Lab 5: Finite State Machines + Datapaths (GCD Calculator)

EEL 4712 – Fall 2014

Objective:

The objective of this lab is to use a finite-state-machine controller integrated with a datapath to calculate the greatest common divisor (GCD) of two numbers, using two different designs.

Required tools and parts:

Quartus2, ModelSim-Altera Starter Edition, Altera DE0 board

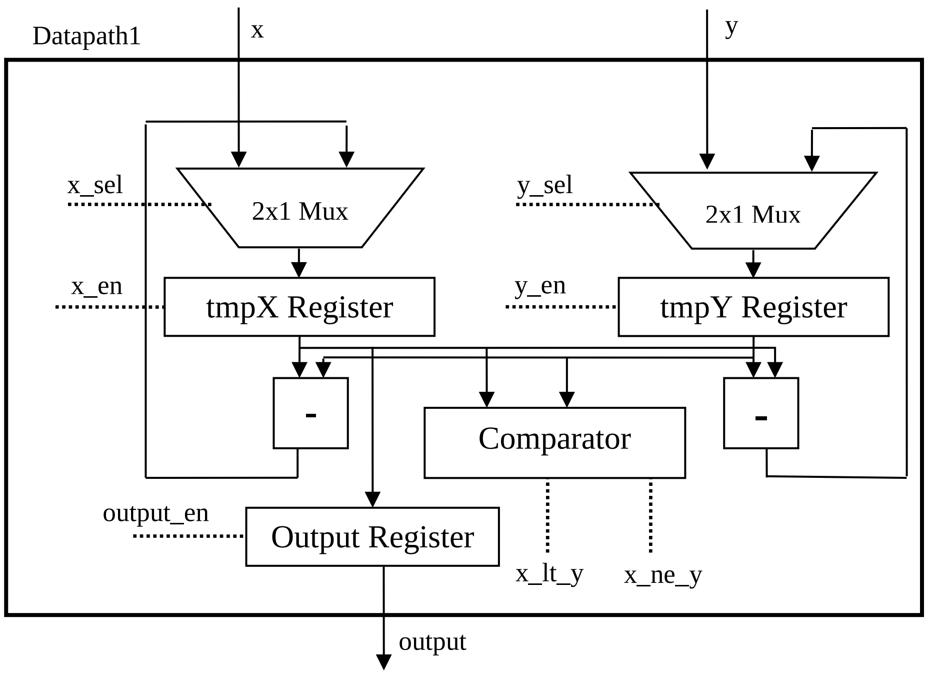
Pre-lab requirements:

Study the following pseudo-code to make sure you understand the basic algorithm for calculating the GCD of two numbers. The code has 2 inputs (*X* and *Y*), and one output (*output*).

```
tmpX = X;
tmpY = Y;
while (tmpX != tmpY) {
    if (tmpX < tmpY)
          tmpY = tmpY-tmpX;
    else
          tmpX = tmpX-tmpY;
}
output = tmpX;
```

1. **Design 1 (FSM+D1)**: In this design, you will implement a custom circuit that implements the GCD algorithm by using the datapath shown below:

*All dotted lines represent control inputs/outputs



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 2x1 mux entity, a subtractor 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 less than output, which connects to the x_lt_y signal, and a not equal output, which connects to the x_ne_y 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.

Using a 2-process model for the FSM controller, implement a controller entity called ctrl1 (store it in ctrl1.vhd) that uses the control signals for the illustrated datapath1 to execute the GCD algorithm.

For the provided gcd entity implement the structural architecture (FSM_D1) that connects the controller ctrl1 to datapath1. You must use the FSM_D1 architecture.

Use the provided testbench (gcd_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 gcd instantiation:

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

2. **Design 2 (FSM+D2):** In this design, you will create a **different** datapath (datapath2) for the GCD algorithm that only uses a **single subtractor**. Add any components and/or control signals that are necessary. Store it the entity datapath2 in datapath2.vhd. You must use a structural architecture.

   Next, implement a revised controller entity called ctrl2 (store it in ctrl2.vhd) that is based on the revised datapath. Add any control I/O that is required.

   For the provided gcd entity implement the structural architecture (FSM_D2) that connects the new controller to the new datapath. You must use the FSM_D2 architecture.

   Use the same provided testbench (gcd_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 gcd instantiation:

   ```
   UUT : entity work.gcd(FSM_D2)
   ```

3. **Top Level:** Create your own top-level entity top_level, stored in top_level.vhd, that instantiates one of the gcd entities (with a width of 8 bits. Top_level has four inputs: ***reset***, ***x, y,*** and ***go*** and two outputs: ***output*** and ***done***. After being reset, the circuit should wait until go becomes 1 (active high), at which point the GCD algorithm should be performed for the given x and y inputs. 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. In other words the circuit shouldn't continuously execute if go is left at 1.**

   The x and y inputs are mapped onto the dip switches, the go signal is mapped onto a button, the output is mapped onto the two 7-segment LEDs (each LED gets 4 bits of the output), and the done signal is mapped onto the decimal point. Make sure to add the 7-segment decoder code to your project. To select a particular architecture for the GCD component, specify the architecture explicitly; for example, to use FSM_D1:

   ```
   U_GCD : entity work.gcd(FSM_D1)
   ```

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

Turn in e-Learning (and print for TA) all VHDL. Also, for Design 2, create and submit a diagram illustrating the structure of your revised datapath2.

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 1 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 Design 2 to the TA, showing how the area changes.
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.**