Microsoft

Catapult Academic User Guide

Date: July 23, 2017
Version: V1.2
Author: Dan Zhang

Revision History

Revision/Date	Notes	Updated by
V1.0 **2/28/2016**	Initial version intended for SDK v1.1	Dan Zhang
V1.2 **7/23/2017**	Updated for SDK v1.2. Skipped v1.1 to sync version to SDK.	Dan Zhang

1 Introduction

Project Catapult is an open research system deployed at TACC in partnership with Microsoft Research. The goal of the project is to investigate the use of field-programmable gate arrays (FPGAs) as data center accelerators to improve performance, reduce power consumption, and open new research avenues of investigation, particularly in Machine Learning. Catapult is currently deployed in production Microsoft datacenters accelerating the Bing search engine.

This document provides an overview of the Catapult SDK and how to develop and debug Catapult applications. For a detailed description of the Catapult shell, read the Catapult Mt Granite Architectural Specification. To learn how to setup, compile, and run a Hello World application on the Catapult machines at TACC, read the Catapult TACC Getting Started Guide.

1.1 Mt Granite FPGA Board Interfaces

Mt Granite contains an Altera Stratix V 5SGSMD5H2F35 and two channels of DDR3 533MHz, each providing 4GB for a total of 8GB. The card connects to the server via eight lanes of Gen3 PCI Express (PCIe), but also has four pairs of SerialLite III connections, routed through two standard mini-SAS connectors.

The PCIe Gen 3.0 8x interface is abstracted with a slot-based interface in software and 128-bit input/output FIFOs in FPGA user logic. It is capable of a peak theoretical bandwidth of around 4GB/s each direction, 8GB/s bi-directional. In real-world tests, we can achieve roughly 3GB each direction, 6GB/s bi-directional. We also provide a 64-bit read/write Soft Register interface for configuration, monitoring, and debugging through PCIe PIO, which supports low bandwidth ~1MB/s.

The two channels of DDR3 are accessible by the FPGA only. The FPGA sends DRAM read/write requests through a 512-bit split-phase interface. Each channel has a peak theoretical bandwidth of 10.7GB/s. With real-world tests, we observe ~5GB/s for sequential writes and ~7GB/s for sequential reads per channel.

Mt Granite boards are connected in a 6 by 8 torus network, each board with 4 full-duplex point-to-point connections through SerialLite III (SL3). Each SL3 connection is represented in FPGA user logic as 128-bit incoming and outgoing FFIOs. Each link can theoretically achieve 766MB/s in each direction, for 4*2*766 = 6.28GB/s of peak aggregate network bandwidth.

1.2 Software Development Kit (SDK) Version 1.2 Changes

Academic Software Development Kit (SDK) v1.2 adds two major features: SerialLite III support with examples, and a simple functional-only co-simulation framework. To support the co-simulation framework, minor changes were made to the SDK folder hierarchy, and software library files were moved to a new Common directory. In addition, Shell infrastructure and library modules have been updated to the latest in Microsoft's internal development tree.

2 Catapult SDK v1.2 Overview

The Catapult SDK v1.2 release is split into separate Microsoft and Altera portions due to IP distribution issues. You must combine both portions by unzipping both ZIP files in the same directory before you can synthesize the design. For directions on installing the Catapult SDK, refer to the Catapult TACC Getting Started Guide.

The Catapult SDK release contains four major components: the driver, the shell, the example user roles, and the example software. The shell is a reusable portion of programmable logic common across applications, while the role is the application logic itself, restricted to a large fixed region of the chip. Developing a Catapult applications has two parts: writing the user role in SystemVerilog, and writing the software application that interfaces with the role in Microsoft Visual Studio C++.

- **Driver**: The Catapult Windows drivers, FPGA programming utilities, and software header files and libraries.
- **Roles**: The Catapult user roles. Currently contains the golden role and several example roles demonstrating PCIe and DRAM. Add your roles here.
    - **AcademicPCIeLoopback**: Loopback test example demonstrating PCIe. Stores data from PCIe->FPGA into a queue, then sends the same data back out from FPGA->PCIe.
    - **AcademicDRAMLoopback**: Loopback test example demonstrating DRAM and PCIe. Loops data from FPGA -> PCIe -> DRAM -> PCIe -> FPGA.
    - **AcademicGolden**: Baseline “golden” image.
    - **AcademicSL3Test**: Example test demonstrating SerialLite 3. Sends data from a sending machine to a receiving machine, through PCIe->FPGA->SL3->FPGA->PCIe.
    - **Common**: Common library files (encrypted).
    - **Sim**: Common scripts for the co-simulation framework.
- **Shells**: The Catapult Academic shell supporting the Mt Granite board. Many of the shell components are encrypted to protect Microsoft IP. **Do not modify the shell!**
- **Software**: Software for interfacing with the user role.
    - **LoopbackStressTest**: Stress test for measuring PCIe bandwidth. Demonstrates multiple ways to interface with the slot-based PCIe mechanism to achieve high bandwidth. The test is compatible with both AcademicPCIeLoopback and AcademicDRAMLoopback.
    - **SL3StressTest**: Stress test for measuring SL3 bandwidth. Demonstrates how to properly configure Shell registers for SL3, and performs a simple send/receive test over a directly connected machine pair. The test is only compatible with AcademicSL3Test.
    - **Common**: Common library files. Contains the PCIe job dispatcher and co-simulation files.
- **CHANGELOG.txt**: Changelog with the changes for each Catapult SDK version.
- **MSFTAcademic.dat**: Quartus decryption license for Catapult SDK encrypted files.
- **MSR_License_Agreement.pdf**: Catapult SDK licensing agreement.

3 Catapult Role Overview

The Catapult SDK contains several sample roles that demonstrate various aspects of how to interface with the Catapult Shell. In addition, there is an encrypted golden role that can be used to generate a golden image to recover the Mt Granite board in case a bad FPGA image is programmed.

In Catapult SDK v1.1, for user convenience and ease of development, we introduced a simplified role interface called SimpleRole, which wraps the original Role interface. Catapult SDK v1.2 extends SimpleRole by adding SL3 support. This user guide presents the updated SimpleRole interface and assumes SimpleRole will be used in your custom Catapult applications. Although using SimpleRole is optional since it's simply a module instantiated within the original Role, we highly recommend its use to reduce design complexity and improve productivity. In addition, the new co-simulation framework introduced in SDK v1.2 only supports SimpleRole. For details on the original Role interface and for low-level details on each interface, read the Catapult Mt. Granite Architectural Specification.

3.1 SimpleRole

SimpleRole attempts to simplify the Role interface by encapsulating interfaces within structs, simplifying and standardizing flow control, and removing uncommonly used signals. To maintain backwards compatibility, rather than replacing the Role interface itself, we place all wrapper logic within the Role and instantiate SimpleRole within the Role. To use SimpleRole, add your application logic to SimpleRole.sv instead of Role.sv.

SimpleRole standardizes upon two different types of flow control, represented by **valid+grant** and **valid+full**. In both cases, a sender is attempting to send a packet of data to a receiver who may or may not be ready to accept the data. With **valid+grant**, the sender always attempts to send the packet by asserting the packet's `valid` bit whenever the sender is ready, and the receiver asserts its `grant` signal when it consumes the packet. In contrast, with **valid+full**, the receiver asserts its `full` signal when it is guaranteed to be ready to receive the packet. The sender waits for `full` to be asserted before sending the data by asserting the packet's `valid` bit.

In this User Guide, when we refer to role (uncapitalized), we are referring to the user application logic within SimpleRole. Role (capitalized) refers to either the original Role interface or Role.sv, depending on context.

The subsequent subsections describe in detail the various interfaces and flow control within SimpleRole. All signals are active high, i.e. an asserted signal is set to 1, while an unasserted signal is set to 0.

3.1.1 PCI Express

PCI Express (PCIe) is the primary method for software to send/receive data to/from the FPGA. The interface has been heavily optimized to provide maximum throughput over DMA. Users should transfer reasonably large data transactions (4KB+, 16KB+ preferred) over several slots (3+, 8+ preferred) to maximize bandwidth.

Although software sees a slot-based interface with 64KB slots, the role communicates with the PCIe interface through single 128-bit input/output queues tagged with metadata. Each queue entry contains a PCIe packet, consisting of 128 bits of data, 16 bits depicting the slot number (currently only bottom 6 bits are used), 4 bits of padding (currently unused), and 1-bit to denote if this is the last entry within this slot transaction. Each slot transaction has a minimum size of 256 bits, i.e. 2 packets. It's up to the user role to aggregate and store this data depending on the data format.

Previously, Roles with disconnected PCIe links could cause host machine blue screens due to Quartus optimizing away the PCIe IP block. To prevent this, we added a PCIe loopback FIFO in the shell, enabled by default to prevent these issues. To bypass the PCIe loopback FIFO in the shell and enable the Role to interact with PCIe, you must explicitly enable this by setting bit 6 in control_register (Shell Register 0). All Shell register configurations are documented in the "Microsoft Project Catapult Mt Granite Shell Architectural Specification" document, in Section 5 "PCIe Memory Mapped Registers". We provide helper functions to enable and disable PCIe in our example software.

Signal name	Width	Description
`valid`	1	Asserted if PCIe packet is valid
`data`	128	PCIe packet data. All PCIe data is broken into 128b packets
`slot`	16	PCIe slot number associated with this packet. Currently only bits [5:0] are used, rest should be set to 0 and ignored by user
`pad`	4	Currently unused, reserved. Should always be set to 0 and ignored by the user
`last`	1	Asserted if this packet is the last packet within this slot transaction

Table 1: PCIePacket struct definition

Signal name	Dir	Width	Description
`pcie_packet_in`	IN	PCIePacket	PCIe incoming packet from the shell. pcie_packet_in.valid is set to 1 only when `pcie_full_out` is 0
`pcie_full_out`	OUT	1	Asserted by the role when it cannot accept a valid PCIe input packet from the shell
`pcie_packet_out`	OUT	PCIePacket	PCIe outgoing packet to the shell. pcie_packet_out.valid is set to 1 if the outgoing packet is valid
`pcie_grant_in`	IN	1	Asserted by the shell when it accepts the PCIe outgoing packet

Table 2: SimpleRole PCIe interface

3.1.2 SoftReg

The SoftReg interface presents a low throughput interface over PCIe PIO to read and write small (64-bits) amount of data. This interface should not be used for bulk data transfer and instead should be used for tasks such as debugging, reading/writing performance counters, configuring the role, setting parameters, etc.

SoftReg read/write operations are initiated by software via Catapult API calls. The role receives SoftRegReq requests consisting of a valid bit, isWrite boolean, 32-bit address, and 64-bit data (valid only if write operation). There is no back-pressure mechanism; the role must handle the operation immediately or buffer the request for later handling. The role responds with data for a read request by asserting the SoftRegResp valid bit along with 64 bits of data. SoftReg read operations can be delayed, cannot be dropped. Failure to eventually respond with data will result in deadlock.

Signal name	Width	Description
`valid`	1	Asserted if SoftReg read/write request is valid
`isWrite`	1	Asserted if request is a write
`addr`	32	Address of read/write
`data`	64	If isWrite is asserted, data to be written

Table 3: SoftRegReq struct definition

Signal name	Width	Description
`valid`	1	Asserted if SoftReg response is valid (only after a read request)
`data`	64	SoftReg read data

Table 4: SoftRegResp struct definition

Signal name	Dir	Width	Description
`softreg_req`	IN	SoftRegReq	Incoming SoftReg read/write request from the shell
`softreg_resp`	OUT	SoftRegResp	Outgoing SoftReg read data response from the role

Table 5: SimpleRole SoftReg interface

3.1.3 DRAM

The Catapult Mt. Granite board supports 2 channels of DRAM, each containing 4GB DDR3-533. Each channel of DRAM is completely independent of the other channel, in terms of memory addressing, bandwidth, flow control, etc. This storage is intended to be used as scratch space for your role if the Altera M20K memory blocks are not sufficient.

Software cannot access FPGA DRAM address space directly. However, you can build in custom functionality into your role to emulate this through PCIe slots or SoftReg. To access DRAM in your role, send a valid MemReq to one of the two DRAM channels. A MemReq consists of the valid bit, isWrite boolean which is asserted for write operations and de-asserted for read operations, a 32-bit address, and 512-bit optional data (only valid if isWrite is asserted). The full 32 bits of addressing is required for addressing all 4GB of each DRAM channel, but addresses must be 64B (512-bit) aligned.

DRAM read requests are split phase transactions. For a read operation, you will need a separate FSM to dequeue the memory read response and process the data. You are not required to pre-allocate buffering for DRAM read response data to prevent deadlock, as this is already provided for you in the Role to SimpleRole interface conversion logic.

Signal name	Width	Description
`valid`	1	Asserted if request is valid
`isWrite`	1	Asserted if request is a write
`addr`	32	Address of DRAM read/write request
`data`	512	If isWrite is asserted, data to be written

Table 6: MemReq struct definition

Signal name	Width	Description
`valid`	1	Asserted if response is valid (only after a DRAM read request)
`data`	512	DRAM read data

Table 7: MemResp struct definition

Signal name	Dir	Width	Description
`mem_req_out[1:0]`	OUT	MemReq	Outgoing DRAM read/write request from role to shell. `mem_req_out.valid` is asserted if the outgoing request is valid
`mem_req_grant_in[1:0]`	IN	1	Asserted by the shell when the DRAM request is accepted
`mem_resp_in[1:0]`	IN	MemResp	Incoming DRAM read data from shell to role
`mem_resp_grant_out[1:0]`	OUT	1	Asserted by the role when the DRAM read response data is accepted

Table 8: SimpleRole DRAM interface

3.1.4 SerialLite III

SerialLite III (SL3) provides bi-directional, point-to-point connections between Catapult Mt Granite boards on multiple machines. Each Mt Granite board has four SL3 links, referred to as NORTH, SOUTH, EAST, and WEST, providing up to 1.5GB/s theoretical peak bandwidth each lane in each direction (TX, RX). The Catapult machines in TACC have been wired to form 6x8 torus networks, but Catapult academic users will not be allocated a full set of 48 machines unless necessary for their research.

The Shell provides a simple FIFO-based interface for each lane, where each lane is synchronized to a unique clock in each direction for a total of 8 clocks. To communicate to Role logic running on the Role clock, Role users must deal with crossing clock domains. SimpleRole simplifies this by instantiating clock domain crossing FIFOs, presenting simplified logic synchronized to a single SimpleRole clock to the user.

Each SL3 lane is split into two virtual channels in each direction, the main data interface and an out-of-band (OOB) interface. The two virtual channels are statically allocated a portion of link bandwidth through time-multiplexing, with the OOB interface only allocated a small percentage of overall bandwidth. The intention is for users to use the main data interface for transmitting data, with the OOB interface for flow control. Table 9 and 10 show the definition of the SL3 data and SL3 OOB interface structs, defined in SL3_Types.sv located in Shells\Common\SL3.

