Whack-a-Mole Game on Basys 3 FPGA

TEAM: LogicForge

MEMBERS:
S Ujwal Niranjan
Sucheth Agar R
Kushal S

Contents

1 Project Overview **2**
1.1 Objective . . . 2
1.2 Scope . . . 2
1.3 Applications . . . 2

2 System Architecture **3**
2.1 Block Diagram . . . 3
2.2 Components . . . 4

3 Modules and Implementation **5**
3.1 Clock Division . . . 5
3.2 Counters . . . 7
3.3 Game Logic . . . 8
3.4 Binary to BCD Conversion . . . 23
3.5 Multiplexer and Decoder . . . 27
3.6 Seven-Segment Display . . . 28
3.7 Top module . . . 29
3.8 Constraints . . . 30

4 Testing and Results **33**
4.1 Simulation Results . . . 33
4.2 Hardware Testing . . . 33

5 Conclusion and Future Work **34**
5.1 Conclusion . . . 34
5.2 Future Enhancements . . . 34

Chapter 1

Project Overview

1.1 Objective

To design and implement a **Whack-a-Mole** game on the Basys 3 FPGA board, utilizing its hardware resources such as LEDs, seven-segment displays, and a clock module. The game includes score tracking and a countdown timer.

1.2 Scope

This project demonstrates the use of FPGA hardware for creating interactive games. It combines multiple hardware modules, including clock dividers, counters, multiplexers, and decoders, to achieve a functional game system.

1.3 Applications

- Educational tools for learning FPGA programming and digital design.
- Interactive gaming applications.
- Demonstration of hardware-based real-time systems.

Chapter 2

System Architecture

2.1 Block Diagram

The system is designed as shown in the block diagram below:

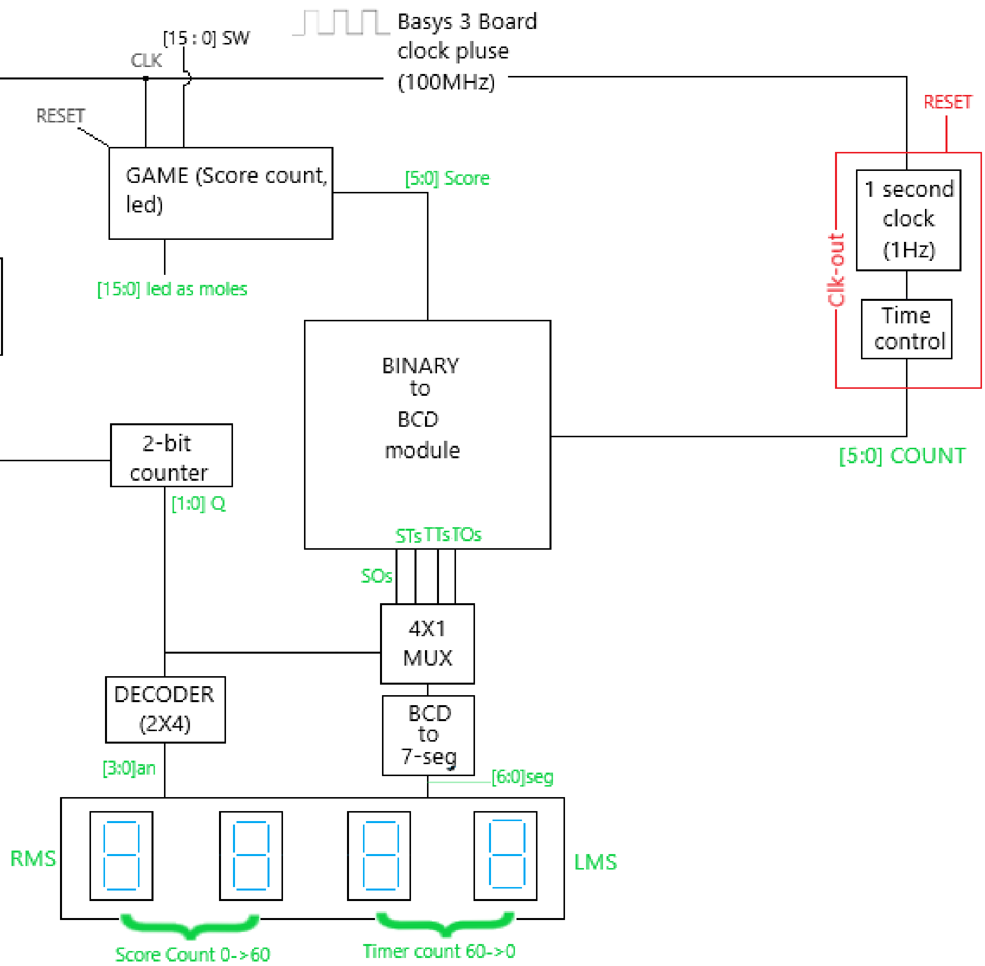


Figure 2.1: Block Diagram of Whack-a-Mole on Basys 3

2.2 Components

1. **Clock Modules**:
   - A 1 Hz clock derived from the 100 MHz board clock to control the countdown timer.
   - A slower clock for controlling LED transitions.
2. **Game Module**:
   - Tracks the user's score based on switch inputs.
   - Uses a 5-bit score counter to update the score.
3. **Binary to BCD Converter**:
   - Converts binary values of the timer and score into BCD format for display.
4. **Multiplexer and Decoder**:
   - The multiplexer selects between timer and score outputs for display.
   - A 2-to-4 decoder drives the seven-segment displays.
5. **Seven-Segment Displays**:
   - Four displays show the countdown timer and score in real-time.
6. **LEDs and Switches**:
   - LEDs simulate the "moles" in the game.
   - Switches serve as inputs for "hitting" the mole.

Chapter 3

Modules and Implementation

3.1 Clock Division

Objective: Generate slower clocks from the 100 MHz onboard clock.

Implementation: A clock divider module is designed to generate a 1 Hz clock for the countdown timer and a slower clock for LED transitions.

Listing 3.1: 0.01 seconds clock

```
module slowclk(
    input clk,
    output reg clk_out
    );
    reg [18:0] counter;
    always @(posedge clk)
    begin
    counter <= counter + 1;
    if(counter == 500_000)
    begin
        counter <= 0;
        clk_out = ~clk_out;
    end
    end
endmodule
```

Listing 3.2: 1 second clock

```
module sec_clk(
    input clk,
    output reg clk_o
    );
    reg [25:0] counter;
    always @(posedge clk)
    begin
    counter <= counter + 1;
    if(counter == 50_000_000)
    begin
        counter <= 0;
```

```
            clk_o = ~clk_o;
        end
    end
endmodule
```

3.2 Counters

Objective: There are 2 counters. 1st for the timer and the 2nd for the 2-bit counter.

Implementation: 1st counter counts from 60 to 0. 2nd counter controls the mux selectline and also the input to the decoder.

Listing 3.3: Timer counter(downcounter)

```
module timer(
    input clk_o,
    input rst,
    output reg [5:0] count
    );
    always @(clk_o)
    begin
    count = 6'b111100;
    if(rst == 1)
        count <= 6'b111100;
    else
        count <= count - 1;
    end
endmodule
```

Listing 3.4: 2-bit counter(upcounter)

```
module counter(
    input clk_out,
    output reg [1:0]Q
    );
    always @(posedge clk_out)
    begin
    if(Q == 3)
        Q = 0;
    else
        Q = Q + 1;
    end
endmodule
```

3.3 Game Logic

Objective: Track the score based on user inputs.

Implementation: A 5-bit counter increments the score when the correct switch is pressed. The module also interfaces with the LED controller to simulate mole behavior.

