**Target in Range**

An Arduino Sonar Obstacle Detector Project

Written with the use of C and the AVR Assembly Language

By The Delta Group

**Δ**

Nick Kasten -  Jesse Lew - Patrick Martinez

1. **Introduction**

When deciding on which project we were going to pursue, our group decided to combine all of our ideas. We decided to use a sonar obstacle detection module, a tri-colored RGB LED, and a button connected to a buzzer.

Our biggest challenge was merging all of these different parts into the same program. We were able to tie everything together in C, which was our prototype version. We decided to use this first so that we could have an idea of how the program will work and what the requirements of our sonar function would be before starting on that piece of more complicated code.

After we completed our C version of the code and confirmed its functionality, we began putting our efforts towards removing the sonar function from the main file and attaching it as an external call to an AVR function.

To accomplish this, we converted the sonar function into a .lst and use it to make a .S using Professor Black's Arduino makefile. However, we were told not to do it that way, so we went back and rewrote it by hand. It ended up being much cleaner and significantly shorter (from 500 lines of Assembly down to 50!). It was easier not to have to worry about including any of the Arduino IDE's files, and to simply assign ports and pins ourselves.

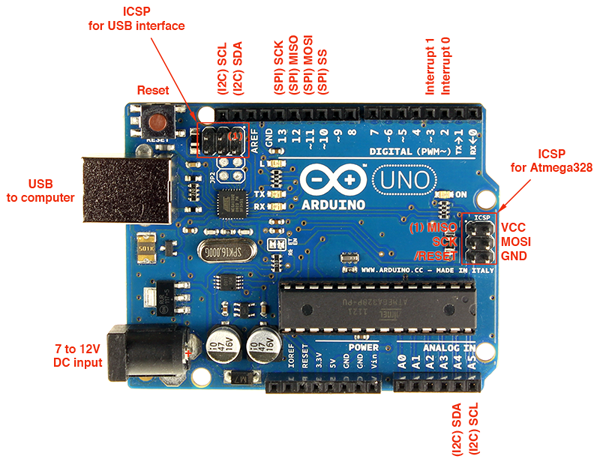
**2) The Microcontroller Platform**

The platform we used was an Arduino Uno equivalent device, the Elegoo Uno. The Elegoo Uno that we obtained is a microcomputer functionally equivalent to the Arduino Uno available at a much lower cost. These may be obtained through many online supermarkets such as [www.Amazon.com](http://www.amazon.com) or [www.ebay.com](http://www.ebay.com).





The Elegoo Uno features an ATmega328 Microcontroller with an 8-bit processor and 16MHz clock speed. It also features a modifiable memory bank to load/run programs. The board has multiple I/O ports to send and receive signals. A Power port so it can run without the USB, and a USB port capable of providing power and writing data.



**3.a) The Test Device**

The main test device in our project is an ultrasonic obstacle sensor, the HC-SR04.



