Part III

In this part, you will have to build a complete datapath and corresponding state machine. We recommend that you still have two separate modules for the datapath and the control just to help you understand the separation of what is in a datapath and what is in a controller. In Part II, the control logic just generated control signals that were inputs to the datapath. In this part, you will see that there is a signal that needs to go from the datapath into the controller so make sure you account for it accordingly.

Division in hardware is the most complex of the four basic operations. Add, subtract and multiply are much easier to build in hardware. For this part, you will be designing a 4-bit restoring divider using a finite state machine.

Figure 5 shows an example of how the restoring divider works. This mimics what you do when you do long division by hand. In this specific example, number 7 (*Dividend*) is divided by number 3 (*Divisor*). The restoring divider starts with *Register A* set to *0*. The *Dividend* is shifted left and the bit shifted out of the left most bit of the *Dividend* (called the most significant bit or MSB) is shifted into the least significant bit (LSB) of *Register A* as shown in Figure 6.

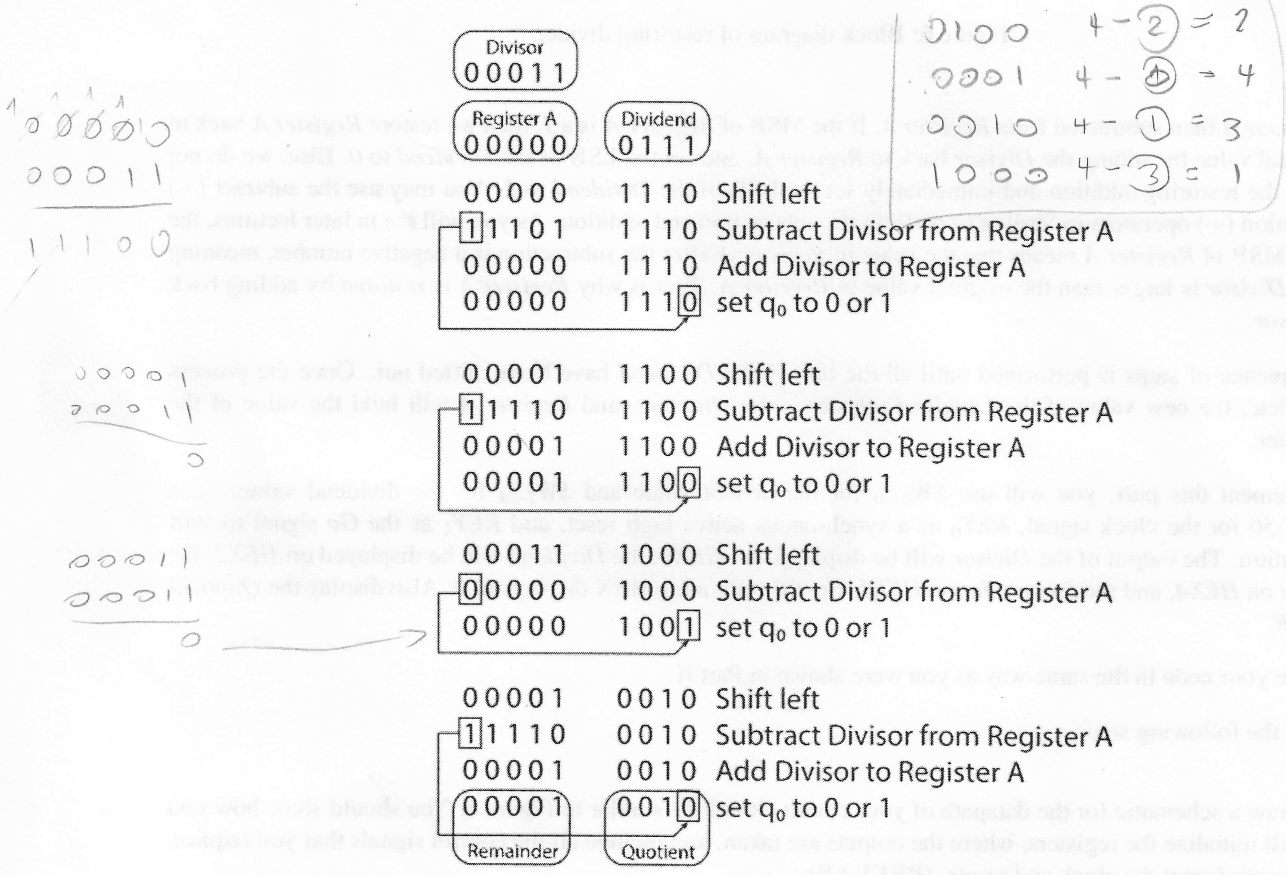

Figure 5: An example showing how the restoring divider works.

