# ADD FOR A SPORTS CLUB-DSD FIRST YEAR PROJECT

Braica Patricia

Project supervisor: Timar Mihai

1. SPECIFICATIONS

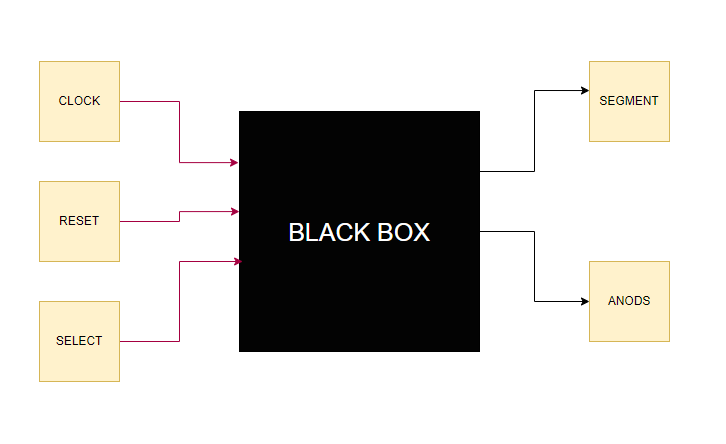
You are required to make an ad with multiple animations. The seven segment display will be used for this purpose. The text will consist of symbols of an available alphabet. The ad will have multiple operating modes (minimum 4) that can be selected by the user from the switches of the FPGA. The quartz oscillator imbedded in the board will be used, it’s frequency will be, of course, divided. Because you cannot represent all letter using the SSD, a maximal alphabet will be created.

The finished product will have 8 different animations for the same text and you will be able to switch the message without difficulty.

1. DESIGN

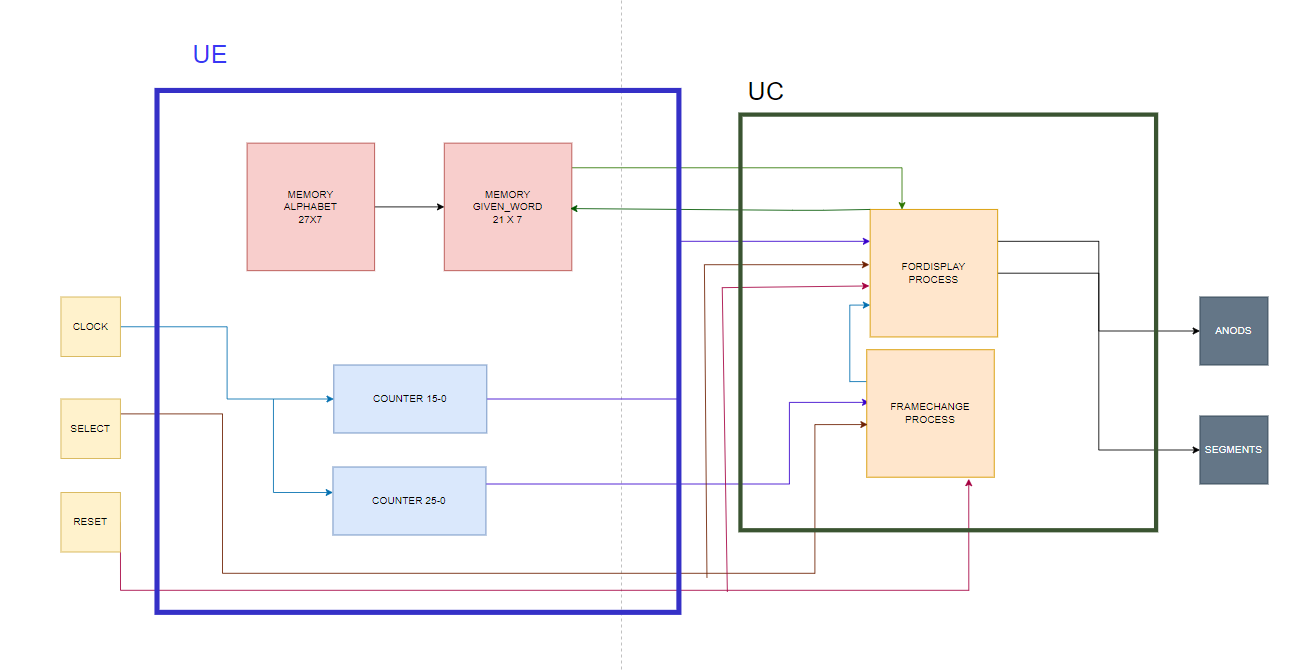
Inputs:

* Clock-100 MHz
* RESET-when pressed nothing is shown
* Select-selects one of the 8 animations
  1. **BLACK BOX**

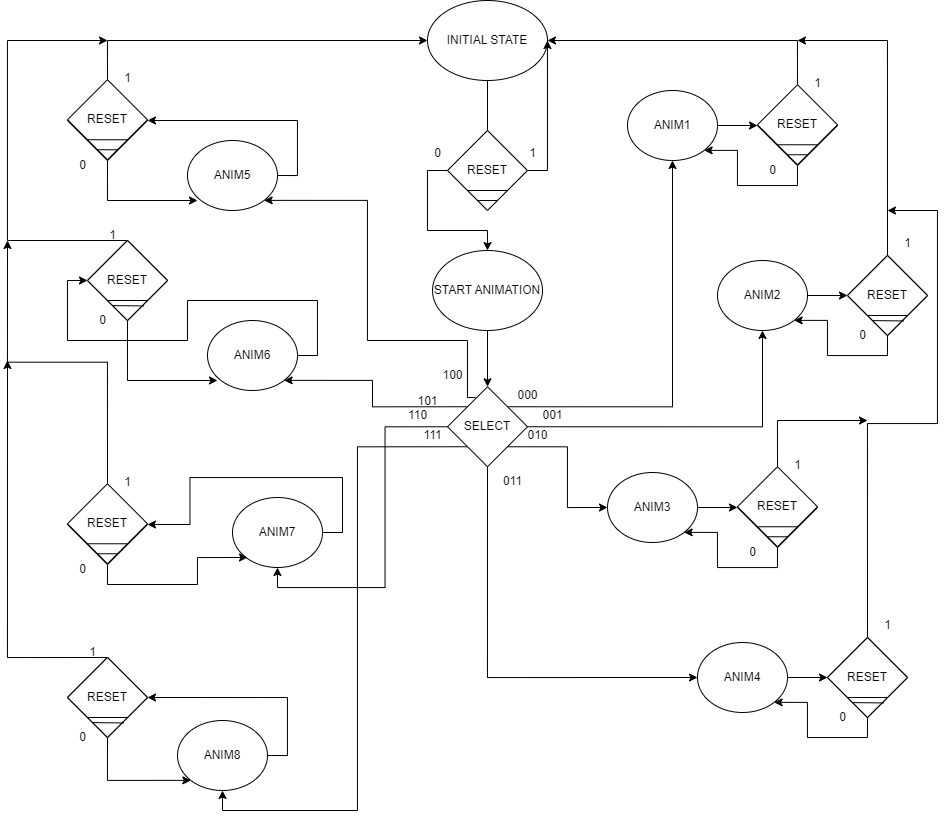


Outputs:

* Segment-the binary codification of a symbol that appears on the SSD at a given moment
* Select-the binary representation of the anods(when anods(n)=1 the n+1’th display will be lit since we start from anods(0))
  1. **CONTROL AND EXECUTION UNIT**

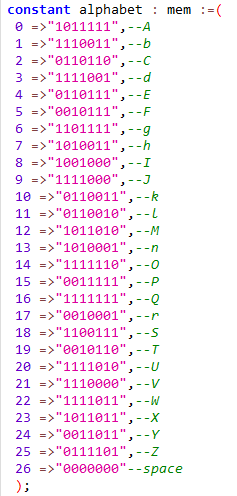
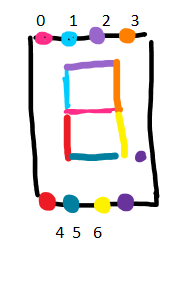
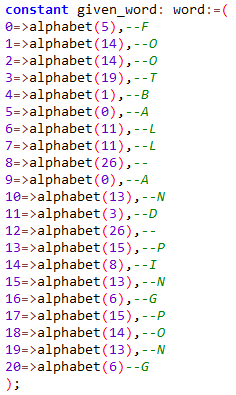


The block diagram was made so it would correspond to the state diagram



The memories used for the project are:

The maximal alphabet encoded The given\_word memory(uses the alphabet memory)

We declare them in the main entity’s architecture with their types as:

**type** mem **is** **array** (26 **downto** 0) **of** **std\_logic\_vector**(6 **downto** 0);  
**type** word **is** **array** (20 **downto** 0) **of** **std\_logic\_vector**(6 **downto** 0);

We use two counters to divide the frequency:

The first is used for the actual display and matches the refreshing rate of the images. The human’s eye has 16ms image refreshing rate.

