WCFCountry
West Central Florida Section

THE WCF EXPERIMENTER

EDITOR: GEOFF HAINES, N1GY ***N1GY@ARRL.NET***

Published Quarterly ***Winter Edition***

FROM THE EDITOR:

As usual, I will again put out a call for articles. This is not a publication about what has N1GY built this season. I need material about what you have constructed. It might be an antenna, a go-kit, a way of powering your station, whatever. Please, write it up, take some photos, send it to me. I guarantee I can fix any spelling or syntax errors and turn what you wrote into a fine article that you will be proud of. That is what editors do. I know I am not the only ham in the WCF that has done a little DIY radio. Show me what you have done so others may be inspired to join the Ham Radio DIY movement. This quarter we actually have an article by Stan, AJ4SN. Hooray!!!

Making Smoke: JABOB

By

Stanley O. Vittetoe, AJ4SN

I recently read a construction article that used the term “JABOB” which stands for “Just A Bunch Of Boxes.” It describes a large construction project in which each subsection is constructed as a box or module. The final project connects them all together and the result is “JABOB.”

I have built several larger projects using the JABOB technique even though I wasn’t aware of the acronym at the time. In fact, I’m in the middle of my latest receiver project, and I am up to two boxes at present.

Let me back up just a little. I am currently using a homebrew receiver and transmitter as my main station, but in the spirit of never being satisfied with anything, I have embarked on a new and improved homebrew receiver project. The new receiver will incorporate a better IF amplifier (the so-called “hycas” IF designed by W7ZOI), a noise blanker (courtesy of Chris Trask, N7ZWY), a CW filter, and an improved audio stage (from W1FB).

If a receiver sounds like a pretty big project, remember that it is nothing more than a series of smaller projects or boxes. Each of the boxes can be built and tested separately and then connected together to form the final product.

At this point in the receiver project, I have the beat frequency oscillator (BFO) and the product detector built. Both circuits were taken from Experiments in Radio Frequency Design, the source for most of my current homebrew projects (schematics are shown below). I am not an engineer, in fact, I work in a community college in a very non-technical job. But I have been a ham for many years, and I have a real passion for building stuff.

I decided to build the BFO and product detector in Bud aluminum enclosures. I chose the heavy enclosure with a lid so that I could completely seal off the BFO energy from the rest of the receiver. The BFO output will be on the order of 1-2 volts. Signals arriving at the receiver will be on the order of 0.1 microvolts. The BFO signal is almost a billion times greater than the typical received signal. I need to keep the strong BFO energy out of the other sensitive stages in the receiver.

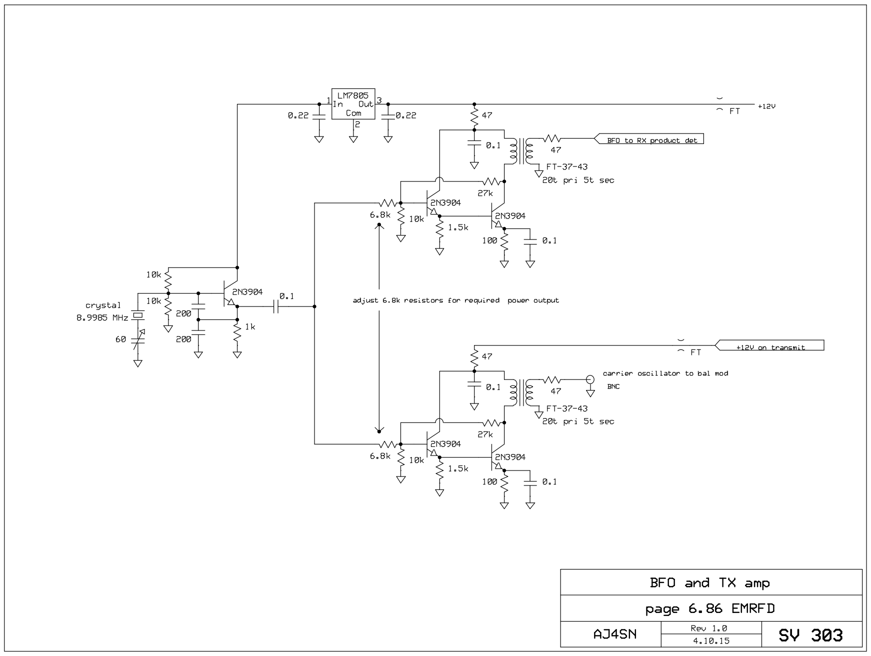
I tapped the enclosures for 10-32 screws to hold the circuit board down. I can also use the screws when I mount the box in a larger case and assemble all the boxes into a complete radio (see photo below). I used small squares of pc board glued to the main pc board for connection points. You can cut these from pc scraps, but I buy mine from QRPme (300 for $10). I also used a feed through capacitor mounted in the enclosure to supply DC power to the stage. This helps keep RF energy from sneaking out of the enclosure on the DC power lead.

Testing of the BFO is a simple process. Before I apply power, I like to review the connections, checking resistances and making sure that I have wired it correctly. I then add a 50 ohm termination, apply power, and look at the output waveform on the scope. In this case it worked first time! The output was 2.8 V p-p which equals 12.9 dBm. I'm using a mini circuits TAK-3H mixer in the product detector which will handle 17 dBm, so this is an acceptable value.