Signal name	Width	Description
`valid`	1	Asserted if packet is valid
`data`	128	Packet data
`last`	1	Asserted if last packet in message

Table 9: SL3DataInterface struct definition

Signal name	Width	Description
`valid`	1	Asserted if packet is valid
`data`	15	Packet data

Table 10: SL3OOBInterface struct definition

Table 11 shows the SL3 SimpleRole interface for each of the four lanes. Each lane is bi-directional, so it is split into transmit (TX) and receive (RX) paths. While the interface appears fully buffered with proper flow control mechanisms, flow control is not implemented. Although there is buffering in the SL3 pipeline, packets may drop if the receiver backpressures the incoming SL3 packet stream. Users should implement a flow control mechanism using the OOB interface. For example, a simple credit-based scheme can be implemented in which the sender is initialized with some number of credits, and credits are consumed when SL3 packets are sent and returned through the OOB interface when the receiver processes packets. We demonstrate this scheme in the provided SL3 example role.

Signal name	Width	Description
`sl_tx_out[3:0]`	SL3DataInterface	Outgoing SL3 packet from role to shell. Should only assert `sl_tx_out.valid` if `sl_tx_full_in` is de-asserted.
`sl_tx_full_in[3:0]`	1	Asserted by the shell if no available space for buffering outgoing packets.
`sl_tx_oob_out[3:0]`	SL3OOBInterface	Outgoing SL3 OOB packet from role to shell. Should only assert `sl_tx_oob_out.valid` if `sl_tx_oob_full_in` is de-asserted.
`sl_tx_oob_full_in[3:0]`	1	Asserted by the shell if no available space for buffering outgoing OOB packets.
`sl_rx_in[3:0]`	SL3DataInterface	Incoming SL3 packet from shell to role.
`sl_rx_grant_out[3:0]`	1	Asserted by the role when accepting an incoming valid SL3 packet.
`sl_rx_oob_in[3:0]`	SL3OOBInterface	Incoming SL3 OOB packet from shell to role.
`sl_rx_oob_grant_out[3:0]`	1	Asserted by the role when accepting an incoming valid SL3 OOB packet.

Table 11: SimpleRole SerialLite III transmit (TX) and receive (RX) interfaces

3.2 Creating Your First Catapult Role

Rather than creating your project from scratch, we recommend copying one of the example projects and modifying it to suit your needs. The AcademicPCIeLoopback role demonstrates a bare minimum “Hello World” level of functionality and is intended to be a good starting point for beginner Catapult users, or as a barebones project for starting a custom project. If you intend on using DRAM, the AcademicDRAMLoopback project demonstrates how to interface with DRAM and provides some useful library modules.

The files listed below represent the minimum number of files required for a working Catapult project utilizing SimpleRole:

- **Project**: Contains Quartus project configuration files
  - **Project.qpf**: Main Quartus project file. Open this file to load the project into Quartus.
  - **Project.qsf**: Main Quartus project settings file. Contains settings and points to the TCL file(s).
  - **Project.tcl**: Quartus settings file. Supports a limited set of TCL script. In Catapult, we keep the project settings entirely in the TCL file rather than in QSF due to issues with Quartus improperly modifying the QSF file.
- **RTL**: Contains project Verilog and SystemVerilog files
  - **App.tcl**: Lists all Verilog/SystemVerilog source files to be loaded
  - **Role.sv**: Role -> SimpleRole wrapper. Do not modify this file.
  - **SimpleDram.sv**: DRAM wrapper for SimpleRole, instantiated within Role.sv. Do not modify this file.
  - **SimpleRole.sv**: The actual role containing user logic.
- **Sim**: Contains co-simulation framework files
  - **runSim.ps1**: Top-level simulation script. Run this script via command line to simulate the design. Modify this file to change the software program, and to change the command-line parameters to be passed to the software program.
  - **build.do**: ModelSim build file. Contains list of libraries and Verilog/SystemVerilog files to be compiled. Modify this file to add new library or Verilog/SystemVerilog files for your Role. If you add new files into the RTL directory, you must modify this file!
  - **SimParams.sv**: Simulation parameters. The parameters in this file replace the Project.tcl parameters for co-simulation. Modify this file to change the parameters, or add new #define parameters.

Even if you plan on modifying the project to suit your needs, several files should not be modified or rarely modified to minimize strange errors. Both Project.qpf and Project.qsf are set to read-only by default to avoid an issue where Quartus sometimes modifies the files incorrectly and breaks the project configuration, which causes strange errors on compilation. Keep all your project changes in the TCL file.

However, if you wish to enable/disable SignalTap or change the SignalTap file, you’ll need to disable read-only mode on Project.qsf. To do this, right-click on the Project.qsf file and select “Properties”. Then, uncheck the “Read-only” box under “Attributes”.

3.3 Modifying Project.tcl

Project.tcl contains the main project settings and should be modified for most custom projects. Below is an explanation of the Project.tcl file for AcademicPCIeLoopback and describes how to modify the file to fit your needs.

```
# POINTER TO SHELL RELEASE
set shell_pointer ../../../Shells/Academic
set target_board "MtGranite"
source $shell_pointer/shell_pre.tcl
```

The first few lines configures the location of the Shell, sets a variable, and sources the Shell portion of the project. These lines should not be modified.

```
# Override default shell parameters
setFpgaUserClock 150
```

This line sets the FPGA role clock frequency. Below is a list of valid clock frequencies, located in Shells\Common\TclScripts\shell_parameters_common.tcl: 25MHz, 50MHz, 75MHz, 125MHz, 133MHz, 150MHz, 160MHz, 166MHz, 175MHz, 180MHz, 187MHz, 200MHz, 220MHz, 233MHz, 250MHz, 275MHz, 300MHz. A common trick is to speed up synthesis time by temporarily lowering the clock frequency. We recommend attempting to target 150MHz for your final designs.

```
setShellParam use_ddr 0
setShellParam use_sl3 0
```

To enable DDR, you must set use_ddr to 1. Similarly, to use SerialLite 3, you must set use_sl3 to 1. However, in Catapult SDK v1.1, SerialLite 3 is not yet available for use and thus is disabled. You should keep use_sl3 to 0. PCIe is always enabled.

```
setShellParam seed_value 6
```

Sometimes, the difference between passing and failing timing is the random seed used for synthesis. If you are slightly off by timing, or having strange timing issues, its sometimes useful to test different seed values. Do not use a seed of 0, this is a known bad value and will almost guarantee a timing failure.

```
source $shell_pointer/shell_post.tcl

# Include application-specific RTL
set app_path     "../RTL"
source $app_path/app.tcl

# Include common
set common_path ../../Common
source $common_path/common.tcl

set_global_assignment -name SYSTEMVERILOG_FILE $common_path/NetworkTypes/NetworkTypes.sv
```

The rest of the lines configure other required portions of the project and should not be modified.

3.4 AcademicPCIeLoopback

AcademicPCIeLoopback demonstrates a bare minimum “Hello World” level of functionality and is intended to be a good starting point for beginner Catapult users, or as a barebones project for starting a custom project. The role simply takes incoming data from PCIe, buffers the data in a FIFO, and sends the data back out to PCIe without any modifications. It can be interfaced with the LoopbackStressTest software program to measure peak PCIe bandwidth.

User logic is contained entirely within SimpleRole.sv.

3.5 AcademicDRAMLoopback

AcademicDRAMLoopback is a more complicated variation of AcademicPCIeLoopback, in which data is sent from software to FPGA through PCIe, written to DRAM, read back from DRAM, and sent from FPGA back to software through PCIe. The project demonstrates how to interface with DRAM, SoftReg, and provides some useful library modules. It is also compatible with the LoopbackStressTest software program, but provides lower performance since it is bottlenecked by DRAM read/write throughput. There are several optimizations that can be made to user logic to significantly improve DRAM throughput and thus overall performance. We leave this as an open exercise for beginners as a simple first project.

User logic is contained within SimpleRole.sv and DramInterleaver.sv. SimpleRole is the top-level user logic module containing the main FSM, DRAM interface logic, and SoftReg configuration logic. To utilize both channels of memory, we included a simple DramInterleaver module, which has the capability of sending all memory requests to channel 0 (default), sending all memory requests to channel 1, or interleaving requests across both memory channels. This configurability is provided via the SoftReg interface: software can write to the proper soft register number to change the DramInterleaver configuration. Users should only change the DramInterleaver configuration at the start of the program to avoid memory address space issues.

Note that the provided DramInterleaver module only has a single input port. To fully saturate both channels of memory, you need to design a dual-ported DramInterleaver module, which is currently not provided. We hope to increase the number of provided library components in the future to further improve ease of development.

3.6 AcademicSL3Test

AcademicSL3Test provides a simple demonstration of how to use the SL3 links by sending data from one machine to another. Data is sent from software to FPGA through PCIe, sent to the receiving machine through SL3, and sent from FPGA to software through PCIe. A single unified Role is used for both the sending and receiving FPGAs, and is configured through the SoftReg interface.

The SL3 interface is not fully buffered: users must create their own flow control mechanism to prevent packets from being dropped. This example project demonstrates a simple credit-based flow control mechanism using the SL3 OOB channel to return credits. The receiver node has a dedicated 4K-entry buffer per link for storing incoming SL3 packets. The sender node is initialized with 4K credits per link (configurable via SoftReg). Whenever the sender sends a packet, the number of available credits is decremented. If no credits are available, the sender stalls until credits are returned by the receiver node. Incoming packets to the receiver node are thus guaranteed space in the dedicated receiving buffer. When the receiver node dequeues from the receiving buffer, credits are returned through the OOB channel. Since OOB is low-bandwidth, we prevent credit returns from becoming the limiting factor for SL3 bandwidth by returning many credits in a single credit return OOB packet. An SL3 message is comprised of one or more packets, with the final packet in a message denoted by the "last" bit being set (see Table 9). The example uses a simple FSM to count the number of packets in a message and returns credits only when the final packet in a message is consumed, where the number of returned credits is equal to the size of the message. A message size larger than the size of the receiving buffer (4K 128-bit packets) will result in deadlock, and the ideal maximum message size is half the size of the buffer to enable double buffering.

4 The Catapult Co-Simulation (Cosim) Framework

The new Catapult co-simulation (cosim) framework introduced in SDK v1.2 allows users to functionally simulate their user roles alongside their software program, potentially reducing debug turnaround times and simplifying the overall debug process. The cosim framework models PCIe and DRAM with pure functional models: any performance results obtained through the co-simulation framework are likely to be inaccurate. With enough demand, we may create higher-accuracy models in the future.

4.1 Limitations and Intended Usage

With limited development resources, we focused on the bare necessities required for a minimal cosim framework that's still useful for end-users. With enough user demand, we may improve the cosim framework feature set and accuracy in the future SDK releases.

The cosim framework only simulates the SimpleRole as the top-level module under test. No part of the Role and Shell is simulated, which includes the Shell registers. The overall top-level module, SimTop.sv, connects SimpleRole to the SoftReg, DRAM, and PCIe functional models. SoftReg, DRAM and PCIe requests are handled instantly without modeling latency and bandwidth, resulting in no backpressure. Since Intel-ModelSim is only 32-bit, the DRAM model cannot simulate all 2x4GB of the memory address range. Instead, the DRAM model currently only simulates the first 1GB of each memory channel.

Therefore, the cosim framework will not return accurate performance results, and will likely not exercise many corner-cases in your design. **Cosim will not replace writing custom testbenches and debugging on actual hardware with SignalTap.** Instead, the intention of the cosim framework is to use its fast turnaround time and high visibility to help the user in the early stages of Role development, fixing simple bugs quickly and reducing the number of times spent waiting hours for Quartus to generate a bitfile and tediously going through waveforms with SignalTap.

4.2 Installation Prerequisites

To use the cosim framework, your Role must contain a Sim folder, which contains the main runSim.ps1 PowerShell script, build.do containing the list of libraries and Verilog/SystemVerilog files needed by your Role, and SimParams.sv containing the #defines for your Role.

The cosim framework currently depends on Microsoft Visual Studio 2015 (v14.0) to compile the functional models, test harness, and Catapult software. Simulation is done using SystemVerilog's Direct Programming Interface (DPI). Since scripts are written in PowerShell, the cosim framework can currently only be run on Windows machines. Future versions may use Python scripts for cross-platform support.

4.2.1 Installing ModelSim

The cosim framework requires Altera-Modelsim (now called Intel-Modelsim) v15.1 to be installed on your machine. You can download and install this for free from their website. Be sure to install the correct version. To use a newer version, you must modify the buildSim.ps1 script to change the hard-coded file path. https://www.altera.com/products/design-software/model---simulation/modelsim-altera-software.html

Intel-Modelsim is not free and requires a license; we provide the Intel-Modelsim v15.1 license for use on TACC machines. If you have already set the LM_LICENSE_FILE system variable following the directions in the "Getting Started with Catapult on TACC" guide, it should already point to a license server containing software licenses for both Quartus v15.1 and Intel-Modelsim v15.1. The license server currently does not contain valid licenses for newer versions of Quartus and Intel-Modelsim.

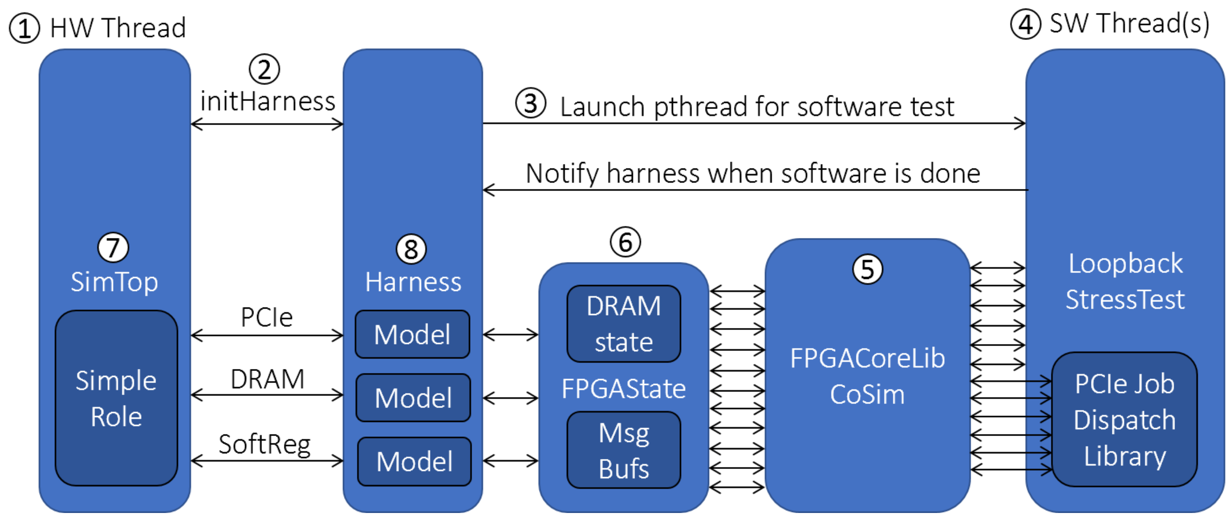


Figure 1: The Catapult Co-Simulation Framework

4.3 How it Works

Figure 1 shows the high-level architecture of the co-simulation framework:

(1) ModelSim simulates SimTop, a specialized simulator-only top-level file, which contains initialization code, the SimpleRole instantiation, and C-DPI hooks to connect the SimpleRole SystemVerilog file to the C interface functional models located inside the test harness.

