Elwood Downey, WBØOEW

8274 North Sunset Ranch Loop, Tucson, AZ 85743; **wb0oew@arrl.net**

Simple and Accurate Variable Frequency RF Signal Generator

This Generator uses an Arduino-controlled direct digital synthesizer stabilized with a GPS receiver, along with a hand full of parts.

This generator produces any frequency between 500 kHz and 40 MHz with an accuracy approaching one part in 10^9, for example 0.01 Hz at 10 MHz. It uses an Arduino Nano, a GPS receiver with antenna, a digital encoder, a small TFT LCD color display, and the Silicon Labs Si5351A direct digital synthesizer (DDS). Together these parts cost me about $150. I will also share some interesting applications for this device.

Operation

My housing and user interface are shown in Figure 1. The user's desired frequency appears across the top of the display. Tapping the screen will underline different groups of three digits. Rotating the encoder will change this group, with faster rotation changing the numbers even more rapidly. The next row reports the error measured during the previous 10 second measurement scaled to the

Figure 1 — Packaging and user interface.

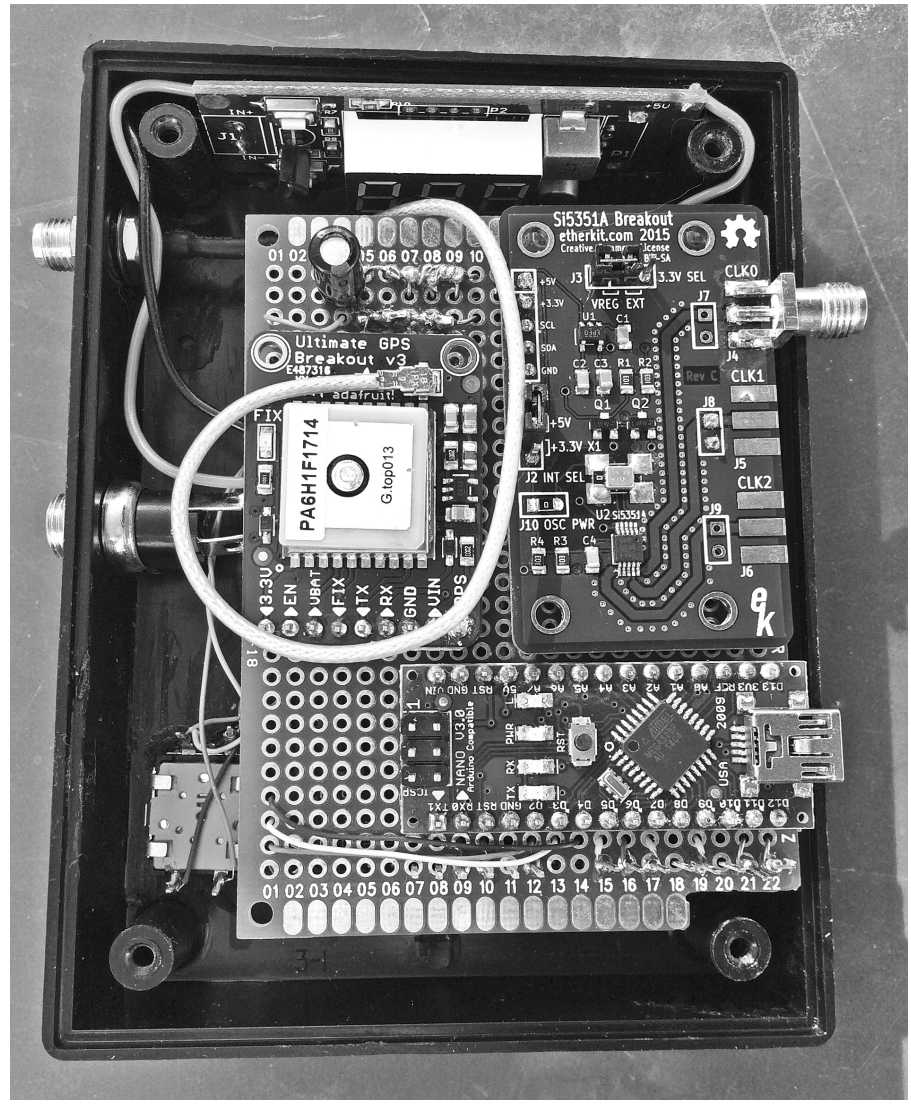

Figure 2 — An inside view shows the method of construction.

