Credit Net Switch SCU, Core Memory and Terminator Card Diagnostics

Version 2.1

Gary Depelteau - depeltea@bnr.ca

This Document

This document outlines the functional requirements and design description required for SCU, Core Memory and Terminator Card diagnostics CreditNet Switch currently being developed by Harvard University and Bell-Northern Research Limited.

- Bell Northern Research
- Harvard University
-
-

Proprietary Notice

Functional description (FN) 11

Functional description (FN)

1.0 WARNING

Although the basic overall contents of this document are still relevant specific details are out of date. Details pertaining to the exact attributes of various SCU HW blocks are definitely out of date. Also, the HW for CRM, Bus-tap controller and Partial CRC were not implemented along with the corresponding SW to test them. If in doubt, please contact the authors.

2.0 Issue/change control

Table 1: Revision History

Issue	Date	Reason	Change	Author
1.0	Feb. 16, 1994	Initial release for internal review	First Draft	Gary Depelteau
2.0	June 8, 1994	Second release for internal review	Update to reflect HW changes, new HW information and include the Terminator Card	Gary Depelteau
2.1	May 9, 1995	Update	Added Warning indicating document is out of date.	Gary Depelteau

3.0 Activity synopsis

This activity, SCU, Core Memory and Terminator Card Diagnostics, provides software for:

1. Switch Control Unit (SCU) hardware verification;
2. SCU system start-up diagnostics;
3. Core Memory (CM) system start-up diagnostics;
4. Terminator Card (TC) hardware verification
5. TC system start-up diagnostics.

This activity, SCU, Core Memory and TC diagnostics, is referred to as simply SCU Diagnostics throughout the rest of this and other documents. The SCU, depicted in "SCU Block Diagram" on page 12, is part of the Switch Control Module (SCM) and is responsible for central control and monitoring of the Core Memory, backplane and other cards in the Credit Net Switch. "SCU Block Diagram" also indicates the connection of the Terminator Card and Core Memory to the backplane. The Core Memory (also referred to as Core Memory Card (CMC), Core Memory, or Core) is used to buffer cells in the switch while they are waiting to be scheduled and routed for transmission from designated output ports. The TC performs BTL-TTL level conversion, backplane data path integrity checking and provides a Core Memory data path access point. SCU Diagnostics tests the SCU, Core Memory and TC to ensure that they are functioning properly and are capable of carrying out their roles in the switch.

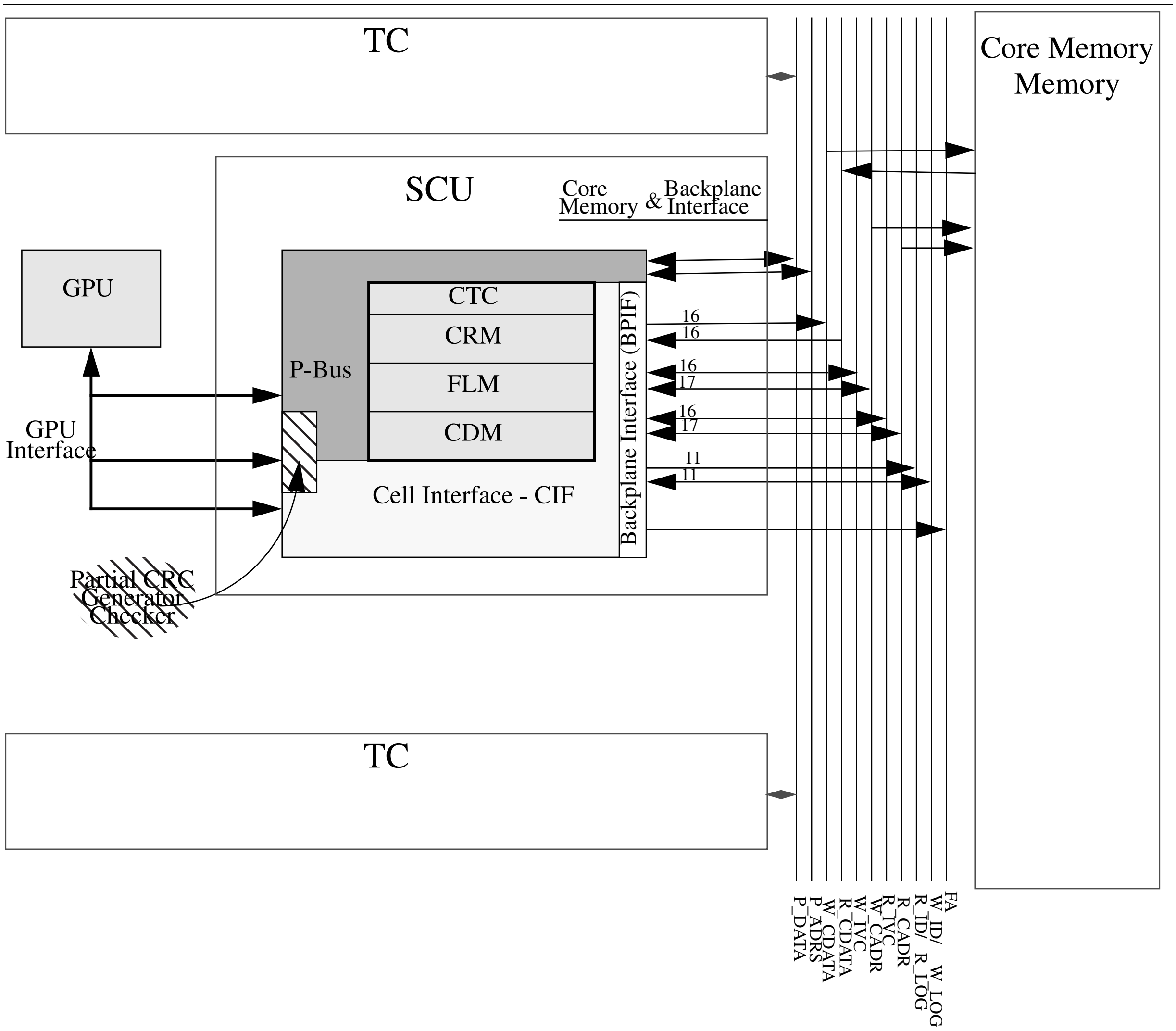

Figure 1SCU Block Diagram

4.0 Overview

4.1 Introduction

SCU Diagnostics execute on the General Processor Unit (GPU), a component of the SCM. These diagnostics are downloaded from the switch workstation after VxWorks real-time operating system. SCU Diagnostics are responsible for performing both hardware verification and system start-up diagnostics. The set of system start-up diagnostics are invoked once GPU Diagnostics have completed by function 'scmSysInitCtl', the switch's initialization control software[3]. A set of hardware verification tests are only performed once per board to ensure that the board has been designed and manufactured correctly. (This set of hardware verification tests is not performed every time the switch boots.) SCU Diagnostics tests the following hardware blocks (all of which reside in the SCU except for the Core Memory and TC) and their major functions. For details please refer to [1] and [4].

- Cell Interface (CIF):
  - Cell transmit;
  - Cell receive;
  - Cell monitoring;
  - Cell integrity;
  - Backplane integrity.
- Partial CRC Generator/Checker:
  - Generates CRC-8/10/32 on a stream of data.
- Free List Manager (FLM):
  - Free List management;
  - Core Memory Buffer Fill (CMBF) Count;
  - FLM Integrity.
- Cell Departure Monitor (CDM)
  - Cell departure monitoring;
  - Free Address (FA) signalling;
  - Chalkmark Searching.
- Core Memory Resource Manager (CRM):
  - Buffer Fill (BF) counters;
  - BF thresholds;
  - Core Memory protection.
- Cell Time Counter (CTC):
  - Cell Period Timer;
  - Interval Timer (IT).
- Processor Bus (P-Bus):
  - SCU interrupt priority encoding;

- Port Module (PM) interrupt monitoring;
- PM reset control;
- Core Memory parity error monitoring.

- Bus-tap Controller (BTC):
  - Cell Bus-Tap (CBT) control;
  - CBT integrity
- Core Memory:
  - Provides storage and retrieval of all cells for the Port Cards (PC) and the SCM.
- Terminator Card (TC)
  - Bus-Transceiver Logic (BTL) and Transistor-Transistor Logic (TTL) signal termination as well as BTL-TTL level conversion;
  - CBT interface;
  - Switch alarms.

4.2 Hardware Requirements

4.2.1 System Start-up Diagnostics Equipment Requirements

As illustrated in "Figure 2 SCU System Start-up Diagnostics H/W Requirements" on page 15, the hardware required by SCU Diagnostics software is:

- SCM with SCU and GPU;
- Switch Backplane;
- Core Memory;
- TC (2).

SCU Diagnostics are executable without the use of any Port Cards (PCs) or other equipment. This ensures the SCM is fully functional when the PCs are initialized and switch integration testing is carried out.

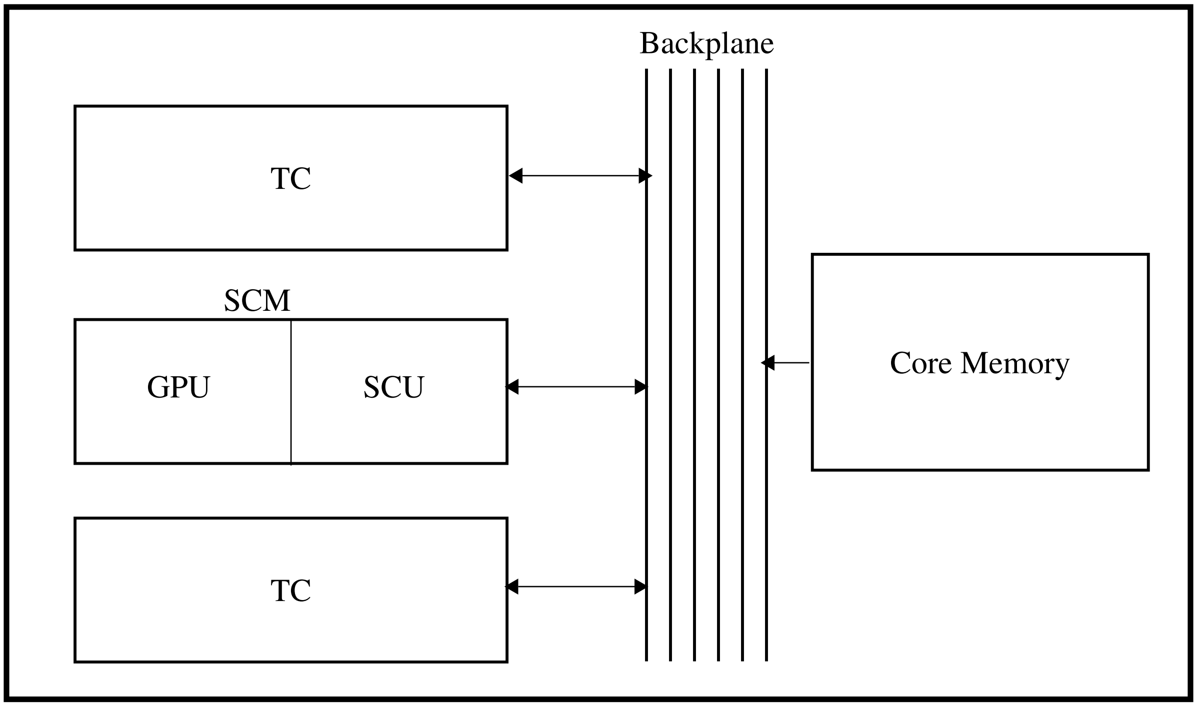

Figure 2 SCU System Start-up Diagnostics H/W Requirements

4.2.2 Hardware Verification Equipment Requirements

In addition to requiring the equipment outline in "System Start-up Diagnostics Equipment Requirements" hardware verification test may also require the following equipment:

- Oscilloscope;
- Logic Analyzer.

5.0 Functional specification

The functional requirements of SCU Diagnostics are determined by the attributes of the SCU, Core Memory and TC. This section lists all the attributes of the various blocks test by this activity. It then defines the requirements for the tests provided by this activity based on these attributes. Finally, tests are summarized with respect to the attributes they cover in Table 2 "Test Case Coverage" .

5.1 Functional Attributes Requiring Testing

This section lists the functional attributes that require testing by SCU Diagnostics according the hardware block in which they reside. For each SCU hardware block, attributes are divided into Functional, Boundary and Performance attributes.

5.1.A Cell Interface Attributes

The CIF's main function is to provide an interface to receive and transmit cells between the GPU and the Core Memory but it also provides selective W_IVC, W_CADR and R_IVC capturing capability. The CIF also provides R_CSIDs to the backplane during read cycles based on R_IVC Lookup Table entries. Cell Header, HEC and payload assembly/disassembly are not performed by the CIF. Please refer to the following diagrams entitled "Figure 3 CIF Data Path" and "Figure 4 CIF IVC FIFOs and Lookup Tables" for more details pertaining to the CIF.

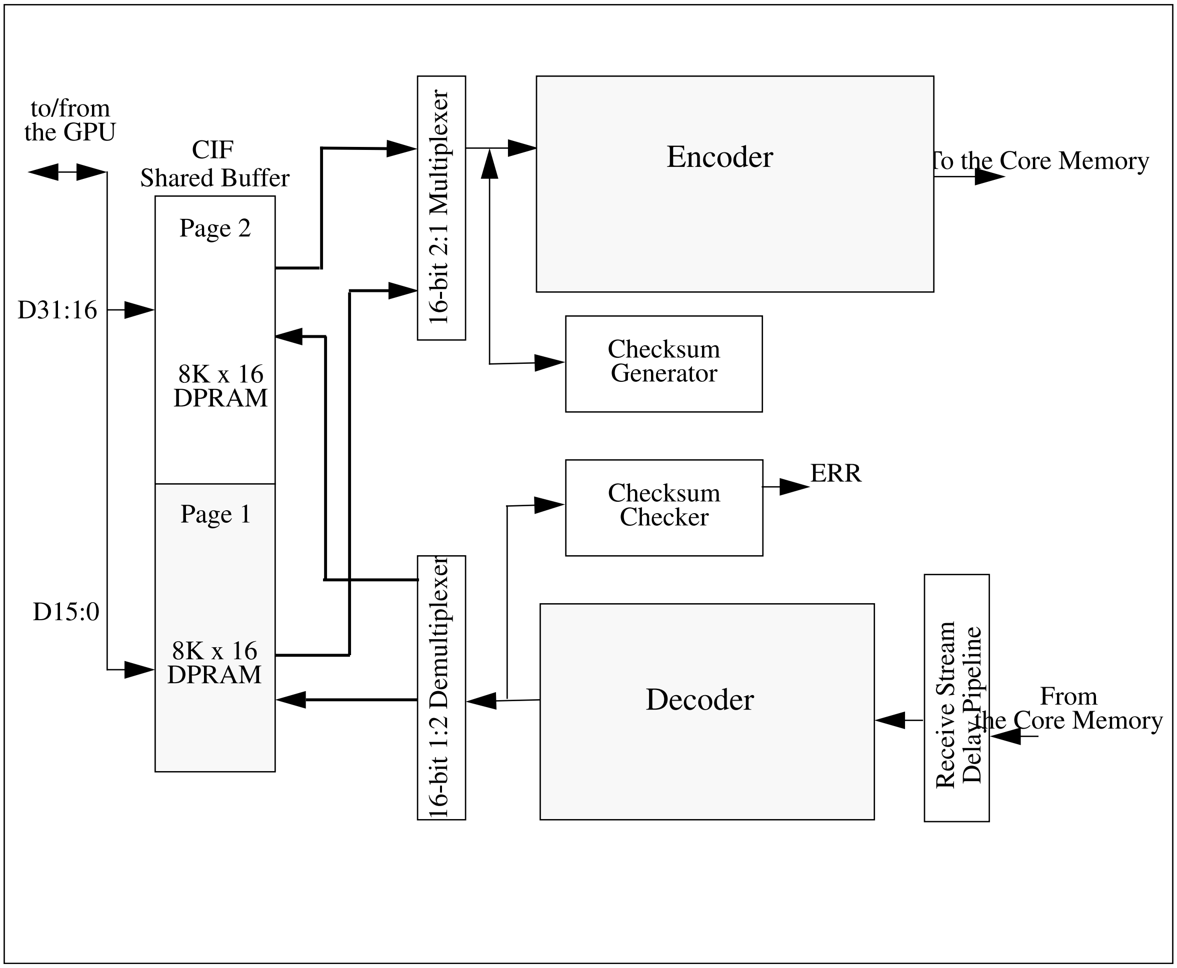

Figure 3 CIF Data Path

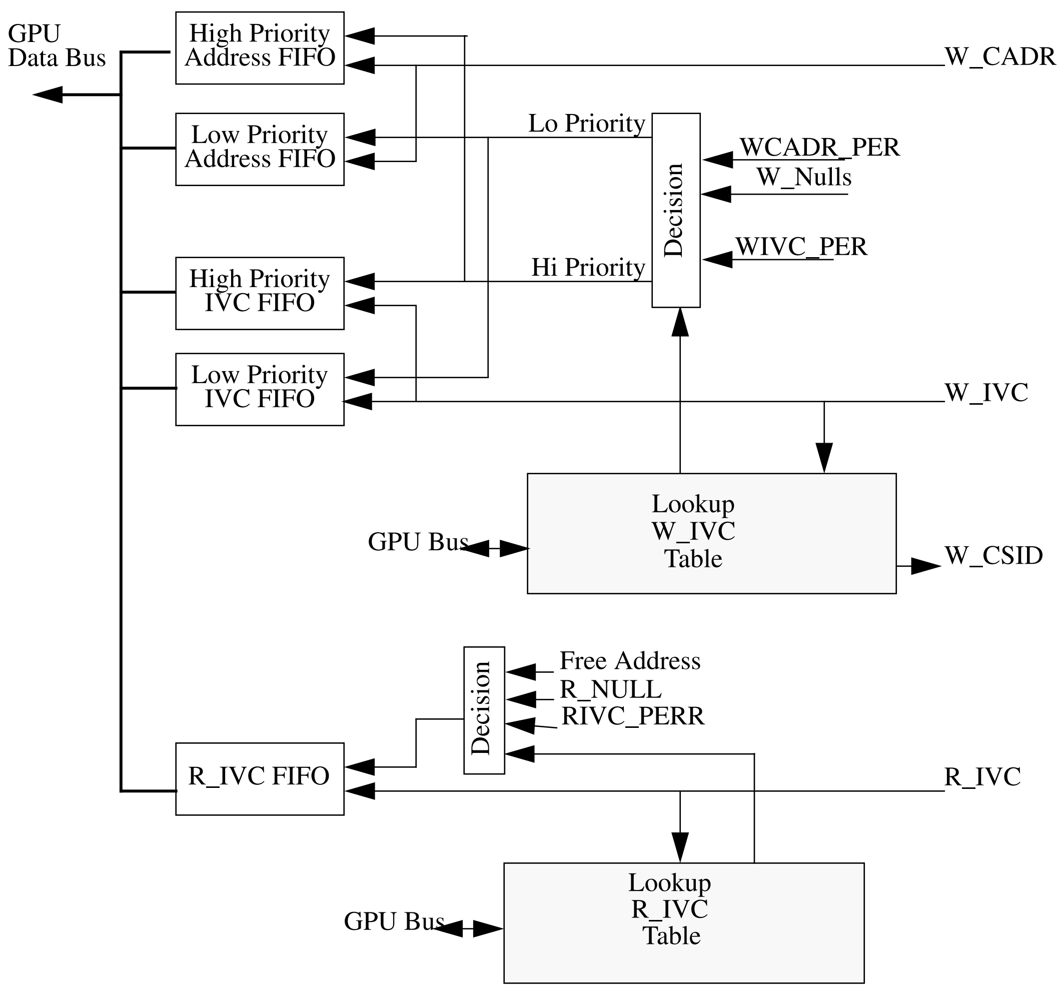

Figure 4 CIF IVC FIFOs and Lookup Tables

The following sections outline the CIF attributes requiring testing.

AF CIF Functional Attributes

Memories and Registers

AF1 Shared Buffer RAM integrity.

AF2 W_IVC Lookup Table RAM integrity.

AF3 R_IVC Lookup Table RAM integrity.

AF4 Control register read/write integrity.

AF5 Data Memory Read Address register read/write integrity.

AF6 Data Memory Write Address register read/write integrity.

AF7 W_IVC_OUT register write integrity.

AF8 R_IVC_OUT/R_CADR_OUT register write integrity.

AF9 W_ID/W_LOG register write integrity.

AF10 Status register read integrity.

AF11 W_IVC_IN1/W_CADR_IN1 register read integrity.

AF12 W_IVC_IN0/W_CADR_IN0 register read integrity.

AF13 R_IVC_IN0 register read integrity.

AF14 R_ID /R_LOG register write integrity.

Initialization

AF15 The high priority W_IVC Data FIFOs' status bits are 00 after reset.

AF16 The low priority W_IVC Data FIFOs' status bits are 00 after reset.

AF17 The R_IVC FIFO's status bits are 00 after reset.

AF18 The RxDATA CRC8 error flag in the CIF status register is 0 after reset.

AF19 The RxDATA code violation flag in the CIF status register is 0 after reset.

AF20 The W_CADR parity error flag in the CIF status register is 0 after reset.

AF21 The W_IVC parity error flag in the CIF status register is 0 after reset.

AF22 The R_CADR parity error flag in the CIF status register is 0 after reset.

AF23 The R_IVC parity error flag in the CIF status register is 0 after reset.

AF24 The GPU Bus parity error flag in the CIF status register is 0 after reset.

Idle Cell Exchange

AF25 The CIF sends an idle cell, stored in the Shared Buffer at cell buffer 0x0, towards the Core Memory when the write one cell bit in the CIF Control register is not set during the SCU's write slot.

AF26 The CIF reads an idle cell from the Core Memory from the CADR currently in the R_IVC_OUT/R_CADR_OUT register when the read one cell bit in the CIF Control register is not set during the SCU's read slot.

Cell Transmit

AF27 A Core Memory write cycle occurs with the W_VALID signal asserted when the Write One Cell and Write Cycle Valid bits are set in the control register.

AF28 A Core Memory write cycle occurs without the W_VALID signal asserted when the Write One Cell bit is set in the control register but the Write Cycle Valid bit is not set.

AF29 The Write One Cell bit in the control register is cleared after the Core Memory write cycle.

AF30 The Write Cycle Valid bit in the control register is cleared after a Core Memory write cycle.

AF31 A CIF Core Memory write cycle causes the cell specified by the Data Memory Write Address register to be transmitted from the shared buffer to the Core Memory at the location specified by the W_CADR. This only occurs if W_VALID is asserted, and W_CE_NULL and WIVC_PERR are unasserted.

AF32 A CIF Core Memory write cycle does not cause a cell to be transmitted to the Core Memory if W_VALID is not asserted.

AF33 A CIF Core Memory write cycle does not cause a cell to be transmitted to the Core Memory if W_CE_NULL is asserted.

AF34 A CIF Core Memory write cycle does not cause a cell to be transmitted to the Core Memory if WIVC_PERR is asserted.

AF35 CRC8 is correctly calculated over 55 bytes of the cell (53 cell bytes plus 2 IVC bytes) and inserted in byte 56 during a Core Memory write cycle when the CRC8 enable bit in the control register is set for the write cycle.

AF36 CRC8 is not calculated for the cell and inserted in byte 56 during a Core Memory write cycle when the CRC8 enable bit in the control register is not set for the write cycle.

AF37 The CRC8 enable bit in the control register remains set after a write cycle for which it was set.

AF38 The CRC8 enable bit in the control register remains clear after a write cycle for which it was clear.

AF39 7-8 encoding is performed on the 56 bytes of the cell (53 cell bytes plus 2 IVC bytes plus 1 CRC8 byte) and the 64 encoded bytes are placed in the Core Memory.

AF40 Core Memory write cycle causes the specified W_IVC to be put out on the backplane.

AF41 Core Memory write cycle causes the W_CSID identified for the W_IVC in the W_IVC Lookup Table to be placed on the backplane.

AF42 Core Memory write cycle causes the contents of the W_ID/W_LOG register to be placed on the backplane on the W_ID and W_LOG CCB buses.

AF43 When set, the DBM Configure bit is cleared on write cycles.

AF44 When set, the DBM Capture enable bit is cleared on write cycles.

Cell Receive

AF45 A Core Memory read cycle occurs with the R_VALID signal asserted when the Read One Cell and Read Cycle Valid bits are set in the control register.

AF46 A Core Memory read cycle occurs without the R_VALID signal asserted when the Read One Cell bit is set in the control register but the Read Cycle Valid bit is not set.

AF47 The Read One Cell bit in the control register is cleared after the Core Memory read cycle.

AF48 The Read Cycle Valid bit in the control register is cleared after a Core Memory read cycle.

AF49 CRC8 is correctly checked over 55 bytes of the cell (53 cell bytes plus 2 IVC bytes) and compared to byte 56 during a Core Memory read cycle when the CRC8 enable bit in the control register is set for the read cycle.

AF50 CRC8 is not checked for the cell during a Core Memory read cycle when the CRC8 enable bit in the control register is not set for the read cycle.

AF51 The CRC8 enable bit in the control register remains set after a read cycle for which it was set.

AF52 The CRC8 enable bit in the control register remains clear after a read cycle for which it was clear.

AF53 8-7 decoding is performed on the 64 bytes of the cell coming from the Core Memory and the 56 decoded bytes are placed in the Shared Buffer.

AF54 A CIF Core Memory read cycle causes the cell at the specified R_CADR in the Core Memory to be read into the cell buffer in the Shared Buffer specified in the Data Memory Read Address register. This only occurs if R_VALID is asserted and RIVC_PERR and RCADR_PERR are unasserted.

AF55 A CIF Core Memory read cycle does not cause a cell to be received from the Core Memory if R_VALID is not asserted.

AF56 A CIF Core Memory read cycle does not cause a cell to be received from the Core Memory if RIVC_PERR is asserted.

AF57 A CIF Core Memory read cycle does not cause a cell to be received from the Core Memory if RCADR_PERR is asserted.

AF58 Core Memory read cycle causes the specified R_CADR to be placed on the backplane.

AF59 Core Memory read cycle causes the specified R_IVC to be placed on the backplane.

AF60 Core Memory read cycle causes the R_CSID identified for the R_IVC in the R_IVC Lookup Table to be placed on the backplane.

AF61 Core Memory read cycle causes the contents of the R_ID/R_LOG register to be placed on R_ID and R_LOG CCB buses on the backplane.

AF62 When set, the DBM Capture enable bit is cleared on read cycles.

Cell Monitoring

AF63 The high priority W_IVC FIFOs capture W_IVC and W_CADR data for W_IVCs whose high priority W_IVC FIFO enable bit is set in the W_IVC Lookup table. This only occurs during Core Memory write cycles during which the W_VALID backplane signal is asserted and WIVC_PERR and W_CE_NULL are unasserted.

AF64 The high priority W_IVC FIFOs does not capture W_IVC and W_CADR data for W_IVCs whose high priority W_IVC FIFO enable bit is not set in the W_IVC Lookup table.

AF65 The high priority W_IVC FIFOs does not capture W_IVC and W_CADR data during cycles for which the W_VALID backplane signal is not asserted.

AF66 The high priority W_IVC FIFOs does not capture W_IVC and W_CADR data during cycles for which the WIVC_ PERR (Write IVC Parity Error) backplane signal is asserted.

AF67 The high priority W_IVC FIFOs does not capture W_IVC and W_CADR data during cycles for which the W_ CE_NULL (Write Common Equipment Null) backplane signal is asserted.

AF68 The low priority W_IVC FIFOs capture W_IVC and W_CADR data for W_IVCs whose low priority W_IVC FIFO enable bit is set in the W_IVC Lookup table. This only occurs during Core Memory write cycles during which the W_VALID backplane signal is asserted and WIVC_PERR and W_CE_NULL are unasserted.

AF69 The low priority W_IVC FIFOs does not capture W_IVC and W_CADR data for W_IVCs whose low priority W_IVC FIFO enable bit is not set in the W_IVC Lookup table.

AF70 The low priority W_IVC FIFOs does not capture W_IVC and W_CADR data during cycles for which the W_VALID backplane signal is not asserted.

AF71 The low priority W_IVC FIFOs does not capture W_IVC and W_CADR data during cycles for which the WIVC_ PERR (Write IVC Parity Error) backplane signal is asserted.

AF72 The low priority W_IVC FIFOs does not capture W_IVC and W_CADR data during cycles for which the W_ CE_NULL (Write Common Equipment Null) backplane signal is asserted.

AF73 The R_IVC FIFO captures R_IVC data for IVCs whose R_IVC FIFO enable bit is set in the R_IVC Lookup table. This only occurs during Core Memory read cycles during which the R_VALID backplane signal is asserted and RCADR_PERR and RIVC_PERR are unasserted.

AF74 The R_IVC FIFO does not capture data for IVCs whose R_IVC FIFO enable bit is not set in the R_IVC Lookup table.

AF75 The R_IVC FIFO does not capture R_IVC data during cycles for which the R_VALID backplane signal is not asserted.

AF76 The R_IVC FIFO does not capture R_IVC data during cycles for which the RCADR_PERR (Read CADR Parity Error) backplane signal is asserted.

AF77 Reading the W_IVC_DATA0_IN register provides the W_IVC and W_CADR data at the head of the high priority W_IVC FIFOs and pops this data off of that FIFOs.

AF78 Reading the W_IVC_DATA1_IN register provides the W_IVC and W_CADR data at the head of the low priority W_IVC FIFOs and pops this data off of that FIFOs.

AF79 Reading the R_IVC_IN register provides the R_IVC at the head of the R_IVC FIFO and pops it off of that FIFO.

AF80 The high priority W_IVC Data FIFOs' status bits are 00 when these FIFOs contain 0 entries.

AF81 The high priority W_IVC Data FIFOs' status bits are 01 when these FIFOs contain 1 to 127 entries.

AF82 The high priority W_IVC Data FIFOs' status bits are 10 when these FIFOs contain 128 to 3968 entries.

AF83 The GPU is interrupted when the high priority W_IVC Data FIFOs' status bits change from 01 to 10 (on the 128th entry).

AF84 The high priority W_IVC Data FIFOs' status bits are 11 when these FIFOs contain 3969 to 4096 entries.

AF85 The GPU is interrupted when the high priority W_IVC Data FIFOs' status bits change from 10 to 11 (on the 3969th entry).

AF86 The FIFOs' error bit in the status register is set when the high priority W_IVC Data FIFOs (W_IVC and W_CADR) contain a different number of entries.

AF87 The low priority W_IVC Data FIFOs' status bits are 00 when these FIFOs contain 0 entries.

AF88 The low priority W_IVC Data FIFOs' status bits are 01 when these FIFOs contain 1 to 127 entries.

AF89 The low priority W_IVC Data FIFOs' status bits are 10 when these FIFOs contain 128 to 3968 entries.

AF90 The GPU is interrupted when the low priority W_IVC Data FIFOs' status bits change from 01 to 10 (on the 128th entry).

AF91 The low priority W_IVC Data FIFOs' status bits are 11 when these FIFOs contain 3969 to 4096 entries.

AF92 The GPU is interrupted when the low priority W_IVC Data FIFOs' status bits change from 10 to 11 (on the 3969th entry).

AF93 The FIFOs' error bit in the status register is set when the low priority W_IVC Data FIFOs (W_IVC and W_CADR) contain a different number of entries.

AF94 The R_IVC FIFO's status bits are 00 when it contains 0 entries.

AF95 The R_IVC FIFO's status bits are 01 when it contains 1 to 127 entries.

AF96 The R_IVC FIFO's status bits are 10 when it contains 128 to 3968 entries.

AF97 The GPU is interrupted when the R_IVC FIFOs' status bits change from 01 to 10 (on the 128th entry).

AF98 The R_IVC FIFO's status bits are 11 when it contains 3969 to 4096 entries.

AF99 The GPU is interrupted when the R_IVC FIFOs' status bits change from 10 to 11 (on the 3969th entry).

AF100 The FIFOs error bit in the status register is not set when the low priority W_IVC Data FIFOs (W_IVC and W_CADR) contain the same number of entries and high priority W_IVC Data FIFOs (W_IVC and W_CADR) contain the same number of entries.

Cell Integrity

AF101 The RxDATA CRC8 error flag in the CIF status register is set when a R_CDATA CRC8 error is encountered.

AF102 The GPU is interrupted when a R_CDATA CRC8 error is encountered.

AF103 The RxDATA CRC8 error flag in the CIF status register is not set when a R_CDATA CRC8 error is not encountered.

AF104 The GPU is not interrupted for a R_CDATA CRC8 error when a R_CDATA CRC8 error is not encountered.

AF105 Cell data integrity. All bytes of the cell header and payload occur at their proper locations within the Shared Buffer as outlined in [1].

Backplane Integrity

AF106 The W_IVC parity error flag in the CIF status register is set only when a W_IVC parity error is encountered.

AF107 The W_IVC parity error flag in the CIF status register is not set only when a W_IVC parity error is not encountered.

AF108 The R_CADR parity error flag in the CIF status register is set when a R_CADR parity error is encountered.

AF109 The R_CADR parity error flag in the CIF status register is not set only when a R_CADR parity error is not encountered.

AF110 The R_IVC parity error flag in the CIF status register is set only when a R_IVC parity error is encountered.

AF111 The R_IVC parity error flag in the CIF status register is not set only when a R_IVC parity error is not encountered.

AB CIF Boundary Attributes

AB1 Reading the W_IVC_DATA0_IN register when the high priority W_IVC and W_CADR FIFOs are empty does not cause the high priority W_IVC FIFOs' status bits to change from 00.

AB2 Reading the W_IVC_DATA0_IN register when the high priority W_IVC and W_CADR FIFOs are empty returns the last value read out of this register unless the SCU has just been reset.

AB3 Reading the W_IVC_DATA0_IN register when the high priority W_IVC and W_CADR FIFOs are empty returns 0 when the SCU has just been reset.

AB4 The high priority W_IVC FIFOs can both hold the maximum 4K entries.

AB5 A Core Memory write cycle, for a W_IVC whose high priority W_IVC FIFO enable bit is set in the W_IVC Lookup table, performed when the high priority W_IVC FIFOs already contain 4K entries, does not result in the W_IVC and W_CADR being inserted into these FIFOs.

AB6 A Core Memory write cycle, for a W_IVC whose high priority W_IVC FIFO enable bit is set in the W_IVC Lookup table, performed when the high priority W_IVC FIFOs already contains 4K entries, does not cause the high priority W_IVC FIFOs' status bits to change from 11.

AB7 Reading the W_IVC_DATA1_IN register when the low priority W_IVC and W_CADR FIFOs are empty does not cause the low priority W_IVC FIFOs' status bits to change from 00.

AB8 Reading the W_IVC_DATA1_IN register when the low priority W_IVC and W_CADR FIFOs are empty returns the last value read out of this register unless the SCU has just been reset.

AB9 Reading the W_IVC_DATA1_IN register when the low priority W_IVC and W_CADR FIFOs are empty returns 0 when the SCU has just been reset.

AB10 The low priority W_IVC FIFOs can both hold the maximum 4K entries.

AB11 A Core Memory write cycle, for a W_IVC whose low priority W_IVC FIFO enable bit is set in the W_IVC Lookup table, performed when the low priority W_IVC FIFOs already contain 4K entries, does not result in the W_IVC and W_CADR being inserted into these FIFOs.

AB12 A Core Memory write cycle, for a W_IVC whose low priority W_IVC FIFO enable bit is set in the W_IVC Lookup table, performed when the low priority W_IVC FIFOs already contains 4K entries, does not cause the low priority W_IVC FIFOs' status bits to change from 11.

AB13 Reading the R_IVC_IN register when the R_IVC FIFO is empty does not cause the R_IVC FIFO's status bits to change from 00.

AB14 Reading the R_IVC_IN register when the R_IVC FIFO is empty returns the last value read out of this register unless the SCU has just been reset.

AB15 Reading the R_IVC_IN register when the R_IVC FIFO is empty returns 0 when the SCU has just been reset.

AB16 The R_IVC FIFO can hold the maximum 4K entries.

AB17 A Core Memory read cycle, for a R_IVC whose R_IVC FIFO enable bit is set in the R_IVC Lookup table, performed when the R_IVC FIFO already contains 4K entries, does not result in the R_IVC being inserted into this FIFO.

AB18 A Core Memory read cycle, for a R_IVC whose R_IVC FIFO enable bit is set in the R_IVC Lookup table, performed when the R_IVC FIFO already contains 4K entries, does not cause the R_IVC FIFO's status bits to change from 11.

AB19 The CIF Shared Buffer memory can hold 16K 16-bit words.

AB20 The R_IVC Lookup Table RAM can hold 8K entries.

AB21 The R_IVC Lookup Table RAM can hold 8K entries.

AP CIF Performance Attributes

AP1 The CIF can transmit at full speed. That is, it can continually transmit cells to the Core Memory in each of its write cycle slots.

AP2 The CIF can transmit at full speed while cycling through all 512 cells buffers in the Shared Buffer. That is, it can continually transmit cells from the Core Memory in each of its write cycle slots while the Data Memory Write Address register is cycling through the 512 addresses in the Shared Buffer.

AP3 The CIF can receive at full speed. That is, it can continually receive cells from the Core Memory in each of its read cycle slots.

AP4 The CIF can receive at full speed while cycling through all 512 cells buffers in the Shared Buffer. That is, it can continually receive cells from the Core Memory in each of its read cycle slots while the Data Memory Read Address register is cycling through the 512 addresses in the Shared Buffer.

AP5 The CIF can perform combined transmit and receive at full speed. That is, it can continually transmit to and receive cells from the Core Memory in each of its write and read cycle slots within the same cell period while the values in the Data Memory Write Address register and Data Memory Read Address register are cycling through the cell addresses in the Shared Buffer.

AP6 The high priority W_IVC Data FIFOs can capture data at full speed. This means that it can capture 16 W_IVCs and W_CADRs per cell period when the high priority W_IVC FIFO enable bits are set for all the W_IVCs occurring during the cell period.

AP7 The low priority W_IVC Data FIFOs can capture data at full speed. This means that it can capture 16 W_IVCs and W_CADRs per cell period when the low priority W_IVC FIFO enable bits are set for all the W_IVCs occurring during the cell period.

AP8 The R_IVC FIFO can capture data at full speed. This means that it can capture 16 R_IVCs per cell period when the R_IVC FIFO enable bits are set for all the R_IVCs occurring the cell period.

5.1.B Partial CRC Generator

NOTE: This section is far from complete. It will be finished once the HW has been completely defined.

The Partial CRC Checker/Generator is a programmable calculating tool used to calculate CRC8/10/16/32 on an input stream of data. The following sections outline the Partial CRC Checker/Generator requiring testing.

BF Partial CRC Checker/Generator Functional Attributes

Memories and Registers

BF1 CRC data input register write integrity.

BF2 Partial CRC calculation output register read integrity.

BF3 Control Register read/write integrity.

CRC Calculation

BF4 With the Previous CRC/New Word bit set to 1 in the control register, writing a 32-bit value written to the Input Register causes it appear in the Output Register by the next time it is read.

BF5 With the Previous CRC/New Word bit set to 0 and the CRC8 bit set, writing a word to the Input Register causes the least significant 8 bits to be included (with the old partial CRC8) in the calculation of a new partial CRC8 which is available in the Output Register the next time it is read.

BB Partial CRC Checker/Generator Boundary Attributes

TBD

BP Partial CRC Checker/Generator Performance Attributes

TBD

5.1.C Free List Manager Attributes

The FLM manages the list of free Core Memory cell addresses and provides a free address on each cell write cycle. Please refer to the following diagram entitled "Figure 5 FLM Block Diagram" for more details pertaining to the FLM.

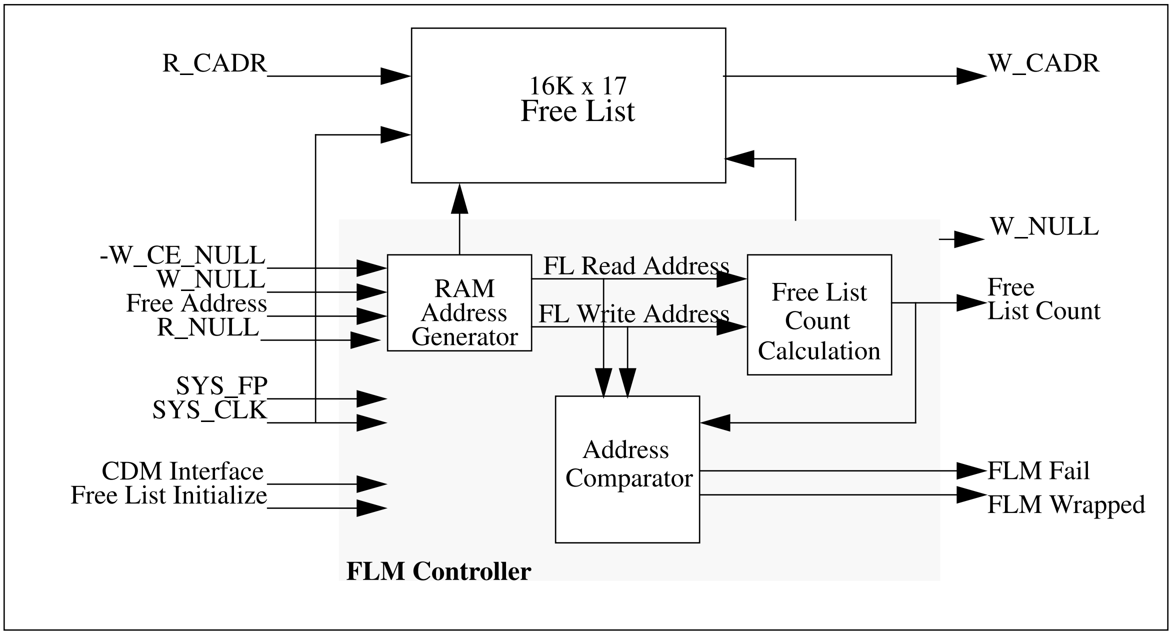

Figure 5 FLM Block Diagram

The following sections outline the FLM attributes requiring testing.

CF <u>FLM Functional Attributes</u>

Memories and Registers

CF1 Free List RAM Integrity.

CF2 FLM Status register read integrity.

CF3 FLM Control register read/write integrity.

FLM Initialization

CF4 The FLM Halt bit in the FLM Control register is set after reset.

CF5 The FLM Empty bit in the FLM Status register is set after reset.

Free List Management

CF6 R_CADRs are placed in the FLM when the FA signal produced by the CDM is asserted.

CF7 W_CADRs are popped off the top of the Free List and placed on the backplane during write cycles in which the W_VALID signal is asserted and the FLM Halt bit is not asserted.

CF8 W_CADRs popped off the Free List and driven on the backplane during write cycles in which the W_VALID signal is not asserted are also recycled and reused four CCB Time-slots later.

CF9 W_CADRs popped off the Free List and driven on the backplane during write cycles in which the W_CE_NULL signal is asserted are also recycled and reused four CCB Time-slots later.

CF10 W_CADRs popped off the Free List are not driven on the backplane during write cycles in which the FLM Halt bit is set but are instead recycled and reused four CCB Time-slots later.

Core Memory Buffer Fill (CMBF) Counter

CF11 The CMBF is initialized to 0 after reset.

CF12 For write cycles which have W_VALID asserted, WIVC_PERR and W_CE_NULL unasserted and the FLM Halt signal unasserted, the CMBF counter is incremented.

CF13 For write cycles with the W_VALID signal unasserted, the CMBF counter is not changed.

CF14 For write cycles with the WIVC_PERR signal asserted, the CMBF counter is not changed.

CF15 For write cycles with the FLM Halt bit asserted, the CMBF counter is not changed.

CF16 For write cycles with the W_CE_NULL signal asserted, the CMBF counter is not changed.

CF17 For read cycles which have R_VALID asserted, FA asserted, RIVC_PERR unasserted and RCADR_PERR unasserted, the CMBF counter is decremented.

CF18 For read cycles with the FA signal unasserted, the CMBF counter is not changed.

CF19 For read cycles with the R_VALID signal unasserted, the CMBF counter is not changed.

CF20 For read cycles with the RIVC_PERR signal asserted, the CMBF counter is not changed.

CF21 For read cycles with the RCADR_PERR signal asserted, the CMBF counter is not changed.

CF22 When set, the FLM Halt bit is not cleared on write cycles.

CF23 When set, the FLM Halt bit is not cleared on read cycles.

FLM Integrity

CF24 The Free List Wrap (FLW) signal in the FLM Status register is asserted only when the FLM goes through the entire FLM memory (which occurs when the FLM internal head pointer is at location zero).

CF25 The GPU is interrupted when the Free List Wraps.

CF26 The Free List Fail (FLF) signal in the FLM Status register is asserted when the CMBF count is not the same as the difference between the internal tail and head pointers when the head pointer is at location zero and when the FLM Halt bit is not asserted.

CF27 The Free List Fail (FLF) signal in the FLM Status register is not asserted when the CMBF count is the same as the difference between the internal tail and head pointers when the head pointer is at location zero.

CF28 The GPU is interrupted when a Free List Fail occurs and FLM Halt is not set.

CF29 The Free List Fail (FLF) signal in the FLM Status register is not asserted when the FLM Halt bit is set.

CF30 The GPU is not interrupted when a Free List Fail occurs but the Halt bit is set.

CB <u>FLM Boundary Attributes</u>

CB1 The Free List can hold the maximum 16K CADRs.

CB2 No more than 16K CADRs can be placed on the Free List.

CB3 The FLM Empty bit in the FLM Status register is set when the FLM contains 0 CADRs.

CB4 The GPU is interrupted when the FLM Empty bit is set and the Halt bit is not set.

CB5 The GPU is interrupted when the FLM Empty bit is set and the Halt bit is set.

CB6 The FLM Empty bit in the FLM Status register is not cleared when a write cycle with W_VALID asserted is performed when the FLM Empty signal is already set.

CB7 The W_CE_NULL signal is asserted for Core Memory write cycles during which the FLM is empty.

CB8 The FLM Full bit in the FLM Status register is set when the FLM contains 16K CADRs.

CB9 The GPU is interrupted when the FLM Full bit is set and the Halt bit is not set.

CB10 The FLM Full bit in the FLM Status register is not cleared when a read cycle causing the FA signal to be asserted occurs when the FLM Full bit is already set.

CB11 For read cycles which have R_VALID asserted, RIVC_PERR unasserted and RCADR_PERR unasserted, the CMBF counter is not changed when it is already 0.

CB12 For write cycles which have W_VALID asserted, WIVC_PERR and W_CE_NULL unasserted, the CMBF counter is not incremented when it is already at its maximum value(16K).

CP FLM Performance Attributes

CP1 The FLM can (re)cycle the full 16K CADRs.

CP2 The FLM pushes all R_CADRs onto the Free List at full speed. (That is when 16 Core Memory read cycles are occurring each cell period for which the CDM's FA signal is asserted.)

CP3 The FLM pops W_CADRs off of the Free List and places them on the backplane at full speed. (That is when 16 Core Memory write cycles are occurring each cell period.)

CP4 The FLM can push and pop CADRs to/from the Free List at full speed. (That is when 16 Core Memory read cycles with FA asserted and 16 Core Memory write cycles are occurring each cell period.)

CP5 The FLM can push R_CADRs onto the Free List when Core Memory read cycles (resulting in the FA signal being asserted) are being performed via the CIF once per cell cycle.

CP6 The FLM can pop W_CADRs off of the Free List and place them on the backplane when Core Memory write cycles are being performed via the CIF once per cell cycle.

CP7 The FLM can push and pop CADRs to/from the Free List when both Core Memory read and write cycles are being performed via the CIF once each per cell cycle.

CP8 The CMBF counter can increment at full speed when 16 Core Memory writes and no Core Memory reads are occurring per cell cycle.

CP9 The CMBF counter can decrement at full speed when 16 Core Memory reads and no Core Memory writes are occurring per cell cycle.

CP10 The CMBF counter increments correctly when the CIF is performing only Core Memory writes at its full speed (one write per cell cycle).

CP11 The CMBF counter decrements correctly when the CIF is performing only Core Memory reads at its full speed (one read per cell cycle).

5.1.D Cell Departure Monitor Attributes

The CDM is responsible for notifying the FLM when it should place a R_CADR on the Free List as well as performing Chalkmark Searches to find stale cells in the Core Memory. Please refer to the following diagram entitled "Figure 6 CDM Block Diagram" for further details on the CDM. The CDM's DBM space is indexed by W_IVC and contains the Distribution Bitmap (DBM) for each of the possible 8K IVCs. The working DBM (wDBM) space is indexed by W_CADR and R_CADR and it stores the current DBMs for each cell buffer in the Core Memory. A DBM contains 16 bits which indicate which ports have to transmit a particular cell, a multicast bit indicating which type of chalkmark searching applies to a cell and a chalkmark bit.

