

CONCEPT to Creation

Practical Applications for
Smart, Connected, and
Secure IoT Solutions



Table of Contents

3 Foreword

4 Engineers Eat Your Veggies

9 Automating and Controlling Greenhouse Lighting

14 Watchdog: Herding Cats Has Never Been Easier

19 Connecting the Microchip AC164160 AVR-IoT WG to Google Cloud IoT

26 Microchip Technology: Helping Connect to the Cloud

CONNECT WITH MOUSER



Terms Of Use: ©2020 Mouser Electronics, Inc., a TTI Berkshire-Hathaway company. Mouser and Mouser Electronics are registered trademarks of Mouser Electronics, Inc., Microchip, logo, product names, service marks, are trademarks, registered trademarks of Microchip. All other trademarks are the property of their respective owners. Names, products, specifications, links, resources, and similar were verified at the time of publication. Article content, reference designs, datasheets, conceptual illustrations, and other images provided herein are for informational purposes only.

DESIGNING A SMARTER FUTURE: Creating IoT-enabled Products to Change the World

Christina Unarut, Mouser Electronics

When we envision a smarter future, we consider smart speakers that play tunes in accordance with our moods, and smart bulbs that paint the room, foreshadowing the weather for the day to come. These friendly reminders signal the start of a new day, new possibilities, and new projects just waiting to be created.

Engineering a future filled with Smart Homes, Smart Cities, Smart Manufacturing, and Smart Agriculture is not only possible—but inevitable. However, “smart” applications are not just the self-serving, cool tech, futuristic functionalities that have become synonymous with the term.

The technology behind our smarter future already exists, and its applications will go beyond the niceties of our day-to-day lives; they will transform the way our society operates. For starters, using IoT to add intelligence and visibility to our commerce networks and supply chains will enable businesses and governments to improve efficiency and reduce waste. Many of the same tools will enable buildings to dramatically reduce their energy and water consumption while making their occupants more comfortable. Likewise, Smart Cities will be able to manage their infrastructure (electricity, water, sewage, and lighting) more effectively, and perhaps even improve the flow of traffic within their congested streets. With each of these applications, as well as a myriad of others, we’re presented with an opportunity to design,

manufacture, and sell products that never existed before—products that will improve the lives of countless users.

So, when we say “smarts” can change the world, we mean it.

Making Food Production More Efficient and Long-Lasting

Imagine applying smart technology to address one of mankind’s greatest needs: The ability to cultivate a more efficient and long-long-lasting food supply to sustain our growing population

Humidity, temperature, CO₂, and pH sensors, along with IoT capabilities, provide engineers with the ability to give society more efficient ways to cultivate crops and provide longstanding nourishment. Beyond this, vertical farming, and the inclusion of smart lighting, paves the way for local farming in urban environments, appealing to the need for the world to revamp its agricultural system over the next few decades in order to meet growing demand.


There are simple mechanisms to add this form of automation to systems these days. Better yet, advancements in these fields have managed to do this at a lower cost than ever before.

Remaining Vigilant and Secure in a Connected Future

As with any transformation, for all its promise, the road to an IoT-enabled future also poses some new and significant challenges to those who travel it. Since

many IoT applications involve mission-critical infrastructure elements, they must be highly secure, highly resilient, and highly resistant to cyber-hacking. Hands-on solutions exist so that you can prototype IoT applications and securely connect to the cloud.

Your Smart Doorbell Rings

Your Mouser package has arrived, and the newest Microchip Technology products are inside. So, what are you waiting for? Step inside to learn more about the IoT and get hands-on with Microchip Technology products and step-by-step project guides so you can change the world, too. 



Christina Unarut,
Mouser Electronics

Christina Unarut is a technical marketing engineer with Mouser Electronics. She joined the company after graduating with her bachelor's in electrical engineering. Today, she provides engineers with the latest information on products and emerging technology through various YouTube videos and e-Books at Mouser.



ENGINEERS Eat Your Veggies

**Technology Advancements
That's No Small Potatoes**

Paul Golata, Joseph Downing, and Michael Parks for Mouser Electronics

ATMEL STUDIO 7 TO MPLAB® X IDE



BLECH! VEGETABLES!

I know they are good for me, but for a long time, I had trouble eating them. I was basically a meat and pasta guy.

Perhaps you think I am crazy! My family definitely doesn't get my aversion to green, leafy things.

Maybe you love their infinite variety of flavors and good nutritional content.

As I age, I admit I like them more and more.

But onions, NO WAY!

Long ago, the Greek philosopher Aristotle (384–322BC) proposed three hierarchies of life forms. Aristotle's lowest level life form was the vegetative or nutritive form. This life form takes in food and is said to have an appetite. The highest form of life was supposed to be humankind, who is a rational animal, meaning that humanity uses intelligence in order to conform its beliefs and actions in the process of reasoning and thinking. In Aristotle's sense, then, I guess I should like vegetables since I take in food and am at least sometimes characterized as rational.

Technology is one component of culture. Technology is assisting humanity in the development of understanding how to cultivate our fields and gardens to produce the foods, plants, and materials that we need to survive and flourish. This cultivation is known as horticulture and agriculture, where *horti-* comes from the Greek word meaning garden and *agri-* derives from the word meaning "field." Microchip Technology is a leading provider of microcontroller, analog, FPGA, connectivity, and power management semiconductors. Microchip Technology provides technology that enables human cultivation.

Climate change, soil degradation, electric and water shortages, pests, severe weather conditions, and excessive nitrogen (N) all play havoc with the cultivation of gardens and fields. Technology allows gardeners and farmers to measure, control, conserve, and cultivate in ways previously not possible. Advancements in sensor technology and associated embedded electronics has driven the costs down significantly. Now, even a part-time greenhouse gardener that possesses only a fraction of the "do-it-yourself" spirit can become empowered to remotely monitor their garden. This project will lay out a design for a device to do just that. The goal is to enable almost anyone with basic electronics and software skills to have the ability to build this monitoring system.

Farmers can now see if their irrigation systems are working, measure soil moisture and temperature using wireless connectivity, and take weather forecasts through the internet into account. Irrigation systems can make critical decisions about where and when to water, fertilize, and use pesticides on the plant. The system includes electricity generation using solar power, running the water pump, interfacing with sensors, and wireless remote monitoring and control. Microchip Technology offers various Digital Signal Controllers (DSC) for solar power conversion and motor control. Let's examine some of the crucial variables that may be monitored to understand the cultivation environment and soil quality.

Temperature

Extremely high and low temperatures can have adverse effects on the health of plants. High temperatures can cause an imbalance in the photosynthesis and respiration processes, which can result in plant suffocation. Conversely, low temperatures can cause plant cells to freeze, which inhibits a plant from getting nutrients.

Humidity

High relative humidity levels prevent a plant from evaporating water as part of its transpiration process. It can also prevent the plant from drawing nutrients from the soil. If these conditions are permitted to continue, the plant could rot or suffer bacterial or mold infections.

Moisture Content

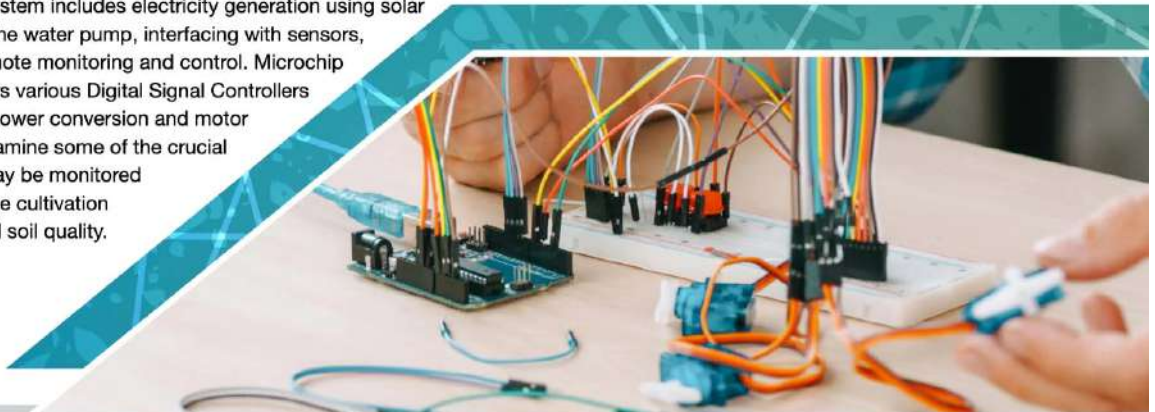
The amount of water in the soil must be maintained at optimal levels for ideal plant growth. Too much water can result in root rot, which prevents the plant from extracting oxygen from the soil. Too little moisture, and nutrients cannot circulate through the plant.

pH Level

In chemistry class, one learned the pH scale is a way to indicate the acidity or basicity of a substance. The scale ranges from 0.0 for acids such as battery acid up to 14.0 for bases such as lye. A pH level of 7.0 is considered neutral. While the ideal pH level can vary from plant to plant, optimum pH levels for soil is between 5.5 and 7.0.

CO₂ Level

Plants of all types absorb carbon dioxide (CO₂) as part of the photosynthesis process to synthesize food from water (H₂O) and CO₂. Oxygen (O₂) is given off as a byproduct. As with the other factors, too much or too little CO₂ can have adverse effects on plant growth and health. As a reference, 250–350 parts per million (ppm) is considered an average concentration in an outdoor ambient environment. Indoor environments with decent air circulation should see a level from 350–1,000ppm.





PROJECT MATERIALS AND RESOURCES

Whether you're a seasoned gardener or just starting, Microchip Technology and Mouser have assembled a project that can provide you with the means to create an ideal and efficient environment to cultivate plants for consumption or transplant. The sensors and IoT capabilities can help you monitor and make changes as needed to help anyone have a green thumb. Let's look at some of the critical items that get employed in this project:

- [Microchip Technology](#) ATSAM10-based STEMMA soil moisture sensor
- Bosch BME280 temperature and humidity sensor
- DFRobot SEN0249 soil pH sensor
- DFRobot [MG-811 CO₂ sensor](#)

Bill of Material (BOM)

Table 1 lists the [bill of materials](#) (BOM), if you wish to use a LoRA RF Development Tool.

QUANTITY	MOUSER P/N	DESCRIPTION
1	556-ATMEGA328PBXMIN	Development Boards & Kits AVR ATMEGA328PB Eval Kit
1	932-MIKROE-2225RF	Development Tools LoRa 2 Click
1	932-MIKROE-1978	Temperature Sensor Development Tools Weather click
1	485-4026	STEMMA Soil Sensor - I ² C Capacitive Moisture Sensor
1	426-SEN0249	Analog Spear Tip pH Sensor / Meter Kit (for Soil and Food Applications)
1	426-SEN0219	Analog IR CO ₂ Sensor
1	854-ZW-MM-20	Jumper Wires ZipWire MI-MI 20cm 40 UnzipWires x20cm
1	932-MIKROE-1097	PCBs & Breadboards BREADBOARD CLEAR S/A 830 POINTS

Table 1: This plant health monitoring project BOM lists all the components you will need.

Table 2 lists the BOM, if you wish to use a Wi-Fi Development Tool

QUANTITY	MOUSER P/N	DESCRIPTION	SCHEMATIC P/N
1	556-ATMEGA328PBXMIN	Development Boards & Kits - AVR ATMEGA328PB Eval Kit	U1
1	932-MIKROE-1769	Wi-Fi Development Tools (802.11) WiFi3 Click	J5
1	932-MIKROE-1978	Temperature Sensor Development Tools Weather Click	J2
1	485-4026	STEMMA Soil Sensor - I ² C Capacitive Moisture Sensor	J1
1	426-SEN0249	Analog Spear Tip pH Sensor / Meter Kit (for Soil and Food Applications)	J4
1	426-SEN0219	Analog IR CO ₂ Sensor	J3
1	854-ZW-MM-20	Jumper Wires ZipWire MI-MI 20cm 40 UnzipWires x20cm	N/A
1	932-MIKROE-1097	PCBs & Breadboards BREADBOARD CLEAR S/A 830 POINTS	N/A
1	485-757	4-Channel Bi Logic Level Eval Board	U2

Table 2: This plant health monitoring project BOM lists all the components you will need.

