Pop-Up Buoy Engineer’s Guide

(Gen 3/Rev 5.3)

Table of Contents

[1. Background 2](#_Toc525309330)

[2. Design Decisions 4](#_Toc525309331)

[2.1. Burn-Wire Release 4](#_Toc525309332)

[2.2. Sensors 5](#_Toc525309333)

[2.2.1. Temperature Sensor 5](#_Toc525309334)

[2.2.2. PAR Sensor 5](#_Toc525309335)

[2.2.3. Pressure Sensors 6](#_Toc525309336)

[2.2.4. GPS 7](#_Toc525309337)

[2.2.5. Iridium Module 7](#_Toc525309338)

[2.3. Mechanical Design 8](#_Toc525309339)

[2.3.1. Trawl Float 8](#_Toc525309340)

[2.3.2. End Cap 8](#_Toc525309341)

[2.3.3. Frame (Strong-back) 9](#_Toc525309342)

[2.3.4. Release Mechanism 9](#_Toc525309343)

[2.4. Electronics 10](#_Toc525309344)

[2.4.1. Arduino MEGA 10](#_Toc525309345)

[2.4.2. RTC 10](#_Toc525309346)

[2.4.3. Batteries 10](#_Toc525309347)

[2.4.4. Temperature Circuit 11](#_Toc525309348)

[3. Circuit Description 12](#_Toc525309349)

[3.1. Arduino Mega 12](#_Toc525309350)

[3.2. Power Regulation 12](#_Toc525309351)

[3.3. MicroSD Card, Accelerometer, and RTC 12](#_Toc525309352)

[3.4. Temperature Measurement 13](#_Toc525309353)

[3.5. PAR Measurement 14](#_Toc525309354)

[3.6. Pressure Measurement 14](#_Toc525309355)

[3.7. GPS 14](#_Toc525309356)

[3.8. Iridium Module (RockBLOCK Mk2) 15](#_Toc525309357)

[4. Construction and Assembly 16](#_Toc525309358)

[4.1. Temperature Sensor 16](#_Toc525309359)

[4.2. PAR Sensor 17](#_Toc525309360)

[4.3. Pressure Sensors 18](#_Toc525309361)

[4.4. Vacuum Port 19](#_Toc525309362)

[4.5. PCB 20](#_Toc525309363)

[4.6. Electronics Stack, Iridium and GPS Antennae 20](#_Toc525309364)

[4.7. Main Battery Installation 20](#_Toc525309365)

[4.8. Frame 22](#_Toc525309366)

[5. Software Development and Programming 23](#_Toc525309367)

[5.1. Arduino 1.0.6 23](#_Toc525309368)

[5.2. Library Changes 23](#_Toc525309369)

[5.2.1. Iridium SBD 23](#_Toc525309370)

[5.2.2. Software Serial 26](#_Toc525309371)

[6. Program Flow 27](#_Toc525309372)

[6.1. General Program Flow 27](#_Toc525309373)

[6.2. Iridium Transmission 28](#_Toc525309374)

[6.3. Iridium Line Rental and Credits 28](#_Toc525309375)

[7. Programming, Calibrating, and Testing the Units 29](#_Toc525309376)

[7.1. Program (Sketch) Descriptions 29](#_Toc525309377)

[7.2. Uploading Programs (Sketches) 30](#_Toc525309378)

[7.3. Calibrating the Temperature Sensors 32](#_Toc525309379)

[7.4. Calibrating the PAR Sensor 33](#_Toc525309380)

# Background

The Pop-Up Buoy was developed as a unique product designed to take measurements while sitting under sea ice in the Arctic - to fill the gap in available technology between year-round deeper moorings and surface-based sampling in the summers. Measuring just under sea ice is of particular interest because the conditions there play a critical role in shaping one of the world’s most highly productive eco-systems. However, because of the harsh and unpredictable environment of shifting and breaking ice, actually acquiring measurements close to the ice during the Winter and Spring without losing instruments has been notoriously difficult. Because of the unique conditions, it needed to be **robust, inexpensive, and expendable, but sophisticated enough to acquire oceanographic-quality data.**

Cost was by far the driving factor in most of the design decisions and led to some atypical outcomes, such as the independent burn wire release and stripped down sensors. Since the floats only provide 1 profile of the water column, it was desirable to keep the cost around $3,000 per float. From a program-level perspective, the extremely low cost (compared to other instruments) is what makes the instrument a viable option for collecting data.

The instruments are designed to be deployed from a research vessel during the ice-free season, where they remain anchored on the bottom for many months until the surface is completely covered in sea ice. At a designated time for each device, a timed release is triggered which allows the buoys to float upward in the water column, collecting a vertical profile of the water column, until they reach the ice. The buoys remain under the ice until they are forced out by break-up and melting, transmitting their data to shore via satellite when they do arrive at the surface.

This third generation of instruments measures temperature, Sea Surface Temperature (when drifting at surface), PAR, Depth, and fluorescence (optional). It also has GPS and Iridium modules for location and transmission.

# Design Decisions

## Burn-Wire Release

## Sensors

### Temperature Sensor

.

### PAR Sensor

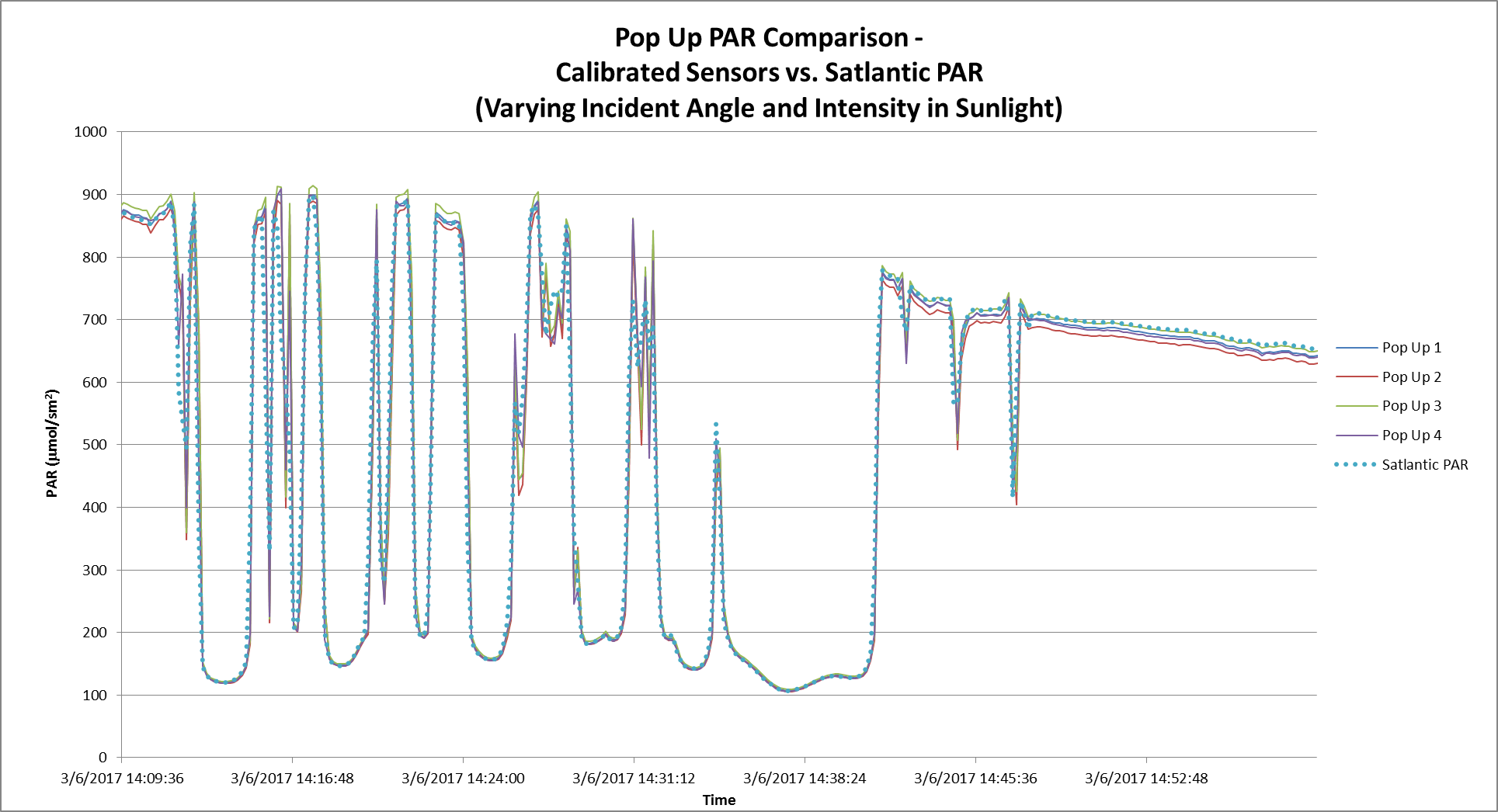
The PAR sensor used on the Pop-Up buoys is a TAG-PARQ by Skye Instruments. This is the same sensor as used on the initial prototype of the Pop-Up Buoys and integrated into the satellite tag provided by the University of St. Andrews. It was shown to have excellent response, even in very low light conditions.