current user frequency and includes quantization errors from loading the integral DDS registers. This row can also report if the GPS PPS (pulse per second) signal is not available or the reference clock is not working. The next row reports the approximate output power in dBm into 50 Ω. Clicking the encoder cycles through four power levels. The remainder of the display shows the current date and time, antenna location, an all-sky map showing the tracks of the satellites currently being used for the fix, the software version, and the best and worst satellite signal-to-noise ratio. These extra data are just for fun and are not used by the generator control algorithm. Note that displaying the time is given low priority and can be as much as one second late.

Construction

Figure 2 shows an inside view of my unit showing my method of construction. The modules are placed on one piece of perf board and wiring is point-to-point underneath. The display is stacked below in this view. The wiring diagram is shown in Figure 3. The level shifting diodes ensure the 5 V Arduino is reliably triggered from the 3.3 V outputs from the GPS PPS and Silicon Labs DDS. I used the Arduino Nano but any Arduino should work with suitable attention to pin numbers. The unit can be powered with USB but I wanted to use my shack 13.8 V dc supply, so I added a USB buck converter[1] from eBay to create 5 V. For the Si5351A I tried both the Etherkit[2] breakout with the temperature controlled crystal oscillator (TCXO) option and the Adafruit board[3], which has no temperature control. Exposed on the bench, the unit with oven control was significantly more stable, but inside the enclosure I could measure no difference, so the final choice just depends on your packaging preference. See Table 1 for a list of the main part numbers and current prices in US$, not including wires, connectors and enclosure.

I did not use any third-party software libraries for the Si5351A, preferring to write code for the "bare metal" to allow me to develop a deeper understanding of how best to use this synthesizer. Similarly, I did not use the Adafruit GPS library but only because I needed just a few features so I could save precious memory by writing my own code. I did use the Adafruit GFX, ILI9341 and FT6206 libraries for their TFT display and touch overlay panel. My source code is available on the **www.arrl.org/qexfiles** web page. You may also check for updates for this project at **www.clearskyinstitute.com/ham/gps**.