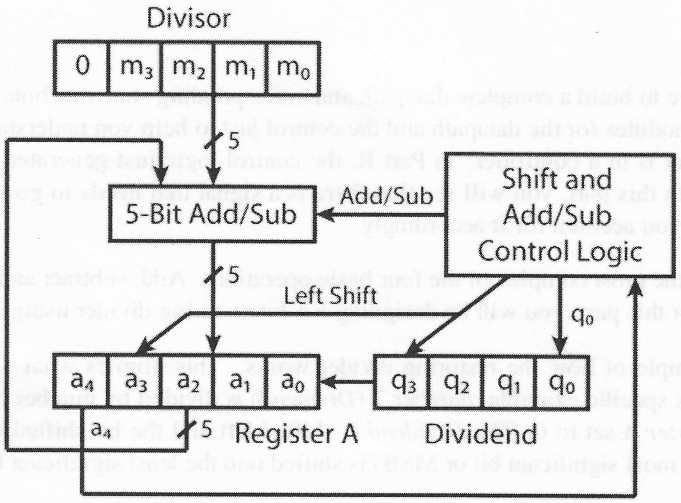

Figure 6: Block diagram of restoring divider.

The *Divisor* is then subtracted from *Register A*. If the MSB of *Register A* is a *1*, then we restore *Register A* back to its original value by adding the *Divisor* back to *Register A*, and set the LSB of the *Dividend* to *0*. Else, we do not perform the restoring addition and immediately set the LSB of the *Dividend* to *1*. You may use the subtract (−) and addition (+) operators in Verilog to perform the subtraction and addition. As you will see in later lectures, the *1* in the MSB of *Register A* means that the value in *Register A* after the subtraction is a negative number, meaning that the *Divisor* is larger than the original value in *Register A*. That is why *Register A* is *restored* by adding back the *Divisor*.

This sequence of steps is performed until all the bits of the *Dividend* have been shifted out. Once the process is complete, the new value of the *Dividend* register is the *Quotient*, and *Register A* will hold the value of the *Remainder*.

To implement this part, you will use SW_{3-0} for the divisor value and SW_{7-4} for the dividend value. Use *CLOCK_50* for the clock signal, KEY_0 as a synchronous active high reset, and KEY_1 as the *Go* signal to start computation. The output of the *Divisor* will be displayed on *HEX0*, the *Dividend* will be displayed on *HEX2*, the *Quotient* on *HEX4*, and the *Remainder* on *HEX5*. Set the remaining HEX displays to *0*. Also display the *Quotient* on *LEDR*.

Structure your code in the same way as you were shown in Part II.

Perform the following steps.

1. Draw a schematic for the datapath of your circuit. It will be similar to Figure 6. You should show how you will initialize the registers, where the outputs are taken, and include all the control signals that you require. Do not forget the clock and resets. **(PRELAB)**
2. Draw the state diagram to control your datapath. Check it by hand simulating the example shown in Figure 5. Hand simulation just means to work through the steps using your schematic and state diagram to check whether you can do the required operations before going through the effort of setting up the simulator. This may not catch all bugs, but it is a good step to make sure you have a design that has a chance of working. **(PRELAB)**
3. Draw the schematic for your controller module. **(PRELAB)**
4. Draw the top-level schematic showing how the datapath and controller are connected as well as the inputs and outputs to your top-level circuit. **(PRELAB)**
5. Write the Verilog code that realizes your circuit. **(PRELAB)**

6. Simulate your circuit with ModelSim for a variety of input settings, ensuring the output waveforms are correct. Start with Figure 5 as an example because it shows you all the steps with the values that should be in the registers at each step. (PRELAB)

7. After you are satisfied with your simulations, download and test the functionality of the circuit on the FPGA board.