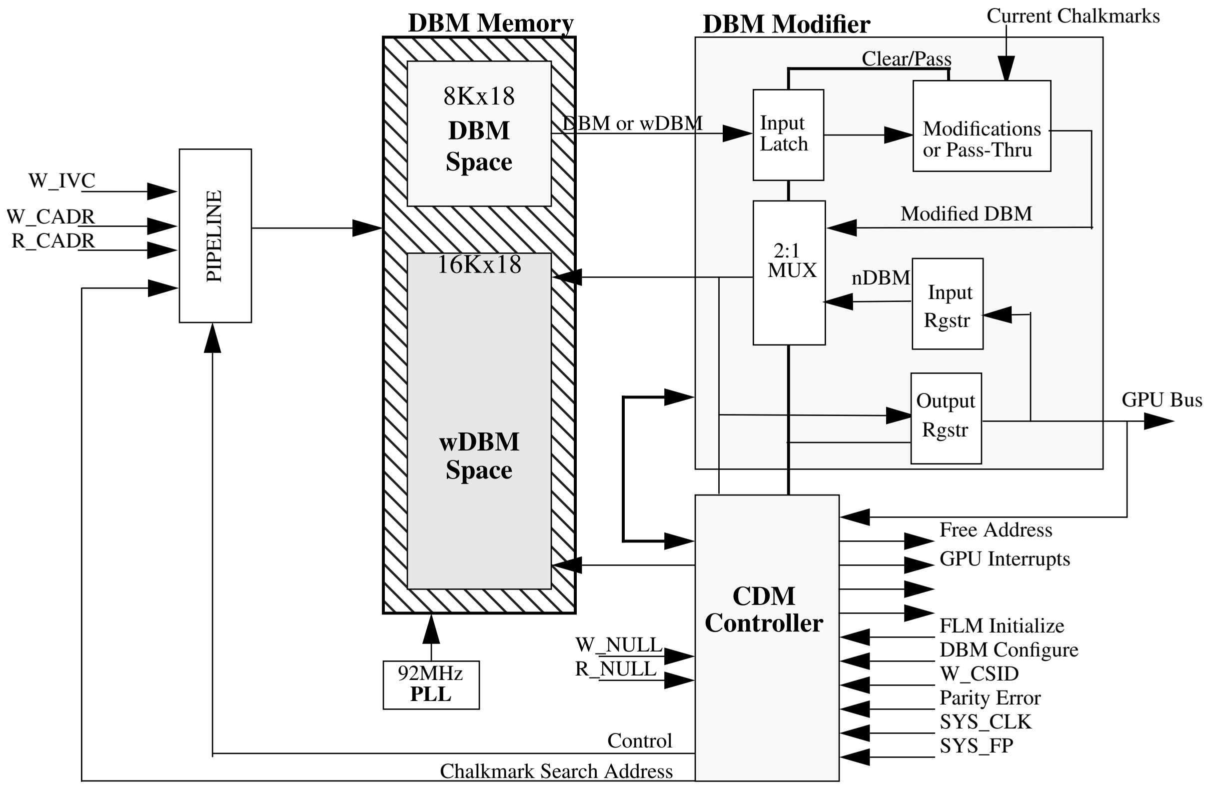

Figure 6 CDM Block Diagram

The following sections outline the CDM attributes requiring testing.

DF CDM Functional Attributes

Memories and Registers

DF1 DBM Space RAM integrity.

DF2 Working DBM (wDBM) Space RAM integrity.

DF3 New DBM Register write integrity.

DF4 New Casting Bit Register write integrity.

DF5 Requested DBM Register read integrity.

DF6 Current Chalkmark Bit register read/write integrity for both the multicast and unicast bits.

DF7 Stale CADR Register readintegrity.

DF8 CDM Control Register read/write integrity.

DF9 CDM Status Register read integrity.

Cell Distribution Monitoring

DF10 The DBM and casting bit indexed by W_IVC is updated by the values in the New DBM and New Casting Bit registers for write cycles via the CIF when the DBM configure bit is set and the W_VALID signal is not asserted.

DF11 For write cycles which have W_VALID asserted, WIVC_PERR, W_CE_NULL are unasserted, the DBM indexed by W_IVC is copied into the wDBM memory at the location indexed by W_CADR.

DF12 For write cycles with the W_VALID signal unasserted, the DBM indexed by W_IVC is not copied into the wDBM at the location indexed by W_CADR.

DF13 For write cycles with the WIVC_PERR signal asserted, the DBM indexed by W_IVC is not copied into the wDBM at the location indexed by W_CADR.

DF14 For write cycles with the W_CE_NULL signal asserted, the DBM indexed by W_IVC is not copied into the wDBM at the location indexed by W_CADR.

DF15 For read cycles with the R_VALID signal asserted, the RIVC_PERR unasserted and the RCADR_PERR unasserted, the wDBM at the location indexed by R_CADR is modified to clear the port bit corresponding to the port issuing the read.

DF16 For read cycles with the R_VALID signal unasserted, the wDBM at the location indexed by R_CADR is not modified.

DF17 For read cycles with the RIVC_PERR signal asserted, the wDBM at the location indexed by R_CADR is not modified.

DF18 For read cycles with the R_CADR_PERR signal asserted, the wDBM at the location indexed by R_CADR is not modified.

Free Address Signalling

DF19 For read cycles with the R_VALID signal asserted, the RIVC_PERR unasserted and the RCADR_PERR unasserted, the FA signal is asserted when the last port bit in the wDBM for the R_CADR is cleared.

DF20 For read cycles, the FA signal is not asserted for a R_CADR when one or more port bits is still set after the wDBM has been modified.

DF21 For read cycles with the R_VALID signal asserted, the FA signal is asserted when the FLM initialization bit in the CIF Control Register is set.

DF22 For read cycles with the R_VALID signal unasserted, the FA signal is asserted when the FLM initialization bit in the CIF Control Register is set.

DBM Memory Capturing

DF23 For write cycles with the W_VALID signal asserted and the DBM Capture Enable bit in the CIF Control Register set, the contents of the DBM Space indexed by the value of the W_IVC are placed in the Requested DBM register.

DF24 For write cycles without the W_VALID signal asserted and the DBM Capture Enable bit in the CIF Control Register set, the contents of the DBM Space indexed by the value of the W_IVC are placed in the Requested DBM register.

DF25 For write cycles without the DBM Capture Enable bit in the CIF Control Register set, the contents of the DBM Space indexed by the value of the W_IVC are not placed in the Requested DBM register.

DF26 For read cycles with the R_VALID signal asserted and the DBM Capture Enable bit in the CIF Control Register set, the contents of the wDBM Space indexed by the value of the R_CADR are placed in the Requested DBM register.

DF27 For read cycles without the R_VALID signal asserted and the DBM Capture Enable bit in the CIF Control Register set, the contents of the wDBM Space indexed by the value of the R_CADR are placed in the Requested DBM register.

DF28 For read cycles without the DBM Capture Enable bit in the CIF Control Register set, the contents of the wDBM Space indexed by the value of the R_CADR are not placed in the Requested DBM register.

Chalkmark Searching

DF29 For write cycles which have W_VALID asserted and WIVC_PERR and W_CE_NULL unasserted, the current unicast chalkmark bit is copied into the chalkmark bit in the wDBM memory at the location indexed by W_CADR if its multicast chalkmark bit is 0.

DF30 For write cycles with the W_VALID signal unasserted, the current unicast chalkmark bit is not copied into the chalkmark bit in the wDBM memory at the location indexed by W_CADR if its multicast chalkmark bit is 0.

DF31 For write cycles with the WIVC_PERR signal asserted, the current unicast chalkmark bit is not copied into the chalkmark bit in the wDBM memory at the location indexed by W_CADR if its multicast chalkmark bit is 0.

DF32 For write cycles with the W_CE_NULL signal asserted, the current unicast chalkmark bit is not copied into the chalkmark bit in the wDBM memory at the location indexed by W_CADR if its multicast chalkmark bit is 0.

DF33 For write cycles which have W_VALID asserted and WIVC_PERR and W_CE_NULL unasserted, the current multicast chalkmark bit is copied into the chalkmark bit in the wDBM memory at the location indexed by W_CADR if its multicast chalkmark bit is 1.

DF34 For write cycles with the W_VALID signal unasserted, the multicast chalkmark bit is not copied into the chalkmark bit in the wDBM memory at the location indexed by W_CADR if its multicast chalkmark bit is 1.

DF35 For write cycles with the WIVC_PERR signal asserted, the current multicast chalkmark bit is not copied into the chalkmark bit in the wDBM memory at the location indexed by W_CADR if its multicast chalkmark bit is 1.

DF36 For write cycles with the W_CE_NULL signal asserted, the current multicast chalkmark bit is not copied into the chalkmark bit in the wDBM memory at the location indexed by W_CADR if its multicast chalkmark bit is 1.

DF37 For read cycles which have R_VALID asserted, RIVC_PERR unasserted and RCADR_PERR unasserted, the current unicast chalkmark bit is copied into the chalkmark bit in the wDBM memory at the location indexed by R_CADR if its multicast chalkmark bit is 0.

DF38 For read cycles with the R_VALID signal unasserted, the current unicast chalkmark bit is not copied into the chalkmark bit in the wDBM memory at the location indexed by R_CADR if its multicast chalkmark bit is 0.

DF39 For read cycles with the RCADR_PERR signal asserted, the current unicast chalkmark bit is not copied into the chalkmark bit in the wDBM memory at the location indexed by R_CADR if its multicast chalkmark bit is 0.

DF40 For read cycles with the RIVC_PERR signal asserted, the current unicast chalkmark bit is not copied into the chalkmark bit in the wDBM memory at the location indexed by R_CADR if its multicast chalkmark bit is 0.

DF41 For read cycles which have R_VALID asserted, RIVC_PERR unasserted and RCADR_PERR unasserted, the current multicast chalkmark bit is copied into the chalkmark bit in the wDBM memory at the location indexed by R_CADR if its multicast chalkmark bit is 1.

DF42 For read cycles with the R_VALID signal unasserted, the current multicast chalkmark bit is not copied into the chalkmark bit in the wDBM memory at the location indexed by R_CADR if its multicast chalkmark bit is 1.

DF43 For read cycles with the RCADR_PERR signal asserted, the current multicast chalkmark bit is not copied into the chalkmark bit in the wDBM memory at the location indexed by R_CADR if its multicast chalkmark bit is 1.

DF44 For read cycles with the RIVC_PERR signal asserted, the current multicast chalkmark bit is not copied into the chalkmark bit in the wDBM memory at the location indexed by R_CADR if its multicast chalkmark bit is 1.

DF45 When the Chalkmark search enable bit in the CDM Control register is set the CDM starts a chalkmark search cycle starting at the wDBM at location 0.

DF46 When the Chalkmark search is enabled, CDM will check the next CADR's wDBM in the search sequence on a Core Memory write cycle with the W_VALID signal unasserted.

DF47 When the Chalkmark search is enabled, CDM will check the next CADR's wDBM in the search sequence on a Core Memory write cycle with the WIVC_PERR signal asserted.

DF48 When the Chalkmark search is enabled, CDM will check the next CADR's wDBM in the search sequence on a Core Memory write cycle with the W_CE_NULL signal asserted.

DF49 When the Chalkmark search is enabled, CDM will not check the next CADR's wDBM in the search sequence on a Core Memory write cycle with the W_VALID signal asserted and the WIVC_PERR and W_CE_NULL signals are unasserted.

DF50 When the Chalkmark search is enabled, CDM will check the next CADR's wDBM in the search sequence on a Core Memory read cycle with the R_VALID signal unasserted.

DF51 When the Chalkmark search is enabled, CDM will check the next CADR's wDBM in the search sequence on a Core Memory read cycle with the RIVC_PERR signal asserted.

DF52 When the Chalkmark search is enabled, CDM will check the next CADR's wDBM in the search sequence on a Core Memory read cycle with the RCADR_PERR signal asserted.

DF53 When the Chalkmark search is enabled, CDM will not check the next CADR's wDBM in the search sequence on a Core Memory read cycle with the R_VALID signal asserted, the RIVC_PERR signal unasserted and the RCADR_PERR signal unasserted.

DF54 The Chalkmark search is not performed when it is not enabled.

DF55 When the search is being performed, the current Multicast chalkmark is compared to the wDBM's chalkmark if the wDBM's Multicast bit is set.

DF56 When the search is being performed, the current Unicast chalkmark is compared to the wDBM's chalkmark if the wDBM's Multicast bit is clear.

DF57 During the search, if a wDBM chalkmark being checked is not the same as its corresponding current chalkmark (determined by the Multicast bit) and all port distribution bits are not zero, the CADR indexing this wDBM will be placed in the Stale CADR register.

DF58 During the search, if a wDBM chalkmark being checked is not the same as its corresponding current chalkmark (determined by the Multicast bit) and all port distribution bits are zero, the CADR indexing this wDBM will not be placed in the Stale CADR register.

DF59 During the search, if a wDBM chalkmark being checked is not the same as its corresponding current chalkmark (determined by the Multicast bit) and all port distribution bits are not zero, the CDM stops cycling through the Chalkmark search at the current CADR.

DF60 During the search, if a wDBM chalkmark being checked is not the same as its corresponding current chalkmark (determined by the Multicast bit) and all port distribution bits are zero, the CDM does not stops cycling through the Chalkmark search at the current CADR.

DF61 During the search, if a wDBM chalkmark being checked is not the same as its corresponding current chalkmark and all port distribution bits are not zero, the GPU is interrupted.

DF62 During the search, if a wDBM chalkmark being checked is not the same as its corresponding current chalkmark and all port distribution bits are zero, the GPU is not interrupted.

DF63 During the search, if a wDBM chalkmark being checked is the same as its corresponding current chalkmark, the search will continue at the next wDBM during the next cycle in which a chalkmark are checked.

DF64 When a stale chalkmark is found and the search halts, the search continues checking wDBMs at the next CADR after that at which a stale chalkmark was found, when the Chalkmark search enable bit is set.

DF65 When a stale chalkmark is found, the CH_ALARM bit in the CDM Status register is asserted.

DF66 When a stale chalkmark has not been found since the last time the CH_ALARM in the CDM Status register bit was de-asserted, the CH_ALARM bit is not asserted.

DF67 When asserted, the CH_ALARM bit in the CDM Status register is de-asserted when the CLR_CH_ALARM bit in the CDM Control register is asserted.

DF68 The CH_ALARM bit in the CDM Status register is not asserted after reset.

DF69 The GPU is interrupt with a chalkmark search complete interrupt when the chalkmarks for all 16K CADRs have been checked.

DB CDM Boundary Attributes

DB1 The FA signal is not asserted when a Core Memory read occurs at a R_CADR whose port distribution bits are already all clear in its wDBM.

DB2 The DBM Space RAM can hold 8K DBMs.

DB3 The wDBM Space RAM can hold 16K wDBMs.

DP CDM Performance Attributes

DP1 The CDM can assert FA at full speed when 16 Core Memory read cycles capable of causing FA to be asserted are occurring per cell cycle.

DP2 The CDM can assert FA fast enough when the CIF is producing read cycles in each of its read slots that are capable of causing FA to be asserted.

DP3 The CDM is capable of modifying wDBMs at full speed when 16 Core Memory reads and 16 Core Memory writes are occurring per cell cycle.

DP4 The CDM is capable of modifying wDBMs when the CIF is performing Core Memory reads and writes at its full speed (one read and one write per cell cycle).

DP5 The Chalkmark search can catch all stale chalkmarks when it is operating at full speed (i.e. when all 16 Core Memory reads and 16 Core Memory writes per cell cycle are either nulled or have Cell Control Bus (CCB) parity errors).

DP6 The Chalkmark search can catch all stale chalkmarks when the CIF is performing Core Memory reads and writes at its full speed (i.e. one read and one write per cell cycle).

5.1.E Core Memory Resource Manager Attributes

The CRM is responsible for monitoring the number of cells each port places in the Core Memory (via Buffer Fill counters) and exerting backpressure on a port when certain Core Memory resource thresholds are exceeded. Please refer to the following diagram entitled "Figure 7 CRM Block Diagram" for more details on the CRM.

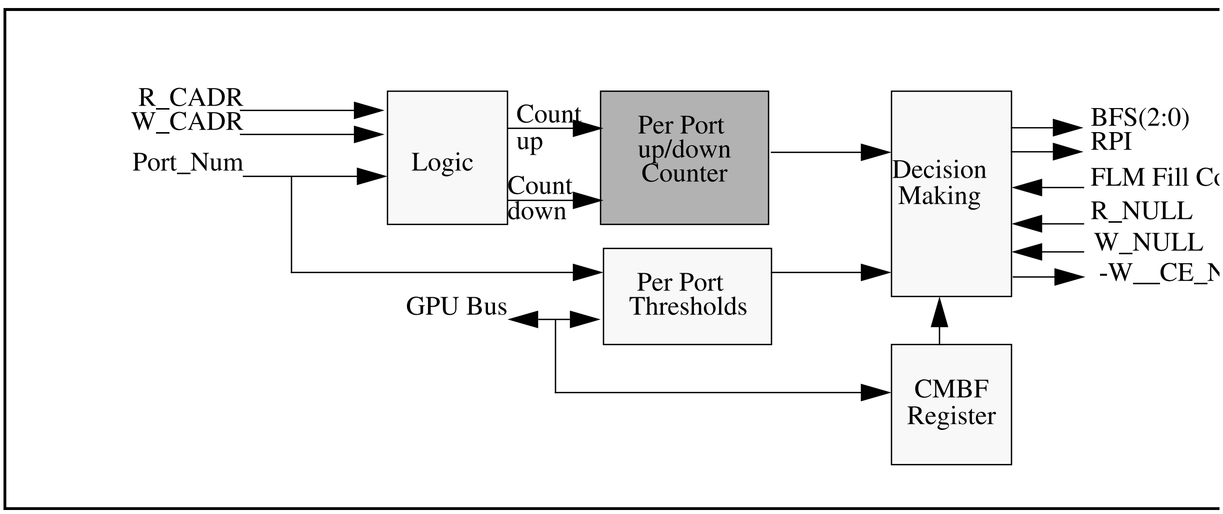

Figure 7 CRM Block Diagram

The following sections outline the CRM attributes requiring testing.

EF <u>CRM Functional Attributes</u>

Registers

EF1 Per port Buffer Fill (BF) threshold registers read/write integrity.

EF2 Core Memory threshold register read/write integrity.

EF3 Port Number Register read/write integrity.

EF4 Per port BF counter read integrity.

Initialization

EF5 The BF counts are 0 for each port after initialization.

Buffer Fill Counters

EF6 For write cycles which have W_VALID asserted and WIVC_PERR and W_CE_NULL unasserted, the BF count for the port identified by W_CSID is incremented.

EF7 For write cycles with the W_VALID signal unasserted, the BF count for the port identified by W_CSID is not changed.

EF8 For write cycles with the WIVC_PERR signal asserted, the BF count for the port identified by W_CSID is not changed.

EF9 For write cycles with the W_CE_NULL signal asserted, the BF count for the port identified by W_CSID is not changed.

EF10 For read cycles which have R_VALID asserted, RIVC_PERR unasserted and RCADR_PERR unasserted and result in the FA signal being asserted, the BF count for the port identified by R_CSID is decremented.

EF11 For read cycles with the R_VALID signal unasserted, the BF count for the port identified by R_CSID is not changed.

EF12 For read cycles with the RIVC_PERR signal asserted, the BF count for the port identified by R_CSID is not changed.

EF13 For read cycles with the RCADR_PERR signal asserted, the BF count for the port identified by R_CSID is not changed.

EF14 For read cycles which do not result in the FA sinal being asserted, the BF count for the port identified by R_CSID is not changed.

EF15 The port BF counnter register contains the port BF for the port indicated by the port number register.

Buffer Fill Thresholds

The CRM has four programmable BF thresholds for each port. There is one RPI and three BF thresholds. The three BF thresholds will be referred to as the upper BF threshold, the middle BF threshold and the lower BF threshold. The three BF thresholds are encoded in to two bits. The RPI and BF status can not be read by the GPU, these bits are instead time-multiplexed on the backplane driving the value of a ports BF status and RPI signals during that ports CCB Time-slot.

EF16 The CRM increases a port's effective BF count by the value of the global Core Memory BF (CMBF) threshold overage (CMBF - CMBF threshold), when it is positive, for the purpose of comparison with the BF and RPI thresholds.

EF17 When the CMBF overage is not positive and a port's BF count is less than its lower BF threshold, the value of the BF Status bits presented to the port is 00.

EF18 When the CMBF overage is not positive and a port's BF count is greater than or equal to the lower BF threshold and less than the middle BF threshold, the value of the BF Status bits presented to the port is 01.

EF19 When the CMBF overage is not positive and a port's BF count is greater than or equal to the middle BF threshold and less than the upper BF threshold, the value of the BF Status bits presented to the port is 10.

EF20 When the CMBF overage is not positive and a port's BF count is greater than or equal to the upper BF threshold, the value of the BF Status bits presented to the port is 11.

EF21 When the CMBF overage is not positive and a port's BF count is less than its RPI threshold, its RPI signal presented to the port is not asserted.

EF22 When the CMBF overage is not positive and a port's BF count is more than or equal to its RPI threshold, its RPI signal presented to the port is asserted.

EF23 When the CMBF overage is positive and the sum of port's BF count and the CMBF overage is less than its lower BF threshold, the value of the BF Status bits presented to the port is 00.

EF24 When the CMBF overage is positive and the sum of port's BF count and the CMBF overage is greater than or equal to the lower BF threshold and less than the middle BF threshold, the value of the BF Status bits presented to the port is 01.

EF25 When the CMBF overage is positive and the sum of port's BF count and the CMBF overage is greater than or equal to the middle BF threshold and less than the upper BF threshold, the value of the BF Status bits presented to the port is 10.

EF26 When the CMBF overage is positive and the sum of port's BF count and the CMBF overage is greater than or equal to the upper BF threshold, the value of the BF Status bits presented to the port is 11.

EF27 When the CMBF overage is positive and the sum of port's BF count and the CMBF overage is less than its RPI threshold, its RPI signal is not asserted.

EF28 When the CMBF overage is positive and the sum of port's BF count and the CMBF overage is more than or equal to its RPI threshold, its RPI signal is asserted.

Core Memory Protection

EF29 When a port has exceeded its highest threshold, which may be its upper BF threshold or its RPI threshold (whichever is greatest), the CRM asserts the W_CE_NULL signal nullifying the port's write cycle.

EF30 When a port has not exceeded its highest threshold, which may be its upper BF threshold or its RPI threshold (whichever is greatest), the CRM does not asserts the W_CE_NULL signal nullifying the port's write cycle.

EF31 When the SCU has exceeded its highest threshold, which may be its upper BF threshold or its RPI threshold (whichever is greatest), the CRM asserts the W_CE_NULL signal nullifying its write cycle.

EF32 When the SCU has not exceeded its highest threshold, which may be its upper BF threshold or its RPI threshold (whichever is greatest), the CRM does not asserts the W_CE_NULL signal nullifying its write cycle.

EB <u>CRM Boundary Attributes</u>

EB1 For read cycles which have R_VALID asserted, RIVC_PERR unasserted, RCADR_PERR unasserted and result in FA being asserted, the Buffer Fill (BF) count for the port identified by R_CSID does not change when it is already 0.

EB2 For write cycles which have W_VALID asserted and WIVC_PERR and W_CE_NULL unasserted, the Buffer Fill (BF) count for the port identified by W_CSID does not increment when it is already at its maximum value(4K - 1).

EP <u>CRM Performance Attributes</u>

EP1 The port BF counters increment at full speed when its corresponding port is performing one Core Memory write per cell cycle.

EP2 The port BF counters decrement at full speed when 16 Core Memory reads are being performed per cell cycle all of which extract cells corresponding to the same port BF counter.

EP3 The port BF counters decrement when the CIF is performing Core Memory reads at full speed (one per cell cycle) to extract cells corresponding to the same port BF counters.

5.1.F Cell Time Counter Attributes

The CTC provides an Interval Timer (IT) which is a programmable event (IVC based) timer as well as a Cell Period Timer which continually increments once per cell period.The following sections outline the CTC attributes requiring testing.

FF <u>CTC Functional Attributes</u>

Register Integrity

FF1 Cell Period Timer read integrity.

FF2 Interval Timer (IT) read integrity.

FF3 IT Start Register read/write integrity.

FF4 IT Stop Register read/write integrity.

Cell Period Timer

FF5 The Cell Period Timer increments once every cell cycle.

Interval Timer

FF6 The IT is initialized to 0 after reset.

FF7 The IT starts counting from 0 when the value of the R_IVC on the backplane is the same as the value of the IVC in the Start Register and the Start Register Mode bit in the IT Start register is 0 (indicating start should be on R_IVC). The R_VALID signal must also be asserted and the RIVC_PERR and RCADR_PERR signals are unasserted.

FF8 The IT does not start counting on a cell read cycle with a R_IVC which does not match the IVC in the start register.

FF9 The IT does not start counting on a R_IVC when the R_VALID signal is not asserted.

FF10 The IT does not start counting on a R_IVC when the RIVC_PERR signal is asserted.

FF11 The IT does not start counting on a R_IVC when the RCADR_PERR signal is asserted.

FF12 The IT does not start counting on a cell write cycle with a W_IVC matching the IVC in the start register when the Start Register Mode bit in the IT Start register is 0 (indicating start should be on R_IVC).

FF13 The IT starts counting from 0 when the value of the W_IVC on the backplane is the same as the value of the IVC in the Start Register and the Start Register Mode bit in the IT Start register is 1 (indicating start should be on W_IVC). For this, the W_VALID signal must be asserted and the WIVC_PERR and W_CE_NULL signals must be unasserted.

FF14 The IT does not start counting on a cell write cycle with a W_IVC which does not match the IVC in the start register.

FF15 The IT does not start counting on a W_IVC when the W_VALID signal is not asserted.

FF16 The IT does not start counting on a W_IVC when the WIVC_PERR signal is asserted.

FF17 The IT does not start counting on a W_IVC when the W_CE_NULL signal is asserted.

FF18 The IT does not start counting on a cell read cycle with a R_IVC matching the IVC in the start register when the Start Register Mode bit in the IT Start register is 1 (indicating start should be on W_IVC).

FF19 The IT stops counting when the value of the R_IVC on the backplane is the same as the value of the IVC in the Stop Register and the Stop Register Mode bit in the IT Stop register is 0 (indicating stop should be on R_IVC). The R_VALID signal must also be asserted and the RIVC_PERR and RCADR_PERR signals are unasserted.

FF20 The IT does not stop counting on a cell read cycle with a R_IVC which does not match the IVC in the start register.

FF21 The IT does not stop counting on a R_IVC when the R_VALID signal is not asserted.

FF22 The IT does not stop counting on a R_IVC when the RIVC_PERR signal is asserted.

FF23 The IT does not stop counting on a R_IVC when the RCADR_PERR signal is asserted.

FF24 The IT does not stop counting on a cell write cycle with a W_IVC matching the IVC in the stop register when the Stop Register Mode bit in the IT Stop register is 0 (indicating stop should be on R_IVC).

FF25 The IT stops counting when the value of the W_IVC on the backplane is the same as the value of the IVC in the Stop Register and the Stop Register Mode bit in the IT Stop register is 1 (indicating stop should be on W_IVC). For this, the W_VALID signal must be asserted and the WIVC_PERR and W_CE_NULL signals must be unasserted.

FF26 The IT does not stop counting on a cell write cycle with a W_IVC which does not match the IVC in the start register.

FF27 The IT does not stop counting on a W_IVC when the W_VALID signal is not asserted.

FF28 The IT does not stop counting on a W_IVC when the WIVC_PERR signal is asserted.

FF29 The IT does not stop counting on a W_IVC when the W_CE_NULL signal is asserted.

FF30 The IT does not stop counting on a cell read cycle with a R_IVC matching the IVC in the stop register when the Stop Register Mode bit in the IT Stop register is 1 (indicating stop should be on W_IVC).

FF31 Once the IT has started counting, it increments once per cell cycle.

FF32 The CTC interrupts the GPU when the IT stops counting.

FB <u>CTC Boundary Attributes</u>

FB1 The Cell Period Timer wraps around to zero when it reaches its maximum value (64K - 1).

FB2 The IT holds at 16M - 1 when it reaches this maximum value.

FB3 The CTC interrupts the GPU when the IT reaches its maximum value of 16M - 1 and the interrupt enable bit is set in the control register.

FB4 The IT indicates an interval one cell cycle long when its stop condition is encountered only one cell cycle after its start condition.

FB5 The CTC interrupts the GPU when the IT stops counting and the interval measured was only one cell cycle.

FP <u>CTC Performance Attributes</u>

Not applicable.

5.1.G Processor Bus Attributes (P-Bus)

The P-Bus is responsible for SCU interrupt priority encoding, Core Memory parity error monitoring, PM interrupt monitoring, PM reset control and alarms status. The following sections outline the P-Bus attributes requiring testing.

GF <u>P-Bus Functional Attributes</u>

Register Integrity

GF1 Core Memory error counter read integrity.

GF2 PM reset register read/write integrity.

GF3 Alarms register read integrity.

GF4 PM interrupt register read integrity.

GF5 Program Enable register write integrity.

GF6 Altera programming register.

GF7 Lattice programming register.

Core Memory Parity Error Counter

GF8 The Core Memory parity error counter is 0 after reset.

GF9 The Core Memory parity error counter increments when the Core Memory parity error signal indicates that a parity error occurred and the Core Memory parity error counter clear bit is de-asserted.

GF10 The Core Memory parity error counter does not increment when the Core Memory parity error signal does not indicates that a parity error occurred.

GF11 The Core Memory parity error counter does not increment when the Core Memory parity error counter clear bit is asserted.

GF12 The GPU is interrupted with a Core Memory parity error interrupt when the counter reaches a pre-determined value.

GF13 The GPU is not interrupted with a Core Memory parity error interrupt when the counter is not at a pre-determined value.

Switch Alarms Register

GF14 The AC fail bit in the alarm register is set if the AC fail signal from the TC is asserted.

GF15 The GPU is interrupted with an AC fail alarm interrupt when the AC fail signal from the TC is asserted.

GF16 The GPU is not interrupted with an AC fail alarm interrupt when the AC fail signal from the TC is not asserted.

GF17 The AC fail bit in the alarm register is not asserted if the AC fail signal from the TC is not asserted.

GF18 The AC fail bit in the alarm register is cleared when the AC fail signal from the TC is de-asserted.

GF19 The Fan fail bit in the alarm register is set if the Fan fail signal from the TC is asserted.

GF20 The GPU is interrupted with a Fan fail alarm interrupt when the Fan fail signal from the TC is asserted.

GF21 The GPU is not interrupted with a Fan fail alarm interrupt when the Fan fail signal from the TC is not asserted.

GF22 The Fan fail bit in the alarm register is not asserted if the Fan fail signal from the TC is not asserted.

GF23 The Fan fail bit in the alarm register is cleared when the Fan fail signal from the TC is de-asserted.

GF24 The Power Supply fail bit in the alarm register is set if the Power Supply fail signal from the TC is asserted.

GF25 The GPU is interrupted with a Power Supply fail alarm interrupt when the Power Supply fail signal from the TC is asserted.

GF26 The GPU is not interrupted with a Power Supply fail alarm interrupt when the Power Supply fail signal from the TC is not asserted.

GF27 The Power Supply fail bit in the alarm register is not asserted if the Power Supply fail signal from the TC is not asserted.

GF28 The Power Supply fail bit in the alarm register is cleared when the Power Supply fail signal from the TC is de-asserted.

GF29 The Core Memory clock fail bit in the alarm register is set if the Core Memory clock fail signal from the Core Memory is asserted.

GF30 The GPU is interrupted with a Core Memory clock fail alarm interrupt when the Core Memory clock fail signal from the TC is asserted.

GF31 The GPU is not interrupted with a Core Memory clock fail alarm interrupt when the Core Memory clock fail signal from the TC is not asserted.

GF32 The Core Memory clock fail bit in the alarm register is not asserted if the Core Memory clock fail signal from the TC is not asserted.

GF33 The Core Memory clock fail bit in the alarm register is cleared when the Core Memory clock fail signal from the Core Memory is de-asserted.

GF34 The Core Memory parity error bit is set when the Core Memory parity error counter reaches a pre-determined value.

GF35 The Core Memory parity error bit is cleared when the GPU performs a write to the Core Memory parity error clear register.

GF36 The Cell Bus-tap error bit is set when the Cell Bus-tap error counter reaches a pre-determined value.

GF37 The Cell Bus-tap error bit is cleared when the GPU performs a write to the Cell Bus-tap error clear register.

Port Module (PM) Reset Control

GF38 Writing a PM's corresponding bit in the Reset Register to 1 causes a reset pulse to occur on the TDM reset line during that PM's CCB slot until the PMs reset bit is changed to 0.

GF39 A PM does not receive a reset pulse during its CCB slot when its corresponding bit in the Reset Register is 0.

GF40 All bits in the PM Reset Register are initialized to zero.

PM Interrupt Monitoring

GF41 All bits in the PM Interrupt register are held clear when the PM Interrupt Rest bit in the PBUS's Interrupt/Alarms Control register is set.

GF42 A PM's corresponding interrupt bit is set in the Interrupt Register while the PM is asserting the TDM PM interrupt signal (on the backplane) during its CCB slot and the PM Interrupt Rest bit in the PBUS's Interrupt/Alarms Control register is set.

GF43 A PM's corresponding interrupt bit is not set in the Interrupt Register when the PM is not asserting the TDM PM interrupt signal (on the backplane) during its CCB slot.

GF44 The SCU is interrupted with a PM interrupt as long as any PM interrupt register bit is set

GF45 The SCU is not interrupted with a PM interrupt if all the bits in the PM Interrupt register are clear

GF46 All bits in the PM Interrupt Register are intiailized to zero.

SCU Interrupt Priority Encoding

GF47 Performs priority encoding of up to 32 interrupt sources within the SCU into five bits which are presented to the i960 processor as XINT4-0. (This produces an 8 bit vector whose five most significant bits are XINT4-0 and three least significant bits are 010.)

GB <u>P-Bus Boundary Attributes</u>

GB1 The Core Memory parity error counter is reset to 0 when the Core Memory parity error counter clear bit is asserted.

GB2 The Core Memory parity error counter wraps around to 0 when it is at its maximum value of 15 and the Core Memory parity error signal is asserted.

GP P-Bus Performance Attributes

Not applicable.

5.1.H Bus-tap Controller Attributes

The BTC is responsible for providing control over the CBT as well as checking its integrity. The following sections outline the Bus-tap Controller attributes requiring testing.

HF BTC Functional Attributes

CBT Control

HF1 Bus Tap Setup register read/write integrity.

HF2 BTC Setup[3:0] in the BTC Setup register selects which data path is routed to the CBT.

HF3 Asserting BTC Setup[4] in the BTC Setup register places BTC[3:0] in BTC control mode.

HF4 De-asserting BTC Setup[4] places BTC Setup[3:0] in Lattice/Altera device programming mode. (This is multiplexed on the BTC Setup[3:0] on the backplane. This enables programming of devices on the common equipment and Core Memory cards.)

CBT Integrity

HF5 The CBT error counter is 0 after reset.

HF6 The CBT error counter increments when the TC asserts the CBT CRC8 error signal, the CBT error enable bit in the BTC control register is asserted and the CBT error counter clear bit is de-asserted.

HF7 The CBT error counter does not increment when the TC does not assert the CBT CRC8 error signal.

HF8 The CBT error counter does not increment when the CBT error enable bit in the BTC control register is not asserted.

HF9 The CBT error counter does not increment when the CBT error counter clear bit is asserted.

HF10 The GPU is interrupted with a CBT error interrupt when the CBT error counter reaches a pre-determined value.

HF11 The GPU is not interrupted with a CBT error interrupt when the counter is not at a pre-determined value.

HB BTC Boundary Attributes

HB1 The CBT error counter is reset to 0 when the CBT error counter clear bit is asserted.

HB2 The CBT error counter wraps around to 0 when it is at its maximum value (?) and the CBT CRC8 error signal is asserted.

HP BTC Performance Attributes

Not applicable

5.1.I Core Memory Attributes

The Core Memory is responsible for buffering cells until they can be transmitted by their destination port or are reclaimed by the SCM. The following sections outline the Core Memory attributes requiring testing.

IF Core Memory Functional Attributes

IF1 Core Memory cell data integrity.

IF2 Cells can be written to any cell address.

IF3 Cells can be read from any cell address.

IF4 The Core Memory provides the cell data path selected by BTC[3:0] (when BTC[4] specifies BTC mode) on the 16 Cell Bus Access (CBA) backplane pins.

IF5 The Core Memory parity error signal is asserted when a CADR parity error is detected.

IB Core Memory Boundary Attributes

IB1 The Core Memory can store the maximum 16K cells of 32 16-bit words.

IP Core Memory Performance Attributes

IP1 An 11.79 Gbit/s bandwidth is supported such that 16 ports can each perform one cell write and one cell read per cell cycle, with each read and write transferring 32 16-bit words.

5.1.J Terminator Card (TC) Attributes

The TC is responsible for BTL and TTL backplane signal termination and conversion, providing an interface between the SCU's BTC and the CBA on the Core Memory and providing an interface between the alarms circuitry of the switch cabinet and the SCU. Each switch is equipped with two terminator cards, one at each end of the backplane. The TC adjacent to slot 0 is referred to as TC0 while the TC adjacent to slot 15 is referred to as TC1. The following sections outline the TC attributes that require testing.

JF TC Functional Attributes

Signal Termination and Conversion

JF1 Provides BTL signal termination of all CCB signals by pulling them up to 2.1V through 39 ohm resistors. The CCB signals are made up of R_IVC, W_IVC, R_CADR, W_CADR, R_ID, W_ID, R_LOG, W_LOG, R_VALID, W_VALID, W_CE_NULL, FC, CSID, TDMRST, RPI, BF_STAT, TDM_IRQ...

JF2 Provides TTL signal termination of the P-Bus (processor address, data and control signals) by pulling them up to 2.9V through 110 ohm resistors.

JF3 Each TC performs BTL-TTL level conversion for half of the W_CADR and half of the R_CADR address bus for the Core Memory.

JF4 Provides a TTL level W_DISABLE signal for the Core Memory and asserts it if W_CE_NULL BTL is asserted and W_VALID is de-asserted.

Cell Bus-Tap Interface

JF5 Drives (7-8) encoded data of the selected cell data path provided by the Core Memory onto 16 backplane pins.

JF6 Performs a CRC8 integrity check on the data of the selected data path (after performing 8-7 decoding) and signals a CRC8 error to the SCU if the CRC8 calculated for the cell does not match that found in the cell.

JF7 Performs a CRC8 integrity check on the data of the selected data path (after performing 8-7 decoding) but does not signal a CRC8 error to the SCU if the CRC8 calculated for the cell matches that found in the cell.

Alarms

JF8 Buffers the fan failure signal coming from the switch cooling unit and drives it back to the SCU.

JF9 Buffers the power supply failure signal coming from the switch power supplies and drives it back to the SCU.

JF10 If the TC is TC0, it receives the major alarm signal coming from the SCU and deactivates the major alarm relay if asserted by the SCU.

JF11 If the TC is TC15 it activates the minor alarm relay contact if either the fan failure or the power supply failure signals are asserted.

JB TC Boundary Attributes

JB1

JB2

JP TC Performance Attributes

JP1

JP2

5.2 Test Cases

The test cases described in the following sections test the SCU, Core Memory and TC attributes listed in previous section "Functional Attributes Requiring Testing" . The tests have been divided into the following groups:

- Memory and Register Tests - Start-up Diagnostics and H/W Verification;
- Basic Functionality Tests - Start-up Diagnostics and H/W Verification;
- Basic Functionality Tests - H/W Verification Requiring Manual Intervention;
- Performance and Integration Tests - Start-up Diagnostics and H/W Verification.

The tests have been grouped and listed in the order in which they are performed. The first three groups of tests have been partitioned in a way so as to minimize the inter-dependance of the various blocks during testing. The ordering of the tests attempts to minimize the dependence on attributes that have not already been tested. These first three groups of tests target the functional and boundary attributes listed earlier in this document.

The third group of tests, "Basic Functionality Tests - Hardware Verification Requiring Manual Intervention" are only performed once per board during hardware verification and require special H/W setup.

The fourth group of tests, "Performance and Block Integration Tests - Start-up Diagnostics and H/W Verification" cover the integration of attributes from various SCU blocks as well as target the performance attributes.

<u>Memory and Register Tests - Start-up Diagnostics and H/W Verification</u>

5.2.1 CIF Register and Reset Test

This test verifies that the CIF FIFO status and errors flags are all 0 after reset and also tests certain CIF registers.

5.2.2 CIF Shared Buffer Memory Test

This test verifies GPU access to the Shared Buffer and its integrity. In particular, it tests for stuck bits, leakage, address decoding errors and address and data line shorts.

5.2.3 W_IVC Lookup Table Memory Test

This test verifies GPU access to the W_IVC Lookup Table Memory and its integrity. In particular, it tests for or stuck bits, leakage, address decoding errors and address and data line shorts.

5.2.4 R_IVC Lookup Table Memory Test

This test verifies GPU access to the R_IVC Lookup Table Memory and its integrity. In particular, it tests for or stuck bits, leakage, address decoding errors and address and data line shorts.

5.2.5 Partial CRC Register and Reset Test

5.2.6 FLM Register and Reset Test

The main purpose of this test is to verify the read/write integrity of the FLM Control register. It also ensures that the value of the CMBF count is zero, the FLM Empty bit is set and the FLM Halt bit is set after the FLM has been reset.

5.2.7 FLM Free List RAM and Status Test

The main purposes of this test are to verify Free List RAM integrity. It also tests FLM initialization and the FLM Empty and Full bits. (Warning: This test relies on the highest priority W_IVC FIFO to test the FLM RAM but this FIFO is not fully tested until the "CIF IVC FIFO Test" .)

5.2.8 CDM Register Test

The main purpose of this test is to verify the CDM Control Register and Current Chalkmark Register integrity and that the CDM Status Register is initialized correctly after reset.

5.2.9 DBM Configure and Capture Test

The main purpose of this test is to verify that the DBM can be configured correctly, DBMs are copied to the appropriate wDBMs on write cycles and that both DBMs and wDBMs can be captured. (This test is necessary before the DBM and wDBM RAM tests since they use DBM configuration and capturing.)

5.2.10 DBM Memory Test

The main purpose of this test is to verify the DBM Space RAM integrity. This test also verifies that DBM space initialization can be performed correctly.

5.2.11 wDBM Memory Test

The main purpose of this test is to verify the wDBM Space RAM integrity. This test also verifies that DBMs are correctly transferred to the appropriate wDBM during Core Memory writey cycles.

5.2.12 CRM Registers and Reset Test

The main purpose of this test is to verify the integrity of the CRM's BF Threshold registers, Port Number register and Core Threshold register. It also verifies that the per port BF counters are intiailized to zero after reset.

5.2.13 CTC Register and Reset Test

The main purpose of this test is to verify the read/write integrity of the CTC's IT Start and IT Stop registers and to verifiy that IT timer is initialized to zero after reset.

5.2.14 P-Bus Register and Reset Test

The main purpose of this test is to verify the read/write integrity of the P-Bus' PM Reset register, that the PM Interrupt and PM Reset registers are initialized to xero and that the Core Memory parity counter is zero after reset.

5.2.15 BTC Control Register Integrity and Reset Test

This test verifies the read/write integrity of the BTC Setup register and that the CBT error counter is 0 after reset. (WARNING: This test must not set the bit corresponding to BTC[4] on the backplane such that it places BTC[3:0] into Lattice/Altera device programming mode.)

<u>Basic Functionality Tests - Start-up Diagnostics and Hardware Verification</u>

5.2.16 CIF IVC FIFO Test

The main purpose of this test is to verify that both WIVC FIFOs and the RIVC FIFO capture data correctly based upon their configuration in the WIVC and RIVC Lookup Tables as well as verify the data integrity of the FIFOs memory and data paths. This test also verifies the status bits associated with each FIFO and its boundary conditions and the ability of the SCU to interrupt the GPU when each FIFO reaches the mid-range and almost-full levels. (Warning: This test assumes that the CRM can assert W_CE_NULL during the CIF cell write slot when the SCM's Upper BF Threshold and RPI threshold are both 0 and the CIF performs a cell write.)

5.2.17 CIF Data Path Integrity and Control and Status Register Test

Perform several cell writes and reads with several patterns in the cells to guarantee that the data written is the same as that received. This test should guarantee there are no data bit shorts and should also verify that the byte ordering written is the same as that read. This test also verifies that several bits in the Control register clear automatically after write and read cycles and that data is not transferred to or from the Core Memory on cycles which are not valid. It also verifies that certain backplane integrity bits in the CIF Status register are not set when they should not be.

5.2.18 CIF CRC8 Generation/Checking Test

This test checks that the CIF's CRC8 checker/generator correctly detect CRC8 errors on cells when they occur and do not detect errors when they are not present. It also ensures that the CRC8 interrupt works correctly and the CRC8 check/generate bit in the CIF Control register automatically clears after cell read and write cycles.

5.2.19 CDM Monitoring and FA Signalling Test

This test verifies that DBM modification and FA signalling are performed correctly under various cell read and write conditions.

5.2.20 FLM Basic Test

This test verifies that the FLM provides W_CADRs to the backplane and increments/decrements the CMBF count only under the proper conditions. It also verifies tfunctioning of the FLM Halt bit.

5.2.21 FLM Boundary and Integrity Test

This test verifies the correct functionality of the CMBF count under normal and boundary boundary conditions as well as verifying the functionality of the FLW, FLF and FL Empty signals.

5.2.22 CDM Chalkmark Update Test

This test verifies that the CDM correctly updates a CADR's wDBM with the correct current chalkmark (unicast or multicast) but only under the correct cell read and write circumstances.

5.2.23 CDM Chalkmark Search Test

This test verifies that the CDM correctly finds all CADRs with stale chalkmarks which fit the stale chalkmark criteria. It verifies that the correct current chalkmark is used for comparison (multicast or unicast) and that CADRs are not reported stale when a cell is not being stored at the CADR. It also tests the mechanism by which the CDM reports the stale CADRs to the GPU.

5.2.24 CRM Port Usage Count Test

On behalf of each port (using its CSID), this test performs a series of valid and invalid read and write cycles in order to verify that the port counters increment/decrement on valid write/read cycles only.

5.2.25 CRM Core Memory Protection Test

This test is similar to the previous one except more cells from other ports will also be placed in the Core Memory so that artificial BF promotion and Core Memory protection can be tested. This test only verifies that SCU cell writes are nulled when the SCU's highest BF threshold is exceeded.

5.2.26 CTC Cell Period Timer Test

This test ensures that the CTC's Cell Period Timer continually increments once per cell period wrapping around to 0 when it reaches 64K-1.

5.2.27 CTC IT Starting and Maximum Interval Test

This test verifies that the IT only start counting at 0 on valid cell write and read cycles with an IVC for which it was programmed to start, increments once per cell period and holds at it maximum value of 16M-1 when it is not stopped. This test also verifies that the IT will not start when any of the above conditions is not met and that it interrupts the GPU when it reaches it maximum value.

5.2.28 CTC IT Timing and Minimal Interval Test

This test verifies that the IT stops counting only on valid cell write and read cycles with an IVC for which it was programmed to stop, measures timed intervals correctly and interrupts the GPU when it is done timing an interval. It also verifies that the CTC can correctly time intervals that are only 1 cell period long.

5.2.29 CBT Integrity Checking Test

This test verifies the functioning of the BTC control of the CBT (to some extent), the CBT integrity checking on the TC and the BTC's CBT error counter and interrupt mechanism. It also tests the CBT error bit in the P-Bus's Alarms register.

5.2.30 Core Memory Test

This test is required to test the Core Memory for stuck bits, leakage, address decoding errors and address and data line shorts from a black box point of view. It also builds a list of cell addresses which are faulty whenever possible and attempts to isolate the 16-bit word with in the Core Memory that was found to be faulty when the test fails. This test can not determine whether the failure occurred because of a faulty memory chip because the error can also be caused by the Core Memory's commutation logic and data paths. Any error information collected by this test however can still provide some isolation information in order to help debug the cause of and errors. The byte patterns used in this test are restricted to the set of bytes that are left after 7 to 8 encoding has been performed on a cell. Because of this, the patterns used to test this memory must be chosen carefully.

<u>Basic Functionality Tests - Hardware Verification Requiring Manual Intervention</u>

5.2.31 WCADR_PERR Test

This test verifies that all tests in sections 5.2.1 through 5.2.28 can be performed again with the WCADR_PERR backplane signal asserted. This verifies that this signal does not have any affect on the operation of the SCU. (It requires that WIVC_PERR on the Backplane is grounded.)

5.2.32 CIF Backplane Parity Error Test

This test verifies that the CIF does not transfer cells to the Core Memory or capture data in the W_IVC FIFOs during cell write cycles with WIVC_PERR asserted. It also verifies that the CIF does not transfer cells from the Core Memory or capture data in the R_IVC FIFO during cell write cycles with either RIVC_PERR or RCADR_PERR asserted. This test also checks that the corresponding bits in the CIF Status register are set when the above parity errors occur. (It requires that WIVC_PERR, RIVC_PERR and RCADR_PERR on the Backplane are grounded separately.)

5.2.33 FLM Backplane Parity Error Test

This test verifies that the CMBF counter in the FLM does not change on cell write cycles with WIVC_PERR asserted or on cell read cycles with RCADR_PERR or RIVC_PERR asserted. (It requires that WIVC_PERR, RIVC_PERR and RCADR_PERR on the Backplane are grounded separately.)

