REQUIREMENTS NOT MET

N/A. All Requirements are met in this lab.

PROBLEMS ENCOUNTERED

Some problems encountered with part 1 and 3 of the lab includes the pre-lab question portion where it had to be researched heavily from the manuals and lecture resources to be completed. The problems encountered with part 2 was with figuring out the configuration needed to be done for the tcc0 timer and ADC interrupt. For part 3, the major issue was figuring out how to setup the print syntax so that the serial output is display correctly. This required the assistance of PI in explaining the necessary steps to be done. For part 4, it was completed with relatively little issues, but more problems arose in part 5 as I could not get the graph from the backpack to work, requiring me to redesign the necessary code for it to function property.

FUTURE WORK/APPLICATIONS

This lab was a good introduction into the implementation and function of ADC and the difference of direct and alternating voltages. This lab is to be the expansion of more complex assembly programs, able to give the users' another way to interact with the microprocessor, enabling new hardware running on a different power source to be combined. With the power source type are no longer restricting the type of hardware being used, the user can now implement a new functions and techniques that can simplify user's task or create new uses. Like the subroutine, the way I program is now changed to be inclusive of ADV for more capability of my programs. Having the ability to use ADC allows for me to expand my hardware access to devices. If given more time, the code of the lab could have been more organized and have a much neater layout to further reduce the likelihood of mistakes and further enhance the understanding of the program. With more time a more compacted or efficient communication can be implemented. Additionally, I could have used better instructions to make the code run more efficiently or learn to write more complex programs.

PRE-LAB EXERCISES

Part 1: USING THE ADC SYSTEM

i. Why must we use the ADCA module as opposed to the ADCB module?

We use the ADCA module as opposed to the ADCB module because port B is not accessible with the robotic backpack attached to the microprocessor.

ii. Would it be possible to use any other ADC configurations such as single-ended, differential with gain, etc. with the current pinout of the OOTB Analog Backpack?

Yes, it is possible to use other ADC configurations such as single-ended, differential with gain, etc. with the current pinout of the OOTB Analog Backpack.

iii. What would the main benefit be of using an ADC system with 12-bit resolution, rather than an ADC system with 8-bit resolution? Would there be any reason to use 8-bit resolution instead of 12-bit results? If so, explain.

The main benefit be of using an ADC system with 12-bit resolution, rather than an ADC system with 8-bit resolution because an 8-bit system will not be able to resolve the smaller, finer changes in the input signal than a 12-bit system; therefore, the sampled signal will not be as accurate a representation of the true signal where greater accuracy can be obtain with more bits to be utilized. The reason to use 8-bit resolution instead of 12-bit results is, if the accuracy of the conversion is not of great importance, to reduce cost and complexity since a 12-bit usually require more work to implement and is of greater expensive. Another thing to remember is that the higher-resolution ADC will generally have a lower maximum clock rate which might be an issue in certain systems.

Part 3: OUTPUTTING SAMPLED DATA WITH UART

iv. What is the decimal voltage value that is equivalent to a 12-bit signed result of 0x073?
Vref=2.5 ADC range= 2^{12}-1=4095=[-2048,2047] slope=(2.5-(-2.5))/4095=5/4095
y=mx+b -> 2.5=(5/4095)(2047)+b -> b=0
0x073 -> 115 y=mx+b=(5/4095)(115)+0=0.14V
The decimal voltage value that is equivalent to a 12-bit signed result of 0x073 is 0.14V.

PSEUDOCODE/FLOWCHARTS

SECTION X (1, 2, etc.)

Part 1: USING THE ADC SYSTEM

Johnny Li
Lab 7

Lab 7 Part 1 Flow Chart

adc_init();
- 12 bit signed, right-adjusted ADCA
- Normal, not freerun mode
- Use 2.5 Vref
- Enable after all ADC initializations, not start conversion.
- Select MUXCTRL

Main();
adc_init();
while(1){ Start conversion.
-Check if finish.
-Record result.}
return 0;

Part 2: SAMPLING AT A SPECIFIC RATE USING EVENTS

Lab 7 Part 2 Flow Chart

adc_init();
- Add- ADC interrupt to triggered when a conversion is complete.
- Conversion start when Event Channel 0.
- PMIE and sei.

tcc0_init();
- Set period at 2Hz.
- Set Timer Mode
- Set prescaler.
- Set event trigger on Event Channel 0.

Main();
- Add- PORTD LED initialize
- tcc0_init();

IRS;	• Store value to check voltage. • Toggle BLUE LED.

Part 3: OUTPUTTING SAMPLED DATA WITH UART

	Lab 7 Part 3 Flowchart
TSR;	• Update global variable result and flag.
USART0;	• Use previous lab USART. • Baud rate at 128kbps.
Main;	• clear global flag. • output voltage to serial terminal.