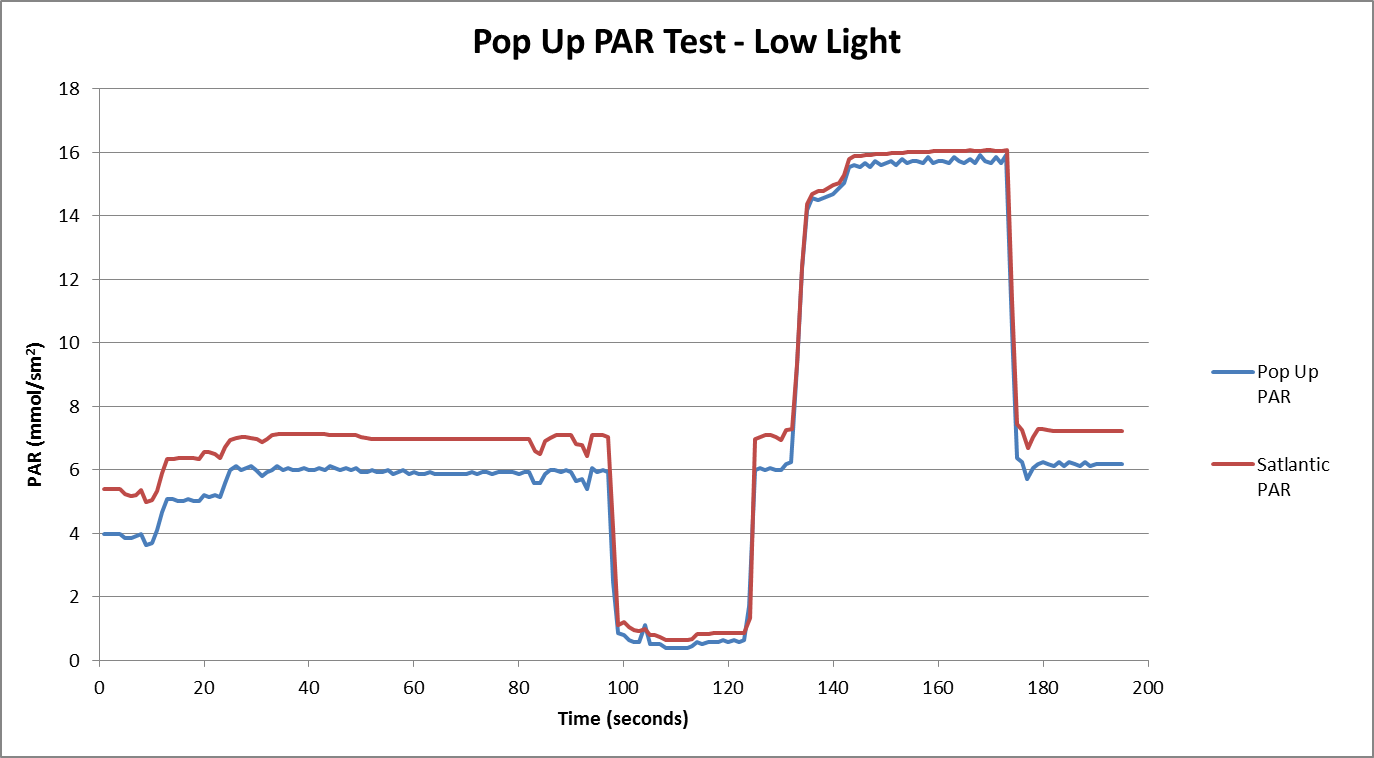
The TAG-PARQ sensor costs only $340 whereas the least expensive alternative (a LICOR PAR sensor) is around $800.

One tradeoff for this sensor is that the TAG-PARQ is a simple photodiode with an analog output in As. An amplifier circuit is built into the main electronics to properly amplify and convert the signal.

A second tradeoff for the TAG-PARQ sensor is that it is not waterproof and must be installed in a separate housing. Ideally, this should be a perfect cosine diffuser so the angle of incident light does not affect the signal. The acrylic housing used in the Pop-Up buoys is not a perfect cosine diffuser, but works well within the 3% margin of error the sensors are calibrated for. Testing showed the acrylic absorption by acrylic can be easily corrected for with a calibration factor, multiplying the measured signal by 1.15.

The graphs below show an example of how the PAR sensors perform compared to a commercial instrument in low light conditions and in response to changing the incident angle of light.





### Pressure Sensor

The Pop-Up Buoys have a [Keller PA-4LD OEM Pressure Transmitter](http://www.keller-druck.com/picts/pdf/engl/4ld_9ld_e.pdf) with embedded signal conditioning that measures pressure. It output digital data using I2C communication and does not require any calibration. It is very accurate sensors (0.7% FS) for a cost of only ~$120 each. These are the same sensors used on PMEL’s Prawler, so both the housing and small attachment PCB were taken directly from Prawler designs.



### GPS

The GPS Chosen for the Pop-Up buoys is a low power, low cost GPS Module. It is the same module used on [Adafruit’s Ultimate GPS Breakout](https://learn.adafruit.com/adafruit-ultimate-gps/overview). This made it an easy choice for prototyping and rapid development, but the module also has excellent sensitivity, time-to-fix, and power requirements – all at low cost. The modules can be purchased from 3rd party sellers such as Adafruit, but are less expensive if purchased directly from the manufacturer.

