XADC-Analog to Digital Convertor

CECS 561
April 9, 2021
Julie Kim

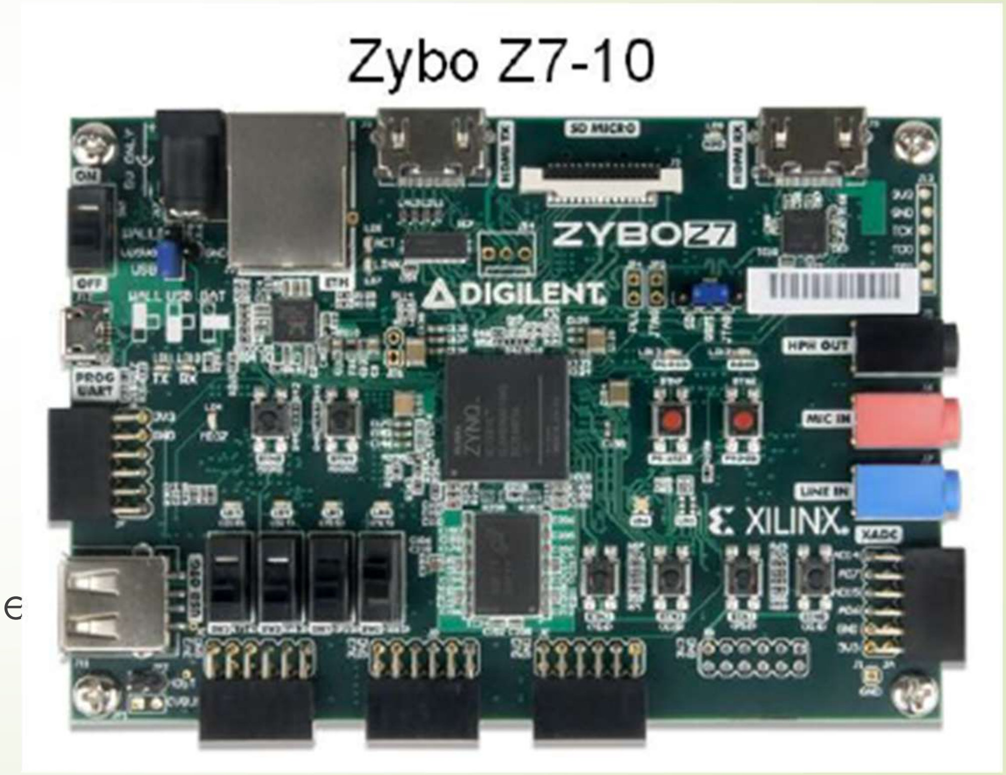
Agenda

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

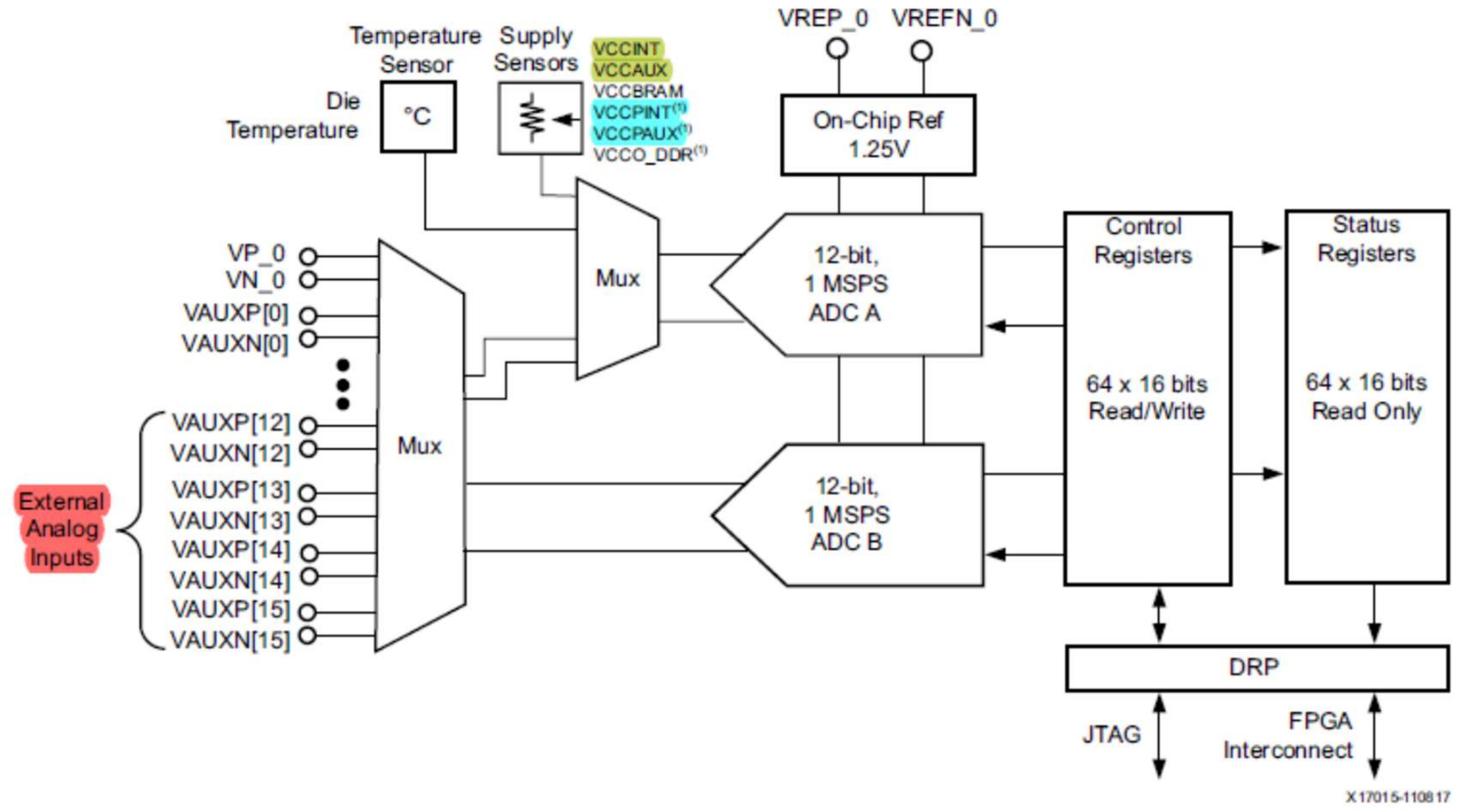


Figure 1-1: **XADC Block Diagram**

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.

