Lab 7

Carter Owens, Kyle Turley

Procedures

Introduction. The goal of this lab was to design and implement a VGA controller in VHDL capable of displaying solid colors and a sequence of national flags on a VGA monitor. The project required creating a VGA timing module, a flag display controller, and integrating user input for flag selection.

The procedures for this lab were as follows:

1. Create a Quartus project for the VGA flag viewer
2. Design and implement a VGA timing module to generate correct sync signals and pixel coordinates
3. Implement a flag display controller to output the correct RGB values for each flag
4. Integrate button debouncing for reliable user input
5. Connect the modules and test the system on the FPGA board

Key Requirements: - Generate correct VGA sync signals for 640x480@60Hz - Output 12-bit RGB color values to the VGA DAC - Display a sequence of national flags, selectable by button press - Debounce user input buttons for flag navigation and reset

Issues, Errors, and Stumbles. We were able to create the VGA module with little issue and display plain colors. However, we were unable to get it to display the flags properly for a significant portion of the lab. Eventually, we discovered that we had accidentally set the VGA module as the top-level module in our Quartus project, rather than the intended flag viewer module. Despite this, the project was still compiling, which made debugging more difficult. Once we corrected the top-level module, the flag display logic worked as expected.

Results

The final implementation met the following requirements:

1. **VGA Timing Generation:**
   - The VGA module generated correct horizontal and vertical sync signals for 640x480@60Hz.
   - Pixel coordinates were correctly output for use by the flag display logic.
2. **Flag Display Controller:**
   - The flag viewer module output the correct RGB values for each flag, with color bands and layouts matching the target designs.
   - Button presses cycled through a sequence of national flags, with a reset button to return to the first flag.
3. **User Interface:**
   - Button debouncing was implemented to ensure reliable flag changes and reset behavior.
   - The system responded smoothly to user input.
4. **System Integration:**
   - The VGA and flag viewer modules were successfully integrated and tested on the FPGA board.
   - The display was stable, and the flag sequence cycled as intended after correcting the top-level module.

Figures and Code

See appendix for source code

Conclusion

This lab provided valuable experience in VGA signal generation, color encoding, and modular VHDL design. The main challenge was a project configuration error that set the wrong top-level module, which delayed progress on the flag display logic. Once resolved, the system functioned as intended, and we gained a deeper understanding of both VGA timing and the importance of correct project setup in Quartus.

Key learnings include: - The importance of careful project configuration and top-level module selection - VGA timing and color encoding for 640x480@60Hz displays - Modular VHDL design and integration - Practical debugging strategies for FPGA projects

Appendix

Flag Viewer Module (`flagviewer.vhd`)

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

entity flagviewer is
    port (
        -- [INPUTS ] --
        clk         : in std_logic;                                 -- 50MHz clock input
        key     : in std_logic_vector(1 downto 0);      -- button inputs

        -- [OUTPUTS] --
        vga_r   : out std_logic_vector(3 downto 0);     -- r pixel value
        vga_b       : out std_logic_vector(3 downto 0);     -- b pixel value
        vga_g       : out std_logic_vector(3 downto 0);     -- g pixel value
        vga_hs  : out std_logic;                                -- HS controller
        vga_vs  : out std_logic                             -- VS controller
    );
end entity flagviewer;

architecture behavioral of flagviewer is
    -- [COMPONENTS] --
    component debounce
        port(
            clk     : in std_logic; -- The clock that drives the state machine
            d_in    : in std_logic; -- The value to debounce
            d_out : out std_logic   -- The debounced value
        );
    end component debounce;

    component vga
        port (
            clk         : in std_logic;                             -- clock input
            pixel       : in std_logic_vector(11 downto 0);         -- the async pixel value
            vga_r   : out std_logic_vector(3 downto 0);             -- r pixel value
            vga_b       : out std_logic_vector(3 downto 0);         -- b pixel value
            vga_g       : out std_logic_vector(3 downto 0);         -- g pixel value
            vga_hs  : out std_logic;                                -- HS controller
            vga_vs  : out std_logic;                                -- VS controller
            x_coord     : out integer range 0 to 639;               -- current x coordinate (0-639)
            y_coord : out integer range 0 to 479                -- current y coordinate (0-479)
        );
    end component vga;

