Viktoria Biliouri
Undergraduate Student
Department of Electrical & Computer Engineering
University of Thessaly

7-SEGMENTS INDICATION DRIVER

PURPOSE

The purpose of this project is to create a four indications driver of the 7-segments LED which exists in Spartan 3 FPGA platform, for the circular display of the message :

012456789abcdEF

For the purpose of our facilitation the project is divided in 4 parts.

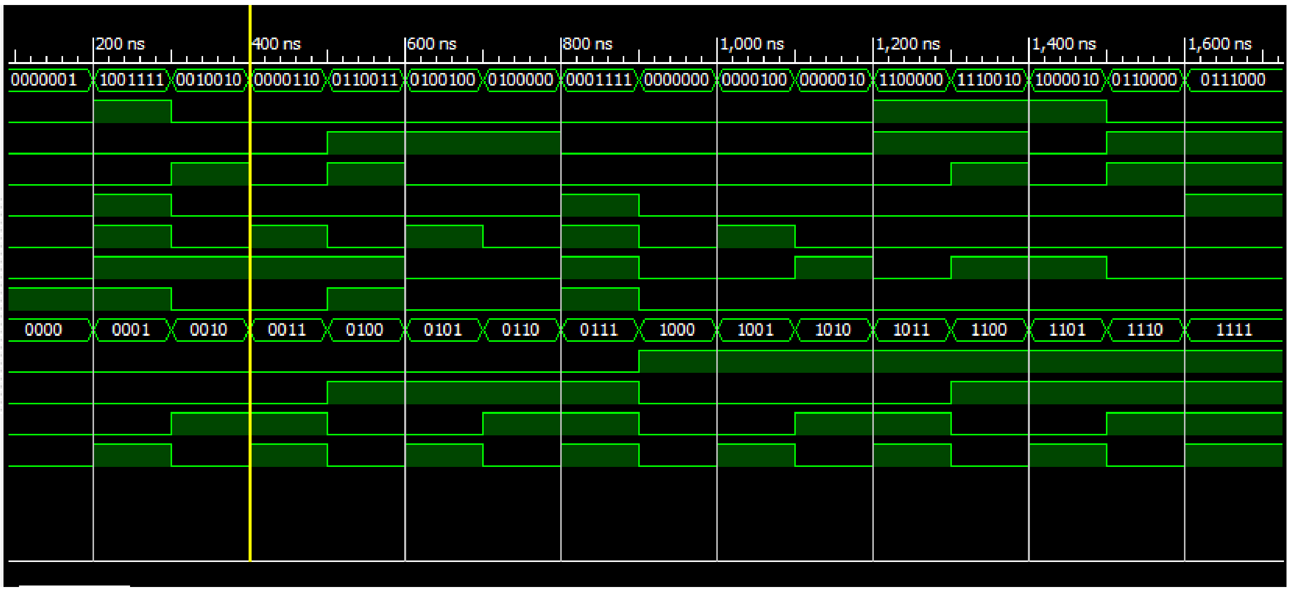
PART A

This part includes the creation of a LED decoder which indicates the segments of the LED that should be open so as to dispay each character. Every character can be expressed in a 4-bit binary number (LEDdecoder module). You are able to see the matching (character – binary value – Led display) in the table below.

CHARACTER	BINARY VALUE	LED Decoder
0	4'b0000	7'b0000001
1	4'b0001	7'b1001111
2	4'b0010	7'b0010010
3	4'b0011	7'b0000110
4	4'b0100	7'b1001100
5	4'b0101	7'b0100100
6	4'b0110	7'b0100000
7	4'b0111	7'b0001111
8	4'b1000	7'b0000000
9	4'b1001	7'b0001000
a	4'b1010	7'b0000010
b	4'b1011	7'b1100000
c	4'b1100	7'b1110010
d	4'b1101	7'b1000010
E	4'b1110	7'b0110000
F	4'b1111	7'b0111000

In order to verify whether the results of this module are correct, or not, I have created a simulation testbench which increases the binary value every 100 ns.

