Introduction

Cosmic Ray Data Acquisition with Arduino-Based Systems

QUEENSBOROUGH COMMUNITY COLLEGE

What Are Cosmic Rays?

Cosmic rays are high-energy particles from outer space that travel through the universe and strike the Earth's atmosphere. Despite the name, they are not rays (like light rays), but rather subatomic particles — mostly protons.

Sources of Cosmic Rays

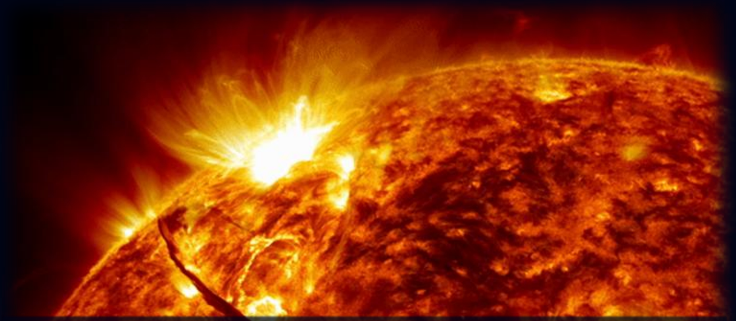


Figure 1.01 – Solar Flares

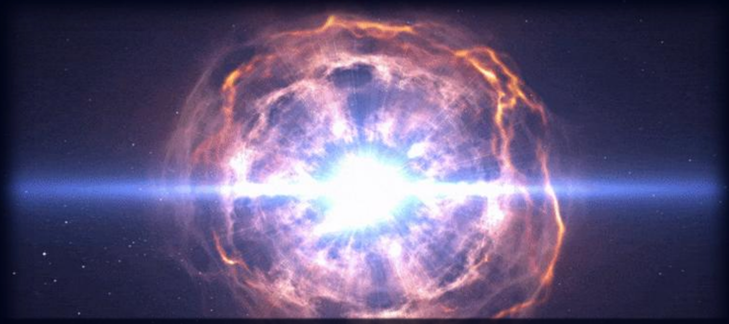


Figure 1.02 – Supernovae

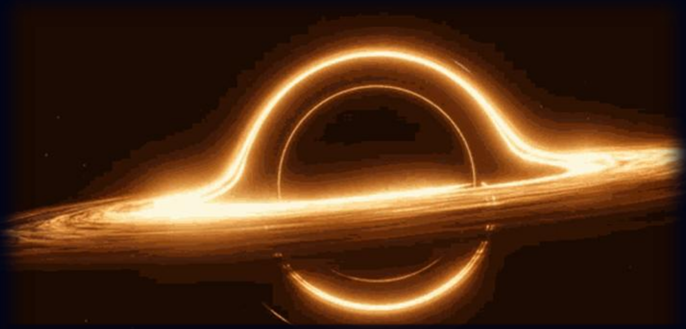


Figure 1.03 – Black Holes

Figure 1.04 – Quasars

Figure 1.05 – Pulsars

Cosmic Ray Showers

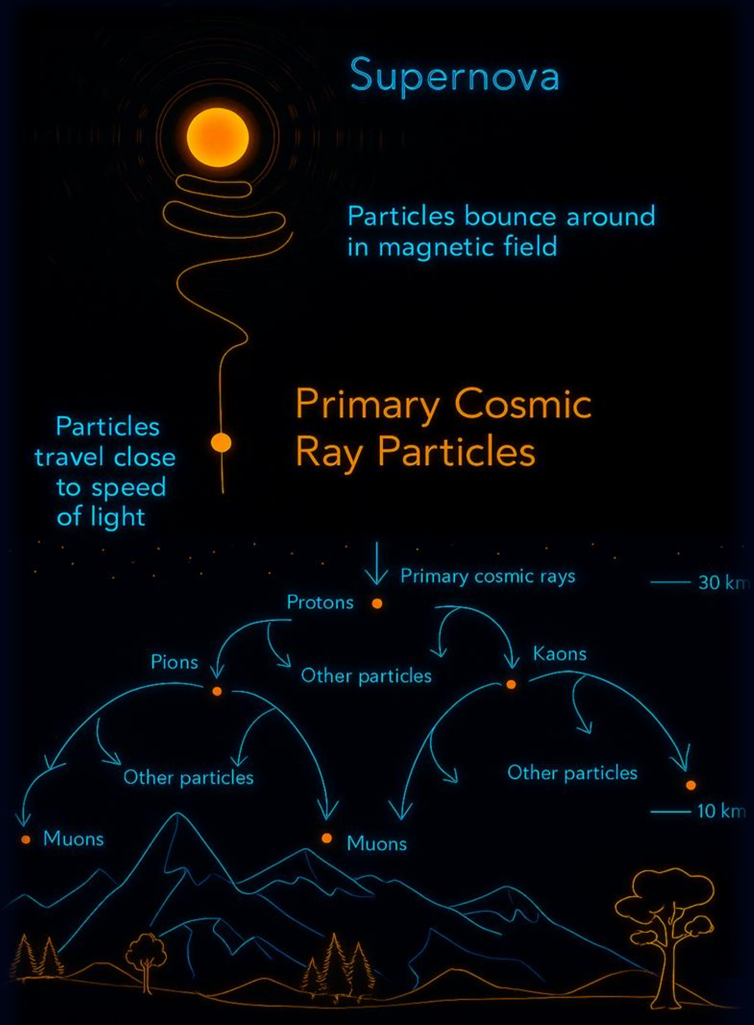


Figure 1.06 – Cosmic Ray Shower

Atmospheric Interactions

When cosmic ray protons collide with molecules in Earth's atmosphere, they initiate extensive air showers that generate a cascade of secondary particles. Many of these secondary particles rapidly decay into muons and neutrinos.

Because muons travel quickly and interact only weakly with matter, they can penetrate all the way to the ground, where they are the primary particles detected by our cosmic ray detectors.

In general, the higher the energy of the incoming cosmic ray, the larger the air shower and the greater the number of secondary particles it produces.

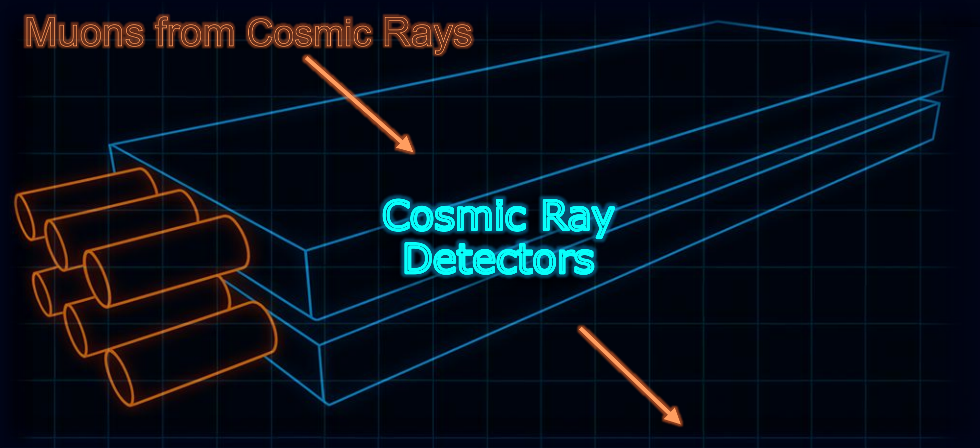


Figure 1.07 – Cosmic Ray Detector

Muon-Induced Photons

Photon emission process

When a muon passes through the scintillation material in the detector, it interacts with the atoms in the material, transferring energy to them.

The scintillator material then releases this excess energy in the form of photons (light).

The amount of light emitted is proportional to the energy deposited by the muon as it passes through the material.

The light guides capture this light and internally reflects it with minimal loss, funneling it efficiently towards the photomultiplier tubes.

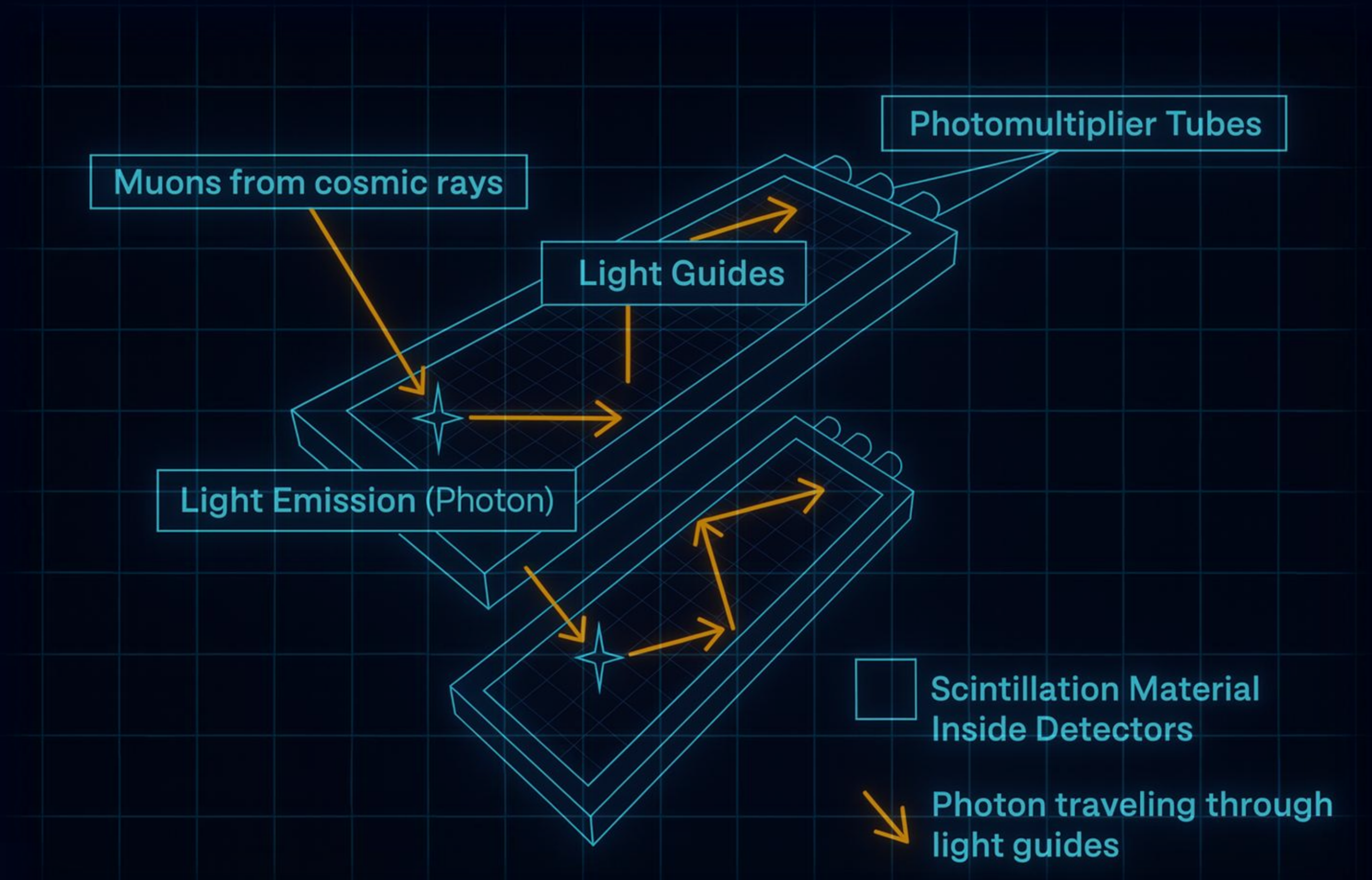


Figure 1.08 – Photon Emission Process

Photomultiplier Tubes (PMTs)

Photon to Electron Conversion & Amplification

The photomultiplier tubes (PMTs), which receive the photons generated by the scintillator material, are responsible for converting it into a measurable electric current.

When a photon reaches the photocathode of the PMT, the photocathode ejects an electron through the photoelectric effect. This electron is then accelerated and directed toward a series of electrodes called dynodes inside the PMT. At each dynode, the electron triggers the release of additional electrons upon impact, creating an amplified cascade that greatly multiplies the original signal.

The resulting large pulse of electrons is collected at the anode, which serves as the final electrode in the chain. The anode gathers the multiplied electrons and converts them into a measurable electrical current which is then passed to a signal processing module.

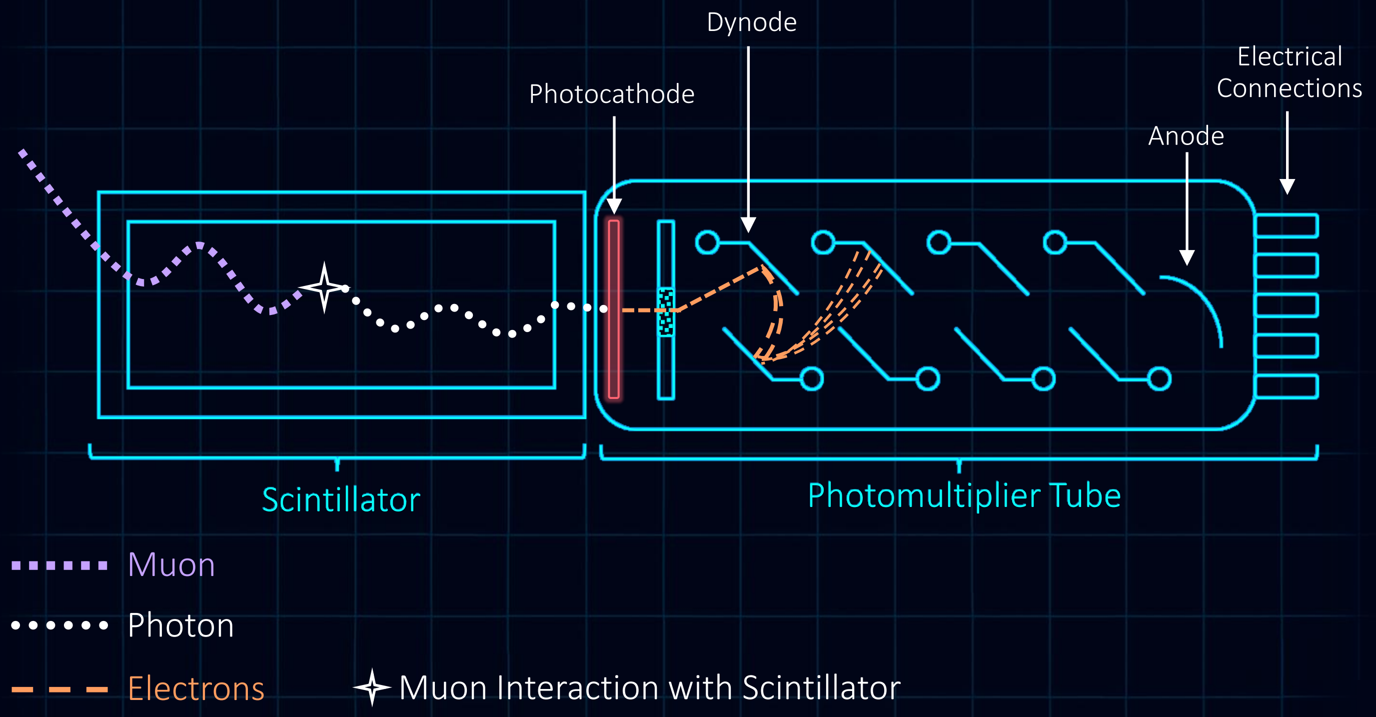


Figure 1.10 – Photomultiplier Tube

Cosmic Ray Detector Setup

Coincidence Detection

However, a single detector cannot reliably confirm a true particle interaction, as random noise or background radiation may produce false triggers.

To address this, we use two detectors stacked atop one another. Each detector then sends their signal, through the PMT to our signal processing module.

This setup enables our signal processing module to check for signals that arrive from both detectors within a narrow time frame and reject false signals, the ones only appearing on one detector, since random noise usually affects only one detector at a time.

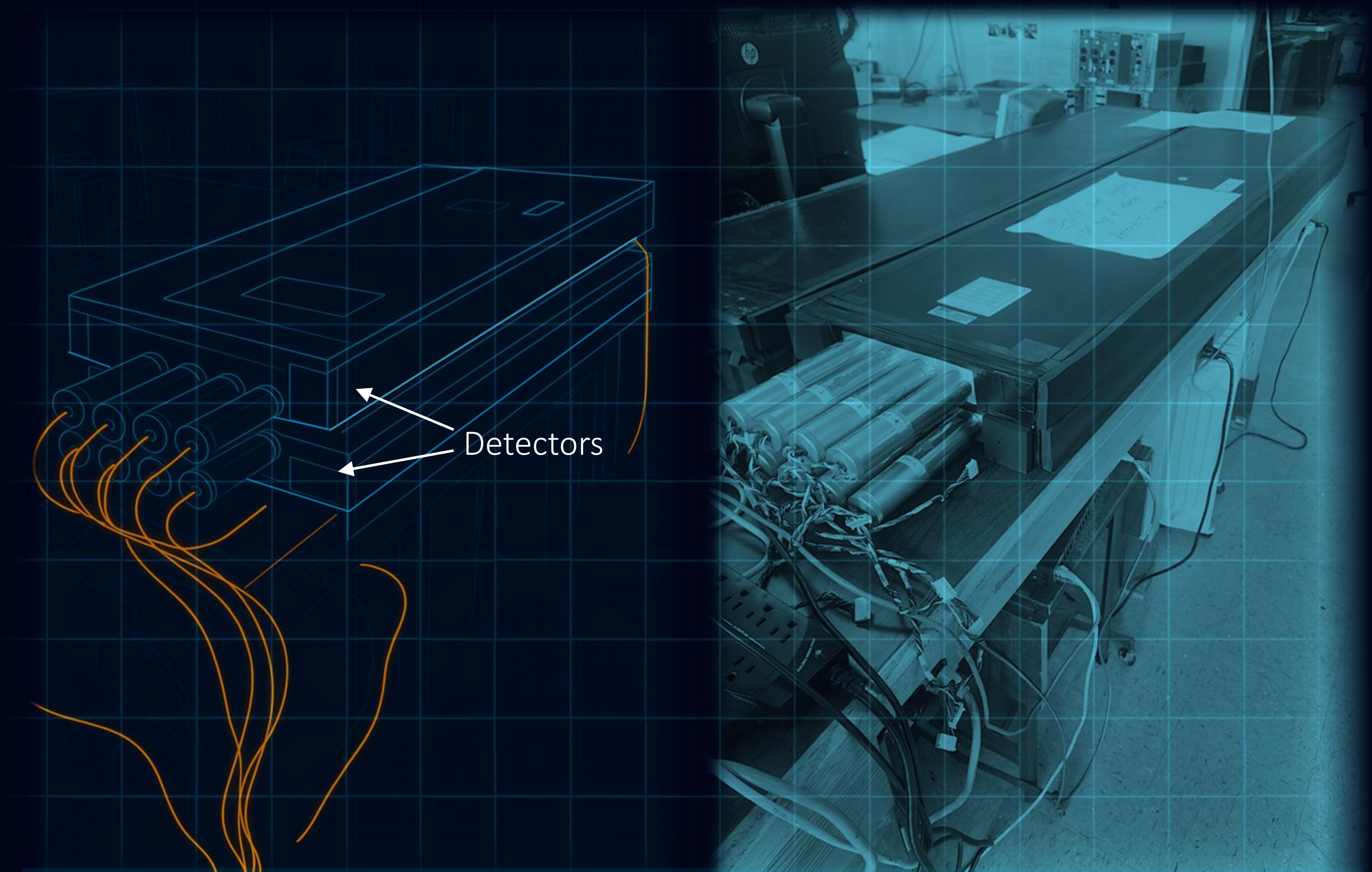


Figure 1.09 – Two Stacked Cosmic Ray Detectors

Signal Processing Module

Noise Filtering

The signal processing module compares the signals from the two stacked detectors. Because valid cosmic ray events (like passing muons) trigger both detectors at nearly the same time, the system uses this coincidence to filter out random noise or background radiation.

Amplification

Amplifies the weak output signal generated by the photomultiplier tube (PMT).

Pulse Shaping

The voltage pulses from the PMT are extremely brief—only about 20 to 40 nanoseconds wide. To accurately measure their peak voltage, each pulse is stretched using an RC integrator circuit with an operational amplifier (op-amp). This makes the pulse easier to analyze and measure.

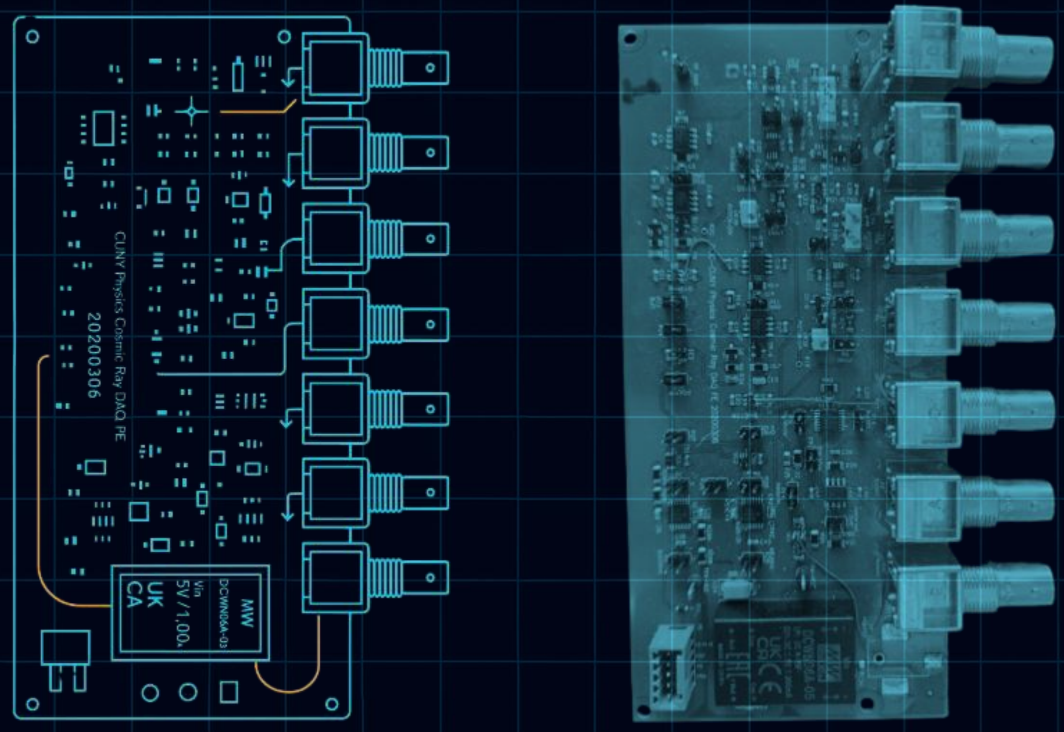


Figure 1.11 – Signal Processing Module

Arduino ATMega 2560

Timestamping

After passing through the Signal Processing Module, the signal is sent to the Arduino ATmega2560. When a muon passes through the detector the microcontroller records the exact moment the signal arrives as a timestamp.

This allows us to track when each event occurred, measure time intervals between them, and analyze event patterns over time.

Voltage Measurements

We also use the ATmega2560 to measure the voltage signal generated when a muon passes through our detector. Each time a muon interacts with the scintillator, it produces a flash of light that's converted into a small voltage pulse by the photomultiplier tube.

The microcontroller reads this voltage, allowing us to capture and analyze the amount of energy deposited by the muon.

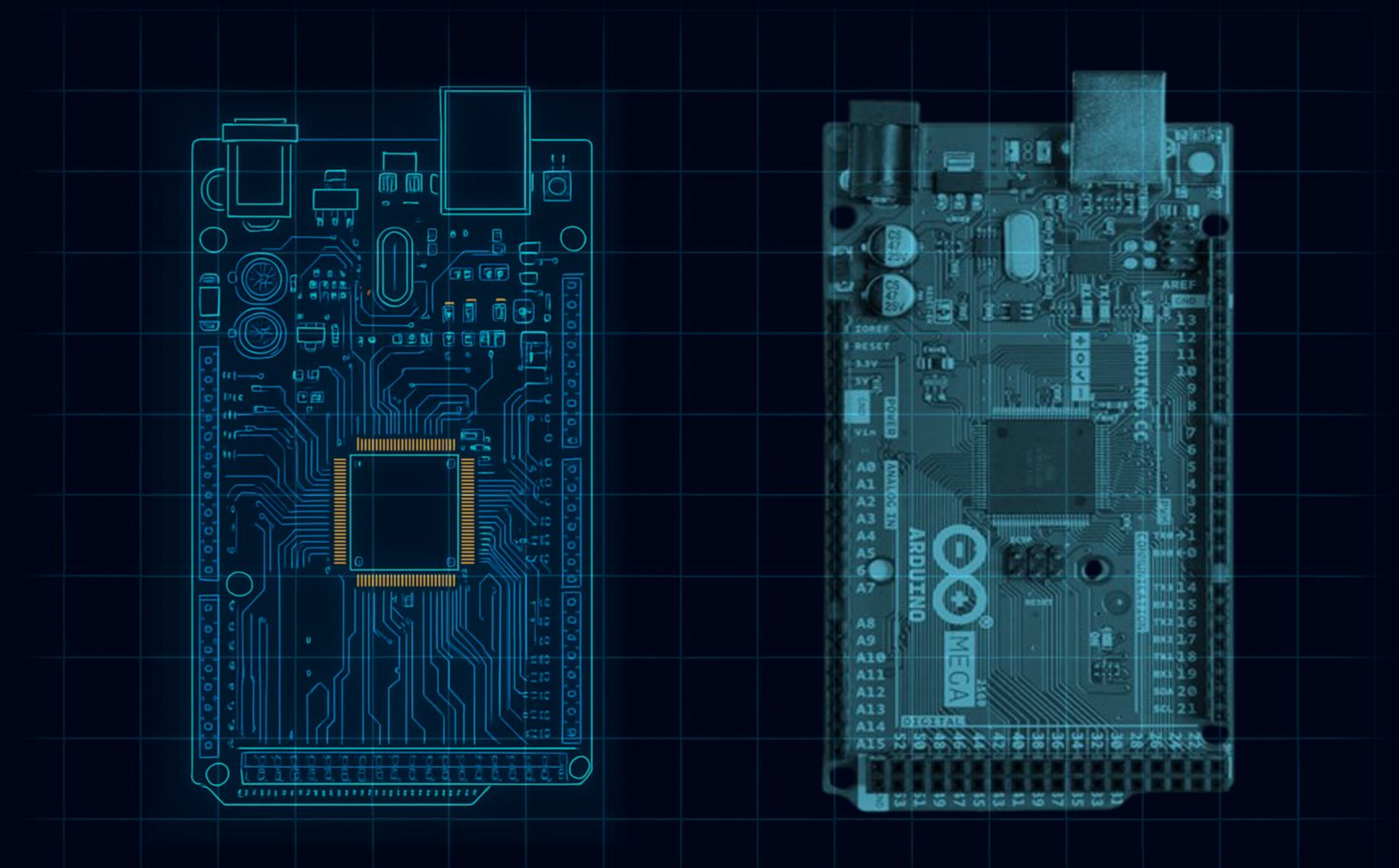


Figure 1.12 – Arduino ATMega 2560 Microcontroller

Analog to Digital Converter (ADC)

Signal Types

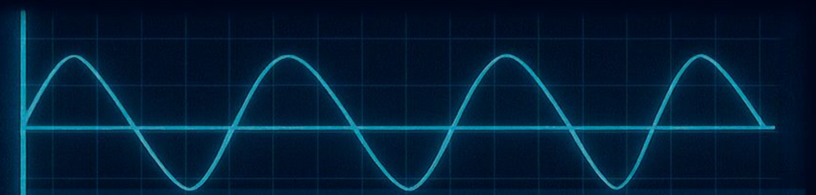


Figure 1.13 – Analog Signal

Analog signals vary smoothly and continuously over time, taking on any voltage within a given range.

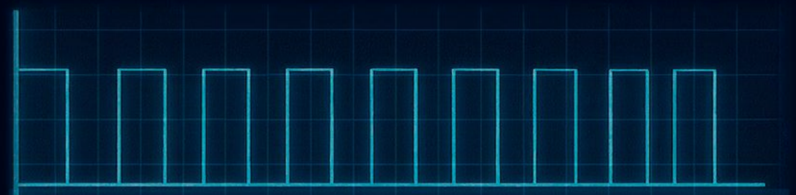


Figure 1.14 – Digital Signal

Digital signals use only two states, representing 0 or 1, which makes them easier for digital systems to process.

Conversion

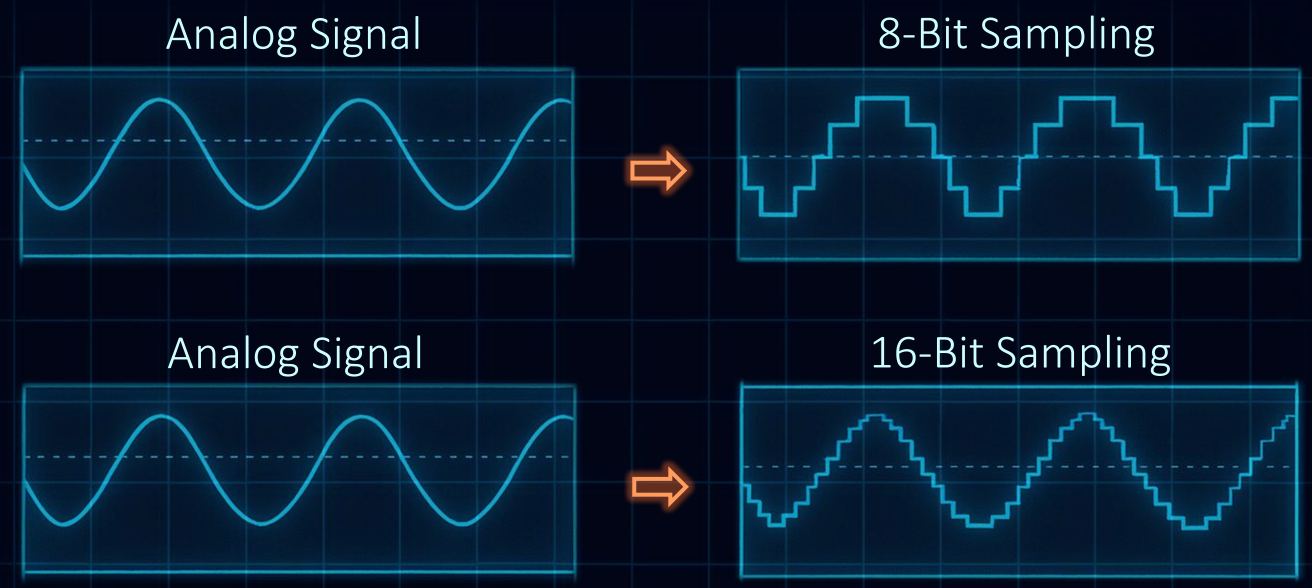


Figure 1.15 – Analog to Digital Conversion Process

To convert an analog signal, the ADC performs a series of checks to determine whether the signal is higher or lower than reference voltages, building a digital (binary) value that approximates the original signal. The precision of this approximation depends on the ADC's bit resolution; on the ATmega2560, the ADC provides 10 bits of resolution.

Timestamping

After passing through the Signal Processing Module, the signal is sent to the Arduino ATmega2560. When a muon passes through the detector the microcontroller records the exact moment the signal arrives as a timestamp.

This allows us to track when each event occurred, measure time intervals between them, and analyze event patterns over time.

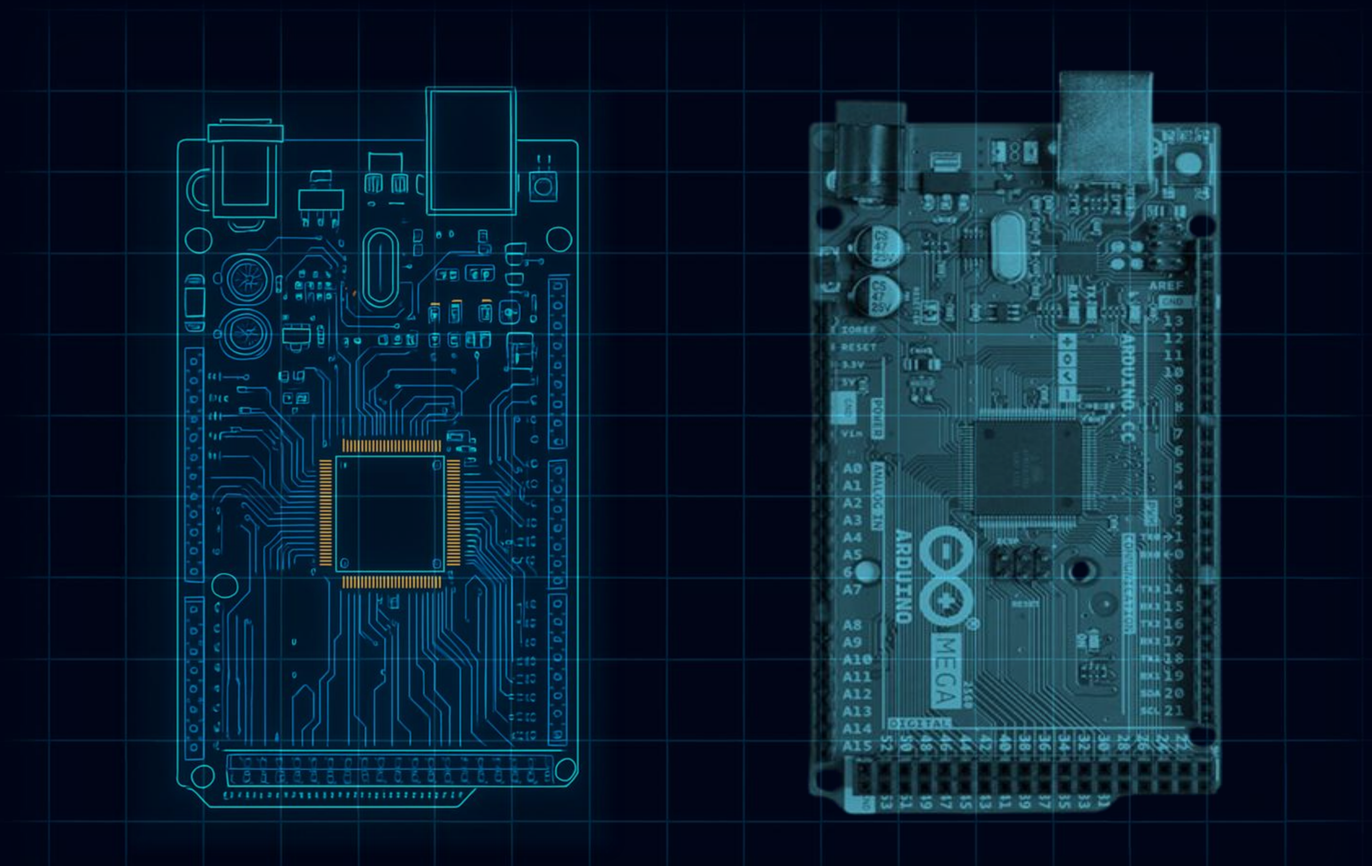


Figure 1.13 – Arduino ATMega 2560 Microcontroller

Applications

Intercollegiate Detection Array

In collaboration with colleges, we plan to create a multi-point detection array capable of capturing wide-area cosmic ray events.

Shower Size & Density

By comparing muon detections across multiple detectors in different boroughs, we can estimate the lateral spread and intensity of an air shower.

Estimating Primary Cosmic Ray Energy

The size and density of detected muon showers serve as indirect indicators of the primary cosmic ray's energy. A wider, denser shower suggests a higher-energy origin, possibly indicating ultra-high-energy cosmic rays (UHECRs).

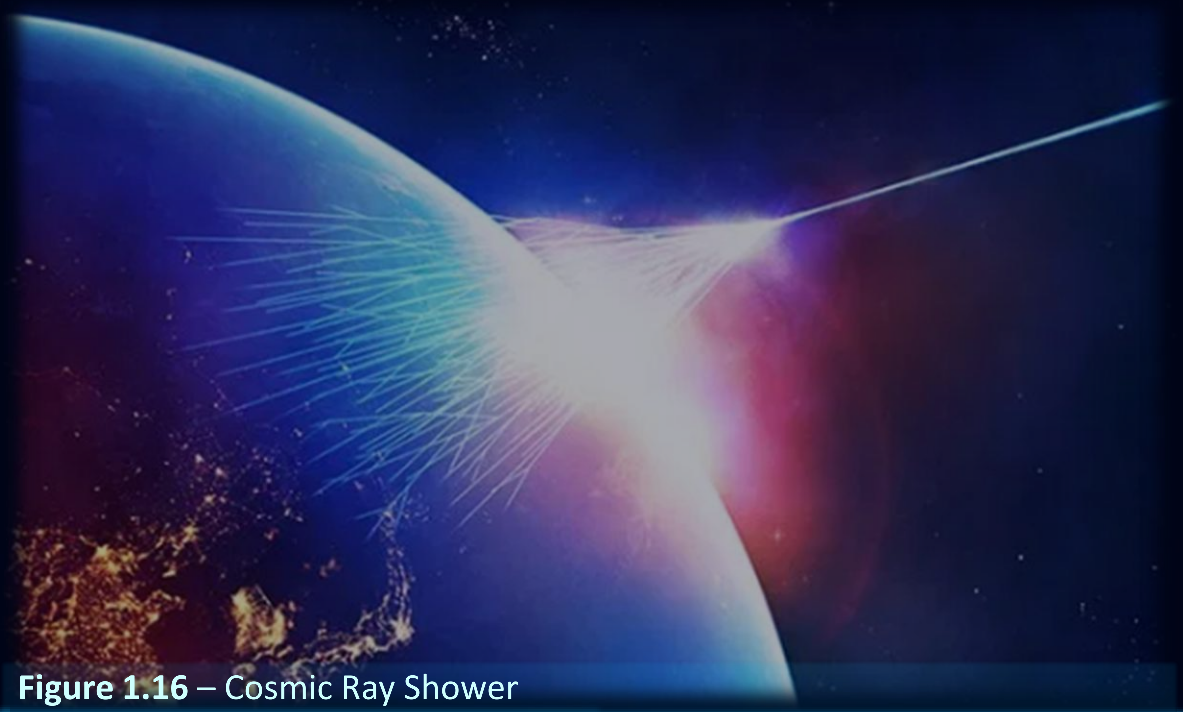


Figure 1.16 – Cosmic Ray Shower

UHECRs have energies exceeding anything we can generate in particle accelerators like the large hadron collider. Studying them allows us to probe fundamental physics at extreme energies, possibly revealing new particles or interactions.

