

This chapter describes the process of debugging hardware and the tools to debug any external memory interface. The concepts discussed can be applied to any IP but focus on the debug of issues using the Altera® DDR, DDR2, DDR3, QDR II, QDR II+, and RLDRAM II IP.

Increases in external memory interface frequency results in the following issues that increase the challenges of debugging interfaces:

- More complex memory protocols
- Increased features and functionality
- More critical timing
- Increased complexity of calibration algorithm

Before the in-depth debugging of any issue, gather and confirm all information regarding the issue.

Memory IP Debugging Issues

Debug issues may not be directly related to interface operation. Issues can also arise at the Quartus® II Fitter stage, or complex designs may have timing analysis issues.

Memory debugging issues can be categorized as follows:

- Resource and Planning Issues
- Interface Configuration Issues
- Functional Issues
- Timing Issues

Resource and Planning Issues

Typically, single stand-alone interfaces should not present any Quartus II Fitter or timing issues. You may find that fitter, timing, and hardware operation can sometimes become a challenge, as multiple interfaces are combined into a single project, or as the device utilization increases. In such cases, the interface configuration is not the issue, the placement and total device resource requirements create problems.

Resource Issue Evaluation

External memory interfaces typically require the following resource types, which you must consider when trying to manually place, or perhaps use additional constraints to force the placement or location of external memory interface IP:

- Dedicated IOE DQS group resources and pins
- Dedicated DLL resources
- Specific PLL resources
- Specific global, regional, and dual-regional clock net resources

Dedicated IOE DQS Group Resources and Pins

Fitter issues can occur with even a single interface, if you do not size the interface to fit within the specified constraints and requirements. A typical requirement includes containing assignments for the interface within a single bank or possibly side of the chosen device.

Such a constraint requires that the chosen device meets the following conditions:

- Sufficient DQS groups and sizes to support the required number of common I/O (CIO) or separate I/O (SIO) data groups.
- Sufficient remaining pins to support the required number of address, command, and control pins.

Failure to evaluate these fundamental requirements can result in suboptimal interface design, if the chosen device cannot be modified. The resulting wraparound interfaces or suboptimal pseudo read and write data groups artificially limit the maximum operating frequency.


Multiple blocks of IP further complicate the issue, if other IP has either no specified location constraints or incompatible location constraints.

The Quartus II fitter may first place other components in a location required by your memory IP, then error at a later stage because of an I/O assignment conflict between the unconstrained IP and the constrained memory IP.

Your design may require that one instance of IP is placed anywhere on one side of the device, and that another instance of IP is placed at a specific location on the same side.

While the two individual instances may compile in isolation, and the physical number of pins may appear sufficient for both instances, issues can occur if the instance without placement constraints is placed before the instance with placement constraints.

In such circumstances, Altera recommends manually placing each individual pin, or at least try using more granular placement constraints.


-  For more information about the pin number and DQS group capabilities of your chosen device, refer to device data sheets or the Quartus II pin planner.

Dedicated DLL Resources

Altera devices typically use DLLs to enhance data capture at the FPGA.

While multiple external memory interfaces can usually share DLL resources, fitter issues can occur when there is insufficient planning before HDL coding. If DLL sharing is required, Altera gives the following recommendations for each instance of the IP that shares the DLL resources:

- Must have compatible DLL requirements—same frequency and mode.
- Exports its autogenerated DLL instance out of its own dedicated PHY hierarchy and into the top-level design file. This procedure allows easy comparison of the generated DLL's mode, and allows you to explicitly show the required DLL sharing between two IP blocks in the HDL

-  The Quartus II fitter does not dynamically merge DLL instances.

Specific PLL Resources

When only a single interface resides on one side or one quadrant of a device, PLL resources are typically not an issue. However if multiple interfaces or IP are required on a single side or quadrant, consider the specific PLL used by each IP, and the sharing of any PLL resources.

The Quartus II software automerges PLL resources, but not for any dynamically controlled PLL components. Use the following PLL resource rules:

- Ensure that the PLL located in the same bank or side of the device is available for your memory controller.
- If multiple PLLs are required for multiple controllers that cannot be shared, ensure that enough PLL resources are available within each quadrant to support your interface number requirements.
- Try to limit multiple interfaces to a single quadrant. For example, if two complete same size interfaces can fit on a single side of the device, constrain one interface entirely in one bank of that side, and the other controller in the other bank.

-  For more information about using multiple PHYs or controllers, refer to the design tutorials on the [List of designs using Altera External Memory IP](#) page of the Altera Wiki website.

Specific Global, Regional and Dual-Regional Clock Net Resources

Memory PHYs typically have specific clock resource requirements for each PLL clock output. For example because of characterization data, the PHY may require that the `phy_clk` is routed on a global clock net. The remaining clocks may all be routed on a global or a regional clock net. However, they must all be routed on the same type. Otherwise, the operating frequency of the interface is lowered, because of the increased uncertainty between two different types of clock nets. The design may still fit, but not meet timing.

Planning Issue Evaluation

Plan the total number of DQS groups and total number of other pins required in your shared area. Use the Pin Planner to assist with this activity.

Decide which PLLs or clock networks can be shared between IP blocks, then ensure that sufficient resources are available. For example, if an IP core requires a regional clock network, a PLL located on the opposite side of the device cannot be used.

Calculate the number of total clock networks and types required when trying to combine multiple instances of IP.

You must understand the number of quadrants that the IP uses and if this number can be reduced. For example, an interface may be autoplaced across an entire side of the device, but may actually be constrained to fit in a single bank.

Optimizing the physical placement ensures that when possible, regional clock networks are used as opposed to dual-regional clock networks, hence clock net resources are maintained and routing is simplified.

As device utilization increases, the Quartus II software may have difficulty placing the core. To optimize design utilization, follow these steps:

- Review any fitter warning messages in multiple IP designs to ensure that clock networks or PLL modes are not modified to achieve the desired fit.
- Use the Quartus II Fitter resource section to compare the types of resources used in a successful standalone IP implementation to those used in an unreliable multiple IP implementation.
- Use this information to better constrain the project to achieve the same results as the standalone project.
- Use the Chip Planner (Floorplan and Chip Editor) to compare the placement of the working stand-alone design to the multiple IP project. Then use LogicLock™ or Design Partitions to better guide the Quartus II software to the required results.
- When creating LogicLock regions, ensure that they encompass all required resources. For example, if constraining the read and write datapath hierarchy, ensure that your LogicLock region includes the IOE blocks used for your datapath pin out.


