VexRISC-V processor implementation on Intel FPGA PAC Card

Report

Part of Intel Research Fellowship

By

Khyamling

Under the Mentorship of

Rajesh Vivekanandam and Israr Ahmed Sheikh

Intel, Bangalore, Karntaka, India

August, 2021

Contents

1 Introduction

RISC-V is an emerging instruction-set architecture suitable for a wide variety of applications, which ranges from simple microcontrollers to high-performance CPUs. RISC-V is an open ISA, modern, extensible and it has a comprehensive infrastructure of open-source specifications, compilers, libraries, operating systems, and Interface IP. RISC-V processor implementations range from a single processor to heterogeneous multiprocessor architecture. Additionally, soft processor implementation allows the ISA to be customized and extended to suit a specific application. This brings the benefits of making the FPGA easier to program by presenting the view of a conventional multi core CPU. Simultaneously, it also enabling FPGA only custom logic optimizations for the key performance sensitive components of the workloads rather than forcing the entire code base to be ported to HDL. In this report, a 32-bit VexRiscv based on RV32I CPU instruction set has been designed using Intel PAC card. The ISA of VeXRiscv was extended to support multiple additional instructions such as standard Extension and privileged instruction.
Further, we have analyzed various RISC-V soft and hard processors by considering the design parameters, frequency, area, and performance. The VexRISC-V processor has emerged as the state-of-the-art FPGA optimized soft processor. Finally, the design has been examined on the Intel FPGA PAC card such as Arria 10 and Stratix 10. The logic synthesis, placement, and routing were performed incrementally changing the clock constraints. The FPGA area utilization and the maximum operating frequency of VexRISC-V. To enhance the performance of the VexRISC-V processor on the Stratix 10 FPGA board, the FPGA optimization strategies, HyperFlex register, and HyperFlex pipelining techniques were employed. After synthesis, placement, and routing, we observed that the optimized design has a higher operating frequency by utilizing the same area. The VexRISC-V implemented on the Agilex FPGA board with HyperFlex techniques achieves the maximum operating frequency 25% higher than Intel Arria 10 PAC card.

2 Generate the source code of VexRISC-V

We did an extensive survey of overlay and soft processor architectures targeting FPGAs and implementing the RISC-V ISA. Based on the survey, we notice that the VexRISC-V processor design is an FPGA optimized soft processor compared to the state-of-the-art. The VexRISC-V architecture is classified into different CPU instances based on size and performance, which is range from small CPU to CPU with Linux balanced. We choose the VexRISC-V full instance of CPU for this project work, which offers a high performance by consuming less area.

The VexRISC-V full Instance features:

- RV32IM instruction set
- 5-stages
- 4KB-ICache,4KB-Dcache.
- Single cycle barrel shifter.
- Debug module
- Catch exceptions
- Static branch

2.1 Steps for generating RTL code of VexRISC-V

First, Install dependencies software on the Linux platform. The Java JDK 8 and Scala Sbt Software are installed already on the Linux platform, then start with step3. Otherwise, follow steps 1 to 4.
1. JAVA JDK 8

- sudo add-apt-repository -y ppa:openjdk-r/ppa
- sudo apt-get update
- sudo apt-get install openjdk-8-jdk -y
- sudo update-alternatives –config java
- sudo update-alternatives –config javac

2.Install scala SBT -(https://www.scala-sbt.org/)

- sudo apt-get update
- sudo apt-get install sbt

3. Download the VexRISC-V repository using below link

- https://github.com/SpinalHDL/VexRiscv

4. To generate the corresponding RTL as a VexRiscv.v file, run the below commands in the root directory of VexRISC-V repository

- sbt "runMain vexriscv.demo.GenFull"

The generated RTL code of VexRISC-V is used for building the softcore processor on the Intel PAC card. The implementation details of VexRISC-V as an Accelerator Functional Unit(AFU) on the Intel PAC card are explained in section(3).

3 Devcloud Access Instructions

3.1 Introduction

The FPGA Devcloud is an Intel hosted cloud service with Intel XEON processors and FPGA acceleration cards. The FPGA Cloud has a number of development tools installed including Acceleration Stack, and Quartus Prime Lite / Prime Pro development tools. The FPGA Cloud hosts high end FPGA accelerator cards to allow users to experiment with accelerated workloads running on FPGAs.

These instructions detail how to connect your Windows/Linux PC to the devcloud which runs Linux. If you are attempting connectivity from a Mac or Linux machine, we do not offer exact instructions, but you can use the ssh key and config file details described here with other ssh terminals.

When you signed up for the Intel Devcloud, you were presented with three different instances to sign-up for: IOTG, ONEAPI or FPGA. IOTG is a standalone setup with its own login. FPGA is a superset of the ONEAPI instance. At the time of this writing, ONEAPI has 225 nodes and FPGA has 12 nodes. With the FPGA instance you can run all of the features of the ONEAPI instance and gain access to an additional 12 machines that have FPGA PAC cards and associated tools.

This set of instructions superscedes the instructions on the OneAPI login instruction site. Once you download your devcloud access key, use these instructions. Also note that if you are inside the Intel firewall, the instructions differ so pay close attention to the changes in the config file if you are an Intel employee accessing the FPGA devcloud.

Note: Please allow 60-90 mins to complete the entire setup. Also note the default account duration is 90 days. You will get reminders to renew your account if you need access for more than 90 days. If you allow your account to expire, you will lose the data in your account. We recommend copying source files back to your local machine for safe keeping.

3.2 Getting an Account

If you already have a Devcloud account, go to Connection Methods. If you do not have a Devcloud account, please use this cloud website landing page to submit a request to access the FPGA Cloud:
https://intelsoftwaresites.secure.force.com/fpgadevcloud

Once you've signed up, you should get an immidate screen response with your new user ID and instructions on how to set up your account. You will also receive a follow up email from Intel Devcloud which can take 1 hr giving you a record of your user ID and your user ID key for login. This is an example of the resulting email which will be sent to you:

Welcome "user name",

We are excited that you chose Intel® FPGA Cloud. Free access. No downloads. No installations. No maintenance.

Your account should already be activated. Below are your credentials for your reference. Unique Access URL: https://devcloud.intel.com/fpga/?uuid=cd1d...

User ID: u12345

UUID Key: cd1d...

Access to Intel® FPGA DevCloud typically expires after 120 days. However, longer access times are granted based on user request. To extend this access, please email fpgauniversity@intel.com and give the extension time needed to complete your project.

Getting started with FPGA Cloud:

- $Accessdetailedinstructionsforenvironmentsetup : https : //github.com/intel/FPGA-Devcloud/tree/master/main/Devcloud\_Access\_Instructionsdevcloud-access-instructions$
- $Ifyouhavetechnicalquestionsorrecommendations, pleasepostthemtoourFPGAforum :$
- $https : //forums.intel.com/s/topic/0TO0P000000MWKFWA4/application-acceleration-with-fpgas$
- If you have unresolved issues, email fpgauniversity@intel.com and give us a detailed description of your problem.

It's all about you and your code. We look forward to the innovations you'll create.
- Your friendly Intel DevCloud Team

3.3 Connection Methods

Once you have an account / email received you are ready to start the process to setup your account within the cloud.

There are different methods of terminal connections. Listed below are a few options you can select in choosing which Terminal application tool you would like to use. This guide focuses on the recommended method 1.

- Linux SSH client
- Windows with Mobaxterm (SSH client)
- Windows with Cygwin
- Windows with PuTTy
- Jupyter Lab (For FPGA devcloud connectivity ssh login-2 to access the head node and follow login script instructions)

3.4 Downloading an SSH key

To start the process:

1. Click on the following link to access the Connect website:
https://devcloud.intel.com/fpga/connect/.

2. Once you sign in, the following page will then be displayed. Click on the "SSH key for Linux/macOS/Cygwin" blue button under Step 3: Get SSH Key to download your key.

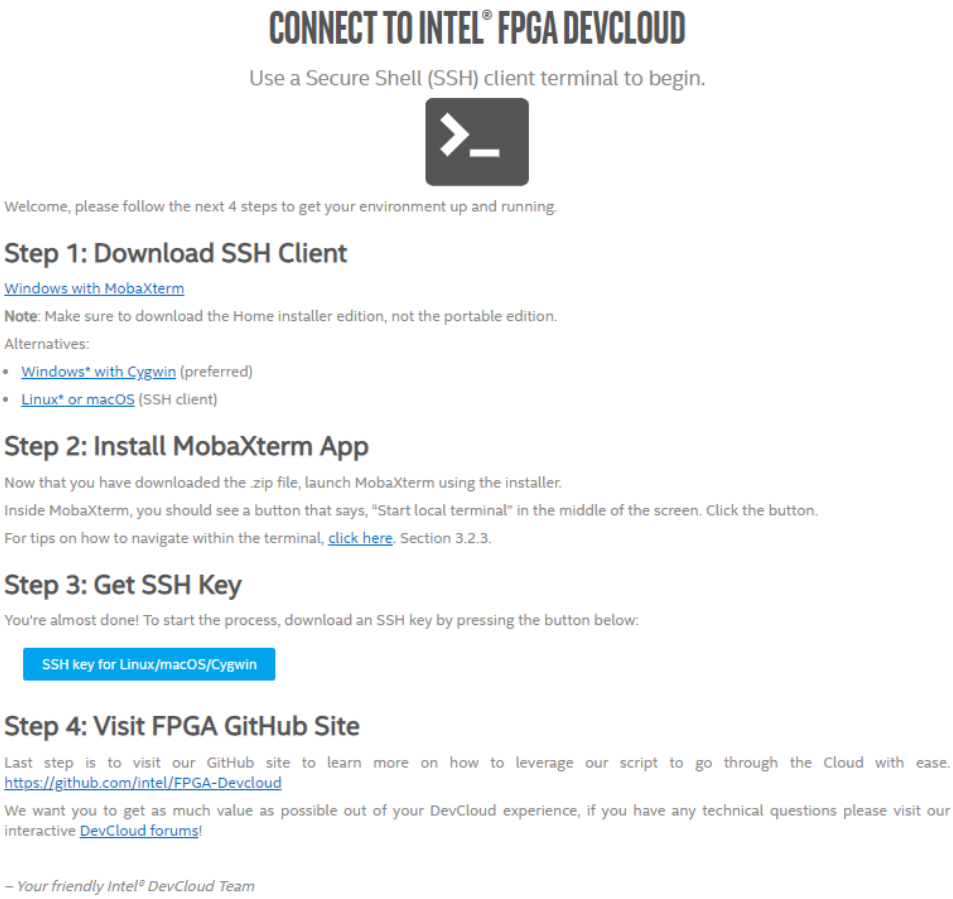

Figure 1: Downloading an SSH key

3. Once you have downloaded the SSH key, follow the below Instructions.

4. Create the directory /.ssh, unless it already exists and move the private SSH key into permanent storage in /.ssh:

```
mkdir -p /.ssh
mv /drives/c/Users/¡user¿/Downloads/devcloud-access-key-12345.txt /.ssh/
```

5. Add the following lines to files /.ssh/config:

```
Host devcloud
// replace with your own user name
User u12345
IdentityFile /.ssh/devcloud-access-key-12345.txt
ProxyCommand ssh -T -i /.ssh/devcloud-access-key-12345.txt
guest@ssh.devcloud.intel.com
```

If you saved your key in a location other than /drives/c/Users//Downloads, insert the correct path and the correct user number that was provided to you in the email.

6. Set the correct restrictive permissions on the private SSH. Run the following commands in terminal:

```
chmod 600  /.ssh/devcloud-access-key-u12345.txt
chmod 600  /.ssh/config
```

The next steps to connect to the Intel Devcloud are different for usage inside and outside the Intel Firewall. Select the correct usage option below:

- Public User
- User Inside Intel Firewall

4 VexRISC-V Custom IP Component Creation

- **Open Quartus**
- **Create Quartus Project**
- **Open Platform Designer**

4.1 Open Quartus Tool

Set up the Quartus pro environment variables using below command

export PATH=$PATH:$< installation - directory >$/quartus/sopc_builder/bin/
export PATH=$PATH:$< installation - directory >$/quartus/bin/

Then lauch the Intel Quartus Pro software on Linux platform Use the following method to start Intel software on Linux

- Type the following command at terminal window.

$< installation - directory >$/quartus/bin/quartus

The above command used to start Intel Quartus Prime pro edition software GUI as shown in Fig.2. The Quartus pro software expands on the capabilities of the Quartus prime standard edition, and provides unique features that support the latest Intel FPGAs. The Pro software offers flexible design methodologies, advanced synthesis, and support the latest Intel FPGA architectures and hierarchical design flow. The following features are unique to the Intel Quartus Prime Pro Edition software.

