|  |
| --- |
| Group Echo |
| Arduino and Sonar Obstacle Detector |
|  |

|  |
| --- |
| Gina Hall, Jennifer Shelby, and Freddy Esteban  11/30/2015 |

# Table of Contents

[Table of Contents 1](#_Toc437808592)

[Table of Figures 2](#_Toc437808593)

[Introduction 3](#_Toc437808594)

[Arduino Uno with the Atmel ATmega328p 3](#_Toc437808595)

[History 3](#_Toc437808596)

[Processor Overview 3](#_Toc437808597)

[Arithmetic Logic Unit (ALU) 4](#_Toc437808598)

[Memory 4](#_Toc437808599)

[General Purpose Register File 5](#_Toc437808600)

[Other Basic Registers 6](#_Toc437808601)

[I/O Ports 7](#_Toc437808602)

[Stack 7](#_Toc437808603)

[AVR Instruction Set 8](#_Toc437808604)

[The HC-SR04 Ultrasonic Sensor 8](#_Toc437808605)

[Features 8](#_Toc437808606)

[Module Pin Assignments (Left to Right) 8](#_Toc437808607)

[Development Tools 9](#_Toc437808608)

[Arduino IDE 9](#_Toc437808609)

[Text Editor 9](#_Toc437808610)

[The Experiment 9](#_Toc437808611)

[Conclusion 10](#_Toc437808612)

[Appendix A – Team Contributions 11](#_Toc437808613)

[Freddy Esteban 11](#_Toc437808614)

[Gina Hall 11](#_Toc437808615)

[Jennifer Shelby 11](#_Toc437808616)

[Appendix B – Source Code 12](#_Toc437808617)

[High Level Language Code (C Language using the Arduino IDE): 12](#_Toc437808618)

[AVR Assembly Language: 13](#_Toc437808619)

[Appendix C - Works Cited 16](#_Toc437808620)

# Table of Figures

[Figure 1: Block Diagram of the AVR Architecture 4](#_Toc437808621)

[Figure 2: Program Memory Map 5](#_Toc437808622)

[Figure 3: Data Memory Map 5](#_Toc437808623)

[Figure 4: General Purpose Register File Organization 6](#_Toc437808624)

[Figure 5: Stack Pointer High (SPH) and Stack Pointer Low (SPL) 6](#_Toc437808625)

[Figure 6: AVR Status Register (SREG) 7](#_Toc437808626)

[Figure 7: HC-SR04 Back (top), HC-SR04 Front (bottom) 8](#_Toc437808627)

[Figure 8: Project Setup 9](#_Toc437808628)

# Introduction

Our group has decided to create device that will let us know when a bucket of chicken feed is low and needs to be filled. We will implement this with the Arduino Uno microcontroller, PING))) ultrasonic sensor, breadboard, a green LED, a red LED, wires, and resistors. After assembling and programming, the green LED will indicate that the chicken feed is less than 10 inches from the sensor; the red LED will indicate that the chicken feed is 10 inches or more from the sensor and then bucket needs to be filled.

# Arduino Uno with the Atmel ATmega328p

We are using the Arduino Uno microcontroller. These can be picked up at the Arduino website ([www.arduino.cc](http://www.arduino.cc)) or nearly any computer parts website. Arduino is an open-source computer hardware and software company, project, and user community. Arduino designs and makes microcontroller kits.

## History

Arduino was introduced in 2005 as a project for the students at the Interaction Design Institute Ivrea in Ivrea, Italy. Massimo Banzi, one of the founders, introduced the Arduino as an inexpensive solution to what students were using for projects. It is an open-source computer hardware and software company. This allows hobbyists and professionals create a variety of devices that can sense/interact with the world. The initial Arduino team was made up of Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis. They got the name from the bar they would meet at in Ivrea called Bar di Re Arduino.

Along with being a teacher at the Design Institute Ivrea and a co-founder of Arduino, Banzi has also worked as a consultant for Prada, Artemide, Persol, Whirlpool, V&A Museum, and Adidas. He is currently a teacher at SUPSI Lugano in Switzerland.

David Mellis is a co-founder of Arduino and is currently researching ways to engage new audiences in using electronics in creative and DIY practices. He has taught at the Copenhagen Institute of Interaction Design in Denmark. His job at Arduino advises on the development of the Arduino IDE and other Arduino software.

