

# Modeling and Realization of the Floating point Inverse Square Root, Square Root, and Division unit (FP ISD) using VHDL and FPGAs

Jaafar Alghazo  
Department of Electrical and Computer Engineering  
University of Central Florida  
Orlando, FL, 32826  
Email: Alghazo@gmail.com

## Abstract:

*In this paper, we model and synthesis a high speed Arithmetic inverse square root, square root, and division (ISD) unit based on existing algorithms similar to the unit in [3]. With area/speed tradeoff limitation, our concentration was on designing high speed Arithmetic units with moderate area increase. Our concentration on the (ISD) unit using digit recurrence algorithms led to the modeling of a robust unit, sharing the same recurrence and same hardware to perform all three operations. Minor differences occur in the initialization stage and number of iterations and final rounding. Synthesis tools were used to evaluate our model and reports showed that our ISD unit has a minimum path delay of 12.443ns. Area was moderate and within range when theoretically comparing three separate units to complete the three operations. We further decrease the latency of the ISD unit by modeling short fast adder in the critical path.*

Keywords: Modeling and Realization, VHDL, FPGAs, Division, Square Root, Inverse Square Root.

## 1. Introduction:

The development in computer arithmetic architectures over the past decades has permitted the performance of computer hardware to maintain its exponential growth. The exponential growth has a disadvantage that of increasing the hardware and software complexity. This rise in complexity also obscures testability, certifiability, adaptability, performance tuning, and evaluation of various trade-offs. This all leads to elevating development costs [1-2].

The Inverse Square Root operation is an essential operation for applications involving graphics, multimedia, and scientific computations. Division and square root are also essential operation in many application involving scientific computation applications.

The robust arithmetic operations are configured for FPGAs design and has the flexibility, like other FPGA-based design algorithm, of being reconfigurable, making our coprocessor more practical for arithmetic intensive research. The coprocessor is also more optimized and robust. FPGA-based robust arithmetic coprocessors are an alternative solution to the costly and time consuming ASICs coprocessors.

In many numerically intensive applications, addition is the most frequent arithmetic operation, followed by multiplication. Division and the other elementary function are the least frequent operations. This is why division and other elementary function did not receive much attention for a long time. But as more numerically intense applications are being used, the need for fast dividers and other elementary function units becomes immanent. The time as well as area of such arithmetic operation circuits is very crucial [1-2].

The time of arithmetic operations in numerically intensive applications can be improved with an application specific integrated circuit (ASIC), designed specifically for a particular application. The development cost and time length of fabrication make such ASIC circuits impractical. An alternative can be Field Programmable Gate Arrays (FPGAs) based arithmetic units. According to applications, a designer or researcher can select from among various algorithms for addition, multiplication, division and elementary functions to form an arithmetic unit suited for that particular application. FPGAs programmability delays make it slower than ASIC designs yet faster than software implementations [1-5].

## 2. THE ISD UNIT

The ISD unit we model here is similar to the one in [3]. Our contribution is the modeling and realization of the ISD unit using VHDL and FPGAs. The ultimate goal of this research is reconfigurable Math Coprocessors.

Researchers would be able to configure their own math coprocessor according to the nature of their research.

In this paper our goal was to model a unified unit that can perform all three operations by sharing the same resources. We chose recursive techniques to model our robust unit. We want our unit to perform all three operations using the same lookup table and recursion steps for IEEE single precision floating point operands. We modeled a robust unit in radix 4 implementation. The reason we set one input port only is that two out of the three operations has one operand input only. We set a control signal to initialize the registers and start the recurrence after operands are in the input registers and are unpacked .

