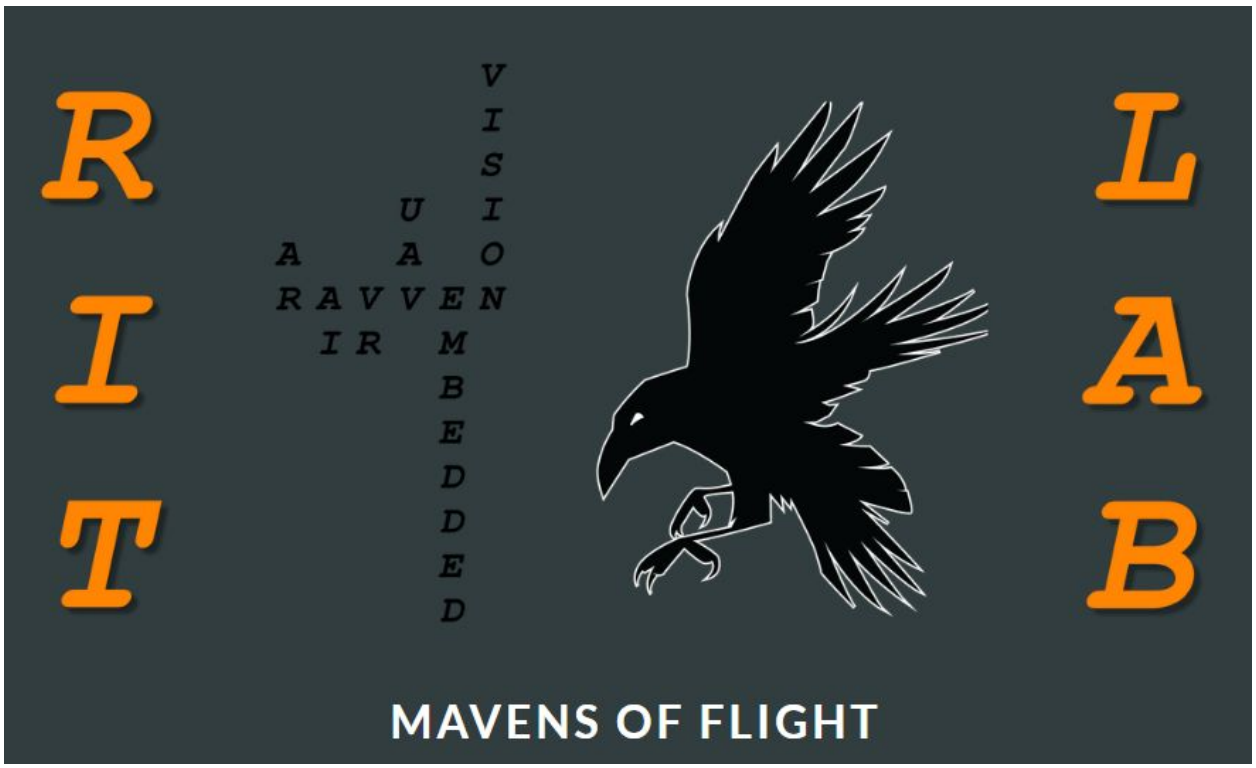
Cordic Arctangent Approximation Hardware Implementation Tech Memo

John J. Niemynski II
12/14/2018



Approximating trigonometric functions for digital signal processing applications has been an increasingly studied area for embedded systems as technology for image processing becomes more widespread. A particularly tricky function to estimate is the arctangent function. A common approach to approximating this function is the Cordic algorithm. The input to the cordic algorithm is the sine and cosine values of some point on the unit circle. These values are then compared to each other to determine the quadrant and from there the cordic algorithm has some predetermined number of constants that ultimately limits how accurate the estimation is. For this arctangent function implementation a cordic algorithm was used. The overall process involved utilizing a matlab model to verify that the error to an actual arctangent process was minimal, a Simulink model was produced and the algorithm verified, VHDL generation from the Simulink model and verified in ModelSim, and finally a hardware implementation verification on a Snickerdoodle system (Zynq 7000).

