Error Checking and Correcting

16

This chapter describes the Error Checking and Correcting (ECC) mechanism used in both the R4000 and R4400 processors.

This chapter also contains a description of the Master/Checker mode used in the R4400 processor.

16.1 Error Checking in the Processor

ECC code allows the processor to detect and sometimes correct errors made when moving data from one place to another.

Two major types of data errors can occur in data transmission:

- hard errors, which are permanent, arise from broken interconnects, internal shorts, or open leads
- soft errors, which are transient, are caused by system noise, power surges, and alpha particles.

Hard errors must be corrected by physical repair of the damaged equipment and restoration of data from backup. Soft errors can be corrected by using error checking and correcting codes.

Types of Error Checking

The processor uses two types of error checking: parity (error detection only), and single-bit error correction/double-bit error detection (SECDED).

Parity Error Detection

Parity is the simplest error detection scheme. By appending a bit to the end of an item of data—called a *parity bit*—single bit errors can be detected; however, these errors cannot be corrected.

There are two types of parity:

- **Odd Parity** adds 1 to any even number of 1s in the data, making the total number of 1s odd (including the parity bit).
- **Even Parity** adds 1 to any odd number of 1s in the data, making the total number of 1s even (including the parity bit).

Odd and even parity are shown in the example below:

Data(3:0)	Odd Parity Bit	Even Parity Bit
0 0 1 0	0	1

The example above shows a single bit in **Data(3:0)** with a value of 1; this bit is **Data(1)**.

- In even parity, the parity bit is set to 1. This makes 2 (an even number) the total number of bits with a value of 1.
- Odd parity makes the parity bit a 0 to keep the total number of 1-value bits an odd number—in the case shown above, the single bit **Data(1)**.

The example below shows odd and even parity bits for various data values:

Data(3:0)	Odd Parity Bit	Even Parity Bit
0 1 1 0	1	0
0 0 0 0	1	0
1 1 1 1	1	0
1 1 0 1	0	1

Parity allows single-bit error detection, but it does not indicate which bit is in error—for example, suppose an odd-parity value of 00011 arrives. The last bit is the parity bit, and since odd parity demands an odd number (1,3,5) of 1s, this data is in error: it has an even number of 1s. However it is impossible to tell *which* bit is in error. To resolve this problem, SECDED ECC was developed.

SECDED ECC Code

The ECC code chosen for processor secondary cache data and tag is single-bit error correction and double-bit error detection (SECDED) code.† SECDED ECC code is an improvement upon the parity scheme; not only does it detect single- and certain multi-bit errors, it corrects single-bit errors.

† The 64-bit data code is a modification of one of the 64-bit codes proposed by M. Y. Hsiao, to include the ability to detect 3- and 4-bit errors within a nibble. The 25-bit tag code was created using the patterns observed in the 64-bit data code.

Secondary Cache Data Bus SECDED Code

The SECDED code protecting secondary cache data bus has the properties listed below:

- It corrects single-bit errors.
- It detects double-bit errors.
- It detects 3- or 4-bit errors within a nibble†.
- It provides 64 data bits protected by 8 check bits, and yields 8-bit syndromes (the *syndrome* is a generated value used to detect an error, and locate the position of the single bit in error).
- It is a minimal-length code; each parity tree used to generate the 8-bit syndrome has only 27 inputs, the minimum number possible.
- It provides byte Exclusive-ORs (XORs) of the data bits as part of the XOR trees used to build the parity generators. This allows selection of byte parity out of the XOR trees that generate or check the code.
- Single-bit errors are indicated either by syndromes that contain exactly three 1s, or by syndromes that contain exactly five 1s (in which bits 0-3 or bits 4-7 of the syndrome are all 1s).‡
- Double-bit errors are indicated by syndromes that contain an even number of 1s.
- 3-bit errors within a nibble are indicated by syndromes that contain five 1s, in which bits 0-3 of the syndrome and bits 4-7 of the syndrome are not all 1s.
- 4-bit errors within a nibble are indicated by syndromes that contain four 1s. Because this is an even number of 1s, 4-bit errors within a nibble look like double-bit errors.

† A nibble is defined here as any group of four bits located within the vertical rules of Figure 16-1.

‡ This makes it possible to decode the syndrome to find which data bit is in error, using 4-input NAND gates, provided a pre-decode AND of bits 0-3 and bits 4-7 of the syndrome is available. For the check bits, a full 8-bit decode of the syndrome is required.

Secondary Cache Tag Bus SECDED Code

The SECDED ECC code protecting the secondary cache tag bus has the following properties:

- It corrects single-bit errors.
- It detects double-bit errors.
- It detects 3- or 4-bit errors within a nibble.
- It provides 25 data bits protected by 7 check bits, yielding 7-bit syndromes.
- It provides byte XORs of the data bits as part of the XOR trees used to build the parity generators. This allows selection of byte parity out of the XOR trees that generate or check the code.
- Single-bit errors are indicated by syndromes that contain exactly three 1s. This makes it possible to decode the syndrome to find which data bit is in error with 3-input NAND gates. For the check bits, a full 7-bit decode of the syndrome is required.
- Double-bit errors are indicated by syndromes that contain an even number of 1s.
- 3-bit errors within a nibble are indicated by syndromes that contain either five 1s or seven 1s.
- 4-bit errors within a nibble are indicated by syndromes that contain either four 1s or six 1s. Because these are even numbers of 1s, 4-bit errors within a nibble look like double-bit errors.

