MINISTERUL EDUCAȚIEI

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

SNAKE

DSD PROJECT

Students: Zotescu Ruth, Tătărău Oana

Project supervisor: Ileana Blaj

2023

1. Specifications

You are part of a team that has to design the game “SNAKE” , designed especially for student entertainment during exam season. The game must have this behavior: you will control the direction of movement of the snake with the goal to eat as much food as possible to achieve the maximum length, the game will be displayed on a monitor connected to the FPGA board by means of a VGA cable.

The game must work as follows:

● In the initial state the snake spawns in the middle of the map, standing still (state “O”), facing right, waiting for the player’s first input to start moving. The initial length is 3 (including the head). Also, the first “peach” spawns in.

● The current direction that the snake is facing is indicated by its blue eyes.

● The movement direction is controlled by pressing the “U” (up), “D” (down), “L” (left), “R”(right) buttons, that determine their respective states, “SU”, “SD”, ”SL”, ”SR”. They follow the rule that, if you are facing, for example right, pressing left will not have any effect on the current movement, i.e., you cannot turn the snake 180◦ in one change of movement.

● The snake is able to “go through walls”, i.e., it will pass through the walls appearing on the other side (just like the adjacency in the Karnaugh map).

● When the snake reaches a “peach” its length will increase by 1, and a new “peach” will be randomly generated.

● The speed of the snake will be increased at every couple eaten “peaches”.

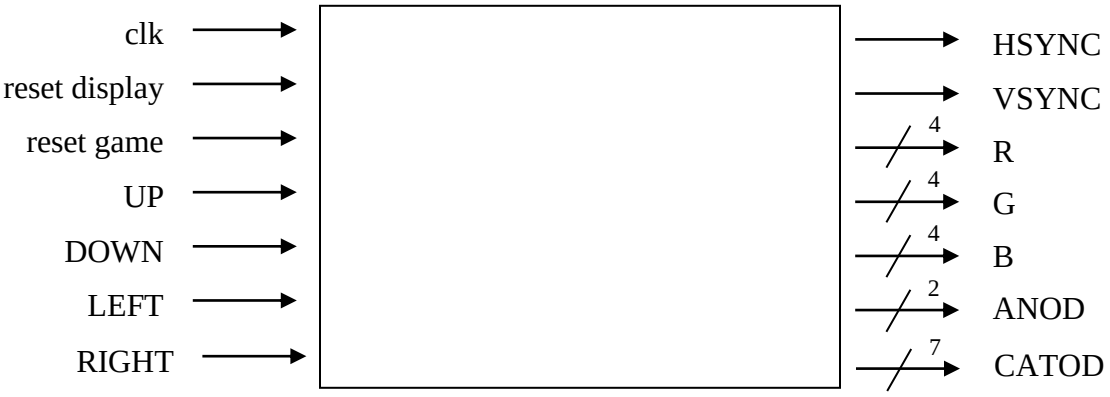
● The score represents the number of eaten “peaches” and will be displayed on the S.S.D.

● The game will end when the snake tries eating itself. Then, it will stop moving, going in state “O”. To restart the game the player will need to reset it manually, by a switch/ button.

● The “reset game” switch/ button is asynchronous.

2. Design

2.1 Black box



2.1.1 The inputs

Most of the design is made synchronously, the clk is used for the display on the monitor, on the anods, for the frequency of movment, etc.

The „reset display" is not completly necesary, its an extra asynchronous feature for turning on and of the screen. It does not however influence the game in any way, even if the screen is turned off, the game goes on.

The funcionality of „reset game" is detailed in the specifications, it is an asynchronous input that resets the game to its initial state.

UP, DOWN, LEFT, RIGHT are the most important inputs, they control the direction of the movement, as mentioned in the specifications.

