Firefox Error-Handling Specification

Revision 0.1

Michael Nielsen (DECWSE::NIELSEN)

Workstation Systems Engineering
Digital Equipment Corporation
100 Hamilton Avenue
Palo Alto, CA 94301
415-853-6779

December 29, 1987

RESTRICTED DISTRIBUTION

Blank Page

Table of Contents

15. Firefox Error-Handling Specification

Revision History

Date	Version	Content/Changes
3 Dec 87	0.1	Preliminary release

Blank Page

15. Firefox Error-Handling Specification

The following is a discussion of hardware errors that can occur in a Firefox workstation and be detected by operating system software. It includes descriptions of the possible errors, error symptoms, recommended error logging, and required error-recovery procedures for the M-bus, L2001 dual-CVAX processor module, L2002 Q-bus adapter module, L2003 workstation I/O module, and L2007 memory module. This chapter assumes familiarity with the M-bus and all of the modules discussed herein.

The general Firefox error strategy is detection of single-bit data errors throughout the system. Specific subsystems may have more or less error detection depending on need and implementation constraints. For example, the M-bus implements single-bit error detection on both control and data signals, the L2007 memory module implements ECC on the DRAM array to achieve double-bit error detection and single-bit error correction because of higher failure rates of the DRAMs, and the L2003 workstation I/O module has no parity on internal data paths because of lack of parity support on the VLSI device controllers.

These error-handling descriptions and recommendations presume that hard failures are detected by diagnostics prior to operating system startup. If hard errors develop on the M-bus or on a critical system module during normal system operations, the system will not be able to continue execution of even the error-handling procedures. Instead, that the hard failure will be detected by diagnostics during the subsequent system restart.

The following sections discuss only single-point failures that are transient or that do not affect a critical system element. Firefox can reliably recover from single-point transient errors in cache data stores and memory arrays provided they do not affect a critical operating system data structure. It can recover from single-point transient errors on the M-bus only if the error affects peer-to-peer communication and the communicating peers can recover. Firefox does not reliably detect multipoint failures, nor is error recovery possible in such situations.

15.1. M-Bus Error Handling

All M-bus interfaces perform error checking on the value of the arbitration, command, data, and status busses and verify the proper sequencing of transactions. M-bus interfaces also perform additional types of error checking appropriate to their functions. For example, M-bus interfaces that support issuing interlocked transactions must maintain the interlock CAM, honor the interlock protocol, and detect violations of the interlock protocol.

When an M-bus error occurs, error-handling software must read the MODTYPE register of each M-bus interface and then select an error-handling routine for the appropriate interface chip type and revision. All of the M-bus interface error-handling procedures follow the same general format:

1. Record the error-logging information from all M-bus interfaces and reenable error logging. Because M-bus double errors mandate a full system restart, it is important to reenable error logging as soon as possible.
2. Use the M-bus interface chip type and revision to select an error-handling routine.
3. To enumerate the possible error states, encode the transaction type and error type. Encoding the error type also implicitly ranks the errors according to dominance, filtering out secondary errors. For example, if an MCMD parity error occurs during P2 of a transaction, the MCMD decoders might also report a spurious invalid command.
4. Verify that the M-bus interface error-logging registers indicate a known error state by comparing the encoded transaction type and error type against templates of valid error cases. The M-bus error-logging registers can represent millions of possible error states, of which

only a small percentage can actually occur. For example, because the MSTAT signals are not driven during P2 of transactions, there cannot be an MSTAT parity error for that cycle. M-bus interfaces that have inconsistent error-logging information are considered faulty and are immediately excluded from further error diagnosis.

5. Determine which module caused the error by comparing the error-logging registers of all the M-bus interfaces in the system. If a single M-bus interface detected an error, or if more than one module detected an error from the same source, log that module as the probable source of the error. The algorithms presented are predicated on at having at least three FBIC-class error-detecting and error-logging M-bus interfaces in the system. If there are less than three such M-bus interfaces, it is not possible to reliably identify the source of the error.
6. The remaining legitimate errors are then classified according to whether they affect systemwide data integrity or only peer-to-peer data integrity. If systemwide data integrity is affected, the system must be shut down and then restarted after completing the error-handling routines.
7. If the error was a peer-to-peer recoverable error, execute the recovery procedure and then return to normal system operation.

15.1.1. FBIC Error Handling

The following is a discussion of error handling for revision 1 of the FBIC; that is, the FBIC MODTYPE register has value 0101XXXX#16.

