QSYS PRO

Generic Component

March 2017

Contents

Introduction

This document is a fundamental guide for user that would like to get started in building a system with Generic Component in Qsys Pro and Quartus Prime. A very simple software is used to verify the functionality of the Generic component created using a Arria 10 development kit. This user guide starts by creating the Quartus Prime project and Qsys Pro files from scratch.

The generic component is a new and special type of IP component that enables isolated IP instantiation and blackboxing in Qsys Pro. Users can select this component from IP catalog and instantiate it into the system. By default, this component will not have an RTL implementation provided within Qsys Pro. The implementation for the component, i.e the declaration and definition of the component is provided by the users later in downstream tools. Optionally, users can choose to link the implementation of the generic component to another file present within the Quartus Project. For example, RTL files, .qsys, .ip, .IPXACT files. There are three implementation types which are IP implementation, HDL implementation and Blackbox implementation

Hardware:

- Arria 10 SoC development kit 10AS066N3F40E2SG
- Power adapter
- USB Blaster

Software:

- Quartus Prime Pro version 17.0
- Please refer to this link for Arria 10 SoC development kit documentation and installation files

You can download the available design file

Creating a new project in Quartus Prime Pro

We shall start designing the hardware by including the components needed into Qsys Pro. First, we would need to create a Quartus Prime Pro project. Refer to section creating new project to see how to create a new project in Quartus Prime Pro.

Building Hardware Design in Qsys Pro

Creating a new Qsys Pro file

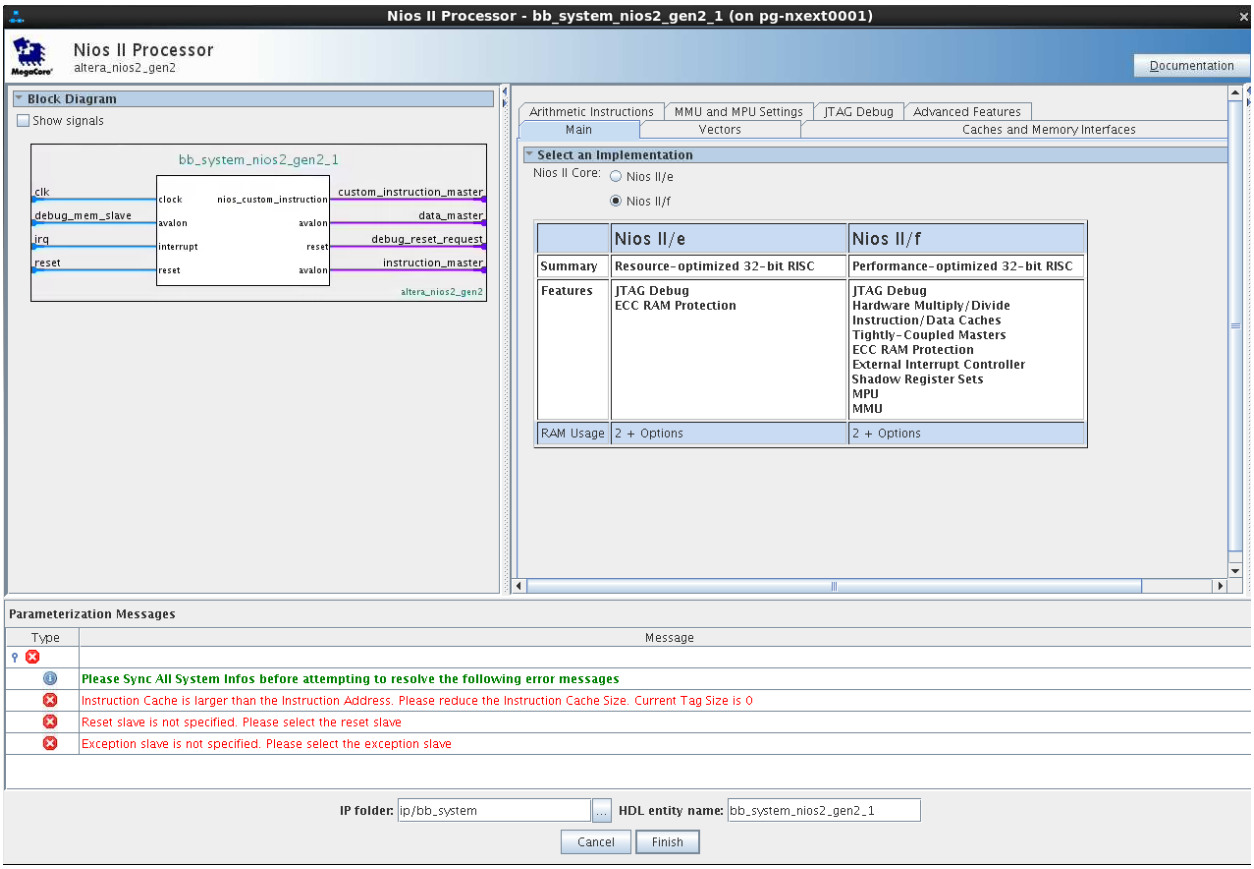
Next, we shall start to create the Qsys Pro file. Refer to section creating Qsys Pro file to see how to create a new Qsys Pro file.

Adding components into the system

1. At the left top corner of the GUI, switch the tab from “Hierarchy” to “IP Catalog” to search for the IPs that is already available in the Catalog to use.
2. At the  icon, type in the name of the IP that you would like to add into the system.

