



**Julie Kim**  
**04/16/2021**

**CECS 561 – Hardware/Software Codesign**  
**Spring 2020**

***Basic System Report***  
***(XADC and VGA)***

The purpose of the project is to implement a new IP in Vivado tool design. The XADC IP takes in analog input and converts the analog value to digital value, the digital value is used to control the brightness of the onboard led. In reverse, the VGA Pmod takes in digital value and output the analog signal as colors, red, green and blue in the VGA monitor.

## Introduction:

XADC is an on-chip hardware system located in PS. The chip has eight dedicated pins that takes in analog input. The analog input voltage is between 3.3 volt and between 1 volt to 0 volt. The input is then converted to digital value. The digital binary value is used to calculate the brightness of onboard led. The switches on the board turn on or off the leds; VGA takes in digital value from 18 programmable logic pins, convert it to analog signal and display the colors on a VGA monitor

The advanced feature is to utilize the raw digital output value as a controller to led PWM in real time. A potentiometer increases or decreases the voltage output to the onboard XADCps, this xadc\_custom IP then output value to control the brightness of leds in real time. The VGA animation is the last advance feature planned. The VGA is another new custom IP that takes in digital value from the XADC and output color, text or picture on a VGA monitor. Digital value that is being converted from XADC IP is use as a control the VGA animation system.

## Operation:

Fist build a voltage divider circuit. The circuit uses a chain of six 1 Kohm resistors in series with two parallel 47 Kohm tied to the 3.3 v. This circuit creates two part of voltages, the first part is 3.3v. The second part is between 1 and 0 volt; these are the next five nodes of the six 1Kohm resistors in series. The circuit source power from the board as 3.3 volt. The voltage drop from 3.3v down immediately to 1volt after the current passes through the two parallel 47 Kohm . Four jumpers are used to connect to the input pins 1 to 4. Pin 4 is chosen to take input of 3.3v from the circuit. The other 3 to 1 are chosen to take input between 1 to 0 in descending order. The output brightness of the leds matches the descending order from pin 4 to pin1.

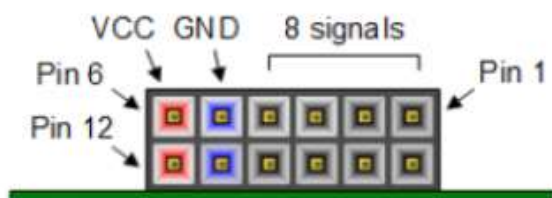
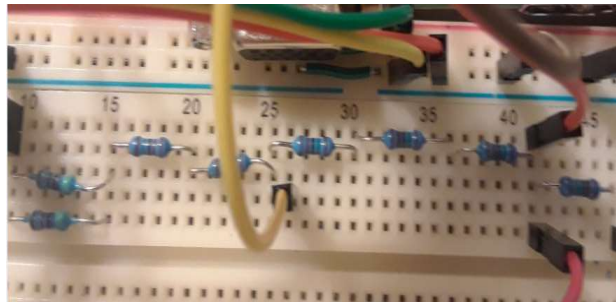


Figure 16. Pmod diagram.



Pin-5 is connected to the ground, pin-6 is the source 3.3v input. The bottom row, pin-7 to pin-12 are all connected to the ground.

To implement the program is simple after the circuit has been built. Connect the four inputs jumper wire in order, descending from pin 4 to pin 1 to the onboard XADC port. Program the board with bit stream, and export to SDK. At the start of the program, turn off all the switches to allow the current voltage value to be read to the XADC, then turn all the switches back on. If any jumper is not being moved, the leds output brightness is based on the current voltage input. The switches turns on or off the leds.

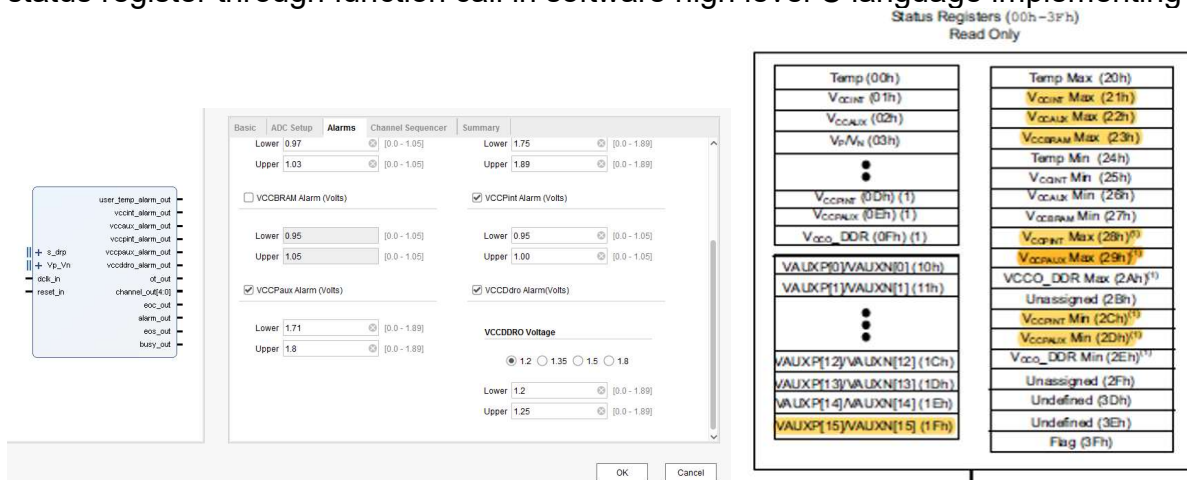
There is no circuit to build when implement the basic Pmod VGA. Very simple by connecting the VGA connector to the Zybo-Z7 JC and JD Pmod port to the VGA monitor. Only use the VGA monitor with