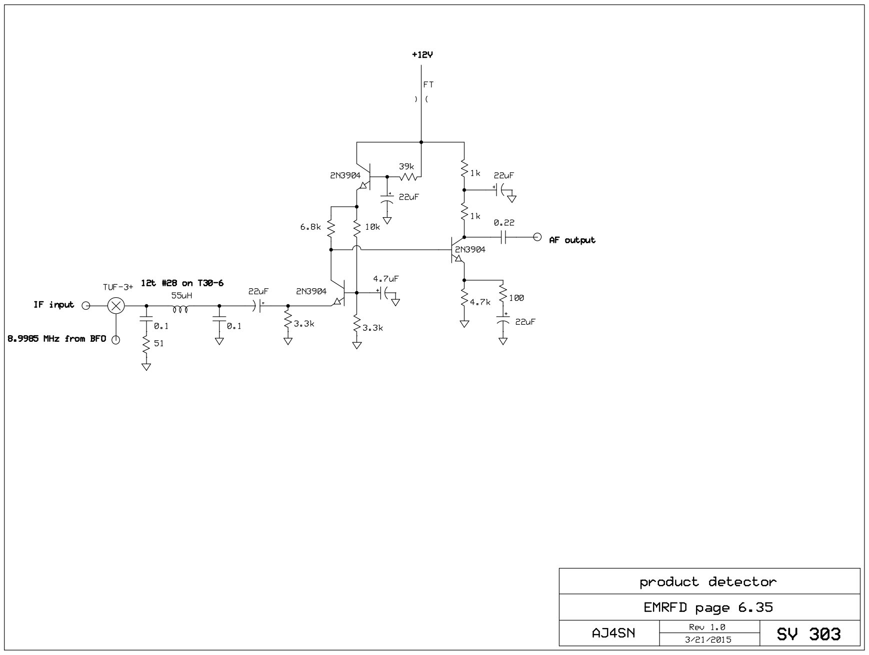
The product detector was also built in a Bud enclosure. The IF for this receiver is 9 MHz. I chose this frequency because I already had some 9 MHz filters and I have a pretty workable VFO that tunes 5.0 to 5.5 MHz. To test the product detector, I injected the 8.9985 MHz BFO signal and a 9 MHz signal from my homebrew signal generator into the product detector. As I hoped, the output was the difference between the two frequencies, 1500 Hz. Of course, if I was listening to a SSB signal, the signal frequency would vary around the 9 MHz center frequency, and the product detector would output the difference between that signal and the BFO, reproducing the transmitted audio.

The next box in this project will likely be the audio output amplifier followed by the RF input filters for 20 and 40 meters. Like they say about eating an elephant, you do it one bite at a time!

