- Parity bit per entry in both the data and instruction TLBs tag arrays.
- Parity bit per entry in the microcode ROM.

Parity checking as described above provides error detection coverage of 53% of the on-chip devices. This error detection coverage number also includes the devices in the branch target buffer since branch predictions are always verified.

If a parity error has occurred internally, processor operation can no longer be trusted. Normally, a parity error on a read from an internal array will cause the processor to assert the IERR# pin and then shutdown. (Shutdown will be entered assuming it is not prevented from doing so by the error.); however, if TR1.NS is set, IERR# will not result in processor shutdown. Execution will continue, but operation will not be reliable. Parity errors on reads during normal instruction execution, reads during a flush operation, reads during BIST and testability cycles, and reads during inquire cycles will cause IERR# to be asserted. The IERR# pin will be asserted for one clock for each clock a parity error is detected and may be latched by the system. The IERR# pin is a glitch free signal, so no spurious assertions of IERR# will occur.

In general, internal timing constraints of the processor do not allow the inhibition of writeback cycles caused by inquire cycles, FLUSH# assertion or the WBINVD instruction when a parity error is encountered. In those cases where an internal parity error occurred during the generation of a writeback cycle, and that cycle was not able to be inhibited, the IERR# pin can be used to recognize that the writeback should be ignored. If an internal parity error occurs during a flush operation, the processor will assert the IERR# pin as stated above, and the internal caches will be left in a partially flushed state. The flush, flush acknowledge, or writeback special cycles will not be run.

22.2 Error Detection at the Processor Interface

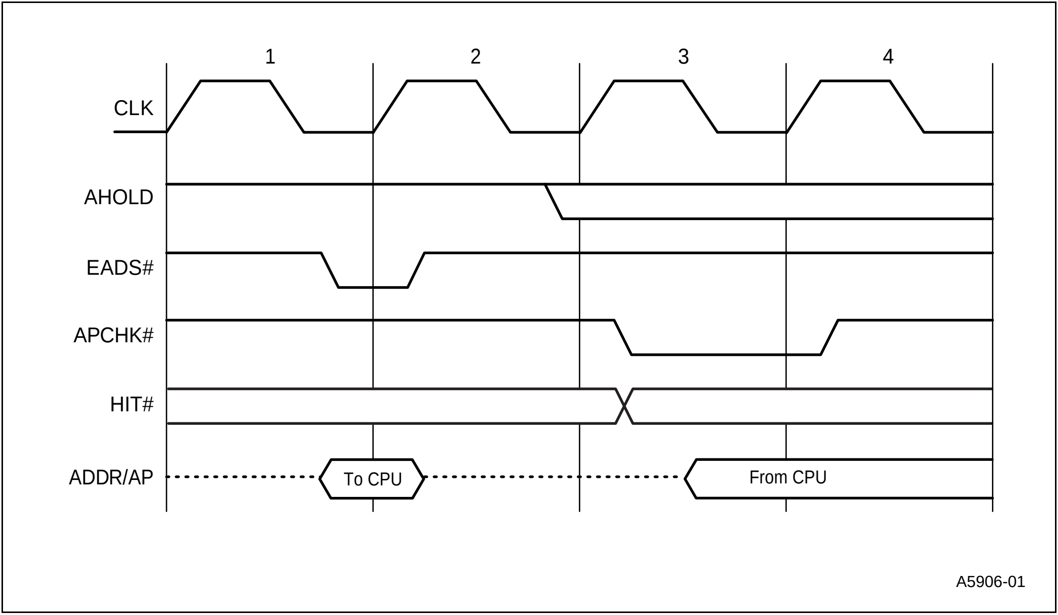
The processor provides parity checking on the external address and data buses. There is one parity bit for each byte of the data bus and one parity bit for bits A31–A5 of the address bus.

22.2.1 Address Parity

A separate and independent mechanism is used for parity checking on the address bus during inquire cycles. Even address parity is driven along with the address bus during all processor initiated bus cycles and checked during inquire cycles. When the processor is driving the address bus, even parity is driven on the AP pin. When the address bus is being driven into the processor during an inquire cycle, this pin is sampled in any clock in which EADS# is sampled asserted. APCHK# is driven with the parity status two clocks after EADS# is sampled active. The APCHK# output (when active) indicates that a parity error has occurred on the address bus during an inquire. Figure 22-1 depicts an address parity error during an inquire cycle. For additional timing diagrams which show address parity, see Chapter 19, “Bus Functional Description.” The APCHK# pin will be asserted for one clock for each clock a parity error is detected and may be latched by the system. The APCHK# pin is a glitch free signal, so no spurious assertions of APCHK# will occur.

