A Guide to Debug Design via OCLA using Instantiation Method

Table of Content

Table of Content 2

Introduction 3

OCLA Debug Sub system 3

  How Debug System works? 3

DUT 4

Creating a project with Raptor New Project Wizard 4

Generating OCLA IP 8

IP files addition to Project 12

Instantiating OCLA IP in Design 15

Running Design using Raptor 16

Communicating with OCLA through CLI Commands 17

Debug System Limitations 19

Appendix A 19

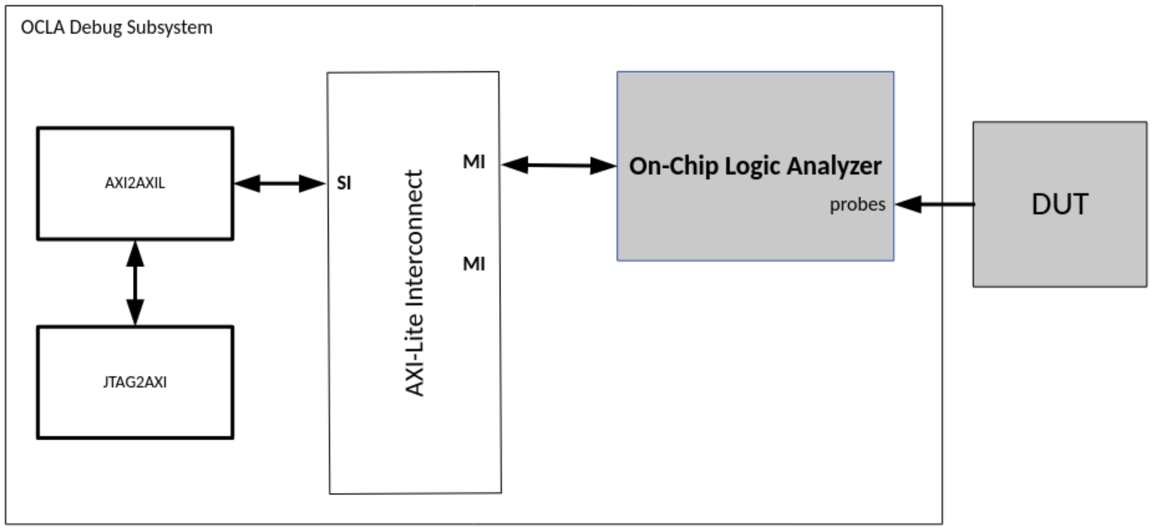
Introduction

OCLA stands for On Chip Logic Analyzer and is an IP developed by Rapid silicon, to provide user a way to probe the internal signals and debug their designs. The utilization of OCLA involves distinct steps, including the generation of the OCLA IP, instantiation within the design, and the probing of internal signals. In this guide, we will explain the process of configuring the On-chip Logic Analyzer (OCLA) IP and demonstrate how this IP can be used to streamline the debugging process.

This document contains a tutorial designed to help you debug complex FPGA designs starting from the start of project, guiding you through the integration of the OCLA IP core, and concluding with the generation of a bitstream using the Raptor toolchain.

OCLA Debug Sub system

The OCLA debug subsystem is composed of several key components, including OCLA itself, Interconnect, AXI2AXIL, and JTAG2AXI IPs. These IPs can all be conveniently generated from the IP catalog available within Raptor. Below figure shows the OCLA Debug sub system block diagram.



How Debug System works?

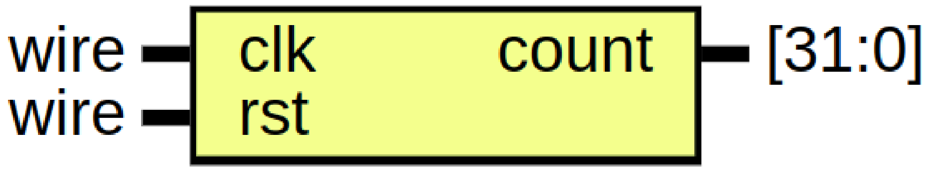
Once the signals from the design are probed, the OCLA IP capture samples using sampling clock and stores these samples in memory. For storage the OCLA uses BRAM of FPGA, so the user should be aware that FPGA is consuming BRAMs. The more samples you want to store the more BRAMs you will consume.

After these samples are stored in OCLA memory (BRAM), then the JTAG2AXI IP issues an AXI based transaction to read the memory of OCLA through AXI Interconnect. The purpose of interconnect is to handle multiple OCLA IP if exist. Once the samples are read from the OCLA memory, these samples are then dumped to vcd and displayed using GTK Wave.

In this tutorial, our primary focus will be on debugging a Design Under Test (DUT) using the OCLA Debug subsystem. For this purpose, we will employ a counter as our DUT within our design. This tutorial will guide you through the steps to effectively debug your design using the OCLA Debug subsystem.

DUT

The "DUT," which stands for "Design Under Test," represents the component we aim to debug in this tutorial. In our case, the DUT is a 32-bit counter. By using the OCLA debug system, our objective is to assess the functionality of this counter and determine whether it operates as expected. Below figure shows the top level of DUT.



The source code for the DUT can be found in Appendix A.

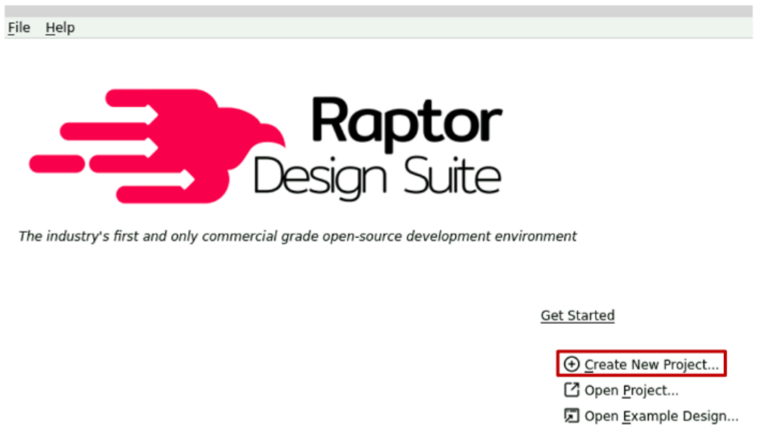
To get started with this tutorial, the first step is raptor project. If you have an existing raptor project, then just open the project by selecting **Open Project.** In that case you don't need to follow the Project Creation section and jump to IP Generation section.

In our case we don't have an existing project, so first we will create a raptor project and then we will proceed to IP Generation section.

