CLUJ-NAPOCA

UNIVERSITATEA
TEHNICĂ
CLUJ-NAPOCA

ADVERTISEMENT

Digital System Design

Freshman Year - Computer Science, Group 30413

Name：Coroian Tudor

Advisor：Lorena Daian

Due Date：23.05.2019

Table of contents

1. Specification of the project

1.1 Given requirements

Design an advertisement with multiple animations. Use the 7-Segments BCD decoder. The text will be composed of symbols of a available alphabet. The advertisement will have multiple functioning regimes (min. 4) which will be selected by the user from the switches of the FPGA board. You will use the embedded quartz oscillator (whose frequency will be divided). Examples of functioning regimes: sliding of the text from left to right, blinking, show letter by letter, etc.

Because of the fact that on a 7-Segments BCD decoder one cannot represent all the letters, a maximal alphabet should be created and the messages will be composed of the letters of that alphabet. The message will be stored in a ROM memory to be easily changed. The project will be done by one student.

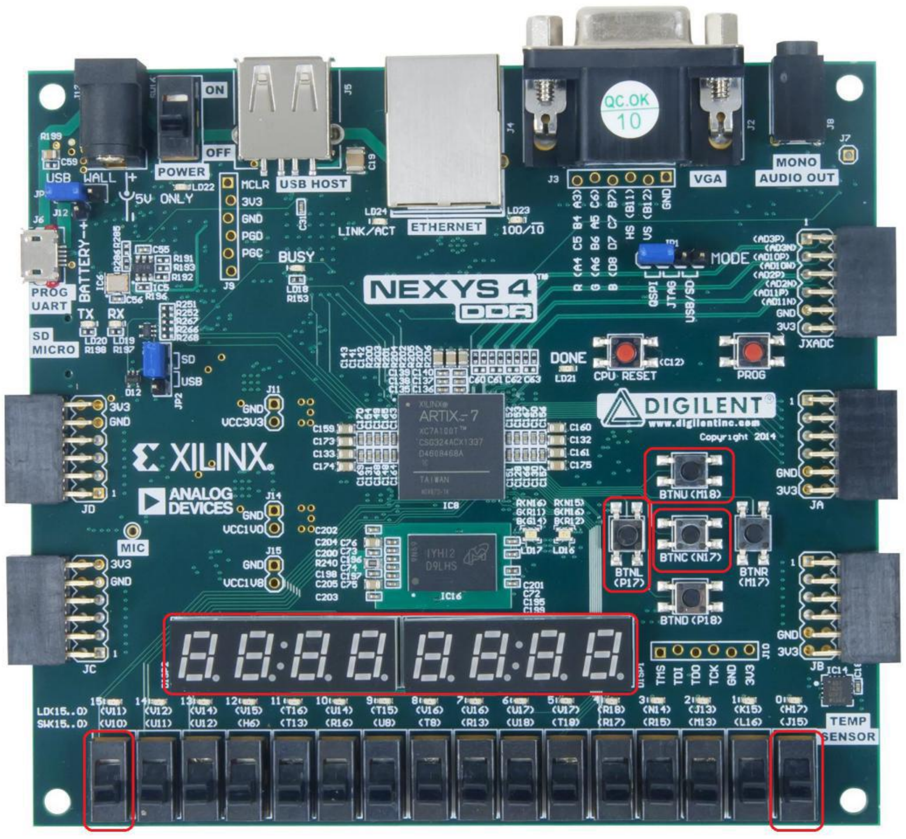
1.2 The FPGA board

For this project, I chose to use the Nexys 4 board, because of the wider 7-Segment Display, which allows for longer messages to be introduces.

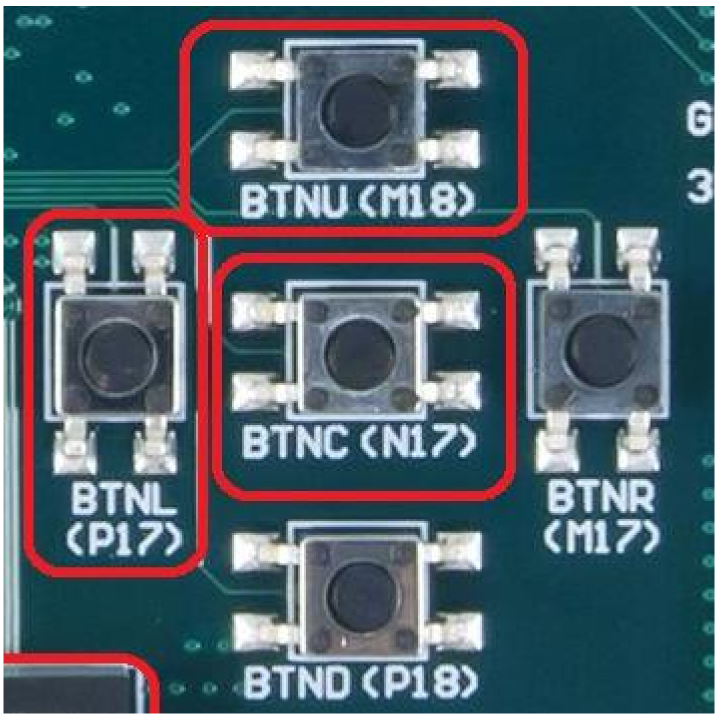
The Nexys 4 board is a complete, ready-to-use digital circuit development platform based on the latest Artix-7™ Field Programmable Gate Array (FPGA) from Xilinx. With its large, high-capacity FPGA (Xilinx part number XC7A100T-1CSG324C), generous external memories, and collection of USB, Ethernet, and other ports, the Nexys 4 can host designs ranging from introductory combinational circuits to powerful embedded processors.

Several built-in peripherals, including an accelerometer, temperature sensor, MEMs digital microphone, a speaker amplifier, and a lot of I/O devices allow the Nexys 4 to be used for a wide range of designs without needing any other components.

Some specifications of the FPGA board will be presented in the next section, along with the used pins names and the used input buttons and switches.



In the previous picture, the used components where highlighted. Next, some more detailed images will show the name and function of each individual component.

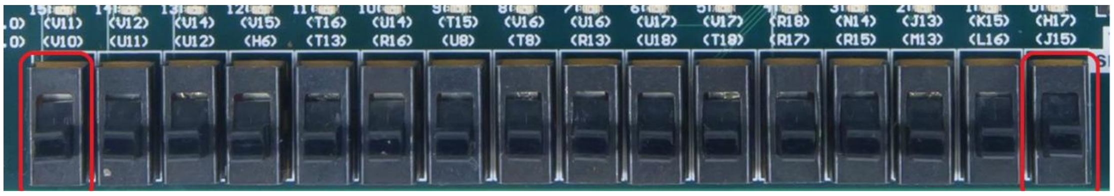


BTNU - Selects the next letter in the alphabet (A to Z, plus an extra space)

BTNL - Move to the left display (relative to the current display)

BTNC - Select the next animation (out of 4)

V10 - Load a new word
J15 - Safety button (cannot select the next letter or the left display if the switch is off)



7-Segments Display - Display the message with its animations

Furthermore, the embedded quartz generator of the FPGA board generates a clock pulse at the frequency of 100 MHz.

2. Initial considerations

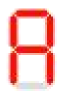
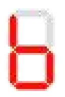
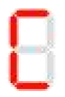
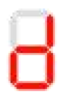
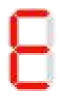
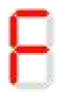
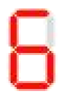
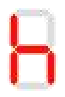
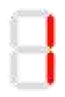
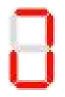
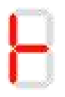
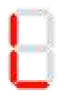
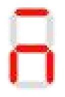
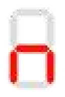
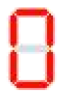
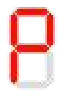
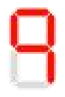
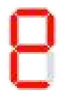
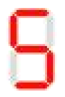
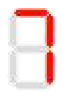
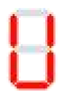
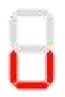
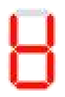
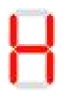
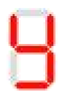
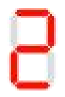
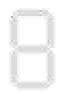
2.1 Personal assessments

Out of interest for this project, I decided not to store a message in a ROM memory, but instead to allow the user to input his own messaged in a simple way, without the need to memorize 27 individual codes, one for each letter. Furthermore, the animations work for any message that is introduced in the system.

Due to this fact, the task was more complicated and a more elaborate solution was needed. However, the system's scheme is easy to understand and will be presented in the next chapter.

