Design of reciprocal unit based on the Newton-Raphson approximation*

Anders Torp (s021884)
Rasmus Winther-Almstrup (s021957)
Michael Boesen (s022035)
Department of Informatics and Mathematical Modelling
Technical University of Denmark

Abstract

A design of a reciprocal unit based on Newton-Raphson approximation is described and implemented. We present two different designs for single precisions where one of them is extremely fast but the trade-off is an increase in area. The solution behind the fast design is that the design is fully redundant.

1 Introduction

Designing a fast division unit is always a challenging objective when designing processors. Division is one of the most important calculations i 3-d graphic and because today's graphic processors need to be faster and faster to handle smooth 3-d graphic the division units also need to optimized for speed.

Division units can be implementet either by digit recurrence or iterative approximation. The purpose of the work behind this technical report is to describe the implementation of a fast reciprocal unit using the Newton-Raphson (NR) method for reciprocal approximation. The reciprocal value $\frac{1}{d}$ can be used to make the division $\frac{x}{d}$, by multiplying x and $\frac{1}{d}$. The Newton-Raphson method uses an initial guess in order to compute the result. We will go into detail on how big the initial table should be if we want to get a specific precision of the result within a certain timeframe. We also introduce a carry-propagate multiplier and later on a carry-save version that is needed by the Newton-Raphson method. The two multipliers are then synthesized in Synopsys which allow for comparison of speed and area use. We do not consider power in this project.

We will design a reciprocal unit that is able to handle single-precision (24 bit) floating point numbers. The reason we want to implement the single precision reciprocal unit is because of the reduced time it take to compute the result. This is useful in divisions where the result is more error tolerant and speed is crucial, eg. in graphics cards.

Two methods to find the reciprocal are the Newton-Raphson method and the multiplicative normalization method. This work deals with the Newton-Raphson method that is introduced in the following section. The rest of the report is structured in the following way. Section 3 looks at the architecture of the lookup table and the overall architecture of the division unit. Section 4 describes how the different units are implemented. Section 5 present the results obtained under synthesis in Synopsys and some possible improvements to the architecture. Finally section 6 summarizes the most important conclusions obtained during this work.

2 Algorithm

The Newton-Raphson (NR) method described as 1 can be used to obtain the root of the function $f(x)$,

$$x[j+1] = x[j] - \frac{f(x[j])}{f'(x[j])} \tag{1}$$

where $x[j]$, $x[j+1]$ are approximations to $f(x) = 0$. It generally applies that if $x[j]$ is an approximation, then $x[j+1]$ is a better approximation.

To find the reciprocal we use the function $f(R) = d - \frac{1}{R}$, whose zero is $\frac{1}{d}$. When inserted in the NR method we get the approximation [2]

$$R[j+1] = R[j](2 - R[j]d) \tag{2}$$

If we denote the error of iteration j $error(j)$, then the error of iteration $j+1$ is $error(j)^2$. It can however occur that the approximation does not converge to zero. Convergence can be ensured if the initial approximation is chosen carefully, and the NR method will eventually produce a correct

*The work behind this technical report is carried out as a special 3-week course together with supervisor Alberto Nannarelli at the Informatics and Mathematical Modelling, Technical University of Denmark.

2-x the normal way:	Invert + 1
10.00000000000000000000000	10.00000000000000000000000
-00.11010101101011100011100	-00.11010101101011100011011
	+1
01.00101010010100011100100	01.00101010010100011100100

Figure 1. How to perform the substraction

result. The time taken by the NR method to ensure a certain accuracy depends on the initial approximation. A common way to ensure certain accuracy in a predefined number of iterations is to use a table which holds initial approximations $R[0]$ that corresponds to values of $\frac{1}{d}$. This can be ensured if we know the precision of the initial guess. We can then by the above definition of the error calculate how many iterations it takes the NR method to give a result that correponds to the chosen precision (single/double precision). The matter of converting the result to a double precision floating point from a single precision, is therefore reduced to a decision on how many iterations the algorithm must run and the bitwidth of the signals and multipliers. This make the decision on the size of the table crucial in our design.

