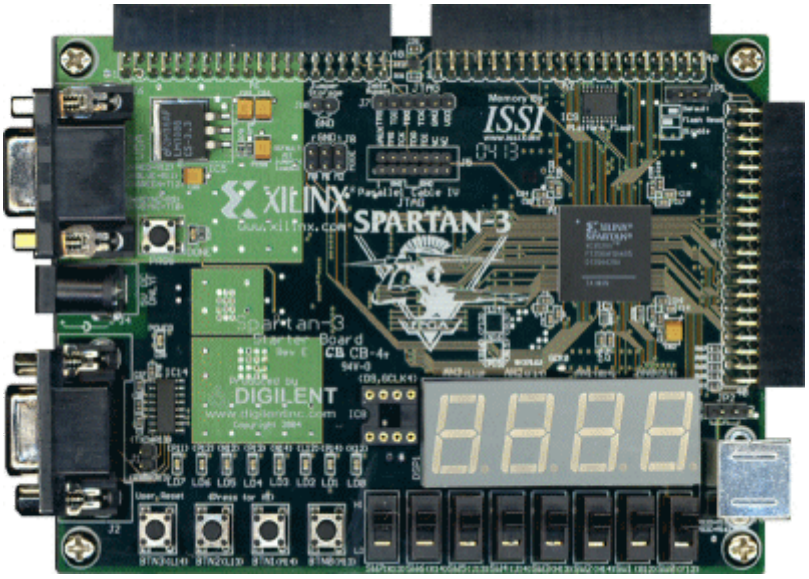
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Πανεπιστήμιο Θεσσαλίας

CE430
DIGITAL CIRCUIT LAB



EXERCISE 1:
7 SEGMENT DISPLAY DRIVER

PATSIANOTAKIS CHARALAMPOS
2116

INTRODUCTION

In this exercise, a message on the 7 segment display monitor is going to get driven by the fpga. The exercise is separated in 4 parts:

- Part A: Decoding the binary value of digit, to driving values for the monitor.
- Part B: Displaying a static message on the monitor.
    - Changing the clock period
    - Filtering the reset signal (which comes from external environment.
    - Driving the AN and the outputs of decoder to monitor.
- Part C: Rotating the message, when a button is pushed.
    - Saving the message in an array of flip flops.
    - Filtering the change_message signal from external environment.
- Part D: Rotating the message with standard delay.
    - Make a FSM for this reason, generating the change_message signal.

All the parts are succesed.

PART A

7 Part Decoder

In this exercise, information is encoded in 4-bit buffer. As a result, information must be decoded to drive the 7 segment display.

Decoding table:

INFO	INPUT	OUTPUT
0	0000	0000011
1	0001	1001111
2	0010	0010010
3	0011	0000110
4	0100	1001100
5	0101	0100101
6	0110	0100000
7	0111	0001111
8	1000	0000000
9	1001	0000100
A	1010	0001000
C	1011	0110001
E	1100	0110000
F	1101	0111000
H	1110	1001000
J	1111	1000111

OFF (0) value for ON segment .
ON (1) value for OFF segment.

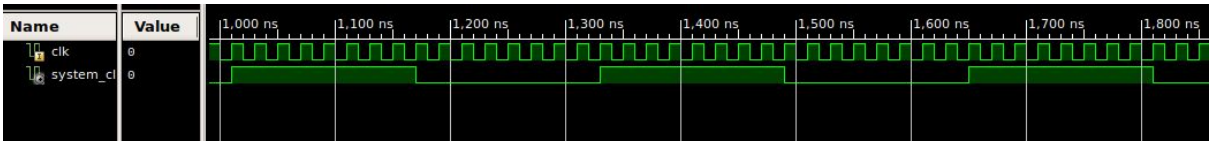
PART B

4-BIT DRIVING

The CLOCK signal

In order to drive correctly our output, we need 320 ns to load our segment. As soon as, FPGA's period is 20 ns, we need to multiple it 16 times, or to divide the frequency 16 times. We will use the new period for the whole circuit, to reduce its complexity.

So, we are going to use the DCM unit which is provided, to divide our frequency. By controlling the waveforms, we can see that the DCM is correctly instantiated.