2.1.2 The outputs

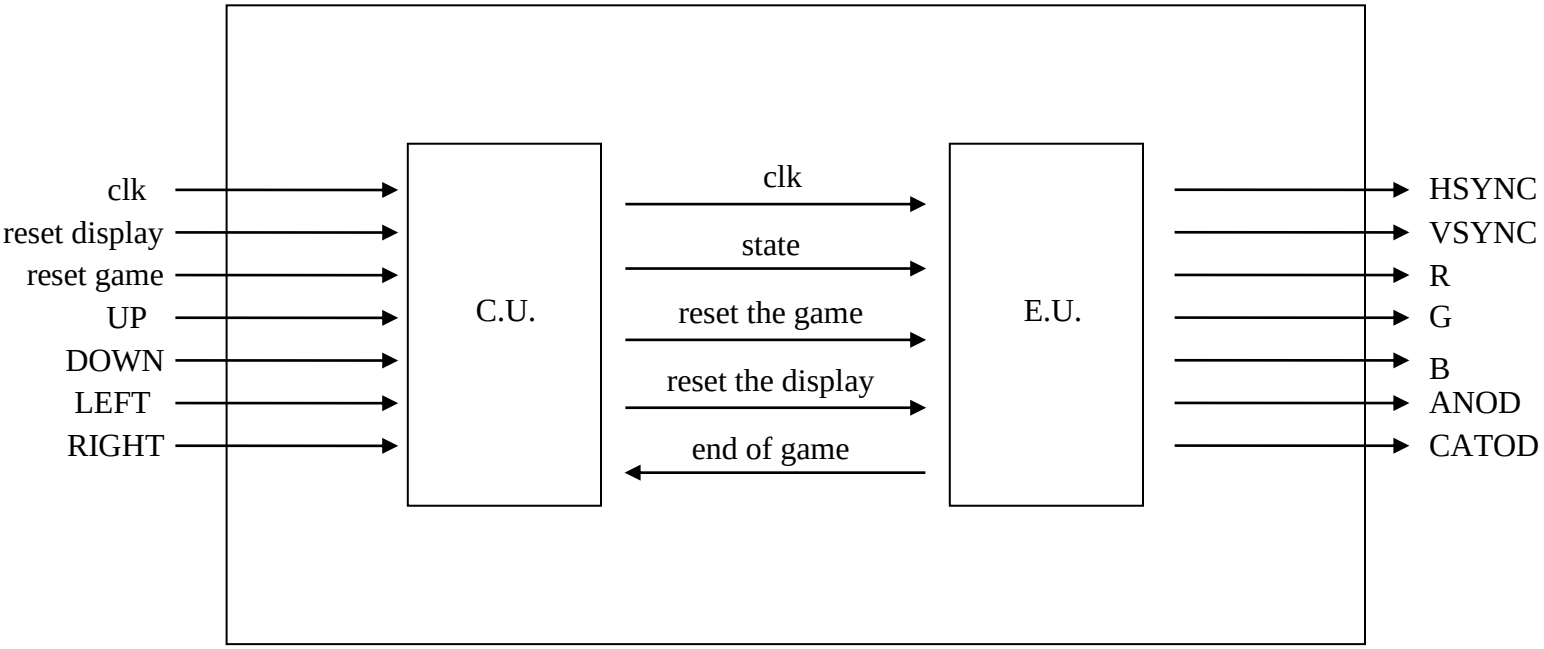
HSYNC makes electron beam restart at next screen's scanline (starts a new line). Pulses on HSYNC signal mark the start and end of a line and ensure that the monitor displays the pixels between the left and right edges of the visible screen area.

VSYNC makes electron beam restart at first screen's scanline (starts a new frame). Pulses on VSYNC signal mark the start and end of a frame made up of video lines and ensure that the monitor displays the lines between the top and bottom edges of the visible monitor screen.

R, G and B are the outputs that determine the colour of each pixel on the monitor. Each color is a combination of the 3 primary colors red, green and blue.

For the displaying the score on the S.S.D. of the board we will use 2 anods, the ouput ANOD selects the anod on which we want to display the value from CATOD in that moment.

2.2 Detail diagram



The control unit mainly holds the current state and determines the next one depending on the inputs, and on the feedback from the execution unit. It also sends to the execution unit the reset instructions.

The execution unit works on a divided clk which is obtained in the control unit by dividing the frequency of the original clk.

