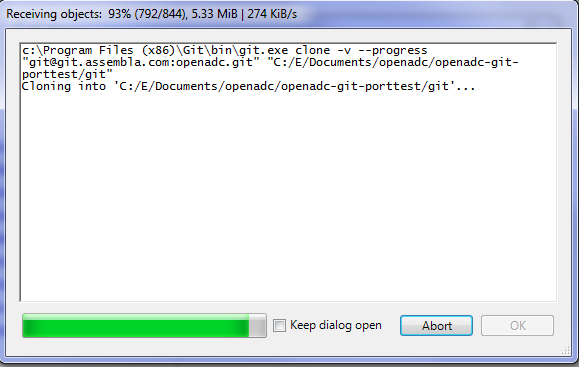
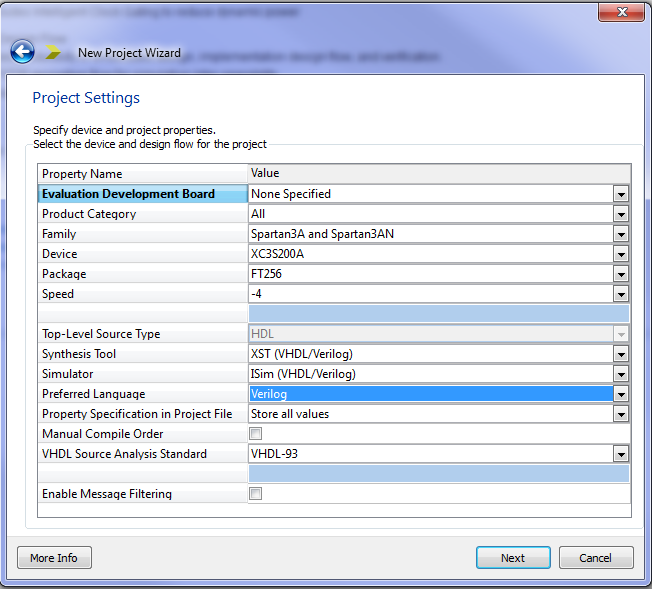
Porting Guide for OpenADC

# Getting Started

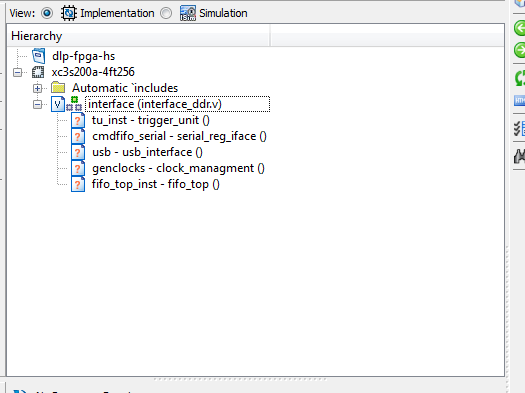
To begin with, you’ll need to get the code. For now just pull from GIT:



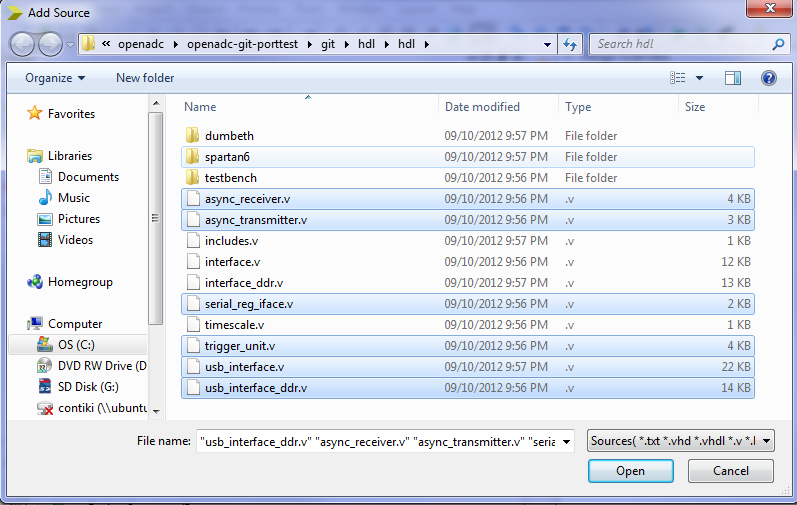
Next, we will create a project in Xilinx ISE Project Navigator.

