10 Milliwatt Si5351A WSPR Beacon

It’s fun to experiment with propagation by seeing how far you can get a signal to travel with a transmitter in the milliwatt range (this is known colloquially as QRPp). For quite a while, this esoteric pursuit was done with flea-powered CW signals, until the invention of the WSPR modulation scheme by Nobel laureate Joe Taylor, K1JT. WSPR is now a very popular method with hams around the world for HF propagation experiments; aided greatly by the WSPRnet online database, which allows hams to get detailed information about where and when their signals are being heard.

Although a fair amount of computation (for a microcontroller at least) is used to encode a WSPR message, the actual modulation scheme is relatively simple 4-FSK, which means that four different symbols can be encoded by transmitting on one of four discrete equidistant frequencies (spaced approximately 1.46 Hz apart). For any method of signal generation with a decent amount of frequency resolution such as a PLL or DDS, this is easily implemented. The Etherkit Si5351A Breakout Board combined with the Si5351Arduino library provides a low-cost, wide range PLL with excellent frequency resolution for a task such as this.

The tricky part is timing the transmissions correctly. WSPR requires transmissions that start on the first second of an even minute. Any transmission that deviates more than about one second either too soon or too late will not be decoded, so it is most important to be sure that you have an accurate clock when attempting to transmit with WSPR. Traditionally, this has been accomplished by using a PC synchronized to an internet time server to act as the controller for the WSPR transmitter (and often it is transmitted as AFSK via USB mode on a radio). As mentioned above, the WSPR modulation scheme is

very easily realized in microcontroller code, but timing becomes a bit more of a challenge. The on-board oscillator for a microcontroller simply does not have the ability to keep accurate time for very long, even if it is initially set accurately. An external RTC module is slightly better, but will still drift too far out of spec for WSPR after a short time. This has typically left a GPS module as the only viable solution for a microcontroller-based WSPR transmitter. While this is the ideal approach, plenty of experimenters don't have GPS modules sitting around ready to be plugged into their Arduinos.

However, by using the Arduino Time library and a internet connected PC, a simple WSPR transmitter can be built that does not require a GPS module for timing nor a separate "big rig" radio with upper sideband capability. The Time library can use various synchronization sources, including grabbing a Unix time string over the Arduino's USB-UART bridge. First, the PC must be accurately synchronized to a NTP time source. If you are using a Linux or Unix based OS, there are a variety of different methods for doing this, so search for documentation for your particular OS distribution. For a Windows PC, you'll need to use a third-party program such as Dimension 4. In all cases, *do not rely* on the settings in your operating system's time control panel for internet time. Typically, these services do not maintain good clock discipline and it's not uncommon for your PC clock to drift far enough out of sync for your WSPR frames to be invalid after a few weeks.

The sending of the Unix time string to the Arduino can be handled by a variety of different programming and scripting languages. Python is the preferred scripting language for such tasks here at Etherkit, so we'll provide a simple example Python script below, but the same thing could be accomplished in many different ways. The firmware in the Arduino will send an ASCII bell character to the host PC when it needs to get a Unix timestamp for synchronization, so the program simply

needs to open the serial port of the Arduino's USB-UART bridge, wait for an incoming ASCII bell, and then send back a properly formatted Unix timestamp. When the Arduino is properly synchronized, it will light up the LED on pin 13 to let you know. Typically you just need to plug the Arduino into your PC, execute the time synchronization script on your PC (python ntpserial.py), and then the pin 13 LED should light up within a few seconds to let you know that the beacon transmitter is ready to go.

Before loading the firmware onto the Arduino, be sure to change the callsign (6 characters or fewer) and locator (exactly 4 characters) variables to the proper values for your station (and perhaps the dBm if you use an amplifier). Set the frequency to any one that you like within the tuning range of the Si5351 by setting the freq variable. You'll also want to be sure to have properly frequency calibrated your Si5351 and use the correction factor in the CORRECTION define near the top of the firmware file. The Etherkit JTEncode library takes care of generating the correct WSPR symbols for you, so there's no need to have the PC use the wsprcode application for this. The if block at the end of the loop() function determines when the beacon will transmit. It is set to transmit every 10th minute of the hour, but this can be tweaked by changing the modulo math in that if statement.

Since the Si5351 output is a square wave, make sure you properly low-pass filter the output before connecting it to an antenna for transmission (you can find tables for low-pass filter generation in a variety of places, including the ARRL Handbook). You may also wish to connect a current-limited LED to pin 12, as this pin will go high to indicate when the beacon is transmitting.