1080p. If the resolution is different, some modification needed in the programmable logic to match the project resolution with the resolution of the monitor. Connect the board to the computer via MicroUSB cable, program the board. To program the board, open the project in Vivado, Open Hardware Manager, Open target, select Auto connect from the drop down menu, Program device (to program the board), and click on Program (the Bitstream of the project is downloaded into the board). The VGA monitor should display animated ball, rainbow color stride and black-white stride.

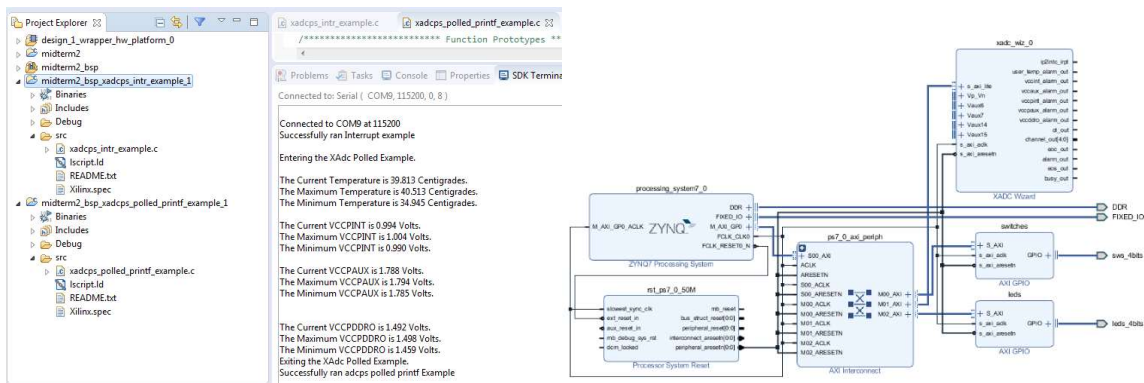
## Theory:

XADC is a 10-bit, 200-ksps analog to digital convertor reside in PS of Zynq 7000. It monitors the onboard supply voltages and temperature. It has 16 auxiliary differential analog-input pairs. XADCps can take in external analog input through the16 auxiliary differential analog-input pairs. It then converts the analog signal into digital signal and output the binary digital value stored in status registers.

There are two ways to access the status register is by JTAG port and Dynamic Reconfiguration Port. In the Vivado design tool, there is a ready to use the xadc\_wizard IP. The xadc\_wizard contains all the configuration of the on board xadc\_ps. It also can be configured to take in external input voltage through the any of the 16 auxiliary differential analog-input pairs (e.g.Vaux0, Vaux1,...Vaux14, Vaux15). The converted data is placed in the status register. The digital output value is read from the status register through function call in software high level C language implementing in SDK.



Below is the example output of using the Vivado provide xadc wizard implementing reading analog inputs, onboard temperature and onboard supply voltage. The Vivado provided xadc wizard is connected to the AXI interconnect in order to communicate with the onboard XADC ps in the zynq processor. The output voltage and the temperature values are read from software platform in SDK, where function calls are used to read and write to register to get the inputs or output values.



Another way of accessing the status register is through the Dynamic Reconfiguration Port (DRP) to read the raw digital value. DRP is the interface between onboard XADC ps and the FPGA logic in PL. DRP port (named as xadc\_wizard, which use DRP) is an HDL function in FPGA that has the instantiation of XADC ps in the function module. The XADC ps instant configures and initialize all the used registers (e.g. control registers are initialized in the XADC ps instant)

In the Basic XADC system, DRP port (and HDL in Verilog function module) is used to communicate between the onboard XADC ps and the FPGA logic(e.g. leds and switches) in PL to access the status register.

The converted digital value result is output to the leds. The led is brighter with its high digital value than the led with its low digital value. PWM is used to control the brightness of the led. The HDL Verilog code hardcodes the ceiling value of PWM as 4070 in decimal when it rolls back to 0. The PWM is counting up from 0. It counts up 1 every clock cycle. PWM value is used to compare against the converted digital value. As long as PWM value is less then converted digital value output, the led has value of 1 (the led is on). The led value is 0 when the PWM is greater than the digital value. The period of time that the PWM has value staying less than the digital value determine the length of time where led value is 1(a.k.a. led brightness). The value of the dipswitch is saved in the slave register, it can then be read via function call, and pass the value to the mWrite function to write the value to slave register 12 to turn on the leds.

