

EE M16 and CS M51A Winter 2019 Section 2

Logic Design of Digital Systems

Dr. Yutao He

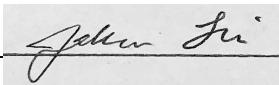
Verilog Lab #3 - Design of Sequential Systems

Due: March 10, 2019

Team ID: \_\_\_\_\_ P07 \_\_\_\_\_

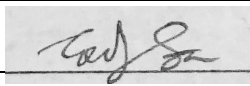
(1) Name: \_\_\_\_\_  
Liu Joshua  
Last First

Student ID: \_\_\_\_\_ 105136031 \_\_\_\_\_

Signature: \_\_\_\_\_  


(2) Name: \_\_\_\_\_  
Fan Cody  
Last First

Student ID: \_\_\_\_\_ 205102002 \_\_\_\_\_

Signature: \_\_\_\_\_  


Date: \_\_\_\_\_ 3/10/2019 \_\_\_\_\_

Result	
Correctness	
Creativity	
Report	
Total Score	

# Verilog Lab #3 Project Requirement

Dr. Yutao He

Due: 3/10/2019

## 1 Objectives

The last project in this quarter is intended to: (1) get you familiar with the design flow of building a sequential circuit from the high-level specification to the final working system composed of logic gates and flip-flops, (2) obtain the hands-on experiences of using flip-flops and connecting the clock signal properly in the sequential system.

## 2 The Project Description

### 2.1 The High-Level Specification

The finite state machine (FSM) to be built in the project is a simple vending machine controller called *iKon*. A *controller* is an important class of sequential systems that produces a set of *control signals* as its states are traversed. These control signals are used to determine actions performed in another system.

Here is how the *iKon* controller is supposed to work. The vending machine delivers a package of gum after it has received 20 cents in coins. The machine has a single coin slot that accepts only nickels and dimes, one coin at a time. A mechanical sensor indicates to the controller if any coin has been inserted into the coin slot and which type of coin it is. A reset button causes two actions in case a customer changes his/her mind: (1) set the controller to the initial state, and (2) drive another mechanism to return all deposited coins. As a result, the controller does not have to generate a control signal that commands return of all coins when the reset button is pressed. The outputs of the controller cause a single package of gum to be released down a chute to a customer, or return 5 cents changes if necessary.

The high-level interface between the *iKon* controller and the rest of the vending machine is shown in Figure 1. The inputs, outputs and the states of the controller are listed in Table 1 and Table 2, respectively.

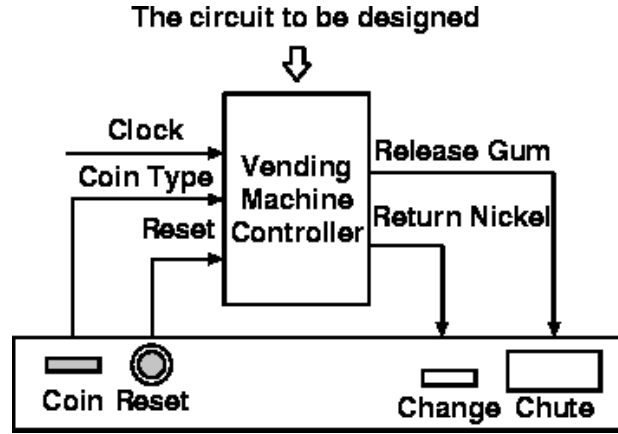


Figure 1: The Block Diagram of a Simple Vending Machine

Inputs		Outputs	
Variables	Values	Variables	Values
Reset	{True (T),False (F)}	Release Gum (RG)	{T,F}
Coin	{Empty (E), Nickel (N), Dime (D) }	Return Nickel (RN)	{T,F}

Table 1: The Inputs and Outputs of the *iKon* Controller

The complete behavior of the *iKon* controller can be specified by the state diagram in Figure 2. To keep the state diagram clear and readable, those transitions caused by the reset input signal are not included. It is obvious though that if the reset input is asserted at any state, the controller will go to the initial state. In addition, only transitions that *explicitly* cause a state change, and the output in states where it is asserted, are included in the diagram. For example, the transition from state 15c to initial state is labeled with  $N/RG$  or  $D/RG, RN$ , which means that the transition can only happen in two cases (except when the reset is asserted): (1) if another nickel is deposited, then a package of gum is released, or (2) if a dime is deposited, then a package of gum is released and 5c (a nickel) is returned.

## 2.2 The Binary-Level Specification

### 2.2.1 Encoding Schemes of Inputs

You must use the encoding schemes shown in Table 3 for the inputs.

States	Descriptions
Init	the initial state
5c	the amount of deposited coins is 5 cents
10c	the amount of deposited coins is 10 cents
15c	the amount of deposited coins is 15 cents

Table 2: The States of the *iKon* Controller

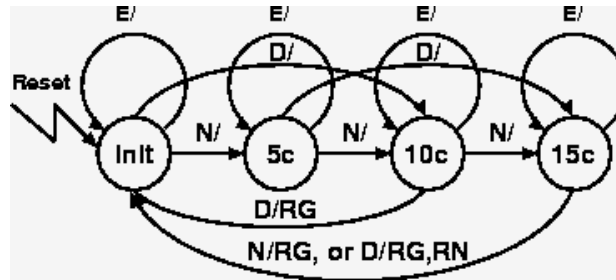


Figure 2: The State Diagram of the *iKon* Controller

### 2.2.2 Encoding Schemes of Outputs

You must use the encoding schemes shown in Table 4 for the outputs.

### 2.2.3 Encoding Schemes of States

You must use the encoding schemes shown in Table 5 for the states.

