GPS Locked Combination Frequency Counter and Frequency Standard Version 3.0

John Price – WA2FZW

Table of Contents

Introduction

This project is an offshoot of the digital VFO project that I developed along with Glenn (VK3PE) and Jim (G3ZQC).

One of the requirements for that project was that one needs to calibrate the frequency of the Adafruit Si5351 module. In order to do that I needed a frequency counter. Having been off the air for some 20 years, I no longer had much in the way of test equipment, so I bought a cheap Chinese frequency counter.

Much to my dismay, that counter didn't display the unit's digit; i.e. it only measures to the nearest 10Hz at RF frequencies! But by doing a lot of painstaking adjustment of the Si5351 calibration factor and interpolating between the values where the ten's digit changed, I was able to get the Si5351 to be used in my VFO pretty close.

But that was a real pain, and since I was testing with several different Si5351 modules, going through that process each time was very time consuming.

Plan "B"! My partners on the VFO project (Glenn & Jim) had both purchased another counter with better resolution on eBay; a BG7TBL FA-2 (shop around; you can probably find it cheaper). But, when I tried to use it to calibrate a Si5351 that I had already calibrated using the long method, the calibration factor was way different than what I had come up with earlier. Since that particular Si5351 seemed to be pretty close to being on frequency in the radio, this seemed to indicate that the new counter's frequency was way off! But note, once I had this counter working, I determined that the FA-2 was indeed *VERY* accurate.

Plan "C"! Jim told me about a very simple counter developed by Scott Harden. This one can be locked to a GPS *1PPS* (One Pulse per Second) signal, which should insure good accuracy.

What the heck! One can't have too many counters so I decided to build this one but with an enhancement. Since the FA-2 counter has a provision to clock it using an external 10MHz source (which seems to be common to a lot of counters), I decided to build the new one such that it could be used as either a stand-alone counter or as a 10MHz frequency standard.

Plan "D"! After I had a few PCBs made for the original version which used the Arduino Mega2560 Mini processor (had some left over from a previous project) I discovered they were becoming hard to find so I redesigned the PCBs to use an Arduino Nano.

Using the physically smaller processor also allowed room on the PCB (which is designed to stack on top of the QRP Labs QLG1 GPS module) allowed me to put the 5V regulator on the PCB. I also added a PCF8574 GPIO expander to read the counter chip, which gave me a few more I/O pins on the Nano.

Here's what my finished unit looks like. The cabinet is a Bud SC-12100 (shop around you might find it a bit cheaper):

The optional gate LED option was added after I built this one, so it's not shown here.

Modification History

In Version 2.9 of the software (no hardware changes) I partially fixed a problem that was causing the time to be a full second behind the actual time. Unfortunately I was not able to fix it completely as the real problem is in the GPS module itself. The 1PPS signal is generated by the GPS unit exactly when the second changes, but the GPS does not send the updated time message to the processor until after the 1PPS signal goes low which is about 0.1 seconds after the pulse starts. Thus, the time displayed is now about 100 milliseconds behind the actual time.

In the Version 3.0 software (again, no hardware changes) I added code to make the gate LED actually work. Somehow I never implemented it in the original version. I also cleaned up some unused code.

Hardware Architecture

Normally one would use a TCXO (Temperature Controlled Crystal Oscillator) or an OCXO (Oven controlled Crystal Oscillator) as a frequency standard, but those can be difficult to adjust using software.

I decided to try a different approach and use a Si5351 to generate the 10MHz signal. I had already written a program to calibrate a Si5351 so I figured I could use that approach to adjust the frequency on the fly should it drift or jitter a bit (which tests by Glenn & Jim indicated that they do).

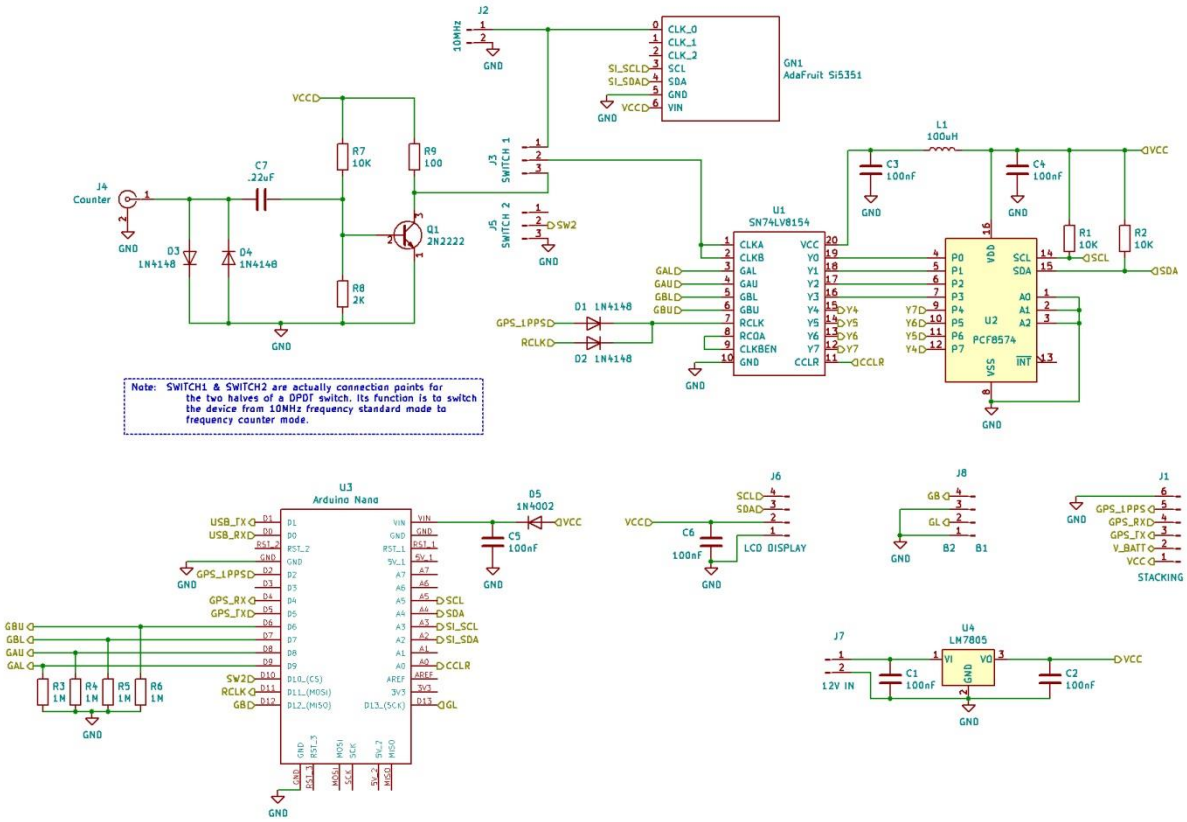
The device is designed to work with the QRP Labs QLG1 GPS Receiver Kit and my PCB is designed to mate up with that GPS and stack on top of it (or under it).