VGA Pmod outputs the color display on a VGA monitor via 18 programmable logic output pins from PL to create an analog VGA output port. There are 16-bit of color, 5-bit red, 6-bit green, 5-bit blue and 2-bit standard sync signals called HS and VS, where HS is Horizontal Sync, and VS is Vertical Sync.

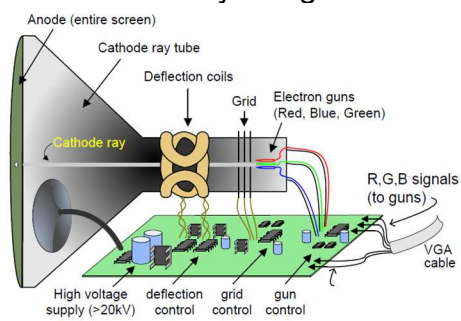


Figure 9. Color CRT display.

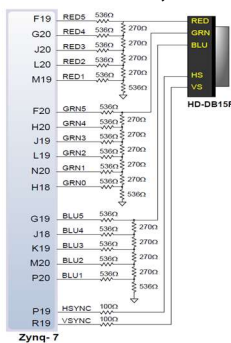
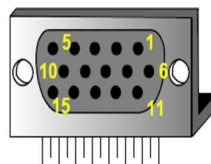


Figure 8. ZYNQ VGA circuit.



Pin 1: Red  
Pin 2: Grn  
Pin 3: Blue  
Pin 4: HS  
Pin 5: GND  
Pin 6: Red GND  
Pin 7: Grn GND  
Pin 8: Blu GND  
Pin 9: VS  
Pin 10: Sync GND

The VGA circuit is a R-2R resistor ladder that convert the digital value from the 18 programmable logic output pins to 32 and 64 analog signal levels red, blue, and green. The 16-bit color can create 65,536 (32x32x64) different colors on the display screen. The analog video color signals ranges from 0V(fully off) to 0.7V(fully on). A video controller circuit is created via the programmable logic, the



Verilog HDL code, in the PL, where the animation of object as well as colors brightness can be configured and controlled.

The basic element to produce the color display on the LCD screen is electron, produced by the Electron guns (figure above). The current sent to the electron guns can be increased or decreased to control the brightness of the display. The electron current is controlled by the timing frequency configured via the programmable logic in PL. The two standard signal to control VGA system timing is HS(Horizontal sync) and VS(Vertical sync). The configuration of HS and VS needs to match the desired screen resolution of a VGA monitor.

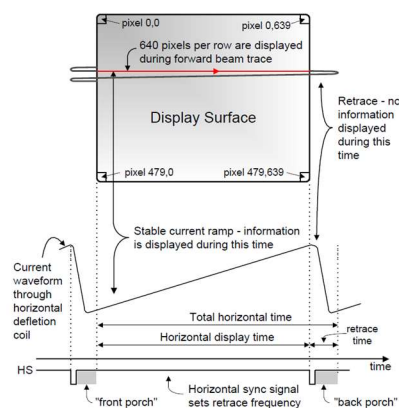


Figure 10. VGA horizontal synchronization.

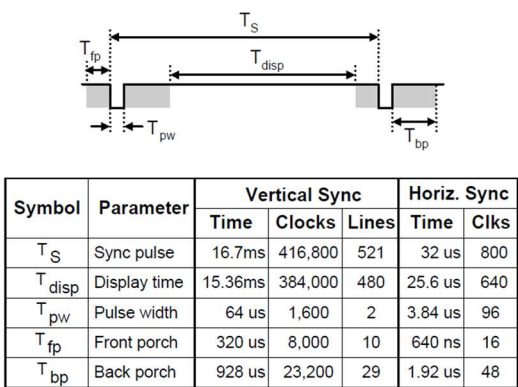


Figure 11. Signal timings for a 640-pixel by 480 row display using a 25MHz pixel clock and 60Hz vertical refresh.

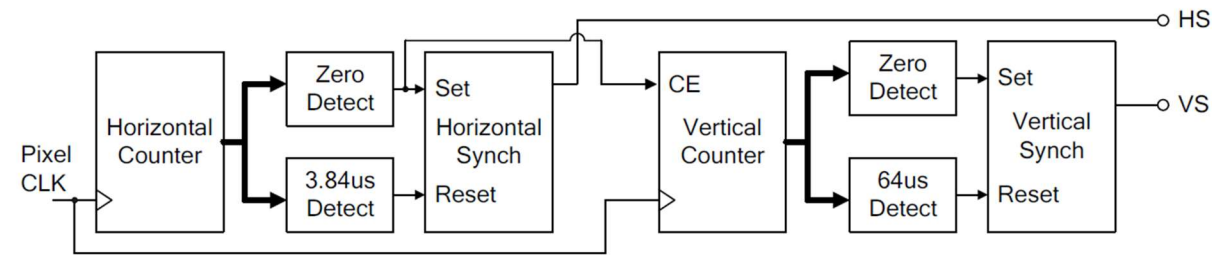


Figure 12. VGA display controller block diagram.

