EE209 Lab – Change We Can Believe In

1 Introduction

In this lab you will complete the control unit and datapath for a vending machine change collector and dispenser. This lab will build on the vending machine exercise from lecture. We will provide you several datapath components. You must combine them and create the state machine to control the operation. In addition, you will write a testbench for the design. **You must work individually on this lab and not in groups.**

2 What you will learn

This lab uses many of the datapath and control unit concepts you have learned in this class. You will use your datapath to make change after a user has bought a drink. You will also implement the state machine that provides the necessary control signals for the datapath. Many possible implementations exist and will be accepted provided it works as specified.

3 Background Information and Notes

1. Overview

The block diagram of the vending machine design is shown below and inputs and outputs are described in the following table.

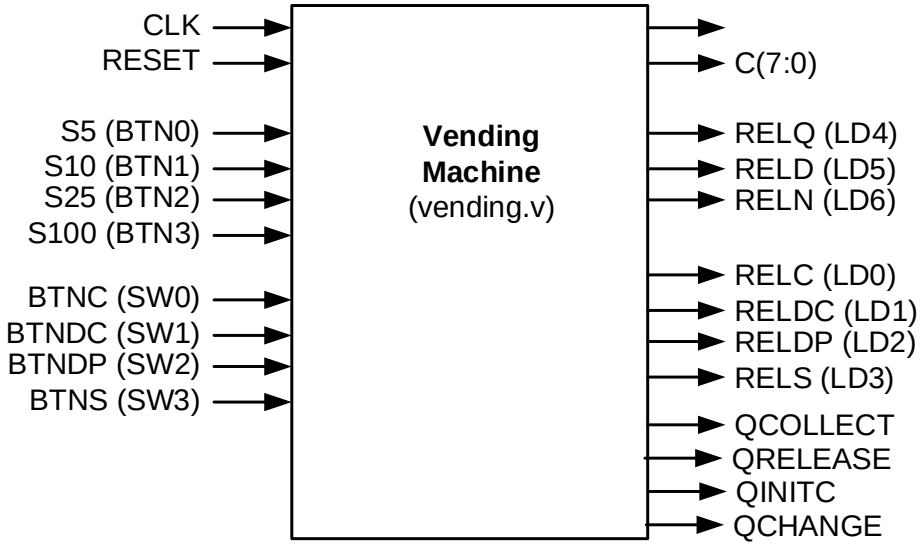

Figure 1 - Block Diagram of Vending Machine Design

Inputs	
CLK & RESET	Note that RESET is active high
S5,S10,S25,S100	Sensor inputs indicating when a nickel, dime, quarter, or dollar bill is inserted. Will be high for 1 clock period each time a coin or bill is entered.
BTNC, BTNDC, BTNDP, BTNS	Button inputs for drink selection. Any number can be active at a time. Each BTN*x* input corresponds to a REL*x* output indicating that the drink should be released.
Outputs	
M(7:0)	Current amount of money collected. Should be the output of the M register. M is in units of cents (pennies)
C(7:0)	Current amount of change needing to be made. After a user has entered enough money and selected their drink costing $1, C should be loaded with M – 100 and then reduced appropriately as each quarter, dime, and/or nickel are released.
RELQ, RELD, RELN	Release outputs for a quarter, dime, or nickel. Should be high for 1 clock cycle per coin released.
RELC, RELDC, RELDP, RELS	Release outputs for the drinks. Note: In our design these will be Mealy outputs and thus could be high for only part of a clock signal, which is fine for our purposes.
QCOLLECT, QRELEASE, QINITC, QCHANGE	States will be displayed on the left-most 7-segment display. We will also use these bits to select either the M value or C value to be displayed on the three right 7-Segment displays.