Error Checking Operation

The processor verifies data correctness by using either the parity or the SECDED code as it passes data from the System interface to the secondary cache, or it moves data from the secondary cache to the primary caches or to the System interface.

System Interface

The processor generates correct check bits for doubleword, word, or partial-word data transmitted to the System interface. As it checks for data correctness, the processor passes data check bits from the secondary cache, directly without changing the bits, to the System interface if the interface is set to ECC mode. If the System interface is set to parity mode, the processor indicates a secondary cache ECC error by corrupting the state of the **SysCmdP** signal.

The processor does not check data received from the System interface for external updates and external writes. By setting the **SysCmd(4)** bit in the data identifier, it is possible to prevent the processor from checking read response data from the System interface.

The processor does not check addresses received from the System interface, but does generate correct check bits for addresses transmitted to the System interface.

The processor does not contain a data corrector; instead, the processor takes a cache error exception when it detects an error based on data check bits. Software, in conjunction with an off-processor data corrector, is responsible for correcting the data when SECDED code is employed.

Secondary Cache Data Bus

The 16 check bits, **SCDChk(15:0)**, for the 128-bit secondary cache data bus are organized as 8 check bits for the upper 64 bits of data, and 8 check bits for the lower 64 bits of data.

System Interface and Secondary Cache Data Bus

The 8 check bits, **SysADC(7:0)**, for the System interface address and data bus provide even-byte parity, or are generated in accordance with a SECDED code that also detects any 3- or 4-bit error in a nibble. The 8 check bits for each half of the secondary cache data bus are always generated in accordance with the SECDED code.

Secondary Cache Tag Bus

The 7 check bits, **SCTChk(6:0)**, for the secondary cache tag bus are generated in accordance with the SECDED code, which also detects any 3- or 4-bit error in a nibble.

The processor generates check bits for the tag when it is written into the secondary cache and checks the tag whenever the secondary cache is accessed.

The processor contains a corrector for the secondary cache tag; the tag corrector is not in-line for processor accesses due to primary cache misses. The processor traps when a tag error is detected on a processor access due to a primary cache miss. Software, using the processor cache management primitives, is responsible for correcting the tag. When executing the cache management primitives, the processor uses the corrected tag to generate write back addresses and cache state.

For external accesses, the tag corrector is in-line; that is, the response to external accesses is based on the corrected tag. The processor still traps on tag errors detected during external accesses to allow software to repair the contents of the cache if possible.

System Interface Command Bus

In the R4000 processor, the System interface command bus has a single parity bit, **SysCmdP**, that provides even parity over the 9 bits of this bus. The **SysCmdP** parity bit is generated when the System interface is in master state, but it is not checked when the System interface is in slave state. In the R4400 processor, input parity is reported through the **Fault*** pin.

When the System interface is set to parity mode, the processor indicates a secondary cache ECC error by corrupting the state of the **SysCmdP** signal.

SECDED ECC Matrices for Data and Tag Buses

The check matrices for data and tags, specifying the distribution of data and check bits across nibbles, are shown in Figures 16-1 and 16-4.

The data bits in Figure 16-1 correspond to **SysAD(63:0)**, **SCData(127:64)**, or **SCData(63:0)**. The check bits in Figure 16-1 correspond to **SysADC(7:0)**, **SCDChk(15:8)**, or **SCDChk(7:0)**.

The check bits in Figure 16-4, shown later in this chapter, correspond to **SCTChk(6:0)** and the data bits in Figure 16-4 correspond to **SCTag(24:0)**.

The parity check matrices shown in these two figures generate the ECC code for a fixed-width data word; they can also locate the data bit in error. In Figure 16-1, the data word length is 64 bits; in Figure 16-4, the data word length is 25 bits.

ECC Check Bits

The R4000 processor provides the following check bits: 16 check bits, **SCDChk(15:0)**, are used for the secondary cache data bus; 7 check bits, **SCTChk(6:0)**, are used for the secondary cache tag bus; 8 check bits, **SysADC(7:0)**, are used for the System interface address and data bus; a single parity bit, **SysCmdP**, is used for the System interface command bus.

In the R4400 processor, the **Fault*** pin reports data parity or any ECC errors received from the System interface during an external update or an external write. The **Fault*** pin also reports errors among the address bits received from the System interface. In each case, the full 64-bit data and 8-bit ECC are significant. This checking is not affected by the state of the disable bit [**SysCmd(4)**] in the data identifier. No exceptions are generated for these checks.

Data ECC Generation

Each of the 64 data bits and 8 check bits has a unique 8-bit SECDED ECC check code; this check code is generated by taking the even parity of the ECC check code for a selected group of data bits. As Figure 16-1 shows, bit locations are numbered from right to left in ascending order, from data bit 0 (furthest right) to data bit 63 (furthest left). For example, data bit 0, in the far right column of Figure 16-1, has an 8-bit check value of $0001\ 0011_2$ (0s are represented in this figure by periods, (.), because they are not used in the calculations).

Figure 16-1 also gives values for the 8 check bits, 7:0. For instance, the 8-bit SECDED ECC code for check bit 6 is in column 6, near the right hand edge of Figure 16-1.