### Iridium Module

The Iridium Module used in the Pop-Up buoys is a [RockBLOCK 9603](https://www.rock7mobile.com/products-rockblock.php). This device is a very low cost SBD (Short Burst Data) Iridium module. Once again, cost was the main driver for choosing this module as they cost only approximately $250 each. Additionally, the modules were made to interface quickly and easily with controllers such as the Arduino boards and can be powered with only 5V DC. Avery thorough [IridiumSBD library](http://arduiniana.org/libraries/iridiumsbd/) was also developed specifically for using the RockBLOCK with an Arduino board, which also facilitated prototyping. Note: Some minor, but critical modifications were made to the IridiumSBD library for the Pop-Up buoys – refer to section 4.2.1 for more details.

The RockBLOCK 9603 allows for 2-way communication with the Pop-Up Buoys. Once the buoys surface, users are able to send commands to re-send any data that is lost or to adjust the interval of position updates once all data has finished sending.

## Mechanical Design

### Trawl Float

12” Trawl floats are used for the Pop-Up Buoy’s main housing because they are very inexpensive, machine well, and the shape is ideal for under the ice. The round and robust housing should is designed to ‘ride’ under the ice, where traditional instruments are easily destroyed. The trawl floats cost only around $35 and are nominally rated for a depth of 600 fathoms (1097meters).

### End Cap

The end cap design is used to consolidate all electronics and sensors into a single package and protect the instrument’s sensors and antennas. Once the electronics package and sensors are assembled together in the end cap, it can be easily assembled or removed as needed for testing and deployment.

All sensors are either flush with the upper surface of the end cap or slightly below to protect them from damage under the ice. This is critical for this instrument. Additionally, the GPS and Iridium antennas are actually mounted under the end cap, further protecting them. These antennas on other instruments are typically very vulnerable to damage or icing, leading to instrument failure. The interference from the end cap does attenuate and distort the communication slightly, but the cap is thin enough that both GPS and Iridium work effectively.

The End Cap has 2 o-ring seals for extra security: One face seal and one piston seal. The trawl floats are too thin to place two piston seals on the cap. The end cap is a half-dovetail groove because the o-ring is in between sizes and the dovetail is needed to hold the o-ring in place while assembling.

A vacuum port is placed in the end cap which is used to pump air out of the float once sealed. This creates a vacuum inside the float and holds the end cap on firmly. A vacuum of ~10psi provides ~300 lbs of force holding the cap down. There is a pressure relief valve on the center of the valve which should be used to relieve pressure when disassembling the float.

### Frame

## Electronics

### Arduino MEGA

The Arduino MEGA was used in the design of this instrument primarily to simplify the design process (that and the principal engineer was not formally trained in electronics, thus had to make some accessions). The Arduino platform is an open-source development platform, which combines a micro-processor, some fundamental software libraries, and a programming environment to make it very easy for newcomers to develop electronics equipment. Since the Arduino is open-source, there are volumes of information online about the specifics which will not be discussed here. The Arduino MEGA is one of many Arduino boards and was chosen because a) it has more input/output pins necessary to control the various signals for this instrument and b) it has extra hardware Serial ports, which are required for the Iridium module.

One main drawback of using the Arduino platform is power consumption. When the Pop-Up Buoy is just sampling sensors, it draws ~80mA. Other simple oceanographic instruments often draw less than 10 mA. This is still tolerable, because the instrument only needs to be on for a short time before going back to sleep. Also, the amount of power required to search for GPS and power the Iridium module while sending messages accounts for the majority of the required power budget and would be necessary regardless of what micro-controller were used.

One other major consideration must be made because the Arduino MEGA also does not have a ‘low-power’ mode. In order to overcome this problem, the Arduino is completely shut down when not sampling. As the Arduino cannot ‘wake itself up’, a RTC with a built in alarm is used to send a signal to a power converter and wake the unit. This is not a traditional or even optimal design, but it does work for the application, keeps cost down, and has proven a reliable method in several other projects (MTR-Duino and TAPS Controller).

In the future, it may be desirable to switch the design to a standard processor used for oceanographic instruments. This would have a few main advantages – a) lower power consumption when sampling, b) smoother communication with terminal programs (Arduinos can be a bit finicky), and c) a more standardized instrument which doesn’t require unique software or equipment. PMEL’s EDD should be consulted or included if choosing to this approach in order to optimize and standardize the equipment as best as possible.

### RTC

The Pop-Up Buoy uses a DS3234 RTC with an internal alarm to ‘wake up’ the unit from sleep at prescribed sample intervals. This is done because the Arduino does not have an on-board clock to do so and must be shut down to conserve power. The alarm is set within the program using the appropriate registers according to the DS3234 datasheet. The Shutdown pin on the LT1129 allows the Arduino MEGA to be completely shut off and then woken with a signal from the RTC.

### Batteries

Standard D-Cell Alkaline Batteries are used for the Pop-Up Buoy, primarily to reduce cost. The additional weight of the alkaline batteries at the bottom of the trawl float is advantageous as it helps to keep the unit upright. The primary voltage regulator requires a minimum 5.5V input, so a 9V pack is used to allow for both de-rating for temperature effects and voltage drop as the battery is drained.

The majority of power is consumed when the unit is on the surface, searching for GPS and sending Iridium messages. See power budget spreadsheet for calculations and programming description for more on power-saving measures.

Battery packs can be obtained from A-Pak batteries (425-820-2272). The first packs cost only ~$100 each.

### Temperature Circuit

The temperature circuit used in this design was adapted from the original MTR (Miniature Temperature Recorder) designed by PMEL. The original MTR used a circuit with a shunt regulator and operational amplifier connected to the thermistor, which produced a voltage sent to a voltage-to-frequency converter, which was finally counted by the micro-processor. In the years since the original design, electronics have seen some marked advances in many areas – one of them being with Analog to Digital Converters. The ADS1100 ADC used in the Pop-Up Buoy is a self-calibrating ADC capable with 0.02% accuracy. This is just *slightly* less than the 0.01% accuracy needed to achieve the 0.01°C design requirement, so a 0.01%, 0.2ppm reference resistor is used not only to track any drift in the sensor, but also to correct for any inaccuracy in the ADC.

# Circuit Description

## Arduino Mega

The heart of the Pop-Up Buoy Controller is an [Arduino MEGA 2650](https://www.arduino.cc/en/Main/arduinoBoardMega2560) which controls read and write functions to various ICs, collects and stores sensor data, and communicates with the GPS and Iridium Modules. It uses I2C and SPI communications as well as digital signals for ICs and sensors, and two hardware serial lines for GPS and Iridium communications.

## Power Regulation

Power is regulated and switched on and off with an [LT1129 Micro-power Low Dropout Regulator with Shutdown](http://cds.linear.com/docs/en/datasheet/112935ff.pdf) (U21). The LT1129 provides 5V power to the board and only consumes  in shutdown mode (when unit is sleeping between samples).

