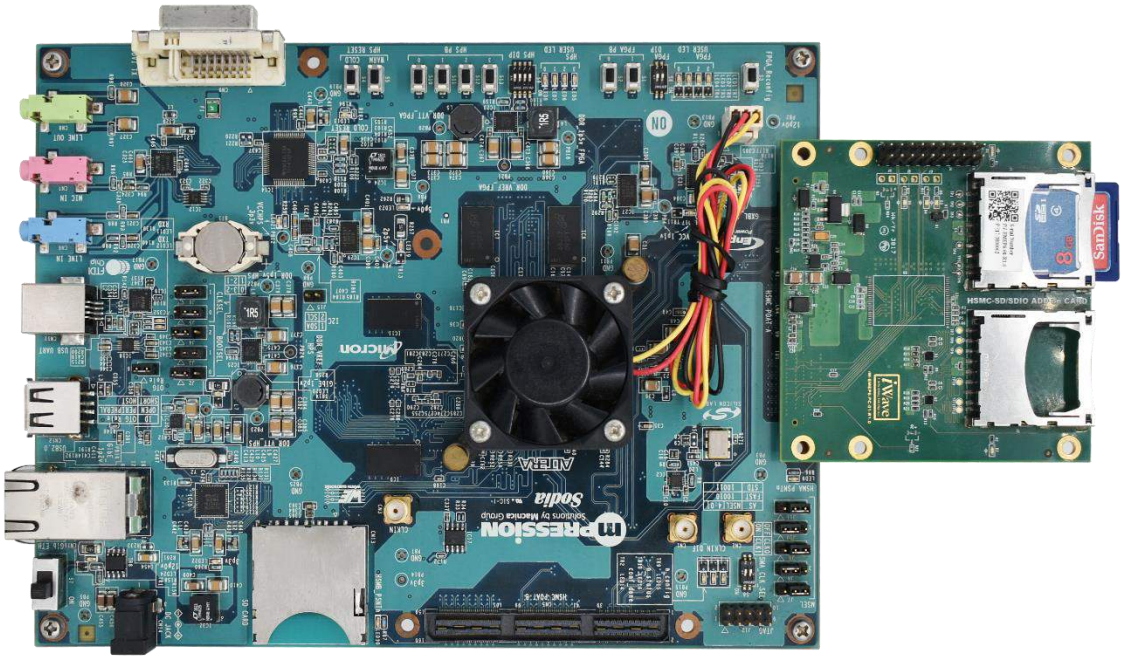
# SD/SDIO Host Controller 3.0 IP Integration Manual



# Table of Content

# List Of Figures

# List Of Tables

# 1 Introduction

## 1.1 Purpose

The purpose of this document is to describe SD/SDIO Host Controller IP Integration details

## 1.2 Scope

This document describes the how to integrate the SD/SDIO Host Controller IP. This documentcontains information about IP interface and integration details.

## 1.3 Reference Document

- SD Physical Specification Version 3.0

## 1.4 Overview

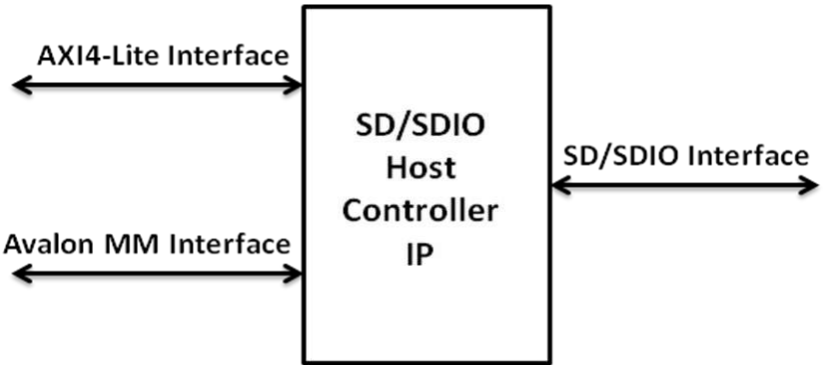



**Figure 1: SD/SDIO Host Controller IP Block Diagram**

SD/SDIO Host Controller defines a standard register set to control SD memory card/SDIO device.

## 1.5 Acronyms and Abbreviations

**Table 1: Acronyms & Abbreviations**

| Term | Meaning |
|---|---|
| ADMA | Advanced Direct Memory Access |
| FPGA | Field Programmable Gate Array |
| SD | Secure Digital |
| SDIO | Secure Digital Input Output |
| HPS | Hard Processor System |
| BFM | Bus Functional Model |
| DUT | Design Under Test |
| HDL | Hardware Description Language |

# 2 IP Component Instantiation

## 2.1 Adding .QXP file and creating the component

### 2.1.1 Steps to add sd_host_controller.qxp file into the design

Step #1: Copy the provided sd_host_controller.qxp file into the working project directory.
Step #2: Create a top module file containing the module instantiation for the SD host controller.Step #3: Open Qsys and in the IP catalog, click on the new component to create the SD host controller component.

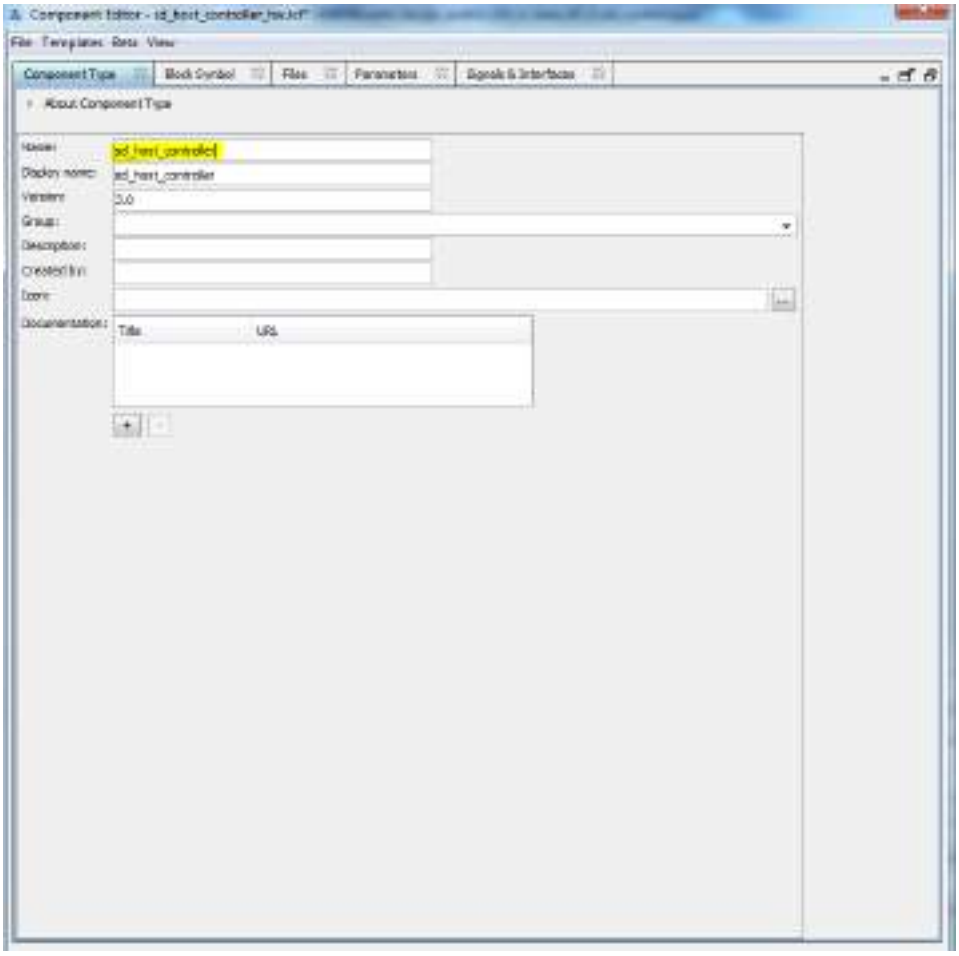



**Figure 2: Step #3 Creating Component**

Step #4: Add the top module file (created for this component) in the synthesis files menu asshown in the figure below.

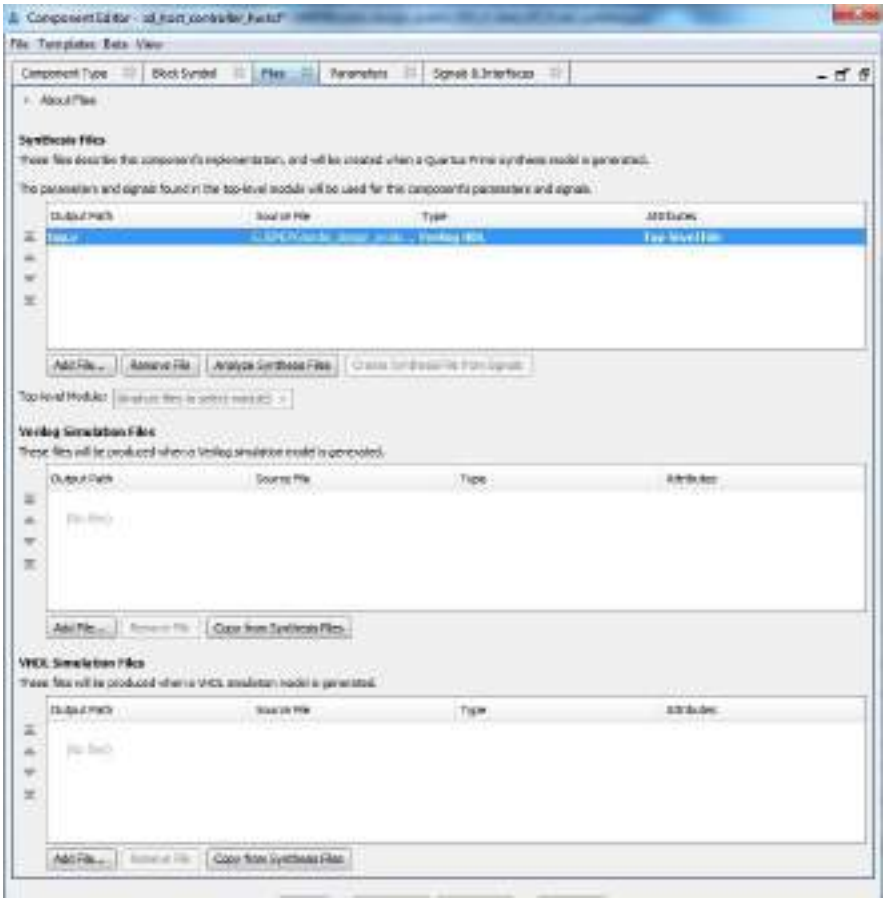


**Figure 3: Step #4 Creating Component**

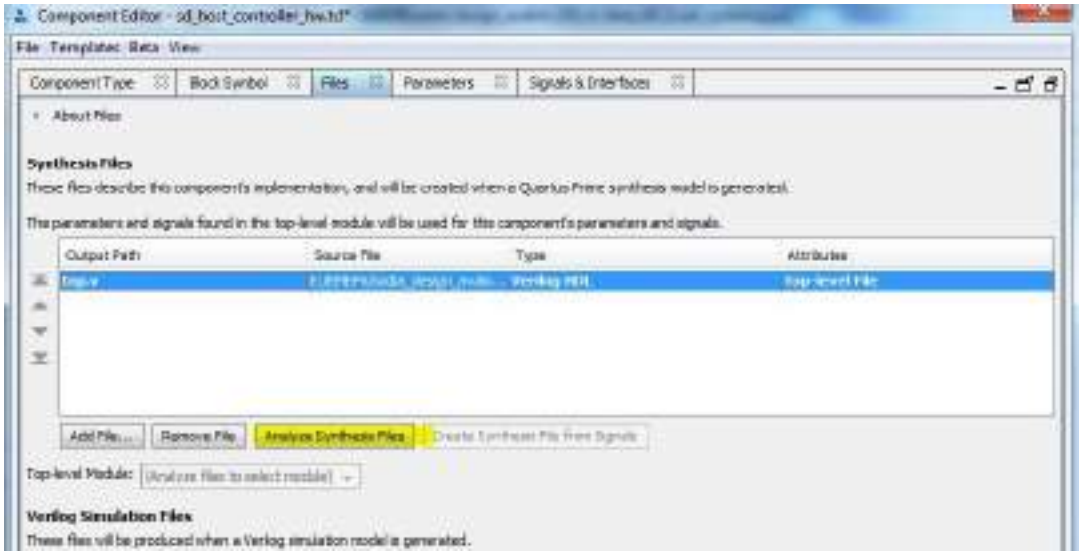



**Figure 4: Step #5 Creating Component**

Step #5: Click on analyze synthesis files

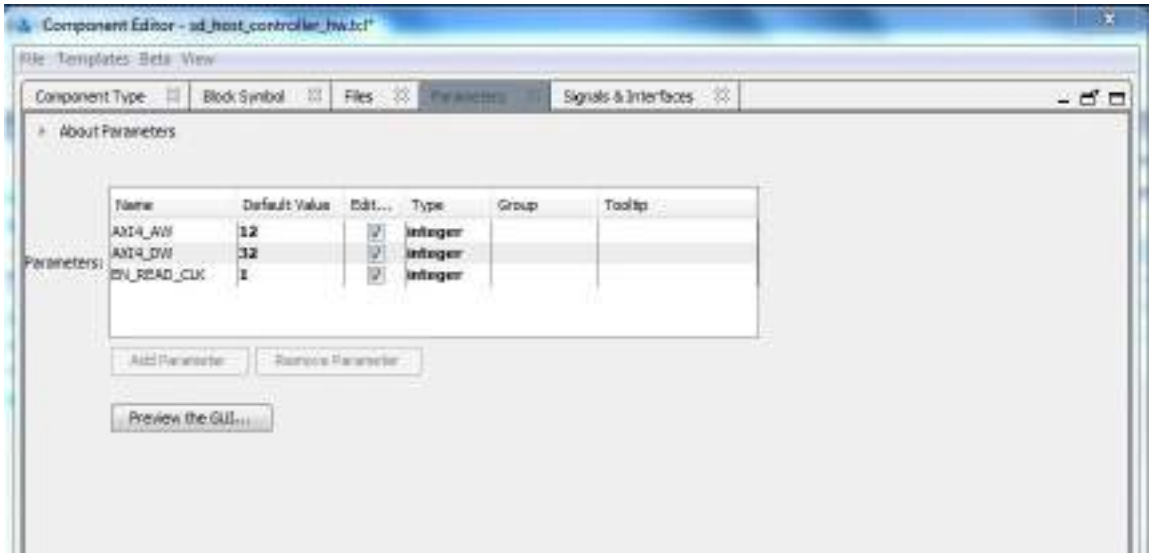



**Figure 5: Step #6 Creating Component**

Step #6: Ensure that analysis will auto detect the parameters defined in the top module file.

