Guide to set up the Pico SDK on windows machine, then compile and flash the hello\_usb.c example to QT Py board

**Prepared by: Sahil Mahendra Mangaonkar**

Resources Information:

1. Terminal: Windows Subsystem for Linux (Ubuntu)1
2. Serial console: Putty
3. System Model: HP ProBook 650 G1
4. Operating system: Microsoft Windows 10 pro
5. System Type: x64-based PC
6. Processor: Intel(R) Core(TM) i5-4300M CPU @ 2.60GHz, 2601 Mhz, 2 Core(s), 4 Logical Processor(s)

1Historically, terminal support is much easier in UNIX-like environments like Linux or macOS.

**Step 1: Installing pico-sdk and pico-examples from GitHub.**

In this step, we’ll install the SDK and examples from GitHub into our local machine.

1. Go to default folder using:

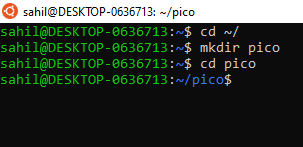
$ cd ~/

1. Make new directory:

$ mkdir pico

1. Get inside the newly created directory:

$ cd pico



Now we’ll clone the pico-sdk and pico-examples git repositories.

1. Clone pico-sdk from git repository:

$ git clone -b master https://github.com/raspberrypi/pico-sdk.git

1. Get inside the pico-sdk folder:

$ cd pico-sdk

1. If we already cloned the project and forgot --recurse-submodules, then we can combine the git submodule init and git submodule update steps by running following command:

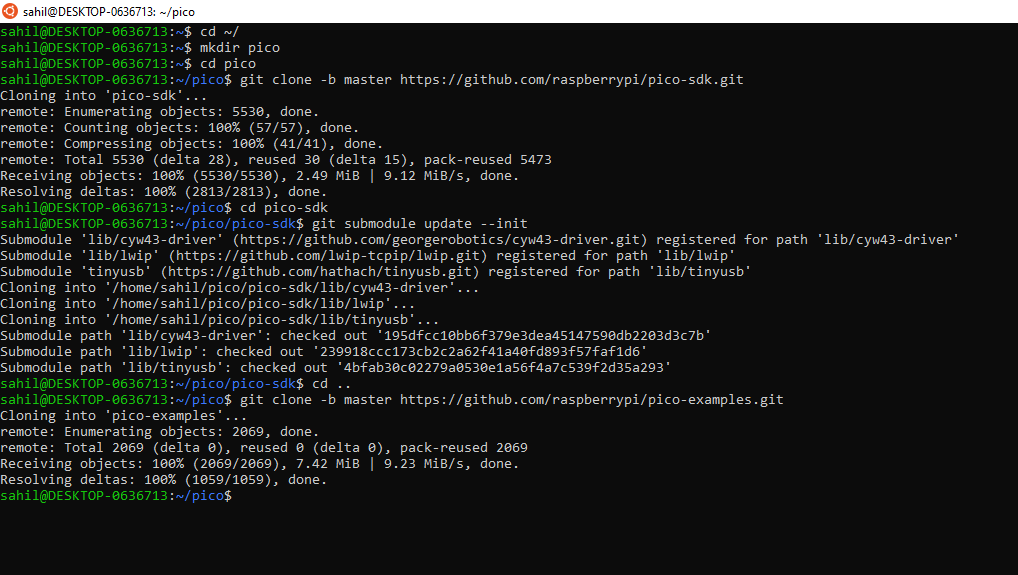
$ git submodule update --init

1. Go back to pico folder:

$ cd ..

1. Clone pico examples from GitHub:

$ git clone -b master https://github.com/raspberrypi/pico-examples.git



Here we have successfully cloned the pico-sdk and pico-examples git repositories into the directory pico.