Both the RTC and Arduino MEGA are connected to the Enable pin on the LT1129. When waking up, the RTC alarm pulls the Shutdown signal high, turning on power to the unit. The Arduino Micro then sends a high signal via the ‘Shutdown’ Pin which keeps the unit running while the alarm is reset and sampling is completed. Eventually, the Arduino Micro sends a low voltage signal via the ‘Shutdown’ pin, forcing the Enable signal low and shutting off power to the unit. R26 is a pull down resistor to ensure the Enable signal is normally low.

3.3V power is needed for the RTC, microSD Card, Accelerometer, and I2C communication with the ADCs and is provided by U18, an [LP5907 Ultra-Low Quiescent Current LDO](http://www.ti.com/lit/ds/symlink/lp5907.pdf). The Arduino MEGA does contain an onboard 3.3V regulator, but is bypassed to prevent drawing too much current through the Arduino and destroying the board.

Several [AO3407A P-Channel MOSFET Transistors](http://www.aosmd.com/pdfs/datasheet/AO3407A.pdf) are used to toggle power to the 5V sensors (Q5), 3.3V Sensors (Q4), GPS (Q6), and Iridium module (Q3), in order to minimize power consumption. Pullup resistors (R19, R20, R22, and R24) ensure the input signal is normally high (power off) and a low voltage signal from the Arduino MEGA turns power on to each when needed. Current limiting resistors R17, R18, R21, and R23 are used for the digital signals to reduce current consumption flowing from the Arduino MEGA and avoid exceeding its maximum.

## MicroSD Card, Accelerometer, and RTC

The microSD Card, Accelerometer, and RTC require SPI communications at 3.3V and the Arduino MEGA operates at 5V. U15, a [MC74VHC1GT125 Non-Inverting Hex Buffer](http://www.ti.com/lit/ds/symlink/cd74hc4049.pdf) which accepts over voltage inputs, is used to lower voltages coming from the Arduino (SCK, MOSI, CS) and U14, a [NC7S14 Single Non-Inverting CMOS Logic Level Shifter](http://www.onsemi.com/pub_link/Collateral/MC74VHC1GT125-D.PDF), is used to raise the MISO signal going to the Arduino MEGA.

U15 is a standard [microSD Card Push-Push Socket](http://www.molex.com/pdm_docs/sd/473340001_sd.pdf) to allow the user to remove the card and download data quickly for testing and calibration, or if a unit is ever recovered.

U17 is an [ADXL345 3-Axis Digital Accelerometer](http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf). Its high resolution enables measurement of inclination changes less than 1.0° and is used to measure the angle of the Pop-Up Buoy from vertical. U10, a [SN74LVC1G32 Single 2-Input Positive OR Gate](http://www.ti.com/lit/ds/symlink/sn74lvc1g32.pdf), is recommended for the ADXL345 to prevent bus traffic from accidentally appearing as an I2C start command.

U20 is a [DS3234 Extremely Accurate Temperature Compensated RTC](http://datasheets.maximintegrated.com/en/ds/DS3234.pdf) which provides both timekeeping function and an alarm to wake the unit up and collect samples at the proper time. B2, a 12mm lithium coin cell battery, powers the RTC when the unit’s primary power is off. U19, an [NC7S14 High Speed Inverter with Schmitt Trigger Input](https://www.fairchildsemi.com/datasheets/NC/NC7S14.pdf), is used to invert the signal coming from the RTC alarm to be the proper voltage for the Shutdown input of the primary voltage regulator, U21.

When the alarm function on the RTC is not active, the INT/SQW pin is held at high impedance. This keeps the input signal to U19 at high voltage, and the output to the Enable pin low (power off). R28 is a pullup resistor to ensure the input signal to U3 is normally high. R29 is a voltage dividing resistor to keep the high side of R28 at an acceptable level for U19. When the alarm function is triggered, the INT/SQW pin on the RTC drops to low impedance, allowing current to flow and driving the input of U3 to low voltage. This causes the output of U19 to be high voltage and the high signal sent to the Shutdown pin turns power to the unit on. U19 and R28/R29 are tied to the primary battery, rather than to the coin cell, to prevent the coin cell battery from draining if the alarm is triggered, but the primary battery is dead or not connected. After all, the alarm function is only needed if the primary battery is available.

D2 and D3 are Schottky Diodes that ensure the shutdown signal coming from the Arduino and shutdown signal coming from U19 do not cause a short. The low forward voltage of the Schottky (450 mV) keeps the voltage at the Shutdown pin at the highest possible voltage to ensure the primary voltage regulator trips on.

## Temperature Measurement

The voltage measurement for temperature is done by U6, an [ADS1100 Self-Calibrating 16-Bit ADC](http://www.ti.com/lit/ds/symlink/ads1100.pdf), which achieves approximately 0.02% absolute accuracy using the settings in this instrument. The ADC requires a highly accurate power source, provided by U4, a [MAX6126 Ultra-High-Precision Voltage Reference](http://datasheets.maximintegrated.com/en/ds/MAX6126.pdf).

The ADC requires I2C communications at ~3V and the Arduino Micro operates at 5V. Q1, Q2, and R5, R7, R9, and R11 are used provide bi-directional I2C communications at the proper voltages. The I2C communications are tied to 3.3V rather than 3V to keep the ADC on an isolated circuit and because the ADC/communication protocol can tolerate the small difference in voltage.

The analog portion of the circuit was adapted from the original MTRs (Miniature Temperature Recorders) designed at PMEL. It provides an almost perfectly linear voltage output between 0.05V and 1.05V when the thermistor is between -5°C (~42k) and +60°C (~2k). The non-linear portion is only at higher temperatures where higher accuracy is not normally needed in oceanographic applications. (See the LTSpice model for more specific info). After the voltage signal is output by an [LT1013 Precision Operational Amplifier](http://www.ti.com/lit/ds/symlink/lt1013.pdf), a gain of 2 is applied internally by the ADC before sending the digital signal to the Arduino Micro. During calibration, the voltage measurement is directly fit to a Steinhart-hart equation, so conversion to resistance or any other values is not needed.

A 0.1°C interchangeable NTC thermistor is used, which can provide 0.1°C accuracy by using calibration coefficients from a different unit. This is advantageous if calibration is not possible for some reason, or calibration info is lost or corrupted. On the Printed Circuit Board, the analog circuit elements are on a separate isolated ground plane, linked to the main ground plane only by one small trace. This is done to reduce any potential ground noise from digital signals in the rest of the circuit.

Since 0.01°C accuracy was needed for this application, but the ADC and voltage reference can only provide ~0.02°C accuracy (~0.02%), R12, an ultra-high precision reference resistor with 0.01% tolerance and ±0.2ppm temperature coefficient, is used to correct for any inaccuracy. U7, an [ADG836 Dual SPDT Switch](http://www.analog.com/media/en/technical-documentation/data-sheets/ADG836L.pdf), is used to swap out the thermistor and reference resistor in order to take the alternate measurement. Low on-state resistance and tight channel-to-channel matching make this switch optimal for providing consistent measurements. The resistors are swapped out by a single digital signal sent from the Arduino Micro which is translated to 3.3V by U5, a [NL17SZ07 Single Non-Inverting Buffer with Open Drain Output](http://www.onsemi.com/pub_link/Collateral/NL17SZ07-D.PDF). R10 is a pullup resistor to make sure the output voltage is normally high and does not float.

