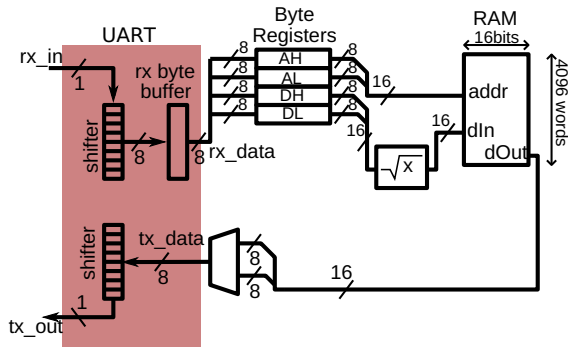


HW 6

Objective

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

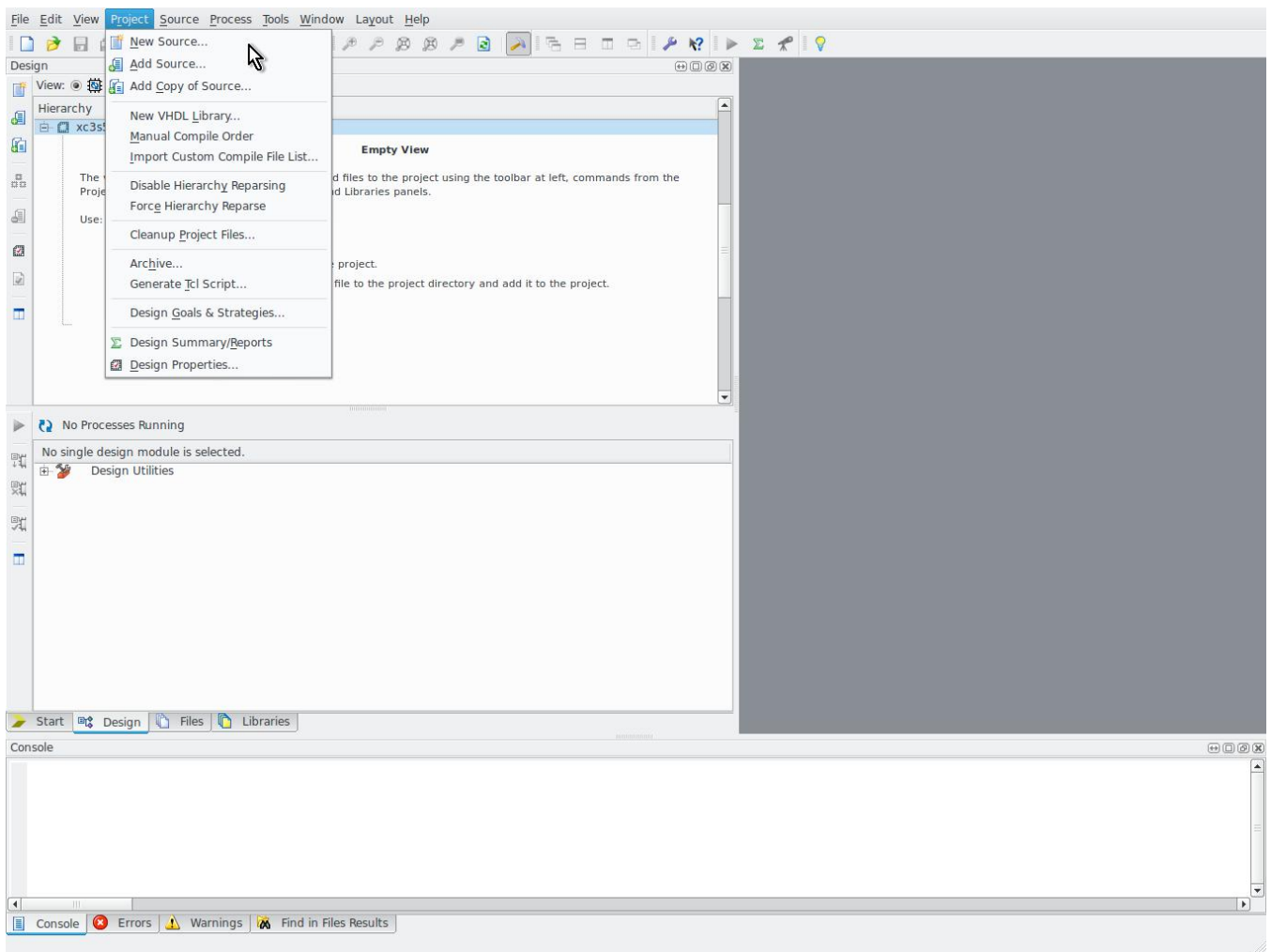
Requirements

- Your design should access 4 bytes from the serial port to initiate action
- Your implementation 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 baud 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 strongly recommended to consider modify the baud rate for the purpose of top-level simulation -- this should be controlled by a SINGLE parameter set in your test-bench. You should not modify your UART implementation files.
 - if required you can instantiate other "mini-versions" of hardware to speed simulation (such as a smaller ram, or smaller CORDIC processor) but your hardware implementation 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-cycle, pipelined square root CORDIC processor

