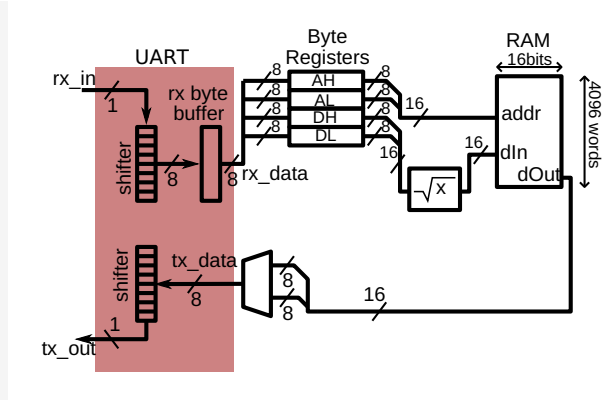
HW 6

Objective

In this project students will learn to use generated IP Cores (block RAM and a CORDIC processor for caculating sqart root). An essential skill to develope is the coding of a control FSM to interface modules. You will also practice making use of parameters in a design to create effecient simulations.

Below is the Datapath, not including control signals (the registers AH,AL,DH,DL and output MUX may be embedded within your statemachine if you so desire). You must identify the status and control signals and create a statemachine to control them.



Due Dates

- Due Monday Dec 4

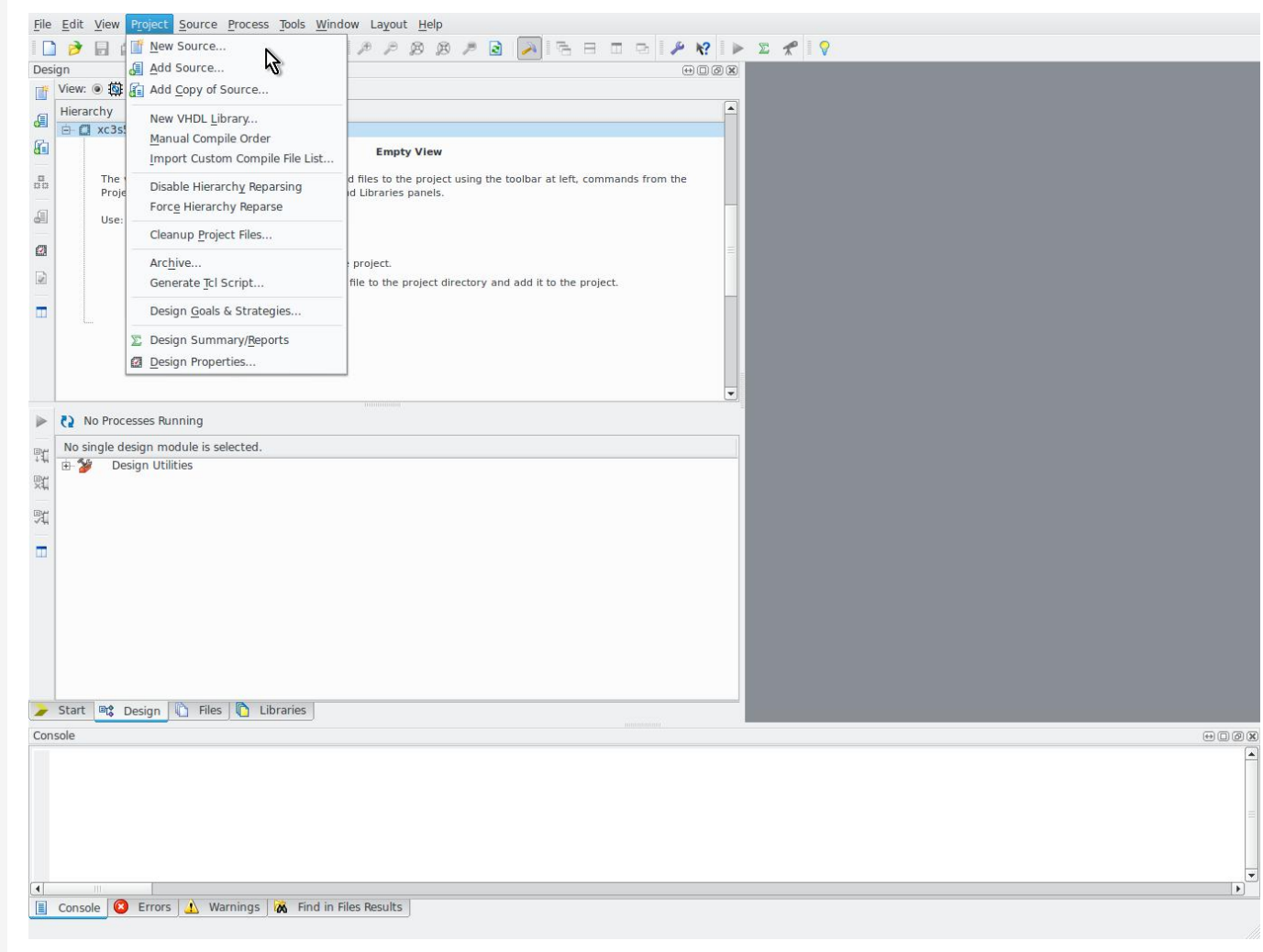
Requirements

- Your design should access 4 btyes from the serial port to initiate action
- Your implmentation should assume 4 bytes will be sent: ADDRESS-HIGH, ADDRESS-LOW, DATA-HIGH, DATA-LOW
- After 4 bytes are received your design should
  - read the contents of the RAM and send the HIGH-BYTE, then LOW-BYTE back through the uart
  - send the data bytes through the CORDIC Sqrt processor to compute the result