Adding Nios II Processor

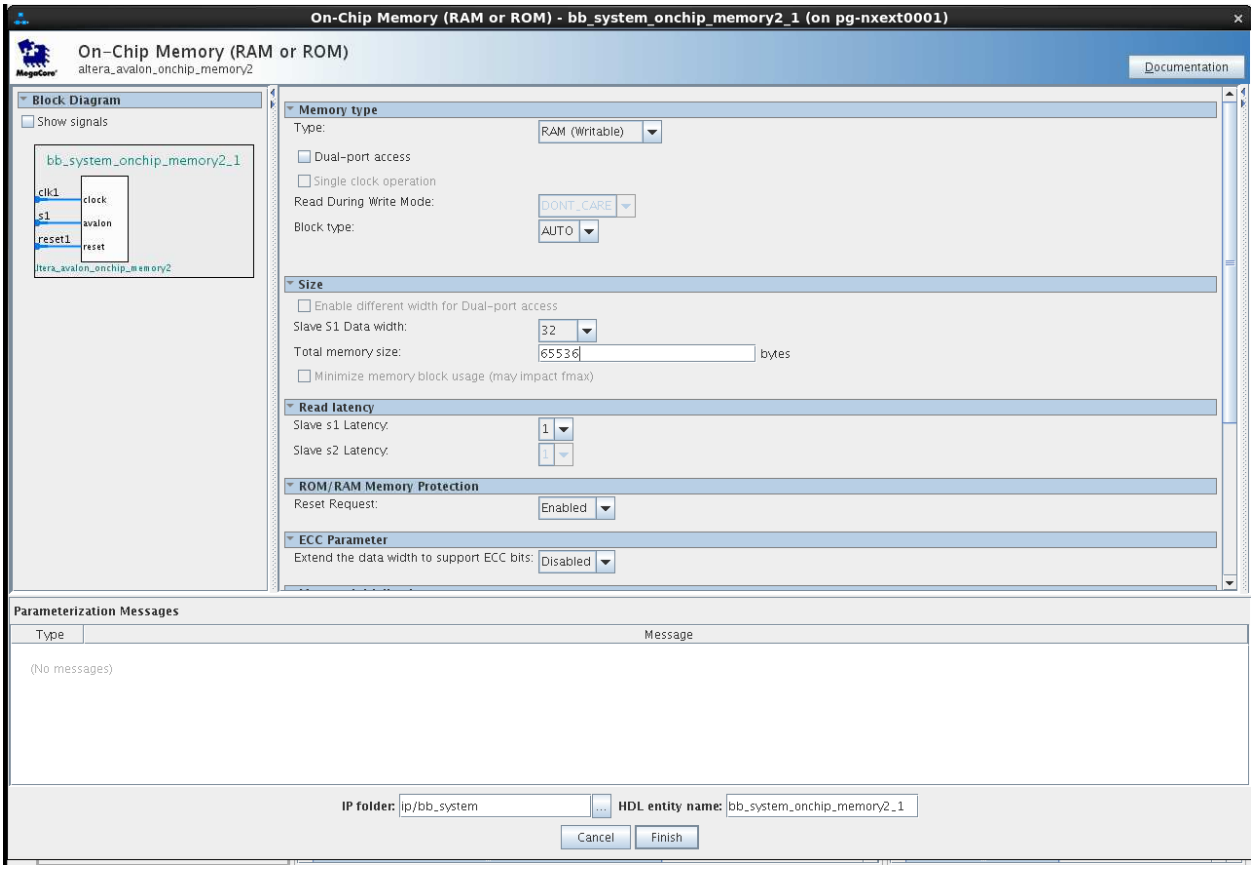
1. Search for “Nios II Processor” in IP Catalog.
2. Double click on the Nios II Processor IP in “Embedded Processors” under “Processors and Peripherals” section.
3. The Parameter box of Nios II Processor will be displayed.



4. Click “Finish”. Leave the settings to be default for now. We can change it later..

Adding On-Chip Memory

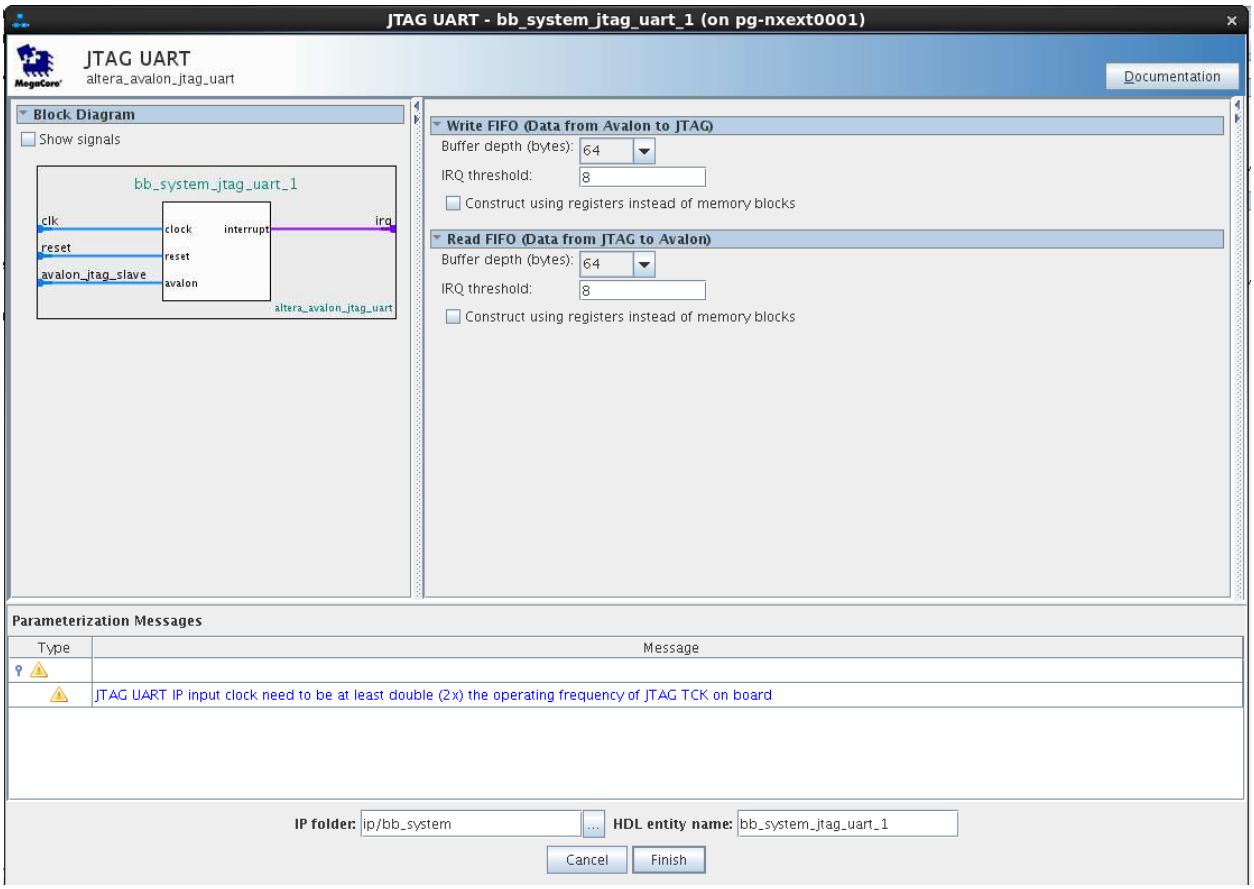
1. Search for On-Chip Memory in the IP Catalog.
2. Double click on the On-Chip Memory (RAM or ROM) IP in “On Chip Memory” under “Basic Functions” section.
3. The Parameter box of On-Chip Memory will be displayed.



