


This chapter describes a debug toolkit for ALTMEMPHY-based high performance controllers. The debug toolkit uses a JTAG connection to a Windows PC. The 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 supports all FPGA device families supported by the high-performance controller (HPC) and high-performance controller II (HPC II) and ALTMEMPHY.

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

The debug toolkit provides detailed information regarding the calibration process. The debug toolkit and the SignalTap II logic analyzer can be run at the same time. However using **Autorun Analysis** in the SignalTap II logic analyzer slows down the JTAG communication with the debug toolkit.

This chapter provides the following information:

- “Debug Toolkit Overview”
- “Install the Debug Toolkit”
- “Modify the Example Top-Level File to use the Debug Toolkit”
- “Use the Debug Toolkit”
- “Interpret the Results”
- “Understand the Checksum and Failure Code”

 The debug toolkit provides information on the failures and calibration results that assist and direct the hardware debug process. The debug toolkit does not fix a failing design. Before you use the debug toolkit, refer to **Debugging Memory IP** in volume 2, section 1, of the *External Memory Interface Handbook*.

Debug Toolkit Overview

The debug TOOLKIT provides the following information:

- Lists the various calibration stages and indicates whether each stage was successful or not.
- States an error code specific to the exact type of calibration failure.

- Provides possible causes for calibration failures.
- Provides graphics to visualize the following parameters:
 - Resync clock phase setup (per pin)
 - Read deskew multipurpose registers (MPR) (prime DQ pins only)
 - Read deskew block training pattern (per pin)
 - Write deskew (per pin)
 - Write leveling report

You can export a **.doom** file from the debug toolkit. This file provides a record of your calibration results and the ALTMEMPHY IOE configuration specific to your system, allowing you to refer to this data offline later.

Install the Debug Toolkit

To install the debug toolkit, follow these steps:

1. Download the debug toolkit, **debug-toolkit.zip**, file from Altera [website](#).
2. Unzip the **debug-toolkit.zip** file.
3. To start the debug toolkit, navigate to the directory where you unzipped the **.zip** file and run **debug-toolkit.exe**.

To install the debug toolkit on a Quartus II production programming PC, follow these steps:

1. On a PC running the Windows OS, copy the **debug-toolkit.zip** file to your project directory or a common programming directory that you also use to program your test platform using a USB-Blaster™ download cable.
2. Unzip the **debug-toolkit.zip** file to either your project folder or a common programming folder.

Modify the Example Top-Level File to use the Debug Toolkit

Before you use the debug toolkit, you must modify your design's example-top-level file, by following these steps:

- [Verify the Design](#)
- [Regenerate the IP](#)
- [Instantiate the JTAG Avalon-MM port in to the Example-Top Level Project](#)
- [Add Additional Signals](#)
- [Add alt_jtagavalon.v to your Quartus II Project Settings Files List](#)
- [Recompile your Quartus II Test Design](#)
- [Program Hardware with Debug Enabled .sof](#)



Your design must follow the recommended flow; refer to the [Recommended Design Flow](#) chapter in volume 1 of the *External Memory Interface Handbook*.

Verify the Design

Ensure your design meets the following conditions:

- The parameters entered into the IP are correct for the memory and data rate.
- The design passes functional simulation.
- The Quartus II project has the correct board trace models specified for the PCB you are using.
- For Cyclone III devices, ensure that the **set t(additional_addresscmd_tpd)** parameter is correctly specified in your **.sdc** file.
- For Arria II GX devices, ensure that the **Address and Command to CK skew** parameter is correctly specified in the **Board Settings** tab of the IP wizard.
- The address and command clock phase is correct, to ensure optimum balanced setup and hold times.
- The Quartus II design successfully closes timing.
- The Quartus II project has the correct pin location assignments for the PCB that you are using.
- The autogenerated IP assignments are correctly applied to the example top-level file.
- The **.sdc** constraint files are correctly applied to the example top-level file.
- The Quartus II settings are correctly applied.
- The R_{UP}/R_{DN} pin locations are correctly specified in the example top-level file if required.
- The SignalTap II logic analyzer is added to the example top-level file.

Before you use the debug toolkit, follow these steps:

1. Edit the example top-level file to enable debugging:
 - a. Open the *<variation name>.v* or **.vhd** and find the `export_debug_port` private value.



Do not edit this value in the file *<variation name>_phy.v* or **.vhd** file.

The value is at the bottom of the file:

```
// =====  
// DDR3 High Performance Controller Wizard Data  
// =====  
// DO NOT EDIT FOLLOWING DATA  
// @Altera, IP Toolbench@  
...  
...  
// Retrieval info: <PRIVATE name = "export_debug_port" value="false"  
type="STRING" enable="1" />
```

- b. Edit the `export_debug_port` private value to true:

```
// Retrieval info: <PRIVATE name = "export_debug_port" value="true"
type="STRING" enable="1" />
```

Regenerate the IP

To regenerate the IP, follow these steps:

1. Open the MegaWizard Plug In Manager, and select **Edit an existing custom megafunction variation**.
2. Select your modified high-performance controller.
3. Click **Next** to open the IP.
4. Click **Finish** to regenerate the IP.

