

CRAY C90™ Series Memory Chip Flawing

HMM-104-A

Cray Research Proprietary

Cray Research, Inc.

Record of Revision

REVISION	DESCRIPTION
----------	-------------

	June 1994. Original printing.
A	March 1995. This revision incorporates updated information for the ME-C2.3.1 offline diagnostic release.
A1	September 1995. This change packet incorporates various technical corrections.

Any shipment to a country outside of the United States requires a letter of assurance from Cray Research, Inc.
--

This document is the property of Cray Research, Inc. The use of this document is subject to specific license rights extended by Cray Research, Inc. to the owner or lessee of a Cray Research, Inc. computer system or other licensed party according to the terms and conditions of the license and for no other purpose.

Cray Research, Inc. Unpublished Proprietary Information — All Rights Reserved.

Autotasking, CF77, CRAY, CRAY-1, Cray Ada, CraySoft, CRAY Y-MP, HSX, MPP Apprentice, SSD, SUPERCLUSTER, SUPERSERVER, UniChem, UNICOS, and X-MPEA are federally registered trademarks and Because no workstation is an island, CCI, CCMT, CF90, CFT, CFT2, CFT77, ConCurrent Maintenance Tools, COS, CRAY-2, Cray Animation Theater, CRAY APP, CRAY C90, CRAY C90D, Cray C++ Compiling System, CrayDoc, CRAY EL, CRAY J90, Cray NQS, Cray/REELlibrarian, CRAY S-MP, CRAY SUPERSERVER 6400, CRAY T3D, CRAY T90, CrayTutor, CRAY X-MP, CRAY XMS, CRInform, CRI/TurboKiva, CS6400, CSIM, CVT, Delivering the power . . . , DGauss, Docview, EMDS, HEXAR, IOS, LibSci, ND Series Network Disk Array, Network Queuing Environment, Network Queuing Tools, OLNET, RQS, SEGLDR, SMARTE, SUPERLINK, System Maintenance and Remote Testing Environment, Trusted UNICOS, and UNICOS MAX are trademarks of Cray Research, Inc.

Requests for copies of Cray Research, Inc. publications should be directed to:

CRAY RESEARCH, INC.
Logistics
6251 South Prairie View Road
Chippewa Falls, WI 54729

Comments about this publication should be directed to:

CRAY RESEARCH, INC.
Service Publications and Training
890 Industrial Blvd.
Chippewa Falls, WI 54729

CRAY C90 SERIES MEMORY CHIP FLAWING

Notational Conventions	3
Spare Chip Feature Overview	4
How It Works	4
Required Hardware	5
Modules	6
Required Software	7
MWS-E Software	7
OWS-E Software	7
UNICOS Software	7
System Configuration	7
UNICOS Installation Tool	7
OWS-E Configfile	8
System Usage	9
bootsys Handling of CPU Sparing File	9
bootsys Handling of Memory Mode	9
xbootsys	10
Clearing the System	10
Dumping the Mainframe System	10
Error Reporting	11
Spare Map File	13
MWS-E, OWS-E, and MCE File Locations and Interconnections	13
MWS-E/OWS-E File Interaction	15
Memory Chip Flaw File Layout	16
Memory Size Considerations	18
Booting UNICOS	19
Dumping UNICOS	19
OWS-E Software Problems and Corrections	19
Booting	20
Loading	20

Dumping	20
Using the Spare Chip Feature	21
Single-byte Errors	21
Bursting Memory Chips	22
Using MCE with MME	22
MCE Flaw Chip Management Window	22
Using the MCE Apply and Save Buttons	24
Apply Button	24
Save Button	25
How to Flaw a Chip	26
Adding a Module Entry	28
Adding a Flaw Chip Entry	29
Manipulating Flaw Chip Information	30
Changing Module Data	31
Swapping Modules	31
Printing a List of Flawed Chips	31
Writing the Flaw Table to the Operating System	31
Applying the Flaw Table	32
Deleting a Selected Flaw Chip Entry	32
Deleting Flaw Chip Data for a Module	32
Deleting All Flaw Table Data	33

Figures



Figure 1. errpt(8) Command Error Example	12
Figure 2. MWS-E, OWS-E, and MCE File Locations and Interconnections	14
Figure 3. Configuration Word Format	16
Figure 4. MCE Flaw Chip Management Window	23
Figure 5. Window Showing Spare Chips Not Present	24

Tables

Table 1. Chip Layout	13
Table 2. Default Flaw Map with No Spares	17
Table 3. CRAY C90 Series Physical Memory Sizes	18

Notational Conventions

The following conventions are used throughout this document:

- The base of a number is decimal unless otherwise stated. All memory references are in octal.
- `Courier` type indicates directory pathnames, filenames, commands, and screen output.
- **Courier bold** type indicates menu selections.
- The `->` symbol indicates holding the MENU mouse button down and moving the mouse pointer to the next menu item.
- Buttons are shown as they appear in a window; for example, .
- Settings are shown as they appear in a window; for example, .
- Pulldown menu selections appear in the left margins when the text indicates you should choose a menu option. An example is shown at left.



Spare Chip Feature Overview

The spare memory chip hardware is available on CRAY C90 series systems that use the MEM4M, HM4M, or HDRM memory modules. The CRAY C90 spare chip hardware provides a mechanism to flaw permanent correctable errors or other persistent errors without returning the entire memory module to Central Repair. This capability results from the extra 2,048 memory chips in a 16-CPU, fully populated memory module system (10% more than the normal number of chips in a 16-CPU system).

Additional memory information is included in the *CRAY C90 Series Hardware Maintenance Manual*, publication number CMM-0502-0A0, and supplementary change packet CMM-0502-0A1. The change packet describes the CRAY C90D HDRM memory module. Refer also to the video *CRAY C90 and CRAY C90D Memory Flawing*, publication number AVT-00179, which is a companion to this memory flawing document.

The spare chip feature enables you to flaw memory chips on a memory module. Cray Research implemented this feature for three reasons: to decrease the number of memory modules returned from the field for a single memory chip replacement, to increase the availability of the system to the customer, and to reduce module movement within the mainframe.

How It Works

The basic process of flawing a memory chip consists of redirecting data in a bank to a different chip in that bank. To accomplish this, each bank has a configuration register that is loaded with a configuration code. This code indicates which chip in the half-bank is failing.

Each half-bank contains 11 memory chips, of which 10 are used. The eleventh is the spare chip. Configuration codes of 0 through 9 are valid codes that indicate which chip is failing. The data normally written to the failing chip is shifted to the next chip. For example, if chip 7 is failing, the data normally written to chip 7 is written to chip 8. The data from chip 8 is written to chip 9, and the data from chip 9 is written to the spare chip. Data is never shifted off the module.

