REQUIREMENTS NOT MET

N/A

PROBLEMS ENCOUNTERED

In Part 1, I was unsure how to verify whether the CdS photoresistor was functioning properly since I did not have access to a multimeter, so I instead confirmed its operation by observing ADC value changes with varying light levels. In Part 2, I mistakenly enabled timer interrupts, which caused issues routing the timer overflow through the Event System; this prevented ADC conversions from triggering, and caused the LED not to toggle; disabling the interrupt fixed the issue. In Part 3, I initially had trouble understanding how to output hexadecimal characters over UART, but after learning to isolate and convert each nibble into its ASCII equivalent, the values displayed correctly. In Part 4, my SerialPlot waveform appeared compressed because the Y-axis was configured for 16-bit signed integers instead of 12-bit signed ADC values; correcting the format resolved the display issue.

FUTURE WORK/APPLICATIONS

With more time and resources, this lab could be expanded into a more advanced oscilloscope or data-logging system. Additional sensors could be connected to the ADC to record multiple analog inputs simultaneously, or data could be stored in external memory for later analysis. The sampling rate could be made adjustable, allowing higher-frequency signals to be captured accurately. These improvements would make the system more versatile for applications such as environmental monitoring, audio signal capture, or testing analog circuits in real time.

PRE-LAB EXERCISES

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

a. We must use the ADCA module because the analog backpack on the µPAD is wired to the Port A pins of the ATxmega128A1U. According to the device datasheet, ADCA is connected to Port A while ADCB is connected to Port B. Since the analog backpack's inputs are routed through Port A, only ADCA can access these analog signals. Port B is not connected to any of the analog input lines used by the backpack, so ADCB cannot be used for this lab.

ii. Would it be possible to use any other ADC configurations such as single-ended, differential, differential with gain, etc. with the current pinout and connections of the OOTB Analog Backpack? Why or why not?

Yes, but only certain ADC configurations are compatible with the current OOTB Analog Backpack connections. Both the CdS cell (PA1–PA6) and the AIN amplifier outputs (PA5–PA4) produce differential signals centered around 2.5 V (when using AREFB), so differential mode is valid and required to measure the voltage difference between these pins. Differential with gain is also valid for the AIN pair, but it is unnecessary for this lab's purposes because the onboard amplifier already scales the signal for the ADC's input range. Single-ended mode is not valid since neither signal is referenced to ground, and internal mode is only valid for measuring built-in chip sources, not the backpack's external inputs.

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

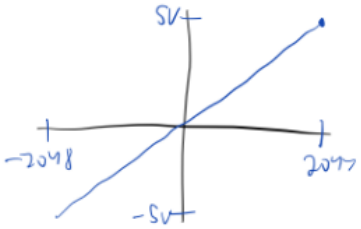
a. A 12-bit ADC provides much finer precision than an 8-bit ADC, dividing the input range into 4096 discrete steps instead of only 256. This allows for more accurate and detailed voltage measurements, which is especially useful when dealing with small signal changes or when high accuracy is required.
b. However, 8-bit resolution can still be useful when precision isn't critical. It offers faster conversions and lower data processing overhead, which can be advantageous in applications where speed is more important than accuracy—such as when sampling high-speed signals or when transmitting large amounts of data over a limited bandwidth.

iv. What is the decimal voltage value that is equivalent to a 12-bit signed result of 0x421, given a voltage range of -5V to +5?

a. 2.58 V

12 bit signed result of 0x421
voltage range of −5 to +5
What is the decimal voltage value?

$0x421 = 4(256) + 2(16) + 1 = 1057$



$\Delta V = m(DV) + b$

12 bit signed
max val = +2047
min val = −2048
total = 4096

$m = \frac{5-(-5)}{2047-(-2048)} = \frac{10}{4095} = 0.0244$

$V_A = \frac{10}{4095}(V_D) + b$

$5 = \frac{10}{4095}(2047) + b$

$b = 0.001221$

$V_A = \frac{10}{4095}(V_D) + 0.001221$

$V_A = \frac{10}{4095}(1057) + 0.001221$

$V_A = 2.5824$

$V_A \approx 2.58\ V$

b.

v. Given a 8-bit signed ADC system with a voltage reference range of -2V to +8V, express the expected digital value in terms of the analog input voltage, using the form VD = f(VA).

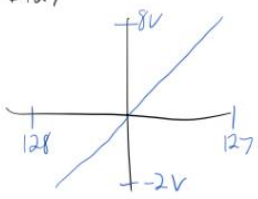
a. VD = 25.5(VA – 3.0196)

8 bit signed ADC system

−2V to +8V

express digital value in terms of analog input voltage

8 bit = 256 values
min val = −128
max val = +127



$m = \frac{10}{255}$

$V_A = \frac{10}{255}V_d + b$

$8 = \frac{10}{255}(127) + b$

$b = 3.0196$

$V_A = \frac{10}{255}V_d + 3.0196$

$\frac{10}{255}V_d = V_A - 3.0196$

$V_d = \frac{255}{10}(V_A - 3.0196)$

$V_D = 25.5(V_A - 3.0196)$

b.

PSEUDOCODE/FLOWCHARTS

SECTION 1

void adc_init(void)

```
//set 12 bit signed mode, right adjusted
        //resolution = 00,    convmode = 1,  ctrlb
//Normal mode
        //freerun = 0 ctrlb
//2.5 voltage reference (refB)
        //refctrl = AREFB
//prescaler = div 64
//configure adca.ch0.ctrl
        //differential mode
        // gain = 1x
//Configure ADCA.CH0.MUXCTRL
        //Select positive input = PA1 (CDS+)
        //Select negative input = PA6 (CDS−)
//enable ADCA, adca.ctrlA
```

Lab7_1.c

```
MAIN:
//init system
        //call adc_init();
        //declare int16_t adc_result to store 12 bit signed adc val
//infinite loop
While(1)
{
//start adc conversion on channel 0
```

//wait for ADC interrupt flag to be set

//clear flag

//store result in adc_result

//breakpoint

}

void tcc0_init(void)

//set count to 0 in CNT

//set period to correspond to 0.25 seconds using pre of 64 in PER

//enable low level interrupts in INTCTRLA

//configure CH0MUX

// CH0MUX = TCC0 overflow (1100 _000)

void adc_init(void) (updated)

//set 12 bit signed mode, right adjusted

//resolution = 00, convmode = 1, ctrlb

//Normal mode

//freerun = 0 ctrlb

//2.5 voltage reference (refB)

//refctrl = AREFB

//prescaler = div 64

//configure adca.ch0.ctrl

//differential mode

// gain = 1x

//Configure ADCA.CH0.MUXCTRL

//Select positive input = PA1 (CDS+)

//Select negative input = PA6 (CDS−)

//enable ADC interrupts in ADCA.CH0.INTCTRL

//use EVCTRL register in ADC module

```
        //make ADC conversion start when event channel 0 is triggered
        // EVSEL to ch0
        //event action to start a single conversion on CH0
        //sweep ch0 only
//enable ADCA, adca.ctrlA
```

Lab7_2.c

```
//include files
//define global variable (adc_result)

MAIN:
//init system
        //call tcc0_init();
        //call adc_init();
//configure BLUE_LED
//enable pmic and global interrupts

//infinite loop
While(1)
{
}
```

ISR(ADCA_CH0_vect)

```
//clear interrupt flag
//save result in signed 16bit int
//toggle blue_PWM LEd
```

Lab7_3.c

(INCLUDES everything from Lab7_2.c, pseudo code is written for updated sections)

ISR(ADCA_CH0_vect)

//clear interrupt flag

//save result in signed 16bit int

//set global flag

//toggle blue_PWM Led

void usartd0_init(void){

//initialize usart pins first

//outset TX pin (PD3 to idle high)

//dirset TX pin

//dirclr RX pin (PF2)

//set to odd parity, 8 bits, 1 stop bit in PORTD.CTRLC

//set baud rate to 94744 (bscale = -7, bsel = 41)

//enable tx and rx

}

MAIN:

//init system

    //call tcc0_init();

    //call adc_init();

    //call usartd0init();

//configure BLUE_LED

//enable pmic and global interrupts

//infinite loop

While(1)