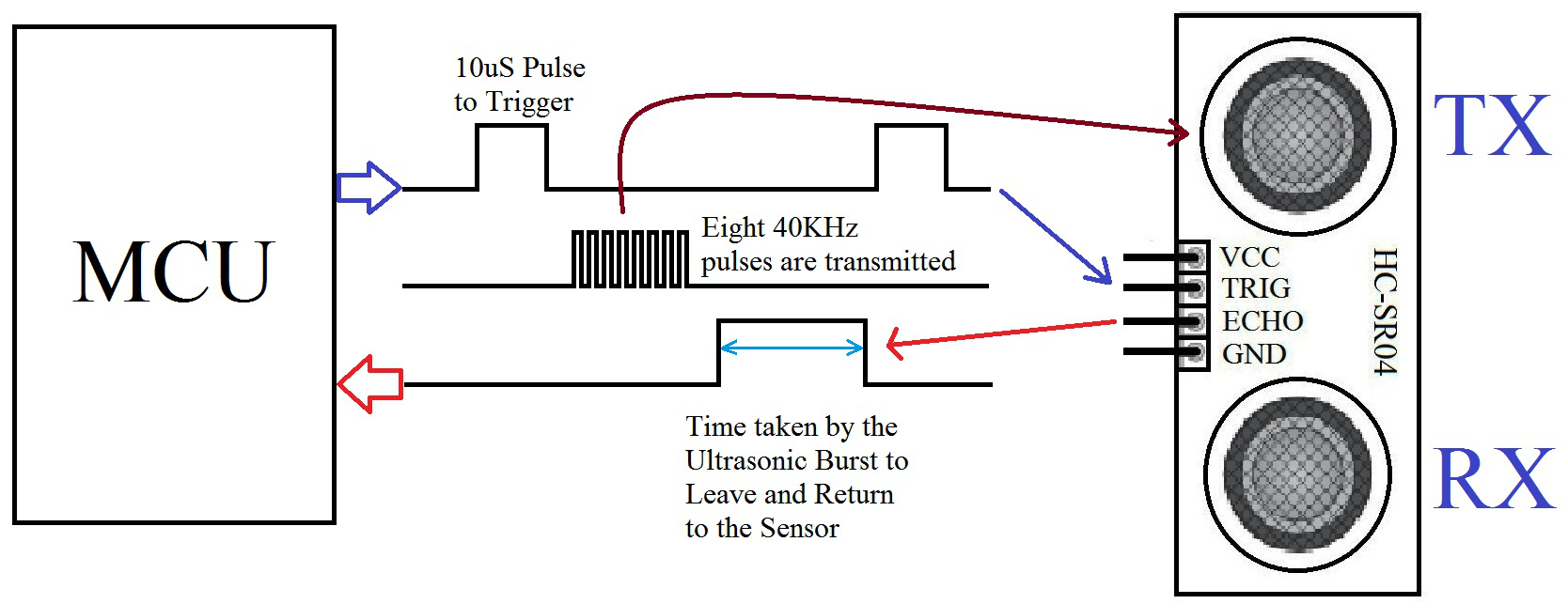


This device has 4 pins:

1. VCC – 5V, positive end of the power supply
2. TRIG – Trigger Pin
3. ECHO – Echo Pin
4. GND – 0V, ground, negative end of the power supply

TRIG and ECHO were used to interface this module with our microcontroller, the Arduino. These are TTL (0 – 5V) input output pins.

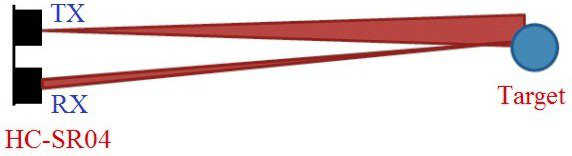
**3.b) How it works**





**We need to:**

1. Provide a TRIGGER signal, at least 10μS High Level (5V pulse)
2. The module transmits 8 40KHz ultrasonic bursts
3. If there is an obstacle in-front of the module, it will reflect the ultrasonic burst. If the signal comes back, ECHO output of the sensor will be in HIGH state (5V) for the duration of time taken for sending and receiving the ultrasonic bursts. Pulse width ranges from about 150μS to 25mS. If no obstacle is detected, the echo pulse width will be about 38ms.
4. We set a timer to measure that duration of time and can calculate the distance.





**3.c) Other Gizmos**

Our C program receives an integer value back from the Assembly Sonar function and we use that value to determine if a target is in range or not. We then display the proper LED light color.

Red = Out of range.

Yellow = Almost in range...

Green = Target in range!

Note: we used a RGB LED. So to create the different colors we wanted, we had to turn on different combinations. Specifically, to create Yellow, we had to turn on both Blue and Green at the same time.

When the target is finally in range, we enable a button to be pushed. The button is normally disabled and does nothing when a target isn't in range. The button sends a signal to a buzzer on the breadboard signifying that the target has been fired upon.