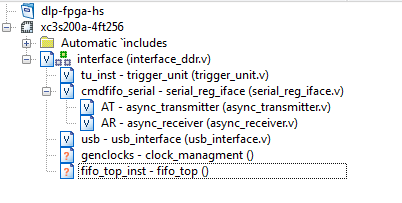


Add the top file: interface\_ddr.v (even if you don’t use DDR!!). It should now look like this:

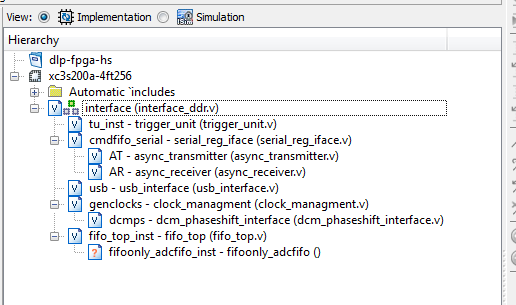


Add some more sources. If you add sources you don’t need just delete them from the project:

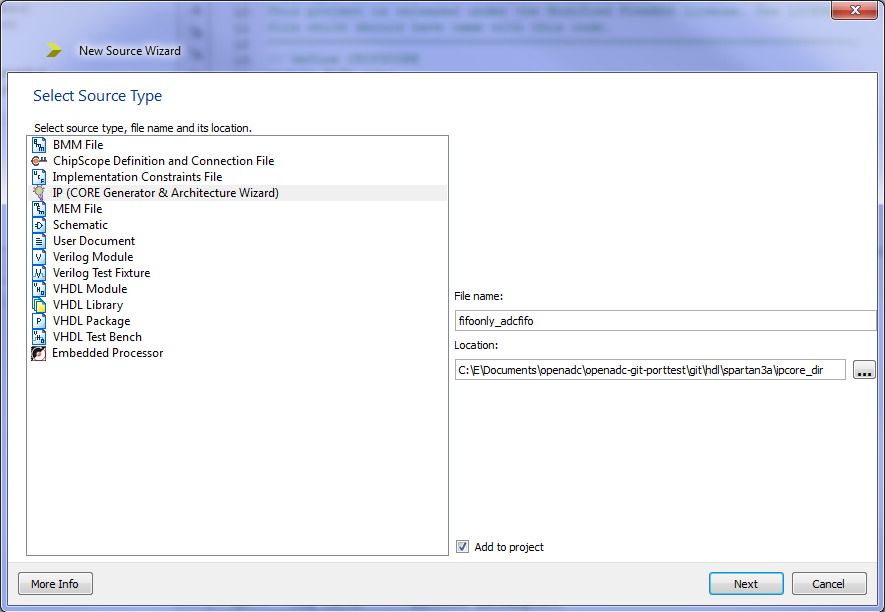


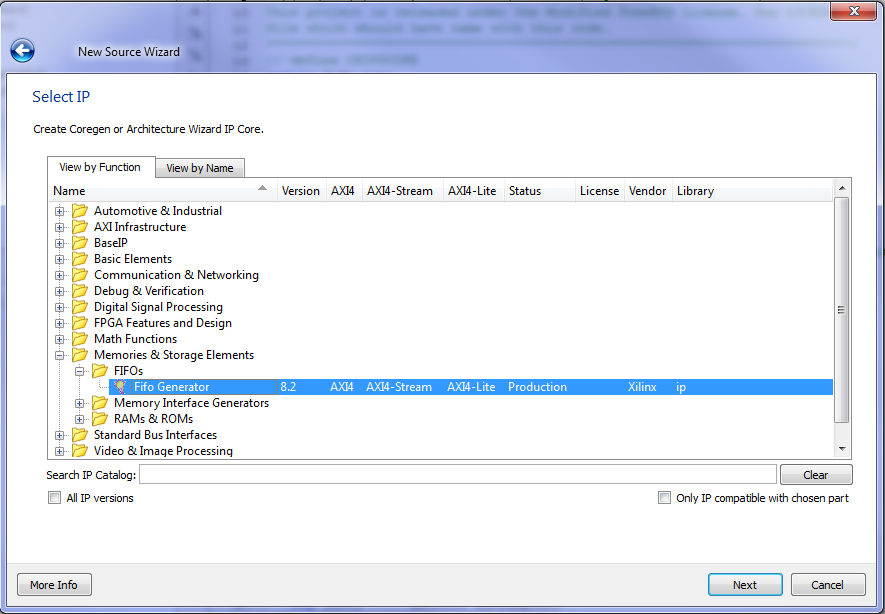