A code of E (hexadecimal) disables the configuration register. Disabling the configuration register leaves the previous data in the register. A configuration code of F (hexadecimal) indicates that there are no failing chips in the half-bank, and no data is shifted. Codes of A through D (hexadecimal) are not used but would be the same as the default code of F. In this case, data is stored in memory chips 0 through 9.

Because a clock load is required after system power is applied, Cray Research uses the trailing edge of the Load Complete signal to trigger the hardware to load default values into the configuration registers. This prevents random values from being used in the system after a power cycle.

The I/O Master Clear signal enables, or *arms*, the configuration registers for writing user flaw information. Either the low-speed MCU cable or the maintenance channel sends the I/O Master Clear signal to the mainframe. After an I/O master clear, a write to the uppermost address of each bank loads the configuration code into the configuration registers. Writing to the configuration registers also disarms them.

Data written to the uppermost bank address of mainframe memory in the flaw format is entered into the configuration registers. The flaw format provides data integrity for the configuration code. If data integrity is compromised, the default code is written.

The flaw format of the MEM4M and HM4M modules differs from the HDRM modules. The flaw format of the MEM4M and HM4M modules consists of three copies of the configuration data in each halfword. Each halfword corresponds to a module in the bank. Two of the three 4-bit configuration codes in each halfword must be equal before the configuration code can be written.

Data is first written into the configuration registers in the write data path. Then, the hardware automatically writes to the configuration registers in the read data path logic. The transfer of the configuration code from the write logic to the read logic occurs through a serial path.

The HDRM module uses a parity scheme to provide data integrity. The memory chips on the HDRM module use bidirectional data paths. The read and write configuration registers are located in the same chip and are written at the same time. The software ensures that all banks are written.

Required Hardware

Cray Research implements the spare chip feature by adding an extra memory chip in each bank on a memory module. A memory module stores 40 data bits of each memory word in 10 memory chips. Using a spare chip for each 40 bits, or halfword, allows for 2 bad chips in each bank. However, you can use the spare chip feature for only 1 bad chip in each halfword. If 2 chips fail in the same halfword, you must replace the module.

Modules

The following three paragraphs describe the memory module types in the CRAY C90 and CRAY C90D computer systems. For more detailed information, refer to the *CRAY C90 Series Hardware Maintenance Manual*, publication number CMM-0502-0A0, and its change packet, publication number CMM-0502-0A1.

MEM4M Module

The CRAY C916 mainframe uses the MEM4M module, which provides up to 1 gigaword (Gword) of central memory and hardware for the spare chip feature.

HM4M Module

The CRAY C98, CRAY C94, CRAY C94A, and CRAY C92A mainframes use the HM4M module, which provides up to 512 Mwords of central memory and hardware and supports the spare chip feature.

HDRM Module

The CRAY C98D, CRAY C94D, and CRAY C92AD mainframes use the HDRM module, which provides up to 2 Gwords of memory and hardware and supports the spare chip feature.

Required Software

The following paragraphs describe the software required to support the spare chip feature.

MWS-E Software

MWS-E software release ME-C2.0.spare.auto or ME-C2.1 or later is required (the latest release is recommended). Refer to “Using the Spare Chip Feature” and “How to Flaw a Chip” later in this document for more information about flawing chips with the MWS-E software.

OWS-E Software

The OWS-E 7.0.5.4 release is required to support the CRAY C90 spare chip feature. Site-specific requirements for CRAY C90 systems with the spare chip feature are documented in the OWS-E installation procedure in the *OWS-E/IOS-E 7.0.5.4 Release and Installation Notes*, publication number RN-5060 7.0.5.4. The OWS-E 7.0.6 and OWS-E 8.0 releases support the spare chip feature with recent changes.

UNICOS Software

UNICOS 7.C.3 or UNICOS 8.0 is required in order to use the spare chip feature of the hardware.

System Configuration

This subsection discusses the UNICOS installation tool and the OWS-E configfile.

UNICOS Installation Tool

The UNICOS 7.C.3 (and UNICOS 8.0) installation tool has two new configuration options that relate to the use of the spare chip feature (for CRAY C90 series systems). The following items are in the Mainframe Hardware Configuration menu:

Memory halving (spare chip)	[C90 only]
Spare memory chip config file	[C90 only]

The `Memory halving` selection defines whether the system is in four-section (half) or eight-section (full) mode, and if it is in four-section mode, which half is active (upper or lower). The memory half is selected with a switch on the mainframe.

You should set the `Spare memory chip config file` selection to the full pathname of the CRAY C90 spare memory chip configuration file on the OWS-E.

These configuration options are written to the `param` file. For example:

```
128 Mwords memory, both halves;

configure C90 spares with "<file>";
```

Other option examples would be `lower half` or `upper half` memory modes. If you specify a null file name, the second line is not written to the parameter file. This alternative exists for CRAY C90 series systems that do not have the spare chip hardware.

OWS-E Configfile

The OWS-E `configfile` parameter, `CPU_XFERFILE`, defines the default pathname for the CRAY C90 memory spares map file as follows:

```
CPU_XFERFILE os/xfer/cpuspares/cpuspares.xfr
```

By default, this entry is commented out in the `configfile`. You must uncomment this entry (remove the leading `#` character). The pathname is relative to the `~cri/etc/<crayname>` directory, which is the same directory in which the `configfile` resides.

In OWS-E 7.0.5.4, the `CPU_SPARESFILE` parameter in the `configfile` is not used. In OWS-E 7.0.6 and OWS-E 8.0, this parameter is used to define the pathname of the file that contains a copy of the last valid spares map loaded by the OWS-E software.

The `M_MEMORY` `configfile` parameter defines the physical memory size. The `dumpsys`, `xdumpsys`, and `mfdump` utilities use this parameter to determine where to place the CRAY C90 memory spares map containing disarm codes. The OWS-E 7.0.6 and OWS-E 8.0 releases no longer use this parameter but determine the end of memory by using a memory address wrap technique, which `bootsys` and `sysclear` already use.

System Usage

This subsection discusses the following topics:

- `bootsys` handling of CPU sparing file and memory mode
- Commands `mfinit` and `mfstart`
- `xbootsys`
- Clearing the system
- Dumping the mainframe system
- Error reporting

bootsys Handling of CPU Sparing File

The following sequence describes `bootsys` operation in the OWS-E 7.0.5.4 release.