You now have a version of the design with debug enabled. Seven new ports with the prefix `dbg_*` are added to the controller instance through the design hierarchy up to the `<variation name>_example_top.v` or `.vhd`.

Instantiate the JTAG Avalon-MM port in to the Example-Top Level Project

To instantiate the JTAG Avalon-MM port, follow these steps:

1. Declare the following wires in `<variation name>_example_top.v` or `.vhd`.

```
wire [12: 0] av_address;
wire av_write_n;
wire [31: 0] av_writedata;
wire av_read_n;
wire av_waitrequest;
wire [31: 0] av_readdata;
```

2. Add the following instances in `<variation name>_example_top.v` or `.vhd`.

```
// inst jtag avalon:
alt_jtagavalon alt_jtagavalon(
.clk (phy_clk),
.rst_n (reset_phy_clk_n),
.av_address (av_address),
.av_write_n (av_write_n),
.av_writedata (av_writedata),
.av_read_n (av_read_n),
.av_readdata (av_readdata),
.av_waitrequest (av_waitrequest)
);
defparam alt_jtagavalon.SLD_NODE_INFO = 203976192;
defparam alt_jtagavalon.ADDR_WIDTH = 13;
defparam alt_jtagavalon.DATA_WIDTH = 32;
defparam alt_jtagavalon.MODE_WIDTH = 3;
```

3. Update the following port connections in the DDR2 or DDR3 SDRAM instance in `<variation_name>_example_top.v` or `.vhd`:
 - a. Locate the PHY or controller instance in the top-level file and locate the following debug port connections:

```
//<< START MEGAWIZARD INSERT WRAPPER_NAME
<variation_name> <variation_name>_inst
(
  .dbg_addr (13'b0),
  .dbg_cs (1'b0),
  .dbg_rd (1'b0),
  .dbg_rd_data (dbg_rd_data_sig),
  .dbg_waitrequest (dbg_waitrequest_sig),
  .dbg_wr (1'b0),
  .dbg_wr_data (32'b0),
```

- b. Change the following debug port connections to:

```
//<< START MEGAWIZARD INSERT WRAPPER_NAME
<variation_name> <variation_name>_inst
(
  .dbg_addr (av_address),
  .dbg_cs (1'b1),
  .dbg_rd (~av_read_n),
  .dbg_rd_data (av_readdata),
  .dbg_waitrequest (av_waitrequest),
  .dbg_wr (~av_write_n),
  .dbg_wr_data (av_writedata),
```

The debug toolkit is added to your example top-level file.

Add Additional Signals

In addition to the standard SignalTap II signals, you can add the following signals during debug to understand the following situations:

- Where calibration failed:
 - `*ctl_init_fail -phy_inst`
 - `*ctl_init_success -phy_inst`
 - `ctl_cal_fail -phy_inst`
 - `ctl_cal_success -phy_inst`
- How much resynchronization margin is available:
 - `*cal_codvw_phase *DT-phy_inst`
 - `*cal_codvw_size *DT-phy_inst`
 - `*codvw_trk_shift *DT-phy_inst`

- What the read and write latency is calibrated as:
 - `ctl_rlat *DT-phy_inst`
 - `ctl_wlat *DT-phy_inst`
- If the PLL is locked and phase stepping as expected:
 - `Locked -altpll_component`
 - `Phasecounterselect *DT-altpll_component`
 - `Phaseupdown -altpll_component`
 - `Phasestep -altpll_component`
 - `phasedone -altpll_component`
 - `dqs_delay_ctrl_export *DT-phy_inst`


 For signals marked with *DT, disable trigger enable in the SignalTap II logic analyzer to reduce memory requirement.

Table 16-1 shows sequencer signals that you can also probe using the SignalTap II logic analyzer, to help you understand where calibration failure is occurring. The signals are in the `<vaiation_name>_alt_mem_phy_seq.vhd` file.


 All signals are active high.

Table 16-1. Sequencer Signals (Part 1 of 2)

