Lab 2 - RTL Design: Vending Machine

20160595 수학과 최규민
20160169 물리학과 최수민

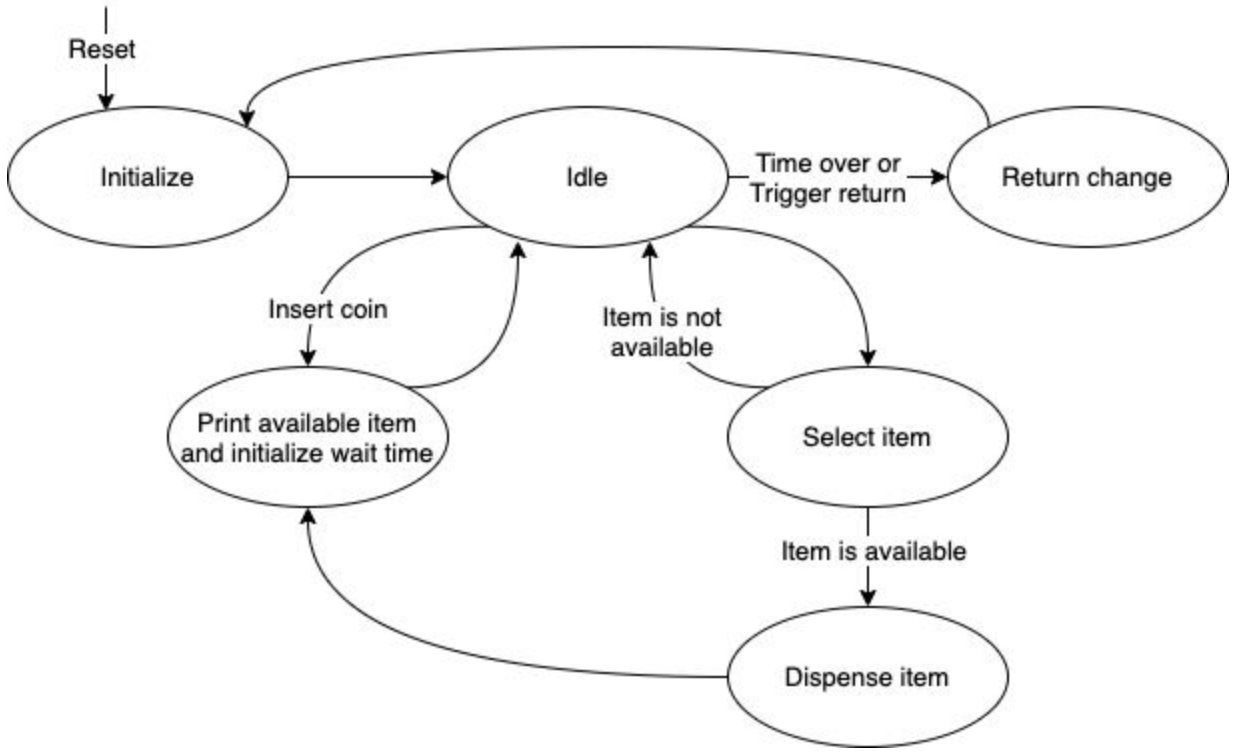
1. Introduction

이번 과제에서는 Simple Finite State Machine인 Vending Machine을 Verilog로 구현한다. Vending Machine 모듈의 input은 여러 종류의 coin과 item, clock signal, reset signal, return button으로 이루어져 있고, output으로는 input에 따라 Machine 내부에서 current price state를 계산하여 구매 가능한 item을 나타내고, 적절한 출력 item과 return coin을 지급한다. Return 버튼을 누르거나, 일정 시간 input이 없을 시 return coin을 지급하고 리셋한다.

이번 과제를 통해 배워야 하는 것은 다음의 두 가지로 요약할 수 있다.

