Lab #8

ECE 4304 Spring 2021

Professor Aly

California State Polytechnic University, Pomona

Ahiezer Lopez	Joe Si	Sander Zuckerman
ahiezerlopez@cpp.edu	joesi@cpp.edu	sazuckerman@cpp.edu
Bronco ID:012715521	Bronco ID: 012636091	Bronco ID: 012644671

4/25/2021

Objective:
Students will create a VGA design with UART controls. The lettering will be displayed on the monitor and switches will be used to change the color of the letters while the UART controls change the position of the letters.

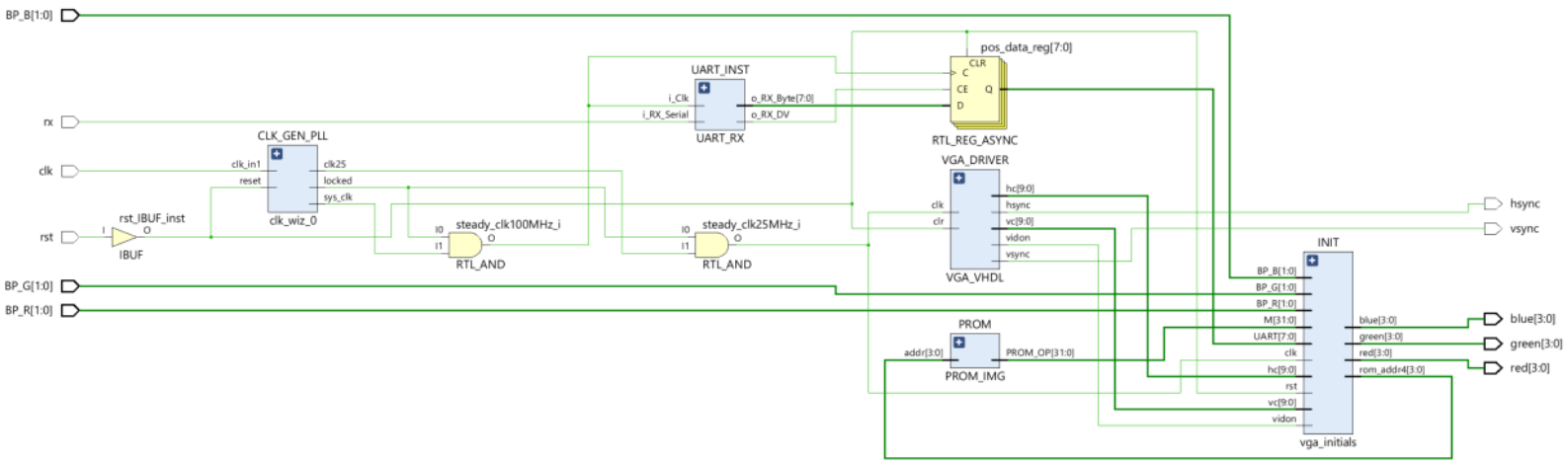
Materials:
- FPGA (Nexys A7-100T)
- Vivado Software
- Computer

Contributions:
This lab was done with the work effort of every person in the group. For this lab, Ahiezer was responsible for the color changes of the lettering and helped with any sort of debugging and or problems with other portions, while Sander was mainly responsible on the UART implementation into the FPGA board, and lastly Joe was responsible of the position changes of the lettering on the monitor.

Design Process:
Using previous code from other labs such as the UART modules, the VGA code provided by lecture, and with the use of Real Term we were able to create a working design. Initially we had implemented the FIFO from previous labs to help with the transferring of data, however that caused more harm than good as it changed the data in between. We decided to fully remove the FIFO buffers and grab the data from Real Term directly. We also decided to switch from Tera Term to Real Term as it made it easier to send data to the FPGA. The data sent from UART, which are 8 bits of data, allowed us to put it through a case statement which had predetermined locations on the monitor and ultimately allowed the UART to be in control of the location of the lettering throughout the monitor. Our main problems throughout the code was getting the UART data to send correctly as we first used the FIFO which caused problems then we were able to fix it through removing the FIFO entirely and switching to Real Term to send the data.