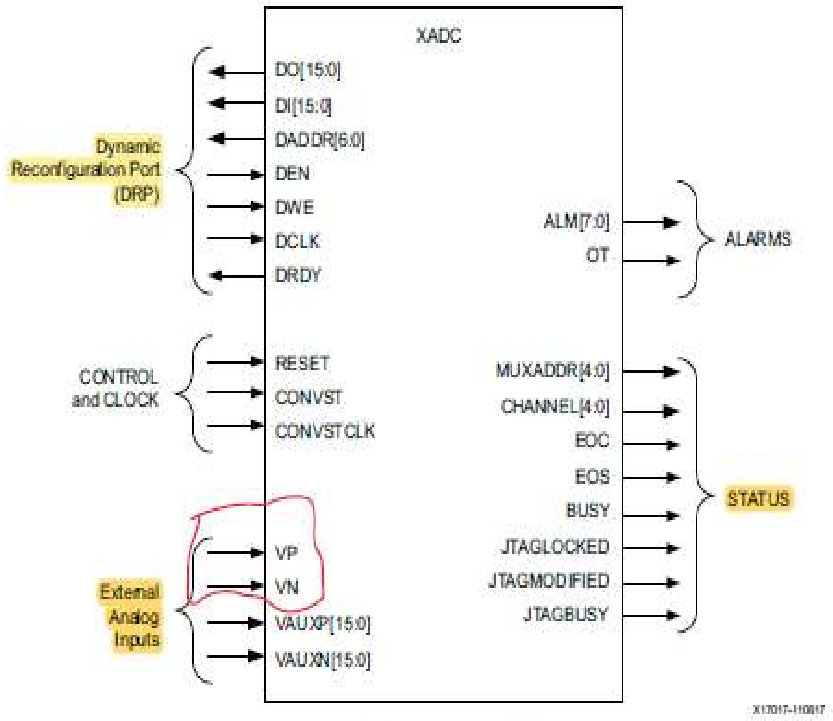
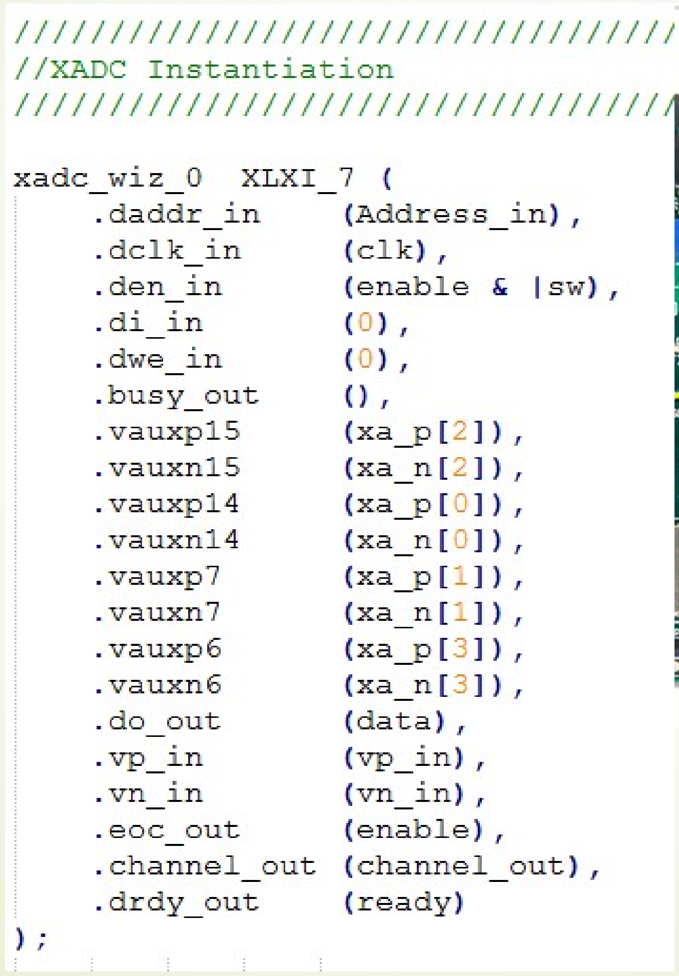