{

//check if global flag is set

```
//once set clear it
//output voltage to serial terminal
        //call send voltage
}

Void send_voltage(int16_t adc_result)
{
//use equation of a line to determine the Analog value from the digital val VA = f(D)

//if result < 0 outchar '-'
//else outchar '+'
//Example of what to do for decimal value:
//Pi = 3.14159…//variable holds original value
//Int1 = (int) Pi = 3 \ 3 is the first digit of Pi
//Transmit the ASCII value of Int1 and then '.'
//Pi2 = 10*(Pi - Int1) = 1.4159…
//Int2 = (int) Pi2 = 1 \ 1 is the second digit of Pi
//Transmit the ASCII value of Int2 digit
//Pi3 = 10*(Pi2 – Int2) = 4.159…
//Int3 = (int) Pi3 = 4 \ 4 is the third digit of Pi
//Transmit the ASCII value of Int3 digit, then a space, and then a 'V'
//output '('
//to output hex value
//shift the num left 4 to get high nibble
        //if the num is < 10 add 0x30 and outchar
        //else num is A-F so subtract 10 and add "A" and outchar
//mask the high bit then do the same
//output ')'
//output CR and LF
}
```

Lab7_4.c (updated changes from Lab7_3.c are shown below)

void tcc0_init(void)

//set count to 0 in CNT

//set period to correspond to 140hz using pre of 64 in PER (223)

//configure CH0MUX

// CH0MUX = TCC0 overflow (1100 _000)

MAIN:

//init system

//call tcc0_init();

//call adc_init();

//call usartd0init();

//configure BLUE_LED

//enable pmic and global interrupts

//infinite loop

While(1)

{

//check if global flag is set

//once set clear it

//output voltage to serial plot

//outchar adc_result low byte (result & 0x00FF)

//outchar adc_result high byte (result >> 8 & 0xFF)

}

Lab7_5.c (updated changes from Lab7_4.c are shown below)

void usartd0_init(void){

//initialize usart pins first

//outset TX pin (PD3 to idle high)

//dirset TX pin

//dirclr RX pin (PF2)

//set to odd parity, 8 bits, 1 stop bit in PORTD.CTRLC

//set baud rate to 94744 (bscale = -7, bsel = 41)

//enable tx and rx

//enable rx interrupts (low level)

}

ISR(USARTD0_RXC_vect)

{

//clear the interrupt flag

//read the character from the data register

//if the character is ‘C’ make configure the ADCA.CH0.MUXCTRl for CDS+ (pa1) and CDS- (pa6)

//if the char is ‘F’ make configure ADCA.CH0.MUXCTRL for AIN+ (pa5) and AIN- (pa4)

//else don’t change it

}

PROGRAM CODE

SECTION 2

Lab7_1.c

```
/*
 * Lab7.c
 *
 * Created: 11/6/2025 11:18:18 PM
 * Author : arist
 */

/*****************************************************
 * Lab 7, Section 22
 * Name: Arion Stern
 * Class #: 11303
 * PI Name: Ian Santamauro
 * Description:
 *   Configures ADCA to sample the CdS photoresistor on
 *   the OOTB Analog Backpack using 12-bit signed mode,
 *   2.5 V reference, and differential inputs.
 *****************************************************/


#include <avr/io.h>


/*****************************************************
 * Name:        adc_init
 * Purpose:     Initialize ADCA to sample the CdS cell
 *              (PA1 = CDS+, PA6 = CDS-) on the Analog Backpack.
 * Inputs:      None
 * Outputs:     None
 *****************************************************/
void adc_init(void)
{
       // set 12-bit signed mode, right adjusted
       // resolution = 00, convmode = 1 (signed)  ? CTRLB
       // normal mode (freerun = 0)
       ADCA.CTRLB = ADC_CONMODE_bm | ADC_RESOLUTION_12BIT_gc;

       // 2.5 V voltage reference (AREFB)
       // REFSEL = AREFB ? REFCTRL
       ADCA.REFCTRL = ADC_REFSEL_AREFB_gc;

       // prescaler = div64 (32 MHz / 64 = 500 kHz)
       ADCA.PRESCALER = ADC_PRESCALER_DIV64_gc;

       // configure ADCA.CH0.CTRL
       // differential mode, gain = 1×
       //ADCA.CH0.CTRL = ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_1X_gc;
       ADCA.CH0.CTRL = ADC_CH_INPUTMODE_DIFF_gc;

       // configure ADCA.CH0.MUXCTRL
       // positive input = PA1 (CDS+), negative input = PA6 (CDS?)
       ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN1_gc | ADC_CH_MUXNEG_PIN6_gc;

       //enable ADC interrupts in ADCA.CH0.INTCTRL
```

```
	ADCA.CH0.INTCTRL = ADC_CH_INTLVL_LO_gc;
	//use EVCTRL register in ADC module
	//make ADC conversion start when event channel 0 is triggered
	// EVSEL to ch0
	//event action to start a single conversion on CH0
	//sweep ch0 only
	ADCA.EVCTRL = ADC_EVACT_CH0_gc | ADC_EVSEL_0123_gc | ADC_SWEEP_0_gc;

	// enable ADCA
	ADCA.CTRLA = ADC_ENABLE_bm;

}
/*****************************************************
 * Name: main
 * Purpose: Continuously perform ADC conversions and
 *          store results for testing.
 * Inputs:  None
 * Outputs: None
 *****************************************************/

int main(void)
{
//init system
//call adc_init();
adc_init();
//declare int16_t adc_result to store 12 bit signed adc val
int16_t adc_result = 0xFFFF;
//infinite loop
while(1)
{
	//start adc conversion on channel 0
	ADCA.CH0.CTRL |= ADC_CH_START_bm;
	//wait for ADC interrupt flag to be set
	while(!(ADCA.CH0.INTFLAGS & ADC_CH_CHIF_bm));
	//clear flag
	ADCA.CH0.INTFLAGS = ADC_CH_CHIF_bm;
	//store result in adc_result
	adc_result = ADCA.CH0.RES;
	//breakpoint
	asm("nop");
}

}
```

Lab7_2.c

```
/*
 * lab7_2.c
 *
 * Created: 11/7/2025 4:49:15 PM
 *  Author: arist
 */

/*****************************************************
 * Lab 7, Section 22
 * Name: Arion Stern
 * Class #: 11303
 * PI Name: Ian Santamauro
 * Description:
 *   Uses the Event System and Timer/Counter C0 to trigger
 *   ADC conversions at 4 Hz. The ADC ISR stores results
 *   and toggles the BLUE_PWM LED each time a sample occurs.
 *****************************************************/



#include <avr/io.h>
#include <avr/interrupt.h>

volatile int16_t adc_result = 0xFFFF;


/*******************************************************
 * Name: tcc0_init
 * Purpose: Configure TCC0 to overflow every 0.25 s and
 *          generate an event on channel 0.
 * Inputs:  None
 * Outputs: None
 *******************************************************/

void tcc0_init(void)
{
	//set count to 0 in CNT
	TCC0.CNT = 0;
	//set period to correspond to 0.25 seconds using pre of 64 in PER
	//per = clk/pre * dur = 7812
	TCC0.PER = 7812;
	//enable low level interrupts in INTCTRLA DONT DO THIS IT BREAKS THE EVENT SYSTEM
	//TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;
	//configure CH0MUX
	// CH0MUX = TCC0 overflow (1100 _000)
	EVSYS.CH0MUX = EVSYS_CHMUX_TCC0_OVF_gc;
	//start TCC0
	TCC0.CTRLA = TC_CLKSEL_DIV64_gc;
}


/*******************************************************
 * Name: adc_init
 * Purpose: Initialize ADCA for differential sampling and
 *          start conversions from Event Channel 0.
 * Inputs:  None
 * Outputs: None
 *******************************************************/
```

```
void adc_init(void)
{
	// set 12-bit signed mode, right adjusted
	// resolution = 00, convmode = 1 (signed)  ? CTRLB
	// normal mode (freerun = 0)
	ADCA.CTRLB = ADC_CONMODE_bm | ADC_RESOLUTION_12BIT_gc;

	// 2.5 V voltage reference (AREFB)
	// REFSEL = AREFB ? REFCTRL
	ADCA.REFCTRL = ADC_REFSEL_AREFB_gc;

	// prescaler = div64 (32 MHz / 64 = 500 kHz)
	ADCA.PRESCALER = ADC_PRESCALER_DIV64_gc;

	// configure ADCA.CH0.CTRL
	// differential mode, gain = 1×
	//ADCA.CH0.CTRL = ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_1X_gc;
	//ADCA.CH0.CTRL = ADC_CH_INPUTMODE_DIFF_gc;
	ADCA.CH0.CTRL = ADC_CH_INPUTMODE_DIFF_gc | ADC_CH_GAIN_1X_gc;


	// configure ADCA.CH0.MUXCTRL
	// positive input = PA1 (CDS+), negative input = PA6 (CDS?)
	ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN1_gc | ADC_CH_MUXNEG_PIN6_gc;

	//enable ADC interrupts in ADCA.CH0.INTCTRL
	ADCA.CH0.INTCTRL = ADC_CH_INTLVL_LO_gc;
	//use EVCTRL register in ADC module
	//make ADC conversion start when event channel 0 is triggered
	// EVSEL to ch0
	//event action to start a single conversion on CH0
	//sweep ch0 only
	ADCA.EVCTRL = ADC_EVACT_CH0_gc | ADC_EVSEL_0123_gc | ADC_SWEEP_0_gc;
	//ADCA.EVCTRL = (0 << 3) | (1 << 0);

	// enable ADCA
	ADCA.CTRLA = ADC_ENABLE_bm;

}

/*****************************************************
 * Name: main
 * Purpose: Initialize timer, ADC, and interrupts, then
 *          run continuously while ADC conversions occur
 *          automatically at 4 Hz.
 * Inputs:  None
 * Outputs: None
 *****************************************************/

int main(void)
{
	//init system
	//call tcc0_init();
	tcc0_init();
	//call adc_init();
	adc_init();
	//configure BLUE_LED
	PORTD.OUTCLR = PIN6_bm;
	PORTD.DIRSET = PIN6_bm;
	//enable pmic and global interrupts
```

```
        PMIC.CTRL = PMIC_LOLVLEN_bm;
        sei();

        //ADCA.CH0.CTRL |= ADC_CH_START_bm;


        //infinite loop
        while(1)
        {
        }


}


/*****************************************************
 * Name: ISR(ADCA_CH0_vect)
 * Purpose: Handle ADC conversion complete interrupt,
 *          store result, and toggle the BLUE_PWM LED.
 * Inputs:  None
 * Outputs: None
 *****************************************************/
ISR(ADCA_CH0_vect)
{
//clear interrupt flag
ADCA.CH0.INTFLAGS = ADC_CH_CHIF_bm;
//save result in signed 16bit int
adc_result = ADCA.CH0RES;
//toggle blue_PWM LEd
PORTD.OUTTGL = PIN6_bm;
}
```