2.2 The used alphabet

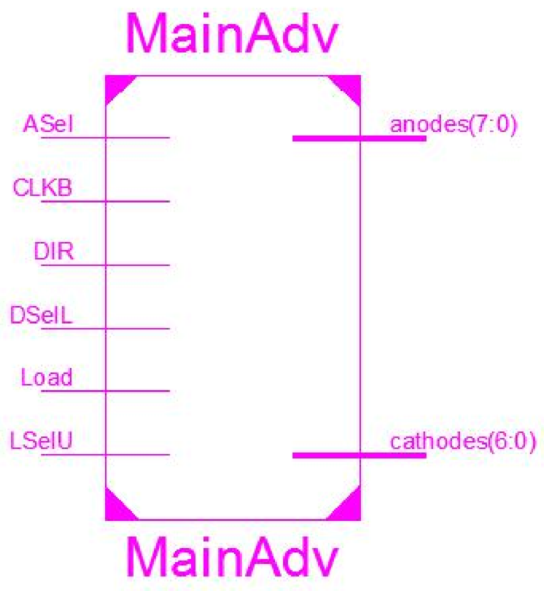
Considering that not all letters can be represented on the 7-Segment Display, I designed a model for each letter of the English alphabet. In the next pictures, the letters A to Z will be represented in order. Also, a model for the space character was designed to allow the user to introduce either one 8 letter long word or two smaller word separated by a blank display.

								
A	B	C	D	E	F	G	H	I
								
J	K	L	M	N	O	P	Q	R
								
S	T	U	V	W	X	Y	Z	Space

3. Overview of the project

3.1 Black box of the design

In the next picture, a simple black box of the design will be presented along with the meaning of the pins.



ASel - Animation Select

CLKB - The clock from the FPGA board at 100 MHz

DIR - Safety switch

DSelL - Display Select Left (move to the display to the left)

Load - Switch on to load a new word; switch of to activate the animations

LSelU - Letter Select Up (select the next letter in the alphabet)

anodes - Anodes of the 7-Segments Display

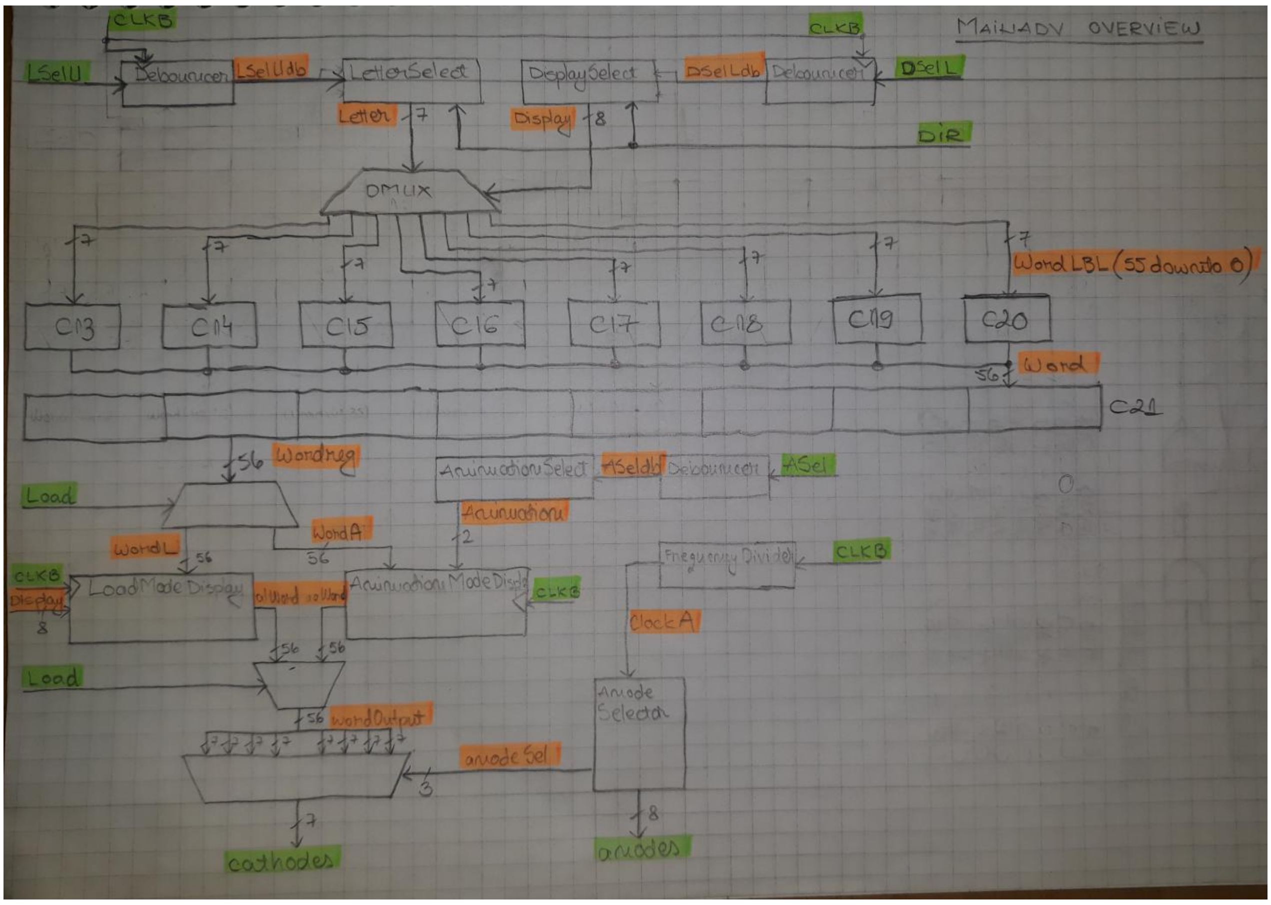
cathodes - Cathodes of the 7-Segments Display

In the next picture, a snippet of the code representing the entity above will be shown.

```
entity MainAdv is
   port ( LSelU : in std_logic; --Letter Select Up
          DSelL : in std_logic; --Display Select Left
          --Actually selects the register in which the letter for the specified display will be stored
          ASel  : in std_logic; --Animation Select
          DIR   : in std_logic;
          CLKB  : in std_logic; --Board clock
          Load  : in std_logic; --Load a word letter by letter
          cathodes : out std_logic_vector (6 downto 0); --BCD decoder segments
          anodes   : out std_logic_vector (7 downto 0)); --BCD decoder anodes for Nexys4
end MainAdv;
```

3.2 Block diagram of the main component

The main component of the whole project is MainAdv (which was previously presented). Next, a picture containing the block diagram of this component is attached. This component links together several more components which will be presented in the next chapters.



Note: The inputs and outputs are highlighted in green and the internal signals are highlighted in orange.

The components used and presented in the picture are :

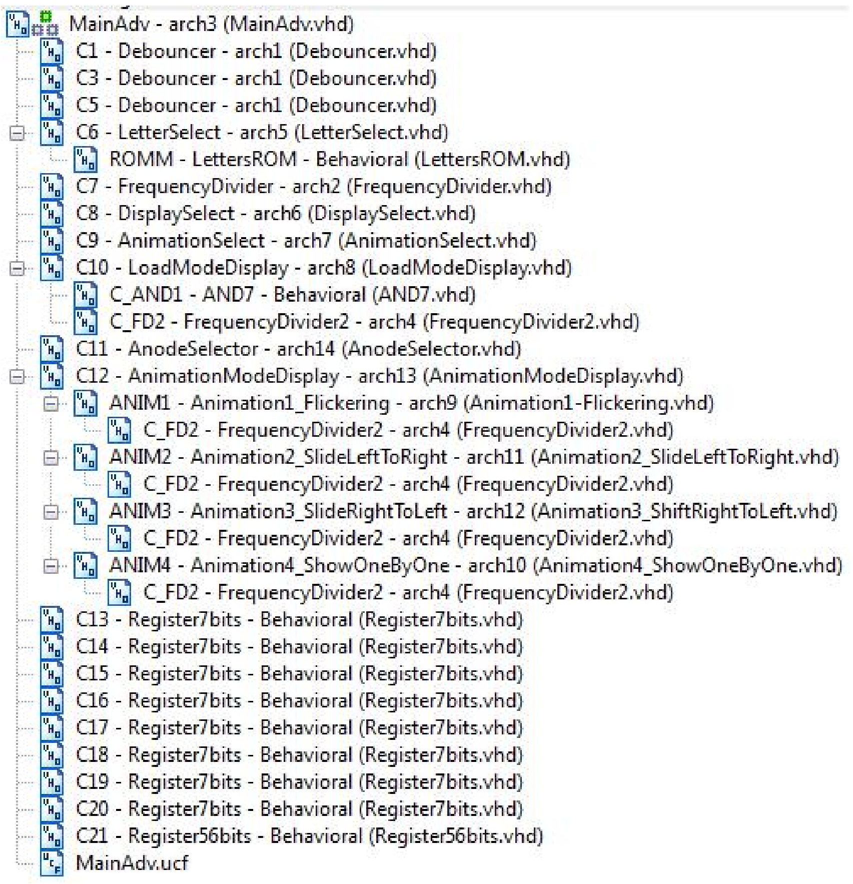
- ❖ Debouncer
- ❖ LetterSelect
- ❖ DisplaySelect
- ❖ Register7bits (C13-C20)
- ❖ Register56bits (C21)
- ❖ AnimationSelect
- ❖ FrequencyDivider
- ❖ LoadModeDisplay
- ❖ AnimationModeDisplay
- ❖ AnodeSelector

4. The components

4.1 General overview

In what regards this project, I went for a structural description for most of the components, but also including a behavioral description when it felt appropriate.

