W_IVC :Write IVC (part of CCB)

22.0 References

[1] The ATM Forum. “ATM User-Network Interface Specification.” Version 2.2. June 21, 1993.

[2] H.T. Kung, Harvard University; Alan Chapman, BNR. “The FCVC (Flow Controlled Virtual Channels) Proposal for ATM Networks.”Version 1.0. June 22, 1993.

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

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

Internal Documents:

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

[6] Gary Depelteau, BNR, “GPU Diagnostics”, ???Sept. 1, 1993.

[7] Ken Grimble, BNR. “MasterCard Specification.” July 6, 1993.

[8] Gajendra Gupta, BNR, “OMEGA Generic Digital Diagnostics_A34”, May 17, 1991.

[9] Shuo Huang, BNR. “Data Manager Specification.” Version 1-0.

[10] Dan Leonida, BNR. “Credit Manager.” Version 0-1.

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

[12] Robert Morris, Harvard. “Scheduler Specification.” Version 1-1. February 26, 1993.

[13] Margo Seltzer and Mike Smith, Harvard University.”Software Specification.” Version 0-2. April 13, 1993.

[14] John Vincent, BNR. “EPATH Specification.” Version.1-1.

[15] John Vincent, BNR. “IPATH Specification.” Version 1.4.

[16] Cliff Young, Harvard University. “Port Controller Specification.” Version 1-1. June 16,1993.

GPU	:General Processor Unit
IVC	:Internal Virtual Connection
LUT	:Look-Up Table
NIS_BOOT	:Not In Service - Booting
NIS_INIT_DIAG	:Not In Service - Initialization and Diagnostics
NIS_READY	:Not In Service - Ready
PIF	:Processor Control Bus Interface
PC	:Port Card
PCC	:Port Card Controller
PVC	:Permanent Virtual Channel
R_CADR	: Read Cell Address (part of CCB)
R_IVC	:Read IVC (part of CCB)
RAM	:Random Access Memory
RCC	:Round trip Cell Count
ROM	:Read Only Memory
RTT	:Round Trip Time
SCM	:Switch Control Module (GPU + SCU)
SCU	:Switch Control Unit
SP	:Single Port
TBD	:To Be Determined
VAR	:Virtual Connection Activation Requestor
VC	:Virtual Connection
W_CADR	:Write Cell Address (part of CCB)

19.10 System Integration Diagnostics

The requirements and design details pertaining to System Integration Diagnostics can be found in the FN and DD for System Integration Diagnostics. This module was included in “System Initialization Software Topology” on page 47 to illustrate how it fits into the overall structure of System Initialization SW.

20.0 Real time estimates

Not applicable.

21.0 Glossary of definitions and abbreviations

ALID	:Active List Identifier
ALM	:Active List Manipulator
ATM	:Asynchronous Transfer Mode
BNR	:Bell Northern Research
CALM	:Credit Active List Manipulator
CCB	:Cell Control Bus
CDM	:Cell Departure Monitor
CIF	:Cell Interface
CLM	:Cell List Manipulator
CMM	:Core Resource Manager
DALM	:Data Active List Manipulator
DP	:Dual Port
EEPROM	:Electrically Erasable-Programmable Read Only Memory
FIFO	:First-In First-Out
FLM	:Free List Manager

19.9.68.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuIvcInit’ External Interface" on page 155.

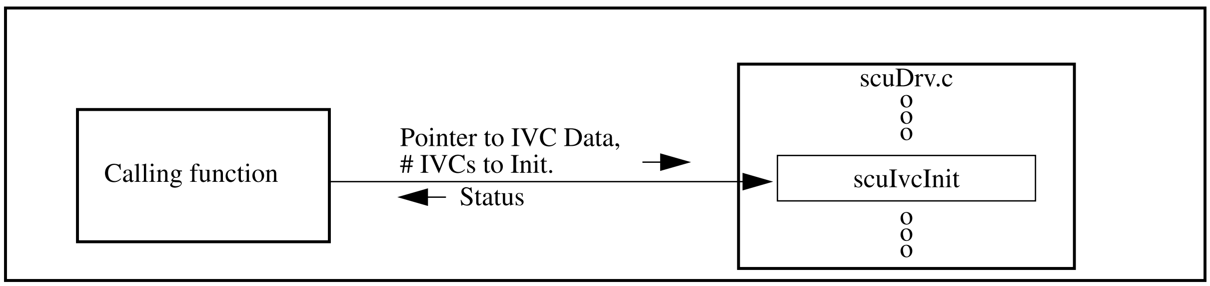


Figure 111 Function ‘scuIvcInit’ External Interface

19.9.68.4 Internal interfaces

For more information see "Function ‘scuIvcInit’ Internal Interface" on page 155.

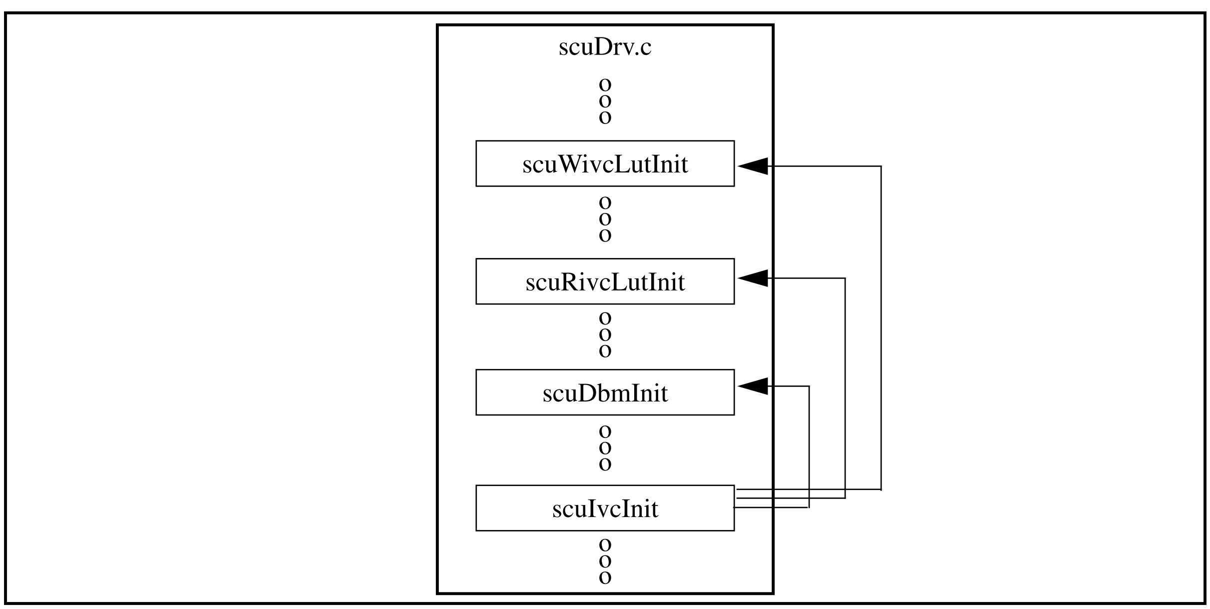


Figure 112 Function ‘scuIvcInit’ Internal Interface

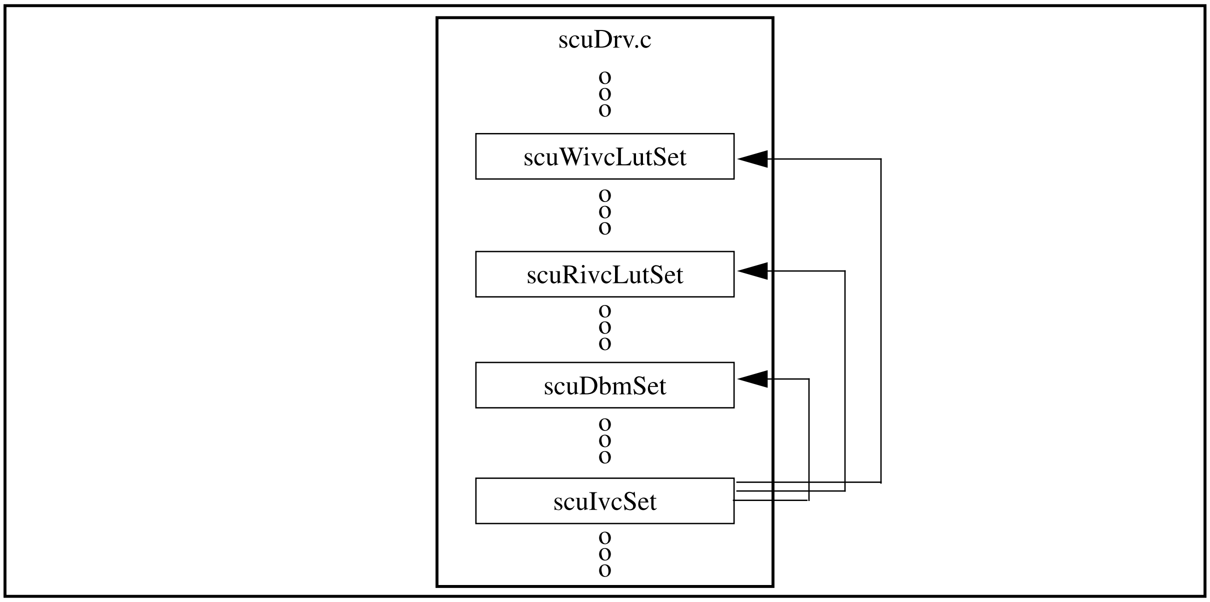


Figure 110 Function ‘scuIvcSet’ Internal Interface

19.9.68 Function ‘scuIvcInit’

19.9.68.1 Purpose

The purpose of function ‘scuIvcInit’ (SCU IVC Initialization) is to setup the R_IVC lookup table, W_IVC lookup table, DBM and casting bit for a set of IVCs. Calling functions pass it a pointer to an array of IVC data structures and the number of IVCs to initialize.

19.9.68.2 Function

Function ‘scuIvcInit’ performs the following:

1. Calls function ‘scuWivcLutInit’ passing it the pointer to IVC data structures and the number of W_IVC lookup table entries to initialize;
2. Calls function ‘scuRivcLutInit’ passing it the pointer to IVC data structures and the number of R_IVC lookup table entries to initialize;
3. Calls function ‘scuDbmInit’ passing it the pointer to IVC data structures and the number of DBM entries to initialize;
4. Returns OK.

19.9.67 Function ‘scuIvcSet’

19.9.67.1 Purpose

The purpose of function ‘scuIvcSet’ (SCU IVC Initialize) is to setup the R_IVC lookup table, W_IVC lookup table, DBM and casting bit for a single IVC. Calling functions pass it an IVC, a R_ICSID, a R_IVC FIFO enable bit, a W_CSID, a high priority W_IVC FIFO enable bit, a low priority W_IVC FIFO enable bit, a DBM and a casting bit.

19.9.67.2 Function

Function ‘scuIvcSet’ performs the following:

1. Calls function ‘scuWivcLutSet’ passing it the IVC, W_CSID, low priority W_IVC FIFO enable bit and high priority W_IVC FIFO enable bit as that passed to this function;
2. Calls function ‘scuRivcLutSet’ passing it the same IVC, R_CSID and R_IVC FIFO enable bit as that passed to this function;
3. Calls function ‘scuDbmSet’ passing it the same DBM, casting bit and IVC as that passed to this function;
4. Returns OK.

19.9.67.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuIvcSet’ External Interface" on page 153.

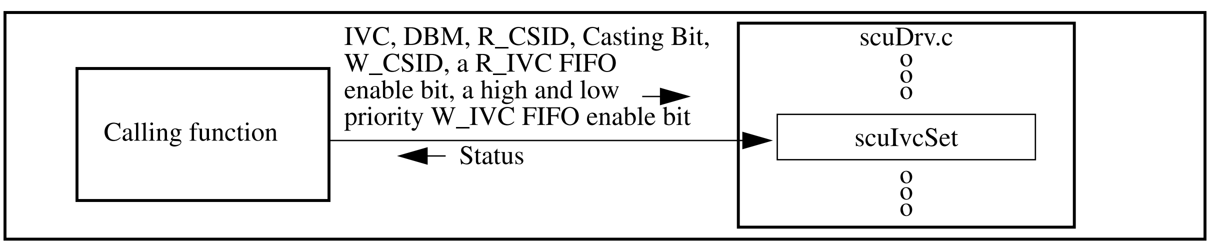


Figure 109 Function ‘scuIvcSet’ External Interface

19.9.67.4 Internal interfaces

For more information see "Function ‘scuIvcSet’ Internal Interface" on page 154.

5. Returns OK.

19.9.66.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuRivcLutSetup’ External Interface" on page 95.

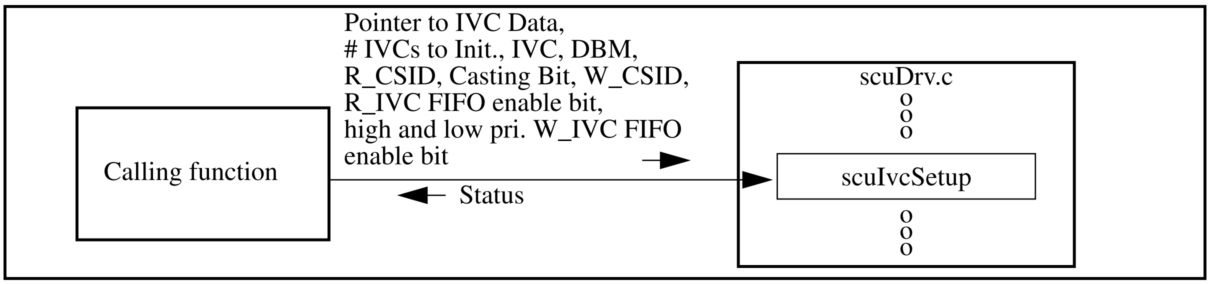


Figure 107 Function ‘scuIvcSetup’ External Interface

19.9.66.4 Internal interfaces

For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

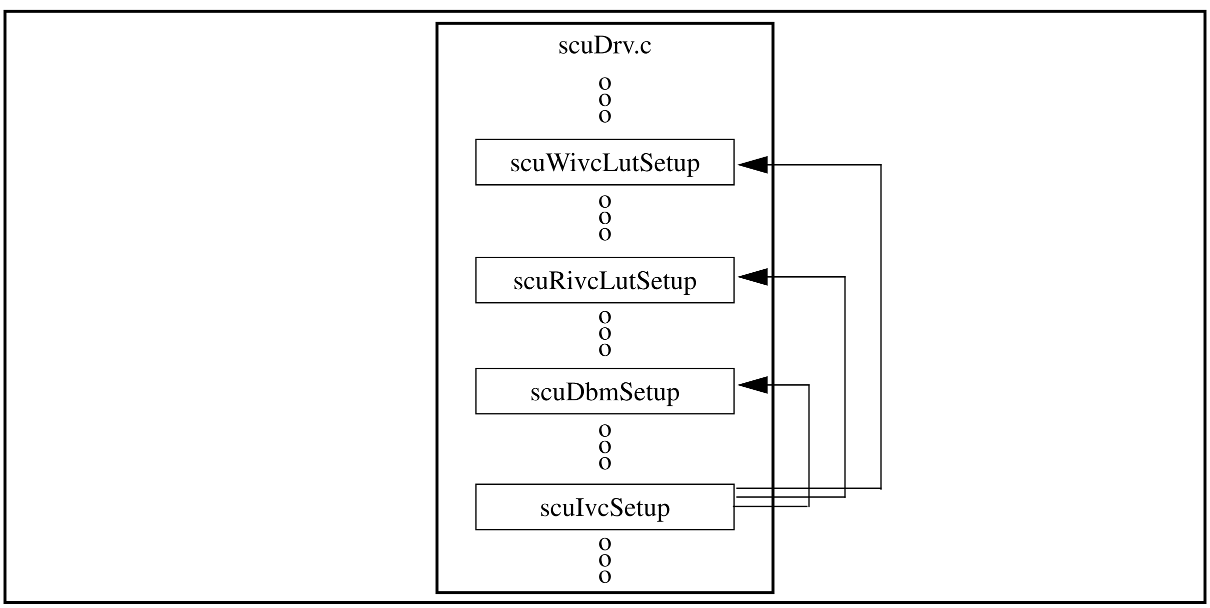


Figure 108 Function ‘scuIvcSetup’ Internal Interface

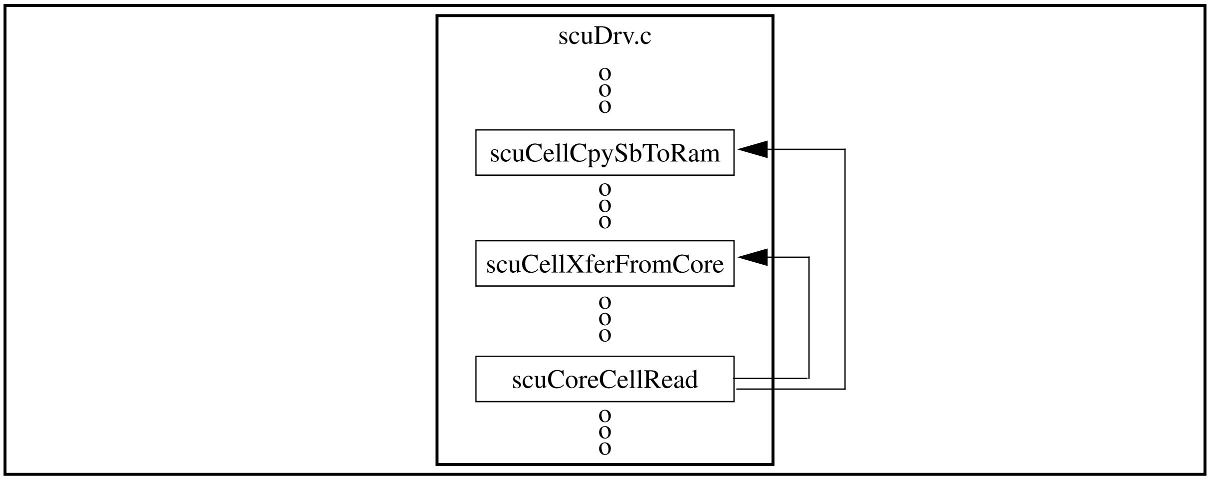


Figure 106 Function ‘scuCoreCellRead’ Internal Interface

19.9.66 Function ‘scuIvcSetup’

19.9.66.1 Purpose

The purpose of function ‘scuIvcSetup’ (SCU IVC Setup) is to setup the R_IVC lookup table, W_IVC lookup table, DBM and casting bit for one IVC or a list of IVCs. Calling functions pass it a pointer to an array of IVC data structures, the number of IVCs to initialize, an IVC, a R_CSID, a R_IVC FIFO enable bit, a W_CSID, a high priority W_IVC FIFO enable bit, a low priority W_IVC FIFO enable bit, a DBM and a casting bit.

19.9.66.2 Function

Function ‘scuIvcSetup’ performs the following:

1. If the number of IVCs to initialize is zero or greater than 16K then it returns ERROR;
2. Calls function ‘scuWivcLutSetup’ passing it the pointer to IVC data structures, the number of W_IVC lookup table entries to initialize, and the same IVC and W_IVC lookup table entry as that passed to this function;
3. Calls function ‘scuRivcLutSetup’ passing it the pointer to IVC data structures, the number of R_IVC lookup table entries to initialize, and the same IVC and R_IVC lookup table entry as that passed to this function;
4. Calls function ‘scuDbmSetup’ passing it the pointer to IVC data structures, the number of DBM entries to initialize, and the same DBM, casting bit and IVC as that passed to this function;

3. Compares the value of the IVC residing in the cell in the Shared Buffer to the IVC passed from the calling function and returns ERROR if there is a mismatch. This situation also results in an appropriate message being sent to the switch User Interface;
4. Calls function ‘scuCellCpySbToRam’ passing it the pointer to the cell data structure at which to copy the cell and the cell index into the Shared Buffer, both of which were passed to this function;
5. Returns OK.

19.9.65.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

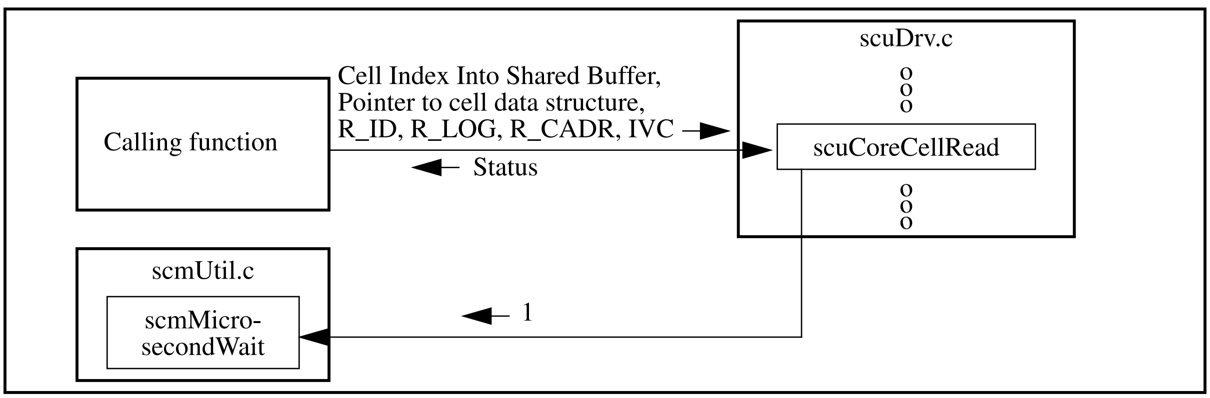


Figure 105 Function ‘scuCoreCellRead’ External Interface

19.9.65.4 Internal interfaces

For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

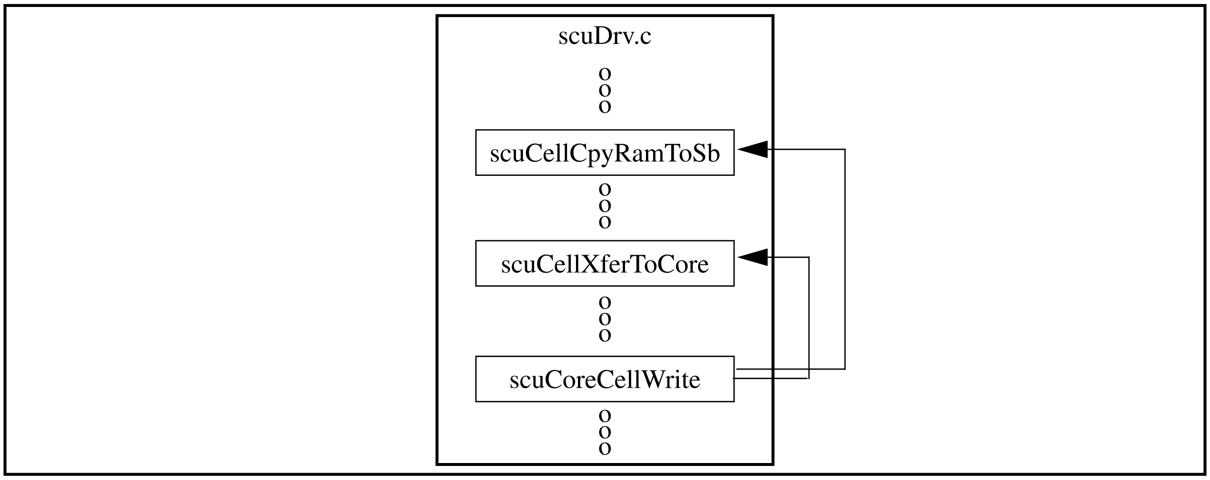


Figure 104 Function ‘scuCoreCellWrite’ Internal Interface

19.9.65 Function ‘scuCoreCellRead’

19.9.65.1 Purpose

The purpose of function ‘scuCoreCellRead’ (SCU Core Cell Read) is to read a cell from the Core Memory into GPU RAM in the form of the “Cell Data Structure” on page 44. The calling function passes ‘scuCoreCellRead’ a pointer to a cell data structure at which to place the cell, a cell index in the CIF’s Shared Buffer indicating which buffer to use for the transfer, a R_CADR indicating which cell to read from the Core Memory, an IVC and R_ID and R_LOG values.

19.9.65.2 Function

Function ‘scuCoreCellRead’ performs the following:

1. Calls function ‘scuCellXferFromCore’ passing it the following:
    - The cell index into the Shared Buffer passed from the calling function;
    - R_ID and R_LOG values passed from the calling function;
    - The IVC value passed from the calling function;
    - The R_CADR passed from the calling function;
    - A CIF Control flag with the CRC8 calculation enable, read one cell command and read cycle valid bits set;
2. Calls function ‘scmMicrosecondWait’ in module ‘scmUtil.c’ to ensure that the CIF has enough time to perform the read cycle;

19.9.64 Function ‘scuCoreCellWrite’

19.9.64.1 Purpose

The purpose of function ‘scuCoreCellWrite’ (SCU Core Cell Write) is to write a cell, in the form of “Cell Data Structure” on page 44, from GPU RAM into the Core Memory. The calling function passes ‘scuCoreCellWrite’ a pointer to a cell data structure to write to the Core Memory, a cell index in the CIF’s Shared Buffer indicating which buffer to use for the transfer and W_ID and W_LOG values.

19.9.64.2 Function

Function ‘scuCoreCellWrite’ performs the following:

1. Calls function ‘scuCellCpyRamToSb’ passing it the pointer to the cell data structure and cell index into the Shared Buffer passed from the calling function of ‘scuCoreCellWrite’;
2. Calls function ‘scuCellXferToCore’ passing it the following:
    - The cell index into the Shared Buffer passed from the calling function;
    - W_ID and W_LOG values passed from the calling function;
    - The IVC found in the cell data structure passed from the calling function;
    - A CIF Control flag with the CRC8 calculation enable, write one cell command and write cycle valid bits set;
3. Returns OK.

19.9.64.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

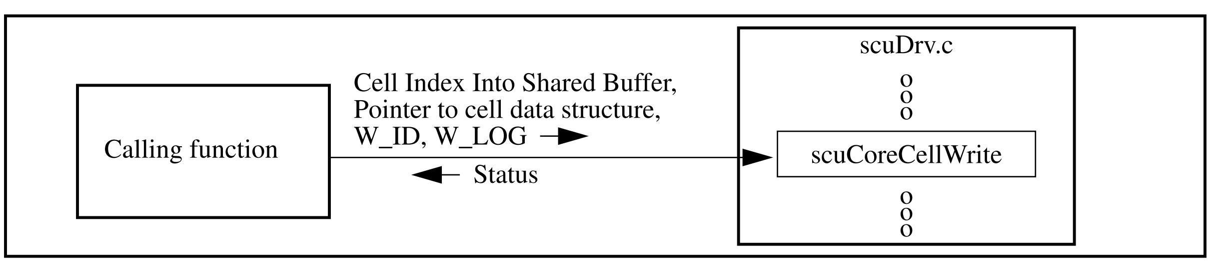


Figure 103 Function ‘scuCoreCellWrite’ External Interface

19.9.64.4 Internal interfaces

For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

19.9.63.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

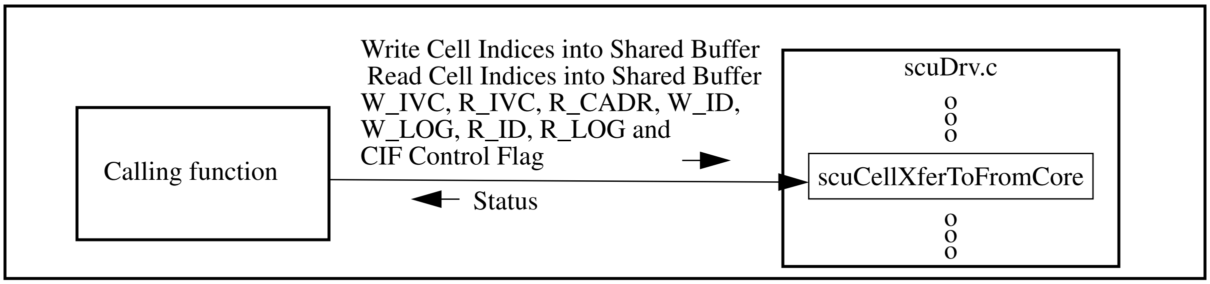


Figure 101 Function ‘scuCellXferToFromCore’ External Interface

19.9.63.4 Internal interfaces

For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

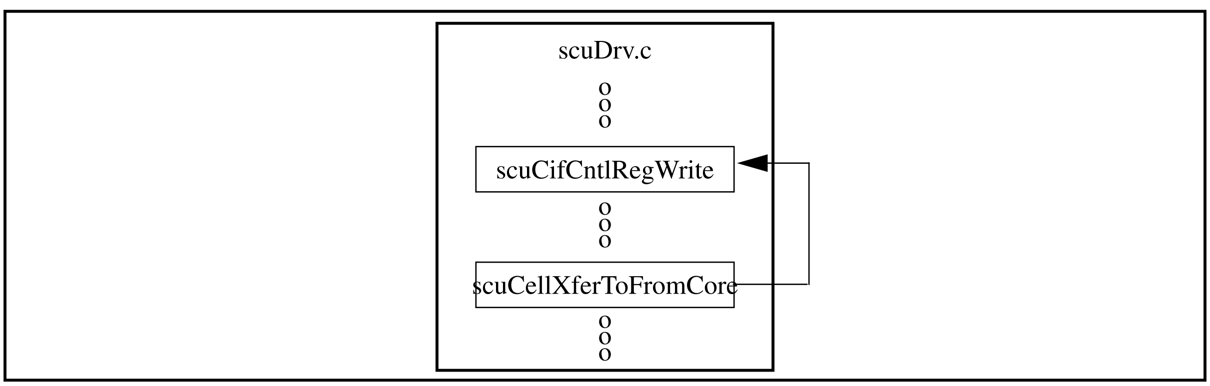


Figure 102 Function ‘scuCellXferToFromCore’ Internal Interface

19.9.63.2 Function

Function ‘scuCellXferToFromCore’ performs the following:

1. ORs together the R_ID and R_LOG passed from the calling function to produce R_ID:R_LOG and writes this to the CIF’s R_ID/R_LOG register;
2. ORs together the W_ID and W_LOG passed from the calling function to produce W_ID:W_LOG and writes this to the CIF’s W_ID/W_LOG register;
3. Returns ERROR if the R_IVC passed from the calling function is greater than or equal to 8K;
4. Returns ERROR if the R_CADR passed from the calling function is greater than or equal to 16K;
5. ORs together the R_IVC and R_CADR passed from the calling function to produce R_IVC:R_CADR and writes this to the CIF’s R_IVC_OUT/R_CADR_OUT register;
6. Returns ERROR if the W_IVC passed from the calling function is greater than or equal to 8K;
7. Writes the value of the W_IVC passed from the calling function to the CIF’s W_IVC_OUT register;
8. Returns ERROR if the destination cell index into the shared buffer passed from the calling function is greater than or equal to 512;
9. Calculates the cell’s destination address in the Shared Buffer from the destination cell index passed from the calling function;
10. Writes the address calculated above into the CIF’s Data Memory Read Address register;
11. Returns ERROR if the source cell index into the shared buffer passed from the calling function is greater than or equal to 512;
12. Calculates the cell’s source address in the Shared Buffer from the source cell index passed from the calling function;
13. Writes the address calculated above into the CIF’s Data Memory Write Address register;
14. Writes the CIF Control Flag to the CIF Control register, by calling function ‘scuCifCntlRegWrite’, to initiate both the read and write cycles;
15. Writes 0x0 to the CIF Control register to stop the CIF from continually performing the same read and write cycles;
16. Returns OK

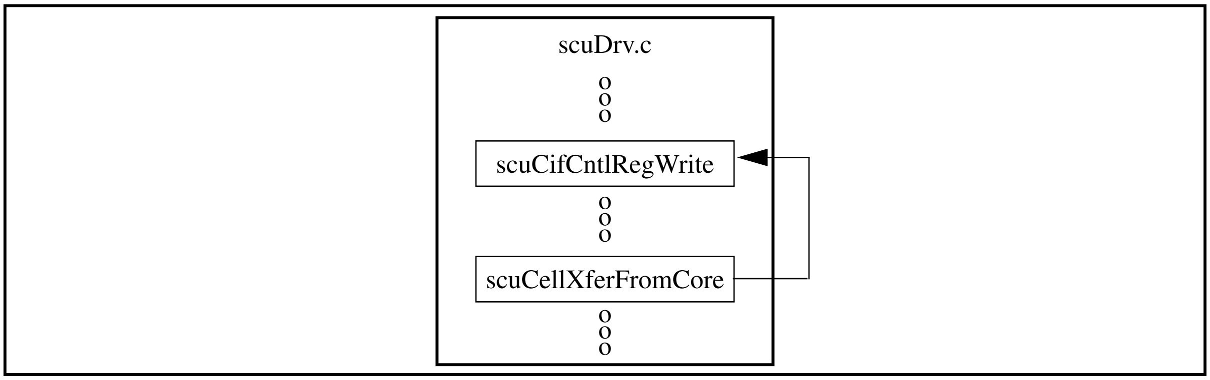


Figure 100 Function ‘scuCellXferFromCore’ Internal Interface

This function is called by functions ‘scuWDbmCapture’, ‘scuFlmFreeListInit’ and scuCoreCellRead’. Please refer to the ‘Internal Interface’ sections for these functions for more information.

19.9.63 Function ‘scuCellXferToFromCore’

19.9.63.1 Purpose

The purpose of function ‘scuCellXferToFromCore’ (SCU Cell Transfer to and from Core) is to perform both a Core Memory cell read cycle and write cycle in the same Cell Control Bus (CCB) slot. The cell are transferred between the CIF’s Shared Buffer and the Core Memory. Similar to ‘scuCellXferToCore’ and ‘scuCellXferToCore’, this function carries out the cell read and write cycles based on the following parameters passed from the calling function:

- Source cell index into the CIF’s Shared Buffer used for the write cycle;
- Destination cell index into the CIF’s Shared Buffer used for the read cycle;
- R_IVC used for the read cycle;
- W_IVC used for the write cycle;
- R_CADR;
- W_ID and W_LOG;
- R_ID and R_LOG;
- CIF Control Flag.

4. Returns ERROR if the R_CADR passed from the calling function is greater than or equal to 16K;
5. ORs together the IVC and R_CADR passed from the calling function to produce R_IVC:R_CADR and writes this to the CIF's R_IVC_OUT/R_CADR_OUT register;
6. Returns ERROR if the cell index into the shared buffer passed from the calling function is greater than or equal to 512;
7. Calculates the cell's destination address in the Shared Buffer from the cell index passed from the calling function;
8. Writes the address calculated above into the CIF's Data Memory Read Address register;
9. Writes the CIF Control Flag to the CIF Control register, by calling function 'scuCifCntlRegWrite', to initiate the write cycle;
10. Writes 0x0 to the CIF Control register to stop the CIF from continually performing the same read cycle;
11. Returns OK

19.9.62.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function 'scmRegTest' Internal Interface" on page 78.

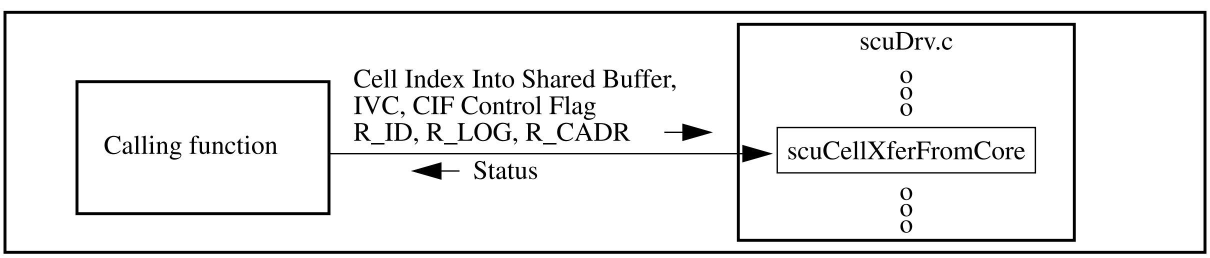


Figure 99 Function 'scuCellXferFromCore' External Interface

19.9.62.4 Internal interfaces

For more information see "Function 'scmRegTest' Internal Interface" on page 78.

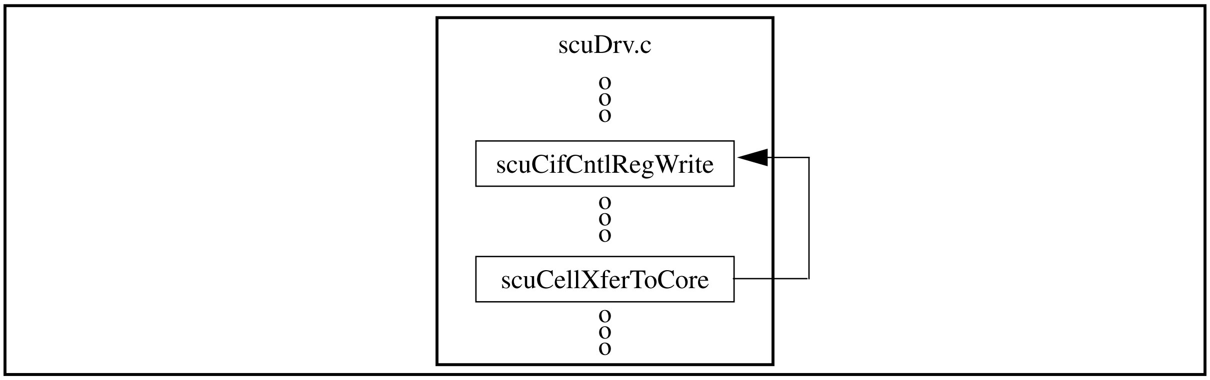


Figure 98 Function ‘scuCellXferToCore’ Internal Interface

This function is called by functions ‘scuDbmSetup’, ‘scuDbmCapture’ and ‘scuCoreCellWrite’. Please refer to the ‘Internal Interface’ sections for these functions for more information.