Simulation results:



<u>PART B</u>
In this part, the verilog code should display the first four digits of the message above ('0123'), on the screen of Spartan 3 platform. It should also connect the button (L14) with the reset signal.

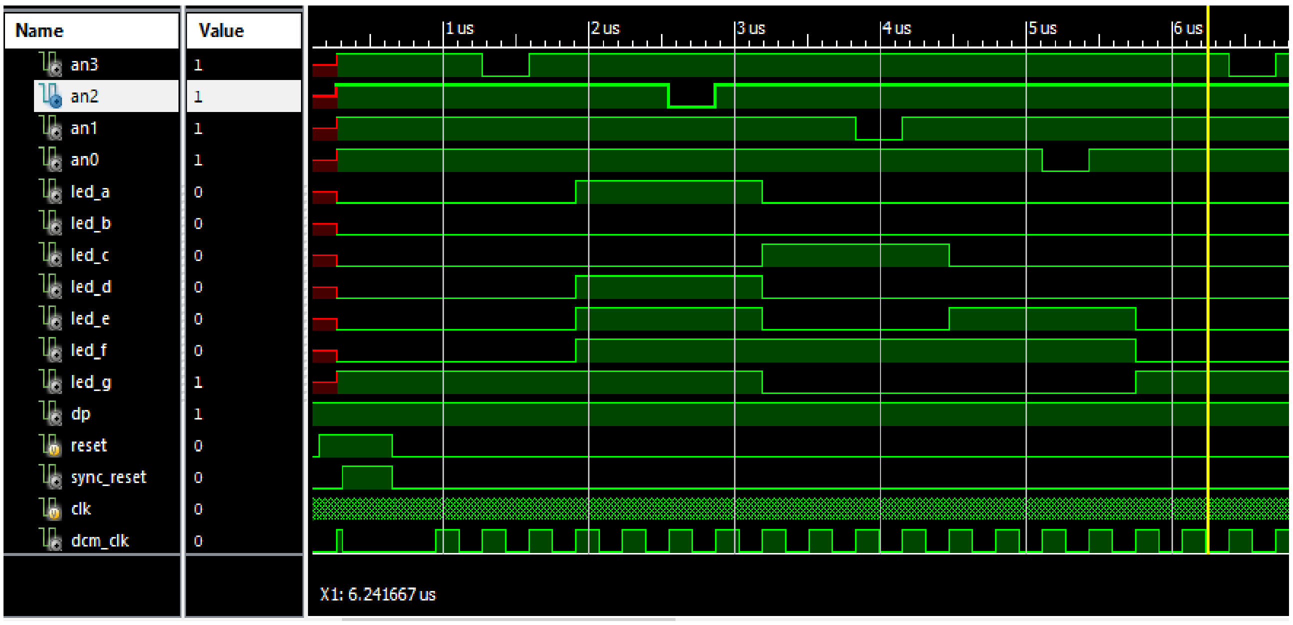
The clock period of Spartan 3 clock is 20ns. However in this project, fpga's clock is divided by the ISE DCM module, and the final clock period is 320ns, as it has increased 16 times.

One problem that occurred is that the FPGA buttons were bouncing some thousand times, so in order for the program to function without mistakes, I have created an antibounce circuit, which controls the number of times that the reset signal value changes (0->1). In particular, the reset signal should be passed from a 2 Flip-Flops circuit, so as to synchronize with the clock, and then, a counter should be created, in order to count how many times in a short period of time the new synchronized signal gets the value 1. When this counter reaches the value of 25000
(25000 x 320ns = 0.08sec = the estimated time that a button is pushed, binary = 110000110101000), it means that the bounces are finished and the final signal can take the value 1. Finally, it is necessary to keep this value as many cicles as the number of cicles that the initial signal is 1, so the counter is decreased. This final signal is used in the rest of the program (`sync_reset`).

The DigitLoader module is used to show which of the screen digits is about to light and which character should be displayed. The character should get its value 2 clock circles before the screen is open, so that it is certain that it has the appropriate value when the digit lights. The time between the digits appearance is controlled by a 4-bit counter, initialized with the value '1111' and decreasing in accordance with the table below.

