Circular iterative CORDIC using Dual Fixed-Point Arithmetic

Daniyah Alaswad, Yazen Alali
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: dhalaswa@oakland.edu, yazenalali@oakland.edu

Abstract—**This work presents an architecture for Circular CORDIC computation in binary dual fixed-point arithmetic that is based on the Circular CORDIC algorithm. It involves designing an embedded system using the ZYBO board. The project includes a Digital system which is the CORDIC, Bus AXI Lite interface, SDK that considered as a software routine that controls the digit system by streaming data through UART. The goal is to be able to get the actual result from dual fixed-point operations within the CORDIC showing in the SDK and compare the result to MATLAB result.**

I. INTRODUCTION

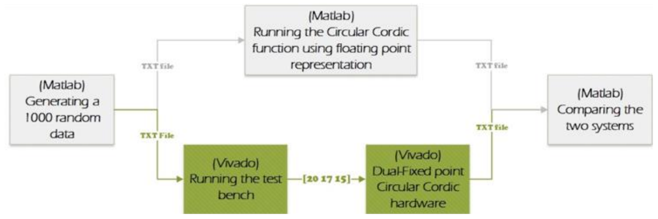
All Digital Signal Processing tasks can be efficiently implemented using processing elements performing vector rotations. The Coordinate Rotation Digital Computer algorithm CORDIC offers the opportunity to calculate all the desired functions in a rather elegant and straightforward way. CORDIC (for Coordinate Rotation Digital Computer), also known as Volder's algorithm, is a simple and efficient algorithm to calculate hyperbolic and trigonometric functions, typically converging with one digit (or bit) per iteration. Is an iterative algorithm for calculating trigonometric functions including sine, cosine, magnitude, and phase? It is particularly suited to hardware implementations because it does not require any multiplies. We decided to use Circular CORDIC using Dual Fixed-point Arithmetic.

The motivation of this project was that Dual fixed point Athematic combines the simplicity of a fixed-point system with the wider dynamic range offered by a floating-point system. Also, by Using a single bit exponent which selects two different fixed-point representations, it allows dynamic scaling of signals throughout the system. Another reason is that there is not enough resources for this Arithmetic and it not widespread. Although, it has better precision than fixed point Arithmetic.

We were successfully able to implement circular CORDIC using fixed point Arithmetic previously, which gave us experience in how it works. Afterword, we decided to make a dual fixed-point CORDIC because we would like to Apply the knowledge acquired in designing the DFX CORDIC and we have not to work with it before. The report will cover the procedure, Results, and Challenges of the project.

II. METHODOLOGY

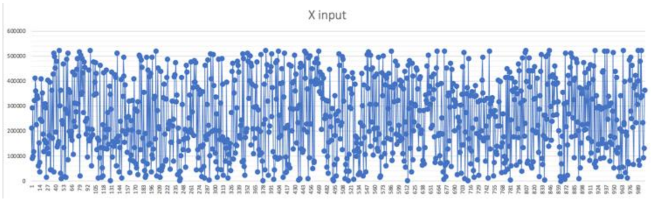
The design of this project contains six different steps which are: data generation via MATLAB, the operation, the CORDIC, the comparison via MATLAB, the AXI-Lite bus interface, and the SDK software output.

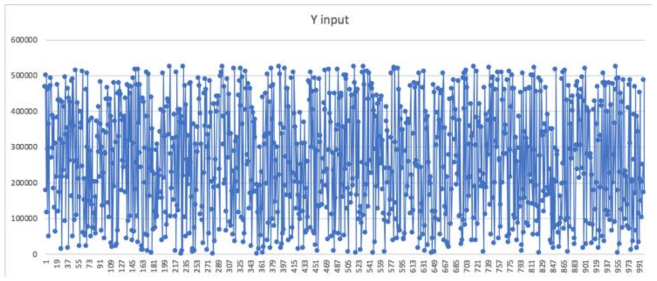

(Matlab) Generating a 1000 random data
TXT file
(Matlab) Running the Circular Cordic function using floating point representation
TXT file
TXT File
(Vivado) Running the test bench
[20 17 15]
(Vivado) Dual-Fixed point Circular Cordic hardware
TXT file
(Matlab) Comparing the two systems