Interface Configuration Issues

This topic describes the performance (f_{MAX}), efficiency (latency and transaction efficiency) and bottleneck (the limiting factor) issues that you can encounter in any design.

Performance Issues

There are a large number of interface combinations and configurations possible in an Altera design, therefore it is impractical for Altera to explicitly state the achievable f_{MAX} for every combination. Altera seeks to provide guidance on typical performance, but this data is subject to memory component timing characteristics, interface widths, depths directly affecting timing deration requirements, and the achieved skew and timing numbers for a specific PCB.

FPGA timing issues should generally not be affected by interface loading or layout characteristics. In general, the Altera performance figures for any given device family and speed-grade combination should usually be achievable.

 To resolve FPGA (PHY and PHY reset) timing issues, refer to the *Analyzing Timing of Memory IP* chapter.

Achievable interface timing (address and command, half-rate address and command, read and write capture) is directly affected by any layout issues (skew), loading issues (deration), signal integrity issues (crosstalk timing deration), and component speed grades (memory timing size and tolerance). Altera performance figures are typically stated for the default (single rank, unbuffered DIMM) case. Altera provides additional expected performance data where possible, but the f_{MAX} is not achievable in all configurations. Altera recommends that you optimize the following items whenever interface timing issues occur:

- Improve PCB layout tolerances
- Use a faster speed grade of memory component
- Ensure that the interface is fully and correctly terminated
- Reduce the loading (reduce the deration factor)

Bottleneck and Efficiency Issues

Depending on the transaction types, efficiency issues can exist where the achieved data rate is lower than expected. Ideally, these issues should be assessed and resolved during the simulation stage because they are sometimes impossible to solve later without rearchitecting the product.

Any interface has a maximum theoretical data rate derived from the clock frequency, however, in practise this theoretical data rate can never be achieved continuously due to protocol overhead and bus turnaround times.

Simulate your desired configuration to ensure that you have specified a suitable external memory family and that your chosen controller configuration can achieve your required bandwidth.

Efficiency can be assessed in several different ways, and the primary requirement is an achievable continuous data rate. The local interface signals combined with the memory interface signals and a command decode trace should provide adequate visibility of the operation of the IP to understand whether your required data rate is sufficient and the cause of the efficiency issue.

To show if under ideal conditions the required data rate is possible in the chosen technology, follow these steps:

1. Use the memory vendors own testbench and your own transaction engine.
2. Use either your own driver, or modify the provided example driver, to replicate the transaction types typical of your system.
3. Simulate this performance using your chosen memory controller and decide if the achieved performance is still acceptable.

Observe the following points that may cause efficiency or bottleneck issues at this stage:

- Identify the memory controller rate (full, half, or quarter) and commands, which may take two or four times longer than necessary
- Determine whether the memory controller is starved for data by observing the appropriate request signals.
- Determine whether the memory controller processor transactions at a rate sufficient to meet throughput requirements by observing appropriate signals, including the local ready signal.

Altera has several versions and types of memory controller, and where possible you can evaluate different configurations based on the results of the first tests.

Consider using either a faster interface, or a different memory type to better align your data rate requirements to the IP available directly from Altera.

Altera also provides stand-alone PHY configurations so that you may develop custom controllers or use third-party controllers designed specifically for your requirements.

Functional Issues

This topic discusses functional issues that occur at all frequencies (using the same conditions) and are not altered by speed grade, temperature, or PCB changes.

Functional Issue Evaluation

Functional issues should be evaluated using functional simulation.

The Altera IP includes the option to autogenerate a testbench specific to your IP configuration, which provides an easy route to functional verification.

The following issues should be considered when trying to debug functional issues in a simulation environment.

Correct Combination of the Quartus II Software and ModelSim-Altera Device Models

When running any simulations, ensure that you are using the correct combination of the Quartus II software and device models. Altera only tests each release of software and IP with the aligned release of device models. Failure to use the correct RTL and model combination may result in unstable simulation environments.

The ModelSim®-Altera edition of the ModelSim simulator comes precompiled with the Altera device family libraries included. You must always install the correct release of ModelSim-Altera to align with your Quartus II software and IP release.

If you are using a full version of ModelSim-SE or PE, or any other supported simulation environment, ensure that you are compiling the current Quartus II supplied libraries. These libraries are located in the *<Quartus II install path>/quartus/eda/sim_lib/* directory.

Altera IP Memory Model

Altera memory IP autogenerates a generic simplified memory model that works in all cases. This simple read and write model is not designed or intended to verify all entered IP parameters or transaction requirements.


The Altera-generated memory model may be suitable to evaluate some limited functional issues, but it does not provide comprehensive functional simulation.

Vendor Memory Model

Contact the memory vendor directly as many additional models are available from the vendors support system.

When using memory vendor models, ensure that the model is correctly defined for the following characteristics:

- Speed grade
- Organization
- Memory allocation
- Maximum memory usage
- Number of ranks on a DIMM
- Buffering on the DIMM
- ECC

 Refer to the **readme.txt** file supplied with the memory vendor model, for more information about how to define this information for your configuration.

During simulation vendor models output a wealth of information regarding any device violations that may occur because of incorrectly parameterized IP.

 Refer to Transcript Window Messages, for more information.

Out of PC Memory Issues

If you are running the ModelSim-Altera simulator, the limitation on memory size, may mean that you have insufficient memory to run your simulation. Or, if you are using a 32-bit operating system, your PC may have insufficient memory.

Typical simulation tool errors include: "Iteration limit reached" or "Error out of memory".

When using either the Altera generic memory model, or a vendor specific model quite large memory depths can be required if you do not specify your simulation carefully.

For example, if you simulate an entire 4-GB DIMM interface, the hardware platform that performs that simulation requires at least this amount of memory just for the simulation contents storage.

 Refer to Memory Allocation and Max Memory Usage in the vendor's **readme.txt** files for more information.

Transcript Window Messages

When debugging a functional issue in simulation, vendor models typically provide a much more detailed checks and feedback regarding the interface and their operational requirements than the Altera generic model.

In general, you should use a vendor-supplied model whenever one is available. Consider using second-source vendor models in preference to the Altera generic model.

Many issues can be traced to incorrectly configured IP for the specified memory components. Component data sheets usually contain settings information for several different speed grades of memory. Be aware data sheet specify parameters in fixed units of time, frequencies, or clock cycles.

The Altera generic memory model always matches the parameters specified in the IP, as it is generated using the same engine. Because vendor models are independent of the IP generation process, they offer a more robust IP parameterization check.