Data bit 63 Nibbles Check bit 6 Data bit 0

Check Bit			43		52											70		61	
Data Bit		6666 3210	55 98	5555 7654	55 32	5544 1098	4444 7654	4444 3210	3333 9876	3333 5432	3322 1098	2222 7654	2222 3210	1111 9876	1111 5432	11 10	9876	54	3210
MSB	27	1111	11. .	11. .	1. . .	1. . .	. . . .	1111	1111	. . . .	1. . .	1. . .	1. . .	1. . .	. . . .	1. 1.	. 1. .	1. . .	1. . .
	27	1111	1. . .	1. . .	1. . .	. 1. .	. . . .	. . . .	. . . .	1111	. 1. .	. 1. .	. 1. .	. 1. .	1111	11. .	11. .	1. 1.	. 1. .
ECC	27	. . . .	1. . .	11. .	1. 1.	. . 1.	1111	1111	. . . .	. . . .	. . 1.	. . 1.	. . 1.	. . 1.	1111	1. . .	1. . .	11. .	. . 1.
Code	27	. . . .	1. 1.	. 1. .	11. .	. . . 1	1111	. . . .	1111	1111	. . . 1	. . . 1	. . . 1	. . . 1	. . . .	1. . .	11. .	1. . .	. . . 1
Bits	27	1. . .	. 1. 1	. . 11	. 1. .	. . . .	1. . .	1. . .	1. . .	1. . .	1111	1111	. . . .	1111	1. . .	11. .	. . . 1	. 1. .	. . . .
	27	. 1. .	11. .	. . 1.	. 1. 1	1111	. 1. .	. 1. .	. 1. .	. 1. .	. . . .	. . . .	1111	1111	. 1. .	. 1. .	. . 11	. 1. .	. . . .
	27	. . 1.	. 1. .	. . 11	11. .	1111	. . 1.	. . 1.	. . 1.	. . 1.	1111	. . . .	. . . .	. . . .	. . 1.	. 1. .	. . 1.	. 1. 1	1111
LSB	27	. . . 1	. 1. .	. . . 1	. 1. .	. . . .	. . . 1	. . . 1	. . . 1	. . . 1	. . . .	1111	1111	. . . .	. . . 1	. 1. 1	. . 11	11. .	1111
Number of 1s in syndrome*		3333	5511	3333	5511	3333	3333	3333	3333	3333	3333	3333	3333	3333	3333	5511	3333	5511	3333

Figure 16-1 Check Matrix for Data ECC Code

NOTE: * This row indicates the number of 1s in the generated syndrome for each data bit in error.

As an example of this process, SECDED ECC for **Data(63:0)** = 0x0000 0000 0000 0001 is generated in the steps below.

1. Find any bits in **Data(63:0)** having a value of 1.

   To determine this, the 16-bit hexadecimal value of 0x0000 0000 0000 0001 must be expanded to its 64-bit binary equivalent before locating the data bit(s) with a value of 1. In this case, the only 1-value in 0x0000 0000 0000 0001 is in column 0.

2. Find the check bits in column 0.

   They are 0001 0011_2.

3. Take even parity of check bits 0001 0011_2.

	ECC	Parity (even)
MSB (7)	0	0
(6)	0	0
(5)	0	0
(4)	1	1
(3)	0	0
(2)	0	0
(1)	1	1
LSB (0)	1	1

4. This even parity value, 0001 0011_2, is sent out over the bus as ECC check bits, **ECC(7:0)**.

The following example uses data with several 1-value bits: **Data(63:0) =** 0x0000 0000 0000 0043.

1. Expand the data to its binary equivalent in order to generate the ECC check bits.

   0x0000 0000 0000 0043 has 1s in the last byte only. The last byte binary value is: 0x43 = 0100 0011_2.

   ```
   column #   7654 3210
   0x0043 =   0100 0011₂
   ```

   Since only columns 0, 1, and 6 have 1s, they are the only columns that can generate the even parity bits.

2. Using Figure 16-1, generate even parity for the ECC check codes in columns 0, 1, and 6:

Column 0 ECC	Column 1 ECC	Column 6 ECC	Parity (even)
0	0	0	0
0	0	0	0
0	1	0	1
1	0	0	1
0	0	1	1
0	0	1	1
1	1	0	0
1	1	1	1

3. This parity value, 0011 1100_2, is sent out over the **ECC(7:0)** check bus.

Detecting Data Transmission Errors

The following procedure detects data transmission errors.

1. System A transmits a 64-bit doubleword together with 8 bits of SECDED ECC (see Figure 16-2).

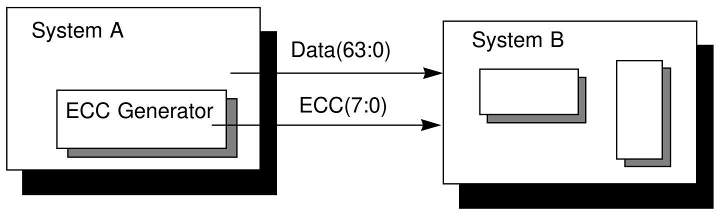

Figure 16-2 Detecting ECC Errors: Transmitting Data and ECC

2. System B receives the data doubleword, together with the byte of ECC check code.
3. To verify proper transmission of the 64-bit doubleword and 8-bit ECC check code, system B generates its own 8-bit ECC check code from the 64-bit doubleword of System A, as shown in Figure 16-3.
4. System B executes an Exclusive-OR (XOR) on the check bits of System A with its own newly-generated ECC check bits, (see Figure 16-3). The output of this XOR is called the *syndrome*.