Counter value	an3	an2	an1	an0
4'b1111	1	1	1	1
4'b1110	0	1	1	1
4'b1101	1	1	1	1
4'b1100	1	1	1	1
4'b1011	1	1	1	1
4'b1010	1	0	1	1
4'b1001	1	1	1	1
4'b1000	1	1	1	1
4'b0111	1	1	1	1
4'b0110	1	1	0	1
4'b0101	1	1	1	1
4'b0100	1	1	1	1
4'b0011	1	1	1	1
4'b0010	1	1	1	0
4'b0001	1	1	1	1
4'b0000	1	1	1	1

Part B Simulation Results:

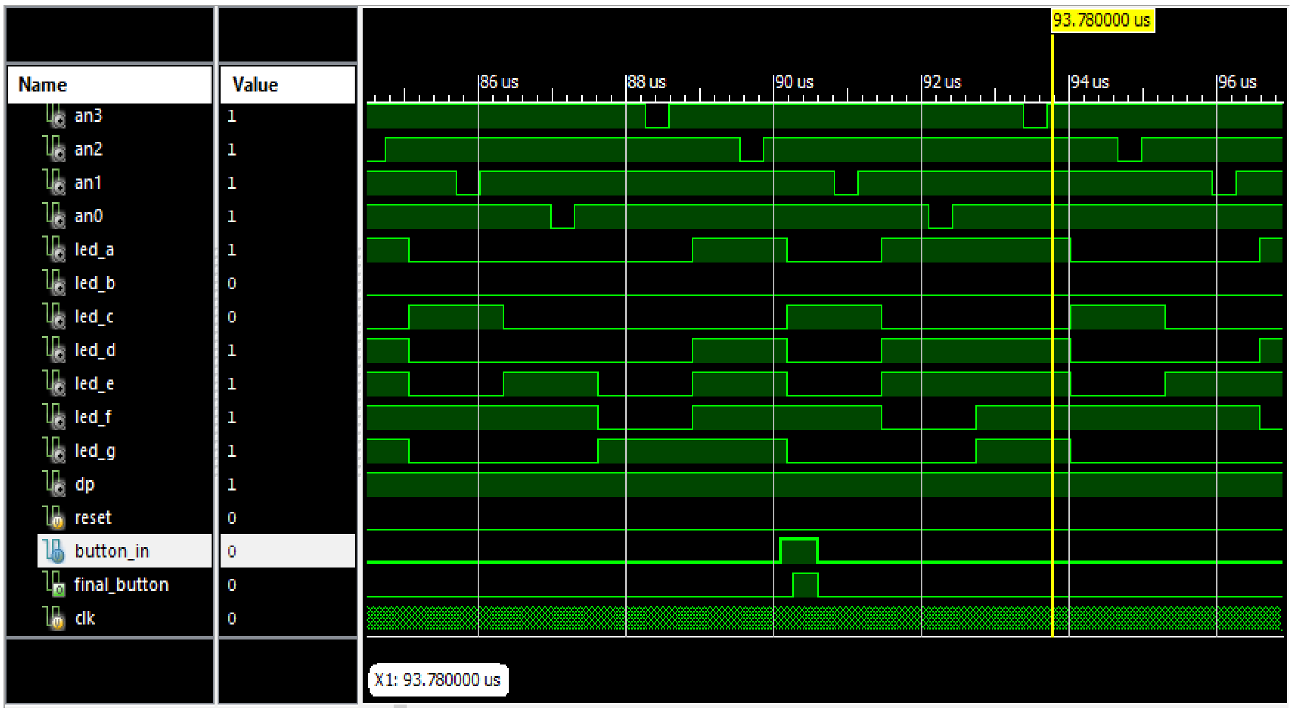


Name	Value
an3	1
an2	1
an1	1
an0	1
led_a	0
led_b	0
led_c	0
led_d	0
led_e	0
led_f	0
led_g	1
dp	1
reset	0
sync_reset	0
clk	0
dcm_clk	0

