PmodAD1 Vivado SDK Project v2

Robert Bara
Robert.Bara@temple.edu

Summary

Lab 6 serves as an introduction to the PmodAD1 analog-to-digital converter (ADC) and pairing the Zybo board with the Diligent Discovery 2 Board using the Zynq SPI Peripherals and utilizing Vivado SDK and Waveforms. The lab is primarily a data collection lab to understand the fundamentals of the hardware but utilizes the ADC signal to send a "bar graph" like pattern across the LEDs of the Zybo board.

Introduction

The first part of the lab is to evaluate how the PmodAD1 can record voltage using SDK. This is accomplished by creating the proper hardware design/implementation and using the sample *AD1Pmod.c* program developed by Dr. Dennis Silage to link the Diligent Discovery 2 Board to the Zybo board and SDK through the PmodAD1. Upon generating a constant input voltage from the Arbitrary Waveform Generator (AWG) to PmodAD1's channel 1, a wave can be received using Waveforms, and SDK can record voltage from 0.1V to 1V. By collecting enough data points, a linear regression utility such as MATLAB or Excel can be used to determine the best fit linear slope and y-intercept of the PmodAD1's data. This will be used for further analysis later.

The Discovery 2's oscilloscope can be used to measure the Slave Select of the PmodAD1. Use this to examine the CS pulse which is an active low. From this the data rate and active duty cycle can be obtained. Furthermore, examine how commenting out the *xil_printf* and *sleep* C statements will affect the active duty cycle and data rate.

By creating a new Vivado project, an identical hardware design can be created, but with the addition of a single GPIO that connects LEDs in channel 1 and switches in channel 2. The sample PmodAD1 C program should then be edited to print ECE3622 within the SDK terminal when SW0 and SW1 are low. When SW0 is high but SW1 is low, the signal of a unipolar ramp at 0.1Hz, ranged from 0 to 1V should turn on the LEDs if their threshold voltages are exceeded, that is: LED0=0.2V, LED1=0.4V, LED2=0.6V, and LED3=0.8V. If the voltage is less than the threshold though, the LEDs will be off and nothing is outputted to SDK except for a string that will explain the event happening. Use the results obtained from part 1 to further analyze the accuracy of these results.

Finally, if SW1 is high and SW0 is low, the input unipolar ramp is followed by a pulse from 0 to 1V yielding 0.1Hz. By using multiple ADC samples, the symmetry of the signal can be estimated every 10seconds and SDK should output the symmetry percentage using:

$$100\ x\ N_{ramp}/(N_{ramp} + N_{pulse})$$

Record and analyze the results taken to determine why the gain obtained from the slope of the PmodAD1 is not unity.

Discussion

Hardware Design

There are two hardware designs to be used for this lab. The first one simply connects the Discovery 2 to the Zybo board by using the PmodAD1 and runs the sample program for data collection. The second hardware design is an extension of the first design, but maps LEDs and Switches as inputs and outputs. Before creating the hardware design, be sure to download the necessary constraint files and create a dummy file to import the *ZynqPmodVivado-Library* into Vivado. This is done by creating a file, selecting settings under project manager, and then adding the library under the *IP Defaults* tab.

Hardware Design 1:

Start the design by creating a project in Vivado that runs using Verilog and select the correct Zynq board that will be used. Create a block design in Vivado and adding the Zynq processing system IP. Run the connection automation, double click on the Zynq7 processor IP and select clock configuration, PL Fabric Clocks, and enable FCLK_CLK1 as a 30MHz clock signal to match the AD1 Pmod.

