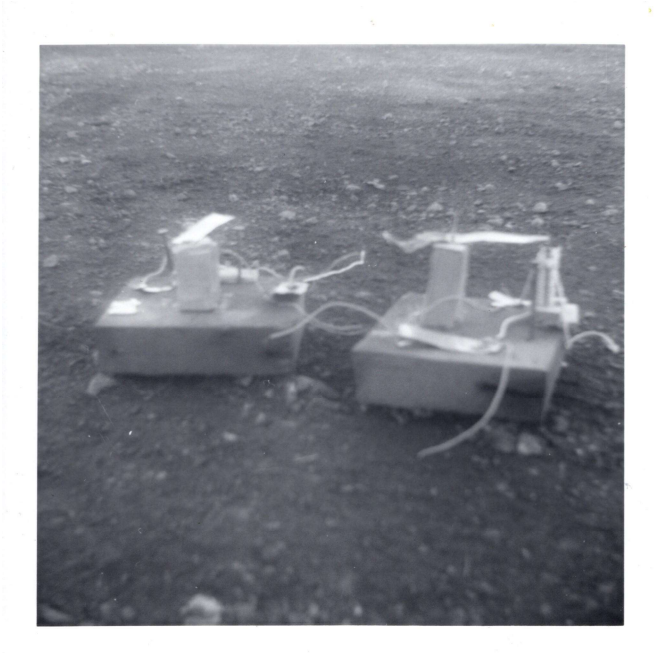
Morse Code Responder
By
Lloyd Johnson

This document is about an Arduino based project that incorporates antique telegraph key/sounders. The project evolved from an earlier project that I call the Morse Code Time Chime or time chime for short. The time chime uses an Arduino, a real time clock module and a relay module to tap out 2 Morse Code digits every hour on the hour. The digits correspond to the current hour in military time format with values from 00 to 23.

The Morse Code Responder contains all the features of the time chime and has the added feature of responding to someone that taps a telegraph key. If someone taps the telegraph key of either of the telegraph key/sounders connected to it, after a 10 second delay, the Morse Code Responder will make both telegraph sounders sound out a random Morse Code message. The message sounds are intended to sound as if someone had tapped it from a remote telegraph key. The messages are short and sent with a dit time of 0.2 seconds and a dah time of 0.6 seconds. Currently, the messages only contain alphabetic letters. Sometime in the future, the messages might include numbers and punctuation but until I learn Morse Code, the messages are limited to letters. A typical message might be: “PLEASE REPEAT LAST MESSAGE”. This is meant to seem like something intelligent is responding.

As a kid, when I finally figured out that a telegraph set is nothing more than an electromagnet with the switch at a remote location, I set about making my own. I was about ten years old (give or take a year) when I took the following photograph. I’m now seventy years old making the following photograph about sixty years old.



I took this photograph using a Kodak Brownie that did not have a flash. All pictures I took with this camera were outside to take advantage of the natural light. The telegraph key consisted of

metal cut from tin cans. The sounder also used some of the same tin for the pivotal beam balanced on a wood block. The coil was several layers of lamp cord wrapped around a large nail. The base was a piece of a 2x6 plank rescued from the firewood pile. It worked (sort of). I recall having one in my room at the top of the stairs and the other at the base of the stairs on the kitchen table. I think I was using a 6V lantern battery to power it. It never worked very well since the pivotal beam (the piece of tin coated steel that gets attracted by the electromagnet) was getting magnetized and wasn't returning when the key was released. Anyway, I attribute that adventure to the start of my interest in electrical engineering which later led me to a BSEE in 1975 followed twenty years later with a MSEE in 1995.

Several months after receiving my MSEE, I changed jobs within the Boeing company. I went from developing test software and test equipment interfaces for aircraft electronic subassemblies to testing software interfaces on the International Space Station (ISS). This transfer required a move from Seattle to Houston. With my new job no longer involving electronics, electronics returned to being a hobby focus instead of a job focus.

