Implementation of Tic-Tac-Toe Game via VGA Interface and VHDL

Appendix

Digital Electronics (ENGS 31/ COSC 56)

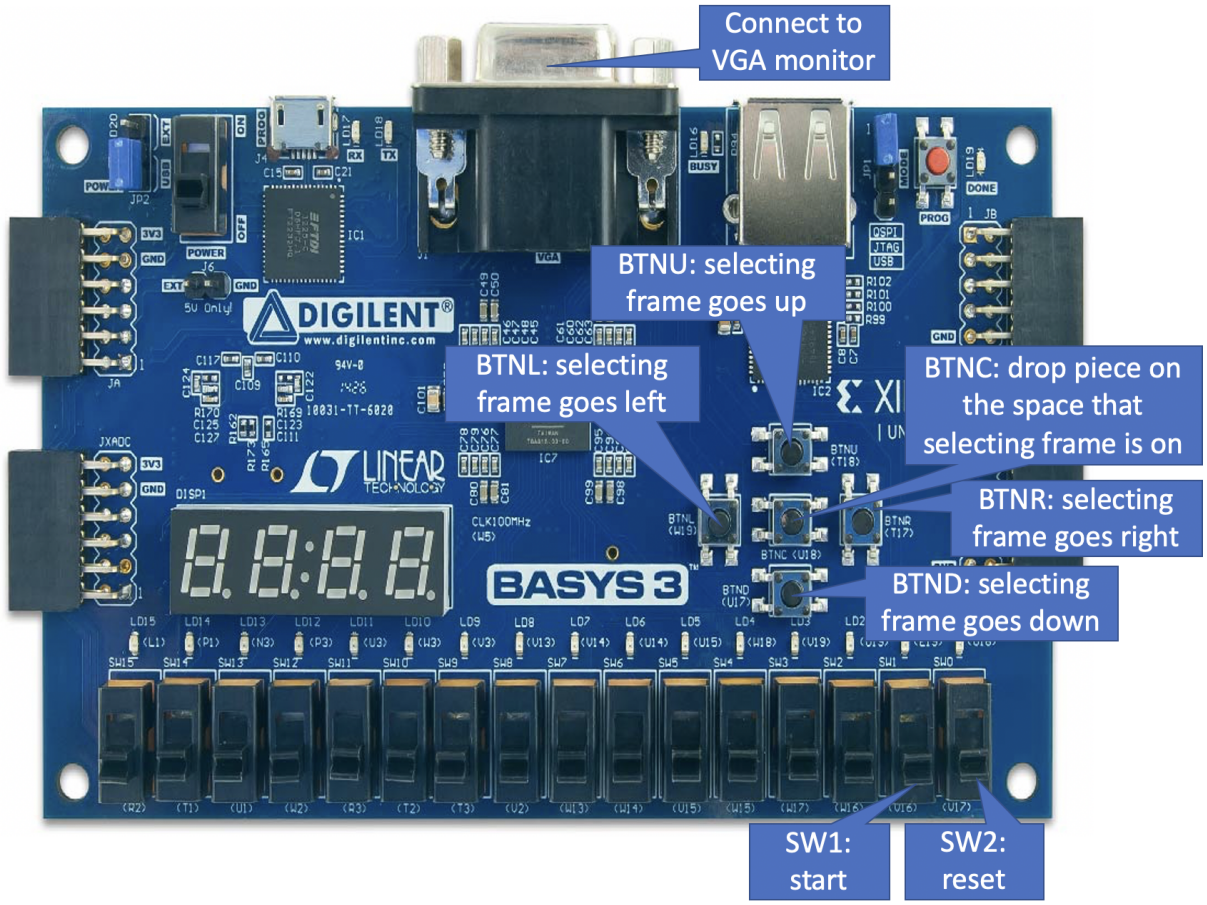
Dartmouth College, Summer 2022

Students: Di Luo, Itish Goel

Instructors: Benjamin Livingston Dobbins, Tad Truex

Catalog

Appendix A: User interface on Basys 3



Appendix B: VGA_driver.vhd

VGA_driver.vhd 8/20/22, 1:33 PM

```
--
--ENGS 31/ CoSc 56
--Final Project: Tic-tac-toe in VGA
--VGA driver
--Implemented based on template: https://www.edaplayground.com/x/6Lwd
--Instructors: Ben Dobbins, Tad Truex
--Student name: Di Luo
--


--
--Library Declarations:
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.math_real.all;
library UNISIM;
use UNISIM.VComponents.all;

--
--Entity Declaration:
--
ENTITY VGA_driver IS
PORT (  vclk_port       : in  STD_LOGIC;  -- 25 MHz clock
    Vsync_port      :   out STD_LOGIC;  -- vertial sync signal to VGA monitor
    Hsync_port      :   out STD_LOGIC;  -- horizontal sync signal to VGA
monitor
        video_on_port   : out STD_LOGIC;  -- accomplish blanking. 1 when
inside display area. 0 when outside display area
    pixel_x_port  : out STD_LOGIC_VECTOR(9 downto 0);   -- current pixel's x
coordinate
        pixel_y_port  : out STD_LOGIC_VECTOR(9 downto 0));  -- current
pixel's y coordinate
end VGA_driver;

--
--Architecture Type:
--


architecture behavior of VGA_driver is
--
```

```
--------------------------------------------------------------------------------
--Signal Declarations:
--
================================================================================
signal H_video_on   : STD_LOGIC := '0';
signal V_video_on   : STD_LOGIC := '0';
--Add your signals here
signal h_counter    : unsigned(9 downto 0) := (others => '0');  -- horizontal
counter
signal v_counter    : unsigned(9 downto 0) := (others => '0');  -- vertical
counter

--VGA Constants (taken directly from VGA Class Notes)
constant left_border  : integer := 48;    -- back porch
constant h_display    : integer := 640; -- active image
constant right_border   : integer := 16;  -- front porch
constant h_retrace    : integer := 96;  -- Hsync pulse width
constant HSCAN      : integer := left_border + h_display
             + right_border + h_retrace; -- =800

constant top_border   : integer := 29;  -- 29
constant v_display    : integer := 480; -- active image
constant bottom_border  : integer := 10;  -- 10
constant v_retrace    : integer := 2;   -- Vsync pulse width
constant VSCAN      : integer := top_border + v_display
             + bottom_border + v_retrace; -- =521

BEGIN

--
================================================================================
--Processes:
--
================================================================================
-- Horizontal counter generating process
h_counter_proc : process(vclk_port)
begin
    if rising_edge(vclk_port) then
        -- counter increments every clk cycle
        if (h_counter < HSCAN - 1) then
            h_counter <= h_counter + 1;
        else
            h_counter <= (others => '0');
        end if;
    end if;
end process h_counter_proc;

-- H_video_on generating process
H_video_on_proc : process(h_counter)
begin
```

```
        -- H_video_on is asserted only in the active display area (left-to-
right)
        if (h_counter < h_display) then
            H_video_on <= '1';
        else
            H_video_on <= '0';
        end if;
end process H_video_on_proc;

-- Hsync generating process
Hsync_proc : process(h_counter)
begin
        -- Hsync is high except for retracing
        if (h_counter >= h_display + right_border) and (h_counter < h_display
+ right_border + h_retrace) then
            Hsync_port <= '0';
        else
            Hsync_port <= '1';
        end if;
end process Hsync_proc;

-- Vertical counter generating process
v_counter_proc : process(vclk_port)
begin
    if rising_edge(vclk_port) then
        -- counter increments every time Hsync is asserted
        if (h_counter = h_display + right_border - 1) then
            if (v_counter < VSCAN - 1) then
                v_counter <= v_counter + 1;
            else
                v_counter <= (others => '0');
            end if;
        end if;
    end if;
end process v_counter_proc;

-- V_video_on generating process
V_video_on_proc : process(v_counter)
begin
    -- V_video_on is asserted only in the active display area (top-to-bottom)
    if (v_counter < v_display) then
        V_video_on <= '1';
    else
        V_video_on <= '0';
    end if;
end process V_video_on_proc;

-- Vsync generating process
Vsync_proc : process(v_counter)
begin
```

```
    -- Vsync is high except for retracing
    if (v_counter >= v_display + bottom_border) and (v_counter < v_display +
bottom_border + v_retrace) then
        Vsync_port <= '0';
    else
        Vsync_port <= '1';
    end if;
end process Vsync_proc;

--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Output Mapping:
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- Only enable video out , when H_video_out and V_video_out are high.
-- It's important to set the output to zero when you aren't actively
displaying video.
-- That's how the monitor determines the black level.
video_on_port <= H_video_on and V_video_on;
pixel_x_port <= std_logic_vector(h_counter);
pixel_y_port <= std_logic_vector(v_counter);

end behavior;

