ASCALON^{IP}

Getting Started and Integration Guide

Revision 2.0
Feb 12, 2025

tenstorrent

Table of Contents

Revision History

Date	Version	Changes
Apr 28, 2025	2.0.1	Added driving information for libcells Added VC Static CDC information Updated environment requirements
Feb 20, 2025	2.0.0	Updated branding, formatting, and proofreading
Dec 9, 2024	1.1.6	Added spyglass lint target Updated package structure diagram
Dec 02, 2024	1.1.5	Formatting
Nov 18, 2024	1.1.4	Added Xcelium support information
Nov 08, 2024	1.1.3	Proofreading and formatting
Oct 13, 2024	1.1.2	Updated: Makefile instructions, filelist paths, testlist names, directory structure, block names Added: additional integration files, PLL, temperature hub, and temperature sensor integration information, information on building physical design and simulation targets, information on building optimized and debug targets, information on dumping testbench waveforms, check_env.sh.
Sep 12, 2024	1.1.1	Updated branding and formatting.
Aug 15, 2024	1.1.0	Updated: branding and formatting, primitive cell integration instructions, memory integration instructions, internally verified simulation/waveform tool versions, references to abbreviated directories to reflect new expanded name structures, references to testbench to reflect new design_verification top-level directory. Added: memory file integration information, Cluster IO integration information
Apr 15, 2024	1.0.0	First release.

Introduction

This guide describes using the TT-Ascalon™ Cluster package, the included testbench, and included scripts and workflows.

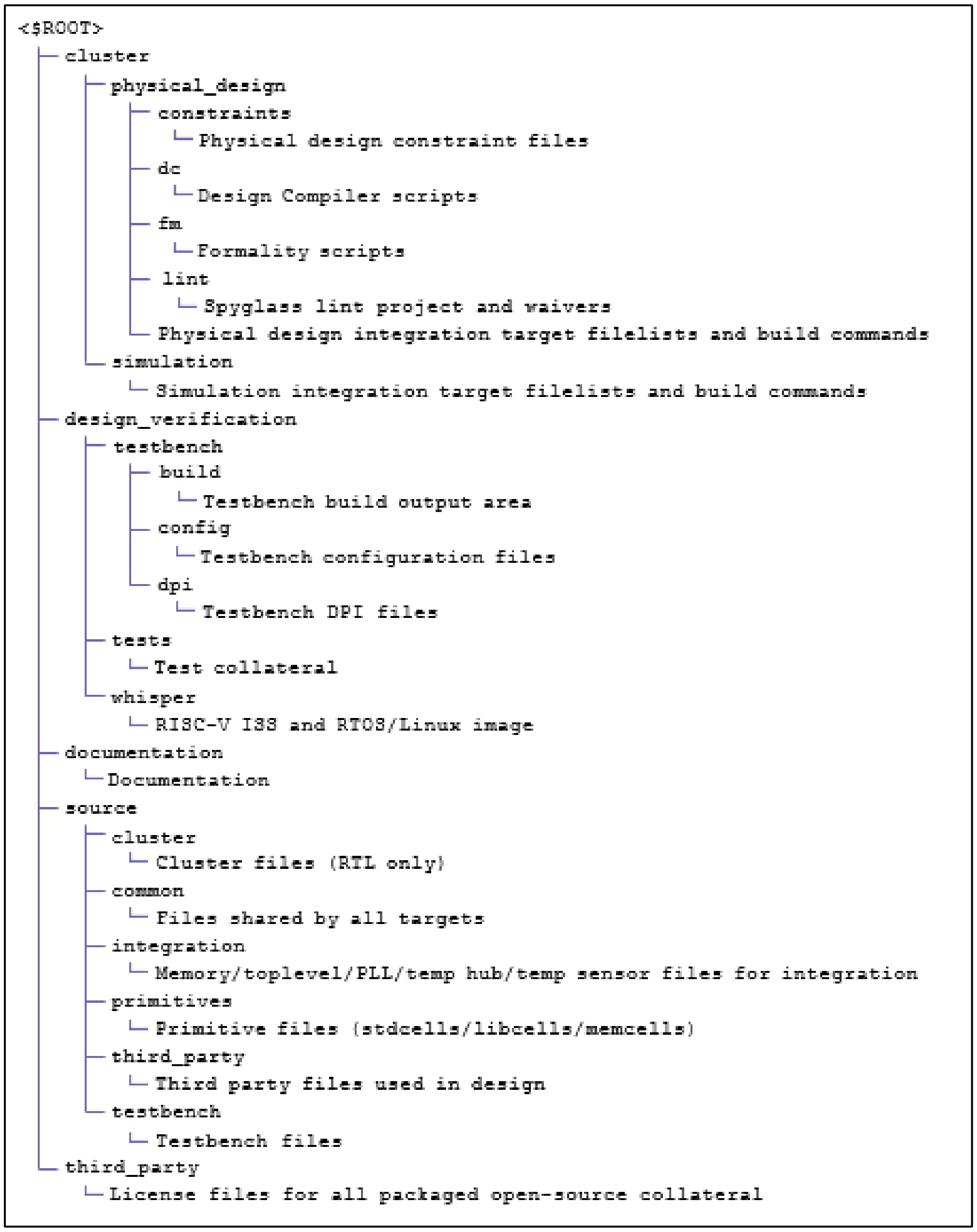
Intended Audience

This document is intended for SoC designers and IP developers familiar with common EDA vendor tools and technology who intend to integrate the TT-Ascalon™ Cluster into a design.

Integration Pre-Requisites

TT-Ascalon™ Cluster Package

Figure 1 shows the TT-Ascalon™ Cluster directory structure after extracting the package. <$ROOT> is the directory that contains the extracted package.
Note: <$ROOT> also refers to the environment variable in the IP package files.



```
<$ROOT>
├─ cluster
│   ├─ physical_design
│   │   ├─ constraints
│   │   │   └─ Physical design constraint files
│   │   ├─ dc
│   │   │   └─ Design Compiler scripts
│   │   ├─ fm
│   │   │   └─ Formality scripts
│   │   ├─ lint
│   │   │   └─ Spyglass lint project and waivers
│   │   └─ Physical design integration target filelists and build commands
│   └─ simulation
│       └─ Simulation integration target filelists and build commands
├─ design_verification
│   ├─ testbench
│   │   ├─ build
│   │   │   └─ Testbench build output area
│   │   ├─ config
│   │   │   └─ Testbench configuration files
│   │   └─ dpi
│   │       └─ Testbench DPI files
│   ├─ tests
│   │   └─ Test collateral
│   └─ whisper
│       └─ RISC-V ISS and RTOS/Linux image
├─ documentation
│   └─ Documentation
├─ source
│   ├─ cluster
│   │   └─ Cluster files (RTL only)
│   ├─ common
│   │   └─ Files shared by all targets
│   ├─ integration
│   │   └─ Memory/toplevel/PLL/temp hub/temp sensor files for integration
│   ├─ primitives
│   │   └─ Primitive files (stdcells/libcells/memcells)
│   ├─ third_party
│   │   └─ Third party files used in design
│   └─ testbench
│       └─ Testbench files
└─ third_party
    └─ License files for all packaged open-source collateral
```

Figure 1 Ascalon Cluster Package Directory Tree

Documentation

TT-Ascalon™ Cluster includes the following documents:

- Getting Started and Integration Guide:
    - Describes the package contents, integration, testbench and the related tools for development.
    - Path: *documentation/ascalon_getting_started_and_integration.pdf*
- Physical Design Reference Guide:
    - Describes the physical design flows and using them with the TT-Ascalon™ Cluster package.
    - Path: *documentation/ascalon_physical_design_reference.pdf*
- Cluster Architecture Manual:
    - Describes the TT-Ascalon™ architecture, including system components, interfaces, supported RISC-V features, registers, and software sequences.
    - Path: *documentation/ascalon_arch_manual.pdf*
- Open-source licenses:
    - Describes the list of open-source licenses used in TT-Ascalon™ Cluster design and testbench environment, in the Markdown format.
    - Path: *documentation/open_source_licenses.pdf*

Compatible EDA Tools

Table 1 Verified and Compatible List of EDA Tools

Type	Version
Simulation	• Synopsys VCS U-2023.03-SP2-5 • Cadence Xcelium 24.03-s003
Waveform	Synopsys Verdi U-2023.03-SP2-5
Synthesis	Synopsys Design Compiler T-2022.03-SP4
Logical Equivalence	Synopsys Formality V-2023.12-SP2
RTL Lint	Synopsys Spyglass V-2023.12-SP1-1
CDC	Synopsys VC Static W-2024.09-SP1

Setting-Up the Test Environment

Perform the following steps to set up the test environment:

- Set the following environment variables:
    - Extract TT-Ascalon™ Cluster and set $ROOT to the extracted directory.
    - Install Synopsys VCS U-2023.03-SP2-5 and set $VCS_HOME to the installed directory.
    - Install Cadence Xcelium 24.03-s003 and set $XCELIUM_HOME to the installed directory.
    - Install Synopsys Verdi U-2023.03-SP2-5 and set $VERDI_HOME to the installed directory.

