Lab 5: Finite State Machines + Datapaths (Nth Factorial Calculator)

EEL 4712 – Fall 2019

Objective:

The objective of this lab is to use a finite state machine integrated with a datapath to calculate the nth factorial, using several different VHDL models.

Required tools and parts:

Quartus2 software package, ModelSim-Altera Starter Edition, Altera DE10 board.

Pre-lab requirements:

1. Study the following pseudo-code to make sure you understand the basic algorithm for calculating n factorial. The code has 1 input (*n*), and one output (*output*). There is also a control input called *go* and a control output called *done*.

```
// inputs: go, n
// outputs: output, done

// reset values (add any others that you might need)

output = 1;
done = 0;


while(1){
  // wait for go to start circuit
  while(go == 0);
  done = 0;

  //store your input into a register
  tempFact = 1;


  for(regN = n; regN >= 1; regN--){
    tempFact = tempFact * regN;
  }

  //assign "output" and assert done
  output = tempFact;
  done = 1;

  // make sure go has been cleared before starting the program again
  while(go == 1);
}
```

FSMD

2. Using the provided entity (factorial.vhd), create a custom circuit that implements the nth factorial algorithm by using the 1-process FSMD model. **This specification must appear in the FSMD architecture of the factorial entity.** After being reset, the circuit should wait until go becomes 1 (active high), at which point the main factorial algorithm should be performed for the given n input. Upon completion, done should be asserted (active high). **Done should remain asserted until the application is started again, which is represented by a 0 on the go signal followed by a 1. The circuit shouldn't continuously execute if go is left at 1.**

   Use the provided testbench (factorial_tb.vhd) to test your architecture. Note that the testbench only tests a single architecture. Therefore, make sure the following line:

```
UUT : entity work.factorial(FSMD)
```

specifies the FSMD architecture (as shown here). For the other parts of the lab, you will specify a different architecture

FSM+D1

3. In this step, you will implement a custom circuit that implements the factorial algorithm by using the datapath shown below:

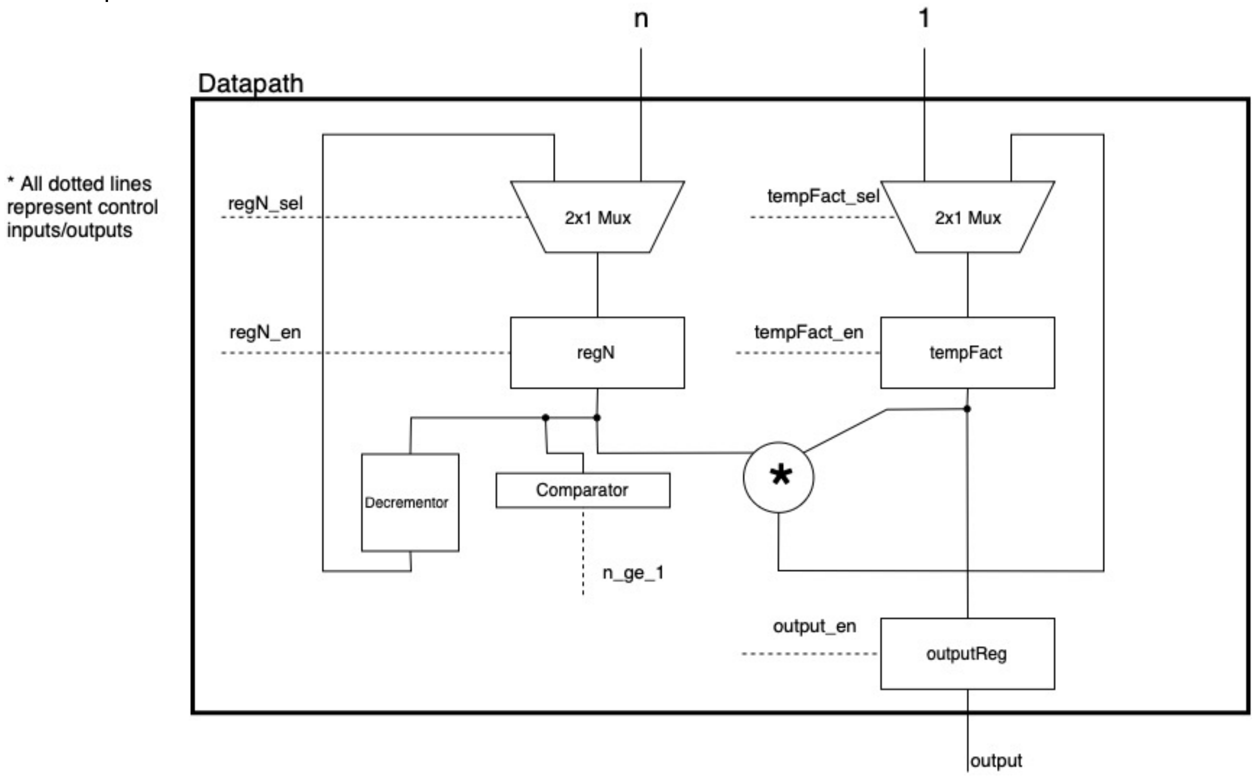

Figure 1. FSM+D1

Implement the datapath by creating an entity datapath1 (store it in datapath1.vhd). You must use a structural description that instantiates all of the components shown. Therefore, you will also need a register entity, a multiplier entity, a 2x1 mux entity, a decrementor entity, and a comparator entity. The register entity must have an enable input that allows/prevents data from being stored. The comparator entity must have a greater than or equal to output, which connects to the n_ge_1 signal. You are free to implement these entities however you like, as long as they have these basic capabilities, and as long as each entity uses a generic for the width.

Next, implement a controller entity called ctrl1 (store it in ctrl1.vhd) that uses the control signals for the illustrated datapath to execute the factorial algorithm. Feel free to use either the 1-process or 2-process FSM model (I **highly** recommend the 2-process model).

For the provided factorial entity implement the structural architecture (FSM_D1) that connects the controller to the datapath. You must use the FSM_D1 architecture.

Use the provided testbench (factorial_tb.vhd) to test your architecture. Note that the testbench only tests a single architecture. Therefore, make sure you use the following line for the factorial instantiation:

```
UUT : entity work.factorial(FSM_D1)
```

Top Level

4. Create your own top-level entity top_level, stored in top_level.vhd, that instantiates one of the factorial entities with an input width of 16 bits and an output width of 16 bits. Because we're limited to the number of switches available on the board, only use the bottom 3 switches (SW3 - SW0) for your input signal and set the remaining bits of the total 16 to zero. Go and reset are mapped onto buttons (you will need to invert the buttons for both), the output is mapped onto the 4 7-segment LEDs (LED3-LED0 where each LED gets 4 bits of the output), and the done signal is mapped onto the least significant decimal point (remember to invert this signal as well). Make sure to add the 7-segment decoder code to your project. To select a particular architecture for the factorial component, you can either use a configuration or can specify the architecture explicitly:

```
U_FACTORIAL : entity work.factorial(FSMD)
```

   See the provided testbench and the top_level entity from lab 3 for examples.

Extra Credit

5. Create a new FSMD (not an FSM+D) architecture for the factorial entity that uses a 2-process model. Note that I have not shown how to do this. The 2-process FSMD is trickier than the 2-process FSM model and the 1-process FSMD model. You will likely run into issues that will test your understanding of VHDL. However, unlike the 1-process FSMD, the 2-process FSMD model has the advantage of not requiring registers on all outputs. Be sure to inform your TA if you are able to get this working.

   Implement the new FSMD in the FSMD2 architecture for the provided factorial entity.

   Use the provided testbench (factorial_tb.vhd) to test your architecture. Note that the testbench only tests a single architecture. Therefore, make sure you use the following line for the factorial instantiation:

```
UUT : entity work.factorial(FSMD2)
```

Turn in all VHDL. For Step 4, create and submit a diagram illustrating the structure of your revised datapath. There must be only one subtractor.

In-lab procedure (do as much as possible ahead of time):

1. Using Quartus, assign pins to each of the top_level.vhd inputs/outputs such that the signals are connected to the appropriate locations on the board.
2. Download your design to the board, and test it for different inputs and outputs. Demonstrate the correct functionality for the TA. Make note of the area requirements for the current architecture.
3. Demonstrate the other architectures (FSMD, FSM_D1, FSM_D2, FSMD2) to the TA, showing how the area changes. Alternatively, if you do not finish this step, see the lab report section. Note that FSMD2 is extra credit.
4. **Be prepared to answer simple questions or to make simple extensions that your TA may request. If you have done the pre-lab exercises, these questions should not be difficult.**

Lab report: (Pre-Lab part only)

Be sure to submit screenshots of your Modelsim waveforms illustrating correct functionality for each of the architectures. Include full window screenshots of Modelsim and compile them into a pdf or word doc with titles for each of the screenshots. Final submission should include all VHDL files (including created testbenches) and your lab report in one zipped folder. Please do NOT include Quartus projects in submissions.

Lab report: (In-lab part only)

If you had any problems with portions of the lab that could not be resolved during lab, please discuss them along with possible justifications and solutions. Also, if you did not demo all implementations for your TA, create a table that shows how the area requirements differ for each implementation. If you had no problems and demoed all parts during lab, this report is not necessary.

If needed, turn the lab report in on e-learning, if explanation is needed for partial credit. Make sure to turn it in to the “lab” section and not the “pre-lab” section.