Step #7: In the signals and interface list, add the required interfaces and signals.
To add aninterface, click on <<*add interface*>> and select required interface.

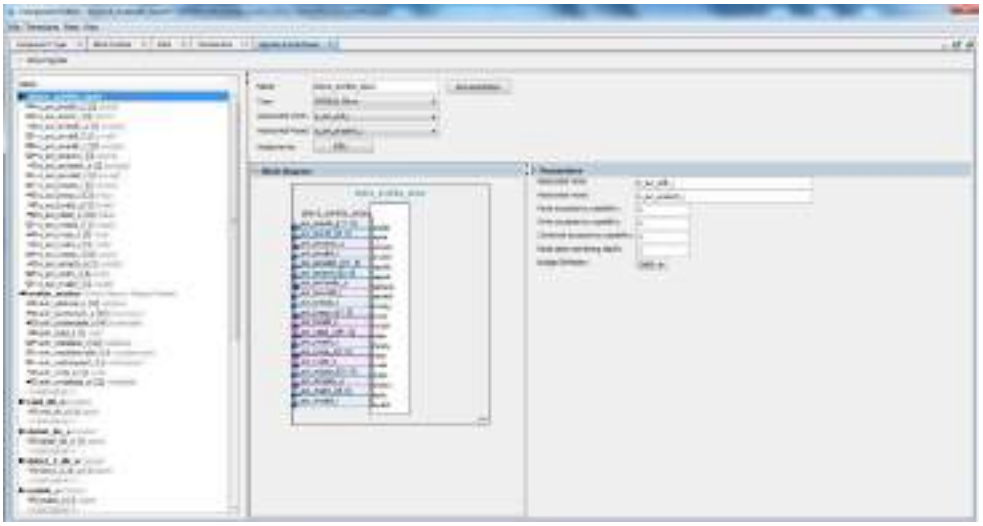


**Figure 6: Step #7 Creating Component**

Step #8: Also, add and match the necessary interface signals with its width.
Ensure that the Avalon master & AXI4 Lite slave interface and their signals are properly defined.

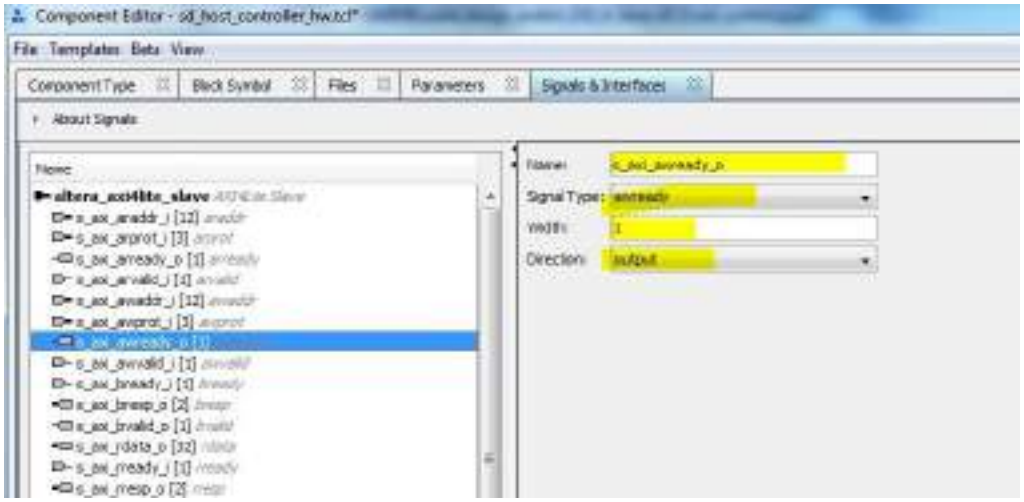



**Figure 7: Step #8 Creating Component**

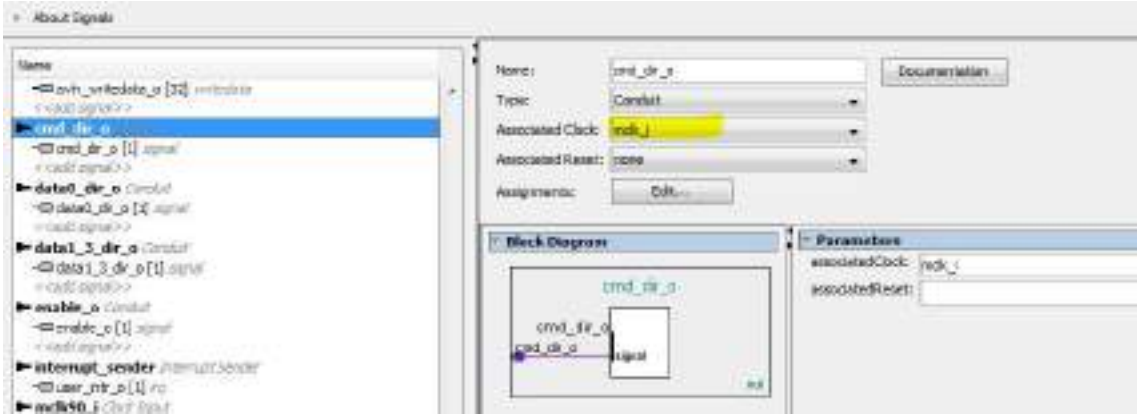



**Figure 8: Step #9 Creating Component**

Step #9: Add the all the SD signals interface signals. Click on <<*add interface*>> and select conduit type. Rename it with required SD signal name as per top module. Associate with mclk_iclock. For SD signals other than conduit type, select clock input/output, reset input, and interruptsender types for respective clock, reset & interrupt signals. Ensure the SD port signals are properly defined as in the top module file.

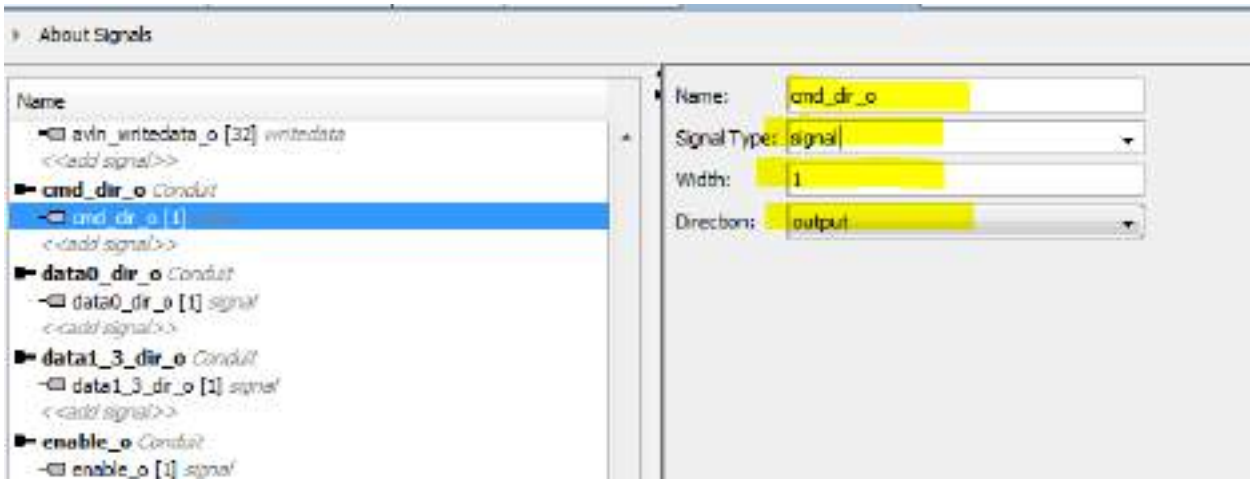



**Figure 9: Step #10 Creating Component**

Step #10: Further, click on <<add signal>> for the added conduit and rename it with SD signalname and set the proper settings i.e., type, width and direction.

Step #11: After all the above steps, save and click finish. You can now see the new componentie, sd_host_controller in the IP catalog.

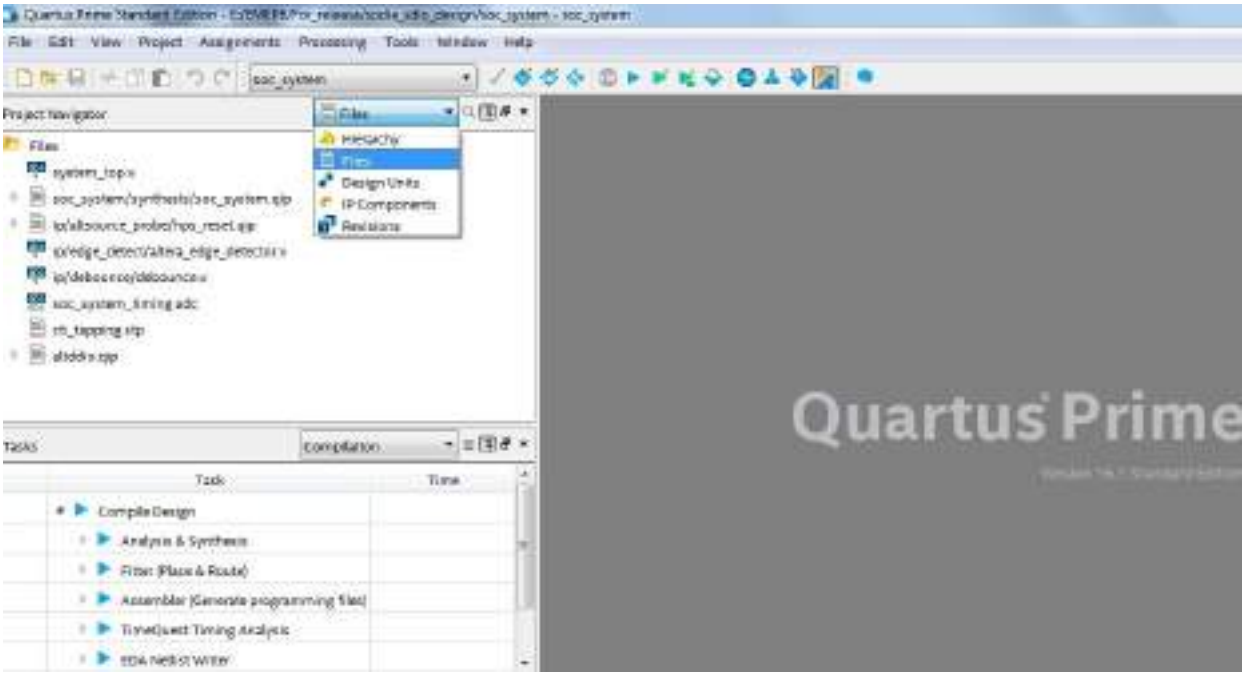



**Figure 10: Step #12 Adding the .qxp file into project navigator**

Step #12: In the project navigator, select file drop menu. Right click on File folder under ProjectNavigator to add new files into project.

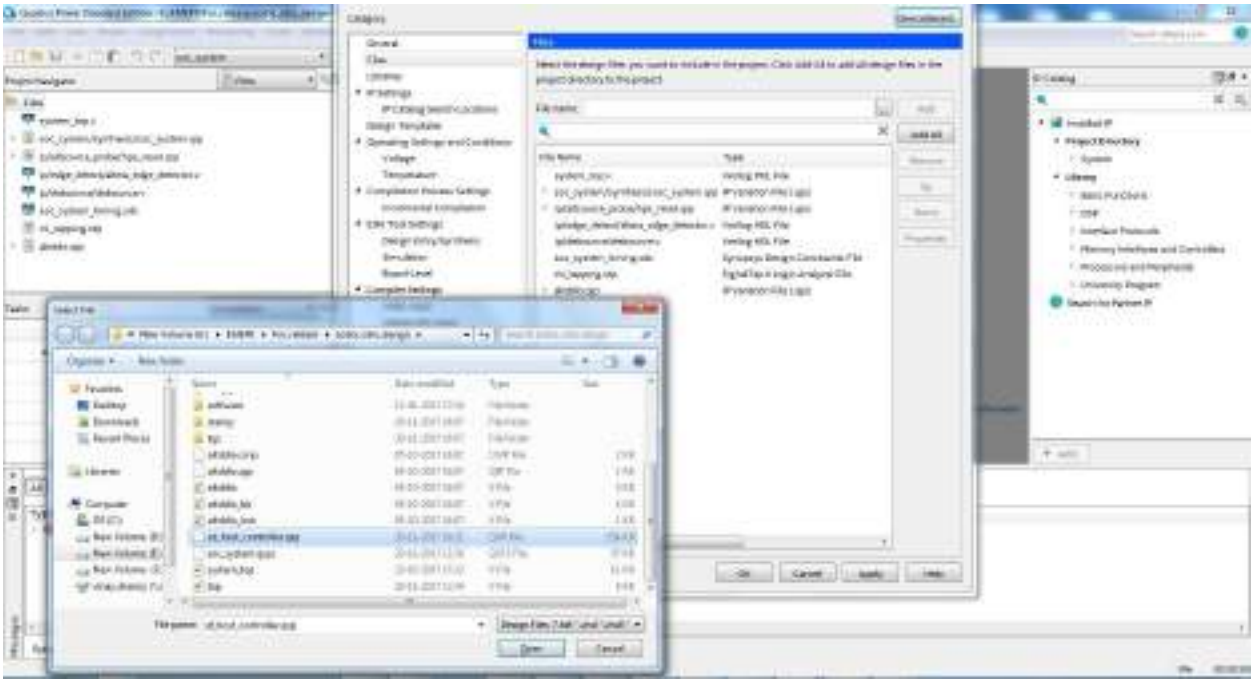


**Figure 11: Step #13 Adding sd_host_controller.qxp file into project**

Step #13: Add the copied sd_host_controller.qxp file from project directory into the project files.