Listing 3.5: Game

```
module game(
    input [15:0] sw,
    input clk,rst,
    output reg [15:0] led,
    output reg [5:0] score
    );

localparam
S000000 = 0,
S000001 = 1,
S000010 = 2,
S000011 = 3,
S000100 = 4,
S000101 = 5,
S000110 = 6,
S000111 = 7,
S001000 = 8,
S001001 = 9,
S001010 = 10,
S001011 = 11,
S001100 = 12,
S001101 = 13,
S001110 = 14,
S001111 = 15,
S010000 = 16,
S010001 = 17,
S010010 = 18,
S010011 = 19,
S010100 = 20,
S010101 = 21,
S010110 = 22,
S010111 = 23,
S011000 = 24,
S011001 = 25,
S011010 = 26,
S011011 = 27,
S011100 = 28,
S011101 = 29,
S011110 = 30,
S011111 = 31,
S100000 = 32,
S100001 = 33,
```

```
S100010 = 34,
S100011 = 35,
S100100 = 36,
S100101 = 37,
S100110 = 38,
S100111 = 39,
S101000 = 40,
S101001 = 41,
S101010 = 42,
S101011 = 43,
S101100 = 44,
S101101 = 45,
S101110 = 46,
S101111 = 47,
S110000 = 48,
S110001 = 49,
S110010 = 50,
S110011 = 51,
S110100 = 52,
S110101 = 53,
S110110 = 54,
S110111 = 55,
S111000 = 56,
S111001 = 57,
S111010 = 58,
S111011 = 59,
S111100 = 60;

    reg [6:0] current_state = 0;
    reg [6:0] next_state = 0;

    always @(posedge clk)
    begin
    if(rst)
        current_state <= S000000;
    else
        current_state <= next_state;
    end


    always @(current_state, sw[15:0])
    begin
    case(current_state)
    S000000:
        begin
        next_state <= S000000;
        led[15:0] = 0;
        score = 6'b000000;
```

```
        led[3] <= 1;
            if(sw[3])
                next_state <= S000001;
            else
                next_state <= S000000;
        end
    S000001:
        begin
        score <= 6'b000001;
        led[15:0] = 0;
        led[12] <= 1;
        if(sw[12])
            next_state <= S000100;
        else
            next_state <= S000001;
        end
    S000010:
        begin
        score <= 6'b000010;
        led[15:0] = 0;
        led[7] <= 1;
        if(sw[7])
            next_state <= S000011;
        else
            next_state <= S000010;
        end
    S000011:
        begin
        score <= 6'b000011;
        led[15:0] = 0;
        led[5] <= 1;
        if(sw[5])
            next_state <= S000100;
        else
            next_state <= S000011;
        end
    S000100:
        begin
        score <= 6'b000100;
        led[15:0] = 0;
        led[1] <= 1;
        if(sw[1])
            next_state <= S000101;
        else
            next_state <= S000100;
        end
    S000101:
        begin
```

```
        score <= 6'b0000101;
        led[15:0] = 0;
        led[11] <= 1;
        if(sw[11])
            next_state = S000110;
        else
            next_state <= S000101;
        end
    S000110:
        begin
        score <= 6'b000110;
        led[15:0] = 0;
        led[15] <= 1;
        if(sw[15])
            next_state <= S000111;
        else
            next_state <= S000110;
        end
    S000111:
        begin
        score <= 6'b000111;
        led[15:0] = 0;
        led[14] <= 1;
        if(sw[14])
            next_state <= S001000;
        else
            next_state <= S000111;
        end
    S001000:
        begin
        score <= 6'b001000;
        led[15:0] = 0;
        led[6] <= 1;
        if(sw[6])
            next_state <= S001001;
        else
            next_state <= S001000;
        end
    S001001:
        begin
        score <= 6'b001001;
        led[15:0] = 0;
        led[9] <= 1;
        if(sw[9])
            next_state <= S001010;
        else
            next_state <= S001001;
        end
```

```
S001010:
    begin
    score <= 6'b001010;
    led[15:0] = 0;
    led[4] <= 1;
    if(sw[4])
        next_state <= S001011;
    else
        next_state <= S001010;
    end
S001011:
    begin
    score <= 6'b001011;
    led[15:0] = 0;
    led[8] <= 1;
    if(sw[8])
        next_state <= S001100;
    else
        next_state <= S001011;
    end
S001100:
    begin
    score <= 6'b001100;
    led[15:0] = 0;
    led[13] <= 1;
    if(sw[13])
        next_state <= S001101;
    else
        next_state <= S001100;
    end
S001101:
    begin
    score <= 6'b001101;
    led[15:0] = 0;
    led[10] <= 1;
    if(sw[10])
        next_state <= S001110;
    else
        next_state <= S001101;
    end
S001110:
    begin
    score <= 6'b001110;
    led[15:0] = 0;
    led[2] <= 1;
    if(sw[2])
        next_state <= S001111;
    else
```

```
            next_state <= S001110;
        end
    S001111:
        begin
        score <= 6'b001111;
        led[15:0] = 0;
        led[1] <= 1;
        if(sw[1])
            next_state <= S010000;
        else
            next_state <= S001111;
        end
    S010000:
        begin
        score <= 6'b010000;
        led[15:0] = 0;
        led[6] <= 1;
        if(sw[6])
            next_state <= S010001;