- Hyper-Aware Design Flow: Use Hyper-Retimimg and Fast forward compilation to reach the highest performance in Intel Agilex and Startix devices.
- Intel Quartus Prime Pro Edition synthesis: integrates new, stricter language parser supporting all major IEEE RTL languages, with enhanced algorithms, and parallel synthesis capabilities. Added support for SystemVerilog 2009.
- Hierarchical project structure: preserve individual post-synthesis, post-placement, and post-place and route results for each design instance. Allows optimization without impacting other partition placement or routing.

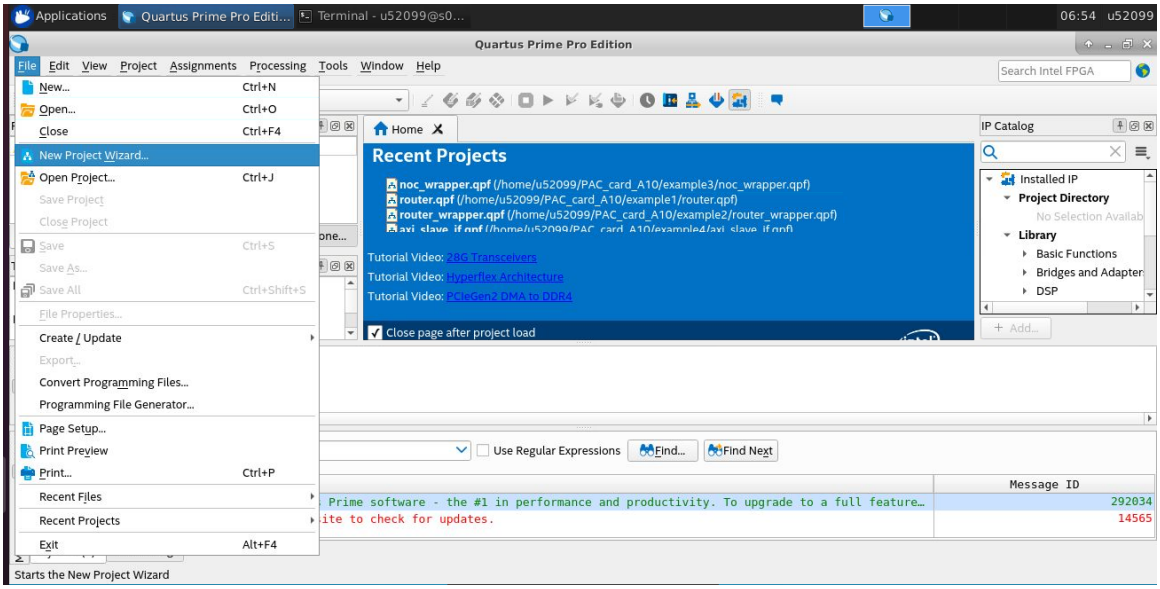

Figure 2: Intel Quartus Prime Pro Edition Software GUI

- Incremental Fitter Optimizations: run and optimize Fitter stages incrementally. Each Fitter stage generates detailed reports.
- Faster, more accurate I/O placement: plan interface I/O in Interface Planner.
- Platform Designer: builds on the system design and custom IP integration capabilities of Platform Designer. Platform Designer in Intel Quartus Prime Pro Edition introduces hierarchical isolation between system interconnect and IP components.
- Partial Reconfiguration: reconfigure a portion of the FPGA, while the remaining FPGA continues to function.
- Block-Based Design Flows: preserve and reuse design blocks at various stages of compilation.

4.2 Create Quartus Project

The pro software organizes and manages the elements of design within a project. The project encapsulates information about your design files, hierarchy, libraries, constraints, and project settings.

To create a new project:
click File> New Project Wizard to quickly setup and open a new project. After we create or open a project, the GUI displays integrated information and controls for the project as shown in Fig. 3

The complete project creation flow as shown in Fig. 4 to 9

4.3 Viewing basic Project Information

We can view the basic information about the project in the Tasks pane, Project Navigator, Compilation Dasboard, Report panel, and Messages window as shown in Fig. 10. The pro software helps to create and manage the logic design files in the project. Logic design files contain the logic that implements the design. To add a logic design to the project by clicking File> New> Verilog HDL File. Finally save the create source file as shown in Fig. 11

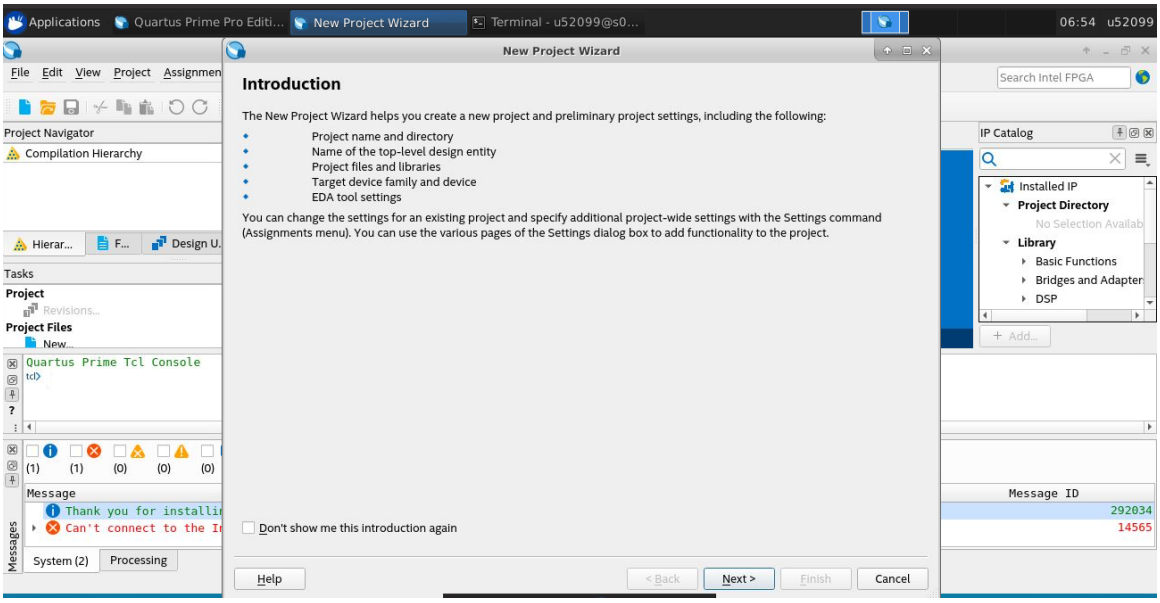

Figure 3: New Project wizard

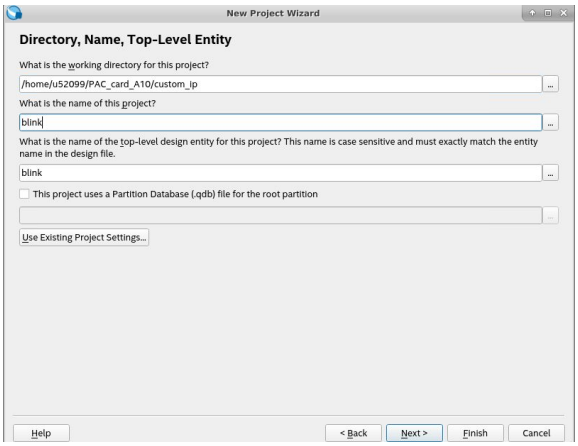

Figure 4: Project directory, name, top level design name

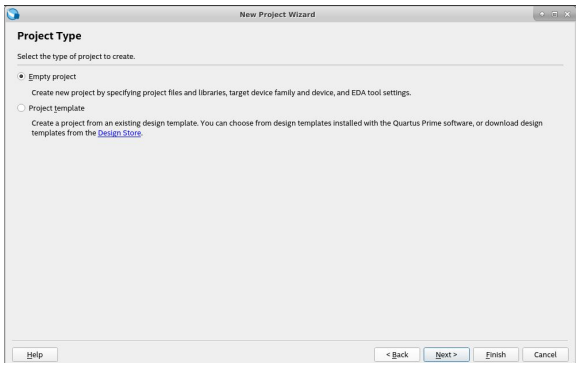

Figure 5: Project Type

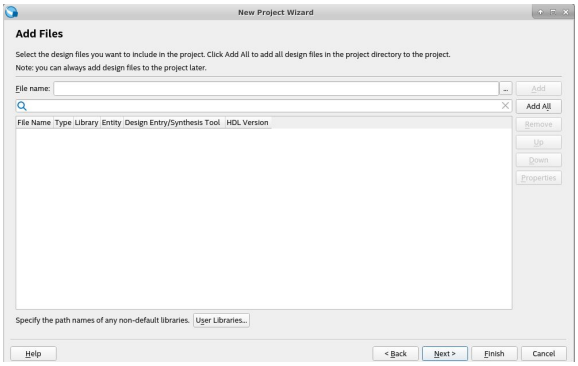

Figure 6: Add RTL file to project

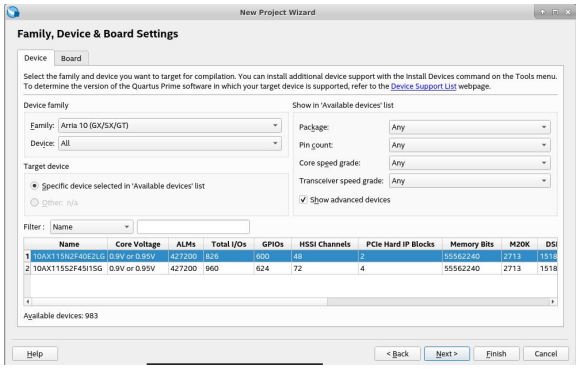

Figure 7: Select FPGA family, device and board

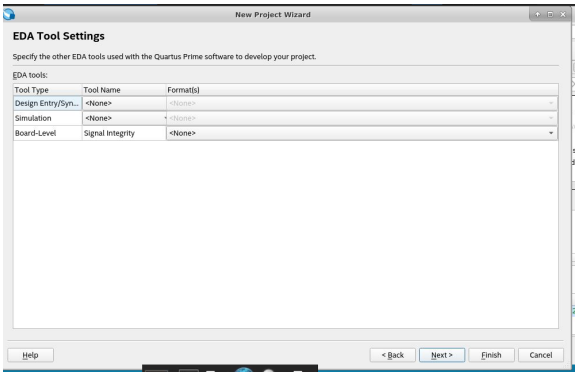

Figure 8: EDA tool setting

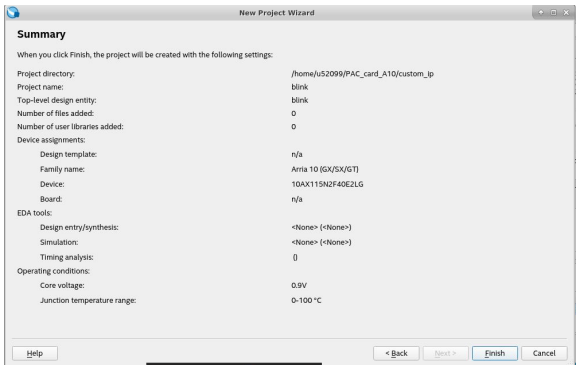

Figure 9: Project summary

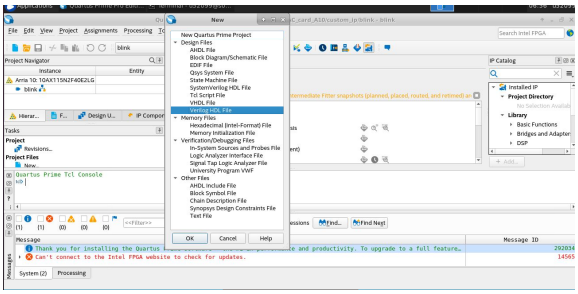

Figure 10: Project window

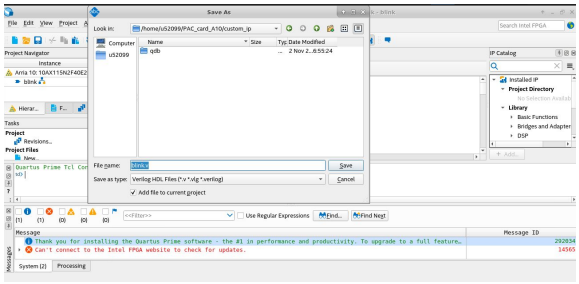

Figure 11: Add source code to project

5 Open Platform Designer

The Intel Quartus Prime pro software includes the platform Designer system integration tool. Platform Designer simplifies the task of defining and integrating custom IP components (IP cores) into FPGA design. Platform Designer automatically creates interconnect logic from high-level connectivity that we specify. The interconnect automation eliminates the time-consuming task of specifying system-level HDL connections.

