EE4301 Lab Report:

16-BIT CORDIC COMPUTER

Authors: Yahya Alhinai

Date: August 4, 2018

TA: Shashank Hegde

Professor: Abolghassem Mahmoodi

INTRODUCTION

The purpose of this lab was to implement CORDIC (COordinate Rotation DIgital Computer) using FPGA. CORDIC is a method for calculating a variety of functions including trigonometric and hyperbolic. In this lab, we were interested in calculating the value of sine and cosine function by giving their corresponding angle. These functions are calculated through an iterative set of vector rotations. At the end of these rotations, the value of the function is easily determined from the (x, y) coordinate. CORDIC is used to replace multiplication operations because multipliers are high-cost in terms of implementation on FPGA. Instead, CORDIC only utilize shift and add operations to determine the desire values.

Using Verilog to describe logic gates circuit, we design, Synthesis, and Simulation a simple circuit through Vivado's environment. As well as conducting behavioral simulation of Verilog design circuit to verify its intended functionally. Afterwards, the Verilog described circuit had been converted to logic gates to be implemented on FPGA basys 3. After making sure the circuit is working flawlessly on Vivado's simulation, a bin stream file had been generated to be pushed to the FPGA basys 3 and stored on the memory chip in the board. The switches on the board were coded to be inputs of the angle and the 4 digits 7-segment display were treated as outputs. In addition, there was a dedicated Button to switch between the result of sine function and cosine function.

MAIN

The CORDIC algorithm was introduced to compute trigonometric functions and generalized to compute linear and hyperbolic functions. The iteration or microrotation in circular coordinates is:

$$\mathrm{x}(i+1) = x(i) + \alpha_i * 2^{-1} * y(i)$$

$$\mathrm{y}(i+1) = y(i) - \alpha_i * 2^{-1} * x(i)$$

$$\mathrm{z}(i+1) = z(i) - \alpha_i * \tan^{-1}(2^{-i})$$

Where $\mathrm{x}(0)$ and $\mathrm{y}(0)$ are the initial coordinates of the vector and the z coordinate accumulates the angle. The coefficient α_i which is either 1 or -1 specifies the direction of each microrotation. In order to have a produce 16-bit precision, 17 iterations are needed to meet this goal. The final coordinates are scaled by the factor:

$$k = \prod_{i=0}^{n} \frac{1}{\sqrt{1 + 2^{-2i}}}$$

The sine and cosine functions can be simultaneously obtained by means of the CORDIC algorithm using circular coordinates and in the rotation mode if the initial vector is set to $x = k$, $y = 0$. In this case, it is not necessary to compensate for the scale factor.

After building up the algorithm, series of tests have been conducted to verify the results. Then, the 7-segment display was implemented to show the actual results of the sine and cosine function right after the angle is giving as an input. It takes 17 iterative cycles for 16-bit CORDIC computer to land on the desired results. Switching between the results of sine and cosine is implemented using a button.

Before pushing the software into basys 3 board, the constrain file had been modified accordingly to accommodate the change in circuits as well as adding a button to switch between states. When the button is not pressed, the board will compute the given inputs and it will output the results from the sine function. When the button is pressed, however, the results will be outputted for the cosine function. The results are shown in the board using 7-segment digits that are in the board. The circuit schematic that implements 16-bit CORDIC computer is shown below.

