PODD ASSEMBLY GUIDE

Prepared by:
Plamena Milusheva pmilusheva@lmnarchitects.com
Chris Savage csavage@lmnarchitects.com

Additional design & assembly work by Belal Abboushi (LMN) and Morgan Redfield (Good Measure Design) used to create this document.

PRINTED CIRCUIT BOARD (PCB)

The printed circuit board (PCB) is the base board to which most of the PODD electronic components attach and is used to route power and signal lines between the various on-board and/or external components. This document assumes resistors, capacitors, ICs, and headers have already been embedded on the PCB (at least the initially-designed set of components).

Preparation: Soldering. While most sockets come pre-attached from the fabricator, there are a few that still need to be added. A 3x1, 0.1”-pitch female header socket needs to be added to pad P21, just above the XBee socket. These I2C test lines are reallocated for use by an ambient light sensor, a post-PCB-design change. 5x1 and 4x1 0.1”-pitch screw terminals can be added to pads J3 and J7, respectively, found on the bottom edge of the board. These lines are used to provide or receive power to/from the board.

To smooth voltage fluctuations in the power drawn by the Teensy, a ~ 47uF capacitor should be attached (soldered) between the Teensy’s “5V” pin socket and the “GND” pin socket adjacent to it on the PCB. This connection can be made on the back side of the PCB. While not necessarily required, this capacitor should help avoid sporadic Teensy resets from occurring when other components suddenly make a large power draw on the shared 3.3V supply (such as Ethernet turning on and/or SD card writing).

POWER

The PODD provides both 3.3V and 5V power lines to various electronic components: most use the former but a couple require the higher voltage. Power can be supplied multiple ways: USB, power-over-ethernet (PoE),[1] an external battery pack, or via any other 5V source. The PCB is designed to charge an attached battery pack when another 5V power source (such as USB) is connected, and draw from the battery when no external power is available.

Power is routed through the sockets labeled J5, J2, J3,[2] and J7; the first is at the top of the board, between the Teensy and ethernet, and the others are all on the bottom edge of the board. Pin connections for each of these sockets are labeled on the back of the PCB. The J2/J3 sockets provide the actual power to the PODD components and a 5V power source *must* be attached

[1] A 5V regulator is required.
[2] Sockets J2 and J3 are duplicates: they are connected to the same set of lines.

there. The J5 socket (intended for external power sources) sends any connected power to the J7 socket, which is intended to charge an attached battery.

Preparation. Make the following connections:

- *USB and/or PoE power.* Wires soldered to the USB ground and power pads on the back side of the Teensy (seen the Teensy section) should be connected to the appropriate pins on the J5 socket. If PoE is available, it must first be converted to 5V (the PODD board does not do this!) then attached to the appropriate pins on the J5 socket. The J5 socket can take any 5V power inputs, not just USB or PoE power. The only difference between the PoE and USB-labeled pins is that the board will draw power from the former if both are active, so the more stable power source should be placed there.

- *Battery pack.* A portable USB charger can be used to power the PODD. Take the battery's USB charging cord (or another USB cord with the same terminals) and cut it in two. Take the part of the cord with the larger plug, pull out the 5V power and ground lines (likely red and black, but use a voltmeter to check), and attach them to the appropriate pins on J3. Any USB data lines in the cable (if present) can be ignored. If the battery is to be charged when USB or PoE power is available, take the other part of the USB cord and attach its 5V power and ground lines to the appropriate pins on J7. The battery can be connected through these USB cables to provide power or be charged when the PODD is running, but make sure the battery is disconnected while assembling the rest of the electronic components.

- *No battery pack.* The PODD can be run entirely off of USB or PoE power by connecting a VCHRG pin on J7 to a 5V pin on J3 with a wire. **Do not short these terminals when a battery is attached as described above!**

The above steps are intended to provide a somewhat robust and flexible power supply arrangement. However, the only actual requirement for the PODD to run is for 5V of power to be supplied through J3.

Issues. The following are known issues/limitations:

- The Teensy may reboot when power supplies are attached/detached or otherwise switched.

- Many portable chargers will shut off automatically when they are not charging anything. As the PODD draws far less power than the cell phones that portable chargers are typically used for, some chargers do not recognize the presence of the PODD and shut off, leaving the PODD without power.

- An external battery cannot be attached or detached while the PODD is running.

These issues may or may not be addressed with a future hardware revision.