So we use a 16 bit counter (15 downto 0) and choose an approximating value.

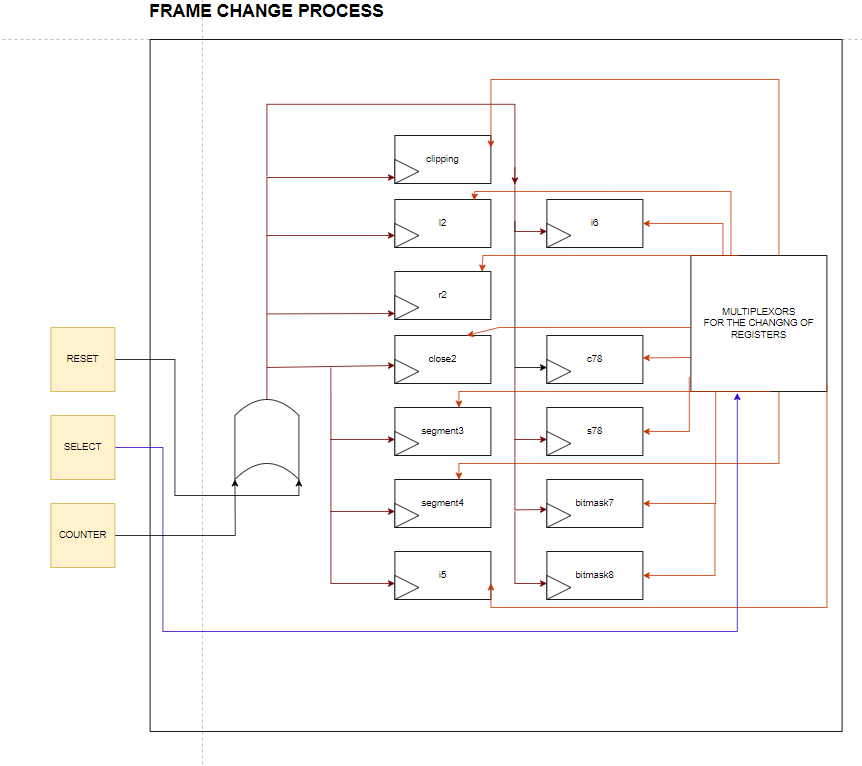
**entity** counter15 **is**  
**port**(  
clk :**in** **std\_logic**;  
o: **out** **std\_logic**);   
**end** counter15;  
**Architecture** count2 **of** counter15 **is**  
**signal** c: **std\_logic\_vector**(15 **downto** 0):=(**others**=>'0');  
**begin**  
  
counting : **process**(clk)  
**begin**  
**if**(clk='1' **and** clk'event) then  
 c<=c+1;  
**if**(c="1111000000000000") **then**  
o<='1';  
**end** **if**;  
**if**(c="1111000000000001") **then**  
c<=(**others**=>'0');  
o<='0';  
**end** **if**;  
**end** **if**;  
  
**end** **process** counting;  
  
**end** count2;

When the counter fills up to that value the signal count2 is transmitted to the for\_display process.

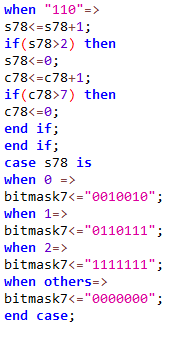
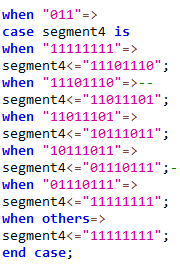
The second counter is used to toggle the changing of frames. I use a big value for the counter approximatively 5000000, half of the actual frequency of the clock. So the execution of the program will be delayed with a half of a second, that is, a new frame will appear half of a second after the previous one.

