Lab 7 Report

Matthew Crump A01841001
Nicholas Williams A02057223

Objectives

The objective of this lab is to implement a VGA driver to show various flags and patterns. While building on previous concepts such as: using a PLL and finite state machines.

Procedure

We started out by reading the lab-7 requirements for the VGA lab, and we reviewed the lecture notes on the VGA protocol and the various timing associated with it. We then used the System Builder to generate a top module for this lab. We used the VGA pin and buttons. We determined that we needed to add a PLL with output clocks of 25 MHz, since we did not use 25.174 MHz, but the number of pixels per row and column were the same, then the final frames per second would be below 60.

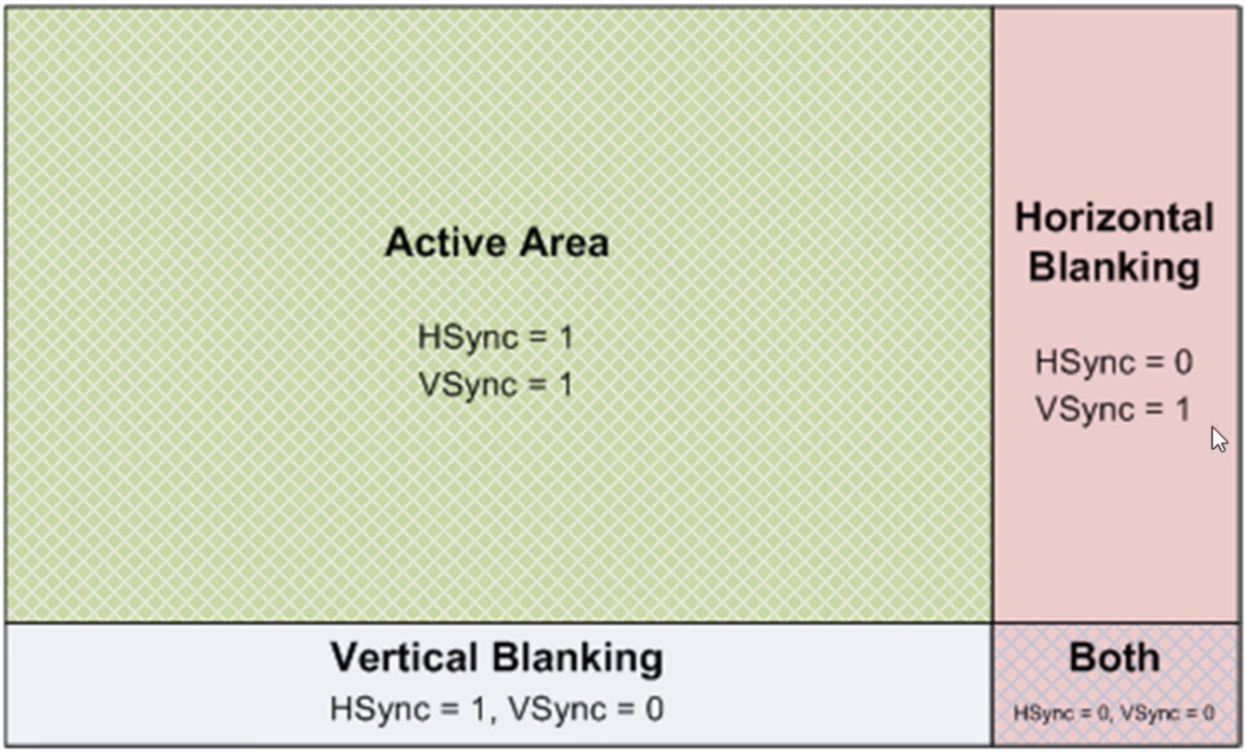

Figure 1. Horizontal and Vertical Sync

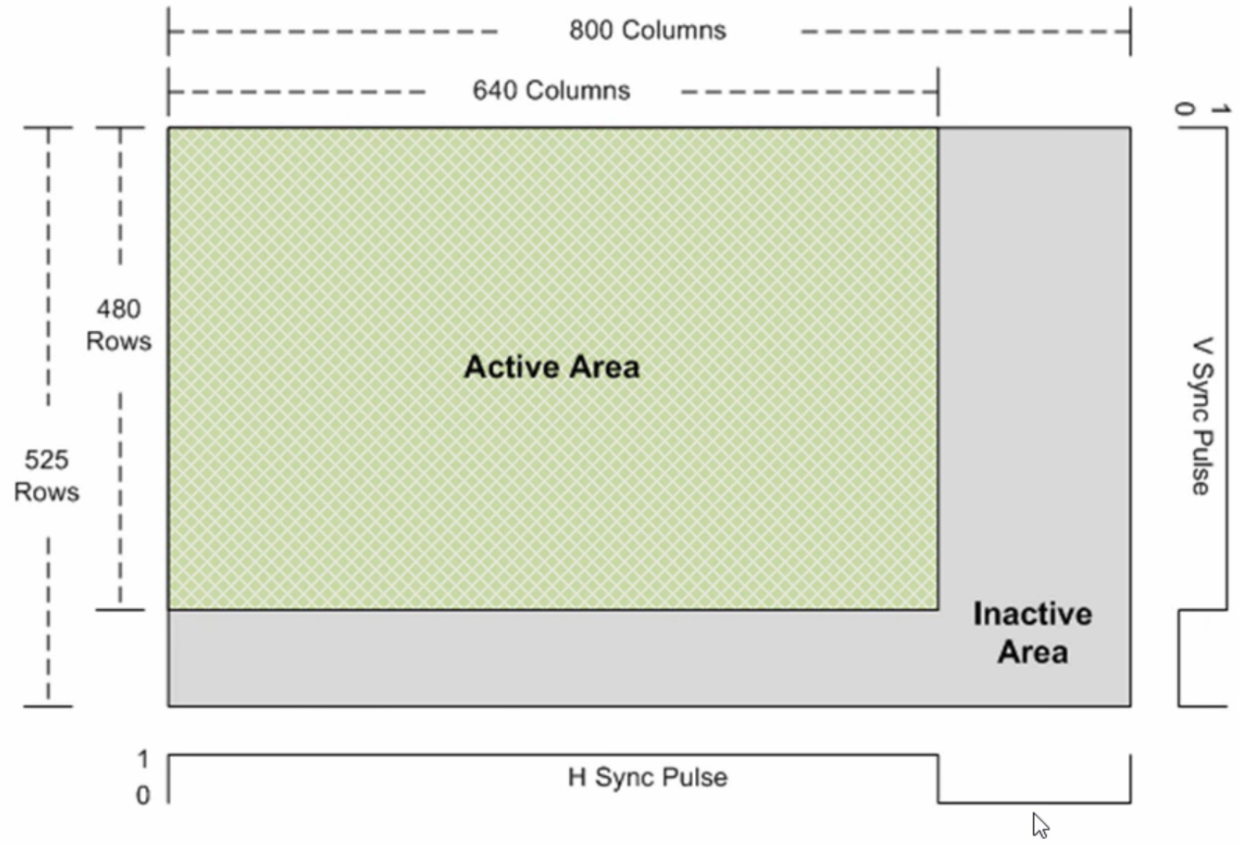

Figure 2. Pixels Per Row and Column

We started out by creating two independent counters, one for the horizontal sync and the other for the vertical sync. This was done so we could focus on just getting the VGA signal recognized by the monitor. After referencing Figure 1 and 2, we got the two sync pulses lined up. Then we worked on displaying colors. To make our state machine easier, a separate process would automatically turn the color to black during the horizontal blanking period so that each state would not have to account for that. We reworked the vertical sync counter to only increment at the end of each horizontal sync. This was done to make the vertical sync counter be in the range of 0 - 480 rather than 0 - 420,000.

To make colors easier, we created 12-bit constants that would then be split out into 4-bit signals for each color channel. This made the state machines much shorter to type rather than assigning three separate values each time. The state machine would display one color while the x or y pixel value was below a certain amount and then switch colors when it was over that value.

Results

We first started by trying to display a red screen to know whether we were synchronized with the VGA monitor properly. Getting the right horizontal and vertical sync was by far the hardest part of the lab, but we eventually got the monitor to display a red screen. We have our vga_sync entity and architecture contained in its own vhd file (see Appendix B). In the future if we need to display something on the VGA monitor we can use the vga_sync file to easily display whatever we need.

Our top model contains three processes inside of the architecture (see Appendix A). The first process controls the transitioning between states based on the “advance” button and the clock. The second process sets the red, green, and blue VGA lines to black for when we are in a horizontal or vertical sync period. The third process is a finite state machine that contains a state for each of the ten flags. Each state controls when each color is set to properly form a flag.