Open Platform Designer from Tools>Platform Designer as shown in Fig. 12

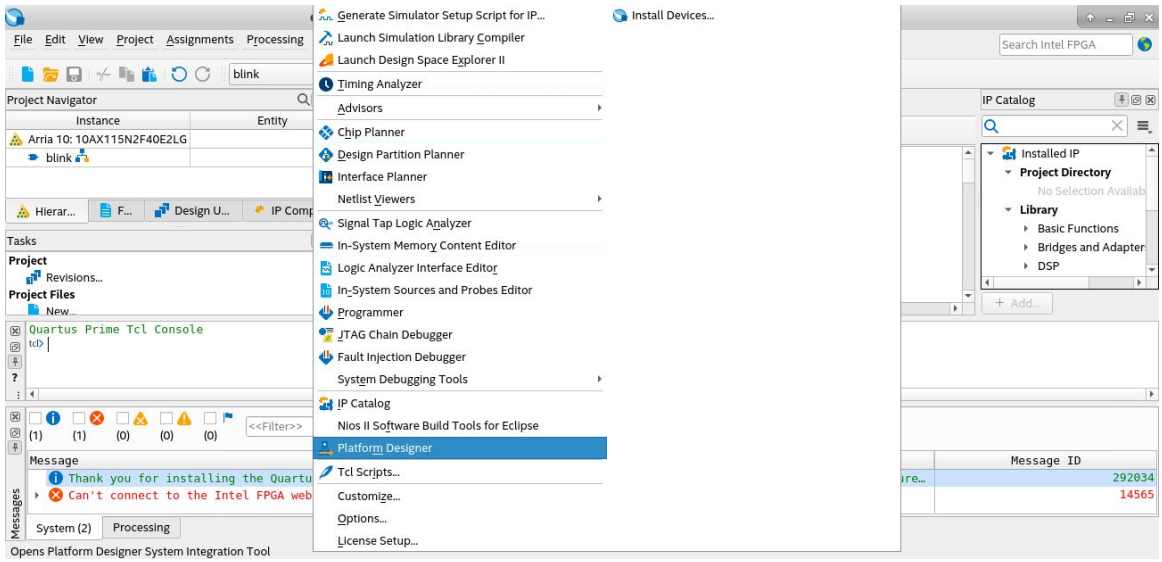

Figure 12: Open Platform Designer

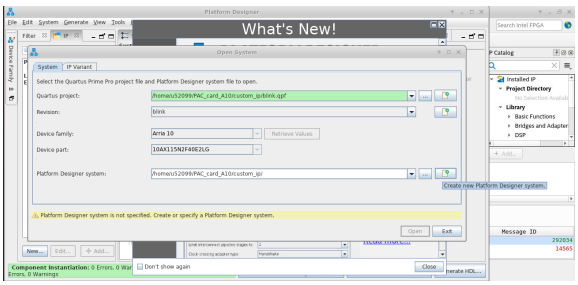

Figure 13: Create Platform Designer

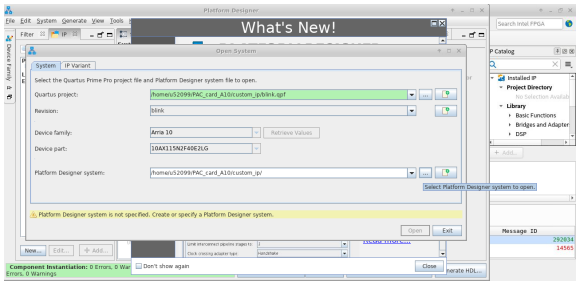

Figure 14: Open Platform Designer

5.1 Creating or opening a platform designer system

To launch platform designer from the Intel Quartus Prime software to create or open a Platform Designer system. When you create or open a system, Platform Designer requires that you specify the Intel Quartus Prime project to contain this system. If this project does not yet exist, you can define a new project from within Platform Designer. Alternatively, you can specify an existing project. When you launch Platform Designer with an Intel Quartus Prime project open, Platform Designer automatically specifies that project by default. After specifying the project, you select an existing system to open, or specify the name of a new system to create.

Follow these steps to create or open a Platform Designer system:

- Select the Platform Designer system, or click the Create New Platform Designer System button as shown in Fig. 13 and specify the name of a new system as shown in Fig. 15 and 16

5.2 Viewing a Platform Designer System

Platform Designer allows you to visualize all aspects of your system as shown in Fig. 17 and 18. By default, Platform Designer displays the contents of your system in the System View whenever you open a system. You can also access other panels that allow you to view and modify various elements of the system. When you select or edit an item in one Platform Designer tab, all other tabs update to reflect your selection or edit.
Click the View menu to interact with the elements of your system in various tabs.

- The System View, Address Map, Interconnect Requirements, and Details tabs display in the main frame.
- By default, the IP Catalog, Hierarchy, and the Device Family tabs appear to the left of the main frame.
- Parameters, System Info, and Component Instantiation tabs appear to the right of the main frame.
- The System Messages and Generation Messages tabs display in the lower portion of Platform Designer

The Platform Designer GUI is fully customizable. You can arrange and display Platform Designer GUI elements that you most commonly use, and then save and reuse useful GUI layouts.

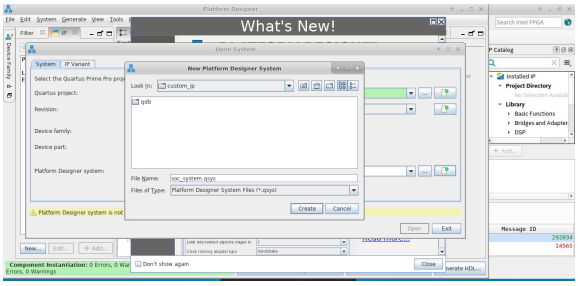

Figure 15: Save the .qsys file Platform Designer

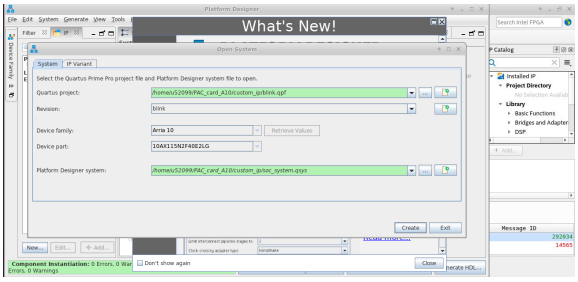

Figure 16: Platform Designer wizard

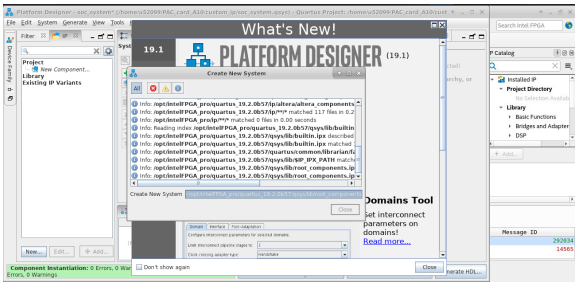

Figure 17: Platform Designer GUI

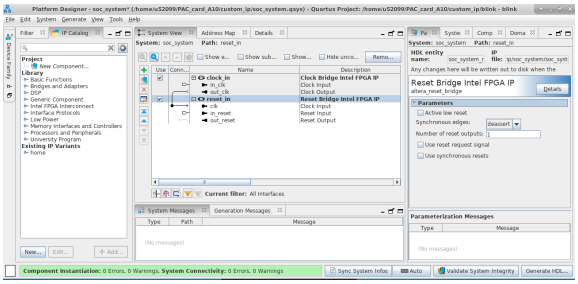

Figure 18: Platform Designer GUI

6 Creating IP Components in the Component Editor

The Platform Designer Component Editor allows you to create and package an IP component and parameterization GUI. When you use the Component Editor to define a component, Platform Designer writes the information to an _hw.tcl file.

The Platform Designer Component Editor, accessed by clicking New Component in the Platform Designer window, allows you to create and package a custom IP component for use in platform design to build SoC system as shown in Fig. 19

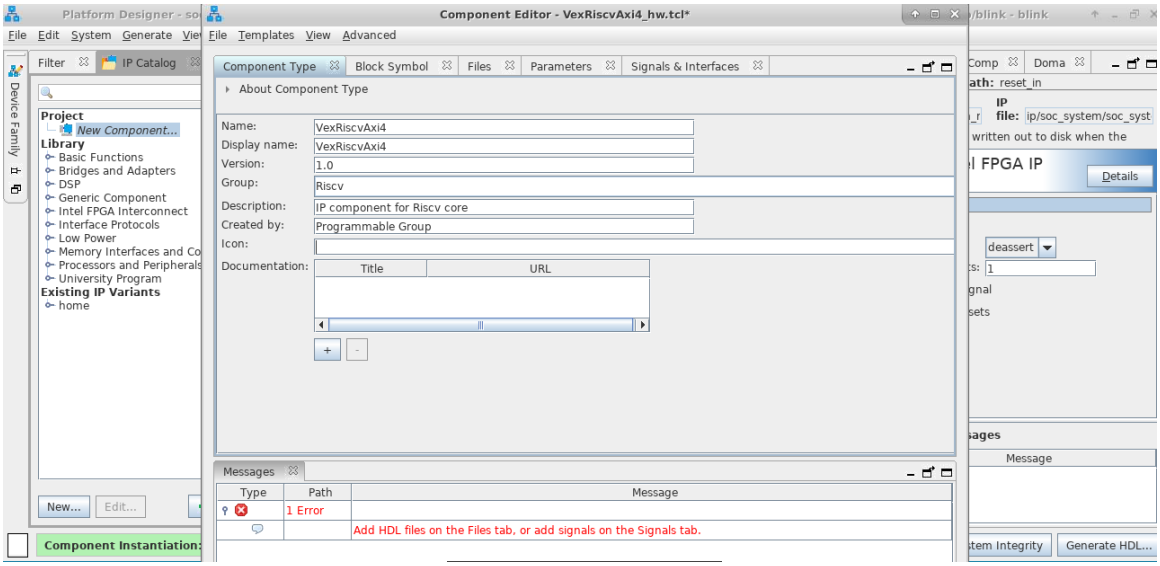

Figure 19: Component Editor

The Platform Designer Component Editor allows you to perform the following tasks:

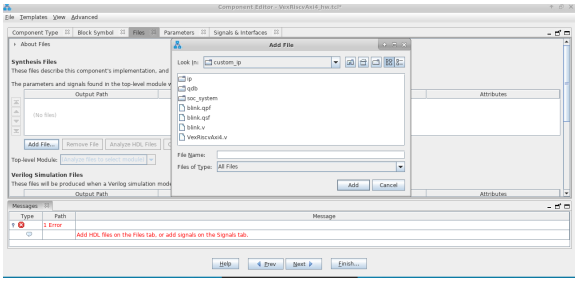

Figure 20: Using HDL Files to Define a Component

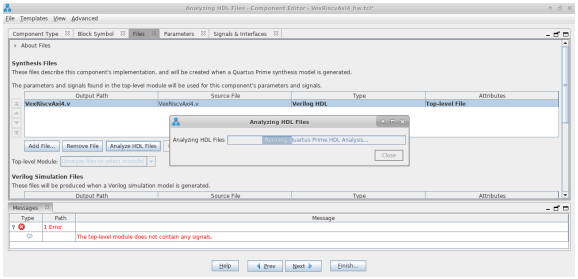

Figure 21: Analyse the HDL file

- Specify component's identifying information, such as name, version, author, etc. details are shown in Fig. 19
- Specify the SystemVerilog, Verilog HDL, VHDL files, and constraint files that define the component for synthesis and simulation.
- Create an HDL template to define a component's interfaces, signals, and parameters.
- Set parameters on interfaces and signals that can alter the component's structure or functionality.

6.1 Files Tab

The Files tab in the Platform Designer Component Editor allows you to specify synthesis and simulation files for custom IP component. If you already have an HDL file that describes the behavior and structure of your component, you can specify those files on the Files tab. If you do not yet have an HDL file, you can specify the signals, interfaces, and parameters of the component in the Component Editor, and then use the Create Synthesis File from Signals option on the Files tab to create the top-level HDL file. The Component Editor generates the _hw.tcl commands to specify the files.
A component uses filesets to specify the different sets of files that you can generate for an instance of the component. The supported fileset types are: QUARTUS_SYNTH, for synthesis and compilation in the Intel Quartus Prime software, SIM_VERILOG, for Verilog HDL simulation, and SIM_VHDL, for VHDL simulation.

Specify HDL Files for Synthesis in the Platform Designer Component Editor

- In the Platform Designer Component Editor, you can add HDL files and other support files with options on the Files tab.