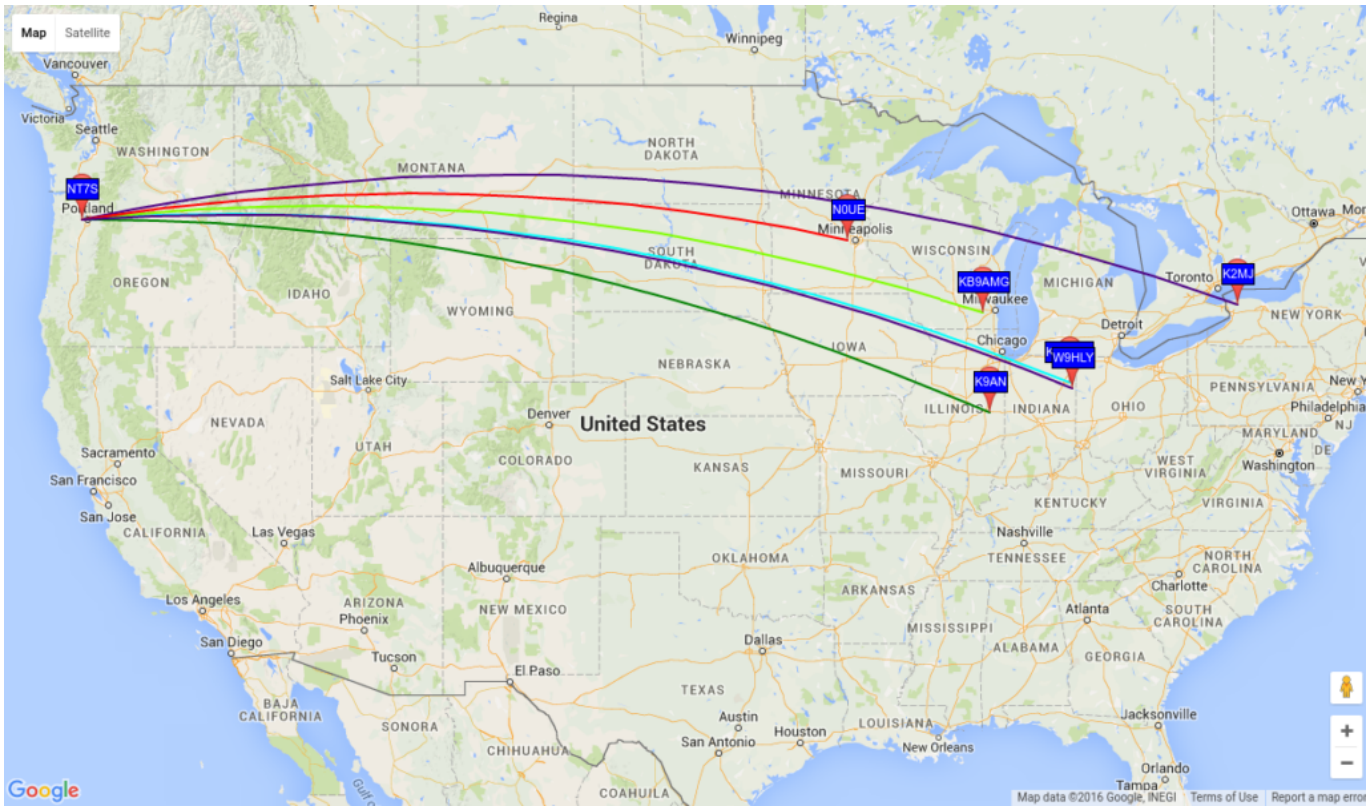
While 10 milliwatts is an awfully sparse amount of power, due to the robust WSPR scheme, you can get spots with that power level and an average HF antenna. As a demonstration at the NT7S shack, we set the beacon to transmit on 20 meters and let

it run for 24 hours. Conditions weren't great, but we did manage to get spotted, especially during the local twilight hours. The antenna here is a basic double (ZS6BKW) up about 12 meters or so.

14 spots:

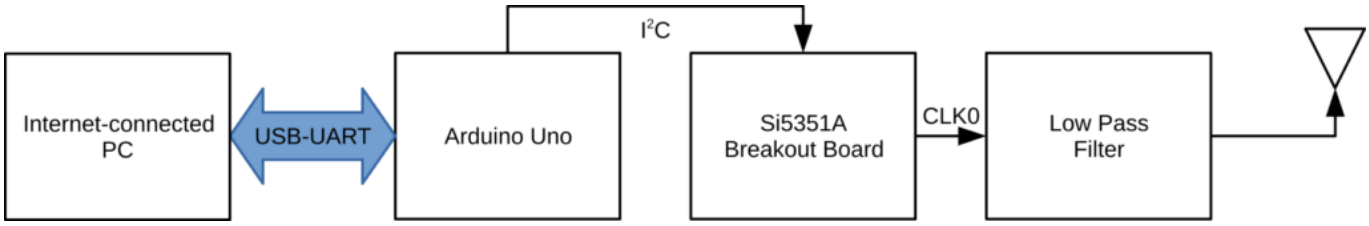
Timestamp	Call	MHz	SNR	Drift	Grid	Pwr	Reporter	RGrid	km	az
2016-04-16 02:50	NT7S	14.097122	-27	-1	CN85nm	0.01	K2MJ	FN03qe	3483	78
2016-04-16 02:40	NT7S	14.097090	-19	0	CN85nm	0.01	KA9LHE	EN70lx	3076	86
2016-04-16 02:40	NT7S	14.097089	-27	0	CN85nm	0.01	W9HLY	EN70mt	3090	86
2016-04-16 02:30	NT7S	14.097089	-25	1	CN85nm	0.01	W9HLY	EN70mt	3090	86
2016-04-16 02:30	NT7S	14.097089	-23	0	CN85nm	0.01	K9AN	EN50wc	2874	90
2016-04-16 01:30	NT7S	14.097093	-18	0	CN85nm	0.01	KB9AMG	EN52tx	2741	83
2016-04-16 00:50	NT7S	14.097093	-22	-1	CN85nm	0.01	KB9AMG	EN52tx	2741	83
2016-04-16 00:40	NT7S	14.097094	-25	0	CN85nm	0.01	KB9AMG	EN52tx	2741	83
2016-04-16 00:40	NT7S	14.097099	-24	-1	CN85nm	0.01	N0UE	EN34fx	2284	81
2016-04-16 00:30	NT7S	14.097099	-22	0	CN85nm	0.01	N0UE	EN34fx	2284	81
2016-04-15 23:40	NT7S	14.097092	-23	0	CN85nm	0.01	KB9AMG	EN52tx	2741	83
2016-04-15 21:10	NT7S	14.097092	-26	0	CN85nm	0.01	KB9AMG	EN52tx	2741	83
2016-04-15 21:00	NT7S	14.097037	-26	0	CN85nm	0.01	K5RHD	DM65pc	1789	125
2016-04-15 19:00	NT7S	14.097088	-22	0	CN85nm	0.01	KA9LHE	EN70lx	3076	86

As you can see, 10 milliwatts on 20 meters WSPR was about to reach out to an area about 3000 km distance, which isn't bad, all things considered.



With some amplification to half a watt or so, it would be very easy to get spots, but there's fun in seeing how far you can send a flea-powered signal and still have it be perfectly decoded (watch for our upcoming 500 mW linear amplifier, which would be a perfect accompaniment to this project). There's plenty of areas where this project could be expanded, so have fun hacking this simple example into something even more useful.

Block Diagram



Bill of Materials

Item	Quantity
Arduino Uno (can substitute other variants)	1
Etherkit Si5351A Breakout Board	1
Low Pass Filter	1
Amplifer (optional)	1
LED (optional)	1
470 ohm 0.25 W resistor (optional)	1
RF connectors of choice	

Wiring

A transmit LED indicator can be wired with a 470 ohm current limiting resistor in series by connecting to pin 12.

Terminal	Arduino Uno Pin
Si5351 SCL	A5
Si5351 SDA	A4
Optional LED/resistor	12/GND
Si5351 5V	5V
Si5351 GND	GND

Usage

Simply load the sketch onto your Arduino Uno, connect the power and I2C lines from the Uno to the Si5351A Breakout Board and then connect an appropriate low-pass filter to the output of the Si5351A Breakout Board CLK0 (or the output of the amplifier, if you are using one). Connect the output of the low-pass filter to a dummy load, then provide power to the Uno, Si5351A Breakout Board, and amplifier (if applicable).

Connect the Arduino USB to your PC and then run the provided Python time synchronization script:
python ntpserial.py
Once the Arduino is able to get a valid timestamp from the PC, the pin 13 LED will light and your beacon will be in transmission mode.

Use your favorite WSPR program and a PC-connected receiver to monitor your transmission in order to ensure that your setup is working correctly. Once you are satisfied that is the case, connect the output of the low-pass filter to an antenna in order to transmit your WSPR transmitter on the air. Don’t forget to change your callsign and the grid locator in the Arduino sketch before you put it on the air.

Extending the Transmitter

With only about 10 to 20 mW of output power, it will be tough to hear this transmitter barefoot, so some sort of amplifier would be very useful to bring the output level to a point where you can reasonably expect to be heard. In the near future, we intend to publish an example design for a linear amplifier that will bring the output power up to 500 mW, and will update this post with the link to it when it is ready.

If you have a GPS module with a serial UART available, you can bypass the PC connection entirely and have the Time library consume time data from the GPS. See the TimeGPS example sketch provided with the Arduino Time library for an example of how to do this.

Since this project uses a connection to an internet-connected PC, there’s no reason why it could not be used in an Internet of Things fashion and be controlled via LAN or over the Internet. This would give you the ability to be a remote control operator, so that you can be away from the transmitter and still maintain control over it.

Arduino Sketch and Python Script

Required Libraries

All of the required libraries are available through the Library Manager in recent versions of the Arduino IDE. This is the preferred method for obtaining the libraries, although the Si5351Arduino and JTEncode libraries can be manually installed by downloading ZIP files from the links below.

Si5351Arduino
JTEncode
Time
Wire

Links

Si5351Arduino Library on GitHub
JTEncode Library on GitHub
Si5351_WSPR.ino sketch on Gist