Lab 3: Hardware-Based True Random Number Generator and Time Control on FPGA

Jose Navarro

ECE5440

4207

**Introduction:**

The Hardware-Based True Random Number Generator and Time Control on FPGA is an extension of the Game Access Control and the Mental Binary Game. The Mental Binary Game is a simple two player game testing the players ability to convert hexadecimal numbers into binary numbers. In this game player one will think of a binary number and input the number to the FPGA using the FPGA switches. Without looking at player one’s switches, player two is to look at the FPGA’s seven segment decoder displaying player one’s number in hexadecimal. Player two is then to input the number, using the switches on the FPGA board, they believe will sum with player one’s number to equal 15. Player two’s goal is to mentally come up with the number that when added to player one’s number the total sum will equal 15 (or F when displayed on the FPGA). Player one’s goal is to come up with a number difficult enough so that player two is not able to come up with the right number to match the original output. If player two comes up with the correct number to match player one’s number player two gets a point. If player two does not come up with the correct number player one gets a point. Players change roles after each point is scored and play as long as they desire. The Game Access Control adds a level of security to the game. In order to play the game a user must input a four-digit password prior to being able to play the game. If the user inputs the correct password the game will be available to play, otherwise the FPGA will continue to accept password attempts. Once allowed to play the game player one and two will be able to input their numbers and play the game. The Game Access also adds a confirmation button. Player inputs will not be displayed until the confirmation button is pressed. The Hardware-Based True Random Number Generator and Time Control eliminates the need for two players. Player one will be substituted by a random number generator. This random number generator will generate the number that player two will need to match in order to score a point. The added time control adds a time limit to the game. Player two will be able to set a timer on the board that will count down the time left in seconds. The aim for player two is to score as many points as possible before time runs out. Once the timer runs out the random number generator will no longer generate numbers and player two will no longer be able to input numbers.

**System Architecture Design:**

Figure 1 below shows the top-level system architecture of the Hardware-Based True Random Number Generator and Time Control on the DE2-115 FPGA board.

A picture containing text

Description automatically generated

*Figure 1. Top-Level System Architecture for Hardware-Based True Random Number Generator and Time Control*

The figure above shows the system as a whole. The added modules to the system from the previous system described in the introduction (Game Access Control on FPGA) include the Random Number Generator, the Two Digit Timer, the One Second Timer and a modified Access Controller. The random number generator is a true random number generator. Using the press of the button from the user the random number generator outputs a digit between zero and fifteen that is dependent on the amount of clock cycles the internal modules have counted.

The Two-Digit Timer is comprised of two single digit timers. The two single-digit timers work together along with the One Second Timer to count down given a time input from the user. The single-digit timers count down from zero to nine but when “stacked” upon each other create the next multiple of ten digits. Therefore, if desired the single digit timers could create a three- or four-digit timer if wired appropriately. The One Second Timer is comprised of other smaller timers. Like the Two-Digit Timer these timers are “stacked” upon one another to create a single One Second Timer that outputs a positive pulse after a single second has passed.

Finally, the Access Controller was modified to “start” and “end” the game. The button used to confirm the input of each number for the access control is used to confirm the amount desired in the game and to start or re-start the game. This is seen in more detail in the figure below that shows the states of the Access Controller, specifically in the final four states.

A close up of text on a white background

Description automatically generated

*Figure 2. FSM of Modified Access Controller*

The Access Control Module has 10 states and is a single procedure FSM. Therefore, there is only a single “always” block in charge of outputs and directing the FSM to the next state. The first state, also the “First Number” state outputs a zero to both player input pass variables and the green LED (indicating access). Regardless of whether the user inputs the correct or incorrect password, so long as the user presses the access button the FSM will eventually reach the Flag Check state. If the password is correct access will be granted, indicated by a one output to green, and player pass variables will pass the input of the player buttons. If the password is incorrect the FSM will move to the First Number state and allow the user another try to input the password. The four final states of the Access Controller control where the player is in the game.

A player is first to input the password, to be approved by the Access Controller. Afterwards the player may request a number from the Random Number Generator and enter their number and confirm their input with the Load Register. Finally, the two numbers are summed together and decoded using a seven segment display as shown.

