Active-HDL PDF Export
PONG workspace

Contents

2 PONG

2.1 PONG.vhd

```
--Cristian Alexis Lopez Bernal
--Angel Daniel Zanahua Soto
--Jose Luis Rubio Manjarrez


library ieee;
    use ieee.std_logic_1164.all;

library work;
    use work.all;

entity PONG is

    port (

        CLK : in std_logic;  -- PIN_17

        BTNS : in std_logic_vector( 2 downto 0 );  -- PIN_144, 32, 31


        VGA_HSYNC : out std_logic;  -- PIN_69
        VGA_VSYNC : out std_logic;  -- PIN_71
        VGA_RED_0 : out std_logic;  -- PIN_40
        VGA_RED_1 : out std_logic;  -- PIN_42
        VGA_RED_2 : out std_logic;  -- PIN_44
        VGA_GRN_0 : out std_logic;  -- PIN_47
        VGA_GRN_1 : out std_logic;  -- PIN_51
        VGA_GRN_2 : out std_logic;  -- PIN_53
        VGA_BLU_0 : out std_logic;  -- PIN_57
        VGA_BLU_1 : out std_logic;  -- PIN_59
        VGA_BLU_2 : out std_logic;   -- PIN_63

        LEDS : out std_logic_vector( 2 downto 0 )  -- PIN_9, 7, 3
    );

end entity;

architecture arch of PONG is

    -- VGA frame size
    constant videoWidth : integer := 3;
    constant totalCols  : integer := 800;
    constant totalRows  : integer := 525;
    constant activeCols : integer := 640;
    constant activeRows : integer := 480;

    -- More VGA signals
    signal hsync         : std_logic;
    signal vsync         : std_logic;
    signal redVideoPorch : std_logic_vector( videoWidth - 1 downto 0 );
    signal grnVideoPorch : std_logic_vector( videoWidth - 1 downto 0 );
    signal bluVideoPorch : std_logic_vector( videoWidth - 1 downto 0 );

    -- Pong signals
    signal hsyncPong : std_logic;
    signal vsyncPong : std_logic;
    signal redVideoPong : std_logic_vector( videoWidth - 1 downto 0 );
    signal grnVideoPong : std_logic_vector( videoWidth - 1 downto 0 );
```

```
    signal bluVideoPong : std_logic_vector( videoWidth - 1 downto 0 );

    -- Clock ( 25MHz )
    signal clock_half : std_logic := '0';  -- Generate 25MHz clock from 50MHz

    -- Debounce buttons
    signal btnPrevSteadyState : std_logic_vector( 2 downto 0 ) := ( others =>
 '1' );
    signal btns_filtered : std_logic_vector( 2 downto 0 );

    -- LEDs
    signal r_leds : std_logic_vector( 2 downto 0 ) := ( others => '0' );


begin

    VGA_RED_0 <= redVideoPorch(0);
    VGA_RED_1 <= redVideoPorch(1);
    VGA_RED_2 <= redVideoPorch(2);

    VGA_GRN_0 <= grnVideoPorch(0);
    VGA_GRN_1 <= grnVideoPorch(1);
    VGA_GRN_2 <= grnVideoPorch(2);

    VGA_BLU_0 <= bluVideoPorch(0);
    VGA_BLU_1 <= bluVideoPorch(1);
    VGA_BLU_2 <= bluVideoPorch(2);

    -- BTNs are configured as active low yet PONG entities expect active high
    btns_filtered <= not btnPrevSteadyState;

    -- LEDs are active low
    LEDS <= not r_leds;


    VGA_Sync_Pulses_instance : entity VGA_Sync_Pulses

        generic map (

            totalCols,
            totalRows,
            activeCols,
            activeRows
        )

        port map (

            clock_half,
            --
            hsync,
            vsync,
            open,
            open
        );


    VGA_Sync_Porch_instance : entity VGA_Sync_Porch

        generic map (

            videoWidth,
            totalCols,
            totalRows,
```

```
            activeCols,
            activeRows
        )

        port map (

            clock_half,
            hsyncPong,
            vsyncPong,
            redVideoPong,
            grnVideoPong,
            bluvideoPong,
            --
            VGA_HSYNC,
            VGA_VSYNC,
            redVideoPorch,
            grnVideoPorch,
            bluVideoPorch
        );


    Pong_Top_instance : entity PONG_TOP

        generic map (

            videoWidth,
            totalCols,
            totalRows,
            activeCols,
            activeRows
        )

        port map (

            clock_half,
            hsync,
            vsync,
            BTNS(2),    --btns_filtered(2), -- reset
            BTNS(1),    --btns_filtered(1), -- up
            BTNS(0),    --btns_filtered(0), -- down
            --
            hsyncPong,
            vsyncPong,
            redVideoPong,
            grnVideoPong,
            bluVideoPong,

            r_leds
        );


--  Debounce_instance_2 : DEBOUNCE_SWITCH
--
--      generic map (
--
--          15  -- ?? at 50MHz
--      )
--
--      port map (
--
--          CLK,
--          BTNS(2),
--          --
```

```
--          btnPrevSteadyState(2)
--      );
--
--  Debounce_instance_1 : DEBOUNCE_SWITCH
--
--      generic map (
--
--          15  -- ?? at 50MHz
--      )
--
--      port map (
--
--          CLK,
--          BTNS(1),
--          --
--          btnPrevSteadyState(1)
--      );
--
--  Debounce_instance_0 : DEBOUNCE_SWITCH
--
--      generic map (
--
--          15  -- ?? at 50MHz
--      )
--
--      port map (
--
--          CLK,
--          BTNS(0),
--          --
--          btnPrevSteadyState(0)
--      );


    -- Generate 25MHz clock
    process( CLK )

    begin

        if rising_edge( CLK ) then

            clock_half <= not clock_half;

        end if;

    end process;

end architecture;
```