4. Click “Finish”. Leave the settings to be default for now. We can change it later.

Adding JTAG UART

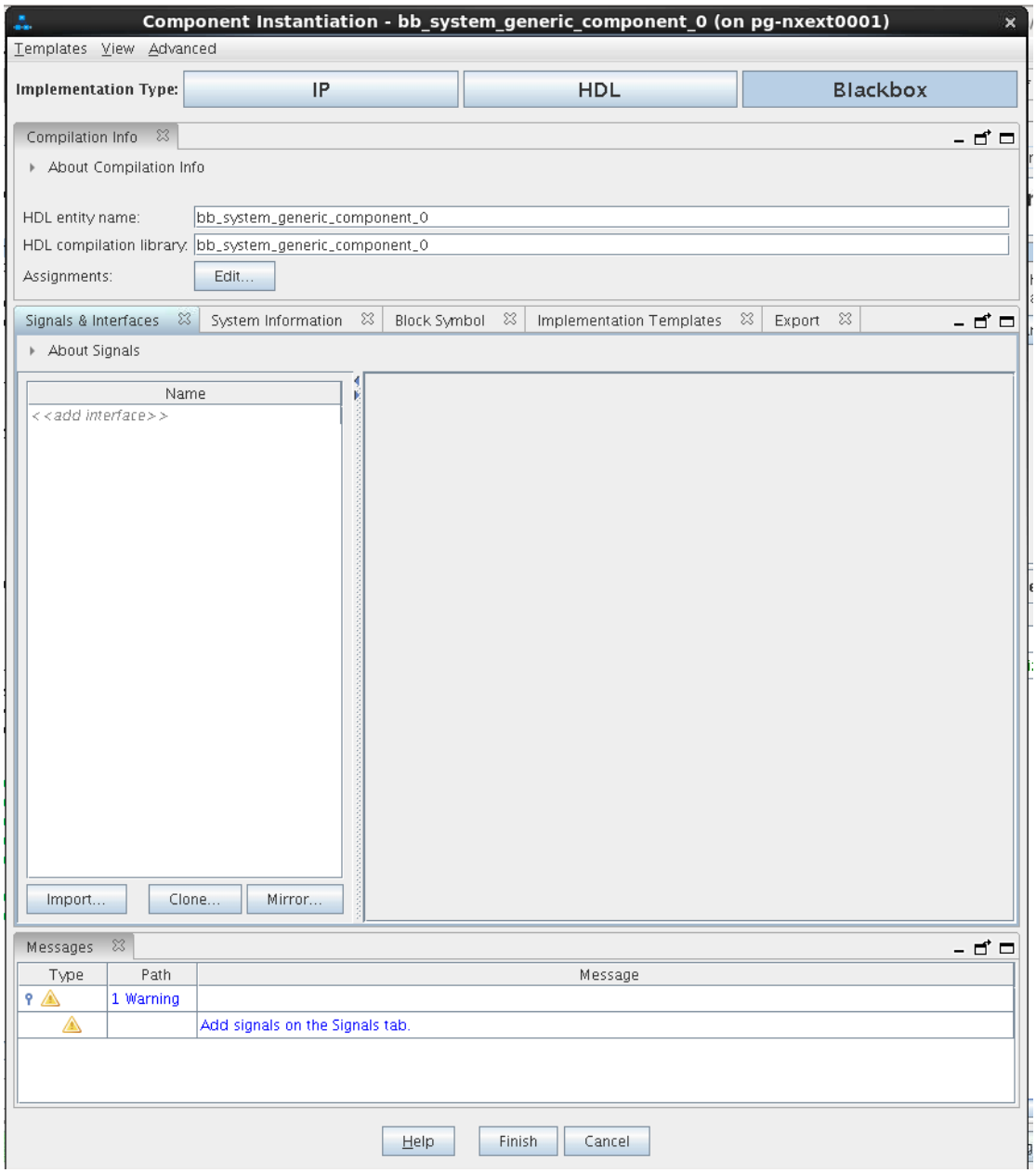
1. Search for Jtag Uart in the IP Catalog.
2. Double click on the Jtag Uart IP in “Serial” under “Interface Protocols” section.
3. The Parameter box of Jtag Uart will be displayed.



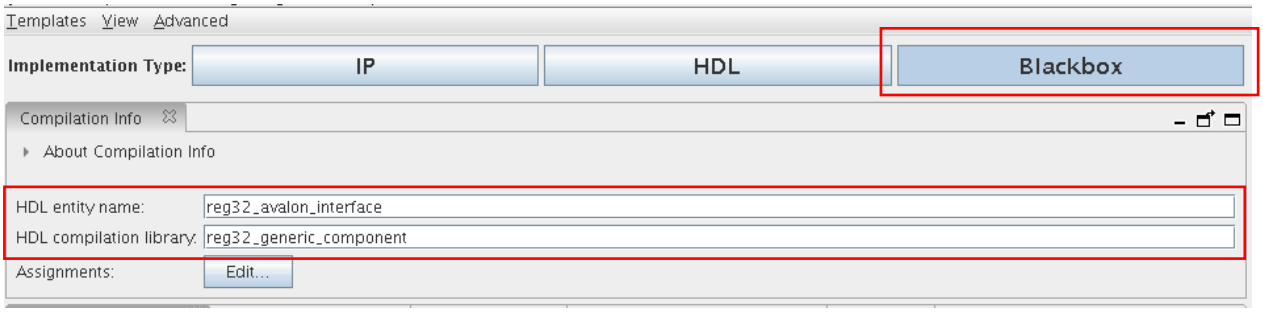
4. Click “Finish”. Leave the settings to be default for now. We can change it later.

