CSE 237C Project 2 Report
Aravind Seetharaman and Shrivatsan Rajagopalan

INTRODUCTION

This report presents the design of a phase detector on the Vivado HLS tool. The individual components of the phase detector include a matched filter section (comprising of four FIR filters) followed by a CORDIC which is used to convert the input X and Y (obtained from the output of the matched filter) from cartesian coordinates to polar coordinates. These two segments together make up the 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?

Response:

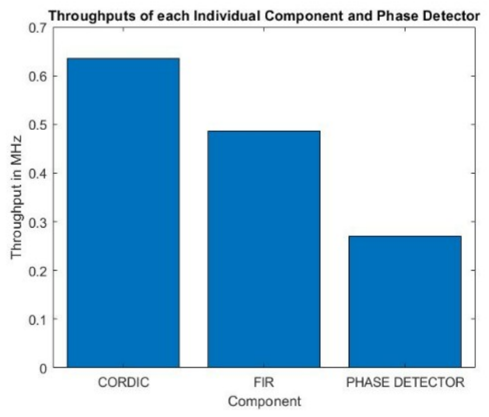
The throughput of the phase detector (**phase detector_optimized 1**) and the throughput of the individual modules (**fir_top_baseline and cordic_baseline**)are listed in the table below.
The total latency of the phase detector module is proportional to the sum of the latencies of the individual modules (FIR and CORDIC).

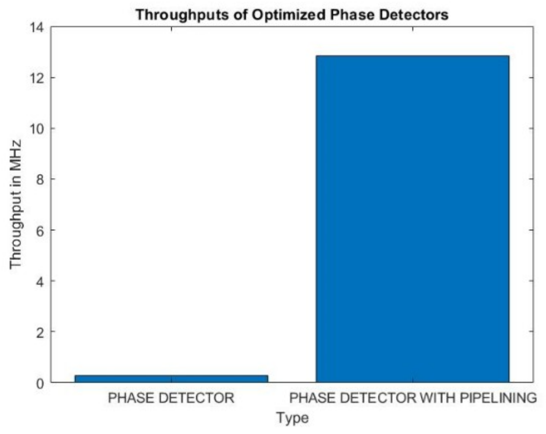
Module	Throughput (in MHz)
FIR	0.487
CORDIC	0.636
Phase Detector	0.2701

The phase detector can be made better by improving the throughputs of the individual modules (FIR and CORDIC). As learnt from the previous exercise with FIR filters, a good way to increase throughput is to introduce pipelining (with an increase in resource cost).

Design	Throughput (MHz)	FFs	LUTs	DSP48Es
Phase Detector optimized 1	0.2701	6305	10442	34
Phase Detector optimized 3 (pipeline)	12.84	62791	68850	268

Pipelining has increased throughput at the cost of resources used as expected. The throughput was calculated using the latency provided in the report file. The latency provided was for all 1024 iterations and was divided by 1024 to get latency of one iteration.





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.

Response:

At each iteration either x or y is multiplied by the corresponding value in Kvalue. As the number of iterations increases we can see that the number stored in Kvalue keeps decreasing with changes in only the least significant decimal digits. As we cannot store beyond a particular number of decimals in our variables this precision is unseen in the final result (as we are forced to round off our values) even if we increase the number of iterations to a very high number.

For optimizing cordic design the bit widths of the individual variables were modified based on their size. 12 bits were assigned for decimal points for all variables as this was found to produce an RMSE that passed the testbench. Three bits were assigned for the integer part of the variables which includes the bit for sign. (**Cordic Optimized 1**)

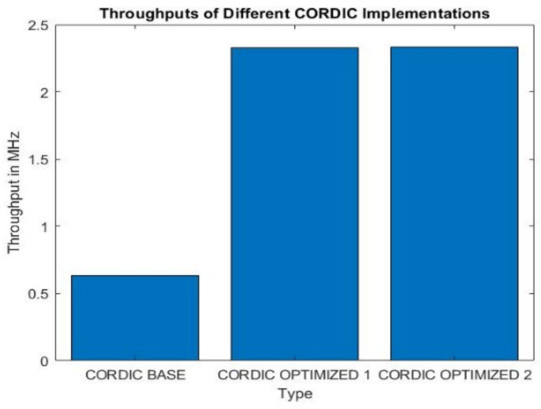
Next, the bit lengths for the data was further lowered to see if accuracy could still be maintained. One decimal point was sacrificed for the variables that store the inputs, outputs and temp variable (the temp variable stores input for swapping). The variable i which goes through the cordic iterations was given 5 bits. (**Cordic Optimized 2**)

