Final Project: Decimal/Hex to Binary Game

Aiyana Arnobit and Vincent Pheng

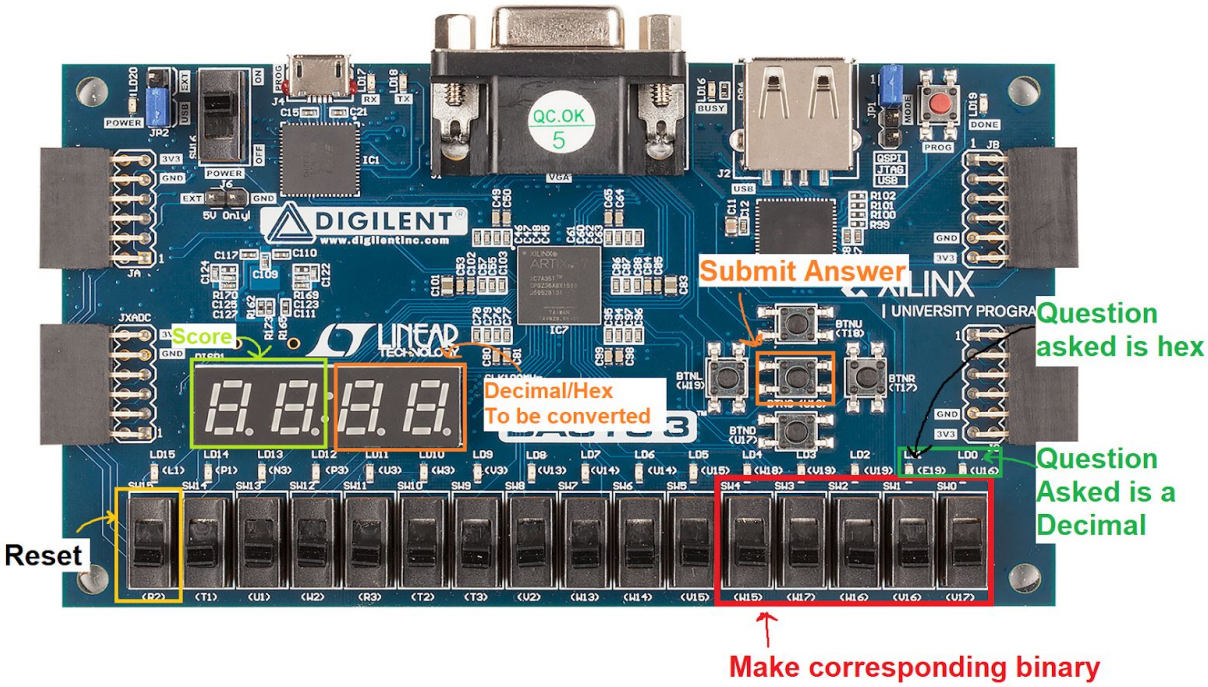
Project Description

For the final project, we have designed a game on the Basys 3 Board where you input the corresponding binary to the decimal/hex number displayed on the 7-segment LED. In total, there are 10 questions the player has to answer. In order to determine if your question is in decimal or hexadecimal, it turns on its corresponding LED, LED0 for decimal and LED1 for hex. To input your answer, you toggle switches 0-4 on and off to create the binary sequence and you press a button to submit your answer. If the answer is correct, your score increases by 1 and goes to the next question. If not, you still get brought to the next question but you don't get the point. Below are the questions asked along with the correct answers for them.