Note: Ensure to include the installed executables to $PATH for easy access and dumping waveforms.

- Install Python 3.6+ to build the testbench.
- Install Python 3.9+ to run the tests.

Note: The *$ROOT/design_verification/testbench/run_test.sh* script automatically creates and sources a Python venv at *$ROOT/design_verification/testbench/tenstorrent_tb_venv* with modules from *$ROOT/design_verification/testbench/requirements.txt*.

- Ensure that your Red Hat Enterprise LINUX platform contains:
    - GCC 12 and toolset binaries included in $PATH and sourced via *source scl_source enable gcc-toolset-12*.
    - GCC 12 libatomic library
    - GBLIC_2.26 header.
    - GLIBCXX_3.4.22 header.
    - CXXABI_1.3.9 header.

Tip: The header files are distributed as a part of the OS **glibc** installation.

- **bc** package: Supports **bc** command.
- **jq** package: Supports **jq** command. This command enables running the design verification test bench *design_verification/testbench/test_status.sh*
- **libnsl:** Supports NIS library.
- **numatl-libs:** Supports NUMA.
- **perl:** Supports Perl.
- **time:** Supports time command.
- **boost1.78-devel**: Needed for testbench.

Using a Container

Using a container is an optional feature. You can build a container using the container file *$ROOT/ design_verification/testbench/Containerfile*
To build and run a collateral inside the container:

1. Execute the following commands using containerization tools such as Docker or Podman.

```
docker build -t tt-ascalon-image -f
$ROOT/design_verification/testbench/Containerfile docker run --rm -it -v
$ROOT:/work -v $VCS_HOME:/vcs_home -v $XCELIUM_HOME:/xcelium_home -v
$VERDI_HOME:/verdi_home tt-ascalon-image
```

2. Start the container and run the following commands:

```
export ROOT=/work
export VCS_HOME=/vcs_home
export XCELIUM_HOME=/xcelium_home export VERDI_HOME=/verdi_home
export PATH=/opt/rh/gcc-toolset-
12/root/usr/bin:${VCS_HOME}/bin:${XCELIUM_HOME}/bin:${VERDI_HOME}/ bin:$$PATH
export SNSPSLMD_LICENSE_FILE=<license> (or alternative license env setup)
```

Integration

This section describes integrating the files from TT-Ascalon™ Cluster package.

Standard Cells

The unencrypted *source/primitives/tt_libcell_*.sv* files provide the necessary standard cells. Ensure that both the following conditions are met, else use the behavioral versions of the cells:

- **SYNTHESIS** is defined.
- **NO_LIBCELL** is not defined.

Implement the cells in files from the table below for synthesis in the corresponding **ifdef** block for the **SYNTHESIS** define.

Most cells are instantiated as D1 (drive-strength 1).. However, in cases where timing is of concern, the synthesis tools to allow size swapping by adding set_size_only attribute to these cells.

- indication of dont_touch (size 1 expected) d0nt_<cell> in instance naming convention
- indication of size_only (size 1 instantiated, but allowed to resize) I_CLG_<cell> in instance naming convention
- <cell> is a cell type nd2, aoi22, nr2, etc

Note: The TT-Acsalon Cluster provides synchronizer wrapper modules for reference. These modules reference other primitive libcell modules and must not be modified.

Table 2 Standard Cells

File	Description
`tt_libcell_aoi22_quad.sv`	4-bit AOI
`tt_libcell_aoi22.sv`	1-bit AOI
`tt_libcell_clkand2.sv`	2-input clock AND gate
`tt_libcell_clkbuf.sv`	Clock buffer
`tt_libcell_clkgate.sv`	Clock gating cell
`tt_libcell_clkinv.sv`	Clock inverter
`tt_libcell_clkmux2.sv`	2-input clock MUX gate
`tt_libcell_clknand2.sv`	2-input clock NAND gate
`tt_libcell_clkor2.sv`	2-input clock OR gate
`tt_libcell_clkxor2.sv`	2-input clock XOR gate
`tt_libcell_dffrxq.sv`	D flip-flop with async active-low reset
`tt_libcell_dffsrxq.sv`	D flip-flop with async set and active-low reset
`tt_libcell_dff.sv`	Standard D flip-flop
`tt_libcell_dffsxq.sv`	D flip-flop with async set
`tt_libcell_fulladder.sv`	Full adder
`tt_libcell_halfadder.sv`	Half adder
`tt_libcell_lhcnqd4.sv`	D Latch with async clear

tt_libcell_metastab_hardened_dffr.sv	D flip-flop that is robust to metastability
tt_libcell_metastab_hardened_dff.sv	D flip-flop with active-low reset robust to metastability
tt_libcell_nd2.sv	2-input AND gate
tt_libcell_nd3.sv	3-input AND gate
tt_libcell_nd4.sv	4-input AND gate
tt_libcell_or2.sv	2-input OR
tt_libcell_stdand2.sv	2-input AND gate
tt_libcell_stdbuf.sv	Standard buffer
tt_libcell_stdinv.sv	Standard inverter
tt_libcell_stdmux2.sv	2-input multiplexer
tt_libcell_stdmux3.sv	3-input multiplexer
tt_libcell_stdnand2.sv	2-input NAND gate
tt_libcell_stdor2.sv	2-input OR gate
tt_libcell_sync2r_np.sv	2-stage synch using 2 clocks with async active-low reset
tt_libcell_sync2r.sv	2-stage synchronizer with async active-low reset
tt_libcell_sync2s.sv	2-stage synchronizer with async set
tt_libcell_sync2.sv	2-stage synchronizer
tt_libcell_sync3r.sv	3-stage synchronizer with async active-low reset
tt_libcell_sync3s.sv	3-stage synchronizer with async set
tt_libcell_sync3.sv	3-stage synchronizer
tt_libcell_sync4r.sv	4-stage synchronizer with async active-low reset
tt_libcell_sync4s.sv	4-stage synchronizer with async set
tt_libcell_sync4.sv	4-stage synchronizer
tt_libcell_xor2.sv	2-input XOR

Memory Files

Standalone files in the *source/integration* directory provide memories for each block are for reference. Some of these files have separate versions used specifically for the cluster physical design/simulation targets and the testbench target. Edit these files similarly to affect each target (**testbench**, **physical_design**, **simulation**).

Table 3 Memory Files

File	Description	Quantity	Configuration
tt_fe_mem_bimodal.sv	Table 0 of the branch direction predictor	2 banks	256x50b each
tt_fe_mem_cic.sv	Stores a few pre-decoded bits of the instruction cache	16 instances	Each 128x78b instance holds data corresponding to 16B of instructions for 2 ways of 128 total indices in the instruction cache.
tt_fe_mem_icache.sv	Data portion of the instruction cache	32 instances	Each 256x78b instance holds 4 of the 8 ways of data for 16B of instructions of 32 of the 128 total

			indices in the instruction cache.
`tt_fe_mem_itag.sv`	Tag portion of the instruction cache	8 instances	Each 64x92b instance holds 2 of the 8 ways of tag information for 64 of the 128 total indices in the instruction cache.
`tt_fe_mem_itlb_ram.sv`	Instruction TLB	1 instance	64x96b
`tt_fe_mem_ittage.sv`	Tables 1-5 of the branch target predictor	2 banks each, except where noted	• 128x78b each • 512x44b each • 512x44b each (4 banks) • 128x88b each 128x88b each
`tt_fe_mem_tage.sv`	Tables 1-6 of the branch direction predictor	Varies	• 512x44b each (2 banks) • 512x44b each (4 banks) 256x50b each (4 banks)
`tt_ls_me1_dcdata.sv`	Data portion of the Level 1 data cache	128 instances	Each 256x78b instance holds 2 of the 4 ways of data for a given word of memory.
`tt_ls_me1_dctag.sv`	Tag portion of the Level 1 data cache. 3 copies of the tag, one for each load pipeline	16 instances per tag copy	Each 256x50b instance holds 1 of the 4 ways of tag information for 256 of the 1024 total indices in the Level 1 data cache.
`tt_ls_me2_l2tlb.sv`	Level 2 TLB	12 instances each	• 256x72b each for the PA portion of the L2 TLB entries. 256x96b each for the VA portion of the L2 TLB entries.
`tt_ls_me2_pht.sv`	History table for the data prefetcher	6 instances	Each256x132b instance holds a portion of the prefetch history information.
`tt_sc_slice_mem.sv`	Cache regions, mapped to anything that is in DRAM region		8MB cache size
`tt_scb_mem.sv`	Hold temporary data for DRAM, SP, and MMR regions	Varies	1R+1W (2 ported) and: • 12 entries x 148bits 48 entries x 160 bits.