Additional Resources

Computing Resources

- Windows-based computer running MPLAB X IDE
- Wi-Fi network
- An account on Medium One, available for a fee
- [Wi-Fi GitHub Software Files](#)
- [LoRa GitHub Software Files](#)
- [AVR® and Arm® Toolchains \(C Compilers\)](#)
- [Microchip ATmega Series Device Support \(2.0.12\)](#)

Software Files

The Microchip Technology [Xplained](#) board is programmable using MPLAB X IDE. In addition to the prebuilt libraries that the parts vendors provide, the project will require additional files already included in the main project file. The first is the firmware file that runs on the Microchip Xplained board. A header (.h) file to store any sensitive data such as passwords and application programming interface (API) keys, which are things one should avoid sharing with the entire world on GitHub®, located in the ArduinoCore/Include folder. Lastly, there is an included library project (ArduinoCore) attached to the main project file that

will include everything necessary for each of the devices. The original project file was created utilizing Atmel Studio 7 and gets imported into MPLAB X IDE. To build this program, one will need to incorporate the AVR toolchain as well as a device support pack. One can find the primary project file in the SmartAgFirmware folder.

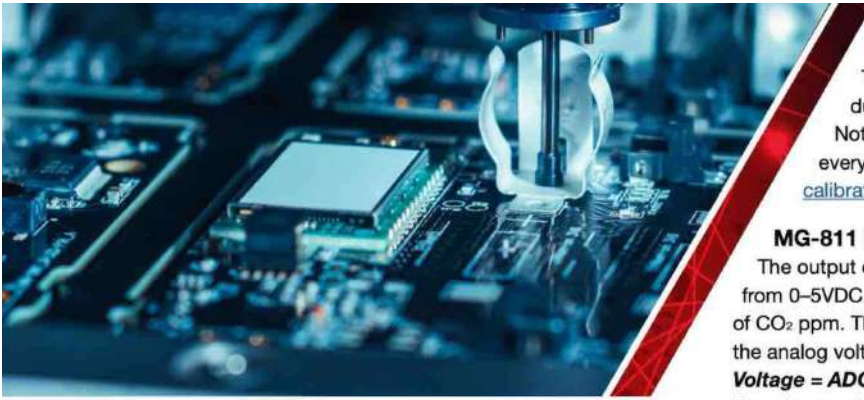
Tools and Other Resources

Finally, these tools are recommended to help complete the project:

- Wire strippers
- Digital multimeter
- Needle nose pliers
- Breadboard
- Pull-up resistors if needed for I²C bus

Project Technology Overview

The sensor boards used in this project communicate back to the Xplained microcontroller board in two ways: By digitally utilizing the Inter-Integrated Circuit (I²C) serial communication protocol or via an analog signal that is read by the microcontroller's analog-to-digital converter (ADC).



The offset is a variable that gets determined during the calibration process for the pH sensor. Note that this sensor must get recalibrated every two (2) hours. Find [more information on the calibration process](#) on the DFRobot site.

MG-811 CO₂ Sensor

The output of the MG-811 CO₂ Sensor is an analog signal from 0–5VDC. The unit of measure is the concentration of CO₂ ppm. The following formulas are used to translate the analog voltage value to a CO₂ concentration in ppm: **Voltage = ADC x (5V/1,024)** where ADC is the value read by the microcontroller's ADC pin connected to the MG-811 CO₂ sensor.

The formula assumes the microcontroller has a 5VDC operating voltage and 10-bit ADC. These numbers may require an adjustment if you use an alternative microcontroller with a different operating voltage or ADC resolution. If the calculated voltage is less than 0.4V, then the sensor has not yet warmed up enough, and no readings should occur at that time. Once the voltage rises above 0.4V, the following formula can be useful to make the final CO₂ concentration calculation: **CO₂ Concentration (ppm) = [(Voltage – 0.4) x 5000]/1.6**

Complete Project Information

Want to grow vegetables, herbs, and fruits year-round? For more project design information utilizing the Xplained board series from Microchip Technology, so you can take advantage of the fruits of their labor without having to get your hands dirty, click on the link below:

To learn more about creating this project with a RF LoRA development tool please follow this link.

<https://www.mouser.com/applications/building-smarter-farm-with-microchip/>

To learn more about creating this project with a Wi-Fi development tool please follow this link.

<https://www.mouser.com/applications/smarter-green-thumb-powered-by-microchip/> 

STEMMA Soil Moisture Sensor

The STEMMA Soil Moisture Sensor Communicates via I²C. The sensor measures the capacitive difference between the two probes and reports and has an integer value from 300 to 500 based on the moisture content of the soil.

BME280 Temperature Sensor

The BME280 Temperature Sensor Communicates via I²C. It reports a floating-point number with a unit of measure of degrees Celsius. The value is convertible to degrees Fahrenheit using the formula: **F = C x 9/5 + 32**

BME280 Humidity Sensor

The BME280 Humidity Sensor Communicates via I²C. It reports a floating-point number with a unit of measure in the percentage of relative humidity.

SEN0249 Soil pH Sensor

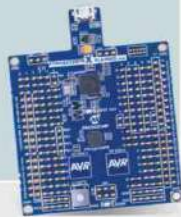
The output of the SEN0249 Soil pH Sensor is an analog signal from 0–5VDC. The pH is a unitless measure that will range from 0 to 14. The formulas to calculate the pH level are: **Voltage = ADC x (5V/1024)** where ADC is the value read by the microcontroller's ADC pin connected to the pH sensor.

The formula assumes the microcontroller has a 5VDC operating voltage and 10-bit ADC. These numbers may require an adjustment if you use an alternative microcontroller with a different operating voltage or ADC resolution. Then calculate the pH level: **pH Value = (3.5 x Voltage) + Offset**

ATMEGA328PB 8-Bit Microcontroller Evaluation Kit

- On-board debugger
- Auto-ID for board identification
- Access to all signals on target MCU

[LEARN MORE](#)



MPLAB® X Integrated Development Environment (IDE)

Atmel Studio 7 to MPLAB® X IDE



 MICROCHIP
Atmel Studio





- Data Visualizer
- Helpful Design Resources
- I/O View

[LEARN MORE](#)



AUTOMATING AND CONTROLLING Greenhouse Lighting

Michael Parks, P.E., for Mouser Electronics

In a previous horticulture-focused project, we examined how temperature, humidity, moisture, pH levels, and CO₂ levels affect plant growth. Here, we examine another key ingredient for plant photosynthesis, which is exposure to light. This follow-on project will allow those with a green thumb to monitor and remotely control artificial light exposure. Controlled Environments (CE) horticulture is a fancy way of saying growing plants with the aid of digital technology. It is becoming an increasingly important mechanism to assist in stabilizing the food supply chain for the planet. In some scenarios, like indoor vertical farms, access to sunlight is not a guarantee. Smart use of artificial lighting is crucial in such situations.

This project will utilize Medium One, an end-to-end IoT cloud platform, to allow an end user to monitor, automate, and remotely control the artificial lighting in a horticulture-oriented facility, be it a greenhouse, vertical farm, or some other environment that requires such functionality.

OVERVIEW

The sensor and controller boards used in this project communicate back to the Xplained microcontroller board in two ways—either digitally utilizing a pulse width modulated signal or a 0V-5V analog signal that is read by the microcontroller's analog-to-digital converter (ADC).

1. MIC3202 HB LED Driver: The MIC3202 has two control pins that operate at logic-level voltage. The first is the EN or ENABLE pin. This pin controls the output (-LED and +LED) being on or off. If the EN pin is drawn high, then power is delivered to the LED power wires. Conversely, if the EN is drawn to ground, the output voltage to the LEDs is cutoff, turning off any LEDs connected to the output.

The second pin, the DIM pin, controls the brightness of the LEDs. The DIM pin looks for a Pulse Width Modulation (PWM) signal to serve as the control signal for the LED brightness. Varying the duty cycle of the PWM signal will result in varying brightness—100 percent duty for full brightness and 0 percent to turn the LEDs off.



Medium One offers easy-to-use resources to create a basic smartphone app to interact with IoT devices.

2. **TEMT6000 Ambient Light Sensor:** This sensor acts like an NPN transistor with the base terminal controlled by the exposure to light. It is configured in a common collector (CC) amplifier circuit topology to allow for subtle changes at the input to drive a larger output signal to the collector emitter (CE) junction going to the microcontroller ADC pin. The brighter the light, the more current and thus a higher voltage on the microcontroller ADC input pin.

SOFTWARE

In this section, we will detail the software involved in this project. This will include the firmware and support files necessary for the Microchip Technology Xplained board as well as the setup of the Medium One Sandbox in your web browser. The source code provided for this project can be edited using Atmel Studio 7 or the Arduino IDE depending on your preference.

To use the Arduino IDE, Microchip Technology provides the needed files on their GitHub site. The following URL will need to be added to the following location in the Arduino IDE: **File → Preferences → Additional Boards Manager URLs**. The URL is linked [here](#).

The software for this project is divided across three parts.

1. Microcontroller firmware is written in C.
2. Cloud application to send and receive commands to/from the end users smartphone and the microcontroller. This code will be written in Python.
3. Smartphone application provided by Medium One and will be configured specifically for this project.

Microchip Xplained Board Firmware

The firmware that runs on the microcontroller board is straightforward. The code does the following, in this order:

Setup

1. Establish a serial debug connection to the host computer at 9600 baud.
2. Attempt to connect the desired wireless network.
3. Connect to the MQTT broker.
4. Subscribe to the MQTT broker submit a topic for any messages sent from Medium One to the microcontroller.
5. Set the various I/O pins as inputs or outputs.

Main Repeating Loop

6. Poll the MQTT server with a heartbeat message to remain connected.
7. Take a reading from the TEMT6000 ambient light sensor.
8. Determine if the system is in manual or automatic mode.
9. If in automatic mode, adjust the LED brightness by mapping the 0 to 1023 digitized brightness signal to a 255 to 0 value that will control the duty cycle of the PWM signal sent on the DIM pin.
10. If in manual control mode, adjust the LED brightness by mapping the 0 percent to 100 percent setpoint set by the user on their smartphone to a 255 to 0 value that will control the duty cycle of the PWM signal sent on the DIM pin.

11. Check to see if at least 1000ms has elapsed since the last message has been sent to the MQTT broker. If so, transmit a new message to the broker with the latest reading from the ambient light sensor.

Handle Any Messages Received from the MQTT Broker

12. Print the received message onto the serial port.
13. Convert the payload to a string.
14. Determine if the string is requesting to toggle the manual control on or off; or if it is an integer that should be assigned as the manual brightness set point.

A quick note on the map function used in the project. The function definition for the map is the following: **map(value, fromLow, fromHigh, toLow, toHigh)**

The map function is very useful for re-mapping one range of numbers to another range of numbers. For example, the ADC has a resolution of 1024, whereas the analog Write function only accepts values of 0 to 255. The map function takes five arguments including:

1. The number to be scaled.
2. The low-end of the range the number is being scaled from.
3. The high-end of the range the number is being scaled from.
4. The low-end of the range the number is being scaled to.
5. The high-end of the range the number is being scaled to.

The map function returns an integer that has been re-mapped to the new scale. For example: **X = map(50, 1, 100, 1, 200);**

In this case, x would be set to 100 because 50 is halfway between 1 and 100 on the old scale and 100 is in the middle of the new scale.

Project Files

The following files can be found in this [project's GitHub repository](#).

- **MicrochipHorticultureLighting.ino**: The project-specific code for this effort is stored in this file. It is based upon an example that is provided by Medium One to demonstrate how embedded devices can interact with the IoT platform backend.
- **secrets.h**: When working on a project that is made publicly available, there is always a risk of exposing sensitive data such as passwords or API keys. Instead of hardcoding such information directly into the firmware and having to remember to alter the code prior to each Git commit, we can create an unpublished header file to store this information.

Libraries

The preprocessor directive `#include` lets us add libraries into our projects. This promotes code re-use. Unless you have very specific needs, re-inventing the wheel is not necessary. This project uses the following libraries (**Figure 1**):

- **ArduinoMqttClient.h**: This library provides a convenient interface to MQTT-based services.
- **WiFi101.h**: This library provides the interface to the ATWINC1500 chip to simplify interacting with Wi-Fi® networks.