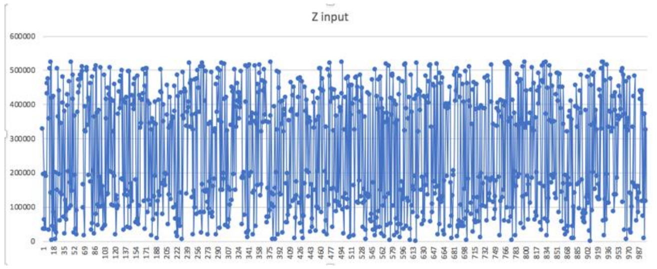
A. *Generation Data via Matlab.*

The first thing in this project was generating a large amount of data to test the circuit that we are going to build. We found out that the best way to approach this is by writing a MATLAB script to generate a large amount of data. We wrote the script to be able to produce a 1000 random number using function rand (). We initialize the dual fix point representation to [20 17 15] with boundaries for num0 and num1 using the equations.

Moreover, we decided to set the value of Xin and Yin to a range of (-8,8), and Zin to a range of (-pi/2, pi/2). We created a function that takes the random floating-point number and converts it to fixed point number. All this operation was in binary, but we need it in integer form to access it easily via testbench. After converting the numbers from binary to integers, MATLAB will print the data to three text files for Xin, Yin, and Zin. Below is the chart of how input data look like.







B. *The operation*

The second part of the project is doing the operation that we are going to use in the CORDIC. We decided to do adder and subtractor. In binary fixed-point arithmetic, we do a similar process to fixed-point arithmetic, but we need to consider two more things. We have first to decide if the number is num0 or num1 then apply range detector to convert three numbers to duel fixed-point numbers before doing the operations. Also, we have to consider in mind the overflow that might happen during the procedure. Here is the circuit of dual fixed-point adder and subtractor. In figure1 below, we can see that the circuit required an FSM and control unit. We have to consider overflow while doing the FSM if needed.

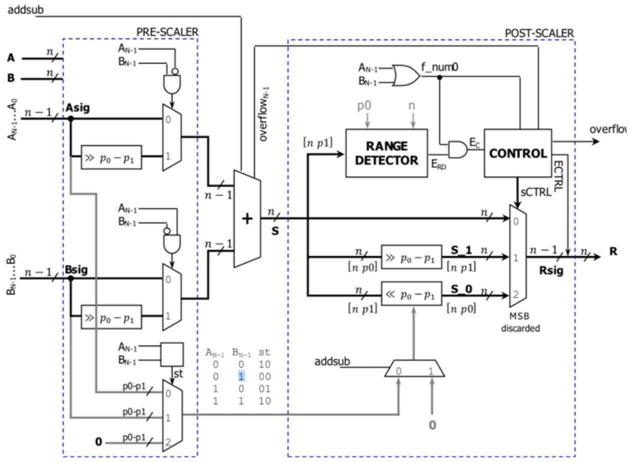


Figure 1: dual fixed-point adder and subtractor.

C. *The CORDIC*

The original circular CORDIC algorithm is described by the following iterative equations, where ⍰ is the index of the iteration. Depending on the mode of operation.
Here is the equation of Circular CORDIC.

$$x_{i+1} = x_i + \delta_i y_i 2^{-i}$$
$$y_{i+1} = y_i - \delta_i x_i 2^{-i}$$
$$z_{i+1} = z_i + \delta_i \theta_i,\ \theta_i = Tan^{-1}(2^{-i})$$

For Rotational mode we used these equations:

$$x_n = A_n(x_0 cosz_0 - y_0 sinz_0)$$
$$y_n = A_n(y_0 cosz_0 + x_0 sinz_0)$$
$$z_n = 0$$

For vectoring mode, we used these equations:

$$x_n = A_n\sqrt{x_0^2 + y_0^2}$$
$$y_n = 0$$
$$z_n = z_0 + tan^{-1}(y_0/x_0)$$

After successfully doing the adder and subtractor for dual fixed point and testing the code via behavior simulation and testbench. We can start connecting our adder and subtractor to the CORDIC. We need to do the mapping and parallel shifter which would be a bit challenging. The parallel shifter would affect the data going through the CORDIC. There is two mode in CORDIC which is rotational mode and vectoring more. We need to create a testbench for both cases. The testbench will read the data from the text files generated by MATLAB in the previous step, and it will generate output files for Xin, Yin, and Zin. In the figure below, we need to add a range detector before adder and subtractor which will help accomplish the goal.

