Patrick Hanrahan A53204304
Benjamin Hobbs A113171489
Frank Yao-Rung Chang A08362337
WES 237C
10/25/2018

Project 2 : Phase Detector

1. Introduction

Design a phase detector by implementing a complex FIR filter and a COordinate Rotation DIgital Computer (CORDIC) IP core. A CORDIC is an efficient method for calculating trigonometric and hyperbolic functions. CORDIC can do a lot of different functions; we will use it to convert Cartesian coordinates (x, y) to the polar coordinates (r, theta). The goal is to do iterations of estimation and checks to discover the phase of the signal.

2. Directory Names

Folder	Description
fir_top_baseline	Baseline | Floating point | Disabled unroll/pipeline
fir_top_optimized1	Optimized | Fixed point | Enabled unroll/pipeline
cordic_baseline	Baseline | FPU math | Disabled unroll/pipeline
cordic_optimized1	Optimized | Fixed point | Enabled unroll/pipeline
Phase_detector_baseline	Baseline | FPU math | Unroll/pipeline
Phase _detector_optmized1	Optimized | Fixed point | Enabled unroll/pipeline
cordic_LUT	Optimized | Fixed point | Enabled unroll

Method	Throughput	BRAM_18K	DSP48E	FF	LUT
FIR Baseline	0.07MHz	2%	3%	1%	4%
FIR Optimized	2.26MHz	0%	3%	8%	17%
CORDIC Baseline	0.57MHz	0%	4%	2%	7%
CORDIC Optimized	4.57MHz	0%	1%	3%	17%
Phase Detector Baseline	NA	2%	7%	3%	12%
Phase Detector Optimized	NA	0%	5%	14%	40%

Summary Points Based on Results:

1. Phase Detector Baseline and Optimized did not yield a throughput because the size of the array is yet to be known by the function.
2. Optimized version of each method performed better than baseline by leveraging fixed point arithmetic (observe throughput) however traded off increase in resource usage.
3. Given I and Q, each of the two phase detectors computed the radius and phase angle of the input data. The Phase Detector Baseline (floating point arithmetic) computed 1024 samples in 22ms (46KS/s) while the Phase Detector Optimized (fixed point arithmetic) computed 1024 samples in 5ms (204.8KS/s).

3. Phase Detector

- **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?**

  A: The phase detector is dependant on the FIR and CORDIC. The IQ input samples will pass through the FIR, or a set of parallel FIR filters that acts as a matching filter. The X and Y out of the matching filter will go through the CORDIC which will find radius r and theta. The slowest link will determine the throughput of the phase detector since one depends on the output of the other. Improving the slowest link (FIR in our case) will improve the overall phase detector design.

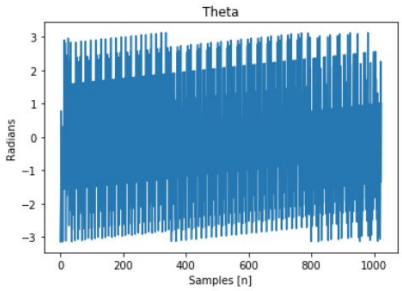


Figure 1. Output of P.D.(theta)

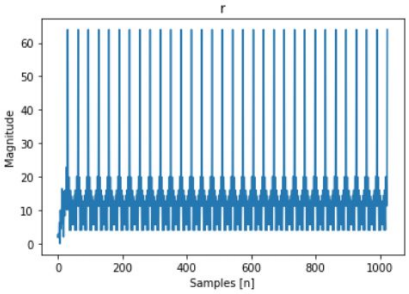
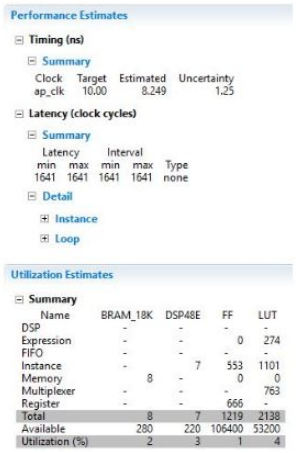


Figure 2. Output of P.D. (r)



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.249	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
1641	1641	1641	1641	none

Detail

Instance

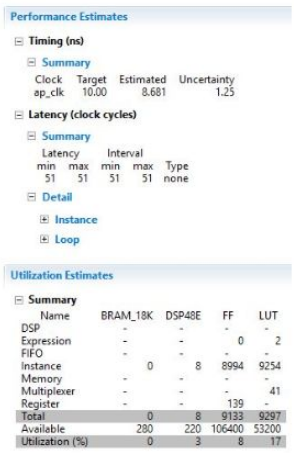
Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	274
FIFO	-	-	-	-
Instance	-	7	553	1101
Memory	8	-	0	0
Multiplexer	-	-	-	763
Register	-	-	666	-
Total	8	7	1219	2138
Available	280	220	106400	53200
Utilization (%)	2	3	1	4