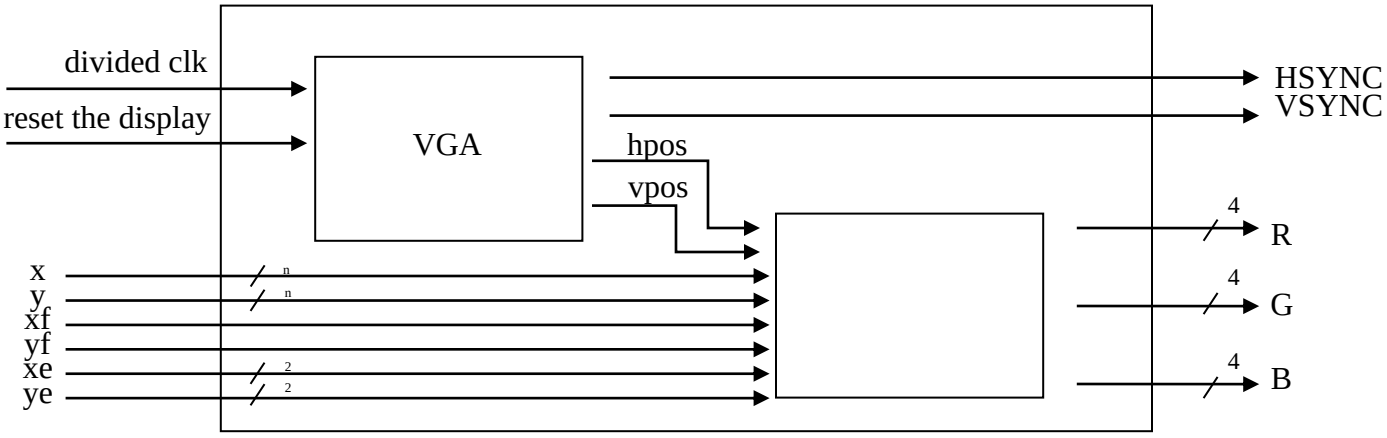
In the execution unit, depending on the state, the position of the snake and of the food will be modified acordingly. Here there is also implemented the display fucntionality on the monitor and also on the S.S.D. The execution unit also receives instructions for reseting the game or the screen.

The feedback it sends to the control unit is in the case of the end of the game, when the snake tries eating itself, with the purpose of changing the state back to standing still.

3.Structure and functionality

3.1 Resources

3.1.1 VGA



The specifications of our screen are:
Resolution: 640 x 480
Pixel frequency: 25.175 MHz

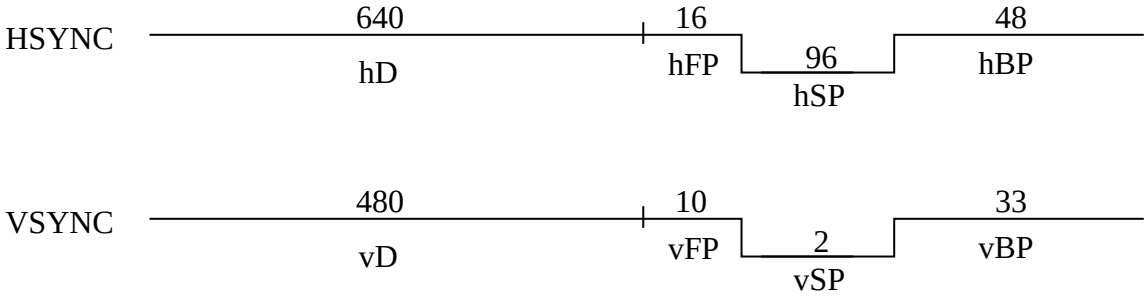
Horizontal timing:

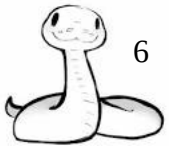
Scanline part	Pixels
Visible area	640
Front porch	16
Sync pulse	96
Back porch	48
Whole line	800

Vertical timing:

Scanline part	Pixels
Visible area	480
Front porch	10
Sync pulse	2
Back porch	33
Whole line	525

The divided clk represent the pixel frequency for our screen , 25 MHz. It is obtained by dividing the original clk twice (since the original clk is 100MHz).