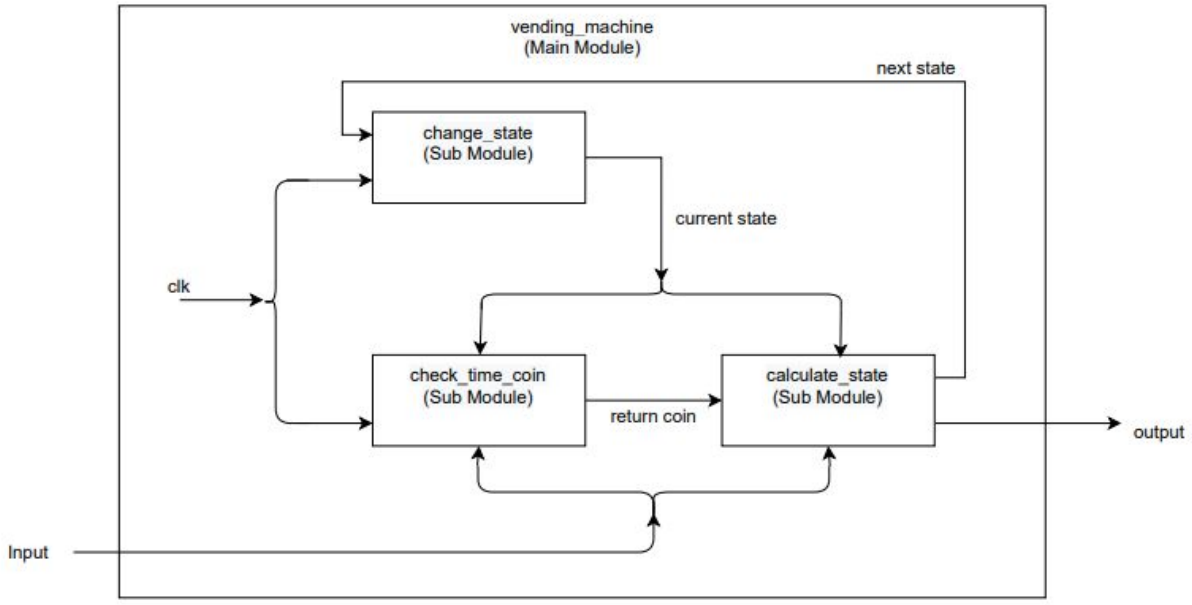
1. RTL 구현을 통해 Synchronous circuit에서 Register와 Combinational logic의 역할을 이해한다.
2. Finite State Machine 구현을 통해 clock의 개념과 Sequential Logic의 설계를 이해한다.

2. Design



[그림 1] Vending Machine의 state 변화를 나타낸 다이어그램.

모듈을 구현하기 전, 먼저 state 변화가 어떤 순서로 일어나는지를 그림 1의 다이어그램을 그려 파악했다. Reset signal이 들어오면 초기화를 한 뒤 휴지모드로 진입한다. 이후 코인이 들어오거나 아이템을 선택함에 따라 각 단계가 진행된다.



[그림 2] Vending Machine의 Block diagram.

모듈은 크게 메인 모듈인 vending_machine과, 서브 모듈인 check_time_and_coin, calculate_current_state, change_state로 나누었다. check_time_and_coin 모듈은 시간이 초과됐거나 return button이 눌렸는지를 확인해 전액을 반환할지를 결정한다. calculate_current_state 모듈은 동전이 들어왔거나 어떤 물건이 선택됐는지에 따라 다음 잔액을 계산하고 물건을 지급할지를 결정한다. change_state 모듈은 calculate_current_state 모듈에서 계산한 잔액을 현재 상태에 반영한다.

2.1. vending_machine 모듈

각 coin의 value와 item의 price를 내부 변수로 저장하고 있으며, 모듈의 input과 함께 이 값들을 서브 모듈로 전달하고 서브모듈의 output을 자신의 output으로 내보낸다. 실질적인 state의 변화와 output 값의 계산은 서브 모듈에서 이루어진다.

2.2. calculate_current_state 모듈

- 들어온 돈과 선택한 아이템과 관련된 state와 output을 계산하는 combinational logic을 구현한다.

해당 로직은 input은 자판기에 있는 잔액(current_total), 들어온 돈의 종류(i_input_coin), 선택한 아이템(i_select_item)이고, 이로부터 다음 잔액(current_total_nxt), 지급된 아이템의 종류(o_output_item), 그리고 선택 가능한 아이템의 종류(o_available_item)을 계산한다.

만약 물건이 선택된다면 물건의 가격이 현재 자판기의 잔액보다 작거나 같을 때 current_total_nxt는 current_total에서 해당 물건의 가격을 뺀 값이 되고 o_output_item은 선택된 물건이 된다. 선택된 물건의 가격이 현재 자판기의 잔액보다 클 경우 current_total_nxt는 current_total값이 되고, o_output_item은 아무것도 없다(0).

만약 돈이 들어온다면 current_total_nxt는 current_total에 들어온 돈의 가격을 더한 값이 된다.

선택 가능한 물건의 종류(o_available_item)는 항상 위의 계산을 마친 후의 current_total_nxt값보다 작거나 같은 물건들이 된다. 만약 돈이 들어왔거나 물건을 지급하는 경우 o_available_item을 보여준다.

