

Project title: Number guessing

Team leader and members: Ellis Chen, Fong Jheng Lin

Date of submission: Dec 13, 2022

### Project Overview and Objectives:

The goal of the project is to implement a number guessing game using SystemVerilog. The system includes several inputs: a restart toggle switch, confirm button and toggle-digits pushbuttons. A restart toggle switch would reset the game anytime when triggered. Three toggle-digits push buttons would let users determine the guessing numbers, and a confirm button would confirm the input value and then do the comparison. The game has three difficulties, and each difficulty has three rounds. Each difficulty has different guessing digits: difficulty one for one digit, two for two digits and three for three digits. Challengers would use push buttons to input numbers and then push a confirm button to guess correct answers under finite attempts and time limit. If challengers guess the correct answer, they would move to the next round. After three rounds of correct guesses, challengers would move to the next difficulty. If the guessing number is wrong, a hint would show "H" or "L" to suggest users to guess higher or lower numbers. If users run out of attempts or time, the game will end with a text showing "you lose" on seven-segments. If challengers pass difficulty 3, they will win the game with a text showing "good job" through a seven-segment display.

## Solution:

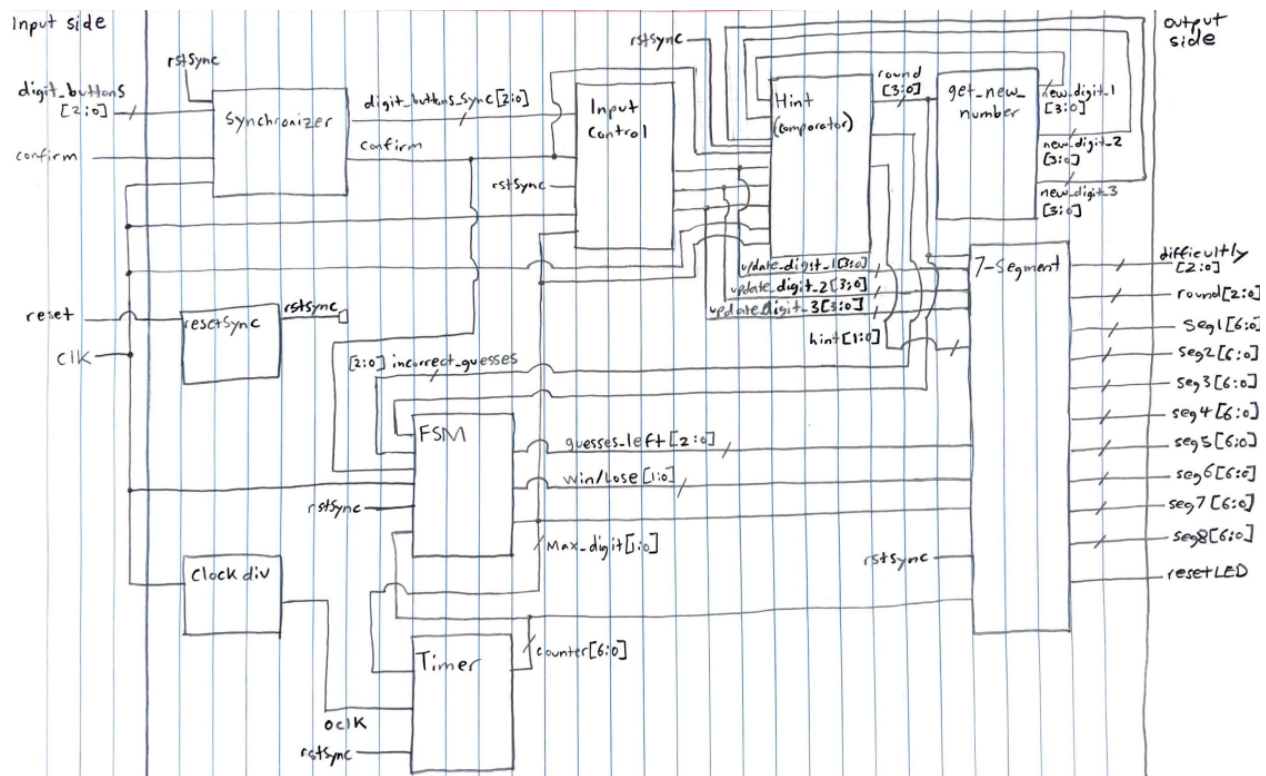


Figure 1. System Diagram

## Design Description:

According to figure 1, the system takes in 5 inputs: 4 pushbuttons and a reset switch, and 15 outputs: 8 seven-segments and 7 LEDs. The system is divided into the following main components: input controls, a comparator for the hint feature, a timer, a way to retrieve new numbers to update the correct answer, seven-segment logic, a clock divider, and several synchronizers.

Figure 2 shows the player interface on the development board. The timer displays how much time is left in the game session, and "Lives" means how many available guesses are left. The hint segment would show "H", "L" or nothing, which indicates the player should guess a higher number, lower number, or their guess was correct respectively. Digits would show what number the player guesses. LEDs represent rounds and difficulty, one light represents one, two represents two, and so on. The restart switch is active low, and it would restart the game if being triggered. Player would use toggle digits to enter the guessing number. Each time the button is pressed, the number will increase by one. Pressing the button when the number is nine will return the value to zero. Once the player determines the guessing number, pushing the confirm button would execute the comparison with the correct answer.

