Raspberry Pi Pico 入门

使用 Raspberry Pi Pico 和其他基于 RP2040 的微控制器板进行 C / C ++开发

树莓派贸易有限公司

科洛芬

版权所有©2020 Raspberry Pi（Trading）Ltd.

RP2040 微控制器的文档已根据"知识共享署名-无衍生品 4.0 国际协议（CC BY-ND）"获得许可。

建立日期：2021-01-21

构建版本：fcd04ef-clean

法律免责声明

RASPBERRY PI（TRADING）LTD（" RPTL"）按现状提供不时修改的 RASPBERRY PI 产品的技术和可靠性数据（包括数据表），按"现状"以及任何明示或暗示的担保，包括但不限于不仅限于此，对针对特定目的的适销性和适用性的默示担保不承担任何责任。在任何情况下，RPTL 在任何情况下都不会在适用法律允许的最大范围内对任何直接，间接，偶发，特殊，特殊或继发性损害（包括但不限于，购买替代商品或服务，使用权损失）承担责任，或利润；或业务中断），无论是由于使用资源，以任何方式订立的合同，严格责任或侵权（包括过失或其他）侵权责任，

RPTL 保留随时对资源或其中描述的任何产品进行任何增强，改进，更正或任何其他修改的权利，恕不另行通知。

该资源面向具有适当设计知识水平的熟练用户。用户对资源的选择和使用以及其中所述产品的任何应用负全责。用户同意对 RPTL 进行赔偿，并使 RPTL 免受因使用资源引起的所有责任，成本，损害或其他损失。

RPTL 授予用户仅与 Raspberry Pi 产品一起使用资源的权限。禁止对资源进行其他所有使用。没有任何其他 RPTL 或其他第三方知识产权的许可。

高风险活动。Raspberry Pi 产品并非为需要故障安全性能的危险环境而设计，制造或设计的，例如在核设施，飞机导航或通信系统，空中交通管制，武器系统或对安全至关重要的应用（包括生命支持）的操作中系统和其他医疗设备），其中产品的故障可能直接导致死亡，人身伤害或严重的物理或环境破坏（"高风险活动"）。RPTL 特别否认对高风险活动的适用性提供任何明示或暗示的保证，并且对于在高风险活动中使用或包含 Raspberry Pi 产品不承担任何责任。

提供 Raspberry Pi 产品须遵守 RPTL 的标准条款。RPTL 的资源条款不会扩展或修改 RPTL 的标准条款，包括但不限于其中的免责声明和保证。

目录

Colophon 1
法律免责声明 1
1. 快速 Pico 设置 4
2. Pico SDK 5
2.1. 获取 Pico SDK 和示例 5
2.2. 安装工具链 6
3. 用 C 闪烁 LED 7
3.1. 建立"眨眼" 7
3.2. 加载并运行"闪烁" 8
3.2.1. 从桌面上 9
3.2.2. 使用命令行 9
4. 在 C 中说" Hello World" 11
4.1. Raspberry Pi Pico 上的串行输入和输出 11
4.2. 建立" Hello World" 12
4.3. 闪烁并运行" Hello World" 13
4.4. 参见" Hello World" USB 输出 13
4.5. 参见" Hello World" UART 输出 ..... 。 14
4.6. 为电路板供电 15
5. 用 SWD 调试 ..... 。 17
5.1. 生成" Hello World"调试 版本 17
5.2. 安装 OpenOCD 17
5.3. 安装 GDB ..... 18 岁
5.4. 使用 GDB 和 OpenOCD 调试 Hello World 19
6. 使用 Visual Studio 代码 21
6.1. 安装 Visual Studio 代码 21
6.2. 加载项目 21
6.3. 调试项目 23
6.3.1. 在 Raspberry Pi Pico 上运行" Hello World" ..... 。 。 24
7. 创建您自己的项目 26
7.1. 调试您的项目 28
7.2. 在 Visual Studio Code 中工作 29
7.3. 自动化项目创建 29
7.3.1. 通过命令行生成项目 31
8. 在其他平台上构建 32
8.1. 在 Apple macOS 上构建 32
8.1.1. 安装工具链 32
8.1.2. 使用 Visual Studio 代码 32
8.1.3. 用 CMake 工具构建 32
8.1.4. 说" Hello World" 33
8.2. 在 MS Windows 上构建 34
8.2.1. 安装工具链 34
8.2.2. 获取 Pico SDK 和示例 37
8.2.3. 从指挥大楼的"Hello World" 行 37
8.2.4. 从 Visual Studio Code 构建" Hello World" 38
8.2.5. 闪烁并运行" Hello World" 39
9. 使用其他集成开发环境 41
9.1. 使用 Eclipse ..... 。 41
9.1.1. 在 Linux 机器上设置 Eclipse for Pico 41
9.2. 使用 CLion 45
9.2.1. 设置 CLion 46
9.3. 其他环境 49
9.3.1. 使用 openocd-svd 49
附录 A：使用 Picoprobe 52
A.1。建立 OpenOCD 52
A.1.1。Linux 52

A.1.2。Windows 52
A.1.3。Mac 54
A2。构建并刷新 picoprobe 55
A.3。Picoprobe 接线 55
A.4。安装 Picoprobe 驱动程序（仅在 Windows 上需要） 56
A.5。使用 Picoprobe 的 UART 57
A.5.1。Linux 57
A.5.2。Windows 57
A. 5.3。Mac 58
A. 6.在 OpenOCD 中使用 Picoprobe 59
附录 B：使用 Picotool 60
B. 1.获取 picotool 60
B.2。建筑 picotool 60
B.3。使用 picotool ..... 。61
B. 3.1。显示信息 61
B.3.2。保存程序 63
B.4。二进制信息 64
B.4.1。基本信息 64
B.4.2。销钉 65
B.4.3。包括二进制信息 65
B.4.4。细节 66
B.4.5。从 CMake 设置公共字段 67

第 1 章 Pico 快速设置

如果要在 Raspberry Pi 4B 或 Raspberry Pi 400 上为 Raspberry Pi Pico 开发，则可以通过运行安装脚本来跳过本《入门指南》中的大多数安装步骤。您可以通过执行以下操作获取此脚本：

```
$ git 的克隆 https://github.com/raspberrypi/pico-setup.git ®
```

1.如果尚未安装 Git，则应先 sudo apt install git 。

然后跑

```
$ pico-setup / pico_setup.sh
```

该脚本将：

- 创建一个名为 pico 的目录
- 安装所需的依赖项
- 下载 pico-sdk ， pico-examples ， pico-extras 和 pico-playground 存储库
- 定义 `PICO_SDK_PATH` ，`PICO_EXAMPLES_PATH` ，`PICO_EXTRAS_PATH` ，并 `PICO_PLAYGROUND_PATH` 在你的~/ .bashrc
- 在 pico-examples / build / blink 和 pico-examples / build / hello_world 中构建眨眼 和 hello_world 示例
- 下载并构建 picotool （请参阅附录 B ）。将其复制到/ usr / local / bin 。
- 下载并构建 picoprobe （请参阅附录 A ）。
- 下载并编译 OpenOCD（用于调试支持）
- 下载并安装 Visual Studio Code
- 安装所需的 Visual Studio Code 扩展（有关更多详细信息，请参见第 6 章）
- 配置 Raspberry Pi UART 以与 Raspberry Pi Pico 一起使用

一旦运行，您将需要重新启动 Raspberry Pi，

```
$ sudo 重新启动
```

使 UART 重新配置生效。

Raspberry Pi 重新启动后，您可以在"编程"菜单中打开 Visual Studio 代码，并按照 6.2 节中的说明进行操作。

第 2 章 Pico SDK

重要

以下说明假定您使用的是 Raspberry Pi Pico，并且如果使用其他基于 RP2040 的板，则某些细节可能会有所不同。他们还假定您正在使用在 Raspberry Pi 4 上运行的 Raspberry Pi OS，或在另一个平台上运行的等效的基于 Debian 的 Linux 发行版。还提供了使用 Microsoft Windows（请参阅第 8.2 节）或 Apple macOS（请参阅第 8.1 节）的用户的替代说明。

Raspberry Pi Pico 围绕 Raspberry Pi 设计的 RP2040 微控制器构建。C / C ++ SDK 和官方的 MicroPython 端口完全支持板上开发。本书讨论了如何开始使用 SDK，并引导您逐步构建，安装和使用 SDK 工具链。

“ TIP

有关官方 MicroPython 端口的更多信息，请参见 **Pico Python SDK** 手册，该手册记录了该端口，以及 Raspberry Pi Press 出版的 Gareth Halfacree 撰写的“**在 Raspberry Pi Pico 上使用 MicroPython 入门**”。

小费

有关 C / C ++ SDK 以及 API 级别文档的更多信息，请参阅 **Pico C / C ++ SDK** 书。

2.1. 获取 Pico SDK 和示例

所述微微实例 储存库（https://github.com/raspberrypi/pico-examples）提供了一组示例的应用程序正在使用的书面微微 SDK（https://github.com/raspberrypi/pico-sdk）。要克隆这些存储库，请先创建一个 pico 目录以保留所有与 pico 相关的结帐。这些说明在/ home / pi / pico 上创建了 pico 目录。

```
$ cd~/
$ mkdir pico
$ cd pico
```

然后克隆 pico-sdk 和 pico-examples git 存储库。

```
$ git clone -b master https://github.com/raspberrypi/pico-sdk.git
$ cd pico-sdk
$ git 子模块更新--init
$ cd ..
$ git clone -b master https://github.com/raspberrypi/pico-examples.git
```

注意

还有其他存储库：pico-extras 和 pico-playground，您可能也对此感兴趣。

2.2. 安装工具链

要在 pico-examples 中构建应用程序，您需要安装一些额外的工具。要构建项目，您将需要用于构建软件的跨平台工具 CMake 以及用于 Arm 的 GNU 嵌入式工具链。您可以 从命令行通过 apt 安装这两种工具。apt 将忽略您已经安装的所有内容。

```
$ sudo apt 更新
$ sudo 易于安装 CMake GCC-臂无- EABI 建立必要®
```

1.需要本地 gcc 和 g ++来编译 pioasm，elf2uf2

第 3 章用 C 闪烁 LED

在编写用于硬件的软件时，通常先打开，关闭然后再打开 LED，这是在新的编程环境中运行的第一个程序。学习如何使 LED 闪烁可使您到达任何地方的一半。我们将继续进行操作，并使连接到 RP2040 引脚 25 的 Raspberry Pi Pico 上的板载 LED 闪烁。

Pico 示例: https：//github.eom/raspberrypi/pico-examples/tree/master/blink/blink.c 第 9 -19 行

```
9   int main（）{
10      const uint LED_PIN = 25；
11      gpio_init（LED_PIN）；
12      gpio_set_dir（LED_PIN，GPIO_OUT）；
13      而（true）{
14      gpio_put（LED_PIN，1）；
15      sleep_ms（250）；
16      gpio_put（LED_PIN，0）；
17      sleep_ms（250）；
18      }
19 }
```

3.1. 建筑物“眨眼”

从我们之前创建的 pico 目录中，使用 cd 进入 pico-examples 并创建一个构建目录。

```
$ cd pico-examples
$ mkdir 构建
$ cd 构建
```

然后，假设您将 pico-sdk 和 pico-examples 存储库并排克隆到同一目录中，则设置 PICO_SDK_PATH：

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

通过运行 cmake ..准备您的 cmake 构建目录。

```
$ cmake ..
从环境（'../../pico-sdk'）使用 PICO_SDK_PATH
PICO SDK 位于/ home / pi / pico / pico-sdk
-构建文件已写入到：/ home / pi / pico / pico-examples / build
```

注意

cmake 将默认为 Release 版本，其中启用了编译器优化并删除了调试信息。要构建一个调试版本，运行 cmake 的-DCMAKE_BUILD_TYPE =调试.. 。我们将在 5.1 节中对此进行探讨。

CMake 现在已经为 pico-examples 树准备了一个构建区域。从这里，可以键入 make 来构建所有示例应用程序。但是，当我们构建眨眼时，我们现在仅通过在输入 make 之前将目录更改为眨眼目录来构建该应用程序。

TIP

调用 化妆 用-j4 将运行四个并行的化妆工作，以加快速度。Raspberry Pi 4 具有 4 个核心，因此-j4 是一个合理的数字。

```
$ cd 闪烁
$ make -j4
ELF2UF2Build boot_stage2_original'ELF2UF2Build'
扫描目标的依赖关系扫描目标的依赖关系[0%]创建目录
[100%]链接 CXX 可执行文件眨眼。精灵[100%]内置目标眨眼
```

在其他目标中，我们现在建立了：

- blink.elf ，供调试器使用
- blink.uf2 ，可以将其拖动到 RP2040 USB Mass Storage Device 上

此二进制文件将使连接到 RP2040 的 GPIO25 的 Raspberry Pi Pico 的板上 LED 闪烁。

3.2. 加载并运行“闪烁”

将软件加载到基于 RP2040 的板上的最简单方法是将其安装为 USB 大容量存储设备。这样做可以将文件拖到板上以对闪存进行编程。继续并使用 micro-USB 电缆将 Raspberry Pi Pico 连接到 Raspberry Pi，并确保按住 BOOTSEL 按钮（图 1）将其强制为 USB Mass Storage Mode。

注意

如果您知道程序将在第一次运行，则通过 USB Mass Storage 方法加载代码非常有用，但是如果您正在开发新的程序，则可能需要对其进行调试。因此，您也可以使用串行线调试接口将软件加载到 RP2040 上，请参阅第 5 章。除了加载软件之外，您还可以：设置断点，检查变量，并检查内存内容。

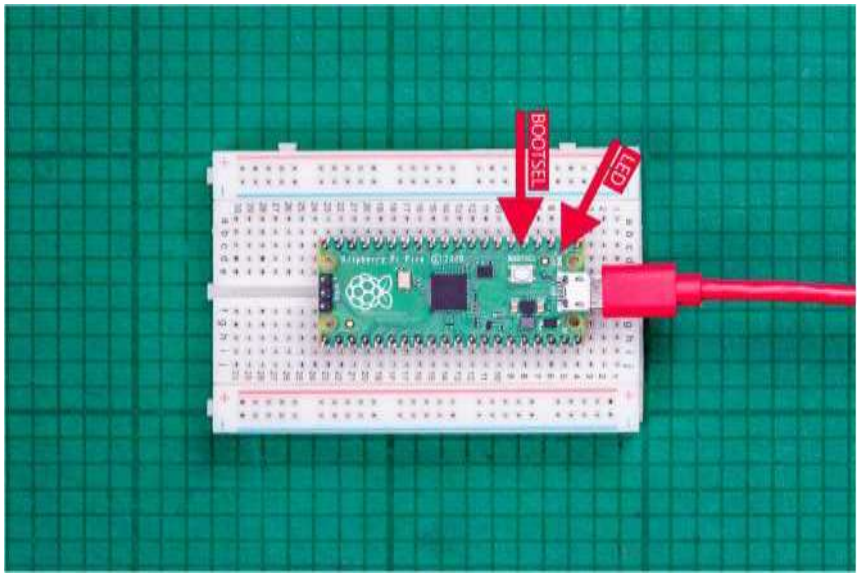
 # 注意

如果未在 Raspberry Pi Pico 上遵循这些说明，则可能没有 BOOTSEL 按钮，请参见图 1。在这种情况下，您应该检查是否有其他方法将闪存 CS 引脚接地，例如跳线，以告诉 RP2040 在引导时进入 BOOTSEL 模式。如果没有这种方法，则可以使用"串行线调试"接口加载代码。

3.2.1. 从桌面

如果您正在运行 Raspberry Pi 桌面，则 Raspberry Pi Pico 应自动安装为 USB 大容量存储设备。从这里，您可以将 blink.uf2 拖放到大容量存储设备上。RP2040 将重新启动，将自身卸载为大容量存储设备，然后开始运行刷新的代码，请参见图 1。

图 1.闪烁 Raspberry Pi Pico 上的板载 LED。箭头指向板载 LED 和 BOOTSEL 按钮。



3.2.2. 使用命令行

例如，如果您通过 ssh 登录，则可能必须手动安装大容量存储设备：

```
$ dmesg | 尾巴
[ 371.973555] SD 0：0：0：0：[SDA]附加的 SCSI 可移动磁盘
$ sudo mkdir -p / mnt / pico
$ sudo mount / dev / sda1 / mnt / pico
```

如果您可以在/ mnt / pico 中 看到文件，则表明 USB Mass Storage Device 已正确安装：

```
$ ls / mnt / pico /
INDEX.HTM INFO_UF2.TXT
```

将您的 blink.uf2 复制到 RP2040：

```
须藤 cp blink.uf2 / mnt / pico 须藤同步
```

RP2040 已经作为 USB 大容量存储设备断开连接并正在运行您的代码，但为了整理整洁，请卸载/ mnt / pico

```
须藤 umount / mnt / pico
```

注意

断开板上电源不会删除代码。重新为开发板供电后，刚刚加载的代码将重新开始运行。如果要从板上删除代码并上载新代码，请在通电时按住 BOOTSEL 开关，以使板进入大容量存储模式。

需要更多细节吗？

本文档说明了如何构建软件并将其加载到 Raspberry Pi Pico。大量的幕后工作将我们的眨眼应用程序变成了二进制程序，而 **Pico C / C ++ SDK** 书则拉开了帷幕，并展示了其中涉及的一些机制。如果您还不担心这种事情，请继续阅读！

第 4 章在 C 语言中说“ Hello World”

在使 LED 闪烁后，大多数开发人员要做的下一步就是创建并使用串行端口，然后说“ Hello World”。

Pico 示例：https：//github.eom/raspberrypi/pico-examples/tree/master/hello_world/serial/hello_serial.c 第 10 -17 行

```
10 int main（）{
11         stdio_init_all（）；
12         而（true）{
13         printf（“你好，世界！\ n” ）
14         sleep_ms（1000 ）；
15         }
16         返回 0；
17 }
```

4.1. Raspberry Pi Pico 上的串行输入和输出

串行输入（stdin ）和输出（stdout ）可以定向到串行 UART 或 USB CDC（USB 串行）。但是默认情况下，stdio 和 printf 将针对默认的 Raspberry Pi Pico UART0。

Raspberry Pi Pico 默认 UART0

GND（引脚 3）
GP0（UART0_TX，引脚 1）
GP1（UART0_RX，引脚 2）

重要

默认的 Raspberry Pi Pico UART TX 引脚（从 Raspberry Pi Pico 输出）是 GP0 引脚，而 UART RX 引脚（进入 Raspberry Pi Pico）是 GP1 引脚。默认 UART 引脚使用板配置文件在每个板上进行配置。树莓裨微微配置中可以找到 https://github.com/raspberrypi/pico-sdk/tree/master/src/ 板/包含/板/pico.h。如果未指定其他板，则 Pico SDK 默认为 Raspberry Pi Pico 的板名称。

Pico SDK 利用 CMake 来控制其构建系统，请参阅第 7 章，利用 pico_stdlib 接口库聚合必需的源文件以提供功能。

Pico 示例：https：//github.com/raspberrypi/pico-examples/tree/master/hello_world/serial/CMakeLists.txt 第 1 -12 行

```
1  add_executable（hello_serial
2              hello_serial.c
3              ）
4
4   #拉入我们的 pico_stdlib，其中汇总了常用功能
5  target_link_libraries（hello_serial pico_stdlib）
7
6   #创建 map / bin / hex / uf2 文件等。
7  pico_add_extra_outputs（hello_serial）
10
11 #通过 pico_set_program_url 添加网址
12 example_auto_set_url（hello_serial）
```