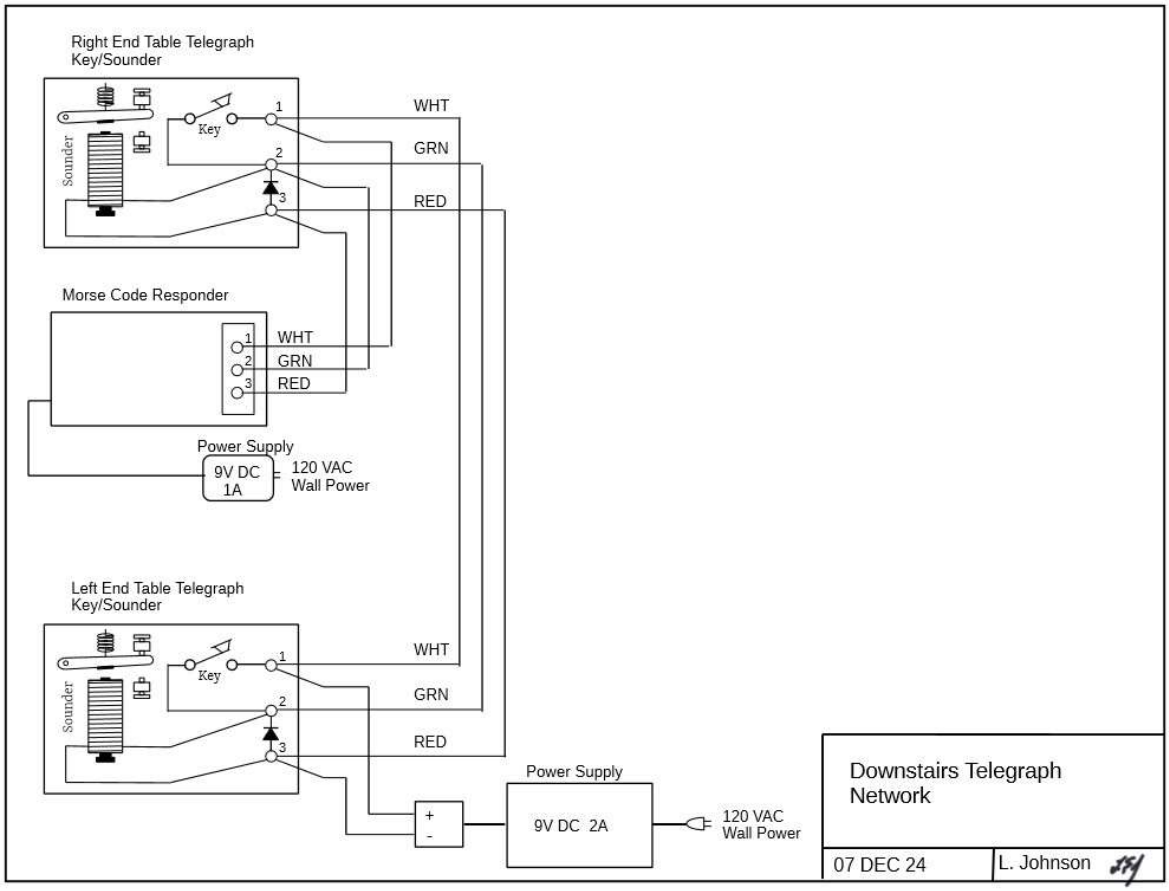
I embraced my renewed interest in hobby electronics by focusing on antique electronics - specifically old radios. The following picture was taken in Houston. The radio on the far left of the piano is a crystal diode radio I built. Next is a crystal diode radio built from a kit that was purchased from eBay. This is followed by a Zenith Trans-Oceanic also purchased from eBay. And finally, there is a one tube regenerative AM radio purchased from the internet. I later learned that the guy selling them was a Boeing retiree who worked in the ISS communications group in the same building as me. He has an interesting website at http://www.xtalman.com/. I had him autograph the bottom of my radio.

In front of the radios is a pair of telegraph key/sounders. I had them connected to each other and a couple of D batteries. For whatever its worth, I was able to send Morse Code from one end of my piano to the other. On the pink foam between the telegraph key/sounders are five RF transmitter and receiver modules. These were purchased from the internet with the idea they could be used with the telegraph key/sounders to do wireless communication. I might still try and do this someday. Meanwhile, the RF transmitter and receiver modules were used in Arduino based sensor modules I designed, built and have situated around our house and property. These modules transmit light, temperature, humidity and motion data to Arduino based receivers which transmit the data via RS-232 to a desktop computer that logs the data. This is another project which I might someday document in depth.

After I retired, we moved to central IL. The telegraph key/sounders now live in our basement on end tables on either side of a sofa.

