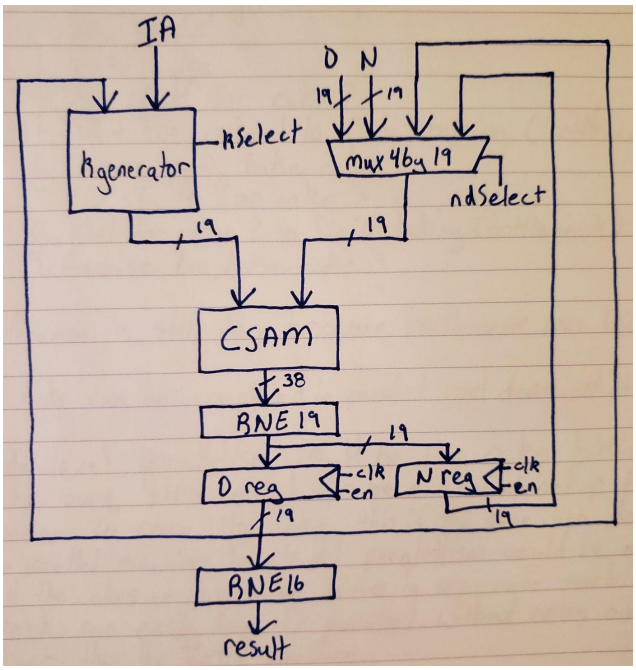
Final Project: Goldschmidt Divider

Tyler Tucker

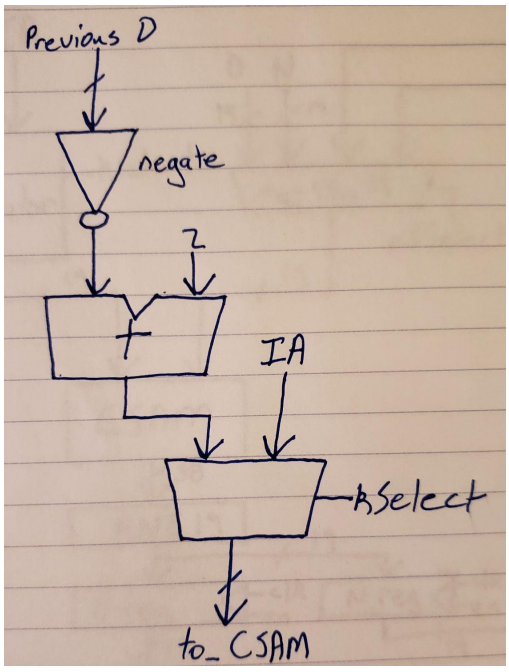
Abstract

In this project we looked at what it takes for a computer to do hardware division. Unlike other mathematical operations, adding, subtracting, multiplying, division is not as straightforward. In order to get semi-usable results we must use the Goldschmidt algorithm to iteratively come to an answer. The cost in both area and time is quite large because of the large Carry Save Array Multiplier used for the main mathematical operation, but these costs are much less than the software costs associated with dividing.



Introduction

In this project we looked at how a computer does division in hardware and why it can be somewhat resource intensive. This design converges quadratically, so for the calculator to get at least 16 bits without error, the design needs to run for at least 4 cycles. Because this design is required to run successively there are registers to store the data between each use. These registers store the data after each run until it is used for the next run. The rest of the design is pretty straightforward, the multiplier does all of the major math operations and the controls in the test bench determine what is being multiplied.



Background

The Goldschmidt algorithm is done in several iterations so there has to be several storage This multiplier normally would use a fairly complicated control FSM to determine what inputs are used for each cycle. Because of the time frame for completing this I was not able to implement a working control unit for my datapath, instead I used the test bench to control all of my control signals during each cycle. The table showing the controls for the test bench are shown below. To the right is the internals of the K generator. I believe my problem with the design is somewhere in the coding of this design, specifically the addition or the selector mux.

Cycle	Multiplicand	Multiplier	K Mux	ND Mux	D Register	N Register
0	IA	D	0	0	1	0
1	IA	N	0	1	0	1
2	K1	IA * D	1	0	1	0
3	K1	IA * N	1	1	0	1
4	K2	K1 * D	1	0	1	0
5	K2	K1 * N	1	1	0	1
6	K3	K2 * D	1	0	1	0
7	K3	K2 * N	1	1	0	1

One of the biggest problems with this lab was not figuring out what code needed to be written but actually writing it. There were several modules that I wrote python scripts to write the tedious code for me, this sped up the production of my code and helped me finish in time.
This also helped with producing valid test vectors, I wrote another script to generate floating point numbers between one and two, calculate a quotient, convert to a binary number, and print it all to two files. One is a binary file and the other is a file that is formatted so that it can be placed directly into the test bench. With all of this set up I was able to move on to the testing phase of my design.

Results

My modeling of a Goldschmidt divider does not appear to work as intended. I believe that there is a problem with the K_n generator but as of writing this report I have not found what that problem is. What I did find is that the algorithm would converge to 0 for a majority of the test cases after 3 or 4 iterations.This is what led me to believe that there was a problem in generating my new K. All of my modules were tested individually before I put them into the datapath so either I have created a K_n generator that is not functioning as the Goldschmidt divider needs it to or my control signals in the test bench are not set up correctly. With that being said I was able to run the divider to get some numbers but these were not anywhere near an actual answer. The area for this design was 1266 square microns with most of the space being taken up by the CSAM. The timing of the design is another area where I had Issues. I believe these issues were related to the whole design not properly functioning, creating a working synthesis that contained a critical path was a problem. Because of this I was unable to get a timing for this design.

N	D	IA	Calculated Q	Actual Q
1.5	1.25	1.125	0	1.2
1.81653779177	1.95653191036	1.504911854181	0	0.92844782246
1.71943975031	1.91740177046	1.48250518487	0	0.89675506552
1.17775773530	1.04481671236	1.24800792812	0	1.48422599077
1.72046789674	1.66200172484	1.87763998763	0	1.03517816560
1.909861148417	1.04776279291	1.75531972650	0	1.82279916916
1.18463349805	1.16177479538	1.07425860561	0	1.01967567445
1.47517157682	1.483774113438	1.86718086158	0	0.99420225994
1.92373220262	1.6205966995	1.42586430765	0	1.18705178357
1.93549562455	1.37825704480	1.86068145552	0	1.404306716115

Conclusion

Although I did not come to the completed project that I wanted to, I still learned a lot of interesting things about how to design a good hardware divider for a computer. The biggest takeaway that I had with this project was to allot more time for debugging the design. The actual coding part of the project did not take too long but getting the actual working design proved troublesome. Like I mentioned above I was able to test the major modules individually and confirm that they worked, but putting them together did not work.