Colophon

дата сборки: 2023-06-14
версия сборки: a6fe703-clean

О SDK

По всему тексту "SDK" относится к нашему Raspberry Pi Pico SDK. Более подробную информацию о SDK можно найти в книге Raspberry Pi Pico C/C++ SDK. Исходный код, включенный в документацию, является Авторское право © 2020-2023 Raspberry Pi Ltd (ранее Raspberry Pi (Trading) Ltd.) и лицензировано в соответствии

Уведомление о правовом отказе от ответственности

ТЕХНИЧЕСКИЕ ДАННЫЕ И ИНФОРМАЦИЯ О НАДЕЖНОСТИ ПРОДУКТОВ RASPBERRY PI (ВКЛЮЧАЯ СПЕЦИФИКАЦИИ), ИЗМЕНЕННЫЕ С ВРЕМЯ ОТ ВРЕМЕНИ (“РЕСУРСЫ”) ПРЕДОСТАВЛЯЮТСЯ RASPBERRY PI LTD (“RPL”) "КАК ЕСТЬ" И ЛЮБЫМИ ЯВНЫМИ ИЛИ ПОДРАЗУМЕВАЕМЫМИ ГАРАНТИЙ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ, ПОДРАЗУМЕВАЕМЫЕ ГАРАНТИИ КОММЕРЧЕСКОЙ ЦЕННОСТИ И ПРИГОДНОСТИ ДЛЯ КОНКРЕТНОЙ ЦЕЛИ ОТРИЦАЮТСЯ. В МАКСИМАЛЬНОЙ СТЕПЕНИ, РАЗРЕШЕННОЙ ПРИМЕНИМЫМ ЗАКОНОДАТЕЛЬСТВОМ, НИ В КОЕМ СЛУЧАЕ СОБЫТИЕ RPL НЕ НЕСЕТ ОТВЕТСТВЕННОСТИ ЗА ЛЮБЫЕ ПРЯМЫЕ, КОСВЕННЫЕ, СЛУЧАЙНЫЕ, ОСОБЫЕ, ПОКАЗАТЕЛЬНЫЕ ИЛИ ЛОГИЧЕСКИ ВЫТЕКАЮЩИЕ УЩЕРБ (ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ЭТИМ, ПРИОБРЕТЕНИЕ ЗАМЕНЯЮЩИХ ТОВАРОВ ИЛИ УСЛУГ; УТРАТА ВОЗМОЖНОСТИ ИСПОЛЬЗОВАНИЯ, ДАННЫЕ, ИЛИ ПРИБЫЛЬ; ИЛИ ПРЕРЫВАНИЕ БИЗНЕСА), НЕЗАВИСИМО ОТ ПРИЧИНЫ И НА ОСНОВАНИИ ЛЮБОЙ ТЕОРИИ ОТВЕТСТВЕННОСТИ, БУДЬ ТО В КОНТРАКТЕ, СТРОГОЙ ОТВЕТСТВЕННОСТИ ИЛИ ДЕЛИКТЕ (ВКЛЮЧАЯ ХАЛАТНОСТЬ ИЛИ ИНЫМ ОБРАЗОМ), ВОЗНИКАЮЩЕМ КАКИМ-ЛИБО ОБРАЗОМ В РЕЗУЛЬТАТЕ ИСПОЛЬЗОВАНИЕ РЕСУРСОВ, ДАЖЕ ЕСЛИ ОНИ ПРЕДУПРЕЖДЕНЫ О ВОЗМОЖНОСТИ ТАКОГО УЩЕРБА.

RPL оставляет за собой право вносить любые дополнения, доработки, исправления или любые другие модификации в РЕСУРСЫ или любые продукты, описанные в них, в любое время и без дополнительного уведомления. РЕСУРСЫ предназначены для квалифицированных пользователей с соответствующим уровнем знаний в области проектирования. Пользователи несут единоличную ответственность за свой выбор и использование РЕСУРСОВ и любое применение продуктов, описанных в них. Пользователь соглашается возместить и обезопасить RPL от всех обязательств, издержек, ущерба или других убытков, возникающих в результате использования им РЕСУРСОВ.

RPL предоставляет пользователям разрешение на использование РЕСУРСОВ исключительно в сочетании с продуктами Raspberry Pi. Любое другое использование РЕСУРСОВ запрещено. Никакая лицензия не предоставляется на какие-либо другие права интеллектуальной собственности RPL или других третьих лиц. ДЕЯТЕЛЬНОСТЬ С ВЫСОКИМ РИСКОМ. Продукты Raspberry Pi не разрабатываются, не производятся и не предназначены для использования во взрывоопасных средах, требующих безотказной работы, например, при эксплуатации ядерных установок, систем навигации или связи воздушных судов, управления воздушным движением, систем вооружения или критически важных приложений (включая системы жизнеобеспечения и другие медицинские устройства), в которых выход из строя изделий может непосредственно привести к смерти, травмам персонала или серьезному физическому ущербу или ущербу окружающей среде (“Деятельность с высоким риском”). RPL определенно отказывается от любых явных или подразумеваемых гарантирует пригодность для деятельности с высоким уровнем риска и не несет ответственности за использование или включение продуктов Raspberry Pi в деятельность с высоким уровнем риска.

.

Продукты Raspberry Pi предоставляются в соответствии со стандартными условиями RPL.
Предоставление RPL РЕСУРСОВ не

расширяет и иным образом не изменяет Стандартные условия RPL, включая, но не ограничиваясь , выраженными в них отказами от ответственности и гарантиями.

Содержание

Глава 1. Быстрая настройка Pico

Если вы разрабатываете для Raspberry Pi Pico на Raspberry Pi 4B или Raspberry Pi 400, большинство шагов установки, описанных в этом руководстве по началу работы, можно пропустить, запустив сценарий установки.

 Примечание

Для этого сценария установки требуется примерно 2,5 ГБ дискового пространства на вашей SD-карте, поэтому перед его запуском убедитесь, что у вас достаточно свободного места. Вы можете проверить, сколько у вас свободного места на диске, с помощью команды df -h.

Вы можете получить этот скрипт, выполнив следующую команду в терминале:

```
$ wget https://raw.githubusercontent.com/raspberrypi/pico-setup/master/pico_setup.sh ①
```

1. Сначала вам следует sudo apt установить wget, если у вас еще не установлен wget.

затем создайте исполняемый файл скрипта с помощью,

```
$ chmod +x pico_setup.sh
```

И запустите его с помощью

```
$ ./pico_setup.sh\
```

Скрипт выполнит:

- Создание каталога с именем pico
- Установку необходимых зависимостей
- Загрузите репозитории pico-sdk, pico-examples, pico-extras и pico-playground
- Определите PICO_SDK_PATH, PICO_EXAMPLES_PATH, PICO_EXTRAS_PATH и PICO_PLAYGROUND_PATH в вашем ~/.bashrc
- Создайте примеры blink и hello_world в pico-examples/build/blink и pico-examples/build/hello_world
- Загрузите и создайте picotool (см. приложение B) и скопируйте его в /usr/local/bin.
- Загрузите и скомпилируйте picoprobe (см. приложение A).
- Загрузите и скомпилируйте OpenOCD (для поддержки отладки)
- Загрузите и установите Visual Studio Code
- Установите необходимые расширения кода Visual Studio (более подробную информацию смотрите в главе 7)
- Настройте Raspberry Pi UART для использования с Raspberry Pi Pico

 ПРИМЕЧАНИЕ
Каталог pico будет создан в папке, в которой вы запустите скрипт pico_setup.sh.

Как только он запустится, вам нужно будет перезагрузить Raspberry Pi,

```
$ sudo reboot
```

чтобы реконфигурация UART вступила в силу. Как только ваш Raspberry Pi перезагрузится, вы можете открыть Visual Studio Code в меню "Программирование" и следовать инструкциям из раздела 7.2.

Глава 2. Пакет SDK

 ВАЖНО

В следующих инструкциях предполагается, что вы используете Raspberry Pi Pico, и некоторые детали могут отличаться, если вы используете другую плату на базе RP2040. Они также предполагают, что вы используете Raspberry Pi OS, работающую на Raspberry Pi 4, или эквивалентный дистрибутив Linux на базе Debian, работающий на другой платформе. Также приведены альтернативные инструкции для тех, кто использует Microsoft Windows (см. раздел 9.2) или Apple macOS (см. раздел 9.1).

Raspberry Pi Pico построен на базе микроконтроллера RP2040, разработанного Raspberry Pi. Разработка на плате полностью поддерживается как C / C++ SDK, так и официальным портом MicroPython. В этой книге рассказывается о том, как начать работу с SDK, и рассказывается о том, как создавать, устанавливать набор инструментов SDK и работать с ним.

 Совет

Для получения дополнительной информации об официальном порте MicroPython смотрите книгу Raspberry Pi Pico Python SDK, в которой описан порт, и "Начало работы с MicroPython на Raspberry Pi Pico" Гарета Халфакри и Бена Эверарда, опубликованную издательством Пресс для Raspberry Pi.

 СОВЕТ

Дополнительные сведения о C/C++ SDK, а также документацию на уровне API смотрите в книге Raspberry Pi Pico C/C++ SDK

.

2.1. Получите SDK и примеры

Репозиторий примеров Pico предоставляет набор примеров приложений, написанных с использованием SDK. Чтобы клонировать эти репозитории, начните с создания каталога pico, в котором будут храниться все проверки, связанные с pico. Эти инструкции создают каталог pico в /home/pi/pico.

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

Затем клонируем репозитории git pico-sdk и pico-examples.

```
$ git clone https://github.com/raspberrypi/pico-sdk.git --мастер ветви
$ cd pico-sdk
$ обновление подмодуля git –инициализация
$ cd ..
$ git clone https://github.com/raspberrypi/pico-examples.git --мастер ветки
```

.

⍰ Предупреждения

Неспособность выполнить приведенную выше команду git submodule update --init будет означать, что модуль tinyusb не будет включен, и в результате функциональность USB не будет скомпилирована в SDK. Это означает, что последовательный порт USB, другие функции USB и пример кода работать не будут.

⍰ ПРИМЕЧАНИЕ

Есть дополнительные репозитории: Pico Extras и Pico Playground, которые также могут вас заинтересовать.

2.2. Установите набор инструментов

Для создания приложений в pico-examples, вам потребуется установить некоторые дополнительные инструменты. Для создания проектов вам понадобится CMake, а кроссплатформенный инструмент, используемый для создания программного обеспечения, и GNU Embedded Toolchain для Arm. Вы можете установить оба из них через apt из командной строки. Все, что вы уже установили, будет проигнорировано apt.
Обновление

```
$ sudo apt
$ sudo apt установить cmake gcc-arm-none-eabi libnewlib-arm-none-eabi build-essential ①
```

1. Для компиляции pioasm и elf2uf2 необходимы собственные gcc и g++.

⍰ ПРИМЕЧАНИЕ
Пользователям Ubuntu и Debian может дополнительно потребоваться также установить libstdc++-arm-none-eabi-newlib.

2.3. Обновление SDK

Когда будет выпущена новая версия SDK, вам нужно будет обновить свою копию SDK. Чтобы сделать это, перейдите в каталог pico-sdk , который содержит вашу копию SDK, и выполните следующие действия;

```
$ cd pico-sdk
$ git pull
$ git  обновление подмодуля
```

⍰ ПРИМЕЧАНИЕ
Если вы хотите получать информацию о новых выпусках, вы можете получать уведомления, настроив пользовательские часы в репозитории pico-sdk. Перейдите к https://github.com/raspberrypi/pico-sdk и затем выберите Просмотр → Пользовательские → Выпуски. Вы будете получать уведомление по электронной почте каждый раз, когда будет выпущен новый SDK.

Глава 3. Мигающий светодиод в C

Когда вы пишете программное обеспечение для аппаратного обеспечения, включение, выключение и затем повторное включение светодиода, как правило, является первой программой, которая запускается в новой среде программирования. Научившись мигать светодиодом, вы окажетесь на полпути к чему угодно. Мы собираемся пойти дальше и мигнуть встроенным светодиодом на Raspberry Pi Pico, который подключен к контакту 25 RP2040.

Примеры Pico: https://github.com/raspberrypi/pico-examples/blob/master/blink/blink.c Строки 9 - 23

```
9 int main() {
10 #ifndef PICO_DEFAULT_LED_PIN
11 #warning blink example requires a board with a regular LED
12 #else
13 const uint LED_PIN = PICO_DEFAULT_LED_PIN;
14 gpio_init(LED_PIN);
15 gpio_set_dir(LED_PIN, GPIO_OUT);
16 while (true) {
17 gpio_put(LED_PIN, 1);
18 sleep_ms(250);
19 gpio_put(LED_PIN, 0);
20 sleep_ms(250);
21 }
22 #endif
23 }
24
```

3.1. Построение "Мигания"

Из каталога pico, который мы создали ранее, перейдите в pico-examples и создайте каталог сборки.

```
$ cd pico-examples
$ mkdir build
$ cd build
```

.

Затем, предполагая, что вы клонировали репозитории pico-sdk и pico-examples в один и тот же каталог бок о бок, установите PICO_SDK_PATH:

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

 СОВЕТ

На протяжении всей этой книги мы используем относительный путь ../../pico-sdk для извлечения SDK для PICO_SDK_PATH. Однако, в зависимости от местоположения вашей проверки, возможно, имеет смысл заменить это абсолютным путем, например /home/pi/pico/pico-sdk.

Подготовьте каталог сборки cmake, запустив cmake ..

```
$ cmake ..
Используя PICO_SDK_PATH из среды ('../../pico-sdk')
PICO_SDK_PATH - это /home/pi/pico/pico-sd
-- Файлы сборки были записаны по адресу: /home/pi/pico/pico-examples/build
```

ПРИМЕЧАНИЕ

. cmake по умолчанию Release сборку выпуска с включенной оптимизацией компилятора и удаленной отладочной информацией. Чтобы создать отладочную версию, запустите cmake -DCMAKE_BUILD_TYPE=Debug ... Мы рассмотрим это позже в разделе 6.1.

ВАЖНО

SDK по умолчанию создаст двоичный файл, ориентированный на Raspberry Pi Pico. Если вы создаете двоичный файл для другой платы, вам нужно передать -DPICO_BOARD=board, где board - это название платы. Например, если вы создаете двоичный файл для Pico W, который вы должны передать, -DPICO_BOARD=pico_w вместе с информацией о вашей беспроводной сети -DWIFI_SSID="Ваша сеть" -DWIFI_PASSWORD="Ваш пароль".

Теперь CMake подготовил область сборки для дерева пико-примеров. Отсюда можно ввести make для создания всех примеров приложений. Однако, поскольку мы создаем blink, мы пока создадим это приложение, только изменив каталог на каталог blink, прежде чем вводить make.

СОВЕТ

Вызов make с помощью -j4 приведет к параллельному запуску четырех заданий make для ускорения процесса. Raspberry Pi 4 имеет 4 ядра, так что j4 - разумное число.

```
$ cd blink
$ make -j4
Сканирование зависимостей целевого ELF2UF2Build
Проверка зависимостей целевого boot_stage2_original
[ 0%] Создание каталогов для 'ELF2UF2Build'
.
.
.
[100%] Связывание исполняемого файла CXX blink.elf
[100%] Встроенный целевой blink
```

Среди других целей мы теперь создали:

- blink.elf, который используется отладчиком
- blink.uf2, который можно перетащить на USB-накопитель RP2040

Этот двоичный файл будет мигать встроенным светодиодом Raspberry Pi Pico, который подключен к GPIO25 RP2040.

Более подробно о примере кода?

В этом документе показано, как создать программное обеспечение и загрузить его на ваш Raspberry Pi Pico. Многое делается за кулисами, чтобы превратить наше приложение blink в двоичную программу, а Raspberry Pi Pico C/C++

SDK book отодвигает занавес и показывает некоторые задействованные механизмы. Если вас еще не беспокоят подобные вещи, читайте дальше!

3.2. Загрузите и запустите "Blink"

Самый быстрый способ впервые загрузить программное обеспечение на плату на базе RP2040 - это установить ее в виде USB-накопителя
Устройство хранения. Выполнение этого действия позволяет перетащить файл на плату для программирования флэш-памяти. Продолжайте и подключите Raspberry Pi Pico к вашему Raspberry Pi с помощью кабеля micro-USB, убедившись, что вы удерживаете нажатой кнопку загрузки

(Рис. 1) при этом принудительно переведите его в режим USB-накопителя.

3.2.1. С рабочего стола

Если вы используете Raspberry Pi Desktop, Raspberry Pi Pico должен автоматически подключаться как USB-накопитель Устройство. Отсюда вы можете перетащить файл blink.uf2 на устройство массового хранения.
RP2040 перезагрузится, размонтирует себя как устройство массового хранения данных и начнет запускать прошитый код, см. рис. 1.

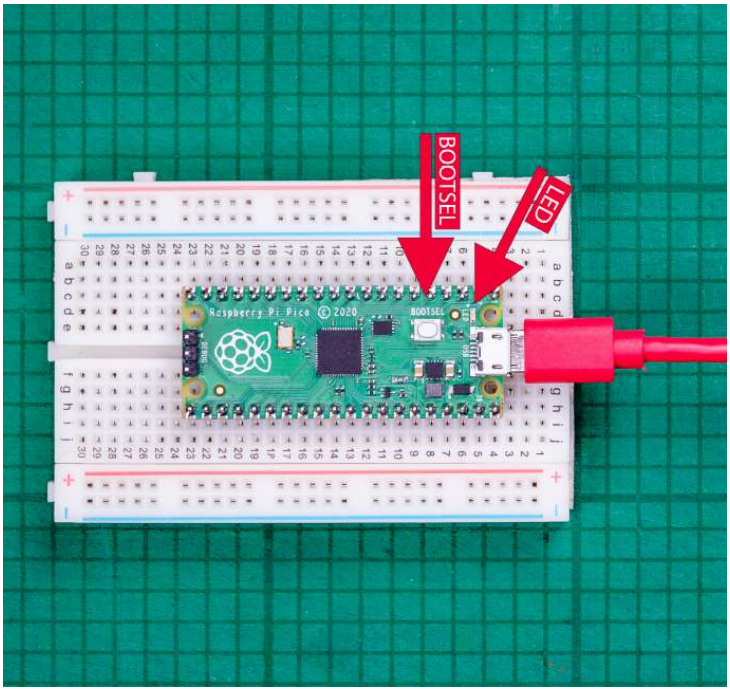


Рисунок 1. Мигает встроенный светодиод на Raspberry Pi Pico.
Стрелки указывают на встроенный Светодиод и кнопка загрузки.

3.2.2. Использование командной строки

СОВЕТ
Вы можете использовать picotool для загрузки двоичного файла UF2 на свой Raspberry Pi Pico, смотрите приложение B.
Например, если вы вошли в систему через ssh, вам, возможно, придется подключить устройство массового хранения вручную:

```
$ dmesg | tail
[ 371.973555] sd 0:0:0:0:0: [sda] Подключенный съемный диск SCSI
$ sudo mkdir -p /mnt/pico
$ sudo mount /dev/sda1 /mnt/pico
```

Если вы видите файлы в /mnt/pico, значит, USB-накопитель подключен правильно:

```
$ ls /mnt/пико/
```

индекс.HTM INFO_UF2.TXT

Скопируйте ваш файл blink.uf2 в RP2040:

```
$ sudo cp blink.uf2 /mnt/pico
$ sudo sinc
```

RP2040 уже отключился как USB-накопитель и запускает ваш код, но для аккуратности отключите

```
/mnt/pico
$ sudo umount /mnt/pico
```

⍰ ПРИМЕЧАНИЕ

Отключение питания от платы не приводит к удалению кода. Когда плата будет снова подключена к источнику питания, код, который вы только что загрузили, начнет выполняться снова. Если вы хотите загрузить новый код на плату (и перезаписать все , что там уже было), нажмите и удерживайте кнопку ЗАГРУЗКИ при включении питания, чтобы перевести плату в режим массового хранения.

3.2.3. Отвлечение: Другие платы

Если вы не следуете этим инструкциям на Raspberry Pi Pico, у вас может отсутствовать кнопка загрузки (как указано в Рисунок 1). На вашей плате может быть какой-то другой способ загрузки кода, который поставщик платы должен был задокументировать:

- Большинство плат имеют интерфейс SWD (глава 5), который позволяет сбросить плату и загрузить код без каких
-либо нажатий кнопок
- Возможно, существует какой-то другой способ вытащить pin-код flash CS (именно так работает кнопка загрузки на
Raspberry Pi Pico), например, пара соединительных контактов, которые должны быть закорочены вместе
- На некоторых платах будет кнопка сброса, но не будет кнопки загрузки, и они могут содержать некоторый код во flash для обнаружения двойного нажатия кнопки сброса и входа в загрузчик таким образом.

Во всех случаях вам следует ознакомиться с документацией к конкретной используемой вами плате, в которой должен быть описан наилучший способ загрузки встроенного ПО на эту плату.

3.2.4. В стороне: Программирование флэш-памяти без помощи рук

Чтобы войти в режим загрузки на вашем Raspberry Pi Pico и загрузить код через USB, вам нужно удерживать кнопку загрузки нажатой, а затем каким-либо образом перезагрузить плату. Вы можете сделать это, отсоединив и подключив USB-разъем или добавив внешняя кнопка для заземления штифта УПРАВЛЕНИЯ.

Вы также можете использовать порт SWD (глава 5) для сброса платы, загрузки кода и запуска процессоров, и это работает, даже если ваш код вышел из строя, поэтому нет необходимости вручную перезагружать плату или нажимать какие-либо кнопки. После того как вы все настроили с помощью building programs и попробовали пример Hello World в следующей главе (глава 4), хорошим следующим шагом будет настройка SWD

.