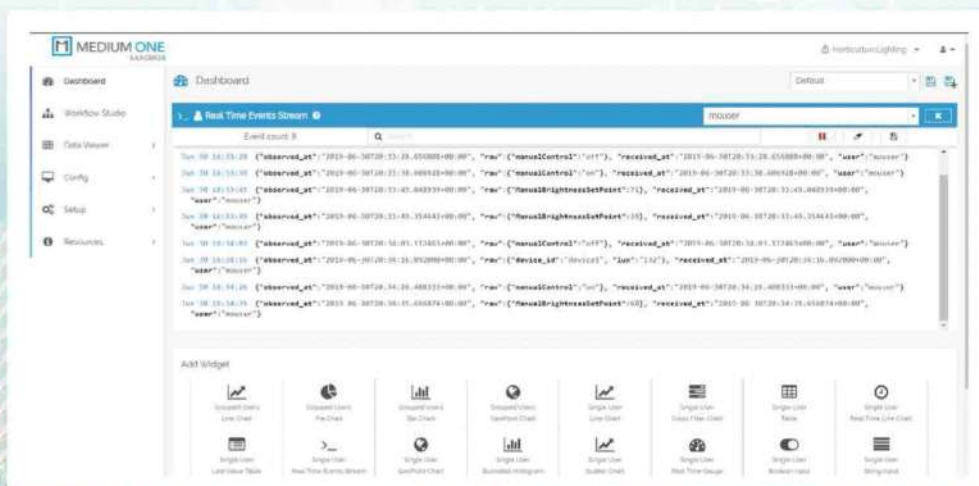


Figure 1: The raw data stream from the microcontroller and smartphone application. (Source: Mouser Electronics)

Configuring Medium One

Medium One provides IoT developers a platform where their IoT devices can communicate and compile data sets from across multiple IoT devices (Figure 2). They provide an excellent tutorial on getting started with their service. This guide will focus instead on the gotchas that we experienced in setting up the Microchip Technology Xplained board to communicate with Medium One via their MQTT API. This project will leverage the Medium One's MQTT (Message Queuing Telemetry Transport) protocol. MQTT is a publish-subscribe-based messaging protocol. It sits atop the TCP/IP protocol. MQTT, as opposed to RESTful APIs, requires a centralized message broker. Thus, endpoint devices cannot directly communicate with each other. This has pros and cons. Whereas RESTful APIs rely on the client to always initiate communications, MQTT allows a server to push data, thanks to the publish/subscribe (Pub/Sub) architecture. RESTful devices can communicate with each other directly. MQTT relies on a centralized message broker (e.g., a server in the cloud) and is much more efficient for transmitting this type of telemetry data.

Cloud and Smartphone Application

Medium One offers easy-to-use resources to create a basic smartphone app to interact with IoT devices. The apps are currently only available for iOS devices, but Android support is expected.

For this project, download the "IoT Controller" by Medium One for the Apple iOS App Store (Figure 3).

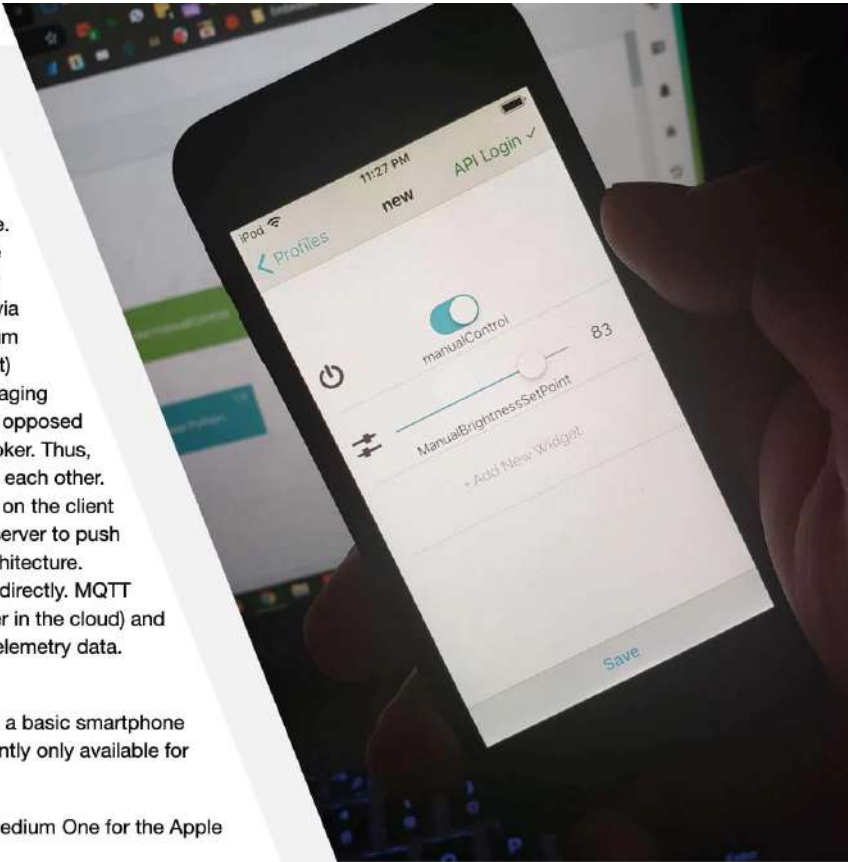


Figure 3: IoT Controller by Medium One (Source: Mouser Electronics)

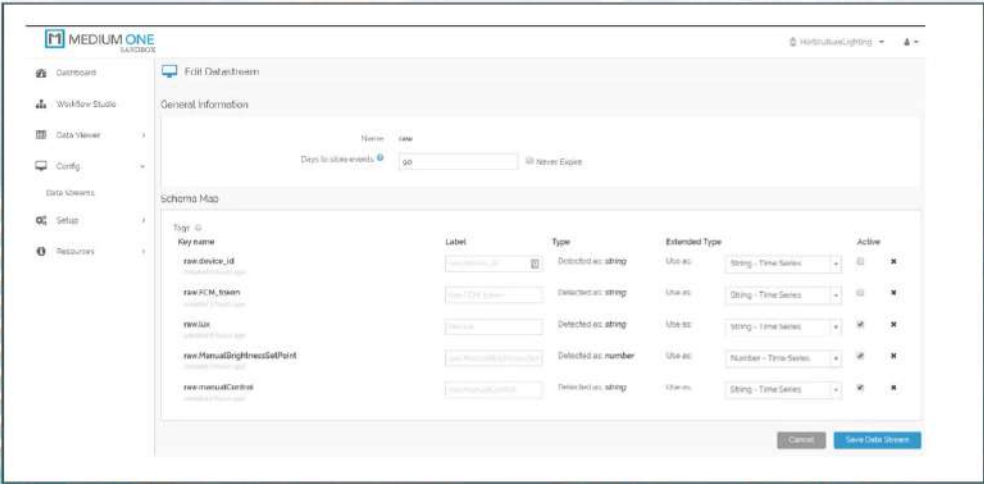


Figure 2: Configuration screen for Medium One. (Source: Mouser Electronics)

PROJECT IN ACTION

Once the project is assembled, hook up the Xplained board to a computer and fire up a serial terminal (**Figure 4**). You should see telemetry from the light sensor flowing as well as MQTT messages to and from the Medium One MQTT broker. If not, check to see if there are any error messages. Some common errors include:

1. Mixing up the line, neutral, and ground wires feeding the power supply.
2. Reversing the V+ and V- wires on the power supply or LED driver board.
3. Reversing the LED- and LED+ wires on the LED light strip or the LED driver board.
4. Reversing power and ground wires to sensors.
5. If powering the device, ensure the power supply can deliver at least 1A of current.
6. The wireless network is not running, or SSID or security key was entered incorrectly. 🚫

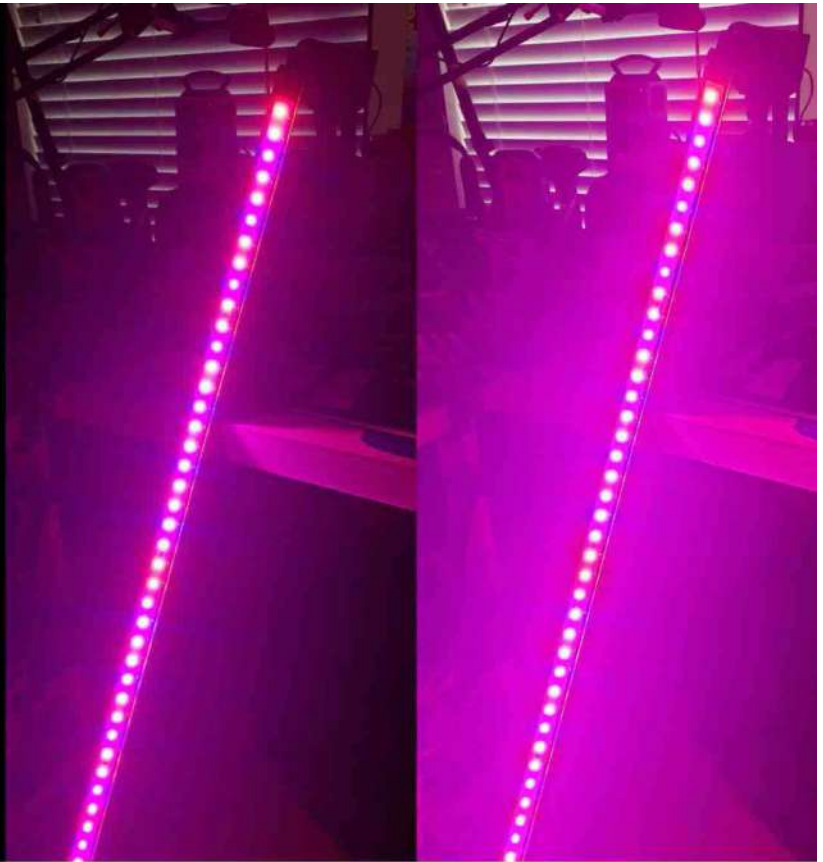


Figure 4: Left: Brightness at approximately 50 percent. Right: Brightness at approximately 90 percent. (Source: Mouser Electronics)

Medium One provides IoT developers a platform where their IoT devices can communicate and compile data sets from across multiple IoT devices.

ATSAMW25-XPRO Evaluation Board

- 33.8mm x 14.9mm compact footprint
- ATECC108A CryptoAuthentication™ engine
- Over-the-air updates

[LEARN MORE](#)

MIC3202 HB LED Driver Evaluation Board

- Dedicated dimming control input
- $\pm 5\%$ LED current accuracy
- 6V to 37V input voltage range

[LEARN MORE](#)



WATCHDOG: Herding Cats Has Never Been Easier

Braddock Bresnahan for
Mouser Electronics

I don't have kids, but I hear they can be a pain in the rear. My heart goes out to the parents carrying three infants wailing at the top of their lungs (the parents wailing, not the children). It reminds me of a time when I was younger: My older brother, my younger sister, and I (all one year apart) were sitting in the car. I don't know why, but all of us were crying at the same time. My mother was driving and, apparently, we were so annoying that we tipped her over a bit. She pulled over, turned around, screamed, "AAAAAAHHH!" and turned back forward. We all went dead silent. She kept on driving and none of us made another sound for the rest of the ride.

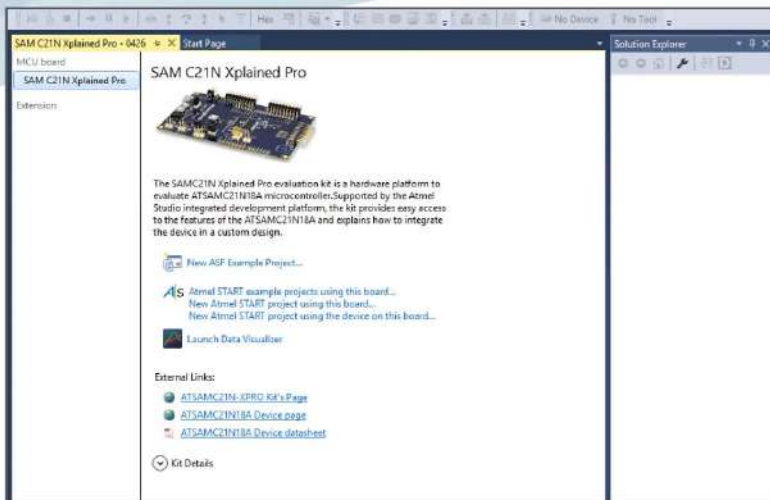


Figure 1: Atmel Studio will detect the board. (Source: Mouser Electronics)

Parenting is a job in itself. Kids can wear you down and down (according to my parents, I was a pro at it). Most parents are willing to take on the extra work, but there is a limit to a person's energy, and there's only so many different things one can pay attention to at the same time. Anything that can offload some of that work would benefit both parents and their children. For this project, I wanted to make something that would ease the burden of parenthood. That was the inspiration behind Watchdog.

