UniPHY External Memory Interface Debug Toolkit 13

2013.12.16

emi_rm_011

The EMIF Toolkit lets you diagnose and debug calibration problems and produce margining reports for your external memory interface.

The toolkit is compatible with UniPHY-based external memory interfaces that use the Nios II-based sequencer, with toolkit communication enabled. Toolkit communication is on by default in versions 10.1 and 11.0 of UniPHY IP; for version 11.1 and later, toolkit communication is on whenever debugging is enabled on the Diagnostics tab of the IP core interface.

The EMIF Toolkit can communicate with several different memory interfaces on the same device, but can communicate with only one memory device at a time.

Architecture

The EMIF toolkit provides a graphical user interface for communication with connections.

All functions provided in the toolkit are also available directly from the `quartus_sh` TCL shell, through the `external_memif_toolkit` TCL package. The availablity of TCL support allows you to create scripts to run automatically from TCL. You can find information about specific TCL commands by running `help -pkg external_memif_toolkit` from the `quartus_sh` TCL shell.

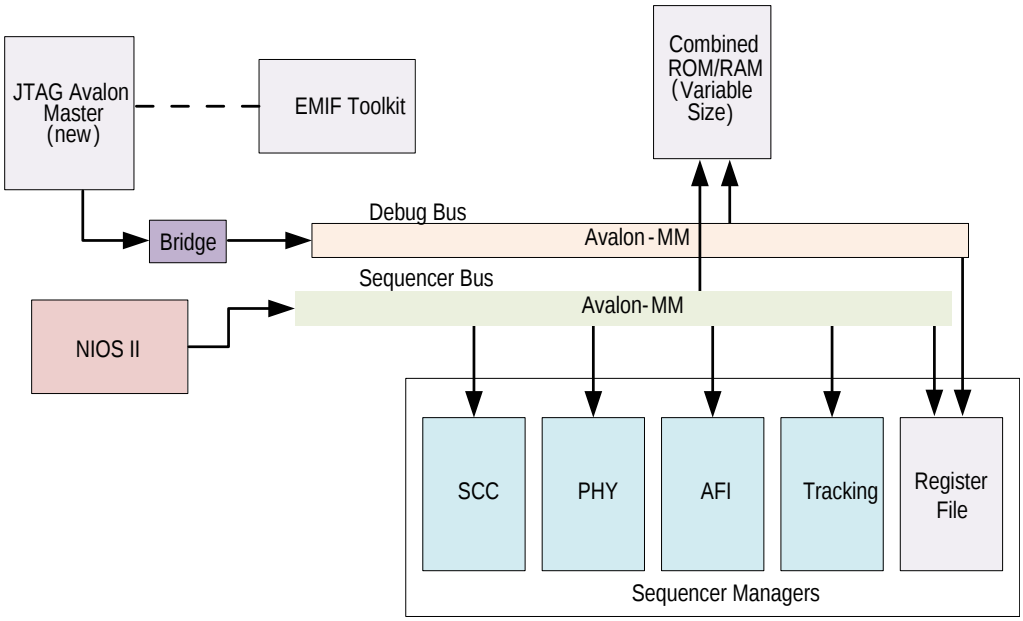
If you want, you can begin interacting with the toolkit through the GUI, and later automate your workflow by creating TCL scripts. The toolkit GUI records a history of the commands that you run. You can see the command history on the History tab in the toolkit GUI.

Communication

Communication between the EMIF Toolkit and external memory interface connections varies, depending on the connection type and version. In versions 10.1 and 11.0 of the EMIF IP, communication is achieved using direct communication to the Nios II-based sequencer. In version 11.1 and later, communication is achieved using a JTAG Avalon-MM master attached to the sequencer bus.

The following figure shows the structure of UniPHY-based IP version 11.1 and later, with JTAG Avalon-MM master attached to sequencer bus masters.

