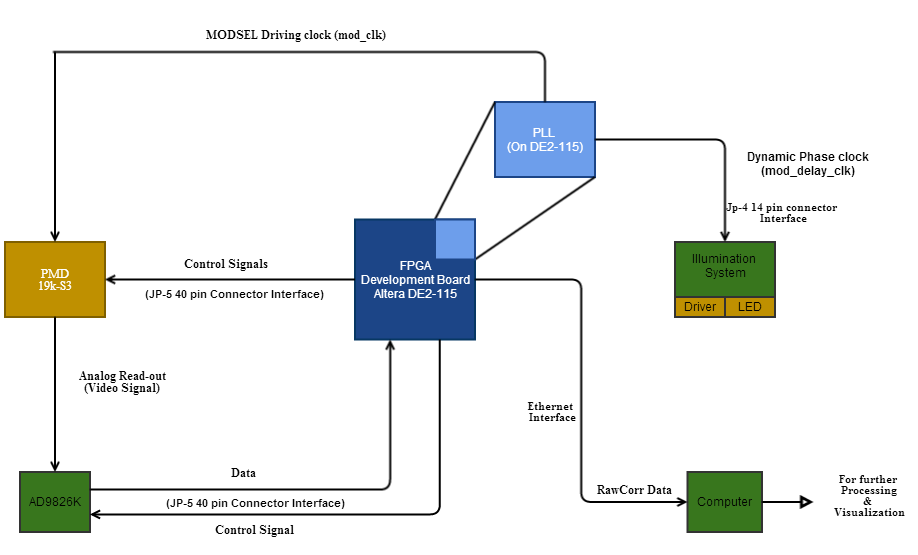
**ToF Ultra-fast Imaging Camera (Documentation)**

**Basic Block Diagram:**

(This part of the document portrays the important block diagrams which model the system operation.)

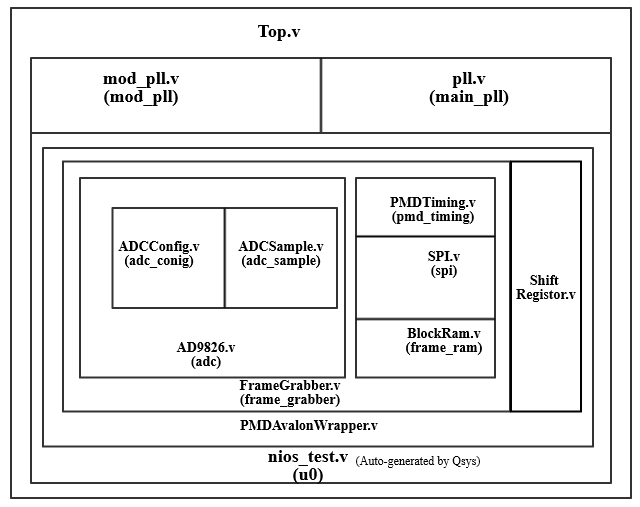


**Code Database:**

(This part of the document depicts various modules/functions/codes which control the overall data and control flow within the scope of the project.)

**Verilog modules:**

Verilog modules establish an interface between PMD Sensor, ADC and FPGA board and controls their whole operation.

**(Fig2: Verilog Module Flow)**

* **ADCConfig.v**

This FSM configure the FPGA and ADC for a bi-directional data flow. It has three states,

1. STATE\_ADDRESS -- Sends the R/W bit, and the three address bits
2. STATE\_DELAY -- Pause the operation for few cycles
3. STATE\_DATA -- Sends out the 9-bit data word

* **ADCSample.v**

This FSM controls the Sampling of ADC and the Readout process. It has three states

1. STATE\_IDLE -- Waits for the sample\_en signal to go high
2. STATE\_SAMPLE -- Sample and Hold
3. STATE\_READOUT -- Pushes the data into the Bus

* **AD9826.v**

Instantiates the ADCConfig.v & ADCSample.v and controls the whole ADC bi-directional flow with FPGA.

* **PMDTiming.v & SPI.v**

These module controls the whole timing operations of the PMD Sensor. They have 16 different states to control the whole flow of IDLE, RESET, INTEGRATION, READOUT, COOLDOWN for PMDTiming, & STATE\_DATA, STATE\_IDLE, STATE\_ADDRESS for SPI . Each of these states meet the timing requirements specified by the PMD Datasheet.

<http://web.media.mit.edu/~achoo/pr/19k3_datasheet.pdf>

* **BlockRam.v**

This module controls the reading and writing of ADC readout in and out of FPGA temporary

memory. The data from ADC is moved into a temporary register and then pushed into the Ethernet for data visualization on PC.

* **FrameGrabber.v**

FrameGrabber instantiates PMD\_timing, SPI, BlockRam & AD9826. This FSM has various states to control,

1. Setting frame size.
2. Grabbing the captured frame from the FPGA temporary memory.

* **ShiftRegister.v**

ShiftRegister module helps in driving the modulating signal to the light source and to the MODSEL terminal of the PMD Sensor.

* **PMDAvalonWrapper.v**

FrameGrabber is instantiated in this module. It interacts with the C-codes via the address bus and helps to set the Integration time & modulation mode. Since the C-code and the PMDAvalonWrapper are in unison through the address bus, the above specified values can also be set during the compilation of C-codes.

All the above specified modules are fed into the Qsys Library, and when the Qsys is generated, it automatically compiles all the Verilog codes and provides a single interfacing code named **nios\_test.v.**

**Qsys can be accessed by going into Tools→ Qsys**

* **PLL.v**

This is a module that is auto generated by using the **Megawizard plug-in manager Wizard**.

This can be accessed by going into Tools → Megawizard plug-in manager. The interactive GUI guides through the process of configuring the PLL for user requirements.

Here we use PLL.v to generate clock signals of different frequencies, but same phase(i.e. 0). These clocks are used to drive various blocks in the FPGA.

* **mod\_pll.v**

This is a module that is auto generated by using the **Megawizard plug-in manager Wizard**. Here we use mod\_pll, to generate two output clocks of same frequency, but one clock signal configured for dynamic phase reconfiguration. The latter one is fed into the light source, and the former into the MODSEL signal of PMD sensor.

* **top.v**

This is highest in the hierarchy, and instantiates three modules, i.e. PLL, mod\_pll & the nios\_test. This acts as a wrapper for the whole Verilog code base.

**MATLAB Code & functions:**

Matlab codes and functions helps in reading the data from the FPGA ethernet adapter into the Computer and Visualization of the read data.

* **pmd\_connect.m**

This function establishes a tcp/ip connection with the FPGA at a specified port and ip address.

* **pmd\_read\_image.m**

This function reads the image data via the tcp port, and shapes it to a specified frame rate and stores it in a variable named **rawCorr**.

* **pmd\_phase\_step.m**

This function writes through the ethernet, specifying the FPGA to increase/decrease the phase of the Light Source.

* **pmd\_600.m**

This matlab code calls the pmd\_read\_image and thereby captures the frames. The number of frames to be captured can be set in this code. After the frame capture is done, code automatically gets takes the FFT of the frames, and stores it in a variable named rawCorrFFT, and then it plots the angle and the amplitude images using the **mview.m** function.

* **mview.m**

This matlab function, inputs the frames captured and provides an interactive GUI to move through all the frames using a mouse scrollbar, and also provides an option to play it at specified frame rates.

**C-codes:**

C-codes builds a framework to establish a TCP connection between the FPGA board and Computer. These codes program to the soft core processor onboard DE2-115 & help in memory allocation. Alongside that, these codes also helps user to set the Integration time without disturbing the Verilog codebase.

There are mainly two project database to handle the C codes.

* **pmd\_tcp\_bsp**

This project helps to generate the Board support packages (BSP) & helps to re allocate the memory changes made in the Verilog code. In order to change update the BSP,

Goto Project Explorer → Right click on **pmd\_tcp\_bsp** → Nios 2 → Generate BSP.

* **pmd\_tcp**

This is the C project which controls the whole interface operation. **hello\_ucosii.c** is the main code that is can be used to update the integration time. This code, dumps the Memory write and retrieval code into the soft core processor and establish a bidirectional communication between FPGA and Computer.

**FPGA Documentation:**

This part of the guide helps in setting up the FPGA board (i.e Altera DE2-115 in this case), to control the timing requirements of PMD 19K-S3 and other peripherals.

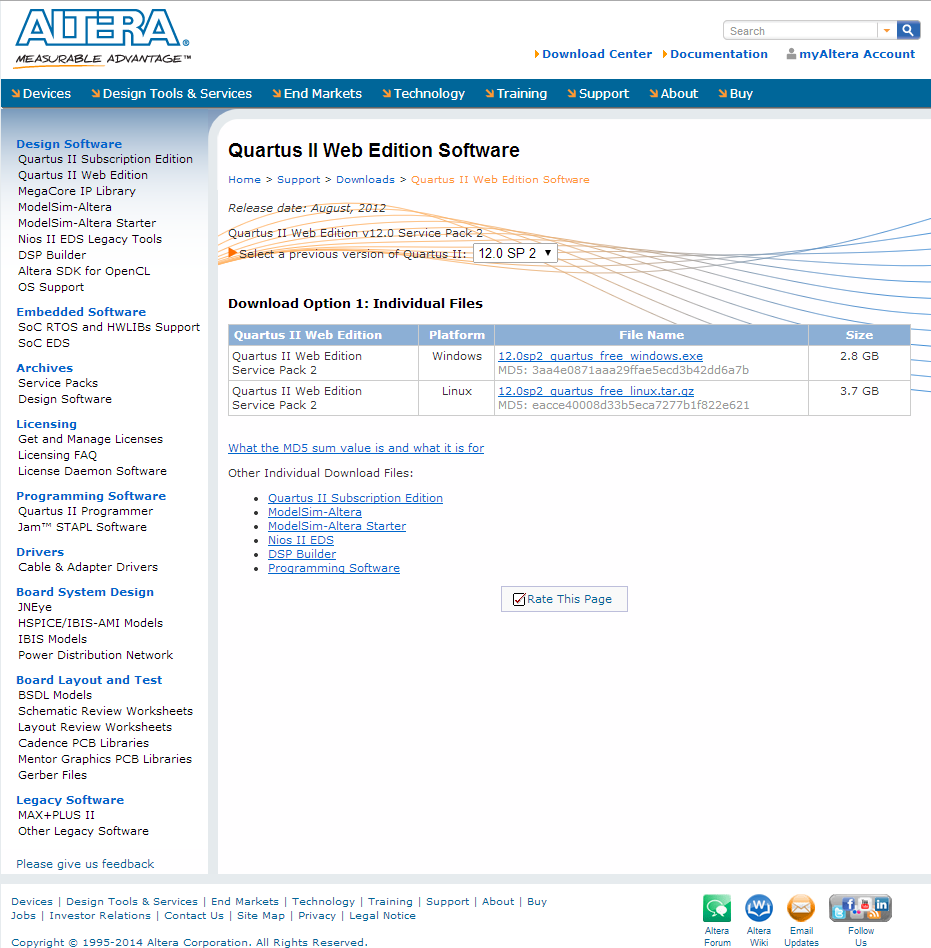
There are two ways of setting up the whole environment,

1. Experience the taste of Verilog by setting up everything from Scratch! (We give the codes though).

2. Use the archived project to directly set up the environment.

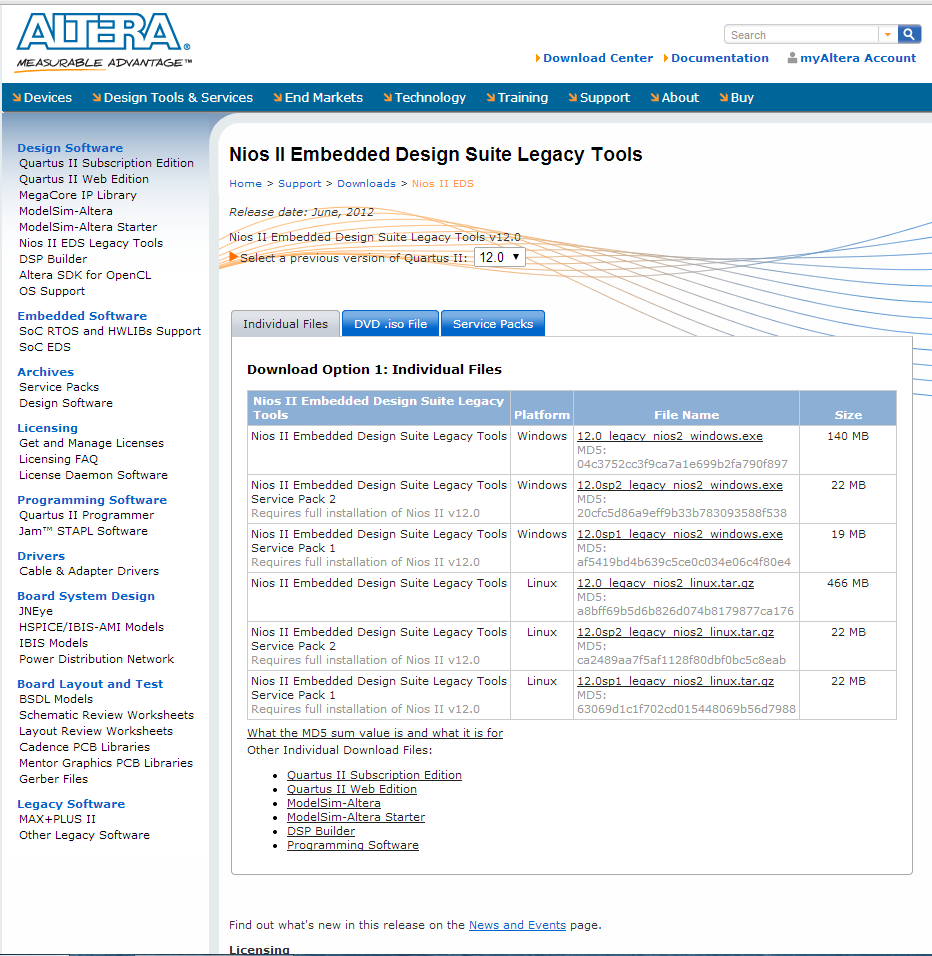
**Setting up required softwares:**

1. Quartus ii web edition:

<https://www.altera.com/download/software/quartus-ii-we/12.0sp2>

2. Nios ii tools:

Download both Version 12.0 and Service pack 2.

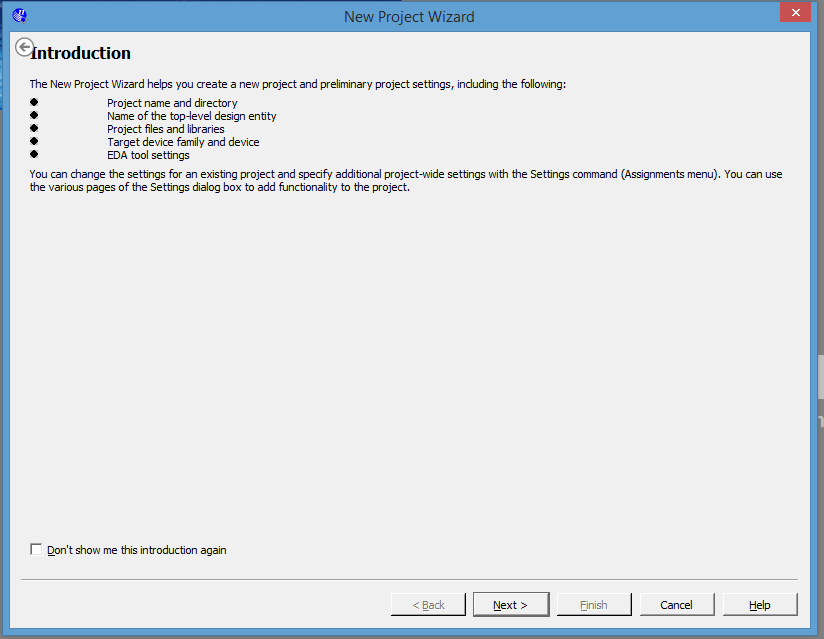
<https://www.altera.com/download/software/nios-ii/12.0>

3. Matlab (Version 2012 or above is preferred).

**Steps:**

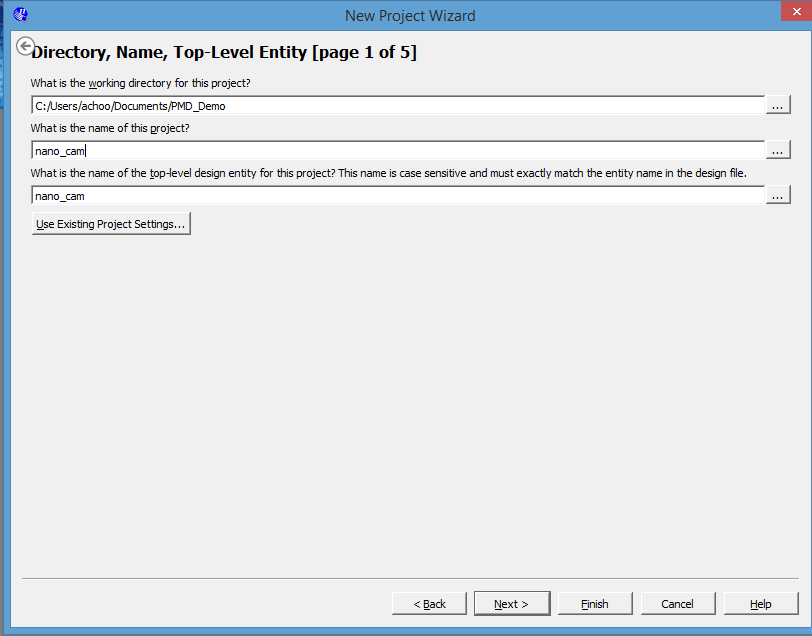
Here are the steps to set up the whole Verilog environment from scratch

1. Once all the softwares are installed and setup, open up Quartus ii → New Project Wizard.

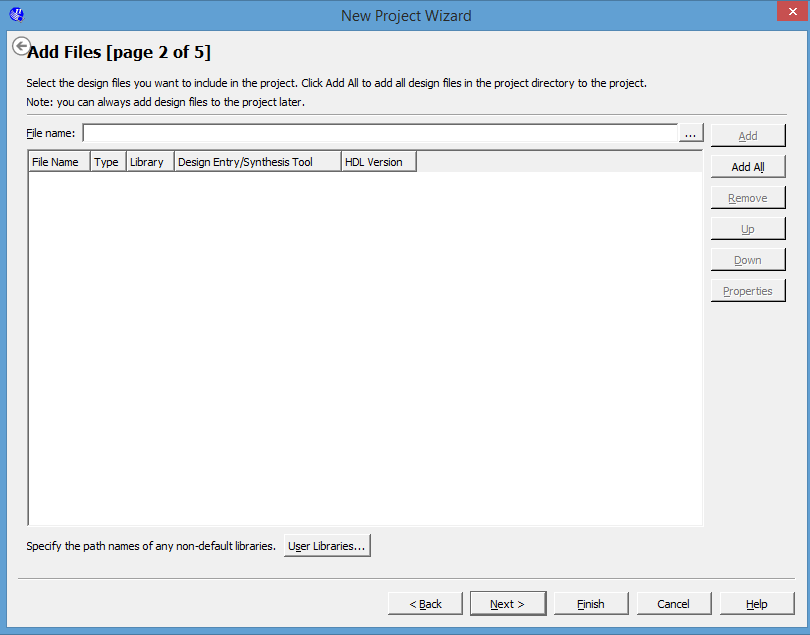


2. Click on Next, and the window below will pop- up.

* Set the default directory of the project.
* Give a name to the project! (We call this a Nano-Cam!).
* You may define the top level entity as “top”, but we can always modify this later.

