

# Lab #0

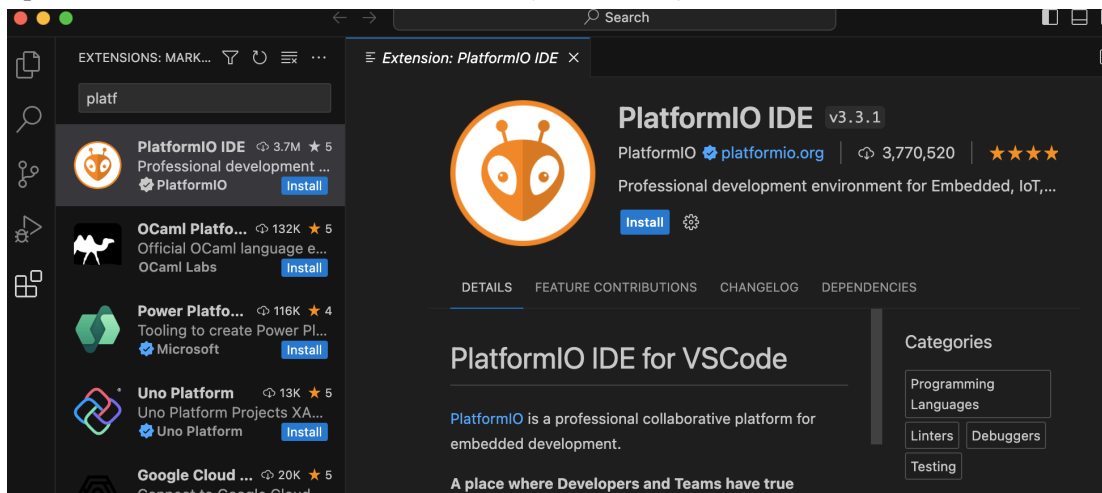
## ECE167: Sensing and Sensor Technologies University of California Santa Cruz

This lab is intended to guide you through setting up the development environment for the hardware and microcontroller you will be using for the rest of the quarter. The microcontroller, or MCU, is an ARM-based STM32. This lab guides you through how to install the required development software onto your own computer. The software environment on the computers in the lab rooms is Windows, but the toolchain for this class should work on any platform. *NOTE: the images in this guide differ from what you see depending on your OS.*

### Part 1: Installing the Software

The development environment we will use in this class is PlatformIO which is available as a plugin for almost all code editors. We will show how to install PlatformIO on top of VSCode but feel free to use your editor of preference. Here are the install instructions:

- Install [VSCode](#) on your machine, or follow the right set up guide for your [editor of choice](#)
- Open VSCode and under the *Extensions* tab (Ctrl-Shift-X) search for PlatformIO and install



- Clone your ECE167 git repository from gitlab (see image on next page). Your 'repo' is automatically generated<sup>1</sup>, and can be found at `git.ucsc.edu/ece167_master/winter-2024-stm32-pilot/<YOUR_UCSC_USERNAME>`

---

<sup>1</sup> If an ECE167 git repository was not automatically generated for you, please contact the course staff