For a better understanding of the used components and also a better view of all the sub-components, next a picture of the component tree of the project is attached.

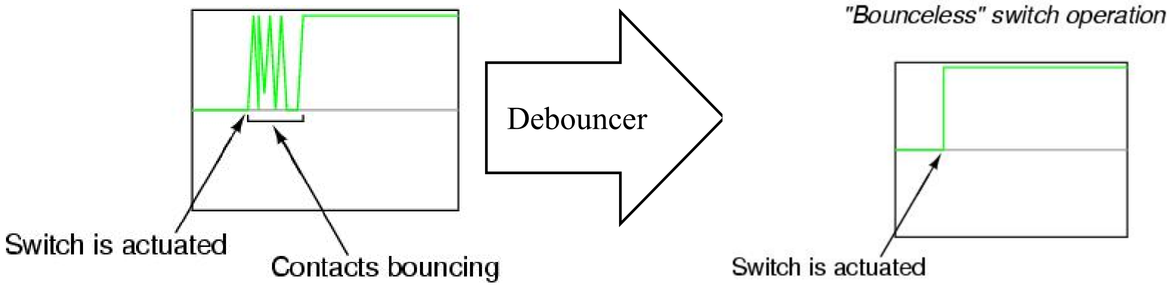


```
MainAdv - arch3 (MainAdv.vhd)
    C1 - Debouncer - arch1 (Debouncer.vhd)
    C3 - Debouncer - arch1 (Debouncer.vhd)
    C5 - Debouncer - arch1 (Debouncer.vhd)
    C6 - LetterSelect - arch5 (LetterSelect.vhd)
        ROMM - LettersROM - Behavioral (LettersROM.vhd)
    C7 - FrequencyDivider - arch2 (FrequencyDivider.vhd)
    C8 - DisplaySelect - arch6 (DisplaySelect.vhd)
    C9 - AnimationSelect - arch7 (AnimationSelect.vhd)
    C10 - LoadModeDisplay - arch8 (LoadModeDisplay.vhd)
        C_AND1 - AND7 - Behavioral (AND7.vhd)
        C_FD2 - FrequencyDivider2 - arch4 (FrequencyDivider2.vhd)
    C11 - AnodeSelector - arch14 (AnodeSelector.vhd)
    C12 - AnimationModeDisplay - arch13 (AnimationModeDisplay.vhd)
        ANIM1 - Animation1_Flickering - arch9 (Animation1-Flickering.vhd)
            C_FD2 - FrequencyDivider2 - arch4 (FrequencyDivider2.vhd)
        ANIM2 - Animation2_SlideLeftToRight - arch11 (Animation2_SlideLeftToRight.vhd)
            C_FD2 - FrequencyDivider2 - arch4 (FrequencyDivider2.vhd)
        ANIM3 - Animation3_SlideRightToLeft - arch12 (Animation3_ShiftRightToLeft.vhd)
            C_FD2 - FrequencyDivider2 - arch4 (FrequencyDivider2.vhd)
        ANIM4 - Animation4_ShowOneByOne - arch10 (Animation4_ShowOneByOne.vhd)
            C_FD2 - FrequencyDivider2 - arch4 (FrequencyDivider2.vhd)
    C13 - Register7bits - Behavioral (Register7bits.vhd)
    C14 - Register7bits - Behavioral (Register7bits.vhd)
    C15 - Register7bits - Behavioral (Register7bits.vhd)
    C16 - Register7bits - Behavioral (Register7bits.vhd)
    C17 - Register7bits - Behavioral (Register7bits.vhd)
    C18 - Register7bits - Behavioral (Register7bits.vhd)
    C19 - Register7bits - Behavioral (Register7bits.vhd)
    C20 - Register7bits - Behavioral (Register7bits.vhd)
    C21 - Register56bits - Behavioral (Register56bits.vhd)
    MainAdv.ucf
```

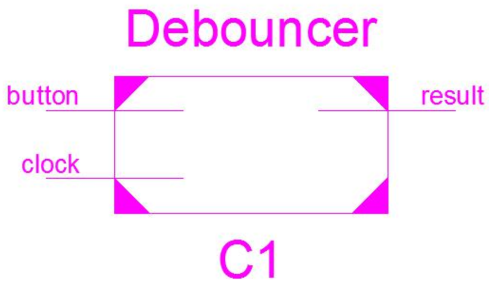
4.2 Debouncer

The Debouncer is used three times in the project. The labels associated with each instantiation are, in order, C1, C3 and C5.

Knowing that due to mechanical issues the button of the FPGA board tends to bounce, giving off a unusable signal, designing a component that solves this problem was necessary.



In what follows, an image of the component will be attached and an explanation for the way it works will be presented.



button -	The signal which needs to be debounced
clock -	Board clock at f=100MHz
result -	The debounced signal

How does it work? Two flip-flops are connected in sequence, such that the output of the first one is connected to the input of the second one. The signal from the button is connected to the first flip-flop. The output signals from the two flip-flops are passed through a XOR gate which reset a counter on 20 bits if one of the two values of the flip-flops changes during the counted amount of time. If no changes occur, the signal is passed on via the result output port.

The formula to determine how many bits the counter should have is:

$$P = \frac{2^N}{f}$$

where ***P*** is the period of stability (which should be around 10 ms), ***N*** is the number of bits in the counter and ***f*** is the frequency of the clock (which in the case of the Nexys4 FPGA board is 100 MHz). After some computations it is clear that ***N*** should equal 20.

Having three buttons used in the design, this component is instantiated as follows:

- C1 for the LSelU button
- C3 for the DSelL button
- C5 for the ASel button

4.3 Frequency Divider

The Frequency Divider is widely used throughout the project. It is needed to reduce the frequency of the FPGA board clock to two different frequencies, each used in separate modules of the design.

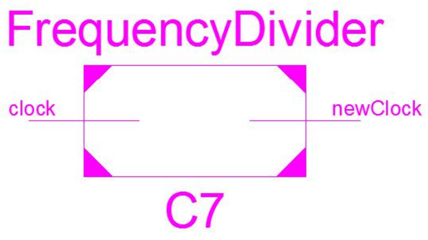
This component is instantiated several times, as follows:

- C7 in the MainAdv component, to reduce the frequency down to 760 Hz

- C_FD2 in LoadModeDisplay and in each Animation component, to reduce the frequency down to 1 Hz

The reason for these frequencies being needed will be explained when discussing the next components.

Bellow, an image of the component will be attached and an explanation for the way it works will be presented.



clock	- Input clock at a certain frequency
newClock	- Output clock at the desired frequency

How does it work? Using a counter on a certain number of bits (depending on the frequency desired) this component divides or "slows down" the clock given as input. Because the size of the counter is declared as a generic value, this component is versatile in many situation, for clocks of any frequency.

For a better understanding, a detailed snippet of the architecture of this divider is presented in Annex1.

4.4 Letter Select

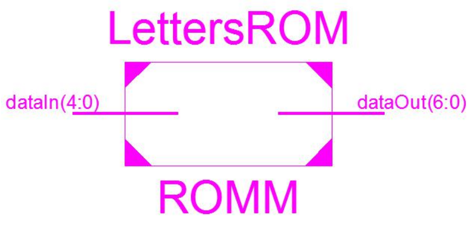
This component has a sub-component called LettersROM which will be presented first to allow for a better description of the big component.

4.4.1. Letters ROM

Due to the fact that I chose to implement a design where the user can introduce any word, with the only limitation being the length, a new problem arises. How can I send some letters to the outputs, without the need of a complicated combinational component that would decipher the input and respond accordingly? The answer is simple. Design a ROM memory to store the code for each letter.

At addresses going from 0 to 26, I stored the values of the cathodes corresponding to each letters as presented in Chapter 2.