During simulation, review the transcript window messages and do not rely on the Simulation Passed message at the end of simulation. This message only indicates that the example driver successfully wrote and then read the correct data for a single test cycle.

Even if the interface functionally passes in simulation, the vendor model may report operational violations in the transcript window. These reported violations often specifically explain why an interface appears to pass in simulation, but fails in hardware.

Vendor models typically perform checks to ensure that the following types of parameters are correct:

- Burst length
- Burst order
- tMRD
- tMOD
- tRFC
- tREFPDEN
- tRP
- tRAS
- tRC
- tACTPDEN
- tWR
- tWRPDEN
- tRTP
- tRDPDEN
- tINIT
- tXPDLL

- tCKE
- tRRD
- tCCD
- tWTR
- tXPR
- PRECHARGE
- CAS length
- Drive strength
- AL
- tDQS
- CAS_WL
- Refresh
- Initialization
- tIH
- tIS
- tDH
- tDS

If a vendor model can verify all these parameters are compatible with your chosen component values and transactions, it provides a specific insight into hardware interface failures.

Simulation

Passing simulation means that the interface calibrates and successfully completes a single test complete cycle without asserting pass not fail (pnf). It does not take into account any warning messages that you may receive during simulation. If you are debugging an interface issue, review and, if necessary, correct any warning messages from the transcript window before continuing.

Modifying the Example Driver to Replicate the Failure


Often during debugging, you may discover that the example driver design works successfully, but that your custom logic is observing data errors. This information indicates that the issue is either:

- Related to the way that the local interface transactions are occurring. Altera recommends you probe and compare using the SignalTap™ II analyzer.
- Related to the types or format of transactions on the external memory interface. Altera recommends you modify the example design to replicate the issue.

Typical issues on the local interface side include:


- Incorrect local address to memory address translation causing the word order to be different than expected. Refer to *Burst Definition* in your memory vendor data sheet.

- Incorrect timing on the local interface. When your design requests a transaction, the local side must be ready to service that transaction as soon as it is accepted without any pause.

 For more information, refer to the *Avalon® Interface Specification*.

The default example driver only performs a limited set of transaction types, consequently potential bus contention or preamble and postamble issues can often be masked in its default operation. For successful debugging, isolate the custom logic transaction types that are causing the read and write failures and modify the example driver to demonstrate the same issue. Then, you can try to replicate the failure in RTL simulation with the modified driver.

An issue that you can replicate in RTL simulation indicates a potential bug in the IP. You should recheck the IP parameters. An issue that you can not replicate in RTL simulation indicates a timing issue on the PCB. You can try to replicate the issue on an Altera development platform to rule out a board issue.

-  Ensure that all PCB timing, loading, skew, and deration information is correctly defined in the Quartus II software, as the timing report is inaccurate if this initial data is not correct.

Functional simulation allows you to identify any issues with the configuration of either the Altera memory controller and or PHY. You can then easily check the operation against both the memory vendor data sheet and the respective JEDEC specification. After you resolve functional issues, you can start testing hardware.

 For more information about simulation, refer to the *Simulating Memory IP* chapter.

Timing Issues

The Altera PHY and controller combinations autogenerate timing constraint files to ensure that the PHY and external interface are fully constrained and that timing is analyzed during compilation. However, timing issues can still occur. This topic discusses how to identify and resolve any timing issues that you may encounter.

Timing Issue Characteristics

Timing issues typically fall into two distinct categories:

- FPGA core timing reported issues
- External memory interface timing issues in a specific mode of operation or on a specific PCB

TimeQuest reports timing issues in two categories: core to core and core to IOE transfers. These timing issues include the PHY and PHY reset sections in the TimeQuest Report DDR subsection of timing analysis. External memory interface timing issues are specifically reported in the TimeQuest Report DDR subsection, excluding the PHY and PHY reset. The Report DDR PHY and PHY reset sections only include the PHY, and specifically exclude the controller, core, PHY-to-controller and local interface. Quartus II timing issues should always be evaluated and corrected before proceeding to any hardware testing.

PCB timing issues are usually Quartus II timing issues, which are not reported in the Quartus II software, if incorrect or insufficient PCB topology and layout information is not supplied. PCB timing issues are typically characterized by calibration failure, or failures during user mode when the hardware is heated or cooled. Further PCB timing issues are typically hidden if the interface frequency is lowered.

Timing Issue Evaluation

Try to fix and identify timing issues in the Quartus II software. Consider the following issues when resolving timing issues.

FPGA Timing Issues

In general, you should not have any timing issues with Altera-provided IP unless you running the IP outside of Altera's published performance range or are using a version of the Quartus II software with preliminary timing model support for new devices. However, timing issues can occur in the following circumstances:

- The **.sdc** files are incorrectly added to the Quartus II project
- Quartus II analysis and synthesis settings are not correct
- Quartus II Fitter settings are not correct

For all of these issues, refer to the correct user guide for more information about recommended settings and follow these steps:

1. Ensure that the IP generated **.sdc** files are listed in the Quartus II TimeQuest Timing Analyzer files to include in the project window.
2. Ensure that **Analysis and Synthesis Settings** are set to **Optimization Technique Speed**.
3. Ensure that **Fitter Settings** are set to **Fitter Effort Standard Fit**.
4. Use **TimeQuest Report Ignored Constraints**, to ensure that **.sdc** files are successfully applied.
5. Use **TimeQuest Report Unconstrained Paths**, to ensure that all critical paths are correctly constrained.

More complex timing issues can occur if any of the following conditions are true:

- The design includes multiple PHY or core projects
- Devices where the resources are heavily used
- The design includes wide, distributed, maximum performance interfaces in large die sizes

Any of these conditions can lead to suboptimal placement results when the PHY or controller are distributed around the FPGA. To evaluate such issues, simplify the design to just the autogenerated example top-level file and determine if the core meets timing and you see a working interface. Failure implies that a more fundamental timing issue exists. If the standalone design passes core timing, evaluate how this placement and fit is different than your complete design.

Use LogicLock regions, or design partitions to better define the placement of your memory controllers. When you have your interface standalone placement, repeat for additional interfaces, combine, and finally add the rest of your design.

Additionally, use fitter seeds and increase the placement and router effort multiplier.

External Memory Interface Timing Issues

External memory interface timing issues are not directly related to the FPGA timing but are actually derived from the FPGA input and output characteristics, PCB timing, and the memory component characteristics.

