1. Paper
   1. Title Page
   2. Abstract
      1. Make sure to write after writing all the other sections: this part should be written last
   3. Introduction
      1. The intro presents the subject of the report and acquaints the reader with the experiment.
   4. Background
      1. Here is where the fully descriptive narrative including the basic theory behind what is trying to be accomplished in the lab is.
      2. Reference anyone’s words you take, or paraphrase
   5. Results
   6. Conclusion
   7. References
   8. Appendices

High Speed Computer Architecture Final Project

Caleb Cunningham

4/30/2021

Abstract

[one sentince summarizing each section]

Introduction

[introduction to report, write second to last]

Background

Initial understanding of this project starts with how division works in computer architecture. When multiplying in binary, the simple solution is to work with long multiplication, where we shift the AND results and add them together, long division does not convert to physical structures nearly as well.

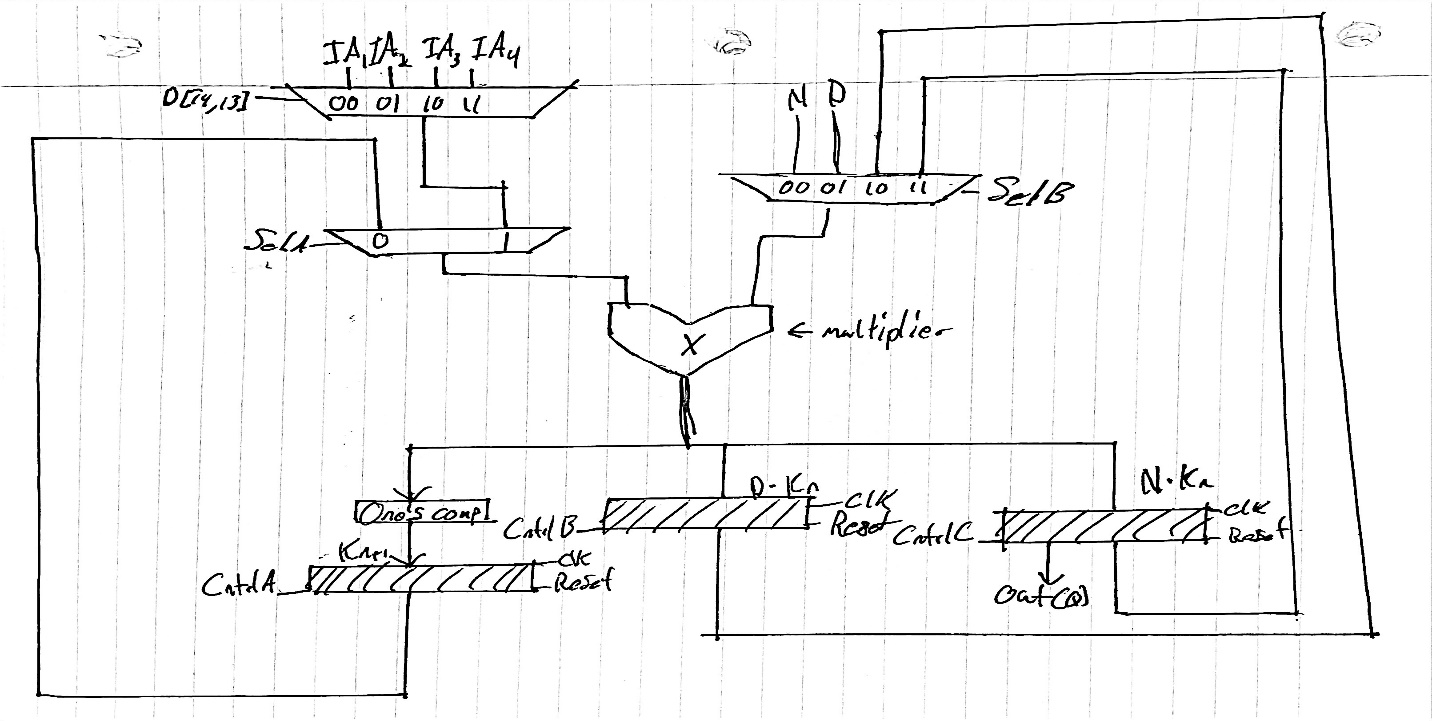
This made physical devices to divide very unattractive and software options were implemented to complete division until the Newton-Raphan Architecture was developed, with the founding principal of multiplying the numerator (N) by 1 over the denominator (D), rather than dividing. This would be completed by substituting an Initial Approximation (IA) for D, and solving the problem by iteratively multiplying IA by D, and finding the next iterations approximation by subtracting the new D value from 2. After iterating a couple of times, the IA value should represent , which is then multiplied by N to finish the operation.