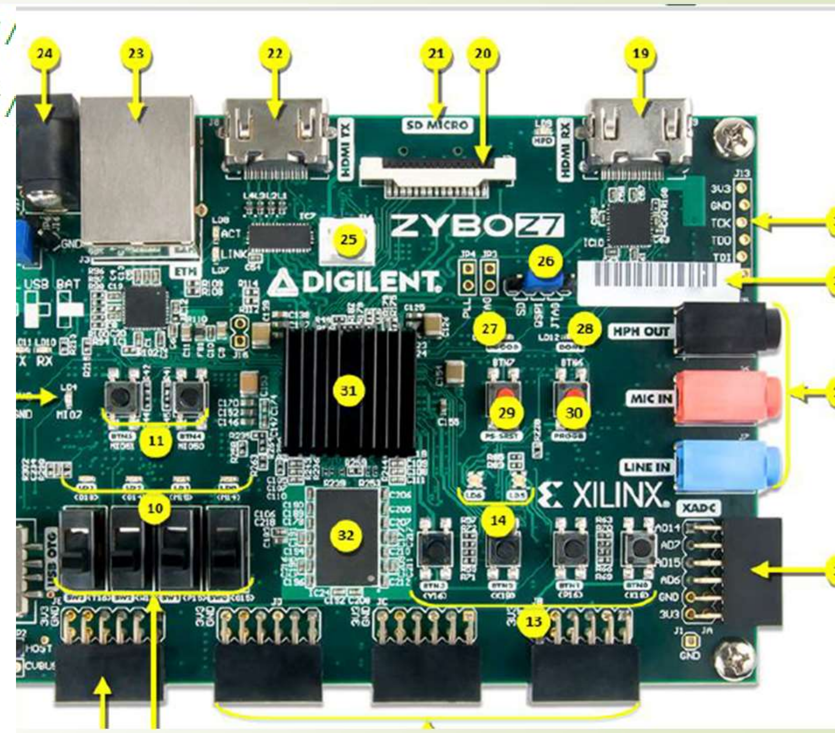
Figure 1-3: **XADC Primitive Ports**

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



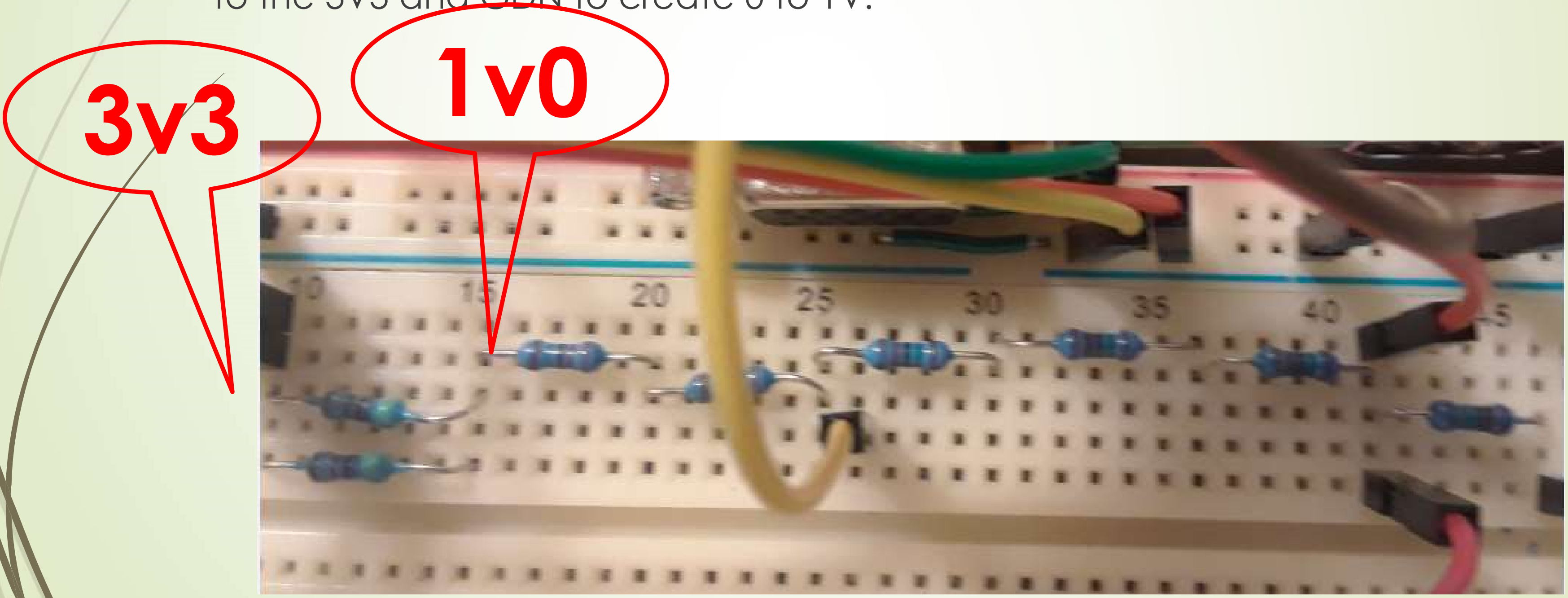
```
/////////////////////////////////////
//XADC Instantiation
/////////////////////////////////////

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.

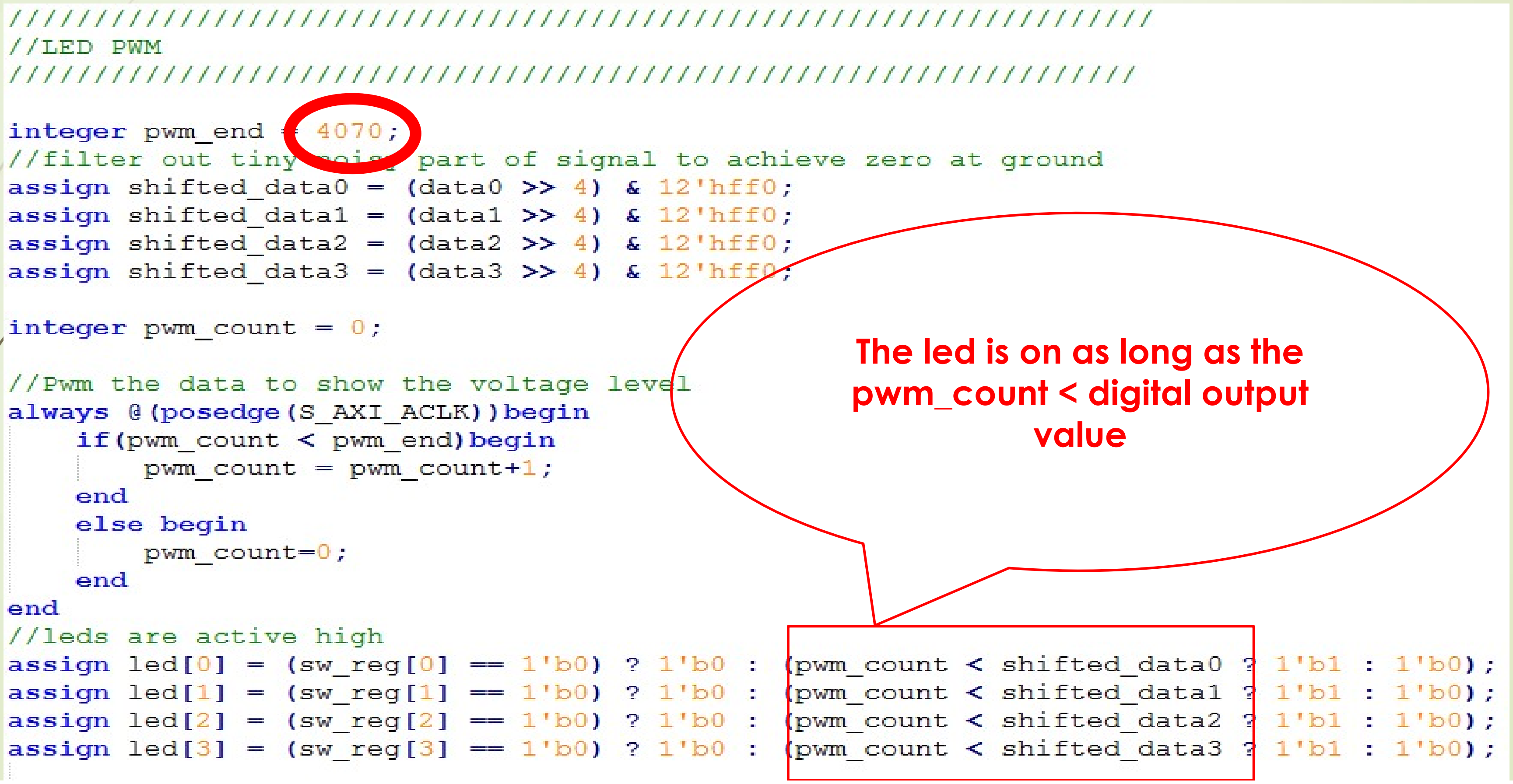


Slide 7

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

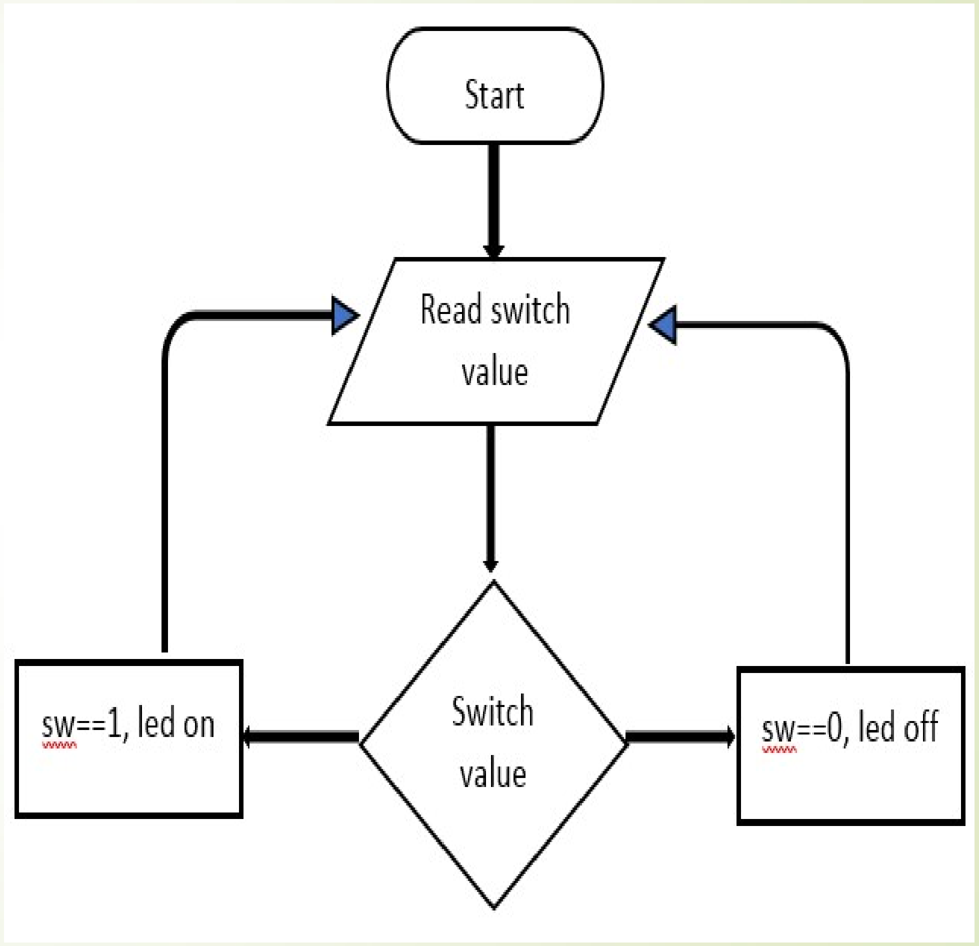