Do not add, remove, or even touch electronic components while power is attached. Disconnect any USB, PoE, external battery, or other power supply before working with the PCB board or anything attached to it. Spikes in power, floating (unattached) pins, or shorted pins while power is supplied can cause damage to various components. Just touching electronic components can short all pins in contact with the finger(s) and it is difficult to handle the board or

components without touching one or more pins. While damage *probably* will not occur, such damage can lead to erratic behavior that is difficult to find the source of.

TEENSY

The microcontroller at the heart of the PODD is a AT90USB1286[3] embedded in a Teensy++ 2.0[4]development board that breaks out many microcontroller pins and provides a USB interface; we will henceforth use "Teensy" to refer to the microcontroller and/or this development board.

Preparation: Soldering & Trace Cutting. Unless purchased with pin headers already attached, two 20-pin headers must be soldered to the Teensy. Short (1.5" - 2") wires must be soldered to the ground and USB 5V pads on the back side of the Teensy; these will be used to route USB power to an external battery attached to the PODD PCB.

Cut the small (almost invisible) trace between the 5V pads[5] on the back of the board—this can be performed with an exacto knife—and check that the pads are disconnected using an ohmmeter: resistance between the pads should increase from less than 1Ω to the MΩ range. This disconnects USB power from the microcontroller, which will be supplied with lower-voltage 3.3V power elsewhere. **Do not omit this step or electrical components may be damaged later.**

Power. The Teensy++ 2.0 as provided by the manufacturer (PJRC) runs at 5V using the 5V power supplied by a USB cable. To reduce power usage, the Teensy can be run at 3.3V instead.[6] To do so, the Teensy must be disconnected from the USB power by cutting the trace between the 5V pads on the rear of the Teensy.[7] Once the trace is cut, the Teensy can no longer be run via a USB cable alone. If you wish to restore the Teensy to its original USB-only operation, just bridge (solder together) the two 5V pads where the trace was cut.

In the PODD, the Teensy will be powered from an external 3.3V supply through the "5V" pin on the Teensy board. Some websites that discuss modifying the Teensy to operate at 3.3V say to bridge (solder together) the two 3V pads after cutting the trace between the 5V pads. This is unnecessary for use in a PODD. [The purpose of the bridge is to allow the USB to be operated in host mode at 3.3V?] The CPU speed (controllable via firmware) should be reduced to 8 MHz or lower when run at 3.3V. This speed can be set via the Arduino IDE when uploading the PODD firmware.

While the USB power must be disconnected from the Teensy microcontroller, the USB power can be rerouted for use elsewhere. The short wires soldered to the Teensy USB ground and power supply pins can be attached to sockets adjacent to the Teensy on the PODD PCB, which will use the 5V to charge an attached external battery pack and/or send it to an on-board buck converter that reduces the voltage down to the 3.3V that the PODD components need.

Note a USB serial interface between a computer and the Teensy is still possible with the USB power disconnected and the Teensy operating at 3.3V.

---

[3] https://www.microchip.com/wwwproducts/en/AT90USB1286
[4] https://www.pjrc.com/teensy/
[5] As shown here: https://www.pjrc.com/teensy/external_power.html
[6] https://www.pjrc.com/teensy/3volt.html
[7] Operating the Teensy in the PODD without disconnecting it from the USB 5V power line will drive all of the electronic components on the PODD board to operate at 5V instead of the intended 3.3V. While the Teensy is designed to operate at 5V, many of the other components are not and may become damaged.

Firmware. The Teensy firmware is the program that controls how the Teensy operates and interacts with various peripherals (like sensors). In order for the Teensy to operate the PODD, the custom PODD firmware must be uploaded to it. This will be discussed in a later section.

SUB-BOARDS

Several breakout boards must be attached to the PCB to provide power, timing, data storage, and networking. The PCB layout is sized so that the below boards can be attached directly to the appropriate header.

Adafruit LM3671 Breakout. The LM3671[8] is a DC-DC stepdown voltage converter, used here to reduce the 5V power supply down to the 3.3V used by most of the PODD components. The PODD PCB layout is designed to make use of the Adafruit LM3671 breakout board, but other voltage converters capable of providing at least 500mA can also be used. However, other voltage converters may have a different pin layout, in which case they should not be directly attached to the PCB header (use wires to connect pins appropriately).