## 2.3 Additional Requirements for The Implementation

In addition to complying with the high-level and binary-level specifications, your design must also satisfy the following requirements:

Coin	$x_1x_0$	Reset	$r$
Empty	00	False	0
Nickel	01	True	1
Dime	11		

Table 3: The Encoding Scheme for Inputs

<b>RG</b>	$z_1$	<b>RN</b>	$z_0$
False	0	False	0
True	1	True	1

Table 4: The Encoding Scheme for Outputs

<b>State</b>	$s_1s_0$
Init	00
$5c$	01
$10c$	11
$15c$	10

Table 5: The Encoding Scheme for States

- **Combinational Logic:** The combinational logic in the system should be implemented with the minimal two-level OR-AND network. **Note** that NOT gates used to generate negated input variables are not counted as one additional level.
- **Flip-flops:** You should use JK flip-flops to store  $s_1$  and  $s_0$ .

## 2.4 Extra Credits

This part is not for the minimal requirement but for those highly motivated minds.

It can be easily seen that there is one basic limitation in the original *iKon* controller - *it does not take quarters*. Suppose that *TripleMint Corp.*, the buyer of the *iKon* controller, is not satisfied with the design and complains to your boss in your company, *NeoElektTech*, who then asks you to design the next generation of *iKon* called *iKonPlus* by modifying your original design. In addition, your company *NeoElektTech* recently purchased a batch of multiplexers at very cheap price through a secret channel.

As a result, your boss decided to implement combinational logic of the *iKonPlus* controller with MUXes instead of OR, AND gates to reduce the cost. Design and implement the *iKonPlus* controller.

Your successful demonstration on the Mojo FPGA board may also receive extra credits.

### 3 Report Outline

Each team is required to submit one report that provides complete documentation of your project including the detailed design steps. As in all technical writing, its purpose is to communicate your work with your colleagues in an efficient and professional way so that your design can be upgraded and maintained even if you are no longer around. As a result, the report should be clear, concise and complete and should contain the following parts:

(1) *Title Page*

It is provided and you just need to fill in your information in the blanks.

(2) *Abstract*

This is the brief high-level description of the project in plain English text.

(3) *The Functions of the Circuit*

It is part of the design work for you to obtain the binary-level specification for the function of the circuit in the form of switching expressions and the schematic. This section should present both minimal switching expressions in OR-AND form and the schematic of the circuit composed of logic gates and flip-flops. Detailed steps of obtaining them should go to Appendix.

(4) *The Verilog Code*

The Verilog code you write is the implementation of the circuit. You must include it in your report with **the names and SID numbers of your team members included**.

(5) *The Simulation Result*

You have to demonstrate that your implementation works as specified by showing the simulation result. In your simulation, you should show the following three sequences: (1) D, N, D; (2) N, N, N, N, and (3) D, Reset. Please clearly write down the necessary information on the waveform diagram so that one of your colleagues who does not know anything about your project could understand the behavior of the system you are trying to implement (Uncommented timing diagrams will be considered incomplete and points will be marked down).

(6) *The Design Review*

This section includes a summary of your experiences throughout the project. It should be no more than two pages and may include such topics as what you have learned, problem encountered during the implementation and the workarounds you came up with, the approach you used, the most important aspects of the project for you, where you spent most of your time and suggestions you want to make. In particular, you may include here the new design scheme you proposed for the *iKonPlus* controller and alternative implementation with MUXes for extra points.

(7) *Team Member Contributions*

Teamwork requires that each member be responsible and assume a relatively equal share of workload. In this section, a detailed description on each member's responsibility and contribution should be presented clearly, including an estimate of percentage of efforts on the project and a summary list of each member in the project.

Each member requires to review and sign the final report on the cover page.

(8) *Appendix - The detailed design worksheet*

This part must be included in your report and must show the complete worksheet during the pencil-and-paper design. It should contain:

- 8.1 inputs, outputs, and states of the system.
- 8.2 encoding schemes of inputs, output, and states.
- 8.3 the state diagram and the state table.
- 8.4 minimization procedure for state and output variables by means of the K-map.
- 8.5 final minimal expressions of the logic functions in forms of OR-AND switching expressions and the final schematic of the circuit.

## 4 Project Submission

You should submit one zipped file named with "Txx.zip" via the on-line submission link. The zipped file should consist of four separate files:

1. The pdf file of your report. It must be named as "Txx.pdf" where xx is your Team ID assigned to you, that is, your report should be called *Txx.pdf*;
2. The Verilog file of your circuit implementation. It should be named as *eem16\_proj3.v*.
3. The Verilog file of your implementation of a JK flipflop. It should be named as *jkff.v*.
4. The testbench file. It should be named as *eem16\_proj3\_tb.v*.
5. If you complete the extra point problem, please also include the verilog file for the enhanced controller named *eem16\_proj3\_bonus.v* and the testbench file named *eem16\_proj3\_bonus\_tb.v*.

## 5 Project Deadline

The report is due by midnight (11:59:59pm) on March 10 (Sunday), 2019. The deadline must be observed strictly and late submissions will be subject to penalty.

## 6 Design Tips

### 6.1 About JK Flip-Flops

In practice, the majority of digital designs that are modeled in Verilog are clocked, synchronous systems using edge-triggered flip-flops. You need to design a JK-flip-flop module in a separate file called *jkff.v* and use it.

Asynchronous input means that the effect of activating the input takes place immediately, independent of the clock signal. It is useful because when designing a sequential system it is important to define the *initial state* of operation. This state must be valid when the circuit is first powered on, or after an initialization phase (activating a *RESET* signal, for example). In general, the initial state is chosen such that all state bits are zeroes. The value zero in all FFs is easily obtained by connecting the clear input of each FF together to the same signal line. Before the circuit is used, this line is made active (high or low, depending on the specific device being used) for a short period of time. This way, the initial state is known when the circuit starts to operate.



## 6.2 About Input RESET

Input RESET is used to set the controller to a valid initial state at any moment. Since the initial state is encoded with all zeros (see Table 5), you can implement the input RESET signal simply by connecting it directly to the asynchronous clear input of a JK flip-flop. This way, whenever the input RESET is asserted (has value 1), all flip-flops will be set to state 0. As a result, you **do not** need to include the input RESET as a variable when you design the circuit.