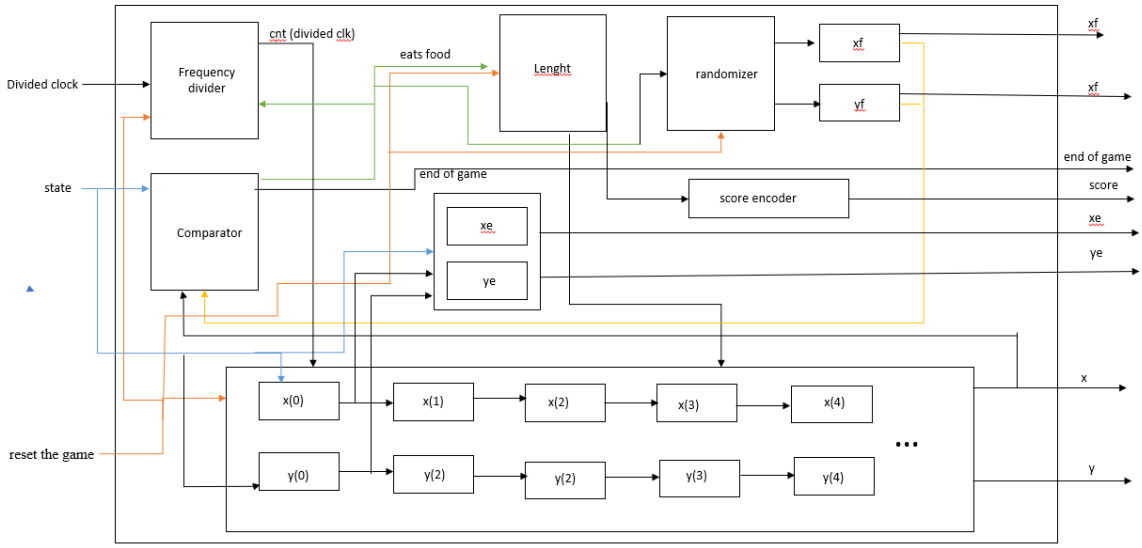
Hpos and vpos are counters that used for the configuration of HSYNC and VSYNC as above. They go from 0 to (640 + 16 + 96 + 48) -1 / (480 + 10 + 2 + 33) -1. While HSYNC and VSYNC are in the display area, hpos and vpos will mark the position (row and column) of a pixel on the screen. With the knowledge of its position we can set it a color by giving specific values to RGB.

Therefore, by knowing the supposing position of the snake's body, the food and eyes (by the coordinates x, y, xf, yf, xe, ye) we can represent them on screen, in different colors. The rest of the screen will be a “default” color, for this project it is black.

The movement illusion is achieved as a result of the internal change of the object's coordinates and the fast refreshing rate of the screen.

While reset the display is active HSYNC and VSYNC will be 0, determining the monitor to be completely black.

3.1.2 Main Component



This is only a representation of what happens. It might not be completely accurate, since we used behavioral description in our VHDL code, NOT STRUCTURAL

In the main component the movement takes place. Here the coordinates of the snake's body and of the food get modified depending on the clk, state and reset the game.

As the snake must gain in speed at some point, a new reference must be used. Here the signal cnt comes in hand as it is divided from the main clk and its incrementation rate increases every 5 eaten "peaches".

In the comparator component, depending on the introduced direction, the coordinates of every block of the snake and of the food are compared to the snake's head, verifying if the head and the tail collide or if the snake ate a "peach". In the case of the

first scenario, eof (end of game) signal is activated. For the latter, the lenght (len) in incremented.

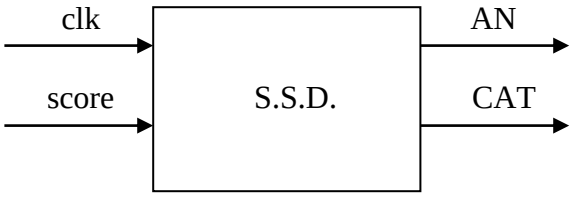
There must exist a piece of memory which holds the coordinates of every block of the snake. The state will alter the head of the snake (x and y cordinates). The next block must follow the head, and the one after it must follow this one and so on. So, if we denote the head by x(0) and the rest by x(i) then at every eaten "peach" x(i) receives x(i-1)'s coordinates.

