Arduino CW Serial IDer

This simple IDer has many operating uses in the shack.

Bob Anding, AA5OY

Over the years, I have written several IDer programs for various microcontrollers, including PIC, Atmel, and now the Arduino. The Arduino platform has become very popular in recent years, due in large part to the support of an enthusiastic community of technical and non-technical members. The Arduino was developed as a teaching platform for students, artists, technologists, and anyone interested in dabbling in electronics.

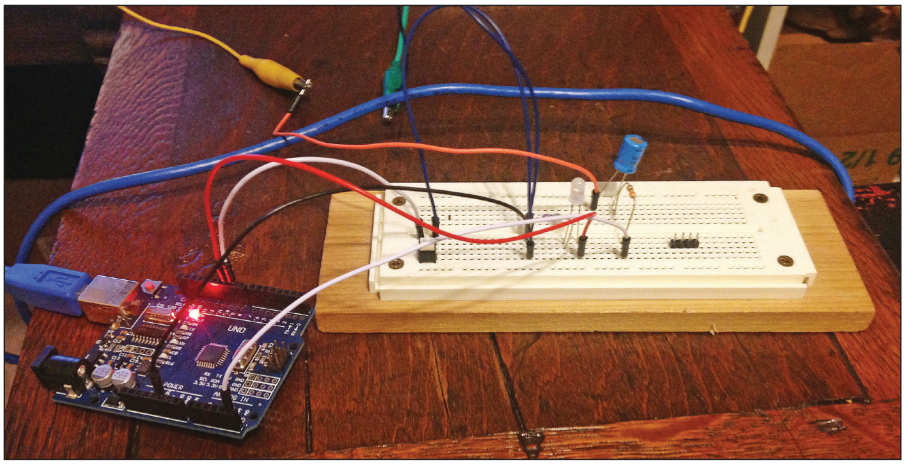
Figure 1 — A breadboard of the CW IDer.

Morse code is a sequence of *dits* and *dahs* that a computer can easily encode as ones and zeros. Once the CW message is encoded and stored, it can be played back reliably. This is the basis for my CW IDer program.

The Hardware, Software, and Interfaces

Figure 1 shows a breadboard prototype of my CW Serial IDer interface. Just load the program, start Arduino in the program mode, and answer a few questions. A simple IDer sketch — an Arduino software program — is available on the *QST* in Depth web page.[1] You can incorporate it into your project. This version of the IDer can find many uses in the shack or on Field Day. It can be adapted to repeater use by simply interfacing the repeater carrier-activated squelch (CAS) with the IDer activation circuit, and the repeater PTT (push-to-talk) with the IDer PTT circuit.

Arduino Hardware

The Arduino has two basic components. The first is the Arduino board, which comes in a variety of sizes and complexities. The board with all its features is supported by the Arduino organization.[2] The boards are open-source and built by a variety of manufacturers around the world. Price ranges from $25 for an official Arduino UNO board, down to less-expensive boards found on **ebay.com**. In addition to the main Arduino board, a host of shields is available. *Shields* are daughter-boards used to interface with the Arduino and enhance its capabilities.

System Software

The second component of the system is the software. The software was developed as an open-source code project. You can download the Arduino integrated development environment (IDE) from the Arduino software download web page.[3] Download the IDE and follow the instructions to install it. Connect the USB cable between the Arduino USB port and your computer. The USB cable powers the Arduino while you are testing the sketch. The system is simple to install and simple to hook up. The IDE software contains fully functioning examples of code sketches. These sketches can be loaded into the Arduino and run. The example sketches are like a set of LEGO® interlocking bricks that you can connect to form a more complex piece with additional features.

Interfaces

Arduino produces different types of boards that lend themselves to different applications. I chose the Arduino UNO R3 for this project. The UNO has several prewired interfaces. The first is the USB connector for interfacing with your computer to download sketches from the IDE software. There is also a RESET button and two rows of general purpose input/output (GPIO) pins.

I was able to put together a working CW IDer using just four GPIO pins and a simple circuit. The schematic (see Figure 2) shows a circuit consisting of an activation switch S1, active message (PTT) LED DS1, and an audio output to speaker SP1. Figure 3 shows a PC board layout for the CW IDer, also available on the *QST* in Depth web page.