1. `bootsys` checks `configfile` for the `CPU_XFERFILE` parameter and sets the `cpu_spares_path`.
2. If you specify the `-S` option, the `cpu_spares_path` is replaced with the value from the command line.
3. If Step 1 or 2 does not define `cpu_spares_path`, the parameter file is checked, and if CPU sparing is declared, `cpu_spares_path` is set to the parameter file value.

In the OWS-E 7.0.6 and OWS-E 8.0 releases, `bootsys` differs slightly; it handles the spares file as stated in the online manual (`man`) page:

If you do not specify the `-S` option, the default map is specified by the UNICOS parameter file; if the parameter file does not specify a configuration map, then the file specified by the `CPU_XFERFILE` parameter in the configuration file is used.

This change to `bootsys` does not affect use of the installation tool. If the pathname set in the installation tool matches the pathname specified in the OWS-E `configfile`, the behavior of `bootsys` will be consistent after the change.

bootsys Handling of Memory Mode

After the parameter file is verified and the `cpu_spares_path` is set (as explained previously), the `cpu_mem_half` (lower/upper/both) is decoded from the parameter file if it is specified. If you specify `cpu_spares_path` in the `bootsys` command, its value is used in setting

the `-S` option of the `mfininit` and `mfinstart` commands. When the `-S` option is set, the `cpu_mem_half` value is checked and is used in setting the `-M` option of the `mfininit` and `mfinstart` commands. If you do not specify a memory mode in the parameter file, the `mfininit` and `mfinstart` commands do not pass to the `-M` option, and both commands assume full-memory mode.

xbootsys

The `xbootsys` command as released with OWS-E 7.0.5.4 does not support the spare chip feature. Systems with the spare chip feature should not use this version of `xbootsys`. With the OWS-E 7.0.6 and later releases, `xbootsys` is modified to incorporate the spare chip feature. As explained previously, configuration parameters specified in the `param` file and OWS-E `configfile` determine use of this feature.

Clearing the System

With the OWS-E 7.0.5.4 release, the `sysclear` feature is upgraded to handle CRAY C90 memory sparing. The OWS-E `sysclear` command supports a command-line option (`-S`) to specify the spares map file pathname, as in `bootsys`. The pathname on the command line overrides the pathname in the `configfile`. The `sysclear` command uses the CSL parameter file to obtain the memory mode. The `sysclear` command always uses the spare map pathname specified in the `configfile`, rather than the one specified in the CSL parameter file.

When installing the OWS-E 7.0.5.4 release, you must run the `~cri/install/newbin.sh` script to install the `sysclear` binary files that support the spare chip feature. Refer to “Site-specific Considerations” in the “Installation Instructions” section of the *OWS-E/IOS-E 7.0.5.4 Release and Installation Notes*, publication RN-5060 7.0.5.4, for more information.

With the OWS-E 7.0.6 and OWS-E 8.0 releases, the standard `sysclear` binary files support the CRAY C90 series spare chip feature.

Dumping the Mainframe System

When dumping the system, it is necessary to write disarm codes to the configuration registers so that the hardware retains the map information with which the system was booted. Both `dumpsys` and `xdumpsys` provide this functionality for all machines, whether or not they use the spare chip feature.

For C916 mainframe types, the `dumpsys`, `xdumpsys`, and `mfdump` dump programs released with OWS-E 7.0.5.4 use an incorrect disarm code, which can result in the inability to take a mainframe dump if memory chips are being spared (refer to SPR 72163). A corrected version of `dumpsys` for use with the OWS-E 7.0.5.4 release is provided on the Customer Service central computer system, `hydra`, through the `getfix` utility by using the `-isfn 220 -r async` options. No corrected version of `xdumpsys` or `mfdump` is available for the OWS-E 7.0.5.4 release. The correction is already applied to the mainframe dumping programs released with OWS-E 7.0.6 and OWS-E 8.0.

Error Reporting

The spare chip flaw map is written to the error log after a UNICOS system boot. No option on the `errprt` command exists to dump this record, but a raw dump of records (`errprt -r`) just after the system boot record produces a recognizable dump of the flaw map record. The raw dump reveals the specific flaw map used to boot the hardware.

Figure 1 shows a memory error example from the `errpt(8)` command.

```

MEMORY ERROR(S)
Count: 1
Initial type: Correctable Final type: Correctable

Software Information:
Current command: memerr locate Current user id: 1800
Swap address: 00000407060 Swap Size: 00000011700
Text address: 00000000000 Text Size: 00000000000
Cpu mask of other processors with this memory access: 00000000
single-tasked: Addressing mode: C90 mode
I Base address:00000000000 Limit: 01777776000
D Base address:00000000000 Limit: 01777776000

Hardware Information:
Machine Serial Number: 4001 CPU: 14 P address: 0005527402d
Subtype: Single-byte error Mode: Port D Instr. Fetch Cpu: 16
Syndrome: 070257 CS: 0 Bank: 0400 Failing bit: __, 62, 61, 60 0xFF
Chip addr: 00000400 Failing module: 1
Data Read: 0621553106054533267556 Failing Address: 000000366400
xp->word:00: 0001325700540000003604 xp->word:08: 1777777777776000370400
xp->word:01: 0000000000000000000000 xp->word:09: 0621553106054533267556
xp->word:02: 0000037777740000370400 xp->word:10: 00000000000000000000400
xp->word:03: 000000000000000000003604 xp->word:11: 0000000000000000004000
xp->word:04: 0000037777740000002000 xp->word:12: 000000000000177776000
xp->word:05: 176665000064000000200 xp->word:13: 0000000000000000366400
xp->word:06: 0004000000000010000000 xp->word:14: 0000000000000000366400
xp->word:07: 0070013510000000000000 xp->word:15: 0000000000000000000000
    
```

Figure 1. `errpt(8)` Command Error Example

The `0xFF` field indicates that both configuration codes (for modules `N` and `N+1`) for this bank are `F` (which would be `0x00000FFF00000FFF` in the memory chip flaw map).

The chip reported by the `errpt` command is not necessarily the physical chip (if a chip flaw is already applied to this bank); UNICOS software tracks the flaw map used but reports on information from the exchange package. If chip 1 is flawed, physical chip 2 becomes logical chip 1, physical chip 3 becomes logical chip 2, and so on, up to the spare chip.

Table 1 shows the flawed chip layout.