## 6.3 About The Clock

During the normal operation, everything in a synchronous sequential system is controlled by the clock. As a result, the clock input of each JK flip-flop must be connected to the clock signal. You will set the clock waveform when you generate your simulation waveforms.

## (2) Abstract

*This is the brief high-level description of the project.*

In this project, we are given the task to create a vending machine logic system that has four different states called iKon. We are assigned the task of creating a combinational and sequential system together in order for the system to work. It has the initial state(Init), five cent state(5c), 10 cent state(10c), and 15 cent state(15c). The vending machine is supposed to output gum when it has 20 cents in the system, and also return a nickel along with gum if it has a total of 25 cents. We were given a basic state diagram, the inputs and outputs, and the encoding schemes. We then needed to extract the state table and then use two JK flip flops to design the sequential aspect of the system, utilizing the JK flip flop excitation equation to determine outputs. We then used Karnaugh maps to minimize the switching expressions(for the flip flops and and outputs) in order to obtain the minimal form for programming in Verilog. Our circuit we created needed to be an OR-AND gate, so we minimized it using product of sums. After writing the Verilog code, we were told to test the functions on the waveform simulation program to test the outputs. Optionally, we were given the instructions to create a better vending machine called iKonPlus that utilizes multiplexers and can take quarters as inputs.

### (3) The Switching Functions of the Circuit

*It is part of the design work for you to obtain the binary-level specification for the function of the circuit in the form of switching expressions and the schematic. This section should present both minimal switching expressions in OR-AND form and the schematic of the circuit composed of logic gates and flip-flops. Detailed steps of obtaining them should go to Appendix.*

On paper, we were able to write out the states table and draw out the K-maps in order to minimize the switching expressions. Below are the minimal switching expressions in OR-AND form as well as the schematic of the circuit for both the outputs of the system and the outputs of our two JK flip flops. We used  $S_1$  and  $S_0$  as the outputs for the JK flip flops.

Minimal switching expressions in OR-AND form:

#### Outputs of System

$$RG = x_0 S_1 (S_0' + x_1)$$

$$RN = x_1 S_0' S_1$$

#### Outputs of JK Flip Flops

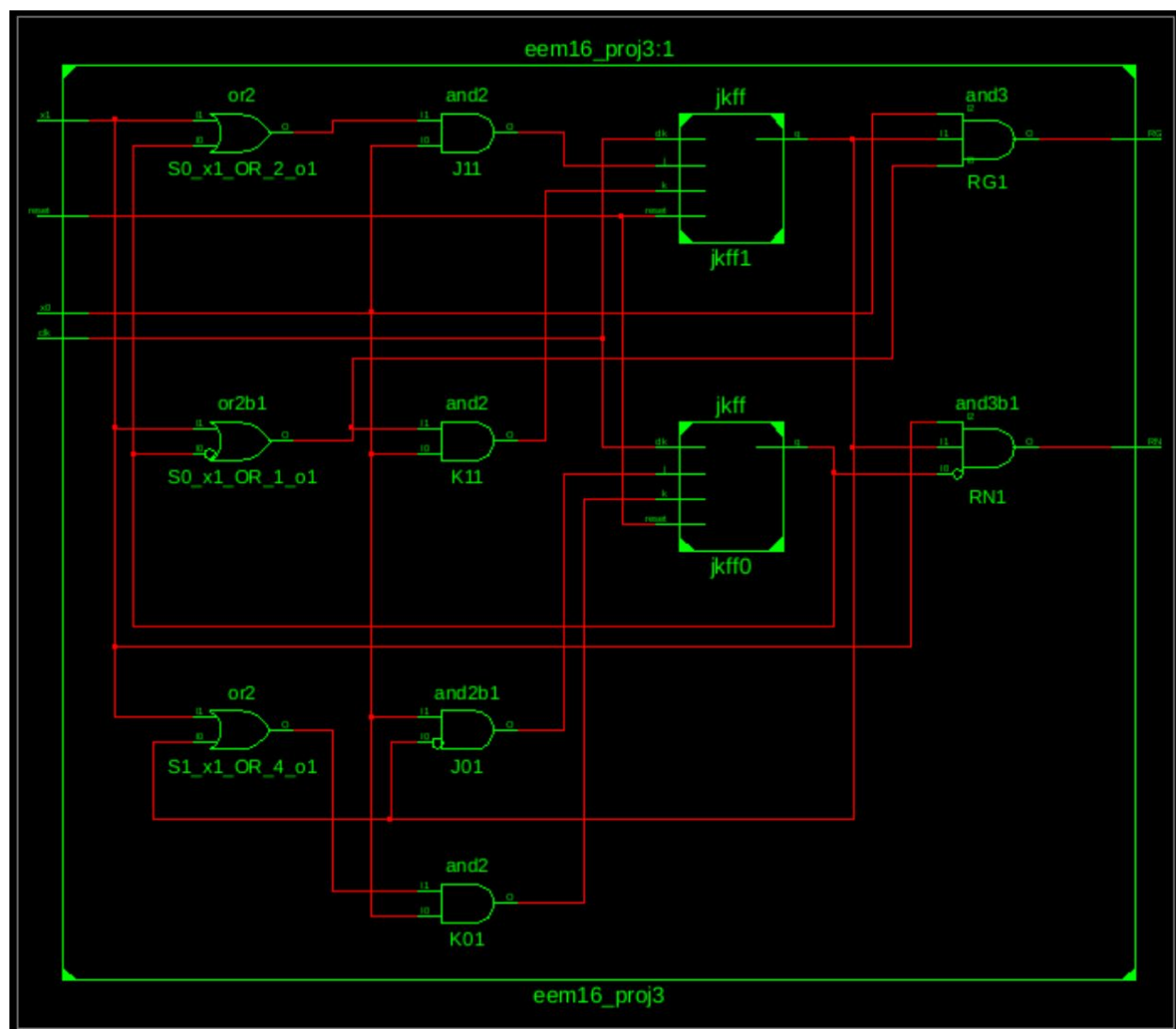
$$J_1 = x_0 (S_0 + x_1)$$

$$K_1 = x_0 (S_0' + x_1)$$

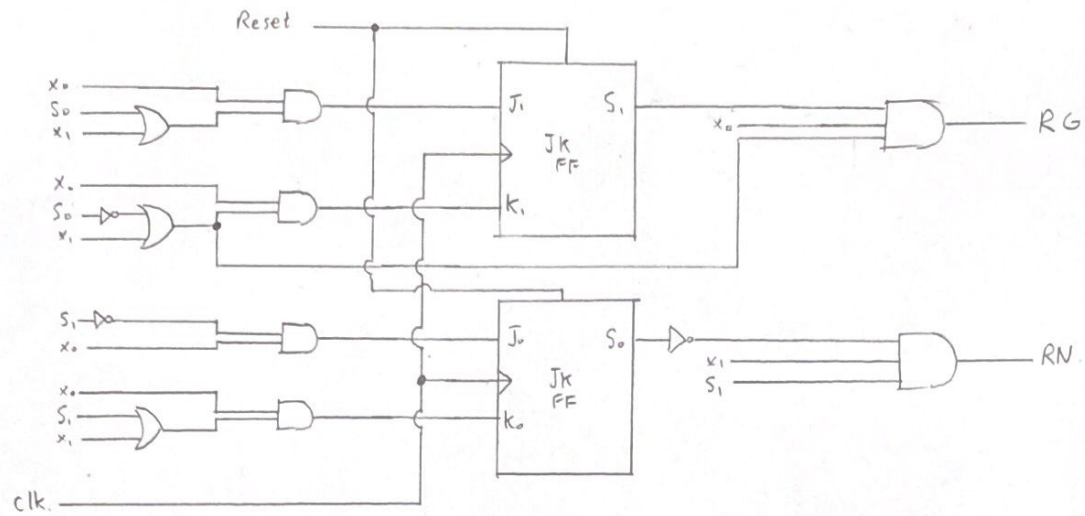
$$J_0 = S_1' x_0$$

$$K_0 = x_0 (S_1 + x_1)$$

In our schematic below, we drew out the combinational (OR-AND gate) part and the sequential part (JK flip flops) together. We also generated an RTL schematic to investigate the network after implementing our code, which shows us that our circuit uses a total of 3 OR gates and 6 AND gates after minimizing through K-maps. The reset and clk are also connected to the JK flip flops but they are not included in the inputs since they are asynchronous. The clock, on the other hand, is essential for all sequential systems.



# iKon Vending Machine schematic (2 JK Flip Flops).



(RTL Schematic from Xilinx in Section 3).

#### (4) The Verilog Code of the Circuit

*The Verilog code you write is the implementation of the circuit. You must include it in your report with the names and SID numbers of your team members included.*

##### **eem16\_proj3.v**

```
`timescale 1ns / 1ps
// Team: Joshua Liu (105136031), Cody Fan (205102002)
// Date: 3/6/19
// Team ID: P07
// Project: Project #3 Design of Sequential Systems
// Class: EE M16 Logic Design of Digital Systems
module eem16_proj3(x1, x0, reset, clk, RG, RN);

    // Declare input and outputs
    input x1, x0, reset, clk;
    output RG, RN;

    // Need wire since these are both inputs and outputs
    wire S1, S0;
    wire J1, K1, J0, K0;

    // Instantiate JK flip flops
    jkff jkff1(reset, clk, J1, K1, S1);
    jkff jkff0(reset, clk, J0, K0, S0);

    // Outputs of system
    assign RG = x0 & S1 & (~S0 | x1);
    assign RN = x1 & ~S0 & S1;

    // Inputs to flip flops
    assign J1 = x0 & (S0 | x1);
    assign K1 = x0 & (~S0 | x1);
    assign J0 = ~S1 & x0;
    assign K0 = x0 & (S1 | x1);

endmodule

/* In this main module file we created the combinational system and also instantiated the JK
flip flops to create and link up the circuit. We used S1, S0, J1, K1, J0, K0 as wires since they are
used in both outputs and inputs.*/
```

## jkff.v

```
`timescale 1ns / 1ps
// Team: Joshua Liu (105136031), Cody Fan (205102002)
// Date: 3/6/19
// Team ID: P07
// Project: Project #3 Design of Sequential Systems
// Class: EE M16 Logic Design of Digital Systems
module jkff(reset, clk, j, k, q);
    input j, k, clk, reset;
    output q;

    wire j, k, clk, reset;
    reg q;

    always @(posedge clk) begin

        if(reset) begin
            q=1'b0;
        end else begin

            case({j, k})
                {1'b0, 1'b0}: begin q = q; end
                {1'b0, 1'b1}: begin q = 1'b0; end
                {1'b1, 1'b0}: begin q = 1'b1; end
                {1'b1, 1'b1}: begin q = ~q; end
            endcase
        end
    end

endmodule

/* In this JK flip flop file, we create the JK flip flop and link the inputs to the outputs. We also
included the asynchronous reset input that resets the system state back to the initial state, Init.
*/
```

```
`timescale 1ns / 1ps
// Team: Joshua Liu (105136031), Cody Fan (205102002)
// Date: 3/6/19
// Team ID: P07
// Project: Project #3 Design of Sequential Systems
// Class: EE M16 Logic Design of Digital Systems
module eem16_proj3_tb;
```

```
// Initialize inputs (100 ns)
```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//      BEGIN      //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Set reset to 1
x1 = 0; x0 = 0; reset = 1; clk = 1; #25;
// Cycle the clock (set LOW to use HIGH again)
clk = 0; #25;
// Set reset to 0
clk = 1; reset = 0; #25;
// Cycle the clock (set LOW to use HIGH again)
clk = 0; #25;

```