3 Architecture

In section 2 the Newton-Raphson method we are using was written as $R[j+1] = R[j](2 - R[j]d)$. This requires two multiplications and a subtraction. The multipliers are necessary, but we can eliminate the need for a subtraction unit, as we will show in the following. The subtraction unit is needed because we need to subtract $R[j]d$ from 2. We know that $R[j]$ is an approximation of $\frac{1}{d}$ and because of this $R[j]d \rightarrow 1$ for j $\rightarrow \infty$. And because the initial approximation is somewhat close to $\frac{1}{d}$ we can infer that roughly $R[j]d \approx 1$ and hereby always smaller than 2. Because we use 2's complement this allows us to use an array of inverters to produce the correct result of the subtraction. An example of this can be found in figure 1. The system then consists of two multipliers, an array of inverters and the table containing the initial values of $R[0]$ (the extra +1 in the figure can be handled in the second multiplier). The multipliers of the system will be described in the following subsections. A sketch of the system can be found in figure 2. This figure shows an iterative approach that uses the same two multipliers in all steps.

3.1 Design considerations

We will implement two versions with each of the two multipliers - an iterative and a non-iterative unrolled pipelined version. Both versions will aim for single precision only but could easily be developed to carry out double precision calculations. This could either be done by

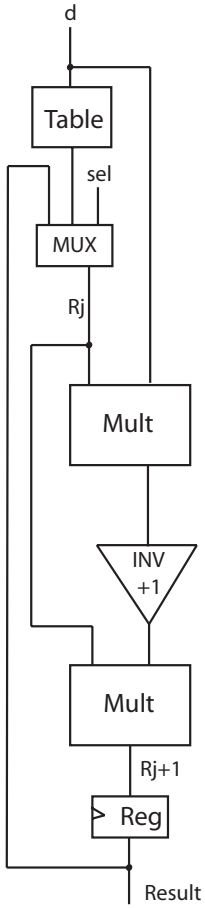

Figure 2. The iterative version

adding two more stages (however the area will increase) or by adding a loop (this will degrade the gain of throughput from the pipeline). In this section we will cover all the decisions necessary to make in order to successfully build the Newton-Raphson reciprocal unit. We have two important decisions to make:

1. The size of the lookup table
2. The architecture of the reciprocal unit

3.2 Table with initial values

Several considerations has to be made when designing a lookup table with the initial approximations. The accuracy of the table depends on the number of inputs/outputs of the table. But the size of the table is not neglectable if we want the initial guess to be as accurate as possible. An extreme case is the table of size 1 ie. only a single initial value. The other extreme is a table that covers all possible combinations of the input mantissa, ie. in single precision

Iteration	Tests	Errors	Error%
0	2^{23}	8226499	98%
1	2^{23}	663172	8%
2	2^{23}	1	$\sim$0%
3	2^{23}	1	$\sim 0\%$

Table 1. Number of errors per iteration

the table should take 24 bits as input and create a 24 bit output. This would make the Newton Raphson approximation useless when considering single precision floats, whereas it would only take two iterations to create a double precision result. The size of this table however makes it infeasible to implement. The table size therefore have to be a compromise of the above mentioned extremes. We have in our design experimented with a 4 input / 4 output and a 5 input / 6 output table. We have chosen to use the latter table. The values of the table is derived from [3] (the values used by the 4/4 were created by us). We chose the 5/6 table as it give an initial 6 bit precisions. From this precision we can obtain a single precision result in 2 iterations and a double precision result in 4 iterations.

Since we have chosen to use a 5 bit in (meaning an entry size of 2^5), 6 bit out table as described in [3], we theoretically only need 2 iterations of the Newton-Raphson algorithm to get the required precision. Firstly we get the initial approximation, which gives us 6 bit of precision. The next iteration gives us 12 bits of precision and the last iterations give us the required 24 bits of precision (the length of a IEEE 754 single precision mantissa). However, we made a C-program to test if this was correct, and the results from this C-program is shown in table 1. The rounding were implemented as truncation. These results justify the use of only 2 stages/iterations in our design.

3.3 The decision of the architecture for the Newton-Raphson unit

To decide on the architecture we propose two alternatives, both shown in figure 3.a and b, note that the register in the middle is a pipeline register. Before we decide on which of the architectures is the best, concerning speed and area, it would be interesting to try and predict which one is the best. This is done by analyzing the main unit in the system, which are the multipliers. We have therefore used Synopsys and synthesized the three types of multipliers (The library used was STM library of standard cell 90 nm).

