EEL 4712C - Digital Design: Lab Report 4

Cole Rottenberg
11062528

March 3rd, 2024

Lab Report

Problem Statement

The lab is broken into 4 individual parts and two groups of two. The first part deals with static timing analysis and the second part deals with implementing a basic VGA driver to display a box on a monitor. The third part deals with implementing a VGA driver to display a moving box on a monitor. The fourth part deals applying the same static timing analysis from the first part to the moving box from the third part.

The 2nd and 3rd part of the lab are the critical design parts of the lab that help us explore the capabilities of the VGA driver and how to implement it. The first and fourth part are more about understanding the timing of the VGA driver and how to properly implement it. The part two is split into three different entities(primarily), the VGA, the VGA sync generator, and the top level component. The inputs to the top level component are relatively static and aren't changing. The outputs are the VGA signals that are sent to the monitor. The signals sent to the monitor are the h_sync, v_sync, and the RGB signals. The system is designed to display a box on the monitor. The sync generator is responsible for generating the sync signals, however the VGA entity is responsible for applying the logic to these signals to display the box on the monitor.

Design

The VGA sync generator is consisted of a single clocked process that iterates through a double condition statement. The first conditionally block check if the `h_count` is equal the `H_MAX` constant defined in our package. If they are equal to eachother, we reset the counter and go onto the next conditional block, which checks if the `v_count` is equal to the `V_MAX` constant defined in our package. If they are equal to eachother, we reset the counter and exit the process. However, if we the first conditional is false, we increment the `h_count` counter. If the second conditional is false, we increment the `v_count` counter. On a conceptual level this builds a sweeping motion across the rows and then columns until the end. The second part of the generator exist outside a process as three conditions for the `h_sync`, `v_sync`, and `video_on` signals.

These signals are then passed up to the VGA entity which uses the `h_count` and `v_count` signals to determine the position of the box on the screen. The `draw` clocked process uses these counts to determine if the current pixel is within the define constants of: `CENTER_X_START`, `CENTER_X_END`, `CENTER_Y_START`, and `CENTER_Y_END`. If the pixel is within these bounds, the `red`, `green`, and `blue` signals are set to "0111", "0011", and "1011" respectively. Outside of the process, existing within the architecture, the `h_sync`, `v_sync`, and `video_on` signals are being outputted to the top level component.

The design of the 3rd part builds of the previous part as it uses an identical VGA sync generator. The VGA entity is modified and new values are used to move the box across the screen as well as change the direction of the box. We also need to make use of a clock divider to slow down the clock signal to 1Hz. This `slow_clk` signal is then used to drive our `obj_move` process. This process is responsible for moving the box across the screen. The `obj_move` process uses a new set of signals and constants to determine the position of the box on the screen. It also uses logic to control directional changes when the box reaches the vertical or horizontal bounds of the screen.

The top level entity controls the final output of the VGA signals to the monitor. The VGA entity passes the following outputs to the top level entity: `h_sync`, `v_sync`, `red`, `green`, `blue`, and `video_on`. The top level entity then passes these signals to the VGA port which is connected to the monitor.

Implementation

The implementation process was relatively straightforward with a "few" hiccups along the way. The first part of implementation was to implement the VGA sync generator. Knowing the defined input of specific clock signals left us to worry about the logic of iterating through the valid sections of the monitor. As seen in the code below, Listing 1, the process iterates through the horizontal and vertical counters. The second part of the implementation was to implement the VGA entity. This was a bit more complex as we had to determine the position of the box on the screen. However, given the constants defined in the package, we were able to easily determine the position of the box on the screen. In Listing 2, we can see the logic used to determine the position of the box on the screen. Keep in mind some description comments were removed for brevity.

In 3rd part of the lab, we had to implement a clock divider to slow down the clock signal to 1Hz. Considering we had already implemented a clock divider in a previous lab, we needed to change a few names of the IO and some generic logic for clock divider control. The implementation of the clock divider can be seen in Listing 3. The implementation of the VGA entity was more comlex as we had to implement a new set of signals and constants to move the box across the screen. We also had to implement logic to control the direction of the box when it reached the bounds of the screen. The implementation of the VGA entity can be seen in Listing 4.

