



DesignNews

Scratch Building Raspberry Pi RP2040 IoT Devices

Day 2:

Assembling an RP2040/Pico/Pico W Linux Toolchain

Sponsored by



Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Attendee Chat’ by maximizing the chat widget in your dock.

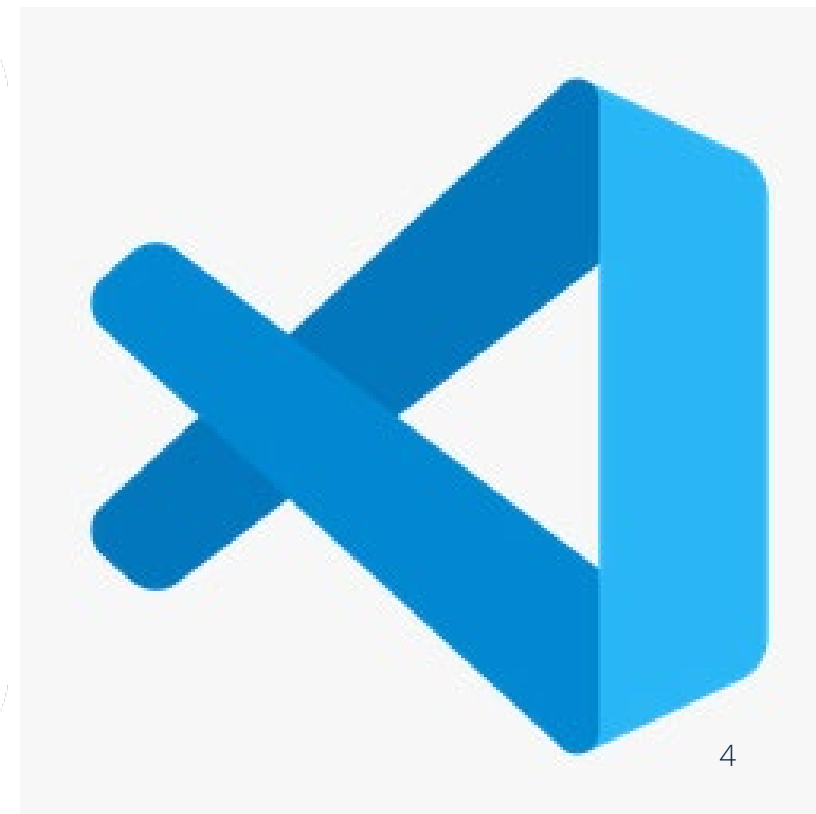
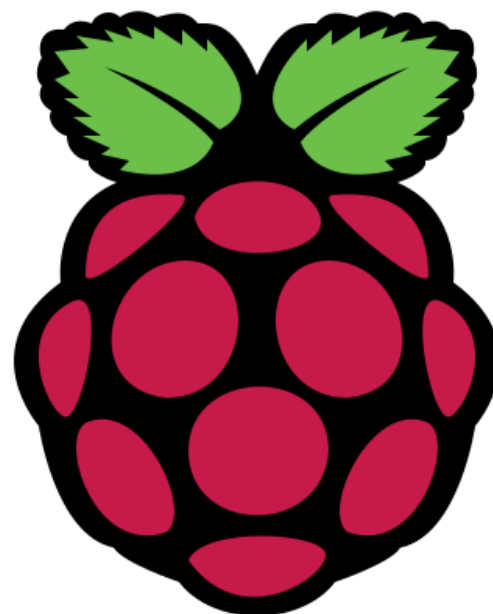


Fred Eady

Visit 'Lecturer Profile' in your console for more details.

AGENDA

- **Install the RP2040 Toolchain**
- **Install the Raspberry Pi Pico C/C++ SDK**
- **Install Visual Studio Code for Linux**
- **Toolchain Installation Verification**



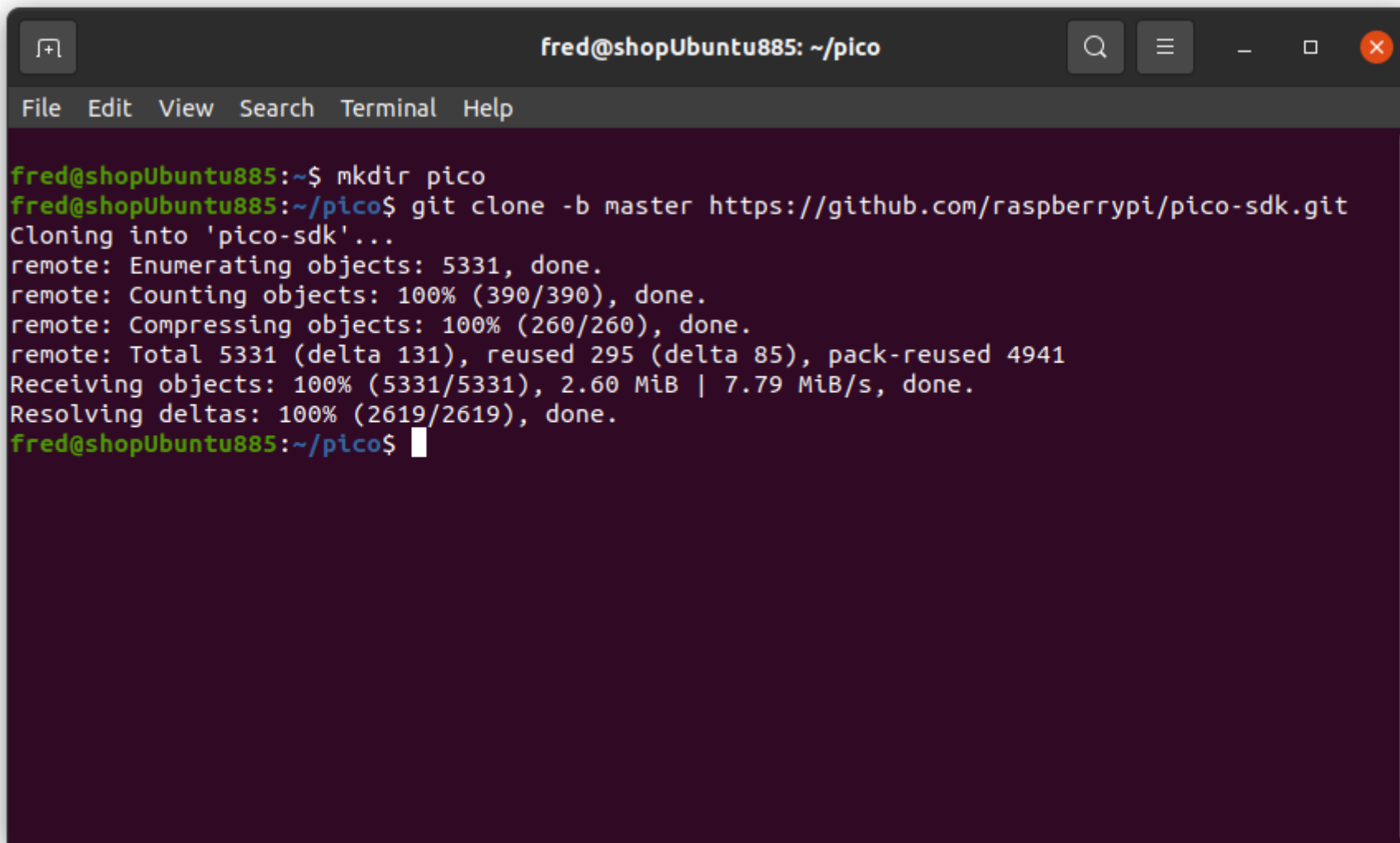
Perform an Ubuntu Update

```
fred@shopUbuntu885: ~  
File Edit View Search Terminal Help  
fred@shopUbuntu885:~$ sudo apt update  
Hit:1 http://us.archive.ubuntu.com/ubuntu focal InRelease  
Get:2 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]  
Get:3 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]  
Get:4 http://us.archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]  
Hit:5 http://packages.microsoft.com/repos/code stable InRelease  
Get:6 http://us.archive.ubuntu.com/ubuntu focal-updates/main amd64 DEP-11 Metadata [278 kB]  
Get:7 http://us.archive.ubuntu.com/ubuntu focal-updates/universe amd64 DEP-11 Metadata [390 kB]  
Get:8 http://us.archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 DEP-11 Metadata [944 B]  
Get:9 http://security.ubuntu.com/ubuntu focal-security/main amd64 DEP-11 Metadata [40.7 kB]  
Get:10 http://security.ubuntu.com/ubuntu focal-security/universe amd64 DEP-11 Metadata [66.3 kB]  
Get:11 http://us.archive.ubuntu.com/ubuntu focal-backports/main amd64 DEP-11 Metadata [7,964 B]  
Get:12 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 DEP-11 Metadata [2,468 B]  
Get:13 http://us.archive.ubuntu.com/ubuntu focal-backports/universe amd64 DEP-11 Metadata [30.5 kB]  
Fetched 1,153 kB in 1s (1,362 kB/s)  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
All packages are up to date.  
fred@shopUbuntu885:~$
```

GCC Toolchain Installation

```
fred@shopUbuntu885: ~  
File Edit View Search Terminal Help  
fred@shopUbuntu885:~$ sudo apt install cmake gcc-arm-none-eabi libnewlib-arm-none-eabi build-essential  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
cmake is already the newest version (3.16.3-1ubuntu1).  
libnewlib-arm-none-eabi is already the newest version (3.3.0-0ubuntu1).  
libnewlib-arm-none-eabi set to manually installed.  
build-essential is already the newest version (12.8ubuntu1.1).  
The following NEW packages will be installed:  
  gcc-arm-none-eabi  
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.  
Need to get 31.6 MB of archives.  
After this operation, 524 MB of additional disk space will be used.  
Do you want to continue? [Y/n] Y  
Get:1 http://us.archive.ubuntu.com/ubuntu focal/universe amd64 gcc-arm-none-eabi amd64 15:9-2019-q4-0u  
buntu1 [31.6 MB]  
Fetched 31.6 MB in 4s (8,550 kB/s)  
Selecting previously unselected package gcc-arm-none-eabi.  
(Reading database ... 327560 files and directories currently installed.)  
Preparing to unpack .../gcc-arm-none-eabi_15%3a9-2019-q4-0ubuntu1_amd64.deb ...  
Unpacking gcc-arm-none-eabi (15:9-2019-q4-0ubuntu1) ...  
Setting up gcc-arm-none-eabi (15:9-2019-q4-0ubuntu1) ...  
Processing triggers for libc-bin (2.31-0ubuntu9.9) ...  
fred@shopUbuntu885:~$
```