**entity** counter25 **is**  
**port**(  
clk :**in** **std\_logic**;  
o: **out** **std\_logic**);   
**end** counter25;  
**Architecture** count1 **of** counter25 **is**  
**signal** c: **std\_logic\_vector**(25 **downto** 0):=(**others**=>'0');  
**begin**  
  
counting : **process**(clk)  
**begin**  
**if**(clk='1' **and** clk'event) then  
c<=c+1;  
**if**(c="10111110101111000010111111") **then**  
o<='1';  
**end** **if**;  
**if**(c="10111110101111000011000000") **then**  
c<=(**others**=>'0');  
o<='0';  
**end** **if**;  
**end** **if**;  
  
**end** **process** counting;  
  
**end** count1;

The output of the counter is transmitted to the frame changing process.



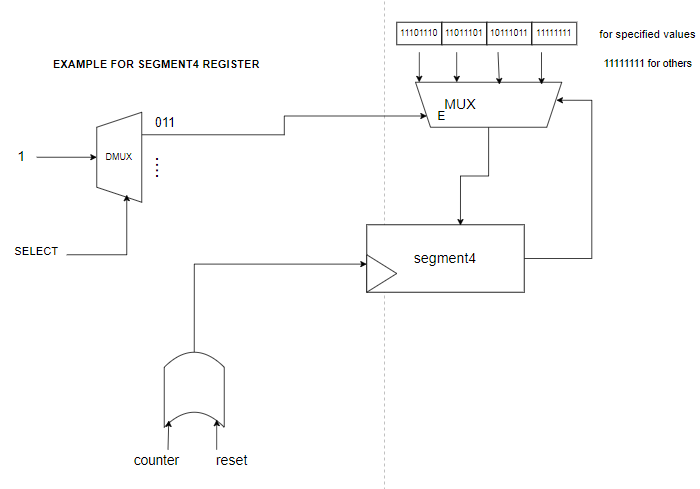
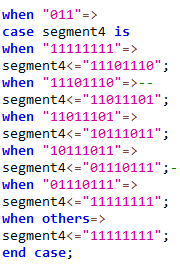
Every register is changed based on the current value so, in addition to the drawing, there is a wire that goes from each register to the **Multiplexors block** . The current value will be used as the select of the future value multiplexor. The **SELECT** acts like an enable for the multiplexors because, depending on the **SELECT** only certain registers change:

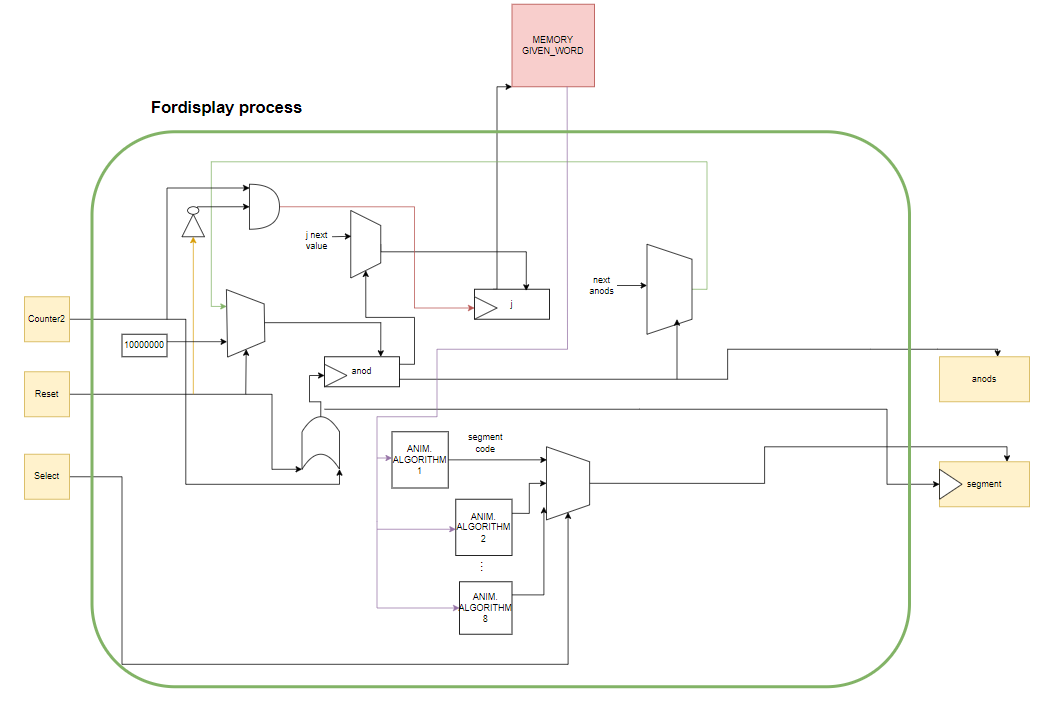
Here the SELECT is “011” and only segment4 changes

