Lab 4 Report: Creating a Hardware Reciprocal Square Root Function

Purpose:
The purpose of this lab is to implement the reciprocal square root function in hardware. Newton's Method and initial conditions will be used to compute the solution as seen by the block diagram in figure 1.

Block Diagram:

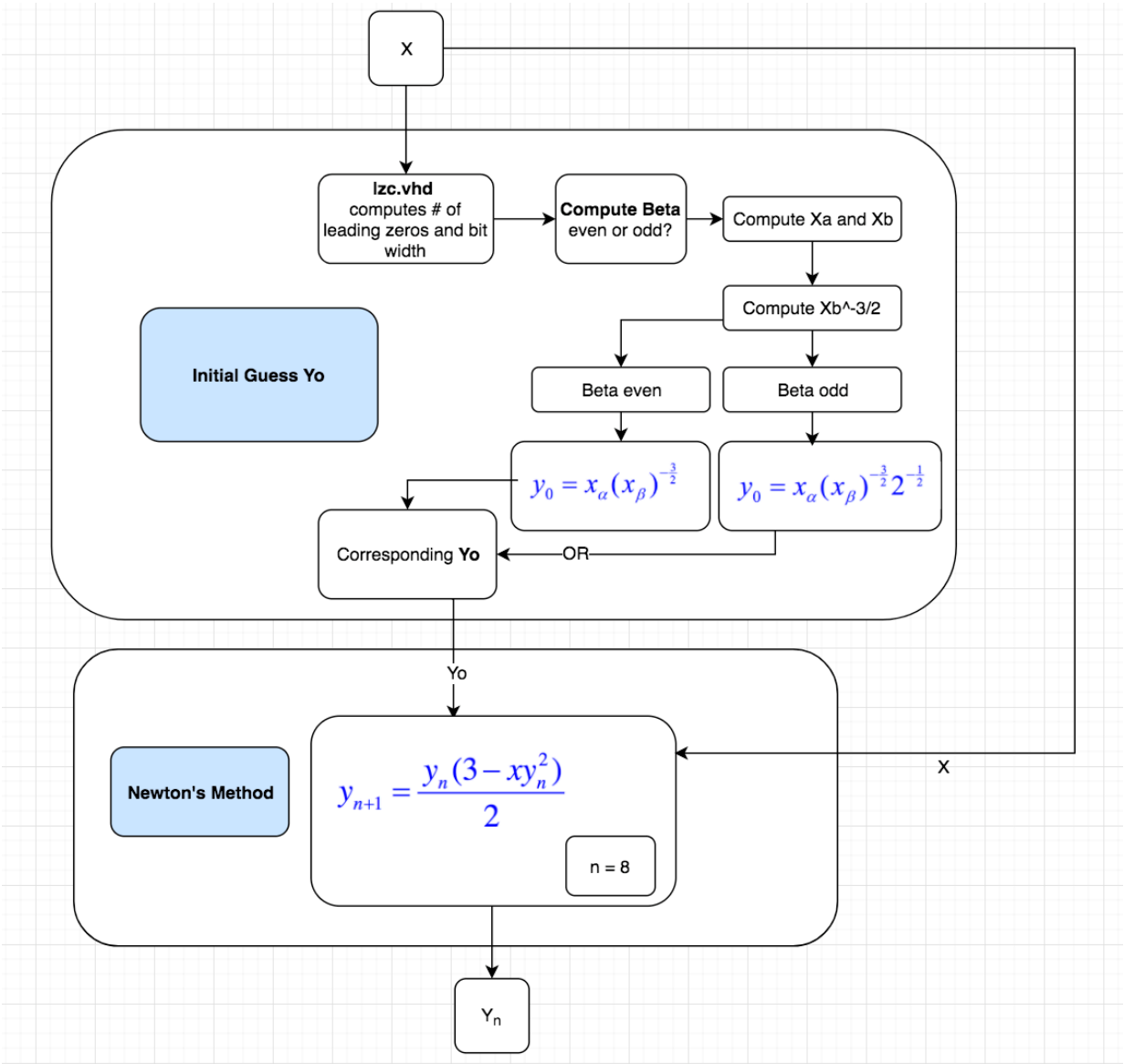

Figure 1: Newton's Method Block Diagram

Major Block Components and Implementation:

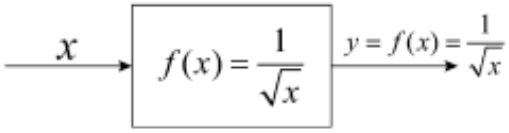

Figure 2: Reciprocal Square Root Function

This function can be solved using Newton's method. F(y) is defined and we use this equation to solve for the positive root F(y) = 0. Using an iterative process we can solve for y_{n+1}. All relevant equations used are defined below.

$$F(y)=\frac{1}{y^2}-x \qquad y_{n+1}=y_n-\frac{F(y_n)}{\frac{dF(y_n)}{dy}} \qquad \frac{dF(y)}{dy}=\frac{-2}{y^3}$$

Using these equations the iterative update formula is determined to be

$$y_{n+1}=\frac{y_n(3-xy_n^2)}{2}$$

The reciprocal square root function is separated into two parts, the initial guess and the iterative formula. Both of these must be computed in order to solve the function. In order to calculate these values a top level VHDL component rsr.vhd is created. This component instantiates two sub-components. One sub-component will compute the initial guess y_0 and the second will implement the iterative formula that solves Newton's Method.

Component 1: Compute Initial Guess y_0:
When solving for y_0, we must consider the fact that arbitrary values cannot be used. The graph in figure 3 shows the areas of convergence for this function.

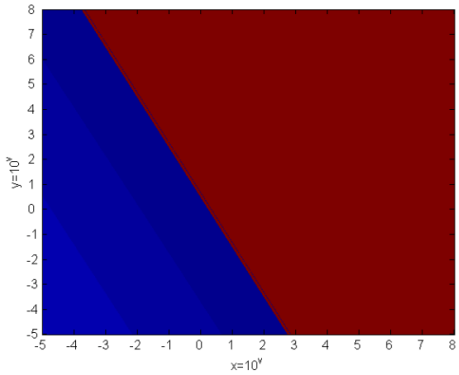

Figure 3: Areas of Convergence for y_0

In Figure 3 above, the blue region corresponds to where the function converges, while the red shows where the function does not converge. This figure demonstrates that the method will not work with large values of y_0. When computing large values of y_0, the slope becomes flat quickly and with finite computer precision, the slope will eventually go to zero. This will result in the failure of Newton's Method.

Sub-component y0.vhd is created to solve for initial guess. First, we determine the number of leading zeros in x with bit-width W. Using the given lzc.vhd component, we can compute the number of leading zeros and report it in the input std_logic_vector. We used values of W=36 and F=18 for this lab. Next we compute beta and determine whether it is even or odd. The value determined for beta is used to calculate beta even and beta odd, both of which are denoted as alpha. Then, the value of x must be determined for both alpha and beta. Using a lookup table we can determine the value of $(x_B)^{-3/2}$. The fractional bits of x_B are the address used for lookup table. With these values calculated, we can solve for y_0 for beta even and odd.

Component 2: Newton's Method

Now that y_0 is computed, a second component was created that contains a state machine that will iterate and find y_n. The number of iterations is based on the numerical precision needed. For this lab exercise the number of iterations was determined with a value of F=18.

Test and Verification:

To test our design and ensure that it works properly, we used Matlab test bench script to compare the calculated values in Matlab to those in Modelsim. This test bench will drive Modelsim and allow for the creation of test vectors in Matlab that will be compared with the output of Modelsim.

Conclusion:

This lab allowed the opportunity to learn how to implement the reciprocal square root function in hardware. To accomplish this task, many steps were required and code was written in VHDL and Matlab in order to verify the design and check for error. The results obtained were very precise. We only had one error in 1000 iterations. This error was equal to $3.815*10^{-6}$. Our min error was zero and average error was $3.815*10^{-9}$. We experienced zero error for almost all runs of the program. Overall, this is an acceptable error and level of precision.