In the event of an address parity error during inquire cycles, the internal snoop will not be inhibited. If the inquire hits a modified line in this situation and an active AHOLD prevents the processor from driving the address bus, the processor will potentially writeback a line at an address other than the one intended. If the processor is not driving the address bus during the writeback cycle, it is possible that memory will be corrupted.

Figure 22-1. Inquire Cycle Address Parity Checking

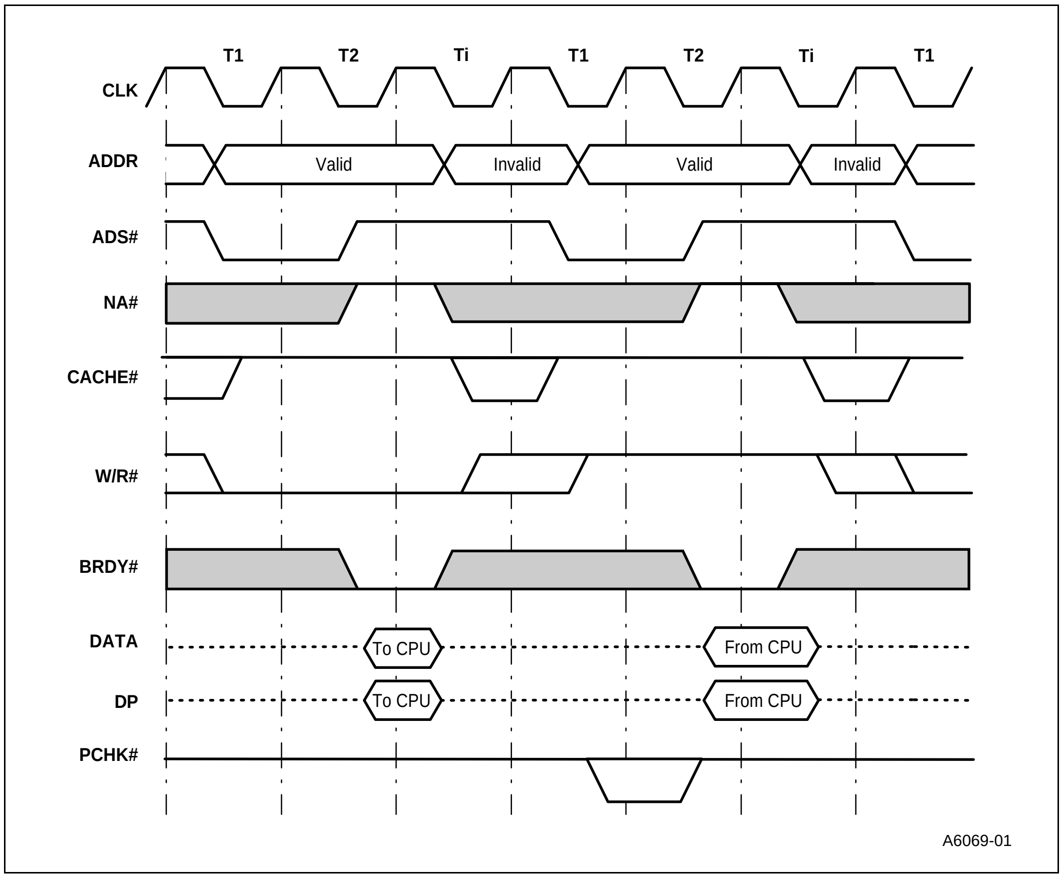


Driving APCHK# is the only effect that bad address parity has on the processor. It is the responsibility of the system to take appropriate action if a parity error occurs. If parity checks are not implemented in the system, the APCHK# pin may be ignored.