## 2.2 Component Instantiation

iWave's SD/SDIO Host Controller IP component can be instantiated in Quartus Qsys in thefollowing manner.

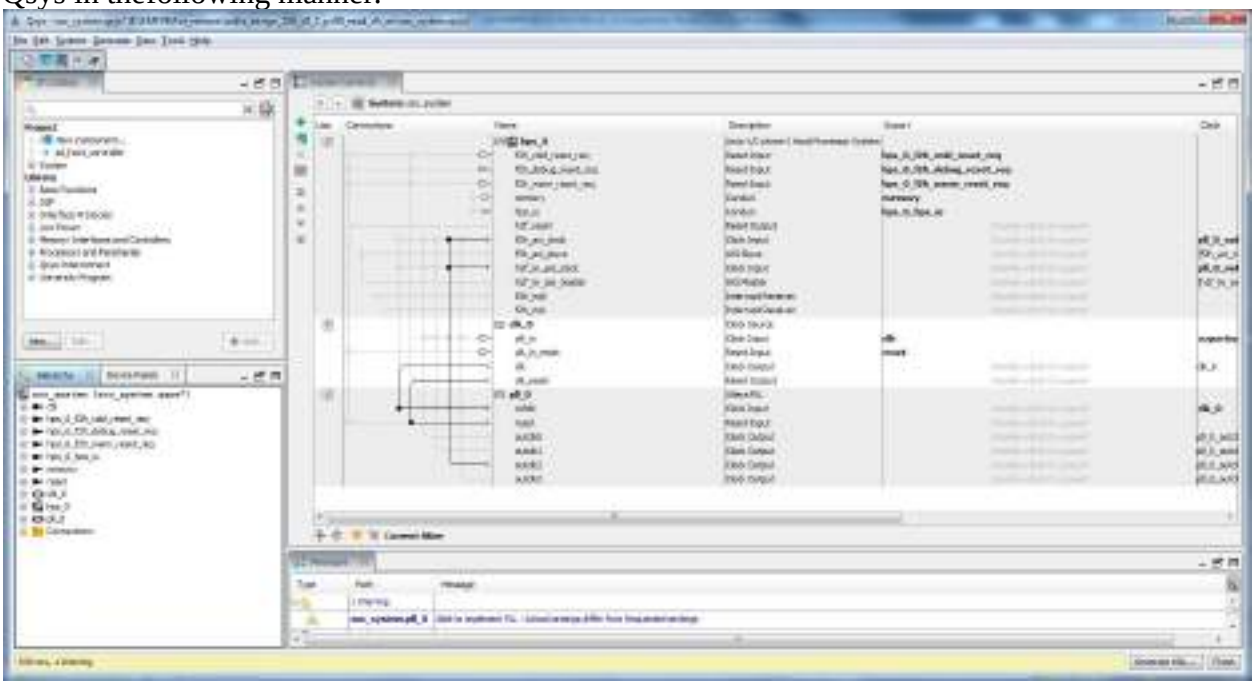


**Figure 12: Component Instantiation**

A Hard Processor System (HPS), clock source and altera pll component are already added in qsys as shown in the above figure. These components are added as part of design to illustrate IPintegration. These components can be added from IP catalog window.Refer the example designQsys for HPS, clock source and altera pll settings. iWave's SD/SDIO host controller IP can be seen under project at the top left of IP catalog.

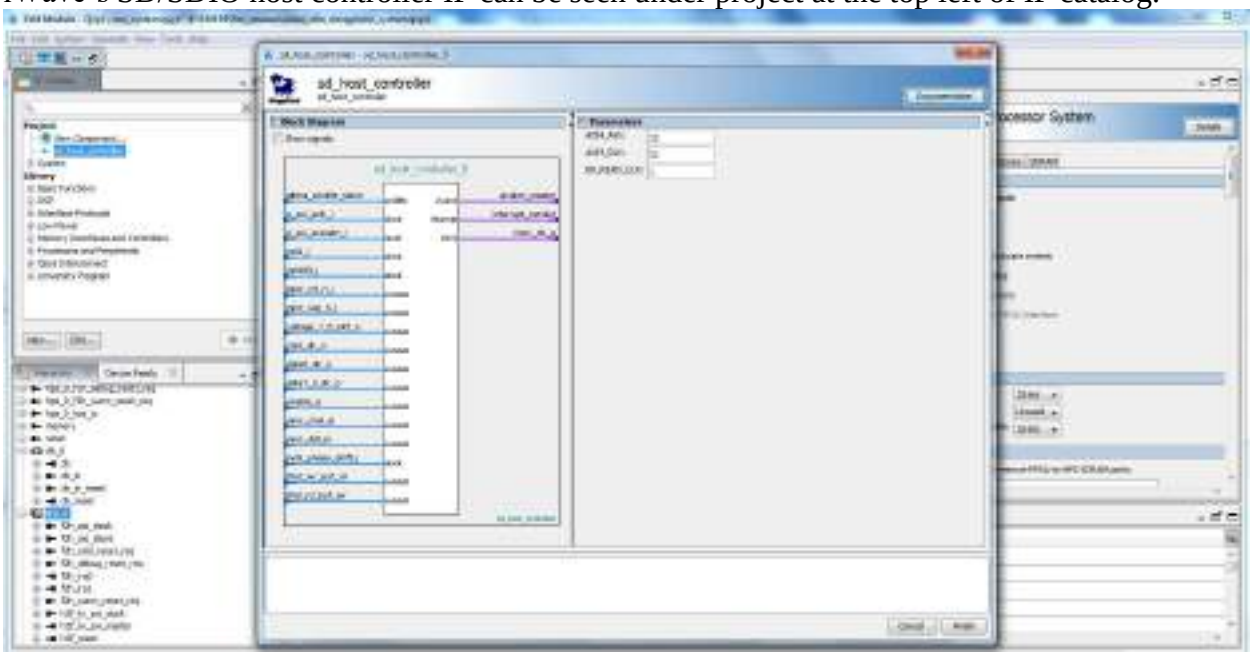



**Figure 13: Step #1 Component Instantiation**

Step #1: As shown the figure above, Double click on the SD host controller component in the IPcatalog. Set the parameters to default and proceed to finish.

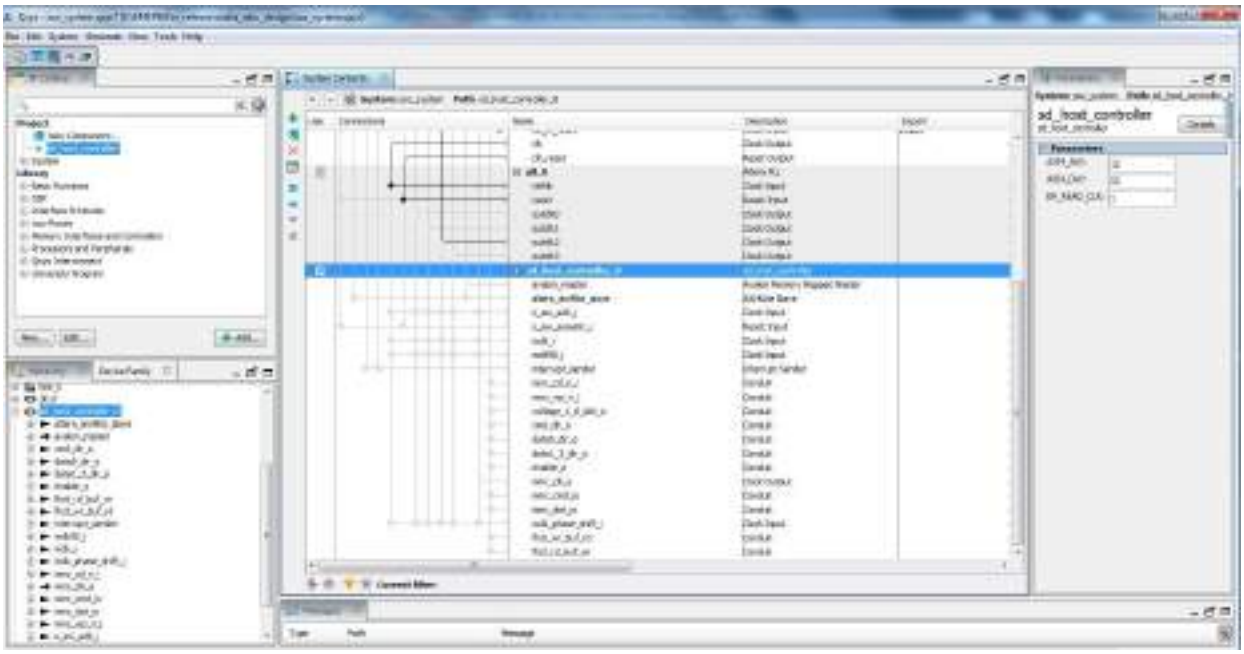



**Figure 14: Step #2 Component Instantiation**

Step #2: SD/SDIO host controller IP component is now included in the Qsys system console.

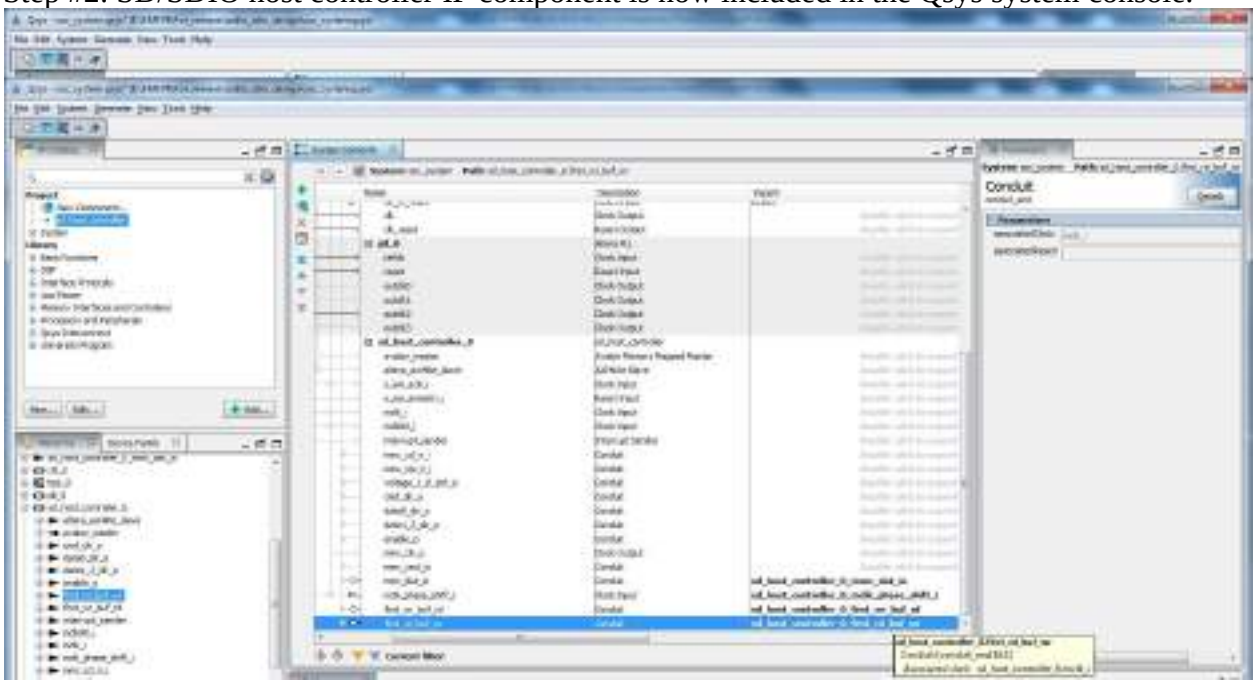


**Figure 15: Step #3 Component Instantiation**

Step #3: Double click on the Export column as shown in the figure to export the correspondingSD signals.

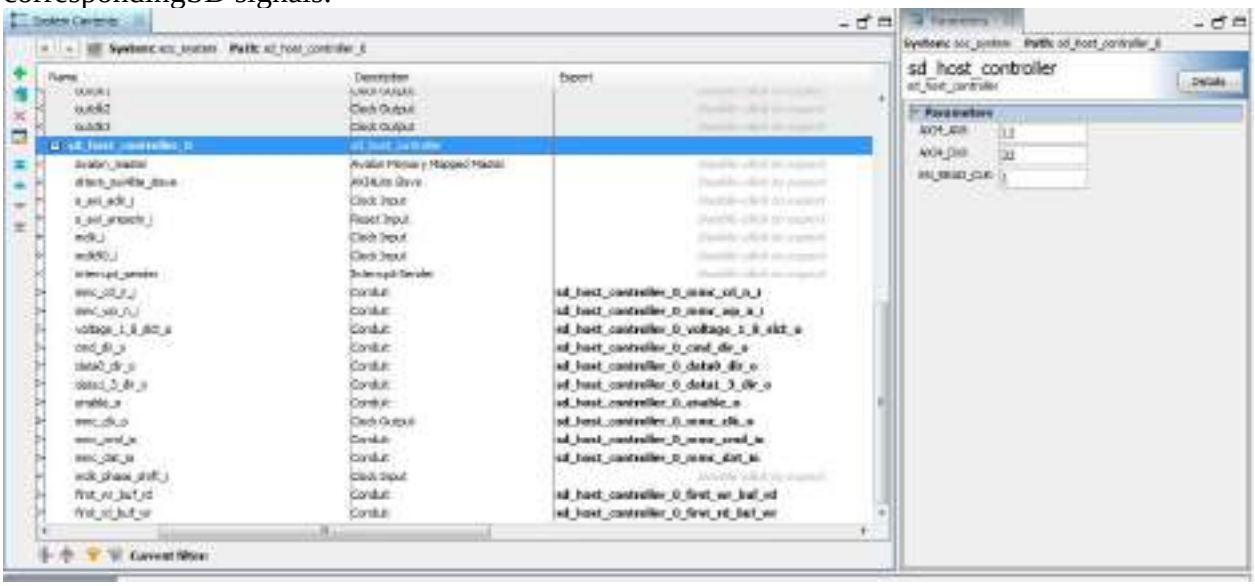


**Figure 16: Step #4 Component Instantiation**

Step #4: Similarly export mmc_clk_o, cmd_dir_o, data0_dir_o, data1_3_dir_o, enable_o, mmc_cd_n_i, mmc_cmd_io, mmc_dat_io and mmc_wp_n_i,first_wr_buf_rd, first_rd_buf_wrand voltage_1_8_slct_o signals.

