CSED311 Lab Report

2023/03/12

LAB1: RTL Design

과목	컴퓨터구조
학번	20180340, 20200195
이름	김재진, 이재윤
담당교수	김광선
제출기간	2023-3-14 23:59:00

1. Introduction

동전, 선택한 아이템의 종류, 아이템의 반환 여부가 input 으로 주어지며, 그에 따라 선택가능한 아이템의 종류, 배출한 아이템, 반환되는 잔돈을 output 으로 출력하는 Vending machine 을 제작해 봄으로써, Vivado 플랫폼 과 Verilog 언어에 대한 친숙함을 늘리고, Finite State Machine 개념에 대한 이해도를 늘리는 것을 목표로 한다.

Vending Machine 의 동작에 관한 use-case 는 다음과 같다.

- **Vending machine use-case**

Assumption: infinite item, change

Sequence
1. Insert money (available money unit: 100, 500, 1000 won) and initialize waiting time(=100)
2. Vending machine shows all available items where (item cost <= current money)
  2.a. decrease waiting time(-1)

3.a. Insert money within the waiting time
  3.a.1. Go to 2 and initialize waiting time(=100)
3.b. Select an item within the waiting time
  3.b.a. Case1: the item is available
    3.b.a.1. The item is dispensed
    3.b.a.2. Go to 2 and initialize waiting time(=100)
  3.b.b. Case 2: the item is unavailable
    3.b.b.1. Nothing happens. Waiting time is not reset.
3.c. No input within the waiting time
  3.c.1 Return changes

a*. Whenever press the return button
  a*.1. Return changes
  a*.2. Go to 1

Figure 1. Vending machine 의 sequence

2. Design

2.1 State Diagram

State 를 4 개로 구성하여 각각의 state 를 전환하는 next state logic 을 구현하였고, 현재 state 와 input 에 의해 output 이 결정되는 mealy machine 으로 FSM 을 설계했다.