Если вы используете Raspberry Pi, вы можете настроить SWD, запустив сценарий pico-setup (глава 1) и подключив 3 провода от вашего Pi к Pico, как показано в главе 5. Также можно использовать отладочный зонд USB to SWD, например, в приложении A показано, как можно использовать один Pico для доступа к SWD-порту второго Pico через USB-порт первого Pico.

Глава 4. Произнесение "Привет, мир" на языке Си

После включения и выключения светодиода следующее, что захочет сделать большинство разработчиков, - это создать и использовать последовательный порт и сказать "Привет, мир".
Примеры Pico: https://github.com/raspberrypi/pico-examples/blob/master/hello_world/serial/hello_serial.c Строки 1

```
0 - 16
10 int main() {
11 stdio_init_all();
12 while (истина) {
13 printf("Привет, мир!\n");
14 sleep_ms(1000);
15 }
16 }
```

4.1. Последовательный ввод и вывод на Raspberry Pi Pico

Последовательный вход (stdin) и выход (stdout) могут быть направлены либо на последовательный UART, либо на USB CDC (USB serial). Однако с помощью стандартные stdio и printf будут нацелены на стандартный Raspberry Pi Pico UART0.

UART0 по умолчанию	Физический вывод	вывод GPIO
GND	3	N/A
UART0_TX	2	GP0
UART0_RX	1	GP1

⍰ ВАЖНО
Выводом Raspberry Pi Pico UART TX по умолчанию (выходящим из Raspberry Pi Pico) является вывод GP0, а выводом UART RX (входящим в Raspberry Pi Pico) - это вывод GP1. Контакты UART по умолчанию настраиваются для каждой платы с использованием файлов конфигурации платы. Конфигурацию Raspberry Pi Pico можно найти в https://github.com/raspberrypi/pico-sdk/blob/master/
src/boards/include/платы/pico.h. В SDK по умолчанию используется название платы Raspberry Pi Pico, если другая плата не указана.

SDK использует CMake для управления своей системой сборки, смотрите главу 8, в которой используется библиотека интерфейса pico_stdlib для объединения необходимых исходных файлов для предоставления возможностей.
Примеры Pico: https://github.com/raspberrypi/pico-examples/blob/master/hello_world/serial/CMakeLists.txt

```
1 add_executable(hello_serial
2 hello_serial.c
3 )
4
5 # подключение общих зависимостей
6 целевых ссылок на библиотеки(hello_serial pico_stdlib)
7
8 # создайте файл map/bin/hex/uf2 и т.д.
9 pico_add_extra_outputs(привет_сериал)
10
11 # добавить URL-адрес через pico_set_program_url
12 example_auto_set_url(hello_serial)
```

Начало работы с Raspberry Pi Pico

Назначение для стандартного вывода может быть изменено с помощью директив CMake, с выводом, направленным на UART или USB CDC, или на оба,

```
pico_enable_stdio_usb(hello_world 1) ①
pico_enable_stdio_uart(hello_world 0) ②
```

1. Включитmь вывод printf через USB CDC (USB serial)

2. Отключить вывод printf через UART

Это означает, что без изменения исходного кода C вы можете изменить назначение stdio с UART на USB.
Примеры Pico: https://github.com/raspberrypi/pico-examples/blob/master/hello_world/usb/CMakeLists.txt

```
1 if (ЦЕЛЕВОЕ устройство tinyusb_device)
2 add_executable(hello_usb
3 привет_usb.c
4 )
5
6 # подключение общих зависимостей
7 целевых ссылок на библиотеки(hello_usb pico_stdlib)
8
9 # включить usb-выход, отключить uart-выход
10 pico_enable_stdio_usb(hello_usb 1)
11 pico_enable_stdio_uart(hello_usb 0)
12
13 # создайте файл map/bin/hex/uf2 и т.д.
14 pico_add_extra_outputs(hello_usb)
15
16 # добавить URL-адрес с помощью pico_set_program_url
17 example_auto_set_url(hello_usb)
18 elseif(PICO_ON_DEVICE)
Сообщение 19(ПРЕДУПРЕЖДЕНИЕ "не удается создать hello_usb, поскольку подмодуль TinyUSB не инициализирован в SDK")
20 endif()
```

4.2. Создайте "Hello World"

Как мы делали для предыдущего примера "Blink", измените каталог на каталог hello_world внутри дерева pico-examples/build и запустите make.

```
$ cd hello_world
$ make -j4
Сканирование зависимостей целевого ELF2UF2Build
[ 0%] Создание каталогов для 'ELF2UF2Build'
.
[ 33%] Связывание исполняемого файла CXX с hello_usb.elf
[ 33%] Встроенный целевой hello_usb
.
[100%] Связывание исполняемого файла CXX с hello_serial.elf
[100%] Встроенный целевой hello_serial
```

Это позволит создать два отдельных примера программ в каталогах hello_world/serial/ и hello_world/usb/.

Среди других целей мы теперь создали:

- serial/hello_serial.elf, который используется отладчиком
- serial/hello_serial.uf2, который можно перетащить на USB-накопитель RP2040 (последовательный двоичный файл UART)
- usb/hello_usb.elf, который используется отладчиком
- usb/hello_usb.uf2, который можно перетащить на USB-накопитель RP2040 (последовательный двоичный файл USB)

Где hello_serial направляет stdio в UART0 на выводах GP0 и GP1, а hello_usb направляет stdio в USB CDC serial.

⍰ Предупреждения

Если вы не инициализировали подмодуль tinyusb при оформлении заказа pico-sdk, то пример последовательного подключения USB CDC не будет работайте, поскольку SDK не будет содержать функций USB.

4.3. Прошейте и запустите "Hello World"

Подключите Raspberry Pi Pico к вашему Raspberry Pi с помощью кабеля micro-USB, обязательно удерживая нажатой кнопку загрузки, чтобы перевести его в режим USB-накопителя. Как только он подключен, отпустите кнопку загрузки, и если вы используете Raspberry Pi Desktop, он должен автоматически подключиться как USB-накопитель. Отсюда вы можете Перетащите файл hello_serial.uf2 или hello_usb.uf2 на устройство массового хранения.

RP2040 перезагрузится, отключив себя как устройство массового хранения, и начнет запускать прошитый код.

Однако, хотя пример "Hello World" сейчас запущен, мы пока не можем увидеть текст. Нам нужно подключить наш главный компьютер к соответствующему интерфейсу stdio на Raspberry Pi Pico, чтобы увидеть выходные данные.

4.4. Смотрите USB-выход "Hello World"

Если вы перетащили двоичный файл hello_usb.uf2, то текст "Hello World" будет перенаправлен на USB serial.

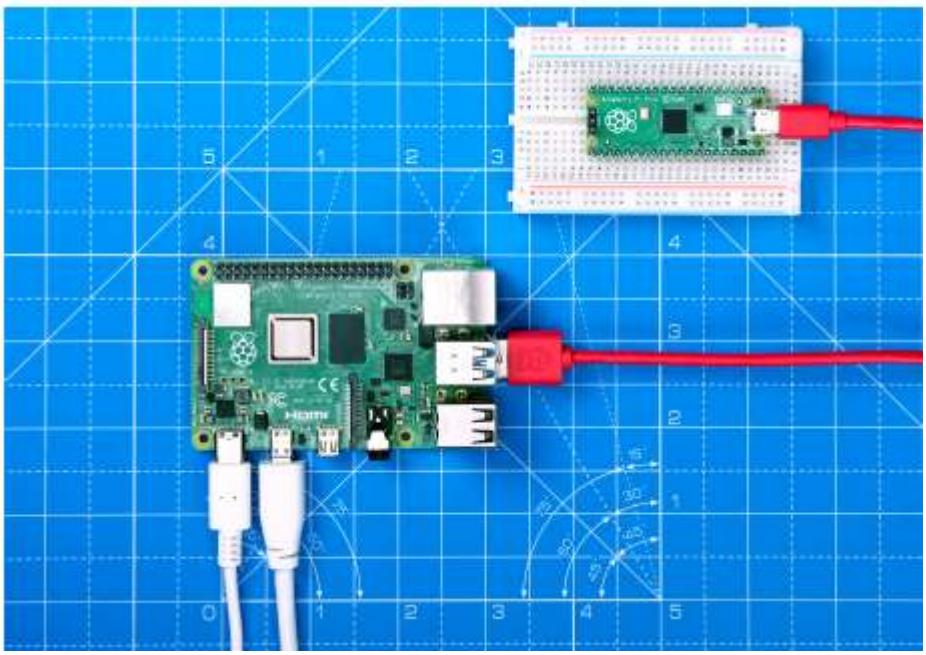


Рисунок 2. Подключение Raspberry Pi к Raspberry Pi Pico через USB.

Если Raspberry Pi Pico подключен непосредственно к Raspberry Pi через USB, см. рис. 2, вы можете просмотреть текст, установив minicom:

```
$ sudo apt установите minicom
```

и откройте последовательный порт:

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

Вы должны увидеть "Привет, мир!" напечатано на консоли.

СОВЕТ

Чтобы выйти из minicom, нажмите CTRL-A, а затем X.

ПРИМЕЧАНИЕ

Если вы собираетесь использовать SWD для отладки (см. главу 6), вам необходимо использовать последовательное соединение на основе UART, поскольку стек USB будет приостановлен, когда ядра RP2040 будут остановлены во время отладки, что приведет к любым подключенным USB-устройства для отключения.

4.5. Смотрите вывод UART "Hello World"

В качестве альтернативы, если вы перетащили двоичный файл hello_serial.uf2, то текст "Hello World" будет направлен в UART0 на выводах GP0 и GP1. Первое, что вам нужно будет сделать, чтобы увидеть текст, - это включить последовательную связь UART на хосте Raspberry Pi. Для этого запустите raspi-config,

```
$ sudo raspi-config
```

и перейдите к параметрам интерфейса → Serial и выберите "Нет", когда вас спросят "Хотите ли вы, чтобы оболочка входа была доступна через serial?" и "Да", когда вас спросят "Хотите ли вы, чтобы аппаратное обеспечение последовательного порта было включено?" Вы должны увидеть что-то вроде Рисунок 3.

Рисунок 3. Включение последовательного UART с помощью raspi-конфигурация на Raspberry Pi.

Выходя из raspi-config, вы должны выбрать "Да" и перезагрузить Raspberry Pi, чтобы включить последовательный порт. Затем вы должны соединить Raspberry Pi и Raspberry Pi Pico вместе со следующим отображением:

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

See Figure 4.

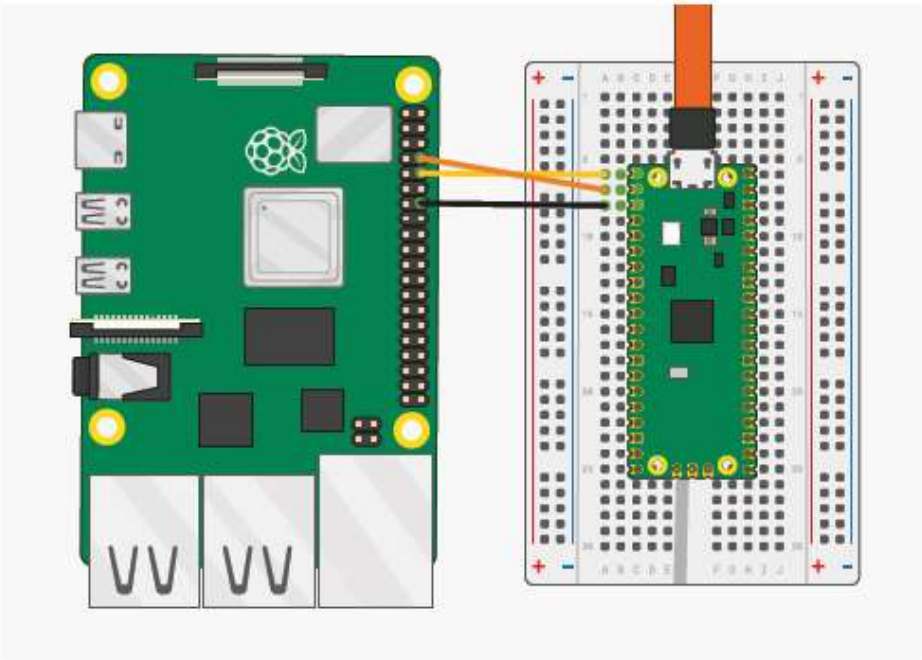


Рисунок 4. *Raspberry* Pi 4 и *Raspberry* Pico с UART0 , соединенные вместе.

Как только две платы будут соединены проводами, если вы еще этого не сделали, вам следует установить minicom:

```
$ sudo apt установите minicom
```

и откройте последовательный порт:

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

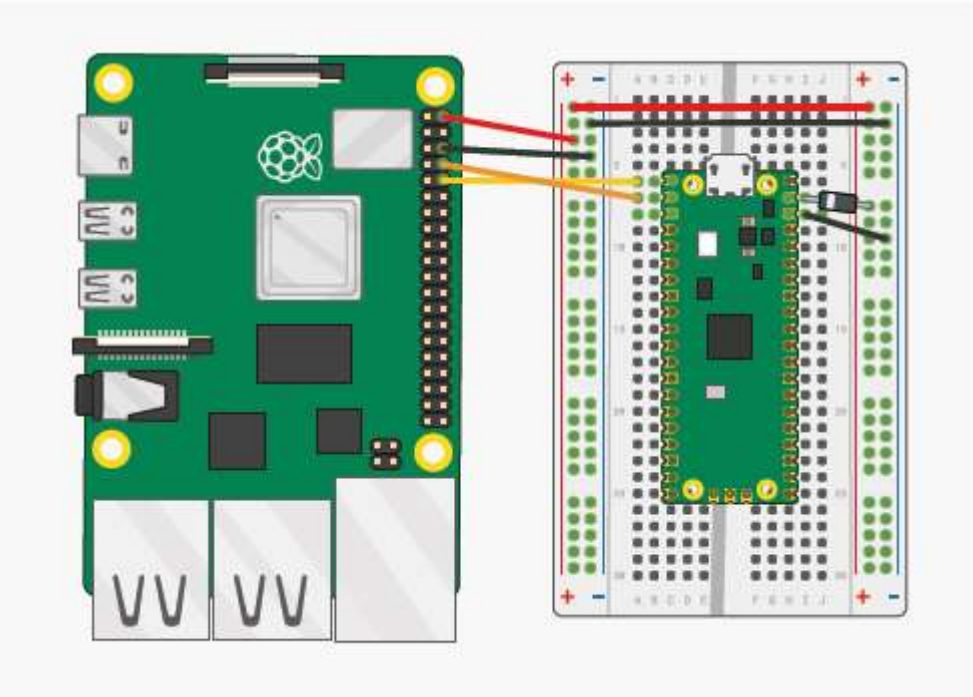
Ты должен увидеть Hello, world! выводится на консоль.

СОВЕТ

Чтобы выйти из minicom, нажмите CTRL-A, а затем X.

4.6. Включение питания платы

Вы можете отсоединить Raspberry Pi Pico от USB и включить питание платы, дополнительно подключив 5-вольтовый вывод Raspberry Pi к Рис. 5. Raspberry Pi и Raspberry Pi Pico подключаются только с помощью выводов GPIO.



Хотя можно подключить вывод Raspberry Pi 5V к выводу Raspberry Pi Pico VBUS, это не рекомендуется.
Короткое замыкание проводов 5 В вместе будет означать, что Micro USB не может быть использован. Исключение составляет использование Raspberry Pi Pico в режиме USB-хоста, в этом случае к выводу VBUS должно быть подключено 5 В.
Вывод 3,3 В является выходным выводом Raspberry Pi Pico, вы не можете запитать Raspberry Pi Pico через этот вывод, и он не должен быть подключен к источнику питания.
Смотрите раздел "Питание" в разделе "Проектирование оборудования с RP2040" для получения дополнительной информации о питании Raspberry Pi Pico.

Глава 5. Программирование на Flash с помощью SWD

Serial Wire Debug (SWD) - это стандартный интерфейс микроконтроллеров на базе Cortex-M, который машина, которую вы используете для разработки вашего кода (обычно называемая хостом), может использовать для сброса платы, загрузки кода во flash и запуска кода. Raspberry Pi Pico предоставляет интерфейс RP2040 SWD на трех выводах у нижнего края платы. Хост может использовать порт SWD для доступа к внутренним устройствам RP2040 в любое время, поэтому нет необходимости вручную перезагружать плату или удерживать кнопку загрузки.

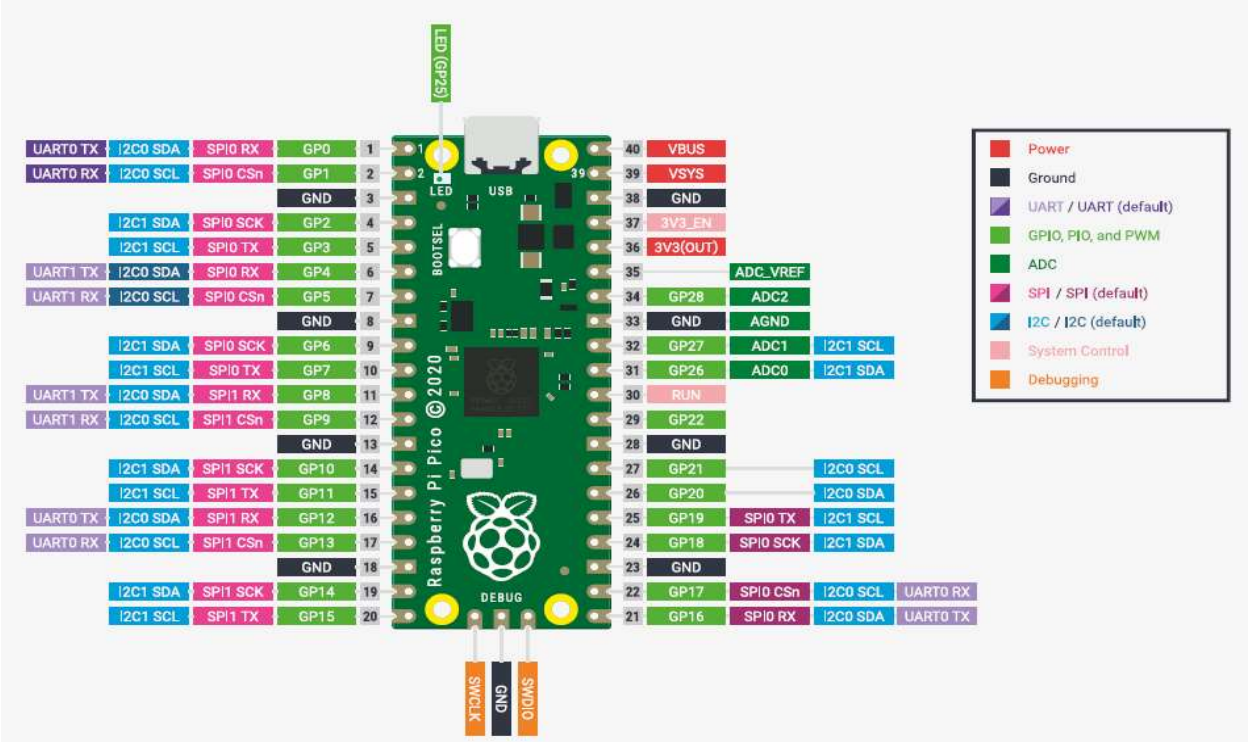


Рисунок 6. Порт SWD обозначен в нижней части этого пико схема распиновки. Соединение заземления (GND) требуется для поддержания хорошей целостности сигнала между хостом и Pico. Вывод SWDIO передает отладочный трафик в обоих направлениях, между RP2040 и хостом. Вывод SWCLK обеспечивает хорошую синхронизацию соединения. Эти контакты подключаются к Выделенному интерфейсу SWD на RP2040, поэтому вам не нужно жертвовать никакими GPIO для использования порта SWD.

На Raspberry Pi вы можете подключить Pi GPIO непосредственно к SWD-порту Pico и загрузить код оттуда. На других машины вам понадобится дополнительная аппаратная часть — отладочный зонд — для подключения вашего хост-компьютера (например, USB-порта) к выводам SWD на Pico. Один из самых дешевых способов сделать это - использовать другой Pico в качестве отладочного зонда, и это описано в приложении A.

В этой главе рассказывается о том, как вы можете подключить свой компьютер к SWD-порту Raspberry Pi Pico и использовать его для записи программ во flash и их запуска.

? СОВЕТ

Если вы используете среду IDE, подобную Visual Studio Code (глава 7), ее можно настроить на автоматическое использование SWD за сцены, поэтому вы нажимаете кнопку воспроизведения, и код запускается, как если бы вы запускали машинный код на своем компьютере.

? ПРИМЕЧАНИЕ

Вы также можете использовать SWD для интерактивных методов отладки, таких как установка точек останова, пошаговое выполнение кода построчно или даже просмотр регистров ввода-вывода непосредственно с вашего компьютера без записи каких-либо данных. Программное обеспечение RP2040. Это описано в главе 6.

5.1. Установка OpenOCD

Чтобы получить доступ к SWD-порту микроконтроллера, вам нужна программа на вашем хост-компьютере, называемая отладочным транслятором, который понимает протокол SWD и знает, как управлять процессором (два Cortex-M0+s в случае RP2040) внутри микроконтроллера. Отладочный транслятор также знает, как взаимодействовать с конкретным отладочным зондом, который вы подключили к порту SWD, и как запрограммировать flash на вашем устройстве.

В этом разделе описывается установка отладочного транслятора под названием OpenOCD.

