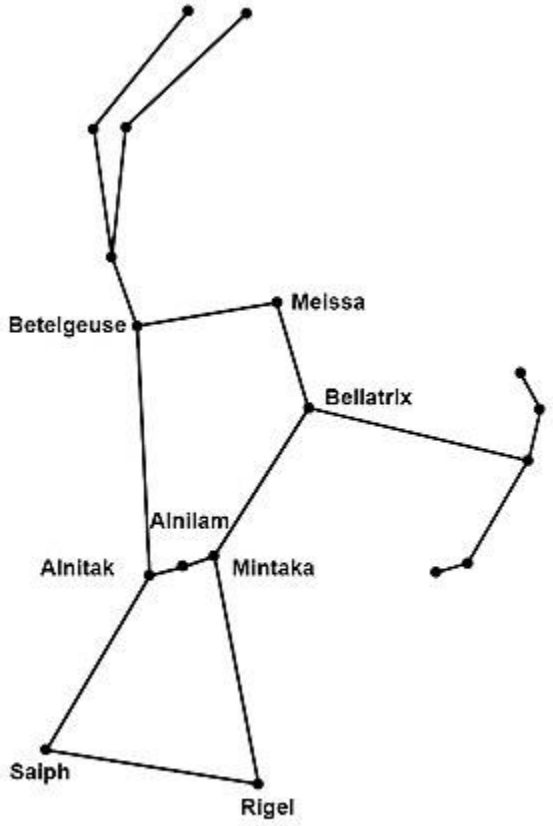


Orion

WSPR BEACON FIRMWARE FOR PICO-BALLOON PAYLOADS USING ARDUINO AND SILICON LABS SI5351A

(ORION RELEASE 1 - DOCUMENT VERSION 0.4 DRAFT)

Michael Babineau | VE3WMB | January 21st, 2020

Contents

Introduction

This open-source software project implements a very low power Arduino HF WSPR Beacon using a Silicon Labs Si5351a clock chip as the beacon transmitter and a GPS receiver module for timekeeping and for determining both location and altitude. It was developed for the Ottawa Valley Mobile Radio Club (OVMRC) sponsored project called ARIES.

This work is dedicated to the memory of my very dear friend Ken Louks, WA8REI(sk)

- Michael Babineau, VE3WMB - November 4, 2019.

Acknowledgements

I wish that I could say that I came up with this code all on my own, but much of it is based on preexisting work done by people far cleverer than I am.

It was derived from the "Simple WSPR beacon for Arduino Uno, with the Etherkit Si5351A Breakout Board", by Jason Milldrum, NT7S whose original code was itself based on "Feld Hell beacon for Arduino" by Mark Vandewettering K6HX and adapted for the Si5351A by Robert Liesenfeld AK6L <ak6l@ak6l.org>.

Timer setup is based on code that was written by Thomas Knutsen LA3PNA. Time code was originally adapted from the TimeSerial.ino example from the Time library.

The original Si5351 Library code from Etherkit (aka NT7S) was replaced by self-contained SI5315 routines written Jerry Gaffke, KE7ER. The KE7ER code was modified by VE3WMB to use 64-bit precision in the calculations to enable the sub-Hertz resolution needed for WSPR and to allow software I²C usage via the inclusion of SoftWire.h.

The QRSS FSKCW Beacon is derived from the "QRSS/FSKCW/DFCW Beacon Keyer" by Hans Summers, GoUPL(copyright 2012) and used with his permission. The original source code is from here :

https://qrp-labs.com/images/qrssarduino/qrss.ino

I also credit Hans, GoUPL as the source of inspiration for the CALIBRATION and PARK features implemented in Orion. Hans is the proprietor of QRP_LABS (www.qrplabs.com)

and his excellent Ultimate3S QRSS/WSPR Transmitter kit incorporates similar features which inspired me to develop this functionality for Orion.

I would like to thank Alain De Carolis, K1FM, who in addition to developing an excellent Beacon Board design (https://github.com/adecarolis/K1FM-Pico-Balloon) for our use with Orion, also had numerous suggestions that helped to improve and evolve the code.

Lastly, I would like to acknowledge the tremendous contribution to this project by Graham Collins, VE3GTC. Graham is my sounding board, my hardware guru, GPS expert and all-around subject matter expert. He is the guy I turned to when I got stuck and didn't know how to proceed. I also credit Graham for the original idea and much of the implementation details for the K.I.S.S. Telemetry scheme. Without his assistance Orion would not have been possible, at least not by me.

Conventions used in this document

For consistency and as an aid to the reader, I have adopted a few conventions used in this document:

- File names for the source code appear as normal text but underlined. This includes Arduino Libraries. i.e. OrionBoardConfig.h , Wire.h
- The use of C language or C pre-processor key words are italicized. i.e. *#define*
- The names of configuration parameters and their nominal values are in bold text. i.e. **BEACON_FREQ_HZ 14097010UL**
- Orion feature names are written in all upper case. i.e. CALIBRATION.
- Pin references used in this document refer to the Arduino naming conventions for Pins, not the ATmega328p naming conventions. Translation between the two will be necessary when creating a board configuration file for a new board.
- Installation instructions for Arduino libraries and Orion itself assume that the reader is using the Windows operating system.

Overview

Why is it called Orion?

Serendipity!

I was trying to think of a short name for this project one evening while sitting on the couch. I happened to look out at the night sky through the window and there was the constellation Orion staring back at me, so Orion it is.

What does it do?

Orion is an Arduino-based WSPR Beacon application. The development was done using the C programming language within the Arduino IDE and using Arduino libraries. The software is flexible enough to support differing hardware configurations, but it assumes that the ATmega328p processor that it is running on has the appropriate Arduino bootloader installed. It also assumes the presence of a GPS module with a one pulse per second (1PPS) signal and a Silicon Labs Si5351a clock chip used as the transmitter. For this first release the assumption is also that the ATmega328p is running at 8 MHz with a 3.3v supply voltage, and thus the Arduino IDE believes that it is talking to an Arduino Pro Mini 8Mhz/3.3v (this can be changed but does require some code modification to do so).

The application uses the fix data from the onboard GPS receiver module to set the time of a software system clock, that runs on UTC time. That system clock drives a simple beacon scheduler that generates events that are then passed to the Orion finite state machine. The reception of specific events by the Orion state machine causes state transitions and the generation of actions which, when handled, do the actual work such as the collection of telemetry data or WSPR transmissions. The GPS-synchronized system clock ensures the accuracy of the starting times for the WSPR transmissions. The timing of the transmissions is controlled by a 1.46 Hz processor timer interrupt to ensure accuracy. GPS also provides position data that is transmitted as telemetry within the WSPR messages. Other telemetry such as temperature and processor voltage are sent in secondary WSPR messages. The one pulse per second (1PPS) signal from the GPS receiver, plus a CALIBRATION clock signal from the Si53351a fed back to pin D5 on the processor, implement a software frequency counter, which allows for self-calibration of the WSPR transmit frequency with a resolution of around one Hertz at 14 Mhz.

This code was developed for Project Aries, supported by the Ottawa Valley Mobile Radio Club (OVMRC) specifically to meet the needs of our Pico-Balloon project. Throughout this development we have attempted to produce code that may also benefit others, either in whole or part.

Where can I find the source code?

Orion is freely available in GITHUB under a GPL3 license at this location :
https://github.com/ve3wmb/OrionWspr

In a nutshell, the GPL3 license allows you to do as you see fit with this software so long as you recognize and state the copyright information and provide a copy of the GPL3 license with any copy or derivative work that you produce and distribute.

See the section titled SOFTWARE LICENSING in the Appendix for more details on the terms of the GPL 3 License.

What does this document cover?

What I hope to convey to the reader in this document is an overall understanding of the software features provided by Orion and sufficient understanding of its configuration to allow potential users to set it up it to run it on their own beacon board, without significant code modification.

I have included some additional information in the Appendix to aid those that might like to modify or adapt the code to their own specific needs. My assumption is that if you are going to attempt this, that you are familiar with Arduino and have some reasonable level of experience in programming real-time systems in C.

As stated in the terms of the GPL3 license, this is free software and as such it comes as is, without any implied warranty or support. If you ask nicely, I will attempt to help you and answer questions via email, time permitting of course. However, I can't teach you how to program in C or fix bugs or compile issues in your own code.

Copyright

Hardware Requirements

This firmware must be run on an Arduino or an ATmega328p microcontroller with the Arduino Pro Mini 8Mhz/3.3v Arduino Bootloader installed.

This code may be modified to run at 16Mhz/5v and if so then it is recommended that you use the Arduino Nano Bootloader. If you wish to use this code on a processor with a non-standard frequency crystal, then you will likely need a custom board configuration file added to your Arduino IDE.

I have tried to keep the AVR/Arduino port usage in the implementation configurable via *#defines* in a file called OrionBoardConfig.h so that this code can more easily be setup to run on different hardware. This was also the motivation for using the SoftWire.h library as the software I²C solution. The SoftWire.h interface mimics the Wire.h library so switching back to using hardware enabled I²C via this configuration is very simple, with no code changes needed. This selection is achieved through conditional compilation of the source.

Orion is compatible with QRP LABS U3S clone boards, such as the DL6OW Stella Boards, and the K1FM V1.3 board, all available from OSHPARK (see the RESOURCES section in the APPENDIX for links).

Orion supports either hardware serial or software serial (using the NeoSWSerial library) to communicate with the GPS module at a default 9600 baud. It assumes that whichever option is used for GPS communication, the opposite is used for the debug serial port connection.

Orion also supports the use of pin D2/D3 External Interrupts or Pin Change Interrupts on another user-defined pin to handle sampling of the CALIBRATION clock output from the Si5351a for the CALIBRATION feature. To utilize CALIBRATION the board must route the GPS 1PP signal to D5 on the processor, as this is the external counter input for the T1 Timer/Counter. Without this connection, self-calibration is not possible.