```

Appendix C: game_logic.vhd


```
--
================================================================================
--ENGS 31/ CoSc 56
--Final Project: Tic-tac-toe in VGA
--Game Logic
--Instructors: Ben Dobbins, Tad Truex
--Student name: Di Luo
--
================================================================================

--
================================================================================
--Library Declarations:
--
================================================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.math_real.all;
library UNISIM;
use UNISIM.VComponents.all;

--
================================================================================
--Entity Declaration:
--
================================================================================
entity game_logic is
    port (
        -- system clock
        clk_port    :   in std_logic;   -- 25 MHz clock
        -- inputs from external ports/input conditioning circuits
    start_port  :   in std_logic; -- start a new game
    reset_port  :   in std_logic; -- clear the chessboard
    drop_port   :   in std_logic;   -- put a piece on the selected position
        up_port     :   in std_logic;   -- move selecting frame up
        down_port   :   in std_logic;   -- move selecting frame down
        left_port   :   in std_logic;   -- move selecting frame left
        right_port  :   in std_logic;   -- move selecting frame right
        -- output to pixel generation circuit
        p1_port     :   out std_logic_vector(8 downto 0);   -- indicates the
blocks that player 1 has dropped a chess piece
        p2_port     :   out std_logic_vector(8 downto 0);   -- indicates the
blocks that player 2 has dropped a chess piece
        sf_port     :   out std_logic_vector(3 downto 0);   -- indicates the
location of the selecting frame
        sf_color_port:  out std_logic;                      -- indicates
which player is dropping next
        p1_win_port  :   out std_logic_vector(8 downto 0);  -- indicates the
blocks that make player 1 win
```

```
blocks that make player 1 win
        p2_win_port  :   out std_logic_vector(8 downto 0));  -- indicates the
blocks that make player 2 win
end game_logic;

--
================================================================================
--Architecture Type:
--
================================================================================
architecture behavior of game_logic is
--
================================================================================
--Signal & Constant Declarations:
--
================================================================================
-- finite state machine
type state_type is (sStart, sPick, sTopLeft1, sTopMid1, sTopRight1,
                                    sCtrLeft1, sCtrMid1, sCtrRight1,
                                    sBtmLeft1, sBtmMid1, SBtmRight1,
                                   sTopLeft2, sTopMid2, sTopRight2,
                                    sCtrLeft2, sCtrMid2, sCtrRight2,
                                    sBtmLeft2, sBtmMid2, SBtmRight2,
                    sDrop1, sDrop2, sCheck, sTie, sP1_win_0, sP1_win_1,
sP1_win_2, sP1_win_3, sP1_win_4, sP1_win_5, sP1_win_6, sP1_win_7,
                                                    sP2_win_0, sP2_win_1,
sP2_win_2, sP2_win_3, sP2_win_4, sP2_win_5, sP2_win_6, sP2_win_7);
signal curr_state : state_type := sStart;
signal next_state : state_type;
-- indicates the blocks that the player has dropped a chess piece
signal p1_position, p2_position :   unsigned(8 downto 0) := (others => '0');
-- indicates the location of selecting frame
signal sf_position: unsigned(8 downto 0);
-- counter that counts how many chess pieces have been dropped
signal counter: unsigned(3 downto 0);

signal p1_position_update_en: std_logic;   -- enable the update of position
of player 1's chess pieces
signal p2_position_update_en: std_logic;   -- enable the update of position
of player 2's chess pieces
signal position_update_reset: std_logic;    -- reset position of chess pieces

signal counter_en: std_logic;   -- enable counter to increment
signal counter_reset: std_logic;    -- reset counter
signal counter_pause: std_logic;    -- if player dropped on an occupied block
counter would pause

signal sf_position_idx: std_logic_vector(3 downto 0); -- indexing selecting
frames, 9 blocks so 4 bits

. .
```

```
78 begin
79 --
   ================================================================================
80 --Processes:
81 --
   ================================================================================
82 -- State update:
83 StateUpdate: process(clk_port)
84 begin
85   if rising_edge(clk_port) then
86     curr_state <= next_state;
87   end if;
88 end process StateUpdate;
89
90 -- Next state logic:
91 NextStateLogic: process(curr_state, start_port, reset_port, drop_port,
   up_port, down_port, left_port, right_port, p1_position, p2_position, counter,
   counter_pause)
92 begin
93   next_state <= curr_state;
94   case curr_state is
95         when sStart =>
96             if reset_port = '0' then
97                 if start_port = '1' then
98                     next_state <= sPick;
99                 end if;
100            end if;
101        when sPick =>
102            if reset_port = '1' then
103                next_state <= sStart;
104            elsif counter = "0000" or counter = "0010" or counter = "0100" or
   counter = "0110" or counter = "1000" then --player 1's round
105                next_state <= sCtrMid1;
106            elsif counter = "0001" or counter = "0011" or counter = "0101" or
   counter = "0111" then --player 2's round
107                next_state <= sCtrMid2;
108            else
109                next_state <= sStart;   -- counter only goes from 0 to 9 so
   should never happen
110            end if;
111        when sCtrMid1 =>
112            if reset_port = '1' then
113                next_state <= sStart;
114            else
115                if drop_port = '1' then
116                    next_state <= sDrop1;
117                else
118                    if up_port = '1' then
119                        next_state <= sTopMid1;
120                    elsif down_port = '1' then
```

```
                        next_state <= sBtmMid1;
                    elsif left_port = '1' then
                        next_state <= sCtrLeft1;
                    elsif right_port = '1' then
                        next_state <= sCtrRight1;
                    end if;
                end if;
            end if;
        when sCtrMid2 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop2;
                else
                    if up_port = '1' then
                        next_state <= sTopMid2;
                    elsif down_port = '1' then
                        next_state <= sBtmMid2;
                    elsif left_port = '1' then
                        next_state <= sCtrLeft2;
                    elsif right_port = '1' then
                        next_state <= sCtrRight2;
                    end if;
                end if;
            end if;
        when sCtrLeft1 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop1;
                else
                    if up_port = '1' then
                        next_state <= sTopLeft1;
                    elsif down_port = '1' then
                        next_state <= sBtmLeft1;
                    elsif right_port = '1' then
                        next_state <= sCtrMid1;
                    end if;
                end if;
            end if;
        when sCtrLeft2 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop2;
                else
                    if up_port = '1' then
```

```
                    if up_port = '1' then
                        next_state <= sTopLeft2;
                    elsif down_port = '1' then
                        next_state <= sBtmLeft2;
                    elsif right_port = '1' then
                        next_state <= sCtrMid2;
                    end if;
                end if;
            end if;
        when sCtrRight1 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop1;
                else
                    if up_port = '1' then
                        next_state <= sTopRight1;
                    elsif down_port = '1' then
                        next_state <= sBtmRight1;
                    elsif left_port = '1' then
                        next_state <= sCtrMid1;
                    end if;
                end if;
            end if;
        when sCtrRight2 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop2;
                else
                    if up_port = '1' then
                        next_state <= sTopRight2;
                    elsif down_port = '1' then
                        next_state <= sBtmRight2;
                    elsif left_port = '1' then
                        next_state <= sCtrMid2;
                    end if;
                end if;
            end if;
        when sTopMid1 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop1;
                else
                    if down_port = '1' then
                        next_state <= sCtrMid1;
```

```
                    elsif left_port = '1' then
                        next_state <= sTopLeft1;
                    elsif right_port = '1' then
                        next_state <= sTopRight1;
                    end if;
                end if;
            end if;
        when sTopMid2 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop2;
                else
                    if down_port = '1' then
                        next_state <= sCtrMid2;
                    elsif left_port = '1' then
                        next_state <= sTopLeft2;
                    elsif right_port = '1' then
                        next_state <= sTopRight2;
                    end if;
                end if;
            end if;
        when sTopLeft1 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop1;
                else
                    if down_port = '1' then
                        next_state <= sCtrLeft1;
                    elsif right_port = '1' then
                        next_state <= sTopMid1;
                    end if;
                end if;
            end if;
        when sTopLeft2 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop2;
                else
                    if down_port = '1' then
                        next_state <= sCtrLeft2;
                    elsif right_port = '1' then
                        next_state <= sTopMid2;
                    end if;
                end if;
```

```
            end if;
        when sTopRight1 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop1;
                else
                    if down_port = '1' then
                        next_state <= sCtrRight1;
                    elsif left_port = '1' then
                        next_state <= sTopMid1;
                    end if;
                end if;
            end if;
        when sTopRight2 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop2;
                else
                    if down_port = '1' then
                        next_state <= sCtrRight2;
                    elsif left_port = '1' then
                        next_state <= sTopMid2;
                    end if;
                end if;
            end if;
        when sBtmMid1 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop1;
                else
                    if up_port = '1' then
                        next_state <= sCtrMid1;
                    elsif left_port = '1' then
                        next_state <= sBtmLeft1;
                    elsif right_port = '1' then
                        next_state <= sBtmRight1;
                    end if;
                end if;
            end if;
        when sBtmMid2 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
```

```
                    next_state <= sDrop2;
                else
                    if up_port = '1' then
                        next_state <= sCtrMid2;
                    elsif left_port = '1' then
                        next_state <= sBtmLeft2;
                    elsif right_port = '1' then
                        next_state <= sBtmRight2;
                    end if;
                end if;
            end if;
        when sBtmLeft1 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop1;
                else
                    if up_port = '1' then
                        next_state <= sCtrLeft1;
                    elsif right_port = '1' then
                        next_state <= sBtmMid1;
                    end if;
                end if;
            end if;
        when sBtmLeft2 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop2;
                else
                    if up_port = '1' then
                        next_state <= sCtrLeft2;
                    elsif right_port = '1' then
                        next_state <= sBtmMid2;
                    end if;
                end if;
            end if;
        when sBtmRight1 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop1;
                else
                    if up_port = '1' then
                        next_state <= sCtrRight1;
                    elsif left_port = '1' then
                        next_state <= sBtmMid1;
```

```
                    end if;
                end if;
            end if;
        when sBtmRight2 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                if drop_port = '1' then
                    next_state <= sDrop2;
                else
                    if up_port = '1' then
                        next_state <= sCtrRight2;
                    elsif left_port = '1' then
                        next_state <= sBtmMid2;
                    end if;
                end if;
            end if;
        When sDrop1 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                next_state <= sCheck;
            end if;
        when sDrop2 =>
            if reset_port = '1' then
                next_state <= sStart;
            else
                next_state <= sCheck;
            end if;
        when sCheck =>
            if (p1_position(0) = '1' and p1_position(1) = '1' and
p1_position(2) = '1') then
                next_state <= sP1_win_0;
            elsif (p1_position(3) = '1' and p1_position(4) = '1' and
p1_position(5) = '1') then
                next_state <= sP1_win_1;
            elsif (p1_position(6) = '1' and p1_position(7) = '1' and
p1_position(8) = '1') then
                next_state <= sP1_win_2;
            elsif (p1_position(0) = '1' and p1_position(3) = '1' and
p1_position(6) = '1') then
                next_state <= sP1_win_3;
            elsif (p1_position(1) = '1' and p1_position(4) = '1' and
p1_position(7) = '1') then
                next_state <= sP1_win_4;
            elsif (p1_position(2) = '1' and p1_position(5) = '1' and
p1_position(8) = '1') then
                next_state <= sP1_win_5;
            elsif (p1_position(0) = '1' and p1_position(4) = '1' and
```

```
p1_position(8) = '1') then
                next_state <= sP1_win_6;
            elsif (p1_position(2) = '1' and p1_position(4) = '1' and
p1_position(6) = '1') then
                next_state <= sP1_win_7;
            elsif (p2_position(0) = '1' and p2_position(1) = '1' and
p2_position(2) = '1') then
                next_state <= sP2_win_0;
            elsif (p2_position(3) = '1' and p2_position(4) = '1' and
p2_position(5) = '1') then
                next_state <= sP2_win_1;
            elsif (p2_position(6) = '1' and p2_position(7) = '1' and
p2_position(8) = '1') then
                next_state <= sP2_win_2;
            elsif (p2_position(0) = '1' and p2_position(3) = '1' and
p2_position(6) = '1') then
                next_state <= sP2_win_3;
            elsif (p2_position(1) = '1' and p2_position(4) = '1' and
p2_position(7) = '1') then
                next_state <= sP2_win_4;
            elsif (p2_position(2) = '1' and p2_position(5) = '1' and
p2_position(8) = '1') then
                next_state <= sP2_win_5;
            elsif (p2_position(0) = '1' and p2_position(4) = '1' and
p2_position(8) = '1') then
                next_state <= sP2_win_6;
            elsif (p2_position(2) = '1' and p2_position(4) = '1' and
p2_position(6) = '1') then
                next_state <= sP2_win_7;
            elsif counter = "1001" then -- if counter goes to 9, meaning
there is no free space on the chessboard
                next_state <= sTie;
            else
                next_state <= sPick;
            end if;
        when others =>
            if reset_port = '1' then
                next_state <= sStart;
            end if;
    end case;
end process NextStateLogic;

-- Output logic:
OutputLogic: process(curr_state)
begin
    -- default values
    p1_win_port <= (others => '0');
    p2_win_port <= (others => '0');
    p1_position_update_en <= '0';
    p2_position_update_en <= '0';
```

```
    position_update_reset <= '0';
    counter_en <= '0';
    counter_reset <= '0';
    sf_position_idx <= "1001";
    sf_color_port <= '0';
    -- output in each state
  case curr_state is
    when sStart =>
            position_update_reset <= '1';
            counter_reset <= '1';
        When sPick =>
        When sTopLeft1 =>
            sf_position_idx <= "0000";
        When sTopMid1 =>
            sf_position_idx <= "0001";
        When sTopRight1 =>
            sf_position_idx <= "0010";
        When sCtrLeft1 =>
            sf_position_idx <= "0011";
        When sCtrMid1 =>
            sf_position_idx <= "0100";
        When sCtrRight1 =>
            sf_position_idx <= "0101";
        When sBtmLeft1 =>
            sf_position_idx <= "0110";
        When sBtmMid1 =>
            sf_position_idx <= "0111";
        When SBtmRight1 =>
            sf_position_idx <= "1000";
         When sTopLeft2 =>
            sf_position_idx <= "0000";
            sf_color_port <= '1';
        When sTopMid2 =>
            sf_position_idx <= "0001";
            sf_color_port <= '1';
        When sTopRight2 =>
            sf_position_idx <= "0010";
            sf_color_port <= '1';
        When sCtrLeft2 =>
            sf_position_idx <= "0011";
            sf_color_port <= '1';
        When sCtrMid2 =>
            sf_position_idx <= "0100";
            sf_color_port <= '1';
        When sCtrRight2 =>
            sf_position_idx <= "0101";
            sf_color_port <= '1';
        When sBtmLeft2 =>
            sf_position_idx <= "0110";
            sf_color_port <= '1';
```

```
        When sBtmMid2 =>
            sf_position_idx <= "0111";
            sf_color_port <= '1';
        When sBtmRight2 =>
            sf_position_idx <= "1000";
            sf_color_port <= '1';
        When sDrop1 =>
            p1_position_update_en <= '1';
            counter_en <= '1';
        when sDrop2 =>
            p2_position_update_en <= '1';
            counter_en <= '1';
        when sCheck =>
        when sP1_win_0 =>
            p1_win_port <= "000000111";
        when sP1_win_1 =>
            p1_win_port <= "000111000";
        when sP1_win_2 =>
            p1_win_port <= "111000000";
        when sP1_win_3 =>
            p1_win_port <= "001001001";
        when sP1_win_4 =>
            p1_win_port <= "010010010";
        when sP1_win_5 =>
            p1_win_port <= "100100100";
        when sP1_win_6 =>
            p1_win_port <= "100010001";
        when sP1_win_7 =>
            p1_win_port <= "001010100";
        when sP2_win_0 =>
            p2_win_port <= "000000111";
        when sP2_win_1 =>
            p2_win_port <= "000111000";
        when sP2_win_2 =>
            p2_win_port <= "111000000";
        when sP2_win_3 =>
            p2_win_port <= "001001001";
        when sP2_win_4 =>
            p2_win_port <= "010010010";
        when sP2_win_5 =>
            p2_win_port <= "100100100";
        when sP2_win_6 =>
            p2_win_port <= "100010001";
        when sP2_win_7 =>
            p2_win_port <= "001010100";
        when others => null;
  end case;
end process OutputLogic;

-- Chess piece positions register file:
```

```
    -- chess piece positions register file:
PositionUpdate: process(clk_port, position_update_reset,
p1_position_update_en, p2_position_update_en, sf_position)
begin
    if rising_edge(clk_port) then
        if position_update_reset = '1' then
            p1_position <= (others => '0');
            p2_position <= (others => '0');
        else
            if (p1_position and sf_position) = "000000000" and (p2_position
and sf_position) = "000000000" then -- player 1/2 is not dropping on the spot
that has already been dropped
                counter_pause <= '0';
                if p1_position_update_en = '1' then
                    p1_position <= p1_position + sf_position;
                elsif p2_position_update_en = '1' then
                    p2_position <= p2_position + sf_position;
                end if;
            else
                counter_pause <= '1';
            end if;
        end if;
  end if;
end process PositionUpdate;

-- Counter register file:
CounterUpdate: process(clk_port, counter_reset, counter_en, counter_pause)
begin
    if rising_edge(clk_port) then
        if counter_reset = '1' then
            counter <= "0000";
        elsif counter_pause = '0' then
            if counter_en = '1' then
                counter <= counter + 1;
            end if;
        end if;
    end if;
end process CounterUpdate;

-- Selecting frame register file:
SelectingFrameUpdate: process(clk_port, sf_position_idx)
begin
    if rising_edge(clk_port) then
        case sf_position_idx is
            when "0000" =>
                sf_position <= "000000001";
            when "0001" =>
                sf_position <= "000000010";
            when "0010" =>
                sf_position <= "000000100";
```

```
            when "0011" =>
                sf_position <= "000001000";
            when "0100" =>
                sf_position <= "000010000";
            when "0101" =>
                sf_position <= "000100000";
            when "0110" =>
                sf_position <= "001000000";
            when "0111" =>
                sf_position <= "010000000";
            when "1000" =>
                sf_position <= "100000000";
            when others =>
                null;
        end case;
    end if;
end process SelectingFrameUpdate;

--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Output Mapping:
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
p1_port <= std_logic_vector(p1_position);
p2_port <= std_logic_vector(p2_position);
sf_port <= sf_position_idx;

end behavior;

