Koç University

College of Engineering

Department of Electrical & Electronics Engineering

Project Report
Design and Implementation of a "Read My Mind" Game

Due date: 17.05.2019

Prepared by: Gül Sena Altıntaş, Kerem Kaya

1. Introduction:

1.1 Aim of the game:

Find the random number generated by the system by making guesses while updating the guess in every trial according to high/low instructions given

1.2 Equipment & Software used:

• IBM Compatible PC with Windows 7 operating system,
• Xilinx ISE v1.47 & Prometheus software packages,
• Prometheus FPGA (Xilinx Spartan 3A) board,
• USB cable for programming.

2. Methodology:

2.1 Finite State Machine:

A finite state machine is a computation model that can be implemented with hardware or software and can be used to simulate sequential logic. A system where particular inputs cause particular changes in state can be represented using finite state machines.[1]

2.2 Design:

Steps of the game:

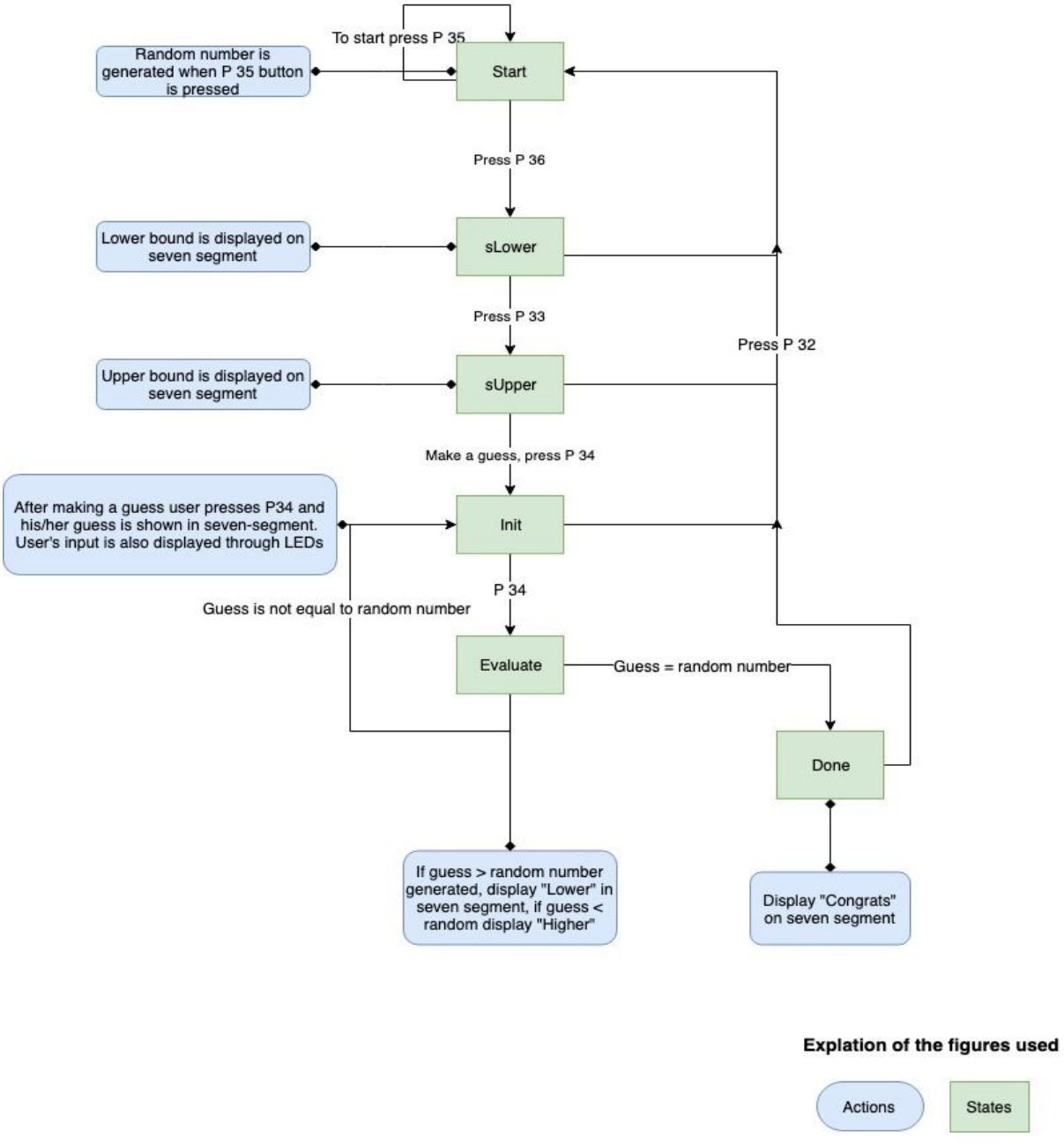
1) User presses P35 button to start the game
2) Game chooses a random 8 bit binary number (from 0 to 255),
3) User presses P36 button
4) Game displays the lower bound on the right-hand side of seven-segment display
5) User presses P33 button
6) Game displays the upper bound on the right-hand side of seven-segment display
7) User enters his/her guess with the help of switches.
8) User presses P34 button after making his/her guess. The guess of the user is shown on the right-hand side of seven-segment display. Then user presses P34 button in order to see the feedback from the game.