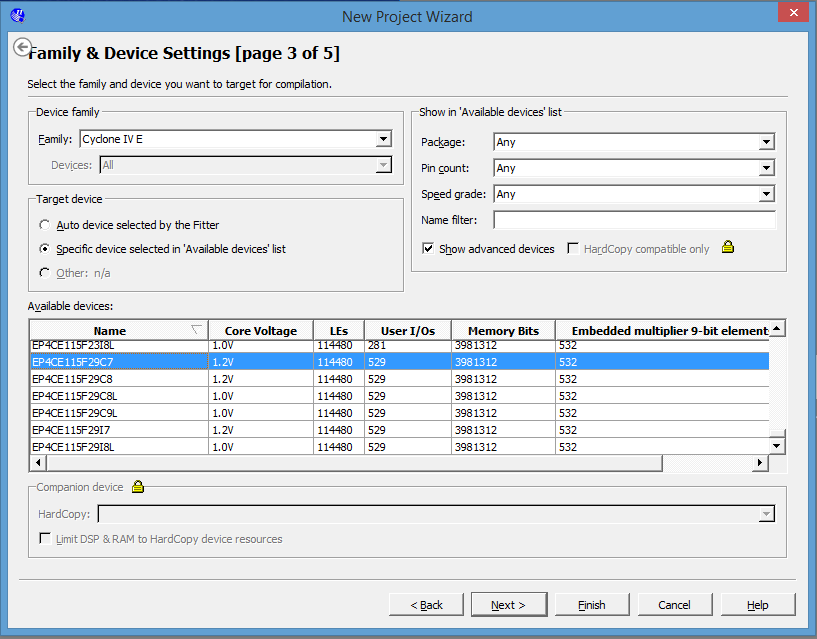


3. We will be adding all the design files later on, so you can just click on next on the below window.



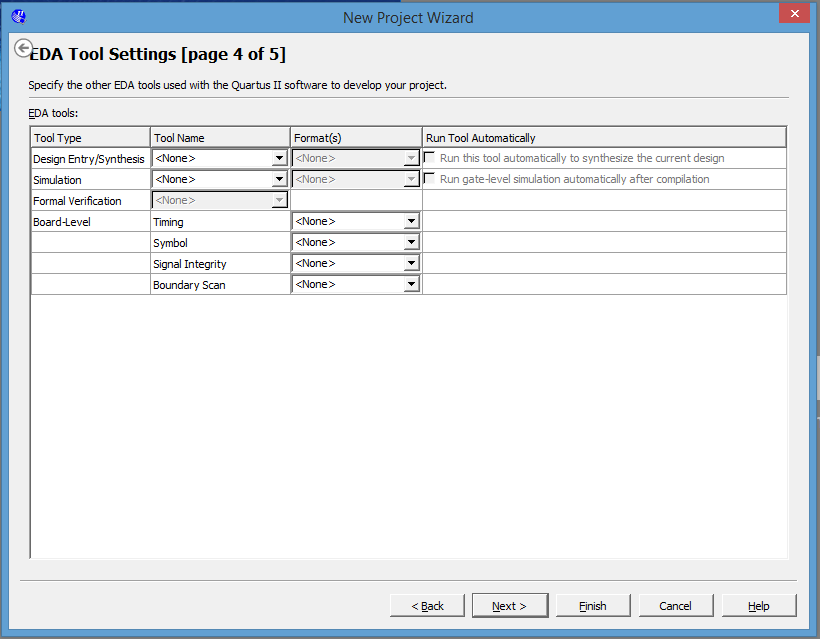
4. This page defines the FPGA Family and the device we are using.

* Select the Family to be Cyclone 4 E.
* And the device will be EP4CE115F29C7
* Click on next to move forward.



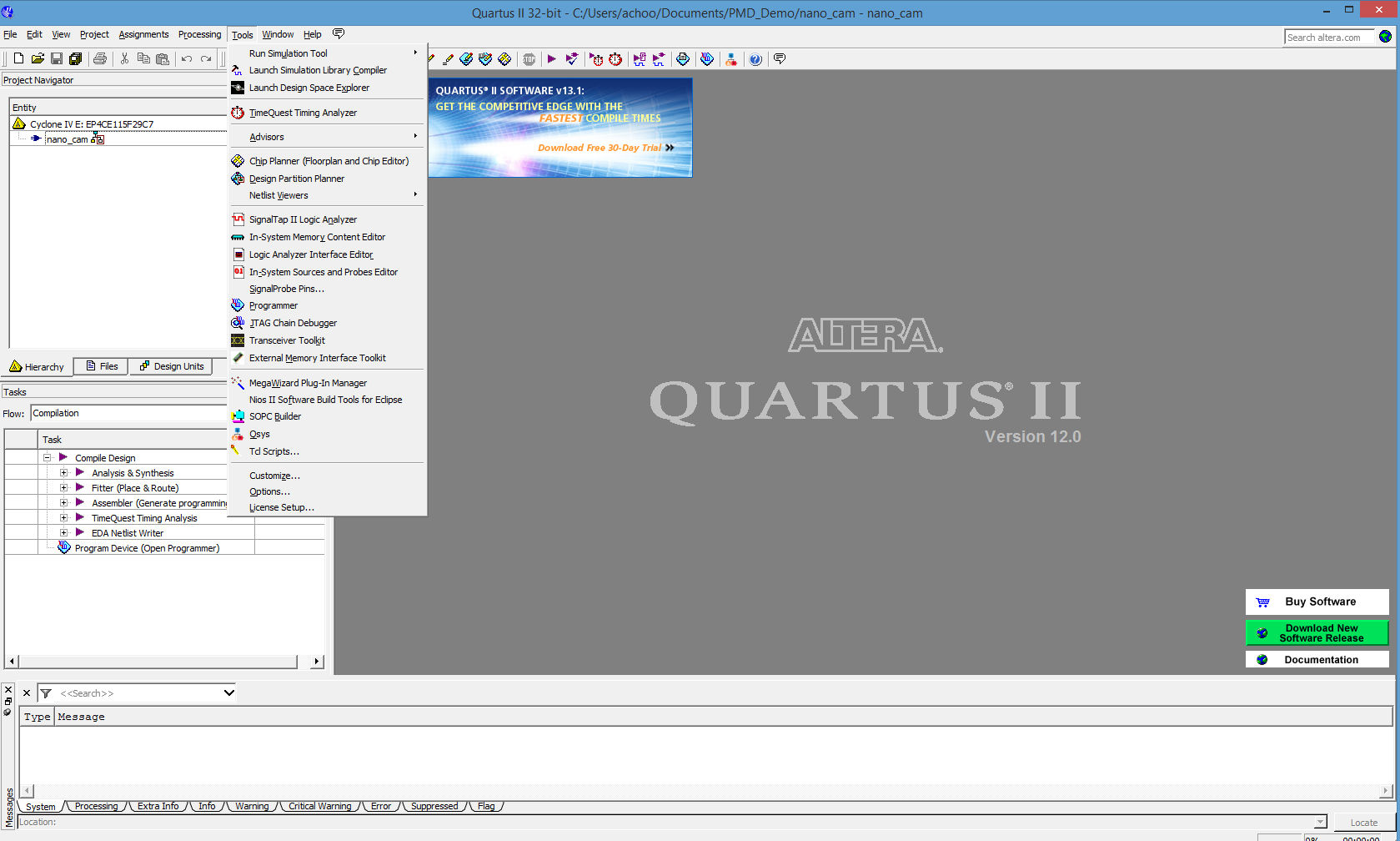
5. This window is to add EDA tools for additional simulation and Verifications, which we can skip and move into next window.

* Finish the project setup wizard.

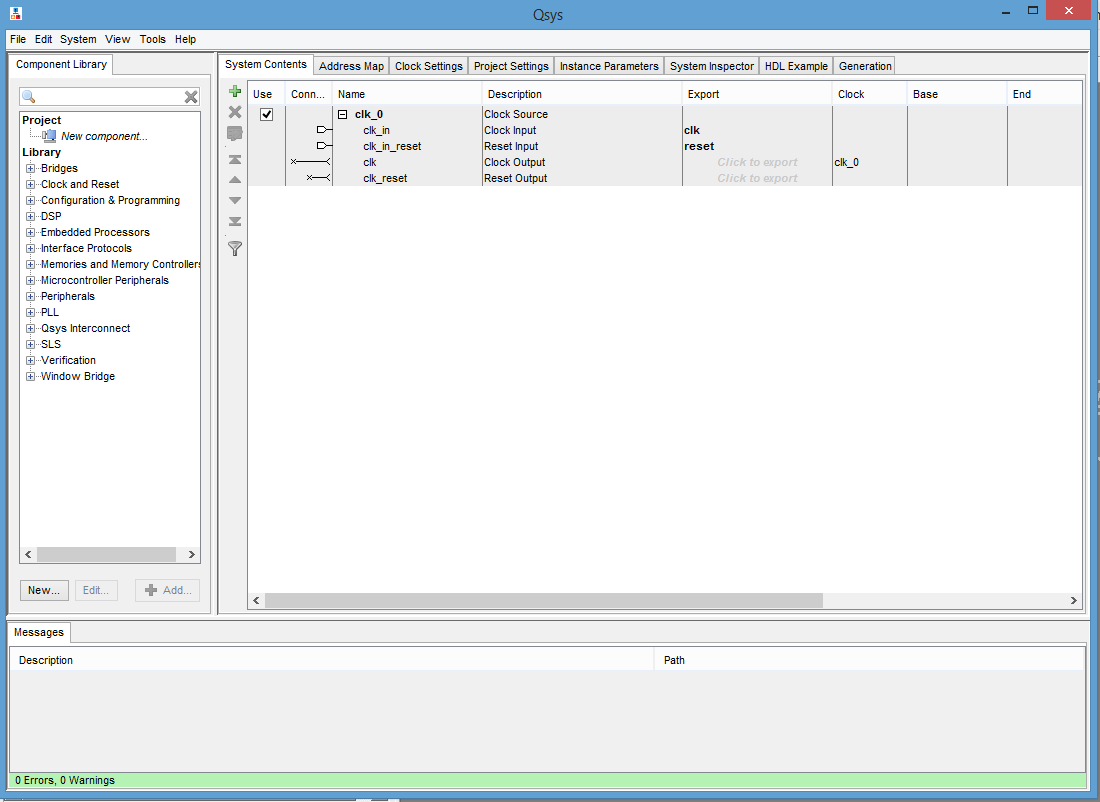


6. Now we will setup the System map. To do this,

Click on Tools → Qsys, on the top left tab.

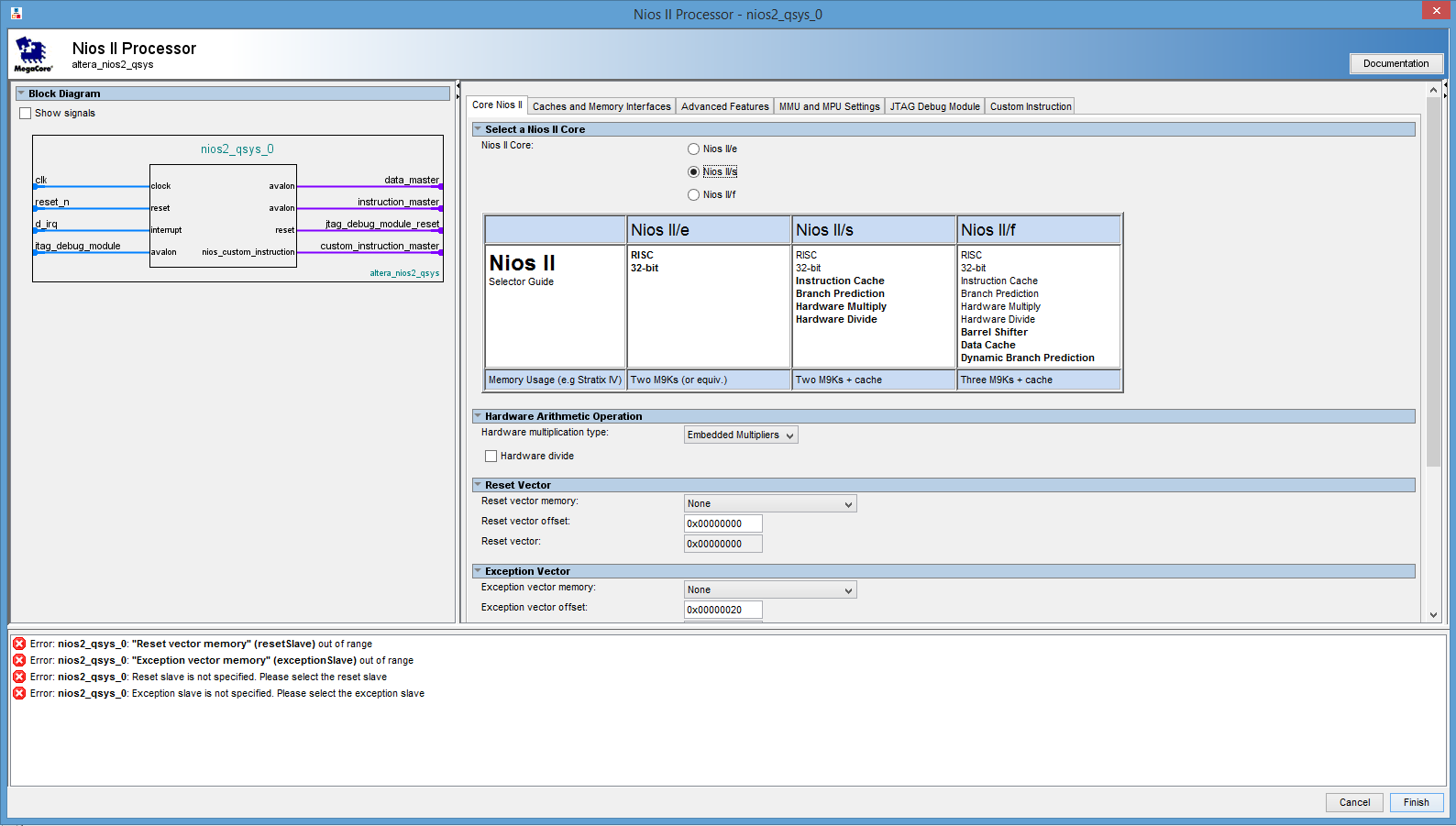


7. Once Qsys is up and running, we should see a window below.

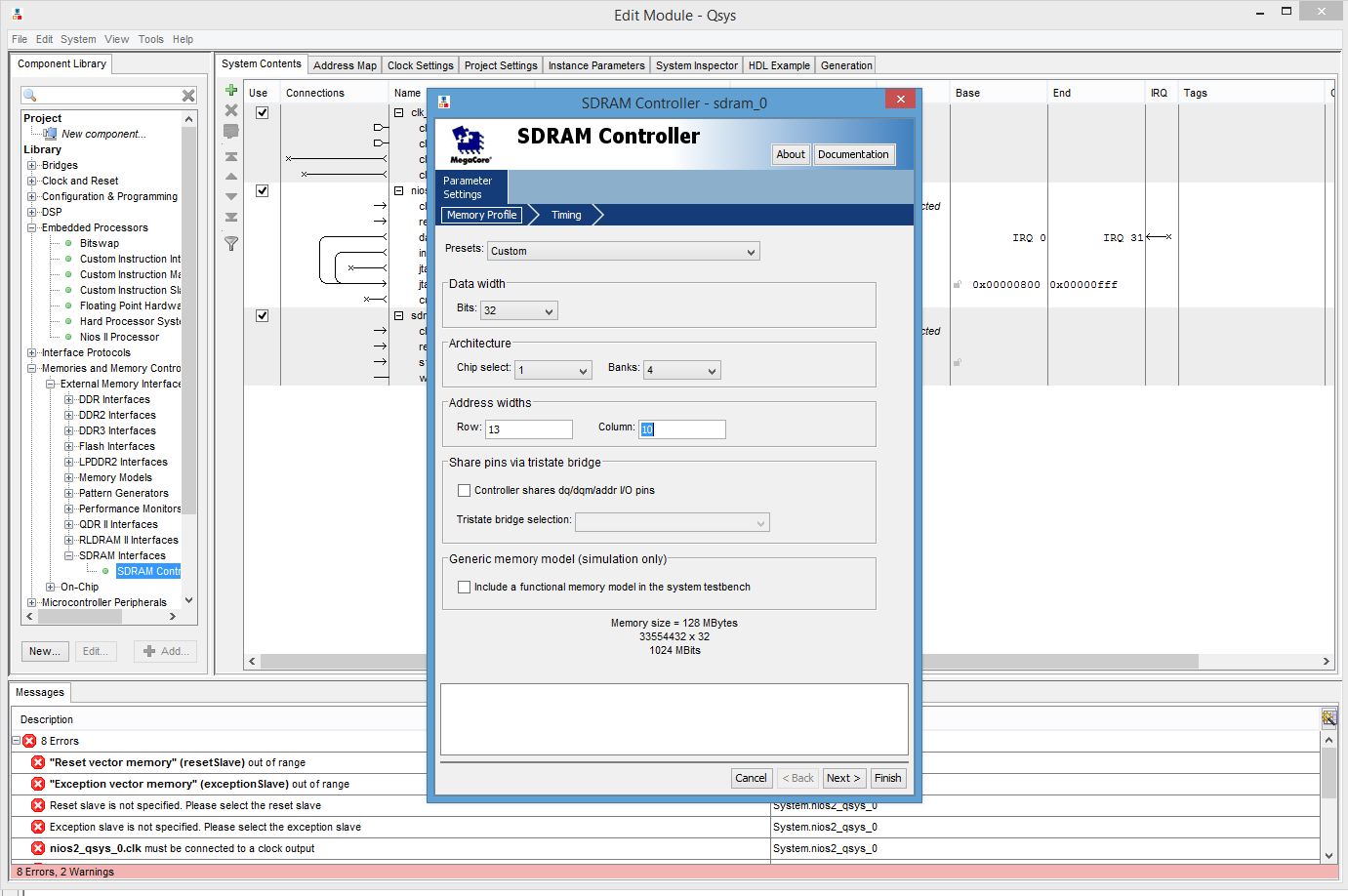


8. Now we need to add all the libraries we need,

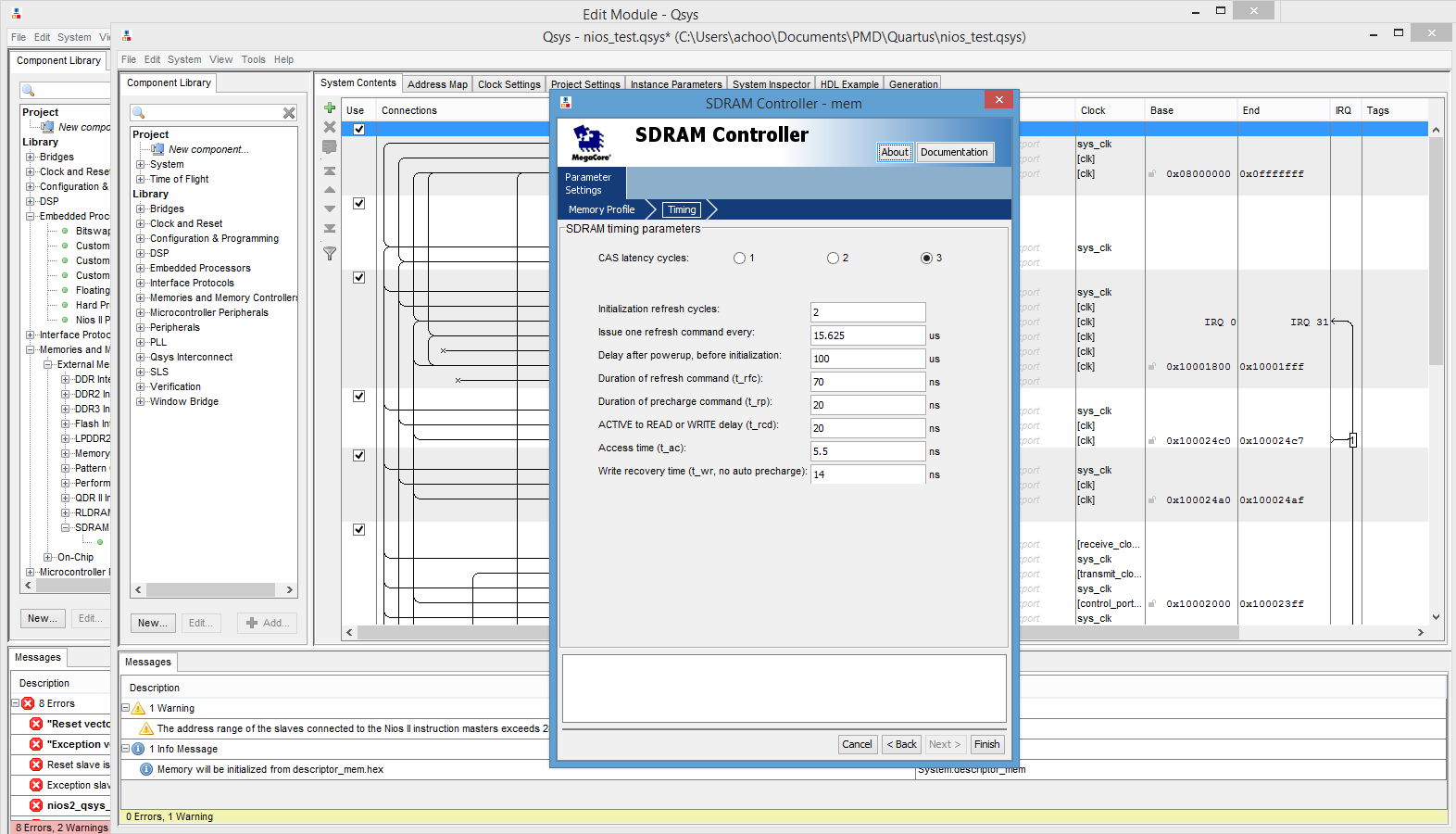
* On the left, click on Embedded Processors → Nios ii Processor.