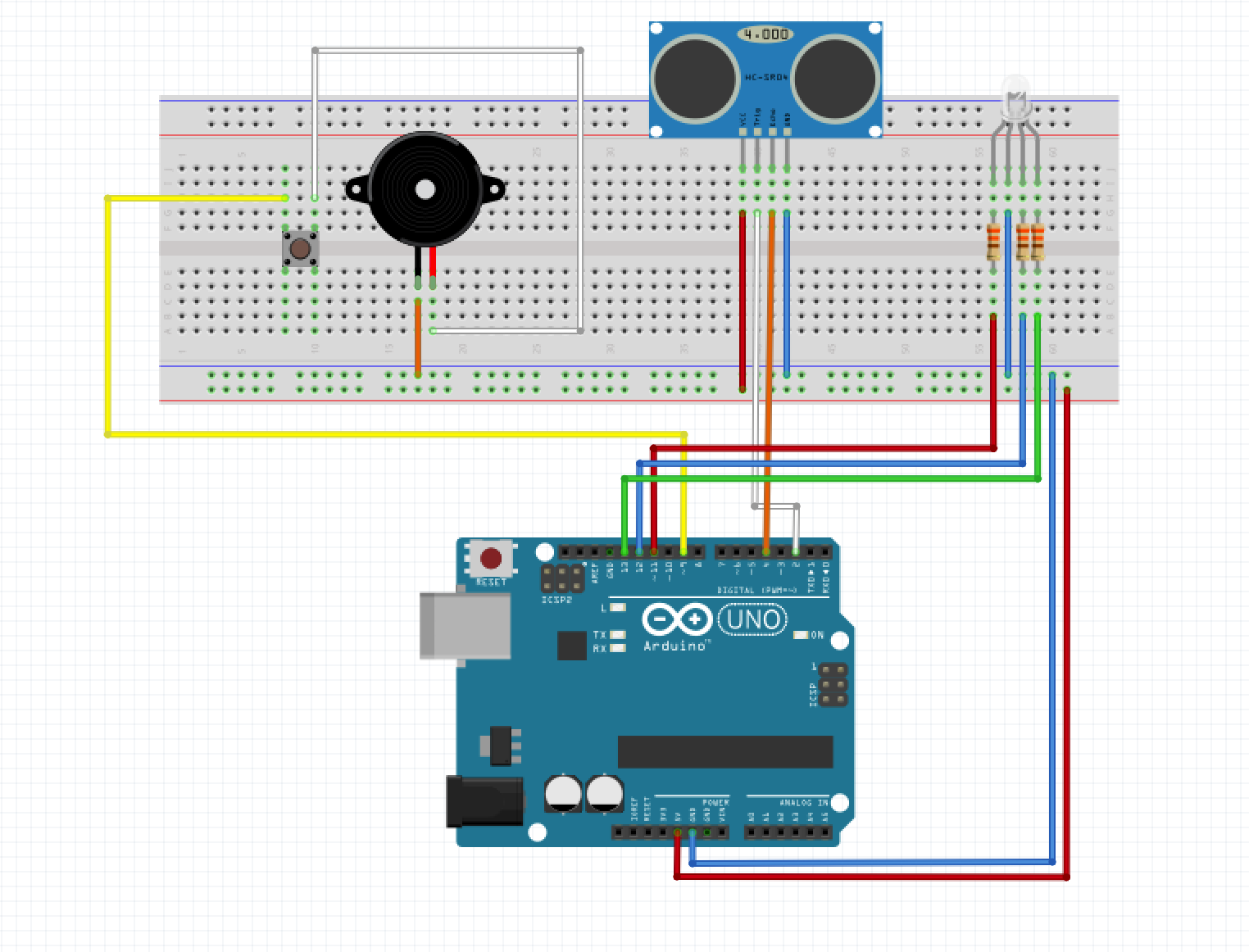
Target destroyed!

**Development Tools**

Generally, folks use the Arduino IDE to program their Arduino Device. We could not use the default as that would beat the point of this learning experiment. We used gVIM and sublime text as the main text editors and to examine our existing text. We extensively used gitHub to share and collaborate. We used Slack chat messaging app as a meeting place and to pass around ideas. And some of us used VMWare player for easier compilation and better integration when working as a team. We also used Fritzing to create a wiring diagram for our report.