## PAR Measurement

The voltage measurement for the PAR sensor is done by U2, another [ADS1100 Self-Calibrating 16-Bit ADC](http://www.ti.com/lit/ds/symlink/ads1100.pdf). This ADC is on an isolated, highly accurate power source – U1, also a [MAX6126 Ultra-High-Precision Voltage Reference](http://datasheets.maximintegrated.com/en/ds/MAX6126.pdf). Both ADS1100 ADCs are tied to the same I2C bus and are identified by unique addresses (determined by unique part numbers when purchasing).

The analog portion of the PAR circuit is a trans-impedance amplifier, designed following the example in [TI Designs TIDU535 Verified Design](http://www.ti.com/lit/ug/tidu535/tidu535.pdf)

The TAG-PARQ sensor is a simple photodiode, which outputs a small but precise current when light at the proper frequency strikes the face of the sensor. R2 and R3 form a voltage divider, biasing the non-inverting input of the op amp at 0.5V. The op amp, U3, is an [ISL 28134 Ultra Low Noise, Zero Drift Rail-to Rail Precision Op Amp](http://www.intersil.com/content/dam/Intersil/documents/isl2/isl28134.pdf) – the 15nV°/C offset drift for this sensor is critical in order to not affect the small signal that is output by the TAG-PARQ sensor. Biasing the non-inverting input at 0.5V is also important to account for the maximum initial offset voltage of the op amp. R1 sets the gain of the circuit at 137,000 – providing a nominal 0.5V to 2.5V signal at the input of the ADC for PAR values of 0 to 1000 molm-2s-1.

See the ‘PAR Circuit Calculations’ spreadsheet and the LT Spice model for more specific information.

## Pressure Measurement

The Pop-Up Buoys have two [Keller PA-4LD OEM Pressure Transmitters](http://www.keller-druck.com/picts/pdf/engl/4ld_9ld_e.pdf) with embedded signal conditioning that measure pressure. The pressure sensors are connected to the same I2C bus as the ADCs and simply provide a digital value for pressure to the Arduino MEGA.

When obtained from the manufacturer, these pressure transmitters have identical I2C addresses and one must be altered to avoid communication problems. Information on how this is done is contained in the [Communication Protocol document](http://www.keller-druck2.ch/swupdate/InstallerD-LineAddressManager/manual/Communication_Protocol_4LD-9LD_en.pdf) and explained in Section 6.1.

## GPS

GPS acquisition is done by U9, a [GlobalTop Technology FGPMMOPA6H Standalone GPS Module](https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPMMOPA6H-Datasheet-V0A.pdf). As soon as power to the module is switched on, it begins searching for a GPS signal and outputs serial data as NMEA output sentences, which is parsed and processed by the Arduino MEGA. Since the GPS module operates at 3.3V and the Arduino MEGA operates at 5V, U8, a [NC7S14 Single Non-Inverting CMOS Logic Level Shifter](http://www.onsemi.com/pub_link/Collateral/MC74VHC1GT125-D.PDF), and U11, a TX and RX 3.3V to 5V – U8 and U11, a [NL17SZ07 Single Non-Inverting Buffer with Open Drain Output](http://www.onsemi.com/pub_link/Collateral/NL17SZ07-D.PDF) are used to translate the serial data to the appropriate voltages.

B1, a 12mm lithium coin cell battery provides backup power to the GPS, which allows the unit to retain memory of previous satellite information and acquire a position lock more quickly.

The GPS unit also uses a [28dB Active External Antenna](https://www.adafruit.com/products/960) which is embedded in the end cap and connected to the SMA connector, J5.

## Iridium Module (RockBLOCK Mk2)

Iridium communications are completed by an external [RockBLOCK Mk2](https://www.rock7mobile.com/products-rockblock.php) module which is mounted to the main board. The iridium module communicates with the Arduino MEGA using one of the MEGA’s hardware serial ports. Apart from power and serial communications, the only other connection needed for the RockBLOCK is a digital signal to the unit’s On/Off pin which is used to toggle whether it is awake or asleep. A current limiting resistor, R25, is used for the signal to reduce current consumption flowing from the Arduino MEGA and avoid exceeding its maximum.

# Construction and Assembly

## Temperature Sensor

1. Connect Micro-Fit receptacle to wires

## PAR Sensor

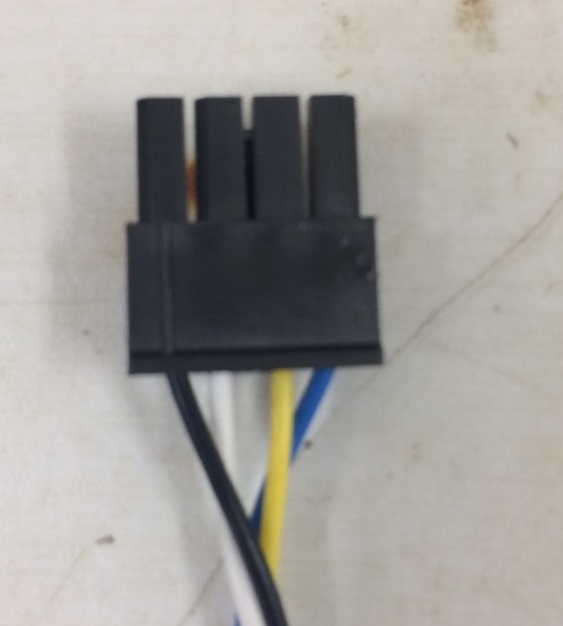
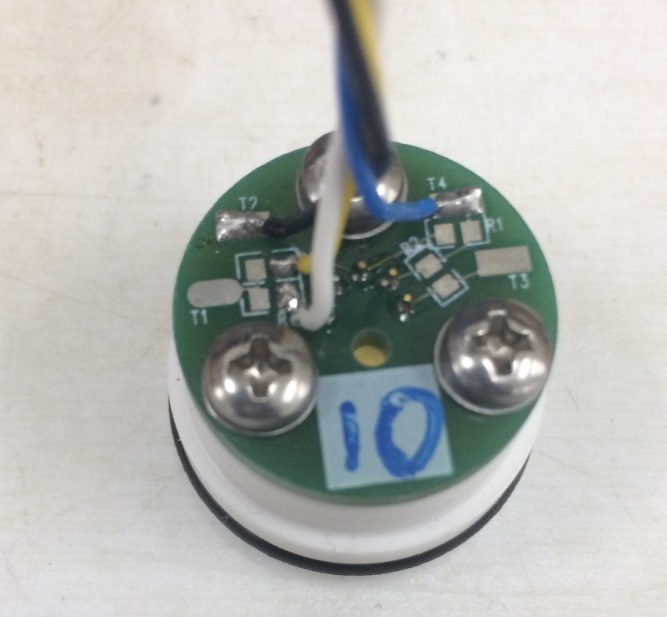
1. Machine acrylic housing per specifications.
2. Solder 6-8” of 24-26AWG wire onto the two sensor leads. **Be careful to observe polarity and connect the cathode (-) of the PAR sensor to the black wire**.
3. Set PAR sensor into housing on a flat working surface with leads facing upwards.
4. Glue PAR Sensor into place using 2 part quick set epoxy. Use a very small amount of epoxy at first to glue the bottom corner between the PAR sensor and housing first. This will ensure the small gap between the sensor and housing is sealed and that no epoxy will leak down onto the white photo-sensitive portion of the sensor.

