1. Introduction

   본 과제는 verilog를 이용해 alu 및 vending machine을 직접 구현함으로써 Verilog 문법 및 하드웨어 설계에 익숙해지는 것을 목표로 삼는다.

2. Design

   Vending machine을 Moore machine 기반의 FSM으로 구현하였다. Top-level module을 포함해 총 6개의 모듈로 구성하였다.

   vending_machine은 top-level module로, 인터페이스 역할을 한다.

   balance_calculator는 매 클럭마다 사용 가능한 금액을 계산하는 Sequential logic이다.

   availability_calculator는 현재 금액으로 구매 가능한 물품을 반환하는 Combinational logic이다.

   item_dispenser는 물품 구매 시 구매된 물건을 반환하는 Sequential logic이다.

   coin_dispenser는 동전 반환을 처리하는 Sequential logic이다.

   timer는 사용자의 마지막 입력 이후 경과한 클럭 수를 계산하는 Sequential logic이다.

3. Implementation

   본 섹션에서는 각 모듈의 주요 동작 원리와 구현 방식을 설명한다. 주요 모듈은 총 6개이며, 각 모듈의 기능과 동작은 다음과 같다.

```
`include "vending_machine_def.v"
module vending_machine (
    input clk, // Clock signal
    input reset_n, // Reset signal (active-low)
    input [`kNumCoins-1:0] i_input_coin, // coin is inserted.
    input [`kNumItems-1:0] i_select_item, // item is selected.
    input i_trigger_return, // change-return is triggered
    output [`kNumItems-1:0] o_available_item, // Sign of the item availability
    output [`kNumItems-1:0] o_output_item, // Sign of the item withdrawal
    output [`kNumCoins-1:0] o_return_coin); // Sign of the coin return
    wire [`kTotalBits-1:0] balance;
    wire [`kTotalBits-1:0] cost;
    wire done;  // reset signal emitted by coin_dispenser, active-high
    wire timer_trigger;
    wire full_reset;
    wire ret;
    assign full_reset = reset_n & ~done;
    assign ret = timer_trigger | i_trigger_return;
    availability_calculator availability_calculator_module(
    .balance(balance),
    .o_available_item(o_available_item));
    balance_calculator balance_calculator_module(
    .clk(clk),
    .reset_n(full_reset),
    .i_input_coin(i_input_coin),
    .cost(cost),
    .balance(balance));
    timer timer_module(
    .clk(clk),
    .reset_n(full_reset),
    .trigger(timer_trigger)
    );
    coin_dispenser coin_dispenser_module(
    .clk(clk),
    .reset_n(full_reset),
    .ret(ret),
    .balance(balance),
    .o_return_coin(o_return_coin),
    .done(done));
    item_dispenser item_dispenser_module(
    .clk(clk),
    .reset_n(full_reset),
    .i_select_item(i_select_item),
    .o_available_item(o_available_item),
    .balance(balance),
    .o_output_item(o_output_item),
    .cost(cost));
endmodule
```

   vending_machine 모듈은 전체 시스템의 인터페이스 역할을 수행하며, 다른 서브 모듈들의 입력 및 출력을 연결한다. 사용자 입력을 처리하고 상태를 관리하며, 하위 모듈에 신호를 전달하고 반환 값을 받아 처리한다.

```
`include "vending_machine_def.v"

module balance_calculator (
    input clk,
    input reset_n,
    input [`kNumCoins-1:0] i_input_coin,
    input [`kTotalBits-1:0] cost,
    output reg [`kTotalBits-1:0] balance);

    wire [`kTotalBits-1:0] coin_value [`kNumCoins-1:0]; // Value of each coin
    assign coin_value[0] = 100;
    assign coin_value[1] = 500;
    assign coin_value[2] = 1000;

    always @(posedge clk) begin
        if (!reset_n)
            balance <= 0;
        else
            balance <= balance - cost + i_input_coin[0] * coin_value[0] +
i_input_coin[1] * coin_value[1] + i_input_coin[2] * coin_value[2];
    end

endmodule
```