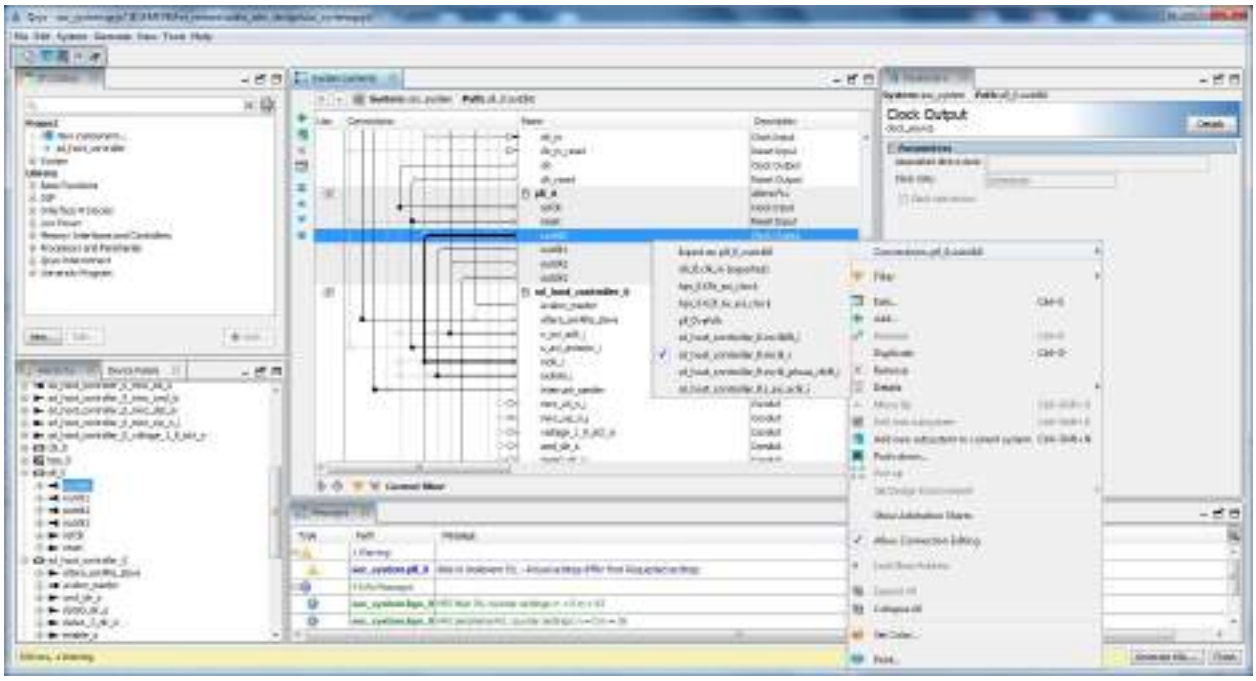


**Figure 17: Step #5 Component Instantiation**

Step #5: Right click on the pll_0 outclk0 and connect it to mclk_i as shown in the above figure.

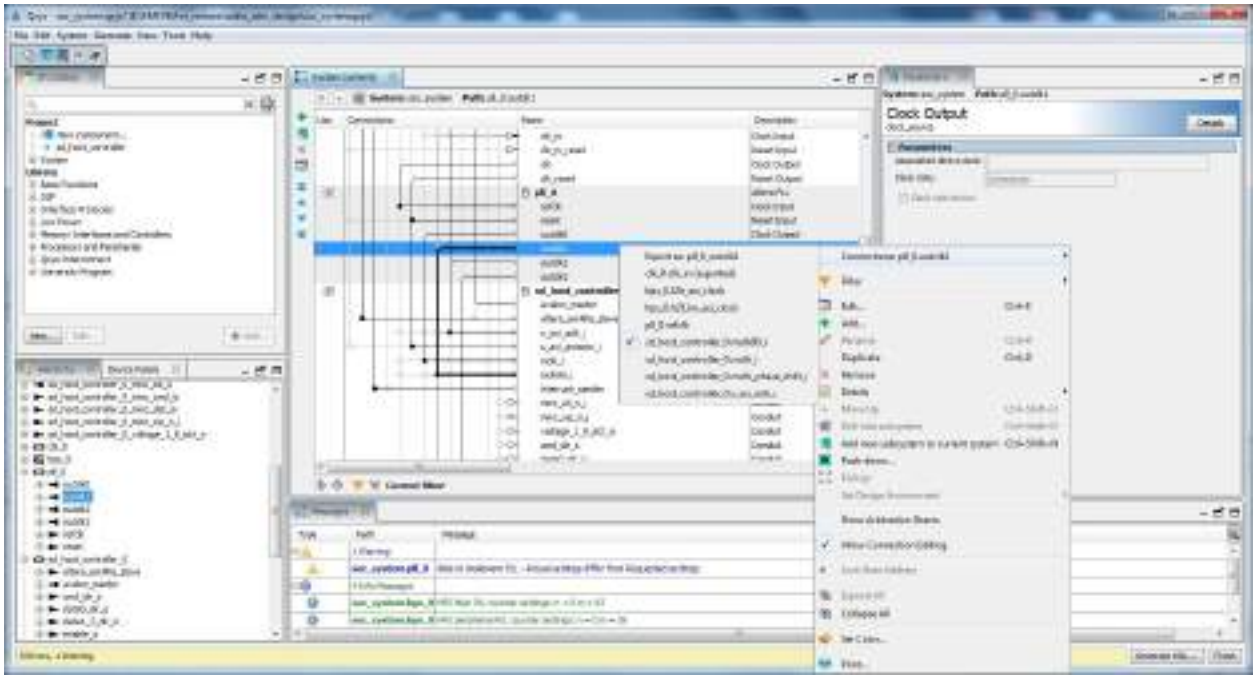


**Figure 18: Step #6 Component Instantiation**

Step #6: Connect outclk1 of the pll to mclk_90_i as shown in above figure.

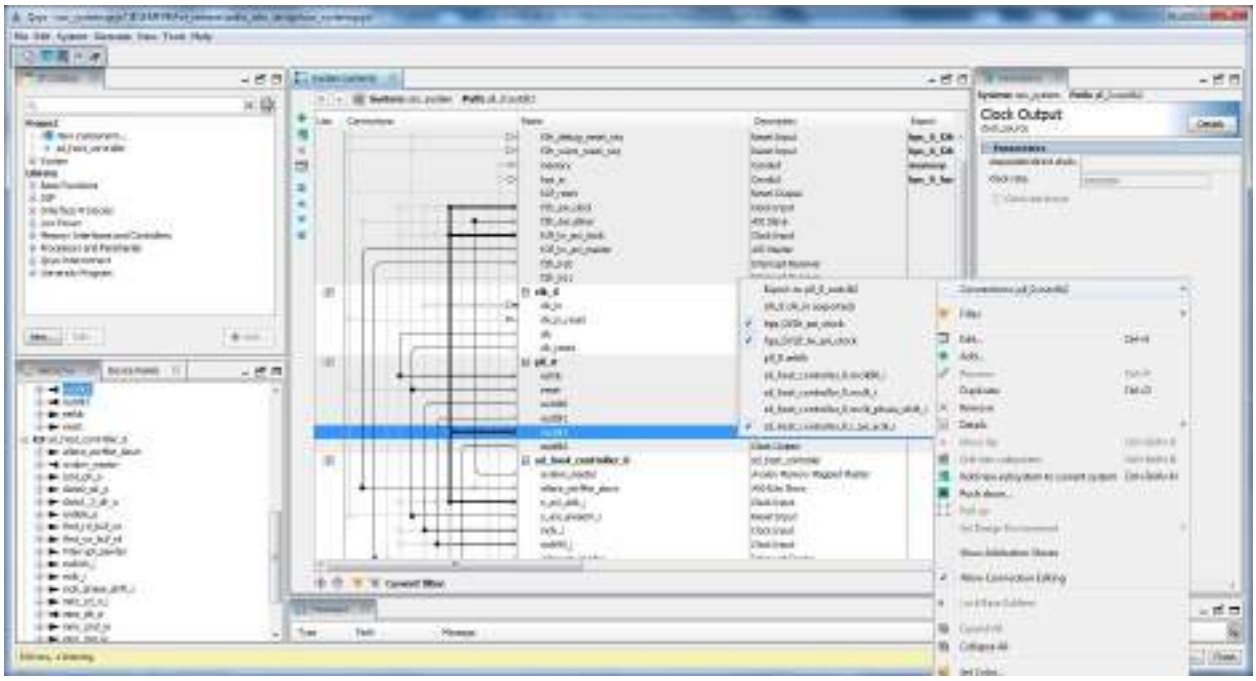



**Figure 19: Step #7 Component Instantiation**

Step #7: Do a similar connection for outclk2 by connecting it to s_axi_aclk_i (SD host controllersignal), f2h_axi_clock and h2f_lw_axi_clock (hps). Connect the outclk3 to mclk_phase_shift_i.

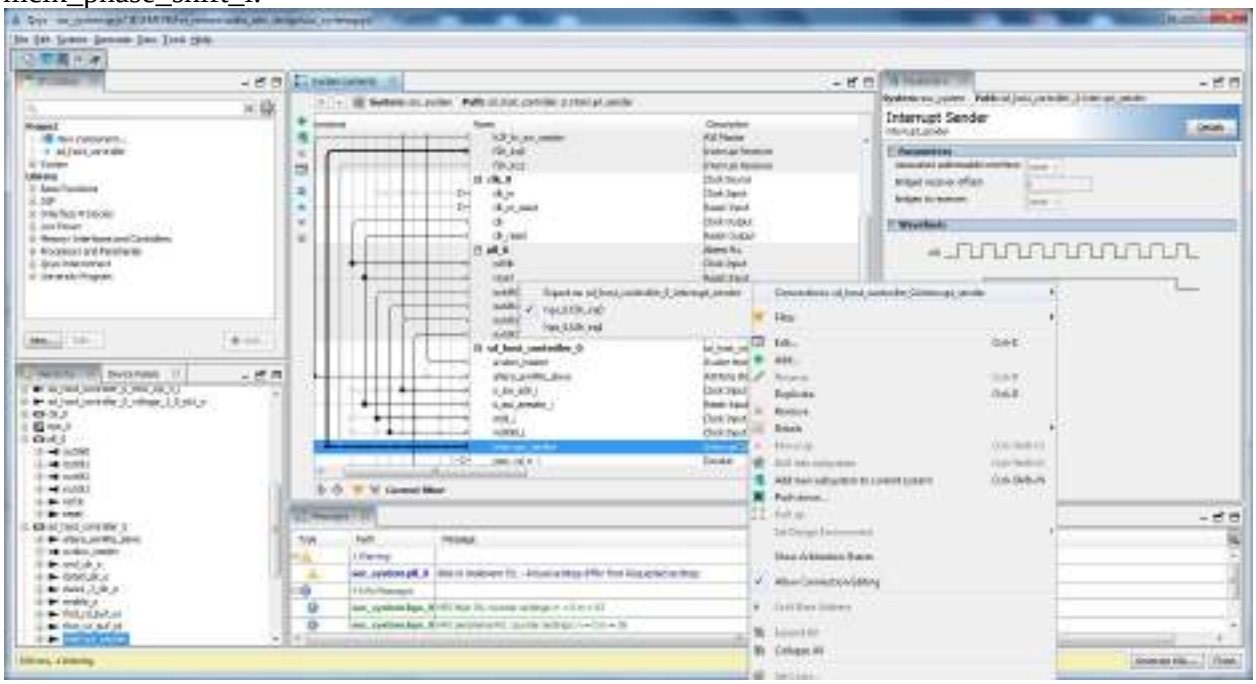



**Figure 20: Step #8 Component Instantiation**

Step #8: Right click on the interrupt_sender signal and connect it to f2h_irq0 of the hps.