Raspberry Pi Pico C/C++ Installation



```
fred@shopUbuntu885: ~/pico
File Edit View Search Terminal Help

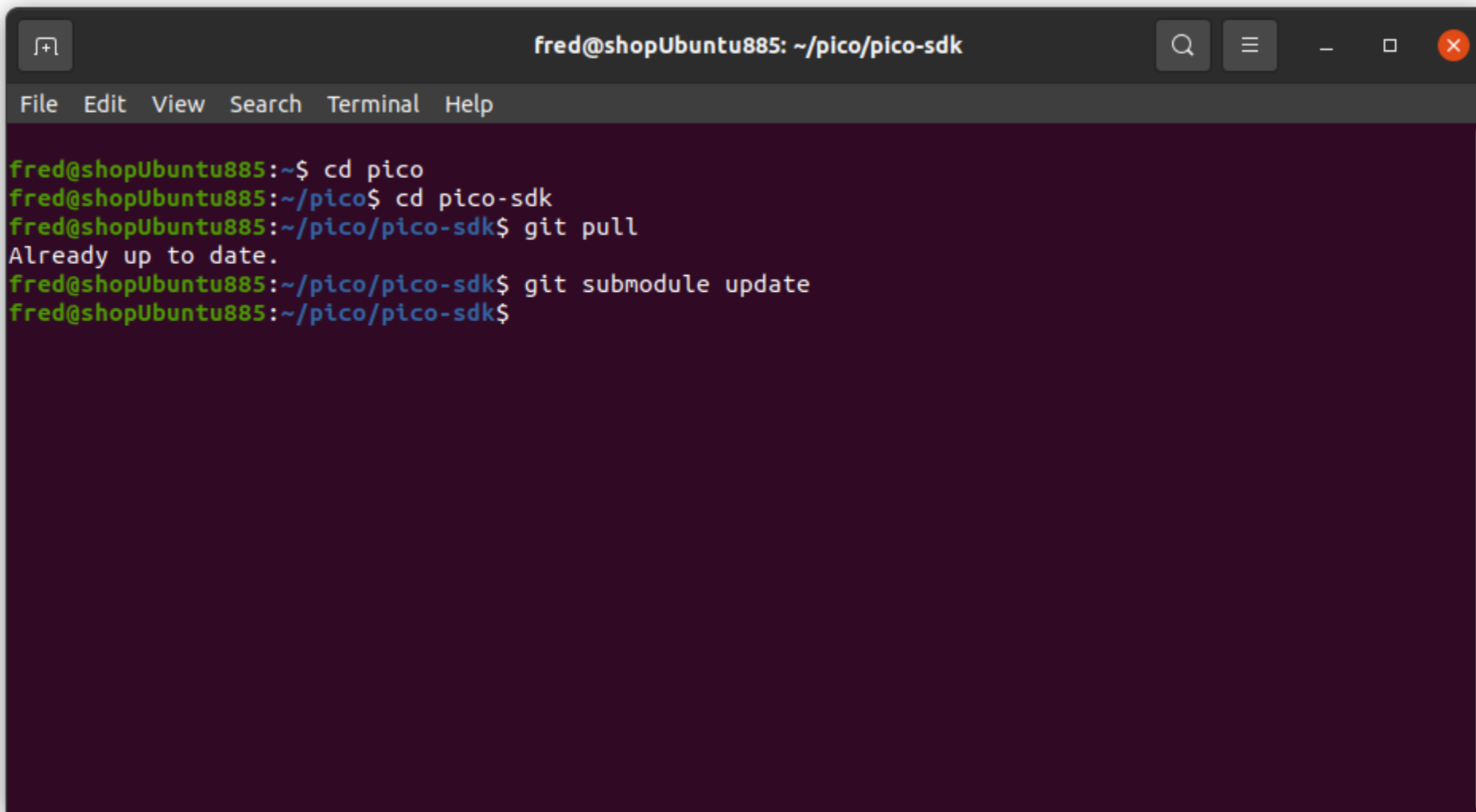
fred@shopUbuntu885:~$ mkdir pico
fred@shopUbuntu885:~/pico$ git clone -b master https://github.com/raspberrypi/pico-sdk.git
Cloning into 'pico-sdk'...
remote: Enumerating objects: 5331, done.
remote: Counting objects: 100% (390/390), done.
remote: Compressing objects: 100% (260/260), done.
remote: Total 5331 (delta 131), reused 295 (delta 85), pack-reused 4941
Receiving objects: 100% (5331/5331), 2.60 MiB | 7.79 MiB/s, done.
Resolving deltas: 100% (2619/2619), done.
fred@shopUbuntu885:~/pico$
```

Raspberry Pi Pico C/C++ Installation

```
fred@shopUbuntu885: ~/pico/pico-sdk
File Edit View Search Terminal Help

fred@shopUbuntu885:~$ mkdir pico
fred@shopUbuntu885:~/pico$ git clone -b master https://github.com/raspberrypi/pico-sdk.git
Cloning into 'pico-sdk'...
remote: Enumerating objects: 5331, done.
remote: Counting objects: 100% (390/390), done.
remote: Compressing objects: 100% (260/260), done.
remote: Total 5331 (delta 131), reused 295 (delta 85), pack-reused 4941
Receiving objects: 100% (5331/5331), 2.60 MiB | 7.79 MiB/s, done.
Resolving deltas: 100% (2619/2619), done.
fred@shopUbuntu885:~/pico$ cd pico-sdk
fred@shopUbuntu885:~/pico/pico-sdk$ git submodule update --init
Submodule 'lib/cyw43-driver' (https://github.com/georgerobotics/cyw43-driver.git) registered
for path 'lib/cyw43-driver'
Submodule 'lib/lwip' (https://github.com/lwip-tcpip/lwip.git) registered for path 'lib/lwip'
Submodule 'tinysub' (https://github.com/hathach/tinysub.git) registered for path 'lib/tinysub'
Cloning into '/home/fred/pico/pico-sdk/lib/cyw43-driver'...
Cloning into '/home/fred/pico/pico-sdk/lib/lwip'...
Cloning into '/home/fred/pico/pico-sdk/lib/tinysub'...
Submodule path 'lib/cyw43-driver': checked out '195dfcc10bb6f379e3dea45147590db2203d3c7b'
Submodule path 'lib/lwip': checked out '239918ccc173cb2c2a62f41a40fd893f57faf1d6'
Submodule path 'lib/tinysub': checked out '4bfab30c02279a0530e1a56f4a7c539f2d35a293'
fred@shopUbuntu885:~/pico/pico-sdk$
```

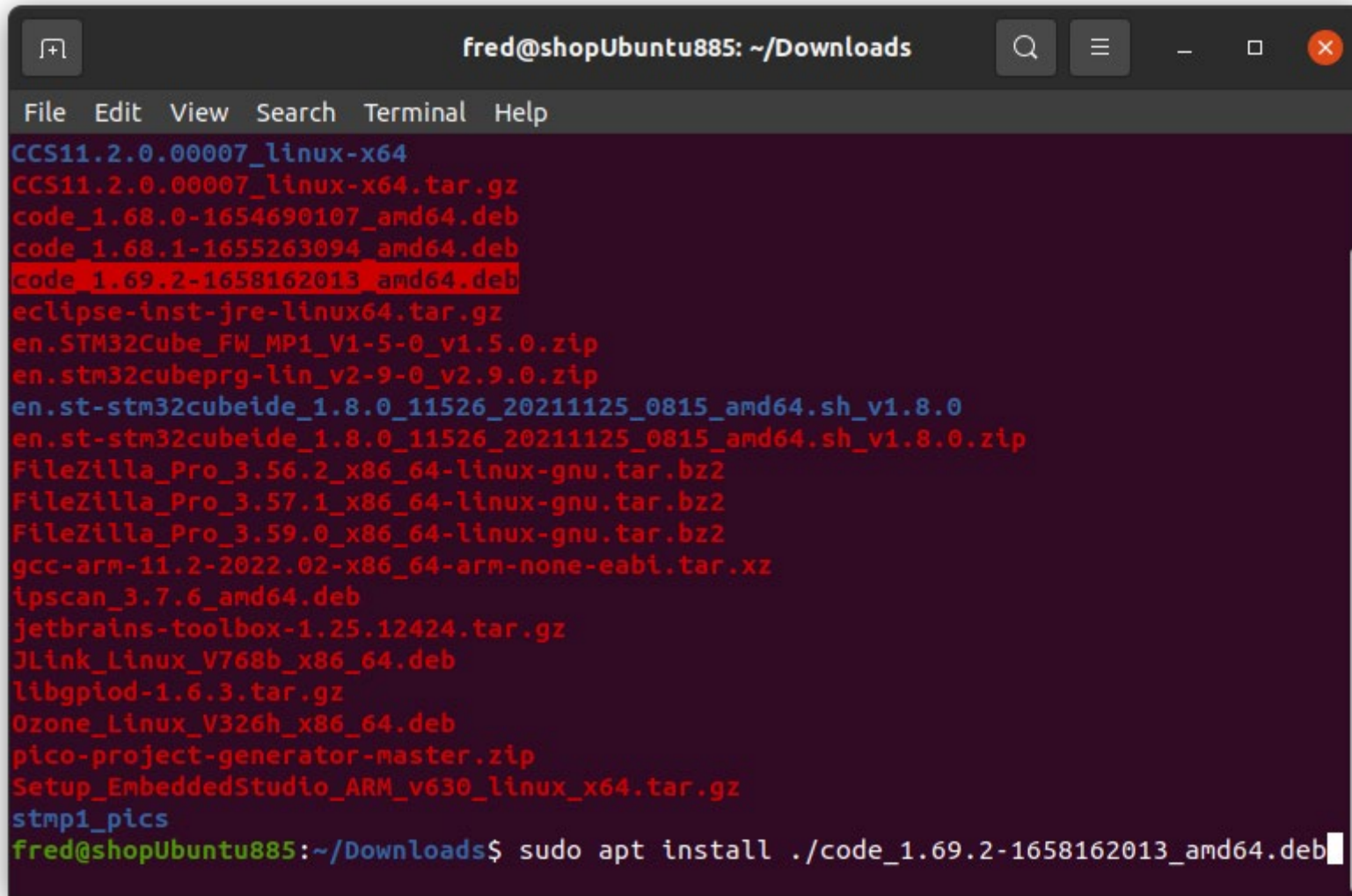
Raspberry Pi Pico C/C++ Installation



```
fred@shopUbuntu885: ~/pico/pico-sdk
File Edit View Search Terminal Help

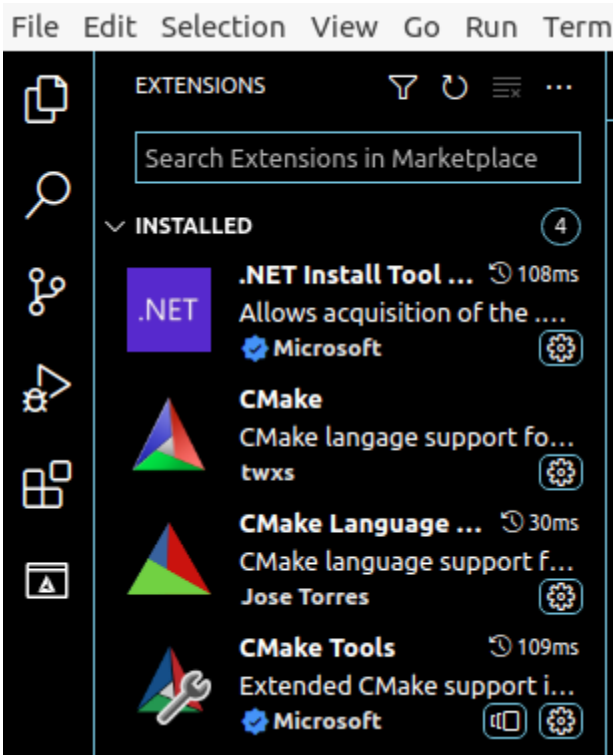
fred@shopUbuntu885:~$ cd pico
fred@shopUbuntu885:~/pico$ cd pico-sdk
fred@shopUbuntu885:~/pico/pico-sdk$ git pull
Already up to date.
fred@shopUbuntu885:~/pico/pico-sdk$ git submodule update
fred@shopUbuntu885:~/pico/pico-sdk$
```