19.9.62 Function ‘scuCellXferFromCore’

19.9.62.1 Purpose

The purpose of function ‘scuCellXferFromCore’ (SCU Cell Transfer from Core) is to read a cell residing in the Core Memory and place it in the CIF’s Shared Buffer. The read is carried out with an IVC value passed from the calling function and cell read is placed in the CIF’s Shared Buffer at the location indicated by a cell index passed from the calling function. This function carries out the read using a CIF Control register flag, R_ID, R_LOG and R_CADR all of which are passed from the calling function.

19.9.62.2 Function

Function ‘scuCellXferFromCore’ performs the following:

1. If the CIF Control Flag passed from the calling function contains bits set which do not pertain to CIF cell read cycles then it:
    - Clears the bits in the CIF Control flag which do not pertain to a read cycle;
    - Sends a message to the Switch User Interface that bits were cleared in the CIF Control Flag;
2. ORs together the R_ID and R_LOG passed from the calling function to produce R_ID:R_LOG and writes this to the CIF’s R_ID/R_LOG register;
3. Returns ERROR if the IVC passed from the calling function is greater than or equal to 8K;

5. Returns ERROR if the cell index into the shared buffer passed from the calling function is greater than or equal to 512;
6. Calculates the cell's source address in the Shared Buffer from the cell index passed from the calling function;
7. Writes the address calculated above into the CIF's Data Memory Write Address register;
8. Writes the CIF Control Flag to the CIF Control register to initiate the write cycle;
9. Writes 0x0 to the CIF Control register to stop the CIF from continually performing the same write cycle;
10. Returns OK

19.9.61.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function 'scmRegTest' Internal Interface" on page 78.

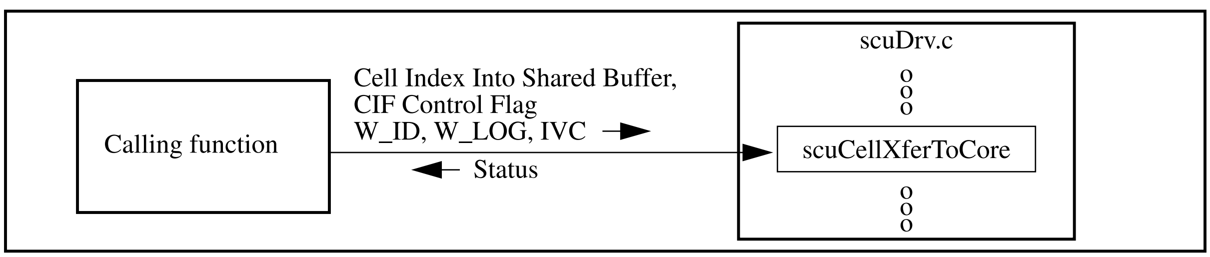


Figure 97 Function 'scuCellXferToCore' External Interface

19.9.61.4 Internal interfaces

For more information see "Function 'scmRegTest' Internal Interface" on page 78.

Figure 96 Function ‘scuFastReset’ External Interface

19.9.60.4 Internal interfaces

Not applicable.

19.9.61 Function ‘scuCellXferToCore’

19.9.61.1 Purpose

The purpose of function ‘scuCellXferToCore’ (SCU Cell Transfer to Core) is to write a cell residing in the CIF’s Shared Buffer into the Core Memory. The write is carried out with an IVC value passed from the calling function and the cell is taken from the CIF’s Shared Buffer from the location indicated by a cell index passed from the calling function. This function carries out the write using a CIF Control register flag, W_ID and W_LOG values also passed from the calling function.

19.9.61.2 Function

Function ‘scuCellXferToCore’ performs the following:

1. If the CIF Control Flag passed from the calling function contains bits set which do not pertain to CIF cell write cycles then it:
    - Clears the bits in the CIF Control flag which do not pertain to a write cycle;
    - Sends a message to the Switch User Interface indicating that bits were cleared in the CIF Control Flag;
2. ORs together the W_ID and W_LOG passed from the calling function to produce W_ID:W_LOG and writes this to the CIF’s W_ID/W_LOG register;
3. Returns ERROR if the IVC passed from the calling function is greater than or equal to 8K;
4. Writes the value of the IVC passed from the calling function to the CIF’s W_IVC_OUT register;

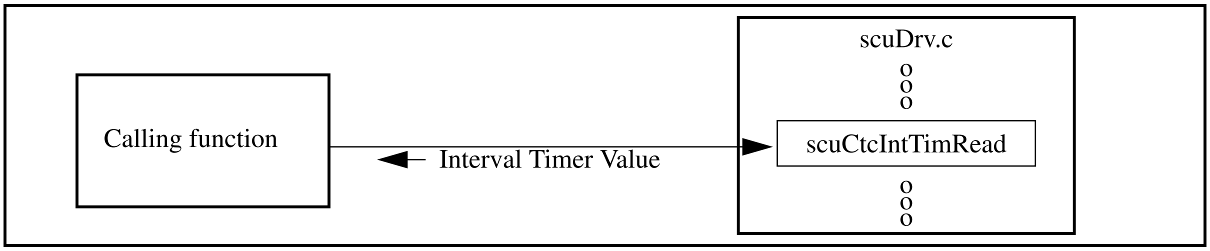


Figure 95 Function ‘scuCtcIntTimRead’ External Interface

19.9.59.4 Internal interfaces

Not applicable.

19.9.60 Function ‘scuFastReset’

19.9.60.1 Purpose

The purpose of function ‘scuFastReset’ (SCU Fast Reset) is to reset the entire SCU bringing it to a fresh power-up state. This function does not attempt to gracefully reset the SCU but instead performs the reset without checking any of the conditions within the switch or SCU.

19.9.60.2 Function

Function ‘scuFastRead’ performs the following:

1. Writes a one to the SCU reset register in order to reset the SCU;
2. Writes a zero to the SCU reset register in order allow it out of the reset state;
3. Sends a message to the switch User Interface indicating that the SCU has been reset.

19.9.60.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

19.9.58.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCtcIntTimProg’ External Interface" on page 139.

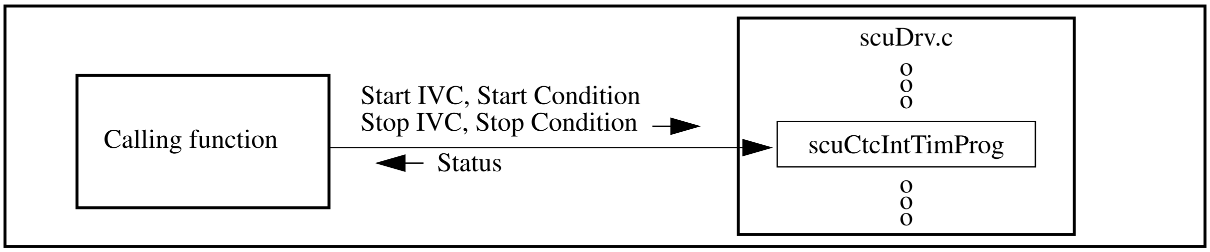


Figure 94 Function ‘scuCtcIntTimProg’ External Interface

19.9.58.4 Internal interfaces

Not applicable.

19.9.59 Function ‘scuCtcIntTimRead’

19.9.59.1 Purpose

The purpose of function ‘scuCtcIntTimRead’ (SCU Interval Timer Read) is to return the value of the CTC’s IT to the calling function.

19.9.59.2 Function

Function ‘scuCtcIntTimRead’ reads the value of the CTC’s IT and returns it to the calling function.

19.9.59.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCtcIntTimRead’ External Interface" on page 140.

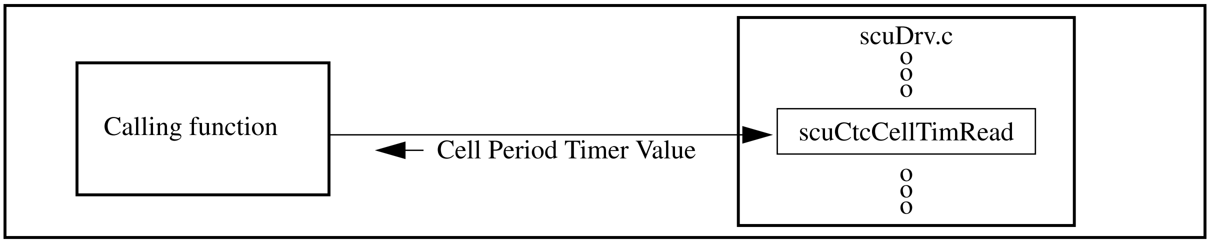


Figure 93 Function ‘scuCtcCellTimRead’ External Interface

19.9.57.4 Internal interfaces

Not applicable.

19.9.58 Function ‘scuCtcIntTimProg’

19.9.58.1 Purpose

The purpose of function ‘scuCtcIntTimProg’ (SCU CTC Interval Timer Program) is to program the CTC’s Interval Timer (IT) start and stop conditions in order to perform interval timing. The timer is programmed according to the stop IVC, start IVC, stop condition and start condition passed from the calling function.

19.9.58.2 Function

Function ‘scuCtcIntTimProg’ performs the following:

1. Reads the CTC IT by calling function ‘scuCtcIntTimRead’;
2. Reads the CTC’s IT by calling function ‘scuCtcIntTimRead’ and returns ERROR if it is not the same as the first IT value read since this would indicate that the IT is in the middle of timing. This situation also results in a message being sent to the switch User Interface to indicate this occurrence;
3. Returns ERROR if the stop IVC passed from the calling function is greater than 8K;
4. Writes the value of the stop IVC and stop condition passed from the calling function to the CTC’s Stop Count Value register;
5. Returns ERROR if the start IVC passed from the calling function is greater than 8K;
6. Writes the value of the start IVC and start condition passed from the calling function to the CTC’s Start Count Value register;
7. Returns OK.

19.9.56.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuPortCountRead’ External Interface" on page 137.

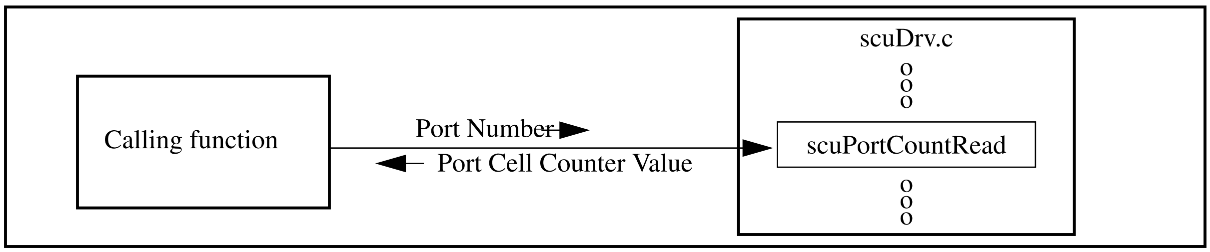


Figure 92 Function ‘scuPortCountRead’ External Interface

19.9.56.4 Internal interfaces

Not applicable.

19.9.57 Function ‘scuCtcCellTimRead’

19.9.57.1 Purpose

The purpose of function ‘scuCtcCellTimRead’ (SCU CTC Cell Period Timer Read) is to read the value of the Cell Time Counter’s (CTC’s) Cell Period Timer (CPT) and return it to the calling function.

19.9.57.2 Function

Function ‘scuCtcCellTimRead’ performs a read of the CTC’s Cell Period Timer and returns it to the calling function.

19.9.57.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "" on page 56.

19.9.55.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCoreThreshRead’ External Interface" on page 136.

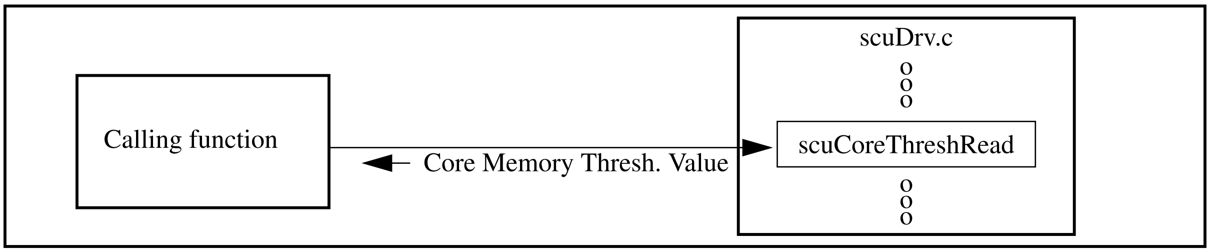


Figure 91 Function ‘scuCoreThreshRead’ External Interface

19.9.55.4 Internal interfaces

Not applicable.

19.9.56 Function ‘scuPortCountRead’

19.9.56.1 Purpose

The purpose of function ‘scuPortCountRead’ (SCU Port Cell Counter Read) is to read a port’s cell counter, located in the CRM, and returns its value to the calling function. The port’s cell counter to be read is determined by a port number passed from the calling function.

19.9.56.2 Function

Function ‘scuPortCountRead’ performs the following:

1. If the port number passed from the calling function is greater than or equal to 16 then it returns 0xFFFF to indicate an error has occurred;
2. Writes the port number passed from the calling function to the CRM’s Port Number register in order to tell the CRM which Port Cell Counter should be presented in the Port Cell Counter register
3. Reads the value from the Port Cell Counter register, multiplies it by 16 and returns it to the calling function.

Switch User Interface indicating this;

- Writes the Core Memory threshold value to the Core Memory threshold register;

3. Returns OK.

19.9.54.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCoreThreshSet’ External Interface" on page 135.

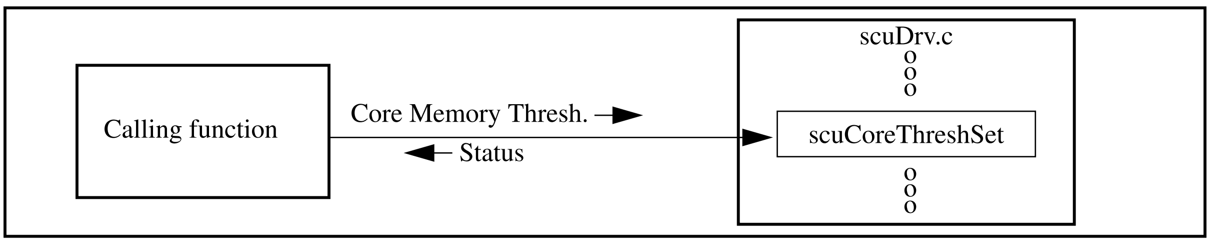


Figure 90 Function ‘scuCoreThreshSet’ External Interface

19.9.54.4 Internal interfaces

Not applicable.

19.9.55 Function ‘scuCoreThreshRead’

19.9.55.1 Purpose

The purpose of function ‘scuCoreThreshRead’ (SCU Core Thresholds Read) is to read the Core Memory Threshold register and return its value to the calling function.

19.9.55.2 Function

Function ‘scuCoreThreshRead’ performs the following:

1. Reads the Core Memory Threshold register;
2. Multiplies the value read from the Core Memory Threshold register by 16 to convert it to a number representing cells;
3. Returns the value of the Core Memory thresold.

19.9.53.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuPortThreshRead’ External Interface" on page 134.

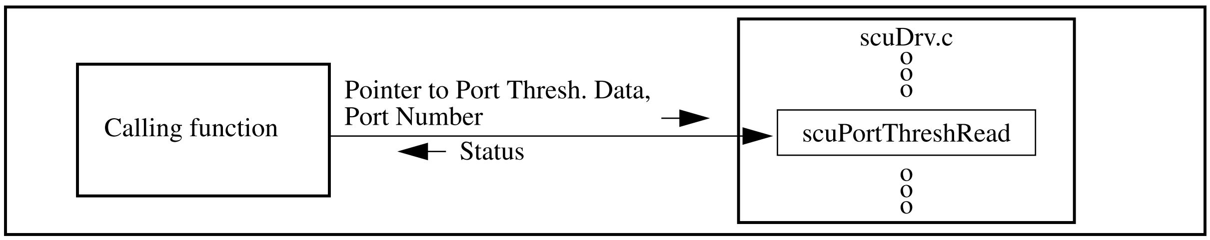


Figure 89 Function ‘scuPortThreshRead’ External Interface

19.9.53.4 Internal interfaces

This function is called by functions ‘scuCoreClkAlarmISR’, ‘scuFlmEmptyISR’and ‘scuFlmFailISR’. For more information, please see the ‘Internal Interface’ sections of these functions.

19.9.54 Function ‘scuCoreThreshSet’

19.9.54.1 Purpose

The purpose of function ‘scuCoreThreshSet’ (SCU Core Memory Threshold Set) is to the configure the CRM’s Core Memory threshold according to a value passed from the calling function. Since the CRM’s Core Memory Threshold register only holds the 10 most significant bits of the threshold, this function must first divide the threshold value passed from the calling procedure by 16 before writing to the register.

19.9.54.2 Function

Function ‘scuCoreThreshSet’ performs the following:

1. If the Core Memory threshold passed from the calling function is greater than 16K then it returns ERROR;
2. If the Core Memory threshold passed from the calling function is less than 16K then it:
    - Divides the threshold by 16 in order to place the 10 most significant bits of the threshold in the 10 least significant bit locations in the threshold register;
    - If the thresholds does not divide evenly by 16 then a warning message is sent to the

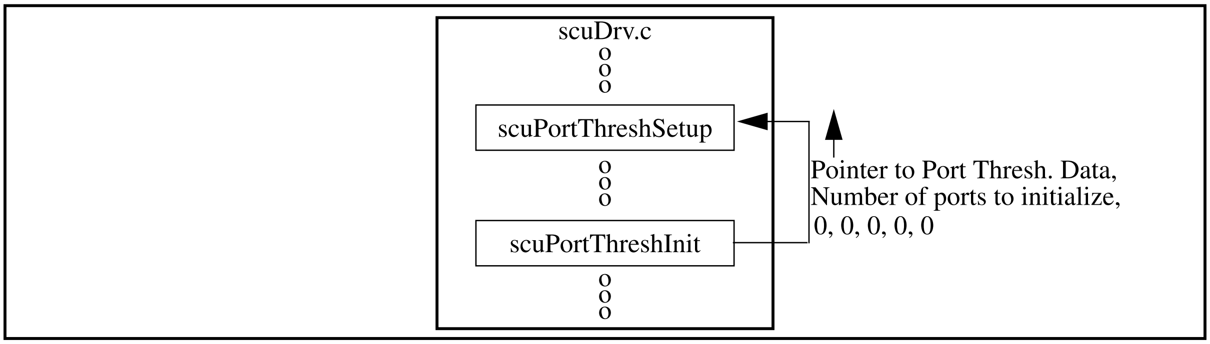


Figure 88 Function ‘scuDbmInit’ Internal Interface

19.9.53 Function ‘scuPortThreshRead’

19.9.53.1 Purpose

The purpose of function ‘scuPortThreshRead’ (SCU Port Thresholds Read) is to read a port’s thresholds and store them at a location pointed to by a Port Threshold Data Structure pointer passed from the calling function.

19.9.53.2 Function

Function ‘scuPortThreshRead’ performs the following:

1. If the port number passed from the calling function is greater than or equal to 16 then it returns ERROR;
2. If the port number is less than 16 then it does the following:
   - Reads the Port Threshold Register indicated by the port number passed from the calling function.
   - Breaks the fields of the Port Threshold Register read above into four bytes, one for each threshold (please see "CRM Threshold Registers Bitmap" on page 130);
   - Multiplies each of the four thresholds by 64 (in order to convert them to a number of cells) and stores them in their respective fields of the Port Threshold Data Structure pointed to by the Port Threshold Data Structure pointer passed from the calling function;
3. Returns OK.

19.9.51.4 Internal interfaces

Not applicable.

19.9.52 Function ‘scuPortThreshInit’

19.9.52.1 Purpose

The purpose of function ‘scuPortThreshInit’ (SCU Port Thresholds Initialization) is to the configure the CRM’s port thresholds for a set of ports. Calling functions pass it a pointer to an array of Port Threshold data structures and the number of port’s thresholds to initialize.

19.9.52.2 Function

Function ‘scuPortThreshInit’ performs the following:

1. Calls ‘scuPortThreshSet’ passing it the pointer to Port Threshold Data structures and the number of ports to initialize passed from the calling function. It also passes a port number of 0 and four port thresholds set to 0;
2. Returns the value returned by ‘scuPortThreshSet.

19.9.52.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuPortThreshInit’ External Interface" on page 132.

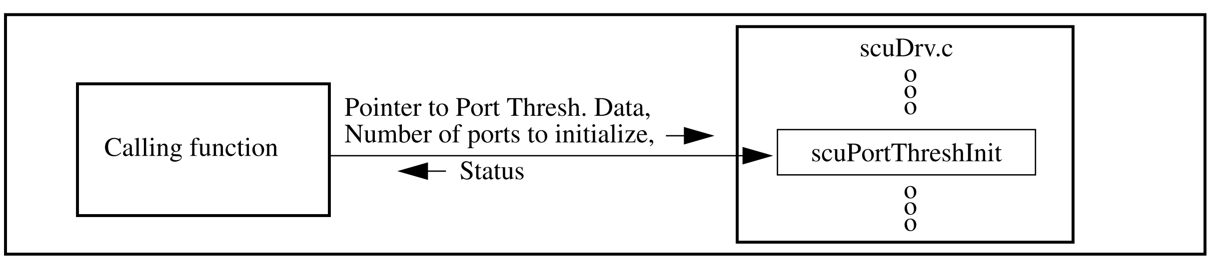


Figure 87 Function ‘scuPortThreshInit’ External Interface

19.9.52.4 Internal interfaces

For more information see "Function ‘scuDbmInit’ Internal Interface" on page 133.

19.9.51 Function ‘scuSinglePortThreshSet’

19.9.51.1 Purpose

The purpose of function ‘scuSinglePortThreshSet’ (SCU Single Port Thresholds Set) is to the configure the CRM’s port thresholds for one port. Calling functions pass it a a port number, a lower BF Threshold, a middle BF Threshold, an upper BF Threshold and an RPI threshold. Since the actual hardware thresholds only store the most significant 8 bits for comparison with 14 bit counters, this function is also responsible for adjusting the values of the threshold supplied by the calling function appropriately.

19.9.51.2 Function

Function ‘scuSinglePortThreshSet’ performs the following:

1. If the port number is greater then 16 then it returns ERROR;
2. Divides each threshold by 64 in order to place the 8 most significant bits of the threshold in the 8 least significant bit locations;
3. If any thresholds do not divide evenly by 64 then a warning message is sent to the Switch User Interface indicating this.
4. ORs the four remaining eight thresholds together according to the following diagram:
5. Writes the port threshold word calculated above to the appropriate port’s threshold register;
6. Returns OK.

19.9.51.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuSinglePortThreshSet’ External Interface" on page 131.

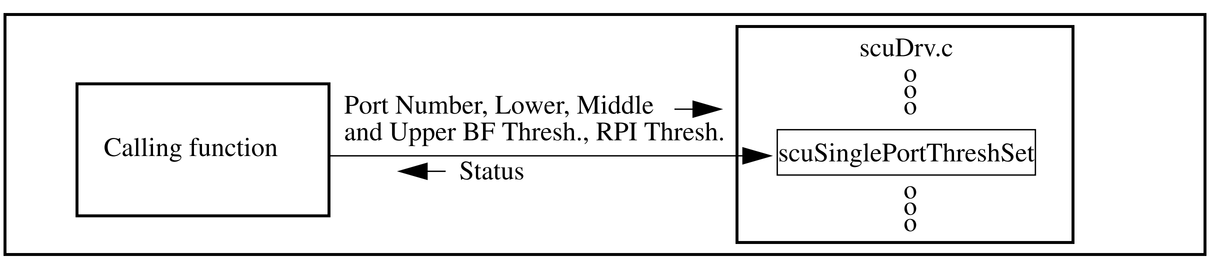


Figure 86 Function ‘scuSinglePortThreshSet’ External Interface

31–24	23–16	15–8	7–0
RPI Threshold	Upper BF Threshold	Middle BF Threshold	Lower BF Threshold

Figure 84 CRM Threshold Registers Bitmap

- Writes the port threshold word calculated above to the appropriate port's threshold register;

4. If the number ports to initialize is 1 then places the port threshold and port numbers passed from the calling function into a port threshold data structure and performs step 2;
5. Returns OK.

19.9.50.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function 'scuPortThreshSet' External Interface" on page 130.

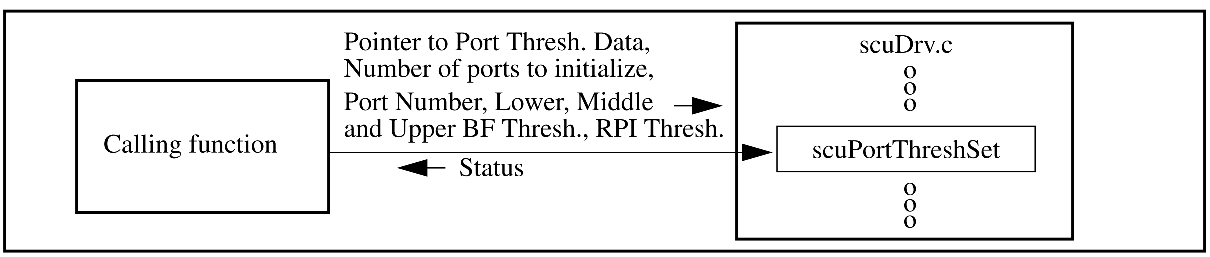


Figure 85 Function 'scuPortThreshSet' External Interface

19.9.50.4 Internal interfaces

Not applicable.

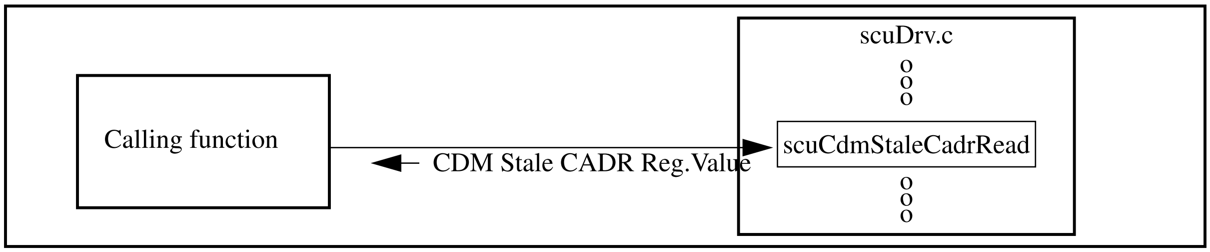


Figure 83 Function ‘scuCdmStaleCadrRead’ External Interface

19.9.49.4 Internal interfaces

Not applicable.

19.9.50 Function ‘scuPortThreshSet’

19.9.50.1 Purpose

The purpose of function ‘scuPortThreshSet’ (SCU Port Thresholds Set) is to the configure the CRM’s port thresholds for one or all ports. Calling functions pass it a pointer to an array of Port Threshold data structures, the number of port’s thresholds to initialize, a port number, a lower BF Threshold, a middle BF Threshold, an upper BF Threshold and an RPI threshold. Since the actual hardware thresholds only store the most significant 8 bits for comparison with 14 bit counters, this function is also responsible for adjusting the values of the threshold supplied by the calling function appropriately.

19.9.50.2 Function

Function ‘scuPortThreshSet’ performs the following:

1. If the number of ports to initialize is greater then 16 then it returns ERROR;
2. If the port number is greater then 16 then it returns ERROR;
3. If the number of port to initialize is less than or equal to 16 then it does the following for each port threshold data structure in the array of port threshold data structures:
    - Divides each threshold by 64 in order to place the 8 most significant bits of the threshold in the 8 least significant bit locations;
    - If any thresholds do not divide evenly by 64 then a warning message is sent to the Switch User Interface indicating this.
    - ORs the four remaining eight thresholds together according to the following diagram:

19.9.48.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCdmCntlRegWrite’ External Interface" on page 127.

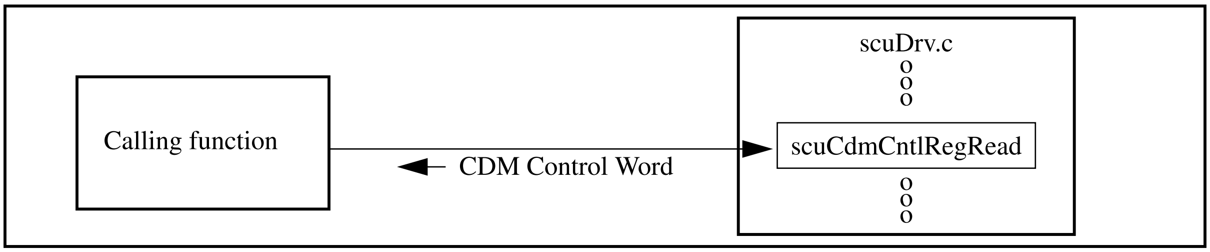


Figure 82 Function ‘scuCDMCntlRegRead’ External Interface

19.9.48.4 Internal interfaces

Not applicable.

19.9.49 Function ‘scuCdmStaleCadrRead’

19.9.49.1 Purpose

The purpose of function ‘scuCdmStaleCadrRead’ (SCU CDM Stale CADR Register Read) is to return the value of the CDM’s stale CADR register to the calling function.

19.9.49.2 Function

This function performs a read of the CDM’s tale CADR register and returns it to the calling function.

19.9.49.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCdmStatRegRead’ External Interface" on page 126.

19.9.47.2 Function

This function writes the control word passed to it by the calling function to the CDM Control register.

19.9.47.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCdmCntlRegWrite’ External Interface" on page 127.

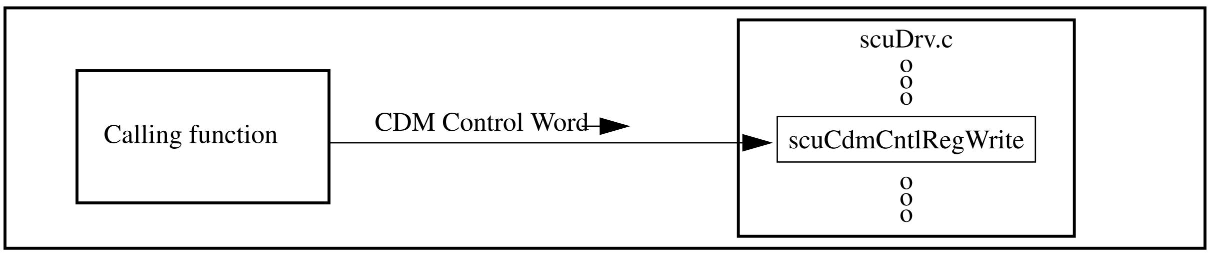


Figure 81 Function ‘scuCdmCntlRegWrite’ External Interface

19.9.47.4 Internal interfaces

Not applicable.

<u>19.9.48 Function ‘scuCdmCntlRegRead’</u>

19.9.48.1 Purpose

The purpose of function ‘scuCdmCntlRegRead’ (SCU CDM Control Register Read) is to read the CDM Control Register and return the value read to the calling function.

19.9.48.2 Function

This function reads the value of the CDM’s 16-bit control register and returns it to the calling function.

19.9.45.4 Internal interfaces

Not applicable.

19.9.46 Function ‘scuCdmStatRegRead’

19.9.46.1 Purpose

The purpose of function ‘scuCdmStatRegRead’ (SCU CDM Status Register Read) is to return the value of the CDM’s Status register to the calling function.

19.9.46.2 Function

This function performs a read of the CDM’s Status register and returns it to the calling function.

19.9.46.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCdmStatRegRead’ External Interface" on page 126.

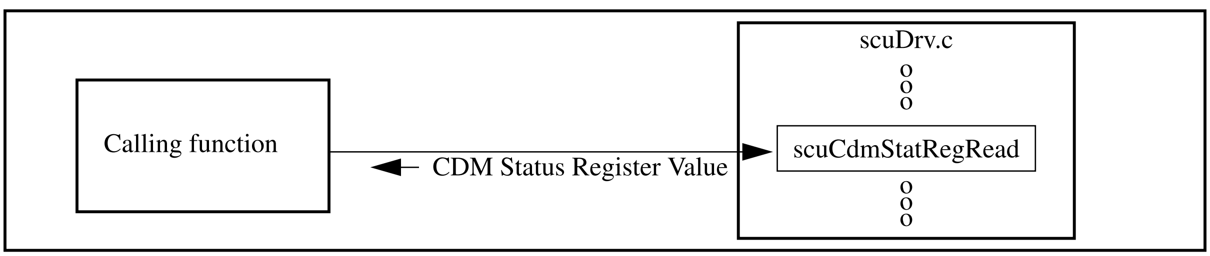


Figure 80 Function ‘scuCdmStatRegRead’ External Interface

19.9.46.4 Internal interfaces

Not applicable.

19.9.47 Function ‘scuCdmCntlRegWrite’

19.9.47.1 Purpose

The purpose of function ‘scuCdmCntlRegWrite’ (SCU CDM Control Register Write) is to write a control word passed from the calling function to the CDM Control register.

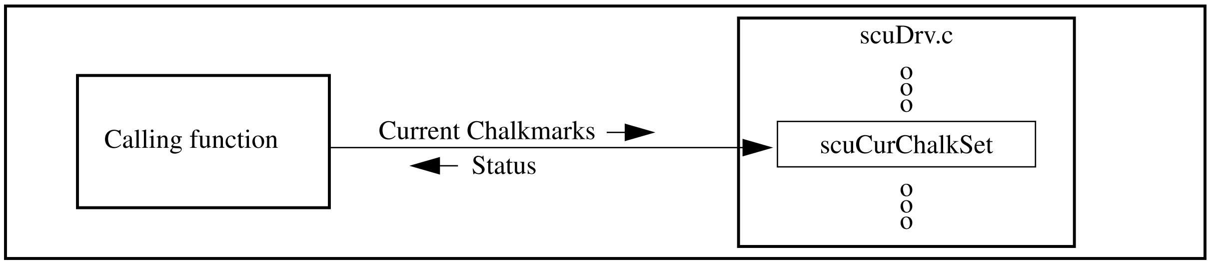


Figure 78 Function ‘scuCurChalkSet’ External Interface

19.9.44.4 Internal interfaces

Not applicable.

19.9.45 Function ‘scuCurChalkRead’

19.9.45.1 Purpose

The purpose of function ‘scuCurChalkRead’ (SCU Current Chalkmark Read) is to return the value of the current chalkmarks (multicast and unicast) to the calling function.

19.9.45.2 Function

Function ‘scuCurChalkRead’ reads the value of the CDM’s Current Chalkmark register and returns it to the calling function.

19.9.45.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCurChalkRead’ External Interface" on page 125.

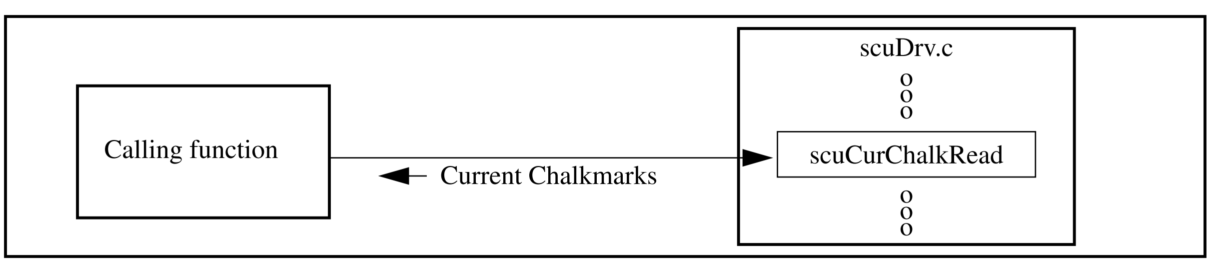


Figure 79 Function ‘scuCurChalkRead’ External Interface

19.9.43.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuChalkSearchReset’ External Interface" on page 124.

Figure 77 Function ‘scuChalkSearchReset’ External Interface

19.9.43.4 Internal interfaces

Not applicable.

19.9.44 Function ‘scuCurChalkSet’

19.9.44.1 Purpose

The purpose of function ‘scuCurChalkSet’ (SCU Current Chalkmark Set) is to set the value of the current chalkmarks (multicast and unicast) to the value passed from the calling function.

19.9.44.2 Function

Function ‘scuCurChalkSet’ performs the following:

1. Returns ERROR if the value of the current chalkmarks passed by the calling function are three or more;
2. Writes the current chalkmark values passed from the calling function to the CDM’s Current Chalkmark register;
3. Returns OK.

19.9.44.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCurChalkSet’ External Interface" on page 125.

19.9.42.2 Function

Function ‘scuChalkSearchDisable’ performs a write to the CDM’s Control register to clear the Chalkmark Search Enable bit.

19.9.42.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuChalkSearchDisable’ External Interface" on page 123.

Figure 76 Function ‘scuChalkSearchDisable’ External Interface

19.9.42.4 Internal interfaces

Not applicable.