Orion Feature Details

BOARD CONFIGURATION

Board configuration is a capability that allows Orion to be easily adapted to different beacon board layouts. It defines a series of parameters that are used during both compilation and execution of the code to allow the software to adapt itself to the particulars of a specific board, with little or no user changes required to the code itself. The differences that it supports are:

- Hardware or software I²C to communicate with the Si5351a clock generator
- Hardware or software serial to talk to the GPS module
- Hardware or software serial for debug output
- Pin wiring configurations for the Si5351a clocks
- The use of either External interrupts or PinChange interrupts for utilization of the GPS 1PPs signal as a time-base for CALIBRATION (External interrupts use pin D2,D3 and PinChange interrupts are used for others).
- Temperature sensor configuration (currently supports processor internal, or external TMP-36 or 1-wire DS1820).
- Initial correction factor for Si5351a clock to be used on startup (this is modified on the fly during subsequent self-calibration cycles).

This capability is provided by the file OrionBoardConfig.h and allows us to support both U3S clone boards such as the Dl6OW boards, as well as the K1FM V1.3 boards without changing any actual code. Examples can be found in the folder board_config_files within the GIThub project.

More details on parameter configuration and customizations are covered in the section CONFIGURING ORION.

BEACON SCHEDULER

Orion uses a fixed scheme for scheduling WSPR transmissions based on a ten minute transmission cycle. The scheduler is implemented by a function called *orion_scheduler()* located in the main file OrionWspr.ino. It generates time events at fixed intervals according to the time provided by the Orion software system clock, which is implemented using the Time.h library. The Orion system clock provides a Unix-like time function based on Epoch time (seconds since 00:00 UTC on January 1, 1970).

In a nutshell, primary WSPR transmissions are triggered at one second after hh:00, hh:10, hh:20, hh:30, hh:40, hh:50, where hh is the hour. (note that WSPR transmissions must always begin at one second after an even minute). These are normal type-1 WSPR messages but with the dBm field overwritten with a value representing the encoded 5th and 6th characters of the Maidenhead Grid Square, calculated based on the position information (latitude/longitude) data received from the GPS receiver prior to transmission. Note that the first four characters of the Maidenhead Grid are already included in type-1 messages in the GRID field. At one second after hh:02, hh:12, hh:22, hh:32, hh:42 and hh:52, a secondary type-1 WSPR message is transmitted with the dBm field overwritten with a value representing the encoded voltage, temperature or altitude, depending on the specific timeslot and as shown in Figure 1 : Orion Beacon Schedule.

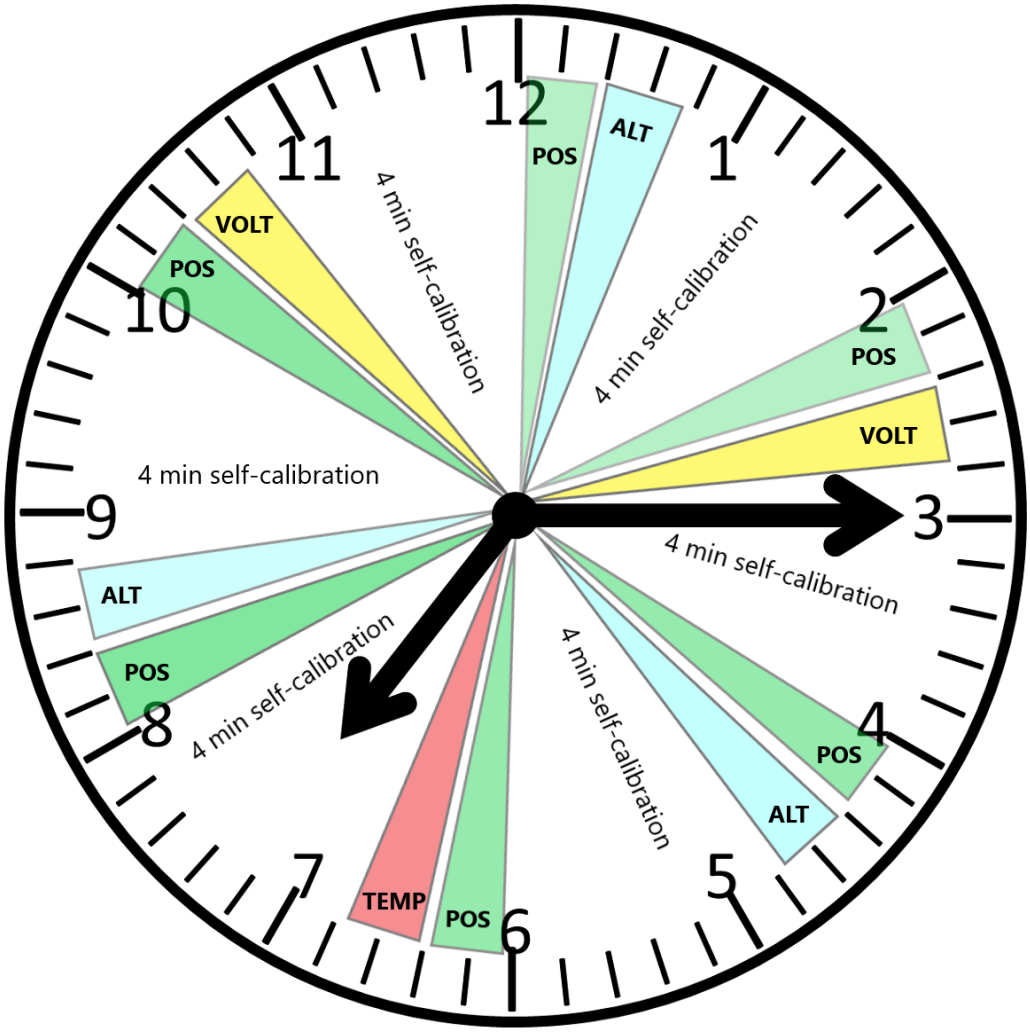


Figure 1 : Orion Beacon Schedule

With reference to Figure 1, each colored pie-slice represents a WSPR transmission. The green slices are the primary transmissions which encode the 5th and 6th characters of the Maidenhead Grid (POS means position) into the dBm message field. The blue, yellow or red slices represent secondary WSPR Transmissions which encode altitude (ALT), voltage (VOLT) and temperature (TEMP) into this same field. Details of this encoding scheme are explained in the TELEMETRY section of this document. Also note that between the secondary transmission of one cycle and the primary transmission of the next cycle there is an approximate four-minute interval during which self-calibration of the WSPR transmitter frequency takes place. This is detailed in the section titled CALIBRATION.

What is not show in Figure 1 is that the scheduler also triggers the re-synchronization of the Orion software system clock at regular intervals, one minute prior to the primary

WSPR transmissions. The software system clock time is only reset if the time data in the current GPS fix is considered valid. This same event also triggers the collection of current telemetry data for use in the next WSPR transmission cycle (i.e. data for both primary and secondary messages).

As stated previously this transmission schedule imposed by the scheduler is fixed in the code. In future we will investigate making this more flexible and more easily configurable to suit the needs of anyone wishing to modify the behavior without making code changes.

PARK

The QRP Labs U3S beacon transmitter implements a feature called PARK MODE which is used to keep the Si5351a temperature consistent during idle periods where the beacon is not transmitting or self-calibrating. In Orion we implement a very simplistic sub-set of the QRP LAB's PARK MODE feature set which we refer to simply as PARK. PARK on Orion simply ensures that when the Si5351a is otherwise idle, it is transmitting a carrier on the PARK clock (defined by parameter **SI5351A_PARK_CLK_NUM**) using a frequency defined by the parameter **PARK_FREQ_HZ**. The idea is to keep the SI5351a warm during idle periods to avoid thermal-induced drift during WSPR transmissions, resulting from the heating of the Si5351a and associated circuitry during transmission and subsequent cooling during idle periods. Due to the limited fixed configuration of the Si5351a VCOA this frequency must be less than 109 Mhz. The nominal value is 108 Mhz.

CALIBRATION (AKA SELF-CALIBRATION)

Self-calibration of the Si5351a clock is made possible by two hardware connections. The first is the connection of a spare Si5351a clock designated by the parameter **SI5351A_CAL_CLK_NUM** (OrionBoardConfig.h) connected to pin D5 on the ATmega328p. This is a necessity as D5 is the external counter input for the 16-bit Timer-1 on the ATmega328p. This is used to count signal pulses from the Calibration clock, using hardware interrupts to measure its frequency. Without this hardware connection self-calibration will not work.

The second hardware connection is the 1PPS signal from the GPS to pin D2/D3 (preferred), or some other unused digital input. If the connection is to either D2 or D3 then External Interrupts will be used, otherwise Pin Change Interrupts will be used. This is configurable in OrionBoardConfig.h (see the sections Board Configuration and Configuring Orion for more details).

During calibration the Si5351a is instructed to generate a calibration signal on the **SI5351A_CAL_CLK_NUM** (OrionBoardConfig.h) using the frequency **SI5351_CAL_TARGET_FREQ** (OrionBoardConfig.h). The processor then samples the frequency (counts the number of pulses over a 10 second interval) using the 1PPS signal from the GPS as an accurate time base. This allows for the determination of the frequency down to 1/10th of a Hz. As we assume an 8 Mhz processor clock speed, the maximum

frequency that we can sample is 4 Mhz but we de-rate this to 3.2 Mhz. So the value of **SI5351_CAL_TARGET_FREQ** (OrionBoardConfig.h) represents a 3.2 Mhz calibration frequency source.

After each ten second measurement interval, the calibration code employs a Huff-and-Puff algorithm to apply a frequency correction step to the Si5351a. The initial calibration after startup uses a coarse correction step designated by **COARSE_CORRECTION_STEP** (OrionCalibration.h) which is nominally one Hertz. Subsequent calibration cycles will apply a **FINE_CALIBRATION_STEP** (OrionCalibration.h) which is 1/10 Hertz. Each calibration cycle is comprised of 23 measurement/correction intervals so the maximum correction that may be applied during initial calibration is 23 Hz and during subsequent calibrations this is 2.3 Hz. There are actually twenty-four samples, but the first sample per cycle is discarded due to start-up errors. The result is that each calibration cycle requires about four minutes to complete.