`tt_trace_mem_sink.sv`	Trace sink SRAM to capture signal trace information when signal tracing is enabled	8 instances	512x64 macros for a total of 32KB
`tt_TTCPLConfig_mem_0_ ext.sv`	CPL memory	4 instances	Single-port, no bit enable, 2048 entries, 74 bit entry size.

Each file contains an instantiation of **tt_rv_mem_model** that is defined in a separate *standalone source/ integration/tt_rv_mem_model.sv file*. You can modify either file as needed to instantiate memories that meet the tile's memory requirement.
Make the following pre-build changes to build and run with the technology swaps you made:

- Remove **+define+TT_BEHAVIORAL_MEM** from **$ROOT/design_verification/testbench/tt_testbench.f.**
- Remove **+define+TT_BEHAVIORAL_MEM** from **$ROOT/simulation/tt_cluster.f**.
- Remove **+define+TT_BEHAVIORAL_MEM** from **$ROOT/physical_design/tt_cluster.f**.

Top-Level Files

Top level files are provided in standalone files for integration and reference in the **source/integration** directory.

Table 4 Top-Level Files

File	Description
tt cluster.sv	**Cluster top-level file**
tt_cl_io.sv	**Cluster IO file**
tt_top_testbench.sv	**Testbench top-level file**
tt_cluster_harness_testbench.sv	**Testbench file that wraps the cluster**
tt_SC_fbchi_pkg.sv	**Package for shared cache interface with fabric CHI**
tt_SC_axi_pkg.sv	**Package for shared cache interface with AXI**

PLL Files

The TT-Ascalon Cluster includes the following files for integrating PLL, temperature sensor, and temperature hub modules in the **source/integration** directory except where noted.

Table 5 PLL Files

File	Description	Refer To
tt_rv_pll_model_generic.sv	PLL file (located in **source/primitives**)	See *"PLL*
tt_clock_controller.sv	PLL clock controller used in **tt_cluster**	
tt_cpl_top.sv	CPL wrapper file	

tt_cpl_pkg.sv	CPL package file	*Integration" on page 18*
tt_pll_controller.sv	PLL controller	
tt_pll_pkg.sv	PLL package	
tt_pll_wrapper.sv	PTL wrapper	

Table 6 Temperature Sensor Files

File	Description	Refer To
tt_rv_temp_sens_model_generic.sv	Temperature sensor file (located in source/primitives)	See *"Temperature Integrations" on page 24*.
tt_rv_thub_model_generic.sv	Temperature hub file (located in source/ primitives)	
tt_thub_controller.sv	Temperature hub controller	
tt_thub_ctrl_pkg.sv	Temperature hub control package	
tt_thub_top.sv	Wrapper file for temperature hub	

Make the following pre-build changes to build and run with the technology swaps you made:

- Add **+define+CUSTOM_PLL**, **+define+CUSTOM_TEMP_SENS**, and **+define+CUSTOM_TEMP_HUB** to **$ROOT/ design_verification/testbench/tt_testbench.f**.
- Add **+define+CUSTOM_PLL**, **+define+CUSTOM_TEMP_SENS**, and **+define+CUSTOM_TEMP_HUB** to **$ROOT/simulation/ tt_cluster.f**.

Add **+define+CUSTOM_PLL**, **+define+CUSTOM_TEMP_SENS**, and **+define+CUSTOM_TEMP_HUB** to **$ROOT/ physical_design/tt_cluster.f**.

Common Files

The TT-Ascalon Cluster RTL includes extra standalone files that ensure all signals from the standalone memory files are visible from the top level and have the form **tt_common_part*.sv**. Do not alter these files when integrating.

Cluster IO

The TT-Ascalon Cluster RTL includes extra standalone files that ensure all signals from the standalone memory files are visible from the top level and have the form **tt_common_part*.sv**.

Overview

Table 7 Cluster IO

File	Description
tt_rv_temp_sens_model_generic.sv	Temperature sensor file (located in source/primitives)
tt_rv_thub_model_generic.sv	Temperature hub file (located in source/ primitives)

`tt_thub_controller.sv`	Temperature hub controller
`tt_thub_ctrl_pkg.sv`	Temperature hub control package
`tt_thub_top.sv`	Wrapper file for temperature hub

Clocks

Connects to the external clock drivers.

Table 8 Clocks

Name	Direction	Description	Comments
i_rf_clk	Input	Reference clock	Fixed Frequency: 100MHz
i_fb_clk	Input	Fabric clock	Expected range: 1GHz – 1.6GHz
i_sc_clk	Input	SOC clock	Fixed Frequency: 400MHz

Resets

Connects to the external clock drivers.

Table 9 Resets

Name	Direction	Description	Comments
i_cluster_cold_reset_n	Input	Cluster cold reset	Async cluster reset
i_cluster_warm_reset_n	Input	CPL warm reset	
i_cluster_sram_hold	Input	SRAM hold	
i_cluster_debug_hold	Input	Debug logic hold	
i_cluster_critical_signal_hold	Input	Critical signal hold	
o_cluster_ndmreset_request	Output	NDM reset request	
i_cluster_ndmreset_process	Input	NDM reset progress	
i_force_ss_to_ref_clock_n	Input	Force all cluster clocks to ref clock during reset sequence	

CHI-Interface

Connects to the coherent network for both the odd/even interfaces.

Table 10 CHI Interface

Even Network Name	Odd Network Name	Direction	CHI Interface

o_coh_m0_syscoreq	**o_coh_m1_syscoreq**	Output	**SYSCOREQ**
i_coh_m0_syscoack	**i_coh_m1_syscoack**	Input	**SYSCOACK**
o_coh_m0_txsactive	**o_coh_m1_txsactive**	Output	**TXSACTIVE**
i_coh_m0_rxsactive	**i_coh_m1_rxsactive**	Input	**RXSACTIVE**
o_coh_m0_txlinkactivereq	**o_coh_m1_txlinkactivereq**	Output	**LINKACTIVEREQ**
i_coh_m0_txlinkactiveack	**i_coh_m1_txlinkactiveack**	Input	**LINKACTIVEACK**
o_coh_m0_txreqflitpend	**o_coh_m1_txreqflitpend**	Output	**TXREQFLITPEND**
o_coh_m0_txreqflitv	**o_coh_m1_txreqflitv**	Output	**TXREQFLITV**
o_coh_m0_txreqflit[N-1:0]	**o_coh_m1_txreqflit[N-1:0]**	Output	**TXREQFLIT[]**
i_coh_m0_txreqlcrdv	**i_coh_m1_txreqlcrdv**	Input	**TXREQLCRDV**
o_coh_m0_txrspflitpend	**o_coh_m1_txrspflitpend**	Output	**TXRSPFLITPEND**
o_coh_m0_txrspflitv	**o_coh_m1_txrspflitv**	Output	**TXRSPFLITV**
o_coh_m0_txrspflit[N-1:0]	**o_coh_m1_txrspflit[N-1:0]**	Output	**TXRSPFLIT[]**
i_coh_m0_txrsplcrdv	**i_coh_m1_txrsplcrdv**	Input	**TXRSPLCRDV**
o_coh_m0_txdatflitpend	**o_coh_m1_txdatflitpend**	Output	**TXDATFLITPEND**
o_coh_m0_txdatflitv	**o_coh_m1_txdatflitv**	Output	**TXDATFLITV**
o_coh_m0_txdatflit[N-1:0]	**o_coh_m1_txdatflit[N-1:0]**	Output	**TXDATFLIT[]**
i_coh_m0_txdatlcrdv	**i_coh_m1_txdatlcrdv**	Input	**TXDATLCRDV**
i_coh_m0_rxlinkactivereq	**i_coh_m1_rxlinkactivereq**	Input	**RXLINKACTIVEREQ**
o_coh_m0_rxlinkactiveack	**o_coh_m1_rxlinkactiveack**	Output	**RXLINKACTIVEACK**

AXI5-Lite-Interface

Connects to the NOC-N for both the Manger and Subordinate interfaces.