        else
            next_state <= S010000;
        end
     S010001://this is where you should stare
        begin
        score <= 6'b010001;
        led[15:0] = 0;
        led[5] <= 1;
        if(sw[5])
            next_state <= S010010;
        else
            next_state <= S010001;
        end
     S010010:
        begin
        score <= 6'b010010;
        led[15:0] = 0;
        led[8] <= 1;
        if(sw[8])
            next_state <= S010011;
        else
            next_state <= S010010;
        end
     S010011:
        begin
        score <= 6'b010011;
        led[15:0] = 0;
        led[9] <= 1;
        if(sw[9])
```

```
            next_state <= S010100;
        else
            next_state <= S010011;
        end
    S010100:
        begin
        score <= 6'b010100;
        led[15:0] = 0;
        led[7] <= 1;
        if(sw[7])
            next_state <= S010101;
        else
            next_state <= S010100;
        end
    S010101:
        begin
        score <= 6'b010101;
        led[15:0] = 0;
        led[11] <= 1;
        if(sw[11])
            next_state <= S010110;
        else
            next_state <= S010101;
        end
    S010110:
        begin
        score <= 6'b010110;
        led[15:0] = 0;
        led[3] <= 1;
        if(sw[3])
            next_state <= S010111;
        else
            next_state <= S010110;
        end
    S010111:
        begin
        score <= 6'b010111;
        led[15:0] = 0;
        led[15] <= 1;
        if(sw[15])
            next_state <= S011000;
        else
            next_state <= S010111;
        end
    S011000:
        begin
        score <= 6'b011000;
        led[15:0] = 0;
```

```
        led[13] <= 1;
        if(sw[13])
            next_state <= S011001;
        else
            next_state <= S011000;
        end
    S011001:
        begin
        score <= 6'b011001;
        led[15:0] = 0;
        led[4] <= 1;
        if(sw[4])
            next_state <= S011010;
        else
            next_state <= S011001;
        end
    S011010:
        begin
        score <= 6'b011010;
        led[15:0] = 0;
        led[12] <= 1;
        if(sw[12])
            next_state <= S011011;
        else
            next_state <= S011010;
        end
    S011011:
        begin
        score <= 6'b011011;
        led[15:0] = 0;
        led[2] <= 1;
        if(sw[2])
            next_state <= S011100;
        else
            next_state <= S011011;
        end
    S011100:
        begin
        score <= 6'b011100;
        led[15:0] = 0;
        led[10] <= 1;
        if(sw[10])
            next_state <= S011101;
        else
            next_state <= S011100;
        end
    S011101:
        begin
```

```
        score <= 6'b011101;
        led[15:0] = 0;
        led[14] <= 1;
        if(sw[14])
            next_state <= S011110;
        else
            next_state <= S011101;
        end
    S011110:
        begin
        score <= 6'b011110;
        led[15:0] = 0;
        led[1] <= 1;
        if(sw[1])
            next_state <= S011111;
        else
            next_state <= S011110;
        end
    S011111:
        begin
        score <= 6'b011111;
        led[15:0] = 0;
        led[5] <= 1;
        if(sw[5])
            next_state <= S100000;
        else
            next_state <= S011111;
        end
    S100000:
        begin
        score <= 6'b100000;
        led[15:0] = 0;
        led[3] <= 1;
        if(sw[3])
            next_state <= S100001;
        else
            next_state <= S100000;
        end
    S100001:
        begin
        score <= 6'b100001;
        led[15:0] = 0;
        led[8] <= 1;
        if(sw[8])
            next_state <= S100010;
        else
            next_state <= S100001;
        end
```

```
S100010:
    begin
    score <= 6'b100010;
    led[15:0] = 0;
    led[6] <= 1;
    if(sw[6])
        next_state <= S100011;
    else
        next_state <= S100010;
    end
S100011:
    begin
    score <= 6'b100011;
    led[15:0] = 0;
    led[2] <= 1;
    if(sw[2])
        next_state <= S100100;
    else
        next_state <= S100011;
    end
S100100:
    begin
    score <= 6'b100100;
    led[15:0] = 0;
    led[12] <= 1;
    if(sw[12])
        next_state <= S100101;
    else
        next_state <= S100100;
    end
S100101:
    begin
    score <= 6'b100101;
    led[15:0] = 0;
    led[15] <= 1;
    if(sw[15])
        next_state <= S100110;
    else
        next_state <= S100101;
    end
S100110:
    begin
    score <= 6'b100110;
    led[15:0] = 0;
    led[9] <= 1;
    if(sw[9])
        next_state <= S100111;
    else
```

```
            next_state <= S100110;
        end
    S100111:
        begin
        score <= 6'b010110;
        led[15:0] = 0;
