CSE237C Lab2 Report

Hanyang Xu A92068025 hax032

Question 1:

Why does the accuracy stop improving after so many iterations?

Because each iteration decreases the size of rotation by half, with many iterations, the change of angle is close to 0 so the accuracy can not improve.

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

Coeff_t is 2bits, it is used only to represent variable sigma which has only value 1 and -1, therefore a fixed 2-bit signed int is enough.

Data_t, Accu_t requires at least 3 bits for integer and 12 bits for decimal. Accu_t needs 3 bits for integer since it cannot exceed the value of +-pi which need 3 bits too represent as a signed number. Accu_t needs 12 bits is determined based on experiment. Using 11 bits to represent fraction part failed the testbench.

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

Data_t and Accu_t's bitswidth are depend on input data. If input data has integer larger than 3, then the current bitswidth will certainly fail since it cannot represent integer more than 3. Also, if the fraction part of more than 15 digit, then the current representation will also lose precision. Therefore, the input can be characterized by their integer range which determined the integer digit of a fixed length data type. The fraction digit and total digit then can be determined by setting a precision target such as RMSE and running testbench on possible input.

Do different variables require different number of bits?

Yes, as showed above, Coeff_t only requires 2 bits while Data_t and Accu_t requires 15 bits. The bits required is ultimately determined by the range and precision requirement of the data. These requirements should be analyzed carefully before implement the system in HLS.

Question 2:

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

Device/Design	Baseline	Optimize1	Optimize2
FF	1969	134	101
LUT	3609	308	455
DSP48E	24	3	1
Frequency	0.128GHz	0.141GHz	0.135GHz

Baseline uses floating point multiply/divide. Optmize1 uses fixed length number and multiply/divide. Optmize2 uses fixed length number and add/shift. Compare Baseline with Optimize1, we can see that use fixed length number dramatically reduces the resources usage. It is because the reduced bits decrease the number of resources that are configured but not actually used. Compare Optimized1 with Optimize2, we can that simple operation also requires less FF and less DSP useable. This is because a shifter is structurally simpler than a multiplier. There are two more observation from these tests. First, shift operation can not be done on

float data type. Second, use simple hardware does not speed up the hardware design without any HLS optimization.

Question 3: How does the ternary operator '?' synthesize? Is it useful in this project?

? is basically another format of a if/else block. Therefore, it is synthesized as a mux with two input signal and a select signal. It is less than a if/else block since there are identical hardware-wise but "?" operator requires an additional variable sigma and multiply by -1 in order to be useful which is more expensive than simple if/else.

Question 4: 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?

Matrix/Tests	Data_t = Float W = 32 I = 2	Data_t = Float W = 32 I = 2 (Uncommented Optimization)	Data_t = ap_fixed<8,3> W = 7 I = 2 MAN_BITS = 5	Data_t = ap_fixed<6,3> W = 7 I = 2 MAN_BITS = 5
Accuracy	RMSE(R) = 0.017018999904394 RMSE(Theta) = 0.038190785795450	RMSE(R) = 0.016871955245733 RMSE(Theta) = 0.038262836635113	RMSE(R) = 0.016041226685047 RMSE(Theta) = 0.017157571390271	RMSE(R) = 0.054267723113298 RMSE(Theta) = 0.067570336163044
Resources	FF = 1129 LUT = 3445 BRAM_18K = 64	FF = 1191 LUT = 19829 BRAM_18K = 0	FF = 18 LUT = 4145 BRAM_18K = 0	FF = 14 LUT = 3121 BRAM_18K = 0
Performance (Clock Cycle)	8.377	8.464	4.537	4.537

The commented-out HLS pragma a specific library resource (core) is used to implement a variable. Here those two lines specified that RAM_1P_LUTRAM is used to implement the lookup table. Therefore, we can see the LUT usage increase from 3445 to 19829 from above table while block ram usage becomes 0. The performance increases greatly due to LUT access's speed advantage compare to block RAM.

Compare the second test with the third test, we can see that the performance increases and resources useable decreases when fewer bit fix-bit number is used. The accuracy is not affected if the resolution of data_t is higher or equal to the resolution of the LUT. However, if we reduce the resolution of data_t to below the resolution of LUT, like what I did in test 4, then the accuracy will decrease.

In general, the performance will increase, and resources usage will decrease when use fix-bit number without paying performance penalty until the data width used is in computation is smaller than the data width in the LUT. The fix-bit number bit width could still continue to go down depends on error tolerance but the error will definitely start to go up as used bit width go down from that point. The advantage of use regular CORDIC is that it saves a lot of storage space since it only has few coefficients to store compare to LUT-based CORDIC where everything is pre-computed and stored.