|  |  |
| --- | --- |
| SETUP PRELIMINARY | |
| A screenshot of a cell phone  Description automatically generated | |
| 6/23/2020Category: Biomedical Devices | Ameen SaleminikDS00001 |

This document is for internal use only and is intended for ALI DJABARRI and other related correspondents.

## Table of Contents

1. Overview

1.1 Introduction

1.2 Specifications

1.3 Setup Diagrams

1.4 Legal Disclaimer/Additional Information

2. Embedded Systems Setup

3. iOS application setup (to be added)

## Overview

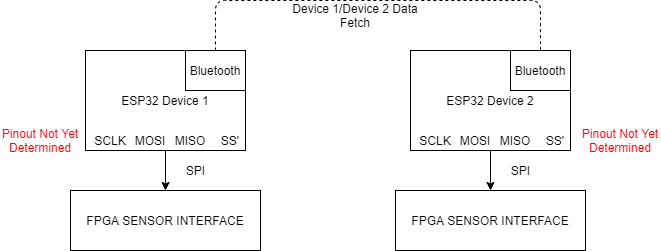
* 1. Introduction

This report explains the integration of two ESP32 microcontrollers with an already existing sensor measurement system. One microcontroller queries for input from a sensor and performs a calculations. Then, a second microcontroller retrieves that calculation over Bluetooth, and applies an appropriate action. One example is measuring blood-glucose levels, then controlling a pump to administer the proper dosage. The application of the entire device is for general biomedical purposes. However, this document only refers to the top level, as shown in the system diagrams below.

* 1. Specifications

This project uses two ESP32 (WROOM-32) to communicate over SPI and BluetTooth. Features of the ESP32 include the following

* Processors:
  + CPU: Xtensa dual-core 32-bit LX6 microprocessor, operating at 160 MHz and performing at up to 600 DMIPS
  + Ultra low power (ULP) co-processor
* Memory: 520 KiB SRAM
* Wireless connectivity:
  + Wi-Fi: 802.11 b/g/n
  + Bluetooth: v4.2 BR/EDR and BLE (shares the radio with Wi-Fi)
* Peripheral interfaces:
  + 12-bit SAR ADC up to 18 channels
  + 2 × 8-bit DACs
  + 10 × touch sensors (capacitive sensing GPIOs)
  + 4 × SPI
  + 2 × I²S interfaces
  + 2 × I²C interfaces
  + 3 × UART
  + SD/SDIO/CE-ATA/MMC/eMMC host controller
  + SDIO/SPI slave controller
  + Ethernet MAC interface with dedicated DMA and IEEE 1588 Precision Time Protocol support
  + CAN bus 2.0
  + Infrared remote controller (TX/RX, up to 8 channels)
  + Motor PWM
  + LED PWM (up to 16 channels)
  + Hall effect sensor
  + Ultra low power analog pre-amplifier
* Security:
  + IEEE 802.11 standard security features all supported, including WFA, WPA/WPA2 and WAPI
  + Secure boot
  + Flash encryption
  + 1024-bit OTP, up to 768-bit for customers
  + Cryptographic hardware acceleration: AES, SHA-2, RSA, elliptic curve cryptography (ECC), random number generator (RNG)
* Power management:
  + Internal low-dropout regulator
  + Individual power domain for RTC
  + 5μA deep sleep current
  + Wake up from GPIO interrupt, timer, ADC measurements, capacitive touch sensor interrupt
  1. System Diagram



*Figure 1, general block diagram for the initial high-level system.*

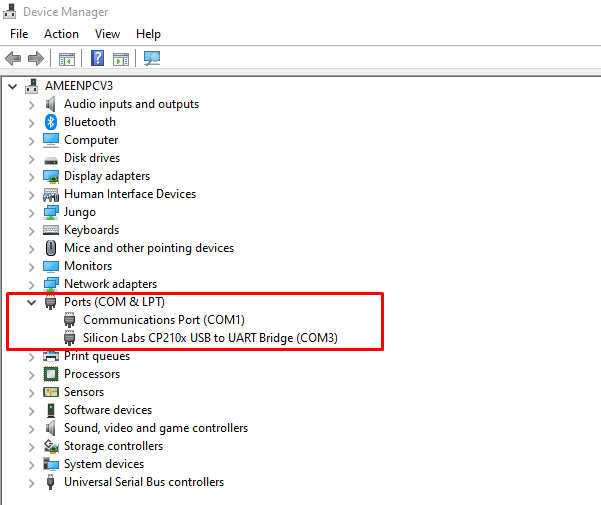
* 1. Legal Disclaimer/Additional Information

Most software running on the ESP32 is under the Apache 2.0 License which allows for commercial use. In the final implementation, all running software should be checked for copyright infringement. As it stands right now, there is no concern. Additionally, the ESP32 with is FCC approved for its already existing firmware, for wireless modules like WIFI and Bluetooth.. However, since we a changing the firmware for our own purposes, we need to get our own FCC approval to use it for our device.