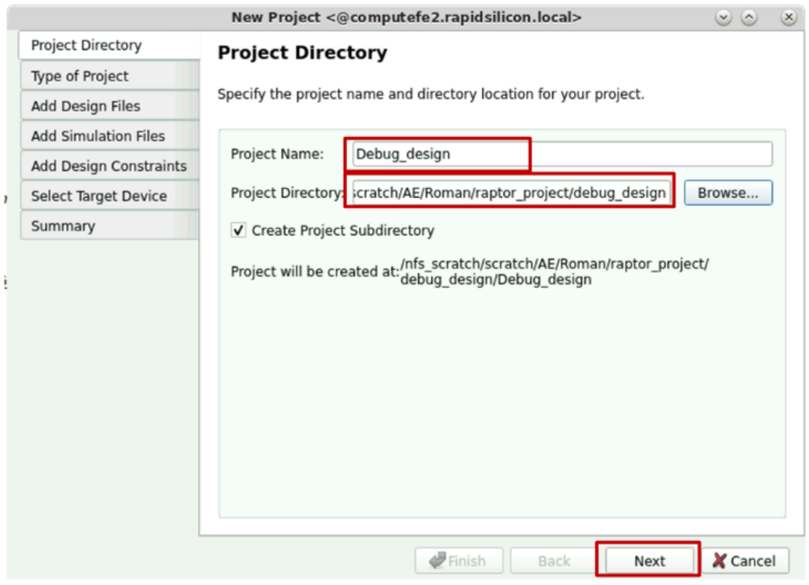
Creating a project with Raptor New Project Wizard

To begin the project creation process, open Raptor by invoking it from the terminal. Once the Raptor graphical user interface (GUI) is launched, proceed by following the instructions outlined below to create your project.

1. Select "Create New project"



2. Type “Project Name” and set “Project Directory”
3. Select “Next”



Project Directory

Specify the project name and directory location for your project.

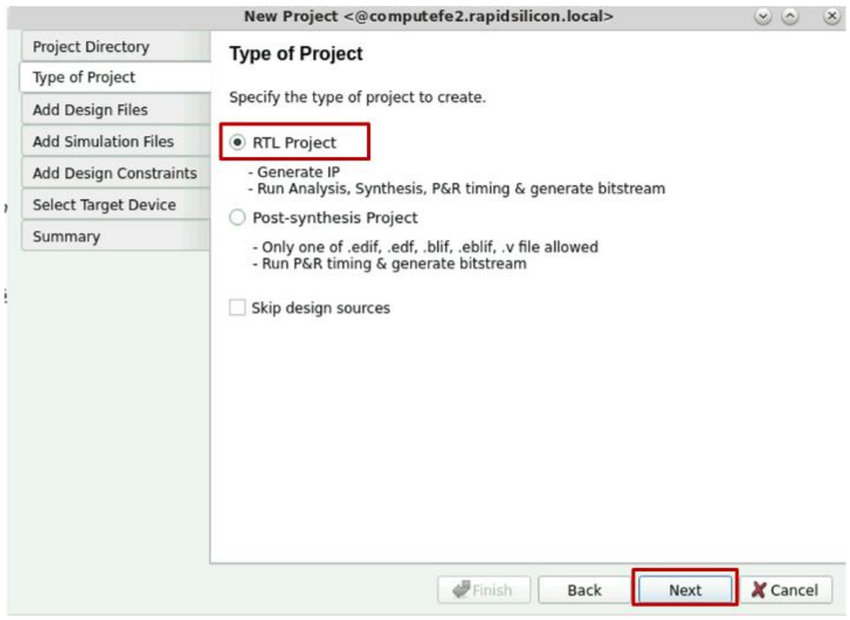
Project Name: Debug_design

Project Directory: ;cratch/AE/Roman/raptor_project/debug_design

✓ Create Project Subdirectory

Project will be created at: /nfs_scratch/scratch/AE/Roman/raptor_project/debug_design/Debug_design

4. Choose the project type as "RTL Project" and click "Next" to proceed.


New Project <@computefe2.rapidsilicon.local>

Type of Project

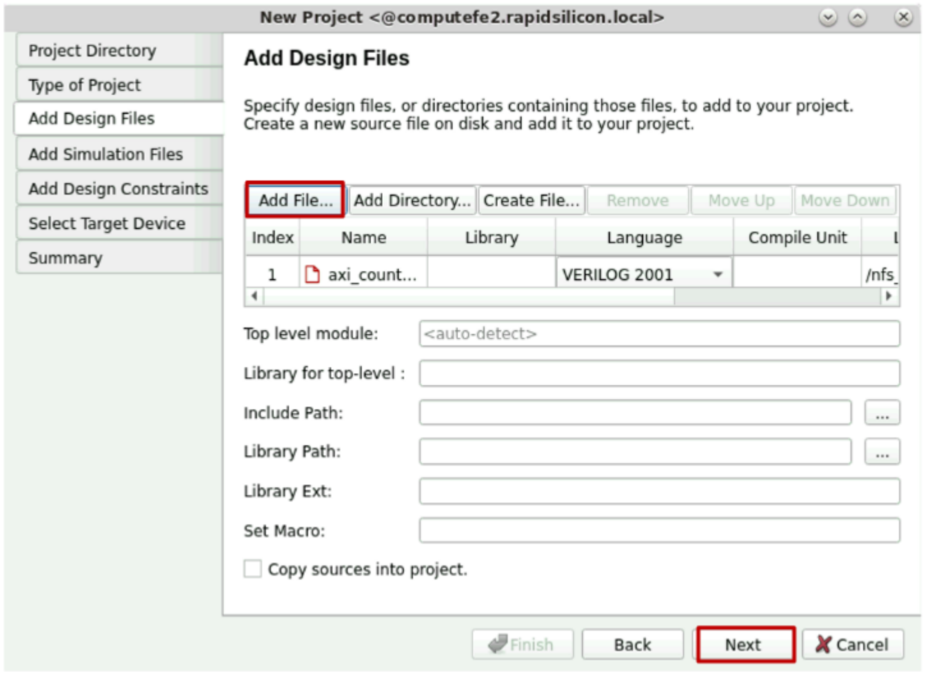
Specify the type of project to create.

[✓ RTL Project]
- Generate IP
- Run Analysis, Synthesis, P&R timing & generate bitstream

Post-synthesis Project
- Only one of .edif, .edf, .blif, .eblif, .v file allowed
- Run P&R timing & generate bitstream

Skip design sources

5. To add your design files, simply click on "Add File." If you don't have any existing files, you can create a new one by selecting "Create File." For the purposes of this tutorial, we will be using a counter as our Design Under Test (DUT). Ensure that you've added the counter design file, then proceed by selecting "Next." To access the code for the counter, please refer to Appendix A.


New Project <@computefe2.rapidsilicon.local>

Add Design Files

Specify design files, or directories containing those files, to add to your project.
Create a new source file on disk and add it to your project.