```
///////////////////////////////////////////////////////////////////
//LED PWM
//////////////////////////////////////////////////////////////////

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

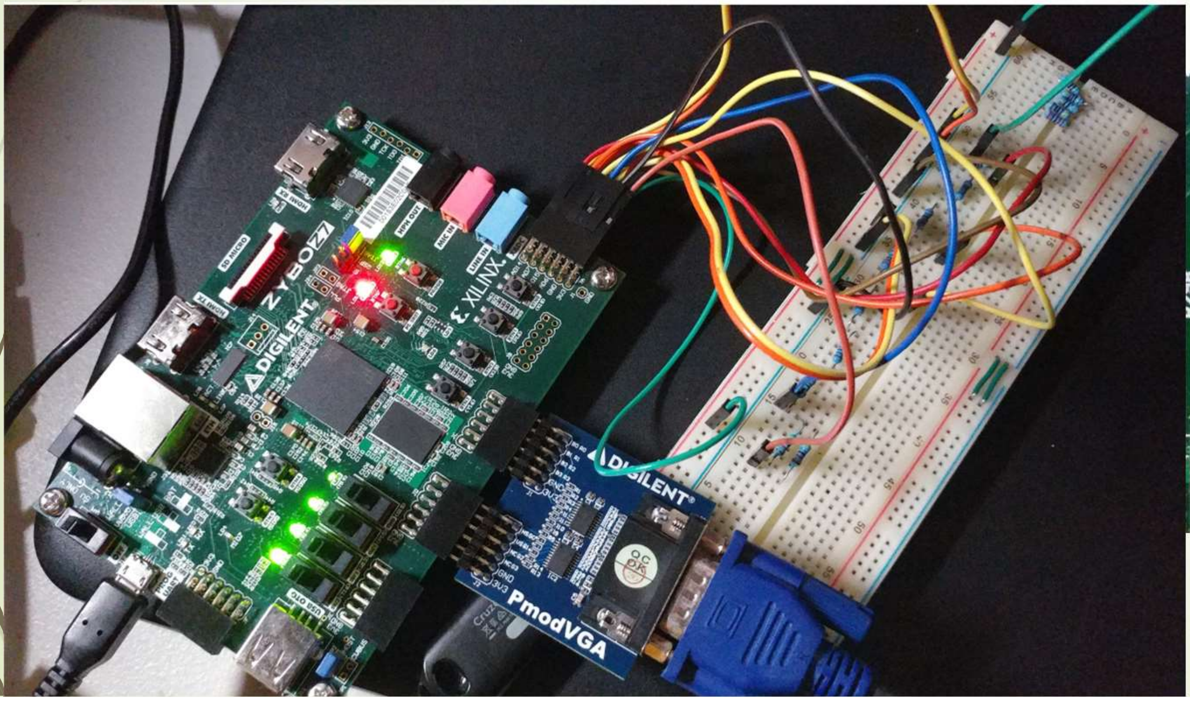
SDK Software Design Flow

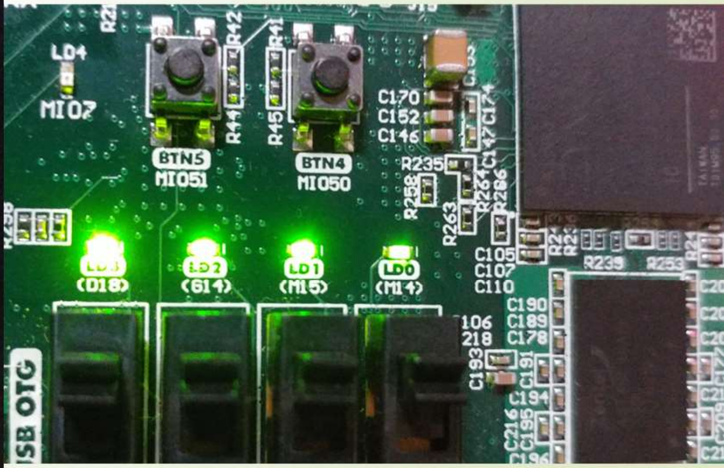
- Read dip switch value from board either to turn on or off the led
- The 0 or 1 value sent to the switch slave register of the user logic module in the custom IP.
- led turns on or off accordingly
- The software code just loop through while (1) { … }
- The brightness of the led depends on the input 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