2. Embedded Systems Setup

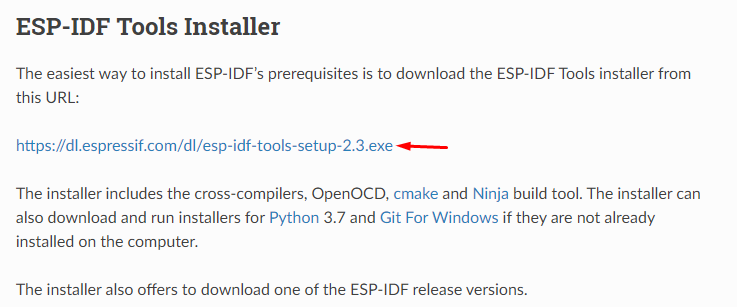
First, extract the code in the .zip file I sent you to somewhere you can easily access it. You will need this later.

Then, plug in the ESP32 into the computer. Open Device Manager and open Ports(COM & LPT). The Silicon Labs USB to UART Bridge should show up under a COM port (COM3 for me). Take note of the number because you also will need this.

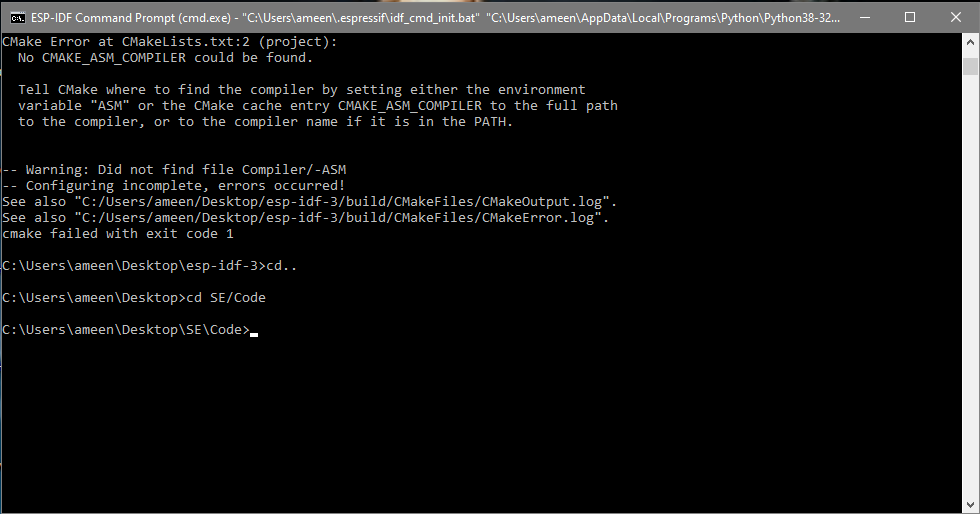


Go to this website and download the ESP-IDF Tools installer

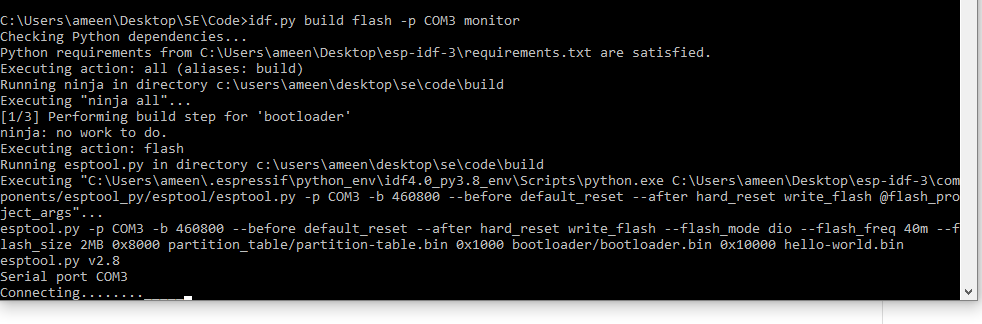
<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/windows-setup.html>



Open the file and let the Installer run. Once done, Open esp-idf command prompt (hit the windows key and type esp-idf, it should pop up), and navigate to where the hello\_world.c code is. For me, it is on my desktop in a folder.



Run idf.py build flash -p COM3 monitor. Change your COM port to what your computer has it as. **Make sure to hold the BOOT button on the ESP32 before running this command. Otherwise, you will get a Timed out waiting for packet header error during the flash stage. You can let go of the boot button once it has finished flashing.**



If everything was correct, you should see this:

