REQUIREMENTS NOT MET

N/A

PROBLEMS ENCOUNTERED

I did encounter a couple small problems throughout the lab but that is normal. However, in the end of the lab, I did have a couple bugs that didn't seem to happen consistently, and I could not figure them out. For example, sometimes my serial plot will randomly be very sensitive, and other times it acts completely normal. But overall, I feel like everything ended up alright.

FUTURE WORK/APPLICATIONS

This has been cool working with the ADC because it helps me understand a lot more about how we transfer and manipulate information. I feel like any form of communication that we learn throughout this course is super important so I don't doubt that I will use this again in the future.

PRE-LAB EXERCISES

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

Because the CDS photocell outputs are connected to PORTA and we are trying to read the CDS photocells.

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? Why or why not?

I believe so. I am honestly not that sure, but I would say yes because those different configurations have to do with the channel and not the ports. Because they don't affect the ports, it wouldn't cause any problems if we changed the configuration.

iii. What would the main benefit be for 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 resolution? If so, explain.

So a higher resolution allows you to be more accurate in your reading. Having 12 bits instead of 8 would let you get closer to the actual value of the input. The only reason I would see to use 8 instead of 12 is if you want a wider range of voltages on purpose. Like say you want an input to fall into one of a handful of bigger voltages. Then less bits would be better.

iv. What is the decimal voltage value that is equivalent to a 12-bit signed result of 0x073?

$$Res = \frac{VINP - VINN}{VREF} * GAIN * (TOP + 1)$$

VINP = ?
Res = 0x073 = 115
VINN = 0
GAIN = 1
TOP + 1 = 2028

Plugging in we get that VINP equals around 0.1404V

PSEUDOCODE/FLOWCHARTS

N/A

PROGRAM CODE

SECTION 1 (consisted of making adc_init, and a main)

```
int main(void)
{
	/* Initialize Channel 0 of the ADCA module. */
	adc_init();

	/* Temporary 8-bit signed variable to store the result. */
	int16_t temp;

    while (1)
    {
		/* Start a conversion. */
		ADCA.CH0.CTRL |= ADC_CH_START_bm;

		/* Wait for the conversion to be completed. */
		while(!(ADCA.CH0.INTFLAGS & ADC_CH_CHIF_bm));

		/* Read the 12-bit result. */
		temp = ADCA.CH0.RES;

		/* Clear interrupt flag. */
		ADCA.CH0.INTFLAGS = ADC_CH_CHIF_bm;

    }
}



void adc_init(void)
{

	/* Signed, 12-bit conversion results. */
	ADCA.CTRLB = ADC_CONMODE_bm | ADC_RESOLUTION_12BIT_gc;

	/* Use external reference voltage on Port B. See doc8385 for reference pin. */
	ADCA.REFCTRL = ADC_REFSEL_AREFB_gc;

	/* Single-ended mode. */
	ADCA.CH0.CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc;

	/* Measure voltage on PA1 and PA6. */
	ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN1_gc | ADC_CH_MUXNEG_PIN6_gc;

	/* Enable the ADC. */
	ADCA.CTRLA = ADC_ENABLE_bm;
}
```

SECTION 2 (consisted of making tcc0_init, and interrupt for that tc and modifying previous code)

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


int main(void)
{
	/* Initialize I/o */
	io_init();

	/* Initialize Interrupts */
	intr_init();

	/* Initialize Channel 0 of the ADCA module. */
	adc_init();

	/* Initialize TC */
	tcc0_init();



    while (1)
    {
    }
}


void adc_init(void)
{

	/* Signed, 12-bit conversion results. */
	ADCA.CTRLB = ADC_CONMODE_bm | ADC_RESOLUTION_12BIT_gc;

	/* Use external reference voltage on Port B. See doc8385 for reference pin. */
	ADCA.REFCTRL = ADC_REFSEL_AREFB_gc;

	/* Single-ended mode. */
	ADCA.CH0.CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc;

	/* Measure voltage on PA1 and PA6. */
	ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN1_gc | ADC_CH_MUXNEG_PIN6_gc;

	/* Enable ADC interrupt to be triggered when a conversion is complete */
	ADCA.CH0.INTFLAGS = ADC_CH_CHIF_bm;

