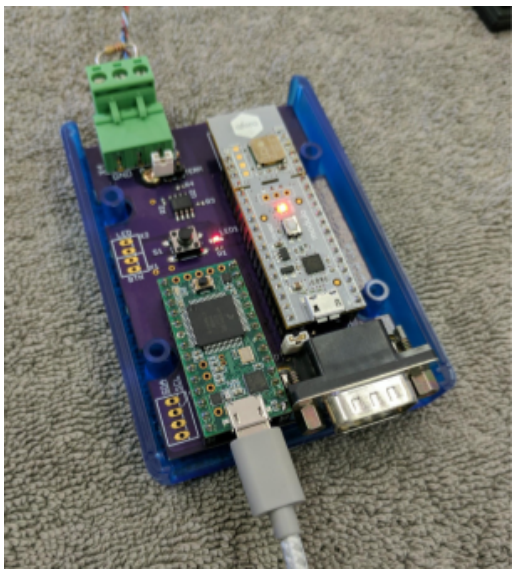
Tech Note

Honeycomb Afero Interface

For Legacy RS232/RS485/Modbus Devices

August 23, 2017 - Version 1.0



1. Background

Users of many legacy industrial, commercial, and medical devices (sensors, actuators, controllers) could benefit from internet connectivity for those devices. However, most lack a turnkey way to connect them. Often, these legacy devices include one of these standard interfaces:

- RS-232
- RS-485
- Modbus (actually a protocol, typically implemented over RS-485/RS-232)

Although the physical (electrical) interfaces are standardized, the format of the data (protocol) is not. Therefore, connecting each of these legacy devices requires a combination of semi-custom hardware and software development to adapt the physical interfaces and protocols for internet connectivity.

In addition to the hardware and protocol adaptation, the following tasks must be addressed:

- Creation of generic data format
- Mobile app development (not needed for all applications)
- Cloud infrastructure (data storage, retrieval, visualization, integration, etc.)
- Security practices

2. Challenges and Opportunities

From the Background section above, one can see that there are multiple nontrivial engineering tasks needed to connect these legacy devices to the cloud.

If developed from scratch, a skilled team could address the tasks in a vertical manner (non-platform, without focus on reusability) in 6-12 months.

To roughly scope the value of improving this situation, industrial sensors alone are a $14B opportunity in the United States, growing at a rate of 6.1% annually. (Refer to https://www.freedoniagroup.com/Sensors.html.)

Using Afero, all required tasks (except for writing the code to adapt a particular protocol) can be handled automatically. This means you can securely connect these legacy devices at a cost of only one day to one month of development effort, and with a fraction of the engineering resources (often only **one engineer** is needed).

3. Honeycomb Reference Solution

Afero provides a simple and secure framework for connecting IoT devices. Typically, Afero technology is embedded in a hardware module, which in turn is embedded in a connected product. This hardware module contains all the software needed for network onboarding and secure data transfer.

The Afero Honeycomb Reference Solution (“Honeycomb”) packages such a hardware module (also referred to as an Afero Secure Radio, or ASR) with a set of legacy serial interfaces (RS-232, RS-485, Modbus) and an external Arduino-compatible MCU (microcontroller). With a Honeycomb, a developer can quickly and securely link a legacy device to the internet.

In addition to the secure connection, the developer can use the Afero Profile Editor tool to quickly define their data model and mobile app UI without writing any software. Once that step is done, the resulting design can be pushed over-the-air (OTA) to the Honeycomb device, mobile phone or tablet, and the cloud. At this point a nearly complete and secure end-to-end IoT solution is deployed, including a live mobile app UI and live cloud API. See Figure 1 below.

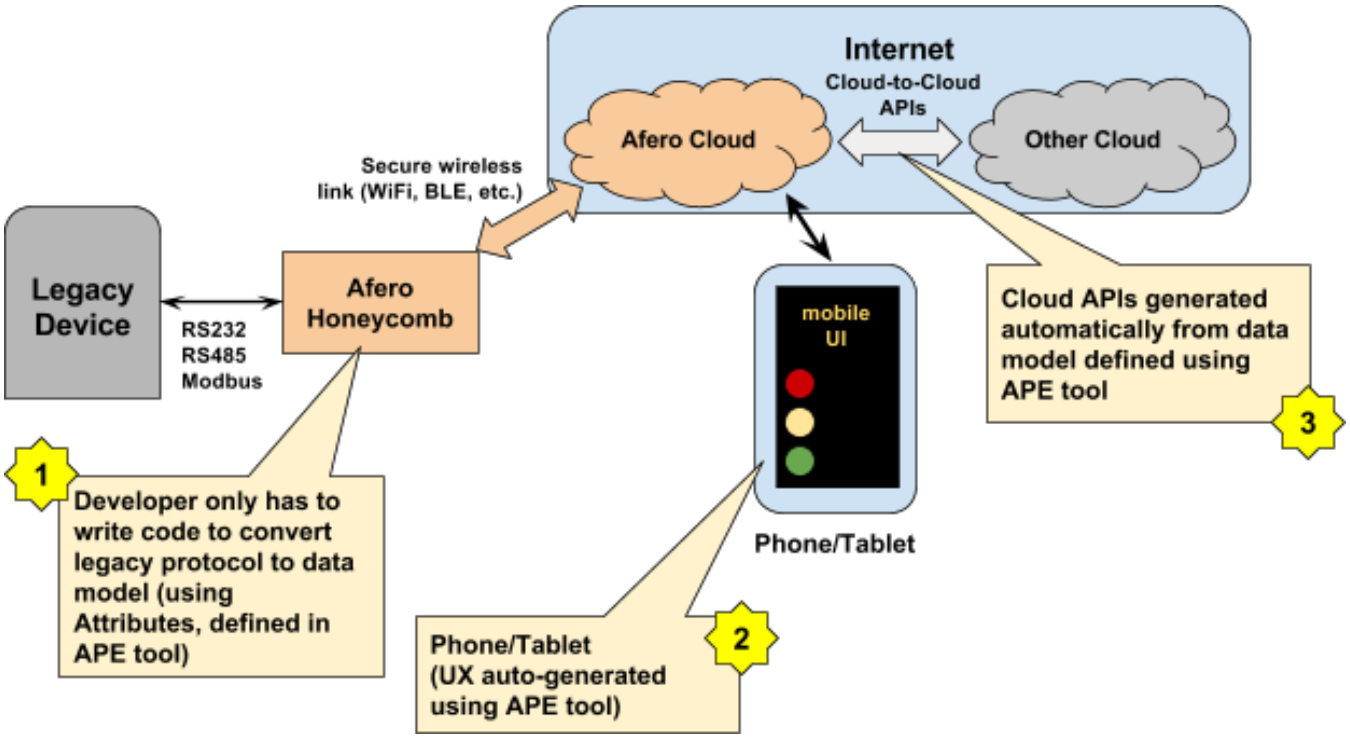


Figure 1: How Afero Honeycomb Connects Legacy Devices

A block diagram of the Honeycomb hardware is shown in Figure 2 below.

