



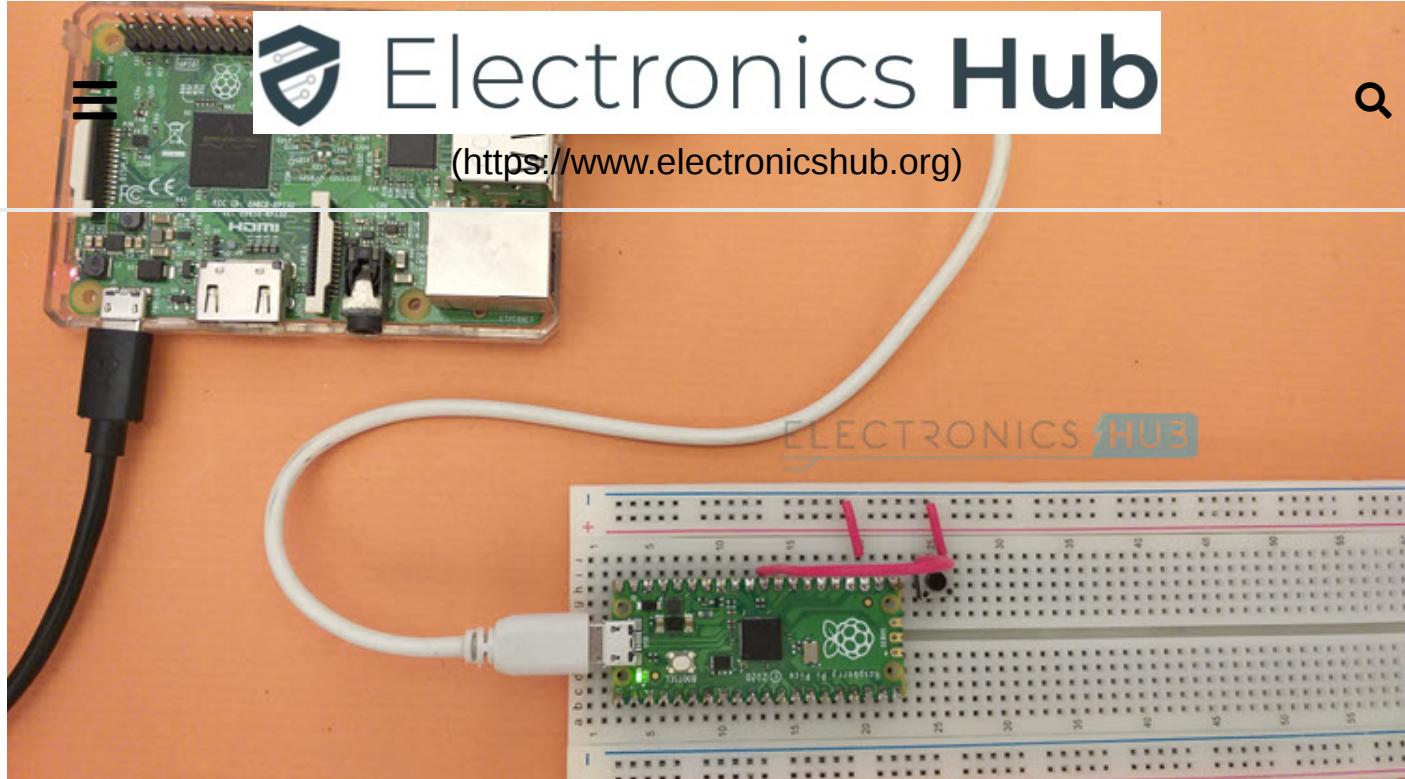
Home (<https://www.electronicshub.org>) →

General (<https://www.electronicshub.org/general/>), Learn (<https://www.electronicshub.org/learn/>)

# Programming Raspberry Pi Pico using C | Getting Started with C SDK

March 5, 2021 By Ravi Teja (Embedded Engineer) (<https://www.electronicshub.org/author/raviteja/>)

In this tutorial, we will learn how to Program Raspberry Pi Pico using C. We will see all the steps required for Programming Raspberry Pi Pico in C Programming Language like downloading Raspberry Pi Pico's C SDK and examples from github, installing all the necessary tools, building the Blinky C Program and finally loading the UF2 file to Pico and running the Blinky example.



<https://www.electronicshub.org/wp-content/uploads/2021/03/Program-Raspberry-Pi-Pico-using-C-Image.jpg>

## Outline



Have You Read These?

Getting Started with Raspberry Pi Pico C SDK

What Computer to use?

Get Pico's C SDK and Examples

    Download C SDK

    Download Examples

Installing the Toolchain

Exploring Blink Example

Build the Blink

Load the Blink and Run it

Bonus: Alternative to Unplugging and Plugging USB Cable

Conclusion

## Have You Read These?

Before proceed  
for Raspberry P



# Electronics Hub

that I made



(<https://www.electronicshub.org/getting-started-with-raspberry-pi-pico/>) and

'Programming Raspberry Pi Pico with MicroPython (<https://www.electronicshub.org/raspberry-pi-pico-micropython-tutorial/>)'.

These tutorials include some basics about Raspberry Pi Pico, the RP2040 Microcontroller, Pinout of Pico, programming environment, MicroPython and many other important topics.

I strongly recommend you to complete those two tutorials to get familiar with Raspberry Pi Pico.

## Getting Started with Raspberry Pi Pico C SDK

If you are an old school hardware engineer (like I am) and has experience in writing firmware, then you will probably be more comfortable in C Programming Language rather than Python or MicroPython for writing applications / firmware. Ironically, MicroPython is written in C.

Even though the Raspberry Pi Foundation promotes Python as the primary programming language for all the Raspberry Pi SBCs, things are quite different when it comes to Microcontrollers.

I consider C as the low-level programming among all the high-level programming languages. If you want complete control of the hardware right down to the individual bit of a register, then C is ultimate choice. Hence, C is the preferred choice of programming for many microcontrollers and embedded systems.

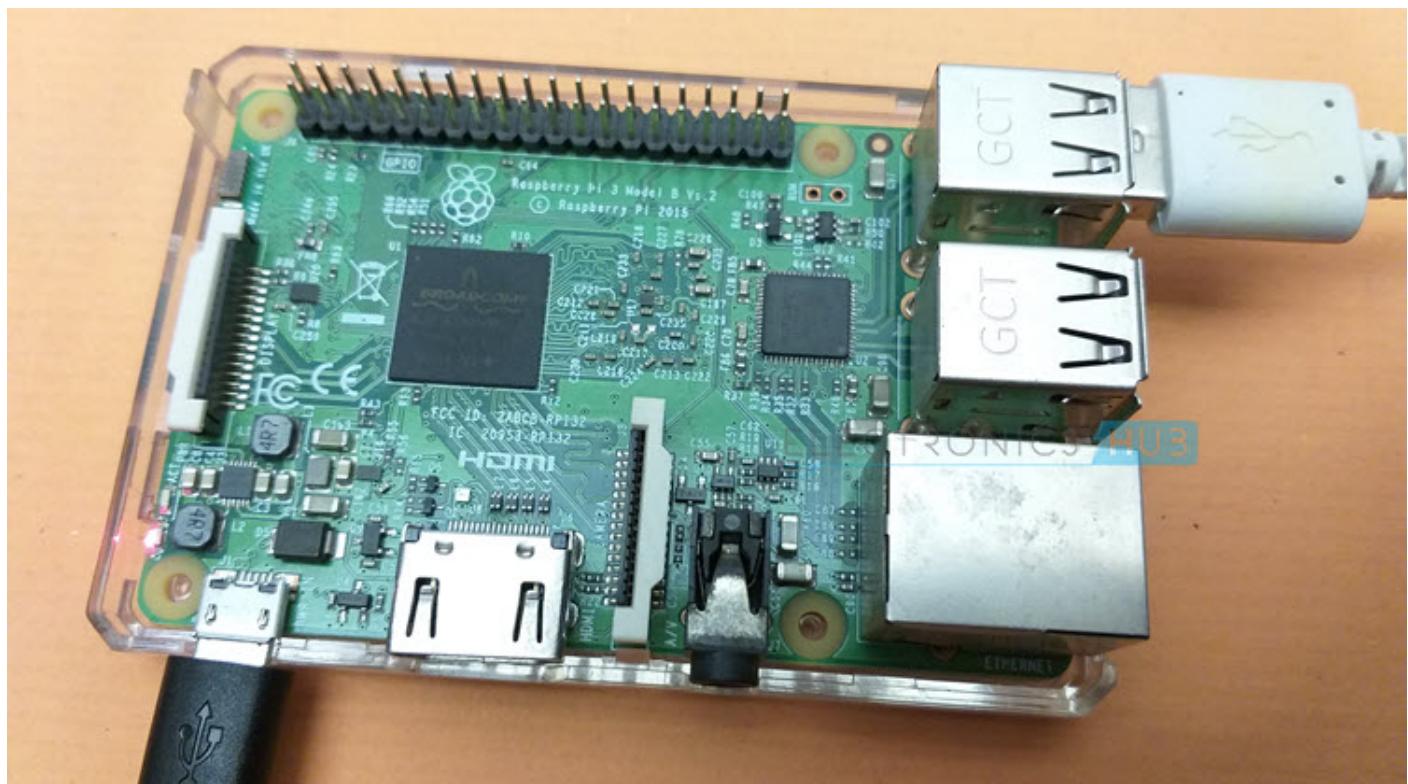
The RP2040 Microcontroller based Raspberry Pi Pico is no exception to this. While