Self-Calibration is triggered at Orion Startup if the defined parameter **SI5351_SELF_CALIBRATION_SUPPORTED** (OrionBoardConfig.h) is true. Note that the starting point for the correction value is set by the parameter **SI5351A_CLK_FREQ_CORRECTION** (OrionBoardConfig.h) and may be either negative or positive. It will vary from board to board. More details will be provided in the section titled BOARD CONFIGURATION.

The goal of self-calibration is to keep the WSPR transmit frequency within the 200 Hz WSPR transmission window. Over time it can maintain a frequency accuracy of around one Hertz at 14 Mhz when used in conjunction with PARK. Note that this is different from the requirement of frequency stability during the transmit cycle which relies primarily on the stability of the Si5351a clock circuit.

A WSPR signal occupies a 6Hz bandwidth and uses 4 tones spaced 1.46 Hertz apart. Decoding requires that the frequency vary no more than at most 4Hz during the almost two-minute transmission duration. This frequency stability is maintained through the use of a TCXO (recommended) in place of a crystal for the Si5351a and also by keeping the Si5351a temperature relatively constant using PARK.

Note that self-calibration addresses the problem of keeping the Si5351a frequency on track. The WSPR protocol is much more sensitive to frequency than to time so we don't currently calibrate and correct the ATmega328p internal clock. It is reset regularly from GPS-data so it could be considered as loosely GPS-disciplined.

FREQUENCY DIVERSITY (AKA QRM AVOIDANCE)

On a per band basis WSPR transmissions are confined to a 200 Hz wide window. WSJT-X decodes about 10Hz beyond the top and bottom end of that so practically the decode window is more like 220 Hz wide. Many WSPR users employ WSJT-X in WSPR mode for both transmission and reception and most do not change default transmit audio offset of

1500 Hz which puts them right in the middle of the 200 Hz window (we know this from analyzing data from WSPRnet.org). The chances of our beacon being decoded improves if we can pick a frequency that is little used as this will reduce the possibility of QRM. A first step is to pick a frequency slightly above or below the center of the WSPR window, this is a no-brainer. We have taken this a step further by implementing a frequency hopping scheme. This is implemented by choosing a base frequency that is 10 Hz above the bottom of the 200 Hz WSPR window (parameter **BEACON_FREQ_HZ** in OrionXConfig.h) and adding a pseudo-random number in the range of 0 to 180 to that base frequency to determine the transmit frequency on each WSPR transmission. By generating a random transmit frequency, statistically we improve our chances of avoiding QRM. With self-calibration capability built-in we can be assured that so long as CALIBRATION is successful that we should be within one or two Hertz of our desired transmit frequency. Without CALIBRATION this scheme becomes a bit riskier as there is a chance that we could end up hopping outside the WSPR window. For that reason, if for any reason CALIBRATION fails, we revert to using a fixed frequency determined by the parameter **FIXED_BEACON_FREQ_HZ** (OrionXConfig.h). Once CALIBRATION has once again measures a generated calibration frequency equal to **SI5351_CAL_TARGET_FREQ** (OrionBoardConfig.h), frequency hopping resumes. This is explained more fully in the section titled GPS LOS/AOS DETECTION AND HANDLING.

GPS LOS/AOS DETECTION AND HANDLING

We know from Telemetry data from previous pico-balloon flights that a loss of signal (LOS) from GPS can be expected occasionally. Because GPS is so critical for position information, Si5351a self-calibration and timing for the WSPR transmission, special handling is required if WSPR transmissions are to continue during a GPS LOS scenario.

GPS LOS Detection

Before we can take steps to deal with the loss of GPS signal, we must first determine how to detect it. This is one of the functions of CALIBRATION. Because self-calibration relies on the 1PPS signal from the GPS for use as a time base for the software implemented frequency counter at the core of the calibration capability, we can take advantage of this to determine when we have a problem with GPS reception. The 1PPS signal is only generated when the GPS on-board receiver has a 3D-fix so we can equate missing 1PPS to GPS LOS.

GPS LOS Handling

With GPS LOS we lose current position information and we also cannot reset the Orion system clock, so the internal software clock will drift over time. The solution to the lack of current position information is to simply use the last valid position data that was received from the GPS to generate telemetry and continue to transmit WSPR. The philosophy is that slightly stale data is better than no data at all. WSPR transmissions within about plus or minus two seconds of UTC time can still be decoded and it is expected that the drift

rate on the Orion system clock will be small enough that Orion can continue to beacon on WSPR without timing issues for several hours in a GPS LOS scenario.

The loss of the 1PPS signal also causes an issue for CALIBRATION. To avoid getting stuck, waiting on 1PPS signal when there is a GPS LOS scenario, Orion utilizes a couple of different guard timers that prevent the code from waiting indefinitely.

In OrionXConfig.h :

#define CALIBRATION_GUARD_TMO_MS 90000 // Guard Timeout value for 1.5 minutes (60,000 ms / minute x 1.5)

#define INITIAL_CALIBRATION_GUARD_TMO_MS 1200000 // Guard Timeout value for 20 minutes (60,000 ms / minute x 20)

These determine how long the software will wait for 1PPs before failing the calibration cycle. As a precaution on calibration failure we disable FREQUENCY DIVERSITY (aka QRM AVOIDANCE) until there are two back-to-back successful calibration cycles. This is to avoid off-frequency WSPR transmission. Once we have failed a calibration cycle, we start a second guard time to keep track of the total duration of the GPS LOS condition:

#define GPS_LOS_GUARD_TMO_MS 1800000 // Guard Timeout value for 30 minutes (60,000 ms / minute x 30)

Expiration of **GPS_LOS_GUARD_TMO_MS** causes Orion to abandon the WSPR transmission cycle and to switch to a QRSS beaconing mode, where it will stay until self-calibration completes successfully. This mode handles both short term GPS LOS as well as the complete loss of GPS capability during flight.

GPS AOS Handling

GPS Acquisition of signal (AOS) is also detected by CALIBRATION. Once the 1PPS signal resumes we have a 3D-fix and the next calibration cycle will then pass, triggering normal WSPR Transmission. FREQUENCY DIVERSITY is re-enabled once the measured frequency generated by the CALIBRATION clock is equal to **SI5351_CAL_TARGET_FREQ** (OrionBoardConfig.h).

FALLBACK QRSS BEACONING

As mentioned above, after detecting GPS LOS and having **GPS_LOS_GUARD_TMO_MS** expire, Orion will transition to a QRSS beacon mode where it alternates between sending **QRSS_MESSAGE** (OrionQrss.h) and attempting a normal self-calibration. It will remain in this loop until a self-calibration cycle succeeds, at which point it will revert to the normal WSPR transmission cycle. The QRSS transmission uses FSKCW10. This is a slow speed CW mode (10 seconds per DIT) where the DAHs are shifted in frequency by

QRSS_BEACON_FSK_OFFSET_HZ (nominally less than +5Hz) at a fixed frequency set by **QRSS_BEACON_FREQ_HZ** (both parameters are #defined in OrionQrss.h).

Normally the total loss of GPS functionality during a flight would be catastrophic as the GPS is key in providing position and altitude information as well as both timing and frequency calibration for the WSPR transmissions. Without a GPS we can't send WSPR for very long as we have stale position information and must rely on the accuracy of the uncorrected software system clock. This was the motivation for implementing a fallback QRSS beacon to send at least a callsign which then may be received by any number of QRSS Grabber stations located worldwide. Without the GPS we don't have position information, but we can at least indicate that the beacon is still functional, which is better than going silent.

We anticipate that normal GPS LOS conditions will be short lived and should not normally result in the switch to QRSS Beacon Mode.

SERIAL DEBUG MONITOR

Rather than littering the code with debug print statements, it was decided early on in development to implement a serial debug monitor. This is a simplistic "shell" that implements one-character commands which are primarily used for controlling the granularity of debug output to the debug serial port. Also implemented within this framework is means to generate software error logs (SWERRs). SWERRs are always output when generated and are not filtered by serial monitor settings.

Initialising Orion Serial Monitor.... Orion firmware version: v0.27b - STELLA 9.1

cmds: v = f/w version, d = debug trace on/off, l = TX log on/off, i= info on/off, q = qrm avoidance on/off, ? = cmd list

>

The meaning of these commands can be easily understood by playing with them. Most simple toggle logging parameters off and on, varying the amount of debug and status information that is output to the serial monitor. It is worth mentioning that as Orion is not an RTOS (real-time operating system) so we don't support concurrency. If Orion is busy doing self-calibration or transmitting, then it is not looking for input from the serial port, so the monitor is non-responsive. There is a short window after self-calibration and before the start of WSPR Transmissions where the monitor will respond to commands. This is quite restrictive but that's the way it is.

Enabling debug trace will output a trace of the Orion State Machine. This output is detailed enough to allow you to follow along in the Orion State Machine Diagram which can be found in the source code folder titled OrionFSMDocs in GIThub. It is helpful for

both debugging as well as understanding the inner workings of the State Machine (OrionStateMachine.cpp) should you wish to modify it.

The initial startup values (ON/OFF) controlling Info Logs, Transmit logs and Debug logging are controlled by parameters defined in OrionXConfig.h. See the section of this document titled CONFIGURING ORION for details.

STARTUP/SHUTDOWN VOLTAGE

Operating Voltage

This is a somewhat experimental feature used to limit the amount of processor activity post Power-On-Reset to allow the voltage to rise and stabilize before attempting to engage in WSPR transmissions. We envision that this could be utilized when using a battery along with the solar cells. The idea is to allow the battery to charge before placing too much of a load on it. On startup, if **VCC_SAMPLING_SUPPORTED** (OrionBoardConfig.h) is defined as **true** then VCC is sampled and if it is less than the value defined by **OPERATING_VOLTAGE_Vx10** (OrionXConfig.h) the processor will be powered down in 8 second intervals over a 10 minutes duration, before the voltage is sampled again. After each eight second power-down the processor is awakened by the hardware watchdog. When using a K1FM board it is possible to keep both the Si5351a and GPS powered down until the Operating Voltage is reached, allowing the battery to charge more quickly due to reduced load. As protection against some unforeseen issue with sampling VCC, there is a Guard timer that is used to limit the time spent waiting for the sampled voltage to reach the value defined by **OPERATING_VOLTAGE_Vx10.** The parameter **OPERATING_VOLTAGE_GUARD_TMO_MS** (OrionXConfig.h) controls the duration of this Guard timer. Its expiry will force the resumption of startup and return to normal beacon operation, preventing Orion from getting stuck.