## 2.1 RADIX-4 INVERSE SQUARE ROOT

For inverse square root (ISQRT), the operand range must be modified. The result exponent is calculated by changing the operand exponent to even and adjusting the operand according to the exponent change. The exponent is then divided by 2 to get final exponent for the result. The operand and result are scaled to  $\frac{1}{4}$  and 1 respectively. The scaled operand will be:

$$g = \begin{cases} 2^{-1}h & \text{when } e_h \text{ odd} \\ 2^{-2}h & \text{when } e_h \text{ even} \end{cases} \quad g \in [0.25, 1), r \in (1, 2] \quad (1)$$

This scaling is made so that the results will remain within the range of IEEE standard. Of course when a result is equal to 2 exactly it should be handled as an exception to equation . The operand scaling will allow simple prescaling of  $h$  . The results will be in the range of the

IEEE standard except for  $R^{(N)} = 1$  and  $R^{(N)} = 2$  .

The digit recurrence algorithms the digits of the result are retired every iteration. After N iterations, the result is represented in radix  $\beta = 2^b$  with:

$$R^{(N)} = R^{(0)} + \sum_{i=1}^N r^{(i)} 4^{-i} \quad (2)$$

we obtained the following recurrence equations:

$$\begin{aligned} w^{(k+1)} &= 4w^{(k)} - r^{(k+1)}4^{(k)} - (r^{(k+1)})^2 C^{(k)} \\ B^{(k+1)} &= B^{(k)} + 2r^{(k+1)}C^{(k)} \\ C^{(k+1)} &= \beta^{-1}C^{(k)} \end{aligned} \quad (3)$$

With

$$\text{initial } B^{(0)} = gR^{(0)} \quad \text{and} \quad C^{(0)} = 2^{-1}4^{-1}g .$$

$r^{(k+1)}$  .  $r^{(k+1)}$  is selected from a look up table using the assimilated values of  $r^{(k+1)} = SEL(4\tilde{w}^{(k)}, \tilde{B}^{(k)})$  .

$B^{(k)} = gR^{(k)} \approx \sqrt{g}$  , with  $B^{(k)}$  in the range

of  $0.5 \leq R^{(k)} < 1$  . The multiplication of operands by a digit is done using multiplexers. Iteration delay is reduced

by retiming the iteration. The selection in this case will be done at the end of the iteration instead beginning. The selection of  $r^{(k+2)}$  is done using estimates of  $4w^{(k+1)}$  and  $B^{(k+1)}$  . Retiming has the advantage of both assimilating  $B^{(k+1)}$  with less logic, and assimilating  $w^{(k+1)}$  simultaneously with the selection function.

With  $\beta = 2^b$  , and  $n$  operand bits, the number of

iteration needed to get a correctly rounded result is  $\left\lceil \frac{n}{b} \right\rceil$  .

The exponent can be calculated at any time using the following equations:

$$e_n = \begin{cases} \frac{-(e_h + 1)}{2} & \text{if } e_h \text{ odd} \\ \frac{-(e_h + 2)}{2} & \text{if } e_h \text{ even} \end{cases} \quad (4)$$

The initial values of for the inverse square root at easily generated and are:

$$w^{(0)} = 2^{-1}(1 - g(R^{(0)})^2)$$

$$\text{With } g = \begin{cases} 2^{-1}h & \text{if } e^h \text{ odd } (g \geq 0.5) \\ 2^{-2}h & \text{if } e^h \text{ even } (g < 0.5) \end{cases} \quad \text{and}$$

$$R^{(0)} = \begin{cases} 1 & \text{if } e^h \text{ odd } (g \geq 0.5) \\ 2 & \text{if } e^h \text{ even } (g < 0.5) \end{cases}$$

$$\text{So } w^{(0)} = \begin{cases} (0.5)(1-h) & \text{if } e^h \text{ even} \\ (0.5)(1-0.25h) & \text{if } e^h \text{ odd} \end{cases} \quad (5)$$

$$B^{(0)} = gR^{(0)} = (0.5)h \quad (6)$$

$$C^{(0)} = 2^{-1}4^{-1}g = \begin{cases} 2^{-3}4^{-1}h & \text{if } e^h \text{ even} \\ 2^{-2}4^{-1}h & \text{if } e^h \text{ odd} \end{cases} \quad (7)$$

[3-11].