Pressing the IDer activation switch

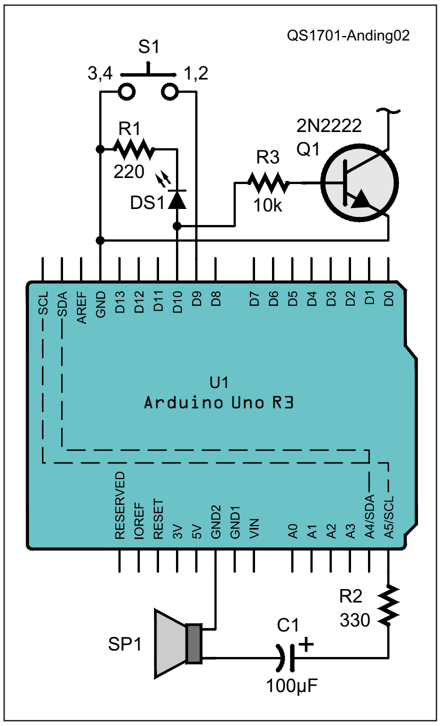

Figure 2 — The CW IDer uses the Arduino Uno R3 and a handful of parts.

C1 — Capacitor, 100 µF, 25 V electrolytic (Digi-Key Part 65-1307-ND)
DS1 — 5mm Red LED (Digi-Key Part C503B-RAN-CY0B0AA1-ND)
R1 — Resistor, 220 Ω ¼ W
R2 — Resistor, 330 Ω ¼ W
R3 — Resistor, 10 kΩ ¼ W
S1 — Momentary contact switch (N/O) (Digi-Key Part Number 450-1650-ND).

starts the timer, which runs for a specific number of seconds determined by the sketch. The timer range is 0 to 32,000 seconds. When the time expires, the software turns on the PTT LED, and a half-second later plays the CW message. When the message is complete, the Arduino waits a half-second and then turns off the PTT LED, which completes the IDer message cycle.

Encoding Morse Characters

The heart of the IDer is the Arduino sketch with the CW message stored in the Arduino memory. It can then be reliably decoded and played back. I chose to encode each CW character with a byte of data, using zeros to represent dits and ones to represent dahs. This way, each CW character is represented by a number between 0 and 255. Because CW characters are only 6 bits long, I added an extra bit at the end to be used as a stop bit. As an example, the CW character "A" equals a decimal 6, or a HEX 0x06, or the binary number 110. We'll use the binary number to decode the CW character. Let's use the binary "110" example above. The sketch encounters, from right to left, the zero first and plays a dit. The second element is a 1, so it plays a dah, and now we are left with a single 1, which is our end-of-character signal. From this simple start, we can build an alphabet, and the numbers 0 through 9, and add some prosigns to complete our set of CW characters.

Serial IDer Sketch

The Simple IDer sketch, explained in the sidebar, "A Simple IDer Sketch," has a user interface that will query the user for a message, a timer length, and words per minute (WPM) speed. These parameters are all stored in the Arduino EEPROM memory. The Serial IDer sketch converts the typed message into code for the IDer and adds the end-of-message (EOM) character automatically. So all we must do is type the letters, numbers, and spaces of the message.

To use the user interface, we load the Serial IDer sketch into the Arduino board. First, open an Arduino IDE window and go to FILE, then select Serial IDer sketch. It's a good idea, at this point, to save a backup sketch by clicking SAVE and giving the sketch a new name. Next, go to the top of the Arduino IDE window and click the VERIFY/COMPILER button. After the compiler does its work, we should see a "done compiling" message at the bottom of the IDE window. We have reached the halfway point.

The heart of the IDer is the Arduino sketch with the CW message stored in the Arduino memory.

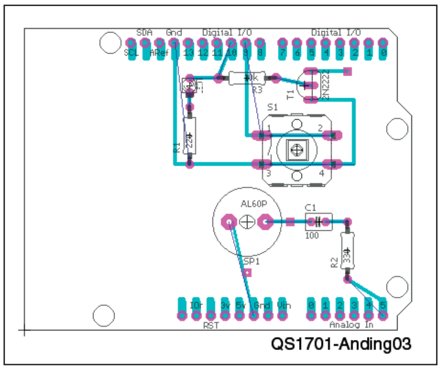

Figure 3 — A printed circuit board for the CW IDer.

The IDer sketch is compiled in the computer and is waiting to be loaded into the Arduino board. Next, click the UPLOAD button (right-arrow button), which will upload the sketch to the Arduino board. After pressing the upload button, the Arduino LEDs blink, and the sketch loads into memory. When all is done, we should get a completion message at the bottom of the IDE window showing percentage of memory used. Once the sketch is loaded into the Arduino board, we open the "Serial Monitor" in the Arduino IDE. The Serial Monitor is found under the TOOLS menu at the top of the IDE window. Click on TOOLS, then click on SERIAL MONITOR to open a second blank Serial Monitor window.