可以使用 CMake 指令更改 stdout 的目标，将输出定向到 UART 或 USB CDC，或同时定向到这两者，

pico_enable_stdio_usb（程序 hello_world 1）® pico_enable_stdio_uart（程序 hello_world 0）@

1. 通过 USB CDC（USB 串行）启用 printf 输出
2. 禁用通过 UART 的 printf 输出

这意味着**无需更改 C 源代码**，就可以将 stdio 的目标从 UART 更改为 USB。

Pico 示例：https：//github.com/raspberrypi/pico-examples/tree/master/hello_world/usb/CMakeLists.txt 第 1-16 行

```
1  add_executable（hello_usb
2              hello_usb.c
3              ）
4
4   #拉入我们的 pico_stdlib，其中汇总了常用功能
5  target_link_libraries（hello_usb pico_stdlib）
7
6   #启用 USB 输出，禁用 uart 输出
7  pico_enable_stdio_usb（hello_usb 1）
8  pico_enable_stdio_uart（hello_usb 0）
11
9   #创建 map / bin / hex / uf2 文件等。
10 pico_add_extra_outputs（hello_usb）
14
11  #通过 pico_set_program_url 添加网址
12 example_auto_set_url（hello_usb）
```

4.2. 建立“ Hello World”

与前面的“眨眼”示例一样，将目录更改为 pico-examples 树内的 hello_world 目录，然后运行 make 。

```
$ cd hello_world
$ make -j4
扫描目标 ELF2UF2Build 的依赖项[   0%]为“ ELF2UF2Build”创建目录
[33%]链接 CXX 可执行文件 hello_usb.elf [33%]已建立目标 hello_usb
[100%]链接 CXX 可执行文件 hello_serial.elf
[100%]建立目标 hello_serial
```

这将在 hello_world / serial /和 hello_world / usb / 目录中构建两个单独的示例程序。

在其他目标中，我们现在建立了：

- serial / hello_serial.elf ，供调试器使用

- serial / hello_serial.uf2 ，可以将其拖动到 RP2040 USB Mass Storage Device（UART 串行二进制文件）上
- usb / hello_usb.elf ，供调试器使用
- usb / hello_usb.uf2 ，可以将其拖到 RP2040 USB Mass Storage Device（USB 串行二进制文件）上

其中 hello_serial 将 stdio 定向到 GP0 和 GP1 引脚上的 UART0，hello_usb 将 stdio 定向到 USB CDC 串行。

4.3. 刷新并运行“ Hello World”

使用 micro-USB 电缆将 Raspberry Pi Pico 连接到 Raspberry Pi，并确保按住 BOOTSEL 按钮将其强制进入 USB Mass Storage Mode。连接后，松开 BOOTSEL 按钮，如果您正在运行 Raspbrerry Pi Desktop，它将自动安装为 USB 大容量存储设备。从这里，您可以将 hello_serial.uf2 或 hello_usb.uf2 拖放 到大容量存储设备上。

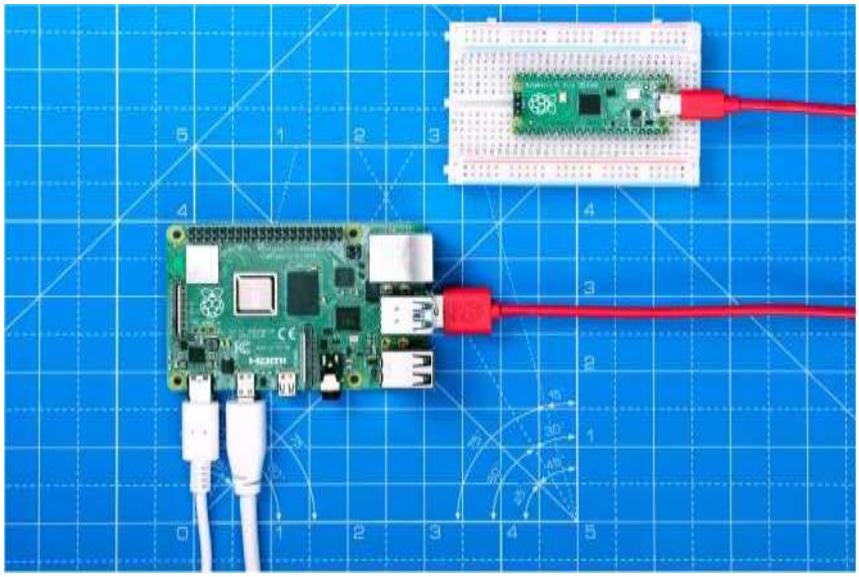
RP2040 将重新启动，将自身卸载为大容量存储设备，并开始运行刷新的代码。

但是，尽管“ Hello World”示例现在正在运行，但我们仍无法看到文本。我们需要将主机连接到 Raspberry Pi Pico 上的标准 UART，以查看输出。

4.4. 请参见“ Hello World” USB 输出

如果拖放了 hello_usb.uf2 二进制文件，则“ Hello World”文本将定向到 USB 串行。

图 2. 通过 USB 将 Raspberry Pi 连接到 Raspberry Pi Pico。



在通过 USB 将 Raspberry Pi Pico 直接连接到 Raspberry Pi 的情况下，请参见图 2，通过安装 minicom 可以看到文本：

```
$ sudo apt 安装 minicom
```

并打开串口：

```
$ minicom -b 115200 -o -D / dev / ttyACM0
```

您应该看到世界，您好！打印到控制台。

要退出小型机，使用 CTRL-A ，然后 X 。

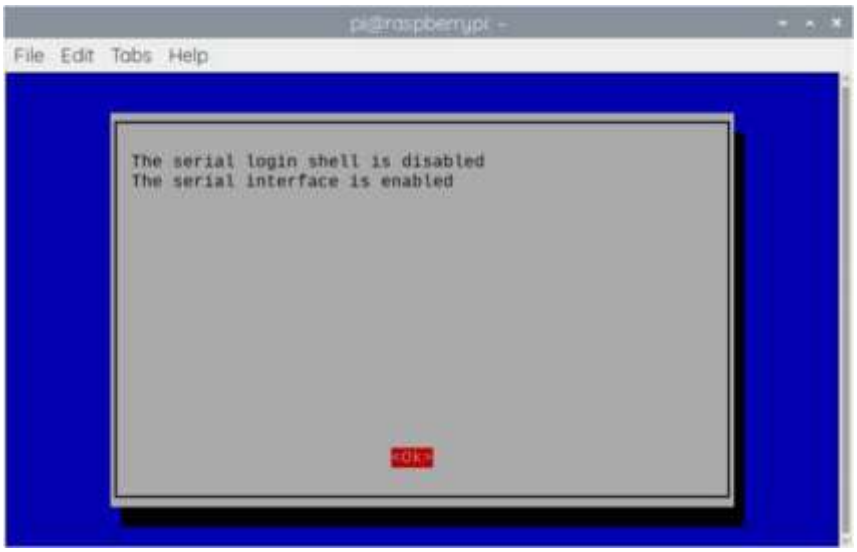
4.5. 请参见“ Hello World” UART 输出

或者，如果拖放了 hello_usb.uf2 二进制文件，则“ Hello World”文本将定向到 GP0 和 GP1 引脚上的 UART0。您需要做的第一件事就是在 Raspberry Pi 主机上启用 UART 串行通信。为此，运行 raspi-config ，

```
$ sudo raspi-config
```

并转到“接口串行选项”，并在询问“您是否希望通过串行访问登录外壳程序”时选择“否”。当系统询问“是否要启用串行端口硬件？”时，选择“是”。您应该看到类似图 3 的内容。

图 3. 在 Raspberry Pi 上使用 raspi-config 启用串行 UART 。



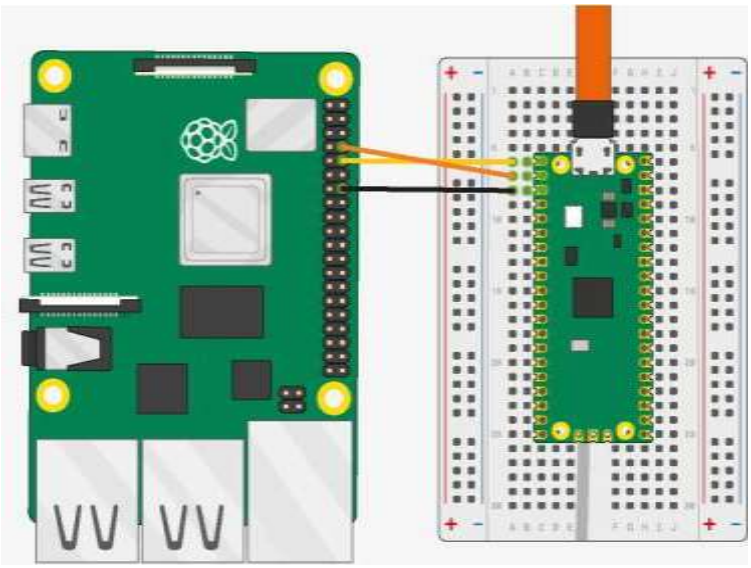
离开 raspi-config 之前，您应该选择“是”并重新启动 Raspberry Pi 以启用串行端口。

然后，应将 Raspberry Pi 和 Raspberry Pi Pico 与以下映射连接在一起：

树莓派	Raspberry Pi Pico
GND（引脚 14）	GND（引脚 3）
GPIO15（UART_RX0，引脚 10）	GP0（UART0_TX，引脚 1）
GPIO14（UART_TX0，引脚 8）	GP1（UART0_RX，引脚 2）

参见图 4。

图 4. Raspberry Pi 4 和 UART0 连接在一起的 Raspberry Pi Pico。



如果尚未将两块板连接在一起，则应安装 minicom：

```
$ sudo apt 安装 minicom
```

并打开串口：

```
$ minicom -b 115200 -o -D / dev / serial0
```

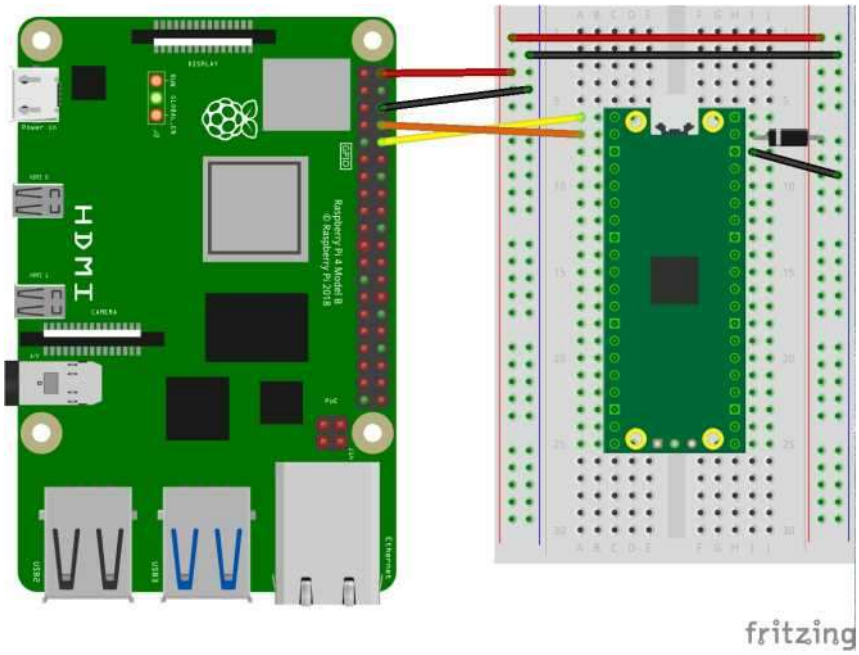
切换 Raspberry Pi Pico 的功能，您应该会看到 Hello，世界！打印到控制台。

要退出小型机，使用 CTRL-A ，然后 X 。

4.6. 为电路板供电

您可以从 USB 拔下 Raspberry Pi Pico 插头，并通过二极管另外将 Raspberry Pi 的 5V 引脚连接到 Raspberry Pi Pico VSYS 引脚，为电路板供电，请参见图 5，在理想情况下，该二极管应该是肖特基二极管。

图5.仅使用GPIO 引脚连接的Raspberry Pi 和Raspberry Pi Pico。



虽然可以将 Raspberry Pi 的 5V 引脚连接到 Raspberry Pi Pico VBUS 引脚，但不建议这样做。将 5V 电源轨短接将意味着无法使用 Micro USB。在 USB 主机模式下使用 Raspberry Pi Pico 时是一个例外，在这种情况下，必须将 5V 连接到 VBUS 引脚。

3.3V 引脚是 Raspberry Pi Pico 上的 OUTPUT 引脚，您不能通过此引脚为 Raspberry Pi Pico 供电，并且不应将其连接到电源。

第 5 章使用 SWD 进行调试

重要

这些说明假定您使用的是 Raspberry Pi Pico，如果您使用的是基于 RP2040 的备用板，则细节可能会有所不同。

Raspberry Pi Pico 提供一个 SWD（单线调试）端口，可用于交互式调试运行在 RP2040 上的二进制文件。但是，要使用它，您首先需要构建二进制文件的特殊调试版本并安装一些其他工具。

5.1. 生成“ Hello World”调试版本

您可以使用 CMAKE_BUILD_TYPE = Debug 来构建“ Hello World””的调试版本，如下所示，

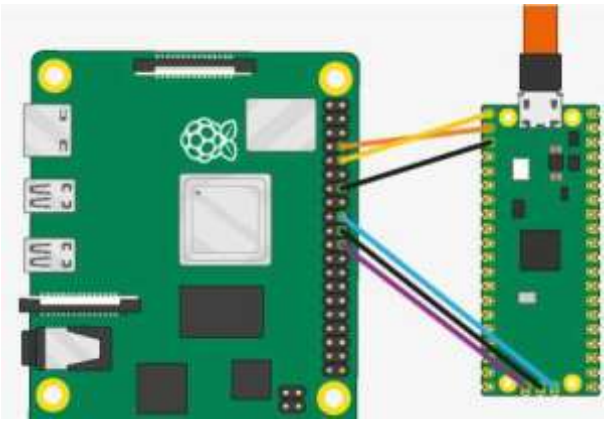
```
$ cd～/ pico / pico-examples /
$ rm -rf 构建
$ mkdir 构建
$ cd 构建
$ export PICO_SDK_PATH = .. / .. / pico-sdk
$ cmake -DCMAKE_BUILD_TYPE =调试..
$ cd hello_world
$ make -j4
```

5.2. 安装 OpenOCD

OpenOCD 是一种调试转换器：它允许主机系统在 RP2040 上加载，运行和调试软件，以及交互地拨动和浏览硬件寄存器。OpenOCD 可以通过许多硬件接口连接到 RP2040 的 SWD 端口，包括从 Raspberry Pi GPIO 进行直接位敲打。

默认配置是在 Pi GPIO 24 上具有 SWDIO，在 GPIO 25 上具有 SWCLK-可以将其连接到 Raspberry Pi Pico，如图 6 所示。

图 6.具有 UART 和 SWD 端口连接在一起的 Raspberry Pi 4 和 Raspberry Pi Pico。两者都直接跳回到 Raspberry Pi 4，而无需使用面包板。



vv vu

如果可能的话，应该将 SWD 端口直接连接到 Raspberry Pi，因为信号的完整性很重要。通过面包板或其他间接方法对 SWD 端口进行布线可能会充分降低信号完整性，从而导致通过该连接加载代码不稳定或完全失败。同样重要的是，也要在两者之间直接连接地线（0v），而不要依赖其他接地路径。

注意，Raspberry Pi Pico 还必须通电（例如通过 USB）才能对其进行调试！您必须建立我们的 OpenOCD 分支才能获得多点 SWD 支持：

O 注意

这些说明假定您要在/ home / pi / pico / openocd 中构建 openocd

```
$ cd~/ pico
$ sudo apt install automake autoconf build-essential texinfo libtool libftdi-dev libusb-1.0-0- dev
$ git clone https://github.com/raspberrypi/openocd.git- 递归--branch rp2040 --depth = 1
$ cd openocd
$ ./引导
$ ./configure --enable-ftdi --enable-sysfsgpio --enable-bcm2835gpio
$ make -j4
$ sudo make install
```

5.3. 安装 GDB

安装 gdb-multiarch，

```
$ sudo apt 安装 gdb-multiarch
```

5.4. 使用 GDB 和 OpenOCD 调试 Hello World

为确保您的 Raspberry Pi 4 和 Raspberry Pi Pico 正确连接在一起，我们可以通过 raspberrypi-swd 接口将 OpenOCD 连接到芯片。

```
$ openocd -f 接口/raspberrypi-swd.cfg -f 目标/rp2040.cfg
```

您的输出应如下所示：

```
信息：rp2040.core0：硬件具有 4 个断点，2 个观察点
信息：rp2040.core1：硬件有 4 个断点，2 个观察点
信息：在 3333 上为 rp2040.core0 启动 gdb 服务器
信息：在端口 3333 上监听 gdb 连接
```

警告

如果您看到类似 Info：DAP init 的错误，则说明您的 Raspberry Pi Pico 电源关闭，接线错误或信号完整性问题。尝试使用其他 GPIO 跳线电缆。

该 OpenOCD 终端需要保持打开状态。因此，继续打开另一个终端，在这个终端中，我们将 gdb 实例附加到 OpenOCD。导航到“ Hello World”示例代码，然后从命令行启动 gdb 。

```
$ cd~/ pico / pico-examples / build / hello_world $ gdb-multiarch hello_world.elf
```

将 GDB 连接到 OpenOCD，

```
（gdb）目标远程 localhost：3333
```

小费

您可以创建一个.gdbinit 文件，这样就不必每次都键入 target remote localhost：3333 。通过 echo“ target remote localhost：3333”>~/ .gdbinit 来执行此操作。但是，这会干扰 VSCode 中的调试（第 6 章）。

并将 hello_world.elf 加载到 Flash 中，

```
（gdb）负载
正在加载节.boot2，大小为 0x100 lma 0x10000000
正在加载节.text，大小为 0x22d0 lma 0x10000100
正在加载节.rodata，大小为 0x4a0 lma 0x100023d0
正在加载节.ARM.exidx，大小为 0x8 lma 0x10002870
正在加载节.data，大小为 0xb94 lma 0x10002878
起始地址 0x10000104，负载大小 13324
传输速率：31 KB /秒，2664 字节/写。
```

然后开始运行。

```
(gdb) 监视器重置初始化
(gdb) 继续
```

重要

如果您在尝试通过 OpenOCD 将二进制文件加载到 Raspberry Pi Pico 时看到类似于在 Flash 中错误完成 Flash 操作错误或使用 vFlashErase 数据包擦除 Flash 错误，则 Raspberry Pi 和 Raspberry Pi Pico 之间的信号完整性可能很差。如果您没有直接在两块板之间连接 SWD 连接，请参见图 6，您应该尝试这样做。另外，您可以尝试降低的值 adapter_khz 中树莓派-swd.cfg 配置文件，试图减少一半，直到你看到电路板之间的连接成功。由于我们在电路板之间进行时序调整是微不足道的，因此不良的信号完整性可能会导致错误。

或者，如果您想在运行可执行文件之前在 main（）处设置断点，

```
(gdb) 监视器重置初始化
(gdb) b 主
(gdb) 继续
线程 1 在/home/pi/pico/pico-examples/hello_world/hello_world.c:11 11      setup_standard_uart（）;处打断点 1，main （）；
```

