Jonathan Hess
Section 3.2

Week 10 Lab Report: Free Lab 2

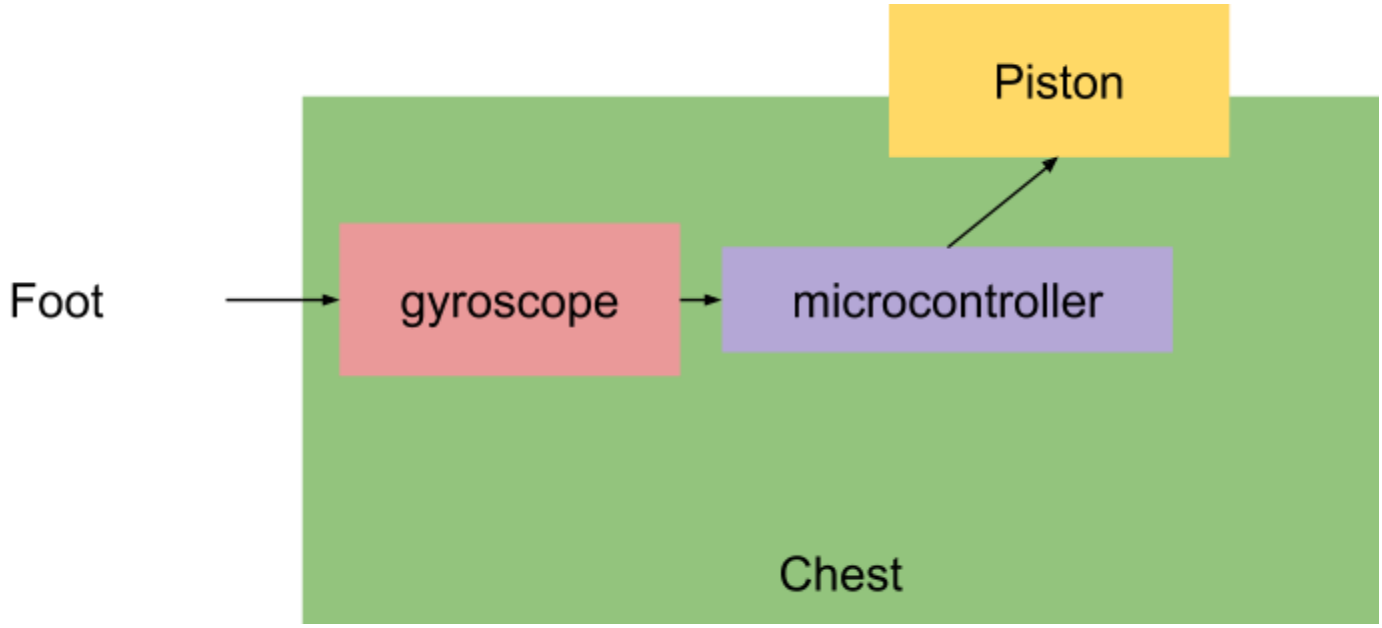
Lab Report Rubric

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	1.5/2	/2
Total	**9/10**	**/10**

Introduction

We got to have another free week so I decided to work on my personal project. I am working on a chest that opens when you kick it as well as learn more about microcontrollers and their programming. I started with the relays and piston then tested and moved to the programming of the microcontroller.