A component must specify an HDL file as the top-level file. The top-level HDL file contains the top-level module. The Synthesis Files list may also include supporting HDL files, such as timing constraints, or other files required to successfully synthesize and compile in the Intel Quartus Prime software. The synthesis files for a component are copied to the generation output directory during Platform Designer system generation. The details of specify HDL files for synthesis and set top-level HDL file in component editor as shown in Fig . 20, 21, 22, 23 and 24

Once analysis is complete and the top-level module is selected, you can view the parameters and signals on the Parameters and Signals & Interfaces tabs. The Component Editor may report errors or warnings at this stage, because the signals and interfaces are not yet fully defined

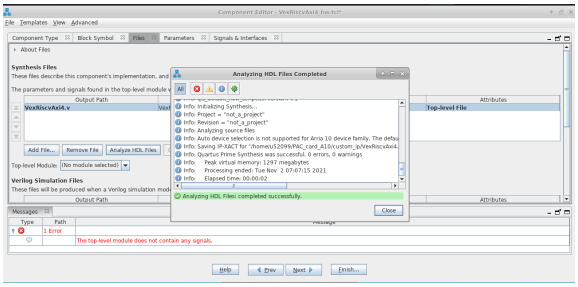

Figure 22: Successfully analyze the HDL file

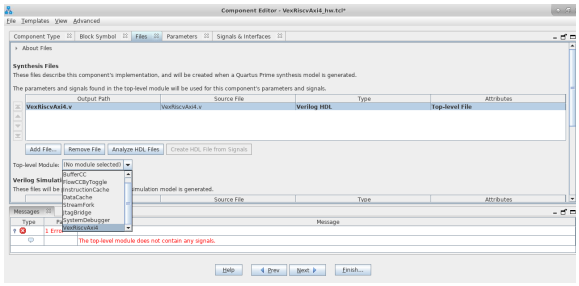

Figure 23: Set the top-module of HDL file for interface

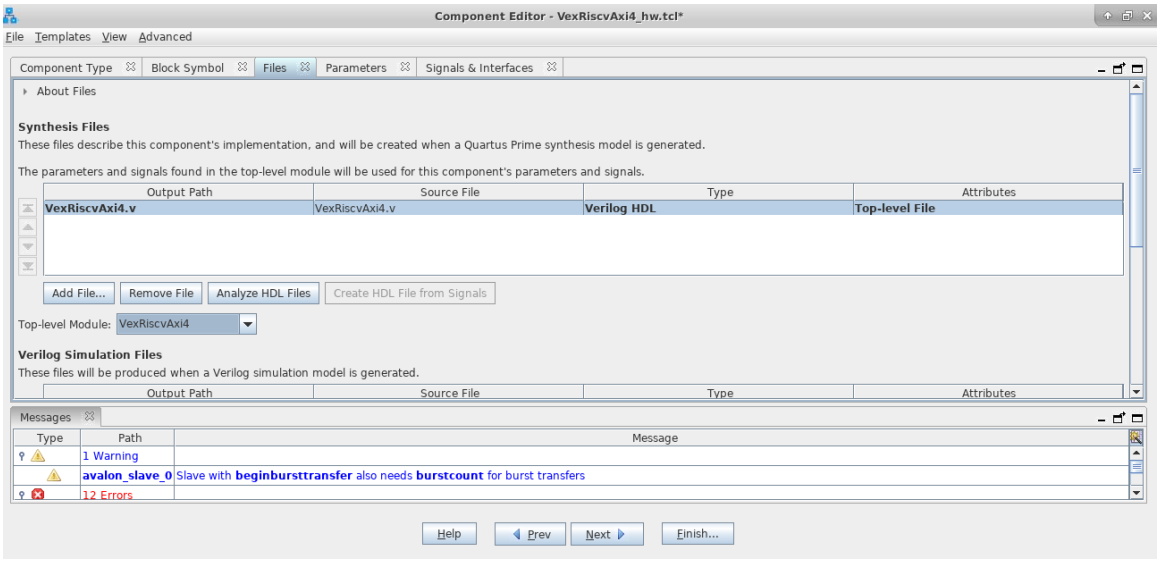

Figure 24: Set the top-module to build custom Interface

6.2 Signal and Interface tab

In the Platform Designer Component Editor, the Signals & Interfaces tab allows you to add signals and interfaces for your custom IP component. As you select interfaces and associated signals, you can customize the parameters. Messages appear as you add interfaces and signals to guide you when customizing the component. In the parameter editor, a block diagram displays for each interface. Some interfaces display waveforms to show the timing of the interface. If you update timing parameters, the waveforms update automatically.

add an interface, click $<<$ *add interface* $>>$ in the left pane. A drop-down list appears where you select the interface type.
To add signals for the selected interface click $<<$ *add signal* $>>$ below the selected interface.

The Interface column allows you assign a signal to an interface. Each signal must belong to an interface and be assigned a legal signal type for that interface. To create a new interface of a specific type, select new $<$ *interfacetype* $>$ from the list; this new interface then become available in the list for subsequent signal assignments. You can highlight all of the signals in an interface and then select an Interface from the list to apply the Interface name to each signal in the interface. You can remove signal that have no assigned to proper signals by clicking Remove as shown in Fig. 25. From, Fig. 26 it can seen that, the conduit interface is added to the IP component. The parameter window appear to modify the interface name, width and set. To add the conduit signal to interface by click

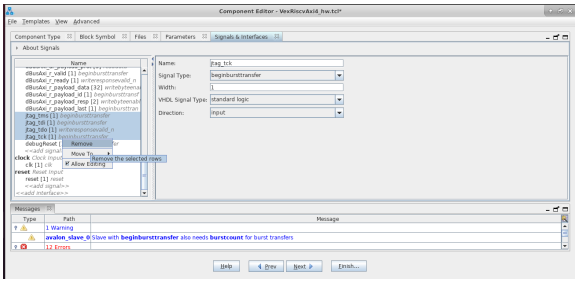

Figure 25: Signal and Interface wizard

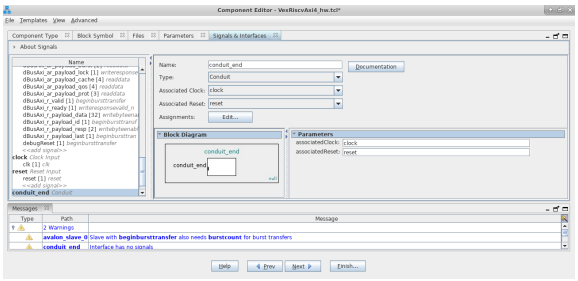

Figure 26: Add Conduit Interface

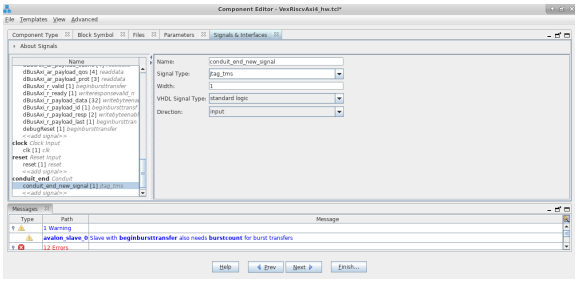

Figure 27: Add Conduit Signal

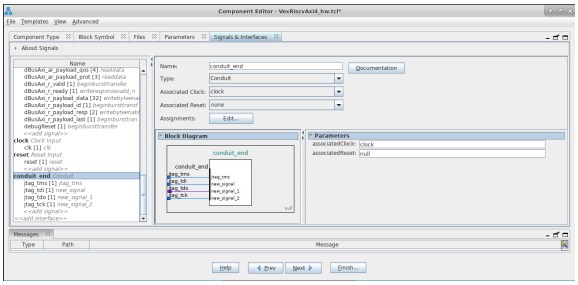

Figure 28: Add Conduit Signal2

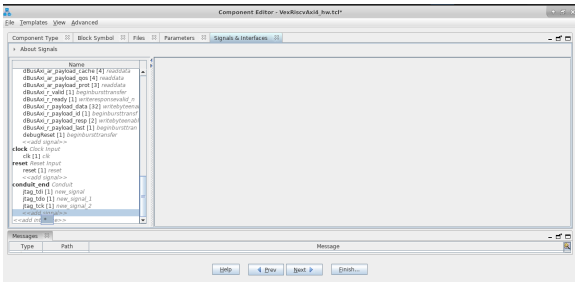

Figure 29: Add Conduit Signal3

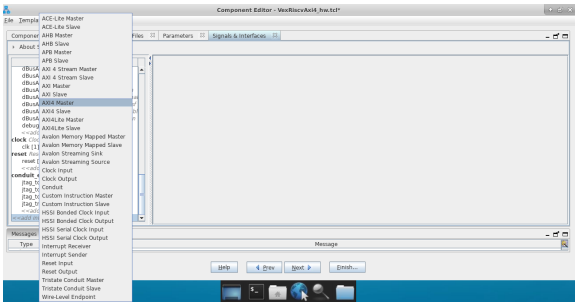

Figure 30: Add AXI4 Master Interface for Custom IP

$<$ *signal add* $>$, it appear below the added Conduit Interface as shown in Fig.27. The VexRisc-v has two AXI master Interface such Instruction master and Data master. Also VexRisc-V has AXI channel such as read data, read address, write data, write address and write response channel,so each channel contains bundle of signals. These AXI Data and Instruction master interface is added to component by renaming the interface to the dBusaxi and iBusaxi interface. Also the signals are added by click on $<$ *signal add* $>$, the complete flow of adding the interface and signals to custom VexRisc-v IP component as shown in Fig.30 - 35.

Once the interface and signals are added into the custom IP module, When you click Finish, Platform Designer creates the component _hw.tcl file with the details that you enter in the Component Editor. When you save the component, it appears in the IP Catalog as shown in Fig. 36 and Fig.37. The created custom IP module appears in the platform designer project directory, to instantiate the custom IP in the design, just click on VexRiscvAxi4, it will instatiated in the design, the details of custom IP, Interface and signals of the components as shown in Fig.37-38

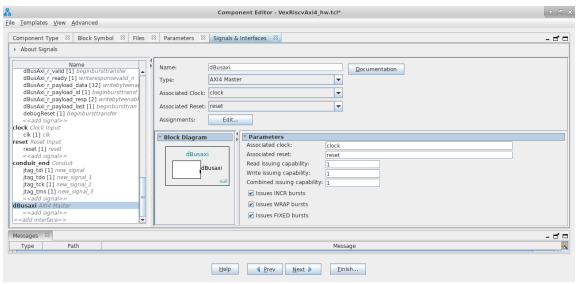

Figure 31: Rename the Interface name by editing the parameter

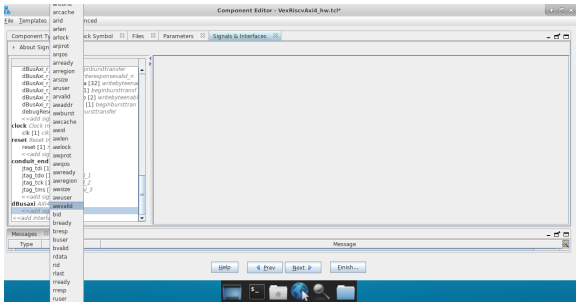

Figure 32: Add AXI4 Master signal for Custom IP

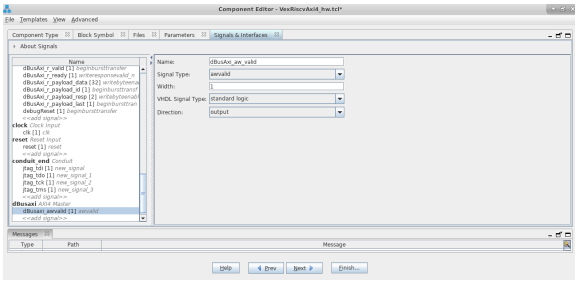

Figure 33: Edit AXI4 signal using edit parameter

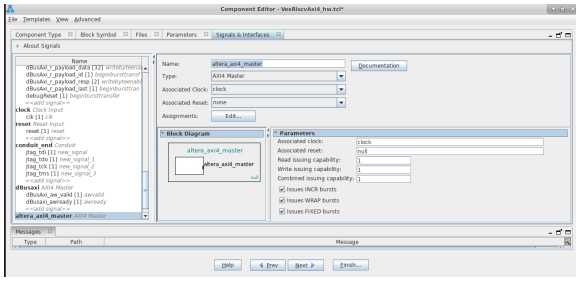

Figure 34: Add another AXI4 Master Interface for Custom IP and Rename the Interface name