在达到断点之前继续，

```
(gdb) 继续
```

要退出 gdb 类型，

```
(gdb) 退出
```

第 6 章使用 Visual Studio 代码

Visual Studio Code（VSCode）是 Microsoft 开发的一种流行的开源编辑器。如果要图形界面编辑和调试代码，建议在 Raspberry Pi 4 上使用集成开发环境（IDE）。

6.1. 安装 Visual Studio 代码

重要

这些安装说明依赖于您已经下载并安装了命令行工具链（请参阅第 3 章），以及下载并构建 OpenOCD 和 GDB 并对其进行配置以进行命令行调试，请参阅第 5 章。

可以从 https://code.visualstudio.com/ 下载下载用于 Raspberry Pi 的 ARM 版本的 Visual Studio Code 。如果您使用的是 32 位操作系统（例如默认的 Raspberry Pi OS），请下载 ARM .deb 文件；如果使用的是 64 位操作系统，请下载 ARM 64 .deb 文件。下载后，双击.deb 软件包，然后按照说明进行安装。

安装下载的.deb 。从命令行打包。cd 到下载文件的文件夹，然后使用 dpkg -i <下载的文件名.deb>进行安装。

安装完成后，安装调试 Raspberry Pi Pico 所需的扩展：

```
代码--install-extension marus25.cortex-debug 代码--install-extension ms-vscode.cmake-tools 代码--install-extension ms-vscode.cpptools
```

最后，从终端窗口中启动 Visual Studio 代码：

```
$ export PICO_SDK_PATH = / home / pi / pico / pico-sdk
$代码
```

确保设置了 PICO_SDK_PATH， 以便 Visual Studio Code 可以找到 Pico SDK。

注意

如果 PICO_SDK_PATH 默认情况下不会在你的 shell 环境中设置你将不得不在每次启动之前 vscode 打开新的终端窗口时间设置，或从菜单开始 vscode。因此，您可能需要将其添加到.profile 或.bashrc 文件中。

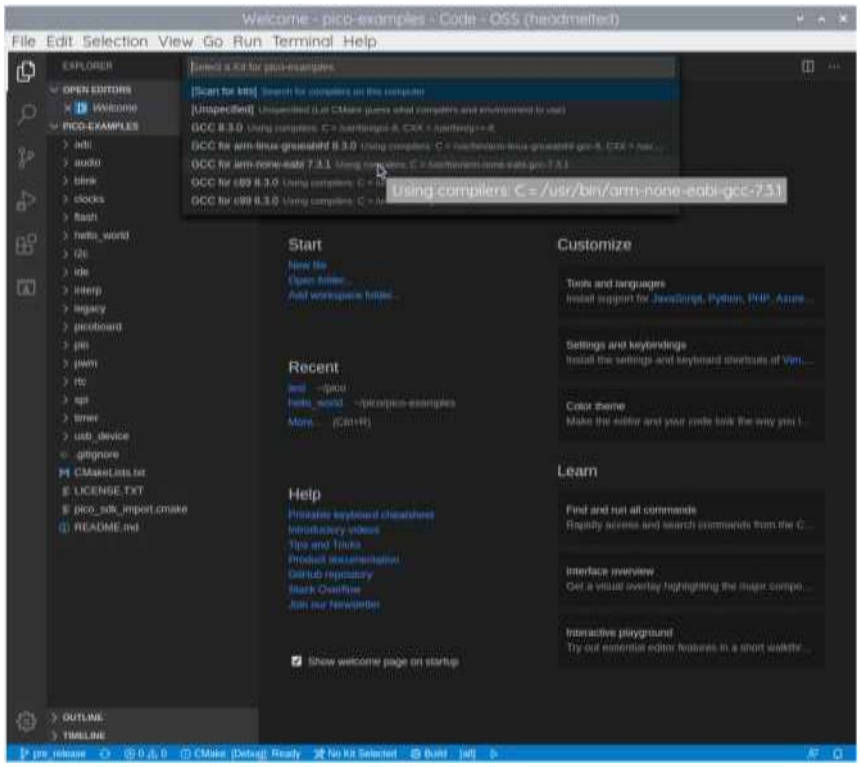
6.2. 载入项目

来吧，打开微微例子通过转到 Explorer 工具栏（文件夹按 Ctrl + Shift + E ），选择"打开文件夹"，并 nagivating 到，/家庭/ PI /微微/微微的例子 在文件弹出。然后单击"确定"将文件夹加载到 VSCode 中。

只要安装了 CMake Tools 扩展，大约一秒钟后，您就会在 vscode 窗口的右下角看到一个弹出窗口。

点击"是"以配置项目。然后将提示您选择编译器，请参见图 7，

图 1.提示为项目选择正确的编译器。



并应从下拉菜单中为"无臂 eabi"选择 GCC 。

小费

如果错过了弹出窗口（将在几秒钟后再次关闭），则可以通过单击 VSCode 窗口底部蓝色栏中的"未选择任何工具包"来配置编译器。

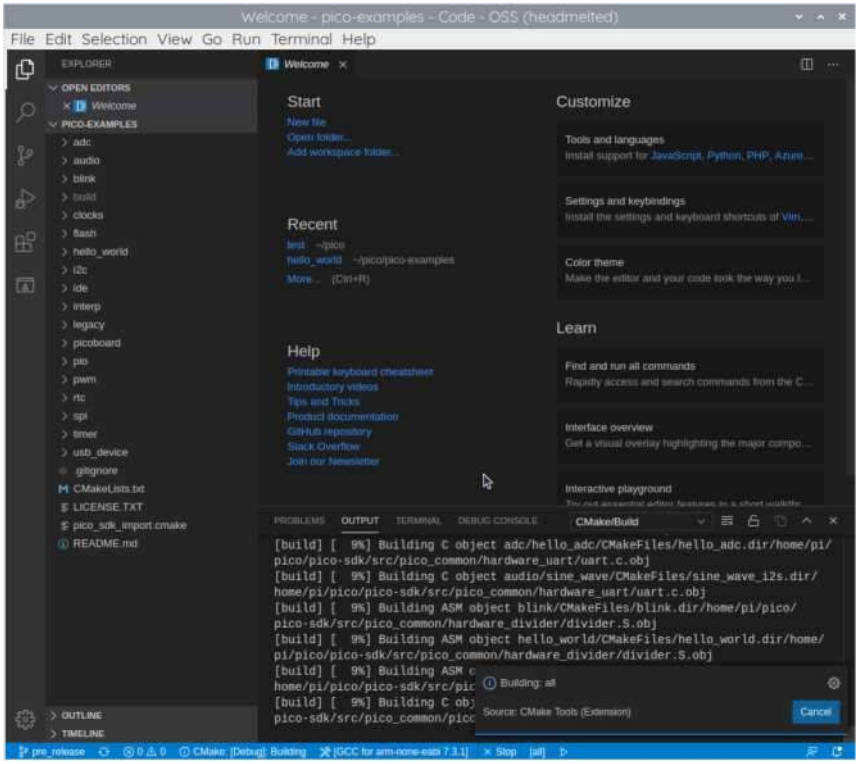
然后，您可以单击蓝色底部栏中的"构建"按钮以在 pico-examples 文件夹中构建所有示例，也可以单击蓝色底部栏中标有" [全部]"的位置。这将显示一个下拉列表，您可以在其中选择一个项目。现在，输入" hello_world"并选择" Hello World"可执行文件。这意味着 VSCode 将仅构建" Hello World"示例，以节省编译时间。

" TIP

您可以通过在蓝色底部栏中单击" CMake：[Debug]：Ready"的位置来构建" Debug"和" Release"可执行文件。默认设置是构建一个启用"调试"的可执行文件，以准备 SWD 调试。

继续并单击窗口底部蓝色栏中的"构建"按钮（带有齿轮）。这将创建构建目录并像在第 3.1 节中手动进行的那样运行 CMake ，然后开始构建本身，请参见图 8。

图 8.在 Visual Studio Code 中构建 pico-examples 项目



就像我们之前在命令行中所做的那样，在其他目标中，我们现在已经构建：

- hello_world.elf ，供调试器使用
- hello_world.uf2 ，可以将其拖动到 RP2040 USB 大容量存储设备上

6.3. 调试项目

所述微微例子 回购包含一个示例的调试配置，将启动 OpenOCD 的，附上 GDB，最后启动 CMake 的被配置为生成应用程序。继续，然后将此文件（launch-raspberrypi-swd.json ）复制到 picoexamples /中。vscode 目录为 launch.json 。我们还提供了 settings.json 文件，建议您也将其复制。此设置。json 从 CMake 插件中删除了一些可能造成混淆的选项（包括试图在主机上运行 Pico 二进制文件的 Debug 和 Run 按钮损坏）。

```
$ cd ~/ pico / pico-examples
$ mkdir .vscode
$ cp ide / vscode / launch-raspberrypi-swd.json .vscode / launch.json
$ cp ide / vscode / launch-raspberrypi-swd.json .vscode / settings.json
```

Pico 示例：https : //github.com/raspberrypi/pico-examples/tree/master/ide/vscode/launch-raspberrypi-swd.json 第 1 行 27

```
1 {
2 " version": " 0.2.0" ,
3 "配置": [
4         {
5         " name": " Pico Debug" ,
6         " cwd": " $ {workspaceRoot}" ,
7         " executable": " $ {command: cmake.launchTargetPath}" ,
8         " request": "启动" ,
9         " type": " cortex-debug" ,
10        " servertype": " openocd" ,
```

```
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26          ]
27}
        //这可能需要是 arm-none-eabi-gdb，具体取决于您的系统" gdbpath": " gdb-multiarch" ,
        "设备": " RP2040"
        " configFiles": [
        " interface / raspberrypi-swd.cfg" ,
        " target / rp2040.cfg"
        ],
        " svdFile": " /home/pi/pico/pico-sdk/src/rp2040/hardware_regs/rp2040.svd" , " runToMain": true ,
        //解决方法是在重新启动" postRestartCommands"时停在主要 位置: [
        "打破主要" ,
        "继续"
        ]
}
```

注意

如果您的 gdb 称为 arm-none-eabi-gdb 而不是 gdb-multiarch，则可能需要修改 launch.json 中的 gdbpath

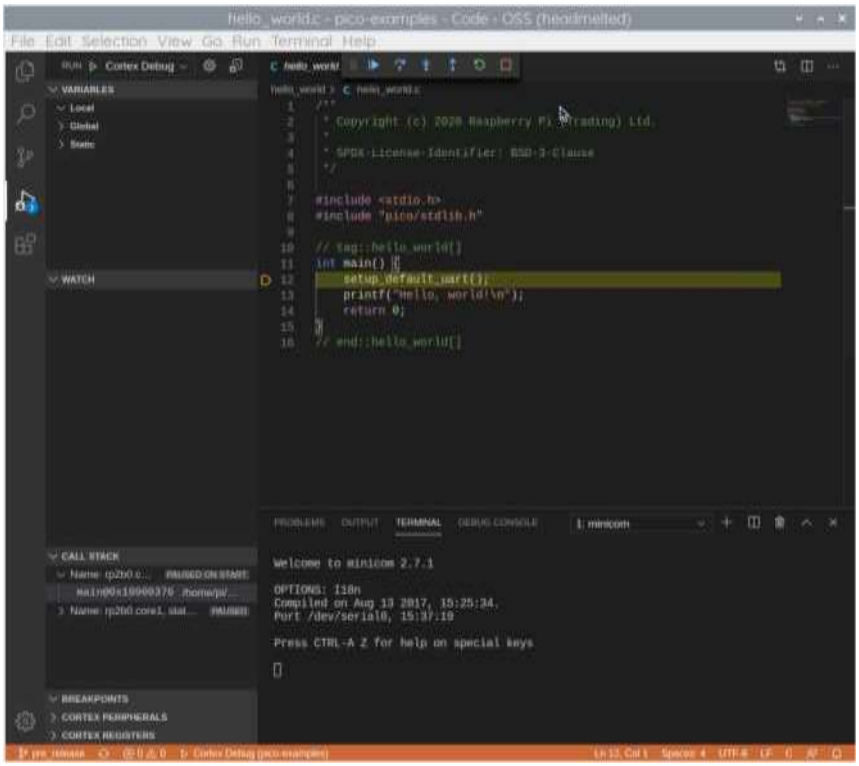
6.3.1. 在 Raspberry Pi Pico 上运行" Hello World"

重要

确保示例" Hello World"代码已作为调试二进制文件（CMAKE_BUILD_TYPE = Debug ）。

现在转到 Debug 工具栏（Ctrl + Shift + D ），然后单击左侧窗格顶部的绿色小箭头（播放按钮），将代码加载到 Raspberry Pi Pico 上并开始调试。

图 9. 在 Visual Studio Code 中调试" Hello World"二进制文件



现在应该将代码加载到 Raspberry Pi Pico 上，并且您应该在窗口的主右侧（上部）窗格中看到" Hello World"的源代码。该代码将开始运行，并将继续到第一个断点一由 launch.json 文件中的 runToMain 指令启用。单击此主要源代码窗口顶部的蓝色小箭头（播放按钮）以继续（F5 ）并开始运行代码。

“ TIP

如果您在底部右侧窗格中切换到"终端"选项卡，下面的 hello_world.c 代码，你可以使用这个打开的 Minicom 里面 VSCode 通过打字来看看从"Hello World"的例子中，UART 输出，

```
$ minicom -b 115200 -o -D / dev / serial0
```

在终端提示符处像我们之前所做的那样，请参阅第 4.5 节。

第 7 章创建自己的项目

继续创建一个目录，将您的测试项目放在 pico-sdk 目录旁边，

```
$ ls -la 总计 16
drwxr-xr-x    7 a员工 224 6四月 10:41 ./
drwx----------@ 27 a员工 864 6四月 10:41 ../
drwxr-xr-x   10 a员工 320 6四月 09:29 pico-examples /
drwxr-xr-x   13 a员工 416 6四月 09:22 pico-sdk /
```

```
$ mkdir 测试$ cd 测试
```

然后在目录中创建一个 test.c 文件，

```
1  #include <stdio.h>
2  #include“ pico / stdlib.h”
3  #include“ hardware / gpio.h”
4  #include“ pico / binary_info.h”
5
6 个常量 LED_PIN = 25 ;
```

© 这些线路将字符串添加到二进制可见使用 picotool ，见附录 B 。

```
8  int main（）{
9
9        bi_decl（bi_program_description（“这是一个测试二进制文件。”））；®
10       bi_decl（bi_1pin_with_name（LED_PIN，“板载 LED”））；
12
13       stdio_init_all（）；
14
15
16
17
18
19
20
21
22
23
24  }
gpio_init（LED_PIN）；gpio_set_dir（LED_PIN，GPIO_OUT）；而（1）{

gpio_put（LED_PIN，0）；sleep_ms（250）；

gpio_put（LED_PIN，1）；puts（“ Hello World \ n”）；sleep_ms（1000）；

}
cmake_minimum_required（版本 3.12）



include（pico_sdk_import.cmake）项目（test_project）pico_sdk_init（）add_executable（测试



      测试
)

pico_enable_stdio_usb（测试 1）®
```

```
pico_enable_stdio_uart（测试 1）@
pico_add_extra_outputs（测试）
target_link_libraries（测试 pico_stdlib）
```

1.这将通过 USB 启用串行输出。

2.这将启用通过 UART 的串行输出。

然后将 pico_sdk_import.cmake 文件从 pico-sdk 安装中的外部文件夹复制到测试项目文件夹。

```
$ cp ../pico-sdk/external/pico_sdk_import.cmake。
```

您现在应该拥有类似这样的内容，

```
$ ls -la 总计 24
drwxr-xr-x 5 a 员工 160 6 四月 10:46 ./
drwxr-xr-x 7 a 员工 224 6 四月 10:41 ../
-rw-r--r-@ 1 个 a 员工 394 6 四月 10:37 CMakeLists.txt
-rw-r--r-- 1 个 a 员工 2744 6 四月 10:40 pico_sdk_import.cmake
-rw-r--r-- 1 个 a 员工 383 6 四月 10:37 测试
```

并可以像以前一样通过“ Hello World”示例进行构建。

```
$ mkdir 构建
$ cd 构建
$ export PICO_SDK_PATH = .. / .. / pico-sdk
$ cmake ..
$使
```

生成过程将产生许多不同的文件。下表显示了重要的内容。

文件扩展名	描述
.bin	原始二进制转储的程序代码和数据
.elf	完整的程序输出，可能包括调试信息
.uf2	UF2 格式的程序代码和数据，可以在将其作为 USB 驱动器安装时拖放到 RP2040 板上
.dis	反汇编已编译的二进制文件
.hex	编译二进制文件的十六进制转储
。地图	.elf 文件随附的映射文件，描述链接器在内存中的排列位置

注意

UF2（USB 闪存格式）是 Microsoft 开发的一种文件格式，用于通过 USB 刷新 RP2040 板。可以在 Microsoft UF2 Specification Repo 上找到更多详细信息。

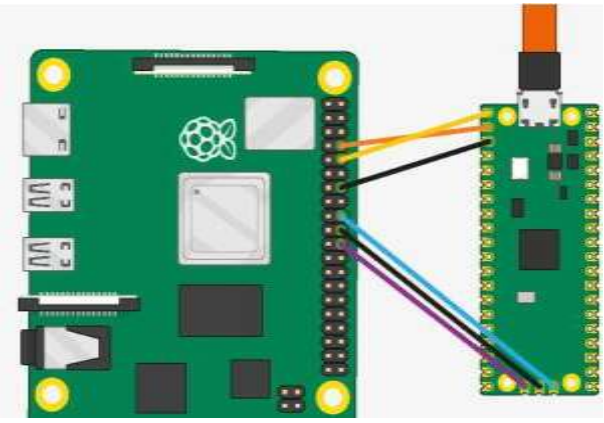
注意

要构建可在 SRAM 中运行的二进制文件而不是 Flash 存储器，您可以使用-DPICO_NO_FLASH = 1 设置 cmake 构建，也可以在 CMakeLists.txt 文件中按每个二进制文件添加 pico_set_binary_type（TARGET_NAME no_flash）来控制它。您可以通过 UF2 将 RAM 二进制文件下载到 RP2040。例如，如果板上没有闪存芯片，则可以使用 UF2 下载在片内 RAM 上运行的二进制文件，因为它仅指定数据所要到达的地址。请注意，您只能下载到 RAM 或 FLASH 中，而不能两者都下载。

7.1. 调试项目

从命令行调试您自己的项目将遵循与 5.4 节中" Hello World"示例所用的相同过程。如图 10 所示连接 Raspberry Pi 和 Raspberry Pi Pico 。

图 10.具有 UART 和 SWD 调试端口的 Raspberry Pi 4 和 Raspberry Pi Pico。两者都直接跳回到 Raspberry Pi 4，而无需使用面包板。



紫外线

然后继续使用 CMAKE_BUILD_TYPE = Debug 如下构建项目的调试版本，

```
$ cd〜/ pico / test
$ rmdir 构建
$ mkdir 构建
$ cd 构建
$ export PICO_SDK_PATH = .. / .. / pico-sdk
$ cmake -DCMAKE_BUILD_TYPE =调试..
$使
```

然后打开一个终端窗口，并使用 raspberrypi-swd 界面附加 OpenOCD。

```
$ openocd -f interface / raspberrypi-swd.cfg -f target / rp2040.cfg
```

此 OpenOCD 终端需要保持打开状态。因此，继续打开另一个终端窗口，并使用以下命令启动 gdb-multiarch

```
$ cd ~/ pico / test / build
$ gdb-multiarch test.elf
```