## 2.2 RADIX-4 DIVISION

To implement the operation in the range of IEEE single precision using the recurrence algorithm developed for inverse square root, we calculate  $R^{(N)} = \frac{y}{g}$  with

$x$  scaled to  $y$  and  $h$  scaled to  $g$  . The recurrence equation for division becomes:

$$w^{(k+1)} = 4w^{(k)} - r^{(k+1)}g \quad (8)$$

With  $w^{(0)} = y - R^{(0)}g$  , with  $w^{(k)}$  bounded

with  $-\rho g < w^{(k)} < \rho g$  to get the digit of the result  $r^{(k+1)}$  .

The combination of this recurrence with the recurrence developed for inverse square root we set  $B^{(0)} = g$  and  $C^{(0)} = 0$ . The requirement must also be met to use the same selection function, implying that  $g \in [0.5, 1)$ , resulting in  $g = (0.5)h$ . With  $R^{(0)} = \text{constant}$ , the range of  $d$  is  $\rho$ . Which would cause a problem when  $\rho < 1$ , because that would not be sufficient for IEEE range  $[1, 2)$ .  $x$  and  $h$  are compared and  $x$  is usually shifted when it is greater than  $h$ , or this could be handled in the unnormalized numbers exception. In order to resolve these points and implement the requirements we set  $R^{(0)} = 0$  and  $y = \frac{y}{8}$ , this will lead to  $\frac{y}{g} \in [\frac{1}{8}, \frac{1}{2})$ , to change this into IEEE range we need to shift two or three position, according if  $\frac{y}{g} \geq \frac{1}{4}$  or  $\frac{y}{g} < \frac{1}{4} \cdot \lceil \frac{n+3}{b} \rceil$ . Iterations are required to calculate the correctly rounded result for division.

In summary, we use the same recurrence equation for division as that developed for inverse square root, with the following initialization:

$w^{(0)} = y = \frac{x}{8}$ ,  $B^{(0)} = g = \frac{h}{2}$ ,  $C^{(0)} = 0$ , and  $R^{(0)} = 0$ . The result will not be in IEEE range and will require a final left shift to be in the range  $[1, 2)$  according to the following:

$$q = \begin{cases} 2^2 R^{(N)} & \text{if } R^{(N)} \geq 0.25 \\ 2^3 R^{(N)} & \text{if } R^{(N)} < 0.25 \end{cases} \quad (9)$$

The final exponent will be:

$$eq = \begin{cases} e^x - e^h & \text{if } R^{(N)} \geq 0.25 \\ e^x - e^{h-1} & \text{otherwise} \end{cases} \quad (10)$$

[3-11].

## 2.3 RADIX-4 SQUARE ROOT

To implement the operation in the range of IEEE single precision using the recurrence algorithm developed for inverse square root, we calculate  $R^{(N)} = \sqrt{y}$  with  $x$  scaled to  $y$ . The result of square root,  $R^{(N)} \in [0.5, 1)$ , in order to use the same selection function. This is achieved by scaling the operand  $x$  so that  $y = 0.5x$  if the exponent is odd and  $y = 0.25x$  if exponent is even. We choose an initial value  $R^{(0)} = 1$ . A final shift will also be required for square root. The final shift will be left shift of one bit to put the final result in the IEEE range. This algorithm requires  $\lceil \frac{n+1}{b} \rceil$  iterations to calculate the correctly rounded result.

The same recurrence for inverse square root is used for square root with the following initializations:

$$w^{(0)} = \begin{cases} 0.5(0.25x - 1) & \text{if } e^x \text{ even} \\ 0.5(0.5x - 1) & \text{if } e^x \text{ odd} \end{cases} \quad (11)$$

$$B^{(0)} = R^{(0)} = 1 \quad \text{and} \quad C^{(0)} = (0.5)4^{-1}$$

$2R^{(N)}$  is rounded to get the rounded result. The final exponent is  $es = \lfloor \frac{ex}{2} \rfloor$  [3-11].