```

Appendix D: pixel_generation.vhd


```
--
================================================================================
--ENGS 31/ CoSc 56
--Final Project: Tic-tac-toe in VGA
--Pixel Generation Circuit
--Instructors: Ben Dobbins, Tad Truex
--Student name: Di Luo
--
================================================================================

--
================================================================================
--Library Declarations:
--
================================================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.math_real.all;

--
================================================================================
--Entity Declaration:
--
================================================================================
entity pixel_generation is
    port (
        -- inputs from VGA_driver
        pixel_x_port    :   in std_logic_vector(9 downto 0);    -- current
pixel's x coordinate
        pixel_y_port    :   in std_logic_vector(9 downto 0);    -- current
pixel's y coordinate
        video_on_port   :   in std_logic;   -- accomplish blanking. 1 when
inside display area. 0 when outside display area
        -- inputs from game_logic
        p1_port         :   in std_logic_vector(8 downto 0);    -- indicates
the blocks that player 1 has dropped a chess piece
        p2_port         :   in std_logic_vector(8 downto 0);    -- indicates
the blocks that player 2 has dropped a chess piece
        sf_port         :   in std_logic_vector(3 downto 0);    -- indicates
the location of the selecting frame
        sf_color_port   :   in std_logic;                       -- indicates
which player is dropping next
        p1_win_port     :   in std_logic_vector(8 downto 0);    -- indicates
the blocks that make player 1 win
        p2_win_port     :   in std_logic_vector(8 downto 0);    -- indicates
the blocks that make player 2 win
        -- output to VGA monitor
        color_port      :   out std_logic_vector(11 downto 0)); -- 12 bits
for color coding, 4 for each of RGB
```

```
for color coding, 4 for each of RGB
end pixel_generation;

--
================================================================================
--Architecture Type:
--=====================================
=====================================
architecture behavior of pixel_generation is
--
================================================================================
--Signal & Constant Declarations:
--
================================================================================
-- Predefined 12-bit colors
constant BLACK  :   std_logic_vector(11 downto 0) := "000000000000";    --
100% black
constant RED75  :   std_logic_vector(11 downto 0) := "100000000000";    --
75% red
constant BLUE75 :   std_logic_vector(11 downto 0) := "000000001000";    --
75% blue
constant GREEN75:   std_logic_vector(11 downto 0) := "000010000000";    --
75% green
constant WHITE75:   std_logic_vector(11 downto 0) := "110011001100";    --
50% white

-- coordinates of centers in each block
constant x_center_0 : integer := 106;
constant y_center_0 : integer := 80;
constant x_center_1 : integer := 319;
constant y_center_1 : integer := 240;
constant x_center_2 : integer := 533;
constant y_center_2 : integer := 400;

signal row, column  :   unsigned(9 downto 0);

begin
row <= unsigned(pixel_y_port);
column <= unsigned(pixel_x_port);

--
================================================================================
--Processes:
--
================================================================================
pixel_rendering: process(row, column, p1_port, p2_port, sf_port,
sf_color_port, video_on_port, p1_win_port, p2_win_port)
begin
    if video_on_port = '1' then
        if row > 0 and row < 160 and column > 0 and column < 213 then
```

```
            if p1_win_port(0) = '1' then
                color_port <= RED75;
            elsif p2_win_port(0) = '1' then
                color_port <= BLUE75;
            else
                if sf_port = "0000" then
                    if sf_color_port = '0' then
                        color_port <= RED75;
                    else
                        color_port <= BLUE75;
                    end if;
                    if row > 10 and row < 150 and column > 10 and column < 203 then
                        case p1_port(0) is
                            when '1' =>
                                if ((column + y_center_0 = row + x_center_0) or (column + row = x_center_0 + y_center_0)) and column > x_center_0 - 50 and column < x_center_0 + 50 and row > y_center_0 - 50 and row < y_center_0 + 50 then
                                    color_port <= RED75;
                                else
                                    color_port <= WHITE75;
                                end if;
                            when others =>
                                case p2_port(0) is
                                    when '1' =>
                                        if (to_integer(column) - x_center_0)*(to_integer(column) - x_center_0) + (to_integer(row) - y_center_0)*(to_integer(row) - y_center_0) > 49*49 and (to_integer(column) - x_center_0)*(to_integer(column) - x_center_0) + (to_integer(row) - y_center_0)*(to_integer(row) - y_center_0) < 50*50 then
                                            color_port <= BLUE75;
                                        else
                                            color_port <= WHITE75;
                                        end if;
                                    when others =>
                                        color_port <= WHITE75;
                                end case;
                        end case;
                    end if;
                else
                    case p1_port(0) is
                        when '1' =>
                            if ((column + y_center_0 = row + x_center_0) or (column + row = x_center_0 + y_center_0)) and column > x_center_0 - 50 and column < x_center_0 + 50 and row > y_center_0 - 50 and row < y_center_0 + 50 then
                                color_port <= RED75;
                            else
                                color_port <= WHITE75;
```

```
                            end if;
                        when others =>
                            case p2_port(0) is
                                when '1' =>
                                if (to_integer(column) - x_center_0)*
(to_integer(column) - x_center_0) + (to_integer(row) - y_center_0)*
(to_integer(row) - y_center_0) > 49*49 and (to_integer(column) - x_center_0)*
(to_integer(column) - x_center_0) + (to_integer(row) - y_center_0)*
(to_integer(row) - y_center_0) < 50*50 then
                                        color_port <= BLUE75;
                                    else
                                        color_port <= WHITE75;
                                    end if;
                                when others =>
                                    color_port <= WHITE75;
                            end case;
                    end case;
                end if;
            end if;
        elsif row > 0 and row < 160 and column > 213 and column < 426 then
            if p1_win_port(1) = '1' then
                color_port <= RED75;
            elsif p2_win_port(1) = '1' then
                color_port <= BLUE75;
            else
                if sf_port = "0001" then
                    if sf_color_port = '0' then
                        color_port <= RED75;
                    else
                        color_port <= BLUE75;
                    end if;
                    if row > 10 and row < 150 and column > 223 and column <
416 then
                        case p1_port(1) is
                            when '1' =>
                                if ((column + y_center_0 = row + x_center_1)
or (column + row = x_center_1 + y_center_0)) and column > x_center_1 - 50 and
column < x_center_1 + 50 and row > y_center_0 - 50 and row < y_center_0 + 50
then
                                    color_port <= RED75;
                                else
                                    color_port <= WHITE75;
                                end if;
                            when others =>
                                case p2_port(1) is
                                    when '1' =>
                                        if (to_integer(column) - x_center_1)*
(to_integer(column) - x_center_1) + (to_integer(row) - y_center_0)*
(to_integer(row) - y_center_0) > 49*49 and (to_integer(column) - x_center_1)*
(to_integer(column) - x_center_1) + (to_integer(row) - y_center_0)*
```

```
(to_integer(row) - y_center_0) < 50*50 then
                                        color_port <= BLUE75;
                                    else
                                        color_port <= WHITE75;
                                    end if;
                                when others =>
                                    color_port <= WHITE75;
                            end case;
                        end case;
                    end if;
                else
                    case p1_port(1) is
                        when '1' =>
                            if ((column + y_center_0 = row + x_center_1) or
(column + row = x_center_1 + y_center_0)) and column > x_center_1 - 50 and
column < x_center_1 + 50 and row > y_center_0 - 50 and row < y_center_0 + 50
then
                                color_port <= RED75;
                            else
                                color_port <= WHITE75;
                            end if;
                        when others =>
                            case p2_port(1) is
                                when '1' =>
                                    if (to_integer(column) - x_center_1)*
(to_integer(column) - x_center_1) + (to_integer(row) - y_center_0)*
(to_integer(row) - y_center_0) > 49*49 and (to_integer(column) - x_center_1)*
(to_integer(column) - x_center_1) + (to_integer(row) - y_center_0)*
(to_integer(row) - y_center_0) < 50*50 then
                                        color_port <= BLUE75;
                                    else
                                        color_port <= WHITE75;
                                    end if;
                                when others =>
                                    color_port <= WHITE75;
                            end case;
                    end case;
                end if;
            end if;
        elsif row > 0 and row < 160 and column > 426 and column < 640 then
            if p1_win_port(2) = '1' then
                color_port <= RED75;
            elsif p2_win_port(2) = '1' then
                color_port <= BLUE75;
            else
                if sf_port = "0010" then
                    if sf_color_port = '0' then
                        color_port <= RED75;
                    else
                        color_port <= BLUE75;
```

```
                        color_port <= BLUE75;
                    end if;
                    if row > 10 and row < 150 and column > 436 and column <
630 then
                        case p1_port(2) is
                            when '1' =>
                                if ((column + y_center_0 = row + x_center_2)
or (column + row = x_center_2 + y_center_0)) and column > x_center_2 - 50 and
column < x_center_2 + 50 and row > y_center_0 - 50 and row < y_center_0 + 50
then
                                    color_port <= RED75;
                                else
                                    color_port <= WHITE75;
                                end if;
                            when others =>
                                case p2_port(2) is
                                    when '1' =>
                                        if (to_integer(column) - x_center_2)*
(to_integer(column) - x_center_2) + (to_integer(row) - y_center_0)*
(to_integer(row) - y_center_0) > 49*49 and (to_integer(column) - x_center_2)*
(to_integer(column) - x_center_2) + (to_integer(row) - y_center_0)*
(to_integer(row) - y_center_0) < 50*50 then
                                            color_port <= BLUE75;
                                        else
                                            color_port <= WHITE75;
                                        end if;
                                    when others =>
                                        color_port <= WHITE75;
                                end case;
                        end case;
                    end if;
                else
                    case p1_port(2) is
                        when '1' =>
                            if ((column + y_center_0 = row + x_center_2) or
(column + row = x_center_2 + y_center_0)) and column > x_center_2 - 50 and
column < x_center_2 + 50 and row > y_center_0 - 50 and row < y_center_0 + 50
then
                                color_port <= RED75;
                            else
                                color_port <= WHITE75;
                            end if;
                        when others =>
                            case p2_port(2) is
                                when '1' =>
                                    if (to_integer(column) - x_center_2)*
(to_integer(column) - x_center_2) + (to_integer(row) - y_center_0)*
(to_integer(row) - y_center_0) > 49*49 and (to_integer(column) - x_center_2)*
(to_integer(column) - x_center_2) + (to_integer(row) - y_center_0)*
(to_integer(row) - y_center_0) < 50*50 then
```

```
                                        color_port <= BLUE75;
                                    else
                                        color_port <= WHITE75;
                                    end if;
                                when others =>
                                    color_port <= WHITE75;
                            end case;
                        end case;
                    end if;
                end if;
            elsif row > 160 and row < 320 and column > 0 and column < 213 then
                if p1_win_port(3) = '1' then
                    color_port <= RED75;
                elsif p2_win_port(3) = '1' then
                    color_port <= BLUE75;
                else
                    if sf_port = "0011" then
                        if sf_color_port = '0' then
                            color_port <= RED75;
                        else
                            color_port <= BLUE75;
                        end if;
                        if row > 170 and row < 310 and column > 10 and column <
203 then
                            case p1_port(3) is
                                when '1' =>
                                    if ((column + y_center_1 = row + x_center_0)
or (column + row = x_center_0 + y_center_1)) and column > x_center_0 - 50 and
column < x_center_0 + 50 and row > y_center_1 - 50 and row < y_center_1 + 50
then
                                        color_port <= RED75;
                                    else
                                        color_port <= WHITE75;
                                    end if;
                                when others =>
                                    case p2_port(3) is
                                        when '1' =>
                                            if (to_integer(column) - x_center_0)*
(to_integer(column) - x_center_0) + (to_integer(row) - y_center_1)*
(to_integer(row) - y_center_1) > 49*49 and (to_integer(column) - x_center_0)*
(to_integer(column) - x_center_0) + (to_integer(row) - y_center_1)*
(to_integer(row) - y_center_1) < 50*50 then
                                                color_port <= BLUE75;
                                            else
                                                color_port <= WHITE75;
                                            end if;
                                        when others =>
                                            color_port <= WHITE75;
                                    end case;
                            end case;
```

```
                    end if;
                else
                    case p1_port(3) is
                        when '1' =>
                            if ((column + y_center_1 = row + x_center_0) or
(column + row = x_center_0 + y_center_1)) and column > x_center_0 - 50 and
column < x_center_0 + 50 and row > y_center_1 - 50 and row < y_center_1 + 50
then
                                color_port <= RED75;
                            else
                                color_port <= WHITE75;
                            end if;
                        when others =>
                            case p2_port(3) is
                                when '1' =>
                                    if (to_integer(column) - x_center_0)*
(to_integer(column) - x_center_0) + (to_integer(row) - y_center_1)*
(to_integer(row) - y_center_1) > 49*49 and (to_integer(column) - x_center_0)*
(to_integer(column) - x_center_0) + (to_integer(row) - y_center_1)*
(to_integer(row) - y_center_1) < 50*50 then
                                        color_port <= BLUE75;
                                    else
                                        color_port <= WHITE75;
                                    end if;
                                when others =>
                                    color_port <= WHITE75;
                            end case;
                    end case;
                end if;
            end if;