The VGA controller circuit logic in the PL generates the HS and VS timing signals and coordinate the delivery of video data based on the pixel clock. The pixel clock is either faster than the Zynq-process clock or slower depending on the desired display. The pixel clock defines the time available to display one pixel of information.

The HS decides the time to display video data horizontally, while the VS decides the time to refresh the screen by starting the next horizontal row when one horizontal row is finished. The VS signal defines the “refresh” frequency of the display, or the frequency at which all information on the display is redrawn. The refresh frequency is a function of the electron beam intensity, related to the electron current.

The timing diagram in figure 12 describe how the HS and VS signal is generated in the PL programmable logic. The horizontal-sync counter is driven by pixel clock to generate HS signal timing, used to locate pixel location on a given row. The output of a vertical-sync counter, incremented with each HS pulse, is used to generate VS signal timings, used to locate any given row.

In short, the timing signal HS and VS configure the intensity of the electron beam that display different color signal and animated velocity on a VGA monitor.

## Hardware Source Code:

In the Vivado PS block designs, there is zynq 700 processor, and in the PL there are AXI GPIO ip and xadc\_custom\_ip. In xadc ip has four files: xadc\_ip\_v1.v file, xadc\_ip\_v1\_0S\_AXI.v file, user\_logic.v file, and xadc\_wiz\_0.v file(In this wizard module, there is an instantiation of XADC ps, where all the controlled registers are initialized, all the I/O ports are connected based on the FPGA design, and the DRP port is used to access the status registers of the onboard XADC ps). The first file is the top level file, it has the instantiation of the S\_AXI.v file. User logic file is instantiated in the S\_AXI file. The xadc\_user\_logic.v file has the instantiation of the xadc\_wiz\_0.v file which provide access to status register through the Dynamic Reconfiguration Port (DRP) and the configuration of all status register, read registers, write registers and control registers to perform the analog to digital conversion.

This hardware design, DRP port is the interface between processor and user logic FPGA design. The digital converted output value in the status register is read through DRP port in the hardware design not in SDK.

Below are snippet code of important files that make the basic system to work.

xadc\_ip\_v1.v→ top level of the custom IP

```
4 module xadc_ip_v1_0 #
5 (
6     // Users to add parameters here
7
8     // User parameters ends
9     // Do not modify the parameters beyond this line
10
11     // Parameters of Axi Slave Bus Interface S_AXI
12     parameter integer C_S_AXI_DATA_WIDTH    = 32,
13     parameter integer C_S_AXI_ADDR_WIDTH    = 4
14 )
15 (
16     // Users to add ports here
17     output [7:0] data_out,
18     output [3:0] led,
19     input [3:0] xa_n,
20     input [3:0] xa_p,
21     // User ports ends
22     // Do not modify the ports beyond this line
23 )
24 // Instantiation of Axi Bus Interface S_AXI
25 xadc_ip_v1_0_S_AXI # (
26     .C_S_AXI_DATA_WIDTH(C_S_AXI_DATA_WIDTH),
27     .C_S_AXI_ADDR_WIDTH(C_S_AXI_ADDR_WIDTH)
28 ) xadc_ip_v1_0_S_AXI_inst (
29     .data_out(data_out),
30     .led(led),
31     .xa_n(xa_n),
32     .xa_p(xa_p),
33     .S_AXI_ACLK(s_axi_aclk),
34     .S_AXI_ARESETN(s_axi_aresetn),
35     .S_AXI_AWADDR(s_axi_awaddr),
36     .S_AXI_AWPROT(s_axi_awprot),
37     .S_AXI_AWVALID(s_axi_awvalid),
38     .S_AXI_AWREADY(s_axi_awready),
39     .S_AXI_WDATA(s_axi_wdata),
40     .S_AXI_WSTRB(s_axi_wstrb),
41     .S_AXI_WVALID(s_axi_wvalid),
42     .S_AXI_WREADY(s_axi_wready),
43     .S_AXI_BRESP(s_axi_bresp),
44     .S_AXI_BVALID(s_axi_bvalid),
45     .S_AXI_BREADY(s_axi_bready),
46     .S_AXI_ARADDR(s_axi_araddr),
47     .S_AXI_ARPROT(s_axi_arprot),
48     .S_AXI_ARVALID(s_axi_arvalid),
49     .S_AXI_ARREADY(s_axi_arready),
50     .S_AXI_RDATA(s_axi_rdata),
51     .S_AXI_RRESP(s_axi_rresp),
52     .S_AXI_RVALID(s_axi_rvalid),
53     .S_AXI_RREADY(s_axi_rready)
54 );
```