Testing

There were multiple methods of testing the design and implementation. The 1st method was to use the newly introduce timing analyzer tool in the Quartus Suite. This tool allowed use to check the timing of the VGA signals and ensure they were within the proper range. As seen in Figure , by adjusting our clock capabilities to match our board's true clock period, we have ample time to display the box on the screen. The 2nd method was to use the top level testbench. The given testbench was used to simulate the VGA signals and ensure the box was being displayed on the screen. I also designed my own testbench to test the VGA entity for the 3rd part of the lab. This testbench was used to simulate the VGA signals and ensure the box was moving across the screen as expected.

Conclusions

The work done in this lab was a great learning experience. The lab helped me make a contextual jump from an abstract concepts such as the VGA sync generator and inner workings of a VGA driver to a more concrete understanding of how to implement these concepts. The lab encountered a few update issues with the VGA entity on the 3rd part of the lab. We ended up implementing a temporary variable to allow for sequential updates to affect the movement and position signals. In the future, I would like to improve my understanding of bitmap logic and how to implement it in VHDL. This would allow me to create more complex designs and understand the inner workings of the VGA driver.

Appendix

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4 use work.VGA_LIB.all;
5
6 entity vga_sync_gen is
7     port (
```

```
        clk       : in std_logic;
        rst       : in std_logic;
        h_count   : out std_logic_vector(COUNT_RANGE);
        v_count   : out std_logic_vector(COUNT_RANGE);
        h_sync    : out std_logic;
        v_sync    : out std_logic;
        video_on  : out std_logic
    );
end entity;

architecture bhv of vga_sync_gen is
    -- Counters
    signal h_counter : unsigned(COUNT_RANGE) := (others => '0');
    signal v_counter : unsigned(COUNT_RANGE) := (others => '0');

begin
    -- Counter Loop
    process(clk)
    begin
        if rising_edge(clk) then
            -- reset counters
            if h_counter = H_MAX then
                -- reset horizontal counter
                h_counter <= (others => '0');
                -- reset vertical counter
                if v_counter = V_MAX then
                    v_counter <= (others => '0');
                else
                    v_counter <= v_counter + 1;
                end if;
            else
                h_counter <= h_counter + 1;
            end if;
        end if;
    end process;

    -- Output counters
    h_count <= std_logic_vector(h_counter);
    v_count <= std_logic_vector(v_counter);

    -- Output Sync signal logic (active low)
    h_sync <= '0' when h_counter >= HSYNC_BEGIN and h_counter <=
       HSYNC_END else '1';
    v_sync <= '0' when v_counter >= VSYNC_BEGIN and v_counter <=
       VSYNC_END else '1';
    video_on <= '1' when h_counter < H_DISPLAY_END and v_counter <
       V_DISPLAY_END else '0';

end architecture;
```

Listing 1: VGA Sync Generator for Part 2

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

```
4  use work.vga_lib.all;
5
6  entity vga is
7      port (clk              : in  std_logic;
8            rst              : in  std_logic;
9            en               : in  std_logic;
10 ____ switch           : in  std_logic_vector(9 downto 0) := (others =>
       '0');
11           img_pos          : in  std_logic_vector(2 downto 0) := (others
                => '0');
12           red, green, blue : out std_logic_vector(3 downto 0) := (others
                => '0');
13           h_sync, v_sync   : out std_logic;
14           video_on         : out std_logic);
15 end vga;
16
17 architecture default_arch of vga is
18
19     signal v_count : std_logic_vector(COUNT_RANGE);
20     signal h_count : std_logic_vector(COUNT_RANGE);
21 __signal temp_h_sync, temp_v_sync : std_logic := '0';
22 __signal temp_video_on : std_logic := '0';
23   -- VGA_SYNC_GEN Signals
24 begin ____
25 __-- VGA MAIN BEGINS
26
27 __sync: entity work.vga_sync_gen
28 ____port map (clk => clk,
29 ________    rst => rst,
30 ________    h_count => h_count,
31 ________    v_count => v_count,
32 ________    h_sync => temp_h_sync,
33 ________v_sync => temp_v_sync,
34 ________video_on => temp_video_on);
35    -- VGA_SYNC_GEN ENDS
36
37 __draw: process(clk, rst)
38 __begin
39 ____if rising_edge(clk) then
40 ______if rst = '0' then
41 ________if unsigned(h_count) >= CENTERED_X_START and unsigned(h_count) <=
       CENTERED_X_END and
42                 unsigned(v_count) >= CENTERED_Y_START and
                      unsigned(v_count) <= CENTERED_Y_END and
43                 temp_video_on = '1' then
44 __________red <= "0111";
45 __________green <= "0011";
46 __________blue <= "1011";
47 ______else
48 __________red <= "0000";
49 __________green <= "0000";
50 __________blue <= "0000";
51 ______end if;
52 ____end if;
```