Table 1. Chip Layout

Physical Chip before Flawing	Physical Chip after Flawing	Bits	Bits
0	0	0 – 3	32 – 35
1	2	4 – 7	36 – 39
2	3	8 – 11	40 – 43
3	4	12 – 15	44 – 47
4	5	16 – 19	48 – 51
5	6	20 – 23	52 – 55
6	7	24 – 27	56 – 59
7	8	28 – 31	60 – 63
8	9	64 – 67	72 – 75
9	Spare	68 – 71	76 – 79
Spare			

You can use the bank and chip information from the memory error report to flaw a specific memory chip. Refer to “Using the Spare Chip Feature” later in this document for more information.

Refer to the video companion to this document called *CRAY C90 and CRAY C90D Memory Flawing*, publication number AVT-00179, for more information and sample procedures demonstrating chip flawing.

Spare Map File

This subsection discusses the following topics:

- MWS-E, OWS-E, and MCE file locations and interconnections
- MWS-E/OWS-E file interaction
- Memory chip flaw file layout
- Memory size considerations
- Booting UNICOS

MWS-E, OWS-E, and MCE File Locations and Interconnections

Figure 2 shows the MWS-E, OWS-E, and mainframe configuration environment (MCE) file locations and interconnections relevant to the spare chip feature. Refer to the figure while reading the following information.

An ASCII spares map file is written to the cross-mounted file on the OWS-E called `os/xfer/cpuspares/cpuspares.xfr`. UNICOS uses this ASCII file to write the flaw map to memory at deadstart time. This file is also used by xelog. Refer to Figure 2 for an illustration of MWS-E, OWS-E, and MCE file locations and interconnections, as well as pathnames.

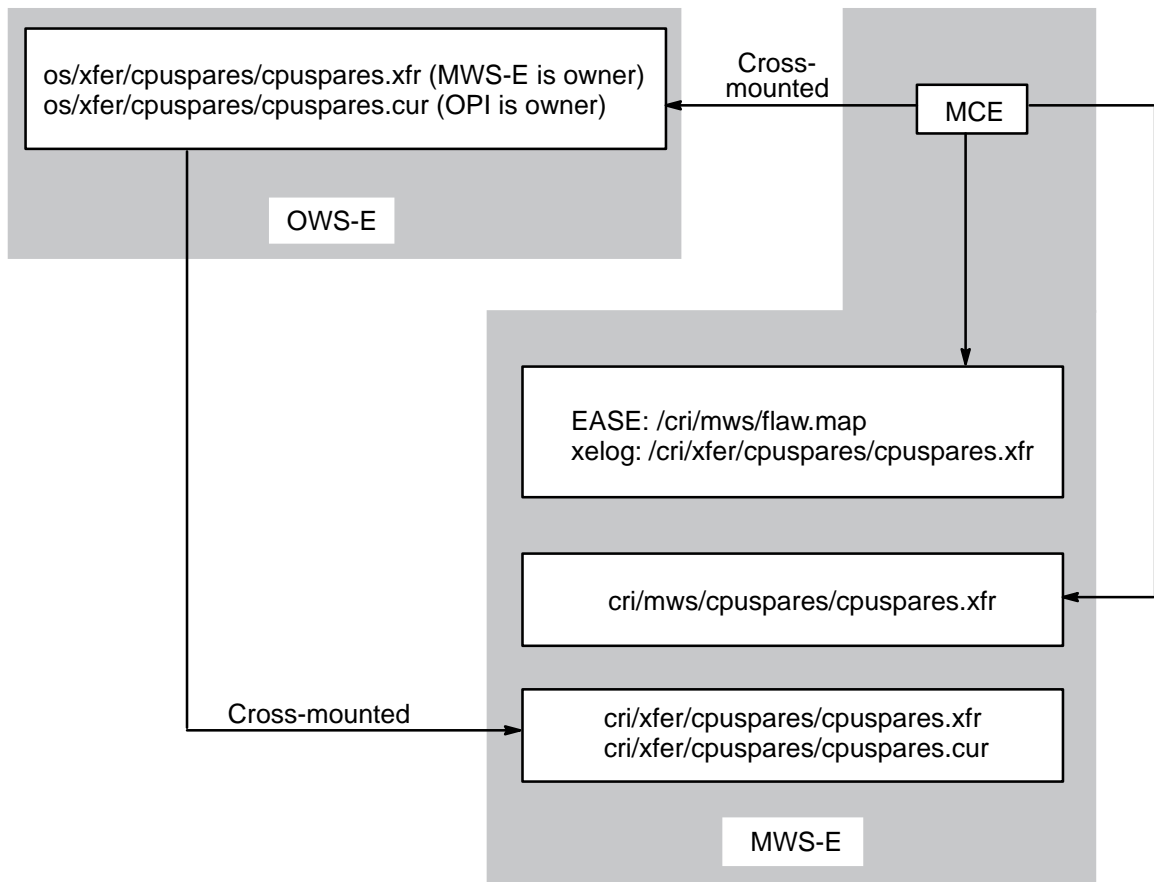


Figure 2. MWS-E, OWS-E, and MCE File Locations and Interconnections

If you suspect that the file on the OWS-E is corrupted, you can compare it to the second ASCII file written on the MWS-E.

A second ASCII file is written on the MWS-E in the file `cri/mws/cpuspares/cpuspares.xfr`. This file is a backup to the one written to the OWS-E.

A binary file is written on the MWS-E for EASE to use, which enables EASE to call out the correct memory chip location. The EASE file is located at `/cri/mws/flaw.map`.

The files shown on the OWS-E (`cpuspare/cpuspare.xfr` and `cpuspare/cpuspare.cur`) physically reside on the OWS-E.

The files `cri/xfer/cpuspare/cpuspare.xfr` and `cri/xfer/cpuspare/cpuspare.cur` are logical copies cross-mounted from the files on the OWS-E. This feature allows the field engineer to verify the contents without having to log on to the OWS-E.

MWS-E/OWS-E File Interaction

The spare chip map is generated on the MWS-E by site engineers. This information is available to OWS-E users in the following manner.

The MWS-E NFS automounts a transfer directory, which resides on the OWS-E. On the MWS-E, this directory is known as `/cri/xfer`. On the OWS-E, this directory is known as `/home/<owsname>/mws/xfer`.

The OWS-E exports this directory through the following entry in its `/etc/exports` file:

```
/home/<owsname>/mws/xfer -access=<mwsname>
```

The OWS-E 7.0.5.4 installation and upgrade scripts set up this entry. The `<mwsname>` is assumed to be of the form `"mws<serial#>"` (for example, `mws4000`). The `<owsname>` is the host name of the OWS-E.