Part 4: VISUALIZING THE ADC CONVERSIONS

	Lab 7 Part 4 Flowchart
Tiva init;	Change Period for 100Hz.
Main;	• Configure ADC to play on sequence with two bytes, one channel, run 16 bit signed value at 1kHz bytes.

Part 5: SWITCHING BETWEEN MULTIPLE INPUTS

Lab 7 Part 5 Flow chart

Usartdo; • Enable interrupts for the UART receive

ISR(Usart); • Receive input character
• Trigger global flag receiver.

Main; • check if the character is a 1 → use cd measurements.
- If the character is a 2 → use 53 analogy signal generated by waveform.
- If any other character do nothing.

PROGRAM CODE

SECTION X (1, 2, etc.)

Part 1: USING THE ADC SYSTEM

```
/*********************************************************************************
;Lab 7 Part 1
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: USING THE ADC SYSTEM
;*********************************INCLUDES*************************************/

#include <avr/io.h>

int main(void){
	//Initialize
	adc_init();

	//Test ADC value
	int16_t test = 0x00;

	//Loop conversion
	while (1){
		//Start ADC
		ADCA_CH0_CTRL |= ADC_CH_START_bm;

		//Stall till conversion is complete
		while( !(ADCA_CH0_INTFLAGS & ADC_CH_CHIF_bm) );

		//Store value to check
		test = ADCA_CH0_RES;

		//Reset
		ADCA_CH0_INTFLAGS = ADC_CH_CHIF_bm;
	}
	return 0;
}
```

Part 2: SAMPLING AT A SPECIFIC RATE USING EVENTS

```
/*********************************************************************************
;Lab 7 Part 2
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: SAMPLING AT A SPECIFIC RATE USING EVENTS
;*********************************INCLUDES*************************************/
#include <avr/io.h>
#include <avr/interrupt.h>

#define BLUE_PWM_LED PIN6_bm

int main(void){
	//Initialize
	PORTD_OUTSET = 0x00; //LED initially off
	PORTD_DIRSET = BLUE_PWM_LED;	//Set output
```

```
	adc_init();
	tcc0_init();

	//Loop conversion
	while (1){}

	return 0;
}
```

Part 3: OUTPUTTING SAMPLED DATA WITH UART

```
/*************************************************************************
;Lab 7 Part 3
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: OUTPUTTING SAMPLED DATA WITH UART
;*********************************INCLUDES*************************************/
#include <avr/io.h>
#include <avr/interrupt.h>
#include "USART.h"

#define BLUE_PWM_LED PIN6_bm
#define m (5.0/4095) //slope
#define b 0 //y-intercept

//Global Variables to store output
int16_t test = 0;    //Digit output
volatile int global_flag = 0;      //Global flag of interrupt

int main(void){
	//Initialize
	PORTD_OUTSET = BLUE_PWM_LED;      //LED initially off
	PORTD_DIRSET = BLUE_PWM_LED;      //Set output

	adc_init();
	tcc0_init();
	usartd0_init();

	//Storage of output characters
	char out[18];

	//Loop conversion
	while (1){
			//Output voltage when global flag gets set
			while (global_flag){
				//Reset
				global_flag = 0;

				float volt = (float) ((m*test)+b); //Decimal voltage value

				//Check for +/-
				if (test<0) { //Negative
					out[0] = '-';
				}
				else { //Positive
					out[0] = '+';
				}

				out[1] = ' '; //Space
				int index = 0;       //Indexing variable
```

```
//Output decimal value
for (int i = 0; i < 3; i++){       //3 digit
       if(volt<0){   //Convert negative to positive
              volt=volt*-1;
       }
       int temp = (int) volt;      //Int1 = (int) Pi = 3
       out[2+index] = temp+'0';    //1 digit

       volt = 10*(volt-temp);      //Pi2 = 10*(Pi - Int1) = 1.4159

       if (i==0){
              out[3] = '.'; //Decimal point
              index++;      //Skip index 3
       }

       index++;      //increment
}

//Syntax
out[6] = ' ';
out[7] = 'V';
out[8] = ' ';
out[9] = '(';
out[10] = '0';
out[11] = 'x';


       // Output the ADC to the serial terminal
int hex3 = (uint8_t)(test>>8)%16;
if(hex3 <= 9){
       out[12] = ((uint8_t) hex3)+'0';    //First hex
}
else if(hex3 > 9){
       switch(hex3){
              case 10: out[12] = 'A';     //First hex
              break;
              case 11: out[12] = 'B';     //First hex
              break;
              case 12: out[12] = 'C';     //First hex
              break;
              case 13: out[12] = 'D';     //First hex
              break;
              case 14: out[12] = 'E';     //First hex
              break;
              case 15: out[12] = 'F';     //First hex
              break;
       }
}
else{
       out[12] = '0';
}

// Output the ADC to the serial terminal
int hex2 = (uint8_t)(test>>4)%16;
if(hex2 <= 9){
       out[13] = ((uint8_t) hex2)+'0';    //First hex
}
else if(hex2 > 9){
       switch(hex2){
              case 10: out[13] = 'A';     //First hex
              break;
```

```
                                        case 11: out[13] = 'B';     //First hex
                                        break;
                                        case 12: out[13] = 'C';     //First hex
                                        break;
                                        case 13: out[13] = 'D';     //First hex
                                        break;
                                        case 14: out[13] = 'E';     //First hex
                                        break;
                                        case 15: out[13] = 'F';     //First hex
                                        break;
                                }
                        }
                        else{
                                out[13] = '0';
                        }

                        // Output the ADC to the serial terminal
                        int hex = (uint8_t)(test)%16;
                        if(hex <= 9){
                                out[14] = ((uint8_t) hex)+'0';     //First hex
                        }
                        else if(hex > 9 ){
                                switch(hex){
                                        case 10: out[14] = 'A';     //First hex
                                        break;
                                        case 11: out[14] = 'B';     //First hex
                                        break;
                                        case 12: out[14] = 'C';     //First hex
                                        break;
                                        case 13: out[14] = 'D';     //First hex
                                        break;
                                        case 14: out[14] = 'E';     //First hex
                                        break;
                                        case 15: out[14] = 'F';     //First hex
                                        break;
                                }
                        }
                        else{
                                out[14] = '0';
                        }

                        out[15] = ')';
                        out[16] = 13; //Return
                        out[17] = 10; //New line

                        //Output everything
                        for (int j=0; j<18; j++){
                                usartd0_out_char(out[j]);
                        }
                }
        }
        return 0;
}
```