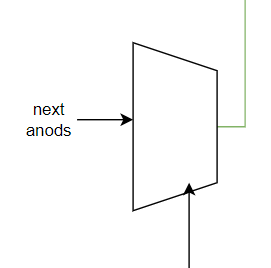
Here the SELECT is “110” and only s78, c78 and bitmask8 change

COMPLETE DESIGN FOR THE SEGMENT4 REGISTER



The same concept is applied for all registers in the frame change process





**if**(anod(0)='1') **then**  
 anod(0):='0';   
 anod(1):='1';  
 **elsif** (anod(1)='1') **then**   
 anod(1):='0';  
 anod(2):='1';  
 **elsif** (anod(2)='1') **then**   
 anod(2):='0';  
 anod(3):='1';  
 **elsif** (anod(3)='1') **then**   
 anod(3):='0';  
 anod(4):='1';  
 **elsif** (anod(4)='1') **then**   
 anod(4):='0';   
 anod(5):='1';  
 **elsif** (anod(5)='1') **then**   
 anod(5):='0';  
 anod(6):='1';  
 **elsif** (anod(6)='1') **then**   
 anod(6):='0';  
 anod(7):='1';  
 **elsif** (anod(7)='1') **then**   
 anod(7):='0';  
 anod(0):='1';   
 **end** **if**;  
 anods<=anod;

This multiplexer does the same operations as a shift register would, the anod goes from 0 to 7 and than this process is repeated

**case** anod **is**  
**when** "00000001"=>  
j:=1;  
**when** "00000010"=>  
j:=2;  
**when** "00000100"=>  
j:=3;  
**when** "00001000"=>  
j:=4;  
**when** "00010000"=>  
j:=5;  
**when** "00100000"=>  
j:=6;  
**when** "01000000"=>  
j:=7;  
**when** "10000000"=>  
j:=0;  
**when** **others**=>  
j:=1;  
**end** **case**;

The register **j** represents the address in memory to be prepared from which the SSD encoding is chosen. It will take the value of the index of the next active anod(if the anod(0) is lit, anod(1) follows so j=1).

* 1. **ANIMATIONS**

The left blocks of code are from the **fordisplay** process and the right from **framechanging** process.

* Animation 1 -Clipping

This animation consists of two frames. The word on the 8 SSDs will be displayed and then disappear.

We will use a variable **clipping** that will take either 1 or 0 and will change its value in the frame changing process. When **clipping** is 0 the word will show and when is 1 will disappear.

clipping<=clipping+1;  
**if**(clipping=2) **then**  
clipping<=0;  
**end** **if**;

**if**(clipping=0) **then**   
segment<=given\_word(j);  
**else**  
segment<=(**others**=>'0');  
**end** **if**;

* Animation 2 –Filling from the outside to the inside and backwards

We will use two variables **l2** and **r2** that begin with the values 0 and 7 and at each frame change their values either approach or move away with 2 positions. The variable **close2** will indicate which way their values change.

**if**(j<=l2 **or** j>=r2) **then**  
segment<=given\_word(j);  
**else**  
segment<=(**others**=>'0');  
**end** **if**;

**if**(close2=1) **then**  
l2<=l2+1;  
r2<=r2-1;  
**if**(l2>r2) **then**  
l2<=l2-2;  
r2<=r2+2;  
close2<=0;  
**end** **if**;  
  
**elsif**(close2=0) **then**  
l2<=l2-1;  
r2<=r2+1;  
**if**(l2<0 **and** r2>7) **then**  
l2<=l2+1;  
r2<=r2-1;  
close2<=1;  
**end** **if**;  
  
**end** **if**;

* Animation 3 – Letters appear in a particular order

We will use the variable **segment3** of size 8 and it is used in this way:

If the **segment3(x)** is 1 the letter will be displayed and if 0 it won’t.