**Our Experiment**

We decided pretty early in our project that we wanted to use the Sonar Obstacle Detector. We were planning to create our program in C and port out a small part of the program into assembly, like the LED light, but we were informed that this would be to simple and we decided to take on the challenge of programming the sonar device in C. Our project is sort of like a “rangefinder” missile test. The idea is that our system will detect if an object is at an optimum distance, then only allow it to FIRE if in range.





The preceding diagram shows how our devices were connected to the microcontroller. It seems like a lot of wires, but it wasn’t too difficult to manage once we got a sense of where each one went. As mentioned before, we started by programming the entire system in C, then ported out the sonar as we wanted to set down our framework for the .S portion.

**Conclusions**

We were successful in our goal of creating a working Arduino sonar project that made use of both C code and Assembly. The device works as intended. It’s great to have a complete AVR sonar file that we can use in future Arduino projects. Group Delta Master Race!

**Contributions**

* **Nick Kasten** - Contributed to the planning stage, the hardware code, the entire software code and testing and troubleshooting efforts
* **Jesse Lew** - Contributed to the planning stage, extensive assembly research, this report and troubleshooting.
* **Patrick Martinez**  - Contributed to the planning stage, C and Assembly linking research, this report and troubleshooting.

**Project Code**

**main.c**

|  |  |
| --- | --- |
|  | **#include <avr/io.h>** |
|  | **#include <util/delay.h>** |
|  |  |
|  | **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/** |
|  | **/\* Defining Constants \*/** |
|  | **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/** |
|  |  |
|  | **//--- PIN Assignments ----------------------------------------------------** |
|  | **#define RED\_PIN 3** |
|  | **#define GREEN\_PIN 4** |
|  | **#define BLUE\_PIN 5** |
|  | **#define BUZZ\_PIN 1** |
|  |  |
|  | **//--- PIN Controls -------------------------------------------------------** |
|  | **#define RED\_ON (PORTB |= (1<<RED\_PIN))** |
|  | **#define RED\_OFF (PORTB &= ~(1<<RED\_PIN))** |
|  | **#define GREEN\_ON (PORTB |= (1<<GREEN\_PIN))** |
|  | **#define GREEN\_OFF (PORTB &= ~(1<<GREEN\_PIN))** |
|  | **#define YELLOW\_ON (PORTB |= (1<<RED\_PIN) | (1<<GREEN\_PIN))** |
|  | **#define YELLOW\_OFF (PORTB &= (0x01))** |
|  |  |
|  | **#define BUZZ\_ON (PORTB |= (1<<BUZZ\_PIN))** |
|  | **#define BUZZ\_OFF (PORTB &= ~(1<<BUZZ\_PIN))** |
|  |  |
|  | **//--- Device Setup -------------------------------------------------------** |
|  | **#define OUTPUT\_CONFIG (DDRB |= (1<<RED\_PIN) | (1<<GREEN\_PIN) | (1<<BUZZ\_PIN))** |
|  | **#define CPU\_PRESCALE(n) (CLKPR = 0x80, CLKPR = (n))** |
|  | **int buzzerState = 1;** |
|  | **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/** |
|  | **/\* Device function definitions \*/** |
|  | **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/** |
|  |  |
|  | **//--- Buzzer -------------------------------------------------------------** |
|  | **void toggleBuzz(void) {** |
|  | **if (buzzerState == 1) {** |
|  | **BUZZ\_OFF;** |
|  | **buzzerState = 0;** |
|  | **} else {** |
|  | **BUZZ\_ON;** |
|  | **buzzerState = 1;** |
|  | **}** |
|  | **}** |
|  | **void buzzEnable(void) {** |
|  | **char ticks = 0;** |
|  | **int blastCount = 0;** |
|  | **while(blastCount <= 100) {** |
|  | **while((TIFR0 & 0x01) == 0) {}** |
|  | **TIFR0 = 1;** |
|  | **ticks++;** |
|  | **if (ticks == 20) {** |
|  | **ticks = 0;** |
|  | **toggleBuzz();** |
|  | **}** |
|  | **blastCount++;** |
|  | **}** |
|  | **}** |
|  |  |
|  | **//--- LED ----------------------------------------------------------------** |
|  | **void LEDColor(int distance) {** |
|  | **YELLOW\_OFF;** |
|  |  |
|  | **if (distance >= 4) {** |
|  | **RED\_ON;** |
|  | **}** |
|  | **else if (distance >= 2) {** |
|  | **YELLOW\_ON;** |
|  | **}** |
|  | **else {** |
|  | **GREEN\_ON;** |
|  | **buzzEnable();** |
|  | **}** |
|  | **}** |
|  |  |
|  | **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/** |
|  | **/\* MAIN function \*/** |
|  | **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/** |
|  |  |
|  | **int main(void) {** |
|  | **//--- Setup ----------------------------------------------------------** |
|  | **int distance;** |
|  | **OUTPUT\_CONFIG;** |
|  |  |
|  | **// Set Timer/Counter to proper speed for buzzer** |
|  | **TIMSK0 = 0;** |
|  | **TCCR0B = 2;** |
|  |  |
|  | **//--- Main Loop ------------------------------------------------------** |
|  | **while(1){** |
|  | **distance = sonar();** |
|  | **LEDColor(distance);** |
|  | **}** |
|  | **return 0;** |
|  | **}** |