Visual Studio Code Debian Installation

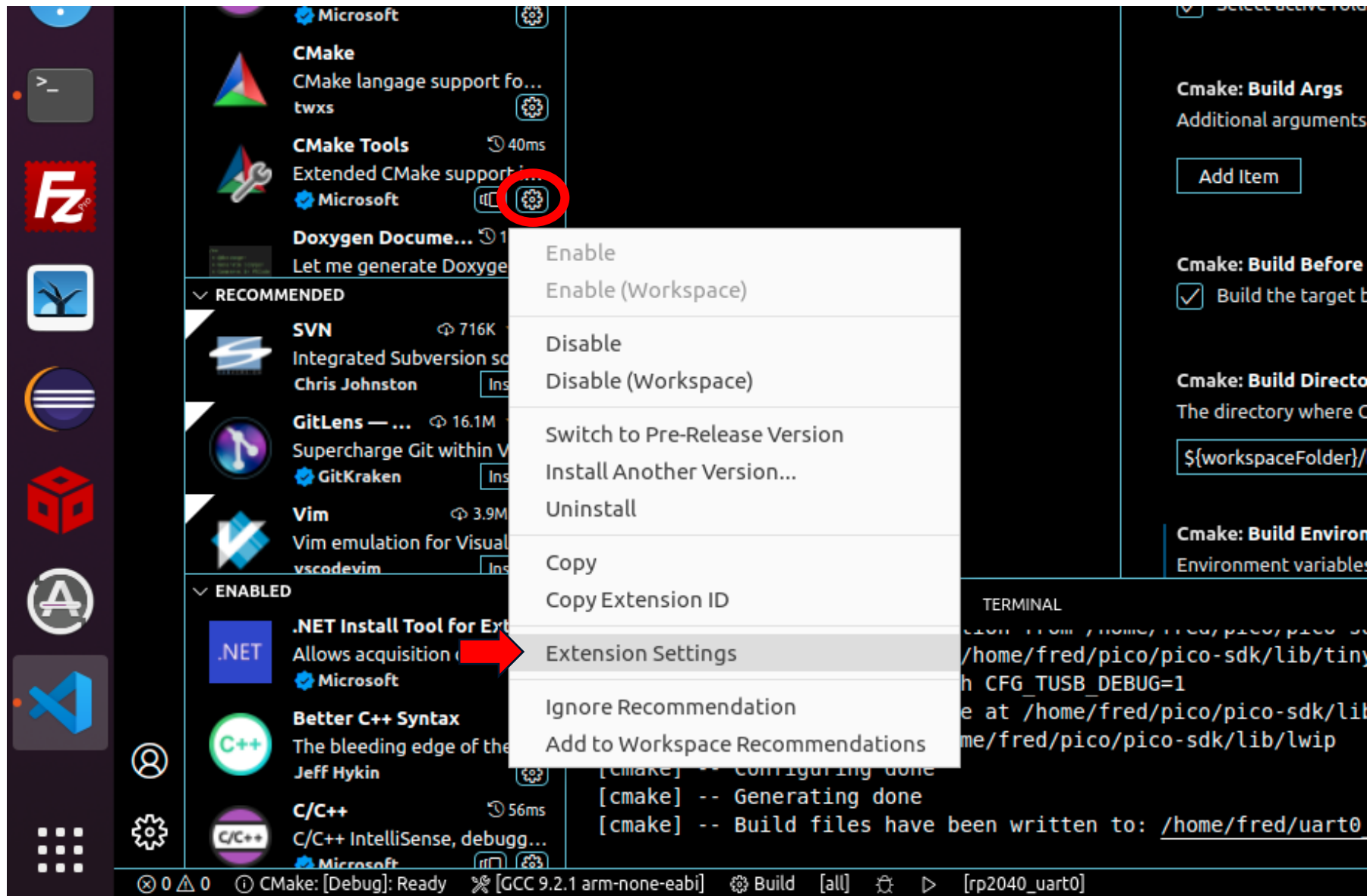


```
fred@shopUbuntu885: ~/Downloads
File Edit View Search Terminal Help
CCS11.2.0.00007_linux-x64
CCS11.2.0.00007_linux-x64.tar.gz
code_1.68.0-1654690107_amd64.deb
code_1.68.1-1655263094_amd64.deb
code_1.69.2-1658162013_amd64.deb
eclipse-inst-jre-linux64.tar.gz
en.STM32Cube_FW_MP1_V1-5-0_v1.5.0.zip
en.stm32cubeprg-lin_v2-9-0_v2.9.0.zip
en.st-stm32cubeide_1.8.0_11526_20211125_0815_amd64.sh_v1.8.0
en.st-stm32cubeide_1.8.0_11526_20211125_0815_amd64.sh_v1.8.0.zip
FileZilla_Pro_3.56.2_x86_64-linux-gnu.tar.bz2
FileZilla_Pro_3.57.1_x86_64-linux-gnu.tar.bz2
FileZilla_Pro_3.59.0_x86_64-linux-gnu.tar.bz2
gcc-arm-11.2-2022.02-x86_64-arm-none-eabi.tar.xz
ipscan_3.7.6_amd64.deb
jetbrains-toolbox-1.25.12424.tar.gz
JLink_Linux_V768b_x86_64.deb
libgplod-1.6.3.tar.gz
Ozone_Linux_V326h_x86_64.deb
pico-project-generator-master.zip
Setup_EmbeddedStudio_ARM_v630_linux_x64.tar.gz
stmp1_pics
fred@shopUbuntu885:~/Downloads$ sudo apt install ./code_1.69.2-1658162013_amd64.deb
```

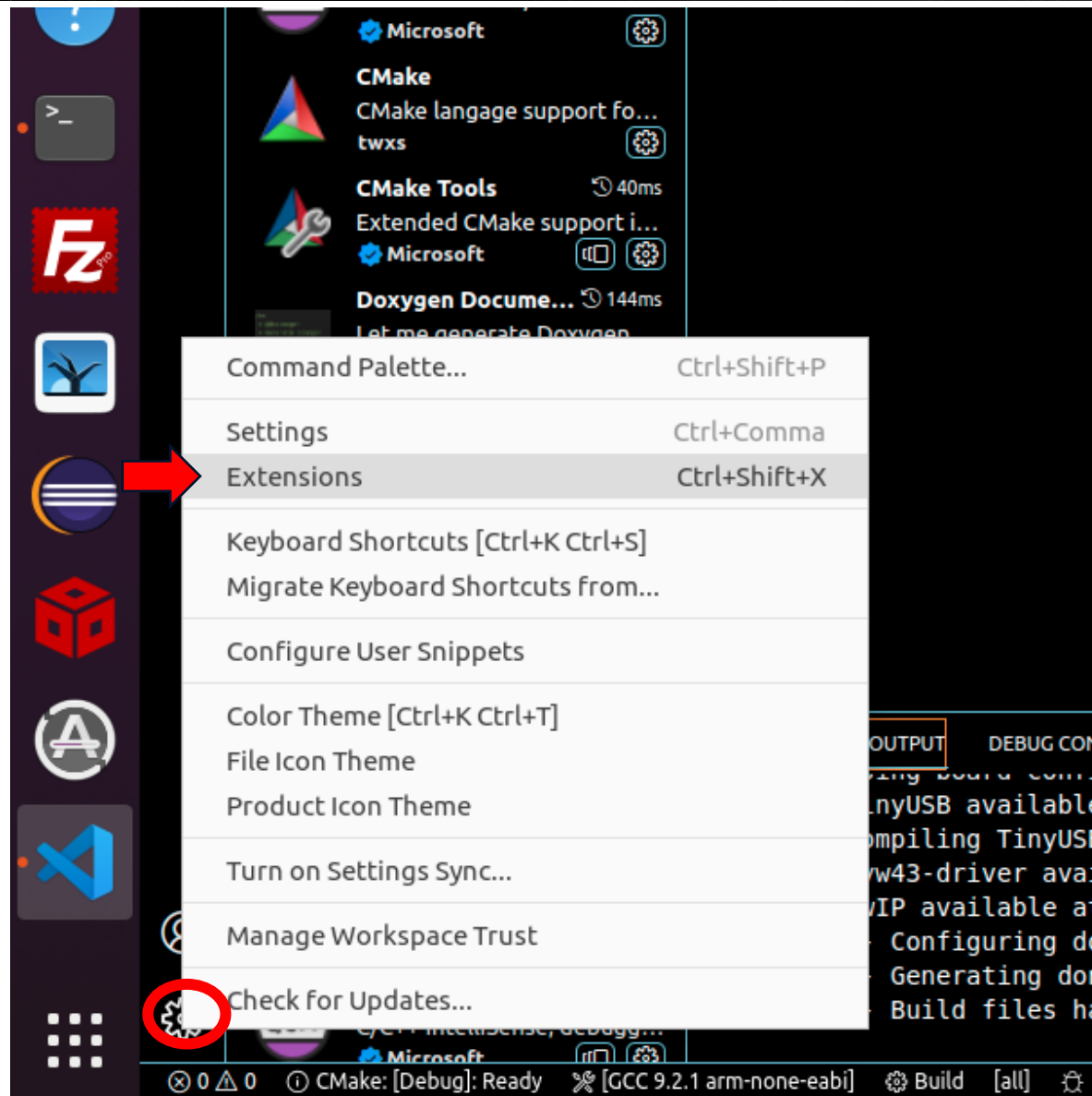
Visual Studio Code Extension Installation