Lab7_3.c

```
/*
 * Lab7_3.c
 *
 * Created: 11/7/2025 7:43:07 PM
 *  Author: arist
 */

/*****************************************************
 * Lab 7, Section 22
 * Name: Arion Stern
 * Class #: 11303
 * PI Name: Ian Santamauro
 * Description:
 *   Creates a voltmeter that samples the CdS cell and
 *   outputs signed voltage readings every second over
 *   USARTD0 at 94,744 bps in decimal and hexadecimal form.
 *****************************************************/

#include <avr/io.h>
#include <avr/interrupt.h>

volatile int16_t adc_result = 0xFFFF;
volatile uint8_t new_data_flag = 0;

/* At 2 MHz SYSclk, 41 BSEL, -7 BSCALE corresponds to 94744 bps */
#define BSEL     (41)
#define BSCALE   (-7)
#define CR (13)
#define LF (10)

//prototypes
void send_voltage(int16_t adc_result);
void tcc0_init(void);
void usartd0_init(void);
void usartd0_out_char(char c);
void adc_init(void);

/*****************************************************
 * Name: tcc0_init
 * Purpose: Configure TCC0 to overflow every 1 s and
 *          trigger ADC conversions via Event Ch 0.
 * Inputs:  None
 * Outputs: None
 *****************************************************/

void tcc0_init(void)
{
	//set count to 0 in CNT
	TCC0.CNT = 0;
	//set period to correspond to 0.25 seconds using pre of 64 in PER
	//per = clk/pre * dur = 7812
	TCC0.PER = 7812 * 4; // 1 second now
	//enable low level interrupts in INTCTRLA DONT DO THIS IT BREAKS THE EVENT SYSTEM
	//TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;
	//configure CH0MUX
	// CH0MUX = TCC0 overflow (1100 _000)
	EVSYS.CH0MUX = EVSYS_CHMUX_TCC0_OVF_gc;
	//start TCC0
	TCC0.CTRLA = TC_CLKSEL_DIV64_gc;
```

```
}

/*****************************************************
 * Name: usartd0_init
 * Purpose: Initialize USARTD0 for 94,744 bps, 8 data bits,
 *          odd parity, and one stop bit.
 * Inputs:  None
 * Outputs: None
 *****************************************************/

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

  /* Configure baud rate. to 94744 */
       USARTD0.BAUDCTRLA = (uint8_t)BSEL;
       USARTD0.BAUDCTRLB = (uint8_t)((BSCALE << 4)|(BSEL >> 8));

  /* Configure remainder of serial protocol. */
//set to odd parity, 8 bits, 1 stop bit in PORTD.CTRLC
       USARTD0.CTRLC =      (USART_CMODE_ASYNCHRONOUS_gc |
                             USART_PMODE_ODD_gc   |
                             USART_CHSIZE_8BIT_gc)       &
                            ~USART_SBMODE_bm;


  /* Enable receiver and/or transmitter systems. */
       USARTD0.CTRLB = USART_RXEN_bm | USART_TXEN_bm;

  /* Enable interrupt (optional). */
       /* USARTD0.CTRLA = USART_RXCINTLVL_MED_gc; */
}


/*****************************************************
 * Name: usartd0_out_char
 * Purpose: Transmit one character through USARTD0.
 * Inputs:  c - character to send
 * Outputs: None
 *****************************************************/

void usartd0_out_char(char c)
{
       while(!(USARTD0.STATUS & USART_DREIF_bm));
       USARTD0.DATA = c;
}


/*****************************************************
 * Name: adc_init
 * Purpose: Initialize ADCA for differential sampling and
 *          start conversions from Event Channel 0.
 * Inputs:  None
 * Outputs: None
 *****************************************************/

void adc_init(void)
{
```

```
	// set 12-bit signed mode, right adjusted
	// resolution = 00, convmode = 1 (signed)  ? CTRLB
	// normal mode (freerun = 0)
	ADCA.CTRLB = ADC_CONMODE_bm | ADC_RESOLUTION_12BIT_gc;

	// 2.5 V voltage reference (AREFB)
	// REFSEL = AREFB ? REFCTRL
	ADCA.REFCTRL = ADC_REFSEL_AREFB_gc;

	// prescaler = div64 (32 MHz / 64 = 500 kHz)
	ADCA.PRESCALER = ADC_PRESCALER_DIV64_gc;

	// configure ADCA.CH0.CTRL
	// differential mode, gain = 1×
	//ADCA.CH0.CTRL = ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_1X_gc;
	//ADCA.CH0.CTRL = ADC_CH_INPUTMODE_DIFF_gc;
	ADCA.CH0.CTRL = ADC_CH_INPUTMODE_DIFF_gc | ADC_CH_GAIN_1X_gc;


	// configure ADCA.CH0.MUXCTRL
	// positive input = PA1 (CDS+), negative input = PA6 (CDS?)
	ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN1_gc | ADC_CH_MUXNEG_PIN6_gc;

	//enable ADC interrupts in ADCA.CH0.INTCTRL
	ADCA.CH0.INTCTRL = ADC_CH_INTLVL_LO_gc;
	//use EVCTRL register in ADC module
	//make ADC conversion start when event channel 0 is triggered
	// EVSEL to ch0
	//event action to start a single conversion on CH0
	//sweep ch0 only
	ADCA.EVCTRL = ADC_EVACT_CH0_gc | ADC_EVSEL_0123_gc | ADC_SWEEP_0_gc;
	//ADCA.EVCTRL = (0 << 3) | (1 << 0);

	// enable ADCA
	ADCA.CTRLA = ADC_ENABLE_bm;

}

/*****************************************************
 * Name: main
 * Purpose: Initialize all modules and output voltage
 *          readings once per second via UART.
 * Inputs:  None
 * Outputs: None
 *****************************************************/

int main(void)
{
	//init system
	//call tcc0_init();
	tcc0_init();
	//call adc_init();
	adc_init();
	usartd0_init();
	//configure BLUE_LED
	PORTD.OUTCLR = PIN6_bm;
	PORTD.DIRSET = PIN6_bm;
	//enable pmic and global interrupts
	PMIC.CTRL = PMIC_LOLVLEN_bm;
	sei();
```

```
	//ADCA.CH0.CTRL |= ADC_CH_START_bm;


	//infinite loop
	while(1)
	{
		//check if global flag is set
		//if set clear it and output voltage to serial terminal (call send voltage)
		if (new_data_flag == 1)
		{
			new_data_flag = 0;
			send_voltage(adc_result);
		}

	}



}

/*****************************************************
 * Name: ISR(ADCA_CH0_vect)
 * Purpose: Store ADC result, set update flag, and toggle
 *          the BLUE_PWM LED each conversion.
 * Inputs:  None
 * Outputs: None
 *****************************************************/

ISR(ADCA_CH0_vect)
{
	//clear interrupt flag
	ADCA.CH0.INTFLAGS = ADC_CH_CHIF_bm;
	//save result in signed 16bit int
	adc_result = ADCA.CH0RES;
	//set global flag
	new_data_flag = 1;
	//toggle blue_PWM LEd
	PORTD.OUTTGL = PIN6_bm;
}


/*****************************************************
 * Name: send_voltage
 * Purpose: Convert ADC result to signed decimal voltage
 *          with two decimal places and hexadecimal value,
 *          then transmit formatted output.
 * Inputs:  adc_result – 12-bit signed ADC value
 * Outputs: None
 *****************************************************/

void send_voltage(int16_t adc_result)
{
	float voltage = 0;
	float abs_voltage = 0;
	int16_t int_part, dec1, dec2;
	//char hex_digits[4];
	uint16_t adc_unsigned;
	//use equation of a line to determine the Analog value from the digital val VA = f(D)
	voltage = (5.0/4095) * adc_result + 6.105e-4;
	//if result < 0 outchar '-'
	//else outchar '+'
```

```
if(voltage < 0)
{
	usartd0_out_char('-');
	abs_voltage = -voltage;
}
else
{
	usartd0_out_char('+');
	abs_voltage = voltage;
}


//Example of what to do for decimal value:
//Pi = 3.14159…//variable holds original value
//Int1 = (int) Pi = 3 ? 3 is the first digit of Pi
//Transmit the ASCII value of Int1 and then ‘.’
//Pi2 = 10*(Pi - Int1) = 1.4159…
//Int2 = (int) Pi2 = 1 ? 1 is the second digit of Pi
//Transmit the ASCII value of Int2 digit
//Pi3 = 10*(Pi2 – Int2) = 4.159…
//Int3 = (int) Pi3 = 4 ? 4 is the third digit of Pi
//Transmit the ASCII value of Int3 digit, then a space, and then a ‘V’
int_part = (int16_t)abs_voltage;
abs_voltage = (abs_voltage - int_part) * 10;
dec1 = (int16_t)abs_voltage;
abs_voltage = (abs_voltage - dec1) * 10;
dec2 = (int16_t)abs_voltage;

usartd0_out_char(int_part + '0');
usartd0_out_char('.');
usartd0_out_char(dec1 + '0');
usartd0_out_char(dec2 + '0');
usartd0_out_char(' ');
usartd0_out_char('V');
usartd0_out_char(' ');
usartd0_out_char('(');
usartd0_out_char('0');
usartd0_out_char('x');

//output hex value
adc_unsigned = (uint16_t)(adc_result & 0x0FFF);

//shift the num left 4 to get high nibble
//if the num is < 10 add 0x30 and outchar
//else num is A-F so subtract 10 and add “A” and outchar
//mask the high bit then do the same
for (int shift = 8; shift >= 0; shift -= 4)
{
	uint8_t nibble = (adc_unsigned >> shift) & 0x0F;

	if (nibble < 10)
	usartd0_out_char('0' + nibble);
	else
	usartd0_out_char((nibble - 10) + 'A');
}

//output ‘)‘
//output CR and LF
usartd0_out_char(')');
usartd0_out_char(CR);
usartd0_out_char(LF);
```