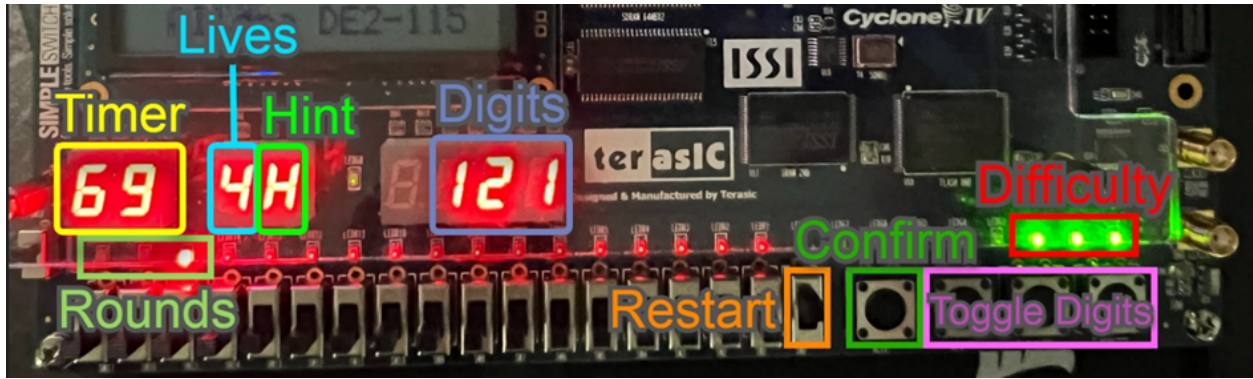


Figure 2. Interface of number guessing game

To create a system that can keep track of the state of the game session and have game parameters updated accordingly, a state diagram was necessary in the implementation. According to figure 3, the state machine has five states, and the game always starts at difficulty 1. The game only moves to the next difficulty state if a certain number of rounds have been successfully completed, eventually reaching the win state. The game will always go to the game-over state when either the timer reaches 0 or too many incorrect attempts have been made by the player. Upon a reset, the game always goes back to the first difficulty state. Figure 4 shows the code snippet of a finite state machine.

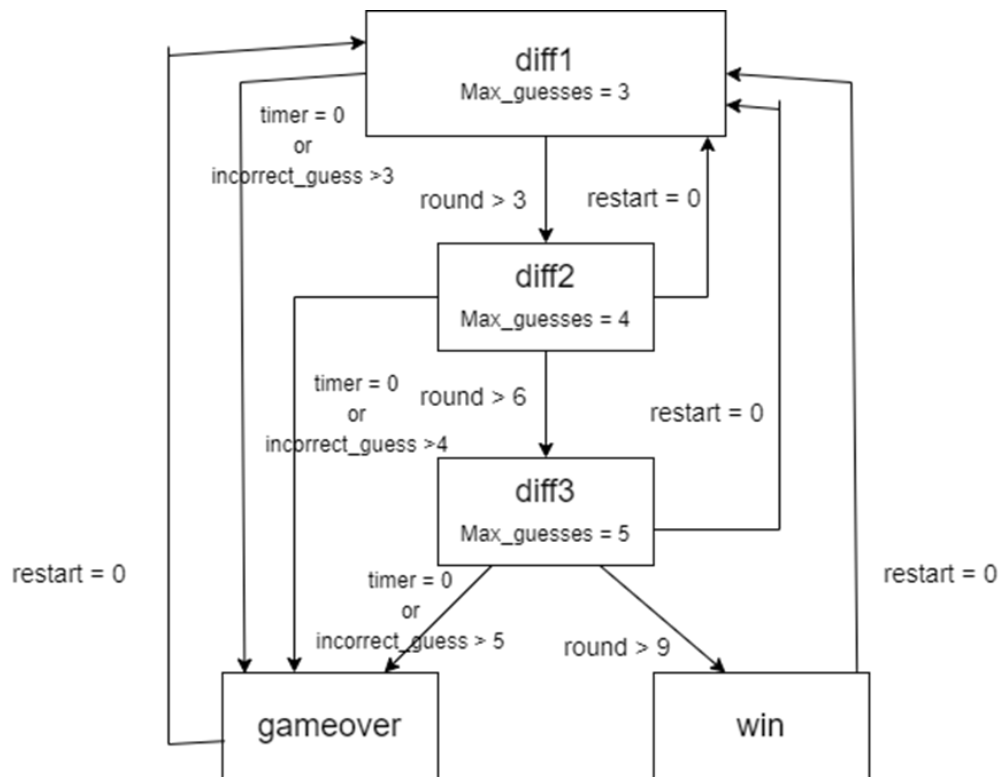


Figure 3. state diagram of finite state machine

```

always_ff @(posedge clk)
begin
    if (!restart) begin
        reg_max_digit <= 2'b01; // reset register for max digit value
    end
    case(presentState)
        diff1: begin
            if (round > 3 ) begin // guess correctly 3x for 1st difficulty
                reg_max_digit <= 2'b10; // increase max digit value to 2
                nextState <= diff2; // move to 2nd difficulty
            end else if ( timer == 0 || incorrect_guesses > Max_incorrect_guesses ) begin
                nextState <= gameover; // time runs out or too many bad guesses
            end else
                nextState <= diff1; // no player activity
        end
    end
end

```

Figure 4. Code snippet of FSM

Module “input control” deals with the behavior of toggling the digits that are displayed for the player, and figure 5 reveals its mechanism. The number of digits a player can guess is dependent on the difficulty of the game, which is decided by the FSM. Players are presented only with digits that are unlocked at the current difficulty level.

Module “timer” determines the max remaining time for each difficulty, figure 6 demonstrates the code snippet. A clock divider would pass the real-time second to the module “timer” and it follows the timing equation of DE2-115 board:  $50\text{MHz} / 2^x = 2\text{ Hz}$ . The clock divider value needed to achieve a 1 Hz clock for the timer was 25,000,000. This was the only module to use the divided clock; all other components run on the 50MHz clock. Figure 7 shows the code setting of the clock divider.

We use a synchronizer to make sure that the inputs from push buttons would be held longer than one clock cycle.

Module “hint123” describes the mechanism of the hint system, and it would output the hint status according to the comparison of the guessing number and correct answer, and then pass the indicator to the seven-segment module to display the hint. Figure 8 demonstrates the code of the hint123 module.

Module “get\_target\_module” is where we set the correct answer for each difficulty, and figure 9 shows part of the code statements.