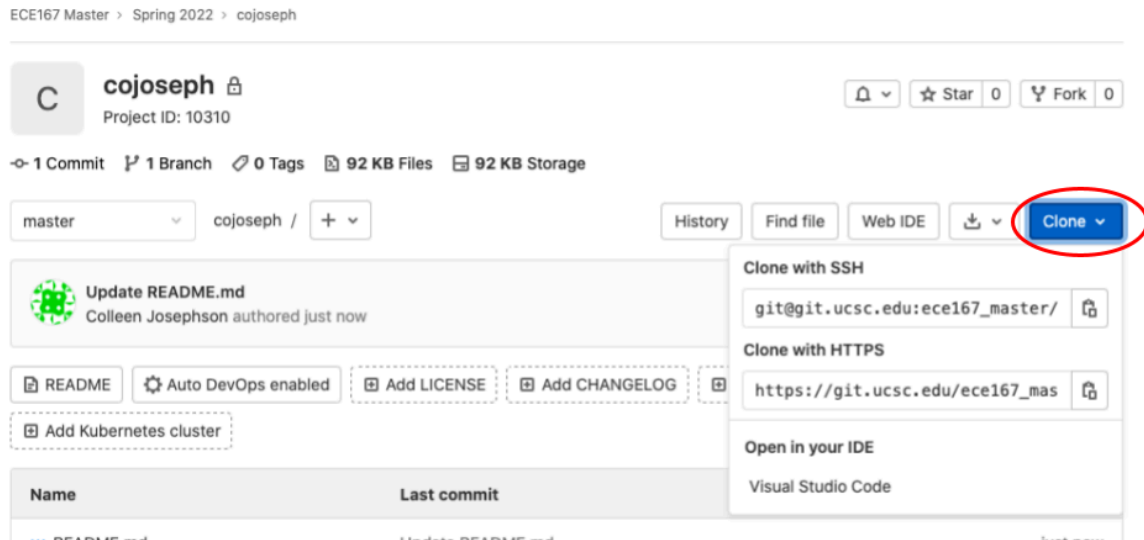


Figure: Cloning your ECE167 repository

## Part 2: Hello World

### Open PlatformIO and Create a New Project



Launch VSCode and select the PlatformIO icon on the left. Select *Create New Project* which will open a PIO Home tab.

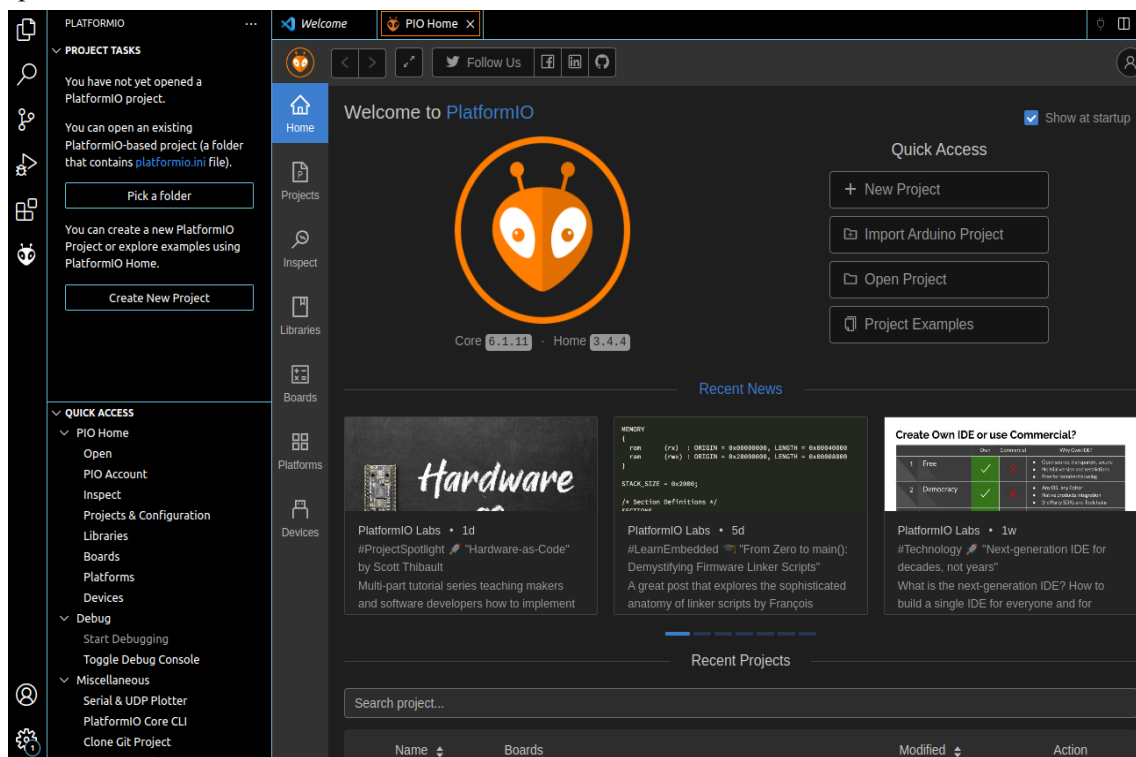


Figure 1: PIO Home tab


Next, press + *New Project*. This will open a project wizard window. Name the project *Lab0*, select **ST Nucleo F411RE** in the board selection, and choose **STM32Cube** for the framework selection. Lastly, uncheck the “use default location” box and navigate to the Lab0 directory in your git repository. Press finish and your blank project will be generated (this may take a minute). Next, open the default *platformio.ini* file and replace the contents of this file with the following block.

```
[env:nucleo_f411re]
platform = ststm32
board = nucleo_f411re
framework = stm32cube
lib_deps = ../../Common
lib_archive = no
```

All the code you write for each lab goes into the src directory within your platformIO project. Create a new file in the src directory through the command line, or by right clicking the src folder on the left side bar and selecting “**New File...**” Call it *main.c* and add the following code block:

```
#include <stdio.h>
#include <stdlib.h>
#include <Board.h>

int main(void) {
    BOARD_Init();
    while (TRUE) {
        printf("Hello World\r\n");
    }
}
```