Questions:

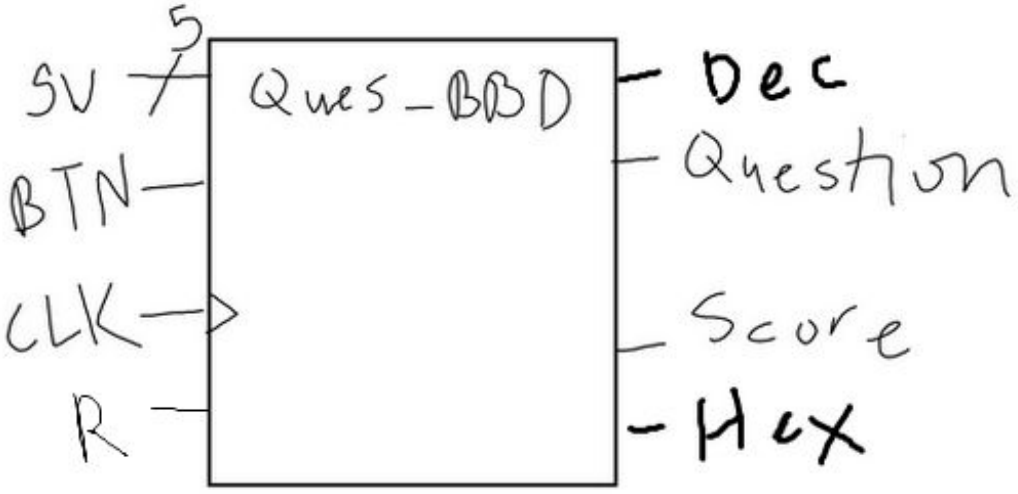
1. 29 - Decimal
    a. Answer: 11101
2. 0D - Hex
    a. Answer: 01101
3. 21 - Decimal
    a. Answer: 10101
4. 13 - Hex
    a. Answer: 10011
5. 09 - Decimal
    a. Answer: 01001
6. 1A - Hex
    a. Answer: 11010
7. 20 - Decimal
    a. Answer: 10100
8. 15 - Hex
    a. Answer: 10101
9. 31 - Decimal
    a. Answer: 11111
10. 11 - Hex
    a. Answer: 10001

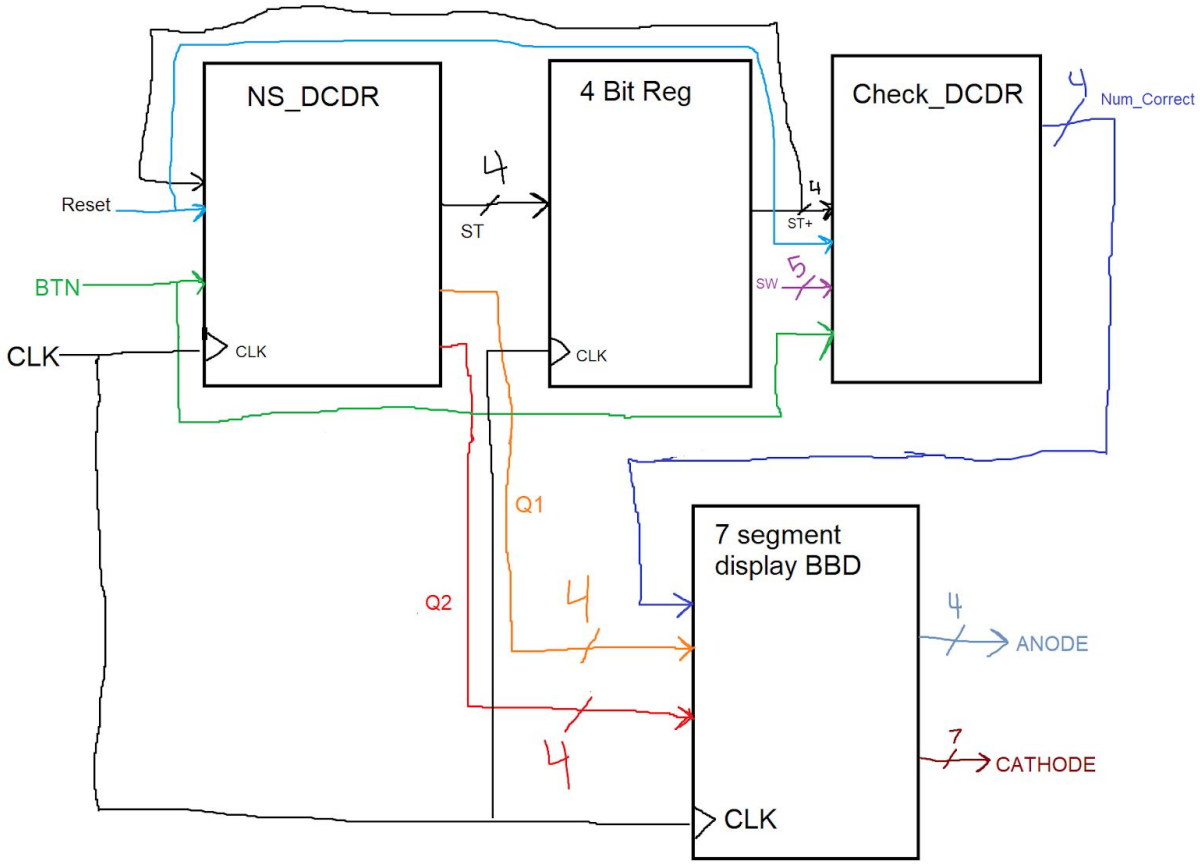
This game is placed on the interface of the Basys 3 Board. Below, are the controls mapped out onto the Basys 3 Board.