(2) During initialization, SimTop calls `initHarness()` within the harness, which initializes the concurrent FPGAState data structure containing all simulated FPGA DRAM state, PCIe slot state, and message passing buffers.

(3) Next, SimTop launches the software program in a separate thread, overriding argc and argv with the passed simulation command line parameters. The simulation runs until the software program exits, at which point the harness will set a bit notifying SimTop that simulation is complete so that SimTop can call `$finish`.

(4) The software program, for example the provided LoopbackStressTest, is compiled with `#define COSIM 1`, and the program runs and calls FPGACoreLib API calls as usual without further modifications. Libraries such as the provided PCIe job dispatch library can run without modifications.

(5) These API calls are re-defined in FPGACoreLib_CoSim, which interacts with the simulated Catapult FPGA state instead of the Catapult hardware board through the drivers.

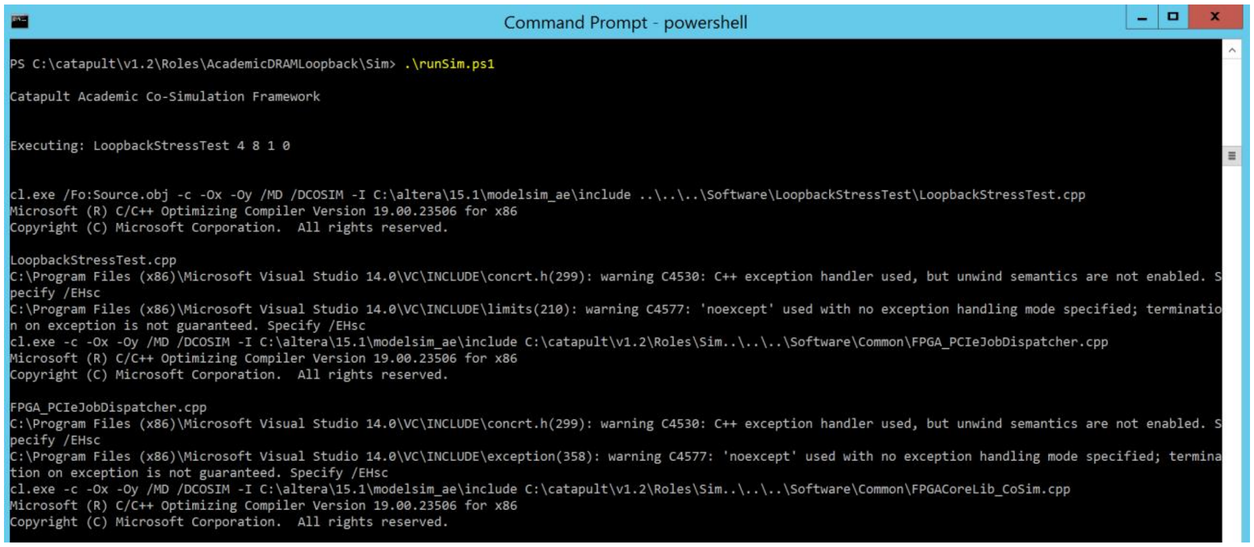
(6) The FPGAState data structure is accessed concurrently by the software program through FPGACoreLib_CoSim, and by SimpleRole through the test harness. It contains all Catapult interface architectural state, including arrays containing both channels of DRAM data (limited to 1GB per channel with 32-bit ModelSim), PCIe input and output slots with full and done bits, and SoftReg pending requests and responses.

(7) To access FPGAState, SimTop services SimpleRole requests and responses, providing the necessary hooks to C-DPI calls in the test harness. Each cycle, SimTop checks to see if incoming data is available through PCIe, enqueueing to SimpleRole if space permits. Similarly, SimTop checks every cycle to see if outgoing PCIe data is available. If so, the data is dequeued and passed to the test harness. DRAM reads and writes are treated as simple array reads and writes, completing instantly. SoftReg requests to SimpleRole can take many cycles to complete, so SimTop uses a split phase interface.

(8) The test harness provides a simple functional model of DRAM, PCIe, and SoftReg without modeling latency or bandwidth, providing SimTop a way to access FPGAState located in C code. DRAM and SoftReg have simple implementations, but require locking to deal with concurrent accesses from software. For PCIe, it converts the slot-based input/output buffer interface into the FIFO-based interface seen by SimpleRole.

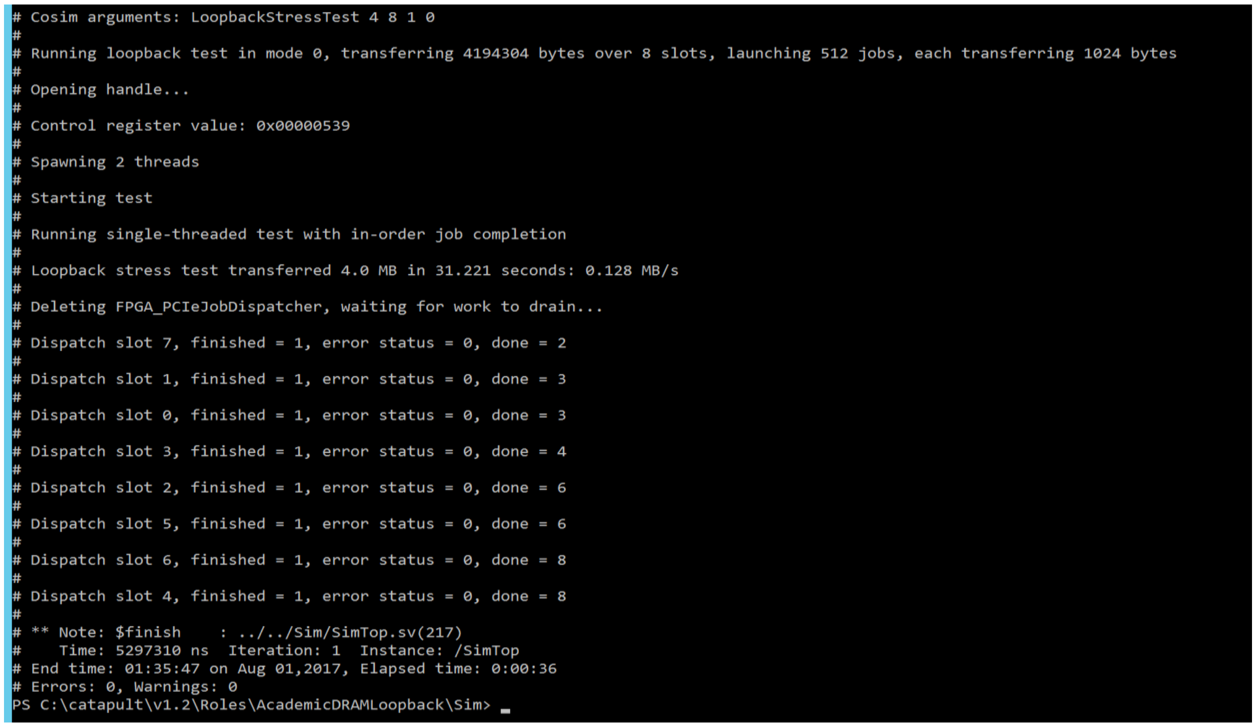
4.4 Running a Simulation

Running a simulation using the co-simulation framework is very simple: navigate to the Sim folder in either AcademicPCIeLoopback or AcademicDRAMLoopback, then execute the runSim.ps1 PowerShell script. The script will first compile your software program, in this case LoopbackStressTest, and all other C/C++ source files using the Microsoft Visual Studio compiler (MSVC) and generate a DLL from the object files.



```
PS C:\catapult\v1.2\Roles\AcademicDRAMLoopback\Sim> .\runSim.ps1

Catapult Academic Co-Simulation Framework


Executing: LoopbackStressTest 4 8 1 0


cl.exe /Fo:Source.obj -c -Ox -Oy /MD /DCOSIM -I C:\altera\15.1\modelsim_ae\include ..\..\..\Software\LoopbackStressTest\LoopbackStressTest.cpp
Microsoft (R) C/C++ Optimizing Compiler Version 19.00.23506 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

LoopbackStressTest.cpp
C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\INCLUDE\concrt.h(299): warning C4530: C++ exception handler used, but unwind semantics are not enabled. S
pecify /EHsc
C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\INCLUDE\limits(210): warning C4577: 'noexcept' used with no exception handling mode specified; terminatio
n on exception is not guaranteed. Specify /EHsc
cl.exe -c -Ox -Oy /MD /DCOSIM -I C:\altera\15.1\modelsim_ae\include C:\catapult\v1.2\Roles\Sim..\..\..\Software\Common\FPGA_PCIeJobDispatcher.cpp
Microsoft (R) C/C++ Optimizing Compiler Version 19.00.23506 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

FPGA_PCIeJobDispatcher.cpp
C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\INCLUDE\concrt.h(299): warning C4530: C++ exception handler used, but unwind semantics are not enabled. S
pecify /EHsc
C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\INCLUDE\exception(358): warning C4577: 'noexcept' used with no exception handling mode specified; termina
tion on exception is not guaranteed. Specify /EHsc
cl.exe -c -Ox -Oy /MD /DCOSIM -I C:\altera\15.1\modelsim_ae\include C:\catapult\v1.2\Roles\Sim..\..\..\Software\Common\FPGACoreLib_CoSim.cpp
Microsoft (R) C/C++ Optimizing Compiler Version 19.00.23506 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.
```

Next, ModelSim builds the SystemVerilog modules, links with the generated DLL from MSVC, and simulates your design.



```
# Cosim arguments: LoopbackStressTest 4 8 1 0
#
# Running loopback test in mode 0, transferring 4194304 bytes over 8 slots, launching 512 jobs, each transferring 1024 bytes
#
# Opening handle...
#
# Control register value: 0x00000539
#
# Spawning 2 threads
#
# Starting test
#
# Running single-threaded test with in-order job completion
#
# Loopback stress test transferred 4.0 MB in 31.221 seconds: 0.128 MB/s
#
# Deleting FPGA_PCIeJobDispatcher, waiting for work to drain...
#
# Dispatch slot 7, finished = 1, error status = 0, done = 2
#
# Dispatch slot 1, finished = 1, error status = 0, done = 3
#
# Dispatch slot 0, finished = 1, error status = 0, done = 3
#
# Dispatch slot 3, finished = 1, error status = 0, done = 4
#
# Dispatch slot 2, finished = 1, error status = 0, done = 6
#
# Dispatch slot 5, finished = 1, error status = 0, done = 6
#
# Dispatch slot 6, finished = 1, error status = 0, done = 8
#
# Dispatch slot 4, finished = 1, error status = 0, done = 8
#
# ** Note: $finish    : ../../Sim/SimTop.sv(217)
#    Time: 5297310 ns  Iteration: 1  Instance: /SimTop
# End time: 01:35:47 on Aug 01,2017, Elapsed time: 0:00:36
# Errors: 0, Warnings: 0
PS C:\catapult\v1.2\Roles\AcademicDRAMLoopback\Sim>
```

Congratulations, you have successfully run a simulation!

5 Catapult Software Overview

Every Catapult project also requires a corresponding software component. The simplest Catapult software program initializes and configures the FPGA, sends input data, waits for results, and reads the generated output data. More advanced software may simultaneously perform part of the computation on FPGA and part of the computation on CPU, perhaps even through a dynamic load balancing mechanism. The Catapult API provides helper functions to enable both types of programming models.

5.1 Catapult API example

The C-based, user-level FPGA Communication Library API provides FPGA-to-CPU communication. The basic unit of communication between the FPGA and the CPU is a **message**. The message size is bounded by the size of the FPGA buffers (currently 64KB).

Figure 1 illustrates a simple loopback application written in C, where the CPU constructs a message to be sent to the FPGA, which is then routed back from the FPGA to the CPU.

```
#define PCIE_HIP_NUM                          0x0

void LoopbackExample()
{
    FPGA_HANDLE fpgaHandle;
    DWORD *pInputBuffer, *pOutputBuffer;
    DWORD whichBuffer = 17;  // which buffer (aka slot) to send the message on
    DWORD sendBytes = 112, recvBytes;

    // Open handle to FPGA
    FPGA_CreateHandle(&fpgaHandle, PCIE_HIP_NUM, 0x0, NULL, NULL);

    // Grab pinned input and output buffers
    FPGA_GetInputBufferPointer(fpgaHandle, whichBuffer, &pInputBuffer);
    FPGA_GetOutputBufferPointer(fpgaHandle, whichBuffer, &pOutputBuffer);

    // Write 112B (7 words) of random data into input buffer
    for (DWORD i = 0; i < sendBytes/sizeof(DWORD); i++)
        pInputBuffer[i] = rand();

    // Send the data to the FPGA
    FPGA_SendInputBuffer(fpgaHandle, whichBuffer, sendBytes, useInterrupt);
    // Wait for the response to come back
    FPGA_WaitOutputBuffer(fpgaHandle, whichBuffer, &recvBytes, useInterrupt);
    // Consume the contents of pOutputBuffer
    // ...
    // Discard the buffer
    FPGA_DiscardOutputBuffer(fpgaHandle, whichBuffer);
    // Close handle.
    FPGA_CloseHandle(fpgaHandle);
}
```

Figure 1: Code snippet illustrating CPU-to-FPGA loopback communication. For brevity, error handling is omitted.

In the example above, the user first opens a handle to the FPGA device using `FPGA_CreateHandle`. The user then obtains pointers to kernel-pinned (user-mapped) regions of input and output slot buffers using `FPGA_GetInputBufferPointer` and `FPGA_GetOutputBufferPointer` calls. It then fills the input buffer (`pInputBuffer`) with the data to send. When the `FPGA_SendInputBuffer` function is called, the message stored in `pInputBuffer` is DMAed to the FPGA buffer over PCIe. The call to `FPGA_WaitOutputBuffer` blocks the user thread until a response arrives from the FPGA. The user is then able to consume the contents of `pOutputBuffer`. Afterwards, the output buffer must be released using `FPGA_DiscardOutputBuffer`.

5.2 The PCIeJobDispatcher Library

While the above example may seem straightforward, to fully utilize available PCIe bandwidth, your application needs to transfer and receive multiple slots in parallel, ideally in any order. A simple way of implementing this is with multiple threads, where each thread is responsible for reading/writing one slot of data.

We have simplified this task by implementing a simple PCIe job dispatcher library class called FPGA_PCIeJobDispatcher. A job is an independent task to be completed on FPGA, consisting of sending 1 input slot of data and receiving 1 output slot of data. Each job cannot send or receive more than 64KB of data, i.e. fits in one slot. On construction, the job dispatcher spawns N job dispatch threads, one thread in charge of each input/output slot pair. Each thread waits for data to send. Once data arrives via the sendJob() function, the thread sends the data to FPGA through its assigned slot, and then waits for output data from FPGA to arrive. On arrival, the thread buffers the data in a separate queue and frees the slot, making the data available via recvJob().