5.2.34 CDM Backplane Parity Error Test

This test verifies that the CDM does not modify wDBMs or update chalkmarks on cell write cycles with WIVC_PERR asserted or on cell read cycles with RIVC_PERR or RCADR_PERR asserted. (It requires that WIVC_PERR, RIVC_PERR and RCADR_PERR on the Backplane are grounded separately.)

5.2.35 CRM Port Usage Count Test with Backplane Parity Errors

On behalf of each port (using its CSID), this test performs a series of valid read and write cycles in order to verify that the port counters do not increment/decrement when the WIVC_PERR, RIVC_PERR or RCADR_PERR signals are asserted. (It requires that WIVC_PERR, RIVC_PERR and RCADR_PERR on the Backplane are grounded separately.)

5.2.36 CRM BF Threshold Test

On behalf of each port (using its CSID), this test performs a series of valid writes and then read cycles in order to verify that the BF threshold bits, RPI and W_CE_NULL presented to a port during its CCB slot are correct for various port usage counter values. (This test requires the use of a scope or logic analyzer and the toggling of a dip switch on the GPU to advance the test from one stage to the next.)

5.2.37 CTC IT Backplane Parity Error Test.

This test verifies that the IT does not start or stop on cell write cycles with WIVC_PERR asserted or cell read cycles with RIVC_PERR or RCADR_PERR asserted. (This test requires the use of a scope or logic analyzer and the toggling of a dip switch on the GPU to advance the test from one stage to the next.)

5.2.38 Core Memory Parity Error Monitoring Test

This test verifies that the P-Bus's Core Memory parity error counter is functioning correctly and that the GPU is interrupted when the counter reaches a pre-determined value and the appropriate bit in the Alarms register is set. (This test requires that the Core Memory parity error backplane signal be asserted manually.)

5.2.39 Switch Alarms Test

This test verifies that the P-Bus is correctly recognizing the AC fail, Fan fail, Power supply fail and Core Memory clock fail alarms, interrupting the GPU with a distinct interrupt for each and setting the appropriate bits in the alarms register are set. (This test requires that each of the four alarms above is manually asserted. It does not require a test function but does require that an ISR for each of the alarms tested that notifies the tester, via the VxWorks shell, that a particular interrupt has occurred.)

5.2.40 PM Reset Register Test

This test verifies that the SCU asserts reset pulses correctly on the TDM backplane PM reset signal according to the contents of the PM Reset register. (This test requires the use of a scope to examine the PM reset signal on the backplane.)

5.2.41 PM Interrupt Test

This test verifies the integrity of the PM Interrupt register and the PM interrupt going to the GPU. (This test requires that the PM Reset signal on the backplane is looped back to the PM Interrupt in order to drive and control the PM Interrupt signal.)

5.2.42 CIF W_LOG/W_ID/R_LOG/R_ID Test

This test performs a series of cell writes and reads while cycling through all the possible _LOG and _ID values to verify that the CIF correctly drives these buses with the values written to the W_LOG, W_ID, R_LOG and R_ID registers. (It requires the use of a logic analyzer to capture data driven on the above buses.)

5.2.43 CIF Encoding/Decoding Test

This test performs cell writes and then reads to verify that 7-8 encoding and 8-7 decoding function correctly. (It requires the use of a logic analyzer capturing CIF data as it leaves the encoder, enters the decoder and leaves the decoder.)

5.2.44 CIF Idle Cell Exchange Test

This test verifies that the CIF exchanges idle cells with the Core Memory when it is not performing cell write or read cycles during its CCB slot. (This test requires the use a logic analyzer to capture the data passing between the CIF and Core Memory during the CIF's CCB slot.)

Performance and Block Integration Tests - Start-up Diagnostics and H/W Verification

Trevor: This is the section that I was hoping you could help out in. When I started it I only intended to try to get by with just one function but you may want to change that. The attributes that were to be covered here are listed toward the end of Table 2. If you do split this up into multiple tests please adjust the table accordingly. These test will take up a separate module and will require a separate section in the DD portion of the document.

5.2.45 Performance and SCU Block Integration Test

This test verifies that the blocks of the SCU function and interact together correctly when the CIF is performing cell reads and writes in successive CCB slots sustained for 256 cells at a time. At this speed, it verifies that the CIF correctly transmits and receivers cells, the W_IVC and R_IVC FIFOs capture data, the CDM correctly asserts FA, the FLM provides W_CADRs and recycles R_CADRs and that the CMBF and port usage counters count correctly. This test targets many of the performance attributes for each SCU block as well as tests that the blocks work correctly together under an integrated test scenario. (Previous tests have not stressed the interaction of the blocks but have mainly tested each block independently.)

5.3 Test Case Coverage

The following table, "Test Case Coverage" , summarizes SCU Diagnostics test cases and their coverage by listing all the attributes tested by each. Attributes listed in bold font indicate that they are the main targets of the test while those listed in plain font are used during the test. The shaded rows at the end of the table list all attributes that can not be tested by this activity.

Table 2 Test Case Coverage

Test Name	Attributes	Brief Description	Result Implications	See Section
"CIF Register and Reset Test"	**AF4**,**AF5**,**AF6**,**AF15**,**AF16**,**AF17**,**AF18**,**AF19**,**AF20**,**AF21**,**AF22**,**AF23**,**AF24**,**AB3**,**AB9**,**AB15**	Verifies that the FIFO status and error bits in the CIF Status Register are 0 after reset. Also performs read/write integrity tests on most CIF registers.	Returns OK or ERROR if test fails. 32-bit status codes will also be forwarded to the Switch Workstation.	5.2.1
"CIF Shared Buffer Memory Test"	**AF1**,**AB19**	A series of reads and writes testing for stuck bits, leakage, address decoding errors and address and data line shorts.	““	5.2.2
"W_IVC Lookup Table Memory Test"	**AF2**,**AB20**	A series of reads and writes testing for stuck bits, leakage, address decoding errors and address and data line shorts.	““	5.2.3

Table 2 Test Case Coverage

Test Name	Attributes	Brief Description	Result Implications	See Section
"R_IVC Lookup Table Memory Test"	**AF3**,**AB21**	A series of reads and writes testing for stuck bits, leakage, address decoding errors and address and data line shorts.	““‘	5.2.4
"Partial CRC Register and Reset Test"		TBD	““‘	5.2.5
"FLM Register and Reset Test"	**CF3**,**CF4**,**CF5**,**CF11**	Tests FLM Control register. Verifies CMBF count, FLM Halt bit and FLM Empty bit values are correct after reset.	““‘	5.2.6
"FLM Free List RAM and Status Test"	**AF8**,AF12,AF27,AF45,AF46,AF58,AF63,**CF1**,**CF6**,CF7,**CB1**,**CB2,CB3,CB6,CB8,CB10,DF21**,**DF22**	A series of FLM initializations followed by Core Memory write cycles will be performed in order to check the Free List Ram for stuck bits, leakage, address decoding errors and address and data line shorts. Other FLM signals are also tested.	““‘	5.2.7
"CDM Register Test"	**DF6**,**DF8,**DF9,**DF68**	A series of reads and writes to the CDM's Control and Current Chalkmark registers to verify their integrity. Also verifies initial value of CDM Status register.	““‘	5.2.8
"DBM Configure and Capture Test"	AF7,AF27,AF28,AF40,**AF43**,**AF44**,**AF62**,CF1,CF6,CF7,DF1,DF2,DF3,**DF10**,**DF11**,**DF23**,**DF24**,**DF25**,**DF26**,**DF27**,**DF28**,DB2,DB3	A series of DBM configurations followed by Core Memory write and read cycles will be performed in order to check the that DBMs cna be configured, DBMs can be transferred into the wDBM RAM and that capturing works correctly.DBM and wDBM RAM for stuck bits, leakage, address decoding errors and address and data line shorts.	““‘	5.2.10
"DBM Memory Test"	AF7,AF28,AF40,**DF1,DF3,DF5,**DF10,DF24,**DB2**	A series of DBM configurations followed by Core Memory write and read cycles will be performed to test DBM RAM integrity.	““‘	5.2.10

Table 2 Test Case Coverage

Test Name	Attributes	Brief Description	Result Implications	See Section
"DBM Memory Test"	AF7,AF27,AF28,AF40,CF1,CF6,CF7,DF1,DF2,**DF3**,DF27,DB2,**DB3**	A series of DBM configurations followed by Core Memory write and read cycles will be performed to test wDBM RAM integrity.	““‘	5.2.10
"CRM Registers and Reset Test"	**EF1,EF2,EF3,**EF4,**EF5**	Performs a series of reads and writes to the BF Threshold registers, the Port Number register and the Core Threshold register to verify their integrity. It alos verifies that the Port BF Counters are initialized to zero.	““‘	5.2.12
"CTC Register and Reset Test"	**FF3**,**FF4**,**FF6**	Verifies the read/write integrity of the CTC's IT Start and IT Stop register and that the IT timer is initialized to 0 after reset	““‘	5.2.13
"P-Bus Register and Reset Test"	GF1**,GF2,**GF4,**GF8,GF40,GF46**	Verifies the read/write integrity of the P-Bus' PM Reset register, that the PM Interrupt and reset register are initialized to zero and that the Core Memory partiy error counter is zero after reset.		5.2.14
"BTC Control Register Integrity and Reset Test"	**HF1,HF5**	Verifies the read/write integrity of the BTC Setup register and that the CBT error counter is 0 after reset.		5.2.15
"CIF IVC FIFO Test"	AF2,AF3,AF4,**AF7,AF8,** AF10**,AF11,AF12,AF13,** AF27,AF28,**AF40,**AF45, AF46**,AF58,AF59,AF63, AF64,AF65,AF67,AF68, AF69,AF70,AF72,AF73, AF74,AF75,AF77,AF78, AF79,AF80,AF81,AF82, AF83,AF84,AF85,AF87, AF88,AF89.AF90,AF91, AF92,AF94,AF95,AF96, AF97,AF98,AF99,AB1,AB2,AB4,AB5,AB6,AB7, AB8,AB10,AB11,AB12, AB13,AB14,AB16,AB17,AB18,**CF1,CF6,CF7,CF9,DF22,EF31,GF47	Performs a series of cell reads and writes to check IVC FIFO data capturing based on the IVC Lokup Tables, IVC FIFO data integrity as well as status and boundary conditions. It also verifies the status bits associated with each FIFO and its boundary conditions and the ability of the SCU to interrupt the GPU when each FIFO reaches the mid-range and almost-full levels.	““‘	5.2.16

Table 2 Test Case Coverage

Test Name	Attributes	Brief Description	Result Implications	See Section
"CIF Data Path Integrity and Control and Status Register Test"	AF1,AF4,AF5,AF6,AF7,AF8,**AF27,AF28,AF29,AF30,AF31,AF32,AF33,AF45,AF46,AF47,AF48,AF54,AF55**,AF58,AF59,**AF105**,**AF107**,**AF109**,**AF111**,CF1,CF6,**CF7,**CF9,DF22,EF31,IF1	Performs a series of cell reads and writes to verify CIF data path integrity as well as the functioning of certain bits in the CIF Control register.	““	5.2.17
"CIF CRC8 Generation/ Checking Test"	AF4,AF5,AF6,AF10,AF27,**AF35,AF36,AF37,AF37,**AF45,**AF49,AF50,AF51,AF51,AF101,AF102,AF103,AF104,**CF1,CF6,CF7**,**DF22,GF47,IF1	Performs several reads and writes to ensure that cell CRC8 errors are only detected when an actaul CRC8 error occurs and that it causes the GPU to be interrupted. It also guarantees that the CRC8 check/generate bit clears after read and write cycles.	““	5.2.18
"CDM Monitoring and FA Signalling Test"	AF4,AF7,AF8,AF12,AF27,AF28,AF40,AF45,AF46,AF58,AF59,AF63,AF65,CF1,CF6,CF7,CF8,CF9,CB3,DF1,DF2,DF3,DF10,DF11,**DF12,DF14,DF15,DF16,DF19,DF20,**DF22,**DB1,**EF31	Verifies that DBM modification and FA signalling are performed correctly under various cell read and write conditions.	““	5.2.19
"FLM Basic Test"	AF4,AF7,AF12,AF27,AF28,AF40,AF45,AF46,AF63,AF65,AF67,CF3,CF6,CF7,**CF8,CF9,CF10,**CF12**,CF13,CF15,CF16,**CF17**,CF19,CF19,CF22,CF23,**DF22,EF31	Verifies that the FLM provides W_CADRs to the backplane and increments/decrements the CMBF count under the proper conditions. It also verifies the functioning of the FLM Halt bit.	““	5.2.20
"FLM Boundary and Integrity Test"	AF4,AF8,AF27,AF45,AF58,**CF2,**CF6,CF7,**CF12,CF17,CF24,CF25,**CB3**,CB4,CB5**,CB8,**CB9,CB11,CB12,**DF3**,**DF10,DF11,DF19,DF22,GF47	Verifies the correct functionality of the CMBF count under boundary conditions as well as verifying the functionality of the FLW, FLF and FL Empty signals. It also verifies that W_CE_NULL is asserted when the FLM is empty.	““	5.2.21
"CDM Chalkmark Update Test"	AF4,AF7,AF8,AF27,AF28,AF40,AF45,AF46,AF58,AF59,CF1,CF6,CF7,CF8,CF9,DF1,DF2,DF3,DF6,DF10,DF15,DF16,DF19,DF22,DF26,**DF29,DF30,DF32,DF29,DF34,DF36,DF29,DF38,DF29,DF42,**EF31	Verifies that the CDM correctly updates a CADR's wDBM with the correct current chalkmark (unicast or multicast) but only under the correct cell read and write circumstances.	““	5.2.22

Table 2 Test Case Coverage

Test Name	Attributes	Brief Description	Result Implications	See Section
"CDM Chalkmark Search Test"	AF4,AF7,AF8,AF27,AF40,AF45,AF58,AF59,CF1,CF6,CF7,DF1,DF2,DF3,DF6,DF7,DF10,DF11,DF19,DF20,DF22,DF29,DF29,DF29,DF29,**DF45,DF54,DF55,DF56,DF57,DF58,DF59,DF60,DF61,DF62,DF63,DF64,DF65,DP5,DP6,**GF47	Performs a series of reads and writes with several current chalkmark settings then performs a stale chalkmard search. It then verifies that the CDM correctly finds the CADRs with stale chalkmarks and reports them to the GPU.	““	5.2.23
"CRM Port Usage Count Test"	AF2,AF3,AF4,AF7,AF8,AF27,AF28,AF40,**AF41**,AF45AF46,AF58,AF59,**AF41**,AF67,CF1,CF6,CF7,CF8,CF9,DF1,DF2,DF3,DF10,DF11,DF19,DF20,DF22,**EF4,EF6,EF7,EF9,EF10,EF11,EF14,EB1,EB2**	Verifies the CRM Port counts track the number of cells inserted in the Core Memory by each port.	““	5.2.24
"CRM Core Memory Protection Test"	AF2,AF3,AF4,AF7,AF8,AF12,AF27,AF40,AF41,AF45,AF58,AF59,AF41,AF63,AF64,AF77,CF6,CF7,CF12,DF1,DF2,DF3,DF10,DF11,DF19,DF22,EF1,EF4,EF6,EF7,EF10,**EF16,EF31,EF32**	With the global BF threshold surpassed, it verifies that the SCM's port BF counts is artificially promoted and that its writes are nulled when its highest threshold has been surpassed. This is also repeated without artificial BF promotion.	““	5.2.25
"CTC Cell Period Timer Test"	**FF1,FF5,FB1**	Ensures that the CTC's Cell Period Timer continually increments once per cell period wrapping around to 0 when it reaches 16K-1.	““	5.2.26
"CTC IT Starting and Maximum Interval Test"	AF4,AF7,AF8,AF27,AF28,AF40,AF45,AF46,AF59,CF6,DF22,**FF2**,FF3,**FF7,FF8,FF9,FF12,FF13,FF14,FF15,FF17,FF18,FF31,FB2,FB3,**GF47	Verifies that the IT only start counting at 0 on valid cell write and read cycles with an IVC for which it was programmed to start, increments once per cell period and holds at it maximum value of 16M-1 when it is not stopped.	““	5.2.27
"CTC IT Timing and Minimal Interval Test"	AF4,AF7,AF8,AF27,AF28,AF40,AF45,AF46,AF59,CF6,DF22,FF2,FF3,FF4,FF7,FF13,**FF19,FF20,FF21,FF24,FF25,FF26,FF27,FF29,FF30**,FF31,**FF32**,**FB4,FB5,**GF47	Verifies that the IT stops counting only on valid cell write and read cycles with an IVC for which it was programmed to stop, measures timed intervals correctly (even those only 1 cell period long) and interrupts the GPU when it is done timing an interval.	““	5.2.28

Table 2 Test Case Coverage

Test Name	Attributes	Brief Description	Result Implications	See Section
"CBT Integrity Checking Test"	AF4,AF5,AF6,AF7,AF8, AF27,AF31,AF35,AF36, AF45,AF54,CF6,CF7,GF3,**GF36,GF37**,GF47,HF1,HF2,**HF3**,HF5,**HF6,HF7, HF8,HF9,HF10,HF11,HB1,HB2,IF4,JF6,JF7**	CIF performs a series of cell writes and reads with CRC8 checking/generation on and off to verify that the TC and BTC check CBT integrity and advise the GPU correctly.		5.2.29
Core Memory Test	AF1,AF4,AF5,AF6,AF27, AF31,AF45,AF54,AF58, CF1,CF6,CF7,**IF1,IF2,IF3,IB1**	A black box Core Memory test which tests for stuck bits, leakage, address decoding errors and address and data line shorts.	““	5.2.30
"WCADR_PERR Test"	See sections 5.2.1,through 5.2.28.	Tests in sections 5.2.1 through 5.2.28 can be performed again with the WCADR_PERR backplane signal asserted to verify that this signal does not have any affect on the operation of the SCU.		5.2.31
"CIF Backplane Parity Error Test"	AF4**,**AF6**,**AF7**,**AF10**,**AF27**,**AF31**,AF34,**AF40**,AF56,AF57,AF66,AF71,,AF76,**AF80**,**AF87**,**AF94**,AF106,AF108,AF110**	Verifies that the CIF does not transfer cells to the Core Memory or capture W_IVC FIFO data when WIVC_PERR is asserted. Also verifies that the CIF does not transfers cells from the Core Memory or captures RIVC_FIFO data when RIVC_PERR or RCADR_PERR is asserted. The bits corresponding to the above parity errors in the CIF Status register is also tested.	““	5.2.32
"FLM Backplane Parity Error Test"	AF4,AF6,AF7,AF27,AF31,AF34,AF45,AF56,AF57,CF6,**CF14**,**CF20**,**CF21**, **CB7**,DF22	Verifies that the CMBF counter in the FLM does not change on cell write cycles with WIVC_PERR asserted or on cell read cycles with RCADR_PERR or RIVC_PERR asserted.	““	5.2.33
"CDM Backplane Parity Error Test"	AF4,AF7,AF8,AF27,AF40,AF45,AF59,CF6,DF10, **DF13**,**DF17**,**DF18**,DF22, DF24,DF27,**DF31**,**DF35**, **DF39**,**DF40**,**DF43**,**DF44**	Verifies that the CDM does not modify wDBMs or update chalkmarks on cell write cycles with WIVC_PERR asserted or on cell read cycles with RIVC_PERR or RCADR_PERR asserted.	““	5.2.34

Table 2 Test Case Coverage

Test Name	Attributes	Brief Description	Result Implications	See Section
"CRM Port Usage Count Test with Backplane Parity Errors"	AF2,AF3,AF4,AF7,AF8, AF27,AF40,AF45,AF59, CF6,DF10,DF22,EF4,EF6,**EF8,EF12,EF13**	Verifies that a port's counter does not increment/decrement when the WIVC_PERR, RIVC_PERR or RCADR_PERR signals are asserted.	““‘	5.2.35
"CRM BF Threshold Test"	AF2,AF3,AF4,AF7,AF8, AF27,AF40,AF45,AF58, CF6,CF7,EF1,EF6,EF10,EF16,**EF17,EF18,EF19, EF20,EF21,EF22,EF23, EF24,EF25,EF26,EF27, EF28,EF29,EF30,**EF31, EF32	Verifies that the BF threshold bits, RPI and W_CE_NULL presented to a port durin its CCB slot are correct for various port usage counter values.	““‘	5.2.36
"CTC IT Backplane Parity Error Test."	AF2,AF3,AF4,AF7,AF8, AF27,AF40,AF45,AF58, CF6,CF7,FF3,FF4,FF7,**FF10,FF11,**FF13,**FF16,FF22,FF23,FF28**	Verifies that the IT does not start or stop on cell write cycles with WIVC_PERR asserted or cell read cycles with RIVC_PERR or RCADR_PERR asserted.	““‘	5.2.37
"Core Memory Parity Error Monitoring Test"	AF4,AF8,AF27,AF31,AF45,AF54,CF6,CF7,DF22, **GF1,**GF3,**GF9,GF10,GF11,GF12,GF13,GF34,GF35,**GF47,**GB1,GB2,IF5**	Performs several cell writes and reads to verify that the P-Bus's Core Memory parity error counter is functioning correctly and that the GPU is interrupted when the counter reaches a pre-determined value and the appropriate bit in the Alarms register is set.	““‘	5.2.38
"Switch Alarms Test"	**GF3,GF14,GF15,GF16, GF17,GF18,GF19,GF20, GF21,GF22,GF23,GF24, GF25,GF26,GF27,GF28, GF29,GF30,GF31,GF32, GF33,GF47**	Vrifies that the P-Bus is correctly recognizing the AC fail, Fan fail, Power supply fail and Core Memory clock fail alarms, interrupting the GPU with a distinct interrupt for each and setting the appropriate bits in the alarms register are set.	““‘	5.2.39
"PM Reset Register Test"	**GF2,GF38,GF39**	Verifies that the SCU asserts reset pulses correctly on the TDM backplane PM reset signal according to the contents of the PM Reset register.	““‘	5.2.40

Table 2 Test Case Coverage

Test Name	Attributes	Brief Description	Result Implications	See Section
"PM Interrupt Test"	GF2,GF38,GF39,**GF41, GF44,GF45,GF43,For more information see "The SCU is not interrupted with a PM interrupt if all the bits in the PM Interrupt register are clear" on page 45.**	This test loops the PM Reset signal back to the PM interrupt signal in order to verify the integrity of the PM Interrupt register and the PM interrupt going to the GPU.	““	5.2.41
"CIF W_LOG/ W_ID/R_LOG/ R_ID Test"	AF4,**AF9,AF14,**AF27,**AF42,**AF45,**AF61**	Verifies that the CIF correctly drives the R_LOG, R_ID, W_LOG and W_ID buses on the backplane based on the values written to the correcponding.	This test will only be performed if time permits.	5.2.42
"CIF Encoding/ Decoding Test"	AF1,AF4,AF5,AF6,AF27, AF31,**AF39,**AF45,**AF53,** AF54,CF6,CF7,DF22	Verifies that 7-8 encoding and 8-7 decoding function correctly.	This test will only be performed if time permits.	5.2.43
"CIF Idle Cell Exchange Test"	**AF25,AF26**	Verifies that the CIF exchanges idle cells with the Core Memory when it is not performing cell write or read cycles during its CCB slot.	This test will only be performed if time permits.	5.2.44
"Performance and SCU Block Integration Test"	**AP1,AP2,AP3,AP4,AP5, AP6,AP7,AP8,CP1,CP5, CP6,CP7,CP10,CP11,DP1,DP2,DP3,DP4,EP1,EP3**	Verifies many performance attributes and integration of the various blocks in the SCU. It does this while performing bursts of 256 cell write and read cycles.	““ Since this test covers so many attributes, only the performance attributes are listed.	5.2.45
TC H/W Verification	**JF1,JF2,JF3,JF4,JF8,JF9,JF10,JF11**	These are attributes are to be covered by Thomas Ng during H/W verification.		
Uncovered Functional Attributes	**AF86,AF93,AF100,CF26,CF27,CF28,CF29,CF30, DF46,DF47,DF48,DF49, DF50,DF51,DF52,DF53**	These attributes could not be tested by this feature. This may because it is not possible to induce a certain type of error or the conditions under which it would be possible to verify the attribute are extremely hard to create.		
Uncovered Performance Attributes.	**CP2,CP3,CP4,CP8,CP9, EP2**	These attributes can not be tested without the use of other Port Cards or other load generating hardware.		

6.0 Operations

6.1 System initiated actions

Not applicable.

6.2 User initiated actions

During System Initialization, all tests outlined in the Functional Specification with the exception of those in "Basic Functionality Tests - Hardware Verification Requiring Manual Intervention" are normally executed. However, the VxWorks Shell provides a human user working at the switch workstation the ability to invoke the SCU Diagnostics controller from the shell (during switch development) in such a way that only specific SCU tests are executed. The user can similarly invoke each test individually.

7.0 Application interfaces

The entities invoking SCU Diagnostics are described in a previous section. (See "User initiated actions" on page 64.) SCU Diagnostics is invoked by calling the corresponding controller passing it a test mask. The test mask contains one bit for each test and determines which tests are executed.

SCU Diagnostics interfaces with the following software entities:

- The switch status code logging facilities, to communicate results to users;
- A function to write to the ID PROM if severe hardware problems are encountered;

8.0 Restrictions/limitations

Not applicable.

9.0 Optionality

Not applicable.

10.0 Activity impact

This activity will impact the CreditNet Switch SCU hardware debugging since the tests developed will be used to diagnose the hardware. Once switches are in service, it will verify SCU sanity during System Initialization and can be used to assist in fault isolation when hardware problems are encountered.

10.1 Associated activities

This activity interacts with System Initialization in the sense that it is a part of System Initialization and is invoked by System Initialization software. It also impacts the User Interface software since the User Interface is responsible for converting some of the results of SCU Diagnostics into a user readable form.

11.0 Future enhancements

Not applicable.

12.0 Glossary of definitions & abbreviations

ATM	:Asynchronous Transfer Mode
BF	:Buffer Fill
BFC	:Buffer Fill Count
BFS	:Buffer Fill Status
BNR	:Bell Northern Research
BTC	:Bus-tap Controller
BTL	:Bus Transceiver Logic
CADR	:Cell Address (in the Core Memory)
CBFC	:Core Memory Buffer Fill Count
CBT	:Cell Bus Tap
CCB	:Cell Control Bus
CDM	:Cell Departure Monitor
CIF	:Cell Interface
CRM	:Core Memory Resource Manager
FIFO	:First-In First-Out
FLM	:Free List Manager
GPU	:General Processor Unit
IVC	:Internal Virtual Connection
NIS_INIT	:Not In Service - Initialization State
NIS_NETWK	:Not In Service - Network State

P-BUS :Processor Bus

PM :Port Module

R_CADR :Read CADR

R_IVC :Read IVC

RTT :Round Trip Time

SCU :Switch Control Unit

SCM :Switch Control Module (consists of a GPU and an SCU)

TBD :To Be Determined

TC :Terminator Card

TDM :Time Division Multiplexed

TTL :Transistor-Transistor Logic

W_CADR :Write CADR

W_IVC :Write IVC

W_CE_NULL :Write Common Equipment Null

13.0 References

Internal Documents:

[1] Melhem Chaar, BNR, “Design Specification - The Credit Network - MasterCard”, Feb. 14, 1994.

[2] Gary Depelteau, BNR, “System Initialization FN”, Sept. 1, 1993.

[3] Gary Depelteau, BNR, “System Initialization DD”, Sept. 16, 1993.

[4] Dave Milton, BNR. “Cell MemorySpecification.”

[5] Dave Milton, BNR. “CreditNet GPU Specification.” Version 1.0.

Design description (DD)

14.0 WARNING

Although the basic overall contents of this document are still relevant specific details are out of date. The HW for CRM, Bus-tap controller and Partial CRC were not implemented along with the corresponding SW to test them. Parity error HW and the verificatoin SW to test it (in scuHVDiag.c) were also not implemented. If in doubt, please contact the authors.

15.0 Issue/change control

Table 3Revision History

Issue	Date	Reason	Change	Author
1.0	February 25, 1994	Initial release for internal review	First Draft	Gary Depelteau
2.0	June 13, 1994	Release for external review	Updated to reflect new HW info and results of FN review.	Gary Depelteau
2.1	Oct 5, 1994	Update	Minor changes to scuSCoreTest (Sec. 16.3.16)	Mark Brezina
2.2	May 9, 1995	Update	Added Warning	Gary Depelteau

16.0 Design approach

16.1 Overview

This activity, Switch Control Unit, Core Memory and Terminator Card Diagnostics, provides software for:

1.Switch Control Unit (SCU) hardware verification;

2.SCU system start-up diagnostics;

3.Core Memory (CM) system start-up diagnostics;

4.Terminator Card (TC) hardware verification

5.TC system start-up diagnostics.

This activity is referred to as simply SCU Diagnostics throughout the rest of this and other documents. For a more detailed description of the SCU, Core Memory and TC please refer to the FN section of this document [5].

16.2 Detailed Description

16.2.1 Bootstrap Phase Testing

For the purpose of performing start-up diagnostics, SCU Diagnostics control software, residing in function ‘scuSUDiagCtl’ (SCU Start-up Diagnostics Controller), is invoked by function ‘scmSysInitCtl’ after the VxWorks real-time operating system has been downloaded from the switch work station [6,7]. At this point in the switch booting process, ‘scuSUDiagCtl’ is invoked with a test mask instructing it to perform all start-up diagnostics tests. For more information see "SCU Diagnostics Test Mask" on page 69. ‘scuSUDiagCtl’ is also responsible for determining the results of the tests performed during SCU Diagnostics and reporting them to the switch User Interface (UI).

16.2.2 Hardware Verification

Users at the workstation can also execute any test or set of tests from the VxWorks Shell. Any subset of the start-up diagnostics can be invoked by calling ‘scuSDiagCtl’ with a test mask having one bit set for each corresponding test to be performed. Any individual test, including the hardware verification tests, can be invoked individually by entering the name of the test at the VxWorks Shell prompt.

When each test is performed, it can print error messages to the VxWorks console and send 32-bit status codes to the switch UI to announce specific errors. When each tests completes, it returns OK or ERROR depending on whether or not the test passes.

16.2.3 Initialization

During the course of performing SCU Diagnostics, the SCU is initialized differently from test to test. This initialization is similar to that performed by SCU Initialization during System Initialization except that IVCs and the free addresses in the FLM are initialized in a manner suitable for performing each test. SCU Driver Software functions are used to perform this initialization. Initialization requirements for each test are described in section 17.0, "Software specification" .

16.2.4 Restart handling

Not applicable.

16.2.5 Data structures

16.2.5.1 SCU Test Mask

The SCU test mask is a 32-bit word passed to function ‘scuSUDiagCtl’ when it is invoked. It contains one bit for each SCU test that can be performed during start-up diagnostics. When ‘scuSUDiagCtl’ receives this test mask it performs each test that has its corresponding mask bit set. For more information see "SCU Diagnostics Test Mask" on page 69.

Table 4 SCU Diagnostics Test Mask

Bit	Test Function Invoked	Bit	Test Function Invoked
0	scuSMCifRegTest	16	
1	scuSMSharBuffTest	17	
2	scuSMWivcLutTest	18	
3	scuSMRivcLutTest	19	
4	scuSMParCrcTest	20	
5	scuSMFlmRegTest	21	
6	scuSMFlmRamStatTest	22	
7	scuSMCdmRegTest	23	
8	scuSMDbmConCapTest	24	
9	scuSMDbmRamTest	25	
10	scuSMWDbmRamTest	26	
11	scuSMCrmRegTest	27	
12	scuSMCtcRegTest	28	
13	scuSMPbusRegTest	29	
14	scuSMBtcRegTest	30	
15		31	

16.3 Design Evolution/Strategy and Topology

As illustrated in section "SCU Software Topology" on page 70, SCU/Core Diagnostics are built on top of SCU Driver and SCU Initialization software. As well, SCU Initialization software is built on top of the SCU Driver software.

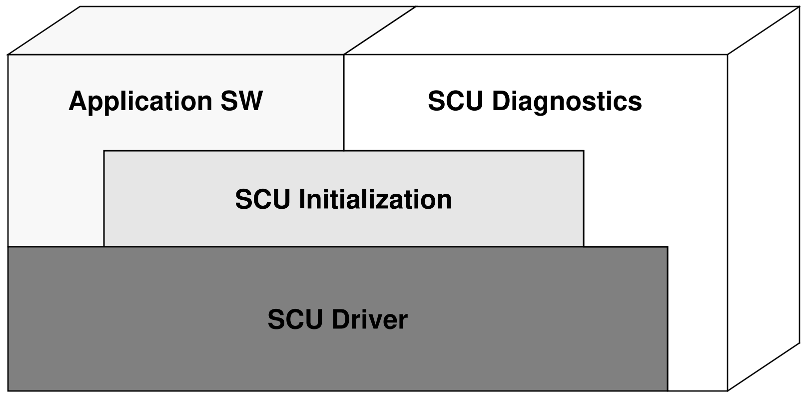

Figure 2 SCU Software Topology

SCU Diagnostics, Driver and Initialization software development occurs somewhat concurrently with development time shifted between the three bodies of software so as to minimize the impact of SCU hardware design changes. SCU Diagnostics tests are designed and coded in the order in which they are expected to be used for SCU design and hardware verification. This corresponds to the order in which they are defined in Section 17.0, "Software specification" .

17.0 Software specification

17.1 Modular structure

SCU Diagnostics software is divided into four modules:

1. Module ‘scuSMDiag.c’ - SCU Start-up Memory Diagnostics:
    - Memory and Register Tests - executed during start-up diagnostics but also used for H/W verification testing;
2. Module ‘scuSDiag.c - SCU Start-up Diagnostics:
    - Basic Functionality Tests - executed during start-up diagnostics but also used for H/W verification testing;
    - Test Control - function ‘scuSUDiagCtl’;
3. Module ‘scuHVDiag.c’ - SCU H/W Verification Diagnostics:
    - Basic Functionality Tests - H/W verification requiring manual intervention. Tests SCU attributes that are not testable by ‘scuSuMemDiag.c’ and ‘scuSuDiag.c’ without manual intervention (ie. requiring jumper, scope, etc.);
4. Module ‘scuPDiag.c’ - SCU Performance Diagnostics:
    - Performance and Integration Tests - executed during start-up diagnostics but also used for H/W verification testing;

For a further description of the different groups of tests please see [5]. For a complete list of the above modules and their functions please refer to the following sections.

17.2 Module ‘scuSMDiag.c’

17.2.1 Function ‘scuSMCifRegTest’

17.2.1.1 Purpose

The purpose of ‘scuSMCifRegTest’ (SCU Start-up Memory Diagnostics CIF Register Test) is to test the read and/or write integrity of the various CIF registers and verify that certain registers have been initialized to correct values. This test assumes that it is called after the SCU has been reset.

17.2.1.2 Function

‘scuSMCifRegTest’ performs the following functions:

1. Initializes result to OK;
2. Calls function ‘scuCifStatRegRead’ to read the CIF Status register;

3. If the CIF Status register does not have all bits set to zero then it sets result to ERROR and calls function ‘gpuUIErrorSend’ in module ‘gpuUIinterface.c’ to send a message to the UI indicating that the CIF did not contain zero after reset;
4. Performs a read of the high priority W_IVC data FIFO register;
5. If the entry read from the FIFO is not equal to zero then it sets result to ERROR and calls function ‘gpuUIErrorSend’ to send a message to the UI indicating that the high priority W_IVC data FIFO register did not contain zero after reset;
6. Performs a read of the low priority W_IVC data FIFO register;
7. If the entry read from the FIFO is not equal to zero then it sets result to ERROR and calls function ‘gpuUIErrorSend’ to send a message to the UI indicating that the low priority W_IVC data FIFO register did not contain zero after reset;
8. Performs a read of the R_IVC FIFO register;
9. If the entry read from the FIFO is not equal to zero then it sets result to ERROR and calls function ‘gpuUIErrorSend’ to send a message to the UI indicating that the R_IVC data FIFO register did not contain zero after reset;
10. Calls function ‘scmRegTest’ in module ‘scmUtil.c’ passing it the address of the CIF control register, a region width of two bytes and the CIF control register’s mask. If ‘scmRegTest returns ERROR then this function sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that the CIF control register testing failed;
11. Calls function ‘scmRegTest’ in module ‘scmUtil.c’ passing it the address of the CIF Data Memory Read Address register, a region width of two bytes and the CIF Data Memory Read Address register’s mask. If ‘scmRegTest returns ERROR then this function sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that the CIF Data Memory Read Address register testing failed;
12. Calls function ‘scmRegTest’ in module ‘scmUtil.c’ passing it the address of the CIF Data Memory Write Address register, a region width of two bytes and the CIF Data Memory Write Address register’s mask. If ‘scmRegTest returns ERROR then this function sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that the CIF Data Memory Write Address register testing failed;
13. Returns result.

17.2.1.3 External interfaces

Function ‘scuSMCifRegTest’ is invoked by ‘scuSuDiagCtl’ in module ‘scuSuDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

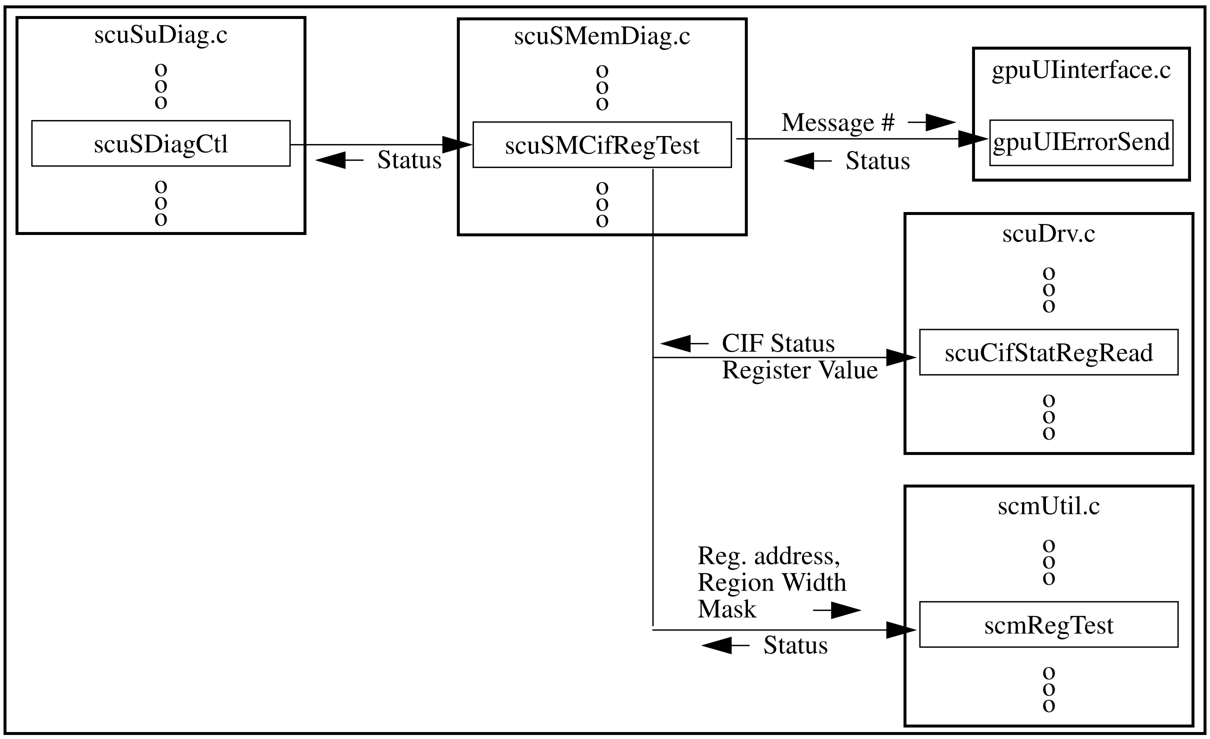

Figure 3 Function ‘scuSMCifRegTest’ External Interface

17.2.1.4 Internal interfaces

17.2.2 Function ‘scuSMSharBuffTest’

17.2.2.1 Purpose

The purpose of ‘scuSMSharBuffTest’ (SCU Start-up Memory Diagnostics Shared Buffer Test) is to verify the integrity of the CIF’s Shared Buffer memory.

17.2.2.2 Function

'scuSMSharBuffTest' performs the following functions:

1. Initializes result to OK;
2. Performs 32-bit writes filling each location in the Shared Buffer with its address. The writes are performed in increasing order;
3. Reads the contents of all 32-bit locations in the Shared Buffer. If the contents of a location are not the same as its address then it sets result to ERROR but continues to check the rest of the locations;
4. Fills in the fields of a MEMORY_REGION data structure according to the parameters pertaining to the Shared Buffer memory;
5. Calls function 'xMemTest' in 'memTest.c' passing it the following:
    - Pointer to the MEMORY_REGION data structure defined above;
    - Number of memories to test equal to 1;
    - Number of iterations to perform equal to 100 (this guarantees fault coverage greater than 99.9%);
    - Pointer to a faulty address variable;
    - Pointer to a memory block variable (indicates which block of memory contained an error);
6. If 'xMemTest' returns ERROR then it sets result to ERROR;
7. Calls function 'bzero' in VxWorks module 'string.c', passing it the start address of the Shared Buffer and its size in bytes, so that this function leaves the buffer filled with zeroes instead of random numbers written by 'xMemTest';
8. Returns result.

17.2.2.3 External interfaces

Function ‘scuSMSharBuffTest’ is invoked by ‘scuSuDiagCtl’ in module ‘scuSuDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

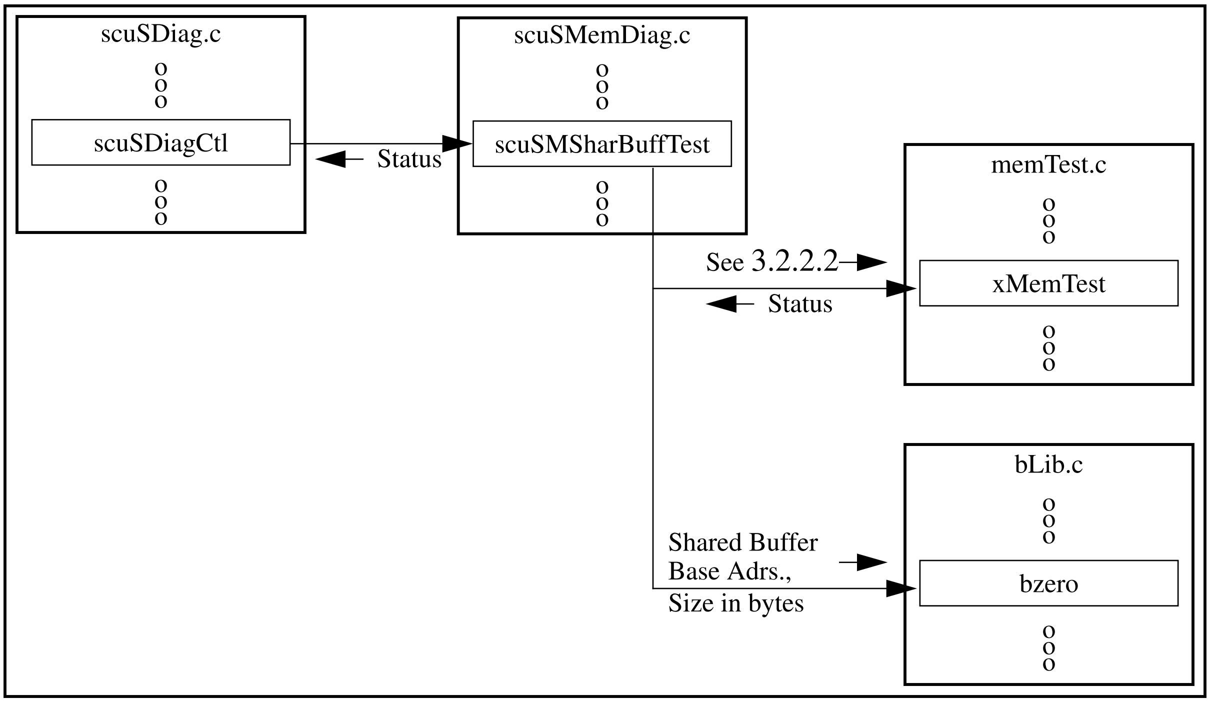

Figure 4 Function ‘scuSMSharBuffTest’ External Interface

17.2.2.4 Internal interfaces

Not applicable.

17.2.3 Function ‘scuSMWivcLutTest’

17.2.3.1 Purpose

The purpose of ‘scuSMWivcLutTest’ (SCU Start-up Memory Diagnostics W_IVC Lookup Table Test) is to verify the integrity of the CIF’s W_IVC Lookup Table memory.

17.2.3.2 Function

‘scuSMWivcLutest’ performs the following functions:

1. Initializes result to OK;
2. Performs 32-bit writes filling each location in the W_IVC Lookup Table with its address. The writes are performed in increasing order;
3. Reads the contents of all 32-bit locations in the W_IVC Lookup Table. If the contents of a location are not the same its address then it set result to ERROR but continues to check the rest of the locations;
4. Fills in the fields of a MEMORY_REGION data structure according to the parameters pertaining to the W_IVC Lookup Table memory;
5. Calls function ‘xMemTest’ in ‘memTest.c’ passing it the following:
    - Pointer to the MEMORY_REGION data structure defined above;
    - Number of memories to test equal to 1;
    - Number of iterations to perform equal to 100 (this guarantees fault coverage greater than 99.9%);
    - Pointer to a faulty address variable;
    - Pointer to a memory block variable (indicates which block of memory contained an error);
6. If ‘xMemTest’ returns ERROR then it sets result to ERROR;
7. Calls function ‘bzero’ in VxWorks module ‘string.c’, passing it the start address of the W_IVC Lookup Table and its size in bytes, so that this function leaves the buffer filled with zeroes instead of random numbers written by ‘xMemTest’;
8. Returns result.

17.2.3.3 External interfaces

Function ‘scuSMWivcLutTest’ is invoked by ‘scuSuDiagCtl’ in module ‘scuSuDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

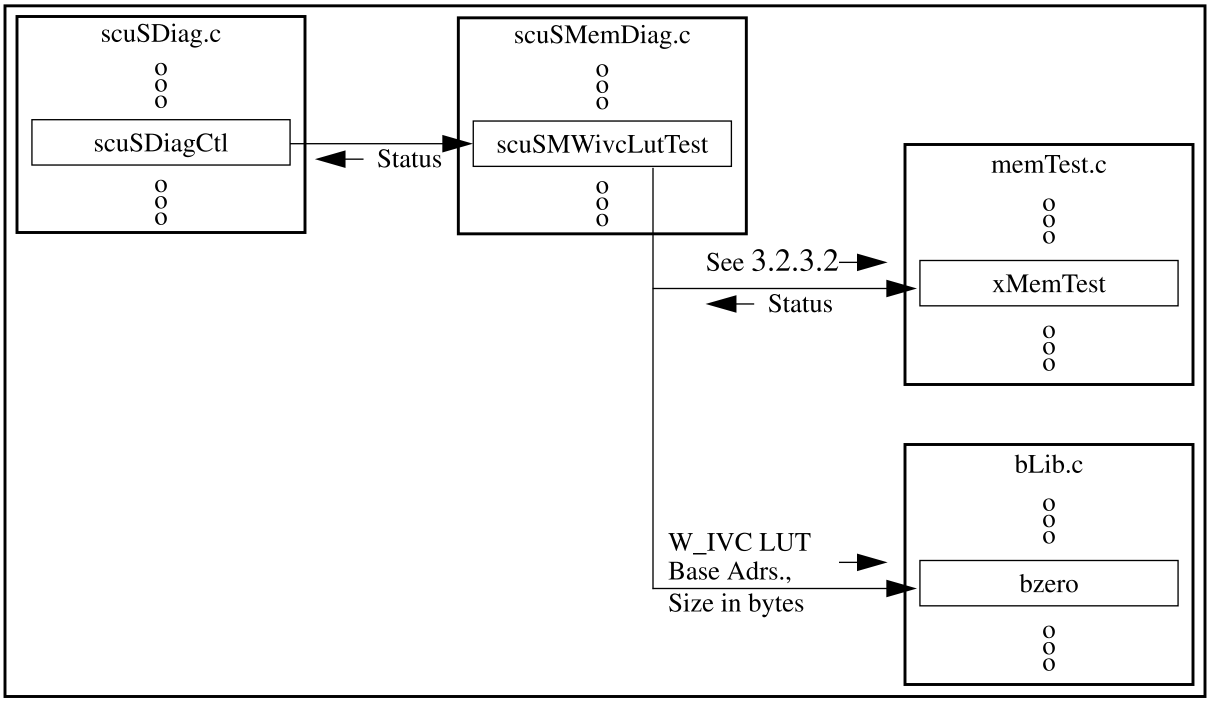

Figure 5 Function ‘scuSMWivcLutTest’ External Interface

17.2.3.4 Internal interfaces

Not applicable.

17.2.4 Function ‘scuSMRivcLutTest’

17.2.4.1 Purpose

The purpose of ‘scuSMRivcLutTest’ (SCU Start-up Memory Diagnostics R_IVC Lookup Table Test) is to verify the integrity of the CIF’s R_IVC Lookup Table memory.