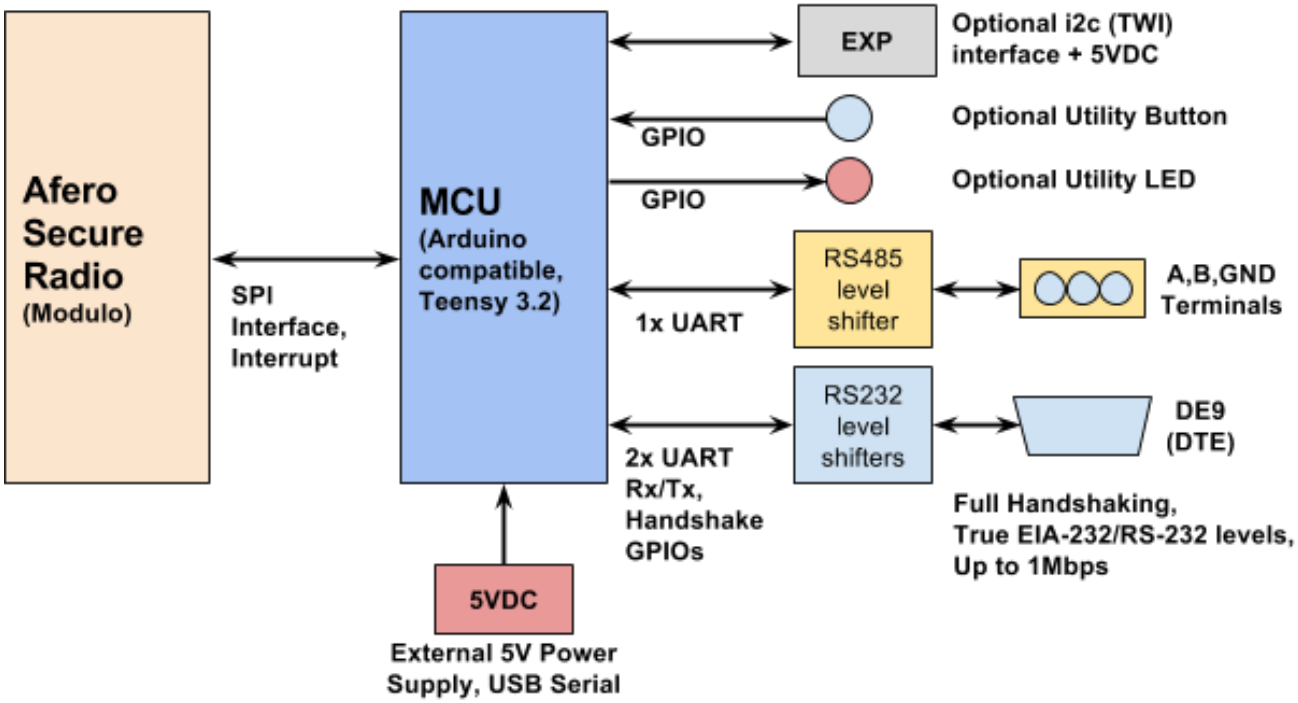


Figure 2: Afero Honeycomb Block Diagram

The Reference Solution includes a PCB layout, which fits in an off-the-shelf Hammond enclosure and exposes the standard connectors used with RS-232 and RS-485 interfaces.

Figure 3 below shows an image of the Honeycomb PCB and callouts for the various features.

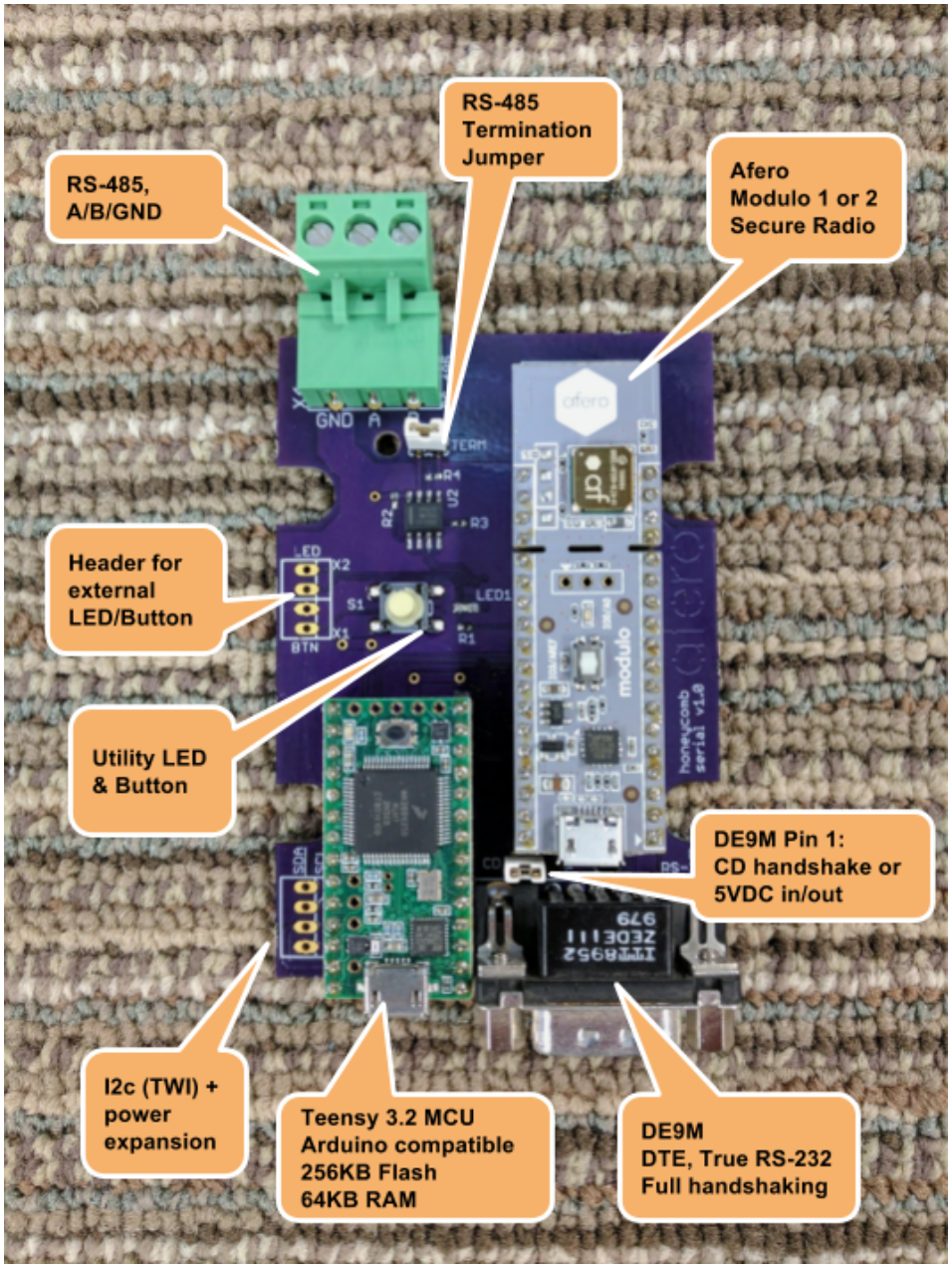


Figure 3: Afero Honeycomb Reference Design Board Overview

The Teensy 3.2 MCU included in Honeycomb is an Arduino-compatible ARM processor running at 96MHz. It includes 256KB of Flash and 64KB of RAM. In addition to secure connectivity, this provides sufficient horsepower to do significant **edge processing** on Honeycomb.

3.1 RS-232 (AKA EIA-232, TIA-232) Port

The RS-232 port is pinned out like a PC. This means it is DTE (Data Terminal Equipment); it looks like a **terminal** rather than DCE (Data Communication Equipment), which is the pinout for something like a **modem**.

In addition to receive (RxD) and transmit (TxD), there are two **output** handshake lines:

- DTR, or Data Terminal Ready
- RTS, or Request To Send

And four input handshake lines:

- CD, or Carrier Detect
- RI, or Ring Indicator
- DSR, or Data Set Ready
- CTS, or Clear To Send

Note that these handshake lines are connected to Teensy I/O pins through an RS-232 level shifter IC. The signals are named for their legacy counterparts, but are completely under software control and can be used for any desired function.

Note also that RTS/CTS are connected to Teensy pins which can be GPIOs or Teensy Serial Port 3 RxD/TxD. This means that **two non-full-handshake RS-232 ports are available** on the DE9 connector if those pins are configured and used at Serial Port 3.

See the diagram below for pinout information.

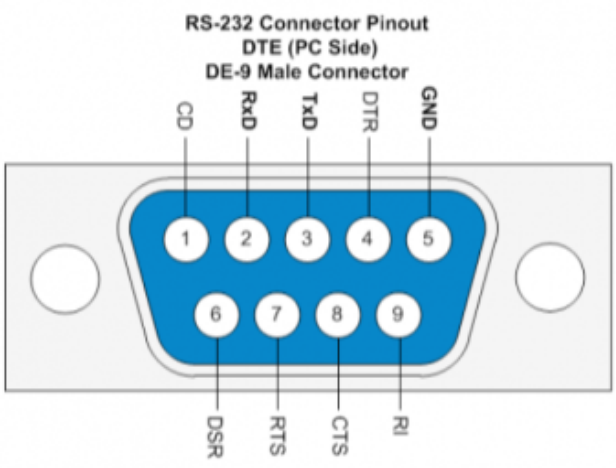


Figure 4: Honeycomb RS-232 Port Pinout

3.2 In Enclosure

Honeycomb is designed to fit in a **Hammond 1593L** plastic enclosure. The end panels can be milled to accommodate the USB and serial connectors. Figure 5 below shows what Honeycomb looks like when mounted in such an enclosure.

Figure 5: Honeycomb Mounted in Hammond 1593L Enclosure

3.3 Modbus Support

Modbus is a common industrial protocol that is physical-interface independent, but is typically used over an RS-485 or RS-232 link. An MCU library for Modbus on Arduino is available to simplify protocol adapter development.

4. Use Cases

4.1 Securely Connect a Group of RS-485 Panel Meters

For this first example we will connect an existing installation of industrial panel meters to the internet using Afero and Honeycomb. In this use case, the meters display whole-building power usage as part of a backup generator system.

There are five Fuji Electric DIN Panel meters installed:

- Two AC RMS Voltage meters, one for each 120V leg feeding the building
- Two AC RMS Ammeters, one for each 120V leg
- One Line Frequency meter

The meters are networked using half-duplex RS-485. These would traditionally connect to a PC or PLC. The RS-485 connection can be used for configuration and monitoring. We will use the Afero Honeycomb reference design to convert this RS-485 meter data into Afero attributes, which will then be securely sent to the Afero Cloud.

The unit looks like this:

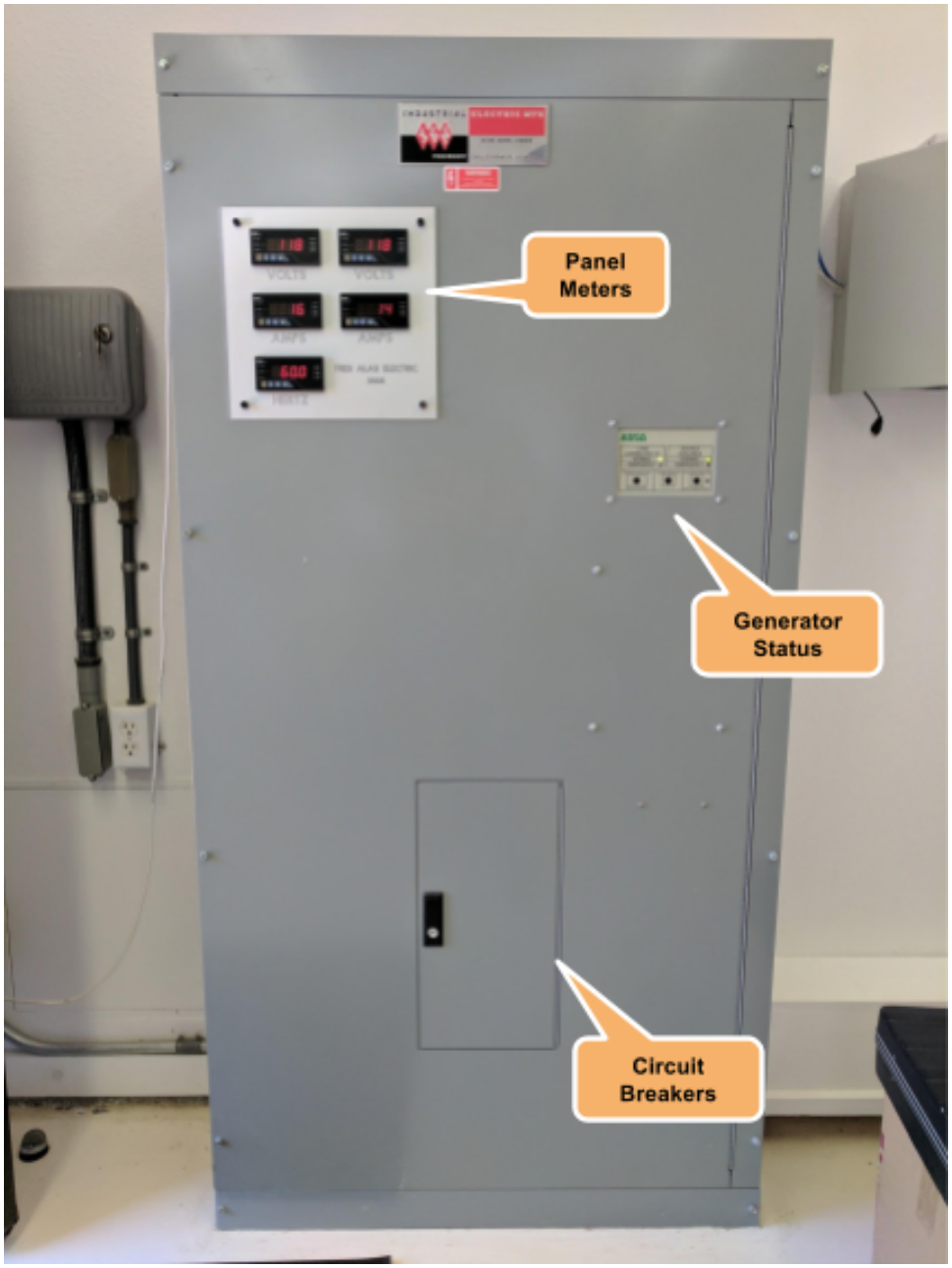


Figure 6: Meter Panel and Generator Control

Here's a closer look at the meters:

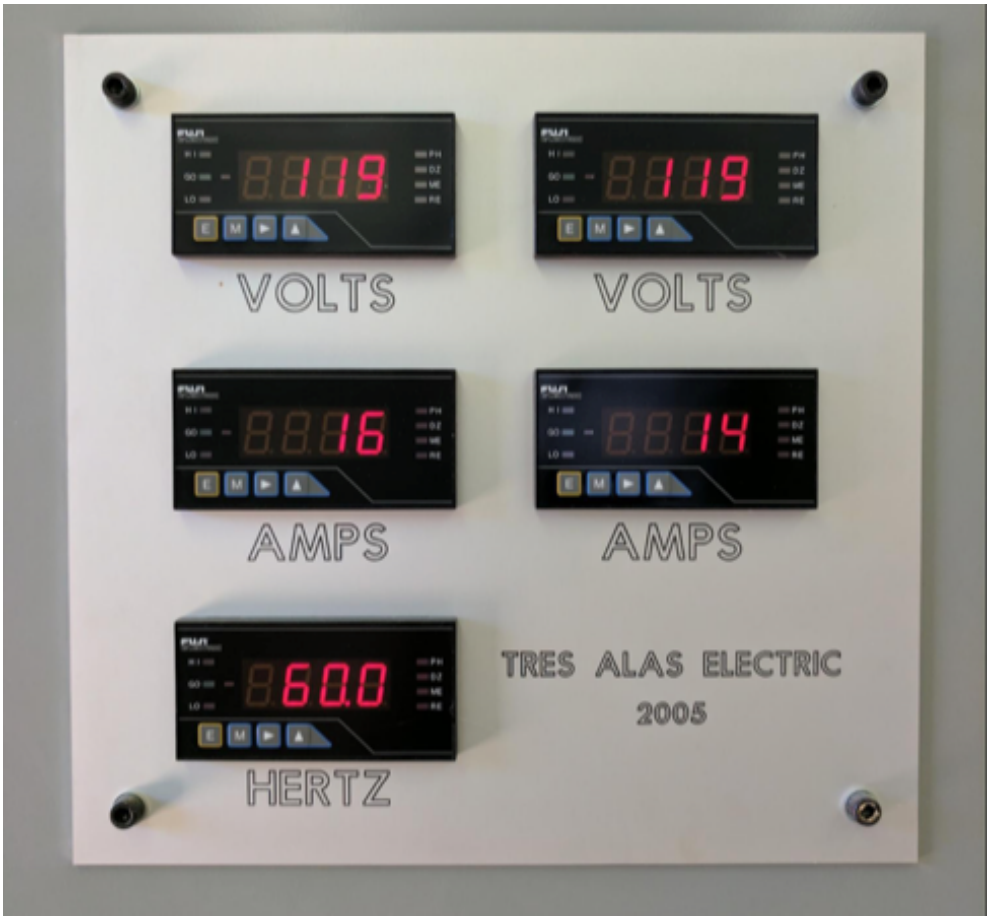


Figure 7: Power Status Meters

The meters contain half-duplex (2-wire) RS-485 serial interfaces. (See the installation manual.) RS-485 is a differential serial physical interconnect frequently used for industrial control applications. It is noise-immune, performs well with long cable runs, and is typically used with low-speed data (e.g., 9600 bps).

The meters are wired in a typical way:

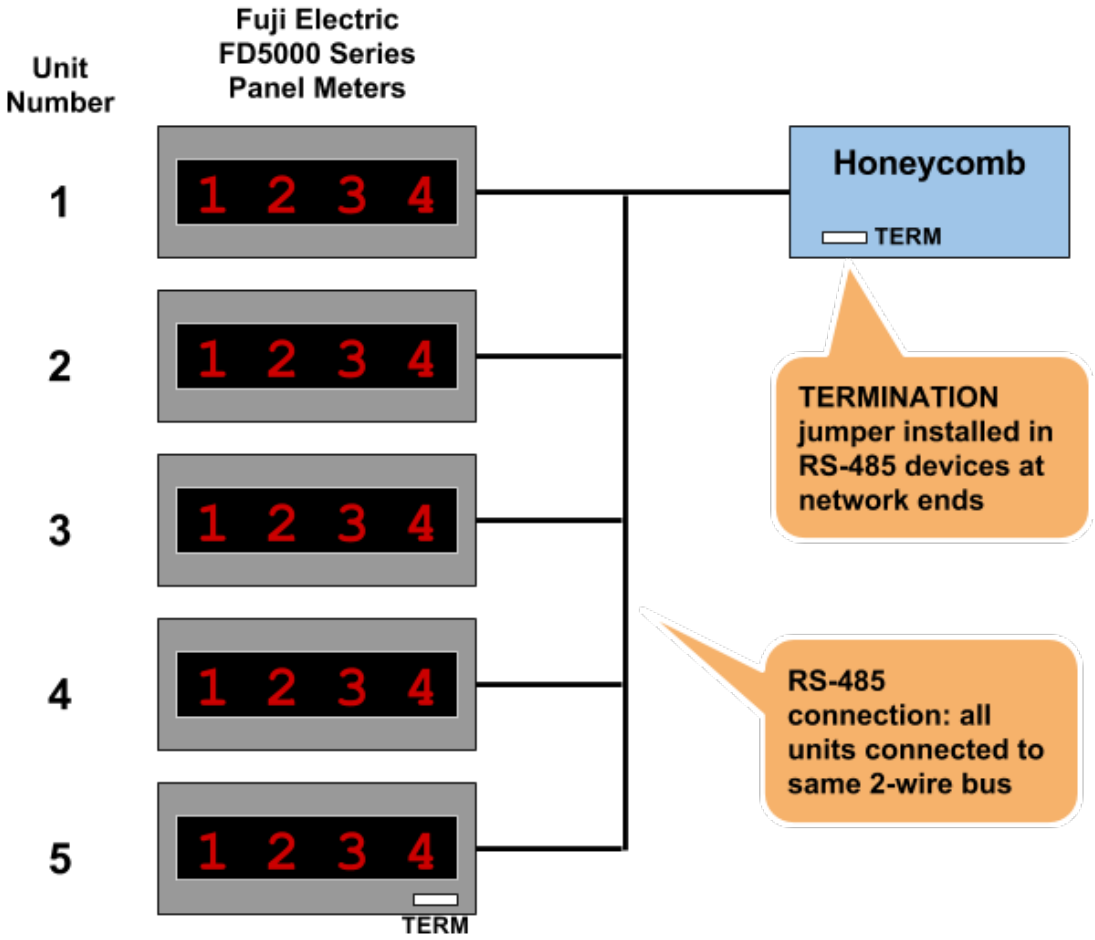


Figure 8: Meters All on Same RS-485 Bus

As you can see, all of the meters are connected to the same wires, which then connect to the RS-485 interface on Honeycomb.