The matlab model utilized a previous model of a study done on a particular Cordic approximation of arctangent using 12 bits [1]. For other systems' reasons, it was decided that a 16 bit fixed point input and output to this model would be used. Due to the internal range of values in the algorithm itself varying from -pi to pi, it was necessary to use 2 integer bits in our fixed point format. Improving this will be discussed in a later section of this tech memo. Another system constraint was limiting the error of approximation to under 1 degree. Thus, it was a 16 bit fixed point, 11 lut, or 11 constant cordic algorithm was compared to the atan2 function inside matlab to verify the implementation would be meet system requirements. The atan2 function is accurate up 64 bits and Mathworks supplies an atan2 function in Simulink, which is why it was used for comparison. The constants that the model produced can be seen in Table1. Figure 1 supplies the error of the cordic algorithm against the atan2 function for 2000 points between -180 and 180 degrees.

Constant #	Value	Constant #	Value	Constant #	Value
1	1.5708	5	0.1244	9	0.0078
2	0.7854	6	0.0624	10	0.0039
3	0.4636	7	0.0313	11	0.002
4	0.245	8	0.0156		

Table 1: 11 Constant Values produced from Matlab model.

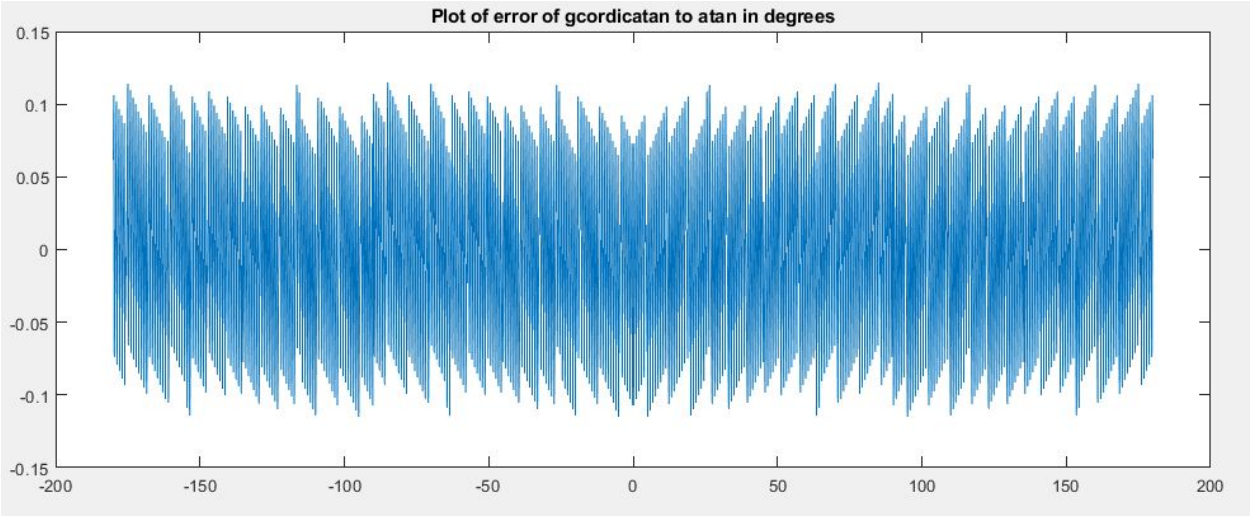


Figure 1: Matlab plot of error between Cordic arctangent algorithm at fixed point 16 bit, 13 fractional bit and atan2 function. Error is between roughly -0.1 and 0.1 degrees for the entire unit circle

The next step was producing the algorithm in Simulink. This is particularly useful as model based design usage in embedded systems in increased. It also makes it much easier to modify and change your design allowing your design to be agnostic to a system if may have been designed for. The only limiting factor is the number of tools Mathworks releases for simulink and which of these modules is compatible with HDL code generation. Luckily, the cordic algorithm uses only constants addition, and comparisons, which makes it a very computationally efficient algorithm. Figure 2 shows the first stage of the algorithm that determines which hemisphere the result is in and the next stage whose logic is repeated for every constant thereafter. Since it is possible that the first two constants can be added together yielding a value greater than 2, it is necessary that the fixed point format used for the algorithm must contain at least two integer bits.