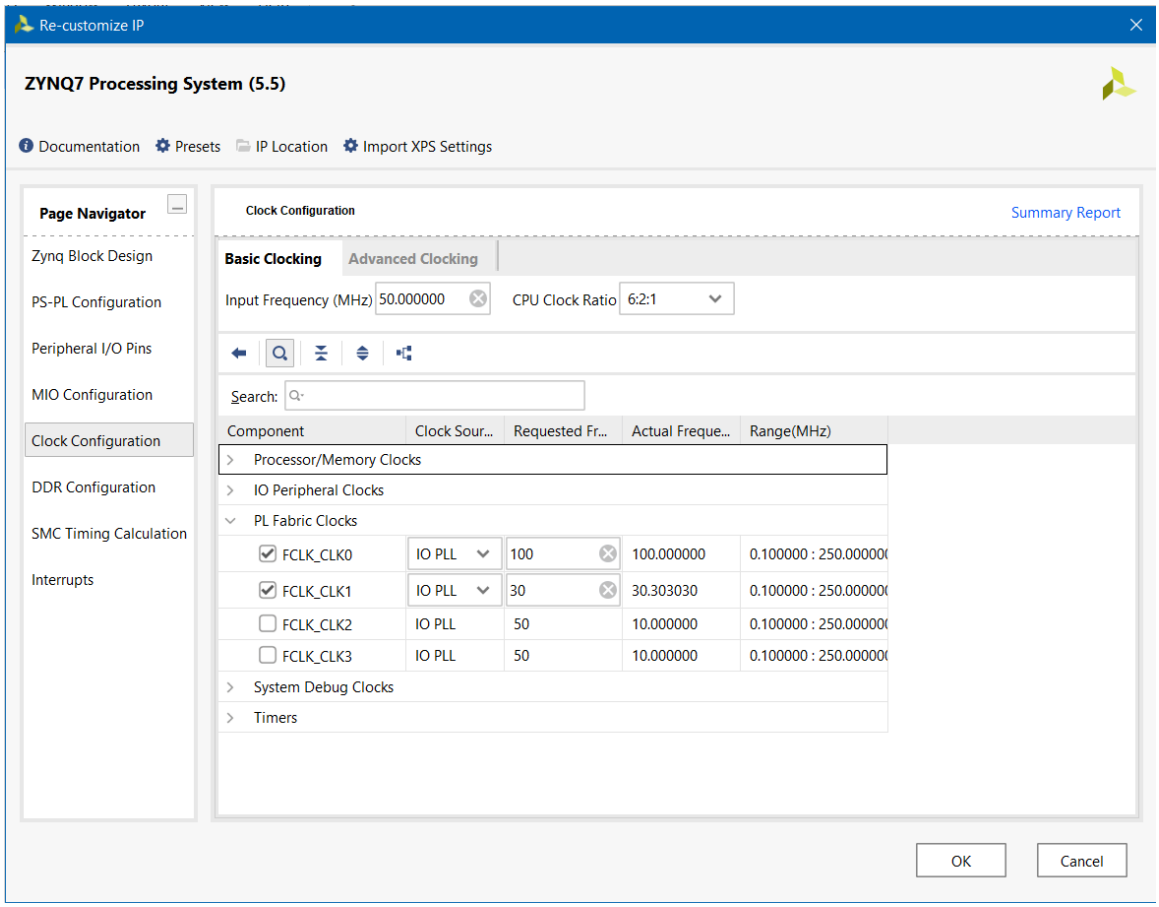


Figure 1. Customizing the Clock signal

Right click on constraints under sources and add the *AD1PmodJE.xdc* constraint file that was made available from Canvas. This will correctly map the custom PmodADC that was implemented through the *ZynqPmodVivado-Library* to SDK. Add the custom AD1Pmod

IP and run the connection automation. Finally, right click and make the following connections external on the AD1Pmod block: AD1dat1, AD1dat2, AD1sync, and AD1sclk. Lastly, connect the AD1clk to the FCLK_CLK1 input that was configured earlier within the Zynq Processor IP. Verify the design, create an HDL wrapper, and generate a bitstream. Upon completion, export to hardware and launch SDK, be sure to include the bitstream. Note: the bitstream may take a long time to generate and ignore the negative clock warning that Vivado will output, it is simply a warning due to the preproduction PmodAD1 that is being used. The final block diagram is as follows:

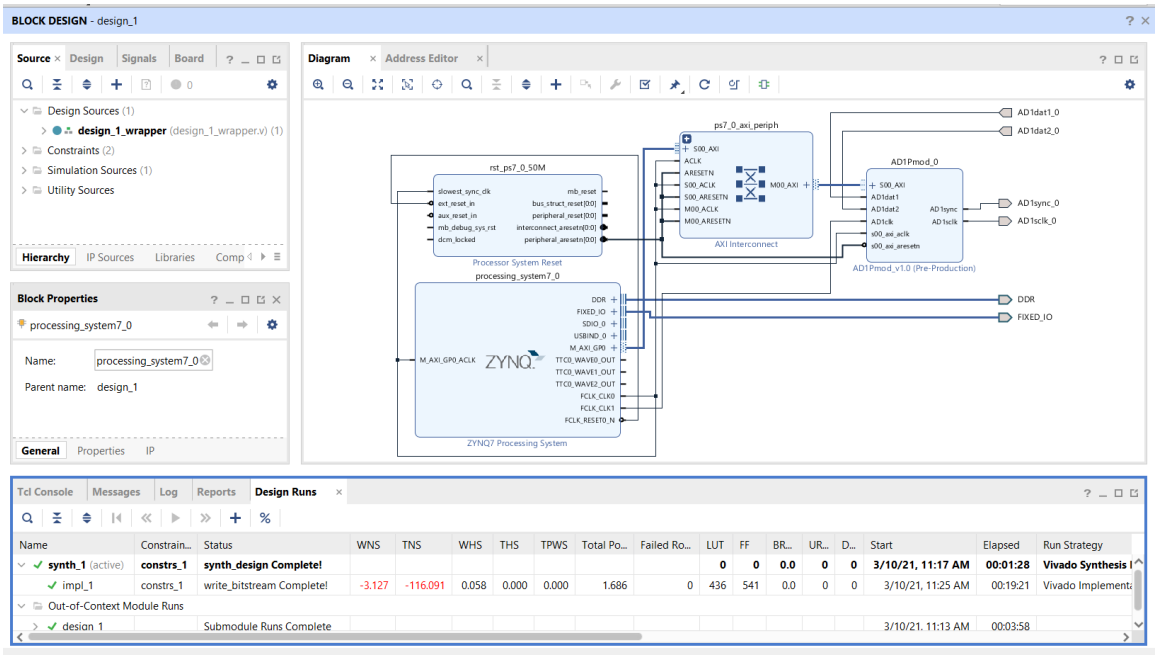


Figure 2. PART 1 Block Diagram

Hardware Design 2

Create a separate Vivado project, for the sake of simplicity/to avoid errors. Follow the same steps as the hardware design but before verifying the design, and creating the wrapper and bitstream, add 1 AXI_GPIO that will wire the Zybo board's Switches as an input and LEDs as outputs by using two channels on the GPIO. Verify the design, create the wrapper, and generate the bitstream as usual. Export the design to hardware and launch SDK.