Adding Generic Component

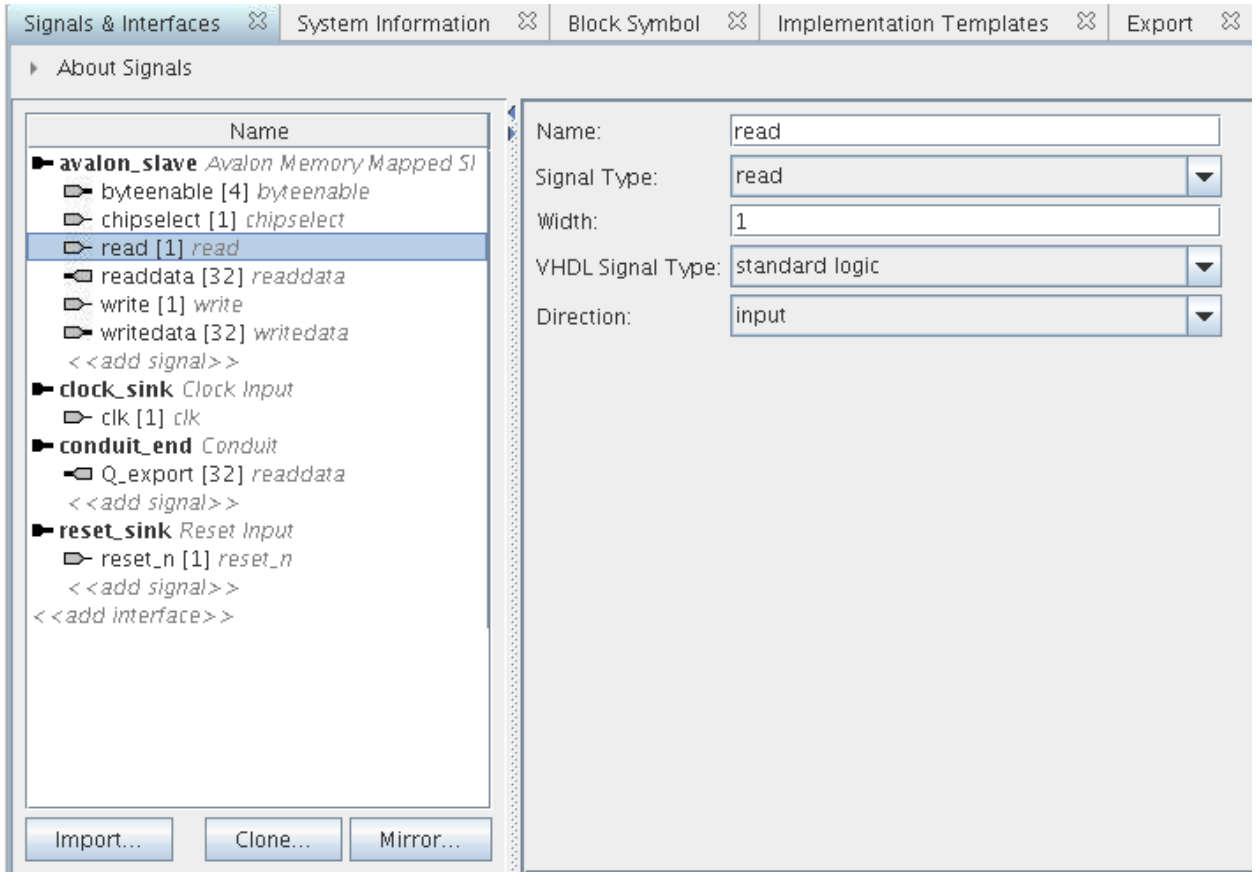
1. Search for Generic Component in the IP catalog
2. Double click on the Generic Component IP under Generic Component section
3. The parameter box for Generic Component IP will be displayed



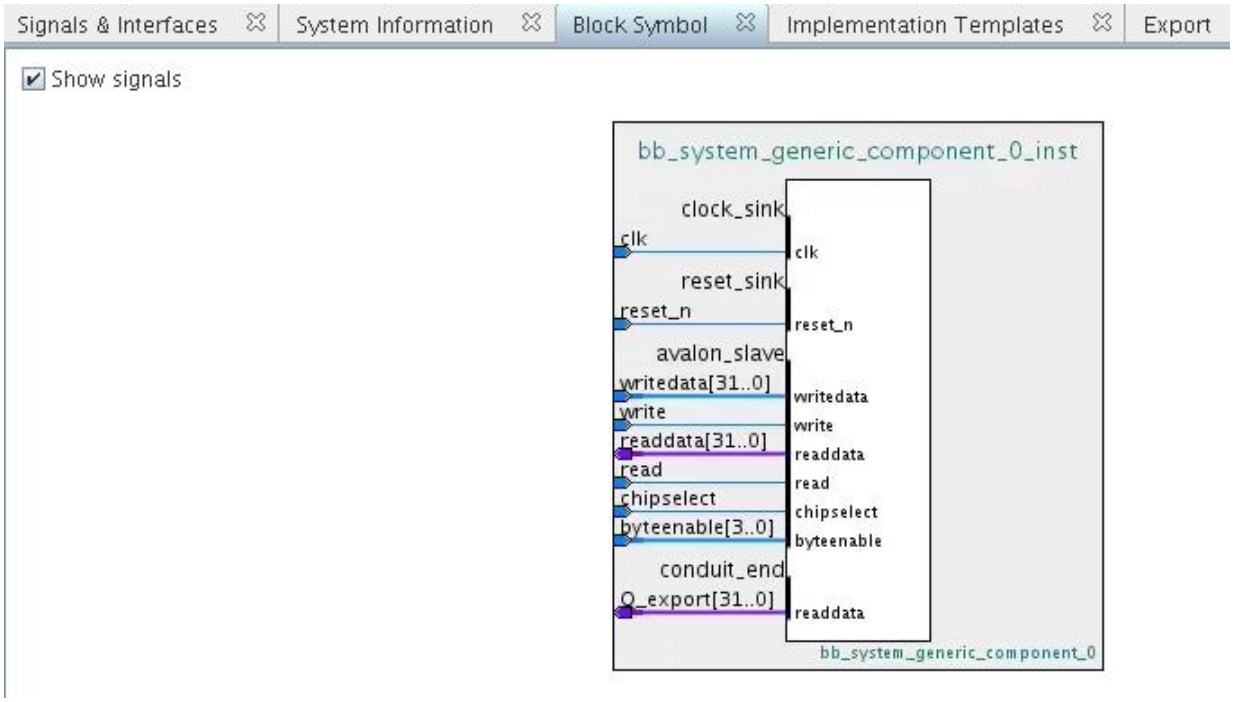
4. In the generic component's parameter tab, you will see the implementation type is point to Blackbox by default. In the compilation info tab, there is HDL entity name and HDL compilation library which users can specify the prefer naming. In this example, we will choose Blackbox implementation and use the entity name as shown in the picture below,



5. Now, you can add signals and interfaces through the signal and interfaces tab. You can add the types of signals and interfaces by clicking the <<add interface>> or <<add signals>>. Other than that, you can also import the signal from any existing IP/IP-XACT files, cloning and mirroring the interfaces from the other IP in your current design. In this example, we will create the signal and interface by adding one by one. After you created an interfaces and signal, you would need to specify the signal type, width, VHDL signal type and direction for each signal created.



6. The system information tab would let the users to specify the address map of the interfaces and other necessary system information associated with the component. In this example, we will not add any information in this tab.
7. Once you have created the interfaces and signal, you can look at the block diagram which is in the block symbol tab. The block diagram shows all the signals and interfaces that you have just created.