**case** segment3 **is**  
**when** "01000000"=>  
segment3<="01000010";  
**when** "01000010"=>  
segment3<="01001010";  
**when** "01001010"=>  
segment3<="01011010";  
**when** "01011010"=>  
segment3<="11011010";  
**when** "11011010"=>  
segment3<="11011110";  
**when** "11011110"=>  
segment3<="11111110";  
**when** "11111110"=>  
segment3<="11111111";  
**when** "11111111"=>  
segment3<="00000000";  
**when** "00000000"=>  
segment3<="01000000";  
**when** **others**=>  
segment3<="00000000";  
**end** **case**;

**if**(segment3(j)='1') **then**  
segment<=given\_word(j);  
**else**  
segment<=(**others**=>'0');  
**end** **if**;

* Animation 4 – Two letters disappear from the word at a time

To be explicit the first and fifth letter, the second and the sixth , …, the fourth and the last. The difference between them is 4. We will use just like in the previous animation a variable **segment4.**

**case** segment4 **is**  
**when** "11111111"=>  
segment4<="11101110";  
**when** "11101110"=>*--*  
segment4<="11011101";  
**when** "11011101"=>  
segment4<="10111011";  
**when** "10111011"=>  
segment4<="01110111";*--*  
**when** "01110111"=>  
segment4<="11111111";  
**when** **others**=>  
segment4<="11111111";  
**end** **case**;

**if**(segment4(j)='1') **then**  
segment<=given\_word(j);  
**else**  
segment<=(**others**=>'0');  
**end** **if**;

* Animation 5 – Right to left display

We will have the variables: **j56**- which is the length of the word-1, so the index of the last symbol and **i5=-7** and it’ s value increments every frame until it reaches **j56**. If the **i5+x** value is between 0 and **j56** the symbol at that position will be displayed.

i5<=i5+1;  
**if**(i5>j56) **then**  
i5<=-7;  
**end** **if**;

j:=j+i5;  
**if**(j>=0 **and** j<=j56) **then**  
segment<=given\_word(j);

**else**  
segment<="0000000";  
**end** **if**;

* Animation 6- Left to right display

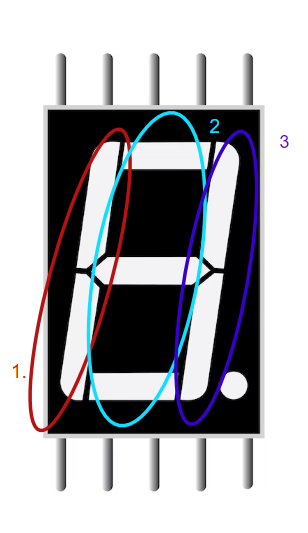
We use the same ideea, the only difference is that **i6** decrements at each frame change

i6<=i6-1;  
**if**(i6<-7) **then**  
i6<=j56;  
**end** **if**;

j:=j+i6;  
**if**(j<=j56 **and** j>=0) **then**  
segment<=given\_word(j);  
**else**  
segment<="0000000";  
**end** **if**;

* Animation 7- Flowing from left to right

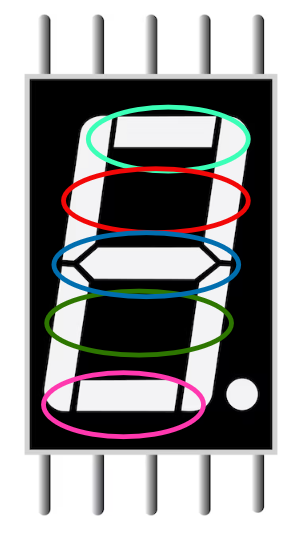
We will use a bitmask that can take 3 values selected using a multiplexer for the three states of the flow animation and use the **and operation** on the bits. Variable **s78** tells which sides of the segment will be turned on(just like in the figure) and **c78** tells what letter we have reached in our animation.