Index	Name	Library	Language	Compile Unit	L
1	axi_count...		VERILOG 2001		/nfs_

Top level module: <auto-detect>

Library for top-level :

Include Path:

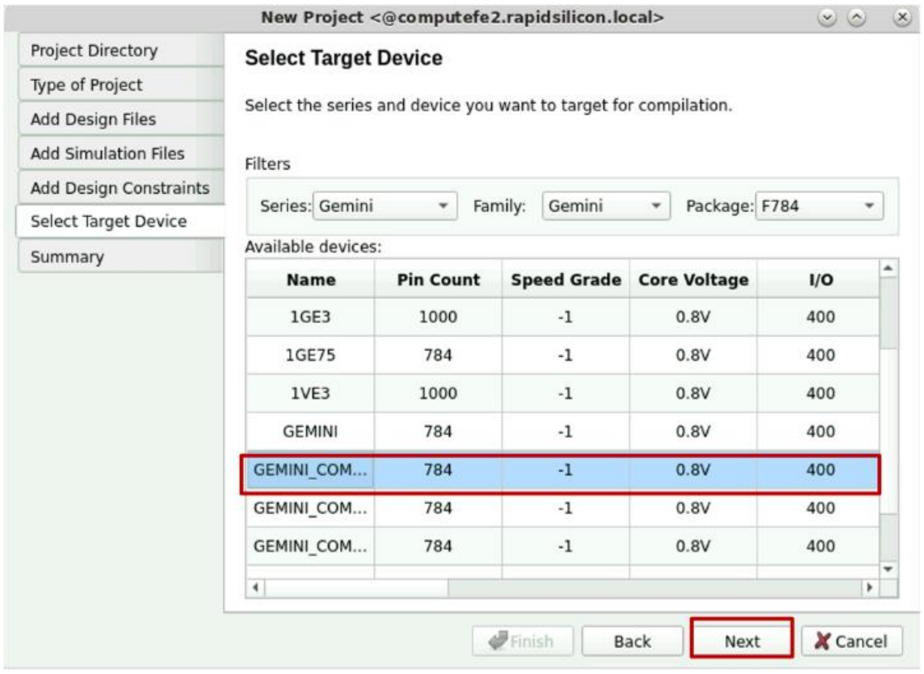
Library Path:

Library Ext:

Set Macro:

Copy sources into project.

6. Select your "Target Device" from the list and click "Next" to continue.



Select Target Device

Select the series and device you want to target for compilation.

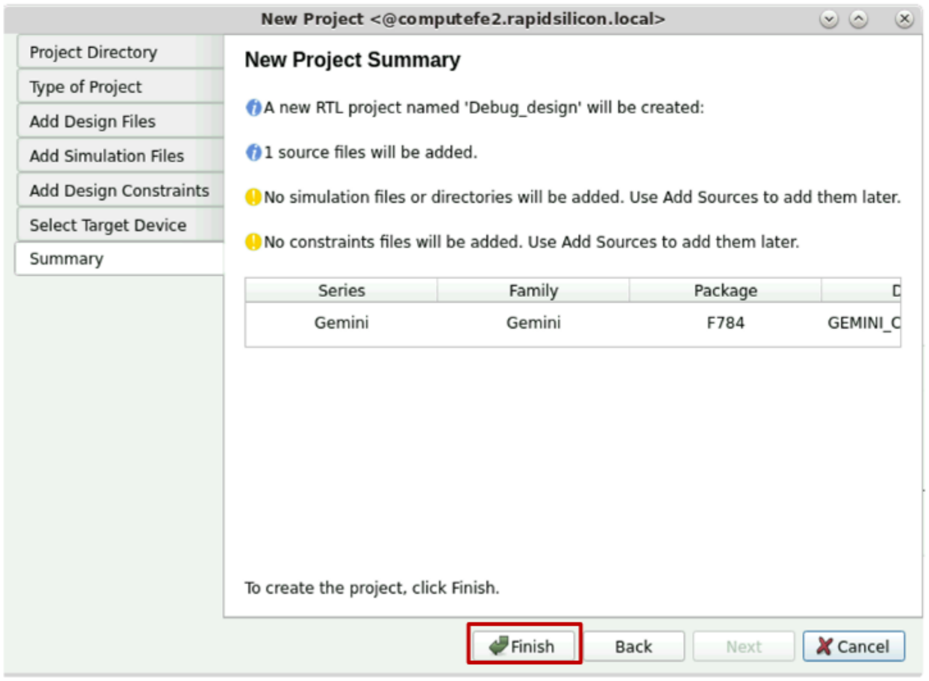
Filters

Series: Gemini Family: Gemini Package: F784

Available devices:

Name	Pin Count	Speed Grade	Core Voltage	I/O
1GE3	1000	-1	0.8V	400
1GE75	784	-1	0.8V	400
1VE3	1000	-1	0.8V	400
GEMINI	784	-1	0.8V	400
GEMINI_COM...	784	-1	0.8V	400
GEMINI_COM...	784	-1	0.8V	400
GEMINI_COM...	784	-1	0.8V	400

7. At this stage, you will find a summary of your project. To complete the process, click "Finish" to close the New Project Wizard.



New Project Summary

A new RTL project named 'Debug_design' will be created:

1 source files will be added.

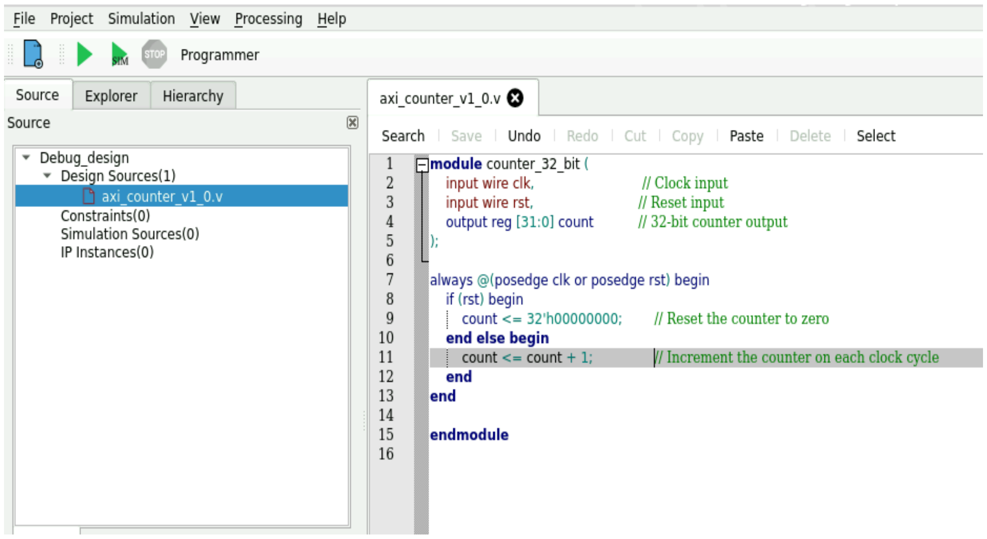
No simulation files or directories will be added. Use Add Sources to add them later.

No constraints files will be added. Use Add Sources to add them later.

Series	Family	Package	D
Gemini	Gemini	F784	GEMINI_C

To create the project, click Finish.