1. After epoxy has fully set, connect Micro-Fit receptacle to wires. The hole drilled for the PAR sensor wires is large enough for the receptacle to fit through.
2. Clean O-ring grooves and install O-rings with a small amount of grease.
3. When uninstalling PAR sensor, use an awl or similar tool to push the edge of the acrylic housing out of the end cap from the bottom.

## Pressure Sensors

1. Machine pressure housing per specifications.
2. Order small pressure sensor PCBs, or get spare PCBs from Prawler parts. The pressure sensor and housing is the exact same as used on the Prawler. (Keller 4-LD)
3. Mount pressure sensor into housing using a small amount of grease on the O-ring.
4. Mount pressure sensor PCB onto back of housing using screws.
5. Solder leads from pressure sensor onto PCB.
6. Solder 6-8” 24-26 AWG wires to PCB. The sensor uses a simple 4 wire connection to the main Pop-Up board, with no resistors or other components needed. The 4 wires and pin-outs are shown below. This information is also shown on the pressure sensor data sheet. Wire colors have been preserved to avoid any confusion. 



1. Clean O-ring groove on housing and install O-ring with a small amount of grease.
2. Connect Micro-Fit receptacle to wires after sensor is installed into end cap. The receptacle will not fit through the small hole meant for the sensor wires.

## Vacuum Port

1. Wrap threads of quick-disconnect coupling with Teflon tape.
2. Screw quick-disconnect coupling into end cap. Tighten until threads are fully engaged (hand tight) using 7/16” socket.



1. Once programming is complete and unit is ready to be deployed, follow instructions in the User Guide for de-pressurizing the housing and installing the vacuum port cover.



## PCB

1. Order PCBs and components (in file Pop-Up 5.3 BOM PCA.xlsx)
2. Assemble PCBs (at [Schippers and Crew](http://schippersandcrew.com/))
3. Install coin cell batteries for both RTC and GPS.
4. Install microSD Card
5. Install Arduino MEGA by carefully aligning and pressing down on the headers on the PCB.
6. Test all functionality of all sensors, GPS, and Iridium module. See Section 6.

## Electronics Stack, Iridium and GPS Antennae

1. Manufacture and test PCB as described above.
2. Install PCB standoffs and mount to PCB Frame. Do not use plastic standoffs, they will shear off.
3. Insert GPS antenna into rectangular well in bottom of end cap. Place foam into well so that GPS antenna does not move and rests firmly up against the bottom of the well.
4. Mount PCB frame onto end cap using ¼”x1” hex cap screws, washers, and lock washers.
5. Bundle and zip tie wires from GPS and Iridium antennae so they are clear of O-ring seals and do not get caught when installing and removing the end cap. Also be sure that access to PCB and standoffs is not blocked.

## Main Battery Installation

1. Order batteries from A-Pak or other battery retailer.
2. Solder 6-18” of 24-26AWG wire onto the battery leads as needed to ensure there is adequate wire to connect the battery wires to the PCB stack on the end cap after the battery has been installed.
3. Connect Micro-Fit receptacle to battery leads.
4. Order spray-foam insulation from McMaster-Carr. Spray-foam insulation bonds very well to both the trawl float and battery pack. It also becomes very rigid and provides strong support once fully expanded, while adding minimal weight.
5. Set trawl float on a level surface, and brace the float so it does not move while the foam is drying.
6. Set the battery carefully into the bottom of the trawl float, making sure it is as level as possible.
7. Tape the battery wires to the top or outside of the trawl float to prevent the wires from being cemented into the bottom of the float.
8. Please note, the spray-foam can be very messy and has a tendency to keep expanding out of the can, after the trigger has been released. **Wear disposable nitrile gloves and be sure that the foam does not drip on O-ring seal surfaces of the trawl float.** If any foam does drip, clean it immediately as it will be very difficult to remove without damaging the surface once it has dried.



1. Begin cementing the battery into place with spray foam. The foam will expand to several times its original size and take 30 minutes or more to expand fully. **Do not allow the battery to move upwards as the spray foam is expanding.** There is not enough reserve space in trawl float and its corners must be contacting the bottom of the trawl float, not resting above it on foam. In order to achieve this, spray the foam in several stages (minimum of 3) and allow the foam to expand and set before spraying more foam. During the first stage of spraying foam, spray a small amount of foam around the 3 bottom corners of the battery pack where it contacts the trawl float and allow the foam to set. This will set the battery pack in place and prevent it from moving during future stages. Once the battery pack is set in place, fill in around the sides and bottom of the battery as best as possible. **Lastly, do not allow foam to fill over the top of the center of the battery, such that it will contact the PCB frame and prevent the end cap from seating fully once installed.** If possible, spray a small amount of foam around the top corners of the battery to hold it down, but do not reduce the clearance needed for the end cap to seat properly.

## Frame

1. Ensure Trawl Float Ear Holes are drilled to 1.00” dia
2. Insert rubber spacers into trawl float ears
3. Connect frame pieces, spacers, and anodes as shown in exploded view shown below
4. Use 5/16” 18-8 Stainless Steel Hardware for connecting all pieces.

# Software Development and Programming

## Arduino 1.0.6

The Pop-up Buoys were designed and testing using the Arduino 1.8.5 IDE (Integrated Development Environment). The Arduino IDE is a software package that lets users write and upload code directly to the microcontroller on the device. Arduino programs and libraries are mostly written in C or C++. There are some nuances to the devices and libraries, all of which can readily be found online.

Version 1.8.5 should always be used with the Pop-Up buoys unless extensive testing is done to ensure that programs work properly. There are difference in base library packages and the way programs are compiled which may have unpredictable consequences.

## Library Changes

The Pop-Up buoys use pre-written libraries for several parts of the software (interfacing with RTC, SD Card, microcontroller pins, etc. This simplifies and accelerates programming a great deal. The downside is that libraries can occasionally have errors or may not be optimized for certain uses in their original form. The following libraries needed to be changed from their original form.

### Iridium SBD

**The updated Iridium SBD library MUST be used for the pop-up buoy to function properly. It is saved as IridiumSBD-1.0-PopUp.**

IridiumSBD was a library written by some guy named Mikal Hart to interface Arduino boards with the RockBlock Iridium Module ([see web link here](http://arduiniana.org/libraries/iridiumsbd/#comments)). The library vastly simplified integrating the Iridium module with a few exceptions.

1. IridiumSBD should only be used when the RockBlock Serial lines are connected to a hardware serial port on an Arduino. If one attempts to use a software serial port, the Arduino will occasionally miss parsing characters and lock up the program. The Arduino MEGA has 4 hardware serial ports (only one other is used for USB communication), so this does not pose any problems.