The FBIC BUSCSR, BUSCTL, BUSADR, and BUSDAT registers are normally updated with the state of various M-bus signals. When the FBIC detects an M-bus error, it asserts appropriate status bits in the BUSCSR register and freezes the state of the BUSXXX registers. Whenever an M-bus error occurs, error-handling software should immediately copy and log the FBIC MODTYPE, BUSCSR, BUSCTL, BUSADR, BUSDAT, FBICSR, RANGE, IPDVINT, CPUID, IADR1, and IADR2 registers. Error-handling software should then reenable error logging by writing, in the following order, 00000000#16 to BUSADR, 00000000#16 to BUSCTL, and FFFFFFFF#16 to BUSCSR. If the BUSCSR<FRZN> bit is not asserted in a given interface, zero the copy of that interface's BUSCTL and BUSADR registers before validating them or using them for error-cause resolution. All further references to FBIC registers in this section refer to the saved copy of the registers.

The BUSXXX registers can represent more than 2,000,000 possible transaction phases and error states. However, many of these error states represent nonexistent M-bus transaction phases. Although considering only the possible M-bus transaction phases reduces the number of representable error states to approximately 300,000 states, many of the remaining error states represent impossible error conditions for the given M-bus transaction phase. There are only 74 M-bus errors a properly functioning FBIC will detect for the various transaction phases.

15.1.1.1. Validating FBIC Error-Logging Registers

To reduce the number of error states, first validate the error-logging information by comparing the value of BUSCTL<PHASE> against the values of the BUSCSR register, the BUSCTL<SVDMCMD> field, the BUSCTL<MASTER, SLAVE> bits, and the BUSADR<31> bit.

Use the following formula to construct a bit vector that encodes the command in a longword:

```
CMD = 1 << (16*BUSADR<31> + BUSCTL<SVDMCMD>)
```

Table 15-1 shows the encodings and masks for the valid M-bus transaction types.

Table 15-1: CMD Encodings and Masks

Transaction	Encoding	Mask
MREAD	5	00000020#16
MWRITET	7	00000080#16
MREADI	9	00000200#16
MWRITE	10	00000400#16
MWRITEU	11	00000800#16
MREADU	14	00004000#16
IREAD	21	00200000#16
IREADI	25	02000000#16
IWRITE	26	04000000#16
IWRITEU	27	08000000#16
IACK	29	20000000#16

Construct a bit vector that encodes the BUSCSR<FRZN, ARB, ICMD, IDAT, MTPE, MDPE, MSPE, MCPE, ILCK, MTO, NOS, SERR>, and BUSCTL<MASTER, SLAVE> bits as shown in Table 15-2. When constructing the mask for the BUSCSR bits, add the first asserted bit found in BUSCSR in the following order: ARB, MCPE, MSPE, MDPE, ICMD, ILCK, MTO, MTPE, SERR, IDAT, NOS, FRZN. This algorithm includes only the dominant error in the mask and thus filters spurious secondary errors.

Table 15-2: ERR Masks

Log Bit	Mask	Error
BUSCSR<FRZN>	80000000#16	Error occurred and logging frozen
BUSCSR<ARB>	40000000#16	Arbitration bus error (MBRQ signals)
BUSCSR<ICMD>	20000000#16	Invalid command received during P2
BUSCSR<IDAT>	10000000#16	Invalid data transmitted
BUSCSR<MTPE>	08000000#16	Tag-store parity error during M-bus probe
BUSCSR<MDPE>	04000000#16	Address/data-bus parity error (MDAL signals)
BUSCSR<MSPE>	02000000#16	Status-bus parity error (MSTAT signals)
BUSCSR<MCPE>	01000000#16	Command-bus parity error (MCMD signals)
BUSCSR<ILCK>	00800000#16	Interlock protocol error
BUSCSR<MTO>	00400000#16	M-bus MBUSY or WAIT status timeout
BUSCSR<NOS>	00200000#16	No slave response to transaction master
BUSCSR<SERR>	00020000#16	Slave ERROR status to transaction master
BUSCTL<MASTER>	00000002#16	FBIC acting as M-bus master
BUSCTL<SLAVE>	00000001#16	FBIC acting as M-bus slave

Then index the VALID_CMDS and VALID_ERRS tables (shown in Table 15-3) with BUSCTL<PHASE> and compare the table entries against CMD and ERR. If a bit is asserted in CMD/ERR that is not asserted in the VALID_CMDS/VALID_ERRS table entry, the error-logging information is inconsistent. The M-bus interface under examination has incurred a hardware failure. After logging this module as the source of the failure, the system should be shut down, and diagnostics should be run on the faulty module. Otherwise, the error-logging information is consistent and error handling for this module should continue.