These wires form an **RS-485 bus**, which is terminated with a resistor at each end. Notice how the TERM jumpers are installed in meter 5 and the Honeycomb devices, because they are at the ends of the bus.

Notice also that the meters have a **Unit Number**. This number is used as an **address** so the Honeycomb (the **RS-485 bus master**) can select a particular meter to read.

OK, so what about the software side? We have only two tasks:

1. Use the Afero Profile Editor to design a data model and mobile app UI.
2. Write C/C++ code on the Honeycomb MCU to "speak" the meter protocol.

4.1.1 Step 1: Design a Data Model

Afero auto-generates the embedded, mobile app, and cloud API components needed, using a data model you define.

A **data model** defines a set of data objects that are sent between the connected device and the Afero Cloud. We call these data objects **attributes**. Each attribute specifies:

- A name
- A data type
- The size of the data (sometimes implicit in the data type)
- A default value
- Whether it is read-only or read-write

You use a graphical tool called the Afero Profile Editor to describe the attributes in your data model to Afero.

These attributes are then referenced in the connected device, the mobile application, and the cloud via the Afero API. All other communication and security details are abstracted by the Afero API.

In the image below, you can see the Profile Editor being used to define the attributes for this application. We have defined six attributes:

- PollRate: How frequently the MCU reads the meters (not the data rate; data is only sent if it changes).
- Leg1Volts: An 8-bit value that holds Volt Meter 1's reading.
- Leg2Volts: Same as Leg1Volts, but for Volt Meter 2.
- Leg1Amps: An 8-bit value that holds Amp Meter 1's reading.
- Leg2Amps: Same as Leg1Amps, but for Amp Meter 2.
- LineFreq: A C-style (null-terminated) string that holds the line frequency reading, which is of the form xx.y, such as 60.0.

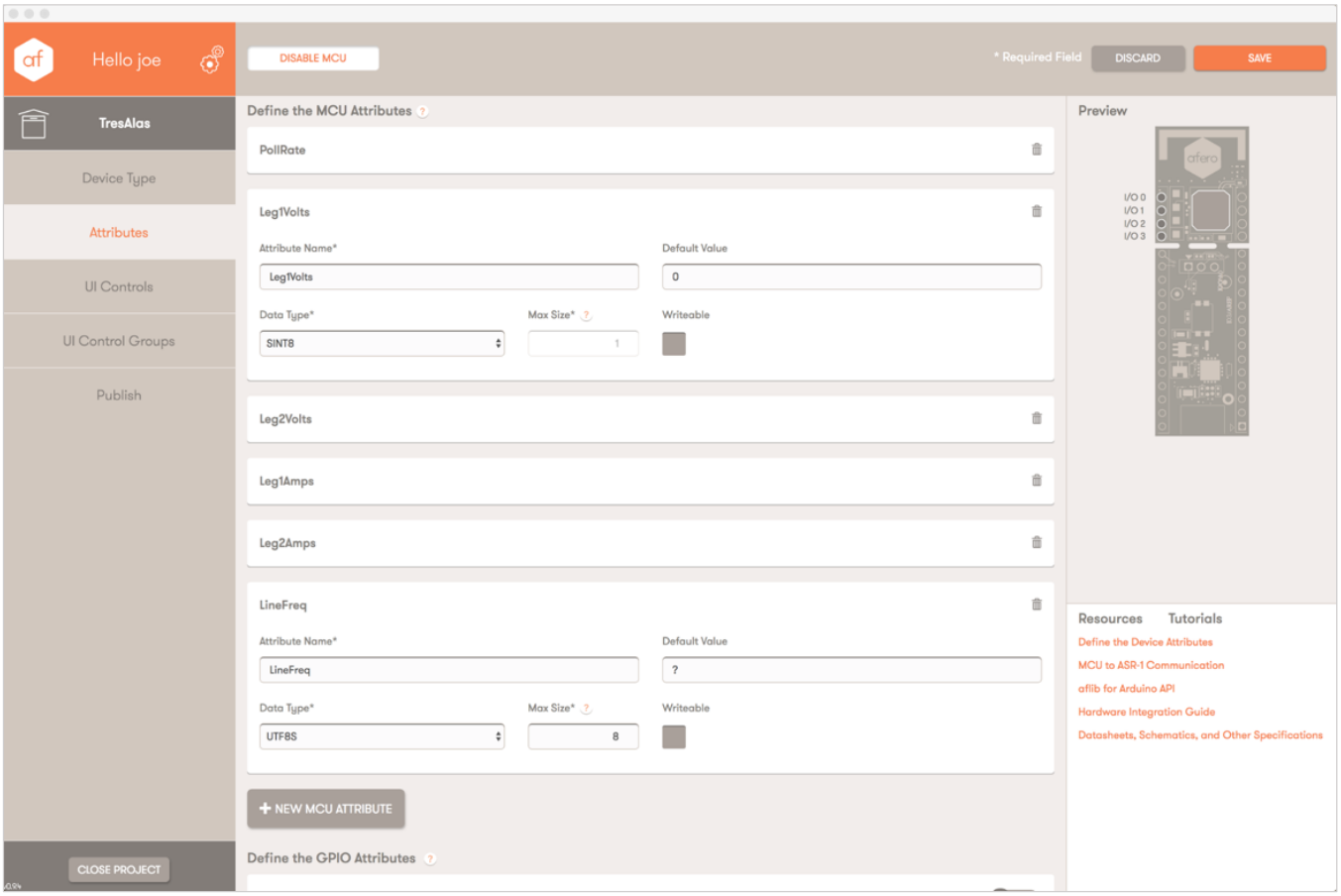


Figure 9: Attribute Definition in the Profile Editor

The Profile Editor will take these attribute definitions and use them to generate:

- **Device Profile**, which will be loaded into the Afero Secure Radio. With this, the MCU running the meter communication code only has to read and write attributes to communicate with the cloud.
- **Mobile App Profile**, which will be loaded into apps using the Afero SDK to render UI for the connected device. This can be used with the Afero Reference App to quickly build and deploy interactive mobile user interfaces **without writing any mobile app software**. See Figure 10 below, a screenshot of the UI that was automatically generated this way for our meter application.
- **Cloud Profile**, which is automatically deployed to the Afero Cloud servers to allow these attributes to be accessed via standard REST API. (Websockets are also available.)

Below is a screenshot of the mobile app UI created for this use case using the Afero Profile Editor. It is important to note that **no software** was written for this mobile app UI, and all meter readings are live. This is a great accelerator: you can get a fully-working mobile app UI built in just a few minutes.