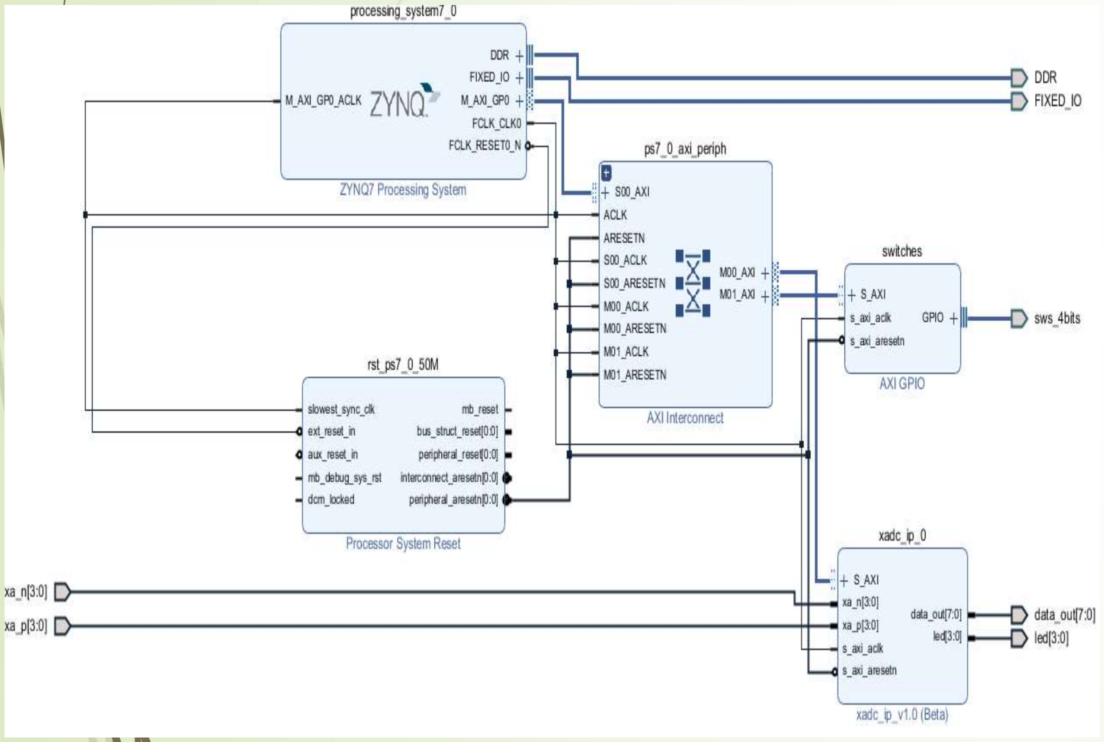


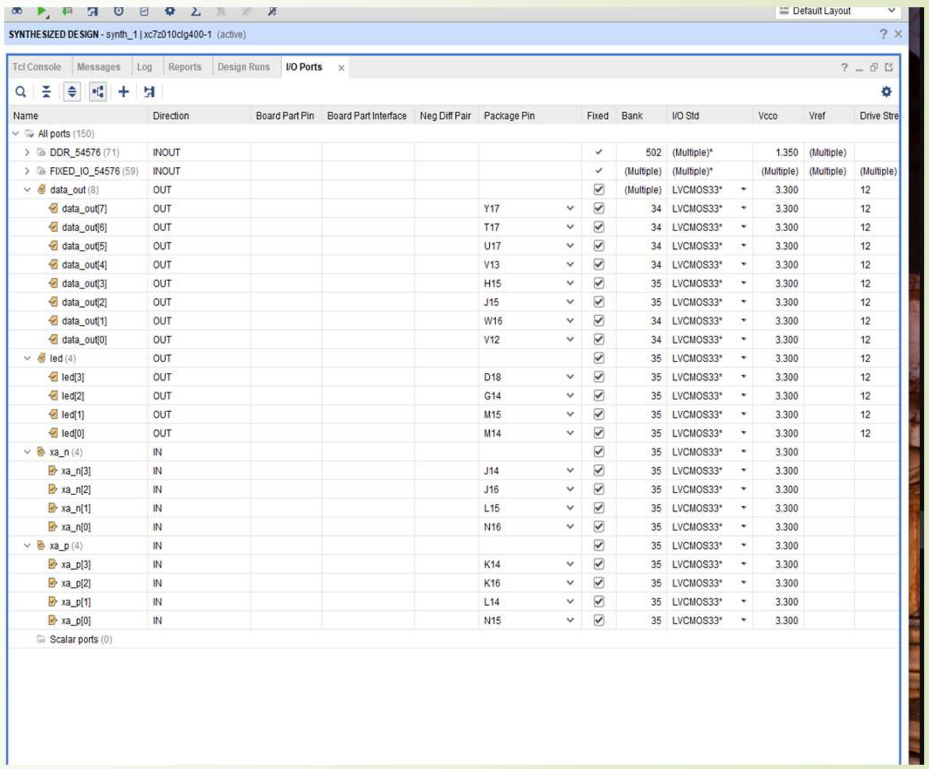
Challenges and Solution

- 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).



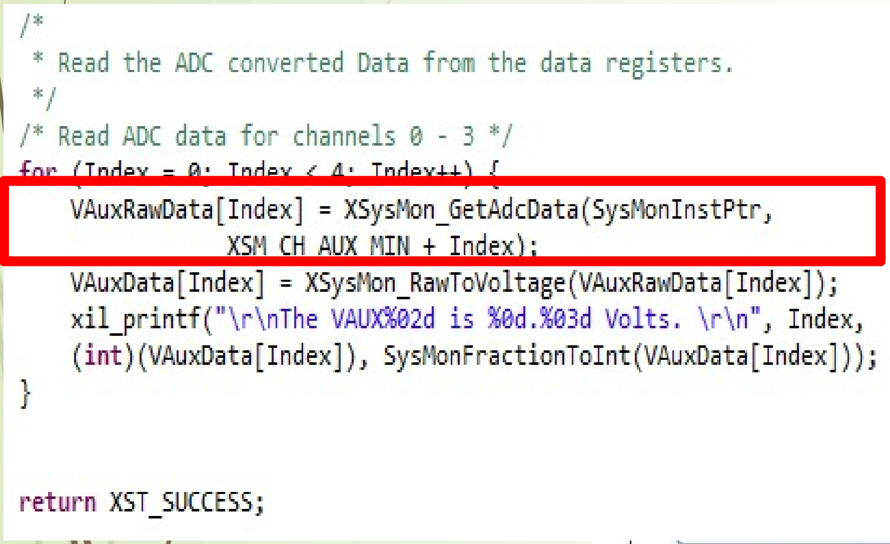


SYNTHESIZED DESIGN - synth_1 | xc7z010clg400-1 (active)

Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Stre
All ports (150)											
DDR_54576 (71)	INOUT					✓	502	(Multiple)*	1.350	(Multiple)	
FIXED_IO_54576 (59)	INOUT					✓	(Multiple)	(Multiple)*	(Multiple)	(Multiple)	(Multiple)
data_out (8)	OUT					✓	(Multiple)	LVCMOS33*	3.300		12
data_out[7]	OUT				Y17	✓	34	LVCMOS33*	3.300		12
data_out[6]	OUT				T17	✓	34	LVCMOS33*	3.300		12
data_out[5]	OUT				U17	✓	34	LVCMOS33*	3.300		12
data_out[4]	OUT				V13	✓	34	LVCMOS33*	3.300		12
data_out[3]	OUT				H15	✓	35	LVCMOS33*	3.300		12
data_out[2]	OUT				J15	✓	35	LVCMOS33*	3.300		12
data_out[1]	OUT				W16	✓	34	LVCMOS33*	3.300		12
data_out[0]	OUT				V12	✓	34	LVCMOS33*	3.300		12
led (4)	OUT					✓	35	LVCMOS33*	3.300		12
led[3]	OUT				D18	✓	35	LVCMOS33*	3.300		12
led[2]	OUT				G14	✓	35	LVCMOS33*	3.300		12
led[1]	OUT				M15	✓	35	LVCMOS33*	3.300		12
led[0]	OUT				M14	✓	35	LVCMOS33*	3.300		12
xa_n (4)	IN					✓	35	LVCMOS33*	3.300		
xa_n[3]	IN				J14	✓	35	LVCMOS33*	3.300		
xa_n[2]	IN				J16	✓	35	LVCMOS33*	3.300		
xa_n[1]	IN				L15	✓	35	LVCMOS33*	3.300		
xa_n[0]	IN				N16	✓	35	LVCMOS33*	3.300		
xa_p (4)	IN					✓	35	LVCMOS33*	3.300		
xa_p[3]	IN				K14	✓	35	LVCMOS33*	3.300		
xa_p[2]	IN				K16	✓	35	LVCMOS33*	3.300		
xa_p[1]	IN				L14	✓	35	LVCMOS33*	3.300		
xa_p[0]	IN				N15	✓	35	LVCMOS33*	3.300		
Scalar ports (0)											