The usefulness of this feature is questionable if you are just using solar power (no battery) or a combination of solar power and a capacitor to power the board. If the parameter DELAY_STARTUP_ON_OP_VOLTAGE (OrionBoardConfig.h) is true this feature is enabled. If it is false then the feature is disabled.

Shutdown Voltage

Similar to OPERATING VOLTAGE, the purpose of SHUTDOWN VOLTAGE is to trigger Orion to stop transmitting when the sampled VCC drops below a preset level, as determined by the parameter **SHUTDOWN_VOLTAGE_Vx10** (OrionXConfig.h)**.** This is an attempt to reduce any glitching that might during beacon operation late in the day, when VCC has dropped to a level where the beacon is barely able to sustain operation. VCC is sampled during the Telemetry phase just prior to the start of a Primary WSPR transmission. At this point the sampled VCC value is compared to **SHUTDOWN_VOLTAGE_Vx10.** If it is at or below this level it triggers the generation of a **LOW_VOLTAGE_EV** to the state machine which in turn generates an

INITIATE_SHUTDOWN_ACTION and a transition to **SHUTDOWN_ST**. At this point Orion goes into a tight infinite loop waiting on a Power-ON Reset for recovery. If you are using a low drop-out voltage regulator on the board then the value of **SHUTDOWN_VOLTAGE_Vx10** should represent a voltage somewhere just above the drop-out voltage of the regulator. This may require some experimentation.

If you want to disable this then ensure that the parameter **SHUTDOWN_ON_LOW_VOLTAGE** (OrionBoardConfig.h) is false**.**

Orion Telemetry – Encoding and Decoding

TELEMETRY – K.I.S.S.

Orion uses a simplistic telemetry scheme that we refer to as K.I.S.S. It utilizes the dBm field in the primary and secondary WSPR messages to encode telemetry data, replacing the transmit power. This is a simplistic scheme that has some limitations but provides enough resolution for our current telemetry needs.

Telemetry – Position (POS)

The GPS position data is used to calculate a six-character Maidenhead Grid Square. The first four characters of the grid are encoded into both the primary and secondary WSPR transmissions using the GRID field of the message, as expected. The last two characters (5th and 6th) we encode into the dBm field of the primary WSPR message only, according to the scheme show in Figure 2.

Figure 2 – 5th and 6th Grid Square Character encoding into primary WSPR message dBm field.

A normal Maidenhead Grid is subdivided into clusters of sub-squares (differentiated by various colored boxes shown in Figure 2) that we refer to as GTC-squares. A GTC-square can be referenced by an x,y pair represented by 0, 1, 2, 3, 4, 5 on the x-axis (grid character 5 = sub-square longitude) and 0, 3, 7 on the y-axis (grid character 6 = sub-square latitude). Note that even though the WSPR dBm field may contain values in the range of 0 to 60 there are only 19 possible combinations that are legal. The last digit of the dBm value is restricted to the values of 0, 3 and 7 by the WSPR coding rules which are strongly enforced by WSJT-X upon reception of a WSPR message (we did try to be sneaky and use other values but they don't work).

For example, to encode the six-character grid square EM76vr (the sub-square in the lower left corner of the upper right green GTC-square in Figure 2) the EM76 would go into the GRID field of the primary WSPR message and we would encode the last two characters (vr) as 57 in the primary WSPR message dBm field. The 5 comes from the rightmost position on the x-axis and the 7 from the upper most value on the y-axis.

A four-character grid square encompasses one degree of latitude by two degrees of longitude or approximately 60 nautical miles by 120 nautical miles at the equator. This scheme using GTC-squares increases the resolution to 1/3 of degree, or approximately 20 nautical miles for latitude, by about ½ of a degree or 30 nautical miles for longitude.

When decoding we lose resolution, as the best we can do is to identify the GTC-square containing the original Maidenhead sub-square, but it is felt that 20 nautical miles by 30 nautical miles is sufficiently detailed for position reporting for a balloon.

Telemetry – Altitude (ALT)

The altitude data received from the GPS is transmitted in the secondary WSPR message sent at hh:02:01, hh:22:01, and hh:42:01, as shown in Figure 1 (Orion Beacon Schedule). Because the dBm field in the WSPR message only supports 19 values we encode the altitude as a series of altitude ranges expressed in meters above ground level (AGL) as shown in Table 1.

Encoded value (WSPR dBm Field)	Altitude Range (m)
0	< 500
3	500-999
7	100-1499
10	1500-1999
13	2000-2499
17	2500-2999
20	3000-3999
23	4000-4999
27	5000-5999
30	6000-6999
33	7000-7999
37	8000-8499

40	8500-8999
43	9000-9499
47	9500-9999
50	10000-10499
53	10500-10999
57	11000-11499
60	>= 15000

Table 1 - Encoding of Altitude into secondary WSPR message dBm field

We take advantage of the precise timing of WSPR to send different telemetry data via the dBm field of the secondary WSPR message, according to the time of transmission, thus effectively multiplexing the data in the time domain.

Telemetry – Voltage (VOLT)

The VCC voltage sampled by the processor is similarly encoded in the dBm field for secondary WSPR transmissions that begin at hh:12:01 and hh:52:01. The telemetry covers a range of sampled voltages from 3.3v to 5.0 in increments of 0.1 volts. An encoded value of 60 represents a measured voltage of greater than or equal to five volts. A voltage of less that 3.3v is represented by 0. The encoding details are shown in Table 2 (Encoding of voltage into secondary WSPR message dBm field).

Encoded value (WSPR dBm Field)	Voltage (V)
0	< 3.3
3	3.3
7	3.4
10	3.5
13	3.6
17	3.7
20	3.8
23	3.9
27	4.0
30	4.1
33	4.2
37	4.3
40	4.4
43	4.5
47	4.6
50	4.7
53	4.8
57	4.9
60	>= 5.0

Table 2 – Encoding of Voltage into secondary WSPR message dBm field

Telemetry – Temperature (TEMP)

Lastly, temperature data sampled from either the ATmega328p internal temperature sensor or an external temperature sensor is encoded and transmitted in the secondary WSPR message at hh:32:01. Encoding details are in Table 3 (Encoding of Temperature into secondary WSPR message dBm field). This scheme covers a temperature range of 100 degrees Celsius to less than minus 50 degrees Celsius with a typical resolution of five degrees Celsius.

Encoded value (WSPR dBm Field)	Temperature Range (degrees C)
0	35 to 100
3	30 to 34
7	25 to 29
10	20 to 24
13	15 to 19
17	10 to 14
20	5 to 9
23	0 to 4
27	-5 to -1
30	-10 to -6
33	-15 to -11
37	-20 to -16
40	-25 to -21
43	-30 to -26
47	-35 to -31
50	-40 to -36
53	-45 to -41
57	-50 to -46
60	< -50

Table 3 – Encoding of Temperature into secondary WSPR message dBm field

Telemetry - Summary

This simple scheme for encoding telemetry data allows us to transmit position information every ten minutes via the primary WSPR message. The time division multiplexing of data in the dBm field of the secondary WSPR messages allows the

transmission of altitude data three times per hour, voltage twice per hour and temperature once per hour.

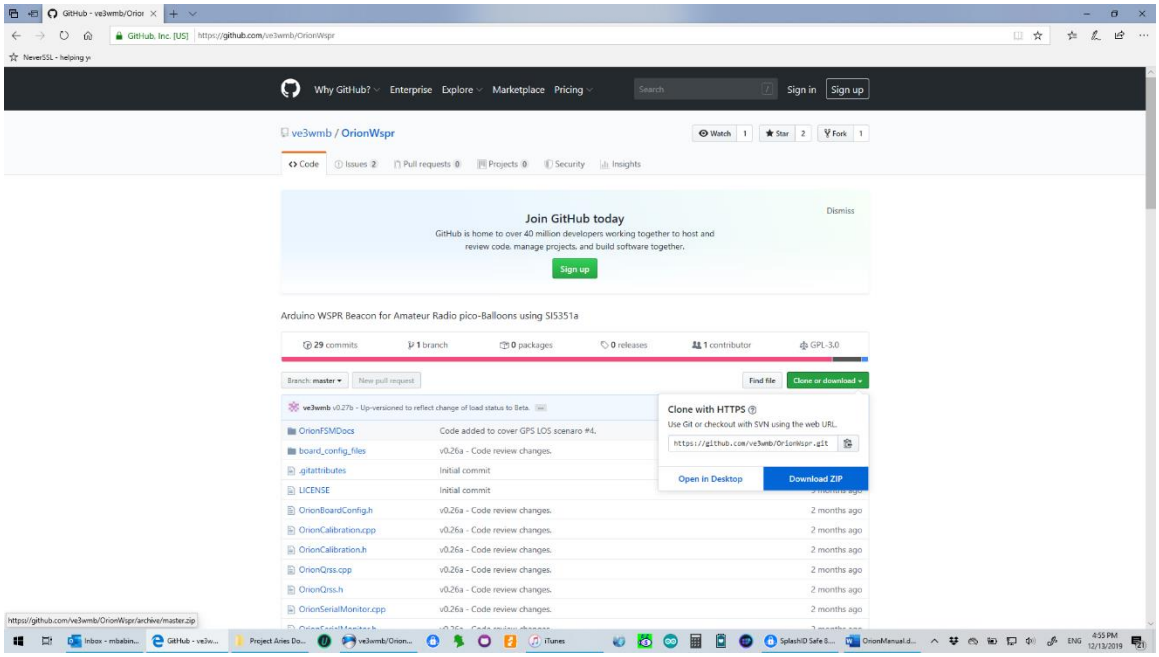
Installing Orion and the required Arduino Libraries

DOWNLOADING AND INSTALLING ORION SOURCE CODE

Orion source is available on GITHUB : https://github.com/ve3wmb/OrionWspr

To download Orion :

- Go to the above link and click on the green *"Clone or Download"* button and select *"Download ZIP"* as shown in the figure below. Save the file.
- Right-click on the zipped file (OrionWspr-master.zip) with your mouse (on Windows) and select "Extract all" from the resulting menu. This will create an extracted folder titled OrionWspr-master.
- Next rename this folder to OrionWspr. The folder name must match the .ino file contained within the folder (i.e OrionWspr.ino), without the suffix.
- Finally copy your OrionWspr folder into the Arduino folder on your computer (this assumes that you have already installed the Arduino IDE).