The food is generated by the randomizer and the eyes depend on the head.

The score is the (lenght -2) (since the starting lenght is 2) encoded in BCD for S.S.D..

The “reset the game” signal resets all the coordinates and the length to its initial state.

3.1.3 S.S.D.

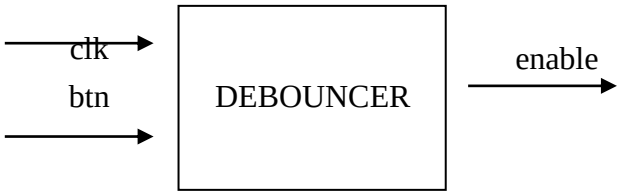


The score is shown on the S.S.D..

Since the score is encoded in BCD on the rightmost anod the bits 3-0 are shown and on the left one the bits 7-4 are shwon. Because we only have the possibility of displaying one digit at a time, we will select the anodes one after another using the AN output, when the first anod is selected, on the CAT signal the first digit will be encoded, and when the secodn anod is selected, , on the CAT signal the second digit will be encoded. The encoding from BCD to seven segments is done thusly:

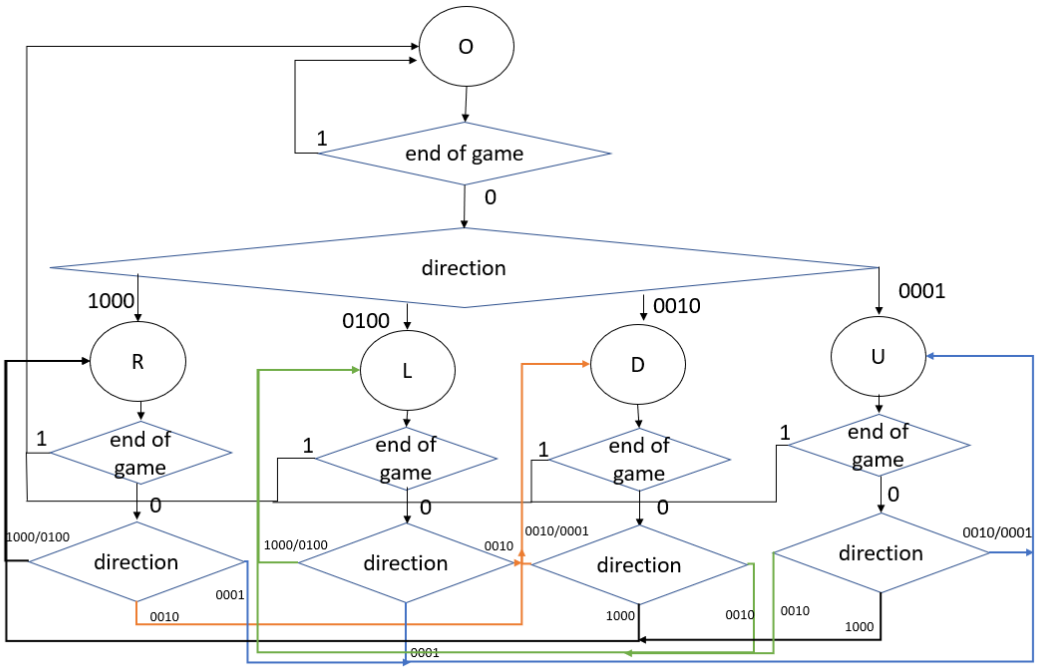
```
case input_decoder is
   when "0000" => cat<="0000001";
   when "0001" => cat<="1001111";
   when "0010" => cat<="0010010";
   when "0011" => cat<="0000110";

   when "0100" => cat<="1001100";
   when "0101" => cat<="0100100";
   when "0110" => cat<="0100000";
   when "0111" => cat<="0001111";

   when "1000" => cat<="0000000";
   when "1001" => cat<="0000100";
   when "1010" => cat<="0001000";
   when "1011" => cat<="1100000";

   when "1100" => cat<="0110001";
   when "1101" => cat<="1000010";
   when "1110" => cat<="0110000";
   when others => cat<="0111000";
end case;
```

3.1.4 Debouncer



The debouncer is a standard debouncer that will compensate for faulty buttons. It also transforms a continuous input into a pluse. Even if you keep the button pressed it will have no effect. This compensates for buttons that might not function properly, staying pressed for longer than they should.