将 GDB 连接到 OpenOCD，并将 test.elf 二进制文件加载到闪存中，

```
(gdb) 目标远程 localhost：3333
(gdb) 负载
```

然后开始运行

```
(gdb) 监视器重置初始化
(gdb) 继续
```

7.2. 在 Visual Studio Code 中工作

如果要使用 Visual Studio Code 而不是从命令行工作，则可以执行此操作，请参阅第 6 章，以获取有关如何配置环境以及如何将新项目加载到开发环境中以编写和构建代码的说明。

如果您还想使用 Visual Studio Code 调试代码并将其加载到 Raspberry Pi Pico 上，则需要为项目创建一个 launch.json 文件。这个例子推出，树莓派-swd.json 在第 6 章应该工作。您需要将其作为.vscode / launch.json 复制到项目目录中。

7.3. 自动化项目创建

已经创建了一些自动化功能，它将自动创建一个"存根"项目，其中包含所有必要的文件以进行构建。如果您想利用此功能，则需要继续进行操作，并从其 Git 存储库中克隆项目创建脚本，

```
$ git clone https://github.com/raspberrypi/pico-project-generator.git
```

然后可以在图形模式下运行它，

```
$ cd pico-project-generator
$ ./pico_project.py --gui
```

它将打开一个 GUI 界面，允许您配置项目，请参见图 11。

图 11. 使用图形项目创建工具创建 RP2040 项目。

您可以通过从 GUI 上的复选框中选择特定功能来将其添加到项目中。这将确保构建系统将适当的代码添加到构建中，并且还将简单的示例代码添加到项目中以显示如何使用该功能。

有许多选项可用，它们提供以下功能。

控制台选项	描述
通过 UART 进行控制台	通过 UART 启用串行控制台。这是默认值。
通过 USB 控制台	通过 USB 启用控制台。该设备将充当 USB 串行端口。可以作为 UART 选项的补充或替代，但请注意，启用后，其他 USB 功能将无法使用。

代码选项	描述
添加 Pico 库的示例	将为默认情况下在构建中的某些标准库功能生成示例代码，例如 UART 支持和硬件分频器。
从 RAM 运行	通常，构建会创建一个二进制文件，该二进制文件将安装到闪存中。这迫使二进制文件直接从 RAM 运行
高级	调出表格，允许选择特定的电路板构建选项。这些选项会更改功能的工作方式，应谨慎使用。

构建选项	描述
运行构建	一旦创建了项目，就对其进行构建。这将产生可下载到 Raspberry Pi Pico 的文件。
覆盖项目	如果指定文件夹中已经存在一个项目，请用新项目覆盖它。这将覆盖您所做的任何更改。
创建 VSCode 项目	除了 CMake 文件之外，还创建适当的 Visual Studio Code 项目文件。

7.3.1. 从命令行生成项目

该脚本还提供了从命令行创建项目的功能，例如

```
$ export PICO_SDK_PATH =“ / home / pi / pico / pico-sdk”
$ ./pico_project.py --feature spi --feature i2c --project vscode 测试
```

该--feature 选项添加相应的库代码来构建，也示例代码，以显示功能的基本用法。您可以添加多个功能，最高可达 RP2040 的内存限制。您可以使用脚本的--list 选项列出所有可用功能。上面的示例增加了对 I2C 和 SPI 接口的支持。

在这里传递--project 选项将意味着.vscode / launch.json ， .vscode / c_cpp_properties。除了 CMake 项目文件之外，还将创建 json 和.vscode / settings.json 文件。

创建完成后，您可以从命令行以常规方式构建项目，

```
$ cd 测试/构建
$ cmake ..
$使
```

或从 Visual Studio 代码中获取。

您可以使用--help 选项提供命令行参数列表，使用图形模式时也将应用这些参数。

需要更多细节吗?

这里应该有足够的内容来向您*展示如何*入门，但是您可能会发现自己想知道*为什么需要*其中一些文件和咒语。该**微微 C / C ++ SDK** 本书将深入到你的项目是如何实际构建，以及如何在我们行的 CMakeLists.txt 文件，这里涉及到 SDK 的结构，如果你发现自己想知道更多的在未来的某个时刻。

第 8 章在其他平台上构建

Raspberry Pi 是支持 RP2040 开发的主要平台，而其他平台（如 Apple macOS 和 Microsoft Windows）也可以得到支持。

8.1. 在 Apple macOS 上构建

使用 macOS 为 RP2040 构建代码与 Linux 非常相似。

8.1.1. 安装工具链

安装取决于 Homebrew，如果您尚未安装 Homebrew ，则应继续进行安装，

```
$ / bin / bash -c“ $（curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh）”
```

然后安装工具链，

```
$ brew install cmake
$酿造水龙头 ArmMbed /自制配方
$ brew install arm-none-eabi-gcc
```

但是之后，您可以按照 Raspberry Pi 的说明为 RP2040 构建代码。一旦安装了工具链，macOS 和 Linux 之间就不会有任何区别，因此请参阅第 2.1 节，并从那里按照说明获取 Pico SDK 并构建“ Blink”示例。

8.1.2. 使用 Visual Studio 代码

Visual Studio Code（VSCode）是跨平台环境，可在 macOS，Linux 和 Microsoft Windows 上运行。继续下载 macOS 版本，解压缩，然后将其拖到“应用程序文件夹”中。

导航到“应用程序”，然后单击图标以启动 Visual Studio 代码。

8.1.3. 使用 CMake 工具进行构建

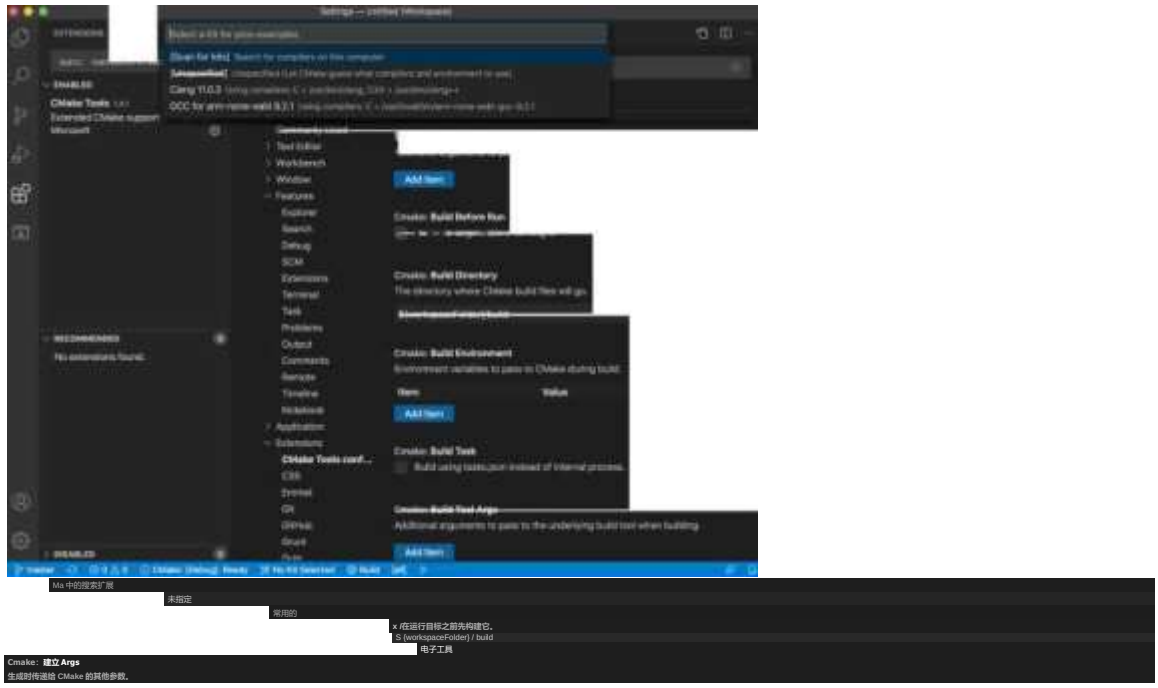
启动 Visual Studio Code 之后，您需要安装 CMake Tools 扩展。单击左侧工具栏中的扩展图标（或键入 Cmd + Shift + X ），然后搜索“ CMake Tools”，然后单击列表中的条目，然后单击安装按钮。

现在，我们需要设置 PICO_SDK_PATH 环境变量。导航到 pico-examples 目录，并创建一个.vscode 目录，并添加一个名为 settings.json 的文件，以告知 CMake Tools Pico SDK 的位置。

```
{
 “ cmake.environment”:        {
        “ PICO_SDK_PATH”:  “ ../ ../pico-sdk”
```

现在转到"文件"菜单，然后单击"将文件夹添加到工作区...”，然后导航到 pico-examples 回购并单击"确定”。项目将加载，并且（可能）会提示您选择编译器，请参见图 12。为您的编译器选择" GCC for arm-none-eabi”。

图 12.提示为项目选择正确的编译器。



继续并单击窗口底部蓝色栏中的"构建”按钮（带有齿轮）。这将创建构建目录并像在第 3.1 节中手动进行的那样运行 CMake ，然后开始构建本身，请参见图 8。

这将产生 ELF，bin 和 uf2 目标，您可以在新创建的 build 目录内的 hello_world 目录中找到它们。UF2 二进制文件可以使用 USB 直接拖放到与计算机相连的 RP2040 板上。

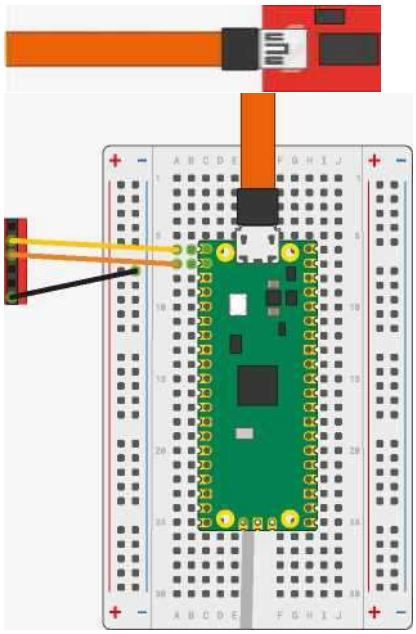
8.1.4. 说“ Hello World”

就像我们在第 4 章中所做的一样，您可以使用 stdio 构建 Hello World 示例，该 stdio 可以路由到 USB CDC（串行）或 GP0 和 GP1 引脚上的 UART0。如果要使用 USB CDC 作为目标输出作为其符合类的设备进行构建，则无需安装驱动程序。您只需要使用终端程序（例如串行或类似程序）来连接到 USB 串行端口。

8.1.4.1. UART 输出

另外，如果你想你想连接到树莓派微微标准的 UART 看到输出，你需要使用一个 USB 转 UART 串行转换您的树莓派的 Pico 连接到 Mac，例如 SparkFun FTDI 基本 板，请参阅图 13。

图 13.连接到Raspberry Pi Pico 的Sparkfun FTDI Basic 适配器



只要您使用的是 Catalina 等最新版本的 macOS，驱动程序就应该已经加载。否则，请参见制造商的 FTDI 芯片驱动器网站。

然后，您应该使用终端程序，例如串行 或类似方式连接到串行端口。串行还包括驱动程序 支持。

8.2. 在 MS Windows 上构建

在 Microsoft Windows 上安装工具链与其他平台有所不同。但是，一旦安装，RP2040 的建筑代码就有些相似。

8.2.1. 安装工具链

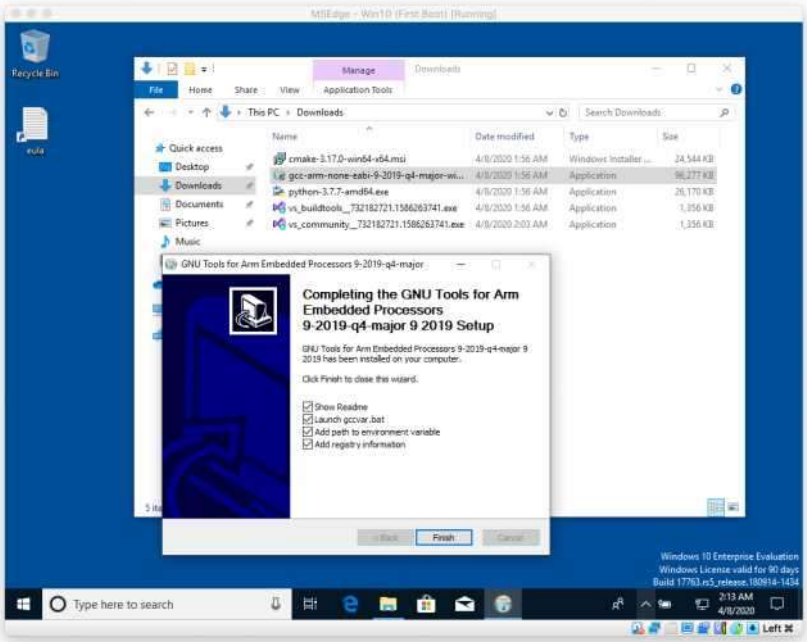
要进行构建，您将需要安装一些额外的工具。

- ARM GCC 编译器
- CMake
- Visual Studio 2019 的生成工具
- Python 3
- Git

下载每个安装程序的可执行安装程序，然后继续并将所有五个软件包安装到 Windows 机器上。

8.2.1.1. 安装 ARM GCC 编译器

图 14.将所需工具安装到 Windows 计算机上。确保将编译器的路径注册为环境变量，以便可以从命令行访问它。



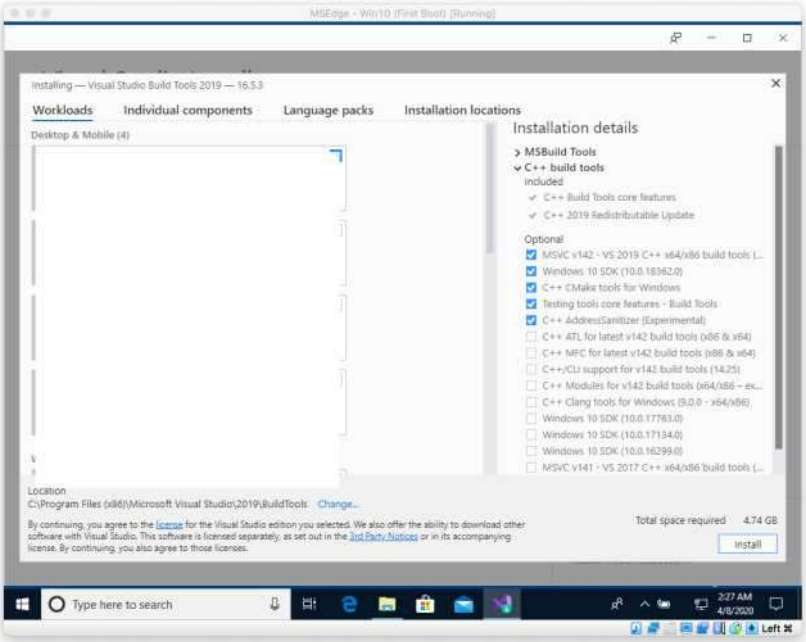
在安装过程中，应在出现提示时选中该框，以将 Windows 外壳程序中的 ARM 编译器路径注册为环境变量。

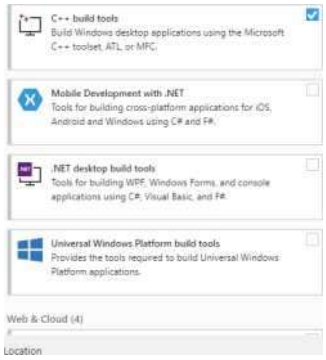
8.2.I.2. 安装 CMake

在安装过程中，当安装程序提示时，将 CMake 添加到所有用户的系统 PATH 中。

8.2.1.3. 安装 Visual Studio 代码

图 15.安装 Visual Studio 2019 的构建工具。



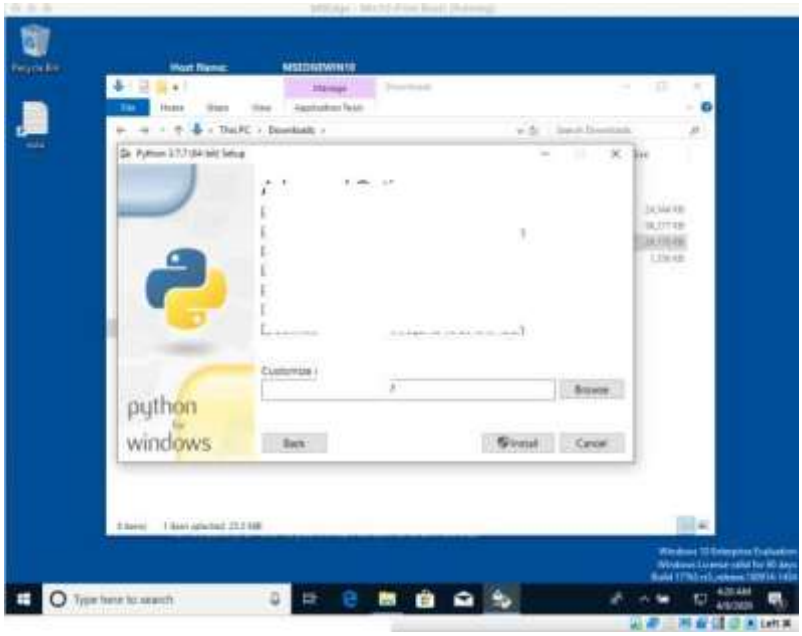


当" Visual Studio 生成工具"安装程序提示时，您仅需要安装 C ++生成工具。

8.2.I.4. 安装 Python 3

在安装过程中，根据安装程序的提示，将所有用户的 Python 3.7 添加到系统 PATH 中。在 Python 安装结束时提示时，您还应该禁用 MAX_PATH 长度。

图16.安装 Python 3.7 选中"高级选项"下的"为所有用户安装"框。



高级选项

0 为所有用户安装
0 将文件与 Python 关联（需要 py 启动器）
0 为已安装的应用程序创建快捷方式
0 将 Python 添加到环境变量
0 预编译标准库
□ 下载调试符号
□ 下载调试二进制文件（需要 VS 2015 或更高版本）
安装位置
C：\ Program Files \ Python37

另外，在安装 Python 时选择"自定义安装"，单击"可选功能"，然后在"高级功能"下选择"为所有用户安装"。

注意

您可能必须进行符号链接，以便 Makefile 可以找到 Python3。为此，请在 Windows 菜单旁边的"运行"窗口中键入 cmd 以打开"开发人员命令提示符"窗口，但在右侧窗格中选择"以管理员身份运行"。使用管理权限打开窗口。然后导航到 C：\ Program Files \ Python37 并进行符号链接。