22.2.2 Data Parity

Even data parity is driven on the DP7–DP0 pins in the same clock as the data bus is driven during all processor initiated data write cycles. During reads, even parity information may be driven back to the processor on the data parity pins along with the data being returned. Parity status for data sampled is driven on the PCHK# pin two clocks after the data is returned. PCHK# is driven low if a data parity error was detected, otherwise it is driven high. The PCHK# pin will be asserted for one clock for each clock a parity error is detected and may be latched by the system. The PCHK# pin is a glitch free signal, so no spurious assertions of PCHK# will occur. Figure 22-2 shows when the data parity (DP) pins are driven/sampled and when the PCHK# pin is driven. For additional timing diagrams that show data parity, see Chapter 19, “Bus Functional Description.”

Figure 22-2. Data Parity During a Read and Write Cycle



Driving PCHK# is the only effect that bad data parity has on the processor. It is the responsibility of the system to take appropriate action if a parity error occurs. If parity checks are not implemented in the system, the PCHK# pin may be ignored.

22.2.2.1 Machine Check Exception as a Result of a Data Parity Error

The PEN# input determines whether a machine check interrupt will be taken as a result of a data parity error. If a data parity error occurs on a read for which PEN# was asserted, the physical address and cycle information of the cycle causing the parity error will be saved in the Machine Check Address Register and the Machine Check Type Register. If in addition, the CR4.MCE is set to 1, the machine check exception is taken. See "Machine Check Exception" on page 22-396 for more information.

The parity check pin, PCHK#, is driven as a result of read cycles regardless of the state of the PEN# input.

22.2.3 Machine Check Exception

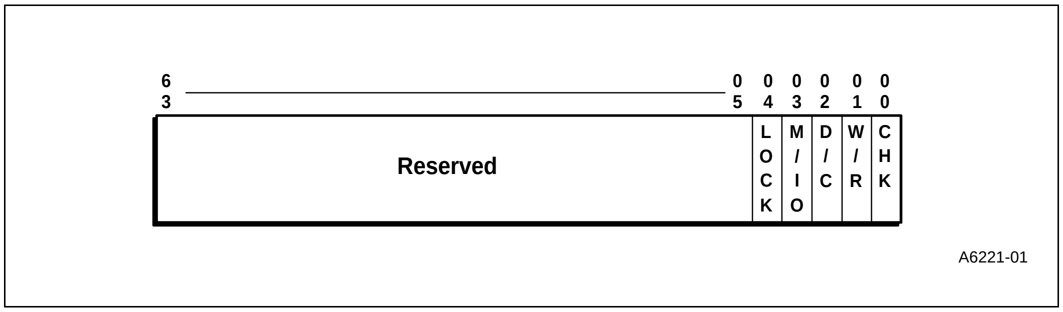
As mentioned in the earlier section, a new exception has been added to the processor. This is the machine check exception which resides at interrupt vector 18 (decimal). In processors previous to the Pentium processor, interrupt vector 18 was reserved and, therefore, there should be no interrupt routine located at vector 18. For compatibility, the MCE bit of the CR4 register will act as the machine check enable bit. When set to "1," this bit will enable the generation of the machine check exception. When reset to "0," the processor will inhibit generation of the machine check exception. CR4.MCE will be cleared on processor reset. In the event that a system is using the machine check interrupt vector for another purpose and the Machine Check Exception is enabled, the interrupt routine at vector 18 must examine the state of the CHK bit in the Machine Check Type register to determine the cause of its activation. Note that at the time the system software sets CR4.MCE to 1, it must read the Machine Check Type register in order to clear the CHK bit.

The Machine Check Exception is an abort; that is, it is not possible to reliably restart the instruction stream or identify the instruction causing the exception. Therefore, the exception does not allow the restart of the program that caused the exception. The processor does not generate an error code for this exception. Since the machine check exception is synchronous to a bus cycle and not an instruction, the IP pushed on to the stack may not be pointing to the instruction which caused the failing bus cycle.

The Machine Check Exception can be caused by one of two events: 1) Detection of data parity error during a read when the PEN# input is active, or 2) The BUSHCK# input being sampled active. When either of these events occur, the cycle address and type will be latched into the Machine Check Address (MCA) and Machine Check Type (MCT) registers (independent of the state of the CR4.MCE bit). If in addition, the CR4.MCE is "1," a machine check exception will occur. When the MCA and MCT registers are latched, the MCT.CHK bit is set to "1" indicating that their contents are valid (Figure 22-3).