3.2 State Diagram



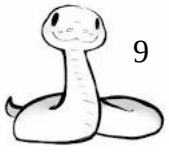
Direction is a signal connected to the inputs thusly:

```
direction(0)<=UP;
direction(1)<=DOWN;
direction(2)<=LEFT;
direction(3)<=RIGHT;
```

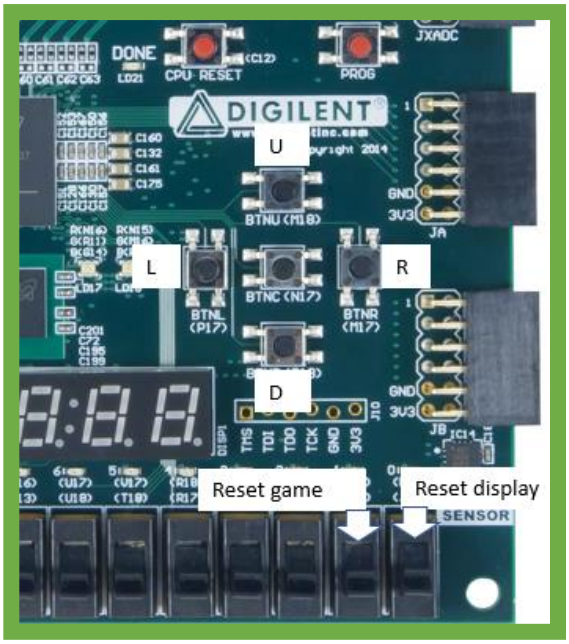
As mentioned in the specifications, when in state R, input LEFT will have no effect, since the snake cannot turn 180° in one move, so it will stay in state R.

This determination of next state based on inputs is done by the E.U..

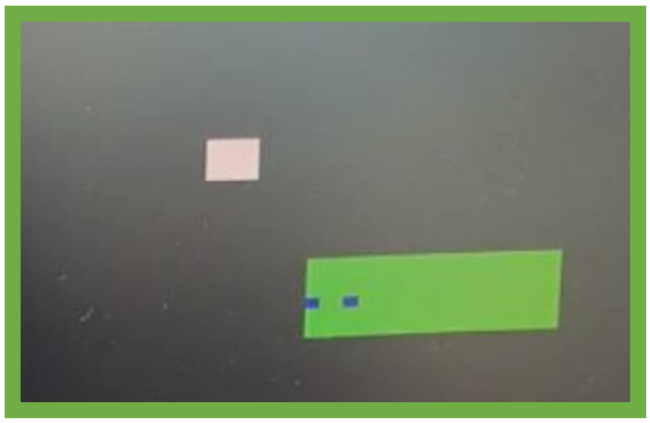
Additionally the E.U. will send to the C.U. the „reset the game” and „reset the display” signal if the respective input is activated.



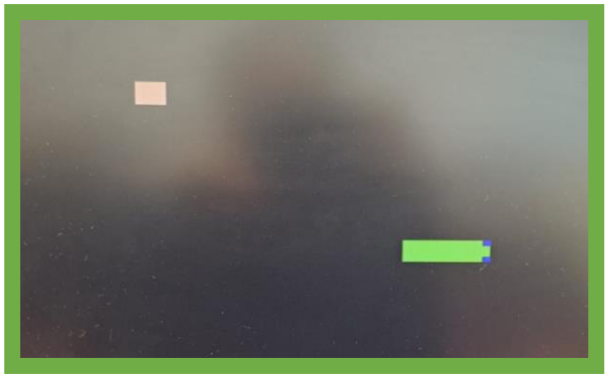
4.Utility and results



The user interface consists of the U,D,L,R buttons and the reset buttons. They are mostly self explanatory, we do not advise fast changes of direction, as it may produce unwanted results.