Module I

Hardware Overview

QUEENSBOROUGH
COMMUNITY COLLEGE

What is Arduino?

Arduino is an open-source electronics platform that combines hardware and software to create interactive projects. It utilizes a variety of microcontroller-based boards, which can be programmed using the Arduino IDE (Integrated Development Environment), a software application where you write the code that sends the Arduino microcontroller instructions.

Data Exchange

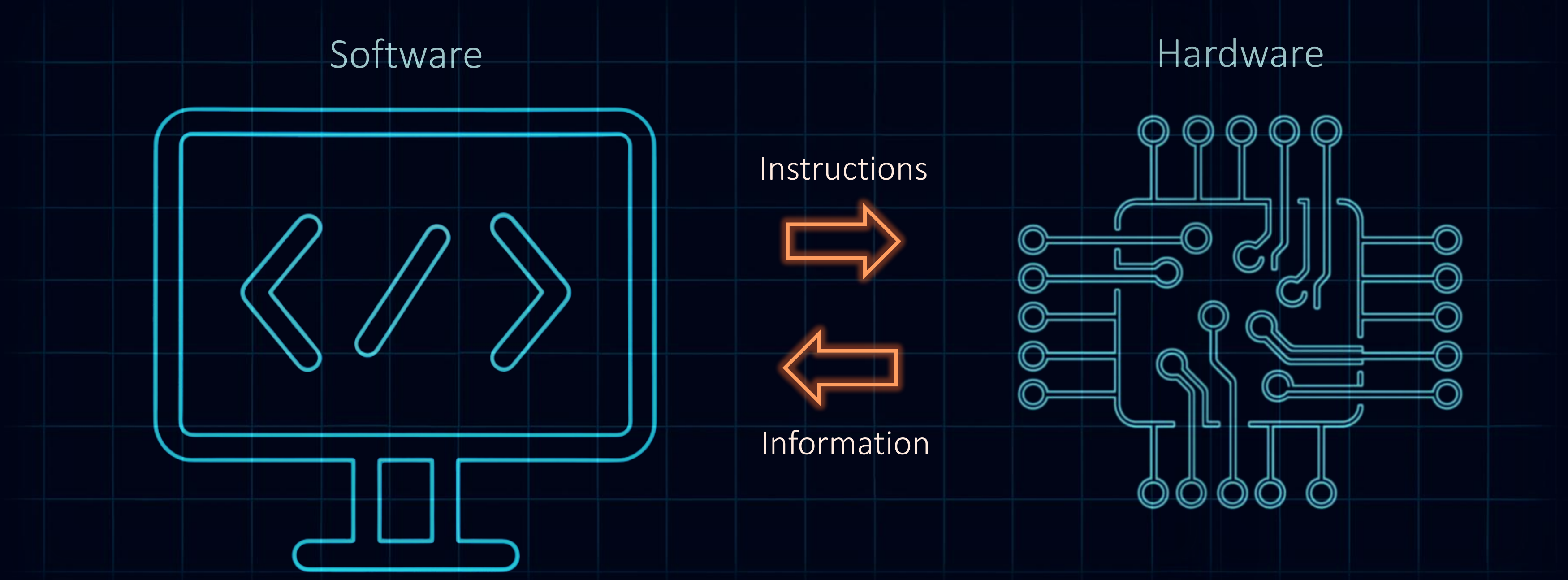


Figure 2.01 – Communication Diagram

Components & Accessories

Your first step should be to familiarize yourself with the hardware you'll be using. Understanding the purpose and function of each component is important for resolving troubleshooting issues and designing effective circuits. It also helps prevent damage by ensuring safe connections and simplifies the integration of components into your projects.

Hardware List:

- Raspberry Pi
- Arduino ATMega 2560
- Adafruit LED Backpack Counter
- Adafruit Ultimate GPS Breakout V3
- Adafruit BMP280 Pressure & Temperature Sensor
- XBee3 Radio Module
- XBee Dongle
- Jumper Wires
- Breadboard

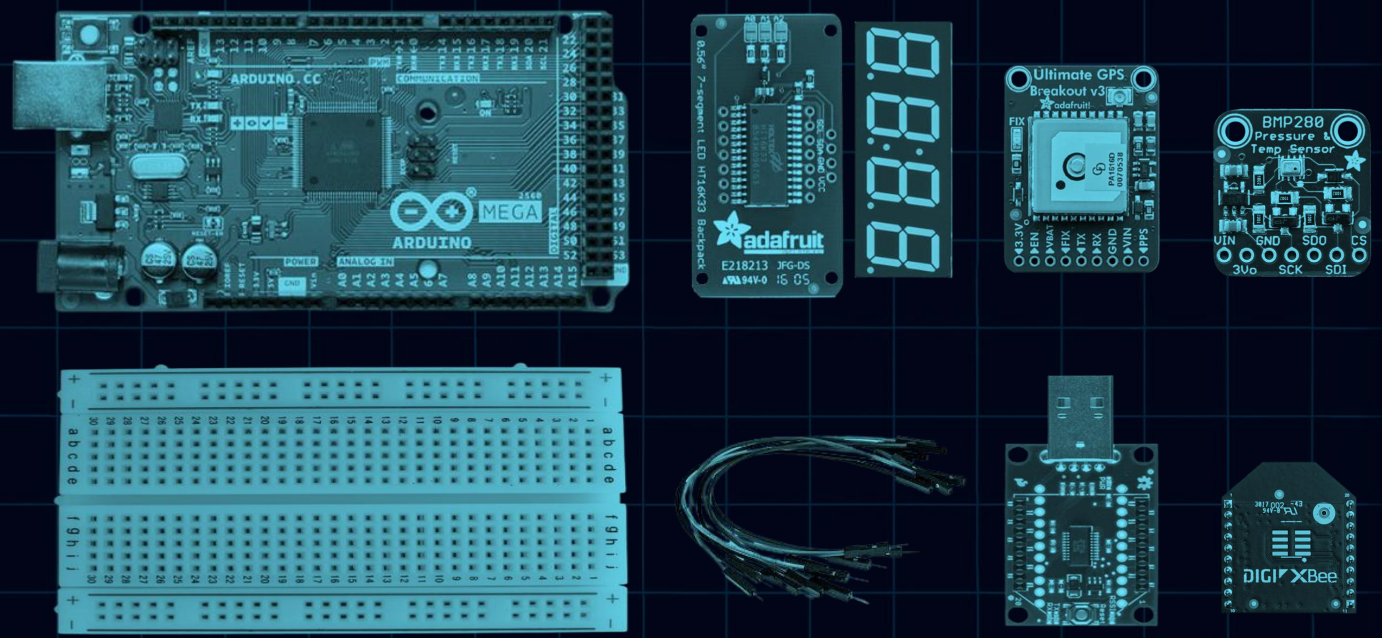


Figure 2.02 – Experiment Hardware

Arduino ATMega2560 Microcontroller

The Arduino ATMega 2560 is a type of microcontroller, which is a small computer on a single circuit board. It is used to control various electronic devices and projects. Imagine it as the "brain" that tells other parts what to do.

Purpose

Inputs:
It can read digital or analog signals (via an onboard ADC) from external equipment, GPS modules, sensors, etc.

Processing:
The microcontroller runs your C++ code. That code can do calculations, filter signals, apply logic decisions, and manage timing — for example, counting pulses or interpreting sensor measurements.

Outputs:
It can also control things like LEDs or motors by sending signals to them.

Refer to the appendix for a full pin breakdown.

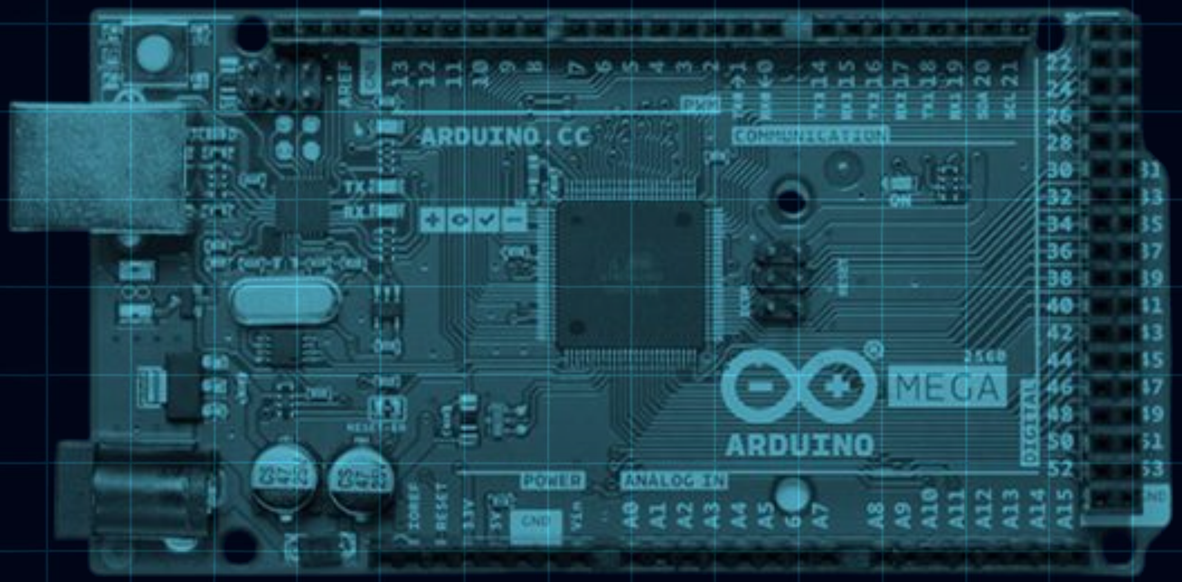


Figure 2.03 – Arduino ATMega 2560 Microcontroller

Adafruit BMP280 Pressure & Temperature Sensor

The BMP280 is a combined barometric pressure and temperature sensor designed for precise environmental measurements. It can measure atmospheric pressure and temperature while also estimating altitude by calculating changes in air pressure. Communicating over I²C or SPI, the BMP280 integrates easily with microcontrollers like Arduino.

Purpose

- Combines barometric pressure and temperature sensing in a compact package
- Measures atmospheric pressure with high precision (±1 hPa)
- Measures temperature with accuracy of around ±1 °C

Refer to the appendix for a full pin breakdown.

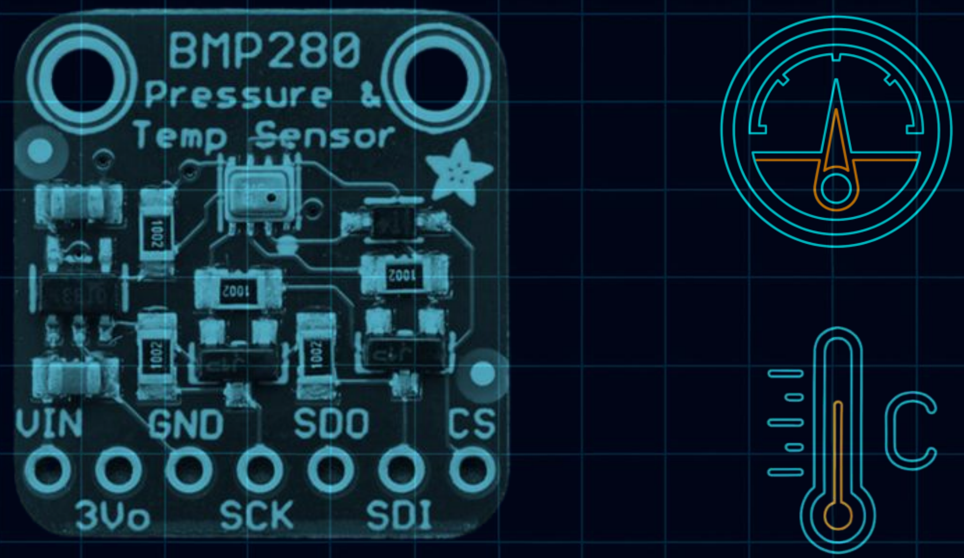


Figure 2.04 – Adafruit BMP280 Pressure & Temperature Sensor

Adafruit LED Backpack Counter

The Adafruit LED Backpack Counter is a display module. It can show numbers, symbols, or simple characters and is commonly used for counters, timers, or status indicators. The LED Backpack Counter offers a simple way to add visual feedback to your circuit designs.

Purpose

- I²C interface for easy wiring with Arduino or other microcontrollers
- Integrated LED driver chip simplifies control of 7-segment or matrix displays
- Supported by Adafruit's open-source libraries for fast setup and coding

Refer to the appendix for a full pin breakdown.

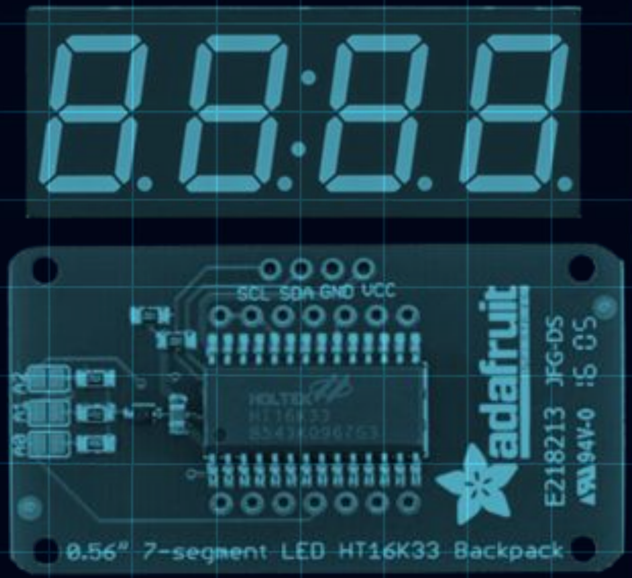


Figure 2.05 – Adafruit LED Backpack Counter

Adafruit Ultimate GPS Breakout V3

The Adafruit GPS module is a compact and highly accurate positioning device that uses signals from global satellite networks to determine location, speed, and time data. Provides precise time data and includes features like built-in antenna support for reliable operation even in challenging environments.

Purpose

- Provides precise location information, including latitude, longitude, and altitude.
- Outputs data in standard NMEA format for easy integration with microcontrollers like Arduino.
- Offers accurate time data based on GPS signals, including UTC (Coordinated Universal Time).

Refer to the appendix for a full pin breakdown.

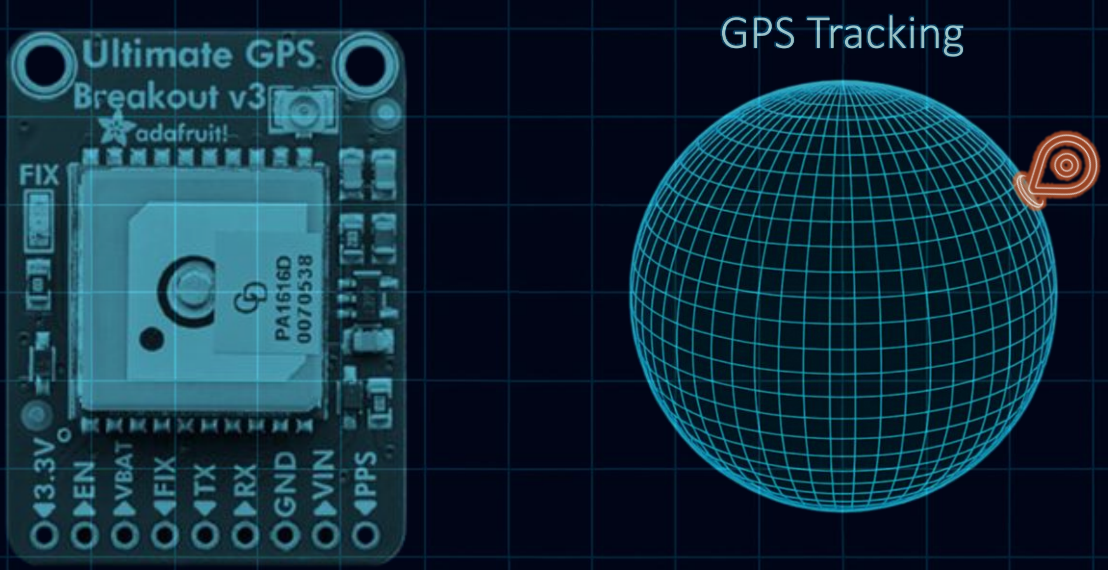


Figure 2.06 – Adafruit Ultimate GPS Breakout V3

XBee Dongle

The XBee dongle is a simple plug-and-play USB device that allows a computer to communicate with XBee radio modules. It acts as a bridge between your PC and an XBee network, enabling configuration, data monitoring, and testing. With built-in drivers and compatibility with tools like XCTU, it offers a way to integrate wireless communication without complicated wiring.

Purpose

- USB plug-and-play interface for simple PC-to-XBee communication
- Works seamlessly with XCTU software for configuration and testing
- Allows wireless programming, debugging, and data monitoring of XBee modules

Refer to the appendix for a full pin breakdown.

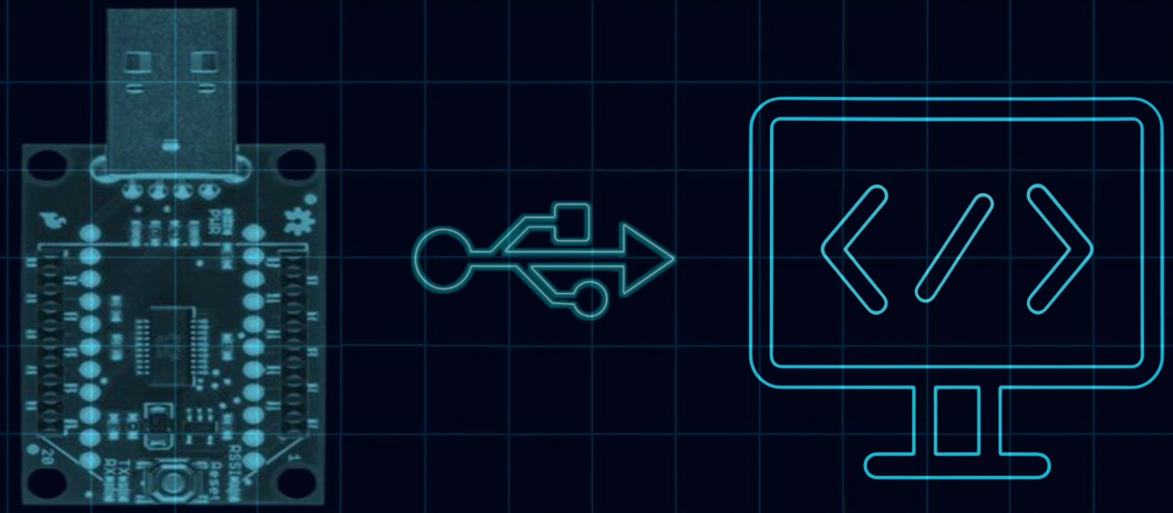


Figure 2.07 – XBee Dongle

XBee3 Radio Module

The XBee3 is a powerful, compact wireless communication module. It offers secure, low-power networking with flexible configuration options, allowing devices to communicate over short to medium distances. Paired with tools like XCTU for configuration and diagnostics, the XBee3 makes it easy to integrate wireless connectivity with minimal effort.

Purpose

- Built-in MicroPython interpreter for simple edge processing without extra microcontrollers
- Offers reliable wireless data transfer
- Easily configured and diagnosed using Digi XCTU software
- Standard serial interface for easy Arduino integration

Refer to the appendix for a full pin breakdown.

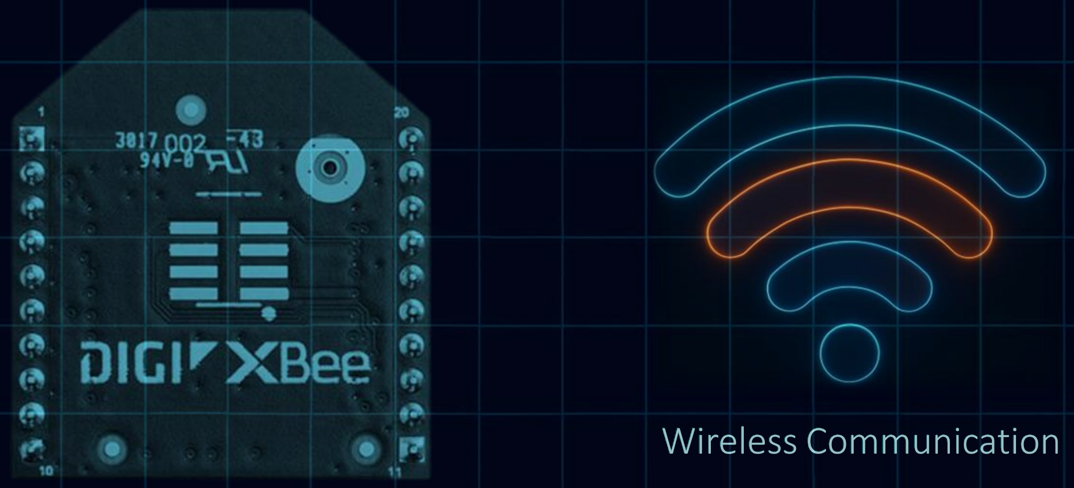


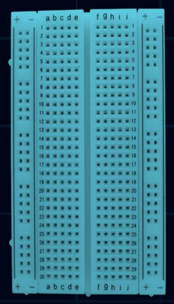
Figure 2.08 – XBee 3 Radio Module

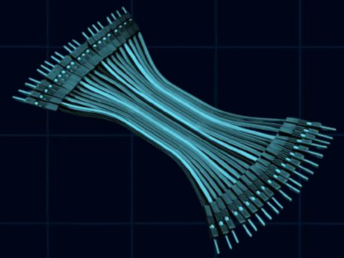
Breadboard & Jumper Wires

Breadboards provide a platform for rapidly prototyping electronic circuits. Their internal metal clips connect rows of holes, allowing easy insertion and rearrangement of components without permanent connections. Paired with male-to-male or female-to-male jumper wires, breadboards enable flexible routing of signals and power, supporting experimentation with sensors and microcontrollers.

Purpose

- Internal metal clips connect the rows of holes horizontally (a-e) & (f-j) but not across the middle divider
- Internal metal clips connect the rows along the positive and negative rails at the edge of the breadboard vertically
- Jumper wires are used to access the pins inserted into the breadboard, transmitting the signals from their respective connections





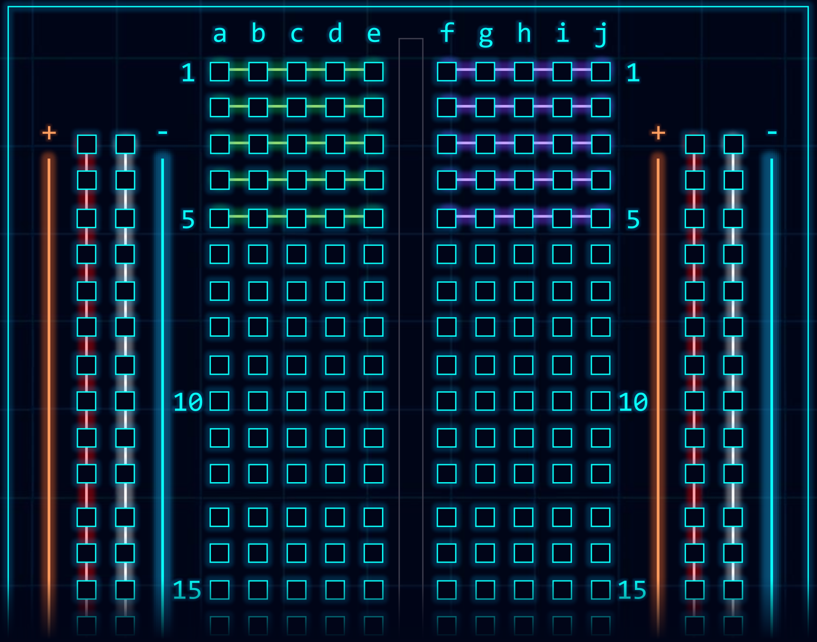


Figure 2.09 – Breadboard Internal Metal Clip Connections

Module II

Software Overview

QUEENSBOROUGH COMMUNITY COLLEGE

Required Software

This experiment relies on a suite of essential software tools to enable effective programming, configuration, and communication. Together, these tools form a cohesive environment that supports efficient development, robust configuration, and reliable data exchange across the entire experimental workflow.

Application List:

- Arduino Integrated Development Environment
- Digi XCTU Configuration & Test Utility Software
- Microsoft Excel
- PuTTY

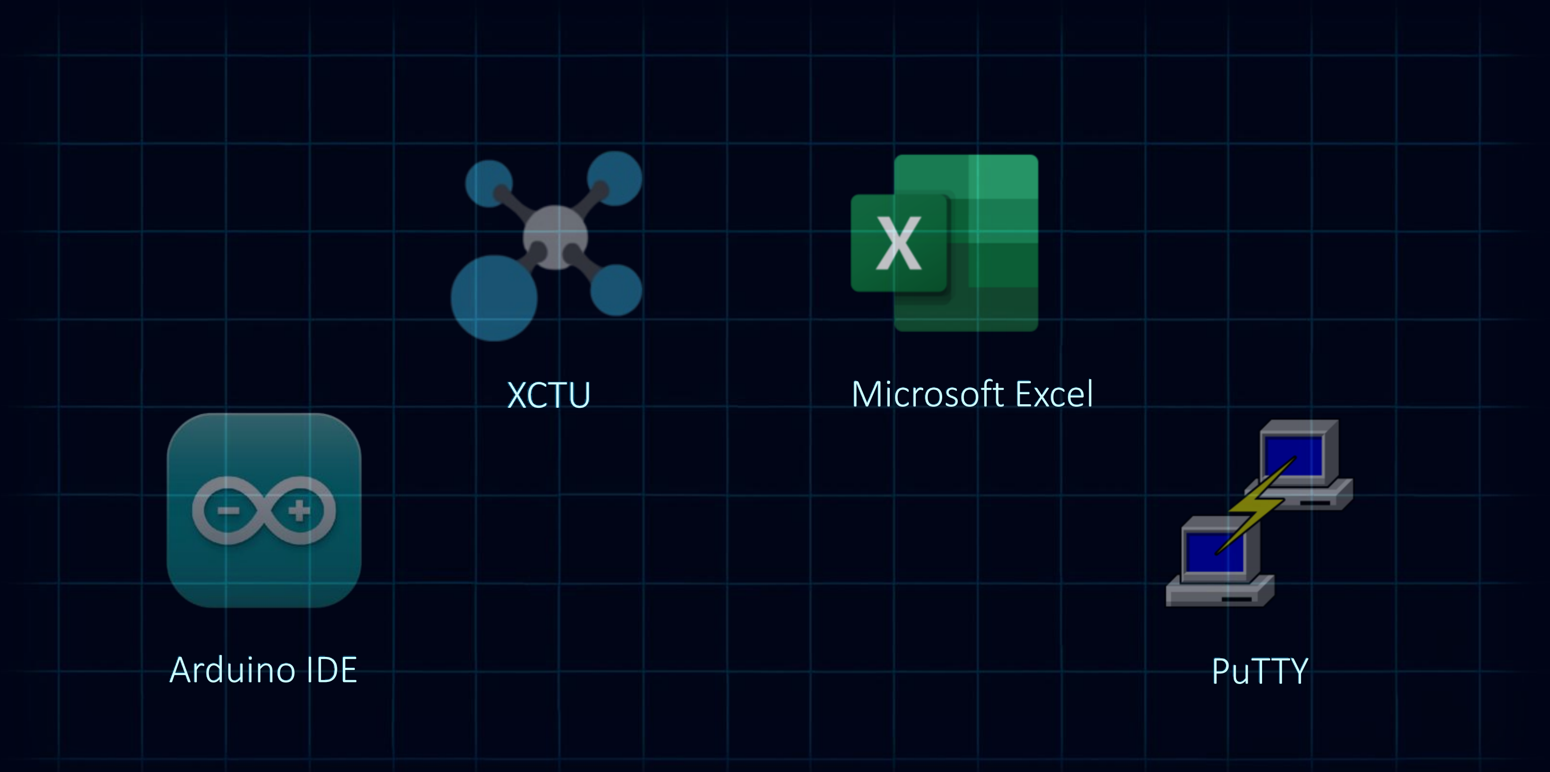


Figure 3.01 – Software Suite

Arduino IDE

The Arduino Integrated Development Environment (IDE) is a free software application used to develop programs for Arduino boards. It provides a code editor, compiler, and a serial monitor to observe data from the board.

Features:

- Supports writing, compiling, and uploading C/C++ code to Arduino boards
- Includes a built-in serial monitor for real-time data observation
- Simplifies managing libraries and third-party board packages
- Provides an intuitive editor with syntax highlighting and basic error checking

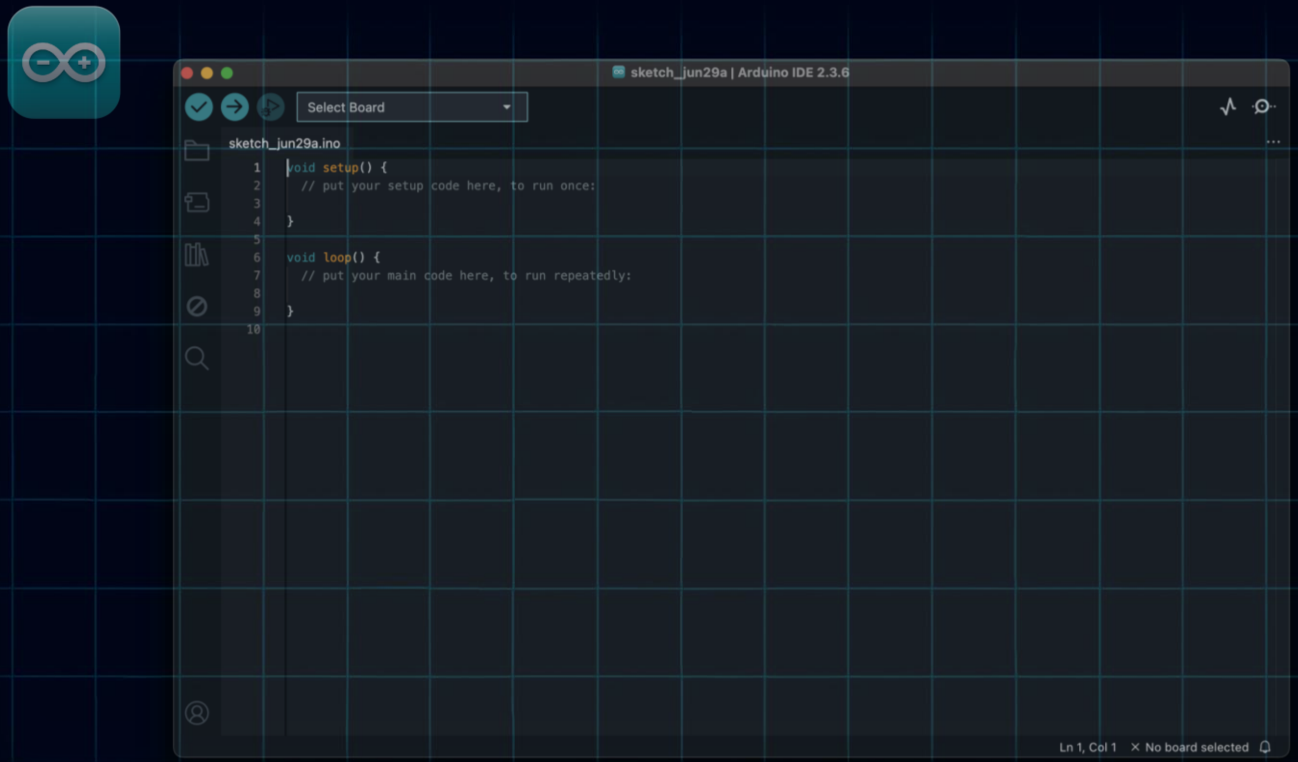


Figure 3.02 – Arduino IDE Interface

Digi XCTU

XCTU is a configuration and testing utility developed by Digi for managing XBee wireless modules. It provides a user-friendly interface to configure device parameters, update firmware, and establish communication settings. XCTU also includes tools for network mapping, range testing, and frame analysis, making it easier to diagnose connection issues and validate wireless performance.

Features:

- Supports firmware updates and parameter adjustments
- Includes network mapping and range-testing tools
- Offers frame analysis for detailed packet inspection
- Simplifies diagnosing and resolving wireless communication issues

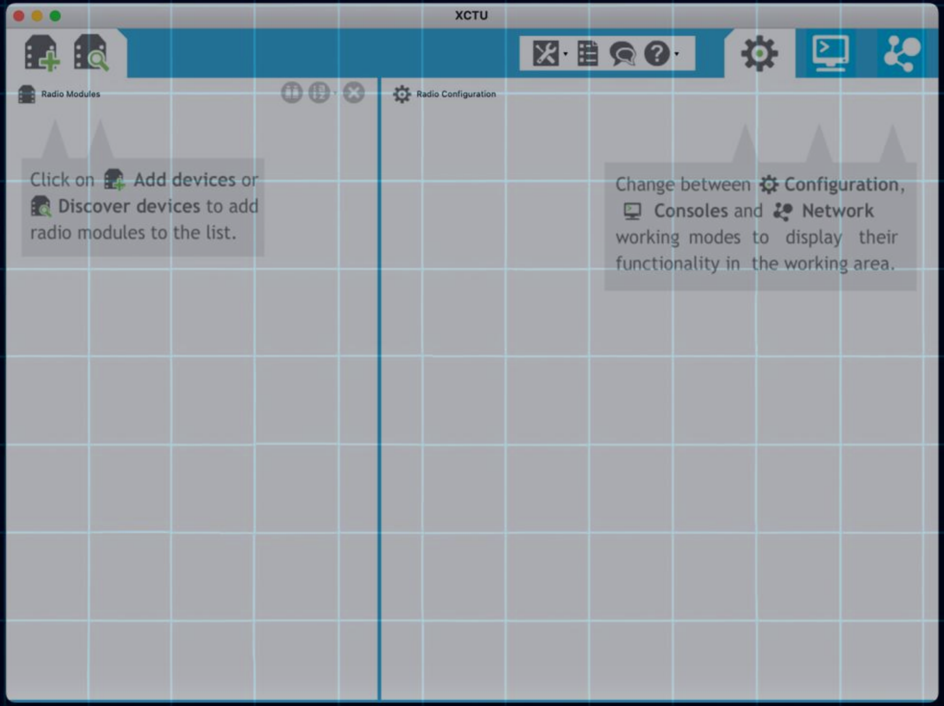


Figure 3.03 – XCTU Interface

Putty

PuTTY is a lightweight, open-source terminal emulator commonly used for serial and network communications. PuTTY can capture and log output from the Arduino IDE or other serial sources to a text file, providing a simple and effective way to archive test data for later analysis.

Features:

- Provides a simple interface for quickly testing bidirectional serial communication with your board.
- Supports multiple protocols but for Arduino, you'll mainly use the serial (COM port) mode.
- Handy for saving serial data to a text file for later analysis or debugging.

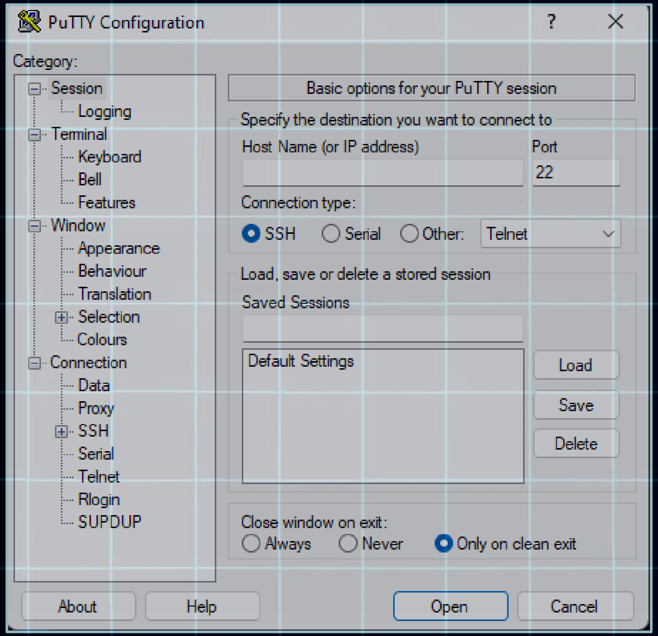


Figure 3.04 – PuTTY Interface

Microsoft Excel

We use Excel to analyze our data because it makes it easy to organize, visualize, and interpret the information we collect. After recording muon detection events, we import the raw data directly from PuTTY into Excel. From there we can plot trends and identify patterns in the timing of events.