```

////////////////////////////////////
//      END      //
////////////////////////////////////

```



```
// Simulation 1 -> D N D (100 ns - 300 ns)
```

```
//////////////////////////////////////////////////////////////////
// BEGIN DIME NICKEL DIME SIMULATION //
//////////////////////////////////////////////////////////////////
// Set clock to 1 and inputs to 11 for dime (25 ns) -> 10c state
clk = 1; x0 = 1; x1 = 1; #25;
// Cycle the clock (set LOW to use HIGH again) (25 ns)
clk = 0; #25;
// Set clock to 1 and inputs to 01 for nickel (25 ns) -> 15c state
clk = 1; x0 = 1; x1 = 0; #25;
// Cycle the clock (set LOW to use HIGH again) (25 ns)
clk = 0; #25;
// Set clock to 1 and inputs to 11 for dime (25 ns) -> Init state, return gum and nickel
clk = 1; x0 = 1; x1 = 1; #25;
// Cycle the clock (set LOW to use HIGH again) (25 ns)
clk = 0; #25;
```

```
// Reset the system
```

```
// Set reset to 1
x1 = 0; x0 = 0; reset = 1; clk = 1; #10;
// Cycle the clock (set LOW to use HIGH again)
clk = 0; #10;
// Set reset to 0
clk = 1; reset = 0; #10;
// Cycle the clock (set LOW to use HIGH again)
clk = 0; #20;
```

```
// NEXT SIMULATION STARTS AT 300 ns
```

```
//////////////////////////////////////////////////////////////////
// END DIME NICKEL DIME SIMULATION //
//////////////////////////////////////////////////////////////////
```

```
// Simulation 2 -> N N N N (300 ns - 600 ns)
```

```
//////////////////////////////////////////////////////////////////
// BEGIN NICKEL NICKEL NICKEL NICKEL SIMULATION //
//////////////////////////////////////////////////////////////////
// Set clock to 1 and inputs to 01 for nickel (25 ns) -> 5c state
clk = 1; x0 = 1; x1 = 0; #25;
// Cycle the clock (set LOW to use HIGH again) (25 ns)
```

```

clk = 0; #25;
// Set clock to 1 and inputs to 01 for nickel (25 ns) -> 10c state
clk = 1; x0 = 1; x1 = 0; #25;
// Cycle the clock (set LOW to use HIGH again) (25 ns)
clk = 0; #25;
// Set clock to 1 and inputs to 01 for nickel (25 ns) -> 15c state
clk = 1; x0 = 1; x1 = 0; #25;
// Cycle the clock (set LOW to use HIGH again) (25 ns)
clk = 0; #25;
// Set clock to 1 and inputs to 01 for nickel (25 ns) -> Init state, return gum
clk = 1; x0 = 1; x1 = 0; #25;
// Cycle the clock (set LOW to use HIGH again) (25 ns)
clk = 0; #25;

```

```

// Reset the system

```

```

// Set reset to 1
x1 = 0; x0 = 0; reset = 1; clk = 1; #25;
// Cycle the clock (set LOW to use HIGH again)
clk = 0; #25;
// Set reset to 0
clk = 1; reset = 0; #25;
// Cycle the clock (set LOW to use HIGH again)
clk = 0; #25;

```

```

// NEXT SIMULATION STARTS AT 600 ns

```

```

/////////////////////////////////////////////////////////////////
//  END NICKEL NICKEL NICKEL NICKEL SIMULATION  //
/////////////////////////////////////////////////////////////////

```

```

// Simulation 3 -> D RESET (600 ns - 900 ns)

```

```

/////////////////////////////////////////////////////////////////
//  BEGIN DIME RESET SIMULATION  //
/////////////////////////////////////////////////////////////////
// Set clock to 1 and inputs to 01 for nickel (25 ns) -> 5c state
clk = 1; x0 = 1; x1 = 0; #25;
// Cycle the clock and reset inputs(set LOW to use HIGH again) (25 ns)
clk = 0; x0 = 0; x1 = 0; #25;
// Set reset to 1 (25 ns) -> Init state, return nothing
clk = 1; reset = 1; #25;
// Cycle the clock (set LOW to use HIGH again) (5 ns)
clk = 0; #5;
// Set reset to 0 (5 ns) -> Init state, return nothing

```

```

clk = 1; reset = 0; #5;
// Cycle the clock (set LOW to use HIGH again) (15 ns)
clk = 0; #15;

// Test state by inputting two dimes which should only return gum
// Set clock to 1 and inputs to 11 for dime (25 ns) -> 10c state
clk = 1; x0 = 1; x1 = 1; #25;
// Cycle the clock (set LOW to use HIGH again) (25 ns)
clk = 0; #25;
// Set clock to 1 and inputs to 11 for time (25 ns) -> Init state, return gum
clk = 1; x0 = 1; x1 = 1; #25;
// Cycle the clock (set LOW to use HIGH again) (25 ns)
clk = 0; #25;

// Reset the system

// Set reset to 1
x1 = 0; x0 = 0; reset = 1; clk = 1; #25;
// Cycle the clock (set LOW to use HIGH again)
clk = 0; #25;
// Set reset to 0
clk = 1; reset = 0; #25;
// Cycle the clock (set LOW to use HIGH again)
clk = 0; #25;