Table 15-3: VALID_CMDS/VALID_ERRS for CMD/ERR Masks

PHASE	VALID_CMD	VALID_ERR
0	2E20EEA1#16	80600003#16
1	00000001#16	80000000#16
2	FFFFFFFF#16	F5800002#16
3	2E204EA1#16	D5400003#16
4	2E204EA1#16	C7420003#16
5	20004EA1#16	CF000003#16
6	00004EA1#16	D5600003#16
7	00004221#16	C6020003#16

15.1.1.2. Determining the Source of the Error

To determine the original source of the error in the system, it is necessary to determine the dominant error in the system. Using the set of error-logging registers that have passed validation, scan through all BUSCSR registers to find the dominant error in the same fashion as the ERR mask was constructed. This may be accomplished by constructing a table with the error masks ranked in order and then stepping through the table checking to see if any BUSCSR register has the corresponding error bit asserted. Example 15-1 illustrates this procedure in Modula-2 pseudocode.

Example 15-1: Determining Dominant System Error

```
procedure Init();
begin
    (* Build table in order of decreasing significance *)
    DominantTable[0].BitPos := ARB_BitPos;
    DominantTable[1].BitPos := MCPE_BitPos;
    ...
    DominantTable[11].BitPos := FRZN_BitPos;
end Init;

procedure DominantError(): integer;
begin
    for i := 0 to high(DominantTable) do
        for j := 0 to MaxErrLog do
            if BitClear(ErrLog[j].BUSCSR, DominantTable[i].BitPos) then
                return(DominantTable[i].BitPos);
            end;
        end;
    end;
    return(0);
end DominantError;
```

Use the dominant system error code obtained in this fashion to select the corresponding error-resolution routine. The following subsections describe error resolution for each of the error cases.

15.1.1.2.1. ARB Error Resolution

The FBIC checks for loss of MBRQ signals during a transaction. For example, the M-bus master should always assert its MBRQ signal during P2. It also checks for multiple MBRQ signals when it is driving the M-bus as a bus master or bus slave.

The first step in resolving arbitration errors is counting the number of modules that believe they are acting as bus masters or slaves by examining the BUSCTL<MASTER> and BUSCTL<SLAVE> bits. If more than one module believes it is the bus master, log all of those modules as potential causes of the bus error. Similarly, if more than one module believes it is the bus slave, log all of those modules as potential causes of the error. In the case of multiple slaves, if software can determine which module should have responded, log only the spurious slaves.

If there is at most one bus master, and at most one bus slave, count the number of modules that logged ARB errors. If more than one FBIC logged ARB errors, log the master of the transaction as the probable cause of the bus error. Example 15-2 illustrates a procedure for determining the master of the transaction. If no module can be conclusively determined as the bus master, log the backplane as the probable cause of the bus error.

Example 15-2: Determining the Transaction Bus Master

```
procedure MasterSlot(): integer;
begin
    for i := 0 to MaxErrLog do
        if ErrLog[i].BUSCTL.MASTER = 1 then
            return(ErrLog[i].Slot);
        end;
    end;

    (*
     * Because of pipelining, errors during the last cycle of
     * transactions that are undetected by the master cause the
     * master to return to idle before receiving MABORT.  Pick
     * the idle slot as the most likely master.
     *)
    for i := 0 to MaxErrLog do
        if ErrLog[i].BUSCTL.PHASE = 0 then
            return(ErrLog[i].Slot);
        end;
    end;


    (*
     * Something bizarre happened.
     *)
    return(8);
end MasterSlot;
```

If only one module logged an ARB error, use that module's BUSCTL<MBRP, MBRM> error-log bits to determine whether another module spuriously asserted its MBRQ signal. If another module is indicated by the BUSCTL<MBRP, MBRM> bits, log that module as the probable cause of the bus error. If none of the BUSCTL<MBRP, MBRM> bits are asserted, log this module as the probable cause of the error. Example 15-3 illustrates a procedure for selecting a slot from the BUSCTL<MBRP, MBRM> bits, given a BUSCTL register and the slot that the register came from.

Example 15-3: Decoding BUSCTL<MBRP, MBRM> Bits

```
procedure WhichSlot(MySlot: integer; BUSCTL: BusCtlType): integer;
begin
    for i := 0 to 6 do
        if BUSCTL.MBRM[i] = 1 then
            if i < MySlot then
                return(i);
            else
                return(i+1);
            end;
        end;
    end;
    if BUSCTL.MBRP = 1 then
        return(MySlot);
    end;
    return(8);
end WhichSlot;
```

15.1.1.2.2. MCPE Error Resolution

To resolve MCMD parity errors, count the number of modules reporting MCPE errors. If more than one module reported MCPE errors, log the bus master as the probable cause of the error. Use the *MasterSlot* routine presented in the ARB error resolution section to determine the bus master. If no bus master can be conclusively selected or the module selected did not log a MCPE error, log the backplane as the probable cause of the error. If only one module reported a MCPE error, log that module as the probable cause of the error.