Part 4: VISUALIZING THE ADC CONVERSIONS

```
/*****************************************************************************
;Lab 7 Part 4
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
```

```
;Description: VISUALIZING THE ADC CONVERSIONS
;*********************************INCLUDES*************************************/
#include <avr/io.h>
#include <avr/interrupt.h>
#include "USART.h"

#define BLUE_PWM_LED PIN6_bm

//Global Variables to store output
int16_t test = 0;    //Digit output
volatile int global_flag = 0;      //Global flag of interrupt

int main(void){
	//Initialize
	PORTD_OUTSET = BLUE_PWM_LED;      //LED initially off
	PORTD_DIRSET = BLUE_PWM_LED;      //Set output

	adc_init();
	tcc0_init();
	usartd0_init();

	//Loop conversion
	while (1){
		//Output voltage when global flag gets set
		if (global_flag){
			//Reset
			global_flag = 0;
			//Output
			usartd0_out_char(((uint8_t) (test >> 8)));      //MSB
			usartd0_out_char(((uint8_t) test));      //LSB
			}
	}
	return 0;
}
```

Part 5: SWITCHING BETWEEN MULTIPLE INPUTS

```
/*********************************************************************************
;Lab 7 Part 5
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: SWITCHING BETWEEN MULTIPLE INPUTS
;*********************************INCLUDES*************************************/
#include <avr/io.h>
#include <avr/interrupt.h>
#include "USART.h"

#define BLUE_PWM_LED PIN6_bm

//Global Variables to store output
int16_t test = 0;    //Digit output
volatile int tflag = 0;     //Conversion flag
volatile int rflag = 0;     //Receiver flag
volatile char c;

int main(void){
	//Initialize
	PORTD_OUTSET = BLUE_PWM_LED;      //LED initially off
	PORTD_DIRSET = BLUE_PWM_LED;      //Set output
	PORTA_OUTSET |= PIN5_bm;    //Control J3
```