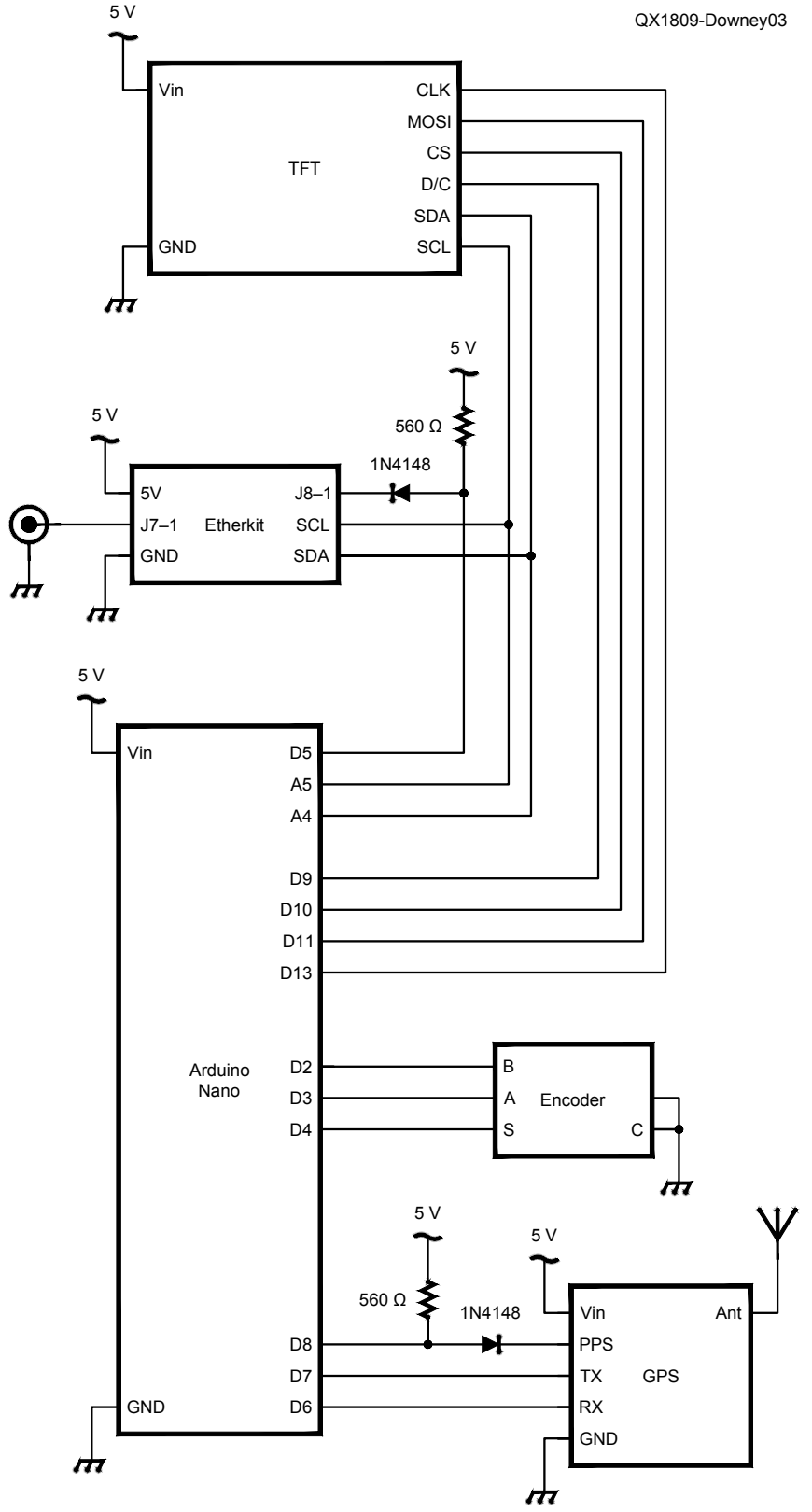

Figure 3 — Wiring diagram of the generator.

Table 1
Parts list not including enclosure and various wiring materials

Part	*Cost*
Adafruit Ultimate GPS breakout – PID 746	$40
Adafruit GPS Antenna External Active – PID 960	$13
Adafruit uFL to SMA adaptor cable – PID 851	$4
Adafruit 2.8" TFT LCD with capacitive touch – PID 2090	$40
Adafruit Rotary encoder – PID 377	$4.50
Either Adafruit Si5351A breakout – PID 2045	$8
Or Etherkit Si5351A breakout with TXCO	$16
Arduino Nano – several sources	$22
1N4148 diode , 2,	$0.50
560 Ω ¼ W resistor, 2,	$0.50

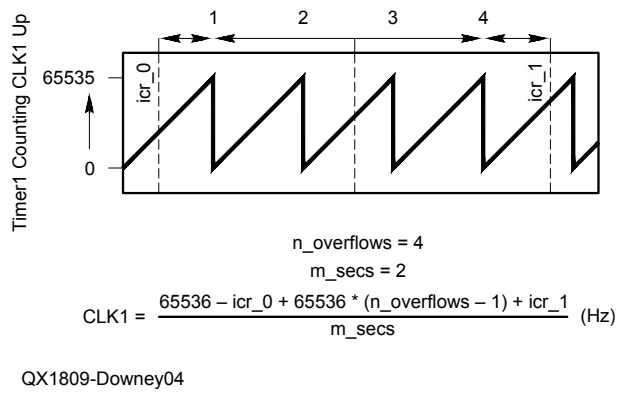

Figure 4 — An example measurement sequence.

