Phase Detector

Arunkumar Ravichandran and Debosmit Majumder

October 24th, 2017

Question 1: What is the throughput of your Phase Detector? How does that relate to the individual components (FIR, CORDIC, etc.)? How can you make it better?

Answer :

Phase Detector Optimized 1:

Throughput of the Phase Detector was found to be -> 1/(8.42 * 10^-9 * 296) = 0.4 MHz.
Throughput of the CORDIC core is -> 1/(8.42 * 10^-9 * 164) = 0.7 MHz.
Throughput of the Complex FIR is -> 1/(7.55 * 10^-9 * 132) = 1 MHz.

As expected, it can be observed, the time interval in between inputs for the Phase Detector is the sum of the time intervals of CORDIC and Complex FIR -> (164 + 132) = 296 clock cycles.

The throughput can be made better by reducing the time intervals of CORDIC and Complex FIR individually.

Phase Detector Optimized 2:

We pipeline the design and observe that the throughput significantly increases. However, the resource usage increases significantly. After pipelining, the throughput is observed to be -> 1/(8.50 * 10^-9) = 116 MHz. To make the design better we need to reduce the number of bits being occupied by the float and instead of using the costly multiplier we can use the shift add. Changing the data types from float to fixed increases the throughput by a small margin compared to the pipelined design.

Component	Throughput
Complex FIR	1MHz
CORDIC Core	0.7MHz
Phase Detector - baseline	0.4Mhz
Phase Detector - pipelined	116MHz

Table 1: Component and their throughputs

Question 2: These questions all refer to the CORDIC design. Why does the accuracy stop improving after so many iterations? What is the minimal amount of bits required for each variable? Does this depend on the input data? If so, can you characterize the input data to help you restrict the number of required bits? Do different variables require different number of bits? You should use ap_int or ap_fixed types if necessary for required bit width.

Answer: The accuracy stops improving after many iterations because the quadrature value of Cr keeps on decreasing by a factor of 0.5 at each iteration. After a certain point, even though the precision is going to be increased after the decimal, we don't store that in the variable(limited number of bits) making the rest of the iterations moot.

Minimum Number of bits:

Input and Output Data - 13 bits in total, 1 bit for sign and 2 bit for integer.

Temporary In-phase and quadrature-phase Variables - 13 bits in total, 1 bit each for sign and integer.

Temporary Theta - 14 bits in total, 1 bit for sign and 2 bits for integer.

For Sigma (-1/+1) - 2 bits in total, 1 each for sign and integer.

The minimum number of bits depend on the input data. As input data value keeps increasing, the temporary in-phase and quadrature-phase values will increase. **Please refer CORDIC Optimized 2**

The data type of input can be made fixed-point to restrict the number of bits required. As seen from Table 2, we can notice that by using different number of bits for different variables, we reduce the resource usage.

	BRAM	DSP	FF	LUT	Throughput(MHz)
Cordic base design	0	10	3253	4800	0.72MHz
Cordic with every variable having the same number of bits	0	14	1020	804	2.241MHz
Cordic with every variable having optimized number of bits	0	4	929	498	2.241Mhz
Cordic with ternary operators	0	4	929	498	2.241MHz
Cordic with pipelining	0	2	4887	2360	120MHz

Table 2: Different designs and their resource usage

Question 3: What is the effect of using simple operations (add and shift) in the CORDIC as opposed to floating-point multiply and divide?

Answer: As we change the data types to fixed-point, the HLS tool synthesizes the multiply and divide functions into shift and add. Hence, the time required for each operation decreases which increases the throughput. The number of DSP48s required also decreases because multiplication and division operations are replaced with simple shift and add. Hence, resource usage also improves significantly.

Question 4: How does the ternary operator '?' synthesize? Is it useful in this project?

Answer: By changing the code to have a ternary operator there was no increase in performance.