17.2.4.2 Function

'scuSMRivcLutest' performs the following functions:

1. Initializes result to OK;
2. Performs 32-bit writes filling each location in the R_IVC Lookup Table with its address. The writes are performed in increasing order;
3. Reads the contents of all 32-bit locations in the R_IVC Lookup Table. If the contents of a location are not the same as its address then it set result to ERROR but continues to check the rest of the locations;
4. Fills in the fields of a MEMORY_REGION data structure according to the parameters pertaining to the R_IVC Lookup Table memory;
5. Calls function 'xMemTest' in 'memTest.c' passing it the following:
    - Pointer to the MEMORY_REGION data structure defined above;
    - Number of memories to test equal to 1;
    - Number of iterations to perform equal to 100 (this guarantees fault coverage greater than 99.9%);
    - Pointer to a faulty address variable;
    - Pointer to a memory block variable (indicates which block of memory contained an error);
6. If 'xMemTest' returns ERROR then it sets result to ERROR;
7. Calls function 'bzero' in VxWorks module 'string.c', passing it the start address of the R_IVC Lookup Table and its size in bytes, so that this function leaves the buffer filled with zeroes instead of random numbers written by 'xMemTest';
8. Returns result.

17.2.4.3 External interfaces

Function ‘scuSMRivcLutTest’ is invoked by ‘scuSuDiagCtl’ in module ‘scuSuDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

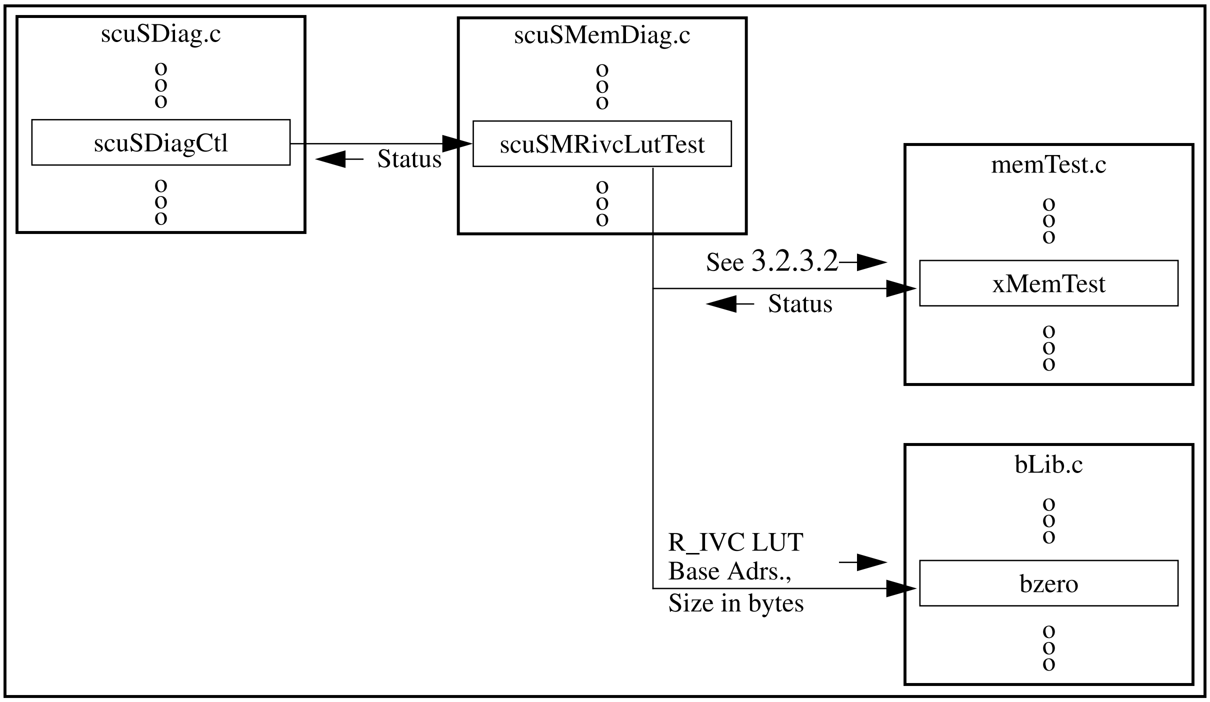

Figure 6 Function ‘scuSMRivcLutTest’ External Interface

17.2.4.4 Internal interfaces

Not applicable.

17.2.5 Function ‘scuSMParCrcTest’

17.2.5.1 Purpose

TBD.

17.2.5.2 Function

17.2.5.3 External interfaces

17.2.5.4 Internal interfaces

17.2.6 Function ‘scuSMFlmRegTest’

17.2.6.1 Purpose

The purpose of ‘scuSMFlmRegTest’ (SCU Start-up Memory Diagnostics FLM Register Test) is to test the read/write integrity of the certain FLM registers and verify that certain registers have been initialized to correct values. This test assumes that it is called after the SCU has been reset.

17.2.6.2 Function

‘scuSMFlmRegTest’ performs the following functions:

1. Initializes result to OK;
2. Calls function ‘scuFlmStatRegRead’ to read the FLM Status register;
3. If the FLM Status register does not have the FLM Empty bit set then it sets result to ERROR and calls function ‘gpuUIErrorSend’ in module ‘gpuUIinterface.c’ to send a message to the UI indicating that the FLM Status register was not initialized correctly after reset;
4. Reads the FLM Control register and if the FLM Halt bit is not set then it sets result to ERROR and calls function ‘gpuUIErrorSend’ in module ‘gpuUIinterface.c’ to send a message to the UI indicating that the FLM Control register was not initialized correctly after reset;
5. Calls function ‘scuFlmCmbfCountRead’ to read the FLM Core Memory Buffer Fill (CMBF) count;
6. If the CMBF count is not equal to zero then it sets result to ERROR and calls function ‘gpuUIErrorSend’ to send a message to the UI indicating that the CMBF count did not contain zero after reset;
7. Calls function ‘scmRegTest’ in module ‘scmUtil.c’ passing it the address of the FLM Control register, a region width of two bytes and the FLM Control register’s mask. If ‘scmRegTest returns ERROR then this function sets result to ERROR and sends a message to

switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that the FLM Control register testing failed;

8. Returns result.

17.2.6.3 External interfaces

Function ‘scuSMFlmRegTest’ is invoked by ‘scuSuDiagCtl’ in module ‘scuSuDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

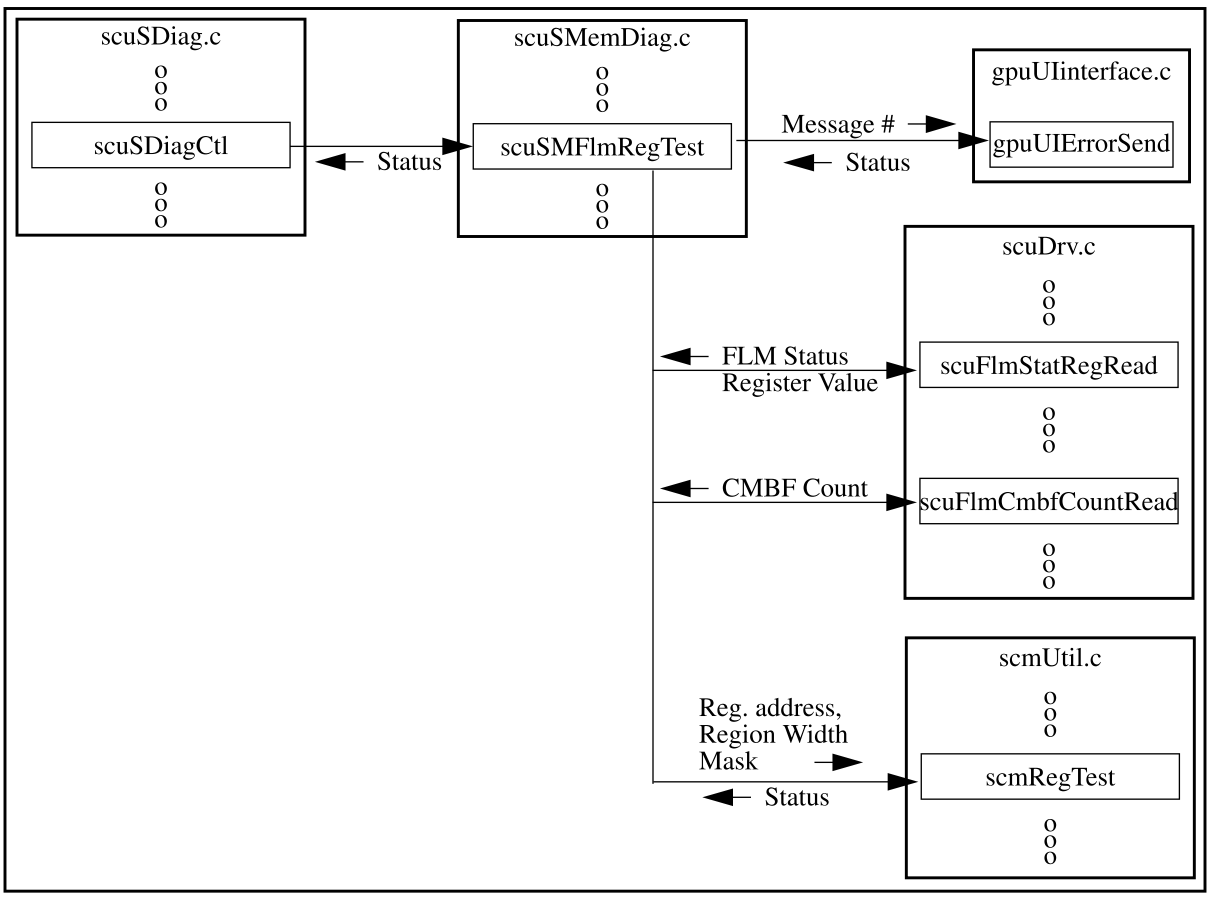

Figure 7 Function ‘scuSMFlmRegTest’ External Interface

17.2.6.4 Internal interfaces

17.2.7 Function 'scuSMFlmRamStatTest'

17.2.7.1 Purpose

The purpose of 'scuSMFlmRamTest' (SCU Start-up Memory Diagnostics FLM Ram and Status Test) is to test the read/write integrity of the FLM RAM and verify the functioning of the FLM Full and FLM Empty bits in the FLM Status register.

17.2.7.2 Function

'scuSMFlmRamStatTest' performs the following functions:

1. Initializes result to OK;
2. Resets the SCU;
3. Sets all CRM port thresholds for the SCM to their maximum values;
4. Sets the Core Memory BF Threshold in the CRM to 16K;
5. Calls function 'scuWivcLutSetup' passing it the following:
    - A NULL pointer;
    - A number of IVCs to initialize of 1;
    - A W_CSID of 5;
    - A W_IVC of 55;
    - A high priority W_IVC FIFO enable flag set to 1;
    - A low priority W_IVC FIFO enable flag set to 0;
6. Fills each location in the FLM with its own address by calling function 'scuFlmFreeListInit' and passing it a number of cell addresses to initialize of 16K;
7. Calls function 'scuFlmStatRegRead' to read the FLM Status register;
8. If the FLM Status register does not have the FLM Full bit set then it calls function 'gpuUIErrorSend' in module 'gpuUIinterface.c' to send a message to the UI indicating that the FLM Full bit in the FLM Status register was not set when the FLM was expected to be full and then returns ERROR;
9. Calls function 'scuFlmFreeListInit' passing it a cell address of 0x4000 and a number of cell addresses to initialize of 1 so that it attempts to place one more cell address on the Free List;
10. Calls function 'scuFlmStatRegRead' to read the FLM Status register;
11. If the FLM Status register does not have the FLM Full bit set then it calls function 'gpuUIErrorSend' in module 'gpuUIinterface.c' to send a message to the UI indicating that the FLM Full bit was cleared on a read occurring when the Free List was already full and then returns ERROR;
12. Repeats the following for CADR from 0 to (16K - 1):

- Calls function ‘scuCellXferToCore’ passing it the following:
    - A W_ID of 0x2;
    - A W_LOG of 0x55;
    - CIF Control flag with the Write One Cell and Write Cycle Valid bits asserted;
    - An IVC of 0x55;
    - A cell index of 0x0;
- Calls function ‘scuCifWivcFifoRead’ to read the high priority W_IVC FIFO;
- If the value returned by ‘scuCifWivcFifoRead’ is not equal to CADR, it calls function ‘gpuUIErrorSend’ in module ‘gpuUIinterface.c’ to send a message to the UI indicating that an FLM RAM test failure occurred when the FLM was initialized with the R_VALID bit asserted and then returns ERROR;

13. Calls function ‘scuFlmStatRegRead’ to read the FLM Status register;
14. If the FLM Status register does not have the FLM Empty bit set then it calls function ‘gpuUIErrorSend’ in module ‘gpuUIinterface.c’ to send a message to the UI indicating that the FLM Empty bit in the FLM Status register was not set when the FLM was expected to be empty and then returns ERROR;
15. Calls function ‘scuCellXferToCore’ passing it the following:
    - A W_ID of 0x2;
    - A W_LOG of 0x55;
    - CIF Control flag with the Write One Cell and Write Cycle Valid bits asserted;
    - An IVC of 0x55;
    - A cell index of 0x0;
16. Calls function ‘scuCifWivcFifoRead’ to read the high priority W_IVC FIFO. If the value returned by ‘scuCifWivcFifoRead’ is not equal to 0x3fff then it calls function ‘gpuUIErrorSend’ in module ‘gpuUIinterface.c’ to send a message to the UI indicating that an FLM RAM test failure occurred when the FLM was initialized with the R_VALID bit asserted and then returns ERROR;
17. Calls function ‘scuFlmStatRegRead’ to read the FLM Status register;
18. If the FLM Status register does not have the FLM Empty bit set then it and calls function ‘gpuUIErrorSend’ in module ‘gpuUIinterface.c’ to send a message to the UI indicating that the FLM Empty bit was cleared after a cell write before which the empty bit was set and then returns ERROR;
19. Fills an array of 16K CADRs with 0x1555;
20. Calls function ‘scuFlmFreeListInit’ to fill the Free List with 0x1555 by passing it the address of the array of 0x1555 CADRs, a number of CADRs to initialize of 16K and a CADR of 0x0;

21. Calls function ‘scuFlmStatRegRead’ to read the FLM Status register;
22. If the FLM Status register does not have the FLM Full bit set then it calls function ‘gpuUIErrorSend’ in module ‘gpuUIinterface.c’ to send a message to the UI indicating that the FLM Full bit in the FLM Status register was not set when the FLM was expected to be full and then returns ERROR;
23. Repeats the following for CADR from 0 to (16K - 1):
    - Calls function ‘scuCellXferToCore’ passing it the following:
        - A W_ID of 0x2;
        - A W_LOG of 0x55;
        - CIF Control flag with the Write One Cell and Write Cycle Valid bits asserted;
        - An IVC of 0x55;
        - A cell index of 0x0;
    - Calls function ‘scuCifWivcFifoRead’ to read the high priority W_IVC FIFO;
    - If the value returned by ‘scuCifWivcFifoRead’ is not equal to 0x1555 then it calls function ‘gpuUIErrorSend’ in module ‘gpuUIinterface.c’ to send a message to the UI indicating that an FLM RAM test failure occurred when the FLM was initialized with the R_VALID bit unasserted and then returns ERROR;
24. Calls function ‘scuFlmStatRegRead’ to read the FLM Status register;
25. If the FLM Status register does not have the FLM Empty bit set then it calls function ‘gpuUIErrorSend’ in module ‘gpuUIinterface.c’ to send a message to the UI indicating that the FLM Empty bit in the FLM Status register was not set when the FLM was expected to be empty and then returns ERROR;
26. Repeats steps 16 through 22 filling the Free List with and checking it for a CADR of 0x2aaa;
27. Repeats steps 16 through 22 filling the Free List with and checking it for a CADR of 0x1555;
28. Repeats steps 16 through 22 filling the Free List with and checking it for a CADR of 0x3fff;
29. Repeats steps 16 through 22 filling the Free List with and checking it for a CADR of 0x0000;
30. Repeats the following walking a one across a CADR variable from lsb to msb and back;
    - Calls function ‘scuFlmFreeListInit’ passing it a NULL pointer, a number of CADRs to initialize of 1 and the CADR variable;
    - Calls function ‘scuCellXferToCore’ passing it the following:
        - A W_ID of 0x2;
        - A W_LOG of 0x55;
        - CIF Control flag with the Write One Cell and Write Cycle Valid bits asserted;

    - An IVC of 0x55;
    - A cell index of 0x0;
  - Calls function ‘scuCifWivcFifoRead’ to read the high priority W_IVC FIFO;
  - If the value returned by ‘scuCifWivcFifoRead’ is not equal to the CADR variable then it calls function ‘gpuUIErrorSend’ in module ‘gpuUIinterface.c’ to send a message to the UI indicating that an FLM RAM test failure occurred when the FLM was initialized with the R_VALID bit unasserted and then returns ERROR;

31. Repeats step 20 walking a zero across a CADR variable;
32. Returns OK.

17.2.7.3 External interfaces

Function ‘scuSMFlmRamStatTest’ is invoked by ‘scuSuDiagCtl’ in module ‘scuSuDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

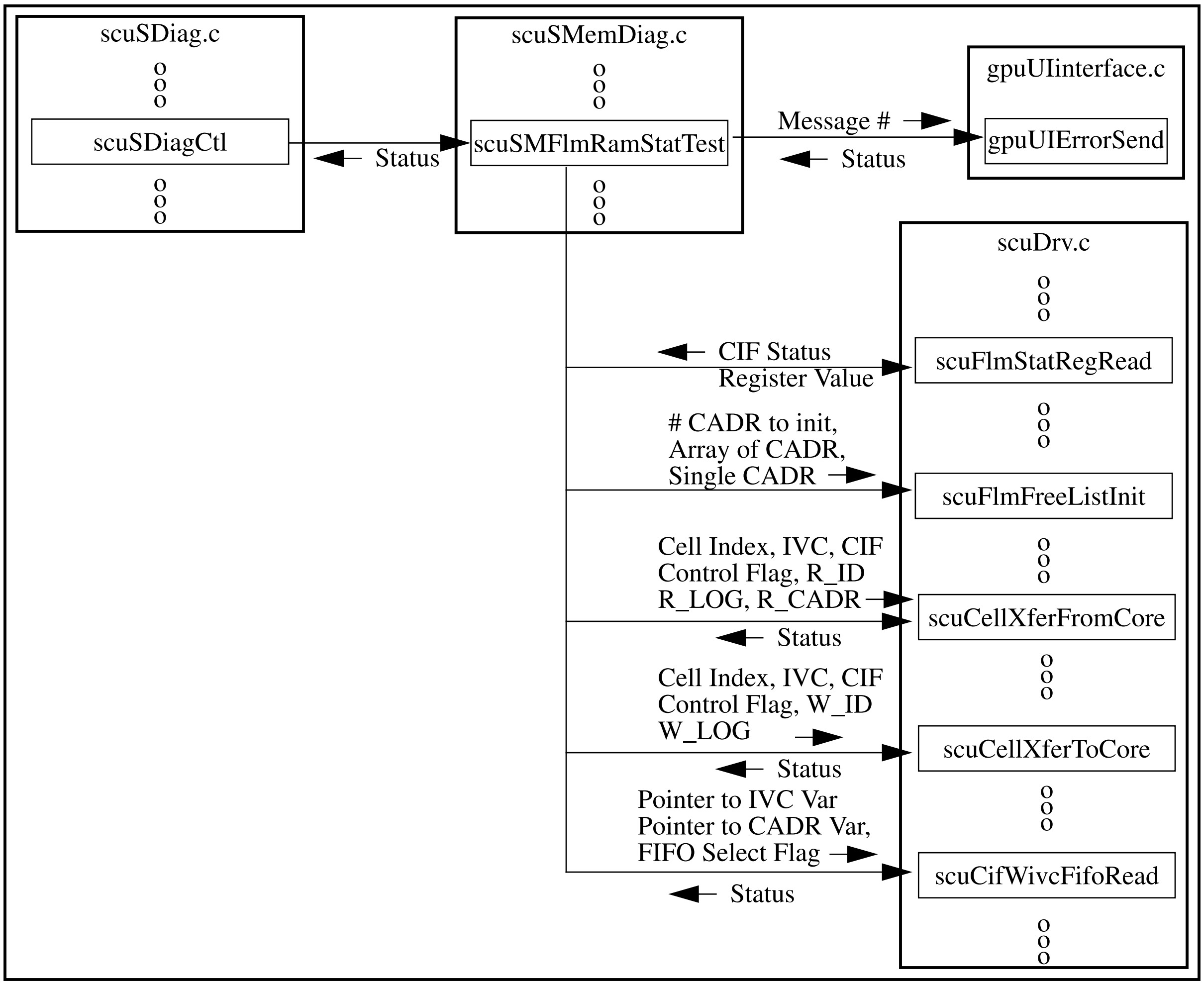

Figure 8 Function ‘scuSMFlmRamStatTest’ External Interface

17.2.7.4 Internal interfaces

17.2.8 Function ‘scuSMCdmRegTest’

17.2.8.1 Purpose

The purpose of ‘scuSMCdmRegTest’ (SCU Start-up Memory Diagnostics CDM Register Test) is to test the read/ write integrity of certain CDM registers and verify that certain registers have been initialized to correct values. This test assumes that it is called after the SCU has been reset.

17.2.8.2 Function

‘scuSMCdmRegTest’ performs the following functions:

1. Initializes result to OK;
2. Calls function ‘scuCdmStatRegRead’ to read the CDM Status register;
3. If the CDM Status register’s CH_ALARM bit is asserted then it sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that the CDM Status register was not initialized correctly;
4. Calls function ‘scmRegTest’ in module ‘scmUtil.c’ passing it the address of the CDM control register, a region width of two bytes and the CDM control register’s mask. If ‘scmRegTest returns ERROR then this function sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that CDM control register testing failed;
5. Calls function ‘scmRegTest’ in module ‘scmUtil.c’ passing it the address of the CDM Current Chalkmark register, a region width of two bytes and the CDM Current Chalkmark register’s mask. If ‘scmRegTest returns ERROR then this function sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that the CDM Current Chalkmark register testing failed;
6. Returns result.

17.2.8.3 External interfaces

Function ‘scuSMCdmRegTest’ is invoked by ‘scuSuDiagCtl’ in module ‘scuSuDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

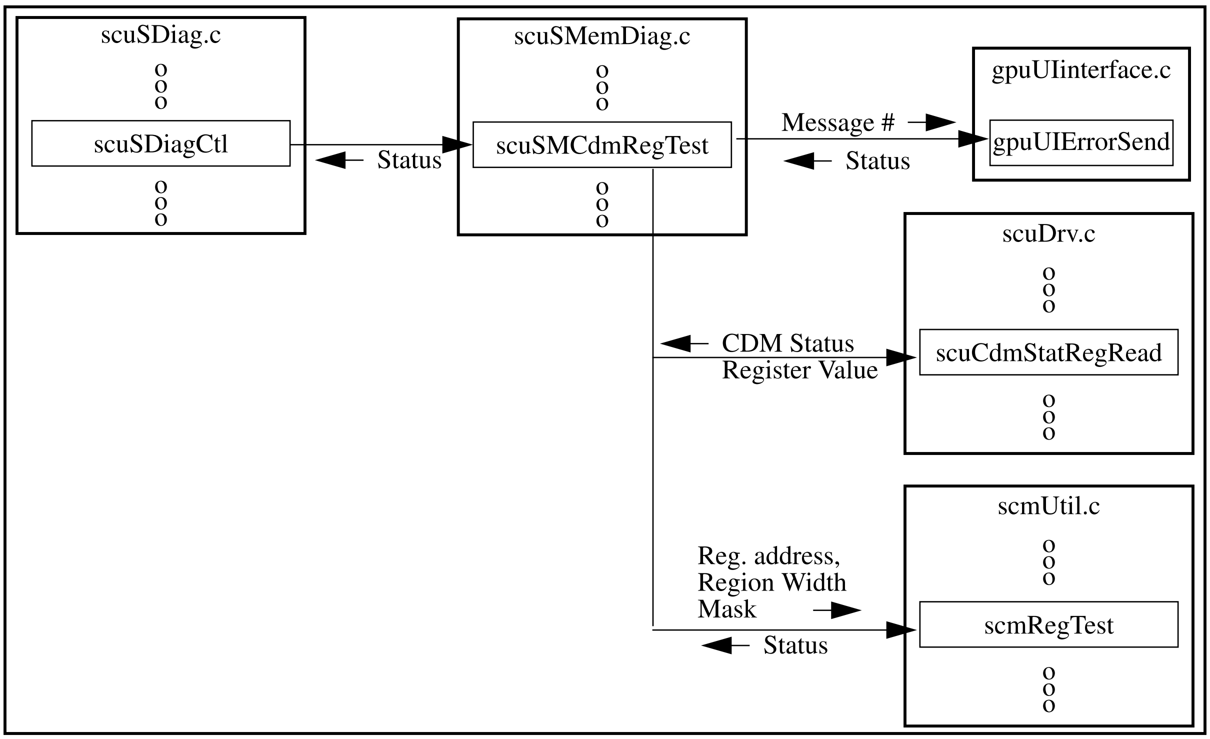

Figure 9 Function ‘scuSMCdmRegTest’ External Interface

17.2.8.4 Internal interfaces

17.2.9 Function ‘scuSMDbmConCapTest’

17.2.9.1 Purpose

The purpose of ‘scuSMDbmConCapTest’ (SCU Start-up Memory Diagnostics Distribution Bit-map (DBM) Configuration and Capture Test) is to verify that the DBM entries can be configured, DBM entries are copied into the appropriate wDBM locations on the appropriate Core Memory write cycles and that DBM and wDBM entries can be captured correctly.

17.2.9.2 Initialization

Table 5 IVC Configuration

IVC	W_CSID	W_ Low Prio. FIFO Enable	W_ High Prio FIFO Enable	R_CSID	R_IVC FIFO Enable	DBM Multicast Cell	DBM Ports[15:0]
0x100	0xa	0	1	0xa	1	0	0xaaaa
0x101	0x5	1	0	0x5	1	0	0x5555

17.2.9.3 Function

‘scuSMDbmConCapTest’ performs the following functions:

1. Initializes result to OK;
2. Resets the SCU;
3. Sets all CRM port thresholds for the SCM to their maximum values;
4. Sets the Core Memory BF Threshold in the CRM to 16K;
5. Calls function ‘scuIvcSetup’ to initialize IVCs 0x100 and 0x101 as indicated in Section 17.2.9.2;
6. Reads the contents of the CIF Control Register. If the DBM Configure bit is asserted then it sets result to ERROR and ends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that DBM Configure bit in the CIF Control register was not de-asserted after a Core Memory write cycle;
7. Calls function ‘scuFlmFreeListInit’ to initialize the Free List with only CADRs 0x0 and 0x1;
8. Calls function ‘scuCellXferToCore’ with an IVC of 0x100 and a CIF Control flag with the Write One Cell, Write Cycle Valid and the DBM Capture bits set;
9. Reads the Requested DBM register and if the value read is not 0xaaaa then it sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that a DBM capture failed with the Write Cycle Valid bit set;
10. Reads the contents of the CIF Control Register. If the DBM Capture bit is asserted then it sets result to ERROR and ends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that DBM Capture bit in the CIF Control register was not de-asserted after a Core Memory write cycle;
11. Calls function ‘scuCellXferToCore’ with an IVC of 0x101 and a CIF Control flag with the Write One Cell and the DBM Capture bits set;

12. Reads the Requested DBM register and if the value read is not 0x5555 then it sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that a DBM capture failed with the Write Cycle Valid bit clear;
13. Calls function ‘scuCellXferToCore’ with an IVC of 0x100 and a CIF Control flag with the Write One Cell bit set;
14. Reads the Requested DBM register and if the value read is 0xaaaa then it sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that a DBM capture occurred without the DBM Capture bit set;
15. Calls function ‘scuCellXferToCore’ with an IVC of 0x101 and a CIF Control flag with the Write One Cell and Write Cycle Valid;
16. Calls function ‘scuCellXferFromCore’ with an IVC of 0x100, a R_CADR of 0x0 and a CIF Control flag with the Read One Cell, Read Cycle Valid and the DBM Capture bits set;
17. Reads the Requested DBM register and if the value read is not 0xaaaa then it sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that a wDBM capture failed with the Read Cycle Valid bit set;
18. Reads the contents of the CIF Control Register. If the DBM Capture bit is asserted then it sets result to ERROR and ends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that DBM Capture bit in the CIF Control register was not de-asserted after a Core Memory write cycle;
19. Calls function ‘scuCellXferFromCore’ with an IVC of 0x101 and a CIF Control flag with the Read One Cell and the DBM Capture bits set;
20. Reads the Requested DBM register and if the value read is not 0x5555 then it sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that a wDBM capture failed with the Read Cycle Valid bit clear;
21. Calls function ‘scuCellXferFromCore’ with an IVC of 0x100 and a CIF Control flag with the Write One Cell bit set;
22. Reads the Requested DBM register and if the value read is 0xaaaa then it sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that a wDBM capture occurred without the DBM Capture bit set;
23. Returns result.

Idea - use ‘scuDbmSetup’ once debugging is complete. While debugging though use ‘scuIvcSetup’ setting up the entire IVC so that the W_LOG, W_ID, R_LOG, R_ID, W_CSID, and R_CSID have ceratin signatures and that the IVCs will be captured in the R_IVC and W_IVC FIFOs in the CIF.

17.2.9.4 External interfaces

Function ‘scuSMDbmConCapTest’ is invoked by ‘scuSuDiagCtl’ in module ‘scuSuDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

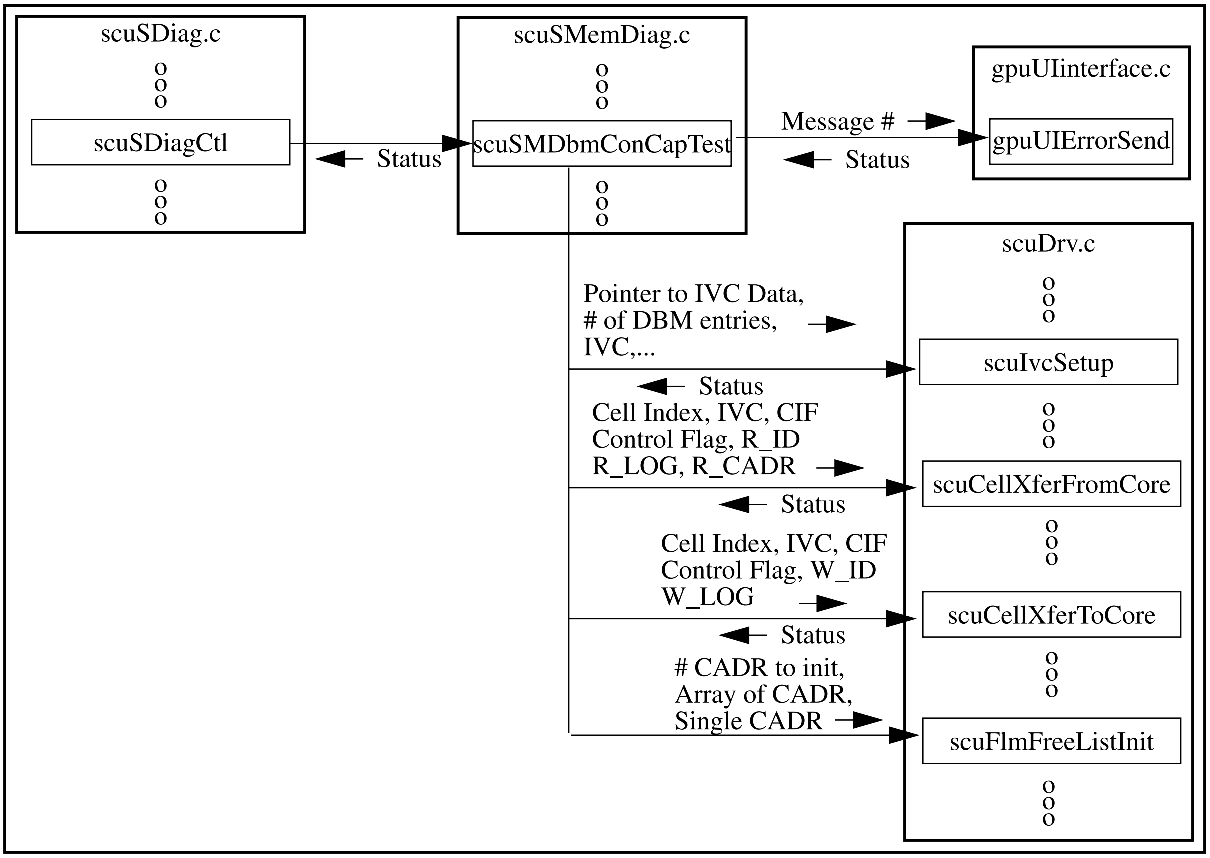

Figure 10 Function ‘scuSMDbmConCapTest’ External Interface

17.2.9.5 Internal interfaces

17.2.10 Function ‘scuSMDbmRamTest’

17.2.10.1 Purpose

The purpose of ‘scuSMDbmRamTest’ (SCU Start-up Memory Diagnostics DBM RAM Test) is to verify the integrity of the DBM RAM.

17.2.10.2 Function

‘scuSMDbmRamTest’ performs the following functions:

1. Calls function ‘scuFastReset’ to reset the SCU;
2. Sets all CRM port thresholds for the SCM to their maximum values;
3. Sets the Core Memory BF Threshold in the CRM to 16K;
4. Calls function ‘scuFlmFreeListInit’ passing it a NULL pointer, a number of CADRs to initialize of 16K and a CADR of 0;
5. Calls function ‘scuDbmSetup’ passing it a pointer to an array of DBM entries, a number of IVCs to initialize of 8K, an IVC equal to 0 and a DBM equal to 0. This is done to fill each DBM entry with its corresponding IVC;
6. Repeats the following for ivc from 0 to (8K - 1):
    - Calls function ‘scuDbmCapture’ with an IVC equal to ivc;
    - If the value returned from ‘scuDbmCapture’ is not the same as ivc then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a DBM RAM test failure occurred and returns ERROR;
    - Calls function ‘scuCellXferToCore’ passing it an IVC equal to ivc and a CIF Control flag with the Write Cycle Valid and Write One Cell bits set (this will place each wDBMs address in the first 8K wDBMs);
7. Repeats the following for ivc from 0 to (8K - 1):
    - Calls function ‘scuDbmSetup’ passing it a NULL pointer, a number of IVCs to initialize of 1, an IVC equal to ivc and a DBM equal to (ivc + 8K);
    - Calls function ‘scuCellXferToCore’ passing it an IVC equal to ivc and a CIF Control flag with the Write Cycle Valid and Write One Cell bits set (this will place each wDBMs address in the second 8K wDBMs);
8. Repeats the following for cadr from 0 to (16K - 1):
    - Calls function ‘scuWDbmCapture’ with an CADR equal to cadr;
    - If the value returned from ‘scuWDbmCapture’ is not the same as cadr then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a wDBM RAM test failure occurred and returns ERROR;
9. Calls function ‘scuFastReset’ to reset the SCU;
10. Calls function ‘scuDbmSetup’ passing it a pointer to an array of DBM entries, a number of IVCs to initialize of 8K, an IVC equal to 0 and a DBM equal to 0. This is done to fill each DBM entry with 0x5555;
11. Repeats the following for ivc from (8K - 1) to 0:
    - Calls function ‘scuDbmCapture’ with an IVC equal to ivc;

- If the value returned from ‘scuDbmCapture’ is not 0x5555 then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a DBM RAM test failure occurred and returns ERROR;
- Calls function ‘scuDbmSetup’ passing it a NULL pointer, a number of IVCs to initialize of 1, an IVC equal to ivc and a DBM equal to 0xaaaa;

12. Repeats the following for ivc from (8K - 1) to 0:
    - Calls function ‘scuDbmCapture’ with an IVC equal to ivc;
    - If the value returned from ‘scuDbmCapture’ is not 0xaaaa then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a DBM RAM test failure occurred and returns ERROR;
    - Calls function ‘scuDbmSetup’ passing it a NULL pointer, a number of IVCs to initialize of 1, an IVC equal to ivc and a DBM equal to 0x5555;
13. Repeats the following for ivc from (8K - 1) to 0:
    - Calls function ‘scuDbmCapture’ with an IVC equal to ivc;
    - If the value returned from ‘scuDbmCapture’ is not 0x5555 then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a DBM RAM test failure occurred and returns ERROR;
    - Calls function ‘scuDbmSetup’ passing it a NULL pointer, a number of IVCs to initialize of 1, an IVC equal to ivc and a DBM equal to 0x0000;
14. Repeats the following for ivc from 0 to (8K - 1):
    - Calls function ‘scuDbmCapture’ with an IVC equal to ivc;
    - If the value returned from ‘scuDbmCapture’ is not 0x0000 then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a DBM RAM test failure occurred and returns ERROR;
    - Calls function ‘scuDbmSetup’ passing it a NULL pointer, a number of IVCs to initialize of 1, an IVC equal to ivc and a DBM equal to 0xffff;
15. Repeats the following for ivc from 0 to (8K - 1):
    - Calls function ‘scuDbmCapture’ with an IVC equal to ivc;
    - If the value returned from ‘scuDbmCapture’ is not 0xffff then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a DBM RAM test failure occurred and returns ERROR;
    - Calls function ‘scuDbmSetup’ passing it a NULL pointer, a number of IVCs to initialize of 1, an IVC equal to ivc and a DBM equal to 0x0000;
16. Repeats the following for ivc from 0 to (8K - 1):
    - Calls function ‘scuDbmCapture’ with an IVC equal to ivc;
    - If the value returned from ‘scuDbmCapture’ is not 0x0000 then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indi-

cating that a DBM RAM test failure occurred and returns ERROR;

17. Calls function ‘scuFastReset’ to reset the SCU;
18. Repeats the following walking a one across the variable dbm from lsb to msb and back to lsb:
    - Calls function ‘scuDbmSetup’ passing it a NULL pointer, a number of IVCs to initialize of 1, an IVC equal to oxo and a DBM equal to dbm;
    - Calls function ‘scuDbmCapture’ with an IVC equal to 0x0;
    - If the value returned from ‘scuDbmCapture’ is not equal to dbm then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a DBM RAM test failure occurred and returns ERROR;
19. Repeats the following walking a zero across the variable dbm from lsb to msb and back to lsb:
    - Calls function ‘scuDbmSetup’ passing it a NULL pointer, a number of IVCs to initialize of 1, an IVC equal to 0x0 and a DBM equal to dbm;
    - Calls function ‘scuDbmCapture’ with an IVC equal to 0x0;
    - If the value returned from ‘scuDbmCapture’ is not equal to dbm then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a DBM RAM test failure occurred and returns ERROR;
20. Returns result.

Idea - use ‘scuDbmSetup’ once debugging is complete. While debugging though use ‘scuIvcSetup’ setting up the entire IVC so that the W_LOG, W_ID, R_LOG, R_ID, W_CSID, and R_CSID have ceratin signatures and that the IVCs will be captured in the R_IVC and W_IVC FIFOs in the CIF.

17.2.10.3 External interfaces

Function ‘scuSMDbmRamTest’ is invoked by ‘scuSuDiagCtl’ in module ‘scuSuDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

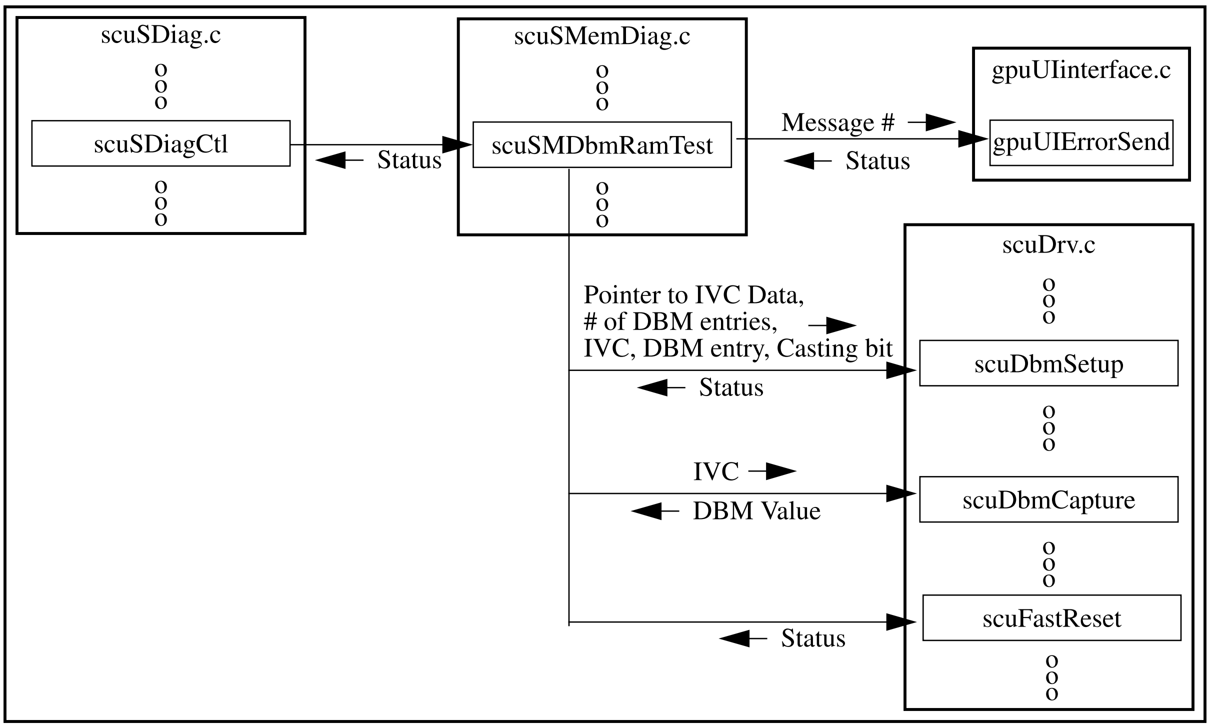

Figure 11 Function ‘scuSMDbmRamTest’ External Interface

17.2.10.4 Internal interfaces

17.2.11 Function ‘scuSMWDbmRamTest’

17.2.11.1 Purpose

The purpose of ‘scuSMWDbmRamTest’ (SCU Start-up Memory Diagnostics Working DBM Ram Test) is to verify that the integrity of the wDBM RAM.

17.2.11.2 Initialization

Table 6 IVC Configuration

IVC	W_CSID	W_ Low Prio. FIFO Enable	W_ High Prio FIFO Enable	R_CSID	R_IVC FIFO Enable	DBM Multicast Cell	DBM Ports[15:0]
0x100	0xa	0	1	0xa	1	0	0xaaaa
0x101	0x5	1	0	0x5	1	0	0x5555
0x102	0xf	0	1	0xf	1	0	0xffff
0x103	0x0	1	0	0x0	1	0	0x0000

17.2.11.3 Function

‘scuSMWDbmRamTest’ performs the following functions:

1. Initializes result to OK;
2. Calls function ‘scuFastReset’ to reset the SCU;
3. Sets all CRM port thresholds for the SCM to their maximum values;
4. Sets the Core Memory BF Threshold in the CRM to 16K;
5. Calls function ‘scuFlmFreeListInit’ passing it a NULL pointer, a number of CADRs to initialize of 16K and a CADR of 0;
6. Calls function ‘scuDbmSetup’ to initialize IVCs 0x100 through 0x103 as indicated in Section 17.2.9.2;
7. Repeats the following for cadr from 0 to (16K - 1):
    - Calls function ‘scuCellXferToCore’ with an IVC of 0x101 and a CIF Control flag with the Write One Cell and Write Cycle Valid (this fills the wDBM with 0x5555);
8. Calls function ‘scuFlmFreeListInit’ passing it a NULL pointer, a number of CADRs to initialize of 16K and a CADR of 0;
9. Repeats the following for cadr from (16K - 1) to 0:
    - Calls function ‘scuWDbmCapture’ with a CADR equal to cadr;
    - If the value returned from ‘scuWDbmCapture’ is not 0x5555 then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a wDBM RAM test failure occurred and returns ERROR;
    - Calls function ‘scuCellXferToCore’ with an IVC of 0x100 and a CIF Control flag with the Write One Cell and Write Cycle Valid (this fills the wDBM with 0xaaaa);

10. Calls function ‘scuFlmFreeListInit’ passing it a NULL pointer, a number of CADRs to initialize of 16K and a CADR of 0;
11. Repeats the following for cadr from (16K - 1) to 0:
    - Calls function ‘scuWDbmCapture’ with a CADR equal to cadr;
    - If the value returned from ‘scuWDbmCapture’ is not 0xaaaa then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a wDBM RAM test failure occurred and returns ERROR;
    - Calls function ‘scuCellXferToCore’ with an IVC of 0x101 and a CIF Control flag with the Write One Cell and Write Cycle Valid (this fills the wDBM with 0x5555);
12. Calls function ‘scuFlmFreeListInit’ passing it a NULL pointer, a number of CADRs to initialize of 16K and a CADR of 0;
13. Repeats the following for cadr from 0 to (16K - 1):
    - Calls function ‘scuWDbmCapture’ with a CADR equal to cadr;
    - If the value returned from ‘scuWDbmCapture’ is not 0x5555 then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a wDBM RAM test failure occurred and returns ERROR;
    - Calls function ‘scuCellXferToCore’ with an IVC of 0x102 and a CIF Control flag with the Write One Cell and Write Cycle Valid (this fills the wDBM with 0xffff);
14. Calls function ‘scuFlmFreeListInit’ passing it a NULL pointer, a number of CADRs to initialize of 16K and a CADR of 0;
15. Repeats the following for cadr from 0 to (16K - 1):
    - Calls function ‘scuWDbmCapture’ with a CADR equal to cadr;
    - If the value returned from ‘scuWDbmCapture’ is not 0xffff then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a wDBM RAM test failure occurred and returns ERROR;
    - Calls function ‘scuCellXferToCore’ with an IVC of 0x103 and a CIF Control flag with the Write One Cell and Write Cycle Valid (this fills the wDBM with 0x0000);
16. Calls function ‘scuFlmFreeListInit’ passing it a NULL pointer, a number of CADRs to initialize of 16K and a CADR of 0;
17. Repeats the following for cadr from 0 to (16K - 1):
    - Calls function ‘scuWDbmCapture’ with a CADR equal to cadr;
    - If the value returned from ‘scuWDbmCapture’ is not 0x0000 then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a wDBM RAM test failure occurred and returns ERROR;
    - Calls function ‘scuCellXferToCore’ with an IVC of 0x102 and a CIF Control flag with the Write One Cell and Write Cycle Valid;
18. Repeats the following for cadr from 0 to (16K - 1):

- Calls function ‘scuWDbmCapture’ with a CADR equal to cadr;
- If the value returned from ‘scuWDbmCapture’ is not 0xffff then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a wDBM RAM test failure occurred and returns ERROR;

19. Calls function ‘scuFastReset’ to reset the SCU;
20. Repeats the following walking a one across the variable dbm from lsb to msb and back to lsb:
    - Calls function ‘scuDbmSetup’ passing it a NULL pointer, a number of IVCs to initialize of 1, an IVC equal to 0x0 and a DBM equal to dbm;
    - Calls function ‘scuFlmFreeListInit’ passing it a NULL pointer, a number of CADRs to initialize of 1 and a CADR of 0x0;
    - Calls function ‘scuCellXferToCore’ with an IVC of 0x0 and a CIF Control flag with the Write One Cell and Write Cycle Valid;
    - If the value returned from ‘scuWDbmCapture’ is not equal to dbm then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a wDBM RAM test failure occurred and returns ERROR;
21. Repeats the following walking a zero across the variable dbm from lsb to msb and back to lsb:
    - Calls function ‘scuDbmSetup’ passing it a NULL pointer, a number of IVCs to initialize of 1, an IVC equal to 0x0 and a DBM equal to dbm;
    - Calls function ‘scuFlmFreeListInit’ passing it a NULL pointer, a number of CADRs to initialize of 1 and a CADR of 0x0;
    - Calls function ‘scuCellXferToCore’ with an IVC of 0x0 and a CIF Control flag with the Write One Cell and Write Cycle Valid (this fills the wDBM with 0x0000);
    - If the value returned from ‘scuWDbmCapture’ is not equal to dbm then it sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that a wDBM RAM test failure occurred and returns ERROR;
22. Returns result.

Idea - use ‘scuDbmSetup’ once debugging is complete. While debugging though use ‘scuIvcSetup’ setting up the entire IVC so that the W_LOG, W_ID, R_LOG, R_ID, W_CSID, and R_CSID have ceratin signatures and that the IVCs will be captured in the R_IVC and W_IVC FIFOs in the CIF.

