20180188 서승연 20180373 김현지

Lab2. RTL design

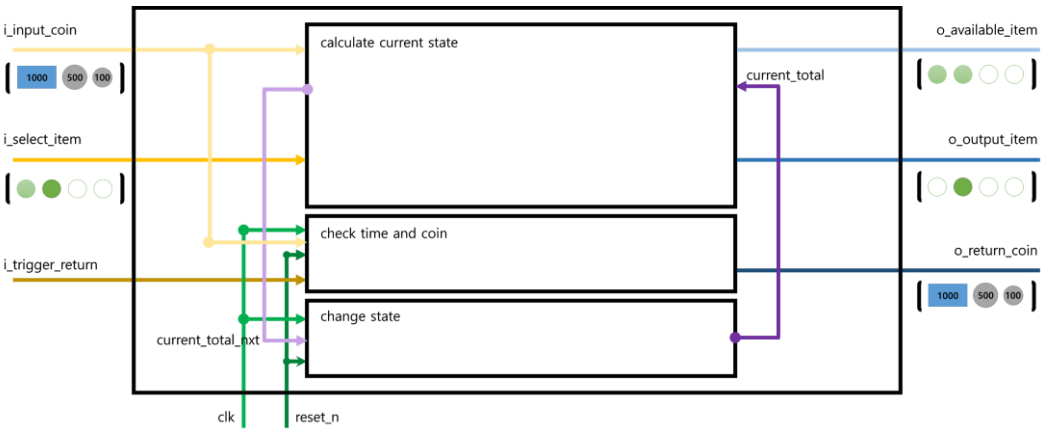
Introduction

이번 과제의 목표는 combinational logic과 sequential logic의 각 역할을 이해하고 Verilog로 간단한 FSM인 vending machine Register-Transfer Level을 구현하는 것이다. RTL은 HDL로 synchronous circuit을 구현하는 디자인 방법이다. synchronous circuit은 register와 combination logic으로 이루어져 있다. 이번 과제를 통해 위 개념들을 이해하고 RTL design을 작성할 수 있다.

Design

Top-Level Module에서는 coin, item, return trigger를 input으로 받아 available item, output item, return coin을 계산한다. Submodules는 check time coin, calculate current state, change state가 있다. *check time coin*은 1) current total과 coin value 정보를 이용하여 return coin을 계산하고 2) wait time을 업데이트하는 sequential logic이다. *calculate current state*는 1) input total, output total, return total을 이용하여 current total nxt를 계산하고 2) available item, output item의 output들을 계산하는 combinational logic이다. *change state*는 조건에 따라 state를 reset하거나 업데이트하는 sequential logic이다.

Top level module인 vending machine에서 submodule을 연결하여 동작하도록 한다. wire 등 세세한 연결을 모두 표시하기엔 한계가 있어 Top level module의 input/output과 state와 관련된 variable의 연결만 표시하였다.



Implementation

1. check_time_and_coin

```
module check_time_and_coin(i_input_coin, i_trigger_return, clk, reset_n, current_total,
                           coin_value, o_output_item, o_return_coin, wait_time);
    input clk;
    input reset_n;

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

    input i_trigger_return;
    input [31:0] coin_value [`kNumCoins-1:0];
    input [`kTotalBits-1:0] current_total;

    output reg [`kNumCoins-1:0] o_return_coin;
    output reg [31:0] wait_time;

    integer i;

    // initiate values
    initial begin
        // TODO: initiate values
        o_return_coin = `kNumCoins'b000;
        wait_time = 0;
    end

    always @(*) begin
        // TODO: o_return_coin
        o_return_coin = `kNumCoins'b000;
        if (i_trigger_return || (wait_time > `kWaitTime)) begin
            if (current_total >= coin_value[2])      o_return_coin[2] = 1'b1;
            else if (current_total >= coin_value[1]) o_return_coin[1] = 1'b1;
            else                                     o_return_coin[0] = 1'b1;
        end
    end

    always @(posedge clk) begin
        if (reset_n) begin
            if (i_input_coin || o_output_item) wait_time <= 0;
            else                               wait_time <= wait_time + 1;
        end
    end
endmodule
```

- initial 구문에서 return coin과 wait time을 초기화한다.
- 1st always 구문
  return coin을 초기화하고 current total의 양에 따라 return coin을 설정한다. trigger return 이 왔거나 timeout이 될 때 값을 설정한다.
- 2nd always 구문
  clk 값이 변할 때마다 wait time을 업데이트한다.