## 2.4 COMBINED RADIX-4 ISD UNIT

The combined unit is designed so that all operations will use the same recurrence implying the same hardware. The unit will execute the operations according to the following phases:

Phase 1: the Initialization and obtaining  $r^{(1)}$

$$w^{(0)} = \begin{cases} 0.5(m - v) & \\ 1 & \text{if } ISQR T \end{cases}$$

$$m = \begin{cases} 0.25x & \text{if } DIV \text{ or } SQR T \text{ (} e^x \text{ even)} \\ 0.5x & \text{if } SQR T \text{ (} e^x \text{ odd)} \end{cases}$$

$$v = \begin{cases} h & \text{if } ISQR T \text{ (} e^h \text{ even)} \\ 0.5h & \text{if } ISQR T \text{ (} e^h \text{ odd)} \\ 0 & \text{if } DIV \\ 1 & \text{if } SQR T \end{cases}$$

$$B^{(0)} = \begin{cases} 0.5h & \text{if } DIV \text{ or } ISQR T \\ 1 & \text{if } SQR T \end{cases}$$

$$C^{(0)} = (0.5)4^{-1} * \begin{cases} 0.25h & \text{if } ISQR T \text{ (} e^h \text{ even)} \\ 0.5h & \text{if } ISQR T \text{ (} e^h \text{ odd)} \\ 0 & \text{if } DIV \\ 1 & \text{if } SQR T \end{cases}$$

$$R^{(0)} = \begin{cases} 2 & \text{if } ISQR T \text{ (} e^h \text{ even)} \\ 1 & \text{if } ISQR T \text{ (} e^h \text{ odd)} \text{ or } SQR T \\ 0 & \text{if } DIV \end{cases}$$

$$r^{(1)} = SEL(4\hat{w}^{(0)}, \hat{B}^{(0)})$$

Where ISQRT (inverse square root), SQRT (square root), DIV (division),  $4\hat{w}^{(0)}$  is the assimilated shifted values of  $w^{(0)}$ , and  $\hat{B}^{(0)}$  is the assimilated values of  $B^{(0)}$ .

Phase 2: from  $k=2-N$ ,

$$R^{(k+1)} = \text{Convert}(R^{(k)}, r^{(k+1)})$$

$$w^{(k+1)} = 4w^{(k)} - B^{(k)}r^{(k+1)} - C^{(k)}(r^{(k+1)})^2$$

$$B^{(k+1)} = B^{(k)} + 2r^{(k+1)}C^{(k)}$$

$$C^{(k+1)} = 4^{-1}C^{(k)}$$

$$r^{(k+2)} = SEL(4\hat{w}^{(k+1)}, \hat{B}^{(k+1)})$$

Phase 3: for  $N+1$ : last residual, correct and round.

Division and square root must be shifted to be in the IEEE range of  $[1, 2)$ .

We implemented our algorithm according to the above set rules and phases. The complete implementation of our algorithm is shown in figures 1, figure 2, and figure 3 below.

The figures show the logic behind the ISD unit, Multiplexers (MUX) were used for multiplication, CPA, CSA, 4:2 Compressors were used for addition. Shift registers were used for shifting the operands when needed. Figure 1 shows the logic behind implementing the recurrences  $B^{(k+1)} = B^{(k)} + 2r^{(k+1)}C^{(k)}$  and  $C^{(k+1)} = 4^{-1}C^{(k)}$ . Where the multiplication of  $2r^{(k+1)}C^{(k)}$  is done using a 4-to-1 MUX. This is then added to  $B^{(k)}$  using a CPA, 8 bits of each are added in a smaller CPA in parallel to get the assimilated bits  $\tilde{B}^{(k+1)}$ . The second recurrence is nothing but a right shift. Figure 2 shows the logic behind the implementation of the recurrence  $w^{(k+1)} = 4w^{(k)} - B^{(k)}r^{(k+1)} - C^{(k)}(r^{(k+1)})^2$ . As can be seen in the figures, the multiplication of  $C^{(k)}(r^{(k+1)})^2$  and  $B^{(k)}r^{(k+1)}$  is done using MUXs. A 4 to 2 CSA is used produce the result. The most significant part of the sum and carry are added using CPA to produce the assimilated  $4\tilde{w}^{(k+1)}$ .