Once we were successfully displaying to our monitor, we were able to easily reproduce the first nine foreign country flags. We simply created a vertical or horizontal counter, based on whether the flag has horizontal or vertical stripes, to determine the transitions between colors on the flag. For the final flag we created a diagonal counter. After each row, the diagonal counter decremented by one so that by the last row of the picture the diagonal counter was at zero.

We designed this lab in sections so that we could always be sure that each piece was working properly. Figure 3 shows the final flow summary for our design to be implemented onto the DE10-lite board. All ten of the flags successfully appear on the monitor in the correct order that they should appear after the advance button is pressed.

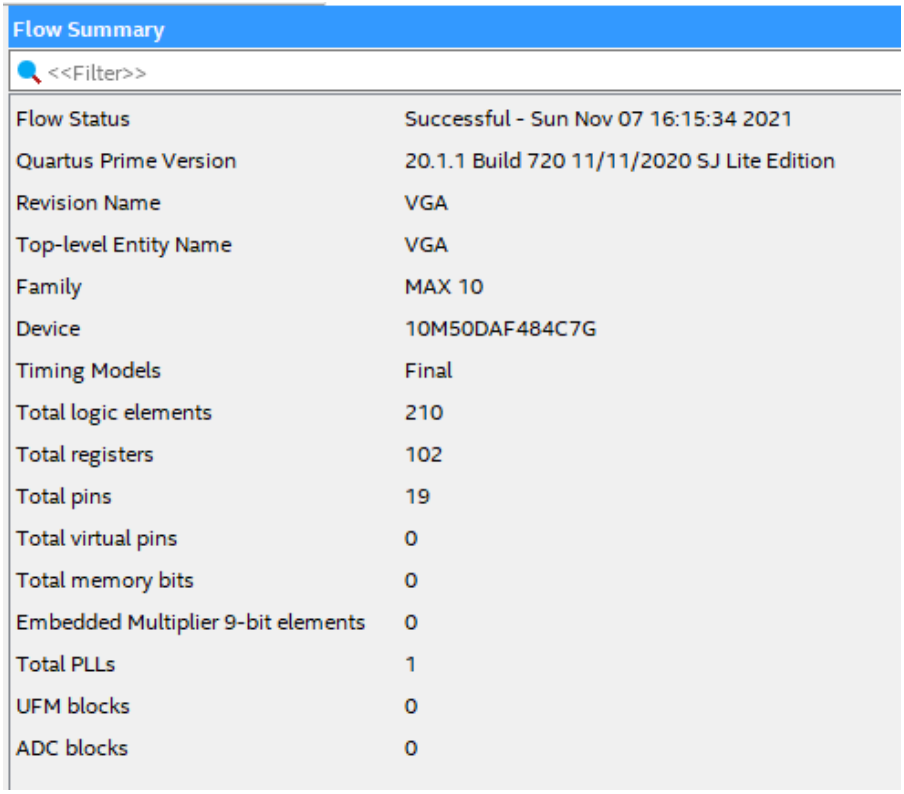

Figure 3. Flow Summary

Conclusion

The purpose of this project was to learn how to use VGA protocol and hardware to interface with a standard computer monitor. We successfully designed, created, and implemented a design that interfaced with a standard computer monitor using VGA protocol and hardware. We created a finite state machine that cycles through ten different foreign country flags to be displayed. We debounced the button presses so that one button press successfully transitions to the next flag. Overall, we achieved the objective of this lab, and developed a design that will most likely be helpful in future labs.

Appendix A

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

entity VGA is
    port (
        ADC_CLK_10 : in std_logic;
        MAX10_CLK1_50 : in std_logic;
        MAX10_CLK2_50 : in std_logic;
        KEY : in std_logic_vector(1 downto 0);
        VGA_B : out std_logic_vector(3 downto 0);
        VGA_G : out std_logic_vector(3 downto 0);
        VGA_R : out std_logic_vector(3 downto 0);
        VGA_HS : out std_logic;
        VGA_VS : out std_logic
    );
end VGA;


architecture behave of VGA is
    constant vga_max_x : natural := 640;
    constant vga_max_y : natural := 480;
    constant flag_v_x1 : natural := 200;
    constant flag_v_x2 : natural := 440;
    constant flag_h_y1 : natural := 160;
    constant flag_h_mid : natural := 240;
    constant flag_h_y2 : natural := 320;