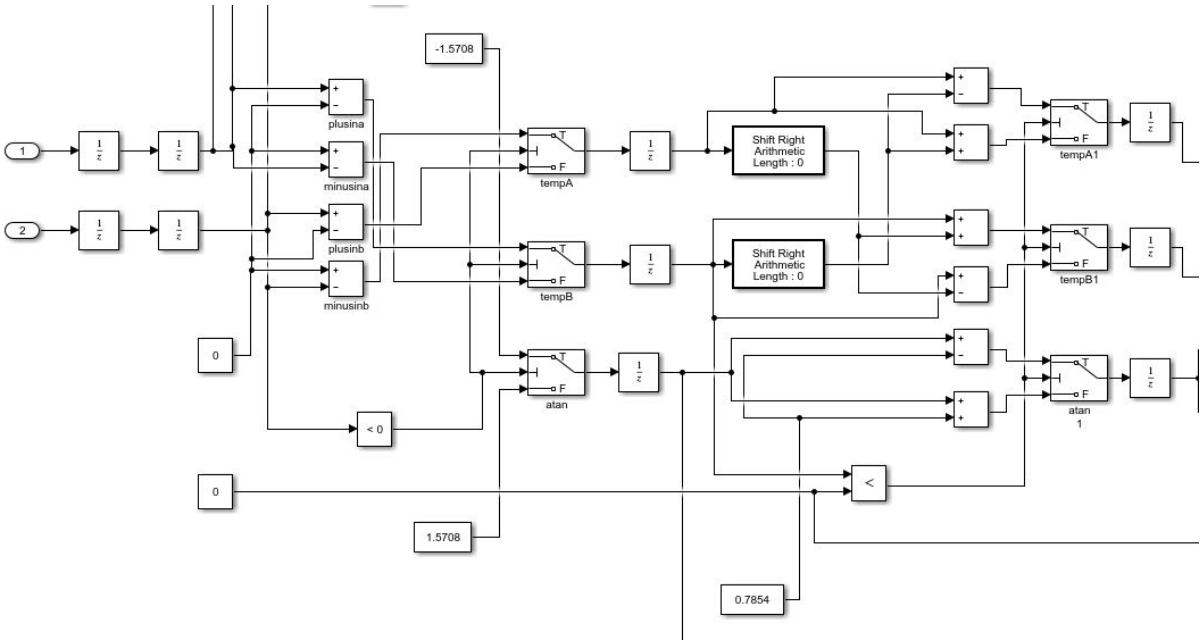


Figure 2: First two stages of the simulink block. Exemplifies comparison of input to 0 to select the value of arctangent and the negative and positive of the input values. Also shows the next stage right shifting in the input values, comparing these to 0 as a sign check and adding or subtracting the inputs depending on the result.

Once this system was produced, it was verified in simulink by inputting 1000 values from -180 to 180 degrees just as the model had done. The test setup also included noise that could be added to the inputs of the model and an atan2 block to compare outputs to, the ability to switch to single values for debugging, and sends input and output data from the simulink block into matlab workspaces to verify the against VHDL data later [Figure 3].

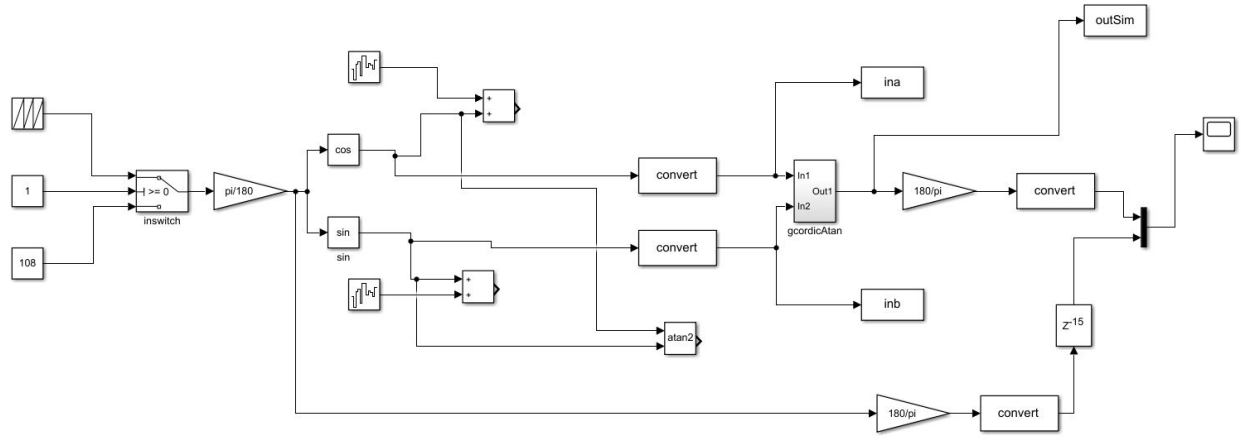


Figure 3:Simulink model test set up connected to inputs and outputs of cordic algorithm