15.1.1.2.3. MSPE Error Resolution

To resolve MSTAT parity errors, count the number of modules reporting MSPE errors. If more than one module reported MSPE errors, log the bus slave as the probable cause of the error. Example 15-4 shows a procedure for determining the bus slave from the error logs. If no bus slave can be conclusively selected, log the backplane as the probable cause of the error. If only one module reported an MSPE error, log that module as the probable cause of the error.

Example 15-4: Determining the Transaction Bus Slave

```
procedure SlaveSlot(): integer;
begin
    for i := 0 to MaxErrLog do
        if ErrLog[i].BUSCTL.SLAVE = 1 then
            return(ErrLog[i].Slot);
        end;
    end;

    (*
     * Because of pipelining, errors during the last cycle of
     * transactions that are undetected by the slave cause the
     * slave to return to idle before receiving MABORT.  Pick
     * the idle slot as the most likely slave.
     *)
    for i := 0 to MaxErrLog do
        if ErrLog[i].BUSCTL.PHASE = 0 then
            return(ErrLog[i].Slot);
        end;
    end;


    (*
     * Something bizarre happened.
     *)
    return(8);
end SlaveSlot;
```

15.1.1.2.4. MDPE Error Resolution

To resolve MDAL parity errors, count the number of modules reporting MDPE errors. If more than one module reported MDPE errors, determine the module that first detected the error. Example 15-5 shows a procedure for determining the first module that detected a given error.

Example 15-5: Determining the First Error-Detector

```
procedure FirstDetector(ErrBitPos: integer; var Phase: integer): integer;
begin
    First := 8;
    Phase := 8;
    for i := 0 to MaxErrLog do
        if (ErrLog[i].BUSCTL.PHASE < Phase) and
          BitClear(ErrLog[i].BUSCSR, ErrBitPos) then
        Phase := ErrLog[i].BUSCTL.PHASE;
        First := ErrLog[i].Slot;
        end;
    end;
    return(First);
end FirstDetector;
```

Use the transaction phase recorded in the first detector's error log to determine which module was driving the MDAL signals at the time of the error. Example 15-6 shows a procedure for determining the slot

driving the M-bus in a given error-log phase. If no bus driver can be conclusively determined, or if the slot selected did not log an MDPE error, log the backplane as the probable cause of the error. Otherwise, log the bus driver as the probable cause of the error.

Example 15-6: Determining the Bus Driver from an Error-Log Phase

```
procedure BusErrDriver(Phase: integer): integer;
begin
    for i := 0 to MaxErrLog do
        if ErrLog[i].BUSCTL.MASTER = 1 then
            if (Phase <= 3) or
              (BitAnd(ErrLog[i].CMD, 00004ea3h) # 0) and (Phase <= 6)) then
                return(i);
        end;
        if ErrLog[i].BUSCTL.SLAVE = 1 then
            return(i);
        end;
    end;
    return(8);
end BusErrDriver;
```

If only one module logged an MDPE error, log that module as the probable cause of the error.

15.1.1.2.5. ICMD Error Resolution

To resolve ICMD errors, count the number of modules that logged ICMD errors. If more than one module logged ICMD errors, log the bus master as the probable cause of the error. Use the *MasterSlot* routine to select the bus master. If only one module logged an ICMD error, log that module as the probable cause of the error.

15.1.1.2.6. ILCK Error Resolution

To resolve ILCK errors, count the number of modules that logged ILCK errors. If more than one module logged ILCK errors, log the bus master as the probable cause of the error. Use the *MasterSlot* routine to select the bus master. If only one module logged an ILCK error, log that module as the probable cause of the error.

15.1.1.2.7. MTO Error Resolution

To resolve MTO errors, count the number of modules that logged MTO errors. If more than one module logged MTO errors, examine the bus-master BUSADR contents to determine whether the slave should have responded to the transaction. If no slave should have responded, log the bus master as the probable cause of the error. If a slave should have responded, log that bus slave as the probable cause of the error. If only one module logged an MTO error, log that module as the probable cause of the error.

15.1.1.2.8. MTPE Error Resolution

To resolve MTPE errors, log all modules reporting MTPE errors as the cause of the error.