2.2 VGA_Sync_Pulses.vhd

```
-- http://www.nandland.com
-- https://youtu.be/7wjTJivsNMM

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--use work.all;

entity VGA_Sync_Pulses is
```

```
    generic (

        TOTAL_COLS  : integer;
        TOTAL_ROWS  : integer;
        ACTIVE_COLS : integer;
        ACTIVE_ROWS : integer
    );

    port (

        CLK       : in std_logic;
        o_HSYNC   : out std_logic;
        o_VSYNC   : out std_logic;
        COL_COUNT : out std_logic_vector( 9 downto 0 );
        ROW_COUNT : out std_logic_vector( 9 downto 0 )
    );
end entity;

architecture arch of VGA_Sync_Pulses is

    signal colCount : integer range 0 to TOTAL_COLS - 1 := 0;
    signal rowCount : integer range 0 to TOTAL_ROWS - 1 := 0;

begin

    --
    o_HSYNC <= '1' when colCount < ACTIVE_COLS else '0';
    o_VSYNC <= '1' when rowCount < ACTIVE_ROWS else '0';

    COL_COUNT <= std_logic_vector( to_unsigned( colCount, COL_COUNT'length )
);
    ROW_COUNT <= std_logic_vector( to_unsigned( rowCount, ROW_COUNT'length )
);

    --
    process( CLK )

    begin

        if rising_edge( CLK ) then

            if colCount = TOTAL_COLS - 1 then

                colCount <= 0;

                if rowCount = TOTAL_ROWS - 1 then

                    rowCount <= 0;

                else

                    rowCount <= rowCount + 1;

                end if;

            else

                colCount <= colCount + 1;

            end if;

        end if;
```

```
    end process;

end architecture;
```

