Lab1 Report

20230683 박한비, 20230642 이채영

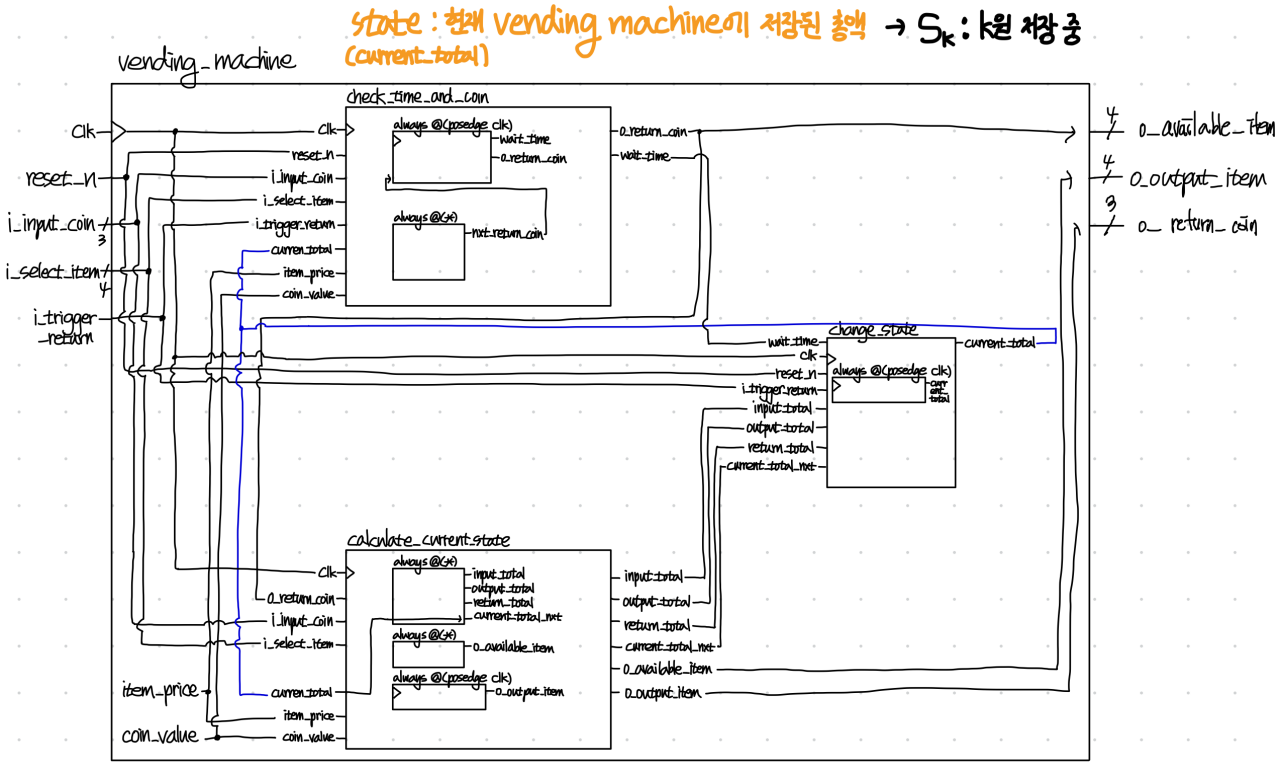
1. Introduction

Lab1에서는 ALU와 간단한 vending machine을 구현한다. 학습 목표는 아래와 같다.

- Verilog의 문법을 습득한다.
- Combinational logic과 sequential logic을 적절하게 이용할 수 있다.
- Finite State Machine 의 개념을 습득하고 moore machine 과 mealy machine 의 동작 차이를 이해한다.

2. Design

Vending machine을 구현하기 위해 그린 설계도는 아래와 같다.



설계도를 보면 알 수 있듯이, vending machine의 output signal인 o_available_item(현재 state에 따라 이용가능한 item 출력), o_output_item(현재 state와 사용자의 상품 선택 입력에 따라 machine 밖으로 나온 item 출력), o_return_coin(현재 state와 사용자의 돈 반환 입력에 따라 machine 밖으로 나온 coin 출력)을 구현하기 위해서 combinational logic과 sequential logic을 적절히 혼합하여 사용하였다.