8. Upon completing the New Project Wizard, your project window will resemble the figure shown below.



```
module counter_32_bit (
    input wire clk,                    // Clock input
    input wire rst,                    // Reset input
    output reg [31:0] count            // 32-bit counter output
);

always @(posedge clk or posedge rst) begin
    if (rst) begin
        count <= 32'h00000000;         // Reset the counter to zero
    end else begin
        count <= count + 1;            // Increment the counter on each clock cycle
    end
end

endmodule
```

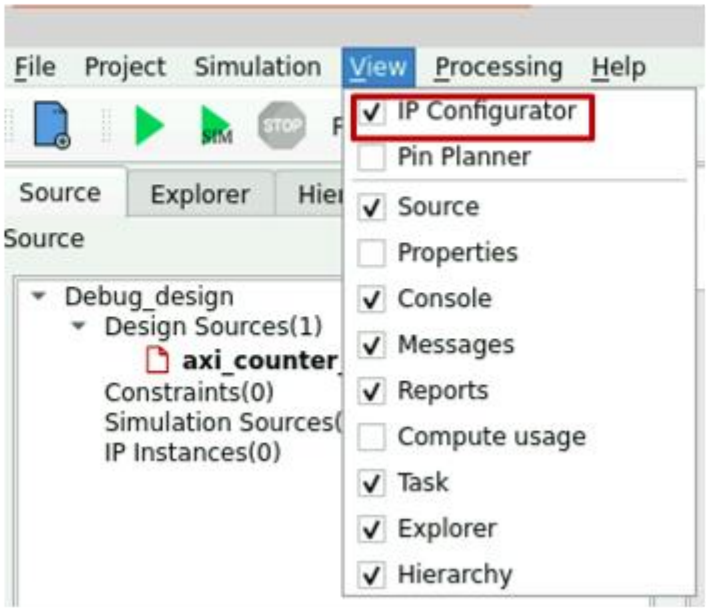
After creating the project and adding the design file, the next step is to generate the OCLA IP using the IP configurator.

Generating OCLA IP

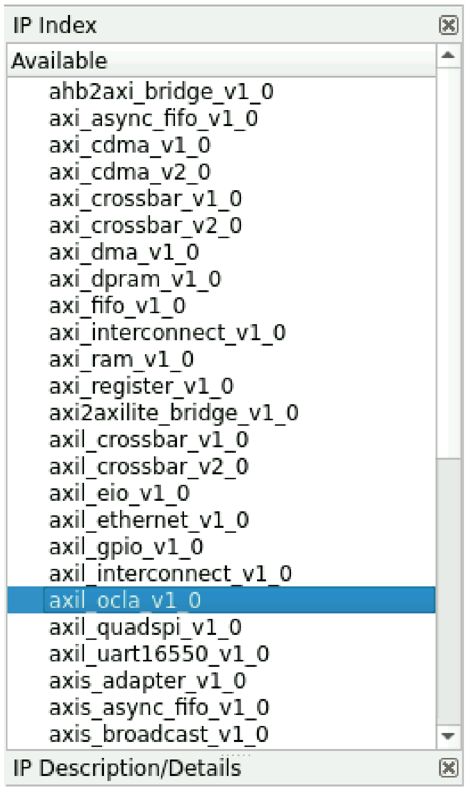
To utilize the OCLA IP, our initial step involves generating the IP according to our specific requirements using the IP configurator. In this step we will generate a customize IP and then in the subsequent step we will integrate it in our design

Let's proceed with the generation of the OCLA IP by following the steps below.

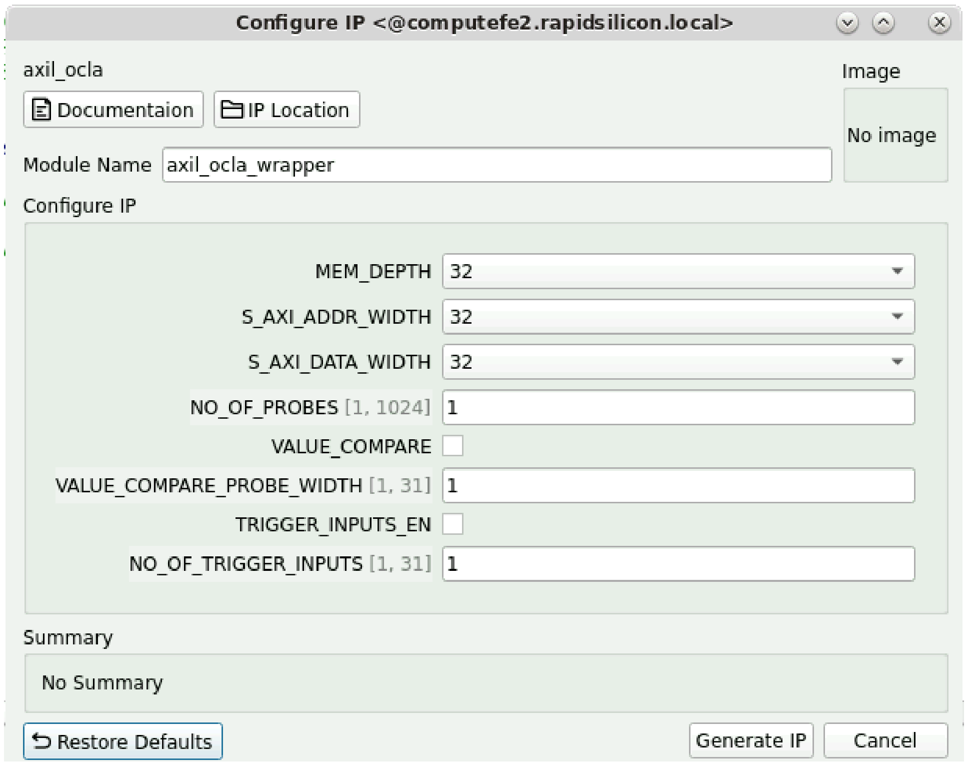
1. Navigate to "View" and check the "IP Configurator". By default, it will be unchecked.



2. After checking the "IP Configurator," an IP catalog list will appear on the left side of Raptor, displaying all available IPs. Choose "axil_ocla_v1_0" from the list.



3. Double-click on the IP, and this action will open the IP configurator tab, as depicted below. From this tab, you can proceed to configure the IP to your specific requirements.