2.3 PONG_TOP.vhd

```
-- https://www.nandland.com/goboard/pong-game-in-fpga-with-go-board-vga.html
-- www.jk-quantized.com

library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

library work;
    use work.all;
    use work.PONG_PACKAGE.all;

entity PONG_TOP is

    generic (

        g_VIDEO_WIDTH : integer;
        g_TOTAL_COLS  : integer;
        g_TOTAL_ROWS  : integer;
        g_ACTIVE_COLS : integer;
        g_ACTIVE_ROWS : integer
    );

    port (

        CLK       : in std_logic;
        i_HSYNC   : in std_logic;
        i_VSYNC   : in std_logic;

        RESTART   : in std_logic;

        P1_UP     : in std_logic;
        P1_DOWN   : in std_logic;

        o_HSYNC   : out std_logic;
        o_VSYNC   : out std_logic;
        RED_VIDEO : out std_logic_vector( g_VIDEO_WIDTH - 1 downto 0 );

        BLU_VIDEO : out std_logic_vector( g_VIDEO_WIDTH - 1 downto 0 );

        GRN_VIDEO : out std_logic_vector( g_VIDEO_WIDTH - 1 downto 0 );

        DEBUG : out std_logic_vector( 2 downto 0 )
    );

end entity;

architecture arch of PONG_TOP is

    type states is (

        s_idle,
        s_running,
        s_P1_Scores,
        s_P2_Scores,
        s_P1_Wins,
        s_P2_Wins
```

```
    );
    signal state : states := s_idle;

    signal hsync : std_logic;
    signal vsync : std_logic;

    signal colCount : std_logic_vector( 9 downto 0 );
    signal rowCount : std_logic_vector( 9 downto 0 );

    -- Divided versions or row/col counters
    -- Allows us to make board 40x30
    signal colCountDiv : std_logic_vector( 5 downto 0 ) := ( others => '0' );
    signal rowCountDiv : std_logic_vector( 5 downto 0 ) := ( others => '0' );

    signal drawAny  : std_logic;

    signal drawBall : std_logic;
    signal ballX    : std_logic_vector( 5 downto 0 );
    signal ballY    : std_logic_vector( 5 downto 0 );
    signal ballUpdated : std_logic;

    signal P1_drawPaddle : std_logic;
    signal P1_paddleY    : std_logic_vector( 5 downto 0 );
    signal P1_paddleTop  : unsigned( 5 downto 0 );
    signal P1_paddleBtm  : unsigned( 5 downto 0 );
    signal P1_score      : integer range 0 to c_scoreLimit := 0;

    signal gameStarted : std_logic := '0';

    signal playerWins      : std_logic := '0';
    signal playerLoses     : std_logic := '0';
    signal drawPlayerWins  : std_logic := '0';
    signal drawPlayerLoses : std_logic := '0';
    signal drawEndScreens  : std_logic := '0';


begin

    DEBUG <= std_logic_vector( to_unsigned( P1_score, DEBUG'length ) );

    -- Drop 4 LSBs to divide by 16
    colCountDiv <= colCount( colCount'left downto 4 );
    rowCountDiv <= rowCount( rowCount'left downto 4 );

    -- Create intermediary signals for P1 and P2 paddle positions
    P1_paddleBtm <= unsigned( P1_paddleY );
    P1_paddleTop <= P1_paddleBtm + to_unsigned( c_P1_paddleHeight, P1_paddleB
tm'length );

    -- Assign color outputs
    drawAny <= drawBall or P1_drawPaddle;
    drawEndScreens <= drawPlayerWins or drawPlayerLoses;

    --RED_VIDEO <= ( others => '1' ) when drawAny = '1' else ( others => '0'
);
    --GRN_VIDEO <= ( others => '1' ) when drawAny = '1' else ( others => '0'
);
    --BLU_VIDEO <= ( others => '1' ) when drawAny = '1' else ( others => '0'
);

    RED_VIDEO <= ( others => '1' )
        when
            ( gameStarted = '1' and drawAny = '1' ) or
```

```
            ( gameStarted = '0' and drawEndScreens = '1' )

        else
            ( others => '0' );

    GRN_VIDEO <= ( others => '1' )
        when
            ( gameStarted = '1' and drawAny = '1' ) or
            ( gameStarted = '0' and drawEndScreens = '1' )

        else
            ( others => '0' );

    BLU_VIDEO <= ( others => '1' )
        when
            ( gameStarted = '1' and drawAny = '1' ) or
            ( gameStarted = '0' and drawEndScreens = '1' )

        else
            ( others => '0' );


    Sync_To_Count_instance : entity Sync_To_Count

        generic map (

            g_TOTAL_COLS,
            g_TOTAL_ROWS
        )

        port map (

            CLK,
            i_HSYNC,
            i_VSYNC,
            --
            hsync,
            vsync,
            colCount,
            rowCount
        );


    P1_Paddle_Control_instance : entity PADDLE_CONTROL

        generic map (

            c_P1_paddleColPos,
            c_P1_paddleHeight
        )

        port map (

            CLK,
            colCountDiv,
            rowCountDiv,
            P1_UP,
            P1_DOWN,
            --
            P1_drawPaddle,
            P1_paddleY
        );
```

```
    Pong_Ball_instance : entity BALL_CONTROL

        port map (

            CLK,
            gameStarted,
            colCountDiv,
            rowCountDiv,
            --
            drawBall,
            ballX,
            ballY,
            ballUpdated
        );


    Draw_Win_Screen_instance : entity DRAW_WIN_SCREEN

        port map (

            CLK,
            playerWins,
            colCountDiv,
            rowCountDiv,
            --
            drawPlayerWins
        );

    Draw_Lose_Screen_instance : entity DRAW_LOSE_SCREEN

        port map (

            CLK,
            playerLoses,
            colCountDiv,
            rowCountDiv,
            --
            drawPlayerLoses
        );


    -- Register synchs to align with output data
    process( CLK )

    begin

        if rising_edge( CLK ) then

            o_VSYNC <= vsync;
            o_HSYNC <= hsync;

        end if;

    end process;


    -- Use state machinge to control state of play
    process( CLK )

    begin

        if rising_edge( CLK ) then
```

```
            case state is

                when s_idle =>

                    gameStarted <= '0'; --0 orig

                    if RESTART = '1' then

                        -- Reset
                        P1_score    <= 0;
                        playerWins  <= '0';
                        playerLoses <= '0';

                        -- Start
                        gameStarted <= '1';
                        state <= s_running;

                    end if;

                when s_running =>

                    if ballUpdated = '1' then

                        -- Player 1's side
                        if ballX = std_logic_vector( to_unsigned( 0, ballX'le
ngth ) ) then

                            if ( unsigned( ballY ) < P1_paddleBtm or
                                 unsigned( ballY ) > P1_paddleTop )
                            then

                                state <= s_P2_Scores;

                            end if;


                        -- Player 2's side
                        -- single player so ignore paddle... otherwise copy P
1 code
                        elsif ballX = std_logic_vector( to_unsigned( c_gameWi
dth - 1, ballX'length ) ) then

                            state <= s_P1_Scores;

                        end if;

                    end if;


                when s_P1_Scores =>

                    if P1_score = c_scoreLimit then

                        state <= s_P1_Wins;

                    else

                        P1_score <= P1_score + 1;

                        state <= s_running;

                    end if;
```

```
                when s_P2_Scores =>

                    state <= s_P2_Wins;

                when s_P1_Wins =>

                    playerWins <= '1'; -- draws fancy end screen

                    state <= s_idle;

                when s_P2_Wins =>

                    playerLoses <= '1'; -- draws fancy end screen

                    state <= s_idle;

                -- Shouldn't get here
                when others =>

                    state <= s_idle;

            end case;

        end if;

    end process;

end architecture;
```