Select the core to be Nios ii/s and then click on finish. 

* Click on Memories and Memory Controllers → External memory interfaces → SDRAM Interfaces → SDRAM Controller.

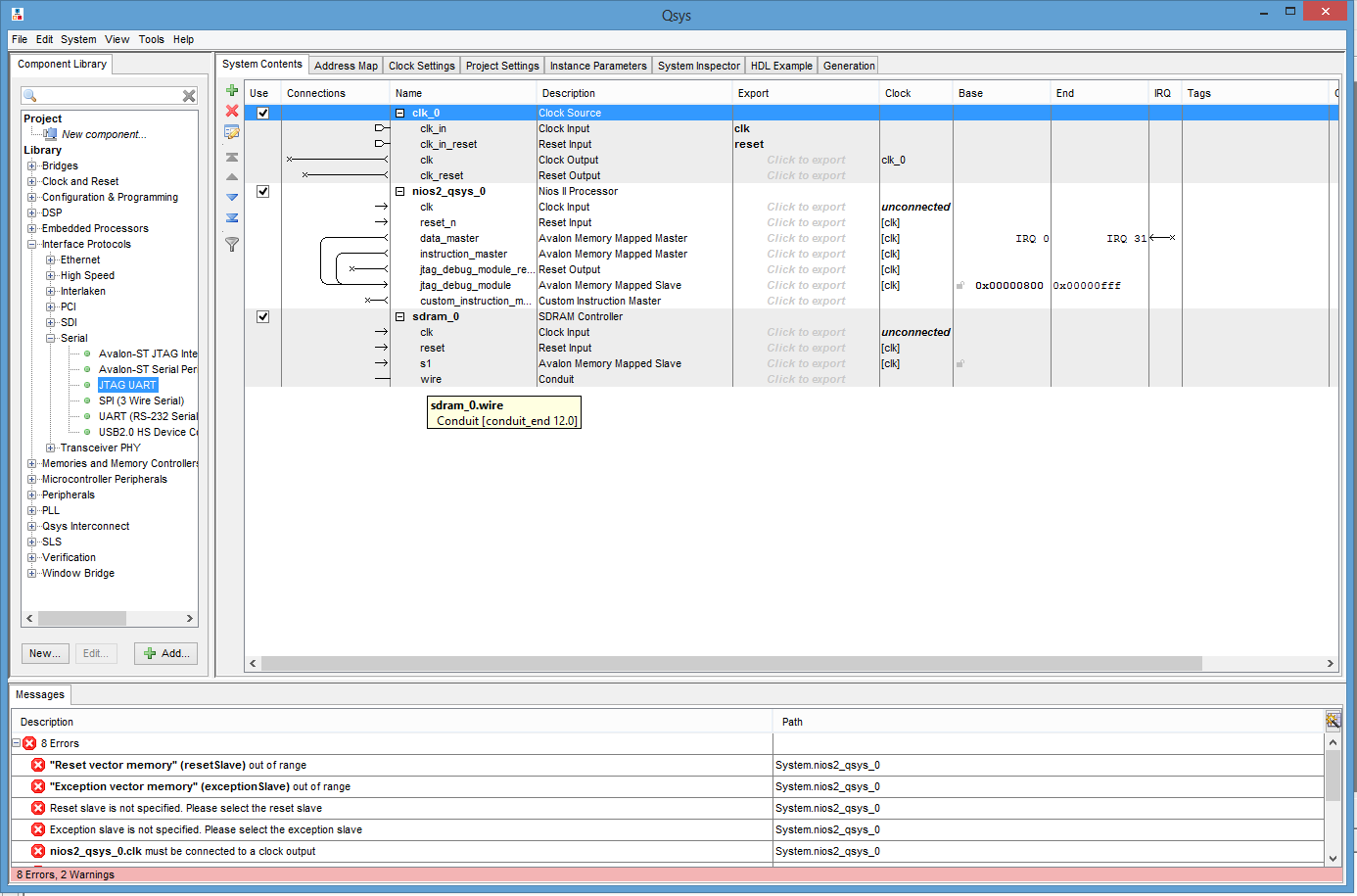


Select and set the values/parameters as indicated by the figure, and click next.

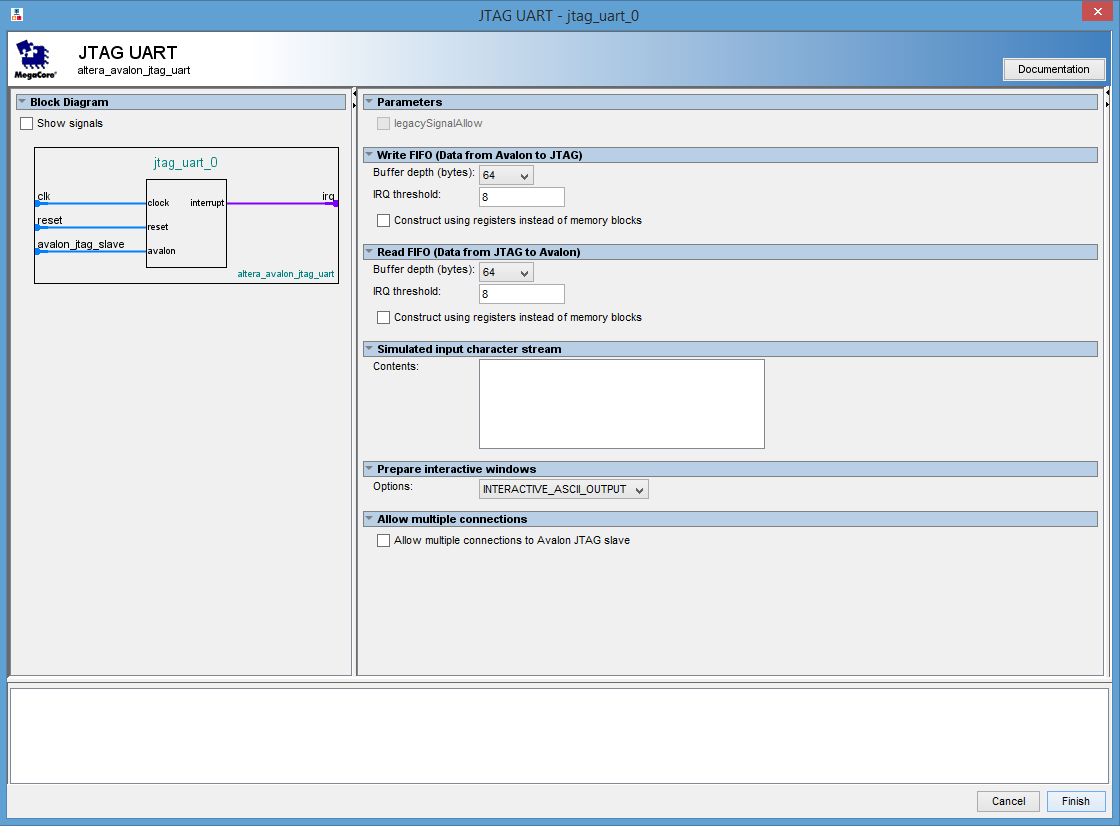


Now leave the timing parameters as it is and click on Finish.

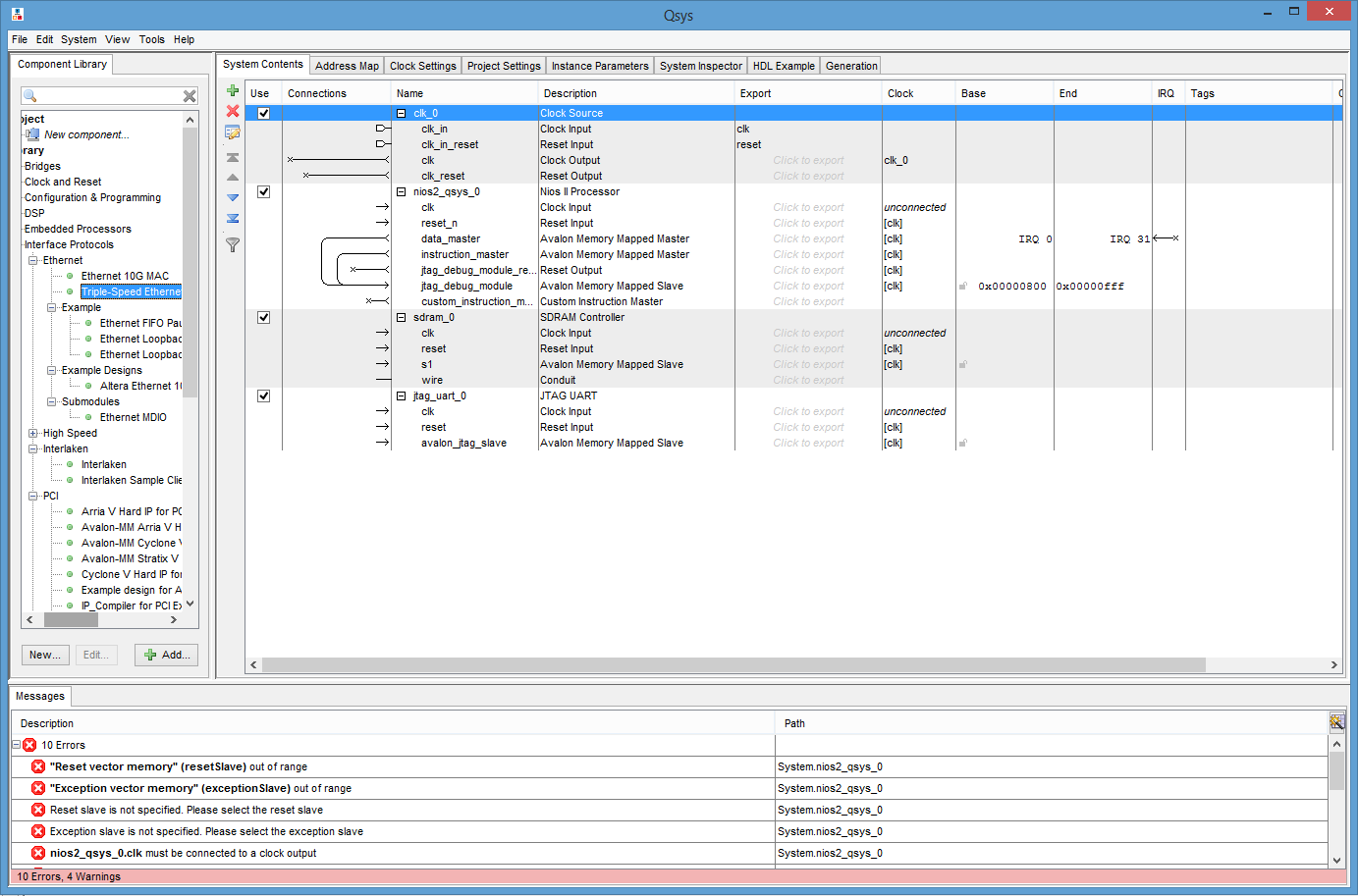
* Click on Interface Protocols → Serial → JTAG UART to create a new JTAG library from your left pane.



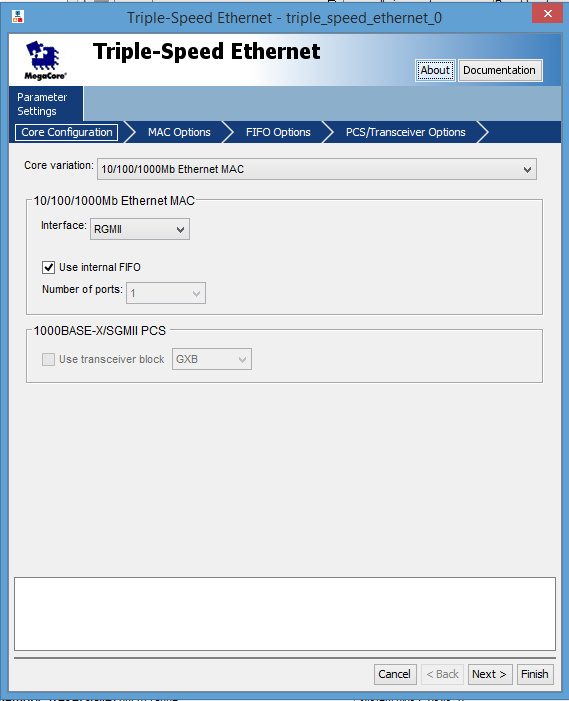
Once the window appears, do not change any settings, click on finish.

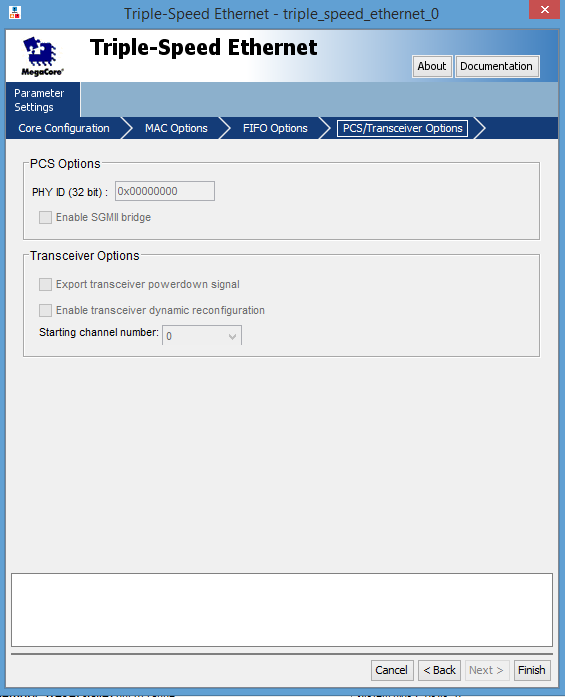
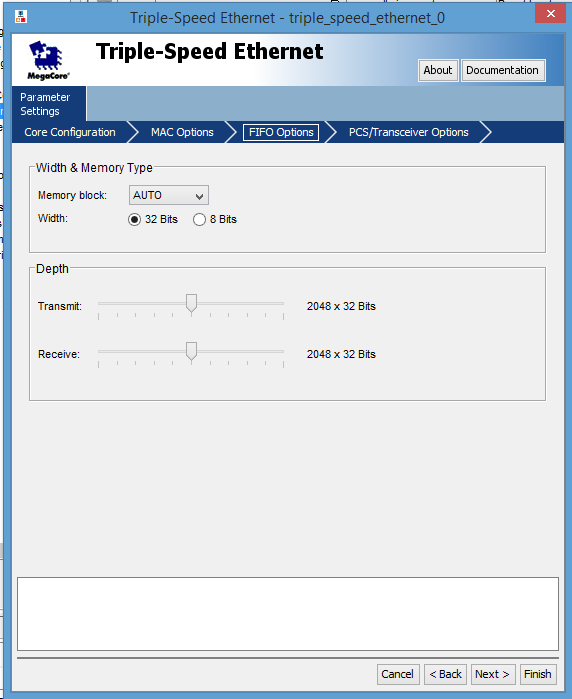
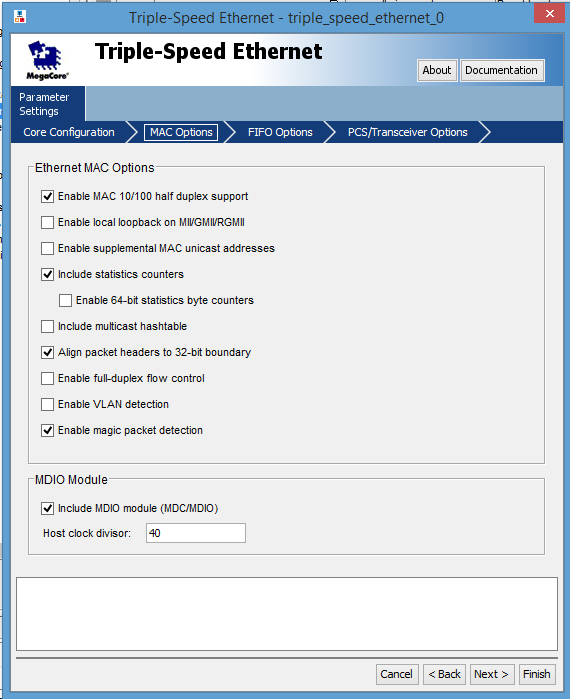


* Click on Interface Protocols → Ethernet → Triple Speed Ethernet.

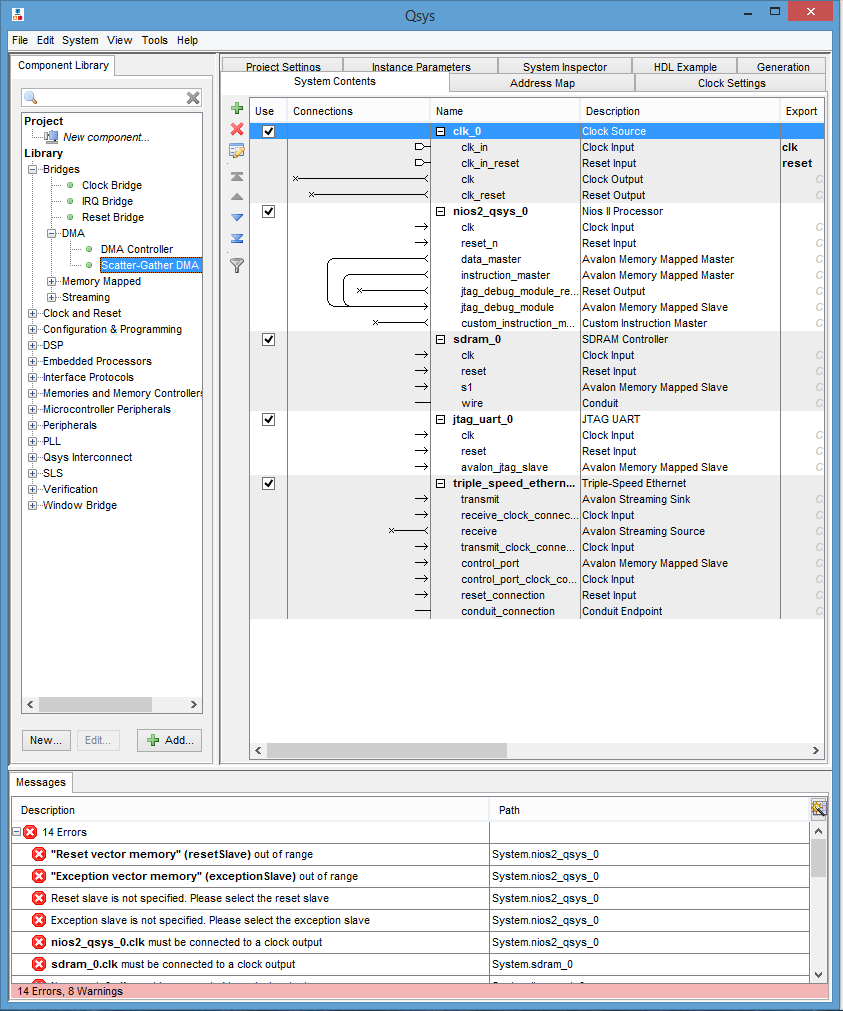


Now follow the below images clicking next, to complete the ethernet library setup. Please enter the values specified in the images to their respective fields.



