CG2111A: Engineering Principles and Practices II

Week 6: Studio 1: Arduino ADC Programming (GRADED LAB)

Objectives:

1. Understand how the Atmega328's ADC peripheral block needs to be configured.
2. Develop Low-Level (Bare-Metal) code for the ADC module for both the Polling and Interrupt Modes of Operation.
3. Experiment with other Analog Sensors (e.g. IR) for the ADC control.

Equipment Needed:

1. Laptop with Arduino IDE installed
2. Arduino Uno Board + Prototyping Board
3. 220, 1.2k, 12k, 22k and 39k Resistor
4. Red LED (x1)
5. Flex Sensor

LAB REPORT:

- Remember to show any necessary workings.
- Code must have proper comments.

1. Introduction

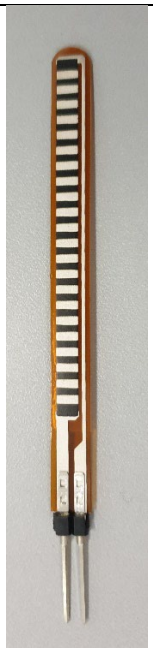
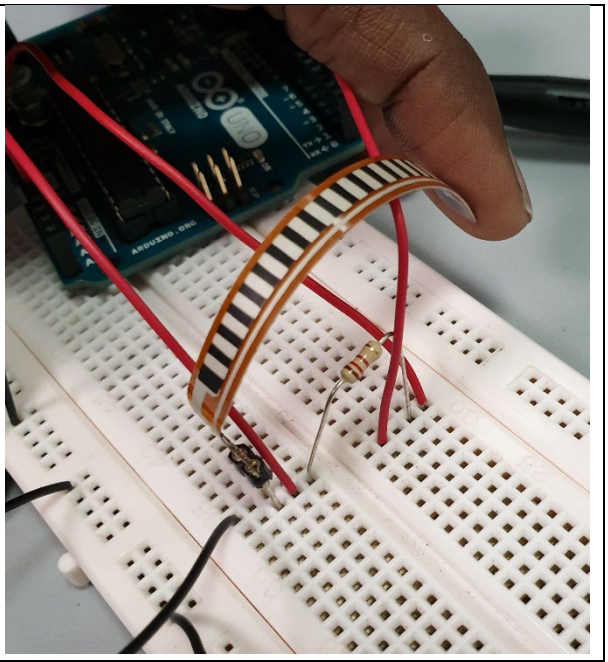
In EPP1, you dealt with Analog Devices in the MBot in the form of IR sensors in order to detect the right and left walls. If you recall, you used the built-in adc library calls to get the sensor value. In this studio, you are going to be using the same ADC modules, but you are going to learn how to program the module bare-metal.

In the e-lecture, https://youtu.be/abF4Z8UQXZE, you should have gotten an idea on how the ADC module in the microcontroller helps us to achieve our objective. By the end of this studio, you should be able to configure the ADC module and use it to perform the necessary conversions.

2. Exercise A

Examine the Flex Sensor that you have given. In EPP1, you used the LDR, a device whose resistance changes with respect to the amount of light it is exposed to. The Flex Sensor is a device that changes its resistance based upon the amount of flex it experiences. Plug the Flex Sensor into the breadboard given.

The following images show the front, back and the way it should be bent (front facing outwards).

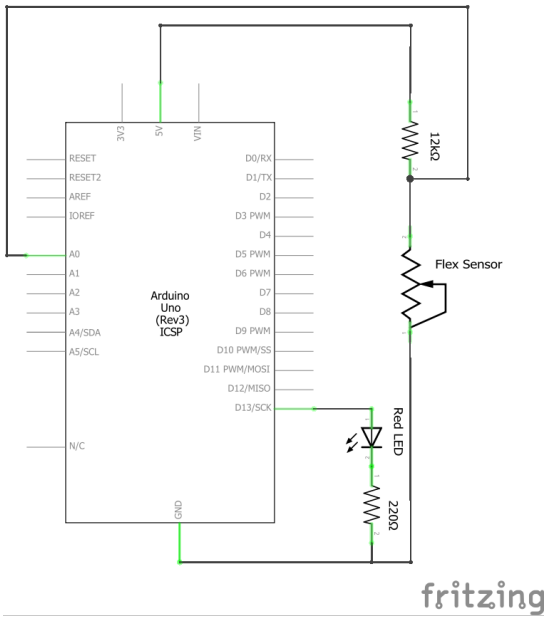
		
Front	**Back**	**Bent**

LAB REPORT

Q1. Measure the Resistance of the Sensor when it is placed straight (without any bending).

Q2. Measure the Resistance of the Sensor when it is bent. Note that in one direction of flexing, there will be very minimal change. In the opposite direction, the change will be very significant. We want to use the direction that shows a significant change in the resistance.

Connect your circuit as shown below.



A sample code for the ADC module is provided in Appendix_A. Compile and download it to your Uno board.

LAB REPORT

Q3. What is the adc reading when the Sensor is placed straight (without any bending). What is the voltage measurement at A0, using a multi-meter or scope?

Q4. What is the adc reading when the Sensor is bent.
What is the voltage measurement at A0, using a multi-meter or scope?

Q5. How can we relate this back to the voltage readings from Q3 and Q4.

Replace the 12k resistor with both 1.2k, 22k and 39k, and fill up the table below. The voltage is the measured value.

LAB REPORT

Q6.

	Flex Sensor Flat		Flex Sensor Bent			
	ADC	Voltage	ADC	Voltage	ADC Range	Voltage Range
1.2K						
12K						
22K						
39k						

LAB REPORT:

Q7. Give a reason why the values for 1.2k are greatly different from the rest of the resistors.