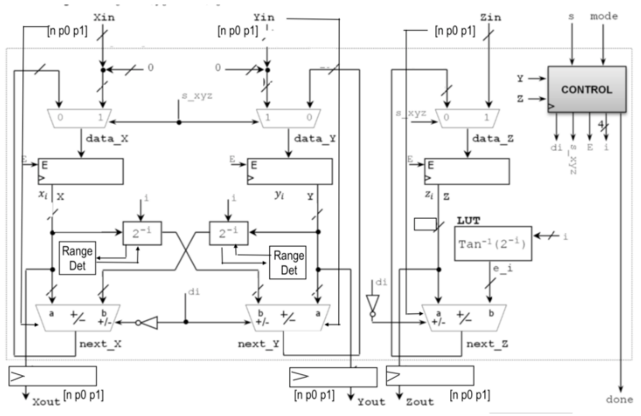
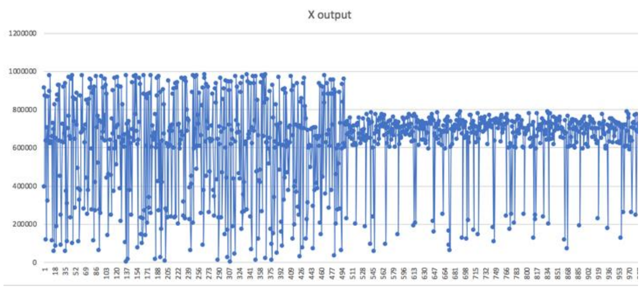
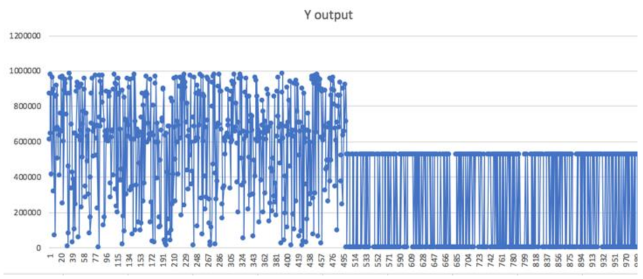
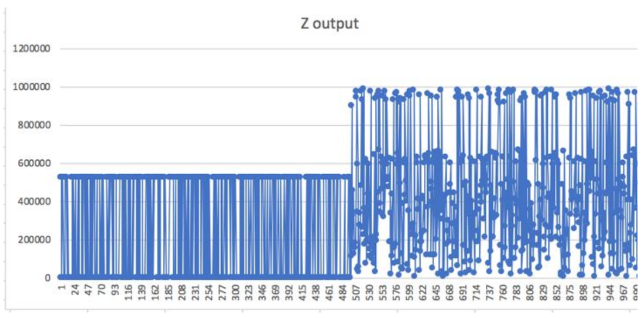


Figure 2: circular fixed point CORDIC circuit.

The output that would be generated by the testbench will look like this:

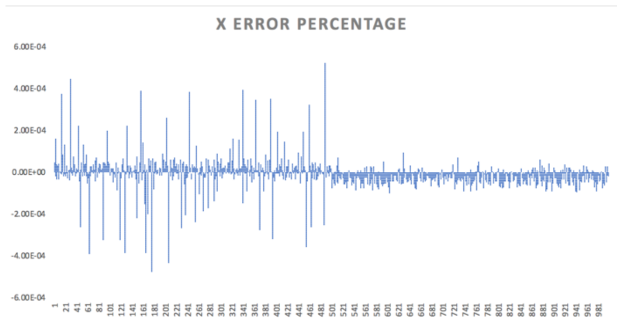


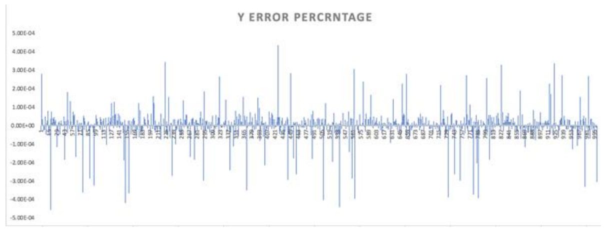


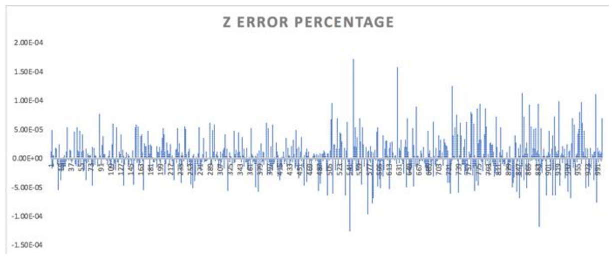


D. *The Comparison Via MATLAB.*

After generating the output of our circuit via VIvado. We decided to check it by compare it to a MATLAB code and discover the error percentage. The MATLAB code will read the data from output text files and convert them to decimal. Then it will read the input text files generated by the previous MATLAB and convert them to decimal too. It will take the input and run it through a CORDIC function in MATLAB to generate an output. After that, it will take that output and compare it with Vivado simulation output. When the comparison is done, MATLAB will Calculate the error percentages. Here we can see how the error percentages look like.