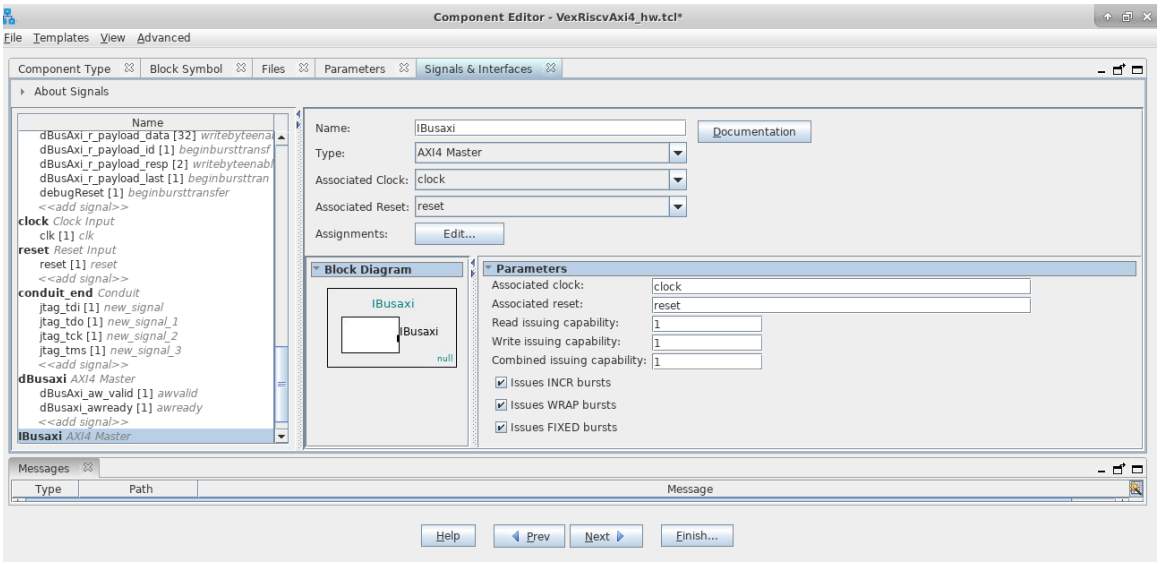

Figure 35: Rename the Interface name reflect as shown in left panel

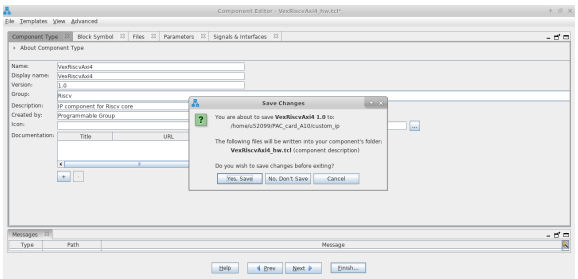

Figure 36: Once all interface and signals are added to component, click finish and save the custom IP

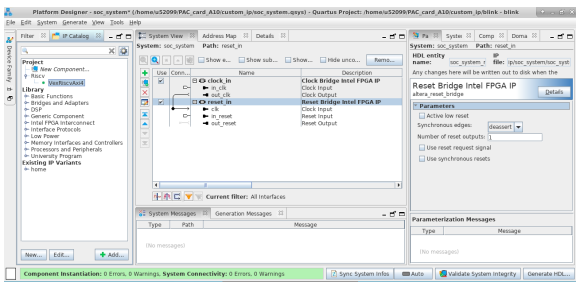

Figure 37: Custom IP appear in Platform designer wizard

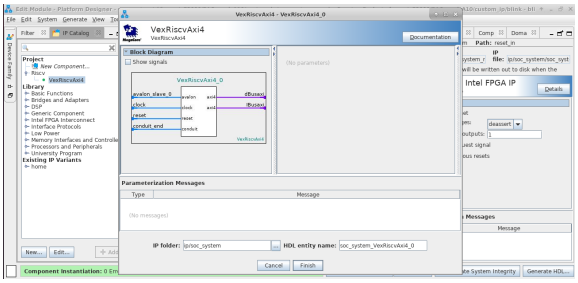

Figure 38: custom IP module with Interfaces

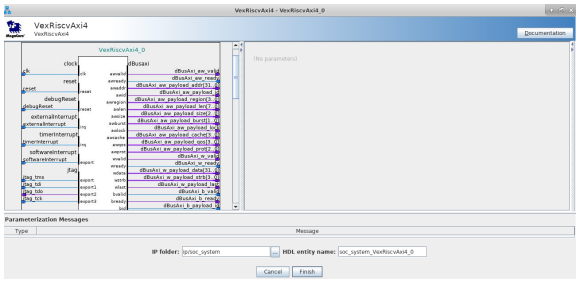

Figure 39: custom IP module with Signals

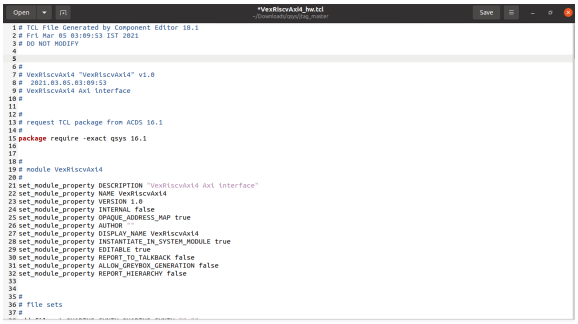

Figure 40: _hw.tcl Created from Entries in the Component Type tab

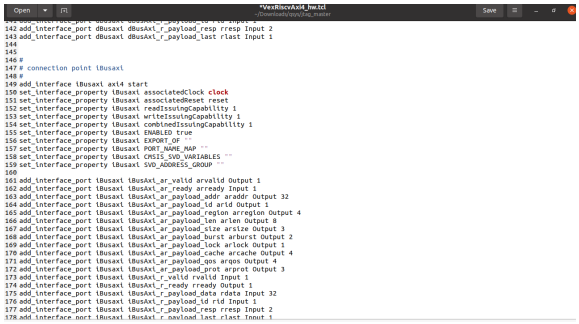

Figure 41: VexRisc-V Instruction and Data master bus with AXI Interface

6.3 _hw.tcl file

Component Files

- $< hdl file >$: Contains the component HDL design files, for example .v, .sv, or .vhd files that contain the top-level module, along with any required constraint files
- $< component\_name > \_hw.tcl >$: The component description file. Each _hw.tcl file defines a single component.

If you require custom features that the Platform Designer Component Editor does not support, for example, an elaboration callback, use the Component Editor to create the _hw.tcl file, and then manually edit the file to complete the component definition.

Edit an IP Component using _hw.tcl file

n Platform Designer, you make changes to a component by right-clicking the component in the System View tab, and then clicking Edit. After making changes, click Finish to save the changes to the _hw.tcl file. You can open an _hw.tcl file in a text editor to view the hardware Tcl for the component. If you edit the _hw.tcl file to customize the component with advanced features, you cannot use the Component Editor to make further changes without over-writing your customized file. The Fig.40, 41 and 42 shows the details of _hw.tcl file. This file can edited manually to add interface and signals. To instantiate the custom IP VexRisc-v core to other project directory just copy the HDL and _hw.tcl file and paste to the project directory, where you want instatiate the custom IP module.

7 VexRISC-V implementation on Intel PAC card

7.1 Getting Started with Accelerator Functional Unit(VexRISC-V) Development

The design flow of VexRISC-V on Intel FPGA PAC card.

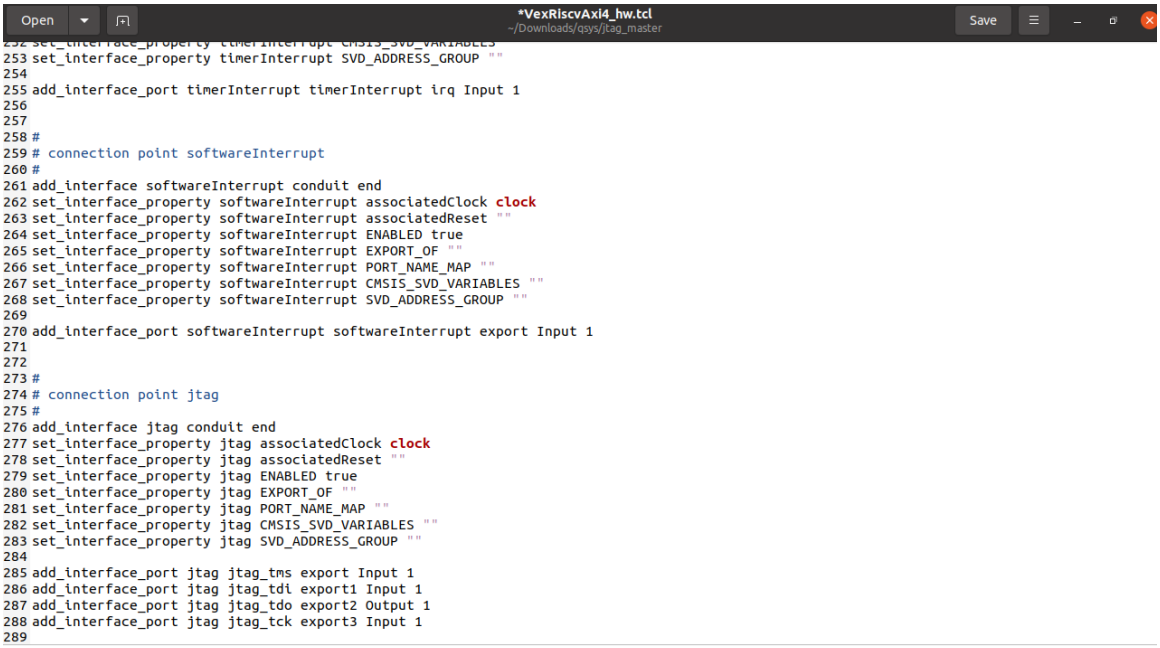

Figure 42: Conduit Interface of Jtag and other vexRisc-v Interface

Design Entry Tools

- Intel Quartus® Prime Pro Edition software.
- Open Programmable Acceleration Engine(OPAE) stack.
- OPAE SDK
- Platform Interface Manager-2.0

Platform Interface Manager(PIM) 2.0

The initial version of PIM supports the CCI-P interface for communication between the Host system and AFUrunning on FPGA. The PIM-2.0 is a significant upgrade over the first version. The AXI, Avalon, and CCI-P host interface options are available through PIM 2.0. Accelerator Functional Units(AFUs)continue to work on the new codebase in compatibility mode.
Update the PIM using the following command
update release.sh

7.2 Design of VexRISC-V on Intel PAC card

We implemented a custom VexRISC-V IP module with AXI interface using the Intel Platform designer tool shown in Fig.43. This custom IP module is used to build an SoC system with VexRISC-V core. The Platform Designer(Qsys) tool is used to build an SoC system with VexRISC-V core as shown in Fig. 44.

The SoC system has following IPs.

- 64Kb Block RAM IP
- JTAG Avalon master bridge IP
- JTAG UART IP

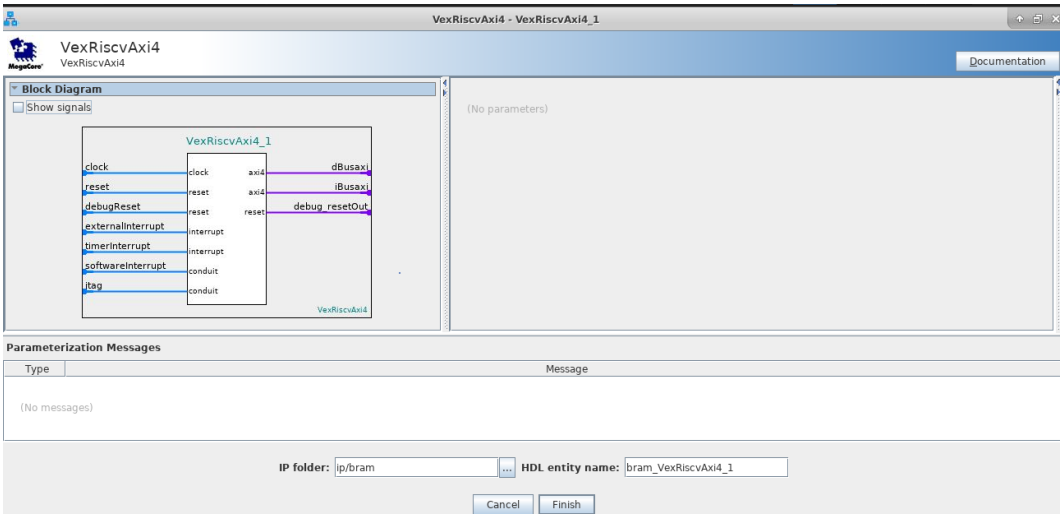

Figure 43: Custom VexRISC-V IP with AXI Interface

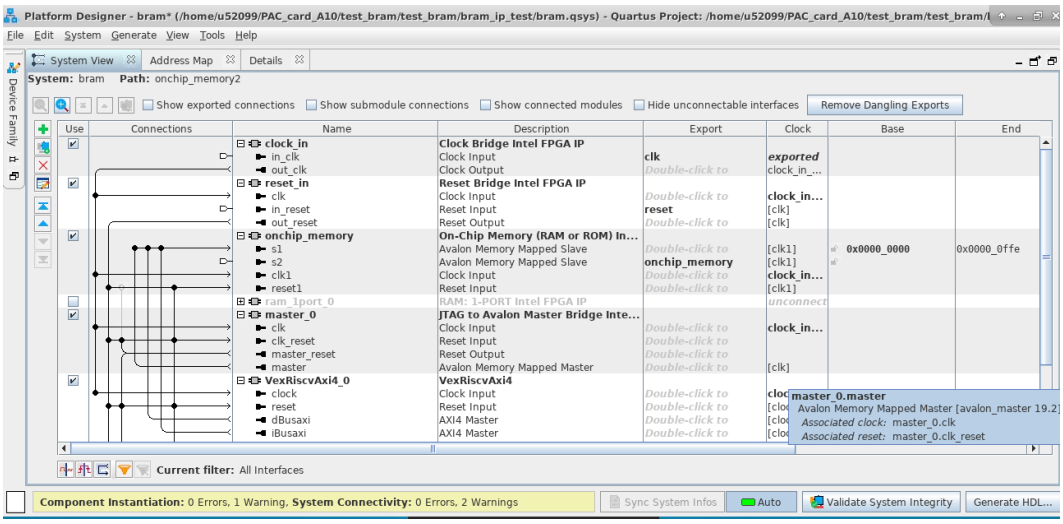

Figure 44: Design of SoC system with VexRISC-V core

- VexRISC-V IP

Here, AFU is the custom implementation of the SoC system with VexRISC-V core. AFU can be accelerated on the OPAE hardware platform.