각각의 output signal을 보자면, o_return_coin은 check_time_and_coin 서브모듈에서, o_available_item과 o_output_item은 calculate_current_state 서브모듈에서 계산하도록 설계하였다.

1) State

이 FSM에서 state는 현재 vending machine에 사용자의 입력에 따라 저장된 총액으로, current_total wire가 해당 state 값을 check_time_and_coin, calculate_current_state 모듈로 전달하여 output 값들이 state 값을 반영하도록 설계하였다. 각각의 output 값들은 state와 사용자의 input의 영향을 받지만 clock 없이 input의 입력만으로는 값이 변하지 않기 때문에 해당 FSM은 moore machine이라는 것을 알 수 있다.

2) check_time_and_coin

해당 모듈에서는 다음과 같은 processs을 구현한다.

- coin 반환까지의 제한시간인 wait_time을 계산(clk signal이 들어올 때마다 감소)하고 사용자의 입력(i_select_item input일 경우 state도 필요), reset_n에 따라 초기화시킨다.
- output o_return_coin을 사용자의 입력(i_trigger_return input) 혹은 타임어택(wait_time == 0)이 있을 경우 계산한다. 또한 reset_n에 따라 초기화한다.

3) calculate_current_state

해당 모듈에서는 다음과 같은 processs을 구현한다.

- current_total을 current_total_nxt와 연결한다.
- change_state에서 다음 state 변환에 이용할 값들인 input_total, output_total, return_total을 사용자의 입력과 state에 따라 계산한다.
- output o_available_item, o_output_item을 사용자의 입력과 state에 따라 계산한다.

4) change_state

해당 모듈에서는 다음과 같은 processs을 구현한다.

- State를 나타내는 current_total을 clock signal이 들어올 때 input_total, output_total, return_total을 이용해 계산하여 state를 업데이트한다.
- State를 reset_n에 따라 초기화시킨다.

3. Implementation

3.1. check_time_and_coin module

입력된 동전(i_input_coin)을 확인하고, 대기 시간(wait_time)이 초과되거나 반환 요청(i_trigger_return)이 들어오면 거스름돈을 계산하여 반환한다.

3.1.1. Combinational logic

```
always @(*) begin
    // TODO: o_return_coin
    remaining_total = current_total;
    nxt_return_coin = o_return_coin;
    if (!reset_n) begin
        // TODO: reset all states.
        nxt_return_coin = 0;
    end

    else if (wait_time == 0 || i_trigger_return) begin
        for (int i = `kNumCoins-1; i >= 0; i = i - 1) begin
            if (remaining_total >= coin_value[i]) begin
                nxt_return_coin[i] = 1;
                remaining_total = remaining_total - coin_value[i];
            end
        end
    end

end
```

거스름돈을 계산하기 위해 반환 후 남은 금액(remaining_total), 거스름돈(nxt_return_coin)을 저장하는 임시 변수를 사용한다. (nxt_return_coin은 이후 sequential logic에서 output으로 반환하기 위해 사용한다.) reset_n 신호를 받으면 거스름돈을 0으로 초기화하고, wait_time이 0이 되거나 반환 요청을 받으면 거스름돈을 계산한다.

3.1.2. Sequential logic

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

    else begin
        o_return_coin <= nxt_return_coin;

        // TODO: update coin return time
        if (i_input_coin) begin
            wait_time <= `kWaitTime;
        end

        else if (i_select_item) begin
            coin_required = 0;
            for (int i = 0; i < `kNumItems; i += 1) begin
                coin_required += i_select_item[i] * item_price[i];
            end
            if (current_total >= coin_required) begin
                wait_time <= `kWaitTime;
            end
        end

        else if (wait_time > 0) begin
            // TODO: update all states.
            wait_time <= wait_time - 1;
        end
    end
end
```

clk에 맞추어 wait_time을 1씩 감소시킨다. 만약 동전 입력(i_input_coin), 물품 선택(i_select_item), reset_n이 있으면 wait_time을 초기화시킨다. 또한 combinational logic에서 계산했던 거스름돈을 output으로 반환한다.(o_return_coin)

3.2. calculate_current_state module

사용자가 투입한 동전(i_input_coin)과 선택한 상품(i_select_item)을 바탕으로 자판기에 있는 현재 금액(current_total)을 업데이트하고, 구매 가능한 상품과(o_available_item) 출력할 상품(o_output_item)을 결정한다.

3.2.1. Combinational logic

