Julie Kim & Srija Shukla

04/16/2021

CECS 561 – Hardware/Software Codesign

Sprint 2020

Final Project 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 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

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.

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.

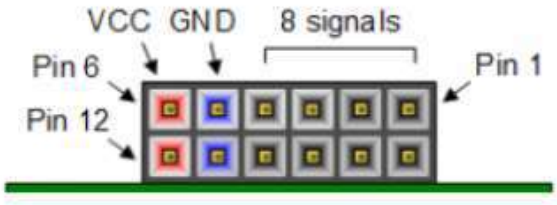
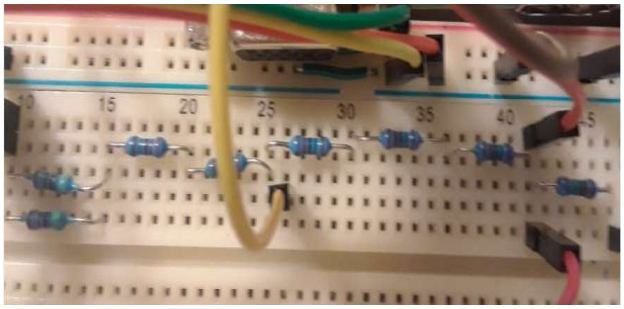


Figure 16. Pmod diagram.



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.

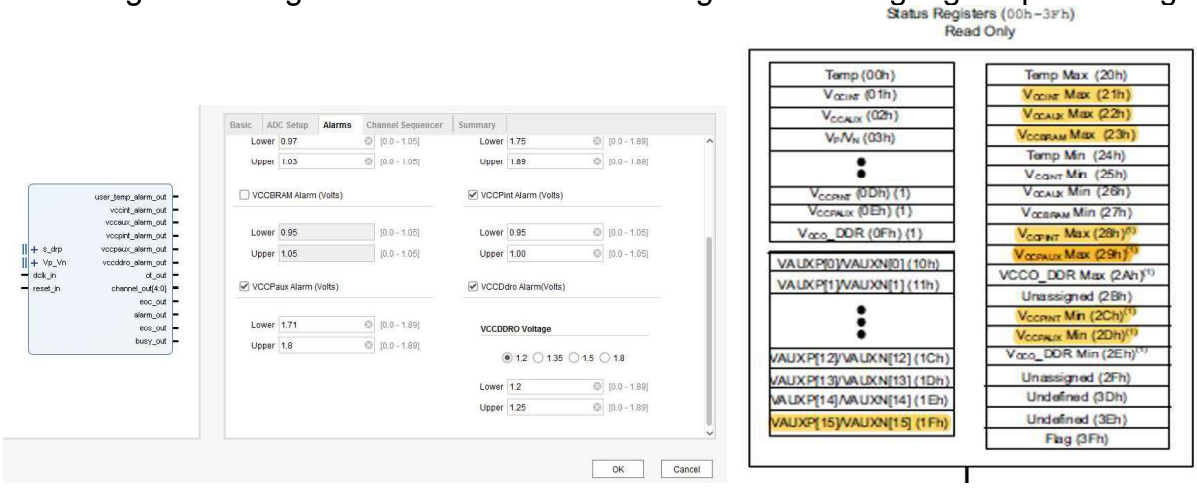
There is no circuit to build when implement the basic Pmod VGA. Very simple by connecting the VGA connector to the Zybo-Z7 JC and JD Pmod port to the VGA monitor. Only use the VGA monitor with

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.

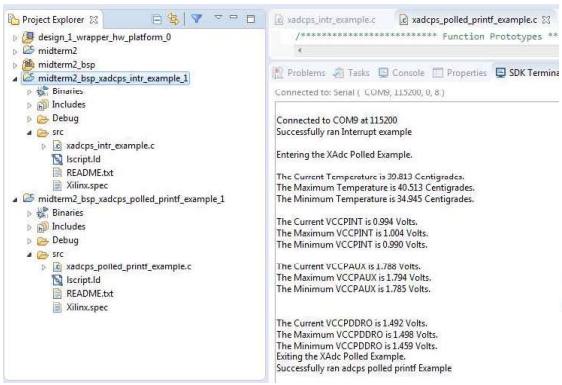
Theory:

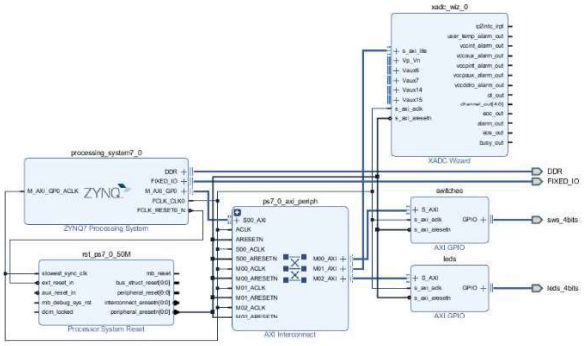
XADC is a 10-bit, 200-ksps analog to digital convertor reside in PS of Zynq 7000. It monitors the onboard supply voltages and temperature. It has 16 auxiliary differential analog-input pairs. XADCps can take in external analog input through the16 auxiliary differential analog-input pairs. It then converts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

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.

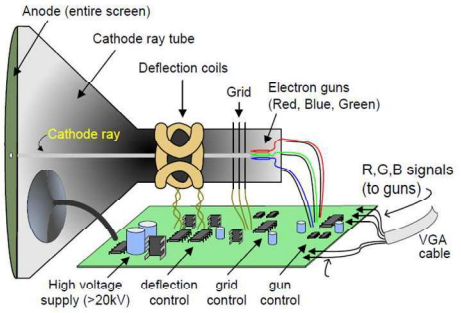


Figure 9. Color CRT display.

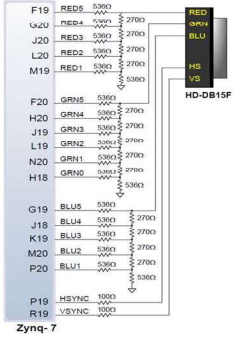
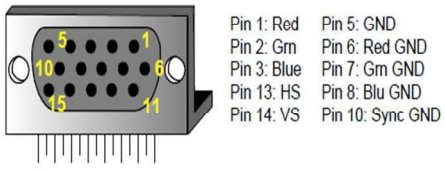


Figure 8. ZYBO VGA circuit.



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.

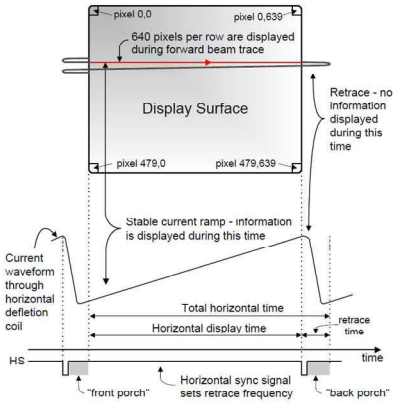
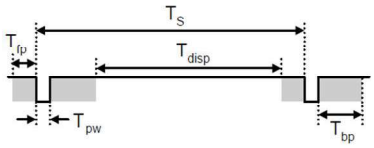


Figure 10. VGA horizontal synchronization.



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.

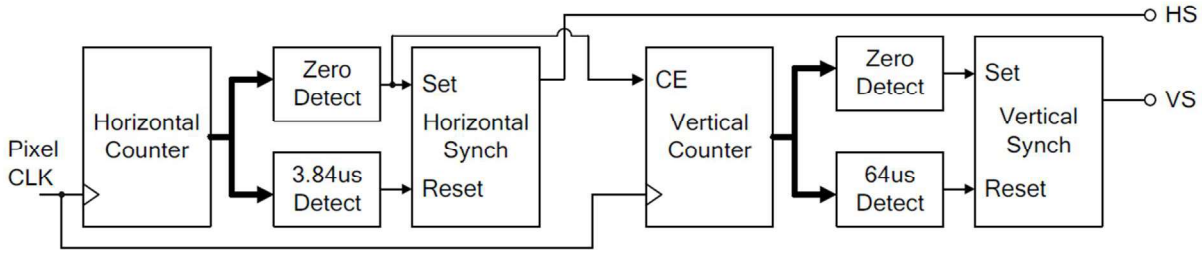


Figure 12. VGA display controller block diagram.