An AFU has two main communication paths between the host:

- FPGA to host
- Host to FPGA (MMIO)

FPGA to host

The FPGA accesses host memory using a 512 bit data path. This data path has separate channels for read and write traffic allowing for simultaneous read and write to occur. The read and write channels support bursts of 1, 2, and 4 cache lines.

Host to FPGA (MMIO)

The host can access a 256 KB address space within the FPGA. This address space contains Device Feature Header (DFHs) and the control and status registers of the AFU hardware. DFHs are small

ROMs that hold metadata about the hardware that are enumerated by the OPAE SDK.

Further, the MMIO address space has been designed for communication between the Host and SoC system(AFU) with VexRISC-V core on the Intel PAC card.

A SoC system with VexRISC-V(AFU) design includes the following components

- RTL description of the SoC system with VexRISC-V(AFU) to accelerate on Intel FPGA PAC card. Fig. 45 shows all modules are integrated to build AFU.

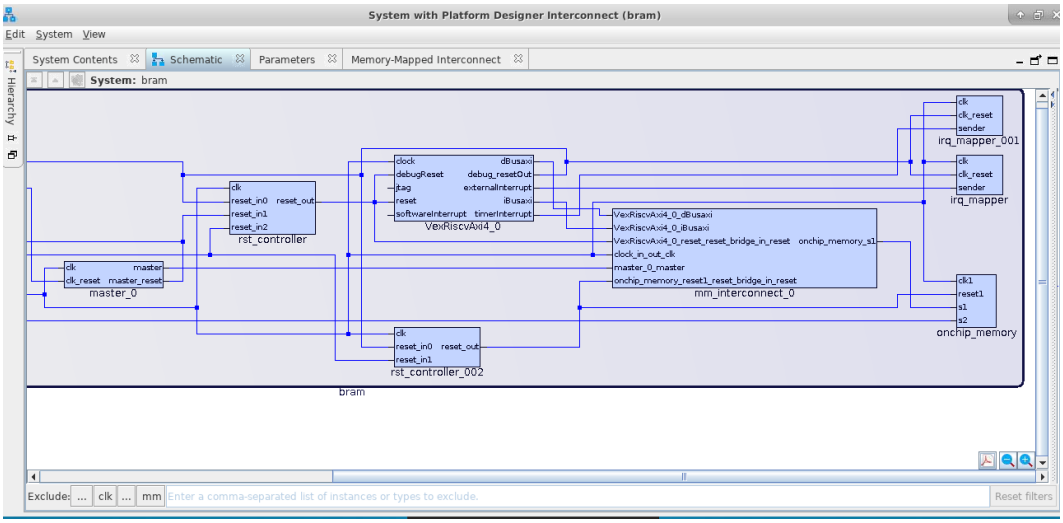

Figure 45: Design of AFU

- RTL description to implement the base requirements placed on AFUs by OPAE (for example DFH, AFU ID in MMIO space) as shown in Fig. 46.

```
 GNU nano 2.9.3                                  afu.sv


always_ff @(posedge clk)
 //always_comb
  begin
        mmio64_if.readdata <= '0;
        mmio64_if.readdatavalid <= mmio64_if.read && ! mmio64_if.waitrequest;
     case (mmio_address)
  //        case (mmio_address)
         16'h0000: // AFU DFH (device feature header)
                begin
                    // Here we define a trivial feature list.  In this
                    // example, our AFU is the only entry in this list.
                   mmio64_if.readdata  <= 64'b0;
                    // Feature type is AFU
                    mmio64_if.readdata[63:60] <= 4'h1;
                    // End of list (last entry in list)
                     mmio64_if.readdata[40] <= 1'b1;
                end
            // AFU_ID_L
            16'h0001: mmio64_if.readdata <= afu_id[63:0];
            // AFU_ID_H
            16'h0002: mmio64_if.readdata <= afu_id[127:64];

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos  M-U Undo  M-A Mark Text
^X Exit      ^R Read File  ^\ Replace   ^U Uncut Text ^T To Spell  ^_ Go To Line M-E Redo  M-6 Copy Text
```

Figure 46: Mandatory AFU CSRs RTL code(DFH, AFU ID)

- Fig. 47 shows the logic to map AFU CSRs into MMIO space.
- Memory mastering logic- The RTL code for local FPGA memory as shown in Fig. 48. Both Host and VexRISC-V access this local memory to read and write the data.
- Debug and monitoring using Signal Tap with the Remote Debug feature. Fig.49 shows the remote debugging of VexRISC-V AFU using signal tap.

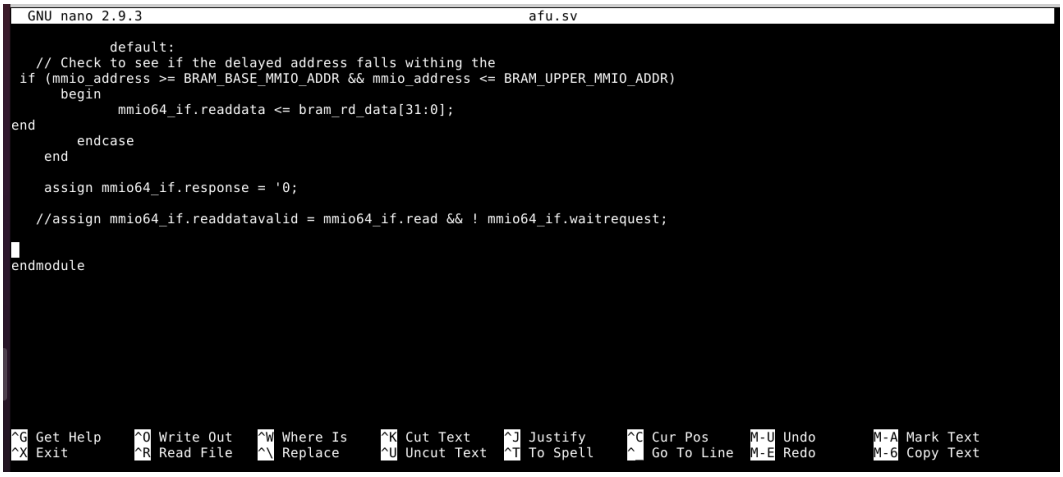

Figure 47: Logic to map AFU CSRs into MMIO space

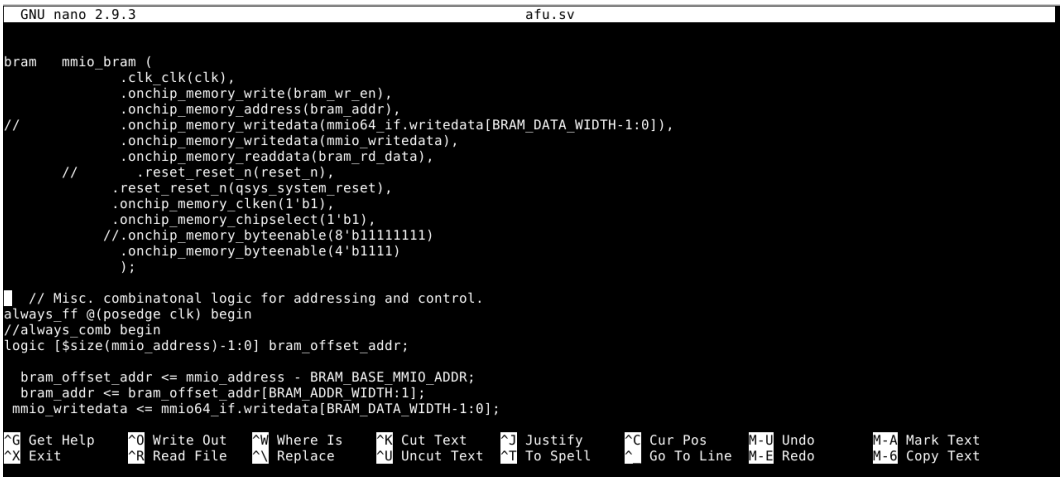

Figure 48: logic for Local memory access

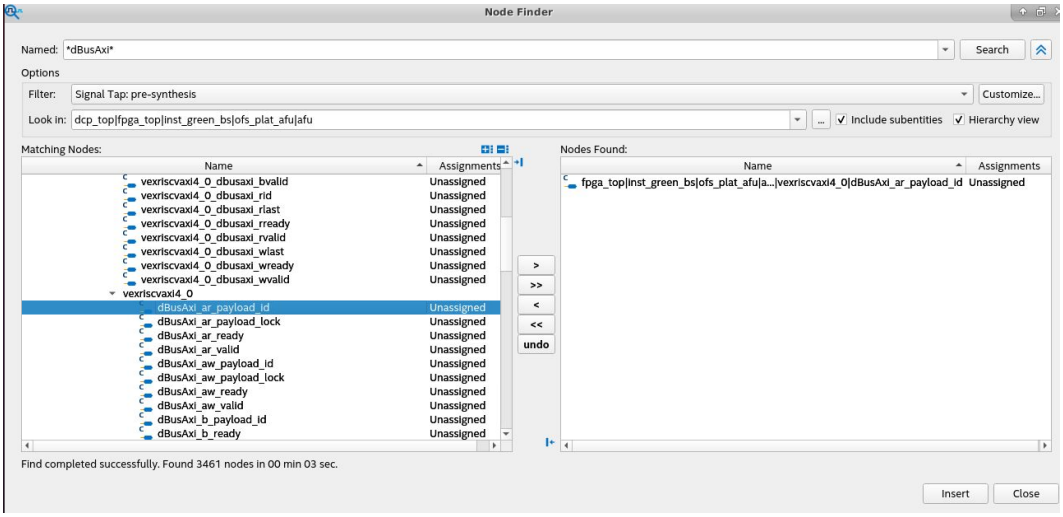

Figure 49: Signal Tap debug

7.3 Build and compile the SoC system with VexRISC-V(AFU)

Development Environment References

- The OPAE_PLATFORM_ROOT environment variable points to the OPAE SDK installation.
- The QUARTUS_BIN environment variable points to the Intel Quartus installation.

Build and compile the design using the following command

- afu_synth_setup –source filelist.txt vexriscv_test.
  The implemented SoC system with VexRISC-V is build using the command as shown in Fig.50

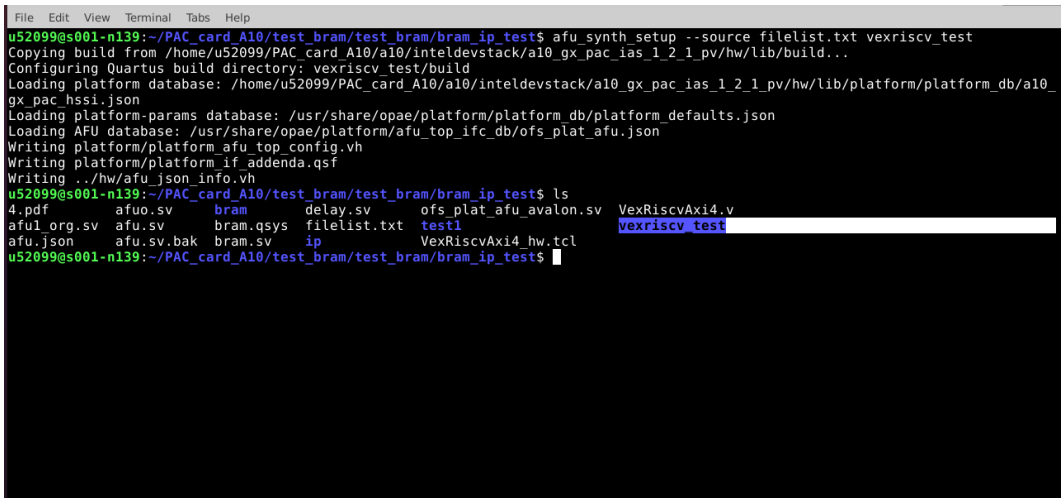

Figure 50: command to build the AFU design

- cd vexriscv_test
- run.sh
  The run.sh command performs the synthesis, design fitter such as place, map, and route, finally the timing analysis of design. Fig. 51 shows the synthesis and fitter of design on the FPGA platform.