Imagine taking your kids to the amusement park or mall, where they'll eagerly wander and explore...amongst the crowds of people and various visual obstacles that keep them out of your line of sight. I remember once as



With Watchdog, my parents and I would have immediately known I was trailing off.

a kid being at a theme park and accidentally following the wrong family around. Watchdog is a wristband-like device that is worn by a parent and their child(ren). Whenever the child and parent go beyond a specific range from each other, a vibration motor alarm would go off, letting them know that they've gone too far. With Watchdog, my parents and I would have immediately known I was trailing off. In this project article, you'll see how to build Watchdog.

DEVELOPING WATCHDOG

In the following sections, we walk you through the process of developing Watchdog.

Configure ATSAMC21N Xplained Pro

We will need to configure the ATSAMC21N Xplained Pro, the BMI160 Accelerometer, and the RN4870 for this project. First, install the drivers for the ATSAMC21N Xplained Pro:

1. Open Atmel Studio.
2. Plug in the [ATSAMC21N Xplained Pro](#) using the micro USB port.

[Atmel Studio](#) will automatically detect the board. As soon as you see a page pop up with the board's name and information, we're ready to start (**Figure 1**).

Configure Drivers

When you're in an Atmel START project, on the left there is a section called **DASHBOARD**. Click that if it is not already selected (blue). Here, we can add, remove, and configure drivers and components for the ATSAMC21N Xplained Pro to use (**Figure 2**). After we add a software component to the project, we can configure it by selecting it; under the diagram it will list all the configuration options that component has. We can get information, rename, or remove it with the buttons to the left of the configuration.

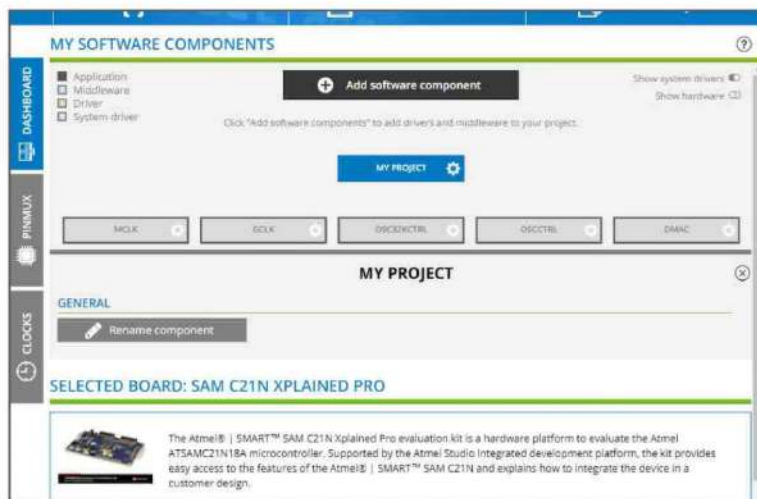


Figure 2: The START dashboard lets you configure board components.
(Source: Mouser Electronics)

Figure 3: The START PINMUX lets you set all pins' functions.
(Source: Mouser Electronics)

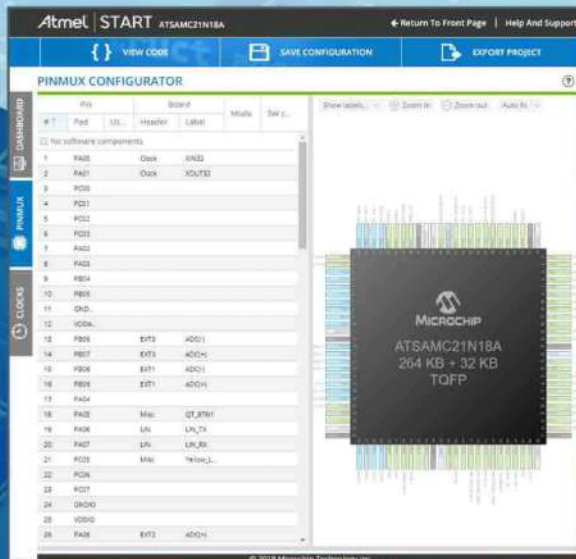
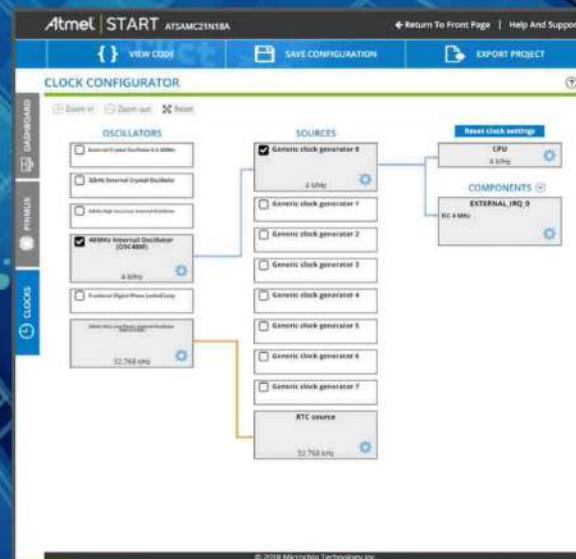


Figure 4: The START clocks let you configure oscillators and clocks.
(Source: Mouser Electronics)



Configure Pins

On the left-hand side of the dashboard, click on the **PINMUX** tab. Here, we can configure every pin on the ATSAMC21N18A microcontroller (**Figure 3**). You can configure a pin by selecting it in the list or by clicking it on the diagram. We can give each pin a unique name to easily reference the pin while coding. We can also select input/output pin modes as well as pull-up or pull-down resistors for each pin. Some pin functions are set using the driver configurations in the dashboard (I²C pins, UART pins, etc.).

Configure Clocks

The **CLOCKS** tab, on the left, allows you to configure... the clocks. Here, we can configure the internal and external oscillators as well as configure the clocks used by the ATSAMC21N Xplained Pro (**Figure 4**). We can add or remove oscillators and clocks by checking the box next to their names. We can also configure each by clicking the gear under their name. Atmel START shows you which oscillators and clocks are needed for the board with dotted arrows and dynamically changes when components are added or removed.

Configure BMI160

Next, we will configure the BMI160. Congratulations! We are now done configuring the BMI160 (it doesn't need configuring). Head on to the next section.

Configure RN4870

You only need to configure the RN4870 if you're planning to use the RN4870 Firmware Update Tool or CoolTerm for debugging. If you skip this section, you will need to find the *Bluetooth*[®] addresses of each band and write them in **usart.h** in **BLE_DEVICE_ADDRESS**.

Some instructions might change depending on if you are using the **RN-4870-SNSR** or the **MIKROE-2543**. For these instructions, we will focus on the RN-4870-SNSR.

Install Driver

Before anything else, the RN4870 driver needs to be installed. To install the driver:

1. Make sure the RN4870 is not in developer mode by setting the switch, at the bottom of the board, to 1 (OFF).
 - When set to ON (ON) the RN4870 is in developer mode and, if plugged into the computer, may corrupt the driver installation.
 - If the status light, when powered, is solid red, then it's in developer mode.
 - If the driver is corrupted, you will need to uninstall the driver and run through the process again, which can be problematic if you don't have administrative privileges on your work computer (I learned the hard way).

2. Plug in the RN-4870-SNSR using the micro USB port on the board.
3. The driver will install automatically when plugged in.

Update Firmware

To use the RN4870 with CoolTerm, it's recommended that we update the firmware using the firmware update tool (**Figure 5**).

Voila! The firmware is updated.

Configure CoolTerm

CoolTerm is a great debugging tool to use with the RN4870. Since the RN4870 uses an ASCII command interface, everything the module can do we can do through CoolTerm.

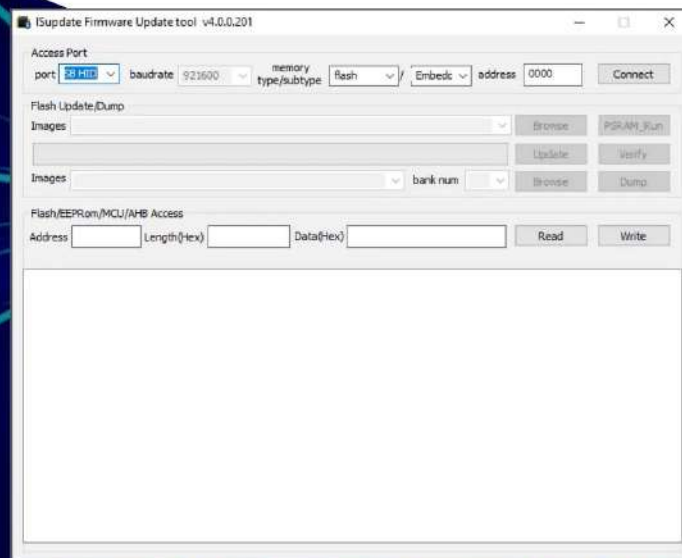
Find Bluetooth Address

Due to technical limitations in the project code, we need to change a hard-coded Bluetooth address in **usart.h** to our RN4870's address. To do this, we will need to first find its address:

1. Plug in the RN4870 (**not** in developer mode).
2. Click **Connect**.

Sometimes in the console you will see a `ÿ` character appear. It doesn't mean anything; but is a useful, but not definitive, indicator that the connection succeeded.

Figure 5: This is the UI for the RN4870 Firmware Update Tool. (Source: Mouser Electronics)>



1. On the keyboard, type `$$$` (the characters will not appear in the console).
2. When a command prompt appears, type `"D"` (capital) then hit enter.
3. The console will populate with the device information. The address should be the 12-digit hexadecimal number beside **BTA**.

When we find the address, we need to write it down in **usart.h** as the **BLE_DEVICE_ADDRESS** definition. This definition has no functional use, but it will help us remember what band has what address. To be able to connect other bands, we need to write that band's address in the **BLE_PARTNER_ADDRESS** definition. We can use the same process to get its address.

Hacking the MIKROE-2543

The MIKROE-2543 does not have the luxury of a micro USB port and will not work with CoolTerm nor the firmware update tool; but if you already have at least one RN-4870-SNSR, you can trick the MIKROE-2543 into working:

1. Remove all jumper connectors in the UART pin grouping (J3) on the RN-4870-SNSR.
2. The pins connected to the micro USB port are on the inside of the UART pin grouping. Holding the module face up, here are the pins from bottom to top: RX, TX, CTS, and RTS:
 - Connect the **SNSR RX pin** to the **2543 RX pin**.
 - Connect the **SNSR TX pin** to the **2543 TX pin**.
 - Connect the **SNSR CTS pin** to the **2543 RTS pin** (if flow control is enabled).
 - Connect the **SNSR RTS pin** to the **2543 CTS pin** (if flow control is enabled).

3. Plug in the RN-4870-SNSR.

The MIKROE-2543 now works like the RN-4870-SNSR.

...a vibration motor alarm would go off, letting them know that they've gone too far.



HARDWARE SETUP

Here is where things get complicated. There are a lot of external parts in this project, so get ready. We'll be using the pins configured in the [Mouser Electronics Github](#) in **Watchdog/atmelstart/atmel_start_config.atstart**. It is best practice to have the board disconnected from power while wiring. It's recommended that we mount the BMI160 and the RN4870 on a breadboard if able.

Power

Power needs to be supplied to the BMI160, the RN4870, and the Vibration motor; everything else can be handled by pins. If you are using the MIKROE-2543, 3.3V and GND are marked on the board.

Vibration Motor

The vibration alarm is the second most important external component (behind the RN4870). The motor cannot be powered by an output pin on the board, as they are current limited, so we will use a transistor to give it more juice.

LEDs

The LEDs are to help the user understand what the band is doing. We are using three green LEDs and one RGB mounted to the breadboard.

BMI160

We need to communicate with the BMI160 to configure motion events and interrupts. We will be using I²C to communicate with the module.