////////////////////////////////////
//   END DIME RESET SIMULATION   //
////////////////////////////////////

// Wait 100 ns for finish (finish at 1000 ns)
#100;

end

endmodule

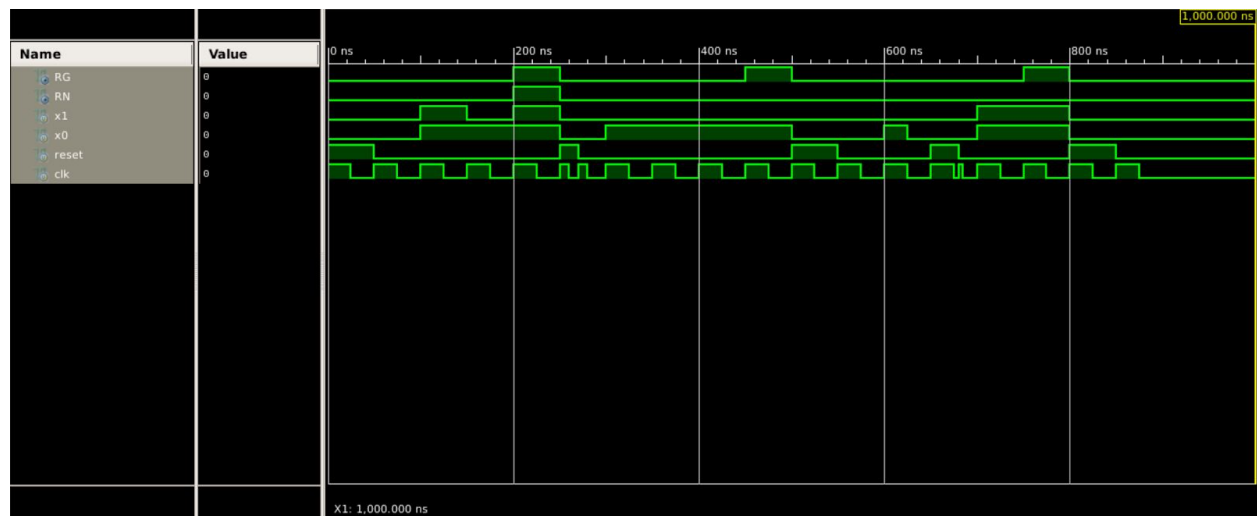
/* In this test bench file, we are given the task to simulate three different cases(D N D, N N N N,
and N Reset). In the code, there are comments saying what everything does and how it tests
the system.*/

```

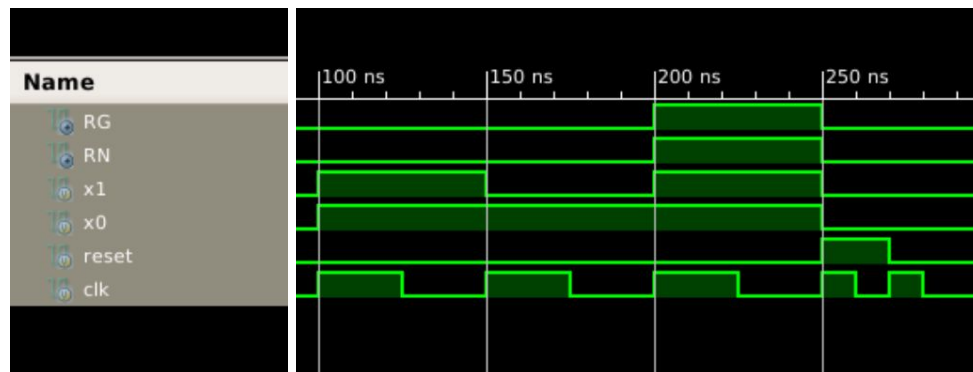
## (5) The Simulation Result

*You have to demonstrate that your implementation works as specified by showing the simulation result. In your simulation, you should show the following three sequences: (1) D, N, D; (2) N, N, N, N, and (3) D, Reset. Please clearly write down the necessary information on the waveform diagram so that one of your colleagues who does not know anything about your project could understand the behavior of the system you are trying to implement (Uncommented timing diagrams will be considered incomplete and points will be marked down).*

In this Timing Diagram, we can see when the inputs and outputs are high or low, specifically the inputs  $x_1$  and  $x_0$ , the reset values(reset), the clock values(clk), and the returning gum and nickel values(RG and RN). For the first 100 ns, we are resetting the circuit to get the sequential aspect to the circuit to work. Then, from 100 ns to 300 ns, we are testing the dime-nickel-dime test. Then, from 300 ns to 600 ns, we are testing the nickel-nickel-nickel-nickel test. Lastly, from 600 ns to 900 ns, we are testing the nickel-reset test. We end the test by resetting the circuit from 900 ns to 1000 ns.

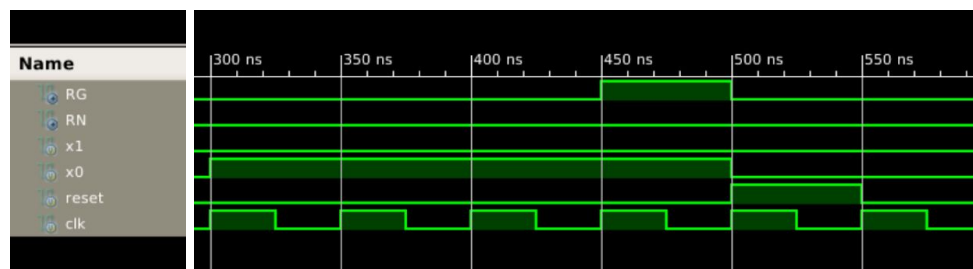


1st simulation: D N D (100 ns to 300 ns)



Here we have already begun the system. We input a dime( $x_1 = 1$  and  $x_0 = 1$ ) at 100 ns with  $\text{clk} = 1$ , then we set the clock back to zero so the system can read another positive edge. At 150 ns, we set the clock to 1 with  $x_1 = 0$  and  $x_0 = 1$  which is a nickel input. Then we cycle the clock again and have it HIGH while the inputs are 1 and 1 for a dime(200 ns). We can then see that the system returns gum and a nickel, as shown by RG and RN being 1 during the 200 ns to 250 ns duration. From 250 ns to 300 ns we are simply just resetting the system back to the initial state to have it ready to run the next simulation by 300 ns.

2nd simulation: N N N N (300 ns to 600 ns)



Here we are simulating the input of four nickels, which should only return gum. We cycle the clock to be high and set the inputs, then the clock to low. Then, we can set the clock to high again for the system to take another input. We do this four times to input four nickels( $x_0 = 1$  and  $x_1 = 0$ ), specifically from 300 ns to 350 ns, 350 ns to 400 ns, 400 ns to 450 ns, and 450 ns to 500 ns. As we can see, at 450 ns to 500 ns, the system has taken in four nickels, and so RG, return gum, is HIGH, so the system returns a piece of gum. We then reset the system back to the initial state from 500 ns to 600 ns so it can be ready for the next simulation at 600 ns.

### 3rd simulation: N Reset (600 ns to 900 ns)



Here we are simulating the input of a nickel, then hitting the reset button. Again, we need to cycle the clock so that when we change the inputs, the clock is one in order for the system to take the input. We input the nickel ( $x_0 = 1$  and  $x_1 = 0$ ) from 600 ns to 650 ns. Then, we set reset to 1 from 650 ns to 700 ns, which makes the system go back to the initial state. To know that it is in the initial state now, we try to input two dimes, and if it was in the initial state as we want, the system is supposed to output gum but not a nickel (20 cents purchases gum). We proceed with the simulation from 700 to 800, where we input two dimes, and we can see that the system returns gum from 750 to 800 ns, which means that the system has indeed 20 cents, which is the result we want. From 800 ns to 900 ns, we reset the system (keep in mind that all these changes in inputs need to have a positive clock edge, which is shown in the timing diagrams).