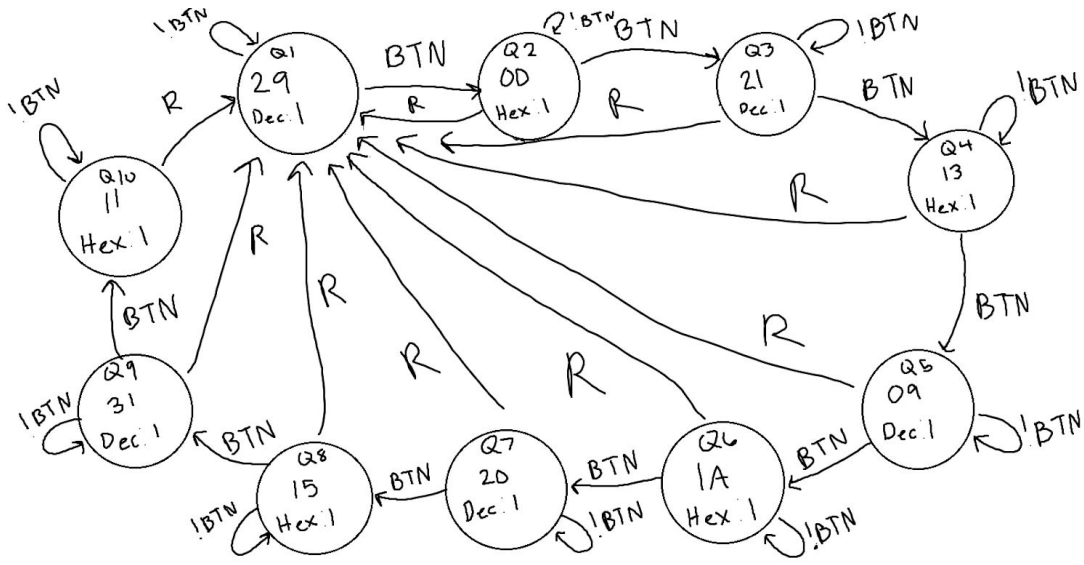


Design

Below are both high and low level designs of our project. It utilizes an FSM and a decoder to control the 7-segment LED display. In the FSM, we use a next state decoder, 4 bit register, and an output decoder.







This is a state diagram of how the FSM functions and how the inputs control the machine traverses through each state.