15.1.1.2.9. SERR Error Resolution

To resolve a SERR error logged by a bus master, use either the saved address from BUSADR or the BUSCTL<MBRP, MBRM> bits together with the *WhichSlot* procedure to determine the bus slave. Log the bus slave as the cause of the error.

15.1.1.2.10. IDAT Error Resolution

To resolve IDAT errors, log all modules reporting IDAT errors as the cause of the error. Refer to module-specific error-recover procedures for any necessary internal module recovery actions.

15.1.1.2.11. NOS Error Resolution

To resolve an NOS error logged by a bus master, use the saved address from BUSADR to determine the bus slave. If a slave should have responded to the transaction address, log the bus slave as the cause of the error. Otherwise, log the bus master as the cause of the error.

15.1.1.2.12. FRZN Error Resolution

To resolve FRZN errors, count the number of modules reporting FRZN errors. If all modules reported a FRZN error and all modules have the same BUSCTL<PHASE> value, log the backplane as the probable cause of the error. If more than one module reported a FRZN error, but one of the modules has a different BUSCTL<PHASE> value, log that module as the probable cause of the error. If only one module logged a FRZN error, log that module as the probable cause of the error.

In the case of only one module logging a FRZN error, the BUSCSR<CTO, CDPE, CTPE> bits should be checked. Because these bits represent local module errors and there is no FBIC logging of the error address, module-specific error-handling procedures must be consulted.

15.1.1.2.13. Resolution Accuracy

To verify the error resolution algorithms presented here, a test system with three FBIC-based modules was constructed. With one module acting as the bus master, one module acting as the bus slave, and one module acting as a bus monitor, transactions were simulated with one-cycle-duration transient faults inserted on the MBRQ/MBUSY, MCMD/MCPAR, MSTAT/MSPAR, and MDAL/MDPAR signals. Transient faults were inserted at four points:

- At the FBIC output of the bus driver
- On the M-bus
- At the FBIC input of the bus master/slave receiver
- At the FBIC input of the bus monitor receiver

Canonical transactions with transient fault insertion representative of 6747 error cases were simulated. The resulting FBIC error logs were gathered and analyzed using the algorithms presented in this section.

Because of idiosyncrasies of the FBIC implementation, 2.9 percent of the transient fault cases did not report errors. For example, an FBIC acting as a monitor of a transaction does not check that exactly one MBRQ signal is asserted on the backplane, so it is insensitive to spurious pulses on MBRQ signals at those times.

In cases in which any FBIC in the system logged an error, the error resolution algorithms identified the correct module 94.5 percent of the time.

15.1.1.3. Determining whether System Recovery Is Possible

If the systemwide dominant error was ARB, MCPE, MSPE, MDPE, ICMD, ILCK, MTO, or MTPE, the system must be shut down and restarted because system data integrity cannot be guaranteed.

If the BUSCSR<DBL> bit is asserted in any error log, regardless of the dominant system error, the system must be shut down and restarted.

If the systemwide dominant error was SERR, IDAT, NOS, or FRZN, the system can be restarted provided the communicating peers can recover from the error. It is recommended that error-handling software keep track of the frequency of these recovered errors for each module, and if an occurance threshold is exceeded, the system be shut down until the problem is rectified. Refer to module-specific error-handling guidelines for any necessary internal module-recovery procedures.

15.1.2. FMDC Error Handling

TBD

15.2. L2001 Dual-CVAX Processor Module Error Handling

The L2001 implements two independent CVAX-based processors. Each CVAX processor has an external 64-Kbyte snoopy cache, 128 Kbytes of ROM, and an FBIC M-bus interface. The processors act as M-bus masters for system I/O space references and interrupt acknowledges. The FBICs also act as M-bus masters to service cache misses and write-throughs. The FBICs act as M-bus slaves for FBIC register access and snoopy cache shared reads and write-throughs. (The FBIC also supports M-bus access to the ROM, but software should not read the ROM of other processors via the M-bus during normal system operation.)

The external-cache tag and data stores have parity. The bus connecting a CVAX to its FBIC is also parity protected, but the ROM is not. It is recommended that the ROM contain a checksum that is verified by software before extensive use is made of the ROM contents.

Because the FBIC generates a MEMERR interrupt to its processor whenever it logs any error condition, this will not be explicitly mentioned in the following descriptions, all of which assume the L2001 has been initialized in accordance with the L2001 "Firefox Workstation Dual-CVAX Processor Functional Specification."

15.2.1. M-Bus Monitor Errors

If the L2001 FBIC detects an M-bus error while it is monitoring a transaction, it freezes its error-logging registers.

15.2.2. M-Bus Slave Errors

L2001 Dual-CVAX processors act as M-bus slaves during FBIC register access and snoopy cache operations. FBIC register access never causes errors unless a bus error occurs in the communication path to the register. Tag- and data-store parity errors can occur during snoopy cache operations.