17.2.11.4 External interfaces

Function ‘scuSMWDbmRamTest’ is invoked by ‘scuSuDiagCtl’ in module ‘scuSuDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

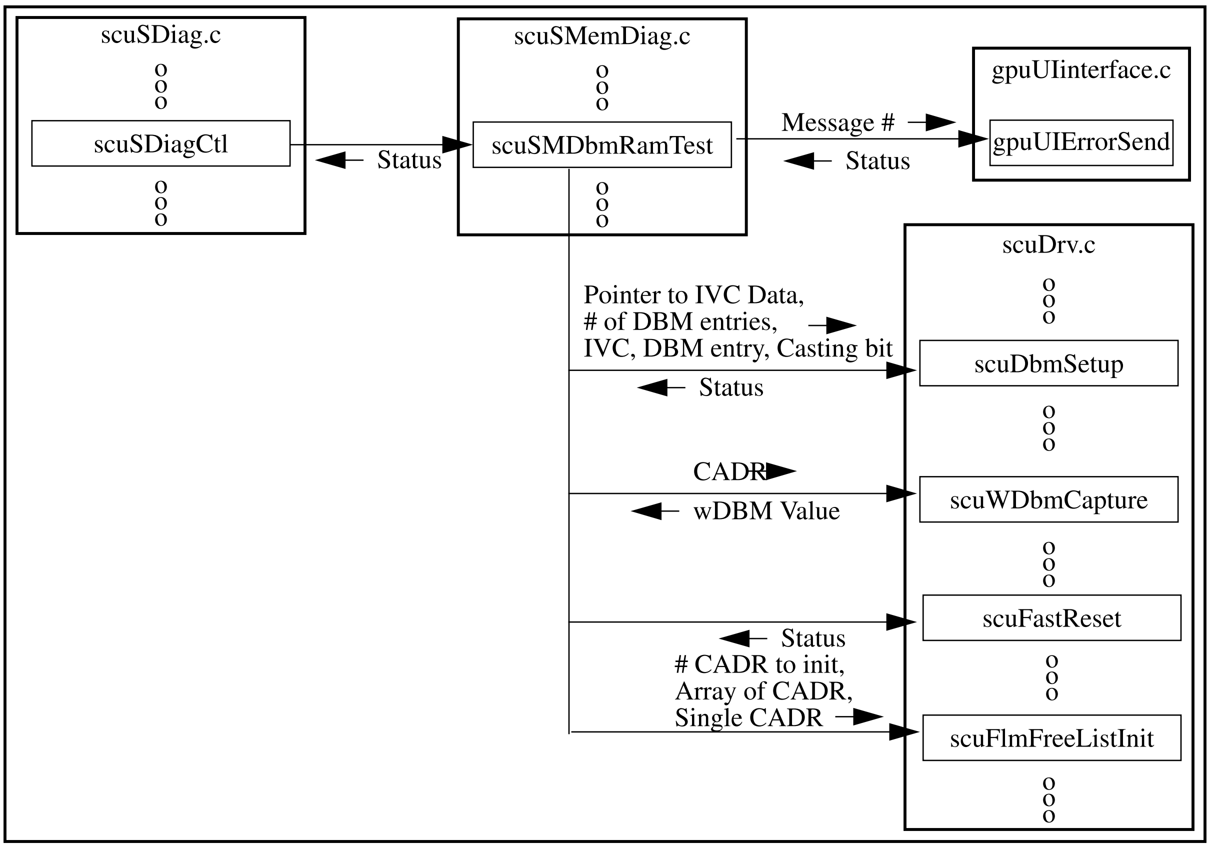

Figure 12 Function ‘scuSMWDbmRamTest’ External Interface

17.2.11.5 Internal interfaces

17.2.12 Function ‘scuSMCrmRegTest’

17.2.12.1 Purpose

The purpose of ‘scuSMCrmRegTest’ (SCU Start-up Memory Diagnostics CRM Register Test) is to test the read/write integrity of certain CRM registers. It also verifies that the CDM’s Port Buffer Fill (BF) counters are initialized to zero after reset. This test assumes that it is called after the SCU has been reset.

17.2.12.2 Function

'scuSMCrmRegTest' performs the following functions:

1. Initializes result to OK;
2. Calls function 'scuFastReset' to reset the SCU;
3. Calls function 'scmRegTest' in module 'scmUtil.c' passing it the address of the CRM Port Number register, a region width of two bytes and the CRM Port Number register's mask. If 'scmRegTest returns ERROR then this function sets result to ERROR and sends a message to switch UI by calling function 'gpuUIErrorSend' passing it a message number indicating that CRM Port Number register testing failed;
4. For each port, this test:
    - Calls function 'scuPortCountRead' passing it the particular port's number;
    - If 'scuPortCountRead' does not return zero for the particular port, then this function sets result to ERROR and sends a message to switch UI by calling function 'gpuUIErrorSend' passing it a message number indicating that the particular CRM Port Count was not initialized to zero;
5. Calls function 'scmRegTest' in module 'scmUtil.c' passing it the address of the CRM Core Memory Threshold register, a region width of two bytes and the CRM Core Memory Threshold register's mask. If 'scmRegTest returns ERROR then this function sets result to ERROR and sends a message to switch UI by calling function 'gpuUIErrorSend' passing it a message number indicating that CRM Core Memory Threshold register testing failed;
6. For each port, this test:
    - Calls function 'scmRegTest' in module 'scmUtil.c' passing it the address of a CRM Port Threshold register, a region width of two bytes and the CRM Port Threshold register's mask. If 'scmRegTest returns ERROR then this function sets result to ERROR and sends a message to switch UI by calling function 'gpuUIErrorSend' passing it a message number indicating that the particular CRM Port Threshold register testing failed;
7. Returns result.

17.2.12.3 External interfaces

Function ‘scuSMCdmRegTest’ is invoked by ‘scuSuDiagCtl’ in module ‘scuSuDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

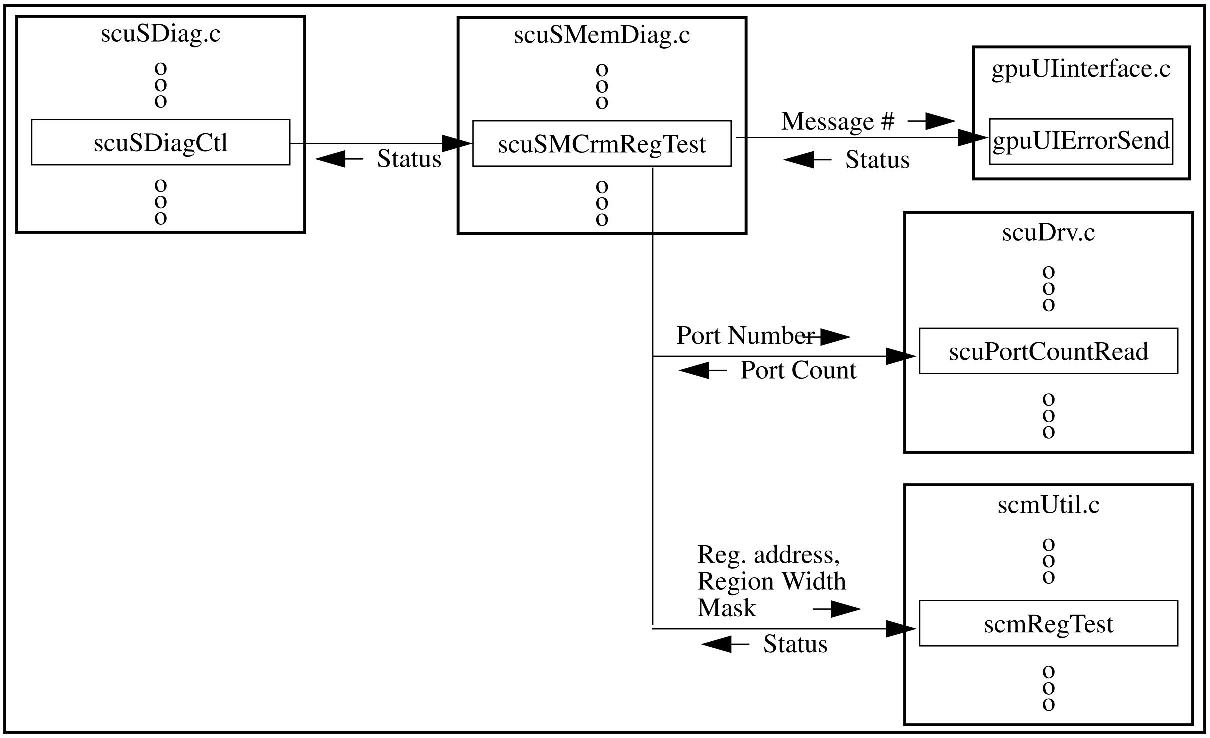

Figure 13 Function ‘scuSMCrmRegTest’ External Interface

17.2.12.4 Internal interfaces

17.2.13 Function ‘scuSMCtcRegTest’

17.2.13.1 Purpose

The purpose of ‘scuSMCtcRegTest’ (SCU Start-up Memory Diagnostics CTC Register Test) is to test the read/write integrity of certain CTC registers. It also verifies that the CTC’s Interval Timer (IT) is initialized to zero after reset. This test assumes that it is called after the SCU has been reset.

17.2.13.2 Function

'scuSMCtcRegTest' performs the following functions:

1. Initializes result to OK;
2. Calls function 'scuFastReset' to reset the SCU;
3. Calls function 'scmRegTest' in module 'scmUtil.c' passing it the address of the CTC's IT Start register, a region width of two bytes and the CTC's IT Start register's mask. If 'scmRegTest returns ERROR then this function sets result to ERROR and sends a message to switch UI by calling function 'gpuUIErrorSend' passing it a message number indicating that CTC's IT Start register testing failed;
4. Calls function 'scmRegTest' in module 'scmUtil.c' passing it the address of the CTC's IT Stop register, a region width of two bytes and the CTC's IT Stop register's mask. If 'scmRegTest returns ERROR then this function sets result to ERROR and sends a message to switch UI by calling function 'gpuUIErrorSend' passing it a message number indicating that CTC's IT Stop register testing failed;
5. Calls function 'scuCtcIntTimRead' to read the CTC's IT. If 'scuCtcIntTimRead' does not return zero then this function sets result to ERROR and sends a message to switch UI by calling function 'gpuUIErrorSend' passing it a message number indicating that CTC's IT was not initialized to zero after reset;
6. Returns result.

17.2.13.3 External interfaces

Function ‘scuSMCdmRegTest’ is invoked by ‘scuSuDiagCtl’ in module ‘scuSuDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

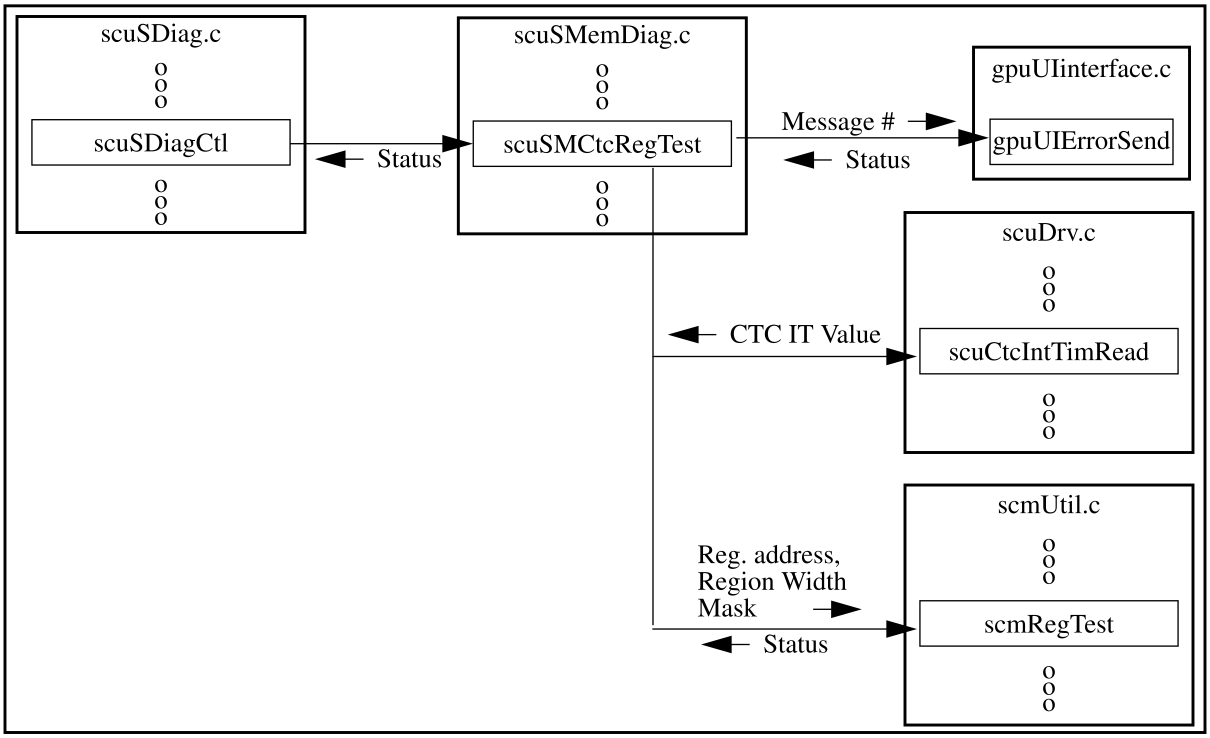

Figure 14 Function ‘scuSMCtcRegTest’ External Interface

17.2.13.4 Internal interfaces

17.2.14 Function ‘scuSMPbusRegTest’

17.2.14.1 Purpose

The purpose of ‘scuSMPbusRegTest’ (SCU Start-up Memory Diagnostics Processor Bus (P-Bus) Register Test) is to test the read/write integrity of certain P-Bus registers. It also verifies that the P-Bus’ Port Module (PM) Reset, PM Interrupt and Core Memory parity error registers are initialized to zero after reset. This test assumes that it is called after the SCU has been reset.

17.2.14.2 Function

‘scuSMPbusRegTest’ performs the following functions:

1. Initializes result to OK;
2. Calls function ‘scuFastReset’ to reset the SCU;
3. Reads the PM Reset register and if all bits are not zero then it sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that P-Bus’ PM Reset register was not initialized correctly;
4. Reads the PM Interrupt register and if all bits are not zero then it sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that P-Bus’ PM Interrupt register was not initialized correctly;
5. Reads the Core Memory parity error count register and if it is not zero then this function sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that P-Bus’ Core Memory parity error count register was not initialized correctly;
6. Calls function ‘scmRegTest’ in module ‘scmUtil.c’ passing it the address of the CTC’s PM Reset register, a region width of two bytes and the CTC’s PM Reset register’s mask. If ‘scmRegTest returns ERROR then this function sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that CTC’s PM Reset register testing failed;
7. Returns result.

17.2.14.3 External interfaces

Function ‘scuSMPbusRegTest’ is invoked by ‘scuSuDiagCtl’ in module ‘scuSuDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

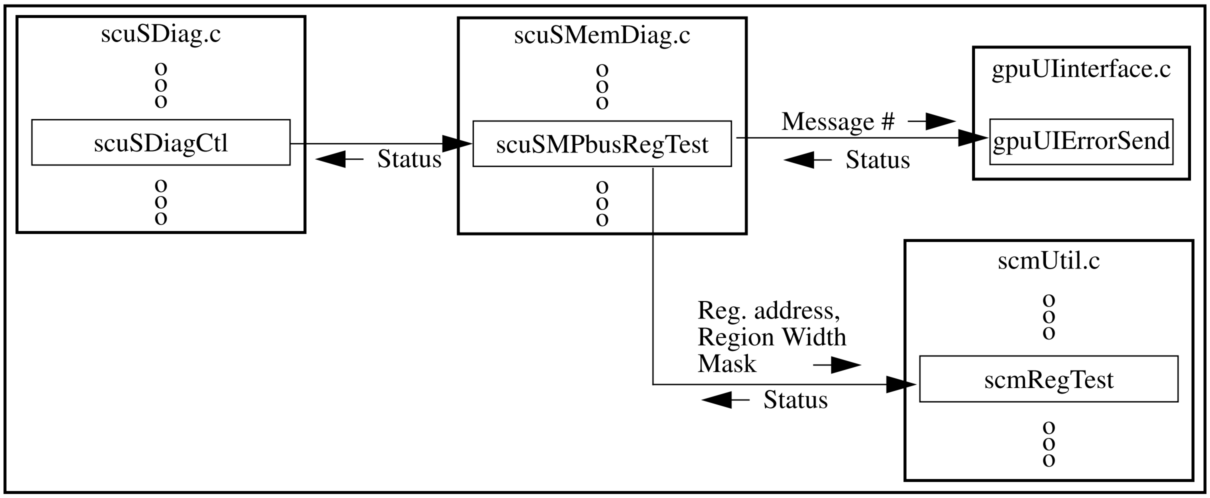

Figure 15 Function ‘scuSMPbusRegTest’ External Interface

17.2.14.4 Internal interfaces

17.2.15 Function ‘scuSMBtcRegTest’

17.2.15.1 Purpose

The purpose of ‘scuSMBtcRegTest’ (SCU Start-up Memory Diagnostics Bus-tap Control (BTC) Register Test) is to verify the read/write integrity of the BTC Setup register and that the CBT error counter is 0 after reset. (WARNING: This test must not set the bit corresponding to BTC Setup[4] on the backplane such that it places BTC Setup[3:0] into Lattice/Altera device programming mode.)

17.2.15.2 Function

‘scuSMBtcRegTest’ performs the following functions:

1. Initializes result to OK;
2. Calls function ‘scuFastReset’ to reset the SCU;

3. Reads the CBT error counter register and if it is not zero then it sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that CBT error counter register was not initialized correctly;
4. Calls function ‘scmRegTest’ in module ‘scmUtil.c’ passing it the address of the BTC Setup register, a region width of two bytes and the BTC Setup register’s mask (without BTC Setup[4] set). If ‘scmRegTest returns ERROR then this function sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number indicating that BTC Setup register testing failed;
5. Returns result.

17.2.15.3 External interfaces

Function ‘scuSMBtcRegTest’ is invoked by ‘scuSuDiagCtl’ in module ‘scuSuDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

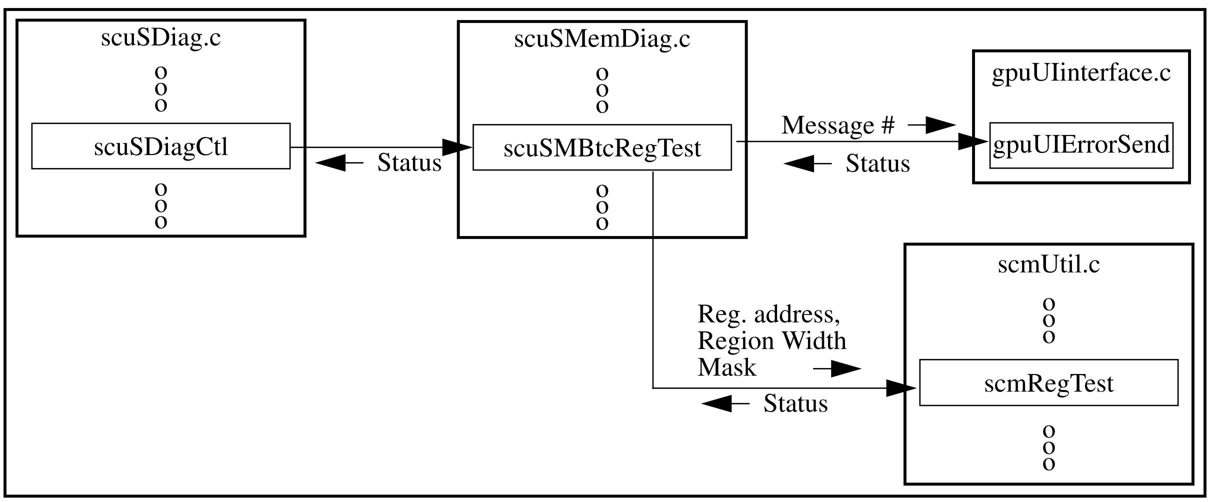

Figure 16 Function ‘scuSMBtcRegTest’ External Interface

17.2.15.4 Internal interfaces

17.3 Module ‘scuSDiag.c’

The functions in this module perform basic functionality tests for start-up diagnostics and H/W verification. Whenever the S/W requirements start-up diagnostics and H/W verification differ, these differences are satisfied using conditional compilation techniques. Since the tests in this module are much more complex than those in ‘scuSMDiag.c’, their functionality is only discussed at a very high level. This module also contains the SCU start-up diagnostics control S/W.

17.3.1 Function ‘scuSDiagCtl’

17.3.1.1 Purpose

The purpose of ‘scuSDiagCtl’ (SCU Start-up Diagnostics Controller) is to control the execution of diagnostics when they are invoked from the VxWorks Shell or other software such as function ‘scmSysInitCtl’. It provides users the ability to invoke any subset of SCU Diagnostics start-up tests by passing it an appropriate test mask. It also sends 32-bit status codes to the workstation UI to inform users of the results of the test.

17.3.1.2 Function

‘scuSDiagCtl’ performs the following functions:

1. Initializes result to OK;
2. Compares the values of the SCU Test Mask with acceptable values.[1] If it specifies an unimplemented test then it:
    - Sends a message to the switch UI indicating that the test mask specifies currently unimplemented tests;
    - Clears the bits in the test mask associated with unimplemented tests (those reserved for future use);
    - Informs the user of the value of the test mask that is being used once bits associated with unimplemented test have been cleared;
3. For each bit set in the test mask:
    - Invokes the corresponding test;
    - If the test returns ERROR then it sends a message to the switch UI indicating the test failed and sets result to ERROR;
4. Returns result;

[1]This must be done since some bits in the SCU Test Mask are reserved for future use and may be accidently set by the user or calling function.

17.3.1.3 External interfaces

‘scuSDiagCtl’ is invoked externally by function ‘scmSysInitCtl’ which passes it the value of the SCU Test Mask. For more information see "SCU Diagnostics Test Mask" on page 69. It also invokes function ‘gpuUIErrorSend’ to send messages to the switch UI. For more information see "Function ‘scuSDiagCtl’ External Interface" on page 108.

‘scuSDiagCtl’ may also be invoked from the VxWorks Shell. For more information see "External Interface Via VxWorks Shell" on page 185.

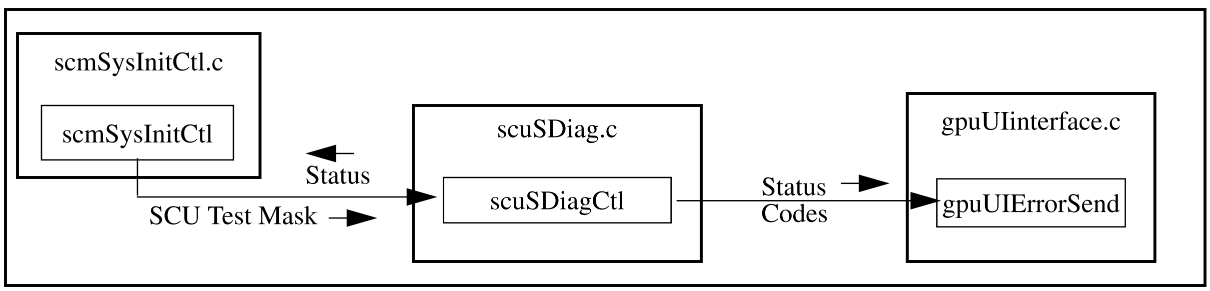

Figure 17 Function ‘scuSDiagCtl’ External Interface

17.3.1.4 Internal interfaces

‘scuSDiagCtl’ internally invokes the SCU Diagnostics test functions. They in turn return STATUS (OK or ERROR) when they complete.

17.3.2 Function ‘scuSCifFifoTest’

17.3.2.1 Purpose

The purpose of function ‘scuSCifFifoTest’ (SCU Start-up FIFO Test) is to verify that both WIVC FIFOs and the RIVC FIFO capture data correctly based upon their configuration in the WIVC and RIVC Lookup Tables as well as verify the data integrity of the FIFOs memory and data paths. This test also verifies the functionality of the status bits associated with each FIFO and its boundary conditions. Also, this test verifies the ability of the SCU to interrupt the GPU when each FIFO reaches the mid-range and almost-full levels. (This test assumes that the CRM can assert W_CE_NULL during the CIF cell write slot when the SCM’s Upper BF Threshold and RPI threshold are both 0 and the CIF performs a cell write.)

17.3.2.2 Function

Function ‘scuSCifFifoTest’ performs the following:

1. Initializes result to OK;
2. Resets the SCU;

3. Initializes the FLM Free List with all 16K CADRs and re-initializes it whenever necessary during the test to avoid relying on the FLM to recycle CADRs;
4. Sets all CRM port thresholds for the SCM to their maximum values;
5. Sets the Core Memory BF Threshold in the CRM to 16K;
6. Initializes several arrays of cell test patterns (CADRs and IVCs) for use during the test;
7. Installs a test Interrupt Service Routines to service the six interrupts for the almost full and full interrupts for each of the three FIFOs;
8. Sets up four IVCs, one with its high priority W_IVC FIFO enable bit set, one with its low priority W_IVC FIFO enable bit set, one with its R_IVC FIFO enable bit set and one without out any FIFO enable bits set;
9. Verifies that IVC data is not captured by the high priority W_IVC FIFO and low priority W_IVC FIFO under the following cell write cycle conditions:
    - The W_VALID signal is not asserted;
    - The IVC used during the cell write cycle is not enabled in the W_IVC lookup table;
    - The W_CE_NULL signal is asserted;
10. Verifies that IVC data is not captured by the R_IVC FIFO under the following cell read cycle conditions:
    - The R_VALID signal is not asserted;
    - The IVC used during the cell read cycle is not enabled in the R_IVC lookup table;
11. Sets up all IVCs to be captured by only the high priority W_IVC FIFO
12. Fills the high priority W_IVC FIFO with IVCs from 0 to (8K - 1) and CADRs from 0 to (16K - 1), by performing cell write cycles appropriately, then reads them out of the FIFO checking the following on the way:
    - The FIFOs status bits increment and decrement correctly when and only when they are supposed to;
    - The GPU is interrupted when the FIFO reaches the almost full level;
    - The GPU is interrupted when the FIFO reaches the full level;
    - The values read from the FIFO match the values that were placed in the FIFO;
13. Attempts to fill the high priority W_IVC FIFO with (4K + 1) entries and read them all back to verify the following:
    - No more than 4K entries can be placed in the FIFO;
    - Attempting to place another entry into the FIFO when it already contains 4K entries does not cause the FIFO's status bits to change from 11;
    - Attempting to read another entry into the FIFO when it is already empty does not cause the FIFO's status bits to change from 00;

14. For several different arrays of test patterns, this function fills the high priority W_IVC FIFO with the arrays' test patterns and then reads them back verifying that the data read from the FIFO matches the data in the array of test patterns;
15. Repeats 11 through 14 for the low priority W_IVC FIFO;
16. Repeats 11 through 14 for the R_IVC FIFO. Data is instead placed in the FIFO by performing cell read cycles appropriately;
17. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to switch UI by calling function 'gpuUIErrorSend' passing it a message number that translates into a string which clearly indicates the cause of the error.

17.3.2.3 External interfaces

Function 'scuSCifFifoTest' is invoked by 'scuSDiagCtl' in module 'scuSDiag.c' during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function 'scuSMCifRegTest' External Interface" on page 73. This function also invokes functions 'intVecGet' and 'intConnect' in the VxWorks module 'intLib.c' [2].

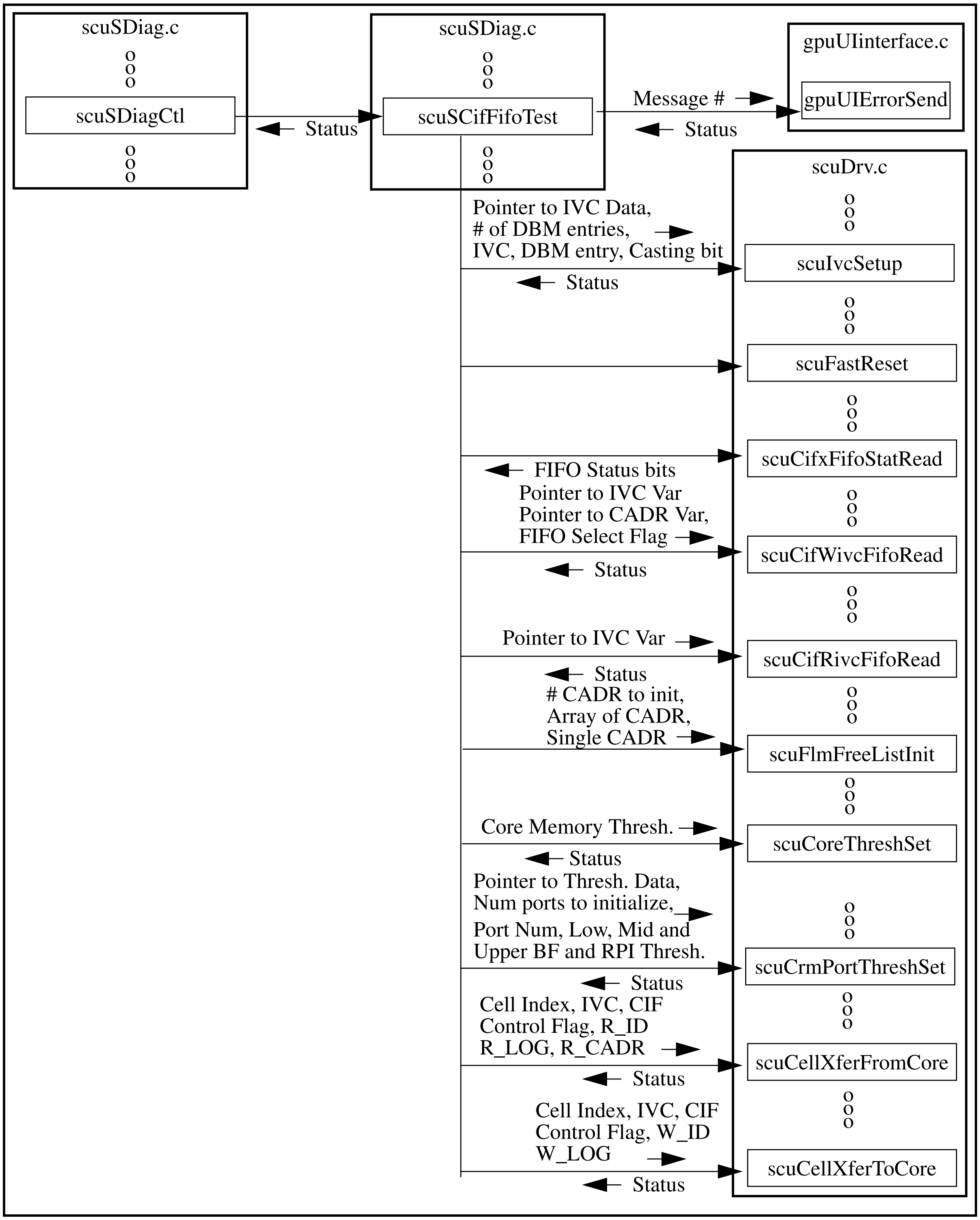

Figure 18 Function 'scuSCifFIfoTest' External Interface

17.3.2.4 Internal interfaces

Function ‘scuSCifFifoTest’ is invoked internally from function ‘scuSDiagCtl’ during SCU Start-up Diagnostics.

17.3.3 Function ‘scuSCifDataPathTest’

17.3.3.1 Purpose

The purpose of function ‘scuSCifDataPathTest’ (SCU Start-up CIF Data Path Test) is to test the integrity of the CIF’s ingress and egress paths. This test also verifies correct byte ordering of cells transferred to and from the Core Memory. It also checks that several bits in the CIF Control register clear automatically after write and read cycles and that data is not transferred to or from the Core Memory on cycles which are not valid. It also verifies that certain backplane integrity bits in the CIF Status register are not set when they should not be.

17.3.3.2 Function

Function ‘scuSCifDataPathTest’ performs the following:

1. Initializes result to OK;
2. Resets the SCU;
3. Sets all CRM port thresholds for the SCM to their maximum values;
4. Sets the Core Memory BF Threshold in the CRM to 16K;
5. Initializes the FLM Free List with only four CADRs;
6. Performs a valid cell write cycle followed by a valid cell read cycle using a cell which has each byte filled with its own byte number. The cell is placed in the cell buffer using 32-bit word writes and read from it using byte reads. Check for the following:
    - Correct byte ordering;
    - Clearing of the Write One Cell and Write Cycle Valid bits in the CIF Control register after the cell write cycle;
    - Clearing of the Read One Cell and Read Cycle Valid bits in the CIF Control register after the cell read cycle;
    - The CRC8 error bit in the CIF Status register is not set;
7. Perform a valid cell write cycle followed by an invalid read cycle and verify that the cell was not read from the Core Memory;
8. Perform an invalid cell write cycle followed by a valid read cycle and verify that the cell was not written to the Core Memory;

9. Perform a valid cell write cycle which results in the W_CE_NULL signal being asserted and follow this by a valid cell read cycle to verify that the cell is not written to the Core Memory. The W_CE_NULL signal is asserted by setting all CRM Port thresholds for port 0 to zero before performing the write cycle;
10. Initializes several arrays of cell test patterns for use during the test;
11. Perform a series of valid cell write cycles and valid cell read cycles using the cell patterns created in 10. Ensure that the entire CIF Shared Buffer is used in the process in order to properly test Shared Buffer access from the CIF Data path side:
    - Verify cell data written to the Core Memory matches that read from the Core Memory;
    - After valid cell write cycles verify that the W_IVC parity error bit in the CIF Status register is not set;
    - After valid read cycles verify that the R_IVC parity error bit and the R_CADR parity error bit are not set in the CIF Status register;
    - The CRC8 error bit in the CIF Status register is not set;
    - Reinitialize the FLM Free List whenever necessary to avoid relying on the FLM and CDM to recycle CADRs;
12. Returns result.

All valid cell read and write cycles are performed with the CRC8 enable bit in the CIF Control register set.

Whenever errors are encountered, this function sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that translates into a string which clearly indicates the cause of the error.

17.3.3.3 External interfaces

Function ‘scuSCifDataPathTest’ is invoked by ‘scuSDiagCtl’ in module ‘scuSDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

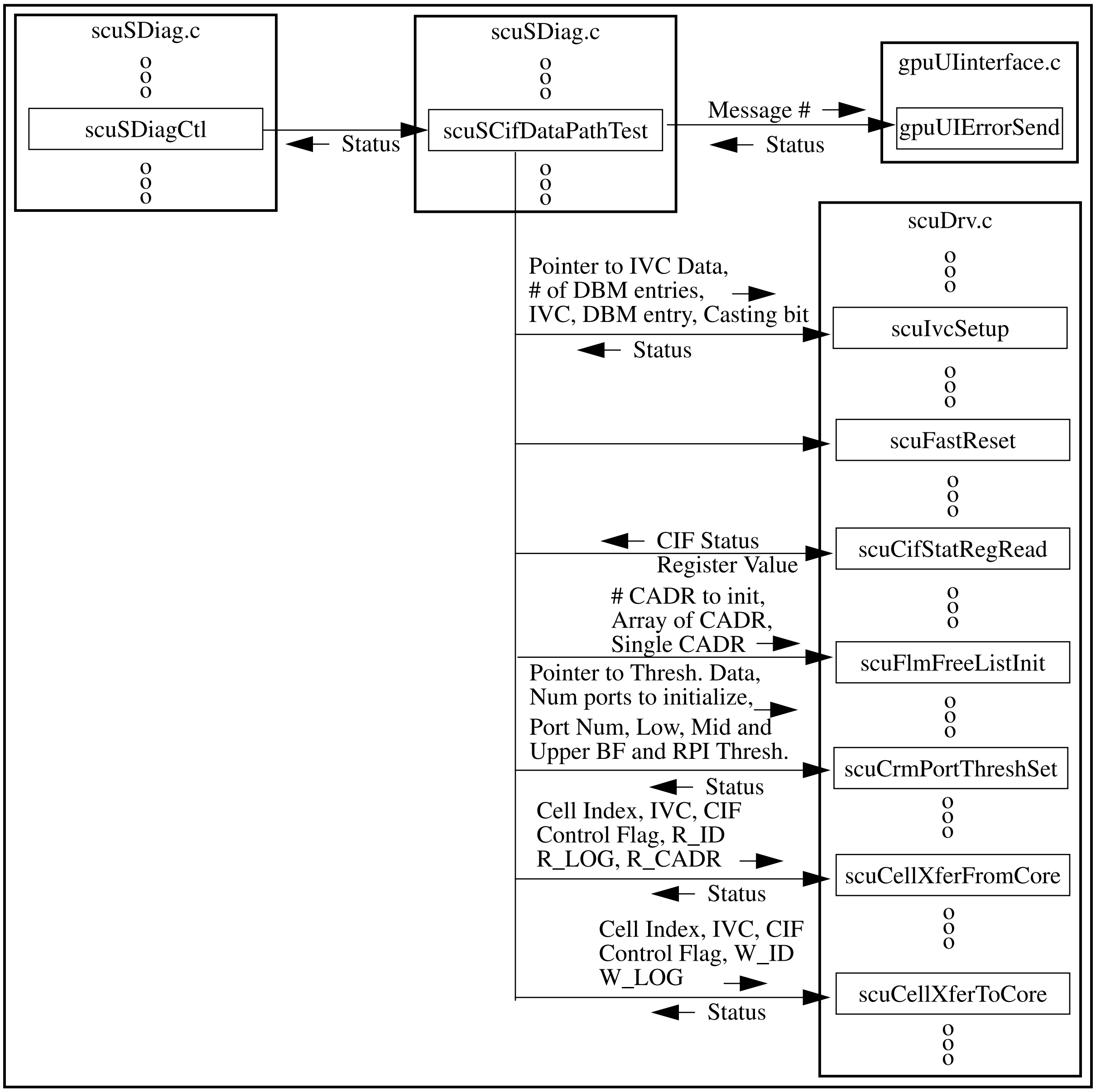

Figure 19 Function ‘scuSCifDataPathTest’ External Interface

17.3.3.4 Internal interfaces

Function ‘scuSCifDataPathTest’ is invoked internally from function ‘scuSDiagCtl’ during SCU Start-up Diagnostics.

17.3.4 Function ‘scuSCifCrc8Test’

17.3.4.1 Purpose

The purpose of function ‘scuSCifCrc8Test’ (SCU Start-up CIF Crc8 Checking/Generation Test) is to verify that the CIF’s CRC8 checker/generator correctly generates a cells CRC8 and detects CRC8 errors on cells only when they occur. It also ensures that the CRC8 interrupt and CRC8 Error bit in the CIF Status register work correctly and that the CRC8 check/generate bit in the CIF Control register automatically clears after cell read and write cycles. Also, this test verifies that a cell’s CRC8 is not generated or checked on cell write and read cycles that do not have the CRC8 enable bit in the CIF Control register set. This test does not perform an exhaustive test of the CRC8 calculation circuit using many cell data patterns since this is done in function ‘scuCifDataPath’.

17.3.4.2 Function

Function ‘scuSCifCrc8Test’ performs the following:

1. Initializes result to OK;
2. Installs a test Interrupt Service Routine to service the CRC8 error interrupt from the CIF;
3. Resets the SCU;
4. Initializes an array of cell test patterns for use during the test;
5. Sets all CRM port thresholds for the SCM to their maximum values;
6. Sets the Core Memory BF Threshold in the CRM to 16K;
7. Initializes the FLM Free List with only eight CADRs;
8. Performs a valid cell write cycle with CRC8 enabled and then a valid cell read cycle with CRC8 enable to read from the same CADR as that written and checks for the following:
    - The CRC8 enable bit in the CIF Control register remains set after the cell write cycle;
    - The CRC8 enable bit in the CIF Control register remains set after the cell read cycle;
    - A CRC8 error interrupt has not been received;
    - The CRC8 error bit in the CIF Status register has not been set;
9. Performs a valid cell write cycle without CRC8 enabled and then a valid cell read cycle with CRC8 enabled to read from the same CADR as that written and checks for the following:
    - The CRC8 enable bit in the CIF Control register remains clear after the cell write cycle;
    - A CRC8 error interrupt has been received;
    - The CRC8 error bit in the CIF Status register has been set;

10. Performs a valid cell write cycle without CRC8 enabled and then a valid cell read cycle without CRC8 enable to read from the same CADR as that written and checks for the following:
    - The CRC8 enable bit in the CIF Control register remains clear after the cell read cycle;
    - A CRC8 error interrupt has not been received;
    - The CRC8 error bit in the CIF Status register has not been set;
11. Performs a valid cell write cycle with CRC8 enabled and then a valid cell read cycle without CRC8 enable to read from the same CADR as that written and checks for the following:
    - A CRC8 error interrupt has not been received;
    - The CRC8 error bit in the CIF Status register has not been received;
12. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to switch UI by calling function 'gpuUIErrorSend' passing it a message number that translates into a string which clearly indicates the cause of the error.

17.3.4.3 External interfaces

Function 'scuSCifCrc8Test' is invoked by 'scuSDiagCtl' in module 'scuSDiag.c' during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function 'scuSMCifRegTest' External Interface" on page 73. This function also invokes functions 'intVecGet' and 'intConnect' in the VxWorks module 'intLib.c' [2]

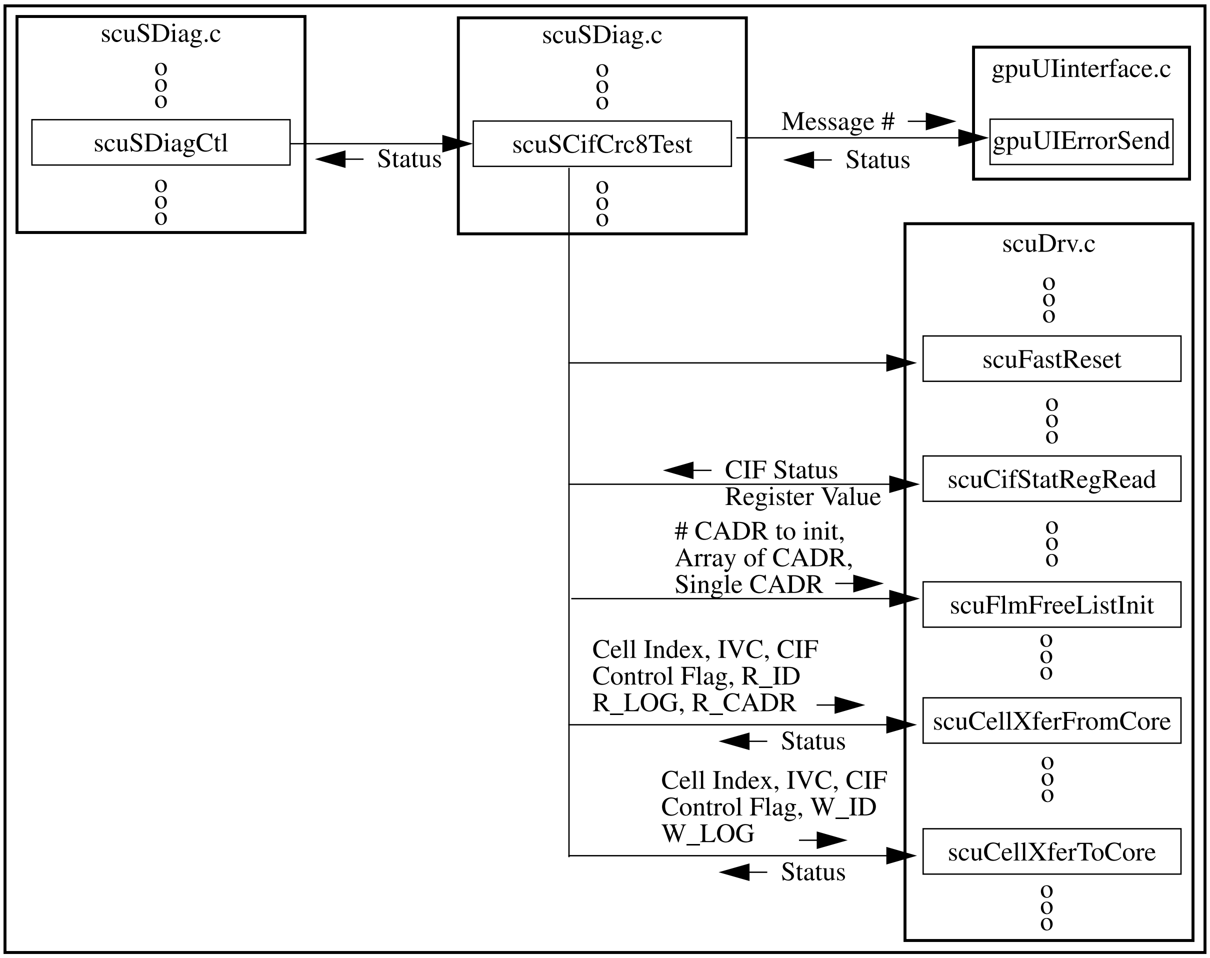

Figure 20 Function ‘scuSCifCrc8Test’ External Interface

17.3.4.4 Internal interfaces

Function ‘scuSCifCrc8Test’ is invoked internally from function ‘scuSDiagCtl’ during SCU Start-up Diagnostics.

17.3.5 Function ‘scuSCdmMonFATest’

17.3.5.1 Purpose

The purpose of function ‘scuSCdmMonFATest’ (SCU Start-up CDM Monitoring and Free Address Signalling Test) is to verify that the CDM correctly modifies the wDBM RAM during cell read cycles performed by the SCU. It also verifies that the wDBM RAM is not modified on inappropriate cell write and read cycles and that the Free Address (FA) signal is only asserted on the appropriate cell read cycles.

17.3.5.2 Function

Function ‘scuSCdmMonFATest’ performs the following:

1. Initializes result to OK;
2. Resets the SCU;
3. Initializes three IVCs to have DBMs of 0x0, 0x1, and 0xFFFF;
4. Sets all CRM port thresholds for the SCM to their maximum values;
5. Sets the Core Memory BF Threshold in the CRM to 16K;
6. Initializes the FLM Free List to have only CADR 0x0;
7. Performs a valid cell write cycle to transfer the DBM of 0x0 to the wDBM RAM at location 0x0;
8. Performs a valid cell read cycle of CADR 0x0 and checks to make sure that the FA signal was not asserted by verifying that the FLM Empty bit in the CIF Status register is set;
9. Initializes the FLM Free List to have only CADR 0x0;
10. Performs a valid cell write cycle with the IVC having a DBM of 0xFFFF such that W_CE_NULL is asserted then checks the wDBM RAM to ensure that the DBM of 0xFFFF was not copied into it;
11. Performs 16K invalid cell write cycles with the DBM Capture bit set and and reads the Requested DBM Register to ensure that the DBM of 0xFFFF was not copied into wDBM RAM;
12. Initializes the FLM Free List to contain all 16K CADRs;
13. Performs 16K valid cell write cycles to fill the wDBM RAM with the DBM 0x1;
14. Performs an invalid cell read cycle of CADR 0x0 then verifies that the corresponding wDBM is still 0x1 and that the FLM Empty bit in the CIF Status Register is set;
15. Performs 16K valid cell read cycles then verifies that the FLM Full bit in the CIF Status register is set;
16. Captures all wDBM RAM entries to ensure that they contain 0x0;

17. Performs 16K valid cell write cycles that can be captured in the high priority W_IVC FIFO and checks for the following:
    - CADRs read from the FIFO correspond to the CADRs used for the cell read cycles in 15;
18. Fills two 8K arrays with DBM patterns which have bit 0 set (corresponding to the SCM);
19. Initializes the FLM Free List to contain all 16K CADRs;
20. Initializes the DBM RAM with the first array of DBM patterns;
21. Performs a series of valid cell write cycles to copy the DBM RAM into the first 8K of the wDBM RAM;
22. Initializes the DBM RAM with the second array of DBM patterns;
23. Performs a series of valid cell write cycles to copy the DBM RAM into the second 8K of the wDBM RAM;
24. Performs valid cell read cycles with all 16K CADRs;
25. Verifies that the FLM Empty bit is set;
26. Captures all wDBM RAM entries to ensure that they have only had bit 0 cleared;
27. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that translates into a string which clearly indicates the cause of the error.

17.3.5.3 External interfaces

Function ‘scuSCdmMonFATest’ is invoked by ‘scuSDiagCtl’ in module ‘scuSDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73. This function also invokes functions ‘intVecGet’ and ‘intConnect’ in the VxWorks module ‘intLib.c’ [2].