investing and pi  
(Software Dev)  Electronics Hub released SDK  
for RP2040 based boards. (<https://www.electronicshub.org>)

Let us see how to Program Raspberry Pi Pico using C Programming Language.

## What Computer to use?

I already discussed this in the ‘Programming Pico with MicroPython’ tutorial. You can use any of the Linux, Windows or Mac based systems as the host computer. But the official documentation of Raspberry Pi Pico focuses on Raspberry Pi Computer running Raspberry Pi OS as the main host system (even though it mentions procedures for other systems).



(<https://www.electronicshub.org/wp-content/uploads/2021/03/Raspberry-Pi-3-Model-B-for-Pico.jpg>)

I do not have the time to work on this project right now, but I will be working on it in the future. I will be using a Raspberry Pi 3 Model B. So, in the future tutorials, I will be showing how to program the Pico in different environments like Windows, Visual Studio Code and others.



# Electronics Hub

i 3 Model B. So,

tions for Pico

Q

## Get Pico's C SDK and Examples

Assuming you are also using a Raspberry Pi computer, login to Pi using either SSH or VNC (if you connect Pi to a monitor, then well and good). I will be using the VNC viewer to open the desktop of Raspberry Pi.

Open the terminal and create a directory called 'pico' at /home/pi/pico. Enter the following commands one after the other.

```
cd ~/
```

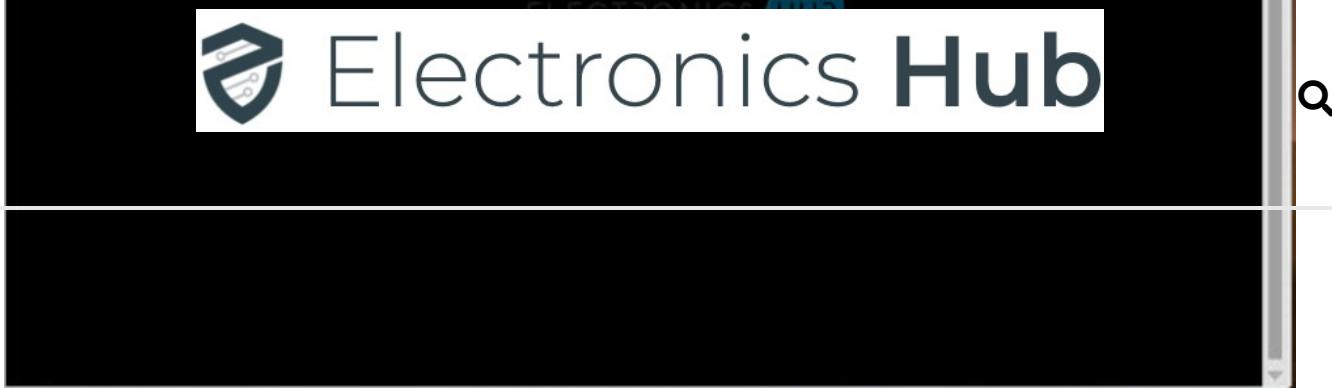
```
mkdir pico
```

```
cd pico
```

A screenshot of a terminal window titled "pi@raspberrypi: ~" with a dark background. The window shows the command line interface with the following text:

```
pi@raspberrypi:~ $ mkdir pico
pi@raspberrypi:~ $ cd pico/
pi@raspberrypi:~/pico $
```

The terminal window has a standard Linux-style interface with a menu bar at the top labeled "File Edit Tabs Help".



(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Download-SDK-1.jpg>)

## Download C SDK

The official repository for Pico's C SDK is <https://github.com/raspberrypi/pico-sdk> (<https://github.com/raspberrypi/pico-sdk>). Let us clone this repo into our newly created 'pico' directory using the following command.

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

A screenshot of a terminal window titled "pi@raspberrypi: ~ /pico". The window shows the command "git clone -b master https://github.com/raspberrypi/pico-sdk.git" being run and its output. The output includes the cloning process, object enumeration, counting, compressing, receiving objects, and resolving deltas. The terminal window has a dark background with light-colored text and a standard Linux-style interface.



(<https://www.electronicshub.org/>)

(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Download-SDK-2.jpg>)

This will create a directory called 'pico-sdk' inside the 'pico' directory. Before proceeding with downloading the examples, you have to initialize the 'tinyusb' submodule in the 'pico-sdk' for USB related functions like USB CDC Serial to work.

So, open 'pico-sdk' directory and initialize the USB submodule using the following commands.

```
cd pico-sdk
```

```
git submodule update --init
```

The screenshot shows a terminal window titled 'pi@raspberrypi: ~ /pico/pico-sdk'. The window has a standard Linux terminal interface with a menu bar (File, Edit, Tabs, Help) and a title bar. The main area displays the following command-line session:

```
pi@raspberrypi:~ $ mkdir pico
pi@raspberrypi:~ $ cd pico
pi@raspberrypi:~/pico $ git clone -b master https://github.com/raspberrypi/pico-sdk.git
Cloning into 'pico-sdk'...
remote: Enumerating objects: 86, done.
remote: Counting objects: 100% (86/86), done.
remote: Compressing objects: 100% (85/85), done.
remote: Total 2425 (delta 1), reused 86 (delta 1), pack-reused 2339
Receiving objects: 100% (2425/2425), 1.35 MiB | 1.77 MiB/s, done.
Resolving deltas: 100% (1042/1042), done.
pi@raspberrypi:~/pico $ cd pico-sdk/
pi@raspberrypi:~/pico/pico-sdk $ git submodule update --init
Submodule 'tinyusb' (https://github.com/raspberrypi/tinyusb.git) registered for path 'lib/tinyusb'
Cloning into '/home/pi/pico/pico-sdk/lib/tinyusb'...
Submodule path 'lib/tinyusb': checked out 'e0aa405d19e35dbf58cf502b8106455c1a3c2a5c'
pi@raspberrypi:~/pico/pico-sdk $
```



(<https://www.electronicshub.org>)

(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Download-SDK-3.jpg>)

## Download Examples

Raspberry Pi Foundation created a bunch of examples for different peripherals like GPIO, ADC, PWM, I<sup>2</sup>C, SPI etc. of Pico. Instead of writing C programs from scratch, we can make use of these examples to get familiar with different functions and APIs. The official repo of Pico examples is <https://github.com/raspberrypi/pico-examples> (<https://github.com/raspberrypi/pico-examples>).