If the FBIC detects a parity error in the tag store during M-bus memory-space transactions, it logs a BUSCSR<MTPE> error and generates an M-bus abort.

If an L2001 external cache is supplying read data to complete a shared memory read and the FBIC detects a data store parity error, the FBIC will log a BUSCSR<IDAT> error and assert the M-bus MDATINV signal while driving read data onto the M-bus.

If an L2001 external cache is receiving write-through data and the M-bus MDATINV signal is asserted, it writes the data store with invalid parity. No explicit error is generated to this processor, although subsequent references to those cache locations will cause a bus parity error.

15.2.3. M-Bus I/O Reads/Writes

If a processor references an I/O device and an M-bus abort occurs during the transaction, the FBIC will log the error and generate a bus-error machine check.

If the processor tries to reference a slot-specific address for an empty M-bus slot, the FBIC will log a BUSCSR<NOS> error and generate a bus-error machine check.

If the processor references a nonexistent region of slot-specific I/O space, an M-bus timeout will generally result, causing the FBIC to log a BUSCSR<MTO> error and generate a bus-error machine check.

If a processor references an I/O device and the device returns M-bus ERROR status, the FBIC will log a BUSCSR<SERR> error and generate a bus-error machine check.

If a processor writes an I/O device, but the FBIC detects a parity error on the CVAX data, it will assert the M-bus MDATINV signal when it drives the data onto the M-bus and log a BUSCSR<IDAT> error.

15.2.4. M-Bus Interrupt Acknowledges

Because of the passive release mechanism, all errors during interrupt acknowledges become passive releases. Consequently, higher-level, software-based failure-detection mechanisms, such as device timeouts, should always be implemented.

15.2.5. Cache References

If the FBIC detects a tag-store parity error during a cache read or write from its processor, the FBIC logs a BUSCSR<CTPE> error and generates a bus-error machine check.

If a cache reference to a dirty line misses and the victim write fails because of an M-bus abort, the FBIC will log the error and generate a bus-error machine check.

If a cache reference to a dirty line misses, and the victim write fails because of no slave response, the FBIC logs a BUSCSR<NOS> error and generates a bus-error machine check. This situation should never occur because the presence of a dirty victim implies that the cache fill occurred successfully.

If a cache reference misses and an M-bus abort occurs during the memory read, the FBIC will log the error and generate a bus-error machine check.

If a cache reference misses and the memory read does not receive a slave response, the FBIC logs a BUSCSR<NOS> error and generates a bus-error machine check.

If a cache reference misses and the memory slave specifies M-bus ERROR status, the FBIC logs a BUSCSR<SERR> error and generates a bus-error machine check.

If a cache reference misses and the memory slave specifies that the data has had a single-bit error corrected, the FBIC generates a CRD interrupt.

If a cache references misses, and the memory slave specifies an uncorrectable data error, the FBIC writes the data store with invalid parity during the cache fill.

If a cache read hits, but the data store has invalid parity and parity checking is enabled, the CVAX generates a parity-error machine check.

If a cache write hits, but the CVAX supplies bad parity, the data store is written with invalid parity. A parity error will be detected by a subsequent CVAX or FBIC cache read.

If a cache write generates a shared write-through and an M-bus abort occurs during the write-through, the FBIC will log the cause of the M-bus abort.

If a cache write-through does not receive a response from the memory module, the FBIC logs a BUSCSR<NOS> error. This situation should never occur because the presence of a cache line implies that

the cache fill occurred successfully.

If the FBIC detects a parity error during a cache write-through, it logs a BUSCSR<IDAT> error and asserts the M-bus MDATINV signal while driving the data onto the M-bus. The data store is also written with invalid parity.

15.2.6. Local I/O Reads/Writes

If a CVAX tries to reference any part of its own slot-specific I/O space other than the ROM, tag-store, and FBIC registers, a local bus timeout will result. The FBIC will log a BUSCSR<CTO> error and generate a bus-error machine check.

If a transient error occurs on the bus connecting the CVAX and FBIC during a tag-store or FBIC register read, it will cause a parity-error machine check, provided that it is a single-bit failure.

If a transient error occurs on the bus connecting the CVAX and FBIC during a tag-store or FBIC register write, it will cause a parity error. The FBIC will log a BUSCSR<CDPE> error, provided that it is a single-bit failure.