Configure IP <@computefe2.rapidsilicon.local>

axil_ocla

Module Name: axil_ocla_wrapper

Configure IP

MEM_DEPTH	32
S_AXI_ADDR_WIDTH	32
S_AXI_DATA_WIDTH	32
NO_OF_PROBES [1, 1024]	1
VALUE_COMPARE	☐
VALUE_COMPARE_PROBE_WIDTH [1, 31]	1
TRIGGER_INPUTS_EN	☐
NO_OF_TRIGGER_INPUTS [1, 31]	1

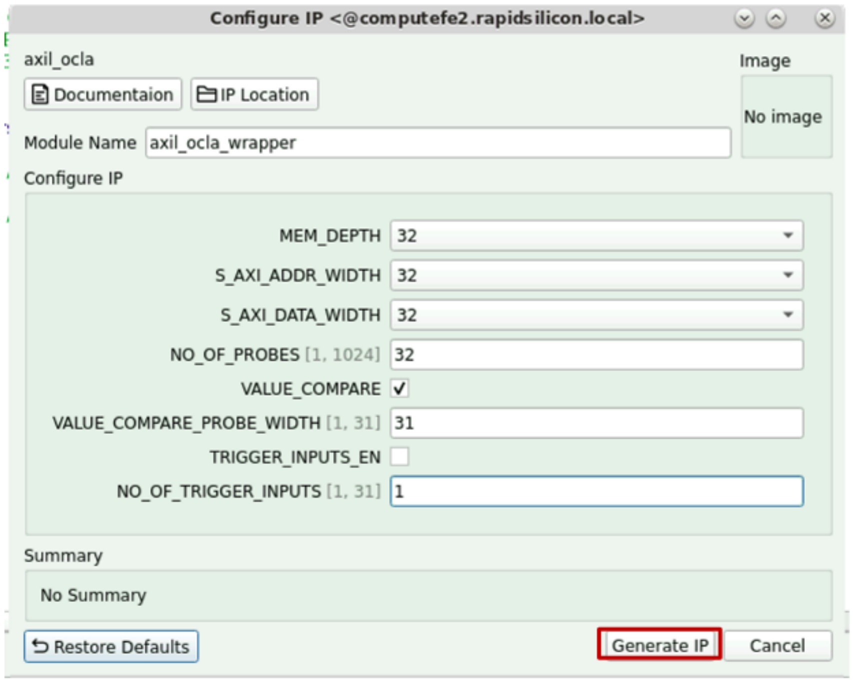
Summary

No Summary

4. Let's proceed to configure all the fields by following the steps below.
   a. Check the "VALUE_COMPARE" field, if you want to compare counter value with a pre-defined value.
   b. Set "VALUE_COMPARE_PROBE_WIDTH" to 32. This corresponds to the 32-bit width of our counter, allowing us to compare all 32 bits. This option is valid only if "VALUE_COMPARE" is checked.
   c. Set "NO_OF_PROBES" to 32, since we want to probe the output of the counter which is 32 bits. This ensures we have enough probes to connect all the bits of the DUT to OCLA.
   d. Keep the "MEM_DEPTH" field at 32. This means OCLA will capture 32 samples. If you need to capture more samples, you can adjust this field, but be aware that it will consume more resources.
   e. Leave all other fields at their default settings.
   f. TRIGGER_INPUTS_EN: This option gives you access to 32 additional probes that can be used for debugging. In this example we are not using it.

For detailed information about each of these fields, please consult the OCLA User Guide.

5. Once you have configured all the fields in step 4, click "Generate IP." as shown below. This action will generate the necessary IP files and a top-level wrapper that we'll use for instantiation in our design.



Configure IP <@computefe2.rapidsilicon.local>

axil_ocla

Module Name: axil_ocla_wrapper

Configure IP

MEM_DEPTH: 32

S_AXI_ADDR_WIDTH: 32

S_AXI_DATA_WIDTH: 32

NO_OF_PROBES [1, 1024]: 32

VALUE_COMPARE: ✓

VALUE_COMPARE_PROBE_WIDTH [1, 31]: 31

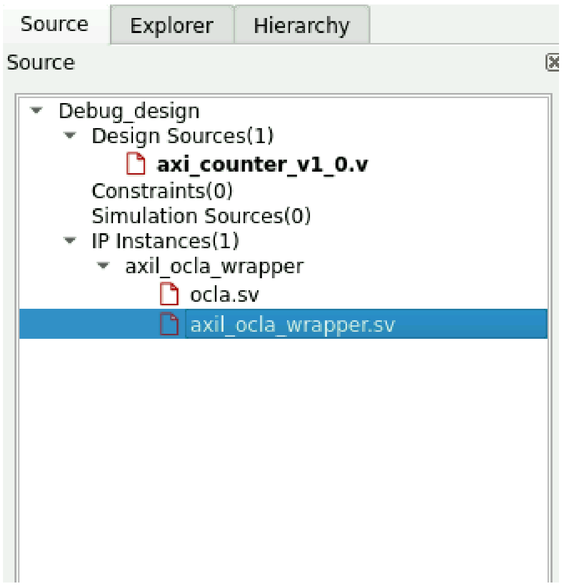
TRIGGER_INPUTS_EN: ☐

NO_OF_TRIGGER_INPUTS [1, 31]: 1

Summary

No Summary

6. After generating the IP, you can locate the source files and top-level wrapper in the "Source" tab under "IP Instances," as illustrated below.



```
Debug_design
  Design Sources(1)
      axi_counter_v1_0.v
  Constraints(0)
  Simulation Sources(0)
  IP Instances(1)
    axil_ocla_wrapper
        ocla.sv
        axil_ocla_wrapper.sv
```

In addition to OCLA, you'll also require the following IPs:

1. JTAG2AXI

2. AXI Interconnect
3. AXI2AXILite Bridge

You can generate these IPs by following all the steps mentioned in the IP Generation section.