s78<=s78+1;  
**if**(s78>2) **then**  
s78<=0;  
c78<=c78+1;  
**if**(c78>7) **then**  
c78<=0;  
**end** **if**;  
**end** **if**;  
**case** s78 **is**  
**when** 0 =>  
bitmask7<="0010010";  
**when** 1=>  
bitmask7<="0110111";  
**when** 2=>  
bitmask7<="1111111";  
**when** **others**=>  
bitmask7<="0000000";  
**end** **case**;

**if**(j<c78) **then**  
 segment<=given\_word(j);  
 **elsif** (j=c78) **then**  
 segment<=given\_word(j) **and** bitmask7;  
 **else**   
 segment<="0000000";  
 **end** **if**;

* Animation 8- Flowing from top to bottom

The principle is the same as the previous one, the difference is that we have 5 states( just like in the picture).

**if**(j<c78) **then**  
 segment<=given\_word(j);  
 **elsif** (j=c78) **then**  
 segment<=given\_word(j) **and** bitmask8;  
 **else**   
 segment<="0000000";  
 **end** **if**;

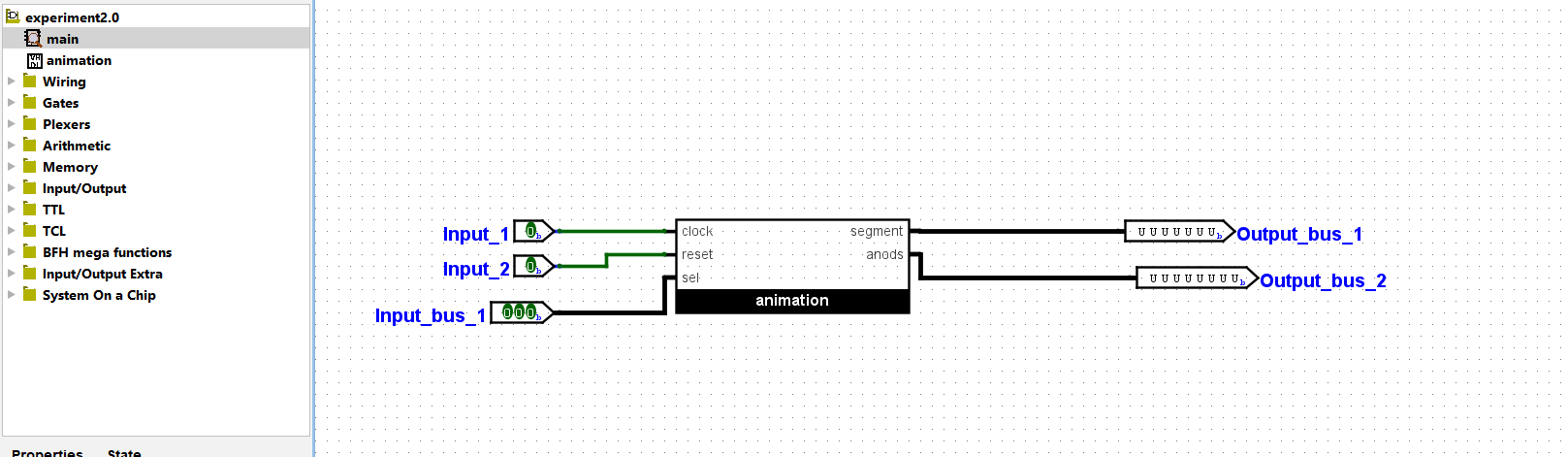
s78<=s78+1;  
**if**(s78>4) **then**  
s78<=0;  
c78<=c78+1;  
**if**(c78>7) **then**  
c78<=0;  
**end** **if**;  
**end** **if**;  
 **case** s78 **is**  
 **when** 0 =>  
 bitmask8<="0000100";  
 **when** 1=>  
 bitmask8<="0001110";  
 **when** 2=>  
 bitmask8<="0001111";  
 **when** 3=>  
 bitmask8<="1011111";  
 **when** 4=>  
 bitmask8<="1111111";  
 **when** **others**=>  
 bitmask8<="0000000";   
 **end** **case**;