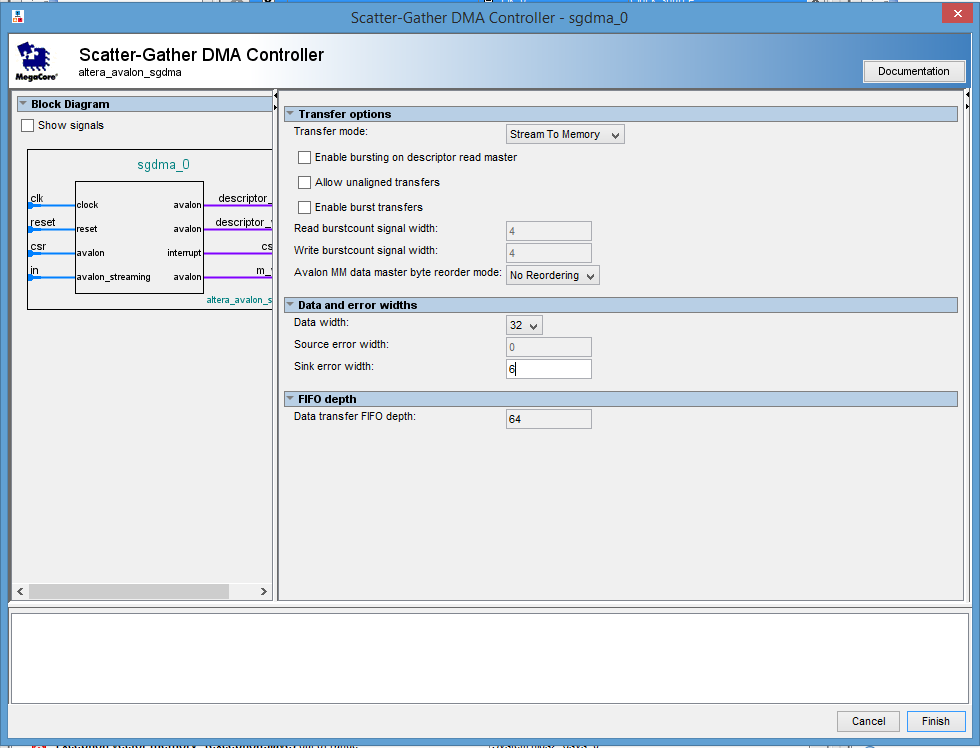


* Click on Bridges → DMA → Scatter-Gather DMA.



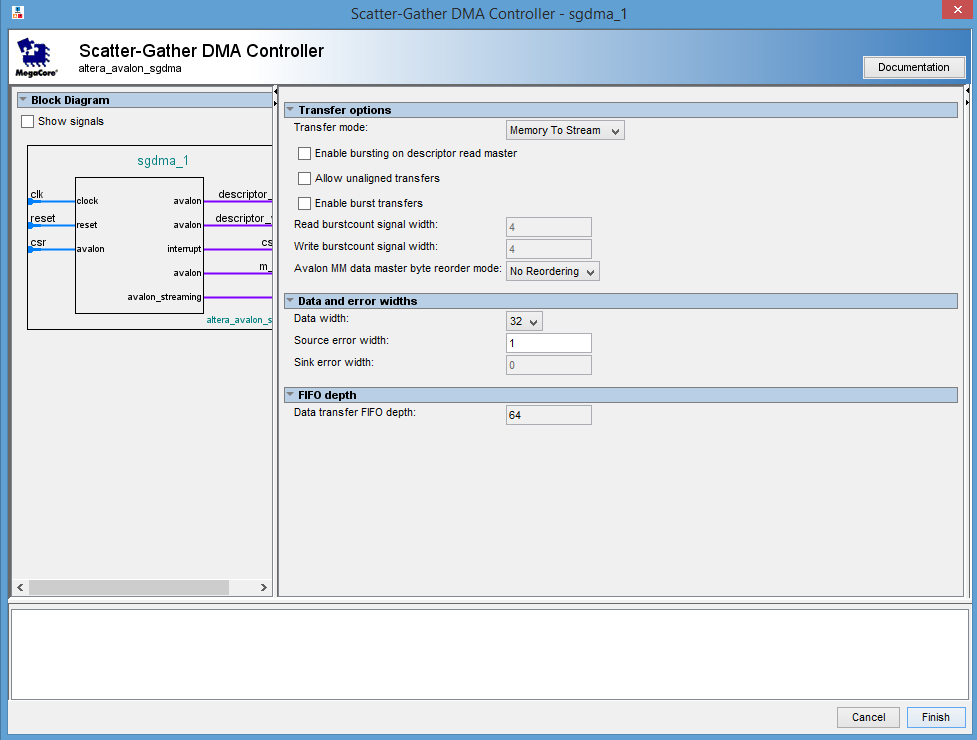
Once the window opens up,

Choose Transfer Mode : Stream to Memory and click on finish.

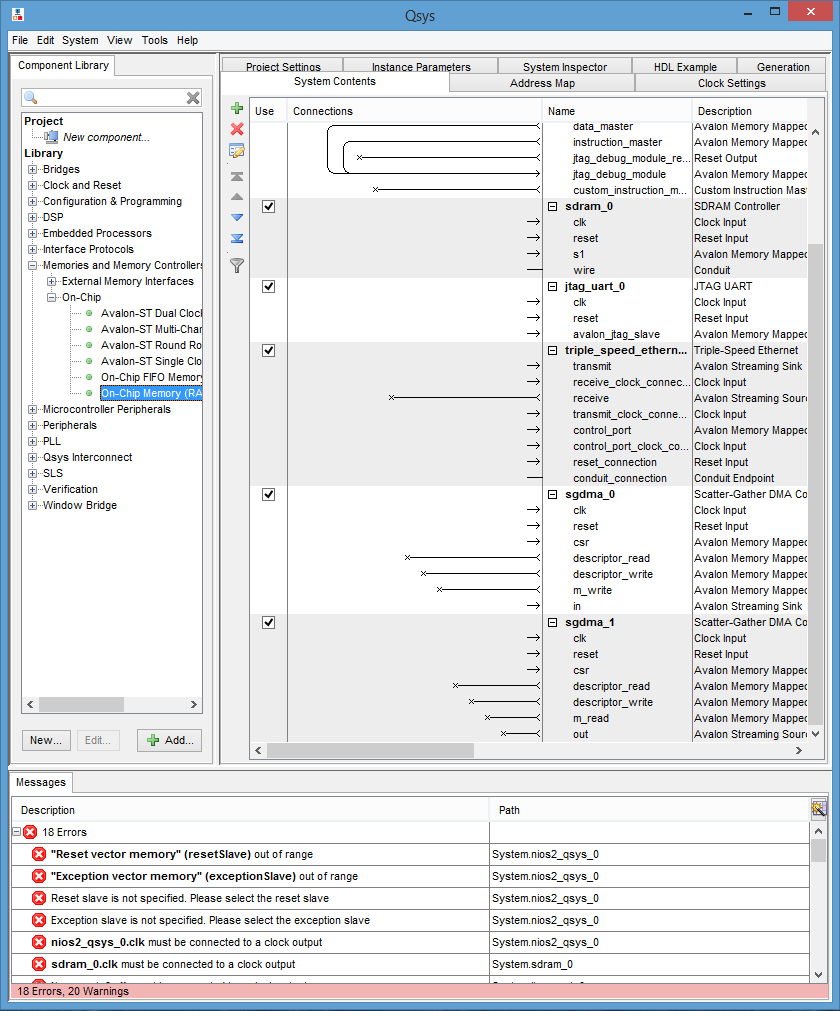


Repeat the above steps to create another Scatter Gather DMA,

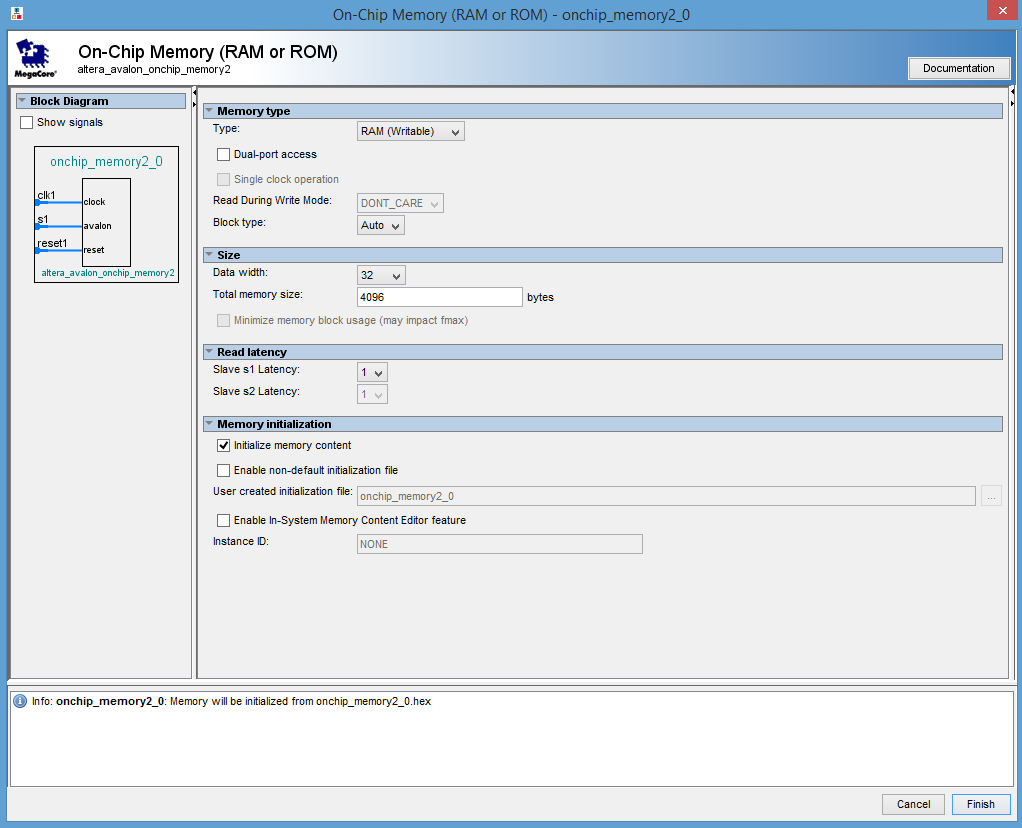
Choose Transfer mode here as: Memory to Stream and click on finish.



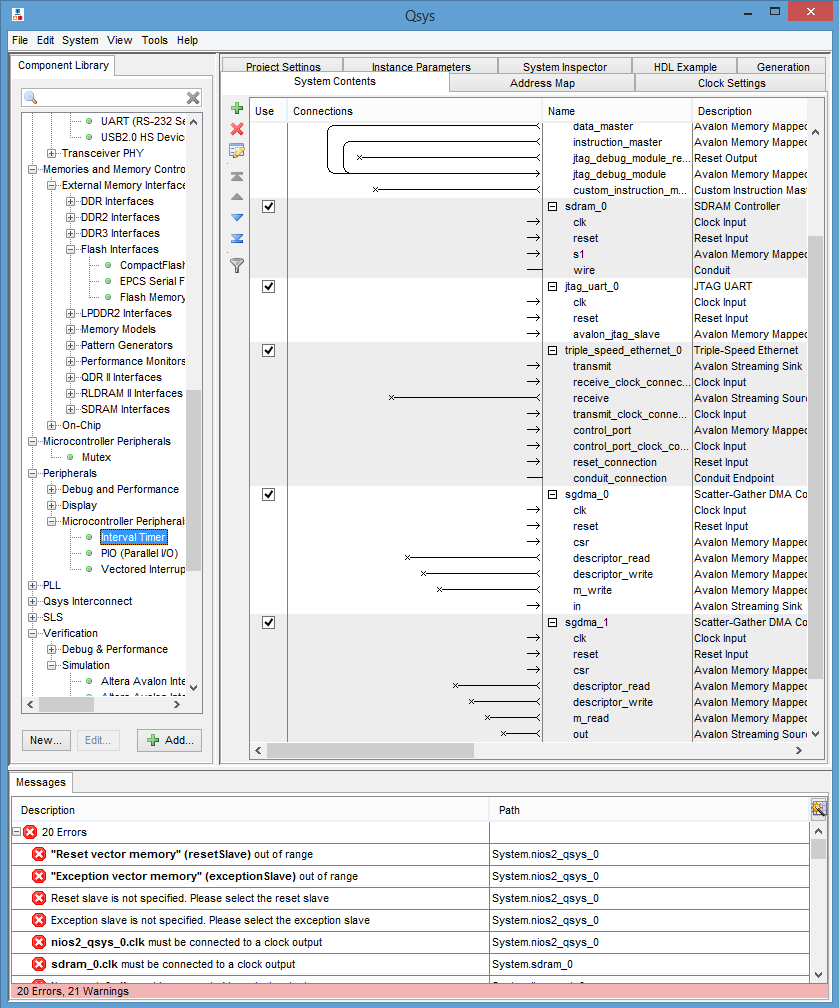
* Click on Memories and Memory Controllers → On Chip → On Chip Memory (RAM).



Once the window open up, do not edit anything, just click on finish.

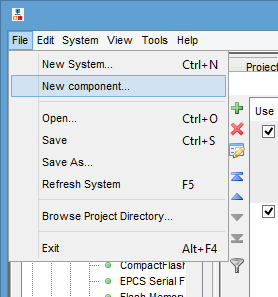


* Click on Peripherals → Microcontroller Peripherals → Interval Timer.

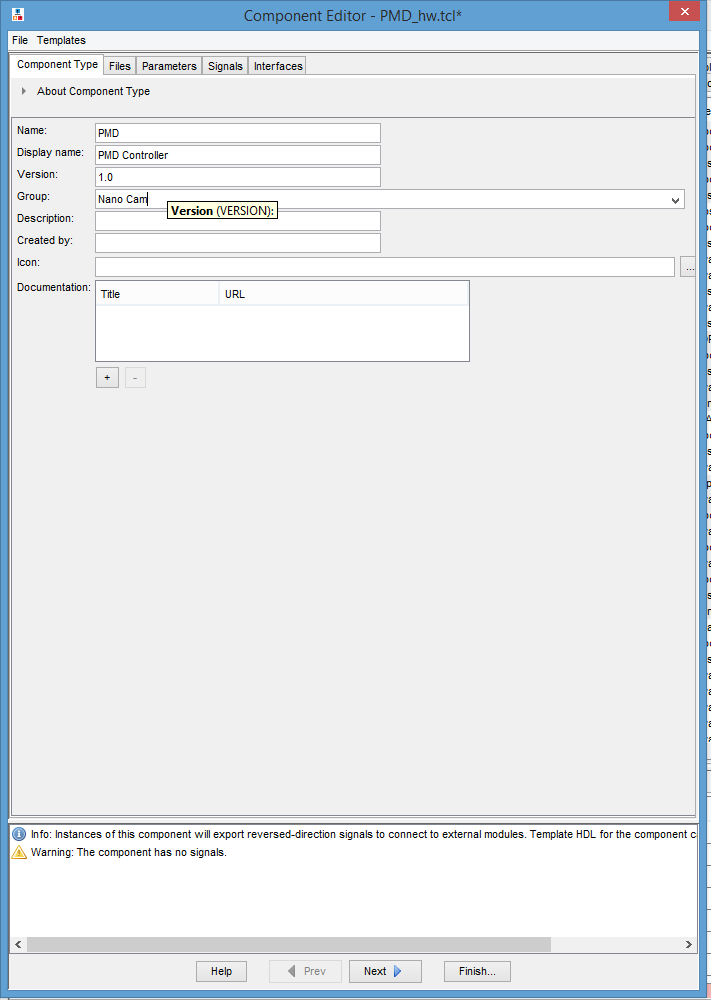


Once the window appears click on finish without editing anything.

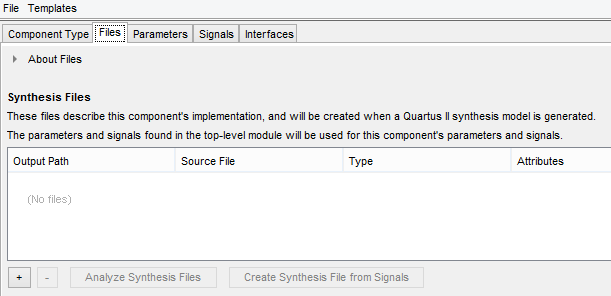
* Now, we will add our custom library for the Sensor, click on File → New component.



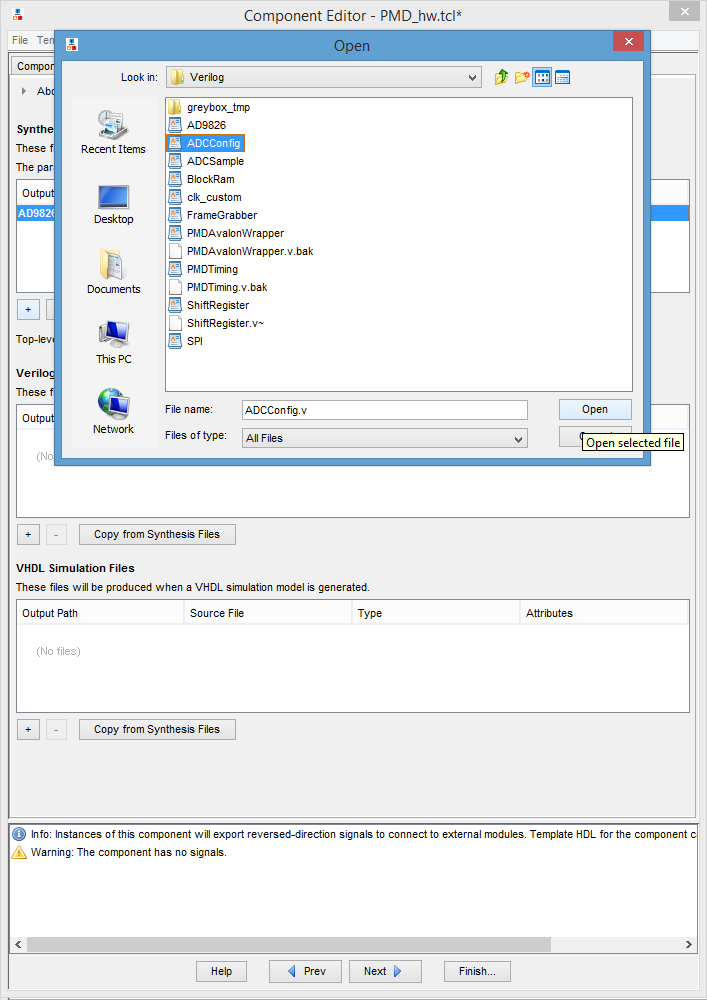
Now enter the following information into the text fields to create the library, and click next.



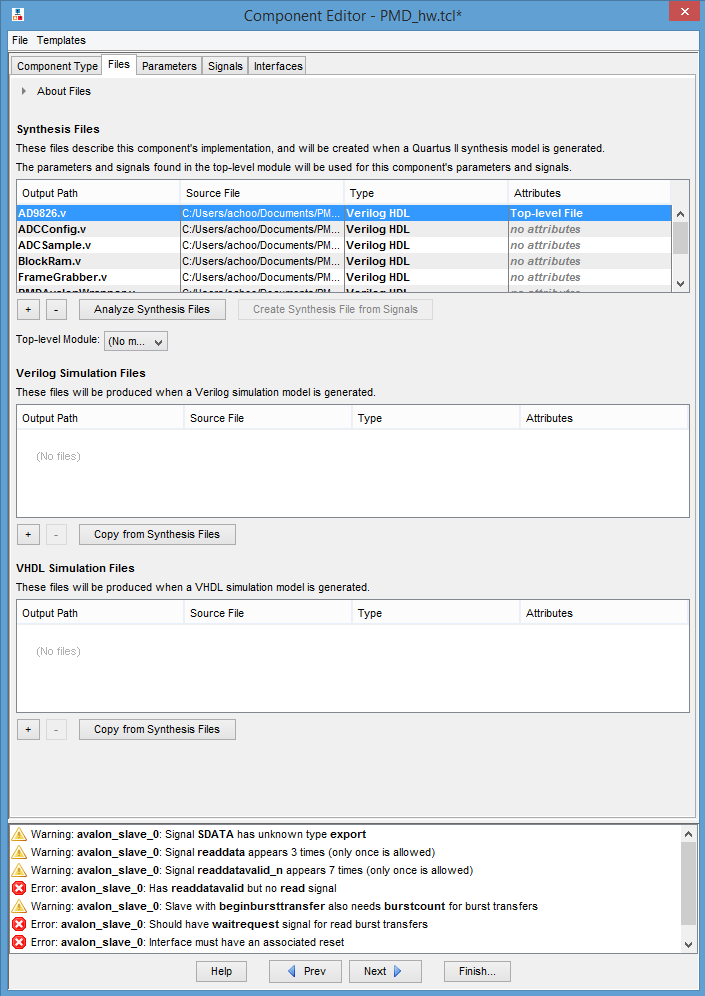
Click on the “+” button at the left corner to add the verilog files.



