Jonathan Hess
Section 3.2

Week 12 Lab Report: Free Lab 4

Lab Report Rubric

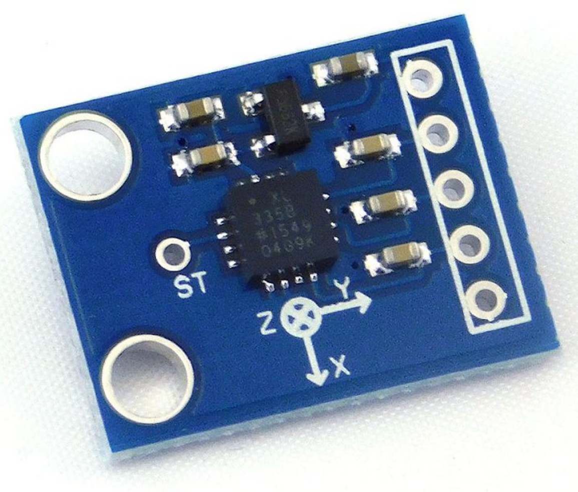
Category	Student Score	Grader Score
Organization		
Appropriate sections	1/1	/1
Appearance and formatting	2/2	/2
Spelling, grammar, sentence structure	1/1	/1
Work		
Experimental procedure	2/2	/2
Results (data, code, figure, graph, tables, etc.)	1.5/2	/2
Conclusion	2/2	/2
Total	**9.5/10**	**/10**

Introduction

Today I was going to finish up the circuit for the motion sensor and then finalize the attiny code to move the piston when moved. I started with a lot of research because there is less information out there then there is for Arduino.

Procedure

I got the Gy-61 (an accelerometer) for detecting increases in acceleration

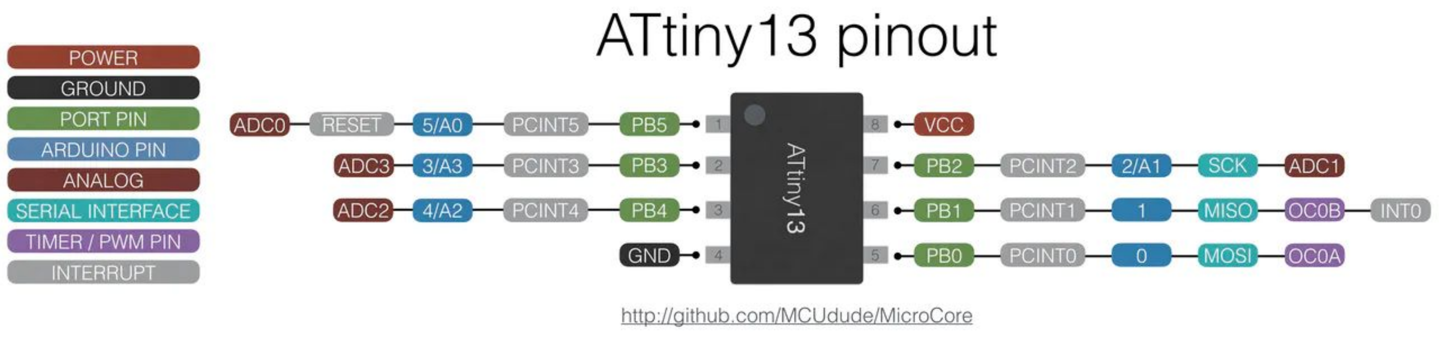


First part of the code was making sure that the accelerometer would work. I set up an arduino and wrote this basic code.

```
void setup() {
  // put your setup code here, to run once:
 Serial.begin(9600);
  pinMode(13, OUTPUT);

}

int X = A3;
int Y = A4;
int Z = A5;

int x =0;
int y = 0;
int z= 0;
bool buff = false;
void loop() {
```

```
          // put your main code here, to run repeatedly:
          x = analogRead(X);
          Serial.print("x = ");
          Serial.println(x);
          if(x> 325){
          digitalWrite(13, HIGH);

          }
        }
```