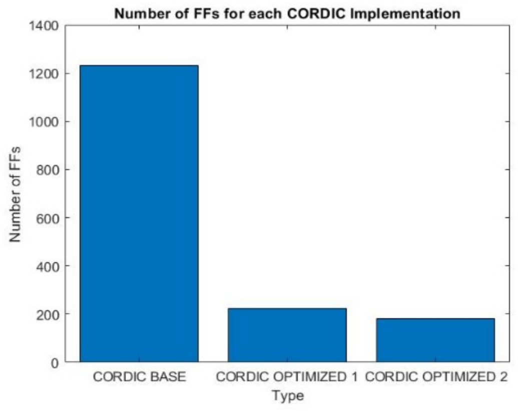
The data types of the variables were modified in the test bench code to check the performance of varying bit width optimizations.

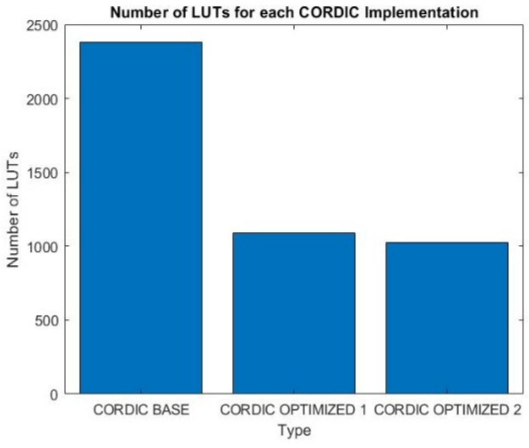
Design	**Throughput (MHz)**	**FFs**	**LUTs**	**DSP48Es**

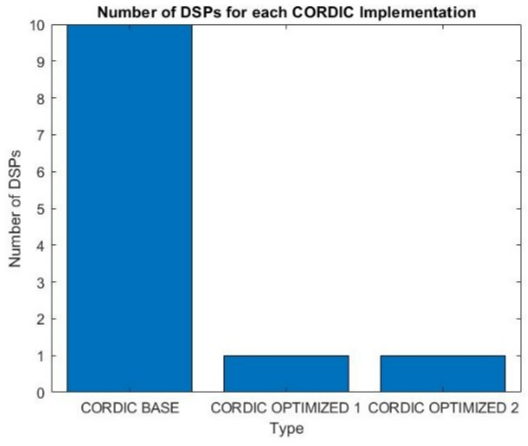
Cordic base	0.636	1230	2383	10
Cordic optimized 1	2.325	223	1087	1
Cordic optimized 2	2.331	181	1025	1

The DSP decrease is due to the fact the multiplications have been replaced with shifts and adds.
Further decrease in FFs and LUTs are observed in cordic optimized 2 with respect to cordic optimized 1 because all variables were given different bit widths based on the data they stored.







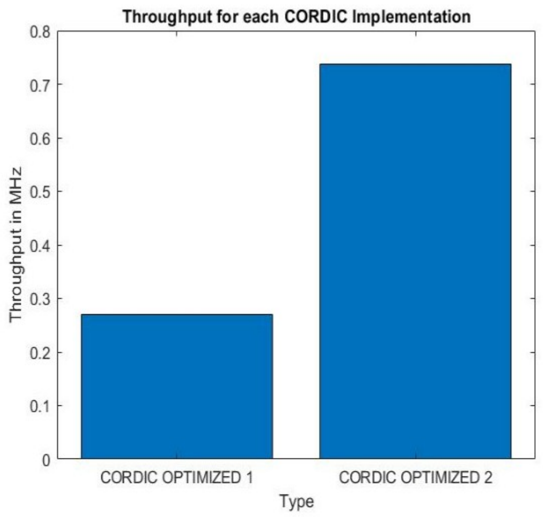


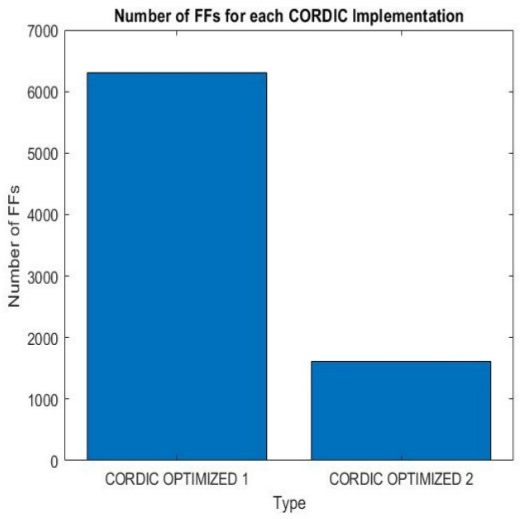
Question 3: What is the effect of using simple operations (add and shift) in the CORDIC as opposed to floating-point multiply and divide?