To access the Serial IDer programming mode, press and hold the Arduino reset button and press the activation button simultaneously. First, release the reset button while still holding the activation button down. The LEDs on the Arduino will flash back and forth for a while. Once they have settled down, only the one green LED should be on. This green LED indicates we're in the program mode. A message will display at the top of the Serial Monitor window asking to input a message. Built into the Serial IDer is a set of codes used to look up the message CW values. Click on the input line at the top of the Serial Monitor window and

A Simple IDer Sketch

The basic IDer sketch can be used as a set-and-forget IDer. It will require editing of a few lines of code and includes a simple method to encode CW characters. We encode a byte of data with a CW character, using zeros to represent dits and the ones to represent dahs. All the Morse code characters can be represented by a number between 0 and 255. Once encoded, the CW characters can be played back without concern of a missing dit or adding an extra dah. A glance at the binary number used to encode the CW character will give a visual check of the Morse code character. The "A" encodes as the binary "110," read right to left, shows us a dit dah, followed by a stop bit. From this example, we can build up the alphabet, the set of numbers 0 to 9, and some prosigns to complete the set of CW characters, shown in Table A.

There are special encodings for end-of-message (EOM) and Word Space (SP) characters. CW uses just six bits plus a stop bit, so the sketch can use these as special control characters. The Arduino has features to calculate the EOM from a predefined array for us. The EOM character isn't needed in this example, but if we were to send multiple messages, we would need to know when one message ends and the next begins.

Download the *Arduino_Simple_CW_IDer.sch.ino* file to your computer. Open the Arduino IDE, and go to the top of the Arduino IDE window and click FILE. Then click OPEN, and browse to the simple IDer sketch and highlight it. Then click OPEN.

The sketch is small, but has functioning CW audio output, an activation switch, a timer, and a PTT LED. With this simple sketch, the message parameters are entered manually. The parameters are located at the top of the sketch. This is where we'll find CW message codes that need to be translated from letters to HEX numbers, a timer length added in seconds, a tone frequency in Hertz, and a CW WPM speed. This sketch performs the entire task of an Arduino IDer or a CW routine to add to your project.

Table A shows the HEX encoding for the letters, numbers, and prosign characters. In playback mode, the IDer will decode the numbers and play the dits and dahs of the message. At the top of the CW IDer sketch, a CW message is stored in an array using a single line of code.

```
char message []={0x06, 0x06, 0x20, 0x0f, 0x1d}; //message =
"AA5OY"
```

When entering a message in the simple CW IDer, look up in Table A the HEX code for each character of a message. For example, to send "AA5OY," we would look up and enter the HEX value of each character of our message inside the curly brackets, separated by a comma. There's no comma needed after the last number.

Three Lines of Code Affect the Playback

The first line of code adjusts the dit duration, which changes the WPM of the CW message. The dit duration range is 35 to 100, which corresponds to approximately 20 and 5 WPM, respectively. To calculate the WPM, enter a message using "paris" [Don't forget to include the word space, SP, after each "paris." — *Ed.*], say, 10 times. Then, using a stop watch, count the number of times "paris" is sent in 1 minute to arrive at the number of words per minute. To change the dit duration, change the value of INT DIT = (56 shown).

```
int dit = 56; // adjust WPM speed (Slow=100, Fast=35)
```

The second line of code adjusts the timer length over a range of 0 to 32,000 seconds. If we were to create a routine to convert seconds to minutes, we would lose the resolution that a second provides. Computers are somewhat lax at calculating time. They perform other tasks besides watching the clock. To get an exact time takes trial and error. In the line of code below, set the INT TIMERLNG = value (here set to 5 seconds) to change the timer length.

```
INT TIMERLNG = 5; // set length of timer in seconds
```

The third line of code adjusts the CW tone frequency INT FREQ = to a value between 0 and 32,000 Hz. I prefer a value between 500 and 900 Hz in the example below.

```
INT FREQ = 900; // CW tone frequency (500 – 900 Hz)
```

Verify and Run

Once edits are done, click the VERIFY/COMPILER button (CHECK button). After the compiler has done its work, there should be a "Done compiling" message at the bottom of the IDE window. The IDer sketch is now compiled, in the computer, and waiting to be loaded into the Arduino board. Click the UPLOAD button (right-arrow button), which uploads the sketch to the Arduino board. It may take a while to load the sketch into the Arduino memory. When all is done, a completion message should appear at the bottom of the IDE window showing percentage of memory used.