E. *AXI interface*

We used the AXI Lite interface to write vector data from the embedded processor to a continuous group of registers on the Programmable Logic IP Core. The AXI Write block only supports the AXI-Lite protocol, allowing for simple, low-throughput memory-mapped communication. Typical uses for this protocol include writing to control and status registers. After finishing the CORDIC we connected it to AXI Lite interface with eight registers three for inputs, three for outputs, one for done, and one for mode and start. We needed to create and modify the IP of the project and using our top file of the CORDIC and assign the number of bits to twenty. Then we implemented a block diagram of the project, so we would be able to generate a bitstream for it.

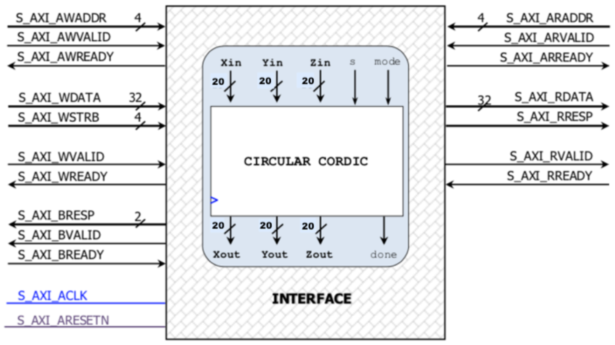
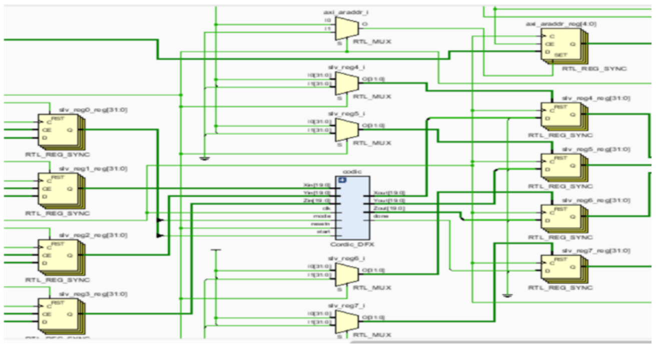
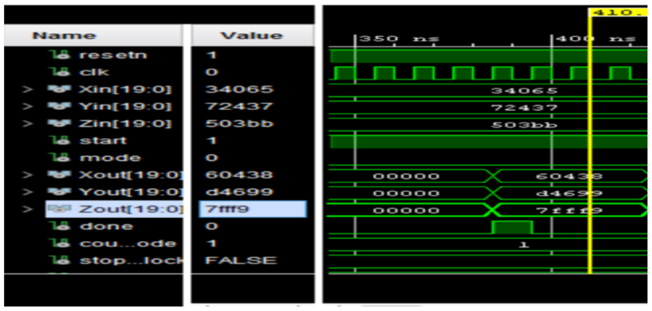


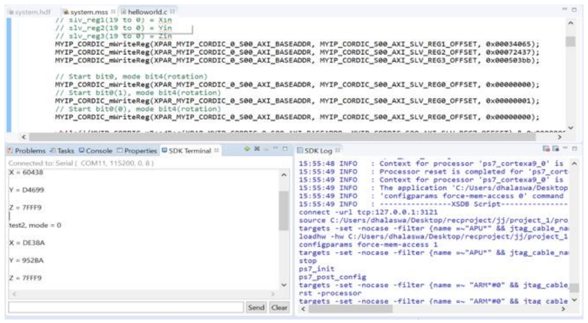
Figure3: AXI LITE fixed point CORDIC.



