22T2 COMP4601 Week 3 Lab/Week 3 Hand-in exercises 13/06/2022
(Notes and questions on PP4FPGAs – Chapter 3 exercises)

During Week 3, you are expected to complete Chapter 5 of the 2020.1 Vivado HLS Tutorial and to carry out selected exercises from Ch. 3 of the text. Your answers to indicated exercises should be submitted electronically as a PDF using WebCMS by **5pm Monday 20 June**. This hand-in is worth 10% of your mark in the course.

Completing the exercises below should take 2 – 3 hours after you have completed Chs 2 – 5 of the Vivado HLS Tutorial. You will first implement the code of Figure 3.3 with various data types. Then you will implement the code of Figure 3.6 and consider further optimizations of the main loop.

Exercises

1. In the spirit of the exercises listed on page 75 of the text, implement the code of Figure 3.3 using floating point data types. You are to compare the performance and resource usage of this baseline with implementations using fixed data types.

   The code from Figure 3.3 is contained in the design file `cordic.cpp`. This file uses `cordic.h` and has a simulation file, `cordic-top.cpp`, associated with it. These files have been included in the `ex1` subdirectory of the `lab3.zip` file for your convenience. Additional files provided in `lab3/ex1` include `cordic-float.tcl` to create the project `cordic_float_prj`, and the header files `cordic-float.h` and `cordic-fixed.h` to replace `cordic.h`, as explained below.

   a. Copy `cordic-float.h` to `cordic.h`
   b. Run `vivado_hls -f cordic-float.tcl` in the Vivado HLS command prompt window.
   c. Run `vivado_hls -p cordic_float_prj` from the Vivado HLS command prompt
   d. Check the source code as well as the header file to make sure you are running the code from Figure 3.3 with floating point data types
   e. Simulate and synthesize the baseline, *solution1*
   f. Create a new solution, *solution2*
   g. Copy `cordic-fixed.h` to `cordic.h` in Windows.
   h. Simulate and synthesize the design using `ap_fixed<12,2>` data types instead of floating point
   i. Create a new solution, *solution3*
   j. Modify the data type of `THETA_TYPE` and `COS_SIN_TYPE` in `cordic.h` to `ap_fixed<16,2>` in the source pane of the Vivado HLS GUI
   k. Simulate and synthesize the design
   l. Compare the performance (estimated clock period and latency), resource utilization and accuracy of the three implementations. Explain the differences in the execution schedules across the three solutions. Report your observations and thoughts in your hand-in for Week 3.
   m. Close Vivado HLS

2. Next, use the files contained in `lab3/ex2` to compare the implementation of the code from Figure 3.3 with the code from Figure 3.6 using the `ap_fixed<12,2>` data type.
    a. Copy `cordic-fixed.h` to `cordic.h`
    b. Run `vivado_hls -f cordic-optimized.tcl` in the Vivado HLS command prompt
    c. Run `vivado_hls -p cordic_optimized_prj` in the Vivado HLS command prompt
    d. Check that the source code and header file correspond to Figure 3.6 with `ap_fixed<12,2>` type variables
    e. Simulate and synthesize the design
    f. In your hand-in for Week 3, report on the similarities and differences in performance, latency, area usage and accuracy between the implementations of *cordic_optimized_prj/solution1* from 2.e with that of *cordic_float_prj/solution2* from 1.h. Explain your findings.

3. Examine the execution schedule in the Analysis perspective for *cordic_optimized_prj/solution1* from 2.e.
    a. Explain the schedule in your hand-in
    b. Try predicting the effect of unrolling the for loop by a factor of 2, then create a new solution (*solution2*) and check your hypothesis. Report your findings in your hand-in. Hint: add an `UNROLL` directive to the `for` loop.
    c. What happens if you change the unroll factor to 4? Create a new solution (*solution3*) and find out. Why doesn't the performance improve? How could you eliminate the bottleneck? Report your findings and discuss them in your hand-in.
    d. Create a new solution (*solution4*) to pipeline the loop without unrolling it. Compare the results with that of unrolling (*solution2*) in your hand-in.
    e. What other optimizations would you try to improve the performance and minimize the increase in area of the implementation? Discuss in your hand-in.

4. How does the constant `NUM_ITERATIONS` affect the area, throughput, and precision? (Create a new solution and edit this constant in `cordic.h`) How does this affect the initial values of `current_cos` and `current_sin`? Do you need to modify the array `cordic_phase`? Can you optimize the data types depending upon the value of `NUM_ITERATIONS`? Respond to these questions in your hand-in for Week 3.