}

Lab7_4.c

```
/*
 * lab7_4.c
 *
 * Created: 11/8/2025 2:38:24 PM
 *  Author: arist
 */
/*****************************************************
 * Lab 7, Section 22
 * Name: Arion Stern
 * Class #: 11303
 * PI Name: Ian Santamauro
 * Description:
 *   Streams 16-bit signed ADC data at 140 Hz over USARTD0
 *   for SerialPlot visualization of the CdS cell waveform.
 *****************************************************/

#include <avr/io.h>
#include <avr/interrupt.h>

volatile int16_t adc_result = 0xFFFF;
volatile uint8_t new_data_flag = 0;

/* At 2 MHz SYSclk, 41 BSEL, -7 BSCALE corresponds to 94744 bps */
#define BSEL     (41)
#define BSCALE   (-7)
#define CR (13)
#define LF (10)

//prototypes
void send_voltage(int16_t adc_result);
void tcc0_init(void);
void usartd0_init(void);
void usartd0_out_char(char c);
void adc_init(void);


/*****************************************************
 * Name: tcc0_init
 * Purpose: Configure TCC0 to overflow at 140 Hz and
 *          trigger ADC conversions via Event Ch 0.
 * Inputs:  None
 * Outputs: None
 *****************************************************/

void tcc0_init(void)
{
	//set count to 0 in CNT
	TCC0.CNT = 0;
	//set period to correspond to 0.25 seconds using pre of 64 in PER
	//per = clk/pre * dur = 7812
	//TCC0.PER = 7812 * 4; // 1 second now

	TCC0.PER = 223; // 140hz now

	//enable low level interrupts in INTCTRLA DONT DO THIS IT BREAKS THE EVENT SYSTEM
	//TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;
	//configure CH0MUX
```

```
	// CH0MUX = TCC0 overflow (1100 _000)
	EVSYS.CH0MUX = EVSYS_CHMUX_TCC0_OVF_gc;
	//start TCC0
	TCC0.CTRLA = TC_CLKSEL_DIV64_gc;
}


/*****************************************************
 * Name: usartd0_init
 * Purpose: Initialize USARTD0 for 94 744 bps with odd
 *          parity and 8-bit data format.
 * Inputs:  None
 * Outputs: None
 *****************************************************/
void usartd0_init(void)
{
	/* Configure relevant TxD and RxD pins. */
	PORTD.OUTSET = PIN3_bm;
	PORTD.DIRSET = PIN3_bm;
	PORTD.DIRCLR = PIN2_bm;

	/* Configure baud rate. to 94744 */
	USARTD0.BAUDCTRLA = (uint8_t)BSEL;
	USARTD0.BAUDCTRLB = (uint8_t)((BSCALE << 4)|(BSEL >> 8));

	/* Configure remainder of serial protocol. */
	//set to odd parity, 8 bits, 1 stop bit in PORTD.CTRLC
	USARTD0.CTRLC =	(USART_CMODE_ASYNCHRONOUS_gc |
	USART_PMODE_ODD_gc	|
	USART_CHSIZE_8BIT_gc)	&
	~USART_SBMODE_bm;


	/* Enable receiver and/or transmitter systems. */
	USARTD0.CTRLB = USART_RXEN_bm | USART_TXEN_bm;

	/* Enable interrupt (optional). */
	/* USARTD0.CTRLA = USART_RXCINTLVL_MED_gc; */
}

/*****************************************************
 * Name: usartd0_init
 * Purpose: Initialize USARTD0 for 94 744 bps with odd
 *          parity and 8-bit data format.
 * Inputs:  None
 * Outputs: None
 *****************************************************/

void usartd0_out_char(char c)
{
	while(!(USARTD0.STATUS & USART_DREIF_bm));
	USARTD0.DATA = c;
}

/*****************************************************
 * Name: adc_init
 * Purpose: Initialize ADCA for differential sampling and
 *          start conversions triggered by Event Ch 0.
 * Inputs:  None
 * Outputs: None
 *****************************************************/
```

```
void adc_init(void)
{
	// set 12-bit signed mode, right adjusted
	// resolution = 00, convmode = 1 (signed)  ? CTRLB
	// normal mode (freerun = 0)
	ADCA.CTRLB = ADC_CONMODE_bm | ADC_RESOLUTION_12BIT_gc;

	// 2.5 V voltage reference (AREFB)
	// REFSEL = AREFB ? REFCTRL
	ADCA.REFCTRL = ADC_REFSEL_AREFB_gc;

	// prescaler = div64 (32 MHz / 64 = 500 kHz)
	ADCA.PRESCALER = ADC_PRESCALER_DIV64_gc;

	// configure ADCA.CH0.CTRL
	// differential mode, gain = 1×
	//ADCA.CH0.CTRL = ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_1X_gc;
	//ADCA.CH0.CTRL = ADC_CH_INPUTMODE_DIFF_gc;
	ADCA.CH0.CTRL = ADC_CH_INPUTMODE_DIFF_gc | ADC_CH_GAIN_1X_gc;


	// configure ADCA.CH0.MUXCTRL
	// positive input = PA1 (CDS+), negative input = PA6 (CDS?)
	ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN1_gc | ADC_CH_MUXNEG_PIN6_gc;

	//enable ADC interrupts in ADCA.CH0.INTCTRL
	ADCA.CH0.INTCTRL = ADC_CH_INTLVL_LO_gc;
	//use EVCTRL register in ADC module
	//make ADC conversion start when event channel 0 is triggered
	// EVSEL to ch0
	//event action to start a single conversion on CH0
	//sweep ch0 only
	ADCA.EVCTRL = ADC_EVACT_CH0_gc | ADC_EVSEL_0123_gc | ADC_SWEEP_0_gc;
	//ADCA.EVCTRL = (0 << 3) | (1 << 0);

	// enable ADCA
	ADCA.CTRLA = ADC_ENABLE_bm;

}

/*****************************************************
 * Name: main
 * Purpose: Initialize modules and continuously send raw
 *          ADC results to SerialPlot at 140 Hz.
 * Inputs:  None
 * Outputs: None
 *****************************************************/

int main(void)
{
	//init system
	//call tcc0_init();
	tcc0_init();
	//call adc_init();
	adc_init();
	usartd0_init();
	//configure BLUE_LED
	PORTD.OUTCLR = PIN6_bm;
	PORTD.DIRSET = PIN6_bm;
	//enable pmic and global interrupts
```

```
	PMIC.CTRL = PMIC_LOLVLEN_bm;
	sei();

	//ADCA.CH0.CTRL |= ADC_CH_START_bm;


	//infinite loop
	while(1)
	{
		//check if global flag is set
		//if set clear it and output voltage to serial terminal (call send voltage)
		if (new_data_flag == 1)
		{
			new_data_flag = 0;
			//send_voltage(adc_result); for other main
			//output voltage to serial plot
			//outchar adc_result low byte (result & 0x00FF)
			//outchar adc_result high byte (result >> 8 & 0xFF)
			usartd0_out_char((uint8_t)(adc_result & 0xFF));
			usartd0_out_char((uint8_t)((adc_result >> 8) & 0xFF));


		}

	}



}

/*****************************************************
 * Name: ISR(ADCA_CH0_vect)
 * Purpose: Handle ADC conversion complete interrupt,
 *          store result, and toggle the BLUE_PWM LED.
 * Inputs:  None
 * Outputs: None
 *****************************************************/

ISR(ADCA_CH0_vect)
{
	//clear interrupt flag
	ADCA.CH0.INTFLAGS = ADC_CH_CHIF_bm;
	//save result in signed 16bit int
	adc_result = ADCA.CH0RES;
	//set global flag
	new_data_flag = 1;
	//toggle blue_PWM LEd
	PORTD.OUTTGL = PIN6_bm;
}

/*****************************************************
 * Name: send_voltage
 * Purpose: (Unused here) Converts and formats ADC result
 *          as voltage for serial output in other sections.
 * Inputs:  adc_result – 12-bit signed ADC value
 * Outputs: None
 *****************************************************/

void send_voltage(int16_t adc_result)
{
	float voltage = 0;
```

```
float abs_voltage = 0;
int16_t int_part, dec1, dec2;
//char hex_digits[4];
uint16_t adc_unsigned;
//use equation of a line to determine the Analog value from the digital val VA = f(D)
voltage = (5.0/4095) * adc_result + 6.105e-4;
//if result < 0 outchar '-'
//else outchar '+'
if(voltage < 0)
{
	usartd0_out_char('-');
	abs_voltage = -voltage;
}
else
{
	usartd0_out_char('+');
	abs_voltage = voltage;
}


//Example of what to do for decimal value:
//Pi = 3.14159…//variable holds original value
//Int1 = (int) Pi = 3 ? 3 is the first digit of Pi
//Transmit the ASCII value of Int1 and then '.'
//Pi2 = 10*(Pi - Int1) = 1.4159…
//Int2 = (int) Pi2 = 1 ? 1 is the second digit of Pi
//Transmit the ASCII value of Int2 digit
//Pi3 = 10*(Pi2 – Int2) = 4.159…
//Int3 = (int) Pi3 = 4 ? 4 is the third digit of Pi
//Transmit the ASCII value of Int3 digit, then a space, and then a 'V'
int_part = (int16_t)abs_voltage;
abs_voltage = (abs_voltage - int_part) * 10;
dec1 = (int16_t)abs_voltage;
abs_voltage = (abs_voltage - dec1) * 10;
dec2 = (int16_t)abs_voltage;

usartd0_out_char(int_part + '0');
usartd0_out_char('.');
usartd0_out_char(dec1 + '0');
usartd0_out_char(dec2 + '0');
usartd0_out_char(' ');
usartd0_out_char('V');
usartd0_out_char(' ');
usartd0_out_char('(');
usartd0_out_char('0');
usartd0_out_char('x');

//output hex value
adc_unsigned = (uint16_t)(adc_result & 0x0FFF);

//shift the num left 4 to get high nibble
//if the num is < 10 add 0x30 and outchar
//else num is A-F so subtract 10 and add "A" and outchar
//mask the high bit then do the same
for (int shift = 8; shift >= 0; shift -= 4)
{
	uint8_t nibble = (adc_unsigned >> shift) & 0x0F;

	if (nibble < 10)
	usartd0_out_char('0' + nibble);
	else
```

```
            usartd0_out_char((nibble - 10) + 'A');
        }

        //output ‘)‘
        //output CR and LF
        usartd0_out_char(')');
        usartd0_out_char(CR);
        usartd0_out_char(LF);
}
```