**Sonar.S**

;-----------------------------------------------;   
; ASM code for the HC-SRO4 Sonar device ;  
; Created by: The Delta Group ;   
;-----------------------------------------------;  
  
#include "config.inc"  
  
 .section .text  
 .global sonar  
  
;-----------------------------------------------;  
; SONAR FUNCTION ;  
;-----------------------------------------------;  
sonar:  
   
 sbi \_DDRD, TRIG\_PIN ; sets pin 2's direction to output => Trig Pin  
 cbi \_DDRD, ECHO\_PIN ; sets pin 4's direction to input <= Echo Pin  
 cbi \_PORTD, TRIG\_PIN ; sets the output to LOW  
send:   
 sbi \_PORTD, TRIG\_PIN ; set Trig Pin to HIGH to send a burst of 8 pings  
 call tenMicroDelay ; HIGH signal must continue for 10 microseconds  
 cbi \_PORTD, TRIG\_PIN ; set the output to LOW after pings are sent  
receive:  
 clr r21 ; clear counter value  
waiting:  
 call hundredCMDelay ; wait long enough for the sound to travel 100 cm (each way)  
 inc r21 ; increment counter  
 sbic \_PIND, ECHO\_PIN ; break out of loop if response received  
 rjmp waiting ; otherwise keep waiting for the response  
 clr r25 ; zero out high bits of return register  
 mov r24, r21 ; set lower bits to the value of counter  
 call tenMicroDelay  
 ret ; return to calling function  
  
;-----------------------------------------------;  
; DELAY FUNCTIONS ;  
;-----------------------------------------------;  
tenMicroDelay:  
 ldi r18, 2 ; set outer loop to 2  
1:  
 ldi r19, 80 ; set inner loop to 80 (160 cycles total)  
2:  
 dec r19 ; decrement inner loop  
 brne 2b ; keep looping until (r19 == 0)  
 dec r18 ; decrement outer loop  
 brne 1b ; keep looping until (r18 == 0)  
 ret ; return after 10 microseconds  
hundredCMDelay:  
 ldi r18, 100 ; set outer loop to 100 (centimeters)  
5:  
 ldi r19, 58 ; set middle loop to 58 (microseconds per centimeter)  
6:   
 ldi r20, 16 ; set inner loop to 16 (cycles per microsecond)  
7:  
 dec r20 ; decrement inner loop  
 brne 7b ; keep looping until r20 == 0  
 dec r19 ; decrement middle loop  
 brne 6b ; keep looping until r19 == 0  
 dec r18 ; decrement outer loop  
 brne 5b ; keep looping until r18 == 0  
 ret ; return after a delay equal to 100 centimeters