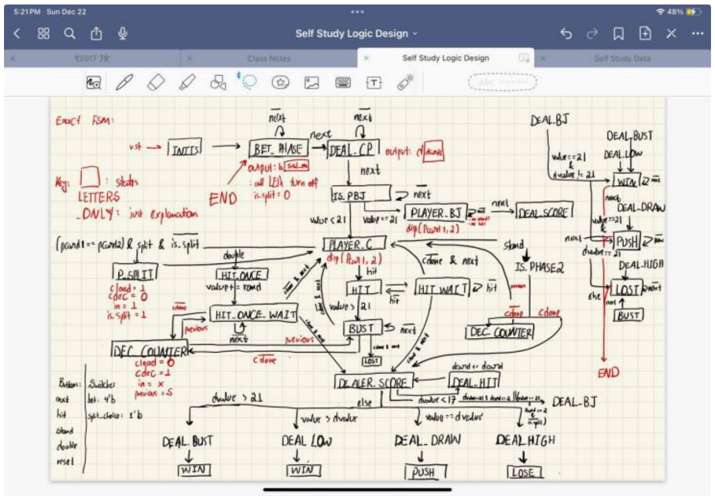
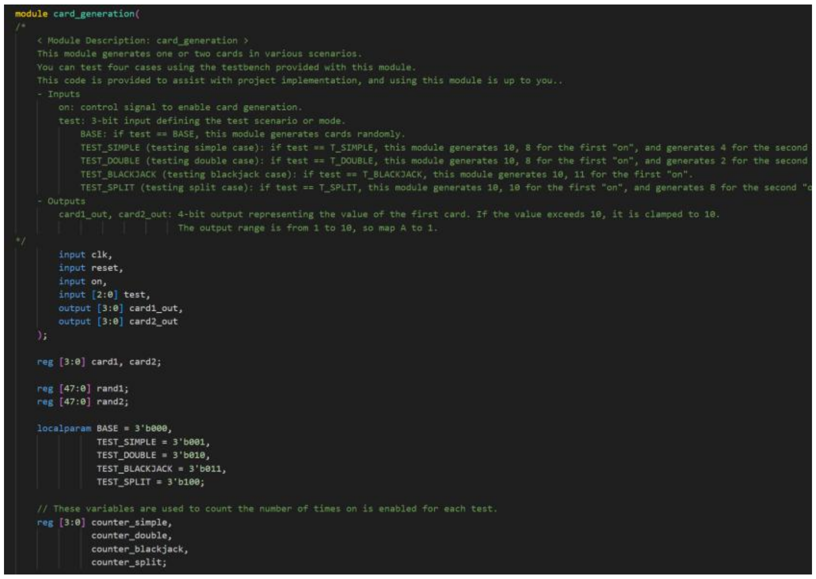
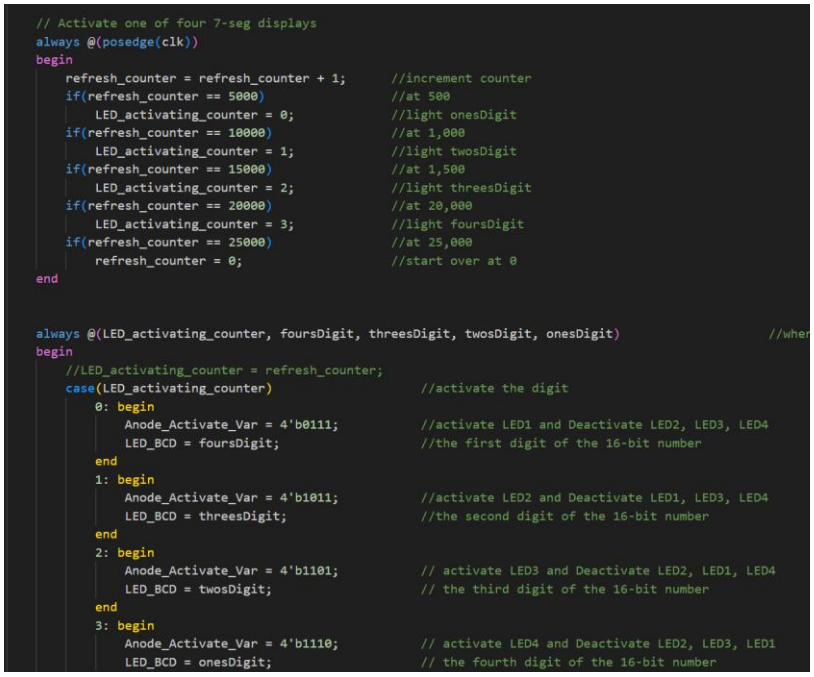
Implementation of a Black Jack game on FPGA board	
Where and When:	Seoul National University, During my 2nd year 2nd Semester
Start-end date:	Project starting date: 2024.12 Project ending date: 2024.12
Description of the project:	This group project was to implement 'black jak' game on FPGA (Field Programmable Gate Array) board using Verilog HDL (Hardware Description Language).
The role you played:	Understanding the game rules, designing state diagrams for the game and implementation of the Verilog code onto the FPGA board.  Self Study Logic Design Exact FSM: rst INITS BET_PHASE DEAL_CP next Key: : states LETTERS_ONLY: just explanation END IS_PBJ PLAYER_BJ DEAL_SCORE PLAYER_C P_SPLIT HIT_ONCE HIT_ONCE_WAIT HIT HIT_WAIT BUST LOST IS_PHASE2 DEC_COUNTER DEALER_SCORE DEAL_HIT DEAL_BUST DEAL_LOW DEAL_DRAW DEAL_HIGH DEAL_BJ WIN PUSH LOSE Buttons: next, hit, stand, double, reset Switches: bet: 4'b, split_choice: 1'b *Figure 1. My FSM design*
Precise description of what you did:	This group project required two tasks to be completed. The first one was **writing a Verilog code** while the second one was **implementing** this code on FPGA. Mainly, I contributed by drafting a state diagram of the game execution upon which the next step of code writing took place. Moreover, I synthesized and executed the Verilog code on to the FPGA to carry out the game.
Lessons learnt:	In this group project, we managed to demonstrate the basic game routines. Working on this project with my teammates, I felt that our team lacked strong communication delaying the execution