Within this directory, at least one subdirectory holds spares maps. For CRAY C90 memory chip sparing, there is a subdirectory named `cpuspare`. This directory contains the ASCII spares file transferred from the MWS-E. This file is named `cpuspare.xfr`.

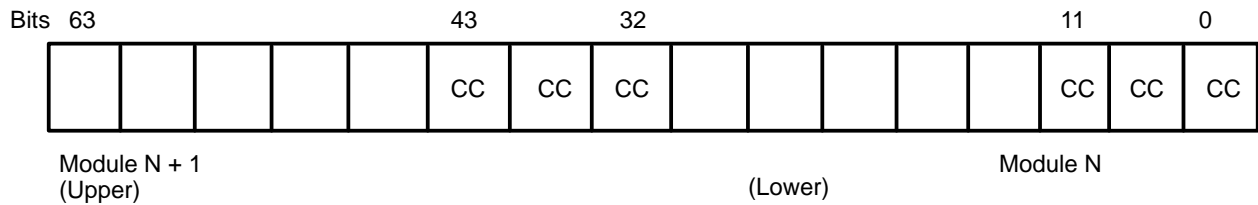
A symbolic link (`ln -s`) is created between `~cri/etc/<crayname>/os/xfer` on the OWS-E and `~mws/xfer` on the MWS-E.

The `~cri` pathname is the one that the OWS-E and UNICOS software use. Therefore, the full pathname for the spares file transferred from the MWS-E is as follows:

```
/home/<owsname>/cri/etc/<crayname>/os/xfer/cpuspares/cpuspares.xfr
```

Memory Chip Flaw File Layout

The file consists of a three-word header, 02000 octal words of spare chip information, and a list of 020 (octal) module serial/revision labels at the end. Figure 3 shows the configuration word format.



The spare chip hexadecimal configuration codes (CCs) are as follows:

- 0 – 9 Deselect chip in row
- A – D Unused – same as default
- E Disable configuration
- F Default configuration

Figure 3. Configuration Word Format

For example, if you flawed bits 4 through 7 (chip 1 on lower module) on bank 1234, the flaw map would include the following line:

```
1234 0x00000FFF00000111
```

Table 2 shows the default flaw map with no spares.

Table 2. Default Flaw Map with No Spares

Contents	Description
Mc904000	Header Mc90XXXX (XXXX is S/N)
93/08/05 13:55:00 CST	Time stamp
2146	Checksum
0000 0x00000FFF00000FFF	02000 configuration words
0001 0x00000FFF00000FFF	
0002 0x00000FFF00000FFF	
0003 0x00000FFF00000FFF	
0004 0x00000FFF00000FFF	
0005 0x00000FFF00000FFF	
0006 0x00000FFF00000FFF	
0007 0x00000FFF00000FFF	
	Lines deleted
1774 0x00000FFF00000FFF	
1775 0x00000FFF00000FFF	
1776 0x00000FFF00000FFF	
1777 0x00000FFF00000FFF	
00 00000 00000	020 module entries
01 00001 00000	
	Lines deleted
16 00016 00000	
17 00017 00000	

Memory Size Considerations

For all models of CRAY C90 systems, except the CRAY C94A system, the physical memory can be fully populated or half populated, as shown in Table 3:

Table 3. CRAY C90 Series Physical Memory Sizes

Model	Serial Number	Half Populated	Fully Populated
C916	4000	512 Mwords	1,024 Mwords
C98	4800	256 Mwords	512 Mwords
C94	4600	128 Mwords	256 Mwords
C94A	4400	128 Mwords	N/A
C92A	4200	64 Mwords	128 Mwords

For sparing purposes, the CRAY C94A system is always configured as 128 Mwords, lower half.

To simplify manipulation of the flaw map, the spare chip map is fixed at 1,024 words (02000 octal). However, when the map is written to memory, only the last word in each memory bank is significant. Because each memory bank equals 1 Mword, the memory size (in Mwords) determines how many words of sparing information the hardware will use.

The MWS-E software replicates spare chip information in the 1024-word map until the map is filled. For example, for a 256-Mword CRAY C94 system, there are four replications of spare chip information, only the last of which is used by the hardware (the last 256 words of physical memory).

For half-populated machines, the 1024-word spare chip map is interleaved so that for every 8 words of map there are 4 words of chip sparing codes and 4 words of filler. If the lower half of memory is populated, the first 4 words contain sparing codes. Likewise, if the upper half of memory is populated, the second 4 words contain sparing codes. When the map is written to memory, the software compresses the filler information and only 512 words of sparing codes are written to the end of physical memory.

Memory size can be artificially reduced by specifying a lower amount in the CSL parameter file. However, this does not affect the load location or the size of the map that must be loaded to use the spare chip feature properly.

Booting UNICOS

When an I/O Master Clear signal is issued, the latches are armed on the memory module. At this point, the flaw map may be written to the upper bank addresses in memory with the `mfchkye`, `mfboot`, `mfsysdmp`, and `clrc90` commands.

When you deadstart UNICOS, it accesses the cross-mounted file `os/xfer/cpuspares/cpuspares.xfr`, which is created and updated automatically by MCE and stored on the OWS-E.

When UNICOS boots, the spares file is also sent with the UNICOS kernel and the CSL parameter file to central memory. The spare chip information (all 02000 octal words) is written to the error log after every UNICOS restart. UNICOS then writes a second time to the same locations. The second write operation is necessary because sparing may change which chips are used in the banks, which makes it impossible to determine what data was written to memory during the first write operation.

Dumping UNICOS

If UNICOS crashes and a dump of UNICOS is taken, the software writes the disarm code to the upper 02000 (octal) locations of mainframe memory. This is necessary because a UNICOS dump uses the I/O Master Clear signal, which arms the configuration registers. Writing the disarm code preserves the previously written flaw map.

OWS-E Software Problems and Corrections

The official revision or update incorporating corrections is OWS-E 8.0.2 (contact your local service representative for ordering information). The supported base releases affected by the problems discussed in the following paragraphs are OWS-E 7.0.6 and OWS-E 8.0.1.2.

Cray Research analysts can obtain copies of fixes on hydra through the `getfix` utility by using the `-isfn 220 -r async` options. Customers can obtain copies of fixes by contacting their local service representative or by filing a Request for Technical Assistance (RTA) in the CRInform program.

Booting

A problem exists in the OWS-E 7.0.5.4 release, for which corrective code is not available. When booting with `xbootsys(8)` (used by OPI, the graphical operator interface utility), the spare chip hardware remains *armed* following the boot procedure. Any subsequent write to the spare chip area of memory can cause the system to crash with uncorrectable memory errors in all sections. This has occurred with faulty CPU hardware and with a user program trying to write to upper memory. Mainframe memory write operations from the operator or maintenance workstations can also affect the spare chip flaw map.

