Design of a 16-bit CORDIC computer

PURPOSE – In this lab you will design and implement a 16-bit CORDIC computer. The design to be implemented is based on a bit-serial configuration. It will take as input a 16-bit signed binary fixed point number, corresponding to an angle in the range 0 to π/2, and use the CORDIC method to find the sine and cosine of this angle. This will be coded in Verilog and implemented on the Basys 3 board.

Introduction

Coordinate Rotation Digital Computer was invented by Jack Volder in 1956 [1] as a replacement for the analog resolver used in the navigation systems of the B-58 bomber. At this time digital hardware was limited and it was a significant challenge to do this in real-time. The CORDIC method was a scheme that replaced the multiplications with shifts and adds. Two shift registers and two serial adder-subtractor were used. A third shift register and serial adder-subtractor was used with a ROM to complete the implementation.

There are two modes in which CORDIC can operated. ROTATION and VECTORING modes. For the calculation of sine and cosine we will operate the CORDIC computer in ROTATION mode for which the equations of interest are

$$\boxed{\begin{aligned} x_{i+1} &= x_i - d_i y_i 2^{-i} \\ y_{i+1} &= y_i + d_i x_i 2^{-i} \qquad\qquad d_i = \begin{cases} d_i = +1 & \text{for } z_i \geq 0 \\ d_i = -1 & \text{for } z_i < 0 \end{cases} \\ z_{i+1} &= z_i - d_i \tan^{-1}(2^{-i}) \end{aligned}}$$

After *n* iterations the above equations produce

$$x_n = \frac{1}{K_n}[x_0 \cos(z_0) - y_0 \sin(z_0)]$$

$$y_n = \frac{1}{K_n}[x_0 \sin(z_0) + y_0 \cos(z_0)]$$

$$z_n = 0$$

where $K_n = k_0 k_1 \cdots k_n = \prod_{i=0}^{n} \frac{1}{\sqrt{1+2^{-2i}}}$ is a constant that, for a given *n*, can be computed offline. For large *n* the constant K_n will converge to 0.607253.

For an angle θ in the range 0 to π/2 the CORDIC method can be used to compute cos(θ) and sin(θ). If we set the initial values as $x_0 = K_n$, $y_0 = 0$, and $z_0 = \theta$, when the computation is complete we have $x_n = \cos(\theta)$, $y_n = \sin(\theta)$, and $z_n=0$. (For n=16 we have $K_{16} = 0.607253$)

FPGA Implementation

The implementation in of CORDIC in an FPGA is discussed in [2]. There are a number of possibilities, the first would be to implement the three equations for x, y, and z directly, this is a bit-parallel implementation. However the bit-parallel implementation requires large shift registers which typically do not map well into an FPGA. A better way is a bit-serial implementation which can be clocked near the maximim possible on an FPGA and avoids the problem with a large shift register. The bit-serial design is typically much more compact and better performing than the bit-parallel design. In this lab be implement a bit-serial design.

Note that your computations in this lab will use signed binary fixed point fractions. These are discussed on pages 242 – 243 of the text [4].

CORDIC Datapath

A bit-serial implementation of CORDIC is shown in the following datapath. This implementation is for 16-bits. The bit-serial implementation corresponds to the original implementation developed by Volder. It consists of 3 sets of shift registers, serial adder-subtractors, and two input multiplexers. Two of the shift registers, x and y, are coupled via 13 input multiplexers to the serial adder-subtractor in the other registers datapath. The third register, z, is input to the third serial adder-subtractor along with an inverse tangent value stored in a table in a serial ROM.