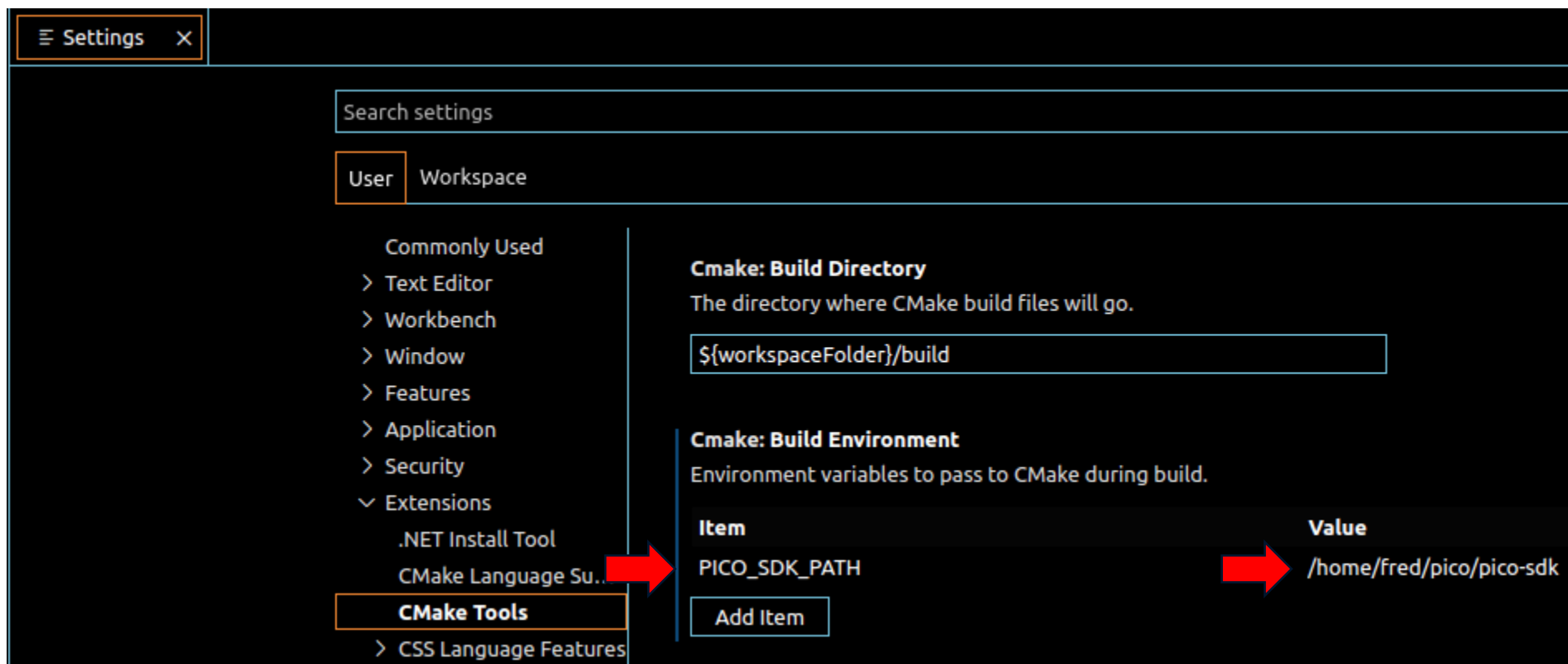
Visual Studio Code Extension Settings



Alternate Path to Extension Settings



Set Absolute Path to Pico SDK



The screenshot shows the Visual Studio Code Settings window. The 'Settings' tab is active. The 'User' settings are selected. The 'CMake Tools' extension is highlighted in the left sidebar. The 'Cmake: Build Directory' is set to `${workspaceFolder}/build`. The 'Cmake: Build Environment' section shows a table with one item, `PICO_SDK_PATH`, set to `/home/fred/pico/pico-sdk`. Red arrows indicate the navigation from the 'CMake Tools' extension in the sidebar to the 'Cmake: Build Environment' section and then to the `PICO_SDK_PATH` entry.

Settings

Search settings

User Workspace

Commonly Used

- > Text Editor
- > Workbench
- > Window
- > Features
- > Application
- > Security
- > Extensions
- .NET Install Tool
- CMake Language Su...
- CMake Tools**
- > CSS Language Features

Cmake: Build Directory
The directory where CMake build files will go.

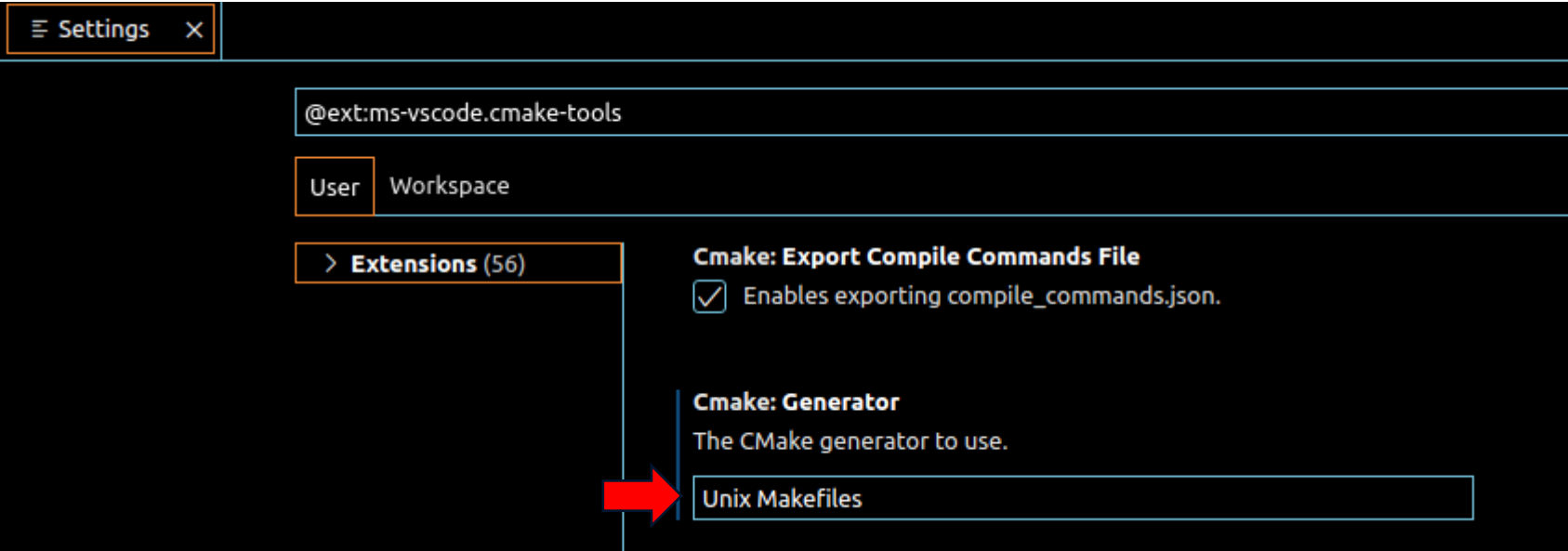
`${workspaceFolder}/build`

Cmake: Build Environment
Environment variables to pass to CMake during build.

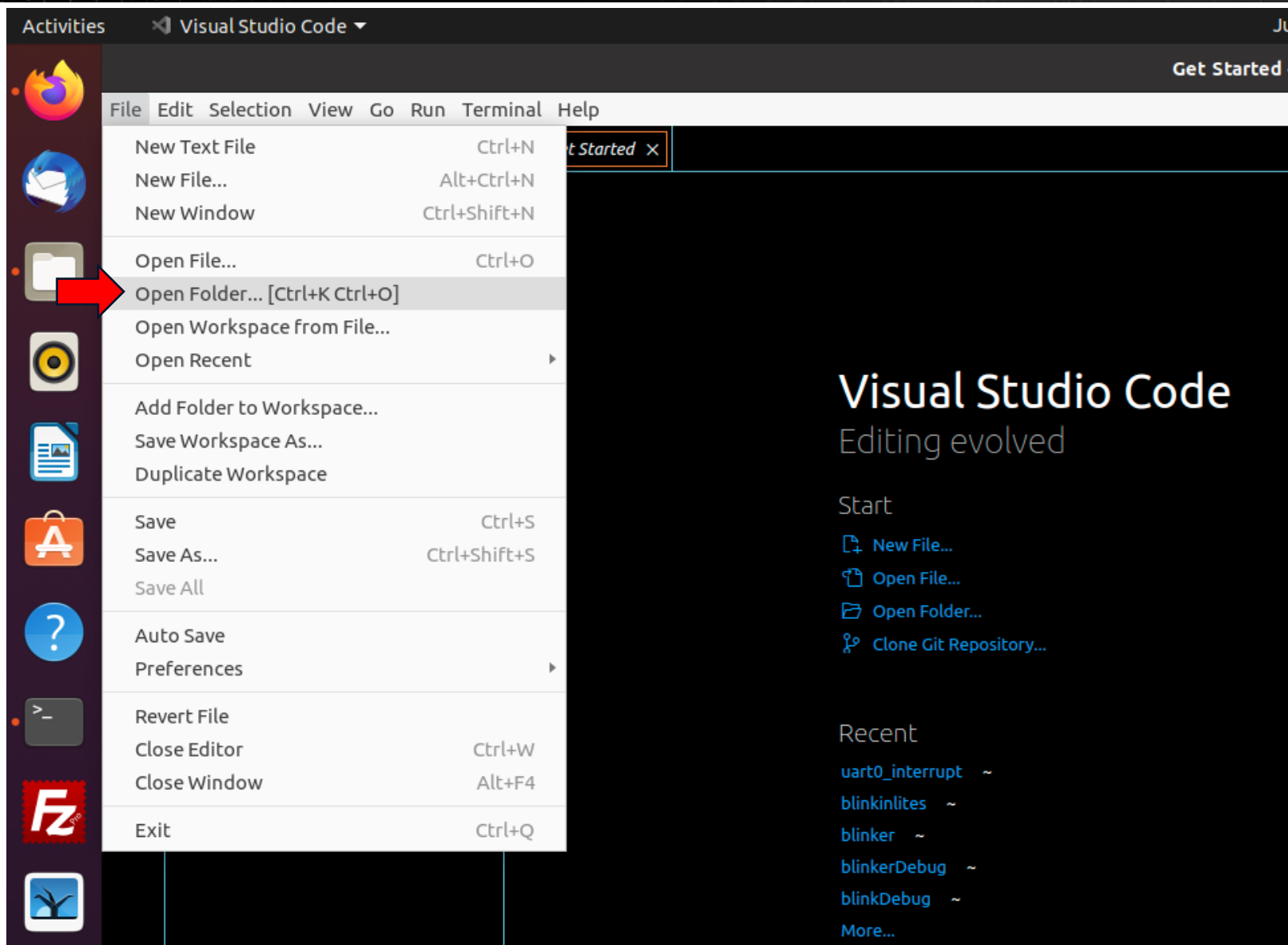
Item	Value
PICO_SDK_PATH	/home/fred/pico/pico-sdk

