EE5516 VLSI Architectures for Signal Processing and Machine Learning

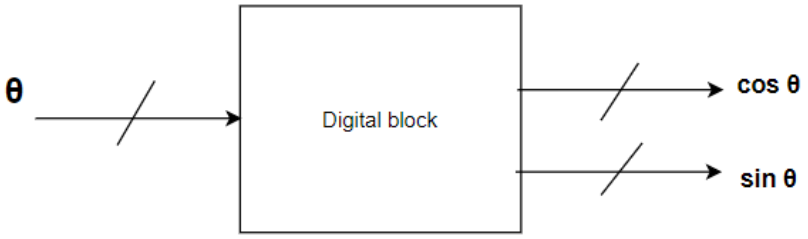
Lab Experiment 4 Report

AMITH P JOY-122001006

Abstract

This experiment involves the design of architecture and implementation in Verilog of a pipelined version of Volder's CORDIC algorithm in the rotation mode. The module takes an angle theta as inputs and returns the cosine and sine of that angle. The values of sine, cosine and theta are given in 2.14 fixed form. The algorithm has a throughput of 1 output per clock cycle and involves 16 stages in estimating sine and cosine values. We also implement a 16 bit barrel shifter using a multiplexer.

1. Introduction



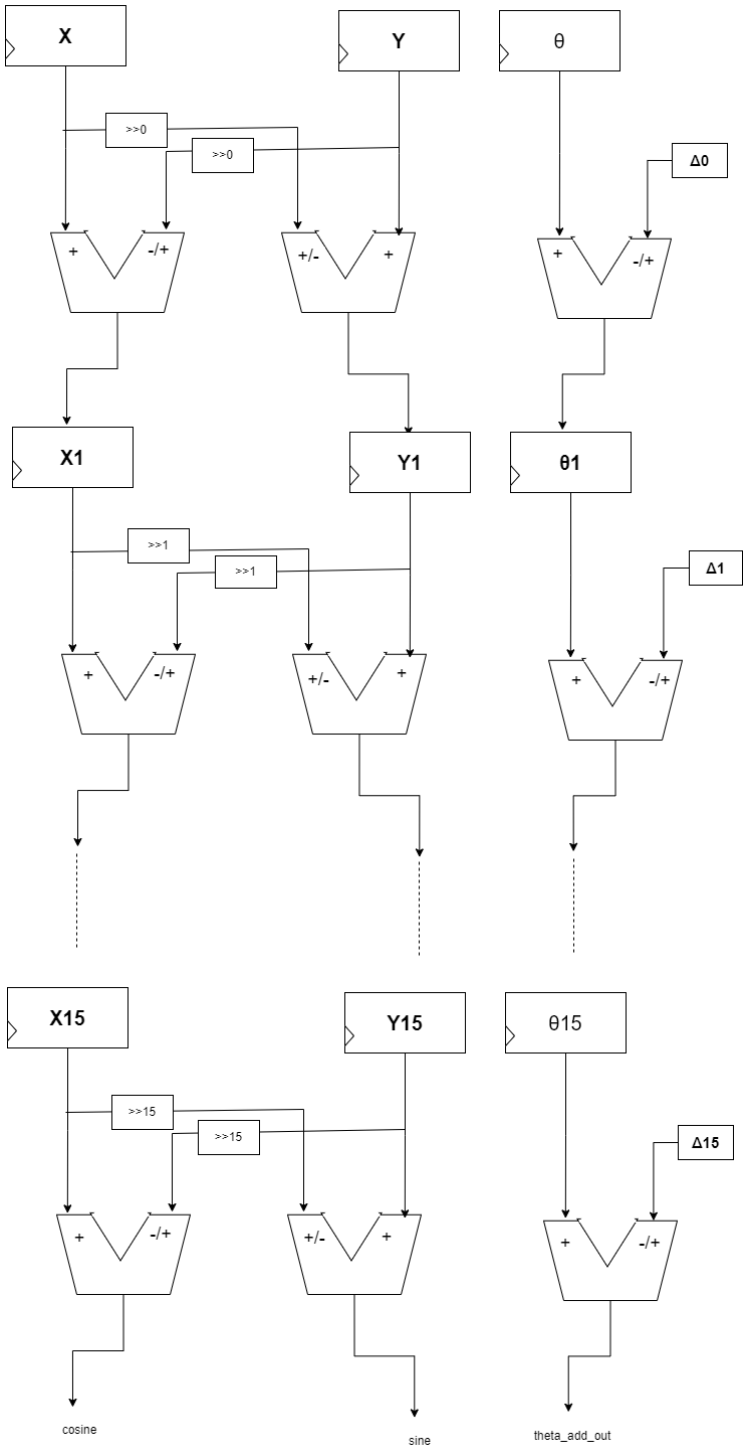
The CORDIC (COordinate Rotation DIgital Computer) algorithm is a simple and efficient method for calculating a variety of mathematical functions, including trigonometric, hyperbolic, and logarithmic functions. It was originally developed for digital computers in the 1950s by Jack Volder, and since then, it has been widely used in a range of applications such as signal processing, robotics, and graphics. The CORDIC algorithm uses a series of elementary rotations to transform a vector from one coordinate system to another. The algorithm works by breaking down the desired rotation into a series of smaller, simpler rotations, each of which can be implemented using only additions, subtractions, and bit-shift operations. The CORDIC algorithm can be used to compute a wide range of mathematical functions, including sine, cosine, tangent, arctangent, hyperbolic sine, hyperbolic cosine, and hyperbolic tangent. Here we use CORDIC to calculate the sine and cosine of an angle. In pipelined CORDIC this is done in 16 stages where each stage is connected to the next stage. The equations for calculation are as follows:

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = cos(\Delta\Theta_i) \begin{bmatrix} 1 & \mp 2^{-i} \\ \pm 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

$$\theta_{e_{i+1}} = \theta_{e_i} \mp \Delta\theta_i$$

Here, i represents the current stage number and i+1 is the next stage.

2. Implementation



The pipelined CORDIC has been implemented using repeating similar arrangements of registers, adder-subtractors and shifters. The inputs x,y and theta are stored in three separate registers. The register with values of x, y and theta at the i^{th} stage is stored in registers named x_i, y_i and $theta_i$. Adder-subtractors are used to calculate the values that will be stored in the next set of registers based on whether the angle in the current stage is positive or negative. The predefined set of CORDIC angles is given directly to the

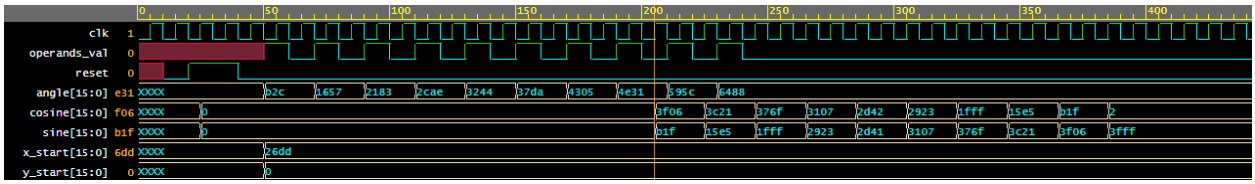
adder-subtractors in each stage correctly eliminating the necessity for a look-up table for storing these CORDIC angles. The MSB of theta in its current state conveys whether the value of theta is positive or negative (1-negative, 0-positive). Shifters are used to divide the values of x and y by the required exponent of 2 for the calculation of x and y for the next stage. The exponent of 2 with which x and y must be multiplied for calculation in each stage is fixed thus eliminating the necessity for a barrel shifter in the hardware. The final value of cosine is given by the adder-subtractor after the register x_{15}, the final value of sine is given by the adder-subtractor after the register y_{15} and the final value of theta will be 0 and given by the adder-subtractor after the register $theta_{15}$. The 16 stages causes a latency of 16 clock cycles for the outputs. We have also avoided using multiplexers, and counters in this version of CORDIC implementation.

3. Experimental Procedure

A testbench to check the working of the designed module has been made. Tasks for driving reset signal, inputs and displaying the outputs have been made in the testbench. The module continuously takes in inputs and calculates output continuously. Thus the output is displayed without delay using the testbench. The input angles are given by reading the hexadecimal values of angles from a text file. The initial value of input x is given as the scaling factor which is equal to the product of cosines of CORDIC angles so that our output the exact values of sine and cosine for any angle between 0 and 90 degrees.