        elsif row > 160 and row < 320 and column > 213 and column < 426 then
            if p1_win_port(4) = '1' then
                color_port <= RED75;
            elsif p2_win_port(4) = '1' then
                color_port <= BLUE75;
            else
                if sf_port = "0100" then
                    if sf_color_port = '0' then
                        color_port <= RED75;
                    else
                        color_port <= BLUE75;
                    end if;
                    if row > 170 and row < 310 and column > 223 and column <
416 then
                        case p1_port(4) is
                            when '1' =>
                                if ((column + y_center_1 = row + x_center_1)
or (column + row = x_center_1 + y_center_1)) and column > x_center_1 - 50 and
column < x_center_1 + 50 and row > y_center_1 - 50 and row < y_center_1 + 50
then
```

```
                                    color_port <= RED75;
                                else
                                    color_port <= WHITE75;
                                end if;
                            when others =>
                                case p2_port(4) is
                                    when '1' =>
                                        if (to_integer(column) - x_center_1)*
(to_integer(column) - x_center_1) + (to_integer(row) - y_center_1)*
(to_integer(row) - y_center_1) > 49*49 and (to_integer(column) - x_center_1)*
(to_integer(column) - x_center_1) + (to_integer(row) - y_center_1)*
(to_integer(row) - y_center_1) < 50*50 then
                                            color_port <= BLUE75;
                                        else
                                            color_port <= WHITE75;
                                        end if;
                                    when others =>
                                        color_port <= WHITE75;
                                end case;
                        end case;
                    end if;
                else
                    case p1_port(4) is
                        when '1' =>
                            if ((column + y_center_1 = row + x_center_1) or
(column + row = x_center_1 + y_center_1)) and column > x_center_1 - 50 and
column < x_center_1 + 50 and row > y_center_1 - 50 and row < y_center_1 + 50
then
                                color_port <= RED75;
                            else
                                color_port <= WHITE75;
                            end if;
                        when others =>
                            case p2_port(4) is
                                when '1' =>
                                    if (to_integer(column) - x_center_1)*
(to_integer(column) - x_center_1) + (to_integer(row) - y_center_1)*
(to_integer(row) - y_center_1) > 49*49 and (to_integer(column) - x_center_1)*
(to_integer(column) - x_center_1) + (to_integer(row) - y_center_1)*
(to_integer(row) - y_center_1) < 50*50 then
                                        color_port <= BLUE75;
                                    else
                                        color_port <= WHITE75;
                                    end if;
                                when others =>
                                    color_port <= WHITE75;
                            end case;
                    end case;
                end if;
```

```
            end if;
        elsif row > 160 and row < 320 and column > 426 and column < 640 then
            if p1_win_port(5) = '1' then
                color_port <= RED75;
            elsif p2_win_port(5) = '1' then
                color_port <= BLUE75;
            else
                if sf_port = "0101" then
                    if sf_color_port = '0' then
                        color_port <= RED75;
                    else
                        color_port <= BLUE75;
                    end if;
                    if row > 170 and row < 310 and column > 436 and column < 630 then
                        case p1_port(5) is
                            when '1' =>
                                if ((column + y_center_1 = row + x_center_2) or (column + row = x_center_2 + y_center_1)) and column > x_center_2 - 50 and column < x_center_2 + 50 and row > y_center_1 - 50 and row < y_center_1 + 50 then
                                    color_port <= RED75;
                                else
                                    color_port <= WHITE75;
                                end if;
                            when others =>
                                case p2_port(5) is
                                    when '1' =>
                                        if (to_integer(column) - x_center_2)*(to_integer(column) - x_center_2) + (to_integer(row) - y_center_1)*(to_integer(row) - y_center_1) > 49*49 and (to_integer(column) - x_center_2)*(to_integer(column) - x_center_2) + (to_integer(row) - y_center_1)*(to_integer(row) - y_center_1) < 50*50 then
                                            color_port <= BLUE75;
                                        else
                                            color_port <= WHITE75;
                                        end if;
                                    when others =>
                                        color_port <= WHITE75;
                                end case;
                        end case;
                    end if;
                else
                    case p1_port(5) is
                        when '1' =>
                            if ((column + y_center_1 = row + x_center_2) or (column + row = x_center_2 + y_center_1)) and column > x_center_2 - 50 and column < x_center_2 + 50 and row > y_center_1 - 50 and row < y_center_1 + 50 then
                                color_port <= RED75;
```

```
                            else
                                color_port <= WHITE75;
                            end if;
                        when others =>
                            case p2_port(5) is
                                when '1' =>
                                    if (to_integer(column) - x_center_2)*
(to_integer(column) - x_center_2) + (to_integer(row) - y_center_1)*
(to_integer(row) - y_center_1) > 49*49 and (to_integer(column) - x_center_2)*
(to_integer(column) - x_center_2) + (to_integer(row) - y_center_1)*
(to_integer(row) - y_center_1) < 50*50 then
                                        color_port <= BLUE75;
                                    else
                                        color_port <= WHITE75;
                                    end if;
                                when others =>
                                    color_port <= WHITE75;
                            end case;
                    end case;
                end if;
            end if;
        elsif row > 320 and row < 480 and column > 0 and column < 213 then
            if p1_win_port(6) = '1' then
                color_port <= RED75;
            elsif p2_win_port(6) = '1' then
                color_port <= BLUE75;
            else
                if sf_port = "0110" then
                    if sf_color_port = '0' then
                        color_port <= RED75;
                    else
                        color_port <= BLUE75;
                    end if;
                    if row > 330 and row < 470 and column > 10 and column <
203 then
                        case p1_port(6) is
                            when '1' =>
                                if ((column + y_center_2 = row + x_center_0)
or (column + row = x_center_0 + y_center_2)) and column > x_center_0 - 50 and
column < x_center_0 + 50 and row > y_center_2 - 50 and row < y_center_2 + 50
then
                                    color_port <= RED75;
                                else
                                    color_port <= WHITE75;
                                end if;
                            when others =>
                                case p2_port(6) is
                                    when '1' =>
                                        if (to_integer(column) - x_center_0)*
(to_integer(column) - x_center_0) + (to_integer(row) - y_center_2)*
```

```
(to_integer(row) - y_center_2) > 49*49 and (to_integer(column) - x_center_0)*
(to_integer(column) - 0) + (to_integer(row) - y_center_2)*(to_integer(row) -
y_center_2) < 50*50 then
                                        color_port <= BLUE75;
                                    else
                                        color_port <= WHITE75;
                                    end if;
                                when others =>
                                    color_port <= WHITE75;
                            end case;
                        end case;
                    end if;
                else
                    case p1_port(6) is
                        when '1' =>
                            if ((column + y_center_2 = row + x_center_0) or
(column + row = x_center_0 + y_center_2)) and column > x_center_0 - 50 and
column < x_center_0 + 50 and row > y_center_2 - 50 and row < y_center_2 + 50
then
                                color_port <= RED75;
                            else
                                color_port <= WHITE75;
                            end if;
                        when others =>
                            case p2_port(6) is
                                when '1' =>
                                    if (to_integer(column) - x_center_0)*
(to_integer(column) - x_center_0) + (to_integer(row) - y_center_2)*
(to_integer(row) - y_center_2) > 49*49 and (to_integer(column) - x_center_0)*
(to_integer(column) - x_center_0) + (to_integer(row) - y_center_2)*
(to_integer(row) - y_center_2) < 50*50 then
                                        color_port <= BLUE75;
                                    else
                                        color_port <= WHITE75;
                                    end if;
                                when others =>
                                    color_port <= WHITE75;
                            end case;
                        end case;
                    end if;
                end if;
            elsif row > 320 and row < 480 and column > 213 and column < 426 then
                if p1_win_port(7) = '1' then
                    color_port <= RED75;
                elsif p2_win_port(7) = '1' then
                    color_port <= BLUE75;
                else
                    if sf_port = "0111" then
                        if sf_color_port = '0' then
                            color_port <= RED75;
```

```
                        color_port <= RED75;
                    else
                        color_port <= BLUE75;
                    end if;
                    if row > 330 and row < 470 and column > 223 and column < 416 then
                        case p1_port(7) is
                            when '1' =>
                                if ((column + y_center_2 = row + x_center_1) or (column + row = x_center_1 + y_center_2)) and column > x_center_1 - 50 and column < x_center_1 + 50 and row > y_center_2 - 50 and row < y_center_2 + 50 then
                                    color_port <= RED75;
                                else
                                    color_port <= WHITE75;
                                end if;
                            when others =>
                                case p2_port(7) is
                                    when '1' =>
                                        if (to_integer(column) - x_center_1)*(to_integer(column) - x_center_1) + (to_integer(row) - y_center_2)*(to_integer(row) - y_center_2) > 49*49 and (to_integer(column) - x_center_1)*(to_integer(column) - x_center_1) + (to_integer(row) - y_center_2)*(to_integer(row) - y_center_2) < 50*50 then
                                            color_port <= BLUE75;
                                        else
                                            color_port <= WHITE75;
                                        end if;
                                    when others =>
                                        color_port <= WHITE75;
                                end case;
                        end case;
                    end if;
                else
                    case p1_port(7) is
                        when '1' =>
                            if ((column + y_center_2 = row + x_center_1) or (column + row = x_center_1 + y_center_2)) and column > x_center_1 - 50 and column < x_center_1 + 50 and row > y_center_2 - 50 and row < y_center_2 + 50 then
                                color_port <= RED75;
                            else
                                color_port <= WHITE75;
                            end if;
                        when others =>
                            case p2_port(7) is
                                when '1' =>
                                    if (to_integer(column) - x_center_1)*(to_integer(column) - x_center_1) + (to_integer(row) - y_center_2)*(to_integer(row) - y_center_2) > 49*49 and (to_integer(column) - x_center_1)*
```

```
(to_integer(column) - x_center_1) + (to_integer(row) - y_center_2)*
(to_integer(row) - y_center_2) < 50*50 then
                                        color_port <= BLUE75;
                                    else
                                        color_port <= WHITE75;
                                    end if;
                                when others =>
                                    color_port <= WHITE75;
                            end case;
                        end case;
                    end if;
                end if;
            elsif row > 320 and row < 480 and column > 426 and column < 640 then
                if p1_win_port(8) = '1' then
                    color_port <= RED75;
                elsif p2_win_port(8) = '1' then
                    color_port <= BLUE75;
                else
                    if sf_port = "1000" then
                        if sf_color_port = '0' then
                            color_port <= RED75;
                        else
                            color_port <= BLUE75;
                        end if;
                        if row > 330 and row < 470 and column > 436 and column <
630 then
                            case p1_port(8) is
                                when '1' =>
                                    if ((column + y_center_2 = row + x_center_2)
or (column + row = x_center_2 + y_center_2)) and column > x_center_2 - 50 and
column < x_center_2 + 50 and row > y_center_2 - 50 and row < y_center_2 + 50
then
                                        color_port <= RED75;
                                    else
                                        color_port <= WHITE75;
                                    end if;
                                when others =>
                                    case p2_port(8) is
                                        when '1' =>
                                            if (to_integer(column) - x_center_2)*
(to_integer(column) - x_center_2) + (to_integer(row) - y_center_2)*
(to_integer(row) - y_center_2) > 49*49 and (to_integer(column) - x_center_2)*
(to_integer(column) - x_center_2) + (to_integer(row) - y_center_2)*
(to_integer(row) - y_center_2) < 50*50 then
                                                color_port <= BLUE75;
                                            else
                                                color_port <= WHITE75;
                                            end if;
                                        when others =>
                                            color_port <= WHITE75;
```

```
                                end case;
                            end case;
                        end if;
                    else
                        case p1_port(8) is
                            when '1' =>
                                if ((column + y_center_2 = row + x_center_2) or
(column + row = x_center_2 + y_center_2)) and column > x_center_2 - 50 and
column < x_center_2 + 50 and row > y_center_2 - 50 and row < y_center_2 + 50
then
                                    color_port <= RED75;
                                else
                                    color_port <= WHITE75;
                                end if;
                            when others =>
                                case p2_port(8) is
                                    when '1' =>
                                        if (to_integer(column) - x_center_2)*
(to_integer(column) - x_center_2) + (to_integer(row) - y_center_2)*
(to_integer(row) - y_center_2) > 49*49 and (to_integer(column) - x_center_2)*
(to_integer(column) - x_center_2) + (to_integer(row) - y_center_2)*
(to_integer(row) - y_center_2) < 50*50 then
                                            color_port <= BLUE75;
                                        else
                                            color_port <= WHITE75;
                                        end if;
                                    when others =>
                                        color_port <= WHITE75;
                                end case;
                        end case;
                    end if;
                end if;
            else
                color_port <= BLACK;
            end if;
        else
            color_port <= BLACK;
        end if;
end process pixel_rendering;

end behavior;
```