Bellow, an image of the component is attached, followed by the truth table of this component.



dataIn - Addresses

dataOut - The values of the cathodes

Letter	dataIn					dataOut						
	4	3	2	1	0	a	b	c	d	e	f	g
A	0	0	0	0	0	0	0	0	1	0	0	0
B	0	0	0	0	1	1	1	0	0	0	0	0
C	0	0	0	1	0	0	1	1	0	0	0	1
D	0	0	0	1	1	1	0	0	0	0	1	0
E	0	0	1	0	0	0	1	1	0	0	0	0
F	0	0	1	0	1	0	1	1	1	0	0	0
G	0	0	1	1	0	0	1	0	0	0	0	0
H	0	0	1	1	1	1	1	0	1	0	0	0
I	0	1	0	0	0	1	0	0	1	1	1	1
J	0	1	0	0	1	0	0	0	0	0	1	1
K	0	1	0	1	0	1	1	1	1	0	0	0
L	0	1	0	1	1	1	1	1	0	0	0	1
M	0	1	1	0	0	0	1	0	1	0	1	0
N	0	1	1	0	1	1	1	0	1	0	1	0
O	0	1	1	1	0	0	0	0	0	0	0	1
P	0	1	1	1	1	0	0	1	1	0	0	0
Q	1	0	0	0	0	0	0	0	1	1	0	0
R	1	0	0	0	1	0	0	1	0	0	0	0
S	1	0	0	1	0	0	1	0	0	1	0	0
T	1	0	0	1	1	0	0	0	1	1	1	1

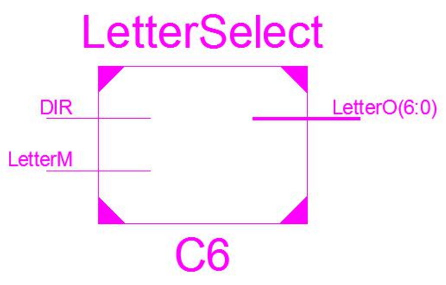
U	1	0	1	0	0	1	0	0	0	0	0	1
V	1	0	1	0	1	1	0	0	0	0	1	1
W	1	0	1	1	0	1	1	0	0	0	0	0
X	1	0	1	1	1	1	0	0	1	0	0	0
Y	1	1	0	0	0	1	0	0	0	1	0	0
Z	1	1	0	0	1	0	0	1	0	0	1	0
Space	1	1	0	1	0	1	1	1	1	1	1	1

To be noted that the a-g cathodes are active-low.

Although this table seems quite hard to follow, the user does not need to remember the codes for each letter, or to even have a table of the codes nearby. By pressing a button, one will cycle through all the letters in alphabetical order. This aspect will be presented in the description of the big component (LetterSelect).

4.4.2 Letter Select

As shown before, this is the component that encapsulates the previously described component (LettersROM). Bellow is attached a picture of this component.



DIR - Safety switch

LetterM - Letter Move (signal from the button LSelU)

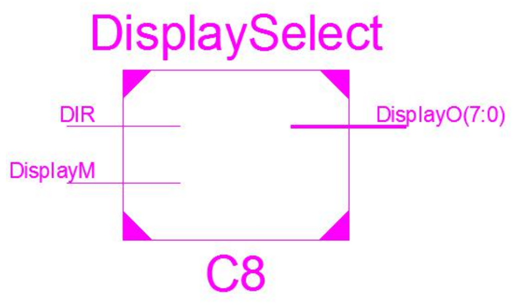
LetterO - Letter Output

For a better understanding, a detailed snippet of the architecture of this component is presented in Annex2. Also, to be reminded that in order for this component to select a new letter, the DIR switch must be turned on.

4.5 Display Select

Similar to LetterSelect, this component cycles between the Displays (better said the registers attached to each display). Here, again the DIR switch must be turned on, in order for the component to cycle through the displays.

Bellow, a picture of the actual component is attached.



DIR - Safety switch

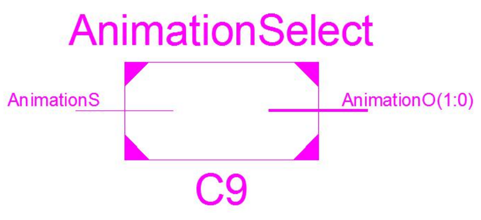
DisplayM - Display Move (signal from the button DSelL)

DisplayO - Display Output

4.6 Animation Select

Similar to the two previously described components, AnimationSelect cycles through all the animations, but only if the Load switch is turned off.

Bellow, a picture of the actual component is attached.



AnimationS - Animation Select (signal from the button ASel)

AnimationO - Animation Output

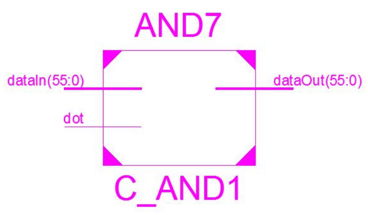
To be noted that the output of this component will be used as an input for the AnimationModeDisplay component.

4.7 Load Mode Display

When describing this component, it is first important to mention the two components that reside within it: a frequency divider, which was presented earlier and an blinking component, AND7.

4.7.1 AND7

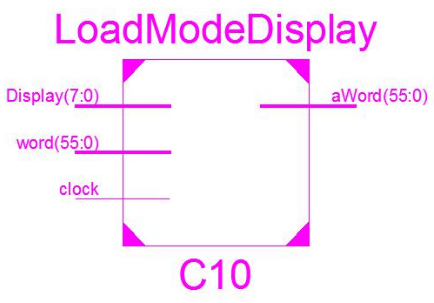
Bellow, an image of this component is attached.



The behaviour of this component is easy to describe. It takes as inputs a vector on 56 bits (which represent the maximum-lenght word that can be represented on the 8 displays) and a dot signal (which is the clock signal, with the frequency divided down to only 1 Hz). The result of this component is a blinking vector on 56 bits, which appears and disappears approximately once every second.

4.7.2 LoadModeDisplay

Bellow, an image of this component is attached.



Display - 8bit vector to select the current display

word - 56 bit vector (the loaded word)

clock - The board clock at f = 100 MHz

aWord - The word with the load animation

What this component does will be presented in the usage instruction chapter.

However, for a better understanding, a detailed snippet of the architecture of this component is presented in Annex3 and Annex4.

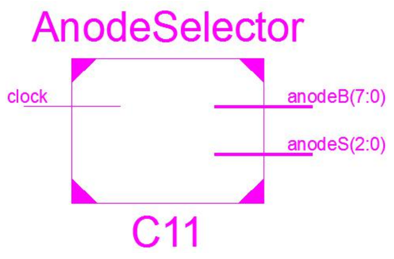
4.8 Anode Selector

On a closer inspection of the FPGA board, one can notice that there are only 7 (8) inputs for the 7-Segment BCD Decoder, instead of the expected 56 (64). Due to this fact, there arises a problem when one tries to show more than one character on the displays.

To overcome this disadvantage, I had to design a component which will cycle through the anodes of the board and also through the groups of cathodes which represent the letter. If this cycle happen at a frequency between 1 kHz and 60 Hz, the opening and closing of the segments of the displays becomes so short that for the human eye it is impossible to notice, thus appearing as a continuous image.

To reach that frequency, I used a frequency divider which was presented earlier (the one that divides the frequency of the board clock at 760 Hz).

Bellow, an image of this component is attached.



clock - Board clock at f = 100 MHz

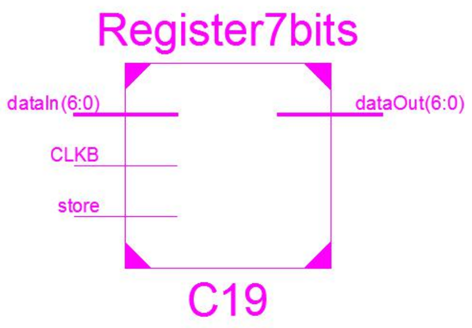
anodeB - Anodes of the board

anodeS - Signal used to cycle between the groups of cathodes

4.9 Register7bits and Register56bits

Although they are two different component, the two registers were designed in the same way. A full description of the design can be seen in Annex5.

Bellow, an image of this component is attached.



The register will store whatever is on the dataIn input if the store input is active, and will output the contents stored, otherwise. Both operation happen on the rising edge of the FPGA board clock.

Note that the 7 bit register is used to store each letter individually, while the 56 bit one store the whole word.

4.10 Animation Mode Display

This component contains four components, each responsible for one of the four animations. Through the press of one button, the user can cycle between the four animations, if the Load switch is inactive.

As mentioned before, the output of the component AnimationSelected is used to cycle between the animations in the following way:

- 00 - ANIM1
- 01 - ANIM2
- 10 - ANIM3
- 11 - ANIM4

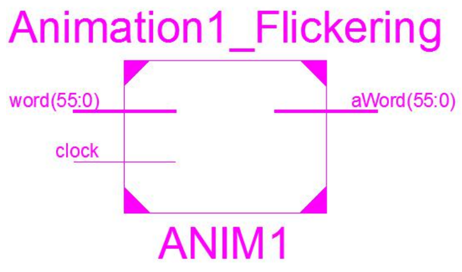
Due to the triviality of this component, it will not be shown. Instead, each of the four components inside it will be discussed thoroughly and they will be presented with the waveform simulations in Active-HDL.