DOWNLOADING AND INSTALLING THE REQUIRED ARDUINO LIBRARIES

Orion uses several non-standard Arduino libraries that provide additional functionality needed for the application. You must download and install these required libraries in order to compile Orion and generate a binary file that the Arduino IDE can load into the ATmega328p.

The list of libraries needed, with GITHUB download links, are as follows :

- JTEncode - https://github.com/etherkit/JTEncode - WSPR Encoding functions.

- Time - (https://github.com/PaulStoffregen/Time) - This provides a Unix-like System Time capability

- SoftI2CMaster (https://github.com/felias-fogg/SoftI2CMaster) Software I²C with SoftWire wrapper that mimics the standard Wire.h library.

- NeoGps - (https://github.com/SlashDevin/NeoGPS) - NMEA and uBlox GPS parser.

- NeoSWSerial - (https://github.com/SlashDevin/NeoSWSerial - Software Serial port communications library.

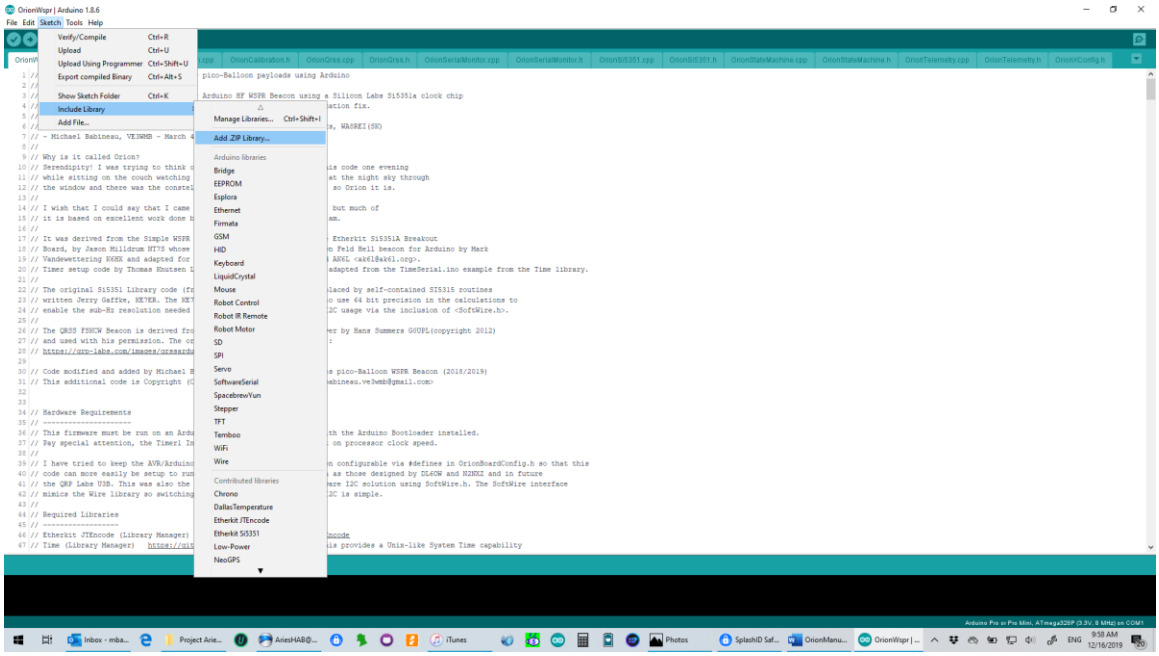
- Chrono (https://github.com/SofaPirate/Chrono) - Simple Chronometer Library

- LowPower (https://github.com/rocketscream/Low-Power) - Lightweight Low Power Library to enable processor power management.

As with Orion, go to the project page for each library in turn and download the Zip file. Next do the following:

- Start up the Arduino IDE
- Select Sketch -> Include Library -> Add .Zip library ... (as shown in the figure below)
- Navigate to the ZIP file containing the library you want to add and click “Open”

- Repeat for all required libraries



Configuring Libraries

Some of the installed libraries require some configuration. If you skip this step, you will get compile or link errors.

NeoGPS

NeoGPS must be configured to use a Nominal Configuration to be compatible with the needs of Orion. This includes support for date, time, lat/long, altitude, speed, heading, number of satellites, HDOP, GPRMC and GPGGA messages. This is the default configuration of the library as downloaded so no additional changes are needed.

NeoSWSerial

- On Windows using File Explorer navigate to your Arduino\libraries\NeoSWSerial-master\src folder and edit NeoSWSerial.h using a text editor (Notepad will work fine but I prefer Notepad++),
- Uncomment the following line which is near the end of the file, save and recompile.

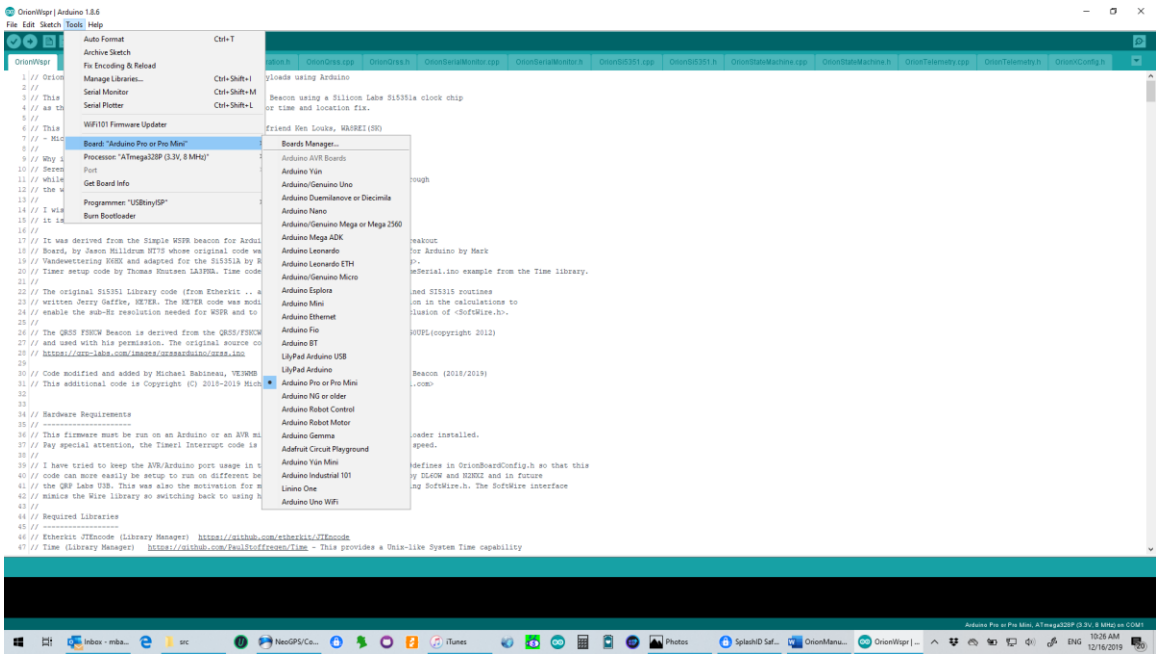
// *#define* **NEOSWSERIAL_EXTERNAL_PCINT** // uncomment to use your own PCINT ISRs

It should look like this without the leading "//".

#define **NEOSWSERIAL_EXTERNAL_PCINT** // uncomment to use your own PCINT ISRs

NeoSWSerial assumes by default that applications are not using PinChange Interrupts in so it defines the ISRs for all ports which will conflict with the Orion definitions.

Before compiling don't forget to set your boardtype within the Arduino IDE as shown (this assumes that you are running your board at 3.3v/8Mhz).



The Orion code should now compile without errors.

Configuring Orion

User modifiable configuration parameters for Orion are contained within the following three files, which appear as separate tabs when the OrionWspr.ino file is opened in the Arduino IDE :

- OrionXConfig.h
- OrionQrssConfig.h
- OrionBoardConfig.h

These files will likely need to be changed to adapt Orion to your needs.

OrionXConfig.h

OrionXConfig.h contains some common type definitions and the definition of parameters that configure the WSPR Beacon but are not board specific.

The parameters and their description are as follows:

- *#define* **DEBUG_LOG_INITIAL OFF** - This value (OFF or ON) defines the initial setting for the output of debug logs. Debug logging produces detailed calibration output as well as detailed pre and post state machine operations. Debug logging can be toggled in the serial monitor via the "**d**" command. This parameter sets the initial value to be applied at system startup. **This value should be OFF for flight.**

- *#define* **TX_LOG_INITIAL OFF** - This value (OFF or ON) defines the initial setting for the output of transmit and telemetry logs. Transmit logging produces summary output for each transmission cycle and telemetry gathering. Transmit logging can be toggled in the serial monitor via the "**l**" command. This parameter just sets the initial value to be applied at system startup. **This value should be OFF for flight.**

- *#define* **INFO_LOG_INITIAL OFF -** This value (**OFF** or **ON**) defines the initial setting for the output of information logs. Info logging produces summary output for each calibration cycle as well as some supplemental information on the setting of the Orion system clock and other system operations. Info logging can be toggled in the serial monitor via the "**i**" command. This parameter just sets the initial value to be applied at system startup. **This value should be OFF for flight.**

- *#define* **ORION_FW_VERSION "v0.27b"** - This defines the Orion software version. It worth noting that this string is output in the debug serial monitor via the "**v**" command. It is a useful way to ensure that your board has the correct load and board configuration file.

>Initialising Orion Serial Monitor.... Orion firmware version: v0.27b - STELLA 9.1

- *#define* **BEACON_FREQ_HZ 14097010UL** - The base frequency in Hertz for use when FREQUENCY DIVERSITY is enabled. The actual transmit frequency is **BEACON_FREQ_HZ** + a random offset (0-180 Hz). This value should be 10 Hz above the bottom of the WSPR window for the chosen band.