The seven-segment module takes all the indicators from other modules, and is responsible for showing the correct displays on board. Figure 10 demonstrates some of the software descriptions.

```

always_ff@(posedge clk)
begin
    if (!restart) begin // reset register values to zero
        temp_digit_1 <= 0;
        temp_digit_2 <= 0;
        temp_digit_3 <= 0;
    end else begin
        if (pushbuttons[0] == 1'b1 && max_digits > 2'd0) begin // difficulty 1 & pushbutton key0
            if (temp_digit_1 < 4'd9)
                temp_digit_1 <= temp_digit_1 + 1'd1; // increment digit by 1 up to '9'
            else
                temp_digit_1 <= 4'd0; // loop back to '0'
        end
    end
end

```

Figure 5. Code snippet of input control

```

always_ff @(posedge clk)
begin
    if (!restart) begin // reset timer values
        diff1 <= 7'd30;
        diff2 <= 7'd60;
        diff3 <= 7'd90;
    end else begin
        if (Max_digit == 1)
            diff1 <= diff1 - 1'b1; // difficulty 1
        if (Max_digit == 2)
            diff2 <= diff2 - 1'b1; // difficulty 2
        if (Max_digit == 3)
            diff3 <= diff3 - 1'b1; // difficulty 3
    end
end
end

```

Figure 6. Code snippet of timer

```

module clockdiv(input logic iclk,
                output logic oclk);

// define the parameter for number of clock edges to count
const int HALF_OF_CLK_CYCLE_VALUE = 32'd25_000_000; // change this value

```

Figure 7. Timer setup of clock divider

```

always_ff@(posedge clk)
begin
    if (!restart) begin
        hint1 <= 2'b11;
        reg_round <= 1;
        reg_incorrect_guess_d1 <= 0;
        reg_incorrect_guess_d2 <= 0;
        reg_incorrect_guess_d3 <= 0;
    end else begin
        if(confirmButton)begin // check player input only when they confirm their guess
            case(Max_digit)
            1: begin // case for single digit comparison
                if(key0 == answer0) begin
                    reg_round++; // move to next round upon correct guess
                    hint1 <= 2'b11; // do not provide player with any hint
                end else begin
                    reg_incorrect_guess_d1++; // record incorrect guesses
                end

                if(key0 > answer0) // checks if player guess is greater than answer
                    hint1 <= 0; // prompts player to guess lower on next attempt
                if(key0 < answer0) // checks if player guess is less than answer
                    hint1 <= 1; // prompts player to guess higher on next attempt
            end
        end
    end
end

```

Figure 8. Behavioral description of hint system

```

always_ff@(posedge clk)
begin
    if (Max_digit == 2'b01) begin // single digit
        case(round)
            4'b0001: begin // 1st number is '2'
                target_digit_3 <= 4'd0;
                target_digit_2 <= 4'd0;
                target_digit_1 <= 4'd2;
            end
            4'b0010: begin // 2nd number is '8'
                target_digit_3 <= 4'd0;
                target_digit_2 <= 4'd0;
                target_digit_1 <= 4'd8;
            end
            4'b0011: begin // 3rd number is '3'
                target_digit_3 <= 4'd0;
                target_digit_2 <= 4'd0;
                target_digit_1 <= 4'd3;
            end
        endcase
    end
    if (Max_digit == 2'b10) begin // double digits
        case(round)

```

Figure 9. Code of “get\_target\_number” module

```

case(guesses) // reserve one 7-segment for number of guesses
0:
    seg3 <= 7'b1000000;
1:
    seg3 <= 7'b1111001;
2:
    seg3 <= 7'b0100100;
3:
    seg3 <= 7'b0110000;
4:
    seg3 <= 7'b0011001;
5:
    seg3 <= 7'b0010010;
default:
    seg3 <= 7'b1111111; // reset
endcase

case(hint1) // reserve one 7-segment for hint
0: // prompt player to guess lower
    seg4 <= 7'b1000111;
1: // prompt player to guess higher
    seg4 <= 7'b0001001;
default: // player guessed correctly; no hint to show
    seg4 <= 7'b1111111;
endcase

```

Figure 10. code demonstration of seven-segment module

## Testing Procedure:

The most simple component to test first was the timer. The goal of the timer testbench was to verify that the timer does count down by one for every positive edge of the clock. Also, the testbench serves to verify that the maximum timer value updates according to the difficulty state of the game; the maximum timer value should update between 30, 60, and 90 for difficulties 1, 2, and 3 respectively. The test cases were not randomized; the goal was to see if the timer counted down all the way to 0 in increments of one. The testbench also uses resets, in which the current timer is expected to reset back to the appropriate maximum value.

The next testbench was for the input control component. The test cases were divided into three parts: testing one digit for the first part, secondly testing two digits, and lastly testing all three digits. For each part, the testbench simulates pressing the push buttons to toggle the digit values. Every push of the button should see the corresponding digit update by 1. Asserts are used to catch instances where the digit value did not wrap around from '9' to '0' or when the corresponding digit did not increment at all. The amount of times the push buttons for all 3 digits were randomized because no matter how many times you press the button to toggle the digits, the digit values should always stay within the range of 0 to 9.

The next component to test was the hint (comparator). The hint module is responsible for comparing the player-selected digits to the answer digits, so the testbench contains test cases that perform checks on comparisons. Test cases always start with a randomized set of digits that are assigned as the answer. The player digits are randomized as well for most of the testbench to add variability to the comparisons. The testbench used asserts to detect errors in the hint system every time the player locks in their guess. The asserts check if the hint system indicates higher when the guess was too low, lower when the guess was too high, or nothing when the guess was correct. The testbench also displays the incrementation of the round upon successful match of digits or incrementation of incorrect guesses upon non-matching digits.

The testbench for the FSM was divided into 5 main tests. Test 1 aimed to verify that the FSM can switch to the game-over state upon the timer running down to 0 in the game. Asserts were used to check if the value of the timer resulted in a game-over state. This test case was not randomized because there was no reason to randomize a timer value. Test 2 aimed to determine if the reset worked, such that the state returned to the first difficulty along with all the game logic associated with that state. This test occurred immediately after the first test to verify that the FSM will go back to the difficulty 1 state from game-over state. Asserts were also used to check if the win/lose condition value had been reset. Test 3 aimed to verify that all game logic values scaled along with the progression of the difficulty levels. When moving up difficulty levels, the number of available guesses and maximum digits both increase. Asserts were used to check these logic elements were updated upon moving to the next difficulty level. No randomized test cases were used for test 3 because the focus was on the transition from one difficulty to the other. Test 4 aimed to verify that FSM would switch to the game-over state upon the player making too many incorrect guesses. No randomized test cases were used, the number of incorrect guesses were incremented by one for each clock cycle. The asserts checked for the win/lose condition to represent the losing value upon the player running out of

guesses. Test 5 was used to verify that the FSM can reach the winning state. The winning state is achieved by setting the round to 10, bypassing all the intermediate levels, in which the assert would check if the win/lose condition represented the winning value.