API Name	Description
`FPGA_PCIeJobDispatcher( FPGA_HANDLE fpgaHandle, uint32_t numSlots)`	Constructs PCIeJobDispatcher, passing in the FPGA handle and the maximum number of slots (max 64). Spawns one thread per slot.
`void sendJob(uint32_t slot, void* inputPtr, void* outputPtr, uint32_t size)`	Schedules a job. Sends a job of SIZE bytes to slot number SLOT. SIZE should be equal to or lower than 65536.
`bool recvJob(uint32_t slot, uint32_t& size)`	Checks if job output is available. Returns TRUE if a job is available on slot SLOT. SIZE is set to the size of the available output data. Slot data is written to the location of OUTPUTPTR, set via sendJob().
`void shutdown()`	Prepares PCIeJobDispatcher for destruction. Blocks until all job dispatch threads are idle, i.e. have no pending work and not waiting for output.
`~FPGA_PCIeJobDispatcher()`	Destroys the PCIeJobDispatcher. The destruction calls shutdown() and thus will block until all job dispatch threads are idle.

Table 9: The FPGA_PCIeJobDispatcher API, a helper library class to make high-throughput PCIe operations easier to write

5.3 Using the PCIe Job Dispatcher

The PCIe job dispatcher is flexible and supports many different programming models. Several example usage methods are demonstrated in the LoopbackStressTest example code. We demonstrate these examples below on a simple scenario in which we have many jobs per slot to be dispatched. We show how to utilize the job dispatcher to complete the jobs as fast as possible.

5.3.1 Single-threaded, in-order job completion

This method shown below is the simplest method to utilize the job dispatcher. Each iteration, we dispatch work items to every slot (Line 3). Then after iterating across every slot, we wait for the jobs to complete in the order in which they were dispatched (Line 8). If the job is not ready, we defer the thread (Line 9) to prevent deadlock. An optimization to this scheme would be performing useful work if the previous jobs are not yet ready.

```
 1  for (uint64_t i = 0; i < jobsPerSlot; i++) {
 2      for (int j = 0; j < numSlots; j++) {
 3          dispatcher->sendJob(j, inputBufs[j], outputBufs[j], jobSize);
 4      }
 5  
 6      for (int j = 0; j < numSlots; j++) {
 7          uint32_t recvJobSize;
 8          while (!dispatcher->recvJob(j, recvJobSize)) {
 9              concurrency::wait(0);
10          }
11      }
12  }
```

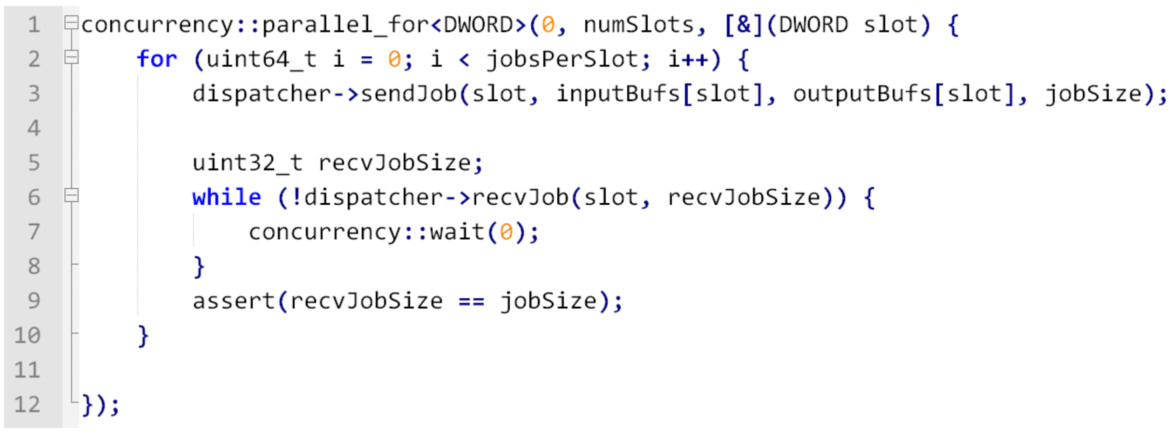
5.3.2 Single-threaded, out-of-order job completion

The method shown below is an optimization of the single-threaded, in-order, job completion technique. We observe that some jobs may take longer to complete than others, and thus we should look for jobs that are done rather than waiting for jobs to complete in the order that they were dispatched. There are no changes to job dispatch (Line 3), but there are significant changes to the job completion detection (Line 7). Rather than iterating over a `for` loop, we use a `while` loop to check jobs. If a job is not completed, we simply skip the job for now and check other jobs until all jobs are completed.

```
 1  for (uint64_t i = 0; i < jobsPerSlot; i++) {
 2      for (int j = 0; j < numSlots; j++) {
 3          dispatcher->sendJob(j, inputBufs[j], outputBufs[j], jobSize);
 4      }
 5  
 6      uint32_t dones = 0;
 7      while (dones != numSlots) {
 8          for (int j = 0; j < numSlots; j++) {
 9              uint32_t recvJobSize;
10              if (dispatcher->recvJob(j, recvJobSize)) {
11                  assert(recvJobSize == jobSize);
12                  dones++;
13              }
14          }
15      }
16  }
```

5.3.3 Multi-threaded

One issue with the single-threaded technique shown above is that we assume each slot computes jobs in lock-step; no slot can start a new job until all slots have completed a job. Also, job dispatch and waiting are sometimes on the critical path of the program. Although not shown in this example, it is common for data to require pre-processing, or for the FPGA output data to be processed or checked for correctness. We can use the `parallel_for` construct in Microsoft Visual Studio's Parallel Patterns Library to easily implement multithreading to solve both of these problems. First, we replace the `for` loop with a `parallel_for` loop with `numSlots` threads (Line 1). Each spawned thread runs a copy of the loop body in parallel, in which each thread sends a job and then waits for the job to complete before starting the next job. Rather than simply waiting for a job to complete, an additional optimization would be to perform more useful work (e.g. pre-processing data or checking results).



```
1   concurrency::parallel_for<DWORD>(0, numSlots, [&](DWORD slot) {
2       for (uint64_t i = 0; i < jobsPerSlot; i++) {
3           dispatcher->sendJob(slot, inputBufs[slot], outputBufs[slot], jobSize);
4
5           uint32_t recvJobSize;
6           while (!dispatcher->recvJob(slot, recvJobSize)) {
7               concurrency::wait(0);
8           }
9           assert(recvJobSize == jobSize);
10      }
11
12  });
```

5.3.4 Multi-threaded with Decoupled Job Dispatch/Completion

One issue with the previous approach is that there is some delay between when a job completes and the next job begins, since the techniques presented so far only have 1 job in flight per slot. To fully maximize PCIe throughput, we can queue up many jobs per slot, such that the PCIe job dispatcher can immediately dispatch the next job after it completes a job. To accomplish this, rather than spawning one thread per slot, we spawn two threads: a job dispatch thread and a job completion thread. Again, we use the parallel_for construct, but this time we nest two loops such that each thread has a unique slot and ID pair. The ID represents if the thread is assigned dispatch (Line 4) or completion (Line 10). The dispatch threads only task is to dispatch work until completion, while the completion threads only task is to wait for jobs to complete and process the output.

```cpp
concurrency::parallel_for<DWORD>(0, numSlots, [&](DWORD slot) {
    concurrency::parallel_for<DWORD>(0, 2, [&](DWORD x) {
        // Slot job dispatch thread
        if (x == 0) {
            for (uint64_t i = 0; i < jobsPerSlot; i++) {
                dispatcher->sendJob(slot, inputBufs[slot], outputBufs[slot], jobSize);
            }
        }
        // Slot job completion thread
        else {
            for (uint64_t i = 0; i < jobsPerSlot; i++) {
                uint32_t recvJobSize;
                while (!dispatcher->recvJob(slot, recvJobSize)) {
                    concurrency::wait(0);
                }
                assert(recvJobSize == jobSize);
            }
        }
    });
});
```

6 Catapult SDK Sample Software

The Catapult SDK contains example software as part of the example projects. Currently, we only include a single program that can interface with both AcademicPCIeLoopback and AcademicDRAMLoopback roles.

6.1 LoopbackStressTest

LoopbackStressTest is a simple program that measures average PCIe bandwidth, and is compatible with both AcademicPCIeLoopback and AcademicDRAMLoopback roles. The program uses the PCIeJobDispatcher and demonstrates the four methods shown above in Section 4.3. PCIe Role logic is disabled by default due to the possibility of incorrect Role logic causing blue screens. We provide helper functions for setting the appropriate bits in Shell Register 0 to enable/disable PCIe.

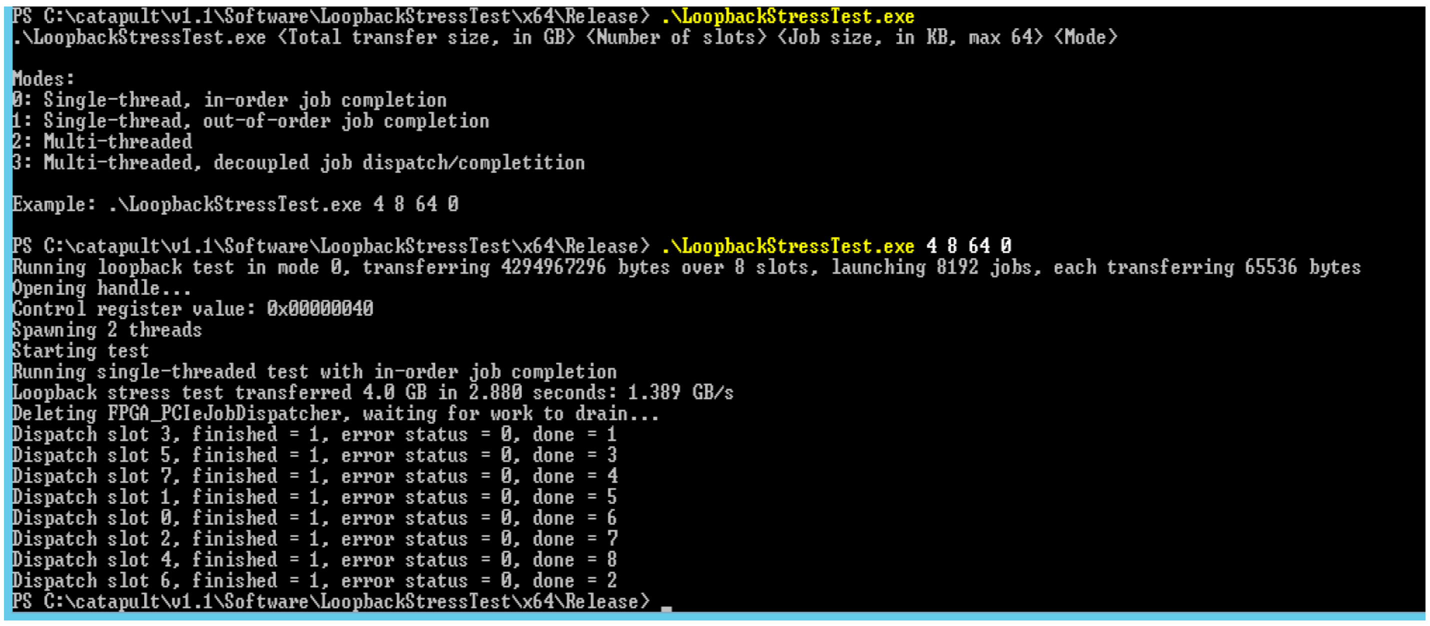
LoopbackStressTest uses the following command line options:

```
.\LoopbackStressTest.exe <Transfer size (GB)> <Number of slots> <Job size
(KB)> <Mode>
```

Where legal modes are:

- 0: Single-threaded, in-order job completion (Section 4.3.1)
- 1: Single-threaded, out-of-order job completion (Section 4.3.2)
- 2: Multithreaded (Section 4.3.3)
- 3: Multithreaded with decoupled job dispatch/completion (Section 4.3.4)

Below shows an example run that sends 4GB of data over 8 slots, with each job transferring 64KB in single-threaded, in-order job completion mode:



```
PS C:\catapult\v1.1\Software\LoopbackStressTest\x64\Release> .\LoopbackStressTest.exe
.\LoopbackStressTest.exe <Total transfer size, in GB> <Number of slots> <Job size, in KB, max 64> <Mode>

Modes:
0: Single-thread, in-order job completion
1: Single-thread, out-of-order job completion
2: Multi-threaded
3: Multi-threaded, decoupled job dispatch/completition

Example: .\LoopbackStressTest.exe 4 8 64 0

PS C:\catapult\v1.1\Software\LoopbackStressTest\x64\Release> .\LoopbackStressTest.exe 4 8 64 0
Running loopback test in mode 0, transferring 4294967296 bytes over 8 slots, launching 8192 jobs, each transferring 65536 bytes
Opening handle...
Control register value: 0x00000040
Spawning 2 threads
Starting test
Running single-threaded test with in-order job completion
Loopback stress test transferred 4.0 GB in 2.880 seconds: 1.389 GB/s
Deleting FPGA_PCIeJobDispatcher, waiting for work to drain...
Dispatch slot 3, finished = 1, error status = 0, done = 1
Dispatch slot 5, finished = 1, error status = 0, done = 3
Dispatch slot 7, finished = 1, error status = 0, done = 4
Dispatch slot 1, finished = 1, error status = 0, done = 5
Dispatch slot 0, finished = 1, error status = 0, done = 6
Dispatch slot 2, finished = 1, error status = 0, done = 7
Dispatch slot 4, finished = 1, error status = 0, done = 8
Dispatch slot 6, finished = 1, error status = 0, done = 2
PS C:\catapult\v1.1\Software\LoopbackStressTest\x64\Release> _
```

6.2 SL3StressTest

SL3StressTest is a simple program that measures SL3 bandwidth, and is only compatible with the provided AcademicSL3Test role. The program demonstrates how to initialize and configure SL3 through the shell registers, determine SL3 status, and how to write a simple program targeting multiple machines with Catapult. The program also uses the PCIeJobDispatcher, but we just use single-threaded mode with in-order commits because SL3 maximum lane bandwidth is substantially lower than PCIe bandwidth.

SL3StressTest uses the following command line options:

```
.\SL3StressTest.exe <Transfer size (MB)> <Number of slots> <Job size in
packets> <Direction> <Mode>
```

To get an accurate bandwidth measurement, you should transfer at least 1024MB (1GB), and ideally at least 4096MB (4GB).

Number of slots should be between 1 and 64, with 8 being near optimal.

Job size is in PCIe packets, where each packet contains 128-bits (16B) of data. Job size must be between 2 (32B) and 4096 (64KB), inclusive. Due to the implemented custom credit return system in SimpleRole for the SL3StressTestExample, the optimal job size is 2048 (32KB) such that double buffering can be used in the 64KB SL3 transmission and reception buffers.

Direction is the SL3 port direction, which can be 0 (North), 1 (South), 2 (East), or 3 (West). The machine directions are statically mapped based on the cable connection configurations in the server room. If you are connecting to a machine in TACC, consult the “TACC torus mapping.xlsx” guide to determine machine torus cable connection mappings.