There are a bunch of source files in the ‘Spartan6’ subdirectory. These files may need to be rewritten for different targets. For now just add them as-is if to see how far off they are. Everything should now look like this:

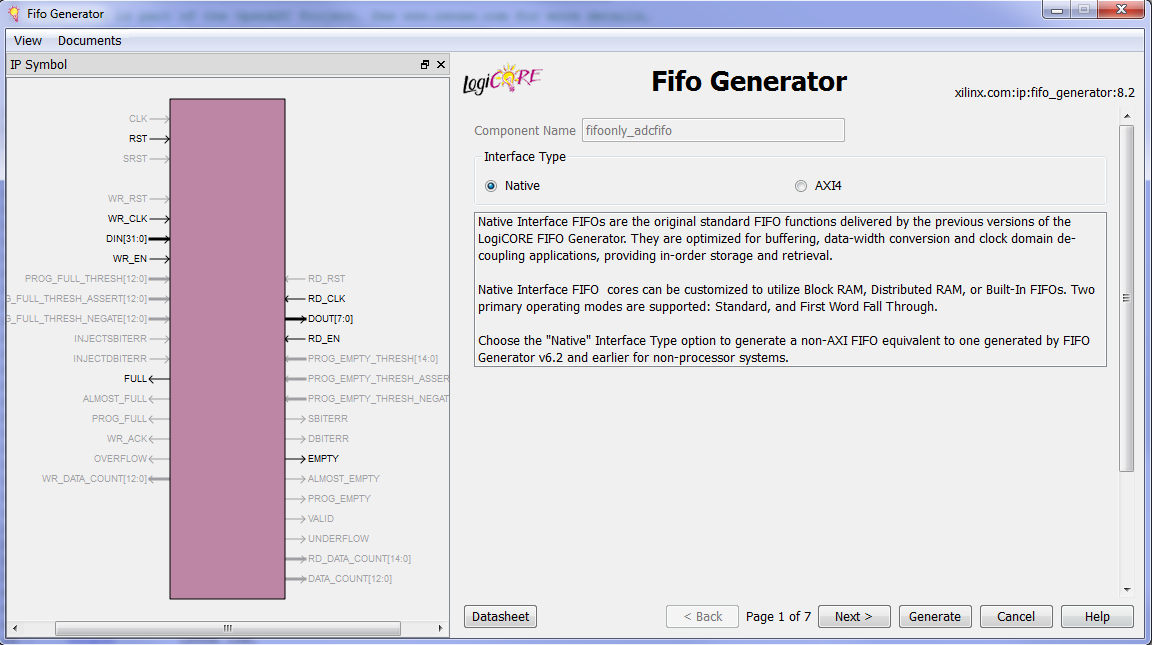


The final source, the fifoonly\_adcfifo you need to generate with ISE CoreGen. The following screen-shots show this setup:

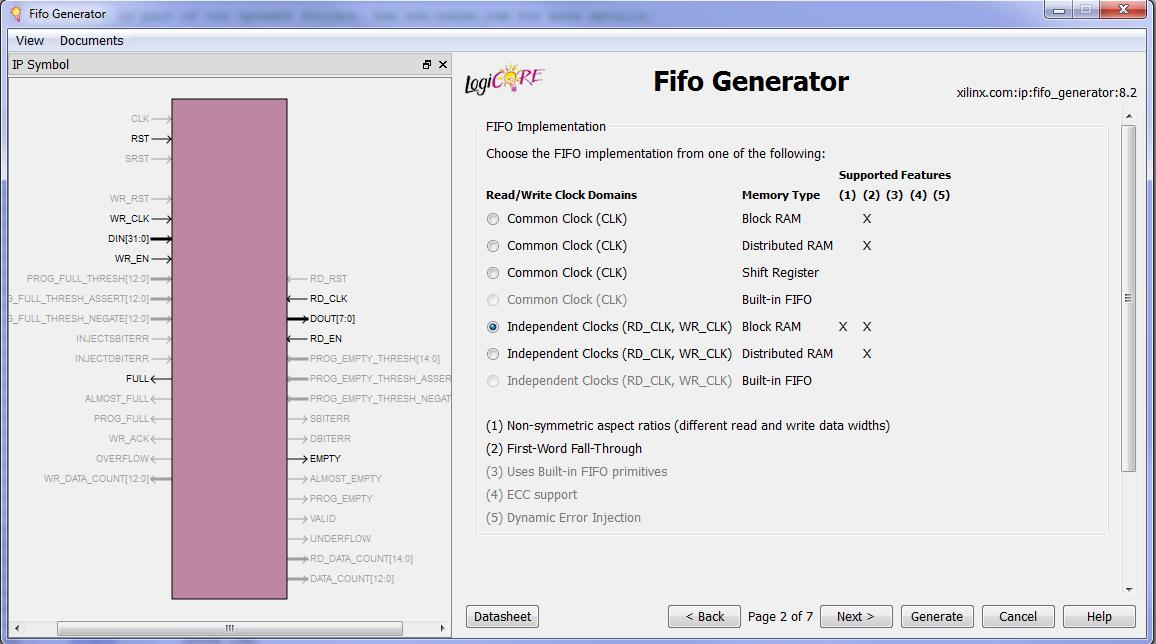




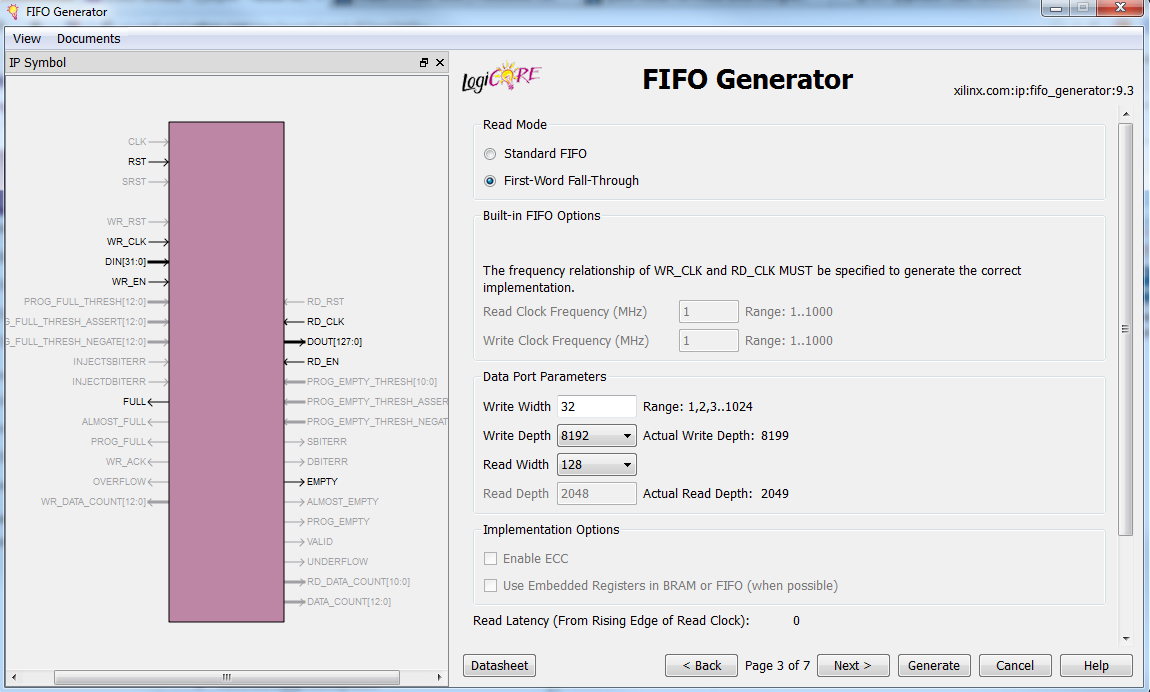
First page: Select ‘Native’ Interface Type:



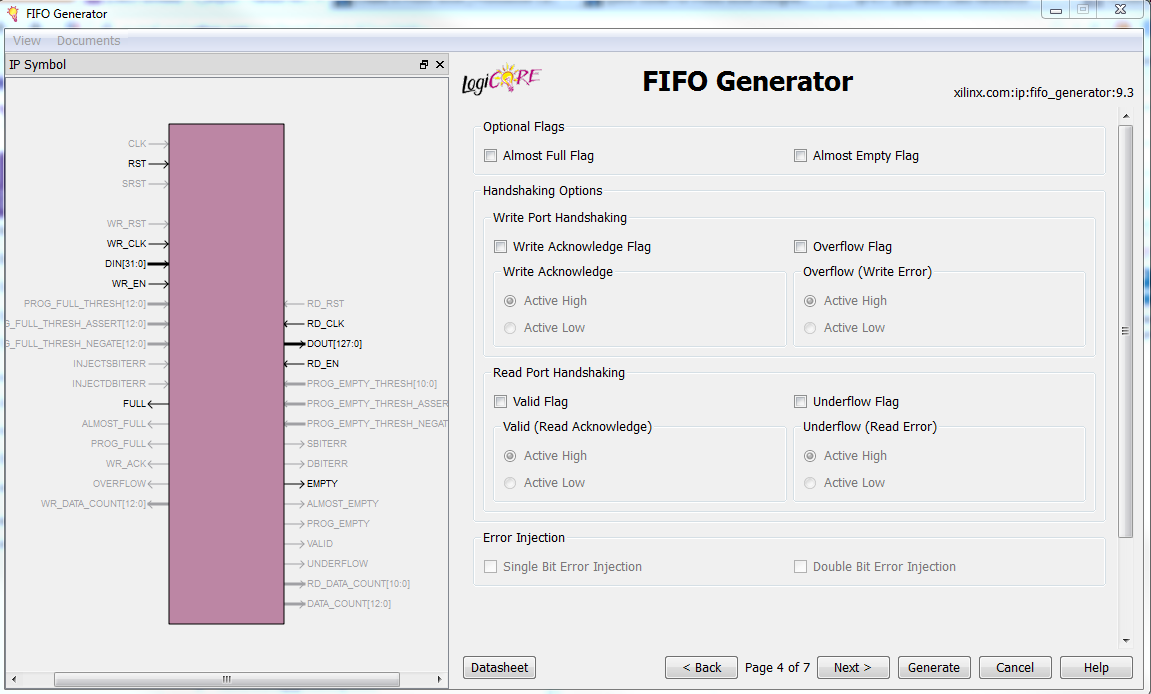
Second page: Select ‘**Independent Clocks, Block RAM**’:



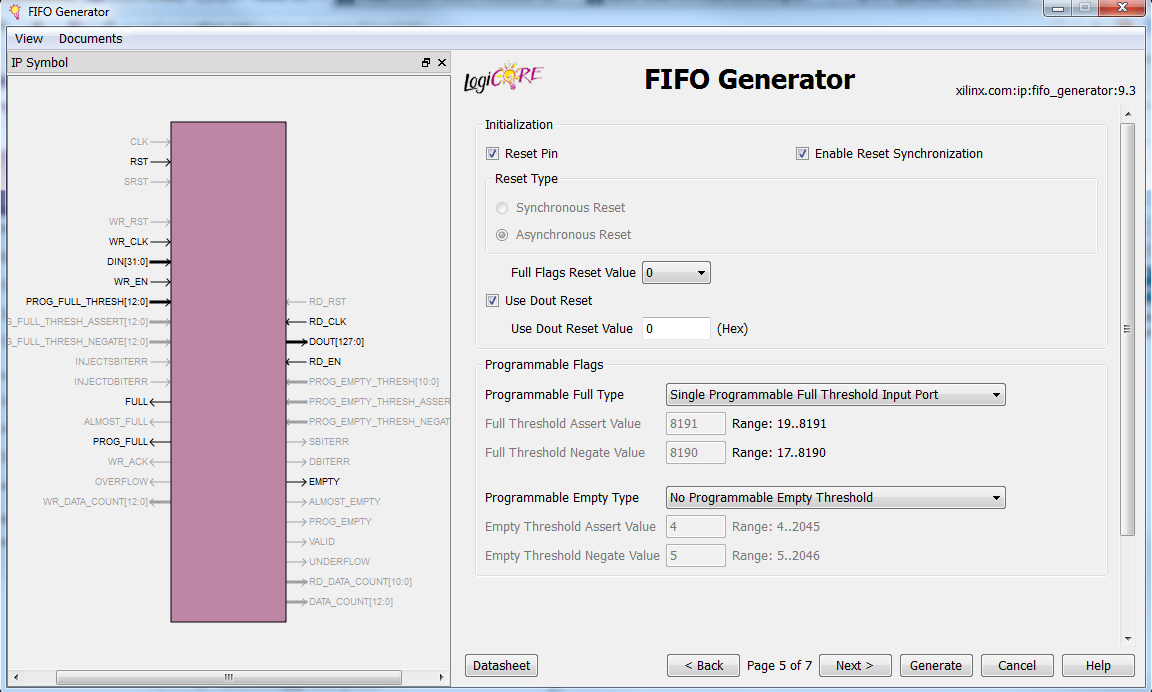
3rd page: Select ‘**First Word Fall-Through**’, ‘**Write Width=32**’, ‘**Read Width=128**’. Select a write depth – exact value depends on your HW, here I’ve used 8192, on a smaller FPGA select a smaller number. Total samples = this depth x 3.



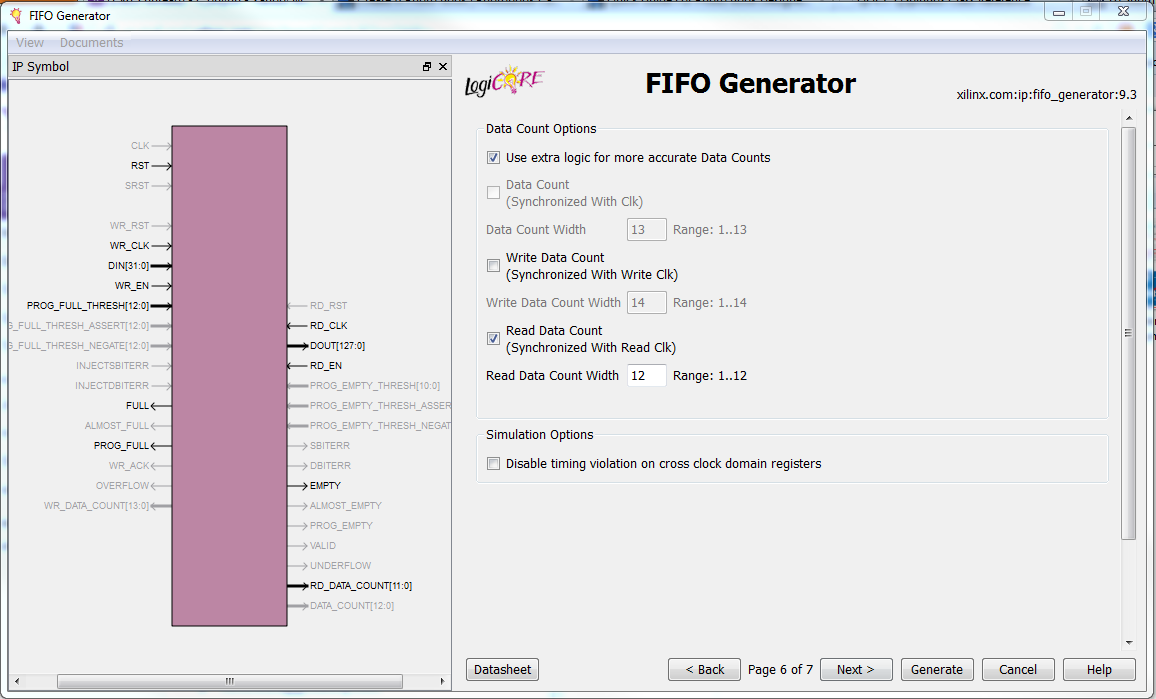
Fourth Page: Everything unchecked (defaults)



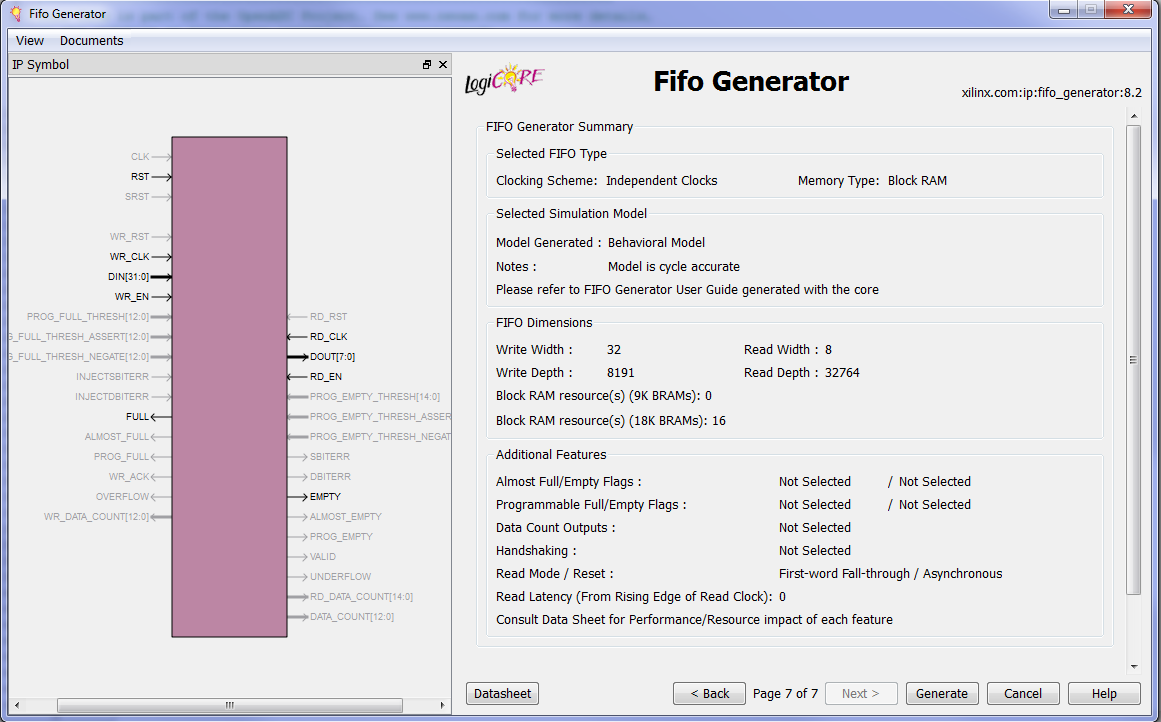
Fifth Page: Reset pin, reset sync, reset Dout to 0. These should all be defaults. Single programmable full threshold input port.