The FPGA input and output characteristics tend to be a predominately fixed value, as the IOE structure of the devices is fixed. Optimal PLL characteristics and clock routing characteristics do have an effect. Assuming the IP is correctly constrained with the autogenerated assignments, and you follow implementation rules, the design should reach the stated performance figures.

The memory component characteristics are fixed for any given component or DIMM. However, consider using faster components or DIMMs in marginal cases when PCB skew may be suboptimal, or your design includes multiple ranks when deration may be causing read capture or write timing challenges. Using faster memory components typically reduces the memory data output skew and uncertainty easing read capture, and lowering the memory's input setup and hold requirement, which eases write timing.


Increased PCB skew reduces margins on address, command, read capture and write timing. If you are narrowly failing timing on these paths, consider reducing the board skew (if possible), or using faster memory. Address and command timing typically requires you to manually balance the reported setup and hold values with the dedicated address and command phase in the IP.

 Refer to the respective IP user guide for more information.

Multiple-slot multiple-rank UDIMM interfaces can place considerable loading on the FPGA driver. Typically a quad rank interface can have thirty-six loads. In multiple-rank configurations, Altera's stated maximum data rates are not likely to be achievable because of loading deration. Consider using different topologies, for example registered DIMMs, so that the loading is reduced.

Deration because of increased loading, or suboptimal layout may result in a lower than desired operating frequency meeting timing. You should close timing in the Quartus II software using your expected loading and layout rules before committing to PCB fabrication.

Ensure that any design with an Altera PHY is correctly constrained and meets timing in the Quartus II software. You must address any constraint or timing failures before testing hardware.

 For more information about timing constraints, refer to the *Analyzing Timing of Memory IP* chapter.

Verifying Memory IP Using the SignalTap II Logic Analyzer

The SignalTap II logic analyzer shows read and write activity in the system.

 For more information about using the SignalTap II logic analyzer, refer to *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in volume 3 of the *Quartus II Software Handbook*

To add the SignalTap II logic analyzer, follow these steps:

1. On the Tools menu click **SignalTap II Logic Analyzer**.
2. In the **Signal Configuration** window next to the **Clock** box, click ... (Browse Node Finder).
3. Type the memory interface system clock (typically `*phy_clk`) in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.
4. Select the memory interface system clock (`<variation name>_example_top | <variation name>:<variation name>_inst | <variation name>_controller_phy:<variation name>_controller_phy_inst | phy_clk | phy_clk`) in **Nodes Found** and click > to add the signal to **Selected Nodes**.
5. Click **OK**.
6. Under Signal Configuration, specify the following settings:
 - For **Sample depth**, select **512**
 - For **RAM type**, select **Auto**
 - For **Trigger flow control**, select **Sequential**
 - For **Trigger position**, select **Center trigger position**
 - For **Trigger conditions**, select **1**
7. On the Edit menu, click **Add Nodes**.
8. Search for specific nodes by typing `*local*` in the **Named** box, for **Filter** select **SignalTap II: pre-synthesis** and click **List**.

9. Select the following nodes in **Nodes Found** and click > to add to **Selected Nodes**:

- local_address
- local_rdata
- local_rdata_valid
- local_read_req
- local_ready
- local_wdata
- local_wdata_req
- local_write_req
- pnf
- pnf_per_byte
- test_complete (trigger)
- ctl_cal_success
- ctl_cal_fail
- ctl_wlat
- ctl_rlat



Do not add any memory interface signals to the SignalTap II logic analyzer. The load on these signals increases and adversely affects the timing analysis.

10. Click **OK**.

11. To reduce the SignalTap II logic size, turn off **Trigger Enable** on the following bus signals:

- local_address
- local_rdata
- local_wdata
- pnf_per_byte
- ctl_wlat
- ctl_rlat

12. Right-click **Trigger Conditions** for the test_complete signal and select **Rising Edge**.

13. On the File menu, click **Save**, to save the SignalTap II .stp file to your project.




If you see the message **Do you want to enable SignalTap II file "stp1.stp" for the current project**, click **Yes**.

14. Once you add signals to the SignalTap II logic analyzer, recompile your design, on the Processing menu, click **Start Compilation**.

15. When the design compiles, ensure that TimeQuest timing analysis passes successfully. In addition to this FPGA timing analysis, check your PCB or system SDRAM timing. To run timing analysis, run the `*_phy_report_timing.tcl` script.
 - a. On the Tools menu, click **Tcl Scripts**.
 - b. Select `<variation name>_phy_report_timing.tcl` and click **Run**.
16. Connect the development board to your computer.
17. On the Tools menu, click **SignalTap II Logic Analyzer**.
18. Add the correct `<your project name>.sof` file to the SOF Manager:
 - a. Click **...** to open the **Select Program Files** dialog box.
 - b. Select `<your project name>.sof`.
 - c. Click **Open**.
 - d. To download the file, click the **Program Device** button.
19. When the example design including SignalTap II successfully downloads to your development board, click **Run Analysis** to run once, or click **Autorun Analysis** to run continuously.

Monitoring Signals with the SignalTap II Logic Analyzer

The following sections detail the memory controller signals you should consider analyzing for different memory interfaces. The list is not exhaustive, but is a starting point.

 For a description of each signal, refer to *Volume 3: Reference Material* of the *External Memory Interface Handbook*.

DDR, DDR2, and DDR3 ALTMEMPHY Designs

Monitor the following signals for DDR, DDR2, and DDR3 SDRAM ALTMEMPHY designs:

- `Local_* -example_driver` (all the local interface signals)
- `Pnf -example_driver`
- `Pnf_per_byte -example_driver`
- `Test_complete -example_driver`
- `Test_status -example_driver`
- `Ctl_cal_req -phy_inst`
- `Ctl_init_fail -phy_inst`
- `Ctl_init_success -phy_inst`
- `Ctl_cal_fail -phy_inst`
- `Ctl_cal_success -phy_inst`
- `Cal_codvw_phase * -phy_inst`
- `Cal_codvw_size * -phy_inst`