Design:
The UART module will be taken in, still taking the 1 bit serial bits however it will fully load into the module and output the desired 8 bits. It is then transferred to the VGA_Initials module where it will essentially replace the switches from the original VGA code designed provided from the lecture and is put through a case statement where it will change the location of the lettering of the monitor. The Color changing was also provided by multiple case statements which change the values of the red, green, and blue data so it can create different shades of colors.

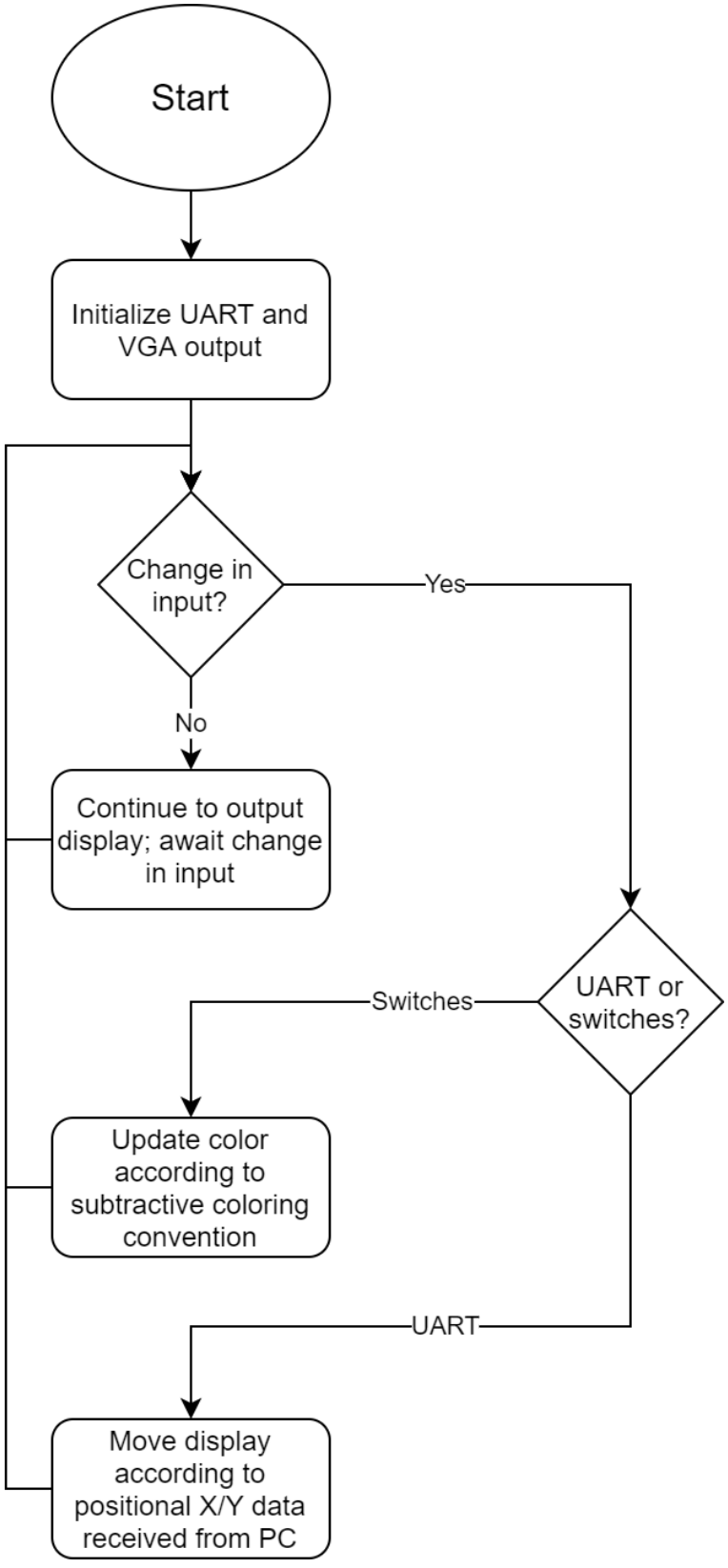
Circuit Diagram:



Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP
✓ synth_1	constrs_1	synth_design Complete!								108	69	0.0	0	0
✓ impl_1	constrs_1	write_bitstream Complete!	4.186	0.000	0.107	0.000	0.000	0.194	0	107	71	0.0	0	0

Implementation:

Logic Flow Diagram



Design:

Top Module

VGA_INITAILS TOP

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24  use IEEE.NUMERIC_STD.ALL;
25  use IEEE.STD_LOGIC_UNSIGNED.ALL;
26  use IEEE.math_real.all;
27  use IEEE.STD_LOGIC_ARITH.ALL;
28
29  -- Uncomment the following library declaration if using
30  -- arithmetic functions with Signed or Unsigned values
31  --use IEEE.NUMERIC_STD.ALL;
32
33  -- Uncomment the following library declaration if instantiating
34  -- any Xilinx leaf cells in this code.
35  --library UNISIM;
36  --use UNISIM.VComponents.all;
37
38  entity vga_initials_top is
39   generic (strip_hpixels :positive:= 800;    -- Value of pixels in a horizontal line = 800
40            strip_vlines  :positive:= 512;    -- Number of horizontal lines in the display = 521
41            strip_hbp     :positive:= 144;    -- Horizontal back porch = 144 (128 + 16)
42            strip_hfp     :positive:= 784;    -- Horizontal front porch = 784 (128+16 + 640)
43            strip_vbp     :positive:= 31;     -- Vertical back porch = 31 (2 + 29)
44            strip_vfp     :positive:= 511;     -- Vertical front porch = 511 (2+29+ 480)
45            clk_bits      :integer := 869
46           );
47      Port ( clk  : in STD_LOGIC;
48             rst  : in STD_LOGIC;
49             BP_R: in STD_LOGIC_VECTOR(1 downto 0);
50             BP_G: in STD_LOGIC_VECTOR(1 downto 0);
51             BP_B: in STD_LOGIC_VECTOR(1 downto 0);
52             hsync: out STD_LOGIC;
53             vsync: out STD_LOGIC;
54             rx   : in std_logic;
55             red  : out STD_LOGIC_VECTOR (3 downto 0);
56             green: out STD_LOGIC_VECTOR (3 downto 0);
57             blue : out STD_LOGIC_VECTOR (3 downto 0)
58           );
59  end vga_initials_top;
60
61  architecture Behavioral of vga_initials_top is
62
63
64
65  component VGA_VHDL
```

```
component VGA_VHDL
    -- Note these numbers are different from a resolution to another (This is for 640x480)
    generic(hpixels :positive:= 800;    -- Value of pixels in a horizontal line = 800
            vlines  :positive:= 512;    -- Number of horizontal lines in the display = 521
            hbp     :positive:= 144;    -- Horizontal back porch = 144 (128 + 16)
            hfp     :positive:= 784;    -- Horizontal front porch = 784 (128+16 + 640)
            vbp     :positive:= 31;     -- Vertical back porch = 31 (2 + 29)
            vfp     :positive:= 511     -- Vertical front porch = 511 (2+29+ 480)
           );
    Port ( clk   : in  STD_LOGIC;
           clr   : in  STD_LOGIC;
           hsync : out STD_LOGIC;
           vsync : out STD_LOGIC;
           hc    : out STD_LOGIC_VECTOR (9 downto 0);
           vc    : out STD_LOGIC_VECTOR (9 downto 0);
           vidon : out STD_LOGIC
         );
end component;

component clk_wiz_0
     port (
           clk25  : out std_logic;
           reset  : in  std_logic;
           locked : out std_logic;
           clk_in1: in  std_logic;
           sys_clk: out std_logic
          );
end component;