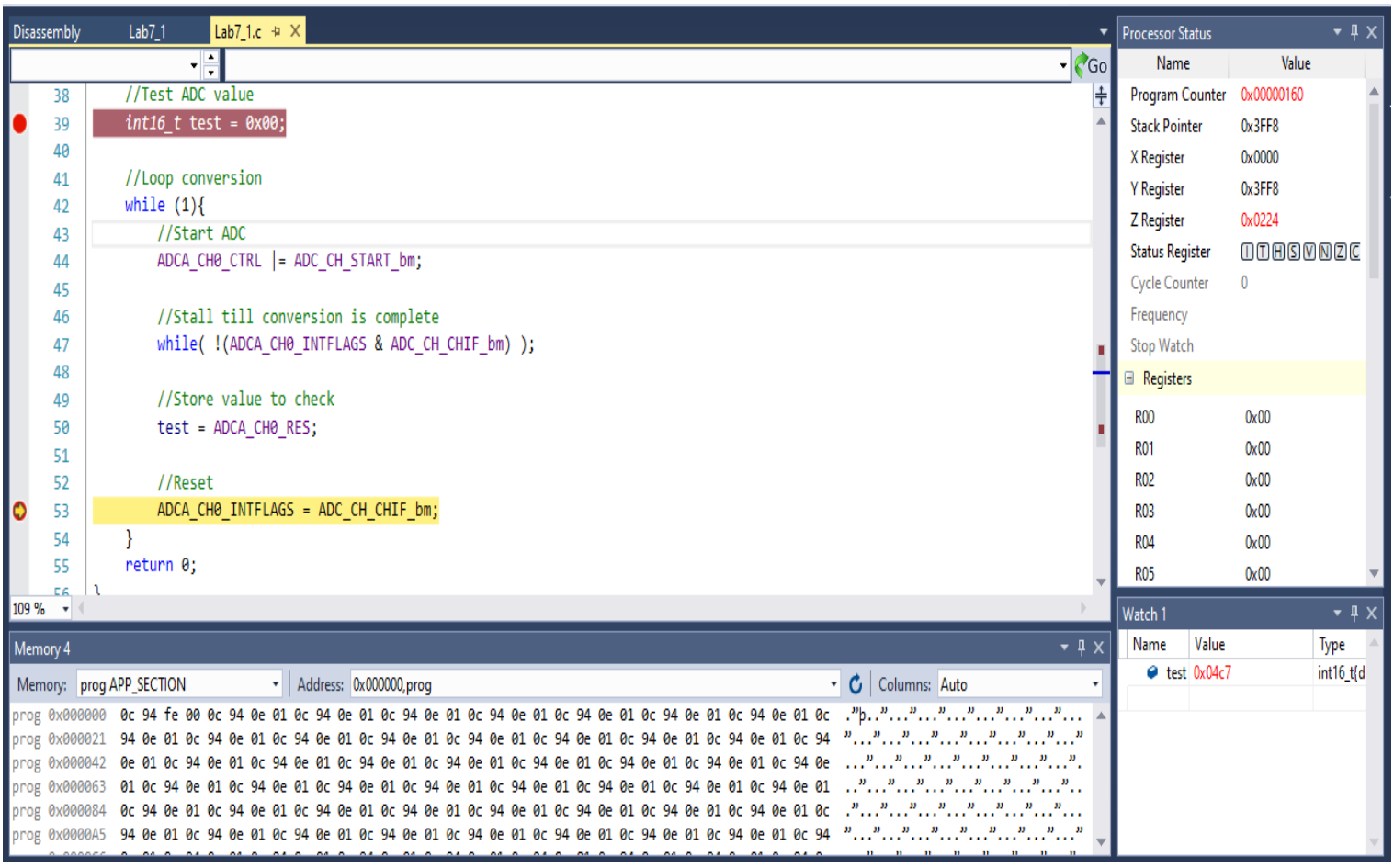
```
	PORTA_DIRSET |= PIN5_bm;

	tcc0_init();
	usartd0_init();
	adc_init();

	//Loop conversion
	while (1){
		//Get input
		if (rflag){
			//Reset
			rflag = 0;
			//Output
			if(c == '1') {
				//CDS
				ADCA_CH0_MUXCTRL = ADC_CH_MUXPOS_PIN1_gc | ADC_CH_MUXNEG_PIN6_gc;
			}
			else if (c == '2') {
				//Analog input jumper
				ADCA_CH0_MUXCTRL = ADC_CH_MUXPOS_PIN4_gc | ADC_CH_MUXNEG_PIN5_gc;
			}
		}
		if (tflag){
			//Reset
			tflag = 0;

			//Output
			usartd0_out_char(((uint8_t) test));	//LSB
			usartd0_out_char(((uint8_t) (test >> 8)));	//MSB
		}
	}
	return 0;
}
```

APPENDIX

Part 1: USING THE ADC SYSTEM

Support Code: Lab7_1.c

```
//Initialize ADC
void adc_init(void){
	//12-bit signed, right-adjusted, Normal, 2.5Vref
	ADCA_CTRLB = ADC_RESOLUTION_12BIT_gc | ADC_CONMODE_bm;
	ADCA_REFCTRL = ADC_REFSEL_AREFB_gc;		//2.5Vref

	//ADC Clock prescaler=512
	ADCA_PRESCALER = ADC_PRESCALER_DIV512_gc;

	//Enable Port A
	PORTA_DIRCLR = PIN1_bm | PIN6_bm;  //PortA input pins

	//Differential input signal with gain
	ADCA_CH0_CTRL = ADC_CH_INPUTMODE_DIFFWGAIN_gc;

	//MUXCTRL pin1 + and pin6 -
	ADCA_CH0_MUXCTRL = ADC_CH_MUXPOS_PIN1_gc | ADC_CH_MUXNEG_PIN6_gc;

	//Enable ADC
	ADCA_CTRLA = ADC_ENABLE_bm;
}
```