- Codvw_trk_shift * -phy_inst
- Ctl_rlat * -phy_inst
- Ctl_wlat * -phy_inst
- Locked -altpll_component
- Phasecounterselect * -altpll_component
- Phaseupdown -altpll_component
- Phasestep -altpll_component
- Phase_done -altpll_component
- Flag_done_timeout -seq_inst:ctrl
- Flag_ack_timeout -seq_inst:ctrl
- Proc_ctrl.command_err -seq_inst:ctrl
- Proc_ctrl.command_result * -seq_inst:ctrl
- dgrb_ctrl.command_err -seq_inst:ctrl
- dgrb_ctrl.command_result * -seq_inst:ctrl
- dgwb_ctrl.command_err -seq_inst:ctrl
- dgwb_ctrl.command_result * -seq_inst:ctrl
- admin_ctrl.command_err -seq_inst:ctrl
- admin_ctrl.command_result * -seq_inst:ctrl
- setup_ctrl.command_err -seq_inst:ctrl
- setup_ctrl.command_result * -seq_inst:ctrl
- state.s_phy_initialise -seq_inst:ctrl
- state.s_init_dram -seq_inst:ctrl
- state.s_write_ihi -seq_inst:ctrl
- state.s_cal -seq_inst:ctrl
- state.s_write_btp -seq_inst:ctrl
- state.s_write_mtp -seq_inst:ctrl
- state.s_read_mtp -seq_inst:ctrl
- state.s_rrp_reset -seq_inst:ctrl
- state.s_rrp_sweep -seq_inst:ctrl
- state.s_rrp_seek -seq_inst:ctrl
- state.s_rdv -seq_inst:ctrl
- state.s_poa -seq_inst:ctrl
- state.s_was -seq_inst:ctrl
- state.s_adv_rd_lat -seq_inst:ctrl
- state.s_adv_wr_lat -seq_inst:ctrl

- state.s_prep_customer_mr_setup -seq_inst:ctrl
- state.s_tracking_setup -seq_inst:ctrl
- state.s_tracking -seq_inst:ctrl
- state.s_reset -seq_inst:ctrl
- state.s_non_operational -seq_inst:ctrl
- state.s_operational -seq_inst:ctrl
- dqs_delay_ctrl_export * -phy_inst
- * = Disable Trigger Enable

UniPHY Designs


Monitor the following signals for UniPHY designs:

- avl_addr
- avl_rdata
- avl_rdata_valid
- avl_read_req
- avl_ready
- avl_wdata
- avl_write_req
- fail
- pass
- afi_cal_fail
- afi_cal_success
- test_complete
- be_reg (QDRII only)
- pnf_per_bit
- rdata_reg
- rdata_valid_reg
- data_out
- data_in
- written_data_fifo|data_out
- usequencer|state*
- usequencer|phy_seq_rdata_valid
- usequencer|phy_seq_read_fifo_q
- usequencer|phy_read_increment_vfifo*
- usequencer|phy_read_latency_counter
- uread_datapath|afi_rdata_en

- `uread_datapath|afi_rdata_valid`
- `uread_datapath|ddio_phy_dq`
- `qvld_wr_address*`
- `qvld_rd_address*`

Hardware Debugging Guidelines

Before starting to debug, confirm the design followed the Altera recommended design flow.

-  Refer to the *Design Flow* chapter in volume 1 of the *External Memory Interface Handbook*.

Always keep a record of tests, to avoid repeating the same tests later. To start debugging the design, perform the following initial steps.

Create a Simplified Design that Demonstrates the Same Issue

To help debugging create a simple design that replicates the issue. A simple design compiles faster and is much easier to understand. Altera's external memory interface IP generates an example top-level file that is ideal for debugging. The example top-level file uses all the same parameters, pin-outs, and so on.

Measure Power Distribution Network

Ensure you take measurements of the various power supplies on their hardware development platform over a suitable time base and with a suitable trigger using an appropriate probe and grounding scheme. In addition, take the measurements directly on the pins or vias of the devices in question, and with the hardware operational.

Measure Signal Integrity and Setup and Hold Margin

Measure the signals on their PCB to ensure that everything looks correct. This information can be vital. When measuring any signal, consider the edge rate of the signal, not just its frequency. Modern FPGA devices have very fast edge rates, therefore you must use a suitable oscilloscope, probe, and grounding scheme when you measure the signals.

You can take measurements to capture the setup and hold time of key signal classes with respect to their clock or strobe. Ensure that the measured setup and hold margin is at least better than that reported in the Quartus II software. A worse margin indicates that a timing discrepancy exists somewhere in the project. However, this timing issue may not be the cause of your problem.

Vary Voltage

Try and vary the voltage of your system, if you suspect a marginality issue. Increasing the voltage typically causes devices to operate faster and also usually provides increased noise margin.

Use Freezer Spray and Heat Gun

If you have an intermittent marginal issue, cool or heat the interface to try and stress the issue. Cooling down ICs causes them to run faster, which makes timing easier. Conversely heating up ICs causes them to run slower, which makes timing more difficult.

If cooling or heating fixes the issue, you are probably looking at a timing issue.

Operate at a Lower Speed

Test the interface at a lower speed. If the interface works, the interface is correctly pinned out and functional. However, if the interface fails at a lower speed, determine if the test is valid. Many high-speed memory components have a minimal operating frequency, or require subtly different configurations when operating at a lower speeds.

For example, DDR, DDR2, or DDR3 SDRAM typically requires modification to the following parameters if you want operate the interface a lower speeds:

- t_{MRD}
- t_{WTR}
- CAS latency and CAS write latency

Find Out if the Issue Exists in Previous Versions of Software

Hardware that works before an update to either the Quartus II software or the memory IP indicates that the development platform is not the issue. However, the previous generation IP may be less susceptible to a PCB issue, masking the issue.

Find out if the Issue Exists in the Current Version of Software

Designs are often tested using previous generations of Altera software or IP. Projects do not always get upgraded for the following reasons:

- Multiple engineers are on the same project. To ensure compatibility, a common release of Altera software is used by all engineers for the duration of the product development. The design may be several releases behind the current Quartus II software version.
- Many companies delay before adopting a new release of software so that they can first monitor Internet forums to get a feel for how successful other users say the software is.
- Many companies never use the latest version of any software, preferring to wait until the first service pack is released that fixes the primary issues.
- Some users may only have a license for the older version of the software and can only use that version until their company makes the financial decision to upgrade.
- The local interface specification from Altera IP to the customer's logic sometimes changes from software release to software release. If you have already spent resources designing interface logic, you may be reluctant to repeat this exercise. If a block of code is already signed off, you may be reluctant to modify it to upgrade to newer IP from Altera..

In all of these scenarios, you must determine if the issue still exists in the latest version of the Altera software. Bugs are fixed and enhancements are added to the Altera IP every release. Depending on the nature of the bug or enhancement, it may not always be clearly documented in the release notes.

Finally, if the latest version of the software resolves the issue, it may be easier to debug the version of software that you are using.