Corrective code was added to `xbootsys` to reload the CRAY C90 memory spare chip flaw map when UNICOS is loaded. This is essentially the same action that the `bootsys(8)` command takes when it invokes the `mfstart(8)` command. Reloading the flaw map disarms the memory area associated with the spare chip hardware and provides a memory-resident record of the map as it was last written.

Loading

A problem exists in the OWS-E 7.0.5.4 release, for which corrective code is not available. In downsized CRAY C90 systems (CRAY C98, CRAY C94, CRAY C94A, and CRAY C92A mainframes), the manner in which the spare chip map is loaded can result in uninitialized memory in the upper banks. Inadvertently reading this memory [by using the OWS-E `peek(8)` command, for instance] may cause the system to crash with an uncorrectable memory error interrupt.

The manner in which the flaw map is written was changed to ensure that all of memory is initialized. The entire map is written once to set the spare chip configuration, and after that, the upper banks of memory are rewritten with the contents of the flaw map to initialize that memory and to provide the memory-resident image.

Dumping

This problem does not exist in the OWS-E 7.0.5.4 release. When you dump the system with the `dumpsys(8)` or `xdumpsys(8)` command, the spare chip hardware remains armed. In some cases, the dumping procedure overwrites parts of the spare chip area of memory, and the dump fails because of I/O errors trying to read memory from the IOS.

When you dump the system, the spare chip area of memory must be disarmed. In the OWS-E 7.0.6 and OWS-E 8.0.1.2 releases, the code to write the disarm codes was changed to use the 32-bit address 0xFFFFFC00, rather than depending on memory size configuration parameters. At this upper memory address, the SunOS kernel converts well-formed I/O requests to block mode before passing them off to the `fy` driver. The `fy` driver had not detected this and wrote them to the truncated address.

Code was added to the `fy` driver to convert these requests back to Cray word mode. These fixes were thoroughly tested at the OWS-E 7.0.6 release level by using the binary files provided. The upcoming OWS-E 8.0.2 release includes corrective code. Corrected binary files are also available for sites running the OWS-E 8.0.1.2 release.

Using the Spare Chip Feature

The following guidelines will help you determine whether you should use the spare chip feature.

Single-byte Errors

A single-byte error (SBE) means that from 1 to 4 bits are failing on the same memory chip.

During a 24-hour period, a memory chip that exhibits a consistent or increasing failure rate is a possible candidate for flawing. Monitor the frequency of failures over a 7-day period; at least 250 hits per day over a 7-day period should be experienced before you consider module chip flawing. However, use the spare chip feature before system performance is compromised.

NOTE: All SBEs should be logged as they occur for individual modules. If a module is returned to Central Repair for a double-bit error or meets the criteria to be returned for flaws, this log and the flaw map should accompany the module. The log enables Central Repair to change chips that do not currently meet the criteria for flawing but which may fail more seriously in the future.

Bursting Memory Chips

A burst is made up of multiple SBEs logged rapidly in a short time. Bursts are not counted in specific numbers; for example, *more than* 500 hits in one second would be considered a burst.

During a 24-hour period, a memory chip that experiences one burst is not a candidate for flawing. If the chip experiences more than three bursts during a 24-hour period, it is a possible candidate for flawing.

During a 7-day period, a memory chip that experiences one burst is not a candidate for flawing. If the burst repeats on more than 3 days during a 7-day period, the chip may be considered for flawing.

Using MCE with MME

The following information applies when you are attempting to start MCE from MME; for information about starting MCE as a stand-alone program, refer to the *CRAY C90 Series Mainframe Offline Diagnostic Manual*, publication number CDM-0505-0D0.

NOTE: If you attempt to run MCE in offline mode, UNICOS will crash if the maintenance channel is enabled. If you attempt to apply a configuration with CPUs in 256K mode to use concurrent mode, UNICOS will crash.

When running in offline mode, MME automatically starts the MCE server. If you have created a `.default` file, the MCE server uses that file and the spares map you last saved with that file. If you do not have a `.default` file, a popup window appears that asks you if you want to start the MCE interface. You must then select a configuration from the main MCE pane and save it. This action directs MME to the correct configuration and also applies the selected flaw map to mainframe memory.

MCE Flaw Chip Management Window

MCE enables you to create a table of flawed chips on the mainframe and spare memory modules if the current configuration contains modules that have spare chips (MEM4M, HM4M, and HDRM modules). When you save an MCE configuration, the OS uses the data to specify which spare chips are used at boot time. The flaw chip table also identifies the flawed chips when the modules are repaired.



To access the MCE - Flaw Chip Management Window, choose **View -> Flaw Chip**, as shown at left, in the MCE base window. If memory modules containing spare chips have been specified as the module type by the ModType parameter (MEM4M, HDRM, or HM4M) in the MCE - Configuration window, the window shown in Figure 4 appears, indicating that flaw chip management is available. If memory modules without spare chips have been specified (MEM1M), the window shown in Figure 5 appears, indicating that flaw chip management is not available.