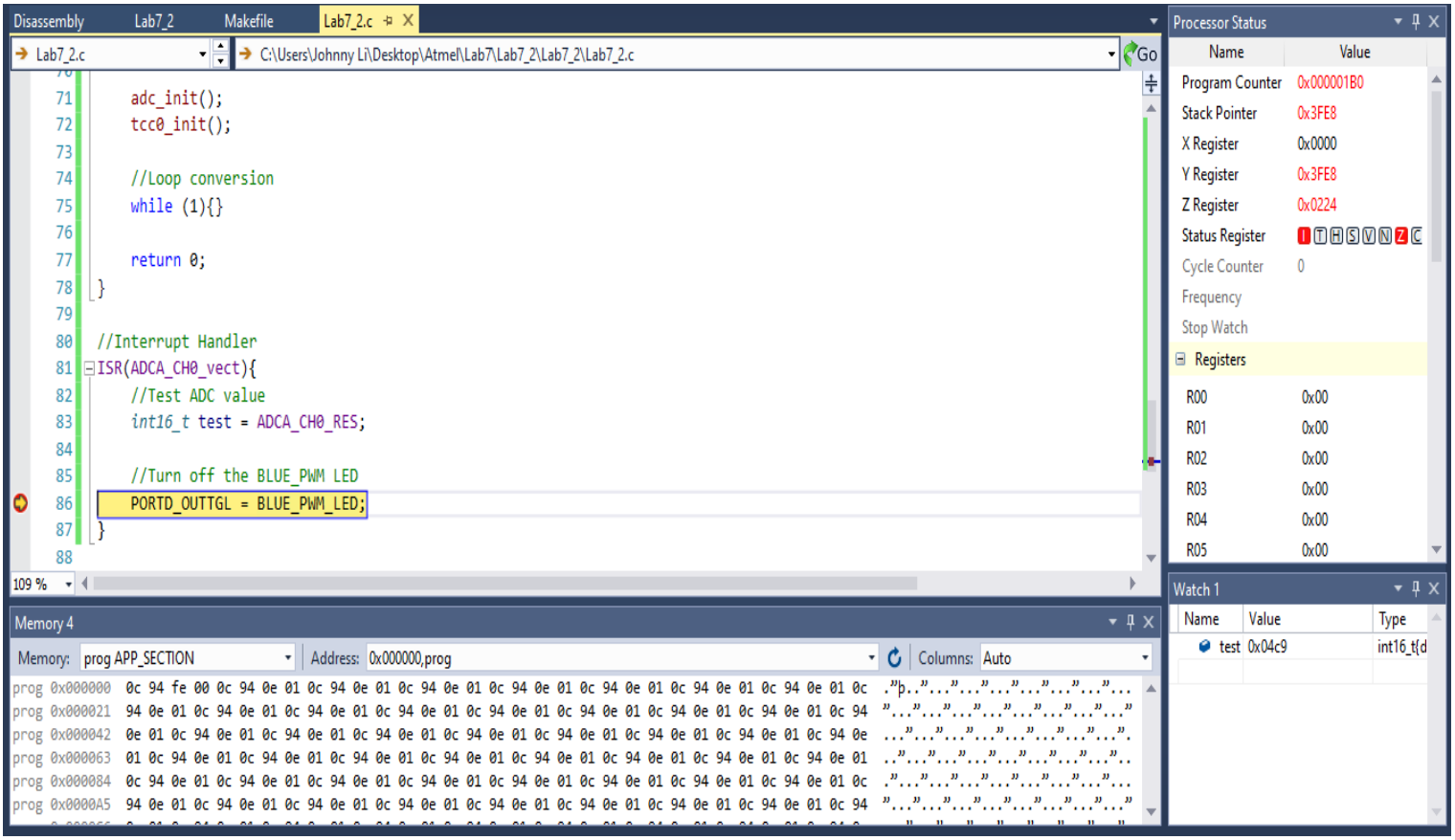
Screenshot 1: Store the 12-bit signed conversion result.