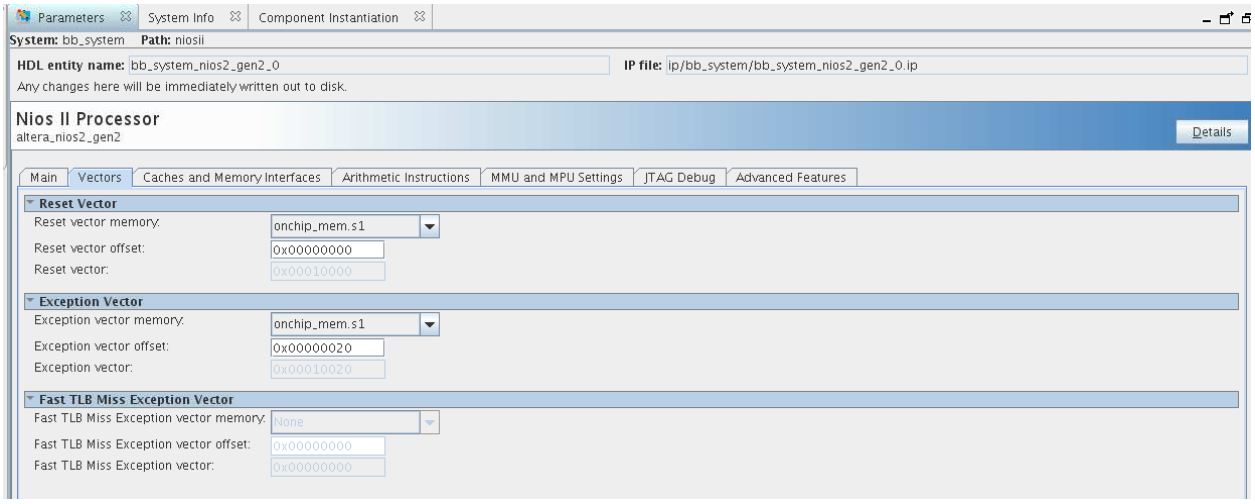


8. The Implementation templates tab helps users to create either Qsys template and HDL templates. The Qsys templates helps users in the way if the users are trying to create another system in Qsys which need to comply with the interfaces and signals of that generic component. The HDL templates help users to create their RTL design that will comply with the interfaces and signals. In this example, the design files are ready and hence these options are not needed.
9. In the export tabs, users can export an IP-XACT files or _hw.tcl file. The IP-XACT file can be imported when you try to create a new interfaces and signals for a new generic component while the _hw.tcl would help to ease user in creating own custom component to be included into Qsys. In this example, these options are not needed.
10. Click “Finish”

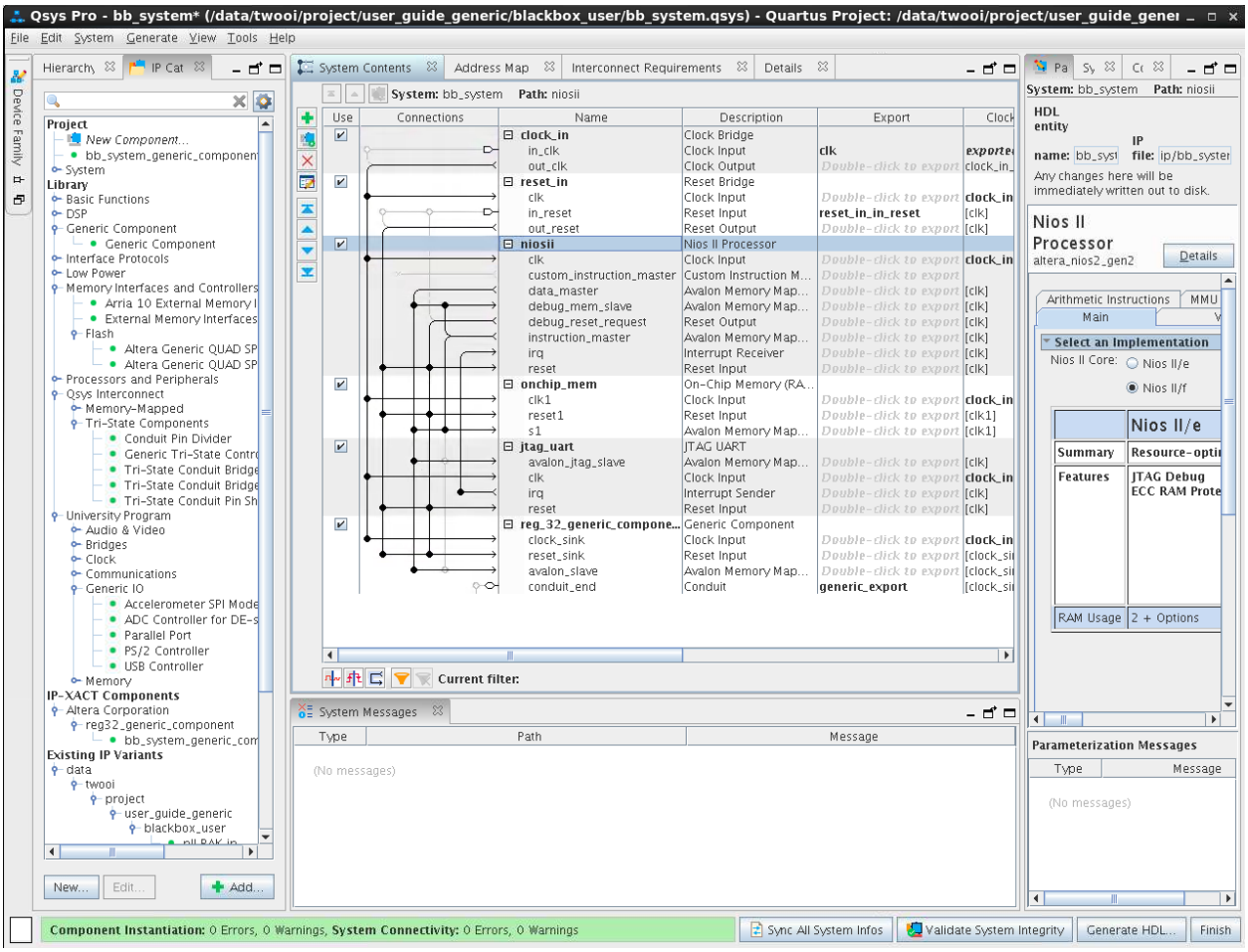
Eliminate error and warning messages

Before you generate the Qsys system, you will see some warning and error messages. This is due to the overlap of base address, mismatched information between what is instantiated in the .qsys with the .ip files and vector of Nios II processor is not set.

