CSC 244: Digital Logic Design
Project 1: Vending Machine

1 Overview

You wake up after an all night study-a-thon to discover you've been transported back in time to London in 1592 right across the street from a place called the Globe Theater. In need of a way to make money you decide to use the knowledge gleaned from your favorite course, CSC-244, to design a vending machine to go outside the theater's entrance. After all what's a show without concessions. Apparently some Shakespear guy is about to debut, and you recall he made some real stinkers. The crowd will definitely want to throw things at the stage to express their displeasure; so, a rotten tomato vending machine will be a sure fire winner. The vending machine will take a fixed amount of money and dispense a rotten tomato based on the following specifications:

1.1 Project Requirements

1.1.a the machine has a single coin slot that accepts farthings (0.25d), ha'pennies (0.5d), and pennies (1d);

1.1.b the machine will dispense a rotten tomato when a penny and farthing (1.25d) has been received;

1.1.c the machine should work no matter what order the money is inserted;

1.1.d the machine must dispense correct change (per farthing) for amounts greater than 1.25d using farthings and ha'penny coins (i.e., up to 0.75d change if 1d is inserted after 1d is already in machine — *states* up to 2d);

1.1.e the machine will dispense the rotten tomato and change at the same time.

1.1.f after dispensing the rotten tomato, the machine must count additional money towards the *next* purchase without needing to be reset;

1.1.g if no money is added after dispensing the rotten tomato, the next clock pulse should take the machine back to $0 for the next customer;

1.1.h the machine must show the current amount of money using 7-segment displays (in farthings);

1.1.i in an output state, the machine must show the amount of change using different 7-segment displays (in dollars); and

1.1.j the machine must have a hidden *asynchronous* reset button that you can use to set the total back to d.

1.2 Additional Course Requirements

Your vending machine must be implemented using a Moore-style finite state machine (FSM) implemented using SystemVerilog (SV) and demonstrated on a DE10-Lite board. Your top-level SV module should only be structural SV, connecting together the DE10-Lite inputs and outputs with your submodules (e.g., see Fig. 1). You must work with a single partner from your lab section **(teams of no more than two)**. You will submit your deliverables to a D2L dropbox, one per group.

2 Deliverables and Deadlines

Item	Due Date
Meet with your partner and discuss who's working on what parts of the project	Sept. 29, by midnight
Preliminary Design,	Tuesday, Oct. 3, before noon to Lab D2L, to be checked off in lab
System Verilog Completed, Tested, and Submitted	Tuesday, Oct. 10, at noon on D2L
Project Demonstration	At the *start* of **your scheduled lab section** Oct. 10 - 12
Final Design Video Report	Sunday, Oct. 15, before midnight on D2L

3 Procedure

3.1 Preliminary Design

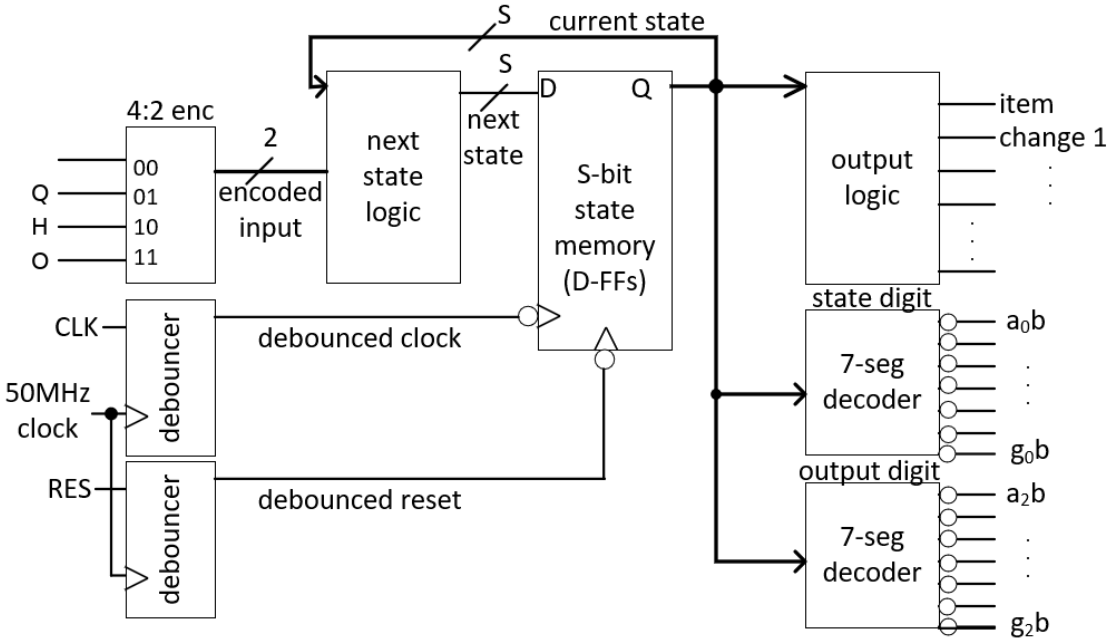

Figure 1: Suggested top-level structural implementation of the vending machine as a Moore-style finite state machine. The current and next state logic signals are S-bits wide, depending on how many bits are required to hold all possible vending machine states. (**Do not use this figure in your report, create your own based on *your* implementation**. **If you use this figure in your report, you will receive a 0 for the formatting part of your grade**). A Visio and SVG (vector graphics, can edit using Inkscape) file is provided to you that you may edit based on your top-level design.