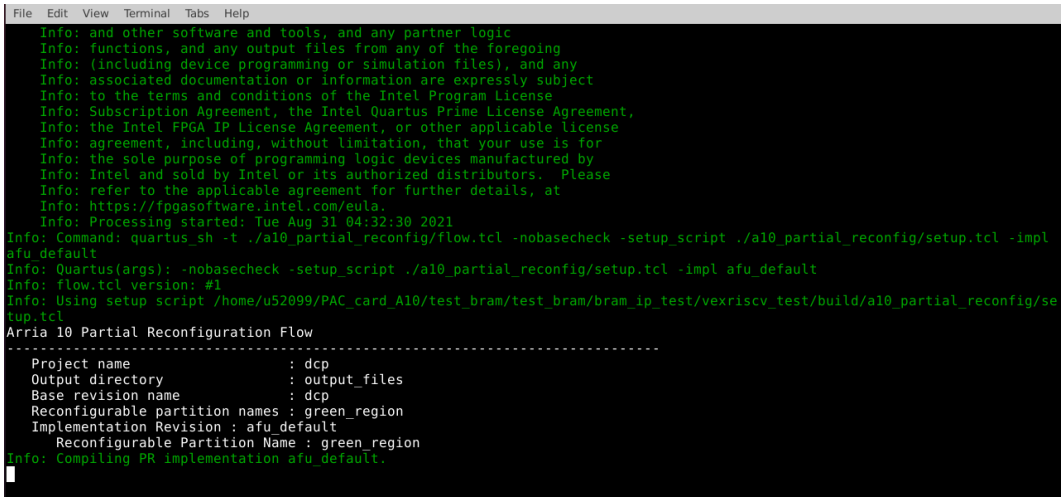

Figure 51: Command to run the AFU design

Authentication of .gbs file to program

The PACSign utility inserts authentication markers into .gbs file targeted for the Intel programmable acceleration cards (PACs) as shown in Fig.52. The PACs will not program .gbs file without proper authentication.

- PACSign SR -t UPDATE -H openssl_manager -i afu.gbs -o unsigned_afu.gbs
  No root key specified. Generate unsigned bitstream? Y = yes, N = no: y
  No CSK specified. Generate unsigned bitstream? Y = yes, N = no: y

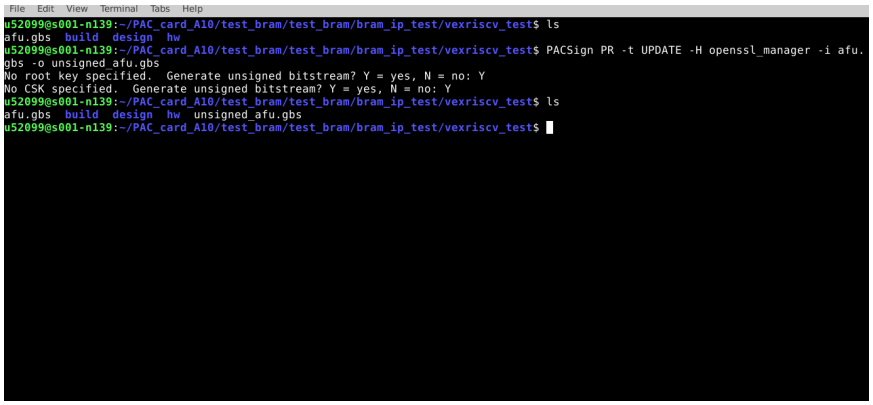

Figure 52: Authentication of AFU

Program Intel PAC card

fpgaconf command configures the FPGA with the accelerator function unit (AFU). Fig.53 shows the programming AFU on the Intel PAC card.

- fpgaconf -B 0x3b unsigned_afu.gbs

Figure 53: Program the AFU on Intel PAC card

7.4 Capturing Signals in SoC system with VexRISC-V(AFU) with signal tap remote debug

The AFU design is debug using a signal tap logic analyzer. The remote debug features of the Intel PAC card is used for debugging the AFU. Here, the Signal Tap GUI running locally on the server with the intel PAC card.

Set up connections for remote debug

- Set up a network connection between the instrumented FPGA AFU and Signal Tap GUI. We can set the network connection based on whether we are using a remote or local debugging configuration.

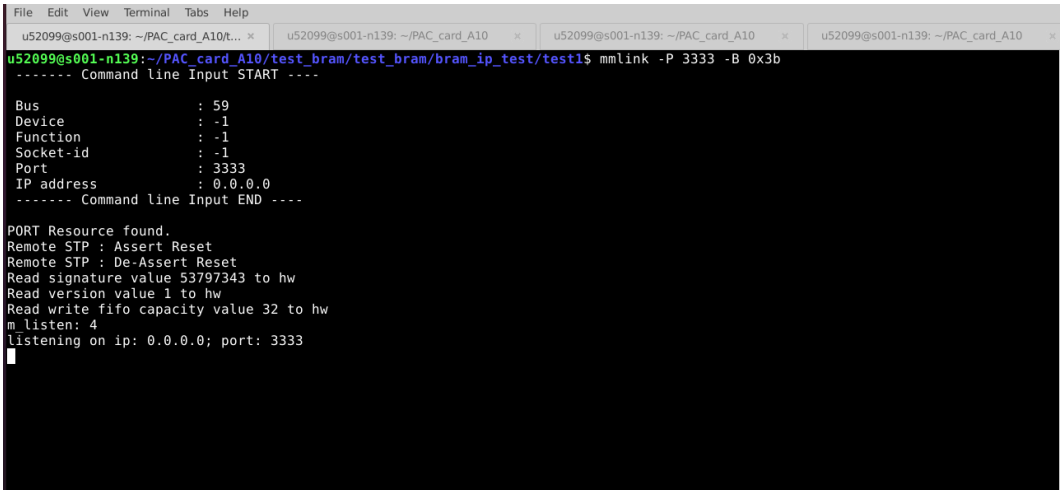

Figure 54: Command for mmlink setup

- Use the OPAE tool mmlink to enable the host system for remote Signal Tap as shown in Fig. 54.

- Open a TCP port to accept incoming connection requests from remote debug hosts.
  - Start System Console for remote debug as shown in Fig.55

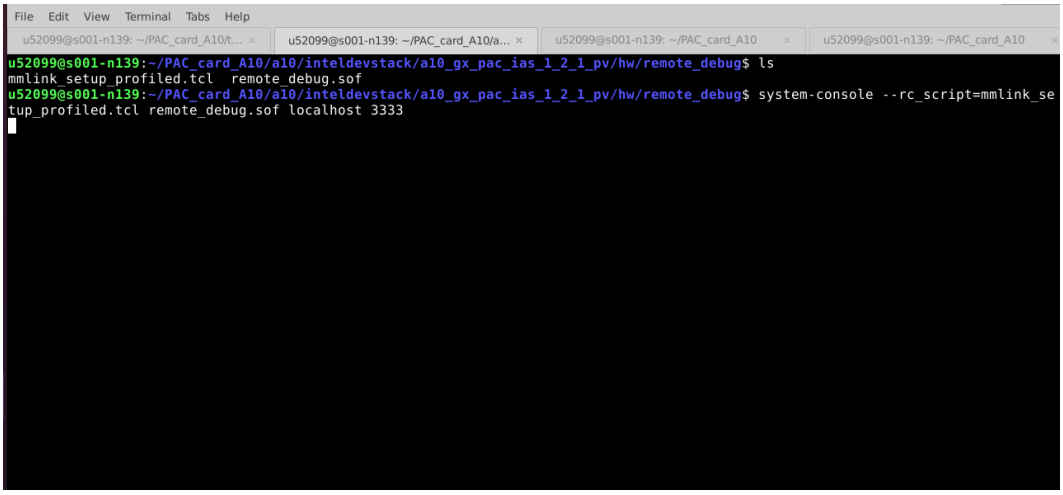

Figure 55: System console

  - Open Intel Quartus Prime GUI on the machine that performs the debug as shown in Fig. 56 and 57.

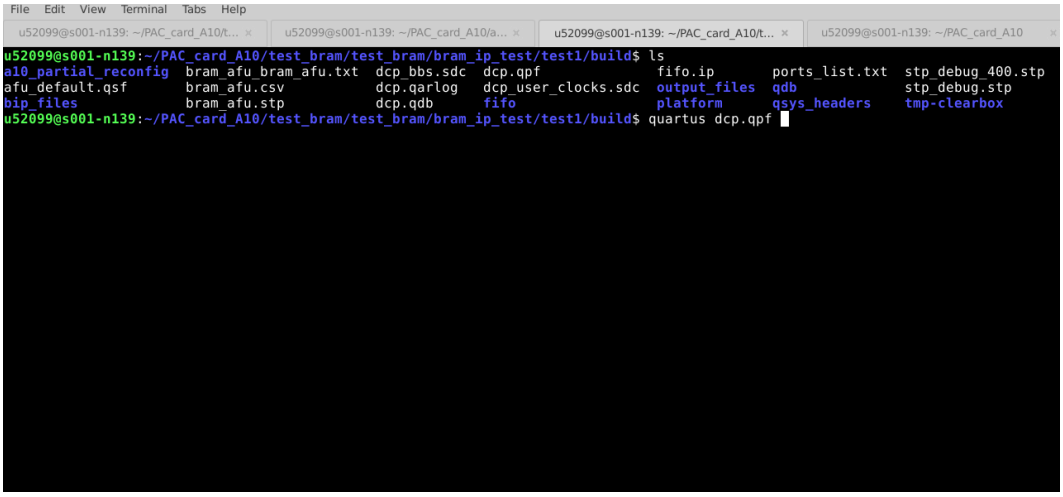

Figure 56: Open AFU design using Quartus software

  - Open Signal Tap GUI. Fig. 58 and 59 show the VexRISC-V core is debug using signal tap. Here, we monitor the VexRISC-V 32-bit register content. When the assembly code is loaded into BRAM, then VexRISC-V starts execution by reading the instruction fromBRAM address space. Once execution is finished, the respective register content is updated.

7.5 Test the VexRISC-V AFU using software code

To test the working of VexRISC-V AFU on Intel FPGA PAC card, the simple load and store assembly program is written as shown below.

```
addi x2 , x0 , 5
sw x2 , 48(x0)
lw x3 , 48(x0)
```