David Cuartielles is the head of the IOIO laboratory and an assistant professor at Malmo University. He is the head researcher for EU projects and designs new types of large scale educational experiences at Arduino.

Tom Igoe heads physical computing courses at ITP/NYU. He advises on issues of user experience in software and hardware, API, and documentation, as well as overall company strategy.

## Processor Overview

The Arduino uses the Atmel ATMEGA328P processor. The chip includes:

32 Kbytes of Program Memory

2.5 Kbytes of RAM

1 Kbytes of EEPROM Data

16 MHz max speed

26 Input/output pins

12 10-bit A/D Channels

6 PWM Channels

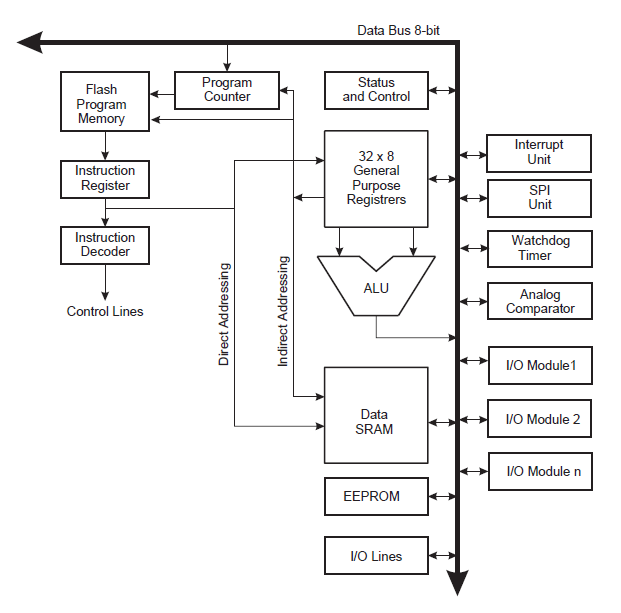


Figure : Block Diagram of the AVR Architecture

## Arithmetic Logic Unit (ALU)

The Atmel ATMEGA328P operates in direct connection with all 32 general purpose registers. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Operations work as register-to-register and register-to-immediate values. The ALU executes arithmetic operations within one clock cycle.

## Memory

The AVR uses Harvard Architecture. This means that the storage and signal pathways for instructions and data are separate. The Arduino takes advantage of Harvard Architecture for its processing speed by concurrent instruction and data access. Also, the program and data memories can be for different widths. Program memory is organized as 16k words (32K bytes) of flash memory and data memory is 2.5K bytes of static RAM (SRAM) and 1K bytes of EEPROM that can only be modified when you load a new program into the chip. All memory spaces are linear and regular.

### Flash Memory:

All AVR instructions are 16 or 32 bits wide, and for software security, the Flash Program memory space is divided into two sections: the Boot Loader Section and Application Program Section. The Boot Loader section is updated in a page by page fashion. Before programming a page with the data stored in the temporary page buffer, the page must be erased. It has at least 10,000 write/erase cycles.