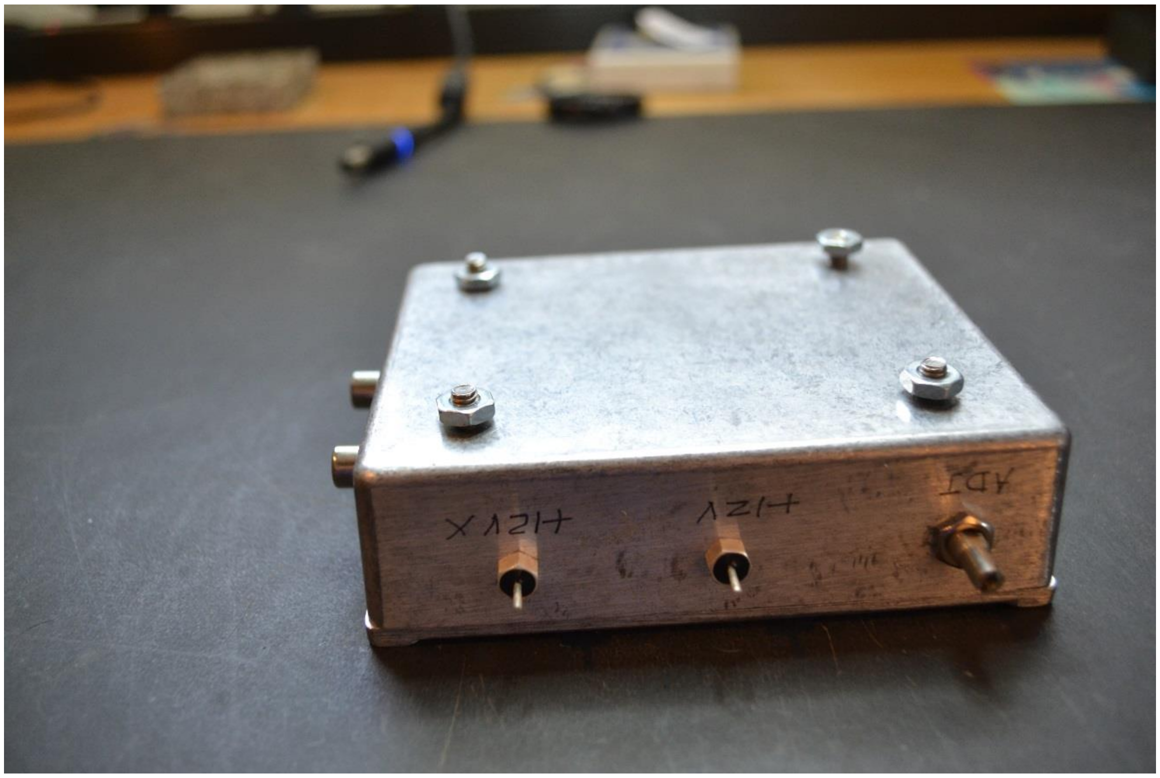
Schematic of the BFO



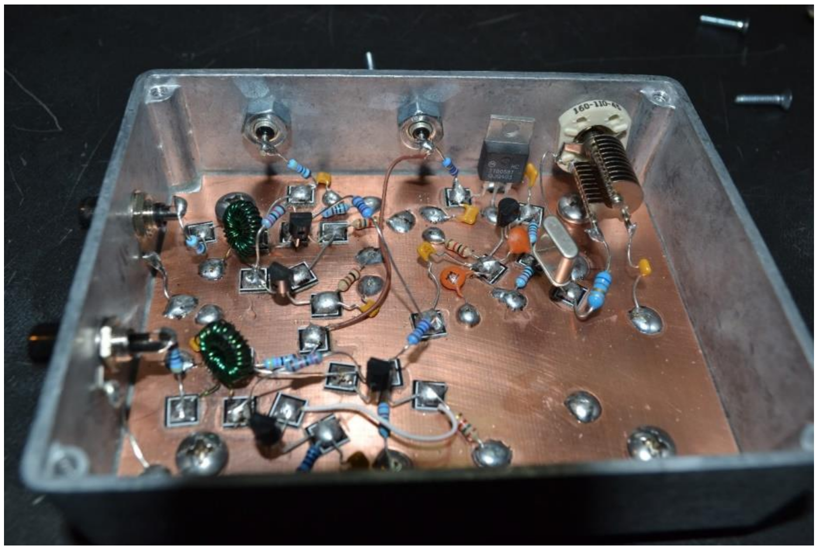
Schematic of the Product Detector



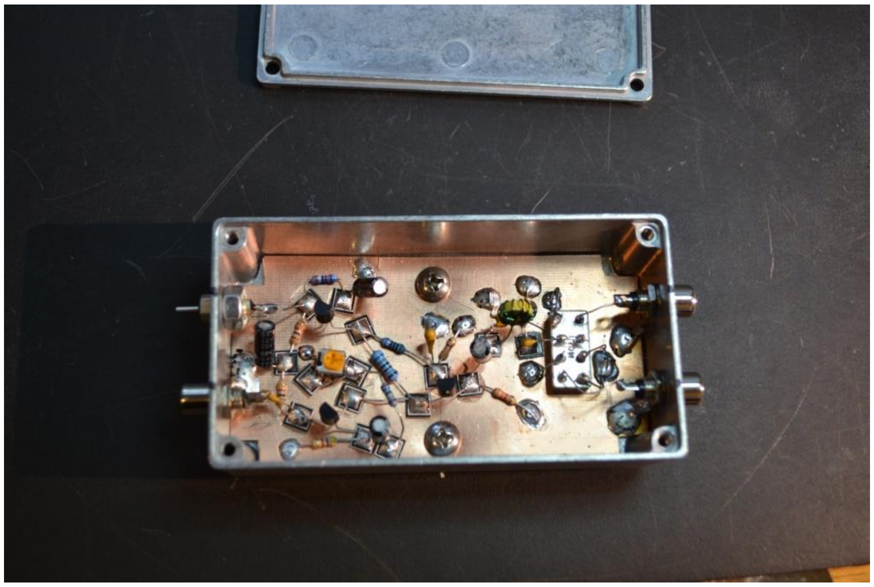
Picture of the outside of the BFO



Picture of the inside of the BFO



Picture of the Product Detector



The “I screwed up story of the quarter” – N1GY

Just a few words on getting messed up at the work bench. Recently I tried building a headset adapter to use the kind of wired headsets that are made for smart phones. I ordered a few headsets and a few TRRS jacks. My first attempt did not work out well. The earphone of the headset had very scratchy audio and I was not sure the mic worked at all. I tried again with a different supplier and had more problems. The audio on receive was much better but no output from the mic at all. I put the whole project away for awhile. Just the other day I was cleaning out my files and came across the data sheet that I had downloaded for the TRRS jack. To my great chagrin I realized that the way I had wired the jack was not right. I had assumed that the numbers 1, 2, 3 and 4 referred to the tip, ring 1, ring 2, and the sleeve in that order. Wrong! For some reason the sleeve is pin 1, tip is 2, ring 1 is 3 and ring 2 is 4. I had to completely rewire the adapter and then test it. The testing is currently on hold because the adapter was designed for my IC-706MKiiG and that radio is currently out for repair. As soon as I get it back, I will test the new headset adapter.

I just figured I would talk about my screw-up in hopes it might prevent someone else from repeating my error. It also pays to read the data sheet before you start a project. Addendum: right after I got my 706 back, the mic selector packed up so I had to build a new one of those. I will get to the adapter soon.

73, Geoff, N1GY

Faster than a speeding bullet, able to bounce off walls in a single bound!
Look! Over by the Arduino.
It's Super sound! ***by Bill Johnson, KI4ZMV***

So you got your first Arduino program working. Great, you now have a thirty-dollar blinking light. How impressive is that? Not very. Perhaps the best ways to show off the power of the Arduino micro controller is to have it do something awesome. Measuring the speed of sound fits that description.

The Project
You can measure the speed of sound using the same principles found in digital tape measures. The difference will be that you fix the distance to a reflecting surface, and measure the time it takes for sound to make the round trip from sender back to receiver. From this you can calculate its speed using the formula:

speed =Distance/time

The Device
The device that makes this all possible is the HCSRO4 transmitter/receiver. I found this one on eBay. Look around. Prices vary. I paid about three dollars.

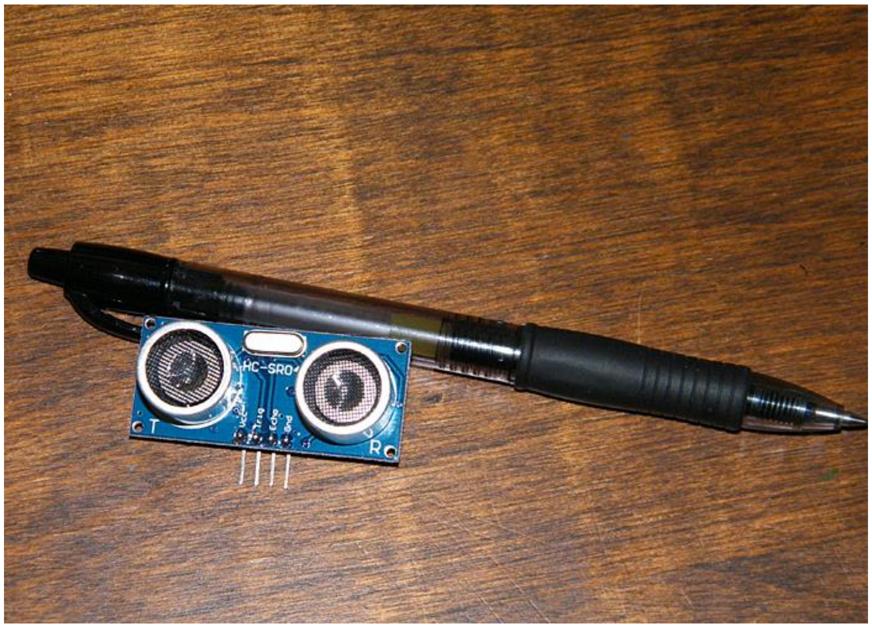


Figure 1. The sensor is relatively small. The left side labeled T is the transmitter; the right, labeled R, is the receiver.

These units have only four connections: ground, echo, trigger, and Vcc. The left side marked T transmits a ping, while the right side labeled R listens for its return. The total distance traveled from the transmitter to a reflecting object and back, divided by the time it takes to make the round trip, is the speed of sound.

Wiring is straightforward. There are only four connections. Connect sensor ground to Arduino ground, Vcc to Arduino five volts, the Trig pin to Arduino pin 13, and the echo pin to Arduino pin ll. Other combinations are possible. How simple is that?