Fig 3. FIR Base



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.681	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
51	51	51	51	none

Detail

Instance

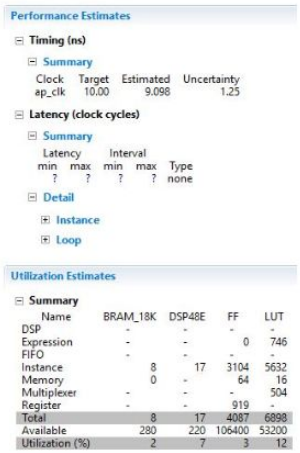
Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	2
FIFO	-	-	-	-
Instance	0	8	8994	9254
Memory	-	-	-	-
Multiplexer	-	-	-	41
Register	-	-	139	-
Total	0	8	9133	9297
Available	280	220	106400	53200
Utilization (%)	0	3	8	17

Fig 4. FIR Optimized



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.098	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
?	?	?	?	none

Detail

Instance

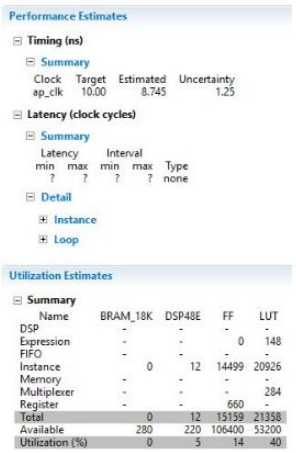
Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	746
FIFO	-	-	-	-
Instance	8	17	3104	5632
Memory	0	-	64	16
Multiplexer	-	-	-	504
Register	-	-	919	-
Total	8	17	4087	6898
Available	280	220	106400	53200
Utilization (%)	2	7	3	12

Fig 5. Phase Detector Base



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.745	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
?	?	?	?	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	148
FIFO	-	-	-	-
Instance	0	12	14499	20926
Memory	-	-	-	-
Multiplexer	-	-	-	284
Register	-	-	660	-
Total	0	12	15159	21358
Available	280	220	106400	53200
Utilization (%)	0	5	14	40

Fig 6. Phase Detector Optimized

4. Cordic

- **Question 2: These questions all refer to the CORDIC design.**
  - **Why does the accuracy stop improving after so many iterations?**

A: The angle accuracy is dependant on the number of bits used to represent the decimal value. As the number of iteration increases to find the phase angle, so does the number of decimal bits needed to capture the change through each iterations.

  - **What is the minimal amount of bits required for each variable?**

A: The minimum number of bits required to represent this input data set is 23 word length bits and 8 integer bits. This means, there are 23 bits in total, 8 of those bits are used to represent signed integer (left of decimal) and 15 bits are used to represent the decimal place (resolution of $1/2^{15}$).

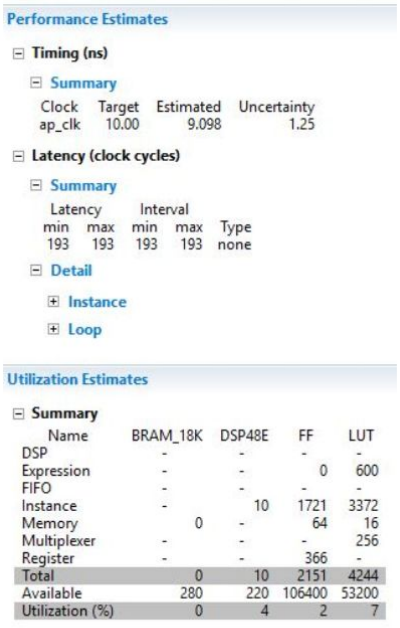
- **Does this depend on the input data? If so, can you characterize the input data to help you restrict the number of required bits?**

  A: Yes, the number of bits used to represent the signed integer of the fixed point is dependant on the input data. The larger the largest value in the data set used, the more bits required to represent the value or else an overflow will occur. This also why the number of bits used to represent the dataset in CORDIC testbench is different from phase detector testbench because the input data range was different.

- **Do different variables require different number of bits?**

  A: Different variables can be represented by the different number of bits however during fixed point arithmetic, the variable with the least number of bits to represent the decimal is upconverted to match the resolution of the variable with most number of bits to represent the decimal to keep precision. In our case, we made a design choice to represent all the variables using fixed point arithmetic by the bit length mentioned above.

- **You should use ap_int or ap_fixed types if necessary for required bit width.**
  A. Our CORDIC was implemented using ap_fixed data types.