Figure 13-1: UniPHY IP Version 11.1 and Later, with JTAG Avalon-MM Master



Calibration and Report Generation

The EMIF Toolkit uses calibration differently, depending on the version of UniPHY-based interface in use. For versions 10.1 and 11.0 UniPHY-based interfaces, the EMIF Toolkit causes the memory interface to calibrate several times, to produce the data from which the toolkit generates its reports. In version 11.1 and later, report data is generated during calibration, without need to repeat calibration. For version 11.1 and later, generated reports reflect the result of the previous calibration, without need to recalibrate unless you choose to do so.

Setup and Use

Before using the EMIF Toolkit, you should compile your design and program the target device with the resulting SRAM Object File (**. sof**). For designs compiled in the Quartus II software version 12.0 or earlier, debugging information is contained in the JTAG Debugging Information file (**.jdi**); however, for designs compiled in the Quartus II software version 12.1 or later, all debugging information resides in the **.sof** file.

You can run the toolkit using all your project files, or using only the Quartus II Project File (**.qpf**), Quartus II Settings File (**.qsf**), and **.sof** file; the **.jdi** file is also required for designs compiled prior to version 12.1. To ensure that all debugging information is correctly synchronized for designs compiled prior to version 12.1, ensure that the **.sof** and **.jdi** files that you used are generated during the same run of the Quartus II Assembler.

After you have programmed the target device, you can run the EMIF Toolkit and open your project. You can then use the toolkit to create connections to the external memory interface.

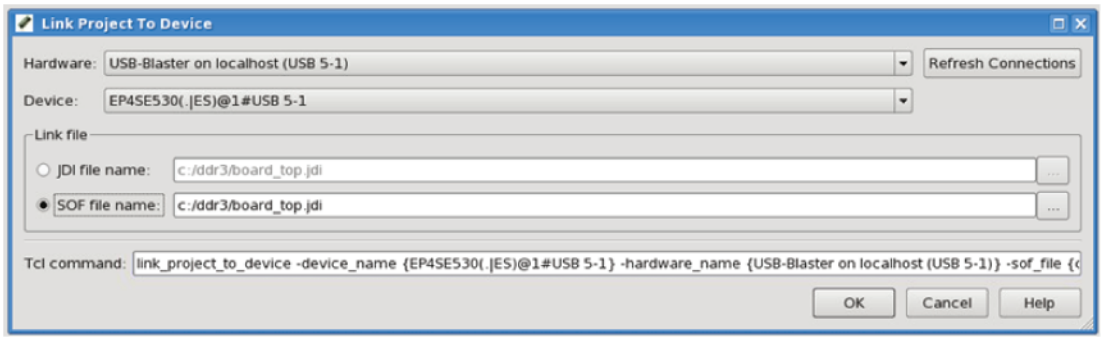
General Workflow

To use the EMIF Toolkit, you must link your compiled project to a device, and create a communication channel to the connection that you want to examine.

Linking the Project to a Device

1. To launch the toolkit, select External Memory Interface Toolkit from the Tools menu in the Quartus II software.
2. After you have launched the toolkit, open your project and click the **Initialize connections** task in the **Tasks** window, to initialize a list of all known connections.
3. To link your project to a specific device on specific hardware, perform the following steps:
   a. Click the **Link Project to Device** task in the **Tasks** window.
   b. Select the desired hardware from the **Hardware** dropdown menu in the **Link Project to Device** dialog box.
   c. Select the desired device on the hardware from the **Device** dropdown menu in the **Link Project to Device** dialog box.
   d. Select the correct **Link file type**, depending on the version of Quartus II software in which your design was compiled:
      - If your design was compiled in the Quartus II software version 12.0 or earlier, select **JDI** as the **Link file type**, verify that the **.jdi** file is correct for your **.sof** file, and click **Ok**.
      - If your design was compiled in the Quartus II software version 12.1 or later, select **SOF** as the **Link file type**, verify that the **.sof** file is correct for your programmed device, and click **Ok.**

Figure 13-2: Link Project to Device Dialog Box



When you link your project to the device, the toolkit verifies all connections on the device against the information in the JDI or SOF file, as appropriate. If the toolkit detects any mismatch between the JDI file and the device connections, an error message is displayed.

For designs compiled using the Quartus II software version 12.1 or later, the SOF file contains a design hash to ensure the SOF file used to program the device matches the SOF file specified for linking to a project. If the hash does not match, an error message appears.