Furthermore, each of the four components has the same inputs and output. Therefore, I will present only the schematic for the first animation, the rest being the same.

Also, those components have inside them a frequency divider, which was presented before, that reduces the frequency for 100 MHz to 1 Hz, so that the animations happen at an interval of approximately 1 second.

4.10.1 Animation1_Flickering

Bellow, an image of this component is attached.

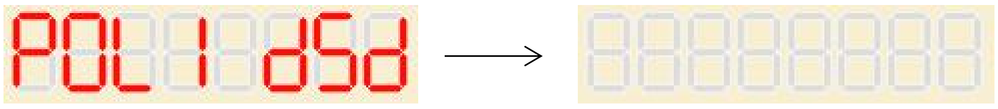


word - The word to be animated

clock - Board clock at f = 100 MHz

aWord - The animated word

The animation is presented in the sequence of images bellow.

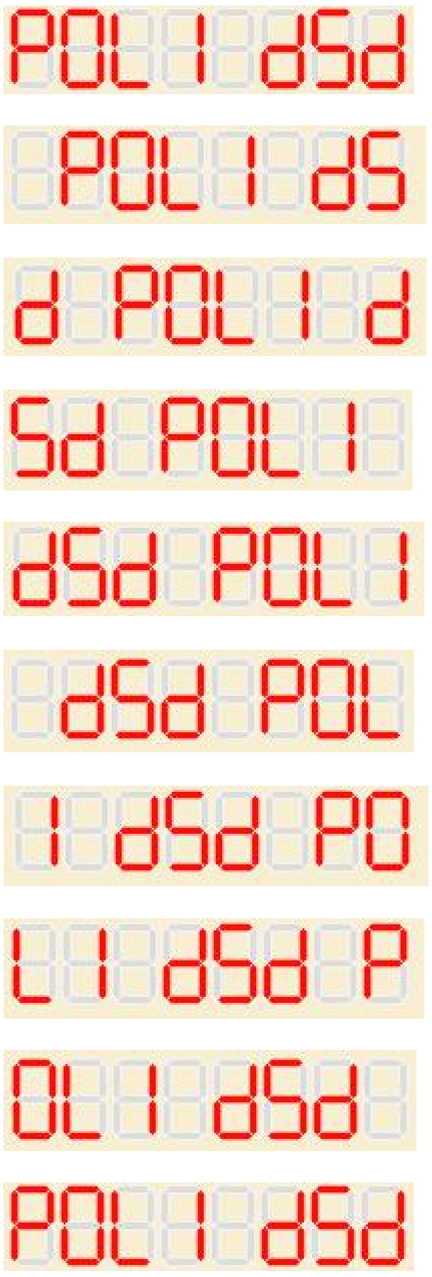


The waveform for this animation can be seen in Annex6.

Note that in the simulation there is no need for the frequency divider, so it will not be attached. This is true for all the components that are going to be presented next.

4.10.2 Animation2_SlideLeftToRight

The animation is presented in the sequence of images bellow.



As it can be seen, there is one extra space added to mark the end and the beginning of the word.

The waveform for this animation can be seen in Annex7.

A snippet from the code of this animation can be seen in Annex8.

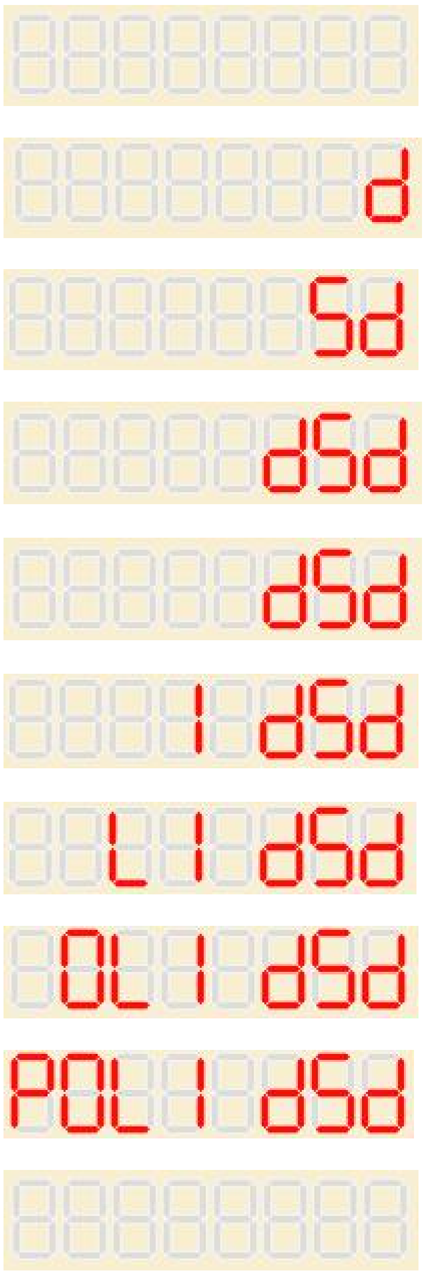
4.10.3 Animation3_SlideRightToLeft

Due to the similarities of this component with the previous one, it is not needed to attach images.

The animation functions in the same way as the one before, but in the opposite direction.

4.10.4 Animation4_ShowOneByOne

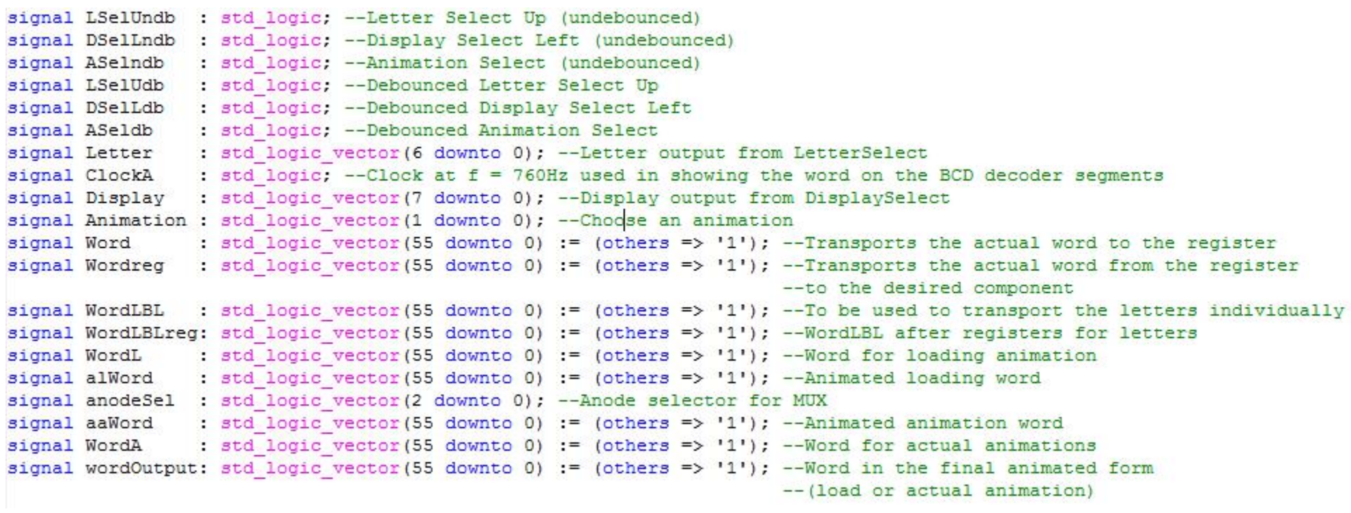
This animation shows the word one letter at a time. The animation is presented in the sequence of images bellow.