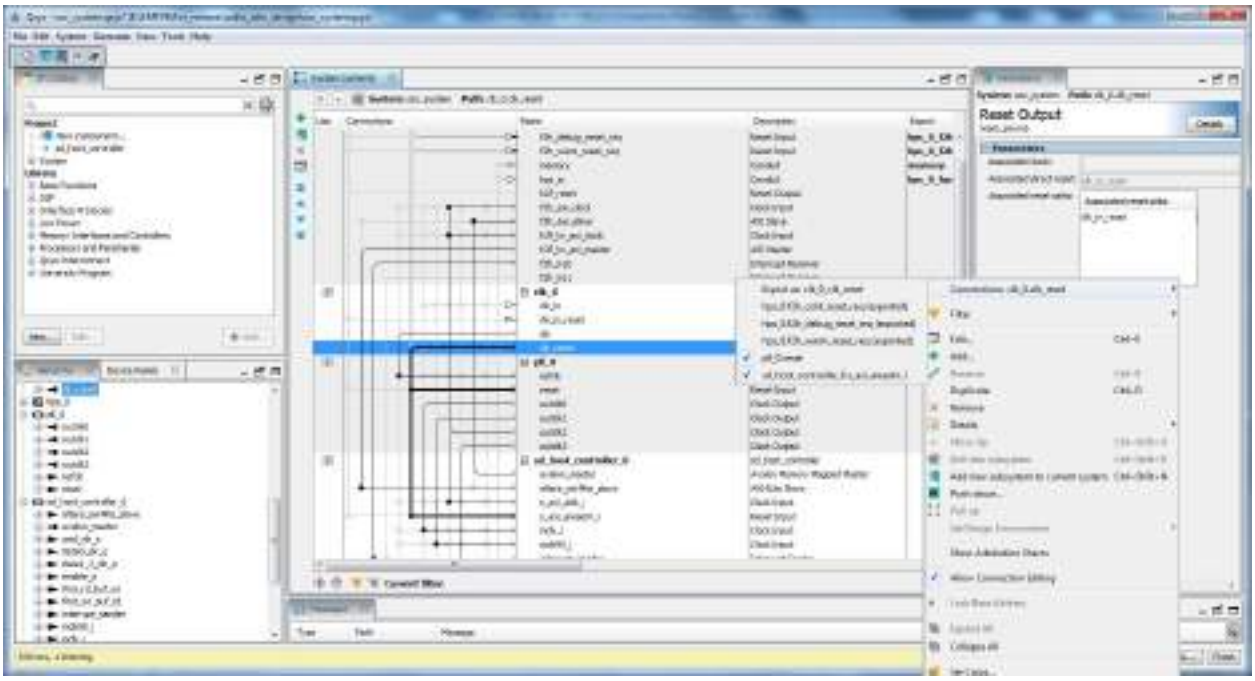



**Figure 21: Step #9 Component Instantiation**

Step #9: Connect clk_reset to pll_0.reset and s_axi_aresetn_i.

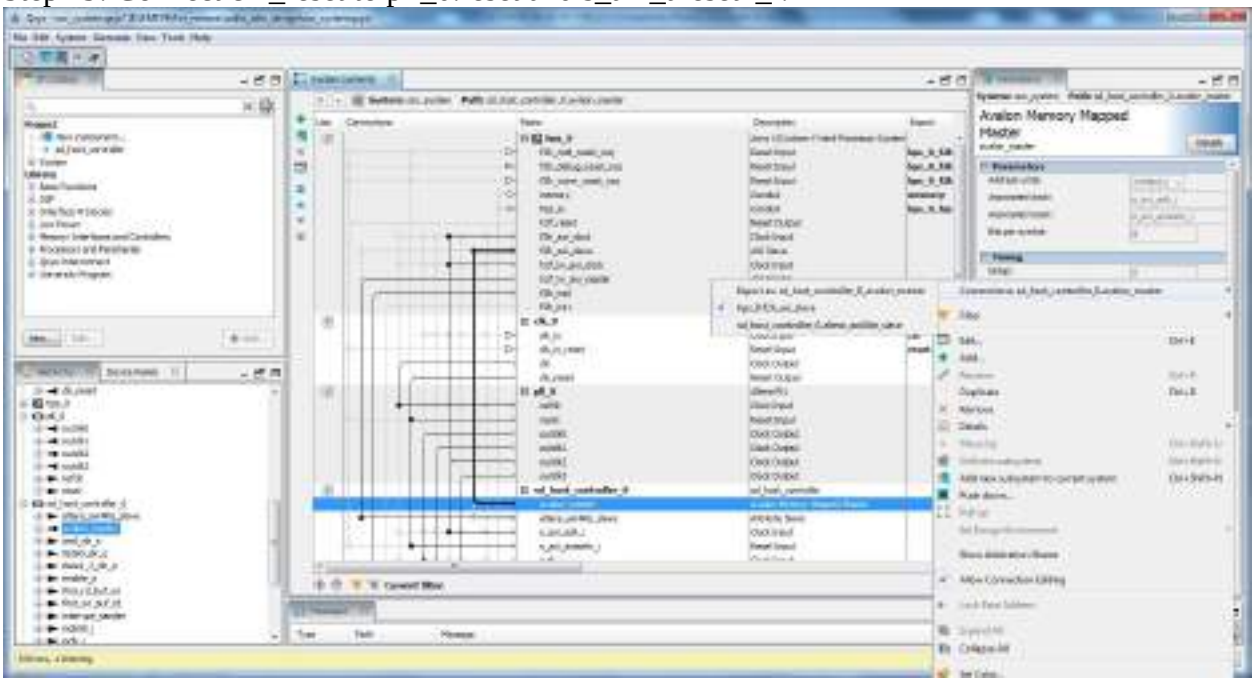



**Figure 22: Step #10 Component Instantiation**

Step #10: Avalon master has to be connected to f2h_axi_slave.

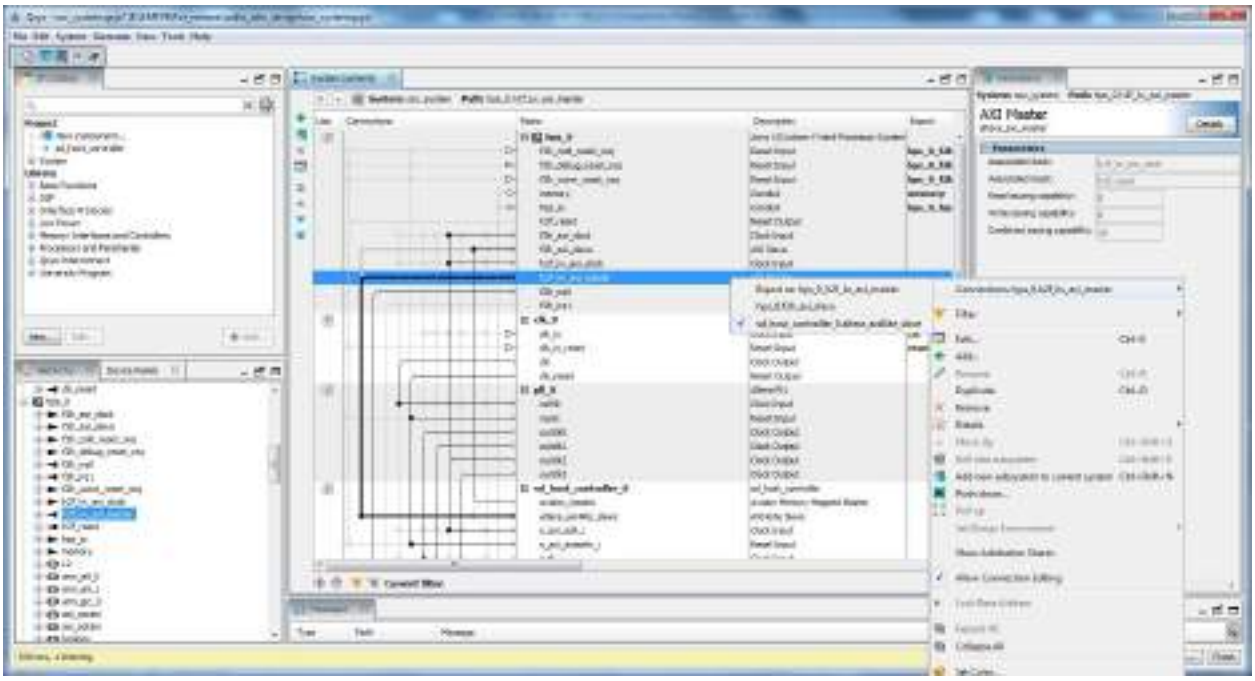


**Figure 23: Step #11 Component Instantiation**

Step #11: h2f_lw_axi_master is to be connected to altera_axi4lite_slave as shown in the figure.

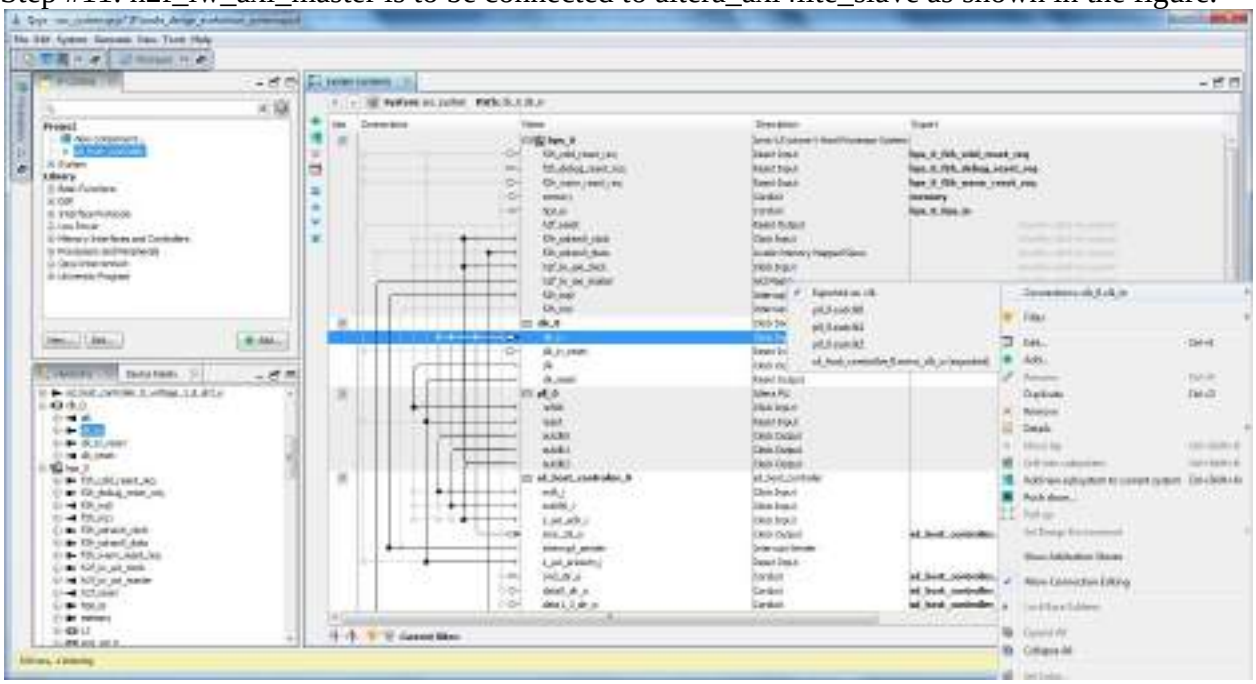


**Figure 24: Step #12 Component Instantiation**

Step #12: Export clk_in signal if you have not exported earlier.

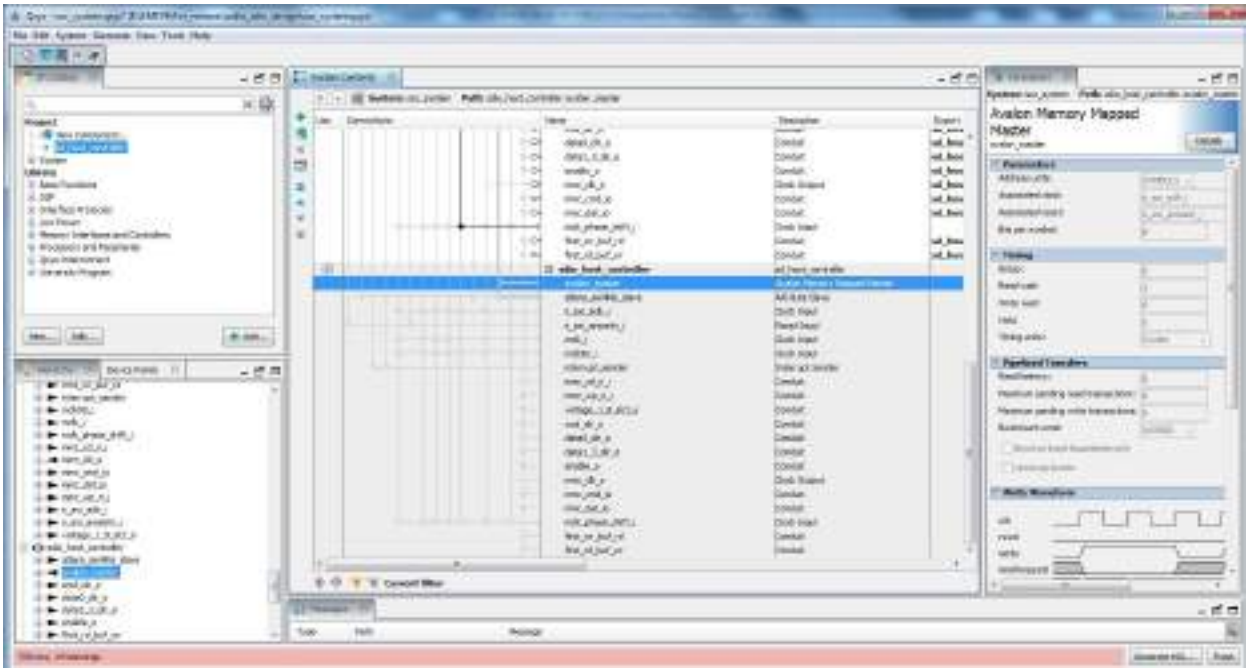


**Figure 25: Step #13 Component Instantiation**

Step #13: Add one more instance of SD host controller component. This is used as SDIO hostcontroller.
Rename (Cltr+R) the added component to sdio_host_controller.