phase of the project to a day before the final demonstration of the project. However, the final moment communication resolved the issue regarding role divisions among my teammates and I, which helped us to finish the project successfully.



```
module card_generation(
/*
    < Module Description: card_generation >
    This module generates one or two cards in various scenarios.
    You can test four cases using the testbench provided with this module.
    This code is provided to assist with project implementation, and using this module is up to you..
    - Inputs
        on: control signal to enable card generation.
        test: 3-bit input defining the test scenario or mode.
            BASE: if test == BASE, this module generates cards randomly.
            TEST_SIMPLE (testing simple case): if test == T_SIMPLE, this module generates 10, 8 for the first "on", and generates 4 for the second
            TEST_DOUBLE (testing double case): if test == T_DOUBLE, this module generates 10, 8 for the first "on", and generates 2 for the second
            TEST_BLACKJACK (testing blackjack case): if test == T_BLACKJACK, this module generates 10, 11 for the first "on".
            TEST_SPLIT (testing split case): if test == T_SPLIT, this module generates 10, 10 for the first "on", and generates 8 for the second "
    - Outputs
        card1_out, card2_out: 4-bit output representing the value of the first card. If the value exceeds 10, it is clamped to 10.
                                  The output range is from 1 to 10, so map A to 1.
*/
        input clk,
        input reset,
        input on,
        input [2:0] test,
        output [3:0] card1_out,
        output [3:0] card2_out
    );

    reg [3:0] card1, card2;

    reg [47:0] rand1;
    reg [47:0] rand2;

    localparam BASE = 3'b000,
               TEST_SIMPLE = 3'b001,
               TEST_DOUBLE = 3'b010,
               TEST_BLACKJACK = 3'b011,
               TEST_SPLIT = 3'b100;

    // These variables are used to count the number of times on is enabled for each test.
    reg [3:0] counter_simple,
              counter_double,
              counter_blackjack,
              counter_split;
```

Figure 2. Code snippets from our project code



```
// Activate one of four 7-seg displays
always @(posedge(clk))
begin
    refresh_counter = refresh_counter + 1;      //increment counter
    if(refresh_counter == 5000)                 //at 500
        LED_activating_counter = 0;             //light onesDigit
    if(refresh_counter == 10000)                //at 1,000
        LED_activating_counter = 1;             //light twosDigit
    if(refresh_counter == 15000)                //at 1,500
        LED_activating_counter = 2;             //light threesDigit
    if(refresh_counter == 20000)                //at 20,000
        LED_activating_counter = 3;             //light foursDigit
    if(refresh_counter == 25000)                //at 25,000
        refresh_counter = 0;                    //start over at 0
end


always @(LED_activating_counter, foursDigit, threesDigit, twosDigit, onesDigit)                    //wher
begin
    //LED_activating_counter = refresh_counter;
    case(LED_activating_counter)                    //activate the digit
        0: begin
            Anode_Activate_Var = 4'b0111;           //activate LED1 and Deactivate LED2, LED3, LED4
            LED_BCD = foursDigit;                   //the first digit of the 16-bit number
        end
        1: begin
            Anode_Activate_Var = 4'b1011;           //activate LED2 and Deactivate LED1, LED3, LED4
            LED_BCD = threesDigit;                  //the second digit of the 16-bit number
        end
        2: begin
            Anode_Activate_Var = 4'b1101;           // activate LED3 and Deactivate LED2, LED1, LED4
            LED_BCD = twosDigit;                    // the third digit of the 16-bit number
        end
        3: begin
            Anode_Activate_Var = 4'b1110;           // activate LED4 and Deactivate LED2, LED3, LED1
            LED_BCD = onesDigit;                    // the fourth digit of the 16-bit number
```

Figure 3. Code snippets from our project code