CE339 - High Level Digital Design

March 18

2016

Snake Game Implementation in VHDL

Michael Bridden
1101557

Contents

Introduction

Top Level

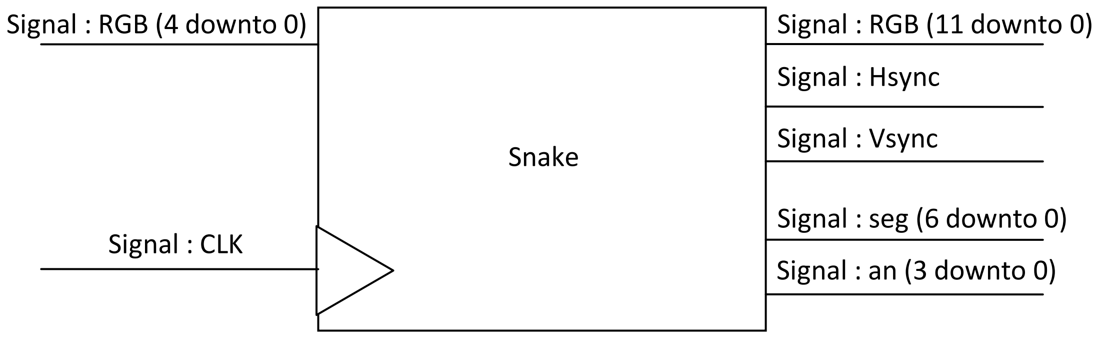
Signal : RGB (4 downto 0)
Signal : CLK
Snake
Signal : RGB (11 downto 0)
Signal : Hsync
Signal : Vsync
Signal : seg (6 downto 0)
Signal : an (3 downto 0)

Fig 1 - Top Level

The top level of the sstem or black box consists of the inputs and outputs of the system. For this projec we use the 5 buttons (btnU - change snake direction up; btnD - change snake direction down; btnL - change snake direction left; btnR - change snake direcion right and btnC for debugging) we use a VGA display which requires the RGB values we are using RGB444 for this project. The other two signals required are horizontal sync and vertical sync these pulses are used for synchronising the device with the monitor.

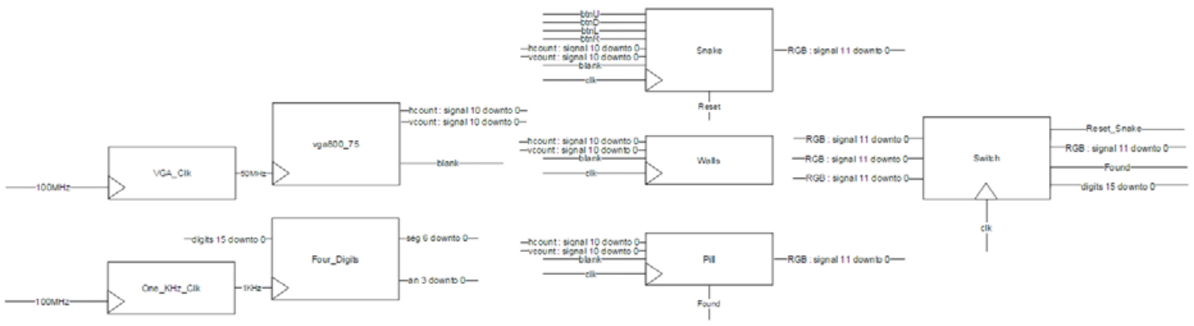

Fig 2 - Top level entities

The signal connections for the top level are shown in fig 2. The vga800_75 entity is a 800x600 driver with a 75Hz refresh rate it requires a 50MHz clock which is provided by VGA_Clk. All the Graphical components are connected to the vga drivers HCount, VCount and blank (not shown above due to overlap). There are several return signals from the switch including reset snake (to reset the position when hit wall); found (to set new position for pill) and digits (provides the current score). The switch has all 3 graphical components attached and therefore is able to detect when these conditions are met.