**Step 2: Install the Toolchain.**

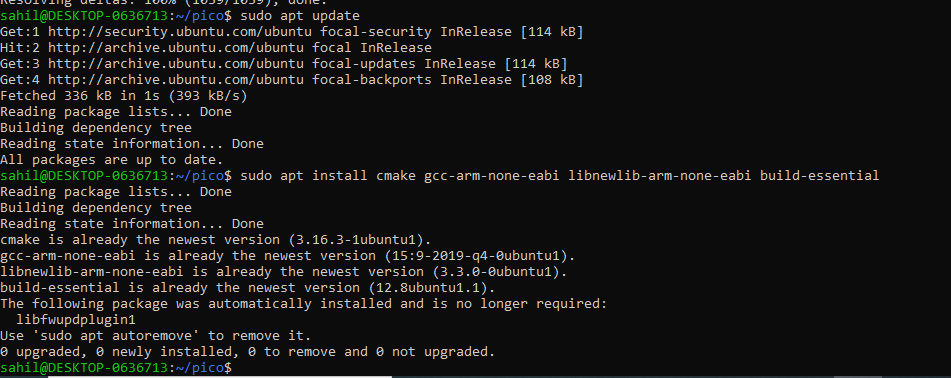
To build the applications in pico-examples, we’ll need to install some extra tools. To build projects we’ll need CMake, a cross-platform tool used to build the software, and the GNU Embedded Toolchain for Arm.

1. Use following command**to download and update the package information from all of the configured sources**:

$ sudo apt update

1. Install CMake and GNU Embedded Toolchain for Arm using following command:

$ sudo apt install cmake gcc-arm-none-eabi libnewlib-arm-none-eabi build-essential



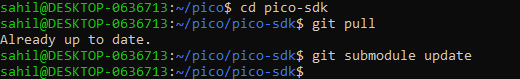
1. Update the SDK:

When a new version of the SDK is released we will need to update we copy of the SDK. To do this, go into the pico-sdk directory which contains our copy of the SDK, and do the following:

$ cd pico-sdk

$ git pull

$ git submodule update



**Step 3: Build program**

In this step we’ll build our program that we want to load into QT Py. From the pico directory we created earlier, cd into pico-examples and create a build directory.

1. Go back to pico folder:

$ cd ..

1. Enter into pico-example directory:

$ cd pico-examples

1. Make new folder build in pico-examples:

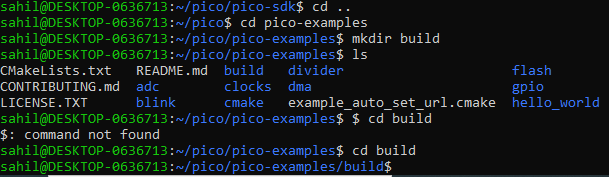
$ mkdir build

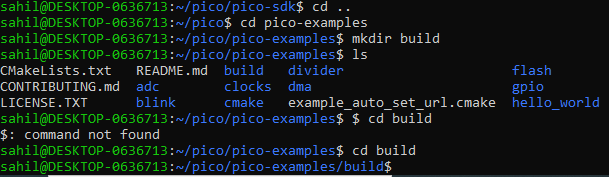
1. Check if the build folder is created:

$ ls

1. Enter into build:

$ cd build





**Step 4: Setting the PICO\_SDK\_PATH and preparing our cmake build directory**

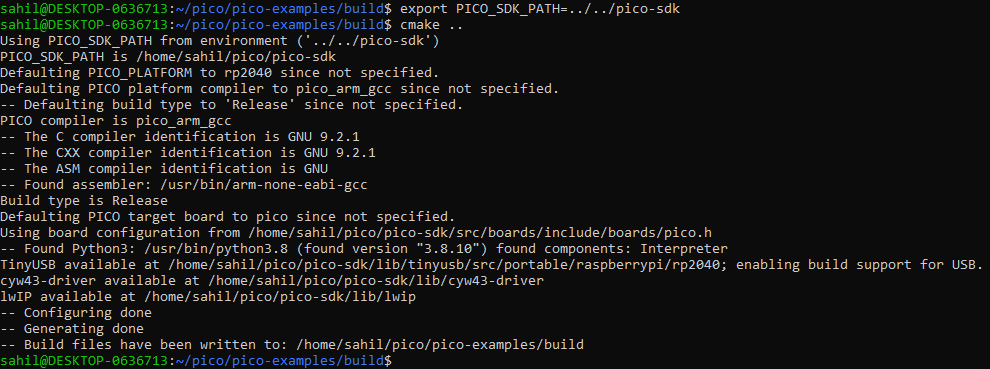
In this step we’ll set PICO\_SDK\_PATH and preparing our cmake build directory. We have cloned the pico-sdk and pico-examples repositories into the same directory i.e. pico side-by-side.

1. Set the PICO\_SDK\_PATH:

$ export PICO\_SDK\_PATH=../../pico-sdk

1. Prepare cmake build directory by running cmake ..

$ cmake ..

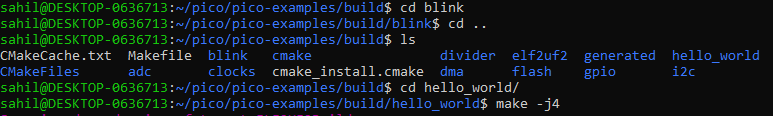


1. Get into hello\_world folder where our example is:

$ cd hello\_world/

1. We’ll use “make” command with -j4 will run four make jobs in parallel to speed it up. My laptop has 4 cores. So -j4 is a reasonable number.

$ make -j4



1. Check contents of hello\_world folder:

$ ls



There will be two separate examples programs in the “hello\_world/serial/” and “hello\_world/usb/” directories inside hello\_world.

Amongst other targets, we have now built:

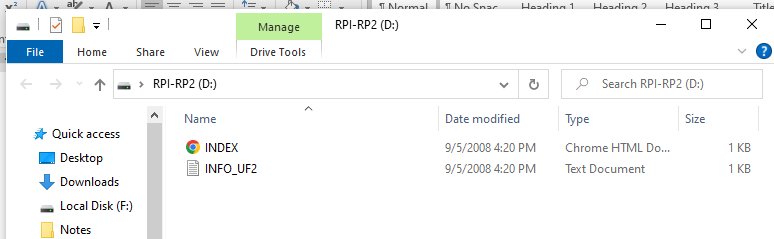
* serial/hello\_serial.elf - which is used by the debugger
* serial/hello\_serial.uf2 - which can be dragged onto the RP2040 USB Mass Storage Device (UART serial binary)
* usb/hello\_usb.elf - which is used by the debugger
* usb/hello\_usb.uf2 - which can be dragged onto the RP2040 USB Mass Storage Device (USB serial binary)

**Step 5: Upload the code**

Now we’ll put our QT Py board into bootloader mode by connecting it to usb port while keeping the boot button pressed and release the button after connecting it to usb port.

This will open a folder named RPI-RP2 with 2 files:

* INDEX.HTM
* INFO\_UF2.TXT



*Note: Don’t use any commands related to mass storage mounting or uploading code on the board directly from Ubuntu terminal such as:*

* *sudo blkid -o list | grep RPI-RP2*

*or*

* *sudo ./picotool save -p blinky\_led -t uf2*

*or*

* *3.2.2. Using the command line of the guide.*

*We aren’t using Ubuntu systems. We just have an Ubuntu terminal using WSL. The Qt py will show up in windows.*

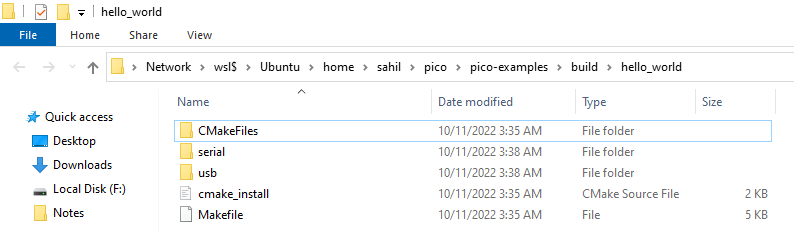
We’ll use a simple trick to make this easy drag and drop.