Port Name	Description
<code>Flag_done_timeout</code>	Calibration stage timeout failure, memory did not respond.
<code>Flag_ack_timeout</code>	Sequencer failed to respond.
<code>state.s_phy_initialise</code>	PHY initialization Stage: wait for DLL lock and <code>init_done</code> .
<code>state.s_init_dram</code>	DRAM initialization stage: reset sequence.
<code>State.s_prog_cal_mrs</code>	DRAM initialization stage: programming mode registers (once per chip select).
<code>state.s_write_ihi</code>	Write internal RAM header initialization.
<code>state.s_cal</code>	Calibration required stage.
<code>state.s_write_btp</code>	Write block training pattern stage: 00001111.
<code>state.s_write_mtp</code>	Write memory training patterns: 00110101.
<code>state.s_rrp_reset</code>	Read resynchronization phase reset: PLL initial condition.
<code>state.s_rrp_sweep</code>	Read resynchronization phase sweep: sweep PLL phases per chip select.
<code>state.s_read_mtp</code>	Read memory training patterns to find correct alignment.
<code>State.s_rrp_seek</code>	Read resynchronization phase setup stage: set PLL to center of valid window.
<code>state.s_rdv</code>	Read data valid stage.
<code>state.s_poa</code>	Postamble calibration stage.
<code>state.s_was</code>	Write datapath setup: write data to DRAM so that latency can be determined.
<code>state.s_adv_rd_lat</code>	Advertise read latency stage.

Table 16–1. Sequencer Signals (Part 2 of 2)

Port Name	Description
state.s_adv_wr_lat	Advertise write latency stage.
state.s_tracking_setup	Tracking setup stage (first pass to setup mimic window).
state.s_prep_customer_mr_setup	Set custom mode register settings (admin).
state.s_tracking	Tracking stage (mimic path tracking in user mode).
state.s_operational	Calibration success: user mode.
state.s_non_operational	Calibration failed or tracking failed in user mode.
state.s_reset	Reset stage.
dgrb_ctrl.command_err	Error in the data gather read bias block.
dgrb_ctrl.command_result[7..0]	Data gather read block (DGRB) error code.
dgwb_ctrl.command_err	Error in the data gather write bias block.
dgwb_ctrl.command_result[7..0]	Data gather write block (DGWB) error code.
admin_ctrl.command_err	Error in the admin (DRAM initialization and control) block.
admin_ctrl.command_result[7..0]	Admin block error code.

Add alt_jtagavalon.v to your Quartus II Project Settings Files List

Before you compile your design, you must add the `alt_jtagavalon.v` file to your projects file list. This `alt_jtagavalon.v` file is included with the debug toolkit.

Recompile your Quartus II Test Design

You must compile your modified design to generate a new `.sof` for testing that includes the debug toolkit code. Altera recommends you ensure that this modified design continues to pass timing analysis. Any timing failures should be assessed and corrected before using the debug toolkit.

Program Hardware with Debug Enabled .sof

To program hardware with the debug enabled `.sof`, program the device using the SignalTap II logic analyzer. Then click **Run Analysis** to run once. Typically, the SignalTap II logic analyzer is initially configured to trigger on the signal `test_complete`, which is fine for working designs.

For designs that are failing calibration, Altera recommends modifying the trigger based on the observed results. Combine this SignalTap II trigger fault isolation activity with use of the debug toolkit. For example:

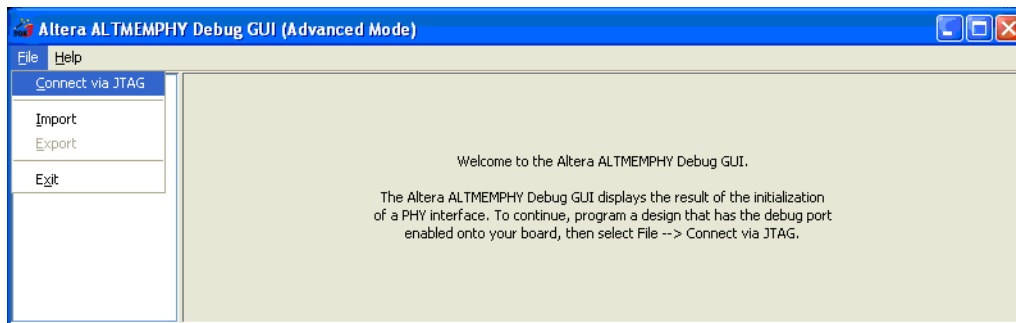
1. Initially trigger on `test_complete`, if the interface works first time.
2. Trigger on `cal_fail`, if the PHY is failing calibration.
3. Trigger on the same `state.s_*` or error code that is reported as the calibration failure point in the debug toolkit.
4. Trigger on `init_fail`, if the memory is failing to initialize.
5. Trigger on `pll_locked`, if the PLL is operating incorrectly.

Use the Debug Toolkit

To use the debug toolkit, follow these steps:

1. Double-click `debug-toolkit.exe`.
2. On the File menu, click **Connect via JTAG** (Figure 16-1).

Figure 16-1. Connect to JTAG

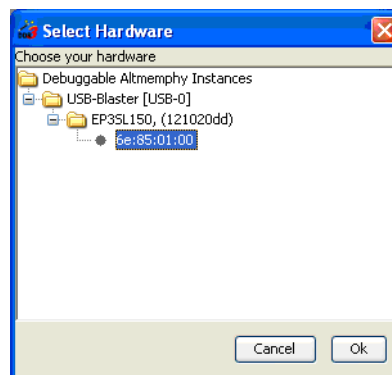


3. Navigate down the hierarchy and click on the Avalon-MM JTAG node (Figure 16-2).



If you encounter connection problems, Altera recommends that you have only a single USB-Blaster™ download cable programming adaptor connected to your PC.