        led[7] <= 1;
        if(sw[7])
            next_state <= S101000;
        else
            next_state <= S100111;
        end
    S101000:
        begin
        score <= 6'b101000;
        led[15:0] = 0;
        led[4] <= 1;
        if(sw[4])
            next_state <= S101001;
        else
            next_state <= S101000;
        end
    S101001:
        begin
        score <= 6'b101001;
        led[15:0] = 0;
        led[10] <= 1;
        if(sw[10])
            next_state <= S101010;
        else
            next_state <= S101001;
        end
    S101010:
        begin
        score <= 6'b101010;
        led[15:0] = 0;
        led[13] <= 1;
        if(sw[13])
            next_state <= S101011;
        else
            next_state <= S101010;
        end
    S101011:
        begin
        score <= 6'b101011;
        led[15:0] = 0;
        led[11] <= 1;
        if(sw[11])
```

```
            next_state <= S101100;
        else
            next_state <= S101011;
        end
    S101100:
        begin
        score <= 6'b101100;
        led[15:0] = 0;
        led[14] <= 1;
        if(sw[14])
            next_state <= S101101;
        else
            next_state <= S101100;
        end
    S101101:
        begin
        score <= 6'b101101;
        led[15:0] = 0;
        led[6] <= 1;
        if(sw[6])
            next_state <= S101110;
        else
            next_state <= S101101;
        end
    S101110:
        begin
        score <= 6'b101110;
        led[15:0] = 0;
        led[9] <= 1;
        if(sw[9])
            next_state <= S101111;
        else
            next_state <= S101110;
        end
    S101111:
        begin
        score <= 6'b101111;
        led[15:0] = 0;
        led[2] <= 1;
        if(sw[2])
            next_state <= S110000;
        else
            next_state <= S101111;
        end
    S110000:
        begin
        score <= 6'b110000;
        led[15:0] = 0;
```

```
        led[8] <= 1;
        if(sw[8])
            next_state <= S110001;
        else
            next_state <= S110000;
        end
    S110001:
        begin
        score <= 6'b110001;
        led[15:0] = 0;
        led[13] <= 1;
        if(sw[13])
            next_state <= S110010;
        else
            next_state <= S110001;
        end
    S110010:
        begin
        score <= 6'b110010;
        led[15:0] = 0;
        led[15] <= 1;
        if(sw[15])
            next_state <= S110011;
        else
            next_state <= S110010;
        end
    S110011:
        begin
        score <= 6'b110011;
        led[15:0] = 0;
        led[3] <= 1;
        if(sw[3])
            next_state <= S110100;
        else
            next_state <= S110011;
        end
    S110100:
        begin
        score <= 6'b110100;
        led[15:0] = 0;
        led[12] <= 1;
        if(sw[12])
            next_state <= S110101;
        else
            next_state <= S110100;
        end
    S110101:
        begin
```

```
        score <= 6'b110101;
        led[15:0] = 0;
        led[1] <= 1;
        if(sw[1])
            next_state <= S110110;
        else
            next_state <= S110101;
        end
    S110110:
        begin
        score <= 6'b110110;
        led[15:0] = 0;
        led[5] <= 1;
        if(sw[5])
            next_state <= S110111;
        else
            next_state <= S110110;
        end
    S110111:
        begin
        score <= 6'b110111;
        led[15:0] = 0;
        led[7] <= 1;
        if(sw[7])
            next_state <= S111000;
        else
            next_state <= S110111;
        end
    S111000:
        begin
        score <= 6'b111000;
        led[15:0] = 0;
        led[10] <= 1;
        if(sw[10])
            next_state <= S111001;
        else
            next_state <= S111000;
        end
    S111001:
        begin
        score <= 6'b111001;
        led[15:0] = 0;
        led[4] <= 1;
        if(sw[4])
            next_state <= S111010;
        else
            next_state <= S111001;
        end
```

```
        S111010:
            begin
            score <= 6'b111010;
            led[15:0] = 0;
            led[14] <= 1;
            if(sw[14])
                next_state <= S111011;
            else
                next_state <= S111010;
            end
        S111011:
            begin
            score <= 6'b111011;
            led[15:0] = 0;
            led[11] <= 1;
            if(sw[11])
                next_state <= S111100;
            else
                next_state <= S111011;
            end
        S111100:
            begin
            next_state <= S111100;
            score <= 6'b111100;
            led[15:0] <= 0;
            end
        endcase
        end
endmodule
```