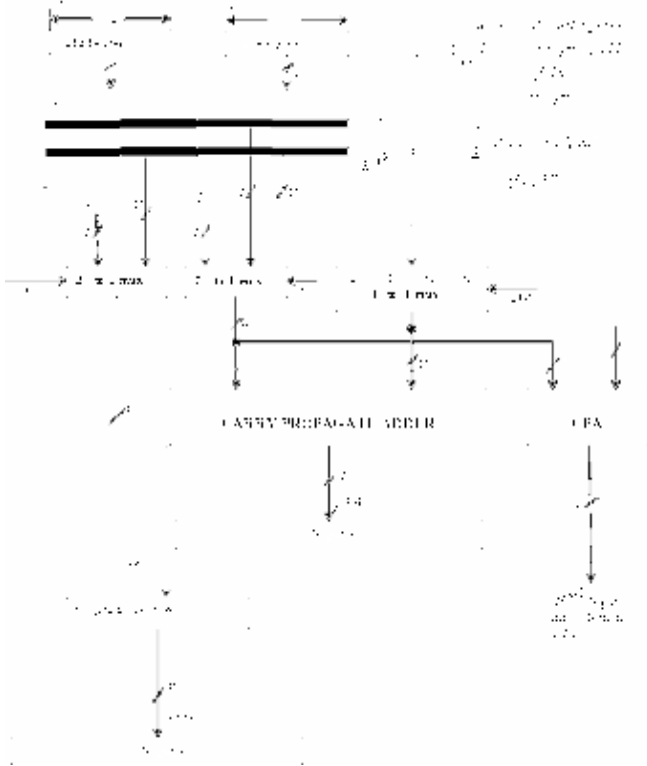


Figure 1. Recurrence of  $C^{(k+1)}$  and  $B^{(k+1)}$

Figure 3 shows the implementation of the lookup table which is implemented as a ROM. The inputs to the lookup table are calculated and the value is output from the lookup table. The lookup table is an important part of any recurrence algorithm and the values of the lookup table have to be generated and stored in either a ROM or PLA. In our model, we designed the ISD unit such that all three operations share the same lookup table. Through careful

analysis and derivation the lookup table that accommodates all three operations was generated.