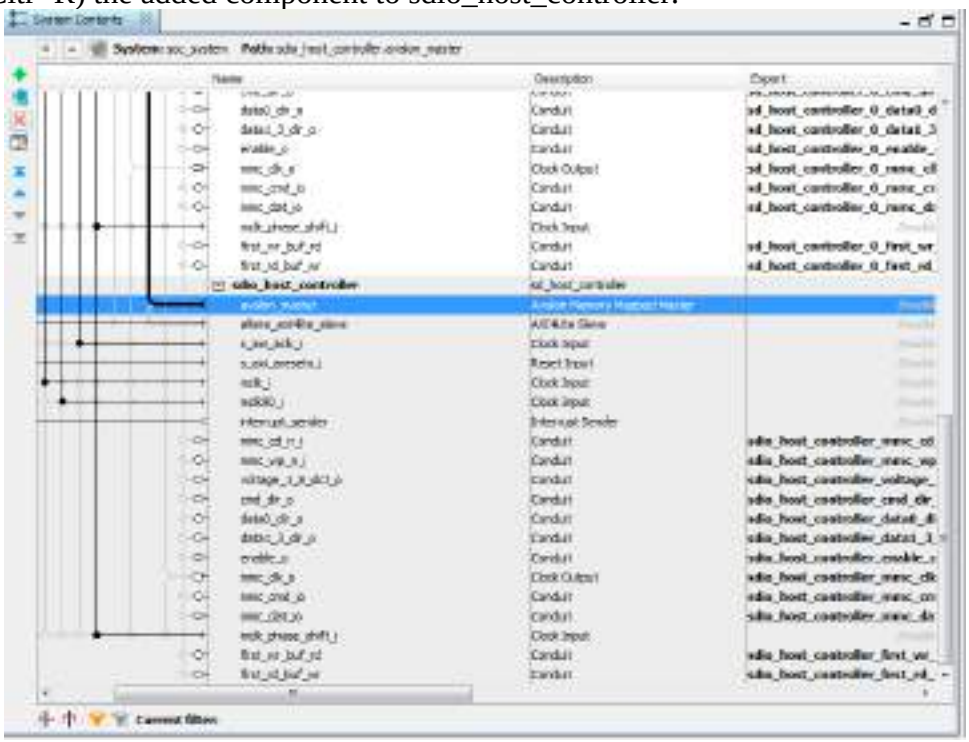


**Figure 26: Step #14 Component Instantiation**

Step #14: Make the necessary connections as described earlier for this SDIO host controllerinstance too.

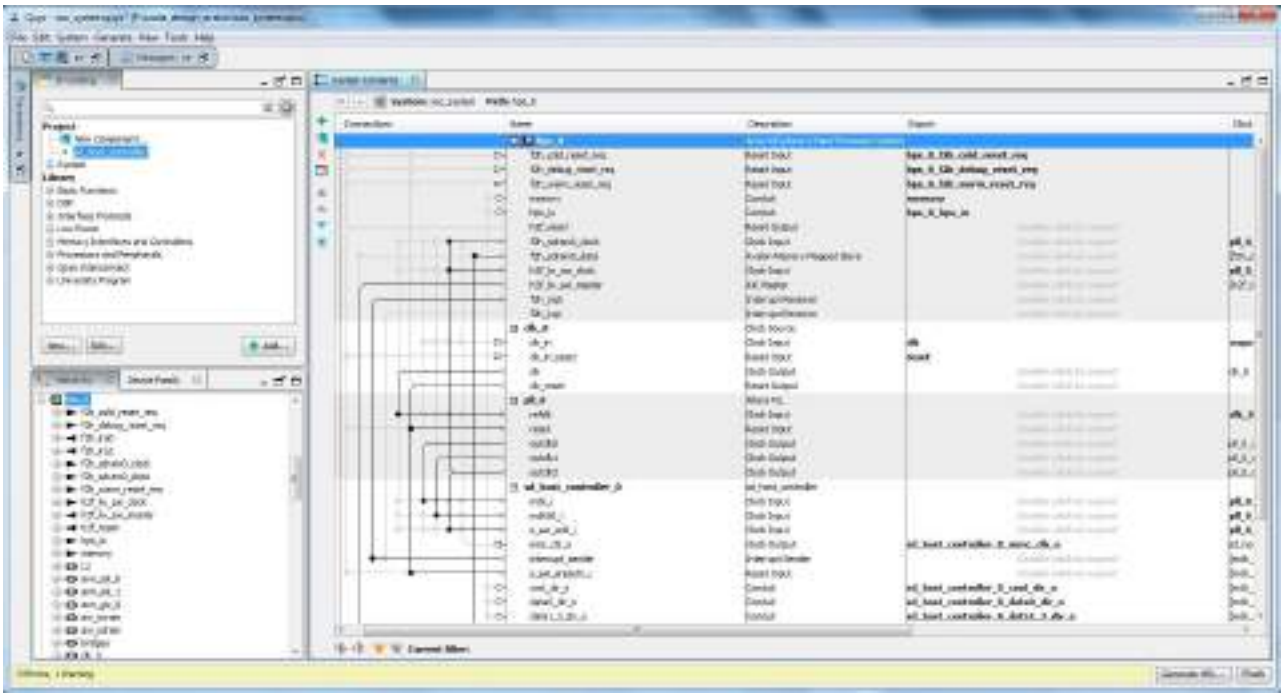


**Figure 27: Step #15 Component Instantiation**

Step #15: After completing the connections, save the design.

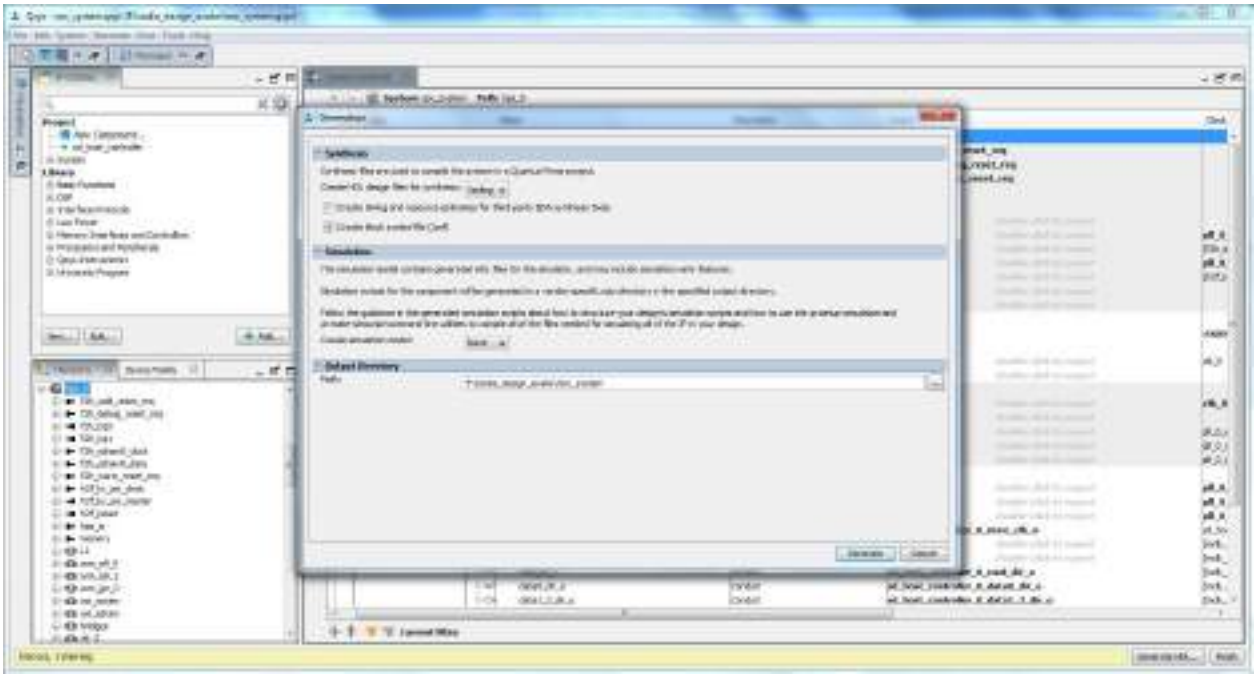


**Figure 28: Step #16 Component Instantiation**

Step #16: Now generate the HDL files for your design by clicking on Generate HDL icon. Keepthe selections as shown in the figure.

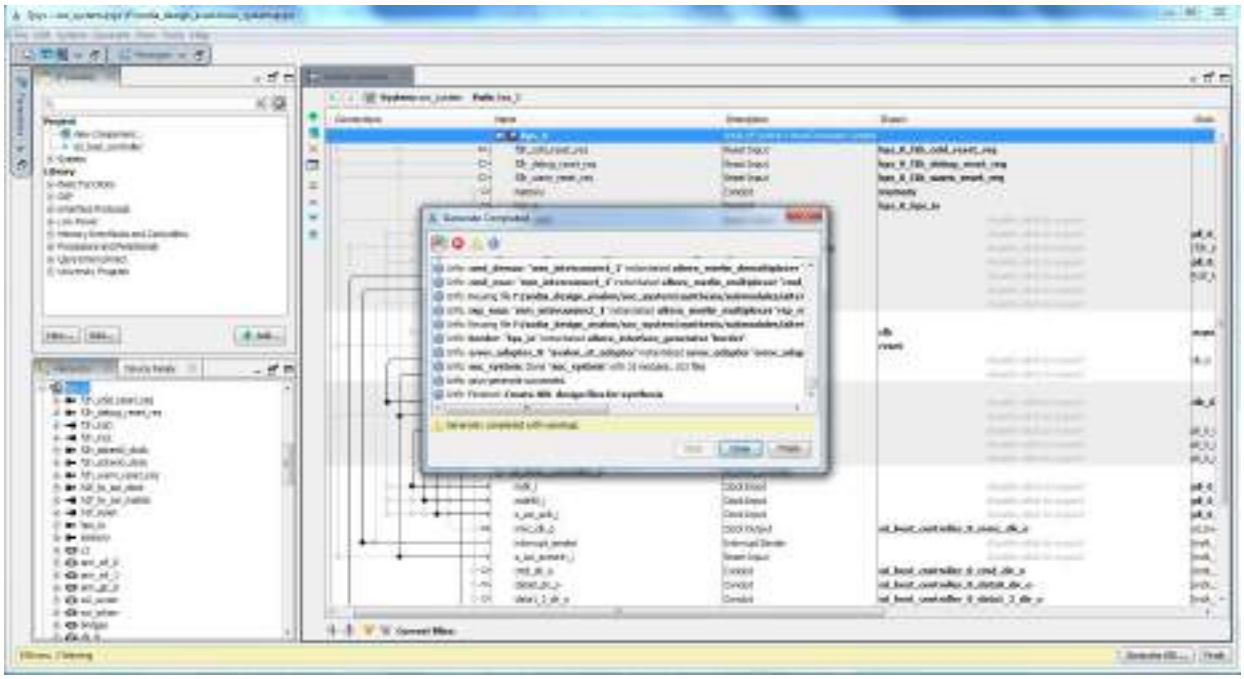


**Figure 29: Step #17 component instantiation**

Step #17: Once the hdl generation is completed without any errors, proceed to finish.

# 3 Implementation Details

## 3.1 Clock Domain

SD/SDIO Host Controller IP uses following clock domains SD Base clocks (mclk_i, mclk90_i and mclk_phase_shift_i) and host processor interface clock.

Here SD Base clocks mck90_i (150 deg. phase shifted) is used while sending data and mclk_phase_shift_i is feedback clock from the SD level translator IC which is used for receivingincoming data.

## 3.2 Constraints

The constraints given here are clock period constraints, create generated clock, set output delay and false path constraints. Refer timing constraint file soc_system_timing.sdc file in the exampledesign folder for more information.

### 3.2.1 Create clock constraint:

```
create_clock -period 10 [get_ports fpga_clk_100]

create_clock -name sd_clk_fb_virtual_clk -period
10.000ns create_clock -name
sdio_clk_fb_virtual_clk -period 10.000ns

create_clock -name "sd_clk_fb" -period 10.000ns [get_ports
{sd_clk_fb_i}] -add create_clock -name "sdio_clk_fb" -period 10.000ns
[get_ports {sdio_clk_fb_i}] -add
```

### 3.2.2 Create generated clock constraint:

This constraint is given for the DDIO out which generated the SD and SDIO clocks.

```
create_generated_clock -name sd_clk_out -source [get_pins
{soc_inst|sd_host_controller_0|sd_host_controller|mmc_host_inst_9|altddio_inst|ALTDDIO_O
U T_component|auto_generated|ddio_outa[0]|muxsel}] [get_ports
{sd_host_controller_0_mmc_clk_o_clk}]
create_generated_clock -name sdio_clk_out -
source [get_pins
{soc_inst|sdio_host_controller|sd_host_controller|mmc_host_inst_9|altddio_inst|ALTDDIO_O
U T_component|auto_generated|ddio_outa[0]|muxsel}] [get_ports
{sdio_host_controller_0_mmc_clk_o_clk}]
```

### 3.2.3 Output Delay Constraints:

Max output delay is defined by the setup time of the card which is 3ns for SDR50 mode minimum is defined by the hold time which is 0. 8ns.We are sending the data w.r.t 150-degreephase shifted clock so, these constraints are given w.r.t clock1 of the PLL.

```
set_output_delay -clock { sd_clk_out } -max 4.5 [get_ports
{sd_host_controller_0_mmc_cmd_io_new_signal
sd_host_controller_0_mmc_dat_io_new_signal[0]
```

sd_host_controller_0_mmc_dat_io_new_signal[1]
sd_host_controller_0_mmc_dat_io_new_signal[2]
sd_host_controller_0_mmc_dat_io_new_signal[3]}]

set_output_delay -clock { sdio_clk_out } -max 4.5 [get_ports
{sdio_host_controller_0_mmc_cmd_io_new_signal
sdio_host_controller_0_mmc_dat_io_new_signal[0]
sdio_host_controller_0_mmc_dat_io_new_signal[1]
sdio_host_controller_0_mmc_dat_io_new_signal[2]
sdio_host_controller_0_mmc_dat_io_new_signal[3]}]

set_output_delay -clock { sd_clk_out } -min -2.3 [get_ports
{sd_host_controller_0_mmc_cmd_io_new_signal
sd_host_controller_0_mmc_dat_io_new_signal[0]
sd_host_controller_0_mmc_dat_io_new_signal[1]
sd_host_controller_0_mmc_dat_io_new_signal[2]
sd_host_controller_0_mmc_dat_io_new_signal[3]}]

set_output_delay -clock { sdio_clk_out } -min -2.3 [get_ports
{sdio_host_controller_0_mmc_cmd_io_new_signal
sdio_host_controller_0_mmc_dat_io_new_signal[0]
sdio_host_controller_0_mmc_dat_io_new_signal[1]
sdio_host_controller_0_mmc_dat_io_new_signal[2]
sdio_host_controller_0_mmc_dat_io_new_signal[3]}]