If the toolkit successfully verifies all connections, it then attempts to determine the connection type for each connection. Connections of a known type are listed in the Linked Connections report, and are available for the toolkit to use.

Establishing Communication to Connections

After you have completed linking the project, you can establish communication to the connections

1. In the Tasks window,

- Click **Establish Memory Interface Connection** to create a connection to the external memory interface.
- Click **Establish Efficiency Monitor Connection** to create a connection to the efficiency monitor.

2. To create a communication channel to a connection, select the desired connection from the displayed pulldown menu of connections, and click **Ok**. The toolkit establishes a communication channel to the connection, creates a report folder for the connection, and creates a folder of tasks for the connection.

   Note: By default, the connection and the reports and tasks folders are named according to the hierarchy path of the connection. If you want, you can specify a different name for the connection and its folders.

3. You can run any of the tasks in the folder for the connection; any resulting reports appear in the reports folder for the connection.

Reports

The toolkit can generate a variety of reports, including summary, calibration, and margining reports for external memory interface connections. To generate a supported type of report for a connection, you run the associated task in the tasks folder for that connection.

Summary Report

The Summary Report provides an overview of the memory interface; it consists of the following tables:

- Summary table. Provides a high-level summary of calibration results. This table lists details about the connection, IP version, IP protocol, and basic calibration results, including calibration failures. This table also lists the estimated average read and write data valid windows, and the calibrated read and write latencies.
- Interface Details table. Provides details about the parameterization of the memory IP. This table allows you to verify that the parameters in use match the actual memory device in use.
- Groups Masked from Calibration table. Lists any groups that were masked from calibration when calibration occurred. Masked groups are ignored during calibration.
- Ranks Masked from Calibration tables (DDR2 and DDR3 only). Lists any ranks that were masked from calibration when calibration occurred. Masked ranks are ignored during calibration.

Calibration Report

The Calibration Report provides detailed information about the margins observed before and after calibration, and the settings applied to the memory interface during calibration; it consists of the following tables:

- Per DQS Group Calibration table. Lists calibration results for each group. If a group fails calibration, this table also lists the reason for the failure.

  Note: If a group fails calibration, the calibration routine skips all remaining groups. You can deactivate this behaviour by running the **Enable Calibration for All Groups On Failure** command in the toolkit.

- DQ Pin Margins Observed Before Calibration table. Lists the DQ pin margins observed before calibration occurs. You can refer to this table to see the per-bit skews resulting from the specific silicon and board that you are using.
- DQS Group Margins Observed During Calibration table. Lists the DQS group margins observed during calibration.
- DQ Pin Settings After Calibration and DQS Group Settings After Calibration table. Lists the settings made to all dynamically controllable parts of the memory interface as a result of calibration. You can refer to this table to see the modifications made by the calibration algorithm.

Margin Report

The Margin Report lists the post-calibration margins for each DQ and data mask pin, keeping all other pin settings constant; it consists of the following tables:

- DQ Pin Post Calibration Margins table. Lists the margin data in tabular format.
- Read Data Valid Windows report. Shows read data valid windows in graphical format.
- Write Data Valid Windows report. Shows write data valid windows in graphical format.

Operational Considerations

Some features and considerations are of interest in particular situations.

Specifying a Particular JDI File

Correct operation of the EMIF Toolkit depends on the correct JDI being used when linking the project to the device. The JDI file is produced by the Quartus II Assembler, and contains a list of all system level debug nodes and their heirarchy path names. If the default **.jdi** file name is incorrect for your project, you must specify the correct **.jdi** file. The **.jdi** file is supplied during the link-project-to-device step, where the ***revision_name*.jdi** file in the project directory is used by default. To supply an alternative **.jdi** file, click on the ellipse then select the correct **.jdi** file.

PLL Status

When connecting to DDR-based external memory interface connections, the PLL status appears in the Establish Connection dialog box when the IP is generated to use the CSR controller port, allowing you to immediately see whether the PLL status is locked. If the PLL is not locked, no communication can occur until the PLL becomes locked and the memory interface reset is deasserted.