SparkFun DS3234 RTC. The DS3234[9] is a temperature-compensated real-time clock (RTC) that maintains the PODD clock time. The SparkFun DS3234 breakout board provides a coin-cell battery holder that allows the clock to maintain time even when the PODD is not powered; a 12mm CR1225 battery (not included) should be installed. The breakout board will be attached to the PCB with the battery holder facing up.

Though the firmware startup menu allows the RTC time to be set, an ethernet-connected PODD will automatically set the time using an NTP server on startup. In addition, during normal operation, an ethernet-connected coordinator PODD will maintain its time regularly with an NTP server and will broadcast that time to all other nodes on the XBee network; the PODD cluster should, within a couple of minutes of joining the XBee network, have their clocks synced within 1-2 seconds of each other. With a functioning PODD setup, timekeeping should need no initialization or other intervention by the PODD operator.

SparkFun MicroSD Breakout. While sensor data is uploaded to a database as it is read, all data is written locally to a micro SD card for safety. This ensures data is not lost in the event of a network or server issue, which may occur and continue unnoticed due to the unattended nature of the PODD deployments. In addition, network congestion and other intermittent issues mean a small portion of the packets can get lost on their way to the database.

WIZnet WIZ812MJ Ethernet Module. The WIZ812MJ board contains a WIZnet W5100 ethernet chip that provides a wired-networking ability to the PODDs. Only strictly necessary for coordinator PODDs, which will receive data from other PODDs over the XBee network and upload that data to a database through the ethernet connection. Drone PODDs will power down this module once they enter normal running mode to reduce power usage.

XBee Pro 900HP S3B. An XBee network is used to transfer data within a PODD cluster, due to its much longer range than WiFi. The 900MHz band was chosen over the more common 2.4GHz band due to better wall penetration and lower congestion. The DigiMesh version was chosen due

[8] http://www.ti.com/product/LM3671
[9] https://www.maximintegrated.com/en/products/analog/real-time-clocks/DS3234.html

to the larger range and resilience of that networking topology.[10] Other XBee modules may work fine, though the pin-outs and serial command configuration must be checked to ensure compatibility with both the PCB layout and the PODD firmware. The XBee module requires a bit of configuration, which is described in its own section below.

I2C Breakout. Due to a switch in some of the sensor components after the PCB was designed, more I2C bus connections need to be made than the PCB provides headers for. As the I2C bus shares the clock and data lines amongst all the I2C components, we can simply split out connections at one of the available I2C headers. To accomplish this, we add a three-pin header to the I2C test points at P21 on the PCB (ground, clock [SCL], data [SDA]). To this, we attach a cut down piece of breadboard with several three-pin headers attached in parallel.

Currently, we use this breakout to provide those three lines to each of the OPT3001 light sensor and Sensirion SPS30 particulate matter sensor. The OPT3001 3.3V power input is attached to the original light sensor's 3.3V header pin, while the SPS30 5V power input is attached to the original particulate matter sensor's 5V header pin. In I2C mode, the SPS30 can operate safely on the 3.3V I2C bus as it will never drive the lines to 5V (which could damage other components on the 3.3V lines). See the later section on the SPS30 for how to ensure it is run in I2C mode.

XBEE

XBee modules (XBee Pro 900HP S3B) provide a low-power, long-range network to transport data from remote PODDs ("drones") to a single coordinator PODD that is connected to the internet. Other than specifying the coordinator, XBee configuration cannot currently be performed within the PODD firmware's startup menu. Instead, initial setup should be done through the use of an XBee Grove development board[11] along with the free XCTU software,[12] which allows the XBee to be fully configured and tested prior to deployment. The use of this setup allows *any* XBee on a network to be reconfigured remotely as long as the Grove-attached XBee is part of that network.