- **normal** consist of two inputs in normal form (two's complement) and the result is also in normal form.
- **cs/normal** consist of one operand in normal form and one in redundant form. This means that this multiplier have 3 inputs and the result is redundant.

Multiplier	Area	Critical path	Max. clk freq.
normal	37812	2.47 ns	405 MHz
cs/normal	57454	1.08 ns	926 MHz
2cs	115753	1.34 ns	747 MHz

Table 2. Synopsys synthesis data for the multipliers

- **2cs** has both operands in redundant form. This means 4 inputs and the result is redundant.

The results can be viewed in table 2. As observed in table 2 the normal multiplier is slow compared to the two other multipliers. When going from the normal multiplier to the **cs/normal** we get a 128% speed increase from a 51% area increase, which we think is actually quite good. When we take a look at the **2cs** mult, we can see that it is performing rather poorly compared to the other two. Compared to the normal mult we get a 84% increase in speed but it comes at the cost of 206% area increase, therefore we would only use this multiplier if speed is of the essense. From this it looks like an alternative with two **cs/normal** would be the best, however at this time we have chosen to implement an alternative consisting of **cs/normal** and a **2cs** multiplier, in this way we also save a CPA, and this might give us a better result. We have chosen to synthesize both, and the results are described in section 5.1.

4 Implementation

In this section we will describe the details concerning the implementation of the Newton-Raphson unit. We will design 2 different systems. One consisting of a **normal** multiplier and a **cs/normal** multiplier, and a fully redundant system consisting of one **cs/normal** and a **2cs** multiplier (Figure 3).

Before we begin describing the two systems, we will take a short look at how the two different multipliers are implemented.

4.1 Carry-propagate multiplier

We are going to use a carry-propagate multiplier (CPM) in our first implementation of the NR method. The CPM is what we would call the conventional multiplier. It is quite easy to implement, as we can just use the built in multiplier in VHDL to generate the CPM. The multiplier have two inputs and one output.

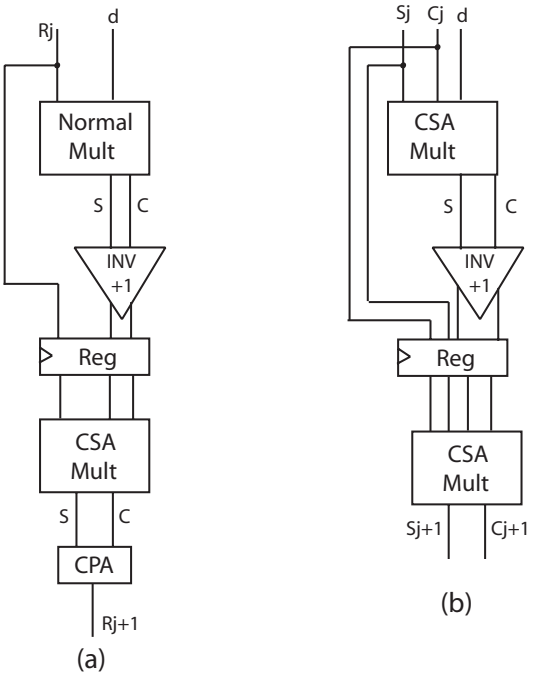

Figure 3. Two alternatives for the CSA multiplier Newton-Raphson architecture.