Code:

DriverFile

```
module DriverFile(
	input R, input BTN, input CLK,
	input reg [4:0] SW,
	output reg [6:0] CATHODE, output reg [3:0] ANODE,
	output HEX, output DEC
	);
	wire [3:0] st;
	wire [3:0] stp;
	reg [3:0] correct;
	reg [3:0] Q1;
	reg [3:0] Q2;

	NS_DCDR ns(.R(R), .BTN(BTN), .CLK(CLK), .STIN(st), .STO(stp), .HEX(HEX),
.DEC(DEC), .Q1(Q1), .Q2(Q2));
	FourReg fr(.A(stp), .CLK(CLK), .R(R), .OUT(st));
	CheckDCDR ch(.IN(st), .BTN(BTN), .CLK(CLK), .R(R), .SW0(SW[0]), .SW1(SW[1]),
.SW2(SW[2]), .SW3(SW[3]), .SW4(SW[4]), .NUM_CORRECT(correct));
	TranslateSevenSegment ss(.CLK(CLK), .Q1(Q1), .Q2(Q2), .CORRECT(correct),
.CATHODE(CATHODE), .ANODE(ANODE));


endmodule
```

NS_DCDR

```
//general structure taken from lab 6 but modified for this project
module NS_DCDR(
	input R, input BTN,
	input [3:0] STIN, input CLK,
	output reg [3:0] STO,
	output HEX, output DEC,
	output reg [3:0] Q1,
	output reg [3:0] Q2
	);
	reg [4:0] state = 0;
	reg [4:0] valid;
	reg [19:0] counter = 0;
	reg anodeCounter;


	//determines state based on state-in value
	always_ff @ (STIN)
	case({STIN})
	4'b0001 : state <= 1;
	4'b0010 : state <= 2;
	4'b0011 : state <= 3;
	4'b0100 : state <= 4;
	4'b0101 : state <= 5;
	4'b0110 : state <= 6;
	4'b0111 : state <= 7;
	4'b1000 : state <= 8;
	4'b1001 : state <= 9;
	4'b1010 : state <= 10;
	default : state <= 1;
	endcase


	//determines how it moves to each state. Only runs on the positive edge of the button
press or the reset trigger
	always_ff @ (posedge BTN, posedge R)
	begin
	if (R == 1) STO <= 4'b0001;
	else if (state == 1 & R == 0) STO <= 4'b0010;
```

```
        else if (state == 2 & R == 0) STO <= 4'b0011;
        else if (state == 3 & R == 0) STO <= 4'b0100;
        else if (state == 4 & R == 0) STO <= 4'b0101;
        else if (state == 5 & R == 0) STO <= 4'b0110;
        else if (state == 6 & R == 0) STO <= 4'b0111;
        else if (state == 7 & R == 0) STO <= 4'b1000;
        else if (state == 8 & R == 0) STO <= 4'b1001;
        else if (state == 9 & R == 0) STO <= 4'b1010;
        else STO <= STIN;
        end

//determines what question will be displayed based on the state it's currently in
        always_ff @ (STIN)
        begin
        case({STIN})
        4'b0001 : begin
                Q1 <= 4'h2;
                Q2 <= 4'h9;
                end
        4'b0010 : begin
                Q1 <= 4'h0;
                Q2 <= 4'hd;
                end
        4'b0011 : begin
                Q1 <= 4'h2;
                Q2 <= 4'h1;
                end
        4'b0100 : begin
                Q1 <= 4'h1;
                Q2 <= 4'h3;
                end
        4'b0101 : begin
                Q1 <= 4'h0;
                Q2 <= 4'h9;
                end
        4'b0110 : begin
                Q1 <= 4'h1;
                Q2 <= 4'ha;
                end
```

```
        4'b0111 : begin
                Q1 <= 4'h2;
                Q2 <= 4'h0;
                end
        4'b1000 : begin
                Q1 <= 4'h1;
                Q2 <= 4'h5;
                end
        4'b1001 : begin
                Q1 <= 4'h3;
                Q2 <= 4'h1;
                end
        4'b1010 : begin
                Q1 <= 4'h1;
                Q2 <= 4'h1;
                end
        default: begin
                Q1 <= 4'h0;
                Q2 <= 4'h0;
                end
        endcase

        end

        assign HEX = ((state % 2) == 0) ? 1 : 0;
        assign DEC = ((state % 2) == 1) ? 1 : 0;


endmodule
```