balance_calculator 모듈은 매 클럭에서 입력된 금액과 사용된 금액을 기반으로 사용 가능한 잔액을 계산한다. 클럭 상승 엣지에서 입력 값을 처리하며, 비동기 reset 신호를 통해 잔액을 초기화할 수 있다.

```
`include "vending_machine_def.v"

module availability_calculator (
    input [`kTotalBits-1:0] balance,
    output reg [`kNumItems-1:0] o_available_item);
    wire [`kTotalBits-1:0] item_price [`kNumItems-1:0]; // Price of each item
    assign item_price[0] = 400;
    assign item_price[1] = 500;
    assign item_price[2] = 1000;
    assign item_price[3] = 2000;

    always @(*) begin
        o_available_item = 4'b0000;
        if (balance >= item_price[0])
            o_available_item = o_available_item | 4'b0001;
        if (balance >= item_price[1])
            o_available_item = o_available_item | 4'b0010;
        if (balance >= item_price[2])
            o_available_item = o_available_item | 4'b0100;
        if (balance >= item_price[3])
            o_available_item = o_available_item | 4'b1000;
    end

endmodule
```

availability_calculator 모듈은 현재 잔액을 기반으로 구매 가능한 품목을 계산하고 반환한다. 조합 논리(combinational logic)로 구현되어 있으며, 클럭 엣지와 상관없이 잔액 변화에 따라 실시간으로 출력 값이 갱신된다. 각 품목의 가격에 따라 구매 가능 여부를 결정하고, 비트마스크로 반환한다.

```
`include "vending_machine_def.v"

module item_dispenser (
    input clk,
    input reset_n,
    input [`kNumItems-1:0] i_select_item,
    input [`kNumItems-1:0] o_available_item,
    input [`kTotalBits-1:0] balance,
    output reg [`kNumItems-1:0] o_output_item,
    output reg [`kTotalBits-1:0] cost);

    wire [`kTotalBits-1:0] item_price [`kNumItems-1:0]; // Price of each item
    assign item_price[0] = 400;
    assign item_price[1] = 500;
    assign item_price[2] = 1000;
    assign item_price[3] = 2000;

    reg [`kNumItems-1:0] selected;
    reg [`kTotalBits-1:0] selected_cost;

    reg [`kTotalBits-1:0] prev_balance;
    reg [`kNumItems-1:0] prev_available_item;

    always @(posedge clk) begin
        if (!reset_n) begin
            prev_balance <= 0;
            prev_available_item <= 0;
        end
        else begin
            prev_balance <= balance;
            prev_available_item <= o_available_item;
        end
    end

    always @(*) begin
        // determine o_output_items
        selected = i_select_item & prev_available_item;
        selected_cost = selected[0] * item_price[0] + selected[1] *
item_price[1] + selected[2] * item_price[2] + selected[3] * item_price[3];
        if (prev_balance >= selected_cost) begin
            o_output_item = selected;
        end
        else begin
            o_output_item = 0;
        end
    end

    reg [`kNumItems-1:0] selected2;
    reg [`kTotalBits-1:0] selected_cost2;

    always @(*) begin
        // determine cost
        selected2 = i_select_item & o_available_item;
        selected_cost2 = selected2[0] * item_price[0] + selected2[1] *
item_price[1] + selected2[2] * item_price[2] + selected2[3] * item_price[3];
        if (balance >= selected_cost2) begin
            cost = selected_cost2;
        end
        else begin
            cost = 0;
        end
    end

endmodule
```