I refer to them as the left end table and right end table respectively. On the lower surface of the right end table is a board with the electronics for the Morse Code Responder. I eventually plan on putting this in a more permanent enclosure. I might put a fake antenna on it to give the illusion you are communicating with someone remote.

The next drawing is the telegraph network I have in our basement. I used LibreOffice Draw to make the next two drawings on an old Dell laptop running Linux Mint. I wasn't sure how to represent a telegraph key/sounder in a schematic, so I did a little exploring on the internet. Most of the images I found for telegraph key/sounders in schematics have a mechanical representation of the sounder's coil and pivotal beam. I tried to duplicate that in the Downstairs Telegraph Network drawing that follows:



The telegraph key/sounders are very simple devices. The key is basically a momentary contact switch between contacts 1 and 2. The coil is between contacts 2 and 3. The telegraphs are powered by a power supply connected to contacts 1 and 3 of the Left End Table Telegraph Key/Sounder. When the key is pressed, contacts 1 and 2 are now connected resulting in a voltage across contacts 2 and 3 which energizes the coil. The resulting magnetic field causes the pivotal beam of the sounder to come down and make loud contact with the adjustable stop. When the key is released, the voltage is removed, and the spring pulls the pivotal beam back up resulting in another loud clack with the upper stop.

I added a diode between contacts 2 and 3 of each sounder. The purpose of this diode is to allow current to flow through the coil after the key is released. If the diode was not there, a large negative voltage spike results that would damage modern electronics. When I made the Arduino based Morse Code Time Chime, I did not have the diode between contacts 2 and 3. The relay contacts of the 4 Chennel Relay Module were rated for 250 V AC 2A, so I didn't think the diode was necessary. After a few days of operation, I heard the relay on the 4 Channel Relay Module click, but the sounders were silent during some of the Morse Code characters. At that time, I was using six 1.5 V D cells connected in series as my power supply. I first suspected the batteries were drained and didn't have enough energy to activate the sounder.

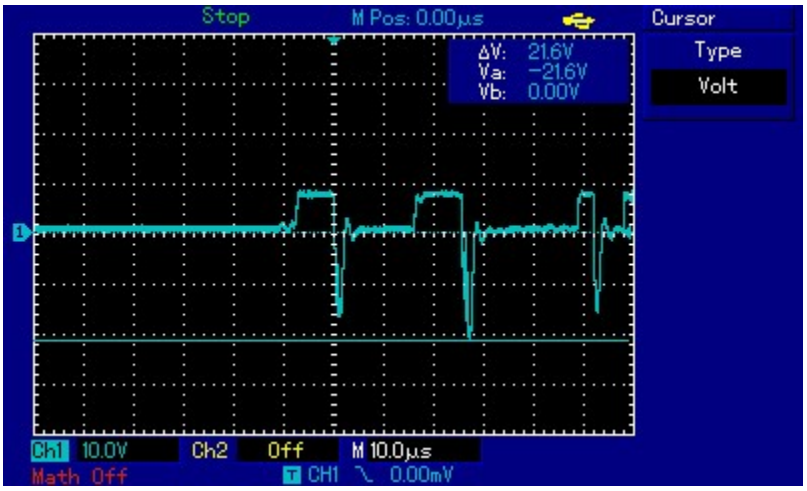
Yet, when the telegraph keys were pressed, the sounders activated just fine ruling out the drained battery theory. Nevertheless, I replaced the batteries and when that didn't help, I replaced the battery power supply with a 9V DC 2A power supply that is plugged into wall power. This did not solve the problem either.

It wasn't until I was testing the optical isolator that I discovered the source of the problem. I remembered enough about the negative spike to consider an optical isolator but failed to realize the damage that spike could cause. The optical isolator light source was a forward biased LED in series with a 220 ohm resistor connected across pins 2 and 3 of one of the sounders. When I pressed the telegraph key, the LED illuminated like it should. I released the key, and the LED turned off which is still good. However, when pressing the telegraph key again, the LED did not illuminate. [1]A single negative spike (that occurred when the key was released) was enough to zap my LED into a nonfunctional state. These spikes also destroyed the relay contacts of one the relays on the 4 Channel Relay module. This was the problem I was observing with the Morse Code Time Chime.

I now have a diode connected across the coil of each sounder. I also put a diode across the LED and series resistor in the optical isolator circuit.