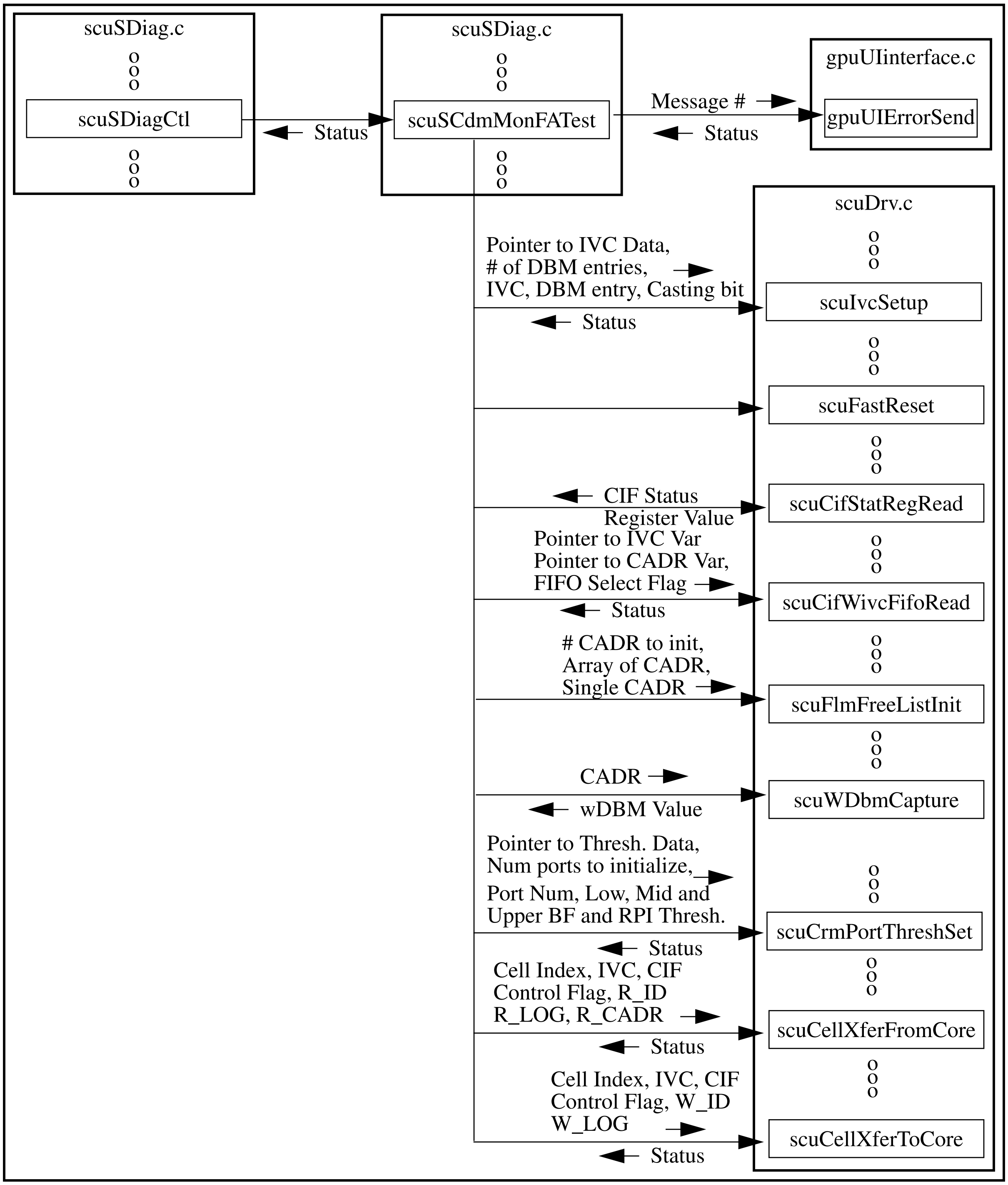

Figure 21 Function 'scuSCdmMonFaTest' External Interface

17.3.5.4 Internal interfaces

Function ‘scuSCdmMonFATest’ is invoked internally from function ‘scuSDiagCtl’ during SCU Start-up Diagnostics.

17.3.6 Function ‘scuSFlmBasicTest’

17.3.6.1 Purpose

The purpose of function ‘scuSFlmBasicTest’ (SCU Start-up FLM Basic Test) is to verify that the FLM recycles W_CADRs during cell write cycles that are invalid, have W_CE_NULL asserted or have FLM Halt asserted. It also verifies that the FLM increments/decrements the CMBF count under the proper conditions. It also verifies the functioning of the FLM Halt bit.

17.3.6.2 Function

Function ‘scuSFlmBasicTest’ performs the following:

1. Initializes result to OK;
2. Resets the SCU;
3. Clears the FLM Halt bit;
4. Sets all CRM port thresholds for the SCM to their maximum values;
5. Sets the Core Memory BF Threshold in the CRM to 16K;
6. Initializes the FLM Free List with CADRs 0x0 through 0x3;
7. Initializes two IVCs with DBMs of 0x1 and 0x3 both of which can have data captured in the high priority W_IVC FIFO;
8. Performs a cell write cycle without the Write Cycle Valid bit set in the CIF Control register and verifies that the CMBF counter does not change;
9. Performs a valid cell write cycle with the IVC having a DBM of 0x3 then checks the high priority W_IVC FIFO to verify that it has the first CADR placed on the Free List in it;
10. Performs a valid cell write cycle which has the W_CE_NULL signal asserted and verifies that the CMBF counter has not changed;
11. Performs a valid cell write cycle with the IVC having a DBM of 0x1 then checks the high priority W_IVC FIFO to verify that it has the second CADR placed on the Free List in it ;
12. Performs a cell write cycle which FLM Halt bit set then verifies that the CMBF counter has not changed and the FLM Halt bit has not been cleared after the write cycle;
13. Clears the FLM Halt bit;
14. Performs a valid cell write cycle with the IVC having a DBM of 0x1 then checks the high priority W_IVC FIFO to verify that it has the third CADR placed on the Free List in it;

15. Performs a valid cell read cycle from the first CADR then verifies that the CMBF count has not changed;
16. Performs a cell read cycle without the R_VALID signal asserted then verifies that the CMBF has not changed;
17. Sets the FLM Halt bit;
18. Performs a valid read cycle from the second CADR then verifies that the CMBF count has decremented and that the FLM Halt bit has not been cleared;
19. Clears the FLM Halt bit;
20. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that translates into a string which clearly indicates the cause of the error.

17.3.6.3 External interfaces

Function ‘scuSFlmBasicTest’ is invoked by ‘scuSDiagCtl’ in module ‘scuSDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.This function also invokes functions ‘intVecGet’ and ‘intConnect’ in the VxWorks module ‘intLib.c’ [2].

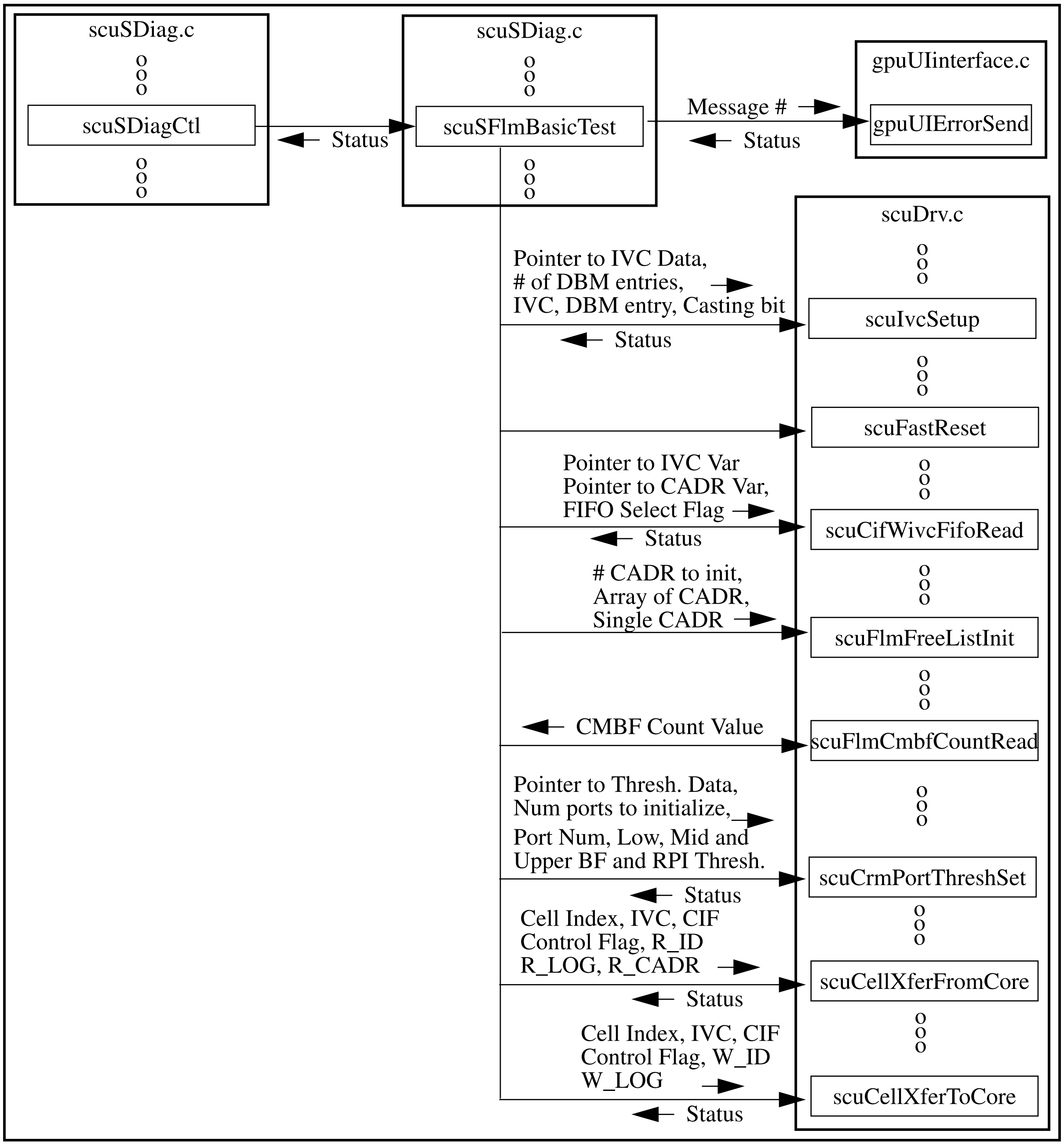

Figure 22 Function ‘scuSFlmBasicTest’ External Interface

17.3.6.4 Internal interfaces

Function ‘scuSFlmBasicTest’ is invoked internally from function ‘scuSDiagCtl’ during SCU Start-up Diagnostics.

17.3.7 Function ‘scuSFlmIntegTest’

17.3.7.1 Purpose

The purpose of function ‘scuSFlmIntegTest’ (SCU Start-up FLM Integrity Test) is to verify the integrity of the CMBF count under boundary conditions as well as verifying the integrity of the FLW, FLF and FL Empty signals.

17.3.7.2 Function

Function ‘scuSFlmIntegTest’ performs the following:

1. Initializes result to OK;
2. Installs a test Interrupt Service Routines to service the following interrupts from the FLM:
    - FLM Empty;
    - FLM Full;
    - FLM Wrap;
3. Resets the SCU;
4. Clears the FLM Halt bit;
5. Initializes one IVC with a DBM of 0x1. This IVC is used for all cell write and read cycles during this test;
6. Sets all CRM port thresholds for the SCM to their maximum values;
7. Sets the Core Memory BF Threshold in the CRM to 16K;
8. Performs a cell read cycle with the Read One Cell, Read Cycle Valid and FLM Init. bits set and checks to ensure that the CMBF counter has not changed from 0;
9. Initializes the FLM Free List with all 16K CADRs;
10. Enables the GPU to receive all FLM interrupts;
11. Performs 16K valid cell write cycles checking that the CMBF increments by one after each write;
12. Verifies that the FLM Full interrupt is received from the FLM;
13. Performs one more FLM Free List initialization placing CADR 0x0 on it;
14. Performs one more valid cell write cycle and verifies that the CMBF count has not changed from 16K;
15. Verifies that the FLM Wrap bit in the FLM Status register has been set and that the FLM Wrap interrupt has been received from the FLM;

16. Performs 16K valid cell read cycles verifying that the CMBF decrements by one after each read;
17. Verifies that the FLM Full interrupt is received from the FLM;
18. Sets the FLM Halt bit;
19. Performs 16K valid cell write cycles;
20. Performs 16K valid cell read cycles and then verifies that the FLM Empty interrupt is received;
21. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that translates into a string which clearly indicates the cause of the error.

17.3.7.3 External interfaces

Function ‘scuSFlmIntegTest’ is invoked by ‘scuSDiagCtl’ in module ‘scuSDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73. This function also invokes functions ‘intVecGet’ and ‘intConnect’ in the VxWorks module ‘intLib.c’ [2].

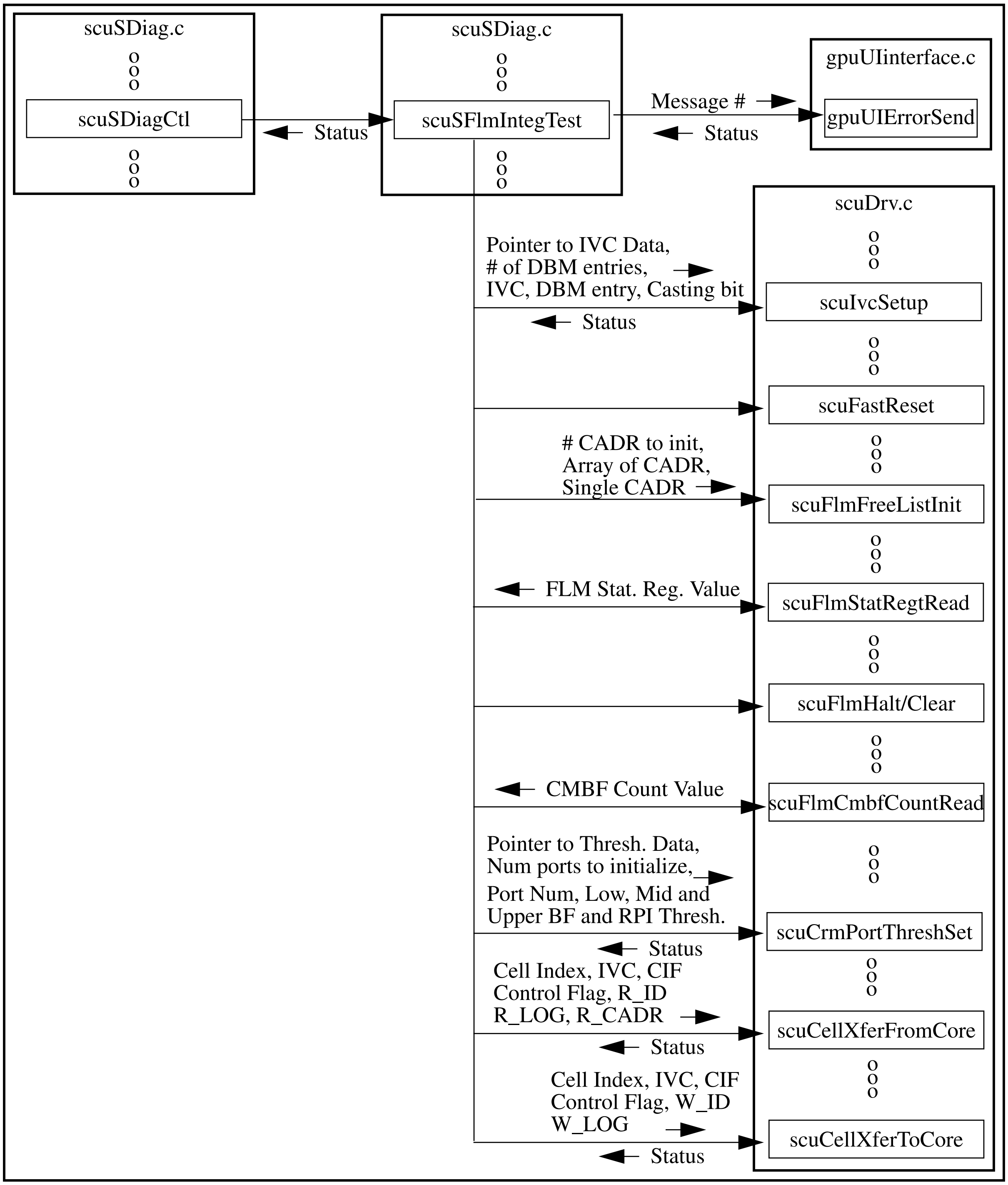

Figure 23 Function 'scuSFlmIntegTest' External Interface

17.3.7.4 Internal interfaces

Function ‘scuSFlmIntegTest’ is invoked internally from function ‘scuSDiagCtl’ during SCU Start-up Diagnostics.

17.3.8 Function ‘scuSCdmCmUpdTest’

17.3.8.1 Purpose

The purpose of function ‘scuSCdmCmUpdTest’ (SCU Start-up CDM Chalkmark Update Test) is to verify that the CDM correctly updates a CADR’s wDBM with the correct current chalkmark (unicast or multicast) but only under the correct cell read and write circumstances.

17.3.8.2 Function

Function ‘scuSCdmCmUpdTest’ performs the following:

1. Initializes result to OK;
2. Resets the SCU;
3. Clears the FLM Halt bit;
4. Sets all CRM port threshold for the SCM to their maximum values;
5. Sets the Core Memory BF Threshold in the CRM to 16K;
6. Initializes two IVCs with the following:
    - A DBM of 0x1 and a casting bit of 0x0;
    - A DBM of 0x1 and a casting bit of 0x1;
7. Initializes the FLM Free List with all 16K CADRs;
8. Set the current Multicast Chalkmark to 0x0 and the current Unicast Chalkmark to 0x1;
9. Performs 16K valid cell write cycles alternating between the two IVCs;
10. Verifies that the CADRs in the wDBM corresponding to multicast cells have a chalkmark of 0x0 and the unicast cells have a chalkmark of 0x1;
11. Set the current Multicast Chalkmark to 0x1 and the current Unicast Chalkmark to 0x0;
12. For all 16K CADRs perform the following:
    - Performs a cell read cycle without the R_VALID signal asserted;
    - Verifies that a wDBM corresponding to a multicast cell still has a chalkmark 0x0 and those corresponding to a unicast cells still have a chalkmark of 0x1;
    - Performs a valid cell read cycle;
13. Verifies that the CADRs in the wDBM corresponding to multicast cells have a chalkmark of 0x1 and the unicast cells have a chalkmark of 0x0;

14. Set the current Multicast Chalkmark to 0x0 and the current Unicast Chalkmark to 0x1;
15. For all 16K CADRs perform the following:
    - Performs a cell write cycle without the W_VALID signal asserted;
    - Verifies that a wDBM corresponding to a multicast cell still has a chalkmark of 0x1 and those corresponding to a unicast cells still have a chalkmark of 0x0;
    - Performs a valid cell write cycle;
16. Verifies that the CADRs in the wDBM corresponding to multicast cells have a chalkmark 0x0 and the unicast cells have a chalkmark of 0x1;
17. Set the current Multicast Chalkmark to 0x1 and the current Unicast Chalkmark to 0x0;
18. Performs a valid cell read cycle for all 16K CADRs;
19. Verifies that the CADRs in the wDBM corresponding to multicast cells have a chalkmark of 0x1 and the unicast cells have a chalkmark of 0x0;
20. Set the current Multicast Chalkmark to 0x0 and the current Unicast Chalkmark to 0x1;
21. For all 16K CADRs perform the following:
    - Performs a valid cell write cycle for which W_CE_NULL is asserted;
    - Verifies that a wDBM corresponding to a multicast cell still has a chalkmark 0x1 and those corresponding to a unicast cells still have a chalkmark of 0x0;
    - Performs a valid cell write cycle;
22. Verifies that the CADRs in the wDBM corresponding to multicast cells have a chalkmark of 0x0 and the unicast cells have a chalkmark of 0x1;
23. Sets both current chalkmarks to 0, performs 16K valid read cycles using all CADRs and then checks to ensure that all chalkmarks in the wDBM RAM are 0x0;
24. Sets both current chalkmarks to 1, performs 16K valid write cycles using all CADRs and then checks to ensure that all chalkmarks in the wDBM RAM are 0x1;
25. Sets both current chalkmarks to 0, performs 16K valid read cycles using all CADRs and then checks to ensure that all chalkmarks in the wDBM RAM are 0x0;
26. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that translates into a string which clearly indicates the cause of the error.

17.3.8.3 External interfaces

Function ‘scuSCdmCmUpdTest’ is invoked by ‘scuSDiagCtl’ in module ‘scuSDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73. This function also invokes functions ‘intVecGet’ and ‘intConnect’ in the VxWorks module ‘intLib.c’ [2].

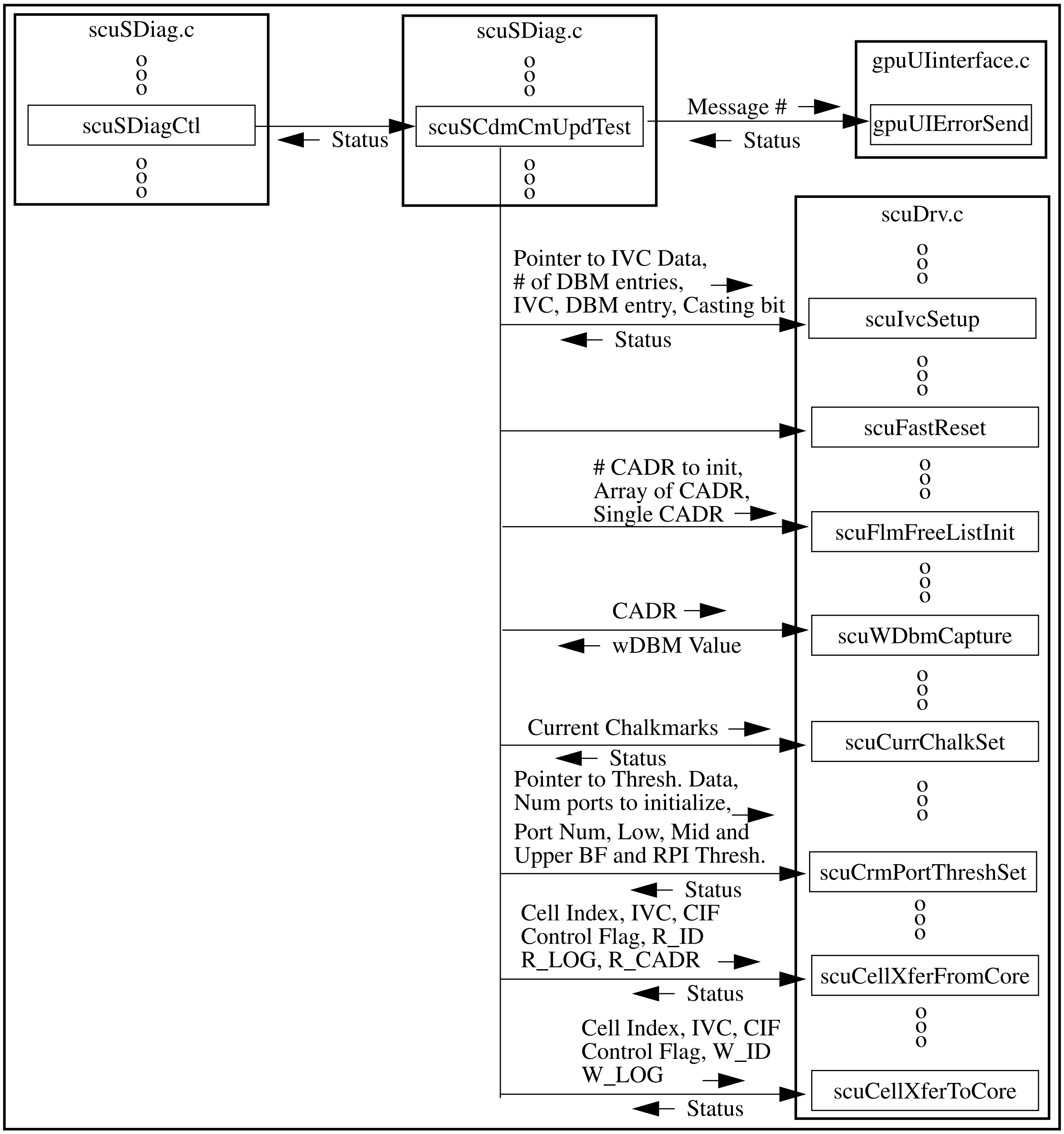

Figure 24 Function 'scuSCdmCmUpdTest' External Interface

17.3.8.4 Internal interfaces

Function 'scuSCdmCmUpdTest' is invoked internally from function 'scuSDiagCtl' during SCU Start-up Diagnostics.

17.3.9 Function 'scuSCdmCmSrchTest'

17.3.9.1 Purpose

The purpose of function 'scuSCdmCmSrchTest' (SCU Start-up CDM Chalkmark Search Test) is to verify that the CDM correctly finds all CADRs with stale chalkmarks which fit the stale chalkmark criteria. It also verifies that the correct current chalkmark is used for comparison (multicast or unicast) and that CADRs are not reported stale when a cell is not being stored at the CADR. It also tests the means by which the CDM reports the stale CADRs to the GPU.

17.3.9.2 Function

Function 'scuSCdmCmSrchTest' performs the following:

1. Initializes result to OK;
2. Resets the SCU;
3. Clears the FLM Halt bit;
4. Sets all CRM port thresholds for the SCM to their maximum values;
5. Sets the Core Memory BF Threshold in the CRM to 16K;
6. Initializes two IVCs with the following:
    - A DBM of 0x1 and a casting bit of 0x0;
    - A DBM of 0x1 and a casting bit of 0x1;
7. Initializes the FLM Free List with all 16K CADRs;
8. Installs a test Interrupt Service Routines to service the CDM's Stale CADR Interrupt;
9. Set the current Multicast Chalkmark to 0x1 and the current Unicast Chalkmark to 0x0;
10. Performs 16K valid cell write cycles alternating between the two IVCs;
11. Perform valid cell read cycles for all CADRs that have their lower three bits equal to 010 and 011;
12. Set the current Multicast Chalkmark to 0x0 and the current Unicast Chalkmark to 0x1;
13. Reset only the FLM;
14. Initialize the FLM Free List with the 4K CADRs whose lower three bits are equal to 100 and 101;
15. Perform 4K valid cell write cycles using the 4K CADRs just initialized and alternating between the two IVCs;
16. Perform 4K valid cell read cycles from the CADRs whose lower three bits are 110 and 111;
17. Enable the GPU to receive the CRM's stale CADR interrupt;

18. Initiate the chalkmark search;
19. Whenever the stale CADR interrupt occurs it performs the following:
    - Verifies that the stale chalkmark bit in the CDM Status register is set and then clears it. Verifies that the bit clears;
    - Read the stale CADR from the CDM's Stale CADR register and saves it in an array;
    - Increments a count tracking the number of stale CADRs found;
    - Re-enables the test;
20. Verifies that 4K stale CADRs were found an that they were only the ones whose lower three bits ended in 000 or 001;
21. Verifies that the chalkmark search complete bit is set at the end of the search;
22. Verifies that the chalkmark search complete bit can be cleared;
23. Repeats the search (steps 16 - 20) with the CIF transferring cells to and from the Core Memory at its full speed. It stops the search before it finds all 4K stale CADRs and resets the search to verify that the search starts again in the wDBM at location 0x0 after it is reset;
24. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to switch UI by calling function 'gpuUIErrorSend' passing it a message number that translates into a string which clearly indicates the cause of the error.

17.3.9.3 External interfaces

Function 'scuSCdmCmSrchTest' is invoked by 'scuSDiagCtl' in module 'scuSDiag.c' during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function 'scuSMCifRegTest' External Interface" on page 73. This function also invokes functions 'intVecGet' and 'intConnect' in the VxWorks module 'intLib.c' [2].

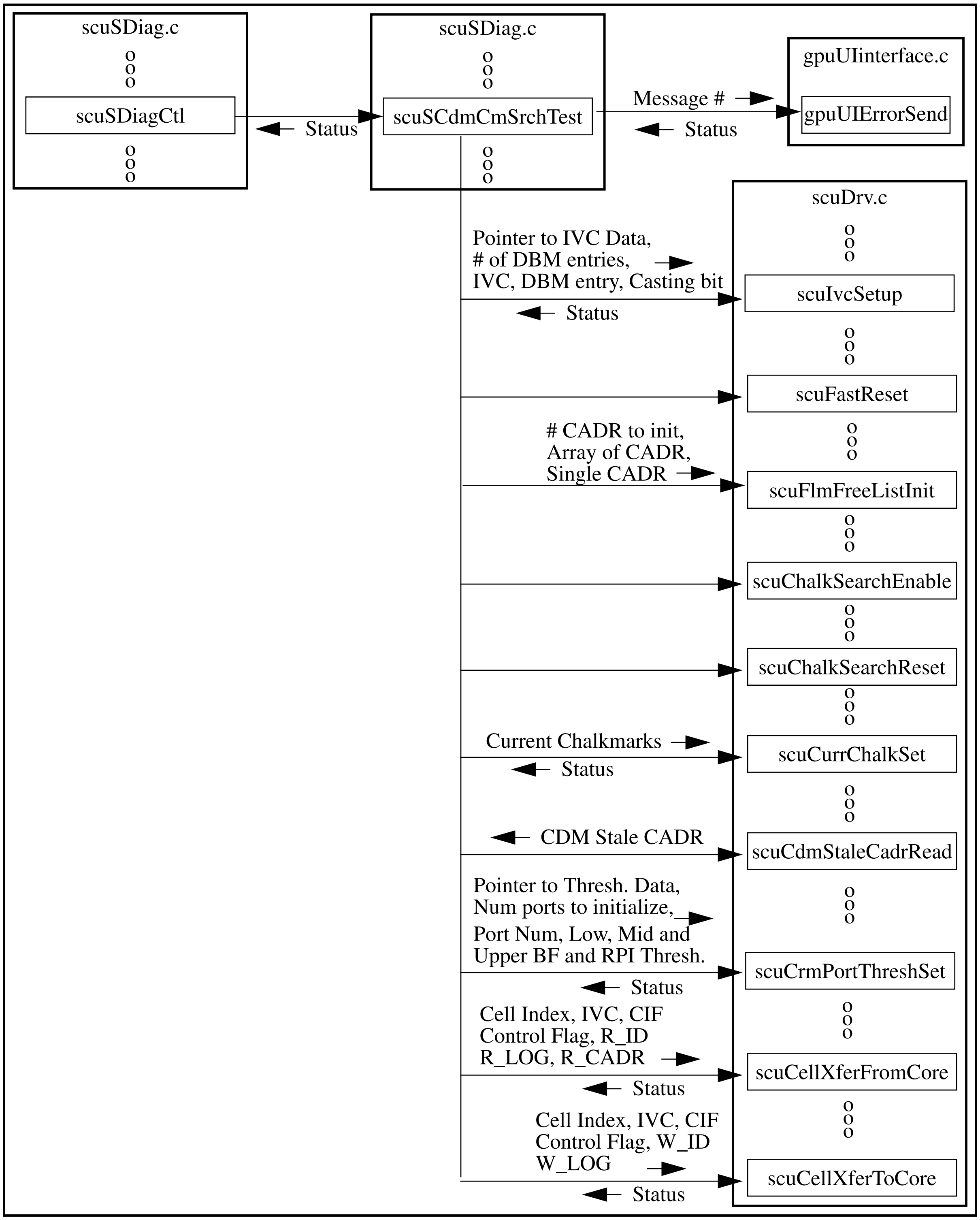

Figure 25 Function ‘scuSCdmCmSrchTest’ External Interface

17.3.9.4 Internal interfaces

Function ‘scuSCdmCmSrchTest’ is invoked internally from function ‘scuSDiagCtl’ during SCU Start-up Diagnostics.

17.3.10 Function ‘scuSCrmPortCntTest’

17.3.10.1 Purpose

The purpose of function ‘scuSCrmPortCntTest’ (SCU Start-up CRM Port BF Counter Test) is to verify that the CRM’s Port BF Counters correctly track the number of cells that each port has in the Core Memory. It also verifies that the counters do not increment or decrement under inappropriate conditions.

17.3.10.2 Function

Function ‘scuSCrmPortCntTest’ performs the following:

1. Initializes result to OK;
2. Resets the SCU;
3. Clears the FLM Halt bit;
4. Sets all CRM port thresholds for the SCM to their maximum values;
5. Sets the Core Memory BF Threshold in the CRM to 16K;
6. Initializes 32 IVCs, two per port, with each ports IVCs having the following:
    - R_CSID and W_CSID corresponding to the port’s number, a DBM of 0x1;
    - R_CSID and W_CSID corresponding to the port’s number, a DBM of 0x3;
7. Initializes the FLM Free List with all 16K CADRs;
8. For each port it performs the following with DBMs of 0x1:
    - Re-initializes the FLM Free List with all 16K CADRs;
    - Performs 15 valid cell write cycles, the first of which uses a DBM of 0x3 (at CADR 0x0);
    - Performs a cell write cycle without the W_VALID signal asserted and then verifies that the port’s counters has not changed;
    - Sets the ports BF thresholds to zero then performs a cell write cycle with W_VALID asserted, which should result in W_CE_NULL being asserted, then verifies that the port’s counter has not changed;
    - Sets the ports BF thresholds back to their maximum values then performs a cell write cycle with the W_VALID signal asserted and verifies that the port’s BF counter has

incremented by one;

- Performs a cell read cycle with the R_VALID signal asserted using the port's IVC which has a DBM of 0x1 and then verifies that the port's BF counter has decremented by one but that all other ports' BF counters have not changed;
- Performs a valid write cycle on behalf of the port and verifies that its counter has decremented;
- Performs a cell read cycle using the port's IVC which has a DBM of 0x3 and then verifies that the port's counter has not changed;
- Performs a cell read cycle without the R_VALID signal asserted using the ports IVC which has a DBM of 0x1 and then verifies that the port's counter has not changed;
- Performs (16K - 16) cell write cycles with the W_VALID signal asserted verifying that the port's BF counter increments once after each 16 write cycles;
- Performs one more cell write cycle with the W_VALID signal asserted then verifies that the port's BF counter does not change when it is already its maximum value of (16K - 1);
- Performs 16K cell read cycles with the R_VALID signal asserted verifying that the port's BF counter decrements once after each 16 read cycle;
- Performs one more cell read cycle with the R_VALID signal asserted then verifies that the port's BF counter does not decrement when it is already 0;

9. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to switch UI by calling function 'gpuUIErrorSend' passing it a message number that translates into a string which clearly indicates the cause of the error.

17.3.10.3 External interfaces

Function 'scuSCrmPortCntTest' is invoked by 'scuSDiagCtl' in module 'scuSDiag.c' during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function 'scuSMCifRegTest' External Interface" on page 73. This function also invokes functions 'intVecGet' and 'intConnect' in the VxWorks module 'intLib.c' [2].

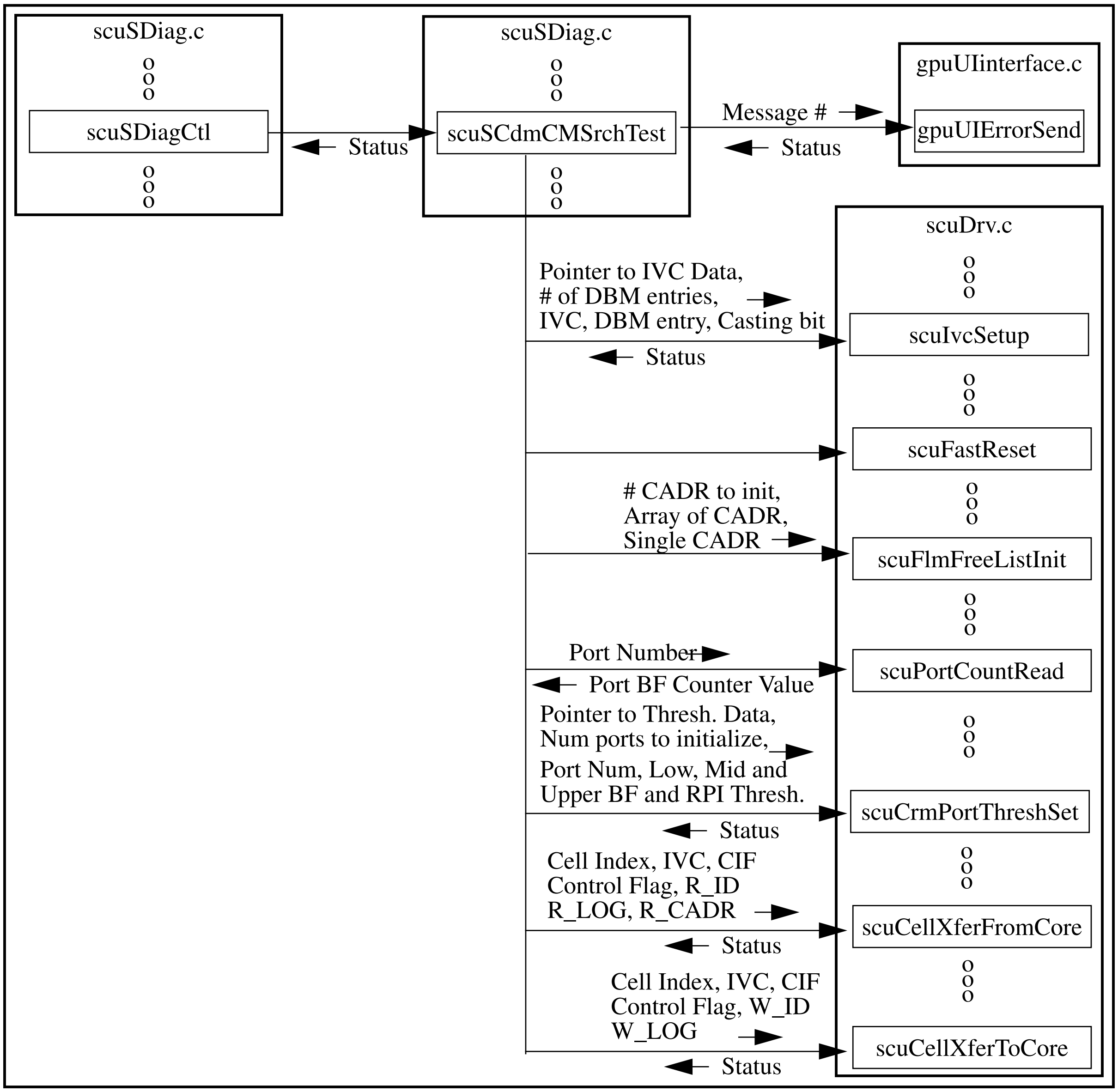

Figure 26 Function ‘scuSCrmPortCntTest’ External Interface

17.3.10.4 Internal interfaces

Function ‘scuSCrmPortCntTest’ is invoked internally from function ‘scuSDiagCtl’ during SCU Start-up Diagnostics.

17.3.11 Function 'scuSCrmProtTest'

17.3.11.1 Purpose

The purpose of function 'scuSCrmProtTest' (SCU Start-up CRM Core Memory Protection Test) is to verify that the CRM's Buffer Fill promotion is functioning and that the CRM asserts W_CE_NULL during the SCU's cell write cycles when the SCU has exceeded the highest of its Upper BF or RPI threshold.

17.3.11.2 Function

Function 'scuSCrmProtTest' performs the following:

1. Initializes result to OK;
2. Resets the SCU;
3. Clears the FLM Halt bit;
4. Set all CRM port thresholds for ports one to 15 to their maximum values;
5. Sets the Core Memory BF Threshold in the CRM to 1K;
6. Sets all CRM port thresholds for the SCM the following:
    - Lower BF threshold set to 0x10;
    - Middle BF threshold set to 0x20;
    - Upper BF threshold set to 0x40;
    - RPI threshold set to 0x80;
7. Initializes 16 IVCs, one per port, with each ports IVCs having the following:
    - R_CSID and W_CSID corresponding to the port's number, a DBM of 0x1;
8. Initializes the FLM Free List with all 16K CADRs;
9. Perform the following with the SCU to verify that it protects the Core Memory when a port exceeds its highest BF threshold:
    - Performs 0x80 cell write cycles with the W_VALID signal asserted and the IVC with a CSID of 0x0;
    - Check that the SCU's port BF count is 0x80;
    - Enables the high priority W_IVC FIFO to capture the SCU's IVC;
    - Performs one more cell write cycle with the W_VALID signal asserted and the SCU's IVC then verifies that the SCU's port counter in the CRM is still at 0x80 and that the high priority W_IVC FIFO did not capture any data pertaining to these write cycles;
    - Performs one cell read cycle with the R_VALID signal asserted;
    - Performs one more cell write cycle with the W_VALID signal asserted and the SCU's IVC then verifies that the SCU's port counter in the CRM is incrementing

again and that the high priority W_IVC FIFO did capture any data pertaining to these write cycles;

- Disables the high priority W_IVC FIFO from capturing the SCU's IVC;

10. Repeats 8. and 9. but with the SCU's Upper BF threshold set to 0x40 and its RPI threshold set to 0x80;
11. Places 0x40 cells in the Core Memory on behalf of each of the other ports using the IVCs setup in 7;
12. Performs the following with the SCU to verify that artificial buffer fill promotion is functioning:
    - Performs 0x60 cell write cycles with the W_VALID signal asserted and the IVC with an CSID of 0x0;
    - Enables the high priority W_IVC FIFO to capture the SCU's IVC;
    - Performs one more cell write cycle with the W_VALID signal asserted and the SCU's IVC then verifies that the SCU's port counter in the CRM is still at 0x60 and that the high priority W_IVC FIFO did not capture any data pertaining to these write cycles;
    - Performs one cell read cycle with the R_VALID signal asserted;
    - Performs one more cell write cycle with the W_VALID signal asserted and the SCU's IVC then verifies that the SCU's port counter in the CRM is incrementing again and that the high priority W_IVC FIFO did capture any data pertaining to these write cycles;
    - Disables the high priority W_IVC FIFO from capturing the SCU's IVC;
13. Repeats 11. and 12. but with the SCU's Upper BF threshold set to 0x80 and its RPI threshold set to 0x40;
14. Verifies again that up to 0x80 cell can be written into the Core Memory;
15. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to switch UI by calling function 'gpuUIErrorSend' passing it a message number that translates into a string which clearly indicates the cause of the error.

17.3.11.3 External interfaces

Function 'scuSCrmProtTest' is invoked by 'scuSDiagCtl' in module 'scuSDiag.c' during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function 'scuSMCifRegTest' External Interface" on page 73.

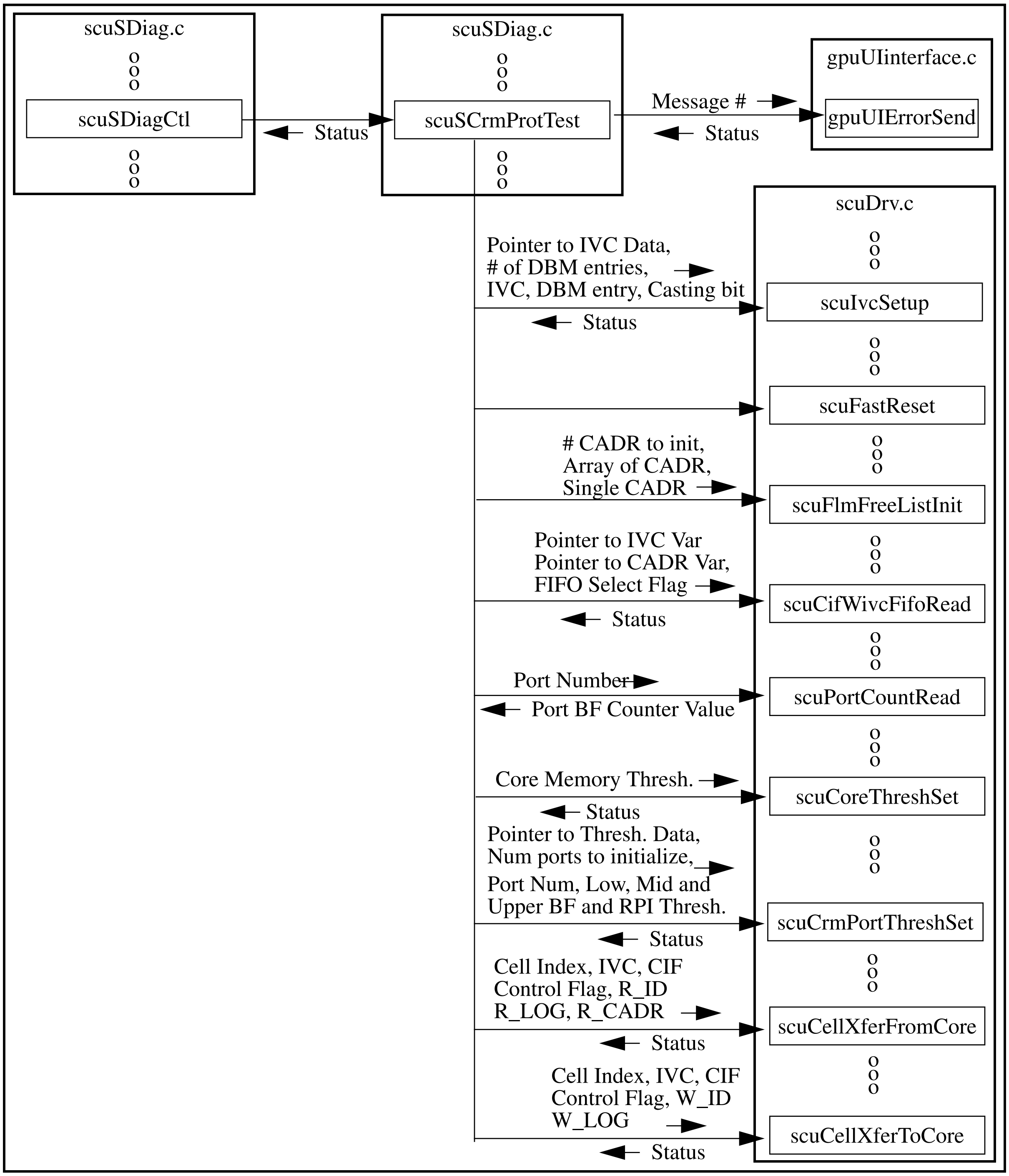

Figure 27 Function 'scuSCrmProtTest' External Interface

17.3.11.4 Internal interfaces

Function ‘scuSCrmProtTest’ is invoked internally from function ‘scuSDiagCtl’ during SCU Start-up Diagnostics.

17.3.12 Function ‘scuSCtcCptTest’

17.3.12.1 Purpose

The purpose of function ‘scuSCtcCptTest’ (SCU Start-up CTC Cell Period Timer Test) is to verify that the Cell Period Timer (CPT) continually increments once per cell period wrapping around from 0 to (16M - 1).

17.3.12.2 Function

Function ‘scuSCtcCptTest’ performs the following:

1. Initializes result to OK;
2. Resets the SCU;
3. Repeats the following for 16M cell periods:
    - Reads the Cell Period Timer;
    - Verifies that the new value of the timer is equal to old value plus one if the old value was not (16M - 1);
    - Verifies that the new value of the timer is equal to zero if the old value was (16M - 1);
    - Stores the current value of the timer as the old value;
    - Delays for the rest of the cell period;
4. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to the switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that translates into a string which clearly indicates the cause of the error.

17.3.12.3 External interfaces

Function ‘scuSCtcCptTest’ is invoked by ‘scuSDiagCtl’ in module ‘scuSDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73.

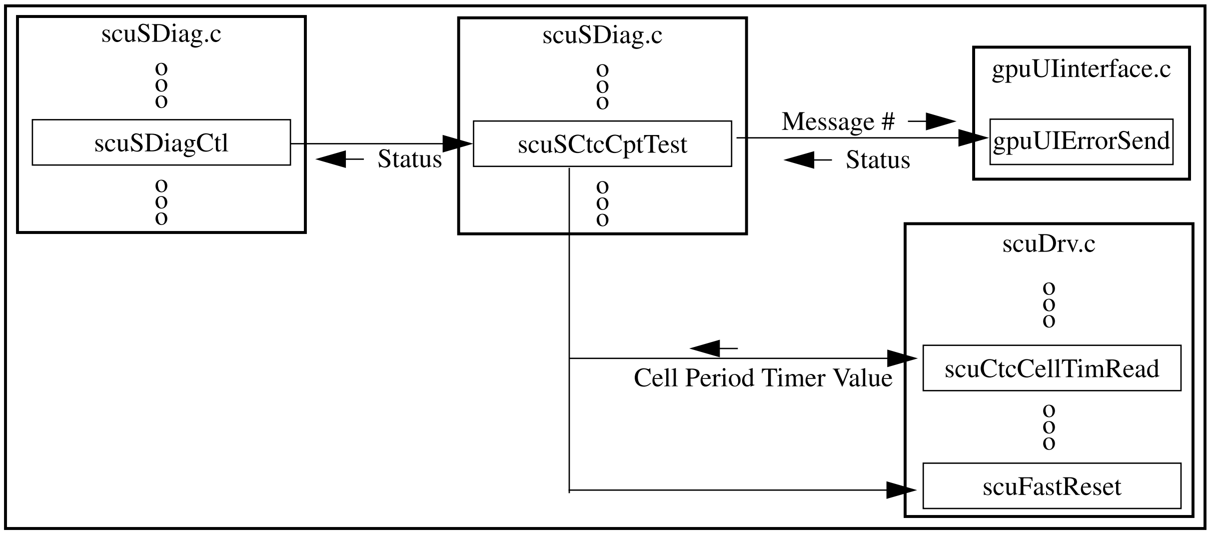

Figure 28 Function 'scuSCtcCptTest' External Interface

17.3.12.4 Internal interfaces

Function 'scuSCtcCptTest' is invoked internally from function 'scuSDiagCtl' during SCU Start-up Diagnostics.

17.3.13 Function 'scuSCtcIt1Test'

17.3.13.1 Purpose

The purpose of function 'scuSCtcIt1Test' (SCU Start-up CTC Interval Timer (IT) First Test) is to verify that the IT only starts timing under the correct conditions for which it is programmed to start, that the timer holds at its maximum value of (1M -1) when it reaches it and that the timer interrupts the GPU when it reaches its maximum value.