We will clone this repo into the 'pico' directory. So, go back to the 'pico' directory and clone the git using the following commands.

```
cd ..
```

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

```
pi@raspberrypi:~/pico
File Edit Tabs Help
pi@raspberrypi:~ $ mkdir pico
pi@raspberrypi:~ $ cd pico
pi@raspberrypi:~/pico $ git clone -b master https://github.com/raspberrypi/pico-sdk.git
Cloning into 'pico-sdk'...
remote: Enumerating objects: 86, done.
remote: Counting objects: 100% (86/86), done.
remote: Compressing objects: 100% (85/85), done.
remote: Total 2425 (delta 1), reused 86 (delta 1), pack-reused 2339
Receiving objects: 100% (2425/2425), 1.35 MiB | 1.77 MiB/s, done.
Resolving deltas: 100% (1042/1042), done.
pi@raspberrypi:~/pico $ cd pico-sdk/
pi@raspberrypi:~/pico/pico-sdk $ git submodule update --init
Submodule 'tinyusb' (https://github.com/raspberrypi/tinyusb.git) registered for path 'lib/tinyusb'
Cloning into '/home/pi/pico/pico-sdk/lib/tinyusb'...
Submodule path 'lib/tinyusb': checked out 'e0aa405d19e35dbf58cf502b8106455c1a3c2a5c'
```



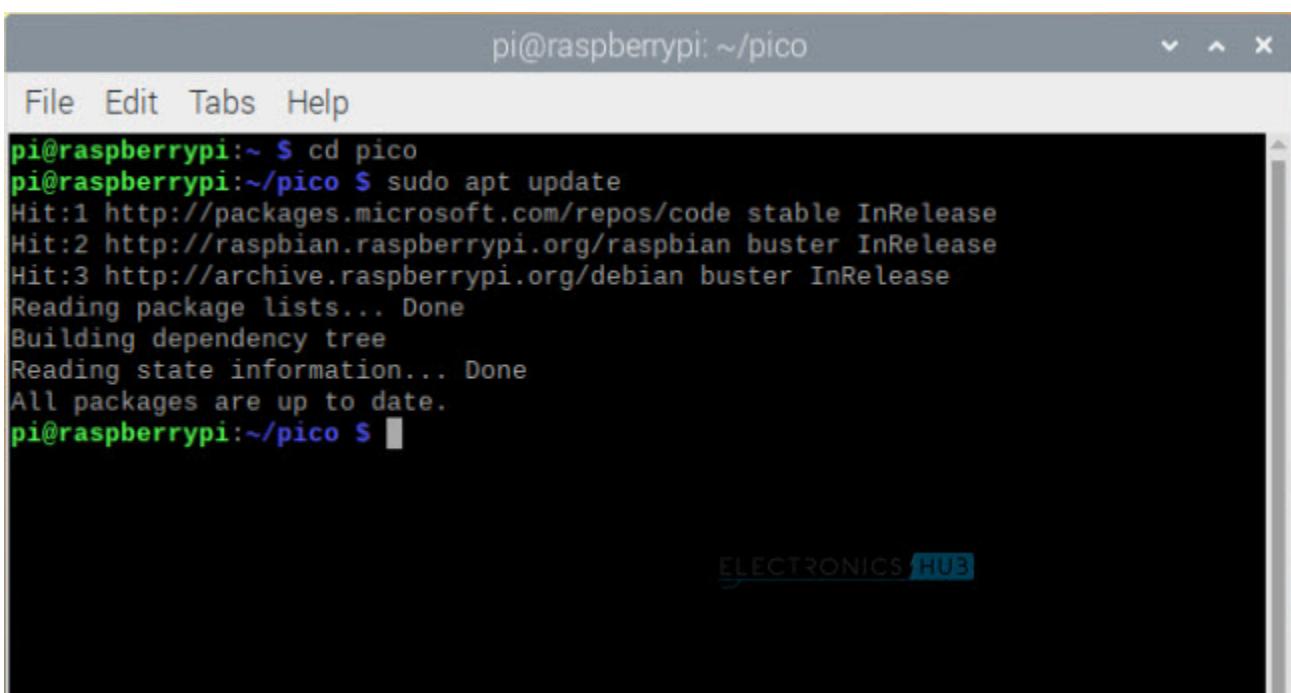
pi@raspberrypi:~/pico/pico-sdk \$ cd  
pi@raspberrypi:~/pico \$ Cloning into 'pico-  
remote: Enumerating  
remote: Counting ob  
remote: Compressing objects: 100% (40/40), done.  
remote: Total 623 (delta 27), reused 41 (delta 19), pack-reduced 566  
Receiving objects: 100% (623/623), 2.23 MiB | 2.58 MiB/s, done.  
Resolving deltas: 100% (200/200), done.  
pi@raspberrypi:~/pico \$

(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Download-Examples-1.jpg>)

## Installing the Toolchain

You just downloaded the C source files for Pico. To compile and build these files, you need additional tools like ARM GCC Compiler, ARM Cortex libraries, CMake build tool etc. We can install all these tools using ‘apt’. First, update the package index using ‘update’ command.

```
sudo apt update
```



pi@raspberrypi: ~/pico  
File Edit Tabs Help  
pi@raspberrypi:~ \$ cd pico  
pi@raspberrypi:~/pico \$ sudo apt update  
Hit:1 http://packages.microsoft.com/repos/code stable InRelease  
Hit:2 http://raspbian.raspberrypi.org/raspbian buster InRelease  
Hit:3 http://archive.raspberrypi.org/debian buster InRelease  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
All packages are up to date.  
pi@raspberrypi:~/pico \$



(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Install-Toolchain-1.jpg>)

Now, you can install the necessary tools using the following command. If you already have any of these tools installed, 'apt' will not install them again.

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

A screenshot of a terminal window titled "pi@raspberrypi: ~/pico". The window shows the command "sudo apt install cmake gcc-arm-none-eabi libnewlib-arm-none-eabi build-essential" being run. The terminal output indicates that the build-essential package is already the newest version (12.6). It lists additional packages to be installed, including binutils-arm-none-eabi, cmake-data, libjsoncpp1, libnewlib-dev, librhash0, libstdc++-arm-none-eabi-newlib, and libuv1. It also suggests cmake-doc, ninja-build, and libnewlib-doc. The terminal then asks if the user wants to continue with the installation, with options [Y/n]. The background of the terminal window has a watermark for "ELECTRONICS HUB".

(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Install-Toolchain-2.jpg>)

**WARNING:** This is a very large installation which occupies approximately 1800 MB of disk space. Make sure you have enough space on the SD Card in which you installed the Raspberry Pi OS. I used a 16 GB card.