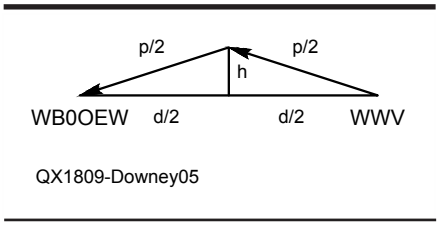

Figure 5 — Idealized path geometry between WBØOEW and WWV.

Theory of Operation

If the GPS receiver can receive from at least four satellites it produces one digital pulse per second (PPS). Clock 1 of the Si5351A is programmed to produce a frequency of 5,000,000.00 Hz. This is fed to an Arduino counter[4] whose value is captured at each PPS signal. These measured counts are used to correct the Clock 1 frequency. The claimed accuracy of the PPS is one part in 10^8 with random jitter. This could be improved another factor of 10 by applying the error measured over 100 seconds, but I did not want to assume the DDS would be perfectly stable for this long, so instead I measure the error over a period of 10 seconds and apply one tenth of this as the correction.

The software continuously compensates for the DDS crystal oscillator error by measuring the value of a known reference frequency, REF_FREQ, being generated with CLK1. It then uses this correction to produce any frequency on CLK0 to the same relative accuracy.

Figure 4 shows an example measurement sequence. The DDS CLK1 is set to generate the REF_FREQ. This is connected to D5 (T1) so it causes Timer1 to count continuously from 0 to 65535 then overflow and repeat. Each overflow triggers an interrupt that increments n_overflows. Meanwhile, the GPS 1 PPS is connected to D8 (ICP1). Each falling edge copies Timer1 (TCNT1) to the Input Capture Register (ICR1) and triggers an interrupt. The first PPS interrupt after a measurement is started saves ICR1 in icr_0. After m_secs more PPS interrupts the ICR1 is saved in icr_1. Thus the total counts during m_secs is the remaining Timer1 counts until the first overflow, plus the number of counts in the next full n_overflows periods, plus the counts until the final PPS.

The Si5351A uses several fixed point configuration registers of the form a + b/c. Care is taken in the software to choose the best combination of these integral values to achieve a total precision of at least 30 bits or about one part in 10^9 to match the effective GPS precision. The desired user frequency is generated on Clock 0 from the same crystal time base so it is also accurate to the same precision, scaled proportional to frequency. The reference corrections run continuously in the background so the user is free to adjust the desired output frequency with the digital encoder on Clock 0 at any time. Note well that this DDS generates square waves, not sine waves. This is ideal for the control purpose on Clock 1, but the Clock 0 user output will be rich in odd harmonics.

Entering the ARRL Frequency Measurement Test

I tested the project by using it for the ARRL Frequency Measuring Test held in April 2018. This test required measurement of frequencies in the 20, 40 and 80 m bands. I tied for second place in a field of 108 with a maximum error of 0.09 Hz. I used an Elecraft KX3 with SpectrumLab[5] running under *wine* on an iMac with macOS 10.13.3. This is an exceptionally versatile and accurate audio measurement tool built for Windows, but I had no problems running it on macOS with *wine* and I have no reason to doubt it would run equally well under *linux*. The File Export Format dialog contained the following three-column definitions; note the third column records the time in my local time zone of UTC-7 hours:

```
# WWV peak_f(990,1010)###0.00#
# GPS peak_f(965,975) ###0.00#
# UNIX time-25200#########0.0
```

I did not make a direct connection between the generator and the receiver, I just used a 10-inch bare wire lying on the desk top near the receiver with one end attached to the Clock 0 output. I set the receiver to USB and tuned so I could hear the FMT signal at about 500 Hz audio tone. I then adjusted the generator and the wire to produce a second tone at about 450 Hz with about equal intensity. I then recorded the audio with both frequencies. Later I measured each audio tone peak frequency with an FFT bin size of 21 mHz. The frequency of the carrier was then computed as the difference between the audio tones added to my generator frequency. To find each FMT frequency I applied the formula,