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.

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_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)
);
```

```
    //////////////////////////////////////////////////////////////////
    //LED PWM
    //////////////////////////////////////////////////////////////////

    integer pwm_end = 4070;
    //filter out tiny noisy part of signal to achieve zero at ground
    assign shifted_data0 = (data0 >> 4) & 12'hff0;
    assign shifted_data1 = (data1 >> 4) & 12'hff0;
    assign shifted_data2 = (data2 >> 4) & 12'hff0;
    assign shifted_data3 = (data3 >> 4) & 12'hff0;

    integer pwm_count = 0;

    //Pwm the data to show the voltage level
    always @(posedge(S_AXI_ACLK))begin
        if(pwm_count < pwm_end)begin
            pwm_count = pwm_count+1;
        end
        else begin
            pwm_count=0;
        end
    end
    //leds are active high
    assign led[0] = (sw_reg[0] == 1'b0) ? 1'b0 : (pwm_count < shifted_data0 ? 1'b1 : 1'b0);
    assign led[1] = (sw_reg[1] == 1'b0) ? 1'b0 : (pwm_count < shifted_data1 ? 1'b1 : 1'b0);
    assign led[2] = (sw_reg[2] == 1'b0) ? 1'b0 : (pwm_count < shifted_data2 ? 1'b1 : 1'b0);
    assign led[3] = (sw_reg[3] == 1'b0) ? 1'b0 : (pwm_count < shifted_data3 ? 1'b1 : 1'b0);

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:

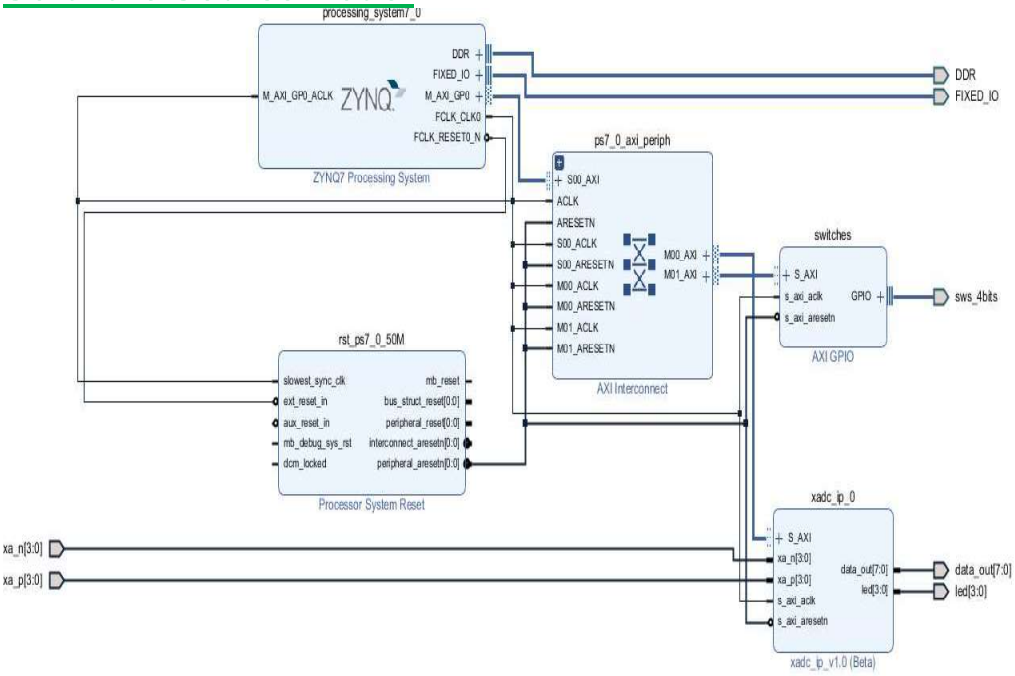
```
entity top is
    Port ( CLK_I : in  STD_LOGIC;
           VGA_HS_O : out  STD_LOGIC;
           VGA_VS_O : out  STD_LOGIC;
           VGA_R : out  STD_LOGIC_VECTOR (3 downto 0);
           VGA_B : out  STD_LOGIC_VECTOR (3 downto 0);
           VGA_G : out  STD_LOGIC_VECTOR (3 downto 0));
end top;

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.

```
#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++);
   }
}
```

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-4ADwI1c-kgA?e=yxK7UP

VGA

https://csulb-my.sharepoint.com/:v:/g/personal/julie_kim_student_csulb_edu/ES2XuPdMJtpDqDI4PwPiJPcBr10PMNRjdB7yKdbJO2ZrZg?e=If0LwV

References:

- 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
- 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

XADC-Analog to Digital Convertor

CECS 561
April 9, 2021
Julie Kim

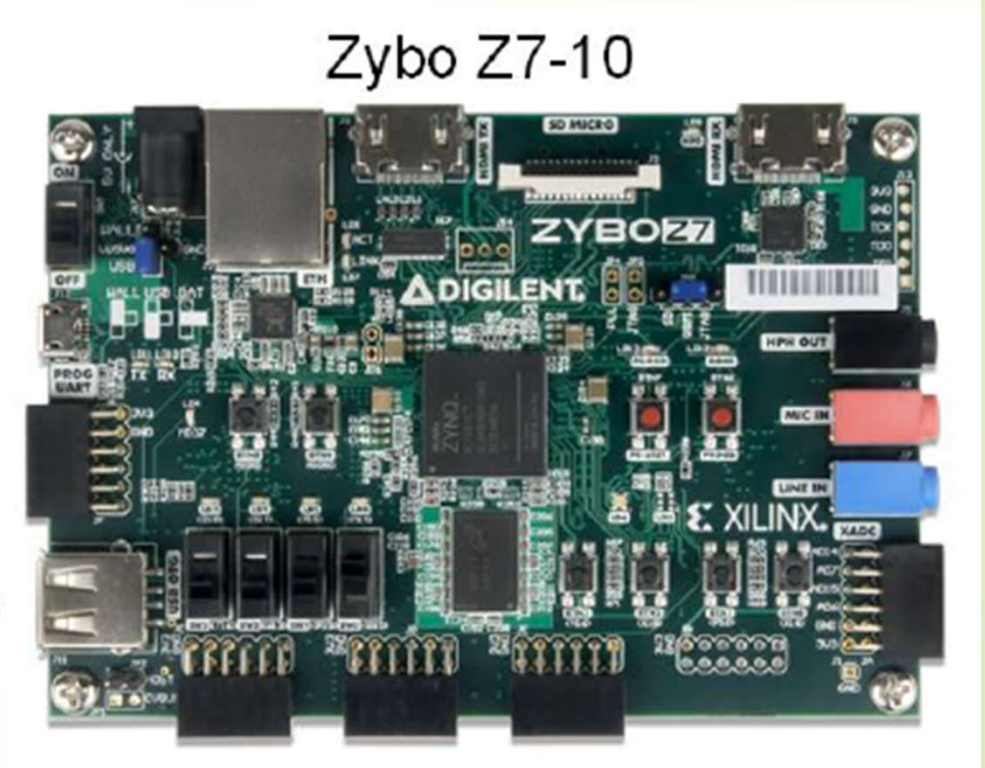
Agenda

1. Introduction to XADC
2. XADC hardware design
3. SDK software design flow
4. Challenges, implementation and solutions
5. Future work
6. References

XADCps Documentation

XADC—Analog to Digital Convertor

- 10-bit, 200-KSPS (kilo samples per second) Analog-to-Digital Converter (ADC)
- Monitoring of on-chip supply voltages and temperature
- 1 dedicated differential analog-input pair and 16 auxiliary differential analog-input pairs
- Automatic alarms based on user defined limits for the on-chip supply voltages and temperature
- Automatic Channel Sequencer, programmable averaging, programmable acquisition time for the external inputs, unipolar or differential input selection for the external inputs
- Optional interrupt request generation



XADCps BLOCK DIAGRAM

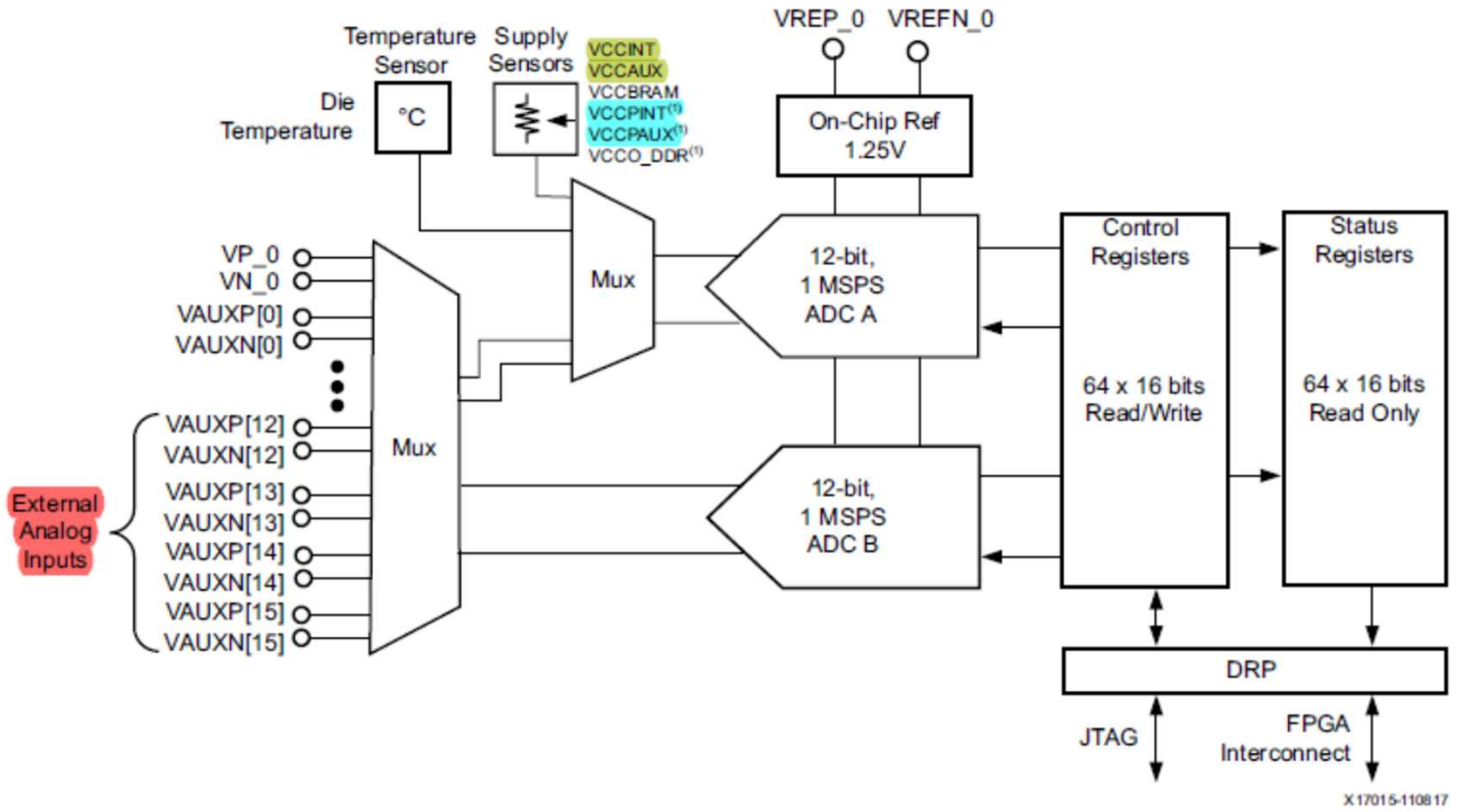


Figure 1-1: **XADC Block Diagram**

Reading Digital Value from Status Registers

- The ADC conversion data is stored in dedicated registers called status registers.
- These registers are accessible through the FPGA interconnect using a 16-bit synchronous read and write port called dynamic reconfiguration port (DRP)
- To allow access to the status register from FPGA logic design, the XADC must be instantiated.
- This mean instantiate XADCps in PL by using Dynamic Reconfiguration Port(DRP).

XADC Ports

Figure 1-3 shows the ports on the XADC primitive, and Table 1-2 describes the functionality of the ports.

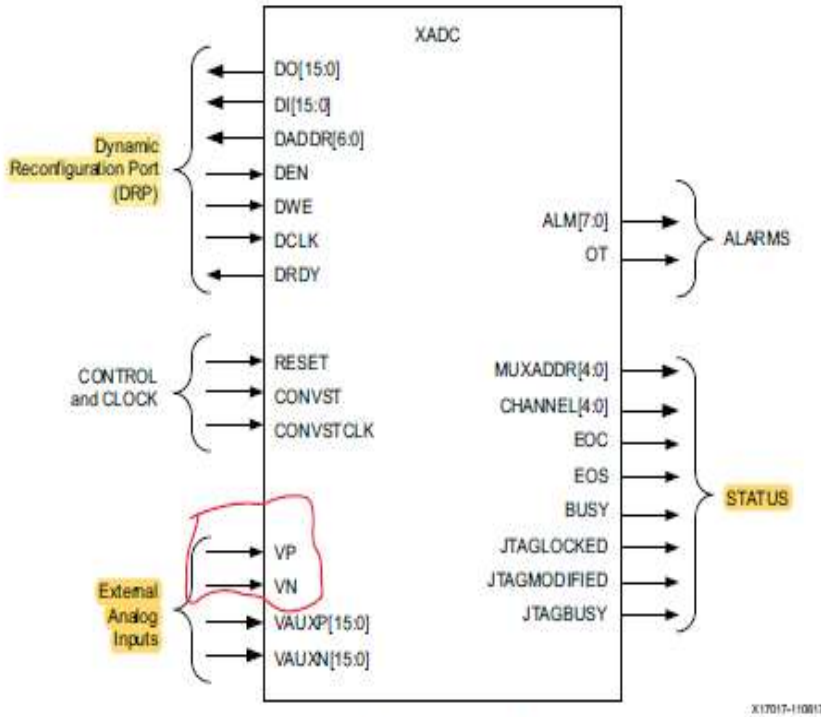
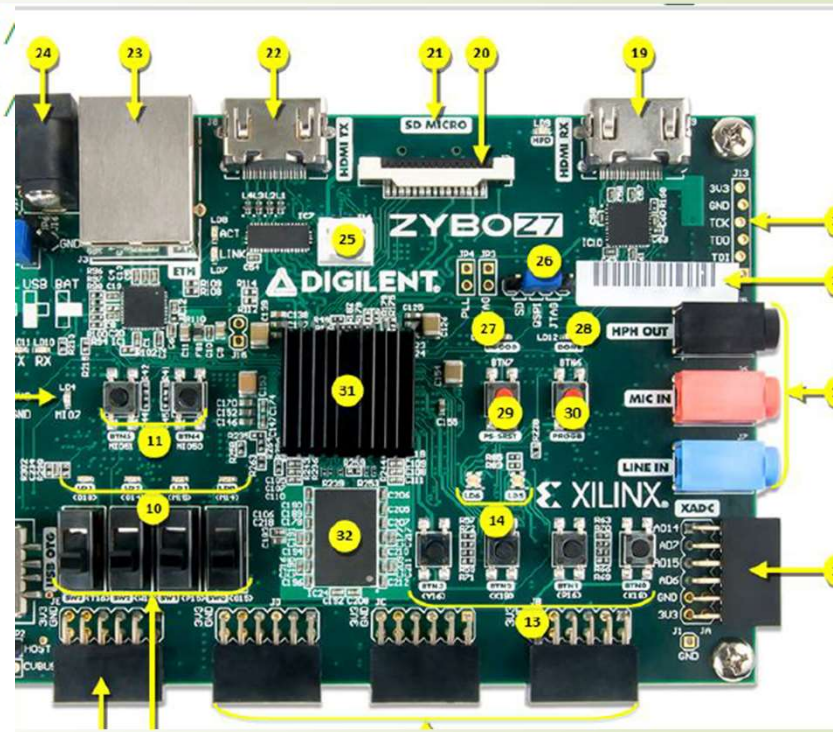


Figure 1-3: **XADC Primitive Ports**

XADC Instantiation Example

- sw is the input from the board to turn on and off the led.
- clk is signal from Zyqn 7000
- There are four auxiliary analog input: vauxp15, vauxp14, vauxp7, vauxp6
- channel_out is the address of each of the four status registers
- data is the digit value correspond to the voltage input of the auxiliary analog input

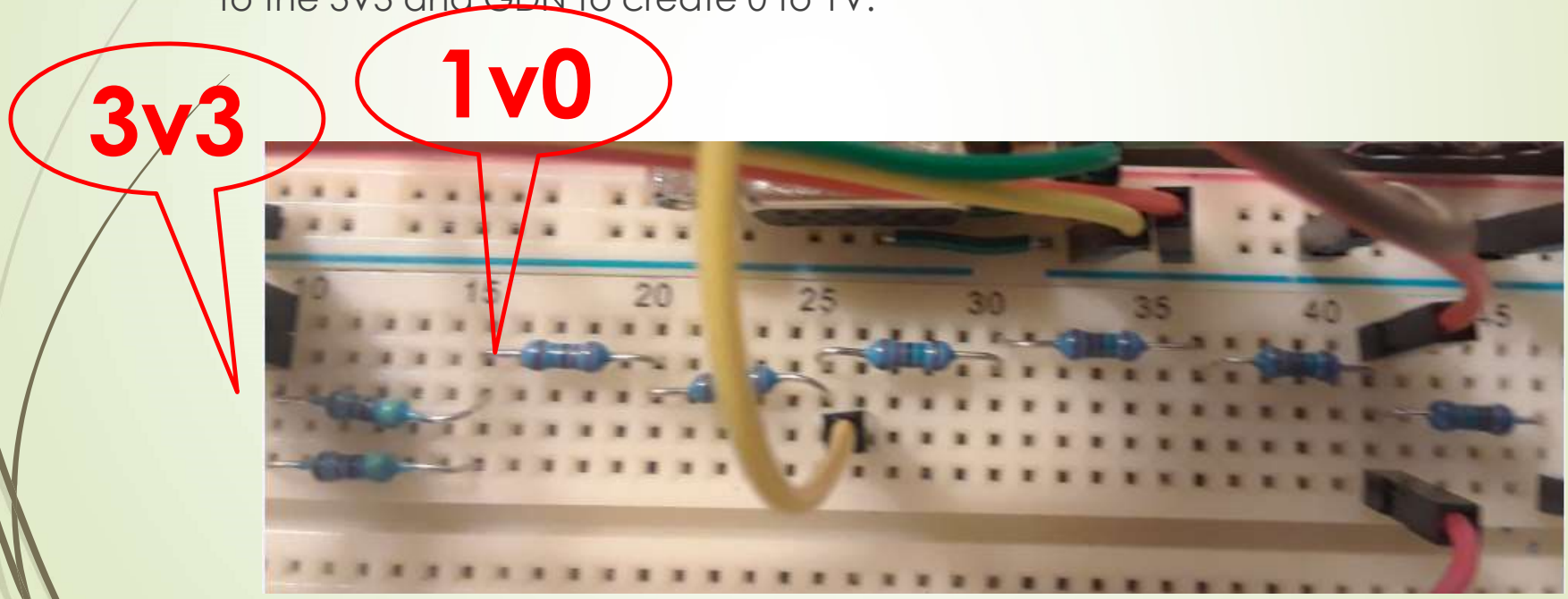
```
////////////////////////////////////
//XADC Instantiation
////////////////////////////////////

xadc_wiz_0  XLXI_7 (
    .daddr_in     (Address_in),
    .dclk_in      (clk),
    .den_in       (enable & |sw),
    .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        (vp_in),
    .vn_in        (vn_in),
    .eoc_out      (enable),
    .channel_out  (channel_out),
    .drdy_out     (ready)
);
```



JK1

Hardware Implementation
Voltage Input Conversion

- Source 3.3 v from Zybo z7 board and supply to the voltage divider circuit
- The circuit uses a chain of 1 Kohm resisters in series with two parallel 47 Kohm tied to the 3V3 and GDN to create 0 to 1V.



Slide 7

JK1 Julie Kim, 4/6/2019

LED Brightness by PWM Value

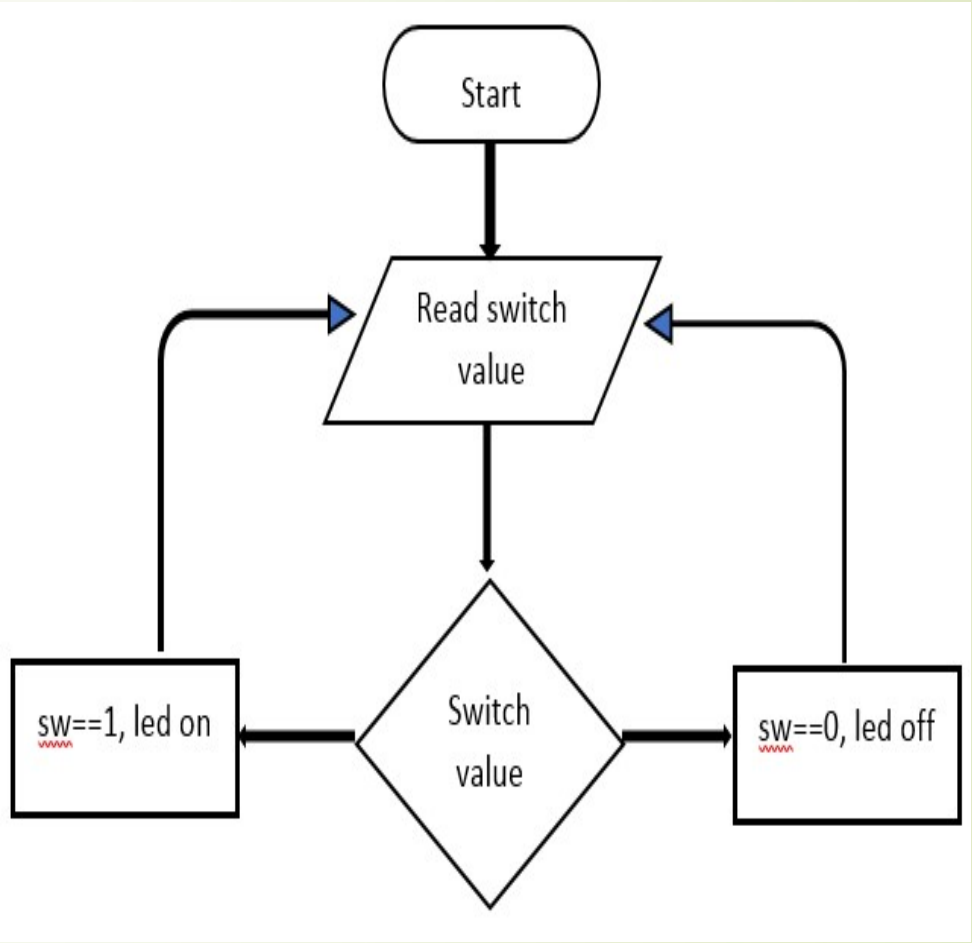
- The voltage value is converted into the digital value. Read the digital value from the status register which is done in hardware design
- Compare the output digital value to the floor value given as **4070** in decimal

```
////////////////////////////////////////////////////////////////////////
//LED PWM
////////////////////////////////////////////////////////////////////////

integer pwm_end = 4070;
//filter out tiny noise part of signal to achieve zero at ground
assign shifted_data0 = (data0 >> 4) & 12'hff0;
assign shifted_data1 = (data1 >> 4) & 12'hff0;
assign shifted_data2 = (data2 >> 4) & 12'hff0;
assign shifted_data3 = (data3 >> 4) & 12'hff0;

integer pwm_count = 0;

//Pwm the data to show the voltage level
always @(posedge(S_AXI_ACLK))begin
    if(pwm_count < pwm_end)begin
        pwm_count = pwm_count+1;
    end
    else begin
        pwm_count=0;
    end
end
//leds are active high
assign led[0] = (sw_reg[0] == 1'b0) ? 1'b0 : (pwm_count < shifted_data0 ? 1'b1 : 1'b0);
assign led[1] = (sw_reg[1] == 1'b0) ? 1'b0 : (pwm_count < shifted_data1 ? 1'b1 : 1'b0);
assign led[2] = (sw_reg[2] == 1'b0) ? 1'b0 : (pwm_count < shifted_data2 ? 1'b1 : 1'b0);
assign led[3] = (sw_reg[3] == 1'b0) ? 1'b0 : (pwm_count < shifted_data3 ? 1'b1 : 1'b0);
```

The led is on as long as the pwm_count < digital output value

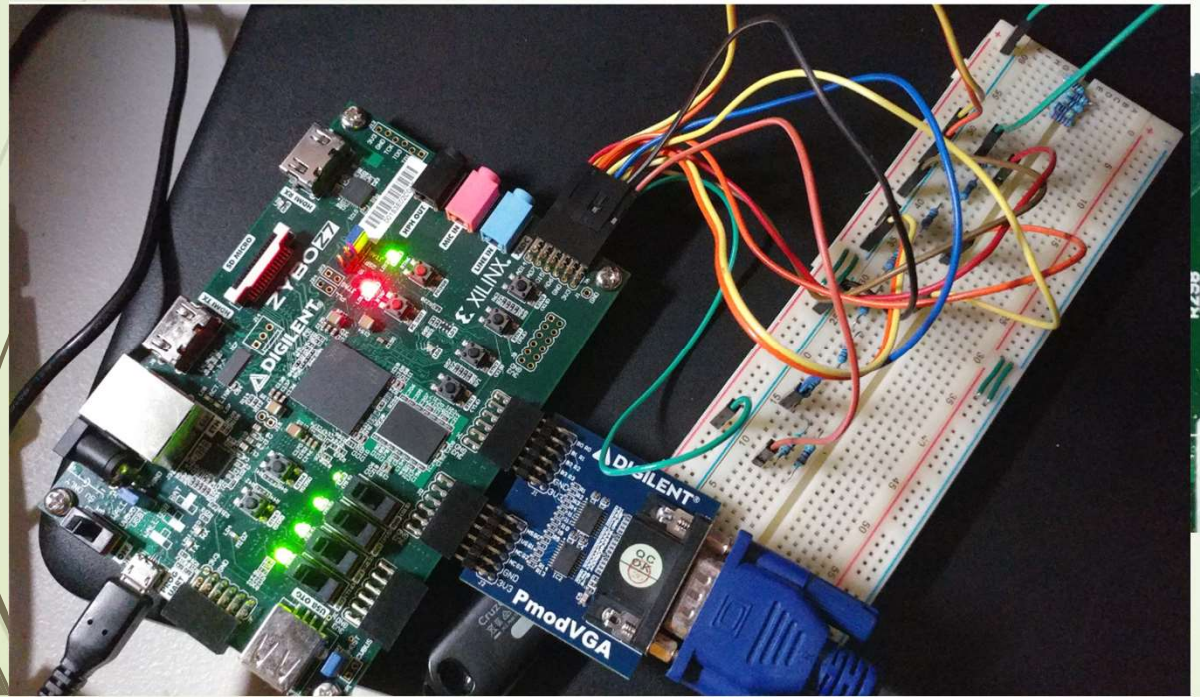
SDK Software Design Flow

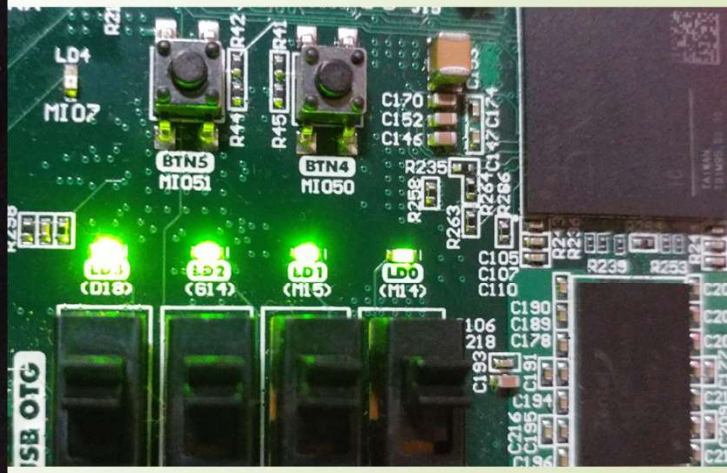
- Read dip switch value from board either to turn on or off the led
- The 0 or 1 value sent to the switch slave register of the user logic module in the custom IP.
- led turns on or off accordingly
- The software code just loop through while (1) { … }
- The brightness of the led depends on the input voltage, either 3.3 volt, 1 volt, 0.9 volt, 0.6 volt or 0.4 volt.
- The higher the voltage, the larger the digital value
- PWM counter output more pulses make led brighter as digital value is higher



Current Result

- This is the result of our custom ip. We use the instantiation of XADC and access the digital value through DRP (Dynamic Reconfiguration Port)
- We read the voltage value from board input ports, convert to digital value, drive the PWM of led all in hardware design in PL
- The brightness of led result merely from our hardware custom ip in PL.
- The SDK software platform only handle the input value of switches



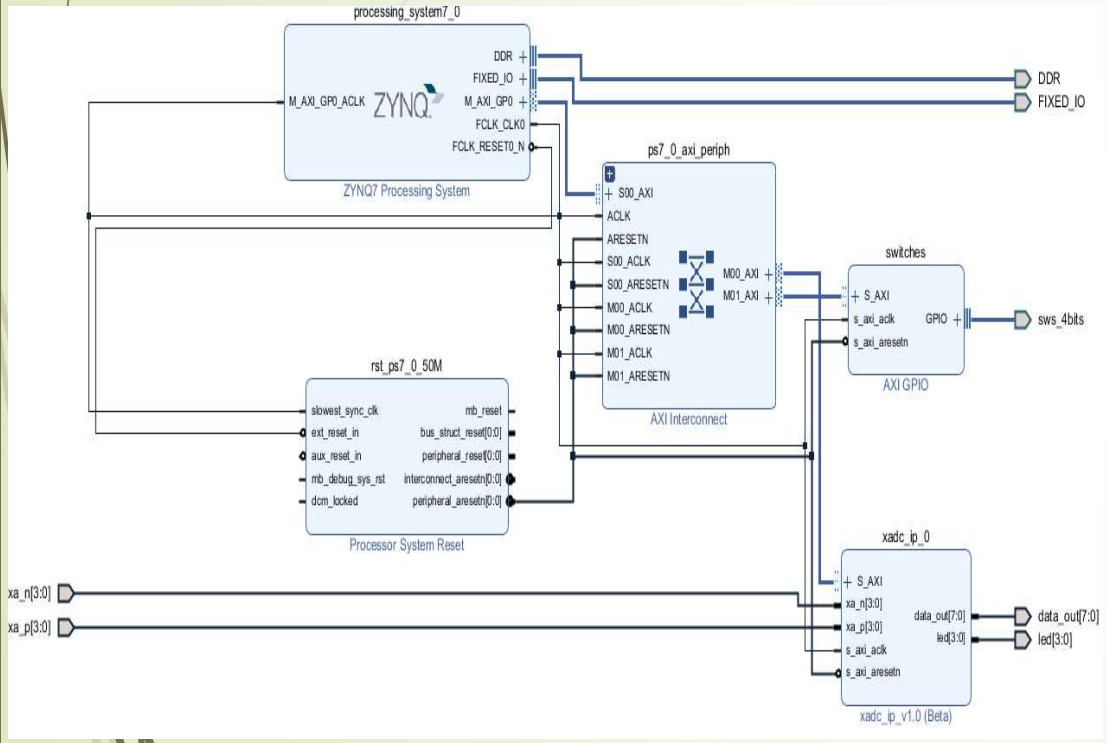


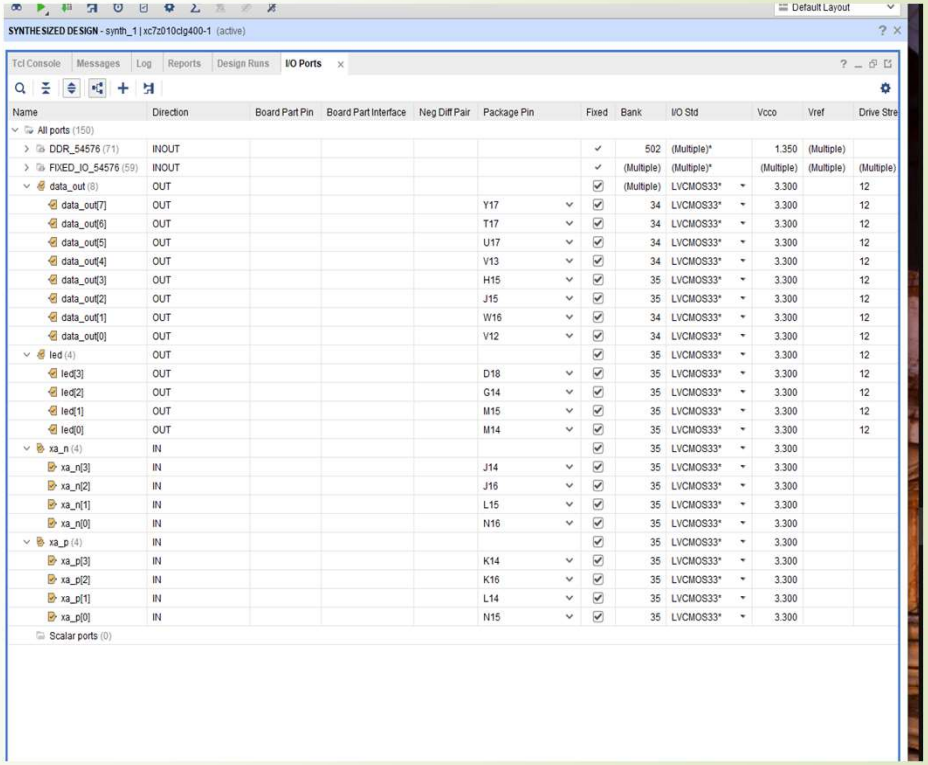
Challenges and Solution

- The example project from Digilent only givesus the functionality of the demo. e.g. the brightness of the led and switches to turn on and off the led.
- We do not have the block design diagram to implement the project both in PS and PL. We have to create our own block design diagram.
- We follow lab1 through lab 4 examples of how to create the Zyqn 7000 processor ip in PS, one XGPIO ip (the switches) in PL
- We design one custom IP name it as **XADC_custom_ip** by following lab3 and lab4
- **XADC_custom_ip** reads 4 analog inputs from the voltage divider circuit, connected to XADC instance to convert input voltage to digital value
- **XADC_custom_ip** than read the digital value through status register, output the PWM value to control the brightness of the leds.
- The switches is to turn on or off the led

Challenges and Solution

- We don't have the xadc_wizard, which is the ip provided by Vivado.
- We instantiate the xadc_wizard in our design and read the status registers through the Dynamic Reconfiguration Port (DRP).



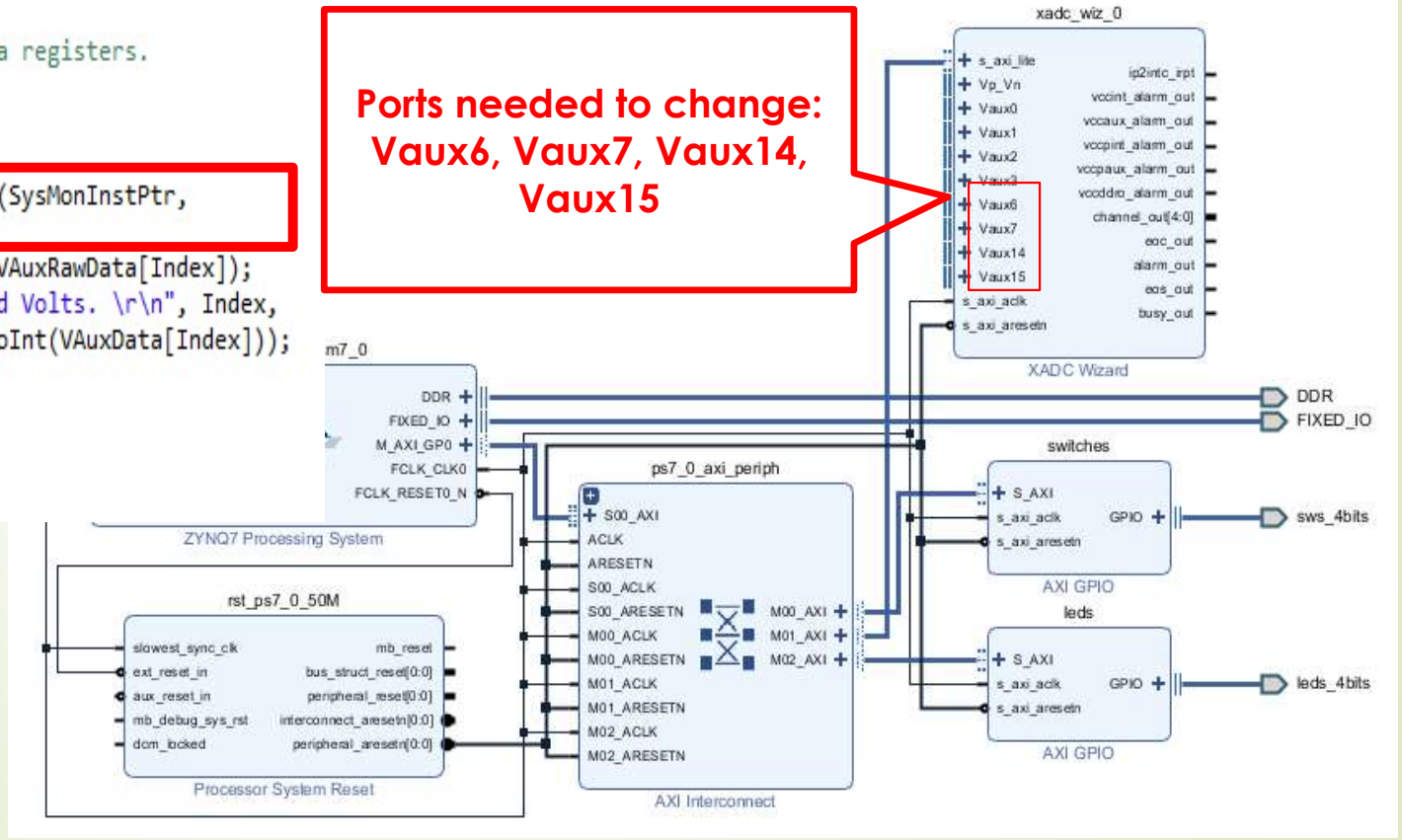


XADC_wizard Example

- Read the status registers by using the read data function provided in the .h file
- In our project we need to read from vaux6, vaux7, vaux14, vaux15
- The example project is reading from vaux0, vaux1, vaux2 and vaux3

```
/*
 * Read the ADC converted Data from the data registers.
 */
/* Read ADC data for channels 0 - 3 */
for (Index = 0; Index < 4; Index++) {
    VAuxRawData[Index] = XSysMon_GetAdcData(SysMonInstPtr,
                XSM_CH_AUX_MIN + Index);
    VAuxData[Index] = XSysMon_RawToVoltage(VAuxRawData[Index]);
    xil_printf("\r\nThe VAUX%02d is %0d.%03d Volts. \r\n", Index,
    (int)(VAuxData[Index]), SysMonFractionToInt(VAuxData[Index]));
}


return XST_SUCCESS;
```

Ports needed to change: Vaux6, Vaux7, Vaux14, Vaux15

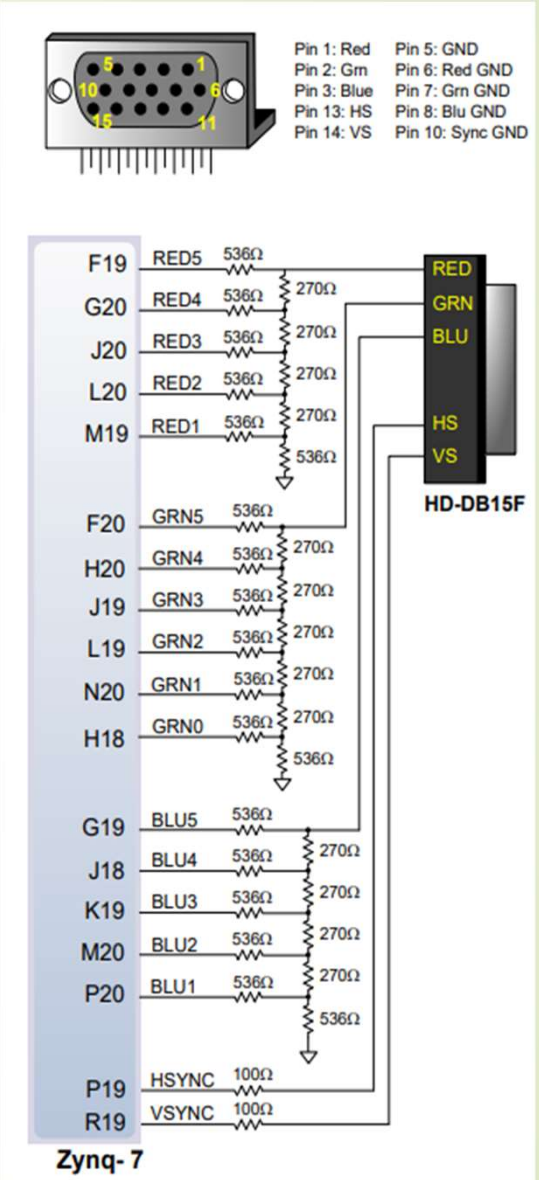


Future Plan

- We want to incorporate the **vivado xadc_wizard** into our project
- Vivado xadc_wizard uses access the status register not by the DRP port (Dynamic Reconfiguration Port).
- We can have two separated IPs as xadc_wizad and our XADC_custom_ip. We don't need to instantiate xadc instance in our custom ip.
- We can access the status register through SDK software platform by calling the function XSysMon_GetAdcData(SysMonInstPtr, address); to read the raw voltage value from analog input.
- We convert the digital value to analog voltage by calling function as: XsysMon_RawToVoltage(data);
- We want to use a potentiometer to control the brightness of led in real time.
- Another plan is we want to use the digital value to control an animation.
- VGA animation output to the monitor

VGA Port

- 18 programmable logic pins to create VGA output port.
- The digital-to-analog conversion is done using R-2R resistor circuit.
- The VGA display to create 32 and 64 analog signal levels red, blue, green VGA signals.
- The video color signals is between 0v and 0.7v



VGA Logic

```
-------------------------------------------------
vga_red <= h_cntr_reg(5 downto 2) when (active = '1' and ((h_cntr_reg < 512 and v_cn
           (others=>'1')          when (active = '1' and ((h_cntr_reg < 512 and not(
           (others=>'1')          when (active = '1' and ((not(h_cntr_reg < 512) and
                                       (not(h_cntr_reg < 512) and (v_cntr_reg(8) =
           (others=>'0');

vga_blue <= h_cntr_reg(5 downto 2) when (active = '1' and ((h_cntr_reg < 512 and v_c
           (others=>'1')           when (active = '1' and ((h_cntr_reg < 512 and not
           (others=>'1')           when (active = '1' and ((not(h_cntr_reg < 512) an
                                       (not(h_cntr_reg < 512) and (v_cntr_reg(8) =
           (others=>'0');

vga_green <= h_cntr_reg(5 downto 2) when (active = '1' and ((h_cntr_reg < 512 and v_
           (others=>'1')           when (active = '1' and ((h_cntr_reg < 512 and no
           (others=>'1')           when (active = '1' and ((not(h_cntr_reg < 512) a
                                       (not(h_cntr_reg < 512) and (v_cntr_reg(8)
           (others=>'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;
```

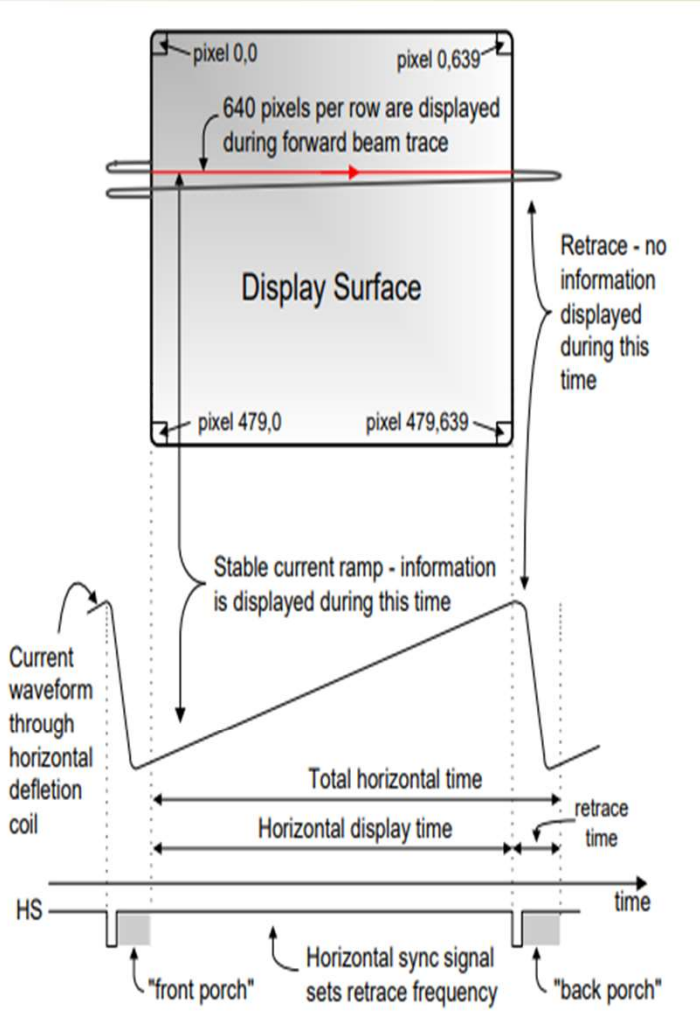


Figure 10. VGA horizontal synchronization.

VGA Current Result

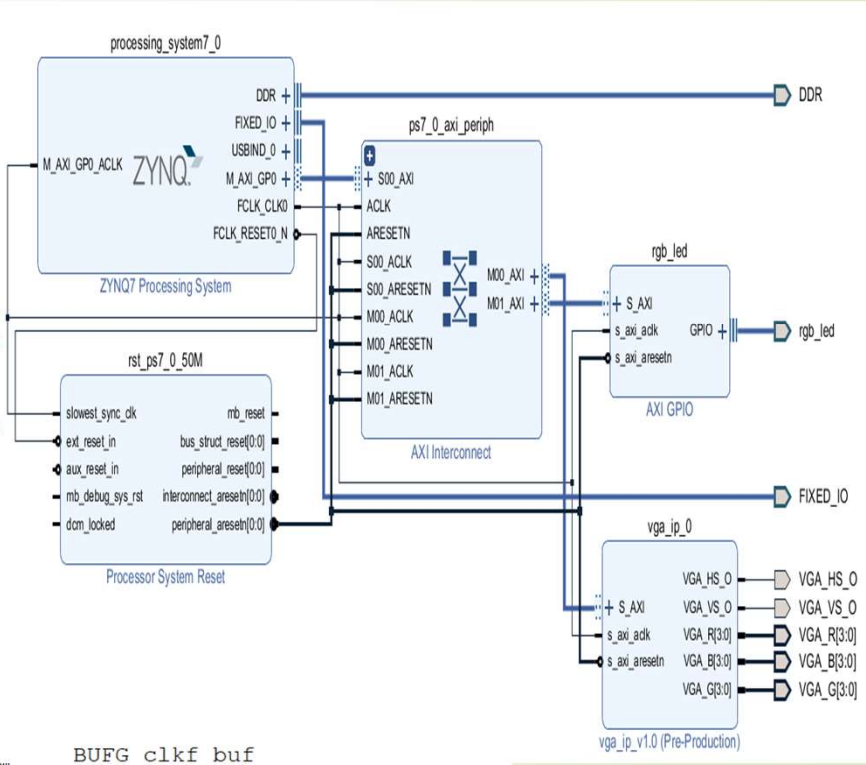
- Implementation (2 errors, 1 critical warning)
  - Place Design (1 critical warning)
    - DRC (1 critical warning)
      - Netlist (1 critical warning)
        - Instance (1 critical warning)
          - Invalid attribute value (1 critical warning)
            - MMCM_ADV (1 critical warning)

[DRC AVAL-46] v7v8_mmcm_fvco_rule1: The current computed target frequency, FVCO, is out of range for cell design_1_i/vga_ip_0/inst/vga_ip_v1_0_S_AXI_inst/vga_inst/clk_div_inst/inst/mmcm_adv_inst. Th FVCO is 371.250 MHz. The valid FVCO range for speed grade -1 is 600MHz to 1200MHz. The cell attribute values used to compute FVCO are CLKFBOUT_MULT_F = 37.125, CLKIN1_PERIOD = 20.00000, and DIVCLK_DIVIDE = 5 (FVCO = 1000 * CLKFBOUT_MULT_F/(CLKIN1_PERIOD * DIVCLK_DIVIDE)).
This violation may be corrected by:
1. The timer uses timing constraints for clock period or clock frequency that affect CLKIN1 to set cell attribute CLKIN1_PERIOD, over-riding any previous value. This may already be in place and, if so this violati resolved once Timing is run. Otherwise, consider modifying timing constraints to adjust the CLKIN1_PERIOD and bring FVCO into the allowed range.
2. In the absence of timing constraints that affect CLKIN1, consider modifying the cell CLKIN1_PERIOD to bring FVCO into the allowed range.
3. If CLKIN1_PERIOD is satisfactory, modify the CLKFBOUT_MULT_F or DIVCLK_DIVIDE cell attributes to bring FVCO into the allowed range.
4. The MMCM configuration may be dynamically modified by use of DRP which is recognized by an ACTIVE signal on DCLK pin.

- Route Design (2 errors)
  - DRC (1 error)
    - Physical Configuration (1 error)
      - Physical design rule (1 error)
        - MMCME2_ADV (1 error)

[DRC PDRC-34] MMCM_adv_ClkFrequency_div_no_dclk: The computed value 371.250 MHz (CLKIN1_PERIOD, net FCLK_CLK0) for the VCO operating frequency of the MMCME2_ADV site MMCME2_ADV_X0Y0 (cell design_1_i/vga_ip_0/inst/vga_ip_v1_0_S_AXI_inst/vga_inst/clk_div_inst/inst/mmcm_adv_inst) falls outside the operating range of the MMCM VCO frequency for this device (600.000 - 1200.000 MHz). The computed value is (CLKFBOUT_MULT_F * 1000 / (CLKINx_PERIOD * DIVCLK_DIVIDE)). Please run update_timing to update the MMCM settings. If that does not work, adjust either the input period CLKINx_PERIOD (20.000000), multiplication factor CLKFBOUT_MULT_F (37.125000) or the division factor DIVCLK_DIVIDE (5), in order to achieve a VCO frequency within the rated operating range for this device.

[Vivado_Tcl 4-16] Error(s) found during DRC. Router not run.

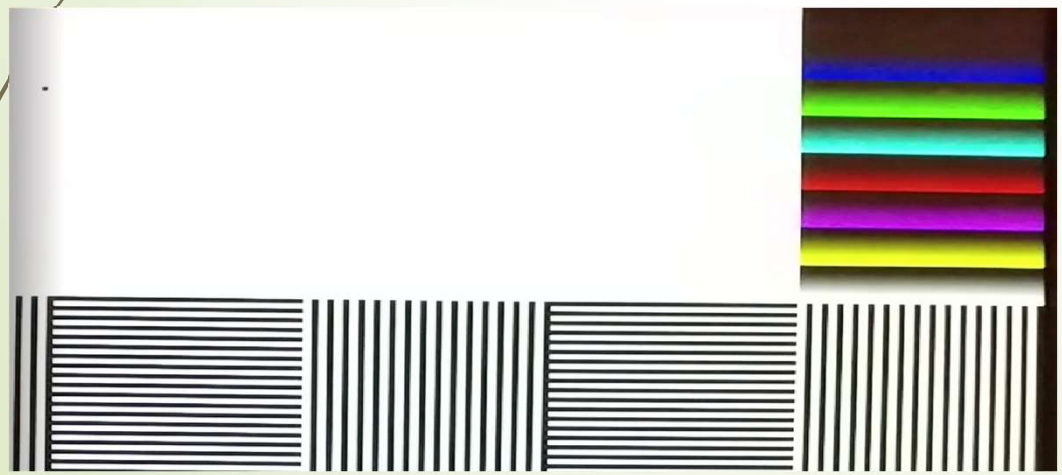


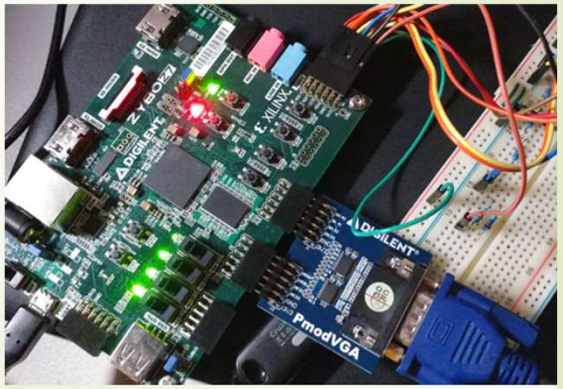

```
BUFG clkf_buf
  (.O (clkfbout_buf_clk_wiz_0),
   .I (clkfbout_clk_wiz_0));
```

```
BUFG clkout1_buf
  (.O    (clk_out1),
   .I    (clk_out1_clk_wiz_0));
```

Pmod VGA Output

- This is the usage of Pmod VGA connected to the Zybo Z7's Pmod port.
- A bouncing box and black, white, and multiple colors of bars are display on a connected VGA monitor.
- The Pmod VGA is controlled by the Zybo through Pmod port JC and JD
- The screen resolution is configure through HDL code.
- Current is 1080p
- Devices: Pmod VGA, and VGA Monitor and cable





References:

- https://reference.digilentinc.com/learn/programmable-logic/tutorials/zybo-xadc-demo/start
- 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
- 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://reference.digilentinc.com/reference/pmod/pmodvga/start
- https://reference.digilentinc.com/learn/programmable-logic/tutorials/zybo-z7-pmod-vga-demo/start
- https://github.com/Digilent/Zybo-Z7-10-Pmod-VGA/releases/tag/2016.4-2
- https://github.com/Digilent/Zybo-Z7-10-Pmod-VGA?_ga=2.93914781.1807014336.1554410949-423883057.1550577685