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

Add a new source to the project:

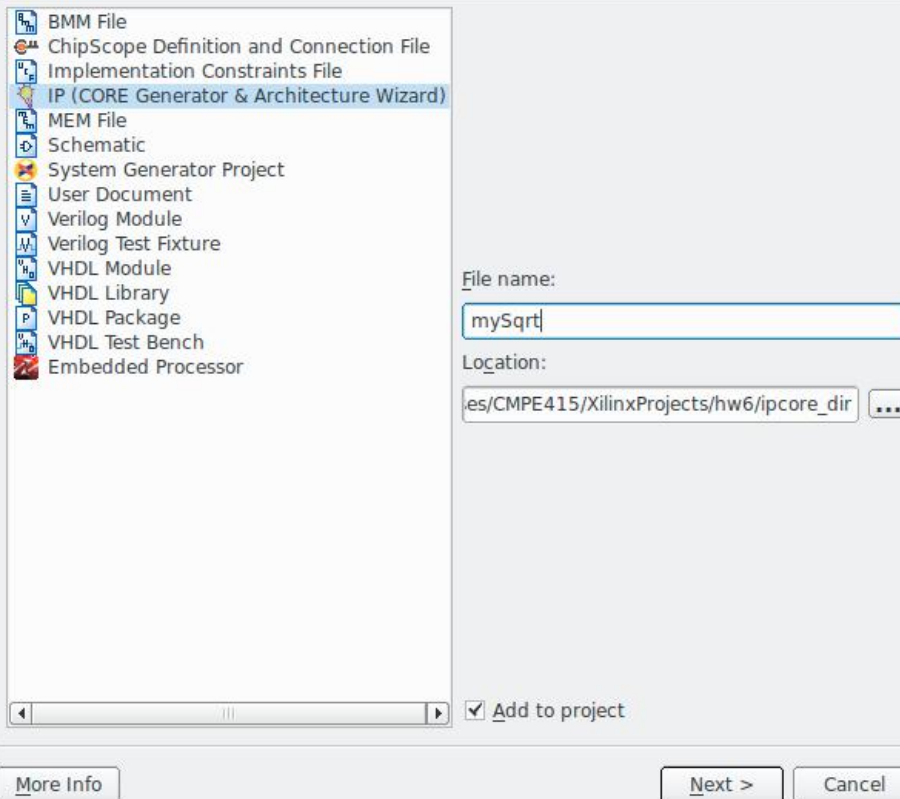


Step 1

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

Select Source Type

Select source type, file name and its location.



The dialog box shows a list of source types on the left. The 'IP (CORE Generator & Architecture Wizard)' option is selected and highlighted in blue. To the right, there are input fields for 'File name:' and 'Location:'. The 'File name' field contains 'mySqrt'. The 'Location' field contains 'es/CMPE415/XilinxProjects/hw6/ipcore_dir' followed by a browse button (...). At the bottom left, there is a checkbox labeled 'Add to project' which is checked. At the bottom right, there are three buttons: 'More Info', 'Next >', and 'Cancel'.