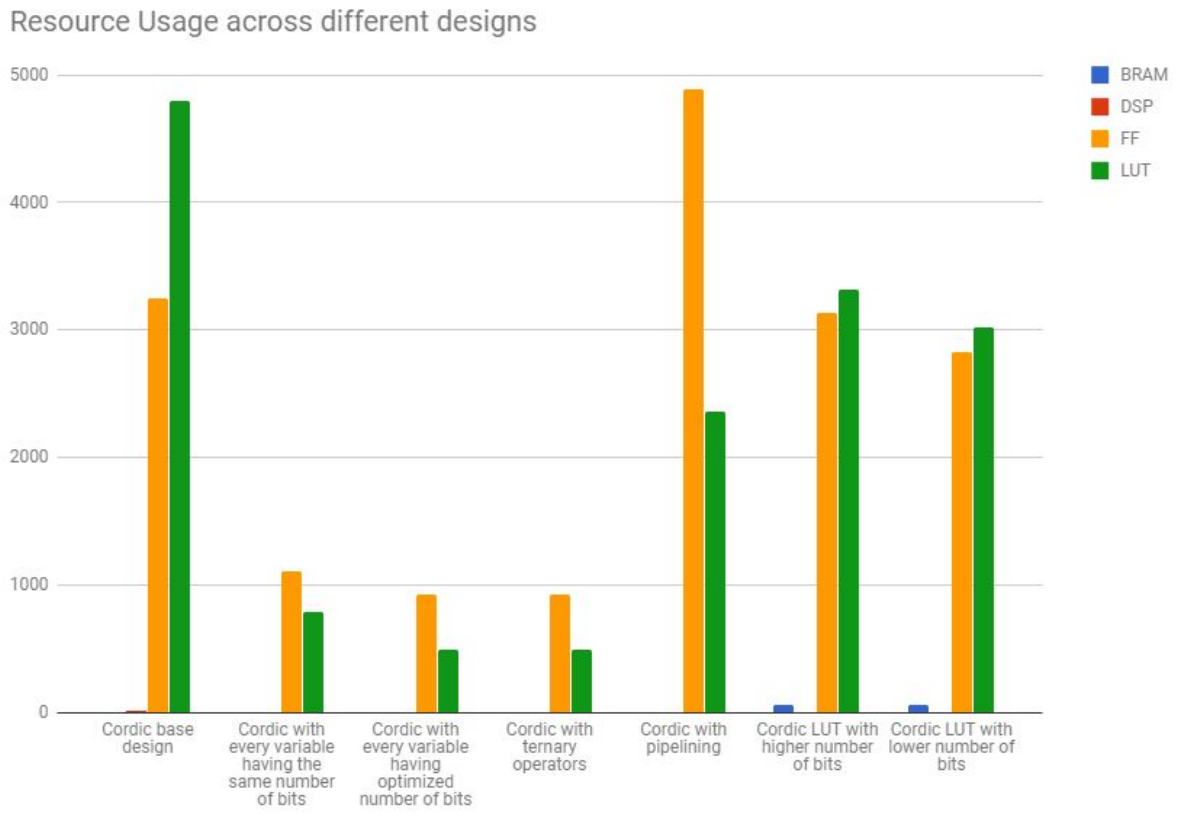
The ternary operators should have been synthesized into multiplexers. But the no of multiplexers remained the same for the base CORDIC design and using ternary operators. So it could mean that the tool has optimization directives to automatically convert the if/else statement into a hardware MUX. This was observed to be not useful in this project. **Please refer CORDIC Optimized 1**

Note:

Cordic Optimized 1 is the design with every variable having the same number of bits and we have used ternary operator. But Ternary operator didn't improve anything in our design.

Cordic Optimized 2 is the design with optimized number of bits for all the variables

Cordic Optimized 3 is the design with pipelining



Question 5: These questions all refer to the LUT-based CORDIC: Summarize the design space exploration that you performed as you modified the data types of the input variables and the LUT entries. In particular, what are the trends with regard to accuracy (measured as error)? How about resources? What about the performance? Is there a relationship between accuracy, resources, and performance? What advantages/disadvantages does the regular CORDIC approach have over an LUT-based approach?

Answer:

1. As the number of bits for the input data, and computed quadrature and imaginary values were reduced, the resource usage reduced and the error value was slightly increasing. The minimum number of bits is 7 in which 2 were used for the non-fractional part. Beyond this, if the number of bits is reduced it caused a RMS error which didn't pass the testbench.

2. Also by using lesser number of LUT entries the error was high compared to the error when we computed with higher number of LUT entries.
The above 2 indicates that as the number of bits or number of entries in the LUT is reduced precision is reducing and the error is increasing.

As seen from the table below these inferences can be made:

1. CORDIC based approach has advantages like very less memory usage.
2. LUT based approach has the advantage of using no DSP because only Look-up operation was being performed; no multiplication or division was done and hence DSP was not needed.
3. LUT based approach will have higher throughput compared to the Cordic baseline implementation.

Design	BRAM	DSP	FF	LUT	Throughput
Cordic Base	0	10	3258	4800	0.72MHz
Cordic LUT with higher number of bits	64	0	3139	3315	22MHz
Cordic LUT with lower number of bits	64	0	2825	3019	22MHz

Table 3: Different designs with their resource usage