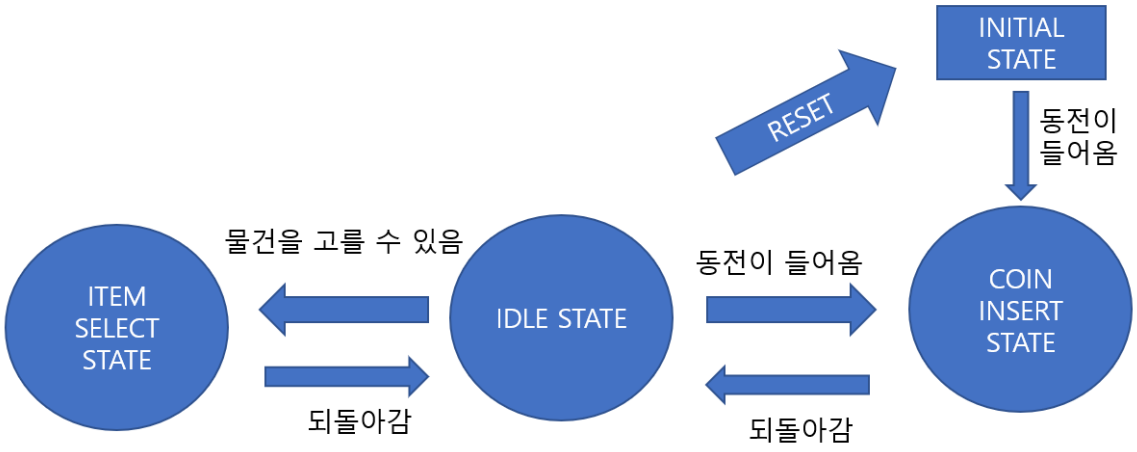

Figure 2. State Diagram

State 는 INIT, IDLE, INSERT, SELECT 로 나눌 수 있다. INIT state 는 Vending machine 의 초기 state 로 Vending machine 에 입력한 금액, 선택가능한 아이템의 종류, 배출된 아이템의 종류를 모두 0 으로 초기화한다. IDLE state 는 Vending machine 의 대기시간이 감소하는 state 로 선택가능한 아이템의 종류와 반환한 총 금액을 업데이트한다. INSERT state 는 동전의 입력을 처리하는 state 로 입력된 동전의 금액을 고려하며 입력된 총 금액을 업데이트한다. SELECT state 는 현재 잔액 범위에서 선택가능한 아이템을 배출하고, 빠져나간 총 금액을 업데이트한다. 또한 INSERT 와 SELECT state 의 경우, 1 clock cycle 이 지나면 다시, IDLE state 로 되돌아간다. 또한 reset 신호가 들어왔을 시 현재 state 에 상관없이 next state 는 INIT state 가 된다. 각각의 state 별로 서로 전환이 가능하게끔 디자인하였다.

2.2 Module Level

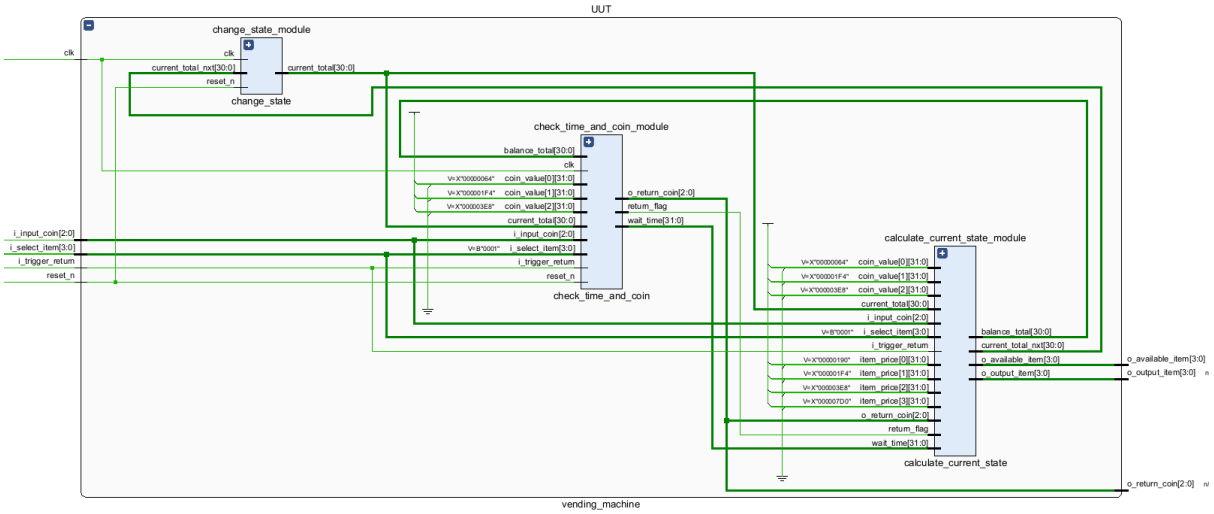

Figure 3. RTL abstraction

A. calculate_current_state

next state 와 output(o_output_item, o_available_item, input_total, output_total, return_total, balance_total)을 현재 state 와 input 에 따라 결정한다. Output 을 결정하는 부분을 간략하게 설명하겠다. 현재 state 혹은 check_time_and_coin 모듈에서 보낸 return_flag input 이 변할 때 마다 always block 이 실행된다. Always block 안에서 현재 state 가 INIT 이면, 모든 output 값이 0 이 되도록 초기화한다. IDLE state 에서는 o_available_item 과 return_total 을 계산한다. INSERT state 에서는 input_total 을, SELECT state 에서는 o_output_item 과 output_total 을 계산한다. Always block 밖에서 Input_total 에 output_total 과 return_total 을 뺀 값을 assign 구문을 활용하여 balance_total 에 할당하고 이를 output 으로 check_time_and_coin 모듈에 보내준다.