When you are linking your project to a device, an error message will occur if the toolkit detects that a JTAG Avalon-MM master has no clock running. You can run the **Reindex Connections** task to have the toolkit rescan for connections and update the status and type of found connections in the Linked Connections report.

Margining Reports

The EMIF Toolkit can display margining information showing the post-calibration data-valid windows for reads and writes. Margining information is determined by individually modifying the input and output delay chains for each data and strobe/clock pin to determine the working region. The toolkit can display margining data in both tabular and hierarchial formats.

Group Masks

To aid in debugging your external memory interface, the EMIF Toolkit allows you to mask individual groups and ranks from calibration. Masked groups and ranks are skipped during the calibration process, meaning that only unmasked groups and ranks are included in calibration. Subsequent mask operations overwrite any previous masks.

Note:

For information about calibration stages, refer to *UniPHY Calibration Stages* in the *Functional Description - UniPHY* chapter.

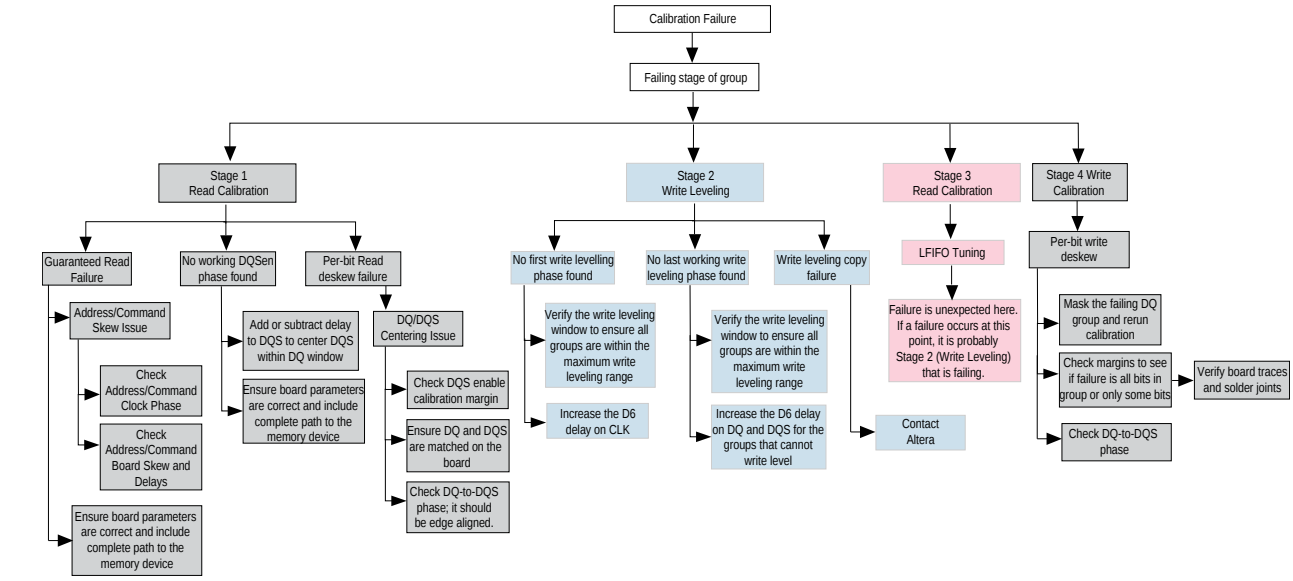
Related Information

Functional Description - UniPHY

Troubleshooting

In the event of calibration failure, refer to the following figure to assist in troubleshooting your design. Calibration results and failing stages are available through the external memory interface toolkit.