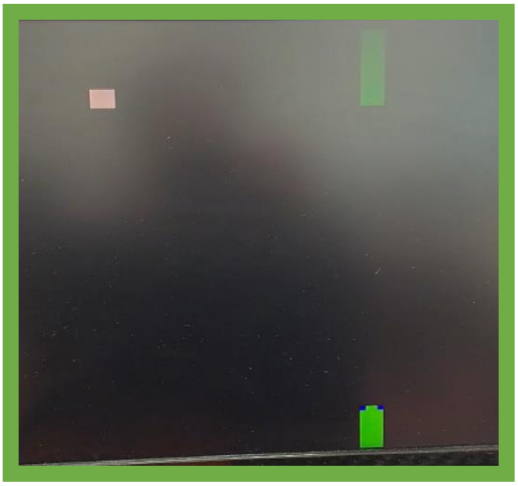


And the game will end when the snake tries eating itslef.

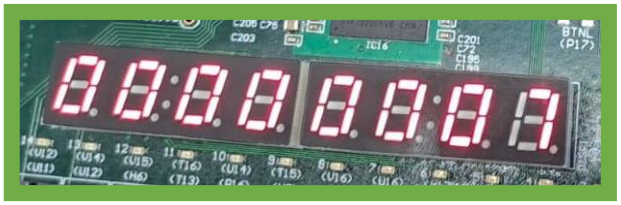


The final result is a moving snake, that eats „peaches" to increase its lenght.

This is a picture of the initial state.



The snake can also go through walls as intended.



Further more the score is displayed on the SSD, such that even if the game ended or the snake reached its maximum lenght, there is still a way to find out the score of the game.

5.Further development

The Snake Game has an endless improving potential.

As its most striking part it's the visual one, the game could have some extra animations besides the moving snake itself. Furthermore, multiple gamemodes could be added.

The game could become more enjoyable for a user if the input was introduced by means an ergonomic keyboard which is less stiffier than the one on the FPGA board.

Another important addition would represent a better debouncer which would prevent the extra activation of an unwanted state that could lead to unwanted and unreversible bugs that reduce the quality of the experience.

The current randomizer is not an effective one as a simple equations was used to generate the coordianetes of the food. There are better such algorithms specifically designed for VHDL. More than that, the food should't be able to spawn in the snake (at a position where there aready is the body of the snake).

The current observed „bugs” are as follows:

There are hidden rows/collumns on the screen, depending on wich screen you use, sometimes the food will spawn there, making it look like it didnt spawn at all, or, if the snake goes out of the visible area, it will look like it dissapeared.

Probably because of the input buttons and the missing debouncer, the snake will verry rarely move diagonally, dissapearing out of the monitor.

Sometimes it will also randomly stop moving, even if we did not code any way for it to go back to the „O” state without dying.

The change of state is made on the rising edge of the 100MHz clock, but the movement is done around once every second, so even if technically the snake should not be able to turn around in one move, if done fast enough this specific occurance is possible.

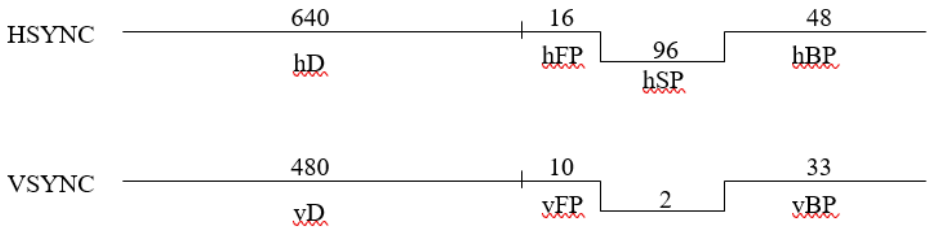
6.Technical justifications for the design

The HSYNC and VSYNC outputs are determined by some constants specific to the monitor, these are:

```
constant hD:integer :=640;
constant hFP:integer :=16;
constant hSP:integer :=96;
constant hBP:integer :=48;
constant hWL:integer :=800;
------------------------------
constant vD:integer :=480;
constant vFP:integer :=10;
constant vSP:integer :=2;
constant vBP:integer :=33;
constant vWL:integer :=525;
```

This values were declared as constants in the architecture for easy modification in case of implementation on a monitor with diffrent specificacions.