Figure 3 below shows the Finite State Machine (FSM) design of the Button Shaper module.

A picture containing text

Description automatically generated

*Figure 3. FSM of Button Shaper Module*

The Button Shaper is a two procedure FSM. Meaning that there are two “always” blocks running simultaneously. One is tasked with outputting values at the correct times and the other is tasked with changing the state the FSM is in. As shown in Figure 2 the Button Shaper has three states. The first state, also known as the initial state, outputs a value of zero and continues to do so, so long as the button is not pressed (an input of 1). The FSM will move onto the next state when button is pressed (an input of 0). The state that it will move on to is known as the pulse state. This will output a value of one for a short period of time and move onto the next state. The next state is known as the wait state. The output at this state will be a zero and the FSM will stay at this state so long the button is pressed (an input of 0) and will move on back to the initial state when the button is let go (an input of 1). The expected wave forms from this FSM according the previous description are shown in Figure 4 below.

A screenshot of a cell phone

Description automatically generated

*Figure 4. Expected Wave Forms for Button Shaper FSM*

**Simulation Results:**

Figure 5 below shows the simulation of the Random Number Generator Module. As shown in the figure the module outputs a random number when the button is pressed for the duration that it is pressed. This output will be given to the Seven Segment Display module which will output the random number generated.

**A screenshot of a computer

Description automatically generated**

*Figure 5. Simulation Results of Random Number Generator Module*

Figure 6 on the next page shows the simulation results of the Two-Digit Timer module. The Digit Timer counted down from fifteen to zero. However, only fifteen to six is shown because of the amount of viewing space allowed by Multisim.

*A screenshot of a computer

Description automatically generated*

*Figure 6. Simulation Results of Digit Timer Counting Down From 15 to 6*

**FPGA Board Testing Results:**

Figure 7 below shows the FPGA when initially turned on with the Hardware-Based True Random Number Generator and Time Control modules. As shown the board does not display any numbers and the players can’t input their number into the board, this is indicated by the two red LED’s lit up.

A lit up city at night

Description automatically generated

*Figure 7. Initial set up of FPGA board when first turned on*

Figure 8 shows the FPGA board after the access has been granted by inputting the correct pass code (4207), indicated by the green LED lighting up and before the player has input any amount of time on the board. As shown despite having flipped switches the displays still show a value of zero.

**A lit up city at night

Description automatically generated**

*Figure 8. FPGA board after access has been granted and player has not confirmed numbers*

Figure 9 on the following page shows the board when the player has input their time value but has not started the game. As shown the player has selected to play for 99 seconds.

A lit up city at night

Description automatically generated

*Figure 9. FPGA board after access has been granted and player has confirmed amount of time desired to play*

Figures 10 and 11 show the board when the player is in the middle of the game. As shown by Figure 10 the player does not match the number, this is indicated by the red LED remaining on. Figure 11 shows when the player does match the random number generated, this is indicated by the second green LED lighting up while the red LED turns off.

A lit up city at night

Description automatically generated

*Figure 10. Player Matches the Random Number Generated Incorrectly While the Clock Counts Down*

A lit up city at night

Description automatically generated

*Figure 11. Player Matches the Random Number Generated Correctly While the Clock Counts Down*

**Video Demo:**

<https://drive.google.com/file/d/17cCVQyT1oIybF6aXO5R3VcoGtWLPiDqC/view?usp=sharing>

**Conclusion:**

The Hardware-Based True Random Number Generator and Time Control allows the user to play the Mental Binary Math Game on their own with randomly generated numbers and play against a clock. Along with the Game Access Control allowing the user/owner of the FPGA to have a simple form of security to ensure not just anyone can play the Mental Binary Math Game. Using the FPGA board and modules written, shown visually in Figure 1, and code in the Appendix below, the lab is fully completed.

