

Arduino "Freq-Mite" for Norcal NC40A

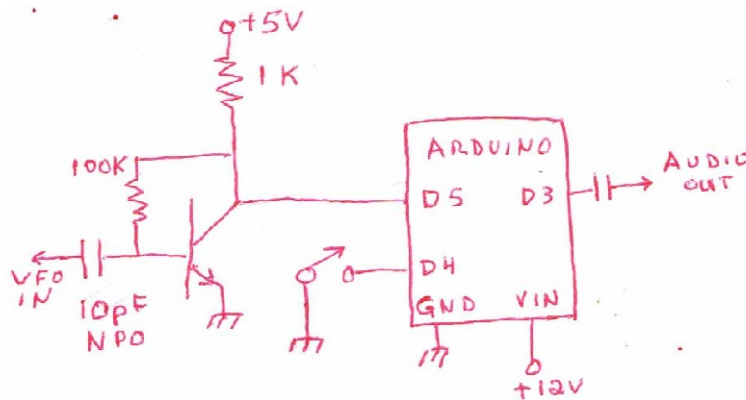
Mike WA8BXN

Jan 2018

Dave Benson's (K1SWL) Freq-Mite is a popular frequency counter used as a digital readout in CW of the operating frequency of QRP transceivers. No longer produced by Dave himself it is currently available from <http://www.4sqr.com/freq-mite.php>

I present here similar functionality using an Arduino processor intended primarily for use with the Norcal NC40A QRP transceiver. Within its limitations noted below it could be used otherwise. Besides being a useful device it should serve as an example of how the functionality might be implemented using an Arduino.

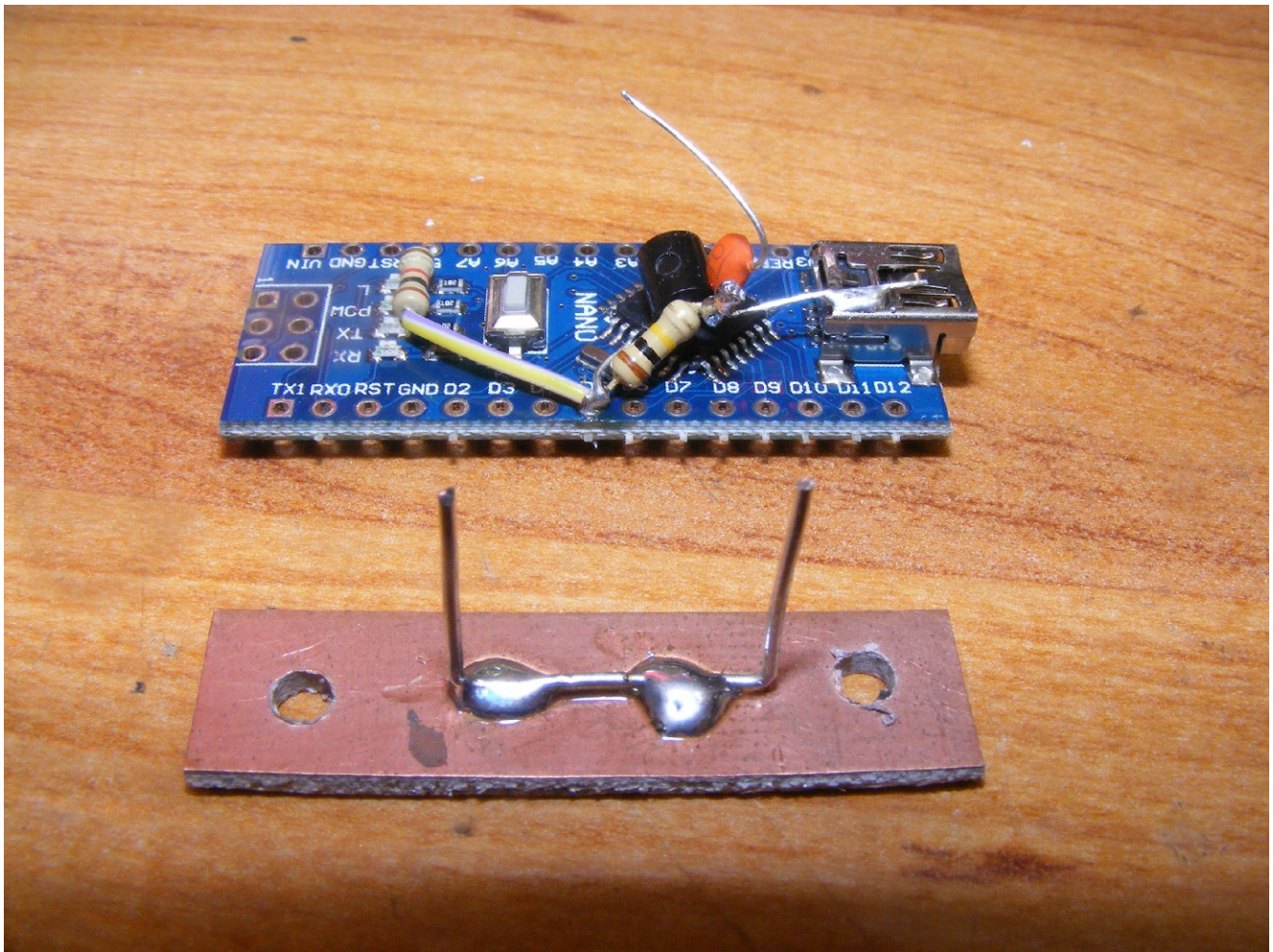
The hardware components are quite simple, shown in the schematic below:



The switch is a push button that when pressed announces the frequency in CW on the audio output. The rest of the circuit on the left is a simple preamp that allows light coupling to the VFO. I used a 2N3904 transistor, many others (such as a 2N2222) should work as well. I used an Arduino Nano processor board, an Arduino UNO board will work as well but is of course larger.

+5 volts is obtained from the Arduino board itself, which is powered by the supply to the NC40A which I indicate as +12 V. VFO input is obtained at the junction of R23 and C7 on the NC40A board and audio output is connected to the R7 and U3 pin 2 junction. For the capacitor shown above connected to D3 of the Arduino I used 100 pF. This value can be adjusted to give desired audio level. The 10 pF capacitor going to the VFO from the transistor could also be changed as needed. Keep it as small as possible to limit its effect on the VFO itself. It should be a NP0 so as to not introduce frequency drift in the VFO.

If desired a piezo buzzer could be connected directly to D3 instead of feeding audio into the NC40A for the CW frequency announcement.



Many Arduino Nano boards don't provide much in the way of mounting the board itself. I used a scrap of circuit board and a U shaped piece of wire soldered to it for mounting. The ends of the wire were soldered to the two ground pins on the Nano board. Other components were soldered to the Nano board with the shell of the USB connector used as a ground point for the emitter of the transistor. Other construction methods could be used as well.

The original Freq-Mite had jumpers to select the IF offset frequency. Since the source code for the program is provided here that offset value is simply specified in one of the lines of code. Similarly adjustment of the CW speed is possible by changing a line of code instead of pressing the button at power up.

Before looking at the program below some general background information is necessary for its understanding. Basically the program functions as a frequency counter that measures the frequency of the VFO in the NC40A, adds the IF offset frequency (around 4.915 MHz) and then generates CW for the operating frequency. When the button is pressed this happens and the frequency in KHz is heard.

To measure frequency one could watch the input signal in the program and count the number of cycles that occur in a given period of time. Testing the input to see when each cycle of input begins in code would greatly limit the frequency limit of what we could count. This would probably limit us to audio frequencies not the around 2 MHz at which the VFO operates.

The trick used that allows us to count frequencies in the MHz range makes use of the fact that most of the processor chips are designed with a counter in the chip that can be used to count cycles in hardware without executing program instructions to recognize each cycle and do the counting. Access to this is not apparent in the normal description of programming in the Arduino environment but does exist and is documented somewhere.

The hardware counter in the processor on the Arduino board is only 16 bits meaning it can only count up to a value of 65535. Beyond that value it just start over counting at zero again. Fortunately an interrupt is generated when the counter overflows and we can in our program count the number of times that occurs. From this we can compute larger counts. If you don't fully understand this detail its OK.

Now our basic approach to frequency counting will be to use that hardware counter to count input cycles for us for some period of time. If we count for one second the resulting count would be the frequency in Hz. We don't need that much accuracy nor do we want to wait a whole second before hearing the frequency so the program only counts for 10 mS. The resulting value we will get will be in terms of hundreds of Hz. By default the program will only report the frequency in KHz. Un-commenting a few lines at the end of the program will allow it to report the frequency including tenths of KHz.

Accuracy of the frequency we get from doing the cycle counting will depend on how accurately we can determine the period of time we do the counting. The Arduino processor board does have a crystal controlled timing oscillator running at 16 MHz. However there is no trimmer to adjust to get it exactly on frequency. As a result our timing should be fairly stable but not absolutely accurate. Provision is made in the program to compensate for that.

Another issue is the value for the IF frequency to add to our frequency count. The crystals used in the filter are nominally 4.915 MHz but where the center of the passband we get is a bit different. Another variable to this is the adjustment of trimmers C34 and C17. A constant in the program can be adjusted for these issues.

The program below can be used as is most likely. Comments about modifications are included following the listing. Note that if you do use a Nano processor and have never done

so before you are likely going to need to install drivers found at <http://bit.ly/2pMF4in>
If you use an Arduino UNO board needed drivers are included with installation of the Arduino programming environment (IDE).