17.3.13.2 Function

Function 'scuSCtcIt1Test' performs the following:

1. Initializes result to OK;
2. Resets the SCU;
3. Sets all CRM port thresholds for the SCM to zero;
4. Sets the Core Memory BF Threshold in the CRM to 16K;
5. Initializes two IVCs, IVCs one and two, with the following:

- R_CSID and W_CSID corresponding to the port 0, a DBM of 0x1;

6. Initializes the FLM Free List with the first 16 CADRs;
7. Clears the FLM Halt bit;
8. Installs a test ISR to handle the IT timing complete interrupt during the test;
9. Verifies that the IT is not runnin before starting testing;
10. Programs the IT to start on a W_IVC equal to IVC one;
11. Performs a cell write cycle with the W_VALID bit asserted using IVC one then verifies that the IT has not started counting (this write cycle ends up having W_CE_NULL asserted);
12. Sets all CRM port thresholds for the SCM to their maximum values;
13. Performs a cell write cycle with the W_VALID bit unasserted using IVC one then verifies that the IT has not started counting;
14. Performs a cell write cycle with the W_VALID bit asserted using IVC two then verifies that the IT has not started counting;
15. Performs a cell write cycle with the W_VALID bit asserted using IVC one then verifies the following:
    - Verifies that the IT starts counting from 0;
    - Verifies that the IT increments once per cell period;
    - Verifies that the IT stops counting when it reaches its maximum value of (1M - 1);
    - Verifies that the GPU is interrupted when the IT reaches (1M - 1);
16. Performs a cell read cycle with the R_VALID signal asserted using IVC one then verifies that the IT has not started counting again;
17. Programs the IT to start on a R_IVC equal to IVC two;
18. Performs a cell read cycle with the R_VALID bit unasserted using IVC two then verifies that the IT has not started counting;
19. Performs a cell read cycle with the R_VALID bit asserted using IVC two then verifies that the IT has not started counting;
20. Performs a cell read cycle with the R_VALID bit asserted using IVC two then verifies the following:
    - Verifies that the IT starts counting from 0;
    - Verifies that the IT increments once per cell period;
    - Verifies that the IT stops counting when it reaches its maximum value of (1M - 1);
    - Verifies that the GPU is interrupted when the IT reaches (1M - 1);
21. Performs a cell write cycle with the W_VALID signal asserted using IVC two then verifies that the IT has not started counting again;

22. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to the switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that translates into a string which clearly indicates the cause of the error.

17.3.13.3 External interfaces

Function ‘scuSCtcIt1Test’ is invoked by ‘scuSDiagCtl’ in module ‘scuSDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73. This function also invokes functions ‘intVecGet’ and ‘intConnect’ in the VxWorks module ‘intLib.c’ [2].

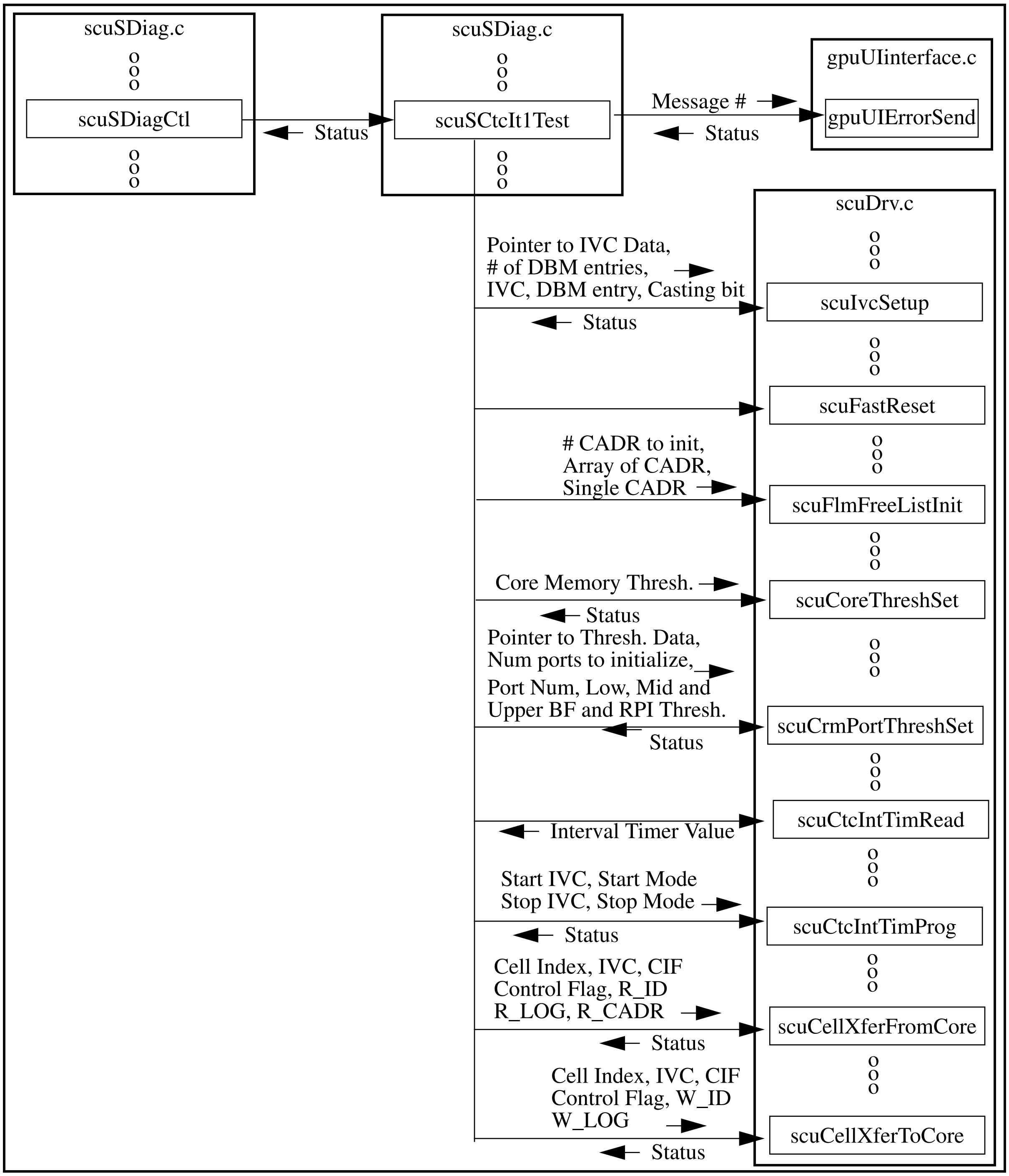

Figure 29 Function 'scuSCtcIt1Test' External Interface

17.3.13.4 Internal interfaces

Function ‘scuSCtcIt1Test’ is invoked internally from function ‘scuSDiagCtl’ during SCU Start-up Diagnostics.

17.3.14 Function ‘scuSCtcIt2Test’

17.3.14.1 Purpose

The purpose of function ‘scuSCtcIt2Test’ (SCU Start-up CTC Interval Timer (IT) Second Test) is to verify that the IT only stops under the correct conditions for which it is programmed to start, that the timer is capable of timing an interval as short as one cell period and that the timer interrupts the GPU finishes timing an interval.

17.3.14.2 Function

Function ‘scuSCtcIt2Test’ performs the following:

1. Initializes result to OK;
2. Resets the SCU;
3. Sets all CRM port thresholds for the SCM to their maximum values;
4. Sets the Core Memory BF Threshold in the CRM to 16K;
5. Initializes two IVCs, IVCs one and two, with the following:
    - R_CSID and W_CSID corresponding to the port 0, a DBM of 0x1;
6. Initializes the FLM Free List with the first 16 CADRs;
7. Clears the FLM Halt bit;
8. Installs a test ISR to handle the IT timing complete interrupt during the test;
9. Programs the IT to start on a W_IVC equal to IVC one and stop on a R_IVC equal to two;
10. Verifies that the IT started timin;
11. Performs a cell write cycle with the W_VALID signal asserted using IVC one to start the IT;
12. Performs a cell read cycle with the R_VALID signal unasserted using IVC two and verifies that the IT has not stopped counting and that the GPU has not been interrupted with the IT timing complete interrupt;
13. Performs a cell read cycle with the R_VALID signal asserted using IVC one and verifies that the IT has not stopped counting and that the GPU has not been interrupted with the IT timing complete interrupt;

14. Performs a cell write cycle with the W_VALID signal asserted using IVC two and verifies that the IT has not stopped counting and that the GPU has not been interrupted with the IT timing complete interrupt;
15. Performs a cell read cycle with the R_VALID signal asserted using IVC two and verifies that the IT has stopped counting and that the GPU has been interrupted with the IT timing complete interrupt;
16. Performs cell write cycle with W_VALID asserted using IVC one followed by a cell read cycle with the R_VALID signal asserted using IVC two on the very next CCB cell period. It then verifies that the IT timer has been stopped with a value of one and that the GPU has been interrupted with the IT timing complete interrupt;
17. Programs the IT to start on a R_IVC equal to IVC two and stop on a W_IVC equal to one;
18. Verifies that the IT started timing;
19. Performs a cell read cycle with the R_VALID signal asserted using IVC two to start the IT;
20. Performs a cell write cycle with the W_VALID signal unasserted using IVC one and verifies that the IT has not stopped counting and that the GPU has not been interrupted with the IT timing complete interrupt;
21. Performs a cell write cycle with the W_VALID signal asserted using IVC two and verifies that the IT has not stopped counting and that the GPU has not been interrupted with the IT timing complete interrupt;
22. Performs a cell read cycle with the R_VALID signal unasserted using IVC one and verifies that the IT has not stopped counting and that the GPU has not been interrupted with the IT timing complete interrupt;
23. Sets all CRM port thresholds for the SCM to zero;
24. Performs a cell write cycle with the W_VALID signal asserted using IVC one and verifies that the IT has not stopped counting and that the GPU has not been interrupted with the IT timing complete interrupt (since this write cycle ends up having W_CE_NULL asserted);
25. Sets all CRM port thresholds for the SCM to their maximum values;
26. Performs a cell write cycle with the W_VALID signal asserted using IVC one and verifies that the IT has stopped counting and that the GPU has been interrupted with the IT timing complete interrupt;
27. Performs a cell read cycle with R_VALID asserted using IVC two followed by a cell write cycle with the W_VALID signal asserted using IVC one on the very next CCB cell period then verifies that the IT timer has been stopped with a value of one and that the GPU has been interrupted with the IT timing complete interrupt;
28. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to the switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that translates into a string which clearly indicates the cause of the error.

17.3.14.3 External interfaces

Function ‘scuSCtcIt2Test’ is invoked by ‘scuSDiagCtl’ in module ‘scuSDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73. This function also invokes functions ‘intVecGet’ and ‘intConnect’ in the VxWorks module ‘intLib.c’ [2].

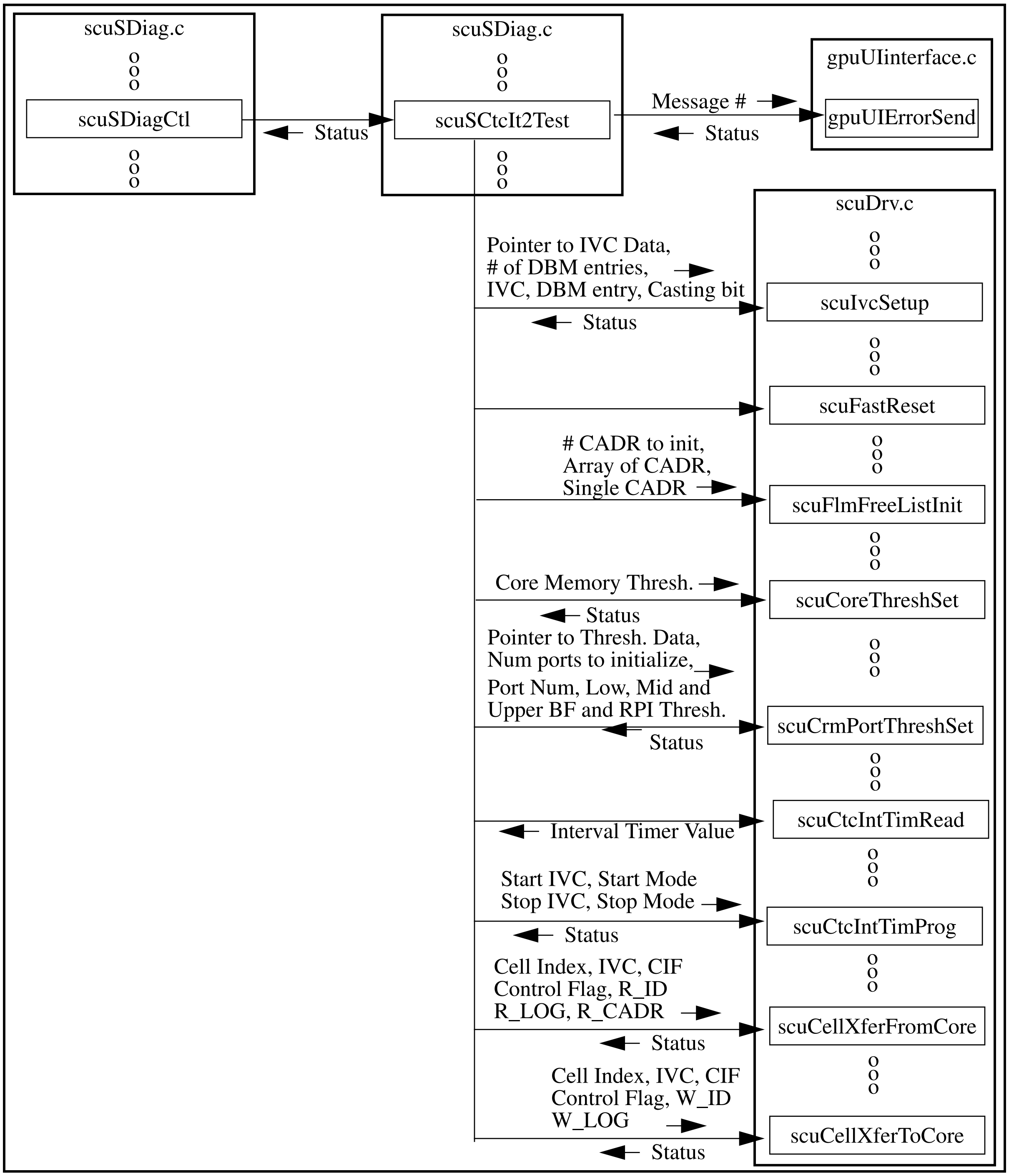

Figure 30 Function ‘scuSCtcIt2Test’ External Interface

17.3.14.4 Internal interfaces

Function ‘scuSCtcIt2Test’ is invoked internally from function ‘scuSDiagCtl’ during SCU Start-up Diagnostics.

<u>17.3.15 Function ‘scuSCbtTest’</u>

17.3.15.1 Purpose

TBD.

17.3.15.2 Function

17.3.15.3 External interfaces

17.3.15.4 Internal interfaces

17.3.16 Function ‘scuSCoreTest’

17.3.16.1 Purpose

The purpose of function ‘scuSCoreTest’ (SCU Start-up Core Memory Test) is to check the Core Memory and the data path to it for stuck bits, leakage, address decoding errors and address and data line shorts from a black box point of view.

17.3.16.2 Function

‘scuSCoreTest’ performs the following:

1. Initializes result to OK;
2. Resets the SCU;
3. Clears the FLM Halt bit;
4. Sets all CRM port thresholds for the SCM to their maximum values;
5. Sets the Core Memory BF Threshold in the CRM to 16K;
6. Initializes one IVC with a R_CSID and W_CSID of zero and a DBM of 0x1;

7. Allocates enough memory to store 16K 16 bit cell patterns;
8. Generates 16k sequential cell patterns;
9. Initializes the FLM Free List with all 16K sequential CADRs;
10. Fills the entire Core Memory with cells which are filled with their own CADRs;
11. Reads the cells from the Core Memory performing the following:
    - Verifies that each cell is filled with its own CADR. If an error is found then it prints out the CADR, the number of the 16-bit word where the error is found, the expected pattern and the pattern found;
    - Verifies that a CRC8 error did not occur;
12. Generates 16K 16 bit cell test patterns according to the following. Startup diagnostics require pseudo-random patterns while hardware verification requires deterministic patterns. A macro is used to select the cell pattern generation code. The cells are then placed in the allocated memory. In both cases the FLM is filled with sequential addresses starting from zero.
13. Fills the entire Core Memory with the cell test patterns;
14. Reads the cells from the Core Memory, traversing the CADRs in the same order in which they were just written, performing the following:
    - Reads each cell from the Core Memory and verifies that it contains the cell test pattern which was just written to it. If an error is found then it prints out the CADR, the number of the 16-bit word where the error is found, the expected pattern and the pattern found;
    - Verifies that a CRC8 error did not occur;
15. Repeats steps 11, 12 and 13 until a desired level of test coverage is achieved. Startup diagnostics require 71 iterations of random patterns to achieve 99.9% probability of stimulating memory faults. Hardware verification requires fewer iterations with a deterministic pattern.
16. Generates 16k “inverse” cell patterns starting from 0x3fff down to 0x0000
17. Initialize the FLM so that the 16K x 16 SRAM devices in the Core are written to filling up the columns first. (The SRAMs are laid out as 128 x 128 16-bit devices. Sequential accesses work their way across the rows and then down to the next row. The order would be 0x0 0x80, 0x100, 0x180,..., 0x3f80, 0x1, 0x81, 0x101, 0x181,..., 0x3f81,...,0x7f, 0xff, 0x17f, 0x1ff,..., 0x3fff)
18. Fills the entire Core Memory with cells which are filled with the ones complement of their own CADRs;
19. Reads the cells from the Core Memory, in the opposite order in which they were just written, performing the following:
    - Reads each cell from the Core Memory and verifies that it is filled with the ones complement of its own CADR. If an error is found then it prints out the CADR, the num-

ber of the 16-bit word where the error is found, the expected pattern and the pattern found;

- Verifies that a CRC8 error did not occur;

20. De-allocates the memory allocated to store the 16K 16 bit test patterns;
21. Performs a walking one test on the Core Memory data path starting at CADR 0x4;
22. Performs a walking zero test on the Core Memory data path starting at CADR 0x4;
23. Returns result;

Whenever errors are encountered, this function sets result to ERROR and sends a message to the switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that translates into a string which clearly indicates the cause of the error. The setting of dip switch two controls the flow of output to VXWorks. Turning this switch off prevents output from being sent to VXWorks.

17.3.16.3 External interfaces

Function ‘scuSCoreTest’ is invoked by ‘scuSDiagCtl’ in module ‘scuSDiag.c’ during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function ‘scuSMCifRegTest’ External Interface" on page 73. This function also invokes functions ‘intVecGet’ and ‘intConnect’ in the VxWorks module ‘intLib.c’ [2].

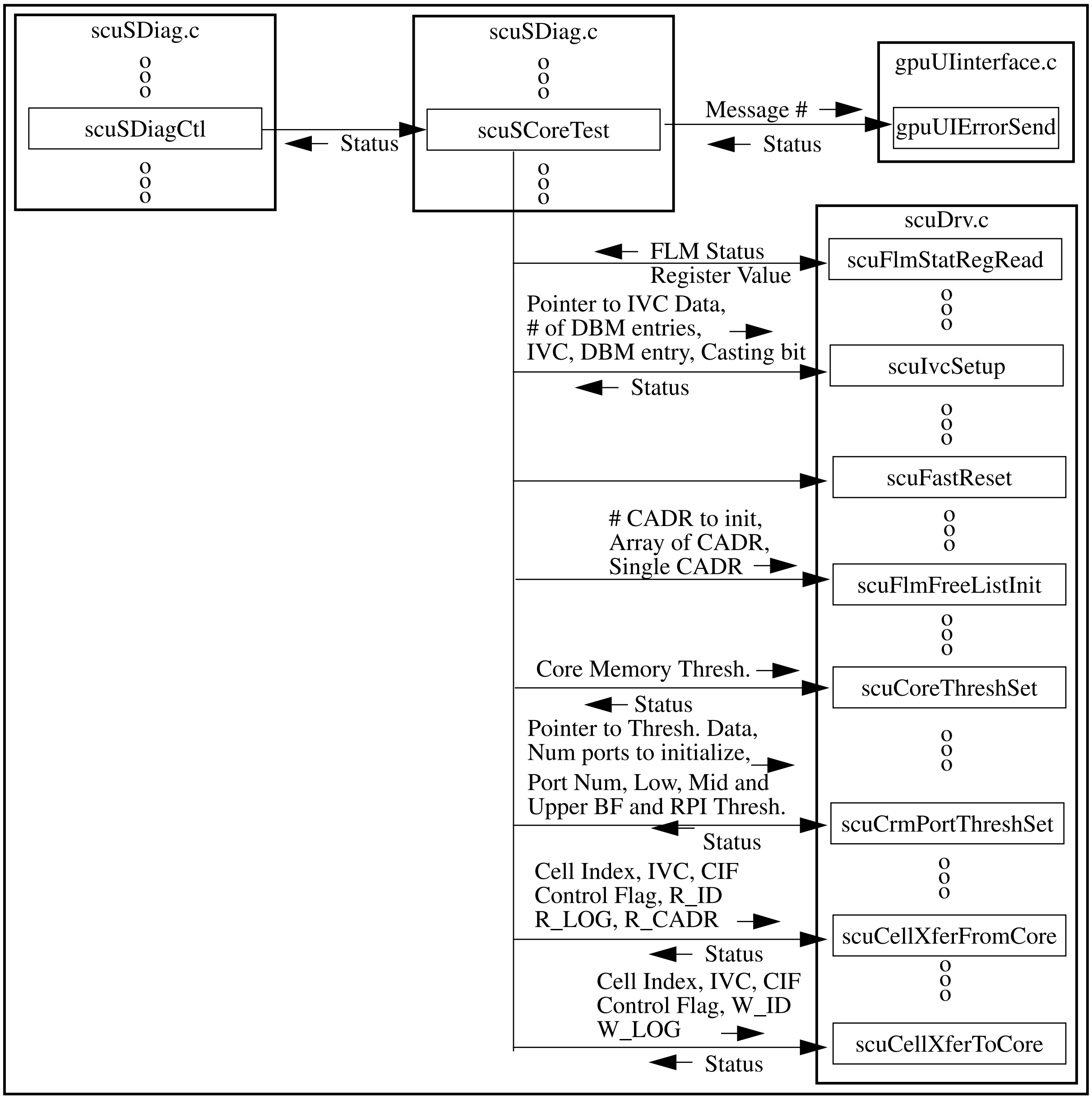

Figure 31 Function ‘scuSCoreTest’ External Interface

17.3.16.4 Internal interfaces

Function ‘scuSCoreTest’ is invoked internally from function ‘scuSDiagCtl’ during SCU Start-up Diagnostics.

17.4 Module ‘scuHVDiag.c’

The following test functions require some manual setup and intervention in order to work properly.

17.4.1 Function ‘scuHVWCadrPETest’

17.4.1.1 Purpose

The purpose of ‘scuHVWCadrPETest’ (SCU HW Verification W_CADR Parity Error Test) is to verify that the W_CADR parity error signal on the backplane does not have any affect on SCU functionality.

17.4.1.2 Function

‘scuHVWCadrPETest’ performs the following functions:

1. Prompts the user to assert the W_CADR_P backplane signal and toggle DIP Switch 2 when complete;
2. Calls function ‘scuSDiagCtl’ passing it a test mask so that it executes all tests in modules ‘scuSMDiag.c’ and ‘scuSDiag.c’;
3. Prompts the user to de-assert the W_CADR_P backplane signal and toggle DIP Switch 2 when complete;
4. Returns the result returned by ‘scuSDiagCtl’.

17.4.1.3 External interfaces

Function ‘scuHVWCadrPETest’ invokes function ‘scuSDiagCtl’ in module ‘scuSDiag.c’. It is callable from the VxWorks Shell prompt. For more information see "Function ‘scuHVWCadrPETest’ External Interface" on page 153.

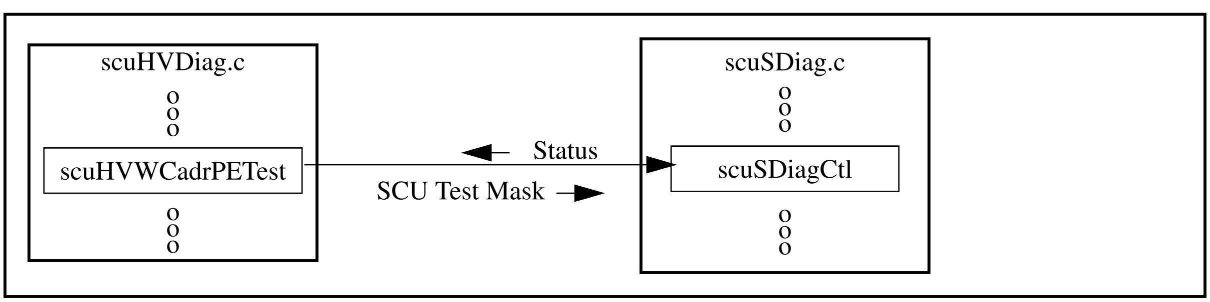

Figure 32 Function ‘scuHVWCadrPETest’ External Interface

17.4.1.4 Internal interfaces

Not applicable.

17.4.2 Function 'scuHVCifPETest'

17.4.2.1 Purpose

The purpose of 'scuHVCifPETest' (SCU HW Verification CIF Parity Error Test) is to verify that the CIF functions correctly when certain parity error signals are present on the backplane. In particular, it verifies that the CIF does not transfer cells to the Core Memory or capture data in the W_IVC FIFOs during cell write cycles with W_IVC_P backplane signal asserted. It also verifies that the CIF does not transfer cells from the Core Memory or capture data in the R_IVC FIFO during cell read cycles with either R_IVC_P or R_CADR_P backplane signals asserted. This test also checks that the corresponding bits in the CIF Status register are set when the above parity errors occur.

17.4.2.2 Function

'scuHVCifPETest' performs the following functions:

1. Initializes result to OK;
2. Resets the SCU;
3. Sets all CRM port thresholds for the SCM to their maximum values;
4. Sets the Core Memory BF Threshold in the CRM to 16K;
5. Initializes two IVCs with the following:
    - IVC one - a R_CSID and W_CSID of zero a DBM of 0x1 and both the high and low priority W_IVC FIFO enable bits set;
    - IVC two - a R_CSID and W_CSID of zero a DBM of 0x1 and the R_IVC FIFO enable bit set;
6. Resets the FLM then initializes the Free List with 4 CADRs;
7. Clears the FLM Halt bit;
8. Prompts the user to jumper the W_IVC_P backplane signal so that it is asserted then toggle DIP Switch 2 on the GPU to continue;
9. Verifies that the W_IVC Parity Error bit in the CIF Status register is set;
10. Initializes cell buffer 0 in the CIF's Shared Buffer to contain all 0xffffs and cell buffer 1 to contain all 0x0s;
11. Performs a valid cell write cycle with IVC one cell buffer 0 then verifies that neither CIF W_IVC FIFO captured data on the write by checking that both FIFOs are empty;
12. Performs a valid cell read cycle with IVC one, cell buffer 1 and R_CADR 0x0 then verifies that the previous write cycle did not transfer the cell into the Core Memory by checking that cell buffer 0 is not equal to cell buffer 1;

13. Prompts the user to unjumper the W_IVC_P backplane signal then toggle DIP Switch 2 on the GPU to continue;
14. Verifies that the W_IVC Parity Error bit in the CIF Status register is clear;
15. Prompts the user to jumper the R_IVC_P backplane signal so that it is asserted then toggle DIP Switch 2 on the GPU to continue;
16. Verifies that the R_IVC Parity Error bit in the CIF Status register is set;
17. Performs a valid cell write cycle with IVC two and cell buffer 0;
18. Performs a valid cell read cycle with IVC two, cell buffer 1 and R_CADR 0x0 then verifies that the CIF R_IVC FIFO did not capture the R_IVC by checking that it is still empty;
19. Verifies that the previous read cycle did not transfer the cell from the Core Memory by checking that cell buffer 0 is not equal to cell buffer 1;
20. Prompts the user to unjumper the R_IVC_P backplane signal then toggle DIP Switch 2 on the GPU to continue;
21. Verifies that the R_IVC Parity Error bit in the CIF Status register is clear;
22. Repeats steps 15. to 21. with the R_CADR_P backplane signal;
23. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to the switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that translates into a string which clearly indicates the cause of the error.

17.4.2.3 External interfaces

Function ‘scuHVCifPETest’ is callable from the VxWorks Shell prompt. For more information see "Function ‘scuHVCifPETest’ External Interface" on page 156.

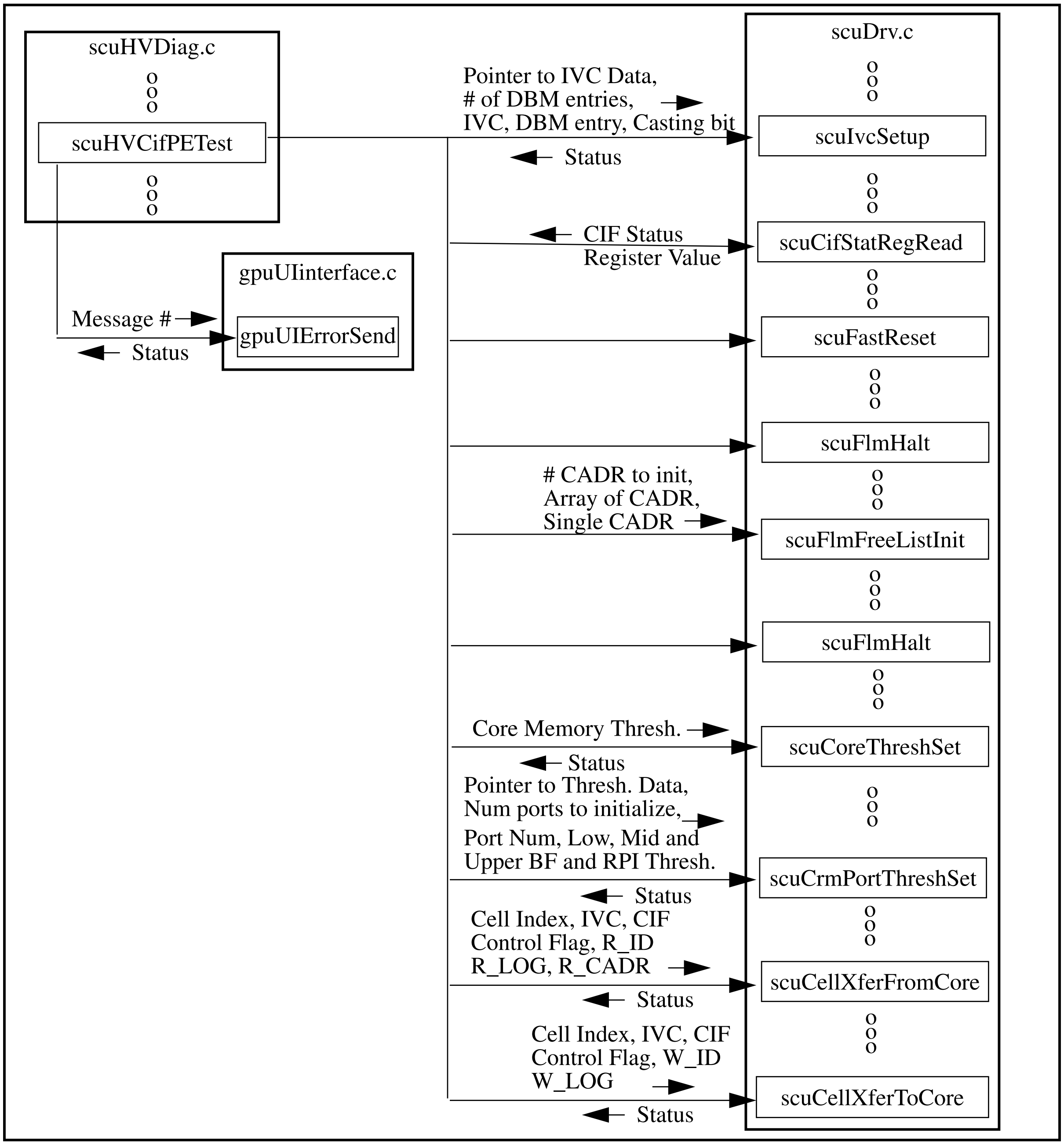

Figure 33 Function ‘scuHVCifPETest’ External Interface

17.4.2.4 Internal interfaces
Not applicable.

17.4.3 Function ‘scuHVFlmCmbfPETest’

17.4.3.1 Purpose

The purpose of ‘scuHVFlmCmbfPETest’ (SCU HW Verification FLM CMBF Count Parity Error Test) is to verify that the FLM’s CMBF counter does not change on cell write cycles with the W_IVC_P backplane signal asserted. It also verifies that the CMBF counter does not change on cell read cycles with either the R_IVC_P or R_CADR_P backplane signals asserted.

17.4.3.2 Function

‘scuHVFlmCmbfPETest’ performs the following functions:

1. Initializes result to OK;
2. Resets the SCU;
3. Sets all CRM port thresholds for the SCM to their maximum values;
4. Sets the Core Memory BF Threshold in the CRM to 16K;
5. Initializes one IVC with the a R_CSID and W_CSID of zero and a DBM of 0x1;
6. Resets the FLM then initializes the Free List with 4 CADRs;
7. Clears the FLM Halt bit;
8. Prompts the user to jumper the W_IVC_P backplane signal so that it is asserted then toggle DIP Switch 2 on the GPU to continue;
9. Performs a valid cell write cycle then verifies that CMBF count is still 0x0;
10. Prompts the user to unjumper the W_IVC_P backplane signal so that it is not asserted then toggle DIP Switch 2 on the GPU to continue;
11. Performs a valid cell write cycle then verifies that CMBF count is 0x1;
12. Prompts the user to jumper the R_IVC_P backplane signal so that it is asserted then toggle DIP Switch 2 on the GPU to continue;
13. Performs a valid cell read cycle with R_CADR 0x0 then verifies that the CMBF count is still 0x1;
14. Prompts the user to unjumper the R_IVC_P backplane signal then toggle DIP Switch 2 on the GPU to continue;
15. Repeats steps 12. to 14 with the R_CADR_P backplane signal;

16. Performs a valid read cycle and verifies that the CMBF count decrements to 0x0;

17. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to the switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that translates into a string which clearly indicates the cause of the error.

17.4.3.3 External interfaces

Function ‘scuHVFlmCmbfPETest’ is callable from the VxWorks Shell prompt. For more information see "Function ‘scuHVFlmCmbfPETest’ External Interface" on page 159.

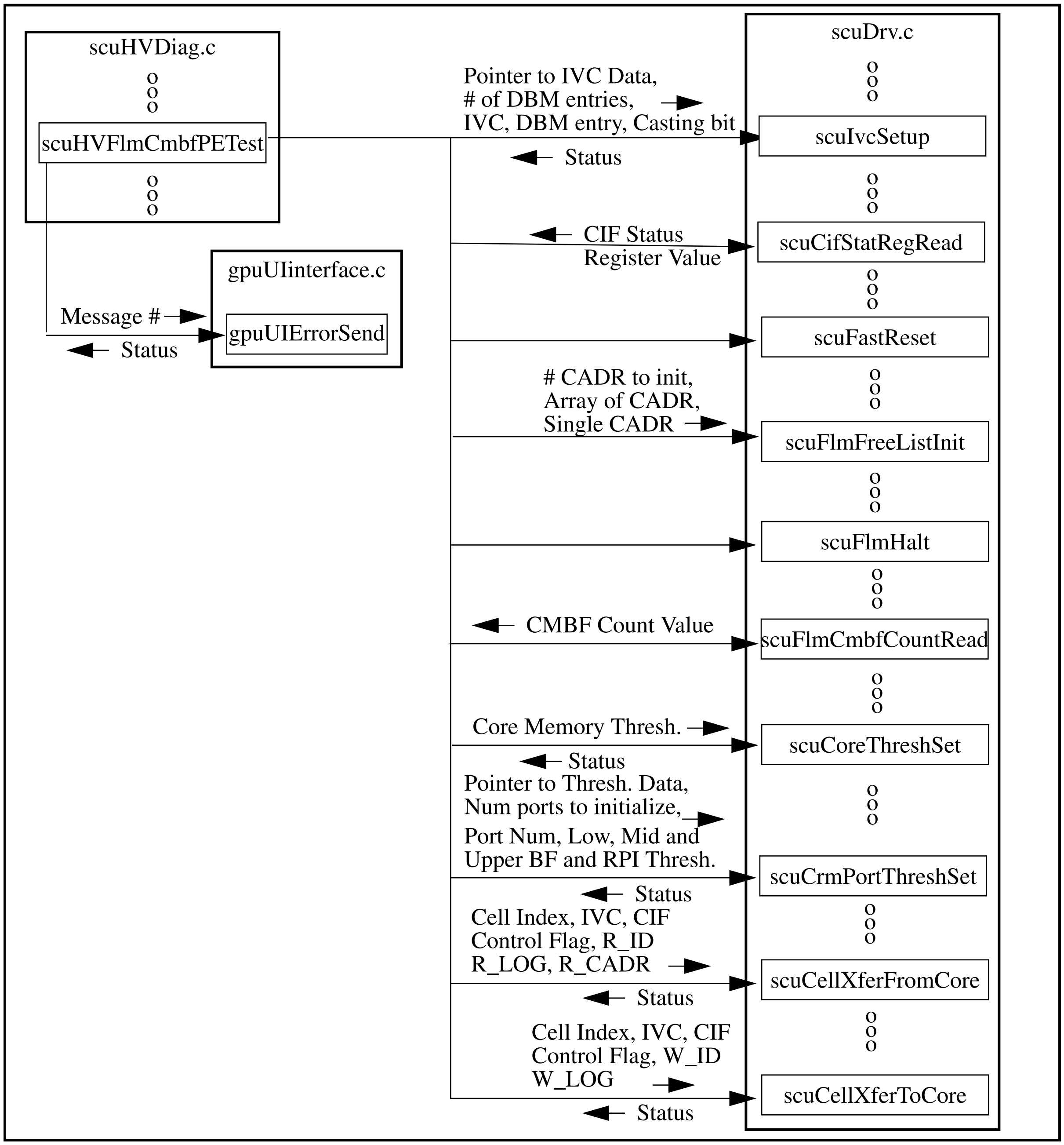

Figure 34 Function ‘scuHVFlmCmbfPETest’ External Interface

17.4.3.4 Internal interfaces

Not applicable.

17.4.4 Function ‘scuHVCdmPETest’

17.4.4.1 Purpose

The purpose of ‘scuHVCdmPETest’ (SCU HW Verification CDM Parity Error Test) is to verify that the CDM does not modify wDBM entries on cell write cycles with the W_IVC_P backplane signal asserted or on cell read cycles with the R_IVC_P or R_CADR_P backplane signals asserted.

17.4.4.2 Function

‘scuHVCdmPETest’ performs the following functions:

1. Initializes result to OK;
2. Resets the SCU;
3. Sets all CRM port thresholds for the SCM to their maximum values;
4. Sets the Core Memory BF Threshold in the CRM to 16K;
5. Initializes two IVCs with the a R_CSID and W_CSID of zero and DBMs of 0x1 and 0x0 respectively;
6. Resets the FLM then initializes the Free List with 4 CADRs;
7. Clears the FLM Halt bit;
8. Sets the current Unicast CM to 0x0 and the Multicast CM to 0x1;
9. Performs 6 valid cell write cycles alternating between the two IVCs;
10. Verifies that the wDBM entry for CADR 0x0 is 0x1;
11. Sets the current Unicast CM to 0x1 and the Multicast CM to 0x0;
12. Alters the DBM for IVC one so that it now has a DBM of 0x0
13. Prompts the user to jumper the W_IVC_P backplane signal so that it is asserted then toggle DIP Switch 2 on the GPU to continue;
14. Performs a valid cell write cycle with IVC one then captures the wDBM entry for CADR 0x0 to verify that it has not changed;
15. Performs a valid cell write cycle with IVC two;
16. Prompts the user to unjumper the W_IVC_P backplane signal so that it is not asserted then toggle DIP Switch 2 on the GPU to continue;
17. Performs a valid cell read cycle with IVC one then captures the wDBM entry for CADR 0x0 to verify that it is 0x1;
18. Performs a valid cell read cycle with IVC one;

19. Prompts the user to jumper the R_IVC_P backplane signal so that it is asserted then toggle DIP Switch 2 on the GPU to continue;
20. Performs a valid cell read cycle with R_CADR 0x0 then captures the wDBM entry for CADR 0x0 to verify that it has not changed from 0x1;
21. Prompts the user to unjumper the R_IVC_P backplane signal then toggle DIP Switch 2 on the GPU to continue;
22. Repeats steps 13. to 15. with the R_CADR_P backplane signal;
23. Resets and enables the chalkmark search;
24. Verifies that CADRs 0x0 through 0x5 are found stale by the search and sends messages numbers indicating the following errors:
    - CADR 0x0 - CM updated with current unicast CM on write cycle with W_IVC_P asserted;
    - CADR 0x1 - CM updated with current multicast CM on write cycle with W_IVC_P asserted;
    - CADR 0x2 - CM updated with current unicast CM on read cycle with R_CADR_P asserted;
    - CADR 0x3 - CM updated with current multicast CM on read cycle with R_CADR_P asserted;
    - CADR 0x4 - CM updated with current unicast CM on read cycle with R_IVC_P asserted;
    - CADR 0x5 - CM updated with current multicast CM on read cycle with R_IVC_P asserted;
    -
25. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to the switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that translates into a string which clearly indicates the cause of the error.

17.4.4.3 External interfaces

Function ‘scuHVCdmPETest’ is callable from the VxWorks Shell prompt. For more information see "Function ‘scuHVCdmfPETest’ External Interface" on page 162.

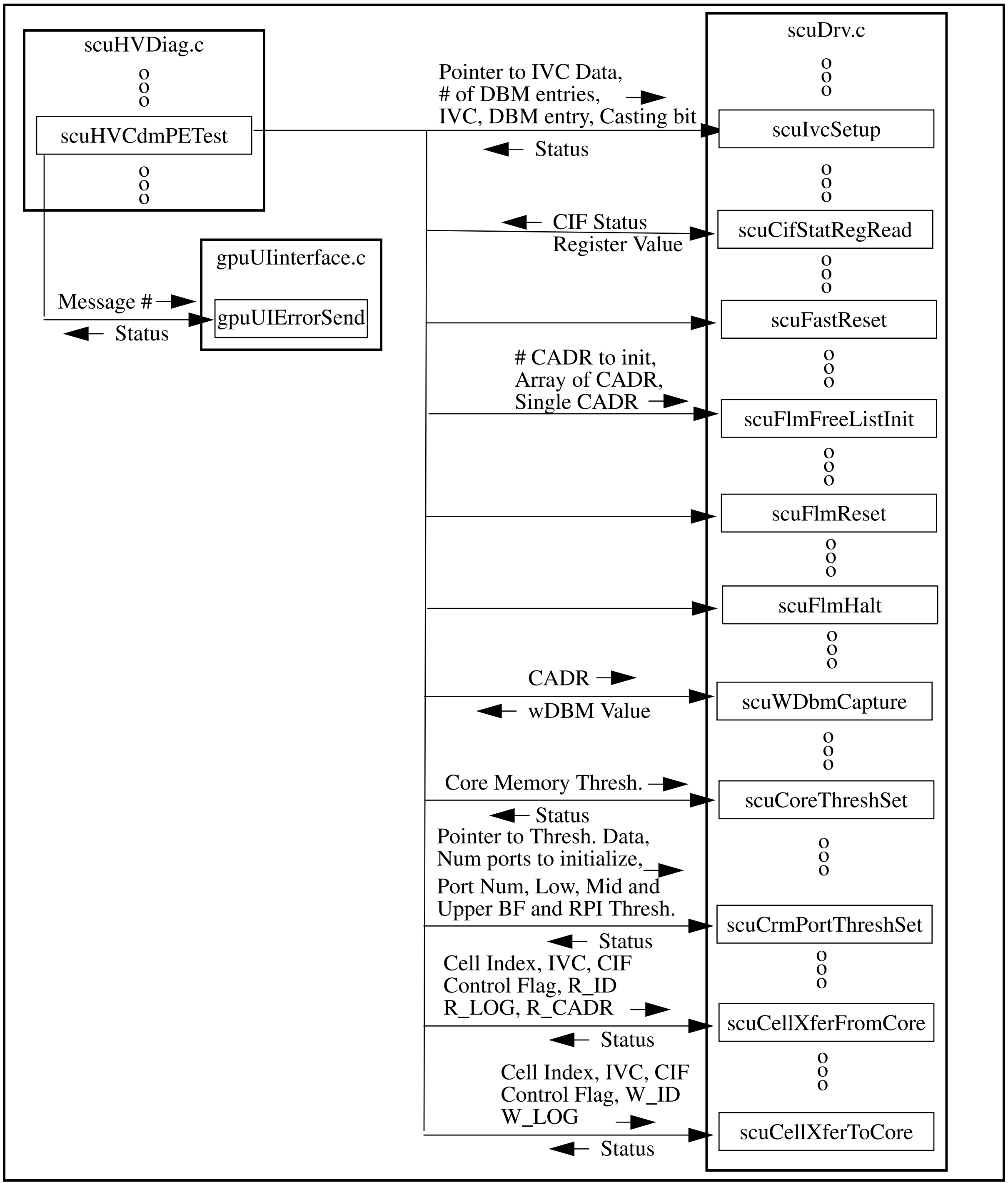

Figure 35 Function 'scuHVCdmfPETest' External Interface

17.4.4.4 Internal interfaces
Not applicable.

17.4.5 Function 'scuHVCrmPETest'

17.4.5.1 Purpose

The purpose of 'scuHVCrmPETest' (SCU HW Verification CRM Parity Error Test) is to verify that the CRM's port usage counters do not change on cell write cycles with the W_IVC_P backplane signal asserted or on cell read cycles with the R_IVC_P or R_CADR_P backplane signals asserted.

17.4.5.2 Function

'scuHVCrmPETest' performs the following functions:

1. Initializes result to OK;
2. Resets the SCU;
3. Sets all CRM port thresholds for the SCM to their maximum values;
4. Sets the Core Memory BF Threshold in the CRM to 16K;
5. Initializes 16 IVCs with the a DBM of 0x1 and R_CSIDs and W_CSIDs so that there is one IVC representing each port;
6. Resets the FLM then initializes the Free List with 16 CADRs;
7. Clears the FLM Halt bit;
8. Prompts the user to jumper the W_IVC_P backplane signal so that it is asserted then toggle DIP Switch 2 on the GPU to continue;
9. Performs valid cell write cycles for each port then reads the port usage counters to verify that they are still 0x0;
10. Prompts the user to unjumper the W_IVC_P backplane signal so that it is not asserted then toggle DIP Switch 2 on the GPU to continue;
11. Performs valid cell write cycles for each port then reads the port usage counters to verify that they are 0x1;
12. Prompts the user to jumper the R_IVC_P backplane signal so that it is asserted then toggle DIP Switch 2 on the GPU to continue;
13. Performs valid cell read cycles for each port form R_CADRs 0x0 through 0xf then reads the port usage counters to verify that they have not changed from 0x1;

14. Prompts the user to unjumper the R_IVC_P backplane signal then toggle DIP Switch 2 on the GPU to continue;
15. Repeats steps 12. to 14 with the R_CADR_P backplane signal;
16. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to the switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that translates into a string which clearly indicates the cause of the error.

17.4.5.3 External interfaces

Function ‘scuHVCrmPETest’ is callable from the VxWorks Shell prompt. For more information see "Function ‘scuHVCrmPETest’ External Interface" on page 165.