Appendix E: system_clock_generator.vhd


```
--
================================================================================
--ENGS 31/ CoSc 56
--Final Project: Tic-tac-toe in VGA
--System Clock Generator
--Source: Lab 6
--Instructors: Ben Dobbins, Tad Truex
--Student name: Di Luo
--
================================================================================

--
================================================================================
--Library Declarations:
--
================================================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.math_real.all;
library UNISIM;
use UNISIM.VComponents.all;

--
================================================================================
--Entity Declaration:
--
================================================================================
entity system_clock_generator is
    generic (CLOCK_DIVIDER_RATIO : integer);
  port (
        input_clk_port    : in std_logic;
        system_clk_port     : out std_logic;
    fwd_clk_port    : out std_logic);
end system_clock_generator;

--
================================================================================
--Architecture Type:
--
================================================================================
architecture behavioral_architecture of system_clock_generator is
--
================================================================================
--Signal Declarations:
--
================================================================================
--CONSTANT FOR SYNTHESIS:
constant CLOCK_DIVIDER_TC: integer := CLOCK_DIVIDER_RATIO/2;
  CONSTANT FOR SIMULATION:
```

```
--CONSTANT FOR SIMULATION:
--constant CLOCK_DIVIDER_TC: integer := 5;

--Automatic register sizing:
constant COUNT_LEN          : integer := integer(ceil( log2(
real(CLOCK_DIVIDER_TC) ) ));
signal system_clk_divider_counter : unsigned(COUNT_LEN-1 downto 0) := (others
=> '0');
signal system_clk_tog       : std_logic := '0';
signal system_clk                   : std_logic := '0';


begin

--
================================================================================
--Processes:
--
================================================================================
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Clock (frequency) Divider):
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Clock_divider: process(input_clk_port)
begin
  if rising_edge(input_clk_port) then
      if system_clk_divider_counter = CLOCK_DIVIDER_TC-1 then     -- Counts to
1/2 clk period
        system_clk_tog <= NOT(system_clk_tog);                  -- T flip flop
      system_clk_divider_counter <= (others => '0');        -- Reset
    else
      system_clk_divider_counter <= system_clk_divider_counter + 1; -- Count
up
    end if;
  end if;
end process Clock_divider;

--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- Clock buffer for the system clock
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- The BUFG component puts the system clock onto the FPGA clocking network
system_clock_buffer: BUFG
    port map (
    I => system_clk_tog,
        O => system_clk);
  system_clk_port <= system_clk;

```

```
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- Clock Forwarding
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

clock_forwarding_ODDR : ODDR
generic map(
  DDR_CLK_EDGE => "SAME_EDGE", -- "OPPOSITE_EDGE" or "SAME_EDGE"
  INIT => '0', -- Initial value for Q port ('1' or '0')
  SRTYPE => "SYNC") -- Reset Type ("ASYNC" or "SYNC")
port map (
  Q => fwd_clk_port, -- 1-bit DDR output
  C => system_clk, -- 1-bit clock input
  CE => '1', -- 1-bit clock enable input
  D1 => '1', -- 1-bit data input (positive edge)
  D2 => '0', -- 1-bit data input (negative edge)
  R => '0', -- 1-bit reset input
  S => '0' -- 1-bit set input
);

end behavioral_architecture;
```

Appendix F: input_conditioning.vhd


```
--
================================================================================
--ENGS 31/ CoSc 56
--Final Project: Tic-tac-toe in VGA
--Input Conditioning
--Source: Lab 5
--Instructors: Ben Dobbins, Tad Truex
--Student name: Di Luo
--
================================================================================

--
================================================================================
--Library Declarations:
--
================================================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.math_real.all;
library UNISIM;
use UNISIM.VComponents.all;

--
================================================================================
--Entity Declaration:
--
================================================================================
entity button_interface is
    Port( clk_port            : in  std_logic;
      button_port         : in  std_logic;
      button_db_port      : out std_logic;
      button_mp_port      : out std_logic);
end button_interface;

--
================================================================================
--Architecture
--
================================================================================
architecture Behavioral of button_interface is
--
================================================================================
--Signal Declarations
--
================================================================================
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Synchronizer
```

```
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
signal double_flop_synchronizer   : std_logic_vector(1 downto 0) := "00";
signal synchronized_button_press  : std_logic := '0';

--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Debouncer
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
type state_type is (BUTTON_OUTPUT_LOW,  LOW_TO_HIGH_TRANSITION,
          BUTTON_OUTPUT_HIGH, HIGH_TO_LOW_TRANSITION);
signal current_state     : state_type := BUTTON_OUTPUT_LOW;
signal next_state        : state_type;
signal timeout_counter   : unsigned(8 downto 0) := (others => '0');
signal reset             : std_logic := '0';
signal timeout           : std_logic := '0';
constant MAXCOUNT        : integer := 500;
signal debounced_input   : std_logic := '0';

--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Monopulser
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
signal mp_delay_register : std_logic := '0';

--
================================================================================
--Processes
--
================================================================================
begin
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Double-Flop Synchronizer:
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Prevents metastability on the input by passing the raw button press through
a
--double flop synchronizer.
synchronize: process(clk_port, double_flop_synchronizer)
begin
    if rising_edge(clk_port) then
        double_flop_synchronizer(1) <= button_port;
    double_flop_synchronizer(0) <= double_flop_synchronizer(1);
    end if;
    synchronized_button_press <= double_flop_synchronizer(0);
end process;

```

```
--
--------------------------------------------------------------------------------
--Debouncer:
--
--------------------------------------------------------------------------------
--State Update Logic
state_update: process(clk_port)
begin
    if rising_edge(clk_port) then
        current_state <= next_state;
    end if;
end process;

--Next State Logic
next_state_logic: process(current_state, synchronized_button_press, timeout)
begin
    next_state <= current_state;
    case current_state is
        when BUTTON_OUTPUT_LOW =>
            if synchronized_button_press = '1' then
                next_state <= LOW_TO_HIGH_TRANSITION;
            end if;

        when LOW_TO_HIGH_TRANSITION =>
            if synchronized_button_press = '0' then
                next_state <= BUTTON_OUTPUT_LOW;
            elsif timeout = '1' then
                next_state <= BUTTON_OUTPUT_HIGH;
            end if;

        when BUTTON_OUTPUT_HIGH =>
            if synchronized_button_press = '0' then
                next_state <= HIGH_TO_LOW_TRANSITION;
            end if;

        when HIGH_TO_LOW_TRANSITION =>
            if synchronized_button_press = '1' then
                next_state <= BUTTON_OUTPUT_HIGH;
            elsif timeout = '1' then
                next_state <= BUTTON_OUTPUT_LOW;
            end if;
    end case;
end process;

--Output Logic
output_logic: process(current_state)
begin
    case current_state is
        when BUTTON_OUTPUT_LOW =>
            reset             <= '1';
```

```
            debounced_input   <= '0';

        when LOW_TO_HIGH_TRANSITION =>
            reset             <= '0';
            debounced_input   <= '0';

        when BUTTON_OUTPUT_HIGH =>
            reset           <= '1';
            debounced_input   <= '1';

        when HIGH_TO_LOW_TRANSITION =>
            reset           <= '0';
            debounced_input   <= '1';
    end case;
end process;
button_db_port <= debounced_input;

--Timer Subroutine
timer: process(clk_port)
begin
    if rising_edge(clk_port) then
        if reset = '1' then
            timeout_counter <= (others => '0');
        else
            timeout_counter <= timeout_counter + 1;
        end if;
    end if;
end process;

--Timeout Flag
timeout_flag: process(timeout_counter)
begin
    if timeout_counter = MAXCOUNT - 1 then
        timeout <= '1';
    else
        timeout <= '0';
    end if;
end process;

--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Monopulser:
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--A monopulsed output is an output that is high for one clock cycle.
monopulser: process(clk_port, debounced_input, mp_delay_register)
begin
  if rising_edge(clk_port) then
    mp_delay_register <= debounced_input;
  end if;
```

```
  end if;
  button_mp_port <= debounced_input and not(mp_delay_register);
end process monopulser;
end Behavioral;