Mode is either 0 (Send) or 1 (Receive). The sending machine transmits data, and the receiving machine stalls until the full amount of data is received.

6.3 Configuring SL3 Shell Registers

All Shell register configurations are documented in the “Microsoft Project Catapult Mt Granite Shell Architectural Specifications”, in Section 5 “PCIe Memory Mapped Registers”. In SL3StressTest, we provide helper functions for accessing and setting the registers relevant for SL3 operation.

6.3.1 Enabling SL3 RX links

After an FPGA reconfig, all receiving SL3 ports are disabled by default. To enable a specific link, set the appropriate bit in control_register2, mapped to Shell register number 5. Bits 16 to 19, inclusive, map to the RX enable bit for NORTH, SOUTH, EAST, and WEST, respectively.

The FPGA shell register interface can only read and write full 32-bit register values. To set a specific bit, you must perform a read-modify-write on the Shell register. We provide the helper functions `enableSL3RecvPort()` and `disableSL3RecvPort()` to perform this functionality for you.

6.3.2 Reading and Writing SL3 Node IDs

Users can assign an 8-bit nodeID value to each machine, initialized to all 1’s on FPGA reconfig, which can be read to determine the machine names in each SL3 direction. The bottom 8 bits of control_register2, mapped to Shell register 5, map to nodeID. We provide a helper function `setNodeID()` to perform the required read-modify-write for you.

Shell register 97 contains SL3 neighbor nodeIDs in each direction NORTH, SOUTH, EAST, and WEST, with the lowest 8 bits for the nodeID of the neighbor in the NORTH direction, and the upper 8 bits for the nodeID of the neighbor in the WEST direction. The results are valid only if SL3 is enabled on both the Catapult board and its neighbor. We provide a helper function `getNodeID()` to read the nodeID in the specified direction.

6.3.3 Determining if a SL3 neighbor is active

Shell register 96 returns overall SL3 link status for each direction NORTH, SOUTH, EAST, WEST, mapped to bits 0-3 inclusive with NORTH being the lowest bit 0. These bits can be read to determine if SL3 is properly configured and ready for sending data in a particular direction. We provide a helper function `neighborExists()` to perform this functionality for you.

7 Using the Catapult Driver Utilities

The Catapult SDK provides several utilities located in Driver\Bin. These utilities allow the user to program and reconfigure the FPGA, as well as examining the FPGA and role status. All utilities require administrative access to run.

7.1 RSU

RSU is used to program and reconfigure the FPGA. Catapult applications are loaded onto flash memory using the -write switch. However, this doesn't program the FPGA with the image. The -reconfig switch programs the application from flash memory onto the FPGA. If some issue arises with the application image, recover with the golden image using -reconfigGolden. To prevent potential issues, Catapult users are not allowed to overwrite the golden image on flash memory.

Switch Name	Description
-write <app.rpd>	Overwrites the application image in flash memory.
`-reconfig`	Reconfigure flash to the application image
`-reconfigGolden`	Reconfigure flash to the golden image
`-pgm <app.sof>`	Load image via Quartus JTAG programmer. Do not use this.

Table 10: RSU.exe functionality

7.2 FPGADiagnostics

FPGADiagnostics is used to dump/query FPGA state, read/write FPGA registers, read flash memory, reconfigure (identical functionality to RSU.exe), and inject sample PCIe DMA traffic from a file. This utility may be useful when debugging, and to determine if there is an issue with the FPGA. In particular, the "FPGADiagnostics.exe -dumpHealth" and "FPGADiagnostics -justdumpregs" commands are often useful when debugging problems.

Switch Name	Description
-dumpHealth	Displays a summary of FPGA/PCIe health. Ignore "bitstream not built cleanly by TFS" warnings.
`-dumpSlotDma <optionalFileName>`	Dumps slot contents into specified file
`-clockCycleCount`	Displays role clock frequency and FPGA uptime
`-writeFlashApp <image.rpd>`	Writes application image to flash (same as RSU)
`-readFlashGolden <image.rpd>`	Reads golden image from flash
`-readFlashApp <image.rpd>`	Reads application image from flash
`-reconfigGolden`	Programs golden image to FPGA (same as RSU)

`-reconfigApp`	Programs application image to FPGA (same as RSU)
`-vendorDeviceId`	Displays device ID
`-justdumpregs`	Dumps all FPGA shell registers
`-justreadreg <regnum>`	Reads a specific FPGA shell register
`-justwritereg <regnum> <value>`	Writes to a specific FPGA shell register
`-justreadsoftreg <regnum>`	Reads a specific FPGA application register
`-justwritesofterg <regnum> <value>`	Writes to a specific FPGA application register
`-currenttemp`	Displays the current FPGA temperature
`-sendinput <infile> <slotNum> <reps>`	Writes data from <infile> to slot for <reps> times

Table 11: FPGADiagnostics.exe functionality

8 Debugging your Catapult Project

The Catapult SDK currently does not have any built-in co-simulation framework for debugging. Therefore, users must write custom tests for their role and/or debug on the FPGA itself by instrumenting the FPGA with SignalTap. All example roles come with SignalTap enabled with pre-defined SignalTap .stp files and may be used as a reference.

8.1 Altera SignalTap II

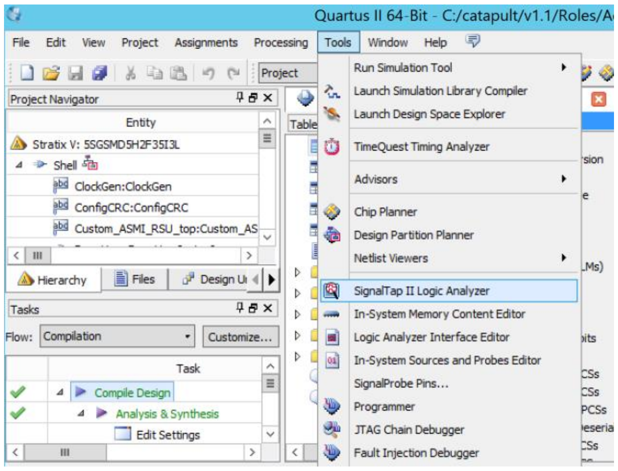
The Altera SignalTap II Embedded Logic Analyzer is a system-level debugging tool that captures and displays real-time signals in a design. By using SignalTap, designs can observe the behavior of hardware running at full speed using a waveform viewer in response to software execution. SignalTap has many strengths, but also many limitations. For moderate to complex designs, we highly recommend that users debug primarily through software testbenches and resort to SignalTap when necessary, e.g. when the design seems to work in simulation but not when running on Catapult.

SignalTap allows users to examine signal states when a user-defined signal event called a “trigger” occurs by automatically instrumenting the design with SignalTap instances and storing captured signals in M20K RAM. While SignalTap in theory supports capturing up to 2048-bits of aggregate signals with up to 128K capture depth, the effective number of signals you can capture is a function of the amount of left-over M20K RAM and routing resources left in your design. Users should carefully select only the most important signals to monitor and create a proper trigger point to minimize resource utilization. Keep in mind that using SignalTap will increase your synthesis time and potentially create new timing violations. Temporarily lowering your role clock frequency while debugging may alleviate these issues.

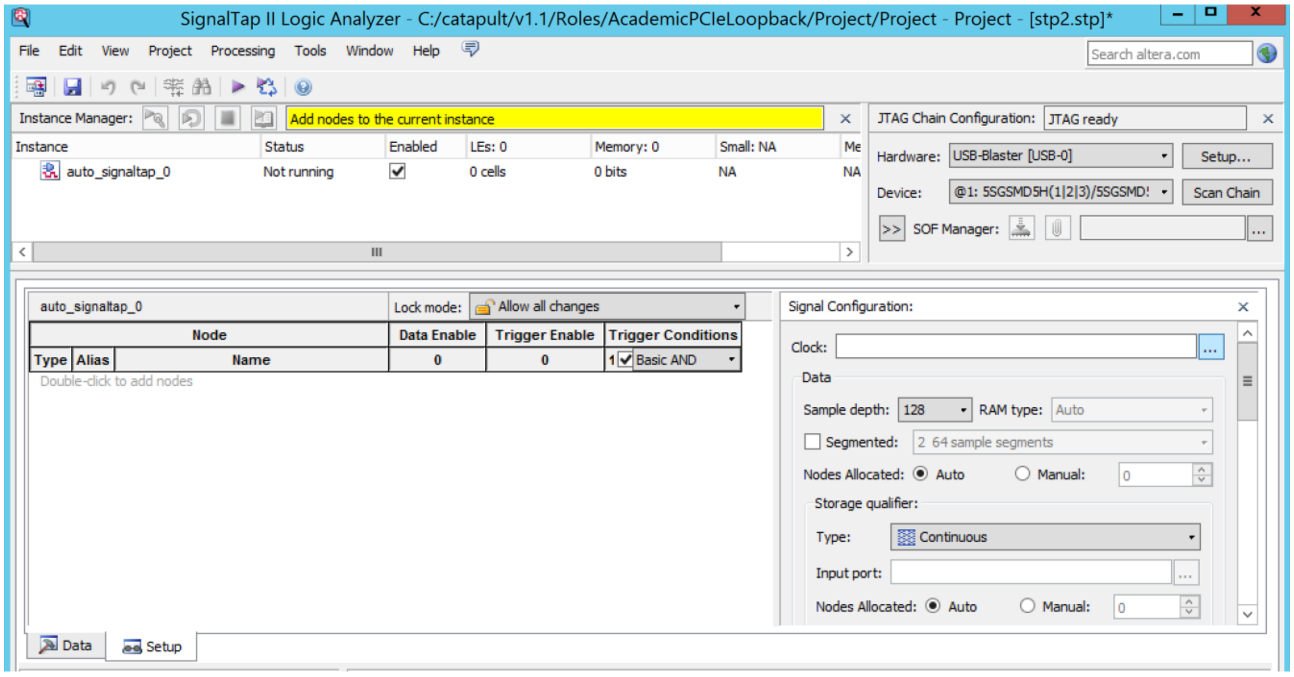
8.2 Adding SignalTap to Your Design

Adding SignalTap to your design involves creating a SignalTap .stp configuration file and enabling SignalTap for your project. SignalTap has many features and can be quite complex to configure. This user guide will only provide an introduction. For more details, refer to Altera resources on their website, for example Design Debugging Using the SignalTap II Embedded Logic Analyzer. YouTube can also be a good source, for example this video from Altera: SignalTap II Embedded Logic Analyzer Basics.

To create or modify the SignalTap configuration file, open Quartus and go to Tools -> SignalTap II Logic Analyzer.


Quartus II 64-Bit - C:/catapult/v1.1/Roles/A
File Edit View Project Assignments Processing Tools Window Help
Project Navigator
Entity
Stratix V: 5SGSMD5H2F35I3L
Shell
ClockGen:ClockGen
ConfigCRC:ConfigCRC
Custom_ASMI_RSU_top:Custom_AS
Hierarchy
Files
Design U
Tasks
Flow: Compilation
Customize...
Task
Compile Design
Analysis & Synthesis
Edit Settings
Run Simulation Tool
Launch Simulation Library Compiler
Launch Design Space Explorer
TimeQuest Timing Analyzer
Advisors
Chip Planner
Design Partition Planner
Netlist Viewers
SignalTap II Logic Analyzer
In-System Memory Content Editor
Logic Analyzer Interface Editor
In-System Sources and Probes Editor
SignalProbe Pins...
Programmer
JTAG Chain Debugger
Fault Injection Debugger

The Instance Manager lists the single SignalTap instance named “auto_signaltap_0” (currently empty). You may find it useful to group different signals and triggers into different instances, but for this example we will only use a single SignalTap instance. Next, to set the SignalTap instance clock, click on the “…” button next to “Clock” under “Signal Configuration”.


File Edit View Project Processing Tools Window Help
Search altera.com
Setup...
Scan Chain
SOF Manager:
Double-click to add nodes
Data
Setup

SignalTap II Logic Analyzer - C:/catapult/v1.1/Roles/AcademicPCIeLoopback/Project/Project - Project - [stp2.stp]*

Instance Manager: Add nodes to the current instance

Instance	Status	Enabled	LEs: 0	Memory: 0	Small: NA	Me
auto_signaltap_0	Not running	✓	0 cells	0 bits	NA	NA

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0]

Device: @1: 5SGSMD5H(1|2|3)/5SGSMD!

auto_signaltap_0 — Lock mode: Allow all changes

Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	0	0	1 ✓ Basic AND

Signal Configuration:

Clock:

Data

Sample depth: 128 RAM type: Auto

Segmented: 2 64 sample segments

Nodes Allocated: ● Auto ○ Manual: 0

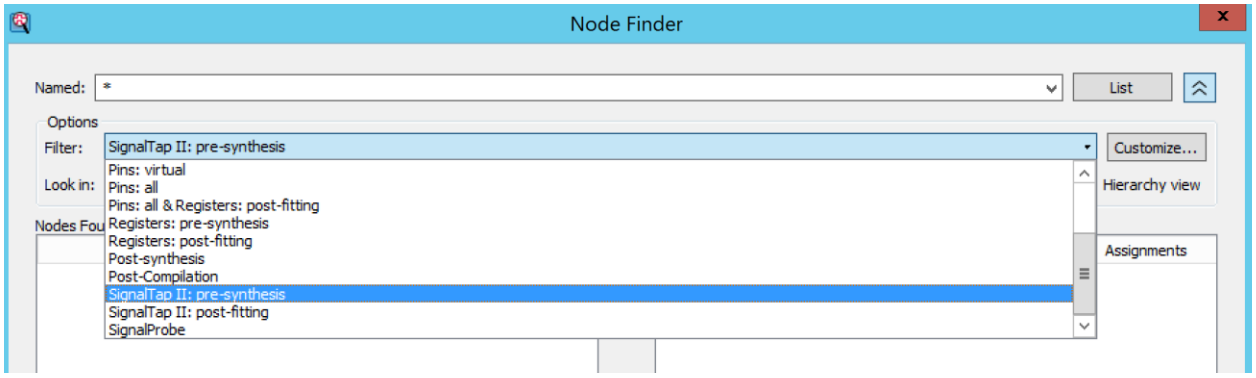
Storage qualifier:

Type: Continuous

Input port:

Nodes Allocated: ● Auto ○ Manual: 0

This opens a Node Finder window, which allows you to select the clock node. Click on the "Filter" drop-down menu and select "SignalTap II: Pre-synthesis". This will allow you to search signals found in your Verilog HDL files. Selecting "SignalTap II: Post-fitting" will search for nodes in the final post-fit layout generated by the Quartus fitter, which generally is less useful.