	/* Make an ADC conversion start when Event Channel 0 is triggered */
	ADCA.EVCTRL = ADC_EVACT1_bm;

	/* Enable the ADC. */
	ADCA.CTRLA = ADC_ENABLE_bm;
}


ISR(TCC0_OVF_vect)
{
	/* Wait for the conversion to be completed. */
	while(!(ADCA.CH0.INTFLAGS & ADC_CH_CHIF_bm));
```

```
	/* Save result into a signed 16-bit integer variable */
	int16_t temp = ADCA.CH0.RES;

	/* Toggle GREEN_PWM led */
	PORTD.OUTTGL = (1<<5);

	TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;
}
```

SECTION 3 (consisted of making usartd0_init and updating previous code)

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

int16_t temp;
//bool flag = false;


int main(void)
{
	/* Initialize I/o */
	io_init();

	/* Initialize Interrupts */
	intr_init();

	/* Initialize Channel 0 of the ADCA module. */
	adc_init();

	/* Initialize TC */
	tcc0_init();

	/* Initialize USART */
	usartd0_init();



    while (1)
    {
		if(GPIO0 == 1)
		{
			GPIO0 = 0;

			//temp = our value in 16 bits, we need 12
			//we can give USART 8 bits at the same time.

			int16_t msb = 1 << 11;
			if(temp & msb)
			{
				out_char('-');
			}
			else
			{
				out_char('+');
			}

			float voltage = ((temp * 2.5) /2048);

			/* Now do the algorithm given to us */
			int Int1 = (int) voltage;
				out_number(Int1);
				out_char('.');
			float voltage2 = 10*(voltage - Int1);
			int Int2 = (int) voltage2;
				out_number(Int2);
			float voltage3 = 10*(voltage2 - Int2);
			int Int3 = (int) voltage3;
				out_number(Int3);
				out_char(' ');
```

```
                        out_char('V');
                        out_char(' ');
                        out_char('(');
                        out_char('0');
                        out_char('x');

                        /* now find the hex value */
                        int high_byte = temp >> 8;
                        int med_byte = ((temp >> 4) & 0x0F);
                        int low_byte = temp & 0x0F;

                        out_hex(high_byte);
                        out_hex(med_byte);
                        out_hex(low_byte);

                        out_char(')');
                        out_char('\n');
                        out_char('\r');
                }
        }
}



void adc_init(void)
{

        /* Signed, 12-bit conversion results. */
        ADCA.CTRLB = ADC_CONMODE_bm | ADC_RESOLUTION_12BIT_gc;

        /* Use external reference voltage on Port B. See doc8385 for reference pin. */
        ADCA.REFCTRL = ADC_REFSEL_AREFB_gc;

        /* Single-ended mode. */
        ADCA.CH0.CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc;

        /* Measure voltage on PA1 and PA6. */
        ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN1_gc | ADC_CH_MUXNEG_PIN6_gc;

        /* Enable ADC interrupt to be triggered when a conversion is complete */
        ADCA.CH0.INTFLAGS = ADC_CH_CHIF_bm;

        /* Make an ADC conversion start when Event Channel 0 is triggered */
        ADCA.EVCTRL = ADC_EVACT1_bm;

        /* Enable the ADC. */
        ADCA.CTRLA = ADC_ENABLE_bm;
}


void usartd0_init()
{
        //BSEL = 1
        //BSCALE = -4

        /* Configure the UART frame. */
        USARTD0.CTRLC = ( USART_CMODE_ASYNCHRONOUS_gc | USART_PMODE_DISABLED_gc | USART_CHSIZE_8BIT_gc );
//need to use 8 bit

        /* Initialize the baud rate */
        USARTD0.BAUDCTRLA = 1;
```

```
	USARTD0.BAUDCTRLB = (-4<<4); //Should really or the high part of 1, but that is 0

	/* Enable the transmitter */
	USARTD0.CTRLB = USART_TXEN_bm;
}


ISR(TCC0_OVF_vect)
{
	/* Wait for the conversion to be completed. */
	while(!(ADCA.CH0.INTFLAGS & ADC_CH_CHIF_bm));

	/* Save result into a signed 16-bit integer variable */
	temp = ADCA.CH0.RES;

	/* Update my flag */
	//flag = true;
	GPIO0 = 1;

	/* Toggle GREEN_PWM led */
	PORTD.OUTTGL = (1<<5);

	TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;
}
```