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.098	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
193	193	193	193	none

Detail

Instance

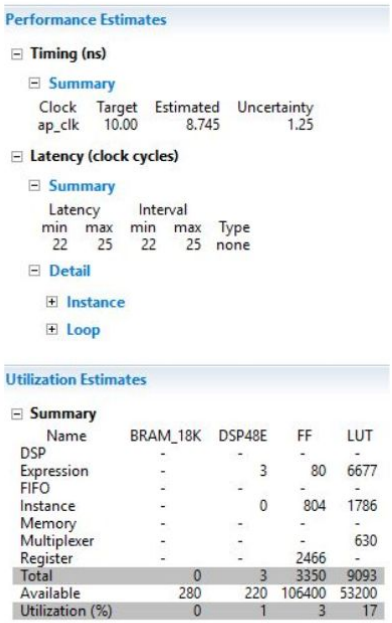
Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	600
FIFO	-	-	-	-
Instance	-	10	1721	3372
Memory	0	-	64	16
Multiplexer	-	-	-	256
Register	-	-	366	-
Total	0	10	2151	4244
Available	280	220	106400	53200
Utilization (%)	0	4	2	7

Fig. 7 Cordic Baseline



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.745	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
22	25	22	25	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	3	80	6677
FIFO	-	-	-	-
Instance	-	0	804	1786
Memory	-	-	-	-
Multiplexer	-	-	-	630
Register	-	-	2466	-
Total	0	3	3350	9093
Available	280	220	106400	53200
Utilization (%)	0	1	3	17

Fig 8. Cordic Optimized

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

A: Adds and shifts can be performed very easily/quickly/efficiently on hardware. Multiplication and division can be done with floating point arithmetic however requires time. CORDIC leverages fixed point arithmetic by converting float numbers to fixed point then performing the mathematical operation. Shift the numbers to the correct (same) base and perform integer multiplication (much faster than floating point arithmetic). Figure 7 and figure 8 shows the throughput of floating point arithmetic implementation (0.57MHz) and fixed point arithmetic (4.57MHz).

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

A: No ternary operator was used in the cordic code. Instead, if/else statements were used to determine which quadrant the data was in and the code manipulates the sign of angle (+/-) accordingly. Normally, a ternary operator works by giving a condition and two possible values to choose from. We were able to get away from this by listing out all of the possible values through the if/else conditions. There could be an impact on performance by the one or two extra line of execution but the if/else is preferred over ternary for readability.