Also, it will take  
relax



# Electronics Hub

: coffee and



(<https://www.electronicshub.org>)

```
[File Edit Tabs Help]
Preparing to unpack .../2-libjsoncpp1_1.7.4-3_armhf.deb ...
Unpacking libjsoncpp1:armhf (1.7.4-3) ...
Selecting previously unselected package librhash0:armhf.
Preparing to unpack .../3-librhash0_1.3.8-1_armhf.deb ...
Unpacking librhash0:armhf (1.3.8-1) ...
Selecting previously unselected package libuv1:armhf.
Preparing to unpack .../4-libuv1_1.24.1-1_armhf.deb ...
Unpacking libuv1:armhf (1.24.1-1) ...
Selecting previously unselected package cmake.
Preparing to unpack .../5-cmake_3.13.4-1_armhf.deb ...
Unpacking cmake (3.13.4-1) ...
Selecting previously unselected package gcc-arm-none-eabi.
Preparing to unpack .../6-gcc-arm-none-eabi_15%3a7-2018-q2-6_armhf.deb ...
Unpacking gcc-arm-none-eabi (15:7-2018-q2-6) ...
Selecting previously unselected package libnewlib-dev.
Preparing to unpack .../7-libnewlib-dev_3.1.0.20181231-1_all.deb ...
Unpacking libnewlib-dev (3.1.0.20181231-1) ...
Selecting previously unselected package libnewlib-arm-none-eabi.
Preparing to unpack .../8-libnewlib-arm-none-eabi_3.1.0.20181231-1_all.deb ...
Unpacking libnewlib-arm-none-eabi (3.1.0.20181231-1) ...
Selecting previously unselected package libstdc++-arm-none-eabi-newlib.
Preparing to unpack .../9-libstdc++-arm-none-eabi-newlib_15%3a7-2018-q2-5+12_all.deb ...
Unpacking libstdc++-arm-none-eabi-newlib (15:7-2018-q2-5+12) ...
Setting up binutils-arm-none-eabi (2.31.1-11+rpi1+11) ...
Setting up gcc-arm-none-eabi (15:7-2018-q2-6) ...
Setting up libuv1:armhf (1.24.1-1) ...
Setting up libnewlib-dev (3.1.0.20181231-1) ...
Setting up libnewlib-arm-none-eabi (3.1.0.20181231-1) ...
Setting up librhash0:armhf (1.3.8-1) ...
Setting up cmake-data (3.13.4-1) ...
Setting up libjsoncpp1:armhf (1.7.4-3) ...
Setting up libstdc++-arm-none-eabi-newlib (15:7-2018-q2-5+12) ...
Setting up cmake (3.13.4-1) ...
Processing triggers for man-db (2.8.5-2) ...
Processing triggers for libc-bin (2.28.10+rpi1) ...
pi@raspberrypi:~/pico $
```

(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Install-Toolchain-3.jpg>)

# Exploring Blink Example

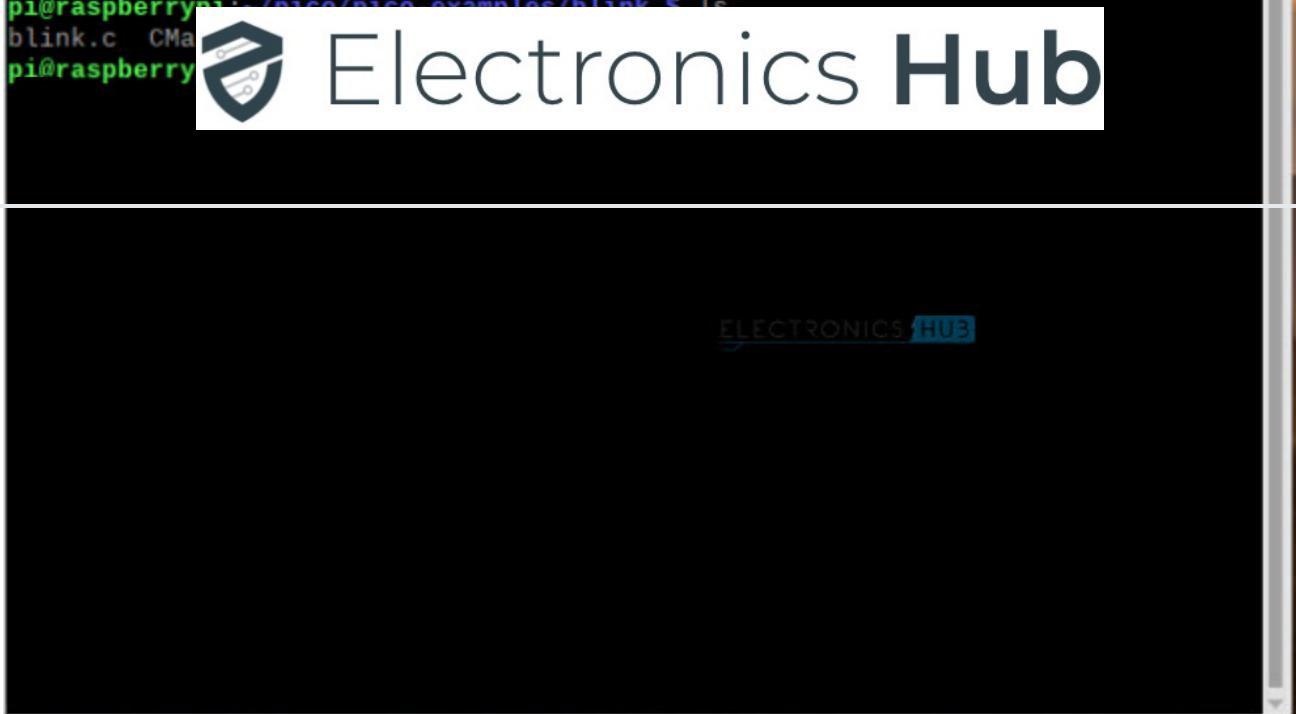
Let us now see the first C example program, which is nothing but the ‘Blink’ program, as expected. Assuming you are still in ‘pico’ directory, open the ‘pico-examples’ directory. This directory contains example for different peripherals and modules. Open ‘blink’ directory.

```
cd pico-examples/blink
```

pi@raspberrypi: ~ /pico/pico-examples/blink

File Edit Tabs Help

```
pi@raspberrypi:~ $ cd pico/pico-examples/blink/
```

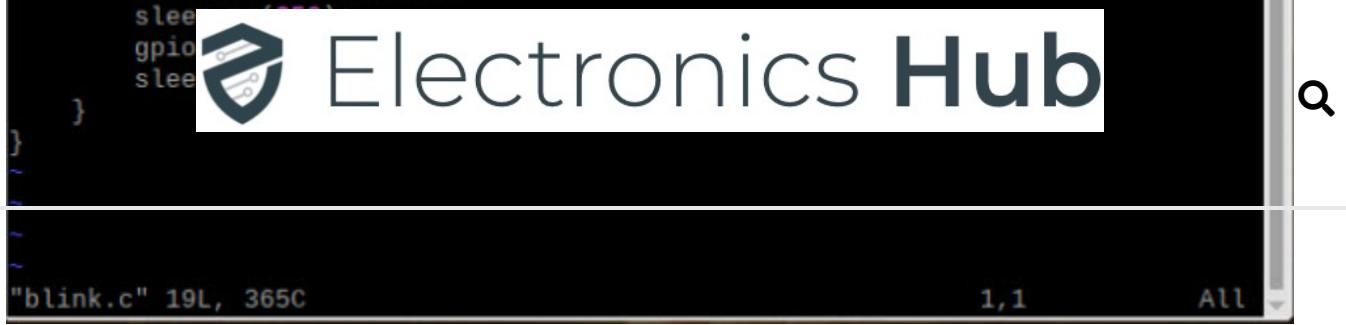


(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Build-Blink-1.jpg>)

Here, you can see the 'blink.c' source code for blinking the on-board LED. If you want to open it, use any editor like 'vim'. If you don't have 'vim', install it using `sudo apt install vim` command.

vim blink.c

A screenshot of a terminal window titled 'pi@raspberrypi: ~ /pico/pico-examples/blink'. The window shows the 'blink.c' file content. The code includes a header guard, a copyright notice for Raspberry Pi (Trading) Ltd., an SPDX license identifier, and a main function that initializes a pin as an output, sets it high, and then enters a loop where it toggles the pin every 100ms using a while loop and a sleep\_ms call.



The screenshot shows a terminal window with the following content:

```

    sleep();
    gpio_init(LED_PIN);
    sleep();
}

"blink.c" 19L, 365C

```

The status bar at the bottom right shows "1,1" and "All". The Electronics Hub logo is visible in the top right corner of the terminal window.