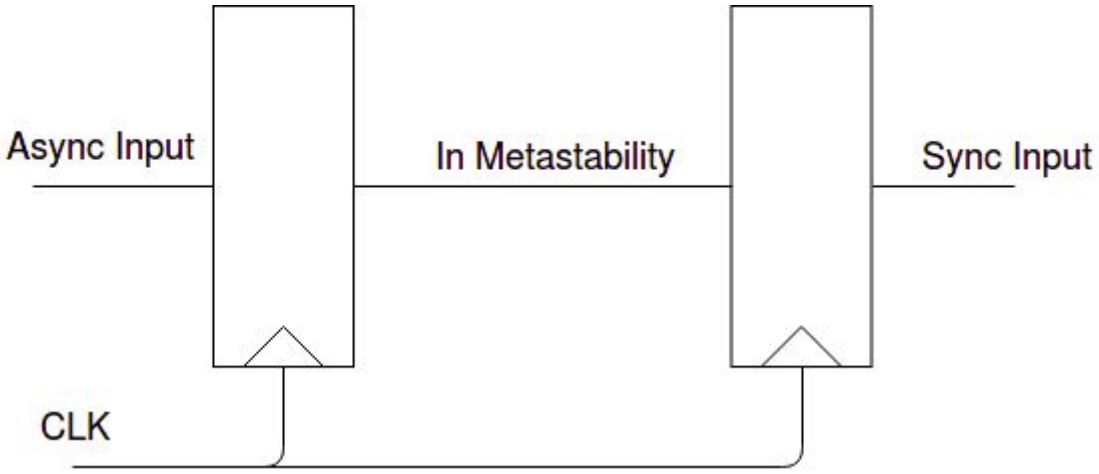


The RESET signal

Reset is a signal given by user. There are 2 points that must be focused.

1. It must be guaranteed that flip flop's setup is not violated in the circuit. Reset value changes in a random time. So, it is not impossible reset value is disabled at setup time, enabling transferring data then and causing metastability.
2. Due to mechanical partitions, reset signal is usually bounced. Which means that user may press one time reset button, but reset signal value changes e.c. 100 times.

For the first one, 2 filp flops are used. In case of setup violation, only the first one gets into metastability. Within a random duration, smaller than the time period, the flip flop gets out of metastability. The fact that the output value is random does not affect the circuit (is going to be explained in next part).



For the second bullet, the reset signal must get sampled.First of all, the synchronised signal gets counted how many clock periods is at ON state. If it is continuously more than a parametric number of clock periods at ON state, an output signal at ON state with one clock period is generated.

A fact is that the asynchronous input, in case it makes setup violation, is going to have the same value untill the other cycle. Otherwise it has not enough duration to be a valid signal. As a result, if the input is not expected, it cannot get sampled and will not be driven to the output, as it will not reach the counter at the desirable value.

Furthermore, DCM requires a reset input, with duration at least 3 clock* periods. As the sampled reset signal lasts 1 period, it has to get multiplied with 3. So the sampled reset gets into a new FSM, which set the reset signal ON for 3 clock periods. Finally, the produced reset signal is used for the DCM and the rest of the circuit.

**clock is DCM CLKInput, so FPGA's clock*

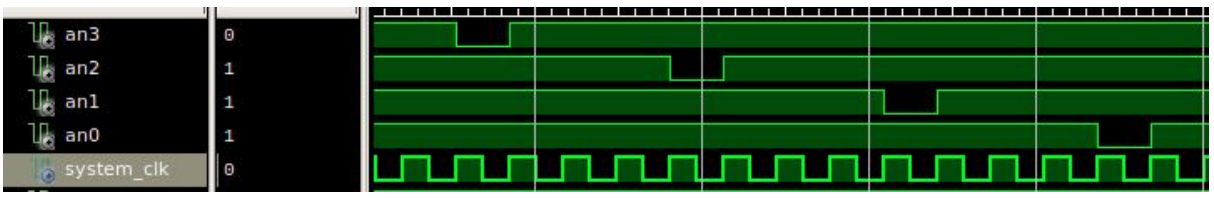
The AN FSM MODULE

After this, the output signals must be secured that are not overlapped. For this reason, an AN FSM is used, implemented with a 4-bit counter, which activates each AN signal every 4 clock periods. For

this exercise, the counter can be described in this table (initialised with at value 1111):