Running the 1000 algorithm points (yellow) against the real values (blue) obtained input into model yields the scope output in figure 4. The initial spike in values is due to the delay in the system requiring 15 clock cycles before the output settles on the correct value.

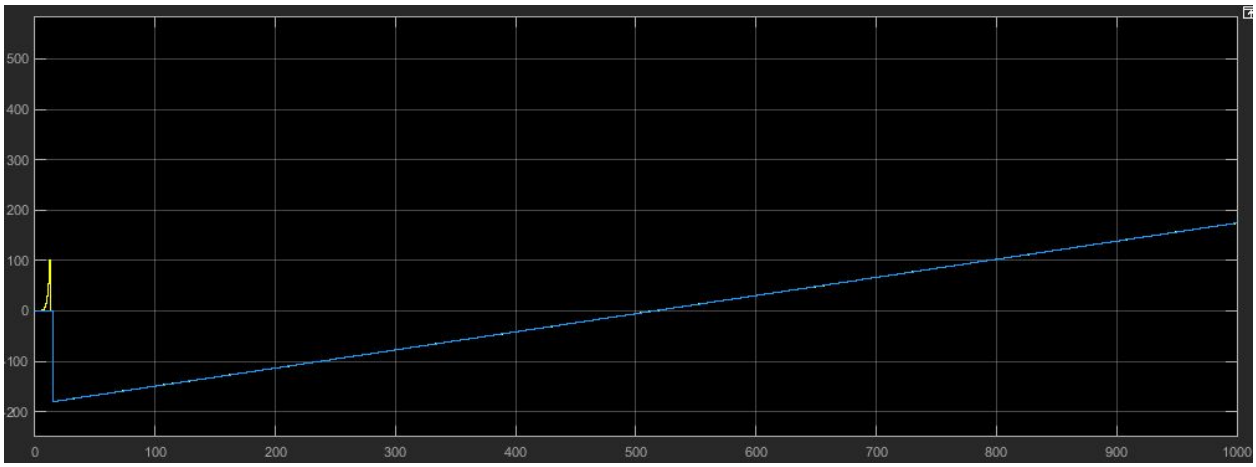


Figure 4: Shows the algorithm values in yellow adhering closely to the real values in blue across all degree values of the unit circle

The testset up was modified to add 0.1 power noise to the input of the algorithm. This was compared to the atan2 function in algorithm across the same domain with the same number of results. FIgure 5 shows that the algorithm produced is robust to noise and also adheres closely to the results achieved by using the atan2 algorithm.