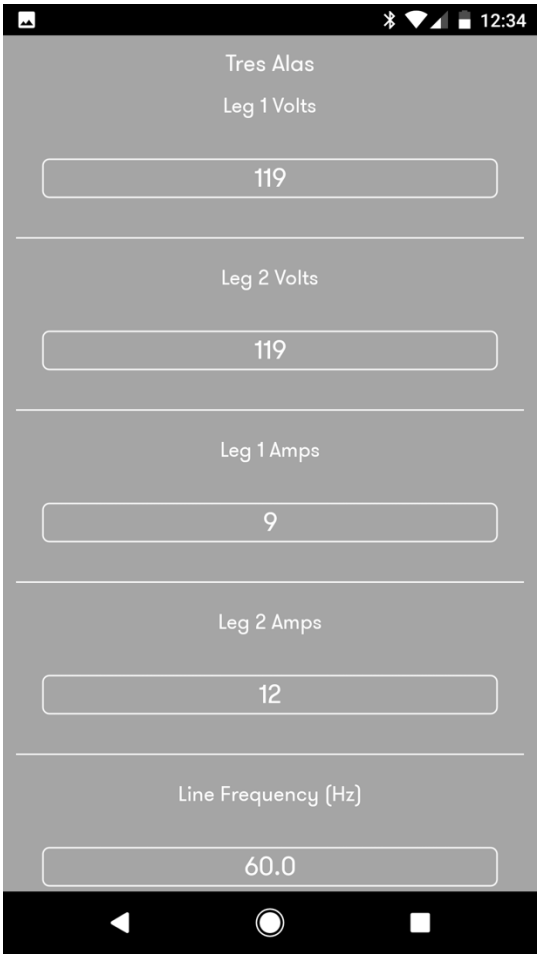


Figure 10: Meter Readings in Mobile App UI

It took less than 30 minutes to design and deploy our data model, and with that we have a live mobile app UI and a live cloud API.

The attribute definitions for this data model have also been downloaded into our Afero Secure Radio development board plugged into Honeycomb; it's ready for the Teensy MCU to read and write those attributes based on the meter readings.

So, now we just need to write the meter-specific interface code and connect the readings to these attributes!

Note that because of this encapsulation, we can use any kind of Afero Secure Radio development board:

- Modulo-1, which is Bluetooth® low energy only, or
- Modulo-2, which is Wi-Fi and Bluetooth low energy, or
- Any future Modulo!

No need to change any embedded, mobile, or cloud code: just plug in the radio we want to use.

4.1.2 Step 2: Write Meter Protocol Code

For the next step, we need to understand the meter's protocol. It is proprietary and described in this document.

This step is where the bulk of the development time was spent: understanding this protocol, figuring out what messages we need to send, and figuring out how to parse the results.

Reading a meter requires these steps:

1. Send the desired meter a "select" command (by Unit Number).
2. Get an ACK.
3. Send the "read measurement" command.
4. Get response.
5. Send a "deselect" command.
6. Parse response.

The flow of the Arduino sketch is as follows:

1. Initialize hardware.
2. Set up a timer to regulate meter polling.
3. Load last user-selected polling rate (stored in EEPROM)
4. Initialize Afero.
5. Fall into main loop.

The main loop then simply does this, forever:

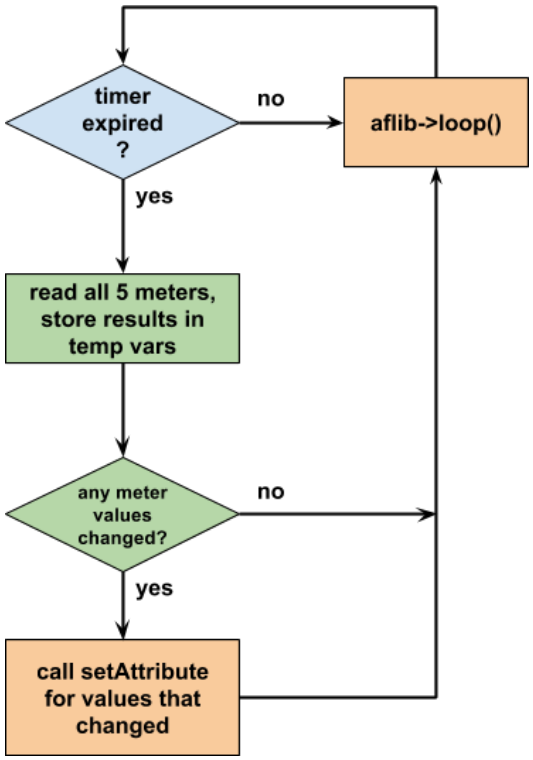


Figure 11: Main Loop Structure

A zipfile with the Arduino sketch and the Afero Profile Editor project can be found here.

The blocks above in **green** are the ones we need to write specifically for the meters.

The blocks in **orange** are provided by Afero, and take care of securely passing the meter data to the cloud, where it can be accessed by mobile apps, SaaS applications, and other web applications.

Done!

And with that, the meters are connected and securely sending their reading data to the cloud as it changes! This project took a little over one day to complete. The bulk of the work was learning and implementing the proprietary meter protocol. Here's a picture of the result, with Honeycomb sitting on top:

Figure 12: Honeycomb in Place, Securely Connecting Meters to the Internet

4.2 Securely Connect an RS-485 Industrial Air Quality Sensor

In this example we'll interface an existing industrial air quality sensor to Afero using Honeycomb.

This sensor is a self-contained unit, manufactured in China. (See Figure 13 below.) It uses a laser sensor and a fan to pull in air/particulates and provides PM1.0, PM2.5, and PM10 sensor readings via an RS-485 interface. Normally this would be tied into a factory or commercial building control system.

We will use Honeycomb to take readings via RS-485 from the sensor, and map those readings onto Afero attributes. Once that is done, the Afero system will securely push the readings up to the cloud.

We will also use an attribute to allow the remote setting of the rate at which Honeycomb polls the sensor for updated readings. The Teensy program/sketch only sends readings to the cloud when they change.