### 3.2.4 Input Delay Constraints:

Input constraint are defined w.r.t to the virtual clock created based on the feedback clock whichis received from the SD level translator

set_input_delay -clock { sd_clk_fb_virtual_clk } -max -1.5 [get_ports
{sd_host_controller_0_mmc_cmd_io_new_signal
sd_host_controller_0_mmc_dat_io_new_signal[0]
sd_host_controller_0_mmc_dat_io_new_signal[1]
sd_host_controller_0_mmc_dat_io_new_signal[2]
sd_host_controller_0_mmc_dat_io_new_signal[3]}]

set_input_delay -clock { sdio_clk_fb_virtual_clk } -max -1.5 [get_ports
{sdio_host_controller_0_mmc_cmd_io_new_signal
sdio_host_controller_0_mmc_dat_io_new_signal[0]
sdio_host_controller_0_mmc_dat_io_new_signal[1]
sdio_host_controller_0_mmc_dat_io_new_signal[2]
sdio_host_controller_0_mmc_dat_io_new_signal[3]}]

set_input_delay -clock { sd_clk_fb_virtual_clk } -min 0.5 [get_ports
{sd_host_controller_0_mmc_cmd_io_new_signal
sd_host_controller_0_mmc_dat_io_new_signal[0]
sd_host_controller_0_mmc_dat_io_new_signal[1]
sd_host_controller_0_mmc_dat_io_new_signal[2]
sd_host_controller_0_mmc_dat_io_new_signal[3]}]

```
set_input_delay -clock { sdio_clk_fb_virtual_clk } -min 0.5 [get_ports
{sdio_host_controller_0_mmc_cmd_io_new_signal
sdio_host_controller_0_mmc_dat_io_new_signal[0]
sdio_host_controller_0_mmc_dat_io_new_signal[1]
sdio_host_controller_0_mmc_dat_io_new_signal[2]
sdio_host_controller_0_mmc_dat_io_new_signal[3]}]
```

### 3.2.5 False path constraints:

```
set_false_path -from [get_clocks
{soc_inst|pll_0|altera_pll_i|general[0].gpll~PLL_OUTPUT_COUNTER|divclk}] -to
[get_clocks
{soc_inst|pll_0|altera_pll_i|general[2].gpll~PLL_OUTPUT_COUNTER|divclk}]

set_false_path -from [get_clocks
{soc_inst|pll_0|altera_pll_i|general[2].gpll~PLL_OUTPUT_COUNTER|divclk}] -to
[get_clocks
{soc_inst|pll_0|altera_pll_i|general[0].gpll~PLL_OUTPUT_COUNTER|divclk}]

set_false_path -from [get_clocks
{soc_inst|pll_0|altera_pll_i|general[1].gpll~PLL_OUTPUT_COUNTER|divclk}] -to
[get_clocks
{soc_inst|pll_0|altera_pll_i|general[2].gpll~PLL_OUTPUT_COUNTER|divclk}]

set_false_path -from [get_clocks
{soc_inst|pll_0|altera_pll_i|general[2].gpll~PLL_OUTPUT_COUNTER|divclk}] -to
[get_clocks
{soc_inst|pll_0|altera_pll_i|general[1].gpll~PLL_OUTPUT_COUNTER|divclk}]

set_false_path -from [get_clocks {sd_clk_fb}] -to [get_clocks
{soc_inst|pll_0|altera_pll_i|general[1].gpll~PLL_OUTPUT_COUNTER|divclk}]

set_false_path -from [get_clocks
{soc_inst|pll_0|altera_pll_i|general[1].gpll~PLL_OUTPUT_COUNTER|divclk}] -to
[get_clocks
{sd_clk_fb}]

set_false_path -from [get_clocks {sdio_clk_fb}] -to [get_clocks
{soc_inst|pll_0|altera_pll_i|general[1].gpll~PLL_OUTPUT_COUNTER|divclk}]

set_false_path -from [get_clocks
{soc_inst|pll_0|altera_pll_i|general[1].gpll~PLL_OUTPUT_COUNTER|divclk}] -to
[get_clocks
{sdio_clk_fb}]

set_false_path -from [get_clocks {sd_clk_fb}] -to [get_clocks
{soc_inst|pll_0|altera_pll_i|general[2].gpll~PLL_OUTPUT_COUNTER|divclk}]

set_false_path -from [get_clocks
```

```
{soc_inst|pll_0|altera_pll_i|general[2].gpll~PLL_OUTPUT_COUNTER|divclk}] -to
[get_clocks
{sd_clk_fb}]

set_false_path -from [get_clocks {sdio_clk_fb}] -to [get_clocks
{soc_inst|pll_0|altera_pll_i|general[2].gpll~PLL_OUTPUT_COUNTER|divclk}]

set_false_path -from [get_clocks
{soc_inst|pll_0|altera_pll_i|general[2].gpll~PLL_OUTPUT_COUNTER|divclk}] -to
[get_clocks
{sdio_clk_fb}]
```

# 4 FPGA Implementation

## 4.1 Resource Utilization

The table below shows the resource utilization summary for Sodia Cyclone V FPGA device(5CSTFD6D5F31I7) for SD/SDIO Host Controller IP.

**Table 2: FPGA Resource Utilization for Sodia Cyclone V device**

| | |
|---|---|
| Logic utilization (in ALMs) | 1201 |
| Total registers | 2192 |
| Total block memory bits | 32928 |

# 5 Simulation

## 5.1 Simulation Environment

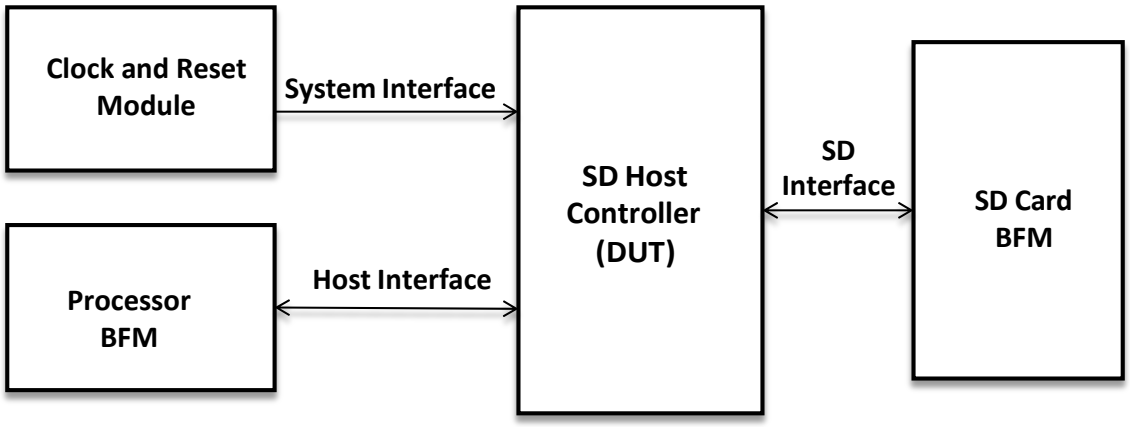



**Figure 30: Simulation Environment**

## 5.2 Simulation Environment Description

The Simulation Environment consists of the SD Host Controller (DUT), Processor BFM, Resetand Clock Module, and SD Card BFM.
Reset and Clock Module generates the entire clock required by DUT and the reset signal. It generates the System clock of 100MHz, SD base clock frequency 100MHz with 0 and 90 degreephase shift. System reset is active low signal, which is de-asserted after 100ns.
Processor BFM does the read write access to DUT, Serve the ADMA request and InterruptRequest.
The SD Card BFM will receive the command/data coming from the SD interface and transmitcorresponding response/data to host through DUT.

## 5.3 Test Case Description

The Single Test case will does the below function

- Initialization of SD card: The SD Data width mode and speed mode configuration is done
- Single block write: 512 Byte Single Block mode write using ADMA
- Single Block Read: 512 Byte Single Block Read using ADMA
- Multi Block Write: 512 Byte Multi Block(4 Block) mode write using ADMA
- Multi Block read : 512 Byte Multi Block(4 Block) mode Read using ADMA

## 5.4 Step to Run Simulation

- Open the Modelsim Tool
- Go the file →Change directory.

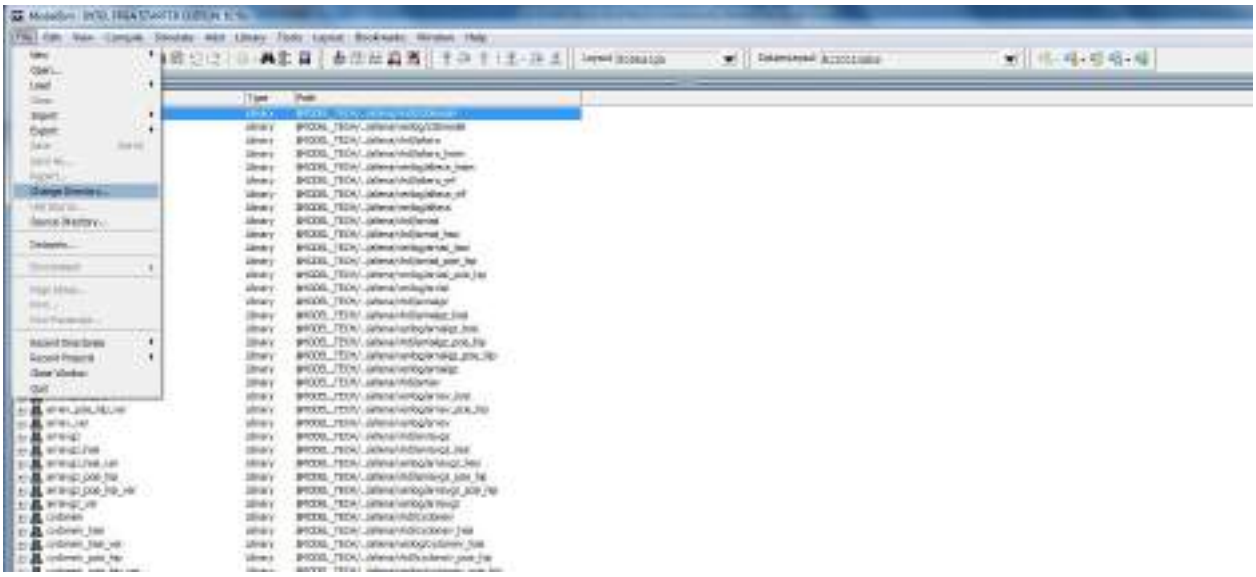


**Figure 31: Change working directory**

- Add the local path were the simulation files are located. Say for eg., E:\iW-EMEP6-FF-R1.0-REL1.0\iW-EMEP6-SI-R1.0-REL1.0\testbench

- The library “Work” is automatically loaded into the tool for the directory as shown in thefigure.

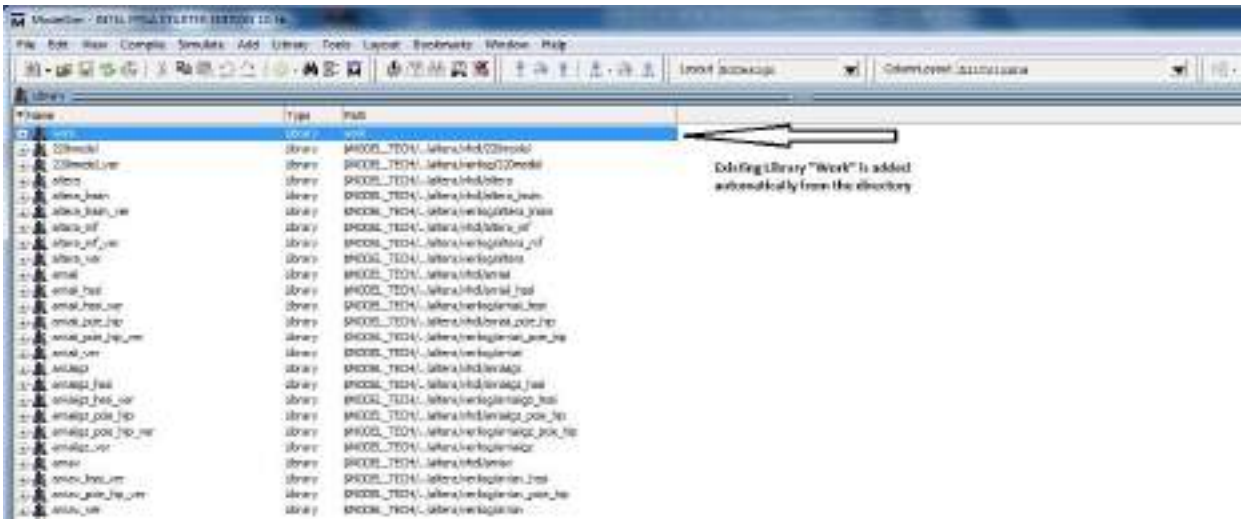
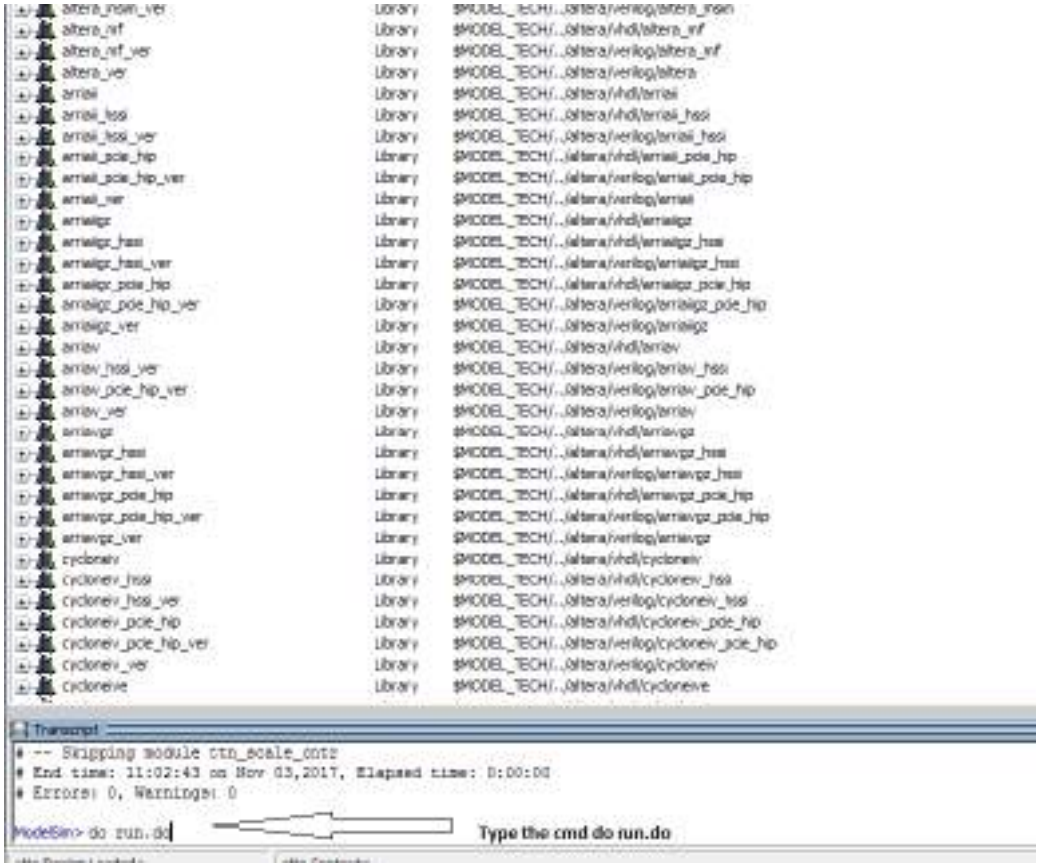