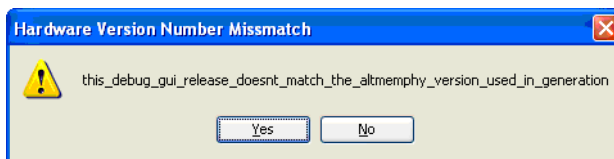
Figure 16-2. Select Hardware



4. If you receive a prompt stating the following message, verify you have the latest debug toolkit, and click **Yes** (Figure 16-3).

```
Hardware Version Number Mismatch -  
this_debug_GUI_release_doesnt_match_the_altmemphy_version_used_in_g  
eneration
```

Figure 16-3. Hardware Mismatch



Interpret the Results

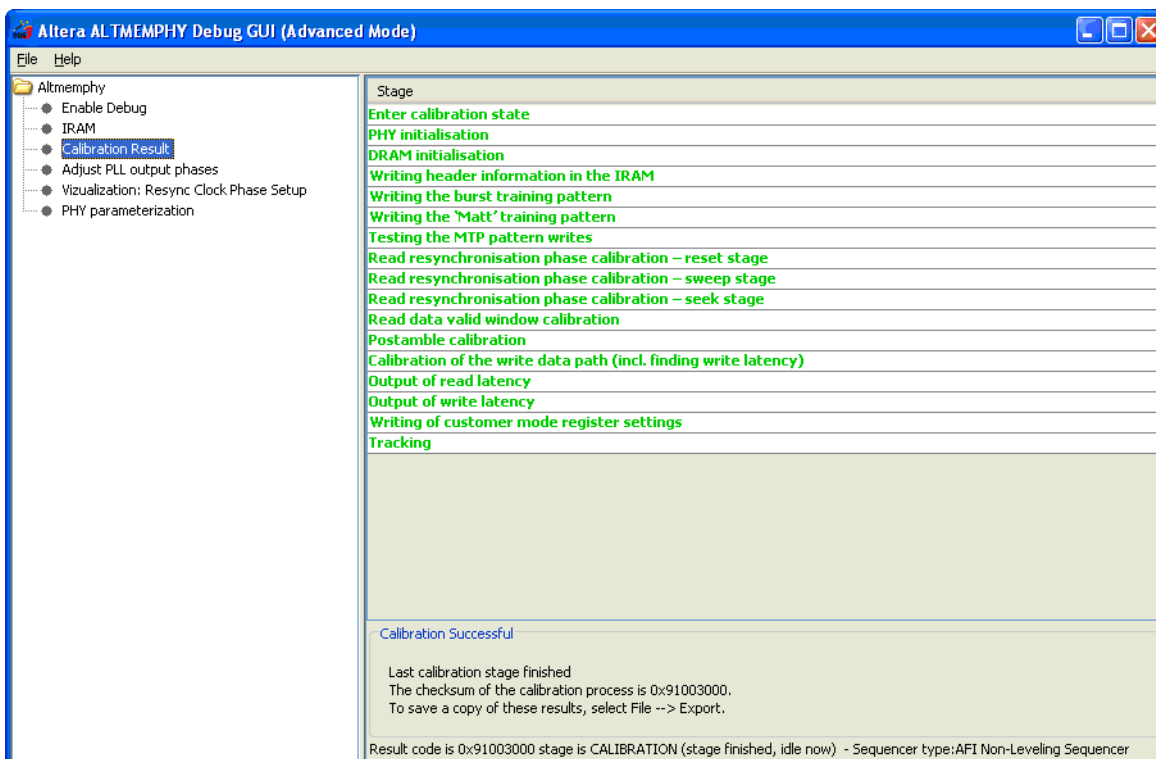
This topic discusses:

- Calibration Successful
- Calibration Fails

Calibration Successful

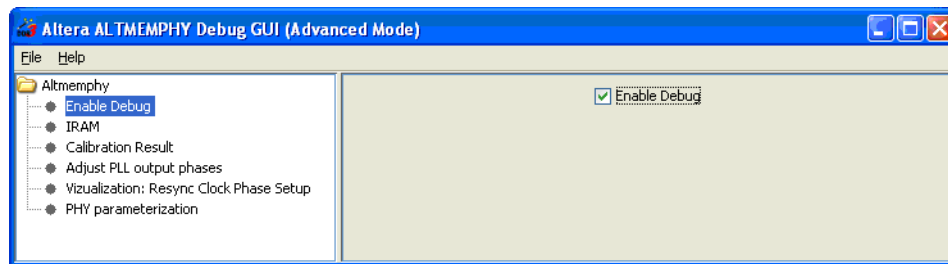
If calibration is successful, you see the following screen (Figure 16-4).