## xadc\_ip\_v1.0\_S\_AXI.v

```

391     else
392     begin
393         // When there is a valid read address (S_AXI_ARVALID) with
394         // acceptance of read address by the slave (axi_arready),
395         // output the read data
396         if (slv_reg_rden)
397         begin
398             axi_rdata <= reg_data_out;    // register read data
399         end
400     end
401 end
402
403 // Add user logic here
404 xadc_user_logic xadc_inst(
405     .S_AXI_ACLK(S_AXI_ACLK),
406     .slv_reg_wren(slv_reg_wren),
407     .slv_reg_rden(slv_reg_rden),
408     .axi_awaddr(axi_awaddr[C_S_AXI_ADDR_WIDTH-1:ADDR_LSB]),
409     .axi_araddr(axi_araddr[C_S_AXI_ADDR_WIDTH-1:ADDR_LSB]),
410     .S_AXI_WDATA(S_AXI_WDATA),
411     .S_AXI_ARESETN(S_AXI_ARESETN),
412     .data_out(data_out),
413     .led(led),
414     .xa_n(xa_n),
415     .xa_p(xa_p)
416 );

```

xadc\_user\_logic\_read.v → has the instantiation of the xadc\_wizard using DRP port as interface between processor and FPGA logic

```

22 module xadc_user_logic(
23     input S_AXI_ACLK,
24     input slv_reg_wren,
25     input slv_reg_rden,
26     input [2:0] axi_awaddr,
27     input [2:0] axi_araddr,
28     input [31:0] S_AXI_WDATA,
29     input S_AXI_ARESETN,
30
31     output [7:0] data_out,
32     output wire [3:0] led,
33     input [3:0] xa_n,
34     input [3:0] xa_p
35 );
36
37 //XADC signals
38 wire enable; //enable into the xadc to continuously c
39 reg [6:0] Address_in = 7'h14; //Address of register in XADC drp corre:
40 wire ready; //XADC port that declares when data is
41 wire [15:0] data; //XADC data
42 reg [15:0] data0, data1, data2, data3;
43 wire [11:0] shifted_data0, shifted_data1, shifted_data2, shifted_data3;
44 wire [4:0] channel_out;
45 reg [1:0] sel;
46 reg [3:0] sw_reg;

```

```

////////////////////////////////////
//XADC Instantiation
////////////////////////////////////

xadc_wiz_0 xadc_wiz_instance (
    .daddr_in      (Address_in),
    .dclk_in       (S_AXI_ACLK),
    .den_in        (enable & {sw_reg}),
    .di_in         (0),
    .dwe_in        (0),
    .busy_out      (),
    .vauxp15       (xa_p[2]),
    .vauxn15       (xa_n[2]),
    .vauxp14       (xa_p[0]),
    .vauxn14       (xa_n[0]),
    .vauxp7        (xa_p[1]),
    .vauxn7        (xa_n[1]),
    .vauxp6        (xa_p[3]),
    .vauxn6        (xa_n[3]),
    .do_out        (data),
    .vp_in         (),
    .vn_in         (),
    .eoc_out       (enable),
    .channel_out   (channel_out),
    .drdy_out      (ready)
);

```

```

117 //////////////////////////////////////////////////
118 //LED PWM
119 //////////////////////////////////////////////////
120
121 integer pwm_end = 4070;
122 //filter out tiny noisy part of signal to achieve zero at ground
123 assign shifted_data0 = (data0 >> 4) & 12'hff0;
124 assign shifted_data1 = (data1 >> 4) & 12'hff0;
125 assign shifted_data2 = (data2 >> 4) & 12'hff0;
126 assign shifted_data3 = (data3 >> 4) & 12'hff0;
127
128 integer pwm_count = 0;
129
130 //Pwm the data to show the voltage level
131 always @(posedge(S_AXI_ACLK))begin
132     if(pwm_count < pwm_end)begin
133         pwm_count = pwm_count+1;
134     end
135     else begin
136         pwm_count=0;
137     end
138 end
139 //leds are active high
140 assign led[0] = (sw_reg[0] == 1'b0) ? 1'b0 : (pwm_count < shifted_data0 ? 1'b1 : 1'b0);
141 assign led[1] = (sw_reg[1] == 1'b0) ? 1'b0 : (pwm_count < shifted_data1 ? 1'b1 : 1'b0);
142 assign led[2] = (sw_reg[2] == 1'b0) ? 1'b0 : (pwm_count < shifted_data2 ? 1'b1 : 1'b0);
143 assign led[3] = (sw_reg[3] == 1'b0) ? 1'b0 : (pwm_count < shifted_data3 ? 1'b1 : 1'b0);
144
145 endmodule

```

There is no custom ip created for the VGA Pmod. There is no block diagram created for the implementation of the basic VGA system. There is only HDL logic bitstream generated and download to the board to turn on the display.

VGA Verilog code snippet as follow:

```

entity top is
    Port (
        CLK_I : in  STD_LOGIC;
        VGA_HS_O : out STD_LOGIC;
        VGA_VS_O : out STD_LOGIC;
        VGA_R : out  STD_LOGIC_VECTOR (3 downto 0);
        VGA_B : out  STD_LOGIC_VECTOR (3 downto 0);
        VGA_G : out  STD_LOGIC_VECTOR (3 downto 0));
end top;

Architecture Behavioral of top is
    component clk_wiz_0
    port
        (-- Clock in ports
        CLK_IN1 : in  std_logic;
        -- Clock out ports
        CLK_OUT1 : out std_logic
        );
    end component;

    --Sync Generation constants