1. Go to “System” and click on “Assign Base Addresses”.
2. Click on “Sync All System Infos”. This will sync what is instantiated in the system which is in the .qsys with the IPs as defined in the .ip files.
3. Click on the niosii IP. At the right side of the GUI, you will see the “Parameter” box displayed.
4. Go to the “Vector” tab.
5. Under “Reset vector”, at the “Reset vector memory” dropbox, select onchip_mem.s1 to set the reset vector to on chip memory.
6. Under “Exception Vector”, at the “Exception vector memory” dropbox, select onchip_mem.s1 to set the exception vector to on chip memory.



7. You can validate the system integrity by clicking on “Validate System Integrity”.
8. The whole system that you have designed will be cleaned with errors and warnings.



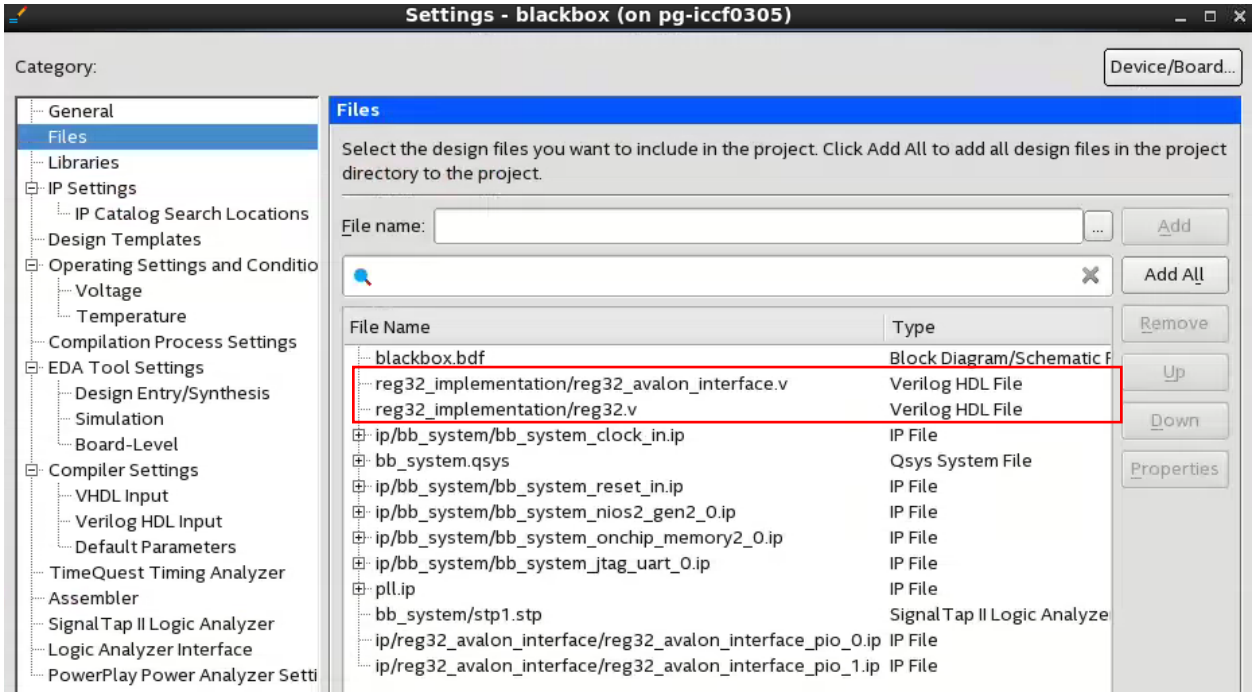
Save the system and generate

1. Go to “File” and click “Save”. This will save all the changes made.
2. Click “Generate HDL…” at the bottom right button of the GUI. This will generate the system in Qsys Pro.

3. After generation completes, the generation files can be obtained in the generated folder.

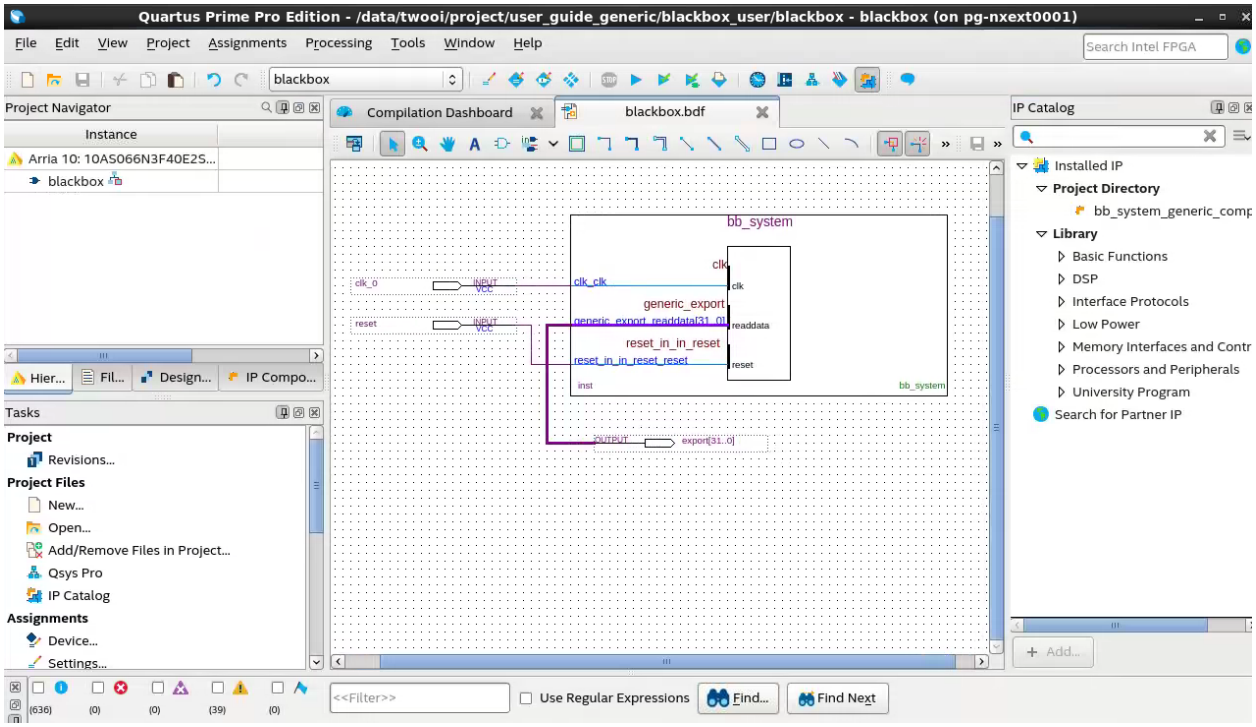
Compilation in Quartus Prime Pro

1. Switch back to Quartus Prime Pro.
2. Now, the implementation files will need to be provided in downstream tool. In this example, the implementation of the component is provided using Quartus. Two design files (reg32.v and reg32_avalon_interface.v) will be included into the project folder. The reg32.v design file is the function of the design while reg32_avalon_interface.v is used to provide the necessary signal to the reg32.v for connecting to an Avalon MM interconnect.



