2024 Spring CSED311 Lab 1

Vending Machine Report

Team ID: 67735

20220312 박준혁, 20220871 홍지우

명예서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

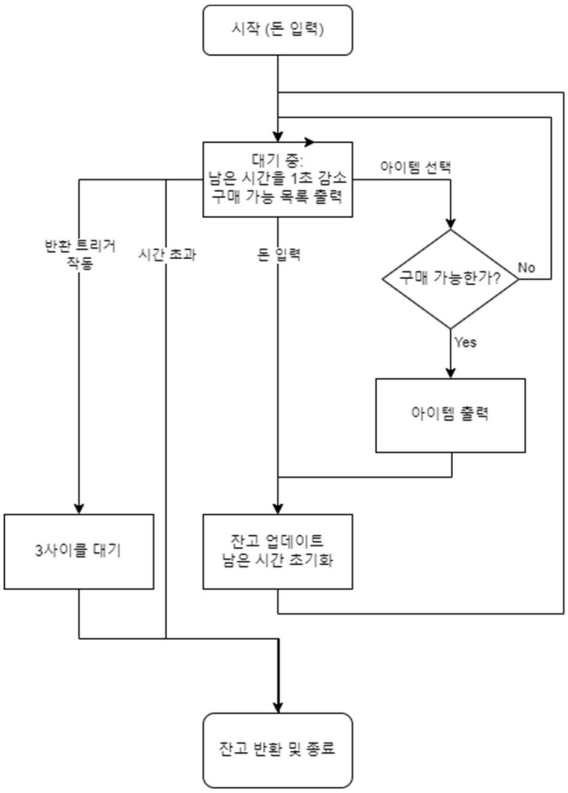
I completed this programming task without the improper help of others.

1 Introduction

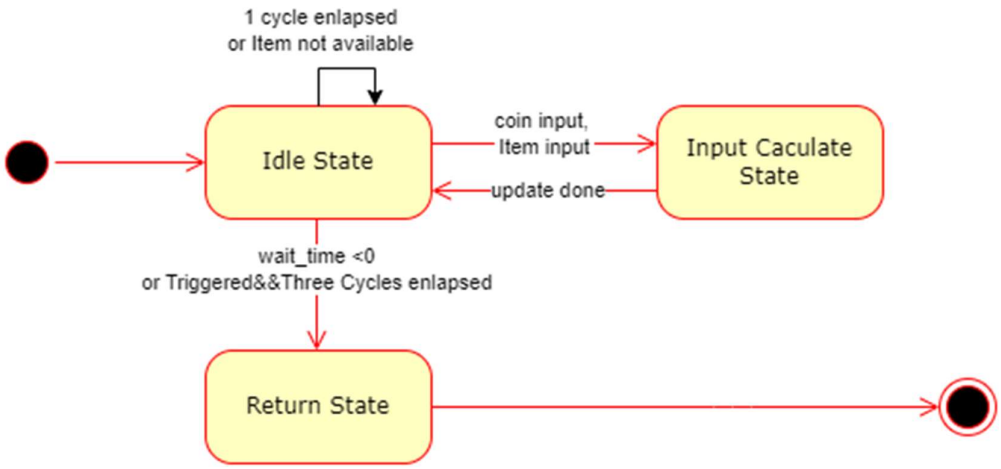
1.1 이번 lab에서는 간단한 finite state machine인 자판기를 구현하였다. 돈 입력, 아이템 고르기, 타이머, 동전 반환을 Moore Machine으로 구현하였다.

2 Design

2.1 순서도



2.2 State Transition Diagram



2.3 FSM Type

Moore Machine: Output depends on state only

전부 계산하고 나서 clk에 맞추어서 업데이트하는 식으로 디자인 하였다.

2.4 Module Design

vending_machine 안에 check_time_and_coin, calculate_current_state, change_state 3개의 모듈이 들어가 있다.

2.4.1 check_time_and_coin

시간을 동기적으로 1씩 줄인다.

시간이 다 되거나 반환 트리거가 작동하면 잔돈을 반환한다.

돈 입력이 있거나, 아이템 입력이 있고 구매 가능할 시 시간을 초기화한다.

2.4.2 calculate_current_state

돈 입력, 아이템 입력, 반환 값들을 계산한다.

또한 구매 가능 목록, 구매한 아이템을 출력한다.