```



```

--***1920x1080@60Hz***-- Requires 148.5 MHz pxl_clk
constant FRAME_WIDTH : natural := 1920;
constant FRAME_HEIGHT : natural := 1080;

constant H_FP : natural := 88; --H front porch width (pixels)
constant H_PW : natural := 44; --H sync pulse width (pixels)
constant H_MAX : natural := 2200; --H total period (pixels)

constant V_FP : natural := 4; --V front porch width (lines)
constant V_PW : natural := 5; --V sync pulse width (lines)
constant V_MAX : natural := 1125; --V total period (lines)

constant H_POL : std_logic := '1';
constant V_POL : std_logic := '1';

--Moving Box constants
constant BOX_WIDTH : natural := 8;
constant BOX_CLK_DIV : natural := 1000000; --MAX=(2^25 - 1)

constant BOX_X_MAX : natural := (512 - BOX_WIDTH);
constant BOX_Y_MAX : natural := (FRAME_HEIGHT - BOX_WIDTH);

constant BOX_X_MIN : natural := 0;
constant BOX_Y_MIN : natural := 256;

constant BOX_X_INIT : std_logic_vector(11 downto 0) := x"000";
constant BOX_Y_INIT : std_logic_vector(11 downto 0) := x"190"; --400

signal pxl_clk : std_logic;
signal active : std_logic;

signal h_cntr_reg : std_logic_vector(11 downto 0) := (others => '0');
signal v_cntr_reg : std_logic_vector(11 downto 0) := (others => '0');

signal h_sync_reg : std_logic := not(H_POL);
signal v_sync_reg : std_logic := not(V_POL);

signal h_sync_dly_reg : std_logic := not(H_POL);
signal v_sync_dly_reg : std_logic := not(V_POL);

signal vga_red_reg : std_logic_vector(3 downto 0) := (others => '0');
signal vga_green_reg : std_logic_vector(3 downto 0) := (others => '0');
signal vga_blue_reg : std_logic_vector(3 downto 0) := (others => '0');

signal vga_red : std_logic_vector(3 downto 0);
signal vga_green : std_logic_vector(3 downto 0);
signal vga_blue : std_logic_vector(3 downto 0);

signal box_x_reg : std_logic_vector(11 downto 0) := BOX_X_INIT;
signal box_x_dir : std_logic := '1';
signal box_y_reg : std_logic_vector(11 downto 0) := BOX_Y_INIT;
signal box_y_dir : std_logic := '1';
signal box_cntr_reg : std_logic_vector(24 downto 0) := (others => '0');

signal update_box : std_logic;
signal pixel_in_box : std_logic;

```

```

begin

clk_div_inst : clk_wiz_0
port map
  (-- Clock in ports
  CLK_IN1 => CLK_I,
  -- Clock out ports
  CLK_OUT1 => pxl_clk);

-----
TEST PATTERN LOGIC
-----

vga_red <= h_cntr_reg(5 downto 2) when (active = '1' and ((h_cntr_reg < 512 and v_cntr_reg < 256) and h_cntr_reg(8) = '1')) else
  (others=>'1') when (active = '1' and ((h_cntr_reg < 512 and not(v_cntr_reg < 256)) and not(pixel_in_box = '1')) else
  (others=>'1') when (active = '1' and ((not(h_cntr_reg < 512) and (v_cntr_reg(8) = '1' and h_cntr_reg(3) = '1')) or
  (not(h_cntr_reg < 512) and (v_cntr_reg(8) = '0' and v_cntr_reg(3) = '1')))) else
  (others=>'0');

vga_blue <= h_cntr_reg(5 downto 2) when (active = '1' and ((h_cntr_reg < 512 and v_cntr_reg < 256) and h_cntr_reg(6) = '1')) else
  (others=>'1') when (active = '1' and ((h_cntr_reg < 512 and not(v_cntr_reg < 256)) and not(pixel_in_box = '1')) else
  (others=>'1') when (active = '1' and ((not(h_cntr_reg < 512) and (v_cntr_reg(8) = '1' and h_cntr_reg(3) = '1')) or
  (not(h_cntr_reg < 512) and (v_cntr_reg(8) = '0' and v_cntr_reg(3) = '1')))) else
  (others=>'0');

vga_green <= h_cntr_reg(5 downto 2) when (active = '1' and ((h_cntr_reg < 512 and v_cntr_reg < 256) and h_cntr_reg(7) = '1')) else
  (others=>'1') when (active = '1' and ((h_cntr_reg < 512 and not(v_cntr_reg < 256)) and not(pixel_in_box = '1')) else
  (others=>'1') when (active = '1' and ((not(h_cntr_reg < 512) and (v_cntr_reg(8) = '1' and h_cntr_reg(3) = '1')) or
  (not(h_cntr_reg < 512) and (v_cntr_reg(8) = '0' and v_cntr_reg(3) = '1')))) else
  (others=>'0');

-----
MOVING BOX LOGIC
-----

process (pxl_clk)
begin
  if (rising_edge(pxl_clk)) then
    if (update_box = '1') then
      if (box_x_dir = '1') then
        box_x_reg <= box_x_reg + 1;
      else
        box_x_reg <= box_x_reg - 1;
      end if;
      if (box_y_dir = '1') then
        box_y_reg <= box_y_reg + 1;
      else
        box_y_reg <= box_y_reg - 1;
      end if;
    end if;
  end if;
end process;

process (pxl_clk)
begin
  if (rising_edge(pxl_clk)) then
    if (update_box = '1') then
      if ((box_x_dir = '1' and (box_x_reg = BOX_X_MAX - 1)) or (box_x_dir = '0' and (box_x_reg = BOX_X_MIN + 1))) then
        box_x_dir <= not(box_x_dir);
      end if;
      if ((box_y_dir = '1' and (box_y_reg = BOX_Y_MAX - 1)) or (box_y_dir = '0' and (box_y_reg = BOX_Y_MIN + 1))) then
        box_y_dir <= not(box_y_dir);
      end if;
    end if;
  end if;
end process;