The Machine Check Address register, and the Machine Check Type register are model specific, read only registers. The Machine Check Address register is a 64-bit register containing the physical address for the cycle causing the error. The Machine Check Type register is a 64-bit register containing the cycle specification information, as defined in Figure 22-3. These registers are accessed using the RDMSR instruction. When the MCT.CHK is zero, the contents of the MCT and MCA registers are undefined. When the MCT register is read (using the RDMSR instruction), the CHK bit is reset to zero. Therefore, software must read the MCA register before reading the MCT register.

Figure 22-3. Machine Check Type Register



The bits in the Machine Check Type Register are defined as follows:

CHK:	This bit is set to 1 when the Machine Check Type register is latched and is reset to 0 after the Machine Check Type register is read via the RDMSR instruction. In the event that the Machine Check Type register is latched in the same clock in which it is read, the CHK bit will be set. The CHK bit is reset to “0” on assertion of RESET. When the CHK bit is “0,” the contents of the MCT and MCA registers are undefined.
M/IO#, D/C#, W/R#:	These cycle definition pins can be decoded to determine if the cycle in error was a memory or I/O cycle, a data or code fetch, and a read or a write cycle. (See the embedded Pentium processor datasheets for detailed pin definitions.)
LOCK:	Set to “1” if LOCK# is asserted for the cycle

22.2.4 Bus Error

The BUSCHK# input provides the system a means to signal an unsuccessful completion of a bus cycle. This signal is sampled on any edge in which BRDY# is sampled, for reads and writes. If this signal is sampled active, then the cycle address and type will be latched into the Machine Check Address and Machine Check Type registers. If in addition, the CR4.MCE bit is set to 1, the processor will be vectored to the machine check exception.

Even if BUSCHK# is asserted in the middle of a cycle, BRDY# must be asserted the appropriate number of clocks required to complete the bus cycle. The purpose of BUSCHK# is to act as an indication of an error that is synchronous to bus cycles. If the machine check interrupt is not enabled, i.e., the MCE bit in the CR4 register is zero, then an assertion of BUSCHK# will not cause the processor to vector to the machine check exception.

The embedded Pentium processor can remember only one machine check exception at a time. This exception is recognized on an instruction boundary. If BUSCHK# is sampled active while servicing the machine check exception for a previous BUSCHK#, it will be remembered by the processor until the original machine check exception is completed. It is then that the processor will service the machine check exception for the second BUSCHK#. Note that only one BUSCHK# will be remembered by the processor while the machine exception for the previous one is being serviced.

When the BUSCHK# is sampled active by the processor, the cycle address and cycle type information for the failing bus cycle is latched upon assertion of the last BRDY# of the bus cycle. The information is latched into the Machine Check Address and Machine Check Type registers respectively. However, if the BUSCHK# input is not deasserted before the first BRDY# of the next bus cycle, and the machine check exception for the first bus cycle has not occurred, then new information will be latched into the MCA and MCT registers, over-writing the previous information at the completion of this new bus cycle. Therefore, in order for the MCA and MCT registers to report the correct information for the failing bus cycle when the machine check exception for this cycle is taken at the next instruction boundary, the system must deassert the BUSCHK# input immediately after the completion of the failing bus cycle and before the first BRDY# of the next bus cycle is returned.

22.2.5 Functional Redundancy Checking

Functional Redundancy Checking (FRC) in the embedded Pentium processor will provide maximum error detection (>99%) of on-chip devices and the processor's interface. A "checker" processor that executes in lock step with the "master" processor is used to compare output signals every clock.

Note: The embedded Pentium processor with MMX technology does not support FRC. Also, FRC is not supported in Dual processor designs.

Two embedded Pentium processors are required to support FRC. Both the master and checker must be of the same stepping and same bus fraction. The processor configured as a master operates according to bus protocol described in this document. The outputs of the checker processor are three-stated (except IERR#, TDO, PICD0, PICD1—however, these signals are not part of FRC) so the outputs of the master can be sampled. If the sampled value differs from the value computed internally by the checker, the checker asserts the IERR# output to indicate an error. A master-checker pair should have all pins except FRCMC#, IERR#, PICD0, PICD1 and TDO tied together.