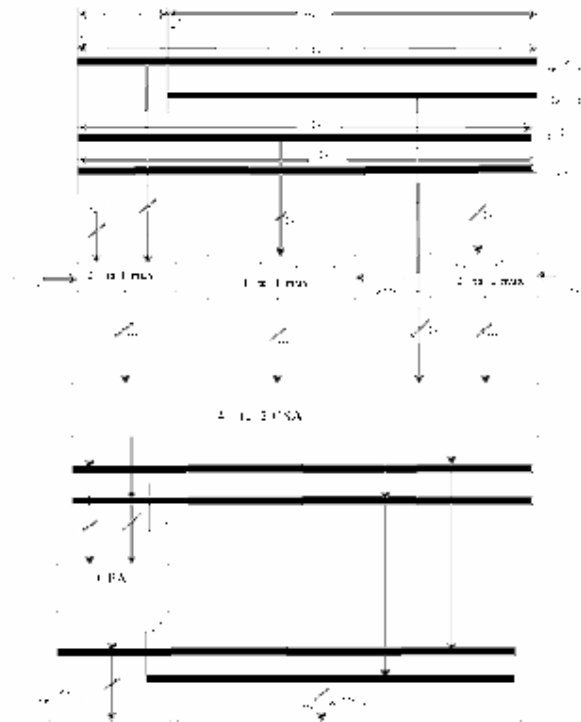


Figure 2. Recurrence of  $w^{(k+1)}$

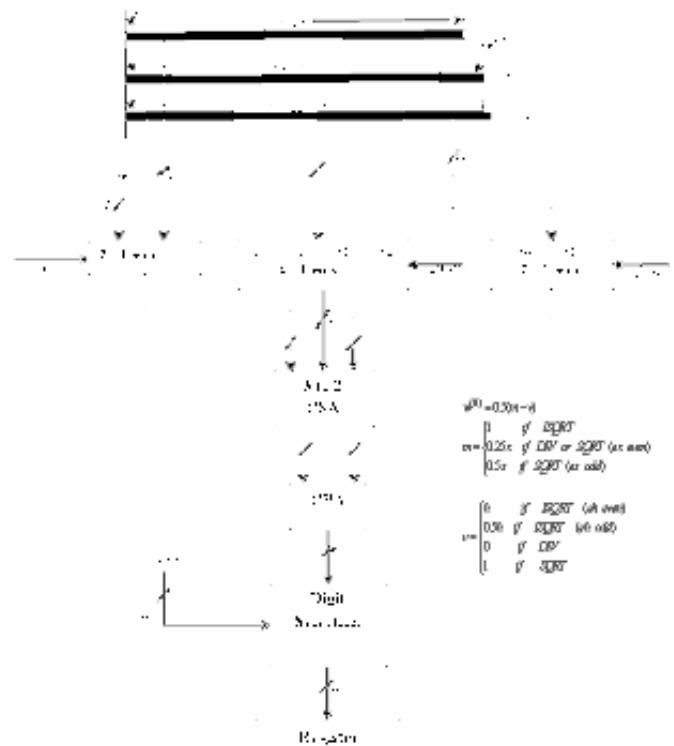


Figure 3. Selection table and selection table inputs

The inputs to the table are  $\tilde{B}^{(k+1)}$  in the form [I.FFFFF] (I is integer, F is fractional) and  $4\tilde{w}^{(k+1)}$  in the form [III.FFFF]. It is thus apparent that  $\tilde{B}^{(k+1)}$  must be scaled by 32 and  $4\tilde{w}^{(k+1)}$  must be scaled by 16. The lookup table is shown in table 1 below.

This table however did not give accurate results for some values. It was tested vigorously and small modifications had to be made to produce a modified lookup table that would results in accurate results for all inputs. The modified table is shown in table 2.

It is important to note here that the modified version of the table was obtained through trial and error and the mathematical explanation and proof is left as future work. As far as the updated table is concerned values that were changed were only some border values. From the table

above it is evident that the table largely remains the same [3-11].

### 3. MODIFICATION ON RADIX-4 ISD UNIT

After designing our Radix-4 ISD unit, we explored different options to enhance the speed of the unit further. In order to speed up the algorithm we have to determine the critical path. Shifting is not part of the critical path as it is implemented in the wiring. To get  $r^{(k+2)}$ , the critical path therefore consists of  $t_{r^{(k+1)}} = t_{mux} + t_{csa} + t_{cpa} + t_{sel}$ . Where  $t_{mux}$  is delay across the multiplexer.  $t_{csa}$  and  $t_{cpa}$  are short carry

Table 1- Selection table for Radix-4 ISD unit

<b>B<sub>L</sub></b>	<b>16</b>	<b>17</b>	<b>18</b>	<b>19</b>	<b>20</b>	<b>21</b>	<b>22</b>	<b>23</b>	<b>24</b>	<b>25</b>	<b>26</b>	<b>27</b>	<b>28</b>	<b>29</b>	<b>30</b>	<b>31</b>
<b>m<sub>1</sub></b>	-13	-14	-14	-15	-16	-17	-17	-18	-18	-19	-21	-21	-21	-23	-24	-24
<b>m<sub>0</sub></b>	-5	-5	-5	-6	-6	-6	-7	-7	-7	-8	-8	-8	-9	-9	-9	-10
<b>m<sub>1</sub></b>	3	4	4	4	4	4	4	5	7	7	7	7	8	8	8	8
<b>m<sub>2</sub></b>	12	13	14	15	15	15	16	17	18	18	19	19	22	22	22	22