Try A Different PCB

If you are using the same Altera IP on a number of different hardware platforms; find out if the issue occurs on all of these hardware platforms, or just one. Multiple instances of the same PCB, or multiple instances of the same interface, on physically different hardware platforms may exhibit different behavior. You can determine if the configuration is fundamentally not working, or if some form of marginality is involved in the issue.

Issues are often reported on the alpha build of a development platform. These are produced in very limited numbers and often have received limited BBT (bare board testing), or FT (functional testing). Hence, these early boards are often more unreliable than production quality PCBs.

Additionally, if the IP is from a previous project to help save development resources, find out if this specific IP configuration works on a previous platform.

Try Other Configurations

Designs are typically quite large, using multiple blocks of IP in many different combinations. Find out if any other configurations work correctly on the development platform. The full project may have multiple external memory controllers in the same device, or may have configurations where only half the memory width or frequency is required. Find out what does and does not work to help the debugging of the issue.

Debugging Checklist

The following checklist is a good starting point when debugging an external memory interface. This chapter discusses all of the items in the checklist.

Check	Item
<input type="checkbox"/>	Try a different fit.
<input type="checkbox"/>	Check IP parameters at the operating frequency (t_{MRD} , t_{WTR} for example).
<input type="checkbox"/>	Ensure you have constrained your design with proper timing deration and have closed timing.
<input type="checkbox"/>	Simulate the design. If it fails in simulation, it will fail in hardware.
<input type="checkbox"/>	Analyze timing.
<input type="checkbox"/>	Place and assign R_{UP} and R_{DN} (OCT).
<input type="checkbox"/>	Measure the power distribution network (PDN).
<input type="checkbox"/>	Measure signal integrity.
<input type="checkbox"/>	Measure setup and hold timing.

Check	Item
<input type="checkbox"/>	Measure FPGA voltages.
<input type="checkbox"/>	Vary voltages.
<input type="checkbox"/>	Heat and cool the PCB.
<input type="checkbox"/>	Operate at a lower or higher frequency.
<input type="checkbox"/>	Check board timing and trace Information.
<input type="checkbox"/>	Check LVDS and clock sources, I/O voltages and termination.
<input type="checkbox"/>	Check PLL clock source, specification, and jitter.
<input type="checkbox"/>	Ensure the correct number of PLL phase steps take place during calibration. If the number stated in the IP does not match the number, you may have manually altered the PLL.
<input type="checkbox"/>	Retarget to a smaller interface width or a single bank.

Categorizing Hardware Issues

The following topic categorizes issues. Identifying which category or groups of category an issue may be classified within allows you to focus on the cause of the issue.

Signal Integrity Issues

Many design issues, even ones that you find at the protocol layer, can often be traced back to signal integrity issues. Hence, you must check circuit board construction, power systems, command, and data signaling to determine if they meet specifications. If infrequent, random errors exist in the memory subsystem, product reliability suffers. As such, electrical verification is vital. Check the bare circuit board or PCB design file. Circuit board errors can cause poor signal integrity, signal loss, signal timing skew, and trace impedance mismatches. Differential traces with unbalanced lengths or signals that are routed too closely together can cause crosstalk.

Characteristics

Signal integrity issues often appear when the performance of the hardware design is marginal. The design may not always initialize and calibrate correctly, or may exhibit occasional bit errors in user mode. Severe signal integrity issues can result in total failure of an interface at certain data rates, and sporadic component failure because of electrical stress. PCB component variance and signal integrity issues often show up as failures on one PCB, but not on another identical board. Timing issues can have a similar characteristic. Multiple calibration windows or significant differences in the calibration results from one calibration to another can also indicate signal integrity issues.

Evaluating Signal Integrity Issues

Signal integrity issues can only really be evaluated in two ways, direct measurement using suitable test equipment like an oscilloscope and probe, or simulation using a tool like HyperLynx or Allegro PCB SI. Signals should be compared against the respective electrical specification. You should look for overshoot and undershoot, non-monotonicity, eye height and width, and crosstalk.

Skew

Ensure that all clocked signals, commands, addresses, and control signals arrive at the memory inputs at the same time. Trace length variations cause data valid window variations between the signals reducing margin. For example, DDR2-800 at 400 MHz has a data valid window that is smaller than 1,250 ps. Trace length skew or crosstalk can reduce this data valid window further, making it difficult to design a reliably operating memory interface. Ensure that the skew figure previously entered into the Altera IP matches that actually achieved on the PCB, otherwise Quartus II timing analysis of the interface is accurate.

Crosstalk

Crosstalk is another issue that is best evaluated early in the memory design phase. Check the clock-to-data strobes, as these are bidirectional. Measure the crosstalk at both ends of the line. Check the data strobes to clock, as the clocks are unidirectional, these only need checking at the memory end of the line.

Power System

Some memory interfaces tend to draw current in spikes from their power delivery system as SDRAMs are based on capacitive memory cells. Rows are read and refreshed one at a time, which causes dynamic currents that can stress any power distribution network (PDN). The various power rails should be checked either at or as close as possible to the SDRAM pins power pins. Ideally, a real-time oscilloscope set to fast glitch triggering should be used for this activity.

Clock Signals

The clock signal quality is important for any external memory system. Measurements include frequency, digital core design (DCD), high width, low width, amplitude, jitter, rise, and fall times.

Read Data Valid Window and Eye Diagram

The memory generates the read signals. Take measurements at the FPGA end of the line. To ease read diagram capture, modify the example driver to mask writes or modify the PHY to include a signal that you can trigger on when performing reads.

Write Data Valid Window and Eye Diagram

The FPGA generates the write signals. Take measurements at the memory device end of the line. To ease write diagram capture, modify the example driver to mask reads or modify the PHY export a signal that is asserted when performing writes.

OCT and ODT Usage

Modern external memory interface designs typically use OCT for the FPGA end of the line, and ODT for the memory component end of the line. If either the OCT or ODT are incorrectly configured or enabled, signal integrity issues exist. If the design is using OCT, R_{UP} or R_{DN} pins must be placed correctly for the OCT to work. If you do not place these pins, the Quartus II software allocates them automatically with the following warning:

```
Warning: No exact pin location assignment(s) for 2 pins of 110 total pins
```

```
Info: Pin termination_blk0~_rup_pad not assigned to an exact location  
on the device
```

```
Info: Pin termination_blk0~_rdn_pad not assigned to an exact location  
on the device
```

If you see these warnings, the R_{UP} and R_{DN} pins may have been allocated to a pin that does not have the required external resistor present on the board. This allocation renders the OCT circuit faulty, resulting in unreliable UniPHY and ALTMEMPHY calibration and or interface behavior. The pins with the required external resistor must be specified in the Quartus II software.