    constant WHITE : std_logic_vector(11 downto 0) := x"FFF";
    constant BLACK : std_logic_vector(11 downto 0) := x"000";
    constant RED : std_logic_vector(11 downto 0) := x"F00";
    constant GREEN : std_logic_vector(11 downto 0) := x"095";
    constant BLUE : std_logic_vector(11 downto 0) := x"00F";
    constant YELLOW : std_logic_vector(11 downto 0) := x"FF0";
    constant ORANGE : std_logic_vector(11 downto 0) := x"F70";
    constant LIME : std_logic_vector(11 downto 0) := x"3F3";
    constant CRIMSON : std_logic_vector(11 downto 0) := x"B11";
    constant BANANA : std_logic_vector(11 downto 0) := x"DC2";
    constant AZULE : std_logic_vector(11 downto 0) := x"109";

    type state_type is (FRANCE, ITALY, IRELAND, BELGIUM, MALI, CHAD, NIGERIA,
IVORY, POLAND, GERMANY, AUSTRIA, CONGO);
    signal flag_state : state_type;

    signal rstn_btn : std_logic;
    signal next_btn : std_logic;
    signal next_delay : std_logic;
    signal next_pulse : std_logic;

    signal vga_clk : std_logic;
    signal vga_x : natural;
    signal vga_y : natural;
    signal pixel_color : std_logic_vector(11 downto 0);
    signal diagonal_x : natural;
begin
```

```
p_next : process (vga_clk)
begin
    if rising_edge(vga_clk) then
        next_btn <= KEY(1);
        next_delay <= next_btn;
        next_pulse <= (not next_btn) and next_delay;
    end if;
end process;

p_blanking : process (vga_clk)
begin
    if rising_edge(vga_clk) then
        if vga_x < vga_max_x then
            VGA_R <= pixel_color(11 downto 8);
            VGA_G <= pixel_color(7 downto 4);
            VGA_B <= pixel_color(3 downto 0);
        else
            VGA_R <= "0000";
            VGA_G <= "0000";
            VGA_B <= "0000";
        end if;
    end if;
end process;

p_state_machine : process (vga_clk)
begin
    if rstn_btn = '0' then
        flag_state <= FRANCE;
    elsif rising_edge(vga_clk) then
        case flag_state is
            when FRANCE =>
                if next_pulse = '1' then
                    flag_state <= ITALY;
                end if;
                if vga_x < flag_v_x1 then
                    pixel_color <= BLUE;
                elsif vga_x < flag_v_x2 then
                    pixel_color <= WHITE;
                else
                    pixel_color <= RED;
                end if;
            when ITALY =>
                if next_pulse = '1' then
                    flag_state <= IRELAND;
                end if;
                if vga_x < flag_v_x1 then
                    pixel_color <= GREEN;
                elsif vga_x < flag_v_x2 then
                    pixel_color <= WHITE;
                else
                    pixel_color <= RED;
                end if;
            when IRELAND =>
                if next_pulse = '1' then
                    flag_state <= BELGIUM;
                end if;
                if vga_x < flag_v_x1 then
                    pixel_color <= GREEN;
                elsif vga_x < flag_v_x2 then
```

```
                pixel_color <= WHITE;
            else
                pixel_color <= ORANGE;
            end if;
        when BELGIUM =>
            if next_pulse = '1' then
                flag_state <= MALI;
            end if;
            if vga_x < flag_v_x1 then
                pixel_color <= BLACK;
            elsif vga_x < flag_v_x2 then
                pixel_color <= YELLOW;
            else
                pixel_color <= RED;
            end if;
        when MALI =>
            if next_pulse = '1' then
                flag_state <= CHAD;
            end if;
            if vga_x < flag_v_x1 then
                pixel_color <= LIME;
            elsif vga_x < flag_v_x2 then
                pixel_color <= BANANA;
            else
                pixel_color <= RED;
            end if;
        when CHAD =>
            if next_pulse = '1' then
                flag_state <= NIGERIA;
            end if;
            if vga_x < flag_v_x1 then
                pixel_color <= AZULE;
            elsif vga_x < flag_v_x2 then
                pixel_color <= BANANA;
            else
                pixel_color <= CRIMSON;
            end if;
        when NIGERIA =>
            if next_pulse = '1' then
                flag_state <= IVORY;
            end if;
            if vga_x < flag_v_x1 then
                pixel_color <= GREEN;
            elsif vga_x < flag_v_x2 then
                pixel_color <= WHITE;
            else
                pixel_color <= GREEN;
            end if;
        when IVORY =>
            if next_pulse = '1' then
                flag_state <= POLAND;
            end if;
            if vga_x < flag_v_x1 then
                pixel_color <= ORANGE;
            elsif vga_x < flag_v_x2 then
                pixel_color <= WHITE;
            else
                pixel_color <= GREEN;
            end if;
        when POLAND =>
```