**Appendix:**

Lab3\_NAVARRO\_J.v:

module Lab3\_NAVARRO\_J(clk, rst, player1Button, accessControlSwitch, accessButton, player2Button, player2Input,

tensDigit, onesDigit, greenLED1, redLED1, greenLED2, redLED2, randomNumber1, sum, player2InputDisplay,

tensDigitDisplay, onesDigitDisplay);

input clk, rst, player1Button, accessButton, player2Button;

input [3:0] accessControlSwitch, player2Input, tensDigit, onesDigit;

output greenLED1, redLED1, greenLED2, redLED2;

output [6:0] randomNumber1, sum, player2InputDisplay, tensDigitDisplay, onesDigitDisplay;

wire buttonOutputAccess, buttonOutputPlayer2, loadPlayer1Pass, loadPlayer2Pass, noBorrowDown;

wire enable, reconfigButton, pulseOut;

wire [3:0] randomNum, onesDigitWire, tensDigitWire, player2InputWire, sumCalculated;

//buttonShaper(clk, rst, buttonInput, buttonOutput);

buttonShaper buttonShaperAccess(clk, rst, accessButton, buttonOutputAccess);

buttonShaper buttonSahperPlayer2(clk, rst, player2Button, buttonOutputPlayer2);

//accessController(clk, rst, loadPlayer1, loadPlayer2, passWord, passWordEnter, loadPlayer1Pass, loadPlayer2Pass,

//green, red, noBorrowDown, reconfigButton, enable);

accessController accessController1(clk, rst, player1Button, buttonOutputPlayer2, accessControlSwitch, buttonOutputAccess,

loadPlayer1Pass, loadPlayer2Pass, greenLED2, redLED2, noBorrowDown, reconfigButton, enable);

//loadRegister(clk, rst, loadPlayer, playerInput, playerOutput);

loadRegister loadRegisterPlayer2(clk, rst, loadPlayer2Pass, player2Input, player2InputWire);

//oneSecondTimer(clk, rst, enable, pulseOut);

oneSecondTimer oneSecondTimer1(clk, rst, enable, pulseOut);

//trueRNG(clk, rst, buttonIn, randomNum);

trueRNG randomNumberGenerator(clk, rst, loadPlayer1Pass, randomNum);

//twoDigitTimer(clk, rst, oneSecTime, timeInput1Button, timeInput1, timeInput10, timeToDisplay1,

//timeToDisplay10, noBorrowDown1);

twoDigitTimer twoDigitTimer1(clk, rst, pulseOut, reconfigButton, onesDigit, tensDigit, onesDigitWire,

tensDigitWire, noBorrowDown);

//adder(num1, num2, sum);

adder adderSumTotal(randomNum, player2InputWire, sumCalculated);

//ledToggle(sum, red, green);

ledToggle ledToggle1(sumCalculated, redLED1, greenLED1);

//sevenSegDecoder(numIn, numOut);

sevenSegDecoder randomNum1(randomNum, randomNumber1);

sevenSegDecoder sum1(sumCalculated, sum);

sevenSegDecoder player2(player2InputWire, player2InputDisplay);

sevenSegDecoder onesDigit1(onesDigitWire, onesDigitDisplay);

sevenSegDecoder tensDigit1(tensDigitWire, tensDigitDisplay);

endmodule

accessController.v:

module accessController(clk, rst, loadPlayer1, loadPlayer2, passWord, passWordEnter, loadPlayer1Pass, loadPlayer2Pass, green, red, noBorrowDown, reconfigButton, enable);

input clk, rst, loadPlayer1, loadPlayer2, passWordEnter, noBorrowDown;

input [3:0] passWord;

output loadPlayer1Pass, loadPlayer2Pass, green, red, reconfigButton, enable;

reg loadPlayer1Pass, loadPlayer2Pass, green, red, reconfigButton, enable;

reg passWordFlag;

//reconfigButton might just be passWordEnter

reg [3:0] state;

parameter state0 = 4'b0000, state1 = 4'b0001;

parameter state2 = 4'b0010, state3 = 4'b0011;

parameter state4 = 4'b0100, state5 = 4'b0101;

parameter state6 = 4'b0110, state7 = 4'b0111;

parameter state8 = 4'b1000, state9 = 4'b1001;

always @ (posedge clk)

begin

//passWordFlag = 1'b1;

if(rst == 0)

begin

state <= state0;

passWordFlag = 1'b1;

end

else

begin

case(state)

state0:

begin

loadPlayer1Pass <= 1'b1; //might need to change to arrows

loadPlayer2Pass <= 1'b0;

green <= 1'b0;

red <= 1'b1;