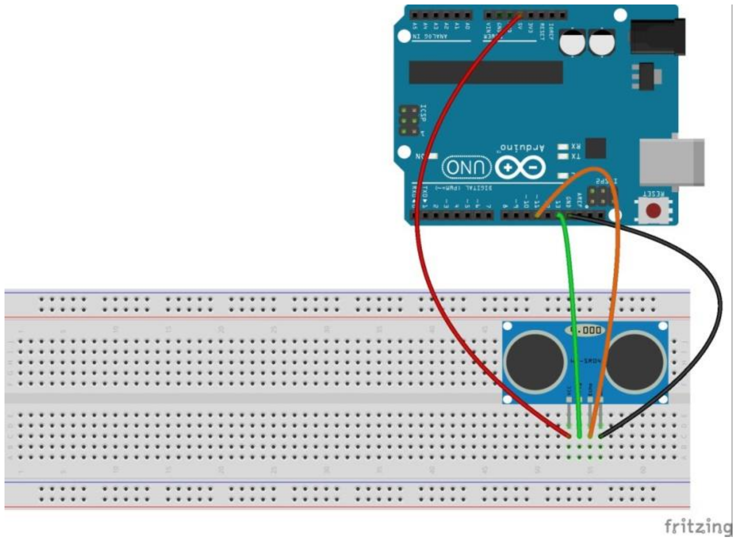


Figure 2. The illustration above from www.toptechboy.com. A slightly modified version of the program, appearing below, also came from this site.

The Test Setup
This is what my six-inch setup looked like. Yes, that is a napkin holder and a box of stick matches. Most likely your setup will be different.

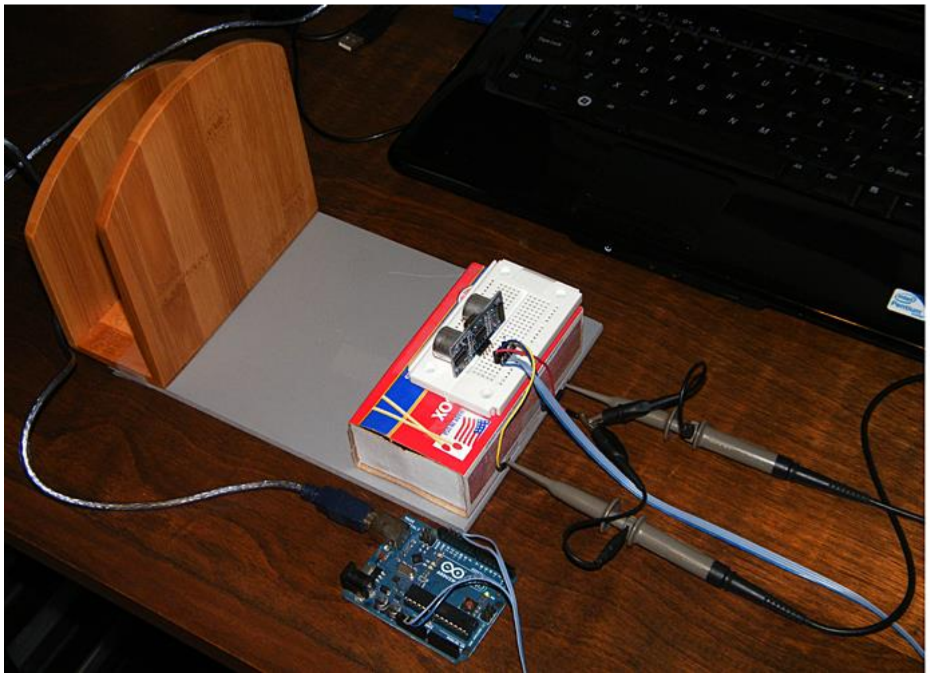


Figure 3 This is the six inch setup. The two probes connect the trig and echo pins to a scope. More below.

To make a reading, the trigger pin is brought LOW with a digital write. A pause follows to let things settle. After the pause the trigger pin is first brought HIGH then LOW again in quick succession. This LOW HIGH LOW sequence initiates, after a fixed delay, a ping and the start of the timing cycle. The ping will travel outward, bounce off a target, and then return, where it is registered by the echo pin. As soon as the echo is received, the timer stops, and the total time is set into the sketch variable called pingTime. PingTime, or travel time is measured in microseconds. Fortunately, the Arduino has a built in pulseIn(pin,state) library command that can be used to accurately measure pulse length.

To get a better picture of what is happening, look at the dual trace scope output below from my Rigol Oscilloscope. Trace 1 shows the short initializing pulse, (upper trace), while trace 2 shows the resultant ping travel time (lower trace).

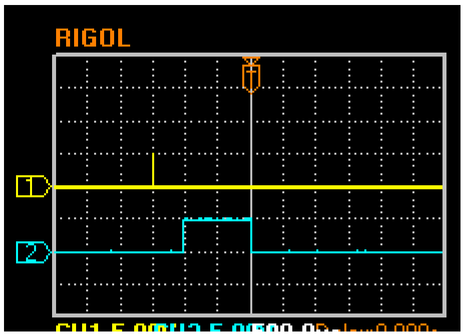


Figure 4. The scope's scale is 500-microseconds per division horizontal, and 5 volts per division vertical.

The short pulse on trace1 was 10-microseconds long. The delay between this start pulse and the beginning of the timing pulse was 460 microseconds. This time is fixed, and independent of both the length of the initial short pulse as well as the measured time. With a distance of six inches from the reflecting surface to the sensors, the travel time was around 870-microseconds at room temperature.

The Arduino Sketch
Start by setting some variables. First, we need variables to identify the trig pin and echo pin. These will be type int. We also need three additional variables, one to represent ping travel time, another to represent the calculated speed of sound, and a third to represent the distance to the target. These will be type float.

In setup we initialize the serial monitor. This will be used for output. This is also the place we set the pin mode for the trig and echo pins.

In loop we bring the trig pin low, wait two seconds, then bring it high for 10 microseconds, then low again. This initializes the pulse read process. (More below.) Next we set the ping time equal to the pulse length reported by the Arduino pulseln function.

Calculation of the speed of sound is a matter of distance traveled divided by time. Distance is twice the target distance in inches, and time is the returned value of pulse length in microseconds. The inches per microsecond must then be scaled to miles per hour. Here is the completed sketch:

The Arduino Sketch

```
int trigPin = 13; //set trig pin to Arduino pin 13
int echoPin= 11; //set echo pin to Arduino pin ll
float pingTime;  //a variable to hold elapsed travel time to and from the target
float speedOfSound; //a variable to hold the speed of sound
float targetDistance=6; //a variable to hold the target distance. This will differ by test condition.


void setup() {
  Serial.begin (9600);  // start the serial monitor
  pinMode(trigPin,OUTPUT); //set the trigPin to OUTPUT
  pinMode(echoPin,INPUT); //set the echoPin to INPUT
}

void loop() {
  digitalWrite(trigPin, LOW);  //pull trig pin low
  delayMicroseconds(2000);  // delay to let things settle
  digitalWrite(trigPin,HIGH); //start initializing short pulse
  delayMicroseconds(10);  //pulse length
  digitalWrite(trigPin,LOW); //pull pulse low
  pingTime=pulseIn(echoPin,HIGH); //set pingTime to measured pulse length
  speedOfSound= 2*targetDistance/pingTime; //calculate speed of sound in inches per microsecond
  speedOfSound=speedOfSound *3600*1000000/63360; // convert to mph
  Serial.print("The speed of sound is "); //this line can be commented out for spreadsheet analysis
  Serial.print(speedOfSound); //print speed of sound
  Serial.println(" miles per hour"); //this line can be commented out for spreadsheet analysis
  delay (1000); //short delay for display purposes
}
```

Note, during testing I found it helpful to comment out the verbiage and print only values. This made transfer to a spreadsheet easier.