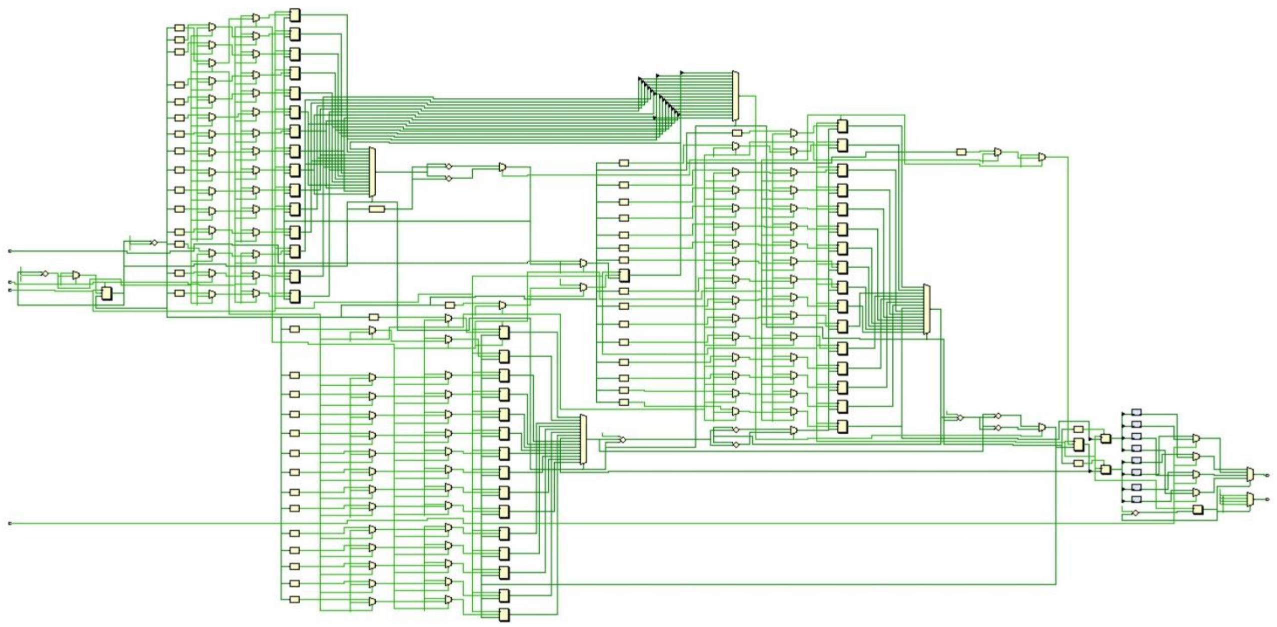


Figure: The circuit schematic that implements 16-bit CORDIC computer

After the bin file was uploaded to the board, different inputs have been provided to make sure and the 16-bit CORDIC computer are working as they meant to be.

CODE EXPLANATION

1. Display refresher:

```
reg [20:0] refresh_counter;

always @(posedge clk) refresh_counter <= refresh_counter + 1;

always @(*) begin
    case (refresh_counter[20:19])
        2'b00: begin
            AN  <= 4'b1110;
            seg <= btnR ? Seg5 : Seg1;
        end
        2'b01: begin
            AN  <= 4'b1101;
            seg <= btnR ? Seg6 : Seg2;
        end
        2'b10: begin
            AN  <= 4'b1011;
            seg <= btnR ? Seg7 : Seg3;
        end
        2'b11: begin
            AN  <= 4'b0111;
            seg <= btnR ? Seg8 : Seg4;
        end
    endcase
end
```

This code is Meant to turn on the 4 digit 7-segment display properly. This display is made to turn one digits at a time to make wiring the display circuit simpler. Therefore, this code is displaying one digit each time but switching really fast between the 4 digits that human eyes will observe the digits working simultaneously. Each digit is corresponding to 2 different values to show sine and cosine. These two different value is determined if the button is pressed or not. If the button is pressed in will show the value of cosine and if the value is not pressed it will show the values cosine.

2. Binary to hexadecimal convertor:

```
To_hex TH0 (Seg1, sin[3:0]);
To_hex TH1 (Seg2, sin[7:4]);
To_hex TH2 (Seg3, sin[11:8]);
To_hex TH3 (Seg4, sin[15:12]);

To_hex TH4 (Seg5, cos[3:0]);
To_hex TH5 (Seg6, cos[7:4]);
To_hex TH6 (Seg7, cos[11:8]);
To_hex TH7 (Seg8, cos[15:12]);
```

This module is responsible for converting binary input to a hexadecimal number that can be sent to one of the 7-segment digits. The module will take 4-bit input and matching it with its corresponding number via a look-up table.