Installation of the Arduino IDE, programming in C and related topics are of course beyond the scope of this discussion but well covered elsewhere.

```

/*
Arduino frequency counter for Norcal NC-40a with CW output
Can be used with VFOs up to around 5 MHz
Runs on Nano or UNO processor boards

Mike WA8BXN Jan 2018
*/

// Port definitions
#define FREQ_IN 5           // Do NOT change this
#define BUTTON 4
#define BUZZER 3

#define SPEED 15
#define ditTIME 1200/SPEED
#define IF_OFFSET 49155
#define CPU_FREQ_ADJ 9951

volatile unsigned int overflowCounter;
unsigned int frequency;

char *code[10]=
  {"-----", ".----", "..---", "...--", "....-",
   ".....", "-....", "--...", "---..", "----."};

// Timer 1 is our counter
// 16-bit counter overflows after 65536 counts
// overflowCounter will keep track of how many times we overflow
ISR(TIMER1_OVF_vect) // Interrupt handler
{
overflowCounter++;
}

void setup()
{
// Timer 1 will be setup as a counter
// Maximum frequency is Fclk_io/2
// (recommended to be < Fclk_io/2.5)
// Fclk_io is 16MHz
TCCR1A = 0;
// External clock source on D5, trigger on rising edge:
TCCR1B = (1<<CS12) | (1<<CS11) | (1<<CS10);
// Enable overflow interrupt
// Will jump into ISR(TIMER1_OVF_vect) when overflowed:
TIMSK1 = (1<<TOIE1);

pinMode(FREQ_IN, INPUT); // This is the frequency input
pinMode(BUTTON, INPUT_PULLUP);
pinMode(BUZZER, OUTPUT);
Serial.begin(9600); // Console debug output
Serial.println("Begin AFA");
OK(); // Startup msg in CW
}

void loop()
{
if (digitalRead(BUTTON)==LOW)
freqCount();
}

```

```

void freqCount ()
{
// Delay 10 mS. While we're delaying Counter 1 is still
// reading the input on D5, and also keeping track of how
// many times it's overflowed
char buf[20];
TCNT1=0;
overflowCounter = 0;
delayMicroseconds (CPU_FREQ_ADJ);
frequency = TCNT1+65536* (unsigned long) overflowCounter; // freq in 100s of Hz
sprintf (buf, "VFO %5d.%d kHz", frequency/10, frequency%10);
Serial.println (buf); // Console debug output
sendFreq ();
}

void dit ()
{
tone (BUZZER, 600);
delay (ditTIME);
noTone (BUZZER);
delay (ditTIME);
}

void dah ()
{
tone (BUZZER, 600);
delay (3*ditTIME);
noTone (BUZZER);
delay (ditTIME);
}

void OK () // send power up "OK"
{
dah (); dah (); dah (); delay (3*ditTIME); dah (); dit (); dah (); delay (3*ditTIME);
}

void sendDigit (int n)
{
char *p=code[n]; // string for this digit
while (*p) // for each character in string
if (*p++=='.') dit (); else dah (); // send dit or dah
delay (2*ditTIME); // pause between digits
}

void sendFreq () // send frequency in CW to audio
{
long f;
int n=0, tenths;
int digits[10];

f=frequency+IF_OFFSET;
Serial.print ("Mixed freq "); Serial.println (f); // Debug output
tenths=f%10;
f=f/10;
while (f) // get individual digits right to left
{
digits [n++]=f%10;
}
}

```

```
        f=f/10;
    }
n--;
while (n>=0)                // send digits left to right
    {
    sendDigit( digits[n--]);
    }
/*                          uncomment for tenths of KHz
dit();dah();dit();delay(3*ditTIME); // send letter R for decimal place
sendDigit(tenths);
*/
}
```

For those that want to fine tune the program, there are 3 lines of code that are easy to change. They are found near the beginning of the listing:

```
#define SPEED          15
#define IF_OFFSET      49155
#define CPU_FREQ_ADJ   9951
```

The value for speed is the simplest. Change 15 to the CW speed you prefer, in WPM.

The other two require experimentation to determine the best values. The following procedure should be used.

The value CPU_FREQ_ADJ allows us to compensate for variations in the 16 MHz CPU clock frequency. Connect a known frequency input to the circuit (1 MHz would be a good value). Start the Arduino IDE and connect the processor board to the computer with a USB cable to get set up for programming the board. Load the source program into the IDE if its not already there.

In the programming IDE, open the debugging monitor window. Press the button to read frequency. In the debug monitor window you will see something like this:

```
Begin AFA
VFO 2123.7 kHz
Mixed freq 70392
```

I just used the VFO rather than a 1 MHz input. With 1 MHz you want to get it to show VFO 1000.0 kHz or what ever your known frequency input actually is. It must be less than around 5 MHz!

Change the CPU_FREQ_ADJ value and recompile and upload the program and run again until you get as close as you can to the correct display.

To adjust the number for IF_OFFSET connect the input to the VFO as would be used for normal operation. Tune the VFO to a know frequency using some independent standard. You could measure the transmit frequency of the NC40A or tune the receiver to hear a signal at a know frequency. Press the button and note the value displayed in the debug monitor window for Mixed freq. Change the number for IF_OFFSET to get it to display the right Mixed freq when the button is pushed.

Concluding remarks

I hope you find this project description useful. I think its fairly easy to actually build and get working, particularly if you have any Arduino experience already. You may have noticed that most of the time the program just sits waiting for that button to be pressed. I heard it begging for more to do and with a few more hardware parts and some more lines of code added a simple iambic keyer with a stored message as well but that is a story for next time!