The Speed Of Sound And Ambient Temperature

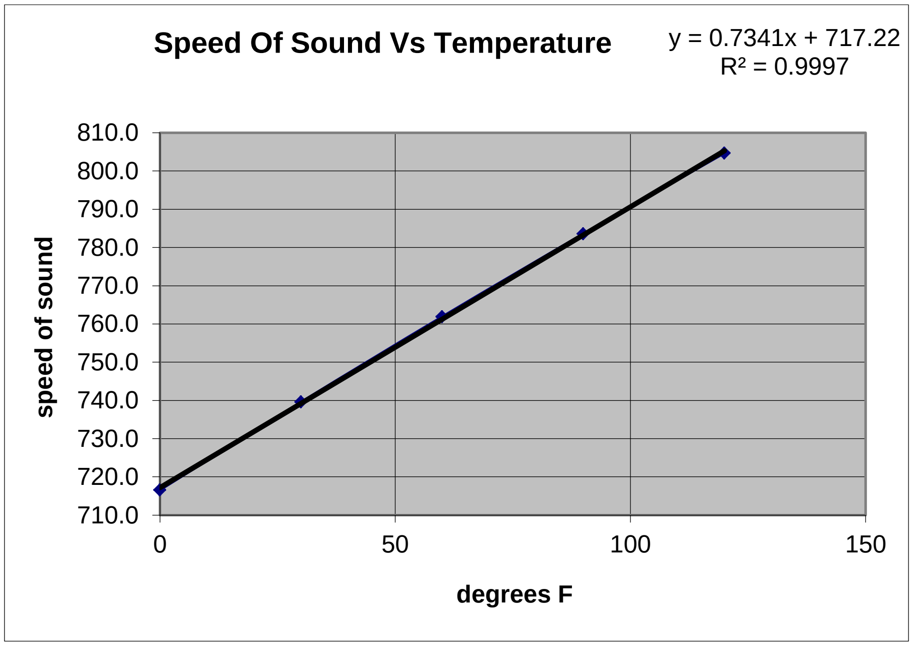
The speed of sound varies with temperature. The following relationship is approximate, but accurate enough for our purposes.

$V=0.7341T_f +717.22$

V is the speed of sound in mph

T_f is the temperature in degrees Fahrenheit

From this we can generate a plot of the speed of sound in miles per hour versus temperature in degrees Fahrenheit, and a brief table to get a sense of the changes you might expect going from room temperature to either lower or higher temperatures.



Degrees F	***Speed of Sound***
0	716.6
30	739.6
60	761.9
90	783.6

Figure 5. Graphical and tabular results from the equation given above. These are the expected values for speed of sound at various temperatures.

From the graph and chart above we should expect the speed of sound to decrease at lower temperatures. Specifically, we would expect a 59 miles per hour decrease in the speed of sound going from an ambient temperature of 80F to 0F degrees.

To test this hypothesis, the apparatus was placed a refrigerator’s freezer section, whose temperature was zero degrees F. Though the data is noisy, it does suggest, that on the average, good agreement between the expected drop of 59 mph. The noise in the data cannot be accounted for at this time. One possible cause might be moisture buildup on the sensor.

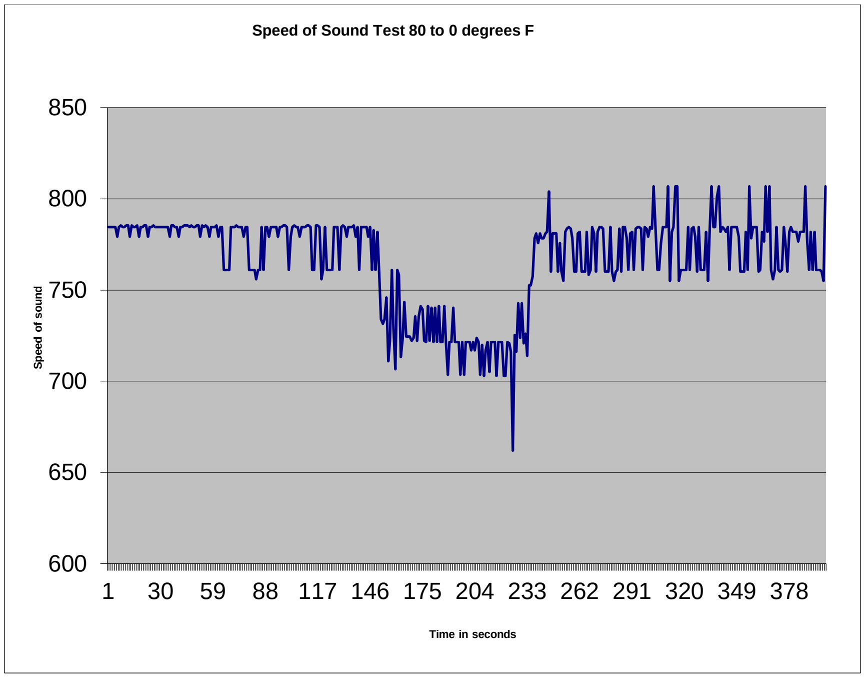
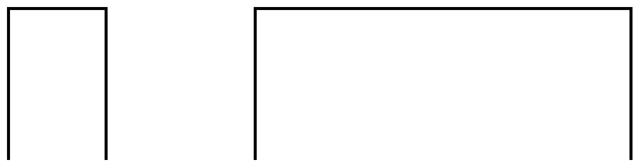


Figure 6. The drop in the speed of sound is obvious.