    -- [SIGNALS] --
    -- Signals that retain the current coordinate of the current pixel
    signal x    : integer range 0 to 639;
    signal y    : integer range 0 to 479;
    -- Signal that retains information of the current pixel color (RGB)
    signal pixel : std_logic_vector(11 downto 0) := (others => '0');

    -- Signals that control state machine behavior that shows the correct, current flag
    signal crnt_flag    : std_logic_vector(3 downto 0)  := "0000";
    signal next_flag    : std_logic_vector(3 downto 0)  := "0000";

    -- Signals that represent the debounced button values
    signal next_db : std_logic;
    signal rst_db  : std_logic;

    -- [CONSTANTS] --
    constant FRANCE     : std_logic_vector(3 downto 0)  := "0000";       -- the first, default flag
    constant ITALY  : std_logic_vector(3 downto 0)  := "0001";
    constant IRELAND    : std_logic_vector(3 downto 0)  := "0010";
    constant BELGIUM    : std_logic_vector(3 downto 0)  := "0011";
    constant MALI       : std_logic_vector(3 downto 0)  := "0100";
    constant CHAD       : std_logic_vector(3 downto 0)  := "0101";
    constant NIGERIA    : std_logic_vector(3 downto 0)  := "0110";
    constant IVORY  : std_logic_vector(3 downto 0)  := "0111";
    constant POLAND     : std_logic_vector(3 downto 0)  := "1000";
    constant GERMANY    : std_logic_vector(3 downto 0)  := "1001";
    constant AUSTRIA    : std_logic_vector(3 downto 0)  := "1010";
    constant CONGO  : std_logic_vector(3 downto 0)  := "1011";

begin
    -- [INTSTANCES] --
    next_db_impl : debounce port map (clk => clk, d_in => not key(0), d_out => next_db);
    rst_db_impl  : debounce port map (clk => clk, d_in => not key(1), d_out => rst_db);

    vga_impl : vga port map(
        clk => clk,
        pixel => pixel,
        vga_r => vga_r,
        vga_g => vga_g,
        vga_b => vga_b,
        vga_hs => vga_hs,
        vga_vs => vga_vs,
        x_coord => x,
        y_coord => y
    );

    -- [PROCESSES] --
    -- Next Flag Machine
    process (clk) begin
        if rising_edge(clk) then
            crnt_flag <= next_flag;

            case (next_flag) is
                when FRANCE =>
                    if x < 213 then pixel <= to_stdlogicvector(x"00f");
                    elsif x >= 213 and x < 426 then pixel <= to_stdlogicvector(x"fff");
                    else pixel <= to_stdlogicvector(x"f00"); end if;

                when ITALY =>
                    if x < 213 then pixel <= to_stdlogicvector(x"0f0");
                    elsif x >= 213 and x < 426 then pixel <= to_stdlogicvector(x"fff");
                    else pixel <= to_stdlogicvector(x"f00"); end if;

                when IRELAND =>
                    if x < 213 then pixel <= to_stdlogicvector(x"0f0");
                    elsif x >= 213 and x < 426 then pixel <= to_stdlogicvector(x"fff");
                    else pixel <= to_stdlogicvector(x"fa0"); end if;

                when BELGIUM =>
                    if x < 213 then pixel <= to_stdlogicvector(x"000");
                    elsif x >= 213 and x < 426 then pixel <= to_stdlogicvector(x"ff0");
                    else pixel <= to_stdlogicvector(x"f00"); end if;

                when MALI =>
                    if x < 213 then pixel <= to_stdlogicvector(x"0f0");
                    elsif x >= 213 and x < 426 then pixel <= to_stdlogicvector(x"ff0");
                    else pixel <= to_stdlogicvector(x"f00"); end if;

                when CHAD =>
                    if x < 213 then pixel <= to_stdlogicvector(x"005");
                    elsif x >= 213 and x < 426 then pixel <= to_stdlogicvector(x"ff0");
                    else pixel <= to_stdlogicvector(x"f00"); end if;