Part 2: SAMPLING AT A SPECIFIC RATE USING EVENTS

Support Code: Lab7_2.c

```
//Initialize TCC0 timer
 void tcc0_init(void){
	//SCK = 2MHz, Prescaler = 1024, Time = 0.5sec
	int period = (2000000/(1024*2));   //2 Hz

	//Set period
	TCC0_PERL = (uint8_t) period;      //Low Period
	TCC0_PERH = (uint8_t) (period>>8); //High Period

	//Normal mode timer
	TCC0_CTRLB = 0x00;

	//Trigger an event on Event Channel 0
	EVSYS_CH0MUX = EVSYS_CHMUX_TCC0_OVF_gc;

	//Set perscaler = 1024
	TCC0_CTRLA = 0x07;
 }

//Initialize ADC
void adc_init(void){
	//12-bit signed, right-adjusted, Normal, 2.5Vref
	ADCA_CTRLB = ADC_RESOLUTION_12BIT_gc | ADC_CONMODE_bm;
	ADCA_REFCTRL = ADC_REFSEL_AREFB_gc;              //2.5Vref

	//ADC Clock prescaler=512
	ADCA_PRESCALER = ADC_PRESCALER_DIV512_gc;

	//Enable Port A
	PORTA_DIRCLR = PIN1_bm | PIN6_bm;  //PortA input pins

	//Differential input signal with gain
	ADCA_CH0_CTRL = ADC_CH_INPUTMODE_DIFFWGAIN_gc;

	//MUXCTRL pin1 + and pin6 -
	ADCA_CH0_MUXCTRL = ADC_CH_MUXPOS_PIN1_gc | ADC_CH_MUXNEG_PIN6_gc;

	//Setup ADC Low Level interrupt
	ADCA_CH0_INTCTRL = ADC_CH_INTMODE_COMPLETE_gc | ADC_CH_INTLVL_LO_gc; //Triggered on flag- when a
conversion is complete

	//Enable Low Level interrupts
	PMIC_CTRL = PMIC_LOLVLEN_bm;
	//Enable global enable interrupts
	sei();

	//ADC conversion start when Event Channel 0 is triggered
	ADCA_EVCTRL = ADC_SWEEP_0_gc | ADC_EVSEL_0123_gc | ADC_EVACT_CH0_gc;

	//Enable ADC
	ADCA_CTRLA = ADC_ENABLE_bm;
}

//Interrupt Handler
ISR(ADCA_CH0_vect){
	//Test ADC value
```

```
	int16_t test = ADCA_CH0_RES;

	//Turn off the BLUE_PWM LED
	PORTD_OUTTGL = BLUE_PWM_LED;
}
```



Screenshot 2: Store the 12-bit signed conversion result in interrupt.

Part 3: OUTPUTTING SAMPLED DATA WITH UART

Support Code: USART.c

```
/*********************************************************************************
;Lab 7 Part 3
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: OUTPUTTING SAMPLED DATA WITH UART
*/

#include <avr/io.h>

//USART Initialization
void usartd0_init(void){
	//Configure TxD and RxD pins
	PORTD.OUTSET = PIN3_bm;
	PORTD.DIRSET = PIN3_bm;
	PORTD.DIRCLR = PIN2_bm;

	//Baud rate: At 2 MHz, 0 BSEL, -3 BSCALE corresponds to 128000 bps */
	USARTD0.BAUDCTRLA = (uint8_t)0;
	USARTD0.BAUDCTRLB = (uint8_t)( (-3 << 4) | (0 >> 8));
```

```
	//8 data bits, no parity, and one stop bit.
	USARTD0.CTRLC = USART_CMODE_ASYNCHRONOUS_gc | USART_PMODE_DISABLED_gc | USART_CHSIZE_8BIT_gc &
~USART_SBMODE_bm;

	//Enable Receiver and/or Transmitter
	USARTD0.CTRLB = USART_RXEN_bm | USART_TXEN_bm;
}

//Output character
void usartd0_out_char(char output){
	//Wait till transmission is done
	while(!(USARTD0.STATUS & USART_DREIF_bm));
	USARTD0.DATA = output;	//output c
}

//Output string
void usartd0_out_string(char *str){
	//Loop char pointer to get string
	while(*str){
		usartd0_out_char(*(str++)); //Output string
	}
}
```