Source Type
BMM File
ChipScope Definition and Connection File
Implementation Constraints File
IP (CORE Generator & Architecture Wizard)
MEM File
Schematic
System Generator Project
User Document
Verilog Module
Verilog Test Fixture
VHDL Module
VHDL Library
VHDL Package
VHDL Test Bench
Embedded Processor

File name: mySqrt

Location: es/CMPE415/XilinxProjects/hw6/ipcore_dir ...

☒ Add to project

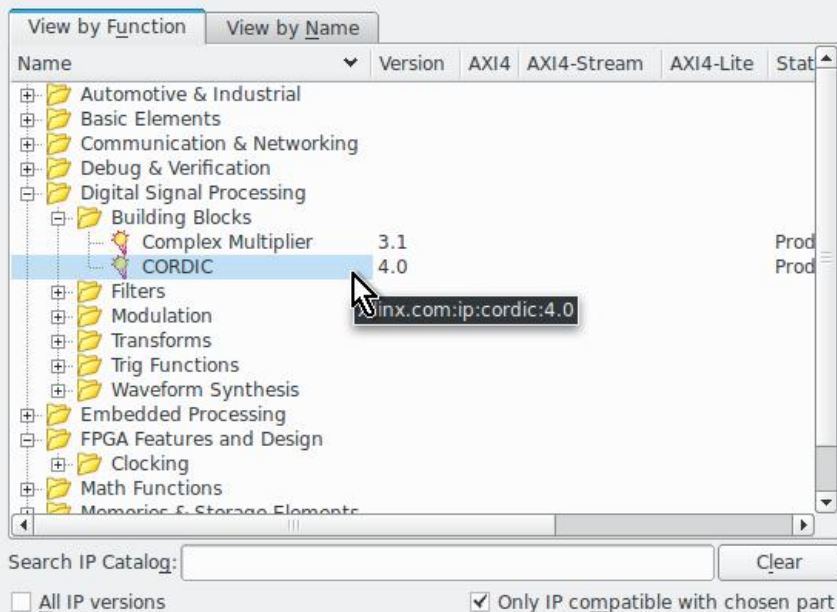
More Info Next > Cancel

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 chosen FPGA in the main project
- Select the CORDIC core

Select IP

Create Coregen or Architecture Wizard IP Core.



The dialog box shows a tree view of IP categories on the left. The 'CORDIC' core is selected under the 'Digital Signal Processing' category. A tooltip for the CORDIC core is visible, showing 'xilinx.com:ip:cordic:4.0'. At the bottom, there is a search bar labeled 'Search IP Catalog:' with a 'Clear' button. Below the search bar, there are two checkboxes: 'All IP versions' (unchecked) and 'Only IP compatible with chosen part' (checked). At the bottom right, there are three buttons: 'More Info', '< Back', and 'Next >'.

View by Function View by Name

Name	Version	AXI4	AXI4-Stream	AXI4-Lite	Status
Automotive & Industrial					
Basic Elements					
Communication & Networking					
Debug & Verification					
Digital Signal Processing					
Building Blocks					
Complex Multiplier	3.1				Prod
CORDIC	4.0				Prod
Filters					
Modulation					
Transforms					
Trig Functions					
Waveform Synthesis					
Embedded Processing					
FPGA Features and Design					
Clocking					
Math Functions					
Memories & Storage Elements					

Search IP Catalog: Clear

☐ All IP versions ☒ Only IP compatible with chosen part

More Info < Back Next > Cancel

Summary

Project Navigator will create a new skeleton source with the following specifications.

Add to Project: Yes

Source Directory:

/home/robucci/Nextcloud/covail/Courses/CMPE415/XilinxProjects/hw6/ipcore_dir

Source Type: IP (CORE Generator & Architecture Wizard)

Source Name: mySqrt2.xco

Core Type: CORDIC; Version: 4.0

[More Info](#)

< [Back](#)

[Finish](#)

[Cancel](#)

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

Documents View

IP Symbol

The diagram shows a purple rectangular block representing the CORDIC IP. On the left, there are three input buses: X_IN[15:0], Y_IN[15:0], and PHASE_IN[15:0]. On the right, there are three output buses: X_OUT[8:0], Y_OUT[8:0], and PHASE_OUT[8:0]. Below these, there are two control inputs: ND and CE, and two control outputs: RFD and RDY. A clock input CLK is at the bottom left.