Only a few major components are needed for the project:

- The QRP Labs QLG1 GPS kit
- An Adafruit (or equivalent) Si5351 module
- An Arduino Nano processor (or a clone)
- An SN74LV8154 counter chip
- A PCF8574 GPIO expander chip
- 4 line by 20 character I2C LCD display

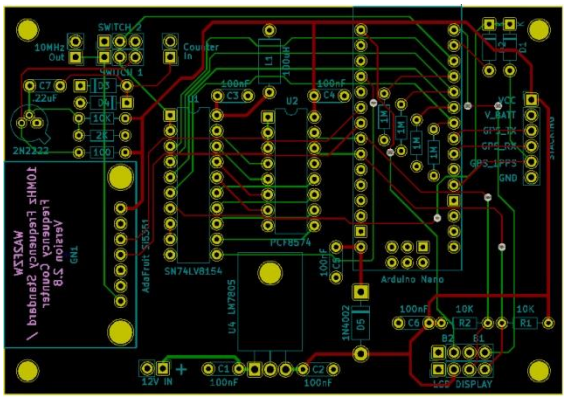
Since I designed this, I notice that QRP Labs now has an Si5351 module enclosed in an oven, which could provide more stability. Note however that my PCB is not designed to make use of this device. The web page for it also describes some modifications to improve its stability adjustments.

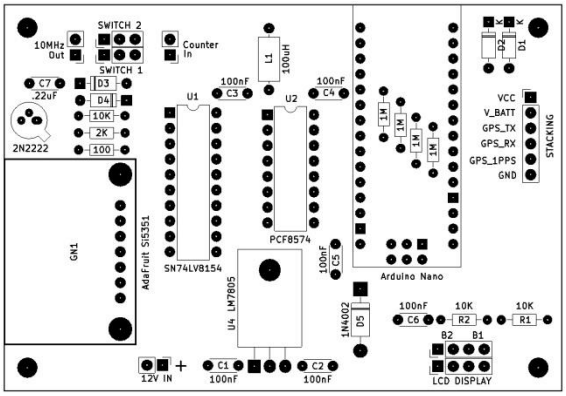
Here's the schematic of the PCB itself:



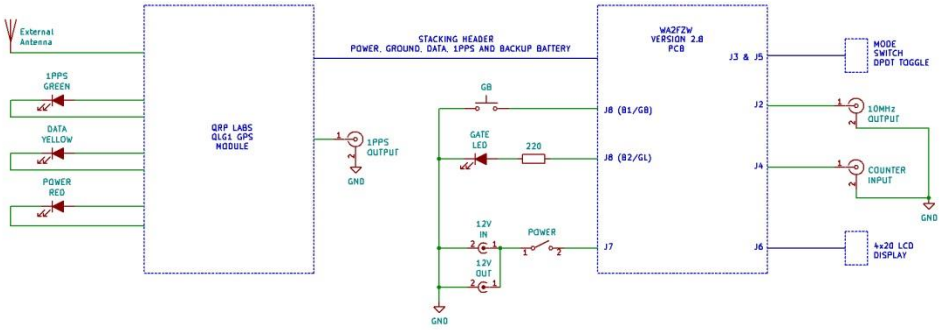
Note, the *GB* (Gate Button) and *GL* (Gate LED) labels on the schematic correspond to the *B1* and *B2* labels on the PCB. If I do new PCBs, this will be corrected.

And here are a couple of pictures of the PCB layout:





Here's a schematic showing the external connections. It might be a bit confusing due to the limitations of the CAD tool I use, but you should be able to figure it out:



If I produce new PCBs, I'll put the 220 ohm resistor for the *GATE LED* on the PCB.

Theory of Operation

The SN74LV8154 Counter

It's pretty simple. The key component is the SN74LV8154 counter. The device has two 16 bit counters that can be chained together to make it a 32 bit counter (which I did here).

The frequency to be measured is fed into both the *CLKA* and *CLKB* inputs but counter "B" only registers a count when the *CLKBEN* (clock "B" enable) pin is low. The *CLKBEN* pin is connected to the *RCOA* pin which goes low when counter "A" is about to rollover back to zero.

The other neat feature of the chip is that when the *RCLK* (read clock) pin goes high, the current count is transferred into a set of registers and will remain there until the next *LOW* to *HIGH* transition of the *RCLK* pin. This gives the software plenty of time to read the counter. The *RCLK* pin is connected to the *1PPS* output of the GPS module.