2.4 BALL_CONTROL.vhd

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

library work;
    use work.PONG_PACKAGE.all;


entity BALL_CONTROL is

    port (

        CLK           : in std_logic;
        GAME_STARTED  : in std_logic;
        COL_COUNT_DIV : in std_logic_vector( 5 downto 0 );
        ROW_COUNT_DIV : in std_logic_vector( 5 downto 0 );

        DRAW_BALL    : out std_logic;
        BALL_X       : out std_logic_vector( 5 downto 0 );
        BALL_Y       : out std_logic_vector( 5 downto 0 );
        BALL_UPDATED : out std_logic
    );

end entity;

architecture arch of BALL_CONTROL is

    signal colIndex : integer range 0 to 2 ** COL_COUNT_DIV'length := 0;
    signal rowIndex : integer range 0 to 2 ** ROW_COUNT_DIV'length := 0;

    signal ballCount : integer range 0 to c_ballSpeed := 0;
```

```
    signal ballX     : integer range 0 to 2 ** COL_COUNT_DIV'length := 0;
    signal ballY     : integer range 0 to 2 ** ROW_COUNT_DIV'length := 0;
    signal ballXPrev : integer range 0 to 2 ** COL_COUNT_DIV'length := 0;
    signal ballYPrev : integer range 0 to 2 ** ROW_COUNT_DIV'length := 0;

    signal drawBall : std_logic := '0';

begin

    DRAW_BALL <= drawBall;
    BALL_X <= std_logic_vector( to_unsigned( ballX, BALL_X'length ) );
    BALL_Y <= std_logic_vector( to_unsigned( ballY, BALL_Y'length ) );

    colIndex <= to_integer( unsigned( COL_COUNT_DIV ) );
    rowIndex <= to_integer( unsigned( ROW_COUNT_DIV ) );


    -- Move ball
    process( CLK )

    begin

        if rising_edge( CLK ) then

            -- Ball stays in middle of screen until game starts
            if GAME_STARTED = '0' then

                ballX <= c_gameWidth / 2;
                ballY <= c_gameHeight / 2;
                ballXPrev <= c_gameWidth / 2 + 1;
                ballYPrev <= c_gameHeight / 2 - 1;

            else

                -- Update ball counter
                if ballCount = c_ballSpeed then

                    ballCount <= 0;

                    BALL_UPDATED <= '1';

                else

                    ballCount <= ballCount + 1;

                    BALL_UPDATED <= '0';

                end if;


                -- Control x position
                if ballCount = c_ballSpeed then

                    ballXPrev <= ballX;

                    -- If ball is moving to right, keep moving to right,
                    --  but check not at wall in which case bounces back
                    if ballXPrev < ballX then

                        if ballX = c_gameWidth - 1 then

                            ballX <= ballX - 1;
```

```
                        else
                            ballX <= ballX + 1;
                        end if;
                    -- Ball moving to left
                    elsif ballXPrev > ballX then
                        if ballX = 0 then
                            ballX <= ballX + 1;
                        else
                            ballX <= ballX - 1;
                        end if;
                    end if;
                end if;