2. One user noted about an issue with the doSBDRB() function. Code has been altered in accordance with this suggestion. See below for details:

“I found a bug in doSBDRB() when using a hardware serial port on the Mega 2560. When reading incoming data the for loop is not waiting for data to arrive. I was only getting the first 1 or 2 bytes written to the buffer I passed it. The rest was old data. I made the following changes and it seems to be working:”

This section of the doSBDRB() function in IridiumSBD.CPP now reads as follows. (Changes in **bold**).

*for (int i=0; i= 1000UL \* atTimeout)*

*return ISBD\_SENDRECEIVE\_TIMEOUT;*

*}*

***// if (stream.available())***

***// {***

*uint8\_t c = stream.read();*

*if (rxBuffer && prxBufferSize)*

*if (\*prxBufferSize > 0)*

*{*

*\*rxBuffer++ = c;*

*(\*prxBufferSize)–;*

*}*

*else*

*{*

*rxOverflow = true;*

*}*

***// }***

***// if (millis() – start >= 1000UL \* atTimeout)***

***// return ISBD\_SENDRECEIVE\_TIMEOUT;***

*}*

***for (int i=0; size > i; ++i)***

***{***

***// wait for the data to come in***

***while (!stream.available())***

***{***

***if (cancelled())***

***return ISBD\_CANCELLED;***

***if (millis() – start >= 1000UL \* atTimeout)***

***return ISBD\_SENDRECEIVE\_TIMEOUT;***

***}***

***uint8\_t c = stream.read();***

***if (rxBuffer && prxBufferSize)***

***if (\*prxBufferSize > 0)***

***{***

***\*rxBuffer++ = c;***

***(\*prxBufferSize)–;***

***}***

***else***

***{***

***rxOverflow = true;***

***}***

***}***

3. Another issue was found with the Pop-Up buoys when receiving messages. Two lines of code were missing at the end of the doSBDRB() function which are needed to process the “*OK\r\n*” string sent by the iridium module at the end of the iridium response.

After receiving a message, the iridium module will respond with

*<<AT+SBDRB[size]\_\_\_\_\_\_\_\_\_\_\_\_[checksum]OK\r\n*

However, the code only processed up to "*…[checksum]*". Issuing another command afterwards meant the "OK\r\n" was still in the stream buffer and would not process the next command properly.

This was solved by simply placing a waitForATResponse() after the checksum processes. (No parameters means it only looks for the terminator "OK\r\n").

This section of the doSBDRB() function in IridiumSBD.CPP now reads as follows. (Changes in **bold**).

*uint16\_t checksum = 256 \* stream.read() + stream.read();*

*console(F("[csum:"));*

*console(checksum);*

*console(F("]"));*

*// Return actual size of returned buffer*

*if (prxBufferSize)*

*\*prxBufferSize = (size\_t)size;*

***console("\r\n"); //added this line to skip to next line in terminal***

***waitForATResponse(); //added this line to process "OK\r\n"***

*return rxOverflow ? ISBD\_RX\_OVERFLOW : ISBD\_SUCCESS;*

### Software Serial

**The updated software Serial library MUST be used for the pop-up buoy to function properly. It is saved as SoftwareSerial-PopUp.**

The Pop-Up buoys use the Software Serial library to read and parse NMEA data from the GPS chip. The Software Serial buffer must be increased from 64 to 512 bytes to ensure no data is lost on long NMEA strings. This does use some extra RAM, but the ArduinoMEGA has plenty of reserve memory to accommodate this change. (Can be verified using the MemoryFree library). This is done by simply changing one line in the file SoftwareSerial.h :

*#define \_SS\_MAX\_RX\_BUFF 512 // RX buffer size //BEFORE WAS 64*

# Program Flow

## General Program Flow

1. Wake Up
2. Determine mode/phase based on current date/time and programmed cutoff dates/times

0 = Initial programming (before start date)

*If mode = 0, display parameters and shut down after user removes USB, don’t take any sample data*

1 = Bottom Sample

2 = Wait for Profile

3 = Under Ice

4 = Ice Free

1. Determine next sample time/date and set RTC alarm
2. Collect Sensor Data, Store to SD Card
3. If mode = 1 (Bottom Sample)

Do nothing else, just go to sleep

If mode = 2 (Wait for Profile)

Take initial depth measurement

Continuously take depth readings and compare to initial measurement

Wait for depth to change by *m* meters

Start profile

Collect profile data at 4 Hz for *p* seconds

Go to sleep

If mode = 3 (Under Ice)

*\*\*This mode is used primarily to preserve battery power if the is expected be under ice for several months after releasing from the bottom. In this mode the unit will only attempt to search for GPS once a day, rather than every hour.\*\**

If hour = 15 (3pm, close to solar noon in most of Alaska)

Search for GPS for *g* seconds, *n* times, with delay in between

If GPS lock is found *n* times

Store GPS location and time on SD Card

Attempt to transmit iridium data

Else

Do nothing else, just go to sleep

If mode =4 (Ice Free)

4. Ice Free

Search for GPS for *g* seconds, *n* times, with delay in between

If GPS lock is found *n* times

Store GPS location and time on SD Card

Attempt to transmit iridium data

Else

Do nothing else, just go to sleep

## Iridium Transmission

Once a GPS lock is found, the unit will attempt to transmit messages back via iridium SBD. Data is parsed from the SD Card and transmitted 100 characters at a time. The files are sent back in order of priority (profile data first, followed by under ice data, followed by pre-release bottom data). If the unit sends a message successfully, but then loses Iridium connection before sending all data (which happens often), it will go to sleep, but set the RTC to wake up the unit in 5 minutes, rather than the under ice sample interval. This 5 minute interval is used to get data back more quickly and take advantage of the calm sea state which is likely when the unit first surfaces. After all data is eventually sent, the unit will revert to sending its most recent GPS location and sample data once per day.

Each time the unit transmits a message, it will automatically check for any received messages. Received messages can tell the unit to change file, file position, or the interval of GPS location updates when all data is finished sending. Additionally, every time the unit wakes up and finds a GPS lock, the first message it attempts to send is “Hello!”. This is done in case any files are corrupted, causing the unit to lock up when sending data from the SD Card. In that case, the user may send a message to the unit, skipping to a different file location or file, and hopefully bypassing the corrupt data.

The SD Card will contain 2 files in addition to the data files: filepos.txt and summary.txt. The first unique message the unit attempts to send is summary.txt (simply a list of the files on the card and how long they are, to gauge how successful the unit was and how much data to expect. Filepos.txt is a single line text file used for the unit to remember how far along it is in the transmission process. Its format is “X,NNNNNNNNNNNNN”, where X indicates which file to open and NNNNNNNNNNNNN is the position to start reading the file (simply the number of characters after the start of the file). This is the exact same format for received messages. Values for X are as follows:

‘p’: “prodat.txt”

‘i’: “icedat.txt”

‘b’: “botdat.txt”

‘z’: if the file (or received message) starts with ‘z’, this should mean that the unit has completed sending all data and is now in sleep mode. NNNNNNNNNNNNN will indicate the interval, in seconds, for how long the unit will wait between sending updates on its most recent GPS location and sample data. By default, this is once per day.

## Iridium Network - DITCO

# Programming, Calibrating, and Testing the Units

## Program (Sketch) Descriptions

Pop-Up Buoy uses 2 different programs, all described below. The Arduino IDE calls these different programs ‘sketches’.

