Julie Kim
05/14/2021

CECS 561 – Hardware/Software Codesign
Spring 20201

Basic and Advanced 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.

I. Introduction:

Phase 1.0 Basic System Description:

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 than 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

Phase 2.0 Add on System Description:

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_custome 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.

II. Operation:

Phase 1.1 Basic System XADC Operation:

Fist build a voltage divider circuit. The circuit uses a chain of six 1 Kohm resisters 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 nods of the six 1Khom 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.

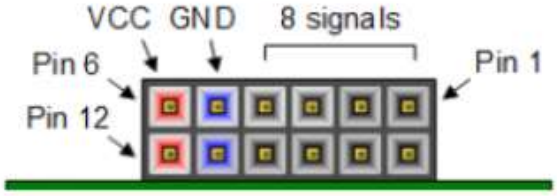
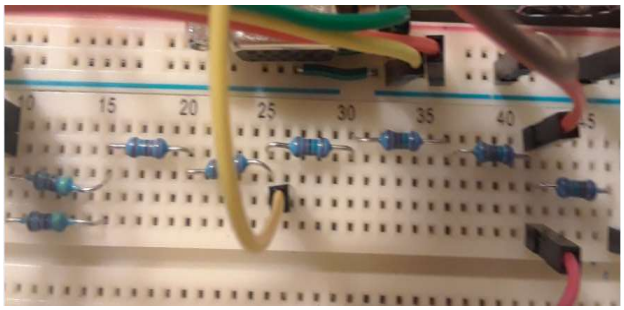


Figure 16. Pmod diagram.



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.

Phase 1.2 Basic System VGA Operation:

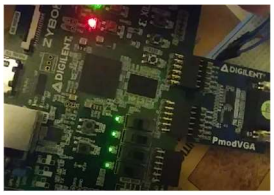
There is no circuit to build when implement the basic Pmod VGA. Very simple by connecting the VGA connector to the Zybo-Z7 JC and JD Pmod port to the VGA monitor. Only use the VGA monitor with 1080p. If the resolution is different, some modification needed in the programmable logic to match the project 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.

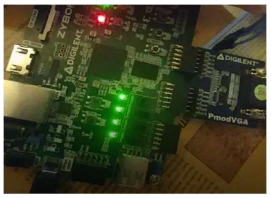
Phase 2.0 Add on Advanced Operation:

Instead building a divider circuit, a potentiometer is used as a controller of the input voltage. The two sides pin of the pot is connected to each vcc and ground. The middle pin is the input of the pin. The four wires connected to the top row of the XADC port are connected to the middle pin of the pot to take in input to the board. The top four pin of XADC port now receive the same analog voltage as the pot turn. The brightness of four leds are the same and update the brightness in real time as the pot turn (to the right increase the voltage and to the left decrease the voltage.

A Pong Game VGA custom IP is built with the input button to move the bar up and down to try to catch the ball. The 4-bits switch changes the colors of the screen. Since there are Red, Green and Blue rgb output, the RED and Blue are fixed, the only color to change is the 4-bit Green rgb. The value of the switch changes the value of Green rgb, changing the vga output monitor background colors. The speed of the ball is faster when leds are brighter, adjusted by turning the Potentiometer.







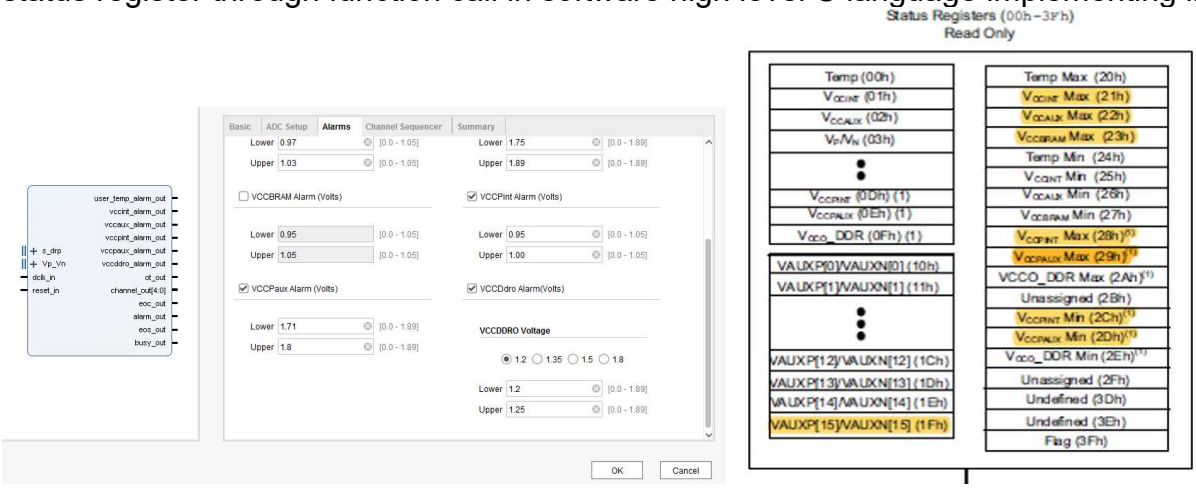
III. Theory:

Phase 1.1 Basic System XADC Theory:

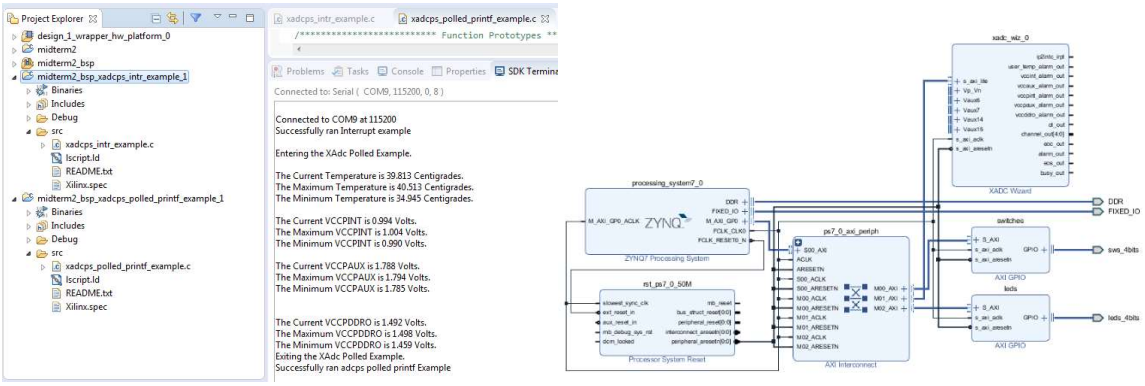
XADC is a 10-bit, 200-ksps analog to digital convertor reside in PS of Zynq 7000. It monitors the onboard supply voltages and temperature. It has 16 auxiliary differential analog-input pairs. XADCps can take in external analog input through the16 auxiliary differential analog-input pairs. It then converts 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 zyqn processor. The output voltage and the temperature values are read from software plateform 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.

Phase 1.2 Basic System VGA theory:

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.

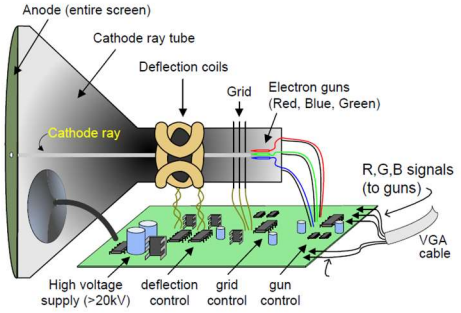


Figure 9. Color CRT display.

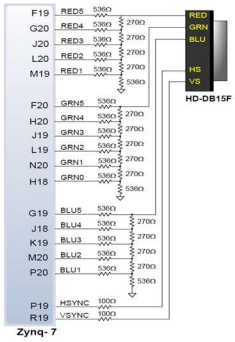
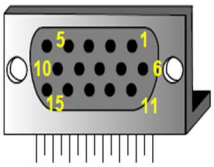


Figure 8. ZYBO VGA circuit.



Pin 1: Red Pin 5: GND
Pin 2: Grn Pin 6: Red GND
Pin 3: Blue Pin 7: Grn GND
Pin 13: HS Pin 8: Blu GND
Pin 14: 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.

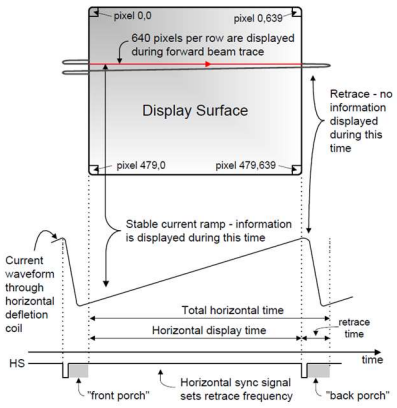
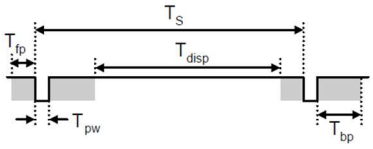


Figure 10. VGA horizontal synchronization.



Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	Front porch	320 us	8,000	10	640 ns	16
T_{bp}	Back porch	928 us	23,200	29	1.92 us	48

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

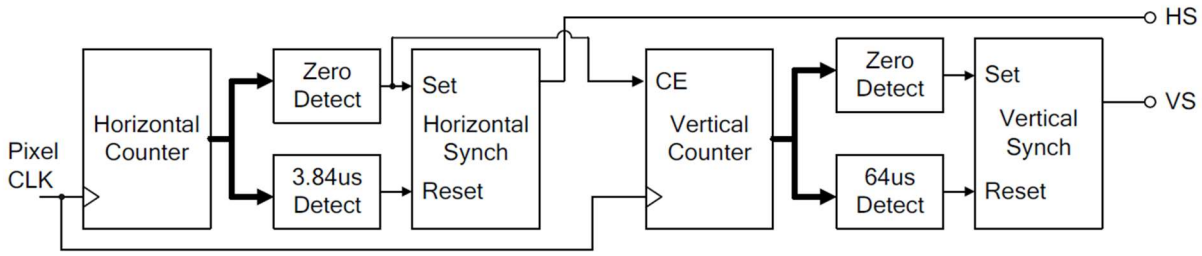


Figure 12. VGA display controller block diagram.