СОВЕТ

Если вы запустили скрипт pico-setup на своем Raspberry Pi (глава 1), OpenOCD уже установлен, и вы можете перейти к следующему разделу.

ПРИМЕЧАНИЕ

В этих инструкциях предполагается, что вы хотите создать openocd в /home/pi/pico/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 --branch rp2040 --recursive --
depth=1
$ cd openocd
$ ./bootstrap
$ ./configure --enable-ftdi --enable-sysfsgpio --enable-bcm2835gpio
$ make -j4
$ sudo make install
```

Теперь OpenOCD должен быть установлен, и вы можете запустить его как openocd со своего терминала.

ПРИМЕЧАНИЕ

На macOS вам, возможно, придется установить более новую версию texinfo с помощью Homebrew.

5.2. Подключение SWD-порта

Вам необходимо подключить провода к порту SWD, чтобы запрограммировать и запустить код на RP2040 через SWD.

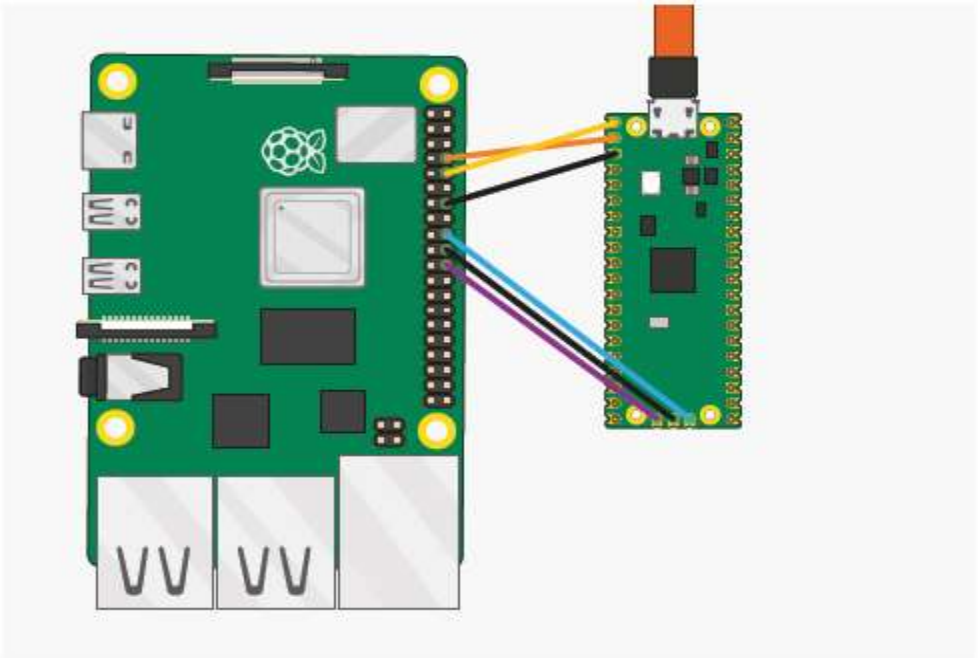


Рисунок 7. Raspberry Pi 4 и Raspberry Pi Pico с подключенными друг к другу портами UART и SWD . Оба они подключаются непосредственно к Raspberry Pi 4 без использования макетной платы.

Для доступа к SWD необходимы только три нижних провода на этой схеме; при желании вы также можете подключить Подключите UART, как показано на 3 верхних проводах, для прямого доступа к последовательному порту Pico. Конфигурация по умолчанию — SWDIO на Pi GPIO 24 и SWCLK на GPIO 25 - и может быть подключена к Raspberry Pi

Raspberry Pi	Raspberry Pi Pico
GND(Pin20)	SWD GND
GPIO24(Pin18)	SWDIO
GPIO(Pin22)	SWCLK

как показано на рисунке 7.

СОВЕТ

Если вы используете другой отладочный зонд, например Picoprobe (приложение A), вам необходимо подключить выводы GND, SWCLK и SWDIO на вашем зонде к соответствующим выводам на вашей Raspberry Pi Pico или другой плате на базе RP2040.

Если возможно, вам следует подключить SWD-порт непосредственно к Raspberry Pi, поскольку важна целостность сигнала; подключение SWD-порта через макетную плату или другие косвенные методы может значительно снизить целостность сигнала, так что загрузка кода через Raspberry Pi может привести к потере целостности сигнала, нестабильности соединения или полному выходу из строя. Важно также подключить провод заземления (0 В) непосредственно между ними и не полагаться на другой путь заземления.

Обратите внимание, что для его отладки Raspberry Pi Pico также должен быть подключен к питанию (например, через USB)! Вы должны создать нашу ветку OpenOCD, чтобы получить работающую поддержку multidrop SWD.

5.3. Загрузка программы

OpenOCD ожидает, что двоичные файлы программы будут в виде файлов .elf (исполняемый формат с возможностью связывания), а не файлов .uf2, используемых в режиме загрузки. SDK создает оба типа файлов по умолчанию, но важно не смешивать их. Предполагая, что вы уже создали пример blink, используя инструкции из главы 3, вы можете выполнить следующую команду, чтобы запрограммировать полученный файл .elf поверх SWD и запустить его:

```
$ openocd -f interface/raspberrypi-swd.cfg -f target/rp2040.cfg -c "program
blink/blink.elfverify reset exit"
```

сброс Приводит RP2040 в чистое исходное состояние, как если бы он только что включился, так что он готов к запуску нашего кода. завершите отключение от RP2040 и завершите работу. Наш недавно запрограммированный код начнет выполняться, как только OpenOCD отключится.

-f interface/raspberrypi-swd.cfg	r OpenOCD использовать GPIO-контакты Raspberry Pi для доступа к SWD-порт ы использовали внешний USB→ SWD-зонд, такой как Picoprobe в приложении A, указали здесь другой интерфейс
-f target/rp2040.cfg	r OpenOCD, что мы подключаемся к плате на базе RP2040. Этот файл .cfg информацию для OpenOCD, такую как тип процессора (Cortex-M0+) доступа к флэш-памяти.
-c	мент используется для передачи серии команд в OpenOCD непосредственно дной строки. OpenOCD также имеет интерактивный интерфейс терминала й мы могли бы вместо этого вводить команды. Команды, которые мы используе ие:
program blink/blink.elf	OpenOCD, чтобы он записал наш файл .elf во flash, при необходимос тельно удалив целевую область flash. Файл .elf содержит всю информаци щую OpenOCD, куда должны быть загружены различные его части и наскольк и части сообщает OpenOCD, чтобы он записал наш файл .elf во flash, ходимости предварительно удалив целевую область flash. Файл .elf содержит во цию, указывающую OpenOCD, куда должны быть загружены различные его час ько велики эти части
verify	Попросите OpenOCD выполнить повторное считывание с флэш-памяти пос программирования, чтобы убедиться, что программирование прошло успешно

reset	Переведите RP2040 в чистое исходное состояние, как если бы он только ч включился, чтобы он был готов к запуску нашего кода. .
exit	завершите отключение от RP2040 и завершите работу. Наш запрограммированный код начнет выполняться, как только OpenOC отключится

Совет

Если вы видите ошибку типа Info: DAP init failed, значит, OpenOCD не смог увидеть RP2040 в используемом им SWD-интерфейсе. Наиболее распространенными причинами являются неправильное питание вашей платы, например, через USB-кабель; неправильная проводка SWD (например, провод заземления не подключен, или SWDIO и SWCLK были заменены местами); или что есть какие-либо неисправности. проблема с целостностью сигнала, вызванная длинными или незакрепленными соединительными проводами.

Чтобы проверить, что вы действительно загрузили новую программу, вы можете изменить blink/blink.c, чтобы светодиод мигал быстрее, а затем перестроить и повторно запустить приведенную выше команду openocd:

```
int main() {
const uint LED_PIN = 25;
gpio_init(LED_PIN);
gpio_set_dir(LED_PIN, GPIO_OUT);
while (true) {
gpio_put(LED_PIN, 1);
// Blink faster! (this is the only line that's modified)
sleep_ms(25);
gpio_put(LED_PIN, 0);
sleep_ms(250);
}
}
```

И затем,

```
$ cd pico-examples/build
$ make blink
# (the application is rebuilt)
$ openocd -f interface/raspberrypi-swd.cfg -f target/rp2040.cfg -c "program
blink/blink.elf
verify reset exit"
```

Глава 6. Отладка с помощью SWD

Помимо сброса платы, загрузки и запуска кода, порт SWD на платах на базе RP2040, таких как Raspberry Pi Pico, можно использовать для интерактивной отладки загруженной вами программы. Это включает в себя такие вещи, как:

- Установка точек останова в вашем коде
- Пошаговое выполнение строка за строкой
- Проверка значений переменных в разных точках программы

В главе 5 показано, как установить OpenOCD для доступа к SWD-порту на вашем Raspberry Pi Pico. Для интерактивной отладки кода нам также нужен отладчик, такой как вездесущий GNU Debugger, GDB.

Обратите внимание, что по умолчанию SDK создает высокооптимизированные двоичные файлы программ, которые могут сильно отличаться с точки зрения потока управления и данных от исходной программы, которую вы написали. Это может сбить с толку, когда вы пытаетесь пошагово просмотреть код в интерактивном режиме, поэтому часто бывает полезно создать отладочную сборку вашей программы, которая оптимизирована менее агрессивно, чтобы реальный поток управления на устройстве более точно соответствовал вашему исходному коду.

6.1. Создайте отладочную версию "Hello World"

⍰ ПРЕДУПРЕЖДЕНИЕ

При использовании SWD для отладки вам необходимо использовать последовательное соединение на основе UART (см. главу 4) в качестве USB-стека будет приостановлено, когда ядра RP2040 будут остановлены во время отладки, что приведет к отключению всех подключенных USB-устройств. Вы не можете использовать последовательное соединение USB CDC во время отладки.

Вы можете создать отладочную версию "Hello World" с помощью CMAKE_BUILD_TYPE=Debug, как показано ниже,

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

6.2. Установка GDB

Установите gdb-multiarch,

```
$ sudo apt install gdb-multiarch
```

Примечание
На macOS вы можете установить gdb с помощью Homebrew,
$ brew install gdb
и, несмотря на предупреждение после установки, вам не нужно кодировать двоичный файл, чтобы использовать его для удаленной отладки.

6.3. Используйте GDB и OpenOCD для отладки Hello World

Убедившись, что ваши Raspberry Pi 4 и Raspberry Pi Pico правильно подключены друг к другу, мы можем подключить OpenOCD к чипу через интерфейс raspberrypi-swd.

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

Ваш вывод должен выглядеть следующим образом:

```
...
Info : rp2040.core0: hardware has 4 breakpoints, 2 watchpoints
Info : rp2040.core1: hardware has 4 breakpoints, 2 watchpoints
Info : starting gdb server for rp2040.core0 on 3333
Info : Listening on port 3333 for gdb connections
```

ПРЕДУПРЕЖДЕНИЕ
Если вы видите ошибку типа Info : DAP init failed, значит, ваш Raspberry Pi Pico либо выключен, неправильно подключен, либо имеет проблемы с целостностью сигнала. Попробуйте использовать другие соединительные кабели GPIO.

Этот терминал OpenOCD необходимо оставить открытым. Итак, продолжайте и откройте другой терминал, в этом мы прикрепим экземпляр gdb к OpenOCD. Перейдите к примеру кода "Hello World" и запустите gdb из командной строки.

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

Подключите GDB к OpenOCD,

```
(gdb) target remote localhost:3333
```

СОВЕТ
Вы можете создать файл .gdbinit, чтобы вам не приходилось каждый раз вводить target remote localhost:3333. Сделайте это с помощью `echo "target remote localhost:3333" > ~/.gdbinit`. Однако это мешает отладке в VSCode (глава 7).

и загрузите hello_serial.elf во flash, приступая к работе с Raspberry Pi Pico.

```
(gdb) load
Loading section .boot2, size 0x100 lma 0x10000000
Loading section .text, size 0x22d0 lma 0x10000100
Loading section .rodata, size 0x4a0 lma 0x100023d0
Loading section .ARM.exidx, size 0x8 lma 0x10002870
Loading section .data, size 0xb94 lma 0x10002878
Start address 0x10000104, load size 13324
Transfer rate: 31 KB/sec, 2664 bytes/write.
```

а затем запустите его.

```
(gdb) monitor reset init
(gdb) continue
```

⍰ ВАЖНО

Если вы видите ошибки, похожие на ошибку завершения работы флэш-памяти или ошибку удаления флэш-памяти с помощью пакета vFlashErase в GDB, когда попытка загрузить двоичный файл на Raspberry Pi Pico через OpenOCD, вероятно, приведет к нарушению целостности сигнала между Raspberry Pi и Raspberry Pi Pico. Если вы не подключаете SWD-соединение напрямую между двумя платами, см. рис. 7, вам следует попробовать это сделать. В качестве альтернативы вы можете попробовать уменьшить значение adapter_khz в файле конфигурации raspberrypi-swd.cfg, пытаясь уменьшить его вдвое, пока не увидите успешное соединение между платами.

Поскольку мы обмениваемся битами между платами, синхронизация незначительна, поэтому плохая целостность сигнала может привести к ошибкам. Или, если вы хотите установить точку останова в main() перед запуском исполняемого файла,

```
(gdb) monitor reset init
(gdb) b main
(gdb) continue

Thread 1 hit Breakpoint 1, main () at /home/pi/pico/picoexamples/
hello_world/serial/hello_serial.c:11
11          stdio_init_all();
```

прежде чем продолжить после достижения точки останова

```
(gdb) continue
```

Чтобы выйти из gdb, введите

```
(gdb) quit
```

Более подробную информацию о том, как использовать gdb, можно найти по адресу https://www.gnu.org/software/gdb/documentation/.

Глава 7. Использование кода Visual Studio

Visual Studio Code (VSCode) - популярный редактор с открытым исходным кодом, разработанный Microsoft. Это рекомендуемая интегрированная среда разработки (IDE) на Raspberry Pi 4, если вам нужен графический интерфейс для редактирования и отладки вашего кода.

7.1. Установка кода Visual Studio

⍰ ВАЖНО
Эти инструкции по установке основаны на том, что вы уже загрузили и установили набор инструментов командной строки (см. Глава 3), а также подключение SWD к вашей плате через OpenOCD (глава 5) и настройка GDB для отладки из командной строки (глава 6). Код Visual Studio (VSCode) может быть установлен в Raspberry Pi OS с помощью обычной процедуры apt:

```
$ sudo apt update
$ sudo apt install code
```

После завершения установки установите расширения, необходимые для отладки Raspberry Pi Pico:

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

Наконец, запустите Visual Studio Code из окна терминала:

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

Убедитесь, что вы указали PICO_SDK_PATH, чтобы код Visual Studio мог найти SDK.

⍰ ПРИМЕЧАНИЕ
Если PICO_SDK_PATH не установлен по умолчанию в среде вашей оболочки, вам придется устанавливать его каждый раз, когда вы открываете новый Окно терминала перед запуском VSCode или запустите VSCode из меню. Поэтому, возможно, вы захотите добавить его в свой
файл .profile или .bashrc.

⍰ ПРИМЕЧАНИЕ
Вы можете настроить intellisense для CMake, изменив поставщика, переключив; Вид → Командная палитра → C/C++:
Изменить поставщика конфигурации… → Инструменты CMake.

7.2. Загрузка проекта

Продолжайте и откройте папку pico-examples, перейдя на панель инструментов проводника (Ctrl + Shift + E), выбрав "Открыть папку", и перейдите к /home/pi/pico/pico-примерам во всплывающем окне файла. Затем нажмите "ОК", чтобы загрузить папку в VSCode. Пока установлено расширение CMake Tools, примерно через секунду вы должны увидеть всплывающее окно в правом нижнем углу окна VSCode.

Нажмите "Да", чтобы настроить проект. Затем вам будет предложено выбрать компилятор, см. рис. 8,

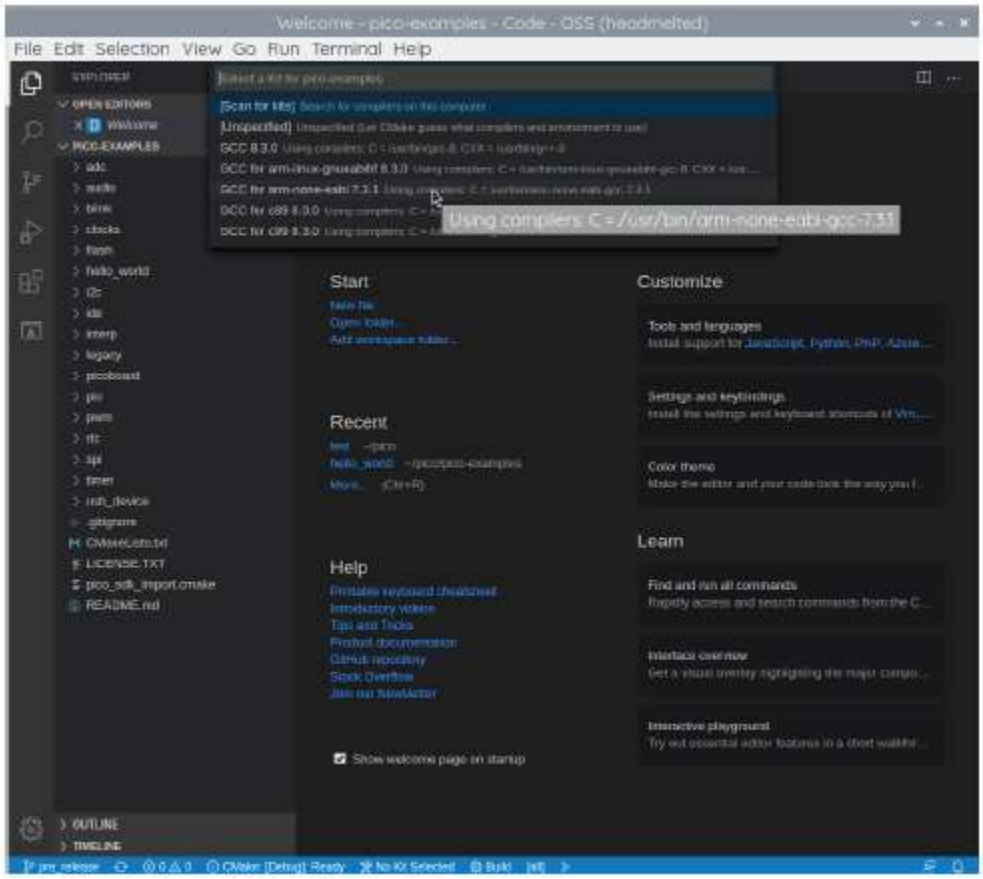


рисунок 8. Предложите выбрать правильный компилятор для проекта.

и вы должны выбрать GCC для arm-none-eabi из выпадающего меню.

💡 СОВЕТ
Если вы пропустите всплывающие окна, которые снова закроются через несколько секунд, вы можете настроить компилятор, нажав на "Нет Выбранный комплект" в синей нижней строке окна VSCode.

Затем вы можете либо нажать на кнопку "Построить" на синей нижней панели, чтобы создать все примеры в папке pico-examples, либо нажать на то место, где написано "[все]" на синей нижней панели. При этом вам откроется выпадающее меню, в котором вы сможете выбрать проект. На данный момент введите "hello_usb" и выберите исполняемый файл "Hello USB". Это означает, что VSCode создаст только пример "Hello USB", экономя время компиляции.

💡 СОВЕТ
Вы можете переключаться между созданием исполняемых файлов "Debug" и "Release", нажав на то место, где написано "CMake: [Debug]: Готово" в синей нижней строке. По умолчанию создается исполняемый файл с поддержкой "Debug", готовый к отладке SWD.

Продолжайте и нажмите на кнопку "Построить" (с зубчатым колесом) в синей нижней панели окна. Это создаст каталог сборки и запустит CMake, как мы делали вручную в разделе 3.1, перед запуском самой сборки, см. рис. 9.

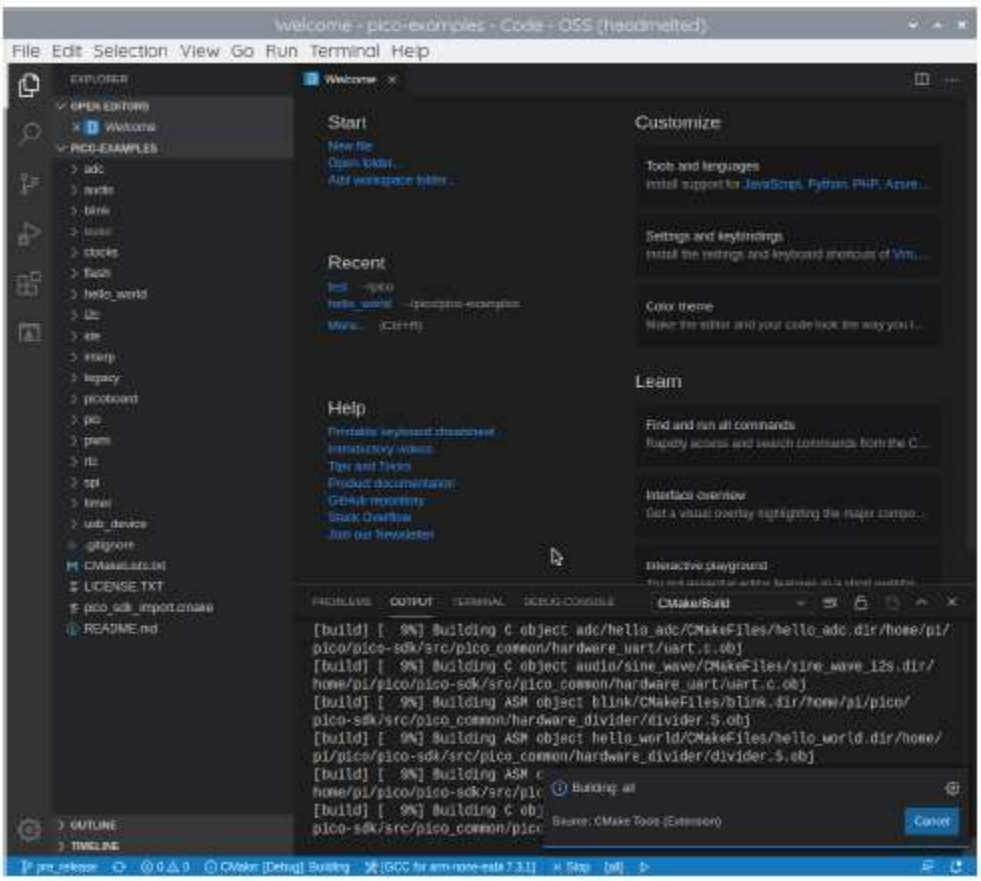


Рисунок 9. Построение проекта pico-examples в Visual Studio Code

Как мы делали ранее из командной строки, среди других целей мы теперь создали:
• hello_usb.elf, который используется отладчиком
• hello_usb.uf2, который можно перетащить на USB-накопитель RP2040

7.3. Отладка проекта

Репозиторий pico-examples содержит пример конфигурации отладки, которая запустит OpenOCD, подключит GDB и, наконец, запустит приложение, для сборки которого настроен CMake. Продолжайте и скопируйте этот файл (launch-raspberrypi-swd.json) в picoexamples/. Vscode каталог как launch.json. Мы также предоставляем файл settings.json, который мы рекомендуем вам также скопировать. Этот файл settings.json удаляет некоторые потенциально сбивающие с толку параметры из плагина CMake (включая неработающие кнопки Debug и Run , которые пытаются запустить двоичный файл Pico на хосте).

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

Примеры пико: https://github.com/raspberrypi/pico-examples/blob/master/ide/vscode/launch-raspberrypi-swd.json

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