19.9.43 Function ‘scuChalkSearchReset’

19.9.43.1 Purpose

The purpose of function ‘scuChalkSearchReset’ (SCU Chalkmark Search Reset) is to reset the CDM’s chalkmark search hardware so that it begins searching the Working DBM memory space chalkmarks at the first location the next time the search is enabled.

19.9.43.2 Function

Function ‘scuChalkSearchReset’ performs the following:

1. Writes to the CDM’s Control register to clear the Chalkmark Search Enable bit;
2. Writes to the CDM’s Control register to set the Chalkmark Reset bit and cause the chalkmark reset;
3. Writes to the CDM’s Control register to clear the Chalkmark Reset bit allowing the chalkmark searching hardware out of reset.

19.9.41 Function ‘scuChalkSearchEnable’

19.9.41.1 Purpose

The purpose of function ‘scuChalkSearchEnable’ (SCU Chalkmark Search Enable) is to instruct the CDM to begin/continue performing a chalkmark search on the Working DBMs in the CDM’s Working DBM memory space.

19.9.41.2 Function

Function ‘scuChalkSearchEnable’ performs the following a write to the CDM’s Control register to set the Chalkmark Search Enable bit.

19.9.41.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuChalkSearchEnable’ External Interface" on page 122.

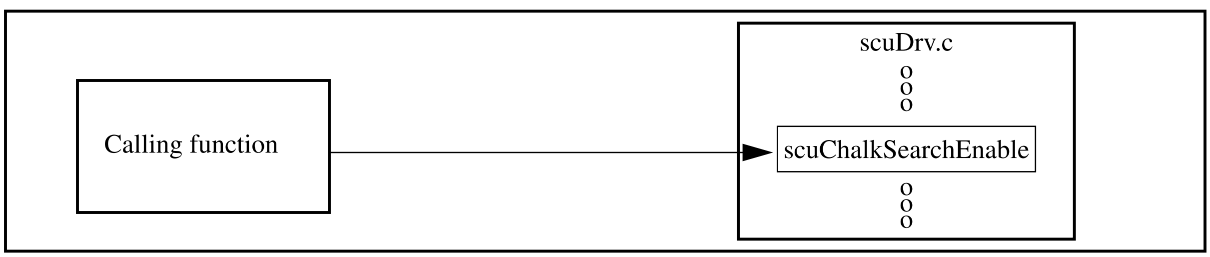


Figure 75 Function ‘scuChalkSearchEnable’ External Interface

19.9.41.4 Internal interfaces

Not applicable.

19.9.42 Function ‘scuChalkSearchDisable’

19.9.42.1 Purpose

The purpose of function ‘scuChalkSearchDisable’ (SCU Chalkmark Search Disable) is to instruct the CDM to discontinue performing a chalkmark search on the Working DBMs in the CDM’s Working DBM memory space.

unasserted; the DBM capture bit in the CIF Control register asserted; and the R_CADR_OUT register contents equal to the CADR passed by the calling function of ‘scu-WDbmCapture’.

2. Returns the value read out of the requested DBM register.

19.9.40.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuWDbmCapture’ External Interface" on page 121.

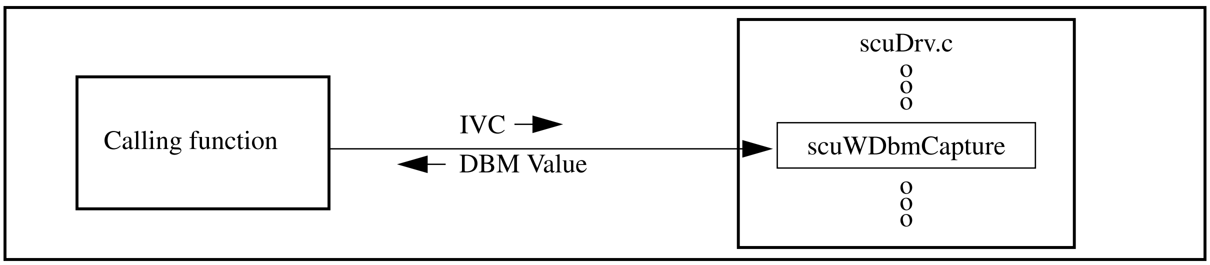


Figure 73 Function ‘scuWDbmCapture’ External Interface

19.9.40.4 Internal interfaces

For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

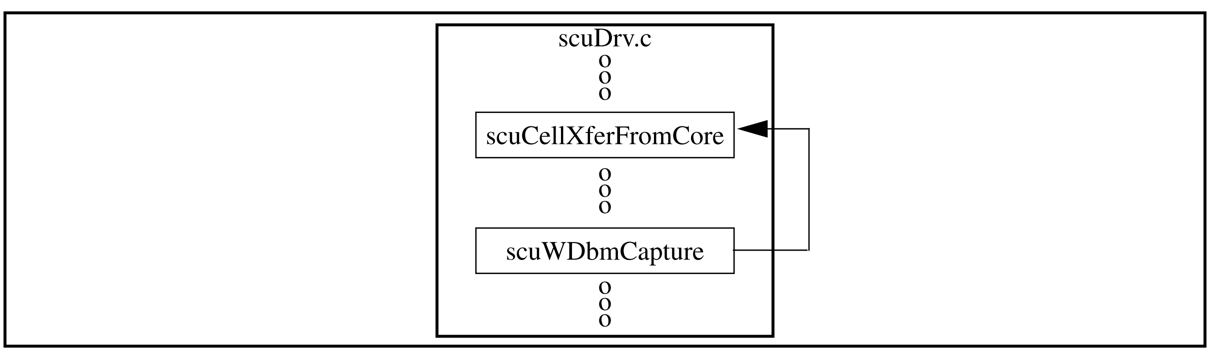


Figure 74 Function ‘scuWDbmCapture’ Internal Interface

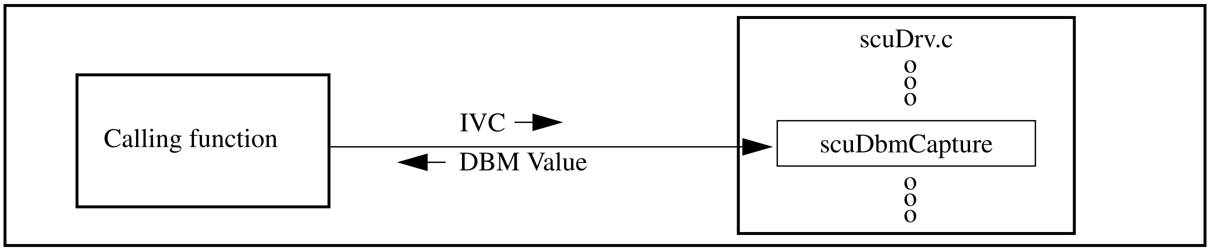


Figure 71 Function ‘scuDbmCaptuire’ External Interface

19.9.39.4 Internal interfaces

For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

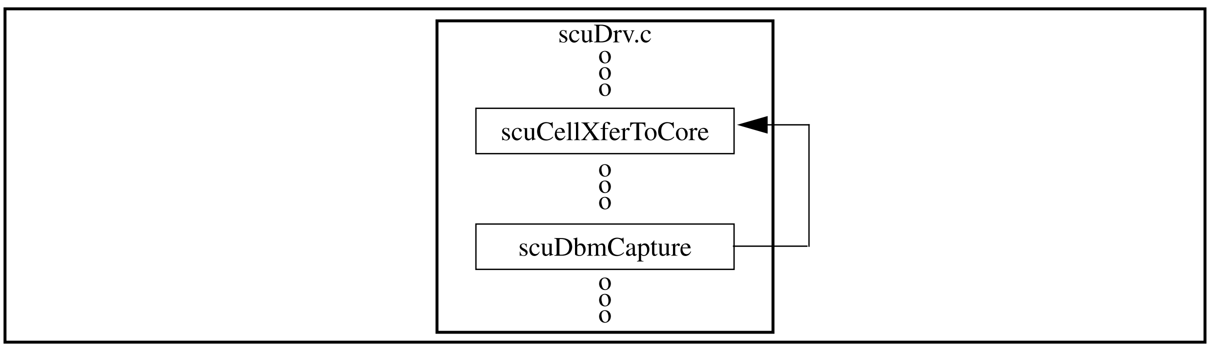


Figure 72 Function ‘scuDbmCapture’ Internal Interface

<u>19.9.40 Function ‘scuWDbmCapture’</u>

19.9.40.1 Purpose

The purpose of function ‘scuWDbmCapture’ (SCU Working DBM Capture) is to return the value of the Working DBM entry (including DBM, casting bit and chalkmark bit) in the CDM’s Working DBM memory space at the location indexed by the value of a CADR passed from the calling function.

19.9.40.2 Function

Function ‘scuWDbmCapture’ performs the following:

1. Performs a Core Memory read cycle by calling function ‘scuCellXferFromCore’ instructing it to perform the read cycle with the following: R_VALID bit in the CIF Control register

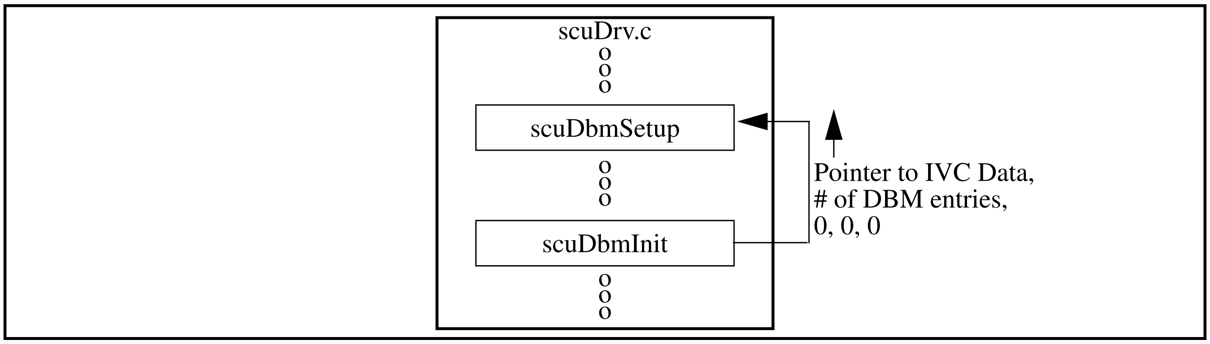


Figure 70 Function ‘scuDbmInit’ Internal Interface

This function is also called by function ‘scuIvcSet’.

19.9.39 Function ‘scuDbmCapture’

19.9.39.1 Purpose

The purpose of function ‘scuDbmCapture’ (SCU DBM Capture) is to return the value of the DBM entry in the CDM’s DBM memory space at the location indexed by the value of an IVC passed from its calling function.

19.9.39.2 Function

Function ‘scuDbmCapture’ performs the following:

1. Performs a Core Memory write cycle by calling function ‘scuCellXferToCore’ instructing it to perform the write cycle with the following: W_VALID bit in the CIF Control register unasserted; the DBM capture bit in the CIF Control register asserted; and the W_IVC_OUT register contents equal to the IVC passed by the calling function of ‘scuDbmSetup’.
2. Returns the value read out of the Requested DBM register.

19.9.39.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuDbmCaptuire’ External Interface" on page 120.

19.9.38 Function ‘scuDbmInit’

19.9.38.1 Purpose

The purpose of function ‘scuDbmInit’ (SCU DBM Initialization) is to configure the casting bit (multicast/unicast bit) and Distribution Bitmap (DBM) for a set of IVCs. Calling functions pass it the pointer to an array of IVC data structures and the number of DBM entries to initialize.

19.9.38.2 Function

Function ‘scuDbmInit’ performs the following:

1. Calls function ‘scuDbmSetup passing it the number of DBMs to initialize and pointer to the array of IVC data structures passed by the calling function as well as an IVC, DBM and casting bit all set to 0x0;
2. Returns the value returned by ‘scuDbmSetup’.

19.9.38.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuDbmInit’ External Interface" on page 118.

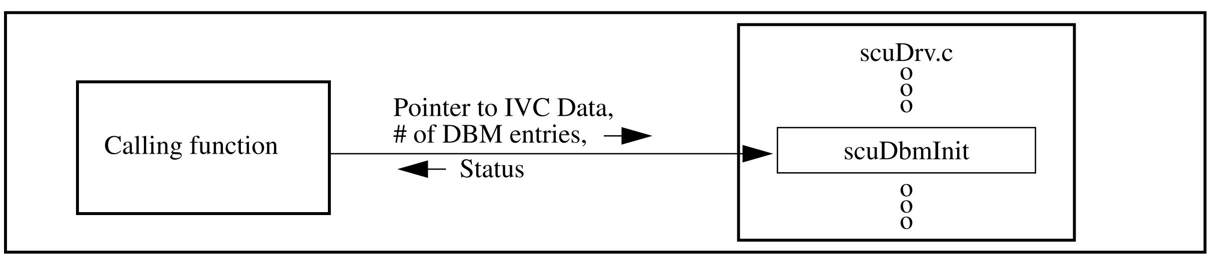


Figure 69 Function ‘scuDbmInit’ External Interface

19.9.38.4 Internal interfaces

For more information see "Function ‘scuDbmInit’ Internal Interface" on page 119.

6. Returns OK.

19.9.37.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuDbmSet’ External Interface" on page 117.

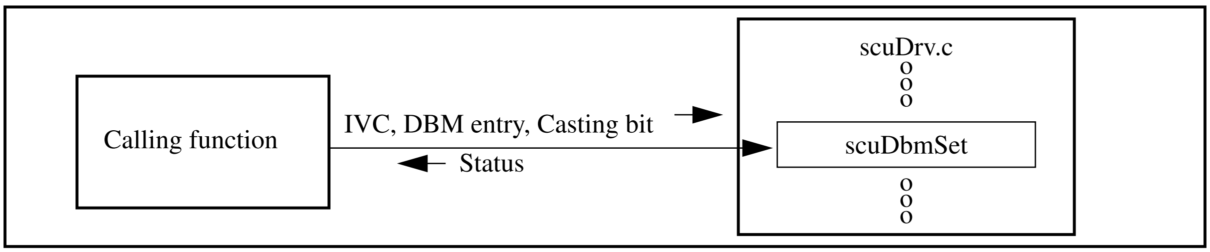


Figure 67 Function ‘scuDbmSet’ External Interface

19.9.37.4 Internal interfaces

For more information see "Function ‘scuDbmSet’ Internal Interface" on page 117.

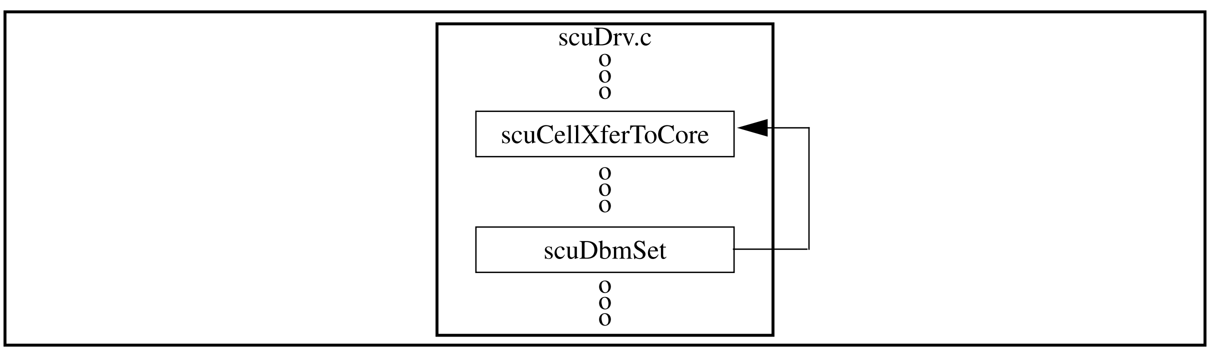


Figure 68 Function ‘scuDbmSet’ Internal Interface

This function is also called by function ‘scuIvcSet’.

19.9.36.4 Internal interfaces

For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

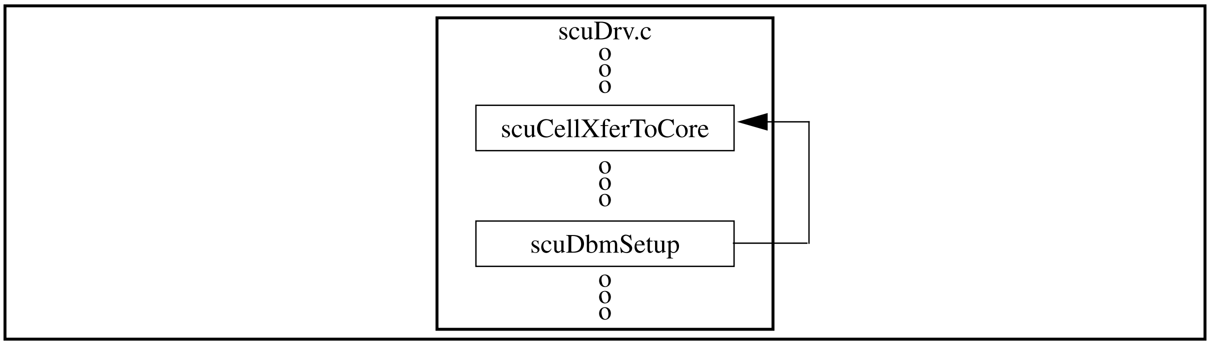


Figure 66 Function ‘scuDbmSetup’ Internal Interface

This function is also called by function ‘scuIvcSetup’. For more information see "Function ‘scuIvcSetup’" on page 151.

19.9.37 Function ‘scuDbmSet’

19.9.37.1 Purpose

The purpose of function ‘scuDbmSet’ (SCU DBM Set) is to configure the casting bit (multicast/unicast bit) and Distribution Bitmap (DBM) for a single IVC. Calling functions pass it an IVC, a DBM and a casting bit value.

19.9.37.2 Function

Function ‘scuDbmSetup’ performs the following:

1. If the IVC passed by the calling function is greater than 16K then it returns ERROR:
2. If the casting bit passed by the calling function is not 0 or 1 then it returns ERROR:
3. Writes the DBM value passed by the calling function to the DBM Setup register;
4. Writes the casting bit value passed by the calling function to the DBM Casting register;
5. Performs a Core Memory write cycle by calling function ‘scuCellXferToCore’ instructing it to perform the write cycle with the following: W_VALID bit in the CIF Control register unasserted; the DBM configure bit in the CIF Control register asserted; and the CIF’s W_IVC_OUT register contents equal to the IVC passed by the calling function of ‘scuDbmSetup’.

19.9.36.2 Function

Function ‘scuDbmSetup’ performs the following:

1. If the number of DBM entries to initialize is zero or greater than 16K then it returns ERROR;
2. If the number of DBM entries to initialize is one then it does the following:
    - If the IVC passed by the calling function is greater than 16K then it returns ERROR:
    - If the casting bit passed by the calling function is not 0 or 1 then it returns ERROR:
    - Writes the DBM value passed by the calling function to the DBM Setup register;
    - Writes the casting bit value passed by the calling function to the DBM Casting register;
    - Performs a Core Memory write cycle by calling function ‘scuCellXferToCore’ instructing it to perform the write cycle with the following: W_VALID bit in the CIF Control register unasserted; the DBM configure bit in the CIF Control register asserted; and the CIF’s W_IVC_OUT register contents equal to the IVC passed by the calling function of ‘scuDbmSetup’.
3. If the number of DBM entries to initialize is two or more, then this function repeats step 2 once for each DBM entry to be initialized. The values written to the DBM Setup, DBM Casting and CIF’s W_IVC_OUT register are instead taken from the array of IVC data structures passed to ‘scuDbmSetup’;
4. Returns OK.

19.9.36.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scmWalkD1Test’ External Interface" on page 73.

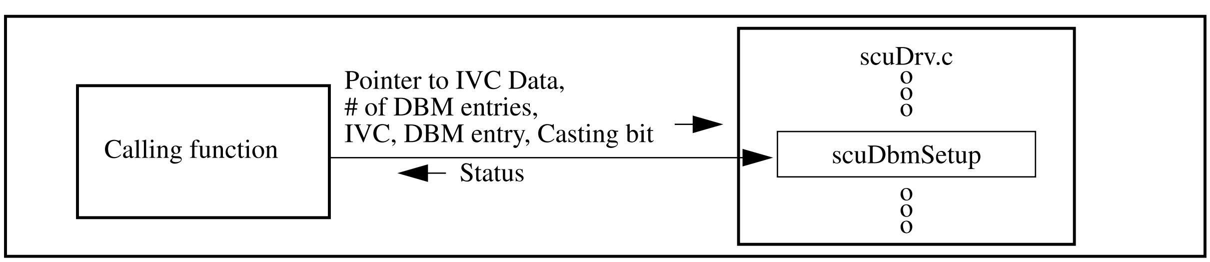


Figure 65 Function ‘scuDbmSetup’ External Interface

19.9.35.2 Function

Function ‘scuFlmPrime’ performs the following:

1. Sets the FLM Prime bit in the FLM Control register;
2. Returns.

19.9.35.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuFlmPrime’ External Interface" on page 114.

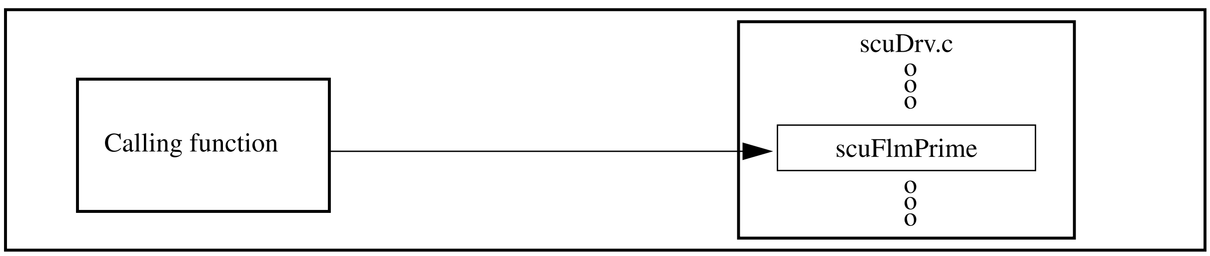


Figure 64 Function ‘scuFlmPrime’ External Interface

19.9.35.4 Internal interfaces

Not applicable.

19.9.36 Function ‘scuDbmSetup’

19.9.36.1 Purpose

The purpose of function ‘scuDbmSetup’ (SCU DBM Setup) is to configure the casting bit (multicast/unicast bit) and Distribution Bitmap (DBM) for one IVC or a list of IVCs. Calling functions pass it the pointer to an array of IVC data structures, the number of DBM entries to initialize, an IVC, a DBM value for the preceding IVC and a casting bit value for the preceding IVC.

19.9.34 Function ‘scuFlmReset’

19.9.34.1 Purpose

The purpose of ‘scuFlmReset’ (SCU FLM Reset) is to reset the FLM to its initial state which includes empty all CADRs from the Free List

19.9.34.2 Function

Function ‘scuFlmReset’ performs the following:

1. Write to the FLM Control register setting the FLM Reset bit;
2. Write to the FLM Control register clearing the FLM Reset bit;

19.9.34.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuFlmHaltClear’ External Interface" on page 112.

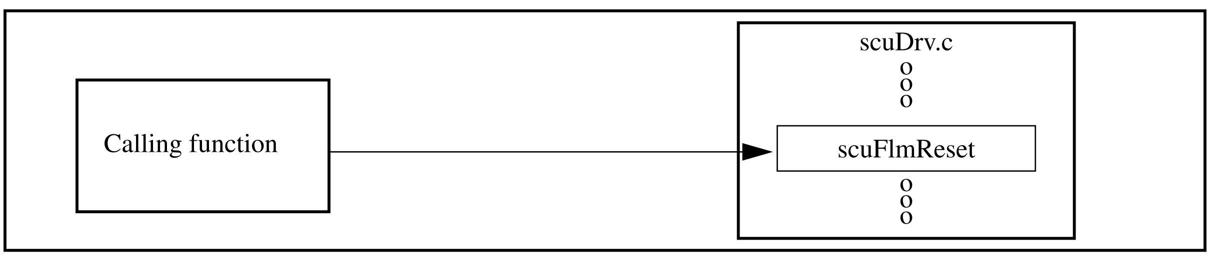


Figure 63 Function ‘scuFlmReset’ External Interface

19.9.34.4 Internal interfaces

Not applicable.

19.9.35 Function ‘scuFlmPrime’

19.9.35.1 Purpose

The purpose of ‘scuFlmPrime’ (SCU FLM Prime) is to prepare the FLM to output the CADRs place on the Free CADR List. The priming action is required to flush out two irrelevant addresses from a pipeline which lies between the FLM RAM and the backplane and place the first two CADRs found in the RAM in the pipeline.

19.9.32.4 Internal interfaces

Not applicable.

19.9.33 Function ‘scuFlmHaltClear’

19.9.33.1 Purpose

The purpose of ‘scuFlmHaltClear’ (SCU FLM Halt Clear) is to clear an FLM halt enabling it to pop W_CADRs off the Free CADR List, assert FLM Fail and FLM Wrapped and increment the CMBF Count when appropriate.

19.9.33.2 Function

Function ‘scuFlmHaltClear’ performs the following:

1. Write to the FLM Control register clearing the FLM Halt bit;
2. Sends a message to the switch user interface indicating that an FLM halt has been cleared.

19.9.33.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuFlmHaltClear’ External Interface" on page 112.

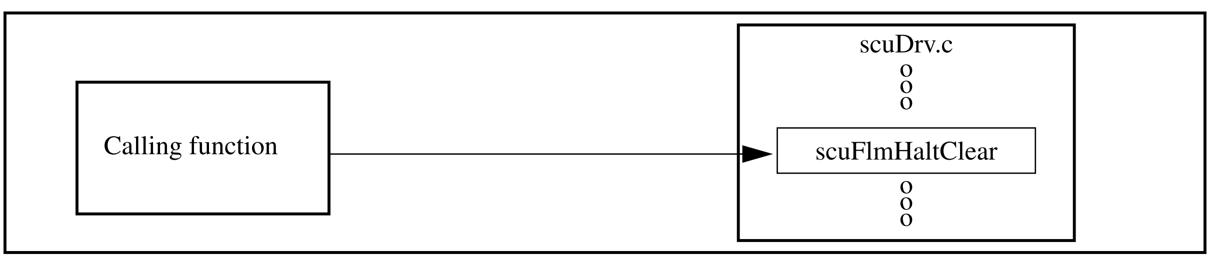


Figure 62 Function ‘scuFlmHaltClear’ External Interface

19.9.33.4 Internal interfaces

Not applicable.

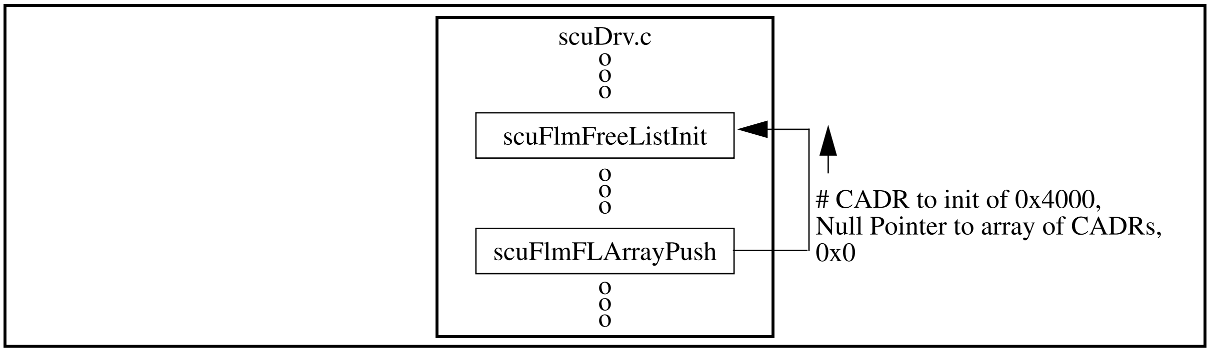


Figure 60 Function ‘scuFlmFLCInit’ Internal Interface

19.9.32 Function ‘scuFlmHalt’

19.9.32.1 Purpose

The purpose of ‘scuFlmHalt’ (SCU FLM Halt) is to halt the FLM from popping W_CADRs off the Free CADR List, asserting FLM Fail and FLM Wrapped and incrementing the CMBF Count.

19.9.32.2 Function

Function ‘scuFlmHalt’ performs the following:

1. Writes to the FLM Control register setting the FLM Halt bit;
2. Sends a message to the switch user interface indicating that the FLM has been halted.

19.9.32.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuFlmHalt’ External Interface" on page 111.

Figure 61 Function ‘scuFlmHalt’ External Interface

19.9.31 Function ‘scuFlmFLCInit’

19.9.31.1 Purpose

The purpose of ‘scuFlmFLCInit’ (SCU FLM Free CADR List Complete Initialization) to place all CADRs from 0 to 16K-1 onto the FLM’s Free CADR List.

19.9.31.2 Function

Function ‘scuFlmFLCInit’ performs the following:

1. Calls function ‘scuFlmFreeListInit’ passing it a number of CADRs to initialize of 16K, a NULL pointer and a single CADR of 0x0;
2. Returns the result returned by ‘scuFlmFreeListInit’.

19.9.31.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuFlmFLCInit’ External Interface" on page 110.

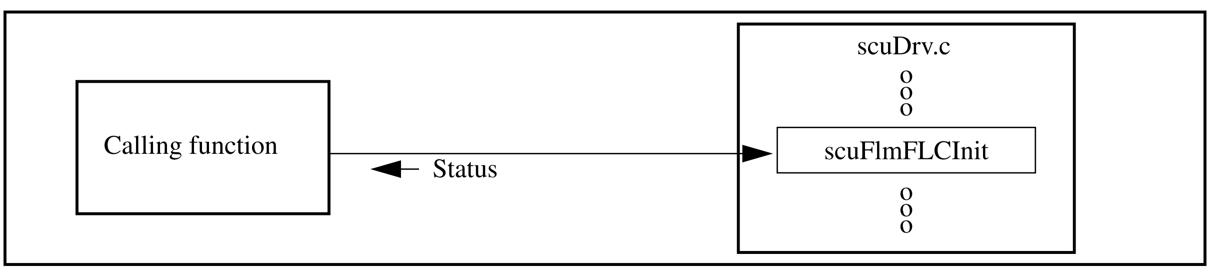


Figure 59 Function ‘scuFlmFLCInit’ External Interface

19.9.31.4 Internal interfaces

For more information see "Function ‘scuFlmFLCInit’ Internal Interface" on page 111.

19.9.30.2 Function

Function ‘scuFlmFLArrayPush’ performs the following:

1. If the number of free CADRs to place on the list (call it numCadr) is 0 or greater than 16K then it returns ERROR;
2. Calls function ‘scuFlmFreeListInit’ passing it the number of CADRs to initialize, a pointer to the array CADRs and a single CADR of 0x0;
3. Returns OK.

19.9.30.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuFlmFLArrayPush’ External Interface" on page 109.

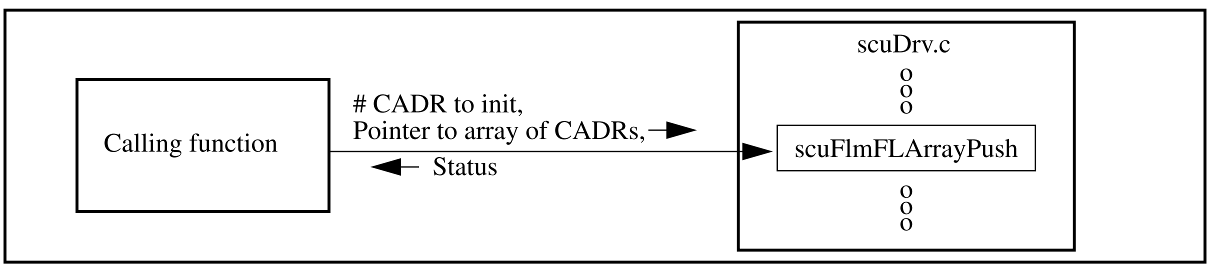


Figure 57 Function ‘scuFlmFLArrayPush’ External Interface

19.9.30.4 Internal interfaces

For more information see "Function ‘scuFlmFLArrayPush’ Internal Interface" on page 109.

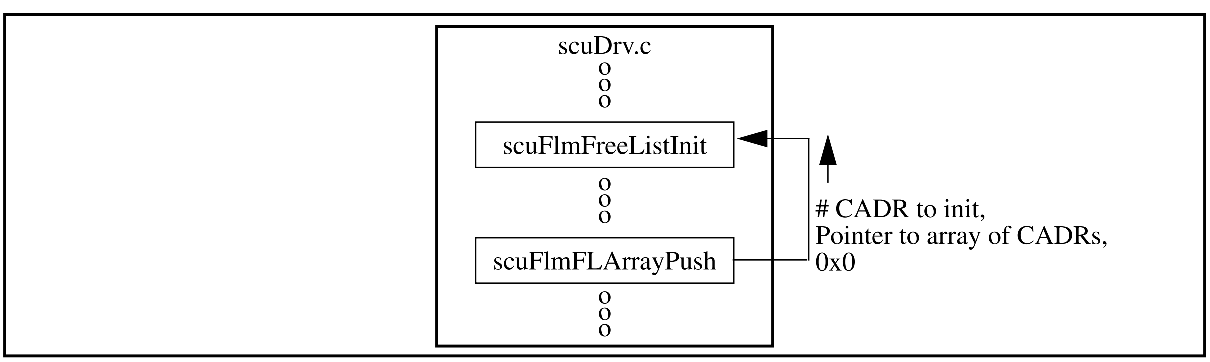


Figure 58 Function ‘scuFlmFLArrayPush’ Internal Interface

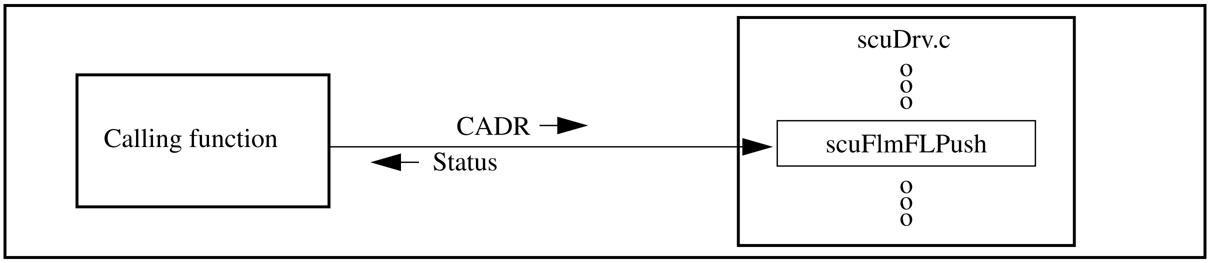


Figure 55 Function ‘scuFlmFLPush’ External Interface

19.9.29.4 Internal interfaces

For more information see "Function ‘scuFlmFLPush’ Internal Interface" on page 108.

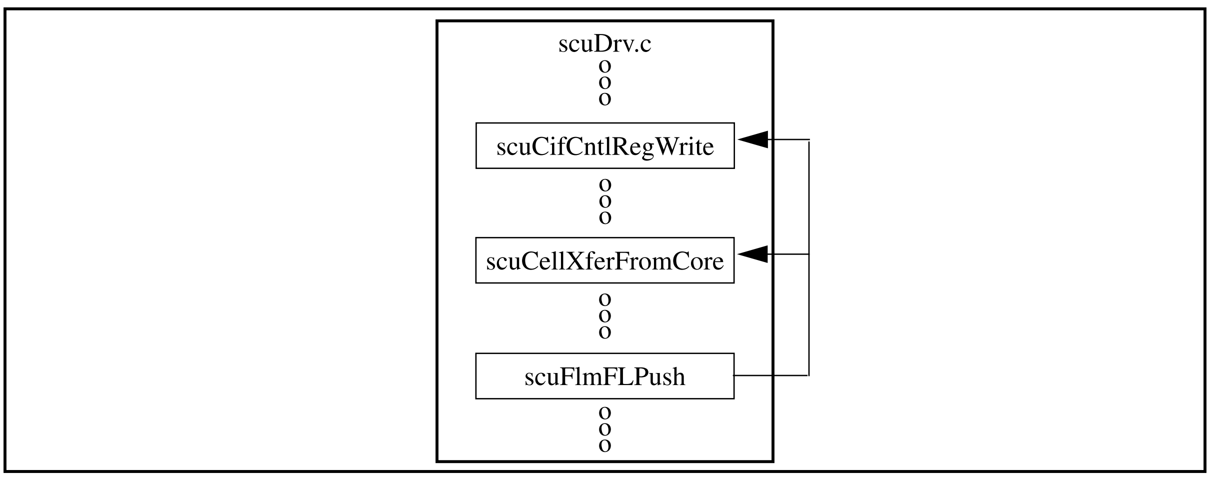


Figure 56 Function ‘scuFlmFLPush’ Internal Interface

19.9.30 Function ‘scuFlmFLArrayPush’

19.9.30.1 Purpose

The purpose of ‘scuFlmFLArrayPush’ (SCU FLM Free CADR List Array Push) is to push a number of CADRs onto the FLM’s Free CADR List. The number of CADRs to push onto the Free CADR List are passed from the calling function and the actual CADRs are passed by the calling function in an array.