PART C

The purpose of part C, is to rotate the whole 16-characters message when a second button (L13) is pressed. In particular, every time we press the button we want to shift the message one position. In order for this to happen, I have created a memory where all the characters are stored, and a counter which counts how many times the button is pressed. This counter indicates the number of positions we should shift the message. Like the process of reset signal, the use of the same antibounce circuit (and synchronizer) is also necessary for the second button, so the final button that I am using in the rest of the process is `sync_button`. Every time someone pushes the button, the counter(`press_number`) is increased and when it reaches it's final value ('1111') it turns into it's initial value '0000'. One essential point that we should mention, is that the circuit should return to its initial status whenever the reset button is pressed.

Part C Simulation Results:



As we can see above, the led values change when button is pressed. The second button signal (final_button) is the synchronized and debounced button signal.

PART D

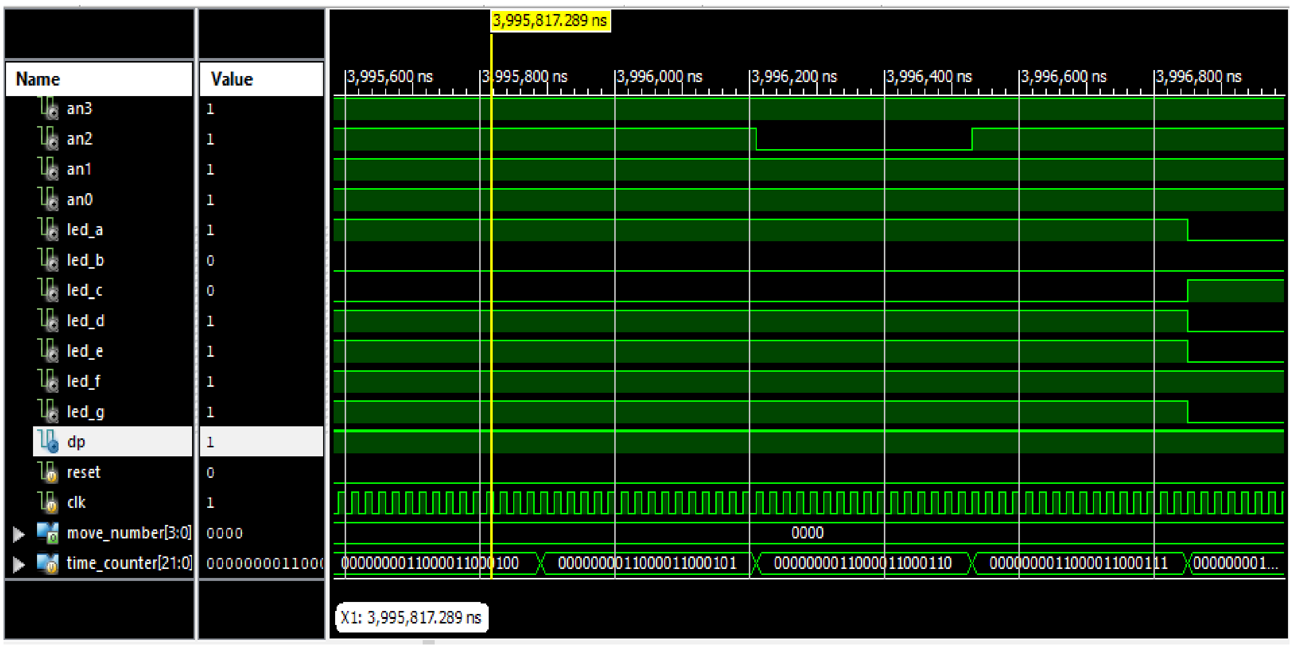
In part D, the message is rotated, as in part C, with the difference that, the process is not controlled by a button, but by a time counter . In particular, we want the message to be shifted every 1.3421 seconds, without the use of a button.