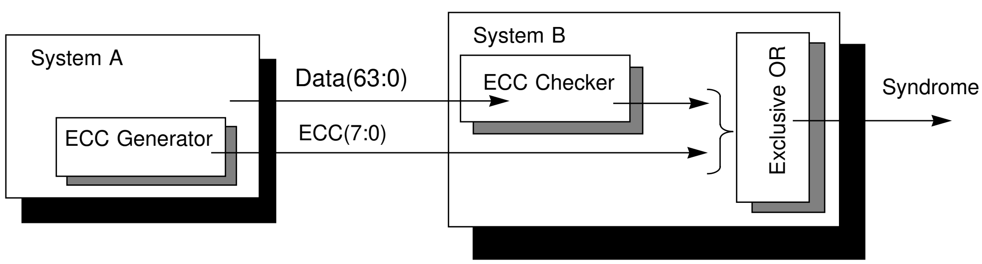

Figure 16-3 Detecting ECC Errors: Deriving the Syndrome

5. If the syndrome is $0000\ 0000_2$, the data System B received, together with the newly-generated ECC check bits from System B, are the same as the data and check bits from System A. If the syndrome is any other value than $0000\ 0000_2$, it is assumed either the received word or the received check bits are in error.

6. Using the data in Figure 16-1, it may be possible to correct either the data bit or check bit in error. Determine if the syndrome is in Figure 16-1 by counting the number on 1s in the syndrome.
   - If the syndrome contains either one, three, or five 1s, the syndrome is in Figure 16-1. Three or five 1s indicates that at least one data bit is in error. A single 1 indicates an ECC check bit is in error.
   - If the syndrome contains two 1s, a double-bit error has been detected, located in two consecutive bits of a nibble. This is not correctable.
   - If the syndrome contains four 1s, a 4-bit error has been detected, located in four consecutive bits of a nibble. This is not correctable.

   If the syndrome is identical to any of the syndromes in the Figure 16-1, the column number of that data or check bit indicates the location of the bit in error. The bit that is in error is corrected by inverting its state (a 1 is changed to 0; a 0 is changed to 1).

The following sections show how to use the check matrices in Figure 16-1 for detecting:

- single data bit error
- single data check bit error
- multiple data bit errors (2 consecutive bits in a nibble)
- multiple data bit errors (3 consecutive bits in a nibble)
- multiple data bit errors (4 consecutive bits in a nibble)

Single Data Bit ECC Error

The following procedure detects and corrects a single data bit ECC error.

1. System A transmits**:**

   Data(63:0) = 0x0000 0000 0000 0000

   and

   ECC(7:0) check code = 0000 0000_2

2. System B receives the following incorrect data:

   Data(63:0) = 0x0000 0000 0000 0001

   and

   ECC(7:0) check code = 0000 0000_2

3. System B regenerates ECC for the received data. The correct ECC check code for:

   Data(63:0) = 0x0000 0000 0000 0001

   is

   ECC(7:0) = 0001 0011_2

4. A syndrome is generated by the XOR of the System A check bits, 0000 0000_2, and the System B regenerated check bits, 0001 0011_2. The resulting syndrome is 0001 0011_2. Since the syndrome has three 1s, look for the column with three 1s in the parity check matrix table.
5. Searching the matrix (Figure 16-1) shows that the syndrome, 0001 0011_2, corresponds to data bit 0. This means the state of received data bit 0 is incorrect.
6. To correct the error, the system inverts the state of the received data bit 0 from a value of 1 to 0.

Single Check Bit ECC Error

The following procedure detects and corrects a single check bit ECC error.

1. System A transmits:

   Data(63:0) = 0x0000 0000 0000 0000

   and

   ECC(7:0) check code = $0000\ 0000_2$

2. System B receives the following incorrect check code:

   Data(63:0) = 0x0000 0000 0000 0000

   and

   ECC(7:0) check code = $0000\ 0001_2$

3. System B regenerates the ECC for the received data. The correct ECC check code for:

   Data(63:0) = 0x0000 0000 0000 0000

   is

   ECC(7:0) = $0000\ 0000_2$

4. A syndrome is generated by the XOR of the System A check bits, $0000\ 0001_2$, and the System B regenerated check bits, $0000\ 0000_2$. The resulting syndrome is $0000\ 0001_2$.

   Since the syndrome has a single 1, it is contained in the check matrix. Figure 16-1 shows that the syndrome, $0000\ 0001_2$, corresponds to check bit 0. This indicates that the state of the received check bit 0 is incorrect. To correct the error, the system inverts the state of the received check bit 0 from a value of 1 to 0.

Double Data Bit ECC Errors

The following procedure detects double data bit ECC errors.

1. System A transmits**:**

   Data(63:0) = 0x0000 0000 0000 0000

   and

   ECC(7:0) check code = 0000 0000_2.

2. System B receives the following incorrect data:

   Data(63:0) = 0x0000 0000 0000 0011

   and

   ECC(7:0) check code = 0000 0000_2

3. System B regenerates the ECC for the received data. The correct ECC check code for:

   Data(63:0) = 0x0000 0000 0000 0011

   is

   ECC(7:0) = 0011 0000_2

4. A syndrome is generated by the XOR of the System A check bits, 0000 0000_2, and the System B regenerated check bits, 0011 0000_2. The resulting syndrome is 0011 0000_2.

   The syndrome of two 1s (or an even number of 1s) indicates that a double-bit error has been detected. Double-bit errors cannot be corrected.

Three Data Bit ECC Errors

The following procedure detects three data bit errors that occur within a nibble.