```
53      end if;
54  end process draw;
55
56
57  -- VGA MAIN ENDS
58  h_sync <= temp_h_sync;
59  v_sync <= temp_v_sync;
60  video_on <= temp_video_on;
61  end default_arch;
```

Listing 2: VGA Entity for Part 2

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  -- Clock Divider turns the 500MHz clock into 50MHz
6  -- The 50Mhz clock is used to drive the VGA display
7  -- 500_000_000 / 10 = 50_000_000
8
9  entity clk_div is
10     generic(
11         clk_in_freq : integer := 1;
12         clk_out_freq : integer := 1
13     );
14     port(
15         clk_in : in std_logic;
16         rst : in std_logic := '0';
17         clk_out : out std_logic
18     );
19 end clk_div;
20
21 architecture Behavioral of clk_div is
22     -- Setting COUNTER_MAX to generic input_frequency
23     constant COUNTER_MAX : integer := clk_in_freq / clk_out_freq - 1;
24     signal counter : integer range 0 to COUNTER_MAX := 0;
25     signal temp_clk : STD_LOGIC := '0';
26
27 begin
28     process(clk_in, rst)
29     begin
30         if rst = '1' then
31             counter <= 0;
32             temp_clk <= '0';
33         elsif rising_edge(clk_in) then
34             if counter = COUNTER_MAX then
35                 counter <= 0;
36                 temp_clk <= not temp_clk;
37             else
38                 counter <= counter + 1;
39             end if;
40         end if;
41     end process;
42
43     clk_out <= temp_clk;
```

```
44
45 end Behavioral;
```

Listing 3: Clock Divider for Part 3

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use work.vga_lib.all;
5
6  entity vga is
7      port (clk              : in  std_logic;
8            rst              : in  std_logic;
9            en               : in  std_logic;
10 ____ switch           : in  std_logic_vector(9 downto 0) := (others =>
       '0');
11           img_pos          : in  std_logic_vector(2 downto 0) := (others
              => '0');
12           red, green, blue : out std_logic_vector(3 downto 0) := (others
              => '0');
13           h_sync, v_sync   : out std_logic;
14           video_on         : out std_logic);
15 end vga;
16
17 architecture default_arch of vga is
18
19     signal v_count : std_logic_vector(COUNT_RANGE);
20     signal h_count : std_logic_vector(COUNT_RANGE);
21     -- Clocks
22     signal p_clk : std_logic := '0'; -- Pixel clock
23     signal slow_clk : std_logic := '0'; -- Slow clock
24     -- Internal Temp signals
25     signal temp_h_sync, temp_v_sync, temp_video_on : std_logic;
26 ____-- x and y coordinates of OBJ
27 ____signal x_pos, y_pos : integer := 200; -- 200 is the center of the
       screen
28 ____signal mov_x, mov_y : integer := 1;
29 ____-- Constants
30 ____constant speed : integer := 2;
31 ____constant size : integer := 64;
32 ____constant X_MAX : integer := 639;
33 ____constant Y_MAX : integer := 479;
34 ____signal v_on : std_logic := '0';
35
36
37
38
39 begin ____
40 __-- Slow Clock Divider splits the 50MHz clock into 1Hz
41 __clk_div: entity work.clk_div
42 ____generic map(
43 ______clk_in_freq => 50e6,
44 ______clk_out_freq => 1
45 ____)
46 ____port map(
```