- *#define* **FIXED_BEACON_FREQ_HZ 14097070UL** - The beacon frequency in Hertz for use when FREQUENCY DIVERSITY is disabled. A frequency slightly offset (~ +/- 20Hz) from the center of the 200 Hz WSPR window for the band of choice is recommended.

- *#define* **PARK_FREQ_HZ 108000000ULL** - Use this frequency on the SI5351A_PARK_CLK_NUM to keep the SI5351a warm to avoid thermal drift during WSPR transmissions. This value must be less than 109 Mhz (This 109 Mhz limitation is imposed because of the fixed VCO configuration used). I suggest using the default value.

Configuration parameters for Primary WSPR Message (i.e. Callsign, 4 character grid square and power out in dBm)

- *#define* **BEACON_CALLSIGN_6CHAR "VE3WMB"** - Your WSPR beacon Callsign, maximum of 6 characters.

- *#define* **BEACON_GRID_SQ_4CHAR "AA01"** - A hardcoded 4 character Maidenehead Grid Square. This will be overwritten with a GPS derived Grid calculated from current GPS latitude and longitude data, so this is only used as an initial value.

- *#define* **BEACON_TX_PWR_DBM 7** - Beacon Power Output in dBm (5mW = 7dBm) . Initial value. This gets overwritten by telemetry.

- *#define* **OPERATING_VOLTAGE_Vx10 0** - This is the sampled VCC value (times 10), required to initiate beacon operation (i.e 33 means 3.3v) at startup . This is only applicable if the OPERATING VOLTAGE feature is enabled by setting **DELAY_STARTUP_ON_OP_VOLTAGE** to true in OrionBoardConfig.h.

- *#define* **SHUTDOWN_VOLTAGE_Vx10 55** - Sampled VCC value, times ten (i.e. 20 means 2.0 volts). Readings below this value will initiate the transition to SHUTDOWN_ST. This is only applicable if the SHUTDOWN VOLTAGE feature is enabled by setting **SHUTDOWN_ON_LOW_VOLTAGE** to true in OrionBoardConfig.h.

- *#define* **OPERATING_VOLTAGE_GUARD_TMO_MS 3600000** - Guard Timeout value for 1 hour (60,000 ms / minute x 60). This prevents Orion from getting stuck forever waiting on the operating voltage.

- *#define* **CALIBRATION_GUARD_TMO_MS 90000** - Guard Timeout value for 1.5 minutes (60,000 ms / minute x 1.5) . This ensures that calibration is skipped (failed) if it takes more than 1.5 minutes for 1PPS signal to appear at the start of a calibration cycle. One and a half minutes is as long as we can wait for calibration to start without jeopardizing the timing for the start of the next transmission cycle.

- *#define* **INITIAL_CALIBRATION_GUARD_TMO_MS 1200000** - Guard Timeout value for 20 minutes (60,000 ms / minute x 20). This is the same idea as **CALIBRATION_GUARD_TMO_MS,** except that it applies to the first calibration cycle after power-on reset.

- *#define* **GPS_LOS_GUARD_TMO_MS 1800000** - Guard Timeout value for 30 minutes (60,000 ms / minute x 30). This timer is started upon the detection of a GPS LOS scenario (calibration fail due to no GPS 1PPS signal). When it expires it triggers the switch to QRSS Beaconing.

OrionQrssConfig.h

This file contains configuration parameters specific to the QRSS Beacon that are not board specific.

- *#define* **QRSS_BEACON_FREQ_HZ 14096810UL** - This is the transmission frequency for the FSKCW10 QRSS Beacon. Typically, the QRSS-Window is below the frequency range for WSPR on most bands.

- *#define* **QRSS_BEACON_FSK_OFFSET_HZ 4** - DITS will be transmitted at **QRSS_BEACON_FREQ_HZ** and DAHS at **QRSS_BEACON_FSK_OFFSET_HZ** higher for FSKCW10. This value should not be greater than 5 (i.e. 5 Hz). The default value of 4 Hertz is recommended.

- *#define* **QRSS_MESSAGE " VE3WMB "** - Put your callsign here, in capital letters. We recommend two spaces before and one after your callsign this will give a cleaner trace on QRSS Grabbers. Note that at QRSS10 speeds (10 second DITs) that this will take approximately 30 minutes to transmit a typical callsign. You might choose to transmit an abbreviated callsign for QRSS, this is the reason that this parameter is separate from the WSPR callsign parameter **BEACON_CALLSIGN_6CHAR**.

- *#define* **POST_QRSS_TX_DELAY_MS 180000** - Delay after QRSS Transmission completion before attempting self calibration (3 minutes). Since we wait about 2 minutes when attempting, calibration this gives about 5 minutes between QRSS transmissions.

- *#define* **FSK_HIGH QRSS_ BEACON_FSK_OFFSET_HZ** – DAH frequency offset from QRSS transmit frequency**. Don't change this.**

- *#define* **FSK_LOW 0** – Dit frequency offset from QRSS transmit frequency**. Don't change this.**

OrionBoardConfig.h – Board Specific parameter definitions

It is worth noting that the folder Arduino\OrionWspr\board_config_files contains some sample OrionBoardConfig,h files for DL6OW STELLA boards (U3S clone) and the K1FM V1.3 boards. If you are using one of these boards or a rough equivalent then copy the relevant file from this directory to Arduino\OrionWSPR, delete the existing OrionBoardConfig.h and then rename the file you have copied to OrionBoardConfig.h, thus replacing the original with the copy.

Below is the description of the configuration parameters :

Boardname

- *#define* **BOARDNAME " - STELLA 9.1"** - This string is output along with the software version using the "**v**" command in the serial monitor. It is useful if you have multiple versions of OrionBoardConfig,h files as it will help to confirm that you are using the correct one once the sketch is uploaded to the ATmeg328p.

>Initialising Orion Serial Monitor.... Orion firmware version: v0.27b - STELLA 9.1

Serial Port Configuration

The following *#defines* select code optionality via conditional compilation:

- *#define* **GPS_USES_HW_SERIAL** - The GPS communicates with the processor is via Hardware serial. Comment this out if the ATmega328p communicates with GPS via Software Serial.

- *#define* **DEBUG_USES_SW_SERIAL** - Debug serial uses a software serial port. Comment this out if the Serial Debug Monitor uses Hardware Serial.

- *#define* **GPS_SERIAL_BAUD 9600** - Baud rate for the GPS Serial port

- *#define* **MONITOR_SERIAL_BAUD 9600** - Baud rate for Orion Serial Monitor

Pins used for software Serial communications.

The following two definitions select the Arduino pins used for software Serial communications.

The assumption is that if software serial is used for communicating with the GPS, that hardware serial is used for the debug monitor, or vice versa. One of the two must use hardware serial, the other software serial.

Note that the choice of pins will impact the setup of the PinChange Interrupts for the NeoSWSerial library. See OrionSerialMonitor.cpp.

- *#define* **SOFT_SERIAL_RX_PIN 12** // MISO of six pin ICSP header on DL6OW boards

- *#define* **SOFT_SERIAL_TX_PIN 11** // MOSI of six pin ICSP header on DL6OW board

PIN definitions for Si5351a software I2C communication

- *#define* **SI5351A_USES_SOFTWARE_I2C** - Processor talks to the Si5351a using software I2C. Comment this out if the ATMEGA328p communicates with the Si5351a via Hardware I2C (i.e via dedicated SDA/SCL pins).

Ignore these if using Hardware I2C with Wire Library to communicate with the Si5351a. These example values are assuming Hardware Pin assignments compatible with the QRP Labs U3S & U3S-clones.

- *#define* **SCL_PIN 1** //PB1
- *#define* **SCL_PORT PORTB**
- *#define* **SDA_PIN 2** //PD2
- *#define* **SDA_PORT PORTD**

Hardware Configuration for CALIBRATION

- *#define* **SI5351_SELF_CALIBRATION_SUPPORTED true** - Set this parameter to false if no self calibration. It requires an unused Si5351 CLK output fed back to Arduino D5 and 1PPS fed to a digital input pin on the ATmega328p. If this is not the case then the parameter must be false.

- ***#define* GPS_PPS_ON_D2_OR_D3** - GPS 1PPS connects to D2 or D3 and thus can use an External Interrupts. Otherwise you must comment this line out and Orion will utilize PinChange Interrupts on another digital input pin. (Either works fine).

- *#define* **CAL_FREQ_IN_PIN 5** - This is the processor pin that the Si5351a clock output designated by **SI5351A_CAL_CLK_NUM** connects to**.** This must be D5 (5) otherwise the CALIBRATION feature will not work. If not D5 then **SI5351_SELF_CALIBRATION_SUPPORTED** must be set to **false.**

GPS Pin Configurations

The following example shows the use of D3 and External Interrupts, as with the K1FM V1.3 Board.

- *#define* **GPS_PPS_PIN 3** - This must be either 2 or 3 (i.e. D2 or D3) to use external interrupts, otherwise Orion will use a PinChangeInterrupt for PPS.

The following shows the configuration used for U3S-clone boards like the Dl6OW Stella boards which use pin A5 and PinChange Interrupt 13.

- *#define* **GPS_PPS_PIN A5** - DL6OW STELLA boards and other U3S Clones use A5/ADC5 (physical pin #28) for PPS. This uses PCINT13

Temperature Sensor Configuration

If one of the following two are defined, this sensor data will be used for temperature telemetry otherwise we default to using the internal temperature sensor in the Atmega328p. At most, only one of these two lines should be uncommented.

- *#define* **DS1820_TEMP_SENSOR_PRESENT** - Dallas Semiconductor DS18020 external One-wire sensor is present.

- *#if defined* (**DS1820_TEMP_SENSOR_PRESENT**)

    #define **ONE_WIRE_BUS A0** // Dallas Temperature One-Wire Sensor change this to match PIN usage

    #endif

    If you are using the DS1820 then you must designate which Arduino Pin **ONE_WIRE_BUS** is connected to.

- *#define* **TMP36_TEMP_SENSOR_PRESENT** - TMP36 - external analog sensor present.

