UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Advertisement with animations implemented on FPGA board

Student name: Istrate Oana Georgia Cristiana

Teacher advisor: Ileana Blaj

Faculty of Automation and Computer Science

Specialization: Computer Science in English

Group 30412

First Year in the Academic year 2024 – 2025

Subject: Digital System Design

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

==========Table of Contents==========

I. Board Specifications ==========

I chose to implement and test this project on a Basys 3 board, developed by Digilent Inc. Basys 3 is an entry-level FPGA development board produced by Digilent, based on the Xilinx Artix-7 FPGA (XC7A35T-1CPG236C). It is commonly used in academic settings for teaching digital design, VHDL/Verilog, and hardware projects.

As shown in the image, the board has a lot of components with different functionality. In my project, I will use the following: Four Seven Segment Displays as outputs for the animation, Slide switches for inputs and the Clock for the counters.

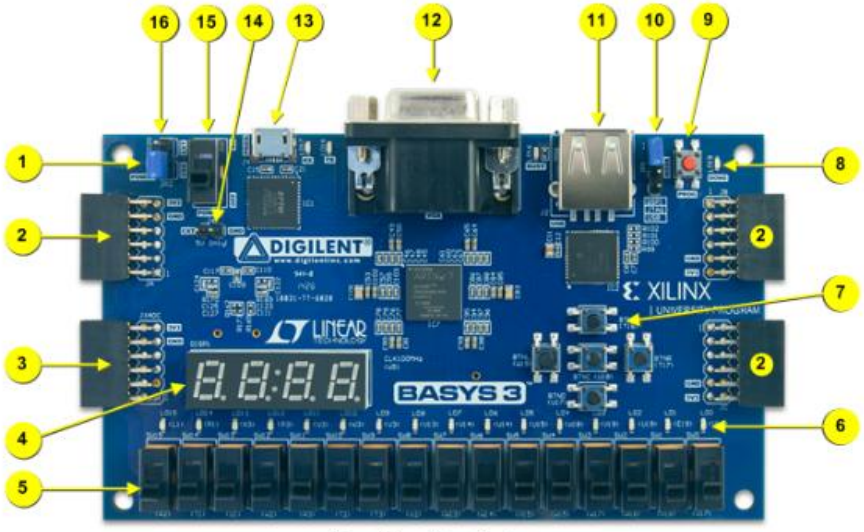


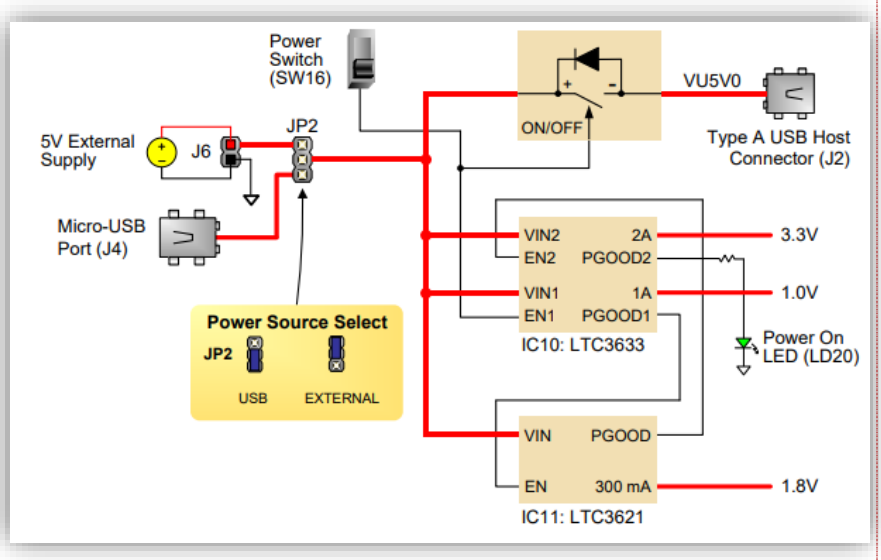
Figure 1. Basys3 board features

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod connector(s)	10	Programming mode jumper
3	Analog signal Pmod connector (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

a. Power Supply

Of course, this board, like any other electric device, wouldn't work at all without electricity.

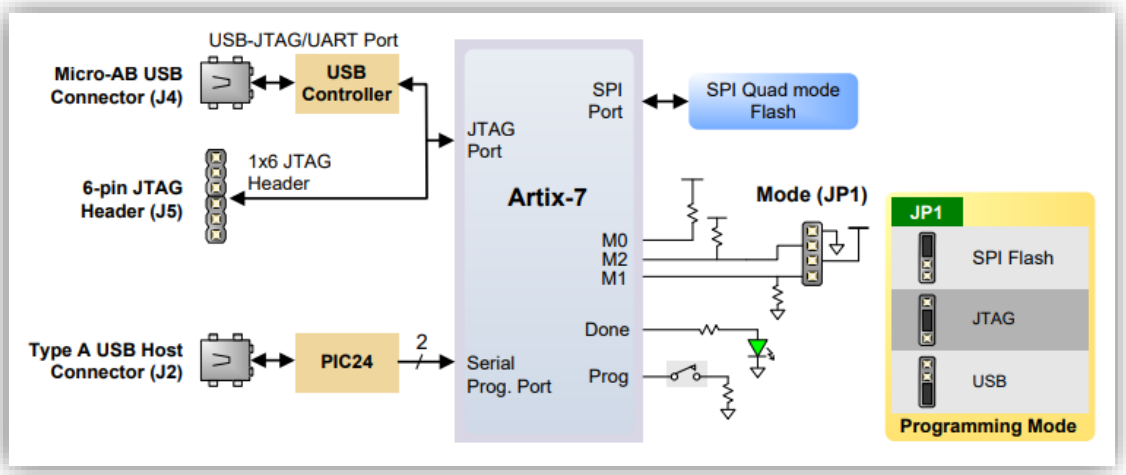
The Basys 3 power supply can be turned on or off by a single logic-level. The board can receive power from the USB-JTAG port, or from a 5V external power supply. Jumper JP3 determines which source is used. I used the USB mode, because it was easier to program and test my code after every small change.



b. Configuration

Now, the board has current running through it, but it is useless if we don't tell the board how to utilize it. We can configure the board in 3 ways:

1. SPI Flash – a file stored in the nonvolatile serial flash device can be transferred to the FPGA using the SPI port.
2. JTAG – a PC can be used to run, through the Vivado suite, the program any time the board is turned on
3. USB – A programming file can be transferred through a USB stick