component PROM_IMG
    generic(DEPTH    :positive:= 16;
            DATA_SIZE:positive:= 32
           );
    Port    ( addr    : in  STD_LOGIC_VECTOR (integer(ceil(log2(real(DEPTH))))-1 downto 0);
              PROM_OP : out STD_LOGIC_VECTOR (DATA_SIZE-1 downto 0)
           );
end component;


component vga_initials
    generic (hbp:positive:=144; vbp:positive:=31; W:positive:=32; H:positive:= 16 );
    Port (
           clk      : in  STD_LOGIC;
```

```
            clk      : in  STD_LOGIC;
            rst      : in  STD_LOGIC;
            vidon    : in  STD_LOGIC;
            hc       : in  STD_LOGIC_VECTOR (9  downto 0);
            vc       : in  STD_LOGIC_VECTOR (9  downto 0);
            M        : in  STD_LOGIC_VECTOR (31 downto 0);
            UART       : in  STD_LOGIC_VECTOR (7  downto 0);
            BP_R: in STD_LOGIC_VECTOR(1 downto 0);
            BP_G: in STD_LOGIC_VECTOR(1 downto 0);
            BP_B: in STD_LOGIC_VECTOR(1 downto 0);
            rom_addr4: out STD_LOGIC_VECTOR (3  downto 0);
            red      : out STD_LOGIC_VECTOR (3  downto 0);
            green    : out STD_LOGIC_VECTOR (3  downto 0);
            blue     : out STD_LOGIC_VECTOR (3  downto 0)
          );
end component;

component UART_RX is
  generic (
    g_CLKS_PER_BIT : integer := 869     -- Needs to be set correctly
    );
  port (
    i_Clk       : in  std_logic;
    i_RX_Serial : in  std_logic;
    o_RX_DV     : out std_logic;
    o_RX_Byte   : out std_logic_vector(7 downto 0)
    );
end component;

signal clk25MHz       :std_logic;
signal sys_clk          : std_logic;
signal locked_pll     :std_logic;
signal steady_clk25MHz:std_logic;
signal steady_clk100MHz:std_logic;

signal hc, vc:std_logic_vector(9 downto 0);
signal video_on :std_logic;

signal IMG:std_logic_vector(31 downto 0);
signal rom_addr4:std_logic_vector(3 downto 0);

signal rx_data, pos_data : std_logic_vector(7 downto 0) := (others => '0');
signal rx_valid : std_logic;
```

```
UART_INST : UART_RX
generic map(g_CLKS_PER_BIT=> clk_bits)
port map(
        i_Clk         => steady_clk100MHz,
        i_RX_Serial       => rx,
        o_RX_DV     => rx_valid,
        o_RX_Byte   => rx_data
);

UART_VALID : process (steady_clk100MHz, rst) begin

    if(rst = '1') then
        pos_data <= (others => '0');
    elsif( rising_edge(steady_clk100MHz) ) then
        if(rx_valid = '1') then
            pos_data <= rx_data;
        end if;
    end if;

end process UART_VALID;

CLK_GEN_PLL: clk_wiz_0 port map (
                                  clk25   => clk25MHz,
                                  reset   => rst,
                                  locked  => locked_pll,
                                  clk_in1 => clk,
                                  sys_clk => sys_clk
                                );
steady_clk25MHz <= locked_pll and clk25MHz;
steady_clk100MHz <= locked_pll and sys_clk;



VGA_DRIVER: VGA_VHDL
                    generic map (
                                  hpixels => strip_hpixels,
                                  vlines  => strip_vlines,
                                  hbp     => strip_hbp,
                                  hfp     => strip_hfp,
                                  vbp     => strip_vbp,
                                  vfp     => strip_vfp

                                  )
```

```
                                  clk   => steady_clk25MHz,
                                  clr   => rst,
                                  hsync => hsync,
                                  vsync => vsync,
                                  hc    => hc,
                                  vc    => vc,
                                  vidon => video_on
                                 );