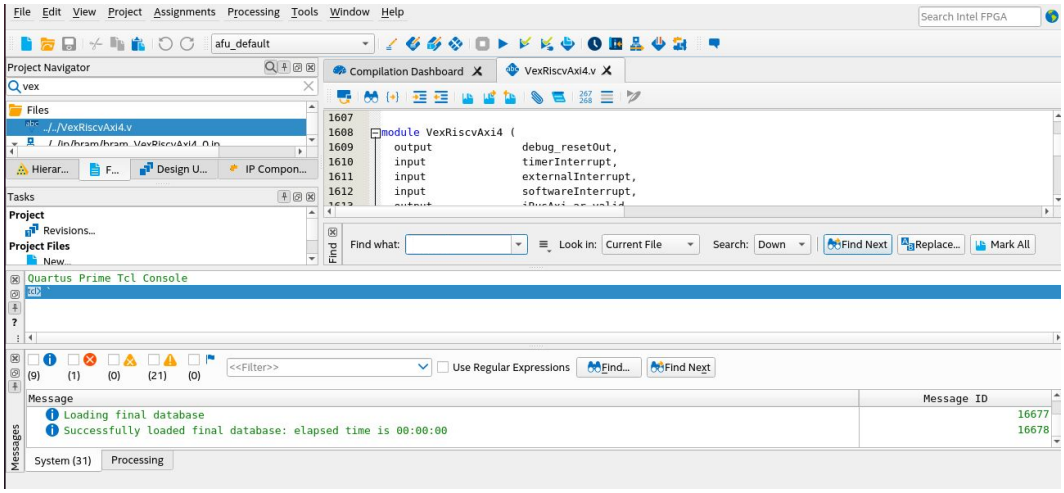

Figure 57: Quartus software

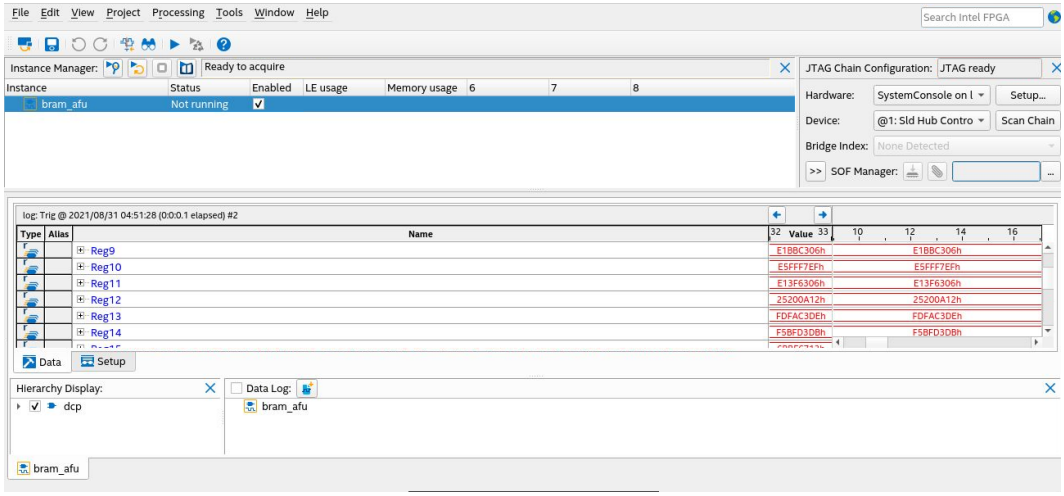

Figure 58: Signal Tap GUI

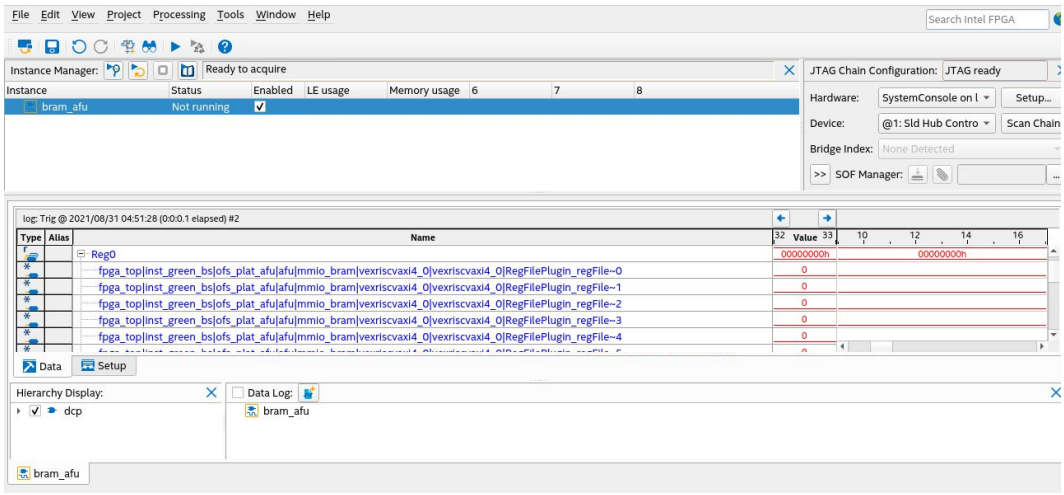

Figure 59: Debug VexRISC-V using signal Tap

Here, addi instruction adds the immediate value 5 and content of register x0 (x0 value is zero), then the result is stored in the x2 register. The instruction SW, store the content of the x2 register into

the memory location 48(x0). Last instruction lw, read the content from memory location 48(x0) and load value into x3 register.

The assembly code is compiled and run using the RISC-V toolchain. The .elf file is generated after compilation and run. To know the machine instruction of assembly code by disassembling .elf file as shown in Table .1.

Table 1: Machine code

PC	Machine Code	Basic Code	Original Code
0x20	0x00500113	addi x2 x0 5	addi x2 x0 5
0x24	0x02202823	sw x2 48(x0)	sw x2 48(x0)
0x28	0x03002183	lw x3 , 48(x0)	lw x3 , 48(x0)

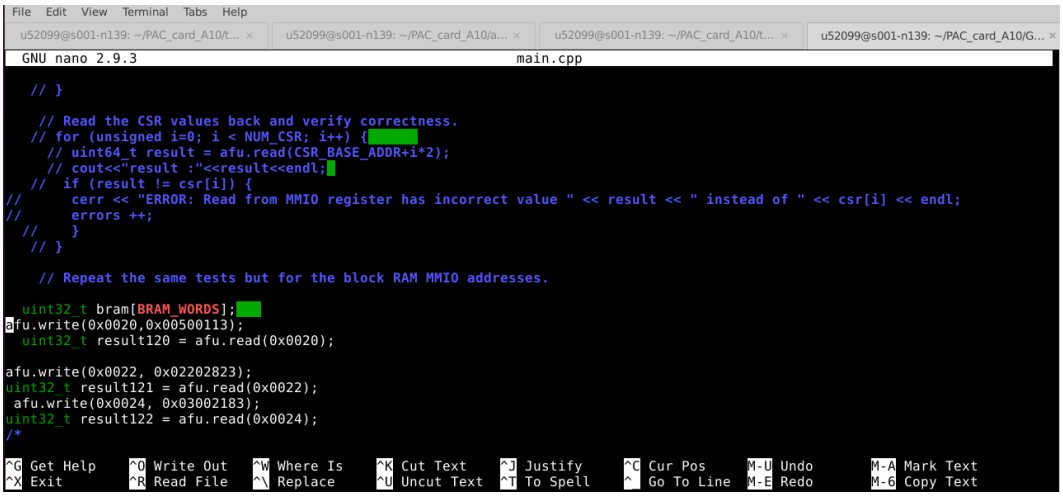

Figure 60: Load the Machine code using OPAE APIs

The machine codes are load into the FPGA block RAM space using the OPAE APIs such as fpgaWriteMMIO64, fpgaReadMMIO64 as shown in Fig. 60 and 61. These APIs feature functions for mapping and accessing control registers through memory-mapped IO. Most FPGA accelerators provide access to control registers through memory-mappable address spaces, commonly referred to as “MMIO spaces”.

Write a 64-bit value to MMIO space using the below function.
fpga_result fpgaWriteMMIO64(fpga_handle handle, uint32_t mmio_num, uint64_t offset, uint64_t value)
Read 64-bit value from MMIO space using the below function.
fpga_result fpgaWriteMMIO32(fpga_handle handle, uint32_t mmio_num, uint64_t offset, uint32_t value)

Fig.62 shows the compilation and execute the OPAE software program. The machine code is loaded into the Block RAM, then VexRISC-V starts execution by reading the instruction from Block RAM. Fig. 63 shows the register content of VexRSIC-V is updated after completing the task. Here,x2 and X3 are the Reg2 and Reg3 in signal tap, we can see the Reg2 and Reg3 are updated with value 5.

We analyze the Block RAM content. Fig.64 shows the Block RAM content is updated with value 5 in the memory address location 48.

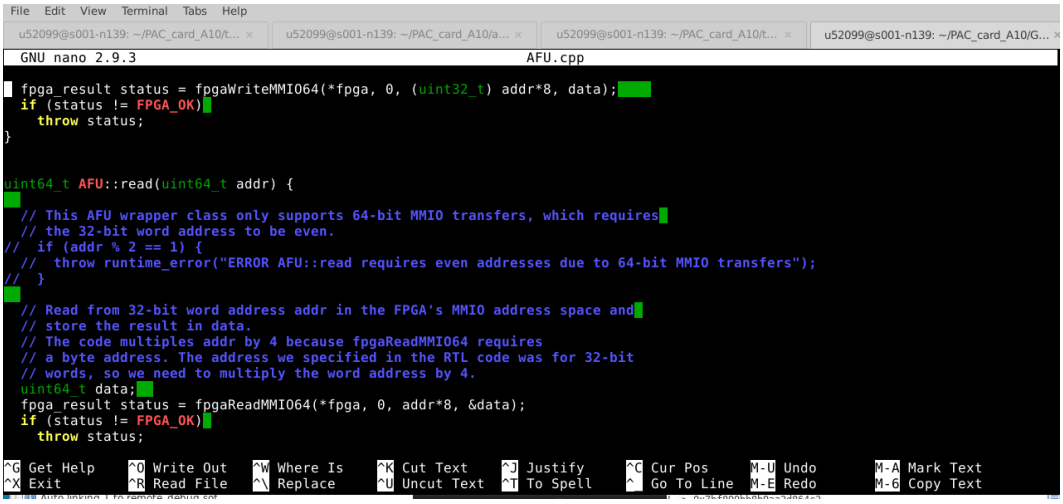

Figure 61: Functions for MMIO space read and write

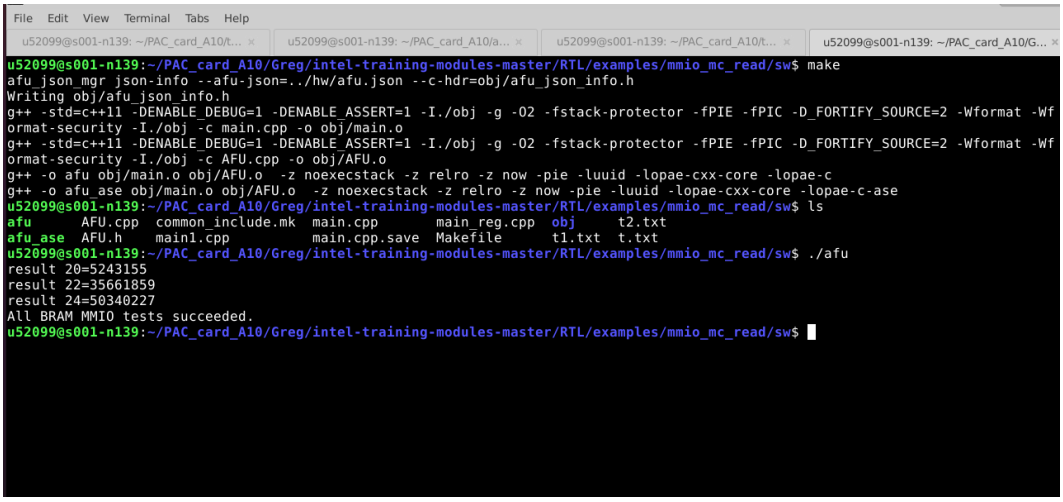

Figure 62: Compile and run the OPAE code

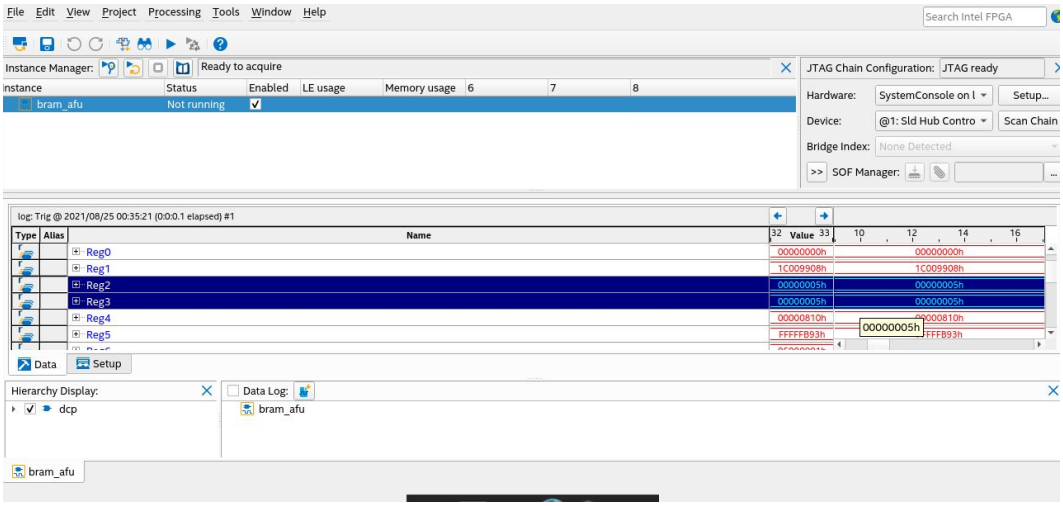

Figure 63: Analyze the register content of VexRISC-V using signa tap

Figure 64: Block RAM memory analysis

8 Source codes generated for the project(Github link) with explanation

https://github.com/khyamling

This repository contains the implementation of VexRISC-V AFU on the Intel FPGA PAC card. We optimize the VexRisc-V processor on Intel FPGA PAC card using optimization strategies and HyperFlex optimizations techniques.
The VexRisc-V has the following features.

- Support the 32-bit RISC-V ISA(RV32IM).
- 5-stage pipeline
- Optimized for Intel FPGA.
- The Instruction cache, Data cache.
- Supports single cycle barrel shifter, debug module, catch exceptions, dynamic branch, memory management unit(MMU).

The AFU with VexRISC-V design, the toplevel and other modules source code is found in ofs_plat_afu_avalon.sv file, the afu.qsf is the Quartus setting file. The synthesis, place, route, and static timing analysis report found in the output_files/ directory. The synthesis result was obtained by synthesizing the VexRisc-V on Intel Quartus Pro 2020.3 software tool with the fastest speed grade and Hyperflex Optimization techniques to get the maximum