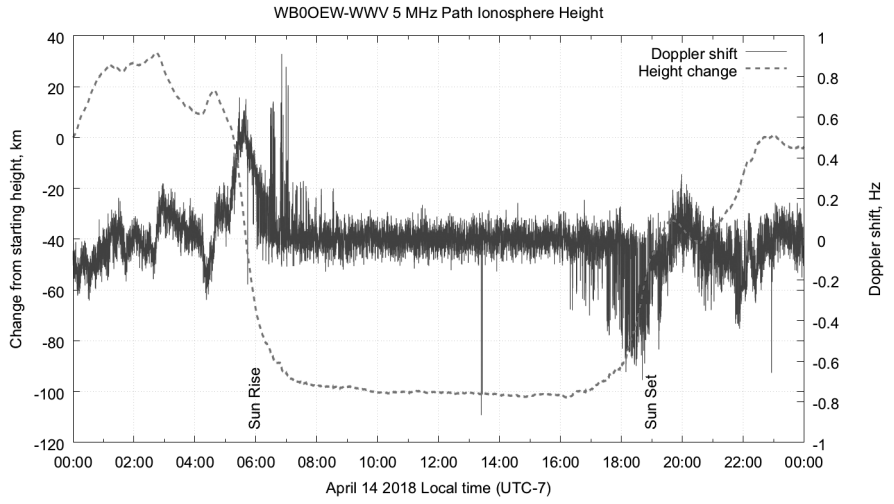

Figure 6 — Doppler shift and ionospheric height change over a 24–hour period.

$$f_{FMT} = f_{GPS} + (A_{FMT} - A_{GPS})$$

where f_{FMT} is the unknown FMT frequency, f_{GPS} is the frequency displayed on the GPS generator, A_{GPS} is the audio frequency of the GPS generator reported by SpectrumLab peak_f(), and A_{FMT} is the audio frequency of the FMT signal, reported by SpectrumLab peak_f()

Note that by using the difference in the measured audio tones this technique relies entirely on the GPS generator for accuracy. The Arduino, receiver and computer time bases must be stable during the test but need not be well calibrated.

Computing Change in Ionospheric Height by Measuring Doppler Shift

I tried measuring the Doppler shift of WWV at 5 MHz. This allowed me to calculate the change in path length from their transmitter in Colorado to my station in Arizona. The distance is about 1000 km so I assumed the path required one ionospheric reflection and no ground reflections. Using this model I estimated the effective height of the ionosphere over time as follows.

The velocity, $v(t)$, at which the source of an electromagnetic wave is moving with respect to the receiver can be computed from its Doppler shift as a function of time[6],

$$v(t) = c\frac{\Delta f(t)}{f_0} \tag{1}$$

In my case, the source is not moving but the path length is changing because the effective reflection height is changed by insolation. For example, if the frequency is increasing, it means the path length is getting shorter because the ionization layer is getting thicker and the effective reflection height is getting lower. Integrating velocity from time 0 to time T lets us find the total path length change, Δp, in this interval as,

$$\begin{aligned}\Delta p(T) &= \frac{c}{f_0}\int_0^T \Delta f(t)dt \\ &\approx \Delta t\frac{c}{f_0}\sum_{i=1}^{T/\Delta t}\Delta f_i\end{aligned} \tag{2}$$

This integration can be performed numerically by maintaining a running sum of each measured frequency value with respect to the known reference frequency every Δt seconds. To convert this path length change to a change in height, I assumed the path simply follows each hypotenuse of two right triangles sharing a common vertical side beneath the point of reflection as shown in Figure 5. From this geometry we have,

$$\left(\frac{p}{2}\right)^2 = h^2 + \left(\frac{d}{2}\right)^2 \tag{3}$$

then from $\partial h/\partial p$ we find the change in height for a given change in path length is approximately

$$\Delta h = \frac{p}{4h}\Delta p \tag{4}$$

Combining Eqns. (2) and (4) we find the height change as a function of time,

$$\Delta h(T) = \frac{\Delta t p c}{4 h f_0}\sum_{i=1}^{T/\Delta t}\Delta f_i \tag{5}$$