4. Experimental Results

The output waveform is as follows:



Here, the angles can be seen to be continuously given as input and the output values of sine and cosine are given out continuously with a latency of 16 clock cycles. The throughput is one per clock cycle.

```
# KERNEL: Cosine  = 0.000000, Sine  = 0.000000
# KERNEL: Cosine  = 0.984741, Sine  = 0.173767
# KERNEL: Cosine  = 0.939514, Sine  = 0.342102
# KERNEL: Cosine  = 0.866150, Sine  = 0.499939
# KERNEL: Cosine  = 0.866150, Sine  = 0.499939
# KERNEL: Cosine  = 0.766052, Sine  = 0.642761
# KERNEL: Cosine  = 0.766052, Sine  = 0.642761
# KERNEL: Cosine  = 0.707153, Sine  = 0.707092
# KERNEL: Cosine  = 0.707153, Sine  = 0.707092
# KERNEL: Cosine  = 0.642761, Sine  = 0.766052
# KERNEL: Cosine  = 0.642761, Sine  = 0.766052
# KERNEL: Cosine  = 0.499939, Sine  = 0.866150
# KERNEL: Cosine  = 0.499939, Sine  = 0.866150
# KERNEL: Cosine  = 0.342102, Sine  = 0.939514
# KERNEL: Cosine  = 0.342102, Sine  = 0.939514
# KERNEL: Cosine  = 0.173767, Sine  = 0.984741
# KERNEL: Cosine  = 0.173767, Sine  = 0.984741
# KERNEL: Cosine  = 0.000122, Sine  = 0.999939
# KERNEL: Cosine  = 0.000122, Sine  = 0.999939
# KERNEL: Cosine  = 0.000122, Sine  = 0.999939
# KERNEL: Cosine  = 0.000122, Sine  = 0.999939
# KERNEL: Cosine  = 0.000122, Sine  = 0.999939
```

5. Conclusion

The pipelined version of implementation of CORDIC in rotation made has been correctly implemented and the output with the expected latency and throughput has been observed.

EdaPlayground link for CORDIC

Implementation of Barrel Shifter

Implementation:

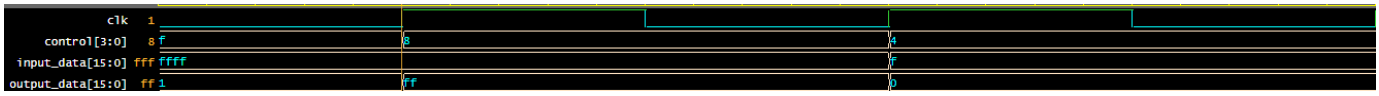
The barrel shifter has been implemented using a multiplexer which will select its output based on the control signal which specifies the number of bits the number has to be right shifted. Hence, the multiplexer will have 16x16 bits as inputs where each set of 16 bits represent the input number right shifted by a number. The 16 bits in the output will determined by the 4 bit control signal to the multiplexer which determines the number of bits shifted.

Experimental Procedure:

A testbench to check the proper working of the multiplexer has been designed. Tasks to drive the input and check the output has also been given in testbench. After a set of inputs are given, the check_output task checks whether the output produced by the barrel shifter is correct or not at every clock cycle. The output is verified by comparing the output value of the barrel shifter module with the input right shifted the same number of times using the logical shift operator.

Experimental Results:

The barrel shifter was given 3 sets of inputs and the output for them are as follows:




```
# KERNEL: output of shifter is:
# KERNEL: 0001
# KERNEL: expected output:
# KERNEL: 0001
# KERNEL: correctly working
# KERNEL: output of shifter is:
# KERNEL: 00ff
# KERNEL: expected output:
# KERNEL: 00ff
# KERNEL: correctly working
# KERNEL: output of shifter is:
# KERNEL: 0000
# KERNEL: expected output:
# KERNEL: 0000
# KERNEL: correctly working
```

The output has been verified to be correct and the barrel shifter shows no delay in its output and doesn't depend on any other signal other than its inputs to giv an output.

Conclusion

The barrel shifter has been correctly implemented and can be used in other midules like CORDIC for division of a number an exponent of 2.

EDAPlaygroind Link For Barrel Shifter