Features:

- **Easy Data Import:**

  Copy raw output from PuTTY directly into Excel for quick access and organization.

- **Graphing Tools:**

  Create scatter plots, line graphs, and histograms to visualize event patterns.

- **Text Manipulation & Math Formulas:**

  Enables users to process text and perform mathematical calculations directly within cells for efficient data analysis and formatting.

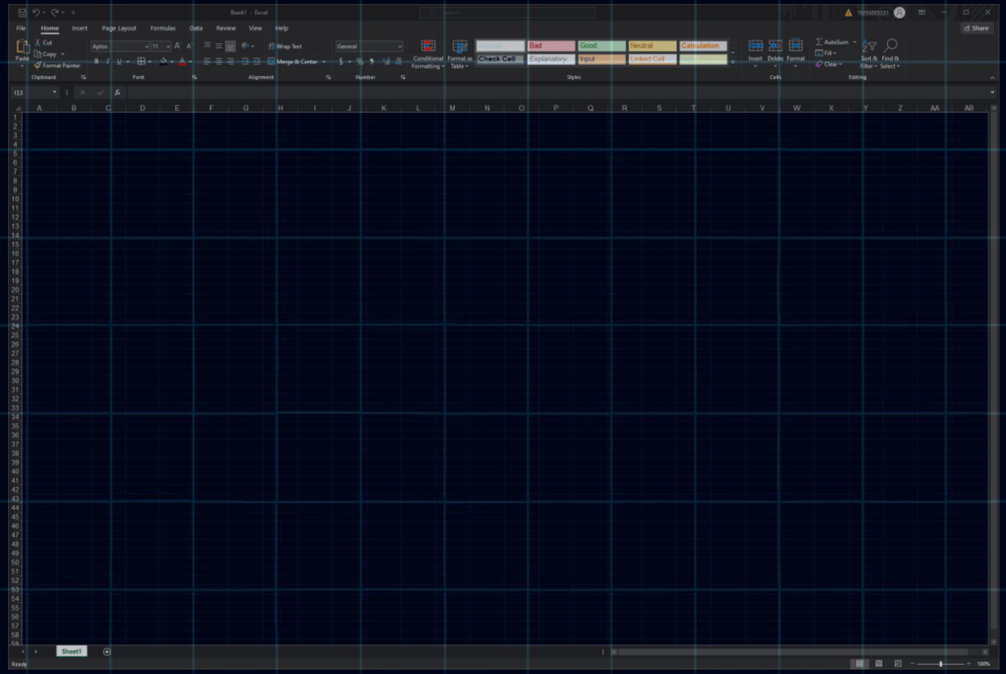


Figure 3.05 – Microsoft Excel Interface

Module III

Arduino IDE Setup

QUEENSBOROUGH COMMUNITY COLLEGE

Arduino to PC USB Connection

Connecting the USB Type-B cable between your computer and the Arduino board establishes a bidirectional communication link, allowing your computer to upload code to the microcontroller while also enabling the Arduino to send data back to the computer.

Connect the USB Type-B cable to your computer and the Arduino board.

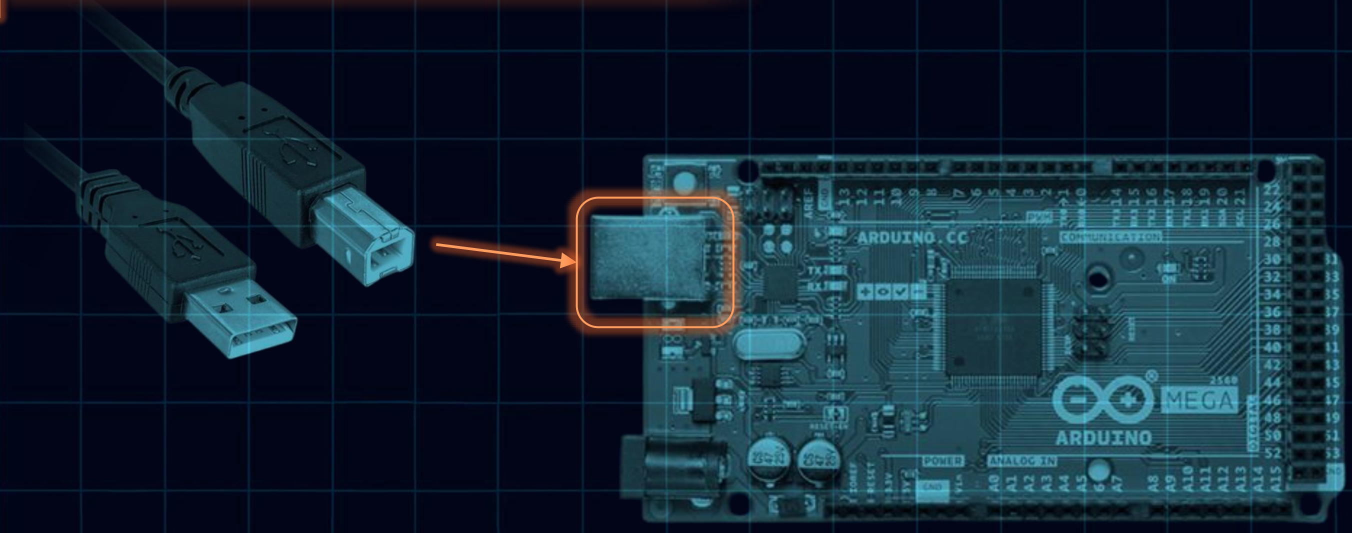
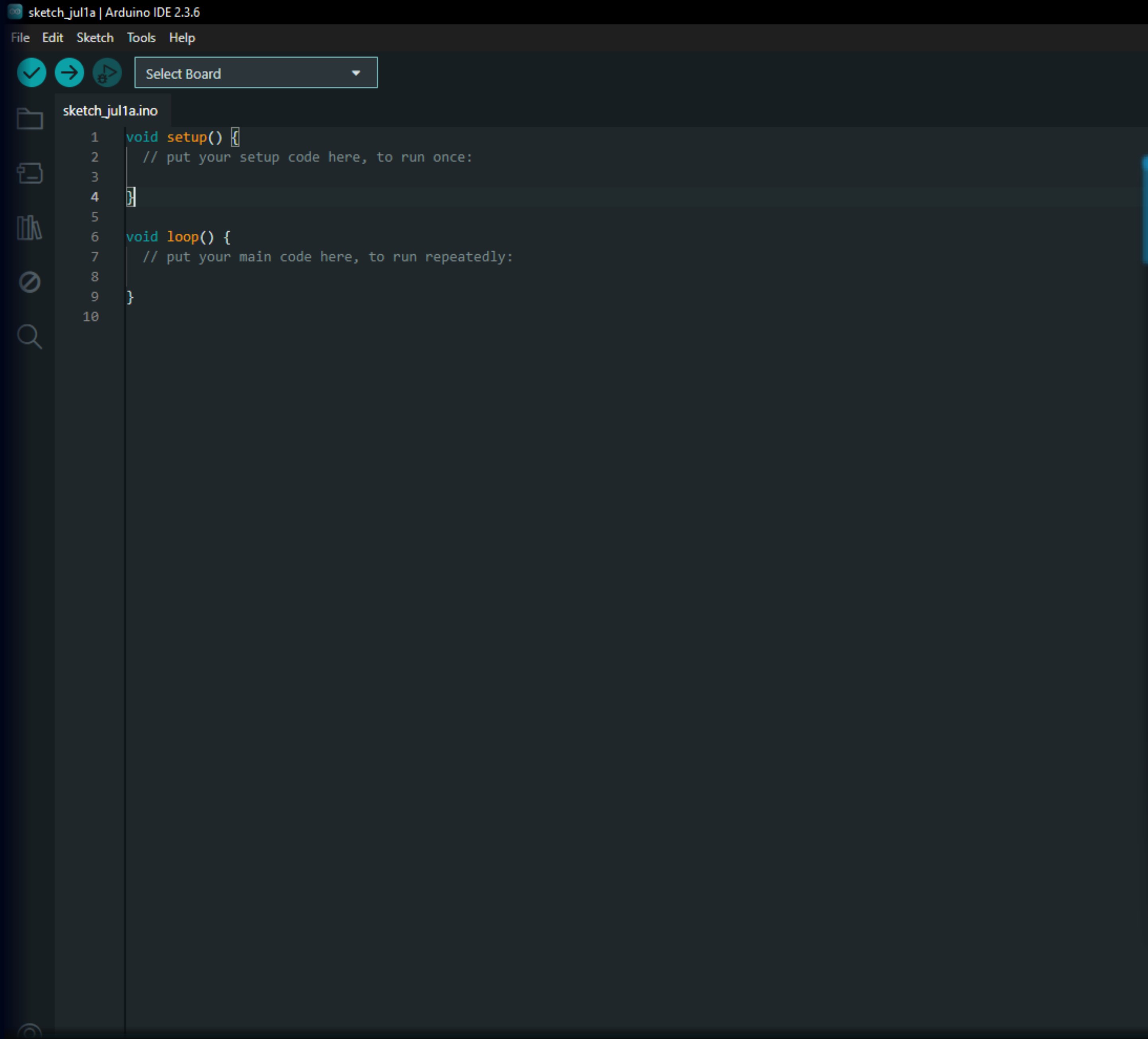


Figure 4.01 – Arduino-PC USB Connection

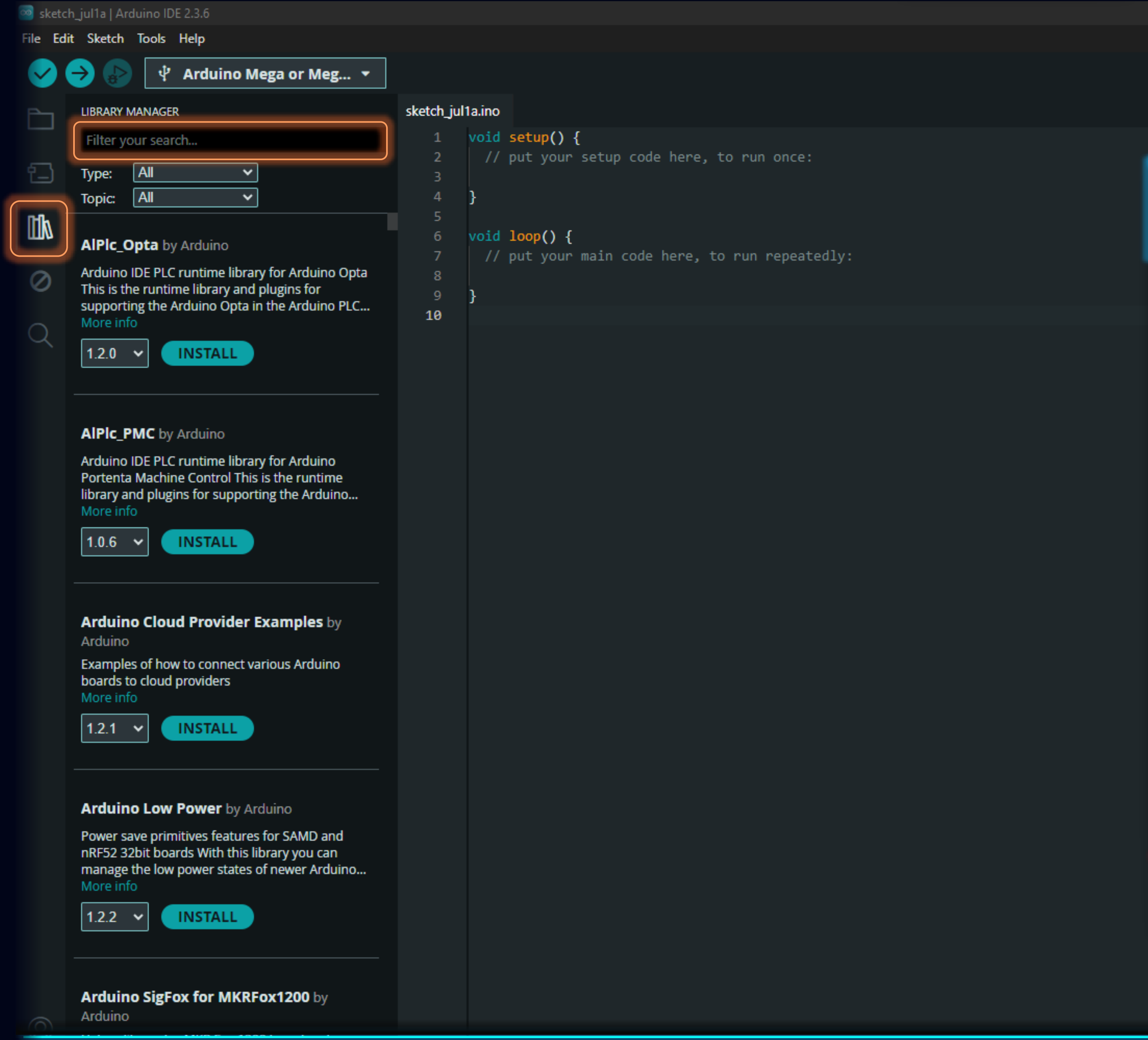


Launch Program

Locate the Arduino IDE icon on your desktop and launch the program

Sketches

The code you write in the Arduino IDE is called a sketch, and the Arduino compiler within the program handles all the setup to convert it into machine language for the microcontroller. It utilizes a simplified subset of C++ with a few custom libraries simplifying C++ to be more accessible for prototyping and hardware interaction.



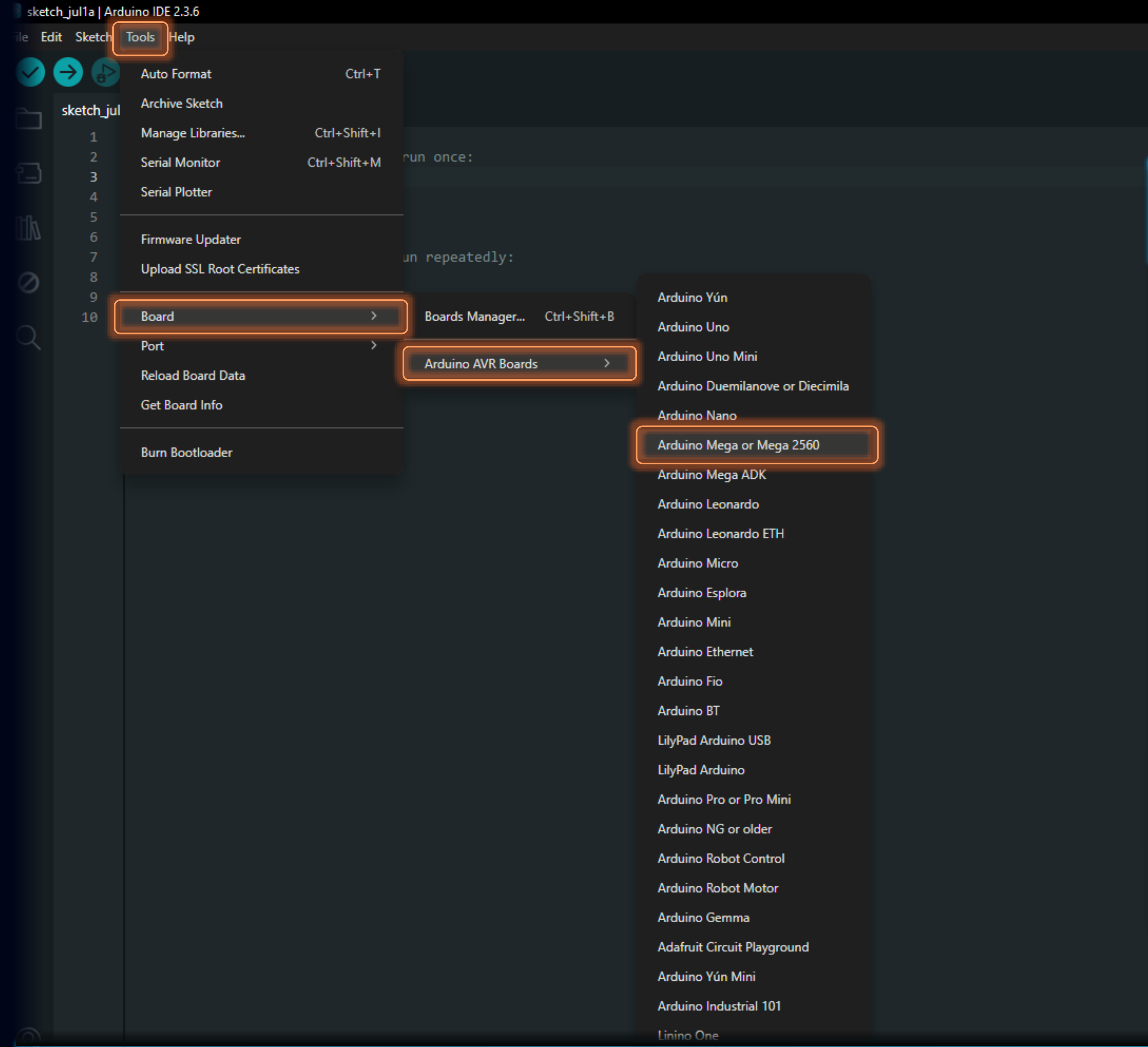
Essential Libraries

1) Click on the Libraries icon in the left-hand panel of the Arduino IDE.
2) Use the search bar to find and install the latest versions of the following libraries:
   - Adafruit BMP280 Library
   - Adafruit GPS Library
   - Adafruit LED Backpack
   - TimerOne

What are Libraries?

Arduino libraries are packaged collections of functions and drivers that extend the Arduino's capabilities to work with specific hardware or features.

Be sure to select Install All when prompted about dependencies.

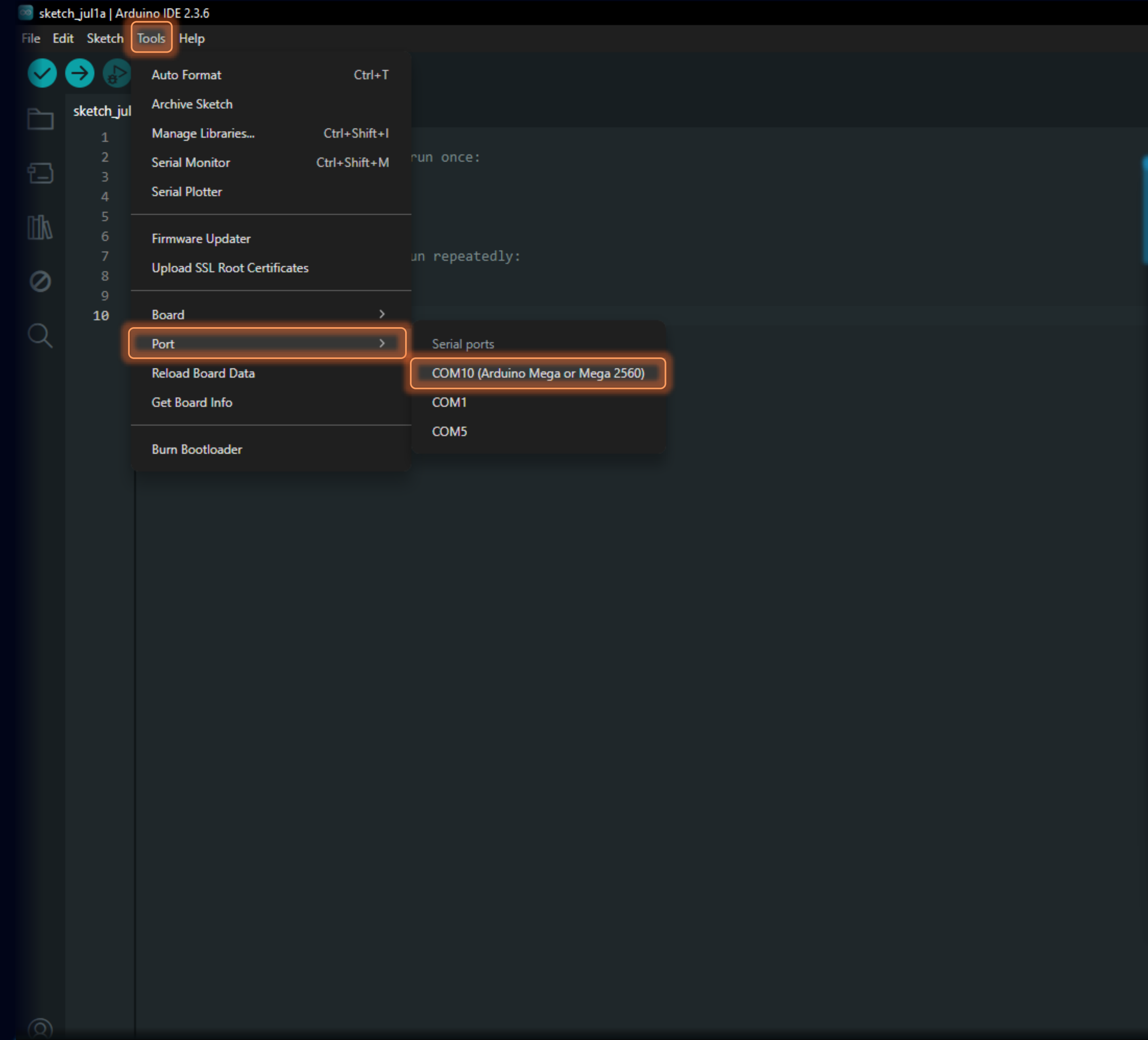


Board Selection

1) Navigate to the toolbar and select Tools.
2) Go to Boards > Arduino AVR Boards
3) Select the model of the board you are using. In this case, it is the Arduino Mega or Mega 2560.

AVR Boards

Selecting the correct board in the Arduino IDE is crucial before uploading your sketch. The board selection tells the IDE which microcontroller architecture, clock settings, and memory map to target during compilation and uploading.



Port Selection

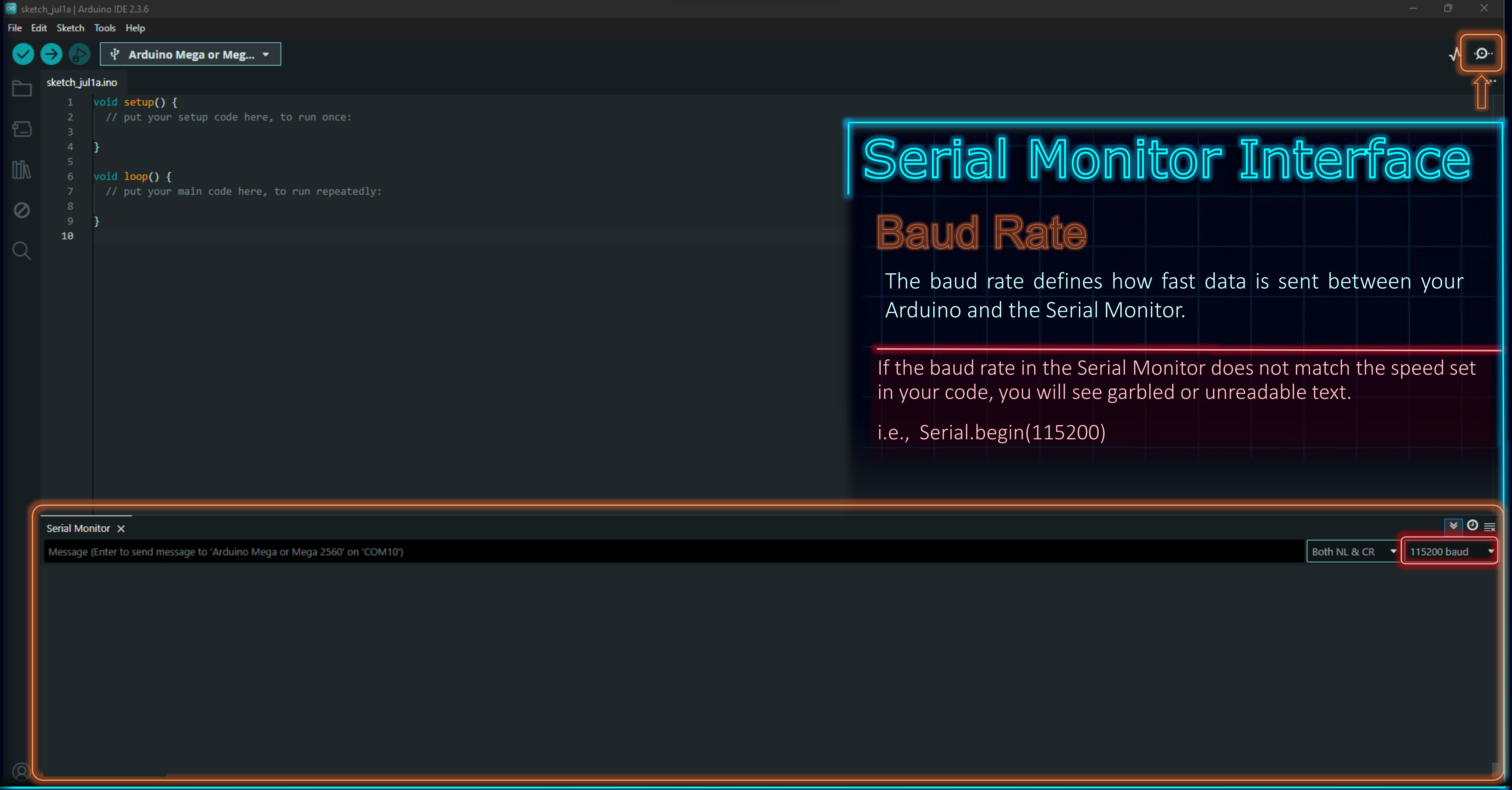
1) Navigate to the toolbar and select Tools.

2) Go to Ports.

3) Select the COM port that shows the model of your microcontroller alongside it.

COM Ports

Choosing the correct port under ensures the IDE knows where to send uploads and where to listen for serial data. If you pick the wrong port, uploads will fail, or you won’t see any serial monitor output.



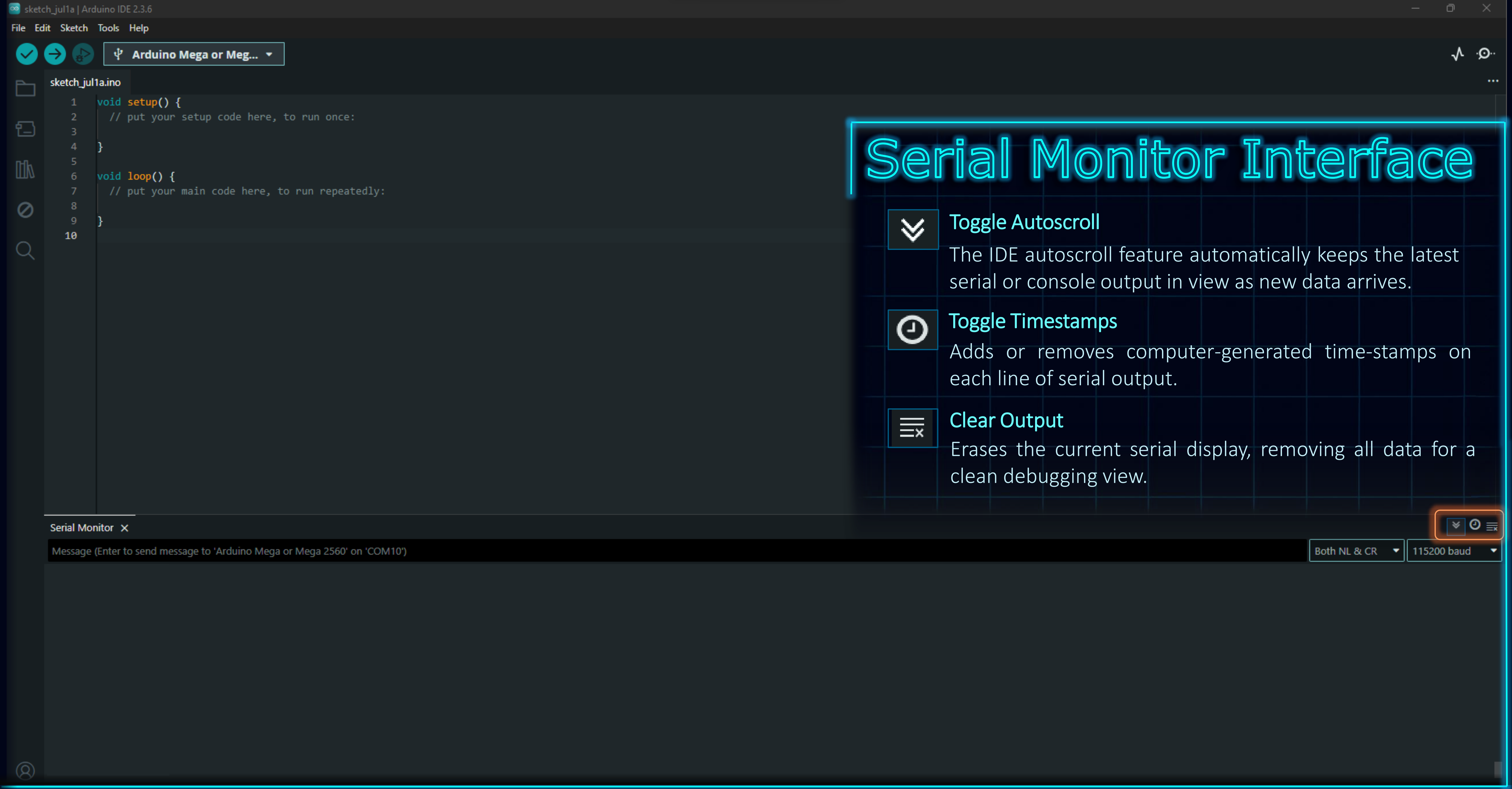
Serial Monitor Interface

Baud Rate

The baud rate defines how fast data is sent between your Arduino and the Serial Monitor.

If the baud rate in the Serial Monitor does not match the speed set in your code, you will see garbled or unreadable text.

i.e., Serial.begin(115200)



Serial Monitor Interface

Toggle Autoscroll

The IDE autoscroll feature automatically keeps the latest serial or console output in view as new data arrives.

Toggle Timestamps

Adds or removes computer-generated time-stamps on each line of serial output.

Clear Output

Erases the current serial display, removing all data for a clean debugging view.

Module IV

Adafruit BMP280 Temperature & Pressure Sensor

QUEENSBOROUGH COMMUNITY COLLEGE

BMP280 Operational Test

Follow the instructions provided to complete an operational test, verifying that the BMP280 sensor is functioning correctly, returning valid environmental data, and properly integrated into the system. Successful completion of this test confirms the sensor is ready for use in your application.

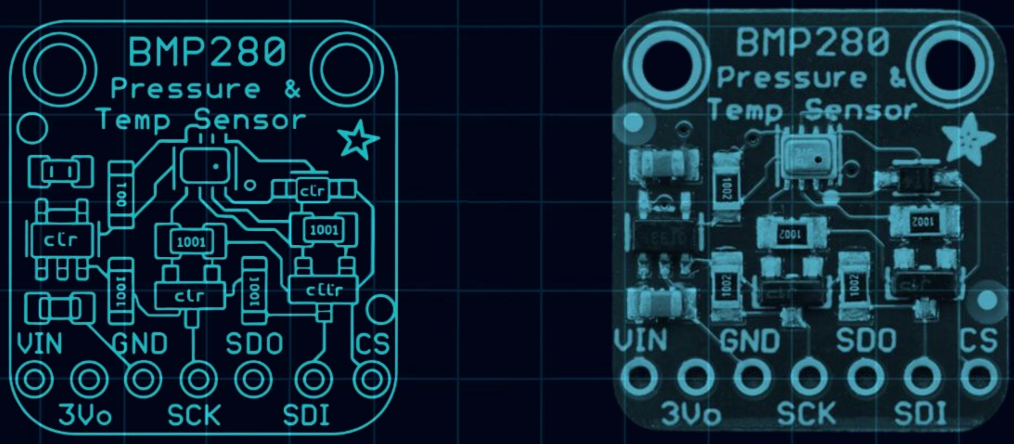
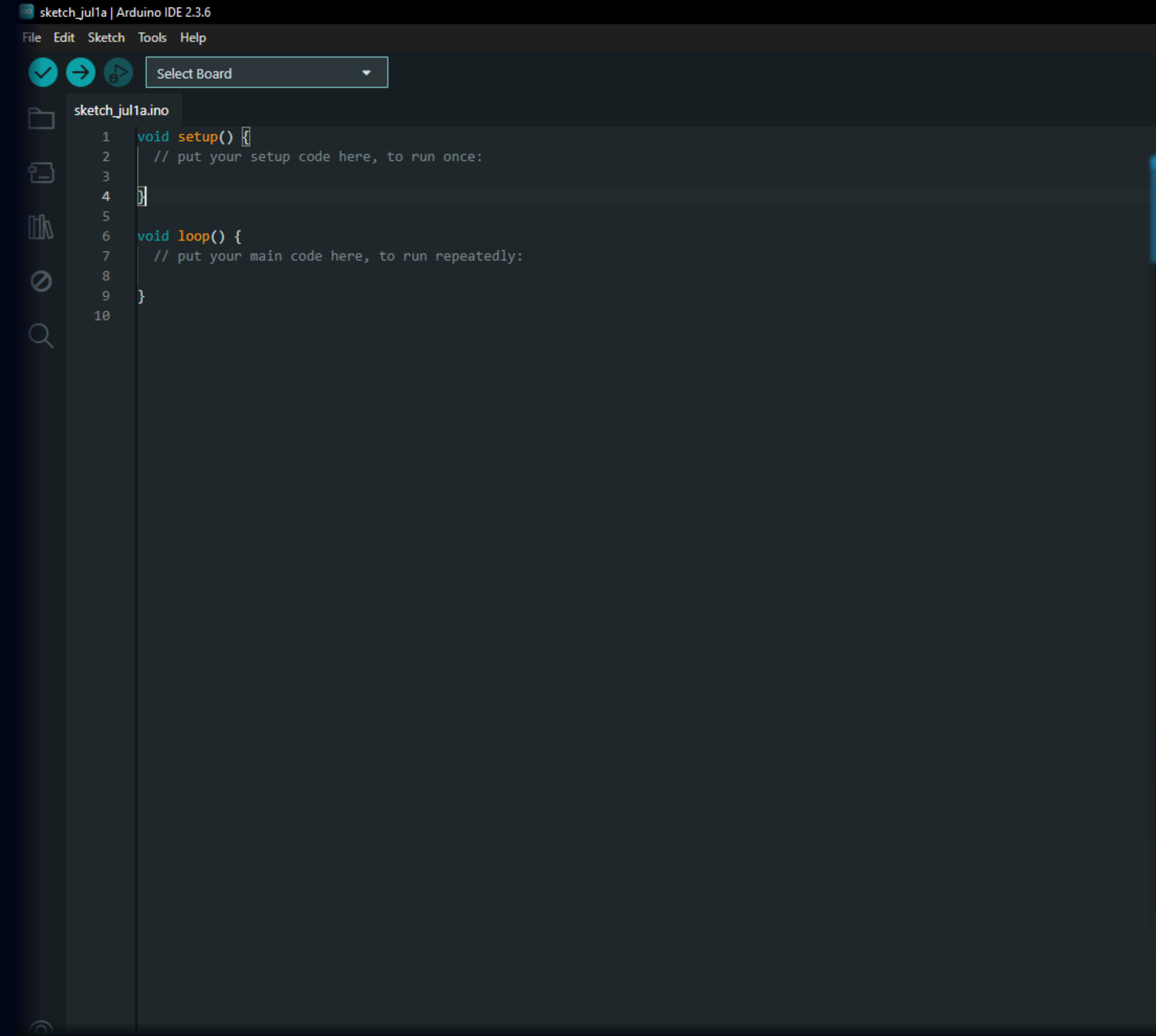


Figure 5.01 – BMP280 Pressure & Sensor Module

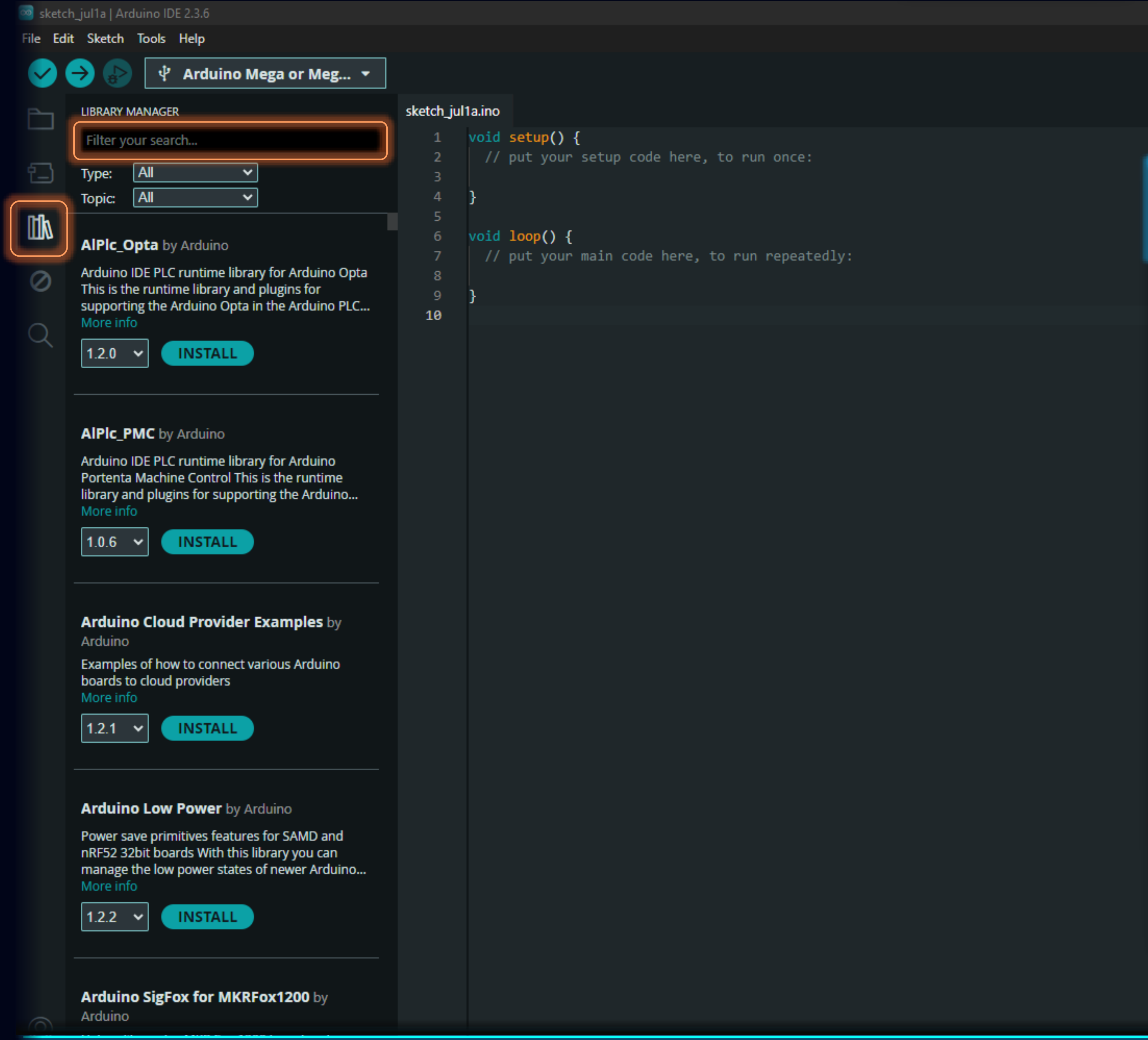


Launch Program

Locate the Arduino IDE icon on your desktop and launch the program.