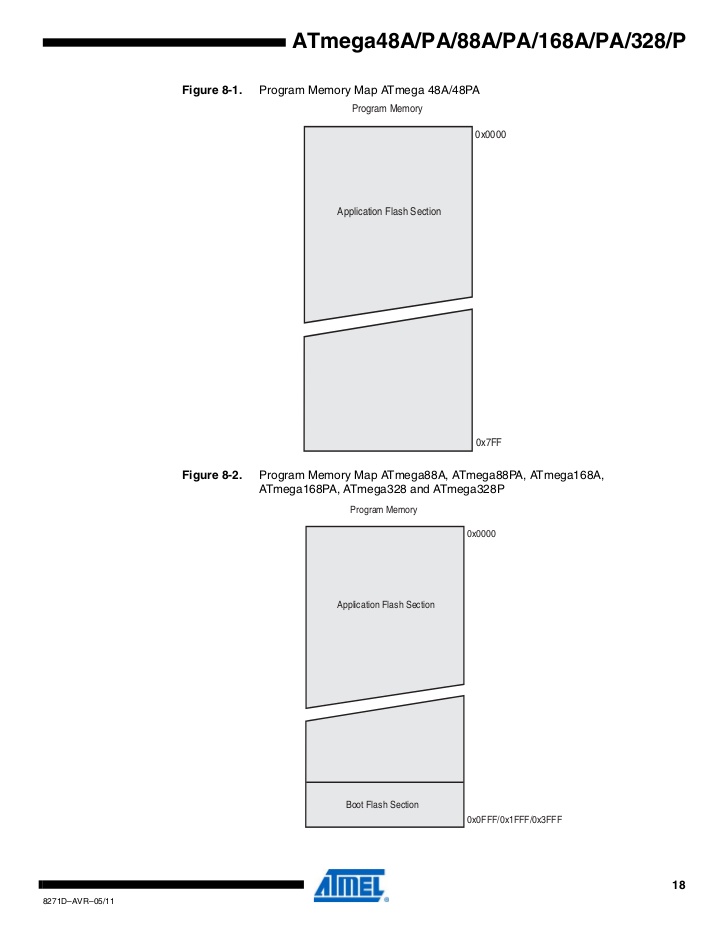


Figure : Program Memory Map

### SRAM Data Memory:

The lower data memory locations address the Register File, the I/O memory, Extended I/O memory and the internal data SRAM. The first 32 locations for the Register File, the next 64 for standard I/O memory, then 160 locations of Extended I/O memory. The next locations address the internal data SRAM. There are five different addressing modes for the data memory:

1. Direct (reaches entire data space)
2. Indirect with Displacement (reaches 63 address locations from the base address given)
3. Indirect (registers R26 to R31)
4. Indirect with Pre-decrement
5. Indirect with Post-increment

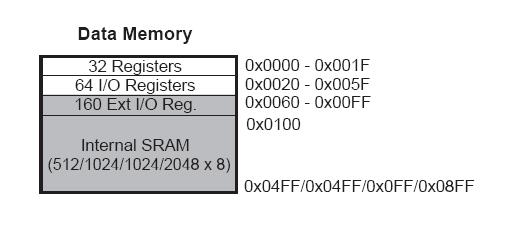


Figure : Data Memory Map

## General Purpose Register File

Most instructions that operate on the Register File have direct access to all registers, and most of them are single cycle instructions. Each register is assigned a data memory address that maps them directly into the first 32 locations of the user Data Space. The organization provides great flexibility in access of the registers. The following input/output schemes are supported:

* One 8-bit output operand and one 8-bit result input
* Two 8-bit output operands and one 8-bit result input
* Two 8-bit output operands and one 16-bit result input
* One 16-bit output operand and one 16-bit result input

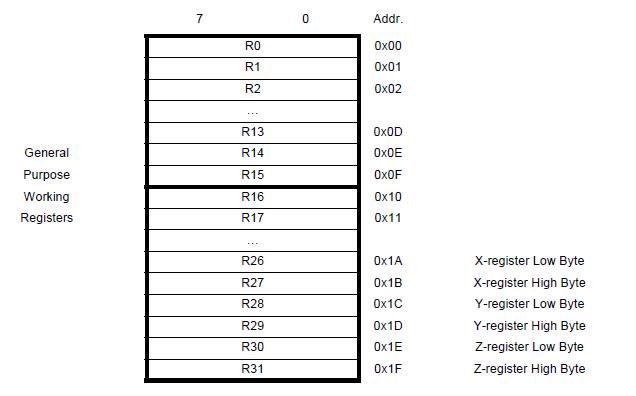


Figure : General Purpose Register File Organization

## Other Basic Registers

The microcontroller also has a few special registers. These include the Program Counter, Stack Pointer, and Status Register (flags).

### Program Counter (PC):

The Program Counter is located in the Flash Program memory. It is stored on the Stack during interrupts and subroutine calls. Because of the way the Flash is organized, the Program Counter can be treated as having two different sections. One consisting of the least significant bits that addresses the words within a page, while the most significant bits are addressing the pages.

### Stack Pointer (SP):

The Stack Pointer Register always points to the top of the Stack. All user programs must initialize the Stack Pointer (accessible in the I/O space) before subroutines or interrupts are executed. The Stack Pointer is implemented as two 8-bit registers in the I/O space, but the number of bits is dependent on the implementation.