Table 11 AXI5-Lite Interface

Manager Port Name	Subordinate Port Name	Direction (Mgr/Sub)	AXI (NOC)	Comments
o_noc_m0_awvalid	**i_noc_s0_awvalid**	Output/Input	**AWVALID**	
i_noc_m0_awready	**o_noc_s0_awready**	Input/Output	**AWREADY**	
o_noc_m0_awid[11:0	**i_noc_s0_awid[9:0]**	Output/Input	**AWID**	• **Sub: 10-bit** • **Mgr: 12-bit]**
o_noc_m0_awaddr[51:0]	**i_noc_s0_awaddr[51:0]**	Output/Input	**AWADDR**	**52-bit address**
o_noc_m0_awsize[2:0]	**i_noc_s0_awsize[2:0]**	Output/Input	**AWSIZE**	**Optional feature for AXI5-Lite.**
o_noc_m0_awprot[2:0]	**i_noc_s0_awprot[2:0]**	Output/Input	**AWPROT**	• **[7:4] RCID** • **[3:0] SRCID**
o_noc_m0_awuser[7:0]	**i_noc_s0_awuser[7:0]**	Output/Input	**AWUSER**	
o_noc_m0_wvalid	**i_noc_s0_wvalid**	Output/Input	**WVALID**	

i_noc_m0_wready	o_noc_s0_wready	Input/Output	WREADY	
o_noc_m0_wdata[511:0]	i_noc_s0_wdata[511:0]	Output/Input	WDATA	
o_noc_m0_wstrb[63:0]	i_noc_s0_wstrb[63:0]	Output/Input	WSTRB	
i_noc_m0_bvalid	o_noc_s0_bvalid	Input/Output	BVALID	
o_noc_m0_bready	i_noc_s0_bready	Output/Input	BREADY	
i_noc_m0_bid[11:0]	o_noc_s0_bid[9:0]	Input/Output	BID	• Sub: **10-bit** • Mgr: **12-bit]**
i_noc_m0_bresp[1:0]	o_noc_s0_bresp[1:0]	Input/Output	BRESP	
o_noc_m0_arvalid	i_noc_s0_arvalid	Output/Input	ARVALID	
i_noc_m0_arready	o_noc_s0_arready	Input/Output	ARREADY	
o_noc_m0_arid[11:0]	i_noc_s0_arid[9:0]	Output/Input	ARID	• Sub: **10-bit** • Mgr: **12-bit]**
o_noc_m0_araddr[51:0]	i_noc_s0_araddr[51:0]	Output/Input	ARADDR	**52-bit address**
o_noc_m0_arsize[2:0]	i_noc_s0_arsize[2:0]	Output/Input	ARSIZE	**Optional feature for AXI5-Lite**
o_noc_m0_arprot[2:0]	i_noc_s0_arprot[2:0]	Output/Input	ARPROT	
o_noc_m0_aruser[7:0]	i_noc_s0_aruser[7:0]	Output/Input	ARUSER	• **[7:4] RCID** • **[3:0] SRCID**
i_noc_m0_rvalid	o_noc_s0_rvalid	Input/Output	RVALID	
o_noc_m0_rready	i_noc_s0_rready	Output/Input	RREADY	
i_noc_m0_rid[11:0]	o_noc_s0_rid[9:0]	Input/Output	RID	• Sub: **10-bit** • Mgr: **12-bit]**
i_noc_m0_rdata[511:0]	o_noc_s0_rdata[511:0]	Input/Output	RDATA	
i_noc_m0_rresp[1:0]	o_noc_s0_rresp[1:0]	**Input/Output**	RRESP	

AXI4-Interface

Connects to the dedicated power management AXI network for both the manager and subordinate interfaces.

Table 12 AXI4 Interface

Manager Port Name	Subordinate Port Name	Direction (Mgr/Sub)	AXI (NOC)	Comments
i_cpl_s0_awvalid	o_cpl_m0_awvalid	Input/Output	AWVALID	
o_cpl_s0_awready	i_cpl_m0_awready	Output/Input	AWREADY	
i_cpl_s0_awid[6:0]	o_cpl_m0_awid[6:0]	Input/Output	AWID	7-bit AXID
i_cpl_s0_awaddr[31:0]	o_cpl_m0_awaddr[31:0]	Input/Output	AWADDR	32-bit addr
i_cpl_s0_awlen[7:0]	o_cpl_m0_awlen[7:0]	Input/Output	AWLEN	
i_cpl_s0_awsize[2:0]	o_cpl_m0_awsize[2:0]	Input/Output	AWSIZE	
i_cpl_s0_awburst[1:0]	o_cpl_m0_awburst[1:0]	Input/Output	AWBURST	
i_cpl_s0_awprot[2:0]	o_cpl_m0_awprot[2:0]	Input/Output	AWPROT	

i_cpl_s0_awuser[3:0]	o_cpl_m0_awuser[3:0]	Input/Output	AWUSER	[3:0] SRCID
i_cpl_s0_wvalid	o_cpl_m0_wvalid	Input/Output	WVALID	
o_cpl_s0_wready	i_cpl_m0_wready	Output/Input	WREADY	
i_cpl_s0_wdata[63:0]	o_cpl_m0_wdata[63:0]	Input/Output	WDATA	
i_cpl_s0_wstrb[7:0]	o_cpl_m0_wstrb[7:0]	Input/Output	WSTRB	
i_cpl_s0_wlast	o_cpl_m0_wlast	Input/Output	WLAST	
o_cpl_s0_bvalid	i_cpl_m0_bvalid	Output/Input	BVALID	
i_cpl_s0_bready	o_cpl_m0_bready	Input/Output	BREADY	
o_cpl_s0_bid[6:0]	i_cpl_m0_bid[6:0]	Output/Input	BID	7-bit AXID
o_cpl_s0_bresp[1:0]	i_cpl_m0_bresp[1:0]	Output/Input	BRESP	
i_cpl_s0_arvalid	o_cpl_m0_arvalid	Input/Output	ARVALID	
o_cpl_s0_arready	i_cpl_m0_arready	Output/Input	ARREADY	
i_cpl_s0_arid[6:0]	o_cpl_m0_arid[6:0]	Input/Output	ARID	7-bit AXID
i_cpl_s0_araddr[31:0]	o_cpl_m0_araddr[31:0]	Input/Output	ARADDR	32-bit addr
i_cpl_s0_arlen[7:0]	o_cpl_m0_arlen[7:0]	Input/Output	ARLEN	
i_cpl_s0_arsize[2:0]	o_cpl_m0_arsize[2:0]	Input/Output	ARSIZE	
i_cpl_s0_arburst[1:0]	o_cpl_m0_arburst[1:0]	Input/Output	ARBURST	
i_cpl_s0_arprot[2:0]	o_cpl_m0_arprot[2:0]	Input/Output	ARPROT	
i_cpl_s0_aruser[3:0]	o_cpl_m0_aruser[3:0]	Input/Output	ARUSER	[3:0] SRCID
o_cpl_s0_rvalid	i_cpl_m0_rvalid	Output/Input	RVALID	
i_cpl_s0_rready	o_cpl_m0_rready	Input/Output	RREADY	
o_cpl_s0_rid[6:0]	i_cpl_m0_rid[6:0]	Output/Input	RID	7-bit AXID
o_cpl_s0_rdata[63:0]	i_cpl_m0_rdata[63:0]	Output/Input	RDATA	
o_cpl_s0_rresp[1:0]	i_cpl_m0_rresp[1:0]	Output/Input	RRESP	
o_cpl_s0_rlast	i_cpl_m0_rlast	Output/Input	RLAST	

External Interface

Configuration (Cluster ID) and interrupt.

Table 13 External Interface

Name	Direction	Comments
`i_cluster_id[3:0]`	Input	Unique cluster ID: - 3:2 - Refers to the Chiplet ID - 1:0 - Refers to the Cluster ID
`o_thub_tj_max`	Output	Abnormal thermal condition when temperature reaches Tj Max (abrupt shutdown).
`o_thub_tj_shutdown`	Output	Abnormal thermal condition when temperature reaches Tj Shutdown (graceful shutdown).
`o_ras_error_core[2:0]`	Output	RAS error observed in core.
`o_ras_error_sc[2:0]`	Output	RAS error observed in shared cache.