1. System A transmits:

   Data(63:0) = 0x0000 0000 0000 0000

   and

   ECC(7:0) check code = 0000 0000_2

2. System B receives the following incorrect data:

   Data(63:0) = 0x0000 0000 0000 0111

   and

   ECC(7:0) check code = 0000 0000_2

3. System B regenerates the ECC for the received data. The ECC check code for:

   Data(63:0) = 0x0000 0000 0000 0111

   is

   ECC(7:0) = 0111 0011_2

4. A syndrome is generated by the XOR of the System A check bits, 0000 0000_2, and the System B regenerated check bits, 0111 0011_2. The resulting syndrome is 0111 0011_2.

   The resulting syndrome has five 1s. Since no four of the 1s are contained in check bits (7:4) or check bits (3:0), three errors have occurred within a nibble. Triple-bit errors within a nibble cannot be corrected.

Four Data Bit ECC Errors

The following procedure detects four data bit errors that occur within a nibble.

1. System A transmits:

   Data(63:0) = 0x0000 0000 0000 0000

   and

   ECC(7:0) check code = $0000\ 0000_2$

2. System B receives the following incorrect data:

   Data(63:0) = 0x0000 0000 0000 1111

   and

   ECC(7:0) check code = $0000\ 0000_2$

3. System B regenerates the ECC for the received data. The ECC check code for**:**

   Data(63:0) = 0x0000 0000 0000 1111

   is

   ECC(7:0) = $1111\ 0000_2$

4. A syndrome is generated by the XOR of the System A check bits, $0000\ 0000_2$, and the System B regenerated check bits, $1111\ 0000_2$. The resulting syndrome is $1111\ 0000_2$.

   Since the resulting syndrome has four 1s (or an even number of 1s), this error is recognized as some variation of a double-bit error. A 4-bit error within a nibble cannot be corrected.

Tag ECC Generation

The 25-bit tag ECC check matrix is similar to the 64-bit data check matrix; the main difference is the number of check bits used and the manner in which the errors are decoded. Figure 16-4 shows the check matrix for the tag bits.

Check Bit		0	12	34	56				
Data Bit		222 432	22 10	11 98	11 76	1111 5432	11 1098	7654	3210
MSB	11	. 1. .	1. . .	1. . .	. . . 1	1111	1. . .	1. . .	1. . .
	13	1. . .	. 1. .	. 1. .	. . 1.	1111	1111	. . . .	. 1. .
ECC	10	. . 1.	1. . .	. . . 1	1. . .	. . . .	1111	. 1. .	. . 1.
Code	10	. 1. .	. 1. .	. . 1.	. 1. .	1. . .	. 1. .	1111	. . . .
Bits	13	1. . .	. . . 1	1. . .	1. . .	. 1. .	. . . .	1111	1111
	11	. . 1.	. . 1.	. 1. .	. 1. .	. . 1.	. . 1.	. . 1.	1111
LSB	14	1111	11. .	11. .	11. .	. . . 1	. . . 1	. . . 1	. . . 1
Number of 1s in syndrome*		3331	3311	3311	3311	3333	3333	3333	3333

Figure 16-4 Check Matrix for the Tag ECC Code

NOTE: * This row indicates the number of 1s in the generated syndrome for each data bit in error.

Summary of ECC Operations

ECC operations are summarized in Tables 16-1 through 16-4.

Table 16-1 Error Checking and Correcting Summary for Internal Transactions

Bus	**Secondary Cache to Primary Cache**	**Primary Cache to Secondary Cache**	**Uncached Load**	**Uncached Store**
Processor or Secondary Cache Data	Checked; Trap on Error	Primary Cache parity checked; Trap on Error	From System Interface	Not Checked
Secondary Cache Data Check Bits	Checked; Trap on Error	Generated	NA	NA
Secondary Cache Tag and Check Bits	Checked; not corrected in Secondary cache; Trap on error	NA	NA	NA
System Interface Address/Command and Check Bits: Transmit	NA	NA	Generated	Generated
System Interface Address/Command and Check Bits: Receive	NA	NA	Not Checked; reported to the **Fault*** pin	NA
System Interface Data	NA	NA	Checked Trap on error†	From Processor
System Interface Data Check Bits	NA	NA	Checked; Trap on Error†	Generated

† If error level (*ERL* bit of the *Status* register) is 1, the error is reported to the **Fault*** pin.

Table 16-2 Error Checking and Correcting Summary for Internal Transactions

Bus	Store to Shared Cache Line	Cache Instruction	Secondary Cache Load from System Interface	Secondary Cache Write to System Interface
Processor or Secondary Cache Data	NA	Check on cache writeback; Trap on Error	From System Interface unchanged	Checked; Trap on Error
Secondary Cache Data Check Bits	NA	Check on cache writeback; Trap on Error	From System Interface unchanged	Checked; Trap on Error
Secondary Cache Tag and Check Bits	Checked on read part of RMW†; correct Secondary cache tag; Trap on Error	Checked; corrected Secondary cache tag*; Trap on Error	Generated	Checked; not corrected; Trap on Error
System Interface Address, Command, and Check Bits: Transmit	Generated	Generated	Generated	Generated
System Interface Address, Command, and Check Bits: Receive	NA	NA	Not Checked	NA
System Interface Data	From Processor	From Primary or Secondary Cache	Checked; Trap on Error‡	From Secondary Cache
System Interface Data Check Bits	Generated	From Primary or Secondary Cache	Checked; Trap on Error‡	From Secondary Cache (**SysCmdP** signal corrupted if System interface set to parity mode)

† Read-Modify-Write cycle
‡ If error level (*ERL* bit of the *Status* register) is 1, the error is reported to the **Fault*** pin.
* Only if the current CACHE op needs to modify and write back the tag.