Add Item

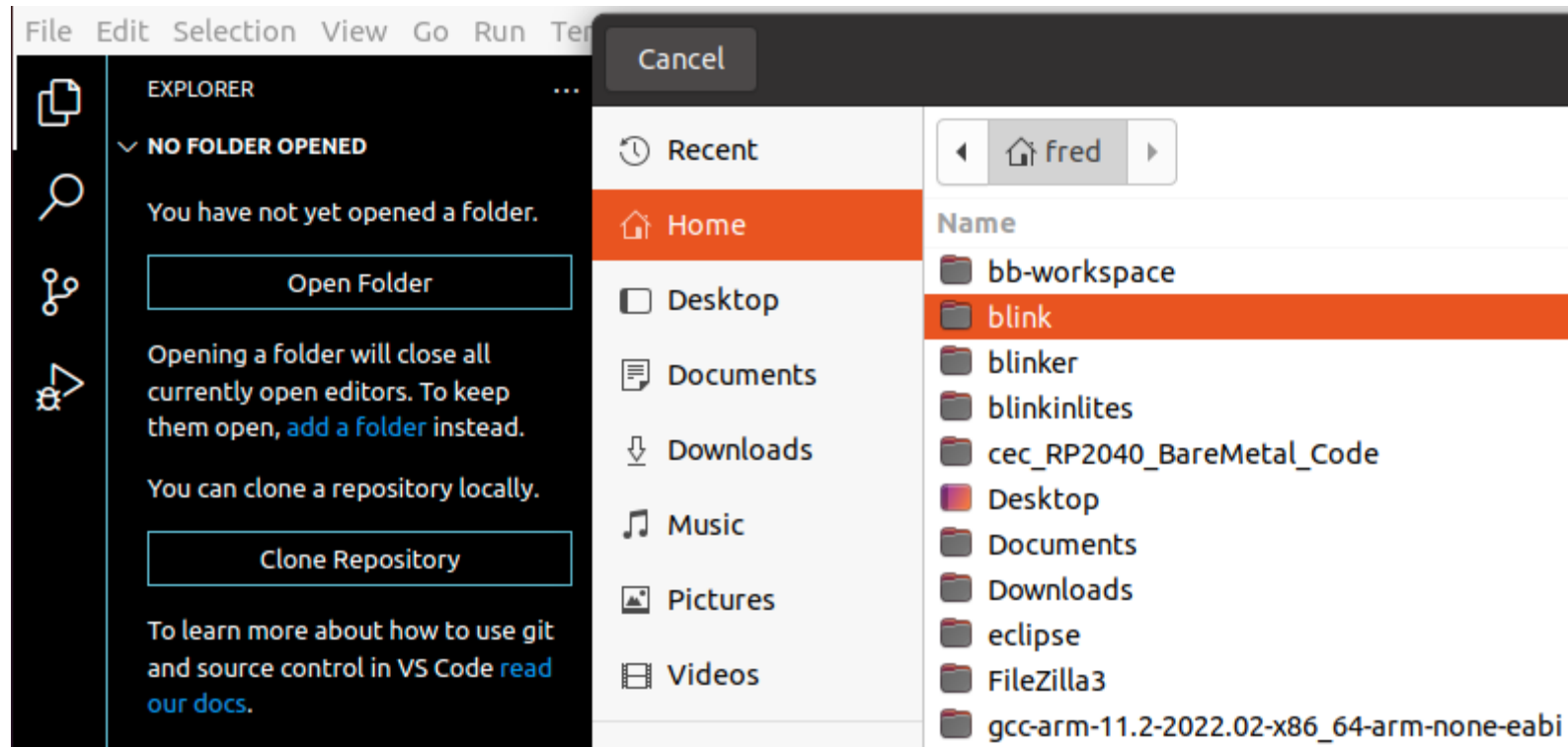
Specify Linux Build System



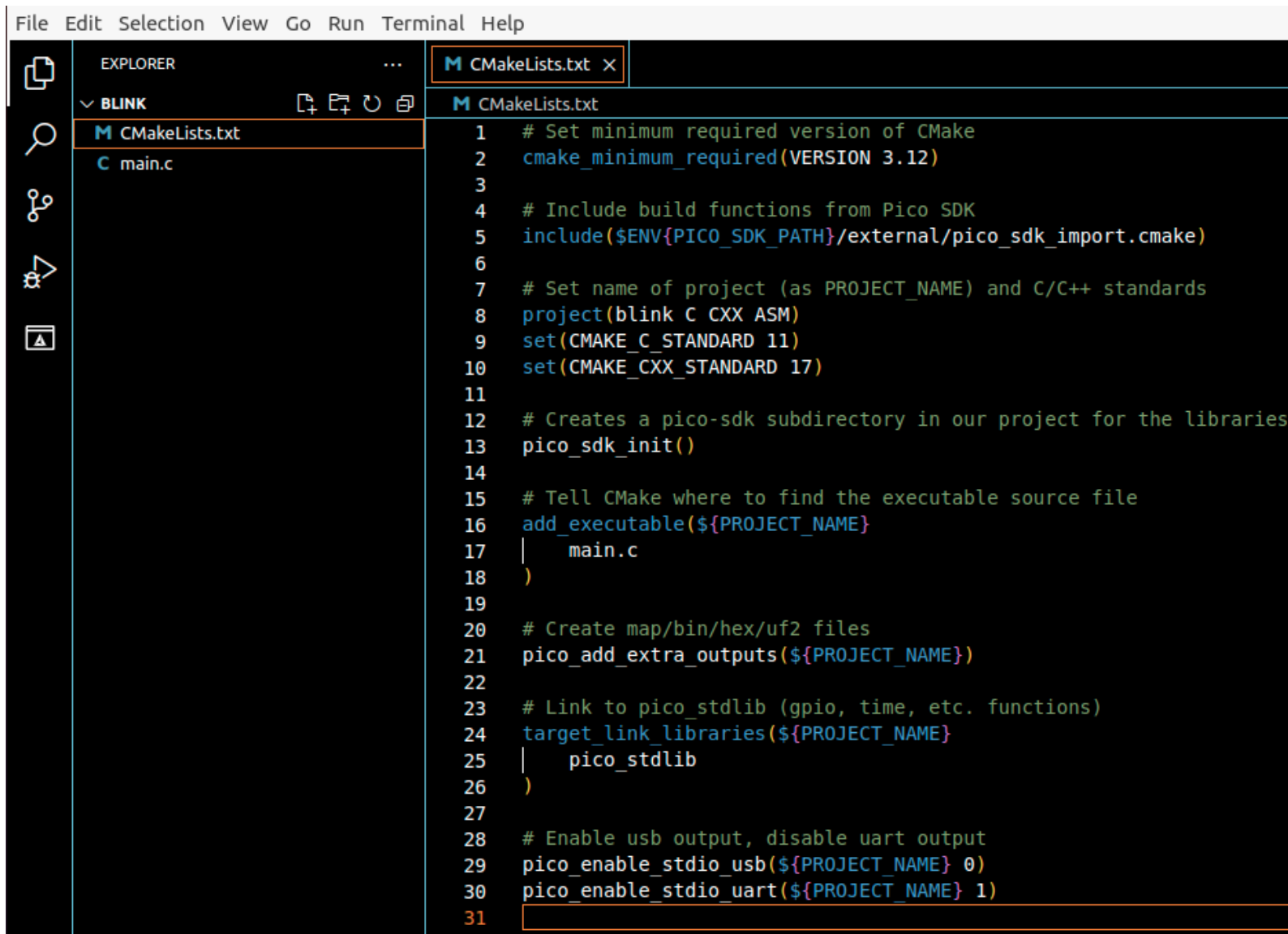
Open Project Folder



Open Project Folder



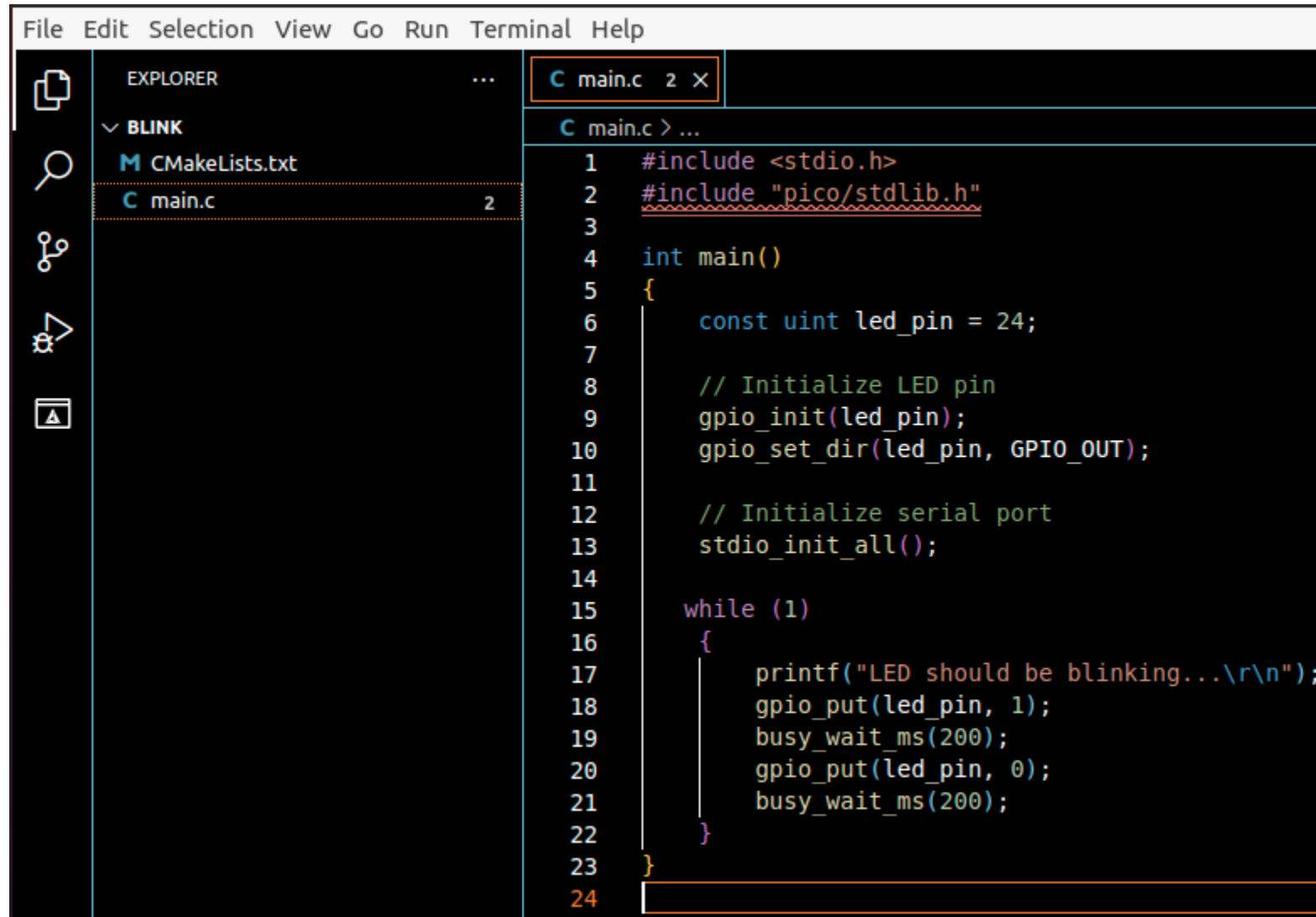
Create CMakeLists.txt



The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left. The Explorer sidebar shows a project named 'BLINK' with two files: 'CMakeLists.txt' and 'main.c'. The 'CMakeLists.txt' file is selected and its content is displayed in the main editor area. The code is a CMake script for a Pico project, setting the minimum required version of CMake to 3.12, including the Pico SDK build functions, setting the project name to 'blink C CXX ASM', and setting the C and C++ standards to 11 and 17 respectively. It also creates a 'pico-sdk' subdirectory, tells CMake where to find the executable source file 'main.c', creates map/bin/hex/uf2 files, links to the Pico stdlib, and enables USB output while disabling UART output.

```
File Edit Selection View Go Run Terminal Help
EXPLORER
  BLINK
    CMakeLists.txt
    main.c
  CMakeLists.txt x
  CMakeLists.txt
1  # Set minimum required version of CMake
2  cmake_minimum_required(VERSION 3.12)
3
4  # Include build functions from Pico SDK
5  include($ENV{PICO_SDK_PATH}/external/pico_sdk_import.cmake)
6
7  # Set name of project (as PROJECT_NAME) and C/C++ standards
8  project(blink C CXX ASM)
9  set(CMAKE_C_STANDARD 11)
10 set(CMAKE_CXX_STANDARD 17)
11
12 # Creates a pico-sdk subdirectory in our project for the libraries
13 pico_sdk_init()
14
15 # Tell CMake where to find the executable source file
16 add_executable(${PROJECT_NAME}
17 |   main.c
18 | )
19
20 # Create map/bin/hex/uf2 files
21 pico_add_extra_outputs(${PROJECT_NAME})
22
23 # Link to pico_stdlib (gpio, time, etc. functions)
24 target_link_libraries(${PROJECT_NAME}
25 |   pico_stdlib
26 | )
27
28 # Enable usb output, disable uart output
29 pico_enable_stdio_usb(${PROJECT_NAME} 0)
30 pico_enable_stdio_uart(${PROJECT_NAME} 1)
31
```