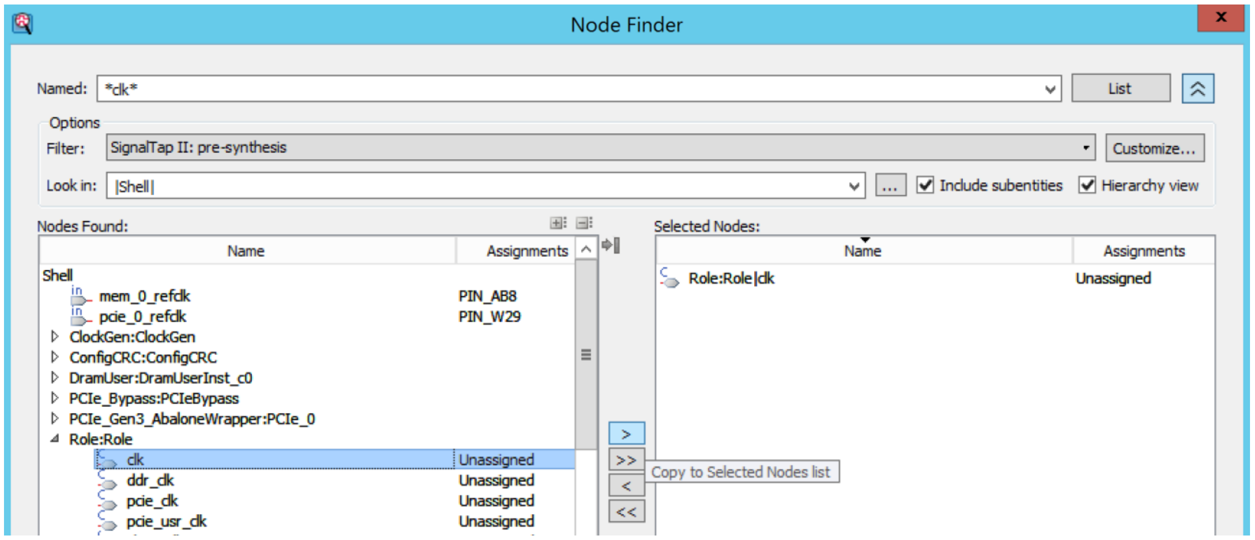


Named: *

Filter: SignalTap II: pre-synthesis

- Pins: virtual
- Pins: all
- Pins: all & Registers: post-fitting
- Registers: pre-synthesis
- Registers: post-fitting
- Post-synthesis
- Post-Compilation
- SignalTap II: pre-synthesis
- SignalTap II: post-fitting
- SignalProbe

Now, search for the role clock node. We know that the role clock is named "clk", but node names are unique, so the actual role clock name is "Role:Role|clk". To avoid the hassle of figuring out the name, you can use wildcards to find all nodes with "clk". Once found, highlight the proper clock, click the ">" button, which copies the node to the Selected Nodes list. Finalize by clicking "OK". You should now see "Role:Role|clk" listed in the "Clock" field under "Signal Configuration".



Named: *clk*

Filter: SignalTap II: pre-synthesis

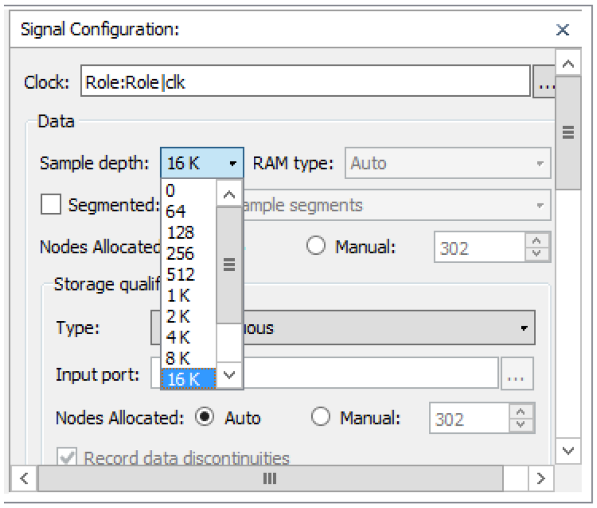
Look in: |Shell|

Name	Assignments
Shell	
mem_0_refclk	PIN_AB8
pcie_0_refclk	PIN_W29
ClockGen:ClockGen	
ConfigCRC:ConfigCRC	
DramUser:DramUserInst_c0	
PCIe_Bypass:PCIeBypass	
PCIe_Gen3_AbaloneWrapper:PCIe_0	
Role:Role	
clk	Unassigned
ddr_clk	Unassigned
pcie_clk	Unassigned
pcie_usr_clk	Unassigned

Selected Nodes:

Name	Assignments
Role:Role|clk	Unassigned

Next, select the number of cycles of data you wish to capture:


Signal Configuration:

Clock: Role:Role|clk

Data

Sample depth: 16 K

RAM type: Auto

Segmented: ample segments

Nodes Allocated · Manual: 302

Storage qualif

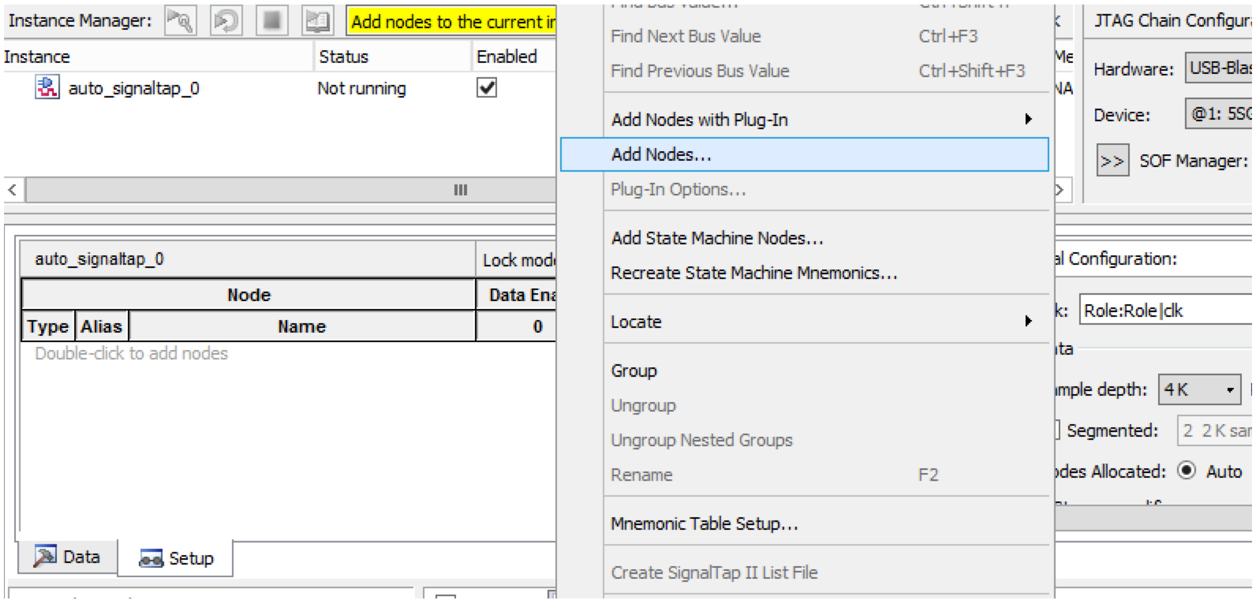
Type: ious

Input port:

Nodes Allocated: (● Auto) ○ Manual: 302

Record data discontinuities

Next, it’s time to add signals. Right-click in the “auto_signaltap_0” table whitespace and select “Add Nodes…”.


Instance Manager: Add nodes to the current ir

Instance	Status	Enabled
auto_signaltap_0	Not running	✓

auto_signaltap_0 · Lock mod

Node			Data Ena
Type	Alias	Name	0
Double-click to add nodes			

JTAG Chain Configur

Hardware: USB-Blas

Device: @1: 5SG

SOF Manager:

l Configuration:

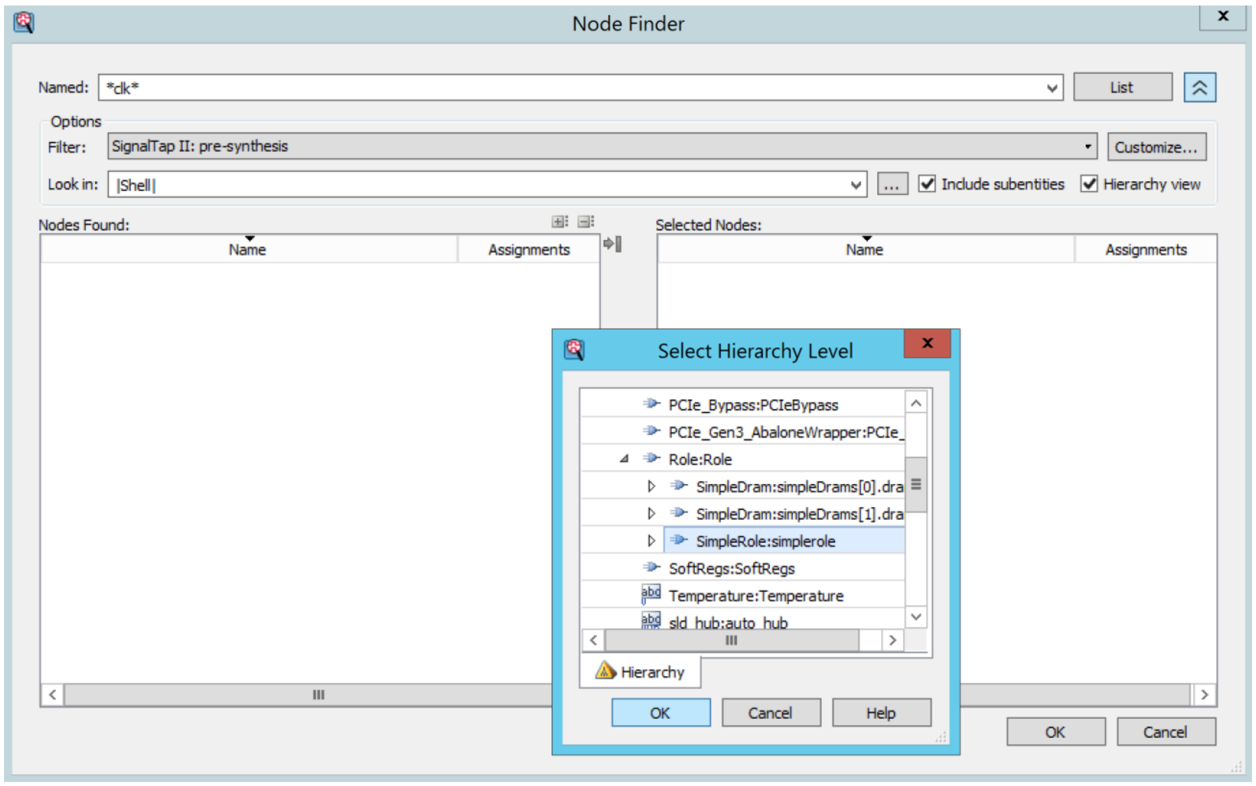
k: Role:Role|clk

mple depth: 4 K

Segmented: 2 2 K sar

odes Allocated: (● Auto)

This will bring you to the Node Finder again. This time, restrict the search space by finding only signals within the SimpleRole. Click on “…” next to the “Look In” field to look only at signals within a specific module. Then, expand “Role” and select “SimpleRole”.



Node Finder

Named: *clk*

Options

Filter: SignalTap II: pre-synthesis

Look in: |Shell|

☑ Include subentities ☑ Hierarchy view

Nodes Found:

Name	Assignments

Selected Nodes:

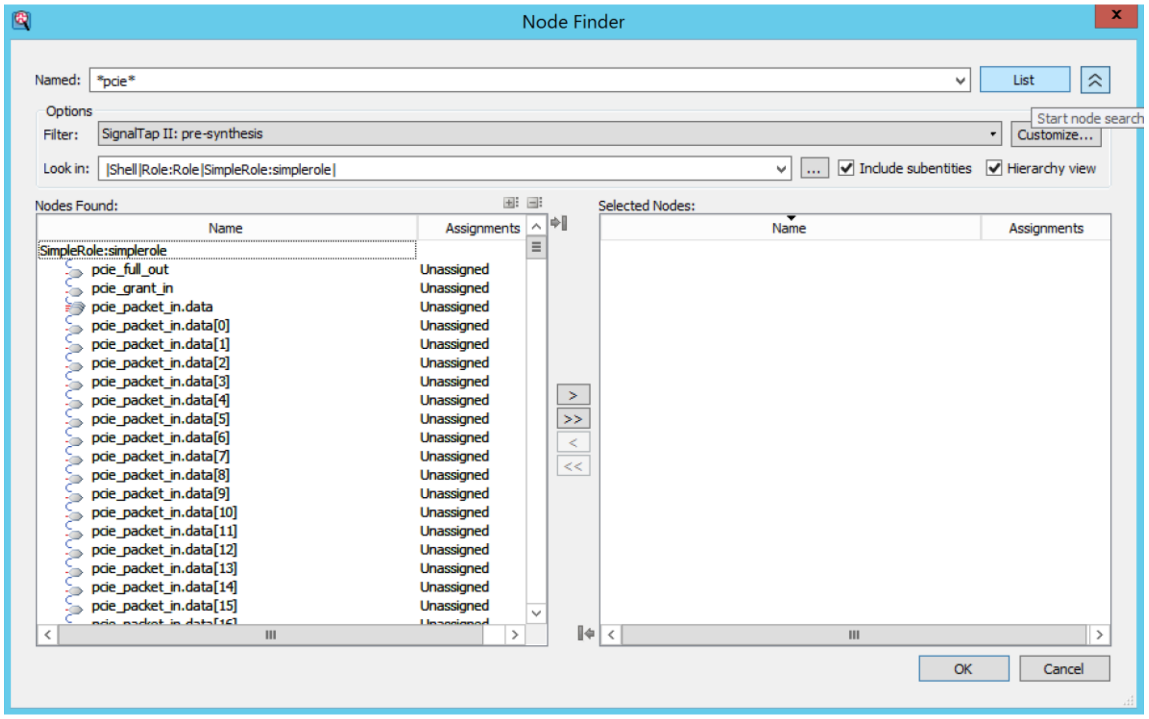
Name	Assignments

Select Hierarchy Level

```
    PCIe_Bypass:PCIeBypass
    PCIe_Gen3_AbaloneWrapper:PCIe_
  ◢ Role:Role
    ▷ SimpleDram:simpleDrams[0].dra
    ▷ SimpleDram:simpleDrams[1].dra
    ▷ SimpleRole:simplerole
    SoftRegs:SoftRegs
    Temperature:Temperature
    sld_hub:auto_hub
```

Hierarchy

To monitor the PCIe signals, enter “*pcie*” in the “Named” field and click “List”. This operation may take some time.