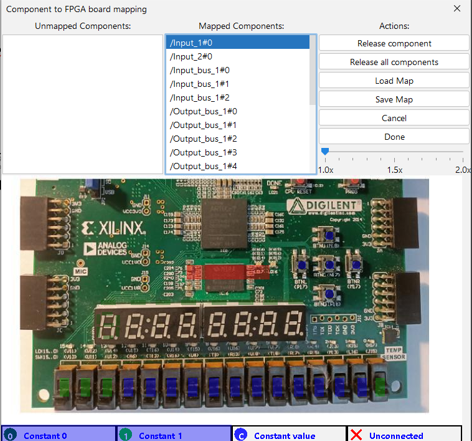
* 1. **How the project functions on the FPGA board**



I used the application LOGISIM EVOLUTION in order to put the project on the board.

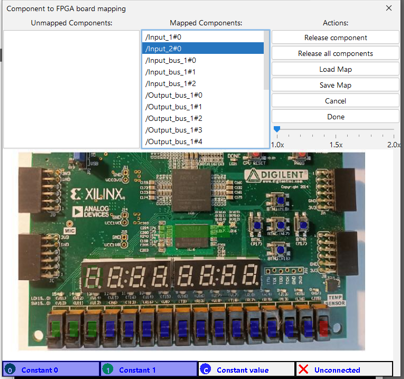
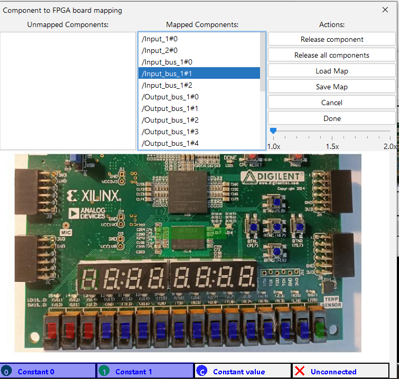
What the main window contains with the project anotations:





Input\_1 represents the clock signal and

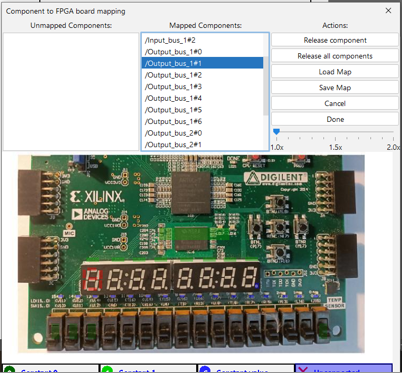
is mapped on the red area where the FPGA in-built clock resides.

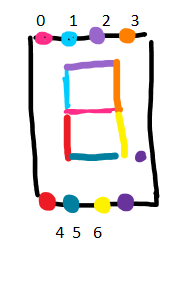
 

Input\_2 represents the reset signal and it is used in order to reset the animations or change the animation. It is mapped on the red coloured switch

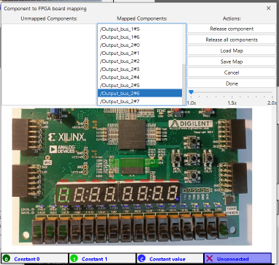
Input\_bus\_1 represents the select signal which is 3 bits-long and I mapped them on the red switches in the following way from left to right:

Input\_bus\_1#2 - Input\_bus\_1#1 - Input\_bus\_1#0





Output\_bus\_1 represents the binary encoding signal on the cathode of the symbol at a given time and I used the encoding mapping given in the following figure. So, for example, the **Output\_bus\_1#0** will be mapped on the pink line, **Output\_bus\_1#1** on the blue line and so on.



Output\_bus\_2 represents the array of the anodes

And it is mapped on the red areas in the following way from left to right:

Output\_bus\_2#0, Output\_bus\_2#1, Output\_bus\_2#2, … , Output\_bus\_2#7

Functioning regime

The moment the project will be loaded on the board the animation will be playing. **In order to change the animation you need to flip the reset switch and only then change the select switches and flip the reset again in order to avoid any errors**.

* 1. **References**

<https://sites.google.com/site/learnvhdl/home/hour-15-state-machine-contd>

<https://digilent.com/reference/programmable-logic/nexys-4-ddr/reference-manual>

<https://surf-vhdl.com/vhdl-array/>

<https://cse.usf.edu/~haozheng/teach/cda4253/doc/vhdl-stmt.pdf>