Table 16-3 Error Checking and Correcting Summary for External Transactions

Bus	Read Request	Write Request	Invalidate Request	Update Request
Processor or Secondary Cache Data	NA	NA	Not Checked	Checked on read part of RMW†; Trap on Error‡
Secondary Cache Data Check Bits	NA	NA	Not Checked	Checked on read part of RMW†; Trap on Error‡; Generation on write part of RMW if written
Secondary Cache Tag and Check Bits	NA	NA	Checked on read part of RMW†; Trap on Error‡; Generation on write part of RMW if written	Checked on read part of RMW†; Trap on Error; Generation on write part of RMW if written
System Interface Address, Command and Check Bits: Transmit	Generated	NA	NA	NA
System Interface Address, Command and Check Bits: Receive	Not Checked; reported to the **Fault*** pin	Not Checked; reported to the **Fault*** pin	Not Checked; reported to the **Fault*** pin	Not Checked; reported to the **Fault*** pin
System Interface Data	From Processor	Checked; Trap on Error	Not Checked	Not Checked; reported to the **Fault*** pin
System Interface Data Check Bits	Generated	Checked; Trap on Error	Not Checked	Not Checked; reported to the **Fault*** pin

† Read-Modify-Write cycle
‡ Only the pair of doublewords accessed on the read portion of RMW is checked.

Table 16-4 Error Checking and Correcting Summary for External Transactions

Bus	Intervention Request Data Returned	Intervention Request State Returned	Snoop Request
Processor or Secondary Cache Data	Checked; Trap on Error	Not Checked	Not Checked
Secondary Cache Data Check Bits	Checked; Trap on Error	Not Checked	Not Checked
Secondary Cache Tag and Check Bits	Checked and corrected on read part of RMW†; Trap on Error; Generation on write part of RMW if written.	Checked and corrected on read part of RMW†; Trap on Error; Generation on write part of RMW if written.	Checked and corrected on read part of RMW†; Trap on Error; Generation on write part of RMW if written.
System Interface Address, Command, and Check Bits: Transmit	Generated	Generated	Generated
System Interface Address, Command, and Check Bits: Receive	Not Checked; reported to the **Fault*** pin	Not Checked; reported to the **Fault*** pin	Not Checked; reported to the **Fault*** pin
System Interface Data	From Secondary Cache	NA	NA
System Interface Data Check Bits	From Secondary Cache	NA	NA

† Read-Modify-Write cycle

16.2 R4400 Master/Checker Mode

The R4400 processor supports four Master/Checker mode configurations, which are designated by boot-mode bit settings: Complete Master, Complete Listener, System Interface Master, and Secondary Cache Master. The boot-mode bits, **SIMasterMd** (mode bit 18) and **SCMasterMd** (mode bit 42), define Master/Checker configurations. Table 16-5 lists the configurations encoded by these bits.

Table 16-5 Boot-Mode Bit Encodings of Master/Checker Modes

SCMasterMd (Bit 42)	SIMasterMd (Bit 18)	Mode
0	0	Complete Master (required for single-chip operation)
1	1	Complete Listener (paired with Complete Master)
1	0	System Interface Master (SIMaster)
0	1	Secondary Cache Master (SCMaster, paired with SIMaster)

For a non-fault tolerant system, these bits must be set to 00_2. This is the *Complete Master* mode.

In a fault tolerant system, there are two possible configurations using the Master-Listener and Cross-Coupled modes described in Table 16-5. These are referred to as *lock-step configurations*, and are described later in this section.

Connecting a System in Lock Step

By operating in lock step, a system with more than one R4400 processor can be configured to improve data integrity. In such a configuration, output signals and I/O buses used during output are connected in parallel between the processors. One processor is defined at boot time as a bus driver, and the remaining processor(s) is defined as a bus monitor. Starting with the assertion of **Reset***, all microprocessors must be synchronous, and execute identical operations on a cycle-by-cycle basis. The processor(s) designated as bus monitor compares the outputs and buses at bus-cycle boundaries, and asserts the **Fault***† signal on any mismatch.

In a lock step operation, the following R4400 signal groups are connected in parallel:

- System interface
- Secondary Cache interface (R4400SC and R4400MC only)
- Interrupt interface

The following signals are not connected in parallel:

- Initialization interface, **ModeClock**, **ModeIn**, and **Reset*** signals
- JTAG interface signals, **JTDO** and **JTMS**
- all Clock/Control interface signals except **VssP** and **VccP**

The remaining processor signals can be connected either in parallel or independently.

† **Fault*** is a non-persistent signal which is synchronous with the System interface. **Fault*** signal timing is determined by the PClock-to-SClock divisor from boot-time mode bit settings.

Master-Listener Configuration

As shown in Figure 16-5, the Master-Listener lock step configuration pairs a Complete Master (mode bits 42 and 18 = 00_2) with a Complete Listener (mode bits 42 and 18 = 11_2). In this configuration, the Complete Listener has disabled output drivers; otherwise, the two R4400 processors operate identically, both receiving the same inputs. On all output cycles, the Complete Listener compares data on the output and I/O buses with expected data, and asserts the **Fault*** signal in the event of a miscomparison.