Node Finder

Named: *pcie*

Options

Filter: SignalTap II: pre-synthesis

Look in: |Shell|Role:Role|SimpleRole:simplerole|

☑ Include subentities ☑ Hierarchy view

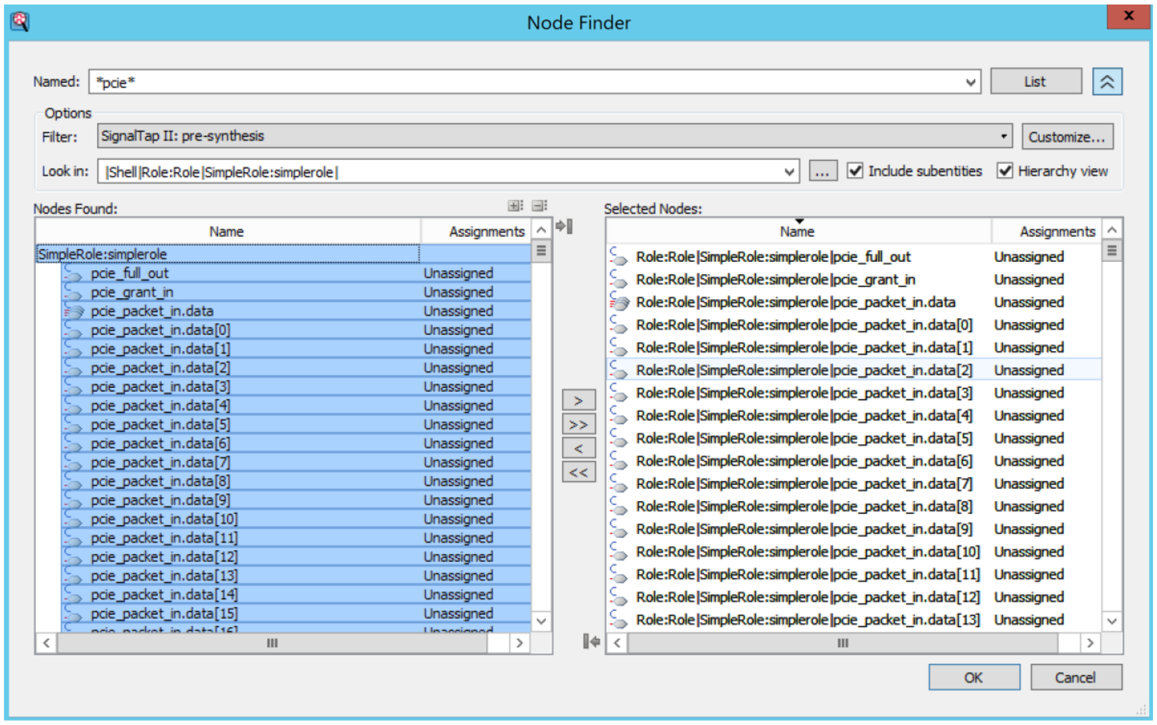
Nodes Found:

Name	Assignments
SimpleRole:simplerole	
pcie_full_out	Unassigned
pcie_grant_in	Unassigned
pcie_packet_in.data	Unassigned
pcie_packet_in.data[0]	Unassigned
pcie_packet_in.data[1]	Unassigned
pcie_packet_in.data[2]	Unassigned
pcie_packet_in.data[3]	Unassigned
pcie_packet_in.data[4]	Unassigned
pcie_packet_in.data[5]	Unassigned
pcie_packet_in.data[6]	Unassigned
pcie_packet_in.data[7]	Unassigned
pcie_packet_in.data[8]	Unassigned
pcie_packet_in.data[9]	Unassigned
pcie_packet_in.data[10]	Unassigned
pcie_packet_in.data[11]	Unassigned
pcie_packet_in.data[12]	Unassigned
pcie_packet_in.data[13]	Unassigned
pcie_packet_in.data[14]	Unassigned
pcie_packet_in.data[15]	Unassigned

Selected Nodes:

Name	Assignments

Copy all the signals over by clicking on “>>”. Confirm the selection by clicking “OK”.



Named: *pcie*

Options

Filter: SignalTap II: pre-synthesis

Look in: |Shell|Role:Role|SimpleRole:simplerole|

☑ Include subentities ☑ Hierarchy view

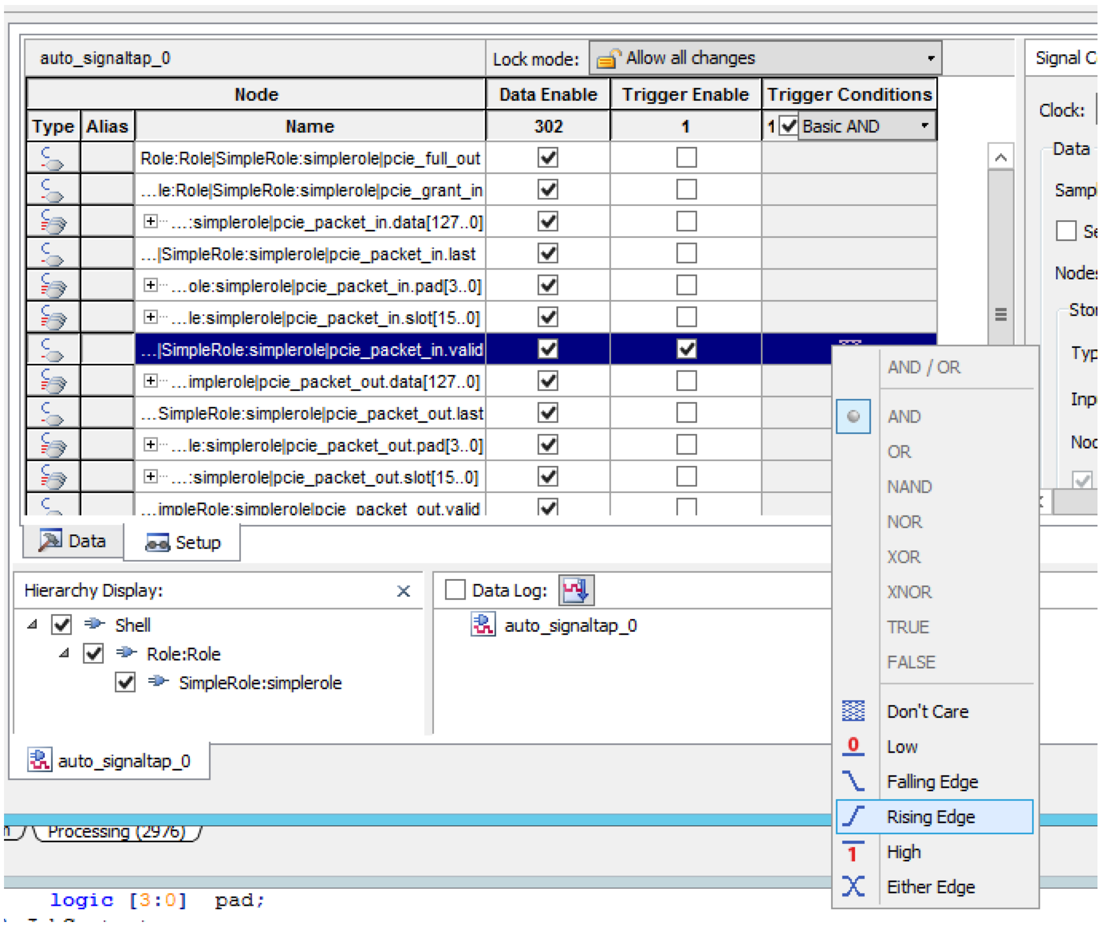
Nodes Found:

Name	Assignments
SimpleRole:simplerole	
pcie_full_out	Unassigned
pcie_grant_in	Unassigned
pcie_packet_in.data	Unassigned
pcie_packet_in.data[0]	Unassigned
pcie_packet_in.data[1]	Unassigned
pcie_packet_in.data[2]	Unassigned
pcie_packet_in.data[3]	Unassigned
pcie_packet_in.data[4]	Unassigned
pcie_packet_in.data[5]	Unassigned
pcie_packet_in.data[6]	Unassigned
pcie_packet_in.data[7]	Unassigned
pcie_packet_in.data[8]	Unassigned
pcie_packet_in.data[9]	Unassigned
pcie_packet_in.data[10]	Unassigned
pcie_packet_in.data[11]	Unassigned
pcie_packet_in.data[12]	Unassigned
pcie_packet_in.data[13]	Unassigned
pcie_packet_in.data[14]	Unassigned
pcie_packet_in.data[15]	Unassigned

Selected Nodes:

Name	Assignments
Role:Role|SimpleRole:simplerole|pcie_full_out	Unassigned
Role:Role|SimpleRole:simplerole|pcie_grant_in	Unassigned
Role:Role|SimpleRole:simplerole|pcie_packet_in.data	Unassigned
Role:Role|SimpleRole:simplerole|pcie_packet_in.data[0]	Unassigned
Role:Role|SimpleRole:simplerole|pcie_packet_in.data[1]	Unassigned
Role:Role|SimpleRole:simplerole|pcie_packet_in.data[2]	Unassigned
Role:Role|SimpleRole:simplerole|pcie_packet_in.data[3]	Unassigned
Role:Role|SimpleRole:simplerole|pcie_packet_in.data[4]	Unassigned
Role:Role|SimpleRole:simplerole|pcie_packet_in.data[5]	Unassigned
Role:Role|SimpleRole:simplerole|pcie_packet_in.data[6]	Unassigned
Role:Role|SimpleRole:simplerole|pcie_packet_in.data[7]	Unassigned
Role:Role|SimpleRole:simplerole|pcie_packet_in.data[8]	Unassigned
Role:Role|SimpleRole:simplerole|pcie_packet_in.data[9]	Unassigned
Role:Role|SimpleRole:simplerole|pcie_packet_in.data[10]	Unassigned
Role:Role|SimpleRole:simplerole|pcie_packet_in.data[11]	Unassigned
Role:Role|SimpleRole:simplerole|pcie_packet_in.data[12]	Unassigned
Role:Role|SimpleRole:simplerole|pcie_packet_in.data[13]	Unassigned

You’ll see that the signal node table has now been populated with your selections. Since we are monitoring an incoming PCIe packet, we should begin monitoring when the incoming PCIe valid bit transitions from 0 to 1. First, de-select “Trigger Enable” for all signals except for the one that we care about: pcie_packet_in.valid. Next, right-click on the “Trigger Conditions” field for pcie_packet_in.valid and select “Rising Edge”.



auto_signaltap_0 | Lock mode: Allow all changes

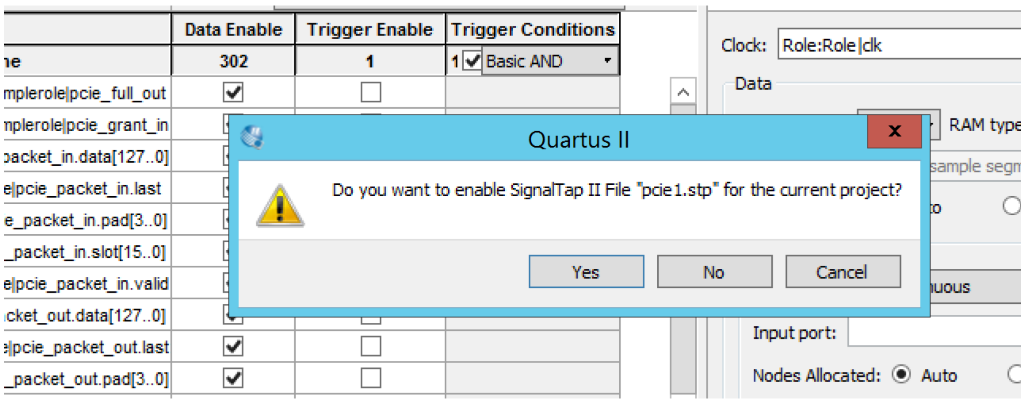
Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	302	1	1 ✓ Basic AND
		Role:Role|SimpleRole:simplerole|pcie_full_out	✓	☐	
		...le:Role|SimpleRole:simplerole|pcie_grant_in	✓	☐	
		⊞ ...:simplerole|pcie_packet_in.data[127..0]	✓	☐	
		...|SimpleRole:simplerole|pcie_packet_in.last	✓	☐	
		⊞ ...ole:simplerole|pcie_packet_in.pad[3..0]	✓	☐	
		⊞ ...le:simplerole|pcie_packet_in.slot[15..0]	✓	☐	
		...|SimpleRole:simplerole|pcie_packet_in.valid	✓	✓	
		⊞ ...implerole|pcie_packet_out.data[127..0]	✓	☐	
		...SimpleRole:simplerole|pcie_packet_out.last	✓	☐	
		⊞ ...le:simplerole|pcie_packet_out.pad[3..0]	✓	☐	
		⊞ ...:simplerole|pcie_packet_out.slot[15..0]	✓	☐	
		...impleRole:simplerole|pcie_packet_out.valid	✓	☐	

Hierarchy Display:
- ✓ Shell
  - ✓ Role:Role
    - ✓ SimpleRole:simplerole

Data Log: auto_signaltap_0

```
logic [3:0]  pad;
```

Next, save the project (File -> Save As) and give the file a unique name. Quartus will ask if you wish to enable this SignalTap file for your project. Select “Yes”.

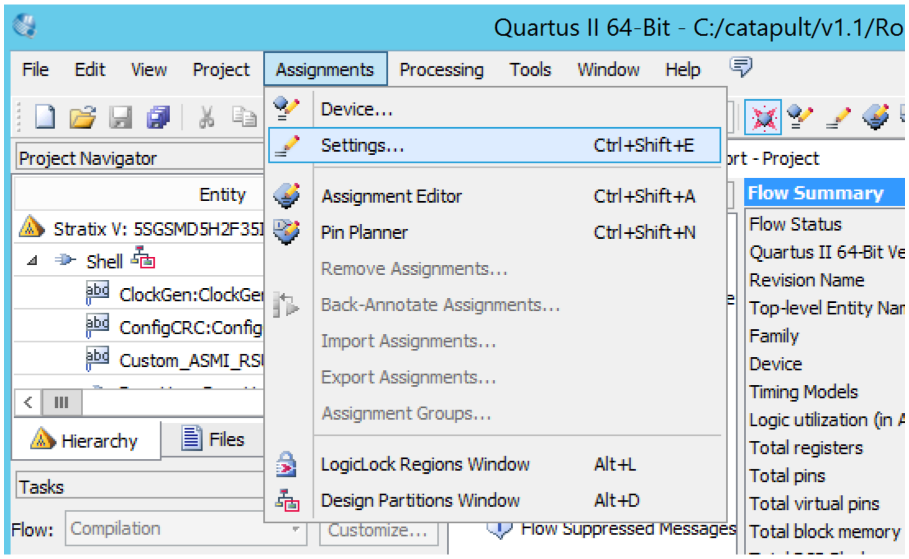


Clock: Role:Role|clk

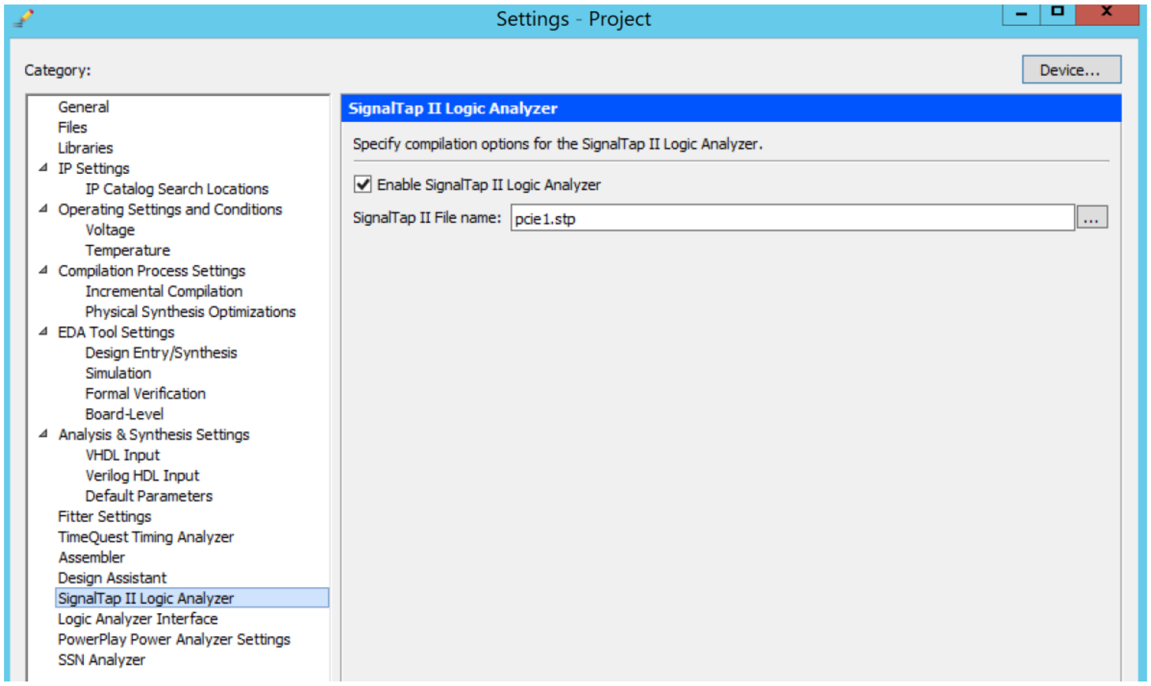
Quartus II

Do you want to enable SignalTap II File "pcie1.stp" for the current project?