Setup:

- Place the XBee in the XBee Grove development board socket, then connect the Grove board to the computer with a USB cable. *Important: Do not add or remove the XBee while the USB is attached.*
- In the XCTU program, use *Add Radio Module* and select the port the XBee is attached to. If you are unsure which port to use, use *Discover Radio Modules* instead.
- Select the radio module on the left side to bring up all the current XBee settings. A blue triangle in the right corner of a setting means a non-default setting. A green triangle indicates a change that has not yet been written to the XBee module (you must press the *Write* button to commit changes). The following parameters should be set (numeric entries are hexadecimal, but entered without a leading '0x'):

[10] Unlike conventional XBee network topologies, the DigiMesh has no coordinator role for messaging and the generic coordinator address (0x0000) is not used. The PODD firmware is responsible for relaying an explicit coordinator PODD address to all the other nodes for message routing.
[11] https://www.digi.com/resources/documentation/Digidocs/90001457-13/
[12] https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu

Setting	Value	Description
HP	**4**	Preamble ID used to differentiate between XBee networks. Changed here only to avoid the commonly used default of 0. [0x0 - 0x7]
ID	**1234**	Network ID, the primary means of differentiating XBee networks. This should be a project-specific value. [0x0000 - 0x7FFF]
NI	*<label>*	Label for this particular XBee module, up to twenty characters. Note the default label is a blank space (‘ ‘): be sure to delete this so your label does not unintentionally begin with a blank.
EE	**1** *(enabled)*	Enable network encryption.
KY	**13579BDF**	Network encryption key. All modules on the network must have the same key to communicate with each other. This should be a project-specific value. Up to 32 hexadecimal characters.
AP	**0** *(transparent mode)*	Serial communication format: the PODD firmware uses transparent mode. Transparent mode is the default, but switching to API mode [1] is necessary to use some of the XCTU features (the mode must be switched back when done).
RP	**5** *(x100ms)*	(Optional) How long the Grove board’s RSSI LED lights up when a packet is received by the XBee. This is only useful for testing on the Grove board. The default value is 28x100ms, too long for the LED indicator to be useful when testing PODDs.

Press the *Write* button to commit the changes.

- If one or more PODDs on the same network are up and running, the XBee can be tested at this point as discussed below. Be sure AP is returned to 0 (transparent mode) when done testing.

- Remove the XBee from the Grove board and place it in the PODD board socket. The small pins are easily bent, so insert and remove XBees from the sockets carefully. To minimize pin bending when removing an XBee, avoid tilting the XBee board (much) as you lift it off the socket; gentle rocking by alternatingly lifting from each end of the board can help. If any pin is badly bent out of place, try to bend it mostly back. When attaching an XBee, the sockets are somewhat forgiving of mildly bent pins, as long as you ensure the pins are aligned with the socket holes before pressing down.

Connecting to an active network. As long as the Preamble ID (HP), Network ID (ID), and encryption key (KY) settings of the Grove-attached XBee are the same as those of other running XBees, the XCTU software can monitor and interact with all XBees on that network. In order for XCTU to do so, the serial interface on the Grove-attached XBee must be switched to API mode (AP = 1). When done using XCTU, return the XBee to transparent mode (AP = 0) before inserting the XBee into the PODD.

Transparent vs. API Modes. There are two different modes for communicating between an XBee and e.g. a microcontroller over the serial interface: transparent mode and API mode. Transparent mode is a limited, but simple-to-use means of passing information from one XBee to another: characters sent to the XBee simply appear as output on the destination XBee, without need for the microcontroller to create or parse network packets itself. This is sufficient for the PODDs, which only send data from each drone to the coordinator. The more low-level API mode requires the controller to handle network packet creation and/or parsing, which is far more difficult to implement in the controller firmware. However, API mode allows far more flexibility in routing and inspecting packets. The XCTU software makes use of this mode to view or modify an active network configuration.

Reconfiguring remote devices. With the Grove-attached XBee in API mode and connected to an active network, press the “Discover radio nodes in the same network” button on the XBee entry in the left panel. Add the discovered devices and they will appear in the left panel. If you select any of these devices, all of its settings will appear to the right. Settings can be changed and committed in the same manner as for the attached XBee. If you change any of the network or encryption settings (HP, ID, EE, KY), you may lose access to the remote XBee once the changes are written to the device. If you intend to change these network settings, say to change the network ID and encryption key as PODDs are transitioned to a new project, change all the desired settings of a remote device and then commit (*Write*) all the changes at once. Do this for all remote devices. After all the remote devices are reconfigured, then make those changes to the Grove-attached XBee. At this point, you should be able to discover and reconnect to the remote devices again.

Monitoring an active PODD network. With the Grove-attached XBee in API mode and connected to an active network, you may see how PODDs are inter-connected or observe data packets being passed over the XBee network. Switch to Network working mode in XCTU and press *Scan* to see all the XBees on the network and the strength of the signals between them. XBee icons can be dragged around the canvas if connection lines overlap. Hover over an XBee to see more information. Press *Stop* to stop the scan when done.