Snake

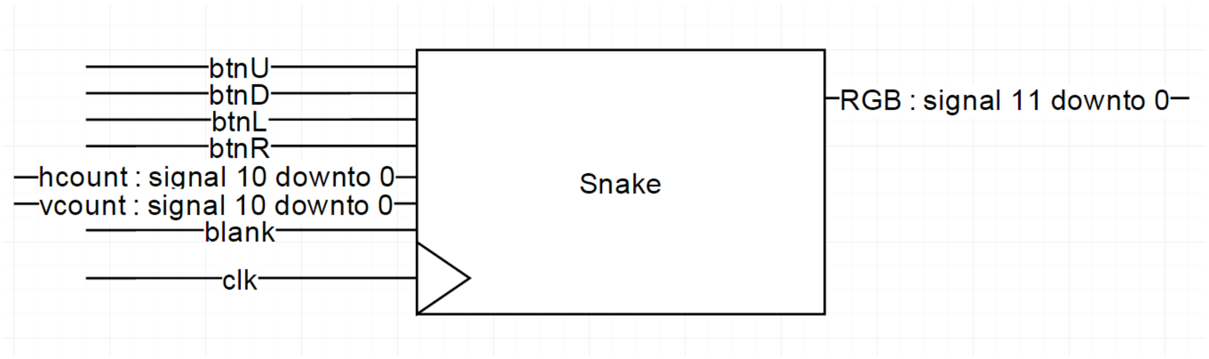

Fig 3 - Snake entity

The snake entity requires the current x,y location of the display which is given by the HCount and VCount signals. This will be AND'd with the blanking signal so it doesn't output during the blanking period. The buttons act as a way to change the direction of the snake however the snake cannot go back on itself so not all button presses are registered. The reset snake input is for when the snake hits a wall as it needs to return to the start position.

The snake is split into 8x8 segments each segment has an x,y,d (x value, y value, direction) there is an equal number of turn segments which store the x,y of the head when a button is pressed and the direction the snake should change at that position. This is what allows the tail of the 3 segment snake to follow the turning pattern.

Walls

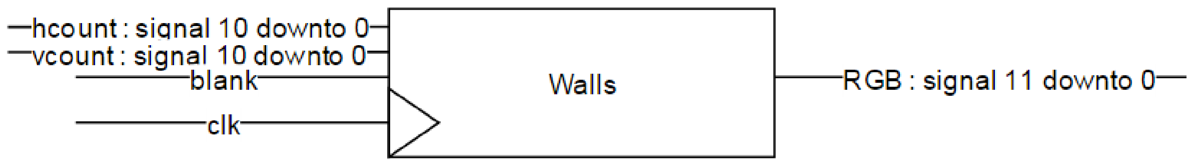

Fig 4 - Walls entity

The wall entity requires the current x,y location of the display which is given by the HCount and VCount signals. This will be AND'd with the blanking signal so it doesn't output during the blanking period. The wall is simply a less than or greater than comparison of a constant with HCount and Vcount.

Pills

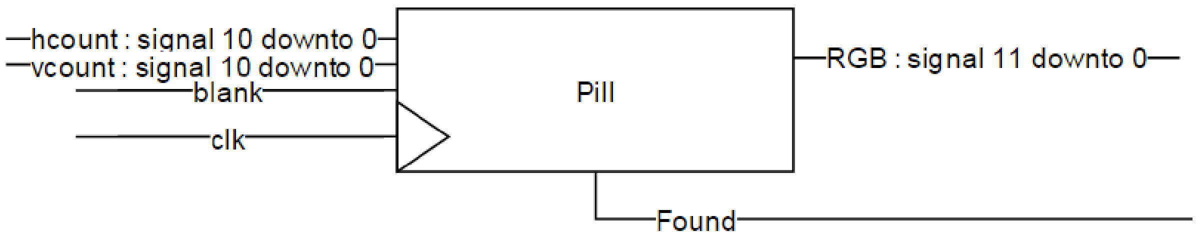

Fig 5 - Pills entity

The pills entity requires the current x,y location of the display which is given by the HCount and VCount signals. This will be AND'd with the blanking signal so it doesn't output during the blanking period. The pill entity requires a found input which occurs when the snake overlaps the pill as it will need to know when to change the x,y coordinates of the pill to create a "new" one.

There is a counter running continuously until the user eats the pill and then new x,y values are given meaning although it's not truly random it is dependent on how long the user takes to get a pill.

Switch

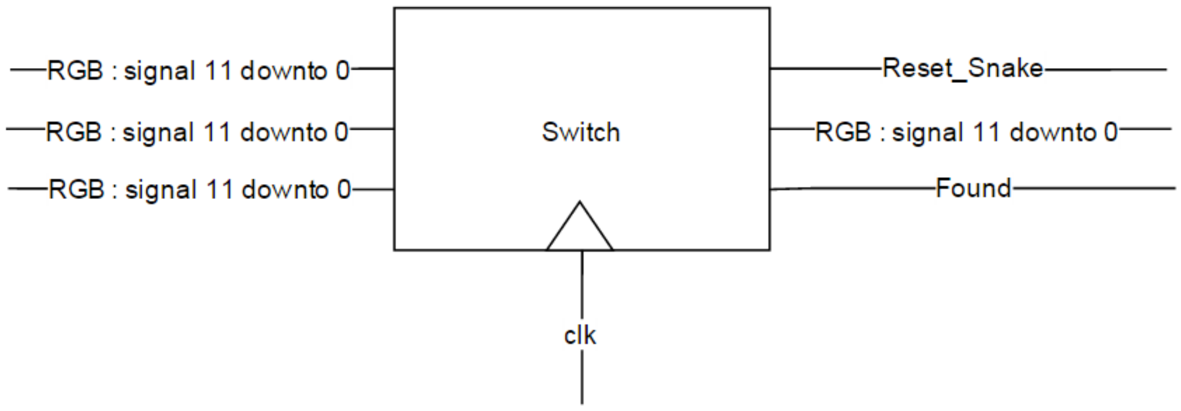

Fig 6 - Switch entity

The switch not only outputs the RGB signals from snake wall and pill but it acts as the feedback to let the graphical entities when an event happens that will reset or increment values within them.