5. Notations

5.1 Internal signals

In what regards the internal signal used in the design, they can be seen highlighted in orange in the block diagram presented in the beginning. For a better understanding of their names and their functions, bellow a snippet from the MainAdv architecture is attached.



```
signal LSelUndb  : std_logic; --Letter Select Up (undebounced)
signal DSelLndb  : std_logic; --Display Select Left (undebounced)
signal ASelndb   : std_logic; --Animation Select (undebounced)
signal LSelUdb   : std_logic; --Debounced Letter Select Up
signal DSelLdb   : std_logic; --Debounced Display Select Left
signal ASeldb    : std_logic; --Debounced Animation Select
signal Letter    : std_logic_vector(6 downto 0); --Letter output from LetterSelect
signal ClockA    : std_logic; --Clock at f = 760Hz used in showing the word on the BCD decoder segments
signal Display   : std_logic_vector(7 downto 0); --Display output from DisplaySelect
signal Animation : std_logic_vector(1 downto 0); --Choose an animation
signal Word      : std_logic_vector(55 downto 0) := (others => '1'); --Transports the actual word to the register
signal Wordreg   : std_logic_vector(55 downto 0) := (others => '1'); --Transports the actual word from the register
                                                                    --to the desired component
signal WordLBL   : std_logic_vector(55 downto 0) := (others => '1'); --To be used to transport the letters individually
signal WordLBLreg: std_logic_vector(55 downto 0) := (others => '1'); --WordLBL after registers for letters
signal WordL     : std_logic_vector(55 downto 0) := (others => '1'); --Word for loading animation
signal alWord    : std_logic_vector(55 downto 0) := (others => '1'); --Animated loading word
signal anodeSel  : std_logic_vector(2 downto 0); --Anode selector for MUX
signal aaWord    : std_logic_vector(55 downto 0) := (others => '1'); --Animated animation word
signal WordA     : std_logic_vector(55 downto 0) := (others => '1'); --Word for actual animations
signal wordOutput: std_logic_vector(55 downto 0) := (others => '1'); --Word in the final animated form
                                                                    --(load or actual animation)
```

In this snippet, every signal declared has a comment representing its use in the design.

5.2 Components

The components notations have already been presented in the previous chapters as well as on the block diagram from the beginning.

6. Why this solution?

Although the large number of components may seem overwhelming, it is actually easier to follow the algorithms and the design because of its modularity.

I chose this approach when solving the problem because I was able to test each component separately to check if it is working properly. Then, after every single component was designed and tested, I connect them up with minimum difficulty.

Also, the fact that I opted for some sort of an interactive project, rather than one with the messages written directly in the code, made the challenge of solving this problem a bit harder. With this in the way, the only solution that came to my mind was this one, to make the design as modular as possible in order to minimize the chances of an error occurring.

7. Usage instructions

The design of this project is user friendly. With a minimal amount of buttons and switches, the user does not have to remember anything about the project other than some simple rules which will be presented now.

1. If the Load switch is on, the user can input a letter via the two buttons destined for that. Switching on the DIR switch makes this possible.

2. It is recommended that the user starts to introduce the word from right to left or from the last letter to the first because of the wait the LetterSelect and DisplaySelect component are designed.

3. When introducing a word, the current display will flicker, letting the user know on which display he is currently positioned.

4. After introducing the whole word, both Load and DIR switches must be turned of. Immediately after that, the first animation (Animation1_Flickering) will be activated.

5. To switch to a different animation, the user must press the button dedicated to the animations (ASel - BTNC, the center button).

6. If the user wants to introduce a new word, this process is repeated.

The usage of this design is not complicated and does not force the user to remember difficult combinations of switches.

8. Further development

The great thing about this design is its portability. Because of the fact that most component use generic parameters, the design can be implemented on almost any board, with the need of a few adjustments.

Also, if needed, the register that holds the whole word can be made bigger so that it will store a word longer than 8 letters. In this scenario, the only viable animations would have to be the sliding ones (Animation2_SlideLeftToRight and Animation3_SlideRightToLeft).

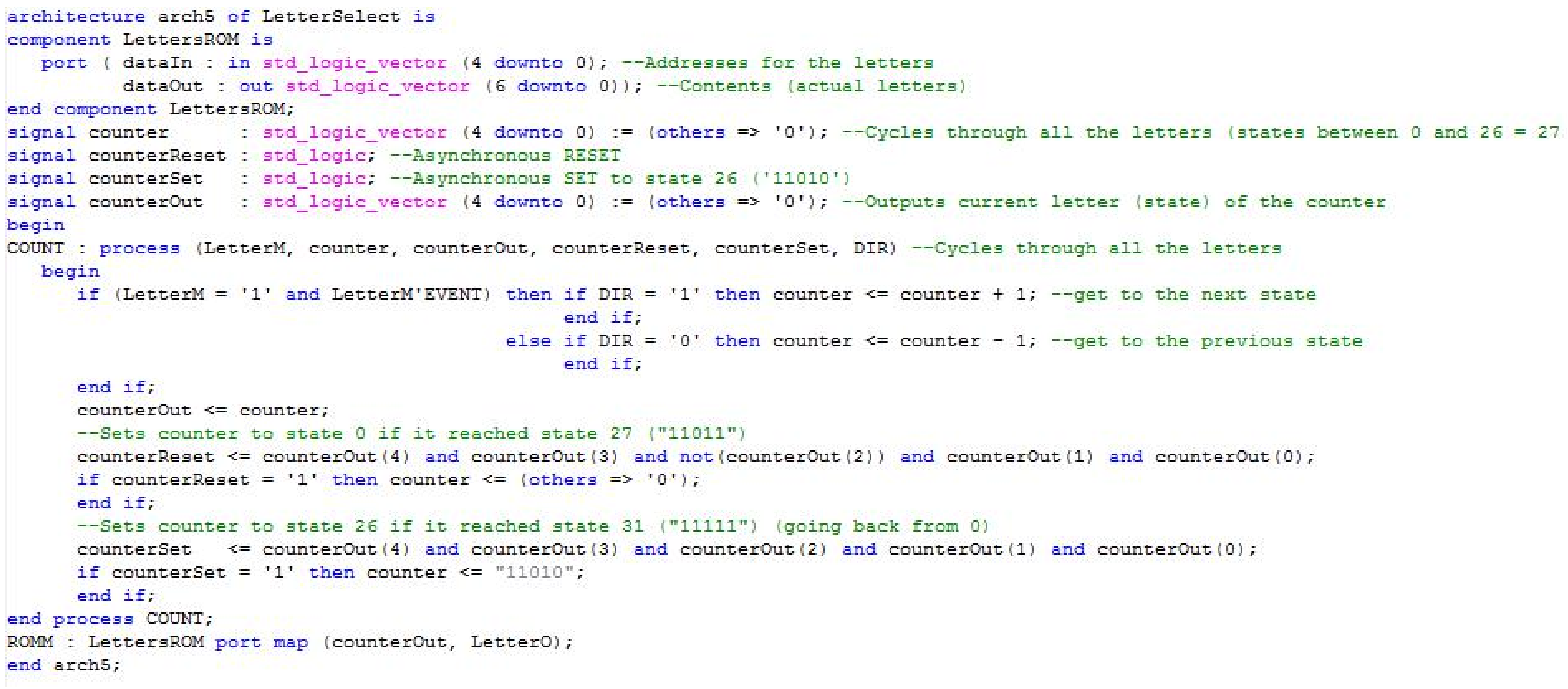
Furthermore, the LetterROM component where the code for all the letters are stored can be increased so that the addresses are on 6 bits (compared to the now 5 bits addresses) in order to allow the display of more characters then just the letters of the English alphabet. Even in this curret state, the ROM memory still has 5 addresses available for 5 new characters.