Switch to Consoles working mode in XCTU and press *Open* to see all packets passing through the Grove-attached XBee. Network packets that the attached XBee sees will be listed under *Frames log*. PODD data packets should appear as type “Receive Packet”. Select one of these packets and scroll down to the bottom of the *Frame details* panel on the right to see the packet contents under the *RF data* entry. If everything is working correctly, that entry should contain the PODD name, the sensor type, the sensor reading, and a timestamp. In a large network, some network packets may not be visible as they can get routed from a drone to the coordinator without being in range of the attached XBee (and, hence, the attached XBee does not see them).

Message routing. Drone (remote) PODDs send sensor data packets directly to the coordinator, while the coordinator occasionally broadcasts the clock time and other useful data to all nodes. Broadcasts are made on an XBee network by setting the destination to 0xFFFF; all XBees on the network will receive broadcast messages and pass them on to their PODD transparently. In a

traditional XBee network, messages can be sent to the XBee coordinator node by setting the destination address to 0x0000. However, DigiMesh networks have no message coordinator role and no XBee modules will receive messages sent to the reserved coordinator address. To get around this limitation, drone PODDs will explicitly set the destination address to the 64-bit serial number of the XBee on the PODD set to be the PODD coordinator. As drones do not *a priori* know the coordinator's serial number, the coordinator will regularly broadcast that number to the whole network. That means drones may take a few minutes to properly route messages if the coordinator role is changed to a different PODD.

SENSORS

The following are the current set of sensors used by the PODDs.

ClosedCube OPT3001 Ambient Light Sensor. The OPT3001 is a calibrated, auto-ranging, digital light sensor that measures illuminance levels over 0.01 - 80,000 lux. We use this sensor on a ClosedCube OPT3001 breakout board. The sensor can be placed under a clear window (e.g. acrylic) without much impact on the results: a consistent 1-10% loss in light level is unlikely to be significant in architectural applications, given the orders-of-magnitude variation in light levels being measured. However, the window should be wide enough not to inhibit low-angle light as side-lighting (from e.g. windows) is common in architectural applications.

The rise of cell phones, with in-built light sensors, has led to cheap, high quality light sensors such as this one, which is cheaper, more accurate, and has higher dynamic range than the analog sensor the PCB was originally designed for. As the PCB was designed for an entirely different type of sensor, there are no headers explicitly designed for the OPT3001. Instead, the OPT3001's I2C lines are connected to the I2C test pins next to the P21 pad on the PCB; the P21 pad is used as the ground. The 3.3V power is supplied through the original light sensor's 3.3V power pin.

SparkFun Electret Microphone. The SparkFun electret microphone breakout board provides an analog speaker signal, boosted by an on-board amplifier. This board can be attached directly to the PCB, but any enclosure would attenuate the sound in that case. Be aware that, as an analog device, the microphone is prone to pick up electrical noise (very long wires are not advised).

The PODD firmware currently samples the microphone tens or hundreds of times per second and uses the average fluctuations of those samples over longer periods to provide an indication of average sound levels. This could be calibrated to unweighted decibel (dBz) levels. However, obtaining the frequency-weighted sound levels (dBa or dBc) typically used for human hearing would require digital signal processing beyond the level this microcontroller and microphone can reasonably perform. Performing that type of analysis is a target for the next iteration of the PODD hardware/firmware.

Honeywell HIH8120 HumidIcon Humidity/Temperature Sensor. A digital humidity and temperature sensor that connects using an I2C bus. This sensor can be attached directly to its PCB headers if the enclosure provides ventilation holes: the PODD generates 0.1 - 0.2W of heat, but over a large enough area that it is unlikely to elevate the temperature around the sensor at any significant level unless the enclosure is tightly sealed (but you may want to test this).

PR222J2 Thermistor. Provides an analog temperature measurement. This small sensor is intended to be placed in a dark (black) sphere to capture radiant energy. The PODD software and documentation variously refer to this thermistor's temperature as the globe or radiant

temperature. However, since any such sphere is going to have a thermal coupling to the ambient air, the thermistor's temperature is *not* the true radiant temperature: the thermistor temperature is somewhere between the ambient air and radiant temperatures (caveat emptor).