Improvements

The current design uses variables to store the x,y coordinates and direction for each segment each turn also stores x,y coordinates and direction. For a long snake this would require a large amount of memory.

```
Address : in STD_LOGIC_VECTOR;

Read_Write : in STD_LOGIC;

I_O : inout STD_LOGIC_VECTOR;
```

if memory where used a longer snake could be achieved then checking if the snake has hit its self would be possible but is as simple as creating a loop to see if any of the snake segments x,y coordinates are the same.

```
if (snake_head_x = snake_block_n_x) and (snake_head_y = snake_block_n_y) then

        reset <= '1'

else

        reset <= '0';

end if;
```

An alternative way would be to buffer the snake's body and move the tail block of the snake in front of the head of the snake this method would still however require a reasonable amount of memory making it no more complex but perhaps more efficient.

SVN Log

Revision: 14

Author: mbridd

Date: 18 March 2016 08:32:32

Message:

Everything completed and implemented pill entity doesnt properly work unsure as to why

----

```
Added : /Assignment2/Assignment2

Added : /Assignment2/Assignment2/-p

Added : /Assignment2/Assignment2/Assignment2.sim

Added : /Assignment2/Assignment2/Assignment2.srcs

Added : /Assignment2/Assignment2/Assignment2.srcs/constrs_1

Added : /Assignment2/Assignment2/Assignment2.srcs/constrs_1/new

Added : /Assignment2/Assignment2/Assignment2.srcs/constrs_1/new/Basys3.xdc

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/50MHz_clk.vhd

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/Game_pts_display.vhd

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/Pills.vhd

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/Snake.vhd

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/VGA_top_level.vhd

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/Wall_gen.vhd

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/clk_1Hz.vhd

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/clk_1MHz.vhd

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/clk_200Hz.vhd

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/clk_50Hz.vhd

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/diagonal_line.vhd

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/four_digit_units.vhd

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/one_digit.vhd

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/rand_pillx_clk.vhd

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/rand_pilly_clk.vhd
```

Added : /Assignment2/Assignment2/Assignment2.srcs/sources_1/new/vga_controller_800_75.vhd

Added : /Assignment2/Assignment2/Assignment2.xpr

Modified : /Assignment2/Assignment2.srcs/constrs_1/new/Basys3.xdc

Added : /Assignment2/Assignment2.srcs/sources_1/new/four_digits.vhd

Added : /Assignment2/Assignment2.srcs/sources_1/new/one_digit.vhd

Added : /Assignment2/Assignment2.srcs/sources_1/new/one_khz_clk.vhd

Modified : /Assignment2/Assignment2.srcs/sources_1/new/pill.vhd

Added : /Assignment2/Assignment2.srcs/sources_1/new/score.vhd

Modified : /Assignment2/Assignment2.srcs/sources_1/new/snake.vhd

Added : /Assignment2/Assignment2.srcs/sources_1/new/switch.vhd

Modified : /Assignment2/Assignment2.srcs/sources_1/new/top.vhd

Modified : /Assignment2/Assignment2.srcs/sources_1/new/walls.vhd

Modified : /Assignment2/Assignment2.xpr

Revision: 13

Author: mbridd

Date: 07 March 2016 17:26:25

Message:

pill.vhd

- Created a "random" pill generator with a found (reset) input for when the snake and pill output to the display at the same time to generate a new pill

Improvements

- Should check the current location of the snake to ensure a pill isn't placed where the snake already is causing the score counter to increment

----

Added : /Assignment2/Assignment2.srcs/sources_1/new/pill.vhd

Modified : /Assignment2/Assignment2.srcs/sources_1/new/top.vhd

Modified : /Assignment2/Assignment2.xpr

Revision: 12

Author: mbridd

Date: 06 March 2016 14:15:19

Message:

snake.vhd

Implemented:

- Fully implemented a 3 block snake

- Variables used for the 3 snake blocks and the 3 possible turns

- Alters the direction of snake blocks when in turn location

- Inc/Dec each snake segments x,y values according to direction of segment

Improvements

- Needs to have a reset input to restore snake to start position when hit wall

- Should move away from variables to store snake segments x,y,d values and turn x,y,d values and use RAM

snake_memory.vhd

- Attempted to implement RAM to move away from using variables as it will easily allow the full implementation possible in order to store all possible snake segment data and turn data.