We now have a working Arduino CW IDer with a programmed message, WPM speed, tone frequency, and timer length. Press the ACTIVATION button to start the timer. The message is played at the end of the timer interval.

Table A
Encoding of the Morse Characters

0x06 = a	0x11 = b	0x15 = c	0x09 = d	0x02 = e
0x14 = f	0x0b = g	0x10 = h	0x04 = i	0x1e = j
0x0d = k	0x12 = l	0x07 = m	0x05 = n	0x0f = o
0x16 = p	0x1b = q	0x0a = r	0x08 = s	0x03 = t
0x0c = u	0x18 = v	0x0e = w	0x19 = x	0x1d = y
0x13 = z				
0x3e = 1	0x3c = 2	0x38 = 3	0x30 = 4	0x20 = 5
0x21 = 6	0x23 = 7	0x27 = 8	0x2f = 9	0x3f = 0
0x29 = /	0x4c = ?	0x00 = SP	0xff = EOM	0x6a = '.'
0x73 = ','	0x56 = '@'			
0x68 = '*' = $\overline{\text{SK}}$	0xd1 = '^' = $\overline{\text{BK}}$	0x2a = '+' = $\overline{\text{AR}}$	0x31 = '$' = $\overline{\text{BT}}$	

Looking Through the Sketch

Look through the main decoder function of sketch in the function void msgReader(). It has an outer loop to read each character and the inner loop to decode dits and dahs. The inner loop reads each character one bit at a time using the "if / else" statements to send dits and dahs, shifting right to left. When there's just a "one" left, the sketch ends the inner loop and checks for another character.

For the cost of an inexpensive Arduino board, some free software, and a little time editing a few lines of code, you have a simple, cheap IDer that you might use in the shack, for a piece of equipment, or for a repeater.

enter the letters/numbers of your message.

When your message is completed, click the SEND button at the end of the input line. This activates the IDer message conversion routine. The routine first shows the message entered, then converts and writes it to the Arduino EEPROM memory. Last, as a double-check, it lists the HEX numbers it has converted in the message.

From this simple start, we can build an alphabet, the numbers 0 through 9, and add some prosigns to complete our set of CW characters.

Next, the interface asks for a WPM speed. Click on the input line at the top of the Serial Monitor window. Enter a dit duration value between 35 and 100. Click the SEND button. A duration of 35 corresponds to approximately 20 WPM, and 100 corresponds to approximately 5 WPM. The interface will next ask for a tone frequency. Click on the input line at the top of the Serial Monitor window and enter a number from 500 to 900 Hz. Click the SEND button.

Last, the interface will ask for timer duration. Click on the input line at the top of the Serial Monitor window, and enter a number from 0 to 32,000 seconds. After the last entry, the interface routine will display SETUP COMPLETED. The Arduino is now loaded and ready to play our message using the chosen parameters.

To activate the IDer, press the activation button, and the green LED on the board will begin to blink, counting down the timer. When the timer reaches zero, it activates the PTT LED, and a half-second later plays the CW message. At the end of the message, it turns off the PTT LED. The IDer now returns to monitoring the activation button.

A user interface makes for an easy setup of the Serial IDer and allows for quick updates of parameters in the field. The message length is not limited to a call sign, because the Arduino has space for hundreds of characters.

If you're interested in expanding this sketch, I would suggest breaking it up into multiple messages using several activation buttons. Setting up multiple activation buttons to play different messages as well as acknowledge beeps is beyond the scope of this article.

With your CW IDer message up and running, you are ready to go.

Notes

[1]www.arrl.org/qst-in-depth
[2]https://www.arduino.cc/
[3]https://www.arduino.cc/en/Main/Software

Bob Anding, AA5OY, was first licensed in 1987 as Technician class, with the call N5LJR, and upgraded in 1989 to Amateur Extra class, AA5OY. He is a member of ARRL and the Jefferson Amateur Radio Club, W5GAD. Bob designed, built, installed, and maintained two local repeaters. Over the years, Amateur Radio has allowed him to enjoy operating and pursuing his interests in designing, building, and programming. You can reach Bob at **aa5oy@arrl.net**.

For updates to this article, see the *QST* Feedback page at www.arrl.org/feedback.

VOTE
Did you enjoy this article? Cast your vote at www.arrl.org/cover-plaque-poll