9. Annexes

9.1 Annex1

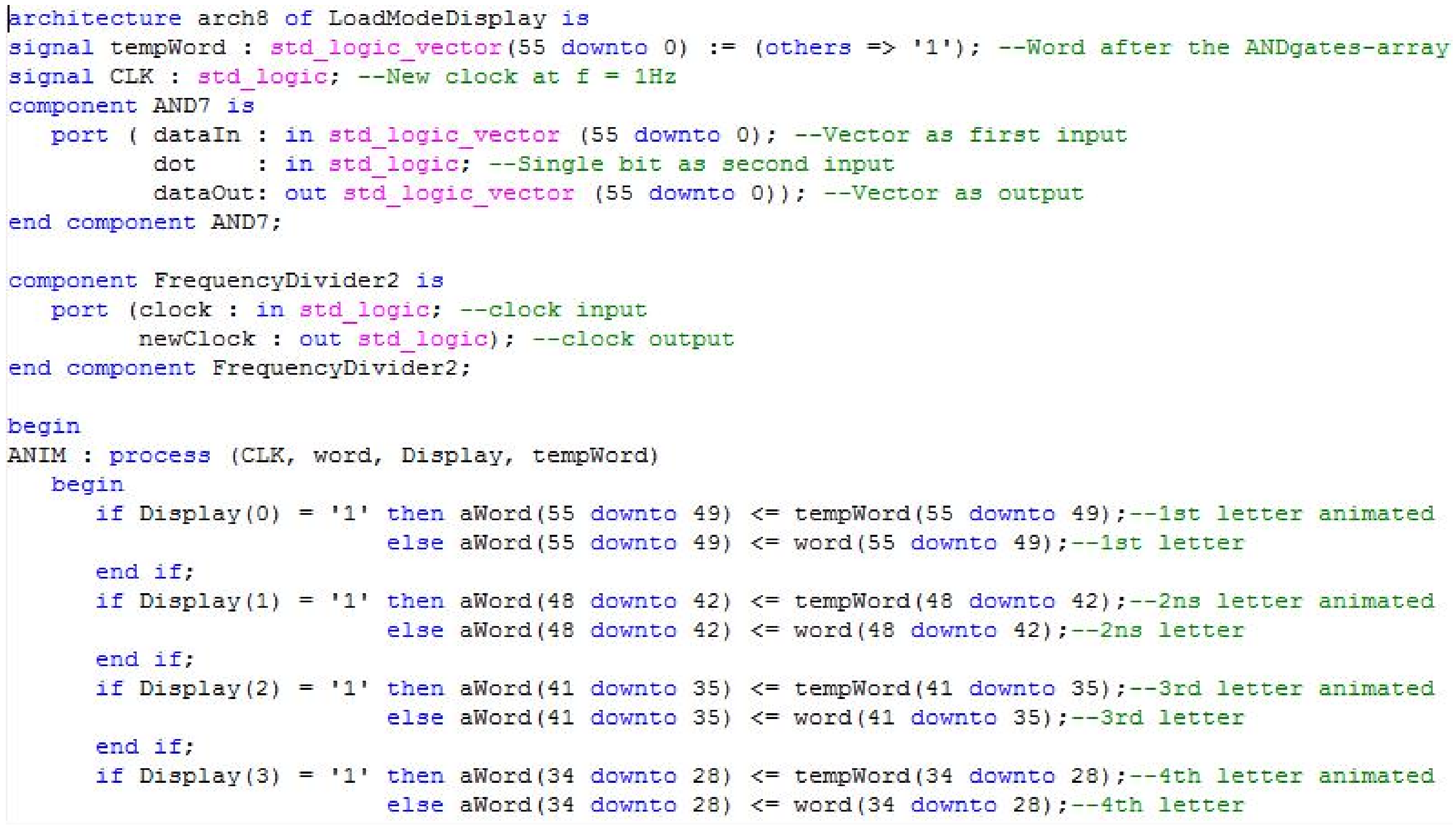
```
entity FrequencyDivider is
   generic (counterSize : integer := 17); --counter size (18) to reduce frequency from 100MHz down to 760Hz
   port (clock : in std_logic; --clock input
         newClock : out std_logic); --clock output
end FrequencyDivider;

architecture arch2 of FrequencyDivider is
signal counter : std_logic_vector (counterSize downto 0) := (others => '0'); --counter in state 0
constant number  : std_logic_vector (counterSize downto 0) := (others => '1'); --final state
begin
fDivider: process (CLOCK)
   begin
      if (clock = '1' and clock'event) then counter <= counter + 1; --increment counter state if rising edge of clock
         if counter(counterSize) = '1' then newClock <= '1'; --output newClock at f = 760MHz
            if counter = number then counter <= (others => '0'); -- reset counter when reaching final state
            end if;
         else newClock <= '0'; --no clock if counter not in final state
         end if;
      end if;
end process fDivider;
end arch2;
```

9.2 Annex2



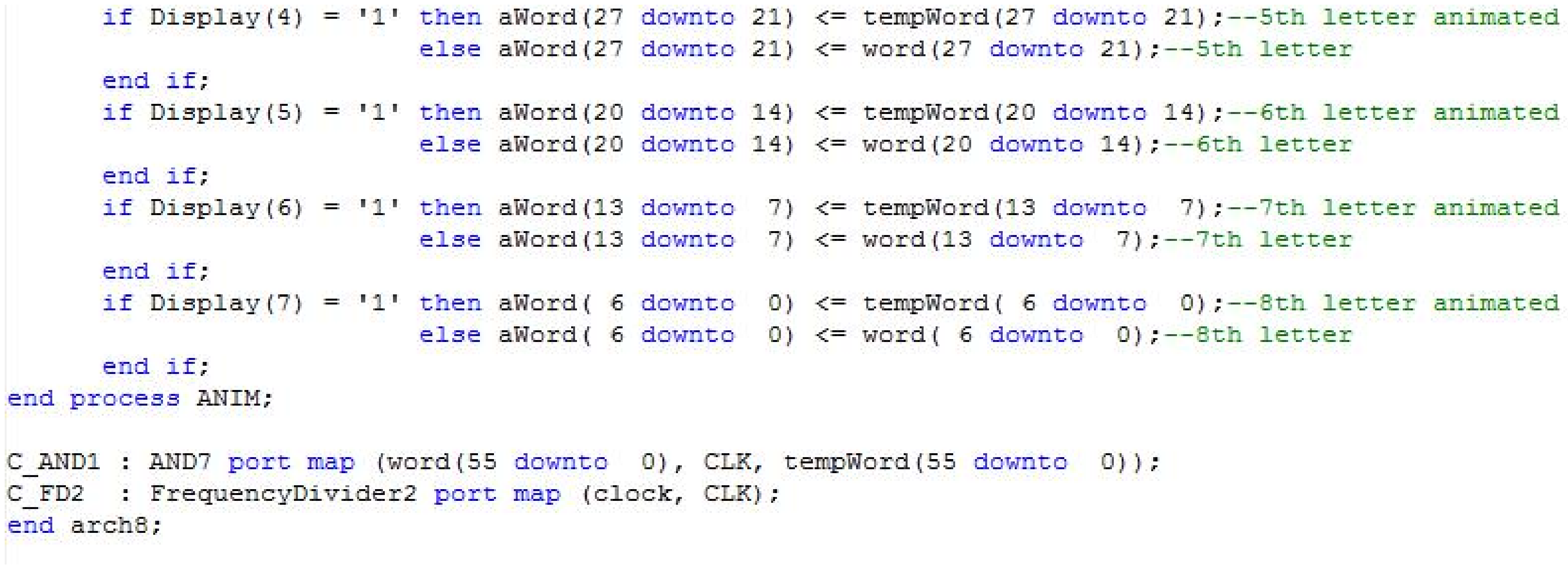
```vhdl
architecture arch5 of LetterSelect is
component LettersROM is
   port ( dataIn : in std_logic_vector (4 downto 0); --Addresses for the letters
          dataOut : out std_logic_vector (6 downto 0)); --Contents (actual letters)
end component LettersROM;
signal counter      : std_logic_vector (4 downto 0) := (others => '0'); --Cycles through all the letters (states between 0 and 26 = 27
signal counterReset : std_logic; --Asynchronous RESET
signal counterSet   : std_logic; --Asynchronous SET to state 26 ('11010')
signal counterOut   : std_logic_vector (4 downto 0) := (others => '0'); --Outputs current letter (state) of the counter
begin
COUNT : process (LetterM, counter, counterOut, counterReset, counterSet, DIR) --Cycles through all the letters
   begin
      if (LetterM = '1' and LetterM'EVENT) then if DIR = '1' then counter <= counter + 1; --get to the next state
                                                end if;
                                           else if DIR = '0' then counter <= counter - 1; --get to the previous state
                                                end if;
      end if;
      counterOut <= counter;
      --Sets counter to state 0 if it reached state 27 ("11011")
      counterReset <= counterOut(4) and counterOut(3) and not(counterOut(2)) and counterOut(1) and counterOut(0);
      if counterReset = '1' then counter <= (others => '0');
      end if;
      --Sets counter to state 26 if it reached state 31 ("11111") (going back from 0)
      counterSet   <= counterOut(4) and counterOut(3) and counterOut(2) and counterOut(1) and counterOut(0);
      if counterSet = '1' then counter <= "11010";
      end if;
end process COUNT;
ROMM : LettersROM port map (counterOut, LetterO);
end arch5;
```

9.3 Annex3



```vhdl
architecture arch8 of LoadModeDisplay is
signal tempWord : std_logic_vector(55 downto 0) := (others => '1'); --Word after the ANDgates-array
signal CLK : std_logic; --New clock at f = 1Hz
component AND7 is
   port ( dataIn : in std_logic_vector (55 downto 0); --Vector as first input
          dot    : in std_logic; --Single bit as second input
          dataOut: out std_logic_vector (55 downto 0)); --Vector as output
end component AND7;

component FrequencyDivider2 is
   port (clock : in std_logic; --clock input
         newClock : out std_logic); --clock output
end component FrequencyDivider2;

begin
ANIM : process (CLK, word, Display, tempWord)
   begin
      if Display(0) = '1' then aWord(55 downto 49) <= tempWord(55 downto 49);--1st letter animated
                          else aWord(55 downto 49) <= word(55 downto 49);--1st letter
      end if;
      if Display(1) = '1' then aWord(48 downto 42) <= tempWord(48 downto 42);--2ns letter animated
                          else aWord(48 downto 42) <= word(48 downto 42);--2ns letter
      end if;
      if Display(2) = '1' then aWord(41 downto 35) <= tempWord(41 downto 35);--3rd letter animated
                          else aWord(41 downto 35) <= word(41 downto 35);--3rd letter
      end if;
      if Display(3) = '1' then aWord(34 downto 28) <= tempWord(34 downto 28);--4th letter animated
                          else aWord(34 downto 28) <= word(34 downto 28);--4th letter
```