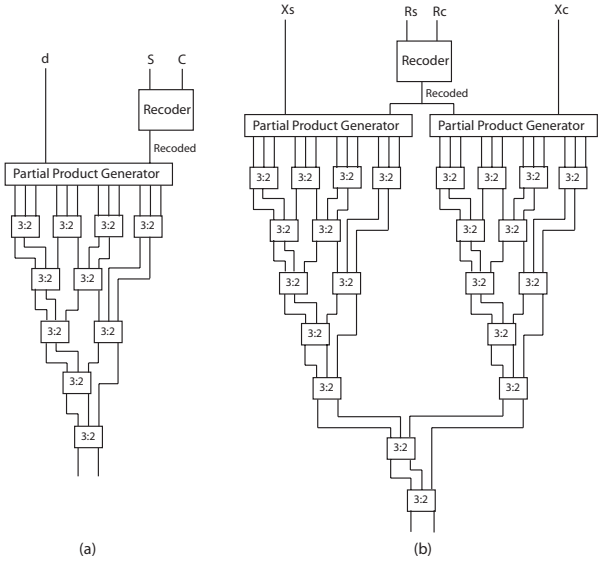

Figure 4. The two CSM

4.2 Carry-save multiplier

We have previously described a carry-propagate multiplier to be used in our system. In this section we will look into a carry-save multiplier (CSM). The carry-save multiplier does not propagate the carry which results in a smaller critical path. The downside however is that the result is given as a partial sum and a carry and we have an increase in area. The smaller critical path can benefit our design since the Newton-Raphson method runs more than one iteration. The partial sum and carry can then be added together using a carry-propagate adder (CPA) at the end in order to get the final result. The CSM consists of multiple generators, an adder tree and rounding at the end.

The CSM approach actually requires two different multipliers as shown in figure 4.a and b. The first multiplier (figure 4.a) takes a standard input and a carry/sum input. The second (figure 4.b) takes two carry/sum inputs. Obviously the second multiplier takes up much more area and because of the extra 4:2 CSA at the end we also expect it to be slower.

The multipliers are implemented as radix-4 and this requires recoding. We have used the same recoding as in [1]. We are recoding $R[j]$ in both multipliers but the double carry/sum multiplier requires a little tweaking: The multiplication in the double carry/sum multiplier is $R[j+1] = R[j]x$, where $x = (2 - R[j]d)$. Since both $R[j]$ and x are in carry/sum form we get the multiplication $R[j+1] = (R[j]_s + R[j]_c)(x_s + x_c)$. We have reduced this to ease the computation and the result is as following (note that we call $R[j] = r$ here): $R[j+1] = (r_s + r_c)x_s + (r_s + r_c)x_c$. This results in using the double amount of multiple generators but relieves us from adding $x_s + x_c$ which can be a time consuming operation. $(r_s + r_c)$ goes into the recoder and thus we do not need to add them in order to do the multiplication.

4.3 Normal multiplier and one cs-normal

The implementation of the unrolled version with the normal multiplier can be viewed in figure 5. The SP_unit of the version with the normal multiplier, looks exactly like figure 3a. So one SP_unit consists of two multipliers and one inverter. When an input d is provided, the table finds the appropriate approximation of the mantissa and sends this to the first SP_unit, here the first iteration of the Newton-Raphson algorithm is carried out. The result is then saved in a register before it proceeds in the next clock-cycle to the last stage (or loops around if it is the iterative version), where the final mantissa is done and ready.

4.4 Fully redundant system

The version with the two CSA-multipliers is exactly the same as the normal multiplier except for the architecture of the SP_unit (figure 5). The SP_unit can be viewed in figure 3b. Therefore we choose not to go into more details about the CSA-multiplier version since they are exactly the same except for the architecture.

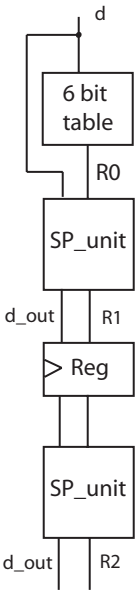

Figure 5. Single precision version unrolled

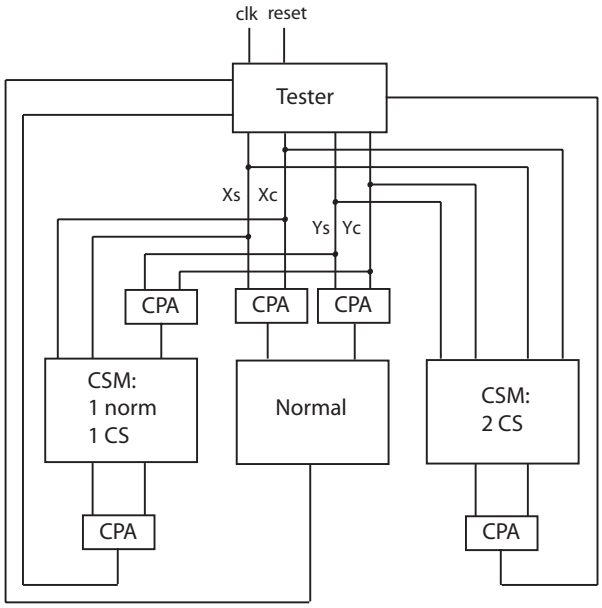

Figure 6. Testing of our multipliers

5 Test and results

Testing the system thoroughly proved to be a quite difficult task. We tested the units individually with a number of random test vectors as well as some thoughtful ones. We even made a self-testing facility by using a C-program to provide the correct results which the simulation then could compare with the result that our divider got. The main unit in our design are the multipliers, so we concentrated heavily on a thorough test of these - which indeed proved to be a very good idea.