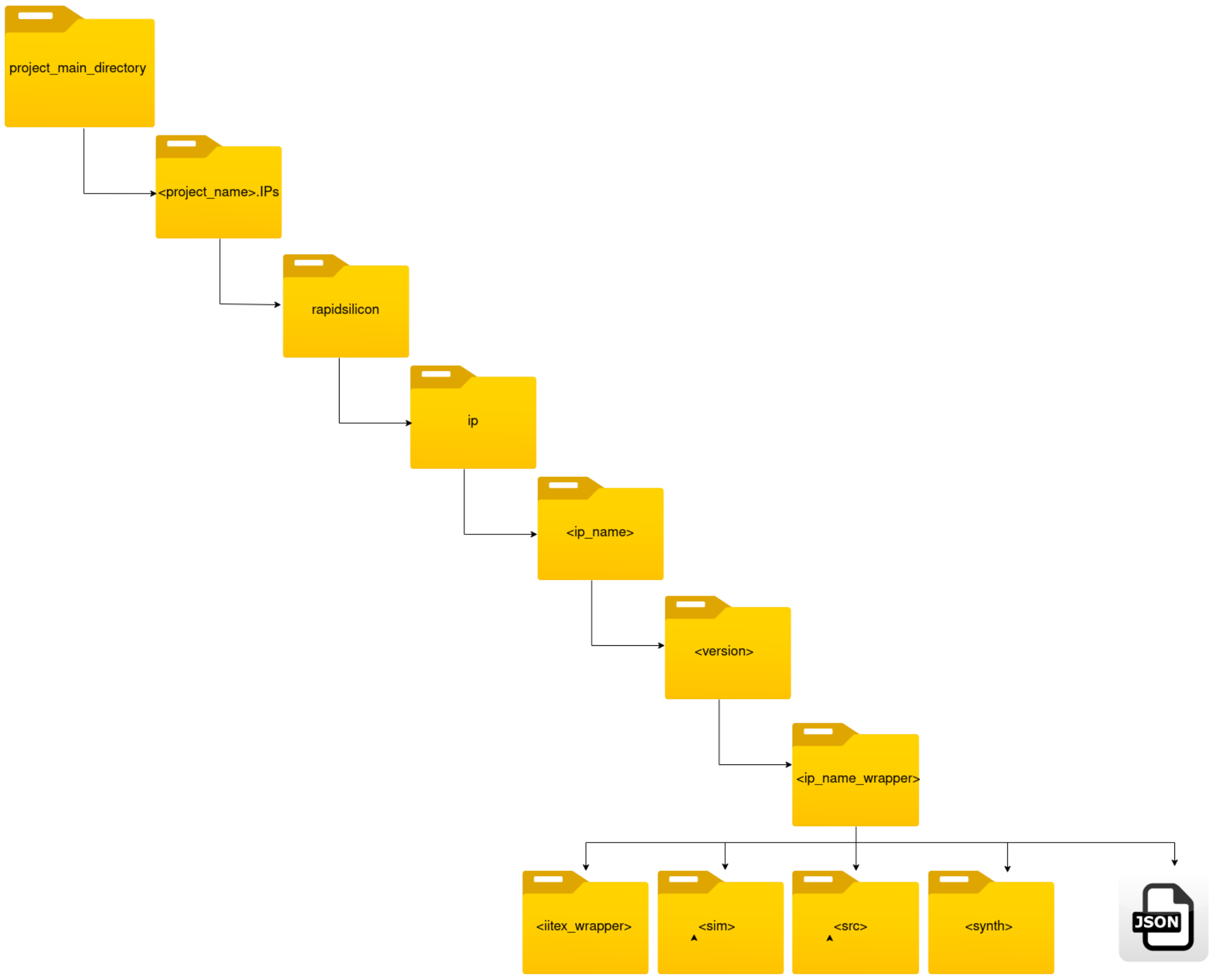
You can find the User Guide of the above IPs in IP catalog for detailed explanation.

Our next step is to add the generated IP files to the project.

IP files addition to Project

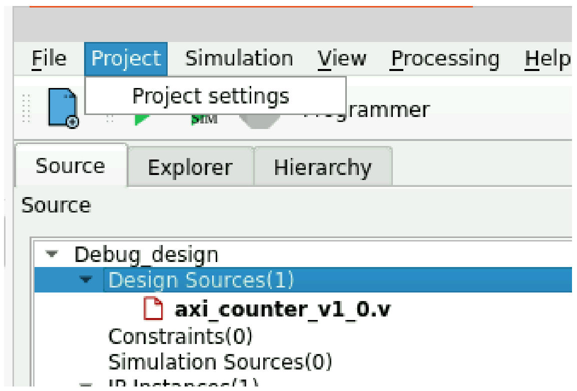
In this section, we will guide you through the process of adding the OCLA IP files, generated by the IP configurator, to your main project.

The IP configurator creates an IP directory within the main project directory. This IP directory contains all the source files and other related files associated with that particular IP. The figure below illustrates the complete hierarchy generated by the IP configurator. Within this hierarchy, the IP source files are located inside the "src" folder.

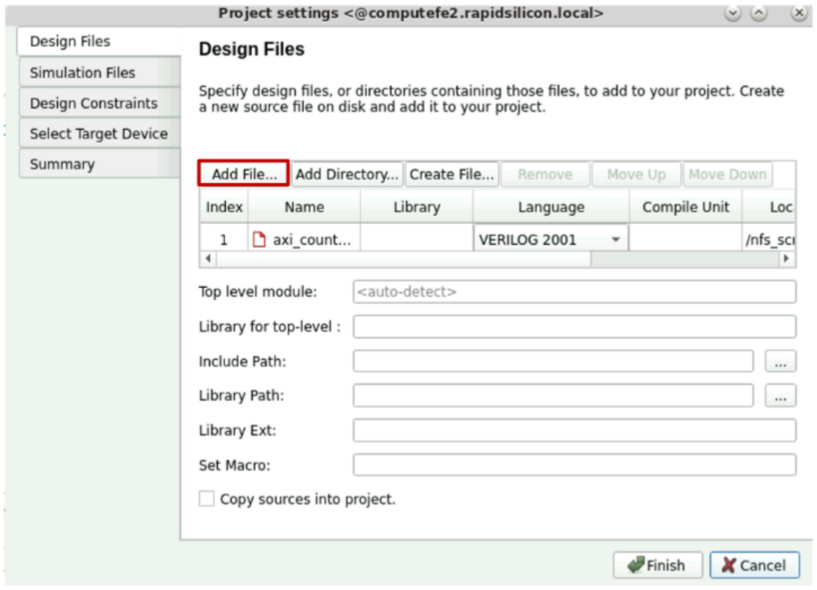


To include these files in the project, navigate to the "src" directory within Raptor and add them. Follow the steps below to add all the IP source files to the main project.

1. Navigate to "Project" tab and select "Project Settings."



2. Once the Project Settings window is open, click on "Add File" and select all the IP source files from the IP source directory mentioned above. Then, click "Finish" to complete the process.

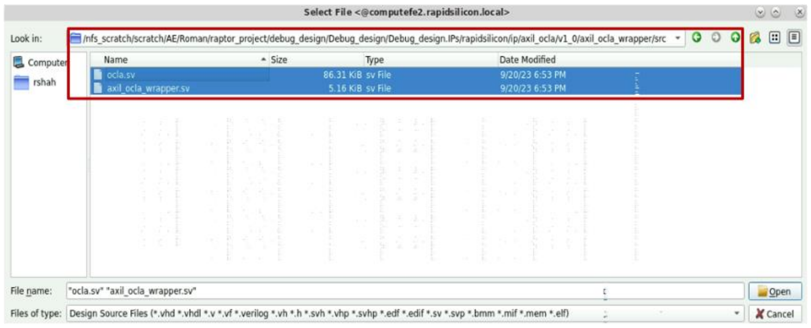


Design Files

Specify design files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project.

Index	Name	Library	Language	Compile Unit	Loc
1	axi_count...		VERILOG 2001		/nfs_sc

Top level module: <auto-detect>
Library for top-level :
Include Path:
Library Path:
Library Ext:
Set Macro:
Copy sources into project.