Translate to Seven Segment

```
module TranslateSevenSegment(input CLK, input [3:0] Q1, input [3:0] Q2, input reg [3:0]
CORRECT,
        output reg [6:0] CATHODE, output reg [3:0] ANODE = 0

        );
        reg [20:0] counter = 0;
        reg [1:0] anodeCounter;

        always_ff @ (posedge CLK)
        begin
        counter <= counter + 1;
        end

        assign anodeCounter = counter[20:19];

        always_ff @ (anodeCounter)
        case(anodeCounter)
        2'b00 : ANODE = 4'b1110;
        2'b01 : ANODE = 4'b1101;
        2'b10 : ANODE = 4'b1011;
        2'b11 : ANODE = 4'b0111;
        default: ANODE = 4'b1111;
        endcase

        reg [3:0] ONE_DIGIT = 0;
        always@(anodeCounter)
        begin
        case(anodeCounter)
        2'b00: ONE_DIGIT = Q2;
        2'b01: ONE_DIGIT = Q1;
        2'b10: begin
                if (CORRECT < 4'ha) ONE_DIGIT <= CORRECT;
                else ONE_DIGIT <= 0;
                end
        2'b11: begin
                if (CORRECT < 4'ha) ONE_DIGIT <= 0;
                else ONE_DIGIT <= 1;
                end
```

```
        default: ONE_DIGIT = 4'b1111;
        endcase
        end

        TranslateToCathode cathodeTranslate(.digit(ONE_DIGIT), .CATHODE(CATHODE));
endmodule
```

TranslateToCathode

```
//taken from lab 4 for displaying on the 7 segment LED
module TranslateToCathode (input reg [3:0] digit, output reg [6:0] CATHODE = 0);
	always_ff @ (digit)
	begin
	//begins case statements that turns on the leds based on the bcd value outputted
	case(digit)
	4'h0 : CATHODE <= 7'b1000000;
	4'h1 : CATHODE <= 7'b1111001;
	4'h2 : CATHODE <= 7'b0100100;
	4'h3 : CATHODE <= 7'b0110000;
	4'h4 : CATHODE <= 7'b0011001;
	4'h5 : CATHODE <= 7'b0010010;
	4'h6 : CATHODE <= 7'b0000010;
	4'h7 : CATHODE <= 7'b1111000;
	4'h8 : CATHODE <= 7'b0000000;
	4'h9 : CATHODE <= 7'b0010000;
	4'ha : CATHODE <= 7'b0001000;
	4'hb : CATHODE <= 7'b0000011;
	4'hc : CATHODE <= 7'b0100111;
	4'hd : CATHODE <= 7'b0100001;
	4'he : CATHODE <= 7'b0000100;
	4'hf : CATHODE <= 7'b0001110;
	default : CATHODE <= 7'b1111111;
	endcase
	end

endmodule
```

FourReg

```
//taken from lab 5 - 4 bit register, but modified to take out the SET input
module FourReg(
	input [3:0] A,
	input CLK,
	input R,
	output reg [3:0] OUT
	);

	DFlipFlop f1(.A(A[0]), .CLK(CLK), .RESET(R), .Q(OUT[0]));
	DFlipFlop f2(.A(A[1]), .CLK(CLK), .RESET(R), .Q(OUT[1]));
	DFlipFlop f3(.A(A[2]), .CLK(CLK), .RESET(R), .Q(OUT[2]));
	DFlipFlop f4(.A(A[3]), .CLK(CLK), .RESET(R), .Q(OUT[3]));

endmodule
```