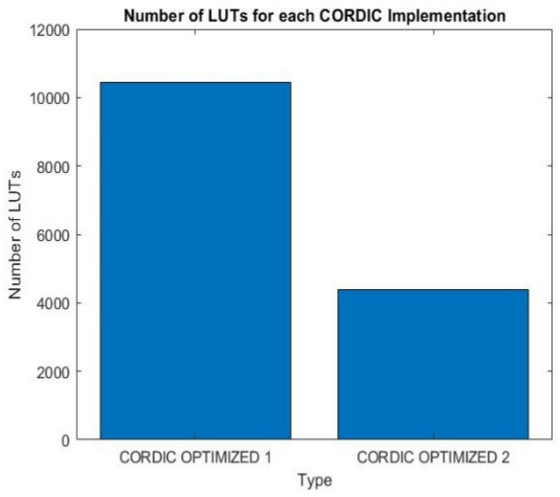
Response:

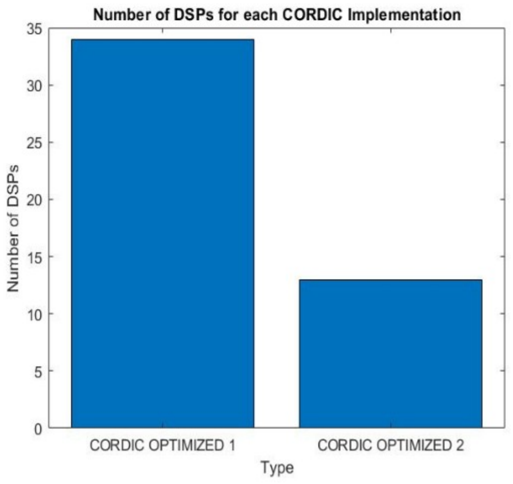
By replacing the floats with fixed point values and replacing the multiplications with shifts and adds should result in a decrease in latency, resources (especially DSPs used for multiplication). By careful selection of bit widths for different data types (as done in Question 2), the above mentioned improvements can be achieved with a negligible loss in accuracy.

Design	Throughput (MHz)	FFs	LUTs	DSP48Es
Phase Detector Optimized 1	0.2701	6305	10442	34
Phase Detector optimized 2	0.7373	1605	4390	13









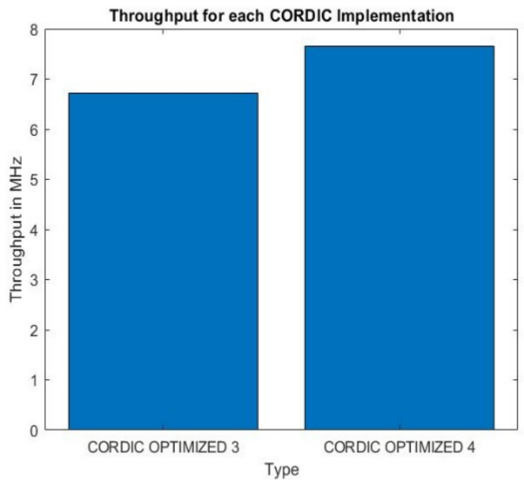
Phase detector_Optimized2 has fixed point variables (for all submodules) and computations in the cordic were done using shifts and add. This has seen an expected drop in FFs, LUTs and DSPs and an increase in throughput.

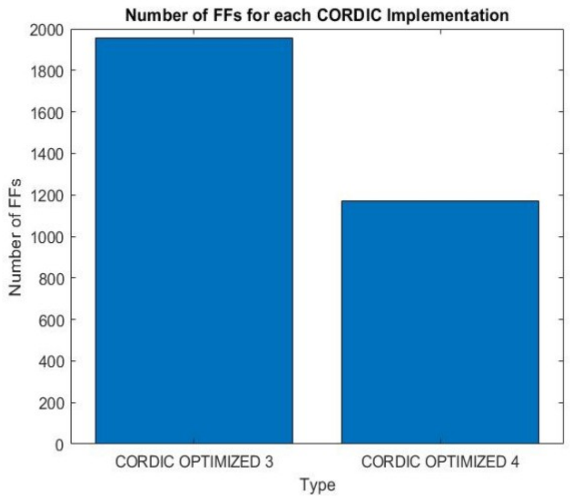
Question 4: How does the ternary operator ‘?’ synthesize? Is it useful in this project?