5. Look Up Table (LUT)

- **Question 5: These questions all refer to the LUT-based CORDIC:**

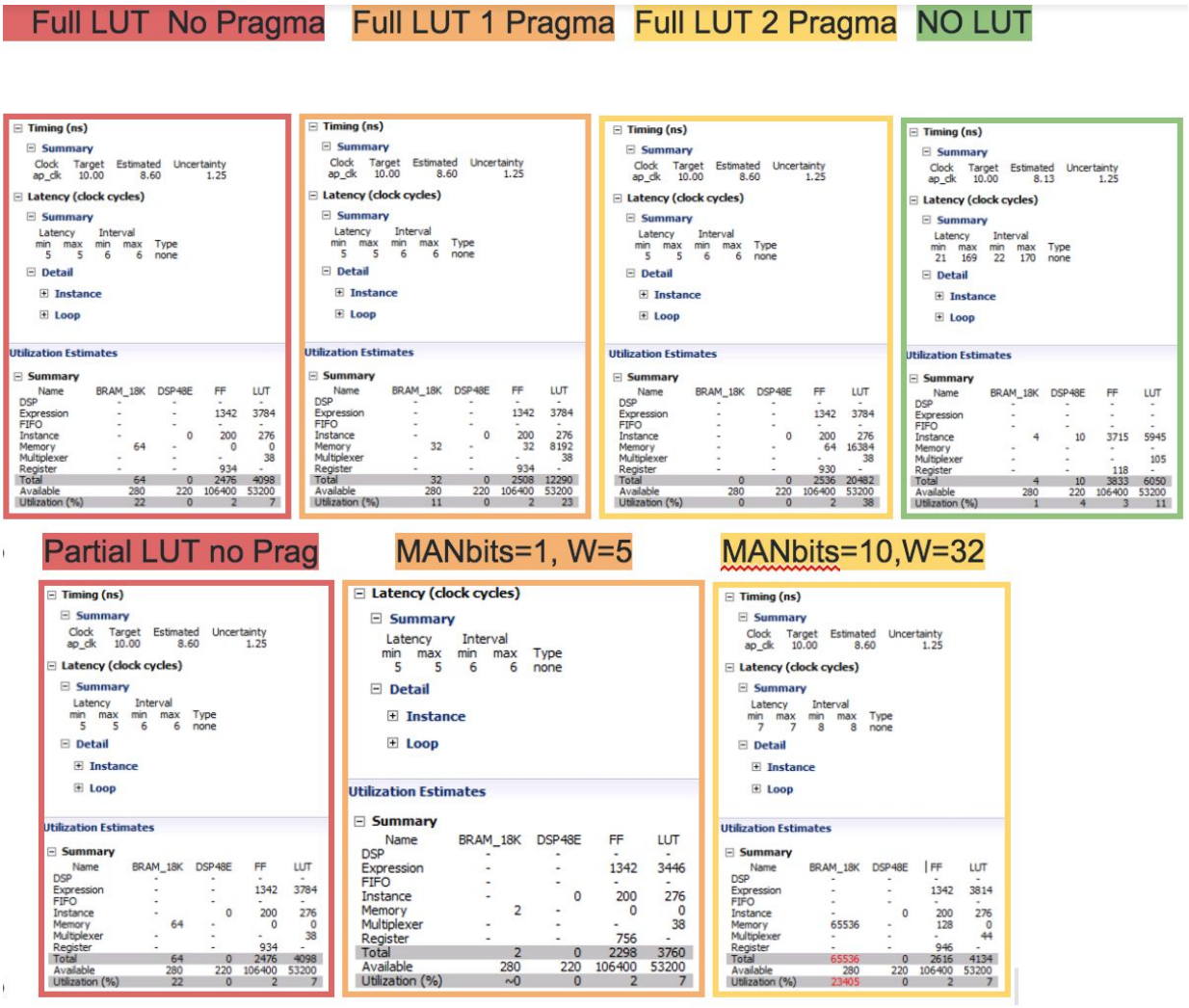


Figure 1. Performance of all implementations.

- **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)?**

  A: Some design space explorations that we explored where implementing different pragmas which prioritized different resources , varied Bit sizes, and also used different variations of the LUT. Now in theory accuracy should remain the same when using “partial” or “full” LUT, but we saw accuracy decrease as you used more extreme versions of the LUT(see Table 1). In terms of BIT size, if you went to a size that was too small, accuracy dramatically decreased. In terms of data type, fixed point can become just as accurate as floating point, however it generally is less accurate.

- **How about resources?**

  A: Utilization of the pragma “#pragma HLS RESOURCE variable=my_LUT_r core=RAM_1P_LUTRAM” makes use of LUT resource instead of BRAMs. Note the increase in LUT and decrease in BRAM as the pragma is implemented for 1 and then for 2 parameters (lut_r & lut_th).

  When the Pragmas are included, the resources used are shifted over to use only 10% BRAM’s and an increase to 23% usage of LUT.

  When LUT is implemented originally (without pragma’s), there is an increase in BRAM’s to 21%, a reduction of DSP to 4%,LUT and FF’s were comparable to before.

  Changing of the BIT sizes decreased resource usage when the BIT size was optimized. However, in the event that too large of a bit size was used, resources dramatically increased.

Floating point versus fixed point used the same amount of resources.

- **What about the performance?**

A: Without the LUT, performance seems drastically reduced with a latency between 22 and 169 seen.

When LUT is implemented originally (without pragma's), Latency goes from 160 without LUT to 5 latency with LUT. When LUT is implemented with pragmas, performance was still 5 latency.

Floating point was slower than fixed point.

- **Is there a relationship between accuracy, resources, and performance?**

A: Using LUT instead of BRAM resources decreases the usage of slower resources on the FPGA and maximizes the usage of fast boolean input type function calculations in LUT. Maximizing resource allocation directly impacts performance since BRAM's run slowly, while LUT's and FF's run quickly. Now when you change the resolution (MAN BITS) and the total size of fixed point representation (W) ,this affects performance inversely. When MAN BITS size increases(see Table 2), you get better error for both R and Theta, however, at the cost of using more BRAMS, which slows performance. When MAN BITS was large, BRAMS were at 100% usage, and total latency decreased. In terms of data type, fixed points are generally faster but has more error when compared to floating type.

- **What advantages/disadvantages does the regular CORDIC approach have over an LUT-based approach?**

  A: The “full” LUT had the most error, while the partial came in second, and the NO LUT had the best error. There is a connection between how you implement the LUT and how much error you get. More LUT generally means more error, but faster processing speeds.

Table 1.	Full LUT 0 PRAGAM	Full LUT 1 PRAGMA	Full LUT 2 PRAGMA	Partial LUT 0 PRAGMA	NO LUT
RMSE(R)	0.01549591	0.01549591	0.01549591	0.000100251	0.00000002
RMSE(THETA)	0.04798147	0.04798147	0.04798147	0.000210534	0.00000000

Table 2.	MANBITS=5, W=32	MAN BITS=1, W=5	MAN BITS=1, W=32	MAN BITS=10, W=32
RMSE(R)	0.000100251	0.000101984500	0.000101984	0.000100065008
RMSE(THETA)	0.000210534	0.157079681754	0.157079681	0.000235032348