For the FPGA, ensure that follow these actions:

- Specify the R_{UP} and R_{DN} pins in either the projects HDL port list, or in the assignment editor (`termination_blk0~_rup_pad/ termination_blk0~_rdn_pad`).
- Connect the R_{UP} and R_{DN} pins to the correct resistors and pull-up and pull-down voltage in the schematic or PCB.
- Contain the R_{UP} and R_{DN} pins within a bank of the device that is operating at the same VCCIO voltage as the interface that is terminated.
- Check that only the expected number of R_{UP} and R_{DN} pins exists in the project pin-out file. Look for Info: Created on-chip termination messages at the fitter stage for any calibration blocks not expected in your design.
- Review the Fitter Pin-Out file for R_{UP} and R_{DN} pins to ensure that they are on the correct pins, and that only the correct number of calibration blocks exists in your design.
- Check in the fitter report that the input, output, and bidirectional signals with calibrated OCT all have the termination control block applicable to the associated R_{UP} and R_{DN} pins.

For the memory components, ensure that you follow these actions:

- Connect the required resistor to the correct pin on each and every component, and ensure that it is pulled to the correct voltage.
- Place the required resistor close to the memory component.
- Correctly configure the IP to enable the desired termination at initialization time.
- Check that the speed grade of memory component supports the selected ODT setting.
- Check that the second source part that may have been fitted to the PCB, supports the same ODT settings as the original

Hardware and Calibration Issues

When you resolve functional, timing, and signal integrity issues, assess the operation of the PHY and its interface calibration.

Hardware and Calibration Issue Characteristics

Hardware and calibration issues have the following definitions:

- Calibration issues result in calibration failing, which typically results in the design asserting the `ctl_cal_fail` signal.
- Hardware issues result in read and write failures, which typically results in the design asserting the pass not fail (`pnf`) signal



Ensure that functional, timing, and signal integrity issues are not the direct cause of your hardware issue, as functional, timing or signal integrity issues are usually the cause of any hardware issue.

Evaluating Hardware and Calibration Issues

Use the following methods to evaluate hardware and calibration issues:

- Evaluate hardware issues using the SignalTap II logic analyzer to monitor the local side read and write interface with the pass or fail or error signals as triggers
- Evaluate calibration issues using the SignalTap II logic analyzer to monitor the various calibration, configuration with the pass or fail or error signals as triggers, but also use the debug toolkit and system consoles when available



For more information about debug toolkits and the type of signals for debugging external memory interfaces, refer to the *ALTMEMPHY External Memory Interface Debug Toolkit* and *UniPHY External Memory Interface Debug Toolkit* chapters in volume 3 of the *External Memory Interface Handbook*.

Consider adding core noise to your design to aggravate margin timing and signal integrity issues. Steadily increase the stress on the interface in the following order:

1. Increase the interface utilization by modifying the example driver to focus on the types of transactions that exhibit the issue.
2. Increase the SNN or aggressiveness of the data pattern by modifying the example driver to output in synchronization PRBS data patterns, or hammer patterns.
3. Increase the stress on the PDN by adding more and more core noise to your system. Try sweeping the fundamental frequency of the core noise to help identify resonances in your power system.

Steadily increasing the stress on the external memory interface is an ideal way to assess and understand the cause of any previously intermittent failures that you may observe in your system. Using the SignalTap II probe tool can provide insights into the source or cause of operational failure in the system.

Additionally, steadily increasing stress on the external memory interface allows you to assess and understand the impact that such factors have on the amount of timing margin and resynchronization window. Take measurements with and without the additional stress factor to allow evaluation of the overall effect.

Write Timing Margin

Determine the write timing margin by phase sweeping the write clock from the PLL. Use sources and probes to dynamically control the PLL phase offset control, to increase and decrease the write clock phase adjustment so that the write window size may be ascertained.

Remember that when sweeping PLL clock phases, the following two factors may cause operational failure:

- The available write margin.
- The PLL phase in a multi-clock system.

The following code achieves this adjustment. You should use sources and probes to modify the respective output of the PLL. Ensure that the example driver is writing and reading from the memory while observing the pnf_per_byte signals to see when write failures occur:

```

//////////////////////////////////
wire [7:0] Probe_sig;
wire [5:0] Source_sig;
PhaseCount PhaseCounter (
    .resetn (1'b1),
    .clock (pll_ref_clk),
    .step (Source_sig[5]),
    .updown (Source_sig[4]),
    .offset (Probe_sig)
);
CheckoutPandS freq_PandS (
    .probe (Probe_sig),
    .source (Source_sig)
);
ddr2_dimm_phy_alt_mem_phy_pll_siii pll (
    .inclk0 (pll_ref_clk),
    .areset (pll_reset),
    .c0 (phy_clk_1x), // hR
    .c1 (mem_clk_2x), // FR
    .c2 (aux_clk), // FR
    .c3 (write_clk_2x), // FR
    .c4 (resync_clk_2x), // FR
    .c5 (measure_clk_1x), // hR
    .c6 (ac_clk_1x), // hR
    .phasecounterselect (Source_sig[3:0]),
    .phasestep (Source_sig[5]),
    .phaseupdown (Source_sig[4]),
    .scanclk (scan_clk),

```

```
.locked (pll_locked_src),
.phasedone (pll_phase_done)
);
```

Read Timing Margin

Similarly, assess the read timing margin by using sources and probes to manually control the DLL phase offset feature. Open the autogenerated DLL using ALT_DLL and add the additionally required offset control ports. This action allows control and observation of the following signals:

```
dll_delayctrlout[5:0], // Phase output control from DLL to DQS pins
(Gray Coded)
dll_offset_ctrl_a_addnsb, // Input add or subtract the phase offset
value
dll_offset_ctrl_a_offset[5:0], // User Input controlled DLL offset
value (Gray Coded)
dll_aload, // User Input DLL load command
dll_dqsupdate, // DLL Output update required signal.
```

In examples where the applied offset applied results in the maximum or minimum `dll_delayctrlout[5:0]` setting without reaching the end of the read capture window, regenerate the DLL in the next available phase setting, so that the full capture window is assessed.

Modify the example driver to constantly perform reads (mask writes). Observe the `pnf_per_byte` signals while the DLL capture phase is manually modified to see when failures begin, which indicates the edge of the window.

A resynchronization timing failure can indicate failure at that capture phase, and not a capture failure. You should recalibrate the PHY with the calculated phase offset to ensure that you are using the true read-capture margin.