(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Build-Blink-2.jpg>)

I made a copy of the code and added comments for easy understanding.

```

1 /**
2  * Copyright (c) 2020 Raspberry Pi (Trading) Ltd.
3  *
4  * SPDX-License-Identifier: BSD-3-Clause
5 */
6
7 #include "pico/stdlib.h"
8
9 int main() {
10     const uint LED_PIN = 25; /* On-board LED. Connected to GPIO 25 */
11     gpio_init(LED_PIN); /* Initialize GPIO function on the Pin */
12     gpio_set_dir(LED_PIN, GPIO_OUT); /* Set the Pin as Output */
13     while (true) {
14         gpio_put(LED_PIN, 1); /* Make the GPIO Pin HIGH */
15         sleep_ms(250); /* Delay of 250ms */
16         gpio_put(LED_PIN, 0); /* Make the GPIO Pin LOW */
17         sleep_ms(250); /* Delay of 250ms */
18     }
19 }
```

[view raw \(https://gist.github.com/elktros/9cdd468f0d1ae176d6529135913f0169/raw/a0a2310aec4466c00ce310b609b2f424422ebb64/Raspberry-Pi-Pico-C-Blink.c\)](https://gist.github.com/elktros/9cdd468f0d1ae176d6529135913f0169/raw/a0a2310aec4466c00ce310b609b2f424422ebb64/Raspberry-Pi-Pico-C-Blink.c)

Raspberry-Pi-Pico-C-Blink.c (<https://gist.github.com/elktros/9cdd468f0d1ae176d6529135913f0169#file-raspberry-pi-pico-c-blink-c>) hosted with ❤ by GitHub (<https://github.com>)

We have to now build this source code and generate a binary file.

# Build the Electronics Hub

Go back to 'pico-examples' directory and create a new directory called 'build'. This (<https://www.electronicshub.org>)

directory contains all the CMake build files for all the example projects. Open the newly created 'build' directory.

```
cd ..
```

```
mkdir build
```

```
cd build
```

Also, export the path for pico-sdk using the following command.

```
export PICO_SDK_PATH=../../pico-sdk
```

The screenshot shows a terminal window titled 'pi@raspberrypi:~/pico/pico-examples/build'. The window has a standard Linux-style title bar with icons for minimizing, maximizing, and closing. Below the title bar is a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The main area of the terminal shows the following command-line session:

```
pi@raspberrypi:~ $ cd pico/pico-examples/blink/
pi@raspberrypi:~/pico/pico-examples/blink $ ls
blink.c  CMakeLists.txt
pi@raspberrypi:~/pico/pico-examples/blink $ vim blink.c
pi@raspberrypi:~/pico/pico-examples/blink $ cd ..
pi@raspberrypi:~/pico/pico-examples $ mkdir build
pi@raspberrypi:~/pico/pico-examples $ cd build/
pi@raspberrypi:~/pico/pico-examples/build $ export PICO_SDK_PATH=../../pico-sdk
pi@raspberrypi:~/pico/pico-examples/build $
```

The word 'ELECTRONICS HUB' is visible in the bottom right corner of the terminal window.

(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Build-Blink-3.jpg>)

You are now rea



# Electronics Hub



(<https://www.electronicshub.org>)

cmake ..

```
pi@raspberrypi:~/pico/pico-examples/build
File Edit Tabs Help
pi@raspberrypi:~/pico/pico-examples/blink $ cd ..
pi@raspberrypi:~/pico/pico-examples $ mkdir build
pi@raspberrypi:~/pico/pico-examples $ cd build/
pi@raspberrypi:~/pico/pico-examples/build $ export PICO_SDK_PATH=../../pico-sdk
pi@raspberrypi:~/pico/pico-examples/build $ cmake ..
Using PICO_SDK_PATH from environment ('../../pico-sdk')
PICO_SDK_PATH is /home/pi/pico/pico-sdk
Defaulting PICO_PLATFORM to rp2040 since not specified.
Defaulting PICO platform compiler to pico_arm_gcc since not specified.
-- Defaulting build type to 'Release' since not specified.
PICO compiler is pico_arm_gcc
PICO_GCC_TRIPLE defaulted to arm-none-eabi
-- The C compiler identification is GNU 7.3.1
-- The CXX compiler identification is GNU 7.3.1
-- The ASM compiler identification is GNU
-- Found assembler: /usr/bin/arm-none-eabi-gcc
Defaulting PICO target board to pico since not specified.
Using board configuration from /home/pi/pico/pico-sdk/src/boards/include/boards/pico.h
-- Found Python3: /usr/bin/python3.7 (found version "3.7.3") found components: Interpreter
TinyUSB available at /home/pi/pico/pico-sdk/lib/tinyusb/src/portable/raspberrypi/rp2040; adding USB support.
-- Could NOT find Doxygen (missing: DOXYGEN_EXECUTABLE)
ELF2UF2 will need to be built
PIOASM will need to be built
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/pico/pico-examples/build
pi@raspberrypi:~/pico/pico-examples/build $
```

ELECTRONICS HUB

(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Build-Blink-4.jpg>)

The 'build' directory is now populated with all the build related files for all the examples. But as we are interested only in the blink example, let us build that example using the make command. First, open the 'blink' directory in the 'build' directory and make the files.

cd blink



```
make -j4
pi@raspberrypi:~/pico/pico-examples/build$ cmake ..
Using PICO_SDK_PATH from environment ('../../pico-sdk')
PICO_SDK_PATH is /home/pi/pico/pico-sdk
Defaulting PICO_PLATFORM to rp2040 since not specified.
Defaulting PICO platform compiler to pico_arm_gcc since not specified.
-- Defaulting build type to 'Release' since not specified.
PICO compiler is pico_arm_gcc
PICO_GCC_TRIPLE defaulted to arm-none-eabi
-- The C compiler identification is GNU 7.3.1
-- The CXX compiler identification is GNU 7.3.1
-- The ASM compiler identification is GNU
-- Found assembler: /usr/bin/arm-none-eabi-gcc
Defaulting PICO target board to pico since not specified.
Using board configuration from /home/pi/pico/pico-sdk/src/boards/include/boards/pico.h
-- Found Python3: /usr/bin/python3.7 (found version "3.7.3") found components: Interpreter
TinyUSB available at /home/pi/pico/pico-sdk/lib/tinyusb/src/portable/raspberrypi/rp2040; adding USB support.
-- Could NOT find Doxygen (missing: DOXYGEN_EXECUTABLE)
ELF2UF2 will need to be built
PIOASM will need to be built
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/pico/pico-examples/build
pi@raspberrypi:~/pico/pico-examples/build$ cd blink
pi@raspberrypi:~/pico/pico-examples/build/blink$ make -j4
```

(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Build-Blink-5.jpg>)

**NOTE:** The '-j' option lets you specify the number of simultaneous jobs to run. Since, my Raspberry Pi 3 has a quad core CPU, I chose 4 simultaneous jobs.

The 'make' command will compile all the source files and generates a bunch of binary files.

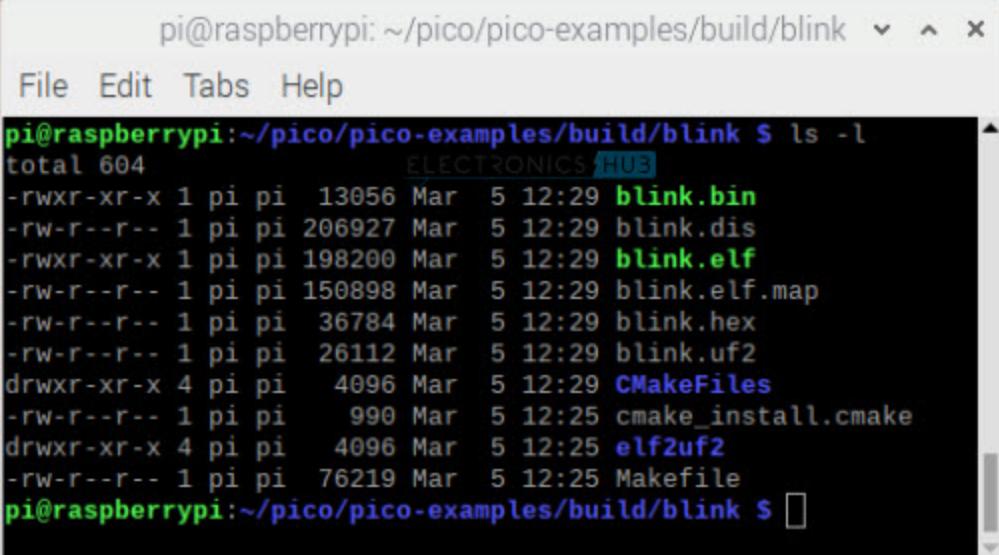
```
pi@raspberrypi:~/pico/pico-examples/build/blink
File Edit Tabs Help
aeabi.S.obj
[100%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico_float/float_in
it_rom.c.obj
[100%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico_float/float_ma
th.c.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico_float/float_
vi_rom_shim.S.obj
[100%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico_malloc/pico_ma
lloc.c.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico_mem_ops/mem_
ops_aeabi.S.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico_standard_lin
k/crt0.S.obj
[100%] Building CXX object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico_standard.lin
```



