ELEC 522, Advanced VLSI
Rice University, Fall 2022
27 October 2022
Due: Tuesday 8 November 2022 at 11:59pm

Project 4: Using VitisHLS to implement a CORDIC module on Zynq System

Goals:
This fall we have looked mostly at the Xilinx System Generator design flow. In this project, we will use again the Xilinx VitisHLS design flow for C to HDL generation. We have discussed CORDIC in the class and several papers are on the Canvas class web site. In this project we will investigate the flow from C or C++ code to FPGA and control from the ARM core in the Zynq SoC. We also plan to use this CORDIC module in the upcoming 5th project on the QR Decomposition.

Project procedures:

1. Use VitisHLS to design a 16 bit signed fixed-point CORDIC module, that is 16 bit signed inputs and outputs. Write C or C++ code for the CORDIC module for the "circular" mode only, that is for sine, cosine, and inverse tangent functions. Some sample C code for CORDIC is in the lecture slides and also on Canvas in the CAD_Tool_Examples folder for Project 4.
2. The CORDIC architecture can be for the serial scheme which is more area efficient, but slower than the pipelined scheme used in the Xilinx Core. You can also use the unrolled architecture used by Xilinx with fixed shifts if the "barrel shifter" proves too difficult to control. (As mentioned above, we have posted some generic C code about CORDIC that may be of use as an example of a floating-point simulation.)
3. There are typically two cores, one for sine/cosine and one for inverse tangent modes in the Sysgen example from class. However, a single core with an input control signal to select modes is the implementation that we want to have more generality to use in the QRD array in Project 5. Please design this single parameterized core.
4. The input data for the sine, cosine mode should be more general so that either the sine and cosines are produced, or a more general rotation with scale factor correction is performed. That means that you should not "hardwire" 0 or 1 or scale factor constants as inputs as in the example Model Composer CORDIC blocks but make these more general variables. This rotation mode with arbitrary inputs with scale factor correction may simplify the QR Decomposition architecture in the next project 5.
5. Generate HDL code with the appropriate architecture constraints. Verify your C or C++ code and HDL code to make sure that they function correctly. Check the scheduling result. Adjust architecture constraints parameters to optimize your architecture (you might need to slightly modify your C or C++ code to adapt it to

your architecture optimization). Describe whether you are using an iterative or a pipelined approach as your goal and the tradeoffs.

6. Repeat the above step until you achieve a good balance between system throughput and hardware resource costs, by changing different combination of optimization methods. Make your choice and generate HDL code for your final system.
7. **Model Composer Export for Simulink Co-Sim:** Export from VitisHLS for Model Composer. The SysGen export will still rely on the "Gateways" used in SysGen for Simulink control. Import your HDL code into a Model Composer model using the VitisHLS block in SysGen. Remember to use a different model or file name between your VitisHLS project and your SysGen hosting model file. Use the VitisHLS export for Sysgen option. Use a testbench to test your system. Synthesize the result for our Zynq chip and report on the FPGA utilization. Also, Generate and verify with Hardware Co-Sim in Model Composer on the ZedBoard. Please complete this step before continuing to the next step to keep your projects separate.
8. **IP Block Export for VITIS and Zynq ARM program control:** You should copy your VitisHLS source code to a new folder and create a new project to keep this part separate of the SysGen export. You will need to modify your VitisHLS C or C++ code to make the I/O ports now AXI Lite interfaces as shown in one of the tutorial examples. Export your design again from VitisHLS but this time as an IP block for IP catalog. **Again, you may want to make a copy of your VitisHLS project in a separate folder to avoid any conflicts between exporting for SysGen and exporting for IP creator.** This may produce some different HDL files than for SysGen. Import your Verilog HDL code generated by VitisHLS into a Vivado project as done in the class demo of the "treeadd." We have posted additional notes/screen capture of the flow and completed some tutorials in class. After completing the wiring of the CORDIC module to the Zynq Processing System and adding the AXI interface, you will build the FPGA bit file in Vivado. You will need to prepare a C or C++ code file to run on the Zynq ARM core for testing your CORDIC IP block. This can be a modified version of your testbench for the VitisHLS simulation. However, you will need to use the memory address for the AXI Lite ports. (Follow the example for "treeadd" and change the names of the labels (#defines) to match your variable names. Export the hardware to the Xilinx Vitis software development kit and start an application project. As in the tutorials, read in the XSA file from Vivado and then create a blank C++ application, and then add in your C++ code. Build and then download and run on the Zedboard. Verify that you can compute proper sines, cosines, and inverse tangents. **Save screenshot of terminal output.**

Requirements:

1. The system should handle arbitrary values of X and Y inputs, that is, the complete unit circle. You should assume simple data types, such as 16 bit two’s complement signed fixed-point numbers. You may want to follow the formats in the Xilinx CORE, LogiCORE IP CORDIC v6.0 or current.

2. Xilinx uses a format described as 1QN where N = word width - 2. It can also be described as Fix(N+2)_N using the System Generator Fix format. A format of 2QN can also be used, and may be easier to interface with. So, if you use 16 bit numbers there will be a sign bit, then two integer bits, and then 13 fractional bits. Although this is confusing, you can follow the fixed type definition in the "treeadd" fixed example where we set up as "ap_fixed<16,3>" in the Vitis HLS code and as "typedef ap_fixed<16,3, AP_RND, AP_SAT> FIXED_TYPE" in the ARM C++ code. You will then need to handle the appropriate number of iterations, stages, and scale factor correction values. Remember that, there is no standard answer for this project.
3. Try to maximize the throughput as much as possible as you can. The methods include but are not limited to maximizing the clock frequency, optimizing loop performance.

Documentation:
In your documentation, please describe how you change your input C code and the architecture constraints to optimize your design. Describe your final choice of the constraint configurations. Explain the advantages of your final design. Explain the timing scheduling results. If you think you have done something special/smart that makes verifying your C code and HDL code faster and easier, please also describe how you do it in your report. Give the timing and resource information from the synthesis report for FPGA.

Also, in your project submission, include your VitisHLS code and project info, your Model composer models where the VitisHLS block was used, your comparison built-in SysGen CORDIC model, the various FPGA usage result reports, and your Simulink model simulation plots showing accuracy.

For the integration with the Zynq ARM core in Vivado and VITIS, include your C or C++ testbench code and terminal results from running on the Zedboard under ARM core control. Also include a screenshot of the Vivado integrated block diagram showing the Zynq processing system, AXI interface, and your created CORDIC IP block.