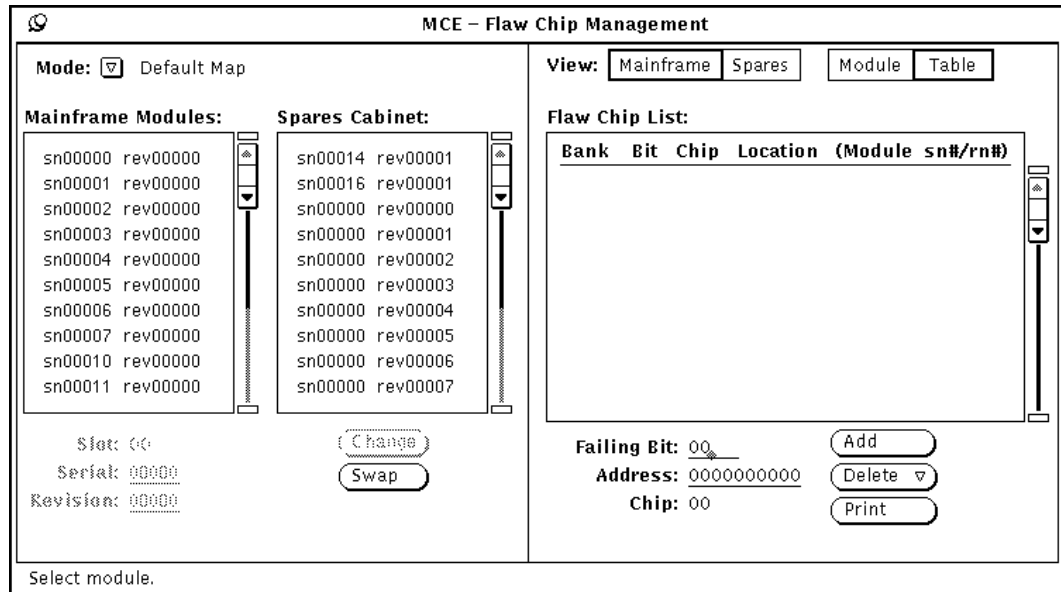


Figure 4. MCE Flaw Chip Management Window

Use the Mode: [v] button in the MCE-Flaw Management window to select the flaw map (refer also to “MCE Flaw Chip Management Window” later in this document). In the flaw management window you can select the Default, Disable, Current, All Spares, or Current + All Spares map. Initially, MCE writes the selected flaw map to memory. The flaw map is then maintained in memory, which requires MME to write the Disable map to memory each time MME initiates an I/O master clear. If you power cycle the mainframe during your maintenance activity, you need to open MCE and apply your current configuration again.

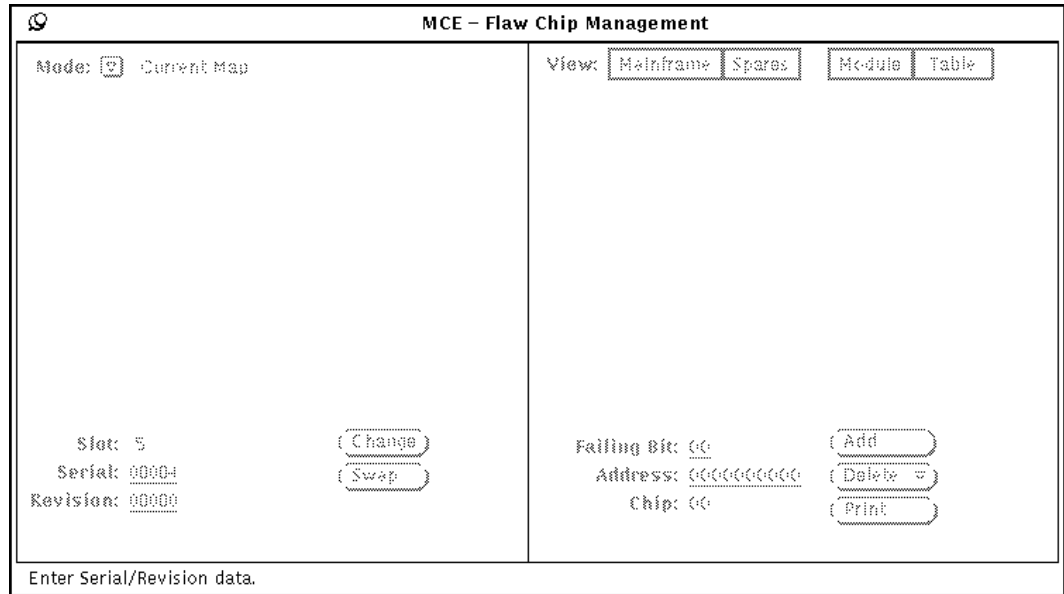


Figure 5. Window Showing Spare Chips Not Present

The MCE - Flaw Chip Management window provides several options for viewing the flaw information you have entered. You can look at the information entered for just the mainframe modules (**Mainframe**) or just the spare modules (**Spares**). You can look at a table of the flaws for all mainframe or spare modules (**Table**) or look at the flaws for just one module (**Module**).

Using the MCE Apply and Save Buttons

The **Apply** and the **Save** buttons reside in the MCE base window.

Apply Button



Choosing **Apply** -> **Write EASE file** configures the hardware with the flaw you have added and writes an EASE file (the default option as shown at left). The following events occur when you click on the **Apply** button:

- The current system configuration data is sent to MME and LME.
- An I/O master clear is performed enabling spare-chip latches to be written.
- The maintenance channel is initialized.

- Spare-chip data is written to memory and disarms the configuration registers.

You can also configure the hardware without writing an EASE file by choosing **Apply -> Do not write EASE file**.

Save Button



To save configuration and flaw data, choose **Save -> Write OS and EASE files** (the default option as shown at left). When you click on this button, the following events occur:

- The current configuration is saved, including spare chip data, under the name specified in the `Save File:` field.
- An ASCII file of spare-chip data is written on the OWS-E for the operating system.
- An ASCII file of spare-chip data is written on the MWS-E for xelog.
- A binary file of spare-chip data is written on the MWS-E for the Error Acquisition SoftwarE program (EASE).

You can also save the configuration and flaw data without writing operating system and EASE files by choosing **Save -> Do not write OS and EASE files**.

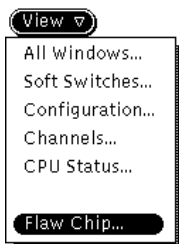
How to Flaw a Chip

To flaw a chip, follow this procedure.

CAUTION
Never run MCE in offline mode when the operating system is running in the mainframe. Offline mode performs functions that will crash the operating system if the maintenance channel is enabled.

1. MCE can run in one of two modes: concurrent or offline. Chip flawing can be performed in either mode, with the following restrictions:
 - a. If UNICOS is running, MCE must be started in concurrent mode.
 - b. If UNICOS is not running, MCE can be started and run in offline mode.

To change MCE mode, click on one of the mode buttons in the MCE base window.



2. Choose **View -> Flaw Chip** in the MCE base window, as shown at left.
3. Select a mode from the Mode: button in the flaw chip management window. The following modes are available:

<u>Mode Setting</u>	<u>Description</u>
Default Map	Do not use spare chips. When this mode is applied, MCE writes the memory module reconfiguration registers with the default code of F and closes the latches.

<u>Mode Setting</u>	<u>Description</u>
Disable Map	Disable the reconfiguration registers. When this mode is applied, MCE writes the disable code of E to the memory module reconfiguration registers, which closes the latches and preserves the previously written flaw information.
Current Map	Use the flawed-chip data the user has entered. When this mode is applied, MCE writes the memory module reconfiguration registers with the user flaw data and closes the latches.
All Spares Map	Use all spare chips. When this mode is applied, MCE writes the memory module reconfiguration registers with randomly selected flaw data and closes the latches.
Current + All Spares Map	Overlay the current map on the all spares map. When this mode is applied, MCE writes the memory module reconfiguration registers with randomly selected flaw data, except where a flaw already exists, and closes the latches.