This was a better solution, but it still wasn’t used because of the time consumed by the additional multiply at the end, and the pipelining required to keep N. This was why the Goldschmidt’s Architecture was developed, by using Binomial theory to establish that the answer could be found by multiplying by which could be approximated with . To find the same method for finding the previous IA is used, to find the average point in the range (a , b), . To find , Subtract 2 by D\* and add the ULP. The Architecture is implemented with the following steps:

1. Get our initial value, and store it in K0.
2. Multiply N by K0, and store the result.
3. Multiply D by K0, which is r0, and store the result.
4. Find value K1 for the next iteration by subtracting r0 from 2.
5. Repeat 1-3 until you reach your desired accuracy.

Then, after multiplying N by , N will approximate the quotient, with the accuracy depending on the number of iterations.

In this project, the requirements listed that 4 IA values be used. The chosen ranges were [1,1.25), [1.25,1.5), [1.5,1.75), [1.75,2) with respective IA values of 0.9, 0.73, 0.62, 0.54. The chosen iterations for this project was 5, as in the test simulation, no given values of N and D in the given range needed more than 5 iterations to reach accuracy to 16 bits.

To complete this architecture, a multiplier, a register, and two types of muxes are required. 

The Block Diagram starts with a multiplier as the centerpiece, made from the supplied RCA from the class, being fed into by two muxes, One picks between the IA and , whereas the other mux picks between the numerator, the denominator, , and . The results of the multiplier are stored in either the register, the , or it feeds into the one’s compliment matrix, which finds , and is stored into the register. Finally, the IA is picked from the 14th and 13th bits from D.

Results

To test all of the parts for this, since there was no physical design, all parts where tested in a test bench in modelsim to test all of our base conditions for each part to make sure they were working correctly, below are tables of tested values for each part.

|  |  |  |  |
| --- | --- | --- | --- |
| Mux2 | Input A | Input S | Output D |
| Test1 | $random | 0 | Same random as in A |
| Test2 | $random | 1 | Same random as in A |

|  |  |  |  |
| --- | --- | --- | --- |
| Mux4 | Input A | Input S | Output D |
| Test1 | $random | 00 | Same random as in A |
| Test2 | $random | 01 | Same random as in A |
| Test3 | $random | 10 | Same random as in A |
| Test4 | $random | 11 | Same random as in A |

|  |  |  |  |
| --- | --- | --- | --- |
| 16 bit Multiplier | Input A | Input S | Output D |
| Test1 | $random | 00 | Same random as in A |
| Test2 | $random | 01 | Same random as in A |
| Test3 | $random | 10 | Same random as in A |
| Test4 | $random | 11 | Same random as in A |

1. How all parts were tested?
2. Graphs of parts
3. Final Accuracy
4. Extra Credit opportunities that were implemented

Conclusions

[wrap up what we’d accomplished through this project]

References

[probably reference his lectures and the shared code]

Appendices

?????

**Spare already written aprts**

To first start this process, it’s important to know the algorithm that we are using to complete this entire project. We are using Goldschmidt’s iteration to use a multiplier to iteratively solve a division problem. In this project we were to work in 16 bits, 15 being fractional, so we are dividing floating point numbers [1,2).

Goldschmidt’s iteration works like this. Instead of creating a new form of architecture for division, we multiply by the inverse of the denominator.

However, if when this is taken to a formula as in the Newton-Raphan architecture you haft to multiply the final approximation for **1/d** by N again. Instead with the Goldschmidt’s method, we use our approximate values, and we multiply both top and bottom, until we reach our required accuracy.

Where K0 is our approximate value, and . We step through the iteration in the following steps:

1. Get our initial value, and store it in K0.
2. Multiply N by K0, and store the result.
3. Multiply D by K0, which is r0, and store the result.
4. Find value K1 for the next iteration by subtracting r0 from 2.
5. Repeat 1-3 until you reach your desired accuracy.

Next part to understand our operation is how we select for our initial value. To pick, we simply take the weighted average of our two endpoints, which for our project using floating point numbers, is 1 and 2.

Which when we take our values of 1 and 2,

Which is the value we are going to use for the rest of our operation. The only remaining question is how many iterations to go to reach our accuracy of 15 fractional bits. To find that out we . . .