                when NIGERIA =>
                    if x < 213 then pixel <= to_stdlogicvector(x"0f0");
                    elsif x >= 213 and x < 426 then pixel <= to_stdlogicvector(x"fff");
                    else pixel <= to_stdlogicvector(x"0f0"); end if;

                when IVORY =>
                    if x < 213 then pixel <= to_stdlogicvector(x"fa0");
                    elsif x >= 213 and x < 426 then pixel <= to_stdlogicvector(x"fff");
                    else pixel <= to_stdlogicvector(x"0f0"); end if;

                when POLAND =>
                    if y < 240 then pixel <= to_stdlogicvector(x"fff");
                    else pixel <= to_stdlogicvector(x"f00"); end if;

                when GERMANY =>
                    if y < 160 then pixel <= to_stdlogicvector(x"000");
                    elsif y >= 160 and y < 320 then pixel <= to_stdlogicvector(x"f00");
                    else pixel <= to_stdlogicvector(x"ff0"); end if;

                when AUSTRIA =>
                    if y < 160 then pixel <= to_stdlogicvector(x"f00");
                    elsif y >= 160 and y < 320 then pixel <= to_stdlogicvector(x"fff");
                    else pixel <= to_stdlogicvector(x"f00"); end if;

                when CONGO =>
                    if y < 480 - x then pixel <= to_stdlogicvector(x"0f0");
                    elsif y > 640 - x then pixel <= to_stdlogicvector(x"f00");
                    else pixel <= to_stdlogicvector(x"ff0"); end if;

                when others => pixel <= to_stdlogicvector(x"aa0");
            end case;
        end if;
        end process;

        -- Current Flag Machine
        process (crnt_flag, next_db, rst_db) begin
            if next_db = '1' then
                if crnt_flag = CONGO then next_flag <= FRANCE;
                else next_flag <= std_logic_vector(unsigned(crnt_flag) + to_unsigned(1, 4)); end if;
            elsif rst_db = '1' then next_flag <= FRANCE;
            else next_flag <= crnt_flag; end if;
        end process;
end architecture behavioral;
```

VGA Module (vga.vhd)

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

entity vga is
    port (
        -- [INPUTS] --
        clk         : in std_logic;                                 -- clock input
        pixel       : in std_logic_vector(11 downto 0);             -- the async pixel value

        -- [OUTPUTS]--
        vga_r   : out std_logic_vector(3 downto 0);                 -- r pixel value
        vga_b   : out std_logic_vector(3 downto 0);                 -- b pixel value
        vga_g   : out std_logic_vector(3 downto 0);                 -- g pixel value
        vga_hs  : out std_logic;                                    -- HS controller
        vga_vs  : out std_logic;                                    -- VS controller
        x_coord : out integer range 0 to 639;                   -- current x coordinate (0-639)
        y_coord : out integer range 0 to 479                    -- current y coordinate (0-479)
    );
end entity vga;

architecture behavioral of vga is
    -- [SIGNALS] --
    signal h_count : integer range 0 to 800 := 0;  -- Horizontal counter
    signal v_count : integer range 0 to 525 := 0;  -- Vertical counter
    signal pixel_clk : std_logic := '0';          -- 25MHz pixel clock
    signal pixel_enable : std_logic := '0';       -- Active display area enable

    -- [CONSTANTS] --
    -- Horizontal timing (pixels)
    constant H_DISPLAY   : integer := 640;   -- Display width
    constant H_FRONT     : integer := 16;    -- Front porch
    constant H_SYNC      : integer := 96;    -- Sync pulse
    constant H_BACK      : integer := 48;    -- Back porch
    constant H_TOTAL     : integer := 800;   -- Total horizontal pixels

    -- Vertical timing (lines)
    constant V_DISPLAY    : integer := 480;  -- Display height
    constant V_FRONT     : integer := 10;    -- Front porch
    constant V_SYNC      : integer := 2;     -- Sync pulse
    constant V_BACK      : integer := 33;    -- Back porch
    constant V_TOTAL     : integer := 525;   -- Total vertical lines
```

```vhdl
begin
    -- [DIRECT CONNECTIONS] --
    -- Map RGB values when in active display area
    vga_r <= pixel(11 downto 8) when pixel_enable = '1' else "0000";
    vga_g <= pixel(7 downto 4)  when pixel_enable = '1' else "0000";
    vga_b <= pixel(3 downto 0)  when pixel_enable = '1' else "0000";

    -- Output current coordinates when in active display area
    x_coord <= h_count when pixel_enable = '1' else 0;
    y_coord <= v_count when pixel_enable = '1' else 0;

    -- [PROCESSES] --
    -- Generate 25MHz pixel clock from 50MHz input clock
    process(clk)
    begin
        if rising_edge(clk) then
            pixel_clk <= not pixel_clk;
        end if;
    end process;

    -- Horizontal and vertical counters
    process(clk)
    begin
        if rising_edge(clk) and pixel_clk = '1' then
            -- Horizontal counter
            if h_count = H_TOTAL - 1 then
                h_count <= 0;
                -- Vertical counter
                if v_count = V_TOTAL - 1 then
                    v_count <= 0;
                else
                    v_count <= v_count + 1;
                end if;
            else
                h_count <= h_count + 1;
            end if;
        end if;
    end process;

    -- Generate sync signals and pixel enable
    process(clk)
    begin
        if rising_edge(clk) and pixel_clk = '1' then
            -- Horizontal sync
            if h_count < (H_DISPLAY + H_FRONT) or
               h_count >= (H_DISPLAY + H_FRONT + H_SYNC) then
                vga_hs <= '1';
            else
                vga_hs <= '0';
            end if;

            -- Vertical sync
            if v_count < (V_DISPLAY + V_FRONT) or
               v_count >= (V_DISPLAY + V_FRONT + V_SYNC) then
                vga_vs <= '1';
            else
```

```
                vga_vs <= '0';
            end if;

            -- Pixel enable (active display area)
            if h_count < H_DISPLAY and v_count < V_DISPLAY then
                pixel_enable <= '1';
            else
                pixel_enable <= '0';
            end if;
        end if;
    end process;
end architecture behavioral;
```