19.9.28.4 Internal interfaces

For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

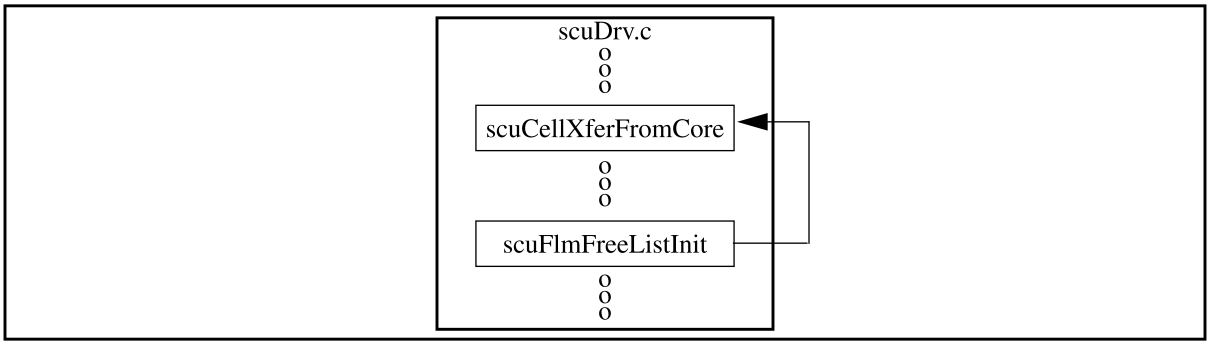


Figure 54 Function ‘scuFlmFreeListInit’ Internal Interface

19.9.29 Function ‘scuFlmFLPush’

19.9.29.1 Purpose

The purpose of ‘scuFlmFLPush’ (SCU FLM Free CADR List Push) is to push a single CADR (passed by the calling function) onto the FLM’s Free CADR List.

19.9.29.2 Function

Function ‘scuFlmFLPush’ performs the following:

1. If the CADR passed by the calling function is greater than 16K then it returns ERROR;
2. Performs a cell read cycle by calling function ‘scuCellXferFromCore’ with the R_CADR equal to the CADR passed by the calling function, the FLM Initialization bit in the CIF control register set and the R_VALID signal unasserted;
3. Clears the FLM Initialization bit in the CIF control register using function ‘scuCifCntlReg-Write’;
4. Returns OK.

19.9.29.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuFlmFLPush’ External Interface" on page 108.

19.9.28 Function ‘scuFlmFreeListInit’

19.9.28.1 Purpose

The purpose of ‘scuFlmFreeListInit’ (SCU FLM Free CADR List Initialize) is to initialize the FLM’s Free CADR List with a number of CADR’s according to a variable indicating the number of CADRs to place on the list, a single CADR and a pointer to an array of CADRs, all of which are passed by the calling function.

19.9.28.2 Function

Function ‘scuFlmFreeListInit’ performs the following:

1. If the number of free CADRs to place on the list (call it numCadr) is 0 or greater than 16K then it returns ERROR;
2. If numCadr is 1 then this function performs a cell read cycle by calling function ‘scuCellXferFromCore’ with the R_CADR equal to the CADR passed by the calling function, the FLM Initialization bit in the CIF control register set and the R_VALID signal unasserted;
3. If numCadr is between 2 and 16K then this function repeats step 2 numCadr times with R_CADR equal to the those in the array of CADRs passed from the calling function;
4. If numCadr is 16K then this function executes a loop indexing a counter from 0 to 16K-1 and repeating step 2 with a R_CADR equal to the loop index;
5. Returns OK.

19.9.28.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuFlmFreeListInit’ External Interface" on page 106.

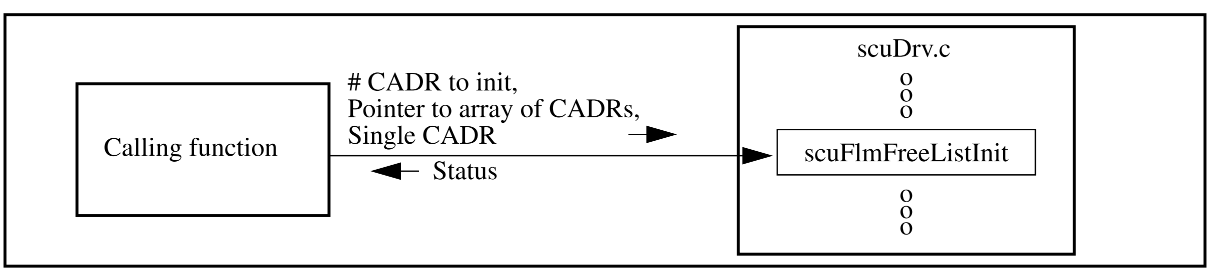


Figure 53 Function ‘scuFlmFreeListInit’ External Interface

19.9.26.4 Internal interfaces

Not applicable.

19.9.27 Function ‘scuFlmCntlRegRead’

19.9.27.1 Purpose

The purpose of function ‘scuFlmCntlRegRead’ (SCU FLM Control Register Read) is to read the FLM control register and return the value read to the calling function.

19.9.27.2 Function

This function reads the value of the FLM’s 16-bit control register and returns it to the calling function.

19.9.27.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuFlmCntlRegRead’ External Interface" on page 105.

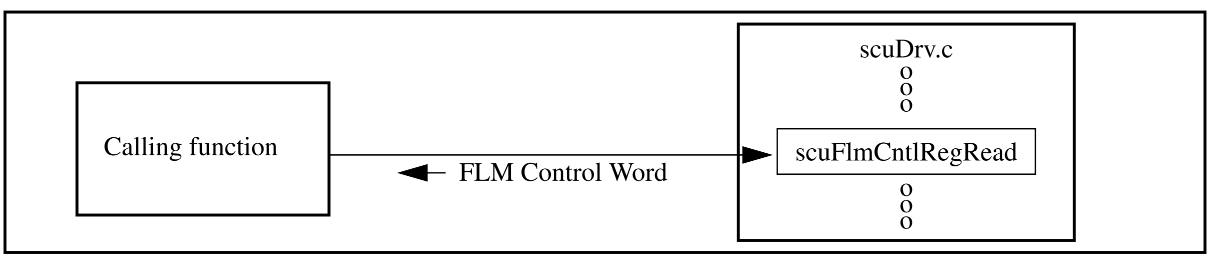


Figure 52 Function ‘scuFlmCntlRegRead’ External Interface

19.9.27.4 Internal interfaces

Not applicable.

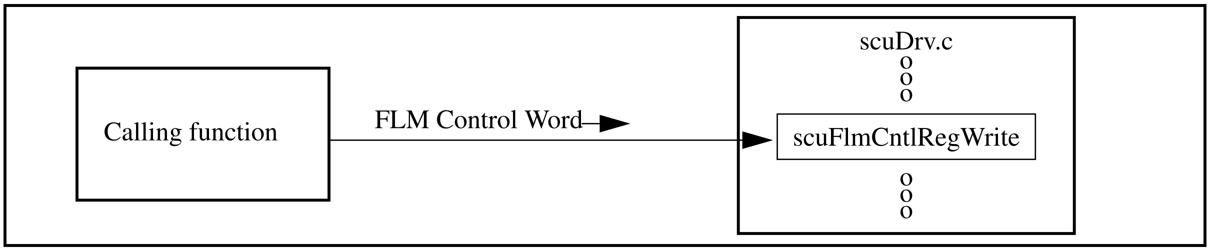


Figure 50 Function ‘scuFlmCntlRegWrite’ External Interface

19.9.25.4 Internal interfaces

Not applicable.

19.9.26 Function ‘scuFlmStatRegRead’

19.9.26.1 Purpose

The purpose of function ‘scuFlmStatRegRead’ (SCU FLM Status Register Read) is to return the value of the FLM’s status register to the calling function.

19.9.26.2 Function

This function performs a read of the FLM’s status register and returns it to the calling function.

19.9.26.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuFlmStatRegRead’ External Interface" on page 104.

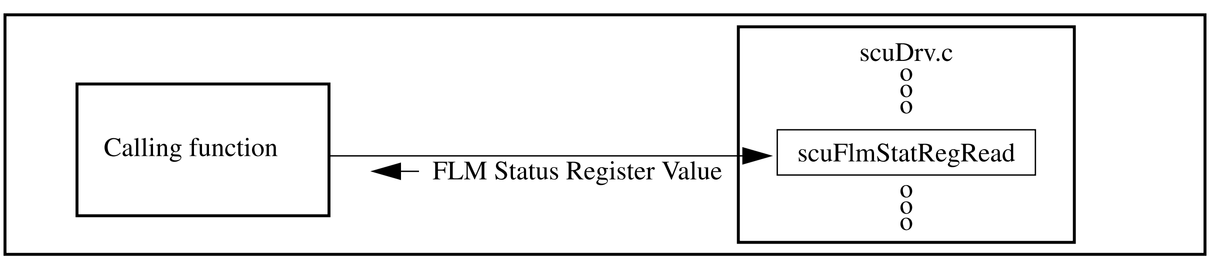


Figure 51 Function ‘scuFlmStatRegRead’ External Interface

19.9.24.2 Function

This function performs a read of the FLM's CMBF Count register and returns it to the calling function.

19.9.24.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function 'scuFlmCmbfCountRead' External Interface" on page 103.

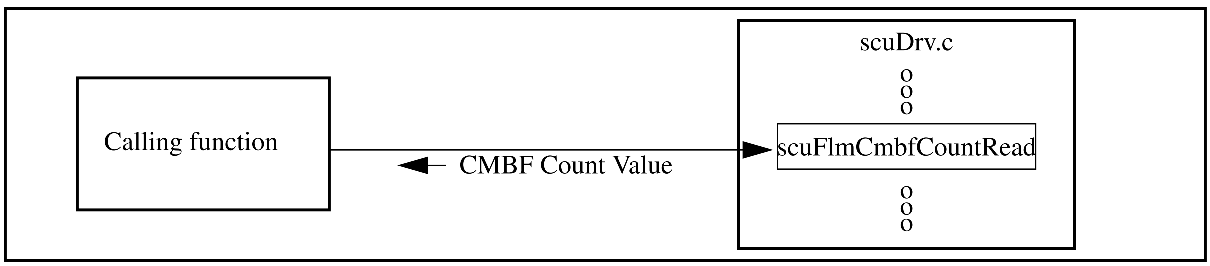


Figure 49 Function 'scuFlmCmbfCountRead' External Interface

19.9.24.4 Internal interfaces

Not applicable.

19.9.25 Function 'scuFlmCntlRegWrite'

19.9.25.1 Purpose

The purpose of function 'scuFlmCntlRegWrite' (SCU FLM Control Register Write) is to write a control word to the FLM control register.

19.9.25.2 Function

This function writes a control word, passed to it by the calling function, to the FLM control register.

19.9.25.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software.

19.9.23 Function ‘scuCifRFifoStatRead’

19.9.23.1 Purpose

The purpose of function ‘scuCifRFifoStatRead’ (SCU CIF R_IVC FIFO Status Read) is to return the value of the R_IVC FIFO’s status bits which are in the range zero to three.

19.9.23.2 Function

Function ‘scuCifRFifoStatRead’ performs the following:

1. Reads the contents of the CIF’s Status register;
2. ANDs the contents of the CIF Status register with the R_IVC FIFO status mask and shifts this until the R_IVC FIFO status bits are the two least significant bits;
3. Returns the R_IVC FIFO status bits.

19.9.23.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software.

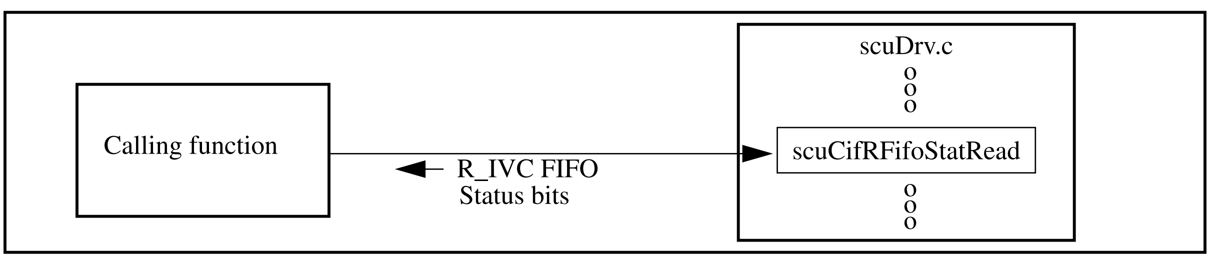


Figure 48 Function ‘scuCifRFifoStatRead’ External Interface

19.9.23.4 Internal interfaces

Not applicable

19.9.24 Function ‘scuFlmCmbfCountRead’

19.9.24.1 Purpose

The purpose of function ‘scuFlmCmbfCountRead’ (SCU FLM CMBF Count Read) is to return the value of the FLM’s (Free List Manager’s) CMBF Count register to the calling function.

19.9.22 Function ‘scuCifLPWFifoStatRead’

19.9.22.1 Purpose

The purpose of function ‘scuCifLPWFifoStatRead’ (SCU CIF Low Priority W_IVC FIFO Status Read) is to return the value of the low priority W_IVC FIFO’s status bits which are in the range zero to three.

19.9.22.2 Function

Function ‘scuCifLPWFifoStatRead’ performs the following:

1. Reads the contents of the CIF’s Status register;
2. ANDs the contents of the CIF Status register with the low priority W_IVC FIFO status mask and shifts this until the high priority W_IVC FIFO status bits are the two least significant bits;
3. Returns the low priority W_IVC FIFO status bits.

19.9.22.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCifRivcFifoRead’ External Interface" on page 99.

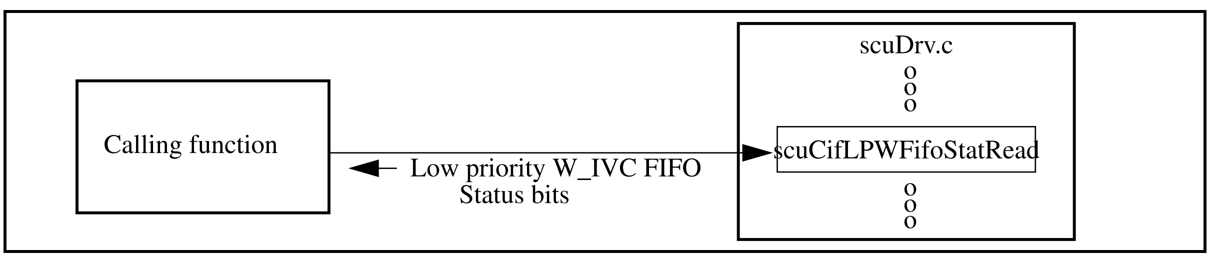


Figure 47 Function ‘scuCifLPWFifoStatRead’ External Interface

19.9.22.4 Internal interfaces

Not applicable

19.9.21 Function ‘scuCifHPWFifoStatRead’

19.9.21.1 Purpose

The purpose of function ‘scuCifHPWFifoStatRead’ (SCU CIF High Priority W_IVC FIFO Status Read) is to return the value of the high priority W_IVC FIFO’s status bits which are in the range zero to three.

19.9.21.2 Function

Function ‘scuCifHPWFifoStatRead’ performs the following:

1. Reads the contents of the CIF’s Status register;
2. ANDs the contents of the CIF Status register with the high priority W_IVC FIFO status mask and shifts this until the high priority W_IVC FIFO status bits are the two least significant bits;
3. Returns the high priority W_IVC FIFO status bits.

19.9.21.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCifRivcFifoRead’ External Interface" on page 99.

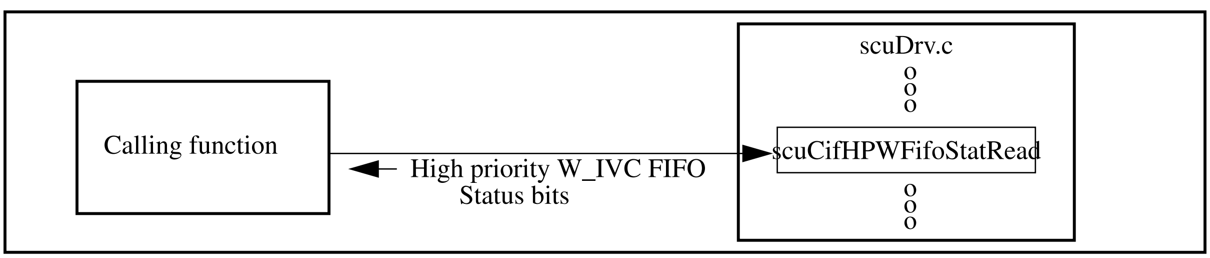


Figure 46 Function ‘scuCifHPWFifoStatRead’ External Interface

19.9.21.4 Internal interfaces

Not applicable

19.9.19.4 Internal interfaces

Not applicable.

19.9.20 Function ‘scuCifRivcFifoRead’

19.9.20.1 Purpose

The purpose of function ‘scuCifRivcFifoRead’ (SCU CIF R_IVC FIFO Read) is to perform a read of the data at the head of the CIF’s R_IVC FIFO and return the IVC found there to a pointer passed from the calling function.

19.9.20.2 Function

Function ‘scuCifRivcFifoRead’ performs the following:

1. Reads the R_IVC FIFO status bits and returns ERROR if they indicate that the FIFO is empty.A message is also sent to the switch User Interface indicating that a R_IVC FIFO read was attempted when the FIFO was empty.;
2. Performs a read of the R_IVC FIFO register;
3. Places the IVC read above in the location pointed to by the calling functions IVC pointer;
4. Returns OK.

19.9.20.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCifRivcFifoRead’ External Interface" on page 99.

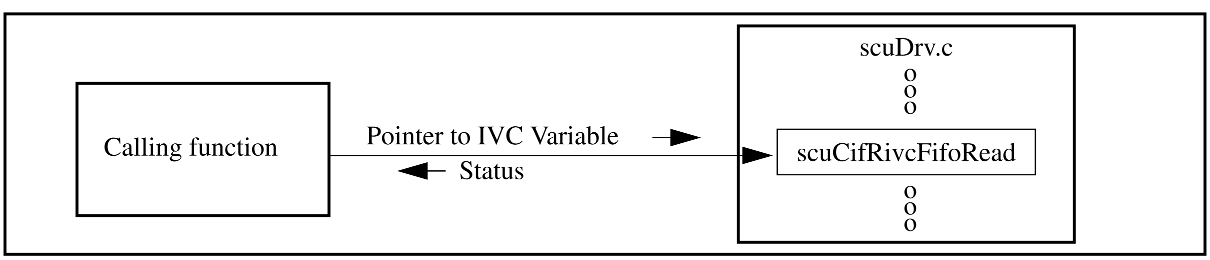


Figure 45 Function ‘scuCifRivcFifoRead’ External Interface

19.9.20.4 Internal interfaces

For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

19.9.19 Function 'scuCifWivcFifoRead'

19.9.19.1 Purpose

The purpose of function 'scuCifWivcFifoRead' (SCU CIF W_IVC FIFO Read) is to perform a read of the data at the head of the CIF's high or low priority W_IVC Data FIFO and return the IVC and CADR found there to the pointers passed from the calling function. A flag passed from the calling function is used to select which W_IVC FIFO should be read.

19.9.19.2 Function

Function 'scuCifWivcFifoRead' performs the following:

1. If the flag indicating which W_IVC data FIFO to read is set to indicate high priority then it reads the high priority W_IVC data FIFO status bits and returns ERROR if they indicate that the FIFO is empty. Otherwise it reads the low priority W_IVC data FIFO status bits and returns ERROR if they indicate that the FIFO is empty; If reading either FIFOs status bits shows that the FIFO is empty then a message is also sent to the switch User Interface indicating that a W_IVC FIFO read was attempted when the FIFO was empty.
2. If the flag indicating which W_IVC data FIFO to read is set to high priority then it performs a read of the high priority W_IVC data FIFO register, otherwise it performs a read of the low priority W_IVC data FIFO register;
3. Places the IVC read above in the location pointed to by the calling functions IVC pointer;
4. Places the CADR read above in the location pointed to by the calling function CADR pointer;
5. Returns OK.

19.9.19.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function 'scuCifWivcFifoRead' External Interface" on page 98.

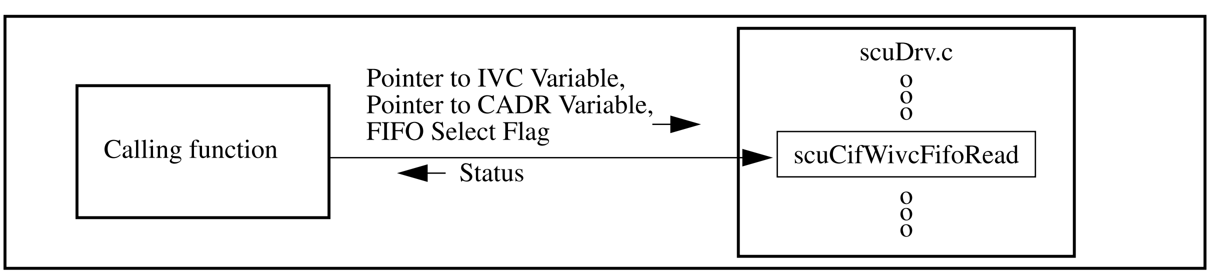


Figure 44 Function 'scuCifWivcFifoRead' External Interface

19.9.18.2 Function

Function ‘scuRivcLutInit’ performs the following:

1. Calls function ‘scuRivcLutSetup’ passing it the pointer to the array of IVC data structures, the number of IVCs to initialize, a R_CSID of 0 and a R_IVC FIFO enable bit set to 0;
2. Returns the value returned from ‘scuRivcLutSetup’.

19.9.18.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuRivcLutInit’ External Interface" on page 97.

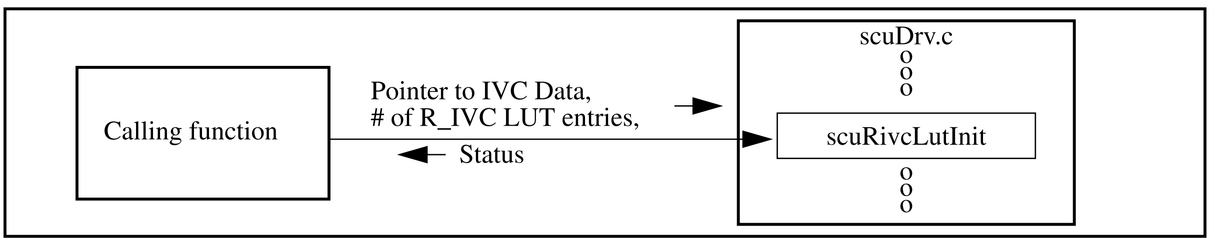


Figure 42 Function ‘scuRivcLutInit’ External Interface

19.9.18.4 Internal interfaces

For more information see "Function ‘scuRivcLutInit’ Internal Interface" on page 97.

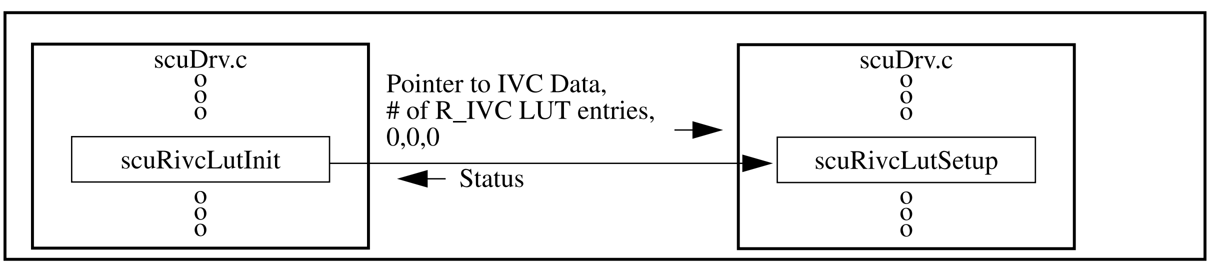


Figure 43 Function ‘scuRivcLutInit’ Internal Interface

19.9.17.2 Function

Function ‘scuRivcLutSet’ performs the following:

1. If R_IVC FIFO enable flag is not equal to 1 or 0 then it returns ERROR;
2. If the R_CSID is greater than 15 it returns ERROR;
3. IF the IVC is greater than 8K it returns ERROR;
4. ORs the R_CSID and the R_IVC FIFO enable flag to produce a R_IVC lookup table entry;
5. Writes the R_IVC lookup table entry value into the R_IVC lookup table at the location indexed by the IVC passed by the calling function for each IVC to be initialized;
6. Returns OK.

19.9.17.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuRivcLutSet’ External Interface" on page 96.

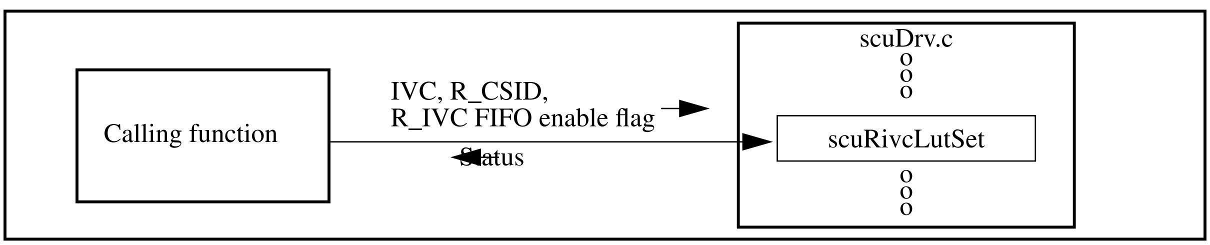


Figure 41 Function ‘scuRivcLutSet’ External Interface

19.9.17.4 Internal interfaces

Not applicable

19.9.18 Function ‘scuRivcLutInit’

19.9.18.1 Purpose

The purpose of function ‘scuRivcLutInit’ (SCU R_IVC Lookup Table Initialization) is to setup R_IVC lookup able entries a list of IVCs. Calling functions pass it a pointer to an array of IVC data structures and the number of R_IVC lookup table entries to initialize.

3. If the number of R_IVC lookup table entries to initialize is one then it performs the following:
    - Checks the R_CSID and returns ERROR if it is greater than or equal to 15;
    - Checks the IVC and returns ERROR if it is greater than or equal to 8K;
    - ORs the R_CSID and R_IVC FIFO enable flag to produce a R_IVC lookup table entry;
    - Writes the R_IVC lookup table entry value into the R_IVC lookup table at the location indexed by the IVC passed by the calling function;
4. If the number of R_IVC lookup table entries to initialize is two or more, then this function performs the above step extracting IVCs and W_IVC lookup table data from the array of IVC data structures passed from the calling function for each IVC to be initialized;
5. Returns OK.

19.9.16.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuRivcLutSetup’ External Interface" on page 95.

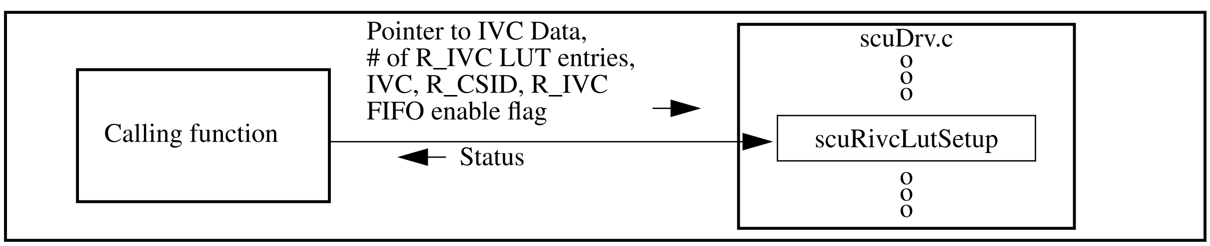


Figure 40 Function ‘scuRivcLutSetup’ External Interface

19.9.16.4 Internal interfaces

For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

19.9.17 Function ‘scuRivcLutSet’

19.9.17.1 Purpose

The purpose of function ‘scuRivcLutSet’ (SCU R_IVC Lookup Table Set) is to set a R_IVC lookup table able entries for one IVC. Calling functions pass it an IVC, a R_CSID and R_IVC FIFO enable flag.

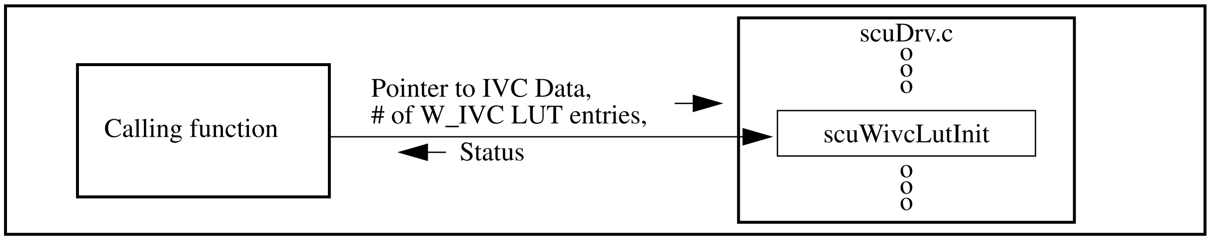


Figure 38 Function ‘scuWivcLutInit’ External Interface

19.9.15.4 Internal interfaces

For more information see "Function ‘scuWivcLutInit’ Internal Interface" on page 94.

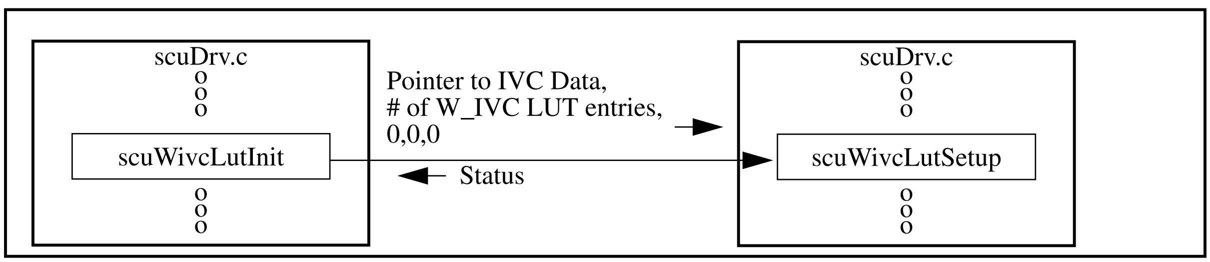


Figure 39 Function ‘scuWivcLutInit’ Internal Interface

19.9.16 Function ‘scuRivcLutSetup’

19.9.16.1 Purpose

The purpose of function ‘scuRivcLutSetup’ (SCU R_IVC Lookup Table Setup) is to setup R_IVC lookup table entries for one IVC or a list of IVCs. Calling functions pass it a pointer to an array of IVC data structures, the number of R_IVC lookup table entries to initialize, an IVC and a R_IVC lookup table entry value.

19.9.16.2 Function

Function ‘scuRivcLutSetup’ performs the following:

1. If the number of R_IVC lookup table entries to initialize is zero or greater than 8K then it returns ERROR;
2. If the R_IVC FIFO enable flag is not equal to 1 or 0 then it returns ERROR;

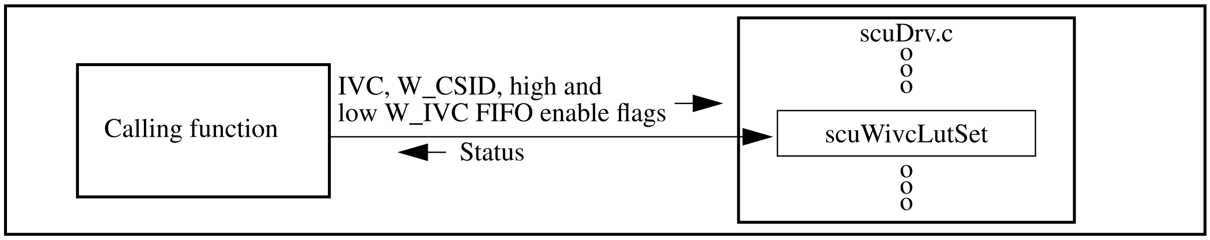


Figure 37 Function ‘scuWivcLutSet’ External Interface

19.9.14.4 Internal interfaces

Not applicable

19.9.15 Function ‘scuWivcLutInit’

19.9.15.1 Purpose

The purpose of function ‘scuWivcLutInit’ (SCU W_IVC Lookup Table Initialization) is to setup W_IVC lookup able entries a list of IVCs. Calling functions pass it a pointer to an array of IVC data structures and the number of W_IVC lookup table entries to initialize.

19.9.15.2 Function

Function ‘scuWivcLutInit’ performs the following:

1. Calls function ‘scuWivcLutSetup’ passing it the pointer to the array of IVC data structures, the number of IVCs to initialize, a W_CSID of 0, a low priority W_IVC FIFO enable bit set to 0 and a high priority W_IVC FIFO enable bit set to 0;
2. Returns the value returned from ‘scuWivcLutSetup’.

19.9.15.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuWivcLutInit’ External Interface" on page 94.

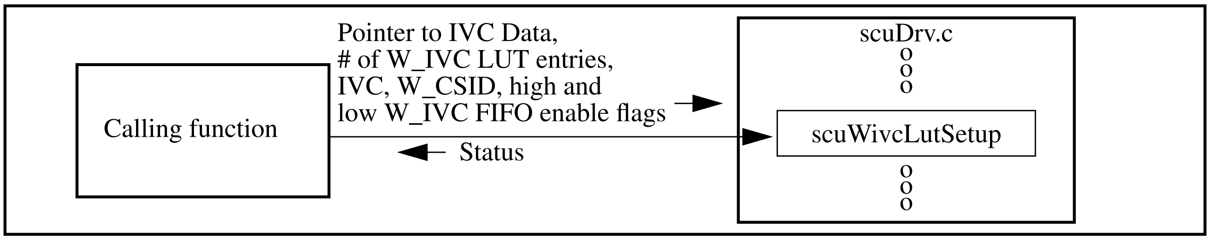


Figure 36 Function ‘scuWivcLutSetup’ External Interface

19.9.13.4 Internal interfaces

For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

19.9.14 Function ‘scuWivcLutSet’

19.9.14.1 Purpose

The purpose of function ‘scuWivcLutSet’ (SCU W_IVC Lookup Table Set) is to set a W_IVC lookup table able entries for one IVC. Calling functions pass it an IVC, a W_CSID, a high priority W_IVC FIFO enable flag and a low priority W_IVC FIFO enable flag.

19.9.14.2 Function

Function ‘scuWivcLutSet’ performs the following:

1. If either the high priority W_IVC FIFO enable flag or a low priority W_IVC FIFO enable flags are not equal to 1 or 0 then it returns ERROR;
2. If the W_CSID is greater than 15 it returns ERROR;
3. IF the IVC is greater than 8K it returns ERROR;
4. ORs the W_CSID, high priority W_IVC FIFO enable flag and a low priority W_IVC FIFO enable flag to produce a W_IVC lookup table entry;
5. Writes the W_IVC lookup table entry value into the W_IVC lookup table at the location indexed by the IVC passed by the calling function for each IVC to be initialized;
6. Returns OK.

19.9.14.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuWivcLutSet’ External Interface" on page 93.

19.9.13 Function ‘scuWivcLutSetup’

19.9.13.1 Purpose

The purpose of function ‘scuWivcLutSetup’ (SCU W_IVC Lookup Table Setup) is to setup W_IVC lookup able entries for one IVC or a list of IVCs. Calling functions pass it a pointer to an array of IVC data structures, the number of W_IVC lookup table entries to initialize, an IVC, a W_CSID, a high priority W_IVC FIFO enable flag and a low priority W_IVC FIFO enable flag.

19.9.13.2 Function

Function ‘scuWivcLutSetup’ performs the following:

1. If the number of W_IVC lookup table entries to initialize is zero or greater than 8K then it returns ERROR;
2. If either the high priority W_IVC FIFO enable flag or a low priority W_IVC FIFO enable flag are not equal to 1 or 0 then it returns ERROR;
3. If the number of W_IVC lookup table entries to initialize is one then performs the following:
    - Checks the W_CSID and returns ERROR if it is greater than or equal to 15;
    - Checks the IVC and returns ERROR if it is greater than or equal to 8K;
    - ORs the W_CSID, high priority W_IVC FIFO enable flag and a low priority W_IVC FIFO enable flag to produce a W_IVC lookup table entry;
    - Writes the W_IVC lookup table entry value into the W_IVC lookup table at the location indexed by the IVC passed by the calling function for each IVC to be initialized;
4. If the number of W_IVC lookup table entries to initialize is two or more, then this function performs the above step extracting IVCs and W_IVC lookup table data from the array of IVC data structures passed from the calling function;
5. Returns OK.

19.9.13.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuWivcLutSetup’ External Interface" on page 92.

19.9.12 Function ‘scuCifRdWrPoll’

19.9.12.1 Purpose

The purpose of function ‘scuCifRdWrPoll’ (SCU CIF Read/Write Poll) poll the CIF Control Register until the Read One Cell and Write One Cell bits are cleared indicating that the CIF is no longer busy executing a cell write cycle.

19.9.12.2 Function

Function ‘scuCifRdWrPoll’ performs the following:

1. Reads the CIF Control Register until the Read One Cell and Write One Cell bits are cleared or the CIF Control Register has been read 10 times;
2. If the CIF Control Register had been read 10 times then this function returns ERROR, otherwise it returns OK;

19.9.12.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCifRdWrPoll’ External Interface" on page 90.