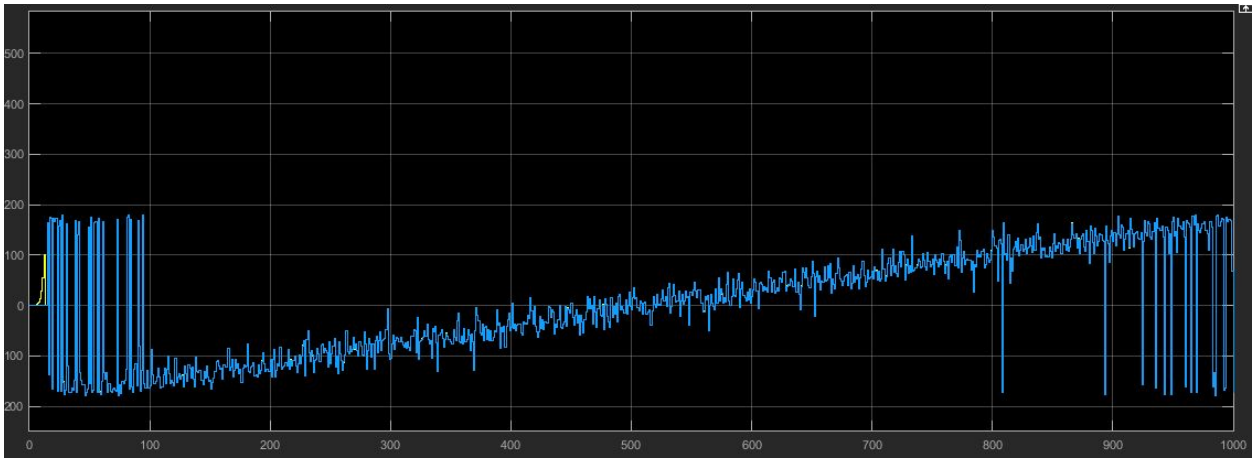


Figure 5: Shows the algorithm values in yellow adhering closely to atan2 values in blue across all degree values of the unit circle

With these results, the the algorithm was considered ready for hardware generation and testing in VHDL.

After the VHDL was generated the generated code was verified in matlab. The ina and inb sources from the first testset in Simulink were recorded and added to text files as inputs for simulation. The testbench recorded outputs every clock cycle for each input and the results were written out to a text file. These outputs were then verified to the Simulink output values that were recorded. These two system output files matched, with the exception of an extra 0 value recorded in the data. This was due to the outputs being 0 at the start and thus the system starting a clock cycle earlier than the data was implemented in Simulink. When this extra value is removed, the data lines up perfectly. Thus our VHDL matches our simulink model perfectly.

An interesting result the VHDL testbench also verified is the delay of our system. If we allow more than 15 clock cycles to run before the input is changed, we can see we get 150 nano second of delay on a 100 MHz clock, or exactly 15 clock cycles of delay from the valid input of our system, to a valid output [Figure 6].

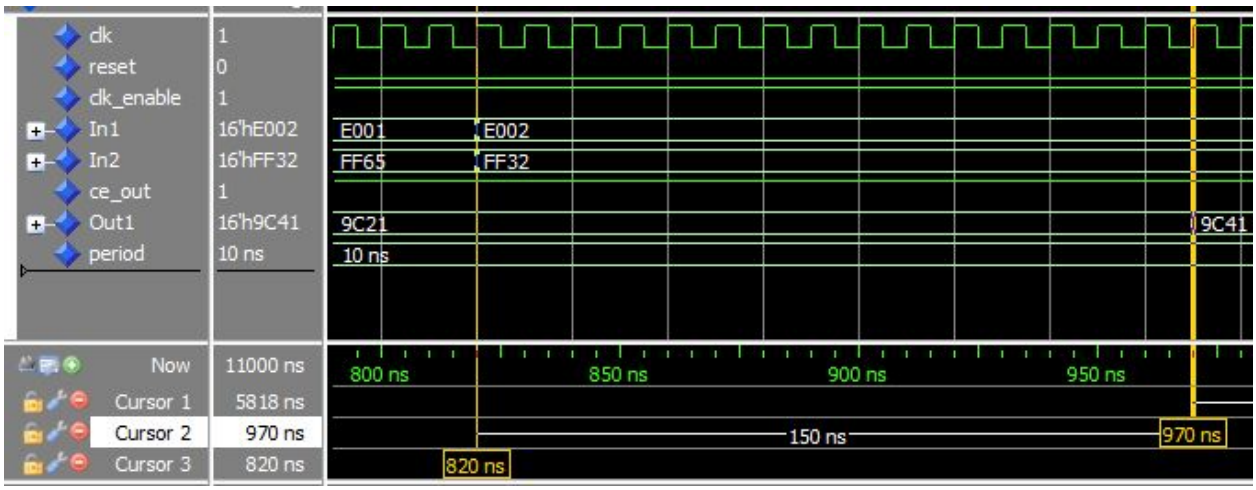


Figure 6: Shows input change at 820 ns simulation time with a new output value at 970 ns later - exactly 15 clock cycles

The VHDL generated from the simulink model was then added into a Vivado project to be targeted onto a Snickerdoodle - a board that houses a Zynq 7000 device. The project included the processing system itself and custom axi busses that are used to interface with the logic block [Figure 7].