Please note that you would need to create your implementation design with the same signals name as what you have created in generic component. Different in signals name and HDL entity name between the implementation files and generic component in Qsys will causes error during compilation in Quartus.

3. In this example, we will use block diagram file (.bdf) as the top level file. Right click on the block symbol file and import the block symbol file generated from Qsys system. After that, assign input pin and output pin to the Qsys system.



4. Navigate to “Processing”, then go to “Start” and click on “Start Analysis & Synthesis”.

Pin assignments

1. Go to “Assignments” and click on “Pin Planner”.
2. For this design, there are only 2 pins that needs to be assigned and the pin location is based on the Arria 10 SoC schematic:
    - reset assigned to PIN_P3
    - clk_0 assigned to PIN_AN18

The pin assignment for generic_export_read_data will be assigned by Fitter. It is not specified in this example because the functionality of this design will be verified by signal tap.

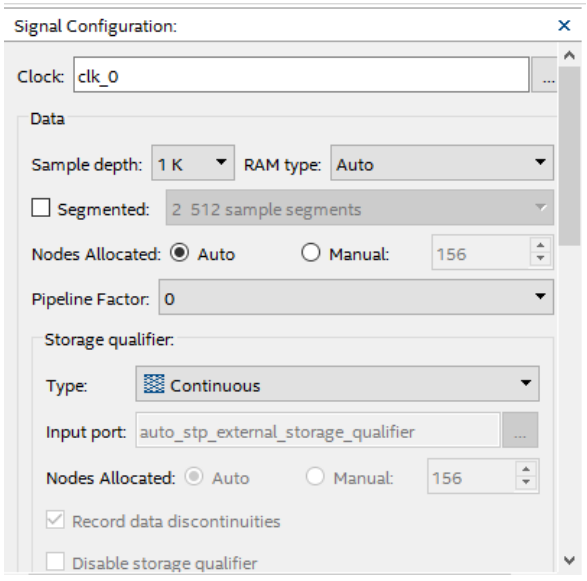
3. Close Pin Planner.

Setup Signal Tap

1. Go to Tools and click Signal Tap Logic Analyzer
2. Create an instance and include all the signal that you wish to examine.

Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	312	312	1 Basic AND
		inst|clock_in|in_clk	☑	☑	
		inst|clock_in|out_clk	☑	☑	
		inst|clk_clk	☑	☑	
		Avalon Wrapper	☑	☑	XXXX1b
		inst|reg_32_generic_component|chipselect	☑	☑	1
		inst|reg_32_generic_component|clk	☑	☑	
		inst|reg_32_generic_component|read	☑	☑	
		inst|reg_32_generic_component|reset_n	☑	☑	
		inst|reg_32_generic_component|write	☑	☑	
		reg32	☑	☑	AND
		inst|reg_32_generic_component|Q_export[31..0]	☑	☑	XXXXXXXXh
		inst|reg_32_generic_component|byteenable[3..0]	☑	☑	Xh
		inst|reg_32_generic_component|from_reg[31..0]	☑	☑	XXXXXXXXh
		...g_32_generic_component|local_byteenable[3..0]	☑	☑	Xh
		inst|reg_32_generic_component|readdata[31..0]	☑	☑	XXXXXXXXh
		inst|reg_32_generic_component|writedata[31..0]	☑	☑	XXXXXXXXh
		Nios	☑	☑	AND
		inst|niosii_data_master_address[17..0]	☑	☑	XXXXXh
		inst|niosii_data_master_byteenable[3..0]	☑	☑	Xh
		inst|niosii_data_master_readdata[31..0]	☑	☑	XXXXXXXXh
		inst|niosii_data_master_writedata[31..0]	☑	☑	XXXXXXXXh
		inst|niosii_instruction_master_address[17..0]	☑	☑	XXXXXh
		inst|niosii_instruction_master_readdata[31..0]	☑	☑	XXXXXXXXh
		inst|niosii_irq_irq[31..0]	☑	☑	XXXXXXXXh

3. Set the clock and the sample depth



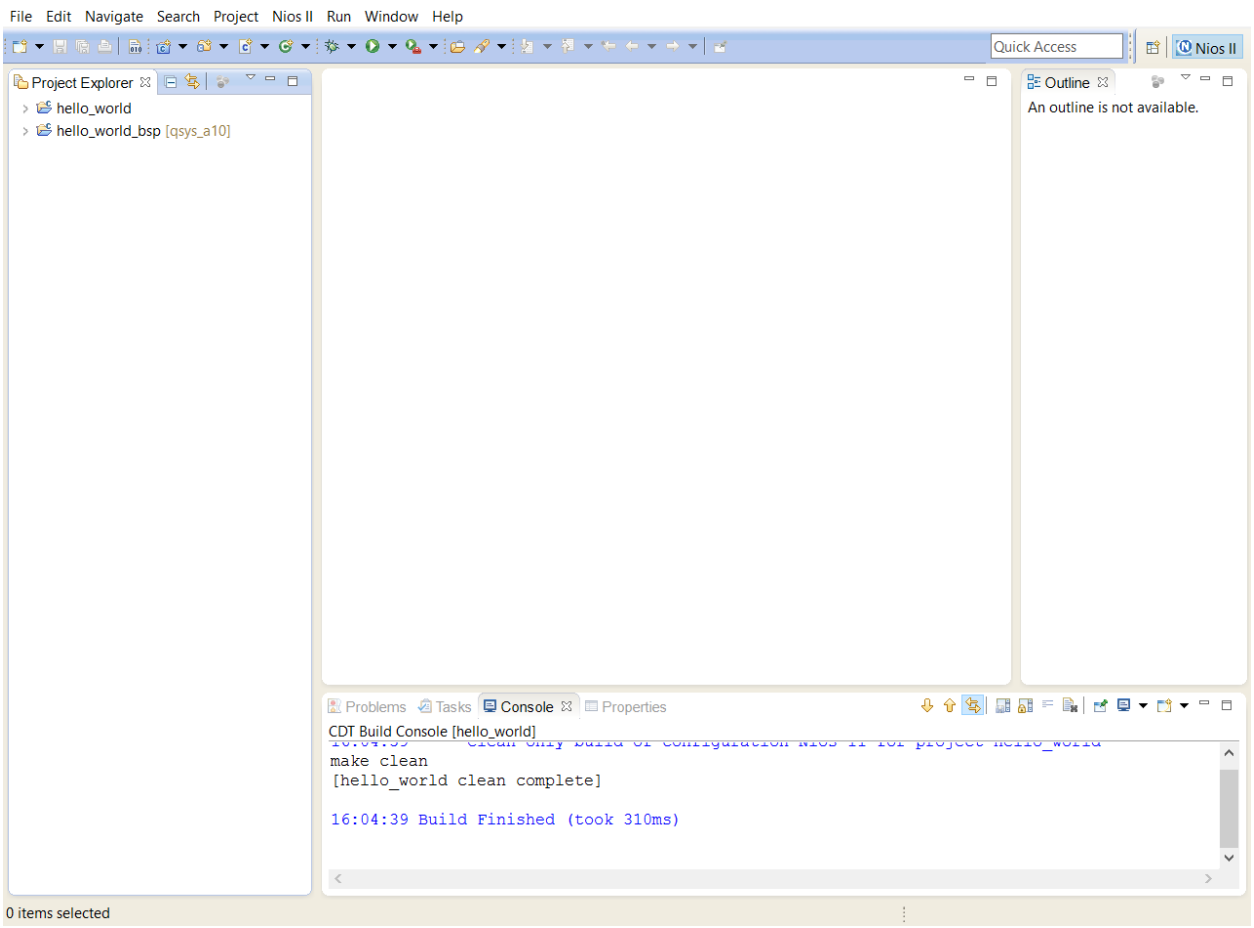
4. Click “Full Compilation” to compile the whole Quartus Prime Pro project.