`o_ras_error_other[2:0]`	Output	RAS error observed in other units.
`i_nmi_valid[7:0]`	Input	Non-maskable interrupt

JTAG Interface

Connects to the system TAP.

Table 14 JTAG Interface

Name	Direction
`i_jtag_trstn`	Input
`i_jtag_tms`	Input
`i_jtag_tdi`	Input
`i_jtag_tck`	Input
`o_jtag_tdo`	Output
`o_jtag_tdo_en`	Output

System Memory Map

The TT-Ascalon testbench uses JSON-based system memory map files when running tests. The default memory map JSON file is saved to **$ROOT/design_verification/testbench/config/memmap_json_path/memmap.json**.

The system memory map contains one entry per region with the following fields:

- **type** – region type, which could be either memory or an IO device (e.g., **clint**, **htif**).
- **tag** – unique region name.
- **base** – region base address.
- **size** – region size in bytes.

Table 15 System Memory Map

Region	Description	Comments
`IO – M/S Level MMRs`	MMRs for 8 cores/SC/CPL etc.	
`IO – HTIF`	RISC-V host target interface for console emulation	
`IO – CLINT/ACLINT`	RISC-V CLINT/ACLINT devices for timer interrupts	
`IO – M/S Level Interrupt Files`	RISC-V AIA Interrupt Files	
`IO – Trickbox`	Platform specific region for custom DV traffic	MSIs, JTAG debug, entry/exit, etc.

PLL

PLL Block

The PLL0 and PLL1 blocks shown in Figure 2 are generic PLL models (**tt_rv_pll_model_generic**) that you can modify to integrate different vendor PLLs.

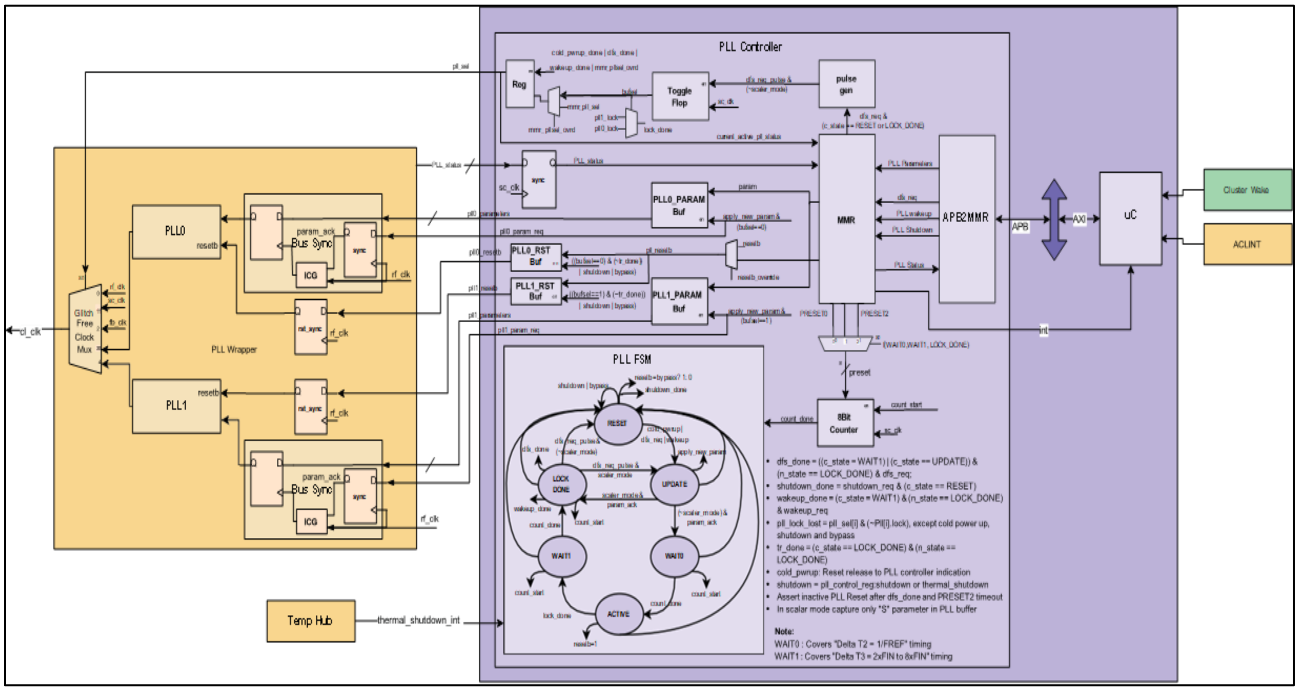


Figure 2 PLL Blocks

Interface Details

tt_rv_pll_model_generic

Table 16 tt_rv_pll_model_generic Interface

Interface Name	Dir	Bit Width	Default Value	ISO Value	Idle Value	Reset Value	Clock Domain	Reset Domain	Async/Sync	Desc.
rf_clk	In	1	1'b0				**rf_clk**			PLL reference clock

pll_clk	Out	1		1’b0	1’b0	1’b0	**cl_clk**			PLL clock output
reset_n	In	1	1’b0				**rf_clk**	**reset_n**	sync	Reference clock domain reset. Resets only in cold power-up.
pll_en	In	1	0				**rf_clk**	**reset_n**	sync	PLL enable control
pll_bypass	In	1	0				**rf_clk**	**reset_n**	sync	PLL bypass control
pll_parameters	In	***PLL_PARAM_ WIDTH***	0				**rf_clk**	**reset_n**	sync	PLL parameters and control signals
pll_status	In	***PLL_STATUS_ WIDTH***		0	0	0	**rf_clk**	**reset_n**	sync	PLL status info
tdr_select	In	*1*	1’b0				**tck**	**trstn**	sync	tdr select
tdr_mode	In	*1*	1’b0				**tck**	**trstn**	sync	tdr mode
tdr_clk	In	*1*	1’b0				**tck**	**trstn**	sync	tdr clk
tdr_rst_n	In	*1*	1’b0				**tck**	**trstn**	sync	tdr rst n
tdr_capture	In	*1*	1’b0				**tck**	**trstn**	sync	tdr capture
tdr_shift	In	*1*	1’b0				**tck**	**trstn**	sync	tdr shift
tdr_update	In	*1*	1’b0				**tck**	**trstn**	sync	tdr update
tdr_readback	In	*1*	1’b0				**tck**	**trstn**	sync	tdr readback
tdr_in	In	*1*	1’b0				**tck**	**trstn**	sync	tdr in
tdr_out	Out	*1*					**tck**	**trstn**	sync	tdr out
scan_mode	In	*1*	1’b0				**tck**		sync	scan mode
scan_clk	In	*1*	1’b0				**tck**		sync	scan clk
scan_en	In	*1*	1’b0				**tck**		sync	scan en
scan_in	In	***NUM_PLL_SCAN_CHAINS***	1’b0				**tck**		sync	scan in
scan_out	Out	***NUM_PLL_SCAN_CHAINS***					**tck**		sync	scan out

tt_pll_wrapper

Table 17 tt_pll_wrapper interface

Interface Name	Dir	Bit Width	Default Value	ISO Value	Idle Value	Reset Value	Clock Domain	Reset Domain	Async/ Sync	Desc.

sc_clk	In	1	1'b0				**sc_clk**			400MHz external clock; used for the entire CPL except PMNW master.
fb_clk	In	1	1'b0				**fb_clk**			1.8GHz external clock; used for fabric clock domain.
rf_clk	In	1	1'b0				**rf_clk**			100MHz external clock; used for reset controller.
cl_clk	Out	1		1'b0	1'b0	1'b0	**cl_clk**			2.7GHz internal clock; for PMNW master.
tb_cl_clk	In	1	1'b0				**tb_cl_clk**		-	TB generated 2.7GHz internal clock; supports verilator and Zebu.
rf_cold_ reset_n	In	1	1'b0				**rf_clk**	**rf_cold_reset_n**	sync	Reference clock domain reset. Resets only in cold power-up.
tb_pll_lock	In	1	1'b0				**rf_clk**	**rf_cold_reset_n**	-	TB generated PLL lock status; supports verilator and Zebu.
pll_sel	In	**clog2(*NUM_PLL*)**	1'b0				**rf_clk**	**rf_cold_reset_n**	async	PLL selection control
CPL_PLL_ Control	In	***[PLL_CPL_ CONTROL_WIDTH] [NUM_PLL]***	0				**rf_clk**	**rf_cold_reset_n**	async	PLL parameters and control signals
PLL_CPL_ Status	Out	***[PLL_CPL_ STATUS_WIDTH] [NUM_PLL]***		0	0	0	**rf_clk**	**rf_cold_reset_n**	async	PLL status info
tdr_select	In	1	1'b0				**tck**	**trstn**	sync	tdr select
tdr_mode	In	1	1'b0				**tck**	**trstn**	sync	tdr mode