Sketches

The code you write in the Arduino IDE is called a sketch, and the Arduino compiler within the program handles all the setup to convert it into machine language for the microcontroller. It utilizes a simplified subset of C++ with a few custom libraries simplifying C++ to be more accessible for prototyping and hardware interaction.



Essential Libraries

1) Click on the Libraries icon in the left-hand panel of the Arduino IDE.
2) Use the search bar to find and install the latest versions of the following library:
   - Adafruit BMP280 Library

What are Libraries?

Arduino libraries are packaged collections of functions and drivers that extend the Arduino's capabilities to work with specific hardware or features.

Be sure to select Install All when prompted about dependencies.

Arduino to PC USB Connection

Connecting the USB Type-B cable between your computer and the Arduino board establishes a bidirectional communication link, allowing your computer to upload code to the microcontroller while also enabling the Arduino to send data back to the computer.

Connect the USB Type-B cable to your computer and the Arduino board.

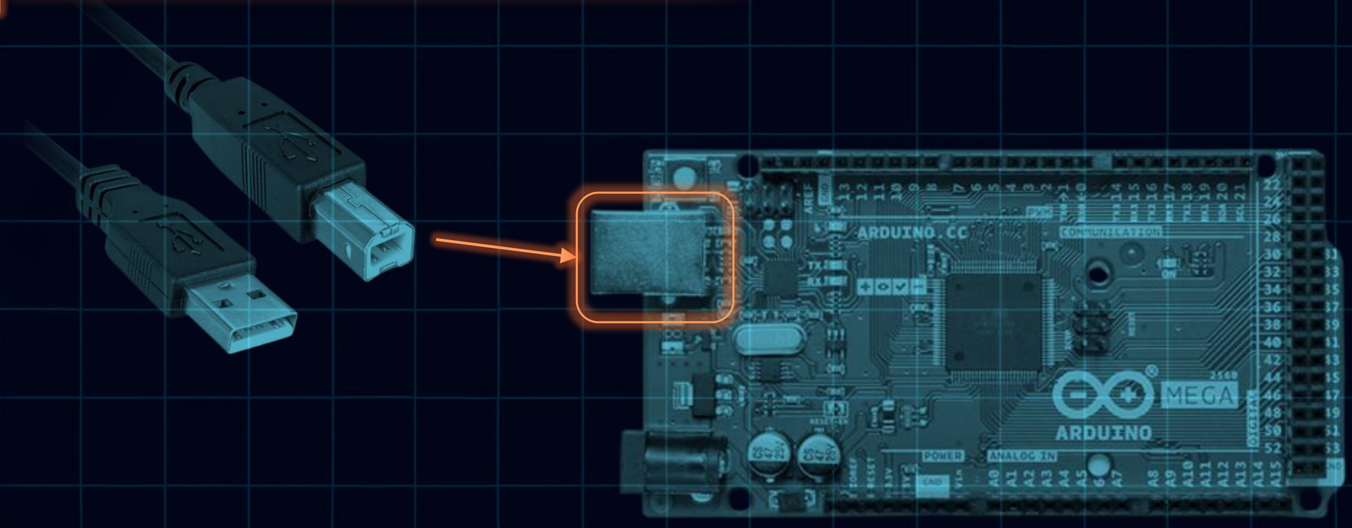


Figure 5.02 – Arduino-PC USB Connection

BMP280 Wiring Setup

The BMP280 uses SPI (Serial Peripheral Interface) to communicate with the Arduino ATMega 2560. SPI is a fast, synchronous protocol ideal for high-speed sensor data transfer. It allows the microcontroller to exchange data with the sensor using a master-slave architecture over just four data lines.

Connections

Attach the BMP280 sensor to the breadboard using the jumper wires to make the following connections.

Arduino	BMP280
3.3V	VIN
GND	GND
Pin 52	SCK
Pin 50	SDO
Pin 51	SDI
Pin 53	CS

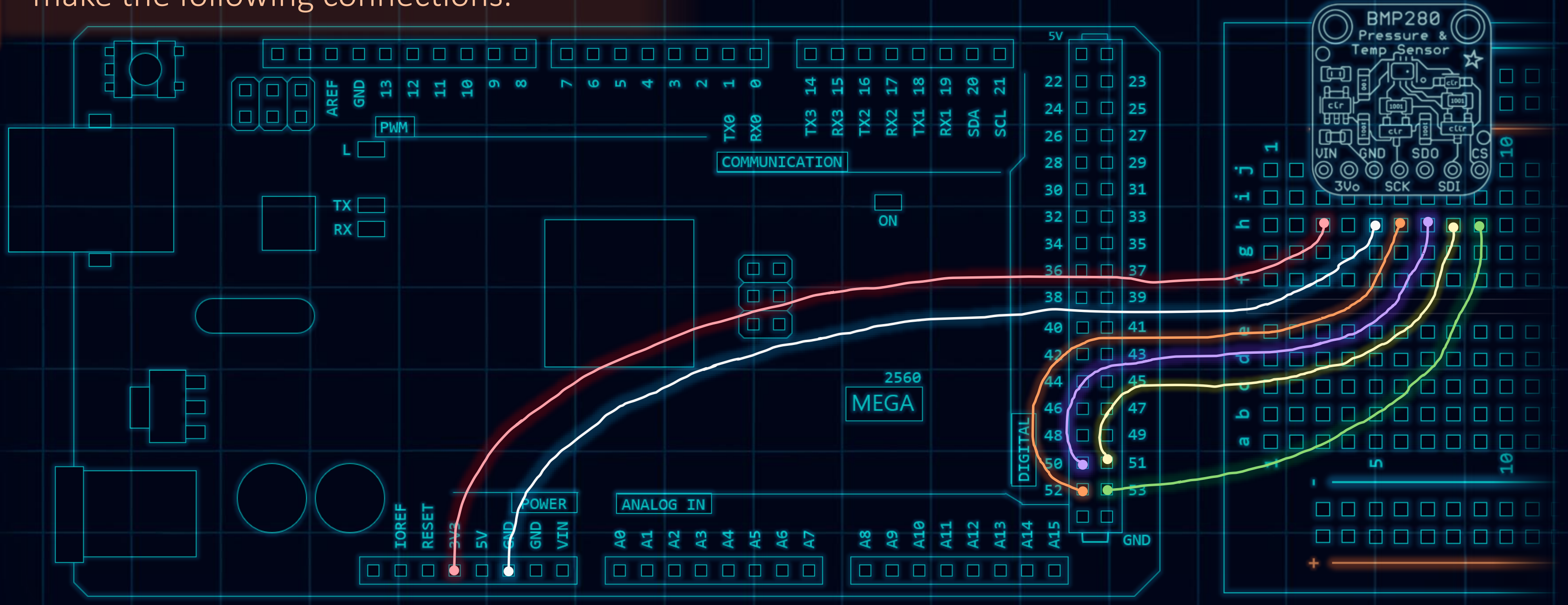


Figure 5.03 – Arduino ATMega2560 - BMP280 Sensor Wiring Setup

QUEENSBOROUGH COMMUNITY COLLEGE

BMP280 Wiring Setup

This photo shows the BMP280 connected via SPI to the Arduino Mega 2560. Verify that the sensor is powered with 3.3V and grounded properly. Take a moment to check that all four SPI lines—MISO, MOSI, SCK, and CS—are cleanly connected and not loose, as even small wiring issues can lead to failed communication or corrupted sensor readings.

Connections

Arduino	BMP280
3.3V	VIN
GND	GND
Pin 52	SCK
Pin 50	SDO
Pin 51	SDI
Pin 53	CS

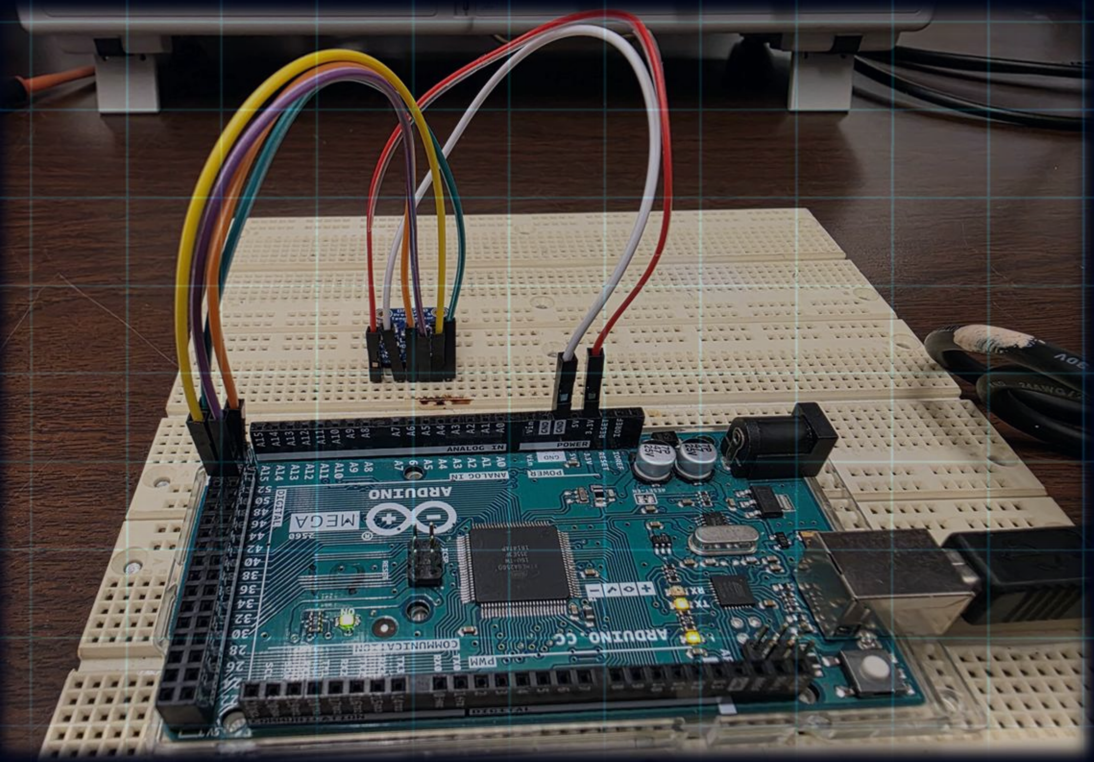


Figure 5.04 – Arduino-BMP280 Sensor Wiring Setup

BMP280 Test Sketch

Code Block

Read through the following code and try understanding the instructions being given to the Arduino.

Once done, paste the following Arduino sketch into the IDE and upload it to verify that the module is functional.

Make sure the baud rate in the Serial Monitor matches the one defined in your code. Here, it is set to 9600 as shown in:

Serial.begin(9600)

```
#include <SPI.h>
#include <Adafruit_BMP280.h>

#define BMP_CS   53
constexpr float seaLevelPressure_hPa = 1015.0f; // Barometric Pressure for Queens, NY

Adafruit_BMP280 bmp(BMP_CS); // Hardware SPI, CS only

void setup() {
  Serial.begin(9600);
  pinMode(BMP_CS, OUTPUT); // Required for SPI on Mega

  if (!bmp.begin()) {
    Serial.println(F("BMP280 not found. Check wiring."));
    while (true) delay(10);
  }

  // FORCED mode: manual one-shot measurement
  bmp.setSampling(Adafruit_BMP280::MODE_FORCED,
                  Adafruit_BMP280::SAMPLING_X1,     // Temp oversampling
                  Adafruit_BMP280::SAMPLING_X1,     // Pressure oversampling
                  Adafruit_BMP280::FILTER_OFF,      // No IIR filter
                  Adafruit_BMP280::STANDBY_MS_1);   // Not used in FORCED
}

void loop() {
  bmp.takeForcedMeasurement(); // A function created by the Adafruit BMP280 library

  float temp = bmp.readTemperature();
  float press = bmp.readPressure();
  float alt = bmp.readAltitude(seaLevelPressure_hPa);

  Serial.print(F("T: ")); Serial.print(temp); Serial.print(F(" *C | "));  // Prints temperature
  Serial.print(F("P: ")); Serial.print(press); Serial.print(F(" Pa | ")); // Prints pressure
  Serial.print(F("Alt: ")); Serial.print(alt); Serial.println(F(" m"));   // Prints approximate altitude

  delay(1000); // Optional: set based on desired logging rate (ms)
}
```

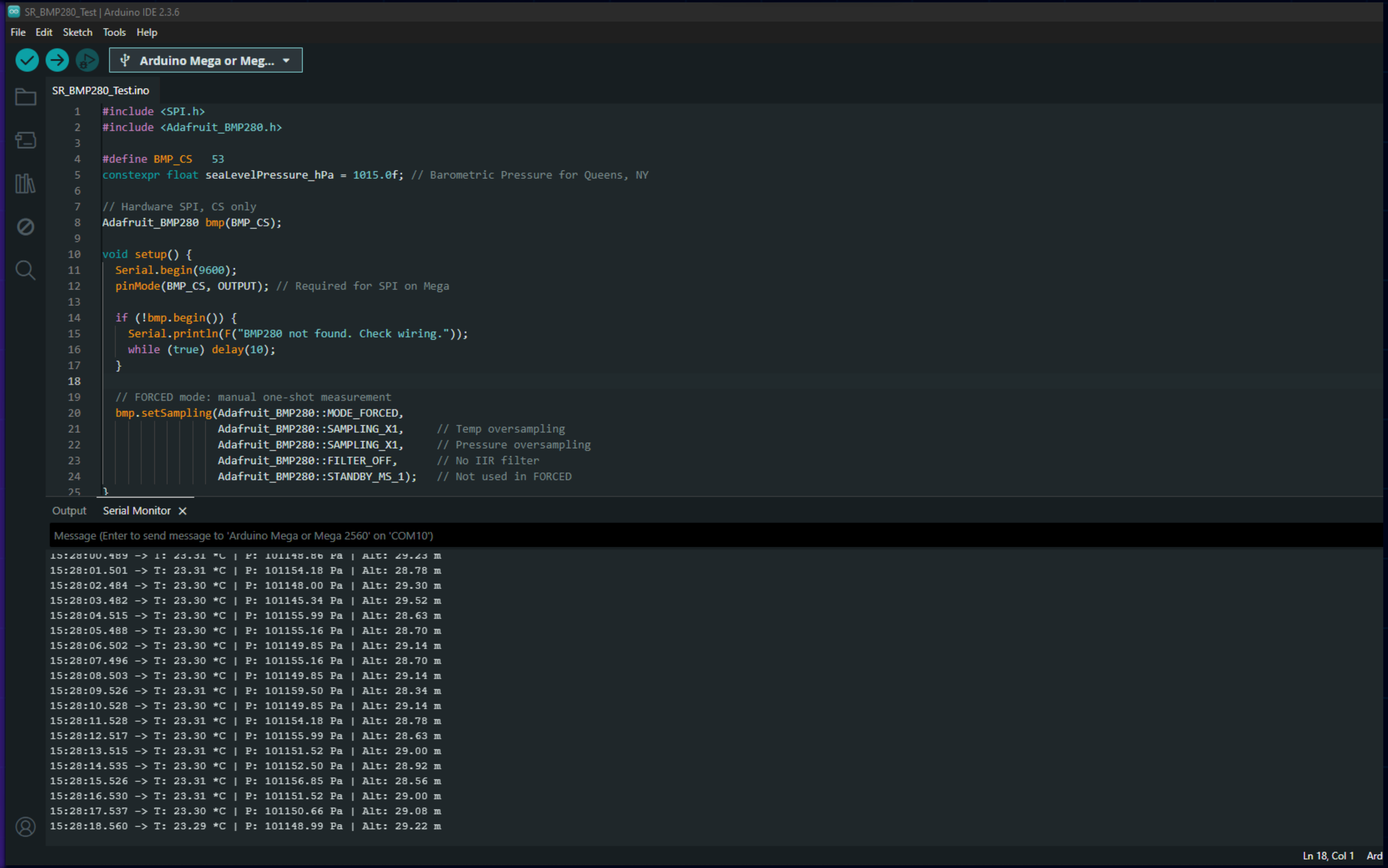
QUEENSBOROUGH COMMUNITY COLLEGE

BMP280 Test Results

Serial Output

The Serial Monitor should display temperature, pressure, and altitude readings once every second, as shown in the example.

If the readings appear consistently and update at the expected one second interval, the sensor is operating correctly.



Module V

Adafruit LED Backpack Counter

QUEENSBOROUGH COMMUNITY COLLEGE

LED Backpack Counter Operational Test

Follow the instructions to complete a functional test of the LED Backpack Counter. This will verify that the display is operating correctly, showing accurate values, and properly communicating with the system. Once the test passes, the display is ready for use in your project.

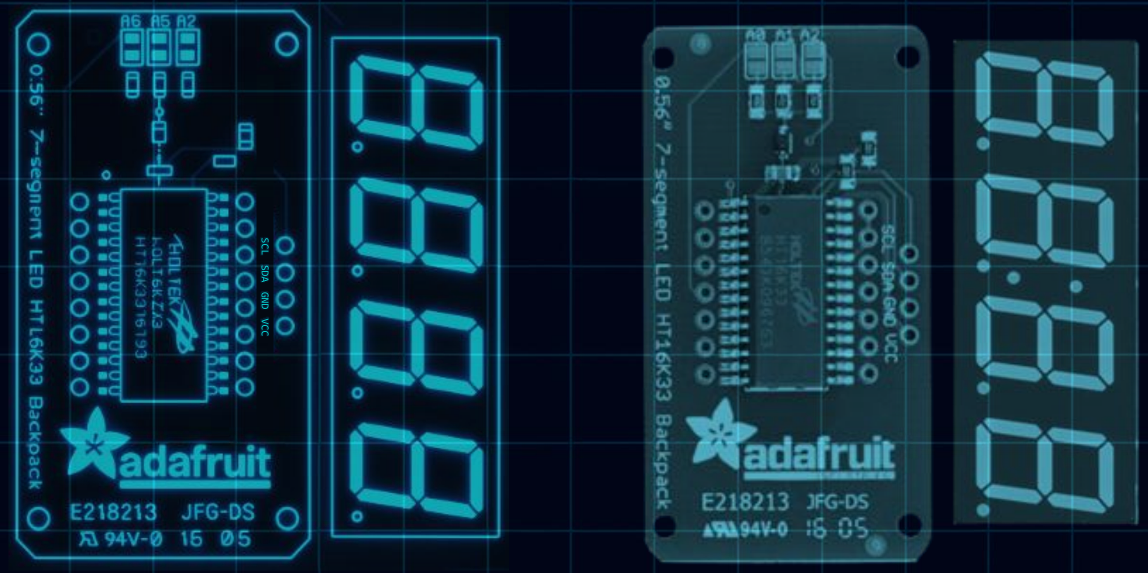
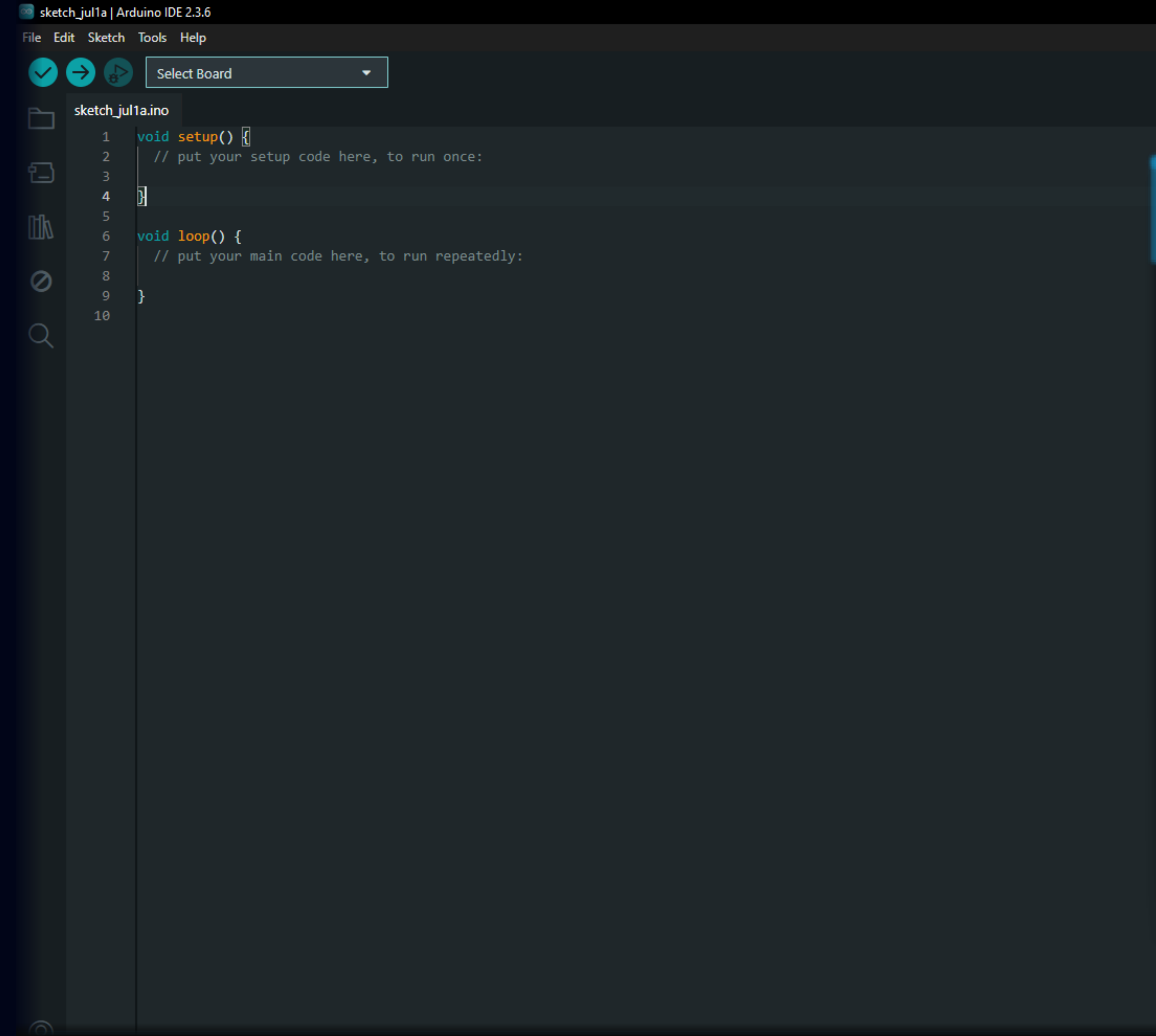


Figure 6.01 – Adafruit LED Backpack Counter

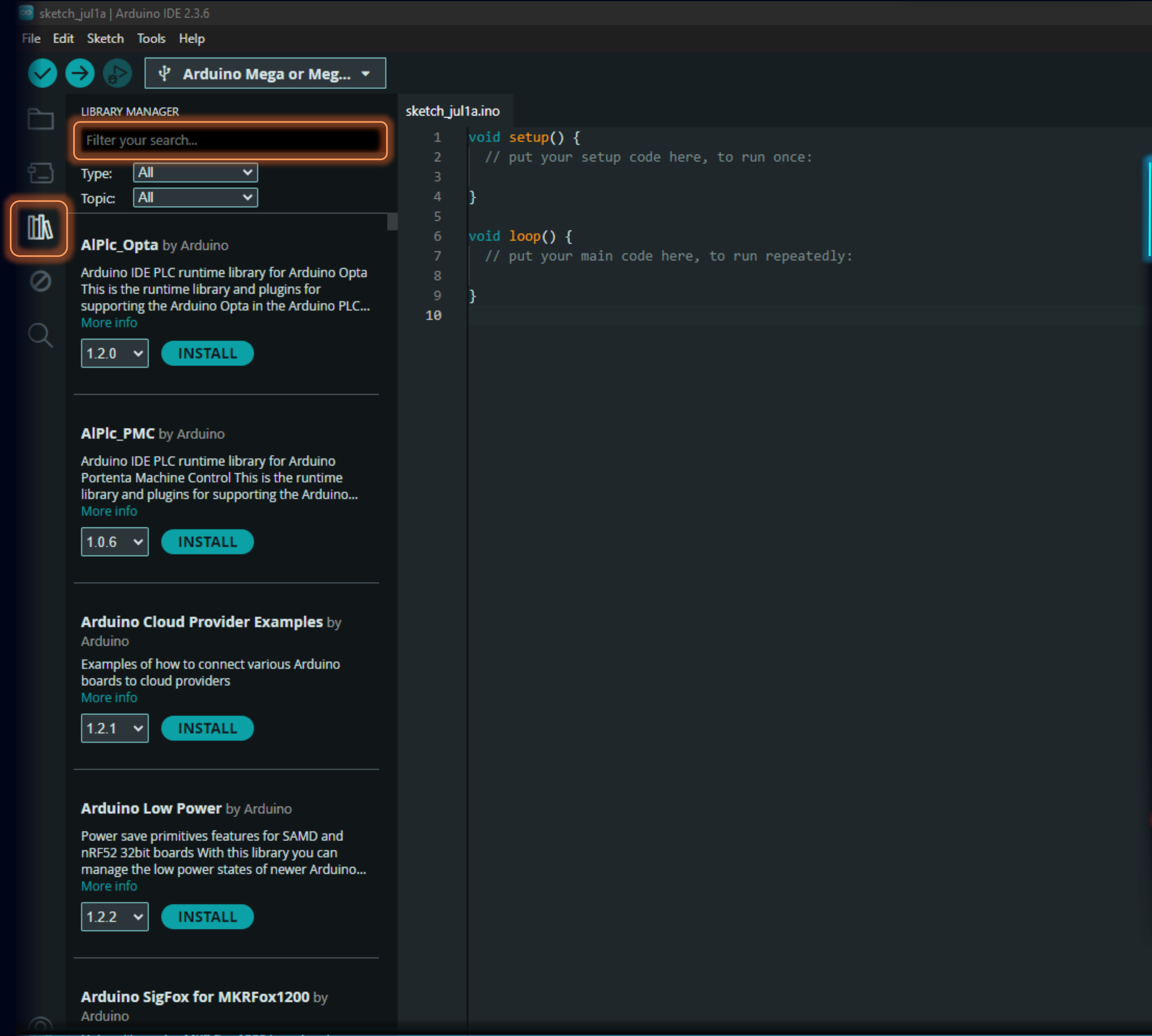


Launch Program

Locate the Arduino IDE icon on your desktop and launch the program

Sketches

The code you write in the Arduino IDE is called a sketch, and the Arduino compiler within the program handles all the setup to convert it into machine language for the microcontroller. It utilizes a simplified subset of C++ with a few custom libraries simplifying C++ to be more accessible for prototyping and hardware interaction.



Essential Libraries

1) Click on the Libraries icon in the left-hand panel of the Arduino IDE.
2) Use the search bar to find and install the latest versions of the following library:
   - Adafruit LED Backpack
   - TimerOne

What are Libraries?

Arduino libraries are packaged collections of functions and drivers that extend the Arduino's capabilities to work with specific hardware or features.

Be sure to select Install All when prompted about dependencies.

Arduino to PC USB Connection

Connecting the USB Type-B cable between your computer and the Arduino board establishes a bidirectional communication link, allowing your computer to upload code to the microcontroller while also enabling the Arduino to send data back to the computer.

Connect the USB Type-B cable to your computer and the Arduino board.

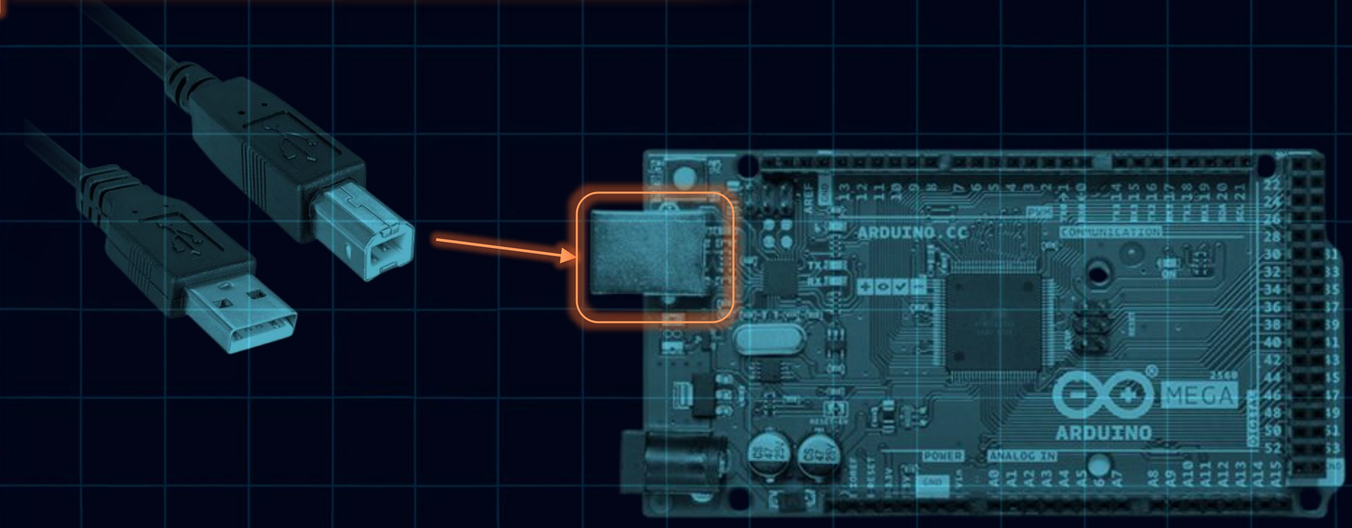


Figure 6.02 – Arduino-PC USB Connection

LED Backpack Counter Wiring Setup

The LED Backpack Counter communicates with the Arduino ATMega 2560 via I2C (Inter-Integrated Circuit). It uses SDA (data) and SCL (clock) lines, which on the Mega 2560 correspond to pins 20 and 21. This setup allows the microcontroller to send numeric data to the display efficiently, with additional connections for power (5V) and ground (GND) to complete the circuit.

Connections

Arduino	Counter
5V	+
GND	-
SCL1	C (SCL)
SDA1	D (SDA)

SCL1 and SDA1 are not labeled on the board surface like the other pins. Instead, their labels are printed directly on the black socket connector.

Attach the LED Backpack Counter to the breadboard using the jumper wires to make the following connections.

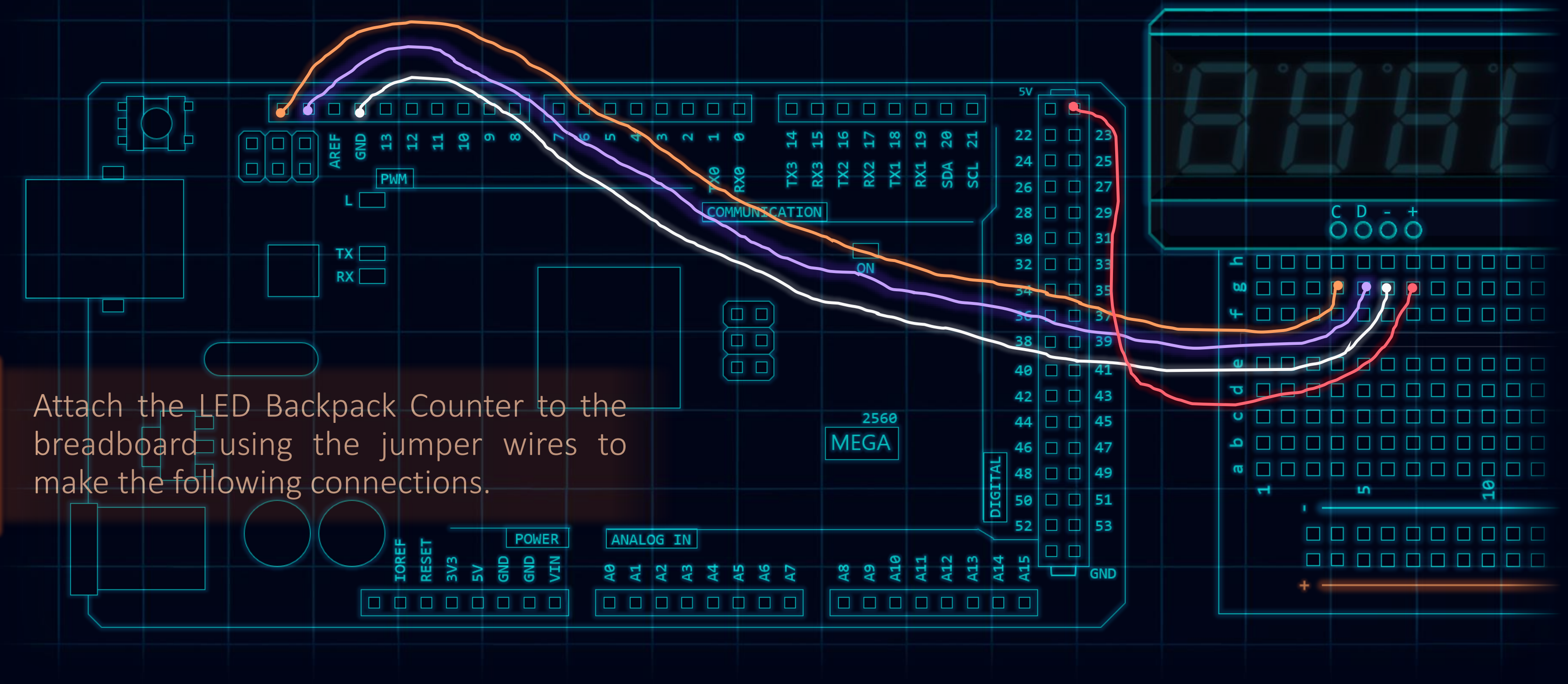


Figure 6.03 – Arduino ATMega2560 - LED Backpack Counter Wiring Setup

LED Backpack Counter Wiring Setup

This photo shows the physical wiring of the LED Backpack Counter connected to the Arduino Mega 2560. Verify that the jumper wires are correctly placed and securely connected. Also make sure the 5V and GND lines are properly seated as any loose power connection can cause the display to flicker or fail to initialize.