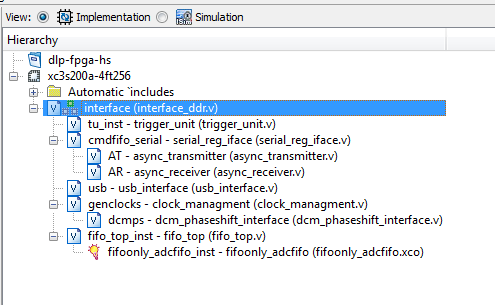
Sixth page: “Read Data Count” and “Use extra logic for more accurate Data Counts”. Width will depend on your exact FIFO depth.



Seventh page: Summary. Can see number of block rams being used here & see how your design is fitting.

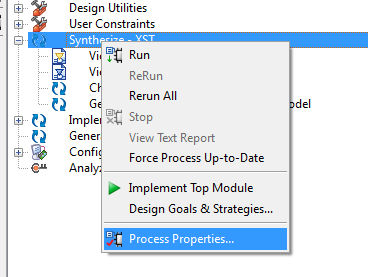


Once you generate this, your file should now look like this:

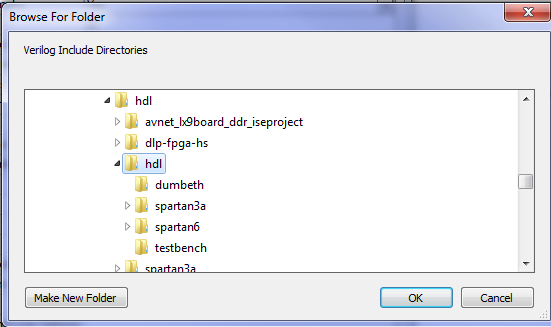


# Compile Settings

Click on Synthesis Properties:



And change the ‘Verilog Include Directories’ to point to the hdl folder (where include.v is):



# Add Setup.v to your Project

You’ll need to create a file called ‘setup.v’ which defines various clock frequencies along with hardware defines. See the LX9 example for a file & edit it.

# See What Happens

Alright… run the synthesis process & see what breaks. I don’t have much advice here – then run implement after that.

Fix all the errors & check through warnings, although it’s still pretty dirty so generates a lot of warnings anyway! You may want to compare with the original project.

If you see warnings about things like ‘ADC\_Data’ being optimized away, you probably made some mistake with the clock setup. If the synthesizer realizes the clock isn’t connected it will optimize away everything basically.

# Setup Hardware Pins, etc

Finally, setup your hardware. Be sure to adjust:

* Clock frequency defines in includes.v (used for UART baud rate calculation)
* UCF File (create it!)
* Possibly DCM settings

# Test It

Power up the board. You should see:

* One LED on solid. This LED is ON (output high) when the reset input is inactive (low). If you hit the reset button see if the LED toggles. If the LED is off & then turns on when you hit reset, you may have reset logic backwards, just invert it in the design file
* One LED is blinking. If this is not the case the clock isn’t running.

Use adc capture example. Hit ‘connect’ with serial port setup, and see what happens. If doesn’t work, things to check:

* Are frequencies defined correctly?
* What baud are you trying? The OpenADC should be run at the highest possible baud – but check what your serial port (or serial-USB) supports. Try something like 115200 defined in includes.v to start, as some converters will silently fail if you set too high a baud rate.
* Internally loopback the txd to rxd (‘assign txd = rxd;’ & comment out the connection of txd to the registers block). Using a serial emulator send data & confirm it loops back. If not you have a hardware issue.

Congratulations! It connects! Now check the following functionality:

* Input a sine wave (~1-2 MHz). Make sure it is smooth – if you see spikes you may need to adjust the clock delay.
* Vary the gain, check the voltage at the op-amp pin varies between 0-1V as you adjust this. Measure this at C39/R16.
* Switching between high/low should toggle the voltage on the gainmode op-amp pin, or just see if the gain seems to be switching by looking at the results on-screen
* Connect a logic-level clock input (careful, don’t blow your FPGA up as there is NO BUFFERING). Confirm you can measure the frequency of this clock.
* Switch trigger mode, confirm when you hit ‘capture’ it freezes until you pull trigger high (hint: you can often just touch it with your hand & noise will pull it high)