```

Appendix G: TicTacToe_shell.vhd


```
--
================================================================================
--ENGS 31/ CoSc 56
--Final Project: Tic-tac-toe in VGA
--Tic-tac-toe Top Level
--Instructors: Ben Dobbins, Tad Truex
--Student name: Di Luo
--
================================================================================

--
================================================================================
--Library Declarations:
--
================================================================================
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;     -- needed for arithmetic
use ieee.math_real.all;       -- needed for automatic register sizing
library UNISIM;           -- needed for the BUFG component
use UNISIM.Vcomponents.ALL;

--
================================================================================
--Entity Declaration:
--
================================================================================
entity TicTacToe_shell is
    port (
        clk_ext_port    : in std_logic;                     -- ext 100 MHz
clock
        start_ext_port  : in std_logic;                     -- switch[1] that
starts the game
        reset_ext_port  : in std_logic;                     -- switch[0] that
resets the game
        drop_ext_port   : in std_logic;                     -- central button
that drops the chess piece
        btnU_ext_port   : in std_logic;                     -- up button that
moves selecting frame up
        btnD_ext_port   : in std_logic;                     -- down button
that moves selecting frame down
        btnL_ext_port   : in std_logic;                     -- left button
that moves selecting frame left
        btnR_ext_port   : in std_logic;                     -- right button
that moves selecting frame right
        color_ext_port  : out std_logic_vector(11 downto 0);-- 12 bits for
color coding, 4 for each of RGB
        Hsync_ext_port  : out std_logic;                    -- horizontal
sync signal to VGA monitor
        Vsync_ext_port  : out std_logic                     -- vertical sync
```

```
        vsync_ext_port  : out std_logic                     -- vertical sync
signal to VGA monitor
    );
end TicTacToe_shell;

--============================================================
--Architecture + Component Declarations
--============================================================
architecture behavior of TicTacToe_shell is
--System Clock Generation:
component system_clock_generator is
  generic (
     CLOCK_DIVIDER_RATIO : integer);
    port (
        input_clk_port    : in  std_logic;    -- 100 MHz clk of FPGA
        system_clk_port     : out std_logic;    -- 25 MHz clk
    fwd_clk_port    : out std_logic);   -- unused, open
end component;

--Input Conditioning:
component button_interface is
    Port( clk_port            : in  std_logic;  -- system clk
      button_port         : in  std_logic;  -- external buttons
      button_db_port      : out std_logic;  -- debounced button output
      button_mp_port      : out std_logic); -- monopulse button output
end component;

--VGA driver:
component VGA_driver is
    port (  vclk_port       : in  STD_LOGIC;  -- 25 MHz clock
            Vsync_port      :   out STD_LOGIC;  -- vertial sync signal to VGA
monitor
            Hsync_port      :   out STD_LOGIC;  -- horizontal sync signal to
VGA monitor
            video_on_port   : out STD_LOGIC;  -- accomplish blanking. 1 when
inside display area. 0 when outside display area
            pixel_x_port  : out STD_LOGIC_VECTOR(9 downto 0);   -- current
pixel's x coordinate
            pixel_y_port  : out STD_LOGIC_VECTOR(9 downto 0));  -- current
pixel's y coordinate
end component;

-- Game logic circuit
component game_logic is
    port (  clk_port    :   in std_logic;   -- 25 MHz clock
            start_port  :   in std_logic; -- start a new game
            reset_port  :   in std_logic; -- clear the chessboard
            drop_port   :   in std_logic;   -- put a piece on the chessboard
            up_port     :   in std_logic;   -- move selecting frame up
            down_port   :   in std_logic;   -- move selecting frame down
```

```
            left_port   :   in std_logic;   -- move selecting frame left
            right_port  :   in std_logic;   -- move selecting frame right
            p1_port     :   out std_logic_vector(8 downto 0);   -- indicates
the blocks that player 1 has dropped a chess piece
            p2_port     :   out std_logic_vector(8 downto 0);   -- indicates
the blocks that player 2 has dropped a chess piece
            sf_port     :   out std_logic_vector(3 downto 0);   -- indicates
the location of the selecting frame
            sf_color_port:  out std_logic;                      -- indicates
which player is dropping next
            p1_win_port :   out std_logic_vector(8 downto 0);    -- indicates
the blocks that make player 1 win
            p2_win_port :   out std_logic_vector(8 downto 0));   -- indicates
the blocks that make player 2 win
end component;

--Pixel generation circuit:
component pixel_generation is
    port (  pixel_x_port    :   in std_logic_vector(9 downto 0);    --
current pixel's x coordinate
            pixel_y_port    :   in std_logic_vector(9 downto 0);    --
current pixel's y coordinate
            video_on_port   :   in std_logic;   -- accomplish blanking. 1
when inside display area. 0 when outside display area
            p1_port         :   in std_logic_vector(8 downto 0);
            p2_port         :   in std_logic_vector(8 downto 0);
            sf_port         :   in std_logic_vector(3 downto 0);
            sf_color_port   :   in std_logic;                       --
indicates which player is dropping next
            p1_win_port     :   in std_logic_vector(8 downto 0);    --
indicates the blocks that make player 1 win
            p2_win_port     :   in std_logic_vector(8 downto 0);    --
indicates the blocks that make player 2 win
            color_port      :   out std_logic_vector(11 downto 0)); -- 12
bits for color coding, 4 for each of RGB
end component;

--============================================================
--Local Signal Declaration
--============================================================
-- clocking
signal system_clk : std_logic;
-- VGA to pixel_gene
signal x: std_logic_vector(9 downto 0);
signal y: std_logic_vector(9 downto 0);
signal video_on: std_logic;
-- input conditioning to game logic
signal drop: std_logic;
signal up: std_logic;
signal down: std_logic;
```

```
115 signal left: std_logic;
116 signal right: std_logic;
117 -- game logic to pixel generation
118 signal p1: std_logic_vector(8 downto 0);
119 signal p2: std_logic_vector(8 downto 0);
120 signal sf: std_logic_vector(3 downto 0);
121 signal sf_color: std_logic;
122 signal p1_win: std_logic_vector(8 downto 0);
123 signal p2_win: std_logic_vector(8 downto 0);
124
125 begin
126
127 --============================================================
128 --Port Mapping + Processes:
129 --============================================================
130 --System Clock Generation:
131 --Wire the system clock generator into the shell with a port map
132 clocking: system_clock_generator
133 generic map(
134   CLOCK_DIVIDER_RATIO => 4)
135 port map(
136   input_clk_port    => clk_ext_port,
137   system_clk_port   => system_clk,
138   fwd_clk_port    => open);
139
140 --Wire the input conditioning block into the shell with a port map:
141 drop_monopulse: button_interface
142 port map(
143     clk_port            => system_clk,
144     button_port         => drop_ext_port,
145     button_db_port      => open,
146     button_mp_port      => drop);
147
148 up_monopulse: button_interface
149 port map(
150     clk_port            => system_clk,
151     button_port         => btnU_ext_port,
152     button_db_port      => open,
153     button_mp_port      => up);
154
155 down_monopulse: button_interface
156 port map(
157     clk_port            => system_clk,
158     button_port         => btnD_ext_port,
159     button_db_port      => open,
160     button_mp_port      => down);
161
162 left_monopulse: button_interface
163 port map(
164     clk_port            => system_clk,
```

```
    button_port         => btnL_ext_port,
    button_db_port      => open,
    button_mp_port      => left);

right_monopulse: button_interface
port map(
    clk_port            => system_clk,
    button_port         => btnR_ext_port,
    button_db_port      => open,
    button_mp_port      => right);

game_logic_control: game_logic
port map(
    clk_port            => system_clk,
    start_port          => start_ext_port,
    reset_port          => reset_ext_port,
    drop_port           => drop,
    up_port             => up,
    down_port           => down,
    left_port           => left,
    right_port          => right,
    p1_port             => p1,
    p2_port             => p2,
    sf_port             => sf,
    sf_color_port       => sf_color,
    p1_win_port         => p1_win,
    p2_win_port         => p2_win);

vga_control: VGA_driver
port map(
    vclk_port       => system_clk,
    Vsync_port      => Vsync_ext_port,
    Hsync_port      => Hsync_ext_port,
    video_on_port   => video_on,
    pixel_x_port  => x,
    pixel_y_port  => y);

pixel_generation_datapath: pixel_generation
port map(
    pixel_x_port    => x,
    pixel_y_port    => y,
    video_on_port   => video_on,
    p1_port         => p1,
    p2_port         => p2,
    sf_port         => sf,
    sf_color_port   => sf_color,
    p1_win_port     => p1_win,
    p2_win_port     => p2_win,
    color_port      => color_ext_port);
```

```
215 end behavior;
216
```