Modified : /Assignment2/Assignment2.srcs/sources_1/new/snake.vhd

Added : /Assignment2/Assignment2.srcs/sources_1/new/snake_memory.vhd

Modified : /Assignment2/Assignment2.xpr

Revision: 11

Author: mbridd

Date: 05 March 2016 21:23:03

Message:

Implemented Wall/Snake Modules

Current mode needs to be multiplexed to allow both to be displayed.

Wall Module:

- Displays white between given x,y values

Improvements:

- Should add constants to define limits allow easier modifications

Snake Module:

- Snake moves at a rate of 4Hz

- Able to turn specific direction depending on button pressed

- Does not turn the opposite way

Improvements:

- Need to add snake memory to store multiple snake segments (x_val,y_val,direction)

- Need to add button memory to store multiple button presses (x_val,y_val,btn_direction)

- Need to check if current snake segment is in button memory x,y co-ordinate and if so change that segments direction value.

Added : /Assignment2/.Xil

Modified : /Assignment2/Assignment2.srcs/constrs_1/new/Basys3.xdc

Added : /Assignment2/Assignment2.srcs/sources_1

Added : /Assignment2/Assignment2.srcs/sources_1/new

Added : /Assignment2/Assignment2.srcs/sources_1/new/counter.vhd

Added : /Assignment2/Assignment2.srcs/sources_1/new/snake.vhd

Added : /Assignment2/Assignment2.srcs/sources_1/new/top.vhd

Added : /Assignment2/Assignment2.srcs/sources_1/new/vga_controller_800_60.vhd

Added : /Assignment2/Assignment2.srcs/sources_1/new/vga_controller_800_75.vhd

Added : /Assignment2/Assignment2.srcs/sources_1/new/walls.vhd

Modified : /Assignment2/Assignment2.xpr

Added : /Assignment2/vivado_pid13748.str

Revision: 10

Author: mbridd

Date: 03 March 2016 01:04:07

Message:

testing

Modified : /Assignment2/Assignment2.srcs/constrs_1/new/Basys3.xdc

Modified : /Assignment2/Assignment2.xpr

Code

Top.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

entity top is
    Port ( clk : in STD_LOGIC;
           btnU : in STD_LOGIC;
           btnL : in STD_LOGIC;
           btnC : in STD_LOGIC;
           btnR : in STD_LOGIC;
           btnD : in STD_LOGIC;
           seg : out STD_LOGIC_VECTOR (6 downto 0);
           an : out STD_LOGIC_VECTOR (3 downto 0);
           Hsync : out STD_LOGIC;
           Vsync : out STD_LOGIC;
           vgaRed : out STD_LOGIC_VECTOR (3 downto 0);
           vgaGreen : out STD_LOGIC_VECTOR (3 downto 0);
           vgaBlue: out STD_LOGIC_VECTOR (3 downto 0));
end top;

architecture Behavioral of top is

signal vga_clk : STD_LOGIC;
signal one_hz_clk : STD_LOGIC;
signal four_digit_clk : STD_LOGIC;
signal wall_clk : STD_LOGIC;
signal blank_signal : STD_LOGIC;
signal hcount_signal : UNSIGNED (10 downto 0);
signal vcount_signal : UNSIGNED (10 downto 0);

signal walls_signal : STD_LOGIC_VECTOR(11 downto 0);
signal snake_signal : STD_LOGIC_VECTOR(11 downto 0);
signal pill_signal : STD_LOGIC_VECTOR(11 downto 0);
signal reset_snake_signal : STD_LOGIC;

signal found_signal : STD_LOGIC;

signal digits : STD_LOGIC_VECTOR(15 downto 0);

begin

vga_clk_unit : entity work.counter(Behavioral)
        Port map (  clk => clk,
                    vga_ck => vga_clk,
                    snake_ck => one_hz_clk);

switch_unit : entity work.switch(Behavioral)
        Port map ( pixel_clk => vga_clk,
                   walls => walls_signal,
                   snake => snake_signal,
                   pill => pill_signal,
                        reset_snake => reset_snake_signal,
                        blank => blank_signal,
                        btnC => btnC,
                        found => found_signal,
                        RGB(11 downto 8) => vgaRed,
                        RGB(7 downto 4) => vgaGreen,
                        RGB(3 downto 0) => vgaBlue,
                        score => digits);

one_khz_clk_unit : entity work.one_khz_clk(Behavioral)
        Port map (ck => clk, one_khz => four_digit_clk);

four_digits_unit : entity work.four_digits(Behavioral)
        Port map (d3 => digits(15 downto 12),
                  d2 => digits(11 downto 8),
                  d1 => digits(7 downto 4),
                  d0 => digits(3 downto 0),
                  ck => four_digit_clk, seg => seg, an => an);

pill_unit : entity work.pills(Behavioral)
        Port map (  pixel_clk => vga_clk,
                    found => found_signal,
                    hcount => hcount_signal,
                    vcount => vcount_signal,
                    blank => blank_signal,
                    RGB => pill_signal);