Support Code: USART.h

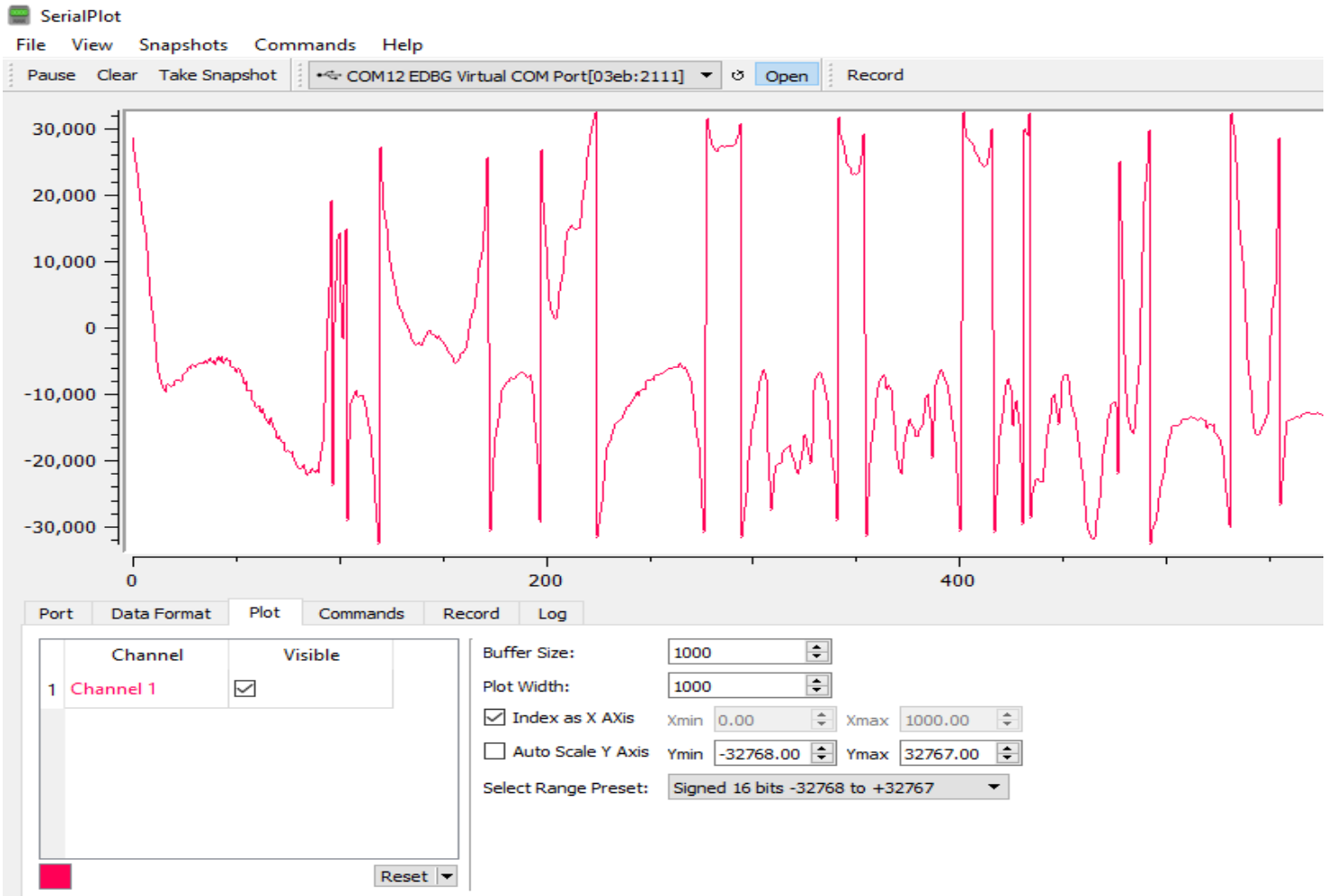
```
/*****************************************************************************
;Lab 7 Part 3
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: OUTPUTTING SAMPLED DATA WITH UART
*/
#ifndef USART_H_
#define USART_H_

//USART Initialization
void usartd0_init(void);

//Output character
void usartd0_out_char(char output);

//Output string
void usartd0_out_string(char *str);

#endif /* USART_H_ */
```

Screenshot 3: Voltage Transmission on Putty

Part 4: VISUALIZING THE ADC CONVERSIONS

Support Code:

```
//Initialize TCC0 timer
void tcc0_init(void){
	//SCK = 2MHz, Prescaler = 1024,
	int period = 5000;   //100 Hz

	//Set period
	TCC0_PERL = (uint8_t) period;      //Low Period
	TCC0_PERH = (uint8_t) (period>>8); //High Period

	//Normal mode timer
	TCC0_CTRLB = 0x00;

	//Trigger an event on Event Channel 0
	EVSYS_CH0MUX = EVSYS_CHMUX_TCC0_OVF_gc;

	//Set perscaler = 1024
	TCC0_CTRLA = 0x03;
}
```



Screenshot 4: Waveform in a the SerialPlot window.