COUNTER	AN0	AN1	AN2	AN3
1111	1	1	1	1
1110	0	1	1	1
1101	1	1	1	1
1100	1	1	1	1
1011	1	1	1	1
1010	1	0	1	1
1001	1	1	1	1
1000	1	1	1	1
0111	1	1	1	1
0110	1	1	0	1
0101	1	1	1	1
0100	1	1	1	1
0011	1	1	1	1
0010	1	1	1	0
0001	1	1	1	1
0000	1	1	1	1

Waveforms for one counter's period:



In this part, a static message of 4 digits is going to be displayed in the monitor. So, a static 4X4 array of flip flops (called message) is setted in the circuit. When reset signal is enabled, the values written are the binary encoding of 0,1,2,3. The signals of this array, are filtered in the AN FSM, in order to send to output the correct digit. For example, when AN0 is enabled, then the output digit must be message[0]. As a continuous, the binary value of the enabled digit is driven to the Decoder of part 1 and is sended to the monitor.

PART C

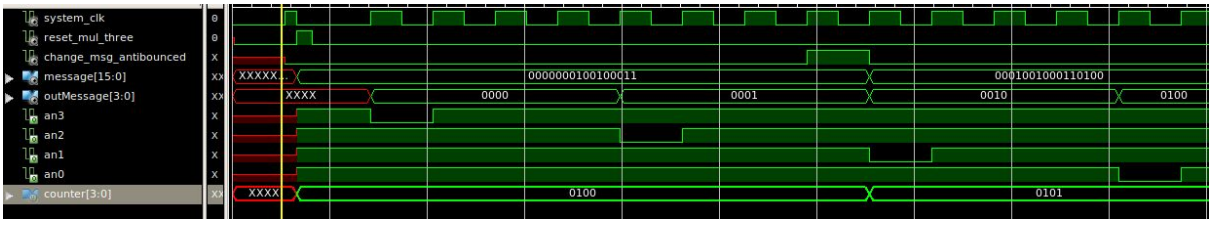
STEP ROTATION BY BUTTON

The next step in this exercise, is to display a 16-digit message, which will get rotated when user pushes a button. To make this, a memory module is needed to get instantiated, in order to save the message. Basic parts of this module :

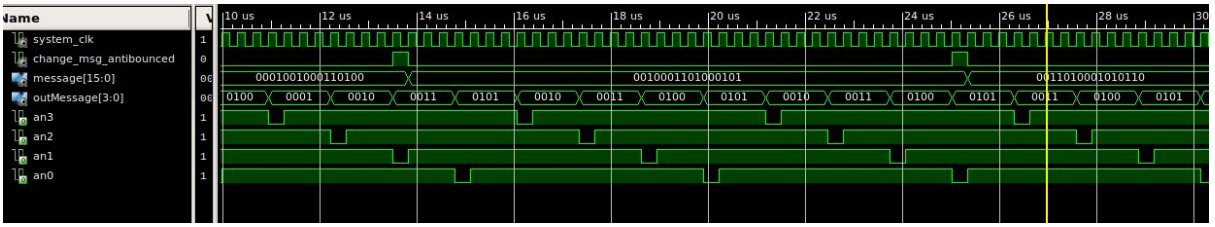
- Inputs:
  - Clock
  - Reset
  - RE (Read Enable)
- Output:
  - a 4X4 table, representing the 4 bits the display is monitoring.
- 16X4 flip flops, as the basic memory where the message is permanently (after reset) saved.
- A counter, representing the next address, going to be added in output.

While the reset signal is enabled, for each address in the memory table, the value of the address is written. Also the output table has values of {mem[3],mem[2],mem[1],mem[0]}, and counter gets value of 4, as the next output digit is the 4th one. The reset is asynchronous.

When RE is enabled, the digits of output table are shifted left, and at address 0, new value is the value of memory table at address with the value of counter.