Programming .sof into Arria 10 SoC development kit

1. Open the Quartus Prime Pro Programmer.
2. Connect the USB Blaster from the Arria 10 SoC development kit to the host machine.
3. Click “Auto Detect” to detect the jtag chain and the device.
4. Add in the .sof into the Programmer and check the “Program/Configure” checkbox.
5. Click “Start” to program the .sof into the board.
6. Close the Programmer after successfully program the .sof.

Building Software Design in Nios II Software Build Tools

1. Go to “Tools” and click on “Nios II Software Build Tools for Eclipse” to open the GUI.
2. Create a workspace for the software project.
3. Go to “File” and then to “New” and click on “Nios II Application and BSP from Template”. This will create the application and BSP project for you. You can choose the existing available template to use in your design.
4. Enter the “SOPC Information File Name” and it will automatically detect the processor which is available in your design.
5. Give your Application Project a name. Then click “Finish”. This will automatically create an application with the name provided and a BSP linked together with the application project that has been created.
6. We will select the “Hello World” template from the given Project Template.



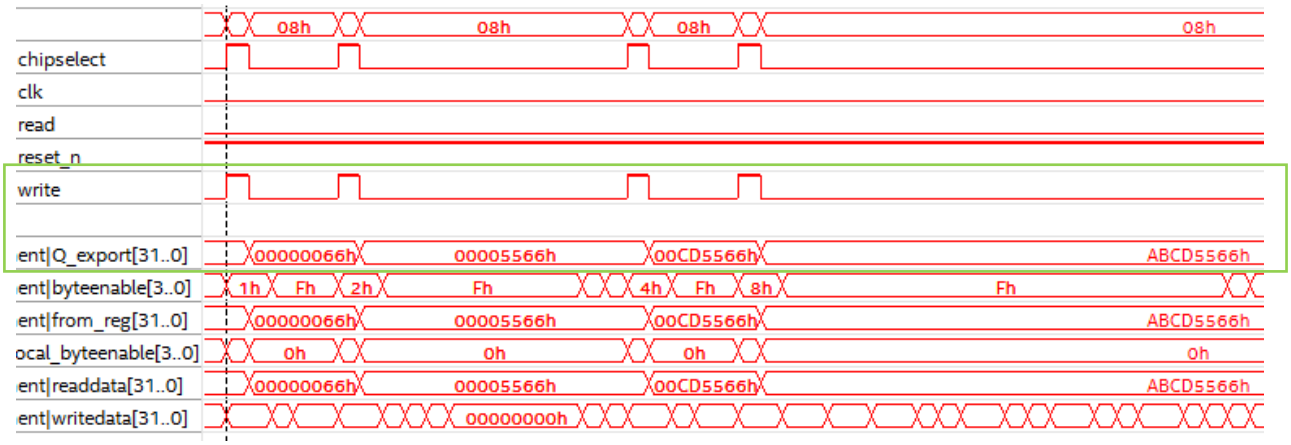
7. Now, we will modify the “hello_world.c” file in order the programming files contains the command to instruct Nios II to write some value into reg32 register created using Generic component. The simple code to write data into reg 32 is shown below.

```
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <io.h>

int main()
{
int reg1;

printf("Test on reg32\n");

printf("Writing to reg32 generic component register...\n");
IOWR_8DIRECT(REG_32_GENERIC_COMPONENT_BASE,0,0x66);
IOWR_8DIRECT(REG_32_GENERIC_COMPONENT_BASE,1,0x55);
IOWR_8DIRECT(REG_32_GENERIC_COMPONENT_BASE,2,0xCD);
IOWR_8DIRECT(REG_32_GENERIC_COMPONENT_BASE,3,0xAB);
printf("Done writing 0xABCD5566 into reg32\n\n");


printf("Reading from reg32 generic component register...\n");
reg1=IORD(REG_32_GENERIC_COMPONENT_BASE,0);
printf("Done\n\n");
printf("data read reg1=%x \n",reg1);


return 0;
}
```

8. Right click on the application project and click “Build Project”. When it is successfully built, it will produce the .elf file in the application folder.

Verification of design functionality using Signal Tap

1. Open Signal Tap Logic Analyzer and click run analysis.
2. Go to Nios II software build tool, download the elf file to the board by right click on the application project and navigate to “Run As” and click on “Nios II Hardware”.
3. You will see Signal Tap captured the value of the register. The write signal is asserted for four times which means that the reg32 is written for four times. Data is written correctly into the reg32 custom component.


08h
08h
08h
08h
chipselect
clk
read
reset_n
write
ent|Q_export[31..0]
00000066h
00005566h
00CD5566h
ABCD5566h
ent|byteenable[3..0]
1h
Fh
2h
Fh
4h
Fh
8h
Fh
ent|from_reg[31..0]
00000066h
00005566h
00CD5566h
ABCD5566h
ocal_byteenable[3..0]
0h
0h
0h
0h
ent|readdata[31..0]
00000066h
00005566h
00CD5566h
ABCD5566h
ent|writedata[31..0]
00000000h