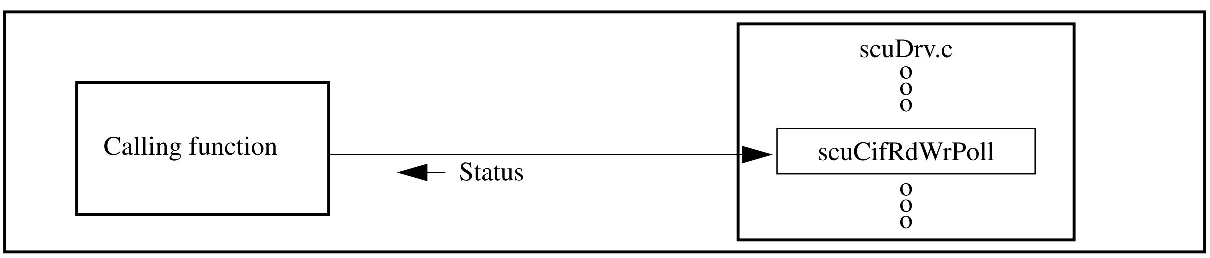


Figure 35 Function ‘scuCifRdWrPoll’ External Interface

19.9.12.4 Internal interfaces

Not applicable.

19.9.11 Function ‘scuCifRDPoll’

19.9.11.1 Purpose

The purpose of function ‘scuCifRdPoll’ (SCU CIF Read Poll) poll the CIF Control Register until the Read One Cell Bit is cleared indicating that the CIF is no longer busy executing a cell read cycle.

19.9.11.2 Function

Function ‘scuCifRdPoll’ performs the following:

1. Reads the CIF Control Register until the Read One Cell bit is cleared or the CIF Control Register has been read 10 times;
2. If the CIF Control Register had been read 10 times then this function returns ERROR, otherwise it returns OK;

19.9.11.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCifRdPoll’ External Interface" on page 89.

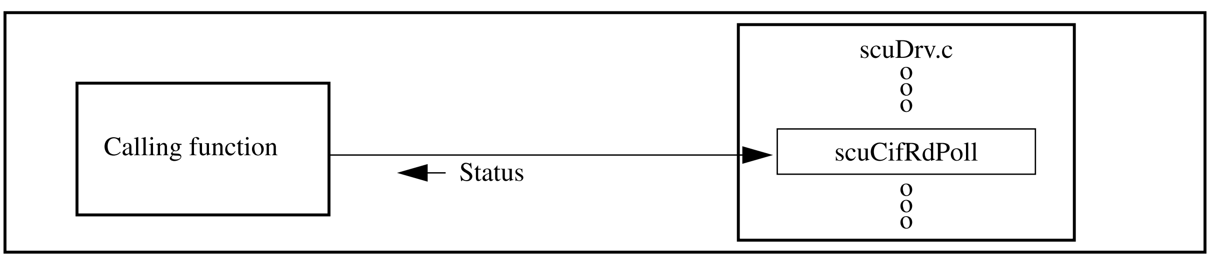


Figure 34 Function ‘scuCifRdPoll’ External Interface

19.9.11.4 Internal interfaces

Not applicable.

19.9.10 Function ‘scuCifWrPoll’

19.9.10.1 Purpose

The purpose of function ‘scuCifWrPoll’ (SCU CIF Write Poll) poll the CIF Control Register until the Write One Cell Bit is cleared indicating that the CIF is no longer busy executing a cell write cycle.

19.9.10.2 Function

Function ‘scuCifWrPoll’ performs the following:

1. Reads the CIF Control Register until the Write One Cell bit is cleared or the CIF Control Register has been read 10 times;
2. If the CIF Control Register had been read 10 times then this function returns ERROR, otherwise it returns OK;

19.9.10.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCifWrPoll’ External Interface" on page 88.

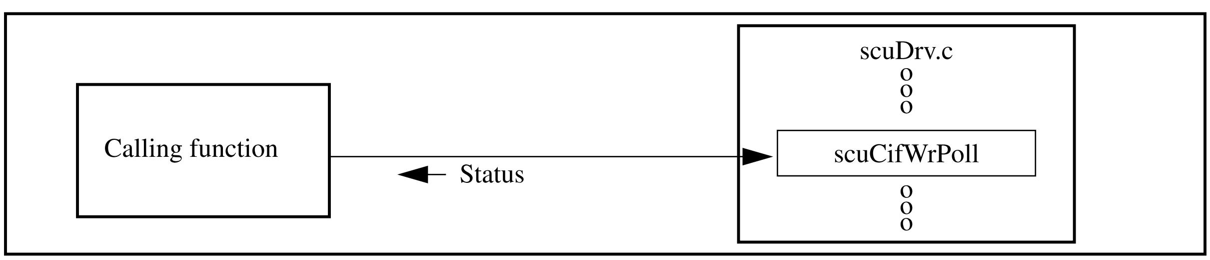


Figure 33 Function ‘scuCifWrPoll’ External Interface

19.9.10.4 Internal interfaces

Not applicable.

19.9.8.4 Internal interfaces

This function is called by functions ‘scuWFifoHPAlFull ISR’, ‘scuWFifoLPAlFull ISR’ and ‘scuRFifoAlFullISR’. For more information, please see the ‘Internal Interface’ sections of these functions.

19.9.9 Function ‘scuCifCntlRegRead’

19.9.9.1 Purpose

The purpose of function ‘scuCifCntlRegRead’ (SCU CIF Control Register Read) is to read the CIF control register and return the value read to the calling function.

19.9.9.2 Function

This function reads the value of the CIF’s 16-bit control register and returns it to the calling function.

19.9.9.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCifCntlRegRead’ External Interface" on page 87.

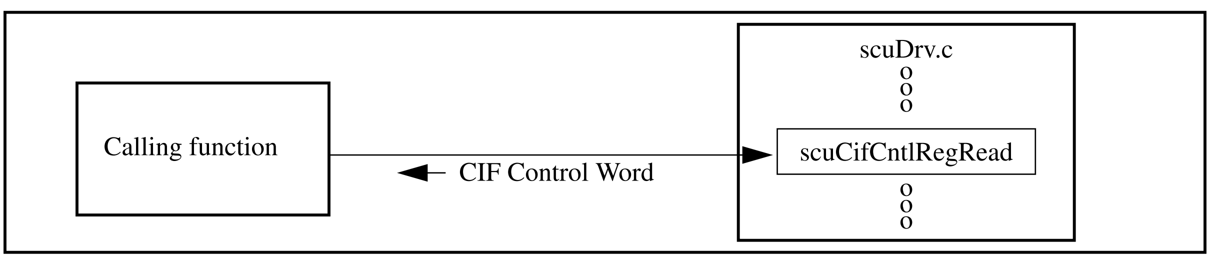


Figure 32 Function ‘scuCifCntlRegRead’ External Interface

19.9.9.4 Internal interfaces

Not applicable.

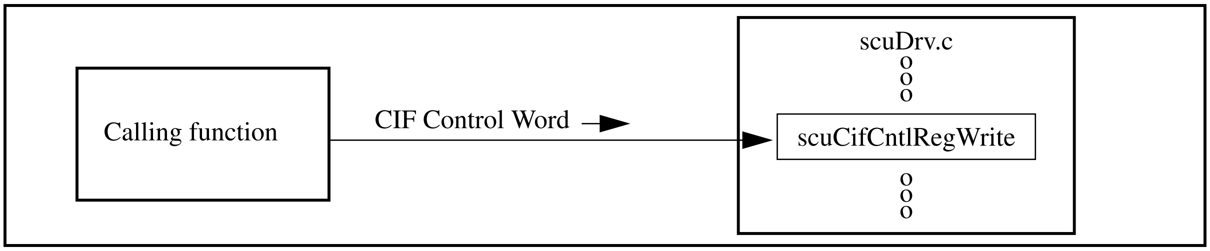


Figure 30 Function 'scuCifCntlRegWrite' External Interface

19.9.7.4 Internal interfaces

Not applicable.

19.9.8 Function 'scuCifStatRegRead'

19.9.8.1 Purpose

The purpose of function 'scuCifStatRegRead' (SCU CIF Status Register Read) is to read the CIF status register and return the value read to the calling function.

19.9.8.2 Function

This function reads the value of the CIF's 16-bit status register and returns it to the calling function.

19.9.8.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function 'scuCifStatRegRead' External Interface" on page 86.

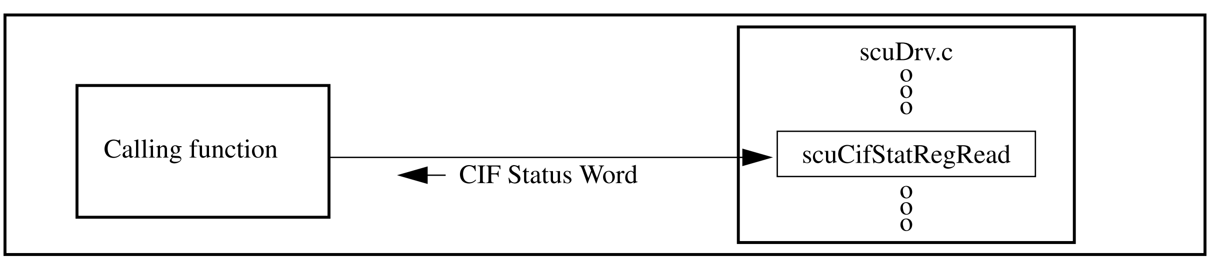


Figure 31 Function 'scuCifStatRegRead' External Interface

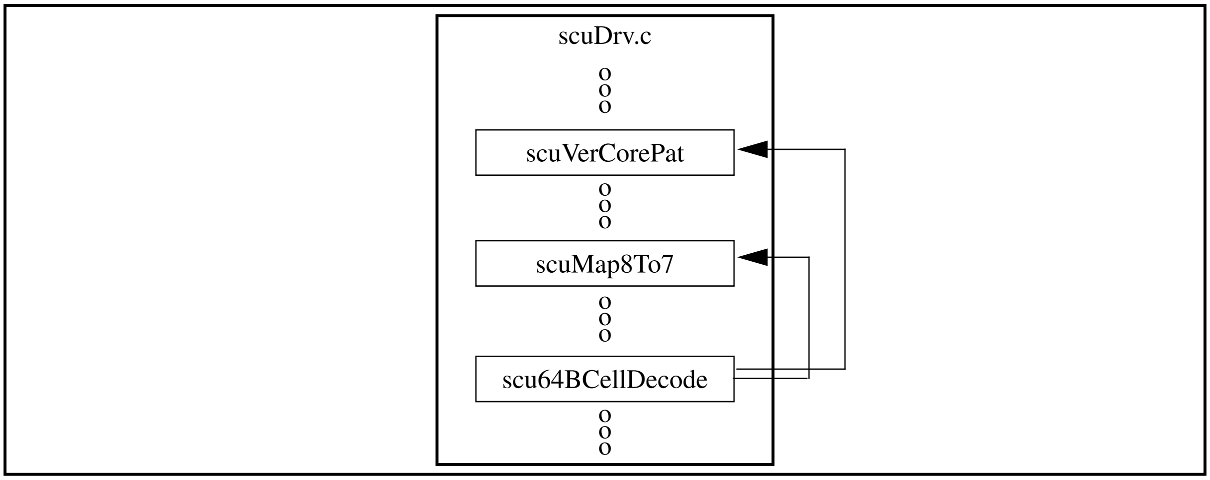


Figure 29 Function ‘scu64BCellDecode’ Internal Interface

19.9.7 Function ‘scuCifCntlRegWrite’

19.9.7.1 Purpose

The purpose of function ‘scuCifCntlRegWrite’ (SCU Write CIF Control Register) is to write a control word to the CIF control register.

19.9.7.2 Function

This function writes a control word passed to it by the calling function to the CIF control register.

19.9.7.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCifCntlRegWrite’ External Interface" on page 86.

2. Converts the 64-byte encoded cell into 64 bytes containing 7-bit patterns using function 'scuMap8To7.' This function returns ERROR if 'scuMap8To7' indicates that an invalid Core Memory byte pattern has been encountered;
3. Converts the 64 7-bit patterns into 56 bytes;
4. Extracts the cell header, IVC and payload from the 56 bytes above and places them at the locations specified by the pointers to the cell data structure provided by the calling function;
5. Returns OK.

19.9.6.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function 'scu64BCellDecode' External Interface" on page 84.

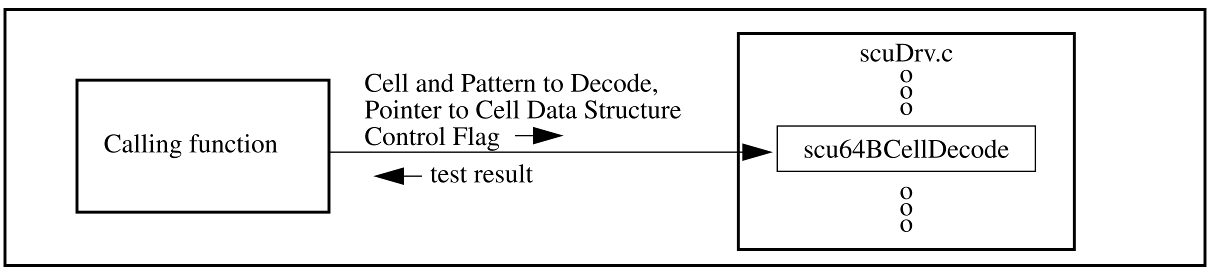


Figure 28 Function 'scu64BCellDecode' External Interface

19.9.6.4 Internal interfaces

For more information see "Function 'scmRegTest' Internal Interface" on page 78.

19.9.5.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuMap8To7’ External Interface" on page 83.

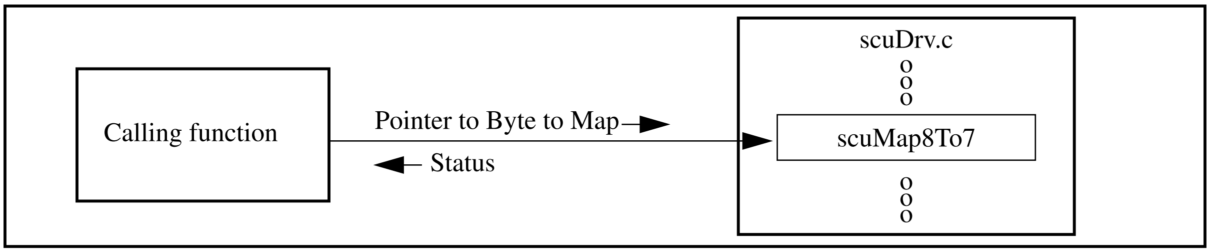


Figure 27 Function ‘scuMap8To7’ External Interface

19.9.5.4 Internal interfaces

For more information see "Function ‘scu64BCellDecode’ External Interface" on page 84.

19.9.6 Function ‘scu64BCellDecode’

19.9.6.1 Purpose

Function ‘scu64BCellDecode’ (SCU 64-Byte Cell Decode) decodes a 64-byte encoded cell of 32 16-bit words (as it would appear in the Core Memory) and produces a 56-byte cell consisting of a cell header, cell IVC and cell payload. Calling functions pass ‘scu64BCellDecode’ a pointer to an array of 32 16-bit words of the cell to be decoded, a pattern which is assumed to be repeated throughout the encoded cell and a control flag indicating whether the function should use the pointer to the encoded cell or generate an encoded cell from the pattern. Calling functions also pass ‘scu64BCellDecode’ a pointer to the cell data structure at which this routine should place the results.

19.9.6.2 Function

Function ‘scu64BCellDecode’ performs the following:

1. If the control flag indicates that this function should generate the cell to be decoded from the pattern passed by the calling function then this function:
    - Calls function ‘scuVerCorePat’ to verify that the pattern contains bytes which are valid Core Memory bytes;
    - Generates a 64-byte encoded cell from the pattern passed by the calling function;

19.9.4.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function 'scuVerCorePat' External Interface" on page 82.

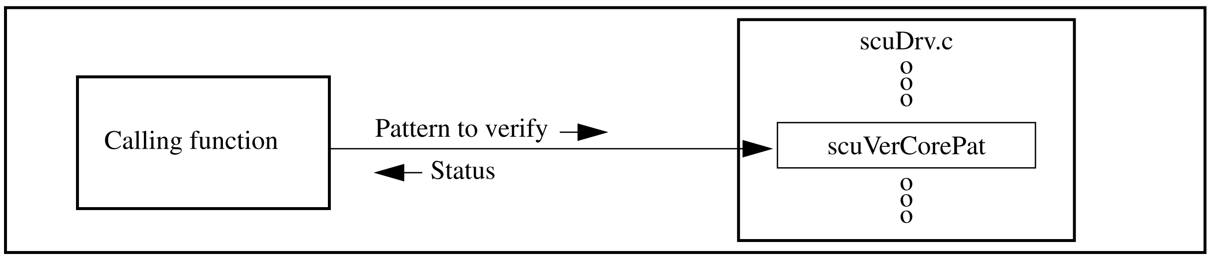


Figure 26 Function 'scuVerCorePat' External Interface

19.9.4.4 Internal interfaces

For more information see "Function 'scu64BCellDecode' External Interface" on page 84.

19.9.5 Function 'scuMap8To7'

19.9.5.1 Purpose

Function 'scuMap8To7' (SCU Map 8 To 7 maps an 8-bit byte from the Core Memory into a 7-bit word as a part of the cell decoding process.

19.9.5.2 Function

'scuMap8To7' performs the following:

1. Increments an index from 0 until the byte to be decoded is found in the cell encoding table at the location indicated by the index;
2. If an entry in the table equals the byte to be mapped then the byte is replaced by the value of the index;
3. Returns ERROR if the above index reaches 128
4. Returns OK.

19.9.3.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCellCpySbToRam’ External Interface" on page 81.

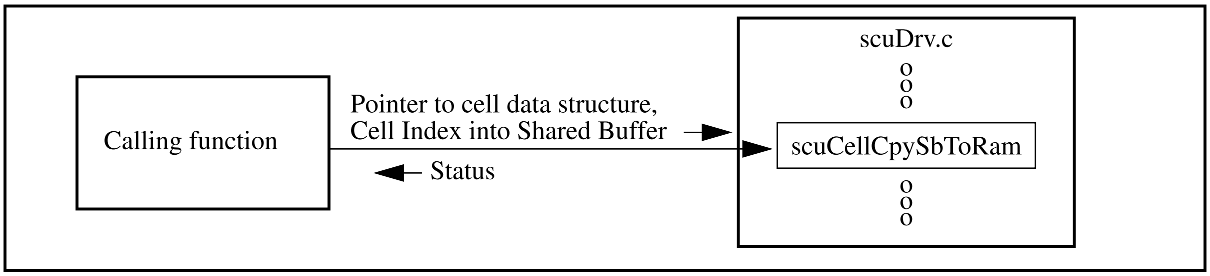


Figure 25 Function ‘scuCellCpySbToRam’ External Interface

19.9.3.4 Internal interfaces

Not applicable.

<u>19.9.4 Function ‘scuVerCorePat’</u>

19.9.4.1 Purpose

Function ‘scuVerCorePat’ (SCU Verify Core Pattern) verifies whether or not a 16-bit pattern contains bytes which are valid Core Memory bytes.

19.9.4.2 Function

‘scuVerCorePat’ performs the following:

1. Searches a cell encoding lookup table for the occurrence of the LSB of the pattern and returns ERROR if it is not found;
2. Searches a cell encoding lookup table for the occurrence of the MSB of the pattern and returns ERROR if it is not found;
3. Returns OK.

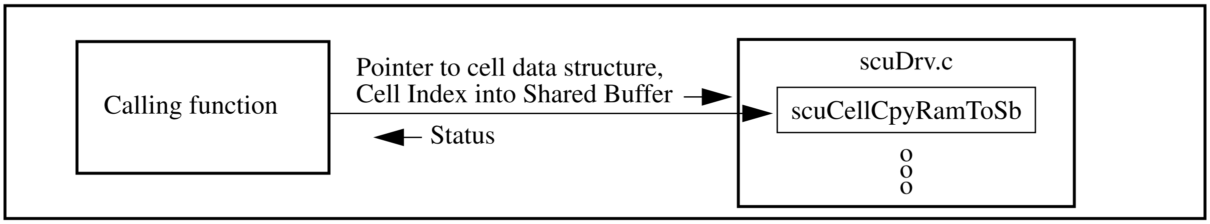


Figure 24 Function ‘scuCellCpyRamToSb’ External Interface

19.9.2.4 Internal interfaces

Not applicable.

19.9.3 Function ‘scuCellCpySbToRam’

19.9.3.1 Purpose

Function ‘scuCellCpySbToRam’ (SCU Cell Copy from CIF Shared Buffer to RAM) copies 55 bytes of a cell (including header, IVC and payload) from a specific cell location in the CIF’s Shared Buffer memory to GPU RAM. Calling functions must pass it a cell index into the CIF’s Shared Buffer indicating which cell to retrieve and a pointer to the cell data structure where it can place the cell.

19.9.3.2 Function

‘scuCellCpySbToRam’ performs the following:

1. Copies the cell’s IVC from bytes two and three of the cell location in the Shared Buffer into the cell data structure’s IVC;
2. Copies the cell’s VPI, VCI, and PLT from bytes four through seven of the cell location in the Shared Buffer to the first four bytes of the header in the cell data structure;
3. Copies the cell payload from bytes 8-55 of the cell location in the Shared Buffer to the payload of the cell data structure;
4. Copies the cell’s HEC from byte 56 of the cell location in the Shared Buffer to the fifth byte of the header in the cell data structure;
5. Returns OK.

19.9 Module ‘scuDrv.c’

19.9.1 Purpose

The purpose of module ‘scuDrv.c’ is to provide a set of low level driver functions to manipulate and control the various blocks of the SCU. These driver functions provide simple one-line function calls that can be called by other software in order to perform various actions with the SCU hardware.

19.9.2 Function ‘scuCellCpyRamToSb’

19.9.2.1 Purpose

Function ‘scuCellCpyRamToSb’ (SCU Cell Copy from RAM to CIF Shared Buffer) copies 55 bytes of a cell (including header, IVC and payload) from GPU RAM to a specific cell location in the CIF’s (Cell Interface’s) Shared Buffer memory. Calling functions must pass it a cell index into the CIF’s Shared Buffer indicating where to store the cell and a pointer to the cell data structure from which the cell can be copied.

19.9.2.2 Function

‘scuCellCpyRamToSb’ performs the following:

1. Copies the cell’s IVC to bytes two and three of the cell location in the Shared Buffer;
2. Copies the cell’s VPI, VCI, and PLT (first four bytes of the header) into bytes four through seven of the cell location in the Shared Buffer;
3. Copies the cell payload to bytes 8-55 of the cell location in the Shared Buffer;
4. Copies the cell’s HEC (byte five of the header) to byte 56 of the cell location in the Shared Buffer;
5. Initializes byte 56 of the cell location in the Shared Buffer to 0. This is where the cells CRC8 will be appended by the CIF and it must be initialized to 0;
6. Returns OK.

19.9.2.3 External interfaces

This function is called by many other functions including those used in SCU Diagnostics, SCU initialization and application software. For more information see "Function ‘scuCellCpyRamToSb’ External Interface" on page 80.[2]

2. In the following sections, ‘External Interface’ figures serve to illustrate the parameters passed to and from functions while ‘Internal Interface’ figures illustrate the calls made between functions in module ‘scuDrv.c’.

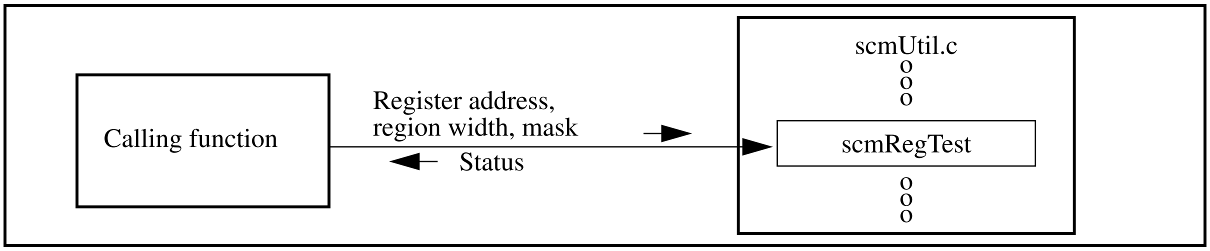


Figure 22 Function ‘scmRegTest’ External Interface

19.8.7.4 Internal interfaces

For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

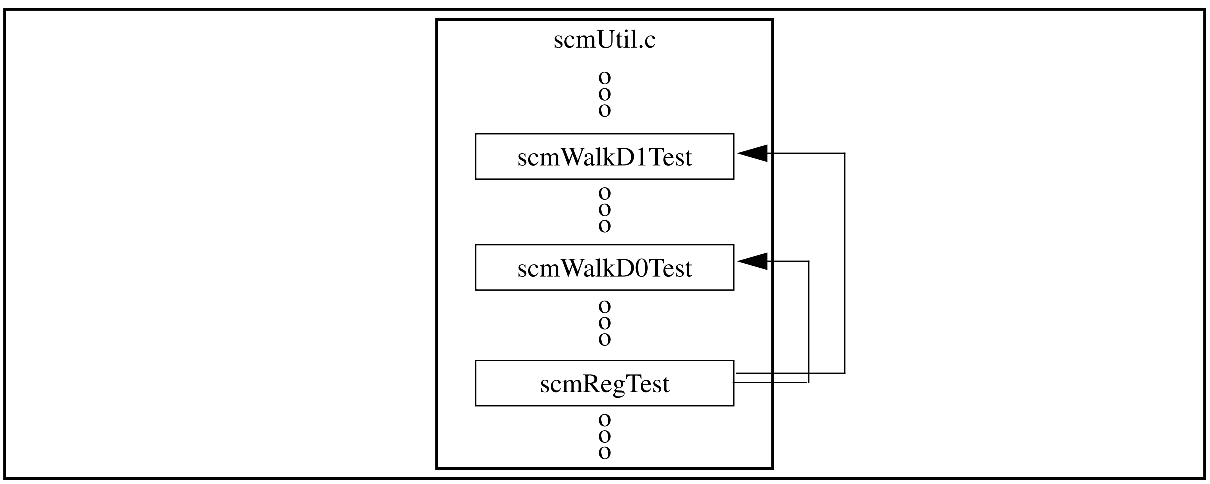


Figure 23 Function ‘scmRegTest’ Internal Interface

19.8.7 Function ‘scmRegTest’

19.8.7.1 Purpose

The purpose of function ‘scmRegTest’ (SCU Register Test) to perform a universal register test on a register indicator by a pointer passed from the calling function. Calling functions also pass a mask indicating which bits in the register under test are valid and the width of the register’s memory region.

19.8.7.2 Function

Function ‘scmRegTest’ performs the following:

1. If the region width for the register is one byte then it sets a most significant bit variable to seven;
2. If the region width for the register is 2 bytes then it sets a most significant bit variable to 15;
3. If the region width for the register is 4 bytes then it sets a most significant bit variable to 31;
4. Calls function ‘scmWalkD0Test’ passing it the pointer to the register, the registers least significant bit set to 0, the most significant bit, the mask and a flag indicating that the test should be performed once with error checking;
5. Returns ERROR if ‘scmWalkD0Test’ returns ERROR;
6. Calls function ‘scmWalkD0Test’ passing it the pointer to the register, the registers least significant bit set to 0, the most significant bit, the mask and a flag indicating that the test should be performed once with error checking;
7. Returns ERROR if ‘scmWalkD1Test’ returns ERROR;
8. Returns OK;

19.8.7.3 External interfaces

For more information see "Function ‘scmWalkD1Test’ External Interface" on page 73.

7. Repeats steps 4, 5, 6;
8. Re-initializes the pointer to point to the base of the address space under test;
9. Reads the location pointed to by the pointer;
10. If the flag indicates that this test is only to be performed once with error checking and if the value read from the location above is not the same as that written there earlier in the test ANDed with the mask, then it returns ERROR;
11. Initializes a pattern to 0x1;
12. ORs the pointer and pattern together;
13. If the pointer is not greater than the end address of the memory space under test then it reads the location pointed to by the pointer. Otherwise it continues to step 17;
14. If the flag indicates that this test is only to be performed once with error checking and if the value read from the location above is not the same as that written there earlier in the test ANDed with the mask, then it returns ERROR;
15. Shifts the bits in the pattern one location to the left;
16. Repeats steps 12, 13 and 14;
17. If the flag indicates that this test is only to be performed once with error checking then it returns OK. Otherwise it repeats the above steps.

All reads and writes are carried out according to the memory region width passed from the calling function.

19.8.6.3 External interfaces

For more information see "Function ‘scmWalkD1Test’ External Interface" on page 73.

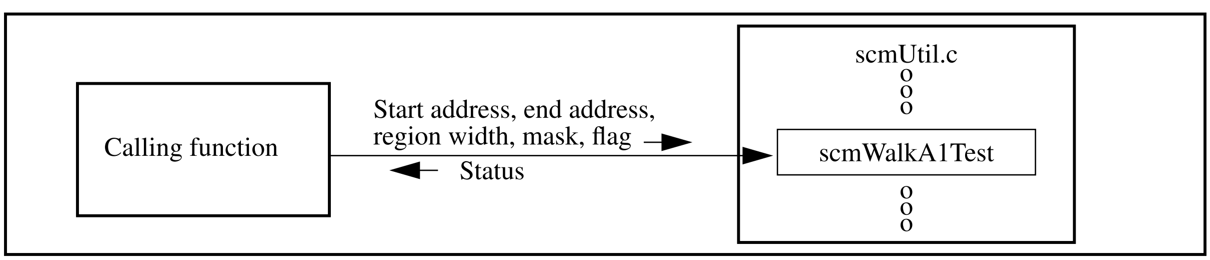


Figure 21 Function ‘scmWalkA1Test’ External Interface

19.8.6.4 Internal interfaces

Not applicable.

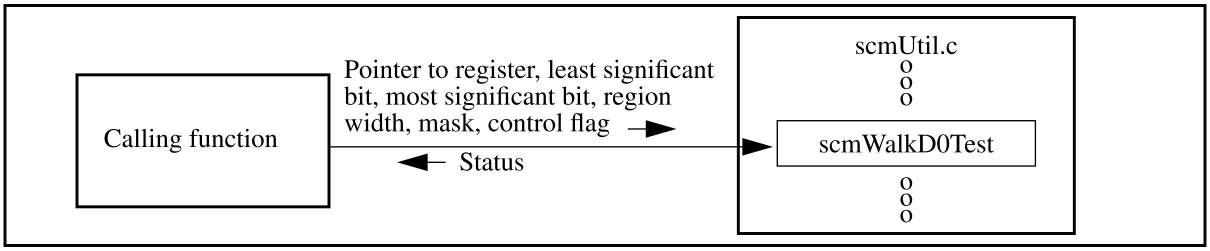


Figure 20 Function ‘scmWalkD0Test’ External Interface

19.8.5.4 Internal interfaces

For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

19.8.6 Function ‘scmWalkA1Test’

19.8.6.1 Purpose

The purpose of function ‘scmWalkA1Test’ (scu Address-bit Walking One Test) is to walk a one across the address bits of a portion of memory placing the address in the memory location and checking for retention. Calling functions pass it a pointer to the base of the address space to test, a pointer to end of the address space, the width of the memory region in which this memory space lies, a mask indicating which bits should be checked and a flag indicating whether the test should be done once with error checking or continuously without error checking. A flag is used to permit this test to be run repeatedly without error checking so that this function can also be used in the initial stages of H/W debugging as well as during start-up diagnostics.

19.8.6.2 Function

Function ‘scmWalkA1Test’ performs the following:

1. Initializes a pointer to point to the base of the address space under test;
2. Writes the value of the pointer ANDed with the mask to the location addresses by the pointer;
3. Initializes a pattern to 0x1;
4. ORs the pointer and pattern together;
5. If the pointer is not greater than the end address of the memory space under test then it writes the value of the pointer ANDed with the mask to the location addresses by the pointer. Otherwise it continues to step 8;
6. Shifts the bits in the pattern one location to the left;

19.8.5 Function ‘scmWalkD0Test’

19.8.5.1 Purpose

The purpose of function ‘scmWalkD0Test’ (SCU Data-bit Walking Zero Test) is to walk a zero across (a portion of) a memory location. Calling functions pass it a pointer to the address location to test, the least significant and most significant bits between which the zero is walked, a mask enabling certain bits to be bypassed, the memory region width and a flag indicating whether the test should be done once with error checking or continuously without error checking. A flag is used to permit this test to be run repeatedly without error checking so that this function can also be used in the initial stages of H/W debugging as well as during start-up diagnostics.

19.8.5.2 Function

Function ‘scmWalkD0Test’ performs the following:

1. Writes all ones (all bits set) to the location under test and reads it back. If the flag indicates that the test is only performed once with error checking then it returns ERROR if the location does not contain all ones.
2. Initializes a pattern to be written to the location under test to have all bits set except the least significant bit at which to start walking;
3. Writes the pattern to the location under;
4. Reads the location under test and compares the value read to that written in step 3. If the flag indicates that the test is only performed once with error checking and the value read is not what was written then it returns ERROR;
5. If the pattern does not have the bit clear corresponding to the most significant bit, then the clear bit is shifted left, one bit at a time until it corresponds to a set bit in the mask. Steps 3 and 4 are repeated;
6. When the clear bit in the pattern reaches the most significant bit then steps 3, 4 and 5 are repeated walking the clear bit to the right back towards the least significant bit.
7. If the flag passed from the calling function indicates that this should only be executed once with error checking then it returns OK. Otherwise it repeats steps 1 through 6.

All reads and writes are carried out according to the memory region width passed from the calling function.

19.8.5.3 External interfaces

For more information see "Function ‘scmWalkD1Test’ External Interface" on page 73.

3. Writes the pattern to the location under;
4. Reads the location under test and compares the value read to that written in step 3. If the flag indicates that the test is only performed once with error checking and the value read is not what was written then it returns ERROR;
5. If the pattern does not have the bit set corresponding to the most significant bit, then the set bit is shifted left, one bit at a time until it corresponds to a set bit in the mask. Steps 3 and 4 are repeated;
6. When the set bit in the pattern reaches the most significant bit then steps 3, 4 and 5 are repeated walking the bit to the right back towards the least significant bit.
7. Writes a zero to the location under test and reads it back. If the flag indicates that the test is only performed once with error checking then it returns ERROR if the location does not contain zero.
8. If the flag passed from the calling function indicates that this should only be executed one with error checking then it returns OK. Otherwise it repeats steps 1 through 6.

All reads and writes are carried out according to the memory region width passed from the calling function.

19.8.4.3 External interfaces

For more information see "Function ‘scmWalkD1Test’ External Interface" on page 73.

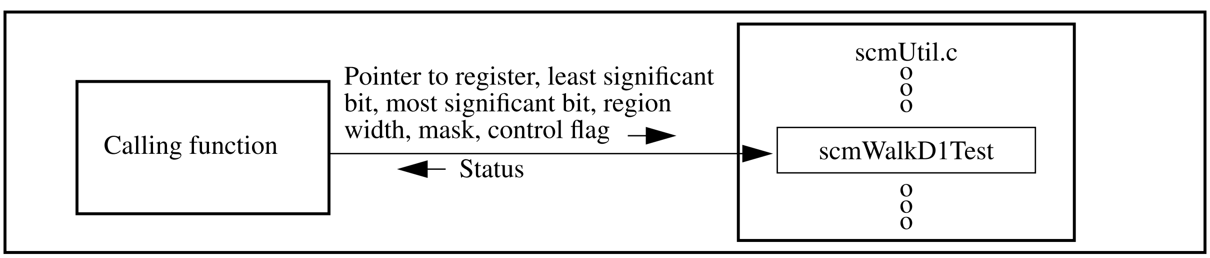


Figure 19 Function ‘scmWalkD1Test’ External Interface

19.8.4.4 Internal interfaces

For more information see "Function ‘scmRegTest’ Internal Interface" on page 78.

19.8.3.3 External interfaces

This function may be called by any function that checks for a DRAM parity error. For more information see "Function ‘scmChkDramParity’ External Interface" on page 72.

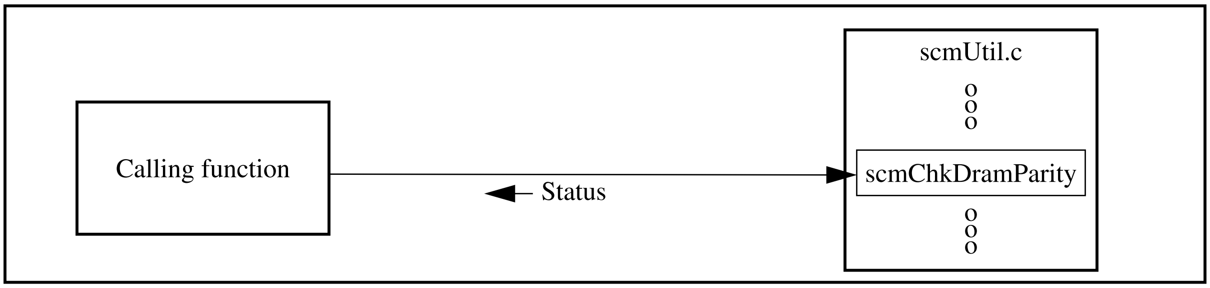


Figure 18 Function ‘scmChkDramParity’ External Interface

19.8.3.4 Internal interfaces

This function is called by ‘scmMarch6Test to test for DRAM parity error. ‘It returns STATUS (OK or ERROR) to indicate whether a DRAM parity error was found.

19.8.4 Function ‘scmWalkD1Test’

19.8.4.1 Purpose

The purpose of function ‘scmWalkD1Test’ (SCU Data-bit Walking One Test) is to walk a one across (a portion of) a memory location. Calling functions pass it a pointer to the address location to test, the least significant and most significant bits between which the one is walked, a mask enabling certain bits to be bypassed, the memory region width and a flag indicating whether the test should be done once with error checking or continuously without error checking. A flag is used to permit this test to be run repeatedly without error checking so that this function can also be used in the initial stages of H/W debugging as well as during start-up diagnostics.