15.3. L2002 Q-Bus Adapter Module Error Handling

The L2002 allows use of Q-bus I/O options in a Firefox system. The L2002 is an M-bus slave device for access to FBIC registers, ROM, CQBIC registers, and Q-bus option registers. The L2002 is an M-bus slave device for interrupt acknowledges to the FBIC, CQBIC, and Q-bus interrupts. To maintain cache consistency, the L2002 can also act as a memory-space slave The L2002 is an M-bus memory-space master for Q-bus option DMA.

The FBIC supports the full M-bus error detection and logging protocol. The CQBIC detects and logs: Q-bus parity errors, nonexistent Q-bus memory and I/O references, no-grant timeouts, no-sack aborts, and M-bus memory errors.

There is no parity protection between the FBIC and CQBIC or on the ROM. It is recommended that the ROM contain a checksum that is verified by software before use of the ROM contents.

The following descriptions assume the L2002 has been initialized in accordance with the L2002 "Firefox Workstation Q-bus Adapter Module Functional Specification."

15.3.1. M-Bus Monitor Errors

If the L2002 FBIC detects an M-bus error, it freezes its error-logging registers and generates an IPL 17 M-bus interrupt.

15.3.2. M-Bus Slave Errors

References to the L2002 can timeout if a nonexistent address within the L2002 slot-specific region is referenced or if there is a failure of the control logic on the module.

If the CQBIC detects error conditions internal to itself or from the Q-bus, it returns ERR to the FBIC for read class transactions or generates a MEMERR to the FBIC for write class transactions. Assertion of the ERR signal causes a processor machine check with a bus-error status code, and the processor FBIC logs a BUSCSR<SERR> error. Assertion of the CQBIC MEMERR signals causes an IPL 16 M-bus interrupt. The CQBIC DSER<7, 5> bits will be asserted depending on whether a nonexistent Q-bus address was referenced or a Q-bus parity error occurred. (Because the L2002 CQBIC is always configured as the Q-bus arbiter, no-grant errors should never occur.) When the system error-handling procedures select the L2002 as the source of errors, the CQBIC DBR, DSER, MEAR, and SEAR registers should be logged. The CQBIC DSER register should then be written with 000000BD#16 to restart CQBIC error logging. Finally, the appropriate Q-bus error-recovery procedures should be executed.

If the L2002's FBIC has a line in its internal cache that had an uncorrectable error when it was read from memory, the FBIC marks the line as containing invalid data. If the FBIC supplies that data to the M-bus during a shared read, victim write, or write-through, it will assert the MDATINV signal, log an IDAT error, and generate an IPL 17 M-bus interrupt. The invalid line can be removed from the L2002 FBIC internal cache by issuing a local-miss-global-hit read transaction (reading M-bus memory through Q-bus memory space) to a memory line that does not have an uncorrectable error. Alternatively, the invalid line can be left in the L2002 FBIC until it is flushed by subsequent Q-bus DMA activity.

15.3.3. M-Bus Master Errors

If the CQBIC issues a memory read to the L2002 FBIC and the memory read incurs an M-bus abort, references nonexistent memory, or reads a memory line with an uncorrectable error, the FBIC logs the appropriate error and returns ERR to the CQBIC. This causes the CQBIC to abort the Q-bus transaction and log a slave memory error by asserting DSER<4>.

If a nonexistent memory reference caused the error, the FBIC BUSCSR<NOS> bit will be asserted. If the L2002 has issued an invalid memory address, there has been a failure of either the CQBIC mapping logic or the L2002 map store. This should be logged as the cause of the failure, the system should be shut down, and diagnostics should be run on the L2002.

After a CQBIC error occurs, the DBR, DSER, MEAR, and SEAR registers should be logged. The CQBIC DSER register should then be written with 000000BD#16 to restart CQBIC error logging, and the appropriate Q-bus error recover procedures should be executed.

If reading a memory line with an uncorrectable error caused the error, the invalid line can be removed from the L2002 FBIC internal cache by issuing a local-miss-global-hit read transaction (reading M-bus memory through Q-bus memory space) to a memory line that does not have an uncorrectable error. Alternatively, the invalid line may be left in the L2002 FBIC until it is flushed by subsequent Q-bus DMA activity.

15.3.4. Q-Bus DMA Errors

If Q-bus option DMA references an address that is not mapped, the CQBIC will let the DMA master timeout. The CQBIC does not log any errors. The Q-bus option should log a bus error and generate an interrupt.

15.4. L2003 Workstation I/O Module Error Handling

The L2003 implements a DSSI mass storage controller, an Ethernet network interface, a DZ serial interface, and miscellaneous system support functions. The L2003 is a passive slave device; it never acts as a M-bus master. The L2003 exists only in I/O space; it neither references nor responds to memory space.

