ELEC 3017 Assignment

In this coursework our aim is to design a pipelined Vectoring CORDIC processor using Carry-Save Arithmetic (CSA). In order to carry out the design you first will need to design the basic components of CORDIC – adder/subtractor, sign extractor and finally putting these components to design a full CORDIC processor. Therefore we will proceed in step-by-step fashion with each step carrying marks as given below.

(a) Design a clocked 16-bit CSA adder/subtractor using System Verilog and test it using Modelsim. You should **strictly follow** the input-output definition as shown below, where VS\_x and VC\_x denote the virtual sum and virtual carry vectors for input x, VS\_y and VC\_y are the virtual sum and virtual carry vectors for input y and VS and VC are the outputs. The signal add\_sub is used to control the operation of the adder/subtractor, add\_sub = ‘0’ will initiate addition operation and ‘1’ will initiate subtraction operation. Note that the add\_sub line is in effect the typical carry input to the adder. Whenever the data\_in signal is asserted the adder will assume the input data are valid for computation and similarly the valid data output will be indicated by asserting data\_out signal. The adder should operate at positive clock edge (clk) and the module could be reset by asserting the reset signal to “high”.



(Marks = 5)

AMENDMENT: Please follow the naming convention of the input/output as shown in the above figure. This figure has been changed since the first version of the assignment. Also some explanation on the add\_sub line is added.

(b) Design a clocked sign-detector circuit for 16-bit CSA number using System Verilog and test it using Modelsim. Once again you should **strictly follow** the input-output definition as shown below where VS and VC are the inputs and sgn is the output (sgn = ‘1’ is positive and sgn = ‘0’ is negative). Again the behaviour of the circuit with respect to clock (clk), reset, data\_in and data\_out as exactly the same as in the problem (a).



(Marks = 5)

(c) Combine the above modules from (a) and (b) to design a 16-bit CSA vectoring CORDIC in pipelined form with 16 elementary rotational stages in System Verilog and test it using Modelsim. The CORDIC processor will give out the result at the positive clock edge of every clock cycle. This design should have a data\_valid\_in signal at the input apart from clock and reset so that when this signal is “high” the CORDIC will consider the data at input as a valid data to process. Similarly at the output it should have a data\_valid\_out signal so that if it is “high” the valid result is outputted. While designing the processor you need to **strictly follow** the nomenclature at the input and output as shown in the following diagram, where x, y and z are the input data:



(Marks = 15)

**Testing strategy:**

For testing of all the three modules (a), (b) and (c) use only 16-bit positive fractional numbers. Write the testbench and give a continuous stream of data into your respective modules to check the final results at successive positive clock edges. Ensure that your modules in (a) and (b) work perfectly before using them to design the final CORDIC processor. Again for the final CORDIC testing, give a continuous stream of data at every positive clock edge and check the output at each clock cycle to ensure its correct working. You should test your designs with as many as possible random data and with different clock frequencies to ensure the correct operations of your designed modules.

**Marking strategy:**

You designs will be checked using automated scripts which will strictly follow the input-output signals and module names as shown in the figures. Therefore any mismatch of input-output and the module names will cause the automated script to fail. In such cases we will not be able to award marks for your designs – be very careful about that.

We will act as “customer”, i.e., we will simply check the correctness of the operation. However small the mistake is will be considered as the design failing to show correct operation and we could not award marks. So be careful to ensure that each of your design contains ALL the design files required for the design (this is a common mistake). You designs will be checked with random data and with different clock frequencies as well as by changing the assertion times of reset and input data valid signals to mimic practical operation scenarios. If your design is correct it will give correct results for any of these conditions. Otherwise, if it works only for a few inputs the design will be considered wrong. The correctness of operation is defined as the output error less than 2-10 compared to the standard calculator based result.

You need to submit each of the designs of (a), (b) and (c) in three separate folders containing all of its respective design files and unambiguously named with the name of the modules. We will download the folders and will run the scripts on the files.

If you have any question please feel free to ask me.

GOOD LUCK!