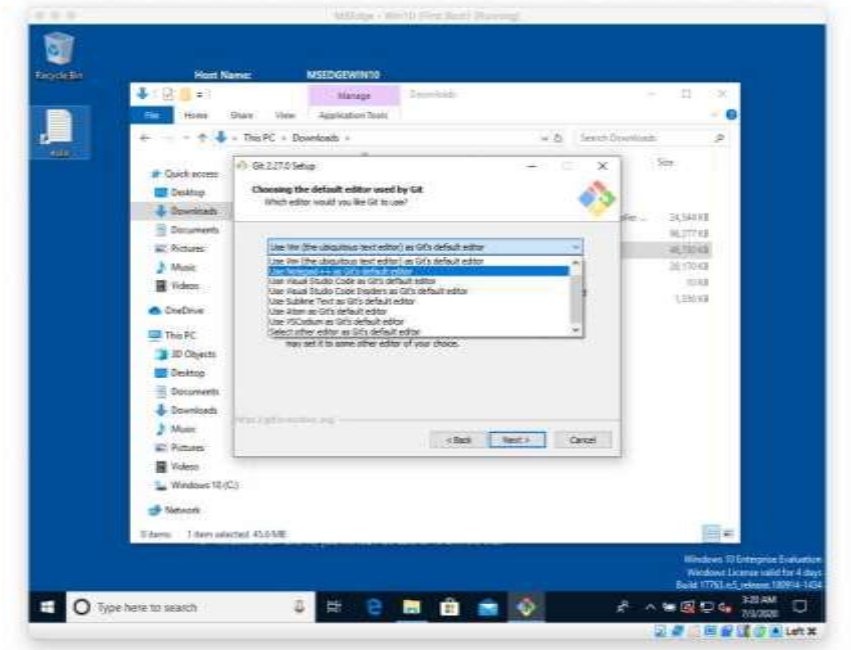
```
C：\ Program Files \ Python37> mklink python3.exe python.exe
```

这 **将**不再是必需的。但是，如果由于 make 找不到 Python 安装而导致构建失败，则应将符号链接添加到可执行文件中。那可能会解决问题。

8.2.1.5. 安装 Git

在安装 Git 时，应确保将默认编辑器更改为远离 vim ，请参见图 17。

图17.安装
吉特



确保选中该复选框以允许使用第三方工具中的 Git，并且除非有很强的理由，否则在安装 Git 时，还应选中"按原样签出，按原样提交"框，选择"使用 Windows" '默认控制台窗口"和"在安装过程中启用对伪控制台的实验性支持"。

8.2.2. 获取 Pico SDK 和示例

```
C：\ Users \ pico \ Downloads> git clone -b master https://github.com/raspberrypi/pico-sdk.git C：\ Users \ pico \ Downloads> cd pico-sdk
C：\ Users \ pico \ Downloads \ pico-sdk> git 子模块更新--init C：\ Users \ pico \ Downloads \ pico-sdk> cd ..
C：\ Users \ pico \ Downloads> git clone -b master https://github.com/raspberrypi/pico-examples.git
```

8.2.3. 从命令行构建" Hello World"

继续并通过从菜单中选择 Windows> Visual Studio 2019> Developer Command Prompt 从 Windows 菜单打开 Developer Command Prompt 窗口。

然后按如下所示设置 Pico SDK 的路径，

```
C：\ Users \ pico \ Downloads> setx PICO_SDK_PATH" .. \ .. \ pico-sdk"
```

现在，您需要**关闭当前的命令提示符窗口**，然后打开第二个命令提示符窗口，在此之前可以正确设置此环境变量，然后再继续。

导航到 pico-examples 文件夹，并按如下所示构建" Hello World"示例，

```
C：\ Users \ pico \ Downloads> cd pico-examples
C：\ Users \ pico \ Downloads \ pico-examples> mkdir 构建
C：\ Users \ pico \ Downloads \ pico-examples> cd build C：\ Users \ pico \ Downloads \ pico-examples \ build> cmake -G" NMake Makefiles" ..
```

```
C: \ Users \ pico \ Downloads \ pico-examples \ build> nmake
```

建立目标。这将产生 ELF，bin 和 uf2 目标，您可以在构建目录内的 hello_world 目录中找到它们。UF2 二进制文件可以使用 USB 直接拖放到与计算机相连的 RP2040 板上。

8.2.4. 从 Visual Studio 代码构建“ Hello World”

现在，您已经安装了工具链，可以安装 Visual Studio Code 并在该环境中而不是从命令行构建项目。

继续下载并安装 Windows 的 Visual Studio 代码。安装后，通过从菜单中选择 Windows> Visual Studio 2019> Developer Command Prompt，从 Windows 菜单中打开 Developer Command Prompt 窗口。然后输入

```
C: >代码
```

在提示符下。这将打开带有所有正确环境变量集的 Visual Studio 代码，以便正确配置工具链。

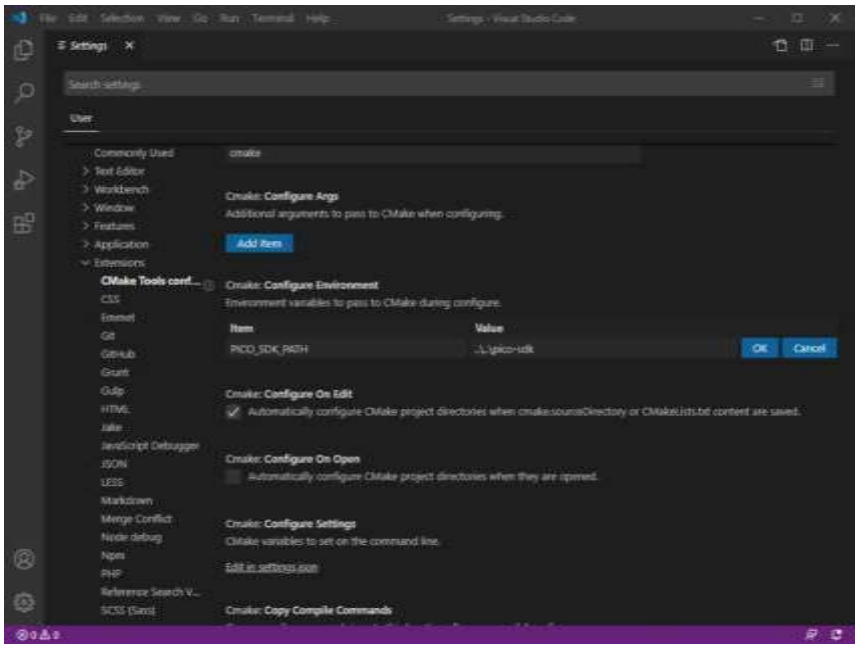
警告

如果通过单击桌面图标或直接从“开始”菜单启动 Visual Studio 代码，则将**无法**正确配置生成环境。尽管可以稍后在 CMake 工具设置中手动完成此操作，但是配置 Visual Studio Code 环境的最简单方法是从已设置了这些环境变量的“开发人员命令提示符”窗口中打开它。

现在，我们需要安装 CMake Tools 扩展。单击左侧工具栏中的扩展图标（或键入 Ctrl + Shift + X ），然后搜索“ CMake Tools”，然后单击列表中的条目，然后单击安装按钮。

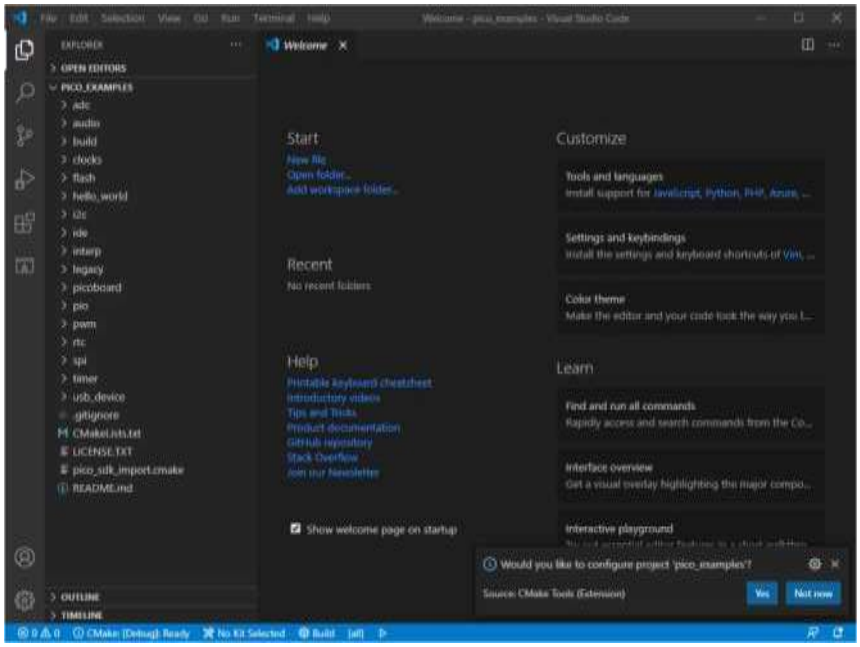
然后，单击界面左侧导航栏底部的齿轮，然后选择“设置”。然后在“设置”窗格中单击“扩展”和“ CMake 工具配置”。然后向下滚动到“ Cmake：配置环境”。单击“ Add Item”，然后将 PICO_SDK_PATH 设置为.. \ .. \ pico-sdk ，如图 18 所示。

图 18.在 CMake 扩展中设置PICO_SDK_PATH 环境变量



现在转到"文件"菜单，然后单击"打开文件夹"，并导航到 pico-examples 回购，然后单击"确定"。系统将提示您配置项目，请参见图 19。为您的编译器选择" GCC for arm-none-eabi"。

图19.提示在 Visual Studio Code 中配置您的项目。



继续并单击窗口底部蓝色栏中的"构建"按钮（带有齿轮）。这将创建构建目录并运行 CMake 并构建示例项目，包括" Hello World"。

这将产生 ELF，bin 和 uf2 目标，您可以在新创建的 build 目录内的 hello_world 目录中找到它们。UF2 二进制文件可以使用 USB 直接拖放到与计算机相连的 RP2040 板上。

8.2.5. 闪烁并运行" Hello World"

使用 micro-USB 电缆将 Raspberry Pi Pico 连接到 Raspberry Pi，并确保按住 BOOTSEL 按钮将其强制进入 USB Mass Storage Mode。该板应自动显示为外部驱动器。现在，您可以将 UF2 二进制文件拖放到外部驱动器上。

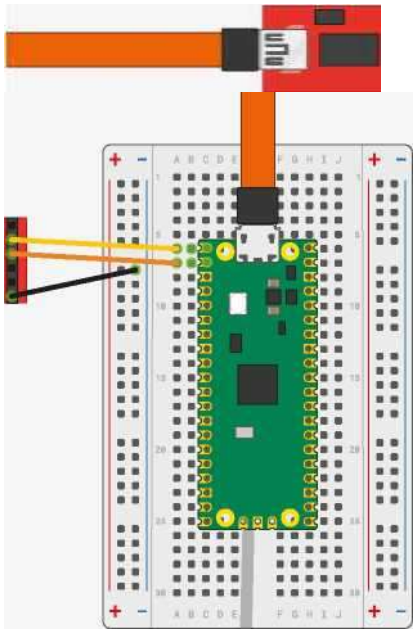
Raspberry Pi Pico 将重新启动，并将其自身卸载为外部驱动器，然后开始运行刷新的代码。

就像我们在第 4 章中所做的那样，您可以将 stdio 路由到 USB CDC（串行）或 GP0 和 GP1 引脚上的 UART0，从而构建 Hello World 示例。如果要使用 USB CDC 作为目标输出作为其符合类的设备进行构建，则无需安装驱动程序。

8.2.5.1. UART 输出

另外，如果你想你想连接到树莓派微微标准的 UART 看到输出，你需要使用一个 USB 转 UART 串行转换您的树莓派的 Pico 连接到 Mac，例如 SparkFun FTDI 基本 板，请参阅图 13。

图20.连接到Raspberry Pi Pico 的Sparkfun FTDI Basic 适配器



只要您使用的是 Windows 10 的最新版本，就应该已经加载了适当的驱动程序。否则，请参见制造商的 FTDI 芯片驱动器网站。

然后，如果尚未安装，请下载并安装 PuTTY。运行它，然后选择" Serial"，在" Speed"框中输入 115,200 作为波特率，并使用 UART 转换器使用的串行端口。如果您不知道这一点，可以使用 chgport 命令进行查找，

```
C：> chgport
COM4 = \设备\ ProlificSerial10
COM5 = \设备\ VCP0
```

这将为您提供活动串行端口的列表。USB 至 UART 串行转换器在此位于 COM5 上。

注意

如果您有多个串行设备，但无法确定哪个是 UART 转 USB 串行转换器，请尝试拔下电缆，然后再次运行 chgport 以查看哪个 COM 端口消失了。

输入速度和端口后，单击"打开"按钮，您应该在终端窗口中看到 Raspberry Pi Pico 的 UART 输出。

第 9 章使用其他集成开发环境

当前，推荐的集成开发环境（IDE）是 Visual Studio Code，请参见第 6 章。但是，其他环境也可以与 RP2040 和 Raspberry Pi Pico 一起使用。

9.1. 使用 Eclipse

Eclipse 是一个多平台集成开发环境（IDE），可用于 x86 Linux，Windows 和 Mac。此外，最新版本现已可用于 64 位 ARM 系统，并且在运行 64 位 OS 的 Raspberry Pi 4/400 系列（4GB 及更高版本）上运行良好。以下说明描述了如何在 Linux 设备上设置 Eclipse 以与 Raspberry Pi Pico 一起使用。尽管与 Raspberry Pi Pico 的连接会有所不同，但其他系统的说明大致相同。有关非 Linux 平台的更多详细信息，请参见第 8.2 节和第 8.1 节。

9.1.1. 在 Linux 机器上为 Pico 设置 Eclipse

先决条件：

- 运行最新版本 Linux 且具有至少 4GB RAM 的设备
- 64 位操作系统。
- CMake 3.11 或更高版本

注意

目前，64 位 Raspberry Pi OS 仍处于 beta 测试中。最新的 Beta 版本可在 http://downloads.raspberrypi.org/raspios_arm64/images/中找到。也可以使用其他 64 位 Linux 发行版，但未经我们测试，例如，用于 Raspberry Pi 的 Ubuntu。请按照通常的步骤在 SD 卡上安装操作系统映像。

如果使用 Raspberry Pi，则应通过在 config.txt 中添加以下内容来启用标准 UART

```
enable_uart = 1
```

您还应该安装 OpenOCD 和 SWD 调试系统。有关如何执行此操作的说明，请参见第 5 章。

9.1.1.1. 安装 Eclipse 和 Eclipse 插件

使用标准说明安装带有嵌入式 CDT 的最新版本的 Eclipse。如果在 ARM 平台上运行，则需要安装 AArch64（64 位 ARM）版本的 Eclipse。所有版本都可以在 eclipse 网站上找到。https://projects.eclipse.org/projects/iot.embed-cdt/downloads

下载适合您系统的正确文件，然后解压缩它。然后，您可以前往解压缩文件的地方并运行“ eclipse”可执行文件来运行它。

```
。/蚀
```

Eclipse 的 Embedded CDT 版本包括 C / C ++开发套件和 Embedded 开发套件，因此拥有为 Raspberry Pi Pico 开发所需的一切。

9.1.1.2. 使用微微示例

Pico 环境的标准构建系统是 CMake。但是，Eclipse 不使用 CMake，因为它具有自己的构建系统，因此我们需要将 pico-examples CMake 构建转换为 Eclipse 项目。

- 在与 pico-examples 文件夹相同的级别上，创建一个新文件夹，例如 pico-examples-eclipse
- 目录更改为该文件夹
- 将路径设置为 PICO_SDK_PATH
  - 导出 PICO_SDK_PATH = <where>

在命令行中输入：

cmake -G" Eclipse CDT4-Unix Makefiles" -D CMAKE_BUILD_TYPE =调试../pico-examples

这将使用原始 CMake 树中的源代码在 pico-examples-eclipse 文件夹中创建 Eclipse 项目文件。

现在，您可以使用"文件"菜单中的"从文件系统打开项目"选项将新项目文件加载到 Eclipse 中。

9.1.1.3. 建造

在项目浏览器中的项目上单击鼠标右键，然后选择 Build 。这将构建所有示例。

9.1.1.4. OpenOCD

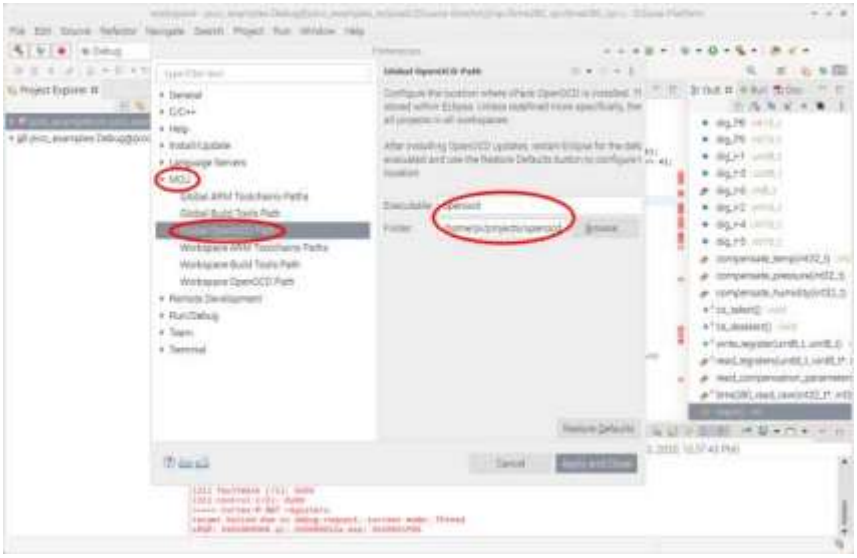
本示例使用 OpenOCD 系统与 Raspberry Pi Pico 通信。在运行代码之前，您需要提供从主机设备到 Raspberry Pi Pico 的 2 线调试连接。在 Raspberry Pi 上，这可以通过 GPIO 连接完成，但是在笔记本电脑或台式设备上，您将需要使用额外的硬件进行此连接。一种方法是使用第二个运行 Picoprobe 的 Raspberry Pi Pico，这在附录 A 中进行了描述。有关调试连接的更多说明，请参见第 5 章。

一旦安装了 OpenOCD 并建立了正确的连接，就需要在运行程序时将 Eclipse 设置为与 OpenOCD 对话。OpenOCD 为 Eclipse 提供了 GDB 接口，调试时使用的是该接口。

要设置 OpenOCD 系统，请从"窗口"菜单中选择"首选项" 。

单击 MCU 箭头以展开选项，然后单击 Global OpenOCD 路径。
对于可执行文件，输入" openocd"。对于文件夹，选择文件系统中从 github 克隆 Pico OpenOCD fork 的位置。

