University of California, Davis

Department of Electrical and Computer Engineering

EEC180B DIGITAL SYSTEMS II Spring Quarter 2014

LAB 6 – RSU: A Functional Unit to Compute Reciprocal and Square Root

Objective

The purpose of this laboratory exercise is to design and optimize a functional unit that computes square root and reciprocal of single precision floating point number on an FPGA. You will use Verilog based design flow to simulate and test your design on the Altera DE2 board. Part of the grade for this project will depend on how efficient your design is in terms of performance, area, and power consumption.

Background

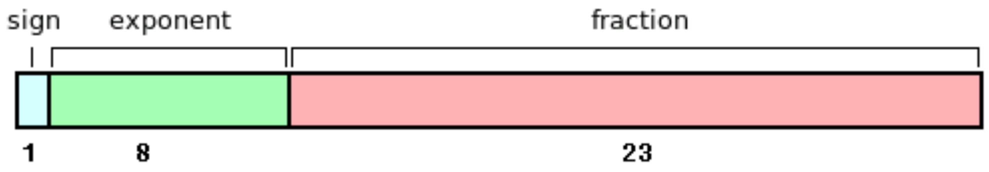


Figure 1 - Floating Point Representation in IEEE 784 Standard

The single precision format is shown in the Figure 1. Numbers in this format are composed of the following three fields:

1-bit sign, S: A value of '1' indicates that the number is negative, and a '0' indicates a positive number.
Bias-127 exponent, e = E -127. This gives us an exponent range from **Emin** = -126 to **Emax** = 127.
Fraction, f: The fractional part of the number. The fractional part must not be confused with the significand, which is 1 plus the fractional part. The leading 1 in the significand is implicit. When performing arithmetic with this format, the implicit bit is usually made explicit. To determine the value of a floating point number in this format we use the following formula:

$$Value = (-1)^S \times 2^{e-127} \times 1.f_{23}f_{22}f_{21}.....f_0$$

The reciprocal and square root can be computed iteratively using Newton-Raphson method starting from an initial value. The choice of the initial value is important as it dictates the speed of convergence, which means the number of iterations required to compute the value or the latency of the functional unit.

Newton-Raphson algorithm is based on a general method to obtain a single zero α of function f (i.e., f(α) = 0 and f_(α) $\neq$0). If x_0 is close enough to α, the following iteration converges towards α:

$x_{i+1} = x_i - f(x_i) / f\_(x_i)$

where f_(x) denotes the derivate of f with respect to x.

To compute the reciprocal 1/d , one could choose f(x) = 1/x – d and the corresponding iteration is $x_{i+1} = x_i * (2 - d*x_i)$

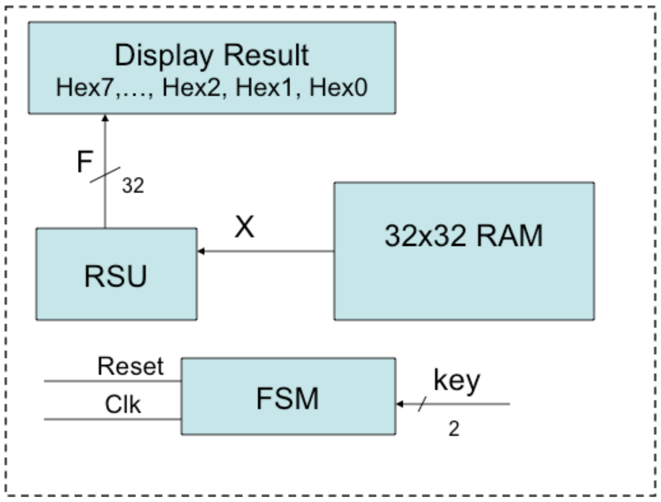
and to compute the square root of a c one could use f(x) = $1/x^2$ – c and the iteration

$$x_{i+1} = \frac{x_i}{2}\left(3 - cx_i^2\right)$$

Note that this results in $\frac{1}{\sqrt{c}}$. You can take the reciprocal of this to get $\sqrt{c}$

The computation of the reciprocal of a numerical value is useful in many algorithms. In particular it can be used to implement floating point division very efficiently - N/D can be implemented by multiplying N by the reciprocal of D. Square root is widely used in many applications in signal processing.

What should you do?



1. Read the following references to understand the algorithm and get an idea of the architecture – (the references are available on SmartSite)
    a. **An Efficient Hardware Implementation for a Reciprocal Unit**
    Andreas Habegger, Andreas Stahel, Josef Goette, and Marcel Jacomet
    b. **Simple Seed Architectures for Reciprocal and Square Root Reciprocal**, Milos Ercegovac, Jean-Michel Muller and Arnaud Tisserand
2. MANDATORY PRELAB: Write a C program to test and validate your understanding of the underlying algorithms.
3. The top-level architecture of your design should be as shown in the figure above.
4. The inputs and outputs of your functional unit should be 32 bits numbers in the IEEE floating-point representation.
5. You should use an on-chip RAM (as in Lab 4) to store the input vector X. You can assume that location 0 of the RAM contains the initial value (seed) and locations 1 to 10 contain the input data set.
6. Your design should be **programmable**, which means, it receives an additional input (denoted by key) that specifies the operation to perform on the input data set. If key is "01" the functional unit should compute the reciprocal of the numbers x[1], x[2], …. x[10] and display the result to the seven-segment display, and if the input is "10" the unit should compute 1/sqrt(x[i]) and if the key input is "11" it should compute sqrt(x[i]) by first computing the 1/sqrt(x[i]) and then taking the reciprocal of the result.
7. Output should be displayed as hexadecimal number on Hex7, Hex6, … Hex2, Hex1 and Hex0. Each result should be displayed for at least 10 seconds.
8. Assume there is a reset switch, which can also serve as the start signal, when de-asserted.
9. Your computation can take any number of clock cycles, i.e. it is not necessary to finish all your computation in one cycle. In fact, it is not possible to do that. However, you should optimize the clock frequency so that the total time to perform the operation should be minimized.
10. You can use * and + operators to infer adders and multipliers.
11. Test your design with a variety of inputs and different initial values or seeds.
12. A fraction of the final grade for the project will depend on how optimized your design is, both in terms of number of resources (Logic elements) and the clock frequency.
13. After completing the project you can improve your course grade by extending this project in interesting ways for extra credit. You can use optimized versions of the adders and multipliers say from Lab 3 and Lab 5 for extra credit if that happens to be on the critical path and you think it will improve the performance. However

everything including extra credit should be finished by June 7, 2014.

Execution of the Project

You will work in a group of 2 for this project.
This project will have **4** milestones.

Milestone	**Points**	**Deadline**
Design and Functional Simulation of **Reciprocal Computation**	100	May 17, 2014
Download Reciprocal Unit on DE2 board and Verify	50	May 24, 2014
Implementation of **Square Root Unit**	100	May 31, 2014
Implementation of the RSU	50	June 7, 2014

Extra Credit Ideas:

1. Pipeline the design to improve the clock frequency of the circuit.
2. Optimize the area of the design by sharing the datapath units between the reciprocal and square root units
3. Redesign either the reciprocal or square root unit using a different algorithm to improve the performance or area.