The PCF8574(A) GPIO Expander

I added this in the Nano version as the Nano didn't quite have enough GPIO pins. The PCF8574 can provide up to 8 logical GPIO pins and is controlled via the I2C bus. It is also capable of generating interrupts, although that capability is not used here.

Either a PCF8574 or a PCF8574A can be used on the PCB. The only difference between the two versions is the I2C address. The PCF8574 address can be selected in the range of 0x28 to 0x2F and the PCF8574A address range is 0x38 to 0x3F. The PCB is set up to use the 0x28 or 0x38 addresses depending upon which version of the chip is in use.

The Mode Switch

The mode switch is a DPDT toggle switch (on the schematic, the two halves are shown as separate switches but they are not).

In the "*Counter*" position, the external input is directed to the 2N2222 preamplifier the output of which is then connected to the clock inputs of the SN74LV8154 and the *SW2* input (D4) to the Arduino is in a *LOW* state so the processor knows the operating mode.

In the "*Standard*" position, the *CLK0* output of the Si5351 is directed to the clock inputs of the SN74LV8154 and *SW2* input (D4) to the Arduino is in a *HIGH* condition (the Arduino has built-in pullup resistors).

The Display

I had originally intended to make it possible to use either an ILI9341 based TFT (thin film transistor) display or one of the commonly available four line by twenty character LCD (liquid crystal) displays. But, after testing with three different libraries for the TFT and three different displays, I was unable to get the TFT to work, thus the only option is now the 4x20 LCD.

The Processor

The processor is an Arduino Nano (or clone). The ones I have are made by Elegoo and came from Amazon.

There are 3 different processor options available in the Arduino IDE when you select the board type as "Arduino Nano". The ones I have require you to select "ATmega328P (Old bootloader)" in order to get the software to load correctly.

The Si5351

The Adafruit (or equivalent) Si5351 clock generator is used to generate the 10MHz frequency standard. It has 3 clock outputs available; we use *CLK0*.

The Si5351 communicates with the processor over an I2C bus. You will notice when you look at the GPIO pin assignments below, that the SPI bus for the Si5351 is on a different pair of pins from the I2C bus used for the PCF8574 and the display. The Si5351 handling software was hijacked from the VFO project and is a modified version of code used in the VFO originally created by T.J. Uebo (JF3HZB).

TJ's Si5351 code does not use the Arduino *Wire* library which most libraries for other I2C peripherals do, and as a result, if the Si5351 is put on the same bus with other I2C peripherals, it hangs the bus.

B1 (GB) and B2 (GL)

On the Version 2.3 PCB, I added the provision to optionally add two pushbutton (or SPST) switches that can be used for whatever one might want to use them for.

Later on, I changed my mind about the buttons. The connection on the PCB for *B1* (*GB* on the schematic) is for a momentary pushbutton that can be used to change the gate time.

After coming up with no practical use for a second button, I repurposed the *B2* connection on the PCB (*GL* on the schematic) to allow one to install a LED which will flash briefly when the gate time has expired. Note that in the picture of my unit above, this LED is not installed as I made the change after building it.

As noted below the *External Connections* schematic above, if I re-do the PCBs I'll change the labels on the board and add the 220 ohm resistor to the PCB.

Both the button and the LED are optional.

Preamplifier

The original version turned out to not be a sensitive as I would have liked, so in Version 2.8, I added a 2N2222 preamplifier between the counter input jack and the mode switch.

Diodes D3 and D4 limit the input voltage to the transistor to approximately +/- 0.7V. A minimum input voltage of about 0.1V (P-P) is required for the counter to count accurately.

Some Construction Notes

My PCB

Resistors R1 and R2 are pullup resistors for the I2C bus for the LCD display and the PCF8574. On the particular display I am using, the PCF8574 serial to parallel adapter board that is mounted on the back of the display has built-in pullups for the bus. If this is the case with your display, R1 and R2 don't need to be installed.

R3 through R6 were added to the Version 2.4 PCB. We were having a problem getting valid readings from the GAL register in the SN74LV8154 counter. Surprisingly, when we put a scope probe on the GAL pin, we suddenly got valid data! Please don't ask me to explain why, but adding the 1MΩ resistors fixed that problem.

But note that these resistors are mounted under the Nano processor, so be sure to install them before installing the Nano (mine is socketed). You use 1/4 watt resistors but 1/8 watt ones will provide a bit more clearance under the Nano.

Should you decide that you only want to use the device in one mode or the other, you can eliminate the mode switch. To use the device in *Frequency Standard* mode only, place a jumper between pins 1 and 2 of the SWITCH 1 header (or pads if you didn't even bother to install the headers). To use the device in *Frequency Counter* mode only, place a jumper between pins 3 and 3 of the SWITCH 1 header and a jumper between pins 2 and 3 of the SWITCH 2 header.

The counter PCB is designed to stack on top of the QRP Labs GPS PCB (or you can put the GPS on the top). I chose to put my PCB on the top.

The QRP Labs QLG1 GPS Receiver Kit

Should you choose to mount my PCB on top of the GPS module you should not mount an SMA external antenna connector on top of the GPS board or you won't be able to connect the antenna once the boards are stacked (please don't ask me how I know this).