Figure 16-4. Calibration Successful



For optimum operation of the debug toolkit, ensure that you turn on **Enable Debug** (Figure 16-5).

Figure 16-5. Enable Debug



Click **internal RAM** to display the calibration memory results (Figure 16-6).


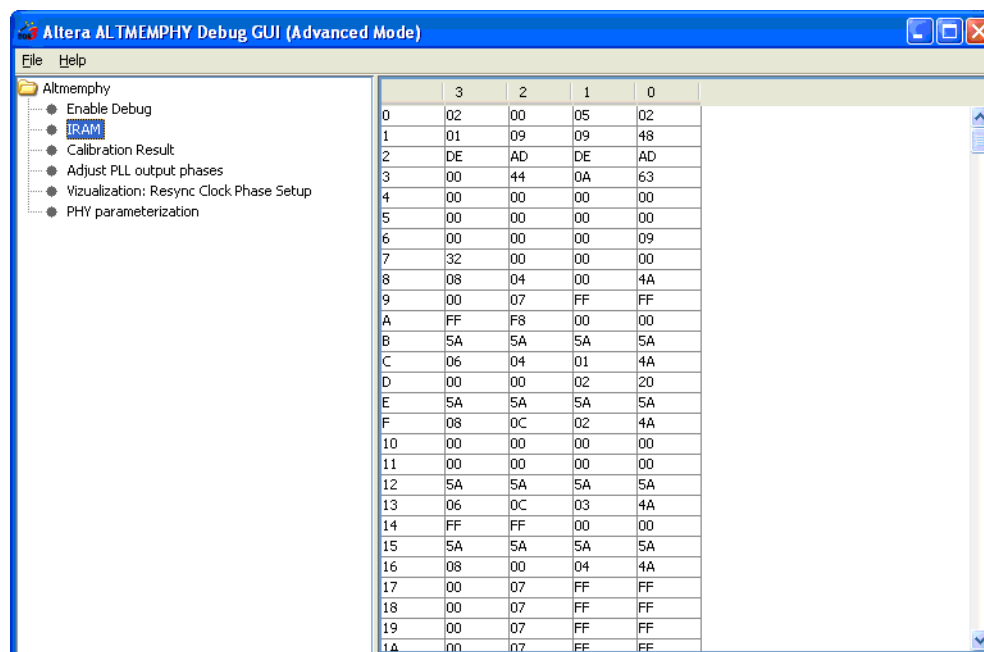
 This setting is not typically used.

Figure 16-6. Calibration Memory Results



The debug toolkit can dynamically alter the PLL clock phases (Figure 16-7).


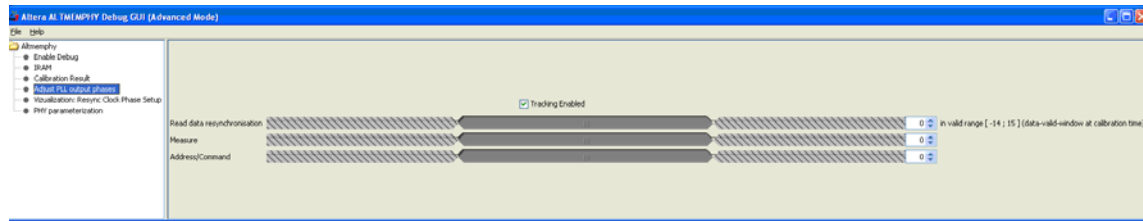


 This setting is not typically used.

Figure 16-7. Altering PLL Clock Phases



The debug toolkit states the number of resynchronization clock phase steps that are valid at calibration time. For example, the resynchronization window size in PLL phase steps at calibration in [Figure 16-7](#) is 26 PLL phase steps wide.