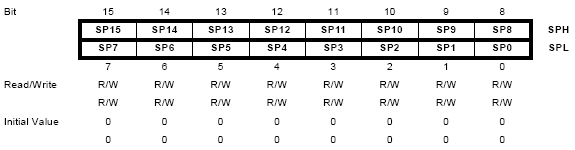


Figure : Stack Pointer High (SPH) and Stack Pointer Low (SPL)

### Status Register:

The Status Register contains information about the result of the most recently executed arithmetic instruction, which can be used for altering program flow in order to perform conditional operations. The status register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. The AVR status register is an 8-bit register. Each bit is initialized at 0, and all bits are read/write.

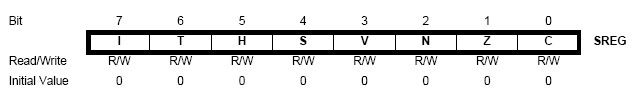


Figure : AVR Status Register (SREG)

#### Bit 7 (I) – Global Interrupt Enable

This bit must be set for interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. This bit is cleared by hardware after an interrupt has occurred, and can be set by various instructions.

#### Bit 6 (T) – Bit Copy Storage

The Bit Copy instruction BLD (Bit LoaD) and BST (Bit STore) use this bit as a source or destination for the operated bit.

#### Bit 5 (H) – Half Carry Flag

The Half Carry Flag, H, indicates a half carry in some arithmetic operation, useful in BCD arithmetic.

#### Bit 4 (S) – Sign Bit

The Sign bit is always an exclusive or between the Negative Flag (N) and the Two’s Complement Overflow Flag (V).

#### Bit 3 (V) – Two’s Complement Overflow Flag

This bit supports two’s complement arithmetic operations.

#### Bit 2 (N) – Negative Flag

The Negative Flag indicates a negative result in an arithmetic or logic operation.

#### Bit 1 (Z) – Zero Flag

The Zero Flag indicates a zero result in an arithmetic or logic operation

#### Bit 0 (C) – Carry Flag

The Carry Flag indicates a carry in an arithmetic or logic operation.

## I/O Ports

The I/O Space is part of the SRAM memory area. All ports have true Read-Modify-Write functionality when used as general digital I/O ports. The direction of one port pin can be changed without unintentionally changing the direction of any other pin. Many pins have multiple purposes that are set up as needed.

## Stack

The Stack is mainly used for storing temporary data, local variables, and return addresses after interrupts and subroutine calls. The Stack’s size is only limited by the total SRAM size and the usage of the SRAM, and it grows from higher to lower memory locations.

## AVR Instruction Set

Instructions fall into three basic categories:

Byte-oriented operations

Bit-oriented operations

Literal and control operations

The AVR only has 130 instructions and can be found on Atmel’s website (<http://www.atmel.com/images/atmel-0856-avr-instruction-set-manual.pdf>).

# The HC-SR04 Ultrasonic Sensor

The HC-SR04 ultrasonic sensor uses sonar to determine distance to an object similar to bats or dolphins. The device works in a range from 2 centimeters to 400 centimeters and is not affected by sunlight or black material like other rangefinders. The transmitter emits bursts of a directional 40 KHz ultrasonic wave when triggered and starts a timer. The pulses travel outwards until they encounter an object which causes the wave to be reflected back towards the unit. Once the ultrasonic receiver detects the reflected wave, the timer stops and distance to the object is calculated.

## Features

5V DC Power Supply

< 2 mA Quiescent Current

15 mA Working Current

< 15 degree Effectual Angle

2 cm – 400 cm Ranging Distance

## Module Pin Assignments (Left to Right)

Pin 1 VCC – 5V power supply

Pin 2 Trig – Trigger Input pin

Pin 3 Echo – Receiver Output pin

Pin 4 GND – Power ground

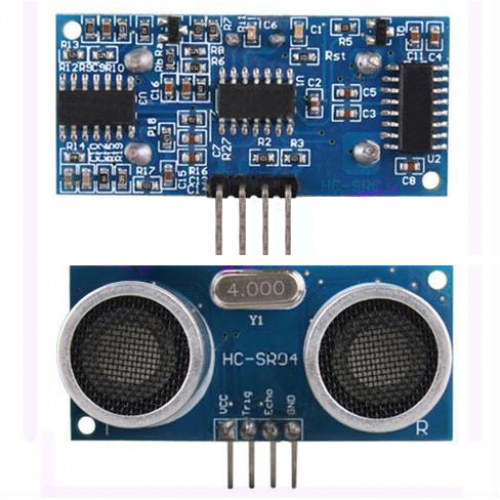


Figure : HC-SR04 Back (top), HC-SR04 Front (bottom)

# Development Tools

The development tools we used included:

Arduino IDE

Text Editor

## Arduino IDE

The Arduino IDE is open-source software for designing high level language programs to run with Arduino projects. The Arduino IDE is a cross-platform application written in Java. It is designed to introduce programming to those who are unfamiliar with software development. We used some tools within the Arduino IDE to compile our AVR assembly language code.