图 21. 在 Eclipse 中设置 OCD 可执行文件的名称和路径。

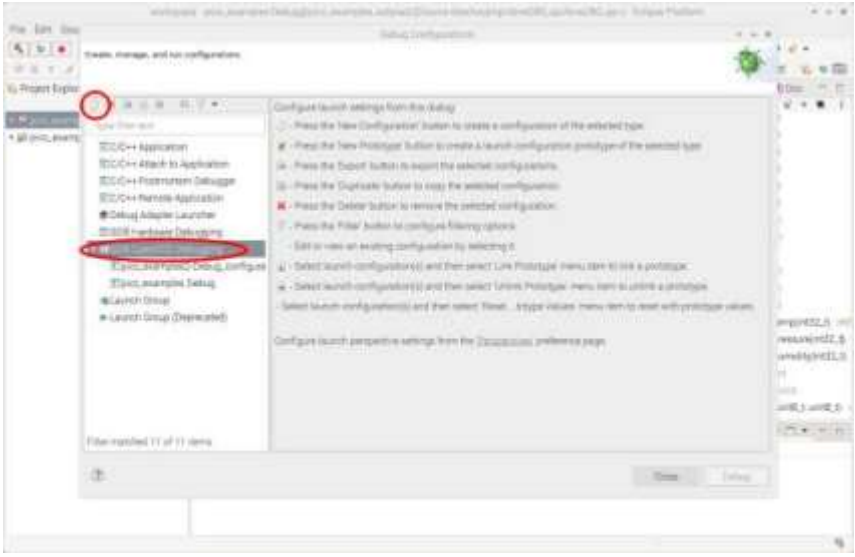


9.1.1.5. 创建运行配置

为了在 Eclipse 中运行或调试代码，您需要设置运行配置。这将设置识别运行代码所需的所有信息，任何参数，调试器，源路径和 SVD 信息。

在 Eclipse 运行菜单中，选择运行配置。要创建调试器配置，请选择 GDB OpenOCD 调试选项，然后选择"新建配置"按钮。

图22.在 Eclipse 中创建一个新的Run / Debug 配置。



9.1.1.5.1. 设置要运行的应用程序

由于 pico-examples 构建会创建许多不同的应用程序可执行文件，因此您需要选择要运行或调试的特定可执行文件。

在"运行"配置页面的"主要" 选项卡上，使用"浏览"选项选择要运行的 C / C ++应用程序。

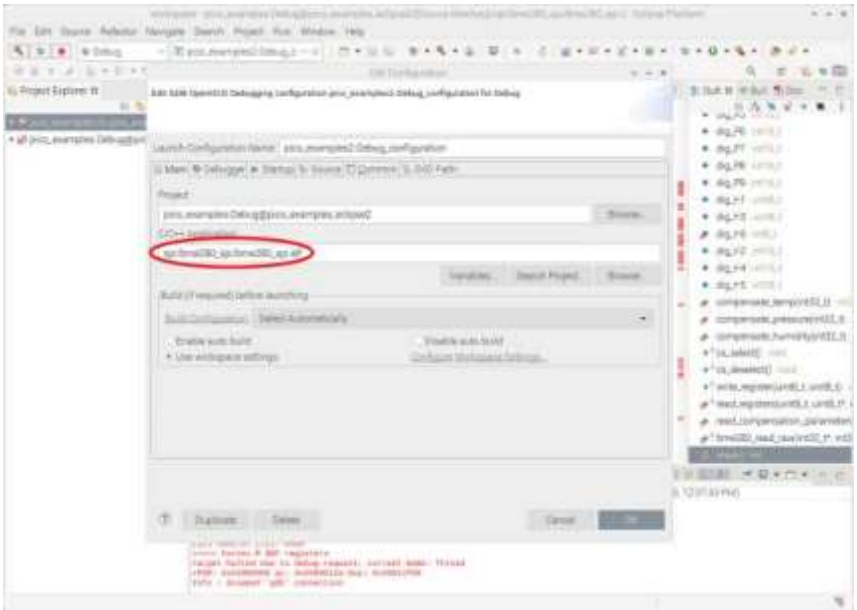
Eclipse 构建将在 Eclipse 项目文件夹的子文件夹中创建可执行文件。在我们的示例中，这是

■■7pico-examples-eclipse / <示例文件夹的名称> / <示例子文件夹的可选名称> /executable.elf

因此，例如，如果我们运行 LED 闪烁示例，可以在以下位置找到它：

■/ pico-examples-eclipse / blink / blink.elf

图23.将可执行文件设置为在 Eclipse 中调试。



9.1.1.5.2. 设置调试器

我们正在使用 OpenOCD 与 Raspberry Pi Pico 进行对话，因此我们需要进行设置。

在"可执行文件"和"实际可执行文件"框中设置 openocd 。我们还需要设置 OpenOCD 以使用 Pico 特定的配置，因此在 Config options 部分中添加以下内容。请注意，您将需要更改路径以指向 Pico 版本的 OpenOCD 的安装位置。

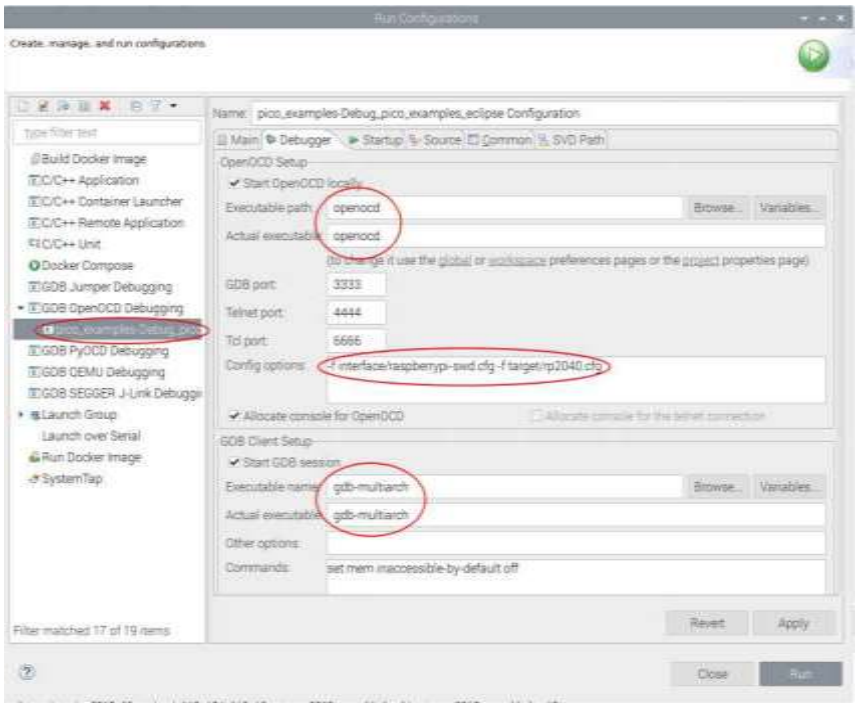
-f 接口/raspberrypi-swd.cfg -f 目标/rp2040.cfg

所有其他 OpenOCD 设置都应设置为默认值。

实际使用的调试器是 GDB。这会与 OpenOCD 调试器进行对话，以与 Raspberry Pi Pico 进行实际通信，但会提供与 IDE 的标准接口。

使用的 GDB 的特定版本是" gdb-multiarch"，因此请在"可执行文件名"和"实际可执行文件"字段中输入此名称。

图24.在 Eclipse 中设置 Debugger 和 OpenOCD。



9.1.1.5.3. 设置 SVD 插件

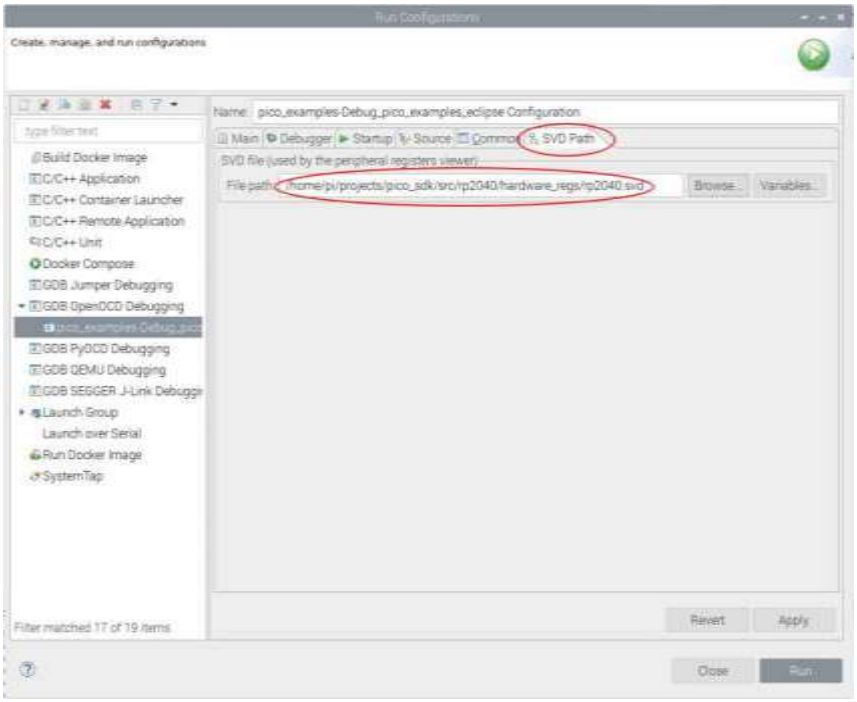
SVD 提供了一种在 Pico 板上查看和设置外围设备寄存器的机制。SVD 文件提供了寄存器的位置和描述，并且 Eclipse 的 SVD 插件将该功能集成到了 Eclipse IDE 中。SVD 插件是嵌入式开发插件的一部分。

选择"启动"配置上的" SVD 路径"选项卡，然后输入 SVD 文件在文件系统上的位置。通常可以在 pico-sdk 源代码树中找到它。

例如

■■■/ pico-sdk / src / rp2040 / hardware_regs / rp2040.svd

图25.在 Eclipse 中设置 SVD 路径。

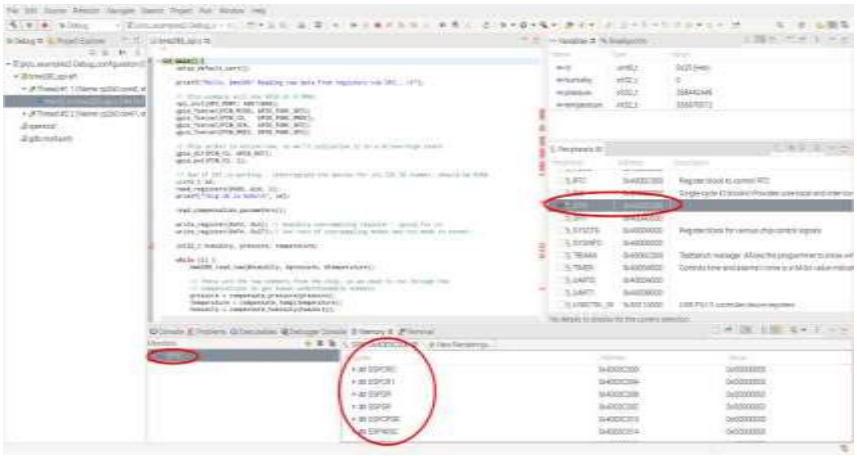


9.1.1.5.4. 运行调试器

运行配置完成并保存后，您可以使用对话框右下方的"运行" 按钮立即启动，或者直接应用更改并关闭对话框。然后，您可以使用"运行菜单调试"选项运行应用程序。

这将 Eclipse 设置为调试透视图，它将显示大量不同的调试和源代码窗口，以及非常有用的 Peripherals 视图，该视图使用 SVD 数据提供对外设寄存器的访问。从这一点开始，这是一个标准的 Eclipse 调试会话。

图26. Eclipse 调试器正在运行，显示了一些可用的调试窗口。



9.2. 使用 CLion

CLion 是 JetBrains 的多平台集成开发环境（IDE），可用于 Linux，Windows 和 Mac。这是一种商业 IDE，通常是专业开发人员（或喜欢 JetBrains IDE 的开发人员）的选择，尽管有免费或降价的许可证可用。它*可以*在 Raspberry Pi 上运行，但是性能并不理想，因此预计您将在台式机或笔记本电脑上使用 CLion。

尽管设置项目，开发和构建是轻而易举的，但是设置调试目前仍不是很主流

。

9.2.1. 设置 CLion

如果您打算使用 CLion，我们假定您已经安装了它或可以从 https://www.jetbrains.com/ clion /安装

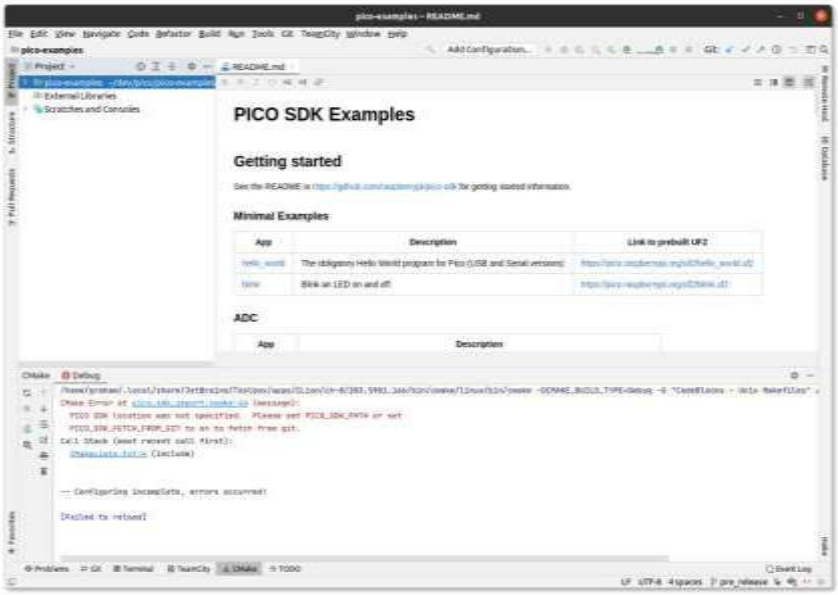
9.2.1.1. 建立一个项目

在这里，我们使用 pico-examples 作为示例项目。

要打开微微例子项目，选择打开-从文件菜单，然后导航并选择微微例子，你检出目录，然后按 OK。

打开后，您将看到类似图 27 的内容。

图 27. 一个新打开的 CLion picoexamples 项目。



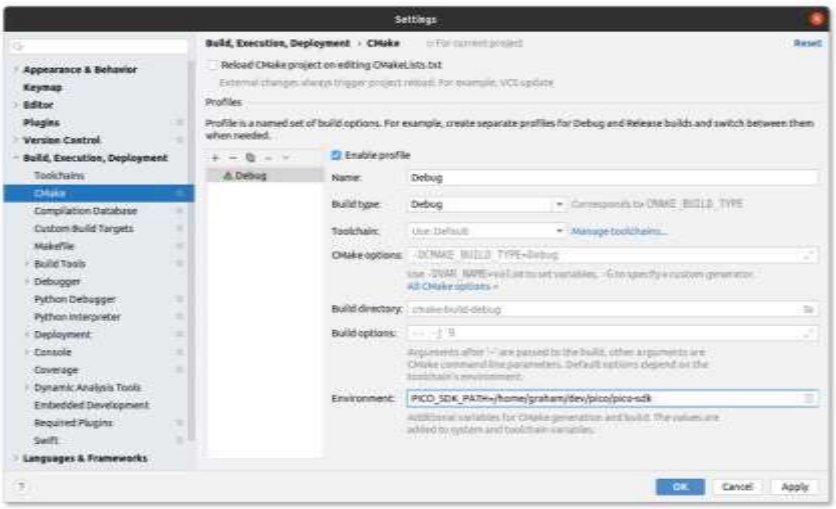
请注意，在底部，CLion 尝试加载 CMake 项目，但是出现了错误。即我们没有指定 PICO_SDK_PATH

9.2.1.1.1. 配置 CMake 配置文件

从"文件"菜单中选择"设置"，然后导航至"构建"，"执行"，"部署"下并选择" CMake"。

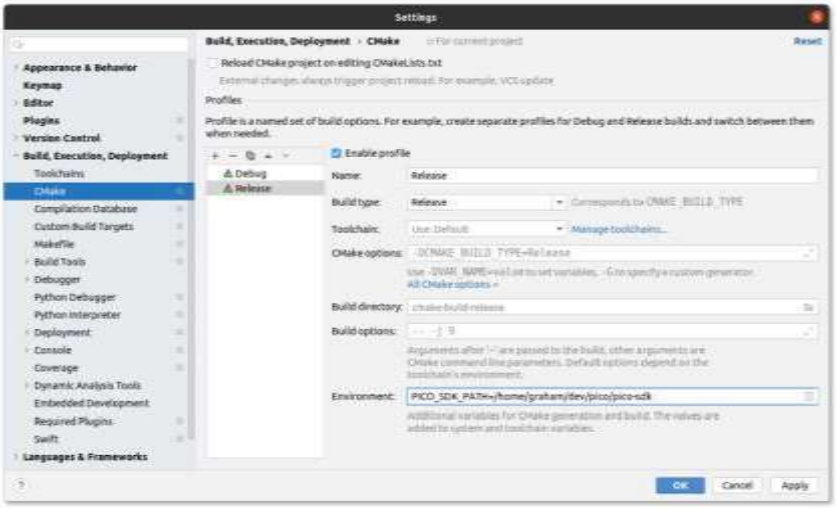
您可以在 Environment: 下设置环境变量 PICO_SDK_PATH，如图 28 所示，也可以在 CMake 选项下将其设置为-DPICO_SDK_PATH = xxx。这些就像从命令行调用 cmake 时的环境变量或命令行 args 一样，因此您可以在此指定 CMake 设置，例如 PICO_BOARD，PICO_TOOLCHAIN_PATH 等。

图28.在 CLion 中配置 CMake 配置文件。



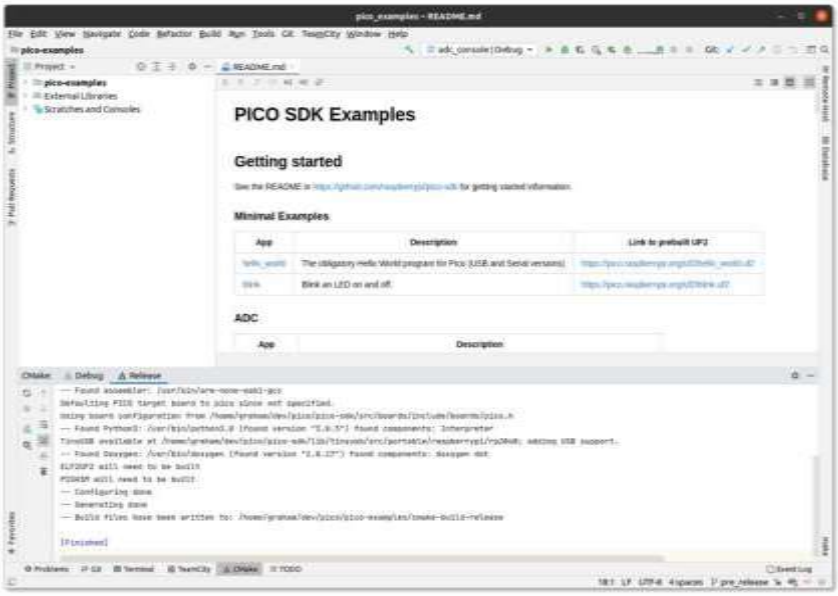
您可以使用不同的设置拥有任意数量的 CMake 配置文件。您可能想通过单击+按钮，然后再次填充 PICO_SDK_PATH，或单击右侧的复制按钮 2，并固定名称和设置来添加 Release 版本，请参见图 29。

图29.在 CLion 中配置第二个 CMake 配置文件。



按 OK 后，您将看到类似图 30 的内容。请注意， 窗口底部的两个配置文件（Debug 和 Release ）有两个选项卡。在这种情况下，选择了“发行”，您可以看到 CMake 安装成功。