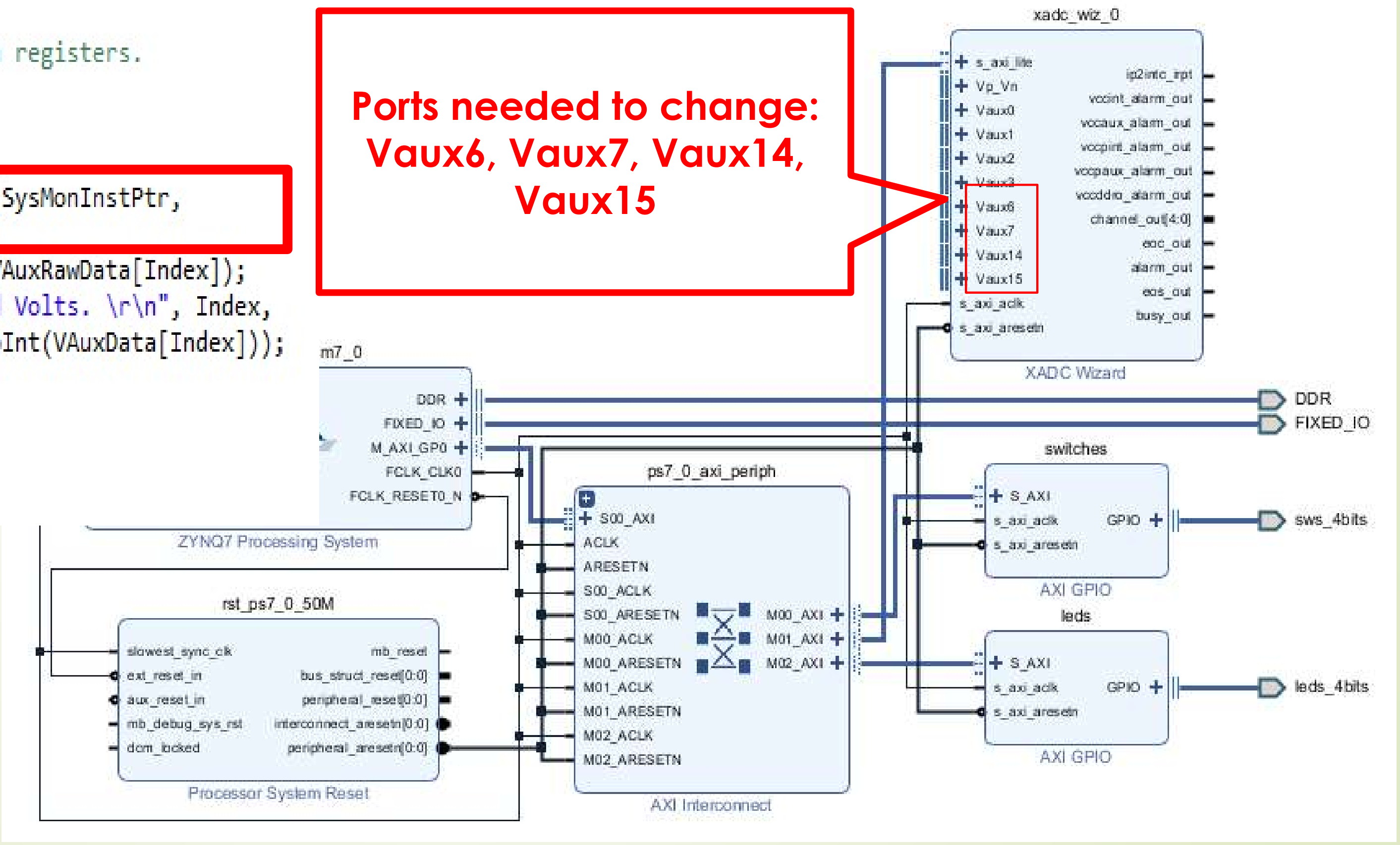
XADC_wizard Example

- 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

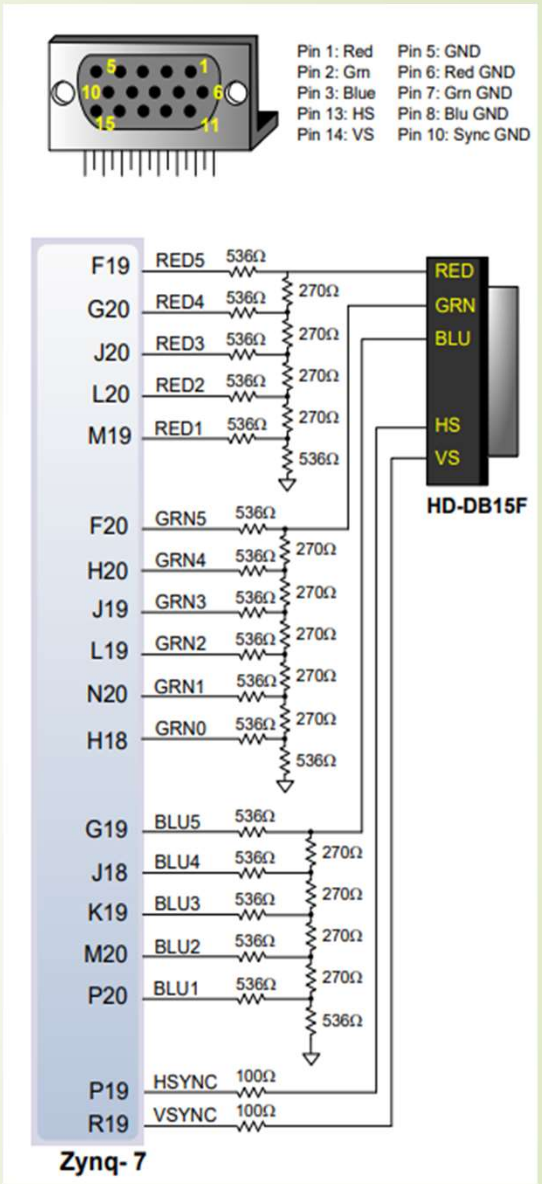


Future Plan

- 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;
```

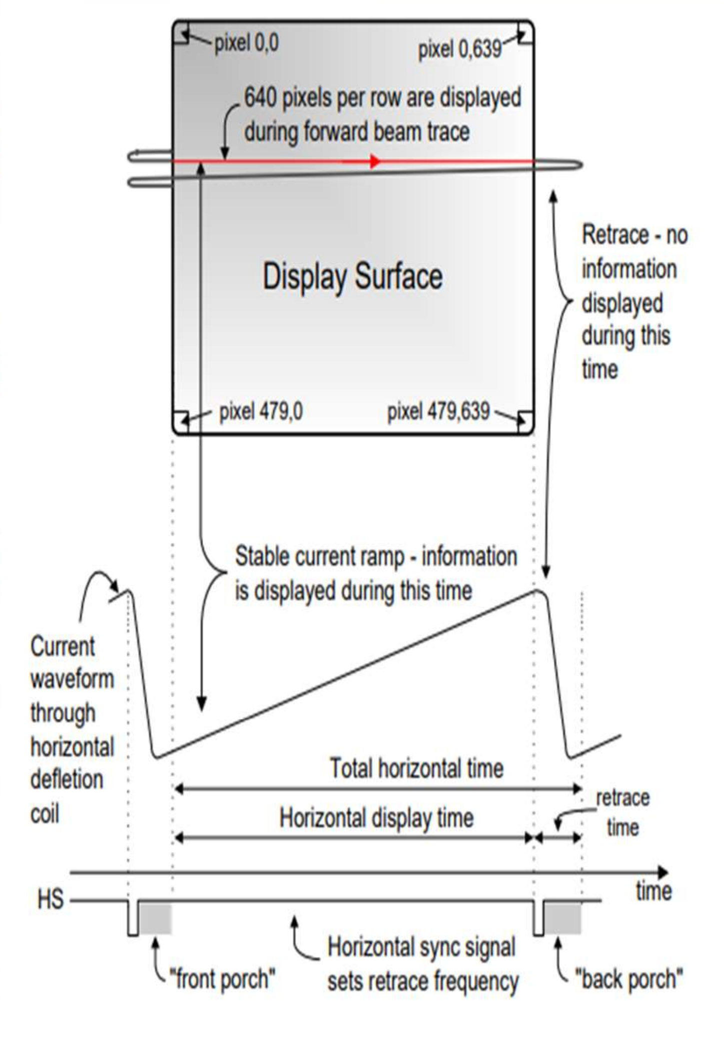
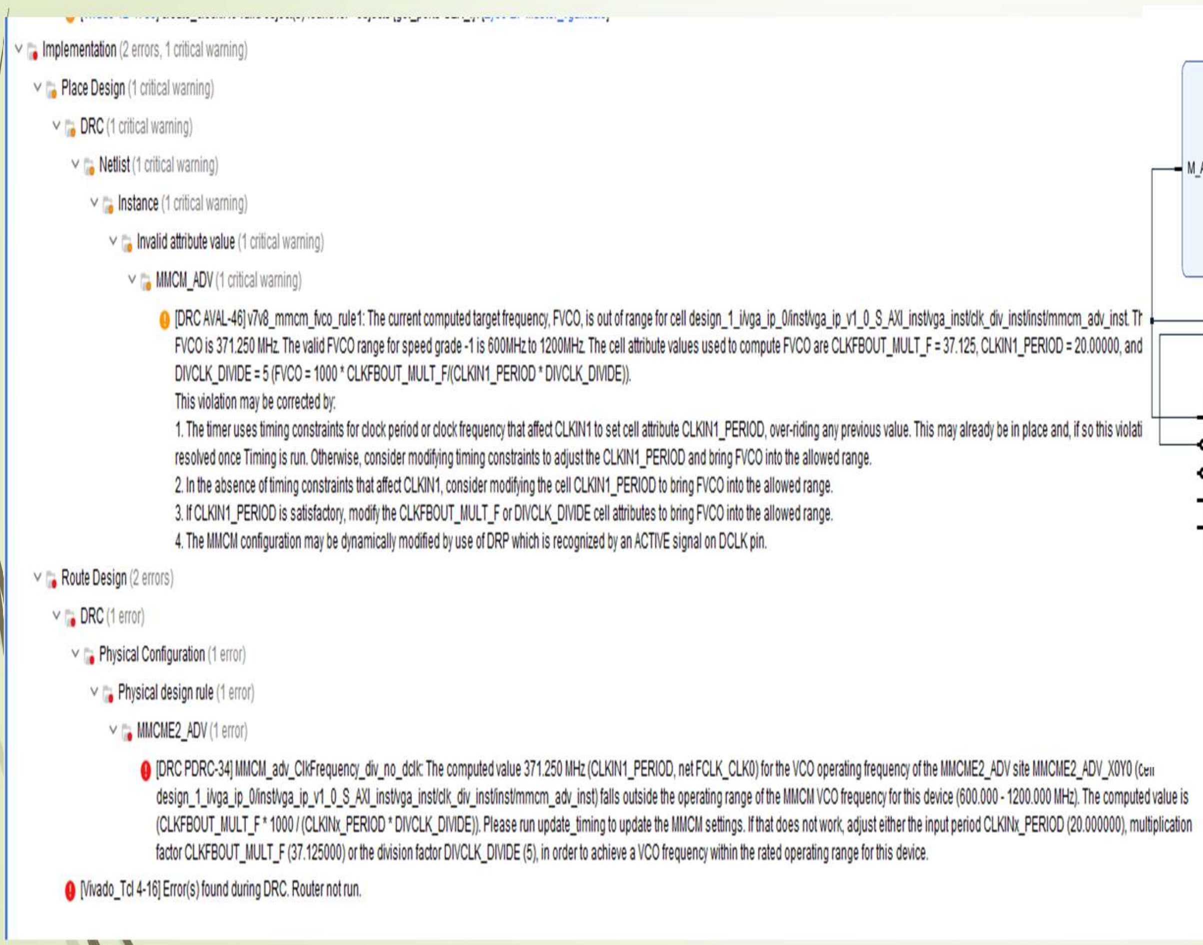
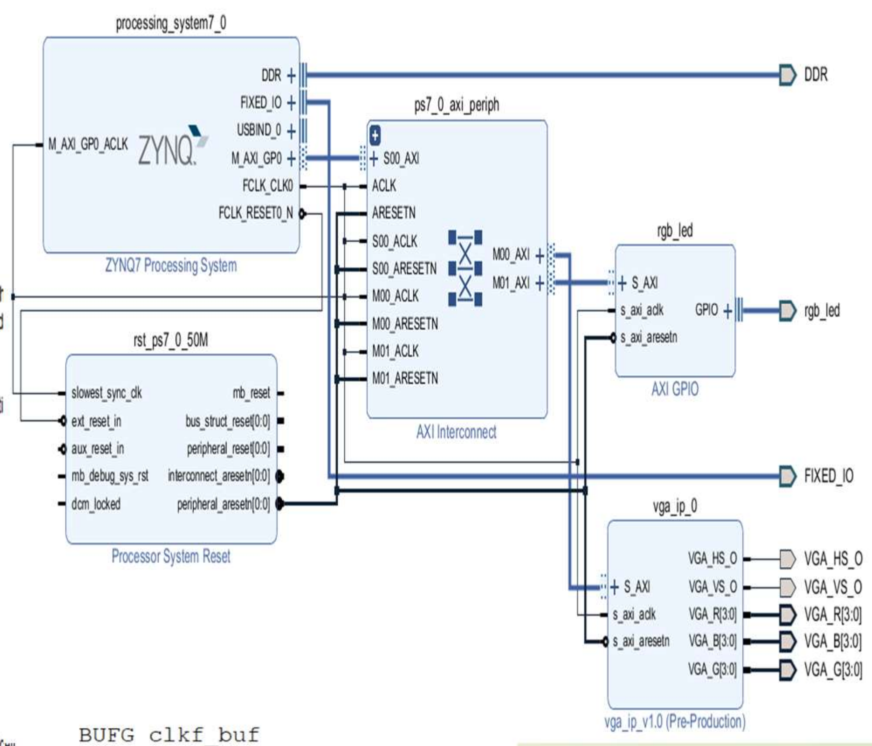

pixel 0,0
pixel 0,639
640 pixels per row are displayed during forward beam trace
Display Surface
Retrace - no information displayed during this time
pixel 479,0
pixel 479,639
Stable current ramp - information is displayed during this time
Current waveform through horizontal deflection coil
Total horizontal time
Horizontal display time
retrace time
HS
time
"front porch"
Horizontal sync signal sets retrace frequency
"back porch"

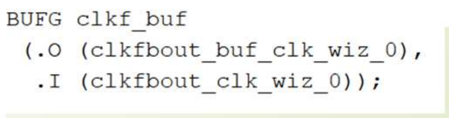
Figure 10. VGA horizontal synchronization.

VGA Current Result

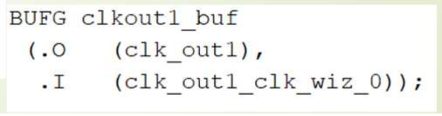


