

Programming the ATTINY 85

The instructions provided within this guide are intended and written for users who are familiar with programming microcontrollers, such as Raspberry Pi Pico or Arduino. A prerequisite requirement to this guide for programming embedded microcontrollers is that the user of these instructions should have intermediate levels of familiarity with the UNIX/Linux terminals and the C/C++ programming language.

Challenges of Embedded Microcontrollers

There are two kinds of microcontrollers: microcontrollers with boards and embedded microcontrollers. The distinction between these classes of microcontrollers is subtle, but important. Microcontrollers with boards, such as the Pico and Arduino, have serial connectivity and bootloaders that allow a direct connection to a computer, typically through USB. Embedded microcontrollers are typically too small to have a physical connector for USB. Instead, they need to connect via discrete pins to communicate with typical computers.

An additional challenge presented by the lack of a bootloader is the need for special software to program them. Other microcontrollers can be fed instructions over USB which the microcontroller can use to program itself. Embedded microcontrollers need to be communicated with in a “language” that is unique to them or their architecture. This guide will be using *avrdude* which can communicate with avr-based microcontrollers like the ATTINY family.

Structure of Instructions

These instructions will be broken into seven parts: required items, installing VScode, setting up PlatformIO, configuration of a project, writing software for the ATTINY 85, setting fuse bits, compile and upload code.

Required Items

The first step in programming the ATTINY 85 has nothing to do with the microcontroller itself. Instead, we need to gather some tools to make this process more efficient and practical.

The following items will be required before continuing:

1. A computer with a USB-A port. The computer will be needed to write software and program the ATTINY 85. USB-A is specified because the programmer does not have an option for USB-C. An adapter may be used.
2. A USB programmer such as the “Tiny AVR programmer”. This will handle the interfacing and communication between the computer and the microcontroller.



Figure 1: A USB-A port

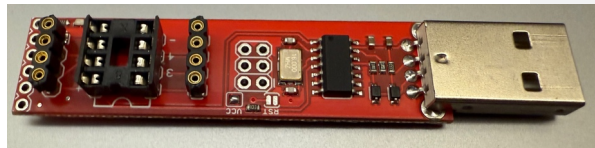


Figure 2: Tiny AVR Programmer by Digispark

3. An ATTINY 85 microcontroller. This will be taking whatever code we decide to write. The possibilities are almost limitless, if the code is under 8 kB.

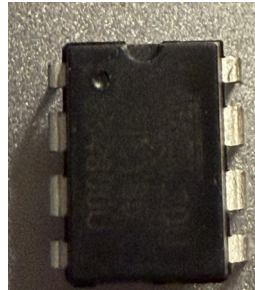


Figure 3: ATTiny85 microprocessor. The “u” notch is considered the top of the chip. Pin 1 is always the first pin to the left.

4. Visual Studio code – This will be used to write our code and an extension known as “PlatformIO” will handle all the tedious behind the scenes terminal commands through a GUI for compilation and upload of our code through the programmer.

Installing VSCode

First and foremost, we will need our software tools. First, navigate to Visual Studio Code's website: <https://code.visualstudio.com>. The website should be able to detect the operating system and provide a download for the appropriate installer. If not, click on the download button and select the appropriate option.

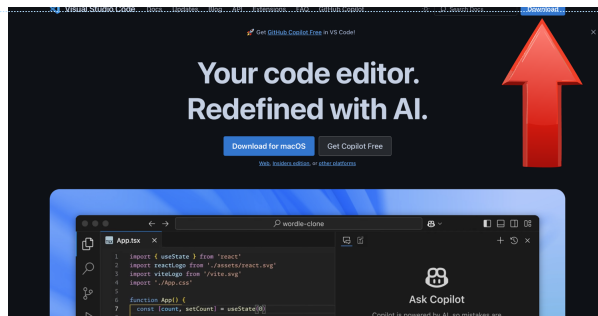


Figure 4: Location of the Download Button as of writing this document.

Commented [CH1]: Maybe center the image like you did for the PlatformIO image on the next page so its easier to see

The exact steps following this will heavily depend on your operating system of choice. This guide will assume you have the requisite knowledge of your OS. This guide will offer one suggestion: read the prompts by the installer, do not spam click through.

Setting Up PlatformIO

Visual studio code is an opensource platform. As such new functionality can be added easily through extensions. The extension you will need is PlatformIO. Once VSCode is open follow the steps below.

1. Navigate to the extensions tab on the side bar.
2. Search “PlatformIO IDE” in the search bar.
3. Click on the extension.
4. Install the extension.

The following image can be used as a guide to install the extensions. The arrows are numbered according to the instructions above.

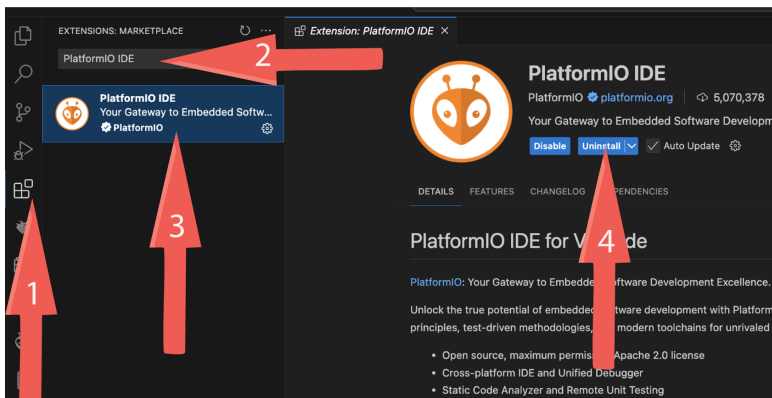


Figure 5: Numbered steps for installing PlatformIO

A fresh install of PlatformIO should not require any additional configuration. This guide will assume that the user has not used it before or has not modified the configuration. Therefore, these instructions will not delve into trouble-shooting steps if something has been previously configured.