## Text Editor

We also used a text editor to write our .S files and Makefile for the project. I used Notepad++ which is open-source source software licensed by GNU General Public License. It supports tabbed editing allowing the user to work with multiple open files in a single window. It supports text encoding formats, auto-completion, syntax highlighting and syntax folding, and a variety of high level languages.

# The Experiment

One of our group members owns chickens. She wanted to come up with an easy way to tell if the bucket of chicken food was empty and needed to be refilled. She came up with the idea of setting up a proximity sensor in the bucket with some LED lights that will let her know when it is time to fill the bucket again. The bucket is 25 cm and the red LED will come on when the feed falls below 5 cm, otherwise, the green LED will be on.

We set this experiment up so that when the bucket is full the green LED is on. When the bucket needs to be refilled, the red LED will be on.

It was a fairly simple set up:

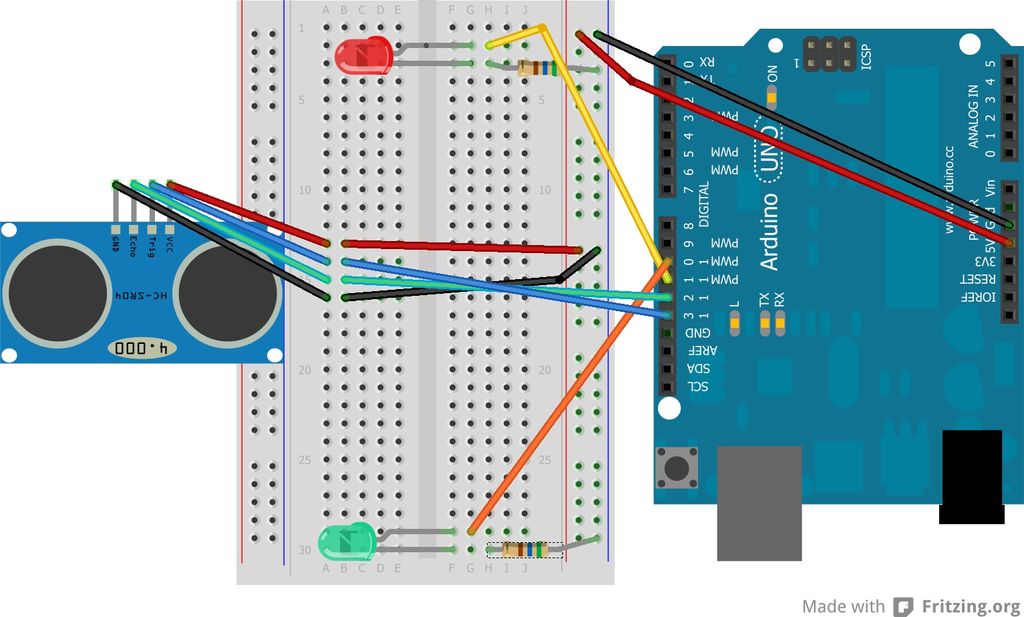


Figure : Project Setup

We placed the ultrasonic sensor on a breadboard. There were 4 wires connected from the sensor on the breadboard to the Arduino Uno. We connected the Trig pin of the sensor to pin 13 and the Echo pin of the sensor to pin 12. A green LED on one side of the breadboard and a red LED on the other each with a 560 Ohm resistor. The green LED was connected to pin 10 and the red LED was connected to pin 11.

# Conclusion

We could not get our AVR assembly code running, it was a little more complicated to implement that we originally thought. There was some difficulty getting the Echo pin to activate because there is a lot of timing that goes into this program. We were able to get the experiment to work using C within the Arduino IDE. The device works exactly as planned and our teammate is able to tell from a distance when it is time to fill the chicken feed bucket. The only problem is the chickens seem to like the red LED. When it is on, they peck at it and pull it off the breadboard. The teammate using the device is going to find a way to encase the project so that the chickens cannot pull out the LED.