B. check_time_and_coin

해당 모듈에서는 wait_time 과 o_return_coin 그리고 return_flag 을 output 으로 뽑아낸다. 먼저 wait_time 은 clock 이 positive edge 마다 1 씩 감소하며 wait_time 이 0 이 되면 더 이상 감소하지 않게 설계하였다. 만약에 reset 신호가 low 가 되면 wait_time 은 `kWaitTime(100)으로 초기화되며

o_return_coin 에는 3'b000 을 할당한다. 현재 state 가 INIT, INSERT, SELECT 라면 wait_time 을 `kWaitTime 으로 초기화한다.

또한 clock 의 positive edge 마다 i_trigger_return 신호가 들어오는 지 혹은 wait_time 이 0 이 되었는 지를 체크한다. 만약에 두 조건 중 하나가 참이라면, 현재 balance_total 을 확인하여 그에 맞게 o_return_coin 을 업데이트하고 return_flag 를 0 과 1 로 toggle (return_flag = !return_flag)시켜 calculate_current_state 모듈에 전달해준다.

C. change_state

Clock 의 positive edge 마다 state 를 next state 로 업데이트한다. 만약 reset 신호가 low 가 되면 현재 state 를 INIT state 로 초기화한다.

3. implementation

```
input clk;
input reset_n;

input [`kNumCoins-1:0] i_input_coin;
input [`kNumItems-1:0] i_select_item;
input i_trigger_return;

output [`kNumItems-1:0] o_available_item;
output [`kNumItems-1:0] o_output_item;
output [`kNumCoins-1:0] o_return_coin;
```

```
// Do not modify the values.
wire [31:0] item_price [`kNumItems-1:0]; // Price of each item
wire [31:0] coin_value [`kNumCoins-1:0]; // Value of each coin
assign item_price[0] = 400;
assign item_price[1] = 500;
assign item_price[2] = 1000;
assign item_price[3] = 2000;
assign coin_value[0] = 100;
assign coin_value[1] = 500;
assign coin_value[2] = 1000;

// Internal states. You may add your own net variables.
wire [`kTotalBits-1:0] current_total;
```

```
// Internal states. You may add your own net variables.
wire [`kTotalBits-1:0] current_total;

// Next internal states. You may add your own net variables.
wire [`kTotalBits-1:0] current_total_nxt;


// Variables. You may add more your own net variables.
wire [`kTotalBits-1:0] input_total, output_total, return_total;
wire [31:0] wait_time;

wire [`kTotalBits-1:0] balance_total;

wire [0:0] return_flag;
```

Figure 4. vending_machine.v

우선 skeleton code 에서 사용처가 명시되지 않았거나, 구현 과정에서 새로 선언된 와이어들에 대해 설명하겠다. current_total/current_total_nxt 은 현재/next state 를 모듈 간에 전달해주는 역할로 사용되었다. 값이 2'b00 이면 INIT, 2'b01 이면 IDLE, 2'b10 면 INSERT, 2'b11 이면 SELECT state 를 뜻한다. balance_total 은 현재 자판기에 남아있는 잔금을 나타낸다. return_flag 는 현재 동전이 반환되었음을 알려주는 1 비트 신호로, o_return_coin 이 동일한 값을 유지할 경우 이를 다른 모듈에서 여러 차례 반환된 것으로 인식하지 못하는 문제를 해결하기 위해 도입하였다.

이후로는 각 모듈의 소스코드에서 주요 부분을 설명하겠다.

A. calculate_current_state

```
always @(i_select_item or i_input_coin) begin

    case(current_total)
        `INIT: begin
            if (i_input_coin) begin
                current_total_nxt = `INSERT;
            end
            else begin
                current_total_nxt = `INIT;
            end
        end

        `IDLE: begin
            if (i_input_coin) begin
                current_total_nxt = `INSERT;
            end

            else if (i_select_item) begin
                is_select_possible = 1'b0;

                for (i = 0; i < `kNumItems; i = i + 1) begin
                    if (i_select_item[i] && (balance_total >= item_price[i]))
                        is_select_possible = 1'b1;
                    end else;
                end

                if (is_select_possible) begin
                    current_total_nxt = `SELECT;
                end
                else
                 begin
                    current_total_nxt = `IDLE;
                end
            end
```

always block 이 i_select_item 과 i_input_coin 이 변했을 때 실행되도록 구현하였다. IDLE state 에서 SELECT state 로 넘어갈 때 현재 잔액보다 큰 금액의 아이템을 선택하면 state 전환이 안되게끔 구현했다. i_select_possible 신호가 1'b1 이 되면 SELECT state 로 넘어가고 아니면 IDLE state 를 유지한다.

B. check_time_and_coin

```
always @(posedge clk) begin
    // TODO: o_return_coin
    o_return_coin = 0;

    if (i_trigger_return || !wait_time) begin
        temp = 0;

        for(i = `kNumCoins - 1; i >= 0; i = i - 1) begin
            if(balance_total - temp >= coin_value[i]) begin
                o_return_coin[i] = 1'b1;
                temp = temp + coin_value[i];
            end
        end
        return_flag = !return_flag;
    end
end

always @(posedge clk) begin
    if (!reset_n) begin
        // TODO: reset all states.
        wait_time <= `kWaitTime;
        o_return_coin <= 0;
    end
    else begin
        // TODO: update all states.
        if (wait_time > 0) begin
            wait_time <= wait_time - 'd1;
        end
    end
end
```

잔돈을 반환하는 조건을 검출하기 위해서는 clock 에 synchronous 하게 체크를 해야 한다. wait_time 이 clock cycle 마다 감소를 하기 때문이다. 하지만 금액의 종류와 그 개수가 고정이 된다는 보장이 없기에 for 문을 사용해야 하고 따라서 이러한 이유로 clock 에 synchronous 한 always block 에 blocking assignment 를 사용하였다. 테스트 케이스에서 동전의 종류가 100, 500, 1000 원으로 고정되었다는 걸 보장한다면 for 문을 사용하지 않고 blocking assignment 를 활용하여 synchronous sequential logic 을 설계할 수 있다. 추가로 o_return_coin 을 설정하는 방식에 대해 기술하자면, 현재 잔액에서 한 cycle 에 최대한 반환할 수 있는 만큼 반환해준다. 이를 위해 내부 변수로 temp integer register 를 선언해주었다. 또한 앞서 설명했듯이 o_return_coin 이 변하지 않을 때 return_total 이 업데이트되지 않는 현상을 수정하고자 반환 조건을 통과하면 return_flag 신호를 뒤집어서 output 으로 calculate_current_state 모듈에 전달해준다.

4. discussion

Vending machine 을 설계할 때 가장 어려웠던 부분이 잔돈을 반환하는 부분이었다. Verilog coding guide 에서 나와 있듯이 clock 에 동기적으로 혹은 비동기적으로 작동하는 지 여부에 따라 non-blocking assignment 를 쓸 지 blocking assignment 를 쓸 지를 유념하며 디자인하였다. 잔돈을

반환하는 조건이 wait_time 에 종속되고 wait_time 과 FSM 의 state 가 clock cycle 마다 변하기 때문에 잔돈을 반환하는 부분은 non-blocking assignment 로 구현하는 것이 맞다고 생각했다. 하지만 바로 전 항목에서 말했듯이 테스트 케이스의 여러 경우를 고려하고자 for loop 을 사용했고 그러다 보니 blocking assignment 를 사용할 수밖에 없었다. 다음 Lab 에서부터는 hidden test case 가 전부 주어지니 이에 맞게 끔 잘 설계할 수 있다고 생각한다.

추가로, sensitivity list 를 @(posedge clk, i_trigger_return)으로 작성하였더니 반환 버튼이 켜졌을 때 always block 이 두 번 실행되는 현상을 목격했다. (i_trigger_return 이 clock 의 positive edge 에 켜진다) i_trigger_return 을 sensitivity list 에서 제거하며 이러한 현상을 해결하였다.

5. conclusion

FSM 은 물론이고 Verilog coding 에 대해서 많은 것을 배울 수 있는 과제였다.