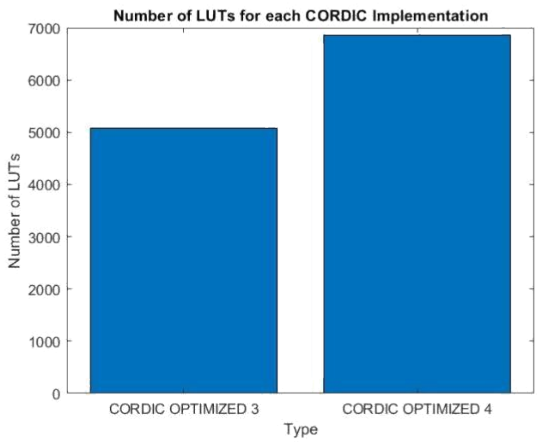
Response:

This step involved replacing a couple of ‘if conditions’ with the ternary operator in the cordic block along with pipelining the design. The idea behind this was to see whether the design with ternary operator would be more suitable to pipelining because of the lack of if condition. The results of the design are shown below. Another design with just pipelining was tried out to see how it performed.

Design	Throughput (MHz)	FFs	LUTs	DSP48Es
Cordic optimized 3 (ternary and pipeline)	6.722	1955	5079	1
Cordic optimized 4 (pipeline)	7.645	1169	6869	1







The use of ternary operator without pipelining yielded comparable latency to the design without the use of ternary operators. Also the design with the use of ternary operator and pipelining (**cordic optimized 3**) produced comparable latencies to the design without use of ternary operator (**cordic optimized 4**).
No great change in throughput could be observed due to the use of ternary operator.

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?

Response:

The LUT based approach for designing a CORDIC was tested using the provided baseline code. The following results were obtained for the partially filled LUT based approach.

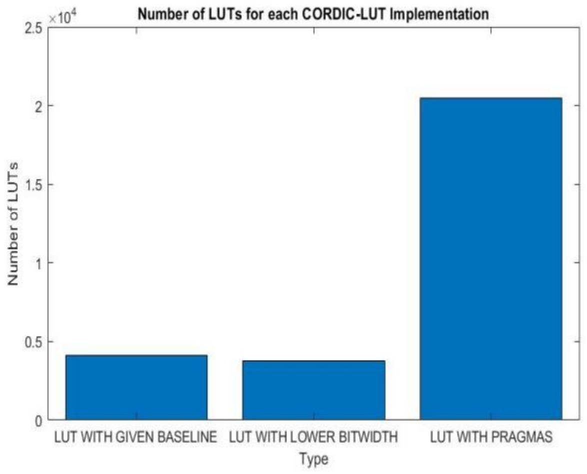
Design	BRAM_18Ks	FFs	LUTs	DSP48Es
Partially filled LUT (no pragma for resource)	64	1134	4098	0
Partially filled LUT (with resource pragma)	0	1194	20482	0

The pragma for resource if uncommented forces the software to use a single port RAM for the my_LUT variable which means reads and writes cannot be done at the same time.

The latency and throughput was found to be the same for both cases and it can be seen that no DSP48Es are used in the LUT based approach. If the single port resource pragma is not enabled, 64 BRAMs which are inherently dual port are used and this meets the deadline with fewer LUTs.
However if the system is forced to only use single port RAMs, then the number of LUTs used increases in order to meet the deadline.

The following table presents the results obtained after running with the LUTs entries filled completely.

Design	BRAM_18Ks	FFs	LUTs	DSP48Es
Partially filled LUT with reduced bit widths	4	950	3772	0



Running the setup with fully filled LUT entries produced high errors in R and theta and did not pass the test bench. Therefore the error threshold was changed to 0.07 and it passed the test bench then.

No change in latency was observed between using fully and partially filled LUTs. Also the usage of the resource pragma in the fully filled LUT case resulted in an increase in the number of LUTs with a drop in BRAM usage. (as observed in the partially filled LUT case)

Comparisons between the fully filled and partially filled LUT setups revealed a peculiar result. While the error in magnitude was more or less comparable between the two scenarios, the error in theta was worse in the fully filled LUT setup when compared to the partially filled LUT setup. From our understanding we were not able to make much sense of this result as we expected the errors to drop when the fully filled LUT setup was used.

The next change was made by varying the number of bits. The value of W (total number of bits) was made 5, with 2 bits for the integer part and 3 bits for the fractional part. **This results in a decrease in number of resources with a slight increase in error. A similar trend was also observed when the bit widths were reduced for the fully filled LUT setup.**

Multiple reports are provided in the code folder for LUT based cordic. Reports 1 and 2 are results obtained from the partially filled LUT without and with pragmas. Reports 3 and 4 are results obtained from the fully filled LUT without and with pragmas. Report 5 is the result of the partially filled LUT with reduced number of bits. The code provided is for the fully filled LUT implementation.