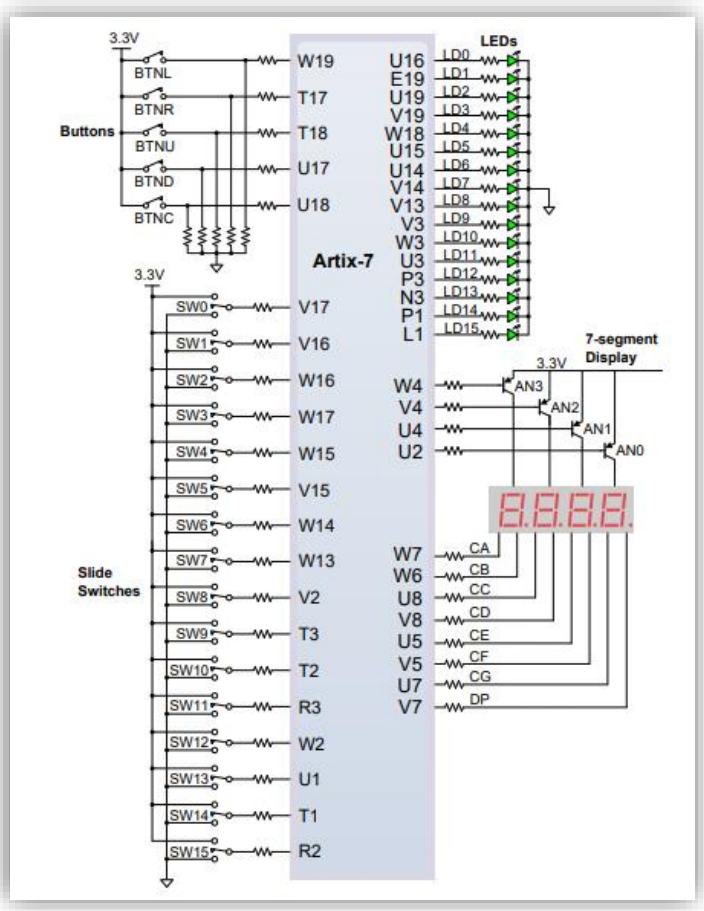
The FPGA configuration data is stored in files called bitstreams. Bitstreams are stored in SRAM-based memory cells within the FPGA. This data defined the FPGA's logic functions and circuit connections, and it remains valid until it is erased by removing board power, by pressing the reset button attached to the PROG input, or by writing a new configuration file using the JTAG port.

c. Oscillators/Clocks

How does the board know how to measure time? To go from one state to another? To count steps? Using a clock, of course! The Basys 3 board has a built-in ***100 MHz oscillator*** connected to pin W5. Later, we will see how I use the clock to control some counters and create animations.

d. Basic Input/Output

Finally, we have power, we have a good configuration, and a clock controlling the timing. But where is the animation displayed and how? And how do we select which animation to display? How do we tell the board that we want to use the leftmost switch? Here is a diagram showing each input/output assigned pins. The push buttons and slide switches are connected to the FPGA via series resistors to prevent damage from inadvertent short circuits. Slide switches generate constant high (up) or low (down) inputs depending on their position.



In this project I will use some switches and the Seven Segment Display. In order to tell the board which input/output from my code I want to put on which pin, I need to create a constrains file. A constrain files tells the Board the relation between code and input/outputs. Without this, we couldn't run the code on the board.

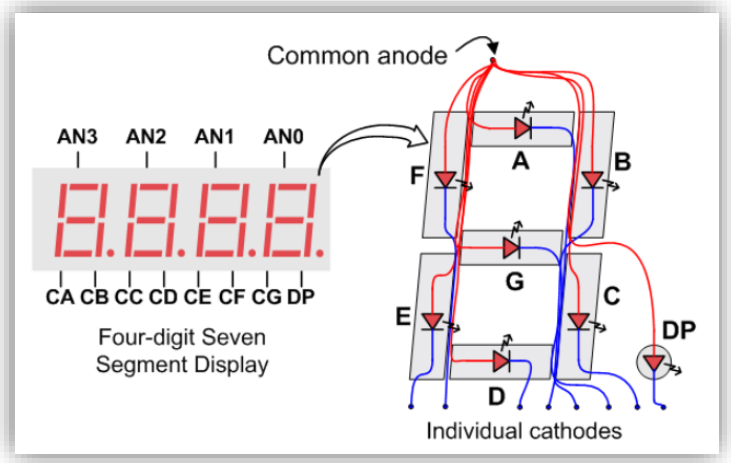
1. *Switches*

My project uses 4 switches. Their scope and action will be discovered later in this document:

```
START           - pin U1
SEL [1]         - pin R2
SEL [0]         - pin T1
RST             - pin V17
EasterEgg [3]   - pin W13
EasterEgg [2]   - pin W14
EasterEgg [1]   - pin W15
EasterEgg [0]   - pin W16
```

2. *Seven Segment Display*

The SSD works on quite a simple principle. In negative logic. So lets say we want to display "8", then the cathodes A, B..., G will all have value 0, since that turns them all on. If we want to display this 8 on all anodes, then we send the data to all of them. So, for a segment to be on, Anode must be active (1) and the corresponding Cathode must be inactive (0).



In this picture, we can see how connectivity works. We don't use DP here.

II. Introduction ==========

Proposals with a maximum score of 20 points

Active-HDL implementation via VHDL code, with top-level structural description and practical functionality demonstrated on FPGA board.

Note: for physical implementation on FPGA boards the project specification is done by VHDL code. The recommended working method is as follows:

1. Specify the project (write VHDL source code) in Active-HDL

2. Functional simulation in Active-HDL

4. Physical testing on the host FPGA board

*A9) Design an advertisement with multiple animations. 7-segment displays will be used. The text to be displayed will consist of symbols of an available alphabet. The advertisement will have several operating modes (**minimum 4**) that can be selected by the user, from the FPGA board switches. Use the quartz oscillator built into the FPGA board (the clock signal will of course have to be split). Examples of operating modes: "flowing" writing from right to left, flickering, letter-by-letter display, etc. Because not all letters can be represented on a 7-segment display, a maximum alphabet will be created and the messages will be composed of the symbols of that alphabet. The message will be stored in a memory so that it can be easily changed.*

You are part of a project and are asked to design a simple advertisement device, with several operating modes. The advertisement should have animations accessed via switches, and those animations must be implemented on the available 7 segment display on the board, using symbols of an available alphabet.

a. Operating Procedures:

Initial state: When START is 0, the advertisement displays the predefined text in a static way.