Appendix H: game_logic_tb.vhd


```
--
================================================================================
--ENGS 31/ CoSc 56
--Final Project: Tic-tac-toe in VGA
--Game Logic Circuit Testbench
--Instructors: Ben Dobbins, Tad Truex
--Student name: Di Luo
--
================================================================================

--
================================================================================
--Library Declarations:
--
================================================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

--============================================================
--Testbench Entity Declaration
--============================================================
entity game_logic_tb is
end game_logic_tb;

--============================================================
--Testbench declarations
--============================================================
architecture testbench of game_logic_tb is

--
================================================================================
--Component Declaration:
--
================================================================================
component game_logic is
    Port (
    -- system clock
        clk_port    :   in std_logic;   -- 25 MHz clock
        -- inputs from external ports/input conditioning circuits
    start_port  :   in std_logic; -- start a new game
    reset_port  :   in std_logic; -- clear the chessboard
    drop_port   :   in std_logic;   -- put a piece on the selected position
        up_port     :   in std_logic;   -- move selecting frame up
        down_port   :   in std_logic;   -- move selecting frame down
        left_port   :   in std_logic;   -- move selecting frame left
        right_port  :   in std_logic;   -- move selecting frame right
        -- output to pixel generation circuit
        p1_port     :   out std_logic_vector(8 downto 0);   -- indicates the
blocks that player 1 has dropped a chess piece
```

```
blocks that player 1 has dropped a chess piece
        p2_port     :   out std_logic_vector(8 downto 0);   -- indicates the
blocks that player 2 has dropped a chess piece
        sf_port     :   out std_logic_vector(3 downto 0);   -- indicates the
location of the selecting frame
        sf_color_port:  out std_logic;                      -- indicates
which player is dropping next
        p1_win_port  :   out std_logic_vector(8 downto 0);  -- indicates the
blocks that make player 1 win
        p2_win_port  :   out std_logic_vector(8 downto 0));  -- indicates the
blocks that make player 2 win
end component;

--============================================================
--Local Signal Declaration
--============================================================
--Simulated input signals
signal clk: std_logic := '0';
signal start: std_logic := '0';
signal reset: std_logic := '0';
signal drop: std_logic := '0';
signal up: std_logic := '0';
signal down: std_logic := '0';
signal left: std_logic := '0';
signal right: std_logic := '0';
--Simulated output signals
signal p1: std_logic_vector(8 downto 0) := (others => '0');
signal p2: std_logic_vector(8 downto 0) := (others => '0');
signal sf: std_logic_vector(3 downto 0) := "0000";
signal sf_color: std_logic := '0';
signal p1_win: std_logic_vector(8 downto 0) := (others => '0');
signal p2_win: std_logic_vector(8 downto 0) := (others => '0');
-- Clock period definitions
constant clk_period: time := 40ns;  -- 25 MHz clock

begin

--============================================================
--Port Mapping + Processes:
--============================================================
-- wire the game logic circuit with local signals
dut: game_logic port map (
    clk_port    => clk,
    start_port  => start,
    reset_port  => reset,
    drop_port   => drop,
    up_port     => up,
    down_port   => down,
    left_port   => left,
    right_port  => right,
```

```
    p1_port     => p1,
    p2_port     => p2,
    sf_port     => sf,
    sf_color_port => sf_color,
    p1_win_port => p1_win,
    p2_win_port => p2_win);

--Timing:
clk_process: process
begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
end process;

--Stimulus Process:
stim_proc: process
begin
    wait for 2*clk_period;   -- hold in sStart state for 2 clk cycles

    --
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  --Test a normal game:
  --
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    -- press start button
    start <= '1';
    wait for clk_period;
    start <= '0';
    wait for clk_period;

    -- Step 1 (player 1)
    -- move left from Central Mid to Central Left
    left <= '1';
    wait for clk_period;
    left <= '0';
    wait for 4*clk_period;

    -- move up to Top Left
    up <= '1';
    wait for clk_period;
    up <= '0';
    wait for 4*clk_period;

    -- drop at Top Left
    drop <= '1';
    wait for clk_period;
    drop <= '0';
    wait for 4*clk_period;
```

```
    -- X__
    -- ___
    -- ___

    -- Step 2 (player 2)
    -- move right from Central Mid to Central Right
    right <= '1';
    wait for clk_period;
    right <= '0';
    wait for 4*clk_period;

    -- move up to Top Right
    up <= '1';
    wait for clk_period;
    up <= '0';
    wait for 4*clk_period;

    -- drop at Top Right
    drop <= '1';
    wait for clk_period;
    drop <= '0';
    wait for 4*clk_period;
    -- X_O
    -- ___
    -- ___

    -- Step 3 (player 1)
    -- move left from Central Mid to Central Left
    left <= '1';
    wait for clk_period;
    left <= '0';
    wait for 4*clk_period;

    -- drop at Central Left
    drop <= '1';
    wait for clk_period;
    drop <= '0';
    wait for 4*clk_period;
    -- X_O
    -- X__
    -- ___

    -- Step 4 (player 2)
    -- move right from Central Mid to Central Right
    right <= '1';
    wait for clk_period;
    right <= '0';
    wait for 4*clk_period;

    -- drop at Central Right
```

```
    drop <= '1';
    wait for clk_period;
    drop <= '0';
    wait for 4*clk_period;
    -- X_O
    -- X_O
    -- ___

    -- Step 5 (player 1)
    -- move left from Central Mid to Central Left
    left <= '1';
    wait for clk_period;
    left <= '0';
    wait for 4*clk_period;

    -- move down to Bottom Left
    down <= '1';
    wait for clk_period;
    down <= '0';
    wait for 4*clk_period;

    -- drop at Bottom Left, 1st column all X, player 1 wins
    drop <= '1';
    wait for clk_period;
    drop <= '0';
    wait for 8*clk_period;
    -- X_O
    -- X_0
    -- X__

    reset <= '1';
    wait for clk_period;
    reset <= '0';
    wait for 8*clk_period;

    --
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  --Test a tie game:
  --
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    -- press start button
    start <= '1';
    wait for clk_period;
    start <= '0';
    wait for clk_period;

    -- Step 1 (player 1)
    -- move left from Central Mid to Central Left
    left <= '1';
```

```
    wait for clk_period;
    left <= '0';
    wait for 4*clk_period;

    -- move up to Top Left
    up <= '1';
    wait for clk_period;
    up <= '0';
    wait for 4*clk_period;

    -- drop at Top Left
    drop <= '1';
    wait for clk_period;
    drop <= '0';
    wait for 4*clk_period;
    -- X__
    -- ___
    -- ___

    -- Step 2 (player 2)
    -- move up to Top Mid
    up <= '1';
    wait for clk_period;
    up <= '0';
    wait for 4*clk_period;

    -- drop at Top Mid
    drop <= '1';
    wait for clk_period;
    drop <= '0';
    wait for 4*clk_period;
    -- XO_
    -- ___
    -- ___

    -- Step 3 (player 1)
    -- move left from Central Mid to Central Left
    left <= '1';
    wait for clk_period;
    left <= '0';
    wait for 4*clk_period;

    -- drop at Central Left
    drop <= '1';
    wait for clk_period;
    drop <= '0';
    wait for 4*clk_period;
    -- XO_
    -- X__
    -- ___
```

```
283
284     -- Step 4 (player 2)
285     -- drop at Central Mid
286     drop <= '1';
287     wait for clk_period;
288     drop <= '0';
289     wait for 4*clk_period;
290     -- XO_
291     -- XO_
292     -- ___
293
294     -- Step 5 (player 1)
295     -- move down to Bottom Mid
296     down <= '1';
297     wait for clk_period;
298     down <= '0';
299     wait for 4*clk_period;
300
301     -- drop at Bottom Mid
302     drop <= '1';
303     wait for clk_period;
304     drop <= '0';
305     wait for 4*clk_period;
306     -- XO_
307     -- XO_
308     -- _X_
309
310     -- step 6 (player 2)
311     -- move down to Bottom Mid
312     down <= '1';
313     wait for clk_period;
314     down <= '0';
315     wait for 4*clk_period;
316
317     -- move left to Bottom Left
318     left <= '1';
319     wait for clk_period;
320     left <= '0';
321     wait for 4*clk_period;
322
323     -- drop at Bottom Left
324     drop <= '1';
325     wait for clk_period;
326     drop <= '0';
327     wait for 4*clk_period;
328     -- XO_
329     -- XO_
330     -- OX_
331
332     -- step 7 (player 1)
```

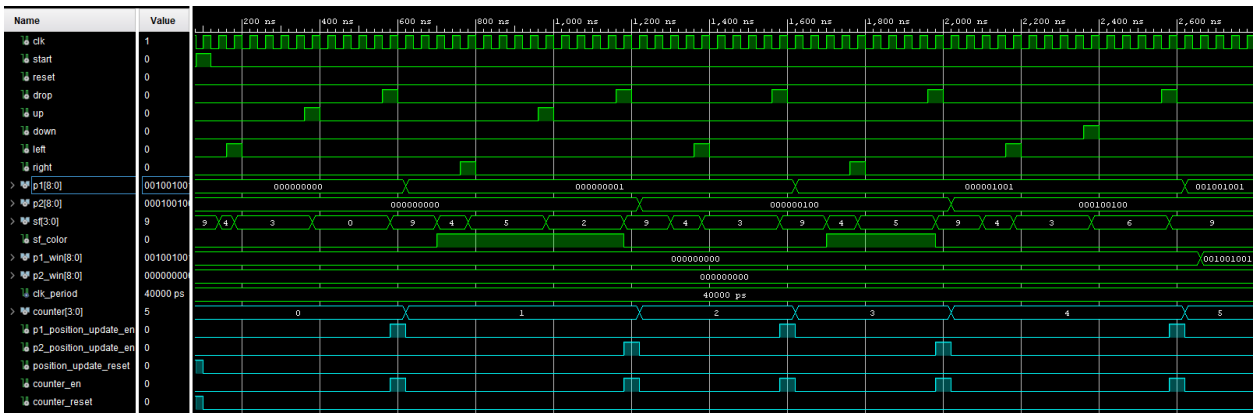
```
    -- move up to Top Mid
    up <= '1';
    wait for clk_period;
    up <= '0';
    wait for 4*clk_period;

    -- move right to Top Right
    right <= '1';
    wait for clk_period;
    right <= '0';
    wait for 4*clk_period;

    -- drop at Top Right
    drop <= '1';
    wait for clk_period;
    drop <= '0';
    wait for 4*clk_period;
    -- XOX
    -- XO_
    -- OX_

    -- step 8 (player 2)
    -- move down to Bottom Mid
    down <= '1';
    wait for clk_period;
    down <= '0';
    wait for 4*clk_period;

    -- move right to Bottom Right
    right <= '1';
    wait for clk_period;
    right <= '0';
    wait for 4*clk_period;

    -- drop at Bottom Right
    drop <= '1';
    wait for clk_period;
    drop <= '0';
    wait for 4*clk_period;
    -- XOX
    -- XO_
    -- OXO

    -- step 9 (player 1)
    -- move right to Central Right
    right <= '1';
    wait for clk_period;
    right <= '0';
    wait for 4*clk_period;
```

```
    -- drop at Central Right
    drop <= '1';
    wait for clk_period;
    drop <= '0';
    wait for 8*clk_period;
    -- XOX
    -- XOX
    -- OXO

    -- Nobody wins. Tie!

    reset <= '1';
    wait for clk_period;
    reset <= '0';
    wait for 8*clk_period;

    --
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  --Test dropping at pre-occupied space:
  --
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    -- press start button
    start <= '1';
    wait for clk_period;
    start <= '0';
    wait for clk_period;

    -- Step 1 (player 1)
    -- drop at Central Mid
    drop <= '1';
    wait for clk_period;
    drop <= '0';
    wait for 4*clk_period;
    -- ___
    -- _X_
    -- ___

    -- Step 2 (player 2)
    -- attempt to drop at Central Mid
    drop <= '1';
    wait for clk_period;
    drop <= '0';
    wait for 4*clk_period;
    -- ___
    -- _X_
    -- ___

    -- Step 2 (player 2)
    -- attempt again to drop at Central Mid
```

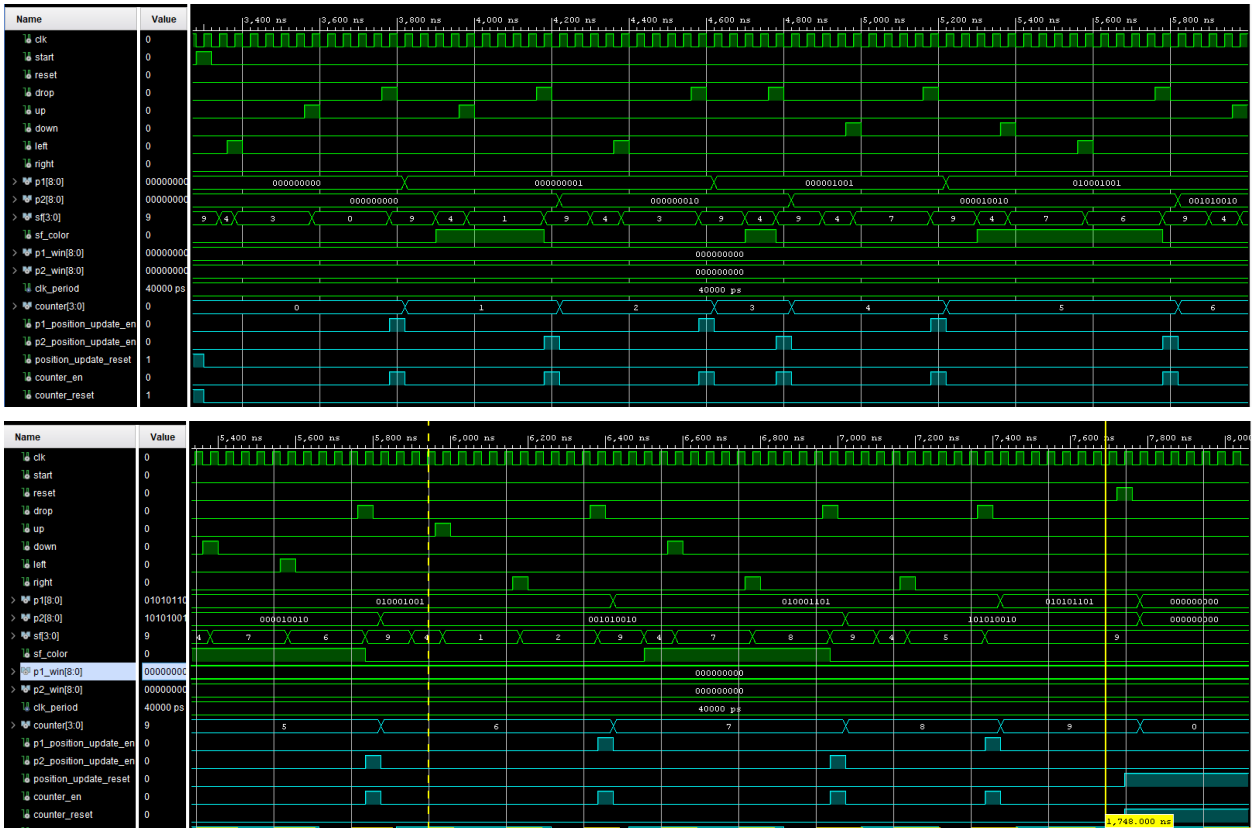
```
    drop <= '1';
    wait for clk_period;
    drop <= '0';
    wait for 4*clk_period;
    -- ___
    -- _X_
    -- ___

    reset <= '1';
    wait for clk_period;
    reset <= '0';
    wait;
end process stim_proc;
end testbench;
```

Appendix I: Test results of game_logic_tb.vhd (Please refer to the comments in **Appendix H** that graphically presents the chessboard)

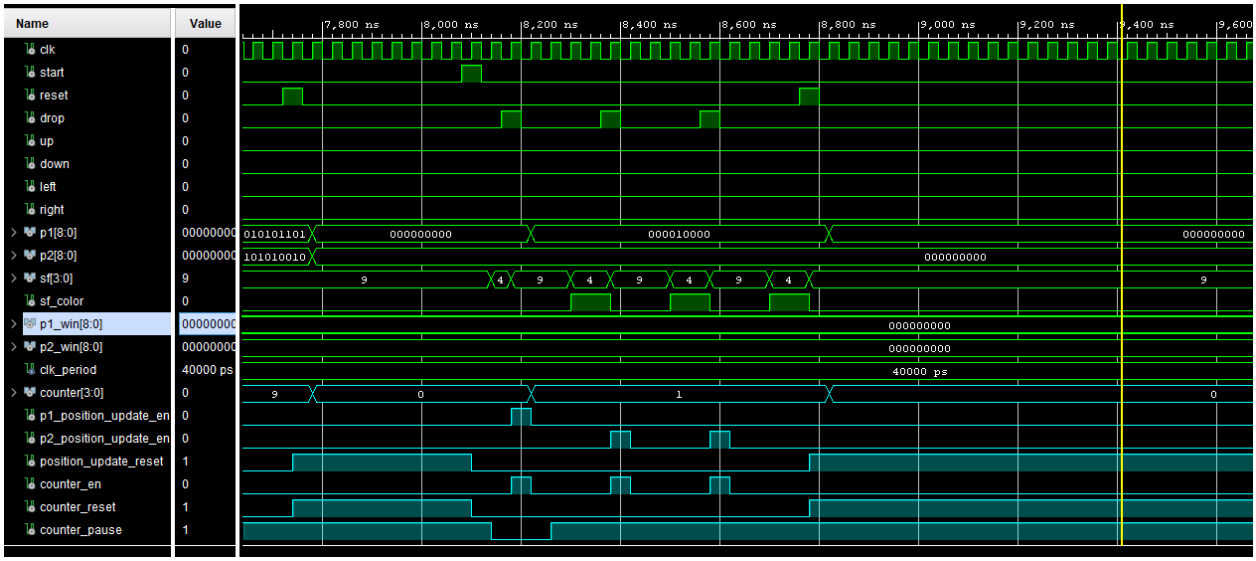
Test a normal game:



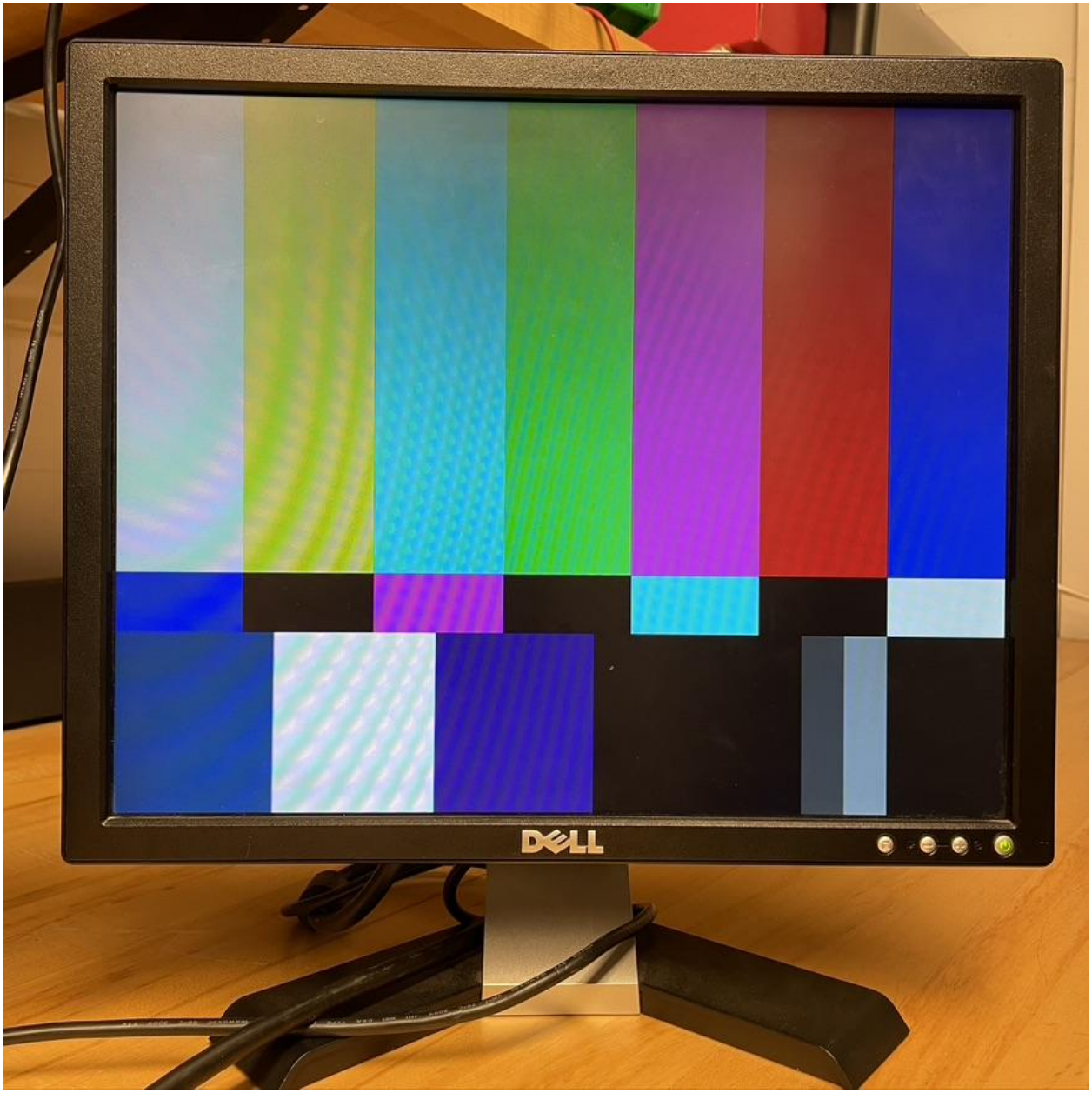
Test a tie game:

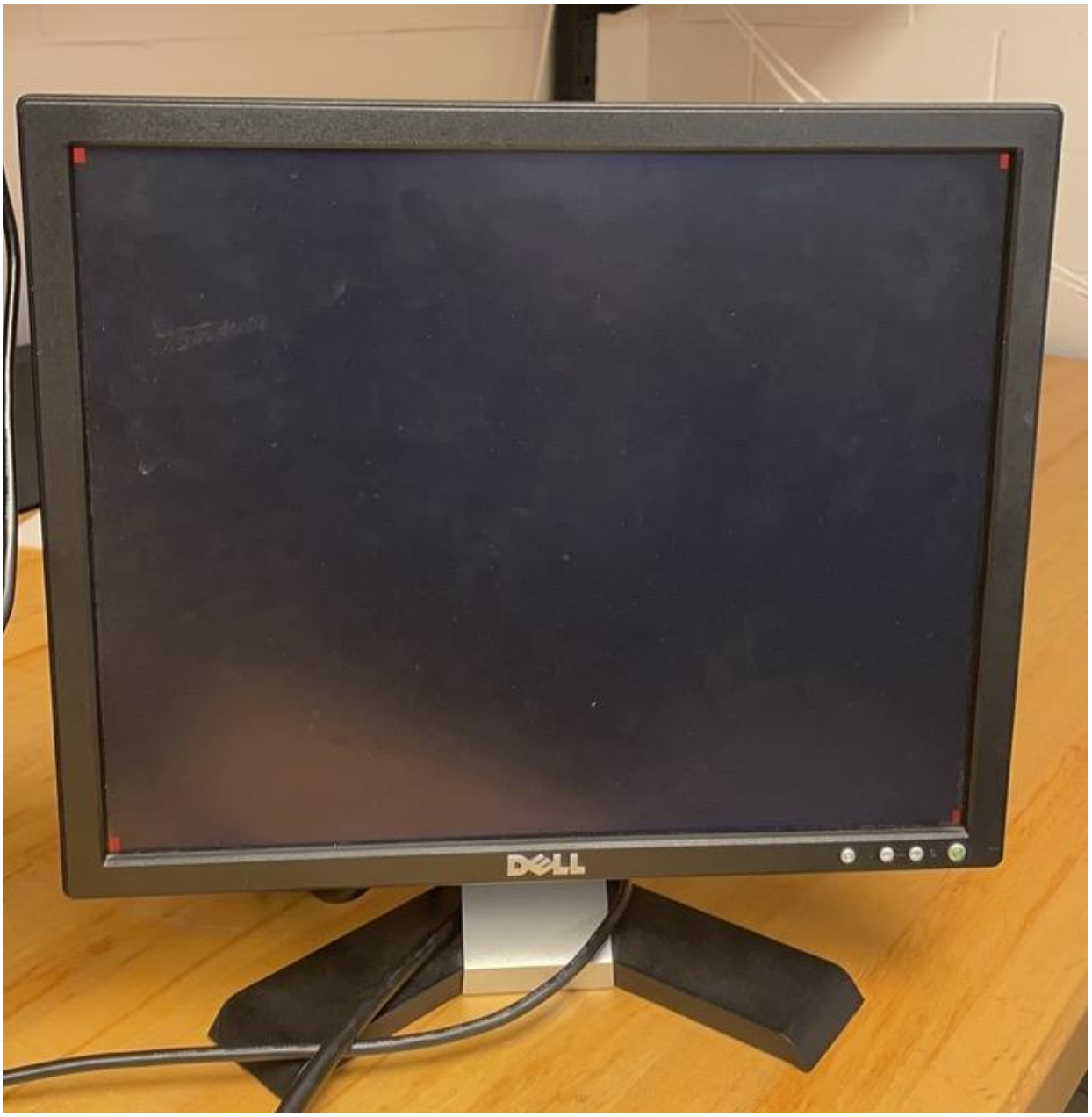


Test dropping at a pre-occupied space: (counter_pause stays high so that the pre-occupied space cannot be dropped again, and the counter doesn't increment.)



Appendix J: VGA driver testing with patterns in vga_test_pattern_12.vhd and the one based on Tad's suggestions




DELL