  - the sqrt result just computed should be stored in the RAM at the same address that was just read
- your hardware design should use 115200 buad rate and require NO modification to compile
- you must include a simulation of the top-level design and discuss it in your report with a parameter to control baud rate. It is srongly recommended to consider modify the buad rate for the purpose of top-level simulation -- this should be conrolled by a SINGLE parameter set in your test-bench. You should not modify your UART implementation files.
  - if required you can instatiate other "mini-versions" of hardware to speed simulation (such as a smaller ram, or samller CORDIC processor) but your hardware implemenation cannot be based on these
- **your FSM controller must as much as possible rely on use of status and control signals to manage the datapath timing, you should not rely on "fixed" waits. This represents a better abstraction.**

Generating a multi-cyle, pipelined squart root CORDIC processor

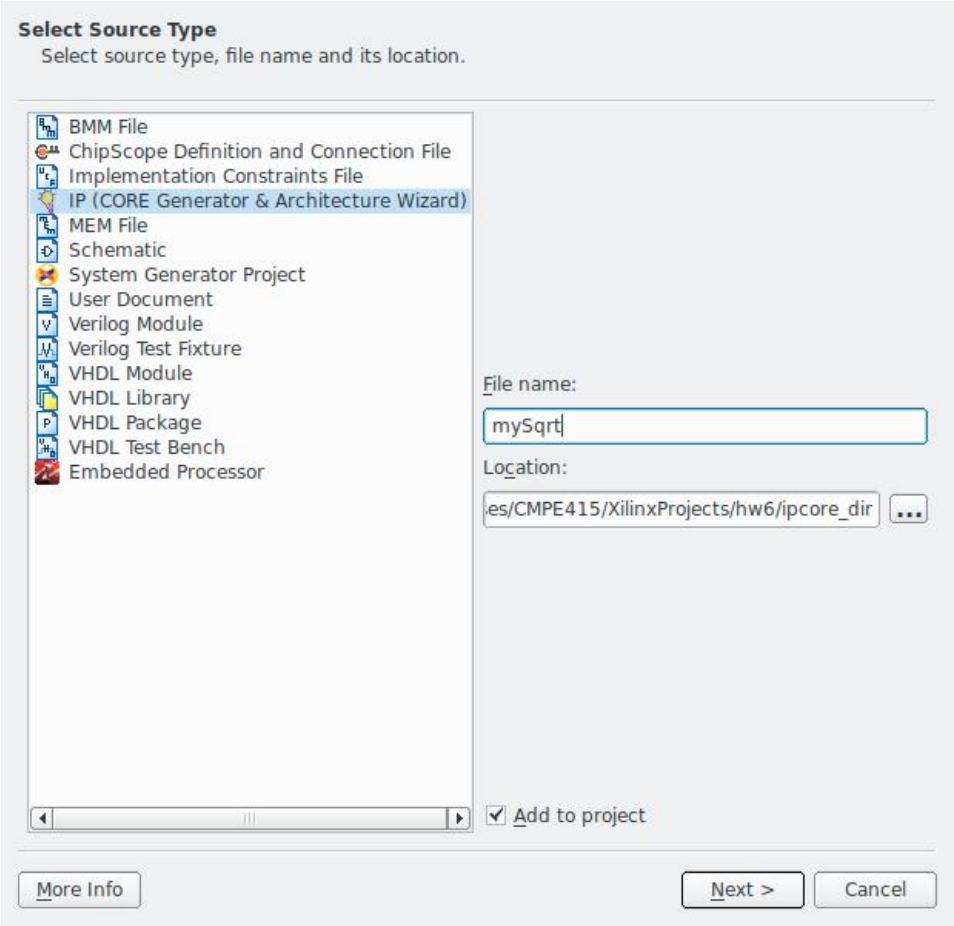
Follow these steps to generate a multi-cycle pipelined square root processor