passWordFlag = 1'b1;

if(passWordEnter == 1'b1)

begin

if(passWord == 4'b0100)

begin

//good

end

else

begin

//not good forever, update flag

passWordFlag = 1'b0;

end

//go to next state

state <= state1;

end

else

begin

//stay in this state

state <= state0;

end

end

state1:

begin

loadPlayer1Pass <= 1'b1; //might need to change to arrows

loadPlayer2Pass <= 1'b0;

green <= 1'b0;

red <= 1'b1;

if(passWordEnter == 1'b1)

begin

if(passWord == 4'b0010)

begin

//good

end

else

begin

//not good forever

passWordFlag = 1'b0;

end

state <= state2;

end

else

begin

//stay in this state

state <= state1;

end

end

state2:

begin

loadPlayer1Pass <= 1'b1; //might need to change to arrows

loadPlayer2Pass <= 1'b0;

green <= 1'b0;

red <= 1'b1;

if(passWordEnter == 1'b1)

begin

if(passWord == 4'b0000)

begin

//good

end

else

begin

//not good forever

passWordFlag = 1'b0;

end

state <= state3;

end

else

begin

//stay in this state

state <= state2;

end

end

state3:

begin

loadPlayer1Pass <= 1'b1; //might need to change to arrows

loadPlayer2Pass <= 1'b0;

green <= 1'b0;

red <= 1'b1;

if(passWordEnter == 1'b1)

begin

if(passWord == 4'b0111)

begin

//good

end

else

begin

//not good forever

passWordFlag = 1'b0;

end

state <= state4;

end

else

begin

//stay in this state

state <= state3;

end

end

state4:

begin

if(passWordFlag == 1'b0)

begin

state <= state0;

end

else

begin

state <= state5;

end

end

state5:

begin

red <= 1'b0;

green <= 1'b1;

state <= state6;

end

state6:

begin

if(passWordEnter == 1)

begin

reconfigButton <= 1;

state <= state7;

end

else

begin

state <= state6;

end

end

state7:

begin

reconfigButton <= 0;

if(passWordEnter == 1)

begin

state <= state8;

end

else

begin

state <= state7;

end

end

state8:

begin

enable <= 1;

loadPlayer1Pass <= loadPlayer1; //might need to change to arrows

loadPlayer2Pass <= loadPlayer2;

if(noBorrowDown == 1)

begin

state <= state9;

end

else

begin

state <= state8;

end

end

state9:

begin

enable <= 0;

//reconfigButton <= 0;

loadPlayer1Pass <= 1'b1; //might need to change to arrows

loadPlayer2Pass <= 1'b0;

if(passWordEnter == 1)

begin

reconfigButton <= 1;

state <= state7;

end

else

begin

state <= state9;

end

end

default:

begin

state <= state0;

passWordFlag = 1'b1;

end

endcase

end

end

endmodule

adder.v:

module adder(num1, num2, sum);

input [3:0] num1, num2;

output [3:0] sum;

reg [3:0] sum;

always @ (num1, num2)

begin

sum = num1 + num2;

end

endmodule

buttonShaper.v:

module buttonShaper(clk, rst, buttonInput, buttonOutput);

input clk, rst, buttonInput;

output buttonOutput;

reg buttonOutput;

reg [1:0] state, nextState;

parameter initialState = 2'b00;

parameter pulseState = 2'b01;

parameter waitState = 2'b10;

parameter state3 = 2'b11;

always @ (posedge clk)

begin

if(rst == 0) //might be a one but not sure

begin

state <= initialState;

end

else

begin

state <= nextState;

end

end

always @ (state, buttonInput)

begin

case(state)

initialState:

begin

buttonOutput = 1'b0;

if(buttonInput == 1)

begin

nextState <= initialState;

end

else

begin

nextState <= pulseState;

end

end

pulseState:

begin

buttonOutput = 1'b1;

nextState <= waitState;

end

waitState:

begin

buttonOutput = 1'b0;

if(buttonInput == 1)

begin

nextState <= initialState;

end

else

begin

nextState <= waitState;

end

end

default:

begin

buttonOutput = 1'b0;

nextState = initialState;

end

endcase

end

endmodule

ledToggle.v:

module ledToggle(sum, red, green);