4. If you wish to add module data, proceed with the following subsection called “Adding a Module Entry.” If you wish to add flaw chip data, skip to “Adding a Flaw Chip Entry.”

Adding a Module Entry

To add a module entry to the Mainframe Modules or Spares Cabinet scroll boxes, perform the following procedure:

1. Click on the appropriate module slot in the Mainframe Modules or Spares Cabinet scroll box. The Mem Slot field displays the selected module slot number.

NOTE: Empty slots in the scroll boxes are indicated by an sn# rev00000 notation. For example, sn00001 rev00000 indicates that slot 1 is empty.

2. Enter the module serial number in the Serial field and press the return key.
3. Enter the module revision number in the Revision field and press the return key.
4. Click on to add the module to the Mainframe Modules or Spares Cabinet scroll box.
5. Continue with Steps 1 through 4 until all module slots have been configured to match your system.
6. When all serial numbers/revision numbers are entered, click on the or button in the MCE base window. For descriptions of the options available with these buttons, refer to “Using the MCE Apply and Save Buttons” earlier in this document.

Adding a Flaw Chip Entry

Information used in this procedure comes from error logger software.

NOTE: The error logger software (errpt, xelog, or EASE) is always running, so when a flawed chip is applied, the error logger software is immediately notified. However, the hardware is not updated until the next time the OS is deadstarted. Therefore, the error logger software reports an error for the next chip up (the same bit) until the OS is rebooted.

To add a flaw chip entry to the `Flaw Chip List` scroll box, perform the following procedure:

1. Enter the failing bit number for the chip in the `Failing Bit` field and press the return key.

NOTE: Placing the cursor at the end of the line and pressing the escape key clears the entry line.

NOTE: Because CRAY C90 series systems store 4 bits in each memory device, only one of the 4 bits in a failing device can be added to the flaw table. It does not matter which of the 4 bits is used; only 1 bit per device is entered.

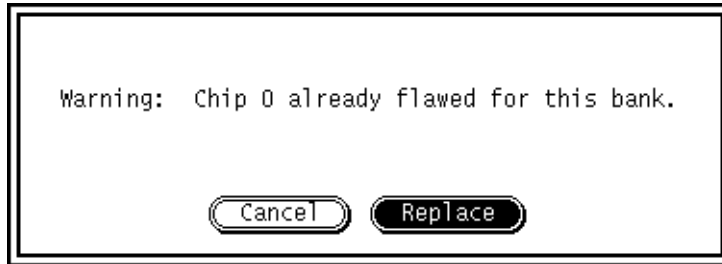
2. Enter the address in the `Address` field and press the return key.

The chip code for the specified chip is automatically calculated using the failing bit and is placed in the `Chip` field.

The chip field represents the physical chip location in the bank for the failing bit. The chip range is 00 to 09.

3. Click on to add the chip information to the `Flaw Chip List` scroll box.

If a flaw already exists for the selected bank, a dialog window opens that requires a response, as shown in the following snap. Respond to the message in this window by clicking on the appropriate button.



4. Click on the or button in the MCE base window. For descriptions of the options available with these buttons, refer to “Using the MCE Apply and Save Buttons” earlier in this document.

Manipulating Flaw Chip Information

Once you have created a table of flaw chip information, you can manipulate it in the following ways:

- Change the serial number or revision for a module
- Swap modules within the Mainframe Modules and Spares Cabinet scroll box
- Print a list of flaws on a module
- Write the flaw information to the OS
- Apply the flaw information
- Delete flaw information from the Flaw Chip List scroll box
- Delete module information
- Delete all flaw information in the table

The following subsections describe these operations. Remember to click on to make changes permanent.

Changing Module Data

To change the serial number or revision data for a module, perform the following procedure:

1. Click on the appropriate module slot in the `Mainframe Modules` or `Spares Cabinet` scroll box. The `Mem Slot` field displays the selected module slot number.
2. Enter the module serial number in the `Serial` field and press the return key.
3. Enter the module revision number in the `Revision` field and press the return key.
4. Click on `Change` to update the module data.

Swapping Modules

You can swap modules in the scroll boxes to indicate you have moved modules in your mainframe. To swap modules in the scroll boxes, perform the following procedure:

NOTE: You can swap only two modules at a time. Be sure to select only two modules for this procedure.


1. Click on the two modules you want to swap. Both modules can be in the `Mainframe Modules` scroll box, or one module can be in the `Mainframe Modules` scroll box and one in the `Spares Cabinet` scroll box.
2. Click on `Swap`; the modules swap places in the scroll boxes.

Printing a List of Flawed Chips



To print a list of the chips you have flawed for a module, click on the module and then click on `Print`.

Writing the Flaw Table to the Operating System

To write the flaw table into a file used by the OS, click on `Save ▾` in the MCE base window or choose `Save -> Write OS and EASE files` (the default option). When the OS is restarted, it reads this file and uses it

to specify which chips should be used. For more information about the  button, refer to “Using the MCE Apply and Save Buttons” earlier in this document.

Applying the Flaw Table

To apply the flaw table for use by MCE, MME, and LME, choose either **Apply -> Write EASE file** (the default) or **Apply -> Do not write EASE file** from the  button in the MCE base window. For more information about the  button, refer to “Using the MCE Apply and Save Buttons” earlier in this document.

NOTE: If MCE is in concurrent mode when you apply a configuration, MCE sends the flaw-map data to MME and LME but does not apply the flaw map to the hardware.

Deleting a Selected Flaw Chip Entry

To delete a flaw-chip table entry, perform the following procedure:

1. Click on the module in the `Mainframe Modules` or `Spares Cabinet` scroll box that contains the chip with the entry you want to delete.
2. Click on the entry in the `Flaw Chip List` scroll box.
3. Choose **Delete -> selected flaw**, as shown at left.



Deleting Flaw Chip Data for a Module

To delete all flaw chip data for a certain module, perform the following procedure:

1. Click on the module in the `Mainframe Modules` or `Spares Cabinet` scroll box for which you want to delete all flaw data.
2. Choose **Delete -> module**, as shown at left.



Deleting All Flaw Table Data

CAUTION
Deleted data is permanently lost. Be sure that you want to delete all flaw table data for the mainframe modules before performing this action.



To delete all flaw table data for the mainframe modules, choose **Delete -> table**, as shown at left. A popup window appears that asks you to verify the deletion.