The Effect Of The Initial Pulse Length

Mentioned above was the apparent independence, within limits, of the initial pulse duration. Trigger pulse lengths of 10 to 200 microseconds were evaluated for their affect on the delay time between the end of the pulse and the start of the timing cycle. None was found.



u-sec	u-sec
Trigger Pulse	Delay Before Timing Start
10	460
20	460

Figure 7. The timing cycle, and the effect of initial pulse/trigger length

The Sensitivity Of Target Distance From the Sensor

The placement of the reflecting surface vis-a-vis the sensor is critical, especially at distances of around six inches. A series of measurements were made to determine the sensitivity of this parameter on the measured speed experimentally. Successive sheets of plywood were added to shorten the path. Each sheet was approximately 0.23 inches in thickness.

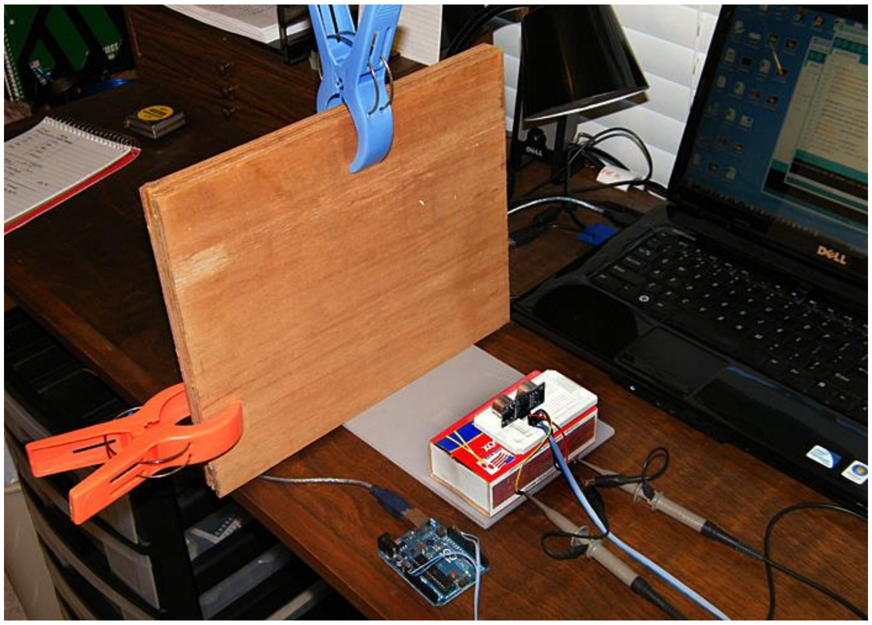
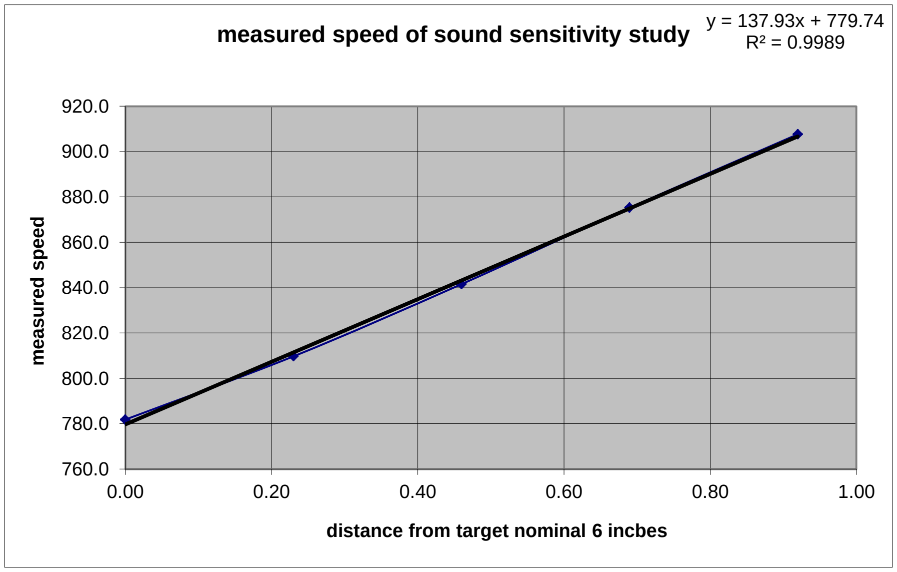


Figure 8. Testing the sensitivity to distance from the sensor at a nominal six inches.

Test results indicate the expected variation in apparent speed due to error in sensor to target distance. From the accompanying chart and its regression equation this is approximately 13.7 mph error for every increment of 0.1 inches. This is quite sensitive.



Dist inches	Speed mph
0.00	781.8
0.23	809.7
0.46	841.6

Figure 9. Sensitivity to distance from the sensor to reflector at nominal six inches.

It is assumed that placement of the sensor would be less sensitive if the distance were farther away. A test was run at thirty-six inches, and the results support the assumption.

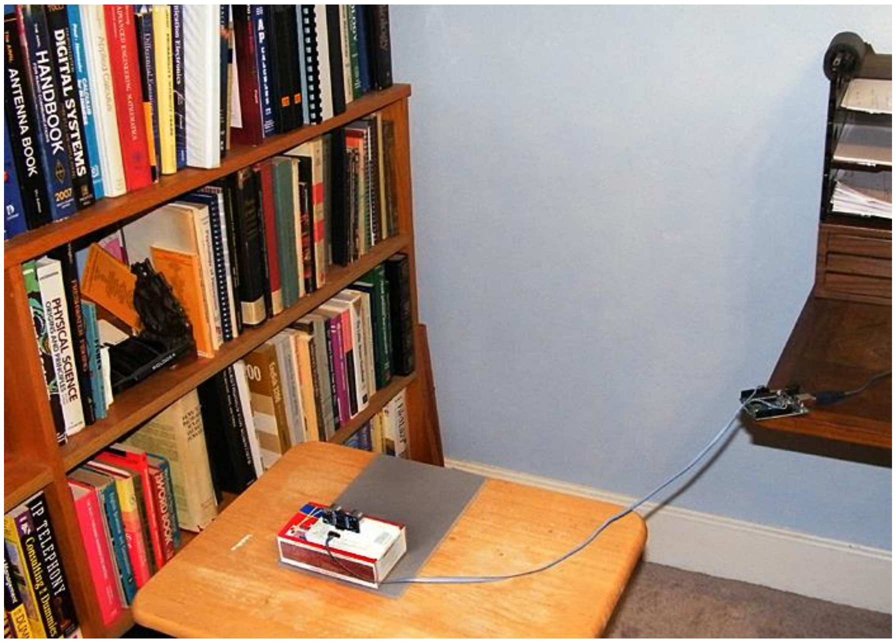
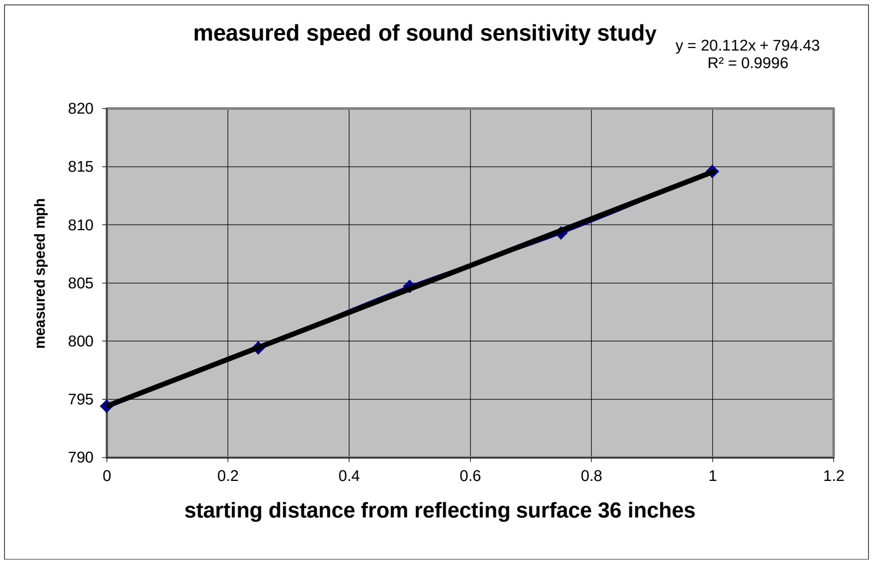