2. calculate_current_state

```
initial begin
    // TODO: initiate values
    o_available_item = `kNumItems'b0000;
    o_output_item = `kNumItems'b0000;

    current_total_nxt = `kTotalBits'd0;

    input_total  = `kTotalBits'd0;
    output_total = `kTotalBits'd0;
    return_total = `kTotalBits'd0;

end

// 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 = `kTotalBits'd0;
    for(i = 0; i < `kNumItems; i = i + 1) begin
        if (i_input_coin[i]) begin
            input_total = coin_value[i];
            i = `kNumItems;
        end
    end

    output_total = `kTotalBits'd0;
    for(i = 0; i < `kNumItems; i = i + 1) begin
        if (o_output_item[i]) begin
            output_total = item_price[i];
            i = `kNumItems;
        end
    end

    return_total = `kTotalBits'd0;
    for(i = 0; i < `kNumCoins; i = i + 1) begin
        if (o_return_coin[i]) begin
            return_total = coin_value[i];
            i = `kNumCoins;
        end
    end

    current_total_nxt = current_total + input_total - output_total - return_total;
end
```

- initial 구문에서 value들을 초기화한다.
- 1st always 구문
  input total은 input coin의 값을 더한 값이다. 이를 초기화한다. output total은 output item의 가격의 합이다. return total은 return coin과 coin value를 활용해 결정한다. 이렇게 설정한 변수들을 이용하여 current total nxt 값을 초기화하여 state를 변경한다.

```
// Combinational logic for the outputs
always @(*) begin
    // TODO: o_available_item
    o_available_item = `kNumItems'b0000;
    for(i = 0; i < `kNumItems; i = i + 1) begin
        if (current_total >= item_price[i]) o_available_item[i] = 1'b1;
    end

    // TODO: o_output_item
    o_output_item = `kNumItems'b0000;
    for(i = 0; i < `kNumItems; i = i + 1) begin
        if (i_select_item[i] && o_available_item[i]) o_output_item[i] = 1'b1;
    end

end
```

-

- 2nd always 구문

  output 값을 설정하는 구문이다. 조건에 따라 값을 설정한다. combinational logic의 always 구문에서는 blocking assignment를 사용한다.

3. change_state

```
module change_state(clk,reset_n,current_total_nxt,current_total);

    input clk;
    input reset_n;
    input [`kTotalBits-1:0] current_total_nxt;
    output reg [`kTotalBits-1:0] current_total;

    // Sequential circuit to reset or update the states
    always @(posedge clk) begin
        if (!reset_n) begin
            // TODO: reset all states.
            current_total <= `kTotalBits'd0;
        end
        else begin
            // TODO: update all states.
            current_total <= current_total_nxt;
        end
    end
endmodule
```

- always 구문

  reset n의 값에 따라 state를 reset하거나 update한다. update 시에는 calculate current state 에서 설정한 current total nxt 값으로 업데이트한다.

Discussion

wait time을 설정하는 과정에서 counter와 같은 개념으로 'kWaitTime으로 초기화 한 후 값을 감소시키며 0이 되는 순간을 감지하려고 했다. 하지만 wait time 감소와 비교가 병렬적으로 일어나기 때문에 wait time 값이 계속 감소하여 0에 머무르지 않아 비교구문을 감지 하지 못하는 상황이 생겼다. 처음에 TimeoutFlag 같은 변수를 선언하여 0이 된 순간에는 더 이상 wait time이 감소하지 않도록 막으려 하였으나, 간단한 방법으로 0으로 초기화하여 증가시키는 방향으로 구현한 후 wait time이 Max 값이 'kWaitTime보다 클 경우로 비교구문을 변경하여 쉽게 해결할 수 있었다.

Conclusion

combinational / sequential logic의 개념을 이해함으로써 요구하는 기능에 따라 modulization 하는 방법을 익혔다. FSM을 설계하고 이를 구현해 HDL 코딩에 좀 더 익숙해졌다.