Figure 13-3: Debugging Tips



EMIF On-Chip Debug Toolkit

The EMIF On-Chip Debug Toolkit allows user logic to access the same calibration data used by the EMIF Toolkit, and allows user logic to send commands to the sequencer. You can use the EMIF On-Chip Debug

Toolkit to access calibration data for your design and to send commands to the sequencer just as the EMIF Toolkit would. The following information is available:

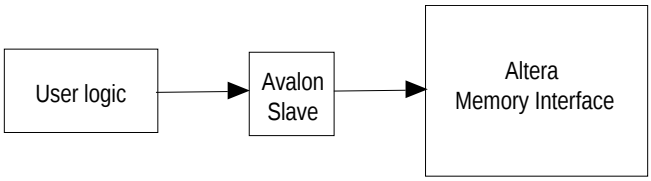
- Pass/fail status for each DQS group
- Read and write data valid windows for each group

In addition, user logic can request the following commands from the sequencer:

- Destructive recalibration of all groups
- Masking of groups and ranks
- Generation of per-DQ pin margining data as part of calibration

The user logic communicates through an Avalon-MM slave interface as shown below.

Figure 13-4: User Logic Access



Access Protocol

The EMIF On-Chip Debug Toolkit provides access to calibration data through an Avalon-MM slave interface. To send a command to the sequencer, user logic sends a command code to the command space in sequencer memory. The sequencer polls the command space for new commands after each group completes calibration, and continuously after overall calibration has completed.

The communication protocol to send commands from user logic to the sequencer uses a multistep handshake with a data structure as shown below, and an algorithm as shown in the figure which follows.

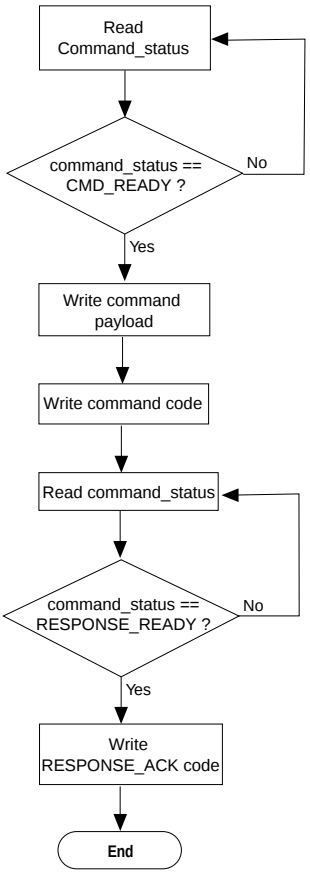
```
typedef struct_debug_data_struct {
...
// Command interaction
alt_u32 requested_command;
alt_u32 command_status;
alt_u32 command_parameters[COMMAND_PARAM_WORDS];...
}
```

To send a command to the sequencer, user logic must first poll the `command_status` word for a value of `TCLDBG_TX_STATUS_CMD_READY`, which indicates that the sequencer is ready to accept commands. When the sequencer is ready to accept commands, user logic must write the command parameters into `command_parameters`, and then write the command code into `requested_command`.

The sequencer detects the command code and replaces `command_status` with `TCLDBG_TX_STATUS_CMD_EXE`, to indicate that it is processing the command. When the sequencer has finished running the command, it sets `command_status` to `TCLDBG_TX_STATUS_RESPONSE_READY` to indicate that the result of the command is available to be read. (If the sequencer rejects the requested command as illegal, it sets `command_status` to `TCLDBG_TX_STATUS_ILLEGAL_CMD`.)

User logic acknowledges completion of the command by writing `TCLDBG_CMD_RESPONSE_ACK` to `requested_command`. The sequencer responds by setting `command_status` back to `STATUS_CMD_READY`. (If an illegal command is received, it must be cleared using `CMD_RESPONSE_ACK`.)

Figure 13-5: Debugging Algorithm Flowchart



Command Codes Reference

The following table lists the supported command codes for the On-Chip Debug Toolkit.

Table 13-1: Supported Command Codes

Command	Parameters	Description
TCLDBG_RUN_MEM_CALIBRATE	None	Runs the calibration routine.
TCLDBG_MARK_ALL_DQS_GROUPS_AS_VALID	None	Marks all groups as valid for calibration.
TCLDBG_MARK_GROUP_AS_SKIP	Group to skip	Mark the specified group to be skipped by calibration.
TCLDBG_MARK_ALL_RANKS_AS_VALID	None	Mark all ranks as valid for calibration
TCLDBG_MARK_RANK_AS_SKIP	Rank to skip	Mark the specified rank to be skipped by calibration.
TCLDBG_ENABLE_MARGIN_REPORT	None	Enables generation of the margin report.