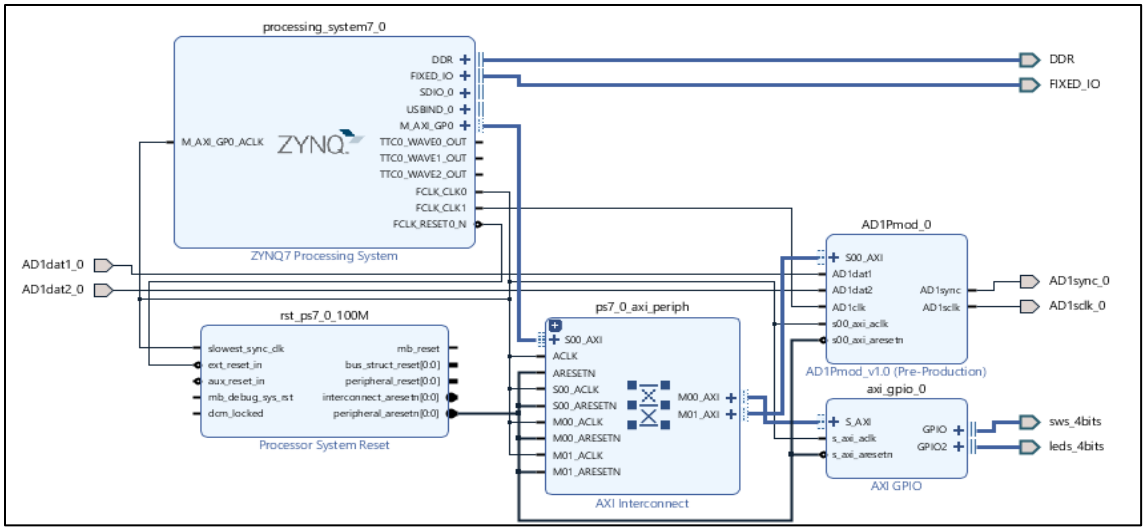


Figure 3. Block Diagram for PART II

Hardware Implementation

Begin by downloading Waveforms to use the Discovery 2 board with the Zybo board. Connect the hydro connector so that the 3-dot logo is faced down on the lower right side, while writing is displayed from the top, and connect the connector into the 12-pin JE Pmod Jack on the Zybo board. Ensure that the connection is being made so that VCC pins are connected to the leftmost pins within the jack, with GND right next to it, and the leftover pins represent the 8signals. Next Connect the A pin Connector with the 3-dot indicator facing up to the PmodAD1 PCB, ensuring that the 3-dot indicator lines up with the CS pin. Use the jumpers on the Discovery 2 and properly connect the them based upon the documentation provided from the kit or the Discovery 2 PDF:

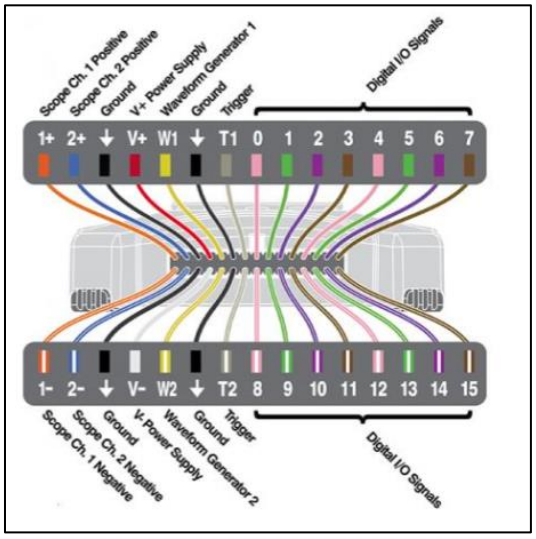


Figure 4. Discovery 2 Jumper connections

For this lab, the Waveform generator, Oscilloscope, and GND connections will be used. GND should be connected to the GND pin of the AD1, while the scope or waveform

jumpers should be connected to the leftmost pin. An example of the Waveform generator and proper connections is as follows:

Figure 5. Zybo to Discovery 2 connections

Software Design

This section will cover both the C code written in SDK and the signals/scope readings implemented by Waveforms

C Code

The sample program *AD1Pmod.c* runs as follows and serves as a template for the C code written in the later tasks.