item_dispenser 모듈은 특정 품목 구매 시 해당 품목의 반환을 처리한다. 클럭 상승 엣지에서 작동하며, 품목 반환 후 잔액을 조정한다.

```
`include "vending_machine_def.v"
module coin_dispenser (
    input clk,
    input reset_n,
    input ret,
    input [`kTotalBits-1:0] balance,
    output reg [`kNumCoins-1:0] o_return_coin,
    output reg done);

    wire [`kTotalBits-1:0] coin_value [`kNumCoins-1:0]; // Value of each coin
    assign coin_value[0] = 100;
    assign coin_value[1] = 500;
    assign coin_value[2] = 1000;

    reg [`kTotalBits-1:0] change_due;
    reg [`kTotalBits-1:0] change_popped;
    reg [1:0] trigger;

    always @(posedge clk) begin
        if (!reset_n) begin
            change_due <= 0;
            change_popped <= 0;
            o_return_coin <= 0;
            trigger <= 0;
            done <= 0;
        end
        else begin
            case (trigger)
                0: begin
                    if (ret) begin
                        change_due <= balance;
                        change_popped <= 0;
                        trigger <= 1;
                    end
                end
                1: begin
                    trigger <= 2;
                end
                2: begin
                    if (change_due >= 1600) begin
                        o_return_coin <= 3'b111;
                        change_due <= change_due - 1600;
                    end
                    else if (change_due >= 1500) begin
                        o_return_coin <= 3'b110;
                        change_due <= change_due - 1500;
                    end
                    else if (change_due >= 1100) begin
                        o_return_coin <= 3'b101;
                        change_due <= change_due - 1100;
                    end
                    else if (change_due >= 1000) begin
                        o_return_coin <= 3'b100;
                        change_due <= change_due - 1000;
                    end
                    else if (change_due >= 600) begin
                        o_return_coin <= 3'b011;
                        change_due <= change_due - 600;
                    end
                    else if (change_due >= 500) begin
                        o_return_coin <= 3'b010;
                        change_due <= change_due - 500;
                    end
                    else if (change_due >= 100) begin
                        o_return_coin <= 3'b001;
                        change_due <= change_due - 100;
                    end
                    else begin
                        o_return_coin <= 3'b000;
                        trigger <= 3;
                    end
                end
                3: begin
                    done <= 1;
                end
            endcase
        end
    end
endmodule
```

coin_dispenser 모듈은 동전 반환 신호가 활성화되면 보유 잔액에서 반환할 금액을 계산하고 반환한다.

```
`include "vending_machine_def.v"

module timer (
    input clk,
    input reset_n,
    output reg trigger);

    reg signed [`kTotalBits-1:0] counter;

    always @(posedge clk) begin
        if (!reset_n) begin
            counter <= `kWaitTime - 1;
            trigger <= 0;
        end
        else begin
            if (counter > 0)
                counter <= counter - 1; // counter goes down to -1
            else if (counter == 0) begin
                trigger <= 1;
                counter <= counter - 1;
            end
            else
                trigger <= 0;
        end
    end

endmodule
```

timer 모듈은 사용자 입력 이후 경과한 클럭 수를 계산한다. 클럭 상승 엣지에서 상태를 갱신하며, 특정 시간이 초과될 경우 자동으로 상태를 초기화한다. 상태 전환 및 동작 타이밍을 제어하기 위한 기준이 된다.

4. Discussion

본 과제에서 Moore Machine 기반의 FSM을 설계하면서 상태 전환 및 출력 동기화 과정에서 몇 가지 문제를 경험하였다. 특히, 동전 반환 및 잔액 계산 과정에서 발생한 비정상적인 동작은 상태 전환 조건과 클럭 동기화를 수정해 해결할 수 있었다. 또한, 각 모듈 간의 신호 전달에서 발생한 지연 문제는 비차단 할당(<=)을 사용하여 개선하였다. 이를 통해 하드웨어 동작에 대한 이해를 깊이 할 수 있었다.

5. Conclusion

본 과제에서는 Verilog를 사용하여 ALU 및 Vending Machine을 설계하고 구현하였다. Moore Machine 기반의 FSM을 사용해 총 6개의 모듈로 구성하였으며, 각 모듈이 정상적으로 상호작용하도록 설계하였다. 구현 과정에서 발생한 상태 전환 오류를 수정해 시스템의 안정성을 확보하였다. 본 과제를 통해 Verilog 문법 및 하드웨어 설계 기법에 대한 이해도를 높일 수 있었다.