Header Files

The UniPHY-based external memory interface IP generates header files which identify the debug data structures and memory locations used with the EMIF On-Chip Debug Toolkit. You should refer to these header files for information required for use with your core user logic. It is highly recommended to use a software component (such as a Nios II processor) to access the calibration debug data.

The header files are unique to your IP parameterization and version, therefore you must ensure that you are referring to the correct version of header for your design. The names of the header files are: **core_debug.h** and **core_debug_defines.h**.

Generating UniPHY IP With the Debug Port

The following steps summarize the procedure for implementing your IP with the EMIF On-Chip Debug Toolkit enabled.

1. Start the Quartus II software and generate a new external memory interface with UniPHY. For QDR II and RLDRAM II protocols, ensure that sequencer optimization is set to **Performance** (for Nios II-based sequencer).
2. On the **Diagnostics** tab of the parameter editor, turn on **Enable EMIF On-Chip Debug Toolkit**.
3. Ensure that the **EMIF On-Chip Debug Toolkit interface type** is set to **Avalon-MM Slave**.
4. Click **Finish** to generate your IP.
5. Find the Avalon interface in the top-level generated file. Connect this interface to your debug component.

```
input  wire [19:0] seq_debug_addr,         //    seq_debug.address
input  wire        seq_debug_read_req,     //        .read
output wire [31:0] seq_debug_rdata,        //        .readdata
input  wire        seq_debug_write_req,    //        .write
input  wire [31:0] seq_debug_wdata,        //        .writedata
output wire        seq_debug_waitrequest, //        .waitrequest
input  wire [3:0]  seq_debug_be,           //        .byteenable
output wire        seq_debug_rdata_valid   //        .readdatavalid
```

6. Find the **core_debug.h** and **core_debug_defines.h** header files in ***<design_name>/<design_name>*_s0_software** and include these files in your debug component code.
7. Write your debug component using the supported command codes, to read and write to the Avalon-MM interface.

The debug data structure resides at the memory address `SEQ_CORE_DEBUG_BASE`, which is defined in the **core_debug_defines.h** header file.

Example C Code for Accessing Debug Data

A typical use of the EMIF On-Chip Debug Toolkit might be to recalibrate the external memory interface, and then access the reports directly using the `summary_report_ptr`, `cal_report_ptr`, and `margin_report_ptr` pointers, which are part of the debug data structure.