Select the 22k resistor to carry on with the rest of the studio session.

3. Exercise B

Modify the code to what is given in Appendix B and observe the effect on the LED.

LAB REPORT:

Q8. Give a reason for the LED's brightness to remain consistent regardless of the flex sensor being bent.

Q9. Measure the highest and lowest frequency of the signal applied to the LED.

4. Exercise C

In the earlier exercises, you implemented the ADC operation using the Polling approach. We will now implement it using Interrupts. Examine the code given in Appendix C. Complete the code in the ISR, compile and download the code onto the board.

LAB REPORT

Q10. Provide the code for the ISR.

LAB REPORT:

Q11. Measure the highest and lowest frequency of the signal applied to the LED.

Q12. Explain why the values for Q11 are different from the values for Q9.

5. Exercise D

With the Duty Cycle still set at 50%, we still don't see any change in the LED brightness. Now let's incorporate the PWM functionality from Week 5 Studio 2 so that we can change the brightness of the LED as we tune the potentiometer. The ADC value must be used to change the PWM Duty-cycle so that we can observe a change in the brightness of the LED. The LED brightness must go the full scale, from being very dim to very bright.

***Hint:** First remap the 10-bit ADC value to the 8-bit OCR0A value. Next expand the range of this value based on the flex sensors readings you got earlier.

LAB REPORT

Q13. Provide the full source code for your solution to Exercise D.

Summary

In this studio, you have learnt how to program the ADC module of the AT328P using the low-level registers. The ADC module is integral in dealing with many types of sensors and if you intend to use it for your robot, discuss your ideas with us so that we can provide the necessary guidance. Good Luck! ☺

APPENDIX A

```
#include "Arduino.h"

unsigned int adcvalue, loval, hival;

void setup() {

  // Clear Bit 0 (PRADC) to turn on power for the ADC module
  PRR &= ~(1 << PRADC);

  //ADEN = 1, ADPS[2:0] = 111 (Prescale = 128)
  ADCSRA |= ((1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0));

  //REFS[1:0] = 01 (AVcc as reference), MUX[2:0] = 000 (Channel 0)
  ADMUX |= ((1 << REFS0));

  // Set PortB Pin 5 as output
  DDRB |= (1 << DDB5);

  Serial.begin(9600);
}

void loop() {

  // ADSC = 1 (Start Conversion)
  ADCSRA |= (1 << ADSC);

  /*Wait for ADSC to go change to '0' to indicate that conversion is
complete*/
  while(ADCSRA & (1 << ADSC));

  loval = ADCL;
  hival = ADCH;
  adcvalue = (hival << 8) | loval;

  Serial.println(adcvalue);
}
```

APPENDIX B

```
#include "Arduino.h"

unsigned int adcvalue, loval, hival;

void setup() {

  // Clear Bit 0 (PRADC) to turn on power for the ADC module
  PRR &= ~(1 << PRADC);

  //ADEN = 1, ADPS[2:0] = 111 (Prescale = 128)
  ADCSRA |= ((1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0));

  //REFS[1:0] = 01 (AVcc as reference), MUX[2:0] = 000 (Channel 0)
  ADMUX |= ((1 << REFS0));

  // Set PortB Pin 5 as output
  DDRB |= (1 << DDB5);
}

void loop() {

  // ADSC = 1 (Start Conversion)
  ADCSRA |= (1 << ADSC);

  /*Wait for ADSC to go change to '0' to indicate that conversion is
complete*/
  while(ADCSRA & (1 << ADSC));

  loval = ADCL;
  hival = ADCH;
  adcvalue = (hival << 8) | loval;

  ledToggle();
  _delay_loop_2(adcvalue);
}

void ledToggle()
{
  PORTB ^= (1 << PORTB5);
}
```

APPENDIX C

```
#include "Arduino.h"

unsigned int adcvalue, loval, hival;

void setup() {

  // Clear Bit 0 (PRADC) to turn on power for the ADC module
  PRR &= ~(1 << PRADC);

  //ADEN = 1, ADPS[2:0] = 111 (Prescale = 128), ADIE = 1
  ADCSRA |= ((1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0)
| (1 << ADIE));

  //REFS[1:0] = 01 (AVcc as reference), MUX[2:0] = 000 (Channel 0)
  ADMUX |= ((1 << REFS0));

  // Set PortB Pin 5 as output
  DDRB |= (1 << DDB5);

  sei();

  // ADSC = 1 (Start Conversion)
  ADCSRA |= (1 << ADSC);
}

void loop() {

  ledToggle();
  _delay_loop_2(adcvalue);
}

void ledToggle()
{
  PORTB ^= (1 << PORTB5);
}

ISR(ADC_vect)
{

  // Provide your code for the ISR

}
```

APPENDIX D

```
#include "Arduino.h"
#include <avr/interrupt.h>

unsigned int adcvalue;
unsigned int remapped_adc;

void InitPWM()
{
  TCNT0 = 0;
  OCR0A = 0;
  TCCR0A = 0b00000001;
  TIMSK0 |= 0b10;
}

void startPWM()
{
  TCCR0B = 0b00000100;
}

ISR(TIMER0_COMPA_vect)
{
      // Provide your code for the ISR
}

ISR(ADC_vect)
{
      // Provide your code for the ISR
}

void setup() {
  // put your setup code here, to run once:
  PRR &= 0b11111110;
  ADCSRA = 0b10001111;
  ADMUX = 0b01000000;
  DDRB |= 0b00100000;

  InitPWM();
  startPWM();
  sei();
  ADCSRA |= 0b01000000;

}

void loop() {}
```