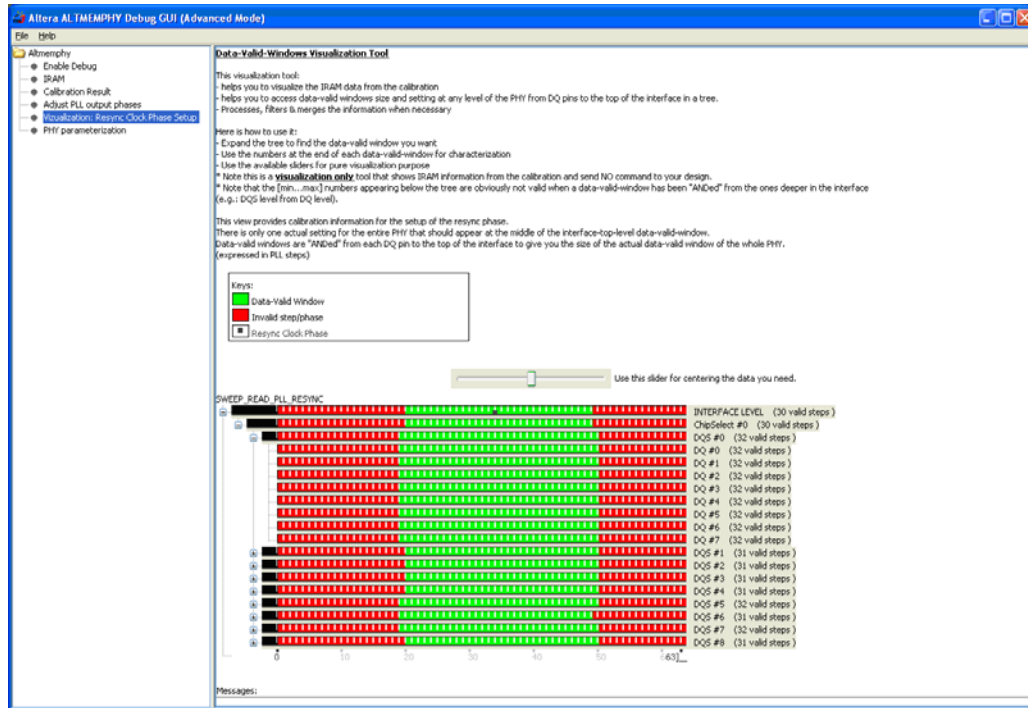
 The address and command phase sweep is limited to either address and command pin margins or address and command core-to-I/O transfer margins.

 In [Figure 16-7](#), moving the slider to the left increases the value, while moving the slider to the right decreases the value.

Click **Visualization: resync clock phase setup** ([Figure 16-8](#)) to show the PHY resynchronization pass and fail results in an expandable tree structure:

- For the whole interface including the chosen phase (black dot)
- On a DQS group basis
- On a per DQ pin basis

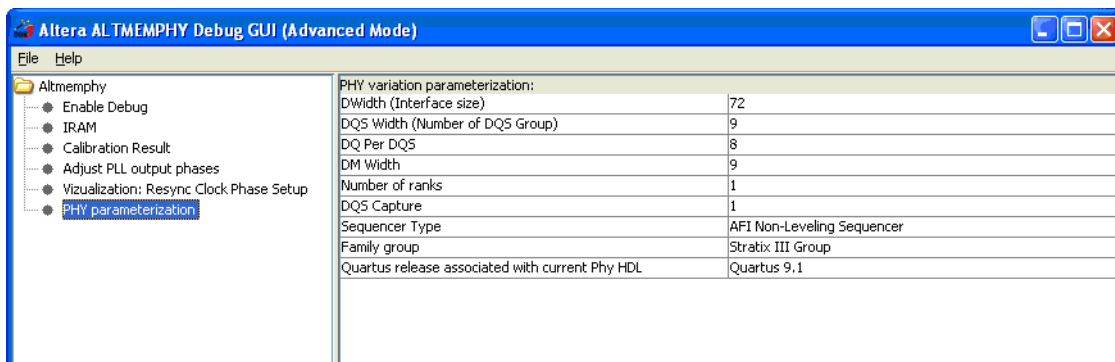
Figure 16-8. Visualization



The debug toolkit additionally states the number of passing phase steps that it finds during calibration. Thus to calculate the resynchronization margin in this design, the resynchronization clock (C6) from the PLL has a phase-shift step resolution of 78.12 ps or 5.62 degrees. So 30 valid steps means that the window size = 2.343 ns or 168.6 degrees.

Click **PHY parameterization** (Figure 16-9), to show the exact calibration configuration of the generated IP.