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 bean 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.

Phase 2.0 Advanced System Potentiometer and Pong Game theory:

The digital output from the XADC is both routed to the leds output and the VGA module. The digital output signal is 16-bits, only the msb-3-bits is sent to the speed of the ball signal. The brighter the leds, the higher the binary 16-bits value, which mean the faster the speed sent to the ball.

By using the clock and Hsyn and Vsyn signals, different colors are outputted on the screen. The ball is set to Black, the tall fixed big bar is set to Blue, the small moving tab is set to Green, and the background colors are varied by 2^8 color with rgb-Blue and rgb-Red fixed (original combination is 2^8 x 2^8 x 2^8). The colors, fixed object, and moving ball and tab are the production of clock timing signal and the digital input converted to analog output. Colorful Pong Game is generated by HDL logic with proper timing controlled.

IV. Hardware Designed:

Phase 1.1 Basic System XADC HDL Code:

In the Vivado PS block designs, there is zynq 700 processor, and in the PL there are AXI GPIO ip and xadc_custome_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

```
    module xadc_ip_v1_0 #
    (
        // Users to add parameters here

        // User parameters ends
        // Do not modify the parameters beyond this line

        // Parameters of Axi Slave Bus Interface S_AXI
        parameter integer C_S_AXI_DATA_WIDTH    = 32,
        parameter integer C_S_AXI_ADDR_WIDTH    = 4
    )
    (
        // Users to add ports here
        output [7:0] data_out,
        output [3:0] led,
        input [3:0] xa_n,
        input [3:0] xa_p,
        // User ports ends
        // Do not modify the ports beyond this line
// Instantiation of Axi Bus Interface S_AXI
    xadc_ip_v1_0_S_AXI # (
        .C_S_AXI_DATA_WIDTH(C_S_AXI_DATA_WIDTH),
        .C_S_AXI_ADDR_WIDTH(C_S_AXI_ADDR_WIDTH)
    ) xadc_ip_v1_0_S_AXI_inst (
        .data_out(data_out),
        .led(led),
        .xa_n(xa_n),
        .xa_p(xa_p),
        .S_AXI_ACLK(s_axi_aclk),
        .S_AXI_ARESETN(s_axi_aresetn),
        .S_AXI_AWADDR(s_axi_awaddr),
        .S_AXI_AWPROT(s_axi_awprot),
        .S_AXI_AWVALID(s_axi_awvalid),
        .S_AXI_AWREADY(s_axi_awready),
        .S_AXI_WDATA(s_axi_wdata),
        .S_AXI_WSTRB(s_axi_wstrb),
        .S_AXI_WVALID(s_axi_wvalid),
        .S_AXI_WREADY(s_axi_wready),
        .S_AXI_BRESP(s_axi_bresp),
        .S_AXI_BVALID(s_axi_bvalid),
        .S_AXI_BREADY(s_axi_bready),
        .S_AXI_ARADDR(s_axi_araddr),
        .S_AXI_ARPROT(s_axi_arprot),
        .S_AXI_ARVALID(s_axi_arvalid),
        .S_AXI_ARREADY(s_axi_arready),
        .S_AXI_RDATA(s_axi_rdata),
        .S_AXI_RRESP(s_axi_rresp),
        .S_AXI_RVALID(s_axi_rvalid),
        .S_AXI_RREADY(s_axi_rready)
    );
```

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_aready),
395             // output the read dada
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 continuosly q
39      reg [6:0] Address_in = 7'h14;   //Adress of register in XADC drp corres
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
```

Phase 1.2 Basic System VGA vhdl Code:

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;
```

Phase 2. Advanced XADC Verilog Code:

There is only slight modification of the hardware source code. The switches are not used to turn on or off the led. In stead the switches are used in my_vga_ip to change the color. Instead of using the switch to toggle input, all the connected wire is tight to the same voltage via the changes of the pot serve as the input and send the output digital value directly to the leds. The logic behind how the pot work is shown in the following hdl code:

xadc_user_logic.v | xadc_user_logic_read_switches.v

```
1   `timescale 1ns / 1ps
2
3   //////////////////////////////////////////////////////////////////////////////////
4   // Company: Digilent
5   // Engineer: Arthur Brown
6   //
7   // Create Date: 09/07/2017 11:30:35 AM
8   // Module Name: top
9   // Project Name: Zybo Z7 XADC Demo
10  // Target Devices: Zybo Z7 10 or 20
11  // Tool Versions: Vivado 2016.4
12  // Description: This demo instantiates an XADC Wizard block and uses it to capture da
13  //              This data is displayed as the brightness of the four onboard LEDs.
14  //
15  // Revision 0.01 - File Created
16  // Additional Comments:
17  //
18  //////////////////////////////////////////////////////////////////////////////////
19
20
21  module xadc_user_logic(
22      input clk,
23      input [7:0] ja,
24      output [3:0] led
25  );
26      reg [6:0] daddr = 0; // address of channel to be read
27      reg [1:0] ledidx = 0; // index of the led to capture data for
28
29      wire eoc; // xadc end of conversion flag
30      wire [15:0] dout; // xadc data out bus
31      wire drdy;
32
33      reg [1:0] _drdy = 0; // delayed data ready signal for edge detection
34
35      reg [7:0] data0 = 0, // stored XADC data, only the uppermost byte
36                data1 = 0,
37                data2 = 0,
38                data3 = 0;
39      reg [7:0] pwm_count; // shared pwm counter
40      reg [7:0] pwm_duty0; // duty cycles for the 4 pwm led brightness controllers
41      reg [7:0] pwm_duty1;
42      reg [7:0] pwm_duty2;
43      reg [7:0] pwm_duty3;
44
45      xadc_wiz_0 myxadc (
46          .dclk_in        (clk),
47          .den_in         (eoc), // drp enable, start a new conversion whenever the las
48          .dwe_in         (0),
49          .daddr_in       (daddr), // channel address
50          .di_in          (0),
51          .do_out         (dout), // data out
52          .drdy_out       (drdy), // data ready
53          .eoc_out        (eoc), // end of conversion
54
55          .vauxn6         (ja[7]),
56          .vauxp6         (ja[3]),
57
58          .vauxn7         (ja[5]),
59          .vauxp7         (ja[1]),
60
61          .vauxn14        (ja[4]),
62          .vauxp14        (ja[0]),
63
64          .vauxn15        (ja[6]),
65          .vauxp15        (ja[2])
66      );
```

```
    always@(posedge clk)
        _drdy <= {_drdy[0], drdy};

    always@(*)
        case (ledidx)
        0: daddr = 7'h1E;
        1: daddr = 7'h17;
        2: daddr = 7'h1F;
        3: daddr = 7'h16;
        default: daddr = 7'h1E;
        endcase

    always@(posedge clk) begin
        if (_drdy == 2'b10) begin // on negative edge
            ledidx <= ledidx + 1;
            case (ledidx)
            0: data0 <= dout[15:8];
            1: data1 <= dout[15:8];
            2: data2 <= dout[15:8];
            3: data3 <= dout[15:8];
            endcase
        end
    end

    always@(posedge clk)
        pwm_count <= pwm_count + 1;

    always@(posedge clk)
        if (pwm_count == 0) begin
            pwm_duty0 <= data0;
            pwm_duty1 <= data1;
            pwm_duty2 <= data2;
            pwm_duty3 <= data3;
        end

    assign led[0] = (pwm_count <= pwm_duty0) ? 1 : 0;
    assign led[1] = (pwm_count <= pwm_duty1) ? 1 : 0;
    assign led[2] = (pwm_count <= pwm_duty2) ? 1 : 0;
    assign led[3] = (pwm_count <= pwm_duty3) ? 1 : 0;
endmodule
```

Adjust the led index so the all led receive the same digital values

The digital output is sent directly to the led

Phase 2.1 Advanced System Pong Game Verilog Code:

```
`timescale 1 ns / 1 ps

    module my_vga_ip_v1_0 #
    (
        // Users to add parameters here

        // User parameters ends
        // Do not modify the parameters beyond this line


        // Parameters of Axi Slave Bus Interface S_AXI
        parameter integer C_S_AXI_DATA_WIDTH    = 32,
        parameter integer C_S_AXI_ADDR_WIDTH    = 4
    )
    (
        // Users to add ports here
       input  wire [ 1:0] btn,
       input  wire [3:0]  sws,
       output wire        hsync, vsync,
       output wire [3:0] VGA_R,
       output wire [3:0] VGA_G,
       output wire [3:0] VGA_B,

       input  wire [7:0]  ja,
       output wire [3:0]  led,
        // User ports ends
        // Do not modify the ports beyond this line
```

```
    );
// Instantiation of Axi Bus Interface S_AXI
    my_vga_ip_v1_0_S_AXI # (
        .C_S_AXI_DATA_WIDTH(C_S_AXI_DATA_WIDTH),
        .C_S_AXI_ADDR_WIDTH(C_S_AXI_ADDR_WIDTH)
    ) my_vga_ip_v1_0_S_AXI_inst (
        .btn(btn),
        .sws(sws),
        .hsync(hsync),
        .vsync(vsync),
        .VGA_R(VGA_R),
        .VGA_G(VGA_G),
        .VGA_B(VGA_B),
        .ja(ja),
        .led(led),
        .S_AXI_ACLK(s_axi_aclk),
        .S_AXI_ARESETN(s_axi_aresetn),
        .S_AXI_AWADDR(s_axi_awaddr),
```