After clicking the button, navigate to the folder which is named “Verilog”, within the supplied codebase.

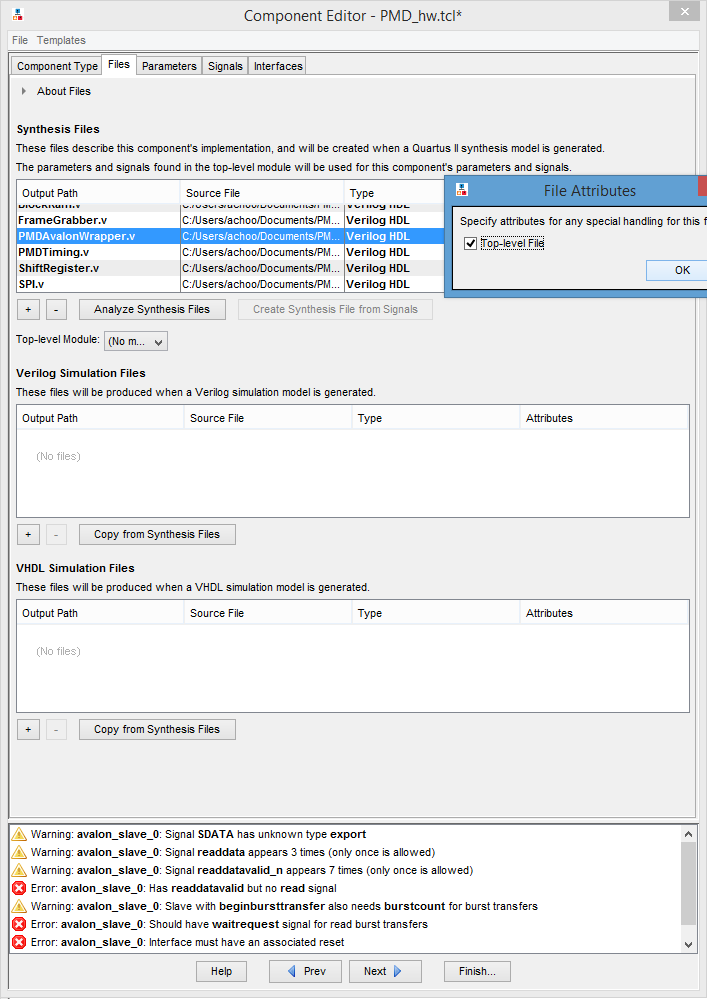


Select a file and click open. Repeat the above steps till you add all the files in Verilog directory.



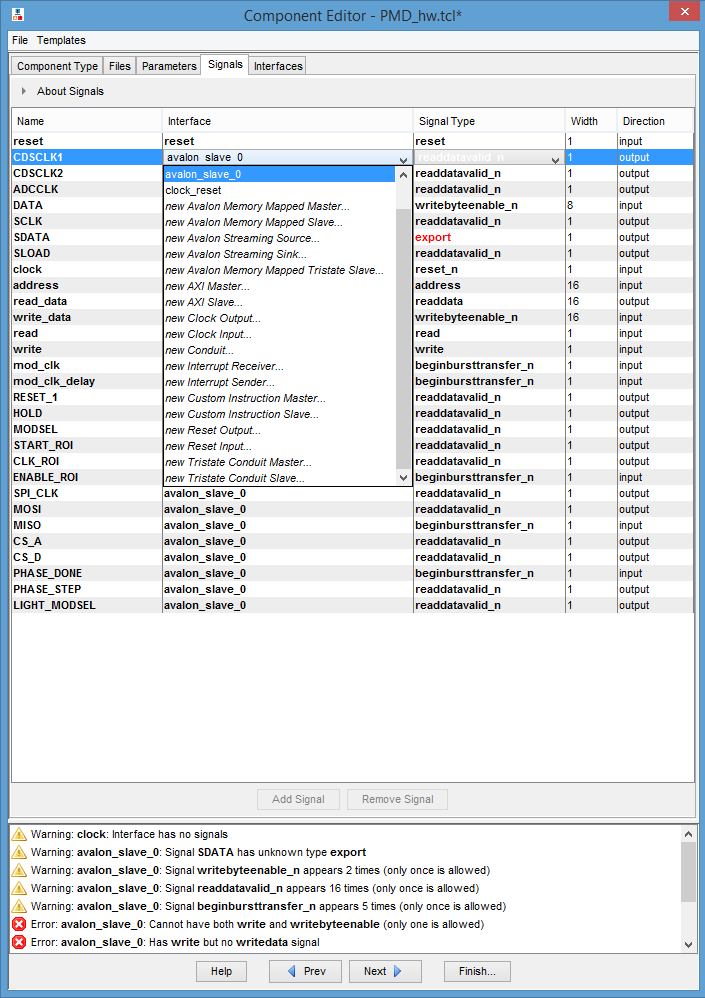
After all the files are added, the wizard automatically assigns Top-level attribute to the file on the top of the list!! But our top level module is PMDAvalonWrapper. So click on the attribute field on ADC9826, and uncheck top-level attribute.

Navigate to the PMDAvalonWrapper, and set the attribute to be top level file.

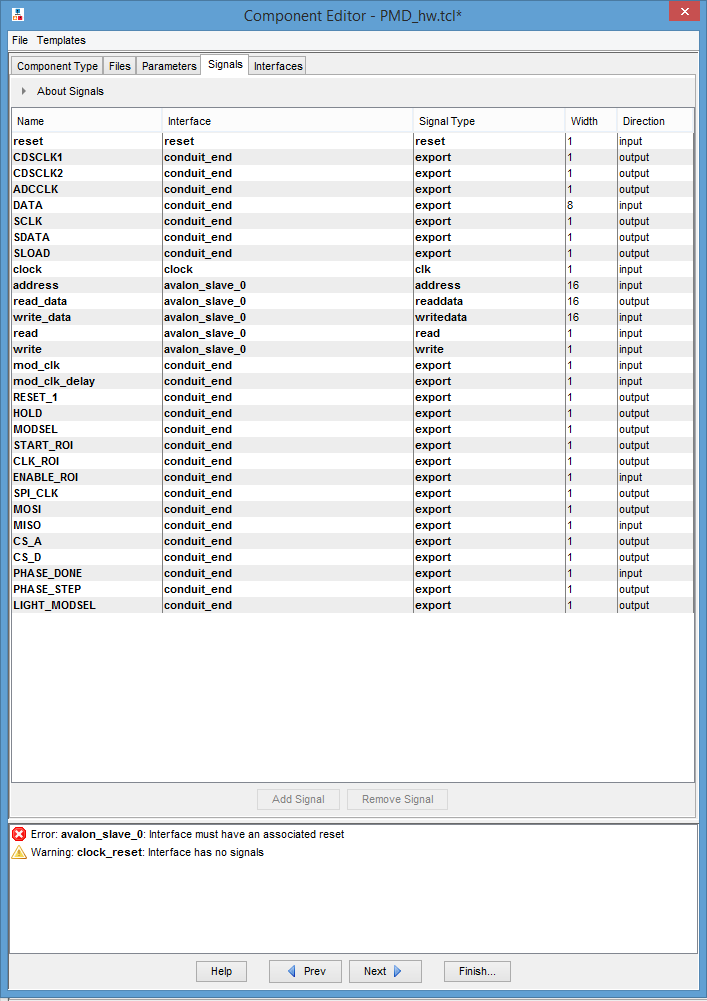


Click on ok, and then click on Analyse Synthesis Files. Once the analysis is done, close the console which pops up.

Now click on the Signals tab from the above pane, to edit the signal attributes.

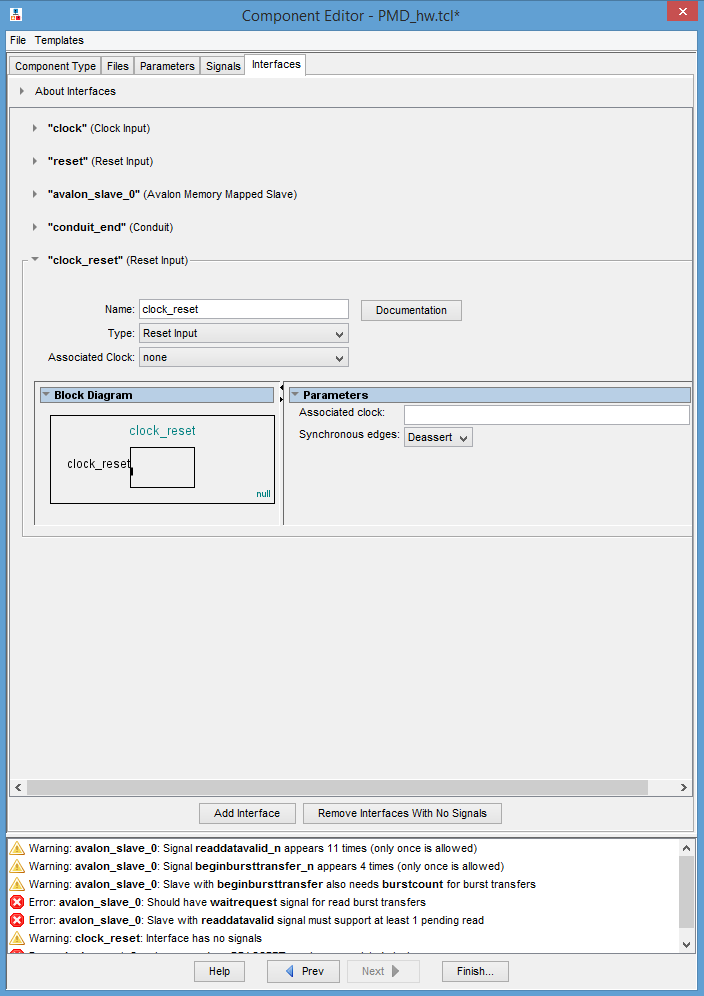


Here edit all the signals and assign them the properties as shown in the below figure,



Once all the signals are edited and set to the specified parameters, click on the next button.

Now go to the interfaces tab, where in select the “clock\_reset(Reset input)” interface and click on **Remove interfaces with no signal.**

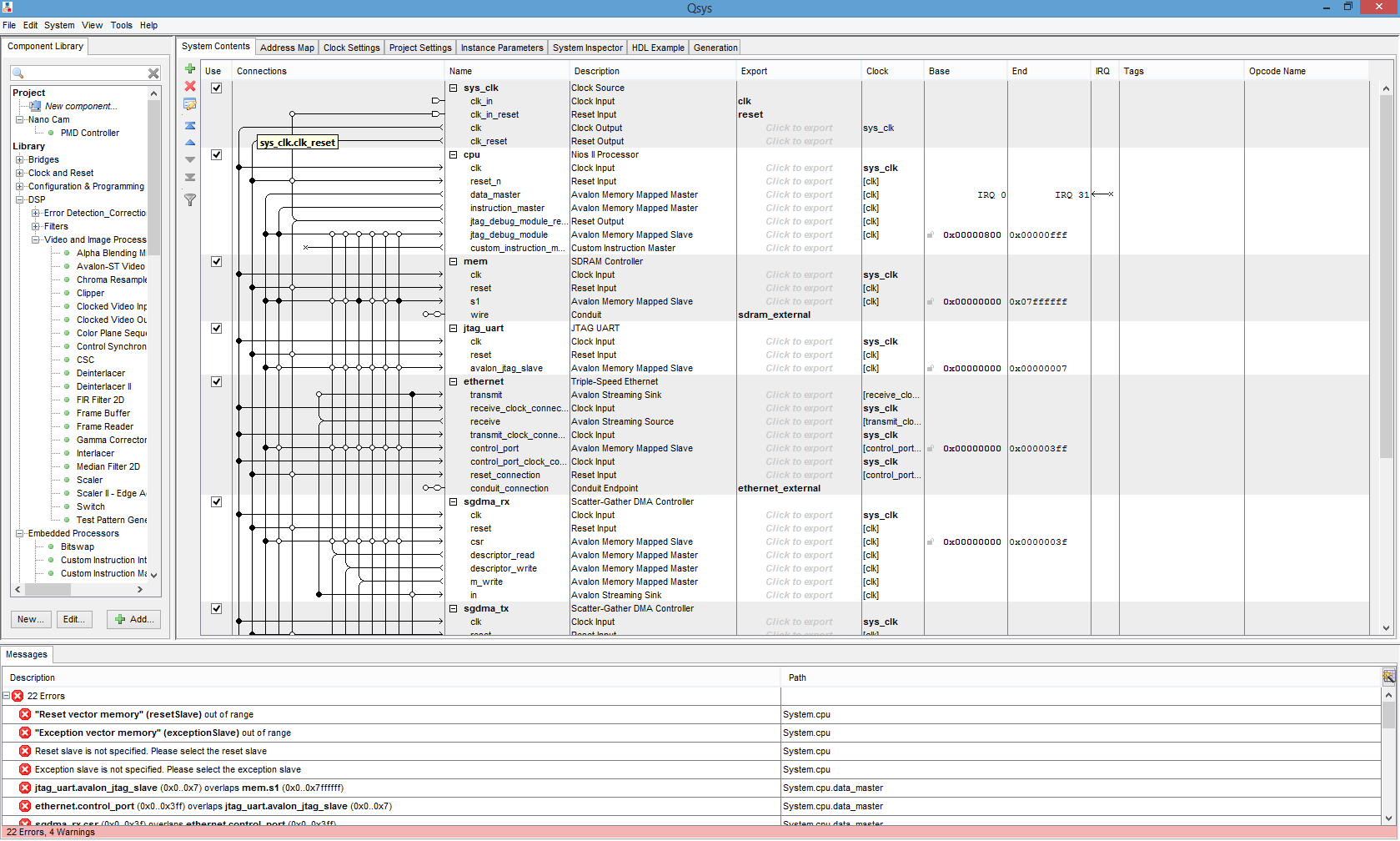


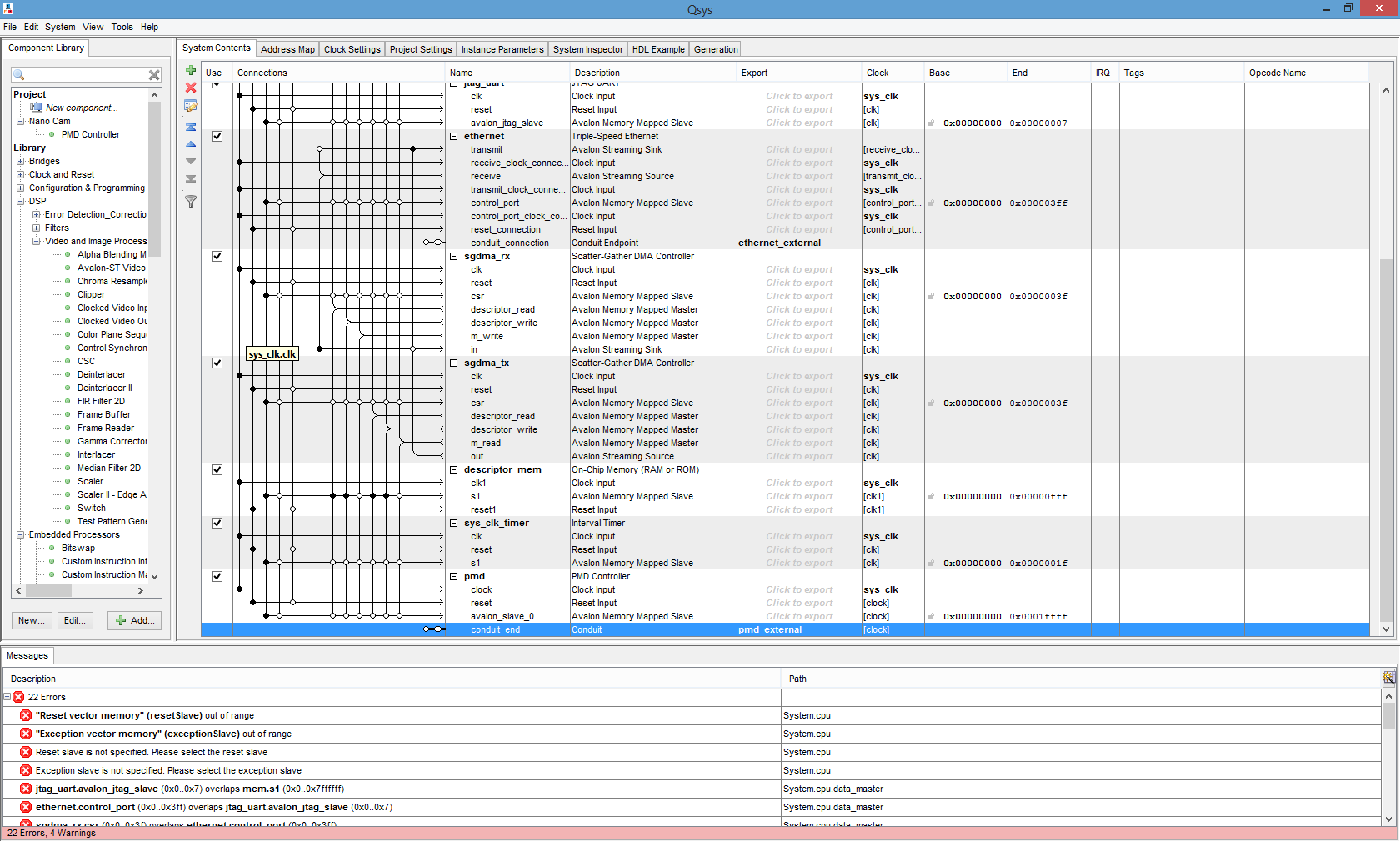
Click on finish to add the Custom Library to our system.

9. Once we have added all the libraries into the system, now we need to,

* connect the data and address busses for communication. To do this, we need to move the cursor towards the connections part of the Qsys window.
* Also **rename all the components** as they are present in the images in order.
* Set the **export status of few signals** and name them accordingly from the Export section as seen in images.

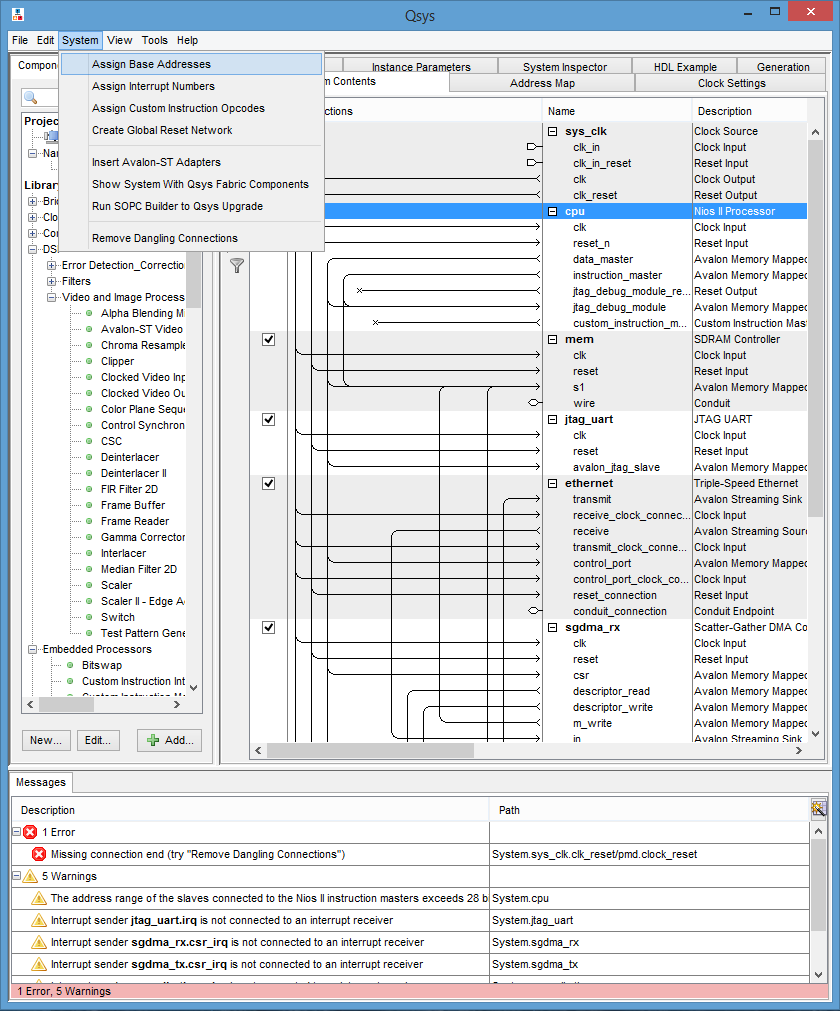
And click on the dots to form the connectivity as shown in the figures.