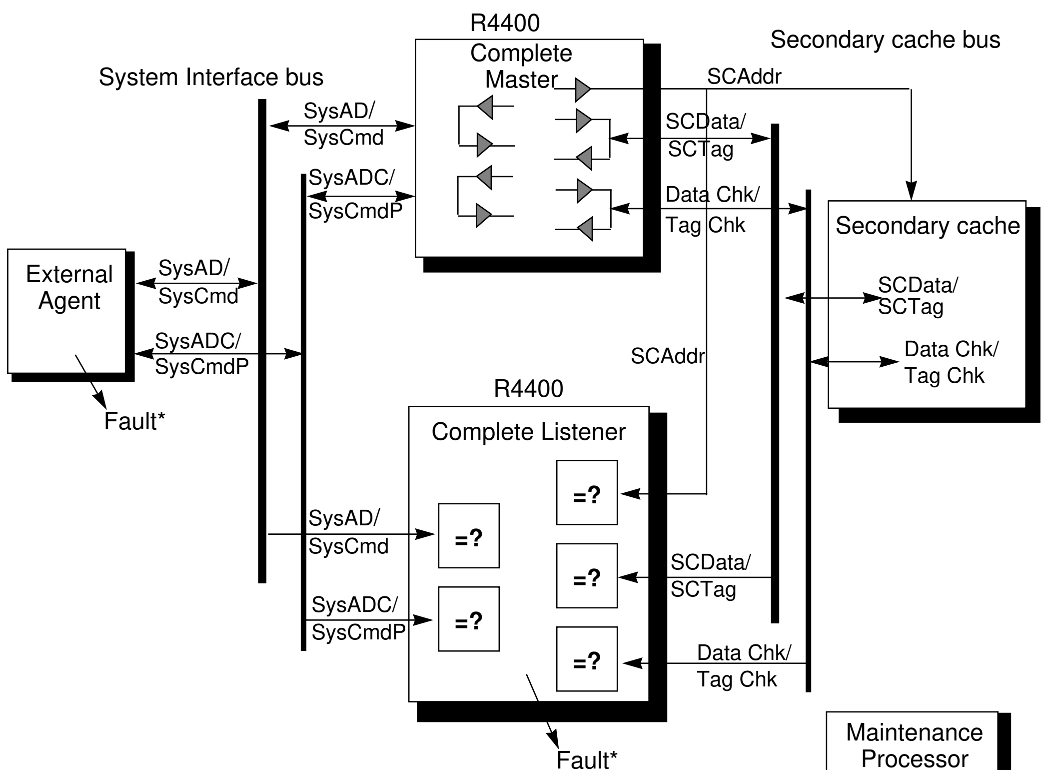

Figure 16-5 Master-Listener Configuration of Master/Checker Mode

Cross-Coupled Checking Configuration

In the Cross-Coupled Checking configuration, one of the R4400 processors drives the data bus pins and is labelled the System Interface Master (mode bits 42 and 18 = 10_2). The other R4400 processor drives the ECC or parity check pins on the same bus and is labelled the Secondary Cache Master (mode bits 42 and 18 = 01_2). This is shown in Figure 16-6.

Both processors monitor the buses and indicate a miscomparison by asserting their respective **Fault*** signals. The **Fault*** signal indicates error conditions not specifically covered by R4400 processor exceptions.†

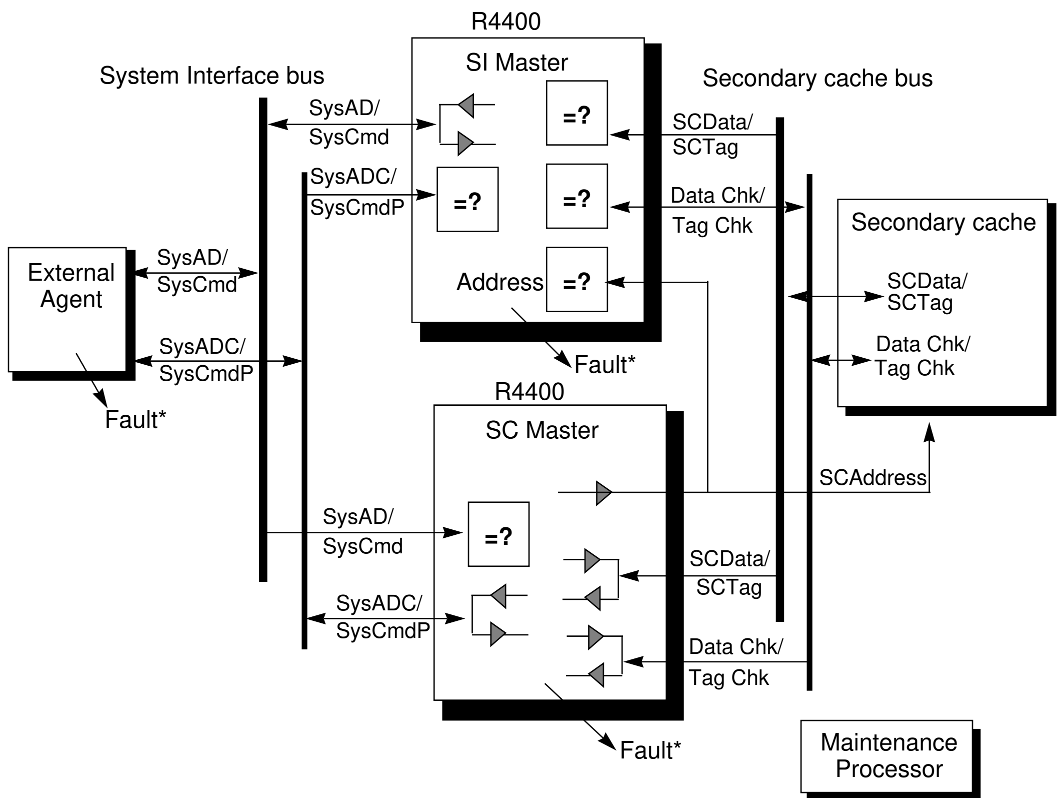

Figure 16-6 Cross-Coupled Configuration of Master/Checker Mode

† This includes such errors as an input parity error at **SysCmd**.