Each **team** will turn in a preliminary design report (PDR) on Tuesday, Oct. 4, before noon to the Lab D2L, to be checked off in lab. **Keep a copy of the document for yourself.** The PDR will at the minimum include:

1. names of team members,
2. a list of inputs and outputs to your machine,
3. a list of assumptions,
4. a list of states required (and number of flip-flops),
5. a state transition diagram,
6. a state transition/output table, and
7. a list of required SystemVerilog modules (with inputs/outputs).

Note that exactly what is required will depend upon your design approach. You do not need to include completed logic equations, but you should have read and understood the information on five- and six-variable K-maps from here: `https://www.electricaltechnology.org/2018/05/karnaugh-map-k-map.html`. Five- and six-variable K-maps will be posted as a supplementary video with example. You should be aware of any constraints imposed by your selected minimization approach and outline the steps that you have taken to ensure that your design will be successful.

An example of Mr. Galipeau's working vending machine top-level design is provided in Fig. 1. This is a good starting point, but you will most likely not follow it exactly. Note that the one-, half-, and quarter-dollar inputs are *encoded* into two-bits that are used in the next state logic. Explain why this is possible in your preliminary report, and **be sure to derive your next state logic using the encoded inputs (not the coins)!**

You should also note the *debouncer* in front of the CLK and RESET inputs. Recall from Lab 4 that a mechanical switch or button will *bounce* before settling, and the bounces will be mistakenly counted as clock cycles in your state machine. Mr. Galipeau has provided a debouncer module on D2L, and negative-edge D (and JK) flip-flop modules (with asynchronous reset) that can be used in your design. You may find these on D2L and in Appendix B.

3.2 DE10-Lite Implementation

Your vending machine FSM will be implemented on your DE10-Lite board using SystemVerilog HDL to create a modular design. You will implement your project using *only structural SystemVerilog* and the behavioral SystemVerilog operators in Table 1[1]. Your top-level module **must only include structural SV**, connecting DE10-Lite I/O to your submodules. The required I/O assignments for your DE10-Lite board are given in Table 2. **Hint:** the 7-segment displays on your DE10-Lite board are *active-low*.

3.3 Project Demonstration

You are to have your project working, tested, and submitted ***before*** noon on Tuesday Oct 10. Your tested and fully functional SystemVerilog code should be submitted to D2L prior to that time. October 10th-12th will be for demonstrating your project to Mr. Galipeau, not to be debugging your project. You must be ready to demonstrate at the *start of lab* that week. There will be a lab exercise on the week of October 10th-12th; if you are unprepared for the project demonstration and/or did not submit final SV files on time **you will be required to write a lab report** for the lab. If you are prepared, you will not be required to write a report.

[1] only exception is for the SV modules provided by Mr. Galipeau

Table 1: Behavioral SystemVerilog Operators

Operator	**Meaning**
~	NOT
&	AND
~&	NAND
^	XOR
~^	XNOR
|	OR
~|	NOR

Table 2: Required DE10-Lite Input/Output

Signal	**DE10-Lite I/O**
One-Dollar (O), Half-Dollar (H), Quarter (Q) Inputs	SW[2:0]
CLK Input	KEY0
RES Input	KEY1
Current State Output	LEDR[S-1:0]
Tomato Output	LEDR[9]
Ha'penny Output	LEDR[8]
Farthing Output	LEDR[7]
(State) Farthings	HEX5
(Change) Farthings	HEX0

You will label your DE10-Lite I/O and the TA will instruct you how to test that your project works. It should work for multiple purchases (not just the first time). It **must not** require a "reset" or extra clock pulse between purchases, but it should handle these cases if they arise. To receive any points for the project, you will need to (a) demonstrate a working project, and (b) submit a project report.

4 Report Guidelines

Each **team** will turn in a final report to D2L, before midnight on Sunday, October 15. A template to follow is provided on D2L.