3.4 Binary to BCD Conversion

Objective: Convert binary values of score and timer into BCD format.

Implementation: A binary to BCD conversion module is implemented to ensure compatibility with the seven-segment displays.

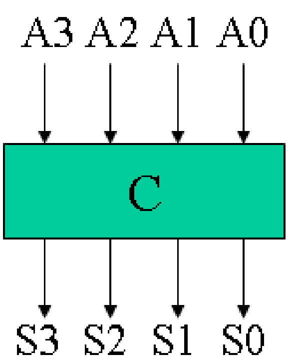
- Shift the binary number left one bit.
- If 8 shifts have taken place, the BCD number is in the Hundreds, Tens, and Units column.
- If the binary value in any of the BCD columns is 5 or greater, add 3 to that value in that BCD column.
- Go to 1.

Operation	Tens	Units	Binary
HEX			**E**
Start			1 1 1 0
Shift 1		1	1 1 0
Shift 2		1 1	1 0
Shift 3		1 1 1	0
Add 3		1 0 1 0	0
Shift 4	1	0 1 0 0	
BCD	**1**	**4**	

Figure 3.1: Convert hex E to BCD

Operation	Hundreds	Tens	Units	Binary	
HEX				F	F
Start				1 1 1 1	1 1 1 1
Shift 1			1	1 1 1 1	1 1 1
Shift 2			1 1	1 1 1 1	1 1
Shift 3			1 1 1	1 1 1 1	1
Add 3			1 0 1 0	1 1 1 1	1
Shift 4		1	0 1 0 1	1 1 1 1	
Add 3		1	1 0 0 0	1 1 1 1	
Shift 5		1 1	0 0 0 1	1 1 1	
Shift 6		1 1 0	0 0 1 1	1 1	
Add 3		1 0 0 1	0 0 1 1	1 1	
Shift 7	1	0 0 1 0	0 1 1 1	1	
Add 3	1	0 0 1 0	1 0 1 0	1	
Shift 8	1 0	0 1 0 1	0 1 0 1		
BCD	2	5	5		

Figure 3.2: Convert hex FF to BCD



A3	A2	A1	A0	S3	S2	S1	S0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Figure 3.3: Truth table for Add-3 Module

Listing 3.6: Shift add 3 algorithm

```
module shift_add3(
    input [3:0] in,
    output reg [3:0] out
    );
    always @(in)
    begin
    case(in)
    4'b0000 : out <= 4'b0000;
    4'b0001 : out <= 4'b0001;
    4'b0010 : out <= 4'b0010;
    4'b0011 : out <= 4'b0011;
    4'b0100 : out <= 4'b0100;
    4'b0101 : out <= 4'b1000;
    4'b0110 : out <= 4'b1001;
    4'b0111 : out <= 4'b1010;
    4'b1000 : out <= 4'b1011;
    4'b1001 : out <= 4'b1100;
    endcase
    end
endmodule
```