The final testbench for the full system did not use randomization because winning the game required entering the correct sequence of digits, and losing the game only required entering digits that were not equal to the predetermined number. The first test case was to check if the game can be finished with a win, so all the correct sequence of digits were entered and asserts were used to check that all game logic values update properly. The second test case checked for a game over on the first difficulty level. The third test case checked for a game over on the second difficulty level. The fourth test case checked for a game over on the third difficulty level. All test cases used asserts to check the messages printed on the seven-segment.



## Results:

The timer component under test works as intended. The maximum timer values were correctly set according to the difficulty level. The counter for the timer decrements by one every positive clock edge, as shown in figure 11. Despite the timer running past 0 into the negatives, this would not be an issue in the greater scope of the game. The reset also works, as the counter reset back to its initial value and count down proceeds as expected.

```
# .main_pane.structure.interior.cs.body.struct
# .main_pane.objects.interior.cs.body.tree
# Timer Value difficulty 1 = 30
# Timer Value difficulty 1 = 29
# Timer Value difficulty 1 = 28
# Timer Value difficulty 1 = 27
# Timer Value difficulty 1 = 26
# Timer Value difficulty 1 = 25
# Timer Value difficulty 1 = 24
# Timer Value difficulty 1 = 23
# Timer Value difficulty 1 = 22
# Timer Value difficulty 1 = 21
# Timer Value difficulty 1 = 20
# Timer Value difficulty 1 = 19
# Timer Value difficulty 1 = 18
# Timer Value difficulty 1 = 17
# Timer Value difficulty 1 = 16
# Timer Value difficulty 1 = 15
# Timer Value difficulty 1 = 14
# Timer Value difficulty 1 = 13
# Timer Value difficulty 1 = 12
# Timer Value difficulty 1 = 11
# Timer Value difficulty 1 = 10
# Timer Value difficulty 1 = 9
# Timer Value difficulty 1 = 8
# Timer Value difficulty 1 = 7
# Timer Value difficulty 1 = 6
# Timer Value difficulty 1 = 5
# Timer Value difficulty 1 = 4
# Timer Value difficulty 1 = 3
# Timer Value difficulty 1 = 2
# Timer Value difficulty 1 = 1
# Timer Value difficulty 1 = 0
# Timer Value difficulty 1 = 127
# Timer Value difficulty 1 = 126
# Reset Occured
# Timer Value difficulty 1 = 30
# Timer Value difficulty 1 = 29
#
# Timer Value difficulty 2 = 60
# Timer Value difficulty 2 = 59
# Timer Value difficulty 2 = 58
# Timer Value difficulty 2 = 57
# Timer Value difficulty 2 = 56
# Timer Value difficulty 2 = 55
# Timer Value difficulty 2 = 54
# Timer Value difficulty 2 = 53
# Timer Value difficulty 2 = 52
# Timer Value difficulty 2 = 51
# Timer Value difficulty 2 = 50
# Timer Value difficulty 2 = 49
# Timer Value difficulty 2 = 48
# Timer Value difficulty 2 = 47
# Timer Value difficulty 2 = 46
# Timer Value difficulty 2 = 45
# Timer Value difficulty 2 = 44
# Timer Value difficulty 2 = 43
# Timer Value difficulty 2 = 42
# Timer Value difficulty 2 = 41
# Timer Value difficulty 2 = 40
# Timer Value difficulty 2 = 39
# Timer Value difficulty 2 = 38
# Timer Value difficulty 2 = 37
# Timer Value difficulty 2 = 36
# Timer Value difficulty 2 = 35
# Timer Value difficulty 2 = 34
# Timer Value difficulty 2 = 33
# Timer Value difficulty 2 = 32
# Timer Value difficulty 2 = 31
# Timer Value difficulty 2 = 30
# Timer Value difficulty 2 = 29
```

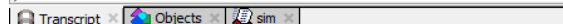
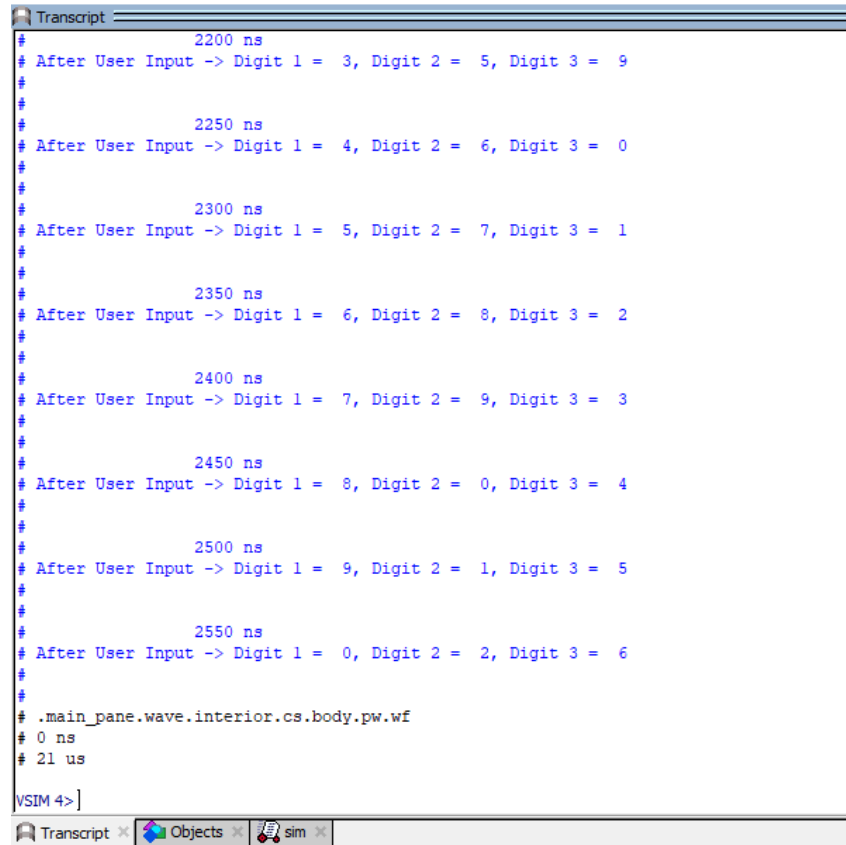