Table 1- I/O Description

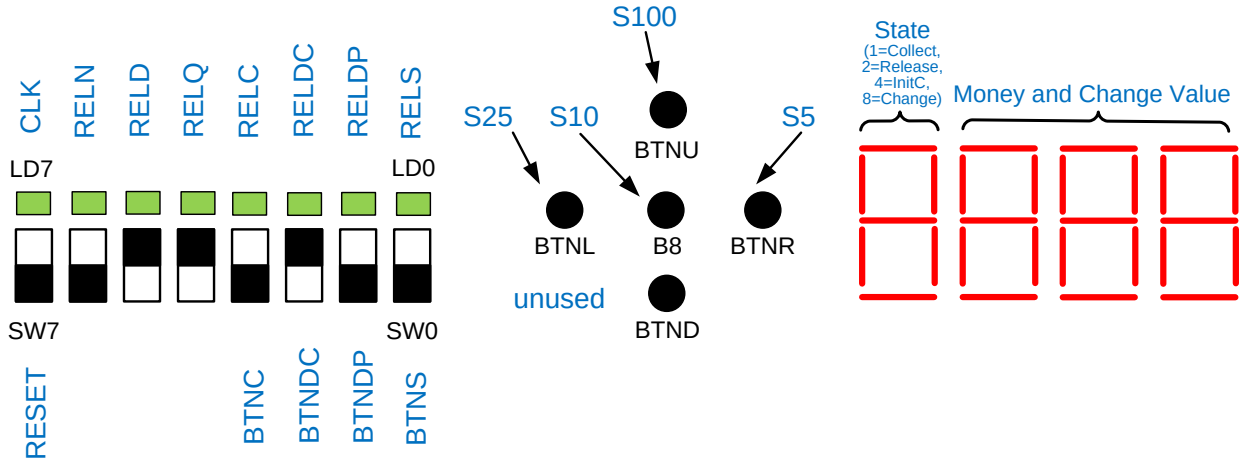

Figure 2 – Mapping of inputs and outputs to FPGA LEDS, buttons, switches, and 7-segment displays.

2. State Machine Control

The state machine control for your vending machine will be a modification of that used in the lecture example. Because we now have to make change as well, we could imagine adding the necessary states for this task all in one large state machine. We will add a state INITC where we can load the C register with the correct amount of change we will need to produce (i.e. M-100). Then we will move to a CHANGE state where we release either Quarters, Dimes, or Nickels based on the value of C. We are done when C equals 0.

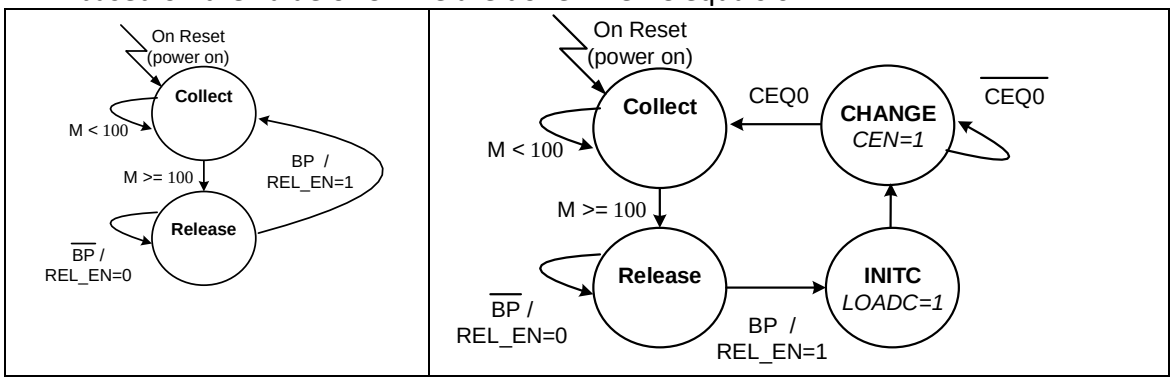

Figure 3 - State Machine from class lecture; Composition of state machines for this lab; one state machine is used during money collection and the other during change dispensing.