Systems Level Perspective



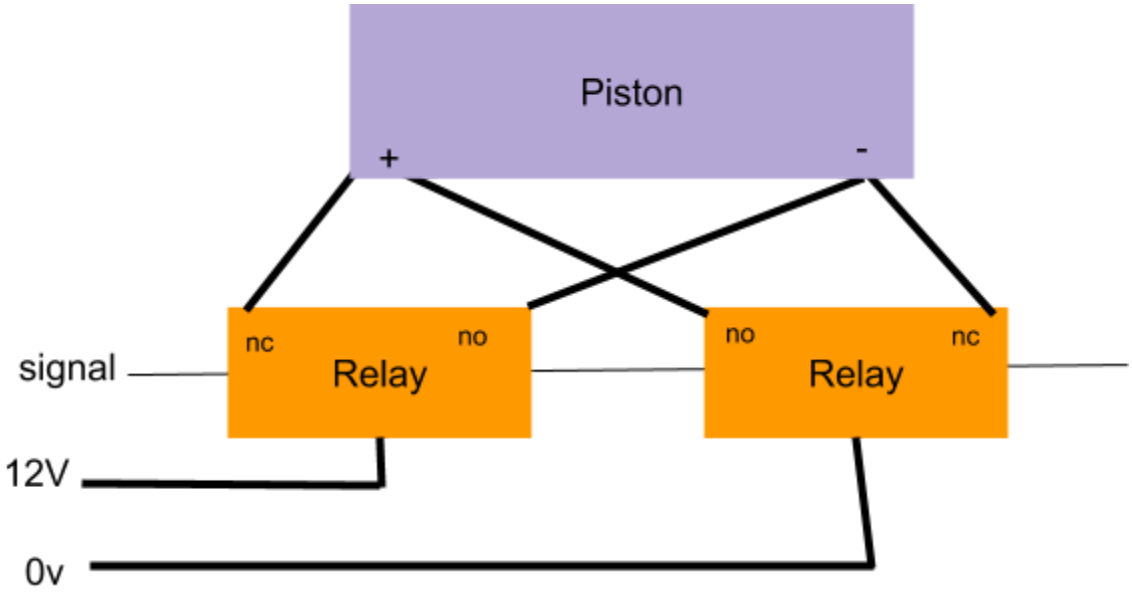
Procedure

State the Task/Problem/Question Attempted

First the goal was to get the piston to work in either direction using relays, then controlled with the microcontroller.

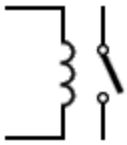
Procedure

RELAYS AND PISTONS



First step was getting the relay set up for both forward and backward. For fine control you could use Mosfets but I decided that it would be simpler to use relays that way there would be isolation from the two circuits. (also makes grounding easier between 5 and 12 volts)

The reason there is isolation is because a relay is actually composed of an electromagnet with metal lead that connects when the electromagnet is on. This means the other circuit is isolated from the signal circuit. (No fried arduinos or other MCUs)



A relay has 3 pins on the switch side: Normally Closed, Normally Open, and Common Ground

The common ground is what is connected to the +12 and ground because we want to control what is connected to those two.

The Normally closed is normally connected so we want to make sure that it makes a circuit with the piston when both relays are closed (else it would short to ground)

The other pins on the relay are for controlling the electromagnet part of the relay. Mine has 3 pins: Vcc, Gnd, and In

Vcc - the 5 volts on arduino

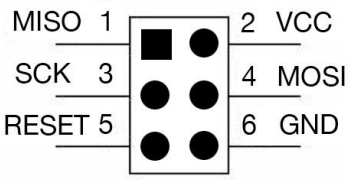
Gnd- the gnd on arduino

In- (normally held high) connects to gnd to close the relay switch

MICROCONTROLLER PROGRAMMING

Next after I made sure that the piston worked, I started working on programming for the Attiny13A. I have an AVR MKII for programming atmel microcontrollers (and other things such as memory).

The MKII has six pins:



VCC- a 5 volt line

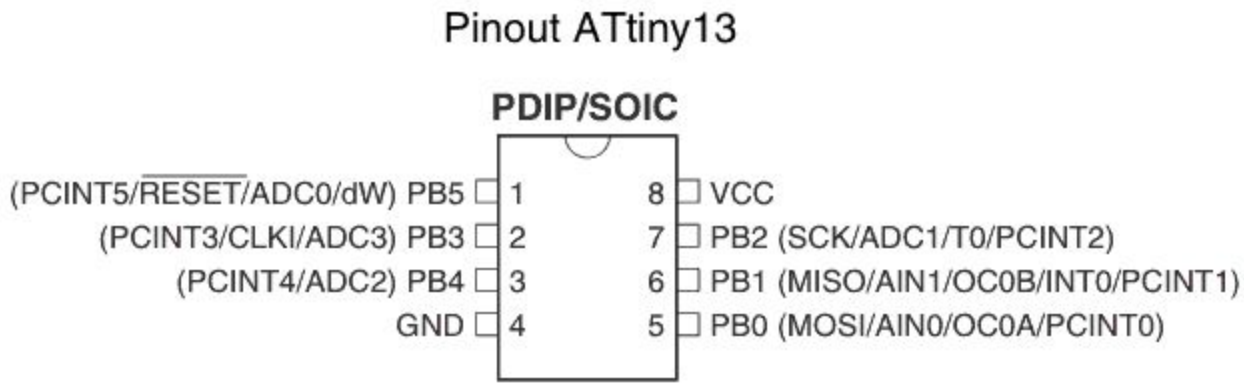
Gnd- Gnd Line

SCK - clock

Miso/Mosi - signal lines

reset- reset probably (should be connected to ground via button, I didn't do that)