snake_unit : entity work.snake(Behavioral)
        Port map (  snake_clk => clk,
                    up_btn => btnU,
                    left_btn => btnL,
                    right_btn => btnR,
                    down_btn => btnD,
                    hcount => hcount_signal,
                    vcount => vcount_signal,
                    blank => blank_signal,
                    reset_snake => reset_snake_signal,
                    RGB => snake_signal);

walls_unit : entity work.walls(Behavioral)
        Port map (  hcount => hcount_signal,
                    vcount => vcount_signal,
                    blank => blank_signal,
                    RGB => walls_signal);

vga_controller_unit : entity work.vga_controller_800_75(Behavioral)
        Port map (  rst => '0',
                    pixel_clk => vga_clk,
                    HS => Hsync,
                    VS => Vsync,
                    hcount => hcount_signal,
                    vcount => vcount_signal,
                    blank => blank_signal);

end Behavioral;
```

Counter.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
```

```vhdl
entity counter is
    Port ( clk : in STD_LOGIC;
           vga_ck : out STD_LOGIC;
           snake_ck : out STD_LOGIC);
end counter;

architecture Behavioral of counter is

begin

    process(clk)
    variable old : STD_LOGIC;
    begin
        if rising_edge(clk) then
            old := not old;
            vga_ck <= old;
            snake_ck <= '1';
        end if;

        if falling_edge(clk) then
            snake_ck <= '0';
        end if;
    end process;

end Behavioral;
```

Switch.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity switch is
    Port ( pixel_clk : in STD_LOGIC;
           walls : in STD_LOGIC_VECTOR (11 downto 0);
           snake : in STD_LOGIC_VECTOR (11 downto 0);
           pill : in STD_LOGIC_VECTOR (11 downto 0);
           blank : in STD_LOGIC;
           btnC : in STD_LOGIC;
           found : out STD_LOGIC;
           reset_snake : out STD_LOGIC;
           RGB : out STD_LOGIC_VECTOR (11 downto 0);
           score : out STD_LOGIC_VECTOR (15 downto 0));
end switch;

architecture Behavioral of switch is

begin

    process
        variable score0 : UNSIGNED(3 downto 0);
        variable score1 : UNSIGNED(3 downto 0);
    begin

    if ((snake = "111111111111") and (pill = "111111111111")) then
        found <= '1';
        if score0 = 10 then
            score0 := (OTHERS=>'0');
            if score1 = 10 then
                score0 := (OTHERS=>'0');
                score1 := (OTHERS=>'0');
            else
                score1 := score1 + 1;
            end if;
        else
            score0 := score0 + 1;
        end if;
    else
        found <= '0';
    end if;

    score(3 downto 0) <= STD_LOGIC_VECTOR(score0);
    score(7 downto 4) <= STD_LOGIC_VECTOR(score1);

    if btnC = '1' then
        found <= '1';
    else
        found <= '0';
    end if;

    if ((snake = "111111111111") and (walls = "111111111111")) then
        reset_snake <= '1';
        score0 := (OTHERS=>'0');
        score1 := (OTHERS=>'0');
    else
        reset_snake <= '0';
    end if;

    wait until rising_edge(pixel_clk);

        if ((walls = "111111111111") or (snake = "111111111111") or (pill =
"111111111111")) and blank = '0' then
            RGB <= "111111111111";
        else
            RGB <= "000000000000";
        end if;

    end process;

end Behavioral;
```

One_khz_clk.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity one_khz_clk is
    Port ( ck : in STD_LOGIC;
           one_khz : out STD_LOGIC);
end one_khz_clk;

architecture Behavioral of one_khz_clk is

begin
    process
        variable cnt : UNSIGNED(17 downto 0); -- local scope tmr_cnt trig
tmr_clk
        begin
        wait until rising_edge(ck);
            if cnt = 100000 then
                cnt :=(OTHERS=>'0');
                one_khz <= '1';
```

```vhdl
            else
                one_khz <= '0';
                cnt := cnt + 1;
            end if;
    end process;
end Behavioral;
```

Four_digits.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity four_digits is
    Port ( d3 : in STD_LOGIC_VECTOR (3 downto 0);
           d2 : in STD_LOGIC_VECTOR (3 downto 0);
           d1 : in STD_LOGIC_VECTOR (3 downto 0);
           d0 : in STD_LOGIC_VECTOR (3 downto 0);
           ck : in STD_LOGIC;
           seg : out STD_LOGIC_VECTOR (6 downto 0);
           an : out STD_LOGIC_VECTOR (3 downto 0));
end four_digits;

architecture Behavioral of four_digits is

signal digit : STD_LOGIC_VECTOR (3 downto 0);
signal cnt : STD_LOGIC_VECTOR (3 downto 0):="1110";

begin

    seg_dec : entity work.one_digit port map(digit,seg);