3. Arctangent look-up table:

```
reg signed [15:0] tan_rom [0:15];

    initial begin
        tan_rom[0]  <= 16'b0011001001000011;
        tan_rom[1]  <= 16'b0001110110101100;
        tan_rom[2]  <= 16'b0000111110101101;
        tan_rom[3]  <= 16'b0000011111110101;
        tan_rom[4]  <= 16'b0000001111111110;
        tan_rom[5]  <= 16'b0000000111111111;
        tan_rom[6]  <= 16'b0000000011111111;
        tan_rom[7]  <= 16'b0000000001111111;
        tan_rom[8]  <= 16'b0000000000111111;
        tan_rom[9]  <= 16'b0000000000011111;
        tan_rom[10] <= 16'b0000000000001111;
        tan_rom[11] <= 16'b0000000000000111;
        tan_rom[12] <= 16'b0000000000000011;
        tan_rom[13] <= 16'b0000000000000001;
        tan_rom[14] <= 16'b0000000000000000;
        tan_rom[15] <= 16'b0000000000000000;
    end
```

The look-up table above is meant to replace the results of $\tan^{-1}(2^{-i})$ in registers and call it when it is needed. This is a great replacement for a high-cast FPGA implementation since the input angle of the tangent function is predetermined.

4. CORDIC algorithm:

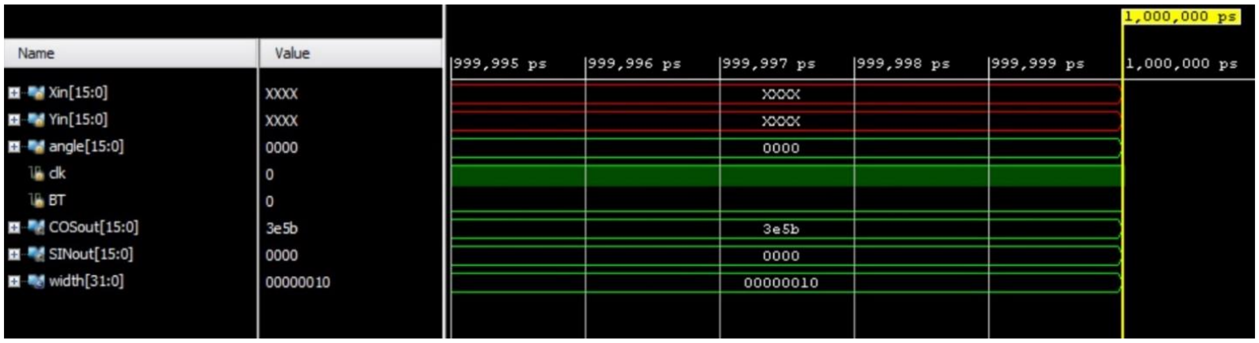
```
    reg signed [16:0] x [0:15];
    reg signed [16:0] y [0:15];
    reg signed [16:0] z [0:15];

    always @(posedge clk)
    begin
        cos <= (i == 16) ? x[15] : 0;
        sin <= (i == 16) ? y[15] : 0;
        if (btnC) begin
            x[0] <= 16'h25dd; // 0.607253 * 2^14
            y[0] <= 0;
            z[0] <= angle;
            i <= 0;
        end
        else if (i < 16)
        begin
            x[i+1] <= z[i][15] ? x[i] + (y[i] >>> i) : x[i] - (y[i] >>> i);
            y[i+1] <= z[i][15] ? y[i] - (x[i] >>> i) : y[i] + (x[i] >>> i);
            z[i+1] <= z[i][15] ? z[i] + tan_rom[i] : z[i] - tan_rom[i];
            i = i + 1;
        end
    end
```

This is the part of the code the responsible for determining the values of sine and cosine at the same time. This operation is meant to be a low-cost FPGA implementation because it only uses shift and addition operations. It works in an iterative order where shift and addition operations are done 16 times consecutively to have a pretty god approximation of the sine and cosine functions. The value will be outputted as soon as it arrives to the last iteration cycle. In addition, there is a dedicated button that work as an input which start the algorithm operation and reset the initial value.