Lab7_5.c

```
/*
 * lab7_5.c
 *
 * Created: 11/8/2025 3:55:46 PM
 *  Author: arist
 */

/*****************************************************
 * Lab 7, Section 22
 * Name: Arion Stern
 * Class #: 11303
 * PI Name: Ian Santamauro
 * Description:
 *   Adds UART control to switch ADC inputs between the
 *   CdS cell and function generator using 'C' or 'F'
 *   commands, while streaming 16-bit data to SerialPlot
 *   at 140 Hz.
 *****************************************************/

#include <avr/io.h>
#include <avr/interrupt.h>

volatile int16_t adc_result = 0xFFFF;
volatile uint8_t new_data_flag = 0;

/* At 2 MHz SYSclk, 41 BSEL, -7 BSCALE corresponds to 94744 bps */
#define BSEL     (41)
#define BSCALE   (-7)
#define CR (13)
#define LF (10)

//prototypes
void send_voltage(int16_t adc_result);
void tcc0_init(void);
void usartd0_init(void);
void usartd0_out_char(char c);
void adc_init(void);

/*****************************************************
 * Name: tcc0_init
 * Purpose: Configure TCC0 to overflow at 140 Hz and
 *          generate an event on channel 0.
 * Inputs:  None
 * Outputs: None
 *****************************************************/

void tcc0_init(void)
```

```
{
	//set count to 0 in CNT
	TCC0.CNT = 0;
	//set period to correspond to 0.25 seconds using pre of 64 in PER
	//per = clk/pre * dur = 7812
	//TCC0.PER = 7812 * 4; // 1 second now

	TCC0.PER = 223; // 140hz now

	//enable low level interrupts in INTCTRLA DONT DO THIS IT BREAKS THE EVENT SYSTEM
	//TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;
	//configure CH0MUX
	// CH0MUX = TCC0 overflow (1100 _000)
	EVSYS.CH0MUX = EVSYS_CHMUX_TCC0_OVF_gc;
	//start TCC0
	TCC0.CTRLA = TC_CLKSEL_DIV64_gc;
}

/*****************************************************
 * Name: usartd0_init
 * Purpose: Initialize USARTD0 for 94 744 bps, odd parity,
 *          and enable receiver interrupt for command input.
 * Inputs:  None
 * Outputs: None
 *****************************************************/

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

	/* Configure baud rate. to 94744 */
	USARTD0.BAUDCTRLA = (uint8_t)BSEL;
	USARTD0.BAUDCTRLB = (uint8_t)((BSCALE << 4)|(BSEL >> 8));

	/* Configure remainder of serial protocol. */
	//set to odd parity, 8 bits, 1 stop bit in PORTD.CTRLC
	USARTD0.CTRLC =      (USART_CMODE_ASYNCHRONOUS_gc |
	USART_PMODE_ODD_gc    |
	USART_CHSIZE_8BIT_gc)       &
	~USART_SBMODE_bm;


	/* Enable receiver and/or transmitter systems. */
	USARTD0.CTRLB = USART_RXEN_bm | USART_TXEN_bm;

	/* Enable interrupt (optional). */
	USARTD0.CTRLA = USART_RXCINTLVL_LO_gc;
}

/*****************************************************
 * Name: usartd0_out_char
 * Purpose: Send one byte through USARTD0.
 * Inputs:  c - character to send
 * Outputs: None
 *****************************************************/

void usartd0_out_char(char c)
{
```

```
	while(!(USARTD0.STATUS & USART_DREIF_bm));
	USARTD0.DATA = c;
}


/*****************************************************
 * Name: adc_init
 * Purpose: Initialize ADCA for differential sampling and
 *          start conversions triggered by Event Ch 0.
 * Inputs:  None
 * Outputs: None
 *****************************************************/
void adc_init(void)
{
	// set 12-bit signed mode, right adjusted
	// resolution = 00, convmode = 1 (signed)  ? CTRLB
	// normal mode (freerun = 0)
	ADCA.CTRLB = ADC_CONMODE_bm | ADC_RESOLUTION_12BIT_gc;

	// 2.5 V voltage reference (AREFB)
	// REFSEL = AREFB ? REFCTRL
	ADCA.REFCTRL = ADC_REFSEL_AREFB_gc;

	// prescaler = div64 (32 MHz / 64 = 500 kHz)
	ADCA.PRESCALER = ADC_PRESCALER_DIV64_gc;

	// configure ADCA.CH0.CTRL
	// differential mode, gain = 1×
	//ADCA.CH0.CTRL = ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_1X_gc;
	//ADCA.CH0.CTRL = ADC_CH_INPUTMODE_DIFF_gc;
	ADCA.CH0.CTRL = ADC_CH_INPUTMODE_DIFF_gc | ADC_CH_GAIN_1X_gc;


	// configure ADCA.CH0.MUXCTRL
	// positive input = PA1 (CDS+), negative input = PA6 (CDS?)
	ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN1_gc | ADC_CH_MUXNEG_PIN6_gc;

	//enable ADC interrupts in ADCA.CH0.INTCTRL
	ADCA.CH0.INTCTRL = ADC_CH_INTLVL_LO_gc;
	//use EVCTRL register in ADC module
	//make ADC conversion start when event channel 0 is triggered
	// EVSEL to ch0
	//event action to start a single conversion on CH0
	//sweep ch0 only
	ADCA.EVCTRL = ADC_EVACT_CH0_gc | ADC_EVSEL_0123_gc | ADC_SWEEP_0_gc;
	//ADCA.EVCTRL = (0 << 3) | (1 << 0);

	// enable ADCA
	ADCA.CTRLA = ADC_ENABLE_bm;

}


/*****************************************************
 * Name: main
 * Purpose: Initialize modules and continuously stream
 *          ADC data to SerialPlot while allowing UART
 *          commands to change input source.