Sample AD1Pmod.C

Initialization

The program begins by defining the necessary headers/libraries to be utilized by the program, as well as obtaining the AD1Pmod Address Editors from Vivado to properly map each input and output:

```
//AD1Pmod.c  PmodAD1 ADC example  ECE3622  c2019 Dennis Silage

#include "xparameters.h"
#include "xil_io.h"
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include <sleep.h>

//AD1Pmod from Address Editor in Vivado
#define AD1acq  0x43C00000  //AD1 acquisition
#define AD1dav  0x43C00004  //AD1 data available
#define AD1dat1 0x43C00008  //AD1 channel 1 data
#define AD1dat2 0x43C0000C  //AD1 channel 2 data
```

Figure 6. Initialization of AD1Pmod.C

Main

The only subroutine is the Main function for this program. The program begins by defining the ADC data that will be read, as well as ADC channel 1 and ADC channel 2. Data collection is initialized as 0 and the program begins to run using a while loop.

```
int main(void)
{
    int adcdav;             //ADC data available
    int adcdat1;            //ADC channel 1 data
    int adcdat2;            //ADC channel 2 data


    xil_printf("\n\rStarting AD1 Pmod demo test...\n");
    Xil_Out32(AD1acq,0);
    sleep(5);
```

Figure 7. Declarations/initialization

By using a nested while loop, the program checks if data is available from the Discovery 2's input wave. If there is, channel 1 begins to read, otherwise it will stop reading data. The data is then simply printed across the SDK terminal using the appropriate delay.

```
    while (1)
    {
        Xil_Out32(AD1acq,0);            //ADC stop acquire
        adcdav=Xil_In32(AD1dav);        //ADC available?
        while(adcdav==1)
                adcdav=Xil_In32(AD1dav);

        while(1)
            {
                Xil_Out32(AD1acq,1);    //ADC acquire
                while(adcdav==0)
                    adcdav=Xil_In32(AD1dav); //ADC data?
                Xil_Out32(AD1acq,0);    //ADC stop acquire
                adcdat1=Xil_In32(AD1dat1);
                adcdat2=Xil_In32(AD1dat2);
                while(adcdav==1)
                    adcdav=Xil_In32(AD1dav); //ADC reset?
                xil_printf("Ch1:  %d\n", adcdat1);
                sleep(1);
                }
    }
}
```

Figure 8. ADC data collection

Modified Program for LED and Switches

The modification to this sample program primarily comes within the initialization and while loop within the main. To include switches as inputs and leds as outputs, additional headers files are declared such as xgpio.h and using the address editor in vivado, switches were mapped to channel 1 while leds were mapped to channel 2.

```
//Lab6 PmodAD1 modification Robert Bara ECE3622 2021
#include "xparameters.h"
#include "xil_io.h"
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include <sleep.h>
#include "xgpio.h"
//GPIO definitions
#define LEDS_DEVICE_ID XPAR_AXI_GPIO_0_DEVICE_ID //GPIO device connected to LED
#define SW_DEVICE_ID XPAR_AXI_GPIO_0_DEVICE_ID //GPIO device connected to SW
XGpio LEDInst, SWInst;
//Defining each channel
#define LED_CHANNEL 2
#define SW_CHANNEL 1
//AD1Pmod from Address Editor in Vivado
#define AD1acq  0x43C00000  //AD1 acquisition
#define AD1dav  0x43C00004  //AD1 data available
#define AD1dat1 0x43C00008  //AD1 channel 1 data
```

Figure 9. Headers and Definitions

The main program then initializes the LEDs and Switches and sets them as inputs/outputs. Once this is done, the AD1 gets initialized and the program begins to run and check for data written using Waveforms to generate various input signals, depending on the task.

```
//-----------------------------------------------------
// INITIALIZE THE PERIPHERALS & SET DIRECTIONS OF GPIO
//-----------------------------------------------------
// Initialise LEDs
status = XGpio_Initialize(&LEDInst, LEDS_DEVICE_ID);
if(status != XST_SUCCESS) return XST_FAILURE;
// Initialise Push Buttons
status = XGpio_Initialize(&SWInst, SW_DEVICE_ID);
if(status != XST_SUCCESS) return XST_FAILURE;
// Set LEDs direction to outputs
XGpio_SetDataDirection(&LEDInst, 2, 0x00);
// Set all switches direction to inputs
XGpio_SetDataDirection(&SWInst, 1, 0xFF);
led=0;
XGpio_DiscreteWrite(&LEDInst, 2, 0); //initialize LEDs as off

//-----------------------------------------------------
//RECORDING DATA AND PERFORMING TASKS
//-----------------------------------------------------
xil_printf("\n\rStarting AD1 Pmod demo test...\n");
Xil_Out32(AD1acq,0);
sleep(5);
```

Figure 10. Initialzing input SW and LEDs

In the while loop, switches and buttons get checked and read to perform tasks based upon the switch inputs:

```
        while(adcdav==1)
                adcdav=Xil_In32(AD1dav);

        while(1)
            {   //read sw and AD1 input
                sw = XGpio_DiscreteRead(&SWInst, 1);
                Xil_Out32(AD1acq,1);    //ADC acquire
                while(adcdav==0)
                    adcdav=Xil_In32(AD1dav); //ADC data?
                Xil_Out32(AD1acq,0);    //ADC stop acquire
                adcdat1=Xil_In32(AD1dat1);
                while(adcdav==1)
                    adcdav=Xil_In32(AD1dav); //ADC reset?
```