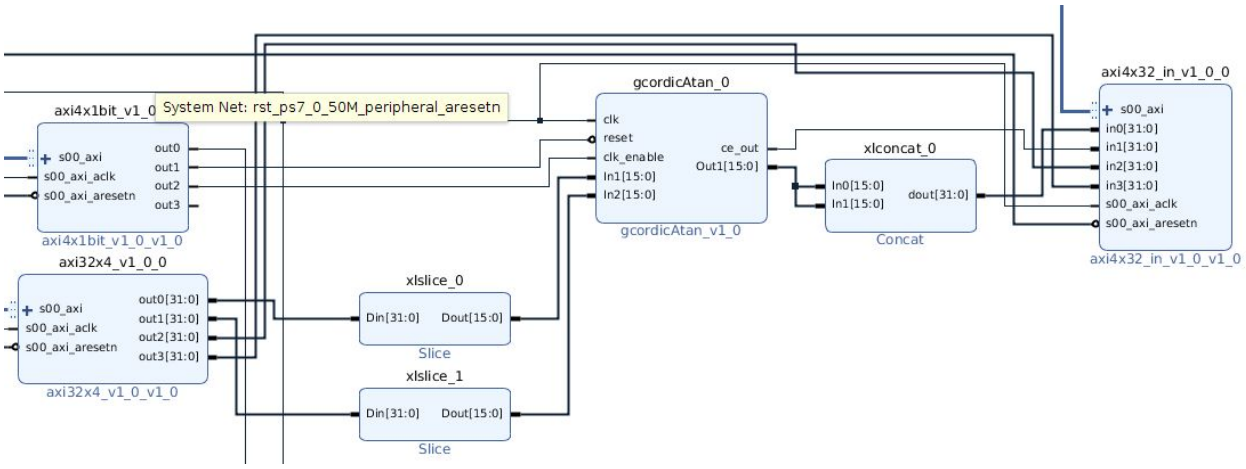


Figure 7: Subsystem of VHDL block design incorporating AXI logic blocks interfacing with the Atan IP

Since Axi register paths are at minimum 32 bits in length, the output of the atan algorithm was concatenated together. This is remembered and handled upon reading the register in the python script later used to evaluate the data.
After synthesis, the cordic arctangent module proved to take up minimal space taking up 529 look up tables (LUTs) in the design [Figure 8.]

Name	Used
design_1_wrapper	1381
design_1_i (design_1)	1381
ps7_0_axi_periph (design_1_ps7_0_axi_periph_0)	559
gcordicAtan_0 (design_1_gcordicAtan_0_0)	529
U0 (design_1_gcordicAtan_0_0_gcordicAtan)	529

Figure 8: Synthesis results showing the gcordicAtan module using 529 LUTs

Once the synthesized code made it onto the board, a python script was written that takes in the same input data used in the VHDL and writes out the output data in the correct format into a text file. The data from here was compared to, and matches, the same data captured on the output of the simulink model and thus the algorithm has been verified.

Cordic Arctangent approximation design summary:

1. The model uses 11 constants or luts specified in Table 1
2. Input A is the Cosine of the desired value normalized to the unit circle
    a. Input values may range from -1 to 1
    b. Input values are in 16 bit fixed point format with 13 fractional bits
3. Input B is the Sine of the desired Value normalized to the unit circle
    a. Input values may range from -1 to 1
    b. Input values are in 16 bit fixed point format with 13 fractional bits
4. Output Values are the approximated arctangent value of a given point on the unit circle
    a. Output values may range from -pi to pi
    b. Out values are 16 bit fixed point format with 13 fractional bits
5. The design includes the following:
    a. Matlab Model
    b. Simulink Model
    c. VHDL Output and testbench
    d. Vivado project targeting a Snickerdoodle (Zynq 7000) Platform
        i. Python Script to debug the above project on platform

In order to replicate this project first clone the repository, run the matlab script, run the simulink model, then create a modelSim project and run the VHDl simulation. From there the system.bin file may be copied over to and programmed on a snickerdoodle device. Copy the Python verification folder over and run the script inside on the snickerdoodle. The output text file from the Python script, and the VHDL may be compared together as well as the simulink output; however, the simulink output is saved in the matlab workspace and will require some manual work to be moved into excel.

Program Versions

Matlab 2018a
Vivado 2018.1
ModelSim 10.7
Python 2.7 (on the Snickerdoodle platform)

Sources

[1] https://github.com/gabrieleara/CordicAtan2
[2]