**Step 6: Loading files into QT Py**

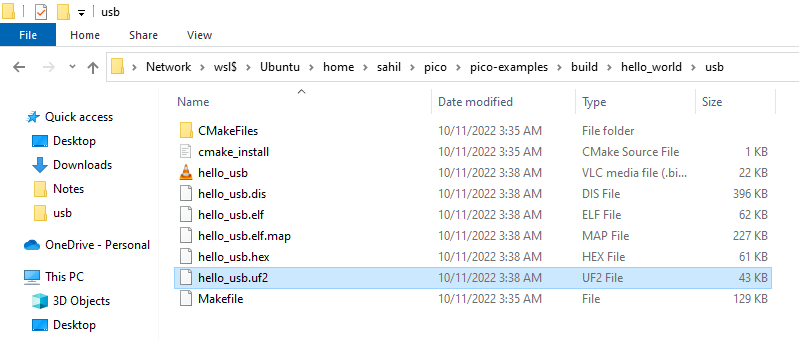
1. Type following command on the terminal:

$ explorer.exe .

This will launch file explorer showing the current Linux directory. We can browse the Linux environment’s file system from there.

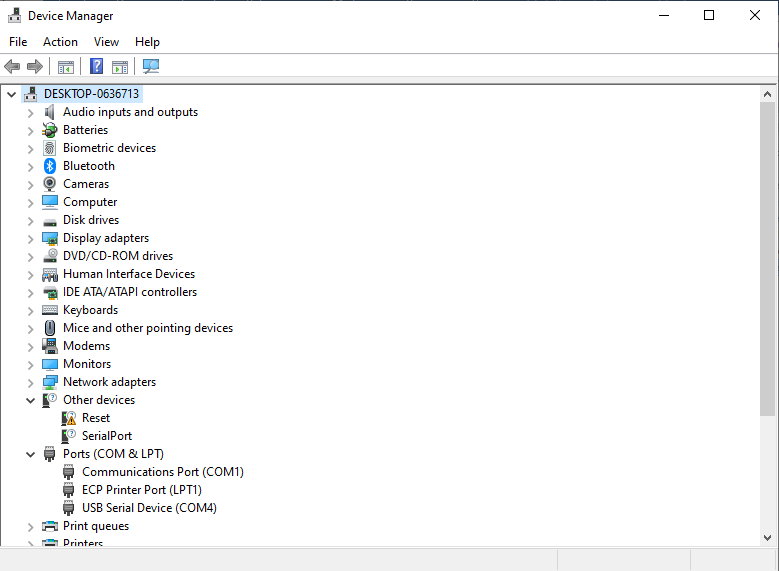


1. We’ll go to usb folder, will copy hello\_usb.uf2 and paste into RPI-RP2 folder:



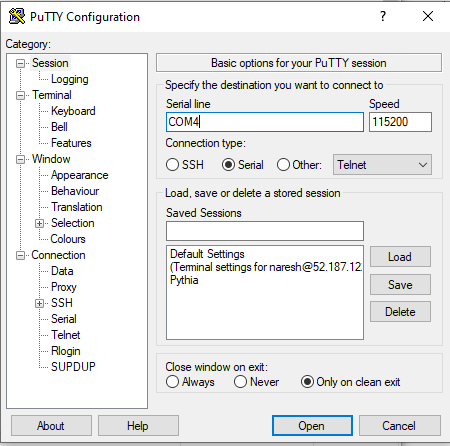
*Note: The QT Py will disappear after copying “.uf2” file into the RPI-RP2 folder. But Don’t fear it’s normal behavior.*

1. Go to device manager and check for a new com port in ports:



So the QT Py device is connected to com port 4.

1. Open Putty which we’ll be using as our serial console. Select Serial. Type COM port address in Serial line and enter speed as 115200.



1. Then click on open button:



-----------------------------------------The End of the Guide-----------------------------------------