Connections

Arduino	Counter
5V	+
GND	-
SCL1	C (SCL)
SDA1	D (SDA)

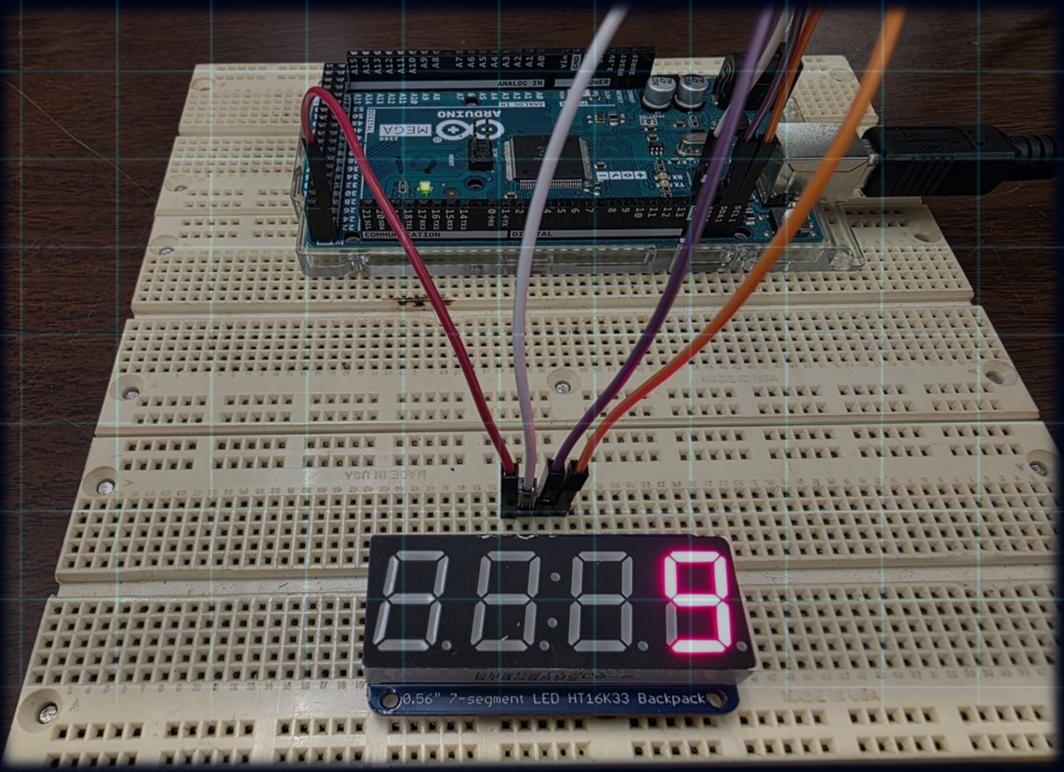


Figure 6.04 – Arduino ATMega2560 - LED Backpack Counter Wiring Setup

LED Backpack Counter Test Sketch

Code Block

Read through the following code and try to understand the instructions being given to the Arduino.

Once done, paste the following Arduino sketch into the IDE and upload it to verify that the module is functional.

Make sure the baud rate in the Serial Monitor matches the one defined in your code. Here, it is set to 9600 as shown in:

Serial.begin(9600)

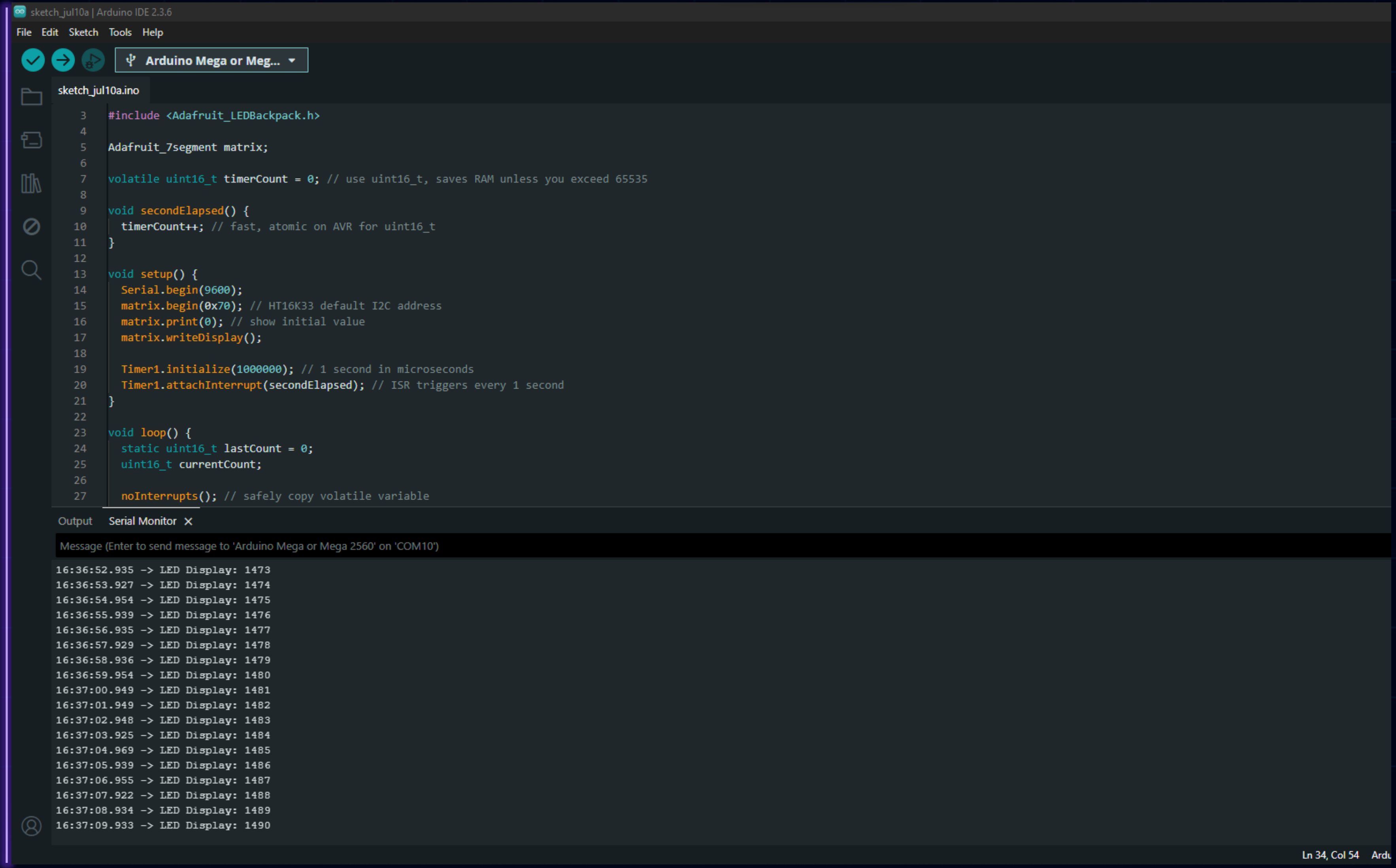
```
#include <Wire.h>
#include <TimerOne.h>
#include <Adafruit_LEDBackpack.h>

Adafruit_7segment matrix;

volatile uint16_t timerCount = 0;  // use uint16_t, saves RAM unless you exceed 65535

void secondElapsed() {
  timerCount++;  // fast, atomic on AVR for uint16_t
}

void setup() {
  Serial.begin(9600);
  matrix.begin(0x70);  // HT16K33 default I2C address
  matrix.print(0);     // show initial value
  matrix.writeDisplay();

  Timer1.initialize(1000000);                 // 1 second in microseconds
  Timer1.attachInterrupt(secondElapsed);      // ISR triggers every 1 second
}

void loop() {
  static uint16_t lastCount = 0;
  uint16_t currentCount;

  noInterrupts();         // safely copy volatile variable
  currentCount = timerCount;
  interrupts();

  if (currentCount != lastCount) {
    lastCount = currentCount;
    matrix.print(currentCount);
    matrix.writeDisplay();  // update only when needed
    Serial.print(F("LED Display: "));
    Serial.print(currentCount);
  }
}
```

LED Backpack Counter Test Results

Serial Output

The LED display should show a count that increases by 1 every second, and the Serial Monitor should display the same count shown on the LED Backpack Counter.

If both displays are synchronized and updating as expected, the system is functioning correctly and ready for use.



Module VI

Adafruit Ultimate GPS Breakout V3

QUEENSBOROUGH
COMMUNITY COLLEGE

Ultimate GPS Breakout V3 Operational Test

Follow the instructions to complete a functional test of the GPS Module. This will verify that the GPS is acquiring a signal, providing accurate location data, and communicating properly with the system. Once the test passes, the module is ready for integration into your project.

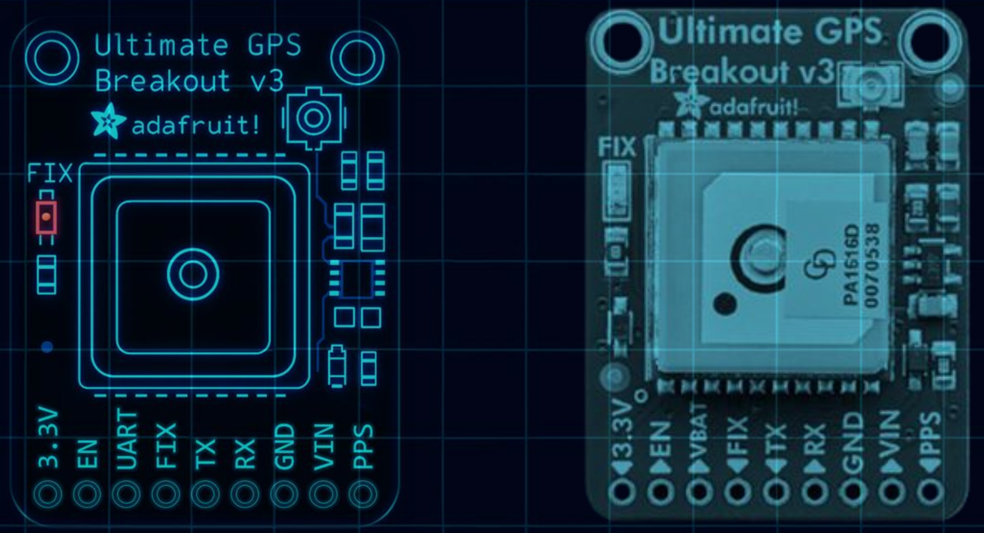


Figure 7.02 – Adafruit Ultimate GPS Breakout V3

Ultimate GPS Breakout V3 Satellite Fix Indicator

The FIX LED indicates the GPS module's lock status. When the module is searching for satellites, the LED blinks rapidly (about once per second). Once it obtains a valid position fix—using signals from at least 3 satellites—the LED slows down and begins blinking once every 15 seconds. This change in blink rate lets you know that the GPS has a reliable location fix and is actively tracking satellites.

> The FIX LED will flash once every 15 seconds after acquiring your position.

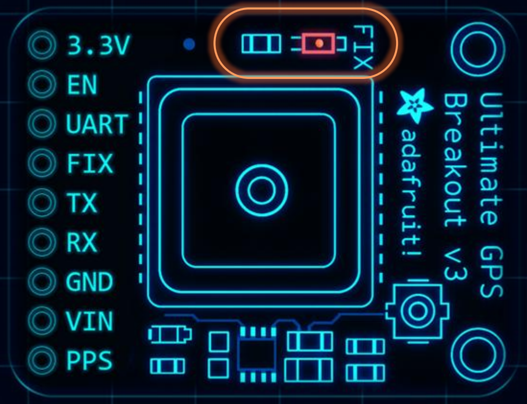


Figure 7.01 – Adafruit Ultimate GPS Breakout V3 Data Output Pins

Ultimate GPS Breakout V3 Output

The GPS module outputs data using the NMEA (National Marine Electronics Association) standard—readable ASCII text sentences sent over a serial connection. These sentences include key information such as time, latitude, longitude, altitude, speed, fix status, and satellite count.

In addition to serial data, the module features a PPS (Pulse Per Second) pin, which outputs a precise 1 Hz pulse aligned with the start of each second. This pulse is highly accurate and is used for time synchronization in applications requiring precise timing.

GPS data is sent through the TX pin using the NMEA format, while the PPS pin outputs a 1Hz pulse (one pulse per second) for accurate time sync.

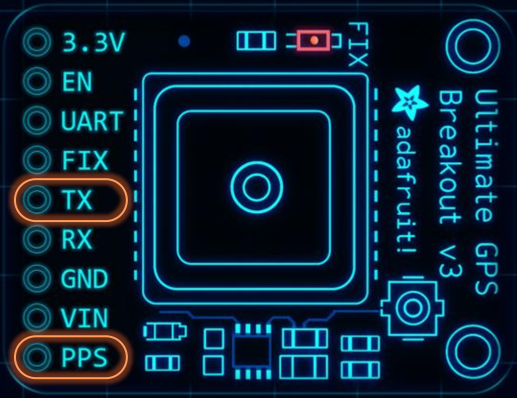


Figure 7.01 – Adafruit Ultimate GPS Breakout V3 Data Output Pins

QUEENSBOROUGH COMMUNITY COLLEGE

NMEA Format Sentence Types

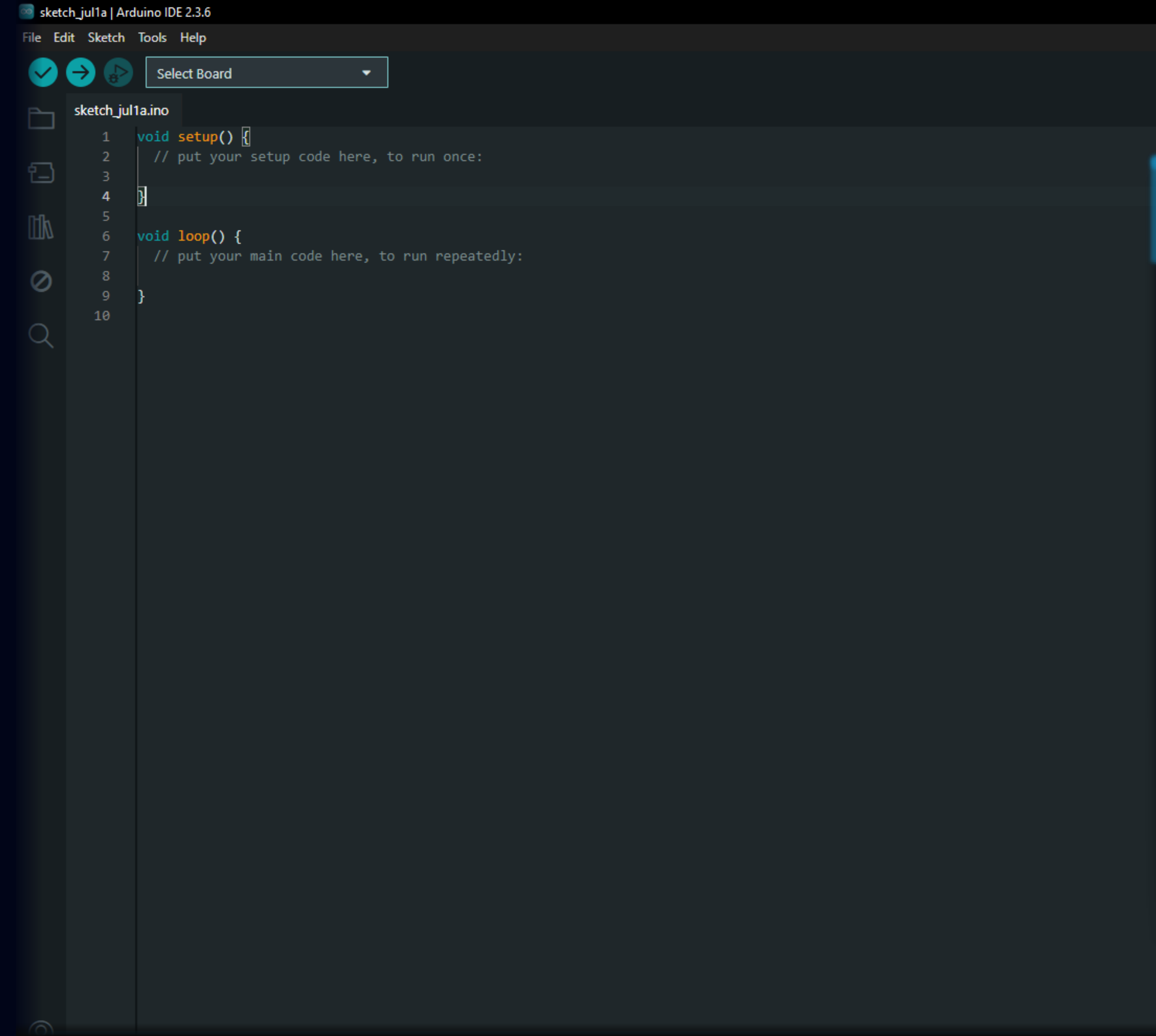
The GPS module outputs data in standard NMEA sentences, each beginning with a $ symbol and carrying specific types of information. Sentence types include:

- $GPGGA – Provides essential fix data including time, latitude, longitude, fix status, number of satellites, and altitude.
- $GPGSA – Lists which satellites are used in the fix and provides Dilution of Precision (DOP) values that indicate GPS accuracy.
- $GPGSV – Describes the satellites currently in view, including their elevation, azimuth, and signal strength.
- $GPRMC – Offers the minimum recommended navigation data: time, date, fix validity, speed over ground, and course over ground.
- $GPVTG – Reports the actual ground track angle and speed in both knots and kilometers per hour.

Each sentence is updated once per second by default and can be selectively enabled or disabled for performance tuning or bandwidth reduction.

Sentence Types	Data Type
GGA	Global Positioning System Fixed Data
GSA	GNSS DOP and Active Satellites
GSV	GNSS Satellites in View
RMC	Recommended Minimum Specific Data
VTG	Course Over Ground and Ground Speed

Refer to the appendix for full sentence breakdowns.

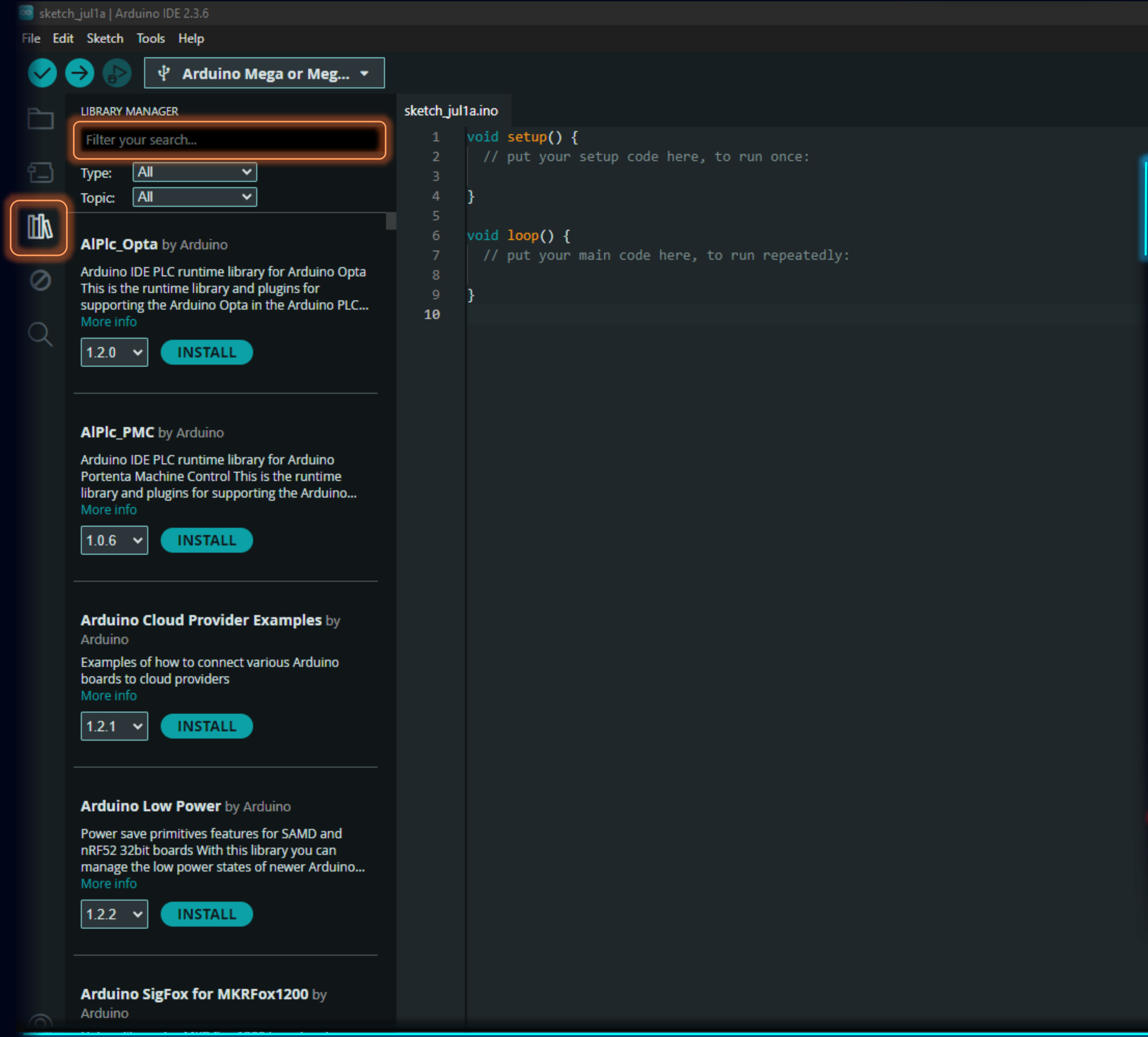


Launch Program

> Locate the Arduino IDE icon on your desktop and launch the program

Sketches

The code you write in the Arduino IDE is called a sketch, and the Arduino compiler within the program handles all the setup to convert it into machine language for the microcontroller. It utilizes a simplified subset of C++ with a few custom libraries simplifying C++ to be more accessible for prototyping and hardware interaction.



Essential Libraries

1) Click on the Libraries icon in the left-hand panel of the Arduino IDE.
2) Use the search bar to find and install the latest versions of the following library:
    - Adafruit GPS Library

What are Libraries?

Arduino libraries are packaged collections of functions and drivers that extend the Arduino's capabilities to work with specific hardware or features.

Be sure to select Install All when prompted about dependencies.

Arduino to PC USB Connection

Connecting the USB Type-B cable between your computer and the Arduino board establishes a bidirectional communication link, allowing your computer to upload code to the microcontroller while also enabling the Arduino to send data back to the computer.

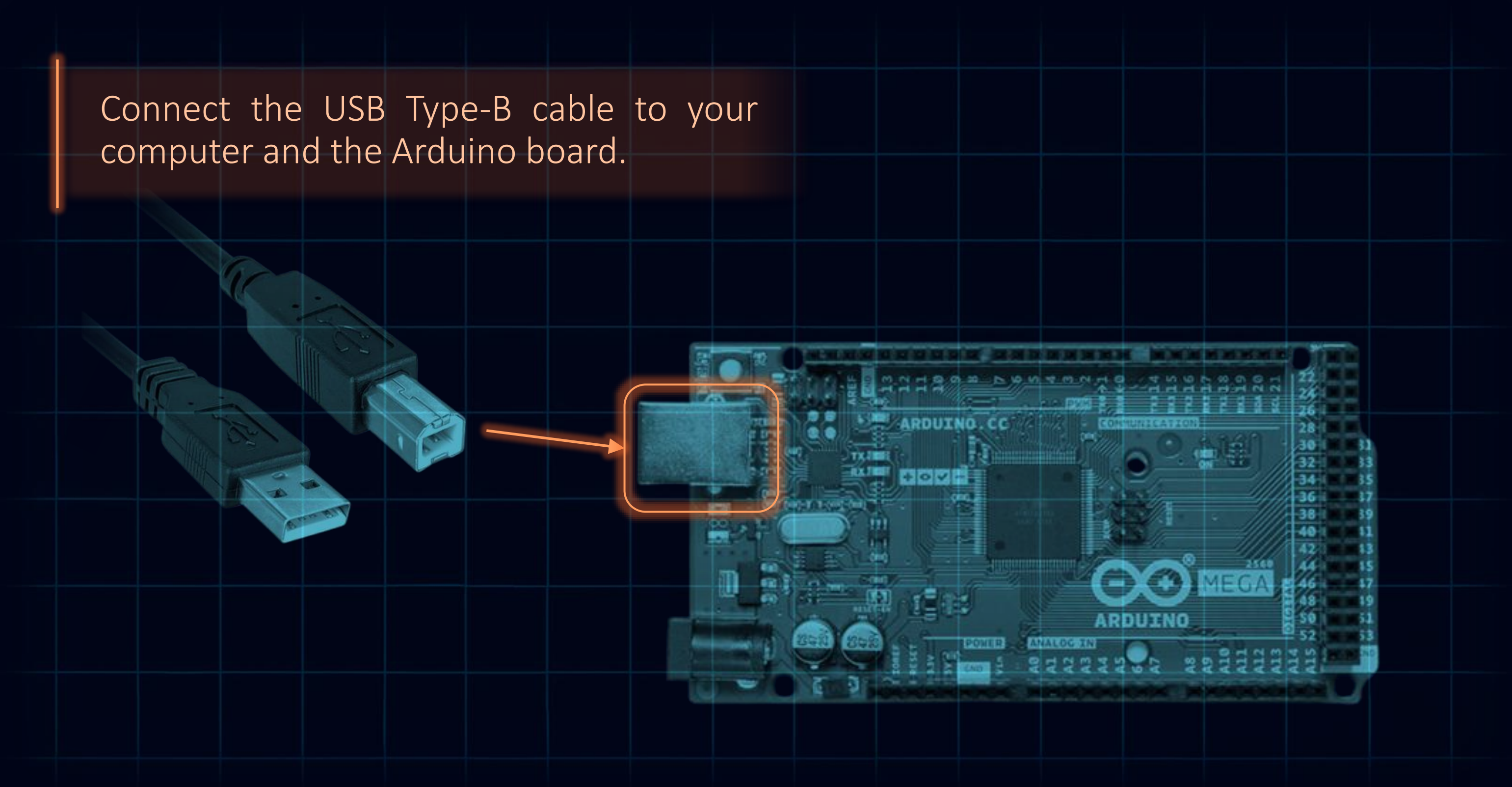


Figure 7.03 – Arduino-PC USB Connection

Adafruit Ultimate GPS Breakout V3 - PPS Wiring

The GPS module provides a PPS (Pulse Per Second) output, which emits a square pulse precisely once per second. This pulse is synchronized to GPS satellite time, which is maintained by atomic clocks onboard the satellites. When connected to the Arduino ATMega 2560, the PPS pin provides a highly accurate time reference.

Connections

Arduino	GPS
GND	GND
5V	VIN
Pin 48	PPS

The cable plugged into the socket at the top right of the GPS module is the satellite antenna. It must be connected for the module to receive satellite signals and provide GPS data.

Attach the GPS module to the breadboard using the jumper wires to make the following connections.

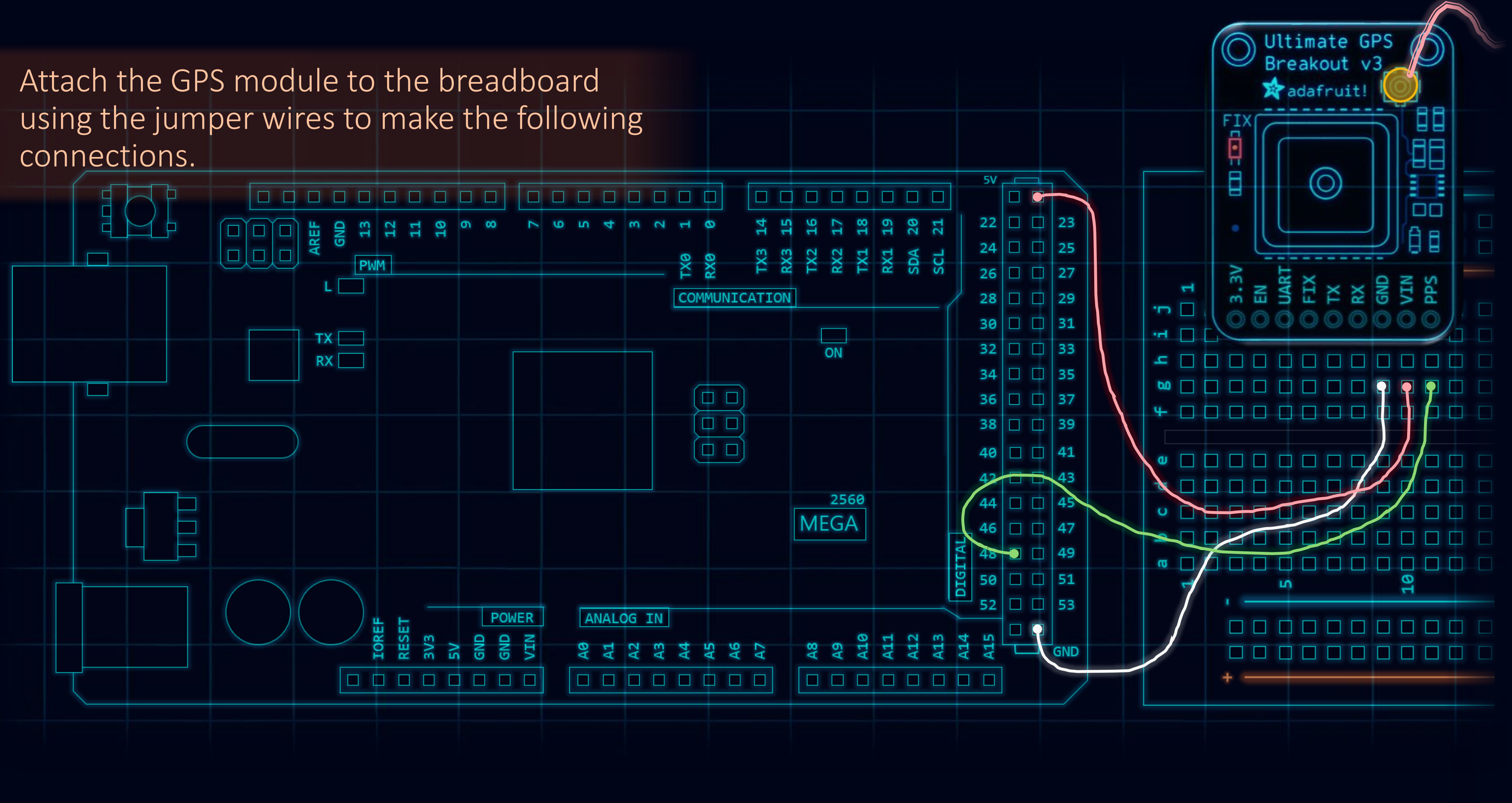


Figure 7.04 – Arduino ATMega2560 - Adafruit Ultimate GPS Breakout V3 - PPS Wiring Setup

Adafruit Ultimate GPS Breakout V3 - PPS Wiring

This photo shows the PPS wiring connected to the Arduino Mega 2560. The PPS output from the GPS module is wired directly to digital pin 48, which serves as the input capture pin for Timer 5. Verify that power and ground are securely connected, as unstable power can disrupt signal accuracy. Lastly, ensure the GPS antenna is properly attached, since the PPS signal is only valid when the module has a satellite fix.

Connections

Arduino	GPS
GND	GND
5V	VIN
Pin 48	PPS

When you power on the GPS module, the FIX LED will start blinking. This indicates that it is searching for satellites. Once a position fix is acquired, the module will begin transmitting NMEA data.

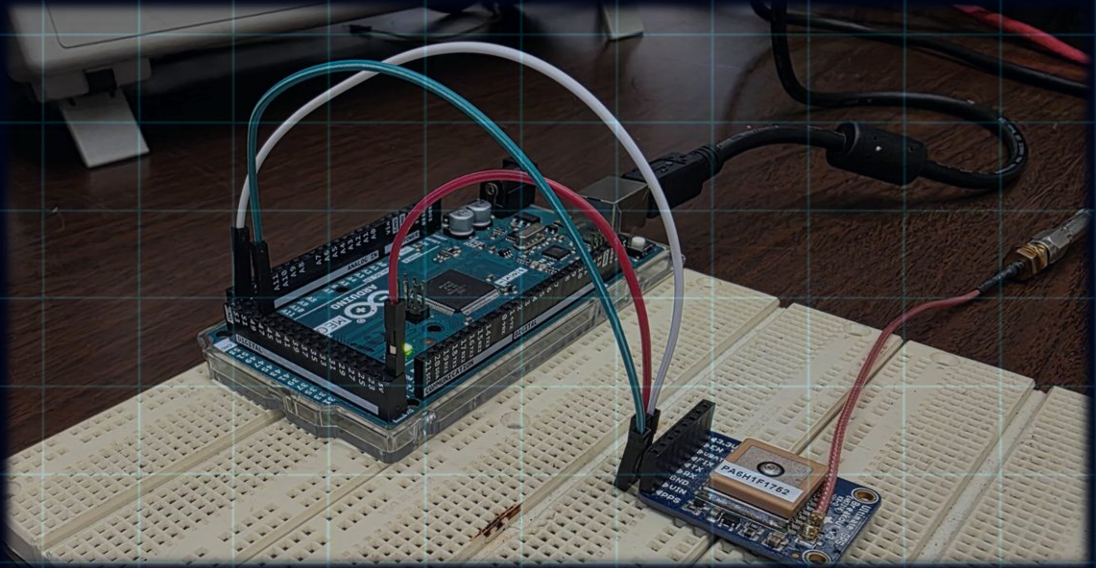


Figure 7.05 – Arduino ATMega2560 - Adafruit Ultimate GPS Breakout V3 Wiring Setup

Adafruit Ultimate GPS Breakout V3 - PPS Test Sketch

Code Block

Read through the following code and try to understand the instructions being given to the Arduino.

Once done, paste the following Arduino sketch into the IDE and upload it to verify that the module is functional.

Make sure the baud rate in the Serial Monitor matches the one defined in your code. Here, it is set to 115200 as shown in:

Serial.begin(115200)

```
#define PPS_PIN 48 // ICP5 on ATmega2560

volatile bool ppsDetected = false;

// Interrupt Service Routine for Timer 5 Input Capture
ISR(TIMER5_CAPT_vect) {
ppsDetected = true;
}

void setup() {
  Serial.begin(115200);
  pinMode(PPS_PIN, INPUT);

// Configure Timer 5 for Input Capture on rising edge, no prescaler
  TCCR5A = 0;
  TCCR5B = _BV(ICES5) | _BV(CS50); // ICES5: rising edge, CS50: no prescaler
  TIMSK5 = _BV(ICIE5); // Enable Timer 5 Input Capture interrupt
}

void loop() {
  if (__builtin_expect(ppsDetected, 0)) {
    ppsDetected = false;
    Serial.println("PPS signal detected on pin 48!");
  }
}
```

QUEENSBOROUGH COMMUNITY COLLEGE

Adafruit Ultimate GPS Breakout V3 - PPS Test Results

Serial Output

The Serial Monitor should display a message each time the GPS module sends a PPS pulse, confirming that a rising edge was successfully detected on pin 48.

If the messages appear once per second, aligning with the GPS's 1 Hz pulse, the PPS signal is being received and processed correctly.

sketch_jul15a | Arduino IDE 2.3.6

File Edit Sketch Tools Help

Arduino Mega or Meg...

sketch_jul15a.ino

```
#define PPS_PIN 48 // ICP5 on ATmega2560

volatile bool ppsDetected = false;

// Interrupt Service Routine for Timer 5 Input Capture
ISR(TIMER5_CAPT_vect) {
ppsDetected = true;
}

void setup() {
  Serial.begin(115200);
  pinMode(PPS_PIN, INPUT);

// Configure Timer 5 for Input Capture on rising edge, no prescaler
  TCCR5A = 0;
  TCCR5B = _BV(ICES5) | _BV(CS50); // ICES5: rising edge, CS50: no prescaler
  TIMSK5 = _BV(ICIE5); // Enable Timer 5 Input Capture interrupt
}

void loop() {
  if (__builtin_expect(ppsDetected, 0)) {
    ppsDetected = false;
    Serial.println("PPS signal detected on pin 48!");
  }
}
```

Output Serial Monitor ×

Message (Enter to send message to 'Arduino Mega or Mega 2560' on 'COM10')

```
13:34:29.628 -> PPS pulse detected on pin 48!
13:34:30.614 -> PPS pulse detected on pin 48!
13:34:31.615 -> PPS pulse detected on pin 48!
13:34:32.627 -> PPS pulse detected on pin 48!
13:34:33.638 -> PPS pulse detected on pin 48!
13:34:34.636 -> PPS pulse detected on pin 48!
13:34:35.602 -> PPS pulse detected on pin 48!
13:34:36.641 -> PPS pulse detected on pin 48!
13:34:37.601 -> PPS pulse detected on pin 48!
13:34:38.626 -> PPS pulse detected on pin 48!
13:34:39.603 -> PPS pulse detected on pin 48!
13:34:40.599 -> PPS pulse detected on pin 48!
13:34:41.619 -> PPS pulse detected on pin 48!
13:34:42.617 -> PPS pulse detected on pin 48!
13:34:43.617 -> PPS pulse detected on pin 48!
13:34:44.636 -> PPS pulse detected on pin 48!
13:34:45.601 -> PPS pulse detected on pin 48!
```

Ln 17, Col 65 Ardu

Adafruit Ultimate GPS Breakout V3 - NMEA Wiring

The GPS module connects to the Arduino Mega 2560 through a UART (Universal Asynchronous Receiver-Transmitter) interface. It uses the TX (transmit) and RX (receive) lines to send and receive serial data, allowing the microcontroller to receive NMEA sentences from the GPS in real time.

Connections

Arduino	GPS
Pin 19 (RX1)	TX
Pin 18 (TX1)	RX
GND	GND
5V	VIN

The cable plugged into the socket at the top right of the GPS module is the satellite antenna. It must be connected for the module to receive satellite signals and provide GPS data.

Attach the GPS module to the breadboard using the jumper wires to make the following connections.

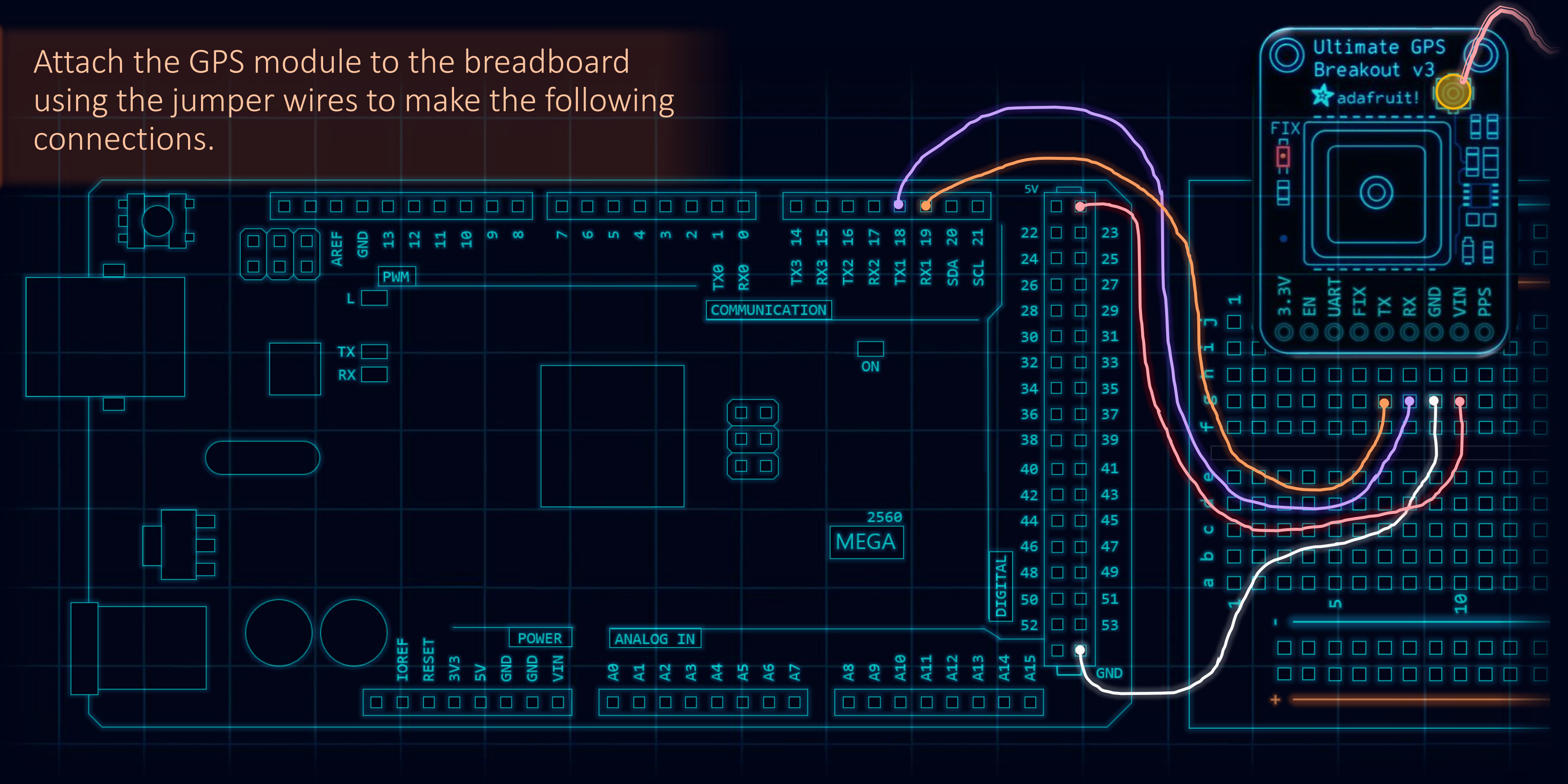


Figure 7.06 – Arduino ATMega2560 - Adafruit Ultimate GPS Breakout V3 Wiring Setup

Adafruit Ultimate GPS Breakout V3 - NMEA Wiring

This photo illustrates the wiring based on the connection chart. Ensure that all jumper wires follow the correct orientation and are firmly seated. Pay special attention to the TX/RX crossover and the power/ground lines, as improper connections can prevent the GPS from sending data or powering up.

Connections

Arduino	GPS
Pin 19 (RX1)	TX
Pin 18 (TX1)	RX
GND	GND
5V	VIN

When you power on the GPS module, the FIX LED will start blinking. This indicates that it is searching for satellites. Once a position fix is acquired, the module will begin transmitting NMEA data.

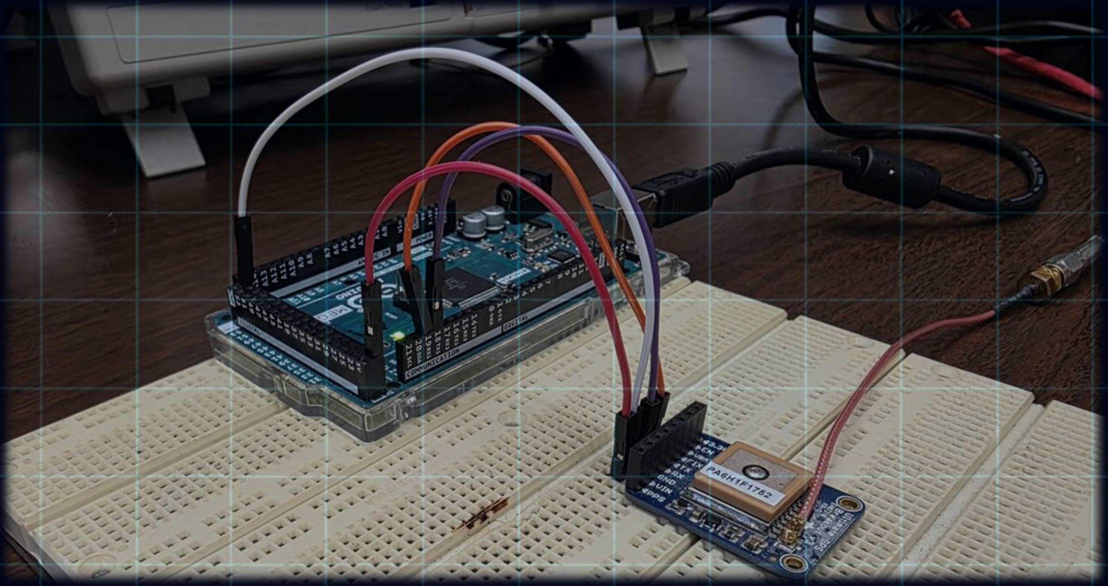


Figure 7.07 – Arduino ATMega2560 - Adafruit Ultimate GPS Breakout V3 Wiring Setup

Adafruit Ultimate GPS Breakout V3 - NMEA Test Sketch

Code Block

Read through the following code and try to understand the instructions being given to the Arduino.

Once done, paste the following Arduino sketch into the IDE and upload it to verify that the module is functional.

Make sure the baud rate in the Serial Monitor matches the one defined in your code. Here, it is set to 9600 as shown in:

Serial.begin(9600)

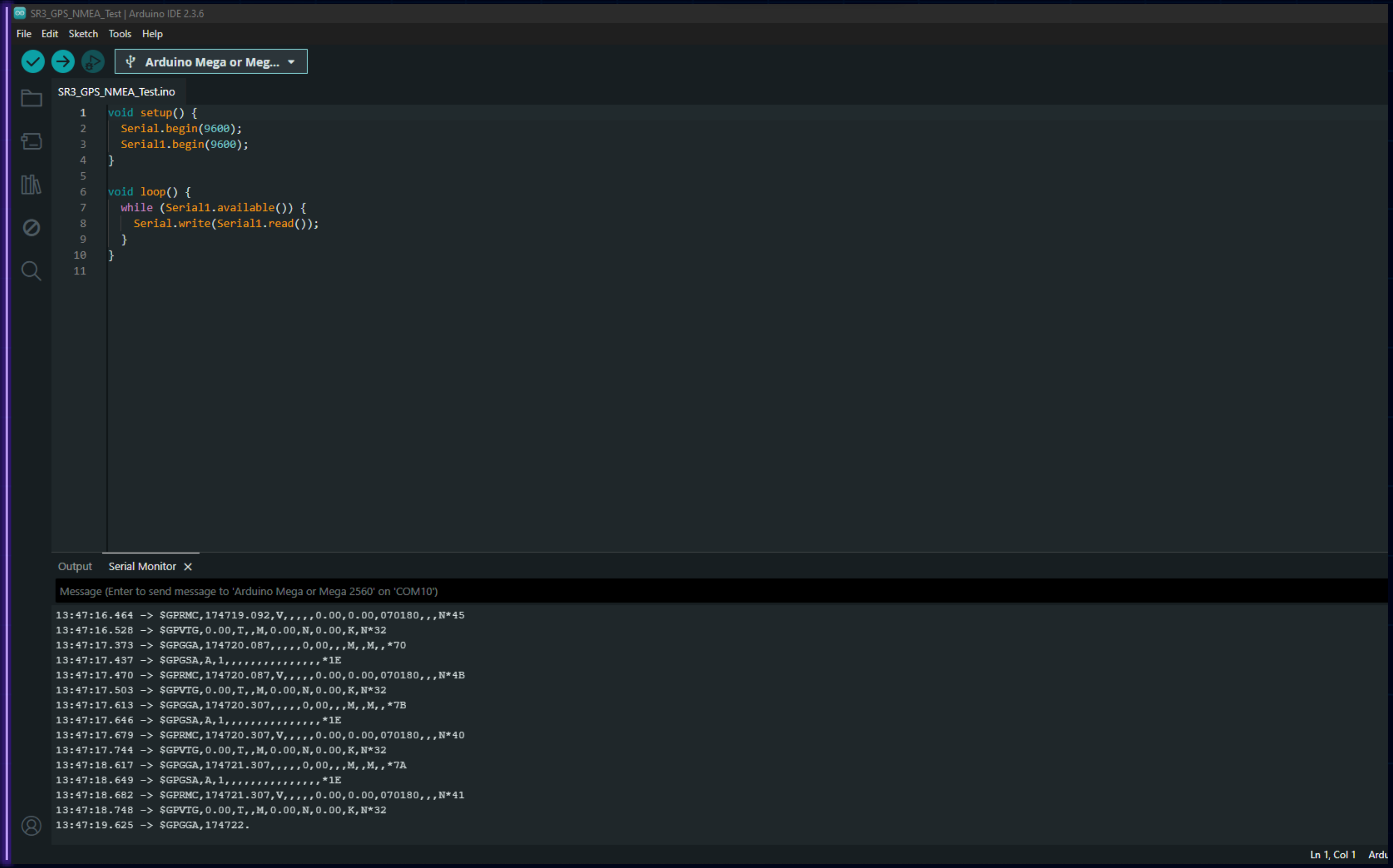
```
void setup() {
  Serial.begin(9600);
  Serial1.begin(9600);
}

void loop() {
  while (Serial1.available()) {
    Serial.write(Serial1.read());
  }
}
```

Adafruit Ultimate GPS Breakout V3 - NMEA Test Results

Serial Output

The Serial Monitor should display NMEA sentences from the GPS receiver, like the example shown here.

If NMEA sentences appear continuously and update once per second, the GPS module is communicating properly with the microcontroller and is ready for use in your application.



Module VII

ATMega2560 – Frequency Characterization

QUEENSBOROUGH COMMUNITY COLLEGE

ATMega2560 – Ceramic Resonators

Ceramic resonators are electronic components that generate stable oscillating signals for clocking microcontrollers and digital systems. They use a piezoelectric ceramic element to create mechanical vibrations at a specific frequency when voltage is applied. These vibrations are converted into an electrical signal that serves as a system clock. Ceramic resonators are valued for their small size, fast startup time, and internal capacitor integration, which simplifies circuit design.

Rated Frequency

The ATmega2560 microcontroller is designed to operate at a rated clock frequency of 16MHz when powered at 5V.

This defines how many cycles the CPU completes each second — 16 million, to be exact — which directly influences how fast it can process instructions and handle time-critical tasks.

Think of the ceramic resonator as the microcontroller's heart — beating 16 million times per second to drive every operation.

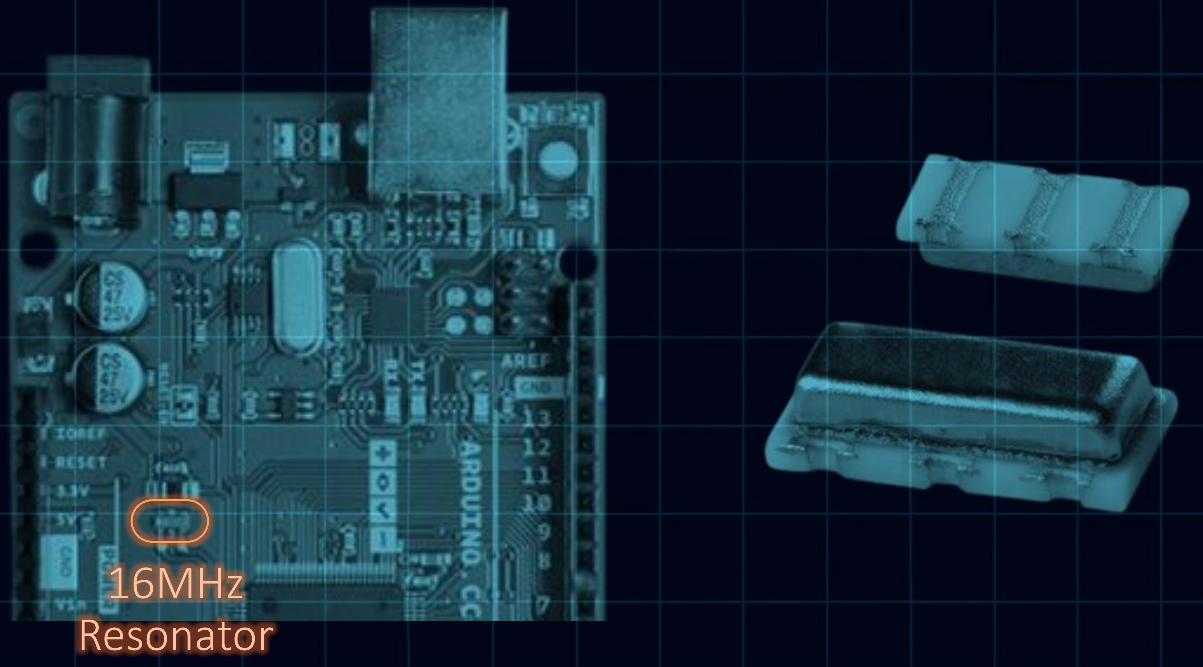


Figure 8.01 – Ceramic Resonators

ATMega2560 - Clock Cycles

Rated at 16MHz, the ATMega2560 completes 16 million cycles per second, and with each cycle, it can perform part of an instruction or complete simple ones entirely. The rated frequency of 16 MHz directly defines how quickly the microcontroller processes instructions, communicates with peripherals, or toggles pins. Faster ratings mean faster execution, making the clock an essential measure of the microcontroller's speed and precision.

Clock Period

The clock period is the amount of time it takes for one complete cycle of the microcontroller's clock signal.

It is directly derived from the clock frequency (16MHz) which defines how many cycles occur each second. To calculate the clock period, you simply take the inverse of the frequency.

A clock rated at 16MHz will take 62.5 nanoseconds (ns) to complete one cycle.

16MHz Ceramic Resonator

(Clock Period Calculation)

$$\frac{\text{1 second}}{\text{16,000,000 cycles}} = \text{62.5 nanoseconds per cycle}$$

Figure 8.02 – Clock Period Calculation

ATMega2560 - Oscillator Drift

Oscillator drift refers to small, gradual changes in the frequency output of a ceramic resonator over time. While ceramic resonators are compact and convenient for clock generation, they are more sensitive to external influences. Factors such as temperature fluctuations, supply voltage variations, mechanical vibration, and material aging can cause the output frequency to deviate from its rated value. For instance, a ceramic resonator rated at 16MHz might oscillate at 16,000,100 Hz or 15,999,800 Hz depending on its environment. Though the drift is typically small (measured in parts per million, or ppm), it can affect long-term timing accuracy in systems that depend on precise intervals.

External Influences

- Supply voltage fluctuations
- Temperature changes
- Aging of the material
- Mechanical stress or vibration

Knowing the actual frequency — accounting for oscillator drift — is essential before using the Arduino in precise timing applications.

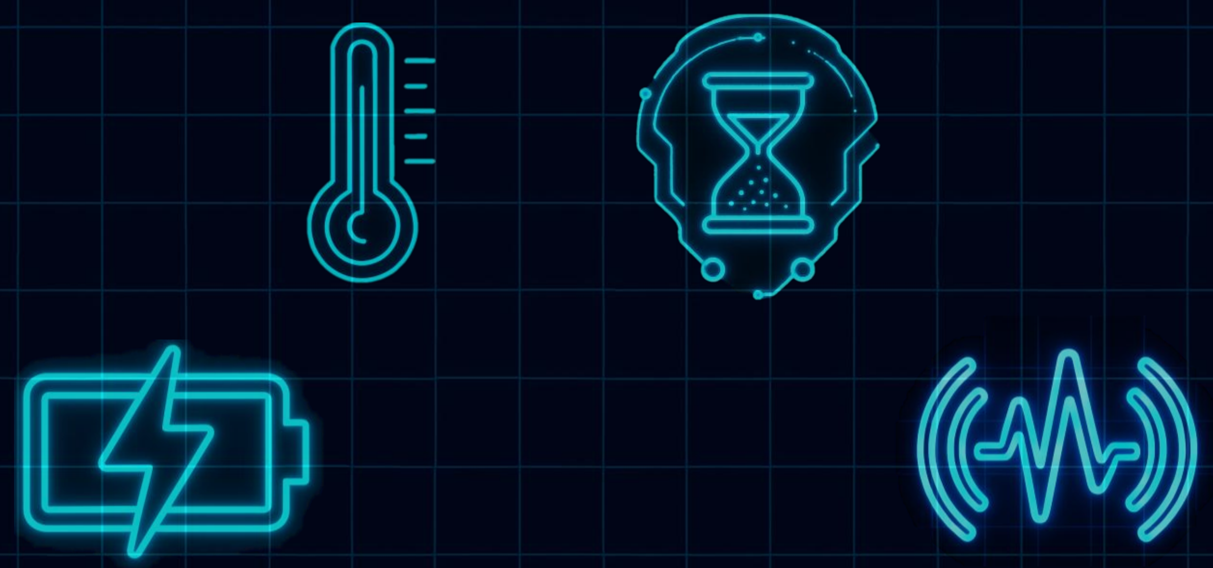


Figure 8.03 – External Influences on Rated Frequency

ATMega2560 Frequency Characterization Test

Use the following instructions to complete a frequency characterization test of the Arduino ATMega2560. This will measure the actual clock frequency of the microcontroller by comparing it against a known accurate reference signal— the PPS (pulse-per-second) output from a GPS module. Since the PPS signal generated by the GPS module is accurate to within ±30 nanoseconds, any significant deviation from the expected cycle count directly reveals the oscillator's drift from its rated frequency.

Terminology

- **Clock Frequency:**

  The number of clock cycles that occur per second, measured in hertz (Hz).

- **Clock Cycle:**

  A single tick of the clock signal. It represents the smallest unit of time in which an instruction or operation can begin or complete.

- **Clock Period:**

  The duration of one clock cycle, measured in seconds.

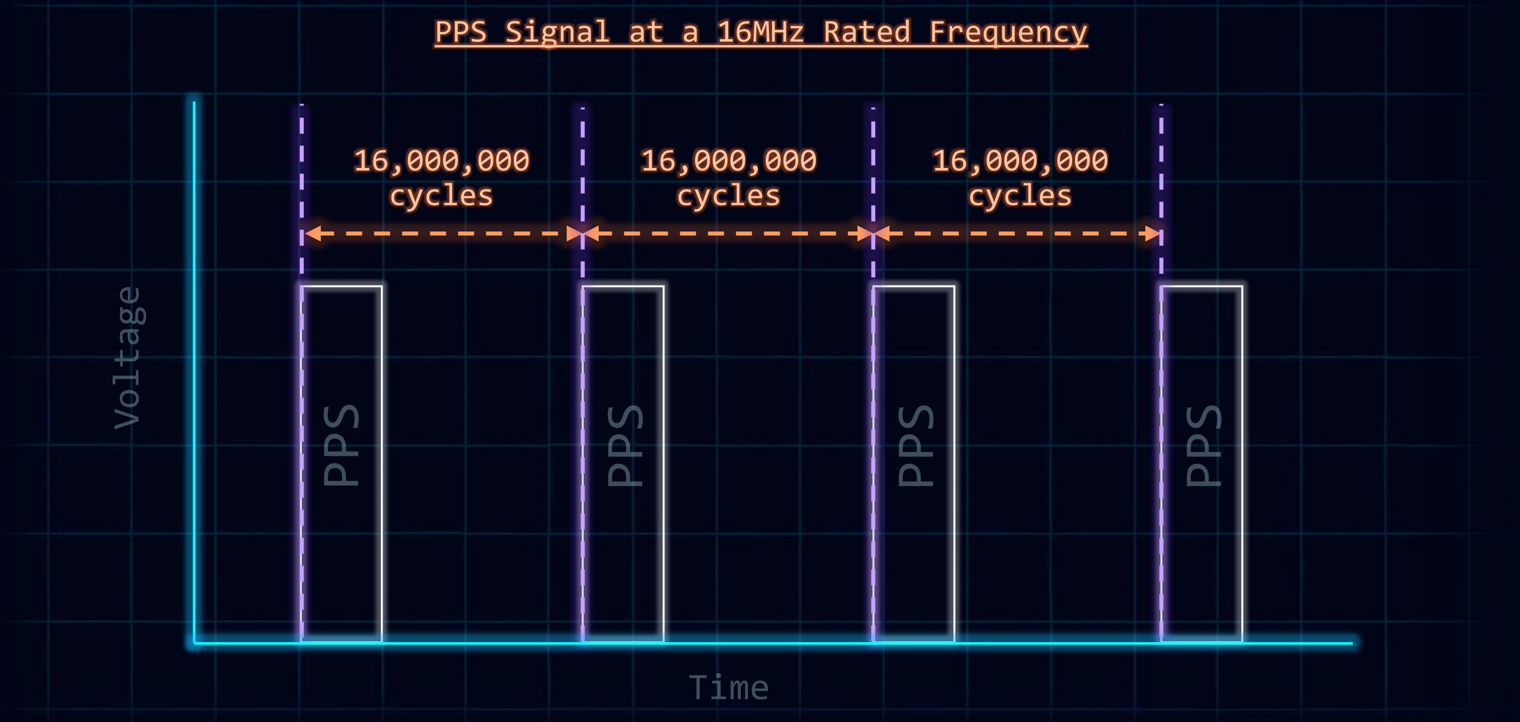
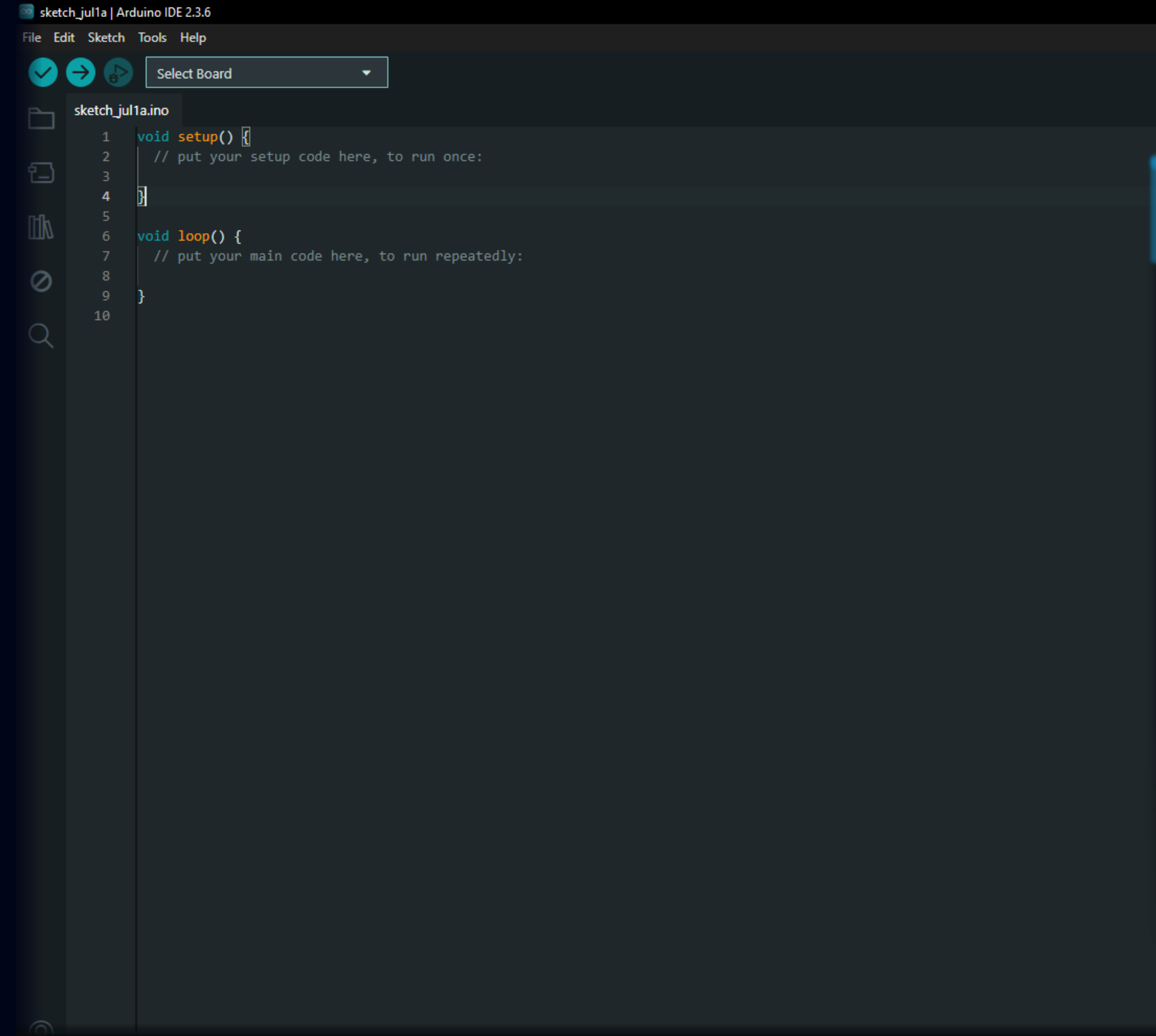


Figure 8.04 – PPS Signal with Expected Interval Timing



Launch Program

Locate the Arduino IDE icon on your desktop and launch the program

Sketches

The code you write in the Arduino IDE is called a sketch, and the Arduino compiler within the program handles all the setup to convert it into machine language for the microcontroller. It utilizes a simplified subset of C++ with a few custom libraries simplifying C++ to be more accessible for prototyping and hardware interaction.

Arduino to PC USB Connection

Connecting the USB Type-B cable between your computer and the Arduino board establishes a bidirectional communication link, allowing your computer to upload code to the microcontroller while also enabling the Arduino to send data back to the computer.

Connect the USB Type-B cable to your computer and the Arduino board.

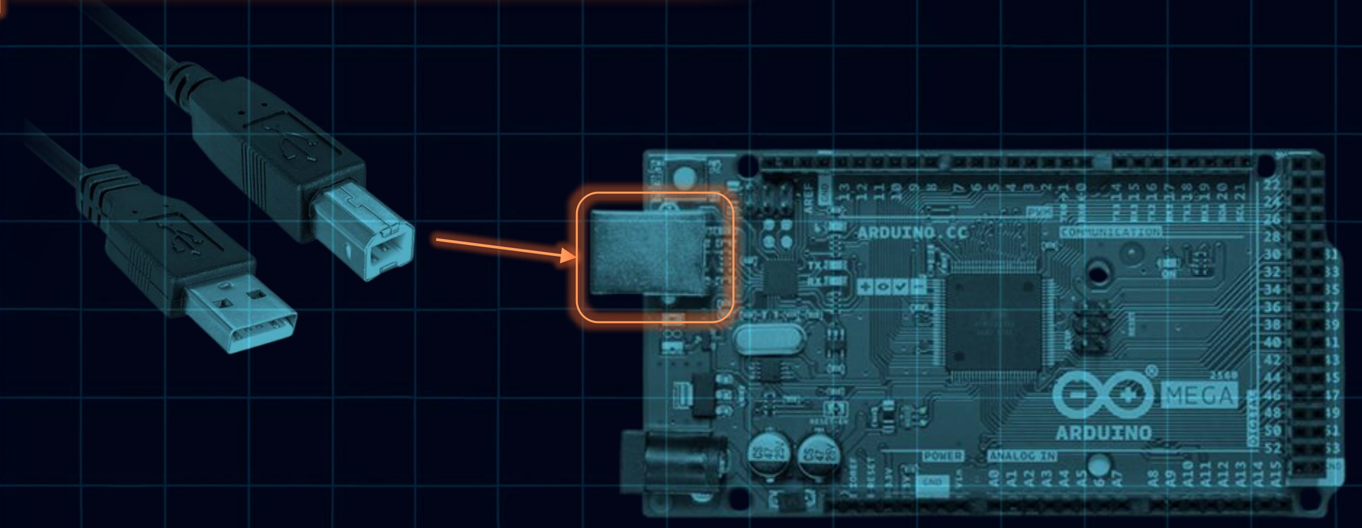


Figure 8.05 – Arduino-PC USB Connection

ATMega2560 Frequency Characterization Wiring Setup

The GPS module provides a PPS (Pulse Per Second) output, which emits a square pulse precisely once per second. This pulse is synchronized to GPS satellite time, which is maintained by atomic clocks onboard the satellites. When connected to the Arduino ATMega 2560, the PPS pin provides a highly accurate time reference.

Connections

Arduino	GPS
GND	GND
5V	VIN
Pin 48	PPS

The cable plugged into the socket at the top right of the GPS module is the satellite antenna. It must be connected for the module to receive satellite signals and provide GPS data.

Attach the GPS module to the breadboard using the jumper wires to make the following connections.

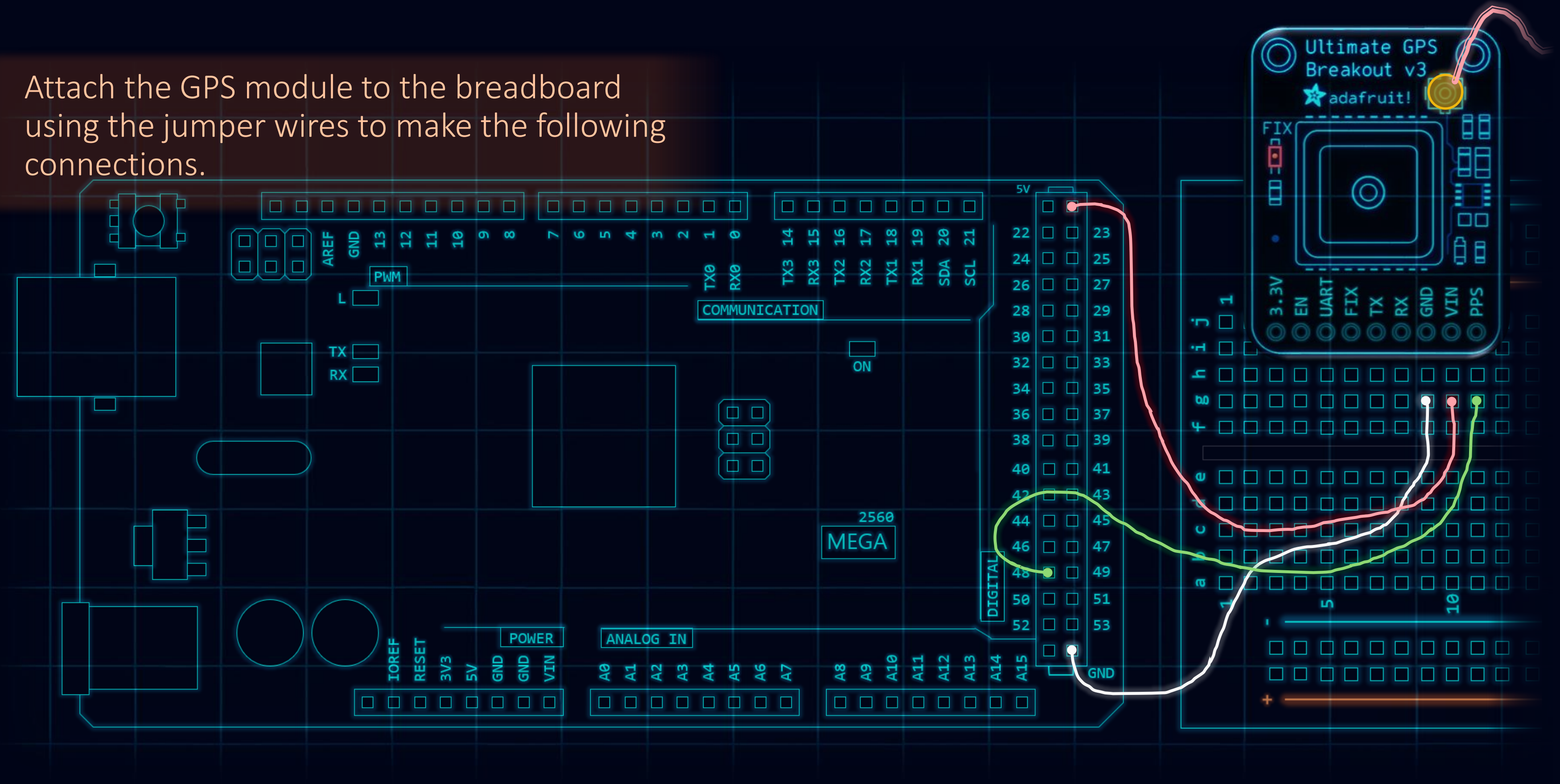


Figure 8.06 – Arduino ATMega2560 - Adafruit Ultimate GPS Breakout V3 - PPS Wiring Setup

QUEENSBOROUGH COMMUNITY COLLEGE

ATMega2560 Frequency Characterization Wiring Setup

This photo shows the PPS wiring connected to the Arduino Mega 2560. The PPS output from the GPS module is wired directly to digital pin 48, which serves as the input capture pin for Timer 5. Verify that power and ground are securely connected, as unstable power can disrupt signal accuracy. Lastly, ensure the GPS antenna is properly attached, since the PPS signal is only valid when the module has a satellite fix.

Connections

Arduino	GPS
GND	GND
5V	VIN
Pin 48	PPS

When you power on the GPS module, the FIX LED will start blinking. This indicates that it is searching for satellites. Once a position fix is acquired, the module will begin transmitting NMEA data.

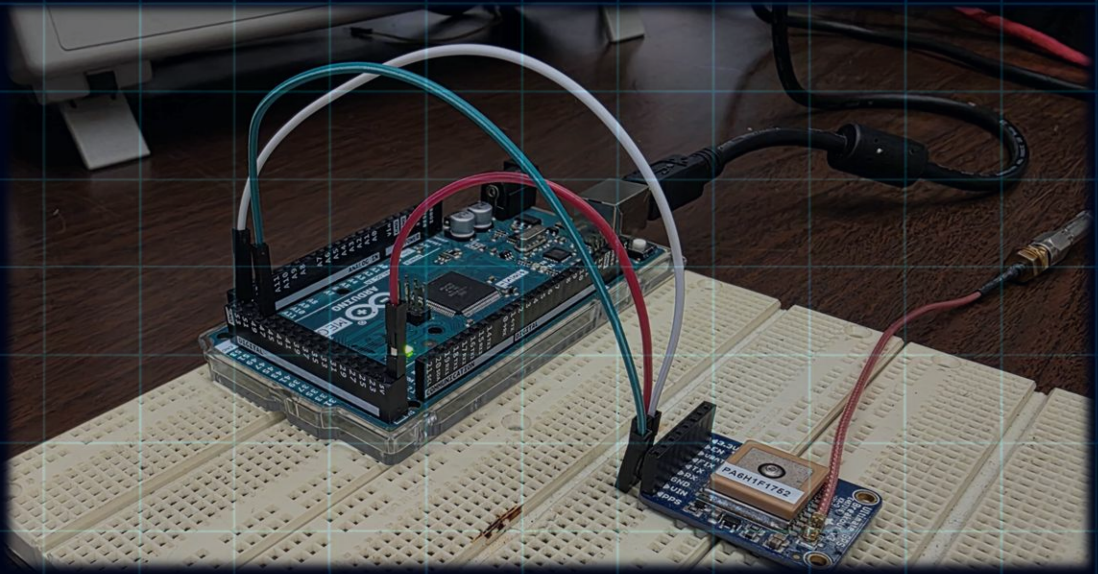


Figure 8.07 – Arduino ATMega2560 - Adafruit Ultimate GPS Breakout V3 Wiring Setup

QUEENSBOROUGH COMMUNITY COLLEGE

Frequency Characterization Test Sketch

Code Block

Read through the following code and try to understand the instructions being given to the Arduino.

Once done, paste the following Arduino sketch into the IDE and upload it to verify that the module is functional.

Make sure the baud rate in the Serial Monitor matches the one defined in your code. Here, it is set to 115200 as shown in:

Serial.begin(115200)

```
#define ICP5_Pin 48

volatile uint32_t ovf = 0;
volatile uint16_t icr = 0;
volatile bool pps = false;

void setup() {
   Serial.begin(115200);
   pinMode(ICP5_Pin, INPUT);
   TCCR5A = 0;
   TCCR5B = (1 << ICES5) | (1 << CS50); // Rising edge, no prescaler
   TIMSK5 = (1 << ICIE5) | (1 << TOIE5); // Enable input capture + overflow
   TCNT5 = 0;
   sei();
}

ISR(TIMER5_OVF_vect) {
   ovf++;
}

ISR(TIMER5_CAPT_vect) {
   uint16_t t = ICR5;
   uint32_t o = ovf;

   if ((TIFR5 & (1 << TOV5)) && t < 1000) o++; // Overflow correction
   icr = t;
   ovf = o;
   pps = true;
}

void loop() {
   static uint32_t last = 0;
   if (pps) {
      noInterrupts();
      uint32_t ticks = ((uint32_t)ovf << 16) | icr;
      pps = false;
      interrupts();

      if (last) {
         Serial.print(F("Total Ticks: "));
         Serial.println(ticks - last);
      }
      last = ticks;
   }
}
```

Frequency Characterization Test Results

Serial Output

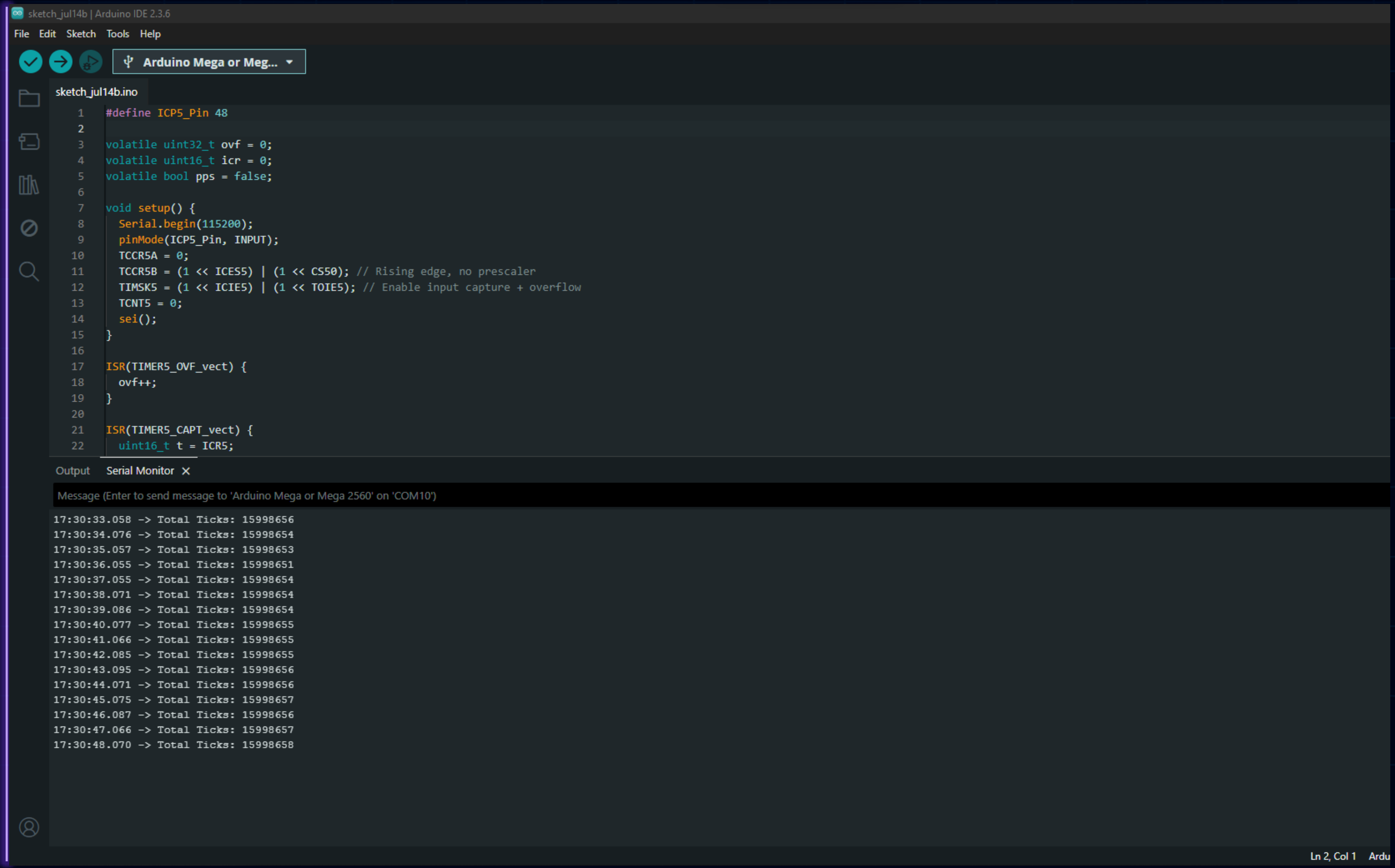
The Serial Monitor should display the number of clock ticks counted between each PPS (Pulse Per Second) signal.

If everything is functioning correctly, these values should be close to 16,000,000 ticks, matching the 16 MHz system clock.

We'll now switch to PuTTY to log this data for post-processing.

By collecting multiple tick values using PuTTY — ideally 1,000 samples or more — we can calculate the true clock period of the resonator by averaging the intervals between PPS pulses.

Be sure to close the Serial Monitor before running PuTTY as only one program can access the COM port at a time.



PuTTY Serial Port Parameters

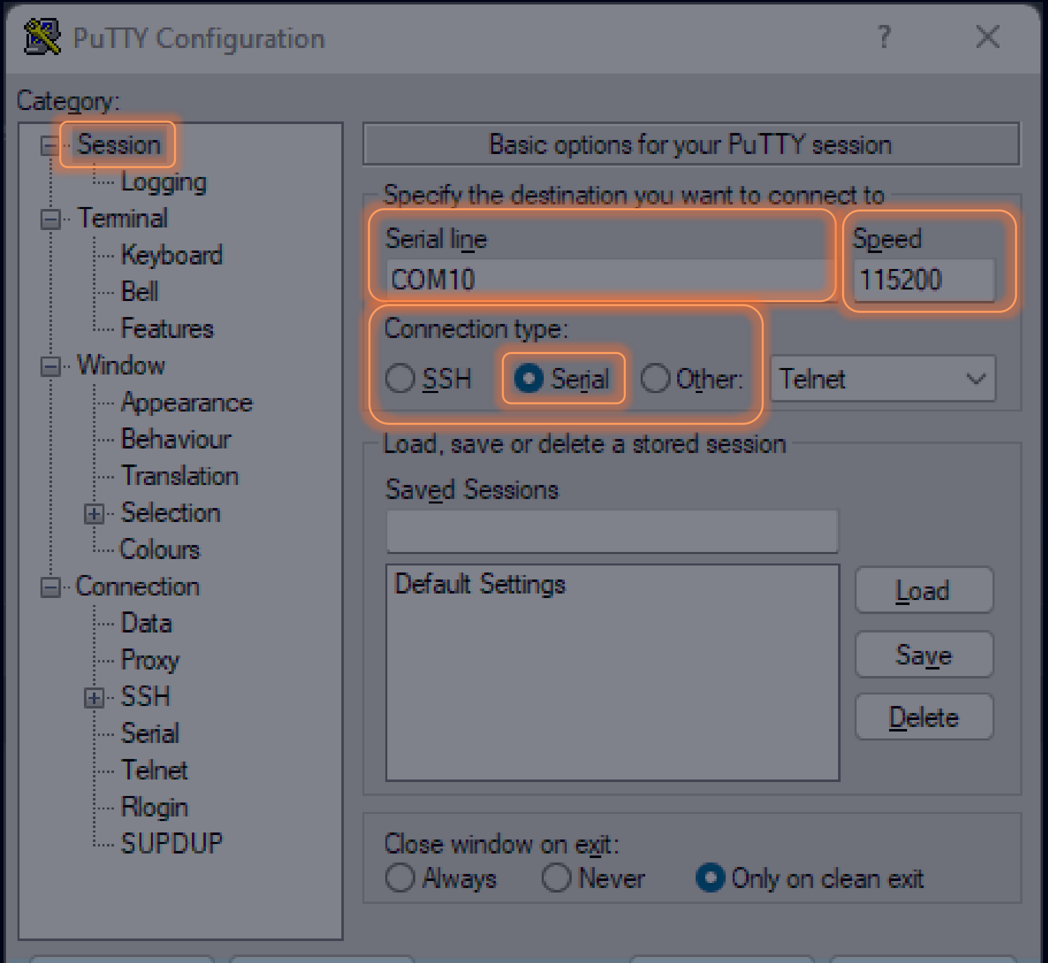
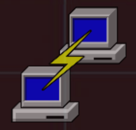


Figure 8.08 – PuTTY - Session Configuration Menu

Session Configuration

- Locate the PuTTY icon on your desktop and launch the program.



- Click on the Sessions menu item.
- Set the connection type to Serial to enable communication between the Arduino and your computer.
- Identify the COM port your ATmega2560 is using (check under Arduino IDE → Tools → Port) and enter it in PuTTY.
- Match the speed to the baud rate in your Arduino sketch. For this setup, use 115200.

PuTTY File Saving Parameters

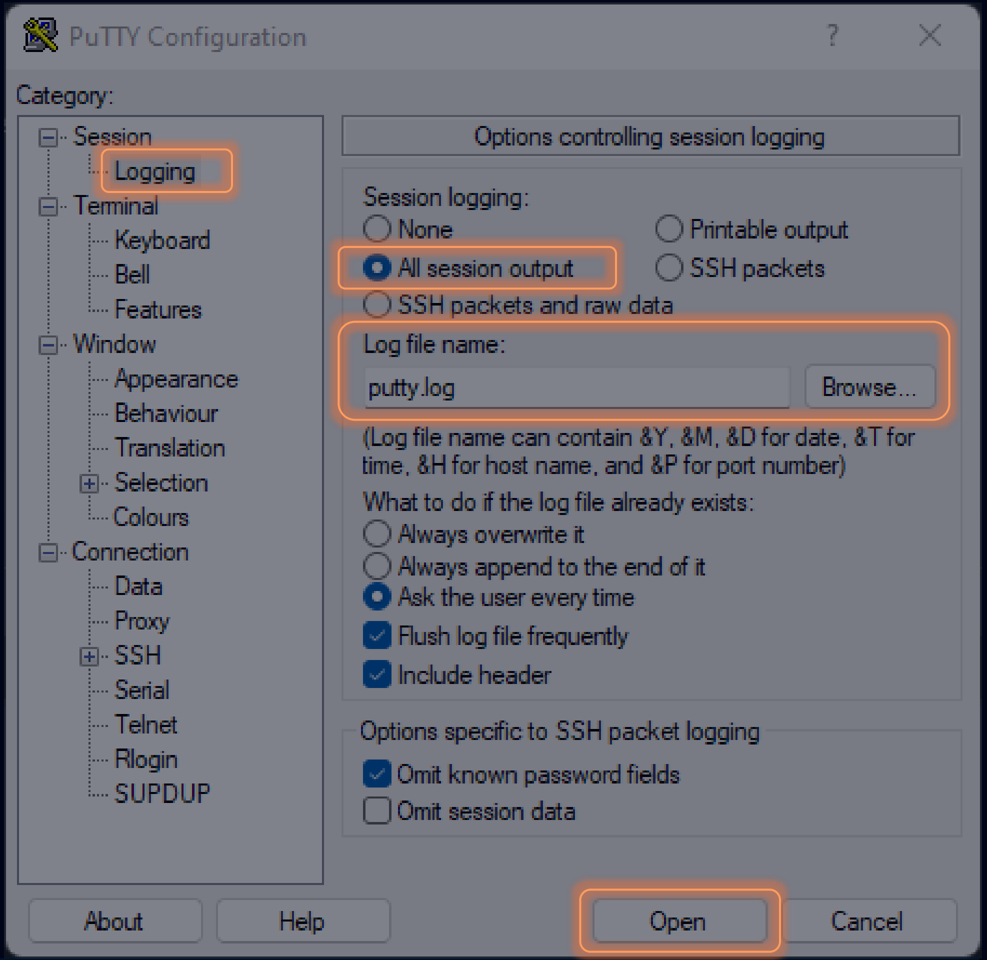


Figure 8.09 – PuTTY - Logging Configuration Menu

Logging Configuration

- Click on the Logging menu item from the left-hand side.
- Select "All Session Output" under the session logging options.
- Enter a name that reflects the data you're collecting. To choose a different save location, click the Browse button.

⚠ You MUST close the Serial Monitor in the Arduino IDE before starting PuTTY. Only one program can access the COM port at a time — failure to do so will cause connection errors.

- Once the Arduino IDE Serial Monitor is closed, hit Open in PuTTY.

Exporting Clock Data for Analysis

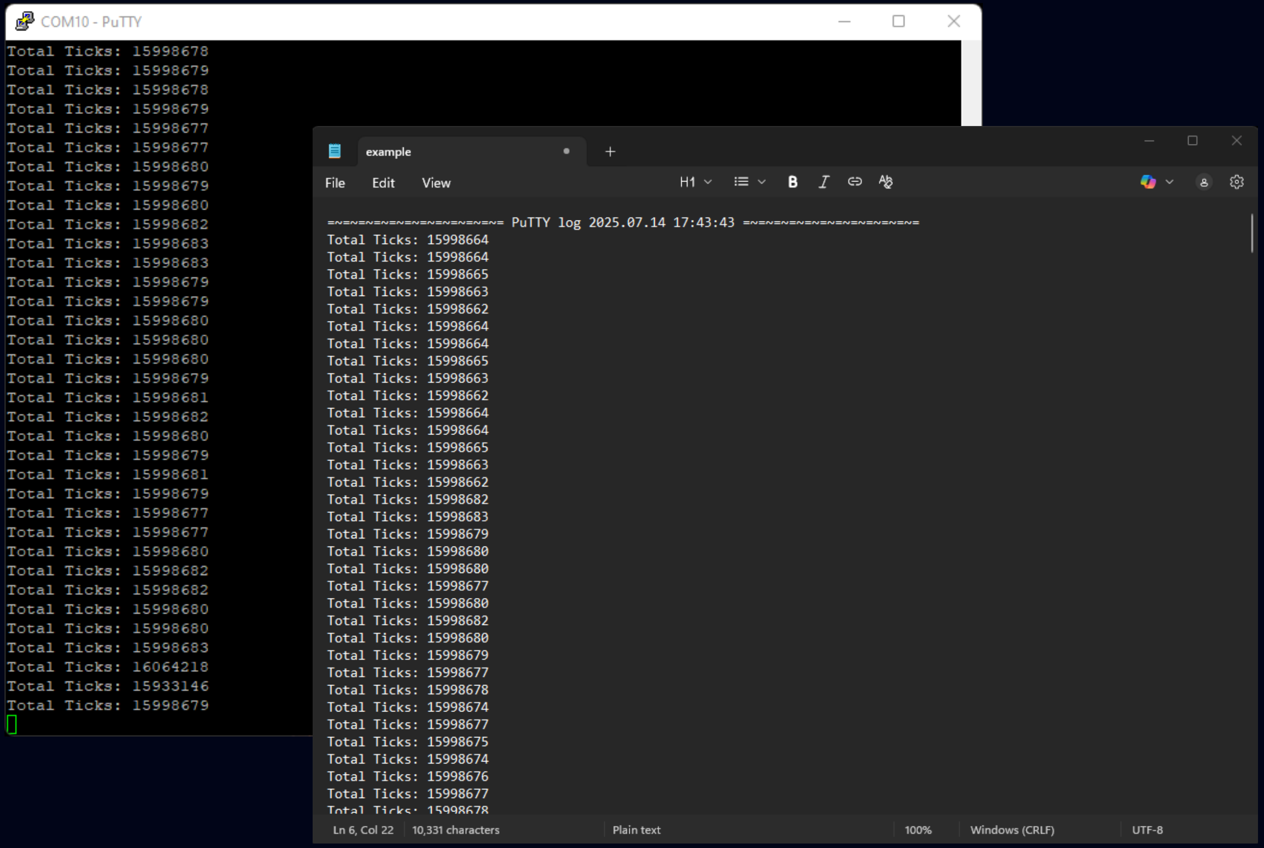


Figure 8.10 – PuTTY: Terminal Window and Text File

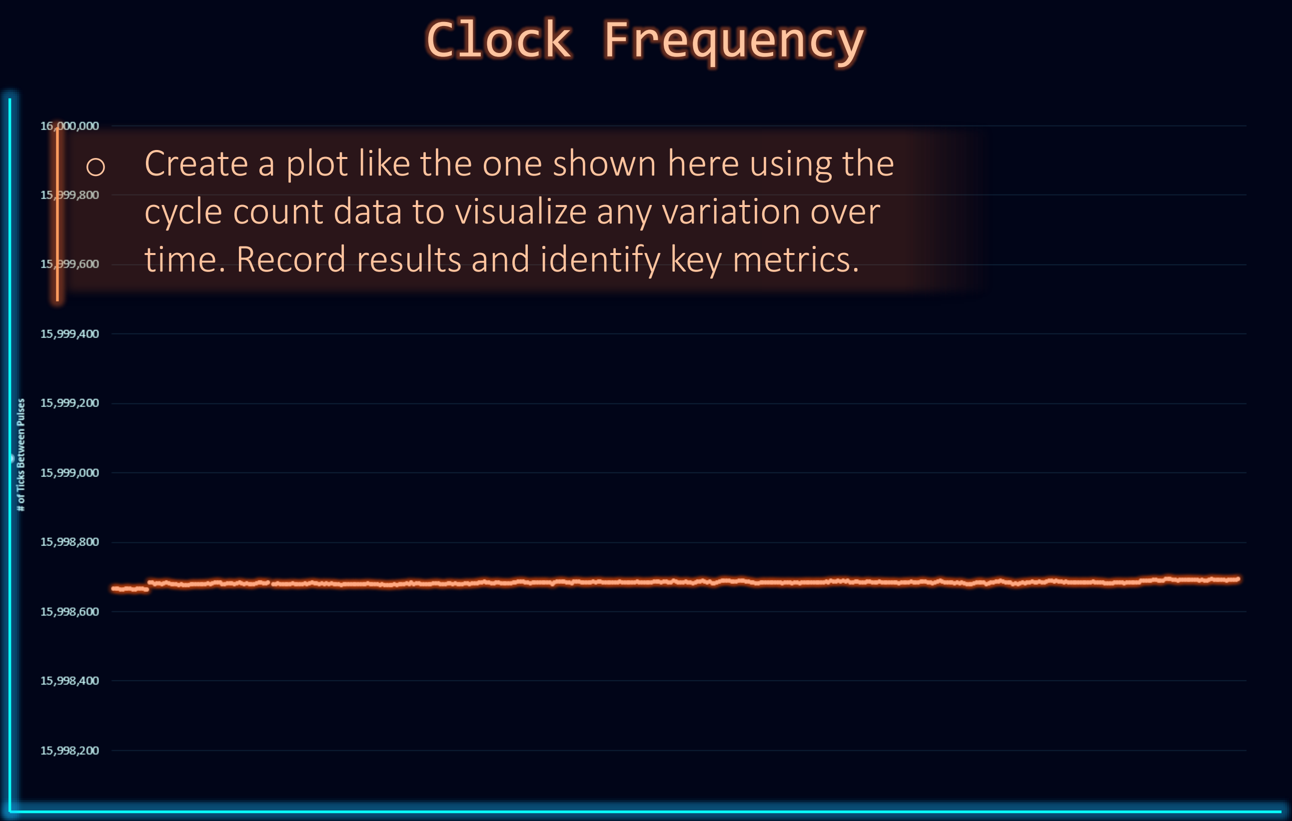
Data Collection

Export to Excel

- Let the program run for at least 20 minutes before closing the PuTTY window.
- When you're done, simply close PuTTY — it will automatically save the session data to the file and location you specified during setup.
- Locate the text file that was generated during the session.
- Open the file and copy all the data from the text file.
- Now, open Excel and paste the values into a single column.

QUEENSBOROUGH COMMUNITY COLLEGE

Data Processing in Microsoft Excel



- Create a plot like the one shown here using the cycle count data to visualize any variation over time. Record results and identify key metrics.

Figure 8.11 – Clock Frequency Scatter Plot

Key Metrics

Clock Frequency Range
Identify the highest and lowest clock frequencies measured between GPS PPS pulses.

Average Clock Frequency (measured in hertz)
Calculate by averaging the clock frequencies between PPS pulses over time.

Average Clock Period (measured in seconds)
The average duration of one clock cycle, calculated as the inverse of the average frequency. Indicates how long each cycle takes to complete on average.

Clock Drift (measured in ppm)
Quantifies how much the system clock deviates from its expected frequency. A positive or negative value shows if the clock runs fast or slow.

Standard Deviation
Measures how much the cycle count varies from second to second. A lower value indicates stable timing; a higher value suggests jitter or noise in the oscillator.

Module VIII

ATMega2560 – Frequency Shift by Temperature

QUEENSBOROUGH COMMUNITY COLLEGE

Frequency Shift by Temperature

In this module, we measure the Arduino’s clock frequency using the PPS signal from the GPS module, as in the previous setup. However, this time we conduct the experiment alongside a temperature sensor to examine how ambient temperature affects the clock’s stability over time, revealing potential temperature-induced oscillator drift.

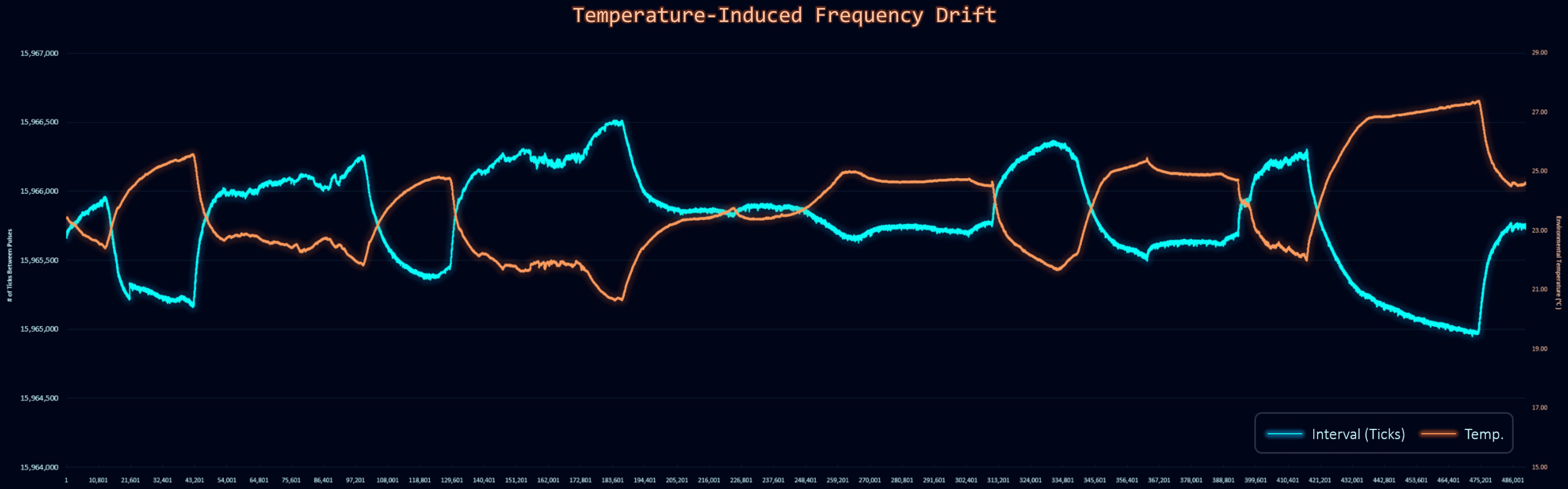


Figure 9.01 – Chart Displaying Temperature-Frequency Relationship

Temperature Drift Wiring Setup Pt. I

The GPS module provides a PPS (Pulse Per Second) output, which emits a square pulse precisely once per second. This pulse is synchronized to GPS satellite time, which is maintained by atomic clocks onboard the satellites. When connected to the Arduino ATMega 2560, the PPS pin provides a highly accurate time reference.

Connections

Arduino	GPS
GND	GND
5V	VIN
Pin 48	PPS

The cable plugged into the socket at the top right of the GPS module is the satellite antenna. It must be connected for the module to receive satellite signals and provide GPS data.

Attach the GPS module to the breadboard using the jumper wires to make the following connections.

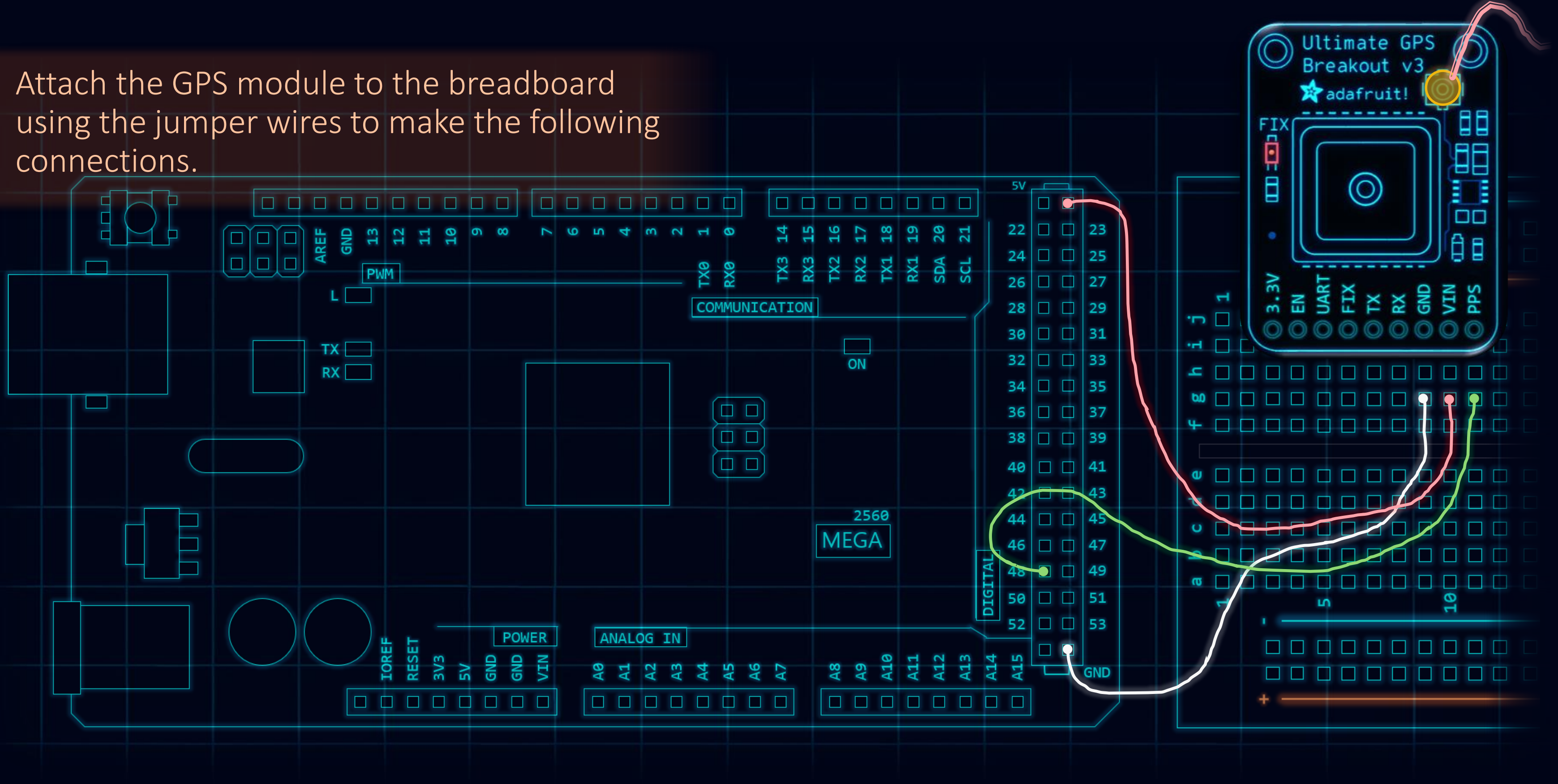


Figure 9.02 – Arduino ATMega2560 - Adafruit Ultimate GPS Breakout V3 - PPS Wiring Setup

Temperature Drift Wiring Setup Pt. II

The BMP280 uses SPI (Serial Peripheral Interface) to communicate with the Arduino ATMega 2560. SPI is a fast, synchronous protocol ideal for high-speed sensor data transfer. It allows the microcontroller to exchange data with the sensor using a master-slave architecture over just four data lines.

Connections

Arduino	BMP280
3.3V	VIN
GND	GND
Pin 52	SCK
Pin 50	SDO
Pin 51	SDI
Pin 53	CS

Attach the BMP280 sensor to the breadboard using the jumper wires to make the following connections.

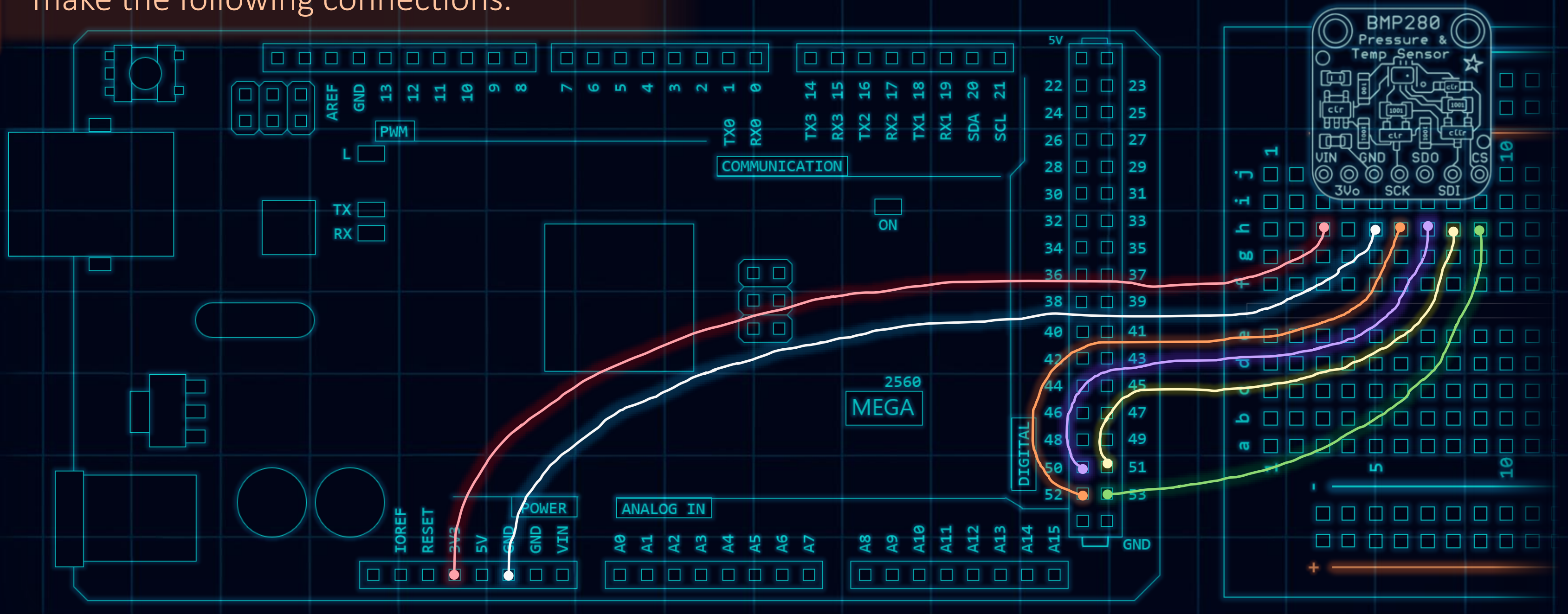


Figure 9.03 – Arduino ATMega2560 - BMP280 Sensor Wiring Setup

Temperature Drift Complete Wiring Setup

This photo shows the Arduino connected to both the GPS module and a temperature sensor. The GPS provides precise PPS timing signals, while the temperature sensor monitors ambient conditions. Together, these components allow us to analyze how temperature drift affects the Arduino's resonator over time by correlating changes in temperature with variations in measured tick counts between PPS pulses.

Connections

Arduino	BMP280
3.3V	VIN
GND	GND
Pin 52	SCK
Pin 50	SDO
Pin 51	SDI
Pin 53	CS

Arduino	GPS
GND	GND
5V	VIN
Pin 48	PPS

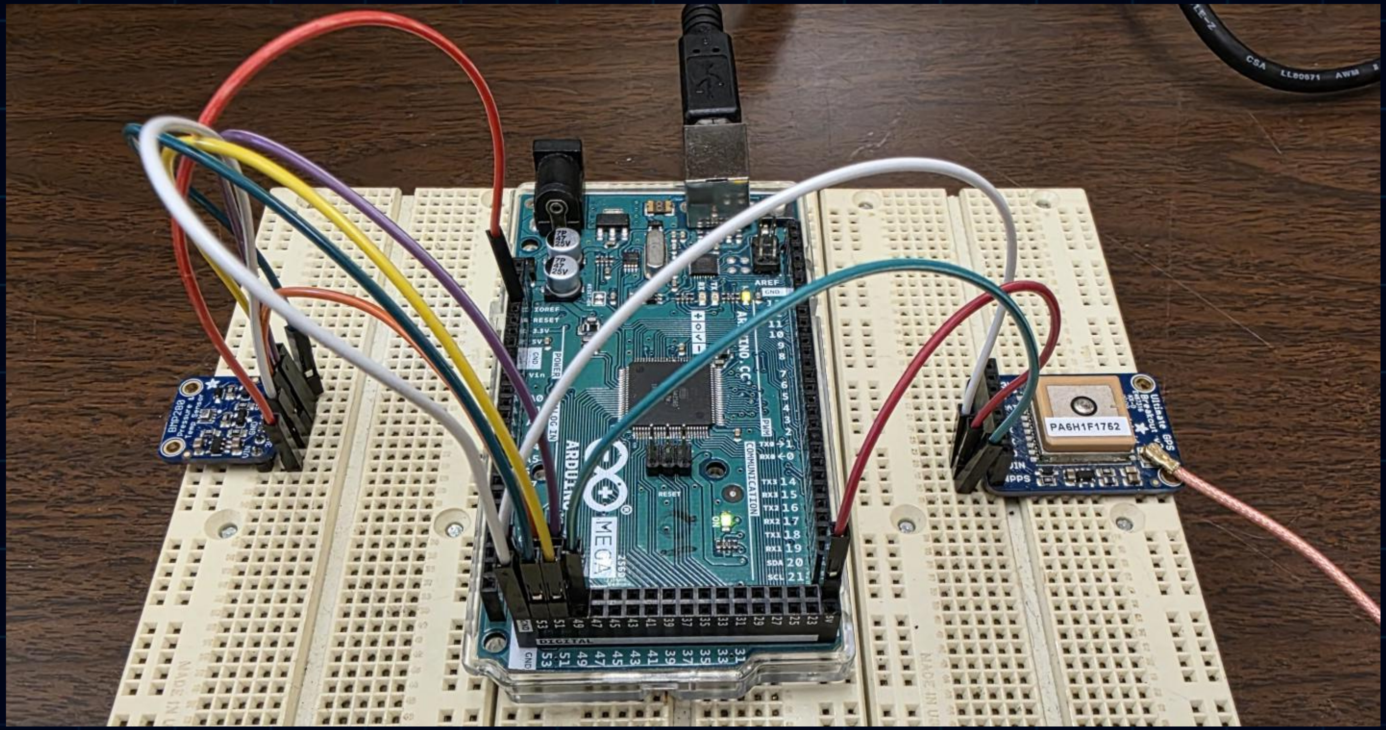


Figure 9.04 – Arduino-BMP280 Sensor Wiring Setup

Temperature Drift Test Sketch

Code Block

Read through the following code and try to understand the instructions being given to the Arduino.