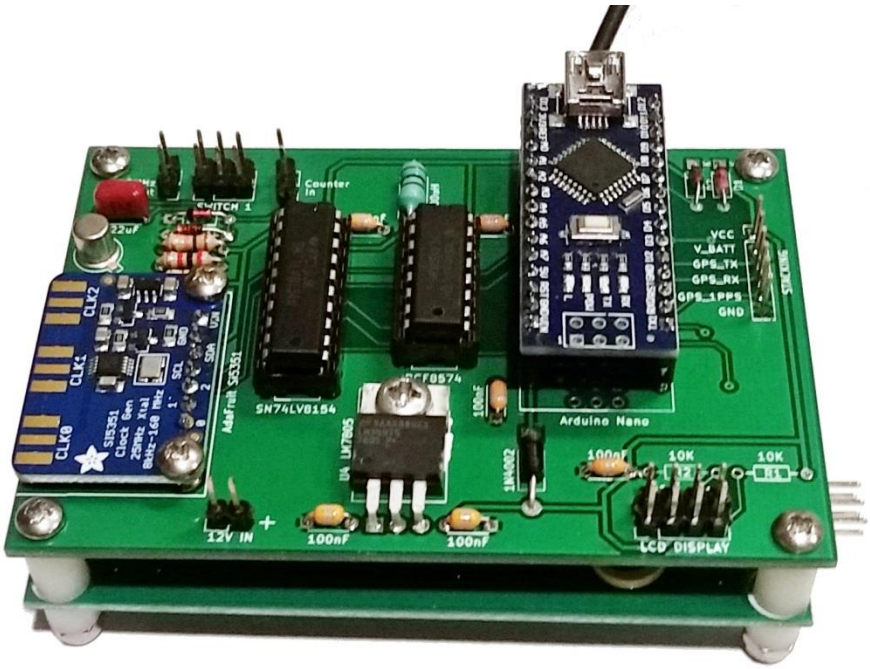
I just hardwired a short length of RG-174 to the GPS PCB with an SMA connector on the other end. If you do this, make sure you add a means of providing some strain relief for the cable.

If you are going to use the built-in antenna on the GPS module instead of using an external antenna, you'll want to put the GPS board on the top and install the gimmick in a plastic box.

The QRP Labs GPS module has three LEDS on the PCB that show power on, data being sent and the *1PPS* signal. It's a good idea to connect the LEDs to the PCB via cables such that they can be viewed from the outside of whatever enclosure you decide to put this into. I also found that you can install the LEDs provided with the kit on the GPS PCB and additionally add external ones (~20mA) using the pads on the GPS module provided for the external LEDs.

The Assembled PCBs

Here's what the Version 2.8 PCB looks like sitting on top of the QRP Labs GPS. The RG-174 pigtail that looks like it's coming out of the USB connector is for the external GPS antenna. Not visible in the picture; I actually drilled a hole in the ground plane area near the edge of the GPS module to allow a small wire tie to serve as a strain relief for the cable.



The Software

Overview

The software is not all that complicated. The Si5351 code was hijacked from the VFO project that Glenn (VK3PE), Jim (G3ZQC) and I developed recently and that is a modified version of the code used in the VFO originally created by T.J. Uebo (JF3HZB).

Like all Arduino software, this program has the two standard functions, *setup* and *loop*. The *setup* function initializes everything need for the program to run and the *loop* function handles all the real work:

- It checks for an incoming *1PPS* pulse from the GPS module and if one has occurred processes it.
- Look for characters coming from the GPS and feeds them to the GPS library.
- In the *Frequency Standard* mode, it compares the frequency read to the *STANDARD_FREQ* (10MHz) and if the frequency read is not correct, it applies a correction factor to the Si5351 module.

  Note that in the *Frequency Standard* mode the reported frequency is an average of *GPS_READ_COUNT* readings. In *Frequency Counter* mode, the readings are not averaged.
- Check to see if the second's component of the time has changed and if so, update the time on the display (the GPS clock counts in hundredths of seconds but we don't want to update the display that often).
- Check to see if the day component of the date changed and if so, update the date on the display.
- Checks for several error conditions and notifies the user. Examples of such errors include loss of GPS lock, the need to recalibrate the Si5351, etc.

There are a number of other functions to handle certain tasks such as formatting text strings, reading the frequency from the counter and calibrating the Si5351, etc. The code contains lots of comment lines describing how things actually work, so I'll not elaborate here.

Reading the Counter

Common to both modes of operation is the actual counter. It is an SN74LV8154, which actually has two 16 bit counters that can be chained together to make a single 32 bit counter. That is how we have it implemented here.

It monitors the frequency input in *Frequency Counter* mode and monitors the *CLK0* output of the Si5351 in *Frequency Standard* mode. We never reset it as long as we have GPS lock, but rather read the 32 bit value whenever we receive the *1PPS* signal from the GPS. Subtracting the previous reading from the current reading gives the current frequency reading. There is a special case for handling when the counter rolls over back to zero.

In *Frequency Standard* mode the software maintains a list of the last *GPS_READ_COUNT* (currently set to 8) frequency readings and reports the average of those readings. This should reduce jitter, particularly when monitoring the Si5351 which is known to have a couple of Hertz of jitter based on some testing done by Jim and Glenn (the one I'm using doesn't seem to have this problem).

In *Frequency Standard* mode, the frequency should always be the *STANDARD_FREQ* (currently set to 10MHz). If the average frequency reading changes by more than *MAX_ERROR* (currently set to zero), the software initiates a correction procedure to adjust the Si5351 output and notifies the user.

In *Frequency Counter* mode the frequency readings are not averaged.

The GPS Module

This device is designed to use the QRP Labs QLG1 GPS Receiver Kit; however any GPS receiver capable of producing the *1PPS* pulse can be used. Note, the *1PPS* signal from the QRP Labs GPS is a *HIGH* pulse. Should you chose to use a GPS receiver that produces a *LOW* pulse, it's simple enough to modify in the software. The PCB is designed to stack on top of the GPS PCB.

The GPS module does two things; most importantly, it sends the *1PPS* pulse that tells the counter chip and the software exactly when the GPS satellites report the start of a new second. According to the datasheet for the YIC5 GPS receiver module used in the QRP Labs kit, the accuracy of the *1PPS* pulse is +/- 11nS.

