Jesse Anzelc

Thomas Turner

6/7/14

EEC 180B

Lab 6 Report

1. Cover sheet.

Attached

2. Result Discussion - Discuss how you implemented each module.

We used fixed point arithmetic to implement the reciprocal unit, inverse square root, and square root. We took advantage of knowing our input D would be bounded 0.5 < D < 1.0 by hard coding our sign bit to be 0 and our exponent bits to be 0x7F = 127. Using a seed value of 1 and the Newton-Raphson method, we iteratively calculated the reciprocal using xi+1 = xi\*(2-D\*xi) . The seed value used for the Newton-Raphson method was also 1 when calculating the inverse square root. The iterative equation used to calculate the inverse square root is .5 \* xi \* (3 - D\*xi\*xi). To calculate the square root, we multiplied the floating point value by its inverse square root [D / sqr(D)].

3. Timing Report (Full FSU or how much you got done)

Attached.

4. Power Report (Full FSU or how much you got done)

Attached.

5. Amount of time it takes to do ONE iteration on both your reciprocal and square root modules. Also include how many times you do your iterations or how you decided how many iterations need to be done. If the number is not fixed, give me an average.

One iteration takes just one clock cycle. We found that 6 iterations worked for all cases tested, but we set our done signal after 7 iterations for safety.

6. All code to Smartsite.

Submitted.

7. Extra Credit - Describe to me why you think you deserve extra credit for something you did to improve the system. If you expanded the range of your inputs or figure out how to do negative numbers, I can give you some extra credit. You can talk to Professor Akella if you have an idea for extra credit as well.

Negative numbers - The solution for reciprocal is trivial, the sign bit just follows the input as reciprocal does not change the sign. For inverse square root, we calculated the IEEE format number as usual, but included an LED signal to represent -i. We did the same for square root, but used a different LED to represent +i.

A different method was used to calculate the inverse square root. This method can be found here: <http://en.wikipedia.org/wiki/Fast_inverse_square_root>

The algorithm works by shifting the IEEE floating point number right by 1. After the shift we use this number to subtract it from a hex value called the “magical number constant” which equals: **0x5f3759df.**

By performing this arithmetic we could get a really good estimate for the inverse square root. To make this calculation more precise, you would input the calculated value in the Newton-Raphson equation. This algorithm improves the clock frequency and also allows the result to converge in 4 clock cycles.