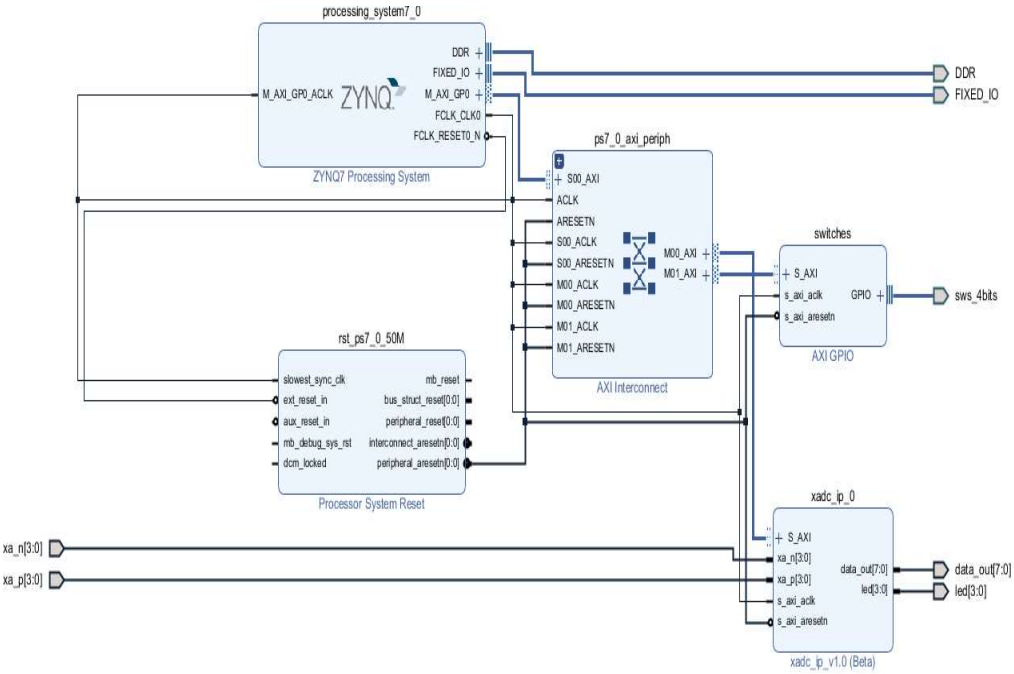
```
module my_vga_ip_v1_0_S_AXI #
(
    // Users to add parameters here

    // User parameters ends
    // Do not modify the parameters beyond this line

    // Width of S_AXI data bus
    parameter integer C_S_AXI_DATA_WIDTH    = 32,
    // Width of S_AXI address bus
    parameter integer C_S_AXI_ADDR_WIDTH    = 4
)
(
    // Users to add ports here
   input  wire [ 1:0] btn,
   input  wire [3:0]  sws,
   output wire        hsync, vsync,
   output wire [3:0] VGA_R,
   output wire [3:0] VGA_G,
   output wire [3:0] VGA_B,

   input  wire [7:0]  ja,
   output wire [3:0]  led,
    // User ports ends
    // Do not modify the ports beyond this line

    // Global Clock Signal
    input wire  S_AXI_ACLK,
    // Global Reset Signal. This Signal is Active LOW
    input wire  S_AXI_ARESETN,
```

```
407
408     // Add user logic here
409     pong_top_an U1(
410         .S_AXI_ACLK(S_AXI_ACLK),
411         .slv_reg_wren(slv_reg_wren),
412         .axi_awaddr(axi_awaddr[C_S_AXI_ADDR_WIDTH-1:ADDR_LSB]),
413         .S_AXI_WDATA(S_AXI_WDATA),
414         .S_AXI_ARESETN(S_AXI_ARESETN),
415         .btn(btn),
416         .sws(sws),
417         .hsync(hsync),
418         .vsync(vsync),
419         .VGA_R(VGA_R),
420         .VGA_G(VGA_G),
421         .VGA_B(VGA_B),
422         .ja(ja),
423         .led(led)
424     );
425     // User logic ends
426
427     endmodule
```

```
`timescale 1ns/1ps
// pong chu top animate module implemented by John Tramel
// 12/28/16
module pong_top_an (
    input S_AXI_ACLK,
    input slv_reg_wren,
    input [2:0] axi_awaddr,
    input [31:0] S_AXI_WDATA,
    input S_AXI_ARESETN,

    input  wire [ 1:0] btn,
    input  wire [3:0]  sws,
    output wire        hsync, vsync,
    output wire [3:0] VGA_R,
    output wire [3:0] VGA_G,
    output wire [3:0] VGA_B,

    input  wire [7:0]  ja,
    output wire [3:0]  led
    );
 wire [7:0]  speed;
 wire [11:0] rgb;
//////////////////////////////////////////////
// signal declaration
//////////////////////////////////////////////
 wire [9:0] pixel_x, pixel_y;
 wire       video_on, cnt3;
 reg  [2:0] rgb_reg;
 wire [2:0] rgb_next;
//////////////////////////////////////////////
// body
//////////////////////////////////////////////
vga_sync vsuut (
    .clk(S_AXI_ACLK),
    .reset(S_AXI_ARESETN),
    .hsync(hsync),
    .vsync(vsync),
    .video_on(video_on),
    .cnt3(cnt3),
    .pixel_x(pixel_x),
    .pixel_y(pixel_y));
pong_graph_animate pgauut (
    .clk(S_AXI_ACLK),
    .reset(S_AXI_ARESETN),
    .btn(btn),
    .sws(sws),
    .speed(speed),
    .video_on(video_on),
    .pix_x(pixel_x),
    .pix_y(pixel_y),
    .graph_rgb(rgb_next));

xadc_user_logic xadcuut (
     .clk(S_AXI_ACLK),
     .ja(ja),
     .led(led),
     .data_out(speed)
);
//////////////////////////////////////////////
// rgb buffer
//////////////////////////////////////////////
 always @(posedge S_AXI_ACLK, posedge S_AXI_ARESETN)
    if (S_AXI_ARESETN)
       rgb_reg <= 3'b000; else
    if (cnt3)
       rgb_reg <= rgb_next;

 assign rgb[11:8] = {4{rgb_reg[2]}};  //Nexys4
 assign rgb[ 7:4] = {4{rgb_reg[1]}};
 assign rgb[ 3:0] = {4{rgb_reg[0]}};
 assign VGA_R = (rgb_reg == 3'b110)? 4'b1111: {4{rgb_reg[2]}};
 assign VGA_G = (rgb_reg == 3'b110)? sws    : {4{rgb_reg[1]}};
 assign VGA_B = (rgb_reg == 3'b110)? 4'b0100: {4{rgb_reg[0]}};
 endmodule
```