Figure 11. Timer Transcript

The test cases for the input control module proved that the digits were updating by a value of 1 every time the push buttons were pressed. Also, it is apparent that the digit value wraps around back to 0 when its prior value was 9. This behavior is shown in figure 12 and it proves that the input control works as expected.



```
Transcript
#
# 2200 ns
# After User Input -> Digit 1 = 3, Digit 2 = 5, Digit 3 = 9
#
#
# 2250 ns
# After User Input -> Digit 1 = 4, Digit 2 = 6, Digit 3 = 0
#
#
# 2300 ns
# After User Input -> Digit 1 = 5, Digit 2 = 7, Digit 3 = 1
#
#
# 2350 ns
# After User Input -> Digit 1 = 6, Digit 2 = 8, Digit 3 = 2
#
#
# 2400 ns
# After User Input -> Digit 1 = 7, Digit 2 = 9, Digit 3 = 3
#
#
# 2450 ns
# After User Input -> Digit 1 = 8, Digit 2 = 0, Digit 3 = 4
#
#
# 2500 ns
# After User Input -> Digit 1 = 9, Digit 2 = 1, Digit 3 = 5
#
#
# 2550 ns
# After User Input -> Digit 1 = 0, Digit 2 = 2, Digit 3 = 6
#
#
# .main_pane.wave.interior.cs.body.pw.wf
# 0 ns
# 21 us
VSIM 4>]
```

The screenshot shows a VSIM 4 transcript window with a title bar and three tabs: 'Transcript', 'Objects', and 'sim'. The transcript text shows a sequence of user inputs at 50 ns intervals from 2200 ns to 2550 ns. Each input results in the three digits (Digit 1, Digit 2, Digit 3) increasing by 1. At 2450 ns, Digit 2 wraps from 9 to 0. At 2550 ns, Digit 1 wraps from 9 to 0. The transcript ends with a prompt 'VSIM 4>]'.

Figure 12. Input Control Transcript

Figure 13 shows the results of the hint system upon the system comparing the player guess with the expected answer. The transcript from figure 13 shows that the asserts were able to detect when the hint provided by the game was correct or incorrect. Also, the transcript also shows that the number of incorrect guess increments every time the player guess did not match the expected answer (either hint was '0' or a '1'). Every time the player's guess and the expected answer matched, the hint would be '3' to signify that the guess was correct and no hint was needed; the round would also increment to reflect the correct guess.

```
# .main_pane.structure.interior.cs.body.struct
# .main_pane.objects.interior.cs.body.tree
#       76 ns, (Difficulty 3) Target number:  7  8  4
#
# Player input:  0  0  9
# Hint = 0, round =  1, incorrect_guess = 1
# Hint is '0' (lower): looks good
#
# Player input:  8  6  0
# Hint = 1, round =  1, incorrect_guess = 2
# Warning: Hint should be '0' (lower)!
#
# Player input:  7  6  4
# Hint = 1, round =  1, incorrect_guess = 3
# Hint is '1' (higher): looks good
#
# Player input:  7  8  4
# Hint = 3, round =  2, incorrect_guess = 3
# Hint is '3' (no hint): looks good
#
#       526 ns, (Difficulty 3) Target number:  3  0  7
#
# Player input:  5  6  8
# Hint = 0, round =  2, incorrect_guess = 4
# Hint is '0' (lower): looks good
#
# Player input:  0  7  4
# Hint = 1, round =  2, incorrect_guess = 5
# Warning: Hint should be '0' (lower)!
#
# Player input:  3  2  7
# Hint = 0, round =  2, incorrect_guess = 6
# Hint is '0' (lower): looks good
#
# Player input:  3  0  7
# Hint = 3, round =  3, incorrect_guess = 6
# Hint is '3' (no hint): looks good
#
#       976 ns, (Difficulty 3) Target number:  8  7  4
#
# Player input:  3  2  8
# Hint = 0, round =  3, incorrect_guess = 7
# Hint is '0' (lower): looks good
#
# Player input:  7  7  2
# Hint = 1, round =  3, incorrect_guess = 0
# Hint is '1' (higher): looks good
#
# Player input:  8  3  4
# Hint = 1, round =  3, incorrect_guess = 1
# Hint is '1' (higher): looks good
#
# Player input:  8  7  4
# Hint = 3, round =  4, incorrect_guess = 1
# Hint is '3' (no hint): looks good
#
# .main_pane.wave.interior.cs.body.pw.wf
# 0 ns
# 26250 ns
VSIM 5>
```



Figure 13. Hint (Comparator) Transcript

The results of test 1 of the FSM component are shown in figure 14. The goal was to verify a game-over state when the timer ran out. The WINorLOSE logic value should be '0' for a gameover, else the game is still in progress if the logic value is a '3'. So, it is expected that WINorLOSE would be set to '0' when the timer runs down to 0. Figure 14 shows that WINorLOSE was set to '0' in the same clock cycle as the timer was set to '-1'; this interesting find was perhaps due to the setup of the testbench, but this test was successful because the FSM did switch to the game over state.

```
# Test 1: Try to achieve Game over state by running out of time
#
# Current Timer = 30
# Current Timer = 29
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 28
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 27
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 26
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 25
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 24
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 23
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 22
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 21
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 20
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 19
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 18
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 17
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 16
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 15
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 14
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 13
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 12
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 11
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 10
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 9
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 8
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 7
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 6
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 5
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 4
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 3
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 2
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 1
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 0
# Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Current Timer = 127
# Max Digit = 1, WINorLOSE = 0, guesses_left = 0
# Game over due to 'time is up' correct
```

Figure 14. FSM Transcript Test 1 (Game over by time-out)

The test case for test 2 was verifying that setting the reset active low would cause a reset in the FSM, such that the game over state from test 1 would be set back to the difficulty 1 state. Figure 15 shows that the reset worked because the WINorLOSE logic value was reset from '0' (indicating game over) to '3' (game in progress). Also, Max Digit and guesses\_left were reset back to difficulty 1 game values.

```
#
# Test 2: Try to reset the win/lose flag to default by resetting game to difficulty 1
#
# Difficulty 1 logic: Max Digit = 1, WINorLOSE = 3, guesses_left = 3
# Reset worked correctly; win/lose condition reset
#
#
```