INIT: vga_initials
    generic map ( hbp =>144,
                  vbp =>31,
                  W   =>32,
                  H   =>16
                )
    Port map (
                   clk       => steady_clk25MHz,
                   rst       => rst,
                   vidon     => video_on,
                   hc        => hc,
                   vc        => vc,
                   M         => IMG,
                   UART      => pos_data,
                   BP_R      => BP_R,
                   BP_G      => BP_G,
                   BP_B      => BP_B,
                   rom_addr4 => rom_addr4,
                   red       =>  red,
                   green     =>  green,
                   blue      =>  blue
         );

PROM: PROM_IMG generic map (
                              DEPTH     => 16 ,
                              DATA_SIZE => 32
                              )
              port map (
                          addr    => rom_addr4,
                          PROM_OP => IMG
                        );


end Behavioral;
```

Layer Modules

UART_RX

```
1  library ieee;
2  use ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.all;
4
5  entity UART_RX is
6    generic (
7      g_CLKS_PER_BIT : integer := 869     -- Needs to be set correctly
8      );
9    port (
10     i_Clk       : in  std_logic;
11     i_RX_Serial : in  std_logic;
12     o_RX_DV     : out std_logic;
13     o_RX_Byte   : out std_logic_vector(7 downto 0)
14     );
15 end UART_RX;
16
17
18 architecture rtl of UART_RX is
19
20   type t_SM_Main is (s_Idle, s_RX_Start_Bit, s_RX_Data_Bits,
21                      s_RX_Stop_Bit, s_Cleanup);
22   signal r_SM_Main : t_SM_Main := s_Idle;
23
24   signal r_RX_Data_R : std_logic := '0';
25   signal r_RX_Data   : std_logic := '0';
26
27   signal r_Clk_Count : integer range 0 to g_CLKS_PER_BIT-1 := 0;
28   signal r_Bit_Index : integer range 0 to 7 := 0;  -- 8 Bits Total
29   signal r_RX_Byte   : std_logic_vector(7 downto 0) := (others => '0');
30   signal r_RX_DV     : std_logic := '0';
31
32 begin
33
34   -- Purpose: Double-register the incoming data.
35   -- This allows it to be used in the UART RX Clock Domain.
36   -- (It removes problems caused by metastabiliy)
37   p_SAMPLE : process (i_Clk)
38   begin
39     if rising_edge(i_Clk) then
40       r_RX_Data_R <= i_RX_Serial;
41       r_RX_Data   <= r_RX_Data_R;
42     end if;
43   end process p_SAMPLE;
44
```

```
-- Purpose: Control RX state machine
p_UART_RX : process (i_Clk)
begin
  if rising_edge(i_Clk) then

    case r_SM_Main is

      when s_Idle =>
        r_RX_DV     <= '0';
        r_Clk_Count <= 0;
        r_Bit_Index <= 0;

        if r_RX_Data = '0' then       -- Start bit c
          r_SM_Main <= s_RX_Start_Bit;
        else
          r_SM_Main <= s_Idle;
        end if;


      -- Check middle of start bit to make sure it's
      when s_RX_Start_Bit =>
        if r_Clk_Count = (g_CLKS_PER_BIT-1)/2 then
          if r_RX_Data = '0' then
            r_Clk_Count <= 0;  -- reset counter sinc
            r_SM_Main   <= s_RX_Data_Bits;
          else
            r_SM_Main   <= s_Idle;
          end if;
        else
          r_Clk_Count <= r_Clk_Count + 1;
          r_SM_Main   <= s_RX_Start_Bit;
        end if;


      -- Wait g_CLKS_PER_BIT-1 clock cycles to sampl
      when s_RX_Data_Bits =>
        if r_Clk_Count < g_CLKS_PER_BIT-1 then
          r_Clk_Count <= r_Clk_Count + 1;
          r_SM_Main   <= s_RX_Data_Bits;
        else
          r_Clk_Count             <= 0;
          r_RX_Byte(r_Bit_Index) <= r_RX_Data;