```
11 // This may need to be "arm-none-eabi-gdb" for some previous builds
12 "gdbPath" : "gdb-multiarch",
13 "device": "RP2040",
14 "configFiles": [
15 "interface/raspberrypi-swd.cfg",
16 "target/rp2040.cfg"
17 ],
18 "svdFile": "${env:PICO_SDK_PATH}/src/rp2040/hardware_regs/rp2040.svd",
19 "runToEntryPoint": "main",
20 // Work around for stopping at main on restart
21 "postRestartCommands": [
22 "break main",
23 "continue"
24     ]
25   }
26  ]
27 }
```

⍰ Примечание

Возможно, вам придется изменить gdbPath в launch.json, если ваша gdb называется arm-none-eabi-gdb вместо gdb-multiarch

Примеры Пико: https://github.com/raspberrypi/pico-examples/blob/master/ide/vscode/settings.json

```
1 {
2 // These settings tweaks to the cmake plugin will ensure
3 // that you debug using cortex-debug instead of trying to launch
4 // a Pico binary on the host
5 "cmake.statusbar.advanced": {
6 "debug": {
7 "visibility": "hidden"
8 },
9 "launch": {
10 "visibility": "hidden"
11 },
12 "build": {
13 "visibility": "hidden"
14 },
15 "buildTarget": {
16 "visibility": "hidden"
17 }
18 },
19 "cmake.buildBeforeRun": true,
20 "C_Cpp.default.configurationProvider": "ms-vscode.cmake-tools"
21 }
```

7.3.1. Запуск "Hello USB" на Raspberry Pi Pico

⍰ ВАЖНО

Убедитесь, что пример кода "Hello USB" был собран в виде двоичного файла для отладки (CMAKE_BUILD_TYPE=Debug).

Теперь перейдите на панель инструментов отладки (Ctrl + Shift + D) и нажмите маленькую зеленую стрелку (кнопка воспроизведения) в верхней части левой панели окна, чтобы загрузить свой код на Raspberry Pi Pico и начать отладку.

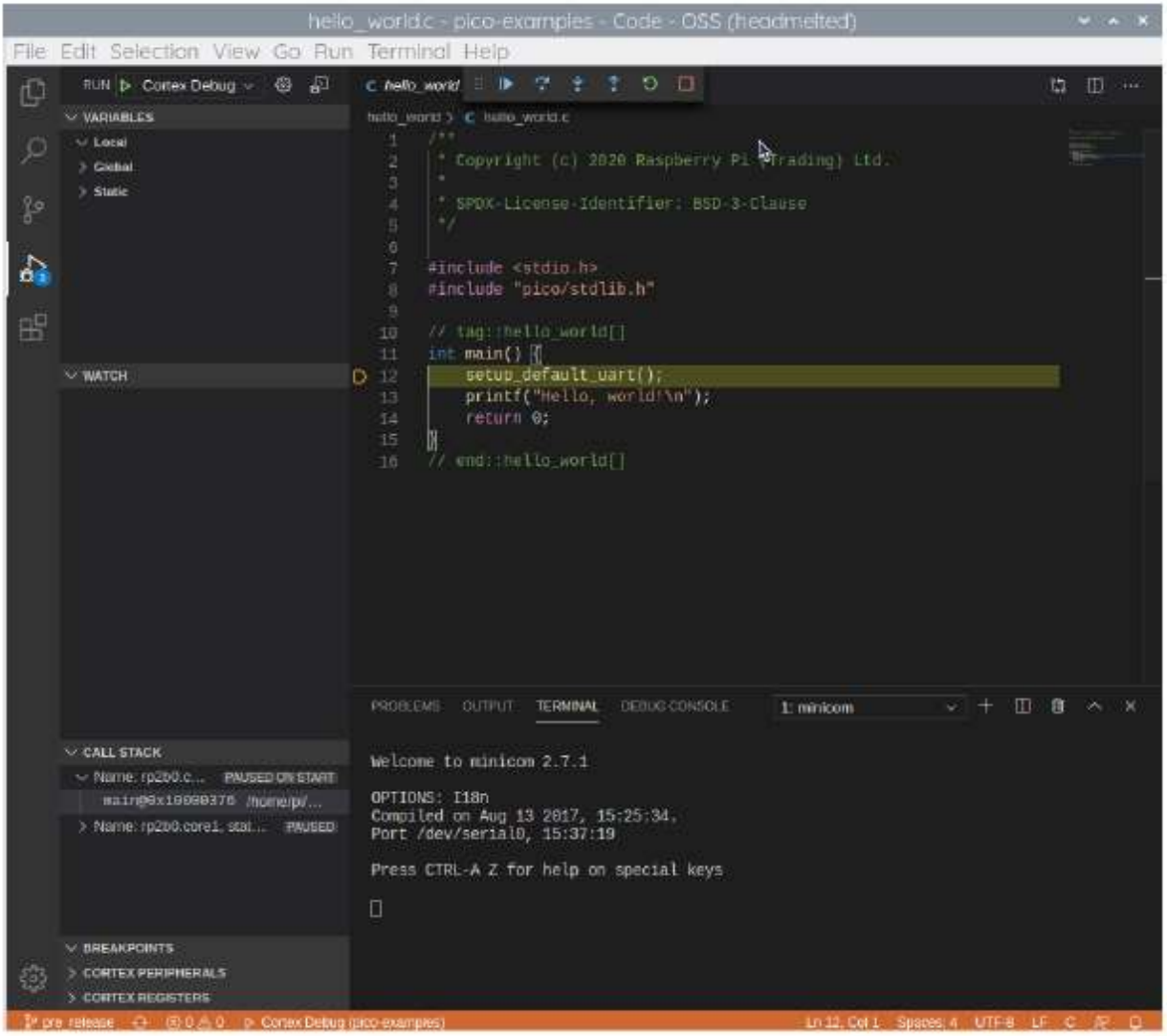


Рисунок 10. Отладка двоичного файла "Hello USB" в Visual Studio Код

Теперь код должен быть загружен в Raspberry Pi Pico, и вы должны увидеть исходный код для "Hello USB" на главной правой (верхней) панели окна. Код начнет выполняться и перейдет к первой точке останова, включенной директивой runToMain в файле launch.json. Нажмите на маленькую синюю стрелку (кнопка воспроизведения) в верхней части этого главного окна исходного кода, чтобы продолжить (F5) и запустить запуск кода.

Совет

Если вы переключитесь на вкладку "Терминал" в нижней правой панели, под кодом hello_usb.c, вы можете использовать это, чтобы открыть minicom внутри VSCode, чтобы увидеть вывод UART из примера "Hello USB", набрав,

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

в командной строке терминала, как мы делали ранее, смотрите раздел 4.4.

Глава 8. Создание собственного проекта

Продолжайте и создайте каталог для размещения вашего тестового проекта, расположенный рядом с каталогом pico-sdk,

```
$ ls -la
total 16
drwxr-xr-x 7 aa staff 224 6 Apr 10:41 ./
drwx------@ 27 aa staff 864 6 Apr 10:41 ../
drwxr-xr-x 10 aa staff 320 6 Apr 09:29 pico-examples/
drwxr-xr-x 13 aa staff 416 6 Apr 09:22 pico-sdk/
$ mkdir test
$ cd test
```

а затем создайте файл test.c в каталоге,

```
1 #include <stdio.h>
2 #include "pico/stdlib.h"
3 #include "hardware/gpio.h"
4 #include "pico/binary_info.h"
5
6 const uint LED_PIN = 25;①
7
8 int main() {
9
10 bi_decl(bi_program_description("This is a test binary."));②
11 bi_decl(bi_1pin_with_name(LED_PIN, "On-board LED"));
12
13 stdio_init_all();
14
15 gpio_init(LED_PIN);
16 gpio_set_dir(LED_PIN, GPIO_OUT);
17 while (1) {
18 gpio_put(LED_PIN, 0);
19 sleep_ms(250);
20 gpio_put(LED_PIN, 1);
21 puts("Hello World¥n");
22 sleep_ms(1000);
23 }
24 }
```

① Встроенный светодиод подключен к GP25 на Pico, если вы создаете для Pico W, светодиод подключен к CYW43_WL_GPIO_LED_PIN. Для получения дополнительной информации смотрите пример Pico Was blink в репозитории примеров Pico на Github.
② Эти строки, добавлять строки для бинарных видна через инструмент Пико, см. Приложение Б.
вместе с файлом CMakeLists.txt ,

picotool, see Appendix B.
along with a CMakeLists.txt file,

```
cmake_minimum_required(VERSION 3.13)
include(pico_sdk_import.cmake)
project(test_project C CXX ASM)
set(CMAKE_C_STANDARD 11)
```

```
cmake_minimum_required(VERSION 3.13)
include(pico_sdk_import.cmake)
project(test_project C CXX ASM)
set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)
pico_sdk_init()
add_executable(test
test.c
)
pico_enable_stdio_usb(test 1)①
pico_enable_stdio_uart(test 1)②
pico_add_extra_outputs(test)
target_link_libraries(test pico_stdlib)
```

1. Это включит последовательный вывод через USB.
2. Это включит последовательный вывод через UART.

Затем скопируйте файл pico_sdk_import.cmake из внешней папки в вашей установке pico-sdk в папку вашего тестового проекта.

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

Теперь у вас должно получиться что-то похожее на это,

```
$ ls -la
total 24
drwxr-xr-x 5 aa staff 160 6 Apr 10:46 ./
drwxr-xr-x 7 aa staff 224 6 Apr 10:41 ../
-rw-r--r--@ 1 aa staff 394 6 Apr 10:37 CMakeLists.txt
-rw-r--r-- 1 aa staff 2744 6 Apr 10:40 pico_sdk_import.cmake
-rw-r--r-- 1 aa staff 383 6 Apr 10:37 test.c
```

и можем построить его так, как мы делали это раньше в нашем примере "Hello World".

```
$ mkdir build
$ cd build
$ export PICO_SDK_PATH=../../pico-sdk
$ cmake ..
$ make
```

В процессе создания будет создано несколько различных файлов.
Важные из них приведены в следующей таблице.

асширения файла	Описание
.bin	Необработанный двоичный дамп программного кода и данных
.elf	Полный вывод программы, возможно, включая отладочную информацию
.uf 2	Программный код и данные в форме UF2, которые жете перетащить на RP2040 плата, когда она подключена как USB-накопитель
.dis	Дизассемблирование скомпилированного двоичного
файл .hex	скомпилированного двоичного
файла .map	Файл карты, сопровождающий файл
.elf	описывающий, где компоновщик расположил сегменты в памяти

Примечание
UF2 (USB Flashing Format) - это формат файла, разработанный Microsoft, который используется для перепрошивки платы RP2040 через USB. Более подробную информацию можно найти в репозитории спецификации Microsoft UF2

ПРИМЕЧАНИЕ
Чтобы создать двоичный файл для запуска в SRAM, а не во флэш-памяти, вы можете либо настроить сборку cmake с помощью -DPICO_NO_FLASH=1 или вы можете добавить pico_set_binary_type(TARGET_NAME no_flash), чтобы управлять им для каждого двоичного файла в вашем CMakeLists.txt файл. Вы можете загрузить двоичный файл RAM в RP2040 через UF2. Например, если на устройстве нет флэш-чипа на вашу плату вы можете загрузить двоичный файл, который запускается во встроенной оперативной памяти, используя UF2, поскольку он просто указывает адреса, по которым отправляются данные. Примечание. вы можете загружать только в RAM или FLASH, но не в оба.

8.1. Отладка вашего проекта

Отладка вашего собственного проекта из командной строки выполняется по тем же процессам, которые мы использовали для "Hello World". Пример приведен в разделе 6.3. Подключите Raspberry Pi и Raspberry Pi Pico, как показано на рисунке 11.

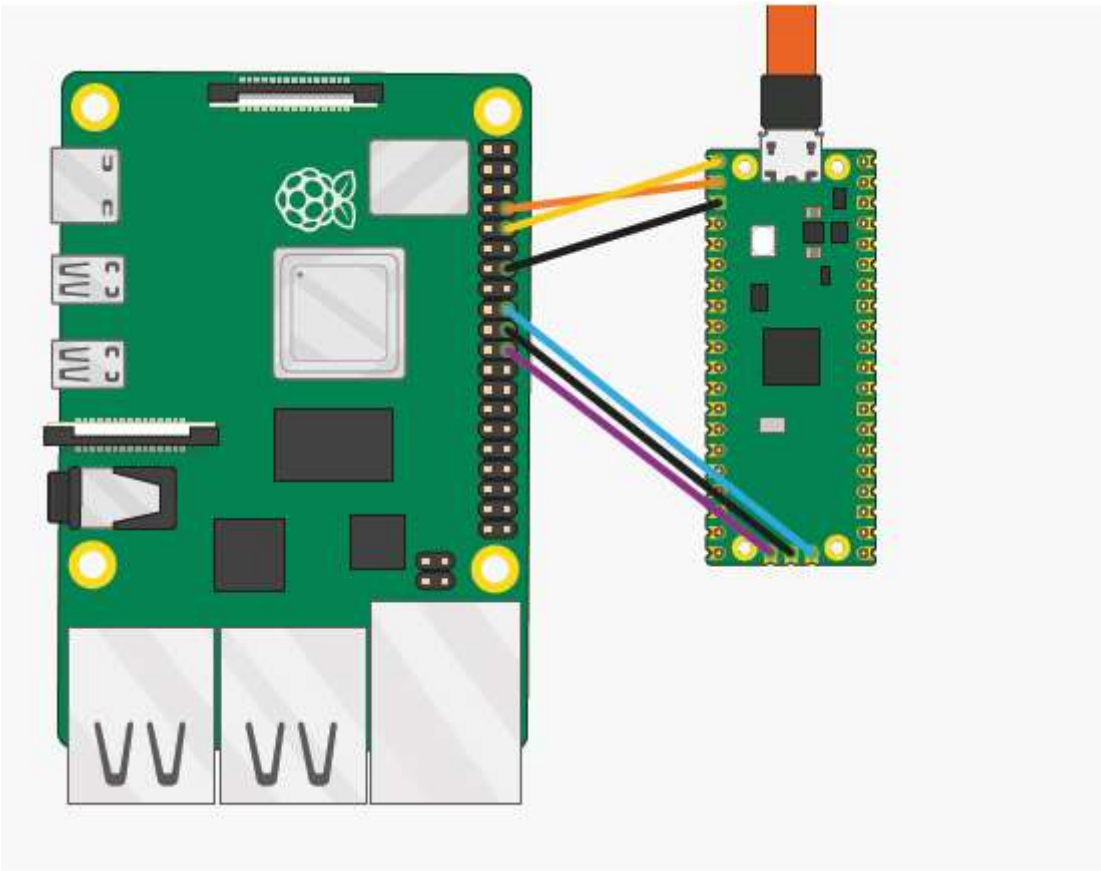


Рисунок 11. Raspberry Pi 4 и Raspberry Pi Pico с
подключенными друг к другу отладочными портами UART и SWD

Оба они подключаются непосредственно к Raspberry Pi 4 без используя макетную доску.
Затем продолжайте и создайте отладочную версию вашего проекта, используя CMAKE_BUILD_TYPE=Debug, как показано ниже,

```
$ cd ~/pico/test
$ rmdir build
$ mkdir build
$ cd build
$ export PICO_SDK_PATH=../../pico-sdk
$ cmake -DCMAKE_BUILD_TYPE=Debug ..
$ make
```

Затем откройте окно терминала и подключите OpenOCD, используя интерфейс raspberrypi-swd.

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

Этот терминал OpenOCD необходимо оставить открытым. Итак, продолжайте и откройте другое окно терминала и запустите gdb-multiarch , используя

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

Подключите GDB к OpenOCD и загрузите двоичный файл test.elf во flash,

```
(gdb) target remote localhost:3333
(gdb) load
```

и затем запустите его,

```
(gdb) monitor reset init
(gdb) continue
```

8.2. Работа с кодом Visual Studio

Если вы хотите работать с кодом Visual Studio, а не из командной строки, вы можете сделать это, подробнее смотрите в главе 7 инструкции о том, как настроить среду и загрузить ваш новый проект в среду разработки, чтобы вы могли писать и компоновать код.
Если вы хотите также использовать Visual Studio Code для отладки и загрузки вашего кода в Raspberry Pi Pico, вам нужно будет создать файл launch.json для вашего проекта. Пример запуска- launch - raspberrypi-swd.json в главе 7 должен сработать. Вам нужно скопировать его как . vscode/launch. json.

8.3. Автоматизация создания проекта

Генератор проектов pico автоматически создает "заглушку" проекта со всеми необходимыми файлами, позволяющими его создавать. Если вы хотите воспользоваться этим, вам нужно будет пойти дальше и клонировать скрипт создания проекта из его репозитория Git,

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

Затем его можно запустить в графическом режиме,

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

который откроет графический интерфейс, позволяющий вам настроить ваш проект, смотрите рисунок 12.

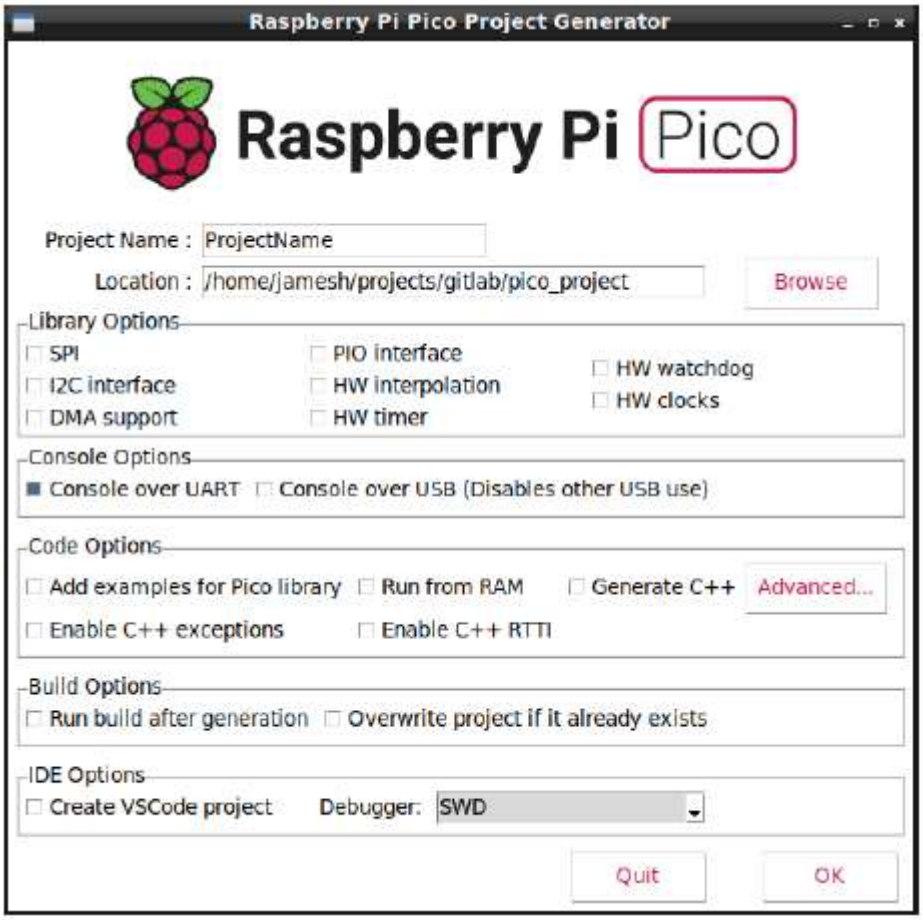


Рисунок 12. Создание проекта RP2040 с использованием графического инструмента создания проектов.

Вы можете добавить определенные функции в свой проект, установив их с помощью флажков в графическом интерфейсе. Это гарантирует, что система сборки добавит соответствующий код в сборку, а также добавит в проект простой пример кода, показывающий, как использовать эту функцию.

Доступно несколько опций, которые обеспечивают следующие функциональные возможности.

Опции консоли	Описание
Консоль через UART	Включите последовательную консоль через UART. Это значение по умолчанию.
Консоль через USB	Включите консоль через USB. Устройство будет выступать в качестве последовательного порта USB. Это можно использовать в дополнение к опции UART или вместо нее, но обратите внимание, что при включенной функции другие функции USB невозможны.

Параметры кода	Описание
Добавить примеры для библиотеки Pico **Add examples for Pico library**	нерирован пример кода для некоторых функций стандартной библиотек анию присутствуют в сборке, например, поддержка UART и разделители HV
Запуск из оперативной памяти **Run from RAM**	Обычно при сборке создается двоичный файл, который будет установлен во флэш-память. Это заставляет двоичный файл работать непосредственно из оперативной памяти

Сгенерируй C++ Generate C++	Сгенерированные исходные файлы будут совместимы с C++
Включить исклю- чения C++ Enable C++ exceptions	Обычно отключен для экономии места в коде.
Включить C++ RTTI Enable C++ RTTI	Включить информацию о типе времени выполнения C++. Обычно отключен для экономииместа в коде.
Расширенный Advanced	крывает таблицу, позволяющую выбрать конкретные варианты сборки и параметры изменяют способ работы функций, и их следует использовать осторожностью.

параметры сборки	Описание
Запустить сборку Run Build	Как только проект будет создан, соберите его. В результате будут созданы файлы, готовые для загрузки на Raspberry Pi Pico.
Перезаписать проект Overwrite Project	Если проект уже существует в указанной папке, перезапишите его новымпроектом. Это перезапишет все изменения, которые вы, возможно, внесли.

Параметры IDE	Описание
Создть проект VSCode Create VSCode Project	А также файлы CMake, также создайте соответствующие файлы проекта Visual Studio Code.
Отладчик Debugger	Выберите, какой отладчик Pico будет использоваться системой отладки VSCode. По умолчанию используется отладка по последовательному каналу.

8.3.1. Генерация проекта из командной строки

Скрипт также предоставляет возможность создавать проект из командной строки, например