19.8.4.2 Function

Function ‘scmWalkD1Test’ performs the following:

1. Writes a zero to the location under test and reads it back. If the flag indicates that the test is only performed once with error checking then it returns ERROR if the location does not contain zero.
2. Initializes a pattern to be written to the location under test to have one bit set corresponding to the least significant bit at which to start walking;

```
6. For end_address to base_address
     Test the word for background;
   If (check DRAM PE) then check for parity error;
```

The first time the algorithm is executed foreground=1010... and the second time foreground=1111... This function should return the address of the failing memory location if any one of the test lines fails, otherwise it returns OK.

19.8.2.3 External interfaces

This function can be called by any function that performs a memory test. For more information see "Function 'scmMarch6Tst' External Interface" on page 71.

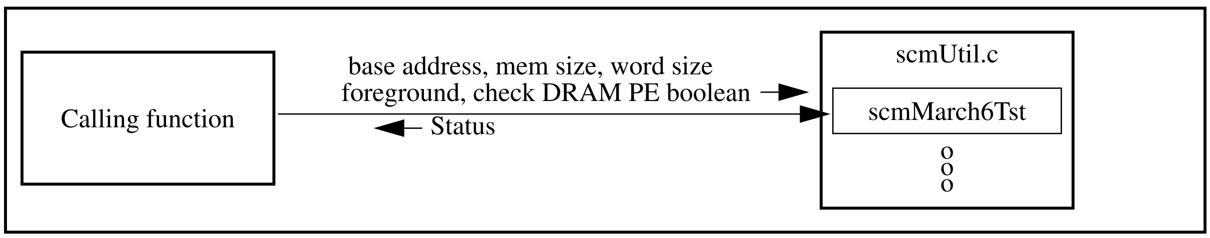


Figure 17 Function 'scmMarch6Tst' External Interface

19.8.2.4 Internal interfaces

This function calls 'scmChkDramParity' to test for DRAM parity error. 'scmChkDramParity' returns STATUS (OK or ERROR) to indicate whether a DRAM parity error was found.

19.8.3 Function 'scmChkDramParity'

19.8.3.1 Purpose

The purpose of function 'scmChkDramParity' (SCM Check DRAM Parity) is to check for a DRAM parity error and returns STATUS to the calling function.

19.8.3.2 Function

'scmChkDramParity' performs the following:

1. Reads the GPU NMI error source register;
2. Checks the DRAM parity error bit in the above register;
3. Clears the NMI and returns ERROR if set the above DRAM parity error bit is set;
4. Returns OK if the DRAM parity error bit is not set.

4. SCU Initialization.
5. User I/F on workstation (workstation commands may be used to initiate a certain form of initialization or reset);
6. Switch Maintenance (under the topics of shutdown/restart. PC reset, etc.).

19.8 Module ‘scmUtil.c’

19.8.1 Purpose

Module ‘scmUtil.c’ (SCM Utilities) serves as a library which contains several utility functions which are called from various other modules at any point during switch operation.

19.8.2 Function ‘scmMarch6Tst’

19.8.2.1 Purpose

This function executes the MARCH-6 memory test. It tests a memory block for faults such as stuck memory, leakages and decoding problems. It also detects faults with parity bits if input boolean, check DRAM PE, is set to TRUE.

19.8.2.2 Function

For a block of memory, function ‘mcMarch6Tst performs the following algorithm twice:

```
1. For base_address to end_address
    Write background;

2. For base_address to end_address
    Test the word for background;
    If (check DRAM PE) then check for parity error;
    Write foreground (foreground=hFFFF XOR background);

3. For base_address to end_address
    Test the word for foreground;
    If (check DRAM PE) then check for parity error;
    Write background;

4. For end_address to base_address (reverse address)
    Test the word for background;
    If (check DRAM PE) then check for parity error;
    Write foreground;

5. For end_address to base_address
    Test the word for foreground;
    If (check DRAM PE) then check for parity error;
    Write background;
```

For more details about the PM Application Software Refer to Harvard Documents.

19.7.9 FLASH bootrom software upgrade

The process of upgrading the FLASH bootrom on the PM involves the following steps:-

- The GPU requests to shutdown the PM gracefully.
- The GPU loads the upgrading software to the PPU using the Communication Interface functions.
- The GPU starts the upgrading process which will involve the Communication Interface functions to transfer the upgraded software image to the PPU memory.
- The PPU will re-program the FLASH memory with the upgrade image.
- The GPU re-initialize the PM (cold start).
- The GPU re-load the Application Software to the PM and Scheduler.
- The PM enters the “running mode”.

19.7.10 PM enters the “running mode”

This is the final step of the initialization process where the PPU starts running the application software. The PPU will not reach this stage unless all the previous steps passed successfully without errors.

19.7.11 Operations

19.7.12 User initiated actions

A user initiates the process of PM Initialization

- By powering up the switch when it is initially off
- By entering a PM initialization command from the workstation UI after inserting a new PM card to the switch or re-booting an existing one.

The UI provides an interface to the command line of VxShell running on the switch. The user issues a command from that interface to initialize a PM. The UI is responsible of displaying the initialization process status and error messages.

19.7.13 Associated activities

The following is a list of possible associated activities:

1. PM Diagnostics;
2. FLEX configuration;
3. Communication Interface;

19.7.6.1 Ingress Cell Path (ICP)

19.7.6.2 TBD

19.7.6.3 Egress Cell Path (ECP)

19.7.6.4 TBD

19.7.6.5 Cell Drop Activetor (CDA)

19.7.6.6 TBD

19.7.6.7 Credit Protocol Manager (CPM)

19.7.6.8 TBD

19.7.6.9 Cell Output Scheduler

1. Scheduler Bootstrap

   Since the only memory mapped to the Scheduler is the shared DPRAM memory with the PPU, the Scheduler finds its IMI, boot code and the scheduler application software in the DPRAM. The PPU asserts the Scheduler RESET line to start booting the Scheduler. Also a WD timer controlled by the PPU to time-out the Scheduler i960CA processor in case of booting failure.

2. Scheduler Initialization

   The following initialization is done by the Scheduler processor:

   - Test the Scheduler to PPU DPRAM.
   - Enable watch dog timers
   - Initialize registers used to interface to other PPU blocks.
   - Initialize the scheduler software data structures.

19.7.7 PM Hardware Blocks Start-Up

TBD

19.7.8 Download PM Application Software

After the PM Initialization and Diagnostics phase passed successfully the PPU enters a stage waiting for the PM application software to be downloaded. The main responsibility of PM Application software is “Software Filtering“ supported by the Communication Interface functions.

Part of the GPU Application Software is responsible for invoking the PM Application Software using the Communication Interface functions.

19.7.4 Configure FLEX Devices

The PPU initialization software is responsible for configuring the FLEX devices, the following steps are involved :-

- Checking for valid configuration information.
- Preparing the device for programming.
- Programming of the device.
- Checking device status before, during, and after programming.

Note: FLEX devices are capable of checking for their own status and the sanity of their configuration data. Their status are then indicated to the PPU via a special signal.

For more information see "PPU Hardware Specifications" . This document includes details about the interface between the PPU and the FLEX devices.

For more information see "FLEX Configuration Design Document" . This document includes details about the functional description of the FLEX configuration software.

19.7.5 PM blocks Diagnostics

For more information see "Port Module Diagnostics" .

19.7.6 PM Hardware Block Initialization

The following sections outline the initialization requirements of the PM blocks. Any hardware blocks, memories or registers mentioned here should pass start-up diagnostics. For more information see "Port Module Diagnostics" .

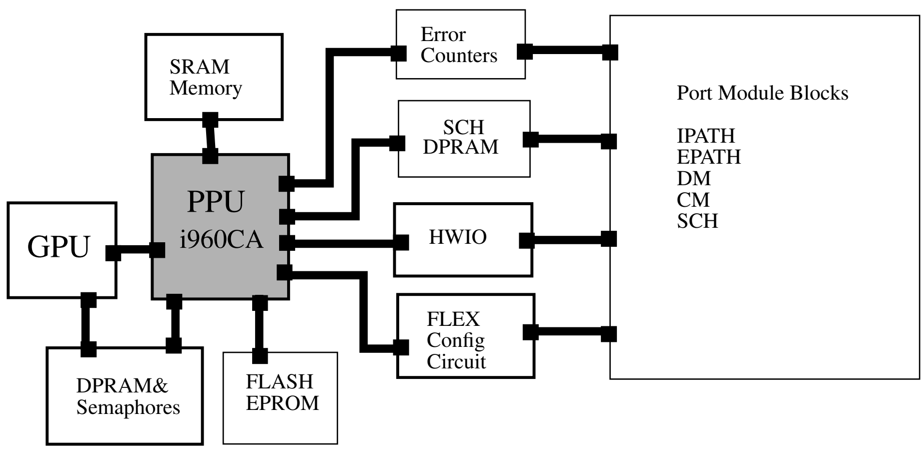
- The PPU Reads the ID-PROM on the PM card to reconfigure the PM memory regions accordingly.

For more information see "Chapter 14, Initialization and system requirements, i960 CA Microprocessor Users Manual" .

19.7.2 PPU Start-up Diagnostics

PPU start-up diagnostics perform the necessary testing on PPU hardware and memories as outlined in "PPU Block Diagram" on page 66.

Figure 1 PPU Block Diagram



For more information see "Port Module Diagnostics" .document.

19.7.3 Communication Interface Establishment

After the PPU finishes the initial diagnostics phase successfully, it establishes the Communication Interface between the PPU and the GPU to enable sending and receiving messages between the two processors. This Communication Interface permits the GPU software to do the following functions during initialization and diagnostics:-

- Receives error and status messages from the PPU while initializing and diagnosing the rest of the PM blocks.
- Monitors PM memory locations.
- Provides read and write access to PM hardware blocks.
- Loads and invokes special diagnostics routines.
- Downloads and give control to the PM Application Software after the PM diagnostics and initialization processes passed successfully.

For more information see "The CreditNet Switch Inter-Processor Communication Interface" .

19.7 Module PM Initialization SW

19.7.1 PPU Bootstrap

The following steps are carried out in order to bootstrap the PPU i960CA processor and leave the processor in a known state. The PM software starts off the FLASH bootrom which is mapped to the PPU i960CA processor boot area.

Part of these steps are carried out from the GPU processor and the rest are carried out from the PPU processor.

GPU steps

The GPU software starts the process that caries out the following steps in order to bootstrap the PPU :-

- Checks if the PM card exists and reports error if the card does not exists.
- Tests the GPU-PPU DRAM memory and semaphores from the GPU side.
- Enables the WatchDog (WD) timer.
- Directs the PPU to take the Initial Boot Record (IBR) and the Software from the DPRAM or the FLASH bootrom.
- De-asserts the RESET line to low value for 1m sec. and then asserts it high to bootstrap the PPU processor.
- Monitors the progress of the PPU booting process and reports status and errors to the User Interface

PPU steps

This section describes the steps that the PPU i960CA processor takes during its initialization.

- The PPU is designed so that the STEST pin is always enabled to executes a bus confidence self test and internal self test program on the processor. Pass or failure of the self tests is indicated by the FAIL pin, since this pin is not accessible by the GPU, WD timer is required to time-out the processor in case of an internal failure.
- When initialization begins the processor uses an Initial Memory Image (IMI) to establish its initial state. The IMI comprises the minimum set of data structure that the PPU processor needs to initialize its system. The IMI performs two functions.

  3 Configure memory regions, bus widths, bytes ordering, external Interrupts and instruction cache configuration.

  4 Provides pointers to the system data structures and first instruction to be executed of the software which is either in the FLASH bootrom or the DPRAM.

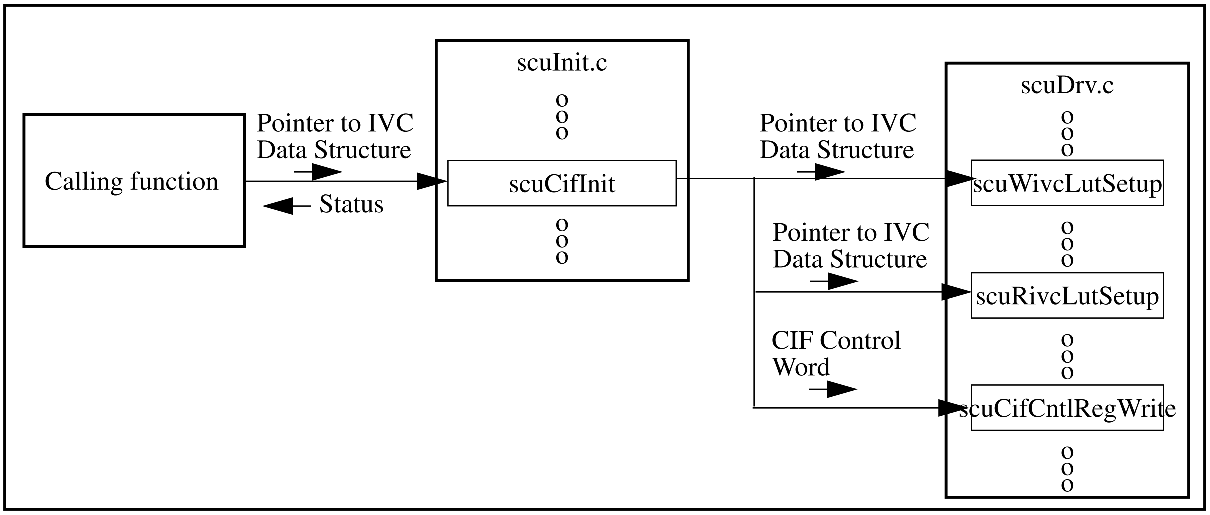


Figure 16 Function ‘scuCifInit’ External Interface

19.5.11.4 Internal interfaces

Not applicable.

19.6 Module and Function Switch Configuration SW

Ihab

19.6.1 Purpose

19.6.2 Function

19.6.3 External interfaces

19.6.4 Internal interfaces

19.5.11 Function ‘scuCifInit’

19.5.11.1 Purpose

The purpose of function ‘scuCifInit’ (SCU Cell Interface (CIF) Initialize) is to initialize the CIF’s IVC Lookup Tables, Shared Buffer, FIFOs and Control Register.

19.5.11.2 Function

Function ‘scuCifInit’ performs the following:

1. Calls function ‘scuWivcLutSetup’ passing it a pointer to the IVC Data structure in order to initialize the W_IVC Lookup Table;
2. Calls function ‘scuRivcLutSetup’ passing it a pointer to the IVC Data structure in order to initialize the R_IVC Lookup Table;
3. Fills the entire Shared Buffer with 0x0 then fills the first and last cells with Idle cells;
4. Resets the CIF’s Low Priority W_IVC FIFO, High Priority W_IVC FIFO and R_IVC FIFO by calling function ‘scuCifCntlRegWrite’ asserting the CIF FIFO Reset bit:
5. Initializes the CIF Control Register by calling function ‘scuCifCntlRegWrite’ with the following:
    - CRC8 generation/checking enabled;
    - Cell encoding/decoding enabled;
    - SCU Reset bit disabled;
    - CIF FIFO Reset bit disabled;
    - All other bits disabled;

19.5.11.3 External interfaces

This function is invoked by function ‘sysInitCtl’ in module ‘sysInitCtl.c’. For more information see "Function ‘scuCifInit’ External Interface" on page 64.

19.5.10.2 Function

Function ‘scuFlmInit’ performs the following:

1. Calls function ‘scuFlmReset’ to reset the FLM’s CMBF and internal pointers;
2. Calls function ‘scuFlmFLCInit’ to initialize the FLM RAM with the complete range of CADR. If this ‘scuFlmInit’ returns ERROR then this function in turn returns ERROR;
3. Calls function ‘scuFlmHaltClear’ to clear the FLM Halt (which prevents the issuing of CADRs from the FLM);
4. Calls function ‘scuFlmPrime’ to remove the first four CADR from FLM RAM and place them in the FLM feedback pipeline:

19.5.10.3 External interfaces

This function is invoked by function ‘sysInitCtl’ in module ‘sysInitCtl.c’. For more information see "Function ‘scuFlmInit’ External Interface" on page 62.

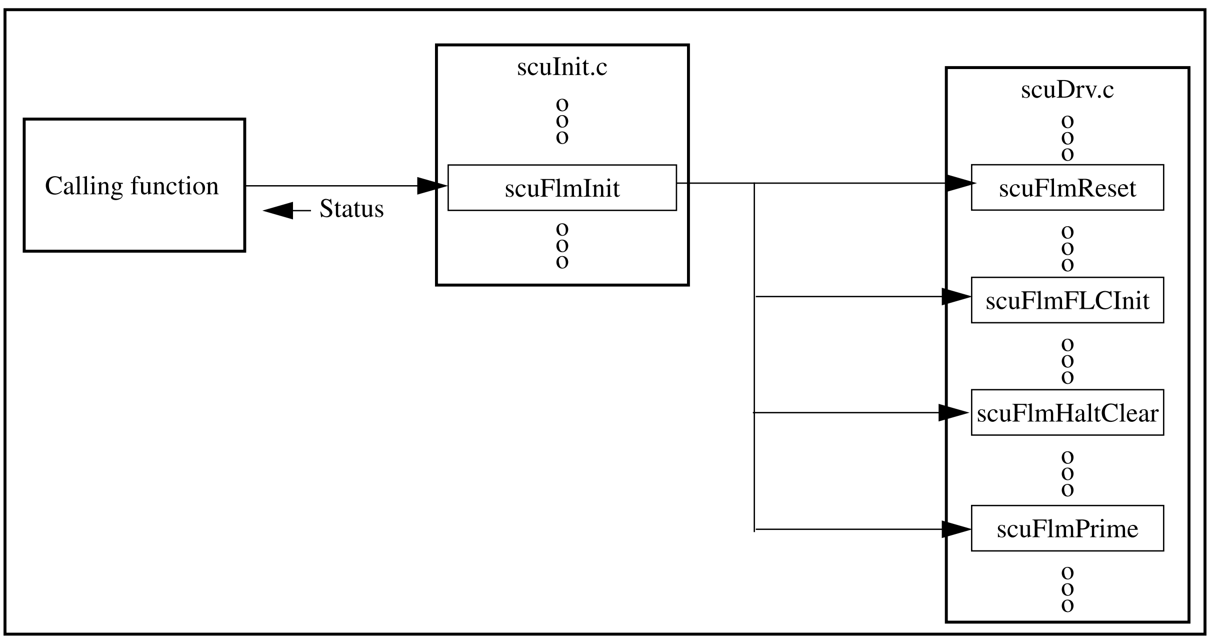


Figure 15 Function ‘scuFlmInit’ External Interface

19.5.10.4 Internal interfaces

This function is invoked by ‘scuHwInit’. For more information see "Function ‘scuHwInit’ Internal Interface" on page 56.

- Deasserts the CM search enable bit;
- Deasserts the clear CM search flag bit;
- Deasserts the CM address reset flag bit;
- Deasserts the clear CM alarm flag bit;

Steps 4. and 5. together are responsible for resetting the CM search address, disabling the CM search and clearing the CM search done flag and CM search alarm flag.

19.5.9.3 External interfaces

For more information see "Function ‘scuCdmInit’ External Interface" on page 61.

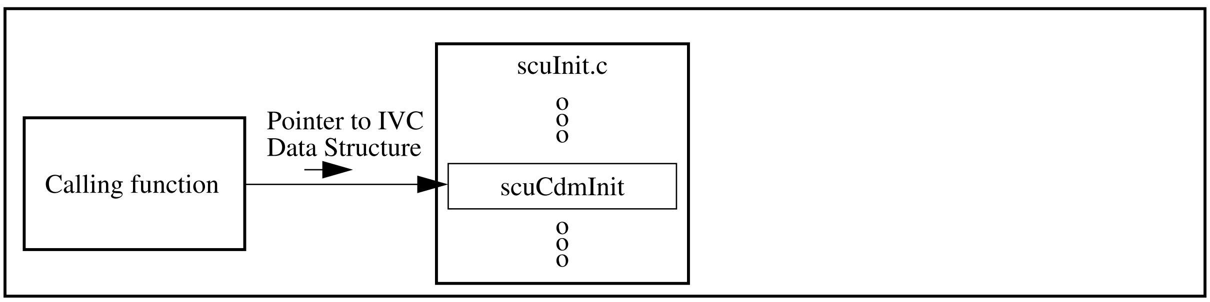


Figure 14 Function ‘scuCdmInit’ External Interface

19.5.9.4 Internal interlaces

This function is invoked by ‘scuHwInit’. For more information see "Function ‘scuHwInit’ Internal Interface" on page 56.

19.5.10 Function ‘scuFlmInit’

19.5.10.1 Purpose

The purpose of function ‘scuFlmInit’ (SCU Free List Manager (FLM) Initialize) is to initialize the FLM with the full range of CADRs and prepare these for use by the cards in the switch as well as initialize the Core Memory Buffer Fill (CMBF) count.

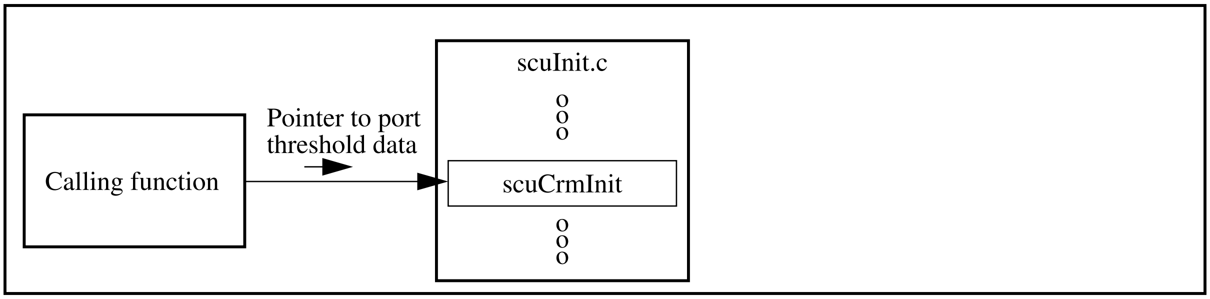


Figure 13 Function ‘scuCrmInit’ External Interface

19.5.8.4 Internal interlaces

This function is invoked by ‘scuHwInit’. For more information see "Function ‘scuHwInit’ Internal Interface" on page 56.

19.5.9 Function ‘scuCdmInit’

19.5.9.1 Purpose

The purpose of function ‘scuCdmInit’ (SCU Core Distribution Manager (CDM) Initialize) is to initialize the CDM’s Distribution Bitmap (DBM) RAM, Working DBM (wDBM) RAM and Chalkmark search facilities to known suitable initial values.

19.5.9.2 Function

Function ‘scuCdmInit’ performs the following:

1. Initializes the current Unicast and Multicast Chalkmarks (CMs) to 0x0;
2. Calls function ‘scuDbmInit’ to initialize the DBMs to 0x0 and casting bits to Unicast for all 8K IVCs;
3. Uses function ‘scuCellXferToCore’ to perform valid Core Memory Card (CMC) write cycles in order to initialize the wDBMs and CMs for all 16K CADRs to 0x0;
4. Writes the CDM Control Register bits according to the following:
    - Deasserts the CM search enable bit;
    - Asserts the clear CM search done flag bit;
    - Deasserts the CM address reset flag bit;
    - Asserts the clear CM alarm flag bit;
5. Writes the CDM Control Register bits according to the following:

TBD when the SCU personality design is finalized.

19.5.7.3 External interfaces

19.5.7.4 Internal interfaces

Not applicable.

19.5.8 Function ‘scuCrmInit’

19.5.8.1 Purpose

The purpose of function ‘scuCrmInit’ (SCU Core Resource Monitor (CRM) Initialize) is to initialize the CRM Port Thresholds and Global BF Threshold to known values.

19.5.8.2 Function

Function ‘scuCrmInit’ performs the following:

1. Initializes the Port BF Threshold for slot 0, the SCU slot, to the following:
    - Lower BF Threshold - 0x100;
    - Middle BF Threshold - 0x1000;
    - Upper BF Threshold - 0x3C00;
    - RPI Threshold - 0x3C00;
2. Initializes all Port BF Thresholds for slots 1 to 15 to 0x0;
3. Initializes the Global BF Threshold to 0x32f0 (80% full);

19.5.8.3 External interfaces

For more information see "Function ‘scuCrmInit’ External Interface" on page 60.

19.5.6.1 Purpose

The purpose of function ‘scuCtcInit’ (SCU Cell Time Counter Initialize) is to initialize the CTC timer to run in real-time clock mode and so that the timer control registers are setup to start and stop on known conditions.

19.5.6.2 Function

Function ‘scuCtcInit’ performs the following:

1. Writes 0x0 to the CTC Control/Status Register to place the CTC counter in real-time counter mode (as opposed to Cell-timer mode);
2. Writes 0x1FFF to the Start Control Register so that the timer starts on a W_IVC of 0x1FFF if it is enabled to do timing;
3. Writes 0x1FFF to the Start Control Register so that the timer starts on a W_IVC of 0x1FFF if it is enabled to do timing;

19.5.6.3 External interfaces

For more information see "Function ‘scuCtcInit’ External Interface" on page 58.

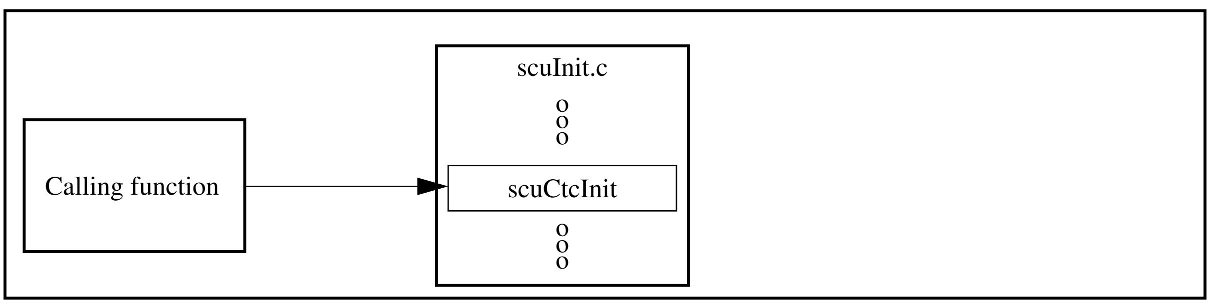


Figure 12 Function ‘scuCtcInit’ External Interface

19.5.6.4 Internal interlaces

This function is invoked by ‘scuHwInit’. For more information see "Function ‘scuHwInit’ Internal Interface" on page 56.

19.5.7 Function ‘scuParCrcInit.c’

19.5.7.1 Purpose

19.5.7.2 Function

19.5.5.1 Purpose

The purpose of function 'scuPifInit' (SCU Processor Bus Initialize) is to initialize the various registers in the PIF.

19.5.5.2 Function

Function 'scuPifInit' performs the following:

1. Writes 0xC0A0 to the Program/Test Enable Register to disable Lattice and FLEX programming as well as FLM and DBM RAM in-circuit testing;
2. Writes 0x0 to the Interrupt/Alarm Control Register to reset all PM interrupts and mask all alarms in the Alarm Status Register;
3. Writes 0x3 to the Interrupt/Alarm Control Register to allow the PM interrupts and alarms in the Alarm Status Register;

19.5.5.3 External interfaces

For more information see "Function 'scuPifInit' External Interface" on page 57.

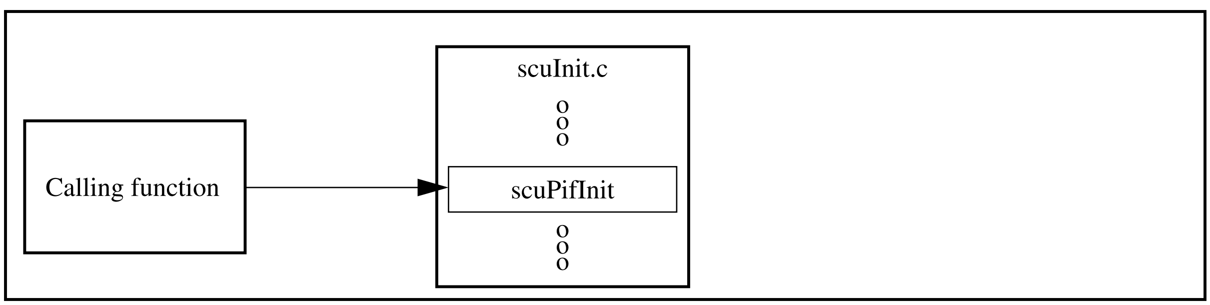


Figure 11 Function 'scuPifInit' External Interface

19.5.5.4 Internal interlaces

This function is invoked by 'scuHwInit'. For more information see "Function 'scuHwInit' Internal Interface" on page 56.

19.5.6 Function 'scuCtcInit'

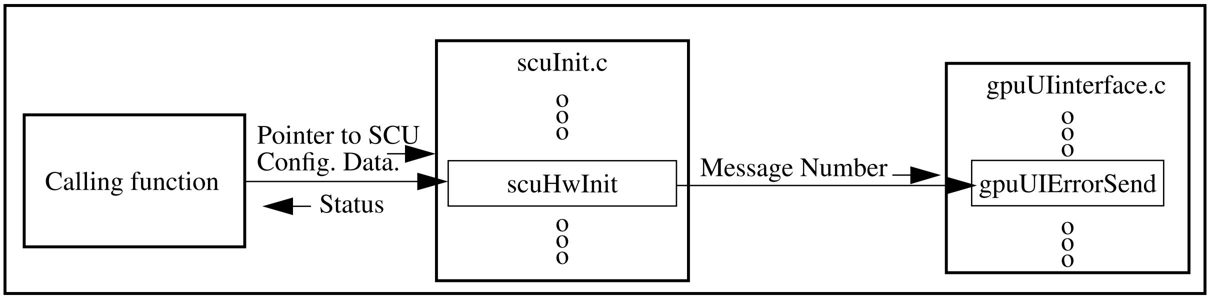


Figure 9 Function ‘scuHwInit’ External Interface

19.5.4.4 Internal interlaces

For more information see "Function ‘scuHwInit’ Internal Interface" on page 56.

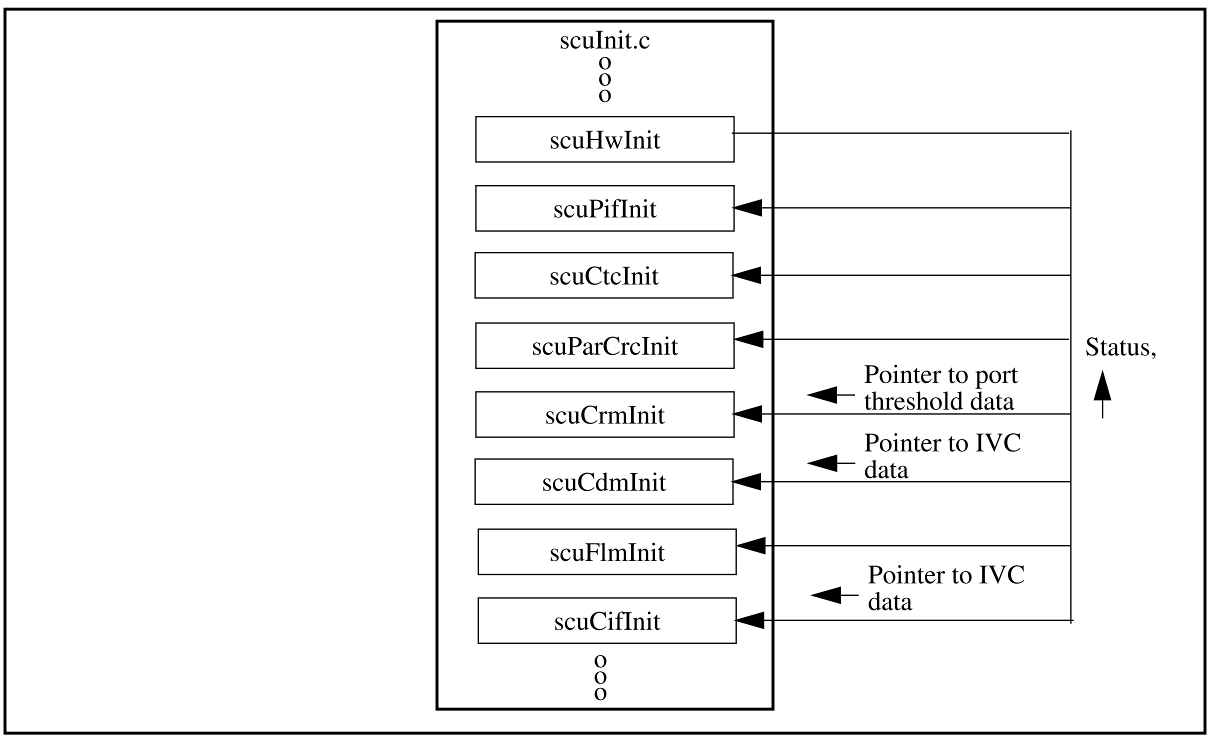


Figure 10 Function ‘scuHwInit’ Internal Interface

19.5.5 Function ‘scuPifInit’

19.5.3.4 Internal interlaces

This function is invoked by ‘scuInitCtl’. For more information see "Function ‘scuInitCtl’ Internal Interface" on page 53.

19.5.4 Function ‘scuHwInit’

19.5.4.1 Purpose

The purpose of function ‘scuHWInit’ (SCU HW Initialize) is to control the initialization of the various SCU HW blocks which are initialized based on the SCU configuration data structure (See "SCU Configuration Data Structure" on page 45.) and report status via the User Interface.

19.5.4.2 Function

Function ‘scuHWInit’ performs the following:

1. Initializes result to OK;
2. Calls function ‘scuPifInit’ to initialize the Processor Bus Interface (PIF);
3. Calls function ‘scuCtcInit’ to initialize the Cell Time Counter (CTC);
4. Calls function ‘scuParCrcInit’ to initialize the Partial CRC calculator;
5. Calls function ‘scuCrmInit’ passing it a pointer to the port threshold data structure, to initialize the Core Resource Manager (CRM);
6. Calls function ‘scuCdmInit’ passing it a pointer to the IVC data structure, to initialize the Core Resource Manager (CRM);
7. Calls function ‘scuFlmInit’ to initialize the Free List Manager (FLM);
8. Calls function ‘scuCifInit’ passing it a pointer to the IVC data structure, to initialize the Cell Interface (CIF);
9. Returns result OK;

This function also keeps track of SCU block initialization progress and sends corresponding status messages to the UI and the VxWorks Shell terminal.

19.5.4.3 External interfaces

This function is invoked by function ‘sysInitCtl’ in module ‘sysInitCtl.c’. It also makes calls to function ‘gpuUIErrorSend’ in the VxWorks library ‘gpuUIinterface.c.’ For more information see "Function ‘scuHwInit’ External Interface" on page 56.

19.5.3.1 Purpose

The purpose of function ‘scuDefDSInit’ (SCU Default Data Structure Initialize) is to initialize a default SCU Configuration Data Structure (See "SCU Configuration Data Structure" on page 45.). Calling functions pass it a pointer to an unitialized SCU configuration data structure.

19.5.3.2 Function

Function ‘scuHWInit’ performs the following:

1. Initializes all 8K IVCs in the SCU IVC Data Structure with the following:
    - R_CSID of 0x0;
    - R_IVC FIFO Enable bit set to disable (0x1);
    - W_CSID of 0x0;
    - Low and High Priority W_IVC FIFO Enable bits both set to disable (0x1);
    - DBM of 0x0;
    - Casting Bit set to Unicast (0x0);
2. Initializes the Port Thresholds for all slots in the Port Threshold Data Structure with the following:
    - Lower BF Threshold of 0x0;
    - Middle BF Threshold of 0x0;
    - Upper BF Threshold of 0x0;
    - RPI Threshold of 0x0.

19.5.3.3 External interfaces

This function is invoked by function ‘sysInitCtl’ in module ‘sysInitCtl.c’. For more information see "Function ‘scuHwInit’ External Interface" on page 56. For more information see "Function ‘scuDefDSInit’ External Interface" on page 54.