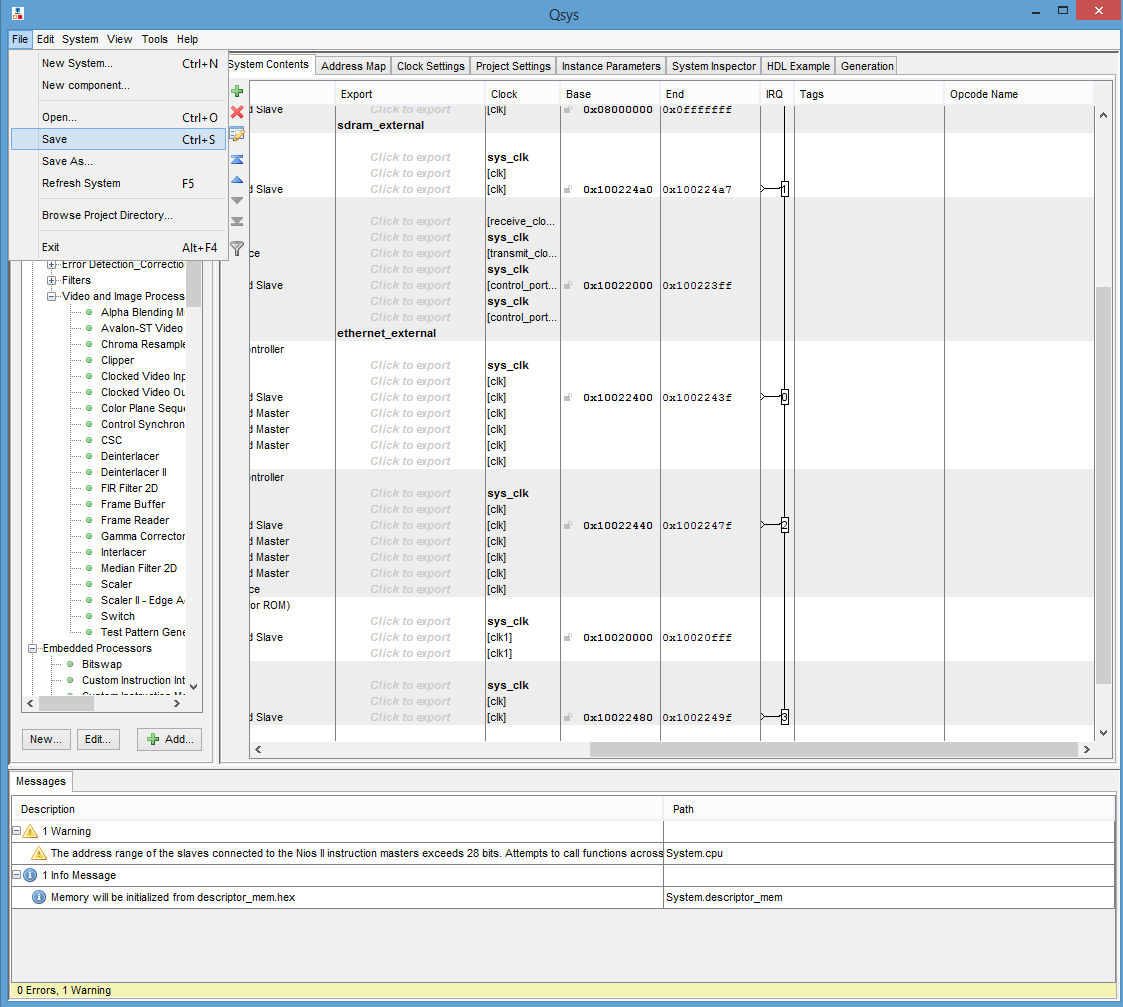


Once the renaming, connectivity & export assignment is done,

Click on the System → Assign Base Address.

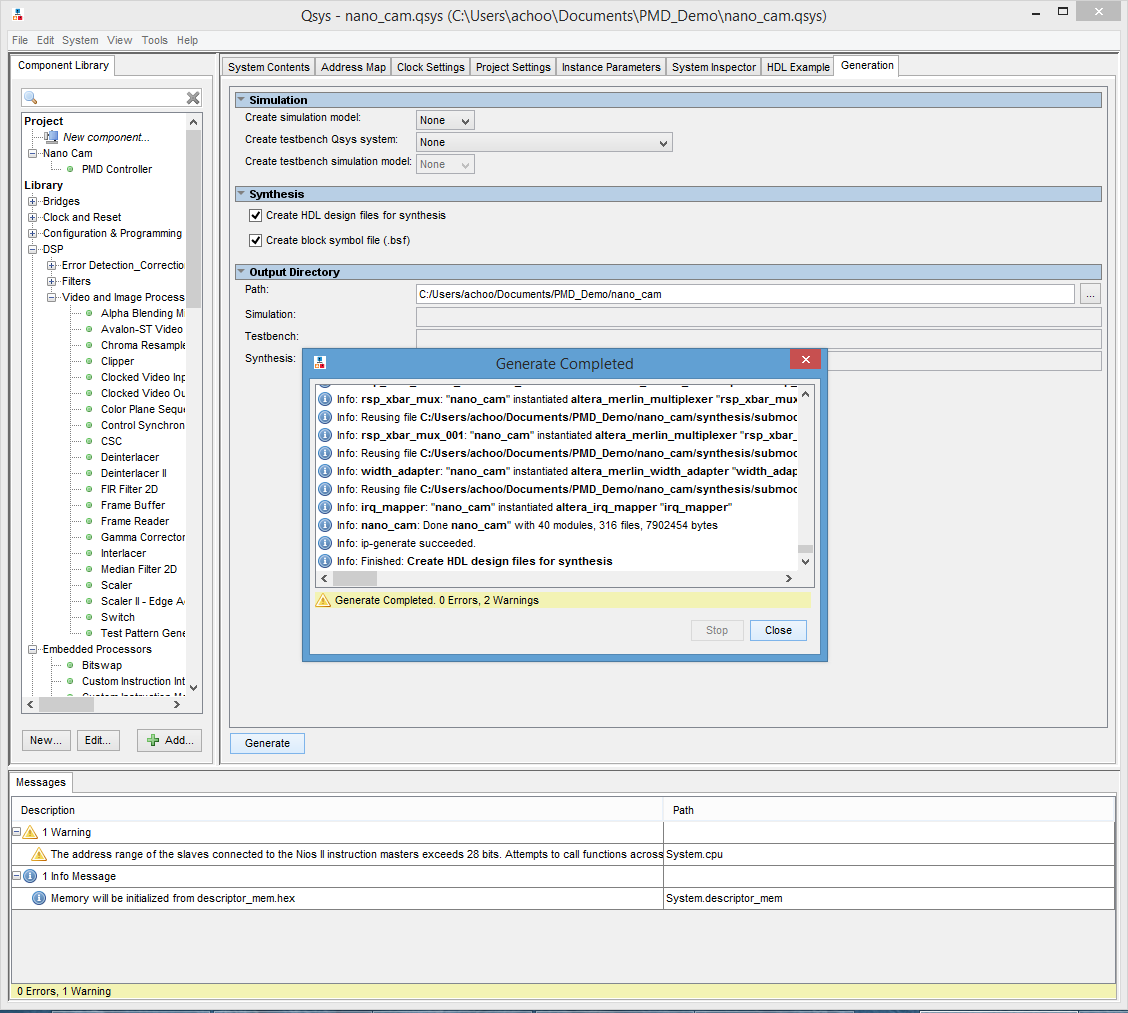


Now we need to assign the interrupt rank/status to few signals. To do this we will go to the IRQ section in the same Qsys window, and assign values (1, 0, 2, 3) in the tiny boxes there.



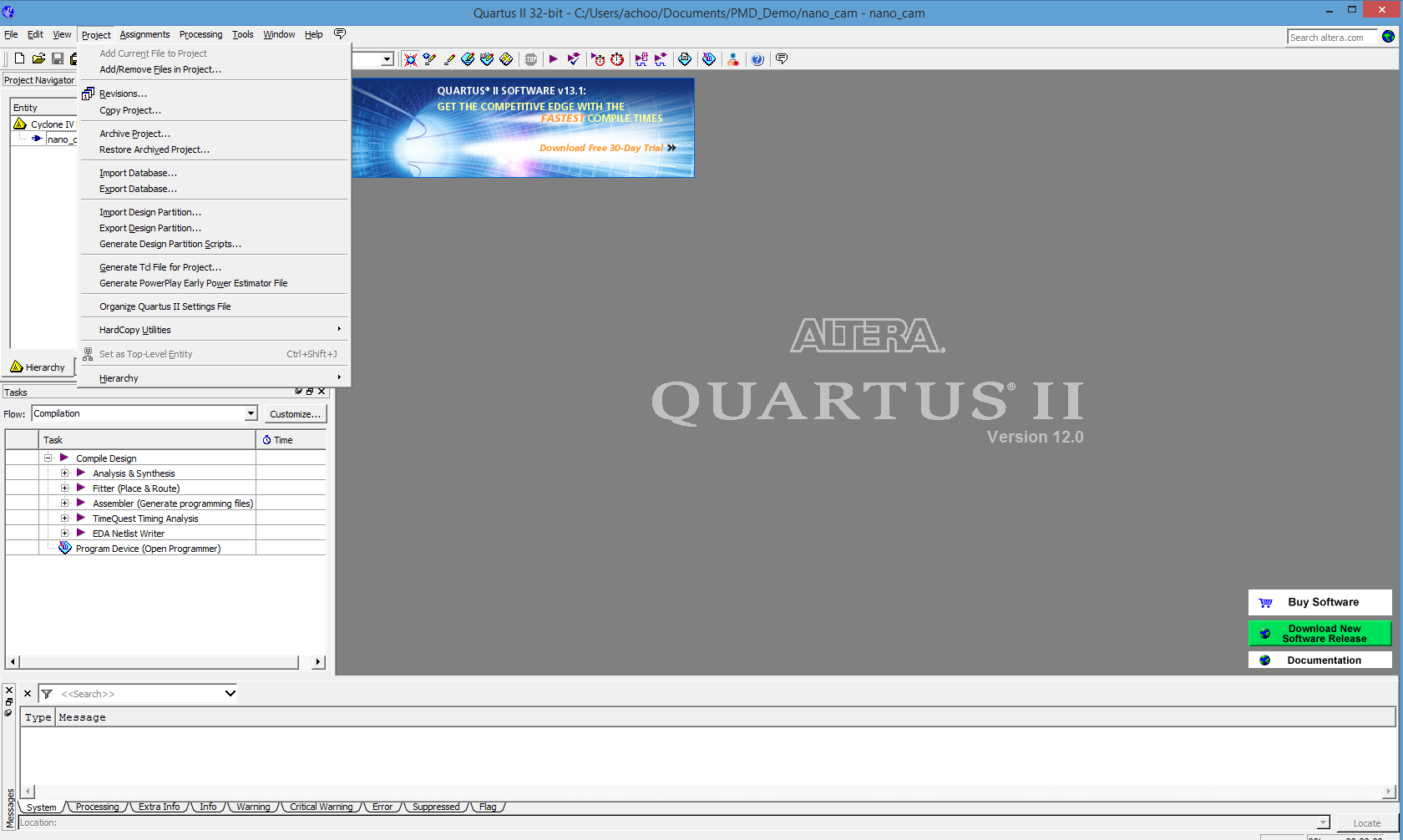
Once the interrupt is setup, save the system by going to File → Save or by using (ctrl + S).

Now go into the Generation tab in Qsys, and click on Generate to create the System!

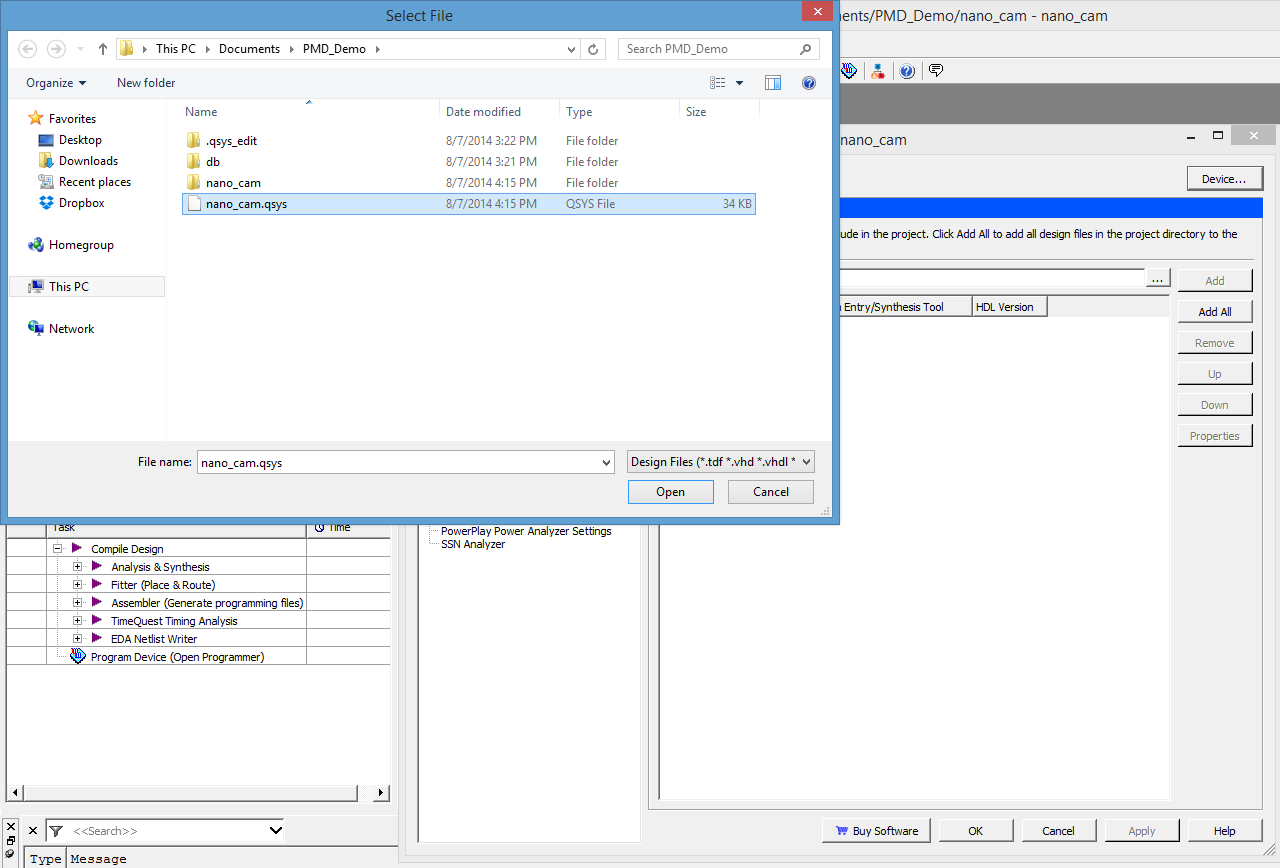


10. Once generate is complete, close the Qsys window. And go back to the Quartus ii project we had created.

11. Click on the Project → Add files into the project.

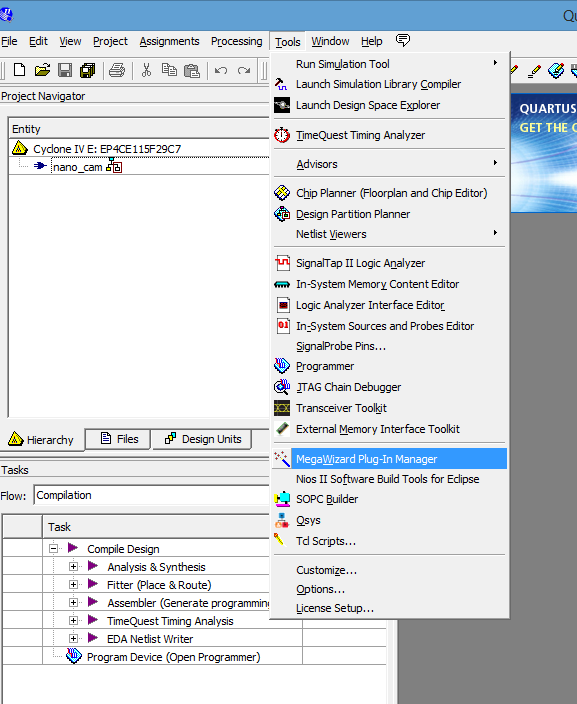


12. Add the nano\_cam.qsys file we just created from the previous steps. To do so, browse to the project location (Done by default).

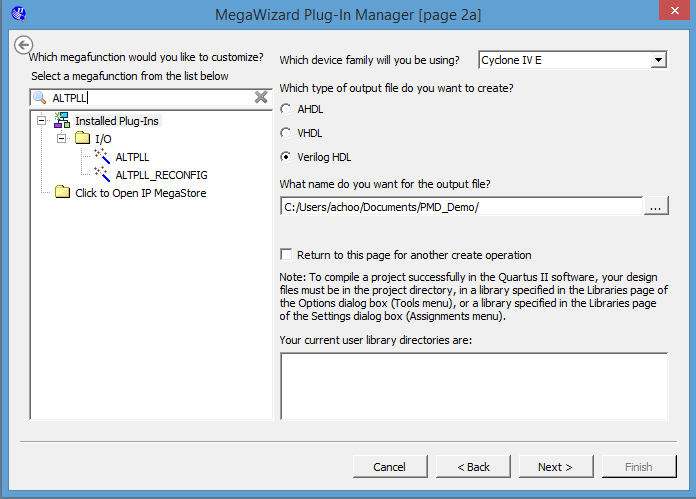


13. Add top.v file which is provided in the code database by the same steps.

14. We now have to create two PLL (Phase locked loops) to control the light and Sensor modulation. To do this, click on Tools → Megawizard Plug In manager → Create a new custom megafunction variable.