Once done, paste the following Arduino sketch into the IDE and upload it to verify that the module is functional.

```
#include <Arduino.h>
#include <SPI.h>
#include <Adafruit_BMP280.h>

#define PPS_PIN 48
#define BMP_CS 53  // chip select for BMP280

constexpr uint16_t OVERFLOW_CORRECTION_THRESHOLD = 1000;

volatile uint32_t OVF5 = 0;

struct Timestamp {
  volatile uint16_t count;
  volatile uint32_t overflow;
  volatile bool flag;
  uint32_t prevTicks = UINT32_MAX;
};

Timestamp pps;
Adafruit_BMP280 bmp(BMP_CS);  // SPI constructor

void printInterval(uint32_t interval, float temperature) {
  Serial.print(temperature, 2);
  Serial.print(F(", "));
  Serial.println(interval);
}

void initTimer5() {
  TCCR5A = 0;
  TCCR5B = (1 << ICES5) | (1 << CS50);  // rising edge, no prescale
  TIMSK5 = (1 << ICIE5) | (1 << TOIE5); // input capture + overflow
  TCNT5 = 0;
}

ISR(TIMER5_OVF_vect) { OVF5++; }

ISR(TIMER5_CAPT_vect) {
  pps.count = ICR5;
  pps.overflow = OVF5;
  pps.flag = true;
}

void setup() {
  Serial.begin(115200);
  pinMode(BMP_CS, INPUT);
  pinMode(PPS_PIN, INPUT);
  initTimer5();

  if (!bmp.begin(BMP_CS)) {
    Serial.println(F("BMP280 fail"));
    while (1);
  }

  bmp.setSampling(
    Adafruit_BMP280::MODE_NORMAL,
    Adafruit_BMP280::SAMPLING_X1,   // fastest temperature
    Adafruit_BMP280::SAMPLING_X1,   // fastest pressure
    Adafruit_BMP280::FILTER_OFF,    // no filter
    Adafruit_BMP280::STANDBY_MS_1   // minimal standby
  );
}

void loop() {
  if (pps.flag) {
    noInterrupts();
    uint16_t t = pps.count;
    uint32_t o = pps.overflow;
    pps.flag = false;
    interrupts();

    if ((TIFR5 & (1 << TOV5)) && t < OVERFLOW_CORRECTION_THRESHOLD) o++;

    uint32_t ticks = (o << 16) | t;
    uint32_t interval = (pps.prevTicks != UINT32_MAX) ? (ticks - pps.prevTicks) : 0;
    pps.prevTicks = ticks;

    if (interval != 0) {
      float temperature = bmp.readTemperature();
      printInterval(interval, temperature);
    }
  }
}
```

Make sure the baud rate in the Serial Monitor matches the one defined in your code. Here, it is set to 115200 as shown in:

Serial.begin(115200)

Temperature Drift Test Results

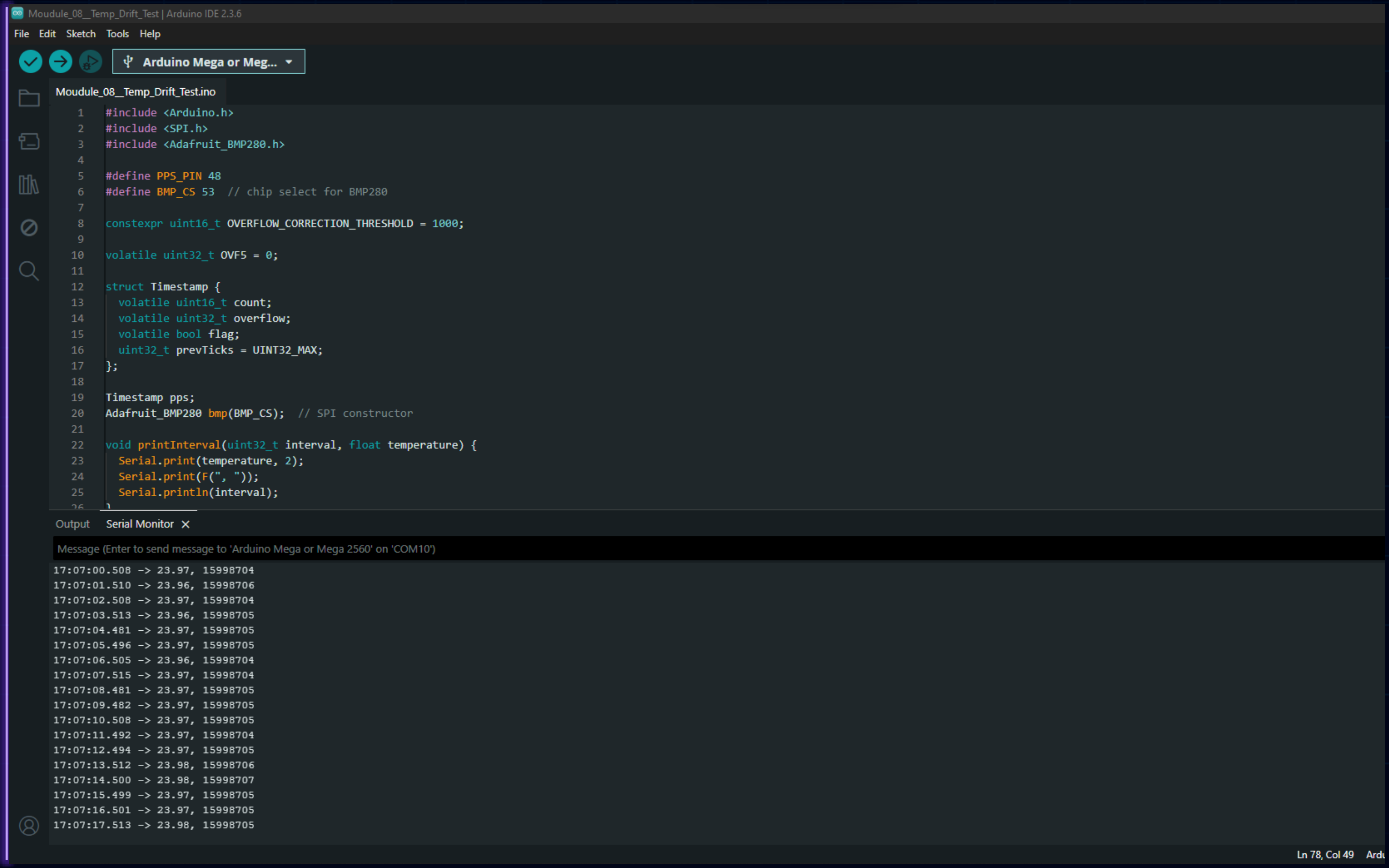
Serial Output

The Serial Monitor should display the data as shown in this example.

- The first value is the temperature reading from the BMP280 sensor (in degrees Celsius).
- The second value is the number of clock ticks counted between two consecutive PPS (Pulse Per Second) signals from the GPS module.

Now that we've confirmed everything is working correctly, it's time to begin recording data using PuTTY.

Be sure to close the Serial Monitor before running PuTTY as only one program can access the COM port at a time.



PuTTY Serial Port Parameters

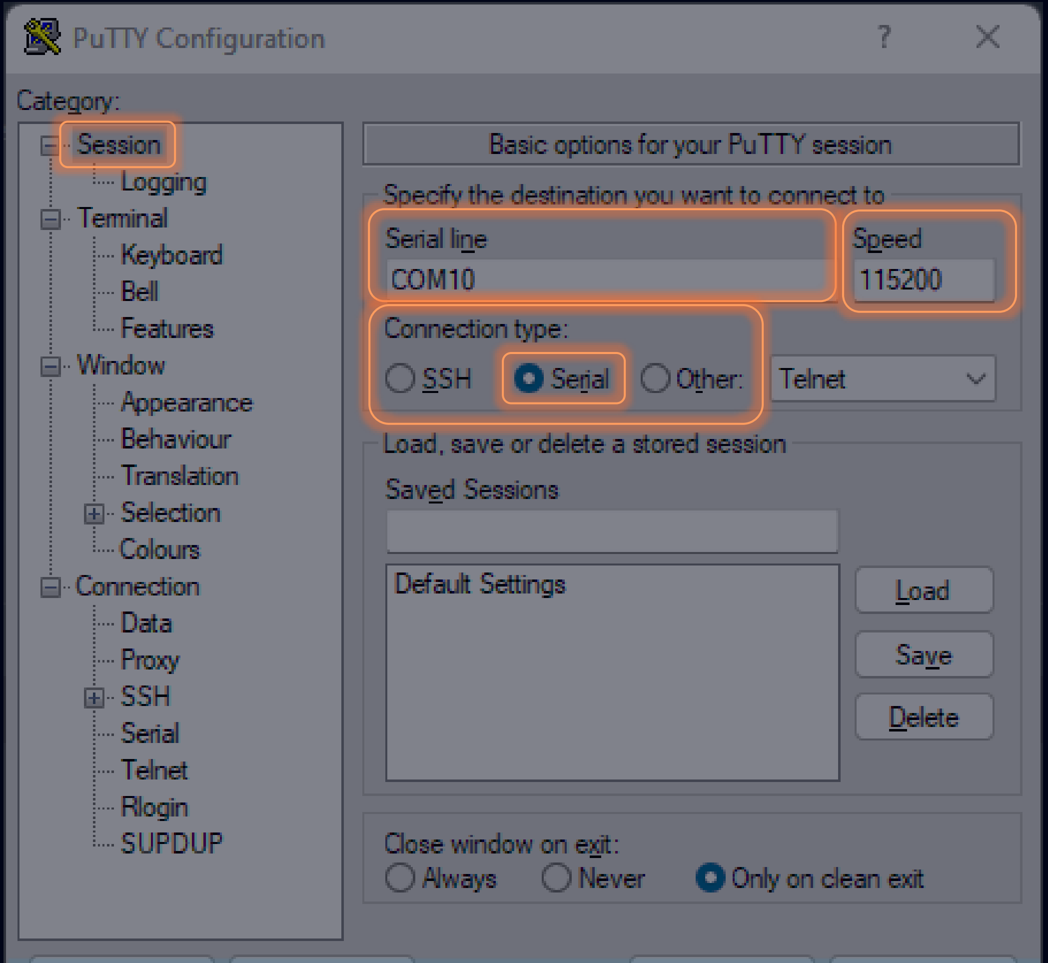


Figure 8.08 – PuTTY - Session Configuration Menu

Session Configuration

- Locate the PuTTY icon on your desktop and launch the program.

- Click on the Sessions menu item.
- Set the connection type to Serial to enable communication between the Arduino and your computer.
- Identify the COM port your ATmega2560 is using (check under Arduino IDE → Tools → Port) and enter it in PuTTY.
- Match the speed to the baud rate in your Arduino sketch. For this setup, use 115200.

PuTTY File Saving Parameters

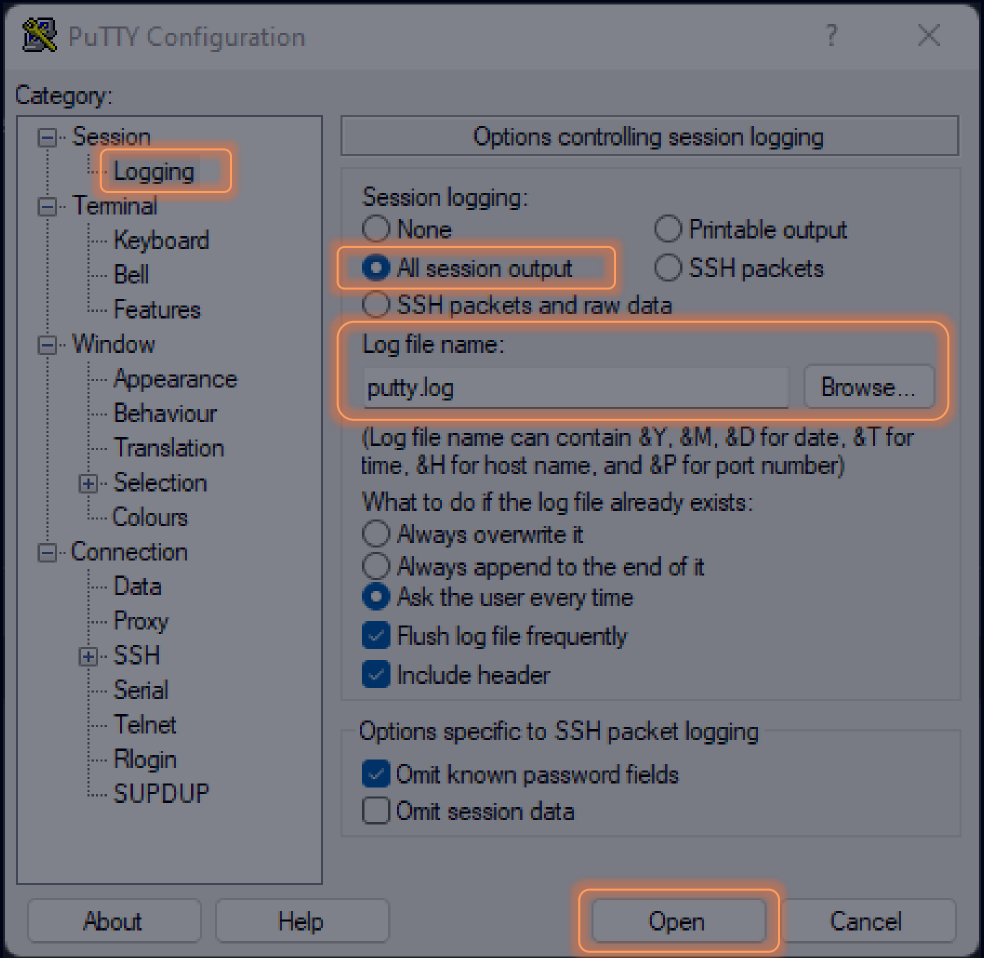


Figure 9.06 – PuTTY - Logging Configuration Menu

Logging Configuration

- Click on the Logging menu item from the left-hand side.
- Select "All Session Output" under the session logging options.
- Enter a name that reflects the data you're collecting. To choose a different save location, click the Browse button.

You MUST close the Serial Monitor in the Arduino IDE before starting PuTTY. Only one program can access the COM port at a time — failure to do so will cause connection errors.

- Once the Arduino IDE Serial Monitor is closed, hit Open in PuTTY.

Exporting Drift Data for Analysis

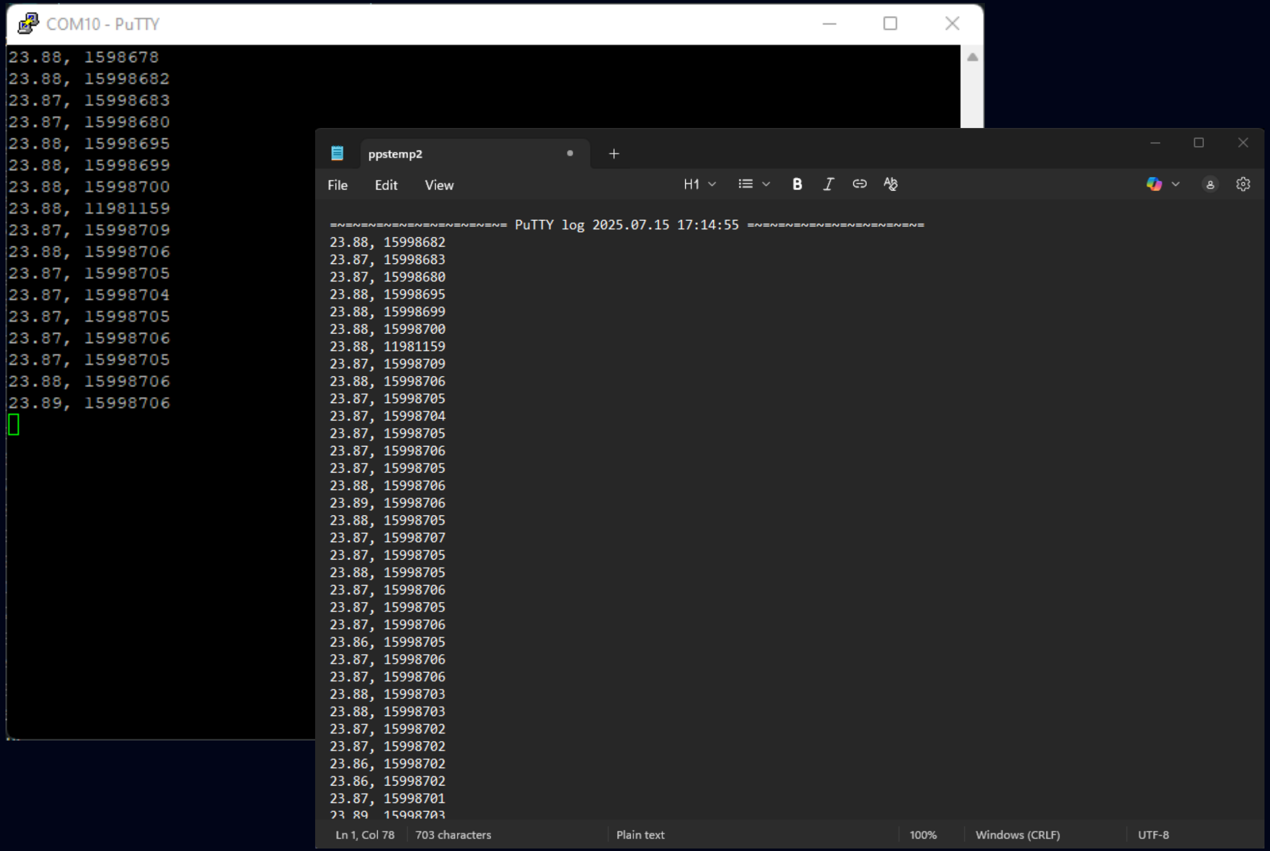


Figure 9.07 – PuTTY: Terminal Window and Text File

Data Collection

Export to Excel

- Let the program run for one week before closing the PuTTY window.
- When you're done, simply close PuTTY — it will automatically save the session data to the file and location you specified during setup.
- Locate the text file that was generated during the session.
- Open the file and copy all the data from the text file.
- Now, open Excel and paste the values into a single column.

Data Processing in Microsoft Excel

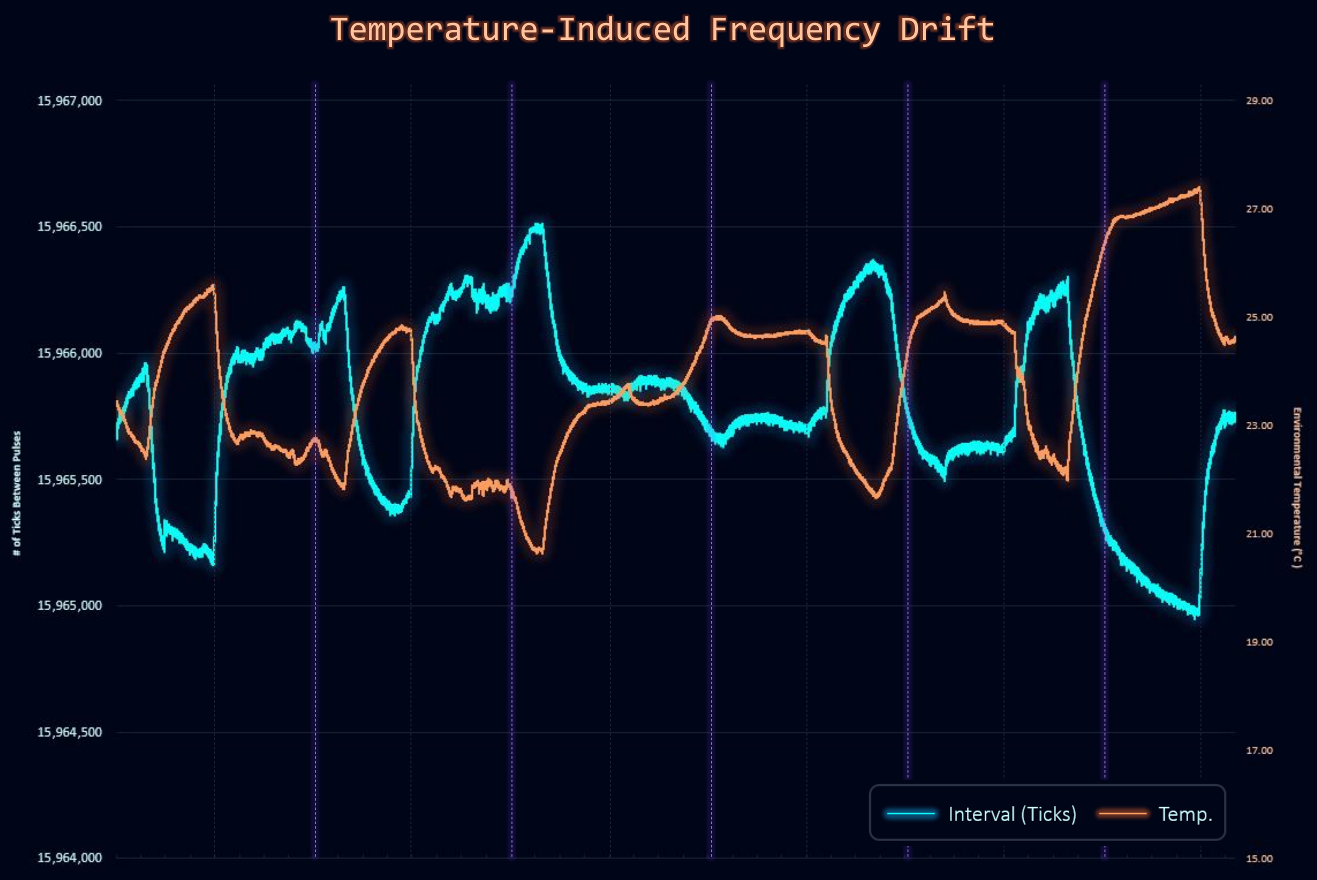


Figure 9.11 – Oscillator Drift Scatter Plot

Key Metrics

Clock Frequency Range
Identify the highest and lowest clock frequencies measured between GPS PPS pulses.

Temperature Range
Identify the minimum and maximum temperatures recorded during the experiment.

Average Clock Frequency (measured in hertz)
Calculate by averaging the clock frequencies between PPS pulses over time.

Average Clock Period (measured in seconds)
The average duration of one clock cycle, calculated as the inverse of the average frequency. Indicates how long each cycle takes to complete on average.

QUEENSBOROUGH COMMUNITY COLLEGE

Data Processing in Microsoft Excel

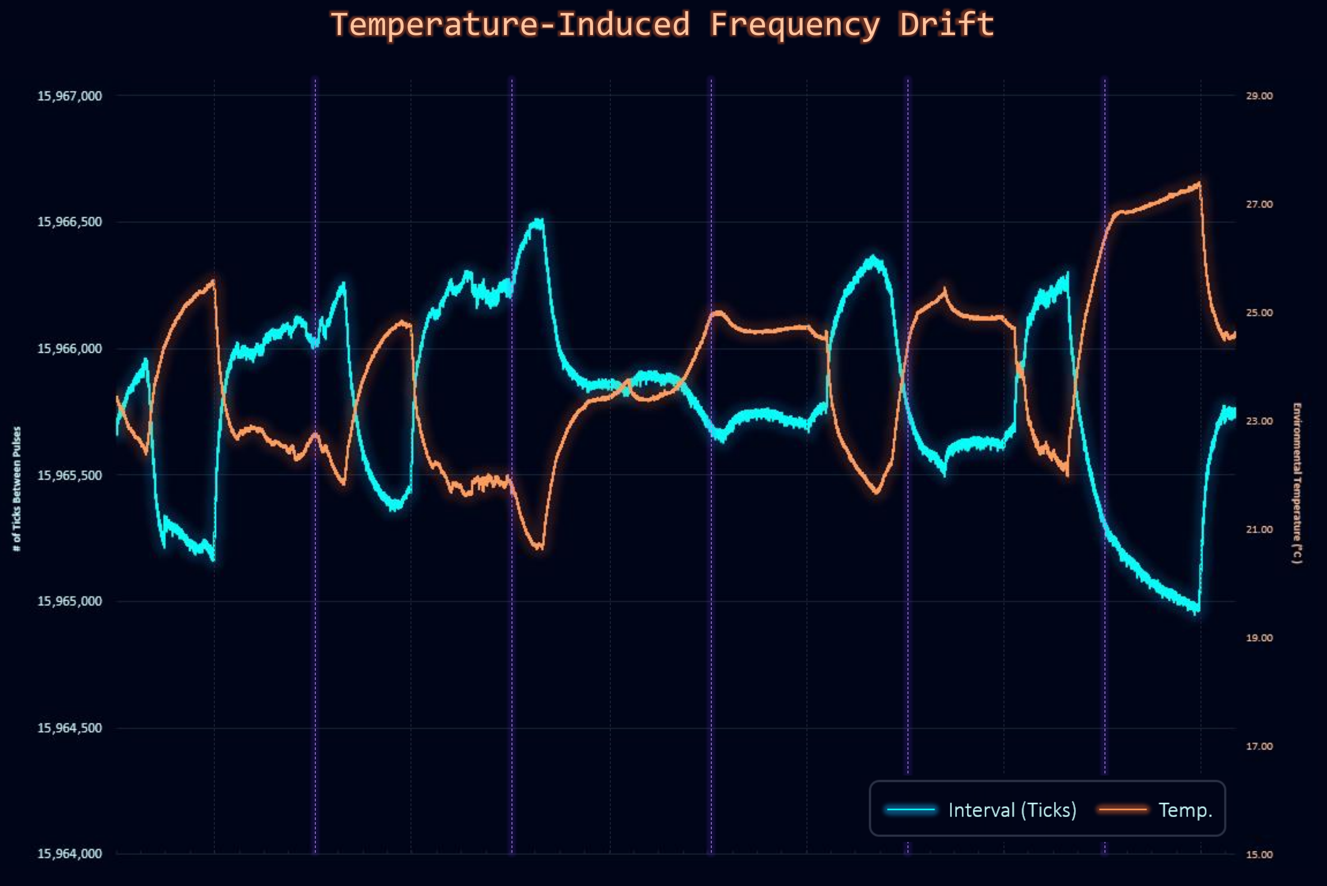


Figure 9.11 – Oscillator Drift Scatter Plot

Key Metrics

Clock Drift (measured in ppm)
Quantifies how much the system clock deviates from its expected frequency. A positive or negative value shows if the clock runs fast or slow.

Standard Deviation
Measures how much the cycle count varies from second to second. A lower value indicates stable timing; a higher value suggests jitter or noise in the oscillator.

Temperature Coefficient of Frequency
Quantifies how much the clock frequency changes with temperature, expressed in parts per million per degree Celsius (ppm/°C).

Thermal Shift Direction & Rate
Identifies whether the clock speeds up or slows down as temperature changes, and how rapidly this shift occurs across a given temperature range.

Module IX

Cosmic Ray Shower Simulation: Timestamping Pulse Generator Signals

QUEENSBOROUGH COMMUNITY COLLEGE

Module X

XBee3 Radio Module – Wireless Induced Latency

QUEENSBOROUGH COMMUNITY COLLEGE

Wireless GPS Data Relay System

To enable wireless GPS data collection, we use a combination of Adafruit GPS and XBee3 radio modules. This setup allows GPS information to be captured by one device and transmitted wirelessly to another system for processing.

Data Flow

- In our circuit, the Adafruit GPS module outputs real-time NMEA data and a PPS (Pulse Per Second) signal through a wired connection to an XBee3 Coordinator.
- The Coordinator then transmits both the NMEA data and the PPS signal wirelessly to a paired XBee3 End Device.
- The End Device then sends the received NMEA data and PPS Signal via a wired connection to the ATMega2560.

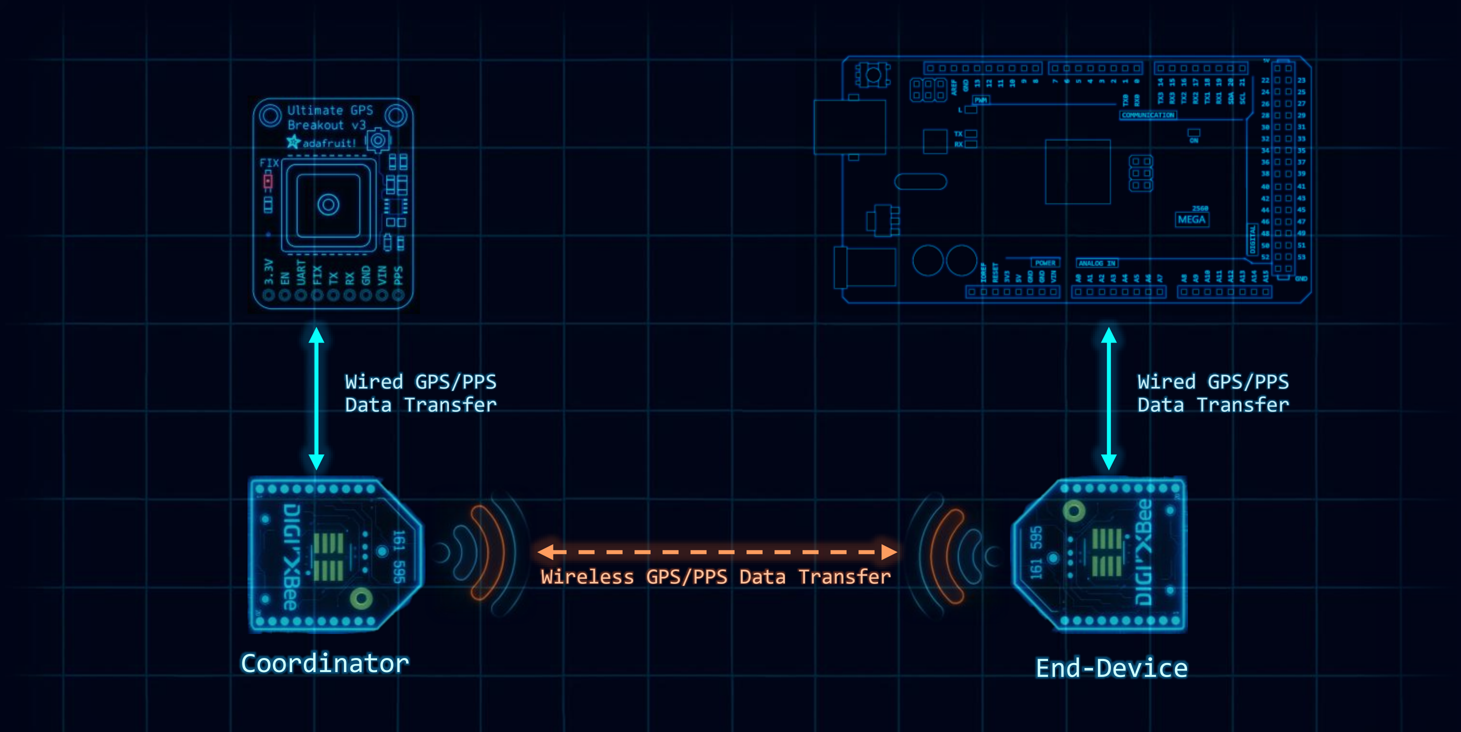


Figure 9.01 – Wireless GPS Data Relay Concept

Timing Implications of Wireless Transmission

While GPS modules generate a highly accurate Pulse Per Second (PPS) signal synchronized to atomic satellite clocks, wirelessly transmitting this signal using XBee3 modules introduces latency. Unlike a direct electrical connection, the PPS signal must be detected, queued, and packetized by the XBee3 Coordinator and then transmitted to the paired XBee3 End-Device which must decode and regenerate the signal.

This entire process can result in a variable delay influenced by factors such as RF interference, signal strength, protocol overhead, and the internal processing time of the radio system. While the general shape and rising edge of the PPS signal may be preserved, its precise alignment with the true UTC second may drift slightly when delivered wirelessly as seen in the chart below.

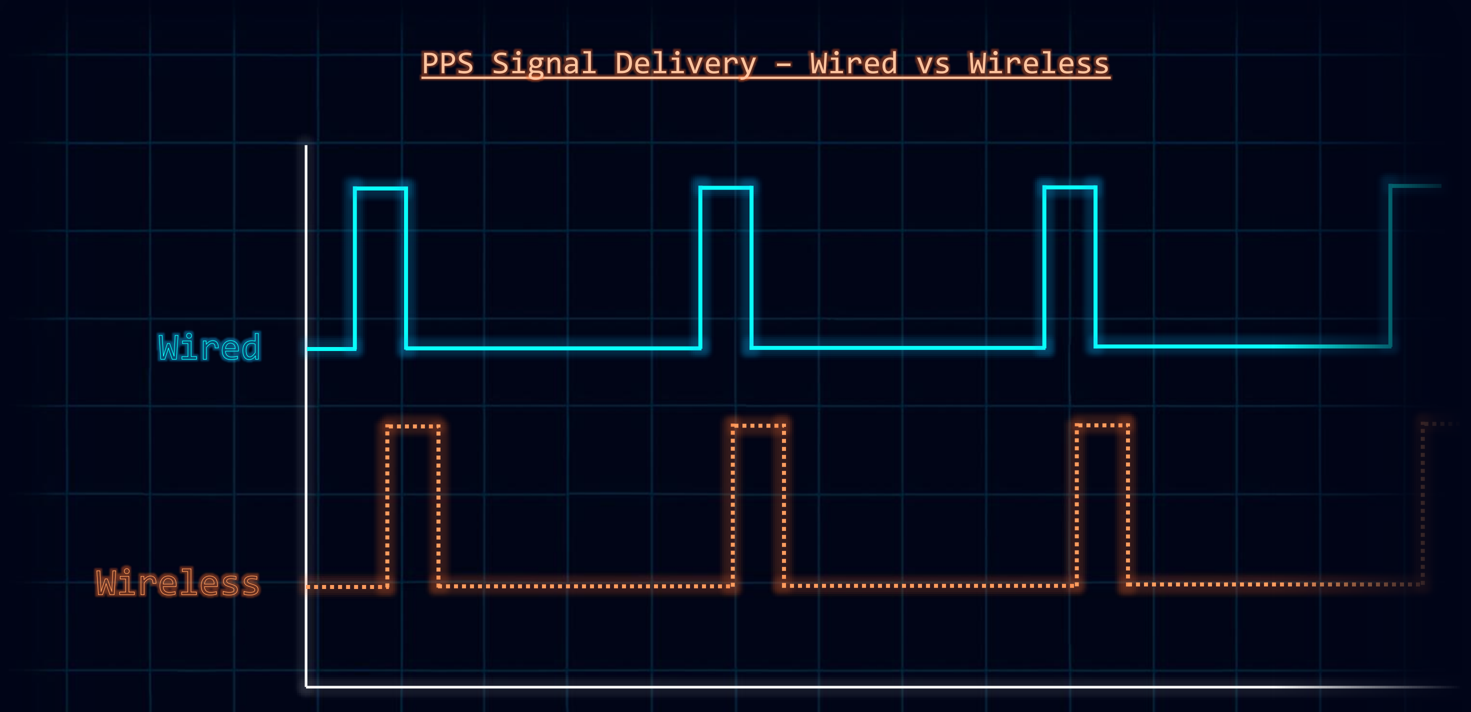


Figure 9.01 – Comparison of Wired vs. Wireless PPS Signal Delivery

Quantifying Wireless Transmission Latency

While GPS modules generate a highly accurate Pulse Per Second (PPS) signal synchronized to atomic satellite clocks, wirelessly transmitting this signal using XBee3 modules introduces latency. Unlike a direct electrical connection, the PPS signal must be detected, queued, and packetized by the XBee3 Coordinator and then transmitted to the paired XBee3 End-Device which must decode and regenerate the signal.

This entire process can result in a variable delay influenced by factors such as RF interference, signal strength, protocol overhead, and the internal processing time of the radio system. While the general shape and rising edge of the PPS signal may be preserved, its precise alignment with the true UTC second may drift slightly when delivered wirelessly as seen in the chart below.

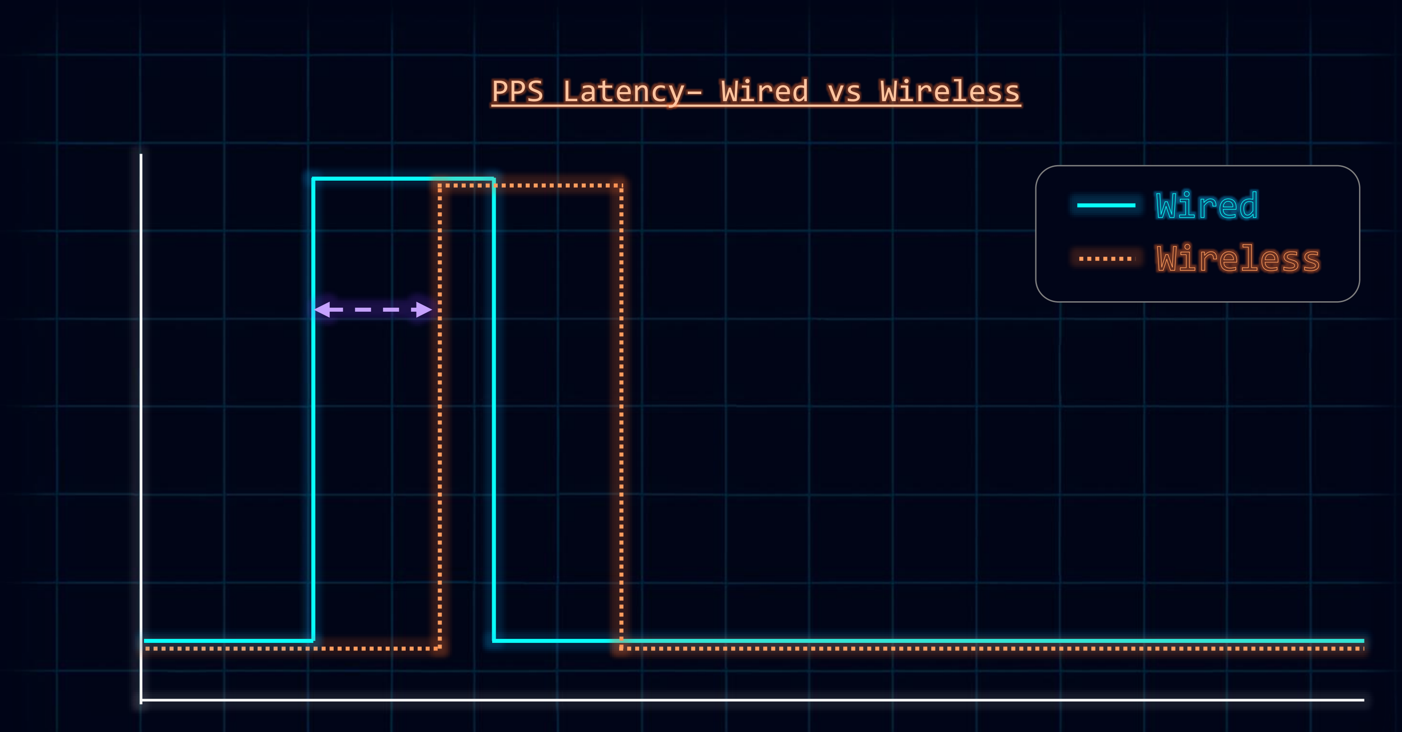


Figure 9.01 – Quantifying Latency in PPS Signals