Figure 10. This setup was used to measure the sensitivity of the sensor to target distance with a nominal distance of 36 inches.



Dist inches	Speed mph
0	794.4
0.25	799.42
0.5	804.7

Figure 11. Sensitivity to distance from the sensor to reflector at nominal thirty-six inches

From the chart,, regression equation, and table above it is clear that the sensitivity to distance from the target is much less. The regression equation suggests only about a 2.0 mph error for every 0.1 inches from nominal. We would expect that the sensitivity would be about six times as great for the nominal six inches versus the nominal thirty-six inch setups. Experimentally, the ratio was 6.9 ratio, or an error of approximately 15%.

Conclusion: We have demonstrated a practical way to measure the speed of sound using an inexpensive sensor and an Aduino. The sensitivity to ambient temperature and target distance were also explored.

A good friend of mine told me that if he ever needed to know the speed of sound he would Google it. This is a reasonable answer. You could also tell your grandson to Google it, or perhaps you could introduce him to the Arduino. How cool is that?

BITS AND BYTES – All About Microcontrollers

By Darrell Davis KT4WX

ARRL WCF Section Manager and ARRL Technical Specialist

Welcome back to our next installment of *Bits and Bytes*. I hope you find this column as enjoyable to read as it is for me to write.

MICROCONTROLLER TERMS: Let us define some more basic microcontroller terms you should be comfortable with.

- **<u>I/O Ports:</u>** This refers to input and outputs to the microcontroller. “I” stands for an input pin and “O” stands for an output pin. Without input or outputs, the usefulness of a microcontroller, or microprocessor for that matter, would be very limited and nearly useless. Input and Output pins have two possible inputs: Low or High. Low is typically at signal ground and High can be several different voltages with respect to signal ground. Typically this is either +5VDC, +3.3VDC, +2.8VDC, or even +1.8VDC, with respect to signal ground depending upon the microcontroller design. Originally nearly all microcontrollers had +5VDC as universal high on a input or output pin but today this is becoming less and less common in newer microcontroller designs. However, at present, there are still many microcontroller designs still out there that use 0VDC and 5VDC to represent low and high respectively. Each microcontroller will tell you in its respective data sheets what pins are input pins only and what pins are output pins only. Some microcontrollers I/O pins can function as one or the other depending upon how they are set by the programmers code. This feature is becoming very common to almost ubiquitous in newer designs as well.

- **<u>ADC</u>**: This is analog to digital conversion. These are microcontroller pins that are dedicated to converting an analog signal of a particular voltage range into a digital value, typically a numerical value. More on how this works later. Many microcontrollers will have one or several ADC port pins, depending upon the design.

- **<u>DAC:</u>** This is digital to analog conversion. This is the exact opposite of ADC. This is where a digital numerical value is converted into an analog voltage value. Many microcontrollers will also have one or several DAC port pins, depending upon the design.

***FEATURED MICROCONTROLLER – ARDUINO*:** Last time we introduced the PICAXE. This time I will introduce the Arduino. The Arduino is a microcontroller platform, that was originally based upon the Atmel AVR series of microcontroller, manufactured by Atmel Semiconductor. The Arduino is an open source platform for hardware and software development, thus making project development much easier. The Atmel AVR microcontroller is programmed with the Arduino programming language, a C++ cousin, based upon the Wiring program language. One programs the Arduino in the Arduino Programming IDE, based upon the Processing programming language.

Arduino boards are a standardized hardware platform used to implement Arduino projects. There are several standard Arduino hardware platforms out there. The most common hardware platform is the Arduino UNO R3. What is nice about the Arduino is the microcontroller programmer is integrated onto the Arduino board so no external programmer is necessary. All the Arduino boards come with this feature.