I thought I was going to have to use all 3 axises to detect change but decided on only using the x because it was sufficient. I also thought that it detected location at the time and didn't realize it was just an accelerometer. I wrote it iniacly to take a value and compare but it had a base value of 320 so I just added five (it might be smart to check 5 in both directions but it worked).



the attiny13 has 4 analog pins (more than enough) with a ADC with 10 bits
https://sites.google.com/site/qeewiki/books/avr-guide/analog-input - useful website

Next would be porting it to an ATtiny13 on atmel studio. Sadly the analog inputs are not as easy with Atmel, you need to work directly with the registers to activate the analog to digital converter (ADC)

THEORY OF OPERATION:

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
ADMUX	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0

ADC Multiplexer Selection Register

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVcc with external capacitor on AREF pin
1	0	Reserved
1	1	Internal 1.1V (ATmega168/328) or 2.56V on (ATmega8)

REF Bits

I am going to drop the last 2 bits because I don't need that much accuracy in this project.

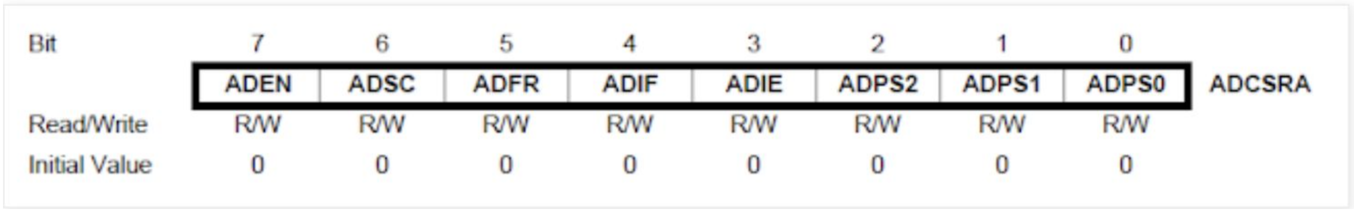
This really got me when I first started playing with micro controllers. Here is the deal. The ADC has 10bit of resolution. However, the AVR is an 8 bit micro controller, so we are always working with 8 bit registers. So in order to get 10bit data we have to split it into 2 registers ADCH and ADCL. We can use a 16 bit register or we could use an 8 bit register and dump the bottom 2 bits (which are the least significant bits anyways). If you choose to use the full 10 bits resolution you should leave ADLAR LOW(0) and make sure you read the ADCL register first because reading the ADCH causes to ADC to update. If you only want to use 8 bit resolution you will want to set the ADLAR to HIGH(1), this way you only have to read ADCH. I guess I could have said, the ADLAR bit lets you control how the AVR is going to output the data. Set ADLAR HIGH(1) if you only want to 8 bit of resolution.

```
void adc_setup (void)
{
  // Set the ADC input to PB2/ADC1
  ADMUX |= (1 << MUX0);
  ADMUX |= (1 << ADLAR);
  ADCSRA |= (1 << ADPS1) | (1 << ADPS0) | (1 << ADEN);
}
```

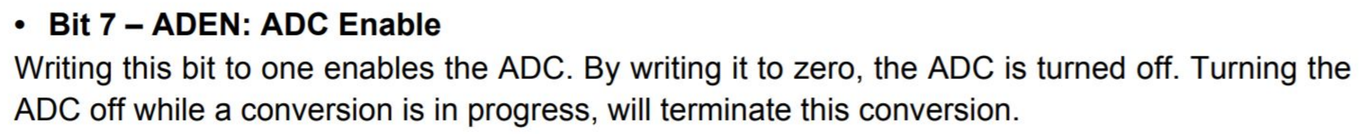
The mux settings http://ww1.microchip.com/downloads/en/DeviceDoc/doc2535.pdf#page=92

The Mux0 creates the 01 at the end (being the ADC1 (pin 7))