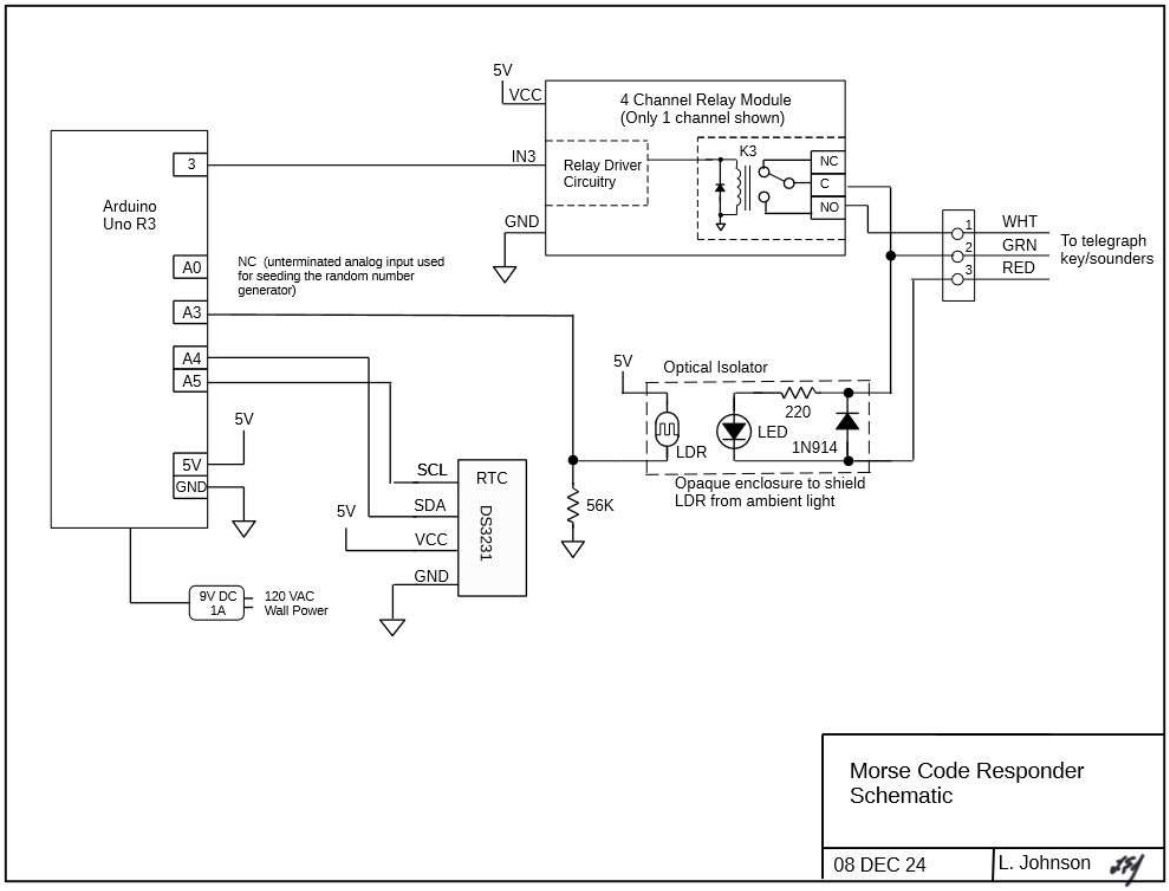
---

[1] The equation for the voltage across an inductor is v=L(di/dt) where L is the value of the inductor's inductance and di/dt is the differential change in current divided by the differential change in time. When a telegraph key is released, the current changes from a couple of amps to 0 amps in a very short time (by short, I mean as in 0 seconds). Any value divided by 0 results in infinity and because the change is negative, it is negative infinity. Once the voltage magnitude reaches some value, the air between the separating contacts is ionized and the energy is discharged through the spark. The values measured with a digital storage oscilloscope (DSO) varied from 20 volts to 50 volts. The following DSO screen shows where -21.6 V was measured.



This is enough to destroy the LED but probably not enough to destroy relay contacts. I suspect some voltage spikes are much greater but due to their short duration, it may be beyond my measurement equipment to capture it.

The next drawing is a schematic of the Morse Code Responder. The Morse Code Responder consists of an Arduino Uno R3, a RTC (Real Time Clock), a 4 Channel Relay Module and an optical isolator. The schematic for the Arduino Time Chime (if it existed) would be identical except it would not include the optical isolator.



The RTC will provide the time to the Arduino when the appropriate function is called by the Arduino software. After some exploring on the internet, I found an example for using an RTC and downloaded the libraries they recommended. The example I used is found at https://www.instructables.com/How-to-Create-a-Clock-Using-Arduino-DS3231-RTC-Mod/. It should be noted that the RTC module requires a battery. I used a CR 2016 battery. Once the time is set in the RTC module it does not need to be set again even if the power is removed from the Morse Code Responder. This is because the CR 2016 battery on the RTC module keeps the RTC running even when its external power is removed.

The 4 channel relay module is a bit of overkill. I only needed one channel for energizing the telegraph sounders. Having an extra channel was convenient when the relay contacts of a channel were accidentally destroyed by not having a diode across the telegraph sounder coils. I chose a 4 channel relay module because I had extra modules that were purchased for another project. The extra channels provide the option to turn on or off other electronic devices at

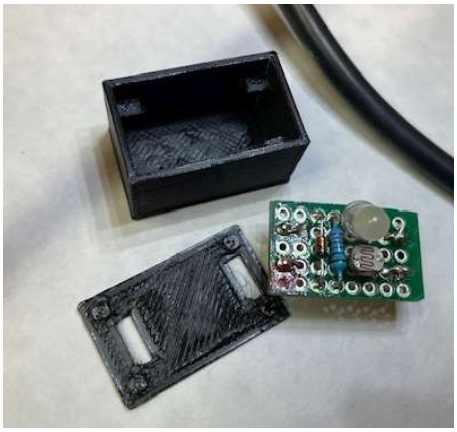
specified times and specified durations. These electronic devices include an LED hourglass and an LED Christmas tree I built from electronic kits over the past few years. The Morse Code Responder might be updated to turn on the LED hourglass briefly each hour and turn the LED Christmas tree on in the morning and off at night (during the Christmas season). The extra channels make this easy to do. But that is for the future.

I am currently using the 3rd channel of the 4 channel relay module for activating the telegraph sounders. This is done by connecting the common and normally open (C and NO) contacts of relay K3 across the contacts of the telegraph keys. When relay K3 closes, it has the same effect as someone pressing a telegraph key. Relay K3 is closed whenever the Arduino puts a logic low on pin 3 which is connected to the IN3 input of the 4 channel relay module. This is done with software that is described later.

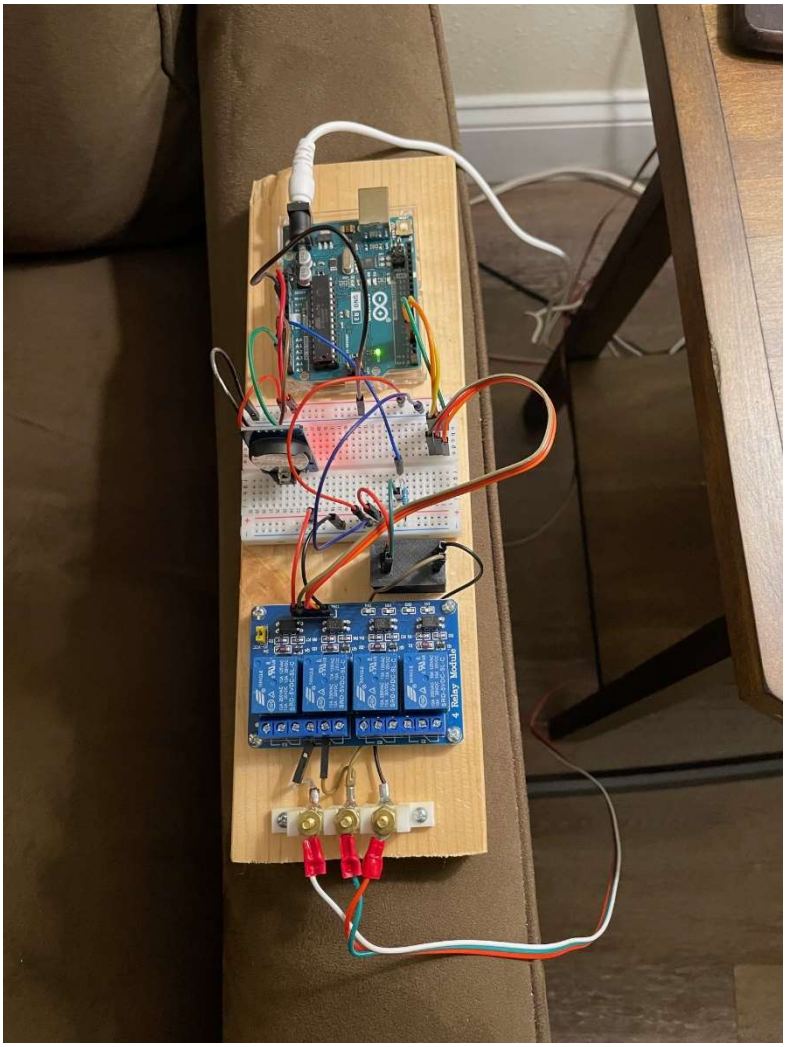
For the Morse Code Responder to respond to telegraph key taps, the Arduino must have an indication that the telegraph key is pressed. I knew about the negative voltage spikes that result from releasing the telegraph key, so I did not want a direct connection between the Arduino and the sounder. I planned to protect the Arduino from these spikes by using an optical isolator. Optocoupler isolation boards are sold on the internet in packages of five. I bought a package of five although I wasn't totally sure how I would adapt them to this project. While waiting for them to arrive, I watched a YouTube video that showed how to make an optical isolator with an LED, an LDR (Light Dependent Resistor) and a few resistors. I decided to make one of these instead.