Figure 11. Initializing AD1

The tasks were written using simple if statements to determine the button inputs. Sw's off lead to a single printing every 5 seconds, while sw0 high will lead to an LED bar graph effect. Based on the data collected within part1, I found the average of the computer units based upon the 21 points I gathered for 0.2V, 0.4V, 0.6V, 0.8V, and 1V, and used that to determine my threshold voltages for each LED.

```
    adcdav=Xil_In32(AD1dav); //ADC reset
if(sw==0)
{   //when switches are low, print every 5 seconds
    xil_printf(" ECE3622 Lab 6 \n");
    sleep(5);   //delay of 5seconds
}
if(sw==0b0001)
{   //LEDs are based upon the average values collected from PT1
    xil_printf("Sw0 is high\t%d\n",adcdat1);
    if(adcdat1>=254) led=0b0001;    //threshold of 0.2V
    if(adcdat1>=505) led=0b0011;    //threshold of 0.4V
    if(adcdat1>=756) led=0b0111;    //threshold of 0.6V
    if(adcdat1>=1007)led=0b1111;    //threshold of 0.8V
    //outputting to LEDs
    XGpio_DiscreteWrite(&LEDInst, 2, led);
}
```

Figure 12. Tasks 4A and 4B

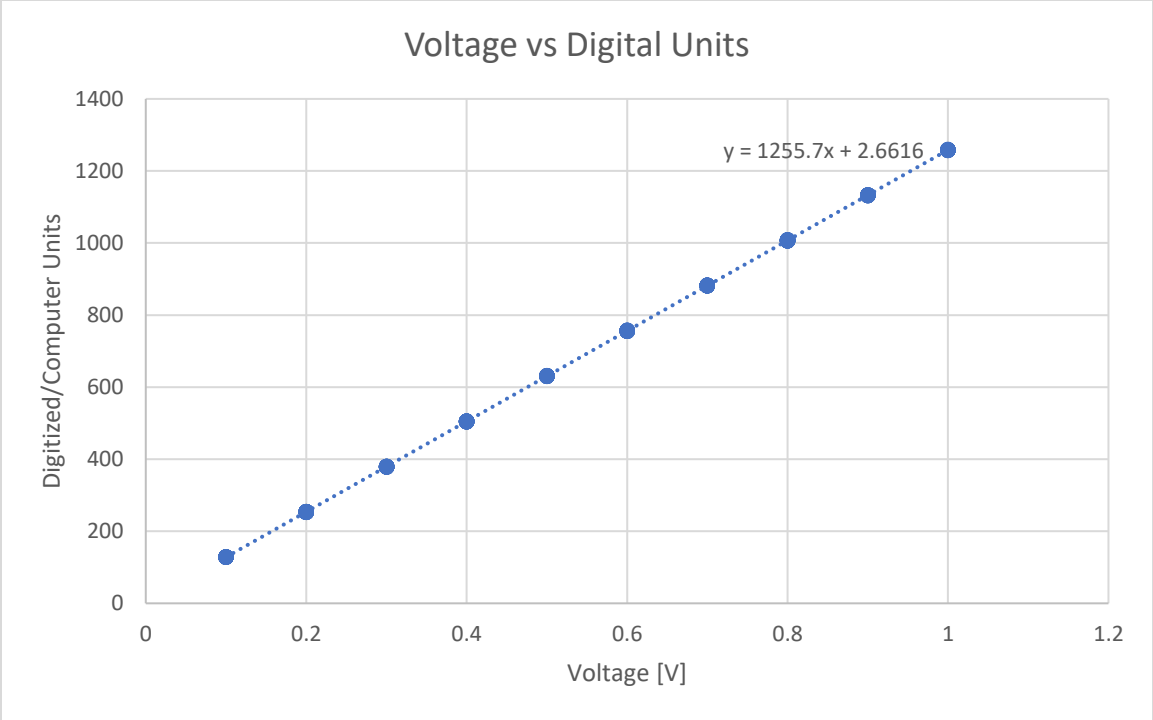
I used the 1V calculation to determine when my signal is a rising ramp less than 1V or will generate a pulse when it is not rising and calculate my symmetry percentage to be outputted to SDK terminal. Given that the equation to calculate symmetry is $100\ x\ N_{ramp}/(N_{ramp} + N_{pulse})$, I generated two counters, 1 to count the number of pulses, and another to count the number of ramps, this was used to calculate my symmetry percentage.

```
if(sw==0b0010)
{   data=adcdat1;
    //Ramp is less than 1V, it is the signal increasing
    if(adcdat1<1258)
    {
        ramp=data;
        countR++;

    }
    //The pulse will be defined by the rest of the signal
    if(adcdat1>=1258)
    {
        pulse=data;
        countP++;

    }
    //calculating symmetry percentage
    sym=100*(countR/(countR+countP));
    printf("ch1 %d\t ramp:%d\tcountR %f\tcountP %f\t pulse:%d\t sym:%f\n", adcdat1, ramp,countR,countP, pulse, sym);
}
```

Figure 13. SW1 high, calculating symmetry

Verification

For task 1, I used a DC waveform with a min of 100mV to max of 1V and used the offset to collect 21 datapoints per 0.1 increments voltage. I copied the data over to Excel and transferred the data into a scatter plot with the obtained line of best fit as follows. I believe the gain of the slope is not unity because the Pmod is quantizing the data experimentally, giving my values a tolerance typically of ±2%, needless to say it's based upon noise and rounding errors, therefore 1 electron will not fully match 1 computer unit. The equation can become more accurate by collecting more points. Furthermore, by increasing the number of bits the AD1 can collect, the gain could also be closer to unity. Another way around this is to create a unity gain buffer which could be accomplished by with the addition of an operational amplifier:



When reading the oscilloscope of the C program as originally intended, the oscilloscope reads the following data:

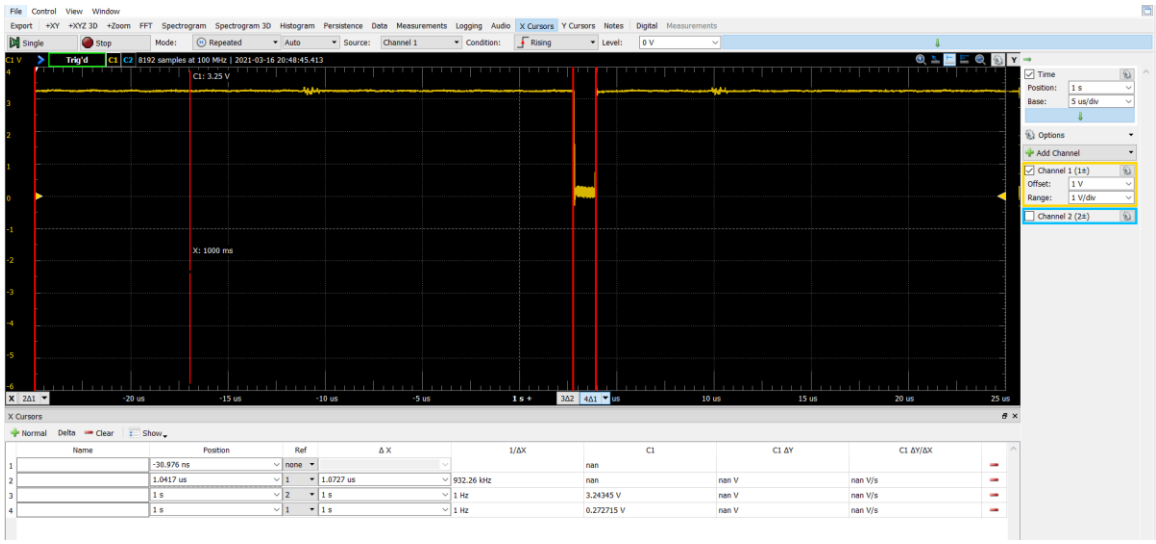


Figure 14. Scope reading with Printf/Sleep statements

When taking out the printf and sleep statements in C, the oscilloscope read as follows:

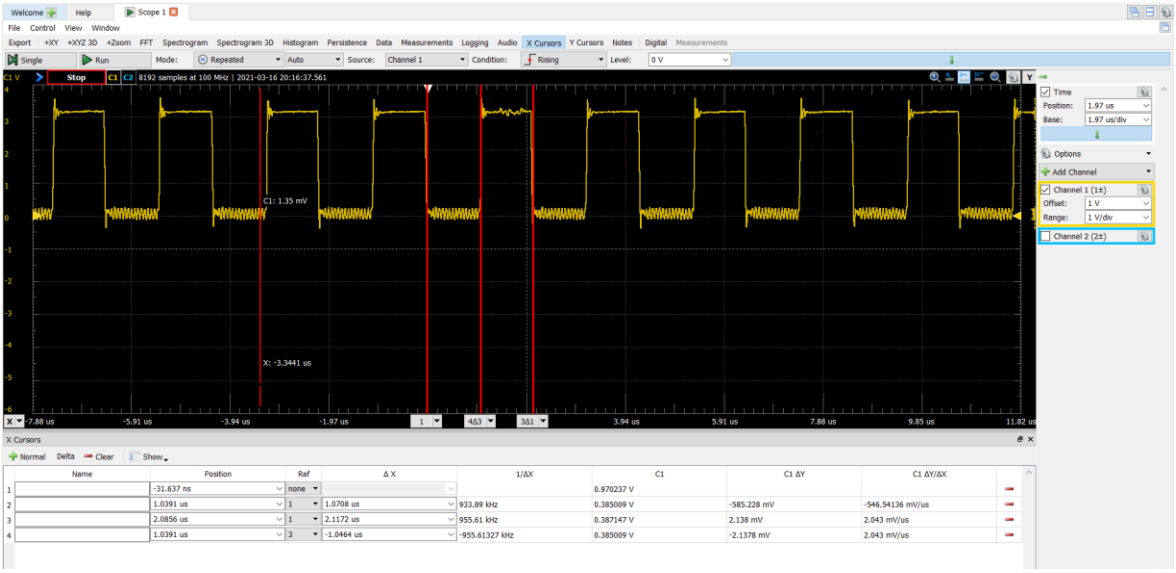


Figure 15. Scope reading without statements

Plugging these values into the Excel sheet that was used to find linear regression, the duty cycles and data rate read as follows:

	With Printf/Sleep	Without Printf/Sleep	%Difference
Time Between Peaks:	1	2.1172E-06	199%
Active low:	1.0727E-06	1.0708E-06	0.17728%
Active High:	1	1.0464E-06	199%
Duty Cycle:	1.0727E-06	0.505762328	200%
Data Rate:	1.00E+00	4.72E+05	200%

By taking the sleep and printf statements out, the program will collect data at a much faster rate, which is clearly shown in the second reading compared to the first, due to the first readings prolonged active high. Also, it is quite astonishing how high of a percentage difference there is between the time it takes with the sleep/print statements vs without, but this also makes sense given that I am comparing seconds to microseconds.