Table 2- modified selection table for Radix-4 ISD

<b>B<sub>L</sub></b>	<b>16</b>	<b>17</b>	<b>18</b>	<b>19</b>	<b>20</b>	<b>21</b>	<b>22</b>	<b>23</b>	<b>24</b>	<b>25</b>	<b>26</b>	<b>27</b>	<b>28</b>	<b>29</b>	<b>30</b>	<b>31</b>
<b>m<sub>1</sub></b>	-13	-14	-14	-15	-18	-18	-18	-18	-18	-19	-20	-20	-20	-23	-24	-24
<b>M<sub>0</sub></b>	-5	-5	-6	-6	-6	-6	-7	-8	-8	-8	-8	-8	-9	-9	-9	-10
<b>M<sub>1</sub></b>	3	4	4	4	4	4	4	5	5	7	7	7	8	8	8	8
<b>M<sub>2</sub></b>	12	13	14	14	15	15	16	17	17	18	19	19	20	20	21	22

save and carry propagate adders respectively.  $t_{sel}$  is the delay for the selection function. As will be shown later this  $t_{r^{(k+1)}}$  has the largest delay in generation of variables.

This is actually the main reason why we use the assimilated bits  $4\tilde{w}^{(k+1)}$  and  $\tilde{B}^{(k+1)}$  is because we want to reduce the delay by using shorter adders, thus faster adders.

The other possible critical paths are:

$$t_{w^{(k+1)}} = t_{mux} + t_{csa} + t_{cpa}$$

$$t_{B^{(k+1)}} = t_{mux} + t_{cpa}$$

In order to further reduce delay time we can implement faster adders. In our case we implanted and tested with a Carry select adder to replace the CPA. A 51bit CPA has a Maximum combinational path delay: 96.398ns as obtained from the Xilinx synthesis report. On the other hand a 51 bit Carry Select Adder has a Maximum combinational path delay: 57.950ns. As a result replacing the CPA with a

Carry select adder makes a significant difference. For a 51 bit CSA the Maximum combinational path delays: 8.234ns. Since the generation of the  $r^{(k+1)}$  multiple will have the largest delay reducing the delay here is critical to speeding up the algorithm. Another approach we took towards cutting this delay was to try and reduce delay in the selection function. As mentioned earlier the selection table ideally takes 7 assimilated bits of  $4w$  as input into the selection table. The inputs are in the following format (III.FFFF), where I and F are integer and fractional parts respectively. Since this input is scaled by 16 we can reduce the inputs to use only 5 inputs instead of 7. The reason we can do this is because scaling the 2 most significant bits gives us values greater or equal to 32 which is higher than the largest useful value in the table of 25. We can therefore identify cases when either or both the 2 MSBs have a value of one and just pass the value 25 as input instead of determining input first. Also in the case that we have to determine the input from the rest of the bits the 5 bit input to scaled integer converter will be smaller and faster due to the fact that it uses fewer inputs [3-11].

## 4. ISD RESULTS

The objective of the ISD unit was to model one units that would be able to do the three operation division, square

root, and inverse square root using the same hardware and same lookup table. The synthesis was generated and the device utilization report extracted shown below:

Number of Slices: 1931 out of 3072 62%  
 Number of Slice Flip Flops: 2050 out of 6144 33%  
 Number of 4 input LUTs: 3209 out of 6144 52%  
 Number of bonded IOBs: 67 out of 166 40%  
 Number of GCLKs: 1 out of 4 25%

Synthesis report showed that our ISD unit has a Minimum period: 12.443ns (Maximum Frequency: 80.366MHz). Our algorithm was able to achieve requirement we wanted to achieve. The results of the three operations are shown below:

### 4.1 DIVISION RESULTS

ISD unit was able to correctly execute the division operation. The figures are named according to their input values.