2.3. check_time_and_coin 모듈

- Wait_time을 관리하고, coin return 조건이 되면 return 가능한 coin을 확인하여 calculate_current_state 모듈에 보내준다.

초기 reset시에 wait_time을 100으로 초기화하고, 이후 Positive clock edge마다 wait_time을 1씩 감소시킨다. Idle state에서는 input_coin이 있을때 마다, 그리고 output 가능한 아이템을 선택하여 아이템이 output될 때마다 wait_time을 다시 100으로 초기화 시켜준다.

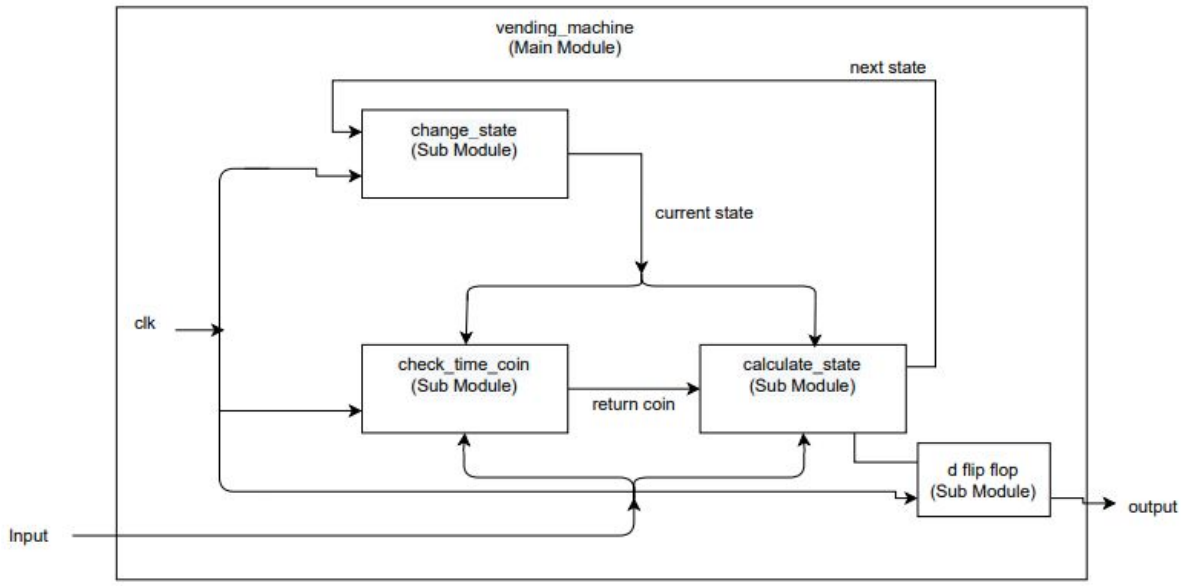
Wait_time이 0이 되어 time over가 되거나, input으로 trigger_return이 들어오면 Return change state로 전환하여 가능한 return coin이 어떻게 되는지 확인하여 calculate_current_state로 o_return_coin 정보를 보내준다. Return change state에서 가능한 모든 coin이 반환되어 current_state가 0이 되었을 때 내부 return_done 시그널을 이용해 초기 state인 initalize state로 돌아가 초기화해준다.

2.4. change_state 모듈

- Clock에 synchronous하게 current_total state를 업데이트해준다.

Positive clock edge마다 reset_n이 0이면 current_total을 초기화하고, 아니면 current_total을 current_total_nxt로 업데이트한다.

3. Implementation



[그림 3] 구현을 진행하면서 수정한 Vending machine의 Block diagram. Output을 전달하기 전에 d flip flop을 추가하여 output이 synchronous하게 전달되도록 변경했다.

구현 도중 o_available_item이 asynchronous하게 출력되는 것을 확인해서, vending_machine의 모든 output들이 synchronous하게 작동하도록 d flip-flop을 도입하였다.

3.1. check_time_and_coin 모듈

```
// update wait time. else statement to avoid underflow
if (wait_time > 0)
        wait_time <= wait_time - 1;
else
        wait_time <= 0;

// return coin when wait_time over or trigger_return
if ((wait_time == 0) || i_trigger_return)
        return_coin_signal <= 1;
```

[그림 4] check_time_and_coin 모듈의 코드. wait_time을 감소시키고, coin을 return하는 작업이 포함되어 있다.