Internally, the Adafruit GPS library maintains a table containing the following data:

- location – the latest position fix
- date – the latest date fix (UTC)
- time – the latest time fix (UTC)
- speed – current ground speed
- course – current ground course
- altitude – latest altitude fix
- satellites – the number of visible, participating satellites
- hdop – horizontal diminution of precision

We only use the time and date, however one could modify the code to extract the other data elements quite easily.

In theory, any GPS unit capable of producing the 1PPS signal can be used in the device. If the GPS is capable of sending NMEA standard GPS messages (NMEA 0183 Version 4.01), that's even better, as the unit will then display the date and time.

But there is one possible problem. Some GPS units have a default baud rate of 4,800. If you choose a unit that operates at 4,800 baud, it won't work here out of the box.

The problem can be solved however by writing a simple program using the *SoftwareSerial* library to send a command to the GPS module to change its baud rate. For modules using the *MediaTek* chipset, the command to change the baud rate to 9,600 is:

$PMTK251,9600*17<CR><LF>

You can also simply change the setting of the *GPS_SPEED* symbol in the *Counter.h* file to 4800, but if you take that approach, you will need to disable all of the NMEA messages except the *$GPRMC* message (See the *Disabling NMEA Messages* section below).

Disabling NMEA Messages

In the setup function in the program file, there are a number of lines of code to send messages to the GPS that look like:

```
//  gps.sendCommand ( PMTK_SET_NMEA_OUTPUT_RMCONLY );
```

If you plan to use the NMEA messages to provide the date and time information, the *$GPRMC* message is the only one needed, so you should un-comment the above line of code and the *delay* statement that follows at least the first time you load the software.

According to the datasheet, the module will retain any changed settings as long as the backup battery has enough voltage to support the internal memory. If the backup battery power is removed the module will revert to the factory settings.

If you are using a GPS that uses a different chipset you should figure out how to disable all but the *$GPRMC* message. The *Adafruit_GPS* library only decodes the *$GPRMC* and *$GPGGA* (location) messages, so anything else that is enabled is just noise. Mine is set up to send both of these messages and that is how the software as distributed is set up.

The program will run fine with all messages enabled ***UNLESS*** you set the *GPS_ECHO* symbol to *true*. With all the messages enabled and the serial output turned on, there are come timing problems that cause bogus frequency readings.

The Si5351 Clock Generator

The Si5351 is only used when the unit is operating in the *Frequency Standard* mode. In that mode it produces a frequency set by the definition of *STANDARD_FREQ* in the *Counter.h* file.

As the intended use of the device in this mode is to act as a 10MHz standard for use with other frequency counters such as the BG7TBL FA-2, the value of *STANDARD_FREQ* is set to 10MHz. But you could change it should you so desire.

Regardless of the setting of *STANDARD_FREQ*, the counter monitoring function should self-correct whenever the frequency wanders.

Exception Conditions

The user should be aware that whenever GPS lock is lost or during the recovery period after a loss of lock or when operating in *Frequency Standard* mode and the unit is performing a self-calibration, the frequency displayed and sent to the serial output will be the last frequency read before the exception occurred. In other words, the unit will not actually try to read the counter when one of the above conditions exists.

The “Hz” tag will have an asterisk (“Hz*”) and the display will indicate the reason why on the status line.

Arduino IDE Setup

This software is designed to be compiled and downloaded to the Arduino Nano using the Arduino IDE. Version 1.8.13 is the version when I'm using now. I don't expect that using newer versions would cause any problems.

You will need three additional non-standard libraries to compile the software; use the links to get the right ones, as there are several libraries out there with the same or similar names:

- **LiquidCrystal I2C** – This library handles the display interface.
- **Adafruit GPS** – This library handles the interface to the GPS module.
- **PCF8574** – This library handles the GPIO expander.

There are 3 different processor options available in the Arduino IDE when you select the board type as "Arduino Nano". The ones I have require you to select "ATmega328P (Old bootloader)" in order to get the software to load correctly; yours might be different.

User Customization

There are a number of things that the user might want to (or need to) modify. All the customizable items are found in the *Counter.h* file.

The file defines all the GPIO pins used. These are defined based on the PCB design and should not be changed unless you are using a PCB of your own design.

Optional Settings in *Counter.h*

The following are the optional hardware related features:

USB_SPEED & GPS_SPEED – Sets the baud rates for the serial monitor and GPS interface. The QRP Labs GPS defaults to 9600 BPS; other GPS units might be different.

PCF_I2C_ADDR – This defines the I2C address of the PCF8574. The normal values are 0x28 if you are using a PCF8574 and 0x38 for the PCF8574A. Should you have one with different address, please feel free to change this value.

SI_I2C_ADDR – The standard I2C bus address for the Si5351 is 0x60, however we have seen a few that are different. Change it if required.

LCD_ADDRESS – You might need to change the I2C address for your display. There are a number of Arduino programs on the web that will scan for I2C devices and report the address of all devices on the bus. Some I have are set for 0x27 and some for 0x3F.

GATE_BUTTON – As of Version 2.8 of the software (and hardware), the *GATE_BUTTON* can be optionally installed and is used to select the *gate time*. If you don't install the optional button, you should comment out the definition of this symbol.

SI_XTAL – This defines the crystal frequency of the Si5351 module. Most use a 25MHz crystal, but there are a few that use a 27MHz crystal. If yours uses a 27MHz crystal, change this value.

SI_CAL – You can use the *Calibrate Si5351* program to determine the calibration factor for your particular Si5351 and set this symbol to the appropriate value. Doing so will cause the program to start in *Frequency Standard* mode on or very close to the *STANDARD_FREQ*, however this is not absolutely necessary.

STANDARD_FREQ – Defines the frequency to be output when operating in the *Frequency Standard* mode. It can be changed if desired.