The HSYNC and VSYNC are configured in this way :



To be able to test easly if we are in the display area. Since the display time is at the beggining of the signal, we only test if hpos<640 and vpos<480.

```
process(div_clk2, hpos, vpos)
begin
if(rising_edge(div_clk2))then
if(hpos<=(hD-1) and vpos<=(vD-1))then
desen<='1';
else
desen<='0';
end if;
end if;
end process;
```

For the moving frequency of the snake, the initial one is set to aproximately 1Hz, using a counter on „power” bits, and using the most significat bit as our divided clock. We did the divider this way for conveniency reasons.

```
constant power: integer:=22;
signal cnt: std_logic_vector(power downto 0):="0000000000000000000000";
```

The way we increase the moving frequecny is by increasing the increment, „a”, that we add to the cnt.

```
process(div_clk2)
begin
if(rising_edge(div_clk2))then
cnt<=cnt+a;
end if;
end process;
```

In this way we can control the rate of acceleration more precisely, since we do not want it to increase dramatically every time.

The way we implemented the snake is like this:

The lenght of the snake is limited (in our care the maximum lenght is 16, including the head). Technically the lenght of the snake is always 16, but the part that is showing is the one determined by the „l” counter/ the score. The way it works is like this:

The excess parts of the snake are stacked one on top of another in the tail.

When the snake moves, we first move the tail, togheter with the whole stack on top of it at the second to last coordinate of the snake. Then the rest of the snake moves like this x(i)<=x(i-1), y(i)<=y(i-1). And then the head moves, depending on the current state.

```
 ------------------------------------------- body moves
l2:for i in lmax downto 1 loop
if(i>=l) then
 x(i)<=x(l-1);
 y(i)<=y(l-1);
 else
 x(i)<=x(i-1);
 y(i)<=y(i-1);
end if;
end loop l2;
 ------------------------------------------- head moves
if((x(0)-20)>=0) then
 x(0)<=x(0)-20;
 else
 x(0)<=(hD-20);
end if;
```

Of course before it moves it checks if it can move in the desired direction/ if it doesnt eat itself. If, by moving in that direction, it would eat itself, the game ends.

Additionally it also checks if the block it is moving to is a „peach”. If it is, the lenght is increased, and another randomized food appears.

```
l7: for i in lmax downto 1 loop
if(x(0)-20=x(i) and y(0)=y(i))then
eof<='1';
end if;
end loop l7;
```

```
if((x(0)-20=xf or(x(0)-20<0 and xf=hD-20)) and y(0)=yf)then
xf<=( (x(1)+x(3)+y(5)+15 )mod 32)*20;
yf<=( (y(7)+x(4)+x(0)+9 )mod 24)*20;
l<=l+1;
if((l mod 5)=0)then
a<=a+1;
end if;
end if;
```

The randomization of the food is done using the current coordinates of the snake, since the exact position from wich the food is eaten is mostly random. This of course has some drawbacks. Since at the beggining the food appears in the exact same spot, the snake eating it from the same side each time is more probable. So the beggining of the game will look mostly the same each time, and if you repeat the same movements each time you play the game, the food will also appear in the same spot each time

```
if((x(0)-20=xf or(x(0)-20<0 and xf=hD-20)) and y(0)=yf)then
xf<=( (x(1)+x(3)+y(5)+15 )mod 32)*20;
yf<=( (y(7)+x(4)+x(0)+9 )mod 24)*20;
l<=l+1;
if((l mod 5)=0)then
a<=a+1;
end if;
end if;
```

For the display, we chose for the size of the food and the of each snake block, to be 20x20 pixels. The display function will chose diffrent RGB values for each case.