Ideally, the LDR has a resistance of several hundred mega-ohms when there is no light and very low resistance when exposed to light. I arbitrarily used a 56K ohm resistor to make a voltage divider circuit with the LDR. The analog inputs on the Arduino have 10 bit resolution and will return integers from 0 to 1023. When the LED was on, Arduino analog input A3 was returning values over 1000. With the LED off, the value was 0. I'm not completely sure how fast the LDR response time is and whether it is fast enough to accurately transmit Morse Code. For this implementation, it does not need to be fast. All that is required is to alert the Arduino software that a telegraph key was pressed. The 220 ohm resistor in series with the LED is there to limit the current. With voltage drops in the wire and power supply internal resistance, the voltage at the input to the optical isolator when a telegraph key was pressed was around 5 volts. The LED forward voltage when using a 5-volt supply and a 220 ohm resistor was 2.9 volts. The voltage across the 220 ohm resistor was measured as 2.1 volts. The current flow through the LED was then calculated as 9.6 mA. The LED was shining bright. This was all expected.

I had built the optical isolator on a small, perforated circuit board. I then designed an opaque enclosure for the circuit board and printed the design with a 3d printer. The following pictures are of the optical isolator components and the optical isolator assembled.



Here is a more detailed picture of the Morse Code Responder in its current configuration:



The terminal strip on the bottom of the picture where the white, green and red wires are connected was printed with my 3D printer. The terminal strip will make it convenient when I disconnect the Morse Code Responder to bring back to my office for software updates. The connections above the terminal strip go to the 4 channel relay module and the optical isolator. The bread board between the Arduino module and the optical isolator holds the RTC module and the 56 kilo-ohm resistor. The breadboard is also used to distribute signals, 5 VDC, and ground to the RTC module, the 4 channel relay module and the optical isolator. If the functionality of the Morse Code Responder is expanded, the breadboard will make those updates easier.

The software running the Morse Code Responder was written in C and was developed on the Arduino IDE (Integrated Development Environment). An Arduino sketch has two sections: setup and loop.

Code that is in the setup section is only executed once when the Arduino is powered on or reset. The setup section contains the function calls for initializing the Arduino serial interface and the RTC (Real Time Clock) interface. The RTC was set with a different Arduino sketch earlier. The Arduino input/out pins that are used to control the relays are configured as outputs by the calls to the function pinMode. When a pin is defined as an output it puts a logic low (0 VDC) on the pin. Unfortunately, this is what energizes the relays so a call to the function digitalWrite is made for each pin in the setup section to set it as a logic high (5 VDC). I defined the constants, RLY_ON and RLY_OFF as LOW and HIGH respectively to make the code more understandable. Since the Morse Code messages are chosen randomly, it is necessary to seed the random number or else the same sequence of messages would result each time the Morse Code Responder is powered on. The recommended way to do this is to perform the analogRead function with an unterminated pin as the argument. The value returned is random (between 0 and 1023) and serves as an argument for the randomSeed function. Now each time the Arduino is initialized a new random sequence of messages will result after a telegraph key is pressed.

The loop section as its name states is the section of code that when execution is complete, it loops back to beginning of the section and runs it again. At the end of this section, I put a 1 second delay so unless there are other delays (and there will be if the relay is commanded to close and open) each loop completes execution in one second.

The code to make the sounders chime the time is executed when the time chime flag, chimeTime is set to TRUE. This flag is set to TRUE whenever both the minutes and seconds are both 0. The current time is obtained by a function call to library function DS3231_get(&t). This accesses time from the RTC module and deposits the value into the time structure, t. This is done once a second near the beginning of each loop just after first setting chimeTime to FALSE. The code that is executed when chimeTime is TRUE consists of 24 different cases corresponding to the hour in 24 hour format (00 to 23). Depending on the hour value, a different sequence of function calls to dit, dat and delay functions are called. The sequence corresponds to the Morse code description for the hour numbers. The details of the dit and dat functions will be described later. Between the two digits corresponding to the hour, a time delay function is called. The length of the delay is equal to 3 times the duration of a dit. This is standard per the entry for Morse Code on Wikipedia.

The code that makes the sounders respond with a random message is executed when the counter start flag, cst is TRUE and the variable, dtc (dead time count) exceeds 100. Each count of the dead time count represents 0.1 seconds so a count of 100 represents 10 seconds. A telegraph key tap results in the counter start flag, cst getting set to TRUE and the dead time count, dtc being set to 0. This means that 10 seconds of no tapping must occur after the last telegraph tap before the Morse Code Responder responds with a random message. A random number between 1 and 10 is generated and a switch/case structure is used to call a function for the message corresponding

to the random number. I had a few messages in mind when I wrote this section of code whereas other messages were made up on the spot. The messages are:

Message Number	Function Name	Message
1	Msg01()	THE QUICK BROWN FOX
2	Msg02()	JUMPED OVER THE
3	Msg03()	LAZY BLACK DOG
4	Msg04()	WHO ARE YOU
5	Msg05()	WHERE ARE YOU
6	Msg06()	I AM AN ARDUINO
7	Msg07()	PLEASE REPEAT LAST MESSAGE
8	Msg07()	WHAT YEAR IS THIS
9	Msg09()	I DO NOT UNDERSTAND
10	Msg10()	ARE YOU ALIVE

The first three messages consist of a well-known phrase where every letter in the alphabet is used. Some of the other messages are meant to simulate an intelligent response whereas others are just meant to be strange. At this time, numbers or punctuation cannot be part of any message.

The loop section ends with a monitored delay of 1 second. This is performed by making 10 consecutive calls to the function tp100ms (time passes - 100 milliseconds).

Following the loop section are the functions definitions. There are 39 function definitions. All function names are preceded by the word, “void” indicating that these functions do not return any values. Since I’ve defined variables at the beginning of the program before the setup and loop section, these variables are now global and there is no need to return them as values or pass them as arguments.

The first two functions are the dit and dah functions. These functions issue the digitalWrite command to open relay, K3 and close relay K3 with a delay in between. The delay time for the dit function is expressed by the variable dit_del (for dit delay). This value is set at the beginning of the program. It is currently set at 0.2 seconds. The delay times for all the Morse code messages are based on multiples of the dit delay, variable dit_del. The time delay in the dah function between the relay close and open commands is 3 times the dit delay. There is also a delay of dit_del seconds following a dit or a dah. The spacing between Morse Code characters (combinations of dits and dahs) is also 3 times the dit delay.

The next ten functions are the messages and are named Msg01 through Msg10. Each of these functions makes calls to the letter functions. The spacing between words is 9 times the dit delay. The spacing between the letters is 3 times the dit delay. This delay is contained within the letter functions.

The next twenty-six functions are the letters themselves. Each function is named with the capital letter of the letter it represents except the letter, F where it needed to be named f() instead of F() since F() is already used somewhere (maybe in a library).

The last function is tp100ms (time passes – 100 mSec). In this function, a check is made of the analog pin connected to the optical isolator.. If the value exceeds 15 then a telegraph key was tapped. The counter start flag, cst is then set to TRUE and the dead time count (dtc) is set to zero. The variable, cst is tested and if TRUE, the dead time counter is incremented. The function then calls the delay function with an argument of 100. This results in a delay of 0.1 seconds.

That is a description of the program as it stands now. A major update that may or may not ever happen is to attempt to read Morse Code from the optical isolator and then act on it. My initial goal would be to react to a number that was tapped in by responding with the message number corresponding to the number tapped.