          -- Check if we have sent out all bits
```

```
90              if r_Bit_Index < 7 then
91                r_Bit_Index <= r_Bit_Index + 1;
92                r_SM_Main   <= s_RX_Data_Bits;
93              else
94                r_Bit_Index <= 0;
95                r_SM_Main   <= s_RX_Stop_Bit;
96              end if;
97            end if;
98
99
100         -- Receive Stop bit.  Stop bit = 1
101         when s_RX_Stop_Bit =>
102           -- Wait g_CLKS_PER_BIT-1 clock cycles for Stop
103           if r_Clk_Count < g_CLKS_PER_BIT-1 then
104             r_Clk_Count <= r_Clk_Count + 1;
105             r_SM_Main   <= s_RX_Stop_Bit;
106           else
107             r_RX_DV     <= '1';
108             r_Clk_Count <= 0;
109             r_SM_Main   <= s_Cleanup;
110           end if;
111
112
113         -- Stay here 1 clock
114         when s_Cleanup =>
115           r_SM_Main <= s_Idle;
116           r_RX_DV   <= '0';
117
118
119         when others =>
120           r_SM_Main <= s_Idle;
121
122       end case;
123     end if;
124   end process p_UART_RX;
125
126   o_RX_DV   <= r_RX_DV;
127   o_RX_Byte <= r_RX_Byte;
128
129 end rtl;
```

VGA_INITIALS

```
38  entity vga_initials is
39      generic (hbp:positive:=144; vbp:positive:=31; W:positive:=32; H:positive:= 16 );
40      Port (
41             clk      : in  STD_LOGIC;
42             rst      : in  STD_LOGIC;
43             vidon    : in  STD_LOGIC;
44             hc       : in  STD_LOGIC_VECTOR (9  downto 0);
45             vc       : in  STD_LOGIC_VECTOR (9  downto 0);
46             M        : in  STD_LOGIC_VECTOR (31 downto 0);
47             UART       : in  STD_LOGIC_VECTOR (7  downto 0);
48             BP_R     : in  STD_LOGIC_VECTOR (1 downto 0);
49             BP_G     : in  STD_LOGIC_VECTOR (1 downto 0);
50             BP_B     : in  STD_LOGIC_VECTOR (1 downto 0);
51             rom_addr4: out STD_LOGIC_VECTOR (3  downto 0);
52             red      : out STD_LOGIC_VECTOR (3  downto 0);
53             green    : out STD_LOGIC_VECTOR (3  downto 0);
54             blue     : out STD_LOGIC_VECTOR (3  downto 0)
55           );
56  end vga_initials;
57
58  architecture Behavioral of vga_initials is
59
60
61  signal C1, R1, rom_addr, rom_pix: std_logic_vector(10 downto 0);
62  signal spriteon, R, G, B: std_logic;
63
64  begin
65
66  C1 <= ("00" & UART(3 downto 0) & "00001");
67  R1 <= ("00" & UART(7 downto 4) & "00001");
68  rom_addr  <= vc - vbp- R1;
69  rom_pix   <= hc - hbp - C1;
70  rom_addr4 <= rom_addr(3 downto 0);
71
72  -- Enable sprite video out when within the sprite region
73
74  REGION_ACTIVE:process(clk,rst)
75                begin
76                if (rst = '1') then
77                  spriteon <= '0';
78                elsif(rising_edge(clk)) then
79                  if (hc >= C1 + hbp) and (hc < C1 + hbp+ W) and (VC >= R1 +vbp) and (vc < R1 + vbp +H) then
```

```
89  OUTP_COLOR:process(clk,rst)
90             begin
91              if (rst = '1') then
92                  red   <= (others =>'0');
93                  green <= (others =>'0');
94                  blue  <= (others =>'0');
95              elsif(rising_edge(clk)) then
96                  if (spriteon = '1' and vidon = '1') then
97
98                      R <= M(31-conv_integer(rom_pix));
99                      G <= M(31-conv_integer(rom_pix));
100                     B <= M(31-conv_integer(rom_pix));
101
102                     case conv_integer(BP_R) is
103                     when 0  => red <=  (M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)));
104                     when 1  => red <=  (M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)) & '0');
105                     when 2  => red <=  (M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)) & '0' & '0');
106                     when 3  => red <=  (M(conv_integer(rom_pix)) & '0' & '0' & '0');
107                     when others => red <= (others => '0');
108                     end case;
109
110                     case conv_integer(BP_G) is
111                     when 0  => green <=  (M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)));
112                     when 1  => green <=  (M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)) & '0');
113                     when 2  => green <=  (M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)) & '0' & '0');
114                     when 3  => green <=  (M(conv_integer(rom_pix)) & '0' & '0' & '0');
115                     when others => green <= (others => '0');
116                     end case;
117
118                     case conv_integer(BP_B) is
119                     when 0  => blue <=  (M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)));
120                     when 1  => blue <=  (M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)) & '0');
121                     when 2  => blue <=  (M(conv_integer(rom_pix)) & M(conv_integer(rom_pix)) & '0' & '0');
122                     when 3  => blue <=  (M(conv_integer(rom_pix)) & '0' & '0' & '0');
123                     when others => blue <= (others => '0');
124                     end case;
125                 end if;
126
127             end if;
128            end process;
129
130 end Behavioral;
131
```

VGA_VHDL

```
25  use IEEE.STD_LOGIC_UNSIGNED.ALL;
26  use IEEE.STD_LOGIC_ARITH.ALL;
27
28  -- Uncomment the following library declaration if usi
29  -- arithmetic functions with Signed or Unsigned value
30  --use IEEE.NUMERIC_STD.ALL;
31
32  -- Uncomment the following library declaration if ins
33  -- any Xilinx leaf cells in this code.
34  --library UNISIM;
35  --use UNISIM.VComponents.all;
36
37  entity VGA_VHDL is
38      -- Note these numbers are different from a resolu
39      generic(
40
41              H_SP    :positive:= 128;   -- Horizontal Sy
42              H_BP    :positive:= 16;    -- Horizontal Ba
43              H_FP    :positive:= 16;    -- Horizontal Fr
44              H_Video :positive:= 640;   -- Horizontal Vi
45
46              hpixels :positive:= 800;    -- Value of pix
47
48              hbp     :positive:= 144;    -- Horizontal B
49              hfp     :positive:= 784;    -- Horizontal f
50
51
52
53
54
55              V_SP    :positive:=  2;     -- Virtical Syn
56              V_BP    :positive:= 29;     -- Virtical Bac
57              V_FP    :positive:= 10;     -- Virtical Fro
58              V_Video:positive:= 480;     -- Virtical Vid
59
60              vlines  :positive:= 521;    -- Number of ho
61              vbp     :positive:= 31;     -- Vertical bac
62              vfp     :positive:= 511     -- Vertical fro
63            );
64      Port ( clk   : in  STD_LOGIC;
65             clr   : in  STD_LOGIC;
66             hsync : out STD_LOGIC;
67             vsync : out STD_LOGIC;
68             hc    : out STD_LOGIC_VECTOR (9 downto 0);
```