Configuration of a Project

1. Navigate to the PlatformIO icon on the sidebar.
2. Click on “Create New Project”. This will open the PlatformIO home page.
3. Under “Quick Access” on the right side of the screen, click “New Project”.
4. Give the project a name.
5. Select the dropdown menu for “Board”. Search for “Generic ATtiny85 (Atmel)” and click on it.
6. Make sure the “Framework” is “Arduino”.
7. Click “Finish”.

Here a quick note will be made regarding frameworks. A framework is a library of code that provides abstractions to simplify code. It is possible to write “bare metal” code without a framework, but doing so is substantially more tedious, and the code must be specific and targeted for individual microprocessors. Doing so will not be covered in these instructions. More details on “bare metal” programming can be found here: https://medium.com/@bradford_hamilton/bare-metal-programming-attiny85-22be36f4e9ca.

Commented [CH2]: Maybe have them choose a directory instead of having the default selected so they know where it gets saved to

Writing Code

To write code inside of your `project`:

1. Expand the “src” directory in the “Explorer” tool bar.
2. Open the file named “main.cpp”
3. Declare functions above the setup section. Initialize variables, constants, and anything else that is needed before the main loop begins executing in the setup section. Write the part of your code that will be continually executed in the loop section. Define functions below the loop section. Please refer to the figure below as an additional aide.

Commented [CH3]: Maybe show capture of the platformio.ini file just to ensure everyone has identical settings

```
Technical Instructions > src > main.cpp > myFunction(int, int)
1  #include <Arduino.h>
2
3  // put function declarations here:
4  int myFunction(int, int);
5
6  void setup() {
7      // put your setup code here, to run once:
8      int result = myFunction(2, 3);
9  }
10
11 void loop() {
12     // put your main code here, to run repeatedly:
13 }
14
15 // put function definitions here:
16 int myFunction(int x, int y) {
17     return x + y;
18 }
```

Figure 6: Arduino “structure”

Writing code for a microprocessor is the entire purpose of these instructions. However, the code that will be written depends entirely on an individual’s purpose.

To brush up on the features, functions, and methods of the Arduino framework, users of this guide may refer to Arduino’s framework documentation. The documentation can be found here: <https://docs.arduino.cc/programming/>.

Setting Fuse Bits

The ATtiny85 has an internal 8MHz clock. Most users will find the speed to be useful, but new chips will arrive with a clock pre-scaler that increases the time each instruction takes to execute by a factor of 8. This clock pre-scaler can be changed by setting fuse bits, which are little more than bits inside of a register. Other fuse bits exist, but we won't modify those.

Some microprocessors used to have fuse bits that were literal fuses and settings could be changed once, because afterward they would "burn" the fuse. Most modern microprocessors use electronic fuse bits now, but in some applications, security bits are still set once and brick the serial pins to prevent reverse engineering of the firmware. This means we get more than one attempt to get our settings right. Typically, these are set once, and the microprocessor will remember them, however, it is useful to leave these in the main source code so the exact settings can easily be retrieved in the future.

Additionally, we will disable interrupts while writing to the fuses and then re-enable them afterwards. Interrupts are unlikely to be called during setup, but it is best practice. Once those tasks are complete, we call our main function.

More information about fuse bits and the settings they impact, can be found in the documentation for a microprocessor. The documentation for the ATtiny85 can be found here: https://ww1.microchip.com/downloads/en/devicedoc/atmel-2586-avr-8-bit-microcontroller-attiny25-attiny45-attiny85_datasheet.pdf.

To set the fuse bits:

1. Open the code you want to write to the microprocessor.
2. Write the following lines in the setup section of code.

```
void setup() {  
  // put your setup code here, to run once:  
  cli(); // Disable interrupts  
  CLKPR = (1<<CLKPCE); // Prescaler enable  
  CLKPR = 0x00; // Clock division factor  
  sei(); // Enable interrupts  
  start();  
}
```

Commented [CH4]: I don't really have attiny experience besides making an LED blink. Had no idea about the clock speed stuff. So this is new to me. Personally I'm usually interested in what the abbreviations stand for. So maybe include CLKPR = Clock Prescaler Register, CLKPCE = Clock Prescaler Change Enable, etc. Then maybe provided a little beeper comments. Something like "Writing 1 to CLKPCE allows us to modify the prescaler in the next instruction" and "Here, we set the actual prescaler value to 0 (no division), meaning the attiny will run at the full 8 MHz"

3. Save the file using the typical keyboard shortcut for your platform, or from the “File” dropdown menu.

Compile and Upload Code

1. Connect the USB programmer to your computer.
2. Insert the ATtiny85 into the programmer. Be sure to align the “u” notch on the microprocessor with the “u” notch of the chip carriage. Failure to do so may result in a dead microprocessor and in extreme circumstances, a fire.

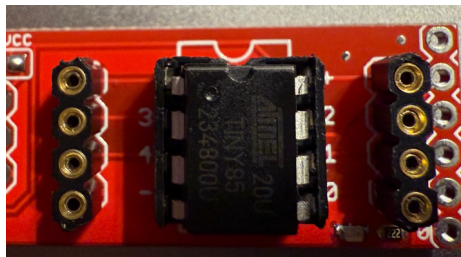




Figure 7: Microprocessor in programmer. Note the alignment of the “u” notches

3. Click the build  button on the bottom toolbar. A message should appear in the console indicating a successful compilation.
4. Click the upload  button. Another success message should appear.
5. Insert the ATtiny85 into your circuit and verify it is operating as intended.