To compile the project, extend a drop down menu by clicking on the down-facing symbol next to the right-facing arrow in the top right of the screen  and select **Build** or simply use alt+ctrl+b. A successful compilation terminal output should resemble the following.

```

Processing nucleo_f411re (platform: ststm32; board: genericSTM32F411RE; framework: stm32cube)
-----
-----
Verbose mode can be enabled via `-v, --verbose` option
CONFIGURATION: https://docs.platformio.org/page/boards/ststm32/genericSTM32F411RE.html
PLATFORM: ST STM32 (16.1.0) > STM32F411RE (128k RAM. 512k Flash)
HARDWARE: STM32F411RET6 100MHz, 128KB RAM, 512KB Flash
DEBUG: Current (blackmagic) External (blackmagic, jlink, stlink)
PACKAGES:
  - framework-stm32cubef4 @ 1.26.2
  - tool-ldscripts-ststm32 @ 0.2.0
  - toolchain-gccarmnoneabi @ 1.70201.0 (7.2.1)
LDF: Library Dependency Finder -> https://bit.ly/configure-pio-ldf
LDF Modes: Finder ~ chain, Compatibility ~ soft
Found 53 compatible libraries
Scanning dependencies...
Dependency Graph
|-- Common @ 0.0.0+20231026131144
Building in release mode
Compiling .pio/build/nucleo_f411re/src/main.o
Checking size .pio/build/nucleo_f411re/firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM:      [          ]    0.3% (used 388 bytes from 131072 bytes)
Flash:    [          ]    1.5% (used 7648 bytes from 524288 bytes)
===== [SUCCESS] Took 1.18 seconds
=====

```

Next, plug the board into your computer and make sure that some of the status LEDs turn on. Then upload the firmware to the microcontroller by using the same drop down menu and selecting **Upload** or using the shortcut alt-ctrl-u. Below is an example of a successful firmware upload.

```

Processing nucleo_f411re (platform: ststm32; board: genericSTM32F411RE; framework: stm32cube)
-----
-----
Verbose mode can be enabled via `-v, --verbose` option
CONFIGURATION: https://docs.platformio.org/page/boards/ststm32/genericSTM32F411RE.html
PLATFORM: ST STM32 (16.1.0) > STM32F411RE (128k RAM. 512k Flash)
HARDWARE: STM32F411RET6 100MHz, 128KB RAM, 512KB Flash
DEBUG: Current (blackmagic) External (blackmagic, jlink, stlink)
PACKAGES:
  - framework-stm32cubef4 @ 1.26.2
  - tool-dfuutil @ 1.11.0
  - tool-ldscripts-ststm32 @ 0.2.0
  - tool-openocd @ 3.1200.0 (12.0)
  - tool-stm32duino @ 1.0.1
  - toolchain-gccarmnoneabi @ 1.70201.0 (7.2.1)
LDF: Library Dependency Finder -> https://bit.ly/configure-pio-ldf
LDF Modes: Finder ~ chain, Compatibility ~ soft
Found 53 compatible libraries
Scanning dependencies...
Dependency Graph
|-- Common @ 0.0.0+20231026131144
Building in release mode
Checking size .pio/build/nucleo_f411re/firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM:      [          ]    0.3% (used 388 bytes from 131072 bytes)
Flash:    [          ]    1.5% (used 7648 bytes from 524288 bytes)
Configuring upload protocol...
AVAILABLE: blackmagic, dfu, jlink, serial, stlink

```

```

CURRENT: upload_protocol = stlink
Uploading .pio/build/nucleo_f411re/firmware.elf
xPack Open On-Chip Debugger 0.12.0-01004-g9ea7f3d64-dirty (2023-01-30-15:03)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
debug_level: 1

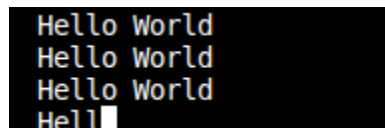
hla_swd
[stm32f4x.cpu] halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x08001e10 msp: 0x20020000
** Programming Started **
** Programming Finished **
** Verify Started **
** Verified OK **
** Resetting Target **
shutdown command invoked
===== [SUCCESS] Took 1.81 seconds
=====

```

To test if our firmware works, we need to open a serial terminal with the NUCLEO board. With the board still connected, use a program like screen (or PuTTY for Windows) to open a serial session at 115200 baud. Here is an example Linux command that accomplishes this:

```
$ screen /dev/<your device name> 115200
```

If everything works, your screen should spam with “Hello World” messages.