                -- Control y position
                if ballCount = c_ballSpeed then
                    ballYPrev <= ballY;
                    -- If ball is moving up, keep moving up,
                    --  but check not at wall in which case bounces back
                    if ballYPrev < ballY then
                        if ballY = c_gameHeight - 1 then
                            ballY <= ballY - 1;
                        else
                            ballY <= ballY + 1;
                        end if;
                    -- Ball moving down
                    elsif ballYPrev > ballY then
                        if ballY = 0 then
                            ballY <= ballY + 1;
                        else
                            ballY <= ballY - 1;
                        end if;
                    end if;
                end if;
            end if;
        end if;
```

```
    end process;


    -- Draw ball
    process( CLK )

    begin

        if rising_edge( CLK ) then

            if colIndex = ballX and rowIndex = ballY then

                drawBall <= '1';

            else

                drawBall <= '0';

            end if;

        end if;

    end process;

end architecture;
```

2.5 DRAW_LOSE_SCREEN.vhd

```
-- www.jk-quantized.com

library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

library work;
    use work.PONG_PACKAGE.all;

entity DRAW_LOSE_SCREEN is

    port (

        CLK           : in std_logic;
        PLAYER_LOST   : in std_logic;
        COL_COUNT_DIV : in std_logic_vector( 5 downto 0 );
        ROW_COUNT_DIV : in std_logic_vector( 5 downto 0 );

        DRAW_LOSE     : out std_logic
    );

end entity;

architecture arch of DRAW_LOSE_SCREEN is

    signal colIndex : integer range 0 to 2 ** COL_COUNT_DIV'length := 0;
    signal rowIndex : integer range 0 to 2 ** ROW_COUNT_DIV'length := 0;

    type pixels is array( 0 to c_gameHeight - 1 ) of std_logic_vector( 0 to c
_gameWidth - 1 );
    signal pixel : pixels;
```

```
begin

    -- Screen setup
    pixel(0)  <= "00000000000000000000000000000000000000000000";
    pixel(1)  <= "00000000000000000000000000000000000000000000";
    pixel(2)  <= "00000000000000000000000000000000000000000000";
    pixel(3)  <= "00000000000000000000000000000000000000000000";
    pixel(4)  <= "00000000000000000000000000000000000000000000";
    pixel(5)  <= "00000000000000000000000000000000000000000000";
    pixel(6)  <= "00000000000000000000000000000000000000000000";
    pixel(7)  <= "00000011111001111100100010001111110000000000";
    pixel(8)  <= "00000010000001000100110110001000000000000000";
    pixel(9)  <= "00000010000001000100101010001000000000000000";
    pixel(10) <= "00000010000001000100100010001111100000000000";
    pixel(11) <= "00000011111001111100100010001000000000000000";
    pixel(12) <= "00000010001001000100100010001000000000000000";
    pixel(13) <= "00000010001001000100100010001000000000000000";
    pixel(14) <= "00000011111001000100100010001111110000000000";
    pixel(15) <= "00000000000000000000000000000000000000000000";
    pixel(16) <= "00000000000000000000000000000000000000000000";
    pixel(17) <= "00000000000000000000000000000000000000000000";
    pixel(18) <= "00000011111001000100111111000111110000000000";
    pixel(19) <= "00000010001001000100100000010001000000000000";
    pixel(20) <= "00000010001001000100100000010001000000000000";
    pixel(21) <= "00000010001001000100111111000111110000000000";
    pixel(22) <= "00000010001000101000100000011000000000000000";
    pixel(23) <= "00000010001000101000100000010100000000000000";
    pixel(24) <= "00000010001000010000100000001001000000000000";
    pixel(25) <= "00000011111000010000111111000100010000000000";
    pixel(26) <= "00000000000000000000000000000000000000000000";
    pixel(27) <= "00000000000000000000000000000000000000000000";
    pixel(28) <= "00000000000000000000000000000000000000000000";
    pixel(29) <= "00000000000000000000000000000000000000000000";


    colIndex <= to_integer( unsigned( COL_COUNT_DIV ) );
    rowIndex <= to_integer( unsigned( ROW_COUNT_DIV ) );

    -- Draw screen
    process( CLK )

    begin

        if rising_edge( CLK ) then

            if PLAYER_LOST = '1' and pixel( rowIndex )( colIndex ) = '1' then

                DRAW_LOSE <= '1';

            else

                DRAW_LOSE <= '0';

            end if;

        end if;

    end process;

end architecture;
```

2.6 DRAW_WIN_SCREEN.vhd

```
-- www.jk-quantized.com

library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

library work;
    use work.PONG_PACKAGE.all;

entity DRAW_WIN_SCREEN is

    port (

        CLK           : in std_logic;
        PLAYER_WON    : in std_logic;
        COL_COUNT_DIV : in std_logic_vector( 5 downto 0 );
        ROW_COUNT_DIV : in std_logic_vector( 5 downto 0 );