As a result, we have verified that our Verilog implementation of the circuit is correct and the iKon vending machine works as the specifications say.

## (6) The Design Review

*This section includes a summary of your experiences throughout the project. It should be no more than two pages and may include such topics as what you have learned, problem encountered during the implementation and the workarounds you came up with, the approach you used, the most important aspects of the project for you, where you spent most of your time and suggestions you want to make. In particular, you may include here the new design scheme you proposed for the iKonPlus controller and alternative implementation with MUXes for extra points.*

In summary, this project was a good experience for us on how to create a real life machine such as a vending machine and how the system takes in money and has different states. Learning the logic behind how the whole system works and combining combinational and sequential systems was fun. We had quite a lot of trouble figuring out the way to implement a sequential system on verilog with the posedge command and instantiating modules, but in the end we were able to get it. This project was a good way for us to practice what we have been learning throughout the whole quarter, as well as solidifying our understandings and design using flip flops. We needed to obtain a state table from a state diagram given inputs, outputs, and encoding schemes, and utilized logic gates, flip flops, the clock, and a reset input to implement the iKon machine.

Specifically, one large problem we encountered was how to use two different modules in different files and how one would combine the combinational and sequential system together, and where to use the “always [...] begin” and “end” keywords. In the end, we found that we coded our outputs from the flip flops incorrectly by setting the outputs of the flip flops as outputs in our main module, which caused us to have a “multiple drivers” error. Changing those to wires fixed the issue. Another problem we encountered was that we were not getting the right outputs from the system, specifically returning the nickel after getting 25 cents. We had to go back to our K-map minimizations and look over all our work to find a small error which then fixed the problem with the system.

When we first got the specifications of the project, we had trouble figuring out where to start, so we wrote everything down on paper to get a better idea. After doing so, we figured out that we needed to create the state table and minimize the expressions that we can obtain from it. Then, we were ready to code in verilog and test out our minimal expressions.

The most important aspects of the project to us were coding in verilog and understanding how to do so with combinational and sequential systems. Testing the waveforms was also a big part that helped us understand how we have to cycle the clock every time and how the clock and reset have to do with the system. In the end we were able to use a total of 3 OR gates and 6 AND gates. We spent most of our time on trying to figure out how to link the combinational and sequential systems together, and where to use input, output, wire, or register.

One suggestion I would like to make is for the Professor and/or the TA's to give a crash course on how to create a sequential system in verilog(flip flops). This way it will be easier for us to write the code without wasting too much time on figuring everything out online.