 * Inputs:  None
 * Outputs: None
 *****************************************************/
```

```
int main(void)
{
	//init system
	//call tcc0_init();
	tcc0_init();
	//call adc_init();
	adc_init();
	usartd0_init();
	//configure BLUE_LED
	PORTD.OUTCLR = PIN6_bm;
	PORTD.DIRSET = PIN6_bm;
	//enable pmic and global interrupts
	PMIC.CTRL = PMIC_LOLVLEN_bm;
	sei();

	//ADCA.CH0.CTRL |= ADC_CH_START_bm;


	//infinite loop
	while(1)
	{
		//check if global flag is set
		//if set clear it and output voltage to serial terminal (call send voltage)
		if (new_data_flag == 1)
		{
			new_data_flag = 0;
			//send_voltage(adc_result); for other main
			//output voltage to serial plot
			//outchar adc_result low byte (result & 0x00FF)
			//outchar adc_result high byte (result >> 8 & 0xFF)
			usartd0_out_char((uint8_t)(adc_result & 0xFF));
			usartd0_out_char((uint8_t)((adc_result >> 8) & 0xFF));


		}

	}



}


/*****************************************************
 * Name: ISR(ADCA_CH0_vect)
 * Purpose: Handle ADC conversion complete interrupt,
 *          store result, and toggle the BLUE_PWM LED.
 * Inputs:  None
 * Outputs: None
 *****************************************************/

ISR(ADCA_CH0_vect)
{
	//clear interrupt flag
	ADCA.CH0.INTFLAGS = ADC_CH_CHIF_bm;
	//save result in signed 16bit int
	adc_result = ADCA.CH0RES;
	//set global flag
	new_data_flag = 1;
	//toggle blue_PWM LEd
```

```
	PORTD.OUTTGL = PIN6_bm;
}

/*****************************************************
 * Name: send_voltage
 * Purpose: Convert ADC result to a formatted voltage
 *          string (decimal and hex) for UART output.
 * Inputs:  adc_result – 12-bit signed ADC value
 * Outputs: None
 *****************************************************/

void send_voltage(int16_t adc_result)
{
	float voltage = 0;
	float abs_voltage = 0;
	int16_t int_part, dec1, dec2;
	//char hex_digits[4];
	uint16_t adc_unsigned;
	//use equation of a line to determine the Analog value from the digital val VA = f(D)
	voltage = (5.0/4095) * adc_result + 6.105e-4;
	//if result < 0 outchar '-'
	//else outchar '+'
	if(voltage < 0)
	{
		usartd0_out_char('-');
		abs_voltage = -voltage;
	}
	else
	{
		usartd0_out_char('+');
		abs_voltage = voltage;
	}


	//Example of what to do for decimal value:
	//Pi = 3.14159…//variable holds original value
	//Int1 = (int) Pi = 3 ? 3 is the first digit of Pi
	//Transmit the ASCII value of Int1 and then '.'
	//Pi2 = 10*(Pi - Int1) = 1.4159…
	//Int2 = (int) Pi2 = 1 ? 1 is the second digit of Pi
	//Transmit the ASCII value of Int2 digit
	//Pi3 = 10*(Pi2 – Int2) = 4.159…
	//Int3 = (int) Pi3 = 4 ? 4 is the third digit of Pi
	//Transmit the ASCII value of Int3 digit, then a space, and then a 'V'
	int_part = (int16_t)abs_voltage;
	abs_voltage = (abs_voltage - int_part) * 10;
	dec1 = (int16_t)abs_voltage;
	abs_voltage = (abs_voltage - dec1) * 10;
	dec2 = (int16_t)abs_voltage;

	usartd0_out_char(int_part + '0');
	usartd0_out_char('.');
	usartd0_out_char(dec1 + '0');
	usartd0_out_char(dec2 + '0');
	usartd0_out_char(' ');
	usartd0_out_char('V');
	usartd0_out_char(' ');
	usartd0_out_char('(');
	usartd0_out_char('0');
	usartd0_out_char('x');
```

```
	//output hex value
	adc_unsigned = (uint16_t)(adc_result & 0x0FFF);

	//shift the num left 4 to get high nibble
	//if the num is < 10 add 0x30 and outchar
	//else num is A-F so subtract 10 and add “A” and outchar
	//mask the high bit then do the same
	for (int shift = 8; shift >= 0; shift -= 4)
	{
		uint8_t nibble = (adc_unsigned >> shift) & 0x0F;

		if (nibble < 10)
		usartd0_out_char('0' + nibble);
		else
		usartd0_out_char((nibble - 10) + 'A');
	}

	//output ‘)‘
	//output CR and LF
	usartd0_out_char(')');
	usartd0_out_char(CR);
	usartd0_out_char(LF);
}

/*****************************************************
 * Name: ISR(USARTD0_RXC_vect)
 * Purpose: Handle received UART characters; switch ADC
 *          inputs to CdS ('C') or Function Gen ('F').
 * Inputs:  None
 * Outputs: None
 *****************************************************/

ISR(USARTD0_RXC_vect)
{
	//clear the interrupt flag (happens automatically when you read from data in the ISR
	//read the character from the data register
	char received_char = USARTD0.DATA;
	//if the character is ‘C’ make configure the ADCA.CH0.MUXCTRl for CDS+ (pa1) and CDS- (pa6)
	if(received_char == 'C')
		ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN1_gc | ADC_CH_MUXNEG_PIN6_gc;
	//if the char is ‘F’ make configure ADCA.CH0.MUXCTRL for AIN+ (pa5) and AIN- (pa4)
	else if(received_char == 'F')
	ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN5_gc | ADC_CH_MUXNEG_PIN4_gc;
	//else don’t change it
}
```