Figure 4: DIV (1.1/0.13)



Figure 5: DIV (11111/5)



Figure 6: DIV (0.34/0.00453)

### 4.2 SQUARE ROOT RESULTS

The square root results in the ISD unit are shown below for floating point single precision operands:

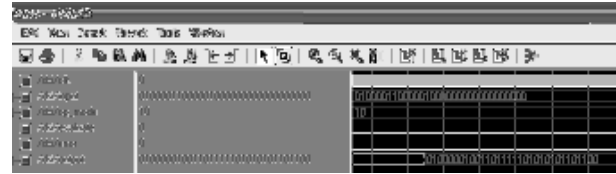


Figure 7: SQRT (132)



Figure 8: SQRT (0.00132)



Figure 9 : SQRT (-5612) Exception Handling

### 4.2.3 INVERSE SQUARE ROOT RESULTS

The inverse square root results are shown below for IEEE singles precision operands:



Figure 10: ISQRT (5.7)



Figure 11: ISQRT (333)

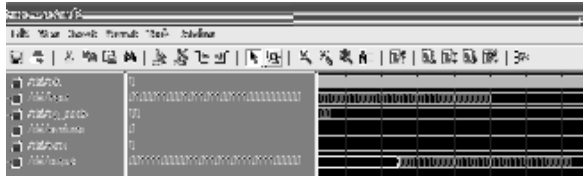


Figure12: ISQRT (11111)



Figure 13: ISQRT (-11111) Exception Handling



Figure 14: ISQRT (0.57)



Figure 15: ISQRT (0.0013245)

[3] T. Lang and E. Antelo, "Radix-4 Reciprocal Square-Root and Its Combination with Division and Square Root." IEEE Transactions on Computers, vol. 52, no. 9, pp.1100-1114, 2003

[4] I. Koren, *Computer Arithmetic Algorithms*, A K peters, 2 ed, 2002.

[5] D. L. Harris, S. F. Oberman, And M.A. Horowitz, "SRT division architectures and implementations," in *Proceedings of the 13<sup>th</sup> IEEE Symposium on Computer Arithmetic*, Asilomar, California, USA, pp.18-25, July, 1997.

[6] S. F. Oberman and M. J. Flynn, "Design Issues in Division and Other Floating Point Operations", *IEEE Transaction of Computers*, vol. 48, No. 2, pp. 154-161, 1997.

[7] S. F. Oberman and M. J. Flynn, "Minimizing the Complexity of SRT Table", *IEEE Transaction on Very Large Scale Integration*, vol. 6, No. 1, pp. 141-149, 1997.

[8] S. F. Oberman and M. J. Flynn, "Division Algorithms and Implementations", *IEEE Transaction of Computers*, vol. 46, No. 8, pp. 833-854, 1997.

[9] M. Ercegovac, T. Lang, and P. Montuschi, "Very High Radix Division with Prescaling and Selection by rounding", *IEEE Transactions on Computers*, vol. 43, no. 8, august 1994.

[10] M. D. Ercegovac and T. Lang, "Simple Radix-4 Division with Operands Scaling," IEEE Transactions on Computers, vol. 39, no. 9, 1990.

[11] J. Cortadella and T. Lang, "High- Radix Division and Square-root with Speculation", *IEEE Transactions on Computers*, vol. 43, no. 8, August 1994, pp. 919-931

## 5. FUTURE WORK

For future work we plan to give a mathematical proof for the lookup table modification. We also plan to explore more units for arithmetic operations. We plan to explore new optimization techniques for the units that were already synthesized and those we plan to develop. Reconfigurable processors will be available from off shelf algorithms. Researchers will be able to configure their own math coprocessors from these off the shelf optimized arithmetic units.

## 6. REFERENCES

- [1] M. J. Flynn and S. F. Oberman, *Advanced Computer Arithmetic Design*, John Wiley & Sons, Inc, 2001  
 [2] J. L. Hennessy and D.A Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufman, 3 ed., 2003.