Address and Command Timing Margin

You set the address and command clock phase directly in the IP. Assuming you enter the correct board trace model information into the Quartus II software, the timing analysis should be correct. However, if you want to evaluate the address and command timing margin, use the same process as in [“Write Timing Margin”](#), only phase step the address and command PLL output (`c6 ac_clk_1x`). You can achieve this effect using the debug toolkit or system console.



Refer to the [ALTMEMPHY External Memory Interface Debug Toolkit](#) and [UniPHY External Memory Interface Debug Toolkit](#) chapters in volume 3 of the *External Memory Interface Handbook*.

Resynchronization Timing Margin

Observe the size and margins, available for resynchronization using the debug toolkit or system console.



Refer to the [ALTMEMPHY External Memory Interface Debug Toolkit](#) and [UniPHY External Memory Interface Debug Toolkit](#) chapters in volume 3 of the *External Memory Interface Handbook*.

Additionally for PHY configurations that use a dedicated PLL clock phase (as opposed to a resynchronization FIFO buffer), use the same process as described in “[Write Timing Margin](#)”, to dynamically sweep resynchronization margin (c4 resynch_clk_2x).

Postamble Timing Issues and Margin

The postamble timing is set by the PHY during calibration. You can diagnose postamble issues by viewing the `pnf_per_byte` signal from the example driver. Postamble timing issues mean only read data is corrupted during the last beat of any read request.

Intermittent Issues

Intermittent issues are typically the hardest type of issue to debug—they appear randomly and are hard to replicate.

Intermittent Issue Evaluation

Errors that occur during run-time indicate a data related issue, which you can identify by the following actions:

- Add the SignalTap II logic analyzer and trigger on the post-trigger `pnf`
- Use a stress pattern of data or transactions, to increase the probability of the issue
- Heat up or cool down the system
- Run the system at a slightly faster frequency

If adding the SignalTap II logic analyzer or modifying the project causes the issue to go away, the issue is likely to be placement or timing related.

Errors that occur at start-up indicate that the issue is related to calibration, which you can identify by the following actions:

- Modify the design to continually calibrate and reset in a loop until the error is observed
- Where possible, evaluate the calibration margin either from the debug toolkit or system console.



Refer to the [ALTMEMPHY External Memory Interface Debug Toolkit](#) and [UniPHY External Memory Interface Debug Toolkit](#) chapters in volume 3 of the *External Memory Interface Handbook*.

- Capture the calibration error stage or error code, and use this information with whatever specifically occurs at that stage of calibration to assist with your debug of the issue.

Debug Toolkit

The debug toolkit is an interface that runs on your PC and enables you to debug your external memory interface design on the circuit board, retrieve calibration status, and perform margining activities. Debug toolkit uses a JTAG connection to a Windows PC.

Altera provides the following types of debug toolkits:

- ALTMEMPHY Debug Toolkit
- UniPHY EMIF Debug Toolkit

ALTMEMPHY Debug Toolkit Overview and Usage Flow

The ALTMEMPHY Debug Toolkit supports the following Altera AFI-based IP:

- ALTMEMPHY megafunction
- DDR2 and DDR3 SDRAM High-Performance Controller and High Performance Controller II



The debug toolkit does not support the QDR II and II+ SRAM, RLDRAM II with UniPHY controllers.

The ALTMEMPHY Debug Toolkit lists and indicates whether calibration stages are successful, states error specific to calibration failure and provides possible causes. However, the debug toolkit does not fix a failing design. You can run the debug toolkit and the SignalTap II logic analyzer at the same time.

The ALTMEMPHY Debug Toolkit usage flow involves the following steps:

1. Before using the debug toolkit, modify the design example top-level file by regenerating the IP with the JTAG Avalon-MM port enabled.
2. Add additional debug and sequencer signals to indicate the location of calibration failure, resynchronization margin, read and write latency, and PLL status.
3. Recompile the design.
4. Connect your pc's download cable (for example, ByteBlaster™ II download cable) to the JTAG port on the development board.
5. Program the device with the debug enabled in your design using the SignalTap II logic analyzer.
6. Run analysis and interpret calibration results using the ALTMEMPHY Debug Toolkit with the SignalTap II logic analyzer.



For more information about the ALTMEMPHY debug toolkit and calibration stages, refer to the *ALTMEMPHY External Memory Interface Debug Toolkit* chapter in volume 3 of the *External Memory Interface Handbook*.

UniPHY EMIF Debug Toolkit Overview and Usage Flow

The UniPHY EMIF Debug Toolkit is a Tcl-based interface and consists of the following parts:

- DDR2 and DDR3 SDRAM Controllers with UniPHY
- Avalon Memory-Mapped (Avalon-MM) slave interface
- JTAG Avalon master

The EMIF toolkit allows you to display information about your external memory interface and generate calibration and margining reports. The toolkit can aid in diagnosing the type of failure that may be occurring in your external memory interface, and help identify areas of reduced margin that might be potential failure points.

The UniPHY Debug Toolkit usage flow involves the following steps:

1. (Optional) Generate your IP core with the CSR port enabled and with the CSR communication interface type properly set.
2. Recompile the design.
3. Connect your pc's download cable (for example, ByteBlaster II download cable) to the JTAG port on the development board.
4. Program the device.
5. Specify project settings using the UniPHY EMIF Debug Toolkit.
6. Generate calibration report and interpret calibration results using the UniPHY EMIF Debug Toolkit.



For more information about the UniPHY EMIF debug toolkit and calibration stages, refer to the *UniPHY External Memory Interface Debug Toolkit* chapter in volume 3 of the *External Memory Interface Handbook*.

Document Revision History

Table 11-1 lists the revision history for this document.

Table 11-1. Document Revision History

Date	Version	Changes
November 2011	4.0	Added Debug Toolkit section.
June 2011	3.0	Removed leveling information from <i>ALTMEMPHY Calibration Stages</i> and <i>UniPHY Calibration Stages</i> chapter.
December 2010	2.1	<ul style="list-style-type: none"> ■ Added new chapter: <i>UniPHY Calibration Stages</i>. ■ Added new chapter: <i>DDR2 and DDR3 SDRAM Controllers with UniPHY EMIF Toolkit</i>.
July 2010	2.0	Updated for 10.0 release.
January 2010	1.2	Corrected minor typos.
December 2009	1.1	Added <i>Debug Toolkit for DDR2 and DDR3 SDRAM High-Performance Controllers</i> chapter and <i>ALTMEMPHY Calibration Stages</i> chapter.
November 2009	1.0	First published.