This report should include everything an audience member might want to know about your project and the steps you took in designing and building it. Include everything from the preliminary design document and more. State tables, state diagrams, excitation logic, flow charts, circuit drawings — whatever you needed to complete your design using your chosen approach. Include lessons learned and a section in the conclusions on how you could improve your project if you had another week to work on it. This is your chance to sell me on your project, so mention any extra features, or anything you feel is extraordinary about your project. **You may assume that your reader has knowledge of combinational logic and flip-flop memory, but has never seen a finite state machine.**

This report will contain copies of your final working SystemVerilog source files (**the source code should be in an Appendix in a `monospace font`, such as courier new**). Your SystemVerilog modules should be *fully commented*. The report should indicate the source of any

Verilog you did not write yourself. Your report should explain *exactly* how your project works, describing *the hardware*, not the SystemVerilog.

Your PDR may be hand-drawn, but any circuits, figures, graphs, and tables included the final report should be created using a software tool, such as Visio, Inkscape, GIMP, KICAD. **Note:** screenshots of the RTL viewer are not quality figures in the context of this report. Turn in one .pdf file of your report per group to a D2L dropbox. Your project and report will be graded according to the rubric in Table 3. Common project report mistakes are:

- students forget to include a top-level diagram of ***their*** project implementation, including their SV names of all logic signals;
- students forget to describe ***in the report*** all of their hardware modules, and how they were designed;
- students forget to define all of their inputs, outputs, and logic signals prior to using them in equations or hardware descriptions;
- students forget to include all of their next state and output combinational logic equations (with defined variable names);
- students forget to include their state transition table, output table, and state encodings;
- students forget to describe all figures, equations, and tables ***in the report.***;
- students forget to mention the DE-10 Lite board and how the FSM design was implemented on the board; and
- one student completes all technical work, and the partner writes the lab report without understanding how the design works.

Student Names:[2]

Table 3: Grading Rubric

score	possible	area
	50	Working Project Demo (basic functionality required for any further credit, points may be deducted for sub-optimal or out of spec implementations). **On-time** team selection, preliminary design document, SystemVerilog, and demonstration. **Followed instructions.**
	6	Abstract and Introduction
	8	Theory
	16	Procedure, Results, and Analysis
	6	Conclusions
	14	Format, Spelling, Grammar, and quality of figures, equations, etc. **Note: a screenshot of the RTL viewer is NOT a quality diagram.**
	100	**Total**

[2]The report sections will be graded according to the lab report rubric.

A Recommended Testing Procedures

This appendix gives recommendations of module testing for project teams that you may wish to do *prior* to integrating each piece into your complete system design. It is good engineering practice to perform unit testing to ensure each individual component works as expected (especially when designing a unit to be integrated into a larger team design). The second form of testing is *systems integration testing*, where you are debugging the system level design. If you try to integrate your system prior to unit testing, you will not be able to distinguish between an integration error, and an individual component error.

A.1 State Memory and Debounced Clock Signal

A good way to test that your state memory is working properly is to combine it with the debouncer, and verify your inputs and outputs. Connect KEY0 of your DE10-Lite board to the D input of the debouncer, the DE10-Lite board's 50 MHz clock signal to the clk_50M input, and connect the D_deb output to an internal `logic` signal called CLK_deb. Connect clock inputs of the S-bits of state memory (made using Mr. Galipeau's `D_FF_neg` module) to your CLK_deb signal. Verify proper operation by connecting 'S' DE10-Lite switches to the state memory inputs, and 'S' red LEDs to the state memory outputs. The outputs should update to the inputs on the press of KEY0. A similar procedure can be used to test the asynchronous reset.

A.2 Input Encoder

To test that your 4:2 encoder (or priority encoder) is working properly, you can connect four DE10-Lite switches to the inputs, and verify the outputs on two red LEDs. **Double check the *endian-ness* of your module** when connecting it into your top-level design (i.e., be aware of MSB and LSB!).

A.3 Next State Logic

Your next state logic is a *combinational* circuit that determines the S-bit next state based on the S-bit current state and your ***encoded*** inputs. To test the proper operation of your next state logic, you can connect the S-bit *current state* and two *encoded inputs* to DE10-Lite switches, and verify that you get the correct *next state* by outputting to 'S' red LEDs.

If you notice an error in your next state logic for a given set of inputs and current state, this can easily be tracked down in your K-maps by checking the corresponding *cell*. For example, if you have 2-bits of state S_1S_0 and two inputs x_1x_0, you would have a 4-variable next state logic equation for each bit of state memory. Assume that in state $S_{1:0}$ =`2'b01` and inputs $x_{1:0}$ =`2'b11`, the next state should be $S^*_{1:0}$ =`2'b11`, but you notice when testing your next state logic sets $S^*_{1:0}$ =`2'b10`. This means that you have an error in your next state equation for S_0, and it is in the cell corresponding to $S_{1:0}$ =`2'b01` and $x_{1:0}$ =`2'b11`, where you forgot to include this cell in one of your groups (hence, your next state bit is calculated as a 0 instead of a 1).