Figure 15. FSM Transcript Test 2 (Reset game)

The results for test 3 shows that the in-game logic values, such as max available guesses and max digits, did scale according to the increase in difficulty. Figure 16 shows that the jump from difficulty 2 to difficulty 3 updated the values for Max digit and guesses\_left. The same case can be applied to difficulty 1 to difficulty 2. The test cases will also hold true in the reverse, difficulty 3 to difficulty 2, but these findings are not significant to the intended progression of the game.

```
#
# Test 3: Try to verify max num digits, max num allowed incorrect guesses, and max timer value scale by difficulty
#
# Try to see updates to difficulty 2
#
# Difficulty 2 logic: Max Digit = 2, WINorLOSE = 3, guesses_left = 4
# Max digits updated properly
# win/lose value good
# Num available guesses updated properly
# Try to see updates to difficulty 3
#
# Difficulty 3 logic: Max Digit = 3, WINorLOSE = 3, guesses_left = 5
# Max digits updated properly
# win/lose value good
# Num available guesses updated properly
#
```

Figure 16. FSM Transcript Test 3 (Verify game parameters update when moving to next difficulty)

Figure 17 shows results of test 4, in which the FSM switched to a game over state after determining that too many incorrect guesses were used up. The transcript from figure 17 shows that the test was performed for difficulty 3, and the guesses\_left value decremented upon every incorrect guess. Upon using up all guesses, the WINorLOSE logic value was set to '0' (signifying a game over). This test showed that it is possible to lose the game on too many incorrect attempts. The test can be replicated for other difficulties, and the test still holds as valid.

```
"
# Test 4: Try to achieve game over state by running out of guesses
#
# Try to use up all 5 incorrect attempts in difficulty 3
#
# Difficulty 3 logic: Max Digit = 3, WINorLOSE = 3, guesses_left = 5
# Difficulty 3 logic: Max Digit = 3, WINorLOSE = 3, guesses_left = 4
# Difficulty 3 logic: Max Digit = 3, WINorLOSE = 3, guesses_left = 3
# Difficulty 3 logic: Max Digit = 3, WINorLOSE = 3, guesses_left = 2
# Difficulty 3 logic: Max Digit = 3, WINorLOSE = 3, guesses_left = 1
# Difficulty 3 logic: Max Digit = 3, WINorLOSE = 3, guesses_left = 0
# Difficulty 3 logic: Max Digit = 3, WINorLOSE = 0, guesses_left = 2
"
```

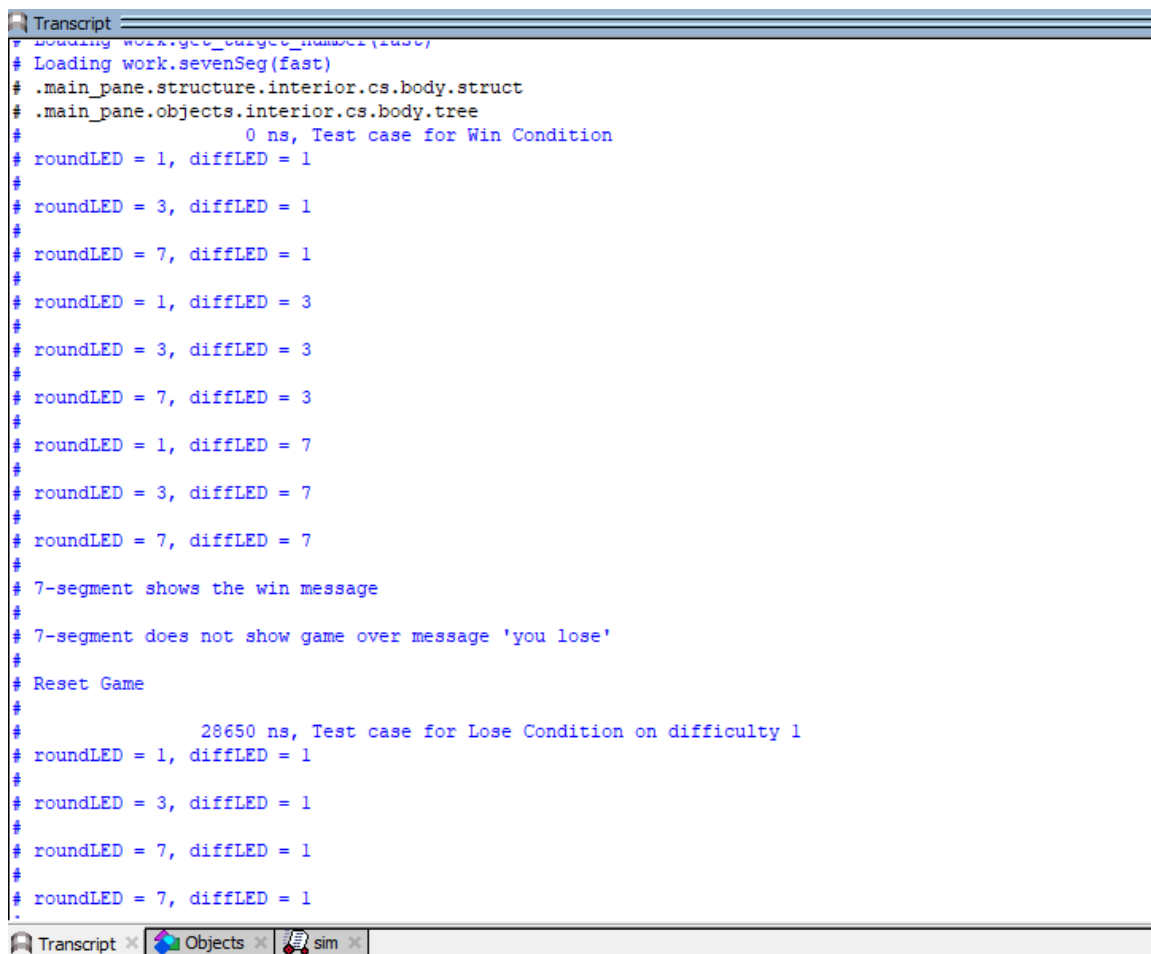
Figure 17. FSM Transcript Test 4 (Game over by too many incorrect guesses)