```

```

update_box <= '1' when box_cntr_reg = (BOX_CLK_DIV - 1) else
    '0';

pixel_in_box <= '1' when (((h_cntr_reg >= box_x_reg) and (h_cntr_reg < (box_x_reg + BOX_WIDTH))) and
    ((v_cntr_reg >= box_y_reg) and (v_cntr_reg < (box_y_reg + BOX_WIDTH)))) else
    '0';

-----
                SYNC GENERATION
-----

process (pxl_clk)
begin
    if (rising_edge(pxl_clk)) then
        if (h_cntr_reg = (H_MAX - 1)) then
            h_cntr_reg <= (others => '0');
        else
            h_cntr_reg <= h_cntr_reg + 1;
        end if;
    end if;
end process;

process (pxl_clk)
begin
    if (rising_edge(pxl_clk)) then
        if ((h_cntr_reg = (H_MAX - 1)) and (v_cntr_reg = (V_MAX - 1))) then
            v_cntr_reg <= (others => '0');
        elsif (h_cntr_reg = (H_MAX - 1)) then
            v_cntr_reg <= v_cntr_reg + 1;
        end if;
    end if;
end process;

process (pxl_clk)
begin
    if (rising_edge(pxl_clk)) then
        if (h_cntr_reg >= (H_FP + FRAME_WIDTH - 1)) and (h_cntr_reg < (H_FP + FRAME_WIDTH + H_PW - 1)) then
            h_sync_reg <= H_POL;
        else
            h_sync_reg <= not(H_POL);
        end if;
    end if;
end process;

process (pxl_clk)
begin
    if (rising_edge(pxl_clk)) then
        if (v_cntr_reg >= (V_FP + FRAME_HEIGHT - 1)) and (v_cntr_reg < (V_FP + FRAME_HEIGHT + V_PW - 1)) then
            v_sync_reg <= V_POL;
        else
            v_sync_reg <= not(V_POL);
        end if;
    end if;
end process;

active <= '1' when ((h_cntr_reg < FRAME_WIDTH) and (v_cntr_reg < FRAME_HEIGHT)) else
    '0';

process (pxl_clk)
begin
    if (rising_edge(pxl_clk)) then
        v_sync_dly_reg <= v_sync_reg;
        h_sync_dly_reg <= h_sync_reg;
        vga_red_reg <= vga_red;
        vga_green_reg <= vga_green;
        vga_blue_reg <= vga_blue;
    end if;
end process;

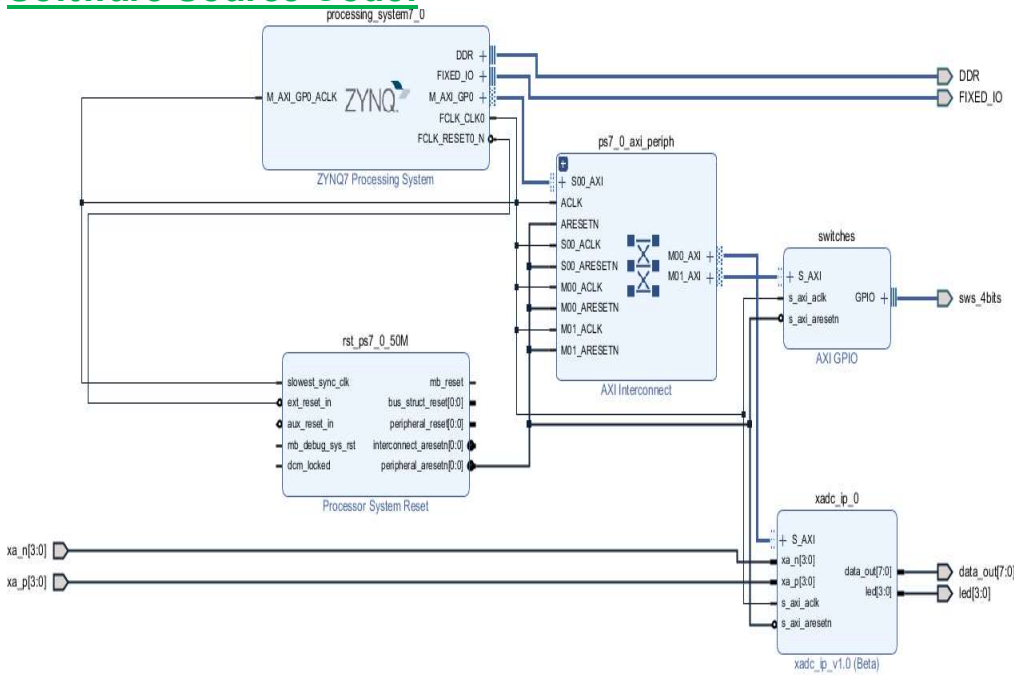
VGA_HS_O <= h_sync_dly_reg;
VGA_VS_O <= v_sync_dly_reg;
VGA_R <= vga_red_reg;
VGA_G <= vga_green_reg;
VGA_B <= vga_blue_reg;

end Behavioral;