GOOD_LOCK – This defines the number of consecutive *1PPS* pulses required to ascertain that the counter is locked to the GPS. It can be changed if so desired.

MAX_ERROR – This defines how far off frequency the Si5351 is allowed to be before the self-calibration sequence is initiated. If the deviation is more than this value, the calibration will be started.

GPS_READ_COUNT – Defines the number of frequency readings that are averaged (only in *Frequency Standard* mode) to determine the frequency being measured. It can be changed but if you make it much bigger the Nano will run out of data memory space.

USE_NMEA – If set to *true*, the program will try to read the NMEA messages from the GPS; if set to *false*, it won't. See the *Disabling NMEA Messages* section above regarding which messages should be enabled and disabled.

GPS_ECHO – If the NMEA messages are being read, setting this to *true* will cause the messages to be displayed on the serial output.

If you change the settings of anything not listed above in the *Counter.h* file, you're on your own!

Frequency Calibration

The Si5351 crystal frequency as defined by the value of the *SI_XTAL* symbol is a nominal value. The actual crystal frequency is almost never going to be exact.

I wrote a program to determine the calibration factor that needs to be applied to a specific Si5351 (*Calibrate Si5351*). You can use this program to determine the calibration factor for the specific Si5351 used in the project however it is not required to calibrate the Si5351 prior to running the program; it does the calibration on the fly.

If you do determine the proper calibration factor, you can set the *SI_CAL* symbol to the known value.

The *Calibrate Si5351* program can be found on GitHub along with the instructions on how to use it should you want to use it.

GPIO Pin Definitions

The following is a summary of the GPIO pin assignments. Note that these represent how I have them assigned on the PCB. In most cases, the names are the same as those found on the schematic.

If you are using a PCB of your own design, you can change the assignments; the only limitation is that the *GPS_1PPS* pin needs to be an interrupt capable pin (D2 & D3 on the Nano). The list is in numerical order.

D0	USB_TX	Transmit half of the USB connection
D1	USB_RX	Receive half of the USB connection
D2	GPS_1_PPS	1 Pulse per Second signal from the GPS
D3	Available	
D4	GPS_RX	GPS receive line (processor transmit)
D5	GPS_TX	GPS transmit line (processor receive)
D6	GBU	Select counter B upper byte for reading
D7	GBL	Select counter B lower byte for reading
D8	GAU	Select counter A upper byte for reading
D9	GAL	Select counter A lower byte for reading

D10	SW2	Connection to mode switch
D11	RCLK	Load the counter value into the registers
D12	GATE_BUTTON_PIN	Optional pushbutton #1
D13	GATE_LED_PIN	Optional gate LED
A0	CCLR	Clear the counter
A2	SI_SDA	Si5351 I2C data line
A3	SI_SCL	Si5351 I2C clock line
A4	SDA	Standard I2C data line
A5	SCL	Standard I2C clock line

Again, note the use of separate I2C bus pin assignments for the Si5351 and PCF8574.

Operation

Connections

To use the device in either mode if you didn't install the on-board GPS antenna on the QRP Labs GPS PCB, then you need to connect an external GPS antenna.

If you are going to use the device as a stand-alone frequency counter, connect the "Counter Input" to whatever you wish to measure. The input is protected against overvoltage by diodes D3 and D4, and the 2N2222 preamplifier allows the unit to count accurately with an input voltage of about 0.1V (P-P) or more.

If you're going to use the device as a frequency standard for another counter (or some other purpose), connect the 10MHz output to whatever device you intend to use it with.

Display Indications

What you see on the display is pretty straightforward in either mode as selected by the “Mode” switch. When the unit is turned on, you will be greeted by the “Splash” screen which will remain visible for 3 seconds (it will now say Version 3.0):

Feel free to change the text on the “Splash” screen as you like. The text is found in the *PaintSplash* function.

The next screen will show the unit in “Startup” mode. The date and/or time may or may not be displayed initially; it seems to depend on how long the unit has been turned off:

If the *USE_NMEA* symbol is set to *false*, the date and time won’t be displayed:

Notice the frequency is 0Hz with an asterisk and the status is shown as “Startup”. Once GPS lock is achieved, you should see the following display (perhaps after a self-calibration as well):

Of course, if you are using the unit as a counter, the top line will indicate so (“Mode: FC”) and the frequency will be whatever you’re feeding into it.

If GPS lock is lost, this is what you’ll see:

Any time the frequency reading is suspect the “Hz” string will be followed by an asterisk.

The four lines of the display are fairly self-explanatory. The first line shows the operation mode and will say either “FS” or “FC” and will show the current setting of the *gate time*.

The second line of the display always shows the frequency in Hertz; if the frequency cannot be ascertained to be accurate, an asterisk will be shown after the “Hz” as in the picture above.

The reason the accuracy cannot be ascertained is shown on the 3rd line; in this case; the GPS lock has been lost. In other words, the unit is not seeing the *1PPS* signal from the GPS. The asterisk will also appear for several readings following a self-calibration or when recovering from a loss of GPS lock.

If GPS lock is lost, we do not attempt to take a new reading from the counter. Testing of the software indicates that the values we get from the counter are nowhere close to what the Si5351 is outputting. Instead, the software will report last reading obtained while the GPS lock was on with an asterisk. The Si5351 is fairly stable over short timeframes (several minutes at least).

If running in *Frequency Counter* mode the last reliable reading will also be displayed with an asterisk but note if the source frequency is changed the change will *NOT* be reported until GPS lock is again obtained.

The status messages that you might see on line three of the display are:

- **GPS Locked** – Everything is running as it should!
- **No GPS Lock**! – The processor is not seeing the *1PPS* pulse from the GPS; the frequency reported is suspect.
- **Startup** – The unit is just starting and has either not achieved GPS lock or taken enough readings to show an accurate frequency.