[1] https://whatis.techtarget.com/definition/finite-state-machine

9) If the guess is incorrect, it gives feedback to the user to update the value as HIGHER or LOWER where HIGHER instructs the user to take a higher guess and LOWER otherwise. (Higher and Lower instructions are displayed starting from the left most anode of the seven-segment display)
10) If the guess is correct, it displays "CONGRATS" on the seven-segment display. User presses P32 button to restart the game[2]

***Button selection is done arbitrarily where P35, P36, P33, P34 and P32 are chosen.

Schematic representation of the states and steps of the game



[2] User can press P32 button at any time to restart the game

Implementation:

The game is implemented as a finite state machine, with the following states:

Start

Start state is the initiation state. When in Start state and the sButton (P35) is pressed the system generates a random 8 bit binary number.

SLower

SLower state is executed if the user presses P36 button after the Start state and displays the lower bound -0- on the seven-segment display.

SUpper

SUpper state is executed if the user presses P33 button after the SLower state and displays the upper bound -255- on the seven segment display.

Init

Init state takes the guess from the user, when the user is ready to submit his/her guess, he/she presses gButton -P34, then the system proceeds to Evaluate state.

Evaluate

Evaluate state evaluates the guess of the user, if correct it shows Congrats and goes to Done state, else gives feedback accordingly and proceeds to Init.

Done

If the user presses Restart button FSM returns back to Start state. User can choose to restart the game at each state.

***We divided the overall process into states as much as we can since it becomes easier to implement and control the system.

2.3 Procedure:

The code regulating the state transitions is shown below. Res is the 8 bit binary number whose value is then transferred to to_BCD function to be displayed on the seven segment, Leds is the 8 bit STD_LOGIC_VECTOR representing the number to be shown on the Leds.

```
--- state transition
--- states are defined with the help of "when"
process(CLK_DIV)
   begin
         if rising_edge(MCLK) then
```

```
case state is
	when start =>
		Res <= "00000000";
		Leds <= Res;
		if (sButton = '1') then
			random_num <= r1;
		elsif (lower_button = '1') then
			state <= sLower;
		end if;

	when sLower =>
		if (lower_button = '1') then
			Res <= lower_bound;
			Leds <= Res;
		elsif (upper_button = '1') then
			state <= sUpper;
		elsif (rButton = '1') then
			state <= start;
		else
			state <= sLower;
		end if;

	when sUpper =>
		if (upper_button = '1') then
			Res <= upper_bound;
			Leds <= Res;
		end if;
		if (gButton = '1') then
			state <= init;
		elsif (rButton = '1') then
			state <= start;
		else
			state <= sUpper;
		end if;

	when init =>
		if (rButton = '1') then
			state <= start;
		elsif (gButton = '1') then
			state <= evaluate;
		else
			state <= init;
		end if;

	when evaluate =>
		if (Guess = random_num) then
			Leds <= Guess;
			state <= done;
		elsif (Guess < random_num) then
			Res <= Guess;
```

```
                                        Leds <= Res;
                                        state <= init;
                                elsif (Guess > random_num) then
                                        Res <= Guess;
                                        Leds <= Res;
                                        state <= init;
                                end if;
                                if (rButton = '1') then
                                        state <= start;
                                end if;

                        when done =>
                                Leds <= Guess;
                                if (rButton = '1') then
                                        state <= start;
                                end if;
                        when others =>
                                state <= start;
                end case;
        end if;
end process;
```