CORDIC

xilinx.com:ip:cordic:4.0

Component Name

Functional Selection

☐ Rotate
☐ Translate
☐ Sin and Cos
☐ Sinh and Cosh
☐ Arc Tan
☐ Arc Tanh
☒ Square Root

Architectural Configuration

☐ Word Serial
☒ Parallel

Pipelining Mode

☐ No Pipelining
☐ Optimal
☒ Maximum

Page 1 of 3

IP Symbol

Implementation Details

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

Documents View

IP Symbol

The diagram shows a purple rectangular block representing the CORDIC IP. On the left, there are three input buses: X_IN[15:0], Y_IN[15:0], and PHASE_IN[15:0]. On the right, there are three output buses: X_OUT[8:0], Y_OUT[8:0], and PHASE_OUT[8:0]. Below these, there are two control inputs: ND and CE, and two control outputs: RFD and RDY. A clock input CLK is at the bottom left.

CORDIC

xilinx.com:ip:cordic:4.0

Data Format

☐ Signed Fraction
☐ Unsigned Fraction
☒ Unsigned Integer

Phase Format

☒ Radians
☐ Scaled Radians

Input/Output Options

Input Width Range: 8..48

☐ Register Inputs

Output Width Range: 5..48

☐ Register Outputs

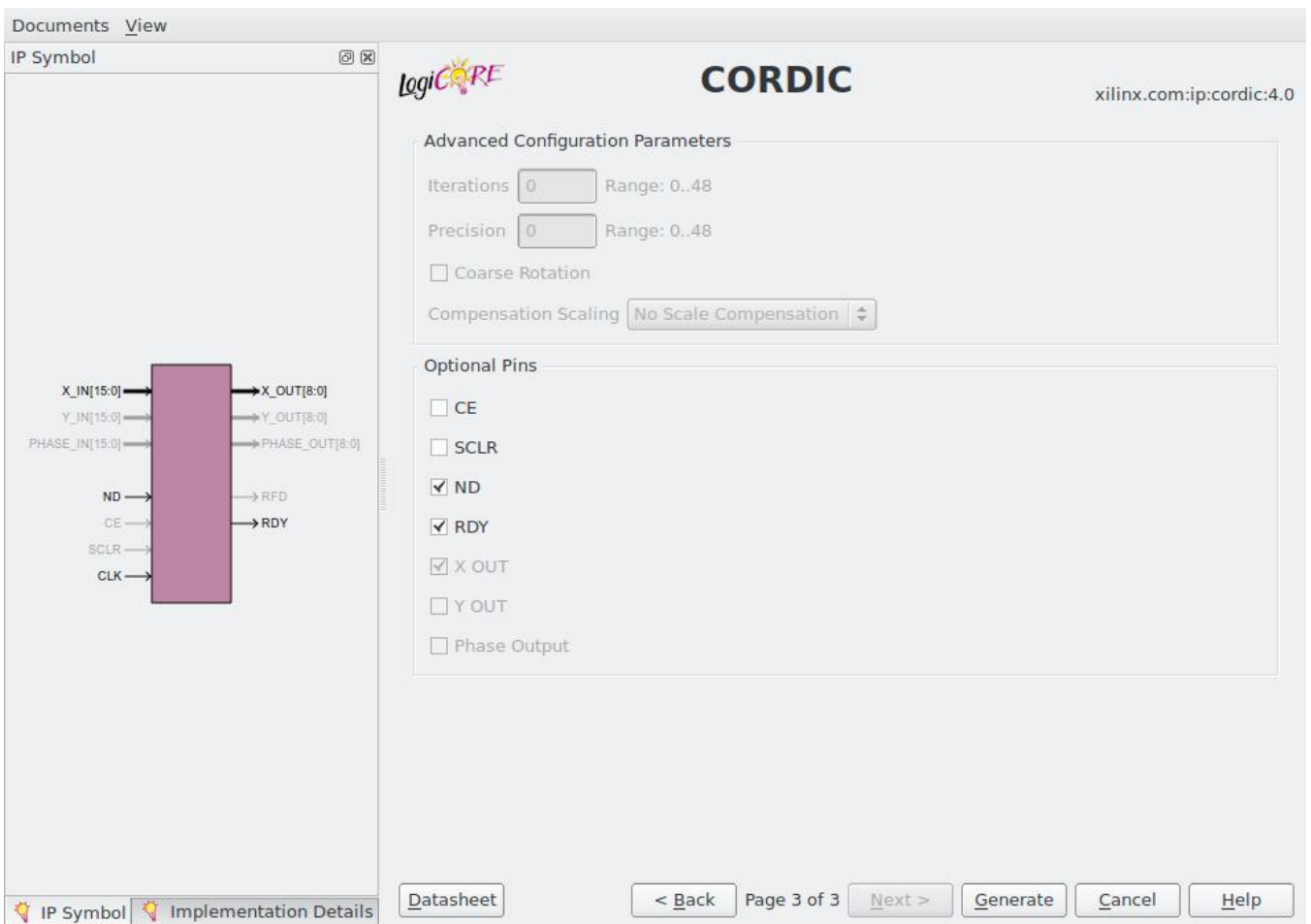
Round Mode

☒ Truncate
☐ Round Pos Inf
☐ Round Pos Neg Inf
☐ Nearest Even

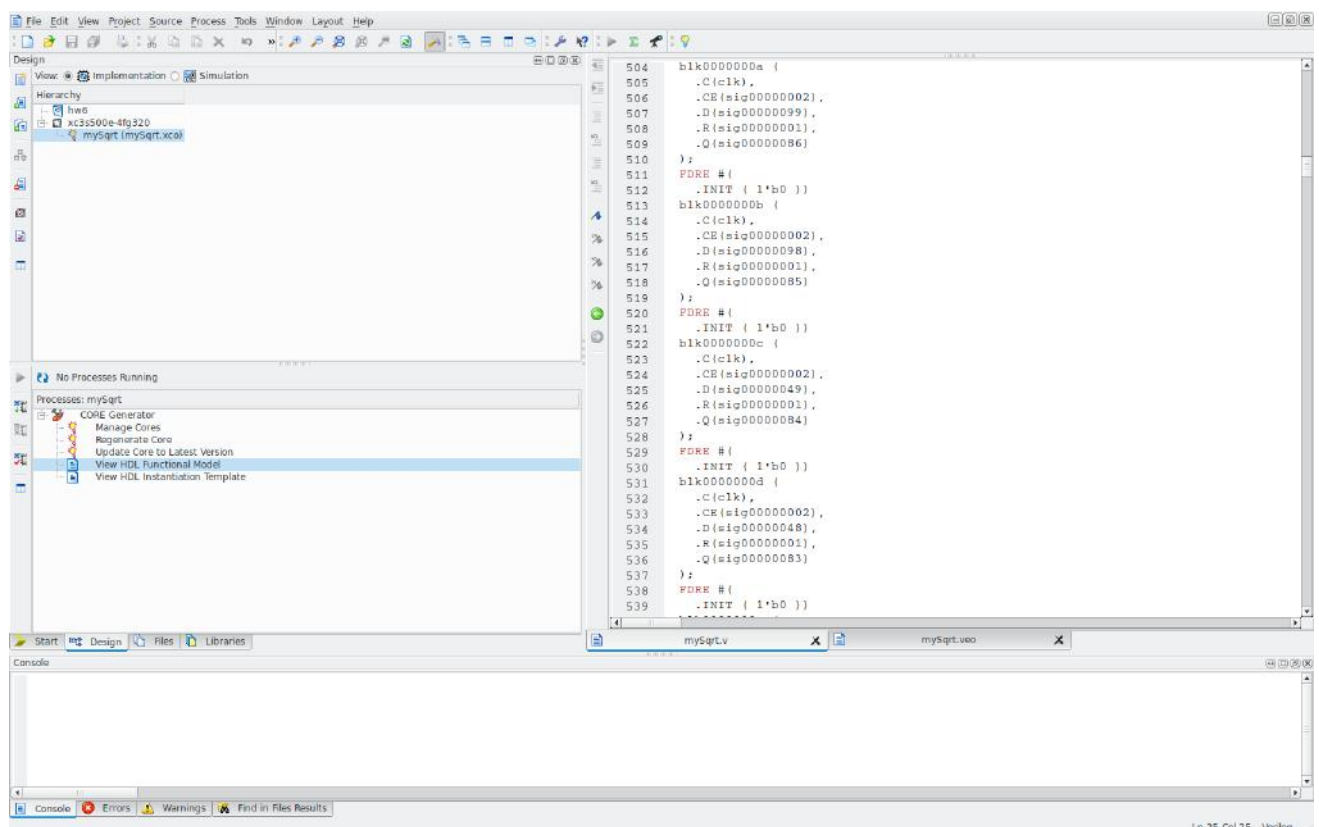
Page 2 of 3

IP Symbol

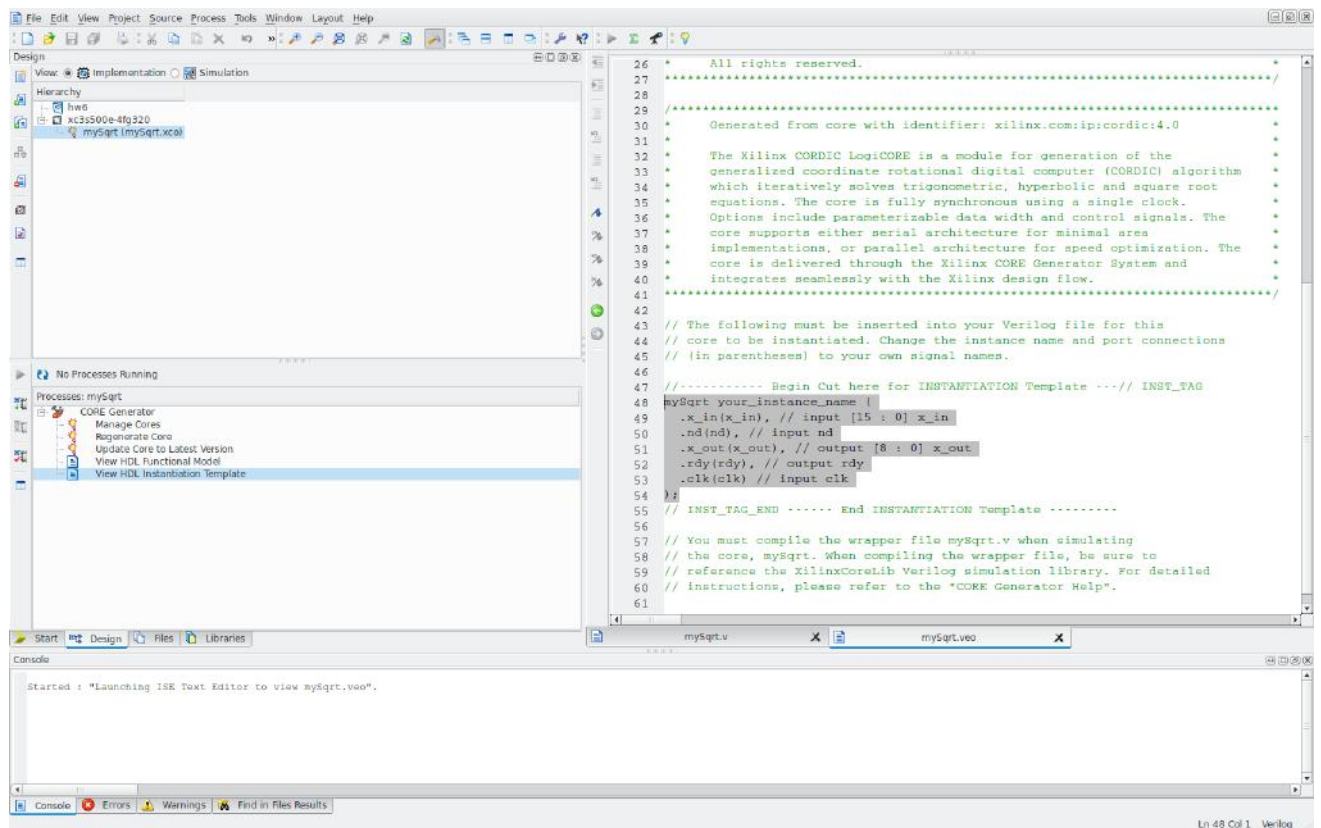
Implementation Details



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



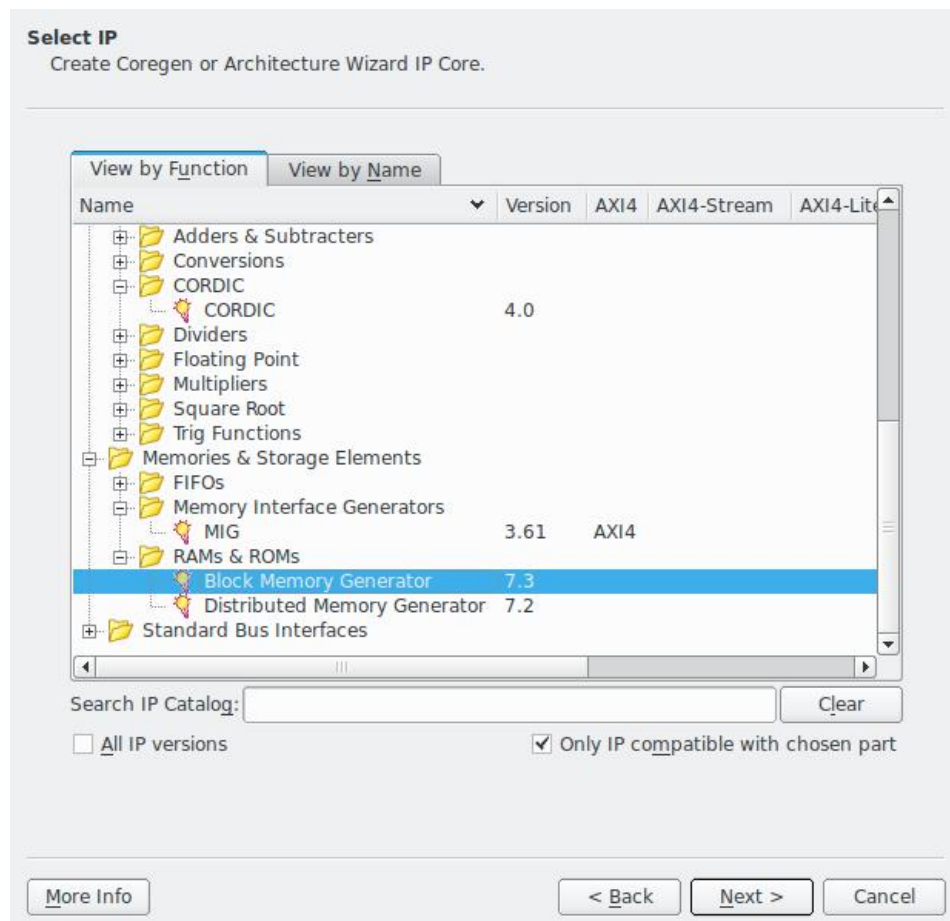
- You can view the template that shows how to instantiate the generated module in your design:



Generating a RAM

We'll generate a ram using the IP core generator. We'll choose to use block RAM (uses specialized blocks of memory hardware on the FPGA). Alternatively the Distributed RAM option would have used 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 implementations.

Here is the selection of Block Memory Generator:



Documentation: 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>
- Documentation 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](#)

-