RN4870

The RN4870 uses UART to communicate with external devices. The pins for the RN-4870-SNSR are different than the pins of the MIKROE-2543.

CONCLUSION

The prototype is now complete. Just plug it in, run the code, and you're off. Watchdog solves a rather small problem. Many people would argue that parents should just pay more attention to their children. I can see the reasoning behind that. Most of the time, parents are paying attention; but no mistake happens intentionally; and when it comes to kids, anything that helps avoid mistakes is welcome. 🐶

SAM C21 Xplained Pro Evaluation Kit

- ATSAMC21J18A Microcontroller
- Embedded Debugger with USB interface
- Application examples in Atmel Software Framework

[LEARN MORE](#)



RN4870 & RN4871 BLE Modules

- Fully qualified Bluetooth® smart module
- ASCII Command Interface API over UART
- Scripting engine for hostless operation

[LEARN MORE](#)



Connecting the Microchip AC164160 AVR-IoT WG to Google Cloud IoT

Greg Toth for Mouser Electronics



The Microchip Technology AC164160 AVR-IoT WG Evaluation Board is an Internet of Things (IoT) development kit for prototyping IoT applications that securely connect to the cloud. In this project, we'll set up an IoT development environment and configure the AVR-IoT WG board and Google Cloud to run an application that reads hardware sensors and securely transmits real-time data to the cloud. The AVR-IoT WG board contains a cryptographic coprocessor that stores and protects the board's private key that authenticates itself in the Google Cloud. Once sensor data is sent to the cloud, the data can be stored, processed, and analyzed using a variety of cloud computing resources. As you perform this project, we'll show you how to modify and rebuild the application software to make your own changes. This process will provide a foundation for prototyping other types of secure IoT applications.

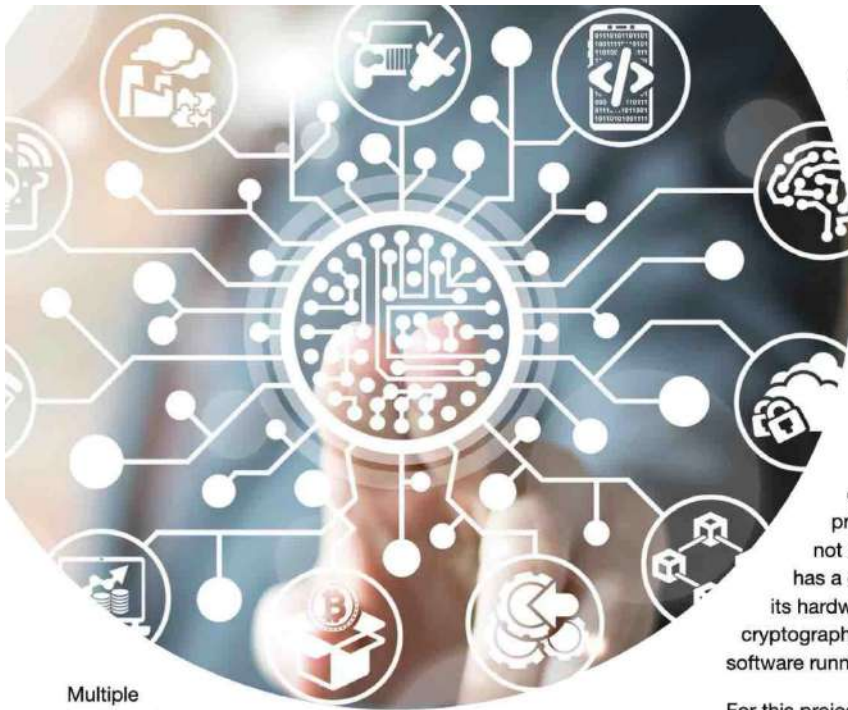
PROJECT TECHNOLOGY OVERVIEW

AC164160 AVR-IoT WG Evaluation Board

The AC164160 AVR-IoT WG (Figure 1) is a development board from Microchip Technology that tests and evaluates the ATmega4808 low-power microcontroller and ATECC608A cryptographic coprocessor along with sensors and a Wi-Fi® communications module. It allows developers to prototype IoT applications that securely connect to cloud computing services through the internet and that support hardware-based security services for authentication, confidentiality, and data integrity.

Figure 1: Microchip Technology AC164160 AVR-IoT WG Evaluation Board. (Source: Mouser Electronics)





Multiple integrated development

environments (IDEs) support the AC164160 AVR-IoT WG board including the Atmel Studio 7 (for Windows®), the MPLAB® X IDE (for Windows, macOS®, and Linux®), and others. The board connects to a PC using a USB cable, which supports programming and debugging from the IDE, drag-and-drop programming, and microcontroller serial port connectivity. The mikroBUS™ expansion connector allows you to expand the board's capabilities with 450+ sensors and actuators offered

by Mikroe through their growing portfolio of Click boards™.

ATECC608A Cryptographic Coprocessor

The **ATECC608A** is a high-security cryptographic device that combines hardware-based key storage with hardware cryptographic accelerators to implement various authentication and encryption protocols. It has an electrically erasable programmable read-only memory (EEPROM) array that can store up to 16 keys, certificates, or other sensitive data with configurable access controls and locking functions to prevent changes. It can generate and store random private keys internally to ensure a private key is not discoverable outside of the device. Each device has a guaranteed unique 72-bit serial number, and its hardware cryptographic accelerators can implement cryptographic operations' orders of magnitude faster than software running on a standard microcontroller.

For this project, we'll use the private key stored in the ATECC608A to authenticate the AVR-IoT board in the Google Cloud IoT (this process eliminates all need to handle or store a private key among the board's software or configuration data). Each AVR-IoT board will have a unique ATECC608A with its own set of public and private keys.

Atmel Studio 7

Firmware programs that run on microcontrollers are typically developed and tested using an IDE—such as Atmel Studio 7—running on a PC. The IDE provides an editor, compiler, linker,

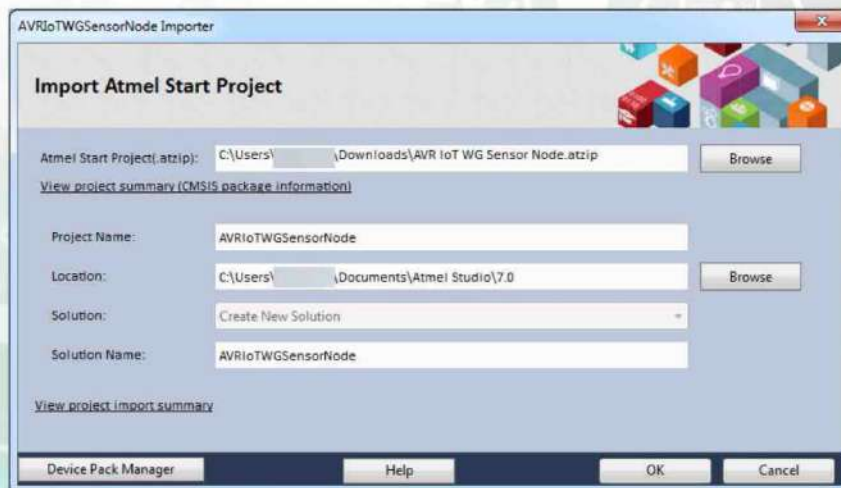


Figure 2: Importing the Atmel START Project into Atmel Studio. (Source: Mouser Electronics)

debugger, and a mechanism for transferring binary program images to a microcontroller.

The AVR-IoT WG board is programmable using several different IDEs, including two that are available from Microchip Technology. For this project, we'll use the free Atmel Studio 7 IDE that runs on Windows. It supports the full range of Microchip Technology AVR® and SAM microcontrollers and is available through the Microchip Technology [Atmel Studio 7 web page](#).

Atmel Studio 7 supports the Nano Embedded Debugger (nEDBG) interface built into the AVR-IoT WG board. The debugging interface allows compiled programs to be downloaded and executed with debugging support for breakpoints and memory inspection.

Atmel START

Atmel START is a free tool available by Microchip Technology on the [Atmel START web page](#). It consists of example projects and libraries you can browse and select to generate your own source code projects. The purpose of this tool is to quickly create new projects containing necessary functions and libraries to connect the AVR-IoT WG board to the Google Cloud IoT through the internet. The libraries interact with the ATECC608A cryptographic coprocessor, the Wi-Fi module, and the onboard sensors and LEDs. They also implement communication and security protocols including Transport Layer Security (TLS), Message Queue Telemetry Transport (MQTT), and JavaScript Object Notation [JSON] Web Tokens (JWT), which Google Cloud employs.

Google Cloud Platform and Google IoT Core

Google Cloud Platform is a collection of computing services and tools that run in Google data centers located throughout the world. The platform provides a large set of services that can be useful building-blocks to create a variety of applications using a pay-as-you-go billing model. Cloud services include computing, storage, database management, networking, monitoring, containerization, mapping, data analytics, pub/sub, machine learning, and IoT. We'll use Google's Cloud IoT Core service to integrate our board with the Google Cloud.

Wi-Fi Access Point

The AVR-IoT WG board contains a built-in Wi-Fi module that connects to the internet through a Wi-Fi access point supporting 802.11 b/g/n standards. The access point must supply a DHCP Internet Protocol (IP) address to the Wi-Fi module and allow outgoing internet traffic without blocks by a proxy or firewall.

gcloud

The [gcloud](#) command-line interface is a tool for interacting with Google Cloud from the command line of a PC. The tool is available as part of the [Google Cloud SDK](#). We'll use gcloud to display real-time data the Google Cloud IoT receives.

THE SETUP (SOFTWARE)

Download the Project

Click **AVR IoT WG Sensor Node**, and then click **Download Selected Example**. The download file is named **AVR IoT WG Sensor Node.atzip**, and the .atzip extension means it's an Atmel Studio project file. Double-click the downloaded AVR IoT WG Sensor **Node.atzip** file to open it in Atmel Studio. You'll see a project import window (**Figure 2**) with default names and directory locations filled in. Click **OK** to complete the import.

If you get a warning message about necessary device updates, it may be because there are Device Pack updates you haven't installed yet.

After the project is imported, it will appear in the Solution Explorer pane (**Figure 3**).

Verify Building the Project Code

Try building the project "as is" by using the **Build → Build Solution** menu command. The project should build without any errors. Note that the current code does generate a few build warnings.

We'll come back to the source code later, after we've set up our Google Cloud IoT environment.

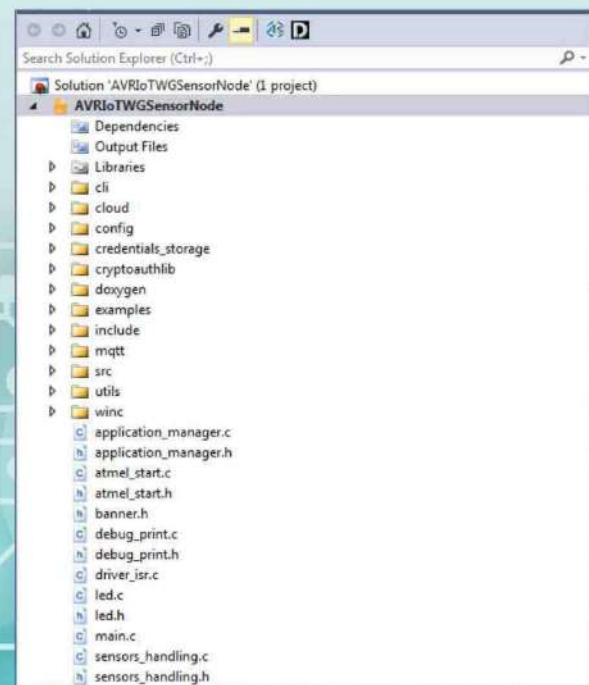


Figure 3: Solution Explorer appears after importing the Atmel START Project. (Source: Mouser Electronics)

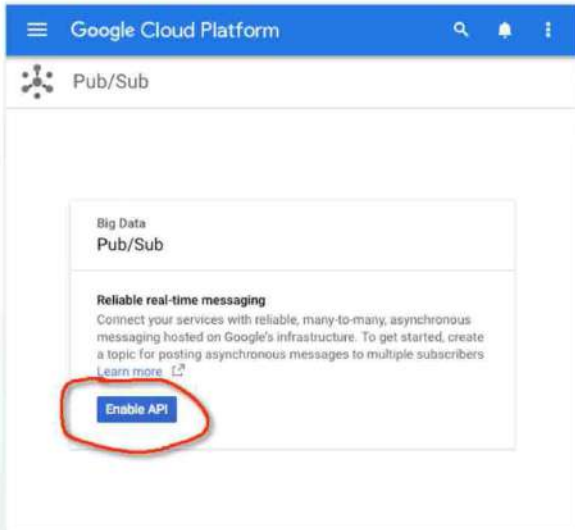


Figure 4: Enabling the Pub/Sub API. (Source: Mouser Electronics)