Copper is the ideal sphere (globe) material. However, a ping-pong ball painted black is much cheaper and should be sufficient if the goal is to identify when/where the radiant temperature differs significantly from the ambient temperature (again, the thermistor does not give the radiant temperature, but correlates with it).

The temperature of the thermistor is determined by measuring the voltage across it when placed in a voltage divider with a resistor of known resistance as is done on the PCB. This setup allows the thermistor's resistance to be calculated and, since the thermistor has a well-known temperature/resistance relationship, the temperature can thus be inferred.

COZIR Ambient CO_2 Sensor (2000, 5000 or 10000 ppm maximum). Low power CO_2 sensor. Can be placed within an enclosure with ventilation holes as CO_2 diffuses relatively quickly at that scale.

These sensors drift over long periods: they should be calibrated prior to first use and then again every few months (if not more) to ensure accurate results. Using a CO_2 meter to obtain the current CO_2 value, the sensor calibration can be performed through the PODD firmware's startup menu. For best results, keep the sensor and reference meter close together and far from human exhalation, and allow the readings to settle for several minutes when performing the calibration. If the sensors are in sealed packaging, they should be removed and exposed to ambient air for an extended period (10+ minutes) prior to calibration. The PODD sensor testing mode can display continuous sensor data and can be used to determine if the readings are stabilized and/or if the sensor maintains the desired value after the calibration.

Alternatively, calibration and CO_2 sensor testing can be performed using the Gaslab[13] program and a FTDI USB-to-serial cable to connect the sensor directly to the computer. To get connected within the Gaslab program, select the port (varies), series (Cozir), and model (Ambient), and press *Connect*. Press *Read CO2* to get the sensor's current CO_2 measurement. Using a CO_2 meter to obtain the current CO_2 value, use *Calibrate to Span* (using the current value) under *Configure Sensor* to calibrate the sensor. Logging data for 1-2 minutes (using the *Auto-Logging* feature) should indicate if the readings are stabilized and maintain the desired value after the calibration. Also under *Configure Sensor*, set the filter value to 32. The filter is used to reduce fluctuations by taking a moving average of previous raw measurements (which are taken at 2 Hz, by default). The PODD firmware will set the filter value automatically on startup.

The COZIR sensors have a long-term auto-calibration mechanism (enabled by default), where the lowest measurement over the calibration interval (default is about a week) is assumed to correspond to the atmospheric background level (400 ppm); the interval and background level are configurable, but not through the PODD firmware or Gaslab software. This auto-calibration mechanism assumes the CO_2 in the space being studied regularly falls to background levels, which is typically the case in indoor spaces which have extended unoccupied periods (nights and/or weekends). While this mechanism can help keep the sensor accurate over long data-taking runs, the auto-calibration information is lost whenever the sensor loses power, in which

[13] https://www.co2meter.com/pages/downloads

case new readings are based upon the last explicit calibration until enough data is collected for a new auto-calibration process (hence, the importance of regular manual calibrations).

SPEC Sensors CO Sensor. This sensor is a fast, low-power carbon monoxide sensor with an analog signal proportional to the CO level. The associated potentiostat circuit (similar to the one shown in the *SPEC Sensor Operation Overview* document) is included on the PODD PCB. However, it is unclear at this time how to interpret the analog output, i.e. what the CO level in ppm is for a given analog reading. In principle, this calibration can be performed by hand with a separate, already-calibrated CO meter and an environment with a significant level of CO. In practice, it has proven difficult to find such an environment and ideal levels for calibration (100-1000 ppm) are hazardous to human health. In the meantime, sensor output can be assumed to be proportional to the CO level, so can PODDs can still give relative CO level variations.

Note that, according to the example potentiostat diagram in the SPEC document, the C and R pins on the sensor may be reversed on the PCB; should that be the case, the situation can easily be rectified by swapping the connecting wires. However, until an adequate test case presents itself, we will be uncertain as to the correct pin connections.

Sensirion SPS30 Particulate Matter Sensor. A fast, relatively low power particulate matter (PM) sensor. The firmware currently extracts PM2.5 and PM10 measurements in $\mu g/m^3$, though the sensor is capable of providing both mass and number concentrations and over a larger set of particle sizes. Any PODD enclosure can inhibit dust from reaching the sensor: ensure the sensor's apertures have adequate exposure to the ambient air. Ventilation holes in the PODD enclosure may or may not be sufficient for this purpose. The *Mechanical Design and Assembly Guidelines for SPS30* document provides guidelines for appropriate enclosure design and permitted SPS30 orientations.