  You'll also see this message if you change modes while the unit is operating.
- **Calibrate** – This only appears when using the unit as a frequency standard. If the frequency differs from the *STANDARD_FREQ* by more than *MAX_ERROR* Hz (currently set to zero), the unit will attempt to recalibrate the Si5351. The unit will remain in this mode for "*GPS_READ_COUNT* +2" readings after the Si5351 is recalibrated.

The bottom line of the display shows the current date and time (UTC) if the *USE_NMEA* symbol is set to *true*. If the GPS is receiving good signals from the satellites, the time will be displayed almost immediately after the splash screen is replaced by an operational one (assuming the GPS is seeing a few satellites), however for some reason, it sometimes takes a few seconds before the date is displayed.

Also as noted earlier, the time is about 100 milliseconds behind the actual time. The problem is in the GPS module itself, so I can't fix it.

Gate Time

Upon startup, the unit will update the frequency every second. If installed, a momentary pushbutton can be used to set the *gate time* to 1S, 2S, 5S or 10S. Each time the button is operated (you need to hold it down for about a full second) the *gate time* will cycle to the next setting.

You can change the gate time selections by changing the values in the *gates* array in the program file. You can also add values.

If you have the gate time LED installed, it will flash briefly whenever the gate time has elapsed. Note, however that when the unit is in *STARTUP* or *CALIBRATE* mode or when GPS lock is lost, the LED will flash at 1 second intervals regardless of the *gate time* setting.

Serial Output

The program sends the frequency data to the serial port in a tab and space character separated format and includes the following information:

- Sequence number
- Date – mm/dd/yyyy
- Time – hh:mm:ss
- Frequency
- Hz or Hz*

For example:

Seq	Date	Time	Frequency
0	01/12/2020	23:01:59	10,000,034 Hz*
1	01/12/2020	23:02:00	10,000,002 Hz*
	I removed some redundant lines here		
11	01/12/2020	23:02:10	10,000,000 Hz
12	01/12/2020	23:02:11	10,000,000 Hz
13	01/12/2020	23:02:12	10,000,000 Hz
14	01/12/2020	23:02:13	10,000,000 Hz
15	01/12/2020	23:02:14	10,000,000 Hz
16	01/12/2020	23:02:15	10,000,000 Hz

If the *USE_NMEA* symbol is set to *false*, the date and time will not be included in the output.

The output can be copied from any program capable of reading the output stream (I simply use the Arduino IDE's Serial Monitor) and saved as a ".txt" file that can be imported into a program such as Microsoft Excel.

To import the file into Excel, open a new Excel workbook and from the menu select "File – Open". When the file selection window opens, you'll need to change the file type selection choice (drop down box above the "Open" and "Cancel" buttons) to "All files" or "Text files", then double click on the saved text file.

The Excel "Text Import Window" will appear. At the top of the window, select "Delimited", then click the "Next >" button.

On the next window, select "Tab" and "Space" and click the "Next >" button. On the next window, just click "Finish".

You will most likely have the resize the columns to the appropriate widths to make the data presentable.

From there, you can save the file as an Excel Workbook and proceed to do any analysis that you like including producing graphs.

Here's what the data from above looks like in Excel (I deleted the redundant calibration lines):

	A	B	C	D	E
1	Seq	Date	Time	Frequency	
2	0	1/12/2020	23:01:59	10,000,034	Hz*
3	1	1/12/2020	23:02:00	10,000,002	Hz*
4					
5	11	1/12/2020	23:02:10	10,000,000	Hz
6	12	1/12/2020	23:02:11	10,000,000	Hz
7	13	1/12/2020	23:02:12	10,000,000	Hz
8	14	1/12/2020	23:02:13	10,000,000	Hz
9	15	1/12/2020	23:02:14	10,000,000	Hz
10	16	1/12/2020	23:02:15	10,000,000	Hz

When the *gate time* is set to something other than 1 second, the data will only be sent to the serial output when the *gate time* has expired except when one of the exception conditions exists, in which case, the data will be sent approximately once per second.

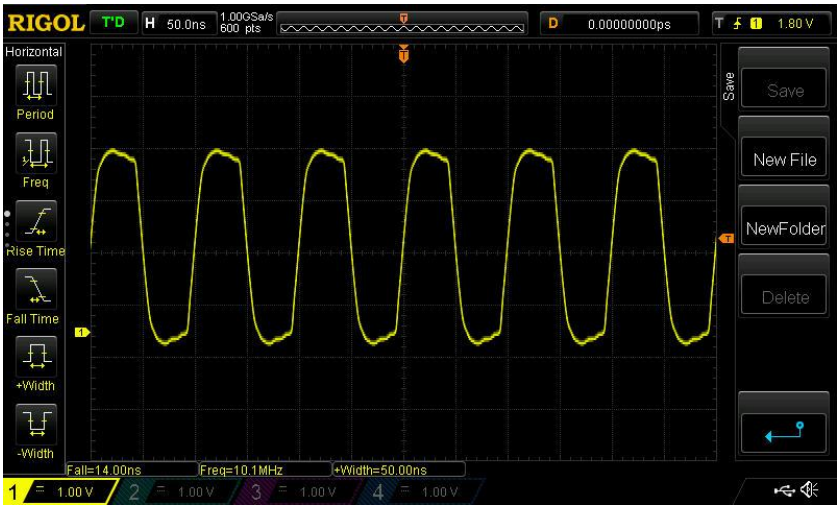
Accuracy and Stability

As best as I have the capability to measure the accuracy of the device, it seems to be pretty good, although not perfect!

There are a few things one must understand. First of all, the counter can only measure integer numbers; in other words the resolution is 1Hz. This applies whether using the device in *Frequency Standard* mode or in *Frequency Counter* mode.