Due to implementation constraints of the VLSI device controllers utilized in the L2003, there is no internal bus parity. It is recommended that the ROM contain a checksum that is verified by software before use of the ROM contents. It is also recommended that disk and network packets contain an CRC value that is generated/checked by the host processors to safeguard against undetected errors in the L2003 buffers and data paths.

The L2003 FBIC supports the full M-bus error detection and logging protocol. It never returns RETRY or ERROR status to the M-bus. The only module-level error possible when referencing the L2003 is an M-bus timeout.

The following descriptions assume the L2003 has been initialized in accordance with the L2003 "Firefox Workstation I/O Module Functional Specification."

15.4.1. M-Bus Monitor Errors

If the L2003 FBIC detects an M-bus error, it freezes its error-logging registers and generates an IPL 17 M-bus interrupt.

15.4.2. M-Bus Slave Errors

An M-bus timeout can result if a nonexistent address within the L2003 slot-specific region is referenced or if there is a failure of the control logic on the module. Device controllers should never report host-bus-related errors. That is, the LANCE network controller should never report a memory error (CSR0<MERR> bit asserted), and the SII disk controller should never report a FIFO overflow error (CSTAT<BER> bit asserted). (Note that the CSTAT<BER> bit may also be asserted for certain types of DSSI bus errors.)

If the LANCE or SII reports a host bus error, the L2003 must be reset by either a module reset or a full system reset. No error recovery of current device controller activity is possible.

Peripheral-related device controller errors are beyond the scope of this chapter.

15.5. L2007 Memory Module Error Handling

TBD

15.6. System Error Analysis and Recovery

The following system error analysis and recovery procedure is recommended whenever a processor receives a bus-error machine check, a parity-error machine check, or an MEMERR interrupt. The operating system should define a semaphore that serializes system error handling to a single processor. That is, system error handling should be treated as a systemwide critical section that is executed by the first processor to detect an error. Subsequent processors will execute the same error-handling procedures but as a degenerate case that takes no additional action. This is the easiest mechanism for dismissing the MEMERR interrupts that all processors receive when an M-bus error occurs.

When a processor receives a bus-related machine check or an MEMERR interrupt, it should first acquire the system error-handler semaphore. It should then copy the error-logging registers of all M-bus interfaces to a memory buffer. If any of the M-bus interfaces indicate an error has been logged, those M-bus interfaces should be reenabled for error logging. If possible, system activity should be minimized during this procedure to minimize the possibility of inadvertently overwriting logs of secondary errors.

If any of the M-bus interfaces indicate an error has occurred, all of the interface error logs should be written to the system error log. The processor should then validate the copy of the error log from each M-bus interface using the validation procedure for the appropriate interface type and revision. If any of the error logs fail the validation process, those modules should be indicated in the system error log as the cause of the error and the system should be shut down.

The error-cause resolution procedures should then be executed to identify the module that caused the error. The selected module or modules should then be indicated in the system error log as the failure cause. If the error belongs to the class of error specified as nonrecoverable by any of the M-bus interfaces, the system should be shut down at this point.

If the error is a BUSCSR<CTPE> or BUSCSR<CDPE> error, the system should be shut down.

If the error is a BUSCSR<CTO> or BUSCSR<NOS> error, the process that issued the reference should be terminated.

If the error is a BUSCSR<SERR> error, the error-handling procedures for the slave module should be invoked. Unless the process that issued the reference is known to have error-recovery capabilities--for example, an expansion I/O bus configuration poller--the process should be terminated.

If the error is a BUSCSR<IDAT> error and it was an I/O-space transaction, the affected I/O devices should be reinitialized.

If the error is a BUSCSR<IDAT> error and it was a memory-space transaction, the line should be flushed from all caches, the line reinitialized in the appropriate memory module via a victim write, and the affected process(es) terminated.

To flush a line from a processor cache perform the following:

- Examine the tag store to determine if the line is present.
- If the line is present, read each byte and rewrite it. If software wishes to isolate the parity error down to the byte-address level, log which bytes cause parity-error machine checks when read. Otherwise, disable parity checking during the reads by clearing the FBICSR<CDPE> bit.
- Victim the cache line by accessing memory at an address congruent to the 64-Kbyte segment containing the cache line.

To flush a line from an I/O module cache, initiate device DMA at another line of memory. For example, for the L2002 a local-miss-global-hit transaction (reading M-bus memory space through Q-bus memory space) can be used. Any secondary BUSCSR<IDAT> errors caused by these cache flushes should be dismissed.

If the error was a parity-error machine check, terminate the affected process.

Finally, release the system error-handler semaphore and resume system operation.