The ADLAR sets the shift to drop the bits



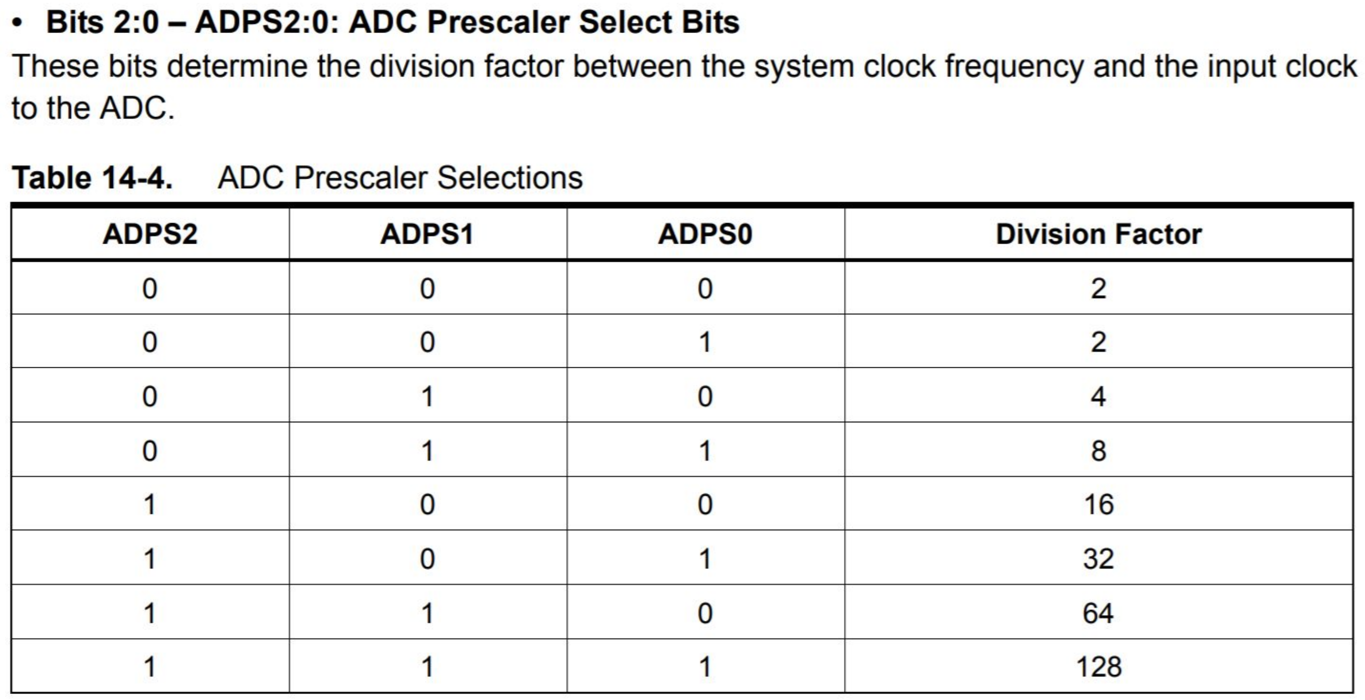
Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



- **Bit 7 – ADEN: ADC Enable**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

enable the ADC



- **Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits**

These bits determine the division factor between the system clock frequency and the input clock to the ADC.

Table 14-4. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

The last three bits set the speed (by dividing the clock signal)

```
int adc_read (void){
   // Start conversion
   ADCSRA  |= (1 << ADSC);

   // Wait to finish
   while (ADCSRA & (1 << ADSC));
    return ADCH;
}
```

In this code it gets the analog input

https://www.youtube.com/watch?v=4mPRZEdmWBc

Results

https://www.youtube.com/watch?v=2zz9vpvJkSQ

I got the circuit working where a movement for the accelerometer would turn the relays on for a few seconds, it actually should just swap the on/off state like the previous code so I fixed that. Overall I got the circuit working with an Attiny rather than an arduino which would be around 75 cents rather than 5 dollars and save a lot of space in the chest. I got to learn about registers for the ADC and more about MCU logic.

Conclusions and Reflection

Going through this project I realized how little I knew about all the registers and abilities a simple microcontroller has. I thought that it might be smart to learn more about this. I went to the library and checked out “Embedded C Programming & the Atmel AVR” and I plan on reading it over the break. I also bought some ADCs, comparators, and Op amps to mess around with over the break and see if I can make this circuit use even less complex parts. I really liked the free labs because of how much freedom they provided. I just wish that they could have been longer (which is why I am taking 186 next semester).