START: When this switch is on, it enables the animation SEL(ection)

ANIMATION 1: when SEL is 00, the text blinks

ANIMATION 2: when SEL is 01, the text slides to right

ANIMATION 3: when SEL is 10, the text slides to left

ANIMATION 4: when SEL is 11, the text blinks each character

EXTRA ANIMATION 1: when the “secret” switches combination is on, the advertisement displays a moving long text

Those extra animations depend only on the EasterEgg switches. Any other input does not affect this and vice versa. Also, the SEL switches are *don’t cares* if START is 0.

Because we can’t really reset a switch (we can’t write code to physically turn it off), the START switch acts as a reset switch for the animations. Whatever animation is on (except the extra ones), if START is 0, the advertisement becomes static. For the Extra Animation, they reset if a “secret” switch if turned off.

So basically, I will assign a switch for RST, but RST will be used only for the counters at code-level.

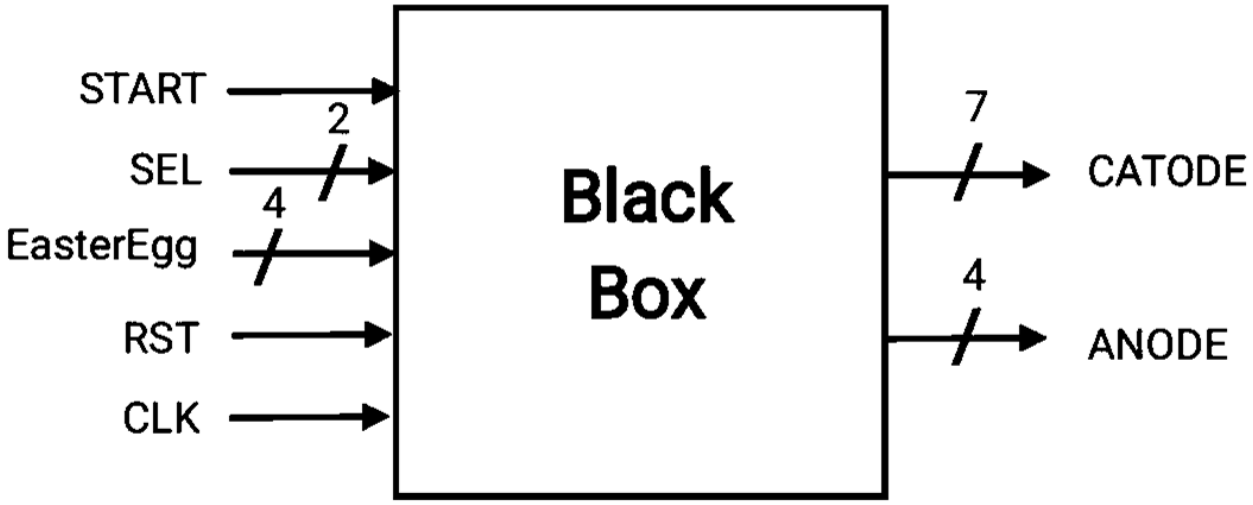
III. Black Box, Command and Execution Units ==========

For programming the project, I used the Vivado suite. To realize this project, I followed some steps:

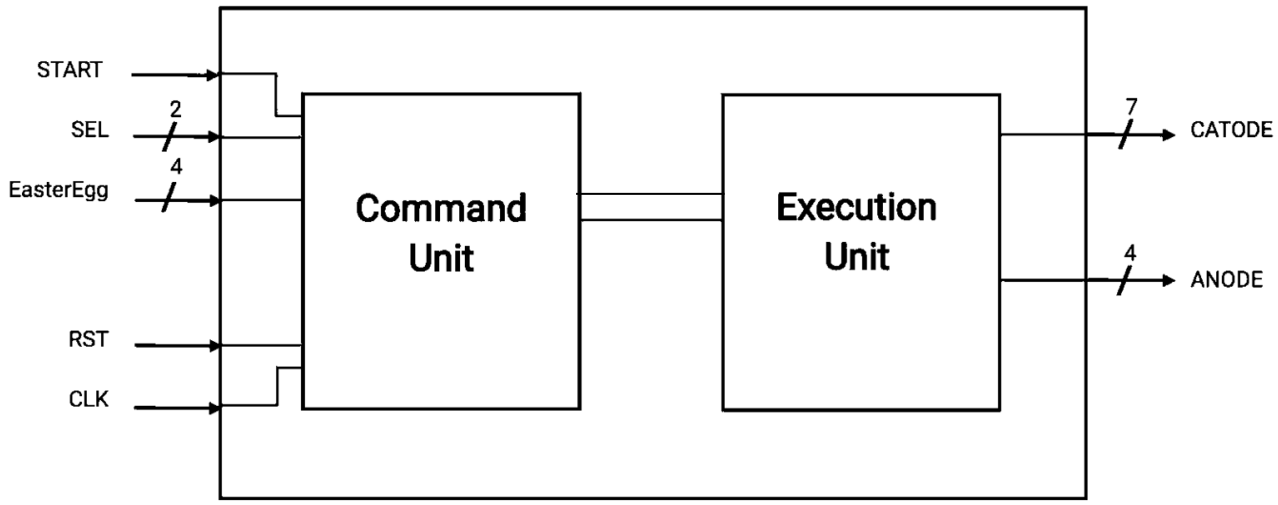
1. *Design the Black Box*
2. *Control Unit and Execution unit*
3. *State Diagram*
4. *Designing the components*
5. *Put everything one by one into the VHDL code and test on the board*

a. Black Box

In the beginning of designing any electronic device, the first step is to realize the black box. Like building a house, first you design the plan: how many floors, rooms, doors etc.. , then you start building it from the foundation to the roof. Back to our project, in the Black Box you specify the inputs and outputs of the final product.