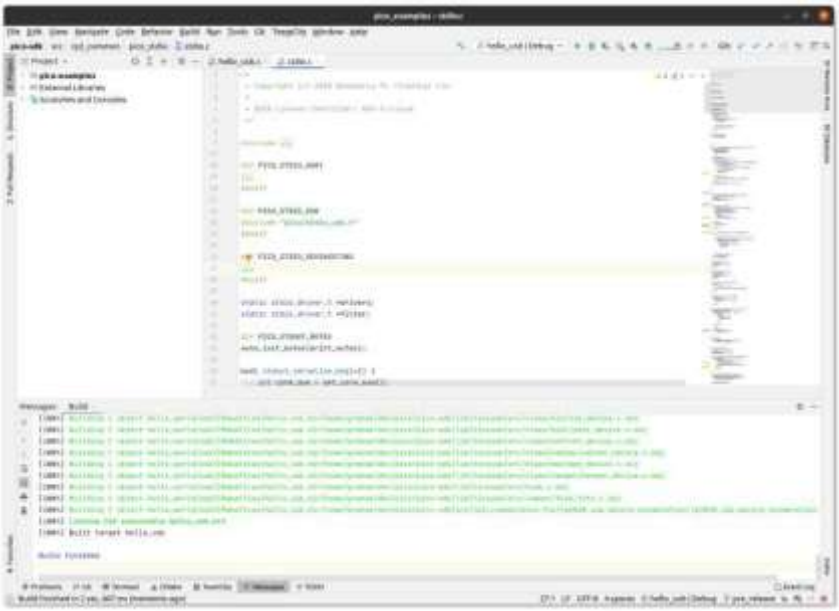
图 30. 在 CLion 中配置第二个 CMake 配置文件。



9.2.1.1.2. 运行构建

现在我们可以选择建立一个或多个目标。例如，您可以导航到工具栏中间的下拉选择器，然后选择或开始输入 hello_usb；然后按其左侧的工具图标进行构建（请参见图 31）。或者，您可以从"构建"菜单对所有目标或其他类型的构建进行完整构建。

图 31. hello_usb 成功构建。



请注意，下拉选择器使您既可以选择要构建的目标，也可以选择要使用的 CMake 配置文件（在这种情况下为 Debug 或 Release 之一）。

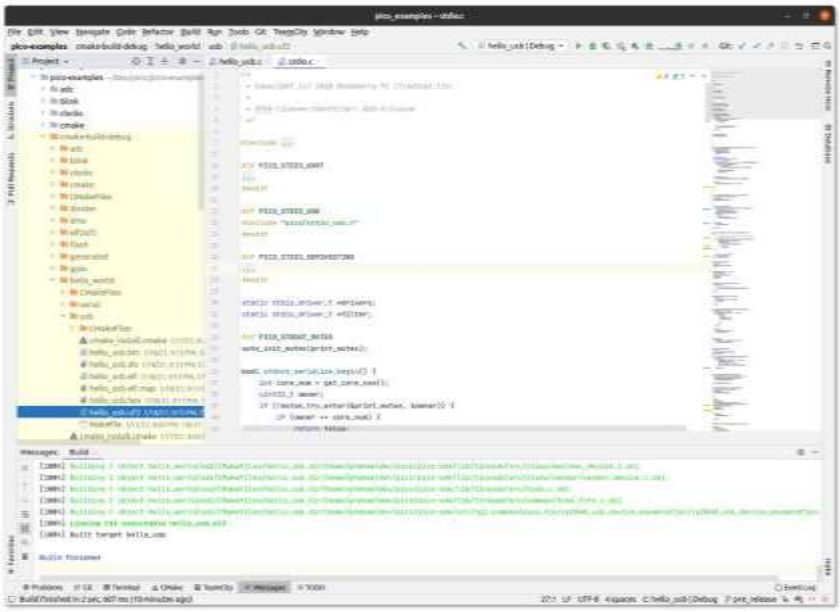
您 还将注意到图 31 所示的另一件事是，在底部状态栏中，您可以再次看到 hello_usb 和 Debug。这些向您显示目标和 CMake 配置文件，这些配置文件用于控制编辑器中的语法突出显示等（在您之前选择 hello_usb 时会自动选择它）。您可以在用户打开的 stdio.c 文件中直观地看到已设置 PICO_STDIO_USB，但未设置 PICO_STDIO_UART（这是 hello_usb 配置的一部分）。Pico SDK 中大量使用库的二进制配置的构建时间，因此这是一个非常好的功能。

9.2.1.1.3. 建立文物

构建工件位于项目根目录下的 cmake-build- <profile>下（请参见图 32）。在这种情况下，这是 cmake-build-debug 目录。

可以在 BOOTSEL 模式下将 UF2 文件复制到 RP2040 设备上，或者可以使用 ELF 进行调试。

图 32.找到hello_usb 构建工件



9.3. 其他环境

有许多可用的开发环境，我们无法在此处描述所有开发环境，但是您可以将其与 Pico SDK 一起使用。IDE 需要做很多事情，才能使 Raspberry Pi Pico 支持成为可能：

- CMake 集成
- 具有远程选项的 GDB 支持
- SVD。并非必需，但使读取外围设备状态更加容易
- 可选的 ARM 嵌入式开发插件。这些类型的插件通常会使支持更加容易。

9.3.1. 使用 openocd-svd

该 OpenOCD 的-SVD 工具是一个基于 Python 的 GUI 工具，使您可以通过 OpenOCD 的和 CMSIS-SVD 的组合获得 ARM 微控制器的外设寄存器。

要安装它，您应该首先安装依赖项，

```
$ sudo apt 安装 python3-pyqt5
$ pip3 install -U cmsus-svd
```

在克隆 openocd-svd git 存储库之前。

```
$ cd ~/ pico
$ git clone https://github.com/esynr3z/openocd-svd.git
```

确保您的 Raspberry Pi 4 和 Raspberry Pi Pico 正确连接在一起，我们可以通过 swd 和 rp2040 配置将 OpenOCD 连接到芯片。

```
$ openocd -f 接口/raspberrypi-swd.cfg -f 目标/rp2040.cfg
```

该 OpenOCD 终端需要保持打开状态。因此，继续打开另一个终端，在这个终端中，我们将 gdb 实例附加到 OpenOCD。

导航到您的项目，然后启动 gdb ，

```
$ cd ~/ pico / test
$ gdb-multiarch test.elf
```

将 GDB 连接到 OpenOCD，

```
(gdb) 目标远程 localhost：3333 并将其加载到 Flash 中，然后开始运行。
(gdb) 负载
(gdb) 监视器重置初始化
(gdb) 继续
```

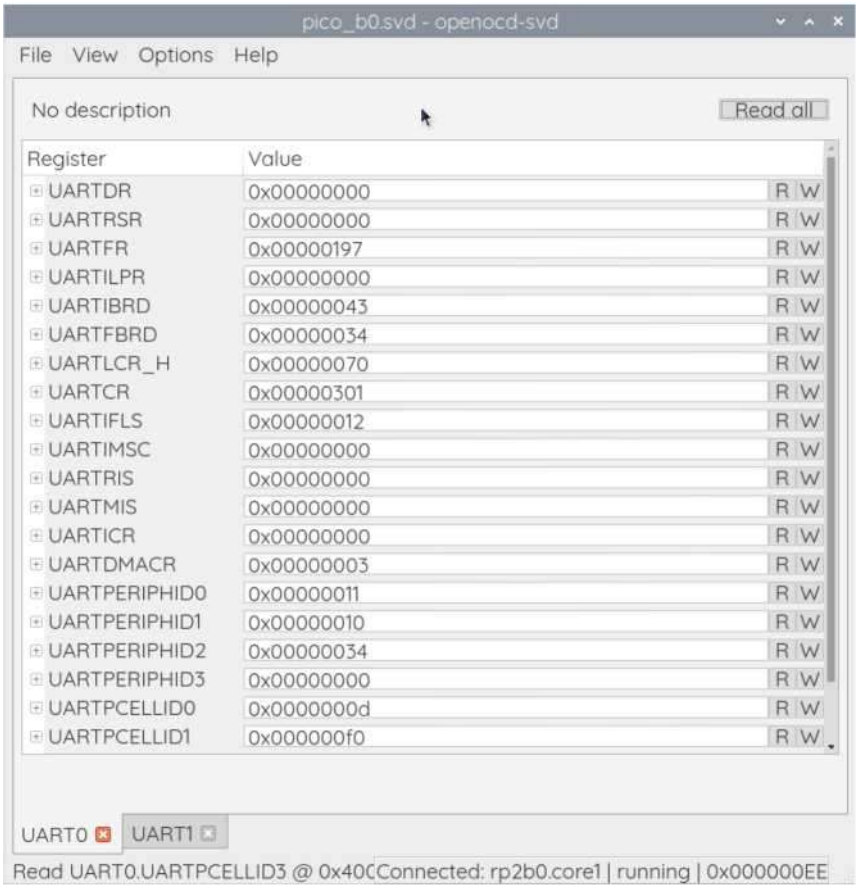
在同时运行 openocd 和 gdb 的情况下，打开第三个窗口，然后启动 openocd-svd 将其指向 Pico SDK 中的 SVD 文件。

```
$ python3 openocd_svd.py /home/pi/pico/pico-sdk/src/rp2040/hardware_regs/rp2040.svd
```

这将打开 openocd-svd 窗口。现在转到"文件"菜单，然后单击"连接 OpenOCD"以通过 telnet 连接到正在运行的 openocd 实例。

这将允许您检查在 Raspberry Pi Pico 上运行的代码的寄存器，请参见图 33。

图33.运行并连接到Raspberry Pi Pico的OpenOCD SVD。



附录 A：使用 Picoprobe

可以使用一个 Raspberry Pi Pico 调试另一个 Raspberry Pi Pico。这可以通过 picoprobe 实现，该应用程序允许 Raspberry Pi Pico 充当 USB SWD 和 UART 转换器。这使得在没有 GPIO 连接的非 Raspberry Pi 平台（例如 Windows，Mac 和 Linux 计算机）上轻松使用 Raspberry Pi Pico。

A.1。构建 OpenOCD

为了使 picoprobe 正常工作，您需要在启用 picoprobe 驱动程序的情况下构建 openocd。

A.1.1。的 Linux

```
$ cd〜/ pico
$ sudo apt install automake autoconf build-essential texinfo libtool libftdi-dev libusb-1.0-0- dev
$ git clone https://github.com/raspberrypi/openocd.git --branch picoprobe --depth = 1
$ cd openocd
$ ./引导
$的./configure --enable-picoprobe ®
$ make -j4
$ sudo make install
```

1.如果您在 Raspberry Pi 上构建，则还可以传递--enable-sysfsgpio --enable-bcm2835gpio，以允许通过 GPIO 引脚对 SWD 进行位 banging。

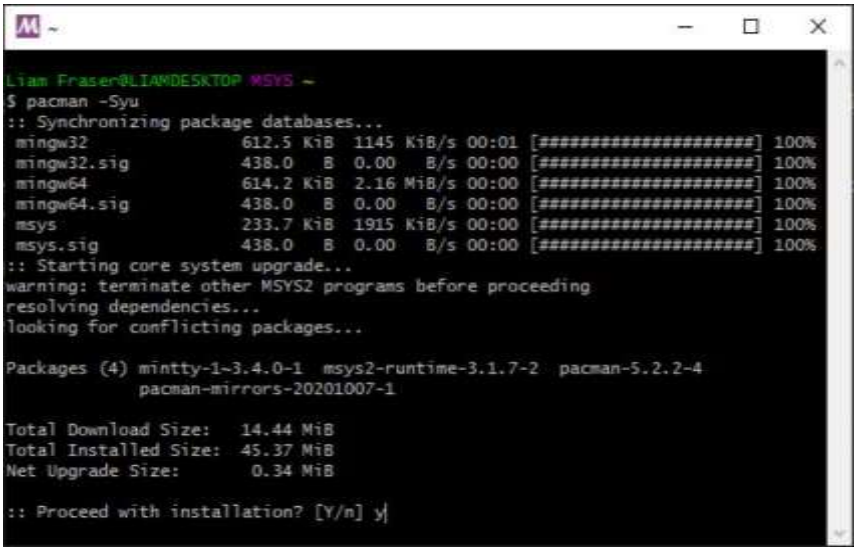
A.1.2。视窗

为了使构建 OpenOCD 尽可能容易，我们将使用 MSYS2。引用他们的网站：“ MSYS2 是工具和库的集合，为您提供了一个易于使用的环境，用于构建，安装和运行本机 Windows 软件。”

从 https://www.msys2.org/下载并运行安装程序。

首先使用以下命令更新软件包数据库和核心系统软件包：

```
吃豆人
```



如果 MSYS2 关闭，请再次启动它（确保选择 64 位版本）并运行

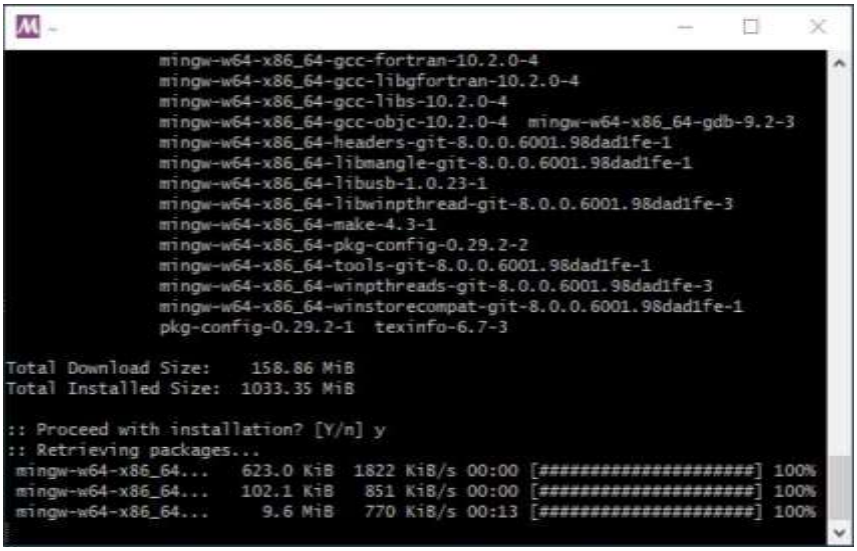
```
吃豆子
```

完成更新。

安装所需的依赖项：

```
pacman -S mingw-w64-x86_64-toolchain git make libtool pkg-config autoconf automake texinfo mingw-w64-x86_64-libusb
```

在安装 mingw-w64-x86_64 工具链时，请按 Enter 键以选择全部。



关闭 MSYS2，然后重新打开 64 位版本，以确保环境启动 GCC。

```
$ cd ~/ pico
$ git clone https://github.com/raspberrypi/openocd.git --branch picoprobe --depth = 1
$ cd openocd
$ ./引导
```

```
$的./configure --enable-picoprobe -禁用 werror ®
$ make -j4
```

1. 不幸的是，需要 disable-werror ，因为并非所有东西都能在 Windows 上正常编译

最后运行 OpenOCD 以检查其是否正确构建。由于未传递任何配置选项，因此可能会出错。

```
$ src / openocd.exe
Open On-Chip Debugger 0.10.0 + dev-gc231502-dirty（2020-10-14-14：37）根据 GNU GPL v2 许可
有关错误报告，请阅读
http://openocd.org/doc/doxygen/bugs.html
Embedded：startup.tcl：56：错误：找不到 openocd.cfg
在程序“脚本”中
在文件“ embedded：startup.tcl”，第 56 行
信息：在端口 6666 上监听 tcl 连接
信息：在端口 4444 上侦听 telnet 连接
错误：必须指定调试适配器，请参见嵌入的“接口”命令：startup.tcl：56：错误：
在程序“脚本”中
在文件“ embedded：startup.tcl”，第 56 行
```

A.1.3。苹果电脑

根据需要安装 brew

```
/ bin / bash -c“ $（curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh）”
```

安装依赖

```
冲泡安装 libtool 的 automake 的 libusb 的 wget 的 pkg 配置 GCC 的 Texinfo ®
```

1. OSX 随附的 texinfo 版本低于构建 OpenOCD 文档所需的版本

```
$ cd〜/ pico
$ git clone https://github.com/raspberrypi/openocd.git --branch picoprobe --depth = 1
$ cd openocd
$ export PATH =“ / usr / local / opt / texinfo / bin：$ PATH” ®
$ ./引导
$ ./configure --enable-picoprobe --disable-werror @
$ make -j4
```

1. 将新版本的 texinfo 放在路径上
2. 不幸的是，需要 disable-werror ，因为并非所有内容都能在 OSX 上正常编译

检查 OpenOCD 运行。由于未传递任何配置选项，因此可能会出错。

```
$ src / openocd
Open On-Chip Debugger 0.10.0 + dev-gc231502-dirty（2020-10-15-07：48）根据 GNU GPL v2 许可
有关错误报告，请阅读
http://openocd.org/doc/doxygen/bugs.html
Embedded：startup.tcl：56：错误：找不到 openocd.cfg
在程序“脚本”中
在文件“ embedded：startup.tcl”，第 56 行
信息：在端口 6666 上监听 tcl 连接
信息：在端口 4444 上侦听 telnet 连接
错误：必须指定调试适配器，请参见嵌入的“接口”命令：startup.tcl：56：错误：
在程序“脚本”中
在文件“ embedded：startup.tcl”，第 56 行
```

A2。构建并刷新 picoprobe

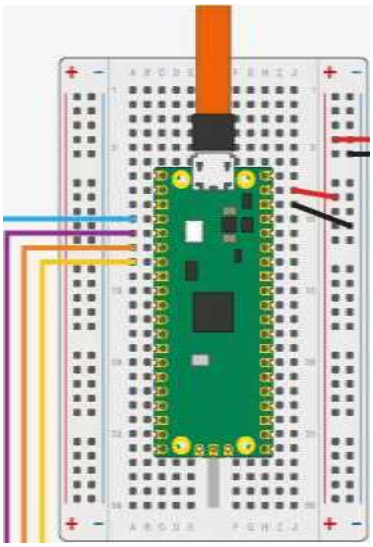
这些构建说明假定您正在 Linux 上运行。或者，您可以从 rptl.io/pico- picoprobe 获得 picoprobe 的 UF2。

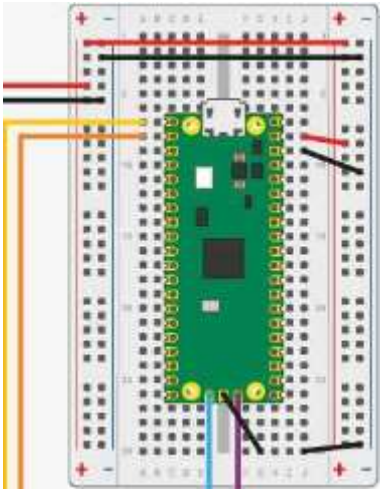
```
cd〜/ pico
git clone https://github.com/raspberrypi/picoprobe.git
光盘 picoprobe
mkdir 构建
光盘制作
cmake ..
使-j4
```

按下 BOOTSEL 按钮启动您想充当调试器的 Raspberry Pi Pico，并拖动 picoprobe.uf2 。

A.3。微型探头接线

图34.在 PicoA（左）和 PicoB（右）之间进行布线，将 Pico A 配置为调试器。请注意，如果 Pico B 是 USB 主机，则需要将 VBUS 连接到 VBUS，以便它可以为 VSYS 提供 5V 电压而不是 VSYS。





两块 Pico 板之间的配线架如图 34 所示。

Pico A GND-> Pico B GND
笔克 A GP2->
微型 B SWCLK
笔克 A GP3
-> Pico B SWDIO
Pico A GP4 / UART1 TX-> Pico B GP1 / UART0 RX
Pico A GP5 / UART1 RX-> Pico B GP0 / UART0 TX

（可选）要从 Pico B 为 Pico A 供电，您还应该接线，

Pico A VSYS-> Pico B VSYS

重要

图 34 上的一个小警告 是，如果 Pico B 是 USB 主机，则您希望将 VBUS 连接到 VBUS，以便它可以为 VSYS 提供 5V 电压而不是 VSYS。

