ELEC 522, Advanced VLSI
Rice University, Fall 2022
22 November 2022
Due: Tuesday 13 December 2022
Design Project 6

4 x 4 Linear System Solver with ARM Core and CORDIC QRD and Matrix-Multiplication Accelerator Integration

Presentation in Lab: ***Monday 5 December at 3 pm or TBD***
Report & Project Files Due: ***Tuesday 13 December at 11:59pm***

Goal: Explore the integration of a programmable processing core with a custom accelerator in the context of Vivado through Xilinx Vitis and Xilinx Model Composer. Assess the performance speed-up between a pure C++ code version of the algorithm and C++ code with links to the custom accelerator.

Please refer to the notes on Vivado / IP Integrator / Vitis for system integration of the ARM core and FPGA module from Model Composer in the ZedBoard on Canvas.

Topic: Solve a 4x4 system of linear equations using hardware acceleration, essentially Ax = b where A is a 4x4 matrix, b is a 4 element vector, and x is the 4 element vector that we are solving for. There will be a QR Decomposition phase, an application of Q transpose to b, and a final back-substitution phase to find x. Integrate the QR Decomposition array from the last Project 5 with the ARM core on the Zynq ZedBoard to understand SoC programming and integration issues. Do the matrix-vector multiplication on the ARM core in C++ code or for extra credit integrate a modified version of your matrix multiplication array from the earlier projects (Project 2 or 3) to do the matrix-vector multiplication. Finally, do the back-substitution phase in C++ code only on the ARM core. Benchmark the area and time performance of this SoC solution using the C++ language timer calls on the ARM core.

You may use the same ap_fixed data format with 16 bit signed data with 3 integer bits and 13 fraction bits. A sample data problem is posted on Canvas in CAD_Tool_Examples>Project_6_Linear_Solver_Notes>QR_fraction_example_unrolled_matlab.txt. Please verify the correct behavior with this data set and then create two additional data sets for further verification. Compare your results with a Matlab floating point solution of the same data sets. Because of the 16 bit data format there will be some accuracy loss. Please note and discuss. Also you may need to change your data format to have a larger integer part and smaller fractional part depending on your input data.

In addition to the FPGA accelerated version, create a totally stand-alone C++ code version of the entire linear equation solver algorithm (with QR Decomposition) to measure timing when using only the ARM core and no accelerators. Use the USB serial port (UART) and “cout” statements on the monitor via the Vitis Terminal tool to display results and also for debugging.

Finally do a timing analysis of the design tradeoffs between the totally stand-alone C++ version and the FPGA SoC accelerated version. You should demonstrate that your array is able to solve three different instances of systems of equations. This helps us to verify that all initialization is done properly, that data is unloaded properly, and that additional problems will solve correctly on your arrays. This also helps to understand the loading and unloading times needed. Please document your timings for each system of equations, and then the total time for all three. Also, please do this for both the stand-alone C++ versions and also the FPGA SoC accelerated version.

Note: Use a "bare metal" approach where your C++ code runs as the only task on the ARM core. You may load in your input data by saving to a ".h" header file. This may simplify testing since file I/O and "scanf" are not easily supported in "bare metal." (We did do a demo of the use of scanf part for data input from the terminal for one of the treeadd examples.) Some sample code (for the ARM C++ language timer) is in resources Files/CAD_Tool_Examples from one of our recent demos, in particular the Project_4_tree_fixed_HLS_Vitis. Also, Project_6_QRD_Notes has files to help with C code for the QRD for running the ARM only version.

Note: The intent of Project 6 is to use the Model Composer QRD from Project 5 and the Matrix-Vector multiplication mode from Project 2 or 3. Some students may prefer to do the entire QRD and the Matrix-Vector in Vitis HLS functions using the Project 4 CORDIC in Vitis HLS. If you do choose to use Vitis HLS, please explain and justify and how you attempt to capture the systolic data movement. It will be an acceptable alternative.

Project Presentation: Please prepare a short PowerPoint presentation on your design progress for ***Monday 5 December 2022 at 3pm*** *based on class approval. Other times can be arranged by appointment if necessary.* Stress the functional description of the project and status on use of the design tools. You should give results on system resources in terms of adders/multipliers and FPGA slice counts and timing speedup / slowdown.

Project Report: As mentioned above, please implement with Vivado / IP Integrator / Vitis and simulate and verify in the lab with hardware on the ZedBoard. Please include your block diagrams and scope / terminal text outputs in the report. Also, please prepare a "tar" or "zip" file of the project presentation file, the report, C++ code, Vivado HLS, and Model Composer model files and upload to Canvas by ***Tuesday 13 December 2022***. That is the last day that all work is due at Rice, so no extension beyond that time.

Extra credit options (10 points each): (1) Use and integrate a modified version of your matrix multiplication array from the earlier projects (Project 2 or 3) to do the matrix-vector multiplication on the PL FPGA fabric instead of doing matrix-vector multiplication on the ARM core for the accelerated version. (2) Pipeline the array so that it can begin to work on the next QR Decomposition while finishing the current QRD to handle multiple sets of linear equations more quickly than a totally sequential approach. (3) Generalize your design to assume that the matrix values are ***complex*** numbers and not real numbers. (4) Adapt and parallelize your all ARM core C++ code for the linear system solver to run on a GPU under CUDA and explore timings for 4x4, and 64x64 systems. Use timing method as in Project 1, our 4x4 data set, and your choice of a random data set for the 64x 64 case.