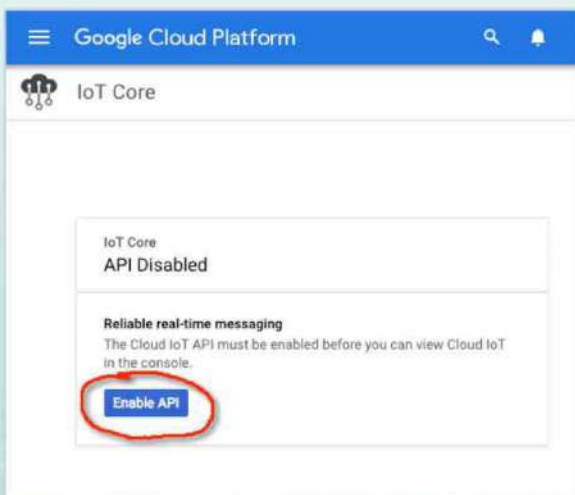


Figure 5: Enabling the IoT Core API. (Source: Mouser Electronics)

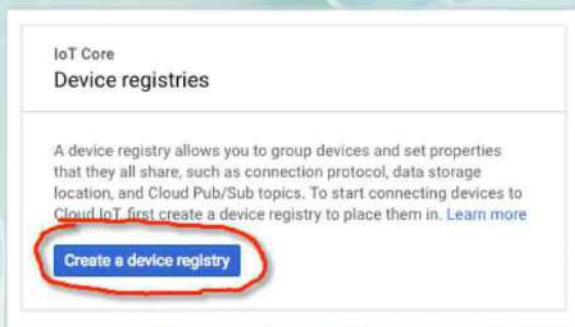


Figure 6: Creating a Device Registry. (Source: Mouser Electronics)

Google Cloud Platform

This step makes the board known to Google Cloud IoT and adds the digital key corresponding to the board's unique ATECC608A cryptographic coprocessor. Use your web browser to navigate to the Google Cloud console at <https://console.cloud.google.com>. You'll need to establish a Google Cloud login account and create a new cloud project with billing support enabled at the project level. The rest of the steps below are performed within the newly created cloud project.

Enable Pub/Sub API

Navigate to the **Pub/Sub** service and enable the Pub/Sub **API** (Figure 4).

Enable IoT Core API

Navigate to the IoT Core page and enable the IoT Core **API** (Figure 5).

Create Device Registry

While still in the IoT Core service, click **Create a device registry** (Figure 6).

You'll enter multiple fields using these settings:

Registry ID: my-iot-registry

Region: us-central1

Protocol: MQTT (checked) and HTTP (checked)

Under "Default telemetry topic," click **Select a Cloud Pub/Sub topic**, and select **Create a topic**. Type the word "events" (lowercase and without quotes) at the end of the topic string and click **Create**. This is the Google Cloud Pub/Sub topic to which incoming MQTT messages will be mapped to.

Under "Default state topic," click **Select a Cloud Pub/Sub topic**, and select **Create a topic**. Type the word "state" (lowercase and without quotes) at the end of the topic string and click **Create**.

Finally, apply the following setting:

Stackdriver Logging: Leave as None

Once you've applied all the settings, click **Create**. The settings should look like **Figure 7**, except you'll have a different project name:

Obtain the Private Key from the Board's Cryptographic Coprocessor

On the PC that's connected to the running AVR-IoT WG board, open a serial terminal program (such as Tera Term) and set the parameters to 9600bauds, 8-data bits, 1-stop bit, and no parity. Set line endings to CR+LF and turn on echo. Hit the **Enter** key

Figure 7: IoT Core Registry Settings. (Source: Mouser Electronics)

a couple times and you should see the board's command-line interface response (**Figure 8**). The exact list of commands may vary slightly, depending on the version of firmware installed on the board during manufacturing.

Type **"device"** (without the quotes) and press **Enter**. The screen will display the unique 18-digit device ID for this board. In the next step, we'll add the letter 'd' to the beginning of the 18-digit ID and use the resulting string as the board's device ID in Google Cloud IoT. For example, if the 18-digit ID is "1234567890ABCDEF12," the device ID in Google Cloud IoT will be "d1234567890ABCDEF12."

Type **"key"** (without the quotes) and press **Enter**. The screen will display the unique public key for the board, corresponding to the board's unique ATECC608A chip. The public key consists of four text lines that begin with -----BEGIN PUBLIC KEY----- and end with -----END PUBLIC KEY----- . We'll use this public key in the next step.

Create the Device and Add the Public Key

In the Google Cloud IoT Core navigation menu, select **Devices**, and click **Create a Device**. Enter these settings:

Device ID: Letter "d" followed by the 18-digit device ID

Device communication: Allow

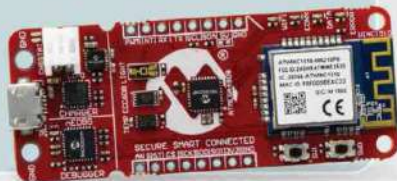
```
Unknown command. List of available commands:
reset
device
key
reconnect
version
cli_version
wifi <ssid>[, <pass>[, <authType>]]
debug
```

Figure 8: AVR-IoT WG Board Command-Line Interface in the Serial Terminal. (Source: Mouser Electronics)

AVR-IoT WG Evaluation Board (AC164160)

- 4x status LEDs (Wi-Fi, CONN, DATA, ERROR)
- USB power, debugging, and UART communication
- Drag-and-drop programmer

[LEARN MORE](#)



Microchip Technology ATECC608A CryptoAuthentication™ Devices

- Cryptographic Co-Processor with Secure Hardware-based Key Storage
- Hardware Support for Symmetric Algorithms
- Guaranteed Unique 72-bit Serial Number

[LEARN MORE](#)



Authentication Input method: Enter manually

Public key format: ES256

Public key value: All four lines of the public key printed on the serial terminal

Public key expiration date: Leave unchecked

Device metadata: Leave empty

Stackdriver Logging: Use registry default setting

The settings should look like **Figure 9**. Once you've entered these settings, click **Create**.

At this point, we've defined the board device in the Google Cloud IoT, and the board is ready to connect.

Figure 9: IoT Core Device Settings. (Source: Mouser Electronics)

Configure the Google Cloud IoT Core Parameters in the Board Software

In this step, we'll adjust several parameters in the source code to use our Google Cloud IoT settings rather than the default values. In Atmel Studio, locate the following files and edit them as follows:

config/iot_sensor_node_config.h

- Change the CFG_PROJECT_ID string to your own Google Cloud project name.
- Change the CFG_REGISTRY_ID string to "my-iot-registry".
- Change CFG_DEBUG_MSG to 1 to enable debug messages.

config/conf_winc.h

- Change the CFG_MAIN_WLAN_SSID string to your own Wi-Fi access point SSID.
- Change CFG_MAIN_WLAN_AUTH to use the correct constant corresponding to your Wi-Fi access point security type (for example, use M2M_WIFI_SEC_WPA_PSK for WPA/WPA2); other values are documented in the source code.

The device ID is generated dynamically by the code at runtime (based on the cryptographic coprocessor's unique ID), so there isn't anything to configure for this device ID.

Next, build the project using the **Build → Build Solution** menu command. Download and run the program on the AVR-IOT board using the **Debug → Start Without Debugging** menu command.

The board should show a brief flash of the LEDs at startup, then the LEDs should illuminate Blue (for Wi-Fi connected), then Green (for Google Cloud IoT connected), then Yellow flashing once a second (for data transmissions to Google Cloud). A Red LED indicates a problem connecting to Wi-Fi or Google Cloud.

PUBLISHING SENSOR DATA MESSAGES TO THE GOOGLE CLOUD IOT

At this point, the board is running the application and publishing sensor messages once a second. Atmel START's example application is coded to format each set of sensor readings as a string in the JSON format, with separate fields for light and temperature: **{"Light":26,"Temp":27.43}**

Monitoring Messages

You can monitor the published sensor messages by creating a subscription using the gcloud command-line tool on your PC then running a short Python program that uses the subscription to receive and print messages.

Use gcloud to create a monitoring subscription (substitute your own project name in place of *lithe-lens-228601*):

```
gcloud --project lithe-lens-228601 pubsub \  
subscriptions create --topic events monitor-sub
```

Install the Google Cloud Pub/Sub Python client package on your PC: **pip install google-cloud-pubsub**

Create a new file named **monitor-sub.py** containing the following Python code; replace *lithe-lens-228601* with your own project name:

```
import time
```

```
from google.cloud import pubsub_v1
```

```
project_id = "lithe-lens-228601"  
subscription_name = "monitor-sub"
```

```
subscriber = pubsub_v1.SubscriberClient()  
# The `subscription_path` method creates a fully qualified  
identifier  
# in the form `projects/{project_id}/subscriptions/  
{subscription_name}`  
subscription_path = subscriber.subscription_path (project_  
id, subscription_name)
```

```
def callback (message):  
    print('Received message: {}'.format(message.data))  
    message.ack()
```

```
subscriber.subscribe(subscription_path, callback=callback)
```

```
# The subscriber is non-blocking. We must keep the main  
thread from  
# exiting to allow it to process messages  
asynchronously in the background.  
print('Listening for messages on {}'.  
format(subscription_path))  
while True:  
    time.sleep(60)
```

Run the Python program using this command:
python monitor-sub.py

As new messages are published to the Google Cloud, they will be printed on your PC like this:


```
Received message:  
{"Light":26,"Temp":27.43}"
```

Press Control-C to terminate the monitor program.

Remember to power off the board when you're done so it doesn't continue to send messages to the Google Cloud and consume cloud resources.

WHERE TO GO NEXT

This project created an end-to-end sensor-to-cloud application that you can modify and extend in various ways. Here are a few examples:

- Dive deeper into the application code by reading Atmel START's example-project user guide and studying its source code.
- Learn more about the ATECC608A cryptographic coprocessor by reading the datasheet and studying its source code.
- Change the Wi-Fi access point and/or Google Cloud connection parameters by modifying the source code.
- Modify the sensor publishing-time interval by modifying the source code.
- Include a timestamp in the sensor data message.
- Use a different Atmel START example project that uses sensors on a Click board connected to the mikroBUS connector.
- Connect the pub/sub topic to a Google Cloud BigQuery table to record sensor data for further analysis.
- Develop a data display application that subscribes to the sensor data topic and displays received data values. 





Helping Connect to the Cloud

Connecting Microchip Technology's AC164164 PIC-IoT to Medium One IoT Cloud

Greg Toth for Mouser Electronics

The [Microchip Technology PIC-IoT WG](#) is an Internet of Things (IoT) development board featuring a PIC® microcontroller, a hardware secure element device, Wi-Fi® connectivity, onboard sensors, and a mikroBUS® connector for adding a variety of expansion boards. The [Medium One](#) IoT Prototyping Sandbox is a cloud-based IoT platform designed to help early stage developers prototype IoT projects or connect existing hardware to the cloud. In this project, we'll set up an IoT development environment using the PIC-IoT WG to read temperature and light sensors and send the data to the Medium One cloud using MQTT. Once the data is in Medium One, it can be processed and viewed using programmable workflows and configurable widget dashboards.

PROJECT TECHNOLOGY OVERVIEW

Microchip Technology PIC-IoT WG Development Board

The PIC-IoT WG (**Figure 1**) is an IoT development board based on the [PIC24J family](#) of extreme low-power microcontrollers. It offers built-in Wi-Fi connectivity for creating IoT applications that communicate wirelessly with other systems and devices. The Wi-Fi module offloads networking tasks from the main CPU to maximize processor resources available for your applications. The board can be powered through USB or a rechargeable battery using the built-in battery charging circuit. The combination of board, microcontroller, and onboard components offer developers a number of features for prototyping IoT designs:

Technical documentation for the PIC-IoT can be found [here](#) and [here](#).