```
Implementation (2 errors, 1 critical warning)
  Place Design (1 critical warning)
    DRC (1 critical warning)
      Netlist (1 critical warning)
        Instance (1 critical warning)
          Invalid attribute value (1 critical warning)
            MMCM_ADV (1 critical warning)
              [DRC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





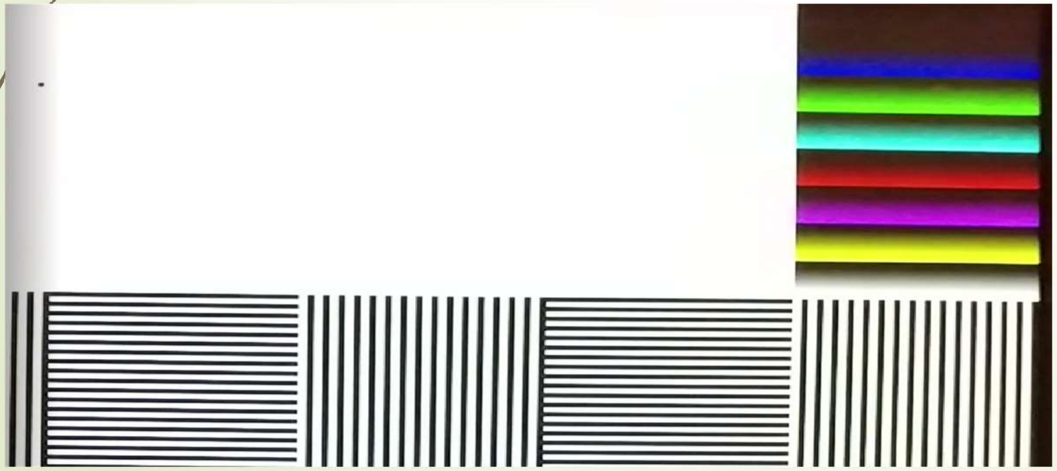
```
BUFG clkf_buf
 (.O (clkfbout_buf_clk_wiz_0),
  .I (clkfbout_clk_wiz_0));
```

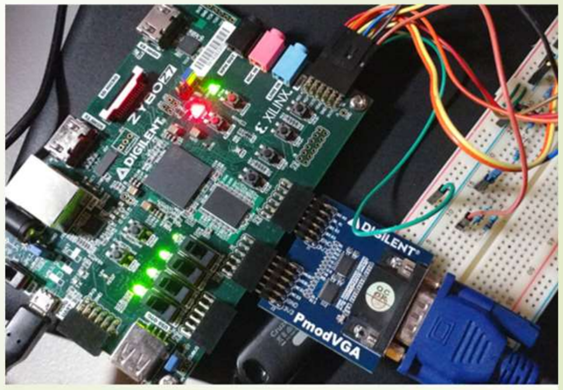


```
BUFG clkout1_buf
 (.O    (clk_out1),
  .I    (clk_out1_clk_wiz_0));
```

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