## Part 3: Testing the Hardware

Now that you have confirmed that your programming environment is up and running, the rest of this lab will have far less detailed instructions. For the remainder of the lab, and in the remainder of this course, you are expected to read the code, look at datasheets, and examine other typical resources to try and figure things out on your own.

### Hello A/D (Reading an analog signal)

Print the A/D reading from the potentiometer on the microcontroller shield using the AD library. One AD pin in particular corresponds to the onboard potentiometer. You can check the IO shield datasheet or schematic (both are on Canvas) to figure out which one it is.

### Simple Tone

Generate a tone with the speaker using the pwm library in the Common directory, [pwm.c/h]. For this part of the lab, you will need the speaker and the sparkfun audio amplifier that comes with your lab kit. You probably also want a breadboard, which can be purchased at BELS if you do not already have one. First, connect a PWM pin from the microcontroller (specifically a PWM pin which this library controls)

to the audio amplifier PCB. Make sure to use a jumper cable (solid core wire), not stranded wire. Solid core wire is the correct diameter for the board, whereas stranded wire will break in the breadboard. Mess around with the duty cycle and frequency of the tone, how does each affect the sound?

### Tone adjusted via potentiometer

Now use the potentiometer A/D reading to change the tone of the speaker. Keep in mind that changing the tone means changing the frequency of the PWM signal. You will notice that changing the frequency with the on-board potentiometer sounds really scratchy and bad. Why? Implement some software filtering to eliminate the scratchy sound. Explain in your lab report what methods you used. You will want to try some different ideas, but be sure to write about what worked and what did not in your report.

*Hint: The potentiometer is probably not linear. You might want to find a polynomial fit, and code it as a function of the potentiometer reading to really linearize it. This can be done using many techniques which we will go over in detail down the road. Alternatively, read up on FIR and IIR filters.*

### Basic filtering of output (single pole RC to smooth tone generation)

By now you may have noticed that the sound from the speaker is not very "smooth." To fix this, you will need to use a low-pass filter to smooth the signal coming from the MCU and going into the audio amplifier. How does the filter change the sound of the speaker? What is the cutoff frequency for the low-pass filter, and why? How does the low-pass change the square wave? Make sure you include the circuit diagram to your solution in the lab report.

### Make some music

Build an instrument to play music with your now harmonious assembly of microcontroller, low-pass filter, audio amp, and speaker. The requirements for the instrument are:

- Map four different tones to the four buttons on the I/O Shield. There are tones provided in the header file but feel free to use your own.
- Read the potentiometer (with appropriate filtering) and use the value as an offset to the tones provided. This means that while the difference between tones will remain the same, the average frequency changes. Experiment with the offset scaling until you find one that makes sense.
- While one of the buttons is held down, that tone will be played through the speaker. Releasing the button should stop the tone. You do not need to worry about what happens when more than one button is pressed at the same time.

### Check-Off and Lab Report

Demonstrate your fully functioning instrument to a TA or tutor. Congratulations! By this point you have completed the first lab! Now you must write your lab report. An example lab report can be found on [Canvas](#).

The goal of a lab report is to ensure that a student who has already taken the class and is fairly well versed in embedded software would be able to recreate your solution. Please be clear and concise, use complete sentences, and maintain a formal voice in your writing. Include the following sections in your report:

1. Title Page (*be sure to include your name, the lab number, the date, the git commit ID, the name of the TA who checked you off as well as the date/time, and also mention any collaborators you worked with*)
  - a. **IMPORTANT:** always include your git commit ID. Without this we cannot grade you.
2. Overview (*describe the goals of the assignment, and include a list of all the parts used in the lab,*)
3. Wiring and Diagrams (*[fritizing](#) is one way to generate high-quality breadboard diagrams*)
4. Project Design (*one section for each section of the lab, be sure to answer any questions posed in the lab instructions; use pseudocode, diagrams and other visuals liberally*)
  - a. Section 1
  - b. Section 2
  - c. Section 3
    - i. Section 3.1
    - ...
    - ii. Section 3.5
5. Testing and Results (*how did you test that your solution achieved the aims; what equipment or code did you use to debug?*)
6. Conclusion (*a brief paragraph telling us how long the lab took you, and any final thoughts*)

## Lab 0 Rubric

- 40% check off (10 point scale)
  - Hello world: 3 points
  - Buttons play 4 different tones: 4 points
  - Button release stops tone: 3 points
- 15% code quality (comments, readability) [Reader]
- 15% code compiles [Reader]
- 30% lab report [Reader]
  - General format followed; all sections of lab addressed (15 points)
  - Writing detailed enough for replication (10 points)
  - Image quality and writing quality (grammar, spelling, appropriate tone) (5 points)