```
$ export PICO_SDK_PATH="/home/pi/pico/pico-sdk"
$ ./pico_project.py --feature spi --feature i2c --project vscode test
```

Параметры --feature добавляют соответствующий библиотечный код в сборку, а также пример кода, показывающий базовое использование функции. Вы можете добавить множество функций, вплоть до ограничения памяти RP2040. Вы можете использовать опцию –list скрипта, чтобы перечислить все доступные функции. В приведенном выше примере добавлена поддержка интерфейсов I2C и SPI.
Здесь передача параметра --project vscode будет означать, что .vscode/launch.json, .vscode/c_cpp_properties.json и .vscode/settings.json также создаются в дополнение к файлам проекта CMake.
После создания вы можете собрать проект обычным способом из командной строки,

```
$ cd test/build
$ cmake ..
$ make
```

или из кода Visual Studio.

Вы можете использовать опцию --help, чтобы предоставить список аргументов командной строки, они также будут применяться при использовании графического режима.

Нужна более подробная информация?

Здесь должно быть достаточно, чтобы показать вам, с чего начать, но вы можете задаться вопросом, почему некоторые из этих файлов и заклинаний необходимы. В книге Raspberry Pi Pico C/C++ SDK рассказывается о том, как на самом деле строится ваш проект и как строчки в нашем CMakeLists.txt файлы здесь относятся к структуре SDK, если вы захотите узнать больше в какой-то момент в будущем.

Глава 9. Построение на других платформах

Хотя основной поддерживаемой платформой для разработки RP2040 является Raspberry Pi, доступна поддержка других платформ, таких как Apple macOS и Microsoft Windows.

9.1. Построение на Apple macOS

Использование macOS для создания кода для RP2040 очень похоже на Linux.

9.1.1. Установка набора инструментов

Установка зависит от Homebrew, если у вас не установлен Homebrew, вам следует продолжить и установить его,

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

Затем установите набор инструментов,

```
$ brew install cmake
$ brew tap ArmMbed/homebrew-formulae
$ brew install arm-нет-eabi-gcc
```

Однако после этого вы можете следовать инструкциям Raspberry Pi для создания кода для RP2040. Как только набор инструментов установлен, различий между macOS и Linux нет, поэтому смотрите раздел 2.1 и следуйте инструкциям оттуда

— пропуская раздел, где вы устанавливаете набор инструментов, — чтобы получить SDK и создать пример "Blink".

Примечание

Если вы работаете на Mac под управлением Apple M1, вам нужно будет установить Rosetta 2, поскольку компилятор Arm по-прежнему скомпилирован только для процессоров x86 и не имеет встроенной версии Arm.

```
$ /usr/sbin/softwareupdate --install-rosetta --agree-to-license
```

9.1.2. Использование кода Visual Studio

Visual Studio Code (VSCode) является кроссплатформенной средой и работает на macOS, а также Linux и Microsoft Окна. Скачайте версию для macOS, распакуйте ее и перетащите в папку "Приложения".

Перейдите в раздел Приложения и нажмите на значок, чтобы запустить Visual Studio Code.

9.1.3. Сборка с помощью CMake Tools

После запуска Visual Studio Code вам необходимо установить расширение CMake Tools. Нажмите на значок расширений на левой панели инструментов (или введите Cmd + Shift + X), найдите " CMake Tools " и щелкните по записи в списке, а затем нажмите на кнопку установить.

Теперь нам нужно установить переменную окружения PICO_SDK_PATH. Перейдите в каталог pico-examples и создайте каталог .vscode и добавьте файл с именем settings.json, чтобы указать CMake Tools местоположение SDK. Дополнительно укажите Visual Studio на расширение CMake Tools.

```
{
	"cmake.environment": {
		"PICO_SDK_PATH":"../../pico-sdk"
	},
}
```

Теперь нажмите на шестеренку внизу панели навигации в левой части интерфейса и выберите "Настройки". Затем на панели настроек нажмите "Расширения" и "Конфигурация CMake Tools". Затем прокрутите вниз до "Cmake: Generator" и введите в поле "Unix Makefiles".

⍰ ПРИМЕЧАНИЕ
В зависимости от вашей локальной настройки вам может не понадобиться вручную устанавливать генератор CMake на "Unix Makefiles". Однако, если вы этого не сделаете, в некоторых случаях Visual Studio по умолчанию будет использовать ninja, а не make, и сборка может завершиться ошибкой как GCC
выводит информацию о зависимостях в слегка неправильном формате, который ninja не может понять.
Если вам все же придется настраивать эту переменную вручную, также возможно, что вам потребуется указать Visual Studio с расширением CMake Tools явно, добавив дополнительную строку в свой файл settings.json,