The board is supported by the MPLAB® X Integrated Development Environment (IDE) and connects to a PC using a

USB cable that supports programming and debugging from the IDE as well as microcontroller serial port connectivity.

The mikroBUS expansion connector allows you to expand the board capabilities with 450+ sensors and actuators offered by [Mikroe](#) through their growing portfolio of Click boards™.

Microchip Technology MPLAB® X Integrated Development Environment (IDE)

Firmware programs that run on microcontrollers are typically developed and tested using an integrated development environment (IDE) running on a personal computer. The IDE provides an editor, compiler, linker, debugger, and a mechanism for transferring binary program images to the microcontroller.

The PIC-IoT WG can be programmed using the Microchip Technology MPLAB X IDE, a NetBeans-based IDE that runs on Windows, Mac®, and Linux® computers. It connects to the PIC-IoT WG using a USB cable that supports programming and debugging. MPLAB X can be downloaded for free from the Microchip Technology MPLAB X site.

Microchip Technology XC16 Compiler

Application source code is converted into machine code for the PIC24 microcontroller by the Microchip Technology XC16 compiler, which integrates into the MPLAB X IDE environment.

Microchip Technology MPLAB® Code Configurator (MCC)

MPLAB Code Configurator is a free, graphical programming environment that generates seamless, easy-to-understand C code to be inserted into your project. It integrates into the MPLAB X IDE environment to accelerate the generation of production ready code.

Project Application Source Code Files

For this project, MPLAB X was used to create an initial set of project source code files that have been modified and extended to work with the Medium One IoT Prototyping Sandbox. The resulting files have been put in a GitHub repository that you can download and use for this project. The project files will be opened in the MPLAB X IDE where they'll be compiled and downloaded to the PIC-IoT board. The project files consist of a main application program and supporting functions for MQTT and I/O processing.

Medium One IoT Prototyping Sandbox

The Medium One IoT Prototyping Sandbox (**Figure 2**) is designed to help early stage developers prototype their IoT project or connect their existing hardware to the cloud. It offers an IoT Data Intelligence platform enabling customers to quickly build IoT applications with less effort. Programmable workflows allow you to build processing logic without having to create your own complex software stack. Configurable dashboards allow you to visualize application data and view real-time data in a variety of formats. Medium One's iOS and Android™ apps allow you to build simple mobile app dashboards that can communicate with your devices through the IoT Prototyping Sandbox.

IoT devices can exchange data with Medium One through either a REST API or MQTT. More detailed information about the Medium One IoT Prototyping Sandbox can be found [here](#) and on the [Medium One](#) site.



Figure 1:
PIC-IoT WG
Development Board
(Source: Mouser Electronics)

THE SETUP (SOFTWARE)

Download and Install MPLAB X, XC16 Compiler and MPLAB Code Configurator.

Web browse to the Microchip Technology MPLAB X site and locate the MPLAB X installer for your type of PC. Run the installer and make sure the following applications are selected for installation:

- MPLAB X IDE
- Device support for 16-bit MCUs

The other items are not needed for this project and can be omitted to save disk space.

At the end of the installation, two web pages will open automatically. One takes you to a page for downloading the MPLAB XC16 Compiler. Download the XC16 installer for your operating system and run it to install the compiler on your machine. You can select the free MPLAB XC16 C Compiler by clicking **Next** in the Licensing Information dialog.

The second web page is for MPLAB Code Configurator. You can either download and install it now from the web page, or you can install it after launching MPLAB X (next step) and install from the **Tools → Plugins → Available Plugins** menu item.

Plug the PIC-IoT board into your PC using the USB cable, then launch MPLAB X. If you didn't already install MPLAB Code Configurator from the web page, install it now from **Tools → Plugins → Available Plugins → MPLAB Code Configurator → Install** and let MPLAB X restart. You can verify whether MPLAB Code Configurator is installed by navigating to **Tools → Plugins → Installed** and looking for MPLAB Code Configurator in the list of installed plugins.

When MPLAB X launches with the PIC-IoT board connected and MPLAB Code Configurator installed, you should see a main screen with a

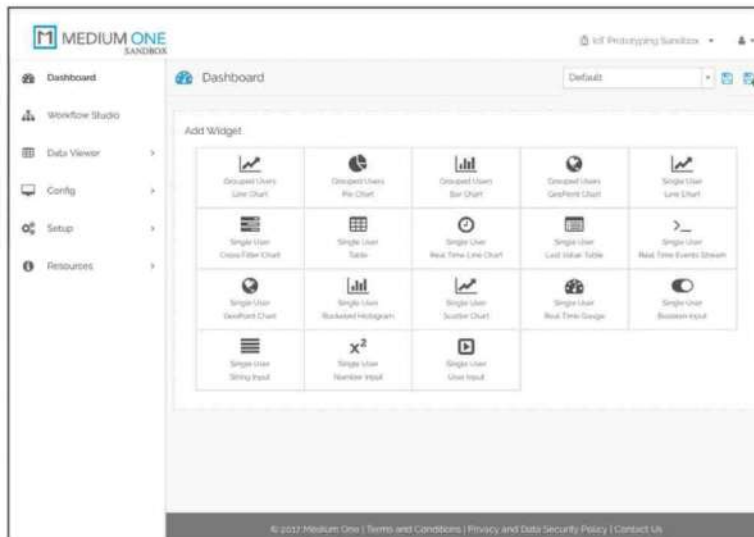


Figure 2: Medium One IoT Prototyping Sandbox. (Source: Mouser Electronics)

Kit Window tab that looks like **Figure 3**. If it doesn't look like this, go back through the PIC-IoT user guide and MPLAB X installation documents to make sure all steps were followed. Also make sure your PIC-IoT board is connected when you start MPLAB X.

Download and Open the Project Application Source Code Files

Web browse to the [GitHub](#) repository and find the **PICIoT_MediumOne_1.0.0.zip** file. Download that file to your computer and unzip it to create a directory named **PICIoT_MediumOne.X**. Move that entire directory and place it in the **MPLABXProjects** folder on your PC.

In MPLAB X, select **File → Open Project...** then select **PICIoT_MediumOne.X** from the list and click **Open Project**. Afterwards the Projects tab should look like **Figure 4**.

We'll come back to the source code files later after setting up Medium One.

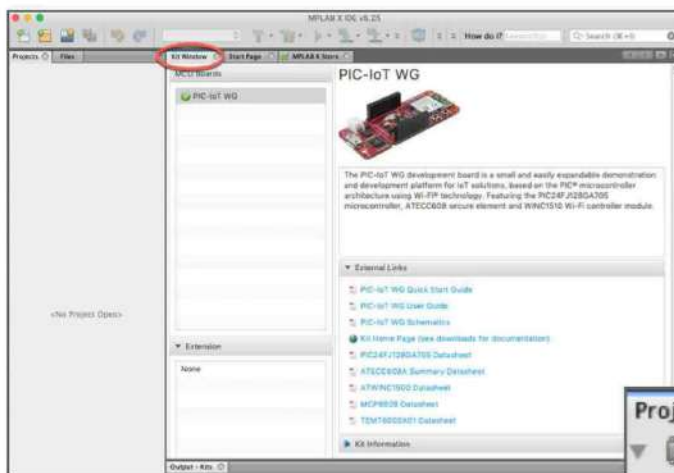


Figure 3: MPLAB X IDE With PIC-IoT Board Detected.
(Source: Mouser Electronics)

Set Up the Medium One IoT Prototyping Sandbox

Web browse to the [Medium One IoT Prototyping Sandbox](#) and log in. Then, you should see an initial dashboard resembling **Figure 2**. Click **Setup → Manage Users → Add New User**. Set **Username** to **mydevice**, create a password of your choosing and enter it in both password fields, then click **Save**. In the Manage API Users list, you should see a new user account having **Login ID** = **mydevice** and an auto-generated user **MQTT ID** like **Figure 5**.

Click **Setup → MQTT** and you should see a **Project MQTT ID** and a set of port numbers like **Figure 6**.

Medium One uses MQTT usernames and passwords for authentication. The MQTT username is created by combining the Project MQTT ID, a forward slash, and the user MQTT ID. For example, if the Project MQTT ID is "ZHxxxxxx0Y" and the user MQTT ID is "sTxxxxxx8w" the corresponding MQTT username would be "ZHxxxxxx0Y/sTxxxxxx8w".

Next, we'll create the MQTT password. Navigate to **Setup → Manage API Keys → Add New API Key**. Set the description to **mydevice**, make sure **Enabled** is check-marked, and click **Save**. The result should look like **Figure 7**.

The MQTT password is created by combining the API Key, a forward slash, and the mydevice user password. For example, if the API Key is "PZxxMBQ" and the mydevice user password is "AaaaBbbb3" the corresponding MQTT password would be "PZxxMBQ/AaaaBbbb3".

The MQTT topic has the following format: "0/Project MQTT ID/User MQTT ID/Device ID".

The Device ID field can be anything and we'll use the PIC-IoT's ATECC608A unique ID (18 hex digits obtained at runtime) as the Device ID. For example, if the Project MQTT ID is "ZHxxxxxx0Y" and the user MQTT ID is "sTxxxxxx8w" the corresponding MQTT topic would be "0/ZHxxxxxx0Y/sTxxxxxx8w/NNNNNNNNNNNNNNNNNN" where NNN...NNN is the ATECC608A unique ID obtained at runtime.

The MQTT username, MQTT password, and MQTT topic strings will get added to the project source code in the next step.

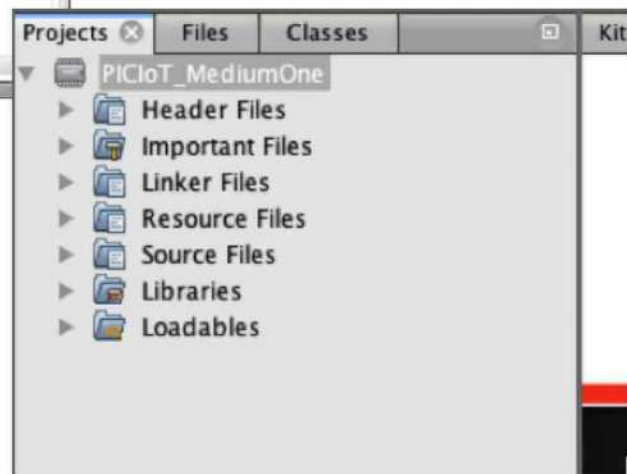


Figure 4: Projects tab after opening the PICIoT_MediumOne Project in MPLAB X. (Source: Mouser Electronics)

Update Application Source Code Files for Medium One Account Parameters

Set Wi-Fi Connection Parameters

In MPLAB X open project file **Header Files/MCC Generated Files/config/conf_winc.h** in the editor. Find these constants and set them to your own Wi-Fi SSID and password info:

- CFG_MAIN_WLAN_SSID
- CFG_MAIN_WLAN_AUTH
- CFG_MAIN_WLAN_PSK

Set Medium One Connection Parameters

Open project file **Header Files/MCC Generated Files/config/IoT_Sensor_Node_config.h** in the editor. Find these constants and set them to your own Medium One MQTT parameter strings as described earlier:

- CFG_MQTT_USERNAME
- CFG_MQTT_PASSWORD
- CFG_MQTT_TOPIC – everything except the final Device ID field, which will be added at runtime

NOTE: The current version of the PIC-IoT code is not able to connect to the Medium One MQTT broker over TLS and this project uses unencrypted MQTT communications between the PIC-IoT and Medium One MQTT broker. The MQTT username and password are sent over the internet in the clear and neither the credentials nor the sensor data are encrypted.

Save the modified file and then build the project. Verify the code compiles without errors. If you see compilation errors, check the changes you made to the **conf_winc.h** and **IoT_Sensor_Node_config.h** files.

Run the Application

Make sure the PIC-IoT board is connected to the PC through USB. Build the program and then download to the PIC-IoT board. The program should be downloaded to the board and

start running. You can connect a serial terminal program such as Tera Term to the USB serial comm port if necessary to monitor debug messages from the PIC-IoT while it's running. Set the serial parameters to 9600,N,8,1.

How the Application Program Works

This program is derived from the PIC-IoT WG Sensor Node example application with the following changes and enhancements:

- Connect to the Medium One MQTT broker instead of Google Cloud.
- Add MQTT username.
- Use the ATECC608A unique ID as the Device ID in the MQTT topic name.
- Use unencrypted MQTT connection.
- Add iteration count data element.
- Set data measurement interval to 5 seconds.
- Generate a JSON formatted MQTT payload message and send to Medium One MQTT broker.