Button Debounce Module (`debounce.vhd`)

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

entity debounce is
    port(
        -- [INPUTS] --
        clk     : in std_logic; -- The clock that drives the state machine
        d_in    : in std_logic; -- The value to debounce

        -- [OUTPUTS] --
        d_out : out std_logic   -- The debounced value
    );
end entity debounce;

architecture behavioral of debounce is
    -- [SIGNALS & CONSTANTS] --
    signal current_state : std_logic_vector(1 downto 0) := "00";
    signal next_state       : std_logic_vector(1 downto 0) := "00";

    constant IDLE           : std_logic_vector(1 downto 0) := "00";
    constant PUSH_DOWN   : std_logic_vector(1 downto 0) := "01";
    constant HOLD           : std_logic_vector(1 downto 0) := "10";
    constant PULL_UP        : std_logic_vector(1 downto 0) := "11";
begin

    -- [PROCESSES] --
    process (clk)
    begin
        if rising_edge(clk) then
            current_state <= next_state;
            case (next_state) is
                when PULL_UP => d_out <= '1';
                when others => d_out <= '0';
            end case;
        end if;

    end process;

    process (current_state, d_in)
    begin
        case (current_state) is
```

```
            when IDLE =>
                -- If the button is zero, then idle.
                -- If the button is one, then start debouncing to 1
                if d_in = '0' then next_state <= IDLE;
                else next_state <= PUSH_DOWN; end if;

            when PUSH_DOWN =>
                -- If the button is zero again, then go back to IDLE
                -- If the button is two again, then HOLD that value
                if d_in = '0' then next_state <= IDLE;
                else next_state <= HOLD; end if;

            when HOLD =>
                -- If the button is zero, begin debouncing to 0
                -- If the button is one, then HOLD that value
                if d_in = '0' then next_state <= PULL_UP;
                else next_state <= HOLD; end if;

            when PULL_UP =>
                -- If the button is zero, then go back to IDLE
                -- If the button is one again, then HOLD that value
                if d_in = '0' then next_state <= IDLE;
                else next_state <= HOLD; end if;

            when others => next_state <= IDLE;
        end case;

    end process;


end architecture behavioral;
```