input [3:0] sum;

output red, green;

reg red, green;

always @ (sum)

begin

if (sum == 15)

begin

red = 0;

green = 1;

end

else

begin

red = 1;

green = 0;

end

end

endmodule

loadRegister.v:

module loadRegister(clk, rst, loadPlayer, playerInput, playerOutput);

input clk, rst, loadPlayer;

input [3:0] playerInput;

output [3:0] playerOutput;

reg [3:0] playerOutput;

always @ (posedge clk)

begin

if(rst == 0)

begin

playerOutput <= 0;

end

else

begin

if(loadPlayer == 1)

begin

playerOutput <= playerInput;

end

end

end

endmodule

oneSecondTimer.v:

module oneSecondTimer(clk, rst, enable, pulseOut);

input clk, rst, enable;

output pulseOut;

wire ms100pulse;

Timer100MS Timer100MS1(clk, rst, enable, ms100pulse);

countTo10 countTo10\_1(clk, rst, enable, ms100pulse, pulseOut);

endmodule

Timer100MS.v:

module Timer100MS(clk, rst, enable, pulseOut);

input clk, rst, enable;

output pulseOut;

wire msPulse;

oneMSTimer oneMSTimer\_1(clk, rst, enable, msPulse);

countTo100 countTo100\_1(clk, rst, enable, msPulse, pulseOut);

endmodule

countTo100.v:

module countTo100(clk, rst, enable, pulseIn, pulseOut);

input clk, rst, enable, pulseIn;

output reg pulseOut;

reg [6:0] count;

always @ (posedge clk)

begin

if(enable == 0)

begin

end

else

begin

if(rst == 0)

begin

count <= 0;

pulseOut <= 0;

end

else

begin

if(pulseIn == 1) // && ((count == 3) | (count > 3))) //100

begin

if((count == 99) | (count> 99))

begin

pulseOut = 1;

count <= 0;

end

else

begin

pulseOut <= 0;

count <= count + 1;

end

end

else

begin

pulseOut <= 0;

end

end

end

end

endmodule

oneMSTimer.v:

module oneMSTimer(clk, rst, enable, pulseOut);

input clk, rst, enable;

output reg pulseOut;

reg [15:0] count;

always @ (posedge clk)

begin

if(enable == 0)

begin

end

else

begin

if(rst == 0)

begin

count <= 0;

pulseOut <= 0;

end

else

begin

if((count == 49999) | (count > 49999)) //50000

begin

count <= 0;

pulseOut <= 1;

end

else

begin

count <= count + 1;

pulseOut <= 0;

end

end

end

end

endmodule

countTo10.v:

module countTo10(clk, rst, enable, pulseIn, pulseOut);

input clk, rst, enable, pulseIn;

output reg pulseOut;

reg [3:0] count;

always @ (posedge clk)

begin

if(enable == 0)

begin

end

else

begin

if(rst == 0)

begin

count <= 0;

pulseOut <= 0;

end

else

begin

if( pulseIn == 1 && ((count == 9) | (count > 9))) //10

begin

count <= 0;

pulseOut <= 1;

end

else if (pulseIn == 1)

begin

count <= count + 1;

pulseOut <= 0;

end

else

begin

pulseOut <= 0;

end

end

end

end

endmodule

sevenSegDecoder.v:

module sevenSegDecoder(numIn, numOut);

input [3:0] numIn;

output [6:0] numOut;

reg [6:0] numOut;

always @ (numIn)

begin

case(numIn)

4'b0000: begin numOut = 7'b1000000; end

4'b0001: begin numOut = 7'b1111001; end

4'b0010: begin numOut = 7'b0100100; end

4'b0011: begin numOut = 7'b0110000; end

4'b0100: begin numOut = 7'b0011001; end

4'b0101: begin numOut = 7'b0010010; end

4'b0110: begin numOut = 7'b0000010; end

4'b0111: begin numOut = 7'b1111000; end

4'b1000: begin numOut = 7'b0000000; end

4'b1001: begin numOut = 7'b0011000; end

4'b1010: begin numOut = 7'b0001000; end

4'b1011: begin numOut = 7'b0000011; end

4'b1100: begin numOut = 7'b1000110; end

4'b1101: begin numOut = 7'b0100001; end

4'b1110: begin numOut = 7'b0000110; end

4'b1111: begin numOut = 7'b0001110; end

default: begin numOut = 7'b1111111; end

endcase

end

endmodule

trueRNG.v:

module trueRNG(clk, rst, buttonIn, randomNum);

input clk, rst, buttonIn;

output [3:0] randomNum;

counter count1(clk, rst, buttonIn, randomNum);

endmodule

counter.v:

module counter(clk, rst, buttonIn, countOutput);

input clk, rst, buttonIn;

output reg [3:0] countOutput;

reg [3:0] count;

assign buttonCount = ~buttonIn;

always @ (posedge clk)

begin

if(rst == 0)

begin

count <= 0;

end

else if((count == 15) | (count > 15))

begin

count <= 0;

end

else

begin

if( buttonCount == 1)

begin

count <= count + 1;

end

else

begin

countOutput <= count;

end

end

end

endmodule

twoDigitTimer.v:

module twoDigitTimer(clk, rst, oneSecTime, timeInput1Button, timeInput1, timeInput10, timeToDisplay1, timeToDisplay10, noBorrowDown1);

input clk, rst, oneSecTime, timeInput1Button;

input [3:0] timeInput1, timeInput10;

output [3:0] timeToDisplay1, timeToDisplay10;

output noBorrowDown1;

wire borrowUp1, noBorrowDown10;

wire dontCare;

//digitTimer(clk, rst, confirmButtonNum, timeAmountInput, borrowUp, noBorrowUp, borrowDown, noBorrowDown, timeToDisplay);

digitTimer digitTimer1(clk, rst, timeInput1Button, timeInput1, borrowUp1, noBorrowDown10, oneSecTime, noBorrowDown1, timeToDisplay1);

digitTimer digitTimer10(clk, rst, timeInput1Button, timeInput10, dontCare, 1, borrowUp1, noBorrowDown10, timeToDisplay10);

endmodule

digitTimer.v:

module digitTimer(clk, rst, confirmButtonNum, timeAmountInput, borrowUp, noBorrowUp,

borrowDown, noBorrowDown, timeToDisplay);

input clk, rst, confirmButtonNum, noBorrowUp, borrowDown;

output reg borrowUp, noBorrowDown;

input [3:0] timeAmountInput;

reg [3:0] timeAmount;

output reg [3:0] timeToDisplay;

always @ (posedge clk)

begin

if(rst == 0)

begin

//reset values

timeToDisplay <= 0;

// borrowUp <= 0;

// noBorrowDown <= 0;

timeAmount <= 0;

end

else

begin

if(confirmButtonNum == 1)

begin

if(timeAmountInput > 9)

begin

timeAmount <= 9;

end

else

begin

timeAmount <= timeAmountInput;

end

timeToDisplay <= timeAmount;

if(timeAmount == 0)

begin

noBorrowDown <= 1;

end

else

begin

noBorrowDown <= 0;

end

end

else

begin

timeToDisplay <= timeAmount;

if( borrowDown == 1)

begin

if(timeAmount > 0)

begin

//check if game is over

if(timeAmount == 1 && noBorrowUp == 1)

begin

borrowUp <= 1;

timeToDisplay <= 0;

timeAmount <= 0;

noBorrowDown <= 1;

//flag that game is over

end

else

begin

timeAmount <= timeAmount - 1;

timeToDisplay <= timeAmount;

noBorrowDown <= 0;

borrowUp <= 0; // this may need to change

end

end

else if(timeAmount == 0)

begin

timeToDisplay <= 0; //might not be necessary

if(noBorrowUp == 1)

begin

//there is nothing to borrow, higher num is 0

end

else if(noBorrowUp == 0)

begin

//there is something to borrow

timeAmount <= 9;

timeToDisplay <= timeAmount;

end

borrowUp <= 1;

//noBorrowDown <= 1;

end

end

else

begin

if(timeAmount==0)

begin

if(noBorrowUp==0)

begin

noBorrowDown<=0;

end

else

begin

borrowUp<=0;

noBorrowDown<=1;

end

end

else

begin

noBorrowDown<=0;

borrowUp<=0;

end

end

end

end

end

endmodule