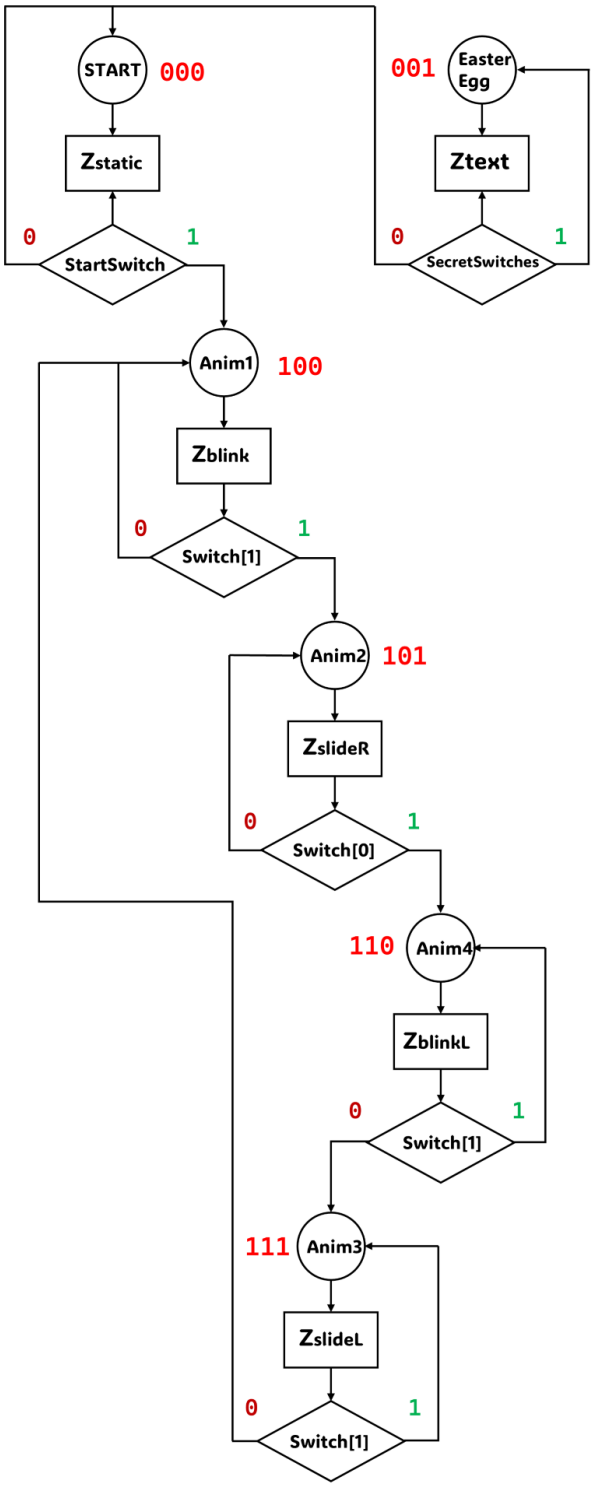
b. Command Unit. Execution Unit



The Command Unit tells the Execution unit what to do. In the Command unit we have the ROM's, the multiplexors, while in the Execution Unit we have the counters.

c. State Diagram

The system state diagram is presented in the next figure:



IV. Components==========

The project has 4 main animations being 4 characters long, and one special animation, featuring a long text. Since I implemented an alphabet in memory, the 4 character **text can be modified at any time**, by modifying the smaller memories corresponding to the animations. The main animations are being available when START is 1. This START enables the SEL switches, which, as the name suggests, selects the animations as follows:

SEL 00 => Animation 1 -- blink

SEL 01 => Animation 2 – slide to the right

SEL 10 => Animation 3 – slide to the left

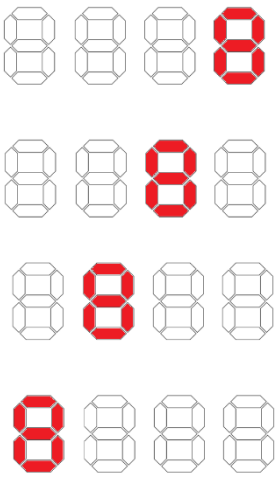
SEL 11 => Animation 4 – letter blink

EasterEgg => long text

a. Seven Segment Display

For those animations I used all 4 seven segment displays on the Basys 3. To display the actual output on it, I used a Case selection like this:

```
case digit_index is
      when "00" =>
          ANODE  <= "1110";
          CATODE <= --output--;
      when "01" =>
          ANODE  <= "1101";
          CATODE <= --output--;
      when "10" =>
          ANODE  <= "1011";
          CATODE <= --output--;
      when "11" =>
          ANODE  <= "0111";
          CATODE <= --output--;
 end case;
```

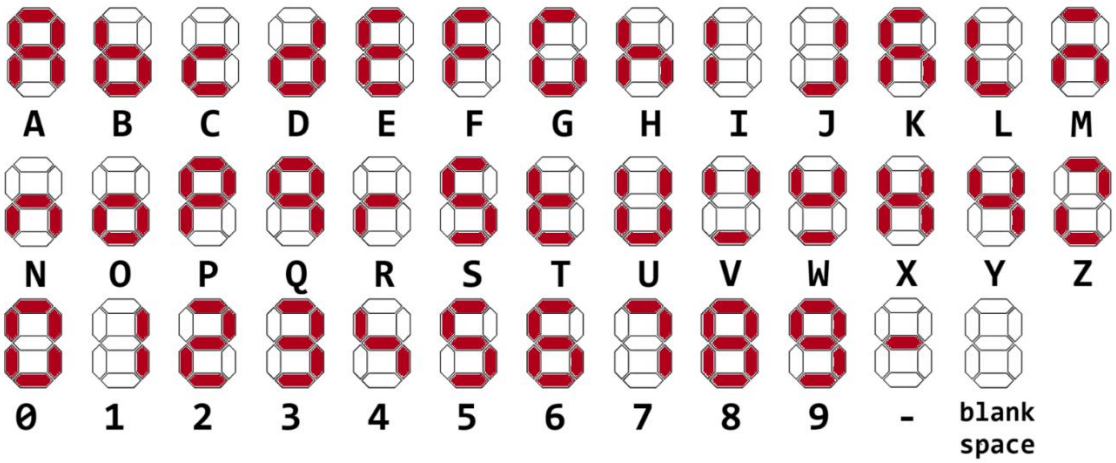


As we notice, each SSD is displayed individually. To trick the human eye into seeing all 4 on at the same time, we quickly switch between SSD's, using digit_index and the method called time multiplexing.

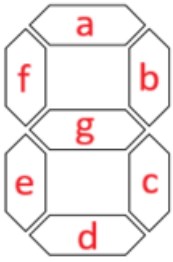
Digit_index takes the value of refresh_counter (19 downto 18), which is a very fast counter that uses the 100MHz clock. Bits 19 and 18 change fast enough to rotate through the digits about 1,250 times per second, which is flicker-free for the human eye.

b. Read Only Memory (ROM) + animations

But what about the actual displayed value? How does our board know that we want to display the letter A? An easy solution to this is the implementation of an alphabet in a Read-only Memory. Below it's a visual description of every character, how it will look like displayed.



Representation is done in binary, using negative logic. So 0 means on and 1 means off. As mentioned before, the seven segments are a b c d e f g, in the order shown in the drawing. A table containing the actual codes for every character can be seen next.