**Figure 32: Library “Work”**

- Enter the command ***do run.do*** in the transcript window as shown. This macro willcompile and run the simulation.




## 5.5 Simulation PASS Criterion

Below print should come in the Transcript window

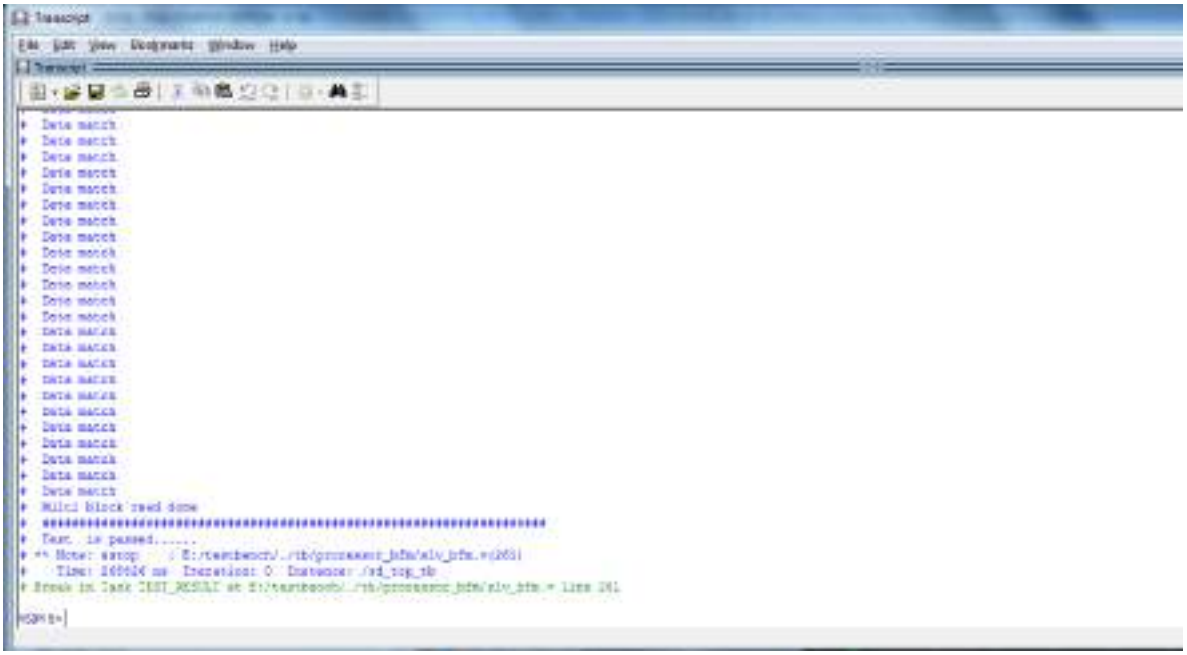



**Figure 33: Transcript window print**

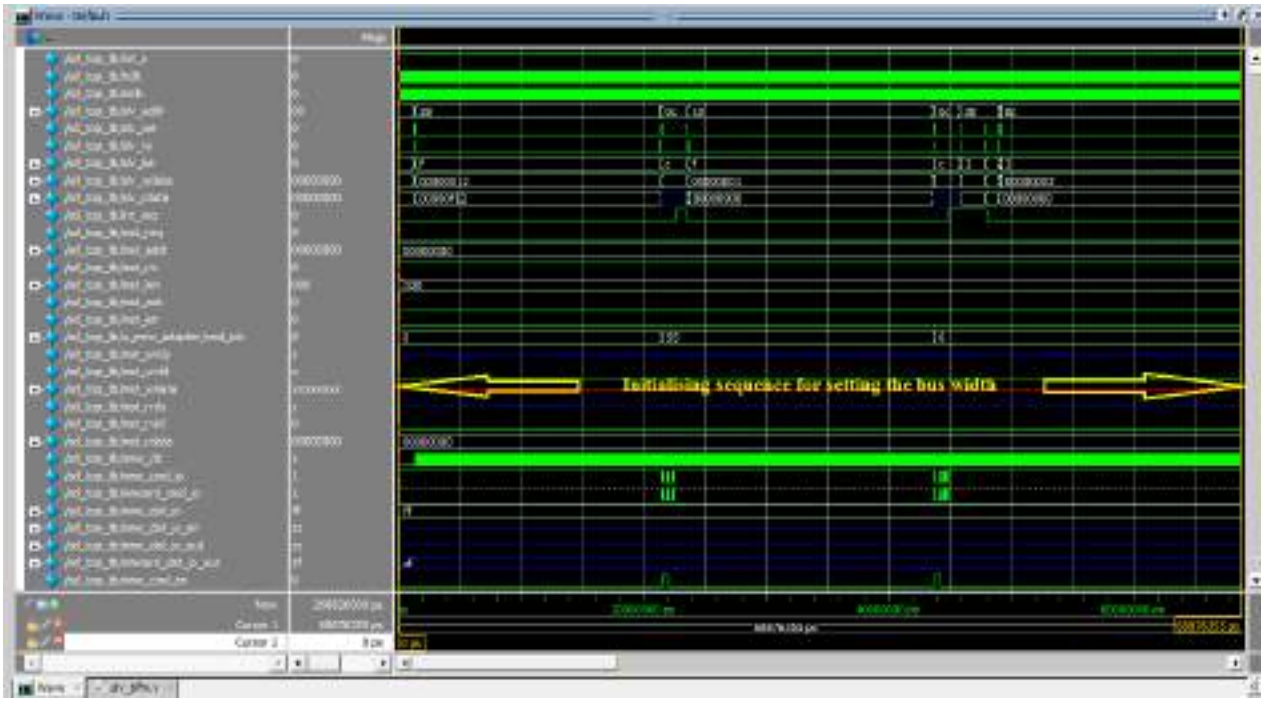



**Figure 34: Initialising the card by setting the bus width and speed**

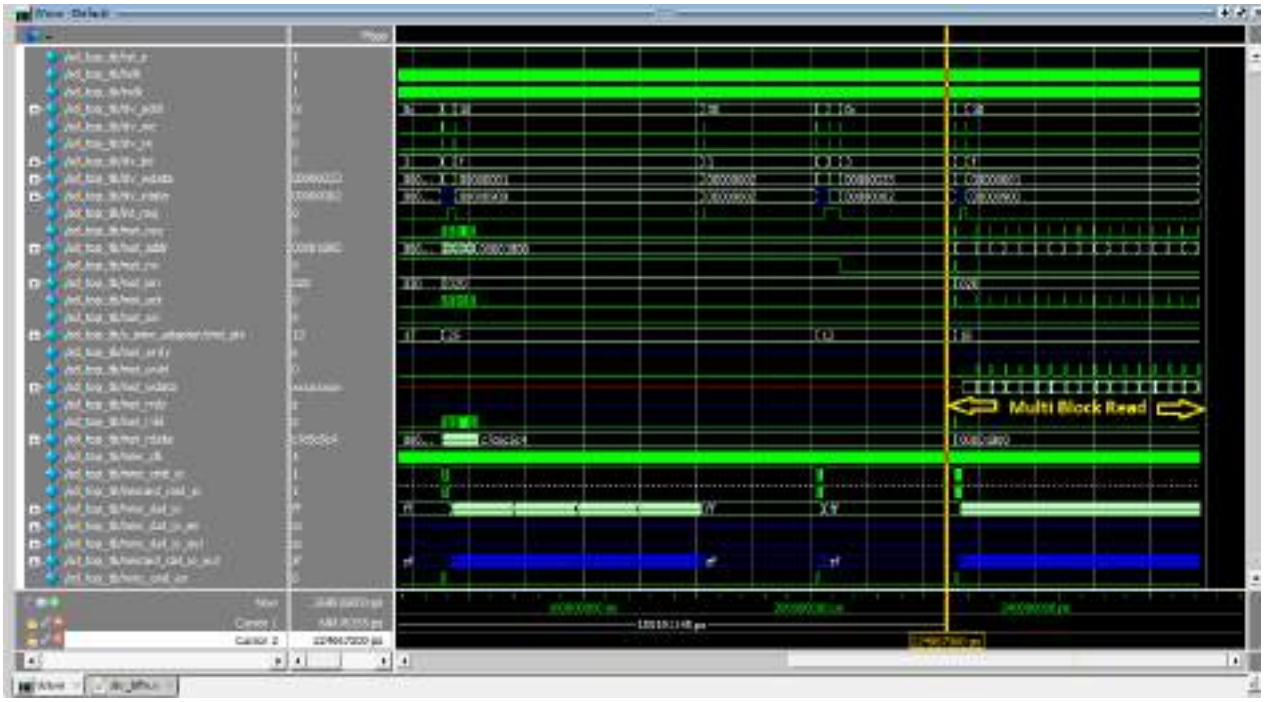



**Figure 35: Multi block read**

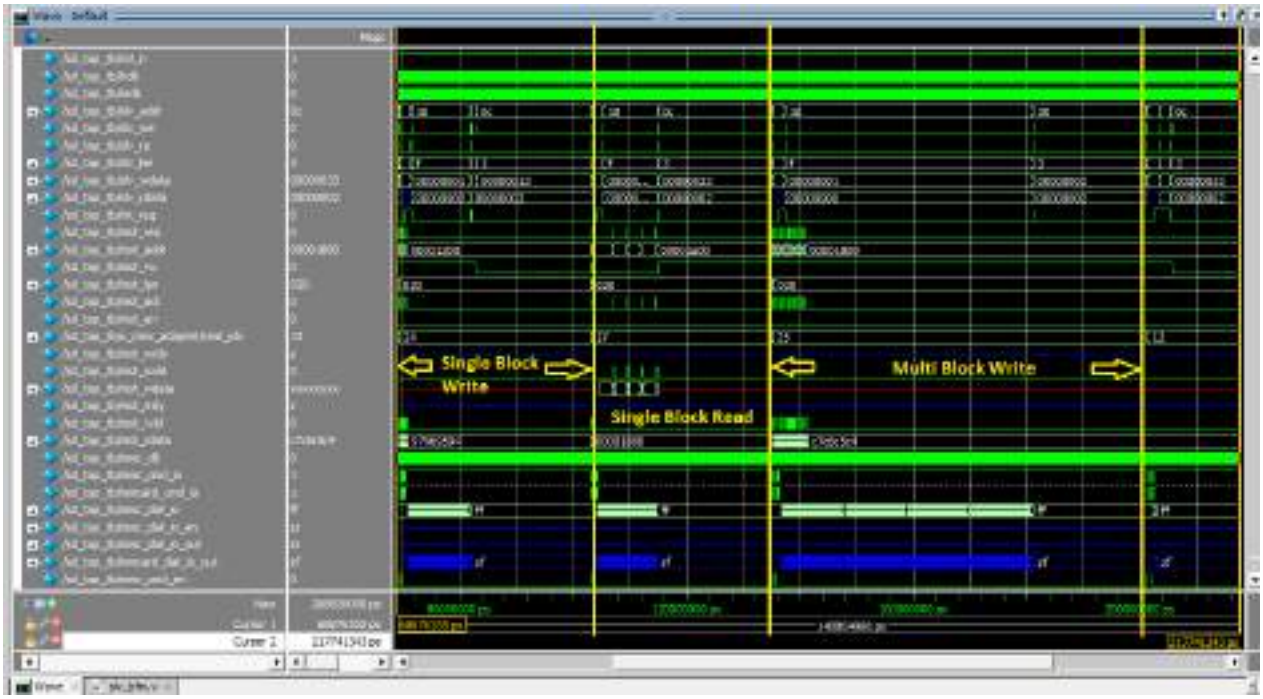



**Figure 36: Data read and write**