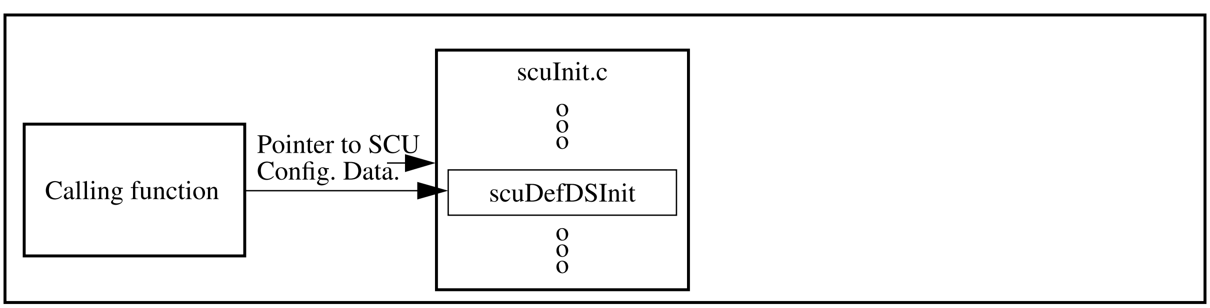


Figure 8 Function ‘scuDefDSInit’ External Interface

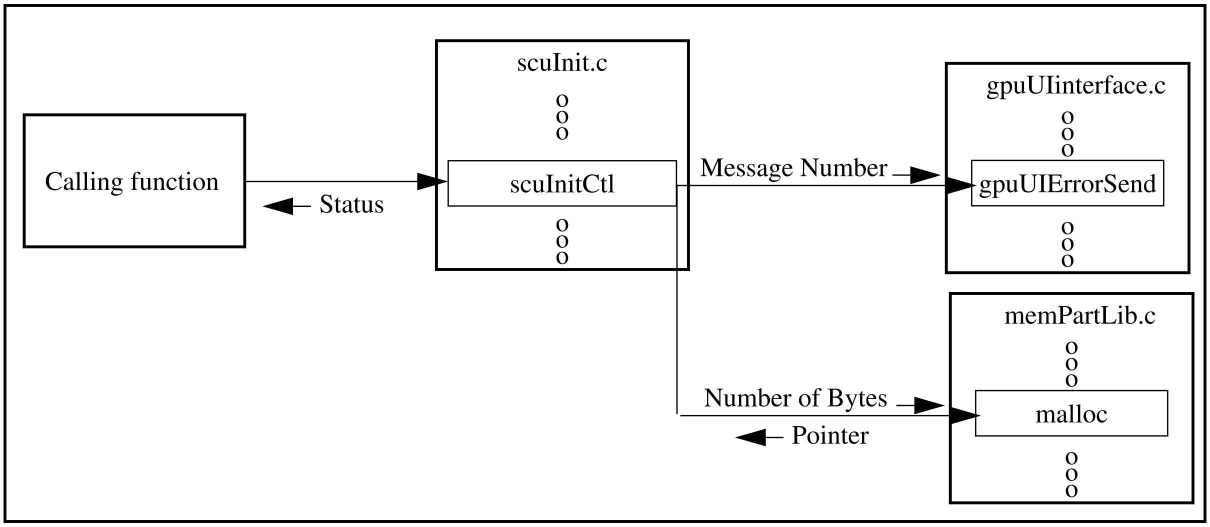


Figure 6 Function ‘sculnitCtl’ External Interface

19.5.2.4 Internal interlaces

For more information see "Function ‘sculnitCtl’ Internal Interface" on page 53.

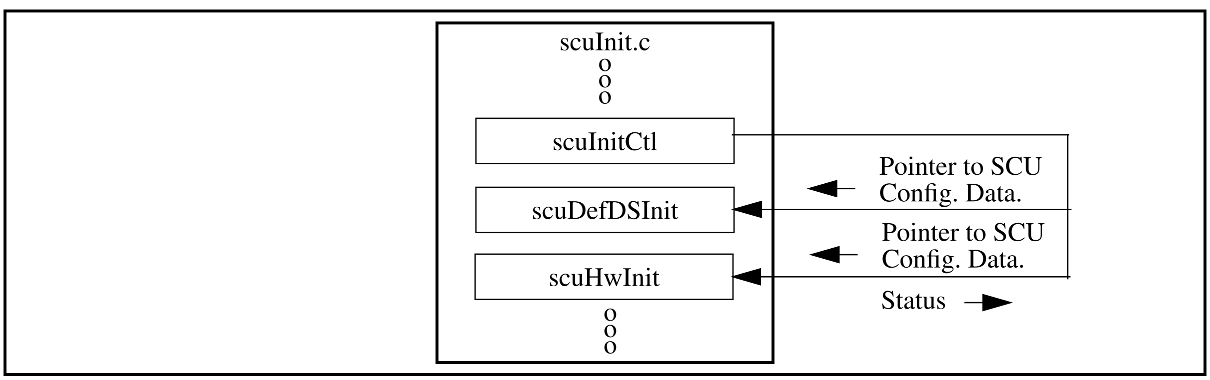


Figure 7 Function ‘sculnitCtl’ Internal Interface

19.5.3 Function ‘scuDefDSInit’

19.5 SCU Initialization - Module ‘scuInit.c’

19.5.1 Purpose

The purpose of ‘scuInitl.c’ (SCU Initialization) is to perform initialization of the SCU configuration data structures and the various SCU hardware blocks.

19.5.2 Function ‘scuInitCtl’

19.5.2.1 Purpose

The purpose of function ‘scuInitCtl’ (SCU Initialize Control) is to control the initialization of the various SCU Configuration Data Structures and HW blocks and report status via the User interface.

19.5.2.2 Function

Function ‘scuHWInit’ performs the following:

1. Initializes result to OK;
2. Allocates memory for a global copy of the SCU Configuration Data Structure;
3. Calls function ‘scuDefDSInit’ to SCU Configuration Data Structure;
4. Calls function ‘scuHWInit’ to initialize the SCU HW blocks. It also sets result to ERROR if ‘scuHWInit’ returns ERROR;
5. Returns result;

This function also keeps track of SCU block initialization progress and sends corresponding status messages to the UI and the VxWorks Shell terminal.

19.5.2.3 External interfaces

This function is invoked by function ‘sysInitCtl’ in module ‘sysInitCtl.c’. It also makes calls to function ‘gpuUIErrorSend’ in the VxWorks library ‘gpuUIinterface.c’ and ‘malloc’ in ‘memPartLib.c.’ For more information see "Function ‘scuInitCtl’ External Interface" on page 53.

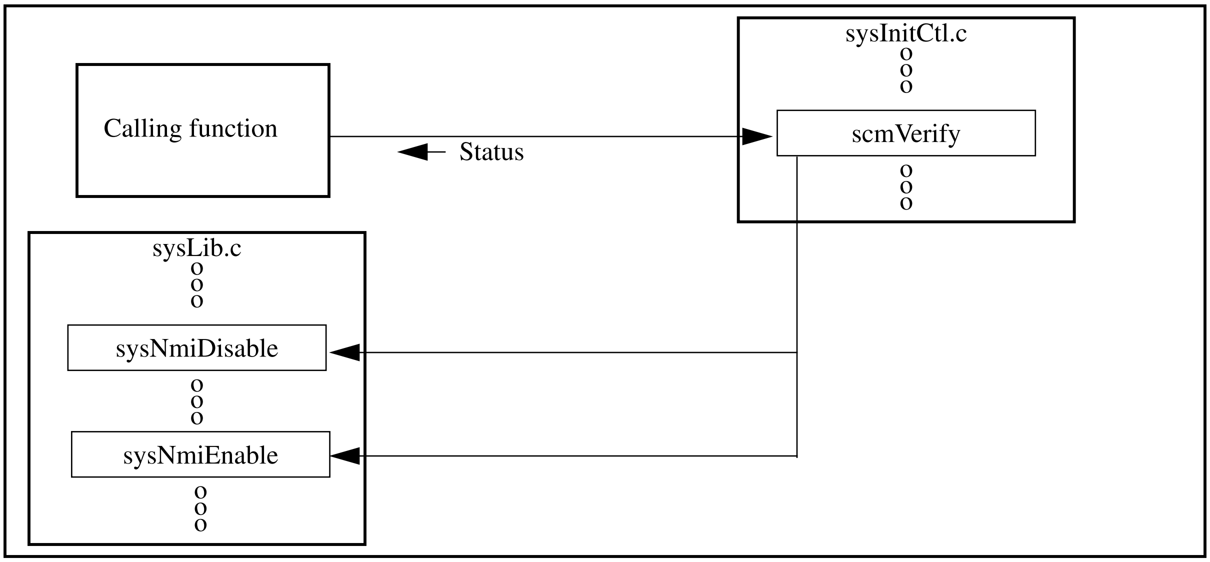


Figure 5 Function 'scmVerify' External Interface

19.2.2.4 Internal interfaces

Not applicable.

19.3 GPU Diagnostics - Module 'gpuDiag.c'

The requirements and design details pertaining to GPU Diagnostics are covered in a separate document. (Please see [5].) Module 'gpuDiag.c' was included in the "System Initialization Software Topology" on page 47 diagram to illustrate how it fits into the overall structure of the System Initialization software.

19.4 SCU Diagnostics - modules 'scuSDiag.c' and 'scuSMDiag.c'

The requirements and design details pertaining to 'scuDiagCtl.c' (SCU Diagnostics) can be found in the FN and DD for SCU, Core and Terminator Card Diagnostics. This module was included in "System Initialization Software Topology" on page 47 to illustrate how it fits into the overall structure of System Initialization SW.

19.2.1.4 Internal interfaces

For more information see "Function ‘sysInitCtl’ Internal Interface" on page 50.

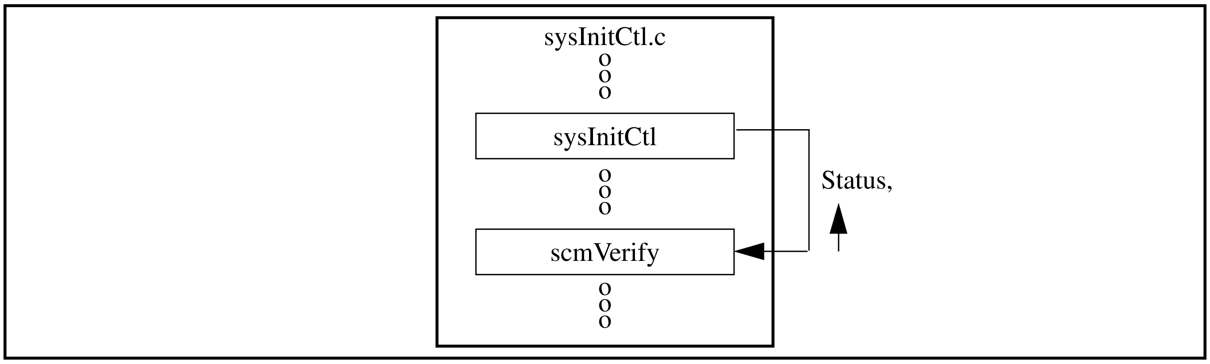


Figure 4 Function ‘sysInitCtl’ Internal Interface

19.2.2 Function ‘scmVerify’

19.2.2.1 Purpose

Function ‘scmVerify’ checks to see whether a GPU is connected to a functioning SCU.

19.2.2.2 Function

Function ‘scmVerify’ performs the following:

1. Checks to see whether GPU NMIs are enabled and disables them if they are enabled;
2. Writes a specific value to the first 32-bit location in the SCU’s CIF Shared Buffer;
3. Reads the first location of the SCU’s CIF Shared Buffer and if the value read is the same as that written in 3. then it sets result to OK. Otherwise it sets result to ERROR;
4. Enables NMIs on the GPU if they were originally enabled in 2.;
5. Returns result;

19.2.2.3 External interfaces

For more information see "Function ‘scmVerify’ External Interface" on page 51.

This function also send messages to the UI via function ‘gpuUIErrorSend’ based on the status returned from the various initialization and diagnostics function it calls.

19.2.1.3 External interfaces

For more information see "Module ‘sysInitCtl.c’ External Interface" on page 49.

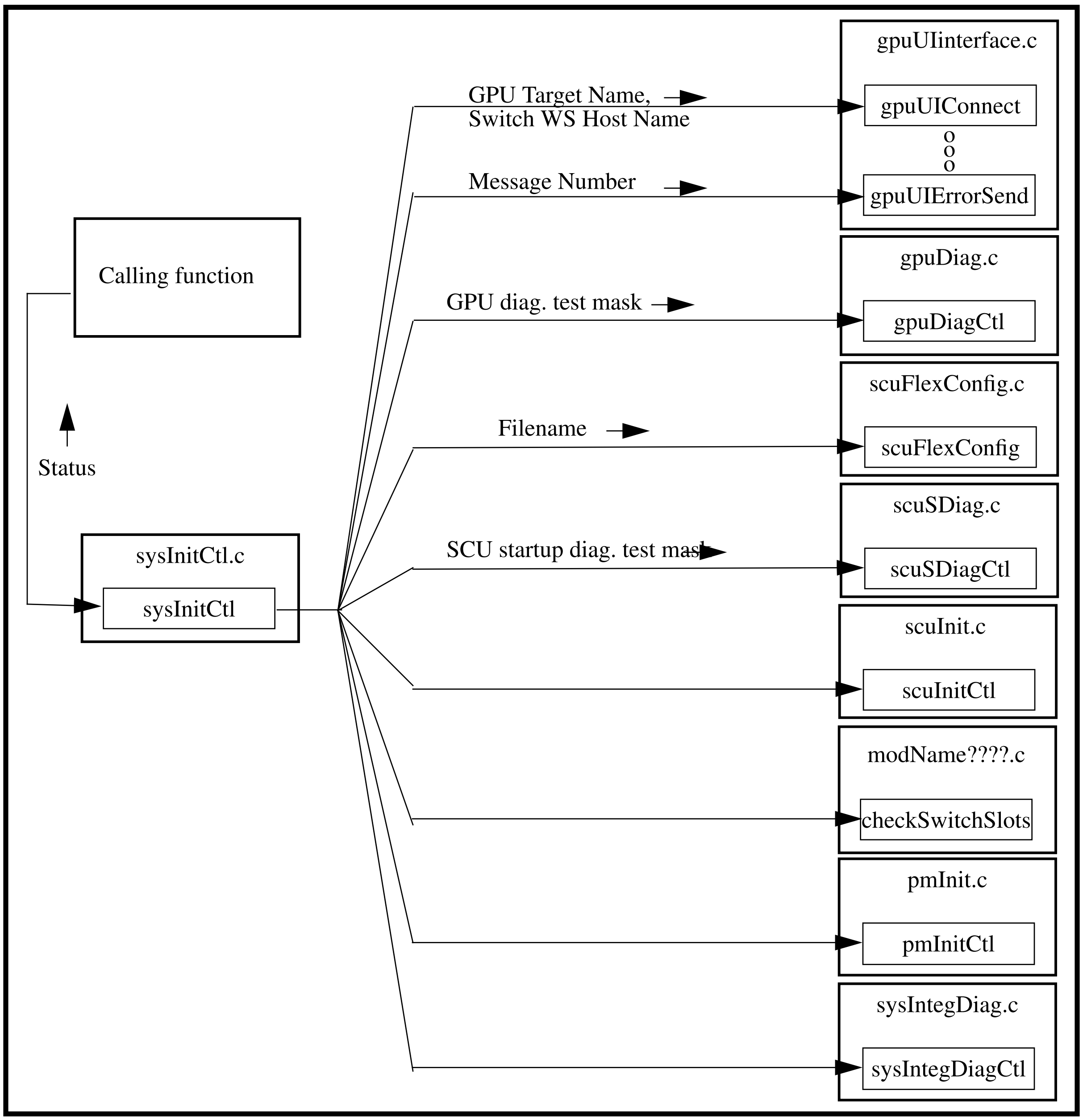


Figure 3 Module ‘sysInitCtl.c’ External Interface

4. Calls function ‘gpuUIErrorSend’ passing it a message number so that the UI indicates that the switch is in the ‘NIS_BOOT’ state;
5. If DIP switch 2 is set, it calls function ‘gpuDiagCtl’ to collect the results of GPU Diagnostics (which have been save in GPU RAM) and passed these results to the Switch User Interface (UI);
6. If ‘gpuDiagCtl’ returns OK it calls function ‘gpuUIErrorSend’ passing it a message indicating that the NIS_BOOT state completed successfully; Otherwise, it calls function ‘gpuUIErrorSend’ passing it a message indicating that the NIS_BOOT state did not complete successfully
7. Calls function ‘gpuUIErrorSend’ passing it a message number so that the UI indicates that the switch is in the ‘NIS_INIT_DIAG’ state;
8. Calls function ‘scuFlexConfig’ passing it the filename of the SCU Flex configuration file. If ‘scuFlexConfig’ returns ERROR then it sets the NIS_INIT_DIAG Status Indicator to ERROR and returns;
9. If DIP switch 3 is set, it calls function ‘scuSDiagCtl’ to perform diagnostics on the SCU hardware blocks. It should pass ‘scuSDiagCtl’ a test mask such that all SCU start-up diagnostics are run;
10. Calls function ‘checkSwitchSlots’ to initialize the Switch Configuration Table (See "Switch Configuration" on page 44.);
11. Calls function ‘pmInitCtl’ to perform initialization and diagnostics on the PMs in the switch. (The PM Diagnostics SW is bypassed if DIP switch 4 is set);
12. If DIP switch 4 is set it calls function ‘tlgDiagCtl’ to perform diagnostics on the TLGs in the switch;
13. If DIP switch 5 is set it calls function ‘sysIntegDiagCtl’ to perform System Integration Diagnostics;
14. Calls function ‘scuInitCtl’ to initialize the SCU hardware blocks for application SW use. This function in turn calls function ‘scuDefDSInit’ to initialize a default SCU initialization data structure and function ‘scuHWInit’ to initialize the SCU HW blocks;
15. Calls function ‘pmInitCtl’ again to reinitialize the PMs for application SW use. This time diagnostics are not performed on the PMs;
16. If either SCU diagnostics or diagnostics on all populated slots failed then this function calls function ‘gpuUIErrorSend’ passing it a message indicating that the NIS_INIT_DIAG state completed unsuccessfully; Otherwise, it calls function ‘gpuUIErrorSend’ passing it a message indicating that the NIS_INIT_DIAG state completed unsuccessfully
17. Calls function ‘gpuUIErrorSend’ passing it a message number so that the UI indicates that the switch is in the ‘NIS_READY’ state;
18. Returns.

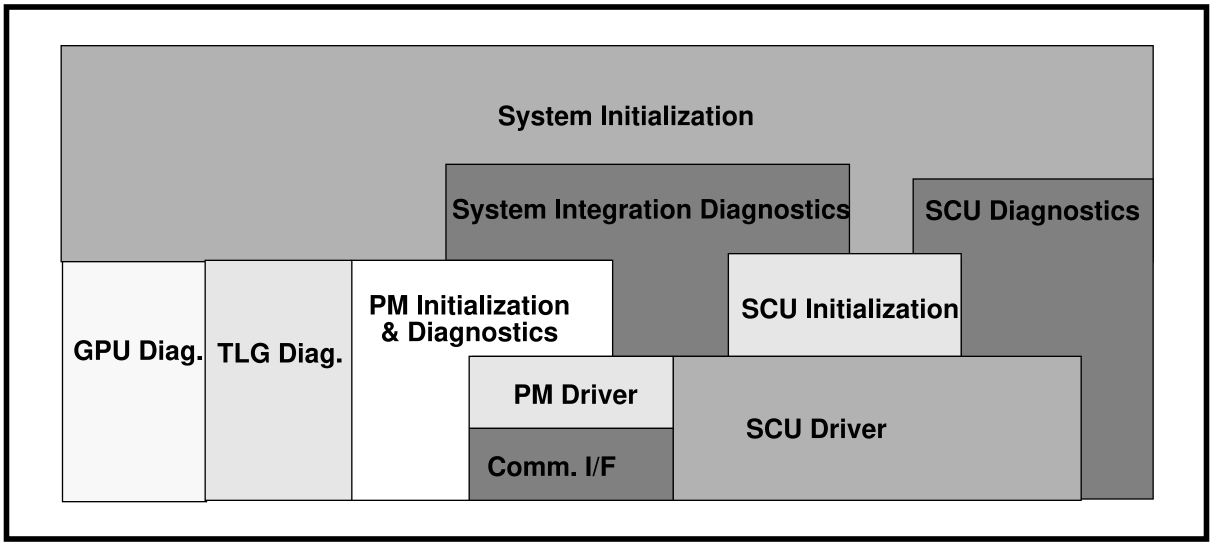


Figure 2 System Initialization Software Topology

19.2 System Initialization - Module ‘sysInitCtl’

Module ‘sysInitCtl.c’ contains functions needed to control the process of System Initialization. The main function, ‘sysInitCtl’, is the control SW for System Initialization while the remaining functions provide functionality required by ‘sysInitCtl’.

19.2.1 Function ‘sysInitCtl’

19.2.1.1 Purpose

Function ‘sysInitCtl’ provides a broad overview of the overall function of System Initialization. It also allows the partitioning of initialization software into independent function and modules making it easier to maintain and understand. It is responsible for initiating communication with the Switch UI, invoking diagnostics on all the HW blocks and initializing the HW blocks.

19.2.1.2 Function

Function ‘sysInitCtl’ performs the following:

1. Calls function ‘gpuUIConnect’ passing it the GPU’s target name and Switch WS host name to initialize communication with the Switch UI;
2. Calls function ‘gpuUIErrorSend’ passing it a message number indicating that the System initialization Controller has started;
3. Calls function ‘gpuUIErrorSend’ passing it a message number indicating that the GPU booted and downloaded VxWorks;

18.2.5.8 NIS_INIT_DIAG Status Indicator

This is a 32-bit integer which indicates the status of System Initialization and is located in GPU RAM at address 0x10000004. This address is not allocated by the VxWorks Operating System or used by the compiler but is instead hardcoded in SW. (Actually the first 1K of GPU RAM, addresses 0x10000000 to 0x100003ff, is shielded from VxWorks and compiler allocation.) The NIS_INIT_DIAG Status Indicator can be either OK (0x0) meaning that application SW can continue to run or ERROR (-1) indicating that a serious HW diagnostics or initialization problem has been encountered and that application SW should not continue to run.

18.2.5.9 PM Initialization Data Structures

See include.sim directory

18.3 Design evolution/strategy

Due to the order in which the SCM and PM hardware will be completed, System Initialization software will be developed approximately in the following order:

1. GPU bootstrap and initialization software;
2. PM and SCM initialization software;
3. System Initialization control software.

19.0 Software specification

19.1 Modular structure

The following figure, "System Initialization Software Topology" outlines the topology of System Initialization SW.

18.2.5.4 SCU IVC Data Table Structure

Tables of this type are composed of an array of a structure with the following elements:

1. IVC;
2. W_CSID
3. High priority W_IVC FIFO enable flag;
4. Low priority W_IVC FIFO enable flag;
5. R_CSID;
6. R_IVC FIFO enable flag;
7. DBM entry;
8. Casting bit.

18.2.5.5 Port Threshold Table Data Structure

Tables of this type are composed of an array of a structure with the following elements:

1. Port Number;
2. Lower Buffer Fill (BF) Threshold;
3. Middle Buffer Fill (BF) Threshold;
4. Upper Buffer Fill (BF) Threshold;
5. Return Path Idle (RPI) Threshold.

18.2.5.6 SCU Configuration Data Structure

This data structure is used to configure the SCU during initialization and is composed of the following data structures and elements:

1. IVC Data Structure;
2. Port Threshold Table Data Structure;

18.2.5.7 W_IVC FIFO Element Data Structure

Tables of this type are composed of an array of a structure with the following elements:

1. IVC;
2. CADR.

18.2.3 Initialization

System Initialization is invoked in one of two stages:

1. GPU Bootstrap (which can be performed in two ways):
    - Switching the power on when the switch is off;
    - Issuing a system cold reset command from the switch workstation.
2. System Initialization Control which can be invoked via:
    - A startup script;
    - From the VxWorks prompt;

18.2.4 Restart handling

This activity does not implement any forms of restart other than power-up reset a cold SW reset.[1]

18.2.5 Data Structures

18.2.5.1 Switch Configuration

Switch configuration is contained in either files switchConf.c or pmConf.c. File pmConf.c contains bit-masks to configure PM Diagnostics and Initialization. File switchConf.c contain the same bit-masks as pmConf.c as well as a bit-mask to configure SCU Diagnostics. Please refer to these files for more information.

18.2.5.2 Cell Data Structure

The cell data structure is used to house cells within the GPU and has the following field:

1. Cell IVC - a 16-bit number;
2. Cell Header - an array of five bytes;
3. Cell Body - an array of 12 32-bit words totalling 48 bytes.

18.2.5.3 Cell Encoding Lookup Table

This data structure is an array of 128 bytes which maps 7-bit patterns produced during cell encoding to 8-bit patterns which can be placed in the Core Memory. The 7-bit pattern will be the index into the array and valid Core Memory patterns will be the elements of the array. This data structure resides in the SCU HW interface module, ‘scuDrv.c’ and is primarily used by function ‘scuMap7To8()’.

1. Future switch software development may implement some or all of the other forms of restart outlined in Section 7.0 in the System Initialization FN.

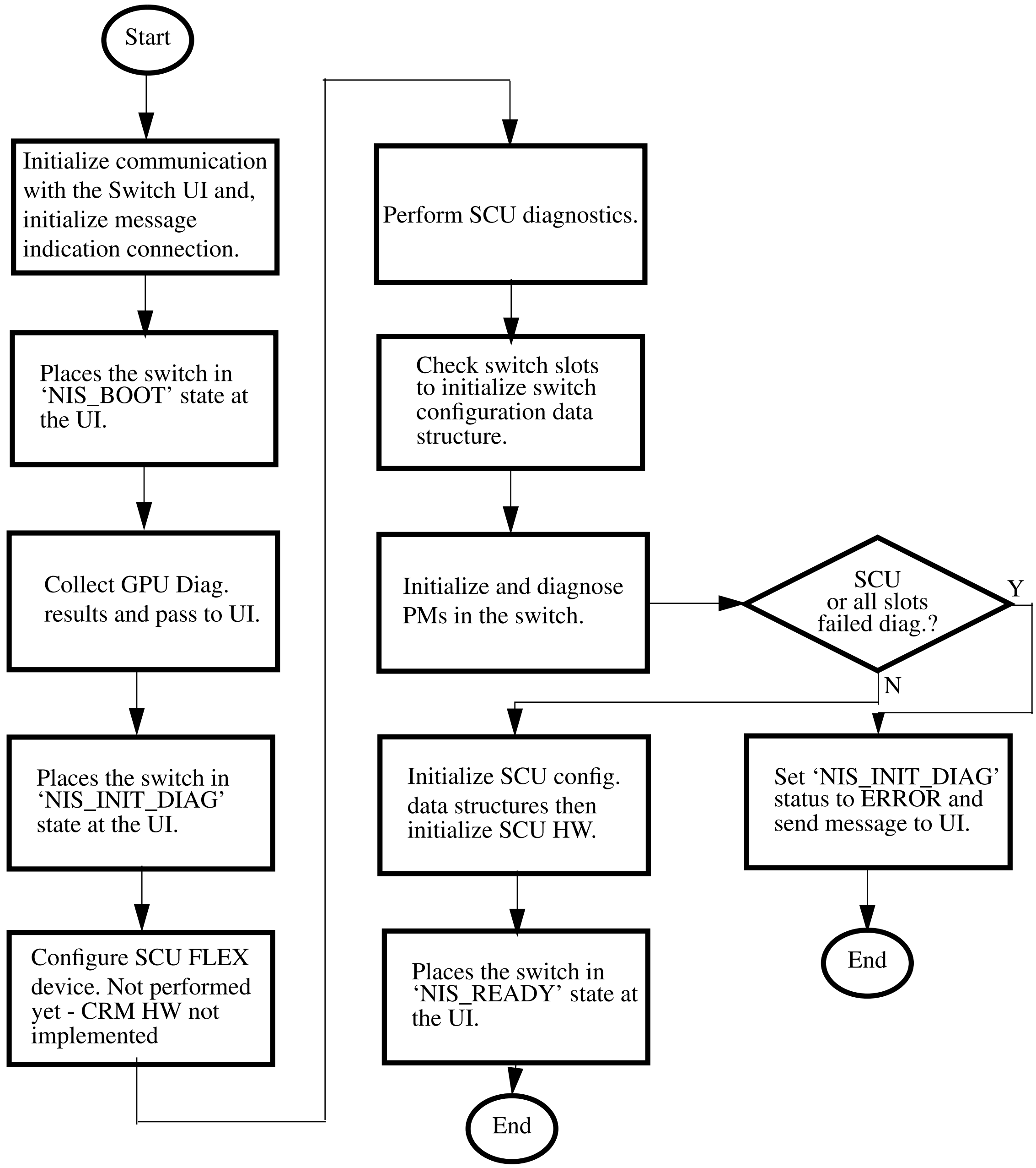


Figure 1 System Initialization Flowchart

18.2.2 Operating System

VxWorks is the real time operating system running on the GPU. During normal operation of the switch, it provides us with multi-task scheduling, context switching and mutual exclusion semaphores. During initialization, an Ethernet I/O driver provided by VxWorks is used to facilitate downloading software files from the switch workstation. There is no operating system in use on the PM.

System Initialization is responsible for bringing a switch through the Not_In_Service_Bootstrap (NIS_BOOT) and Not_in_Service_Initialization_Diagnostics (NIS_INIT_DIAG) States and leaving it in the Not_in_Service_Ready (NIS_READY State). NIS_BOOT performs GPU and VxWorks bootstrap as well as GPU Diagnostics while NIS_INIT_DIAG performs start-up diagnostics on the remaining switch hardware, software downloading and hardware configuration. Once System Initialization is complete the switch should be ready for the NIS_READY State which is the state in which switch control is turned over to the application software (SW)

Immediately after power up or a user initiated system cold reset, the Switch Control Module's (SCM's) General Processing Unit (GPU) begins booting out of an EEPROM residing on the GPU. After booting the GPU, a startup script can be run which initiates the downloading of all switch software (including all Port Module (PM) software) from the switch workstation placing it in GPU RAM. The script then invokes the System Initialization control SW which performs initialization and diagnostics on the SCU and all PMs. For more information see "System Initialization Flowchart" on page 43.

The process of System Initialization is controlled from the GPU by module 'sysInitCtl.c', the switch initialization controller module. This module invokes PM initialization control SW which controls PM initialization from the GPU. Once System Initialization is complete, 'sysInitCtl.c' Harvard application SW may be downloaded and invoked.

18.2 Detailed Description

18.2.1 System Initialization Flow

The flow chart in Fig. 1 illustrates the order in which events occur during System Initialization at a very high level. It is not intended to indicate all the status messages sent to the switch UI during System Initialization and it assumes that all diagnostics are performed. (HW diagnostics can actually be bypassed using the GPU's DIP switches - see "Function 'sysInitCtl'" on page 47) This flow chart basically governs the contents of controller module 'sysInitCtl.c'. For more information please refer to the particular sections describing these modules later in this document.

Design description (DD)

16.0 WARNING

Although the basic overall contents of this document are still relevant, it is possible that certain details are out of date. If in doubt, please contact the authors.

17.0 Issue/change control

Table 2Revision History

Issue	Date	Reason	Change	Author
0.0	September 16, 1993	Initial release for internal review	First draft	Gary Depelteau
1.0	October 22, 1993	Release for external review	Update	Gary Depelteau
1.1	December 9, 1993	Update	Added to 'mcUtil.c'	Gary Depelteau
2.0	December 13, 1994	Update	Major updating, changed all 'mc' to 'scu' or 'scm'	Gary Depelteau
3.0	May 4, 1995	Update	Added PM stuff	Ihab Mahna
3.1	May 9, 1995	Update	Hopefully final update near project end	Gary Depelteau

18.0 Design approach

18.1 Overview

The purpose of this activity is to provide a high level view of the functional requirements and design of System Initialization for the CreditNet Switch.

Appendix A - Software Download

All software, including GPU, PCC and Scheduler software, is downloaded from the workstation. PCC and Scheduler software is first downloaded to the GPU and then to the PCC. Scheduler software is then downloaded from the PCC to the Scheduler. Downloading will be carried out via shared memories shared between the GPU and each PCC and between each PCC and its corresponding Scheduler.

[7] Ken Grimble, BNR. “MasterCard Specification.” July 6, 1993.

[8] Shuo Huang, BNR. “Data Manager Specification.” Version 1-0.

[9] Dan Leonida, BNR. “Credit Manager.” Version 0-1.

[10] Robert Morris, Harvard. “Scheduler Specification.” Version 1-1. February 26, 1993.

[11] Margo Seltzer and Mike Smith, Harvard University.”Software Specification.” Version 0-2. April 13, 1993.

[12] John Vincent, BNR. “EPATH Specification.” Version.1-1.

[13] John Vincent, BNR. “IPATH Specification.” Version 1.4.

[14] Cliff Young, Harvard University. “Port Controller Specification.” Version 1-1. June 16,1993.

[] System SW/Users Guide???

PIF :Processor Bus Interface

PCU :Port Control Unit

PM :Port Module

PPU :Port Processing Unit

R_CADR :Read Cell Address

R_IVC :Read IVC (lines on the switch backplane)

RAM :Random Access Memory

SCM :Switch Control Module

SCU :Switch Control Unit

TBD :To Be Determined

TLG :Test Load Generator

VC :Virtual Connection

W_CADR :Write Cell Address (lines on the switch backplane)

W_IVC :Write IVC (lines on the switch backplane)

15.0 References

[1] The ATM Forum. “ATM User-Network Interface Specification.” Version 2.2. June 21, 1993.

[2] H.T. Kung, Harvard University; Alan Chapman, BNR. “The FCVC (Flow Controlled Virtual Channels) Proposal for ATM Networks.”Version 1.0. June 22, 1993.

[3] WIndRiver Systems, “VxWorks Programmer’s Guide 5.1.” December 1993.

Internal Documents:

[4] Melhem Chaar, BNR, “Design Specification - The Credit Network - MasterCard”, Feb. 15, 1994.

[5] Gary Depelteau, BNR, “GPU Diagnostics”, Feb. 16, 1994.

[6] Gary Depelteau, BNR, “SCU and Core Memory Diagnostics”, Feb. 16, 1994.

The first phase of the switch's development will only implement complete system initialization and various forms or subsets of initialization or reset will not be implemented until future phases of the switch.

13.0 Open Issues

5. What are the different degrees of initialization we should support? Are the following applicable:
   i. Warm restart - maybe bypass some/all diagnostics;
   ii. Initialize a PC that has just been replaced when its port already has some VCs defined and while the rest of the switch is running. This would be nice for robustness but is it needed in the first version of an experimental prototype switch;
   iii. Initialize a PC that has just been inserted into a previously empty slot. this should not interrupt the normal operation of the switch.

14.0 Glossary of definitions & abbreviations

ATM	:Asynchronous Transfer Mode
BNR	:Bell Northern Research
CDM	:Cell Departure Monitor
CIF	:Cell Interface
CRM	:Core Resource Manager
DALM	:Data Active List Manipulator
DP	:Dual Port
FIFO	:First-In First-Out
FLM	Free List Manager
GPU	:General Processor Unit
GUI	:Graphical User Interface
IDPROM	:Identification Programmable Read-Only Memory
IVC	:Internal Virtual Connection
LUT	:Look-Up Table

7.2 System initiated actions

When running the System initialization start-up script described in 5.2 "Switch Start-up Script/ Software Download" the entire initialization process is automated.

7.3 User initiated actions

A human user can initiate the process of System Initialization by powering up the switch when it is initially off or by entering an initialization command from the workstation. The user must also start up the Switch User Interface GUI on the switch workstation in order to see the status information sent from the switch.

8.0 Application interfaces

Not Applicable.

9.0 Restrictions/limitations

At first, System Initialization will only perform complete switch initialization following switch power up. It will not perform any of the forms of reset or restart described in issue 2 in section 7.0.

10.0 Optionality

Not Applicable.

11.0 Activity impact

This activity will impact all other activities associated with this project in the sense that it makes it possible to implement all other activities by initializing the switch's hardware and data structures and establishing an operating system or kernel on which other software layers can build.

11.1 Associated activities

The following is a list of possible associated activities:

1. Diagnostics;
2. Communication Interface;
3. User I/F on workstation (workstation commands may be used to initiate a certain form of initialization or reset);
4. Switch Maintenance (under the topics of shutdown/restart. PC reset, etc.).

12.0 Future enhancements

7.1 User Interface

In the early stages of System Initialization, during GPU/VxWorks Bootstrap, status information must be reported via the serial port on the GPU. For more information see "GPU/VxWorks Bootstrap" on page 19.

After GPU/VxWorks Bootstrap, when ethernet communication has been established, System Initialization status is reported to the Switch User Interface. For more information see "Switch User Interface" on page 34. Detailed status information is presented on a slot by slot basis in the form of a table seen below and high level status is presented in the box labelled “Switch Status”. For further details regarding the Switch User Interface please see [].