tdr_clk	In	1	1'b0				**tck**	**trstn**	sync	tdr clk
tdr_rst_n	In	1	1'b0				**tck**	**trstn**	sync	tdr rst n
tdr_capture	In	1	1'b0				**tck**	**trstn**	sync	tdr capture
tdr_shift	In	1	1'b0				**tck**	**trstn**	sync	tdr shift
tdr_update	In	1	1'b0				**tck**	**trstn**	sync	tdr update
tdr_readback	In	1	1'b0				**tck**	**trstn**	sync	tdr readback
tdr_in	In	1	1'b0				**tck**	**trstn**	sync	tdr in
tdr_out	Out	NUM_PLL					**tck**	**trstn**	sync	tdr out
scan_mode	In	1	1'b0				**tck**		sync	scan mode
scan_clk	In	1	1'b0				**tck**		sync	scan clk
scan_en	In	1	1'b0				**tck**		sync	scan en
scan_in	In	NUM_PLL_ SCAN_CHAINS	1'b0				**tck**		sync	scan in
scan_out	Out	NUM_PLL_ SCAN_CHAINS] NUM_PLL								scan out

tt_pll_controller

Table 18 tt_pll_controller interface

Interface Name	Dir	Bit Width	Default Value	ISO Value	Idle Value	Reset Value	Clock Domain	Reset Domain	Async / Sync	Desc.
`sc_clk`	In	1	0				`sc_clk`			400MHz external clock; used for entire CPL except PMNW master
`sc_cold_reset_n`	In	1	0				`sc_clk`		sync	SOC clock domain cold reset
`force_ss_to_ref_clock_n`	In	1	0				`sc_clk`		async	Control to force all clocks to reference clock
`pll_interrupts_in`	In	***NUM_PLL_INT_IN***	0						async	PLL input interrupts
`pll_interrupts_out`	Out	***NUM_PLL_INT_OUT***		0	0	0			sync	PLL output interrupts
`paddr`	In	***APB_ADDR_WIDTH***	0				`sc_clk`	`sc_cold_reset_n`	sync	APB address bus
`pprot`	In	3	0				`sc_clk`	`sc_cold_reset_n`	sync	APB protection
`psel`	In		0				`sc_clk`	`sc_cold_reset_n`	sync	APB sel

penable	In		0				**sc_clk**	**sc_cold_reset_n**	sync	APB enable
pwrite	In		0				**sc_clk**	**sc_cold_reset_n**	sync	APB write
pwdata	In	APB_DATA_WIDTH	0				**sc_clk**	**sc_cold_reset_n**	sync	APB write data
pstrb	In	APB_DATA_WIDTH/8	0				**sc_clk**	**sc_cold_reset_n**	sync	APB strobe
pready	Out			0	0	0	**sc_clk**	**sc_cold_reset_n**	sync	APB ready
prdata	Out	APB_DATA_WIDTH		0	0	0	**sc_clk**	**sc_cold_reset_n**	sync	APB read data
pslverr	Out			0	0	0	**sc_clk**	**sc_cold_reset_n**	sync	APB slave error
pll_sel	Out	PLL_SEL_WIDTH		0	0	0	**sc_clk**	**sc_cold_reset_n**	sync	PLL selection control
PLL_CPL_Status	In	PLL_CPL_STATUS_WIDTH	0				**sc_clk**	**sc_cold_reset_n**	async	PLL status info
CPL_PLL_Control	Out	CPL_PLL_CTRL_WIDTH		0	0	0	**sc_clk**	**sc_cold_reset_n**	async	PLL parameters and control signals

PLL Swapping

This section describes two PLL swapping use cases.

Case 1: PLL Controller Sequences Matches Swapped PLL Requirements

- Update the following RTLs:
  - **source/primitives/tt_rv_pll_model_generic.sv**
  - **source/integration/tt_pll_pkg.svh**
- Update **source/primitives/tt_rv_pll_model_generic.sv**:
  - Instantiate the required PLL model under **ifdef CUSTOM_PLL**.
  - Define **CUSTOM_PLL** as part of the compile.
  - Integrate PLL interfaces with **tt_rv_pll_model_generic i**nterface:
    - > Connect **ref_clk** to the PLL reference clock interface and **pll_clk** to PLL clock out.
    - > Use the **pll_parameters** input interface (driven from the PLL controller MMR) to control PLL parameters. Max supported PLL parameters: 125 bits.
    - > Use the **`pll_status`** output interface (mapped to the PLL controller MMR) to capture the PLL status. x supported PLL status: 15 bits.
    - > Tie off unused control inputs.
    - > Open unused status outputs.
- Update **`source/integration/tt_pll_pkg.svh`**:
  - Configure the default frequency by defining local PLL parameters that define PoR values in accordance with the PLL requirement.

Case 2: Update PLL Controller Sequences for Swapped PLL

- Update the RTL:
  - **source/primitives/tt_rv_pll_model_generic.sv**
  - **source/integration/tt_pll_pkg.svh**
  - **source/integration/tt_pll_wrapper.sv**
  - **source/integration/tt_clock_controller.sv**
  - **source/integration/tt_cluster.sv**
  - **source/integration/tt_cpl_top.sv**
  - **source/integration/tt_pll_controller.sv**
- Update **source/integration/tt_pll_controller.sv**:
  - Develop the PLL controller per the PLL sequence requirements.
  - Generate the required MMRs (**cc_csr** module) and hook them to the APB interface.
  - Design requirements:
    - > Drive **pll_sel** to select **rf_clk** when **force_ss_to_ref_clock_n** is set to **0**, else select the required PLL clock.
    - > Set pll_inactive to **1** when PLLs are shutting down or waking up to prevent lock lost interrupt to CPL.
    - > Update **pll_interrupts_in** and **pll_interrupts_out** based on interrupt mapping.
    - > Take care of all CDC crossings between the **pll_controller** (in the **SC_CLK** domain) and

tt_pll_wrapper (in the **rf_clk domain**).

- Update **source/integration/tt_pll_pkg.svh**:

Update/define required parameters per the **tt_pll_controller** and **tt_rv_pll_model_generic** changes.

- Update `source/integration/tt_pll_wrapper.sv`:
  - Update the number of PLL instantiations.
  - Update the **tt_rv_clk_mux_glitch_free** instantiation based on the number of PLL instantiations.
- Update **source/primitives/tt_rv_pll_model_generic.sv**:
  - PLL Instantiation
  - Integration updates
- Update **source/integration/tt_cpl_top.sv**, **source/integration/ tt_cluster.sv**, and **source/integration/tt_clock_controller.sv**:
  - Integration updates per the new interface additions in the

tt_pll_controller and **tt_rv_pll_model_generic** modules.

Temperature Integrations

This section describes how to integrate the Temperature Hub, Temperature Sensor, and Temperature Controller.