F. SDK OUTPUT

When we were successfully able to generate bitstream we can launch SDK. We have to write a c code that will test rotational and vectoring cases we would like to check. We can test the code multiple times and compare it to the behavior simulation of dual fixed-point CORDIC in order to make sure that the answer is correct and that it is working on hardware.


```
Name            Value
resetn          1
clk             0
Xin[19:0]       34065
Yin[19:0]       72437
Zin[19:0]       503bb
start           1
mode            0
Xout[19:0]      60438
Yout[19:0]      d4699
Zout[19:0]      7fff9
done            0
cou...ode       1
stop...lock     FALSE
```



```
// slv_reg1(19 to 0) = Xin
// slv_reg2(19 to 0) = Yin
// slv_reg3(19 to 0) = Zin
MYIP_CORDIC_mWriteReg(XPAR_MYIP_CORDIC_0_S00_AXI_BASEADDR, MYIP_CORDIC_S00_AXI_SLV_REG1_OFFSET, 0x00034065);
MYIP_CORDIC_mWriteReg(XPAR_MYIP_CORDIC_0_S00_AXI_BASEADDR, MYIP_CORDIC_S00_AXI_SLV_REG2_OFFSET, 0x00072437);
MYIP_CORDIC_mWriteReg(XPAR_MYIP_CORDIC_0_S00_AXI_BASEADDR, MYIP_CORDIC_S00_AXI_SLV_REG3_OFFSET, 0x000503bb);

// Start bit0, mode bit4(rotation)
MYIP_CORDIC_mWriteReg(XPAR_MYIP_CORDIC_0_S00_AXI_BASEADDR, MYIP_CORDIC_S00_AXI_SLV_REG0_OFFSET, 0x00000000);
// Start bit0(1), mode bit4(rotation)
MYIP_CORDIC_mWriteReg(XPAR_MYIP_CORDIC_0_S00_AXI_BASEADDR, MYIP_CORDIC_S00_AXI_SLV_REG0_OFFSET, 0x00000001);
// Start bit0(0), mode bit4(rotation)
MYIP_CORDIC_mWriteReg(XPAR_MYIP_CORDIC_0_S00_AXI_BASEADDR, MYIP_CORDIC_S00_AXI_SLV_REG0_OFFSET, 0x00000000);
```

```
Connected to: Serial ( COM11, 115200, 0, 8 )
X = 60438

Y = D4699

Z = 7FFF9

test2, mode = 0

X = DE38A

Y = 952BA

Z = 7FFF9
```

```
15:55:48 INFO   : Context for processor 'ps7_cortexa9_0' is
15:55:49 INFO   : Processor reset is completed for 'ps7_cort
15:55:49 INFO   : Context for processor 'ps7_cortexa9_0' is
15:55:49 INFO   : The application 'C:/Users/dhalaswa/Desktop
15:55:49 INFO   : 'configparams force-mem-access 0' command
15:55:49 INFO   : ----------------XSDB Script--------------
connect -url tcp:127.0.0.1:3121
source C:/Users/dhalaswa/Desktop/recproject/jj/project_1/pro
targets -set -nocase -filter {name =~"APU*" && jtag_cable_na
loadhw -hw C:/Users/dhalaswa/Desktop/recproject/jj/project_1
configparams force-mem-access 1
targets -set -nocase -filter {name =~"APU*" && jtag_cable_na
stop
ps7_init
ps7_post_config
targets -set -nocase -filter {name =~ "ARM*#0" && jtag_cable
rst -processor
targets -set -nocase -filter {name =~ "ARM*#0" && jtag cable
```

III. *Experimental Setup*

In the experiment we are going to use ZYBO board to test the result which will communicate with the SDK software, the main part will be done in VIVADO 2018 software using VHDL and C language.

IV. *Results/ Challenges*

Some of the challenges we faced was generating a large amount of data in short time and deciding how many registers do we need for the interface. Also, we encountered some problem with the parallel shifter.

After comparing the outputs from MATLAB with our simulation, we had results that could give us the indication of how much accurate our hardware is. Comparing these two output is actually comparing floating point values with dual-fixed point values. Table below shows the maximum errors we got in X, Y and Z values.

Output Values	X	Y	Z
Max error in %	**0.026%**	**0.022%**	**0.0085%**

Table: Maximum error in X, Y and Z

V. *Conclusions*

We were successfully able to Apply the knowledge of DFX arithmetic in coding. Also, it is evident that DFX does almost give accurate results compared with floating numbers with much less operation time. We did Apply the knowledge of different interfaces such as AXI Bus and IP. for future work, and we could Increase the number of data. Moreover, Test more dual fixed format to find out the most suitable one. We could Also do a partial configuration that changes hardware format as needed. Also we could use the idea of reading the data from SD card to SDK.

REFERENCES

[1] Llamocca, Daniel. “Embedded System Design for Zynq SoC “.” Reconfigurable Computing Research Laboratory. N.p., n.d. Web. 8 Mar 2018.