Figure 18 shows the results of test 5, in which the goal was to achieve a winning state. The transcript showed that the WINorLOSE logic value was '1' (signifying a win). This test involved entering the sequence of digits correctly until all rounds have been completed for all difficulties. This test proves that it is possible to beat the game.

```
"
# Test 5: Try to win the game
#
# Max Digit = 3, WINorLOSE = 1, guesses_left = 0
# Winning state has been achieved
# .main_pane.wave.interior.cs.body.pw.wf
# 0 ns
# 21 us
VSIM 7>
```

Figure 18. FSM Transcript Test 5 (Win the game)

The final testbench was used to verify that all components worked together. The results shown in figure 19 show the outputs that the player would expect when interacting with the board. The outputs that they see are the LEDs pertaining to the round, the LEDs pertaining to the difficulty, and the seven-segments displaying the win or lose message. The test case for winning the game displays the values for roundLED and diffLED. Round 1 is equivalent to roundLED = 1, round 2 is equivalent to roundLED = 3, and round 3 is equivalent to roundLED = 7. The same relationship holds true between difficulty level and diffLED. The testbench displays the status of the 7-segment display, whether the 7-segment shows the appropriate message for the state of the game. According to the results of the test case shown in figure 19, the system was able to achieve a state where: the player wins the game, all round and difficulty LEDs light up, and the seven-segment displays the correct message. The other test cases that pertain to losing the game are not shown in figure 19, but the changes in states for these test cases can be seen in the waveform from figure 20. From the waveform, it is clear that the game values like the round and incorrect guesses only update when the user confirms their guess. Also the FSM states are clearly shown in the waveform, and these states correctly reflect the logic needed to achieve them. Overall, the testbench shows that the game works as intended.



```

Transcript
# Loading work.get_target_number(1000)
# Loading work.sevenSeg(fast)
# .main_pane.structure.interior.cs.body.struct
# .main_pane.objects.interior.cs.body.tree
# 0 ns, Test case for Win Condition
# roundLED = 1, diffLED = 1
#
# roundLED = 3, diffLED = 1
#
# roundLED = 7, diffLED = 1
#
# roundLED = 1, diffLED = 3
#
# roundLED = 3, diffLED = 3
#
# roundLED = 7, diffLED = 3
#
# roundLED = 1, diffLED = 7
#
# roundLED = 3, diffLED = 7
#
# roundLED = 7, diffLED = 7
#
# 7-segment shows the win message
#
# 7-segment does not show game over message 'you lose'
#
# Reset Game
#
# 28650 ns, Test case for Lose Condition on difficulty 1
# roundLED = 1, diffLED = 1
#
# roundLED = 3, diffLED = 1
#
# roundLED = 7, diffLED = 1
#
# roundLED = 7, diffLED = 1
#

```

Transcript × Objects × sim ×

Figure 19. Full system Transcript

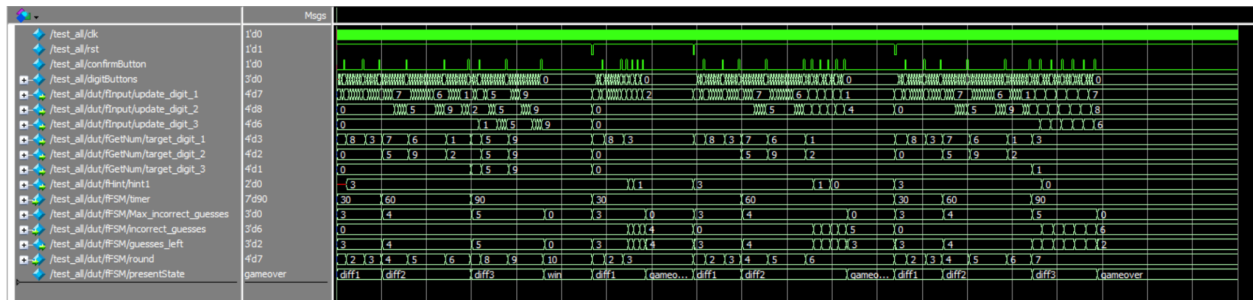


Figure 20. Full system Waveforms

The implementation on the board worked as expected from the behavior seen in simulation. The one aspect of the simulation that was not possible to test in the full design was the use of a clock divider for the timer component. The clock for the time needed to be divided down for the player to see, but that was the only component that ran off the divided clock. Simulation regarding the timer in the full system was not achieved within the given project time due to difficulties. However, the timer worked perfectly with the full system design when implemented on the board. During on-board testing, the timer had a frequency of 1 Hz because we measured the countdown rate to be 1 unit per second.

## Analysis:

All components of the number-guessing game have met the requirements that we proposed, and the game as a whole responds as specified by various user inputs on the board. However, there was a question commonly brought up during the poster session regarding if the game randomly generated the answers for every round. The current design does not make use of any random number generation. Randomized answers would have been difficult to verify on the board due to limited I/O; verifying game logic with predefined answers proved to be greatly beneficial to ensuring the core gameplay elements worked properly. For future work, randomized answers will be a strong focus to replace the fixed values used in the current design.

In regards to the timing analysis of the game, the slack value was reported as 7.379 ns. Figure 21 shows the setup slack waveform. Our design can be clocked faster down to a 12.621 ns period or up to roughly 79 MHz. According to the results shown in figure 22, the longest delay was 9.959 ns. According to the compilation summary shown by figure 23, our design used up 557 logic elements and 191 total registers. According to the recommendations in figure 24, the combinational logic between several registers and the seven-segment outputs contribute to the majority of the longest delay path. If we were to reduce the combinational logic suggested by the timing closure report from figure 24, it would most likely lead to shorter delays, faster clocking potential, and less logic elements required in our design. Lastly, we discovered that our design had redundant connections according to the recommendation shown in figure 25. This type of issue was not apparent in simulation or on-board. We discovered that the timing analyzer serves as a great tool to diagnose these path-related issues in future work.