Figure 13: PPM 2.5 Dust Sensor, Based on Sharp Sensor

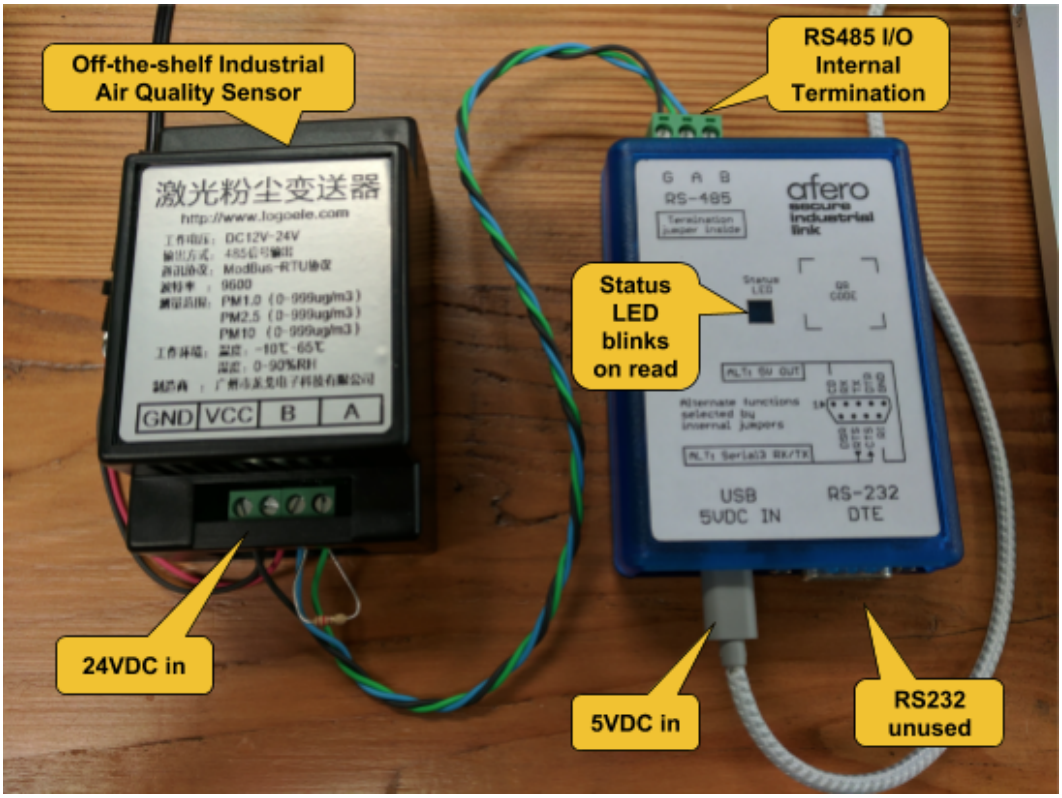


Figure 14: Hardware Setup

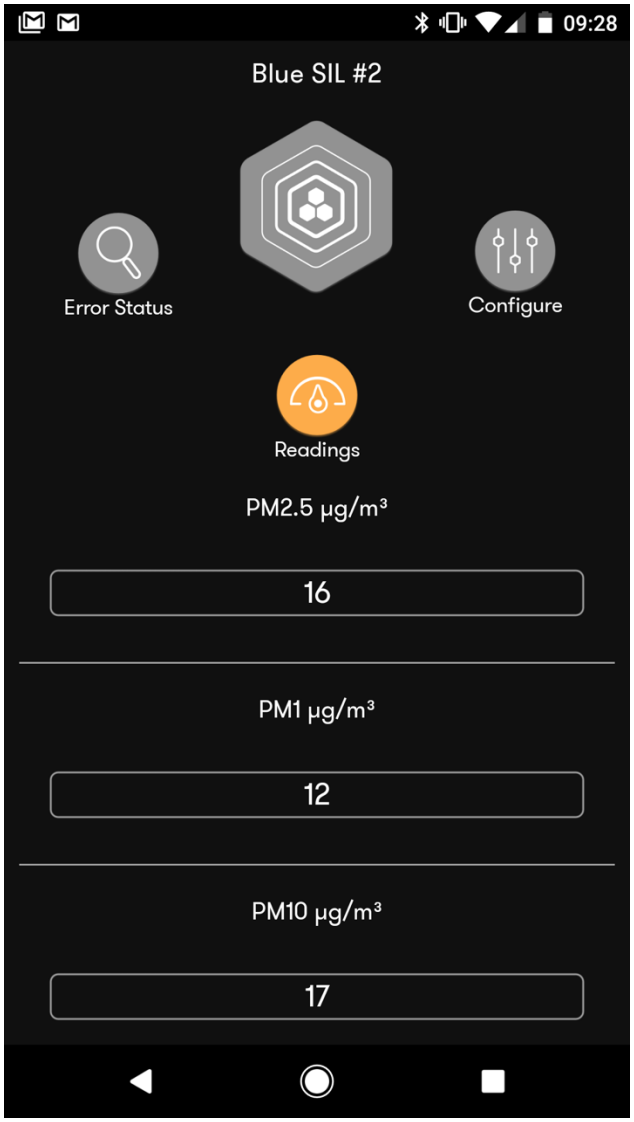


Figure 15: UI Built with Afero Profile Editor to Display PM Readings

To understand the meanings of these readings, the US EPA PM2.5 guidelines are reproduced below. Note that these are average readings over a 24-hour period. With this sensor connected to the cloud via Afero, analyzing this data is reduced to pulling data using the provided cloud API.

24-Hour $PM_{2.5}$ Standard ($\mu g/m^3$)

$PM_{2.5}$	Air Quality Index	$PM_{2.5}$ Health Effects	Precautionary Actions
0 to 12.0	**Good 0 to 50**	Little to no risk.	None.
12..1 to 35.4	**Moderate 51 to 100**	Unusually sensitive individuals may experience respiratory symptoms.	Unusually sensitive people should consider reducing prolonged or heavy exertion.
35.5 to 55.4	**Unhealthy for Sensitive Groups 101 to 150**	Increasing likelihood of respiratory symptoms in sensitive individuals, aggravation of heart or lung disease and premature mortality in persons with cardiopulmonary disease and the elderly.	People with respiratory or heart disease, the elderly and children should limit prolonged exertion.
55.5 to 150.4	**Unhealthy 151 to 200**	Increased aggravation of heart or lung disease and premature mortality in persons with cardiopulmonary disease and the elderly; increased respiratory effects in general population.	People with respiratory or heart disease, the elderly and children should avoid prolonged exertion; everyone else should limit prolonged exertion.
150.5 to 250.4	**Very Unhealthy 201 to 300**	Significant aggravation of heart or lung disease and premature mortality in persons with cardiopulmonary disease and the elderly; significant increase in respiratory effects in general population.	People with respiratory or heart disease, the elderly and children should avoid any outdoor activity; everyone else should avoid prolonged exertion.
250.5 to 500.4	**Hazardous 301 to 500**	Serious aggravation of heart or lung disease and premature mortality in persons with cardiopulmonary disease and the elderly; serious risk of respiratory effects in general population.	Everyone should avoid any outdoor exertion; people with respiratory or heart disease, the elderly and children should remain indoors.

Source: U.S. Environmental Protection Agency

Figure 16: US Evaluation of PM2.5 Levels

4.3 Securely Connect an RS-485 Industrial Temperature Controller

In this example we'll interface an existing industrial Temperature Control Unit (TCU) to Afero using Honeycomb.