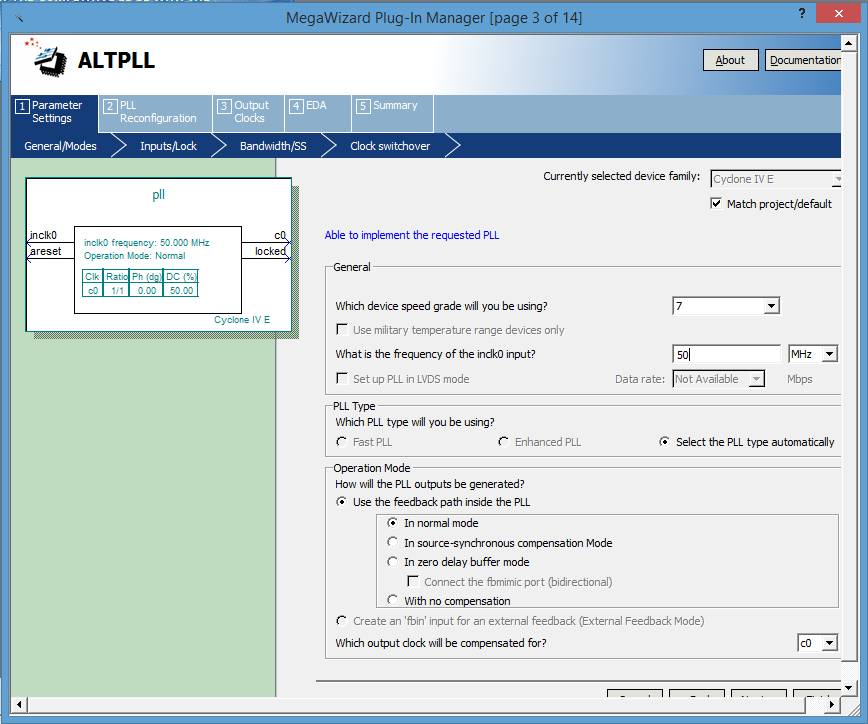


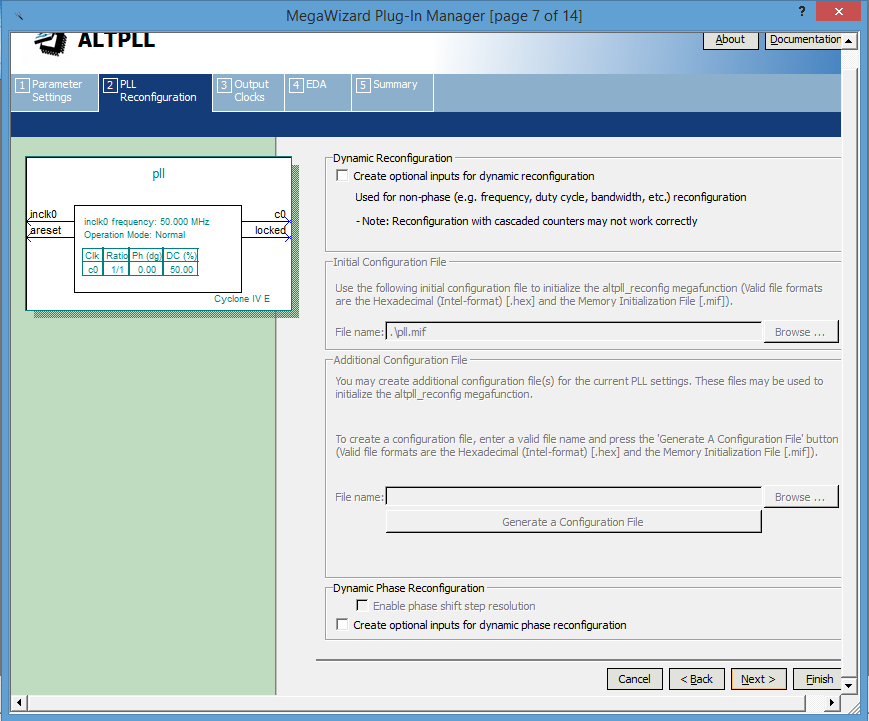
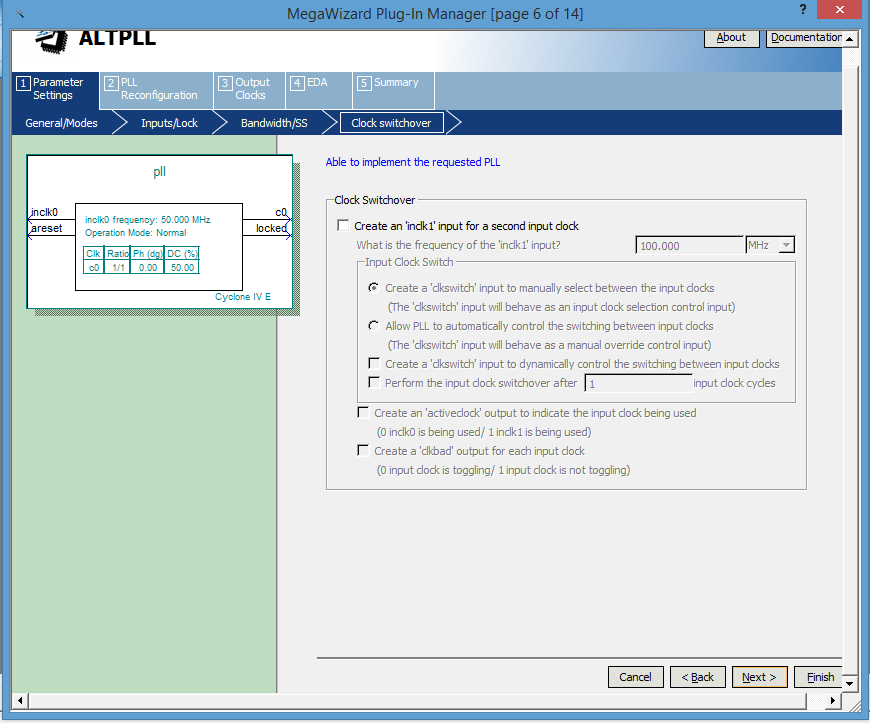
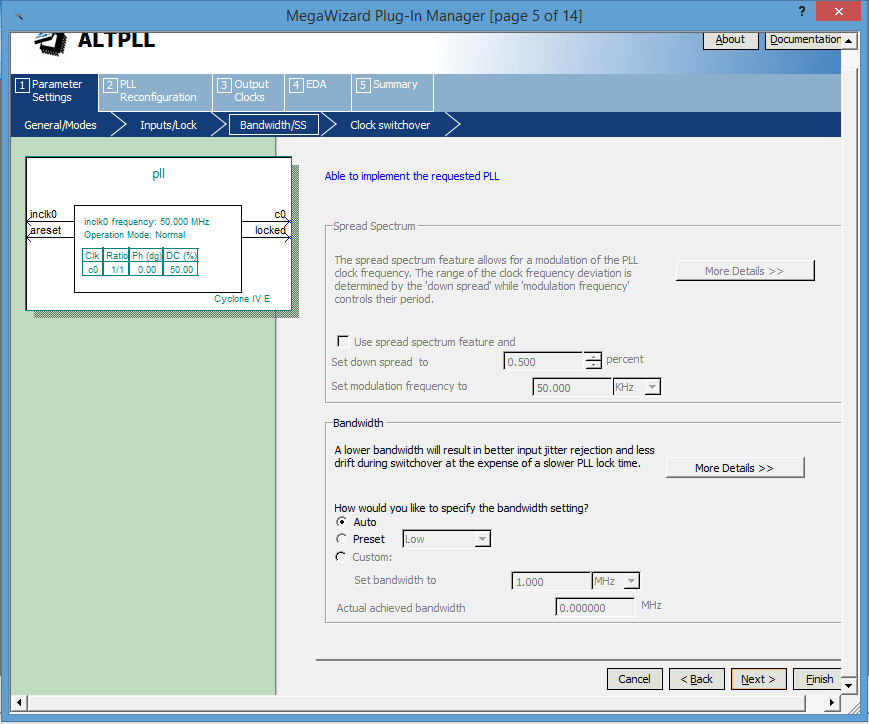
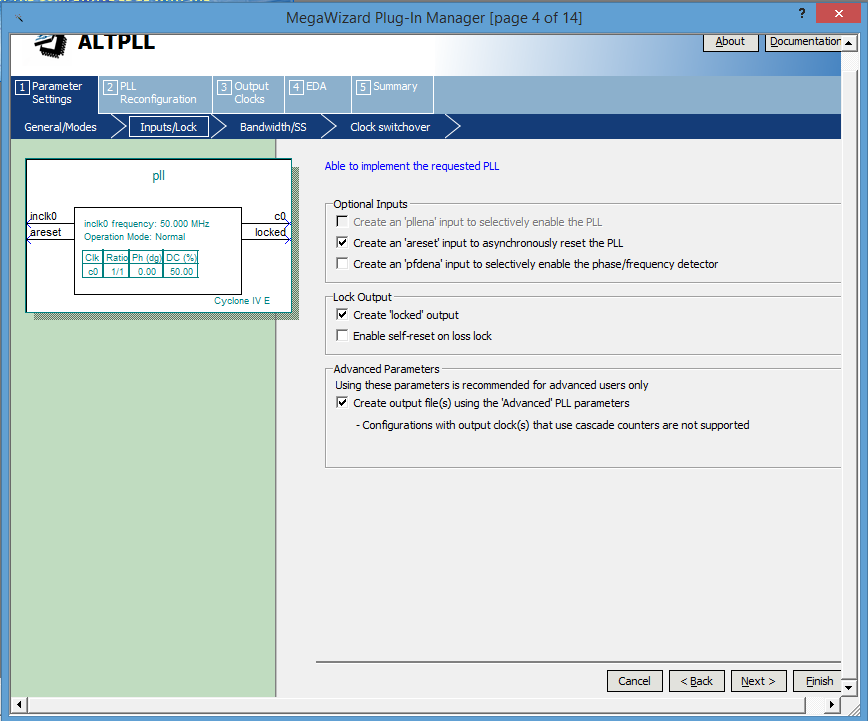
Search for ALTPLL,

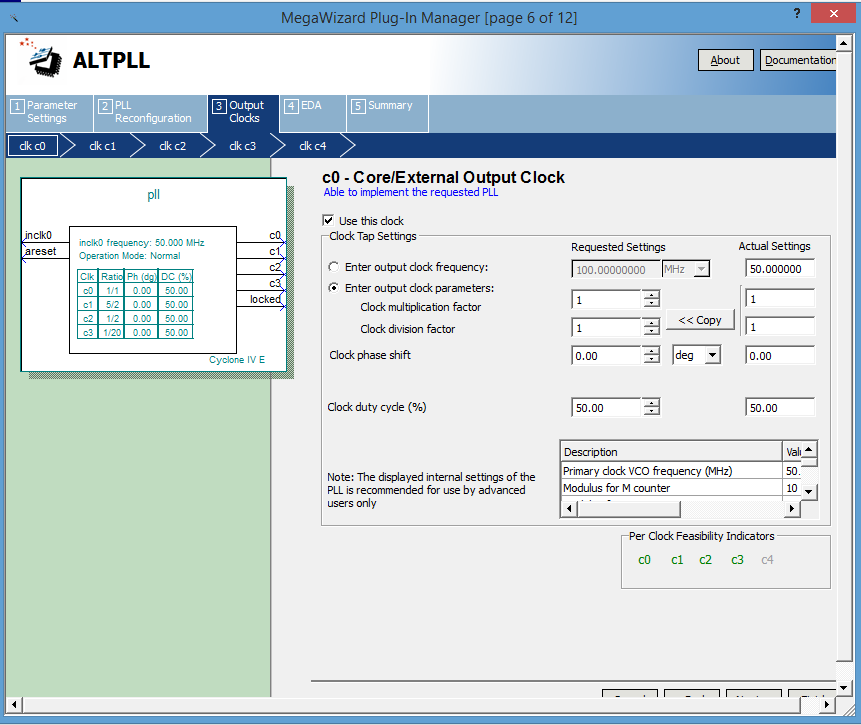


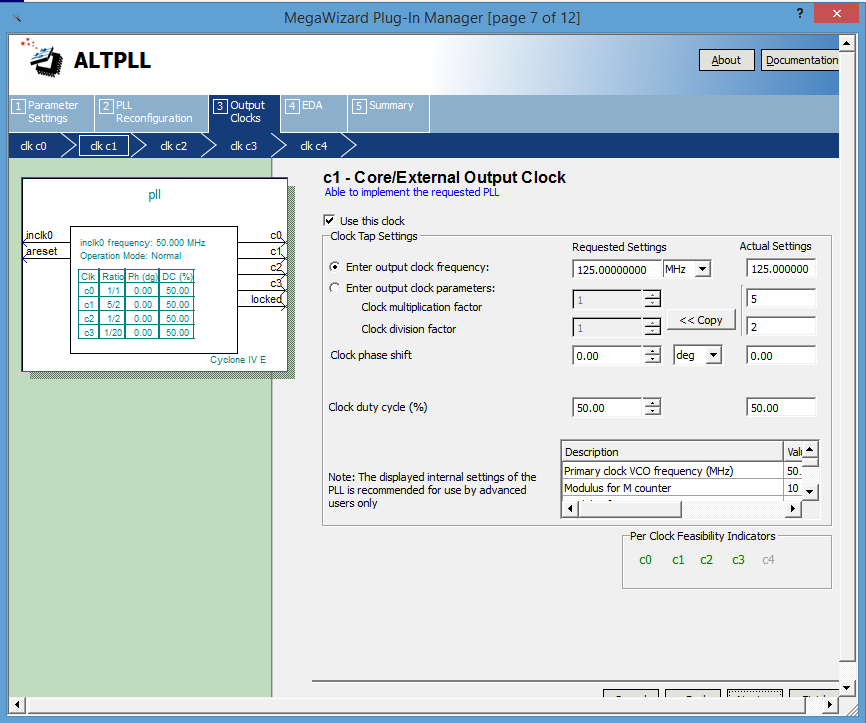
Specify the name of the output file as “pll” and click next.

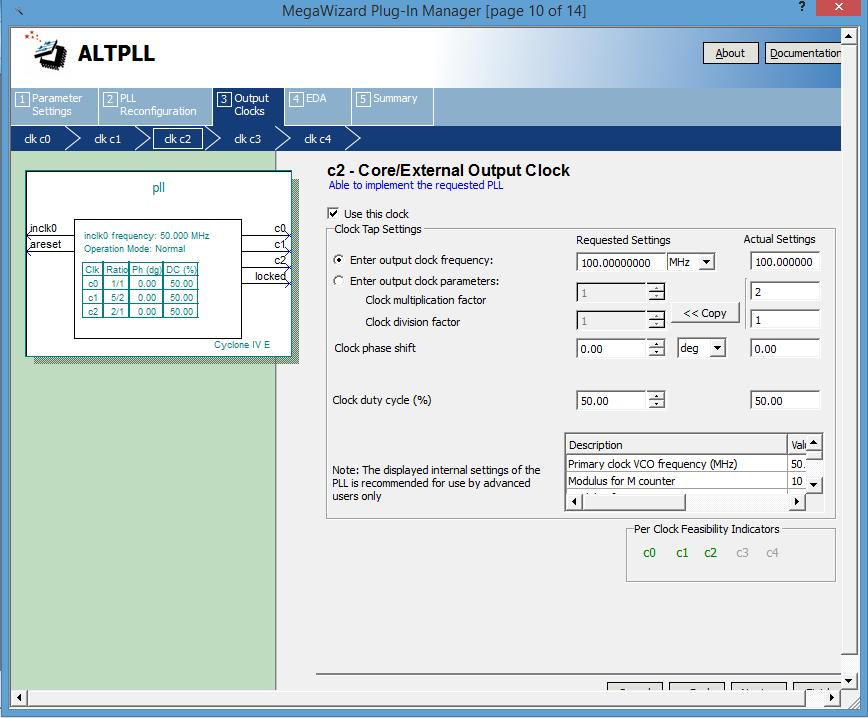
Follow the below images to navigate through each window and set the parameters likewise. Keep clicking on next to go into next window.

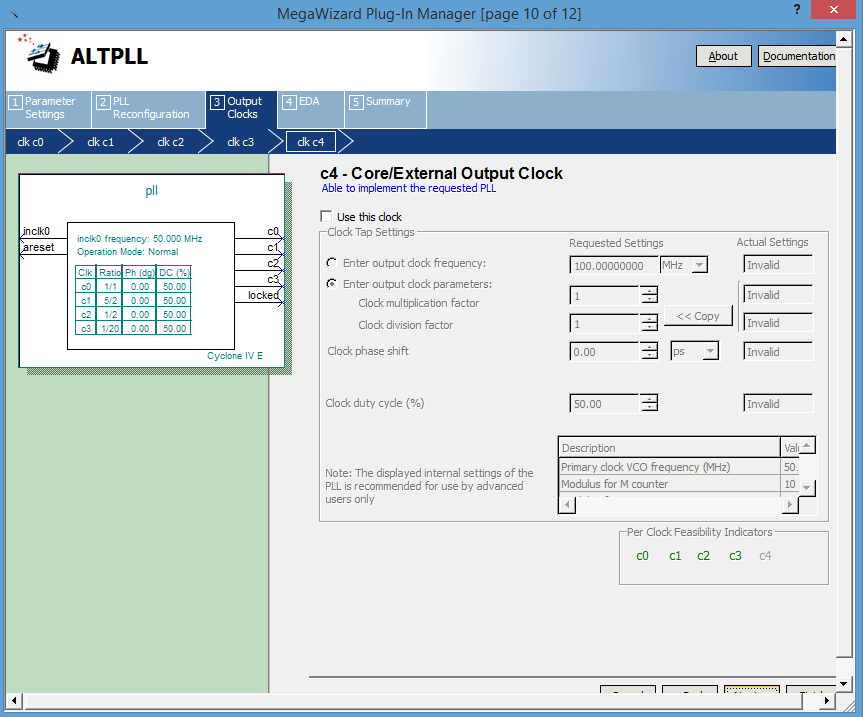
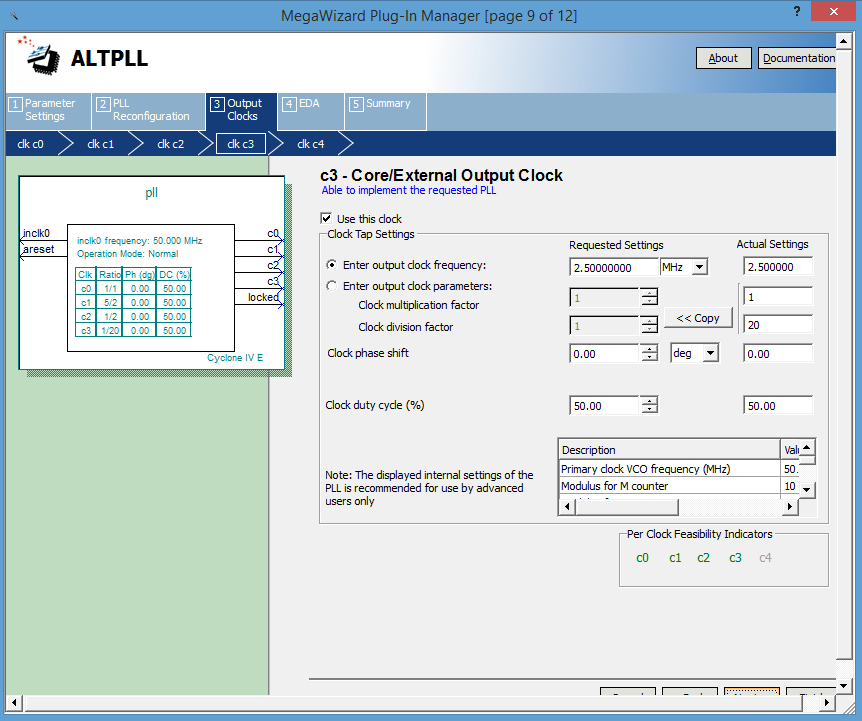


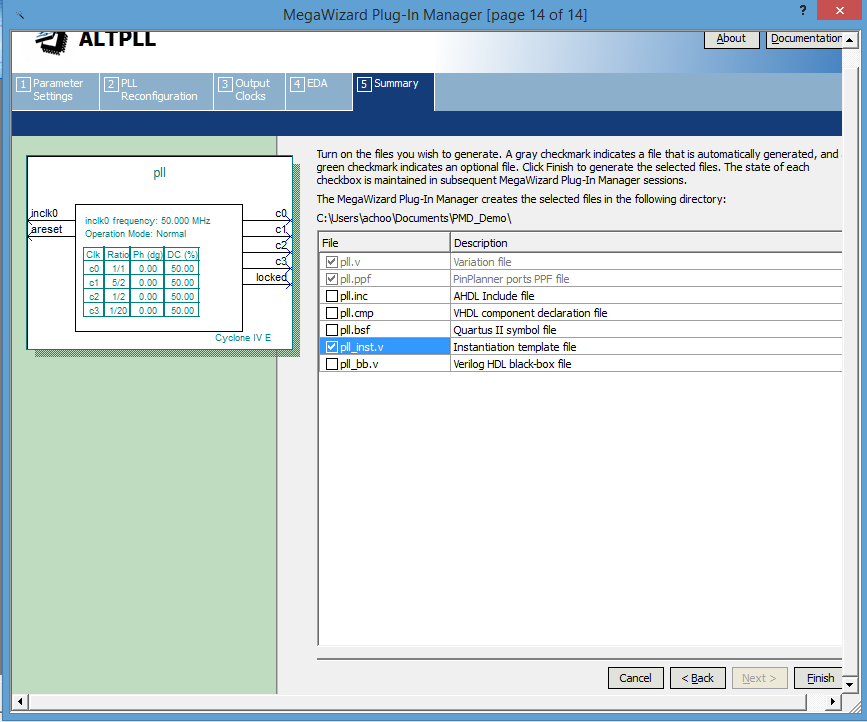
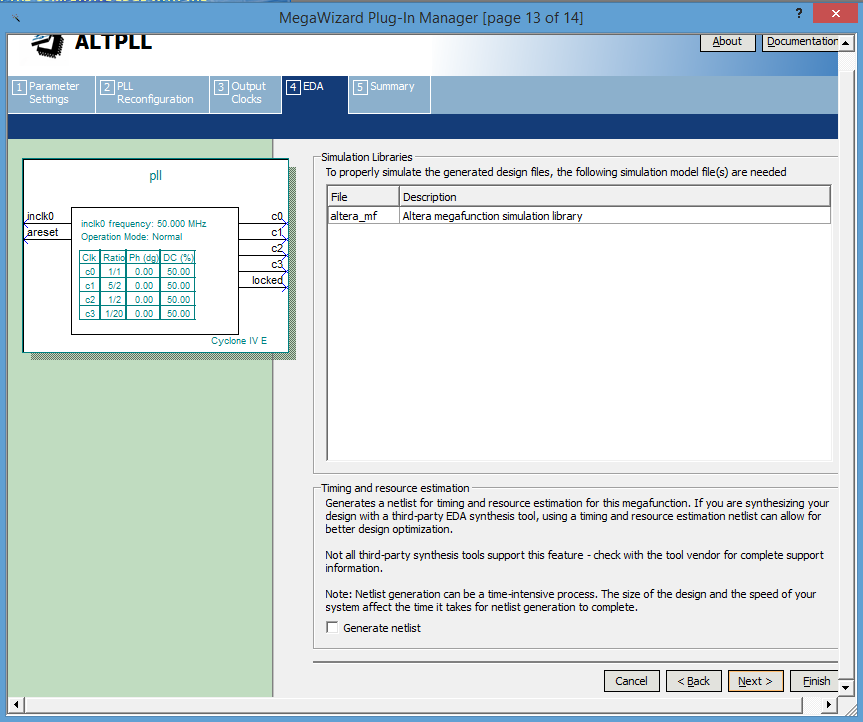