k/new\_delete.cpp.o  
[100%] Building C object binary\_info.c.obj  
[100%] Building C object io\_uart.c.obj  
[100%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2\_common/pico\_stdio\_uart/stdio\_uart.c.obj  
[100%] Linking CXX executable blink.elf  
[100%] Built target blink  
pi@raspberrypi:~/pico/pico-examples/build/blink \$

(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Build-Blink-6.jpg>)

If you list out the contents of the 'blink' directory in the 'build' directory, you can see the list of generated files.



pi@raspberrypi: ~/pico/pico-examples/build/blink \$ ls -l  
total 604  
-rwxr-xr-x 1 pi pi 13056 Mar 5 12:29 blink.bin  
-rw-r--r-- 1 pi pi 206927 Mar 5 12:29 blink.dis  
-rwxr-xr-x 1 pi pi 198200 Mar 5 12:29 blink.elf  
-rw-r--r-- 1 pi pi 150898 Mar 5 12:29 blink.elf.map  
-rw-r--r-- 1 pi pi 36784 Mar 5 12:29 blink.hex  
-rw-r--r-- 1 pi pi 26112 Mar 5 12:29 blink.uf2  
drwxr-xr-x 4 pi pi 4096 Mar 5 12:29 CMakeFiles  
-rw-r--r-- 1 pi pi 990 Mar 5 12:25 cmake\_install.cmake  
drwxr-xr-x 4 pi pi 4096 Mar 5 12:25 elf2uf2  
-rw-r--r-- 1 pi pi 76219 Mar 5 12:25 Makefile  
pi@raspberrypi:~/pico/pico-examples/build/blink \$

(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Build-Blink-7.jpg>)

Of all these files, we are interested in the '.uf2' file named 'blink.uf2'. If you remember the MicroPython tutorial, this is file format of the MicroPython binary which can be easily uploaded in to Raspberry Pi Pico simply by dragging and dropping (after setting the Pico in bootloader mode).

## Load the Blink and Run it

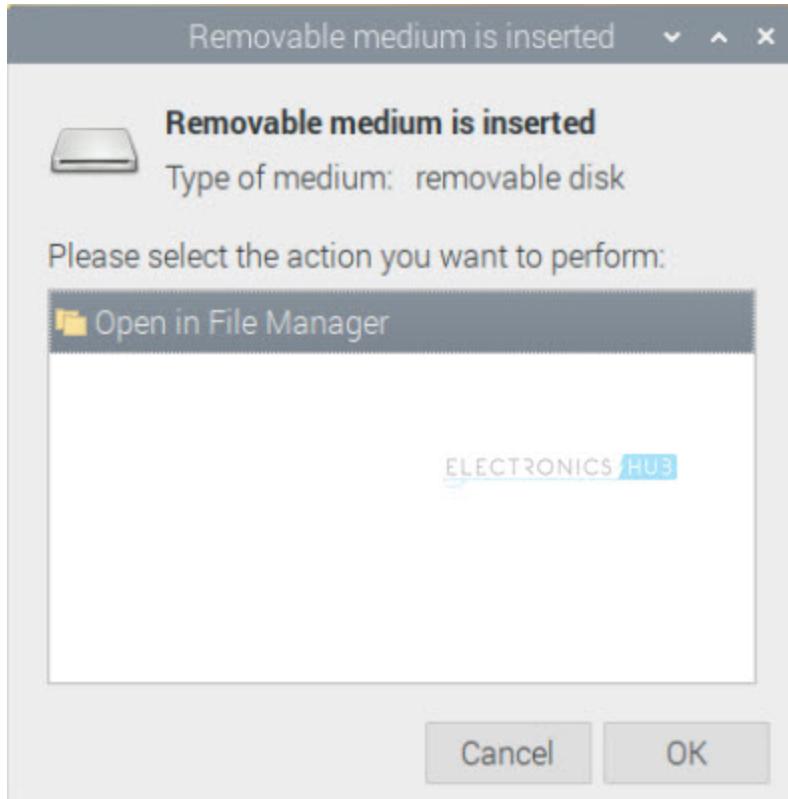
The simplest way to upload the program to Raspberry Pi Pico is to put it in Bootloader

mode, which will be explained later. You can simply drag-and-drop the file onto the Pico.

# Electronics Hub

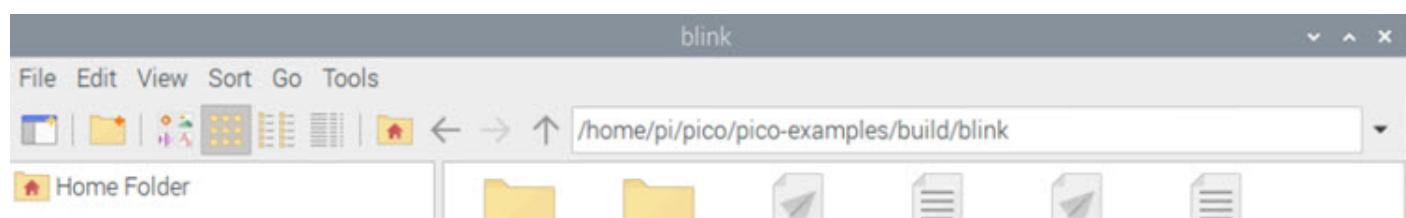
(<https://www.electronicshub.org>)