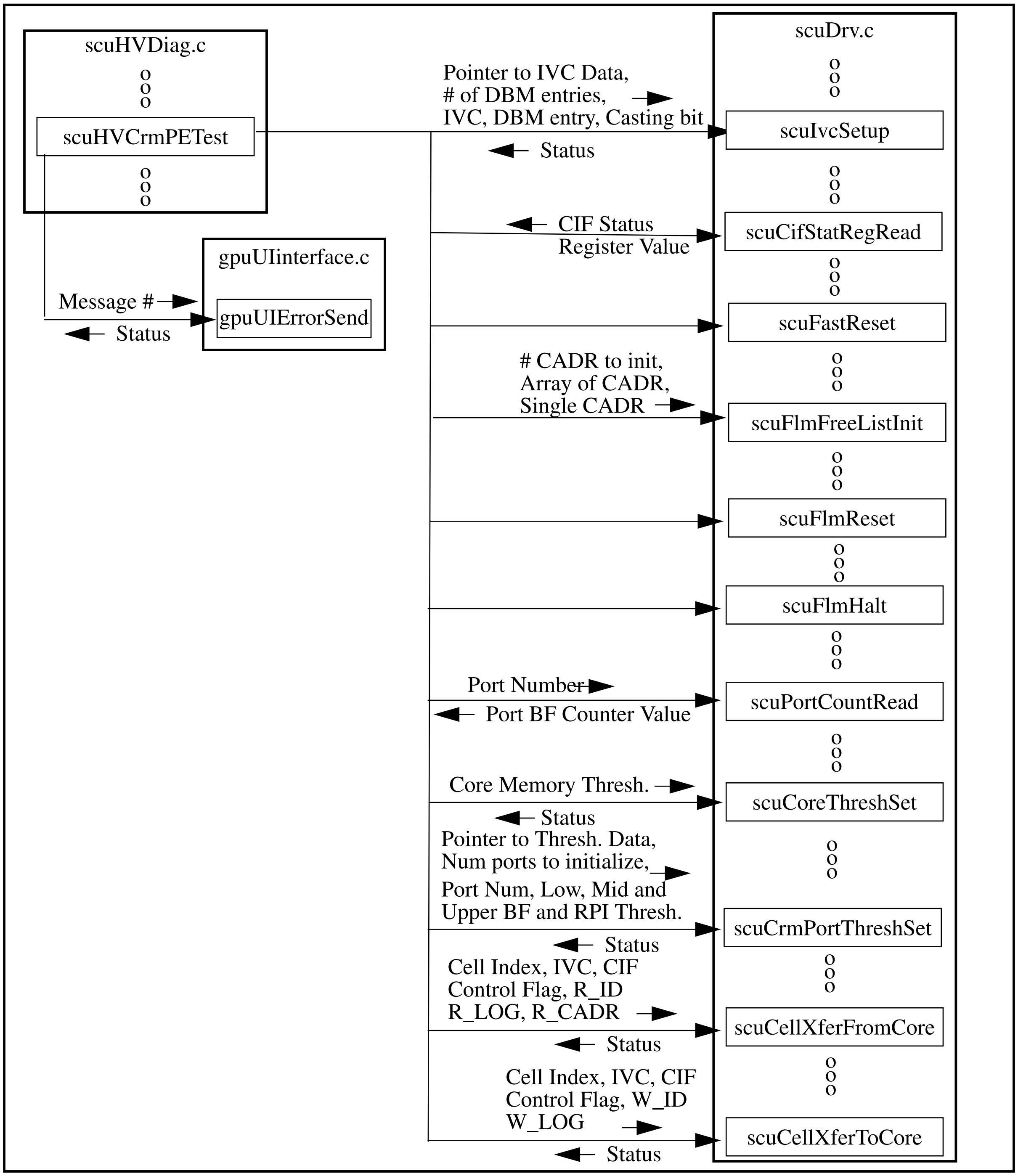

Figure 36 Function 'scuHVCrmPETest' External Interface

17.4.5.4 Internal interfaces

Not applicable.

17.4.6 Function ‘scuHVCrmBFStatTest’

17.4.6.1 Purpose

The purpose of ‘scuHVCrmBFStatTest’ (SCU HW Verification CRM BF Status Test) is to verify that the values on the BF Status and RPI signals presented to each port during its CCB are correct for a range of port counter values and certain threshold values.

17.4.6.2 Function

‘scuHVCrmBFStatTest’ performs the following functions:

1. Resets the SCU;
2. Initializes 16 IVCs with the a DBM of 0x1 and R_CSIDs and W_CSIDs so that there is one IVC representing each port;
3. Resets the FLM then initializes the Free List with 16K CADRs;
4. Clears the FLM Halt bit;
5. Sets all CRM port thresholds for the SCM the following:
    - Lower BF threshold set to 0x10;
    - Middle BF threshold set to 0x20;
    - Upper BF threshold set to 0x30;
    - RPI threshold set to 0x40;
6. Sets the Core Memory BF Threshold in the CRM to 16K;
7. Prompts the user to check that the BF Status values presented to port 0 during its CCB are 0x0 and the RPI is 0x0 during the following cell write cycles;
8. Performs 0xf valid cell write cycles for port 0;
9. Prompts the user to check that the BF Status values presented to port 0 during its CCB are 0x1 and the RPI is 0x0 during the following cell write cycles;
10. Performs 0x10 valid cell write cycles for port 0;
11. Prompts the user to check that the BF Status values presented to port 0 during its CCB are 0x2 and the RPI is 0x0 during the following cell write cycles;
12. Performs 0x10 valid cell write cycles for port 0;
13. Prompts the user to check that the BF Status values presented to port 0 during its CCB are 0x3 and the RPI is 0x0 during the following cell write cycles;
14. Performs 0x10 valid cell write cycles for port 0;

15. Prompts the user to check that the BF Status values presented to port 0 during its CCB are 0x3 and the RPI is 0x1 during the following cell write cycles;
16. Performs 0x10 valid cell write cycles for port 0;
17. Prompts the user to check that the BF Status values presented to port 0 during its CCB are 0x3 and the RPI is 0x0 during the following cell read cycles;
18. Performs 0x10 valid cell read cycles for port 0;
19. Prompts the user to check that the BF Status values presented to port 0 during its CCB are 0x2 and the RPI is 0x0 during the following cell read cycles;
20. Performs 0x10 valid cell read cycles for port 0;
21. Prompts the user to check that the BF Status values presented to port 0 during its CCB are 0x1 and the RPI is 0x0 during the following cell read cycles;
22. Performs 0x10 valid cell read cycles for port 0;
23. Prompts the user to check that the BF Status values presented to port 0 during its CCB are 0x0 and the RPI is 0x0 during the following cell read cycles;
24. Performs 0xf valid cell read cycles for port 0;
25. Repeats steps 8 through 25 for ports 1 through 15;
26. Repeats steps 5 through 25 again this time verifying that the BF, RPI and W_CE_NULL signals are asserted when Artificial Buffer Fill Promotion is occuring. For this, the Core Memory BF Threshold is set to 0 and the per port BF threshold are set as follows:
    - Lower BF threshold set to 0x20;
    - Middle BF threshold set to 0x40;
    - Upper BF threshold set to 0x60;
    - RPI threshold set to 0x80;
27. Returns OK.

17.4.6.3 External interfaces

Function ‘scuHVCrmBFStatTest’ is callable from the VxWorks Shell prompt. For more information see "Function ‘scuHVCrmBFStatTest’ External Interface" on page 168.

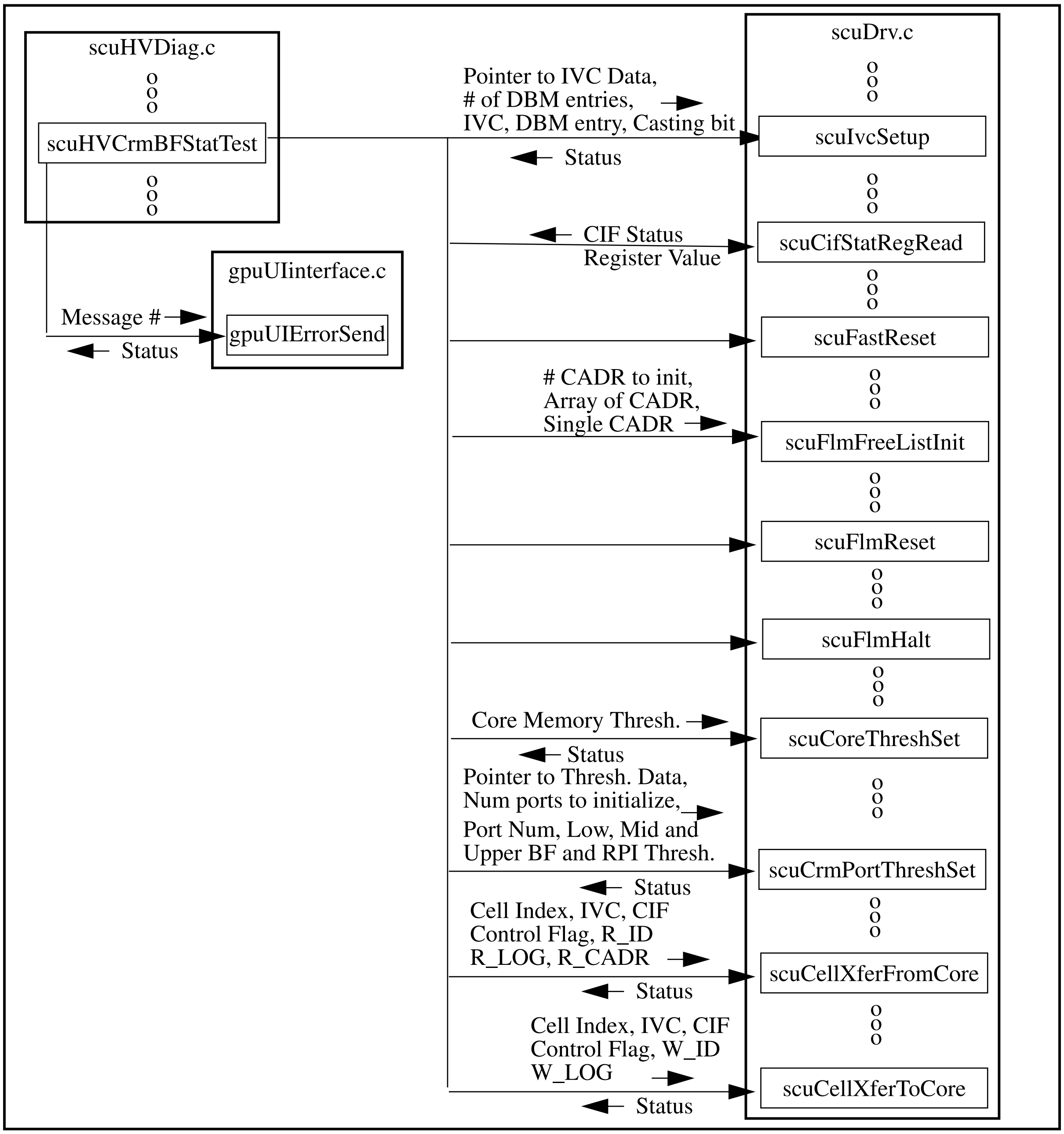

Figure 37 Function ‘scuHVCrmBFStatTest’ External Interface

17.4.6.4 Internal interfaces
Not applicable.

17.4.7 Function 'scuHVCtcItPETest'

17.4.7.1 Purpose

The purpose of 'scuHVCtcItPETest' (SCU HW Verification CTC Parity Error Test) is to verify that the CTC's Interval Timer does not start or stop on cell write cycles with the W_IVC_P signal asserted or on cell read cycle with either the R_IVC_P or R_CADR_P signals asserted.

17.4.7.2 Function

'scuHVCtcItPETest' performs the following functions:

1. Initializes result to OK;
2. Resets the SCU;
3. Sets all CRM port thresholds for the SCM to their maximum values;
4. Sets the Core Memory BF Threshold in the CRM to 16K;
5. Initializes two IVCs, one and two, with a R_CSIDs and W_CSIDs of zero and a DBM of 0x1;
6. Resets the FLM then initializes the Free List with 4 CADRs;
7. Clears the FLM Halt bit;
8. Prompts the user to jumper the W_IVC_P backplane signal so that it is asserted then toggle DIP Switch 2 on the GPU to continue;
9. Programs the IT to start on a W_IVC and IVC one and stop on a W_IVC and IVC two;
10. Performs a valid cell write cycle using IVC one then verifies that the IT has not started timing;
11. Programs the IT to start on a R_IVC and IVC one and stop on a W_IVC and IVC two;
12. Performs a valid cell read cycle using IVC one then verifies that the IT has started timing;
13. Performs a valid cell write cycle using IVC two then verifies that the IT has not stopped counting;
14. Prompts the user to unjumper the W_IVC_P backplane signal so that it is not asserted then toggle DIP Switch 2 on the GPU to continue;
15. Performs a valid cell write cycle using IVC two then verifies that the IT has stopped counting;

16. Prompts the user to jumper the R_IVC_P backplane signal so that it is asserted then toggle DIP Switch 2 on the GPU to continue;
17. Programs the IT to start on a R_IVC and IVC one and stop on a R_IVC and IVC two;
18. Performs a valid cell read cycle using IVC one then verifies that the IT has not started timing;
19. Programs the IT to start on a W_IVC and IVC one and stop on a R_IVC and IVC two;
20. Performs a valid cell write cycle using IVC one then verifies that the IT has started timing;
21. Performs a valid cell read cycle using IVC two then verifies that the IT has not stopped counting;
22. Prompts the user to unjumper the R_IVC_P backplane signal then toggle DIP Switch 2 on the GPU to continue;
23. Repeats steps 16. to 23 with the R_CADR_P backplane signal;
24. Returns result.

Whenever errors are encountered, this function sets result to ERROR and sends a message to the switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that translates into a string which clearly indicates the cause of the error.

17.4.7.3 External interfaces

Function ‘scuHVCtcItPETest’ is callable from the VxWorks Shell prompt. For more information see "Function ‘scuHVCtcItTest’ External Interface" on page 172.

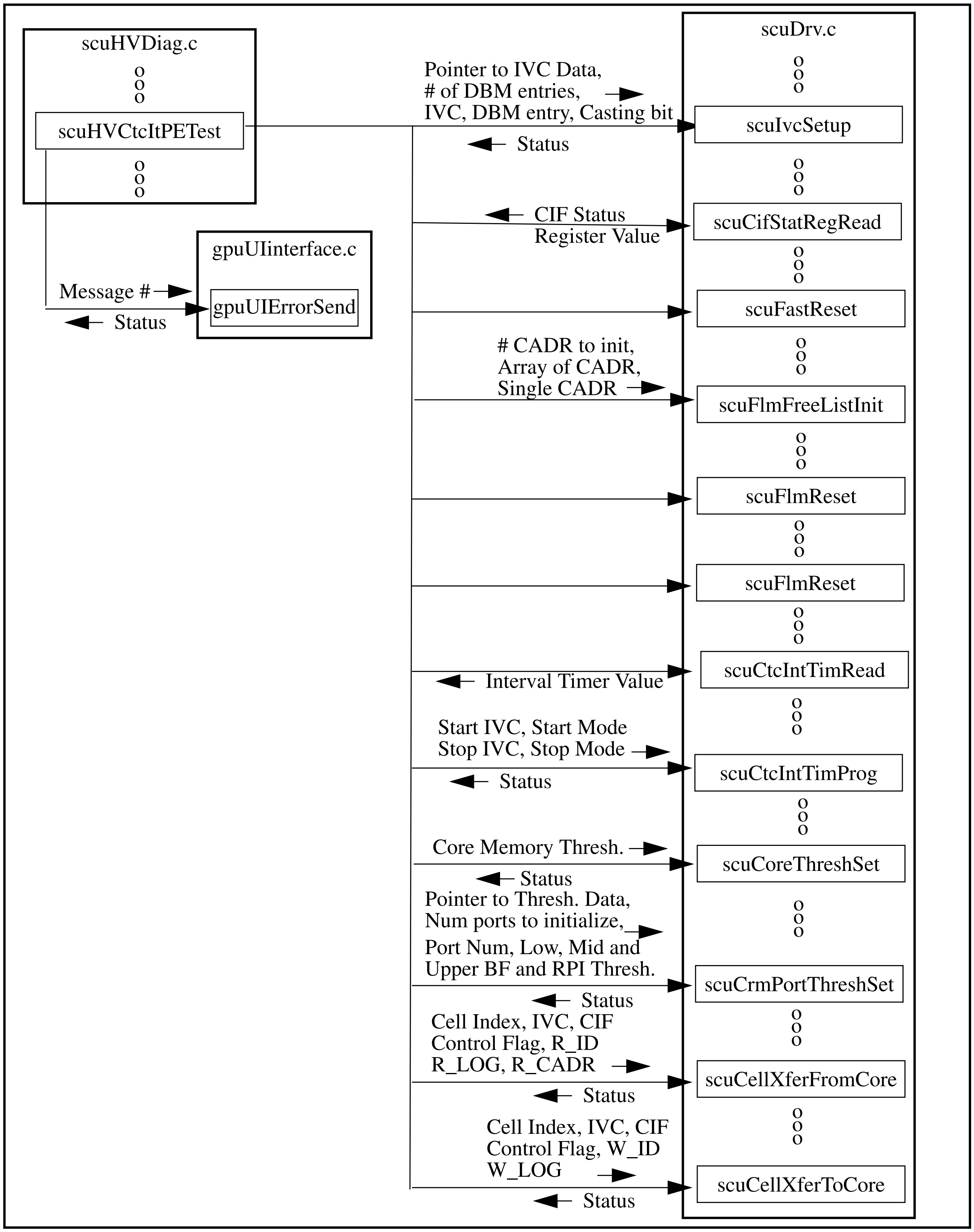
scuHVDiag.c
o
o
o
scuHVCtcItPETest
o
o
o
gpuUIinterface.c
Message #
gpuUIErrorSend
Status
scuDrv.c
Pointer to IVC Data,
# of DBM entries,
IVC, DBM entry, Casting bit
Status
scuIvcSetup
CIF Status
Register Value
scuCifStatRegRead
scuFastReset
CADR to init,
Array of CADR,
Single CADR
scuFlmFreeListInit
scuFlmReset
scuFlmReset
Interval Timer Value
scuCtcIntTimRead
Start IVC, Start Mode
Stop IVC, Stop Mode
Status
scuCtcIntTimProg
Core Memory Thresh.
Status
scuCoreThreshSet
Pointer to Thresh. Data,
Num ports to initialize,
Port Num, Low, Mid and
Upper BF and RPI Thresh.
scuCrmPortThreshSet
Status
Cell Index, IVC, CIF
Control Flag, R_ID
R_LOG, R_CADR
scuCellXferFromCore
Status
Cell Index, IVC, CIF
Control Flag, W_ID
W_LOG
scuCellXferToCore
Status

Figure 38 Function 'scuHVCtclTest' External Interface

17.4.7.4 Internal interfaces
Not applicable.

17.4.8 Function 'scuHVCorePECountTest'

TBD.

17.4.9 Function 'scuHVSwitchAlarmsTest'

TBD.

17.4.10 Function 'scuHVPMResetTest'

17.4.10.1 Purpose

The purpose of 'scuHVPmResetTest' (SCU HW Verification Port Module Reset Signal Test) is to verify that the SCU asserts the TDM PM reset signal correctly on the backplane according to the contents of the PM Reset register.

17.4.10.2 Function

'scuHVPmResetTest' performs the following functions:

1. Resets the SCU;
2. Writes 0x0 to the PM Reset register then prompts the user to verify that each PM is seeing its TDM reset signal asserted in its slot and then toggle DIP Switch 2 on the GPU to continue;
3. Walks a 0 across the PM Reset register prompting the user to check the TDM reset signal for each slot and then toggle DIP Switch 2 on the GPU to continue;
4. Walks a 1 across the PM Reset register prompting the user to check the TDM reset signal for each slot and then toggle DIP Switch 2 on the GPU to continue;
5. Writes 0xFFFF to the PM Reset register then prompts the user to verify that no PM is seeing its TDM reset signal asserted in its slot and then toggle DIP Switch 2 on the GPU to continue;
6. Returns OK.

17.4.10.3 External interfaces

Function ‘scuHVPmResetTest’ is callable from the VxWorks Shell prompt. For more information see "Function ‘scuHVPmResetTest’ External Interface" on page 173.

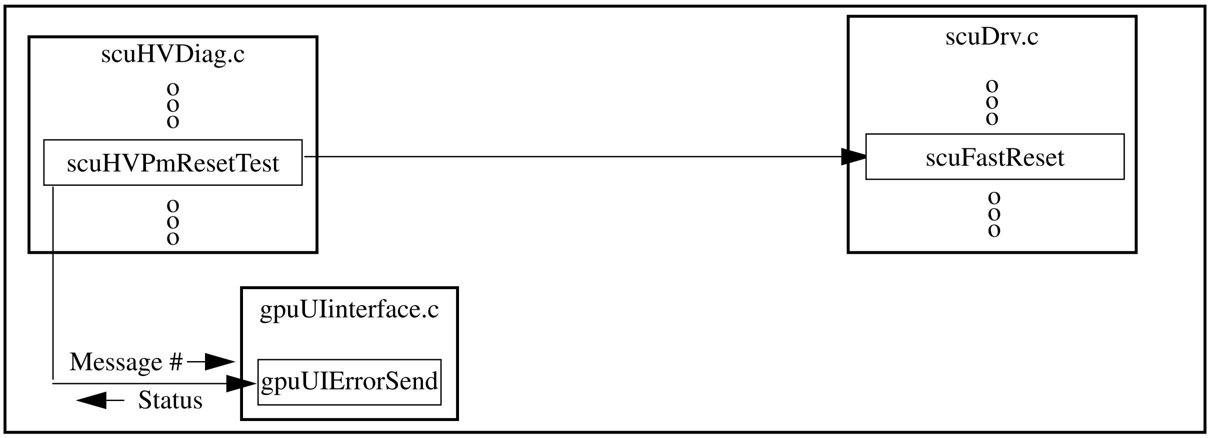

Figure 39 Function ‘scuHVPmResetTest’ External Interface

17.4.10.4 Internal interfaces

Not applicable.

17.4.11 Function ‘scuHVPMIntTest’

17.4.11.1 Purpose

The purpose of ‘scuHVPmIntTest’ (SCU HW Verification Port Module Interrupt Test) is to verify that the bits in the PM Interrupt register are properly set based on which slot has the TDM PM Interrupt signal asserted. It also verifies that the GPU is interrupted with a PM Interrupt when the PM Interrupt register changes from 0xffff to having one of its bits cleared.

17.4.11.2 Function

‘scuHVPmIntTest’ performs the following functions:

1. Initializes result to OK;
2. Installs a test Interrupt Service Routines to service the PM Interrupts from the PBIF;
3. Resets the SCU;
4. Prompts the user to loop the PM Interrupt signal to the PM Reset signal on the backplane then toggle DIP Switch 2 on the GPU to continue;

5. Place the PM Interrupt register in reset and leave it there;
6. Write 0x0 to the PM Reset Register then verify that the PM Interrupt register is 0x0 (indicating no interrupts) and that the PM Interrupt did not interrupt the GPU;
7. Let the PM Interrupt register out of reset then verify that the PM Interrupt register is 0xffff and the PM Interrupt is received;
8. Walks a 0 across the PM Reset Register then verifies that a 1 walks across the PM Interrupt register;
9. Walks a 1 across the PM Reset Register then verifies that a 0 walks across the PM Interrupt register;
10. Writes 0xFFFF to the PM Reset register and verifies that PM Interrupt register contains 0x0 and that the PM Interrupt was not received;
11. Returns OK.

Whenever errors are encountered, this function sets result to ERROR and sends a message to the switch UI by calling function ‘gpuUIErrorSend’ passing it a message number that translates into a string which clearly indicates the cause of the error.

17.4.11.3 External interfaces

Function ‘scuHVPmIntTest’ is callable from the VxWorks Shell prompt. For more information see "Function ‘scuHVPmIntTest’ External Interface" on page 174.

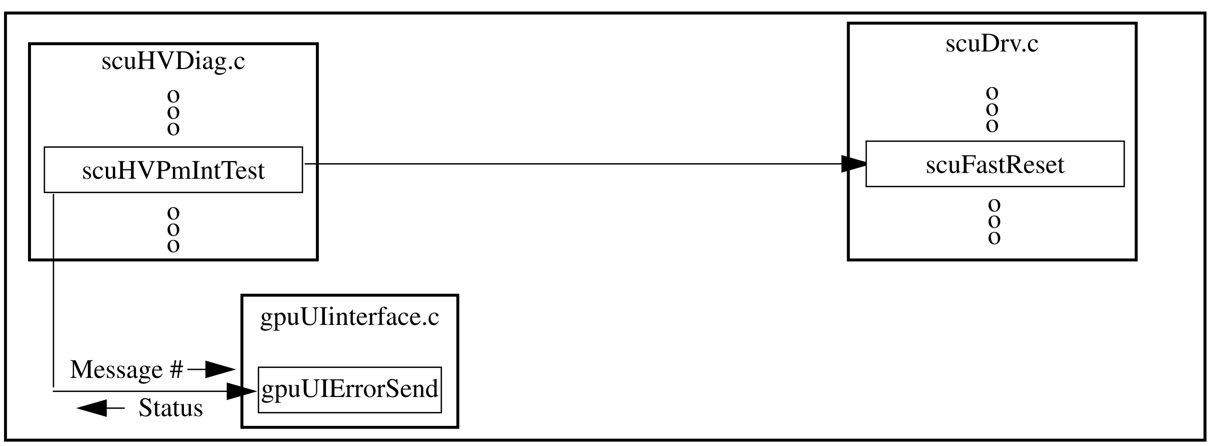

Figure 40 Function ‘scuHVPmIntTest’ External Interface

17.4.11.4 Internal interfaces

Not applicable.

17.4.12 Function 'scuHVIdLogTest'

17.4.12.1 Purpose

The purpose of 'scuHVIdLogTest' (SCU HW Verification CIF ID/LOG Test) is to verify that the SCU correctly drives the contents of the W_ID and W_LOG registers onto the W_ID and W_LOG buses during the SCU CCB of an SCU cell write cycle. It also verifies that the SCU correctly drives the contents of the R_ID and R_LOG registers onto the R_ID and R_LOG buses during the SCU CCB of an SCU cell read cycle.

17.4.12.2 Function

'scuHVIdLogTest' performs the following functions:

1. Resets the SCU;
2. Sets all CRM port thresholds for the SCM to their maximum values;
3. Sets the Core Memory BF Threshold in the CRM to 16K;
4. Initializes one IVCs with a R_CSIDs and W_CSIDs of zero and a DBM of 0x1;
5. Resets the FLM then initializes the Free List with all 16KCADRs;
6. Clears the FLM Halt bit;
7. Prompts the user to set up a logic analyzer to capture data from the W_ID, W_LOG, R_ID and R_LOG buses on the backplane;
8. Performs a cell read and write cycles setting all bits in the W_ID, W_LOG, R_ID and R_LOG registers;
9. Prompts the user to verify that all bits on the W_ID, W_LOG, R_ID and R_LOG buses were set during the above cell read and write cycles;
10. Performs a series of cell read and write cycles walking ones across the W_ID, W_LOG, R_ID and R_LOG registers;
11. Prompts the user to verify that ones correctly walked across the W_ID, W_LOG, R_ID and R_LOG buses during the above cell read and write cycles;
12. Performs a series of cell read and write cycles walking zeroes across the W_ID, W_LOG, R_ID and R_LOG registers;
13. Prompts the user to verify that zeroes correctly walked across the W_ID, W_LOG, R_ID and R_LOG buses during the above cell read and write cycles;
14. Performs a cell read and write cycle clearing all bits in the W_ID, W_LOG, R_ID and R_LOG registers;
15. Prompts the user to verify that all bits on the W_ID, W_LOG, R_ID and R_LOG buses were clear during the above cell read nad write cycles;

16. Returns OK.

17.4.12.3 External interfaces

Function ‘scuHVIdLogTest’ is callable from the VxWorks Shell prompt. For more information see "Function ‘scuHVIdLogTest’ External Interface" on page 177.

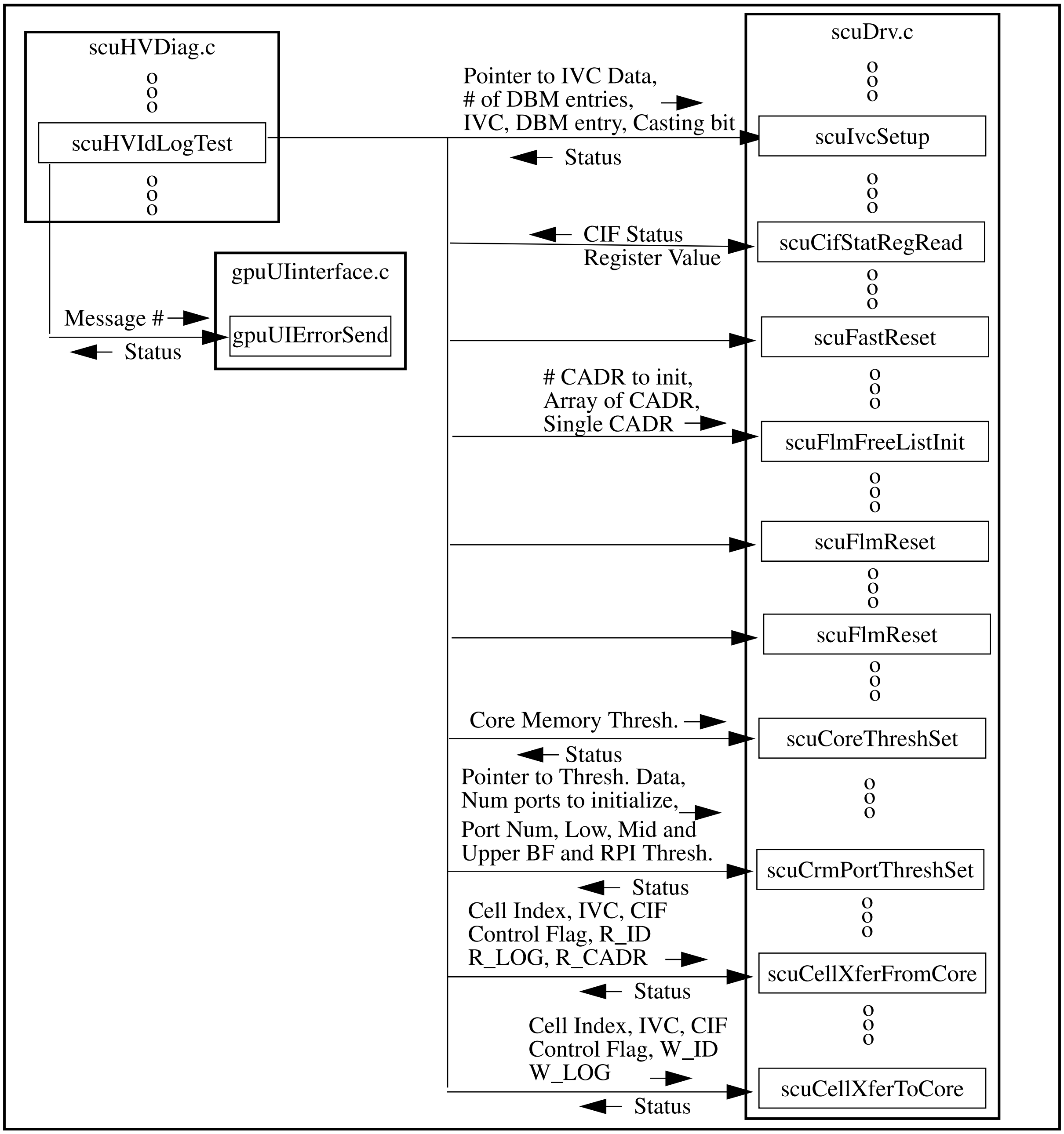

Figure 41 Function 'scuHVIdLogTest' External Interface

17.4.12.4 Internal interfaces

Not applicable.

17.4.13 Function 'scuHVEncDecTest'

17.4.13.1 Purpose

The purpose of 'scuHVEncDecTest' (SCU HW Verification CIF Encoding/Decoding Test) is to verify CIF correctly performs 7 to 8 cell encoding and 8 to 7 cell decoding on cells transferring to and from the Core Memory.

17.4.13.2 Function

'scuHVEncDecTest' performs the following functions:

1. Resets the SCU;
2. Sets all CRM port thresholds for the SCM to their maximum values;
3. Sets the Core Memory BF Threshold in the CRM to 16K;
4. Initializes one IVCs with a R_CSIDs and W_CSIDs of zero and a DBM of 0x1;
5. Resets the FLM then initializes the Free List with only CADR 0x0;
6. Clears the FLM Halt bit;
7. Prompts the user to set up a logic analyzer to capture data on the CIF data path as it enters the encoder, exits the encoder, enters the decoder and exits the decoder;
8. Performs a cell write cycle with cell data whose encoded values are known;
9. Prints out the contents of the cell and its encoded equivalent;
10. Prompts the user to verify that the data entering the encoder matches the cell data and the data exiting the encoder matches the known encoded values for the cell;
11. Performs a cell read cycle from CADR 0x0;
12. Prompts the user to verify that the data entering the decoder matches the known encoded values for the cell and the data exiting the decoder matches the original cell data;
13. Returns OK.

17.4.13.3 External interfaces

Function 'scuHVEncDecTest' is callable from the VxWorks Shell prompt.

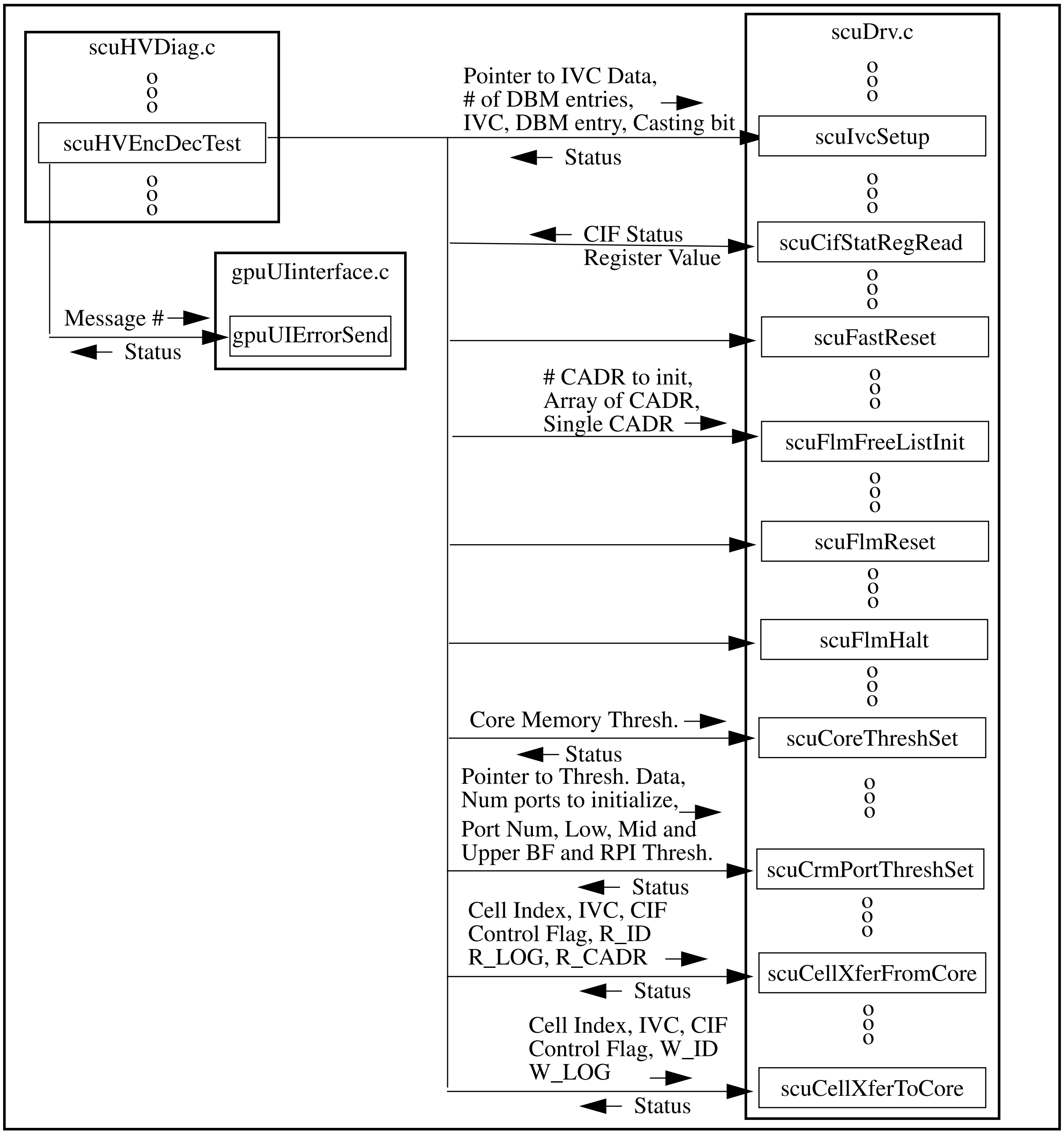

Figure 42 Function ‘scuHVIdLogTest’ External Interface

17.4.13.4 Internal interfaces

Not applicable.

17.4.14 Function ‘scuHVIdleTest’

17.4.14.1 Purpose

The purpose of ‘scuHVIdleTest’ (SCU HW Verification CIF Idle Test) is to verify CIF exchanges idle cells with the Core Memory during SCU CCBs in which the CIF is not being used to transfer real cells.

17.4.14.2 Function

‘scuHVIdleTest’ performs the following functions:

1. Resets the SCU;
2. Sets all CRM port thresholds for the SCM to their maximum values;
3. Sets the Core Memory BF Threshold in the CRM to 16K;
4. Initializes one IVCs with a R_CSIDs and W_CSIDs of zero and a DBM of 0x1;
5. Resets the FLM then initializes the Free List with only CADR 0x0;
6. Clears the FLM Halt bit;
7. Initializes a cell to be used for the idle cell exchange and places it in the Shared Buffer;
8. Performs a valid cell write cycle to place the idle cell in the Core Memory at CADR 0x0;
9. Prompts the user to set up a logic analyzer to capture data on the CIF data path as it enters the encoder and exits the decoder;
10. Prompts the user to verify that the data passing through the CIF data path (in both directions) is the idle cell data found in cell buffer 0x0 in the CIF Shared Buffer;
11. Returns OK.

17.4.14.3 External interfaces

Function ‘scuHVIdleTest’ is callable from the VxWorks Shell prompt.

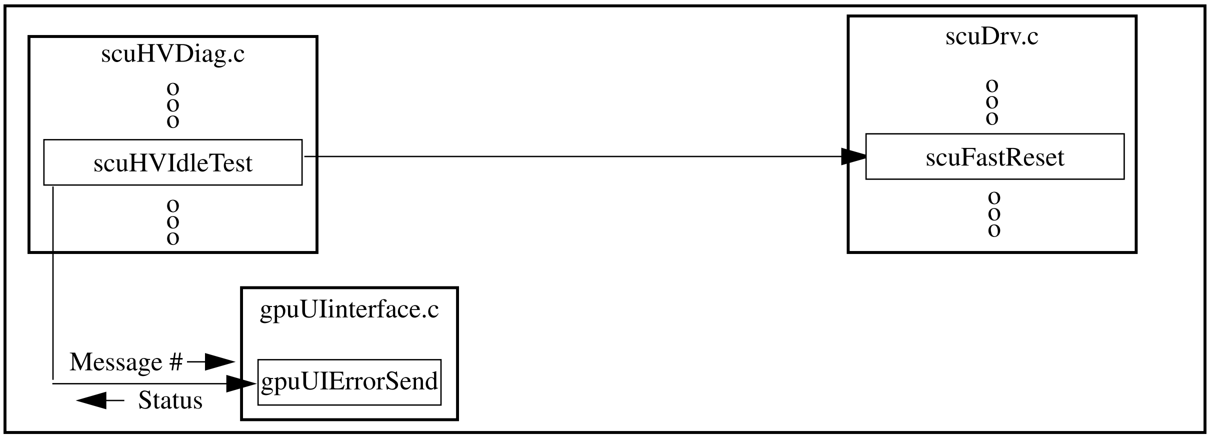

Figure 43 Function ‘scuHVIdLogTest’ External Interface

17.4.14.4 Internal interfaces

Not applicable.

17.5 Module ‘scuPDiag.c’

Trevor: This is where you can place your design for the performance tests. Please use section 3.4.1 as a template for the design if you can.

The functions in this module...

17.5.1 Function ‘scuSCdmCMSrchTest’

17.5.1.1 Purpose

The purpose of function ‘scuSCdmCMSrchTest’ (SCU CDM Chalkmark Search Test) is to verify that the CDM correctly finds all CADRs with stale chalkmarks which fit the stale chalkmark criteria. It also verifies that the correct current chalkmark is used for comparison (multicast or unicast) and that CADRs are not reported stale when a cell is not being stored at the CADR. It also tests the mechanism by which the CDM reports the stale CADRs to the GPU.

17.5.1.2 Function

Function ‘scuSCdmCMSrchTest’ performs the following:

1. Initializes result to OK;
2. Resets the SCU;
3. Clears the FLM Halt bit;
4. Sets all CRM port thresholds for the SCM to their maximum values;
5. Initializes two IVCs with the following:
    - A DBM of 0x1 and a casting bit of 0x0;
    - A DBM of 0x1 and a casting bit of 0x1;
6. Initializes the FLM Free List with all 16K CADRs;
7. Set the current Multicast Chalkmark to 0x1 and the current Unicast Chalkmark to 0x0;
8. Performs 16K valid cell write cycles alternating between the two IVCs;
9. Perform valid cell read cycles for all CADRs that have their lower three bits equal to 010 and 011;
10. Set the current Multicast Chalkmark to 0x0 and the current Unicast Chalkmark to 0x1;
11. Reset only the FLM;
12. Initialize the FLM Free List with the 4K CADRs whose lower three bits are equal to 100 and 101;
13. Perform 4K valid cell write cycles using the 4K CADRs just initialized and alternating between the two IVCs;
14. Perform 4K valid cell read cycles from the CADRs whose lower three bits are 110 and 111;

15. Enable the GPU to receive the CRM's stale CADR interrupt;
16. Initiate the chalkmark search;
17. Whenever the stale CADR interrupt occurs perform the following:
    - Verify that the stale chalkmark bit in the CDM Status register is set and then clear it. Verify that the bit clears;
    - Read the stale CADR from the CDM's Stale CADR register and save it in an array;
    - Increment a count tracking the number of stale CADRs found;
    - Re-enable the test;
18. Verify that 4K stale CADRs were found an that they were only the ones whose lower three bits ended in 000 or 001;
19. Verify that the chalkmark search complete bit is set at the end of the search;
20. Verify that the chalkmark search complete bit can be cleared;
21. Repeat the search (steps 16 - 20) with the CIF transferring cells to and from the Core Memory at its full speed. Stop the search before it finds all 4K stale CADRs and reset it to verify that the search starts again in the wDBM at location after it is reset;
22. Returns result.

Whenever errors are encountered, this functions sets result to ERROR and sends a message to switch UI by calling function 'gpuUIErrorSend' passing it a message number that translates into a string which clearly indicates the cause of the error.

17.5.1.3 External interfaces

Function 'scuSCdmCMSrchTest' is invoked by 'scuSDiagCtl' in module 'scuSDiag.c' during System Initialization but is also callable from the VxWorks Shell prompt. For more information see "Function 'scuSMCifRegTest' External Interface" on page 73. This function also invokes functions 'intVecGet' and 'intConnect' in the VxWorks module 'intLib.c' [2].

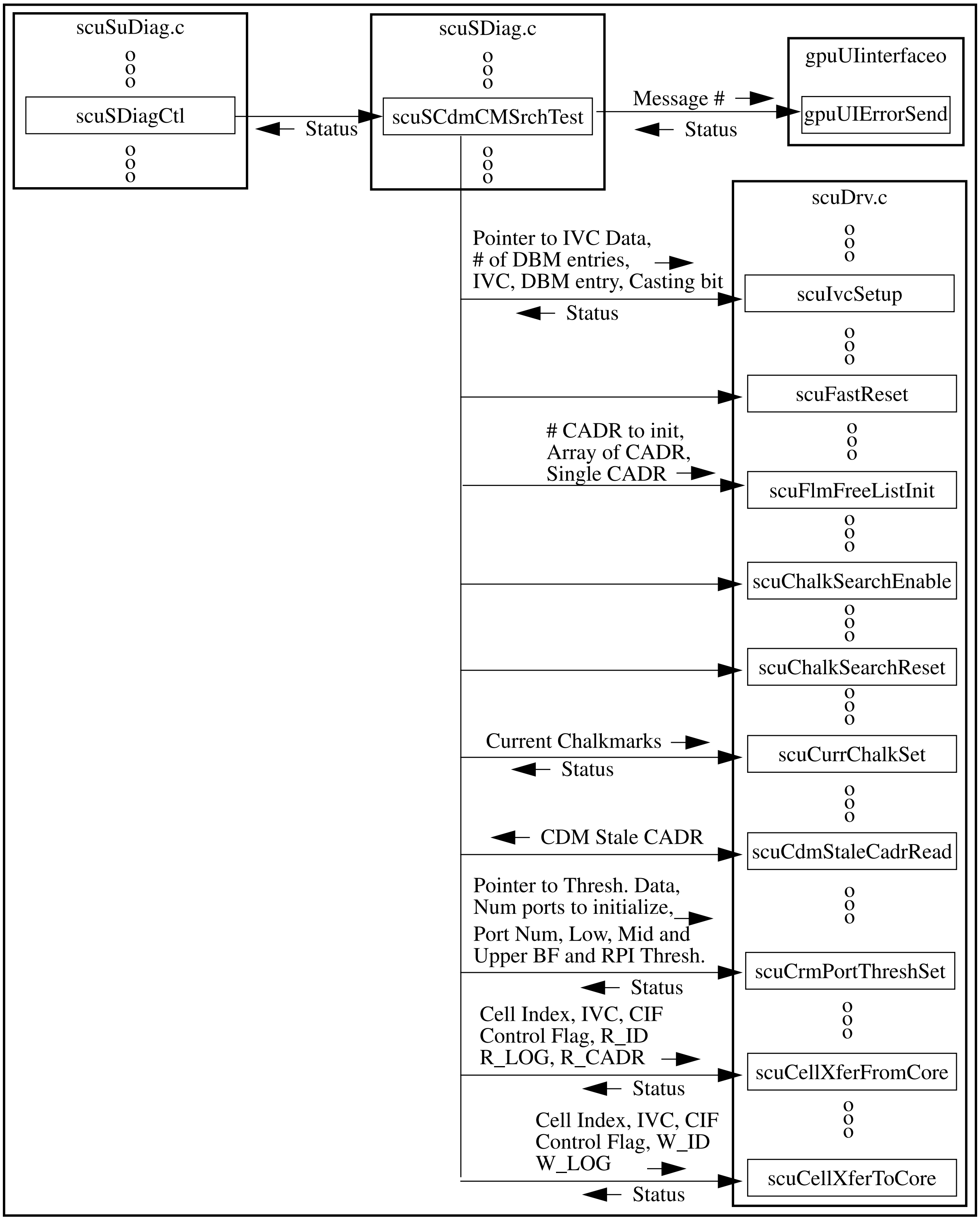

Figure 44 Function 'scuSCdmCMSrchTest' External Interface

17.5.1.4 Internal interfaces

Function ‘scuSCdmCMSrchTest’ is invoked internally from function ‘scuSDiagCtl’ during SCU Start-up Diagnostics.

17.6 External Interface Via VxWorks Shell

All functions in module ‘scmScuuDiag.c’ can be invoked by a user from the command line interface of the VxWorks Shell on the console of the switch workstation. When invoked in this manner, functions execute at the same priority level as the shell. These tests may also be spawned as separate tasks from the shell, using VxWorks function ‘taskSpawn’, and can be given a higher or lower priority than the shell. For more information on invoking functions from the shell, please see the VxWorks Programmer’s Guide.[2]

The ability to invoke functions from the shell will give the user the ability to perform any test in module ‘scmScuDiag.c’ individually by calling the corresponding function or to execute any set of tests by calling function ‘scuSUDiagCtl’ and passing it an appropriate test mask.

CAUTION: performing any hardware diagnostics test has adverse affects on the SCU/Core and will require re-initialization afterwards. Any user IVC data will be lost when tests are invoked. SCU/Core Diagnostics should only be invoked when the switch is not supporting any IVCs and/or it is impossible to gracefully shutdown existing IVCs and data.

18.0 Real time estimates

19.0 Glossary of definitions & abbreviations

ATM :Asynchronous Transfer Mode

BF :Buffer Fill

BFC :Buffer Fill Count

BFS :Buffer Fill Status

BNR :Bell Northern Research

[2]When switch development is complete, the VxWorks Shell should be removed and this functionality should be supplied by a (Graphical) User Interface.

BTC	:Bus-tap Controller
CADR	:Cell Address (in the core)
CBFC	:Core Buffer Fill Count
CBT	:Cell Bus Tap
CCB	:Cell Control Bus
CDM	:Cell Departure Monitor
CIF	:Cell Interface
CMBF	:Core Memory Buffer Fill
CRM	:Core Resource Manager
CTC	:Cell Time Counter
FIFO	:First-In First-Out
FLM	:Free List Manager
GPU	:General Processor Unit
IVC	:Internal Virtual Connection
NIS_INIT	:Not In Service - Initialization State
NIS_NETWK	:Not In Service - Network State
P-BUS	:Processor Bus
PM	:Port Module
PCU	:Port Control Unit
R_CADR	:Read CADR
R_IVC	:Read IVC
RTT	:Round Trip Time
SCU	:Switch Control Unit
SCM	:Switch Control Module (consists of a GPU and an SCU)

TBD	:To Be Determined
TC	:Terminator Card
TDM	:Time Division Multiplexed
W_CADR	:Write CADR
W_IVC	:Write IVC
W_CE_NULL	:Write Common Equipment Null
W_NULL	:Write Null

20.0 References

[1] Wind River Systems, “VxWorks Programmers Guide.” Release 5.1. February, 1993.

[2] Wind River Systems, “VxWorks Reference Manual.” Release 5.1. March, 1993.

Internal Documents:

[3] Alan Chapman, Osama Bahgat, “Feature List for the CreditNetwork Switch”, Oct. 5, 1993.

[4] Melhem Chaar, BNR, “Design Specification - The Credit Network - MasterCard”, Feb. 14, 1994.

[5] Gary Depelteau, BNR, “SCU, Core Memory and Terminator Card Diagnostics FN”, June 8, 1994.

[6] Gary Depelteau, BNR, “System Initialization FN”, Sept. 1, 1993.

[7] Gary Depelteau, BNR, “System Initialization DD”, Sept. 16, 1993.

[8] Gary Depelteau, BNR, “GPU Diagnostics FN”, Sept. 16, 1993.

[9] Dave Milton, BNR. “CreditNet GPU Specification.” Version 1.0