Create main.c



The screenshot shows a code editor interface with a menu bar (File, Edit, Selection, View, Go, Run, Terminal, Help) and a sidebar labeled 'EXPLORER'. The sidebar shows a project named 'BLINK' with two files: 'CMakeLists.txt' and 'main.c'. The 'main.c' file is selected and its content is displayed in the main editor area. The code is a C program for blinking an LED on a Raspberry Pi Pico. It includes the standard C library and the Pico standard library, initializes the LED pin (24) and the serial port, and then enters a loop that prints a message and toggles the LED state every 200ms.

```
File Edit Selection View Go Run Terminal Help

EXPLORER
  BLINK
    CMakeLists.txt
    main.c

C main.c 2 X

C main.c > ...
1  #include <stdio.h>
2  #include "pico/stdlib.h"
3
4  int main()
5  {
6      const uint led_pin = 24;
7
8      // Initialize LED pin
9      gpio_init(led_pin);
10     gpio_set_dir(led_pin, GPIO_OUT);
11
12     // Initialize serial port
13     stdio_init_all();
14
15     while (1)
16     {
17         printf("LED should be blinking...\r\n");
18         gpio_put(led_pin, 1);
19         busy_wait_ms(200);
20         gpio_put(led_pin, 0);
21         busy_wait_ms(200);
22     }
23 }
24
```

Select the Compiler Kit

main.c - blink - Visual Studio Code

main.c 2 x

C main.c > ...

```
1 #include <stdio.h>
2 #include "pico/stdlib.h"
3
4 int main()
5 {
6     const uint led_pin = 24;
7
8     // Initialize LED pin
9     gpio_init(led_pin);
10    gpio_set_dir(led_pin, GPIO_OUT);
11
12    // Initialize serial port
13    stdio_init_all();
14
15    while (1)
16    {
17        printf("LED should be blinking...\r\n");
18        gpio_put(led_pin, 1);
19        busy_wait_ms(200);
20        gpio_put(led_pin, 0);
21        busy_wait_ms(200);
22    }
23 }
24
```

Select a Kit for blink

- [Scan for kits] Search for compilers on this computer
- [Unspecified] Unspecified (Let CMake guess what compilers and environment to use)
- GCC 10.3.0 x86_64-linux-gnu Using compilers: C = /usr/bin/gcc-10
- GCC 11.2.1 arm-none-eabi Using compilers: C = /usr/bin/arm-none-eabi-gcc, CXX = /usr/bin/arm-none-eabi-g++
- GCC 9.2.1 arm-none-eabi Using compilers: C = /usr/bin/arm-none-eabi-gcc, CXX = /usr/bin/arm-none-eabi-g++
- GCC 9.4.0 arm-linux-gnueabi Using compilers: C = /usr/bin/arm-linux-gnueabi-gcc, CXX = /usr/bin/arm-linux-gnueabi-g++
- GCC 9.4.0 arm-linux-gnueabihf Using compilers: C = /usr/bin/arm-linux-gnueabihf-gcc, CXX = /usr/bin/arm-linux-gnueabihf-g++
- GCC 9.4.0 x86_64-linux-gnu Using compilers: C = /usr/bin/gcc, CXX = /usr/bin/g++

> OUTLINE

> TIMELINE

[cmake] -- Build files have

0 0 CMake: [Debug]: Ready [GCC 9.2.1 arm-none-eabi] Build [all]

Execute CMake

main.c - blink - Visual Studio Code

terminal Help

C main.c 2 X

C main.c > ...

```
1 #include <stdio.h>
2 #include "pico/stdlib.h"
3
4 int main()
5 {
6     const uint led_pin = 24;
7
8     // Initialize LED pin
9     gpio_init(led_pin);
10    gpio_set_dir(led_pin, GPIO_OUT);
11
12    // Initialize serial port
13    stdio_init_all();
14
15    while (1)
16    {
17        printf("LED should be blinking...\r\n");
18        gpio_put(led_pin, 1);
19        busy_wait_ms(200);
20        gpio_put(led_pin, 0);
21        busy_wait_ms(200);
22    }
23 }
24
```

Select a build variant

- Debug Disable optimizations - include debug information.
- Release Optimize for speed - exclude debug information.
- MinSizeRel Optimize for smallest binary size - exclude debug information.
- RelWithDebInfo Optimize for speed - include debug information.

> OUTLINE

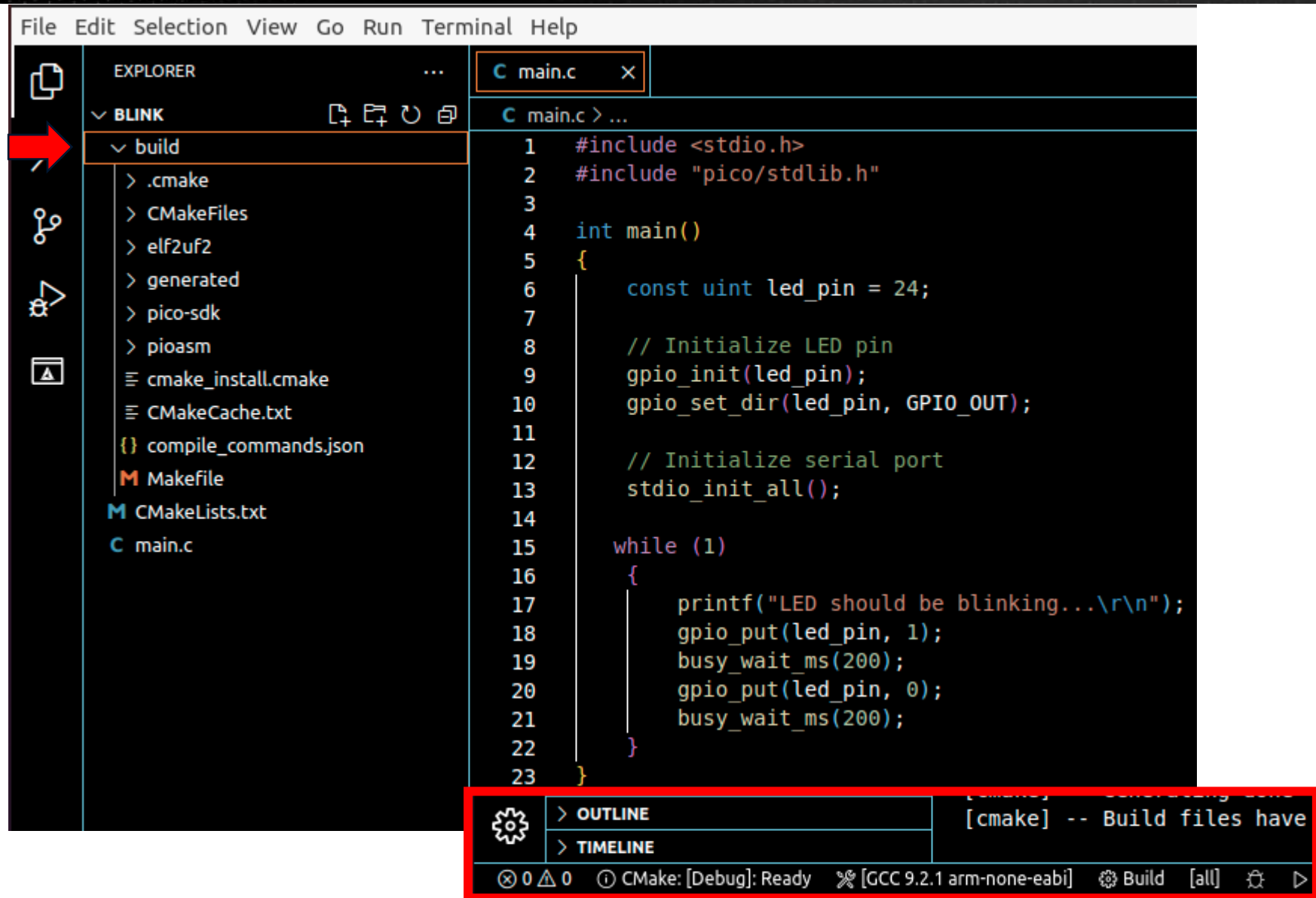
> 1. LINE

[CMake] Generating build files

[cmake] -- Build files have

0 0 CMake: [Debug]: Ready [GCC 9.2.1 arm-none-eabi] Build [all]

Execute Cmake Results

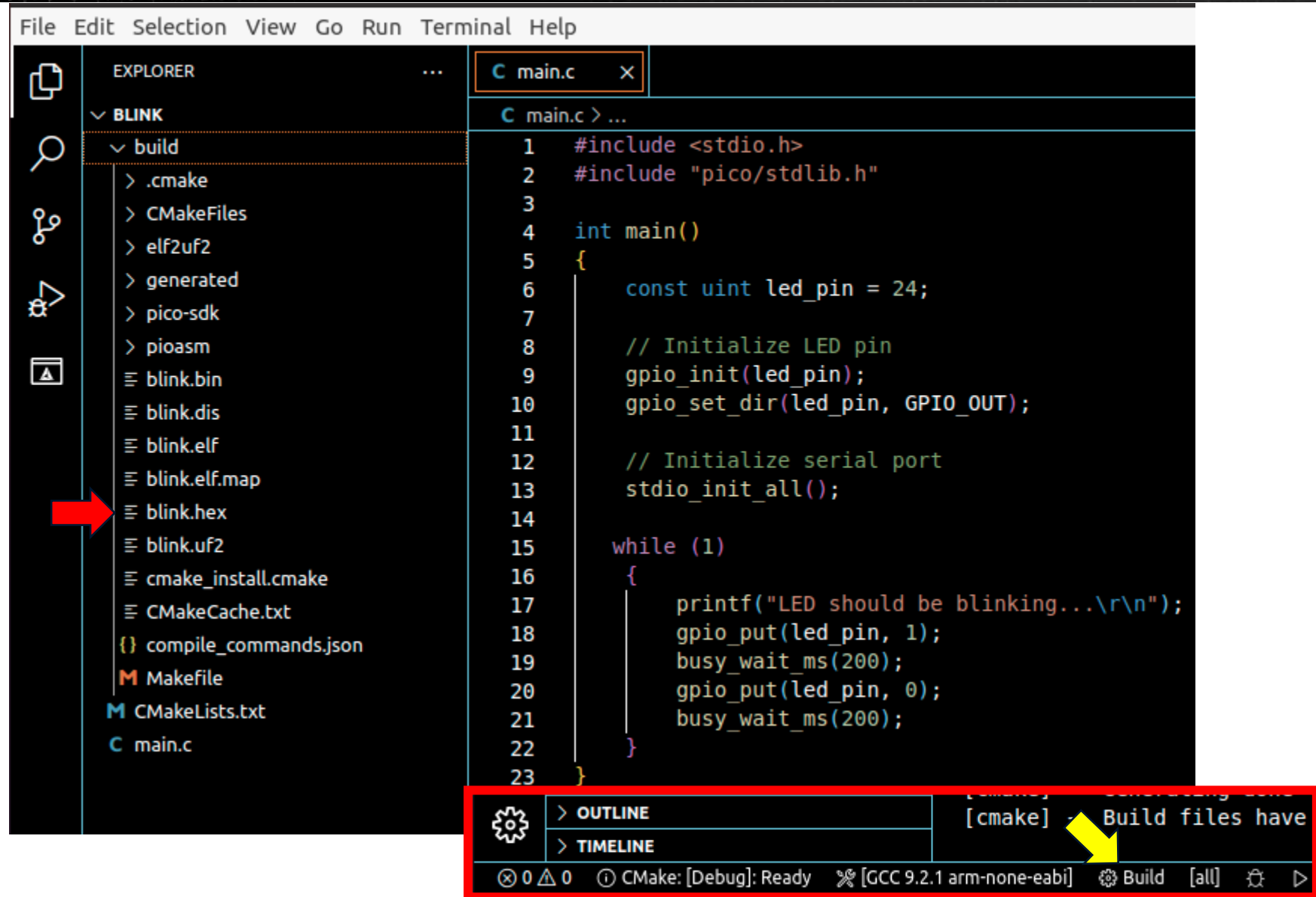


The screenshot shows the Visual Studio Code interface with the following components:

- EXPLORER:** The file tree on the left shows a project named "BLINK" with a "build" folder expanded. A red arrow points to the "build" folder. Other files listed include ".cmake", "CMakeFiles", "elf2uf2", "generated", "pico-sdk", "pioasm", "cmake_install.cmake", "CMakeCache.txt", "compile_commands.json", "Makefile", "CMakeLists.txt", and "main.c".
- EDITOR:** The main editor window displays the "main.c" file. The code is as follows:

```
1  #include <stdio.h>
2  #include "pico/stdlib.h"
3
4  int main()
5  {
6      const uint led_pin = 24;
7
8      // Initialize LED pin
9      gpio_init(led_pin);
10     gpio_set_dir(led_pin, GPIO_OUT);
11
12     // Initialize serial port
13     stdio_init_all();
14
15     while (1)
16     {
17         printf("LED should be blinking...\r\n");
18         gpio_put(led_pin, 1);
19         busy_wait_ms(200);
20         gpio_put(led_pin, 0);
21         busy_wait_ms(200);
22     }
23 }
```
- OUTPUT:** The bottom panel shows the "OUTLINE" view with the following text: "[cmake] -- Build files have".
- STATUS BAR:** The bottom status bar shows "0 0", "CMake: [Debug]: Ready", "[GCC 9.2.1 arm-none-eabi]", "Build", "[all]", and a play button icon.

Execute Build



File Edit Selection View Go Run Terminal Help

EXPLORER

BLINK

build

- > .cmake
- > CMakeFiles
- > elf2uf2
- > generated
- > pico-sdk
- > pioasm
- ≡ blink.bin
- ≡ blink.dis
- ≡ blink.elf
- ≡ blink.elf.map
- ≡ blink.hex
- ≡ blink.uf2
- ≡ cmake_install.cmake
- ≡ CMakeCache.txt
- { } compile_commands.json
- Makefile
- CMakeLists.txt
- main.c

C main.c

```
1 #include <stdio.h>
2 #include "pico/stdlib.h"
3
4 int main()
5 {
6     const uint led_pin = 24;
7
8     // Initialize LED pin
9     gpio_init(led_pin);
10    gpio_set_dir(led_pin, GPIO_OUT);
11
12    // Initialize serial port
13    stdio_init_all();
14
15    while (1)
16    {
17        printf("LED should be blinking...\r\n");
18        gpio_put(led_pin, 1);
19        busy_wait_ms(200);
20        gpio_put(led_pin, 0);
21        busy_wait_ms(200);
22    }
23 }
```

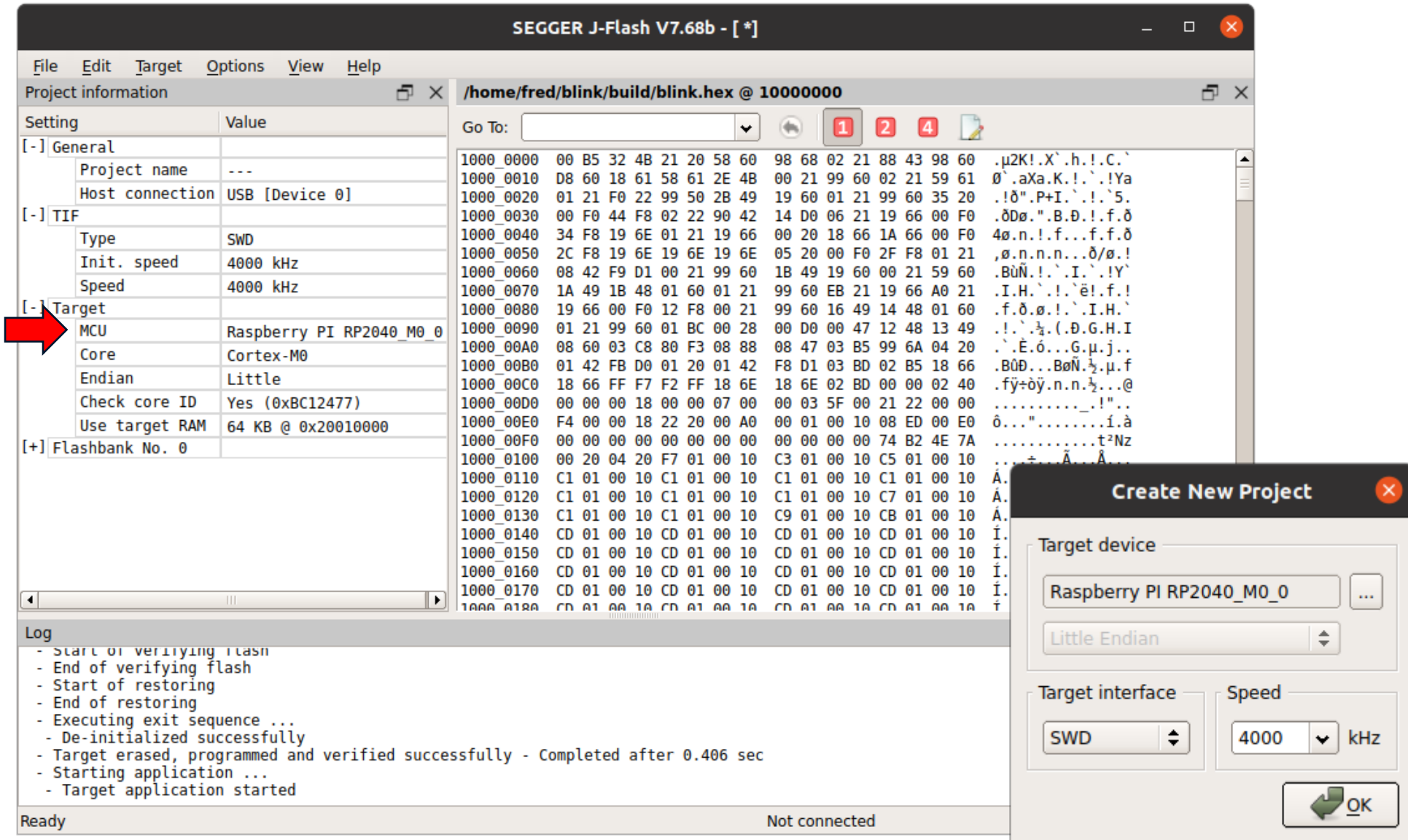
OUTLINE

TIMELINE

[cmake] Build files have

Build [all]

Program the RP2040



The screenshot displays the SEGGER J-Flash V7.68b interface. The main window shows the project information for a file named `/home/fred/blink/build/blink.hex @ 10000000`. The 'Target' section is expanded, showing the configuration for the Raspberry PI RP2040_M0_0 target. A red arrow points to the 'Target' section. The 'Flashbank No. 0' is also visible. The 'Log' section at the bottom shows the progress of the flashing process, including 'Start of verifying flash', 'End of verifying flash', 'Start of restoring', 'End of restoring', 'Executing exit sequence ...', 'De-initialized successfully', 'Target erased, programmed and verified successfully - Completed after 0.406 sec', 'Starting application ...', and 'Target application started'.

The 'Create New Project' dialog box is open, showing the 'Target device' as 'Raspberry PI RP2040_M0_0' and the 'Target interface' as 'SWD'. The 'Speed' is set to '4000 kHz'. The 'Endianness' is set to 'Little Endian'. The 'OK' button is highlighted.

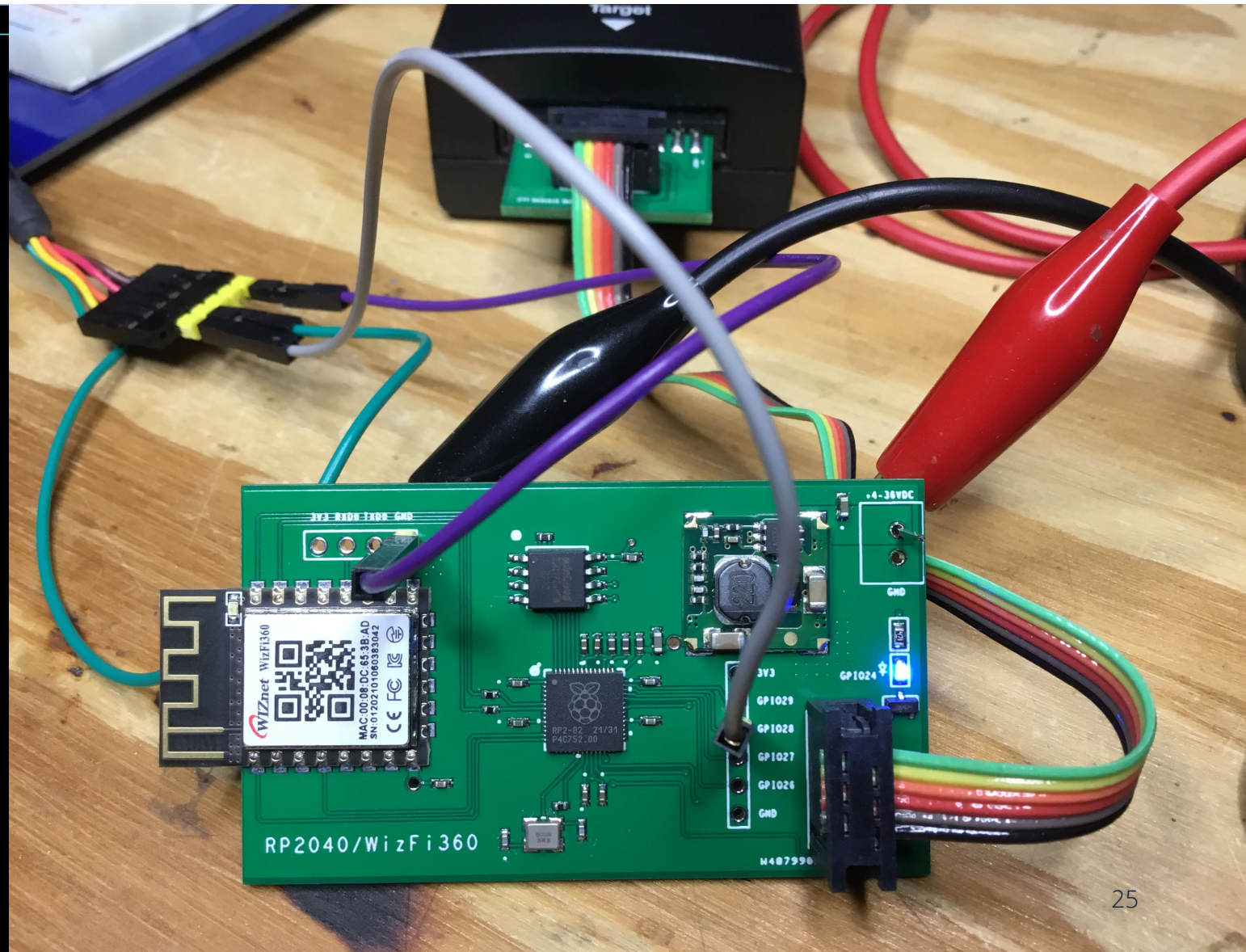
Log

- Start of verifying flash
- End of verifying flash
- Start of restoring
- End of restoring
- Executing exit sequence ...
- De-initialized successfully
- Target erased, programmed and verified successfully - Completed after 0.406 sec
- Starting application ...
- Target application started