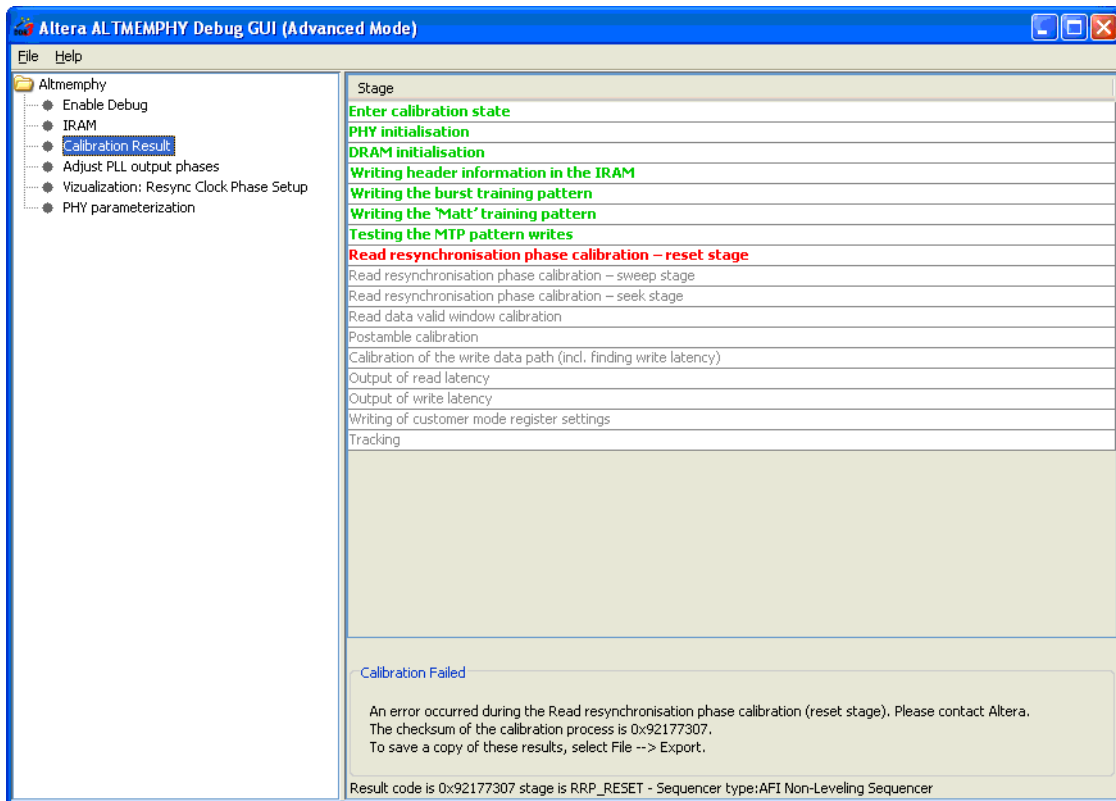
Figure 16-9. PHY Parameterization



Calibration Fails

If calibration fails, you see the following screen (Figure 16-10).

Figure 16-10. Calibration Fails



The stage at which calibration fails is highlighted in red; the stages that have successfully passed are in green. When possible, the debug toolkit also provides a possible cause for the failure code.

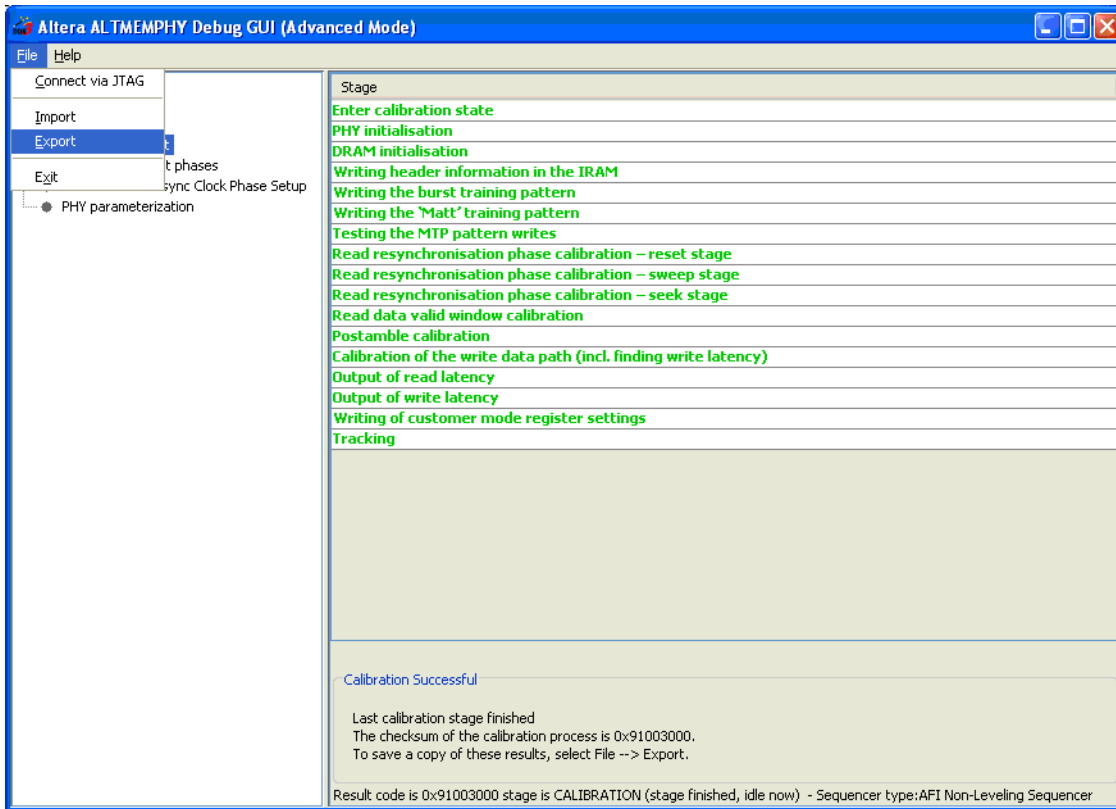
This failure code is: 0x92177307, for more information on failure codes, refer to “Understand the Checksum and Failure Code” on page 16-15.