- *#if defined* (**TMP36_TEMP_SENSOR_PRESENT**)

    #define **TMP36_PIN A1** // TMP36 Temperature sensor. Change to match PIN usage.

    #endif

If you are using a TMP36 then **TMP36_PIN** must designate which Arduino pin is being used for this sensor.

Parameters for VCC Sampling

- *#define* **VCC_SAMPLING_SUPPORTED true** - Board has the capability to sample VCC on **Vpwerbus** using **VpwerDivider** as a multiplying factor. Must be either **true** or **false**.

- *#define* **Vpwerbus A3** - ADC input for Vpwrbus for measuring battery voltage.

- *#define* **VpwerDivider 1.3333** - Multiplying factor for ADC voltage divider. This value will depend on the choice of resistor values used for the divider.

- *#define* **DELAY_STARTUP_ON_LOW_VOLTAGE false** – This parameter enables or disables the STARTUP VOLTAGE feature. This is experimental, it is recommended that this should be set to false.

- *#define* **SHUTDOWN_ON_LOW_VOLTAGE false** – This parameter enables or disables the SHUTDOWN VOLTAGE feature. This is experimental, it is recommended that this should be set to false.

Si5351a Clock Configuration Parameters

- *#define* **SI5351A_PARK_CLK_NUM 1** - The Si5351a Clock Number output used to mimic the QRP Labs U3S Park feature. This should be an unterminated clk port.

- *#define* **SI5351A_CAL_CLK_NUM 2** - Calibration Clock Number. This clock output must physically terminate on processor Pin D5 for the calibration capability to work.

- *#define* **SI5351A_WSPRTX_CLK_NUM 0** - The Si5351a Clock Number output used for the WSPR Beacon Transmission and also for QRSS Beaconing.

- *#define* **SI5351A_CLK_FREQ_CORRECTION 0** - This parameter will be specific to your board and depends on the Si5351a crystal frequency. Start with zero and allow the board to run through several self-calibration cycles with debug on. Monitor the calibration logging until you see that the target frequency (3.2 Mhz) is reached during calibration and note the correction factor applied at that point. Substitute that value for this parameter, recompile and upload the new load. This provides a reasonable initial correction factor for calibration and will help the calibration algorithm converge more quickly after startup. Note that this correction factor may be a negative number.

- *#define* **SI5351BX_XTALPF 3** - 1:6pf 2:8pf 3:10pf - assuming 10 pF, otherwise change

If using 27mhz crystal for Si5351a, set XTAL=27000000, MSA=33. Then vco=891mhz

- *#define* **SI5351BX_XTAL 2500000000ULL** - Crystal freq in centi-Hertz
- *#define* **SI5351BX_MSA 35** - VCOA is at 25mhz*35 = 875mhz

Parameters dependent on Processor CPU Speed (Assumption is 8Mhz Clock)

The following parameter **WSPR_CTC** is used for Timer1 that generates a 1.46 Hz interrupt. This results in an interrupt that triggers every 682.68 milliseconds to drive the WSPR transmission.

THE CURRENT VALUE for WSPR_CTC ASSUMES AN 8 MHZ PROCESSOR CLOCK !

If you are using most Arduinos (i.e Nano, UNO etc) they use a 16 Mhz clock and the value 10672 must be substituted.

- ie. *#define* **WSPR_CTC 10672** // CTC value for WSPR on Arduino using 16 Mhz clock (i.e. Nano, Uno etc)

The formula to calculate WSPR_CTC is: 1.4648 = CPU_CLOCK_SPEED_HZ / (PRESCALE_VALUE) x (WSPR_CTC + 1)

- *#define* **WSPR_CTC 5336** - CTC value for WSPR on Arduino using an 8 Mhz clock (i.e. Arduino Pro Mini 3.3v 8 Mhz)

- *#define* **SI5351_CAL_TARGET_FREQ 320000000ULL** - This is calculated as CPU_CLOCK_SPEED_HZ / 2.5 expressed in hundredths of Hz. Assumes an 8 Mhz

clock. If using a 16 Mhz processor clock then this can safely be doubled to 64**0000000ULL** (i.e 6.4 Mhz) for better calibration accuracy.

Power Disable Feature Parameters for SI5351a and GPS

The K1FM V1.3 boards support the capability to control the supply voltage to both the Si5351 and GPS via a pair of power Mosfets. This allows Orion the option of switching these off at startup to reduce power consumption. This feature is combined with STARTUP VOLTAGE management to reduce power consumption until the operating voltage is reached. The parameters that control this capability are :

- *#define* **GPS_POWER_DISABLE_SUPPORTED** - K1FM 1.2 boards and greater support the ability to enable/disable VCC power to GPS
- *#ifdef* **GPS_POWER_DISABLE_SUPPORTED**

  #define **GPS_POWER_DISABLE_PIN** **7** // Pin D7

  // Enable PIN for GPS VCC (LOW = enabled, HIGH = disabled)

  #endif

- *#define* **SI5351_POWER_DISABLE_SUPPORTED**- K1FM 1.2 boards and greater support the ability to enable/disable VCC power to Si5351a
- *#ifdef* **SI5351_POWER_DISABLE_SUPPORTED**

  #define **TX_POWER_DISABLE_PIN** **6** // Pin D6

  // Enable PIN for Si5351a TX VCC (LOW = enabled, HIGH = disabled)

  #endif

For boards that don't support these features the parameters **GPS_POWER_DISABLE_SUPPORTED** and **SI5351_POWER_DISABLE_SUPPORTED** must be commented.

Miscellaneous Pin Configurations

- *#define* **ANALOG_PIN_FOR_RNG_SEED A1** - This is the pin used to generate the seed for the Arduino Random number generator, using an analog read. This must be a free analog pin (not connected to anything).

- *#define* **TX_LED_PRESENT** - WSPR Transmission in progress while lit
- *#define* **SYNC_LED_PRESENT** - Orion system clock synched with GPS when LED is lit.

Comment these out if not using an LED to indicate WSPR TX or GPS Time Sync. These are useful when experimenting with a solderless breadboard prototype to gain extra visibility of Beacon Operation. It is not expected that they would be used on flight hardware (i.e. during flight no one is going to see LEDs flashing).

The following defines designate which pins are used for TX and Time Synch if TX_LED_PRESENT and SYNC_LED_PRESENT are defined. They are ignored otherwise.

#define **TX_LED_PIN** **4** // TX LED on D4.

#define **SYNC_LED_PIN** **7** // LED on PIN D7 indicates GPS time synchronization.

Note the most Arduino Boards have a built-in LED that can be used for either of the above purposes referred to as LED_BUILTIN.

Appendix

RESOURCES

Open source WSPR Beacon Boards suitable for use with Orion

- DL6OW STELLA Boards – U3S Clone (available from OSHPARK) https://oshpark.com/profiles/DL6OW

- K1FM V1.3 Board (available from OSHPARK) - https://oshpark.com/shared_projects/ud3vH8MF

  Github: https://github.com/adecarolis/K1FM-Pico-Balloon

Arduino Resources

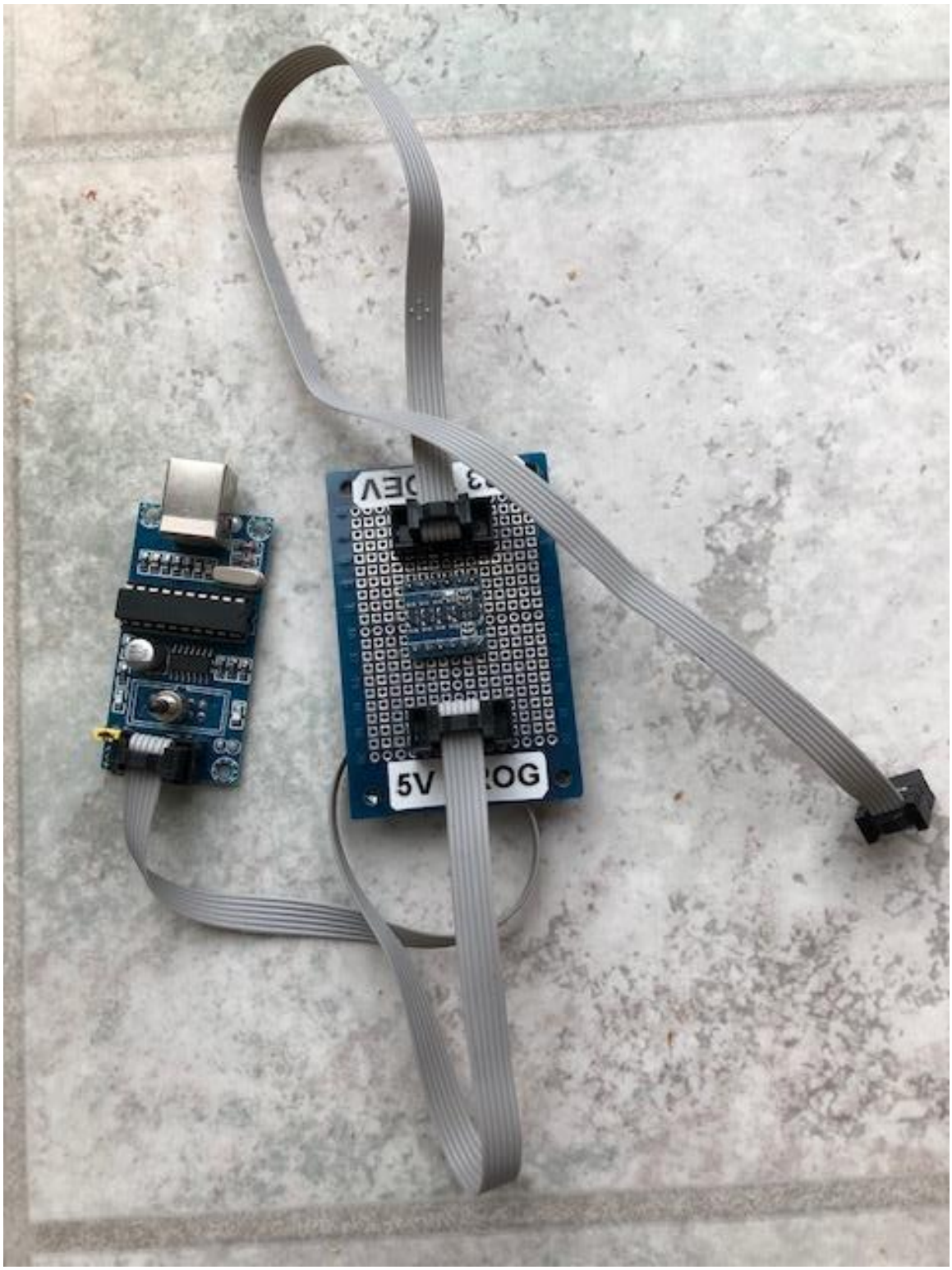
- Arduino IDE Download - https://www.arduino.cc/en/Main/Software