In *Frequency Standard* mode, the stability of the Si5351 comes into play. Monitoring the output of the Si5351 with my BG7TBL FA-2 frequency counter, I normally see less than 0.5Hz of jitter over a few minutes, but that may be due to the behavior of the FA-2 or the Si5351 itself; I have no way of positively determining which.

On the scope, I see no sign of jitter on my test unit:



However, on at least one Si5351 module that Glenn has he sees quite a bit of jitter. It could be that it simply depends on the particular Si5351 being used.

In *Frequency Standard* mode the displayed frequency (with GPS lock) is an average of *GPS_READ_COUNT* readings. Those readings are stored as floating point numbers (even though the counter only reads integers) and the average is computed using floating point math.

In *Frequency Counter* mode, the readings are not averaged.

As described above, when the device is in *Frequency Standard* mode, anytime the (averaged) output frequency varies by 1Hz, a self-calibration process is initiated, and again, the amount of correction is computed using floating point math.

But here's the rub: The Arduino Nano has issues with floating point math! As near as I've been able to determine, the errors are in the 5th or 6th decimal places, so the errors are probably relatively small, but they do exist.

For what I intend to use it for, the accuracy is acceptable as it will stay within 1Hz of the *STANDARD_FREQ*.

Frequency Range

The SN74LV8154 counter chip is only rated to 40MHz, however, my unit seems to be accurate to about 48MHz; your mileage may vary!

Known Bugs & Glitches

Software wise, so far, so good, but there are three minor issues and one major issue that I haven't been able to solve.

Arduino Nano Using the "New" Bootloader

For some reason, when using the processors with the "New" Bootloader, the unit will go from "Startup" mode to "No GPS Lock!" even though the 1PPS signal is being sent from the GPS module to the processor. This behavior has been seen on two different builds and so far I have not been able to figure out why.

In *STARTUP* Mode

When the unit is first turned on, as noted previously the display will show it to be in "Startup" mode. If you have good signals from the satellites, it will remain in "Startup" mode for only about 10 seconds.

However, if GPS lock is not obtained, the display will continue to show “Startup” until GPS lock is obtained. I tried to modify the screen indication to show that the unit was in “Startup” mode with or without GPS lock, but there was no easy way to do so. This is why it’s a good idea to install the LEDs for the GPS module so they are visible on the outside of the enclosure.

Compiler Warning

It’s not a problem, but when you compile the software you will get a warning message from the Arduino IDE about low memory. The IDE generates this message anytime the amount of global data exceeds 75% of the processor’s available memory space for data storage.

Gibberish on the Display

On the Version 2.8 PCB, I added diode D5 to prevent the Arduino from trying to power the display and Si5351 when connected to the PC. You can connect it to a PC to update the software without turning on the rest of the power to update the software, but if you turn the power on ***AFTER*** connecting the USB cable, the display will contain gibberish!

There are two solutions. First is to power the unit up before connecting the USB cable. The second fix would be to install a pushbutton connected between the Arduino’s *RESET* pin and ground so the processor can be restarted easily.

But Wait, There’s More!

As a bonus, there is another program included in the distribution kit called *Where_MI*.

This program runs on the same hardware as the counter/standard and provides traditional GPS functionality. It will tell you your latitude, longitude, attitude and 6-character Maidenhead grid square.

In order for it to work, both the *$GPRMC* & *$GPGGA NMEA* messages must be enabled (see the section on *Disabling NMEA Messages* for more information).

The software is very similar to that for the counter/standard program so I won't bother providing a detailed description here. The comments in the program file should be sufficient.

Suggestion Box

I welcome any suggestions for further improvements. Please feel free to email me at WA2FZW@ARRL.net.

Bill of Materials

The following lists do not include optional parts such as IC sockets, for example.

Designation	Description
PCB	WA2FZW Version 2.8 printed circuit board
GPS Module	The QRP Labs QLG1 GPS kit (an External GPS antenna is optional)
Display	4 line by 20 character I2C LCD display
GN1	Adafruit (or equivalent) Si5351 module
U1	SN74LV8154 counter chip
U2	PCF8574 GPIO expander chip
U3	Arduino Nano processor (or a clone)
U4	LM7805 Regulator (TO-220 package)
Q1	2N2222 Transistor
J1	6-pin female stacking header
J2	2-pin male header or RG-174 pigtail with appropriate connector.
J3 & J5	6-pin (2x3) male header
J4	2-pin male header or RG-174 pigtail with appropriate connector.
J6 & J8	8-pin (2x4) male header
J7	2-pin male header
R1 & R2	10K 1/4W Resistor (optional – See *Some Construction Notes- My PCB*)
R3 – R6	1M 1/8W or 1/4W Resistor
R7	10K 1/4W Resistor

Designation	Description
R8	2K 1/4W Resistor
R9	100Ω 1/4W Resistor
C1 – C6	100nF Capacitor
C7	0.22uF Capacitor
D1 – D4	1N4148 Diode
D5	1N4002 Diode (or equivalent)
Mode Switch	DPDT Toggle switch
Power Switch	SPST Toggle switch
B1 – B2	SPST Pushbutton (only B1 is currently used by the software)
LEDs	The QRP Labs GPS kit comes with 1 each red, green and yellow LED. These can be mounted on the GPS PCB or wired via cables. You can also mount the provided LEDs on the GPS PCB and add external ones (~20mA). You can also add an optional LED to indicate when the *gate time* has expired. This needs to be connected to the connector labeled *B2* on the PCB in series with a 220 ohm resistor.
12V In & Out	Whatever connectors you like.
Counter Input	Whatever connector you like (BNC, SMA, etc.)
10MHz Output	Whatever connector you like (BNC, SMA, etc.)
1PPS Output	Whatever connector you like (BNC, SMA, etc.)