These connect to the Attiny13A on the same pins (except Miso/Mosi those are inverted)



Using this I made a breadboard for programming (so I wouldn't have to plug and unplug everything with every update)

MICROCONTROLLER CODE

I found this code on the web and broke it down to understand what it does and how.

```
#include <avr/io.h>
#include <util/delay.h>

#define LED_PIN PB0                // PB0 as a LED pin

int main(void)
{

        /* setup */
        DDRB = 0b00000001;          // set LED pin as OUTPUT
        PORTB = 0b00000000;         // set all pins to LOW

        /* loop */
        while (1) {

                PORTB ^= _BV(LED_PIN);  // toggle LED pin
                _delay_ms(5000);
        }
}
```

SOURCE: https://blog.podkalicki.com/attiny13-blinky-with-delay-function/

Half of the problem is solved because of the comments.

In the setup it has the line DDR(port letter) which stands for Data Direction Register (meaning input and output pins)

This is used to define input and output pins it sets the last bit (B0) to 1 setting it as an output (all the other are input)

Then PORTB sets the high or low states of each pin via bit

The last fun bit of code is PORTB ^= _BV(LED_PIN);

The Code _BV(LED_PIN) is actually the equivalent to (1 << LED_PIN) this means it is shifting the 1 (in bits = 0001) to the left the number of times of LED_PIN (so if LED_Pin was 2 then it would result in 0100)

While researching this I learned that this is not good to use because it causes problems porting and confusion because it is a AVR library define

This used the ^= operator which takes the “exclusive or” of PORTB and _BV(LED_PIN);

This means that if PORTB was 00001 and because LED_PIN is 0 it would be doing this comparison