        DRAW_WIN      : out std_logic
    );

end entity;

architecture arch of DRAW_WIN_SCREEN is

    signal colIndex : integer range 0 to 2 ** COL_COUNT_DIV'length := 0;
    signal rowIndex : integer range 0 to 2 ** ROW_COUNT_DIV'length := 0;

    type pixels is array( 0 to c_gameHeight - 1 ) of std_logic_vector( 0 to c
_gameWidth - 1 );
    signal pixel : pixels;


begin

    -- Screen setup
    pixel(0)  <= "00000000000000000000000000000000000000000";
    pixel(1)  <= "00000000000000000000000000000000000000000";
    pixel(2)  <= "00000000000000000000000000000000000000000";
    pixel(3)  <= "00000000000000000000000000000000000000000";
    pixel(4)  <= "00000000000000000000000000000000000000000";
    pixel(5)  <= "00000000000000000000000000000000000000000";
    pixel(6)  <= "00000000000000000000000000000000000000000";
    pixel(7)  <= "00000001000111110000111110001111100000000";
    pixel(8)  <= "00000001000001000000100000010000000000000";
    pixel(9)  <= "00000001000001000000100000010000000000000";
    pixel(10) <= "00000001000001000000111110001111100000000";
    pixel(11) <= "00000001000001000000000010001000000000000";
    pixel(12) <= "00000001000001000000000010001000000000000";
    pixel(13) <= "00000001000001000000000010001000000000000";
    pixel(14) <= "00000001000001000000111110001111100000000";
    pixel(15) <= "00000000000000000000000000000000000000000";
    pixel(16) <= "00000000000000000000000000000000000000000";
    pixel(17) <= "00000000000000000000000000000000000000000";
    pixel(18) <= "00000000010001000000000100000000000000000";
    pixel(19) <= "00000000010001000000000100000000000000000";
    pixel(20) <= "00000000010001000000000100000000000000000";
    pixel(21) <= "00000000011111001111100100000000000000000";
    pixel(22) <= "00000000000001000000000100000000000000000";
    pixel(23) <= "00000000000001000000000100000000000000000";
    pixel(24) <= "00000000000001000000000100000000000000000";
```

```
    pixel(25) <= "00000000000001000000001000000000000000";
    pixel(26) <= "00000000000000000000000000000000000000";
    pixel(27) <= "00000000000000000000000000000000000000";
    pixel(28) <= "00000000000000000000000000000000000000";
    pixel(29) <= "00000000000000000000000000000000000000";


    colIndex <= to_integer( unsigned( COL_COUNT_DIV ) );
    rowIndex <= to_integer( unsigned( ROW_COUNT_DIV ) );

    -- Draw screen
    process( CLK )

    begin

        if rising_edge( CLK ) then

            if PLAYER_WON = '1' and pixel( rowIndex )( colIndex ) = '1' then

                DRAW_WIN <= '1';

            else

                DRAW_WIN <= '0';

            end if;

        end if;

    end process;

end architecture;
```

2.7 PADDLE_CONTROL.vhd

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

library work;
    use work.PONG_PACKAGE.all;


entity PADDLE_CONTROL is

    generic (

        g_PADDLE_X : integer;
        g_PADDLE_HEIGHT : integer
    );

    port (

        CLK : in std_logic;

        COL_COUNT_DIV : in std_logic_vector( 5 downto 0 );
        ROW_COUNT_DIV : in std_logic_vector( 5 downto 0 );

        UP   : in std_logic;
        DOWN : in std_logic;

        DRAW_PADDLE : out std_logic;
```

```
        PADDLE_Y : out std_logic_vector( 5 downto 0 )
    );

end entity;

architecture arch of PADDLE_CONTROL is

    signal colIndex : integer range 0 to 2 ** COL_COUNT_DIV'length := 0;
    signal rowIndex : integer range 0 to 2 ** ROW_COUNT_DIV'length := 0;

    signal paddleCount_enable : std_logic;
    signal paddleCount : integer range 0 to c_paddleSpeed := 0;

    signal paddleY : integer range 0 to c_gameHeight - g_PADDLE_HEIGHT - 1 :=
 0;

    signal drawPaddle : std_logic := '0';

begin

    DRAW_PADDLE <= drawPaddle;
    PADDLE_Y <= std_logic_vector( to_unsigned( paddleY, PADDLE_Y'length ) );

    colIndex <= to_integer( unsigned( COL_COUNT_DIV ) );
    rowIndex <= to_integer( unsigned( ROW_COUNT_DIV ) );

    -- Move paddle only when one button is pressed
    paddleCount_enable <= UP xor DOWN;


    -- Move paddle
    process( CLK )

    begin

        if rising_edge( CLK ) then

            -- Update paddle counter when either switch is pressed and held
            if paddleCount_enable = '1' then

                if paddleCount = c_paddleSpeed then

                    paddleCount <= 0;

                else

                    paddleCount <= paddleCount + 1;

                end if;

            else

                paddleCount <= 0;

            end if;


            -- Update paddle location
            if UP = '1' and paddleCount = c_paddleSpeed then

                -- If already at top, don't update
                if paddleY /= 0 then
```

```
                    paddleY <= paddleY - 1;

                end if;

            elsif DOWN = '1' and paddleCount = c_paddleSpeed then

                -- If already at bottom, don't update
                if paddleY /= c_gameHeight - g_PADDLE_HEIGHT - 1 then

                    paddleY <= paddleY + 1;

                end if;

            end if;

        end if;

    end process;


    -- Draw paddle
    process( CLK )

    begin

        if rising_edge( CLK ) then

            if ( colIndex = g_PADDLE_X and
                 rowIndex >= paddleY and
                 rowIndex <= paddleY + g_PADDLE_HEIGHT )
            then

                drawPaddle <= '1';

            else

                drawPaddle <= '0';

            end if;

        end if;

    end process;

end architecture;
```

2.8 PONG_PACKAGE.vhd

```
-- https://www.nandland.com/goboard/pong-game-in-fpga-with-go-board-vga.html

library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

package PONG_PACKAGE is

    ------------------------------------------------------------------------
    -- Constants
    ------------------------------------------------------------------------

    -- Set width and height of game board
    --  Note also have to change col/rowCountDiv accordingly in PONG_TOP.vhd
```

```
    constant c_gameWidth  : integer := 40;
    constant c_gameHeight : integer := 30;

    -- Set score target
    constant c_scoreLimit : integer := 5;

    -- Set height (in board game units) of paddle
    constant c_P1_paddleHeight : integer := 6;
    constant c_P2_paddleHeight : integer := c_gameWidth;

    -- Set speed of paddle movement.
    --  In this case, paddle moves one board game unit every 50ms
    --   that button is held down (assuming a 25MHz clock)
    constant c_paddleSpeed : integer := 1250000;

    -- Set speed of ball movement.
    --  In this case, ball moves one board game unit every 50ms
    --   (assuming a 25MHz clock)
    constant c_ballSpeed : integer := 1250000;

    -- Set column index to draw player paddles
    constant c_P1_paddleColPos : integer := 0;
    constant c_P2_paddleColPos : integer := c_gameWidth - 1;


end package;
```

2.9 Sync_To_Count.vhd

```
-- http://www.nandland.com
-- https://youtu.be/7wjTJivsNMM

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--use work.all;

entity Sync_To_Count is

    generic (

        TOTAL_COLS : integer;
        TOTAL_ROWS : integer
    );

    port (

        CLK     : in std_logic;
        i_HSYNC : in std_logic;
        i_VSYNC : in std_logic;

        o_HSYNC   : out std_logic;
        o_VSYNC   : out std_logic;
        COL_COUNT : out std_logic_vector( 9 downto 0 );
        ROW_COUNT : out std_logic_vector( 9 downto 0 )
    );
end entity;

architecture arch of Sync_To_Count is

    signal vsync : std_logic := '0';
    signal hsync : std_logic := '0';
```

```
    signal framestart : std_logic;

    signal colCount : unsigned( 9 downto 0 ) := ( others => '0' );
    signal rowCount : unsigned( 9 downto 0 ) := ( others => '0' );

begin

    -- Looking for rising edge on vertical sync to reset counters
    framestart <= '1' when vsync = '0' and i_VSYNC = '1' else '0';

    o_VSYNC <= vsync;
    o_HSYNC <= hsync;

    ROW_COUNT <= std_logic_vector( rowCount );
    COL_COUNT <= std_logic_vector( colCount );

    -- Register syncs to align with output data
    process( CLK )

    begin

        if rising_edge( CLK ) then

            vsync <= i_VSYNC;
            hsync <= i_HSYNC;

        end if;

    end process;

    -- Keep track of row/column counters
    process( CLK )

    begin

        if rising_edge( CLK ) then

            if framestart = '1' then

                colCount <= ( others => '0' );
                rowCount <= ( others => '0' );

            else

                if colCount = to_unsigned( TOTAL_COLS - 1, colCount'length )
then

                    colCount <= ( others => '0' );

                    if rowCount = to_unsigned( TOTAL_ROWS - 1, rowCount'lengt
h ) then

                        rowCount <= ( others => '0' );

                    else

                        rowCount <= rowCount + 1;

                    end if;

                else

                    colCount <= colCount + 1;
```

```
                end if;

            end if;

        end if;

    end process;

end architecture;
```