Random number generation by LFSR:

- The number produced by the system is a pseudo random number. It is generated by using Linear Feedback Shift Register. In short, generating linear feedback random generator requires a right-shift operation and XOR operation.

- Throughout the report we used the term “random” number for “pseudo random” number generated by Linear Feedback Shift Register.

- We used a separate component to generate random number in our project.Using separate component enabled to continue generating numbers without interruption when the user presses restart button to return back to “Start” state.

- LFSR is a shift register whose input bit is a linear function of its previous state. The most commonly used linear function of single bits is XOR. Thus, an LFSR is most often a shift register whose input bit is driven by the exclusive-or (XOR) of some bits of the overall shift register value. [3]

- LFSR code is shown below.

```
        begin
```

[3] https://www.researchgate.net/publication/236109080_FPGA_Implementation_of_8_16_and_32_Bit_LFSR_with_Maximum_Length_Feedback_Polynomial_using_VHDL

```
   linear_feedback <= not(count(15) xor count(11));
   process (MCLK, rButton)
   begin
         if (rButton = '1') then
               count <= (others=>'0');
         elsif (rising_edge(MCLK)) then
               if (sButton = '1') then
               count <= ( count(14) & count(13) & count(12)&count(11) & count(10) & count(9)
& count(8) & count(7)&count(6) & count(5) & count(4) & count(3) & count(2) & count(1) &
count(0) & linear_feedback);
               end if;
         end if;
   end process;
   c_out <= count;

 r1signal(0)<=c_out(0) xor c_out(15);
 r1signal(1)<=c_out(1) xor c_out(14);
 r1signal(2)<=c_out(2) xor c_out(13);
 r1signal(3)<=c_out(3) xor c_out(12);
 r1signal(4)<=c_out(4) xor c_out(11);
 r1signal(5)<=c_out(5) xor c_out(10);
 r1signal(6)<=c_out(6) xor c_out(9);
 r1signal(7)<=c_out(7) xor c_out(8);

 -- process (r1signal)
 r1(0) <= r1signal(0);
 r1(1) <= r1signal(1);
 r1(2) <= r1signal(2);
 r1(3) <= r1signal(3);
 r1(4) <= r1signal(4);
 r1(5) <= r1signal(5);
 r1(6) <= r1signal(6);
 r1(7) <= r1signal(7);
```

To BCD function:

Binary coded decimal (BCD) provides a simple conversion between a binary number and the decimal number. We used “to_BCD” function in order to convert binary number entered by the user with the help of switches into decimal number. So that the system can compare the input entered with the random number generated and give a feedback accordingly.

In order to execute to_BCD function we used the commonly known “Double Dabble” algorithm which iterates 8 times, increments digits that are greater than 4 by 3, and left-shifts the bits. Addition of 3 ensures that the value is carried out properly in base 10.