The processors are configured either as a master or a checker by driving the FRCMC# input to the appropriate level while RESET is asserted. If sampled low during reset, the processor enters checker mode and three-states all outputs except IERR# and TDO (IERR# is driven inactive during reset). This feature is provided to prevent bus contention before reset is completed. The final master/checker configuration is determined when RESET transitions from high to low. The final master/checker configuration may not be changed other than by a subsequent RESET.

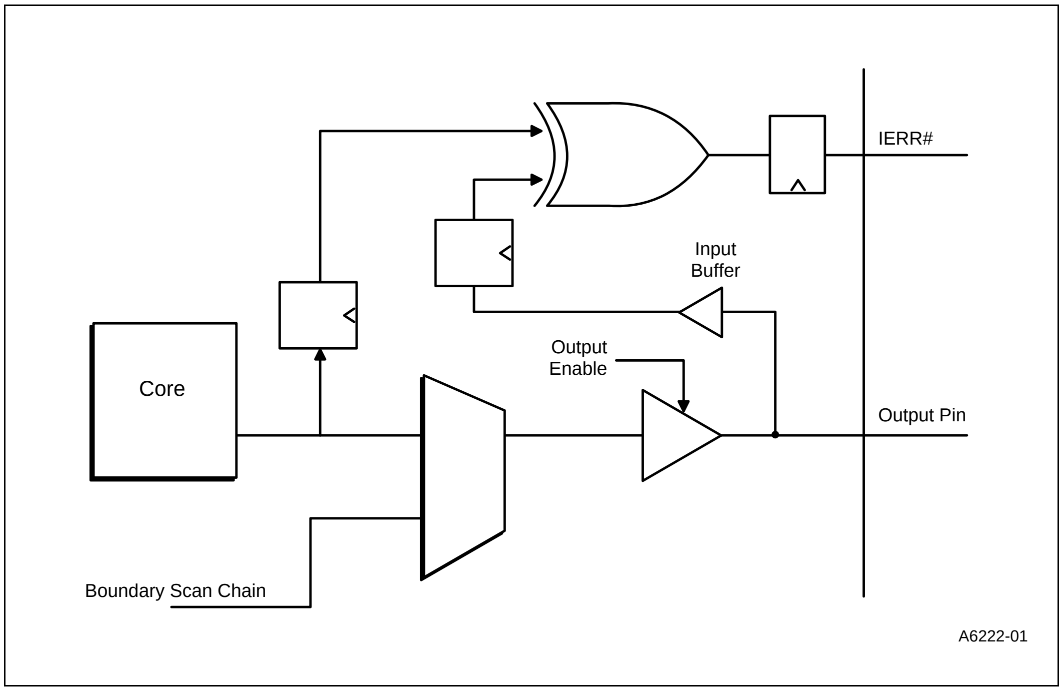
The IERR# pin reflects the result of the master-checker comparison. It is asserted for one clock, two clocks after the mismatch. It is asserted for each detected mismatch, so IERR# may be low for more than one consecutive clock. During the assertion of RESET, IERR# will be driven inactive. After RESET is deasserted, IERR# will not be asserted due to a mismatch until two clocks after the ADS# of the first bus cycle (i.e., in the third clock of the first bus cycle). IERR# will reflect pin comparisons thereafter. Note that IERR# may be asserted due to an internal parity error prior to the first bus cycle. It is possible for FRC mismatches to occur in the event that an undefined processor state is driven off-chip, therefore no processor state should be stored without having been previously initialized.

In order for the master-checker pair to operate correctly, the system must be designed such that the master and the checker sample identical input states in the same clock. All asynchronous inputs should change state in such a manner that both the master and checker sample them in the same state in the same clock. The simplest way to do this is to design all asynchronous inputs to be synchronously controlled.

The TDO pin is not tested by FRC since it operates on a separate clock. Note that it is possible to use boundary scan to verify the connection between the master and checker by scanning into one, latching the outputs of the other and then scanning out. 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.

Figure 22-4 illustrates the configuration of output pins with respect to FRC. The comparators at each output compare the value of the package pin with the value being driven from the core to that pin, not the value driven by boundary scan to that pin. Therefore, during the use of boundary scan, FRC mismatches (IERR# assertion) can be expected to occur.

Figure 22-4. Conceptual IERR# Implementation for FRC


IERR#
Input Buffer
Output Enable
Core
Output Pin
Boundary Scan Chain
A6222-01