V. Software Designed:

Phase 1.0 Basic System XADC Software Source Code:



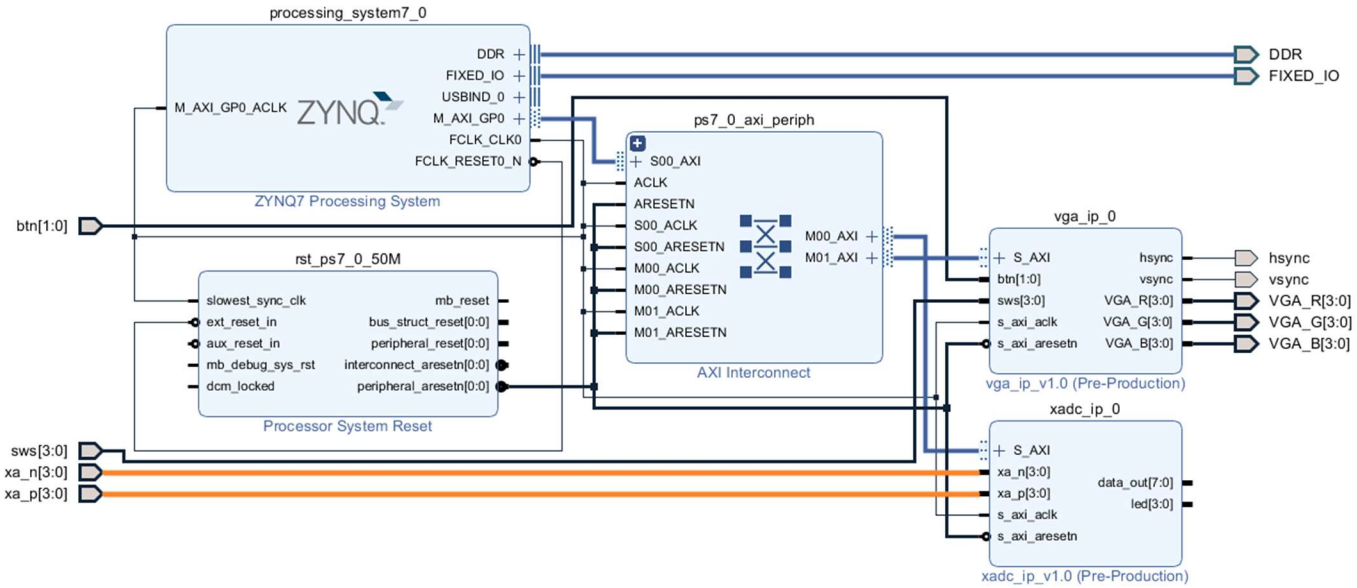
In SDK development environment, initialize the AXI GPIO dipswitches, use the dip switches value to write to the slave register of the xadc_custom_ip. The program loops through by keep reading the switch value to turn on or off the leds.

```
#include "xparameters.h"
#include "xgpio.h"
#include "xadc_ip.h"

//====================================================

int main (void)
{

   XGpio dip_sw;
   int i, dip_sw_check;

   xil_printf("-- Start of the Program --\r\n");

   XGpio_Initialize(&dip_sw, XPAR_SWITCHES_DEVICE_ID); // Modify this
   XGpio_SetDataDirection(&dip_sw, 1, 0xffffffff);

while (1)
    {
       dip_sw_check = XGpio_DiscreteRead(&dip_sw, 1);
       xil_printf("DIP Switch Status %x\r\n", dip_sw_check);

       // output dip switches value on LED_ip device
       XADC_IP_mWriteReg(XPAR_XADC_IP_0_S_AXI_BASEADDR, 12, dip_sw_check);

       for (i=0; i<9999999; i++);
    }
}
```

Phase 2.1 Advanced System XADC and VGA:

Since output signal from the XADC convertor is not connected to the VGA custom IP internally, the two custom IPs are independent. The input signal only controls the variation light of the LED, and the speed of the animated ball is set internally in VGA custom IP.