```
{
	"cmake.environment": {
		"PICO_SDK_PATH": "../../pico-sdk"
	},
	"C_Cpp.default.ConfigurationProvider": "ms-vscode.cmake-tools"
}
```

Затем перейдите в меню "Файл" и нажмите "Добавить папку в рабочую область...", перейдите в репозиторий pico-examples и нажмите "Ок". Проект загрузится, и вам (вероятно) будет предложено выбрать компилятор, см. рис. 13. Выберите "GCC для arm-noneeabi" для вашего компилятора.

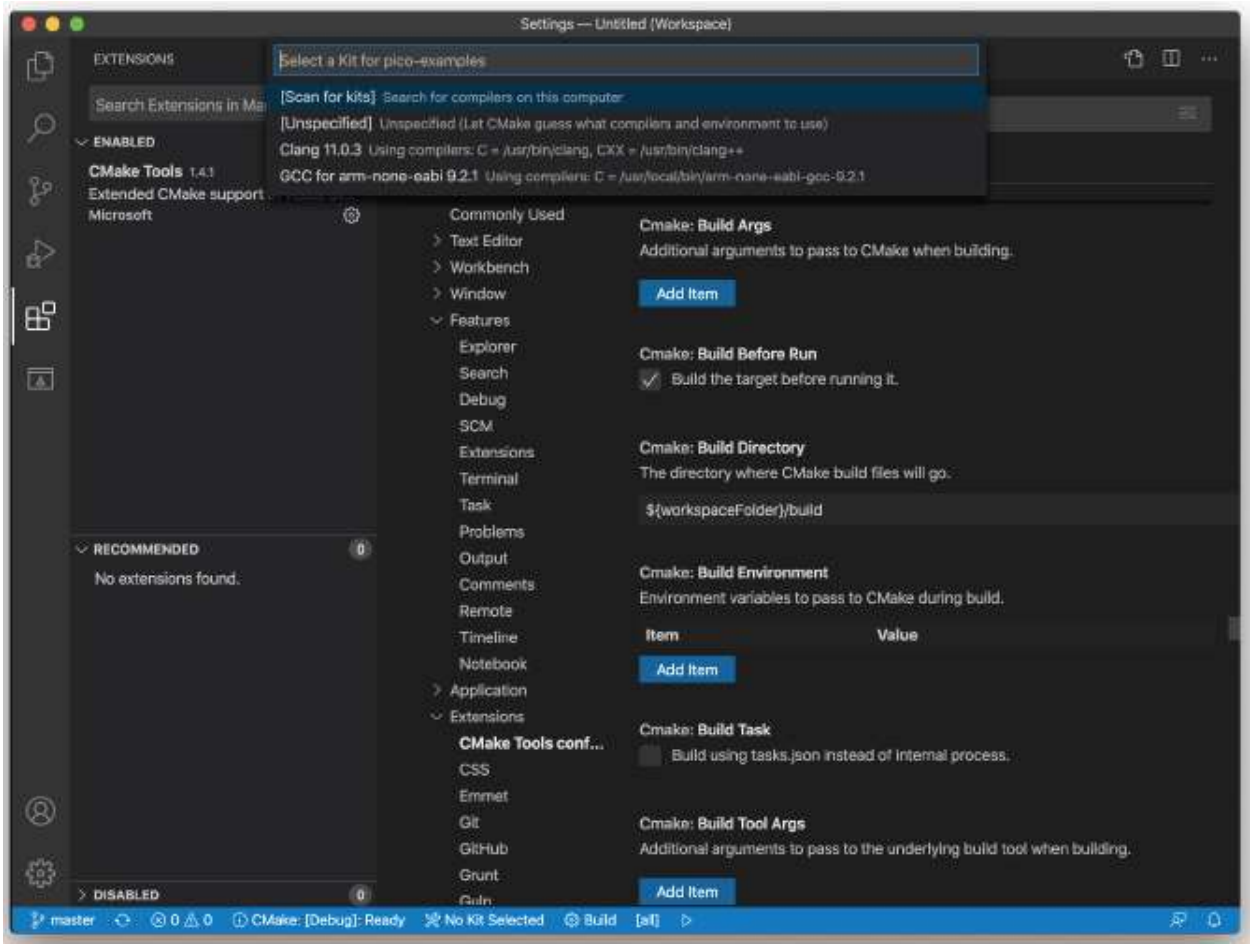


Рисунок 13. Запрос на выбор правильного компилятора для проекта.

Наконец, продолжайте и нажмите на кнопку "Построить" (с зубчатым колесом) в синей нижней панели окна. Это создаст каталог сборки и запустит CMake, как мы делали вручную в разделе 3.1, перед запуском самой сборки, см. рис. 9.

Это приведет к созданию целевых объектов elf, bin и uf2, вы можете найти их в каталогах hello_world/serial и hello_world/usb внутри недавно созданного каталога сборки. Двоичные файлы UF2 можно перетаскивать непосредственно на плату RP2040, подключенную к вашему компьютеру с помощью USB.

9.1.4. Сказать "Привет, мир"

Как мы делали ранее в главе 4, вы можете создать пример Hello World с помощью stdio, маршрутизируемого либо на USB CDC (последовательный), либо на UART0 на выводах GP0 и GP1. Установка драйвера не требуется, если вы создаете с USB CDC в качестве целевого выхода, поскольку это устройство соответствует классу. Вам просто нужно использовать терминальную программу, например Serial или аналогичную, для подключения к последовательному порту USB.

9.1.4.1. Вывод UART

В качестве альтернативы, если вы хотите, вы можете подключиться к стандартному UART Raspberry Pi Pico, чтобы увидеть выходные данные, которые вам понадобятся чтобы подключить Raspberry Pi Pico к компьютеру Mac, используйте последовательный конвертер USB в UART, например SparkFun FTDI Базовая плата, см. рис. 14.

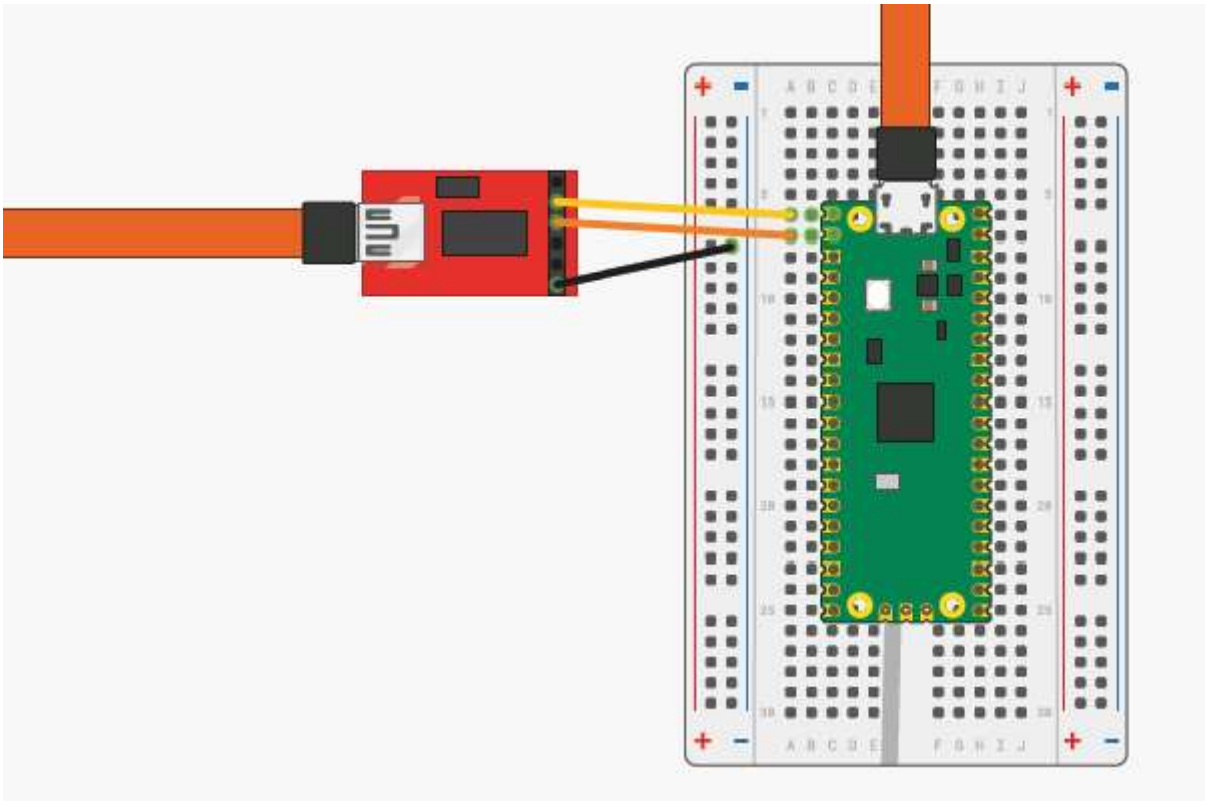


Рисунок 14. Sparkfun Базовый адаптер FTDI , подключенный к Raspberry Pi Pico

Пока вы используете последнюю версию macOS, такую как Catalina, драйверы уже должны быть загружены. В противном случае обратитесь к веб-сайту производителя за драйверами микросхем FTDI. Затем вам следует использовать терминальную программу, например Serial или аналогичную, для подключения к последовательному порту. Serial также включает поддержку драйверов.

9.2. Использование MS Windows

Установка набора инструментов в Microsoft Windows 10 или Windows 11 несколько отличается от других платформ. Однако после установки код сборки для RP2040 в чем-то схож.

⍰ ВНИМАНИЕ

Использование Raspberry Pi Pico с Windows 7 или 8 официально не поддерживается, но его можно заставить работать.

9.2.1. Установка набора инструментов

Если вы разрабатываете для Raspberry Pi Pico в Microsoft Windows, установку набора инструментов можно выполнить с помощью установщика Windows Pico. Продолжайте и загрузите последнюю версию установщика с Github, https://github.com/raspberrypi/pico-setupwindows и запустите ее. Мастер установки, см. рис. 15, проведет вас по установке набора инструментов.

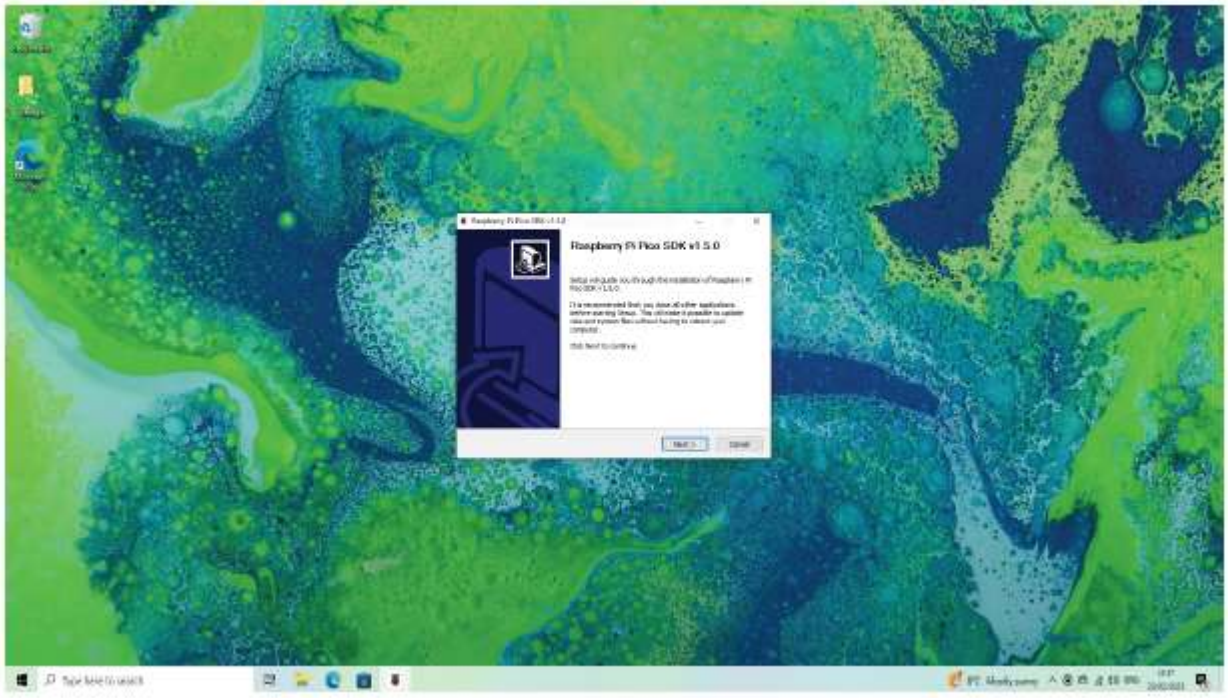


Рисунок 15. Установка Raspberry Pi Pico SDK с использованием Windows Pico установщика.

После завершения установки мастер предложит вам, следует ли отображать руководство, и при необходимости клонировать и создать репозиторий pico-examples из Github.

Прилагаемое программное обеспечение:

- Arm GNU Toolchain
- CMake
- Ninja
- Python 3.9
- Git для Windows
- Код Visual Studio
- OpenOCD

9.2.1.1. Запуск кода Visual Studio

В меню "Пуск" найдите ярлык Pico - Visual Studio Code в папке Raspberry Pi Pico SDK <версия>. То ярлык устанавливает необходимые переменные окружения, а затем запускает код Visual Studio.

 ПРИМЕЧАНИЕ

Если у вас ранее был установлен Visual Studio Code, мы часто сталкиваемся с проблемами из-за случайных настроек, вставленных другими расширениями или пользователем в существующую установку. Если это относится к вам, пожалуйста, перейдите к установщику Pico Wiki для получения контрольного списка известных проблем и решений.

9.2.1.2. Открытие примеров

При первом запуске Visual Studio Code с помощью сочетания клавиш меню "Пуск" откроется репозиторий pico-examples. Чтобы позже повторно открыть репозиторий примеров, вы можете открыть копию, установленную по адресу C:\ProgramData\Raspberry Pi\Pico SDK<version>\pico-examples.

9.2.1.3. Построение примеров

Visual Studio Code спросит, хотите ли вы настроить проект pico-examples при его первом открытии; нажмите "Да" в этом приглашении, чтобы продолжить. (Если вы пропустили приглашение, найдите значок "колокольчик" в правом нижнем углу.) Затем вам будет предложено выбрать набор — выберите набор инструментов Pico ARM GCC - Pico SDK с записью GCC arm-none-eabi. Если запись GCC Pico ARM является отсутствует, выберите Не указано, чтобы SDK автоматически определял компилятор.

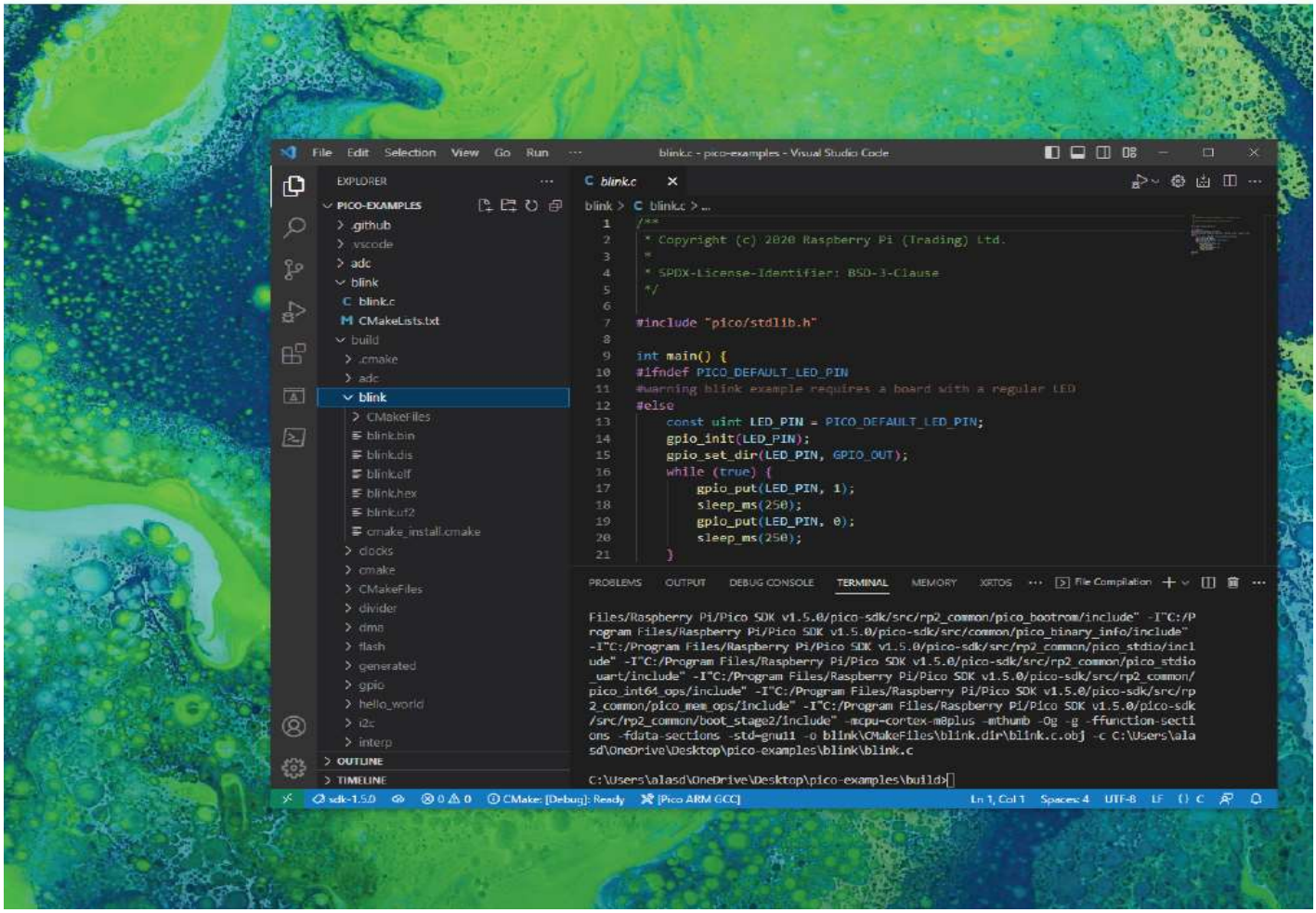


Рисунок 16. Визуальный Студийный код и репозиторий pico-examples

.

Чтобы создать один из примеров, нажмите кнопку "CMake" на боковой панели. Вам должно быть представлено древовидное представление примеров проектов; разверните проект, который вы хотели бы создать, и щелкните маленький значок сборки справа от названия цели, чтобы создать этот конкретный проект.
Чтобы вместо этого создать все, нажмите кнопку "Создать все проекты" в верхней части представления схемы проекта CMake.

ПРИМЕЧАНИЕ

Вы можете найти дополнительную информацию о том, как использовать код Visual Studio, в руководстве по установке Pico для Windows.

ВАЖНО

Если у вас возникли проблемы с установкой или использованием набора инструментов в Windows

9.2.2. Альтернативная установка вручную

ПРЕДУПРЕЖДЕНИЕ

Ручная установка набора инструментов в MS Windows сложна и не рекомендуется.
Для сборки вам потребуется установить некоторые дополнительные инструменты.

- Arm GNU Toolchain (вам нужно, чтобы имя файла заканчивалось на -arm-none-eabi.exe)
- CMake
- Инструменты сборки для Visual Studio 2022
- Python 3.10
- Git

Загрузите исполняемый установщик для каждого из них по ссылкам выше, а затем внимательно следуйте инструкциям в следующих разделах, чтобы установить все пять пакетов на свой компьютер с Windows.

9.2.2.1. Установка Arm GNU Toolchain

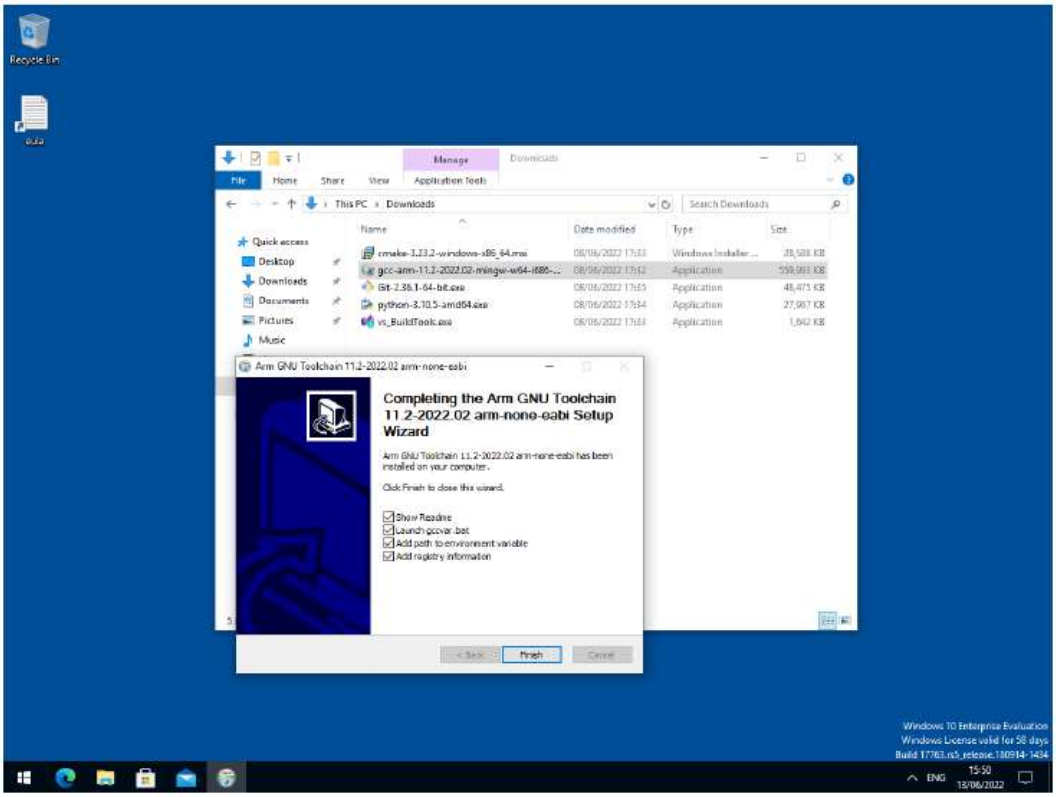


Рисунок 17. Установка Arm GNU Набор инструментов. Убедитесь, что вы зарегистрировали путь к компилятору в качестве переменной окружения, чтобы он был доступен из командной строки.

Во время установки вы должны установить флажок, чтобы зарегистрировать путь к компилятору Arm в качестве переменной окружения в оболочке Windows при появлении соответствующего запроса.

9.2.2.2. Установка CMake

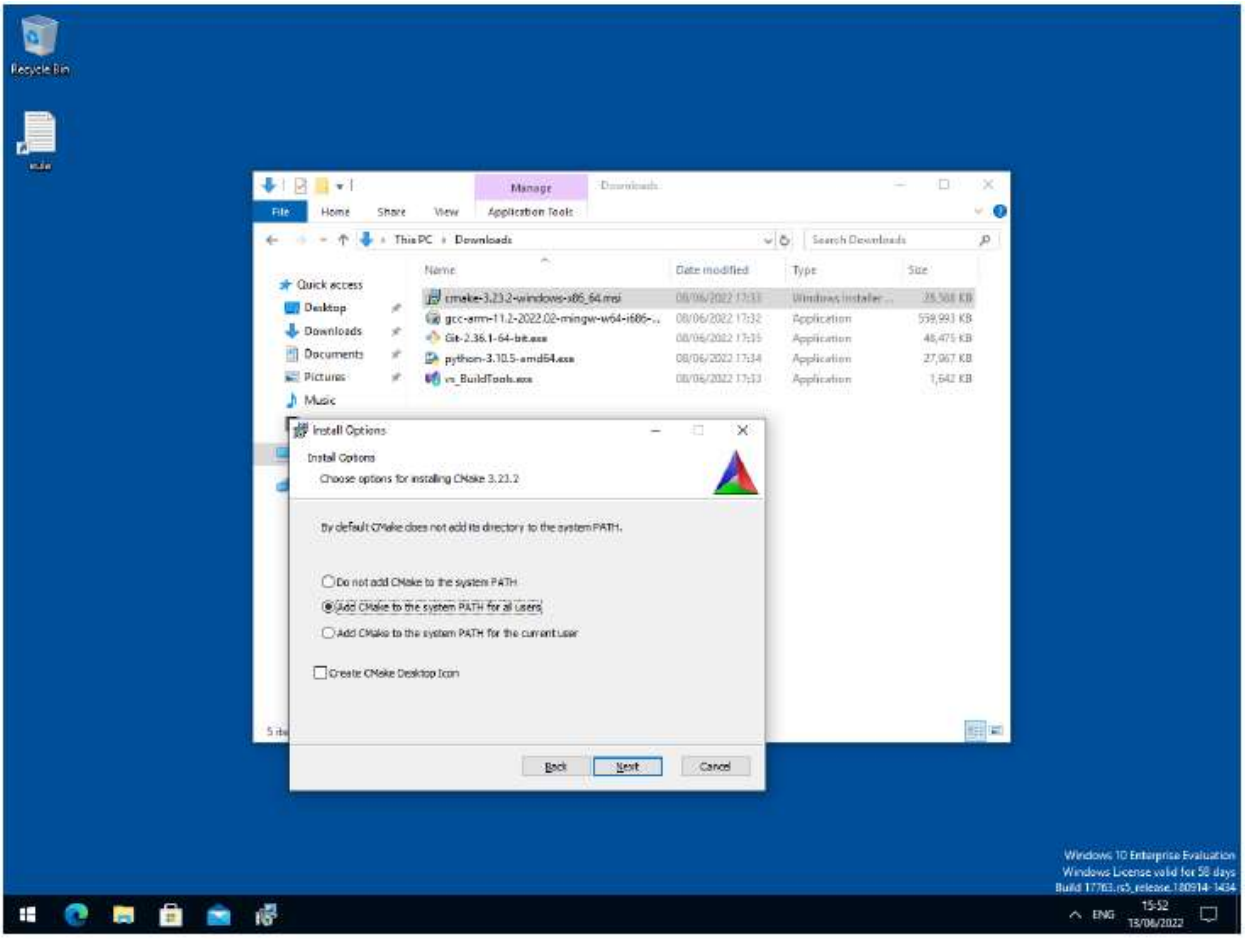


Рисунок 18. Установка CMake.

Во время установки добавьте CMake в системный путь для всех пользователей по запросу установщика.

9.2.2.3. Установка средств сборки для Visual Studio 2022

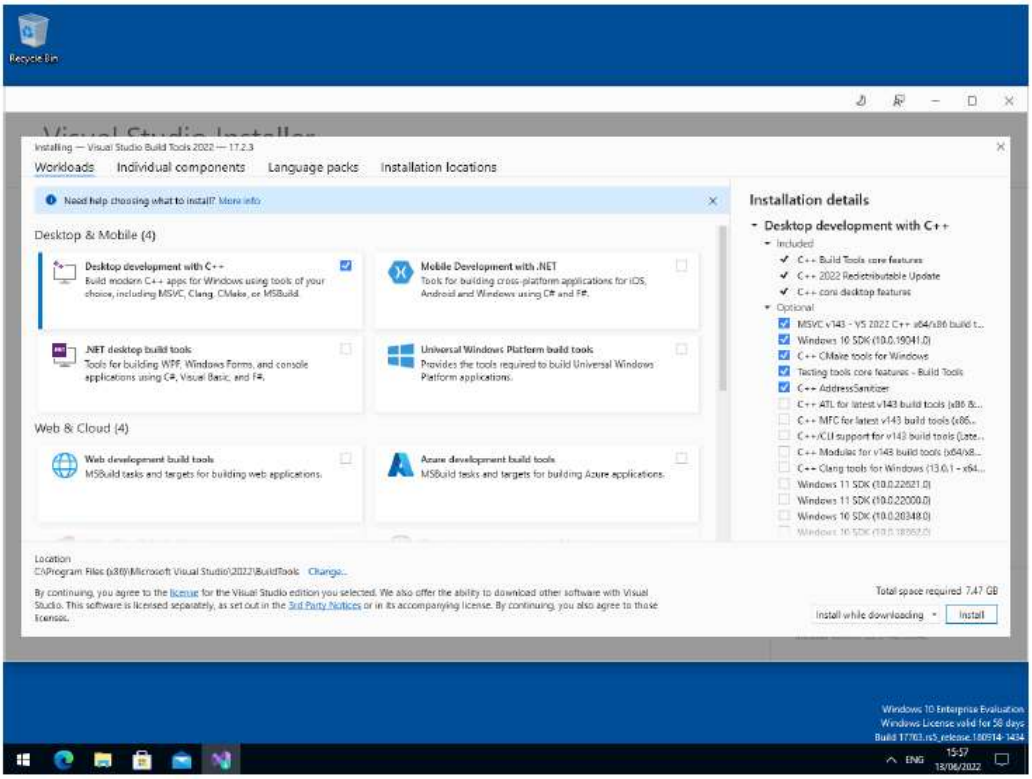


Рисунок 19. Установка инструментов сборки для Visual Studio 2022.

При появлении запроса в программе установки Build Tools for Visual Studio вам необходимо установить только инструменты сборки C++.

⍰ ПРИМЕЧАНИЕ

Вы должны установить полный пакет "Windows 10 SDK", поскольку SDK потребуется для локальной сборки инструментов pioasm и elf2uf2. Удаление его из списка установленных элементов будет означать, что вы не сможете создавать двоичные файлы Raspberry Pi Pico.

9.2.2.4. Установка Python 3.10

Во время установки убедитесь, что он установлен "для всех пользователей", и добавьте Python 3.10 в системный путь по запросу установщик. Вам следует дополнительно отключить ограничение длины MAX_PATH при появлении запроса в конце установки Python.

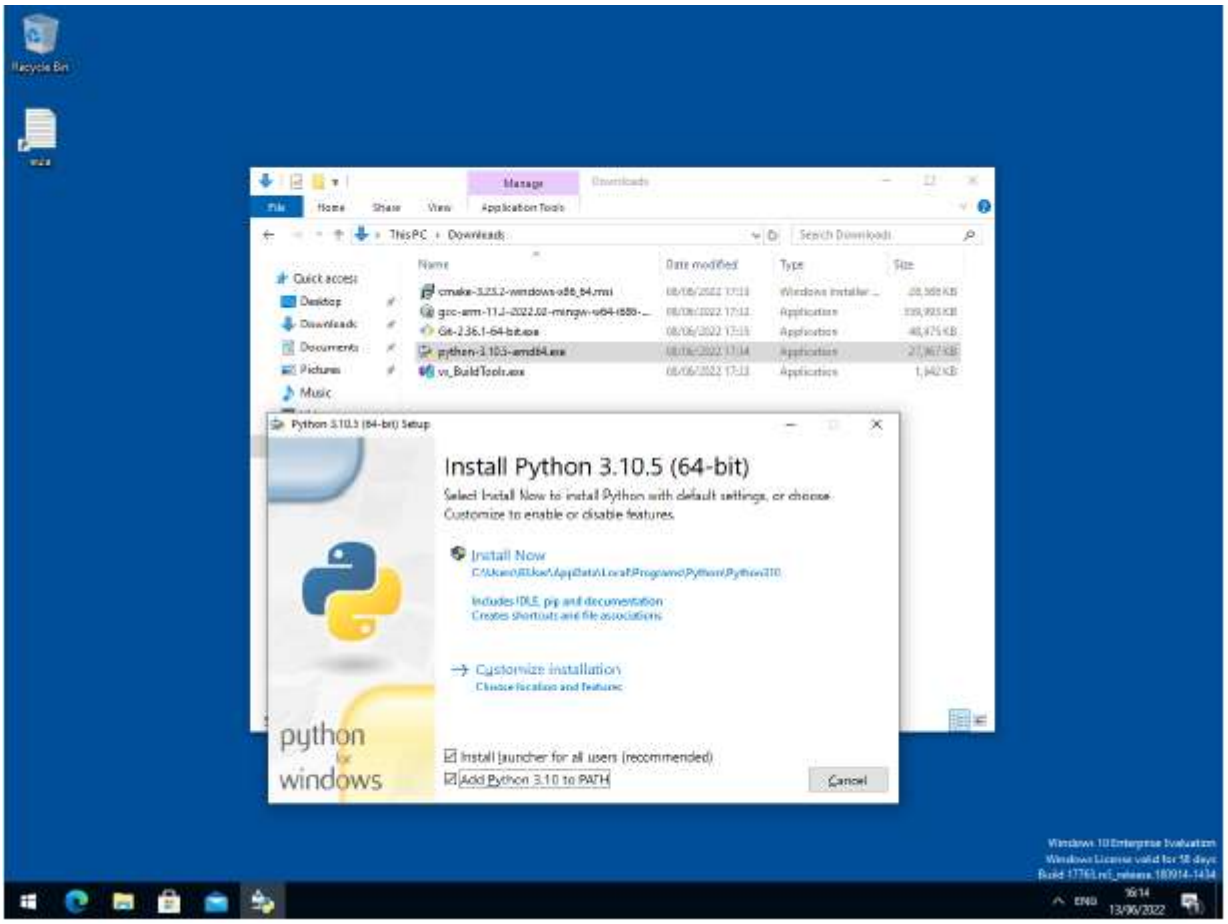


Рисунок 20. Установка Python 3.10 установите флажок "Добавить Python 3.10 в PATH".

9.2.2.5. Установка Git

При установке Git вам следует убедиться, что вы изменили редактор по умолчанию на vim, см. рисунок 21.

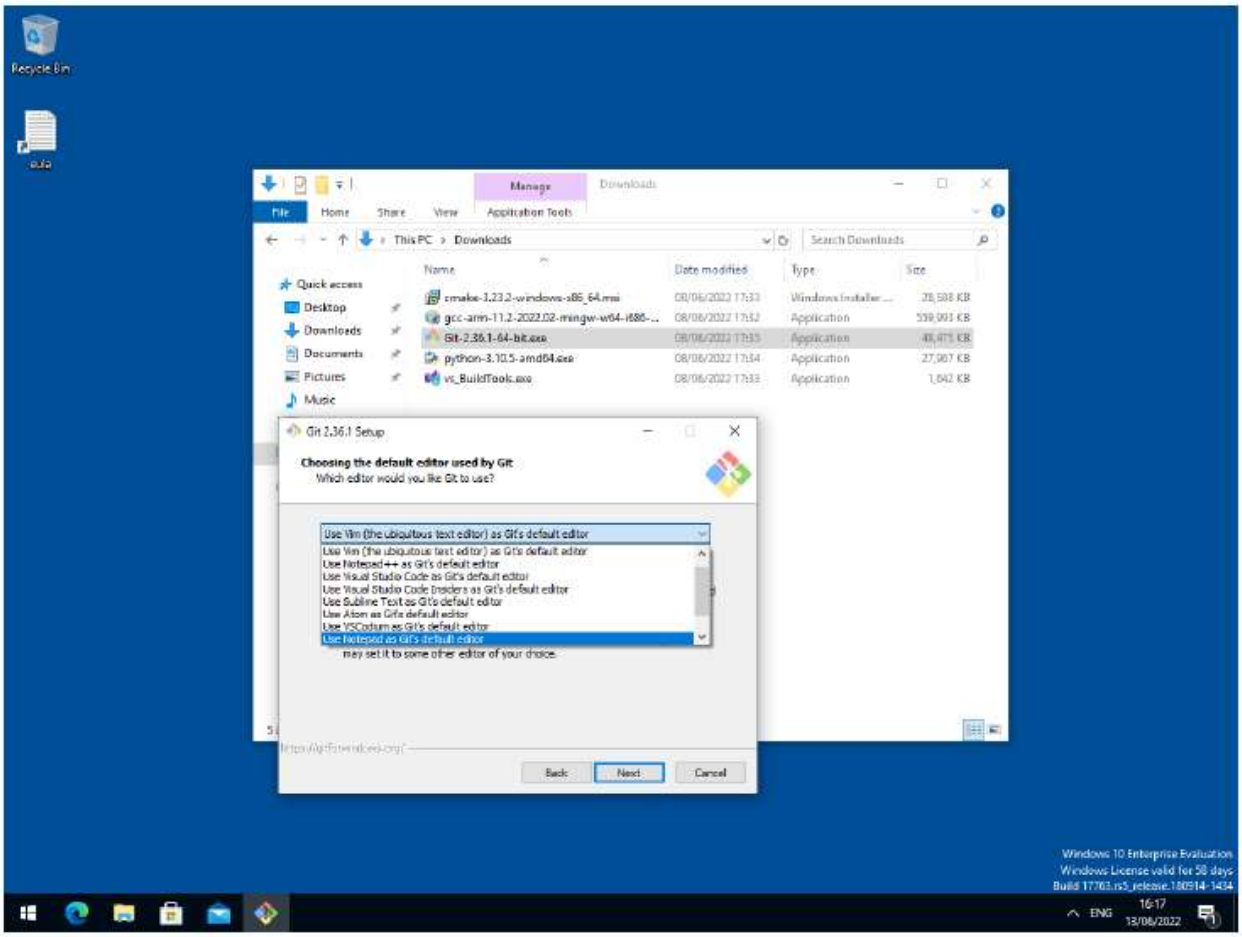


Рисунок 21. Установка Git.

Убедитесь, что вы установили флажок, разрешающий использование Git из стороннего программного обеспечения, и, если у вас нет веских причини для иного, при установке Git вам также следует установить флажок "Оформить заказ как есть, зафиксировать как есть", выбрать "Использовать консольное окно Windows по умолчанию" и "Включить экспериментальный поддержка псевдо-консолей" в процессе установки.

9.2.2.6. Получение SDK и примеров