```
                if next_pulse = '1' then
                    flag_state <= GERMANY;
                end if;
                if vga_y < flag_h_mid then
                    pixel_color <= WHITE;
                else
                    pixel_color <= RED;
                end if;
            when GERMANY =>
                if next_pulse = '1' then
                    flag_state <= AUSTRIA;
                end if;
                if vga_y < flag_h_y1 then
                    pixel_color <= BLACK;
                elsif vga_y < flag_h_y2 then
                    pixel_color <= RED;
                else
                    pixel_color <= BANANA;
                end if;
            when AUSTRIA =>
                if next_pulse = '1' then
                    flag_state <= CONGO;
                end if;
                if vga_y < flag_h_y1 then
                    pixel_color <= RED;
                elsif vga_y < flag_h_y2 then
                    pixel_color <= WHITE;
                else
                    pixel_color <= RED;
                end if;
            when CONGO =>
                if next_pulse = '1' then
                    flag_state <= FRANCE;
                end if;
                if vga_x < diagonal_x then
                    pixel_color <= GREEN;
                elsif vga_x < diagonal_x + 160 then
                    pixel_color <= YELLOW;
                else
                    pixel_color <= RED;
                end if;

                if vga_y = 0 then
                    diagonal_x <= 480;
                elsif vga_x = 0 then
                    diagonal_x <= diagonal_x - 1;
                end if;
            when others =>
                flag_state <= FRANCE;
        end case;
    end if;
end process;

rstn_btn <= KEY(0);

VG0: entity work.VGA_Sync(behave)
port map (
    vga_clk => vga_clk,
    vga_hs => VGA_HS,
    vga_x => vga_x,
```

```
        vga_vs => VGA_VS,
        vga_y => vga_y
    );

    PL0: entity work.PLL(syn)
    port map (
        inclk0 => MAX10_CLK1_50,
        c0  => vga_clk
    );

end behave;
```

Appendix B

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

entity VGA_Sync is
    port (
        vga_clk : in std_logic;
        vga_hs : out std_logic;
        vga_x : out natural;
        vga_vs : out std_logic;
        vga_y : out natural
    );
end VGA_Sync;


architecture behave of VGA_Sync is

    constant hs_count_max : natural := 800;
    constant hs_disp : natural := 640;
    constant hs_fp : natural := 18;
    constant hs_sync : natural := 92;
    constant hs_bp : natural := 50;
    signal hs_count : natural := 0;

    constant vs_count_max : natural := 525;
    constant vs_disp : natural := 480;
    constant vs_fp : natural := 10;
    constant vs_sync : natural := 12;
    constant vs_bp : natural := 33;
    signal vs_count : natural := 0;

begin

    p_hs_count : process (vga_clk)
    begin
        if rising_edge(vga_clk) then
            if hs_count >= hs_count_max-1 then
                hs_count <= 0;
            else
                hs_count <= hs_count + 1;
            end if;
        end if;
    end process;

    p_hs : process (vga_clk)
    begin
        if rising_edge(vga_clk) then
            if hs_count > hs_disp + hs_fp and hs_count < hs_count_max - hs_bp
then
                vga_hs <= '0';
            else
                vga_hs <= '1';
            end if;
        end if;
    end process;
```

```
    p_vs_count : process (vga_clk)
    begin
        if rising_edge(vga_clk) then
            if vs_count >= vs_count_max-1 then
                vs_count <= 0;
            elsif hs_count = hs_count_max-1 then
                vs_count <= vs_count + 1;
            end if;
        end if;
    end process;

    p_vs : process (vga_clk)
    begin
        if rising_edge(vga_clk) then
            if vs_count > vs_disp + vs_fp and vs_count < vs_count_max - vs_bp
then
                vga_vs <= '0';
            else
                vga_vs <= '1';
            end if;
        end if;
    end process;

    vga_x <= hs_count;
    vga_y <= vs_count;

end behave;
```