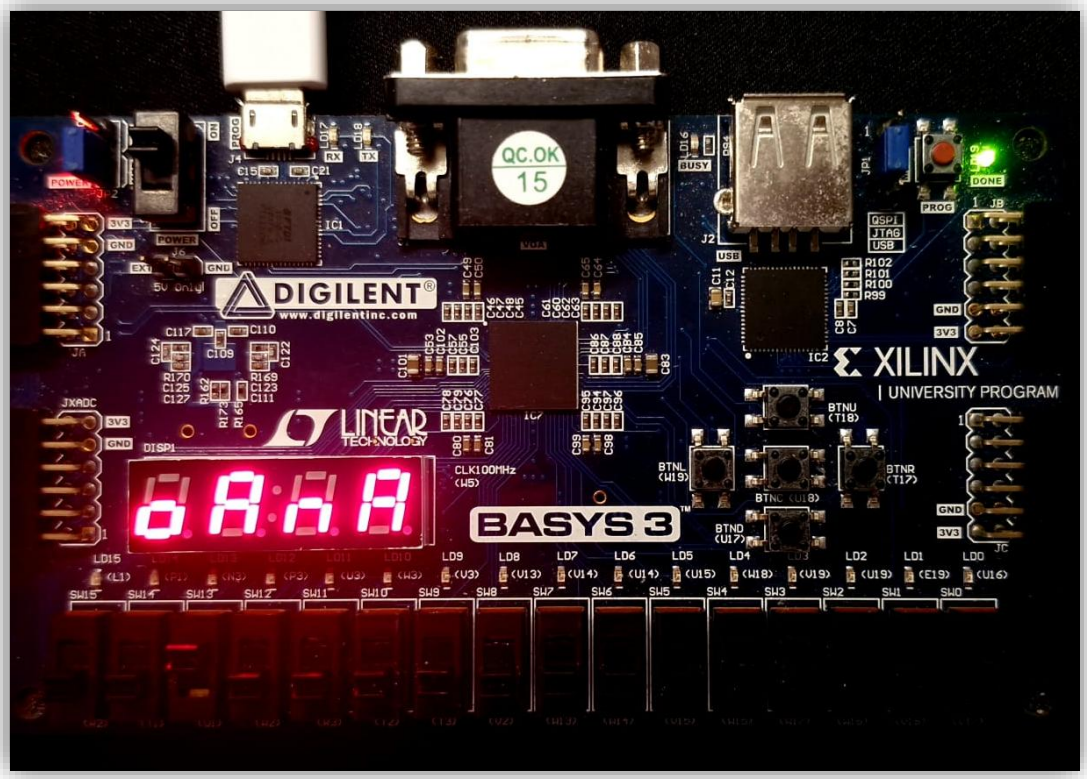


Character	abcdefg
A	0001000
B	0011111
C	1110010
D	1000010
E	0110000
F	0111000
G	0100001
H	1101000
I	1111001
J	1000111
K	0101000
L	1110001
M	0101010
N	1101010
O	1100010
P	0011000
Q	0001100
R	1111010
S	0100100
T	1110000
U	1000001
V	1010101
W	1010100
X	1001000
Y	1001100
Z	0010011
0	0000001
1	1001111
2	0010010
3	0000110
4	1101100
5	0100100
6	0100000
7	0001111
8	0000000
9	0000100
-	1111110
blank space	1111111

ROM memories were also a big help in the making of animations. Below are pictures of the animations I implemented in this project. This project was built in a modular way. That means that, in order to display and animate a different work / text, all we'd have to do is to modify the said memories, looking at the alphabet.

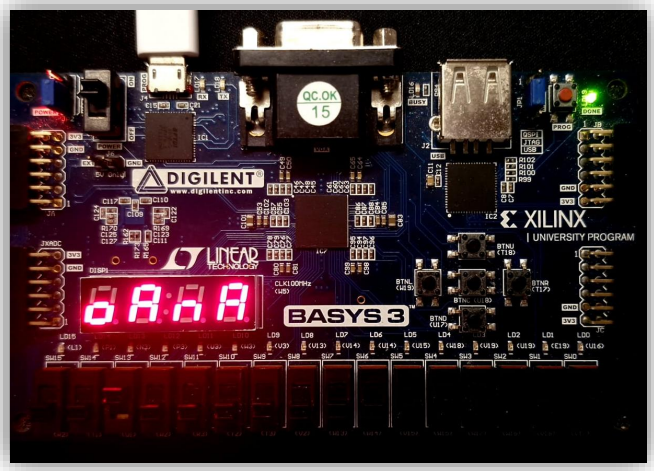
Static

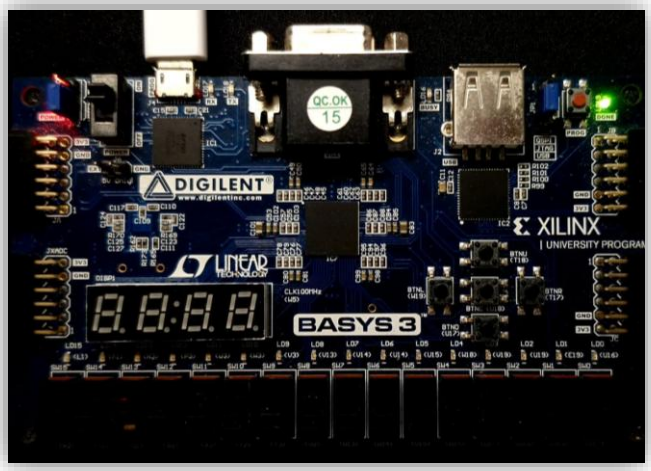
The Memory ***startMemory*** has 4 characters, featuring the 4- character long word that will be displayed, in our case, O, A, N and A.



Animation 1 - Blink

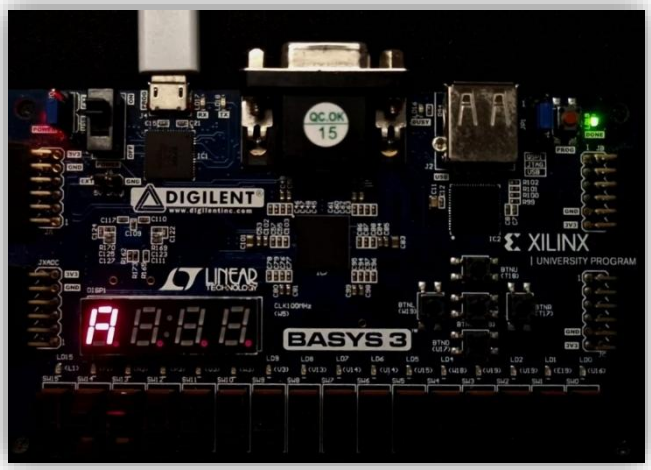
The Memory ***startMemory*** has 4 characters, featuring the 4 character word that will blink. In our case, O, A, N and A.