2.10 VGA_Sync_Porch.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--use work.all;

entity VGA_Sync_Porch is

    generic (

        VIDEO_WIDTH : integer;
        TOTAL_COLS  : integer;
        TOTAL_ROWS  : integer;
        ACTIVE_COLS : integer;
        ACTIVE_ROWS : integer
    );

    port (

        CLK         : in std_logic;
        i_HSYNC     : in std_logic;
        i_VSYNC     : in std_logic;
        i_RED_VIDEO : in std_logic_vector( VIDEO_WIDTH - 1 downto 0 );
        i_GRN_VIDEO : in std_logic_vector( VIDEO_WIDTH - 1 downto 0 );
        i_BLU_VIDEO : in std_logic_vector( VIDEO_WIDTH - 1 downto 0 );

        o_HSYNC     : out std_logic;
        o_VSYNC     : out std_logic;
        o_RED_VIDEO : out std_logic_vector( VIDEO_WIDTH - 1 downto 0 );
        o_GRN_VIDEO : out std_logic_vector( VIDEO_WIDTH - 1 downto 0 );
        o_BLU_VIDEO : out std_logic_vector( VIDEO_WIDTH - 1 downto 0 )
    );

end entity;


architecture arch of VGA_Sync_Porch is

    constant frontPorchHorz : integer := 18;
    constant backPorchHorz  : integer := 50;
    constant frontPorchVert : integer := 10;
    constant backPorchVert  : integer := 33;

    signal w_hsync : std_logic;
    signal w_vsync : std_logic;
    signal r_hsync : std_logic := '0';
    signal r_vsync : std_logic := '0';

    signal colCount : std_logic_vector( 9 downto 0 );
    signal rowCount : std_logic_vector( 9 downto 0 );
```

```
    signal redVideo : std_logic_vector( VIDEO_WIDTH - 1 downto 0 ) := ( other
s => '0' );
    signal grnVideo : std_logic_vector( VIDEO_WIDTH - 1 downto 0 ) := ( other
s => '0' );
    signal bluVideo : std_logic_vector( VIDEO_WIDTH - 1 downto 0 ) := ( other
s => '0' );

    component Sync_To_Count is

        generic (

            TOTAL_COLS : integer;
            TOTAL_ROWS : integer
        );

        port(

            CLK     : in std_logic;
            i_HSYNC : in std_logic;
            i_VSYNC : in std_logic;

            o_HSYNC   : out std_logic;
            o_VSYNC   : out std_logic;
            COL_COUNT : out std_logic_vector( 9 downto 0 );
            ROW_COUNT : out std_logic_vector( 9 downto 0 )
        );

    end component;

begin

    o_HSYNC <= r_hsync;
    o_VSYNC <= r_vsync;

    sync_inst : Sync_To_Count

        generic map (

            TOTAL_COLS,
            TOTAL_ROWS
        )

        port map (

            CLK,
            i_HSYNC,
            i_VSYNC,

            w_hsync,
            w_vsync,
            colCount,
            rowCount
        );

    -- Modify HSync and VSync signals to include front/back porch
    process( CLK )

    begin

        if rising_edge( CLK ) then

            if ( to_integer( unsigned( colCount ) ) < frontPorchHorz + ACTIVE
_COLS or
```

```
                to_integer( unsigned( colCount ) ) > TOTAL_COLS - backPorchH
orz - 1 ) then
                r_hsync <= '1';
            else
                r_hsync <= w_hsync;
            end if;

            if ( to_integer( unsigned( rowCount ) ) < frontPorchVert + ACTIVE
_ROWS or
                to_integer( unsigned( rowCount ) ) > TOTAL_ROWS - backPorchV
ert - 1 ) then
                r_vsync <= '1';
            else
                r_vsync <= w_vsync;
            end if;
        end if;
    end process;

    -- Align input video to modifier synch pulses (2 clock cycles of delay)
    process( CLK )
    begin
        if rising_edge( CLK ) then
            redVideo <= i_RED_VIDEO;
            grnVideo <= i_GRN_VIDEO;
            bluVideo <= i_BLU_VIDEO;

            o_RED_VIDEO <= redVideo;
            o_GRN_VIDEO <= grnVideo;
            o_BLU_VIDEO <= bluVideo;
        end if;
    end process;
end architecture;
```