DFlipFlop

```
//taken from lab 5 - 4 bit register. Modified to have the SET input taken out
module DFlipFlop(
        input A,
        input CLK,
        input RESET,
        output reg Q
        );

        always_ff @ (posedge CLK) begin
        if (RESET == 1) Q = 0;
        else Q = A;
        end

endmodule
```

CheckDCDR

```
module CheckDCDR(
        input [3:0] IN, input BTN, input CLK, input R,
        input SW0, input SW1, input SW2, input SW3, input SW4,
        output reg [3:0] NUM_CORRECT = 0
        );
        reg [3:0] valid;

//determines if the switch that's inputted is valid for that current state
        always_ff @ (SW0 or SW1 or SW2 or SW3 or SW4)
        case({SW4, SW3, SW2, SW1, SW0})
        5'b11101 : valid <= 1;
        5'b01101 : valid <= 2;
        5'b10101 : valid <= 3;
        5'b10011 : valid <= 4;
        5'b01001 : valid <= 5;
        5'b11010 : valid <= 6;
        5'b10100 : valid <= 7;
        5'b10101 : valid <= 8;
        5'b11111 : valid <= 9;
        5'b10001 : valid <= 10;
        default : valid <= 11;
        endcase

//if the state its in has a correct input, the number correct is incremented
        always_ff @ (posedge BTN, posedge R)
        begin
        if (R == 1) NUM_CORRECT <= 0;
        else if (valid == IN) NUM_CORRECT <= NUM_CORRECT + 1;
        end

endmodule
```

Constraint File

```
set_property PACKAGE_PIN W5 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]

set_property PACKAGE_PIN U18 [get_ports BTN]
set_property IOSTANDARD LVCMOS33 [get_ports BTN]

set_property PACKAGE_PIN E19 [get_ports HEX]
set_property IOSTANDARD LVCMOS33 [get_ports HEX]

set_property PACKAGE_PIN U16 [get_ports DEC]
set_property IOSTANDARD LVCMOS33 [get_ports DEC]

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 R2 [get_ports R]
set_property IOSTANDARD LVCMOS33 [get_ports R]

set_property PACKAGE_PIN W7 [get_ports CATHODE[0]]
set_property IOSTANDARD LVCMOS33 [get_ports CATHODE[0]]

set_property PACKAGE_PIN W6 [get_ports CATHODE[1]]
set_property IOSTANDARD LVCMOS33 [get_ports CATHODE[1]]

set_property PACKAGE_PIN U8 [get_ports CATHODE[2]]
set_property IOSTANDARD LVCMOS33 [get_ports CATHODE[2]]
```

```
set_property PACKAGE_PIN V8 [get_ports CATHODE[3]]
set_property IOSTANDARD LVCMOS33 [get_ports CATHODE[3]]

set_property PACKAGE_PIN U5 [get_ports CATHODE[4]]
set_property IOSTANDARD LVCMOS33 [get_ports CATHODE[4]]

set_property PACKAGE_PIN V5 [get_ports CATHODE[5]]
set_property IOSTANDARD LVCMOS33 [get_ports CATHODE[5]]

set_property PACKAGE_PIN U7 [get_ports CATHODE[6]]
set_property IOSTANDARD LVCMOS33 [get_ports CATHODE[6]]

set_property PACKAGE_PIN U2 [get_ports ANODE[0]]
set_property IOSTANDARD LVCMOS33 [get_ports ANODE[0]]

set_property PACKAGE_PIN U4 [get_ports ANODE[1]]
set_property IOSTANDARD LVCMOS33 [get_ports ANODE[1]]

set_property PACKAGE_PIN V4 [get_ports ANODE[2]]
set_property IOSTANDARD LVCMOS33 [get_ports ANODE[2]]

set_property PACKAGE_PIN W4 [get_ports ANODE[3]]
set_property IOSTANDARD LVCMOS33 [get_ports ANODE[3]]

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets BTN_IBUF]
```