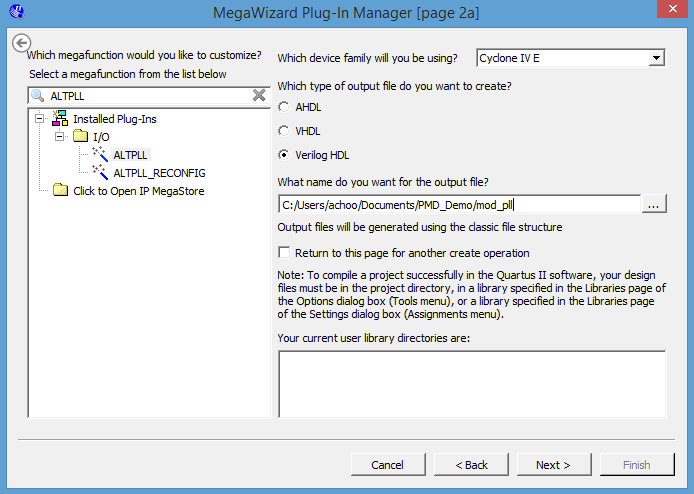


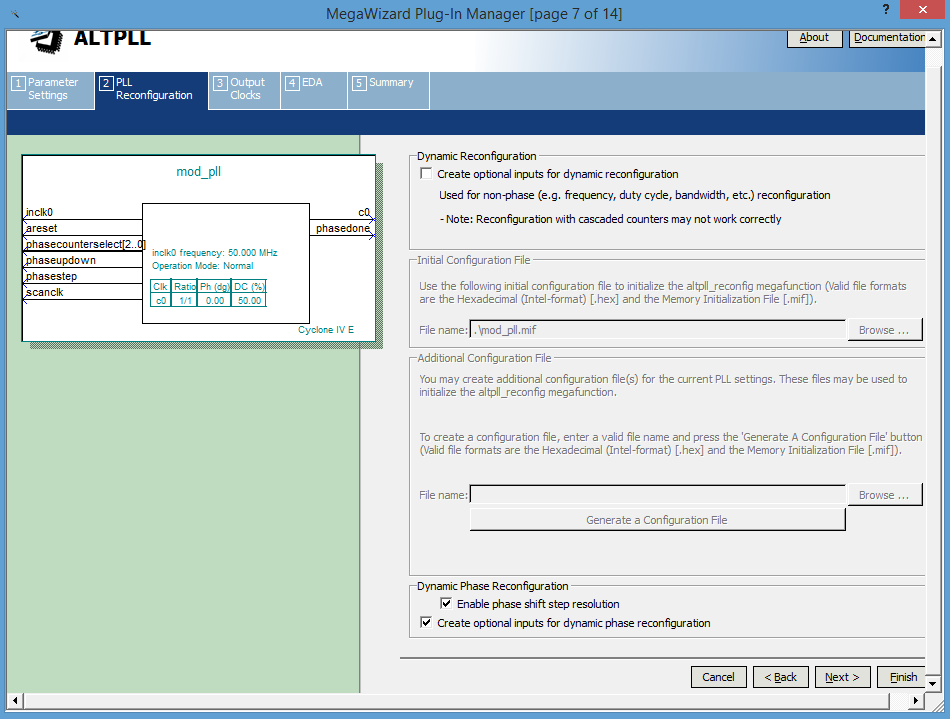
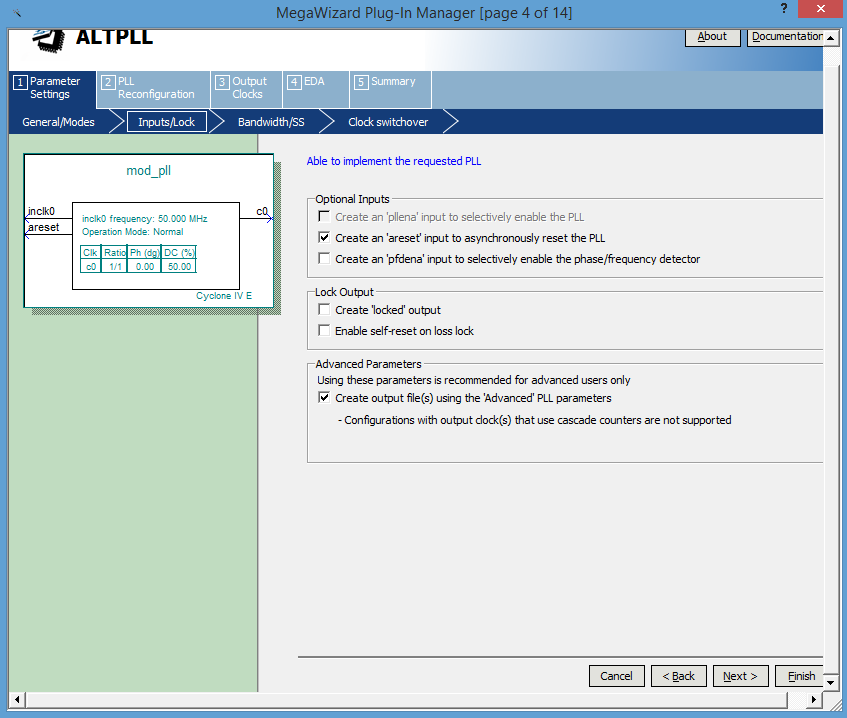
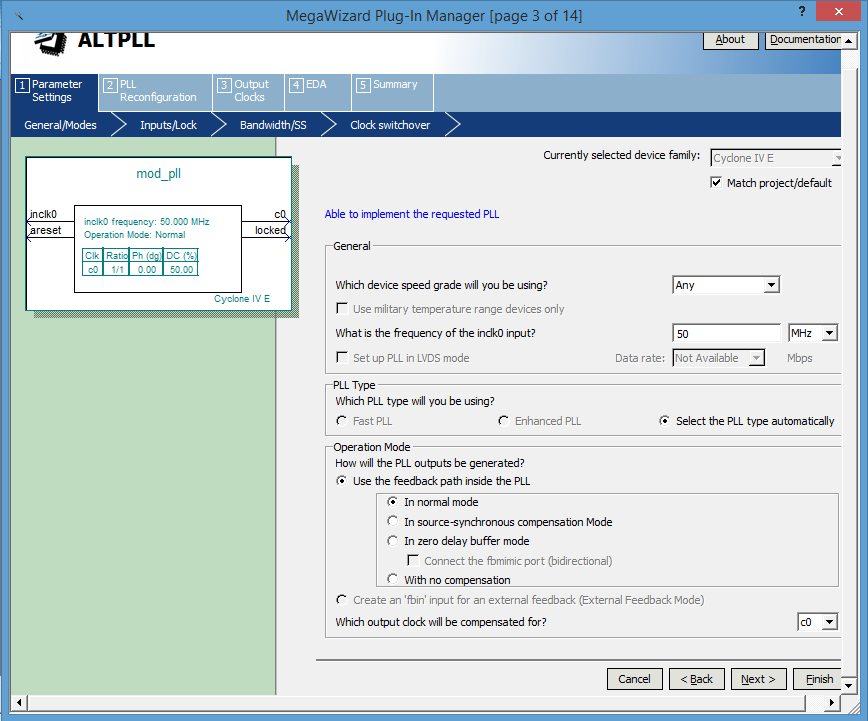
Click on finish to complete the creation of the PLL. **If a window pops up asking to automatically update the IP, click OK**.

Now similarly we have to create another PLL, named “mod\_pll”. This is to give a dynamic phase to one of the signal fed to the light source.

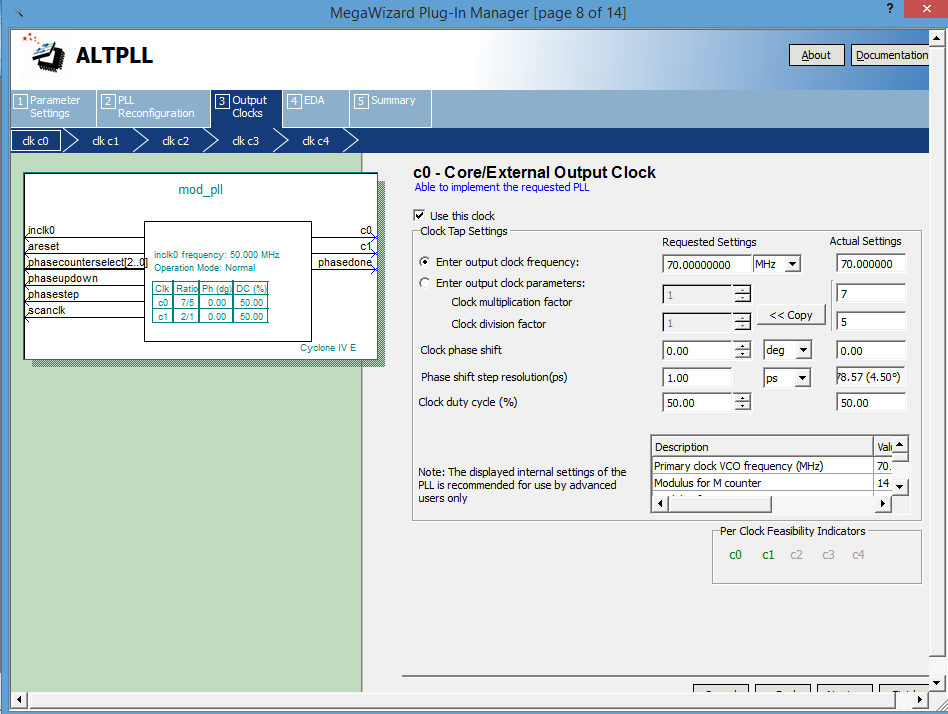
To do this, follow the steps depicted in the pictures below.

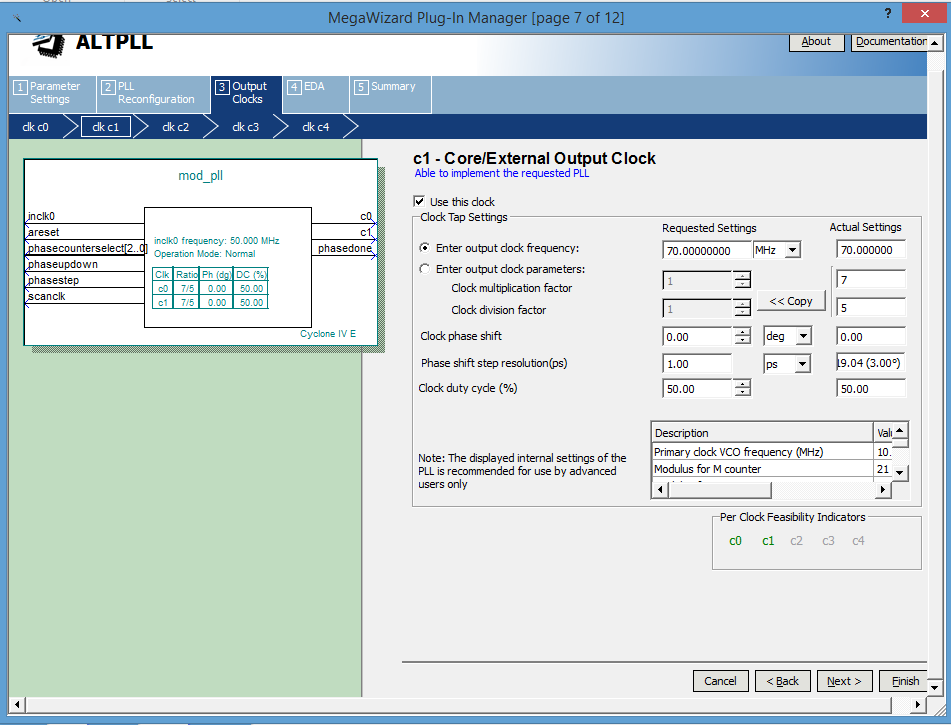
(Note that here only two output clocks are used, i.e c0 & c1).





Here we enable the Dynamic Phase reconfiguration and then the Enable phase step resolution.

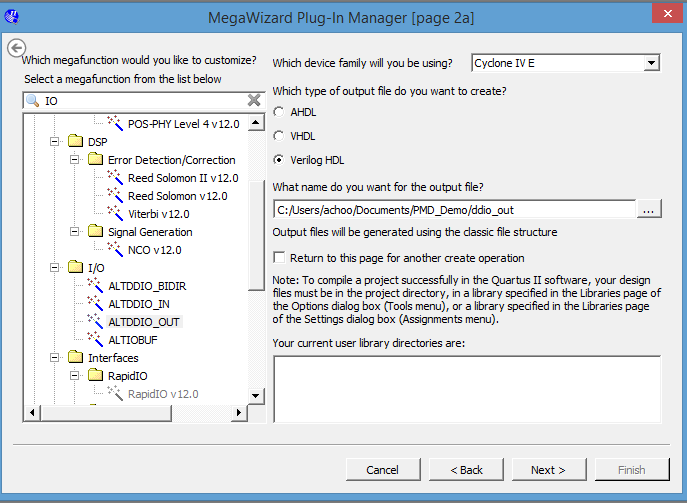




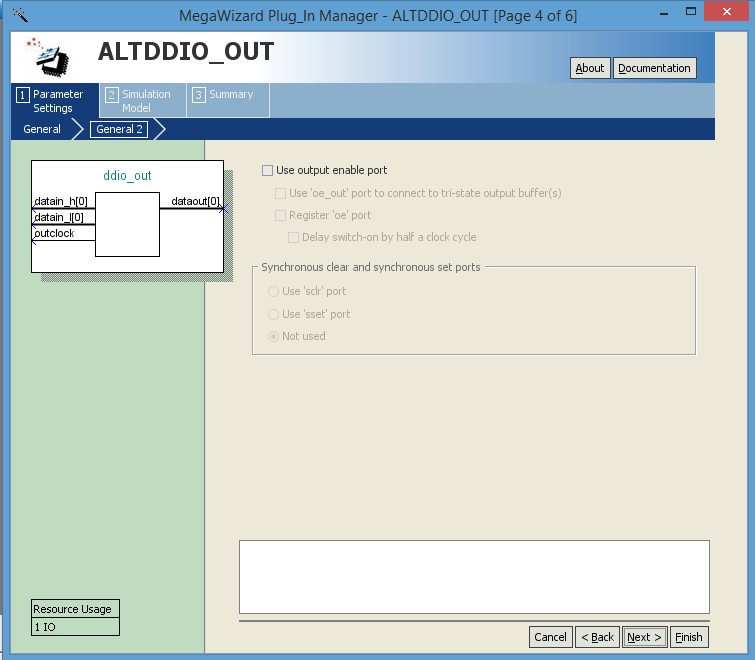
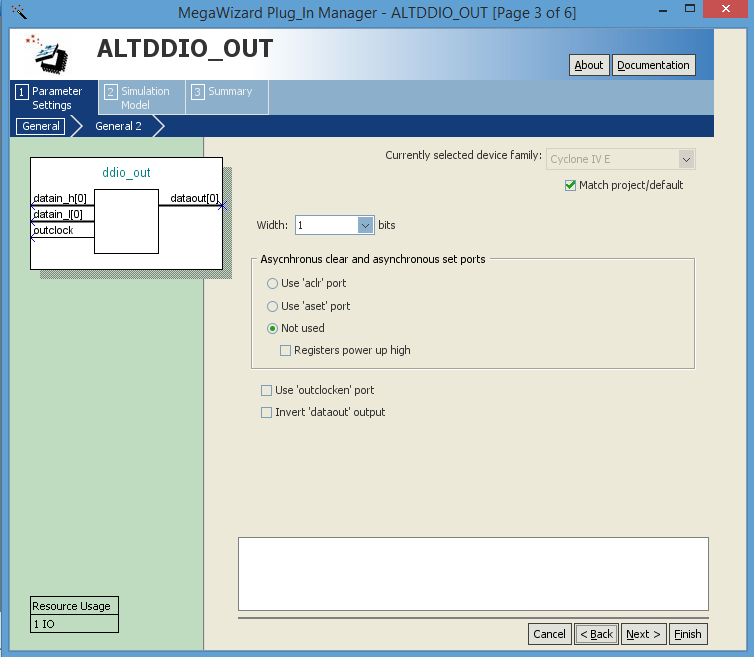
Click on finish to complete the create the PLL.

15. Now we need to add an IO extension as well for clocking an external pin. To do this, go to Tools → Megawizard Plug In Manager → Create new custom megafunction variation.

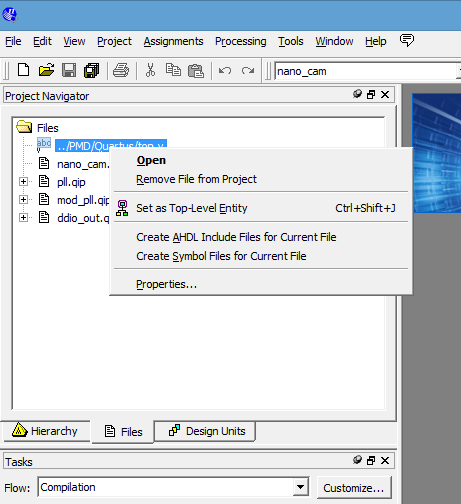
Now search for IO, and select “ALTDDIO\_OUT”.



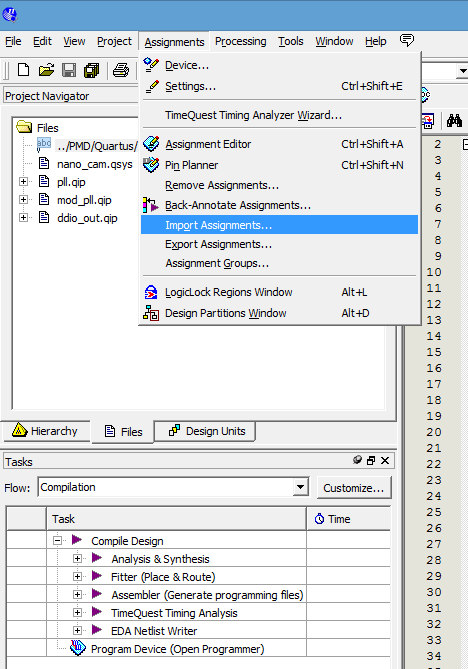
Click on next and follow the images below to complete the whole setup.



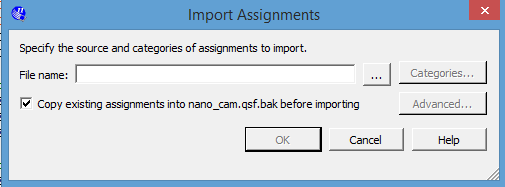
16. Click on the files tab in the project navigator. And set the top.v as the top level Entity.



17. Finally go to the Assignments → Import Assignments.



Browse to the nano\_cam.qsf file which is provided in the code database and click open.



Once this si done, we are ready to compile and program!

18. Click on the Processing → Start Compilation.

This takes around 5-6 minutes to compile and after the we are good to upload and capture some cool depth images!!