APPENDIX

Supporting ASM/C Code and Additional Information

- Included/Referenced Files and Headers:
- Additional Screenshots/Images:

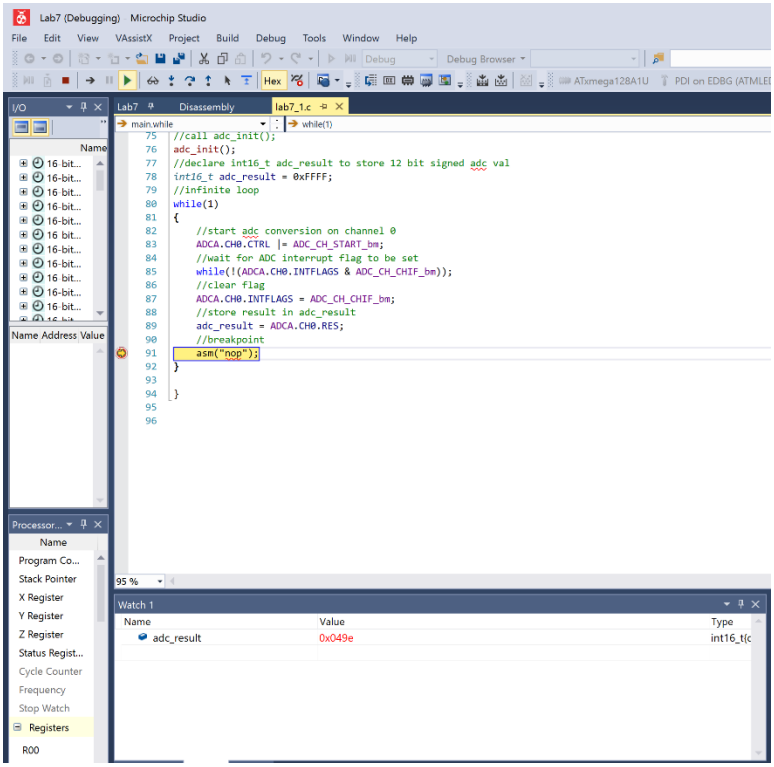


Figure 1: Debug session in Microchip Studio confirming correct ADC initialization and conversion. The variable adc_result holds a signed 12-bit value (0x049E) corresponding to the CdS photoresistor voltage, verifying proper differential ADC operation.

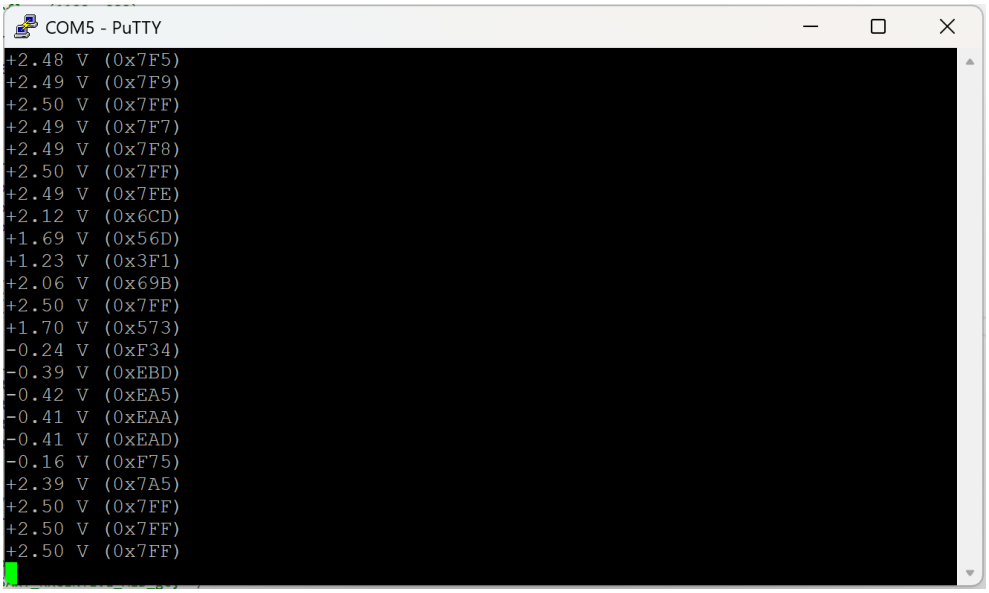


Figure 2: Serial terminal output from the voltmeter program displaying signed voltage readings from the CdS photoresistor. Each line shows the measured voltage in decimal volts and its corresponding 12-bit hexadecimal ADC value, confirming correct conversion and formatting.

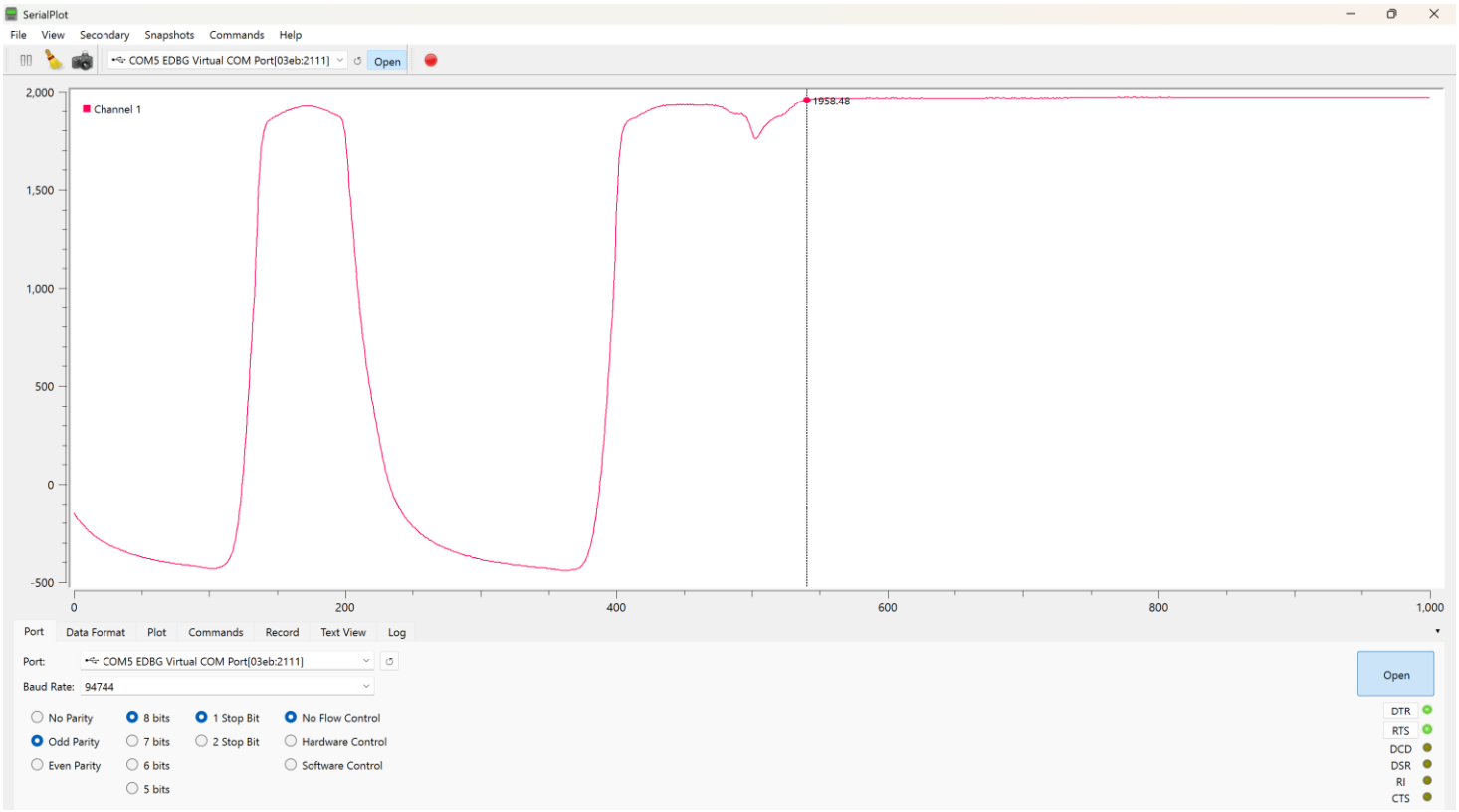


Figure 3: SerialPlot display showing 12-bit ADC samples from the CdS photoresistor while alternating between exposing it to natural room lighting and covering the sensor. The waveform demonstrates correct 140 Hz sampling, stable UART transmission, and sensitivity to changing illumination.