If you accidentally hit “No” or you would like to make sure that the correct SignalTap file is enabled in your project, go to Assignments -> Settings in the main Quartus window.



Next, select “SignalTap II Logic Analyzer” as the Category and make sure “Enable SignalTap II Logic Analyzer” is selected, and that the file name is correct.

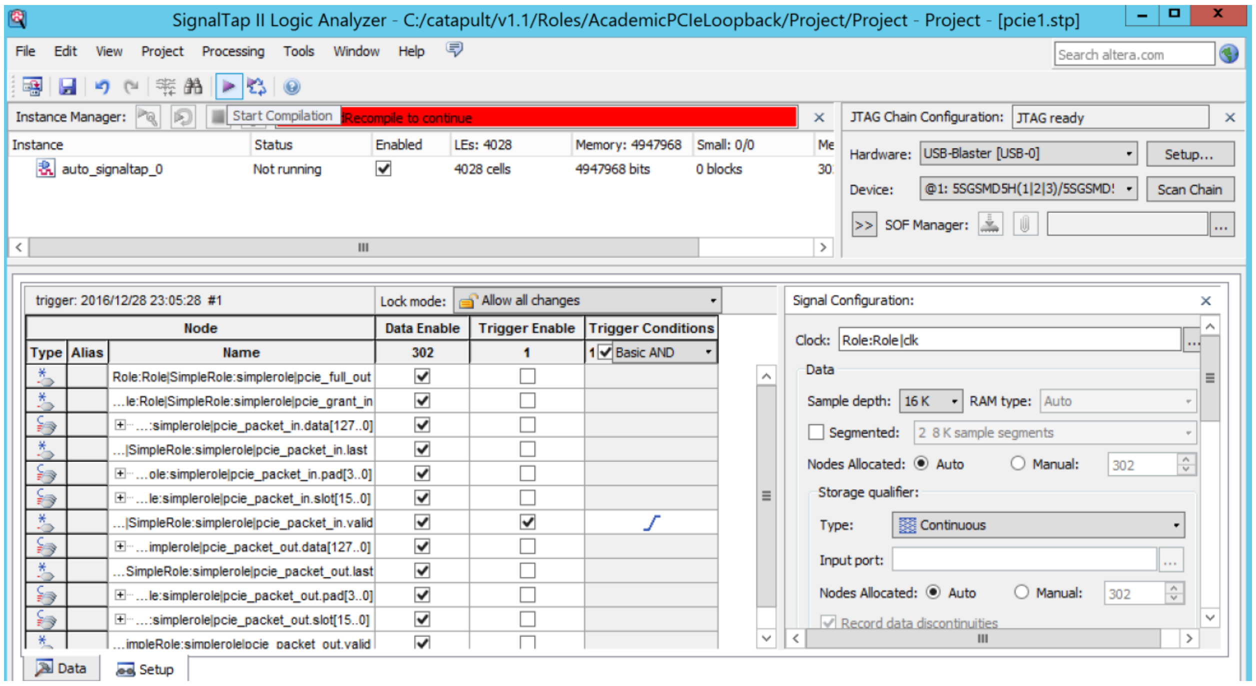


Specify compilation options for the SignalTap II Logic Analyzer.

[✓] Enable SignalTap II Logic Analyzer

SignalTap II File name: pcie1.stp

We are now ready to recompile our design with SignalTap. In theory, you can select “Rapid Recompile” to only recompile the SignalTap portion of the design. However, we have found in practice that Rapid Recompile will sometimes fail with errors late in the recompilation process, wasting a lot of time. Therefore, we recommend performing a full re-compile. Perform the full re-compile by clicking on the “Start Compilation” button.



SignalTap II Logic Analyzer - C:/catapult/v1.1/Roles/AcademicPCIeLoopback/Project/Project - Project - [pcie1.stp]

Instance Manager: Start Compilation Recompile to continue

JTAG Chain Configuration: JTAG ready

Instance	Status	Enabled	LEs: 4028	Memory: 4947968	Small: 0/0
auto_signaltap_0	Not running	✓	4028 cells	4947968 bits	0 blocks

Hardware: USB-Blaster [USB-0]

Device: @1: 5SGSMD5H(1|2|3)/5SGSMD!

trigger: 2016/12/28 23:05:28 #1

Lock mode: Allow all changes

Type	Alias	Name	Data Enable	Trigger Enable	Trigger Conditions
			302	1	1 ✓ Basic AND
		Role:Role|SimpleRole:simplerole|pcie_full_out	✓	☐	
		...le:Role|SimpleRole:simplerole|pcie_grant_in	✓	☐	
		...:simplerole|pcie_packet_in.data[127..0]	✓	☐	
		...|SimpleRole:simplerole|pcie_packet_in.last	✓	☐	
		...ole:simplerole|pcie_packet_in.pad[3..0]	✓	☐	
		...le:simplerole|pcie_packet_in.slot[15..0]	✓	☐	
		...|SimpleRole:simplerole|pcie_packet_in.valid	✓	✓	⎽/
		...implerole|pcie_packet_out.data[127..0]	✓	☐	
		...SimpleRole:simplerole|pcie_packet_out.last	✓	☐	
		...le:simplerole|pcie_packet_out.pad[3..0]	✓	☐	
		...:simplerole|pcie_packet_out.slot[15..0]	✓	☐	
		...impleRole:simplerole|pcie_packet_out.valid	✓	☐	

Signal Configuration:

Clock: Role:Role|clk

Data

Sample depth: 16 K RAM type: Auto

☐ Segmented: 2 8 K sample segments

Nodes Allocated: [● Auto] ○ Manual: 302

Storage qualifier:

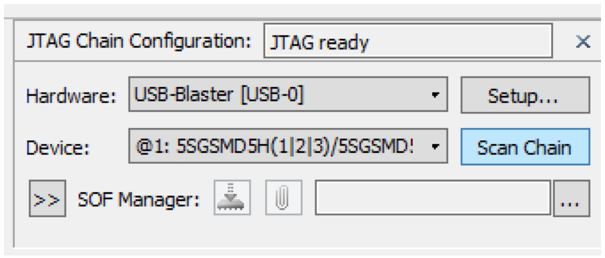
Type: Continuous

Input port:

Nodes Allocated: [● Auto] ○ Manual: 302

✓ Record data discontinuities

After compilation is completed, generate the RPD file (genRpd.ps1) and write the image to FPGA (-write, -reconfig). See Catapult TACC Getting Started Guide for more details. Once the image is loaded onto the FPGA, click the “Scan Chain” button under “JTAG Chain Configuration”.

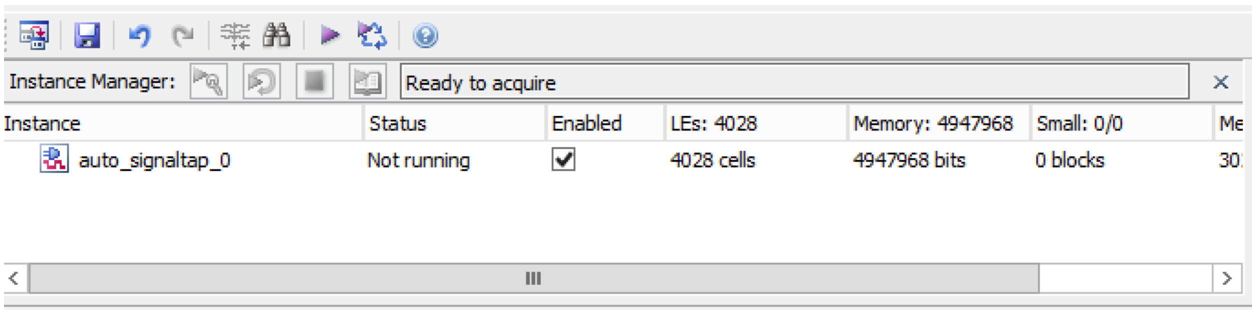


JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0]

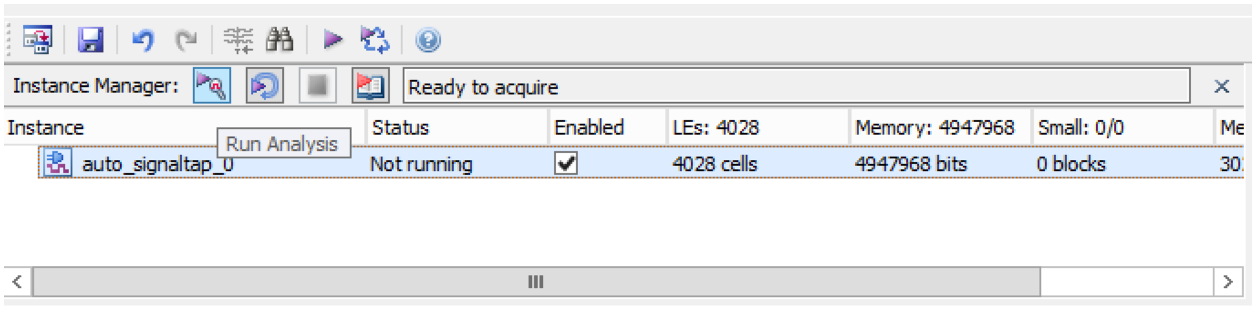
Device: @1: 5SGSMD5H(1|2|3)/5SGSMD!

If the image instrumented with SignalTap has been properly programmed to the FPGA, the status will change from “Program the device to continue” to “Ready to acquire”.


Instance Manager: Ready to acquire

Instance	Status	Enabled	LEs: 4028	Memory: 4947968	Small: 0/0	Me
auto_signaltap_0	Not running	✓	4028 cells	4947968 bits	0 blocks	30

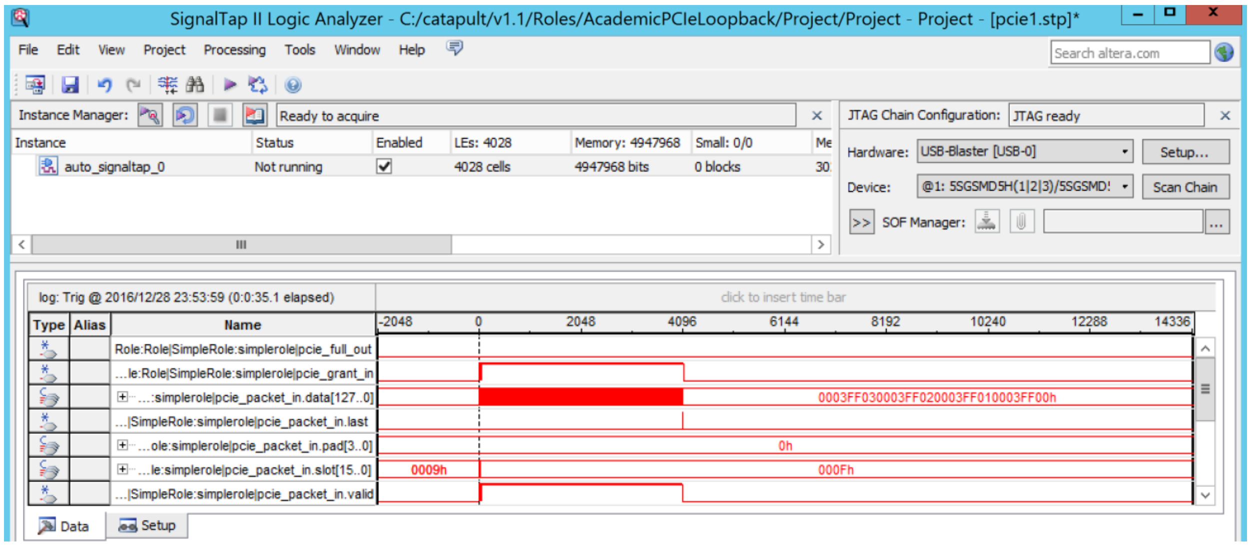
Select the instance and click “Run analysis”. The status will transition to “Acquisition in progress”.


Instance Manager: Ready to acquire

Run Analysis

Instance	Status	Enabled	LEs: 4028	Memory: 4947968	Small: 0/0	Me
auto_signaltap_0	Not running	✓	4028 cells	4947968 bits	0 blocks	30

SignalTap is now waiting for the trigger condition to be met. Run your application in a separate window. When the trigger condition is met, SignalTap will collect the data and show the waveform results in the “Data” tab. Congratulations, you have now successfully run SignalTap!


SignalTap II Logic Analyzer - C:/catapult/v1.1/Roles/AcademicPCIeLoopback/Project/Project - Project - [pcie1.stp]*

Instance Manager: Ready to acquire

Instance	Status	Enabled	LEs: 4028	Memory: 4947968	Small: 0/0	Me
auto_signaltap_0	Not running	✓	4028 cells	4947968 bits	0 blocks	30

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0]

Device: @1: 5SGSMD5H(1|2|3)/5SGSMD!

log: Trig @ 2016/12/28 23:53:59 (0:0:35.1 elapsed)

Name	Value
Role:Role|SimpleRole:simplerole|pcie_full_out	
...le:Role|SimpleRole:simplerole|pcie_grant_in	
...:simplerole|pcie_packet_in.data[127..0]	0003FF030003FF020003FF010003FF00h
...|SimpleRole:simplerole|pcie_packet_in.last	
...ole:simplerole|pcie_packet_in.pad[3..0]	0h
...le:simplerole|pcie_packet_in.slot[15..0]	0009h, 000Fh
...|SimpleRole:simplerole|pcie_packet_in.valid	