I set h to the F-layer height[7] of 300 km, and d to1000 km for which p is1170 km. The measurements were taken every $\Delta t = 5$ s, and $f_0 = 5$ MHz. This yields a scale factor of 292 m s per sample.

Putting it all together we have one result over a 24-hour period shown in Figure 6. This was measured using an active receiving loop antenna[8] connected to an RFSpace Cloud-IQ SDR receiver (see: **rfspace.com/RFSPACE/CloudIQ.html**) read with gqrx (see: **gqrx.com**). and piped into SpectrumLab running under *wine* all on macOS 10.13.3. The radio was set to about 4999 kHz USB so the WWV carrier produced an approximately 1 kHz audio tone. The GPS reference was set to 4,999,970 Hz to produce an approximately 970 Hz audio tone. It is only the difference in tone frequencies that matters. SpectrumLab was programmed to find the peak of each of these two frequencies and store the difference. These were summed with *perl* (see: **https://www.perl.org**) and plotted with *gnuplot* (see: **www.gnuplot.info**). The SpectrumLab output file was named wwv-5-doppler.txt with three columns for the measured WWV and GPS reference audio frequencies and the local time in UNIX format. The key gnuplot commands include:

```
gnuplot> set xdata time
gnuplot> set timefmt "%s"
gnuplot> set format x "%H:%M"
gnuplot> set y2tics
gnuplot "< perl -n -a -e
   '$df=$F[0]-$F[1]-30;
   $h-=292*$df/1000;
print \"$F[2] $h $df\n\";'
   wwv-5-doppler.txt" using
   1:3 axis x1y2
title 'Doppler shift', ""
   using 1:2 title 'Height
   change' linewidth 3
```

The height scale is only approximate but you can see some interesting trends on this day. The plot begins at local midnight. The height increased a bit more until sustaining a fairly steady maximum for much of the remainder of the night. After an interesting bump an hour before sunrise it dropped quickly then leveled off during the daytime hours. As sunset approached the height rose again quickly. The bounce back an hour later looks suspiciously like the mirror image of the bump before sunrise. It looks as if the remainder of the night may not be as high as the previous night. These plots are easy to make and fun to ponder. I have recorded several days on different bands in this manner and all contain varied and interesting characteristics.

I hope you enjoy using this variable frequency generator. Feel free to contact me if you have any questions.

Elwood Downey, WBØOEW, has been licensed continuously with the same call sign since 1974. He enjoys building telescope control systems and related astronomical instrumentation. Elwood maintains a small Amateur Radio web page at ***www.clearskyinstitute.com/ham****.*

Notes

[1]**https://www.ebay.com/itm/DC-6-5-40V-To-5V-2A-USB-Charger-DC-DC-Step-down-Buck-Converter-Voltmeter-Module/321787604751?hash=item4aec092f0f:g:0ClAAOxygPtS4cu5**.

[2]Etherkit Si5351 breakout board with TXCO, **https://www.etherkit.com**.

[3]Adafruit products, tutorials and software are available at **https://www.adafruit.com**.

[4] For details on programming the counters see Chapter 20 in the Atmel data sheet at **www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf.**

[5]SpectrumLab is available as a free download at **www.qsl.net/dl4yhf/spectra1.html**.

[6]**https://en.wikipedia.org/wiki/Doppler_effect.**

[7]This is a typical height for the F layer responsible for HF propagation; see **https://en.wikipedia.org/wiki/Ionosphere**.

[8]I used this amplifier: **active-antenna.eu/amplifier-kit**, with two coplanar loops 1 m in diameter with a center height of about 3 m.