Add a new source to the project:



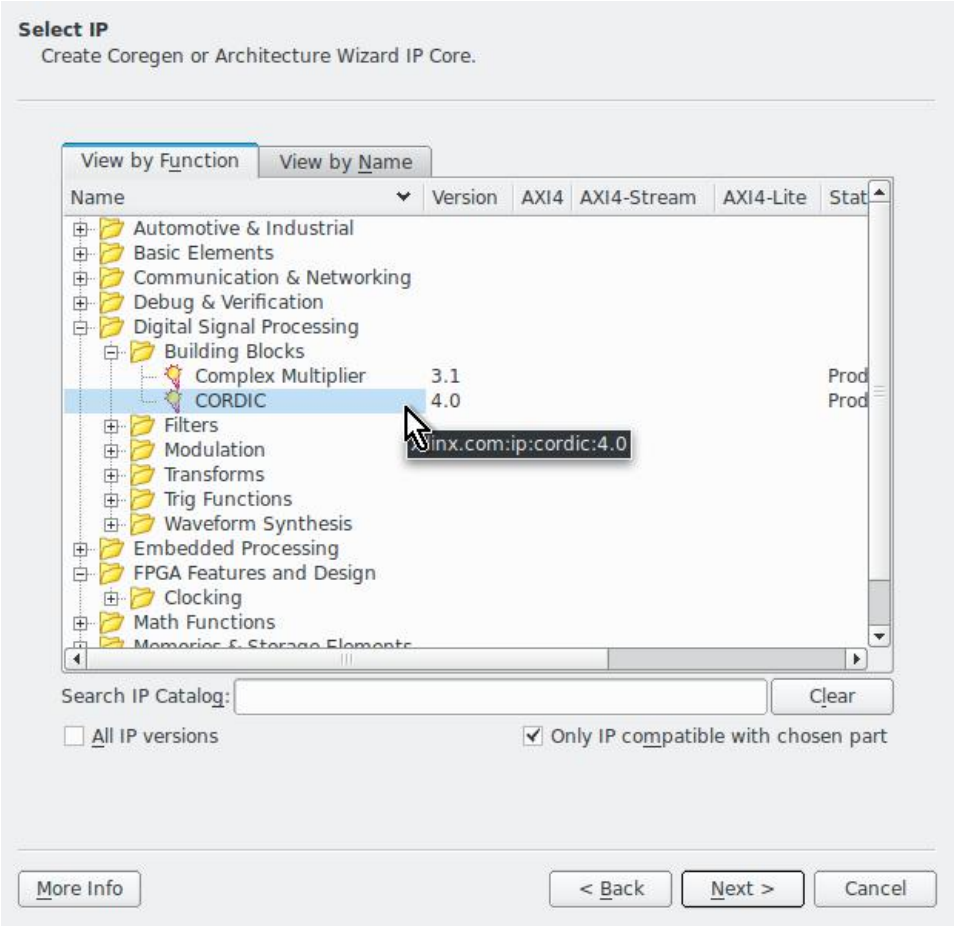
Step 1

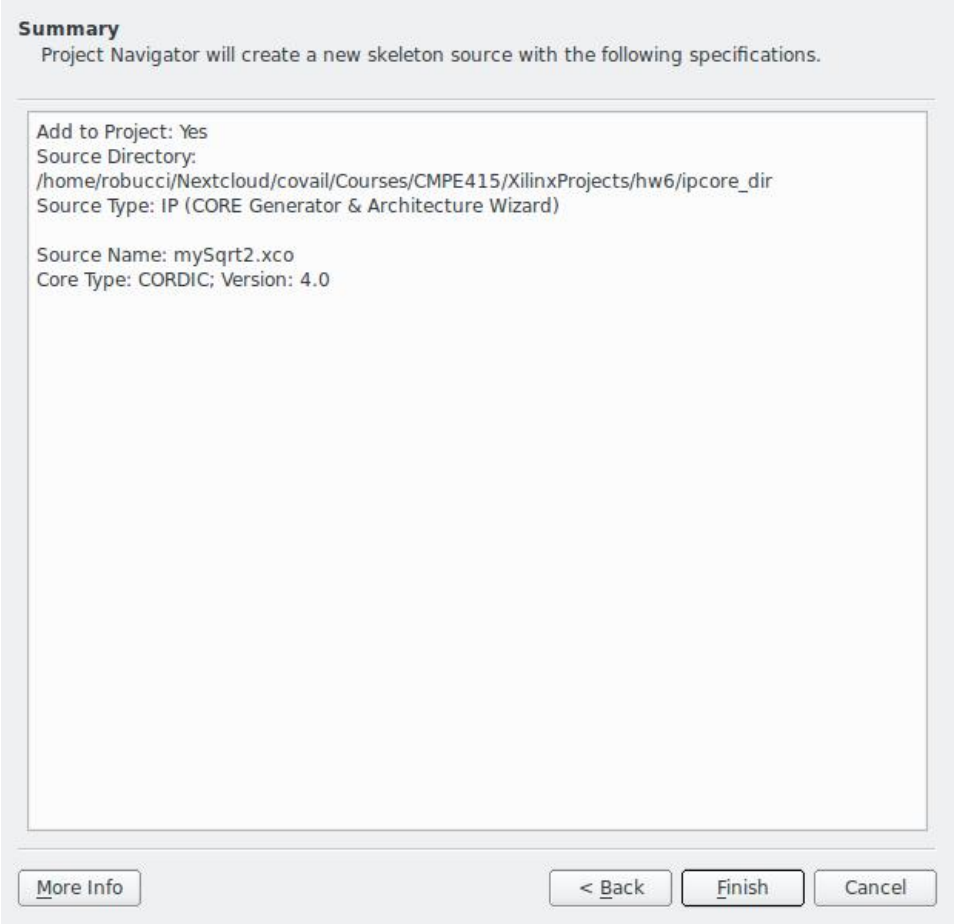
- Choose (IP CORE Generator ...)
- and provide a module name,
- select to automatically add to project
- click Next:



Step 2

- Find the check box and enable "Only IP compatible with chosen part" to filter the list of cores to those supported by the tool for the choosen FPGA in the main project
- Select the CORDIC core