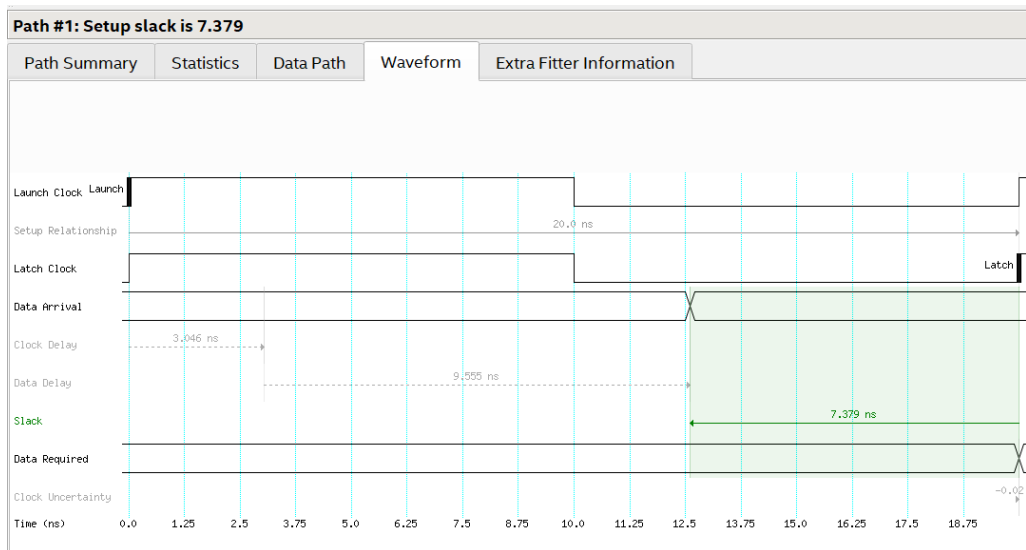


Figure 21. Slack waveform

**Path #1: Delay is 9.959**

Path Summary		Statistics	Data Path				
	Property	Value	Count	Total Delay	% of Total	Min	Max
1	Data Delay	9.959					
2	Number of Logic Levels	9					
3	Physical Delays						
1	IC		10	7.341	74	0.000	1.165
2	Cell		11	2.618	26	0.000	0.429

Figure 22. Path delay summary

Flow Status	Successful - Tue Dec 13 13:13:46 2022
Quartus Prime Version	21.1.1 Build 850 06/23/2022 SJ Lite Edition
Revision Name	Number-Guessing
Top-level Entity Name	top_level
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	557 / 114,480 ( < 1 % )
Total registers	191
Total pins	69 / 529 ( 13 % )
Total virtual pins	0
Total memory bits	0 / 3,981,312 ( 0 % )
Embedded Multiplier 9-bit elements	0 / 532 ( 0 % )
Total PLLs	0 / 4 ( 0 % )

Figure 23. Compilation summary

## Top Recommendations

★★★★★

Reduce the levels of combinational logic for the path from **fsm:fFSM|reg\_max\_digit[0]** to **sevenSeg:f7Seg|seg1[5]**

- **Issue:** [Long Combinational Path](#)
- **From:** fsm:fFSM|reg\_max\_digit[0]
- **To:** sevenSeg:f7Seg|seg1[5]
- **Timing Analysis:** [report timing](#)
- **Extra levels of combinational logic:**
  - 6

★★★★★

Reduce the levels of combinational logic for the path from **fsm:fFSM|reg\_max\_digit[0]** to **sevenSeg:f7Seg|seg2[6]**

- **Issue:** [Long Combinational Path](#)
- **From:** fsm:fFSM|reg\_max\_digit[0]
- **To:** sevenSeg:f7Seg|seg2[6]
- **Timing Analysis:** [report timing](#)
- **Extra levels of combinational logic:**
  - 5

★★★★★

Reduce the levels of combinational logic for the path from **fsm:fFSM|reg\_max\_digit[0]** to **sevenSeg:f7Seg|seg2[6]**

- **Issue:** [Long Combinational Path](#)
- **From:** fsm:fFSM|reg\_max\_digit[0]
- **To:** sevenSeg:f7Seg|seg2[6]
- **Timing Analysis:** [report timing](#)
- **Extra levels of combinational logic:**
  - 5

Figure 24. Recommendations to reduce combinational logic

## Top Recommendations

★★

Duplicate the nodes specified in the details for the path from **fsm:fFSM|reg\_max\_digit[0]** to **sevenSeg:f7Seg|seg2[6]**

- **Issue:** [Inter-path Competition](#)
- **From:** fsm:fFSM|reg\_max\_digit[0]
- **To:** sevenSeg:f7Seg|seg2[6]
- **Timing Analysis:** [report timing](#)
- **Nodes to duplicate:**
  - fsm:fFSM|reg\_max\_digit[0]
  - timer:fTimer|counter[1]~1
  - sevenSeg:f7Seg|Decoder0~12

Figure 25. Recommendation to remove interpath competition

## Task Breakdown:

Ellis:

Responsible for implementing the input control logic module, get target number (get new answer) module, synchronizer and reset modules, and the top level module. Also responsible for drawing out the FSM diagram and system diagram. Worked on flushing out the test cases for all modules, and recording results for the report. Responsible for conducting the timing analysis portion. Helped with the poster content and did some design demonstrations.

Fong:

Responsible for implementing the FSM module, hint (comparator) module, seven-segment module, and timer module. Worked on developing test cases for all the modules. Did the majority of the design demonstration and poster content. Also responsible for I/O planning and testing on the board.

## Conclusion:

Our project was to design a number-guessing game, in which the player toggles between a set of digits to guess the correct sequence of numbers determined by the game logic. The design needed to be broken up into numerous components for modularity and more distinct debugging potentials. The input control module was responsible for controlling which digits were visible and interactable by the player. The timer module was responsible for counting down the time. The hint (comparator) module was responsible for comparing the player's guess with the predefined answers for each round. Lastly, the FSM module was responsible for determining the difficulty level of the game, win or lose state, and miscellaneous game logic values. All components, both separately and together, behaved as designed in simulation. The game functioned as specified in our proposal, and we are confident that all requirements of our project have been met. The one thing that we learned from the development of this project was planning. Initially, we discussed our plans verbally and started coding immediately after. This was the wrong approach because we both made modules that were incompatible with each other on a conceptual level. We learned from this mistake and resolved the issues by drawing out a detailed system diagram and a detailed FSM diagram, such that we would eliminate miscommunication and assumptions in later stages of development.