```
84  CHSS:process(clk, clr)
85       begin
86          if (clr = '1') then
87              hc_reg <= (others =>'0');
88              vsenable <= '0';
89          elsif(rising_edge(clk)) then
90             if (hc_reg = hpixels -1) then
91                 hc_reg <= (others =>'0');
92                 vsenable <= '1';
93             else
94                 hc_reg <= hc_reg + 1;
95                 vsenable <= '0';
96             end if;
97          end if;
98       end process;
99
100 -- Generate hsync pulse (Horizontal Sync Pulse is lo
101
102 GHSS:process(clk,clr)
103      begin
104         if (clr = '1') then
105             hsync <= '0';
106         elsif(rising_edge(clk)) then
107             if (hc_reg <H_SP) then
108                 hsync <= '0';
109             else
110                hsync <= '1';
111             end if;
112         end if;
113      end process;
114
115 -- Counter for the Vertical sync signal
116
117 CVSS:process(clk,clr)
118      begin
119         if (clr = '1') then
120             vc_reg <= (others =>'0');
121         elsif(rising_edge(clk)) then
122             if (vsenable = '1') then
123                 if (vc_reg = vlines -1) then
124                     vc_reg <= (others =>'0');
125                 else
```

```
            if (vsenable = '1') then
                if (vc_reg = vlines -1) then
                    vc_reg <= (others =>'0');
                else
                    vc_reg <= vc_reg + 1;
                end if;
            end if;
        end if;
     end process;

-- Generate vsync pulse
-- Vertical Sync Pulse is low when vc is 0 -1
GSYNCV:process(clk,clr)
       begin
        if (clr = '1') then
            vsync <= '1';
        elsif(rising_edge(clk)) then
            if (vc_reg <V_SP) then
                vsync <= '0';
            else
                vsync <= '1';
            end if;
        end if;
       end process;

-- Enable video out when within the proches

ENV:process(clk,clr)
    begin
        if (clr = '1') then
            vidon <= '0';
        elsif(rising_edge(clk)) then
            if ((hc_reg < hfp) and (hc_reg > hbp)) and ((vc_reg <vfp) and (vc_reg > vbp)) then
                   vidon <= '1';
            else
                   vidon <= '0';
            end if;
        end if;
    end process;

    vc <= vc_reg;
    hc <= hc_reg;
end Behavioral;
```

PROM_IMG

```
entity PROM_IMG is
    generic(DEPTH     :positive:= 16;
            DATA_SIZE:positive:= 32
           );
    Port    ( addr    : in  STD_LOGIC_VECTOR (integer(ceil(log2(real(DEPTH))))-1 downto 0);
              PROM_OP : out STD_LOGIC_VECTOR (DATA_SIZE-1 downto 0)
           );
end PROM_IMG;

architecture Behavioral of PROM_IMG is

type mem_type is array (0 to (2**addr'length)-1) of std_logic_vector(DATA_SIZE-1 downto 0);
signal mem: mem_type:= (
                        "01111110000001100000110100000010",
                        "01000001000001100000110100000010",
                        "01000000100001010001010100000010",
                        "01000000010001010001010100000010",
                        "01000000001001010001010100000010",
                        "01000000000101001010010100000010",
                        "01000000000101001010010100000010",
                        "01000000000101001010010111111110",
                        "01000000000101000100010100000010",
                        "01000000000101000100010100000010",
                        "01000000000101000100010100000010",
                        "01000000000101000000010100000010",
                        "01000000001001000000010100000010",
                        "01000001000001000000010100000010",
                        "01000010000010000000010100000010",
                        "01111110000001000000010100000010"
                        );

begin

PROM_OP <= mem(conv_integer(addr));


end Behavioral;
```

Analysis:
Although fairly straightforward, this lab did take some tweaking. Mainly, converting the positional switch inputs to UART and instead having the switches control the colors were the only hassle. Tweaking the switches to control the colors did not prove to be much of an issue, however, having the FPGA intake UART commands and convert them to positional data took most of the mantime. The original FIFO had to be removed as it no longer served a purpose, and our team moved from Tera Term, our original serial communication program on our computers, to Real Term, a program capable of sending binary and hex literals rather than pure ASCII. Through simple trial and error, and adjustments where needed, we successfully created the final code.

Conclusion:
In this lab, we mainly focused on the conversation between the PC, the FPGA, and its outputted display. Using the provided code, we were able to modify the data through our board's VGA pin to update a display given inputs of both on-board switches, and external UART commands. We successfully created an efficient design that fulfills all the instructed criteria.