Lab 5: Finite State Machines + Datapaths (N^{th} Factorial Calculator)

EEL 4712 – Fall 2021

Objective:

The objective of this lab is to implement the n^{th} factorial with a datapath. **Consult all the VHDL files submitted as part of the lab for entity definitions**.

Required tools and parts:

Quartus2 software package, ModelSim-Altera Starter Edition, Altera DE10-Lite 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 the circuit
  while(go == 0);
  done = 0;

  //Load your input into a register
  tempFact = 1;
  regN = n ;

  //Compute the factorial
  while(regN>=1){
    tempFact = tempFact * regN;
    regN--;
  }

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

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

Figure 1. Factorial pseudo-code

Datapath Design

2. Using the provided entity (factorial.vhd), create a custom circuit that implements the n^{th} 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

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

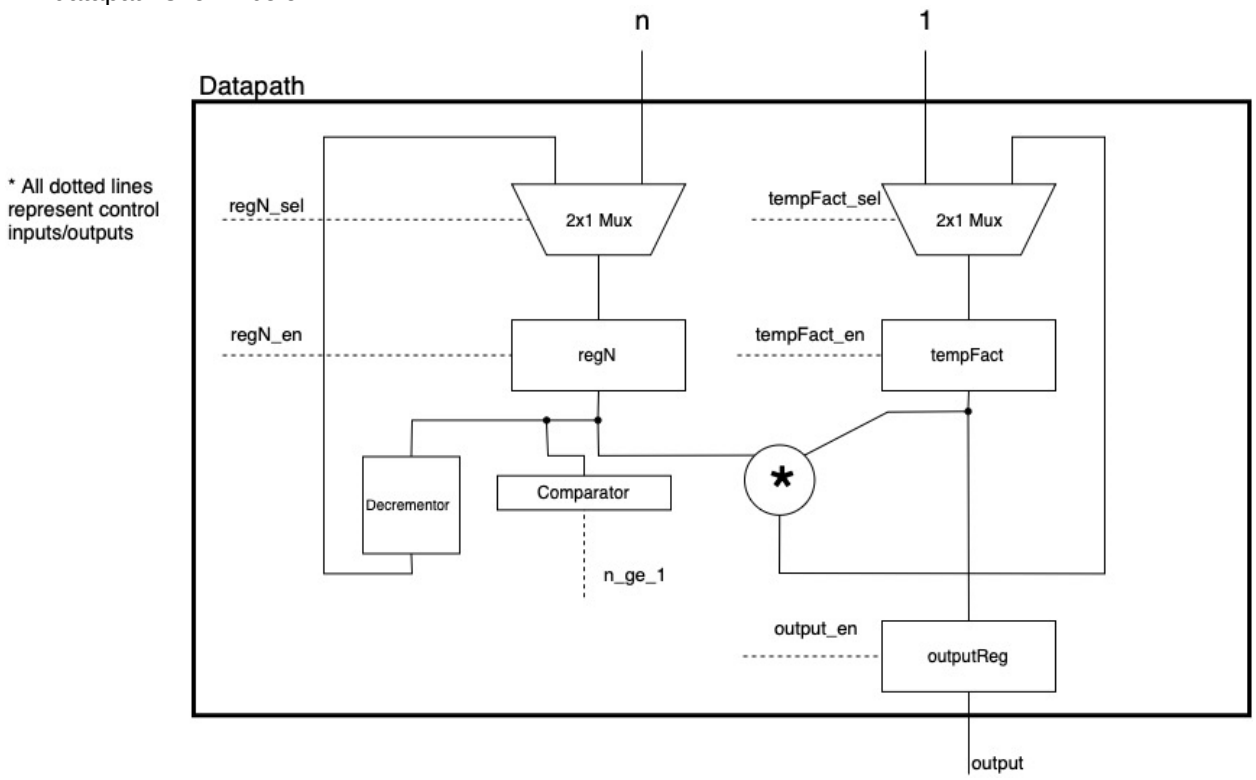

Figure 2. Factorial datapth

Implement the datapath by editing the file **datapath1.vhd**. You must use a structural description that instantiates all of the components shown in Figure 2. 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)
```

Test on the MAX10 board

Create a top-level entity (in a file **top_level.vhd**). Use the structural description to instantiate your datapath implementation of the factorial function. Because we are limited to the number of switches available on the board, only use the bottom 3 switches (SW2 - SW0) for your input signal n and set the remaining bits of the total 16 to zero. Use the switches/buttons of your choice to map the go and reset inputs. The done signal is mapped onto LEDR0. The result of the factorial calculation is displayed on the 7-segment leds. You can implement a 7-segment controller to display the results or use the one provided with the lab.

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

1. Using Quartus, show the pin assignment of the top_level.vhd inputs/outputs such to the TA.
2. Download your design to the board, and test it for different inputs and outputs. Demonstrate the correct functionality to the TA. Make note of the area requirements for the current architecture.
3. **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.