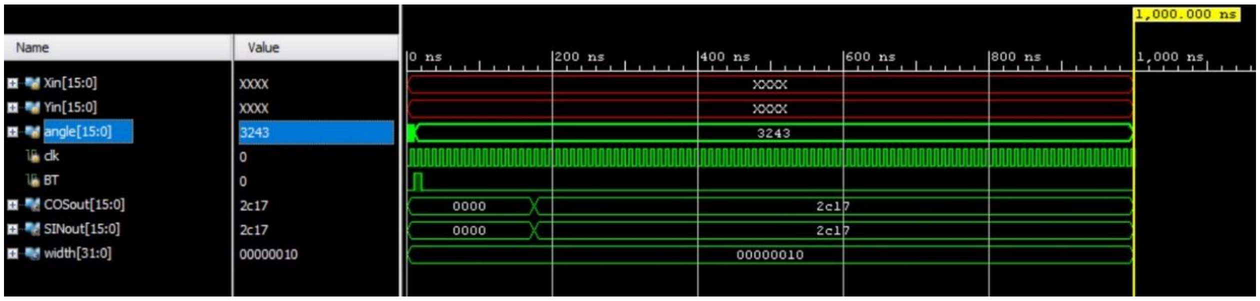
WAVEFORMS FROM TESTBENCH

The testbench results were crucial to set up the algorithm currently to make sure it will function flawlessly before building up the rest of the circuit. After the algorithm was set up to convert an angle to its corresponding sine and cosine values, the testbench was conducted to verify the functionality of the CORDIC algorithm. The testbench values that was tested are 0, 45, and 90 degree as the lab was set up to only calculate the values between 0 and 90 degree.



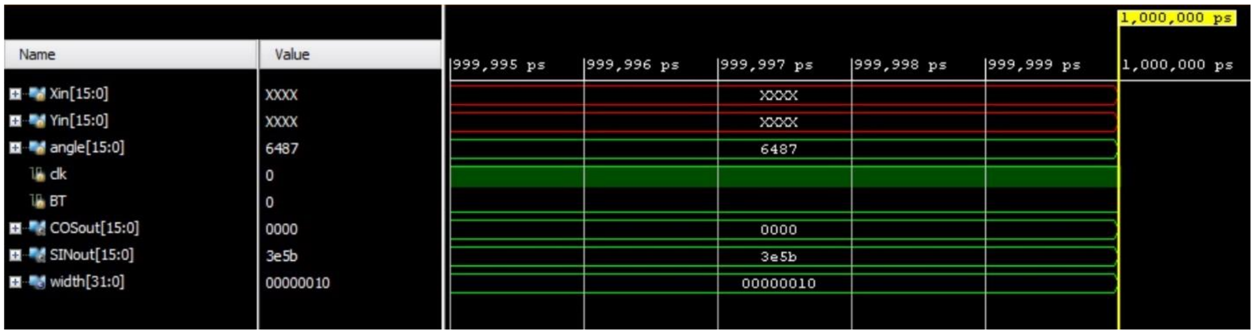
Name	Value
Xin[15:0]	XXXX
Yin[15:0]	XXXX
angle[15:0]	0000
clk	0
BT	0
COSout[15:0]	3e5b
SINout[15:0]	0000
width[31:0]	00000010

Figure: The tested angle (0 degree) and its corresponding sine and cosine value shown as an output



Name	Value
Xin[15:0]	XXXX
Yin[15:0]	XXXX
angle[15:0]	3243
clk	0
BT	0
COSout[15:0]	2c17
SINout[15:0]	2c17
width[31:0]	00000010

Figure: The tested angle (45 degree) and its corresponding sine and cosine value shown as an output



Name	Value
Xin[15:0]	XXXX
Yin[15:0]	XXXX
angle[15:0]	6487
clk	0
BT	0
COSout[15:0]	0000
SINout[15:0]	3e5b
width[31:0]	00000010

Figure: The tested angle (0 degree) and its corresponding sine and cosine value shown as an output

SUMMARY

The goals for Labs 8 have been met. The implementation of 16-bit CORDIC computer circuit have been successful. The generated circuit had been pushed for the FPGA basys 3 board after design, Synthesis, and Simulation the circuit in vivado and conducting behavioral simulation of Verilog design circuit to verify its intended functionally. The results on the board reflect the intended purpose of circuit as the switches on the board were coded to be inputs and 7-segment display were coded to be outputs. The results on the board has been verified and the code has been fully optimized. The improvement that can be made to improve power consumption or gates utilization is using a single register for each function to implement shift and add operations instead of 16 registers that was used in this lab.