2.4.3 change_state

바뀐 값들을 토대로 잔고를 업데이트한다.

리셋된 경우 잔고를 0으로 만든다.

3 Implementation

3.1 check_time_and_coin

3.1.1 Combinational Logic

wait_time 이 0이 되거나 트리거되고 나서 3사이클 지나면
반환할 코인 값을 계산한다.

```
always @(*) begin
    temp_return_total=0;
    o_return_coin=0;
    // time over: return
    // triggered: return after three cycles
    // caculate coins
    if(wait_time <0 || threeCyclesCounter ==3) begin
        for(i=`kNumCoins-1; i>=0; i=i-1) begin
            if(coin_value[i] <= current_total - temp_return_total) begin
                o_return_coin[i] = 1;
                temp_return_total = temp_return_total + coin_value[i];
            end
        end
    end
end
```

3.1.2 Sequential Logic

기본적으로 clk에 맞추어 시간을 1씩 감소시킨다.
i_input_coin, i_select_item, reset_n이 있으면 시간을 초기화한다.
트리거가 되었을 때 시간을 3 흘려보내는 역할도 한다.

```
// update time
always @(posedge clk ) begin
    // return after 3 cycles when triggerd
    if(i_trigger_return==1) begin
        threeCyclesCounter<=threeCyclesCounter+1;
    end
    else begin
        threeCyclesCounter<=0;
    end

    // reset time
    if (!reset_n) begin
        wait_time <= `kWaitTime;
    end
    // decrease time
    else begin
        wait_time <= wait_time - 1;
    end



    // update coin return time when input exists
    for(i=0; i < `kNumCoins; i = i + 1) begin
            if(i_input_coin[i] == 1) begin
                wait_time <= `kWaitTime;
            end
    end
    for(i=0; i < `kNumItems; i = i + 1) begin
        if(i_select_item[i] == 1 && item_price[i] <= current_total) begin
            wait_time <= `kWaitTime;
        end
    end
end
```

3.2 calculate_current_state

3.2.1 Combinational Logic

3.2.1.1 Caculate next values: input값에 따라 값을 계산한다.

```
// Combinational logic for the next states
always @(*) begin
    // send calculated values to the output
    // those values will be integrated in change_state.v
    input_total = 0;
    output_total = 0;
    return_total = 0;
    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
    for(i=0; i < `kNumCoins; i = i + 1) begin
        if(o_return_coin[i] == 1) begin
            return_total = return_total + coin_value[i];
        end
    end
    for(i = 0; i < `kNumItems; i = i + 1) begin
        if(i_select_item[i] == 1 && item_price[i] <= current_total) begin
            output_total = output_total + item_price[i];
        end
    end

    current_total_nxt = current_total;
end
```

3.2.1.2 Calculate Outputs: 구매한 아이템, 구매가능 아이템 출력

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

    // TODO: o_output_item
    o_output_item = 0;
    for(i = 0; i < `kNumItems; i = i + 1) begin
        if(i_select_item[i] == 1 && item_price[i] <= current_total) begin
            o_output_item[i] = 1;
        end
    end
end
```

3.3 change_state

3.3.1 Sequential Logic: 리셋시 돈 초기화, 바뀐 잔고 업데이트

```
// 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.
        // calculate from output of calculate_current_state module
        current_total <= current_total_nxt + input_total - return_total - output_total;
    end
end
```

4 Discussion

4.1 wait_time 관련

check_time_and_coin에서 wait_time을 서로 다른 로직에서 접근하니까 에러가 발생하였다. 따라서 하나의 sequential logic 안에서 전부 처리해 주는 방식으로 구현하였다.

4.2 input_total, output_total, return_total 관련

이 세가지 값은 원래 calculate_current_state의 output으로만 사용되게 skeleton code에 짜여 있었다. 그러나 이 값들을 사용하지 않자 warning이 계속 뜨는 문제가 있었다.

따라서 change_state의 입력값으로 넣어준 뒤, change_state에서 계산하는 방식으로 구현하였다.

5 Conclusion

이번 lab을 통해서 sequential logic, combinational logic과 <=, =의 차이(non-blocking, blocking)를 이해할 수 있게 되었다. 또한 FSM을 직접 구현하므로써 주어진 문제를 어떻게 설계하는 지에 대해서도 알아볼 수 있었다.