00001^00001 which returns 00000 - with the LED PIN LOW

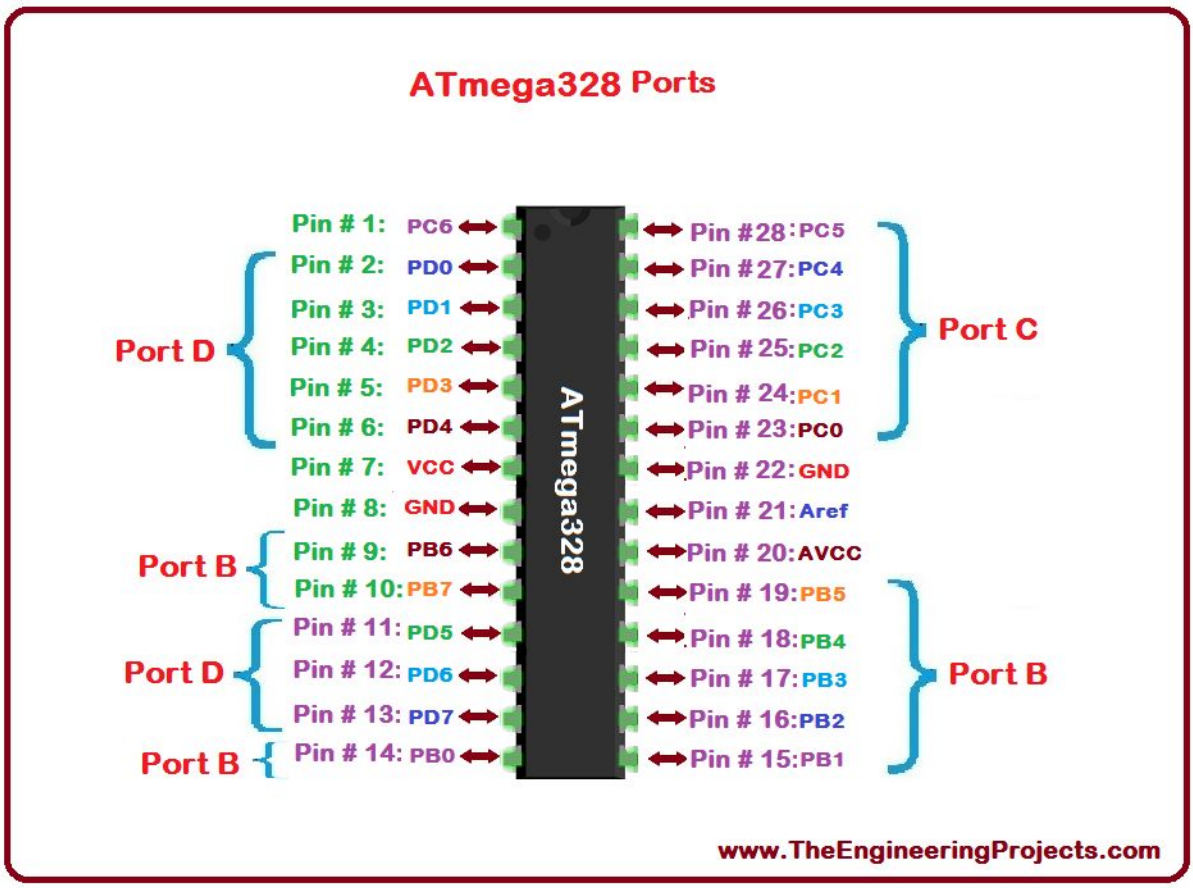
THEN once PORTB = 00000

00000^00001 which returns 00001 - with the LED PIN HIGH

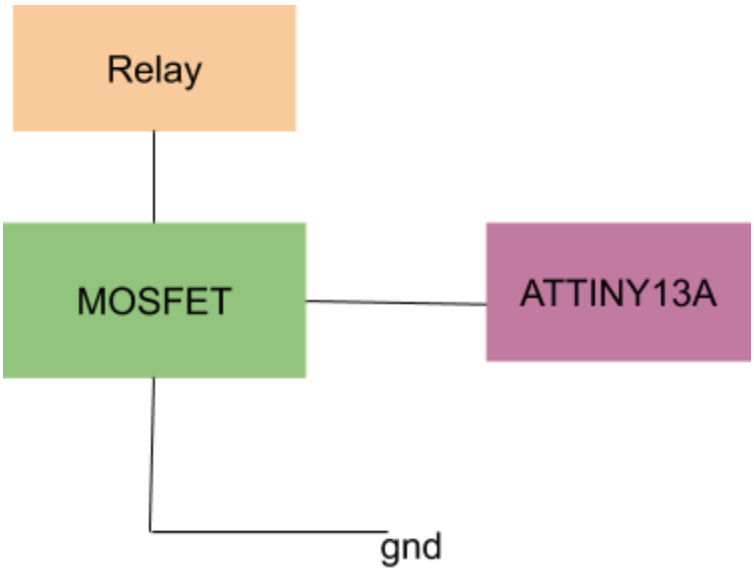
This effectively switches the PIN on and OFF every cycle

The last line delays for 5000 milliseconds

The attiny only has 1 port so it seems pointless to call it port B but if you were using the arduino MCU it is broken up like this



Now that we have a simple way of turning on and off pins we can use it for turning the relay on and off



Just add MOSFET to make it pull relay pin to ground (probably could replace with a normal transistor

Results

Results

I learned how the pins worked in the ATTINY13A and basics to flashing the memory onto the microcontroller. I also got to play with the piston and use my trusty bitwise operators again. Next week I hope to upload the code and have it actually moving the piston and then work on the motion detector.

Conclusions and Reflection

I learned a lot about how microcontrollers work and refreshed some electrical concepts. I probably could have done something more creative with the code after learning how it worked. I also should have looked at the documentation for what the code _BV as well as for what else can be done for programming. I hope next week to work on setting up the piston to the microcontroller and get started on the motion sensor. I think that knowing bitwise operators are one of the most important things for a electrical engineering programmer to know because they are useful for low level programming where the aim is efficiency (sacrificing a bit of readability though).