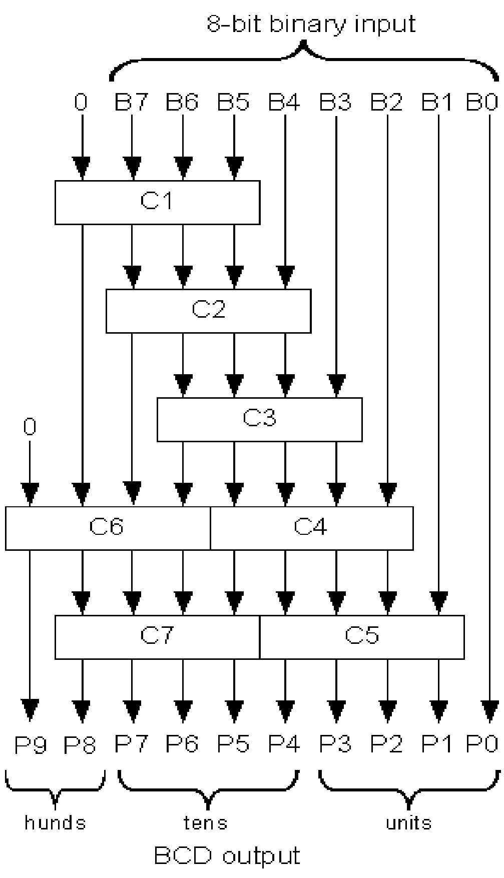


Figure 3.4: Binary-to-BCD Converter Module

Listing 3.7: Binary to BCD

```
module Bin_to_Bcd(
    input [5:0] inp,
    output [3:0] ones,
    output [3:0] tens
    );
    wire [3:0] c1,c2,c3,c4,c5,c6,c7;
    wire [3:4] d1,d2,d3,d4,d5,d6,d7;

    assign d1 = {3'b000, inp [5]};
    assign d2 = {c1[2:0], inp[4]};
    assign d3 = {c2[2:0], inp[3]};
    assign d4 = {c3[2:0], inp[2]};
    assign d5 = {c4[2:0], inp[1]};
    assign d6 = {1'b0,c1[3],c2[3],c3[3]};
    assign d7 = {c6[2:0],c4[3]};

    shift_add3 SA1(.in(d1),.out(c1));
    shift_add3 SA2(.in(d2),.out(c2));
    shift_add3 SA3(.in(d3),.out(c3));
    shift_add3 SA4(.in(d4),.out(c4));
    shift_add3 SA5(.in(d5),.out(c5));
    shift_add3 SA6(.in(d6),.out(c6));
    shift_add3 SA7(.in(d7),.out(c7));

    assign ones = {c5[2:0],inp[0]};
    assign tens = {c7[2:0],c5[3]};
endmodule
```

3.5 Multiplexer and Decoder

Objective: Manage the display of timer and score on the seven-segment displays.

Implementation:

- A 4-to-1 multiplexer selects between timer and score outputs.
- A 2-to-4 decoder drives the seven-segment displays.

Listing 3.8: Decoder

```
module decoder(
    input [1:0] Q,
    output reg [3:0] an
    );
    always @(Q)
    begin
    case(Q)
    2'b00: an <= 4'b0001;
    2'b01: an <= 4'b0010;
    2'b10: an <= 4'b0100;
    2'b11: an <= 4'b1000;
    endcase
    end
endmodule
```

Listing 3.9: Multiplexer

```
module mux(
    input [3:0] To,Tt,So,St,
    input [1:0] Q,
    output reg [3:0] Y
    );
    always @(To or Tt or So or St or Q)
    begin
    case(Q)
    2'b00 :  Y = Tt;
    2'b01 :  Y = To;
    2'b10 :  Y = St;
    2'b11 :  Y = So;
    endcase
    end
endmodule
```

3.6 Seven-Segment Display

Objective: Display the countdown timer and score.

Implementation: A BCD-to-seven-segment decoder drives the displays. Each segment is updated in real-time based on the game logic.

Listing 3.10: BCD to seven segment

```
module bcd_to_7seg(
    input [3:0] Y,
    output reg [6:0] seg
    );
    always @(Y)
    begin
    case(Y)
    4'b0000 : seg <= 7'b0000001;
    4'b0001 : seg <= 7'b1001111;
    4'b0010 : seg <= 7'b0010010;
    4'b0011 : seg <= 7'b0000110;
    4'b0100 : seg <= 7'b1001100;
    4'b0101 : seg <= 7'b0100100;
    4'b0110 : seg <= 7'b0100000;
    4'b0111 : seg <= 7'b0001111;
    4'b1000 : seg <= 7'b0000000;
    4'b1001 : seg <= 7'b0001100;
    endcase
    end
endmodule
```

3.7 Top module

Objective: Integrate all the modules.