9.4 Annex4

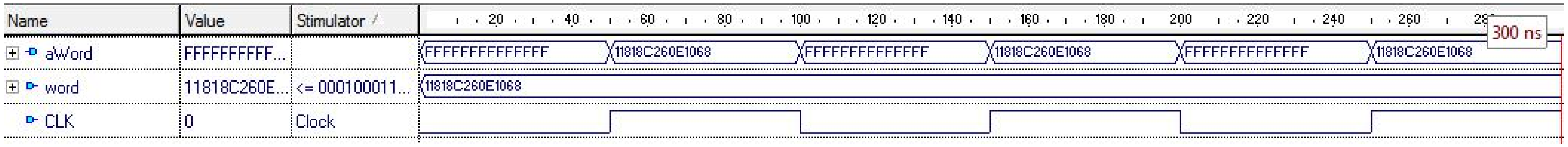


```
        if Display(4) = '1' then aWord(27 downto 21) <= tempWord(27 downto 21);--5th letter animated
                             else aWord(27 downto 21) <= word(27 downto 21);--5th letter
        end if;
        if Display(5) = '1' then aWord(20 downto 14) <= tempWord(20 downto 14);--6th letter animated
                             else aWord(20 downto 14) <= word(20 downto 14);--6th letter
        end if;
        if Display(6) = '1' then aWord(13 downto  7) <= tempWord(13 downto  7);--7th letter animated
                             else aWord(13 downto  7) <= word(13 downto  7);--7th letter
        end if;
        if Display(7) = '1' then aWord( 6 downto  0) <= tempWord( 6 downto  0);--8th letter animated
                             else aWord( 6 downto  0) <= word( 6 downto  0);--8th letter
        end if;
end process ANIM;

C_AND1 : AND7 port map (word(55 downto  0), CLK, tempWord(55 downto  0));
C_FD2  : FrequencyDivider2 port map (clock, CLK);
end arch8;
```

9.5 Annex5

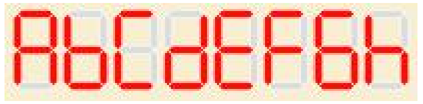
```
entity Register7bits is
   generic ( registerSize : integer := 6); --Size of the register is 56 bits
   port ( dataIn : in std_logic_vector (registerSize downto 0); --Data to store inside the register
          CLKB   : in std_logic; -- Board Clock at f = 100MHz
          store  : in std_logic; -- store = 1 => store the data (Load signal from outside)
                                 -- store = 0 => output the data
          dataOut: out std_logic_vector (registerSize downto 0)); --Output the stored data
end Register7bits;

architecture Behavioral of Register7bits is
signal tempData : std_logic_vector (registerSize downto 0); --Temporary keep data inside the register
begin
HOLD : process(dataIn, CLKB, tempData)
   begin
      if CLKB = '1' and CLKB'EVENT then if store = '1' then tempData <= dataIn;
                                        end if;
      end if;
      dataOut <= tempData;
end process HOLD;

end Behavioral;
```

9.6 Annex6

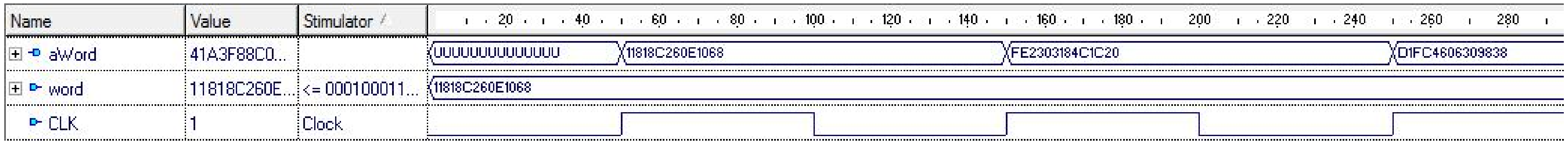


Name	Value	Stimulator
aWord	FFFFFFFFFF...	
word	11818C260E...	<= 000100011...
CLK	0	Clock

As it can be seen in the simulation, the output of the component cycles between the message and all segments closed. The input word was: 0001000 1100000 0110001 1000010 0110000 0111000 0100000 1101000, namely:



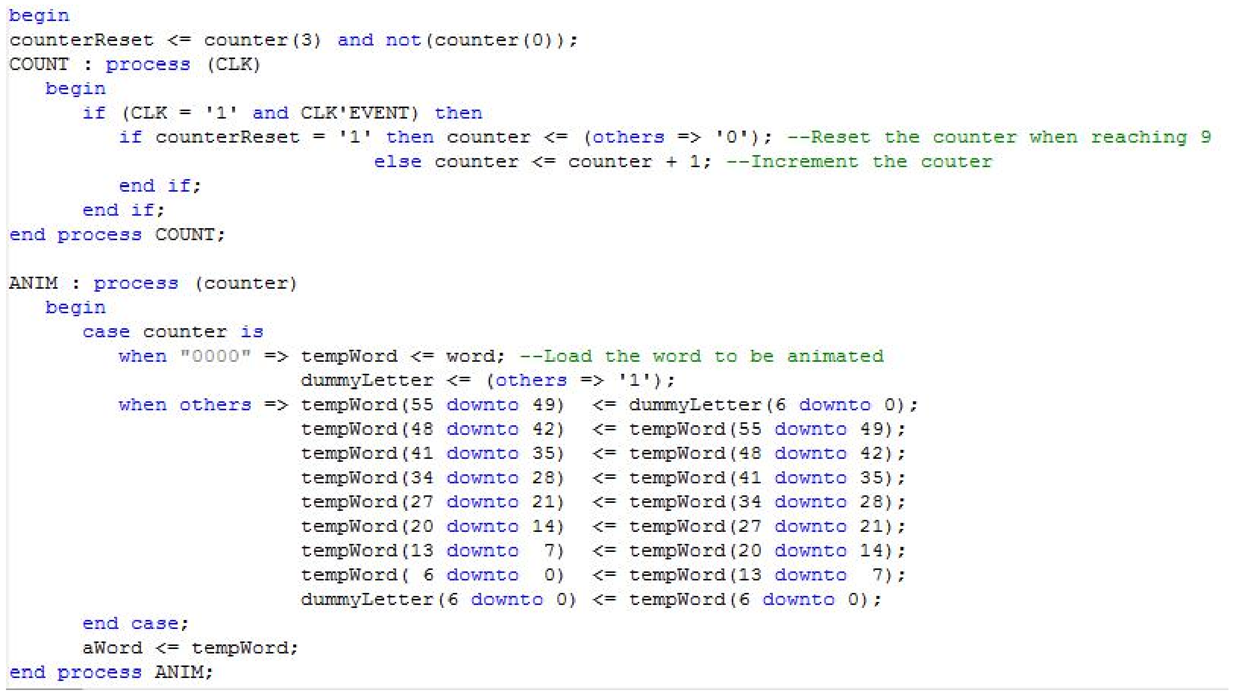
9.7 Annex7



Name	Value	Stimulator
aWord	41A3F88C0...	
word	11818C260E...	<= 000100011...
CLK	1	Clock

As it can be seen, the animated word is sliding at each clock cycle. The input sequence is the same as in Annex6.

9.8 Annex8



```vhdl
begin
counterReset <= counter(3) and not(counter(0));
COUNT : process (CLK)
   begin
      if (CLK = '1' and CLK'EVENT) then
         if counterReset = '1' then counter <= (others => '0'); --Reset the counter when reaching 9
                                else counter <= counter + 1; --Increment the couter
         end if;
      end if;
end process COUNT;

ANIM : process (counter)
   begin
      case counter is
         when "0000" => tempWord <= word; --Load the word to be animated
                        dummyLetter <= (others => '1');
         when others => tempWord(55 downto 49)  <= dummyLetter(6 downto 0);
                        tempWord(48 downto 42)  <= tempWord(55 downto 49);
                        tempWord(41 downto 35)  <= tempWord(48 downto 42);
                        tempWord(34 downto 28)  <= tempWord(41 downto 35);
                        tempWord(27 downto 21)  <= tempWord(34 downto 28);
                        tempWord(20 downto 14)  <= tempWord(27 downto 21);
                        tempWord(13 downto  7)  <= tempWord(20 downto 14);
                        tempWord( 6 downto  0)  <= tempWord(13 downto  7);
                        dummyLetter(6 downto 0) <= tempWord(6 downto 0);
      end case;
      aWord <= tempWord;
end process ANIM;
```