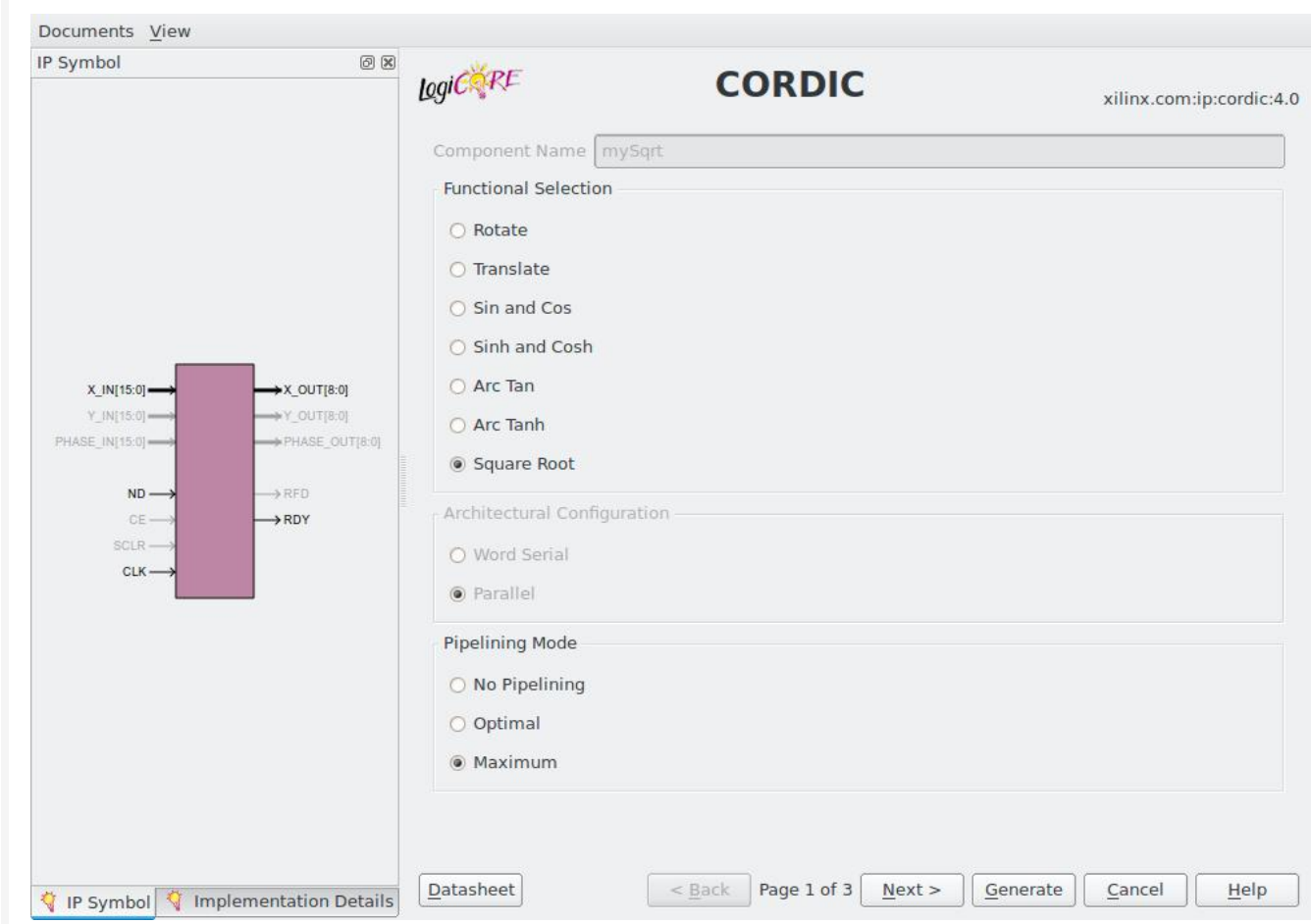


Step 3

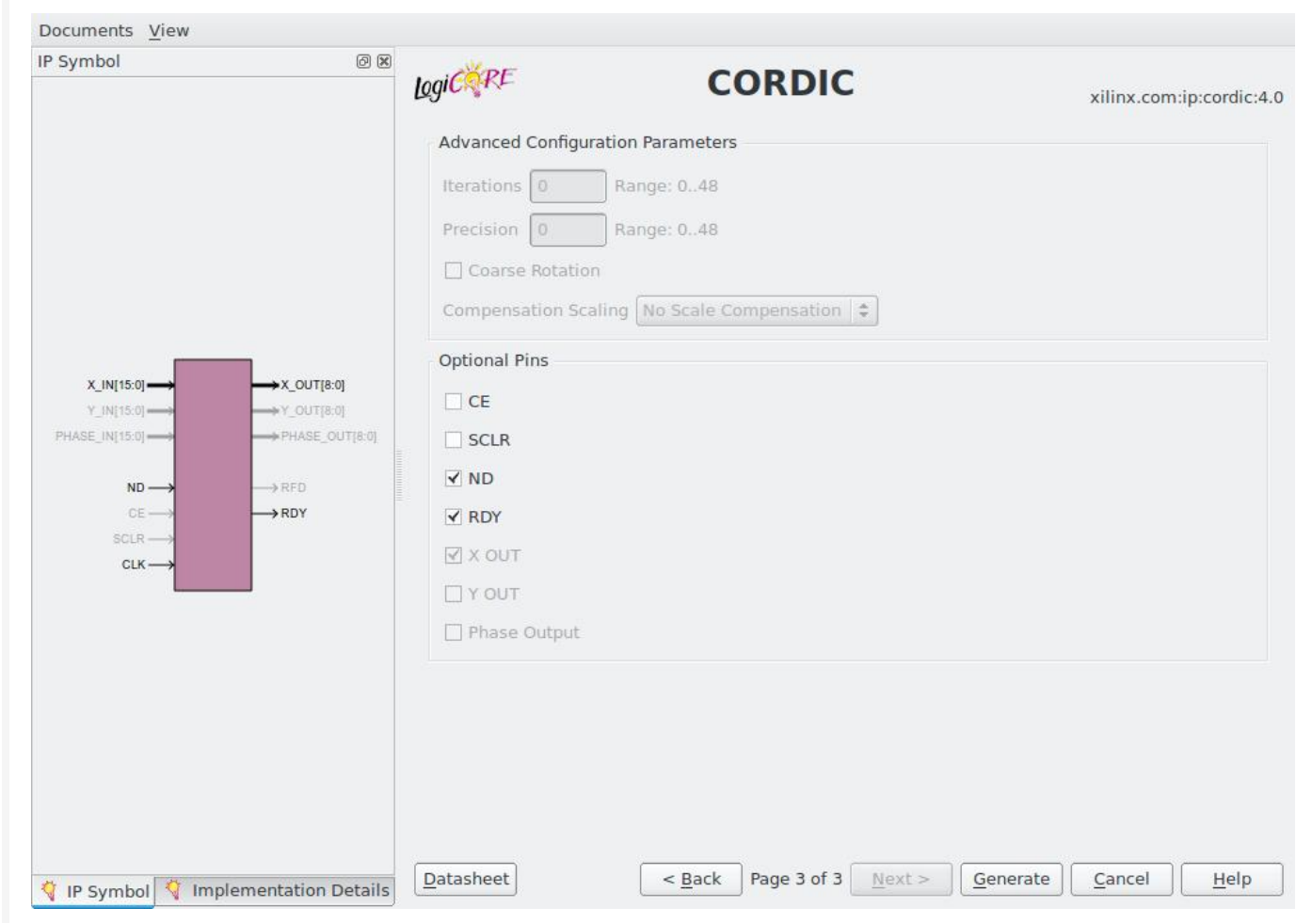
You must now configure the options Documentation for the CORDIC core explaining the options and how to use the generated core can be found here https://www.xilinx.com/support/documentation/ip_documentation/cordic_ds249.pdf