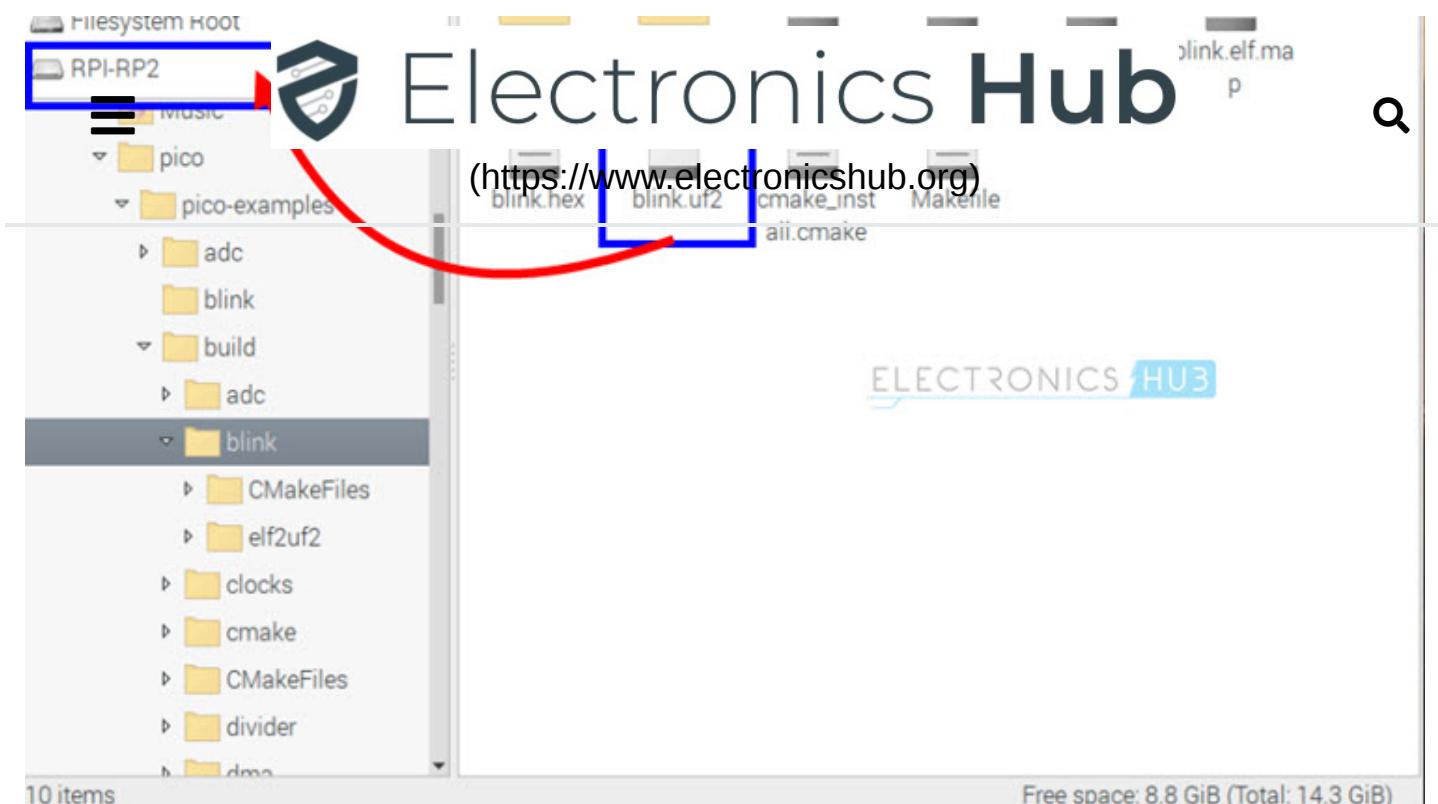
To put the Pico in Bootloader mode first plug-in the micro USB cable to Raspberry Pi Pico and hold the 'BOOTSEL' button while plugging in the USB cable to Raspberry Pi (or the host computer). The Pico will mount as a mass storage device.



(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Load-Blink-1.jpg>)

Open the file manager, browse to the pico/pico-examples/build/blink directory. Simply drag and drop the 'blink.uf2' file onto Pico. The Pico will reset (and gets disconnected from the computer) and the LED starts blinking.





(<https://www.electronicshub.org/wp-content/uploads/2021/03/Pico-C-Load-Blink-2.jpg>)

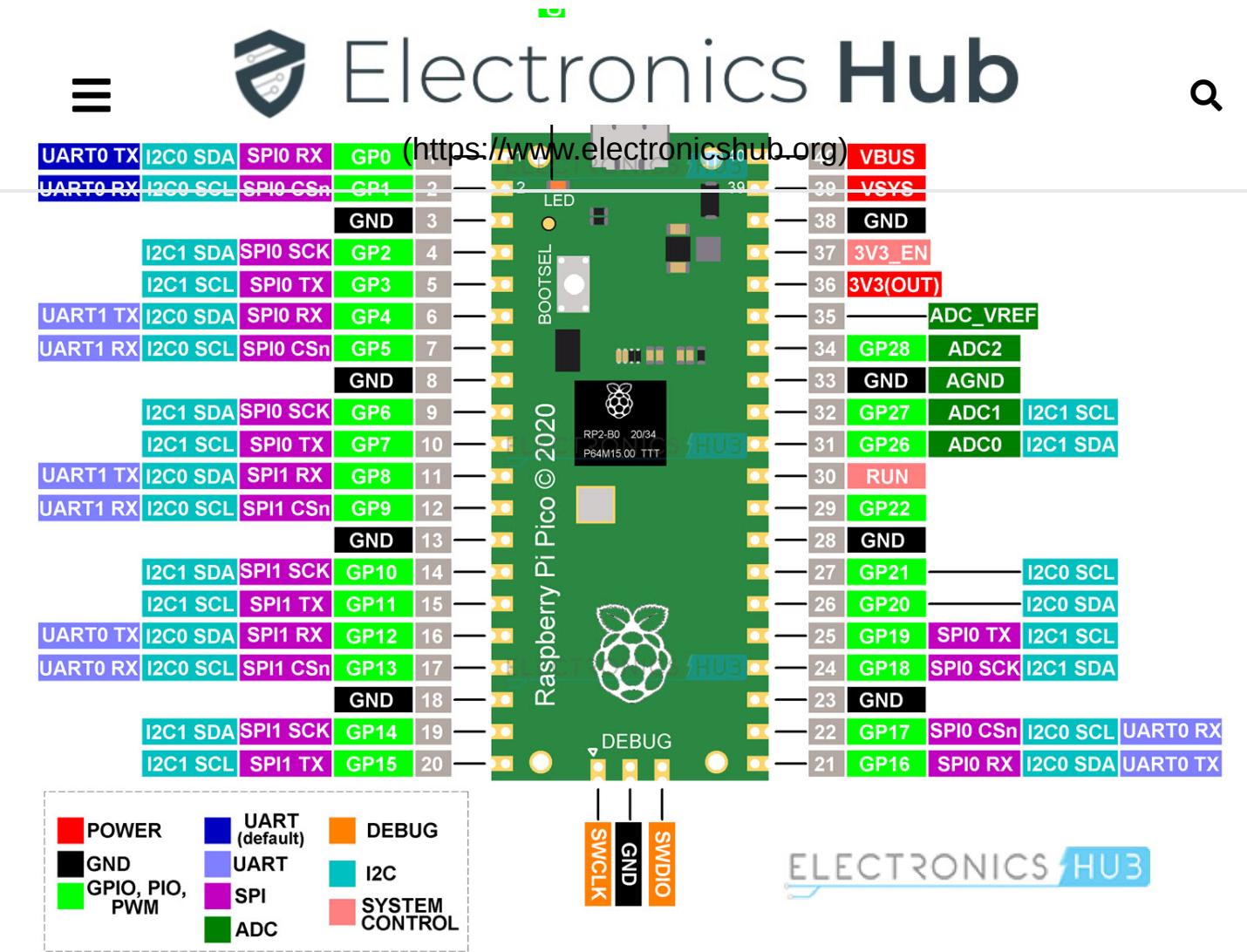
## Bonus: Alternative to Unplugging and Plugging USB Cable

Are you tired of unplugging and plugging the Raspberry Pi Pico to put into bootloader mode? Then I will show you a work around so that you don't have to do that every time you want upload a program to Pico.

What we are actually doing when unplugging and plugging Pico is resetting it. So, if we can find another way to reset Pico, then we can keep the Pico plugged in to the computer. As it turns out, there is rather a simple way to reset Pico.

If you remember the Pinout of Raspberry Pi Pico, there is a pin called 'RUN'. This is pin number 30 on Raspberry Pi Pico.