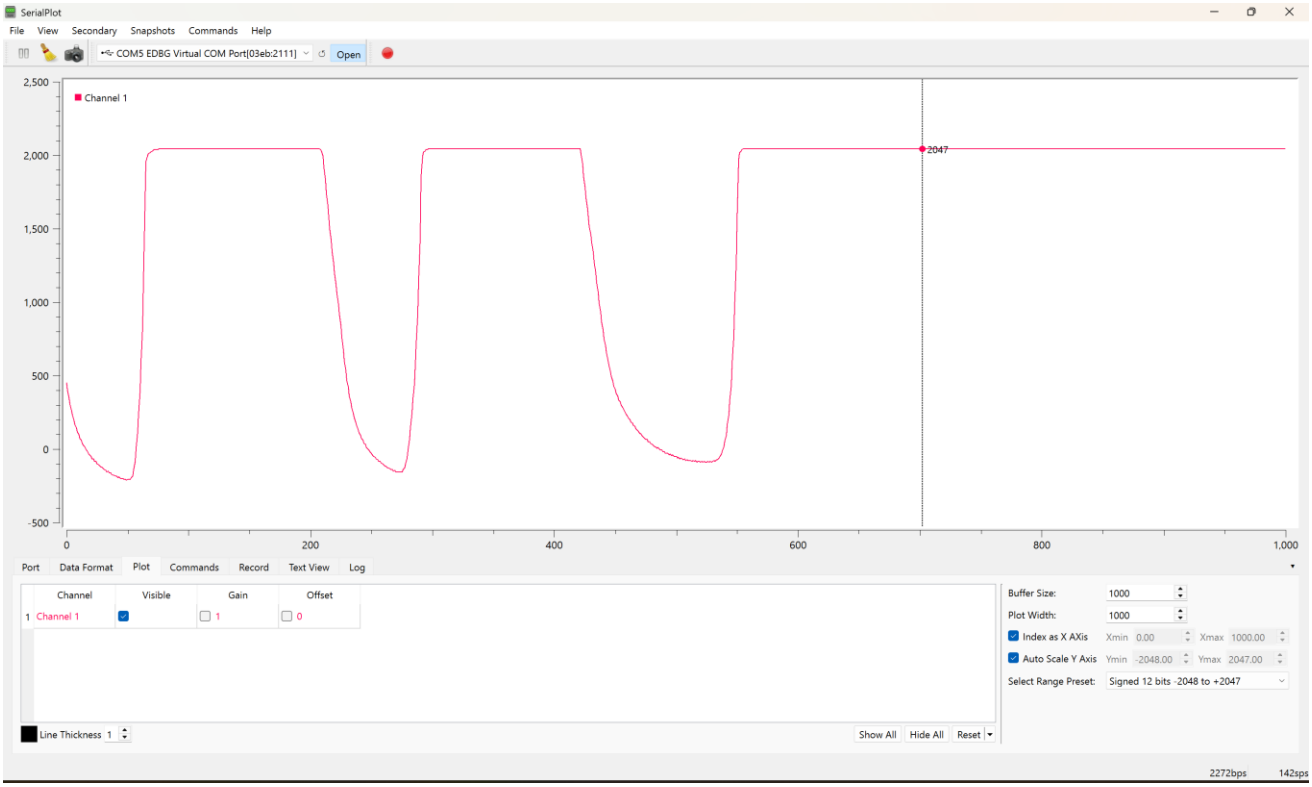


Figure 4: SerialPlot waveform captured while alternately shining a flashlight on the CdS sensor and covering it. The large amplitude swings confirm proper 12-bit ADC response and real-time visualization of light-intensity variations.

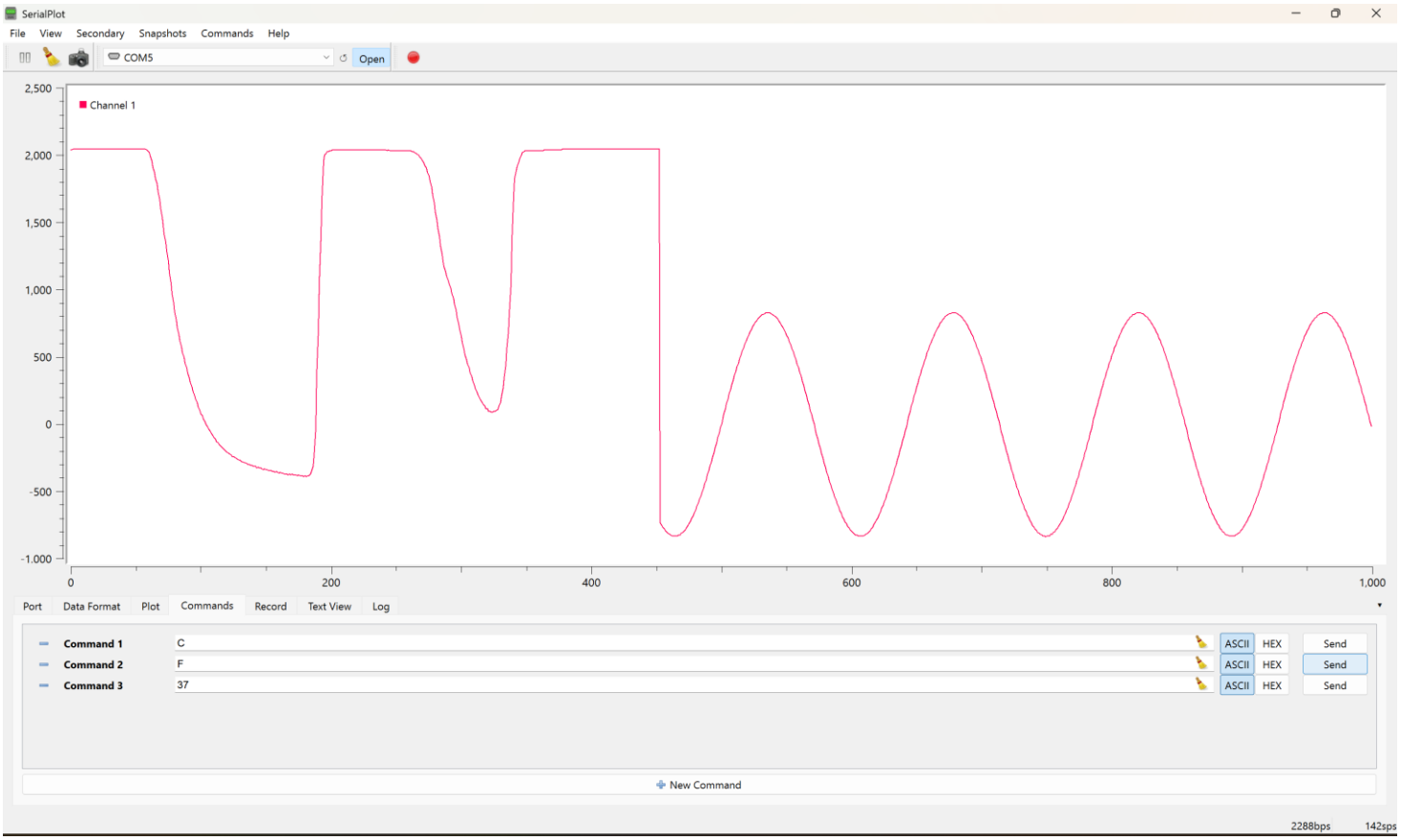


Figure 5: SerialPlot output showing 12-bit ADC data before and after sending UART commands 'C' and 'F'. The initial section displays the CdS photoresistor waveform responding to light changes, while the latter section shows a 5 V sine wave (0 V offset) from the DAD function generator. This confirms correct UART command handling and ADC channel switching between analog inputs.