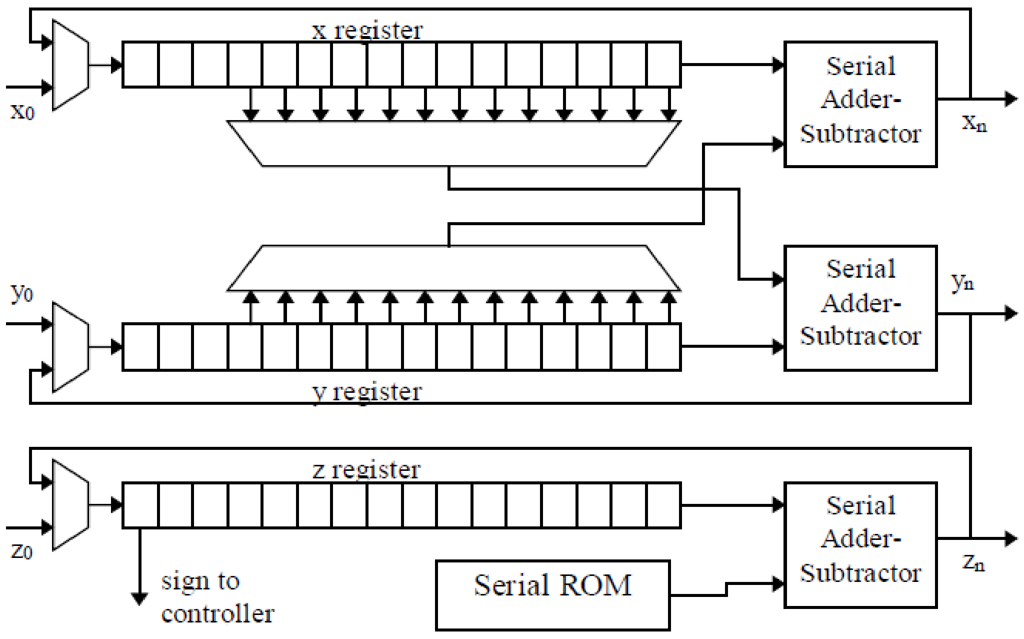

Figure 1. Datapath of the 16-bit CORDIC computer

This hardware is based on three, 16-bit shift registers. Each register is clocked into a 1-bit adder-subtractor. The x and y values for each iteration are contained in these registers. These registers are interconnected. The z register is used with a serial ROM which contains a table of inverse tangent values used in updating the z register each iteration. You will need to compute the values to put in the

ROM (easily done with something like MATLAB.) The values are all represented as signed two's complement fixed point fractions. Use the format 2.14 (2 bits to the left of the binary point and the remaining 14-bit, the fractional part, to the right of the binary point.)

A Verilog model of this can be built from several components. With reference to the figure 1 for the datapath we need three 16-bit parallel in/ parallel out shift registers, three 1-bit adder-subtractor units, three 2-input and two 13-input multiplexers, and a ROM with thirteen, 16-bit words.

Controller

In operation we require a total of *n* times *k* clocks where *n* is the number of iterations and *k*-bits is the word size (16-bits for the hardware shown in figure 1. The initial values of x_0, y_0, and z_0 are loaded into the corresponding registers through the multiplexers. Each iteration the values in the x and y registers are added or subtracted to the appropriately shifted values of the *y* and *x* registers. The sign of the msb of the *y* register is used to determine if we should add or subtract. For the *z* registers the value is added or subtracted to the appropriate arc tangent value contained in the ROM. A simple state machine or counters are needed to keep track of the iterations, amount of shift, and ROM address. On completion the results are read directly from the *x* and *y* registers.
You should list the inputs and outputs to the controller and construct a state diagram or state table.

Input and Output

Once you have the CORDIC Verilog code debugged you should add input and output modules. For this lab initial values will be loaded into the x, y, and z registers. Initially x will be loaded with the value of K_n and y will be loaded with 0. These can be hardwired into your design. The initial value in the z register will correspond to the angle θ, in radians. You should use the 16 switches on the Basys 3 board to select the value of θ to be entered. A pushbutton should be used to enter the z value.

After the initial values are loaded into the x, y, and z registers a second push button should be used to start each iteration. Each iteration will consist of 16 shift operations at the end of which each of the x, y, and z registers will have been updated. You should display the value in the x register using the four 7-segment displays. To implement this in the Basys 3 you need to create a constraints file.

Verifying the design

A good first step is to first calculate the intermediate values for all the iterations with bit parallel equations in a programming language of your choice. This will serve as a baseline to compare against and to verify if your intermediate steps are correct. Calculate for easy to verify input values such as 0, pi/6, pi/2, pi. Next simulate the CORDIC module alone with a test bench and verify the intermediate states and the result.

To verify your design is working your TA will give you an initial angle to load in the z register and will then ask you to step through each iteration to verify you get the correct intermediate and final results on the FPGA board.

SUMMARY – In this lab you designed a Verilog of a CORDIC computer, implemented it on a Digilent Basys 3 board, and verified its correctness.

Lab Notebook deliverables:

- Brief description of what you did in the lab, what steps were followed
- Discussion of results:
    a. What were the test cases you used and discuss why you chose the set of test cases? Explain how your test cases verify each input angle.
    b. Attach a snippet of the reports of the tool.
        i. State transition diagrams and state table. (Refer to the description in the notes section)
        ii. Table and Simulation waveforms (behavioral only) showing all the test cases used above. All test cases must be visible in the waveforms; make sure the FSM state is visible in the waveform.
        iii. Schematic screenshot of synthesized design.
        iv. Utilization report: synthesis and implementation.
        v. Timing report.
        vi. Power report

        NOTE: Only attach relevant sections, not the entire report.
- Summary or Conclusion:
    a. Were all design goals met?
    b. What were the difficulties encountered?
    c. Any improvement you would like to make time permitting? Etc.…
- Verilog Source code: (Font: Courier, size 8pt, single line spacing, no paragraph spacing)

REFERENCES

[1] J. E. Volder, "The CORDIC Trigonometric Computing Technique," I*RE Transactions on Electronic Computers*, EC-8 (3), 330 – 334.

[2] R. Andraka, "A Survey of CORDIC Algorithms for FPGAs,", in Proceedings of rthe 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays (FPGA '98), Monterrey, Feb 22-24, 1998, pp. 191-200.

[3] A. P. Taylor, "How to use the CORDIC algorithm in your FPGA design," Xilinx Xcell Journal, issue 79 (second quarter 2012), 50-55

[4] C. H. Roth, L. K. John, B. K. Lee, *Digital Systems Design Using* Verilog, Cengage, 2016