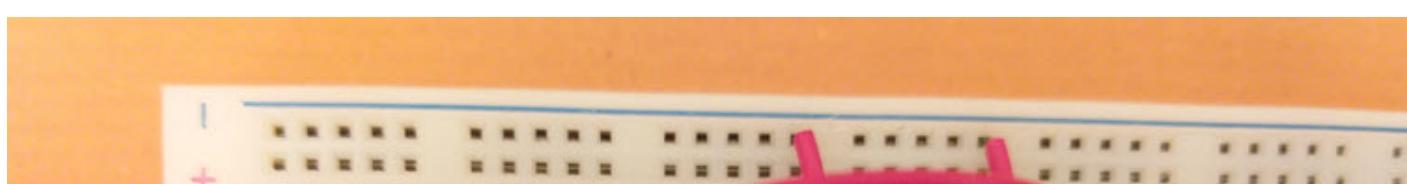


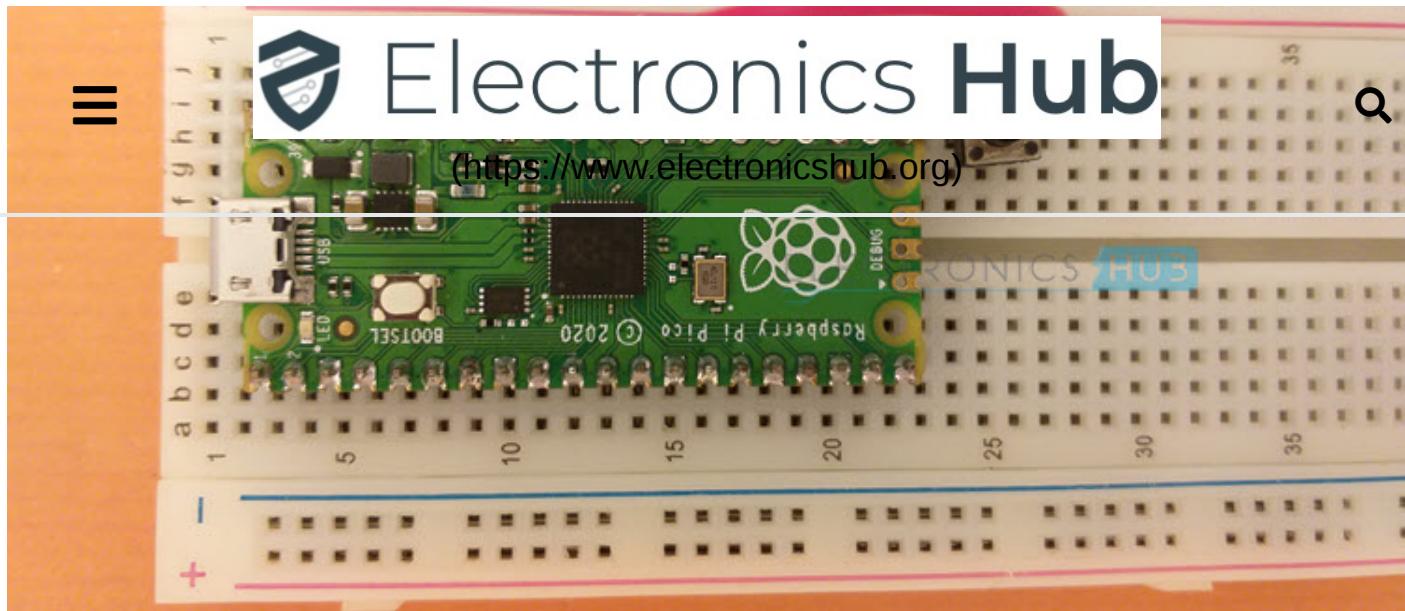


(<https://www.electronicshub.org/wp-content/uploads/2021/02/Raspberry-Pi-Pico-Pinout.jpg>)

This is actually the global reset pin of RP2040 Microcontroller. When this pin is pulled LOW, the RP2040 microcontroller will reset.

So, I connected a push button between RUN pin and GND. Whenever I push the button, it will reset the microcontroller. Let us call this button as Reset Button.





(<https://www.electronicshub.org/wp-content/uploads/2021/03/Program-Raspberry-Pi-Pico-using-C-Reset.jpg>)

Now, to put the Raspberry Pi Pico in Bootloader Mode, first push and hold the Reset Button. Then, push and hold the BOOTSEL button. Now, first release the Reset button and then after a second, release the BOOTSEL button.

That's it. Your Pico is now in Bootloader Mode. You can keep the USB cable connected to Pico and the host computer (Raspberry Pi) but you can easily reset or select bootloader mode.

If you just want to reset the Pico, simply push the Reset Button once and release it.

## Conclusion

A complete beginner's guide to Getting Started with C SDK for Pico. You learned how to Program Raspberry Pi Pico using C Programming Language by understanding how to download the Pico C SDK, install all the necessary tools, build the source code and upload the binary to Raspberry Pi Pico.



# Electronics Hub

pi-pico-using-



- How says Setup Raspberry Pi Pico Serial Programming?... (<https://www.electronicshub.org/raspberry-pi-pico-serial-programming/>)

Hello, I can follow this tutorial until "git submodule update --init". With this command I get

- Learn How to Create Your Own New Project for... (<https://www.electronicshub.org/what-can-i-do-with-my-new-project/>)

- How to Program Raspberry Pi Pico with Visual Studio Code?

Reply (<https://www.electronicshub.org/program-raspberry-pi-pico-with-visual-studio-code/>)

- Programming Raspberry Pi Pico with MicroPython | A March 6, 2021 at 7:08 am (<https://www.electronicshub.org/program-raspberry-pi-pico-using-c/#comment-2425444>)

→ (<https://www.electronicshub.org/raspberry-pi-pico-micropython-tutorial/>)

- Learn how to Program and Debug Raspberry Pi Pico with SWD

It should be git submodule update 'doubledash'init  
(<https://www.electronicshub.org/programming-raspberry-pi-pico-with-swd/>)  
git submodule update (-)(-)init without parenthesis.

- Getting Started With Raspberry Pi Pico | An Introduction

There's problem with text editor (<https://www.electronicshub.org/getting-started-with-raspberry-pi-pico/>)

Reply



**Ravi Teja**

March 6, 2021 at 7:15 am (<https://www.electronicshub.org/program-raspberry-pi-pico-using-c/#comment-2425446>)

says:

Take a look at the image next to the command.

Reply

**john black**

March 11, 2021 at 6:38 am (<https://www.electronicshub.org/program-raspberry-pi-pico-using-c/#comment-2425951>)

says:

There is an error in <https://devconnected.com/how-to-add-and-update-git-submodules/> (<https://devconnected.com/how-to-add-and-update-git-submodules/>) git submodule update --init should read git submodule update --init --recursive. Otherwise thank you for a great article.

Reply



# Electronics Hub



(<https://www.electronicshub.org>)

**David Francis** April 7, 2021 at 11:22 am (<https://www.electronicshub.org/program-raspberry-pi-pico-using-c/#comment-2428312>)  
says:

I was hoping to find out how to load the SDK on to a Linux Mint computer, any suggestions?

Reply

↪ **Ravi Teja** April 8, 2021 at 2:34 am (<https://www.electronicshub.org/program-raspberry-pi-pico-using-c/#comment-2428355>)  
says:

The installation on any Linux machine should be the same. You can follow the same steps for Linux Mint as well. Since Mint is based on Ubuntu, you might have to install an additional library “libstdc++-arm-none-eabi-newlib” (after installing the toolchain).

Reply

## Leave a Reply

Your email address will not be published. Required fields are marked \*

Comment

Name \*



# Electronics Hub



(<https://www.electronicshub.org>)

Email \*

Website

Post Comment



# Electronics Hub



WWW ELECTRONICSHUB.ORG  
W.FA OUT WWW  
COHUB.WIN  
(https://www.electronicshub.org)

ook. Will .co sta

com @gra

/ele /use m.c

ctro /eh r/el om

nics g) ectr /eh

hub g) onic or

nic g) shu g)

Get Our Latest Newsletters

Get Great Content That You'll Love.  
No Ads Or Spam. We Promise.

Sign Up

Enter your email

Sign Up

Your Privacy Is Important To Us

General ()



Projects ()



Projects ()



Tutorials ()





# Electronics Hub



(<https://www.electronicshub.org>)

Contact

(<https://www.electronicshub.org/contact/>)

---

Affiliate Disclosure(<https://www.electronicshub.org/affiliate-disclosure/>) |

Disclaimer(<https://www.electronicshub.org/disclaimer/>) |

Terms and Conditions(<https://www.electronicshub.org/terms-and-conditions/>) |

Privacy Policy(<https://www.electronicshub.org/privacy-policy/>)

Copyright © 2021 Electronicshub.org