The signals that are connected in parallel and driven from the System Interface Master (*1* in Figure 16-6) include:

- **SysAD(63:0)**
- **SysCmd(8:0)**
- **SCAPar(2:0)**

Signals that are connected in parallel and driven from the Secondary Cache Master (*2* in Figure 16-6) include:

- **SysADC(7:0)**
- **SysCmdP**
- **ValidOut***
- **Release***
- **SCAddr(17:1)**
- **SCAddr0(W:Z)**
- **SCOE***
- **SCWr(W:Z)***
- **SCData(127:0)**
- **SCDChk(15:0)**
- **SCTag(24:0)**
- **SCTChk(6:0)**
- **SCDCS***
- **SCTCS***

It should be noted that the fault detection mechanism associated with the **Fault*** pin does not cause any exceptions; the processor continues to run normally regardless of the state of the **Fault*** signal. It is up to external logic to handle an asserted **Fault*** signal.

Fault Detection

Fault detection of an output miscomparison occurs at the end of the bus cycle (the length of the cycle is programmed at boot-mode time; see Chapter 9). When the R4400 processor is in master state, outputs at the System interface are checked at the end of every System interface cycle. At the Secondary Cache interface, outputs are checked at the end of each read or write cycle.

SCAPar(2:0) transition and check times are delayed from the rest of the Secondary Cache interface by one **PClock**. **SCAPar(2:0)** transitions occur one **PClock** after **SCAddr** transitions, or when the R4400 is changing from a read cycle to a write cycle without an address change. **SCAPar(2:0)** signals do not follow the timing of **SCWr*** signals, which are set separately through the programming of the boot-time mode bits.

The R4400 processor has an internal fault detection latency of 4 PClocks (clock cycles are described in Chapter 10), whereupon **Fault*** is synchronized with the System interface. An output fault detected and propagated through the R4400 processor internal fault logic in a prior System interface cycle is reported in the current cycle.

In Complete Master mode, output fault reporting is disabled for the Secondary Cache interface, but enabled for the following System interface signals**:** **SysCmd**, **SysCmdP**, **SysAD**, **SysADC**, **ValidOut***, and **Release***.

Reset Operation

When the R4400 processor is a Complete Listener, SIMaster, or SCMaster, an assertion of **Reset*** after the initial boot sequence is significant.

If **Reset*** is asserted a second time and subsequently deasserted, the R4400 processor changes to Forced Complete Master mode and drives all outputs.

If **Reset*** is asserted and deasserted a third time, the R4400 processor returns to its prior mode, as programmed by the boot-mode bits.

On any subsequent assertion and deassertion of **Reset***, the processor alternates between the two modes described above: the mode determined by boot-time mode bits if the Master/Checker mode is Complete Listener, SIMaster, or SCMaster, or Forced Complete Master mode.

In Forced Complete Master mode, the **Fault*** pin reports all output faults, not just faults of the System interface as are reported in Complete Master mode.

Fault History

Two internal fault history bits, **Output Fault History** and **Input Fault History**, record output faults and certain input faults reported through the **Fault*** pin. These bits are cleared with each deassertion of **Reset***.

The two fault history bits are readable when **Reset*** is asserted, and the **Fault*** pin changes from reporting live faults to indicating which fault history bit was set when **Reset*** was deasserted in the previous cycle. The **ModeIn** pin acts as selector; if **ModeIn** = 0, **Fault*** indicates the inverted state of the **Output** fault history bit. If **ModeIn** = 1, **Fault*** indicates the inverted state of the **Input** fault history bit.

The fault history bits can be reset (cleared) while the R4400 processor is running by asserting 1 to the **ModeIn** pin. Consequently, **ModeIn** must be held to 0 to maintain the status of the fault history bits. Table 16-6 presents this information in tabular form.

Table 16-6 R4400 Fault History Bit Encoding

Boot/Reset Controls	ModeIn Pin	Fault History Bits	Fault* Pin	Master/Checker Mode
VccOk just asserted (goes from 0 to 1)	Used as boot-mode bits; scan data	N/A	N/A	N/A
Reset* just deasserted (goes from 0 to 1)	N/A	Cleared to 0	N/A	N/A
Reset* deasserted in normal operation	0	Set and latched, if fault occurs	Live faults are reported	N/A
Reset* deasserted in normal operation	1	Cleared	Live faults are reported	N/A
Reset* just asserted (goes from 1 to 0)	N/A	N/A	N/A	Changed, toggling between mode bits and Forced Complete Master
Reset* just asserted (R4400 is reset)	0	Output Fault History bit is connected to the **Fault*** pin	N/A	N/A
Reset just asserted (R4400 is reset)	1	Input Fault History bit is connected to **Fault*** pin	N/A	N/A