Implementation: All the modules are initialized in this module and the input and output of the system is declared.

Listing 3.11: Top Module

```
module top_module(
    input clk,rst,
    input [15:0] sw,
    output [15:0] led,
    output [6:0] seg,
    output [3:0] an
    );

    wire clk_out;
    wire timer_clk_out;
    wire [3:0] mux_out;
    wire [1:0] counter_out;
    wire [3:0] To,Tt,So,St;
    wire [5:0] score;
    wire [5:0] timer;

    Bin_to_Bcd UBCD1(.inp(score),.ones(So),.tens(St));
    Bin_to_Bcd UBCD2(.inp(timer),.ones(To),.tens(Tt));

    mux UMUX(.To(To),.Tt(Tt),.So(So),.St(St),.Q(counter_out),.Y(mux_out));

    slowclk USCk(.clk(clk),.clk_out(clk_out));

    counter U2BC(.clk_out(clk_out),.Q(counter_out));

    decoder UD(.Q(counter_out),.an(an));

    bcd_to_7seg UBCDSEG(.Y(mux_out),.seg(seg));

    sec_clk USC(.clk(clk),.clk_o(timer_clk_out));

    timer UT(.clk_o(timer_clk_out),.rst(rst),.count(timer));

    game UG(.sw(sw),.clk(clk),.led(led),.rst(rst),.score(score));

endmodule
```

3.8 Constraints

Objective: Show where the inputs and outputs of the module is present on the basys 3 board.

Implementation: All the ports are mapped on the basys 3 Board.

Listing 3.12: Constraints

```
# Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
        set_property IOSTANDARD LVCMOS33 [get_ports clk]
        create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [

# Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]
set_property PACKAGE_PIN T2 [get_ports {sw[10]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[10]}]
set_property PACKAGE_PIN R3 [get_ports {sw[11]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[11]}]
set_property PACKAGE_PIN W2 [get_ports {sw[12]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[12]}]
set_property PACKAGE_PIN U1 [get_ports {sw[13]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[13]}]
set_property PACKAGE_PIN T1 [get_ports {sw[14]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[14]}]
set_property PACKAGE_PIN R2 [get_ports {sw[15]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[15]}]


# LEDs
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
```

```
        set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
set_property PACKAGE_PIN W18 [get_ports {led[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
set_property PACKAGE_PIN U15 [get_ports {led[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
set_property PACKAGE_PIN U14 [get_ports {led[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
set_property PACKAGE_PIN V13 [get_ports {led[8]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[8]}]
set_property PACKAGE_PIN V3 [get_ports {led[9]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[9]}]
set_property PACKAGE_PIN W3 [get_ports {led[10]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[10]}]
set_property PACKAGE_PIN U3 [get_ports {led[11]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[11]}]
set_property PACKAGE_PIN P3 [get_ports {led[12]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[12]}]
set_property PACKAGE_PIN N3 [get_ports {led[13]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[13]}]
set_property PACKAGE_PIN P1 [get_ports {led[14]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[14]}]
set_property PACKAGE_PIN L1 [get_ports {led[15]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[15]}]



#7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
```

```
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]


set_property PACKAGE_PIN U2 [get_ports {an[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]


##Buttons
set_property PACKAGE_PIN U18 [get_ports rst]
        set_property IOSTANDARD LVCMOS33 [get_ports rst]
```

Chapter 4

Testing and Results

4.1 Simulation Results

The individual modules were simulated using Vivado. The following results were observed:

- Clock divider outputs verified for 1 Hz and slower clock frequencies.
- Game logic accurately tracked scores and timer functionality.
- Binary to BCD conversion correctly translated binary values for display.
- Multiplexer and decoder outputs successfully controlled the seven-segment displays.

4.2 Hardware Testing

The design was implemented on the Basys 3 FPGA board, and the following were observed:

- Real-time updates on the seven-segment displays for timer and score.
- LED transitions simulated mole behavior effectively.
- Switch inputs registered user actions accurately.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The Whack-a-Mole game was successfully implemented on the Basys 3 FPGA board. It demonstrated the effective use of hardware resources for real-time applications and provided an engaging way to interact with FPGA hardware.

5.2 Future Enhancements

- Add more complex game logic to increase difficulty levels.
- Implement sound feedback for user interactions.
- Explore power optimization techniques for the design.