```
C:/Users/pico/Downloads> git clone https://github.com/raspberrypi/pico-sdk.git --
branch master
C:/Users/pico/Downloads> cd pico-sdk
C:/Users/pico/Downloads/pico-sdk> git submodule update --init
C:/Users/pico/Downloads/pico-sdk> cd ..
C:/Users/pico/Downloads> git clone https://github.com/raspberrypi/pico-examples.git
--branch master
```

9.2.2.7. Создание "Hello World" из командной строки

Продолжайте и откройте окно командной строки разработчика в меню Windows, выбрав Windows > VisualStudio 2022 > Developer Command Prompt for VS 2022 из меню. Затем укажите путь к SDK следующим образом,

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

Теперь вам нужно закрыть текущее окно командной строки и открыть второе окно командной строки разработчика , где эта переменная окружения теперь будет установлена правильно, прежде чем продолжить.
Перейдите в папку pico-examples и создайте пример "Hello World" следующим образом,

```
C:\Users\pico\Downloads> cd pico-examples
C:\Users\pico\Downloads\pico-examples> mkdir build
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
```

для создания целевого объекта. Это приведет к созданию целевых объектов elf, bin и uf2, вы можете найти их в каталогах hello_world/serial и hello_world/usb внутри вашего каталога сборки. Двоичные файлы UF2 можно перетаскивать непосредственно на плату RP2040, подключенную к вашему компьютеру с помощью USB.

9.2.2.8. Создание "Hello World" из кода Visual Studio

Теперь, когда вы установили набор инструментов, вы можете установить Visual Studio Code и создавать свои проекты внутри этой среды, а не из командной строки.
Продолжайте, скачайте и установите Visual Studio Code для Windows. После установки откройте команду разработчика Окно командной строки в меню Windows, выбрав **Windows > Visual Studio 2022 > Developer Command Prompt for VS2022** из меню. Затем введите,

```
C:> code
```

в командной строке. Это откроет код Visual Studio со всеми правильными переменными окружения, установленными таким образом, чтобы набор инструментов был правильно настроен.

⍰ ПРЕДУПРЕЖДЕНИЕ

Если вы запустите Visual Studio Code, щелкнув по значку на рабочем столе или непосредственно из меню "Пуск ", среда сборки будет настроена неправильно. Хотя позже это можно будет сделать вручную в настройках CMake Tools, самый простой способ настроить среду Visual Studio Code - просто открыть ее с помощью команды разработчика Окно запроса, в котором эти переменные среды уже установлены.
Теперь нам нужно установить расширение CMake Tools. Нажмите на значок расширений на левой панели инструментов (или введите Ctrl + Shift + X), и найдите "CMake Tools" и нажмите на запись в списке, а затем нажмите на кнопку установить.
Затем нажмите на шестеренку внизу панели навигации в левой части интерфейса и выберите "Настройки". Затем на панели настроек нажмите "Расширения", а затем " CMake Tools". Затем прокрутите вниз до "**Cmake: Configure Environment**". Нажмите на "Добавить элемент" и установите PICO_SDK_PATH равным ..\..\pico-sdk, как показано на рисунке 22.

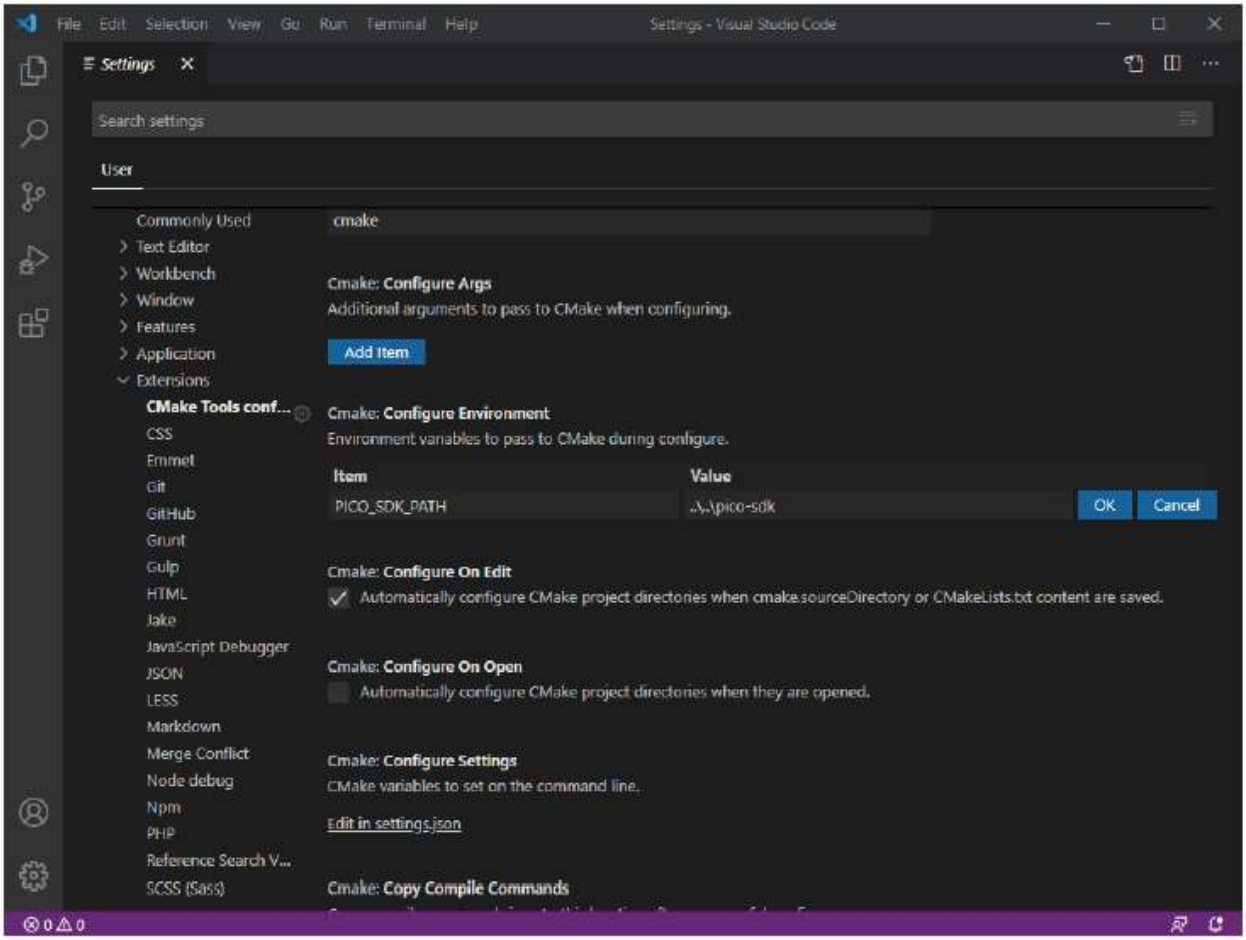


Рисунок 22. Установка PICO_SDK_PATH Переменной окружения в CMake Extension. Дополнительно вам нужно будет прокрутить вниз до "Cmake: Generator" и ввести в поле "NMake Makefiles".

⍰ ВАЖНО

Если вы не измените "Cmake: Generator", Visual Studio по умолчанию будет использовать ninja, и сборка может завершиться сбоем, поскольку GCC выводит информацию о зависимостях в слегка неправильном формате, который ninja не может понять.

Теперь закройте страницу настроек и перейдите в меню "Файл", нажмите "Открыть папку" и перейдите к репозиторию pico-examples и нажмите "Выбрать папку". Вам будет предложено настроить проект, см. рисунок 23. Выберите "GCC для arm-none-eabi" для вашего компилятора.

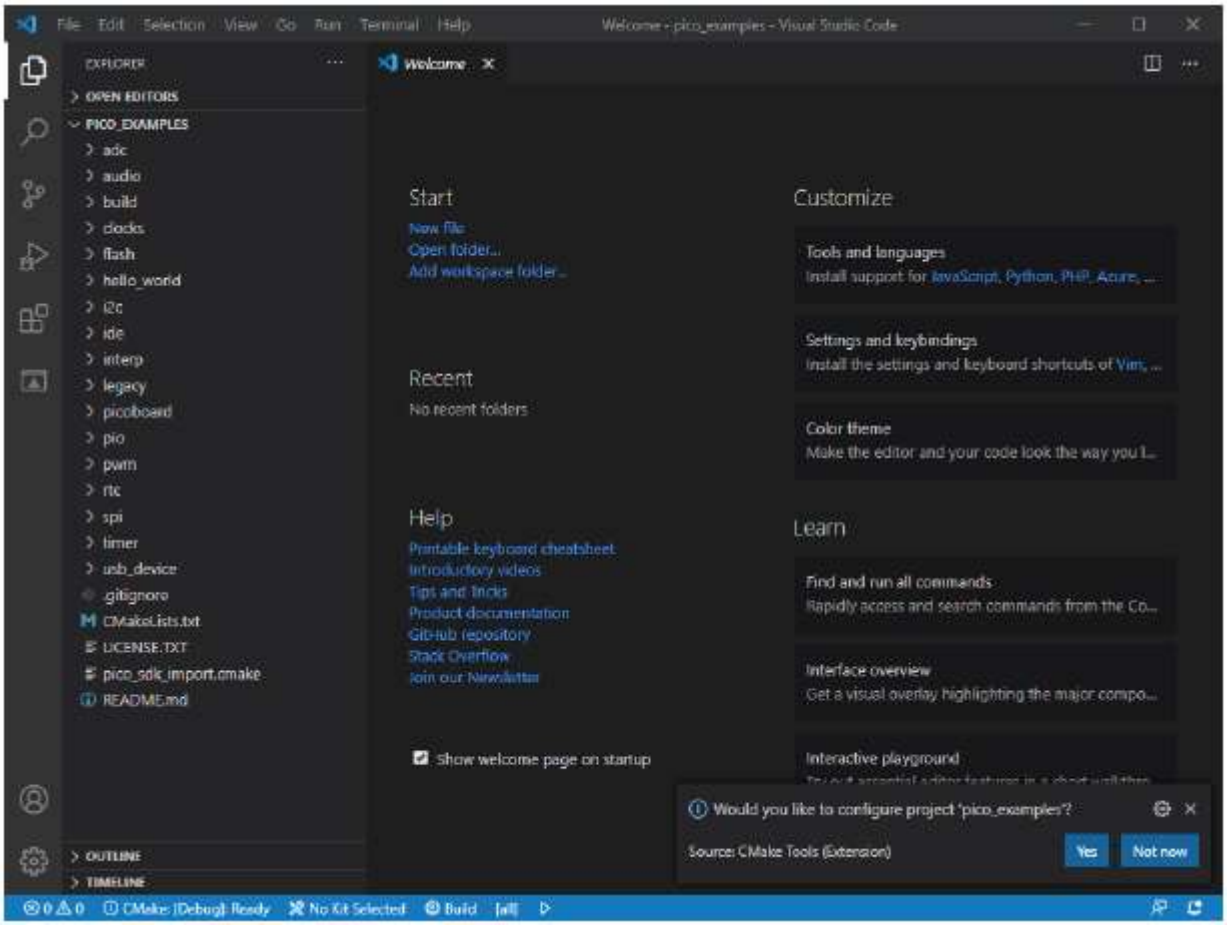


Рисунок 23. Запрос на настройку вашего проекта в Visual Studio Code.

Продолжайте и нажмите на кнопку "Build" (с зубчатым колесом) в синей нижней панели окна. Это создаст каталог сборки, запустит CMake и соберет примеры проекта, включая "Hello World".
Это приведет к созданию целевых объектов elf, bin и uf2, вы можете найти их в каталогах hello_world/serial и hello_world/usb внутри вновь созданного каталога сборки. Двоичные файлы UF2 можно перетаскивать непосредственно на плату RP2040, подключенную к вашему компьютеру с помощью USB.

9.2.2.9. Мигание и запуск "Hello World"

Подключите Raspberry Pi Pico к вашему Raspberry Pi с помощью кабеля micro-USB, обязательно удерживая нажатой кнопку BOOTSEL, чтобы перевести его в режим USB-накопителя. Плата должна автоматически отображаться как внешний диск. Теперь вы можете перетащить двоичный файл UF2 на внешний диск.
Raspberry Pi Pico перезагрузится, размонтирует себя как внешний диск и начнет запускать прошитый код.
Как мы делали в главе 4, вы можете создать пример Hello World с помощью stdio, маршрутизируемого либо на USB CDC (последовательный), либо на UART0 на выводах GP0 и GP1. Установка драйвера не требуется, если вы создаете с USB CDC в качестве целевого выхода, поскольку это устройство соответствует классу.

9.2.2.9.1. Вывод UART

В качестве альтернативы, если вы хотите подключиться к стандартному UART Raspberry Pi Pico, чтобы увидеть выходные данные, вам нужно будет подключить Raspberry Pi Pico к компьютеру с помощью последовательного преобразователя USB в UART, например SparkFun FTDI Базовая плата, смотрите рисунок 24.

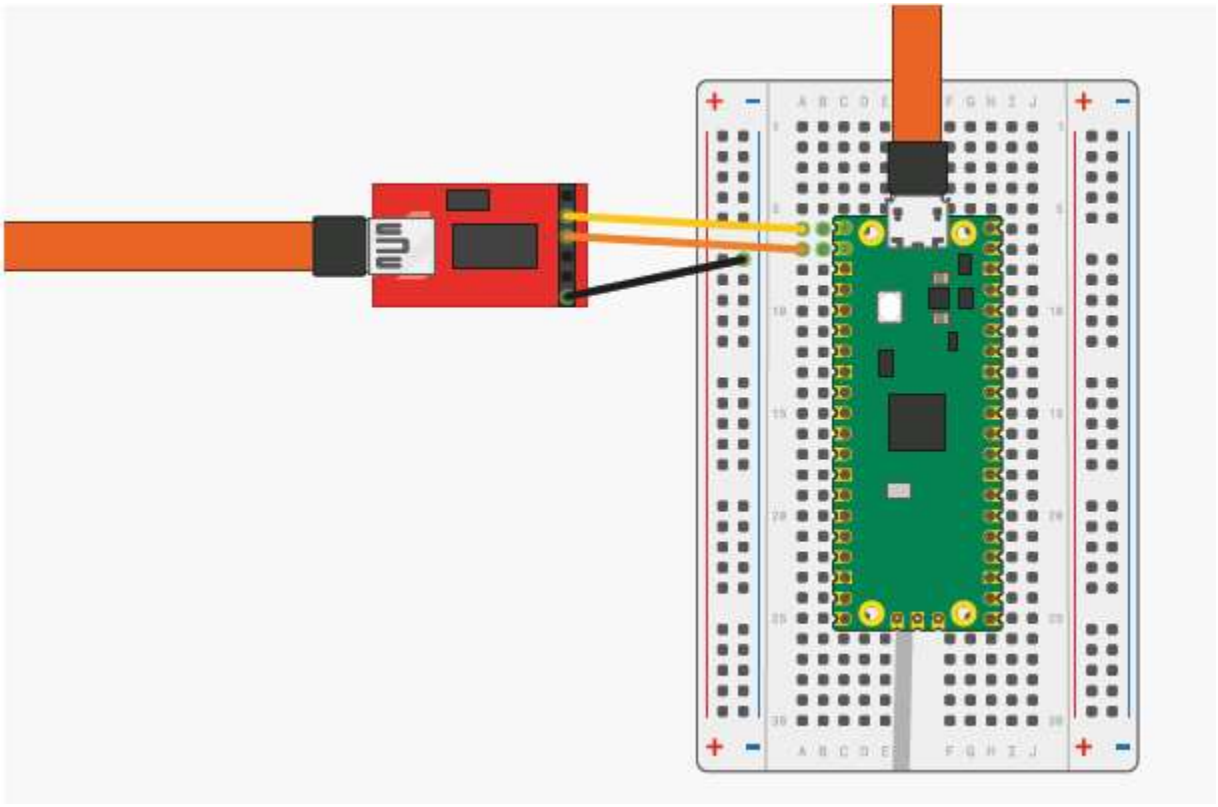


Рисунок 24. Sparkfun Базовый адаптер FTDI, подключенный к Raspberry Pi Pico

Пока вы используете последнюю версию Windows 10, соответствующие драйверы уже должны быть загружены. В противном случае обратитесь к веб-сайту производителя за драйверами микросхем FTDI.

Затем, если у вас их еще нет, скачайте и установите PuTTY. Запустите его и выберите "Последовательный", введите 115 200 в качестве скорости передачи данных в поле "Скорость" и последовательный порт, который использует ваш UART-конвертер. Если вы этого не знаете, вы можете выяснить это с помощью команды chgport,

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

это даст вам список активных последовательных портов. Здесь последовательный преобразователь USB в UART находится на COM5.

Примечание

Если у вас несколько последовательных устройств и вы не можете определить, какое из них является вашим последовательным преобразователем UART в USB, попробуйте отсоединить кабель и снова запустить chgport, чтобы увидеть, какой COM-порт исчезает.

После ввода скорости и порта нажмите кнопку "Открыть", и вы должны увидеть вывод UART с Raspberry Pi Пико в окне вашего терминала.

Глава 10. Использование других интегрированных Сред разработки

В настоящее время рекомендуемой интегрированной средой разработки (IDE) является Visual Studio Code, см. главу 7. Однако с RP2040 и Raspberry Pi Pico можно использовать и другие среды.

10.1. Использование Eclipse

Eclipse - это мультиплатформенная интегрированная среда разработки (IDE), доступная для x86 Linux, Windows и Mac. Кроме того, последняя версия теперь доступна для 64-разрядных ARM-систем и хорошо работает на Raspberry Pi 4/400. (4 ГБ и выше) под управлением 64-разрядной операционной системы. Следующие инструкции описывают, как настроить Eclipse на устройстве Linux для использования с Raspberry Pi Pico. Инструкции для других систем будут в целом аналогичными, хотя подключения к Raspberry Pi Pico будут отличаться. Смотрите раздел 9.2 и раздел 9.1 для получения более подробной информации о платформах, отличных от Linux.

10.1.1. Настройка Eclipse для Pico на компьютере с Linux

Предпосылки:

- Устройство под управлением последней версии Linux с объемом оперативной памяти не менее 4 ГБ
- 64-разрядная операционная система.
- CMake 3.11 или новее

Если вы используете Raspberry Pi, вам следует включить стандартный UART, добавив следующее к config.txt

```
enable_uart=1
```

Вам также следует установить OpenOCD и систему отладки SWD. Инструкции о том, как это сделать, приведены в главе 5.

10.1.1.1. Установка Eclipse и плагинов Eclipse

Установите последнюю версию Eclipse со встроенным CDT, используя стандартные инструкции. Если вы работаете на платформе ARM, вам нужно будет установить AArch64 (64-разрядную ARM) версию Eclipse. Все версии можно найти на Eclipse вебсайт. https://projects.eclipse.org/projects/iot.embed-cdt/downloads

Загрузите правильный файл для вашей системы и извлеките его. Затем вы можете запустить его, перейдя в то место, откуда он был извлечен , и запустив исполняемый файл 'eclipse'.