The following code sample illustrates:

```
/*
 * DDR3 UniPHY sequencer core access example
*/
```

```
#include <stdio.h>
#include <unistd.h>
#include <io.h>
#include "core_debug_defines.h"

int send_command(volatile debug_data_t* debug_data_ptr, int command,
int args[], int num_args)
{
volatile int i, response;

// Wait until command_status is ready
do {
  response = IORD_32DIRECT(&(debug_data_ptr->command_status), 0);
} while(response != TCLDBG_TX_STATUS_CMD_READY);

// Load arguments
if(num_args > COMMAND_PARAM_WORDS)
{
// Too many arguments
return 0;
}
for(i = 0; i < num_args; i++)
{
IOWR_32DIRECT(&(debug_data_ptr->command_parameters[i]), 0, args[i]);
}
// Send command code
IOWR_32DIRECT(&(debug_data_ptr->requested_command), 0, command);
// Wait for acknowledgment
do {
  response = IORD_32DIRECT(&(debug_data_ptr->command_status), 0);
} while(response != TCLDBG_TX_STATUS_RESPOSE_READY && response !=
TCLDBG_TX_STATUS_ILLEGAL_CMD);
// Acknowledge response
IOWR_32DIRECT(&(debug_data_ptr->requested_command), 0,
TCLDBG_CMD_RESPONSE_ACK);
// Return 1 on success, 0 on illegal command
return (response != TCLDBG_TX_STATUS_ILLEGAL_CMD);
}
int main()
{
  volatile debug_data_t* my_debug_data_ptr;
  volatile debug_summary_report_t* my_summary_report_ptr;
  volatile debug_cal_report_t* my_cal_report_ptr;
  volatile debug_margin_report_t* my_margin_report_ptr;
volatile debug_cal_observed_dq_margins_t* cal_observed_dq_margins_ptr;

  int i, j, size;
  int args[COMMAND_PARAM_WORDS];
  // Initialize pointers to the debug reports
  my_debug_data_ptr = (debug_data_t*)SEQ_CORE_DEBUG_BASE;
  my_summary_report_ptr =
(debug_summary_report_t*)(IORD_32DIRECT(&(my_debug_data_ptr-
>summary_report_ptr), 0));
  my_cal_report_ptr =
```

```
(debug_cal_report_t*)(IORD_32DIRECT(&(my_debug_data_ptr-
>cal_report_ptr), 0));
  my_margin_report_ptr =
(debug_margin_report_t*)(IORD_32DIRECT(&(my_debug_data_ptr-
>margin_report_ptr), 0));

  // Activate all groups and ranks
  send_command(my_debug_data_ptr, TCLDBG_MARK_ALL_DQS_GROUPS_AS_VALID,
 0, 0);
  send_command(my_debug_data_ptr, TCLDBG_MARK_ALL_RANKS_AS_VALID, 0,
0);
  send_command(my_debug_data_ptr, TCLDBG_ENABLE_MARGIN_REPORT, 0, 0);

  // Mask group 4
args[0] = 4;
  send_command(my_debug_data_ptr, TCLDBG_MARK_GROUP_AS_SKIP, args, 1);

  send_command(my_debug_data_ptr, TCLDBG_RUN_MEM_CALIBRATE, 0, 0);

  // SUMMARY
  printf("SUMMARY REPORT\n");
  printf("mem_address_width: %u\n",
IORD_32DIRECT(&(my_summary_report_ptr->mem_address_width), 0));
  printf("mem_bank_width: %u\n", IORD_32DIRECT(&(my_summary_report_ptr-
>mem_bank_width), 0));
  // etc...

  // CAL REPORT
  printf("CALIBRATION REPORT\n");
  // DQ read margins
  for(i = 0; i < RW_MGR_MEM_DATA_WIDTH; i++)
  {
  cal_observed_dq_margins_ptr = &(my_cal_report_ptr-
>cal_dq_in_margins[i]);
  printf("0x%x DQ %d Read Margin (taps): -%d : %d\n", (unsigned
int)cal_observed_dq_margins_ptr, i,
  IORD_32DIRECT(&(cal_observed_dq_margins_ptr->left_edge), 0),
  IORD_32DIRECT(&(cal_observed_dq_margins_ptr->right_edge), 0));
  }
  // etc...
  return 0;
}
```

Debug Report for Arria V and Cyclone V SoC Devices

The UniPHY External Memory Interface Debug Toolkit and EMIF On-Chip Debug Toolkit do not work with Arria V and Cyclone V SoC devices. Debugging information for Arria V and Cyclone V SoC devices is available by enabling a debug output report, which contains similar information.

Enabling the Debug Report for Arria V and Cyclone V SoC Devices

To enable a debug report for Arria V or Cyclone V SoC devices, perform the following steps:

1. Open the **<*design_name*>/hps_isw_handoff/sequencer_defines.h** file in a text editor.
2. In the **sequencer_defines.h** file, locate the following line: `#define RUNTIME_CAL_REPORT 0`
3. Change `#define RUNTIME_CAL_REPORT 0` to `#define RUNTIME_CAL_REPORT 1`, and save the file.
4. Generate the board support package (BSP) with semihosting enabled, or with UART output.

The system will now generate the debugging report as part of the calibration process.

Document Revision History

Date	Version	Changes
December 2013	2013.12.16	Maintenance release.
November 2012	2.2	• Changes to *Setup and Use* and *General Workflow* sections. • Added *EMIF On-Chip Debug Toolkit* section • Changed chapter number from 11 to 13.
August 2012	2.1	Added table of debugging tips.
June 2012	2.0	• Revised content for new UnIPHY EMIF Toolkit. • Added Feedback icon.
November 2011	1.0	Harvested 11.0 *DDR2 and DDR3 SDRAM Controller with UniPHY EMIF Toolkit* content.