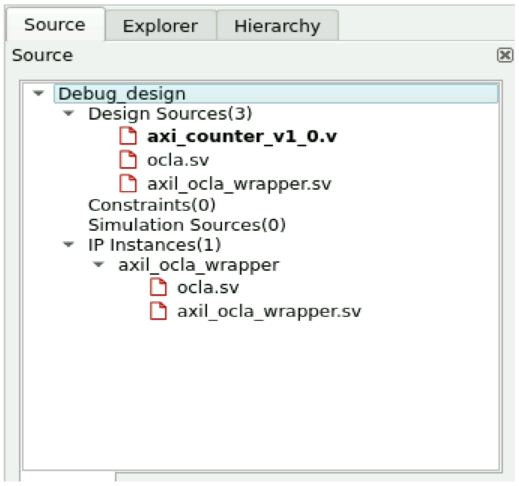


Look in: /nfs_scratch/scratch/AE/Roman/raptor_project/debug_design/Debug_design/Debug_design.IPs/rapidsilicon/ip/axil_ocla/v1_0/axil_ocla_wrapper/src

Name	Size	Type	Date Modified
ocla.sv	86.31 KiB	sv File	9/20/23 6:53 PM
axil_ocla_wrapper.sv	5.16 KiB	sv File	9/20/23 6:53 PM

File name: "ocla.sv" "axil_ocla_wrapper.sv"
Files of type: Design Source Files (*.vhd *.vhdl *.v *.vf *.verilog *.vh *.h *.svh *.vhp *.svhp *.edf *.edif *.sv *.svp *.bmm *.mif *.mem *.elf)

3. In the "Source" tab, under "Design Sources," you will now see that the IP files are available and included in your project.

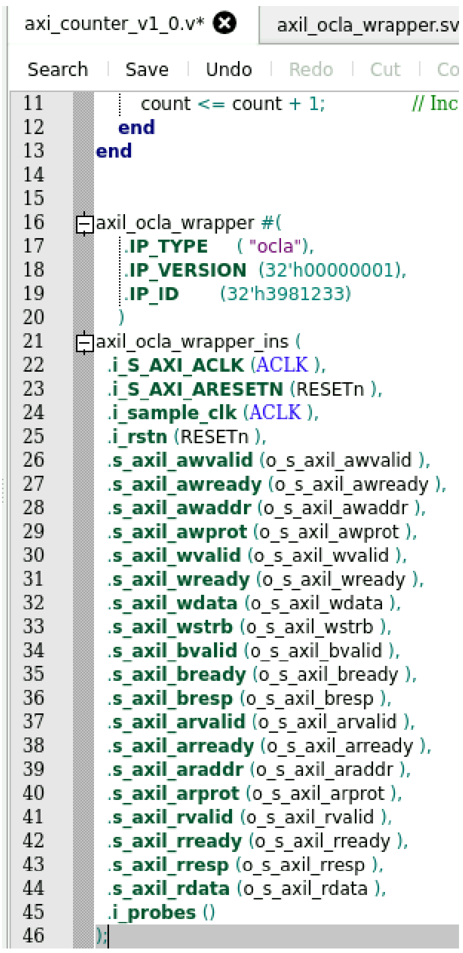


```
Debug_design
  Design Sources(3)
      axi_counter_v1_0.v
      ocla.sv
      axil_ocla_wrapper.sv
  Constraints(0)
  Simulation Sources(0)
  IP Instances(1)
    axil_ocla_wrapper
        ocla.sv
        axil_ocla_wrapper.sv
```

Instantiating OCLA IP in Design

With all the necessary IPs and design files in place, this section will guide you through the process of instantiating the OCLA IP within design.

1. Create an instance of the OCLA IP within your design, which you intend to debug as shown below.



```
        count <= count + 1;            // Inc
    end
end


axil_ocla_wrapper #(
    .IP_TYPE     ( "ocla"),
    .IP_VERSION  (32'h00000001),
    .IP_ID       (32'h3981233)
    )
axil_ocla_wrapper_ins (
  .i_S_AXI_ACLK (ACLK ),
  .i_S_AXI_ARESETN (RESETn ),
  .i_sample_clk (ACLK ),
  .i_rstn (RESETn ),
  .s_axil_awvalid (o_s_axil_awvalid ),
  .s_axil_awready (o_s_axil_awready ),
  .s_axil_awaddr (o_s_axil_awaddr ),
  .s_axil_awprot (o_s_axil_awprot ),
  .s_axil_wvalid (o_s_axil_wvalid ),
  .s_axil_wready (o_s_axil_wready ),
  .s_axil_wdata (o_s_axil_wdata ),
  .s_axil_wstrb (o_s_axil_wstrb ),
  .s_axil_bvalid (o_s_axil_bvalid ),
  .s_axil_bready (o_s_axil_bready ),
  .s_axil_bresp (o_s_axil_bresp ),
  .s_axil_arvalid (o_s_axil_arvalid ),
  .s_axil_arready (o_s_axil_arready ),
  .s_axil_araddr (o_s_axil_araddr ),
  .s_axil_arprot (o_s_axil_arprot ),
  .s_axil_rvalid (o_s_axil_rvalid ),
  .s_axil_rready (o_s_axil_rready ),
  .s_axil_rresp (o_s_axil_rresp ),
  .s_axil_rdata (o_s_axil_rdata ),
  .i_probes ()
);
```

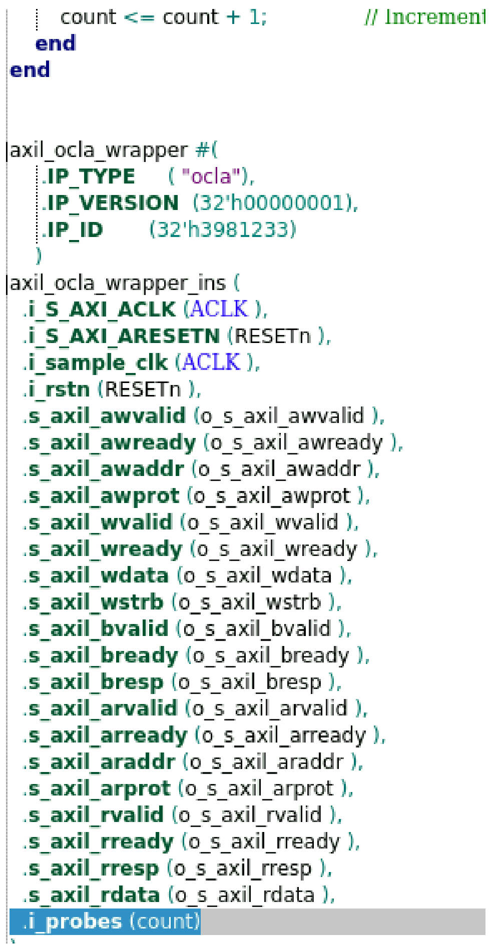
In the above figure you can see that there are three parameters of OCLA IP. All these parameters are auto generated.

The first parameter is IP_TYPE. This parameter defines the type of IP and should not be modified.

The second one is IP_VERSION. This parameter specifies the IP version, and similarly, it should not be altered. Keep it in its default state.

The last one is IP_ID. The IP configurator will automatically assign a unique ID for each OCLA instance you generate. If you intend to instantiate the OCLA IP multiple times in your design, it's recommended to generate the IP separately for each instance. Each time you generate the IP, the IP configurator will assign a different ID. However, if you've generated the IP once and intend to instantiate it multiple times, it becomes your responsibility to ensure that you manually change the last digit of IP_ID , ensuring that it remains unique for each instance.