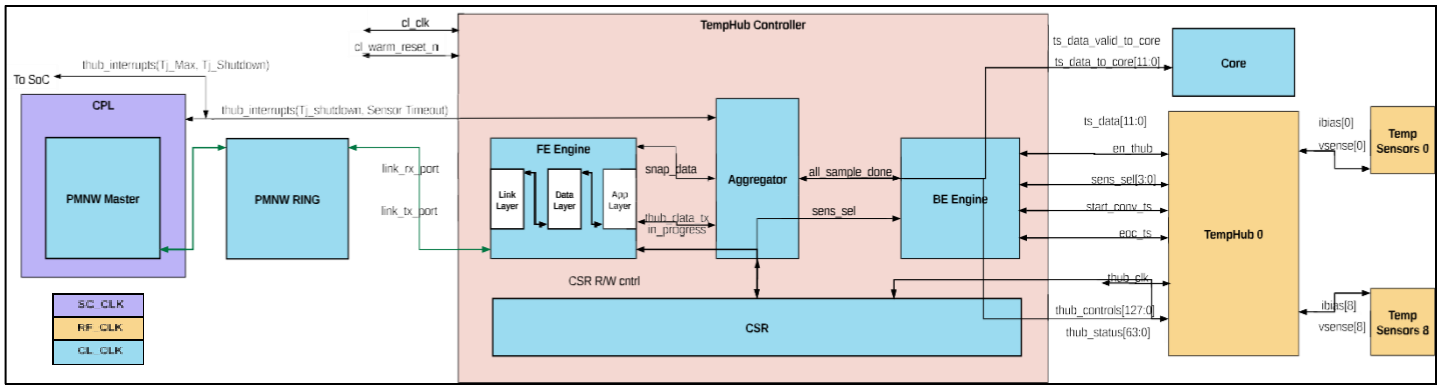


Figure 3 Temperature Integration Block Diagram

Interface Details

thub_controller

Table 19 thub_controller Interface

Interface Name	Dir	Bit Width	Default Value	ISO Value	Idle Value	Reset Value	Clock Domain	Reset Domain	Async / Sync	Desc.
i_cl_clk	In	1	1'b0				**cl_clk**			2.7GHz internal clock; for PMNW master
i_cl_warm_reset_n	In	1	1'b0				**cl_clk**		sync	Cluster clock domain reset. Resets both in cold and warm power-up.
i_link_rx_port	In	**link_t**	0				**cl_clk**	**cl_warm_reset_n**	sync	PMNW ring input.
o_link_tx_port	Out	**link_t**		0	0	0	**cl_clk**	**cl_warm_reset_n**	sync	PMNW ring output.
o_thub_interrupts	Out	***NUM_THUB_INT_OUT***		0	0	0	**cl_clk**	**cl_warm_reset_n**	async	Temperature Hub interrupts: **Tj_Max** and **Tj_Shutdown**
o_en_thub	Out	1		0	0	0	**cl_clk**	**cl_warm_reset_n**	async	Temperature Hub enable control.
o_sens_sel	Out	4		0	0	0	**cl_clk**	**cl_warm_reset_n**	async	Sensor selection control to Temperature Hub.

o_start_conv_ts	Out	1		0	0	0	**cl_clk**	**cl_warm_reset_n**	async	Conversion trigger to Temperature Hub.
i_eoc_ts	In	1	0				**cl_clk**	**cl_warm_reset_n**	async	End of conversion response from Temperature Hub.
i_ts_data	In	***SENS_DATA_WIDTH***	0				**cl_clk**	**cl_warm_reset_n**	async	Converted Temperature Sensor data.
o_thub_controls	Out	128		0	0	0	**cl_clk**	**cl_warm_reset_n**	async	Spare Temperature Hub controls from CSR for generic usage.
o_thub_controls_valid	**Out**	**1**		**0**	**0**	**0**	**cl_clk**	**cl_warm_reset_n**	**async**	Valid indication Temperature Hub controls.
i_thub_controls_ready	**In**	**1**	0				**cl_clk**	**cl_warm_reset_n**	**async**	Ready indication from Temperature Hub after sampling controls.
o_thub_fuse_controls	**Out**			**0**	**0**	**0**	**cl_clk**	**cl_warm_reset_n**	**async**	Fuse controls for Temperature Hub.
i_thub_status	**In**	**64**	0				**cl_clk**	**cl_warm_reset_n**	**async**	Spare Temperature Hub status to CSR for generic usage. Multibit sync assumed mutually exclusive individual status bits.

o_ts_data_to_core	Out	SENS_DATA_WIDTH		0	0	0	cl_clk	cl_warm_reset_n	sync	Aggregated Temperature Sensor data to core.
o_ts_data_valid_to_core	Out	1		0	0	0	cl_clk	cl_warm_reset_n	sync	Temperature Sensor data valid indication to core.

tt_rv_thub_model_generic

Table 20 tt_rv_thub_model_generic Interface

Interface Name	Dir	Bit Width	Default Value	ISO Value	Idle Value	Reset Value	Clock Domain	Reset Domain	Async/ Sync	Desc.
i_clk	In	1	0				rf_clk			100MHz external clock; used for Reset Controller.
i_reset_n	In	1	0				cl_clk		sync	Cluster clock domain reset. Resets both in cold and warm power-up.
i_en_thub	In	1	0				cl_clk	cl_warm_reset_n	async	Temperature Hub enable control.
i_sens_sel	In	*SENS_SEL_WIDTH*	0				cl_clk	cl_warm_reset_n	async	Sensor selection control to Temperature Hub.
i_start_conv_ts	In	1	0				cl_clk	cl_warm_reset_n	async	Conversion trigger to Temperature Hub.
o_eoc_ts	Out	1		0	0	0	cl_clk	cl_warm_reset_n	async	End of conversion response from Temperature Hub.
o_ts_data	Out	*SENS_DATA_WIDTH*		0	0	0	cl_clk	cl_warm_reset_n	async	Converted Temperature Sensor data.

i_thub_controls	In	*THUB_CTRL_WIDTH*	0				**cl_clk**	**cl_warm_ reset_n**	async	Spare Temperature Hub controls from CSR; for generic usage.
i_thub_ controls_valid	In	1	0				**cl_clk**	**cl_warm_ reset_n**	async	Valid indication Temperature Hub controls.
o_thub_ controls_ready	Out	1		0	0	0	**cl_clk**	**cl_warm_ reset_n**	async	Ready indication from Temperature Hub after sampling controls.
i_thub_fuse_ controls	In	*THUB_FUSE_ CTRL_WIDTH*	0				**cl_clk**	**cl_warm_ reset_n**	async	Fuse controls for Temperature Hub.
o_thub_status	Out	64		0	0	0	**cl_clk**	**cl_warm_ reset_n**	async	Spare Temperature Hub status to CSR for generic usage. Multi-bit sync assumes mutually exclusive individual status bits.
i_scan_en	In	1	0							DFT scan enable.
i_scan_in	In	1	0						sync	DFT scan in.
o_scan_out	Out	1		0	0	0			sync	DFT scan out.
ibias_ts	In/ Out	***NUM_SENS_PER_ HUB***							sync	**ibias** from sensor.

vsense_ts	In/ Out	*NUM_SENS_PER_ HUB*								**vsense** from sensor.
test_out_ts	In/ Out	1								Analog test bus.
vol_ts	In/ Out	14								Voltage sensor input; measures voltage.

Swapping the Temperature Hub

This section describes two use cases when swapping the Temperature Hub.

Case 1: Matching Controller Sequences

- Update the **source/primitives/tt_rv_thub_model_generic.sv RTL**.
- Update **source/primitives/tt_rv_thub_model_generic.sv**:
  - Instantiate in **`ifdef CUSTOM_TEMP_HUB**.
  - Define **CUSTOM_TEMP_HUB** as part of the compile.
  - Integrate **THUB** interfaces with the **tt_rv_thub_model_generic** interface.
    - Use the **thub_controls_to_hub** input interface (driven from the Temperature Hub controller MMR) to control Temperature Hub controls. Max supported PLL parameters: 127 bits
    - Use the thub_status output interface (mapped to the Temperature Hub controller MMR) to capture the Temperature Hub status. Max supported PLL status: 63 bits
    - Use the **thub_fuse_controls** input interface (driven from the Temperature Hub controller MMR) to control the Temperature Hub fuse. Max supported Temperature Hub fuses: 62 bits
    - Tie off unused control inputs.
    - Open the unused status output.

Case 2: Swapping Controller Sequences

- Update the following RTLs:
  - **source/primitives/tt_rv_thub_model_generic.sv**
  - **source/integration/tt_thub_top.sv**
  - **source/integration/tt_thub_controller.sv**
  - **source/integration/tt_thub_ctrl_pkg.svh**
- Update **source/integration/tt_thub_controller.sv** and **source/integration/tt_thub_ctrl_pkg.svh**

based on the Temperature Hub requirements.

- Integrate the Temperature Hub in **source/primitives/tt_rv_thub_model_generic.sv**.

Update the interface changes in **source/integration/tt_thub_top.sv**.

rv_temp_sens_model_generic

tt_rv_temp_sens_model_generic consists of third-party Temperature Sensors. You can update this model based on the third-party IP, interface, and connectivity to the Temperature Hub.

Testbench

The drop includes a testbench that interfaces with and tests a TT-Ascalon Cluster.

Requirements

Verify that your test system meets the following requirements:

- **Supported EDA tools:** See *“Verified EDA Tools” on page 2*.
- **Environment:** See *“Environment Requirements” on page 2*.
- **Container:** See *“Using a Container (Optional)” on page 3* if you are using a container to run your tests.

Building the Testbench

Execute the following command to build the testbench:
make build TARGET=testbench TOOL=<sim> TYPE=<model type>

Where **TYPE** denotes the build type for the target:

- **opt** - for optimized models that build and run quickly.
- **dbg** - for debug models that support dumping waveforms.