A.4 Output Logic

In a Moore-style FSM, your output logic only depends on your current state. You can easily test your output *combinational* logic by connecting 'S' DE10-Lite switches as your current state inputs, and one red LED per output (e.g., item, change). Verify that for each valid state of your FSM, the output logic is correct.

A.5 7-Segment Decoders

The 7-segment decoders are also Moore-style FSM outputs, where the display only depends on the current state (amount of money in the machine). You can verify that your ***active-low*** 7-segment display combinational logic works correctly by connecting 'S' DE10-Lite switches as your current state inputs, and HEX1 and HEX0 as your state and change (in dollars), respectively. **Hint:** unlike the 7-segment display we derived in class, your vending machine does not have to output the full range 0-F. You may be able to use this fact to create simpler logic equations for each segment. Alternately you can reuse the module you developed in class to drive the 7-segment displays

B Required SystemVerilog Modules

In this section, SystemVerilog modules are provided for a negative-edge triggered D (and JK) flip-flop with asynchronous reset, and a button/switch debouncer. You must make use of these two devices in your project (SystemVerilog is also available on D2L). The `D_FF_neg` module sets $Q^* = D$ on the negative-edge of $CLKb$ (Qb^* will get D'). The debouncer uses a 50 MHz clock from the FPGA (PIN_P11 or PIN_N14) as an input to $CLK50M$. After the *Anoisy* input has settled, 1ms later the A input will equal A_noisy. The debouncer is guaranteed to only output one transition of A per button press/release.

B.1 Negative-Edge Triggered D Flip-Flop with Asynchronous Reset

```
/*
*       Author: Dr. Hansen
*       Date: Feb. 14, 2017
*       Description: implements a negative-edge D-flipflop.
*               On the negative edge of CLKb, Q := D.
*
*       Inputs:
*               D - D input
*               CLKb - negative edge clock input
*               RSTb - asynchronous, negative edge reset
*
*       Outputs:
*               Q - outputs D on negedge-edge of clk
*               Qb - outputs D' on negedge-edge of clk
*/

module D_FF_neg(
  input logic D,CLKb,RSTb,
  output logic Q,Qb
);
  always_ff@(negedge CLKb, negedge RSTb)
  begin
    if(RSTb == 1'b0)
    begin
      Q <= 1'b0;
      Qb <= 1'b1;
    end
    else
    begin
      Q <= D;
      Qb <= ~D;
    end
  end
endmodule
```

B.2 Negative-Edge Triggered JK Flip-Flop with Asynchronous Reset

```
/*
*       Author: Dr. Hansen
*       Date: Feb. 16, 2022
*       Description: implements a negative-edge JK-flipflop.
*               On the negative edge of CLKb, Q := JQ' + KQ
*               Asynchronous, negative-edge reset
*
*       Inputs:
*               J - J input
*               K - K input
*               CLKb - negative edge clock input
*               RSTb - asynchronous, negative edge reset
*
*       Outputs:
*               Q - outputs JQ' + KQ on negedge-edge of CLKb
*               Qb - outputs Q' on negedge-edge of CLKb
*
*       History:
*               Feb. 16, 2022 - created
*/

module JK_FF_neg(
input logic J,K,CLKb,RSTb,
output logic Q,Qb
);

        always_ff@(negedge CLKb, negedge RSTb)
        begin
                if(RSTb == 1'b0)
                begin
                        Q <= 1'b0;
                end
                else
                begin
                        Q <= (J&Qb) | (K&Q);
                end
        end

        assign Qb = ~Q;
endmodule
```

B.3 Debouncer

```
/*
        Author: Dr. Hansen
                Date: Feb. 9, 2022
                Description: takes a 1-bit noisy input and
                outputs a 1-bit clean signal
        Inputs:
                A_noisy - signal to be debounced
                CLK50M - 50 MHz clock from the DE10 board (PIN_P11)
        Outputs:
                A - debounced signal to be used in your circuit
*/
module debouncer(
        input logic A_noisy,
        input logic CLK50M,
        output logic A
);
        logic [15:0] COUNT;
        parameter [15:0] COMPARE = 50_000; //1 millisecond

        logic t_d, t_r, Anext;

        //      1 ms timer
        always_ff@(posedge CLK50M)
        begin
                if(t_r)
                        COUNT <= 16'd0;
                else
                        COUNT <= COUNT + 16'd1;
        end
        assign t_d = (COUNT >= COMPARE);

        //next-state logic
        assign Anext = (A_noisy & t_d) | (A & ~t_d);

        //state
        always_ff@(posedge CLK50M)
                A <= Anext;

        //output logic
        assign t_r = t_d | (~A & ~A_noisy) | (A & A_noisy);
endmodule
```