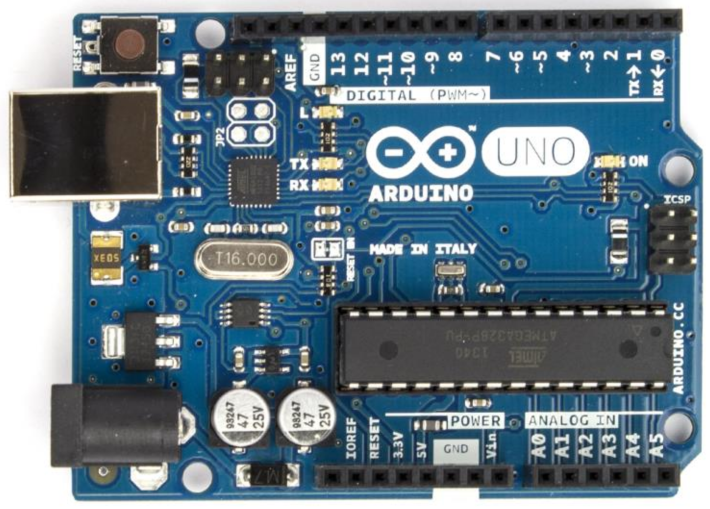
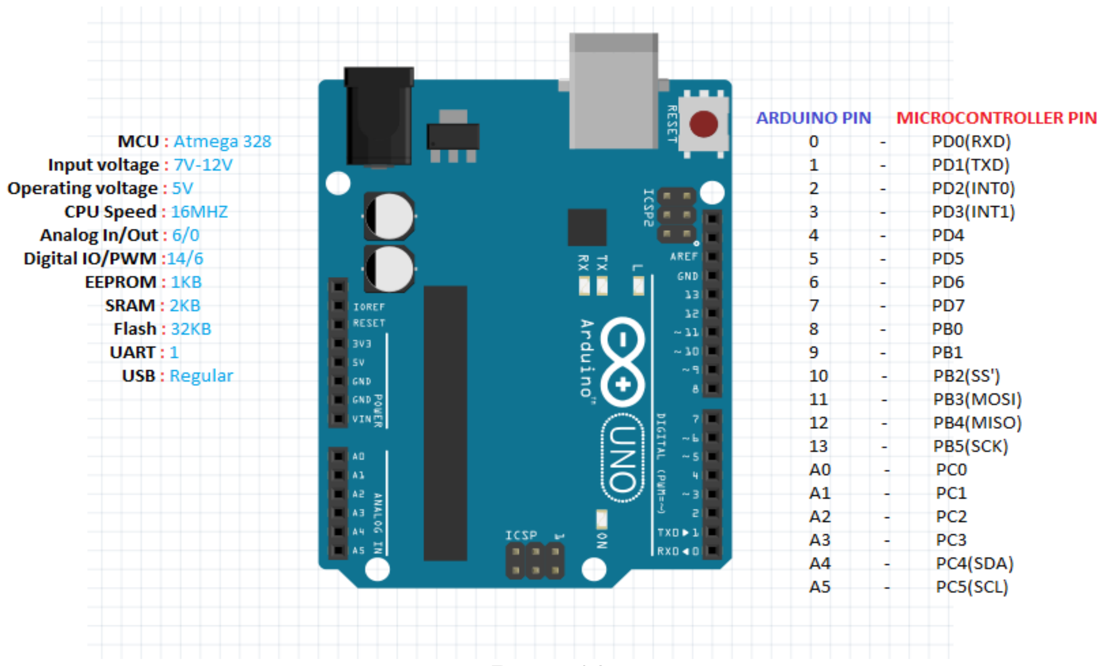


Figure 1: The Arduino UNO R3 – Top View



MCU : Atmega 328
Input voltage : 7V-12V
Operating voltage : 5V
CPU Speed : 16MHZ
Analog In/Out : 6/0
Digital IO/PWM :14/6
EEPROM : 1KB
SRAM : 2KB
Flash : 32KB
UART : 1
USB : Regular

ARDUINO PIN		MICROCONTROLLER PIN
0	-	PD0(RXD)
1	-	PD1(TXD)
2	-	PD2(INT0)
3	-	PD3(INT1)
4	-	PD4
5	-	PD5
6	-	PD6
7	-	PD7
8	-	PB0
9	-	PB1
10	-	PB2(SS')
11	-	PB3(MOSI)
12	-	PB4(MISO)
13	-	PB5(SCK)
A0	-	PC0
A1	-	PC1
A2	-	PC2
A3	-	PC3
A4	-	PC4(SDA)
A5	-	PC5(SCL)

Figure 2: Arduino UNO R3 Specifications and Pinouts

The Arduino UNO can even be purchased at Radio Shack (as of this writing). There is also the Arduino NANO and Arduino DUE that are common place as well. There are more models out there but these are some of the most common.

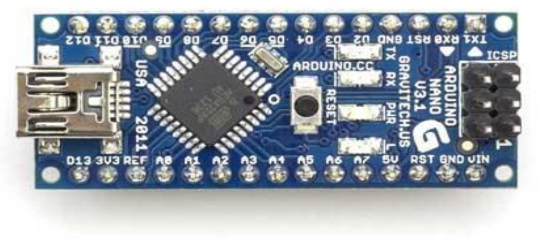
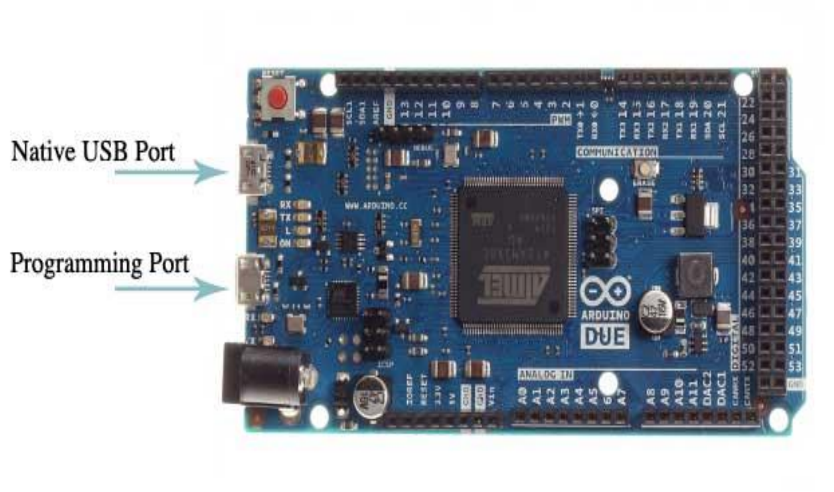


Figure 2: Arduino NANO – Picture Not To Scale

The Arduino NANO can be incorporated into any project that can accommodate a 40 Pin DIP integrated circuit. and the



Arduino DUE is an ARM based implementation of the Arduino and has many more ports and "horsepower" for higher end projects as well. The regular Arduino boards have a clock speed of 16 MHZ, the DUE has a clock of 84 MHz, over 5 times faster. This allows the Arduino to take advantage of the speed and power of the ARM Cortex series of microcontrollers.

You may view my OpenOffice Impress presentation that I gave at the last West Central Florida Section Technical Conference and at several local club meetings.

http://arrlwcf.org/download/wcftechconference_2015/IntroductionToTheArduino.odp.

The slide show is in .odp format and you will need to view it in OpenOffice which you can download and install for free from ***http://www.openoffice.org***. In the slide presentation, there is more information about the Arduino are some good resources mentioned concerning the Arduino. That is all for this installment of *Bits and Bytes.* Until next time, keep your soldering iron hot and your microcontroller code coming.