NOTES ON PROGRAMMING YOUR BOARD AND INSTALLING A BOOTLOADER

ICSP Programmer (for burning bootloader on ATmega328p chip and also can be used for programming)

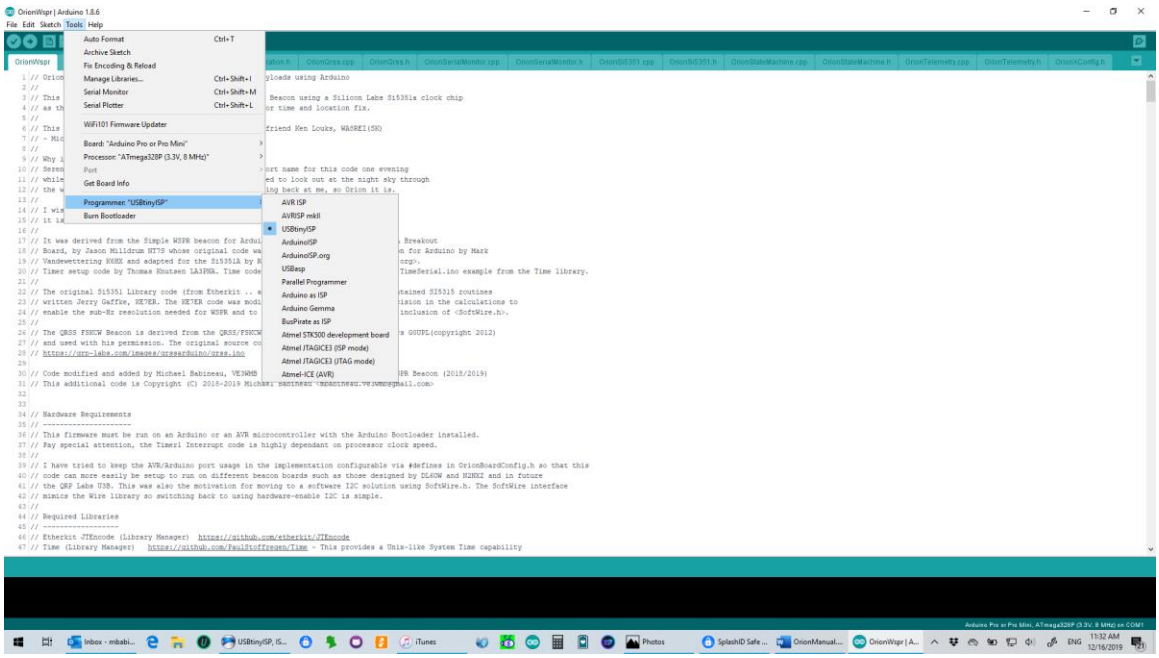
We recommend the USBTinyISP (https://learn.adafruit.com/usbtinyisp) or one of the USBTIny clones like this one : (https://www.universal-solder.ca/product/usbtinyisp-isp-avr-programmer-atmega-attiny-arduino/).

If you are running your board at 3.3v then you will need to build a 3.3v to 5V bi-directional adaptor to use this ICSP progrmmer as it is 5v only. Inexpensive 3.3v to 5v bidirectional level, multi-pin convertor boards can be wired on a prototype board to produce a home-brew solution that looks something like this (USBTinyISP on left, 3.3v adaptor board on right).

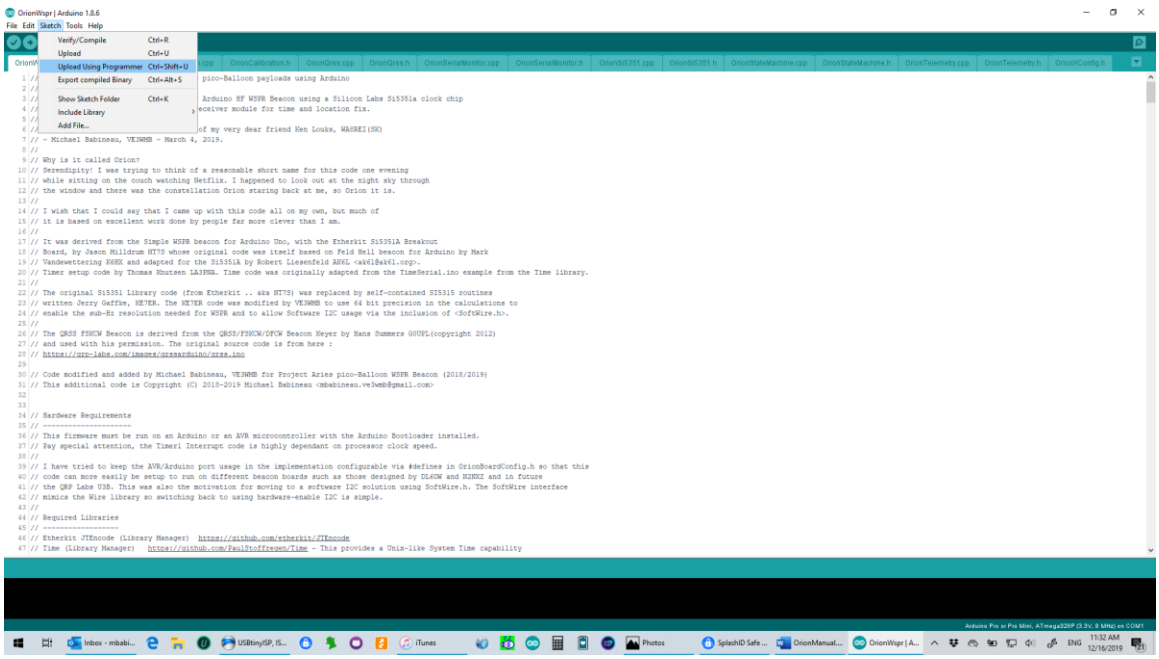


This will also allow you to reprogram the ATmega328p as many times as needed via the ICSP header, without fear of damaging other components that are designed for 3.3v only.

To program using the USBTinyISP you need to set the programmer type as follows :



When uploading code instead of selecting Upload, you use Upload using programmer.



This is by no means intended as a comprehensive description of using an ICSP programmer. The reader is referred to the following links for more details :

https://www.arduinolab.net/in-circuit-system-programming-icsp-using-usbtinyisp-and-arduino-ide/

https://www.instructables.com/id/Atmega-Programming-with-USBtinyISP-and-Arduino/

MODIFYING ORION CODE TO SUIT YOUR OWN NEEDS

Some users will no doubt want to modify Orion to meet their specific needs and under the GPL3 license this is allowed and in fact, encouraged. The easiest way to do this is to create a fork in Github as K1FM has done with his Gemini project (a derivate of Orion) at :

https://github.com/adecarolis/GeminiWspr

For those of you that want to dig deeper into the code, below is a description of what is contained in each file/tab to help you find your way around.

Orion file Descriptions

OrionWspr.ino

This is the main .ino file and as such it has the Arduino mandated *loop()* and *setup()* functions located at the end of the file to make them easier to find.

Other functionality of note is:

- *process_orion_sm_action()* – handles the actions generated by the state machine to actually do work like transmitting WSPR, doing calibration etc.
- *orion_scheduler()* – implements the schedule of generated time events.

It is also worth noting that any Telemetry related functionality that is not found in OrionTelemetry.cpp will be found in this file.

OrionBoardConfig.h

This is described in detail in the section titled Configuring Orion. In short this contains all board specific *#defines*.

OrionCalibration.h / OrionCalibration.cpp

These files contain most of the definitions and code related to the CALIBRATION feature including the Timer/Counter-1 interrupt (ISR) setup and handling. There is one exception and that is *handle_calibration_result()* and that is located in OrionWspr.ino.

OrionQrss.h / OrionQrss.cpp

As the name implies, these files contain the definitions, configuration and implementation for the FSKCW10 QRSS beacon. It is worth noting that the QRSS Beacon text used is not the WSPR callsign but defined by QRSS_MESSAGE which can be found in OrionQrss.h.

OrionSerialMonitor.h / OrionSerialMonitor.cpp

These contain the definitions and code for the serial debug monitor.

OrionSi5351.h / OrionSi5351.cpp

Definitions and functions for controlling the Si5351a.

OrionStateMachine.h / OrionStateMachine.cpp

The .h file not only contains the state definitions for the state machine but also defines the enumerated types for actions and events. The meat of the .cpp file is the function *orion_state_machine()* which as the name suggests implements the logic of the state machine.

It is worth noting that the operation of the Orion state machine, along with the interrelated events and actions are documented in a state diagram that can be found in Arduino\OrionWspr\OrionFSMDocs\CurrentOrionWsprStateMachine.pdf. The integer value following the colon in the state, action and event names represents the value in the corresponding enumerated type for a given state, action or event. Thus, that diagram can be used as an aid in making sense of the output of the serial monitor debug "d" command which produces pre and post state machine debugging logs.

OrionTelemetry.h / OrionTelemetry.cpp

These two files implement the K.I.S.S. telemetry scheme defined earlier in this document. If you are looking for a function that is telemetry related but doesn't seem to be in OrionTelemetry.cpp , try OrionWspr.ino .

OrionXConfig.h

This file contains defines that are specific to the operation of the Wspr Beacon itself, independent of what board it runs on. Since it is "included" by almost every other file in the project it also has some common type definitions that are used throughout Orion.

SOFTWARE LICENSING

GPL 3 License

publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.