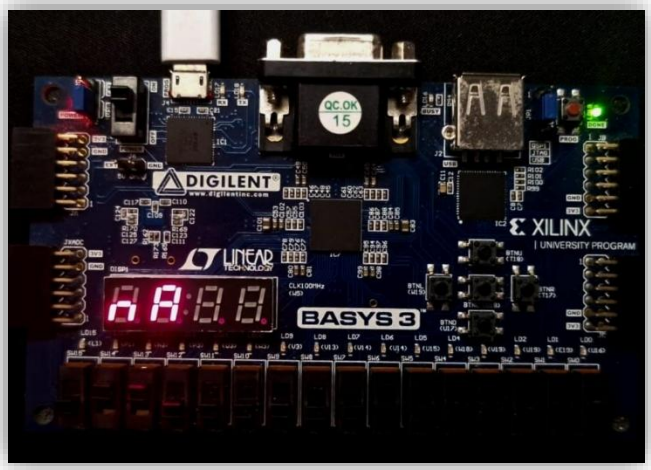


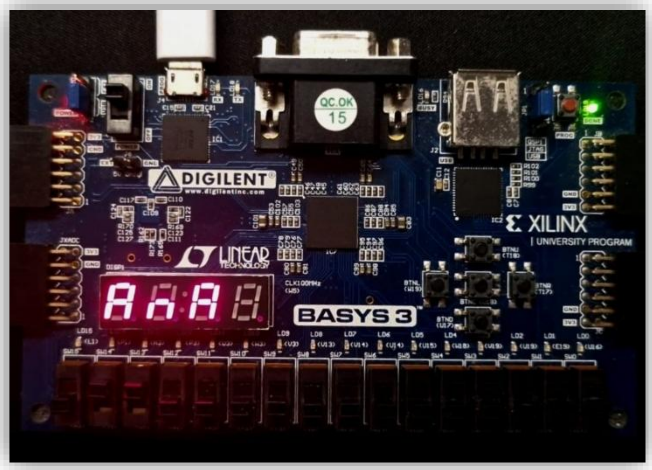


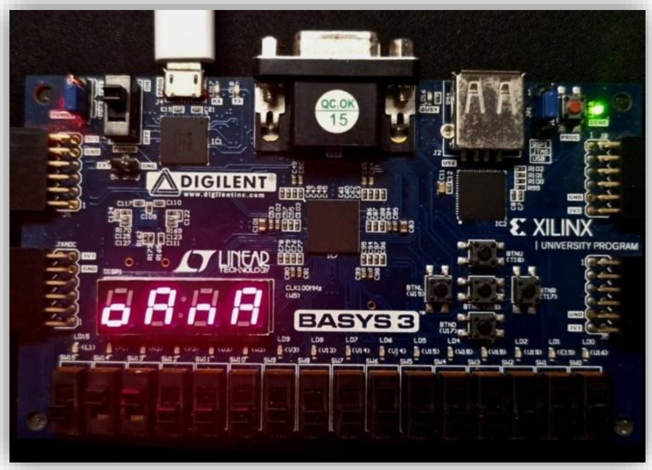
Animation 2 – shift to right

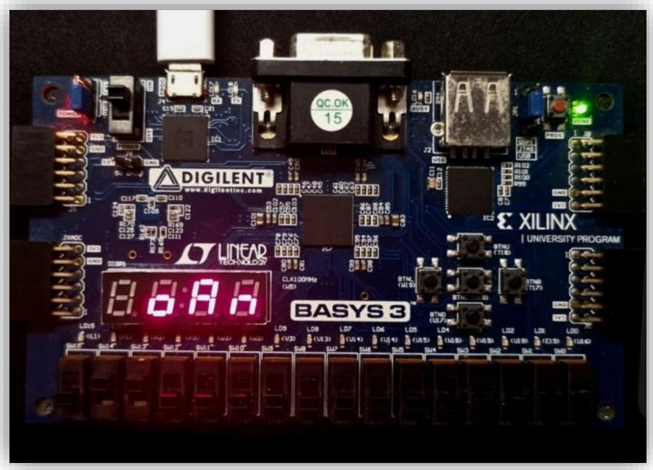
The Memory ***shiftMemory*** has 8 characters, featuring the 4 characters from startMemory and 4 blank spaces, for a nice shifting.

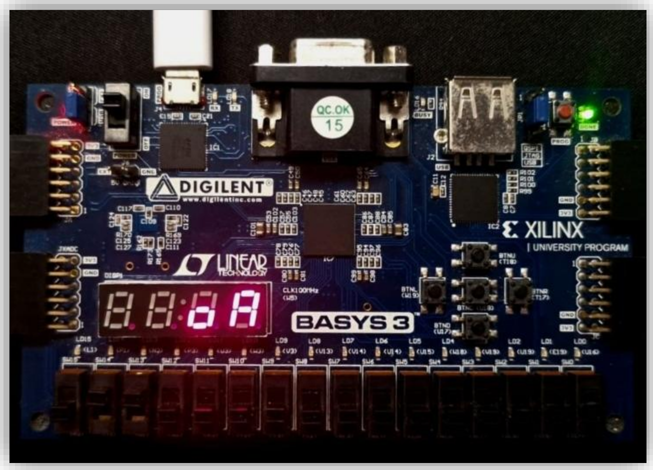


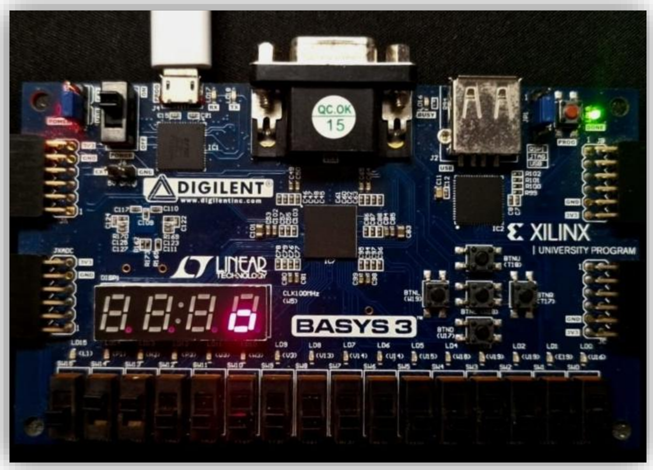


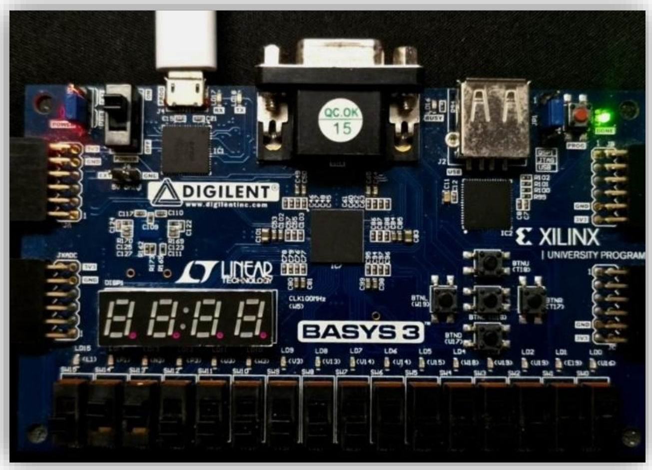

DIGILENT
XILINX
UNIVERSITY PROGRAM
BASYS 3


DIGILENT
XILINX
UNIVERSITY PROGRAM
BASYS 3


DIGILENT
XILINX
UNIVERSITY PROGRAM
BASYS 3


DIGILENT
XILINX
UNIVERSITY PROGRAM
BASYS 3


DIGILENT
XILINX
UNIVERSITY PROGRAM
BASYS 3


DIGILENT
XILINX
UNIVERSITY PROGRAM
BASYS 3

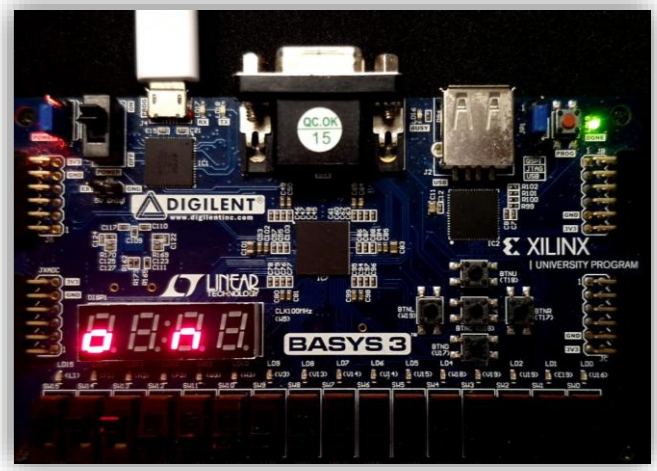
Animation 3 – shift to left

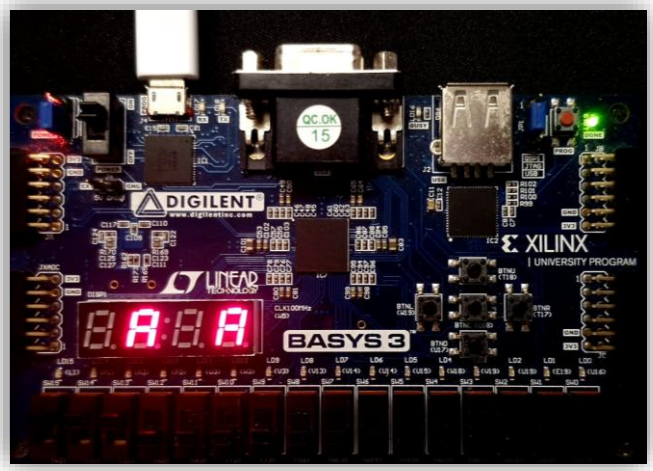
The Memory ***shiftMemory*** has 8 characters, featuring the 4 characters from startMemory and 4 blank spaces, for a nice shifting.

I will not include pictures for this animation since the animation phrases are the same as on Animation 2, but on reverse order.

Animation 4 – blink 2 by 2 characters

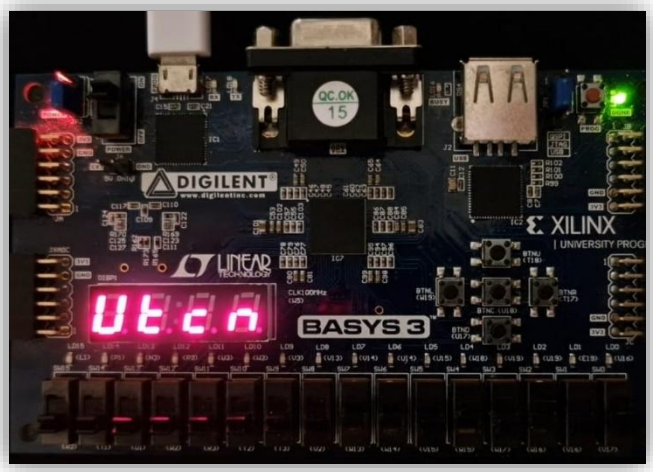
The Memory ***startMemory*** has 4 characters. We will display the first and third anode when blink signal is 0, and the second and fourth anode when blink is 1.

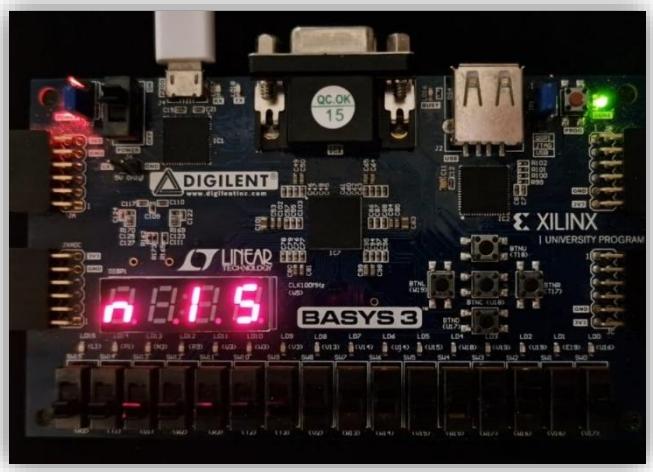


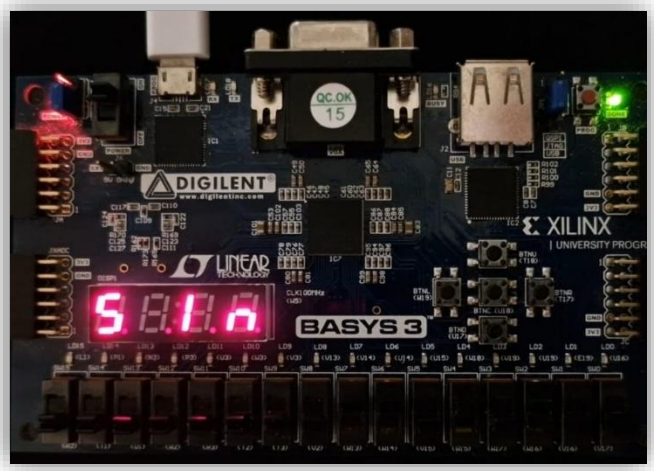


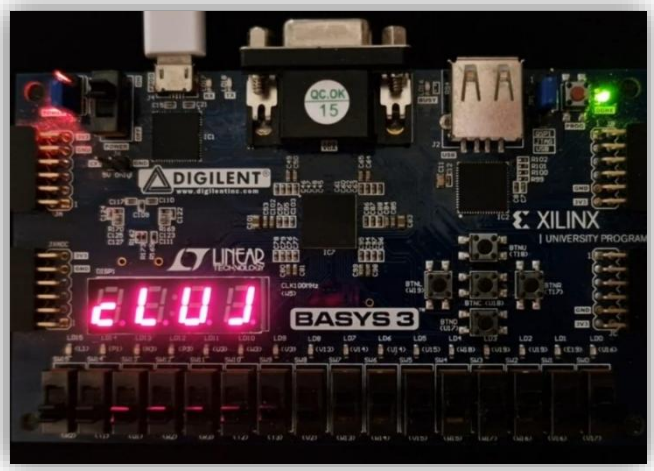
Extra Animation - EasterEgg

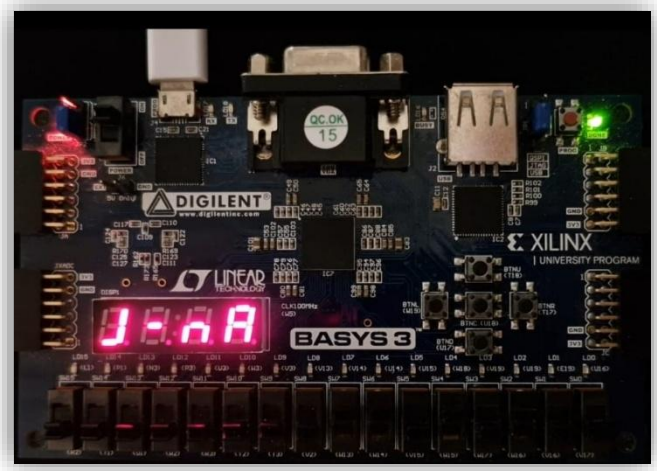
The Memory ***easterEggMemory*** has 32 characters. The animation is like Animation 2 and 3, but this time a longer text is displayed: *Utcn is in Cluj-Napoca*.

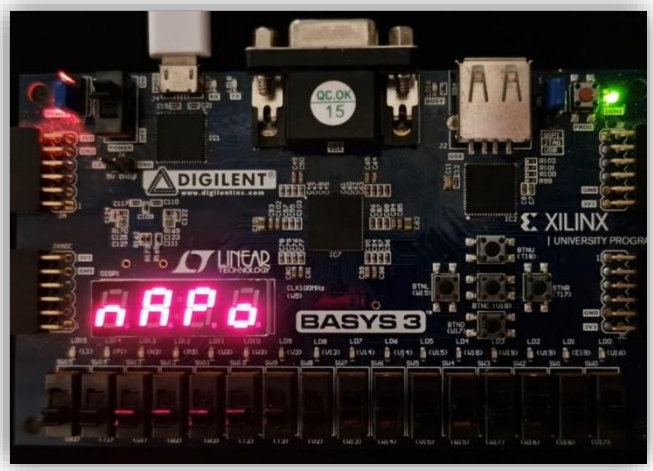


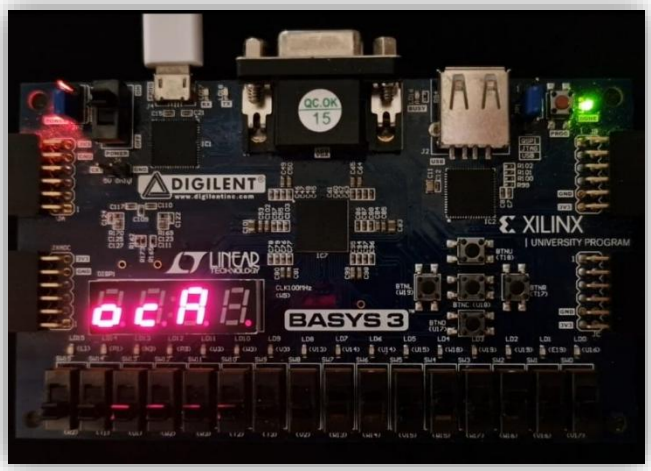


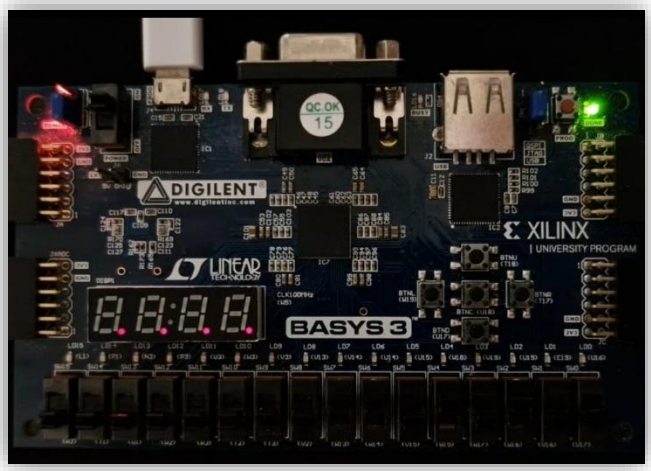





DIGILENT
XILINX
BASYS 3


DIGILENT
XILINX
BASYS 3


DIGILENT
XILINX
BASYS 3


DIGILENT
XILINX
BASYS 3

c. Counters

Since **VHDL is not like a high-level language**, we don't have access to "wait" or "delay" functions. Instead, we need counters to simulate time. In my project, I used the following counters and signals:

refresh_counter – Used to refresh the display and animations
–It increases with each clock tick (100MHh ~ 10 ns)
–Controls animations speed and digit multiplexing (digit_index)

shift_counter – Used to control the shift animations
–counts to advance the scroll index

digit_index – Used to cycle between the 4 anodes

blink – Used for blink animation

shift_slide – Acts as a trigger that tells the animation to slide to next step

The ones not mentioned have similar use to the ones above. Counters and signals play a major role in this project, and in order to not mess up the currently working animations, I opted for adding new ones when needed. This could be improved in the future.

V. Further Development ==========

As every little or big thing on this Earth, this project isn't perfect. But there is always room for improvement! Here are some things I believe could make this project better in the future:

- Better use of counters, reuse them in more places, therefore less counters used

- Less ROM used, only one (besides the alphabet)

- More dynamic animations, like a "snake" that goes through segments, then makes a text appear.

- Another 4 SSD's or more for larger advertisements.

- Possibility of inserting from a keyboard the text of the advertisement.