2. Connect the signals you want to debug to the "i_probe" port of the OCLA IP. In this tutorial, we are focusing on debugging a 32-bit counter, so we will connect the counter signal “count”to this probe. You can see the connection in the figure below.



```
    count <= count + 1;            // Increment
  end
end


axil_ocla_wrapper #(
  .IP_TYPE     ( "ocla"),
  .IP_VERSION  (32'h00000001),
  .IP_ID       (32'h3981233)
  )
axil_ocla_wrapper_ins (
  .i_S_AXI_ACLK (ACLK ),
  .i_S_AXI_ARESETN (RESETn ),
  .i_sample_clk (ACLK ),
  .i_rstn (RESETn ),
  .s_axil_awvalid (o_s_axil_awvalid ),
  .s_axil_awready (o_s_axil_awready ),
  .s_axil_awaddr (o_s_axil_awaddr ),
  .s_axil_awprot (o_s_axil_awprot ),
  .s_axil_wvalid (o_s_axil_wvalid ),
  .s_axil_wready (o_s_axil_wready ),
  .s_axil_wdata (o_s_axil_wdata ),
  .s_axil_wstrb (o_s_axil_wstrb ),
  .s_axil_bvalid (o_s_axil_bvalid ),
  .s_axil_bready (o_s_axil_bready ),
  .s_axil_bresp (o_s_axil_bresp ),
  .s_axil_arvalid (o_s_axil_arvalid ),
  .s_axil_arready (o_s_axil_arready ),
  .s_axil_araddr (o_s_axil_araddr ),
  .s_axil_arprot (o_s_axil_arprot ),
  .s_axil_rvalid (o_s_axil_rvalid ),
  .s_axil_rready (o_s_axil_rready ),
  .s_axil_rresp (o_s_axil_rresp ),
  .s_axil_rdata (o_s_axil_rdata ),
  .i_probes (count)
```

3. Instantiate all the other required IPs in your design and establish the necessary connections as shown in the above block diagram.
   a. Connect the AXI Interface of JTAG2AXI IP to slave interface of AXI2AXILITE IP, while the JTAG interface of JTAG2AXI IP will become the ports for top wrapper, through which it will communicate with JTAG device.
   b. The master interface of AXI2AXILITE will connect to slave interface of AXI Interconnect.
   c. The master interface of AXI Interconnect will connect with AXI interface of OCLA.
   d. Connect the probe of OCLA with DUT signals whom user wants to debug.

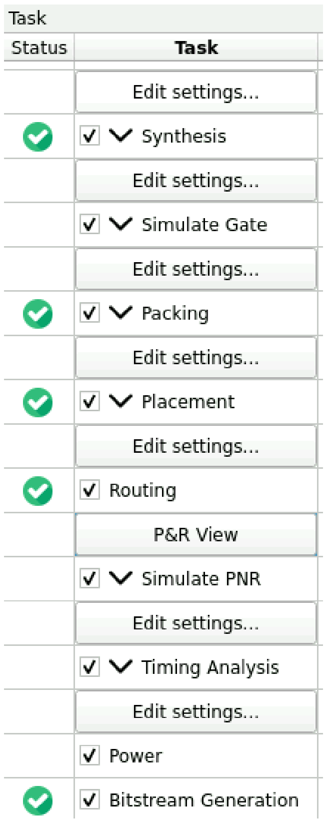
Now our design is ready to be compiled using Raptor.

Running Design using Raptor

To generate the bitstream for your design, which will be uploaded to the target device, follow these steps.

1. Run Analysis of design.
2. Run Synthesis
3. Run Packing, placement and then routing
4. Generate Bitstream.

You can conveniently accomplish each of these steps by simply double-clicking on the respective task in the task list available on the left side in Raptor as shown in figure below.



Task

Status	Task
	Edit settings...
✓	✓ Synthesis
	Edit settings...
	✓ Simulate Gate
	Edit settings...
✓	✓ Packing
	Edit settings...
✓	✓ Placement
	Edit settings...
✓	✓ Routing
	P&R View
	✓ Simulate PNR
	Edit settings...
	✓ Timing Analysis
	Edit settings...
	✓ Power
✓	✓ Bitstream Generation

With the bitstream generated, the next step is to load it onto the target FPGA. After programming the FPGA, we will use some commands (CLI) to configure the OCLA and then examine the OCLA buffer to verify the functionality of the counter.

Communicating with OCLA through CLI Commands

In this section, we will explain how to communicate with OCLA through the JTAG interface using CLI commands.

The first thing is to start the debugger session. To initiate a debugger session, start by issuing the following command:

```
debugger session start -f <filepath .bitasm>
```

Once the session has started, the next step involves configuring the OCLA. During this configuration, you will set the trigger mode, which can be either pre-trigger, post-trigger, or centered trigger.
To set these configuration issue the command below

```
debugger config -i 1 -m pre-trigger
```

-i / --instance Specify the index of the OCLA instance to configure. The index starts from 1. The allowed values are 1 and 2 as per OCLA IP.
-m / --mode Configure the trigger mode of the OCLA instance.

-m {**pre-trigger** | post-trigger | center-trigger | disable}

Next is to configure a trigger channel, type and probe signal to monitor in a specified OCLA instance. So, use the command below to configure OCLA channel.

```
debugger config_channel -i 1 -c 1 -e edge -t rising -v number -p <signal name>
```

-c / --channel Specify the channel to configure. NOTE: There are 4 channels in the current OCLA IP, so the valid numbers are from 1 to 4
-e / --event Configure the trigger option of the specified channel.

-e {**edge** | level | value_compare}

-t / --type Configure the trigger option type according to the specified trigger option.

-t {**rising** | falling | either | high | low | equal | lesser |greater}

-v / --value Compare value. **NOTE**: Only required for **value_compare** trigger type.
-p / --probe Configure a probe signal to monitor for the specified channel.

So now all the triggers are set and Arm the OCLA and start sampling when trigger hit. To start OCLA send the command shown below

```
debugger start -i 1 -t seconds -o <output file path>
```

-t / --timeout Optional. Command timeout in second.
-o / --output Optional. Specific the filepath to write the output to. If not specified, a default filepath will be chosen.

After initiating the debugger by issuing the above command, it will read the OCLA buffer and generate a file. The next step is to open that file in GTK Wave for further analysis.

To read that file in GTK Wave send the command below.

```
debugger show_waveform -i {input filepath}
```

-i / --input Optional. Specific the input file path to open in GTK Wave. If not specified, a default file path will be used.

NOTE: When input option is not specified, use the default path and filename as in “debug start” command output option.

Debug System Limitations

The current debug system has a few limitations which are following.

1. This system only allows 2 OCLA instances
2. User is responsible to set base address for both OCLA instance.
3. A maximum of 4 triggers can be set for each OCLA instance

Appendix A

```
module counter_32_bit (
input wire clk,
input wire rst,
output reg [31:0] count


always @(posedge clk or posedge rst) begin
if (rst) begin
count <= 32'h00000000;
end else begin
count <= count + 1;
end
end


endmodule
```