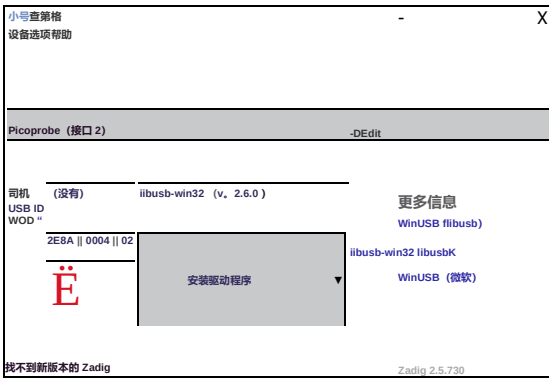
A.4。安装 Picoprobe 驱动程序（仅在 Windows 上需要）

Picoprobe 设备具有两个 USB 接口：

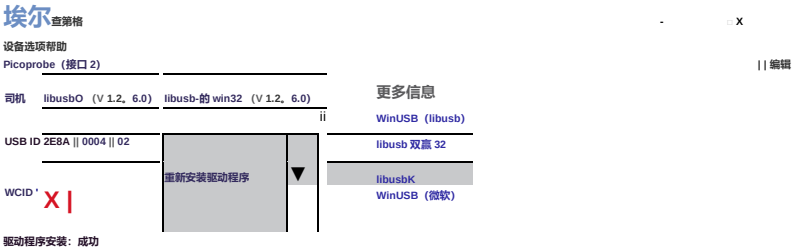
1. 符合类的 CDC UART（串行端口），这意味着它可以在 Windows 上直接使用
2. 供应商专用的 SWD 探测数据接口。这意味着我们需要安装驱动程序以使其运行。我们将为此使用 Zadig（http://zadig.akeo.ie）。

下载并运行 Zadig。

从下拉框中选择 Picoprobe（接口 2）。选择 libusb-win32 作为驱动程序。



然后选择安装驱动程序。



A.5。使用 Picoprobe 的 UART

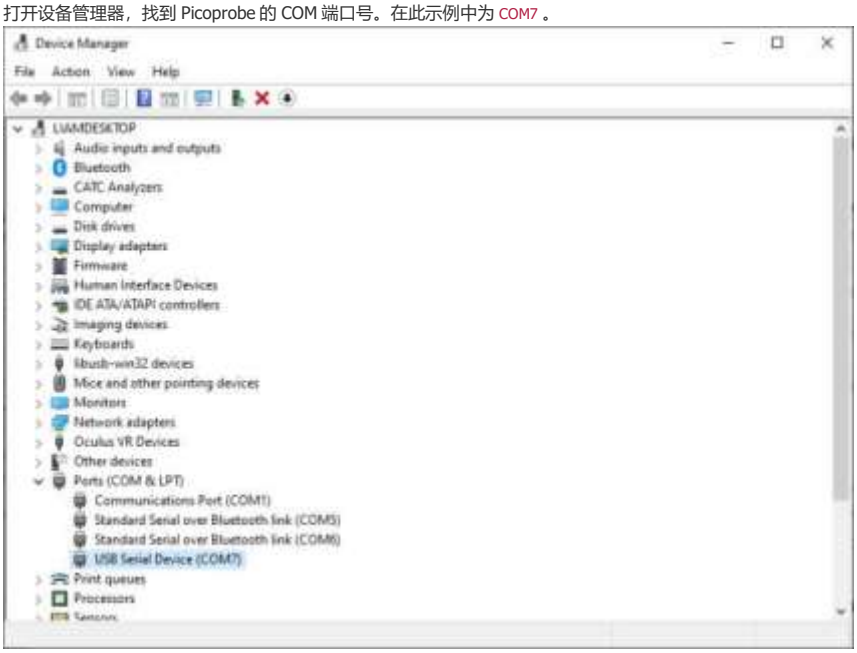
A.5.1。的 Linux

```
须藤 minicom -D / dev / ttyACM0 -b 115200
```

A.5.2。视窗

下载并安装 PuTTY https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

打开设备管理器，找到 Picoprobe 的 COM 端口号。在此示例中为 COM7 。



打开腻子。在连接类型下选择串行 。然后键入您的 COM 端口的名称以及 115200 作为速度。

PuTTV 配置

类别：

届会

记录-■■终端 ■■■■ 键盘

| k 贝尔

我 - 特点

H 窗

我 我-外观

行为

我 ！■■■■ 翻译

我 ！■■■■ 选择

我大号-颜色

连接

数据

PuTTY 会话的基本选项

指定您要连接的目的地

串行线 速度

C0M7

连接类型：

O 缺陷 O Telnet O Rlogin ObSH

1152DCI

®系列

加载，保存或删除存储的会话

保存的会话

关闭退出窗口：O 始终 O 永不

关于	救命	打开	取消

选择打开以启动串行控制台。您现在可以运行您的应用程序了！

A.5.3。苹果电脑

```
brew install minicom
minicom -D /dev/tty.usbmodem1234561 -b 115200
```

A.6。将 Picoprobe 与 OpenOCD 结合使用

所有平台均相同

```
src / openocd -f 接口/picobrobe.cfg -f 目标/rp2040.cfg -s tcl
```

像往常一样连接 GDB

```
目标远程本地主机：3333
```

附录 B：使用 Picotool

可以将信息嵌入 Raspberry Pi Pico 二进制文件中，可以使用名为 picotool 的命令行实用程序进行检索。

B.1。获取 picotool

该 picotool 实用程序是在一个单独的存储库中的微微 SDK。如果您尚未运行 pico-setup 脚本，则需要克隆并构建它。

```
$ git clone -b master https://github.com/raspberrypi/picotool.git
$ cd picotool
```

您还需要安装 libusb（如果尚未安装），

```
$ sudo apt-get install libusb-1.0-0-dev
```

注意

如果您要在 macOS 上构建 picotool，则可以使用 Homebrew 安装 libusb，

$ brew 安装 libusb

如果您使用的是 Microsoft Windows，则可以直接从 libusb.info 站点下载并安装 libusb 的 Windows 二进制文件。

B.2。建筑 picotool

可以如下构建 picotool，

```
$ mkdir 构建
$ cd 构建
$ export PICO_SDK_PATH =〜/ pico / pico-sdk
$ cmake ../
$使
```

这将在 build / picotool 目录中生成 picotool 命令行二进制文件。

注意

如果您在 Microsoft Windows 上构建，则应按以下方式调用 CMake，

C：\ Users \ pico \ picotool> mkdir 构建

C：\ Users \ pico \ picotool> cd 构建

C：\ Users \ pico \ picotool \ build> cmake .. -G“ NMake Makefiles”

C：\ Users \ pico \ picotool \ build> nmake

B.3。使用 picotool

该 picotool 二进制包括命令行帮助功能，

```
$ picotool 帮助
微型工具：
在 BOOTSEL 模式下与 RP2040 设备交互或与 RP2040 二进制文件交互的工具
SYNOPSYS：
微型工具 信息 [-b ] [-p] [-d] [-l] [-a] [--bus <总线>] - 地址 <addr>]
微型工具 信息 [-b ] [-p] [-d] [- l] [-a] <文件名> [-t <类型>]
微型工具 加载 [-v ] [-r] <文件名> [-t <类型>] [-总线 <总线>] [- -address <地址>
微型工具 保存 [-p ] [--bus <总线> ] [-地址<addr>] <文件名> [-t <类型>]
微型工具 保存 -一种 [ - 总线 <巴士>] [-地址<addr>] <文件名> [- t <类型>]
微型工具 保存 -r <来自> <至> [- -bus <总线>] [-地址 <addr>] < 文件名> [-t <类型>
微型工具
]验证[--bus <总线>] [-地址<addr>] <文件名> [-t <类型>] [-r <从> <到>]
picotool 重新启动[-a] [-u] [--bus <总线>] [-地址<addr>] picotool 帮助[<cmd>]
指令：
info 显示来自目标设备或文件的信息。
不带任何参数的情况下，它将显示所有已连接的 RP2040 设备的基本信息。
BOOTSEL 模式
负载 加载存储在文件到设备中的程序/存储器范围。
保存 将设备闪存中存储的程序/内存保存到文件中。
验证检查设备内容是否与文件中的内容匹配。
重新启动重新启动设备
显示常规帮助或特定命令的帮助
使用“ picotool 帮助<cmd>”了解更多信息
```

注意

大多数命令都需要以 BOOTSEL 模式连接 RP2040 设备。

B.3.1。显示信息

因此，SDK 现在提供了*二进制信息*支持，可以轻松存储 picotool 可以找到的 紧凑信息（请参阅下面的 B.4 节）。该信息的命令是用于读取这些信息。

重要

自 picotool

可以在 BOOTSEL 模式下从一个或多个连接的 RP2040 设备中读取信息，也可以从文件中读取信息。该文件可以是 ELF，UF2 或 BIN 文件。

```
$ picotool 帮助信息

信息：

显示来自目标设备或文件的信息。

不带任何参数，它将在 USB 启动中显示所有连接的 RP2040 设备的基本信息

模式

SYNOPSYS：

picotool 信息[-b] [-p] [-d] [-l] [-a] [--bus <bus>] [--address <addr>]

picotool 信息[-b] [-p] [-d] [-l] [-a] <文件名> [-t <类型>]

选项：

显示信息

-b，    -基本
        包括    基本信息。这是默认值
-p，    -大头针
        包括    密码信息
-d      - 设备
        包括    设备信息
-l      - 建立
        包括    建立属性
-一种，  - 所有
        包括    所有资讯

        目标选择：
        在 BOOTSEL 模式下以一台或多台连接的 RP2040 设备为目标（默认）
        --bus <总线>
通过 USB 总线号过滤设备
        --address <地址>
通过 USB 设备地址过滤设备
        定位文件
        <文件名>
档案名称
        -t <类型>
明确指定文件类型（uf2 | elf | bin），忽略文件扩展名
```

例如，通过按住 Raspberry Pi Pico 的大容量存储模式，将其连接到计算机。

将 BOOTSEL 按钮插入 USB 之前。然后打开"终端"窗口并键入，

```
$ sudo picotool 信息
计划信息
名称：          hello_world
功能：标准输出到 UART
```

要么，

```
$ sudo picotool 信息-a
计划信息
名称：                hello_world
```

```
    功能:                    标准输出到 UART
    二进制开始: 0x10000000
    二进制结尾:          0x1000606c
固定引脚信息
20: UART1 发送
21: UART1 接收

建立 信息
    建立日期:                    2020 年 12 月 31 日
    构建属性: 调试构建
    设备信息
    闪光灯大小:          2048K
    ROM 版本: 2
```

欲获得更多信息。或者，您可以按以下方式获取有关使用的引脚的信息，

```
$ sudo picotool info -bp 程序信息名称:       hello_world

    功能: 标准输出到 UART

固定引脚信息 20: UART1 TX 21: UART1 RX
```

该工具还可用于仍在本地文件系统上的二进制文件，

```
$ picotool info -a lcd_1602_i2c.uf2 文件 lcd_1602_i2c.uf2:

  计划信息

    名称:        lcd_1602_i2c
    网站:        https://github.com/raspberrypi/pico-examples/tree/HEAD/i2c/lcd_1602_i2c
    二进制开始: 0x10000000
    二进制结束: 0x10003c1c

    固定引脚信息
    4: I2C0 SDA
    5: I2C0 SCL
    建造信息
    建立日期: 2020 年 12 月 31 日
```

B.3.2。保存程序

保存使您可以将设备的一系列内存或程序或整个闪存保存到 BIN 文件或 UF2 文件中。

```
$ picotool 帮助保存
保存:
将设备上闪存中存储的程序/内存保存到文件中。
SYNOPSYS:
picotool 保存[-p] [--bus <总线>] [-地址<addr>] <文件名> [-t <类型>]
```

```
        picotool save -a [--bus <总线>] [-地址<addr>] <文件名> [-t <类型>]
        picotool save -r <从> <至> [--bus <总线>] [-地址<addr>] <文件名> [-t <类型>]
        选项：
        选择要保存的数据
        -p，-程序
仅保存安装的程序。这是默认值
        -a，-全部
保存所有闪存
        -r，--range
节省内存范围；请注意，范围扩展到 256 个字节边界
        <来自>
十六进制的低位地址
        <至>
十六进制的上限地址
        源设备选择
        --bus <总线>
通过 USB 总线号过滤设备
        --address <地址>
通过 USB 设备地址过滤设备
        文件保存到
        <文件名>
档案名称
        -t <类型>
明确指定文件类型（uf2 | elf | bin），忽略文件扩展名
```

例如，

```
$ sudo picotool info 程序信息名称：  lcd_1602_i2c

        网站：https : //github.com/raspberrypi/pico-examples/tree/HEAD/i2c/lcd_1602_i2c
        $ picotool 保存 spoon.uf2
        保存文件：[==============================]                    100%
        将 51200 字节写入 Spoon.uf2
        $ picotool 信息勺.uf2
        文件 spoon.uf2：
        计划信息
        名称：          lcd_1602_i2c
        网站：https : //github.com/raspberrypi/pico-examples/tree/HEAD/i2c/lcd_1602_i2c
```

B.4。二进制信息

二进制信息可以在机器上找到，并且通常可以在机器上使用。我之所以这样说，通常是因为任何人都可以包括任何信息，我们可以从我们的信息中分辨出来，但这取决于他们是否使自己的数据自我描述。

B.4.1。基本信息

当您拿起 Pico 却不知道上面有什么时，此信息非常方便！

基本信息包括

- 程序名称
- 程序说明

- 程序版本字符串
- 程序建立日期
- 程序网址
- 程序结束地址

- 程序功能，这是一个由二进制文件中的单个字符串构建的列表，可以在 SDK 中显示（例如，我们将为 UART stdio 提供一个列表，为 USB stdio 提供一个列表）
  - 构建属性，这是类似于字符串的列表，用于与二进制文件本身有关的事情（例如，Debug Build）

B.4.2。别针

当您有一个名为 hello_world.elf 的执行程序时，这当然很方便，但是您忘了它是为基于 RP2040 的板而开发的，因为不同的板可能有不同的引脚。

静态（固定）引脚分配可以非常紧凑的形式记录在二进制文件中：

```
$ picotool info --pins sprite_demo.elf 文件 sprite_demo.elf:
    固定    引脚信息
    0-4:            红色 0-4
    6-10:           绿色 0-4
11-15：蓝色 0-4
    16:             同步
    17:             垂直同步
    18:             显示启用
    19:             像素时钟
    20:             UART1 发送
    21:             UART1 接收
```

B.4.3。包括二进制信息

二进制信息通过宏在程序中声明；对于前面的示例：

```
        $ picotool 信息--pins sprite_demo.elf
        文件 sprite_demo.elf:
固定引脚信息
0-4:                红色 0-4
6-10:               绿色 0-4
11-15：蓝色 0-4
16:             同步
17:             垂直同步
18:             显示启用
19:             像素时钟
20:             UART1 发送
21:             UART1 接收
```

setup_default_uart 函数中

只有一行：bi_decl_if_func_used（bi_2pins_with_func（PICO_DEFAULT_UART_RX_PIN，PICO_DEFAULT_UART_TX_PIN，GPIO_FUNC_UART））；
将存储两个引脚号和功能 UART，然后通过 picotool 将其解码为它们的实际功能名称（UART1 TX 等）。该 bi_decl_if_func_used 确保如果包含函数被调用时只包含二进制信息。

同样，视频代码包含以下几行：

```
bi_decl_if_func_used（bi_pin_mask_with_name（0x1f << （PICO_SCANVIDEO_COLOR_PIN_BASE +
PICO_SCANVIDEO_DPI_PIXEL_RSHIFT），“红色 0-4”））；
```

B.4.4。细节

设计事物时要浪费尽可能少的空间，但是您可以使用预处理器 var PICO_NO_BINARY_INFO = 1 来解决所有问题。此外，任何插入二进制信息拐杖的 SDK 代码都由其自己的 preprocesor var 单独排除。

你需要，

```
#include“ pico / binary_info.h”
```

标头中有一堆 bi_ 宏

```
#define bi_binary_end（end）
#定义 bi_program_name（名称）
#define bi_program_description（说明）
#define bi_program_version_string（version_string）
#define bi_program_build_date_string（date_string）
#define bi_program_url（url）
#定义 bi_program_feature（功能）
#define bi_program_build_attribute（attr）
#定义 bi_1pin_with_func（p0，func）
#定义 bi_2pins_with_func（p0，p1， func）
#定义 bi_3pins_with_func（p0，p1， p2， func）
#定义 bi_4pins_with_func（p0，p1， p2， p3， func）
#定义 bi_5pins_with_func（p0，p1， p2， p3， p4，func）
#define bi_pin_range_with_func（plo，phi，func）#define bi_pin_mask_with_name（pmask，标签）
#定义 bi_pin_mask_with_names（pmask，标签）
#define bi_1pin_with_name（p0，name）#define bi_2pins_with_names（p0，name0，#define bi_3pins_with_names（p0，name0，#define
bi_4pins_with_names（p0，name0，
p1，名称 1）
p1，name1，p2，name2）
p1，name1，p2，name2，p3，name3）
```

利用基础宏，例如

```
#define bi_program_url（url）bi_string（BINARY_INFO_TAG_RASPBERRY_PI，BINARY_INFO_ID_RP_PROGRAM_URL，URL）
```

然后，您可以使用 bi_decl（bi_blah（---））无条件地包含二进制信息等等，或使用

bi_decl_if_func_used（bi_blah（—））来获取二进制信息，如果链接器未将封闭函数包含在二进制文件中，则可能会删除二进制信息（请考虑--gc-sections）。

例如，

```
1  #include <stdio.h>
2  #include“ pico / stdlib.h”
3  #include“ hardware / gpio.h”
4  #include“ pico / binary_info.h”
5
5  常量 LED_PIN = 25 ;
7
6  int main（）{
9
7      bi_decl（bi_program_description（“这是一个测试二进制文件。”））；
8      bi_decl（bi_1pin_with_name（LED_PIN，“板载 LED”））；
12
13
14
15
16
17
18
19
20
21
22
23}
setup_default_uart（）；

gpio_set_function（LED_PIN，GPIO_FUNC_PROC）；gpio_set_dir（LED_PIN，GPIO_OUT）；

而（1）{

    gpio_put（LED_PIN，0）；sleep_ms（250）；

    gpio_put（LED_PIN，1）；

    puts（“ Hello World \ n”）；sleep_ms（1000）；

}
```

当用 picotool 查询时，

```
$ sudo picotool 信息-a test.uf2
文件 test.uf2：
计划信息
名称：          测试
描述：          这是一个测试二进制文件。
功能：          标准输出到 UART
二进制开始：     0x10000000
二进制结尾：     0x100031f8
固定 引脚信息
0：  UART0 TX
1：  UART0 接收
25：板载 LED

建造信息
建立日期：2021 年 1 月 4 日
```

在输出中显示我们的信息字符串。

B.4.5。从 CMake 设置公共字段

您也可以直接从项目的 CMake 的文件中，例如，设置字段

pico_set_program_name（富“未富”）® pico_set_program_description（富“这是一个富”）pico_set_prorgam_version_string（富“0.00001a”）pico_set_program_url（富“ www.plinth.com / foo “）

1.名称“ foo”将是默认名称。

注意

所有这些都是作为命令行参数传递给编译的，因此，如果您打算使用引号，换行符等，则可以更好地在代码中使用 bi_decl 对其进行定义。

Raspberry Pi 是 Raspberry Pi Foundation 的商标
树莓派贸易有限公司