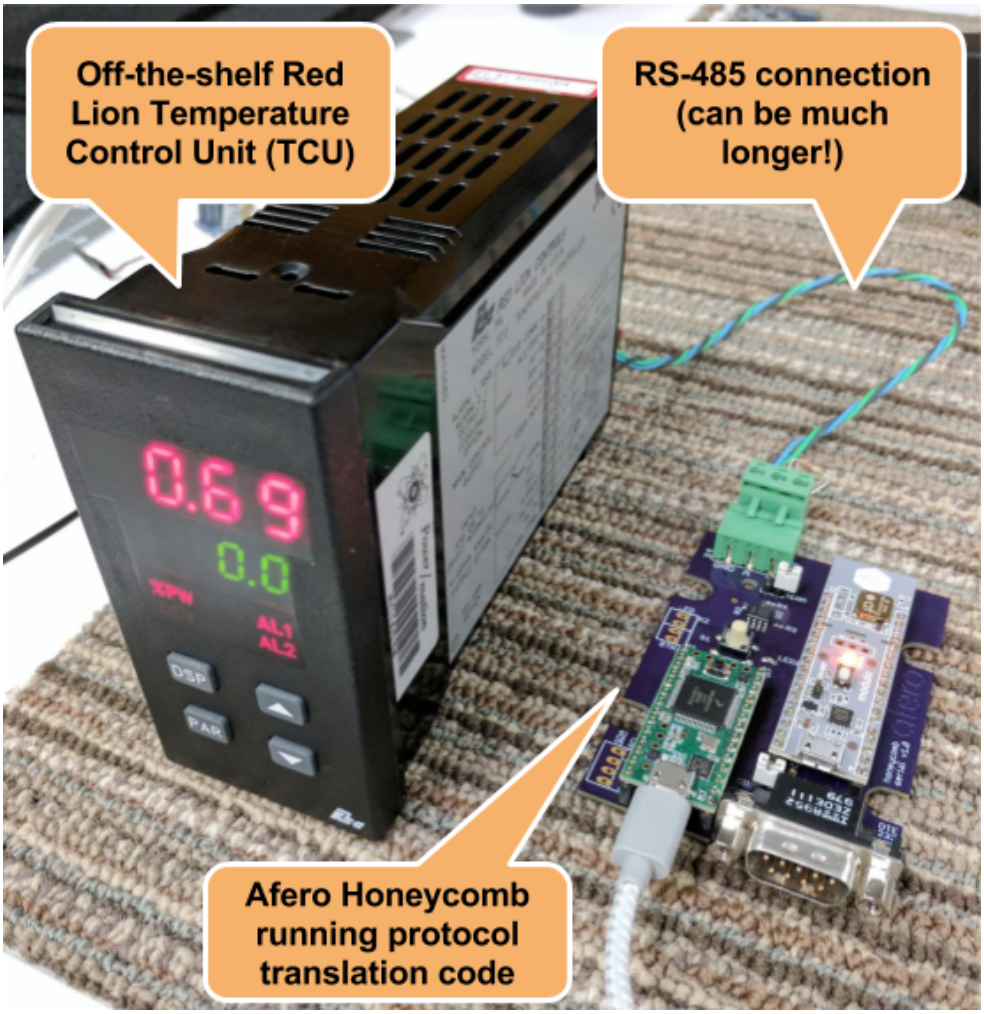
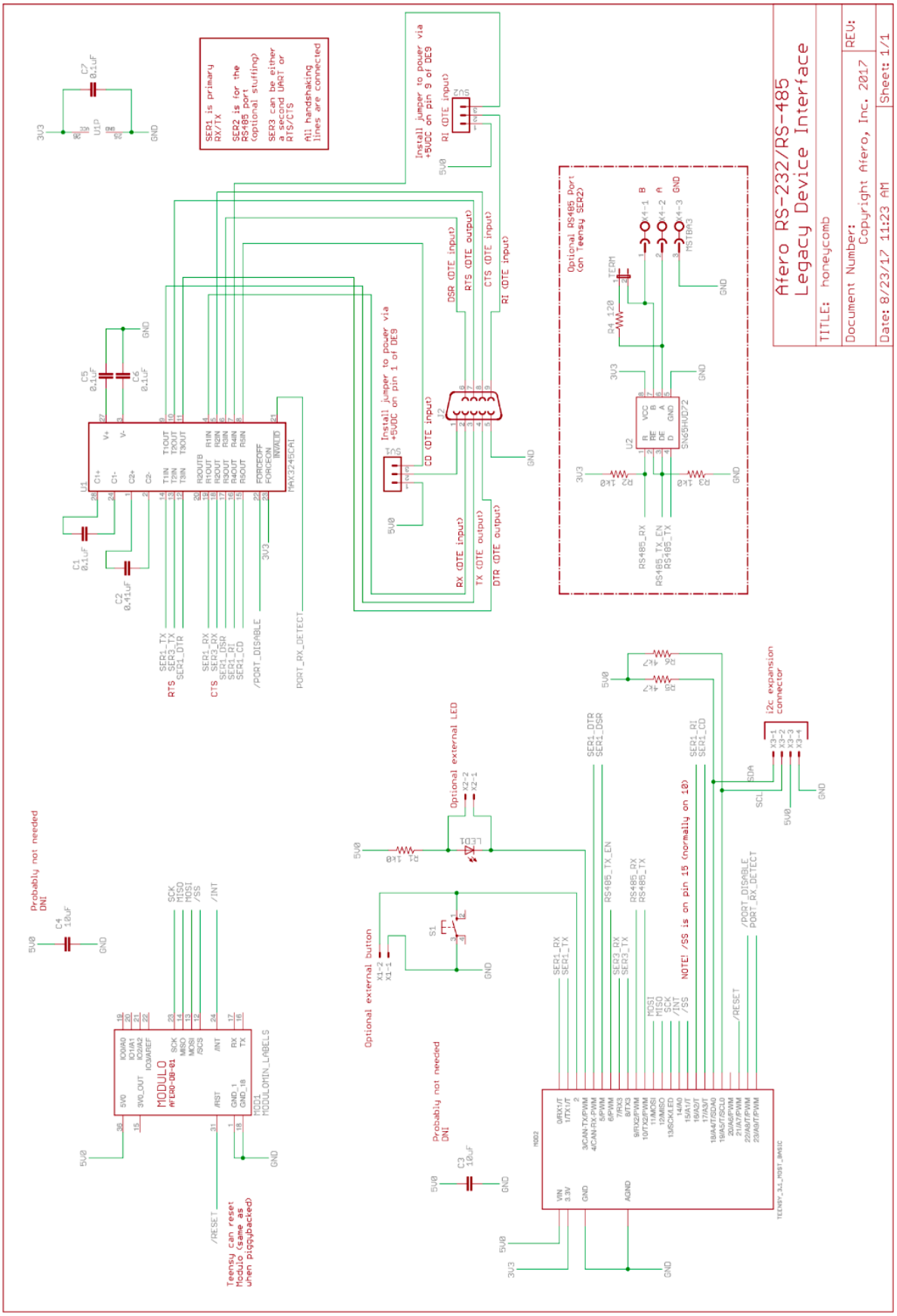


Figure 17: Honeycomb securely connects this industrial meter to the Internet

5. Schematic

You can download the schematic in PDF or view it on the page below. **Hint**: For better viewing, in Adobe Acrobat, rotate the image (View > Rotate View > Clockwise menu), then zoom in as needed.


Afero RS-232/RS-485
Legacy Device Interface
TITLE: honeycomb
Document Number:
Copyright Afero, Inc. 2017
REV:
Date: 8/23/17 11:23 AM
Sheet: 1/1
SER1 is primary RX/TX
SER2 is for the RS485 port (optional stuffing)
SER3 can be either a second UART or RTS/CTS
All handshaking lines are connected
Install jumper to power via +5VDC on pin 9 of DE9
Install jumper to power via +5VDC on pin 1 of DE9
Optional RS485 Port (on Teensy SER2)
MAX3245ECAI
SN65HVD72
MODULO
MODULOMIN_LABELS
Optional external LED
Optional external button
Probably not needed
i2c expansion connector
NOTE! /SS is on pin 15 (normally on 10)
Teensy can reset Modulo (same as when piggybacked)