5.1 Testing the multipliers

We tested the multipliers by setting up a testbench as described in figure 6. The testbench consist of a "tester" unit which provides the multipliers with some test vectors and examines them at the end, and our three kinds of multipliers. The idea in this is that the normal multiplier, which just uses the built in multiplier is expected to always provide the correct result and thereby testing the CSM multipliers.

5.2 Results

In this section we will present the results obtained from synthesizing our two alternatives in Synopsys. As described in the report we have decided to examine two kinds of architectures. The details for the two architectures can be seen in table 3. As observed (in table 3) alternative nr. 1 is very slow. This is because it has the small slow normal multiplier (see table 2). It also shows that the area increase when we go from the iterative to the unrolled version almost doubles, but the speed only rises approximately 67.7%. But the throughput is of courses increased. Therefore we would not recommend alternative 1, primarily because of the very slow multiplier.

Alternative 2 seems to be the best. From the iterative version we get a 94% speed increase from a 70% area increase. The unrolled version yields a 56% speed increase and an 88% area increase compared to the alternative 1 unrolled, which actually is not that impressive.

5.3 Improvements

During our work with the CSA multiplier we have learned and identified a few things which we could have done better or which could save area and speed. The critical path for the fully redundant version is through the recoder, the partiel product generator and the adder tree in the second multiplier. Because the result from the recoder in the second multiplier is always the same as for the first multiplier, the recoder could be removed. This means that in stead of having two recoder in each stage, we only need one. This

Architecture	Area [μm^2]	Crit. path [ns]	Clk. freq
Alt. 1 iterative	96279	3.93	254 MHz
Alt. 1 unrolled	177146	2.66	375 MHz
Alt. 2 iterative	164019	2.02	495 MHz
Alt. 2 unrolled	309517	1.29	775 MHz

Table 3. Results obtained from Synopsys

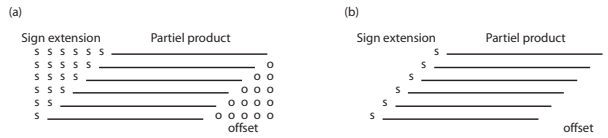

Figure 7. Reducing the size of the adder tree

would decrease the critical path and make it faster. Because the critical path probably would shift from the **2cs** multiplier to the **cs-normal** multiplier. This is also one of the rare cases where an improvement in speed also gives a decrease in area.

Another place to reduce the area is in the adder tree. The sign extension and the offset as can be seen on figure 7a illustrates that we are wasting a lot of area. Some effort could be put in minimizing the adder tree, to look like figure 7b. The last thing that we would recommend is designing a deeper pipeline. Probably by putting a pipeline stage between the partiel product generator and the adder tree.

6 Conclusion

We have successfully made a Newton-Raphson unit, which is able to calculate $\frac{1}{d}$. Our priority at this project was to synthesize our implementations and get some results. We have proposed two different kinds of architectures and identified the best one by focusing on speed and area use. The fastest proved to be a fully redundant (alternative 2) unrolled, reaching 775 MHz with the area use of 309517 μm^2.

We furthermore proposed a way to further enhance the speed and decrease the area use by make the two multipliers share the result from the recoder, so that the only one of them needs one. This could probably shift the critical path form the **2cs** multiplier to the **cs-normal** multiplier and hereby increasing speed and decreasing area.

References

[1] J.D. Bruguera and T. Lang. Implementation of the fft butterfly with redundant arithmetic. *IEEE transactions on Circuits and Systems II: Analog and Digital Signal Processing vol. 43 Issue 10*, pages 717–723, 1996.

[2] Milos D. Ercegovac and Thomas Lang. *Digital Arithmetic*. Morgan Kaufmann, 2004.

[3] Debjit Das Sarma and David W. Matula. Measuring the accuracy of rom reciprocal tables. *IEEE transactions on computers vol. 43 Issue 8*, pages 932–940, 1994.