MQTT Payload Format

MQTT messages are formatted as JSON strings according to the Medium One MQTT payload specification. Here's an example message:

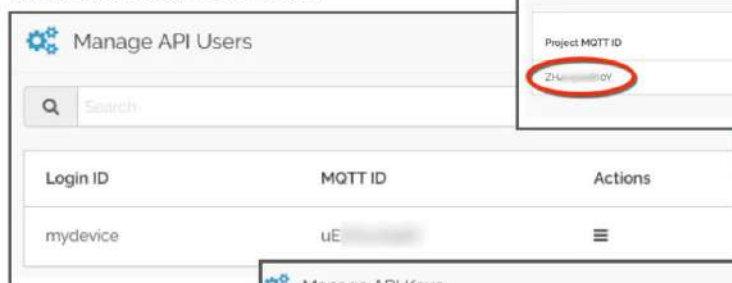
```
{"event_data":{"light":40,"tempc":31.87,"iteration":32}}
```

Fields:

- light = light sensor value
- tempc = temperature in degrees Celsius
- iteration = application loop counter

Try heating or cooling the temperature sensor and covering the light sensor to see the data values change.

Figure 5: Newly created user ID with auto-generated MQTT ID. (Source: Mouser Electronics)

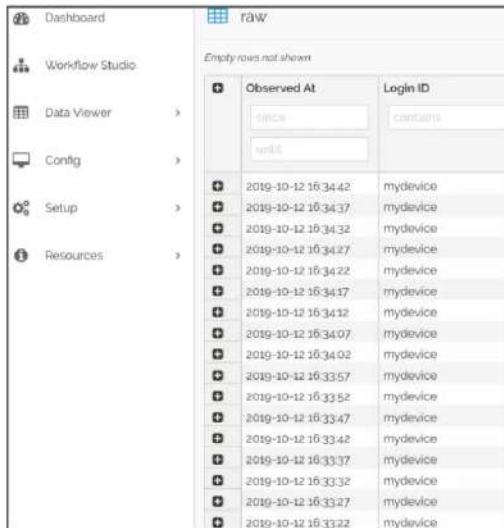


Project MQTT ID	TCP Port	Secure TCP Port	Websocket Port	Secure Websocket Port
2f4e6688000f	61629	61620	61629	61630

Figure 6: Project MQTT ID. (Source: Mouser Electronics)

API Key	Description	Enabled	Actions
C2f...	mydevice	<input checked="" type="checkbox"/>	⋮

Figure 7: Newly Created API Key. (Source: Mouser Electronics)



Observed At	Login ID
2019-10-12 16:34:42	mydevice
2019-10-12 16:34:37	mydevice
2019-10-12 16:34:32	mydevice
2019-10-12 16:34:27	mydevice
2019-10-12 16:34:22	mydevice
2019-10-12 16:34:17	mydevice
2019-10-12 16:34:12	mydevice
2019-10-12 16:34:07	mydevice
2019-10-12 16:34:02	mydevice
2019-10-12 16:33:57	mydevice
2019-10-12 16:33:52	mydevice
2019-10-12 16:33:47	mydevice
2019-10-12 16:33:42	mydevice
2019-10-12 16:33:37	mydevice
2019-10-12 16:33:32	mydevice
2019-10-12 16:33:27	mydevice
2019-10-12 16:33:22	mydevice

Figure 8: Raw Message Display. (Source: Mouser Electronics)

View Data in the Medium One Dashboard

In the Medium One dashboard navigate to **Data Viewer** → **Data Streams** and click raw **Events**. You should see raw messages (**Figure 8**) being received from the PIC-IoT. Click the "+" sign to view message details.

Click **Dashboard** on the top left, then click **Add Widget** → **Single User Real Time Events Stream** to add an event stream widget to the dashboard.

In the Select user dropdown, select **mydevice**. You should now see messages appearing in the Real Time Events Stream widget (**Figure 9**). Click the **save** icon in the upper right corner to save your modified dashboard.

Add More Widgets

To display more widgets, we need to enable specific data fields contained in the message payload. Navigate to **Config** → **Data Streams** and click on **raw Events**. The Schema Map should be pre-populated with fields detected in the incoming messages; however, they are currently disabled. Check-mark the **Active** box on **raw.iteration**, **raw.light**, and **raw.tempc**, then click **Save Data Stream**. These fields are now available for use in other dashboard widgets.

Back on the dashboard, click the **Single User Last Value Table** widget and select the mydevice user within the widget. Click the widget's **Tag Config** icon to the right of the mydevice user selection and check-mark **raw:iteration**, **raw:light**, and **raw:tempc**, then click **Save**. The Last Value Table should now populate with the most recent received values for each field (**Figure 10**). Click the **save** icon towards the upper right corner to save the updated dashboard.

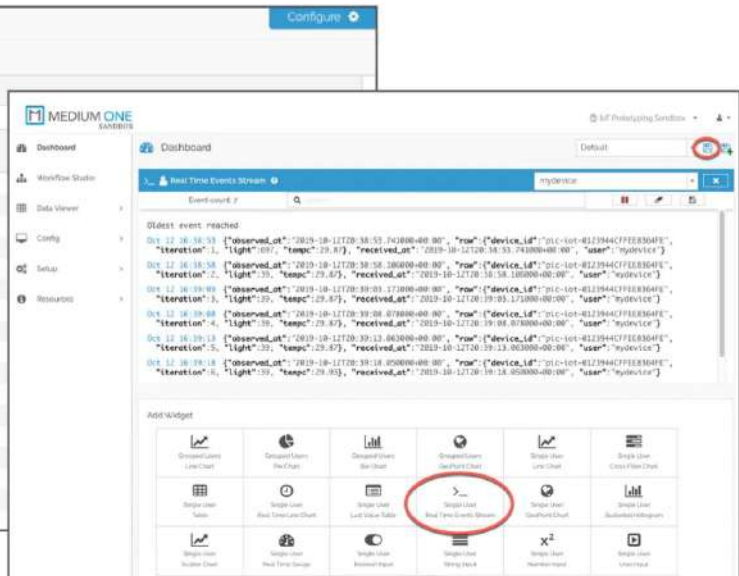


Figure 9: Real Time Events Stream Widget Display. (Source: Mouser Electronics)

Now let's add dashboard widgets for the temperature and light sensors and iteration counter. Click **Single User Real Time Gauge** and select the **mydevice** user. Click the widget's **Tag Config** icon and check-mark the **raw:iteration**, **raw:tempc**, and **raw:light** rows, then click **Save**. The updated dashboard should look like **Figure 11**. Click the dashboard **save** icon to save the updated dashboard. Try heating or cooling the temperature sensor and covering the light sensor to see the gauge values change.

At this point, your PIC-IoT board is running continuously, periodically reading the temperature and light sensors and transmitting data measurements to the Medium One cloud. Remember to power off the PIC-IoT board when you're done, otherwise the board will continue to send messages to Medium One and consume daily message allotments.

WHERE TO GO NEXT

This project created an end-to-end sensor-to-cloud application that sends real-time sensor data to the Medium One IoT Prototyping Sandbox. It can be modified and extended in a number of ways, and here are a few examples:

- Dive deeper into the application code and board hardware by reading the PIC-IoT documentation and studying the source code.
- Add more widgets to the Medium One dashboard, such as a real-time line chart of temperature and light readings.
- Learn about the Medium One Workflow Studio, which lets you create data processing workflows to transform your sensor data.

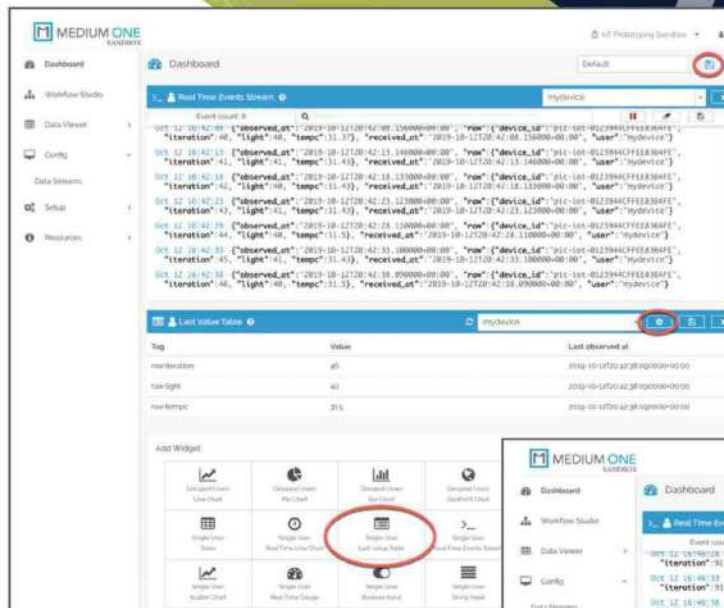


Figure 10: Last Value Table Widget Display.
(Source: Mouser Electronics)

- Experiment with the Medium One mobile apps.
- Implement bi-directional communications with the Medium One cloud.
- Modify the sensor data publishing interval by modifying the PIC-IoT application source code.
- Connect other types of sensors to the PIC-IoT board and include the data in MQTT messages.
- Enhance the MQTT communications to support MQTT over TLS.

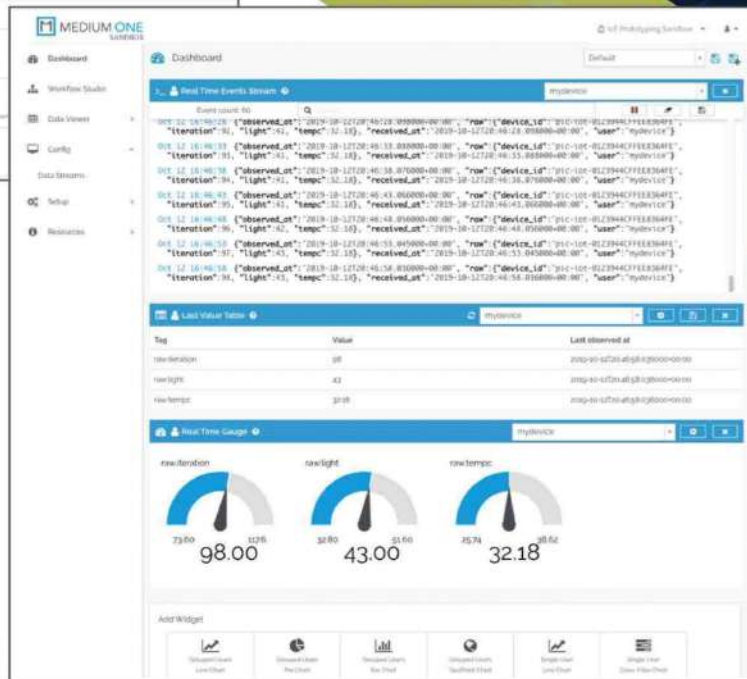


Figure 11: Real Time Gauge Widgets Added to Dashboard. (Source: Mouser Electronics)

Microchip Technology PIC-IoT WG Development Board (AC164164)

- PIC24FJ256 16-bit Microcontroller
- ATECC608A CryptoAuthentication™
- MIC33050 Voltage Regulator

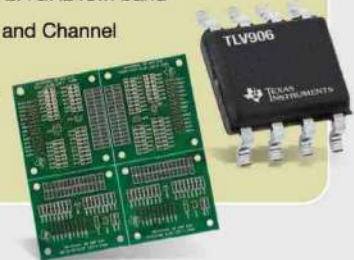
[LEARN MORE](#)



Microchip Technology ATWINC15x0 SmartConnect IoT Modules

- Single spatial stream in 2.4GHz ISM band
- Advanced Equalization and Channel Estimation
- Hardware accelerators for OTA security

[LEARN MORE](#)





Mouser stocks the
widest selection
of the newest products



[mouser.com/microchip](https://www.mouser.com/microchip)

Worldwide leading authorized distributor of
semiconductors and electronic components.



The Newest Products for Your Newest Designs®

Mouser and Mouser Electronics are registered trademarks of Mouser Electronics, Inc. Other products, logos, and company names mentioned herein, may be trademarks of their respective owners.