```
47      clk_in => clk,
48      rst => rst,
49      clk_out => slow_clk
50  );
51
52  -- VGA SYNC_GEN BEGINS
53  sync: entity work.vga_sync_gen
54      port map (clk => clk,
55                rst => rst,
56                h_count => h_count,
57                v_count => v_count,
58                h_sync => temp_h_sync,
59              v_sync => temp_v_sync,
60              video_on => temp_video_on);
61     -- VGA_SYNC_GEN ENDS
62
63     -- The object moves around the screen and will bounce off the edges
64     obj_move: process(slow_clk, rst)
65      begin
66          if rising_edge(slow_clk) then
67              if rst = '1' then
68                  x_pos <= 200;
69                  y_pos <= 200;
70                  mov_x <= 1;
71                  mov_y <= 1;
72              elsif en = '1' then
73                  if x_pos + size >= X_MAX or x_pos <= 0 then
74                      mov_x <= -mov_x;
75                  end if;
76                  if y_pos + size >= Y_MAX or y_pos <= 0 then
77                      mov_y <= -mov_y;
78                  end if;
79                  x_pos <= x_pos + mov_x * speed;
80                  y_pos <= y_pos + mov_y * speed;
81              end if;
82          end if;
83      end process obj_move;
84
85
86  draw: process(clk, rst)
87  begin
88      if rising_edge(clk) then
89          if rst = '0' then
90              if unsigned(h_count) >= to_unsigned(x_pos, h_count'length) and
       unsigned(h_count) < to_unsigned(x_pos + size, h_count'length) and
91                  unsigned(v_count) >= to_unsigned(y_pos, v_count'length)
                        and unsigned(v_count) < to_unsigned(y_pos + size,
                        v_count'length) and
92                  temp_video_on = '1' then
93                  red <= "0111";
94                  green <= "0011";
95                  blue <= "1011";
96              else
97                  red <= "0000";
```

```
                green <= "0000";
                blue <= "0000";
            end if;
        end if;
    end if;
end process draw;

-- VGA MAIN ENDS
h_sync <= temp_h_sync;
v_sync <= temp_v_sync;
video_on <= temp_video_on;
end default_arch;
```

Listing 4: VGA Entity for Part 3

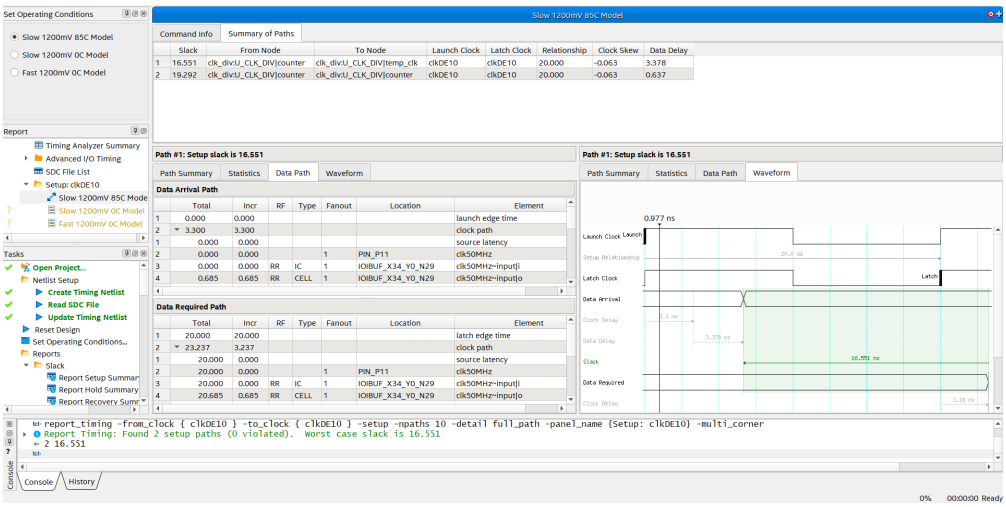

Figure 1: VGA Part 3 Timing Analysis