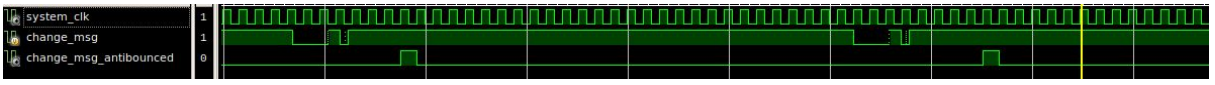


When reset enabled



While AN changes, or change message signal (the sampled) is enabled

At this part, we are going to get the RE signal via a button. The FPGA button which is going to be used is the L13. So, like the RESET button, an antibouncer is needed, in order to prevent the effects explained above. The module used for the reset signal, is used also in this occasion. The difference here is that the system clock is used here (320 ns) in place of the FPGA's clock (20 ns) used for reset.



Change message signal sampling.

The output table replaces the static 4X4 array from part2, and get filtered from the AN FSM, to reach the decoder.

PART D

STEP ROTATION BY STANDARD DELAY

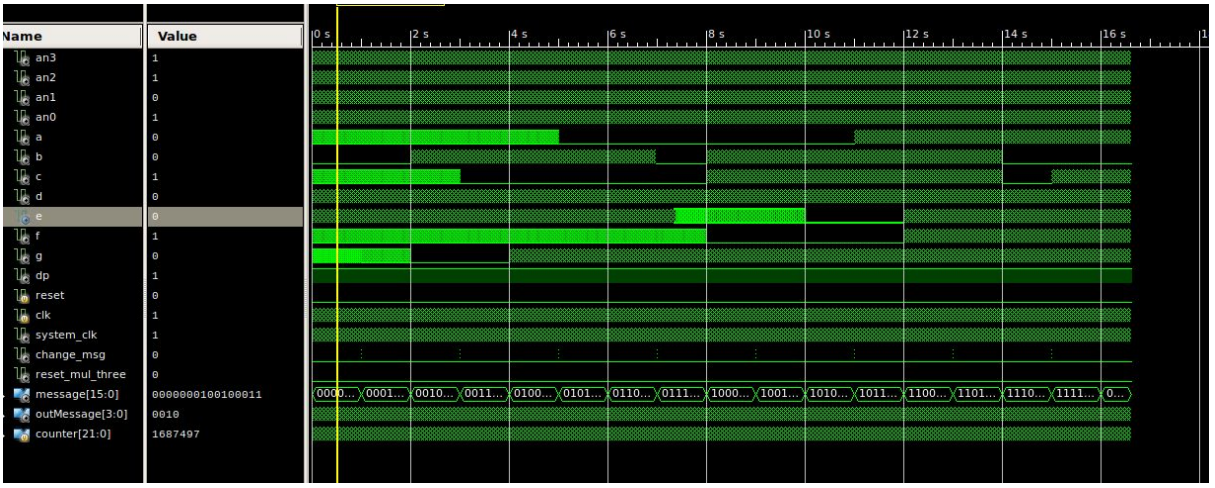
For the final step, the change message button, must be replaced by a module that enable the RE signal of memory module, at a standard delay of 1 second. In order to have this, a new FSM with a counter is instantiated in the circuit. The output of the FSM is connected with the RE input of memory.

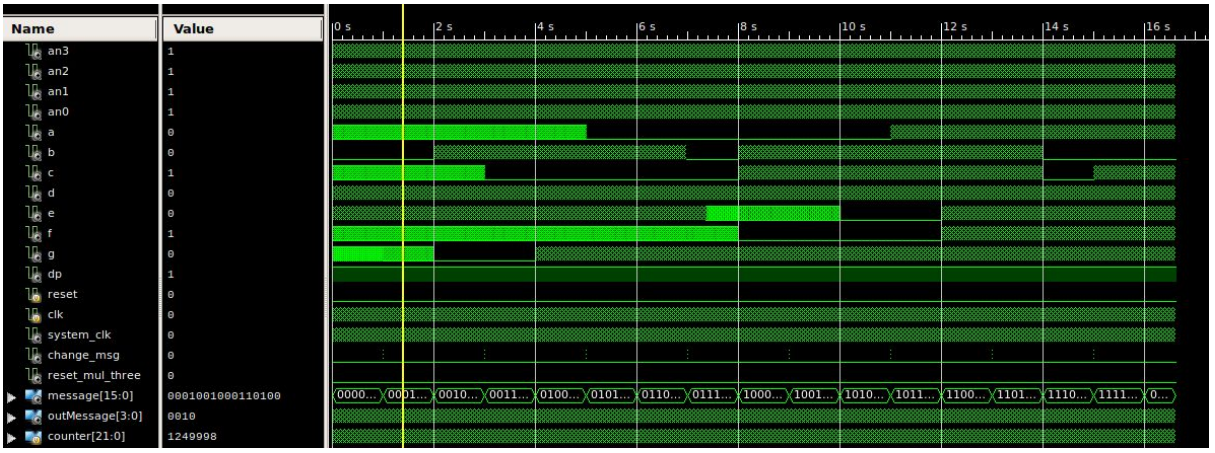
First of all, in order to check that the logic is correct, the counter is a signal of 3 bits and each time it has the value of 111, the output signal is enabled.

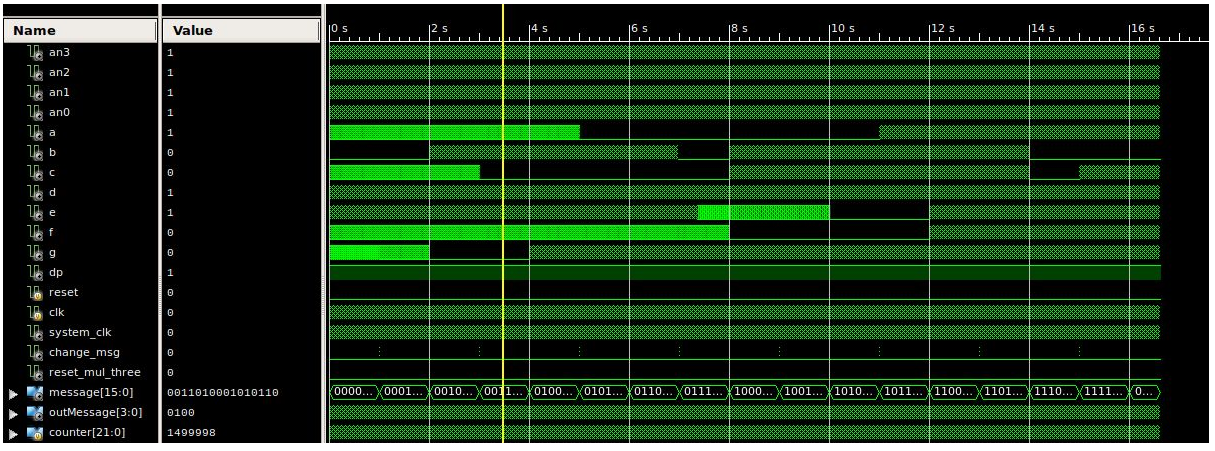
As the logic of the FSM, and general circuit is checked as correct, the circuit must get into real conditions. Which means, the counter must get real size. The system clock has a period of 320 ns, and the message must change in about 1s. So the RE signal must be enabled each

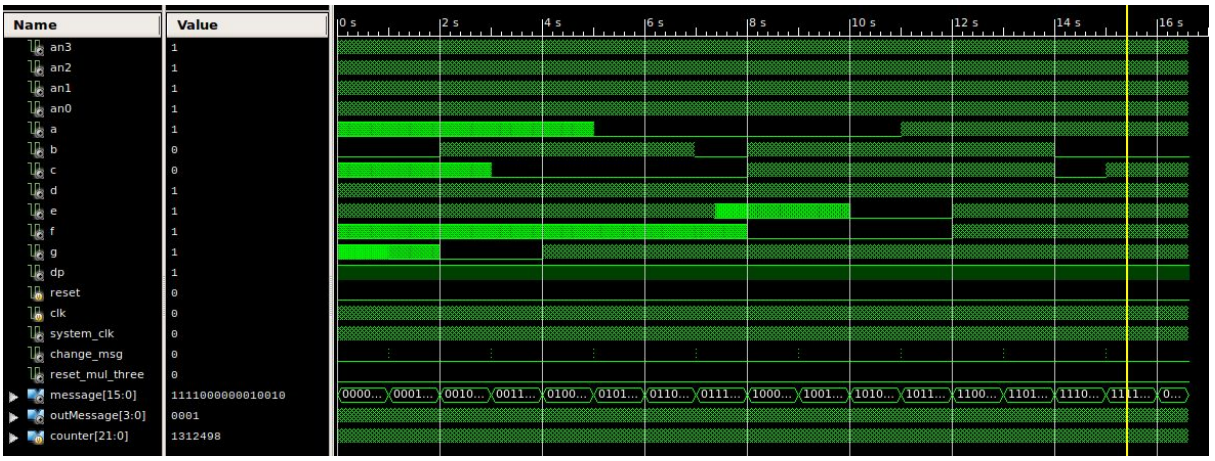
3.125.000 clock cycles. In order to reach this number, a 22-bit counter must be instantiated.