    PROCESS
    BEGIN
        if rising_edge(ck) then
            cnt(1) <= cnt(0);
            cnt(2) <= cnt(1);
            cnt(3) <= cnt(2);
            cnt(0) <= cnt(3);
        end if;
    END PROCESS;

    an <= cnt;

    WITH cnt SELECT
        digit <= d3 WHEN "0111",
                 d2 WHEN "1011",
                 d1 WHEN "1101",
                 d0 WHEN "1110",
                 "----" WHEN OTHERS;
end Behavioral;
```

One_digit.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity one_digit is
    Port ( digit : in STD_LOGIC_VECTOR (3 downto 0);
           seg : out STD_LOGIC_VECTOR (6 downto 0));
end one_digit;

architecture Behavioral of one_digit is
```

```vhdl
begin

    WITH digit SELECT
    seg <= "1000000" WHEN "0000", --0
           "1111001" WHEN "0001", --1
           "0100100" WHEN "0010", --2
           "0110000" WHEN "0011", --3
           "0011001" WHEN "0100", --4
           "0010010" WHEN "0101", --5
           "0000010" WHEN "0110", --6
           "1111000" WHEN "0111", --7
           "0000000" WHEN "1000", --8
           "0011000" WHEN "1001", --9
           "1111111" WHEN OTHERS;

end Behavioral;
```

Snake.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity snake is
    Port ( snake_clk : in STD_LOGIC;
           up_btn : in STD_LOGIC;
           left_btn : in STD_LOGIC;
           --center_btn : in STD_LOGIC;
           right_btn : in STD_LOGIC;
           down_btn : in STD_LOGIC;
           hcount : in UNSIGNED (10 downto 0);
           vcount : in UNSIGNED (10 downto 0);
           blank : in STD_LOGIC;
           reset_snake : in STD_LOGIC;
           RGB : out STD_LOGIC_VECTOR (11 downto 0));
end snake;

architecture Behavioral of snake is

begin

    process

    variable cnt : UNSIGNED(26 downto 0);
    variable address : UNSIGNED(1 downto 0);

    constant up : UNSIGNED(1 downto 0) := "11";
    constant left : UNSIGNED(1 downto 0) := "10";
    constant right : UNSIGNED(1 downto 0) := "00";
    constant down : UNSIGNED(1 downto 0) := "01";

    constant snake1_start_x : UNSIGNED(6 downto 0) := "0001001"; --9;
    constant snake1_start_y : UNSIGNED(6 downto 0) := "0001000"; --7;

    constant snake2_start_x : UNSIGNED(6 downto 0) := "0001000"; --8;
    constant snake2_start_y : UNSIGNED(6 downto 0) := "0001000"; --7;

    constant snake3_start_x : UNSIGNED(6 downto 0) := "0000111"; --7;
    constant snake3_start_y : UNSIGNED(6 downto 0) := "0001000"; --7;

    variable snake1_x : UNSIGNED(6 downto 0);
    variable snake1_y : UNSIGNED(6 downto 0);
    variable snake1_d : UNSIGNED(1 downto 0);

    variable snake2_x : UNSIGNED(6 downto 0);
    variable snake2_y : UNSIGNED(6 downto 0);
    variable snake2_d : UNSIGNED(1 downto 0);

    variable snake3_x : UNSIGNED(6 downto 0);
    variable snake3_y : UNSIGNED(6 downto 0);
    variable snake3_d : UNSIGNED(1 downto 0);

    variable turn1_x : UNSIGNED(6 downto 0);
    variable turn1_y : UNSIGNED(6 downto 0);
    variable turn1_d : UNSIGNED(1 downto 0);

    variable turn2_x : UNSIGNED(6 downto 0);
    variable turn2_y : UNSIGNED(6 downto 0);
    variable turn2_d : UNSIGNED(1 downto 0);

    variable turn3_x : UNSIGNED(6 downto 0);
    variable turn3_y : UNSIGNED(6 downto 0);
    variable turn3_d : UNSIGNED(1 downto 0);

    variable direction : UNSIGNED(1 downto 0); -- Right (0); Down (1); Left
(2); Up (3) inital 0

    begin

        if reset_snake = '1' then --reset snake position and clear turn
memory
            snake1_x := snake1_start_x;
            snake1_y := snake1_start_y;
            snake2_x := snake2_start_x;
            snake2_y := snake2_start_y;
            snake3_x := snake3_start_x;
            snake3_y := snake3_start_y;

            snake1_d := right;
            snake2_d := right;
            snake3_d := right;

            turn1_x := (OTHERS=>'0');
            turn1_y := (OTHERS=>'0');
            turn1_d := right;

            turn2_x := (OTHERS=>'0');
            turn2_y := (OTHERS=>'0');
            turn2_d := right;