Save the Calibration Results

Often the calibration failure stage, the reported suggested failure cause, or the combined debug toolkit result and waveforms viewed in the SignalTap II logic analyzer provide enough detail to resolve the failure directly. However, you may wish to save your calibration results, so that you can refer to them later.

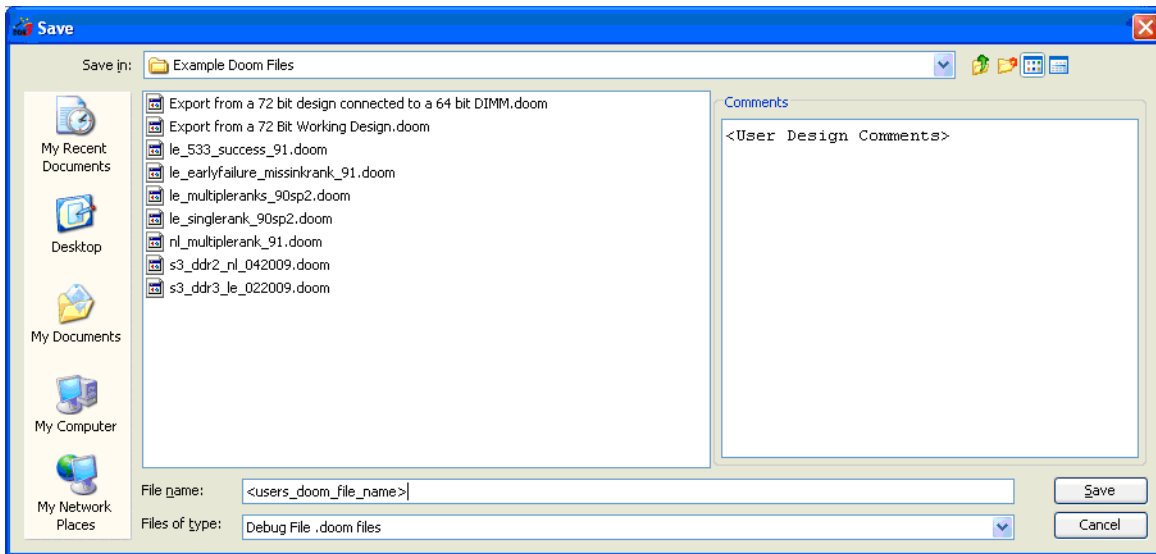
With your calibration process results still displayed on the File menu, click **Export** (Figure 16-11).

Figure 16-11. Export



In the **Save** dialog box (Figure 16-12), specify a file name and any comments to help with the identification and understanding of the controller configuration that you have evaluated, and details on what you may have tested.

Figure 16-12. Save



You can save this **.doom** file with a Quartus II archive (**.qar**) file of the test design, and a copy of the captured SignalTap II waveform files, as a single design archive. This archive provides a record of your calibration results and the ALTMEMPHY IOE configuration specific to your system, so you can refer to this data at a later date.

Understand the Checksum and Failure Code

The debug toolkit checksum provides a direct correlation to the exact stage that calibration failed and the error code for that failure.

For example, the hexadecimal code in the format `0xAABBCCDD` represents the full 32-bit contents of the calibration status register.

The code and the subcode have the following definitions:

- The code or calibration stage is the first byte (DD) or [7..0]
- The subcode or error code is the third byte (BB) or [23..16]

Take the hexadecimal value of each code and convert that to decimal. When you have these two numbers in decimal format, you can open the **failuremessages_nl.csv** spreadsheet file and look up the likely causes of your calibration failure.


 In a passing interface, these two numbers are zero.

Table 16-2 shows the codes that correspond to the indicated calibration stages.

Table 16-2. Calibration Stages

Stage	Code
Enter calibration state	0
PHY initialization	1
DRAM initialization	2
Writing header information in the internal RAM	3
Writing the burst training pattern	4
Writing more training patterns	5
Testing more training pattern writes	6
Read resynchronization phase calibration—reset stage	7
Read resynchronization phase calibration—sweep stage	8
Read resynchronization phase calibration—seek stage	9
Read data valid window calibration	10
Postamble calibration	11
Calibration of the write datapath (including finding write latency)	12
Output of read latency	13
Output of write latency	14
Writing of customer mode register settings	15
Tracking	16

You can find the same information from the SignalTap II logic analyzer if the `*_ctrl.command_err`, `*_ctrl.command_result` and `state.s_*` signals are added. The command error and state signals identify within which calibration stage the interface fails. The corresponding command result then includes the same information as the subcode.

For more information on the stages of calibration, refer to “ALTMEMPHY Calibration Stages” in section 1, chapter 2, of this volume.

Document Revision History

Table 16-3 lists the revision history for this document.

Table 16-3. Document Revision History

Date	Version	Changes
November 2011	1.0	Harvested 11.0 Debug Toolkit for DDR2 and DDR3 SDRAM Controllers with ALTMEMPHY IP content.