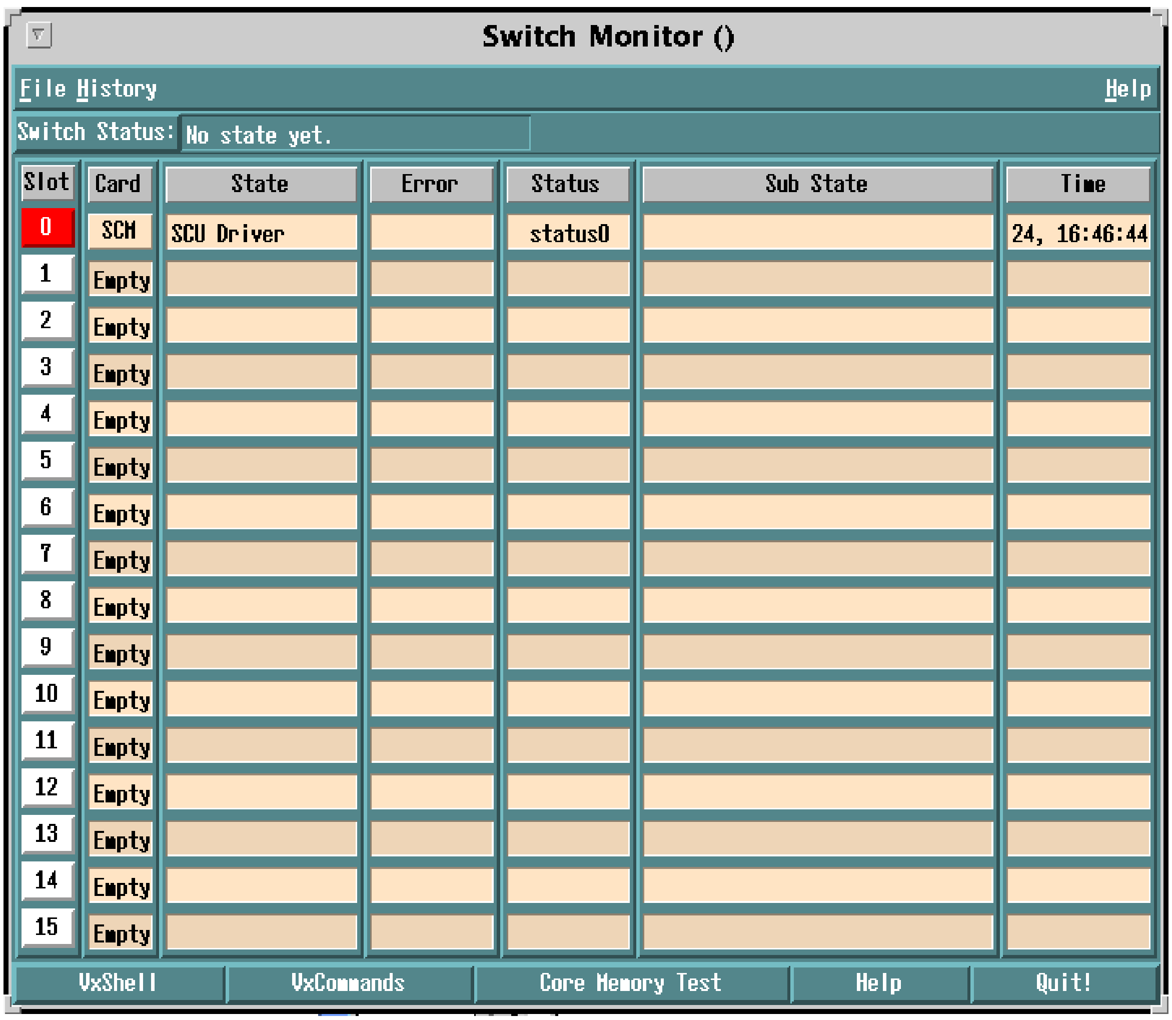


Figure 3 Switch User Interface

Primitive of PPU Tests		
Test Name	Attributes	Descriptions
Memory Test	A3f, A19f,A27f	This test verifies proper PPU access to the static memory, DPRAM and memories on PM hardware blocks.it insures integrity. In particular, it tests for stuck bits, leakage, address decoding errors and address and data line shorts. A series of Read and write operations are carried out in the PM memories in order to verify its integrity.
Semaphore Test	A4f, A5f	This test verifies a proper Semaphores “take” and “give” operations, also verifies simultaneous access to semaphores form the PPU and the GPU.
Interrupt Test	A8f, A9f, A12f, A10f, A11f, A14f, A25f, A26f, A32f, A33f	This test verifies the cause and action of all interrupts that interrupt the PPU, also covers disabling interrupts from triggering these actions.
WD Timer Test	A10f, A11f	This test verifies that WD timer resets the PPU in case of not feeding it, also check disable WD function.
RTC Timer Test	A25f, A26f	This test verifies that the RTC interrupts the PPU every 17ms also verifies disabling this clock from interrupting the PPU.
Broadcast Test	A15f, A16f, A17fA1f, A18f,	This test verifies that the GPU has a broadcast access to all existing PM(s)
FLASH Programming Test	A23f, A24f	This test verifies that the PPU could program and access the FLASH memory.
FLEX Programming Test	A34f-A40f	This test verifies that the PPU could configure and erase all FLEX devices also verifies the sanity of these devices.
Memory Access Test	A6f, A7f, A24f, A29f, A32f	This test performs some verifications specific to a memory blocks which is not covered by the “Memory Test” such as Quad read and write for static memory and DPRAM, Loopback and prefetch memory function.
Note: These tests can be used as part of the start-up diagnostics for the TUC Card		

6.0 Application interfaces

Not Applicable.

7.0 Associated activities

The following is a list of possible associated activities:

1 Core Memory Diagnostics (as part of System Initialization).

2. GPU diagnostics. [5]

A39f If nSTATUS_0 is equal to high value, this indicates that one of the devices in the first group has an operational or configuration error.

A40f If nSTATUS_1 is equal to high value, this indicates that one of the devices in the second group has an operational or configuration error.

5.6.4 Ingress Cell Path (ICP) Attributes

5.6.5 Engress Cell Path (ECP) Attributes

5.6.6 Cell Drop Activator (CDA) Attributes

5.6.7 Credit Protocol Manager (CPM) Attributes

5.6.8 Cell Output Scheduler Attributes

5.6.8.1 PPU to SCH DPRAM

A41f The PPU read from or write to the DPRAM 8Kx16bits.

A42f The PPU boots the Scheduler processor (cold reset) by asserting the SCH_RESET line, both the boot software and the IBR will reside in the SCH DPRAM memory mapped to the PPU processor.

A43f The PPU synchronize the scheduler i960 processor by making the scheduler software reads a memory location puts the scheduler in a number of wait state.

5.7 Test Cases

This is required to verify the design of the PM Card. The tests in this sections are designed with the following objectives:

- To help isolate hardware fault.
- To be as independent as possible.
- To be the foundation for other tests.

The sequence of the test based on testing the interfaces between the PPU and other components of the PM card. See “PPU Block Diagram” in the DD section of this document.

Primitive of PPU Tests

Test Name	Attributes	Descriptions
Processor Reset Test	A1f, A2f	This test verifies The PPU processor sanity, also checks processor configuration. After passing this test the PPU i960CA processor is ready to cary on its function.

5.6.3.7 PPU to HWIO Interface

A27f The PPU reads from or writes to any memory block on the PM hardware through the HWIO interface. Each memory block has a bus width and a number of wait states.

A28f The PPU switches the HWIO interface into a loopback mode, read and write operations in A26f will go through the HWIO and loopback data and address to another memory locations. This mode is only used to check the HWIO interface.

A29f The PPU prefetch data from any memory block on the PM by reading a shadow address[2], a number of cell cycles should pass before reading the real address to get the data. The number of cell cycles depends on which memory block you prefetch.

A30f When the PPU unmask the bus error interrupt any invalid read or write to a memory block on the PM through the HWIO, generates a bus error interrupt to the PPU.

A31f When the PPU masks the interrupt by setting the bus error mask bit, invalid read or write operations will not generate any bus error interrupts.

A32f Each of the PM hardware blocks contains error counter(s). When any of these counters overflows or reaches a certain threshold, the PM hardware block generates an interrupt to the PPU. To know which error counter causes that interrupt, check error counters values in that block.

A33f The PPU masks the error counters interrupts by setting a mask bit, error counters overflow will not cause any interrupt to the PPU.

5.6.3.8 PPU to FLEX devices Interface

This interface enables the PPU to program the in-circuit programmable FLEX devices.

A34f Erase any device in the first group which includes all even FLEX devices by initializing the nCONFIGx [3], DCLK_0 and DATA0_0 lines to (1,0,1). The nSTATUS_0 indicates if the erase process passed successfully or not.

A35f Erase any device in the second group which includes all odd FLEX devices by initializing the nCONFIGx, DCLK_1 and DATA0_1 lines to (1,0,1). The nSTATUS_1 indicates if the erase process passed successfully or not.

A36f Program the FLEX device in the first group as described in the [] Both nSTATUS_0 and CONFIG_DONE_0 indicates if the FLEX device programming is done successfully or not.

A37f Program the FLEX device in the second group as described in the [] Both nSTATUS_1 and CONFIG_DONE_1 indicates if the FLEX device programming is done successfully or not.

A38f If the FLEX configuration data is not sane, the CONFIG_DONE_x should be high.

---

[2]Shadow address is equal to the real address lower bits with the expiation that one of its higher bits is set to one.

---

[3]nCONFIGx is the nCONFIG line for the FLEX device number x for example nCONFIG1 is the nCONFIG line for the FLEX device number 1.

A9f The PPU post an IRQ interrupt to the GPU, the interrupt will not go through if the GPU interrupt level is set to a value higher than the PPU interrupt priority level.

A10f The GPU enables the NMI of the Watchdog Timer (WD)- There is a timer which counts to 1 second and must be written to by the PPU in order to verify processor sanity. If the processor does not write to the WD timer within 1 second, an NMI will be generated to the PPU. If the PPU still does not write to the WD timer within another second, the PPU will then be reset.

A11f The GPU disables the NMI of the WD Timer, If the WD is not fed the NMI will not be generated.

A12f The GPU interrupt the PPU by posting an IRQ to the PPU, the NMI service routine on the PPU will be requested upon posting this interrupt.

A13f The GPU able to read the contents of the ID PROM of an existing PM card.

A14f A bus error will be generated to the GPU upon reading an ID PROM of a non-existing PM card.

A15f In broadcast mode A1f, A2f resets all PM cards simultaneously

A16f In broadcast mode A3f only writes a value to same location in DPRAM in all PM cards.

A17f In broadcast mode A10f, A11f the GPU could enable and disable WD timers on all PM cards.

A18f In broadcast mode A12f the GPU could post an interrupt to all PM cards simultaneously.

5.6.3.4 PPU to SRAM Interface

A19f The PPU can write to and read from the static SRM 256K bytes memory.

5.6.3.5 PPU to FLASH memory Interface

A20f The PPU programs a byte on 512K FLASH memory by setting the FLASH to the programming mode and then writes a byte programming sequence, the byte programming sequence checks for programming success by polling the byte location value.

A21f The PPU reads a byte on the 512K FLASH (like any memory read operation).

A22f The PPU erases the whole FLASH memory by writing the erase sequence.

A23f The PPU programs the whole FLASH memory using A20f for all the 512K bytes of the FLASH memory.

A24f The PPU reads the whole FLASH memory using A21f for all the 512K bytes of the FLASH memory.

5.6.3.6 PPU to RTC (Real Time Counter) Interface

A25f The PPU enables the (RTC), it starts generating an interrupt to the PPU every 17m sec.

A26f The PPU masks RTC interrupts by setting the (RTC) mask bit, the RTC stops generating interrupts to the PPU.

3. Initialize and establish the Communication Interface with the General Processing Unit GPU via the DPRAM.
4. Configure hardware FLEX devices on the PM.
5. Execute a start-up-diagnostics on all Port Modules PM hardware blocks.
6. Re-Initialize all PM memories and hardware blocks preparing for the application software.
7. PM hardware blocks start-up in the proper functional order waiting for the PM Application Software to be loaded.
8. Download PM Application Software.
9. Upgrade the FLASH bootrom software.

This is a little note to remind you to initialize each port's BF thresholds in the CRM once the PM is initialized. For more information see "Core Resource Manager (CRM)" on page 24.

5.6.3.2 PM Diagnostics

5.6.3.3 PPU to GPU Interface

A1f The GPU boots the PPU processor by asserting the PPU_RESET line low for 1m sec and then assert it high. The boot software will reside on the pre-programmed FLASH bootrom mapped to the PPU processor. In this case the GPU sets a bit that direct the PPU to use the Initial Boot Record (IBR) part of the FLASH bootrom.

A2f The GPU boots the PPU processor by asserting the PPU_RESET line low for 1m sec, then asserts it high. The boot software will reside in the DPRAM memory mapped to the PPU processor. in this case the GPU sets a bit that direct the PPU to use the Initial Boot Record (IBR) part of the DPRAM memory.

A3f The PPU reads from and write to the DPRAM 8Kx16bits without corruptions.

A4f The PPU sets any of the 8 DPRAM semaphores to value 1 i.e take the semaphore. The GPU can not take any of the DPRAM semaphores.

A5f The PPU set any of the 8 DPRAM semaphores to value 0 i.e give the semaphore. The GPU can take all semaphores.

A6f The PPU can write to a DPRAM memory location, if the GPU reads the same location at the same time, the GPU hold off till the PPU write operation is done.

A7f The GPU can write to a DPRAM memory location, if the PPU reads the same location at the same time, the PPU hold off till the GPU write operation is done

A8f The PPU post an IRQ interrupt to the GPU, the interrupt will go through if its interrupt level is set to a value lower than the PPU interrupt priority level. The GPU detects which card posted that interrupt by checking the IRQ register.

56. CTC Cell Period Timer read;
57. CTC IT program;
58. CTC IT read;
59. SCU reset;

5.6.1.3.2 High-level Drivers

60. Read and write cells (one cell read, one cell write in the same CCB) from/to the Core;
61. Write a cell residing at a cell address in the Shared Buffer into the Core;
62. Read a cell at a CADR in the Core into the Shared Buffer;
63. Write a cell residing in GPU RAM into the Core Memory;
64. Read a cell residing in the Core Memory into GPU RAM;
65. SCU IVC setup. Sets up the R_IVC Lookup Table entry, W_IVC Lookup Table entry and DBM entry for one or a group of IVCs;
66. SCU IVC initialization for a single IVC;
67. SCU IVC initialization for a set of IVCs based on data passed in an array;

Note: This is not intended to be a complete list of SCU Driver functionality capable of fully supporting application software needs.

5.6.2 Core Memory

The Core Memory provides the SCM and PMs the ability to store and retrieve cells travelling through the switch.

5.6.2.1 Core Memory Diagnostics

Core Memory diagnostics performed during System Initialization are discussed in “SCU and Core Memory Diagnostics” [6].

5.6.2.2 Core Memory Initialization

There is no specific need to initialization the cell buffers in the Core Memory to contain specific data before using the switch. From a system point of view, the Core Memory can be considered initialized as long as all 16K CADRs (with the exception of the CADR used for idle cell exchange) are placed on the Free CADR List in the SCU’s FLM.

5.6.3 PM

5.6.3.1 PM Initialization

The following steps must be carried out in order to initialize the Port Module PM.

1. Port Processing Unit (PPU) Bootstrap.
2. Execute a start-up diagnostics on all PPU’s memories and interfaces to other blocks.

28. FLM Free CADR List Initialization of a set of CADRs based on data passed in an array;
29. FLM Free CADR List Initialization of a single CADR;
30. FLM Free CADR List Initialization of all 16K CADRs;
31. FLM halt;
32. FLM halt clear;
33. FLM reset (i.e. to initial state of empty);
34. FLM priming - pops the two CADRs at the head of the Free CADR List and places them in the two register feedback pipeline that lies between the FLM Free CADR List RAM and the Cell Control Bus (CCB) on the backplane;
35. CDM DBM setup - can setup a DBM and casting bit for one or a set of IVCs;
36. CDM DBM set - can set a DBM and casting bit for one IVC;
37. CDM DBM initialization - can initialize DBMs and casting bits for a set of IVCs based on data passed in an array;
38. CDM DBM capture - returns the DBM and casting bit for given IVC;
39. CDM wDBM capture - returns the wDBM and CM bit for given CADR;
40. CDM Chalkmark search enable;
41. CDM Chalkmark search disable;
42. CDM Chalkmark search reset;
43. CDM Current chalkmark set;
44. CDM Current chalkmark read;
45. CDM Control Register write;
46. CDM Control Register read;
47. CDM Status Register read;
48. CDM Stale CADR read;
49. CRM BF and RPI thresholds setup for a given port or set of ports;
50. CRM BF and RPI thresholds setup for a single port;
51. CRM BF and RPI thresholds setup for a set of ports based on data passed in an array;
52. CRM Port BF and RPI threshold read;
53. CRM global switch BF threshold set;
54. CRM global switch BF threshold read;
55. CRM port usage counter read;

4. Map an 8-bit byte from the Core to a 7-bit word (part of the cell decoding process);
5. 8-7 64-byte cell decode. Performs 8-7 decoding (the same as that done bye the CIF) on a 64-byte to produce an unencoded 56-byte cell.
6. CIF Control Register write;
7. CIF Control Register read;
8. CIF Status Register read;
9. CIF Write Poll - polls CIF Control Register until the CIF is no longer busy performing a write cycle;
10. CIF Read Poll - polls CIF Control Register until the CIF is no longer busy performing a read cycle;
11. CIF Read/Write Poll - polls CIF Control Register until the CIF is no longer busy performing a read or write cycle;
12. CIF W_IVC Lookup Table setup - can setup one or a set of IVCs based on data passed in an array;
13. CIF W_IVC Lookup Table initialization- can initialize a set of IVCs based on data passed in an array;
14. CIF W_IVC Lookup Table entry setup - can setup a single IVC;
15. CIF R_IVC Lookup Table setup - can setup one or a set of IVCs based on data passed in an array;
16. CIF R_IVC Lookup Table initialization- can initialize a set of IVCs based on data passed in an array;
17. CIF R_IVC Lookup Table entry setup - can setup a single IVC;
18. CIF high/low priority W_IVC FIFO data read;
19. CIF R_IVC FIFO read;
20. CIF high priority W_IVC FIFO status read;
21. CIF low priority W_IVC FIFO status read;
22. CIF R_IVC FIFO status read;
23. FLM CMBF read;
24. FLM Control register write;
25. FLM Control register read;
26. FLM Status register read;
27. FLM Free CADR List Initialization of a single CADR, all 16k or a subset of the 16k CADRs;

5.6.1.2.5 Cell Time Counter (CTC)

The CTC has a both a non-programmable real-time counter and a programmable Interval Timer (IT) which can start and stop based on programmed start and stop IVC conditions. Both the real-time counter and IT count in units of cell period times. The CTC has the following initialization requirements:

- IT start control register - initialize to start on a W_IVC equal to 0x1fff;
- IT stop control register - initialize to stop on a W_IVC equal to 0x1fff;

5.6.1.2.6 Processor Bus Interface (PIF)

The PIF provides an interface between the GPU's external bus and the SCU's internal address and data buses and the processor bus on the switch backplane. It also contains the registers below with the following initialization requirements:

- Program/test enable register - initialize as follows:
    - Disable Lattice programming and the effects of the Lattice programming register;
    - Disable Flex programming and the effects of the Flex programming register;
    - Disable SCU-PSEL bus access which disables programming the Core Memory Lattices from the GPU;
- Reset all PM and switch alarms interrupts via the Interrupt/Alarm Control register;
- Interrupt/Alarm Control register - initialize as follows:
    - disable the major alarm drive bit (turns on the major alarm lamp on the switch cabinet);
    - deasserts the interrupt reset bit;
    - deasserts the alarms mask bit;

5.6.1.2.7 Partial CRC Generator/Checker

TBD.

5.6.1.3 SCU Driver

The SCU Driver software provides a set of low level driver functions that let high level software manipulate SCU entities and prepare cells for the transfer to/from the Core via the Shared Buffer in the CIF. It is utilized by SCU Diagnostics, SCU Initialization Switch Integration Diagnostics and application software. SCU Driver software is required to provide the following functionality:

5.6.1.3.1 Low-level Drivers

1. Copy a 56-byte cell from RAM into the CIF's Shared Buffer at a certain cell address;
2. Copy a 56-byte cell from a cell address in the CIF's Shared Buffer to a RAM buffer;
3. Verify that a 16-bit pattern is a valid Core Memory word;

FLM and also asserts the W_CE_NULL signal);

5.6.1.2.3 Cell Departure Monitor (CDM)

The CDM issues the Free Address (FA) signal when a cell for a certain IVC has been transmitted by all the PMs that are supposed to transmit it. It can also be programmed to perform a chalkmark search to locate aging cells in the Core Memory. It has the following initialization requirements:

- Unicast chalkmark - initialize to 0;
- Multicast chalkmark - initialize to 0;
- Working Distribution Bitmap (wDBM) RAM (indexed by W_CADR) - initialize all DBMs to 0x0 and the chalkmarks to 0;
- Distribution Bitmap (DBM) RAM (indexed by IVC) - initialize the DBM to 0x0 and casting bit to unicast for all IVCs;
- Clear the CM search done and CM alarms bits in the CDM Status register;
- Resets the CM search address;
- CDM Control register - initialize as follows:
    - deasserts the CM search enable bit;
    - deasserts the clear CM search flag bit;
    - deasserts the CM address reset flag bit;
    - deasserts the CM alarm flag bit;

5.6.1.2.4 Core Resource Manager (CRM)

The CRM monitors the number of cells each port has in the Core Memory and asserts threshold status signals to a port when it exceeds programmed Buffer Fill (BF) and Return Path Idle (RPI) thresholds. It can also nullify a ports cell write cycle if it has exceeded it highest programmable threshold and artificially promote a port's usage count when a global BF threshold has be exceeded. It has the following initialization requirements:

- Initialize BF and RPI thresholds for the each port (slots 1 to 15) to 0. This causes a port's cell write cycles to be nulled. PM initialization is responsible for setting each PM's thresholds later in the initialization process once the PMs have been initialized. For more information see "PM Initialization" on page 28.;
- Initialize the BF and RPI thresholds for the SCU (slot 0) to the following:
    - Lower BF Threshold - 0x100;
    - Middle BF Threshold - 0x1000;
    - Upper BF Threshold - 0x3C00;
    - RPI Threshold - 0x3C00;
- Initialize global switch BF threshold to 0x32f0 (80% full);
- Initialize per port BF counters to 0.

5.6.1.2 SCU Initialization

The following sections outline the initialization requirements of the SCU hardware blocks and Core Memory. A large part of this involves initializing tables which configure IVCs to acceptable not-in-use values. (Harvard application SW is responsible for configuring all IVCs.) Please refer to Fig. 2 throughout this section.

5.6.1.2.1 Cell Interface (CIF)

The cell interface provides a cell data path between the GPU and Core Memory, selectively captures W_IVC, W_CADR and R_IVC data storing it in corresponding FIFOs and provides R_CSID and W_CSID to the other SCU blocks. It has the following initialization requirements:

- W_IVC Lookup Table - initialize the high and low priority W_IVC/W_CADR FIFO enable bits to disabled (1) and W_CSID to 0 for all 8K IVCs;
- R_IVC Lookup Table - initialize the R_IVC FIFO enable bit to disabled and R_CSID to 0 for all 8K IVCs;
- Initialize the Shared Buffer to have idle cells in the first and last locations and the rest of the cell buffers to be filled with 0x0;
- Reset the CIF high priority W_IVC FIFO, low priority W_IVC FIFO and R_IVC FIFO;
- Initialize control register with the following:
    - CRC8 generation/checking enabled;
    - Cell encoding/decoding enabled;
    - SCU Reset bit disabled;
    - CIF FIFO Reset bit disabled;
    - All other bits disabled;

5.6.1.2.2 Free List Manager (FLM)

The FLM manages the Free CADR List of unused Core Memory CADRs issuing W_CADRs during valid cell write cycles for the all slots in the switch to use when writing cells into the Core Memory. It also places freed up R_CADRs back in the Free CADR List and updates a Core Memory Buffer Fill (CMBF) count to reflect the number of cell buffers currently being used in the Core Memory. It has the following initialization requirements:

- Reset the FLM to initialize the CMBF count to zero and point the Free CADR List pointers to location 0 in the Free CADR List RAM;
- Initialize FLM Free CADR List to store pointers to all cell buffers in the Core Memory, except for those reserved for idle cells (or those which did not pass diagnostics);
- Prime the FLM to pop the first four CADRs off of the Free CADR List and place them in the two register feedback pipeline that lies between the FLM Free CADR List RAM and the Cell Control Bus (CCB) on the backplane;
- Clear the FLM halt condition. (Halt prevents CADRs from being popped out of the

5.5 Processor-Processor Communication Establishment

The GPU can communicate with each PPU via a Dual Port (DP) memory resident on each PM. Ihab will be responsible for implementing the software side of the protocol through which the GPU communicates with the PPUs. A small but fast subset of this protocol be used to download PPU software and PM initialization data and once this is carried out the complete GPU-PPU communication protocol can be initiated.

5.6 HW Block Diagnostics and Initialization

5.6.1 SCU

Figure 2 illustrates the basic block level architecture of the SCU and the locations of the various memories. registers and programmable devices requiring initialization and diagnostics. (M) denotes memory blocks, (R) denotes registers and (PD) denotes programmable devices. For further details regarding the functionality and design of the SCU please see [4].

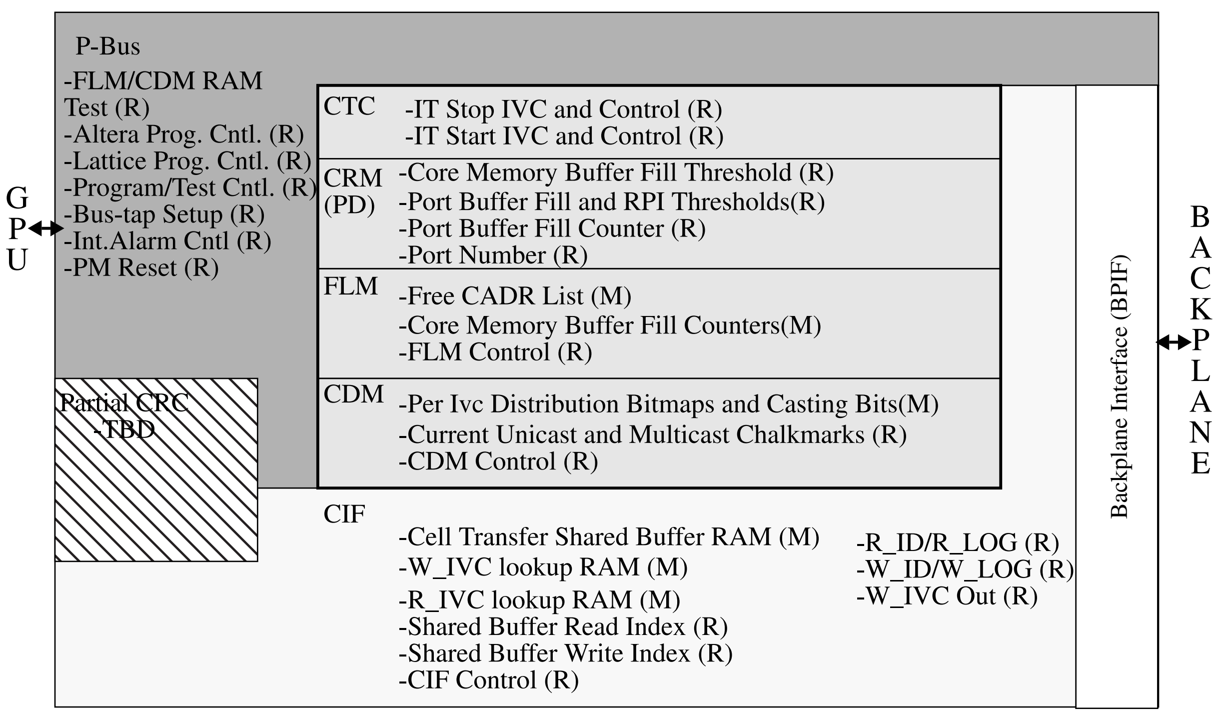


Figure 2 SCU Initialization Requirements

5.6.1.1 SCU Diagnostics

SCU diagnostics is required to verify the integrity of SCU hardware as outlined in "SCU and Core Memory Diagnostics" [5].

5.3 System Initialization Control

The System Initialization Control SW is responsible for ensuring that the following steps are carried out in order to initialize the switch [11]:

1. Initialization of communication with the Switch UI;
2. Placement of the switch in state ‘NIS_BOOT’;
3. Invoking GPU Diagnostics Control SW to report GPU Diagnostics results to the Switch UI [5];
4. Placement of the switch in state ‘NIS_INIT_DIAG’;
5. SCU FLEX configuration (not needed at the moment since the CRM HW has not been implemented);
6. Performing SCU and Core Memory diagnostics [6];
7. Performing PM Diagnostics and initialization(this includes establishing switch configuration);
8. Performing Switch Integration Diagnostics []; (This step has not been automated during System Initialization.)
9. Initialization of SCU hardware blocks for application SW;
10. Placement of the switch in state ‘NIS_READY’;

This SW is also responsible for sending high level messages to the UI regarding System Initialization progress during the above steps.

5.4 Switch Configuration Establishment

Switch Configuration Establishment SW must acquire the following information about a switch:

- Indicate whether a slot contains a PC, load generator card, performance monitor card or is empty;
- Indicate the correct version of each card installed in a slot if more than one version exists;
- Indicate the number of ports belonging to a PC and the speeds of those ports.[1]

The information should be acquired during initialization by polling all ports in the switch.

[1] Future versions of PCs may have 1, 4, 8, 12 or 16 ports as long as the total number of ports in a switch does not exceed 65 (counting the SCM as one port).

- Perform diagnostics on the GPU;
- Run switch start-up script;

During bootstrap, the status of diagnostics and the booting process is displayed via the GPU's serial port. Midway through bootstrap, a countdown is presented through the serial port giving the user opportunity to change the VxWorks booting parameters. Hitting any key at or before this countdown stops the booting process and brings up the boot command prompt. Entering 'c<return>' at this prompt provides the ability to change the boot parameters. The paragraph below is an example of the kind of dialogue presented via the serial port and the boot parameters used to boot VxWorks. For more information on VxWorks boot parameters and booting VxWorks see [3].

```
Press any key to stop autoboot...
 0
auto-booting...

boot device          : ei
processor number     : 0
host name            : bcars782b
filename             : /bnr/3rdparty/glan_sw/WindRiver/vw/config/gpu/vxWorks
inet on ethernet (e) : 47.53.32.4:ffff0000
host inet (h)        : 47.53.32.1
user (u)             : nobody
ftp password (pw)    : nobody
flags (f)            : 0x8
target name (tn)     : gpu1
startup script (s)   : /bnr/3rdparty/glan_sw/scripts/sysInit.scr
```

VxWorks boot parameters are stored in the GPU's Card ID FLASH.

5.2 Switch Start-up Script/Software Download

A switch start-up script, which can be run once GPU/VxWorks Bootstrap is complete, is responsible for the following:

1. Downloading (BNR developed) HW interface, initialization, diagnostics and Switch Operating System SW;
2. Invoking System Initialization Control SW (See "System Initialization Control" on page 21.);
3. Downloading (Harvard developed) application SW;
4. Invoking application control SW.

The HW interface, initialization, diagnostics and Switch Operating System SW downloaded in step 1. is linked together as several large object files prior to downloading. These object files are partitioned into GPU, SCM and Port Card specific SW. For more information see "Appendix A - Software Download" on page 39.

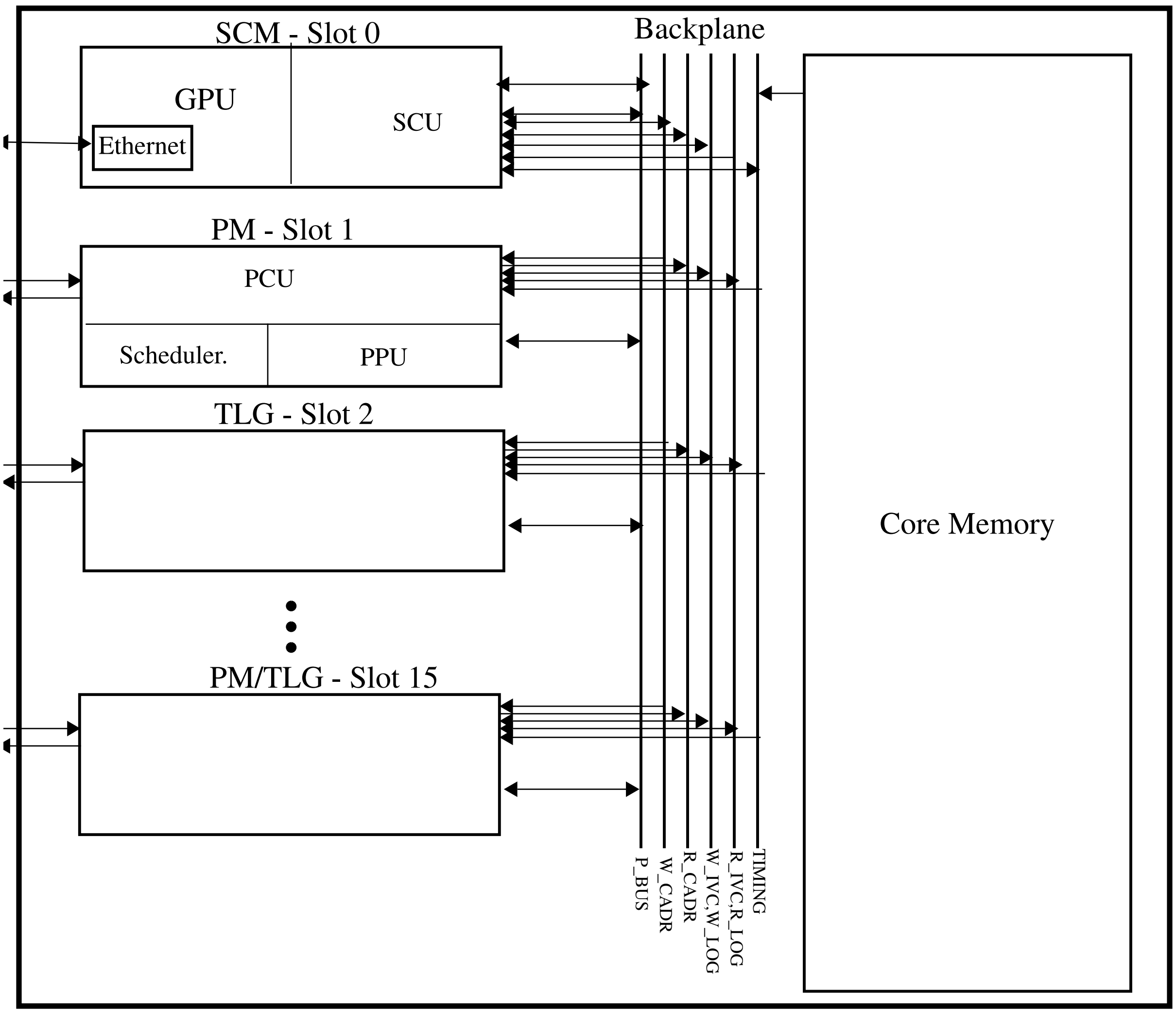


Figure 1CreditNet Switch Architecture

5.0 Functional specification

This section outlines the functional requirements of System Initialization. Please refer to the flowchart in the DD section of this design document for exact order in which initialization steps are executed.

5.1 GPU/VxWorks Bootstrap

GPU bootstrap has the following requirements:

- Configure the processor and leave it in a known state;
- Establish an Ethernet link to the switch workstation;
- Start up the VxWorks kernel;

3.0 Activity synopsis

The purpose of this activity is to provide a high level view of the functional requirements and design of System Initialization for the CreditNet Switch.

System Initialization is responsible for bringing a switch through the Not_In_Service_Bootstrap (NIS_BOOT) and Not_in_Service_Initialization_Diagnostics (NIS_INIT_DIAG) States and leaving it in the Not_in_Service_Ready (NIS_READY State. NIS_BOOT performs GPU and VxWorks bootstrap as well as GPU Diagnostics while NIS_INIT_DIAG performs start-up diagnostics on the remaining switch hardware (HW), software downloading and HW configuration. Once System Initialization is complete the switch should be ready for the NIS_READY State which is the state in which switch control is turned over to the application software (SW)

There are also subsets of System Initialization, such as resetting a single portcard, which may also be supported during later phases of switch development. This activity should attempt to partition software in such a way to allow maximum software reuse while implementing the various forms of initialization and reset.

4.0 Overview

Many functions must be carried out when the CreditNet Switch is powered up in order to ensure that it is capable of supporting virtual connections. These functions can be grouped as follows:

- GPU/VxWorks Bootstrap (including workstation link establishment and VxWorks initialization);
- Software download;
- Switch configuration establishment;
- Processor to processor communication establishment;
- Hardware block diagnostics and initialization;
- Status reporting to the Switch User Interface (UI).

Figure 1 illustrates the general architecture of the CreditNet Switch and should be referred to throughout the course of this document. For more information see "CreditNet Switch Architecture" on page 19. The three different cards identified in the figure are the Switch Control Module (SCM), Port Module (PM) and Test Load Generator (TLG). An SCM is further broken down into the Switch Control Unit (SCU) and the General Processing Unit (GPU). The PM is divided into the Port Processor Unit (PPU), Port Control Unit (PCU) and Scheduler. The blocks labelled “SCU” and “PCU” are expanded on in future diagrams.

Functional description (FN)

1.0 WARNING

Although the basic overall contents of this document are still relevant, it is possible that certain details are out of date. If in doubt, please contact the authors.

2.0 Issue/change control

Table 1: Revision History

Issue	Date	Reason	Change	Author
0.0	July 22, 1993	Initial release for internal review		Gary Depelteau
1.0	August 11, 1993	Release for external review		Gary Depelteau
1.1	September 1, 1193	Update		Gary Depelteau
1.2	Feb. 22, 1994	Update	Changed MC to SCM, MCP to GPU and updated SCM (SCU) initialization	Gary Depelteau
2.0	September 27, 1994	Update	Major updating and restructuring	Gary Depelteau
3.0	May 4, 1995	Update	Fill in some sections specific to the PMs	Ihab Mahna
3.1	May 9, 1995	Update	Hopefully final update near project end.	Fary Depelteau

Functional description (FN) 17

CreditNet Switch System Initialization

Version 3.1

Gary Depelteau

‘depeltea@bnr.ca’

Ihab Mahna

‘ihab@bnr.ca’

This Document

This document outlines the functional requirements, design description and man-machine interface required for System Initialization for the CreditNet ATM switch currently being developed by Harvard University and Bell-Northern Research Limited.

- Bell Northern Research
- Harvard University
-
-

Security Notice