ECE 510

Synthesis and Verification of Finite State Machines

Final Project Report

Synchronous Implementation of 8-bit Square Root Computer

by

Liong-Huat Lim (Leon)

with supervision from

Professor Alan Mishchenko

Problem Discussion

Synchronous Square Root Computing Machine

Background

Computing square root of a real number is a fundamental mathematical operation. Before the invention of calculator and computer, people would use method such as bisection and Newtonian algorithm to manually calculate square root. Today, a few buttons pushed, we get the result in no time. In this paper we will discuss the implementation of a synchronous square root machine using a simple algorithm. In addition, we will compare the result of different implementations.

Square Root Algorithm

1) Split the given number into groups, each containing two digits, starting from the least significant bit. (In our example, these groups will be "89", "34", and "3".)

2) Find the largest number, which when multiplied by itself does not exceed the number, created by the first two-digit group. Write down this number as the first digit of the result. (In our case, the first two-digit number is "3" and the number, which when multiplied by itself does not exceed "3", is "1".)

3) Subtract the square of this number from the number given by the first two digits and concatenate this number with the next two digits from the initial number. (In our case, we have to subtract "1 x 1 = 1" from "3" and then concatenate it with "34". As a result, we get 234.)

4) Multiply by two (for binary numbers, it will be just a shift!) the number, which constitutes the result computed so far, and concatenate it with the largest digit, which satisfies the following condition: the product of this digit and the number resulting from concatenation does not exceed the number received at the previous step. (In our case, the result computed so far is "1"; multiplied by two it will be "2", and the digit to be selected is "8" because "28 x 8 = 224 < 234", while "29 x 9 = 261 > 234".)

5) Write down the digit, which we have just found (in our case, "8") as the next digit of the result.

6) Subtract 224 from 234 and concatenate it with the next group of numbers. In our case it is "1089".

7) If the resulting number (in our case it is "1089") is equal to 0 (the square root is extracted exactly), stop. Else go to step 4.

If the square root cannot be extracted exactly, algorithm should be stopped at step 7 when we have found enough digits belonging to the floating-point part of the result.

To implement this machine we will need shift registers, a subtractor, a comparator, and a few registers to hold intermediate values. Of course, we will also need a FSM to controller the flow of the data.

Implementation

There are several major modules in the design: ResultReg, Subtractor, Comparator, IntReg, InputReg, FSM and a 3-bit counter. Please refer to Appendix for the block diagram. The FSM is clock at the rising edge of the clock whereas the registers are clock on the falling edge. This is done to provide maximum setup and hold time for the changing signals.

ResultReg

The ResultReg is a 8-bit shift register. However, only 6 of the bits are used to store result since only 4 is needed for the whole number result, the other 2 bits are used to store the binary point part of the result. There are three control signal going into this module: SelOne, LdResultReg and Concat1. SelOne simply selects whether a ‘1’ or ‘0’ is to be loaded into the shift register. LdResultReg is the Clock Enable that allows data to be loaded into the register. Concat1 is a special control signal that concatenate a ‘1’ to the left shifted (multiply by 2) result. The result register is not really shifted to multiply by 2. Instead, I take bit [5..0], concatenate it with “01” and send the resulting vector to comparator and subtractor.

IntReg

This is the intermediate registers that holds the remainder of the subtractor. It has two control signals: LdIntReg, SelAdder. LdIntReg is the clock enable signal that allows data to be loaded into the registers. SelAdder selects whether to load subtractor output or previvious IntReg output into the registers.

Substractor

This is a 8-bit subtractor. It is purely combinational logic.

Comparator

This is a 8-bit comparator. Its output Smaller is HIGH when input B is smaller than input B. This is also a combinatorial logic.

InputReg

This is a 8-bit shift register that can shift two position in one clocking period. It has two control signal: LdIO and ShiftData. LdIO simply loads the input data from external system during initial state. ShiftData allows shift register to shift two position at a time.

FSM

The FSM has 9 states. Please refer to Appendix for STG. The outputs are purely Moore type implementation. The inputs to the FSM are Reset, Start, IsZero, Smaller and Ready. Reset will put the FSM into the initial state and Start will kick-start the computation process. IsZero tells the FSM whether the most significant two bits are zero. If it is, it simply goes into a wait state until they are not zero. This is done to save a few cycles when the input contains leading zeros. Smaller tells the FSM whether the number found satisfy the requirement mentioned in step 4 of the algorithm. Ready tells the FSM that it has computed the result. Ready comes from a counter.

Results and Analysis

In this section we will compare the synthesis result of implementation of the SquareRoot machine.

Synchronous Implementations

OneHot Encoding

```
Primitive reference count:
Number of flip-flops 40
CARRY       11
DFFE        40
INV          1
LUT         44
LUT_CARRY    11

Clocks:
-------
                        Required  Estimated
Period   Rise    Fall   Freq      Freq      Signal
(ns)     (ns)    (ns)   (MHz)     (MHz)
.............................................................
 40       0      20      25.00    -1.00     default
 -1      -1      -1   -1000.00    33.47      Clock
```

Binary Encoding

```
Primitive reference count:
Number of flip-flops 35
CARRY       11
DFFE        35
INV          1
LUT         53
LUT_CARRY    11
```

```
Clocks:
-------
                     Required  Estimated
Period   Rise    Fall    Freq      Freq       Signal
(ns)    (ns)    (ns)    (MHz)     (MHz)
..............................................................
 40       0      20      25.00     -1.00     default
 -1      -1      -1      25.00     26.23     Clock
```

Asynchronous Implementation

```
Primitive reference count:
Number of flip-flops  0
CARRY                 19
INV                   6
LUT                   26
LUT_CARRY                      17


Timing Path Groups:
------------------
                          Required   Estimated
                          Delay      Delay
From         To             (ns)       (ns)
...........................................................
(I)          (O)            40.00      57.12
```

Conclusions

The implementation of the square root algorithm using FSM and synchronous logic was successful. I was able to download the design into an Altera FPGA evaluation board and demonstrate the design to the class. One way to improve the circuitry is to combine the comparator with the subtractor. The new subtractor must have an underflow indicator. This signal can be used by FSM to determine which number is greater.

Appendix