In terms of the tasks determined by the switches, the screenshot below demonstrates when SW1 is high it is calculating the symmetry. Switching all switches off simply prints ECE3622 Lab 6 every 5 seconds.

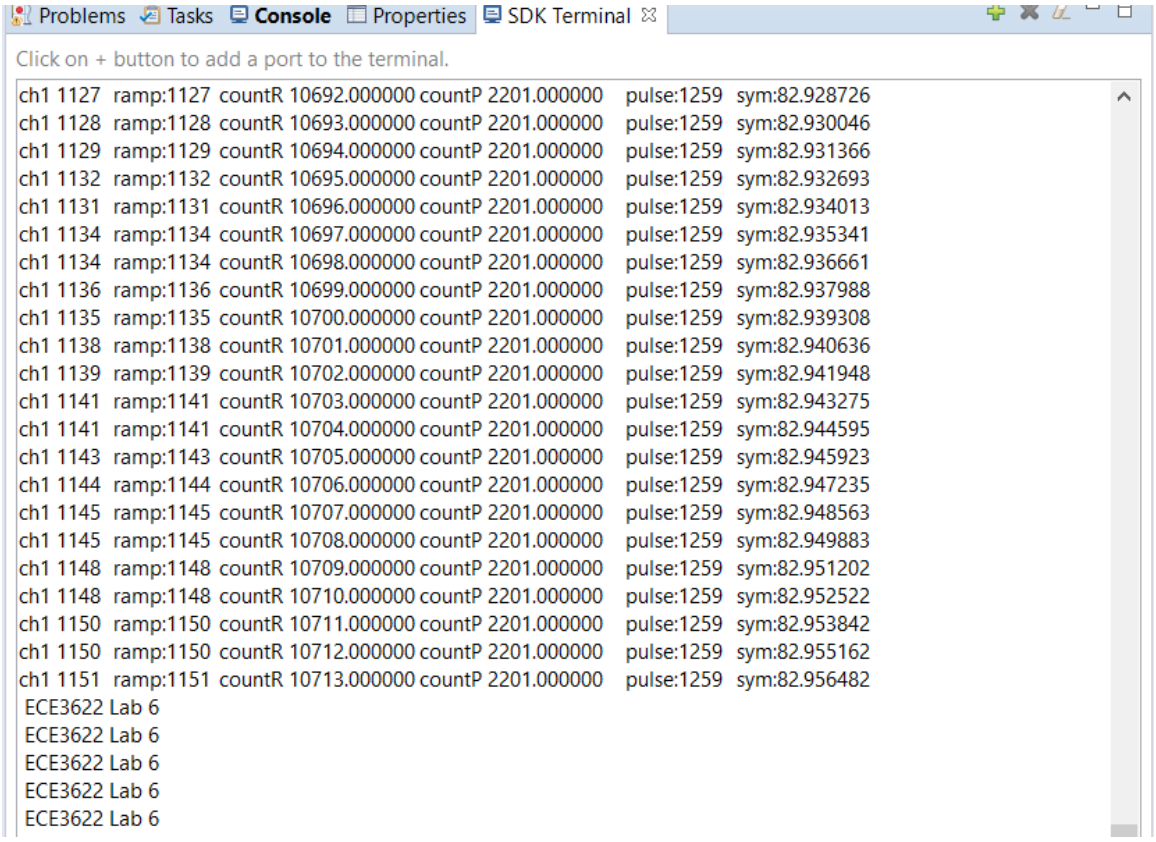


Figure 16. Sw off and Sw1 high SDK outputs

The symmetry calculation could be flawed within a number of ways, because of the way I implemented my algorithm, it's accuracy increases based on the data sample, that is to say, the number of pulses/ramps will balance out the ratio which calculates the symmetry. The error within this, is changing symmetry in waveforms, will still properly calculate the symmetry, however the new number of ramps/pulses need to outweigh the amount of data collected from the previous sample. I had tried several ways to work around this but given the amount of time I had to spend working out my hardware setup with everyone since something was wrong internally with my original Zybo board, I decided to just state with the error since I do not want to fall behind in lab. One work around I did to help my error is to implement a counter reset when the switches are all low, this helps because symmetry can be changed/calculated much faster if the counters are reset each time. My video will discuss/show this further.

In terms of the LEDs, here is a simple example of the threshold voltage turning on LED 2

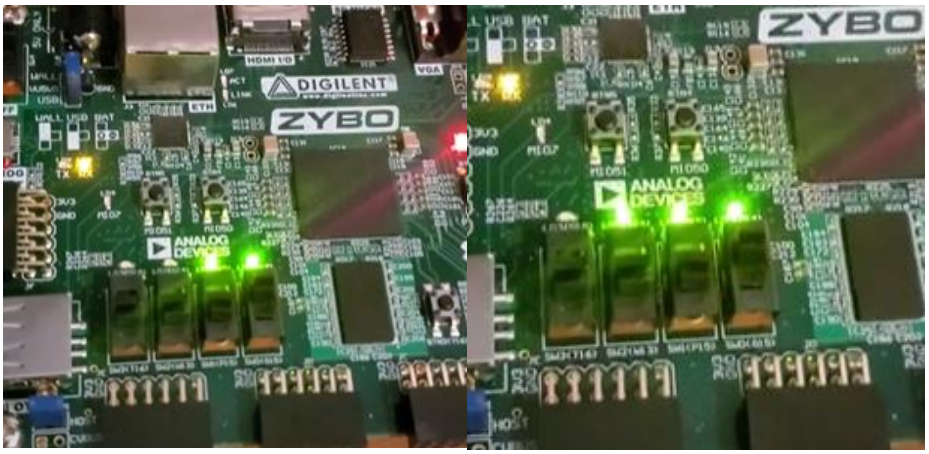


Figure 17A and 17B. Turning on LEDs based on threshold voltage

Video Link:

https://youtu.be/et0vTJerdgU

Conclusion

Despite the hardware issues I was having until Tuesday, this lab was not that difficult to grasp in terms of introducing how to use waveforms and the oscilloscope. I think the hardest part for me was understanding how the data I was collecting was related, and figuring out the symmetry percentage algorithm, which works but definitely has it's flaws, that possibly with more time and assistance, I could minimize.

Appendix

```
//Lab6 PmodAD1 modification Robert Bara ECE3622  c2021

#include "xparameters.h"
#include "xil_io.h"
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include <sleep.h>
#include "xgpio.h"
//GPIO definitions
#define LEDS_DEVICE_ID XPAR_AXI_GPIO_0_DEVICE_ID //GPIO device connected to
LED
#define SW_DEVICE_ID XPAR_AXI_GPIO_0_DEVICE_ID //GPIO device connected to SW
XGpio LEDInst, SWInst;
//Defining each channel
#define LED_CHANNEL 2
#define SW_CHANNEL 1
//AD1Pmod from Address Editor in Vivado
#define AD1acq      0x43C00000   //AD1 acquisition
#define AD1dav      0x43C00004   //AD1 data available
#define AD1dat1     0x43C00008   //AD1 channel 1 data

int main(void)
{
       int adcdav;                      //ADC data available
       int adcdat1;              //ADC channel 1 data
       int sw, led;
       int ramp=0, pulse=0,data=adcdat1;
       float sym;
       float countR=0,countP=0;
       int status;
       //-----------------------------------------------------
       // INITIALIZE THE PERIPHERALS & SET DIRECTIONS OF GPIO
       //-----------------------------------------------------
       // Initialise LEDs
       status = XGpio_Initialize(&LEDInst, LEDS_DEVICE_ID);
       if(status != XST_SUCCESS) return XST_FAILURE;
       // Initialise Push Buttons
       status = XGpio_Initialize(&SWInst, SW_DEVICE_ID);
       if(status != XST_SUCCESS) return XST_FAILURE;
       // Set LEDs direction to outputs
       XGpio_SetDataDirection(&LEDInst, 2, 0x00);
       // Set all switches direction to inputs
       XGpio_SetDataDirection(&SWInst, 1, 0xFF);
       led=0;
       XGpio_DiscreteWrite(&LEDInst, 2, 0); //initialize LEDs as off

       //-----------------------------------------------------
       //RECORDING DATA AND PERFORMING TASKS
       //-----------------------------------------------------
       xil_printf("\n\rStarting AD1 Pmod demo test...\n");
       Xil_Out32(AD1acq,0);
       sleep(5);
```

```
	while (1)
	{
		Xil_Out32(AD1acq,0);				//ADC stop acquire
		adcdav=Xil_In32(AD1dav);		//ADC available?
		while(adcdav==1)
				adcdav=Xil_In32(AD1dav);

		while(1)
			{
				//read sw and AD1 input
				sw = XGpio_DiscreteRead(&SWInst, 1);
				Xil_Out32(AD1acq,1);	//ADC acquire
				while(adcdav==0)
					adcdav=Xil_In32(AD1dav); //ADC data?
				Xil_Out32(AD1acq,0);	//ADC stop acquire
				adcdat1=Xil_In32(AD1dat1);
				while(adcdav==1)
					adcdav=Xil_In32(AD1dav); //ADC reset
				if(sw==0)
				{	//when switches are low, print every 5
seconds
					countR=0;
					countP=0;
					xil_printf(" ECE3622 Lab 6 \n");
					sleep(5);	//delay of 5seconds
				}
				if(sw==0b0001)
				{	//LEDs are based upon the average values
collected from PT1
					xil_printf("Sw0 is high\t%d\n",adcdat1);
					if(adcdat1>=254) led=0b0001;	//threshold
of 0.2V
					if(adcdat1>=505) led=0b0011;	//threshold
of 0.4V
					if(adcdat1>=756) led=0b0111;	//threshold
of 0.6V
					if(adcdat1>=1007)led=0b1111;	//threshold
of 0.8V
					//outputting to LEDs
					XGpio_DiscreteWrite(&LEDInst, 2, led);
				}
				if(sw==0b0010)
				{	data=adcdat1;
					//Ramp is less than 1V, it is the signal
increasing
					if(adcdat1<1258)
					{
						ramp=data;
						countR++;

					}
					//The pulse will be defined by the rest of
the signal
					if(adcdat1>=1258)
					{
```

```
                                        pulse=data;
                                        countP++;

                                    }
                                    //calculating symmetry percentage
                                    sym=100*(countR/(countR+countP));
                                    printf("ch1 %d\t ramp:%d\tcountR %f\tcountP
%f\t pulse:%d\t sym:%f\n", adcdat1, ramp,countR,countP, pulse, sym);
                                }

        }
}
}
```