```



## Software Source Code:



In SDK development environment, initialize the AXI GPIO dipswitches, use the dip switches value to write to the slave register of the xadc\_custom\_ip. The program loops through by keep reading the switch value to turn on or off the leds.

```

1  #include "xparameters.h"
2  #include "xgpio.h"
3  #include "xadc_ip.h"
4
5  //=====
6
7  int main (void)
8  {
9
10     XGpio dip_sw;
11     int i, dip_sw_check;
12
13     xil_printf("-- Start of the Program --\r\n");
14
15     XGpio_Initialize(&dip_sw, XPAR_SWITCHES_DEVICE_ID); // Modify this
16     XGpio_SetDataDirection(&dip_sw, 1, 0xffffffff);
17
18     while (1)
19     {
20         dip_sw_check = XGpio_DiscreteRead(&dip_sw, 1);
21         xil_printf("DIP Switch Status %x\r\n", dip_sw_check);
22
23         // output dip switches value on LED_ip device
24         XADC_IP_mWriteReg(XPAR_XADC_IP_0_S_AXI_BASEADDR, 12, dip_sw_check);
25
26         for (i=0; i<9999999; i++);
27     }
28 }

```



## Conclusion:

The basic system is considered as a successful implementation. It is fairly simple and straight forward by following instruction of the github example. The switches turn on and off the leds, the leds brightness differs according to the input voltage from the voltage divider circuit. The only challenge is the creating the block design in Vivado. A research is made to find the XADC datasheet to understand the usage of JTAG port versus the DRP port. The way to access the status register data through DRP enable user logic FPGA in PL to directly read data from status register and manipulate the digital output value in PL hardware design.

The basic system for the VAG Pmod is also simple. The HDL logic is provided and the implementation description is clear and simple. The video display with the box animation is successfully implemented and quit interesting. However, the timing of the VGA circuit does not match with the processor timing when creating the my\_vga\_ip custom ip. More work is needed to incorporate the HDL logic into the SDK software development platform to further implement the project to the next level.

Demo Video link:

XADC:

[https://csulb-my.sharepoint.com/:v:/g/personal/julie\\_kim\\_student\\_csulb\\_edu/EYFQxCjRnfhEm-p1fo7QtRYBybPfSh6a2j-4ADwl1c-kgA?e=yxK7UP](https://csulb-my.sharepoint.com/:v:/g/personal/julie_kim_student_csulb_edu/EYFQxCjRnfhEm-p1fo7QtRYBybPfSh6a2j-4ADwl1c-kgA?e=yxK7UP)

VGA

[https://csulb-my.sharepoint.com/:v:/g/personal/julie\\_kim\\_student\\_csulb\\_edu/ES2XuPdMJtpDqDI4PwPiJPcBr10PMNRjdB7yKdbJO2ZrZg?e=lf0LwV](https://csulb-my.sharepoint.com/:v:/g/personal/julie_kim_student_csulb_edu/ES2XuPdMJtpDqDI4PwPiJPcBr10PMNRjdB7yKdbJO2ZrZg?e=lf0LwV)

## References:

- [https://www.xilinx.com/support/documentation/user\\_guides/ug480\\_7Series\\_XADC.pdf](https://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf)
- [https://www.xilinx.com/support/documentation/ip\\_documentation/xadc\\_wiz/v3\\_3/pg091-xadc-wiz.pdf](https://www.xilinx.com/support/documentation/ip_documentation/xadc_wiz/v3_3/pg091-xadc-wiz.pdf)
- [file:///C:/Xilinx/SDK/2018.3/data/embeddedsw/XilinxProcessorIPLib/drivers/xadcps\\_v2\\_3/doc/html/api/index.html](file:///C:/Xilinx/SDK/2018.3/data/embeddedsw/XilinxProcessorIPLib/drivers/xadcps_v2_3/doc/html/api/index.html)
- <https://reference.digilentinc.com/learn/programmable-logic/tutorials/github-demos/start>
- [https://github.com/Digilent/Zybo-Z7-10-Pmod-VGA?\\_ga=2.93914781.1807014336.1554410949-423883057.1550577685](https://github.com/Digilent/Zybo-Z7-10-Pmod-VGA?_ga=2.93914781.1807014336.1554410949-423883057.1550577685)