            turn3_x := (OTHERS=>'0');
            turn3_y := (OTHERS=>'0');
            turn3_d := right;
        end if;

        if (((hcount>=(snake1_x*8)) and (hcount<=(((snake1_x+1)*8)-1))
        and (vcount>=(snake1_y*8)) and (vcount<=(((snake1_y+1)*8)-1)))

        or ((hcount>=(snake2_x*8)) and (hcount<=(((snake2_x+1)*8)-1))
        and (vcount>=(snake2_y*8)) and (vcount<=(((snake2_y+1)*8)-1)))

        or ((hcount>=(snake3_x*8)) and (hcount<=(((snake3_x+1)*8)-1))
        and (vcount>=(snake3_y*8)) and (vcount<=(((snake3_y+1)*8)-1)))) and
blank = '0' then
            RGB <= "111111111111";
        else
            RGB <= "000000000000";
        end if;

        if rising_edge(snake_clk) then
            if cnt = 25000000 then -- 4Hz timer
                cnt := (OTHERS=>'0');

                -- store button press x,y,d --
                if (snake1_d = left or snake1_d = right) and up_btn = '1'
then

                    if address = 2 then
                        turn3_x := snake1_x;
                        turn3_y := snake1_y;
                        turn3_d := up;
                    elsif address = 1 then
                        turn2_x := snake1_x;
                        turn2_y := snake1_y;
                        turn2_d := up;
                    elsif address = 0 then
                        turn1_x := snake1_x;
                        turn1_y := snake1_y;
                        turn1_d := up;
                    end if;
                    address := address + 1;
                end if;

                if (snake1_d = left or snake1_d = right) and down_btn = '1'
then
                    if address = 2 then
                        turn3_x := snake1_x;
                        turn3_y := snake1_y;
                        turn3_d := down;
                    elsif address = 1 then
                        turn2_x := snake1_x;
                        turn2_y := snake1_y;
                        turn2_d := down;
                    elsif address = 0 then
                        turn1_x := snake1_x;
                        turn1_y := snake1_y;
                        turn1_d := down;

                    end if;
                    address := address + 1;
                end if;

                if (snake1_d = up or snake1_d = down) and left_btn = '1'
then
                    if address = 2 then
                        turn3_x := snake1_x;
                        turn3_y := snake1_y;
                        turn3_d := left;
                    elsif address = 1 then
                        turn2_x := snake1_x;
                        turn2_y := snake1_y;
                        turn2_d := left;
                    elsif address = 0 then
```

```
                        turn1_x := snake1_x;
                        turn1_y := snake1_y;
                        turn1_d := left;
                    end if;
                    address := address + 1;
                end if;

                if (snake1_d = up or snake1_d = down) and right_btn = '1'
then
                    if address = 2 then
                        turn3_x := snake1_x;
                        turn3_y := snake1_y;
                        turn3_d := right;
                    elsif address = 1 then
                        turn2_x := snake1_x;
                        turn2_y := snake1_y;
                        turn2_d := right;
                    elsif address = 0 then
                        turn1_x := snake1_x;
                        turn1_y := snake1_y;
                        turn1_d := right;
                    end if;
                    address := address + 1;
                end if;
                -- store button press x,y,d --

                -- snake 1 direction change --
                if (snake1_x = turn1_x) and (snake1_y = turn1_y) then
                    snake1_d := turn1_d;
                end if;

                if (snake1_x = turn2_x) and (snake1_y = turn2_y) then
                    snake1_d := turn2_d;
                end if;

                if (snake1_x = turn3_x) and (snake1_y = turn3_y) then
                    snake1_d := turn3_d;
                end if;
                -- snake 1 direction change --

                -- snake 2 direction change --
                if (snake2_x = turn1_x) and (snake2_y = turn1_y) then
                    snake2_d := turn1_d;
                end if;

                if (snake2_x = turn2_x) and (snake2_y = turn2_y) then
                    snake2_d := turn2_d;
                end if;

                if (snake2_x = turn3_x) and (snake2_y = turn3_y) then
                    snake2_d := turn3_d;
                end if;
                -- snake 2 direction change --

                -- snake 3 direction change --
                if (snake3_x = turn1_x) and (snake3_y = turn1_y) then
                    snake3_d := turn1_d;
                    address := (OTHERS=>'0'); --reset turn memory address
end of snake passed first turn
                end if;
```

```vhdl
if (snake3_x = turn2_x) and (snake3_y = turn2_y) then
    snake3_d := turn2_d;
end if;

if (snake3_x = turn3_x) and (snake3_y = turn3_y) then
    snake3_d := turn3_d;
end if;
-- snake 3 direction change --