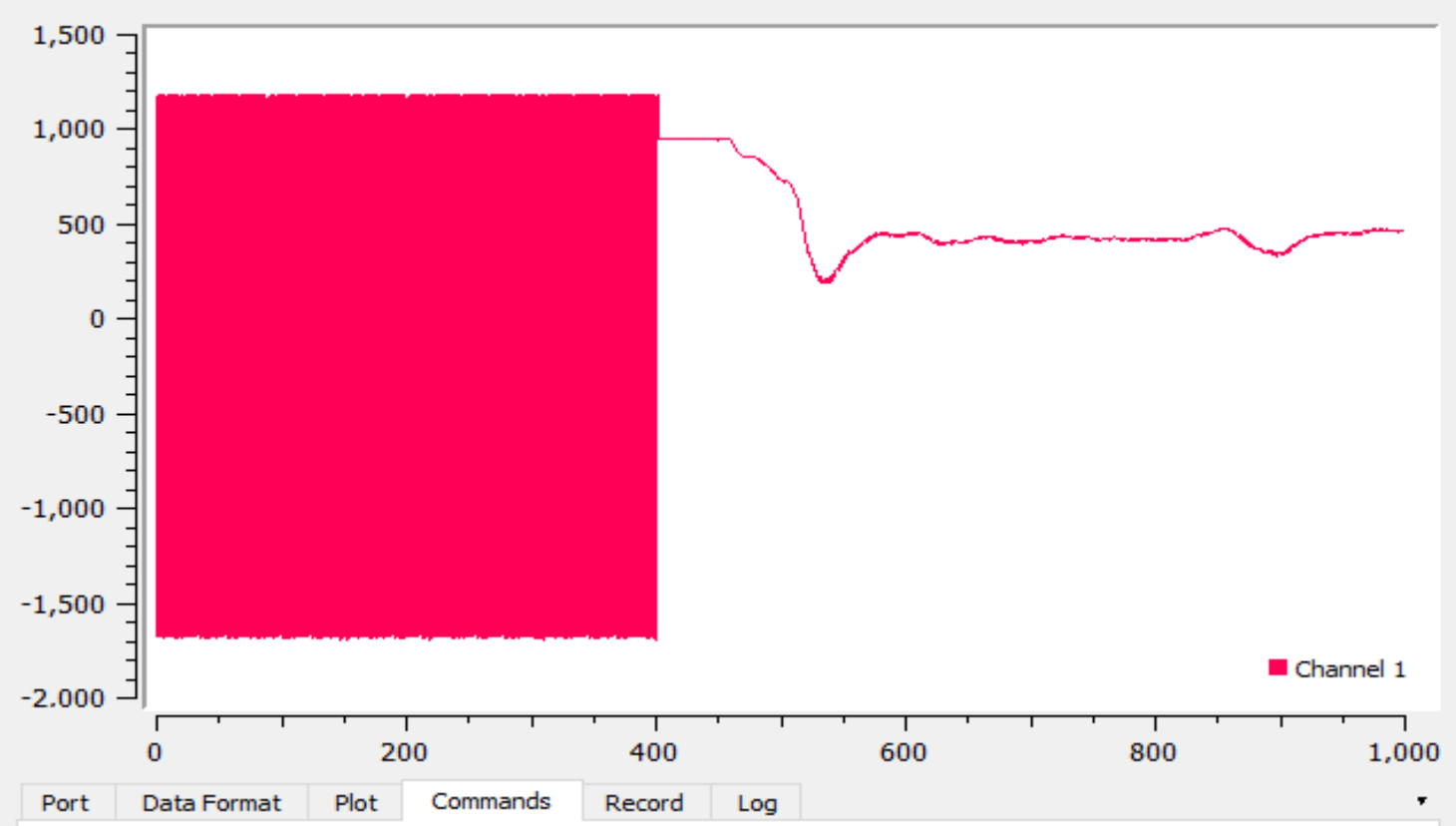
Part 5: SWITCHING BETWEEN MULTIPLE INPUTS

Support Code: USART.c

```
/*********************************************************************************
;Lab 7 Part 5
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: SWITCHING BETWEEN MULTIPLE INPUTS
*/

#include <avr/io.h>

//USART Initialization
void usartd0_init(void){
	//Configure TxD and RxD pins
	PORTD.OUTSET = PIN3_bm;
	PORTD.DIRSET = PIN3_bm;
	PORTD.DIRCLR = PIN2_bm;

	//Baud rate: At 2 MHz, 0 BSEL, -3 BSCALE corresponds to 128000 bps */
	USARTD0.BAUDCTRLA = (uint8_t)0;
	USARTD0.BAUDCTRLB = (uint8_t)( (-3 << 4) | (0 >> 8));

	//8 data bits, no parity, and one stop bit.
	USARTD0.CTRLC = USART_CMODE_ASYNCHRONOUS_gc | USART_PMODE_DISABLED_gc | USART_CHSIZE_8BIT_gc &
~USART_SBMODE_bm;

	//Enable Receiver and/or Transmitter
	USARTD0.CTRLB = USART_RXEN_bm | USART_TXEN_bm;

	//Enable interrupt
	USARTD0.CTRLA = USART_RXCINTLVL_LO_gc;
}
```

Support Code: Lab7_5.c

```
//Interrupt Handler
ISR (ADCA_CH0_vect){
	//Test ADC value
	test = ADCA_CH0_RES;

	//Set conversion flag
	tflag = 1;

	//Turn off the BLUE_PWM LED
	PORTD_OUTTGL = BLUE_PWM_LED;
	//Toggle J3 pin
	PORTA_OUTTGL |= PIN5_bm;
}

//Receiver Handler
ISR (USARTD0_RXC_vect){
	//Get input
	c = USARTD0.DATA;

	//Set receiver flag
	rflag = 1;
}
```



Screenshot 5: Serial Plot of analog input jumper (2) to CdS cell data (1).