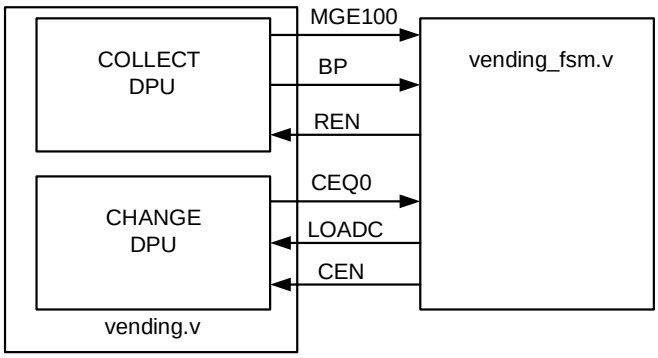

Figure 4 – Interconnection of DPU and CU for collection and change-making.

The inputs and outputs of the state machines are described below:

Signal	Description
MGE100	Comparator output if M(7:0) ≥ 100_{10}
BP	Button press (OR'ing of all the drink buttons)
REN	Release Enable indicating a drink may now be released because enough money has been input. Should be generated by the FSM
LOADC	Signal telling the C(7:0) register to be loaded with M-100. Should be generated by the FSM.
CEQ0	Comparator output if C(7:0) [current change left] is zero.
CEN	Change Enable indicating a quarter, dime, or nickel should be released in the current clock if C > 0. Should be generated by the FSM

Table 2 - State Machine Input / Output Signal Descriptions

3. Datapath Units

The datapath for money collection was designed in class. A datapath is also needed to make change. This datapath's main goal is to maintain the current amount of change that needs to be dispensed as well as generating the quarter, dime, and nickel release signals. A description of the datapath and its outputs is shown below.

```
if(LOADC = 1)
  C*[7:0] = M[7:0] – 100₁₀
else if (CEN = 1  AND  C ≥ 25)
  RELQ = 1
  C* = C – 25₁₀
else if (CEN = 1  AND  C ≥ 10  AND  C < 25)
  RELD = 1
  C* = C – 10₁₀
else if (CEN = 1  AND  C ≥ 5  AND  C < 10)
  RELN = 1
  C* = C – 5
else      [ C must be 0]
  do nothing (i.e. C = C)
```

Your datapath can be designed in anyway you see fit. It will likely look similar to the collection datapath with some differences (i.e. you are subtracting rather than adding, you have to load the C register with M-100, etc.). Identify the various operations being performed above and use those datapath components. As a design technique, write out a table of all the cases (from the if statements) of the inputs you need to add/subtract. When there are difference choices for an value you will need a mux. Then use the conditions in the if statements to derive the select bit logic. Do this by identifying the binary signals (i.e. LOADC, CEN, C>=25, etc.) and using them to create a truth table for the desired select bits. Most of the change datapath will come directly from the description above so study it carefully.

4. Verilog Notes and Project Specific Components

We have provided you many components that you can use. Take a few minutes and open each file and look at it. Understand its function. **Note**: In these files we are using high-level Verilog descriptions. Rather than instantiating basic gates and lower-level components we use a "behavioral" description which Xilinx can then take and figure out the exact gates. You will learn more about this description syntax in the next CENG course. But it is good experience to see some of it now.

moneymux.v: We have generated a mux that takes a 2-bit select number and produces the 8-bit binary constants: 5_{10} (when S=00), 10_{10} (when S=01), 25_{10} (when S=10), and 100_{10} (when S=11).

mux21_8bit.v: We have generated this component for you to use if desired. It implements a 2-to-1, 8-bit wide mux.

comp8.v: We provide an 8-bit unsigned comparator that will provide LT, GT, and EQ outputs.

reg8e.v: We provide an 8-bit register with enable to store the M and C values.

adder8.v: We provide an 8-bit adder

pe4_2.v: We provide a 4-to-2 priority encoder with active-hi valid output

vending_fsm.v: You will need to implement the control state machine in this file. You may use an encoded- or one-hot-state machine design approach.

Multi-bit Constants: In Verilog to generate a multi-bit constant we use the syntax such as `8'b001110001`. The first number is how many bits we will write, then an apostrophe (or tick) followed by the character, b, and then the actual 8-bit value. If we wanted to write a 4-bit value we could write: `4'b0110`. If we wanted to write a 6-bit value we could write `6'b100011`.