-- inc/dec relevant counters for snake 1 --
if snake1_d = right then
    snake1_x := snake1_x + 1; -- inc x count
end if;

if snake1_d = down then
    snake1_y := snake1_y + 1; -- inc y count
end if;

if snake1_d = left then
    snake1_x := snake1_x - 1; -- dec x count
end if;

if snake1_d = up then
    snake1_y := snake1_y - 1; -- dec y count
end if;
-- inc/dec relevant counters for snake 1 --

-- inc/dec relevant counters for snake 2 --
if snake2_d = right then
    snake2_x := snake2_x + 1; -- inc x count
end if;

if snake2_d = down then
    snake2_y := snake2_y + 1; -- inc y count
end if;

if snake2_d = left then
    snake2_x := snake2_x - 1; -- dec x count
end if;

if snake2_d = up then
    snake2_y := snake2_y - 1; -- dec y count
end if;
-- inc/dec relevant counters for snake 2 --

-- inc/dec relevant counters for snake 3 --
if snake3_d = right then
    snake3_x := snake3_x + 1; -- inc x count
end if;

if snake3_d = down then
    snake3_y := snake3_y + 1; -- inc y count
end if;

if snake3_d = left then
    snake3_x := snake3_x - 1; -- dec x count
end if;

if snake3_d = up then
    snake3_y := snake3_y - 1; -- dec y count
end if;
```

```
                -- inc/dec relevant counters for snake 3 --

            else
                cnt := cnt + 1;
            end if;
        end if;

    end process;

end Behavioral;
```

Walls.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity pills is
    Port ( pixel_clk : in STD_LOGIC;
           found : in STD_LOGIC;
           hcount : in UNSIGNED (10 downto 0);
           vcount : in UNSIGNED (10 downto 0);
           blank : in STD_LOGIC;
           RGB : out STD_LOGIC_VECTOR(11 downto 0));
end pills;

architecture Behavioral of Pills is
begin

    process
        variable pill_x : UNSIGNED(10 downto 0);
        variable pill_y : UNSIGNED(10 downto 0);
        variable pill_new_x : UNSIGNED(10 downto 0);
        variable pill_new_y : UNSIGNED(10 downto 0);

        constant min_x : UNSIGNED(10 downto 0) := "00000111000";
        constant min_y : UNSIGNED(10 downto 0) := "00000111000";
        constant max_x : UNSIGNED(10 downto 0) := "01011101000";
        constant max_y : UNSIGNED(10 downto 0) := "01000100000";
    begin

        if (pill_new_x = (max_x-8)) then
            pill_new_x := min_x;
        else
            pill_new_x := pill_new_x+1;
        end if;
        if (pill_new_y = (max_y-8)) then
            pill_new_y := min_y;
        else
            pill_new_y := pill_new_y+1;
        end if;

        wait until rising_edge(pixel_clk);
            if (found = '0') then

                if ((hcount > (pill_x*8)) and (hcount < (((pill_x+1)*8)-1))
and (vcount >= (pill_y*8)) and (vcount <= (((pill_y+1)*8)-1))) then
                    RGB <= "111111111111";
                else
                    RGB <= "000000000000";
                end if;
            else
```

```
                pill_x := pill_new_x;
                pill_y := pill_new_y;
            end if;
    end process;

end Behavioral;
```

Pills.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity pills is
    Port ( pixel_clk : in STD_LOGIC;
           found : in STD_LOGIC;
           hcount : in UNSIGNED (10 downto 0);
           vcount : in UNSIGNED (10 downto 0);
           blank : in STD_LOGIC;
           RGB : out STD_LOGIC_VECTOR(11 downto 0));
end pills;

architecture Behavioral of Pills is
begin

    process
        variable pill_x : UNSIGNED(10 downto 0);
        variable pill_y : UNSIGNED(10 downto 0);
        variable pill_new_x : UNSIGNED(10 downto 0);
        variable pill_new_y : UNSIGNED(10 downto 0);

        constant min_x : UNSIGNED(10 downto 0) := "00000111000";
        constant min_y : UNSIGNED(10 downto 0) := "00000111000";
        constant max_x : UNSIGNED(10 downto 0) := "01011101000";
        constant max_y : UNSIGNED(10 downto 0) := "01000100000";
    begin

        if (pill_new_x = (max_x-8)) then
            pill_new_x := min_x;
        else
            pill_new_x := pill_new_x+1;
        end if;
        if (pill_new_y = (max_y-8)) then
            pill_new_y := min_y;
        else
            pill_new_y := pill_new_y+1;
        end if;

        wait until rising_edge(pixel_clk);
            if (found = '0') then

                if ((hcount > (pill_x*8)) and (hcount < (((pill_x+1)*8)-1))
and (vcount >= (pill_y*8)) and (vcount <= (((pill_y+1)*8)-1))) then
                    RGB <= "111111111111";
                else
                    RGB <= "000000000000";
                end if;
            else
                pill_x := pill_new_x;
                pill_y := pill_new_y;
            end if;
    end process;
end Behavioral;
```