The code for the to_BCD function is included below.

```
        function to_bcd (bin : std_logic_vector (7 downto 0)) return std_logic_vector is
         variable i : integer:=0;
         variable bcd : std_logic_vector (11 downto 0) := (others => '0');
         variable bint : std_logic_vector (7 downto 0) := bin;

         begin
         for i in 0 to 7 loop -- repeat 8 times
                  bcd (11 downto 1) := bcd (10 downto 0); -- shifting the bits
                  bcd (0) := bint(7);
                  bint (7 downto 1) := bint (6 downto 0);
                  bint (0) := '0';

                  if(i < 7 and bcd(3 downto 0) > "0100") then --add 3 if BCD digit is greater than 4.
                  bcd(3 downto 0) := bcd(3 downto 0) + "0011";
                  end if;

                  if(i < 7 and bcd(7 downto 4) > "0100") then --add 3 if BCD digit is greater than 4.
                  bcd(7 downto 4) := bcd(7 downto 4) + "0011";
                  end if;

                  if(i < 7 and bcd(11 downto 8) > "0100") then  --add 3 if BCD digit is greater than
4.
                  bcd(11 downto 8) := bcd(11 downto 8) + "0011";
                  end if;

         end loop;
         return bcd;
   end to_bcd;
```

Pin Assignment

The Pins used are defined as below.

```
NET "MCLK" LOC = "P40";

NET "Guess<0>" LOC = "P15";
NET "Guess<1>" LOC = "P12";
NET "Guess<2>" LOC = "P5";
NET "Guess<3>" LOC = "P4";
NET "Guess<4>" LOC = "P94";
NET "Guess<5>" LOC = "P90";
NET "Guess<6>" LOC = "P88";
NET "Guess<7>" LOC = "P85";

NET "rButton" LOC = "P32";
NET "sButton" LOC = "P35";
NET "gButton" LOC = "P34";
NET "lower_button" LOC = "P36";
NET "upper_button" LOC = "P33";
```

```
NET "Leds<4>" LOC = "P93";
NET "Leds<5>" LOC = "P89";
NET "Leds<6>" LOC = "P86";

NET "Anodes<0>"  LOC = "P50";
NET "Anodes<1>"  LOC = "P49";
NET "Anodes<2>"  LOC = "P52";
NET "Anodes<3>"  LOC = "P56";
NET "Anodes<4>"  LOC = "P59";
NET "Anodes<5>"  LOC = "P57";
NET "Anodes<6>"  LOC = "P60";
NET "Anodes<7>"  LOC = "P61";

NET "SevenSegment<0>"  LOC = "P64";
NET "SevenSegment<1>"  LOC = "P98";
NET "SevenSegment<2>"  LOC = "P73";
NET "SevenSegment<3>"  LOC = "P72";
```

NET "Leds<0>" LOC = "P16"; NET "Leds<1>" LOC = "P13"; NET "Leds<2>" LOC = "P6"; NET "Leds<3>" LOC = "P3";	NET "SevenSegment<4>" LOC = "P65"; NET "SevenSegment<5>" LOC = "P62"; NET "SevenSegment<6>" LOC = "P71";

What are the similarities with the labs?

- Taking input from the switches
- Moving to next state according to button pressed
- Using LEDS to display input entered through switches

What are the differences we added?

- Creating random number in a specific range using linear feedback
- Evaluating the input entered by the user and the number created by the system
- Adding a restart button, returning FSM to initial state

3. Experimental Results:

We showed our code and bit file to Mr. Erzin and he approved that our game is working perfectly. He also told us that there is no need to attach pictures of FPGA board to our report.
In order to create a sense of suspicion we decided to show the lower/higher instructions after pressing the button, also in order not to confuse the player we decided to keep the bounds from 0 to 2^8-1 rather than using 2's complement system.

4. Discussion and Conclusion:

Throughout the project we experienced the design and implementation of a project from scratch, tested our ideas, modified them according to our capabilities and knowledge. After performing this project, we both had an idea about how to perform system design and test. At the end of the project, we both were capable of doing the below actions:

• Specify a simple digital system,
• Separate the system into one or more datapaths and controls,
• Design the datapaths for the system,
• Design the controls for the system,
• Integrate the datapaths and controls into an overall system,
• Devise a thorough test for the system,
• Implement the system using Xilinx Foundation Software, and
• Test and debug the system.

In overall, it was an educative experience for us. We reviewed the entire course material while testing and improving our project. Also, it was so useful for us to get familiarize using Xilinx.