Verilog `assign` statement: At this point in the class you may be tired of instantiating separate AND, OR, NOT, NAND, XOR, etc. gates one by one. In Verilog there is a slightly quicker way to describe this logic. It is with an assign statement. By using it you can describe several physical gates with a single statement. The syntax is as follows:

```
assign output_wire = input1_wire OP input2_wire OP … inputN_wire;
```

Example 1:

```
assign z = (x & y) | (x ^ z);
```

is equivalent to:

```
and(t1, x, y);
xor(t2, x, z);
or(z, t1, t2);
```

A mux could be described in a single line as:

```
assign Y = (~S & I0) | (S & I1);
```

Valid operators are: **&** = and, **|** = or, **^** = xor, **~** = not.

The output must be declared as a **wire**. You may feel free to use this approach rather than instantiating individual gates when describing your logic in this lab.

4 Prelab

None.

5 Procedure

You should complete the control and datapath unit for the vending machine. You should then create a testbench to test your circuit. Once you are satisfied with your testing process, you can program your FPGA board with the design.

1. Download the `vending.zip` project file. Extract the files to a folder.

2. The vending machine project has a completed top-level file (`vending_top.v`) that will interface your design to the switches, buttons, and displays on the FPGA board as long as you produce a working design that matches the I/O shown in the block diagram earlier in this lab.

3. You will need to add your datapath design in the `vending.v` file. Some portion of the collection datapath is complete.

4. Implement the state machine in the `vending_fsm.v` file and output the given signals in the skeleton that is provided in that file. It is recommended you use a one-hot approach.

5. Check your design for syntax or circuit errors by synthesizing it. Make sure no errors are present and fix any warnings that you can.

6. Switch to Simulation view and open the provided test bench. In the initial block at the bottom of the file add input stimulus that will cause a full sequence of the vending machine operation (i.e. turn on sensors at various times to add up to over 100, cause a drink button to be pressed, then give time for the change to be made). Generally, you should cause input signals to turn on for a full cycle. To do this, assign a signal to 1, wait a clock cycle (#n;), and then turn the signal of and start the next. You can assign several signals in the same clock cycle as needed.

7. In the processes pane, click on properties of "Simulate Behavioral Model" and set the "Simulation Run Time" to match the testbench length. Simulate your design until you are satisfied it is working. (Note: Changing the M and C values to Radix..Unsigned will show the positive decimal value of the 8-bit numbers and make it easier to ensure your circuit is working). **We would encourage you to try a few different money collection sequences that will exercise your change dispenser. Don't just use a single sequence.**

8. Synthesize (if you haven't already) and implement your design checking for errors. Set the Startup Clock to "JTAG Clock" under the properties of "Generate Programming File". Then generate the programming file.

9. Program your FPGA using the Adept tool and ensure it functions correctly. The corresponding I/O (push button, switch, LED assignments) are shown in Figure 1. Note: **Because we have a slow running clock; you will have to push the money sensor push buttons down for at least one full clock cycle for it to register.**

10. Demonstrate BOTH the working FPGA design and bring up the simulation waveform and explain to your TA briefly what is happening. We want to know that you understand the operation of your circuit in a clock by clock basis.

11. Submit the specified files on our website: vending.v, vending_fsm.v, vending_tb.v

6 EE209 Lab Grading Rubric

Student Name: _____

TA Initials for valid simulation and correct student explanation of its operation: _____

TA Initials for Correct FPGA demonstration: _____

Item	Outcome	Score	Max.
FSM			
• NSL Correct	Yes / No		1
• Outputs Correct	Yes / No		1
Datapath			
• C Register will initialize with M-100	Yes / No		1
• C Register enabled appropriately	Yes / No		1
• Constant {100,25,10,5} select logic (money mux) is correct	Yes / No		1
• Appropriate comparison results generated	Yes / No		1
Simulation			
• Testbench describes a valid usage of the vending machine operation	Yes / No		1
• Correct simulation and student able to explain what is happening in each clock (TA signature)	Yes / No		1
Correct demonstration (TA signature)	Yes / No		2
SubTotal			10
Late Deductions (-1 pts. per day)			
Total			10
Open Ended Comments:			