**Setup and Sensor Check** (*PopUp\_5.3\_SetupAndSensorCheck.ino)*

This program is used during the assembly and testing phases to test that various sensors and utilities on the units are working properly. Once uploaded and the serial monitor is launched, the user is provided a status summary of the unit and a menu to check sensors and utilities one by one. If there are any problems that require changes such as re-connecting a sensor, it is usually necessary to close the serial monitor and re-upload the program. This will allow all sensors to initialize properly.

For details and instructions on using the Setup and Sensor Check Program, see the Pop-Up Buoy User’s Guide.

**Deployment** (*PopUp\_Deployment\_5.3\_Arduino\_1.8.5.ino)*

This program is used for deployments and calibrations. All deployment parameters (dates, sample intervals, etc.) must be manually entered into the program before uploading.

Once the program is uploaded and the serial monitor is launched, the user will be provided a long dialog confirming deployment parameters and verifying unit status. This dialog should be reviewed carefully for any possible errors.

For details and instructions on configuring units for deployment, see the Pop-Up Buoy User’s Guide.

## Uploading Programs (Sketches)

**To upload different programs to the Pop-Up Buoy, follow the steps below:**

1. Connect all necessary sensors, antennas, coin cell batteries, and micro SD Card to unit electronics.
2. Connect unit to Power Supply (9V Battery or Power Supply with 9V, 1A with power on). Note: **Do not use the main battery packs for testing or calibrating as it will deplete power necessary for the final deployment.**
3. Connect unit to computer via USB cable
4. Open the Arduino IDE (Integrated Development Environment). **Be sure to use version 1.8.5.**
5. Open appropriate sketch
6. Select Board
   * *Tools 🡪 Board 🡪 Arduino Mega 2650 or Mega ADK*
7. Select COM Port
   * Check *Device Manager 🡪 Ports* to find correct port
   * *Tools 🡪 Serial Port 🡪 COM\_\_*
8. Compile Sketch
   * *Ctrl + R* or *Sketch 🡪 Verify/Compile*
   * Should Read “Done Compiling” when complete
9. Upload Sketch
   * *Ctrl + U* or *File 🡪 Upload*
   * Should Read “Done Uploading” when complete
10. Launch Serial Monitor
    * *Ctrl + Shift + M* or *Tools 🡪 Serial Monitor*
    * Note: The baud rate should be set to 115200 (lower right-hand corner after launching Serial Monitor)
    * Note: 9V Power MUST be supplied and turned on before the serial monitor is launched. It is possible to upload sketches without external power, but power must be supplied for sensors to initialize properly when serial monitor is launched.
11. To upload a different sketch, close the serial monitor and begin at Step 5. It is not necessary to disconnect the unit from power or close the Arduino IDE.

## Calibrating the Temperature Sensors

The data stored on the SD Card is a voltage reading from one of the unit’s ADC (Analog-to-Digital Converter). The voltage value is fit directly to a Steinhart-Hart curve during the calibration process so that no accuracy is lost by making conversions. Temperature can be calculated using the following equation:

Where:

T = Temperature in °C

x = ADC reading

A,B, and C = Steinhart-Hart Coefficients

The simplest way to program the unit to calibrate the temperature sensors is to use the deployment program and just have the unit take samples as if it were on the bottom waiting for release. **Calibrations should always be done using a sample interval of 30 seconds** – shorter sample intervals may cause small errors due to self-heating in the thermistor.

1. Follow procedures in section 6.1 to upload the sketch named *PopUp\_5.3\_SetupAndSensorCheck.ino.*
2. Set the RTC (Enter ‘2’ to set RTC time/date.) **Ensure the RTC is synchronized with the clock on the temperature bath.**
3. Check that RTC is keeping time (Enter ‘1’ – display Status)
4. Check that SD Card is connected and communicating properly (Enter ‘1’ – display Status)
5. Test the Thermistor probes to ensure it is connected and responding to temperature changes. A response should be visible just by holding the probe between 2 fingers.
6. Verify that the Temperature Reference is reading appropriate values. (Should be approximately 50 kOhms)
7. Close the Serial Monitor
8. Follow procedures in section 6.1 to upload the sketch named *PopUp\_Deployment\_5.3\_Arduino\_1.8.5.ino*
9. Set the value for BOTTOM\_SAMPLE\_INTERVAL to 30 seconds (within #define statements, line 42)
10. Set *unitStart* to an appropriate time in the near future
11. Set *releaseTime* to a time long after the calibration will be over. There is no risk of running out of storage space on the SD Card.
12. Compile the sketch (*Ctrl + R* or *Sketch 🡪 Verify/Compile*)
13. Upload the sketch (*Ctrl + U* or *File 🡪 Upload*)
14. Launch the Serial Monitor (*Ctrl + Shift + M* or *Tools 🡪 Serial Monitor*)
15. Carefully read the deployment dialog and verify all settings
16. Disconnect the unit from USB cable
17. Connect to 9V power (battery or power supply) before the unitStart time/date
    1. The unit can be temporarily disconnected from power as long as it is reconnected before the unitStart time/date. This way, multiple units can be connected and run off a single power supply for the calibration if desired.
18. Immerse temperature probes in temperature bath and start temperature bath program
19. Once temperature bath program has completed, download data from SD Card and process data
    1. Open a previous calibration fileto see instructions and examples of how to process calibration data. Always use *Paste Values*.
       1. Download .CAL file from temperature bath computer
       2. Follow numbered instructions on sheet *TBxxxxx* to process .CAL file
       3. Format data from individual Pop-Up units and input into *PopUpTempCal\_xxx* sheets. Follow numbered instructions and add new sheets if needed
       4. Paste data into *TempCalSummary* Sheet and run *genssh* Macro for each unit
20. Record calibration coefficients. Store data and coefficients from temperature calibration for records and post processing data. (Data transmitted by the Pop-Up buoys is uncalibrated.)

## Calibrating the PAR Sensor

The data stored on the SD Card is a voltage reading from one of the unit’s ADC (Analog-to-Digital Converter). The PAR Sensor is a photodiode which outputs a small current which is proportional to the amount of light hitting the sensor. The sensors are calibrated by the manufacturer and provide a known current output vs. incoming PAR.

PAR can be calculated using the following equation:

Where:

PAR = Photosynthetically active radiation (in molm-2s-1)

y = ADC reading

y0 = ADC reading at 0 molm-2s-1

m = calibration from manufacturer (uA/molm-2s-1)

0.000668268 is a fixed conversion factor for the circuit which converts voltage to current. Details on this can be found in the document *PAR circuit calculations - Rev 5.3.xlsx*.

0.73 is a scaling factor to account for the amount of light that is absorbed/reflected by the acrylic housing. This was found experimentally by comparing a calibrated Satlantic PAR sensor to several Pop-Up buoy PARs mounted together and exposed to identical light conditions.

The only factor that must be calibrated is y0. This is found by exposing the sensor to zero light and measuring the response. This is most easily done by using the SetupandSensorCheck sketch and recording the response from the PAR sensor with the end cap placed upside down on a black surface (no incoming light).