Positive clock edge마다 wait_time을 1씩 감소시킨다. Else문은 wait_time의 underflow를 방지한다. Wait_time이 0이 되거나, trigger_return시 coin을 return시키는 작업을 시작한다.

```
if(return_coin_signal) begin

        //return available coin
        temp_current = current_total;
        for (i = 0; i < `kNumCoins; i = i + 1) begin
                if (coin_value[i] <= temp_current) begin
                        o_return_coin[i] = 1;
                        temp_current = temp_current - coin_value[i];
                end
                else
                        o_return_coin[i] = 0;
        end

        // when return is done, reset
        if (current_total == 0)
                return_done = 1;
        else
                return_done = 0;
```

[그림 5] check_time_and_coin 모듈의 코드. 남아있는 돈이 0이 될 때까지 coin을 반환한다.

Return_coin_signal이 오면 현재 return 가능한 coin들의 종류를 계산하여 calculate_current_state로 연결시키고, return이 전부 끝나면 다시 리셋시켜준다.

3.2. calculate_current_state 모듈

```
// Combinational logic for the next states
always @(*) begin
        // when coin is returned, update current_total_nxt
        if (return_coin != 0) begin
                current_total_nxt = current_total - return_coin;
        end
        // when item is selected and it is available
        else if (selected_item != 0 && selected_item <= current_total) begin
                current_total_nxt = current_total - selected_item;
                o_output_item = i_select_item;
        end
        // when coin is inserted or item is selected but unavailable
        else begin
                current_total_nxt = current_total + inserted_coin;
                o_output_item = `kNumItems'b0;
        end

        // check available items using current_total_nxt
        for (i = 0; i < `kNumItems; i = i + 1) begin
                if (item_price[i] <= current_total_nxt)
                        o_available_item[i] = 1;
                else
                        o_available_item[i] = 0;
        end

end
```

[그림 6] calculate_current_state 모듈의 코드.

Input과 check_time_and_coin에서 결정된 return해야 할 coin 종류에 따라서 다음 state가 될 current_total_nxt를 업데이트한다. 업데이트된 current_total_nxt를 이용하여 available_item을 체크해 output해주고, coin이 들어올 때 혹은 item이 dispense될 때 available_item을 print해준다.

3.3. change_state 모듈

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

[그림 7] change_state 모듈의 코드.

Positive clock edge마다 다음 state로 update해주고, reset이 input으로 들어오면 0으로 초기화시켜준다.

3.4. D flip-flop 모듈

```
// d flip flop to make output synchronous
always @(posedge clk) begin
    if (!reset_n) begin
        o_available_item <= `kNumItems'b0;
        o_output_item <= `kNumItems'b0;
        o_return_coin <= `kNumCoins'b0;
    end
    else begin
        o_available_item <= o_available_item_async;
        o_output_item <= o_output_item_async;
        o_return_coin <= o_return_coin_async;
    end
        // print available items
        if (i_input_coin || o_output_item)
            $strobe("o_available_item = %0b", o_available_item);
end
```

[그림 8] d_flip_flop 모듈의 코드.

Output이 clock에 synchronous하게 만들기 위해서 d flip-flop을 도입한다. 이 모듈은 positive clock edge마다 output을 업데이트 시켜준다.

4. Discussion

- 초기 구현 시 o_available_item 을 출력하는 부분이 한 사이클씩 빠르게 출력되는 것이 관찰되었다. 이유를 고민하던 중 우리의 구현이 Finite State Machine 중 Mealy Machine임을 확인하였고, 이는 output이 clock에 비동기적(asynchronous)하게 변경되는 특징을 가진다. 그 해결책으로 combinational logic에서 output을 d flip-flop을 거치게 하여 모든 output을 synchronous하게 만들어, o_available_item 이슈를 해결할 수 있었다.

- o_available_item 출력 시 display가 아니라 strobe를 사용했다. display는 그 당시의 값을 출력하는데, strobe는 해당 timestamp가 끝난 시점의 값을 출력한다. 따라서 non-blocking assignment가 있을 경우 display를 사용했을 때 의도한 값이 아닌 다른 값이 출력될 수 있다. 본 구현에서 o_available_item이 synchronous하게 구현되어서 값을 제대로 출력하려면 strobe를 사용해야 했다.

5. Conclusion

RTL 구현을 통해 Synchronous circuit에서 Register와 Combinational logic의 역할을 이해한다. Finite State Machine인 Vending machine 구현을 통해 clock의 개념과 Sequential Logic의 설계를 이해할 수 있었다.