The Block diagram with two separate non-connected XADC IP and VGA IP:

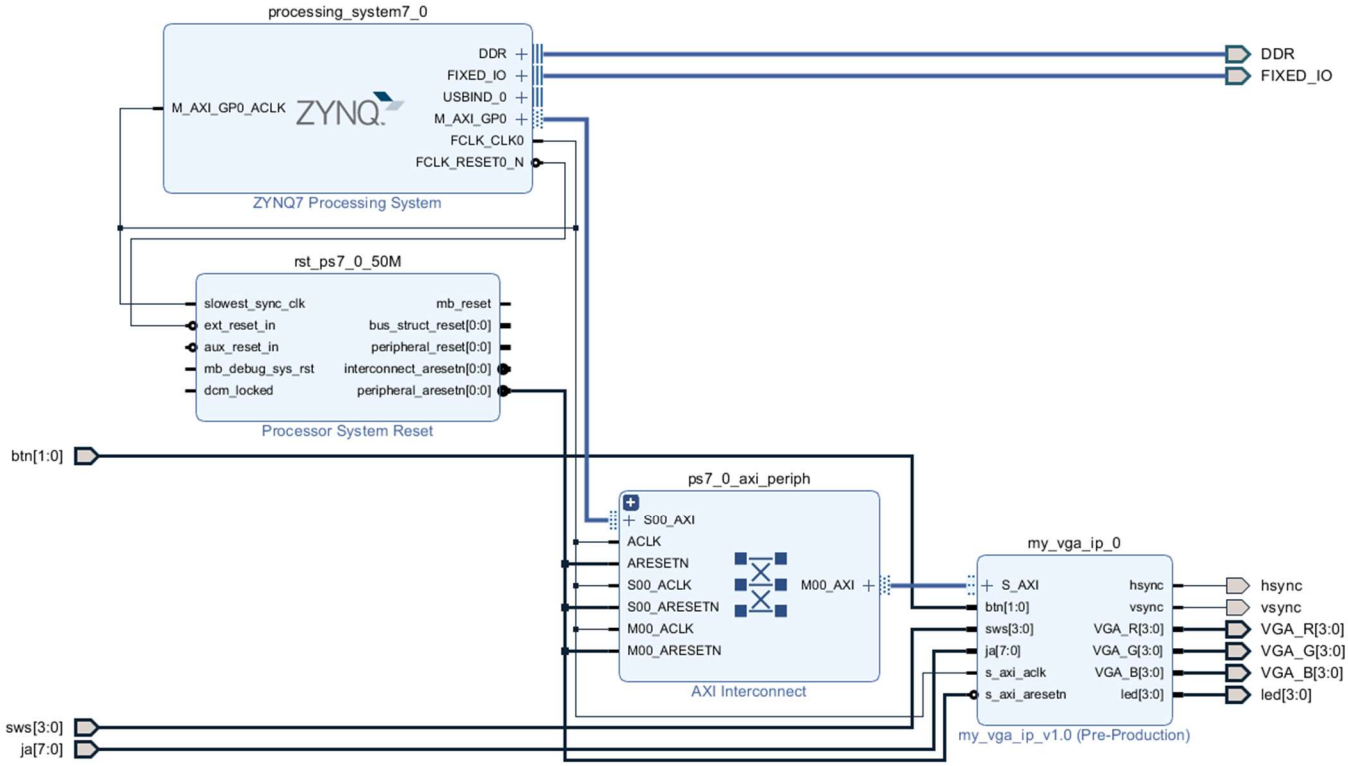


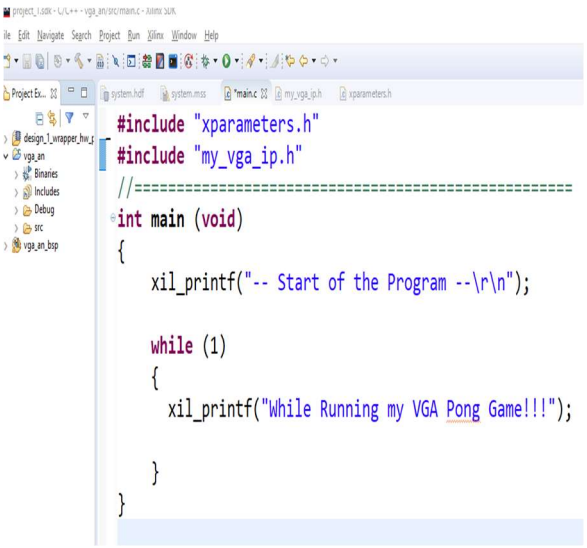
The two-orange signal is not connected to the vga_ip. And the Pong Game animation have a fixed speed is 2, BALL_V_P = 2 and BALL_V_N = -2

Phase 2.2 Advanced System Pong Game:

To control the speed of the animated ball, digital output of the XADC is directly sent to the VGA IP internally. The two custom IP are joint as one HDL module.

The Block diagram with one custom IP is the combination of XADC and VGA logic





```
#include "xparameters.h"
#include "my_vga_ip.h"
//================================================
int main (void)
{
    xil_printf("-- Start of the Program --\r\n");

    while (1)
    {
      xil_printf("While Running my VGA Pong Game!!!");

    }
}
```

Address Map for processor ps7_cortexa9_[0-1]

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
ps7_intc_dist_0	0xf8f01000	0xf8f01fff		REGISTER
ps7_scutimer_0	0xf8f00600	0xf8f0061f		REGISTER
ps7_slcr_0	0xf8000000	0xf8000fff		REGISTER
ps7_scuwdt_0	0xf8f00620	0xf8f006ff		REGISTER
ps7_l2cachec_0	0xf8f02000	0xf8f02fff		REGISTER
ps7_scuc_0	0xf8f00000	0xf8f000fc		REGISTER
ps7_pmu_0	0xf8893000	0xf8893fff		REGISTER
ps7_afi_1	0xf8009000	0xf8009fff		REGISTER
ps7_afi_0	0xf8008000	0xf8008fff		REGISTER
ps7_afi_3	0xf800b000	0xf800bfff		REGISTER
ps7_afi_2	0xf800a000	0xf800afff		REGISTER
ps7_globaltimer_0	0xf8f00200	0xf8f002ff		REGISTER
my_vga_ip	0x43c00000	0x43c0ffff	S_AXI	REGISTER
ps7_dma_s	0xf8003000	0xf8003fff		REGISTER
ps7_iop_bus_config_0	0xe0200000	0xe0200fff		REGISTER
ps7_xadc_0	0xf8007100	0xf8007120		REGISTER
ps7_ddr_0	0x00100000	0x3fffffff		MEMORY
ps7_ddrc_0	0xf8006000	0xf8006fff		REGISTER
ps7_ocmc_0	0xf800c000	0xf800cfff		REGISTER
ps7_pl310_0	0xf8f02000	0xf8f02fff		REGISTER
ps7_uart_1	0xe0001000	0xe0001fff		REGISTER
ps7_coresight_comp_0	0xf8800000	0xf88fffff		REGISTER
ps7_scugic_0	0xf8f00100	0xf8f001ff		REGISTER
ps7_dev_cfg_0	0xf8007000	0xf80070ff		REGISTER

VI. 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.

However, to use built in XADC IP is a challenge not being able to overcome. As a result, the project fails to connect two custom IPs in the SDK software platform level. There are two independent XADC and VGA custom IP. Also, another challenge is little knowledge of VHDL language, to build advanced system based on VHDL language is quite hard. Instead of continuing with the basic VGA system, a new VGA module, Pong Game written in Verilog, is implement. By proper connection of the internal signals, the project can be successfully implemented when combining XADC module with the Pong Game module to have a closed ended project.

With further examination, it is also possible to have connected the two custom IP by sending the output signal and write it to one of the slave registers. Then read value from the slave register as the input signal to the VGA IP. This input is then written to slave register of the VGA IP as an input speed to control the Pong ball. Because of time constraint, the project needs to be ended with the combination of XADC-VGA as one custom IP. The project is not hard to build; it uses all the basic knowledge from the course.

Demo Video link:

- Potentiometer changes LED: https://csulb-my.sharepoint.com/:v:/g/personal/julie_kim_student_csulb_edu/ERFVTfc862ZKkIVXGFZ07dYBWW1_KhGFhIrlVB-n3aiWSg?e=38H6Mo

- Pong Game Animation: https://csulb-my.sharepoint.com/:v:/g/personal/julie_kim_student_csulb_edu/ERkP6eHb_Y1GuxDGshmffKYBp_mjw3-HfdSbVCv_btXovQ?e=9nTEzk

- Ball Speed: https://csulb-my.sharepoint.com/:v:/g/personal/julie_kim_student_csulb_edu/ERSjpJSDOh1Jql7zSslSirMBQh3Q-Bab_MIjYXnH5Q_AOg?e=iSGxtN

- Switches change Screen Colors: https://csulb-my.sharepoint.com/:v:/g/personal/julie_kim_student_csulb_edu/ESTBAdenzndHhFp9si0P0X4BA5zs3DDfarueZrr5ctRz6A?e=Qr5AHs

References:

- https://github.com/Digilent/Zybo-Z7-10-Pmod-VGA?_ga=2.93914781.1807014336.1554410949-423883057.1550577685
- https://reference.digilentinc.com/learn/programmable-logic/tutorials/github-demos/start
- file:///C:/Xilinx/SDK/2018.3/data/embeddedsw/XilinxProcessorIPLib/drivers/xadcps_v2_3/doc/html/api/index.html
- https://www.xilinx.com/support/documentation/ip_documentation/xadc_wiz/v3_3/pg091-xadc-wiz.pdf
- https://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf
-