The device has many uses and we definitely want to make other projects in the future, although we will probably code those in a high level language rather than AVR assembly language.

# Appendix A – Team Contributions

There were three members of our group, Freddy Esteban, Gina Hall, and Jennifer Shelby

Freddy Esteban

Freddy Esteban had no experience in programming before enrolling. He had always liked working with computers, but he only knew how to use them, but never to program them. Freddy started to learn programming on his own, but he felt that he needed more help, so he enrolled in ACC. He was introduced to web programming, which he found quite interesting. He is now continuing as a web developer, and found a job in this field. Freddy wrote code for the sensor.S file

## Gina Hall

Gina Hall had very little programming experience before enrolling as a college student. Before then, she dabbled in making a few web pages in HTML. While at UT Austin, she took Programming Fundamentals and Data Structures in Java. At ACC, while transitioning to Texas State, she took Data Structures in C++ and now Assembly. Her ultimate goal is to work with analyzing data, relational databases, and natural language processing. Gina helped write code for the high level language version as well as code for the sensor.S file

## Jennifer Shelby

Jennifer originally enrolled to ACC for transfer to a four-year university for Electrical Engineering, but she found she had a bigger passion for software rather than hardware and changed her major to Computer Science after one semester. She is transitioning to Texas State to earn her Bachelor of Science degree and is mostly interested in cybersecurity. Jennifer contributed to the group project by setting up the .S file. She set up the Stack and initialized the chip. She also wrote the group report.

# Appendix B – Source Code

## High Level Language Code (C Language using the Arduino IDE):

/\*

 \* October 2015

 \* Group Echo Project

 \* Freddy Esteban, Gina Hall, Jennifer Shelby

 \*

 \* Chicken Feeder Alert

 \* This device will be installed into the lid of a 25cm tall chicken feeder,

 \* pointing down towards the food. When the food level falls to below 5cm,

 \* the red alert light will turn on. Otherwise, while full, the green light

 \* will be on.

 \*

 \* Parts Used:

 \* Arduino Uno

 \* HC-SR04 Ping distance sensor (max distance 500cm)

\*/

#define trigPin 13      //Arduino pin for the sensor's trigger pin

#define echoPin 12      //Arduino pin for the sensor's echo pin

#define ledRed 11       //Arduino pin for the red led (food is empty)

#define ledGreen 10     //Arduino pin for the green led (food is full)

void setup() {

  Serial.begin (9600);        //open serial monitor at 9600 baud to see ping results

  pinMode(trigPin, OUTPUT);

  pinMode(echoPin, INPUT);

  pinMode(ledRed, OUTPUT);

  pinMode(ledGreen, OUTPUT);

}

void loop() {

  long duration;

  long cm;      //distance in centimeters

  digitalWrite(trigPin, LOW);   // low is off, high is on

  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH);

  cm = (duration/2) / 28;   //(round trip microseconds)/28 microseconds per cm

  Serial.print(cm);

  Serial.println(" distance");

  Serial.print(duration);

  Serial.println(" time");

  // This is where the LED On/Off happens

  //Food levels are considered "empty" when distance from lid > 20cm

  if (cm > 20) {

    digitalWrite(ledRed,HIGH);    //red is on

    digitalWrite(ledGreen,LOW);   //green is off

    Serial.println("EMPTY\n");

  } else {

    digitalWrite(ledRed,LOW);     //red is off

    digitalWrite(ledGreen,HIGH);  //green is on

    Serial.println("FULL\n");

  //error/out of range

  //16" = ~41cm : >41 \* 28 = ~1150

  } if (cm >= 41 || cm <= 0){

    Serial.println("Out of range\n");

  }

  delay(3000);     //in microseconds

}

## AVR Assembly Language:

### “sensor.S” File:

;Group Project 2

;Group Echo – Jennifer Shelby, Gina Hall, Feddy Esteban

;Date: December 1, 2015

;.equ PIND, 0x09 ;the input PIND register

.equ DELAY, 200 ;delay constant

#include “config.inc”

.section .text

.org 0x00

.global main

main:

;set up the stack

ldi r28, (RAMEND & 0x00ff)

ldi r29, (RAMEND >> 8)

out \_SPH, r29

out \_SPL, r28

;initialize the chip

call init

main\_loop:

call Trigger\_OFF ;send LOW to the TRIG\_PIN

call System\_Delay

call Trigger\_ON ;send HIGH to TRIG\_PIN

call System\_Delay

call Trigger\_OFF ;send LOW to TRIG\_PIN

call System\_Delay

;trig pin set to HIGH

call System\_Delay

rjmp main\_loop ;repeat the loop

ret

init:

eor r1, r1 ;zero out the r1 register

out \_SREG, r1 ;initialize the status register

;set up the LED port

sbi RED\_LED\_DIR, RED\_LED\_PIN ;set pin 11 on port b for output

cbi RED\_LED\_PORT, RED\_LED\_PIN ;start with the LED off

;set up the LED port

sbi GREEN\_LED\_DIR, GREEN\_LED\_PIN ;set pin 10 on port c for output

cbi GREEN\_LED\_PORT, GREEN\_LED\_PIN ;start with the LED off

;initialize trigger

sbi TRIG\_DIR, ECHO\_PIN ;set trigger pin 13 for output

cbi TRIG\_PORT, ECHO\_PIN ;start with the LED off

ret

;the delay of 20000 cycles on the 16mhz chip

System\_Delay:

ldi r17, DELAY ;outer loop

SD\_L1:

ldi r18, DELAY ;inner loop

SD\_L2:

dec r18

brne SD\_L2

dec r17

brne SD\_L1

ret

;trigger routines

Trigger\_ON:

sbi TRIG\_PORT, TRIG\_PIN ;set the trigger bit in PORTD high

ret

Trigger\_OFF:

cbi TRIG\_PORT, TRIG\_PIN ;set the trigger bit in PORTD low

ret

Toggle\_RED:

in r24, RED\_LED\_PORT ;get current port b values

ldi r25, (1 << RED\_LED\_PIN) ;LED pin number

eor r24, r25 ;toggle bit

out RED\_PORT, r24 ;write it back in place

ret

Toggle\_GREEN:

in r24, GREEN\_LED\_PORT ;get current port b values

ldi r25, (1 << GREEN\_LED\_PIN) ;LED pin number

eor r24, r25 ;toggle bit

out GREEN\_PORT, r24 ;write it back in place

ret

### Makefile:

#Makefile for Group Project 2

TARGET = Sensor

ASRCS = $(wildcard \*.S)

INTTBL = FALSE

#Board Definition

MCU = atmega328p

F\_CPU = 16000000

#Do not modify anything below this line except the PORT setting

Ifeq ($(OS), Windows\_NT)

include c:/usr/local/include/Makefile.win

PORT = COM3

else

UNAME\_S = $(shell uname –s)

ifeq ($(UNAME\_S), Linux)

include /usr/local/include/Makefile.unx

PORT = /dev/ttyACM0

endif

ifeq ($(UNAME\_S), Darwin)

include /usr/local/include/Makefile.mac

PORT = /dev/cu.usbmodem1411

endif

endif

# Appendix C - Works Cited

“About Us.” *Arduino.* Arduino, 2015. Web. 1 Dec. 2015.

"ATmega48A/PA/88A/PA/168A/PA/328/P." *Atmel*. Atmel Corporation, 2015. Web. 1 Dec. 2015. <http://www.atmel.com/images/atmel-8271-8-bit-avr-microcontroller-atmega48a-48pa-88a-88pa-168a-168pa-328-328p\_datasheet\_complete.pdf>.

"Atmel AVR 8-Bit Instruction Set." *Atmel Corporation*. 2014. Web. 10 Dec. 2015. <http://www.atmel.com/images/atmel-0856-avr-instruction-set-manual.pdf>.

Black, Roie. "Embedded Machines." *COSC2325-002 PC Architecture and Machine Language*. 2015. Web. 1 Dec. 2015. <http://www.co-pylit.org/courses/cosc2325/lectures/lecture14/index.html>.

"HC-SR04 User Guide." *Electronic Components and Accessories*. NiteOwl Computing, 2015. Web. 1 Dec. 2015. <https://www.mpja.com/download/hc-sr04\_ultrasonic\_module\_user\_guidejohn.pdf>.

"Arduino." *Wikipedia*. 9 Dec. 2015. Web. 10 Dec. 2015. <https://en.wikipedia.org/wiki/Arduino>.