Note: The Xcelium debug model requires an existing Verdi installation with $VERDI_HOME properly set in the environment in order to dump waveforms.
If the testbench fails to build, then you can check the full build log located in **$ROOT/design_verification/testbench/build/<sim>/<type>/tt_testbench_<sim>_<type>_build.log**.

Running the Testbench

Running a Testlist

Execute the following commands to run a Testlist (see *“Tests” on page 35* for all included testlists):
make build TARGET=testbench TOOL=<sim> TYPE=<model type> (if not already built) make testlist TARGET=testbench TOOL=<sim> TYPE=<model type> TESTLIST=<testlist>

Using LSF (Optional)

The **design_verification/testbench/run_testlist.sh** script used in the **make** flow can launch tests using LSF but may be subject to varying environment requirements. Add the **bsub** arguments used in **run_testlist.sh** by setting and exporting **TT_EXTRA_LSF_OPTS="<opts>"** before running. The LSF **stdout** will be saved to **$ROOT/design_verification/testbench/runs/<test group>/<test name>_<sim>/lsf.log**. Set **LSF=true** in the **make** command to run the Testlist with LSF.

Running an Individual Test

Execute the following command to run an individual test:

make test TARGET=testbench TOOL=<sim> TYPE=<model type> TESTLIST=<testlist> TEST='<test args>'

The test executes in a separate run directory under **$ROOT/design_verification/testbench/runs/<TESTLIST>/ <testname>_<sim>_<model type>** with a **run.log**. If a test fails to launch or generate, then all command output will also be available in **$ROOT/design_verification/testbench/runs/<TESTLIST>/ <test>_<sim>_<model_type>/launch.log**. If a test behaves unexpectedly, then please provide the contents of the corresponding runs/ directories to Tenstorrent for debug via your Tenstorrent Customer Success team member.

Checking Test Results

The **testlist** command prints the status of each test as it runs and can also be viewed in result.json in the run directory. You should expect some tests to fail. These tests will start with **expected_fail_**. **design_verification/testbench/test_status.sh** checks the result statuses for all tests under a given directory. By default, **test_status.sh** prints all test failures beneath the current working directory. You can check test results from a different directory by calling **test_status.sh <directory>**. You can also print the status of both passing and failing tests by calling t**est_status.sh <directory> true**. Printing test statuses displays the test name, status, and any applicable failure message. For example:

```
design_verification/testbench/test_status.sh $ROOT/design_verification/testbench/runs/simple_tests true
Checking status of tests in / testbench dhrystone_100_iter_vcs          [ Pass ]:
Directory: /design_verification/testbench/runs/dhrystone/dhrystone_100_iter_vcs expected_fail_test_1_vcs
            [ Fail ]:
Directory: /design_verification/testbench/runs/simple_tests/expected_fail_test_1_vcs Message: "Simulation
had error --
Error: Failed stop: Hart 0: write to to-host: 3 "
hello_world_vcs  [ Pass ]:
Directory /design_verification/testbench/runs/simple_tests/hello_world_vcs test_1_vcs          [ Pass ]:
Directory /design_verification/testbench/runs/simple_tests/test_1_vcs
```

Manually Running a Test Outside of Provided Flows

Execute the **export RUNFILES_DIR=$ROOT/design_verification/testbench** command prior to executing other commands to manually run a test outside of the **make** flow, **run_test.sh**, or **run_testlist.sh** scripts.

Compiling and Running a Custom C Test

The TT-Ascalon Cluster package provides support for compiling custom C tests via the RISC-V GNU toolchain and running them on the TT-Ascalon Cluster testbench. To do this:

1. cd $ROOT/design_verification
2. Install the RISC-V GNU toolchain (See riscv-gnu-toolchain* for environment requirements and installation steps).
3. $ROOT/design_verification/testbench/cc_test_generator -n <name> -s <source test c file>
4. make test TARGET=testbench TOOL=<sim> TYPE=<model type> TESTLIST=<directory name to use> TEST='<test args>'

The TT-Ascalon Cluster includes the following sample C test file: $ROOT/design_verification/tests/ simple_tests/hello_world_debug.c.

Dumping Waveforms

Dump waveforms by setting the TYPE for builds and testlist/test running to dbg, then add +fsdb_cycle_on=<cycle number> and (optionally) +fsdb_cycle_off=<cycle number> to the testlist(s) that contain the desired test(s) to denote the start and end cycles to dump the FSDB.

Viewing Design Hierarchy and Waveforms

You can use Verdi to open waveforms using the output present in the dbg build directory for a given target by executing the verdi -dbdir <path/to/target_simv.daidir> command.

Tests

The TT-Ascalon cluster includes the following tests:

• Simple: Execute the following command to run a series of short tests as a sanity check. This includes a test that is expected to fail:

make testlist TOOL=<sim> TARGET=testbench TESTLIST=simple

• Benchmark: This testlist contains only a dhrystone test. Execute the following command:

make testlist TOOL=<sim> TARGET=testbench TESTLIST=benchmark

• Compatibility: This testlist containing a RISC-V compatibility testlist that includes thousands of tests. Execute the following command:

make testlist TOOL=<sim> TARGET=testbench TESTLIST=compatibility

• Feature: This testlist tests TT-Ascalon-specific RISC-V features. Execute the following command:

make testlist TOOL=<sim> TARGET=testbench TESTLIST=feature

Integration Targets

The TT-Ascalon Cluster includes two versions of the simulation and physical design target for the TT-Ascalon Cluster:

- simulation with defines and configuration for use with simulators
- physical_design with defines and configurations for use with synthesis tools.

Requirements

Verify that your integration system meets the following requirements:

- Supported EDA tools: See “Verified EDA Tools” on page 2.
- Environment: See “Environment Requirements” on page 2.
- Container: See “Using a Container (Optional)” on page 3 if you are using a container to run your tests.

Building Integration Targets

Execute the following command to build integration targets:

```
make build TARGET=<target> TOOL=<sim> TYPE=<model type>
```

Where:

- TARGET denotes which target to build for:
- simulation
- physical_design
- TYPE denotes the target build type:
- opt is for optimized models that build and run quickly.
- dbg is for debug models that support dumping waveforms.

If the target fails to build, then you can find the full build log in $ROOT/<target>/build/<sim>/<type>/ tt_cluster_<sim>_<type>build.log.

Linting Integration Targets

Execute the following command to lint your integration target:

```
make lint TOOL=<tool>
```

The TOOL variable denotes which linting tool to use (spyglass).
The lint tool only runs on the physical design target; the TARGET option is not used.
You can find Spyglass reports in $ROOT/cluster/physical_design/cluster_lint_spyglass/.

CDC Constraints

CDC constraints for use with Synopsys VC Static clock domain crossing tools are provided under $ROOT/cluster/physical_design/cdc/.

Whisper

Whisper is a reference RISC-V model. The Whisper executable is included in design_verification/whisper/ whisper. The provided TT-Ascalon Cluster testbench natively incorporates whisper for co-simulation checking, but Whisper can also run standalone binaries, as described in the Whisper User Guide (Markdown format) located at design_verification/whisper/README.md.

Linux

design_verification/whisper/linux/ provides a Linux Whisper image. Execute the following commands to boot Linux with Whisper:

```
$ROOT/design_verification/whisper/whisper --configfile $ROOT/design_verification/whisper/linux/ whisper.json --
target $ROOT/design_verification/whisper/linux/bootrom.elf --target $ROOT/
design_verification/whisper/linux/fw_payload.elf --fromhost 0x70000000 --tohost 0x70000008 -- quitany --raw
```

RTOS

design_verification/whisper/rtos/ provides a binary RTOS for use with Whisper. Execute the following commands to run RTOS with Whisper:

```
$ROOT/design_verification/whisper/whisper --configfile $ROOT/design_verification/whisper/rtos/ whisper.json -s
0x10000 $ROOT/design_verification/whisper/rtos/bootrom.elf $ROOT/
design_verification/whisper/rtos/RTOSDemo64.axf
```

RTOS boots in less than a second, prints the message Hello FreeRTOS!, and then spins waiting for a timer interrupt. Press [CTRL]+[C] to stop the run. You can also configure Whisper to send a timer interrupt to RTOS every 20 microseconds by executing the following commands:

```
$ROOT/design_verification/whisper/whisper --configfile $ROOT/design_verification/whisper/rtos/ whisper.json -s
0x10000 $ROOT/design_verification/whisper/rtos/bootrom.elf $ROOT/
design_verification/whisper/rtos/RTOSDemo64.axf –alarm 20
```

This causes RTOS to keep sending a message to the sample application. The output will indicate sent and received messages. Press [CTRL]+[C] to stop the run.