## (7) Team Member Contributions

*Teamwork requires that each member be responsible and assume a relatively equal share of workload. In this section, a detailed description on each member's responsibility and contribution should be presented clearly, including an estimate of percentage of efforts on the project and a summary list of each member in the project. Each member requires to review and sign the final report on the cover page.*

During this project, we both worked an equal amount in all areas. Joshua worked on finding the minimal expressions, and Cody helped check all the work and debug. We both worked the same amount trying to figure out how to code the system in verilog, and did the waveforms together. We both split up the report and did an equal amount of work. We both say that each person contributed 50% to the project and everything was done without any big issues.



## (8) Appendix - The Detailed Design Worksheet

*This part must be included in your report and must show the complete worksheet during the pencil-and-paper design. It should contain:*

*8.1 inputs, outputs, and states of the system.*

*8.2 encoding schemes of inputs, output, and states.*

*8.3 the state diagram and the state table.*

*8.4 minimization procedure for state and output variables by means of the K-map.*

*8.5 final minimal expressions of the logic functions in forms of OR-AND switching expressions and the final schematic of the circuit.*

\*Attached below\*

P07

Joshua Liu (105136031)  
Cody Fan (205102002)

## (8) Appendix - The detailed design worksheet.

## 8.1 Inputs, outputs, and states of the system

Inputs:

Reset:

{ True (T), False (F) }

Coin:

{ Empty (E), Nickel (N), Dime (D) }

Outputs:

Release Gum (RG):

{ T, F }

Return Nickel (RN):

{ T, F }

States: Init initial

5c amount deposited is 5 cents

10c amount deposited is 10 cents

15c amount deposited is 15 cents.

## 8.2 Encoding schemes of inputs, outputs, and states.

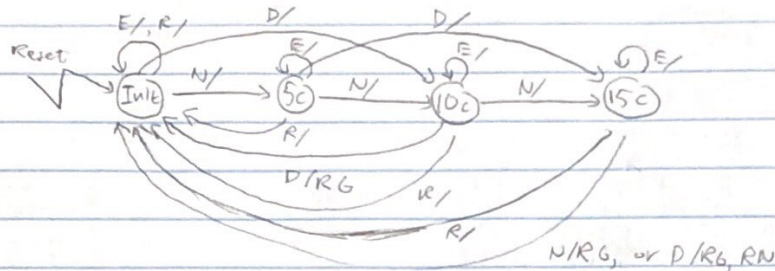
Inputs: Reset	r	Coin	$x_1 x_0$
False	0	Empty (E)	00
True	1	Nickel (N)	01
		Dime (D)	11

Outputs: RG	$z_1$	RN	$z_0$
False	0	False	0
True	1	True	1

States: State	$s_1 s_0$
Init	00
5c	01
10c	11
15c	10

Use: OR-AND network, JK Flip-flops.

### 8.3 the state diagram and the state table



State Table

PS	Inputs					
	$x_1 x_0 = 00$	$x_1 x_0 = 01$	$x_1 x_0 = 11$	$x_1 x_0 = 00$	$x_1 x_0 = 01$	$x_1 x_0 = 11$
00	00,00	01,00	11,00	0 -	0 -	1 -
01	01,00	11,00	10,00	0 -	1 -	1 -
11	11,00	10,00	00,10	-0	-0	-1
10	10,00	00,10	00,11	-0	-1	-1
	NS Outputs R G RN			J, K <sub>1</sub>		
	$x_1 x_0 = 00$	$x_1 x_0 = 01$	$x_1 x_0 = 11$			
	0 -	1 -	1 -			
	-0	-0	-1			
	-0	-1	-1			
	0 -	0 -	0 -			

PS	NS	
	0	1
0	0 -	1 -
1	-1	-0
	J(+)	K(+)

### 8.4

#### Minimization Procedure

$J_1$ $x_1 x_0$	00	01	11	10	
$S_1 S_0$	00	0	0	1	-
	01	0	1	1	-
	11	-	-	-	-
	10	-	-	-	-

$K_1 \backslash x_1 x_0$	00	01	11	10
$S_1 S_0$				
00				
01	-	-	-	-
11	0	0	1	-
10	0	1	1	-

$$J_1 = (x_0)(S_0 + x_1)$$

$$K_1 = (x_0)(S_0' + x_1)$$

$$J_0 = (S_1')(x_0)$$

$$K_0 = (x_0)(S_1 + x_1)$$

$J_0$ $x_1 x_0$	00	01	11	10	
$S_1 S_0$	00	0	1	1	-
01	-	-	-	-	-
11	-	-	-	-	-
10	0	0	0	-	-

$K_0$ $x_1 x_0$	00	01	11	10
$S_1 S_0$	00	-	-	-
	01	0	0	1
	11	0	1	1
	10	-	-	-



Outputs

RG

$s_1 s_0 \backslash x_1 x_0$	00	01	11	10
00	0	0	0	-
01	0	0	0	-
11	0	0	1	-
10	0	1	1	-

RN

$s_1 s_0 \backslash x_1 x_0$	00	01	11	10
00	0	0	0	-
01	0	0	0	-
11	0	0	0	-
10	0	0	1	-

$$RG = x_0 s_1 (s_0' + x_1)$$

$$RN = x_1 s_0' s_1$$

8.5 Final minimal expressions and schematic ← next page.

$$\bar{J}_1 = (x_0) (s_0 + x_1)$$

$$K_1 = (x_0) (s_0' + x_1)$$

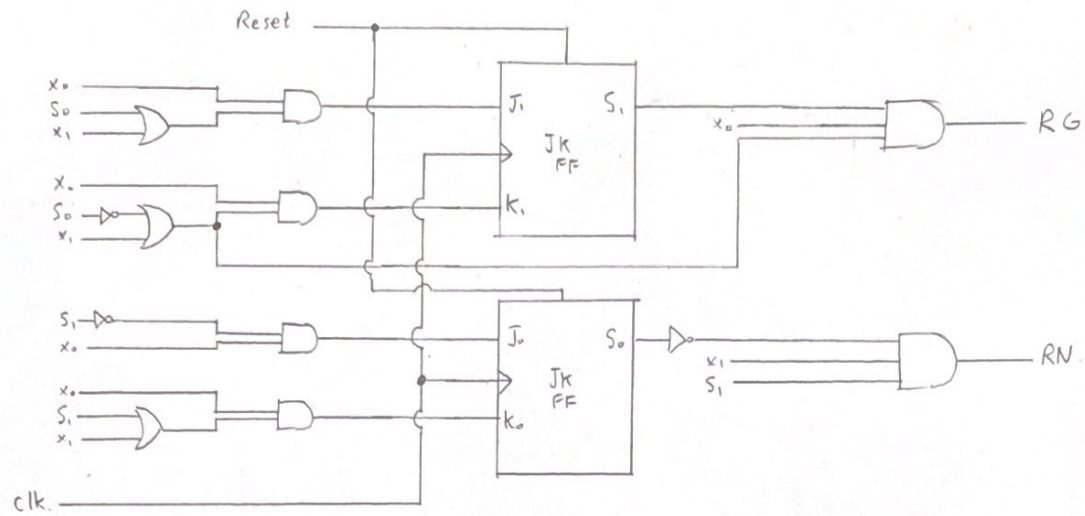
$$J_0 = (s_1') (x_0)$$

$$K_0 = (x_0) (s_1 + x_1)$$

$$RG = x_0 s_1 (s_0' + x_1)$$

$$RN = x_1 s_0' s_1$$

# iKon Vending Machine schematic (2 JK Flip Flops).



(RTL Schematic from Xilinx in Section 3).