In each image, not every option may not be discussed but be sure to check and match ALL OPTIONS to what is depicted

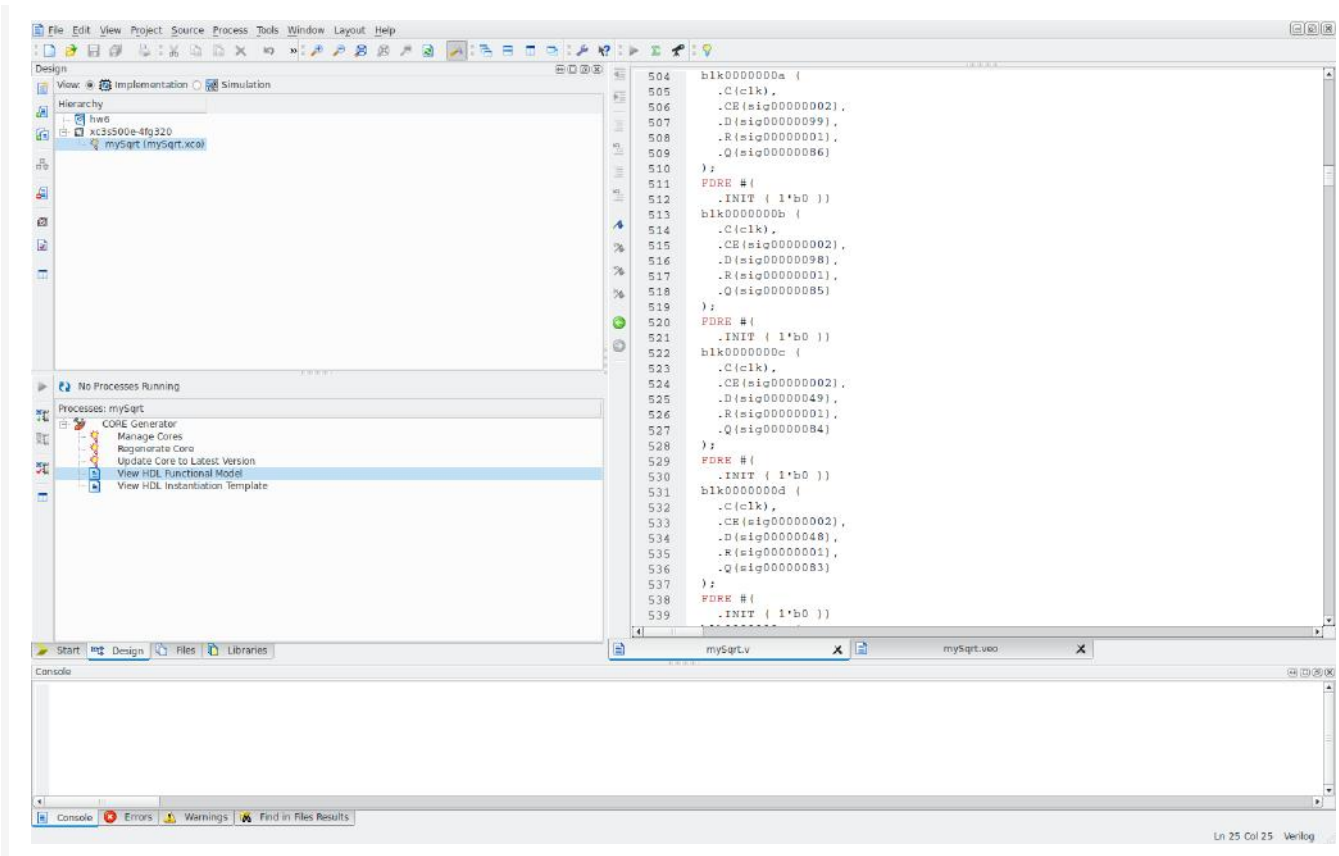
- Choose the Squart Root option



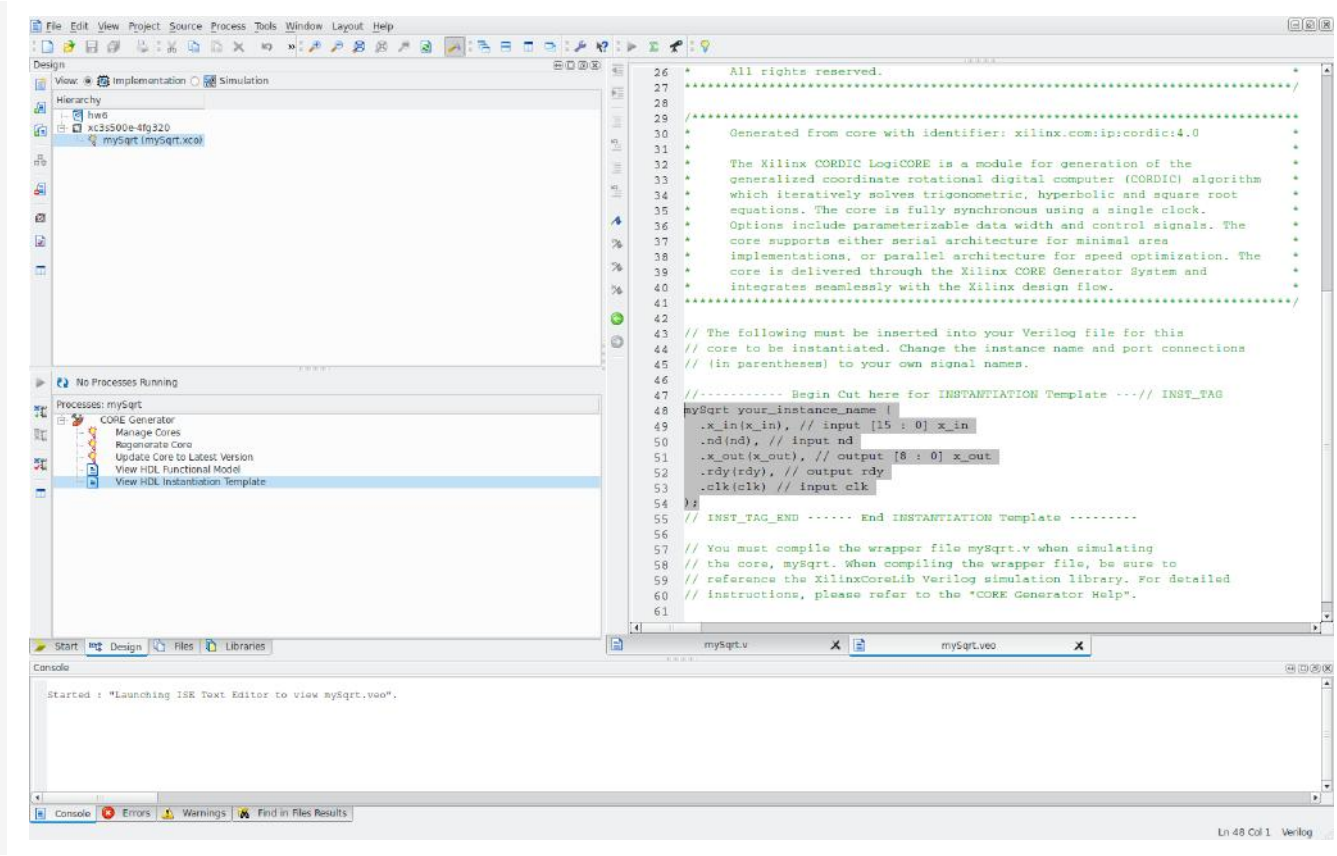
- Configure as Unsigned Integer instead of Fraction. This assumes integers instead of fixed point representation



- Finally click Generate --**The generation process may take some time**
- You can view the core functional model which can be used for simulation:



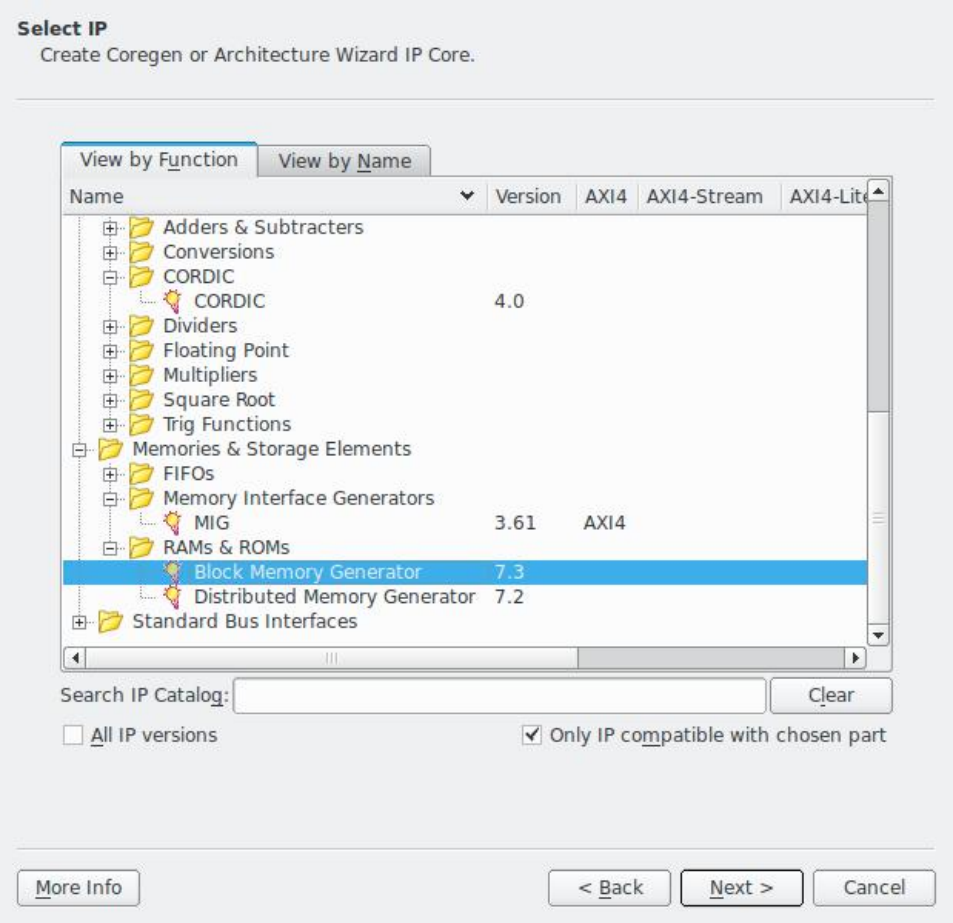
- You can view the template that shows how to instatiate the generated module in your design:



Generating a RAM

We'll generate a ram using the IP core generator. We'll choose to use block RAM (uses speciallized blocks of memory hardware on the FPGA). Alternatively the Distributed RAM option would haved use the LUT as RAM. You do not need to use an IP core generator or vendor specific to create RAM, but it is typically recommended. Use of Vendor-Specific Tool-Generated Cores and would be avoided for portable implemenations.

Here is teh seletion of Block Memory Generator:



Documention: https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v7_3/pg058-blk-mem-gen.pdf

On **page 3** of options, set **Write Depth to 4096** to allow 4096 memory locations

Your Final Instatiation Template should look something like this:

```
myBlockRAM your_instance_name (
.clka(clka), // input clka
.wea(wea), // input [0 : 0] wea
.addra(addra), // input [11 : 0] addra
.dina(dina), // input [15 : 0] dina
.douta(douta) // output [15 : 0] douta );
```

UART

- you can use code provided at https://eclipse.umbc.edu/robucci/cmpe415/attachments/uart.v
- Documention for UCF : Figure 7.3 https://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf#page=62
- If using windows, I strongly recommend use of RealTerm

Controller FSM

- You'll need to design a case-statement-based FSM to control the modules and make "top-level" system functional

What to be sure to turn in

- Submit the whole CLEANED ISE project folder(source files used to generate it) and instead of submitting only Verilog(.v) files (YOUR COMMENTING OF CODE WILL BE GRADED).
- Submit bit files in a separate directory
- Create and hand in multiple Verilog testbench modules that test your design
- Create a report that briefly explains your design and your testing. You must have one testbench for each module.
- Include the output of your Verilog testbench(s) in your report (THIS IS EXPLICITLY GRADED) with additional explanation about each testbench as needed to convince someone that each part of your design works and your simulation-based testing of each module is sufficient.

Provide Files:

- spartan 3e user guide https://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf
- uart_loopback.v
- uart_loopback_tb.v
- memory_hex_in.txt
- hw6.ucf

-