Ready Not connected

Enable Alternate UART0

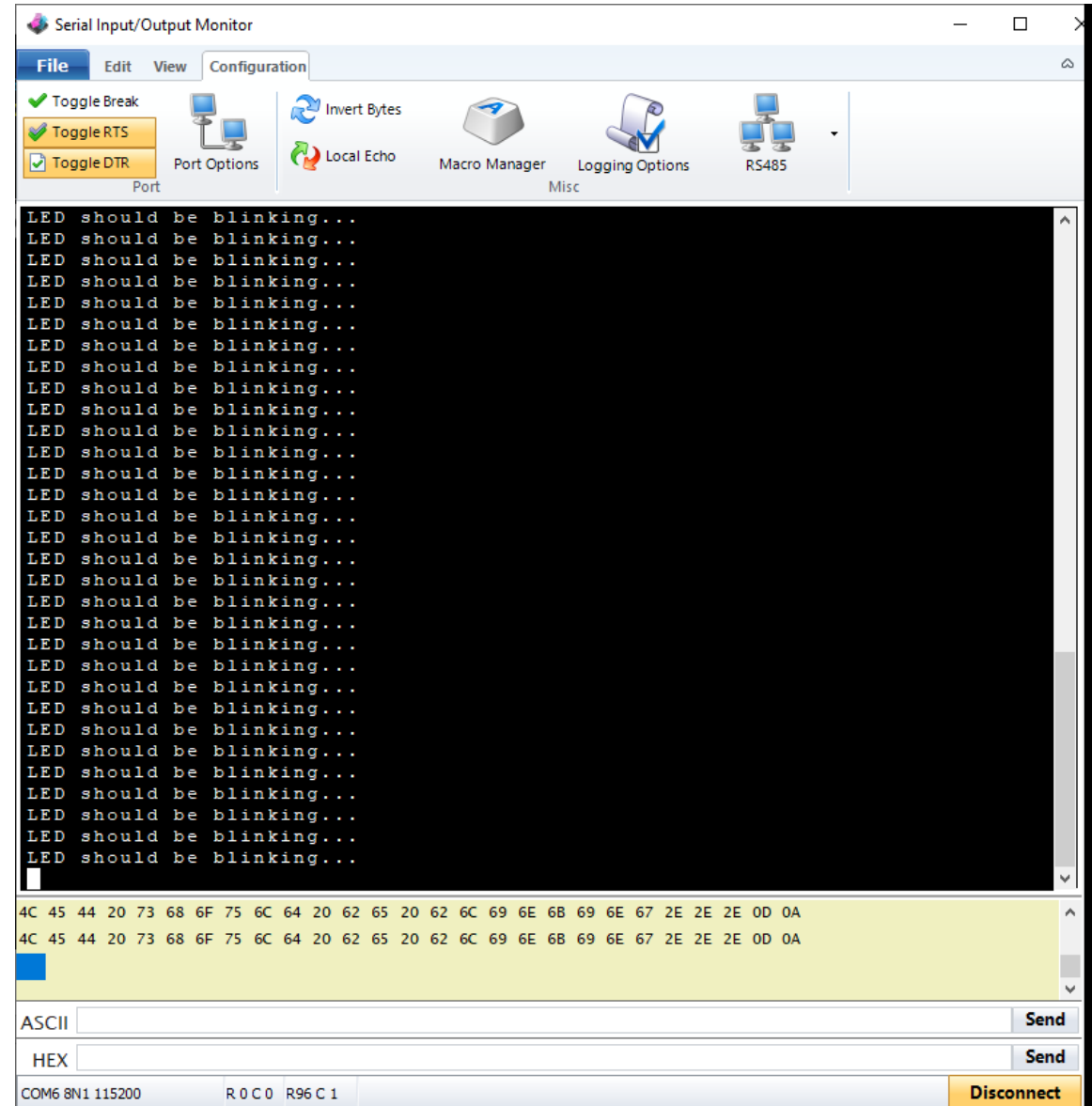
```
C main.c > ...
1  #include "pico/stdlib.h"
2
3  #define UART0_ID  uart0
4  #define BAUD_RATE 115200
5  #define DATA_BITS 8
6  #define STOP_BITS 1
7  #define PARITY     UART_PARITY_NONE
8
9  // define uart pins
10 #define UART0_TX_PIN 28
11 #define UART0_RX_PIN 29
12 #define LED_PIN      24
13
14 int main()
15 {
16     // Initialize LED pin
17     gpio_init(LED_PIN);
18     gpio_set_dir(LED_PIN, GPIO_OUT);
19
20     // Initialize serial port
21     gpio_set_function(UART0_TX_PIN, GPIO_FUNC_UART);
22     gpio_set_function(UART0_RX_PIN, GPIO_FUNC_UART);
23     uart_init(UART0_ID, 115200);
24     uart_set_hw_flow(UART0_ID, false, false);
25     uart_set_format(UART0_ID, DATA_BITS, STOP_BITS, PARITY);
26     uart_set_fifo_enabled(UART0_ID, false);
27
28     while (1)
29     {
30         uart_puts(UART0_ID, "LED should be blinking...\r\n");
31         gpio_put(LED_PIN, 1);
32         busy_wait_ms(200);
33         gpio_put(LED_PIN, 0);
34         busy_wait_ms(200);
35     }
36 }
```



Enable Alternate UART0

C main.c > ...

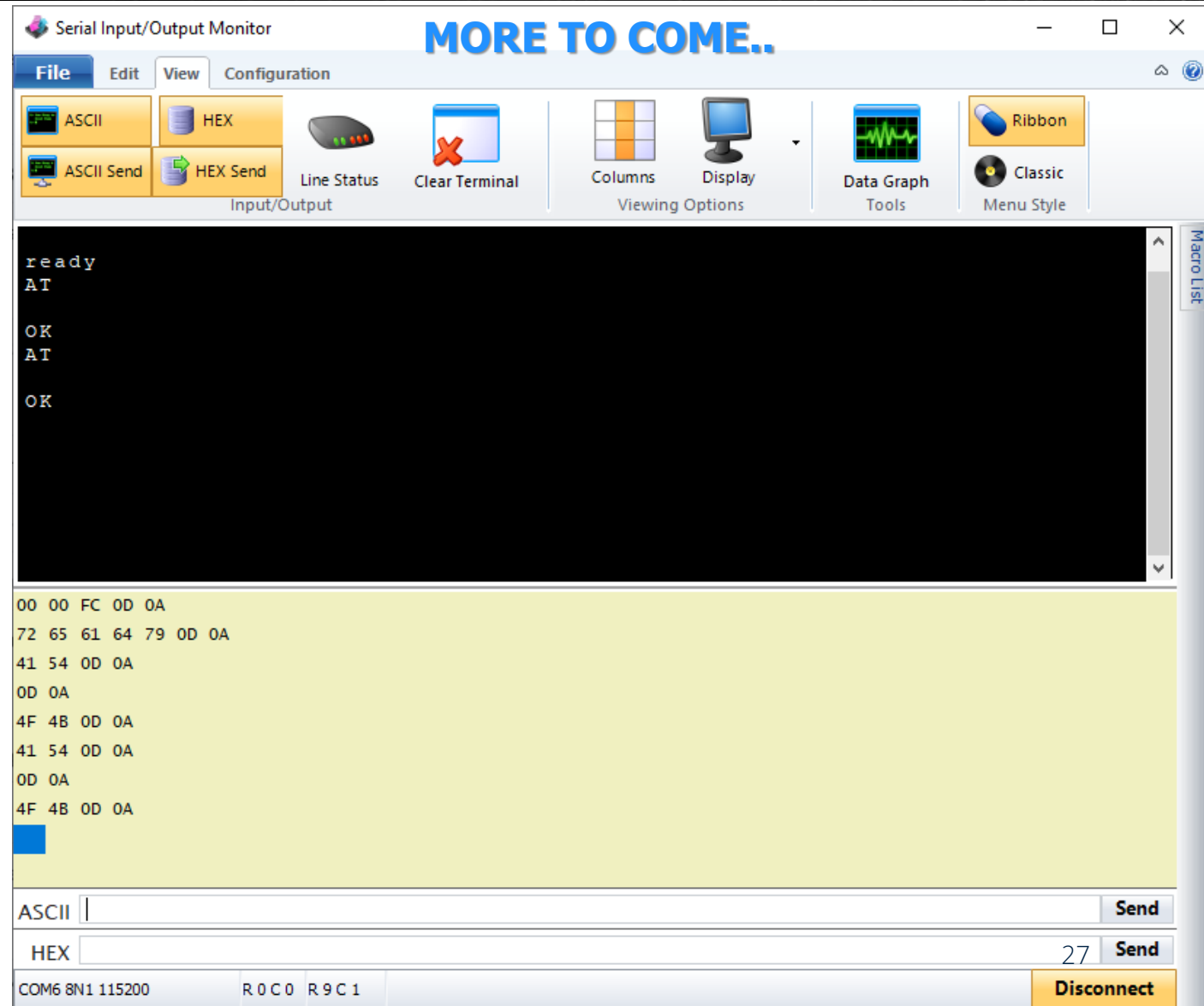
```
1  #include "pico/stdlib.h"
2
3  #define UART0_ID    uart0
4  #define BAUD_RATE  115200
5  #define DATA_BITS 8
6  #define STOP_BITS  1
7  #define PARITY      UART_PARITY_NONE
8
9  // define uart pins
10 #define UART0_TX_PIN 28
11 #define UART0_RX_PIN 29
12 #define LED_PIN      24
13
14 int main()
15 {
16     // Initialize LED pin
17     gpio_init(LED_PIN);
18     gpio_set_dir(LED_PIN, GPIO_OUT);
19
20     // Initialize serial port
21     gpio_set_function(UART0_TX_PIN, GPIO_FUNC_UART);
22     gpio_set_function(UART0_RX_PIN, GPIO_FUNC_UART);
23     uart_init(UART0_ID, 115200);
24     uart_set_hw_flow(UART0_ID, false, false);
25     uart_set_format(UART0_ID, DATA_BITS, STOP_BITS, PARITY);
26     uart_set_fifo_enabled(UART0_ID, false);
27
28     while (1)
29     {
30         uart_puts(UART0_ID, "LED should be blinking...\r\n");
31         gpio_put(LED_PIN, 1);
32         busy_wait_ms(200);
33         gpio_put(LED_PIN, 0);
34         busy_wait_ms(200);
35     }
36 }
```



Thank you for attending!!!

Please consider the resources below:

- raspberrypi.org
- **RP2040 Datasheet**
- **Raspberry Pi Pico C/C++ SDK**
- **SEGGER J-Link**





DesignNews

Thank You

Sponsored by