```
// Combinational logic for the next states
always @(*) begin
    // TODO: current_total_nxt
    // You don't have to worry about concurrent activations in each input vector (or array).
    // Calculate the next current_total state.
    input_total = 0;
    output_total = 0;
    return_total = 0;

    if (i_input_coin) begin
        for(i = 0; i < `kNumCoins; i = i + 1) begin
            if (i_input_coin[i] == 1) begin
                input_total = input_total + coin_value[i];
            end
        end
    end

    if (i_select_item) begin
        for (i = 0; i < `kNumItems; i = i + 1) begin
            if (current_total >= i_select_item[i] * item_price[i] && i_select_item[i] == 1) begin
                output_total = output_total + item_price[i];
            end
        end
    end

    if (o_return_coin) begin
        for (i = 0; i < `kNumCoins; i = i + 1) begin
            if (o_return_coin[i] == 1) begin
                return_total = return_total + o_return_coin[i] * coin_value[i];
            end
        end
    end

    current_total_nxt = current_total;
end

// Combinational logic for the outputs
always @(*) begin
    // TODO: o_available_item
    // TODO: o_output_item
    o_available_item = 0;
    for(i = 0; i < `kNumItems ; i = i + 1) begin
        if (current_total >= item_price[i]) begin
            o_available_item[i] = 1;
        end
    end
end
```

사용자가 투입한 동전(i_input_coin)에 따라 input 값을 계산하고, 선택한 상품(i_select_item)과 current_total을 고려하여 output 값을 계산하고, 거스름돈으로 반환할 금액을 계산한다. input_total, output_total, return_total은 change_state 모듈에서 current_total을 계산하는데 활용한다. 또한 구매 가능 상품 여부(o_available_item)를 출력한다.

3.2.2. Sequential logic

```
always @(posedge clk) begin
    o_output_item <= 0;
    for (i = 0; i < `kNumItems; i = i + 1) begin
        if (current_total >= i_select_item[i] * item_price[i] && i_select_item[i] == 1) begin
            o_output_item[i] <= 1;
        end
    end
end
```

상품 출력 여부(o_output_item)을 갱신한다.

3.3. change_state module

현재 자판기가 가진 총 금액(current_total)을 동전 투입 금액(input_total), 상품 구매 금액(output_total), 반환 금액(return_total) 정보를 반영하여 업데이트한다. Sequential logic으로만 구성되었다.

3.3.1. Sequential logic

```
// Sequential circuit to reset or update the states
always @(posedge clk) begin
   if (!reset_n || i_trigger_return || wait_time == 0) begin
      // TODO: reset all states.
      current_total <= 0;
   end

   else begin
      current_total <= current_total_nxt + input_total - output_total - return_total;

   end
end
```

리셋(reset_n) 또는 반환 요청(i_trigger_return)이 들어오면 current_total을 0으로 초기화한다. 이 외의 경우에는 input, output, return 값들을 반영하여 current_total을 업데이트한다.

4. Discussion

Verilog를 이용해 vending machine을 구현하며 latch, clk의 올바른 사용에 관련된 에러를 주로 겪었다.

4.1. Latch

combinational logic에서 의도하지 않은 latch 구조인 경우, latch 추론이 발생할 수 있음을 알리는 경고이다. if-else if와 같은 조건문을 사용했을 때 조건부 할당이 누락되면

발생한다. 모든 경우에 대해 명시적으로 값을 할당하면 되므로, always @(*)문의 처음 부분에 값을 초기화하는 방식으로 에러를 해결하였다.

4.2. o_output_item 처리

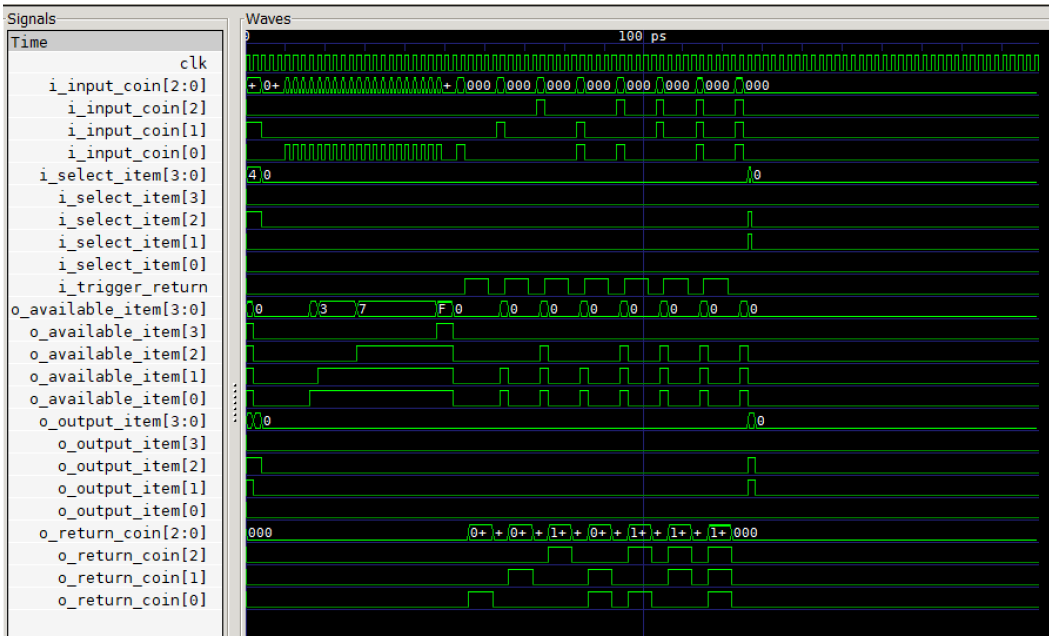
처음 구현을 할 때는 o_output_item을 combinational logic에서 처리하였는데, 값이 제대로 업데이트되지 않고 모든 비트가 0으로만 출력되었다. Combinational logic의 경우 입력 값이 바뀔 때마다 즉시 출력 값을 변경하는데, o_output_item은 상품이 선택된 후 클럭이 한번 지나야 제대로 출력되어야 한다. 따라서 combinational logic을 사용했을 때 입력이 순간적으로 변했다가 돌아올 경우 출력이 유지되지 않을 수 있다.

또한 사용자의 상품 선택 입력(i_select_item)이 1이 될 때 o_output_item이 설정되는데, i_select_item은 짧은 순간만 1이 될 수 있다. Combinational logic을 사용하면 i_select_item이 0으로 돌아갔을 때 o_output_item도 즉시 0이 되어 값이 제대로 반영되지 않을 수 있다. 따라서 Sequential logic으로 바꾸어 구현하니 값이 제대로 반영되었다.

5. Conclusion

```
### SIMULATING ###
 initial test sim_time: 8
PASSED  : o_available_item: 0, expected 0

 Available item test sim_time: 10
PASSED : available item: 1, expected 0b0001
PASSED : available item: 3, expected 0b0011
PASSED : available item: 7, expected 0b0111
PASSED : available item: 15, expected 0b1111

 Return_coin test sim_time: 52
PASSED : return coin: 1, expected 0b001
PASSED : return coin: 2, expected 0b010
PASSED : return coin: 4, expected 0b100
PASSED : return coin: 3, expected 0b011
PASSED : return coin: 5, expected 0b101
PASSED : return coin: 6, expected 0b110
PASSED : return coin: 7, expected 0b111

 Select item test sim_time: 122
PASSED : output item: 0b110, expected 0b0110

success: 13 / 13
```

구현한 vending machine이 주어진 testbench를 모두 통과하였다.



파형 또한 의도한대로 출력된 것을 확인할 수 있다.

Lab1을 통해 combinational logic, sequential logic과 non-blocking, blocking의 차이를 이해할 수 있게 되었다. 또한 Finite State Machine, Moore/Mealy Machine 등 디지털 시스템 설계의 중요한 개념들을 복습하고, HDL을 이용한 설계에 익숙해질 수 있었다.