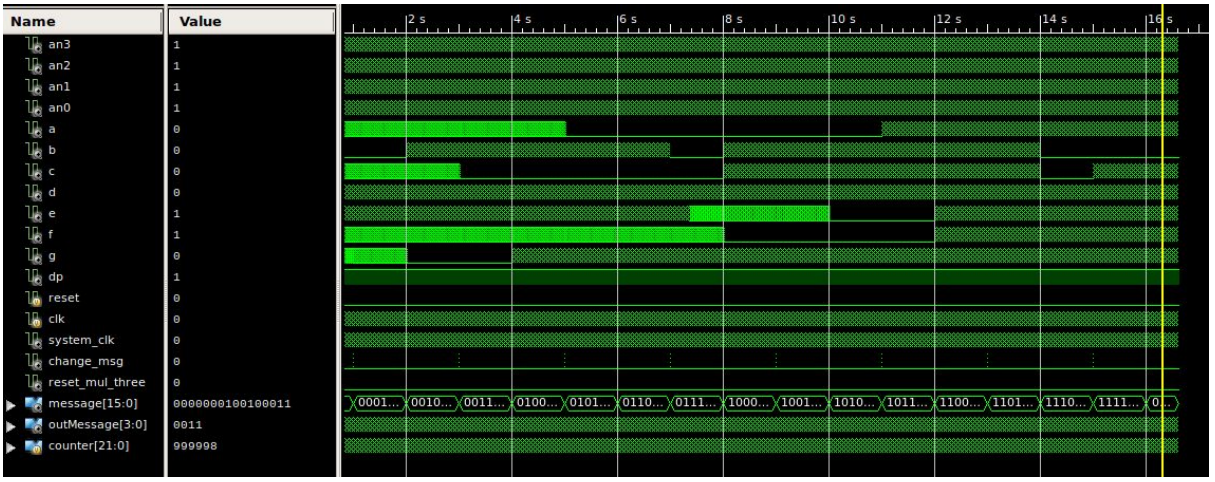
After a simulation of 37 hours, the waveforms prove that the message rotates each 1 sec, correctly.








Name
Value
an3 1
an2 1
an1 1
an0 1
a 1
b 0
c 0
d 1
e 1
f 1
g 1
dp 1
reset 0
clk 0
system_clk 0
change_msg 0
reset_mul_three 0
message[15:0] 1111000000010010
outMessage[3:0] 0001
counter[21:0] 1312498
0 s
2 s
4 s
6 s
8 s
10 s
12 s
14 s
16 s
0000... 0001... 0010... 0011... 0100... 0101... 0110... 0111... 1000... 1001... 1010... 1011... 1100... 1101... 1110... 1111...


Name
Value
an3 1
an2 1
an1 1
an0 1
a 0
b 0
c 0
d 0
e 1
f 1
g 0
dp 1
reset 0
clk 0
system_clk 0
change_msg 0
reset_mul_three 0
message[15:0] 0000000100100011
outMessage[3:0] 0011
counter[21:0] 999998
2 s
4 s
6 s
8 s
10 s
12 s
14 s
16 s
0001... 0010... 0011... 0100... 0101... 0110... 0111... 1000... 1001... 1010... 1011... 1100... 1101... 1110... 1111...