Low power here is relative to other PM sensors: this sensor draws more power than all other PODD components combined during operation (excluding ethernet, which is disabled on all but the coordinator). For longer battery-operated runs, this sensor should be sampled sparingly, at maybe hourly intervals. The PODD firmware will turn off power to this sensor until two minutes before taking a reading: the sensor needs to run 1-2 minutes before the readings stabilize. After the reading, the firmware will turn off the sensor to save power.

As the PCB was designed for a different PM sensor (the Amphenol SM-PWM-01C), the SPS30 does not directly plug into a dedicated PCB header. Instead, the SCL and SDA lines are connected to the I2C test pins next to the P21 pad on the PCB; a small break-out board was created to split out these I2C test pins as the light sensor also attaches there (see Sub-Boards section above). The SPS30's 5V line is attached to the center pin of the PCB's original PM sensor header. Both the select (SEL) and ground pins on the SPS30 are attached to grounds on the PCB's PM header (both the first and last pins on that header are grounds). Connecting SEL to ground is necessary to select I2C communication; leaving this unconnected means the SPS30 will attempt to communicate through UART on those same lines (potentially driving unsafe 5V levels on those 3.3V lines).

The PM sensor uses a fan to maintain airflow, but can still collect dust. To clear out this dust, the sensor will run the fan at high speed for 10-12 seconds every 6-7 days of continuous power. This self-cleaning cycle can also be started manually through the PODD startup menu.

FIRMWARE

The PODDs are operated by custom firmware on the Teensy microcontrollers. The PODD firmware goes through a startup sequence, printing status messages to serial output as it goes through this sequence. That sequence includes a short period where measurements will continually be pulled from each of the sensors, which allows for a quick assessment of whether the sensors are operating correctly. Most of the way through the sequence, the user will be prompted to enter an interactive mode. If no response is given over the serial interface within 30 seconds, the PODD will complete its startup sequence and enter its normal operating phase, where it regularly collects measurements and sends them on to a database. If the user provides any response (even just an empty string) during that prompt period, the PODD will enter an interactive menu system, which can be used to view or change node/project settings, set sensor reading intervals, test or calibrate sensors, etc. The menu system can be exited, in which case startup will complete and the PODD will enter its normal operating phase. The serial interface can be accessed through the Arduino IDE (*Tools → Serial Monitor*) if that software is installed and the Teensy is connected to the computer via USB.

The firmware, third-party supporting libraries, and various documentation are available on the LMNts github repository.[14] A copy of the repository can be downloaded by clicking on the *Clone or download* button and selecting *Download ZIP*. In order to upload the firmware to the Teensy board, both the Arduino IDE[15] and Teensyduino[16] software packages must be installed. Then copy all the directories in the PODD *Software/Libraries* folder to your system's Arduino libraries folder (on Windows, this is probably *Documents/Arduino/libraries*). Open the PODD sketch (*Software/SensorPod_FW/SensorPod_FW.ino*) in the Arduino IDE. Under the *Tools* menu, change the board to "Teensy++ 2.0", the CPU speed to "8 MHz". With the Teensy connected to the computer by a USB cable and the correct port selected on the Arduino IDE, the firmware can now be uploaded to the PODD.

KNOWN ISSUES

In addition to issues discussed above:

- The PODD will often fail to grab an IP address on the network. It is unclear if this is a firmware, hardware, or network issue. However, the coordinator PODD will regularly try to (re)connect to the network when it does not have a valid connection, so disconnection periods tend to be limited unless the network itself is inaccessible.
- In order to save power, portable battery packs now routinely turn off if an insufficient amount of current is being drawn, as the onboard 3.3V → 5V voltage converters often drain power even when no loads are connected. The PODDs draw far less power than recharging cell phones (the typical use case for portable battery packs) and, in nearly all commercially-available battery packs, fail to pull a sufficient amount of current to keep the packs on. We are currently using TalentCell battery packs, which utilize a physical switch to turn on and off power to a standard USB plug. One drawback of these packs is that they use 12V barrel plugs to charge (charger usually included), rather than USB: they cannot be recharged using the power routing system designed into the PODD PCB.

[14] https://github.com/lmnts/PODD
[15] https://www.arduino.cc/en/Main/Software
[16] https://www.pjrc.com/teensy/teensyduino.html