In order for this to happen, we need a 22-bit counter(time_counter), which is increased every time the clock gets the '1' value. When this counter gets the value 22'b1, the message is moved by one position. This is accomplished by the creation of a second 4-bit counter(move_number), which counts the times the message should be shifted, in accordance with the time counter. Every time the time counter gets its high value, it should return to its initial value ('22'b0') and the move number counter should increase. In addition to this, every time the move number counter gets the value '1111', it should return to its initial value '0000'.

The reset signal also exists in this part of the project, with exactly the same function as in part B and C, and whenever the reset button is pressed, the process returns to its initial status.

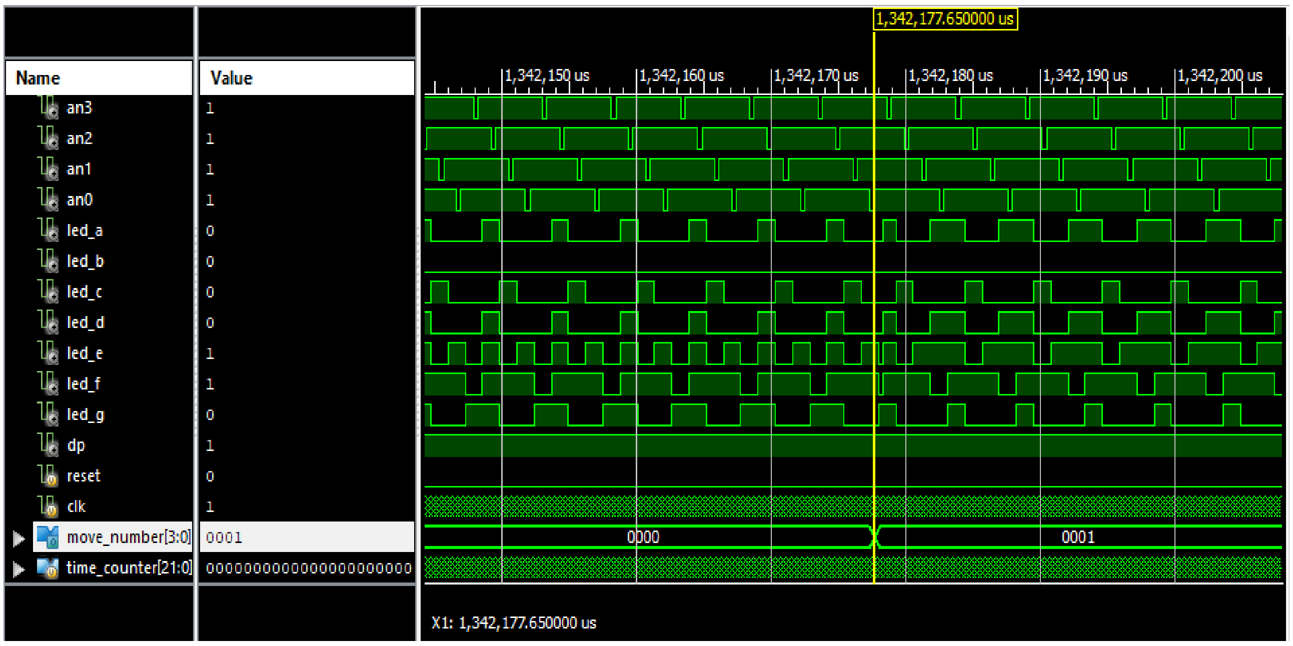
The rest of the modules of this part, function with the same way as in the previous parts.

Part D Simulation Results:



Name	Value
an3	1
an2	1
an1	1
an0	1
led_a	1
led_b	0
led_c	0
led_d	1
led_e	1
led_f	1
led_g	1
dp	1
reset	0
clk	1
move_number[3:0]	0000
time_counter[21:0]	000000001100(

As seen above, time counter increases every time the dcm clock is 1. The position of the characters is defined according to the move number counter, as below:



Name	Value
an3	1
an2	1
an1	1
an0	1
led_a	0
led_b	0
led_c	0
led_d	0
led_e	1
led_f	1
led_g	0
dp	1
reset	0
clk	1
move_number[3:0]	0001
time_counter[21:0]	0000000000000000000000