SECTION 4 (Here we mainly modified code and made it work with serial plot)

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

int16_t temp;
//bool flag = false;


int main(void)
{
	/* Initialize I/o */
	io_init();

	/* Initialize Interrupts */
	intr_init();

	/* Initialize Channel 0 of the ADCA module. */
	adc_init();

	/* Initialize TC */
	tcc0_init();

	/* Initialize USART */
	usartd0_init();



    while (1)
    {
		if(GPIO0 == 1)
		{
			GPIO0 == 0;

			//temp = our value in 16 bits, we need 12
			//we can give USART 8 bits at the same time.

			int8_t high = temp >> 8;
			int8_t low = temp & 0x00FF;

			out_binary(low);
			out_binary(high);

		}
    }
}




void adc_init(void)
{

	/* Signed, 12-bit conversion results. */
	ADCA.CTRLB = ADC_CONMODE_bm | ADC_RESOLUTION_12BIT_gc;

	/* Use external reference voltage on Port B. See doc8385 for reference pin. */
	ADCA.REFCTRL = ADC_REFSEL_AREFB_gc;

	/* Single-ended mode. */
```

```
	ADCA.CH0.CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc;

	/* Measure voltage on PA1 and PA6. */
	ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN1_gc | ADC_CH_MUXNEG_PIN6_gc;

	/* Enable ADC interrupt to be triggered when a conversion is complete */
	ADCA.CH0.INTFLAGS = ADC_CH_CHIF_bm;

	/* Make an ADC conversion start when Event Channel 0 is triggered */
	ADCA.EVCTRL = ADC_EVACT1_bm;

	/* Enable the ADC. */
	ADCA.CTRLA = ADC_ENABLE_bm;
}

void usartd0_init()
{
	//BSEL = 1
	//BSCALE = -4

	/* Configure the UART frame. */
	USARTD0.CTRLC = ( USART_CMODE_ASYNCHRONOUS_gc | USART_PMODE_DISABLED_gc | USART_CHSIZE_8BIT_gc );
//need to use 8 bit

	/* Initialize the baud rate */
	USARTD0.BAUDCTRLA = 1;
	USARTD0.BAUDCTRLB = (-4<<4); //Should really or the high part of 1, but that is 0

	/* Enable the transmitter */
	USARTD0.CTRLB = ( USART_TXEN_bm | USART_RXEN_bm );
}



ISR(TCC0_OVF_vect)
{
	/* Wait for the conversion to be completed. */
	while(!(ADCA.CH0.INTFLAGS & ADC_CH_CHIF_bm));

	/* Save result into a signed 16-bit integer variable */
	temp = ADCA.CH0.RES;

	/* Update my flag */
	//flag = true;
	GPIO0 = 1;

	/* Toggle GREEN_PWM led */
	PORTD.OUTTGL = (1<<5);

	TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;
}
```

SECTION 5 (modify code and create the function to switch between reading two inputs)

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

int16_t temp;


int main(void)
{
	/* Initialize I/o */
	io_init();

	/* Initialize Channel 0 of the ADCA module. */
	adc_init();

	/* Initialize TC */
	tcc0_init();

	/* Initialize USART */
	usartd0_init();

	/* Initialize Interrupts */
	intr_init();

    while (1)
    {
		int8_t high = temp >> 8;
		int8_t low = temp & 0x00FF;

		out_binary(low); //do low first
		out_binary(high);
    }
}


/***************INITIALIZATIONS*************/

void adc_init(void)
{

	/* Signed, 12-bit conversion results. */
	ADCA.CTRLB = ADC_CONMODE_bm | ADC_RESOLUTION_12BIT_gc;

	/* Use external reference voltage on Port B. See doc8385 for reference pin. */
	ADCA.REFCTRL = ADC_REFSEL_AREFB_gc;

	/* Single-ended mode. */
	ADCA.CH0.CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc;

	/* Enable ADC interrupt to be triggered when a conversion is complete */
	ADCA.CH0.INTFLAGS = ADC_CH_CHIF_bm;

	/* Measure voltage on PA1 and PA6. */
	//ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN1_gc | ADC_CH_MUXNEG_PIN6_gc;

	/* Make an ADC conversion start when Event Channel 0 is triggered */
	ADCA.EVCTRL = ADC_EVACT1_bm;

	/* Enable the ADC. */
```

```
	ADCA.CTRLA = ADC_ENABLE_bm;
}


void usartd0_init()
{
	//BSEL = 1
	//BSCALE = -4

	/* Configure the UART frame. */
	USARTD0.CTRLC = ( USART_CMODE_ASYNCHRONOUS_gc | USART_PMODE_DISABLED_gc | USART_CHSIZE_8BIT_gc );
//need to use 8 bit

	/* Initialize the baud rate */
	USARTD0.BAUDCTRLA = 1;
	USARTD0.BAUDCTRLB = (-4<<4); //Should really or the high part of 1, but that is 0

	/* Enable the transmitter and receiver */
	USARTD0.CTRLB = ( USART_TXEN_bm | USART_RXEN_bm );

	USARTD0.CTRLA = 0b010000; //make low level. couldn't find the bm

}


/******************ISRs************/

ISR(TCC0_OVF_vect)
{
	/* Wait for the conversion to be completed. */
	while(!(ADCA.CH0.INTFLAGS & ADC_CH_CHIF_bm));

	/* Save result into a signed 16-bit integer variable */
	temp = ADCA.CH0.RES;

	/* Toggle GREEN_PWM led */
	PORTD.OUTTGL = (1<<5);

	TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;
}


ISR(USARTD0_RXC_vect)
{
	while(!(USARTD0.STATUS & USART_DREIF_bm)); //wait

	if(USARTD0.DATA == 'C')
	{
		/* Measure voltage on PA1 and PA6. */
		ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN1_gc | ADC_CH_MUXNEG_PIN6_gc;
	}
	else if(USARTD0.DATA == 'J')
	{
		/* Measure voltage on PA4 and PA5. */
		ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN4_gc | ADC_CH_MUXNEG_PIN5_gc;
	}
}
```

APPENDIX

Here are some functions that are in multiple files and are not really changed in between each part. They are put separate so there is not a bunch of repeated code above:

```
/*******************DIFFERENT OUT FUNCTIONS***************/
/* there is probably a good way to do all of these in one, but
I kept having problems */
void out_number(int character)
{
	char convert = '0' + character;

	USARTD0.DATA = convert;
	while(!(USARTD0.STATUS & USART_DREIF_bm)); //wait
}

void out_char(char character)
{

	USARTD0.DATA = character;
	while(!(USARTD0.STATUS & USART_DREIF_bm)); //wait
}

/* I know there is probably a better way to print hex values
	but I don't know what else to do and I am running out of time.
	this is better than no solution */
void out_hex(int character)
{
	switch(character)
	{
		case 10:
			out_char('A');
			break;
		case 11:
			out_char('B');
			break;
		case 12:
			out_char('C');
			break;
		case 13:
			out_char('D');
			break;
		case 14:
			out_char('E');
			break;
		case 15:
			out_char('F');
			break;
		default:
			out_number(character);
	}

}
```

```
/*******************DIFFERENT INITIALIZATIONS***************/
void tcc0_init(void)
{
	TCC0.CNT = 0;
	TCC0.PER = (2000000/1) / 100;
	TCC0.CTRLA = TC_CLKSEL_DIV1_gc;

	/* Clear OVFIF to be safe */
	TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;

	EVSYS.CH0MUX = EVSYS_CHMUX_TCC0_OVF_gc;
}

void intr_init(void)
{
	/* Enable low level interrupts in the PMIC. */
	PMIC.CTRL = PMIC_LOLVLEN_bm;

	/* Enable interrupts globally. */
	sei();
}

void io_init(void)
{
	/* Set Green LED as output */
	PORTD.OUTSET = (1<<5) | (1<<3);
	PORTD.DIRSET = (1<<5) | (1<<3);
}
```