```
$ ./eclipse
```

Встроенная CDT-версия Eclipse включает в себя C/C++ development kit и Embedded development kit, поэтому в ней есть все необходимое для разработки для Raspberry Pi Pico.

10.1.1.2. Использование pico-примеров

Стандартной системой сборки для среды Pico является CMake. Однако Eclipse не использует CMake, поскольку у него есть своя собственная система сборки, поэтому нам нужно преобразовать сборку CMake из pico-примеров в проект Eclipse.

- На том же уровне, что и папка pico-examples, создайте новую папку, например pico-examples-eclipse
- Измените каталог на эту папку
- Укажите путь к PICO_SDK_PATH

```
$ export PICO_SDK_PATH=<wherever>
```

- В командной строке введите:

```
$ cmake -G"Eclipse CDT4 - Unix Makefiles" -DCMAKE_BUILD_TYPE=Debug ../pico-examples
```

Это создаст файлы проекта Eclipse в нашей папке pico-examples-eclipse, используя исходный код из исходного дерева CMake.Теперь вы можете загрузить свои новые файлы проекта в Eclipse, используя опцию Открыть проект из файловой системы в меню Файл.

10.1.1.3. Строительство(Building)

Щелкните правой кнопкой мыши на проекте в обозревателе проектов и выберите "Построить". Это приведет к созданию всех примеров.

10.1.1.4. OpenOCD

В этом примере используется система OpenOCD для взаимодействия с Raspberry Pi Pico. Перед запуском кода вам необходимо будет обеспечить 2-проводные отладочные подключения от хост-устройства к Raspberry Pi Pico. На Raspberry Pi это можно сделать с помощью подключений GPIO, но на ноутбуке или настольном устройстве вам потребуется использовать дополнительное оборудование для этого подключения. Один из способов - использовать второй Raspberry Pi Pico под управлением Picoprobe, который описан в приложении A. Подробнее инструкции по отладочным подключениям можно найти в главе 5. Как только OpenOCD установлен и установлено правильное подключение, Eclipse необходимо настроить для взаимодействия с OpenOCD при запуске программ. OpenOCD предоставляет интерфейс GDB для Eclipse, и именно этот интерфейс используется при отладке.

Чтобы настроить систему OpenOCD, выберите "Настройки" в меню "Окно". Нажмите на стрелку MCU, чтобы расширить параметры, и выберите Глобальный путь OpenOCD. Для получения исполняемого файла введите “openocd”.

Для папки выберите расположение в файловой системе, куда вы клонировали форк Pico OpenOCD с github.

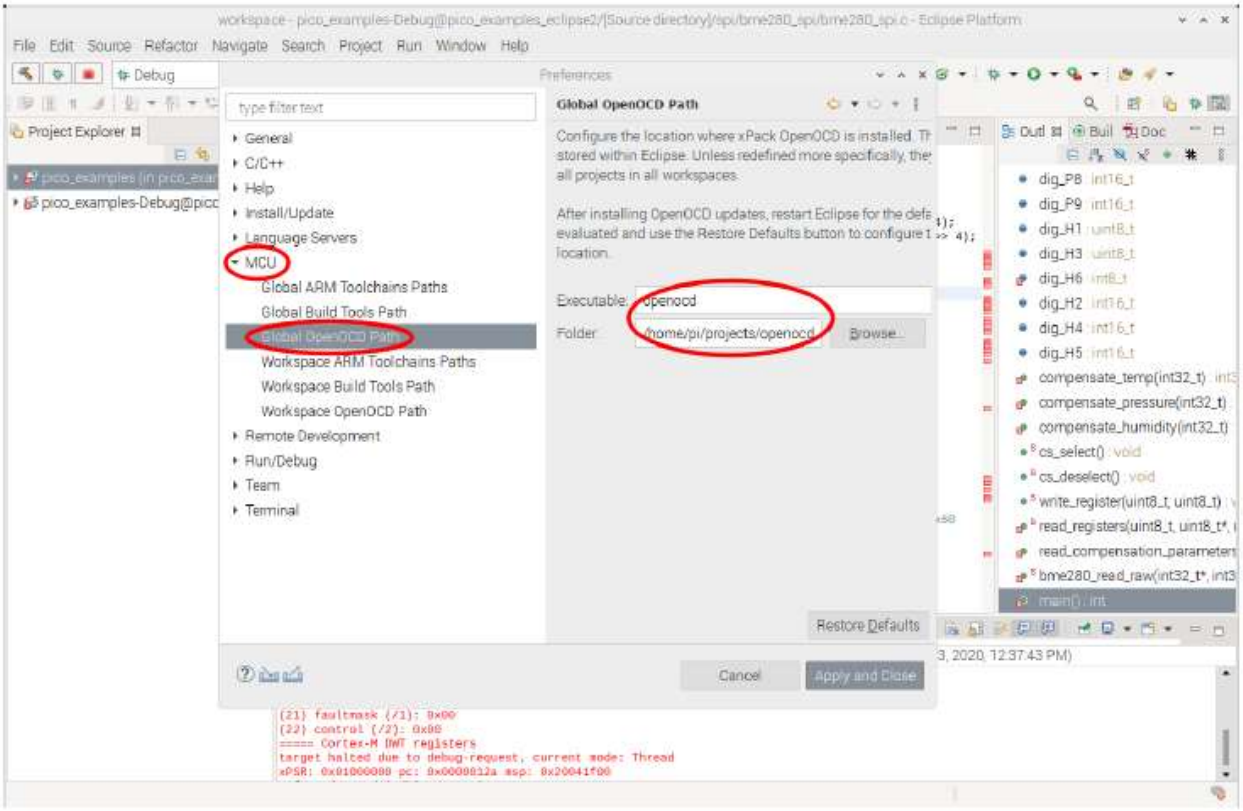


Рис. 25. Установка Исполняемый ОКР имя и путь в Eclipse.

10.1.1.5. Создание конфигурации запуска

Для запуска или отладки кода в Eclipse вам необходимо настроить конфигурацию запуска. При этом настраивается вся информация, необходимая для идентификации запускаемого кода, любые параметры, отладчик, пути к исходным кодам и информация SVD. В меню запуска Eclipse выберите пункт **Run Configurations**(Запустить конфигурации). Чтобы создать конфигурацию отладчика, выберите опцию отладки GDB OpenOCD , затем нажмите кнопку Создать конфигурацию.

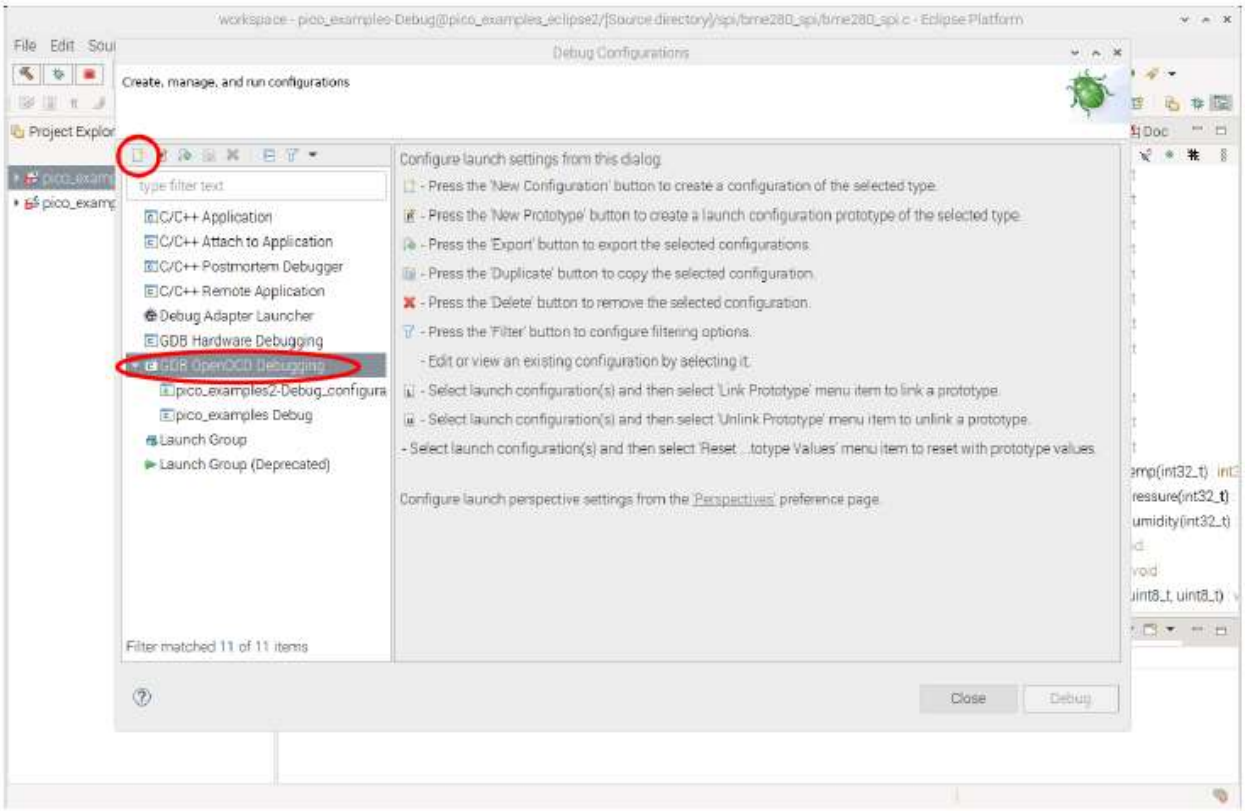


Рисунок 26. Создание новой конфигурации запуска/отладки в Eclipse

10.1.1.5.1. Настройка приложения для запуска

Поскольку сборка pico-examples создает множество различных исполняемых файлов приложений, вам необходимо выбрать, какой именно из них будет запущен или отлажен.

На главной вкладке страницы конфигурации запуска используйте опцию **Browse**(Обзор), чтобы выбрать приложения C/C++, которые вы хотите запустить. Сборка Eclipse создаст исполняемые файлы во вложенных папках папки проекта Eclipse. В нашем примере это

.../pico-examples-eclipse/<имя папки примера>/<необязательное имя вложенной папки примера>/исполняемый файл.elf

Так, например, если мы запустим пример мигания светодиода, это можно найти по адресу:

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

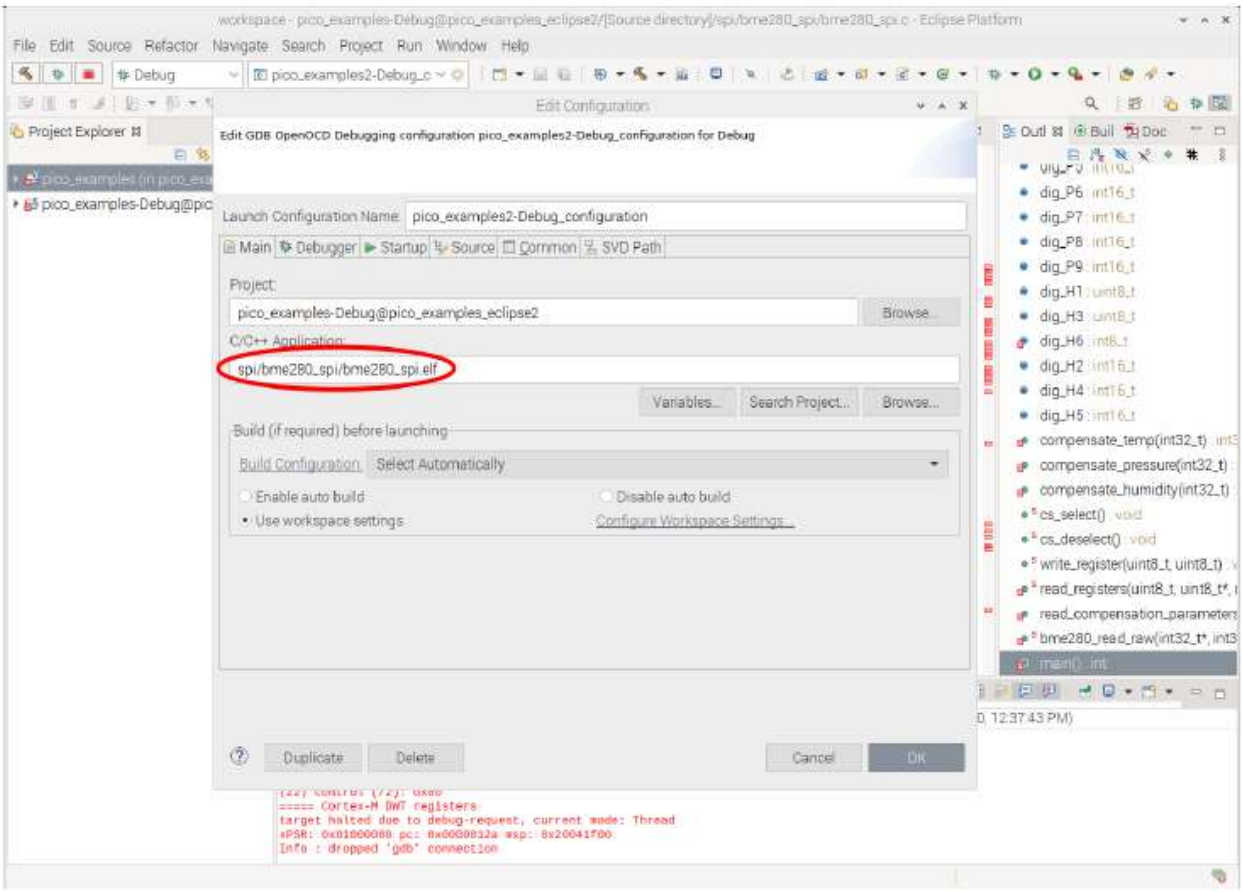


Рисунок 27. Настройка исполняемого файла для отладки в Eclipse.

10.1.1.5.2. Настройка отладчика

Мы используем OpenOCD для взаимодействия с Raspberry Pi Pico, поэтому нам нужно это настроить.
Установите openocd в поле `Executable box` и в поле `Actual Executable box` (Фактический исполняемый файл).
Нам также нужно настроить OpenOCD для использования конкретной конфигурации Pico, поэтому в разделах параметров конфигурации добавьте следующее. Обратите внимание, что вам нужно будет изменить путь, чтобы он указывал на местоположение, в котором установлена версия OpenOCD для Pico.

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

Все остальные настройки OpenOCD должны быть установлены на значения по умолчанию.
Фактически используется отладчик GDB. Это взаимодействует с отладчиком OpenOCD для фактической связи с Raspberry Pi Pico, но предоставляет стандартный интерфейс для IDE.
Конкретная используемая версия GDB - "gdb-multiarch", поэтому введите это в поля "Имя исполняемого файла" и "Фактический исполняемый файл".

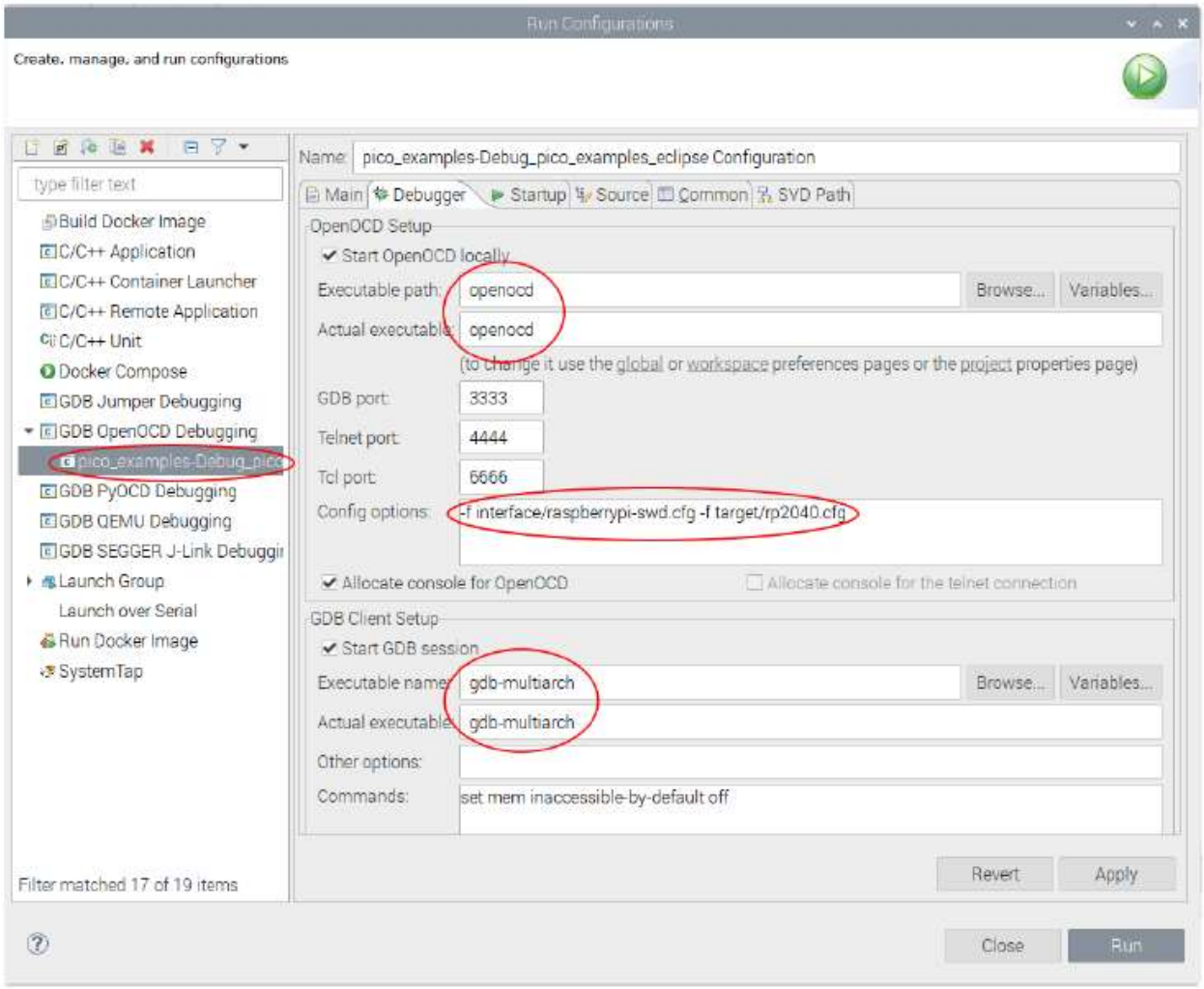


Рисунок 28. Настройка отладчика и OpenOCD в Eclipse.

10.1.1.5.3. Настройка плагина SVD

SVD предоставляет механизм для просмотра и настройки периферийных регистров на плате Pico. SVG-файл содержит расположение регистров и описания, а плагин SVG для Eclipse интегрирует эту функциональность в Eclipse IDE. Плагин SVD поставляется, как часть встроенных плагинов для разработки. Выберите вкладку Путь к SVG в конфигурации запуска и введите местоположение в файловой системе, где находится файл SVD. Обычно это можно найти в дереве исходных текстов pico-sdk.

Напр.

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

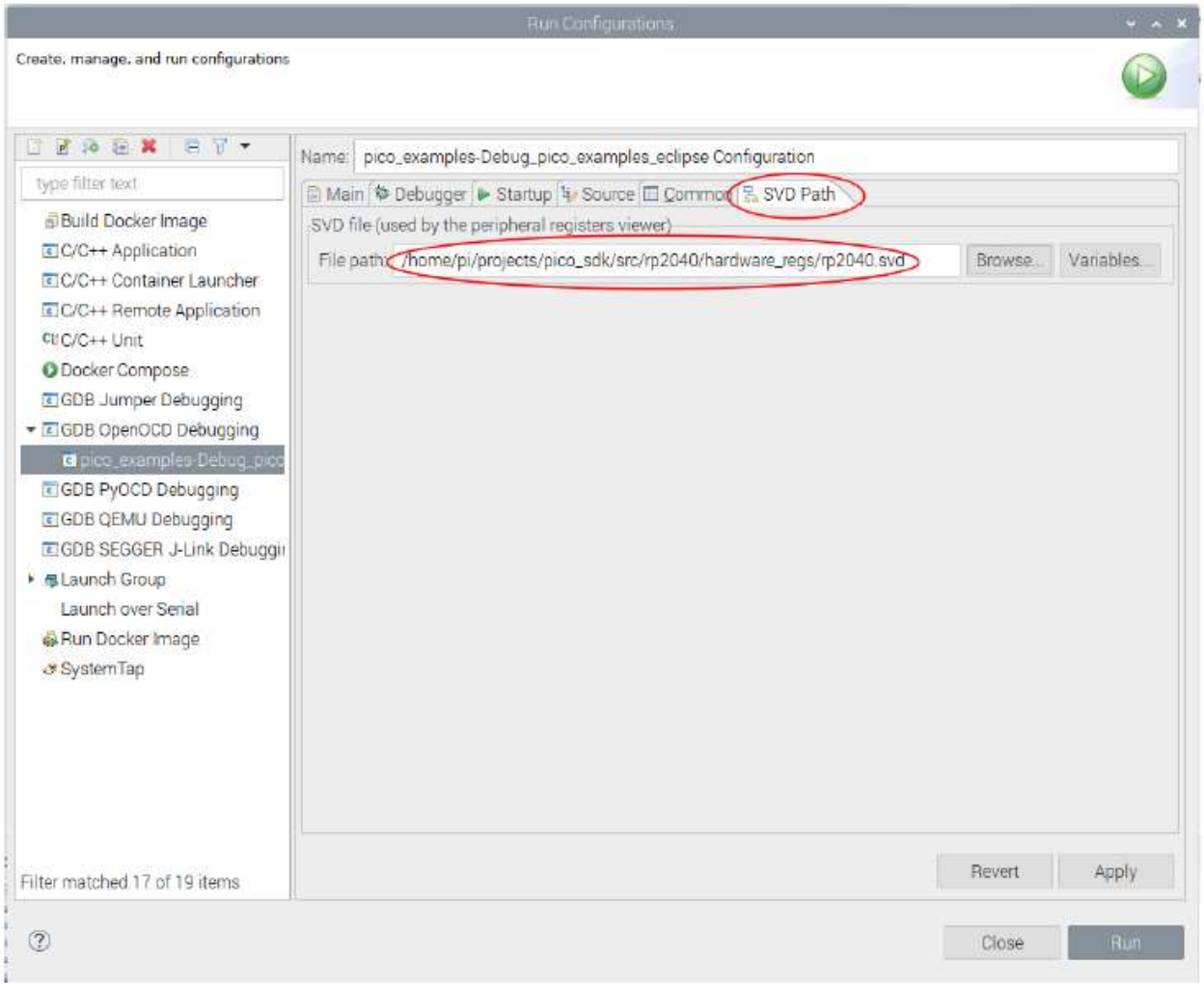


Рисунок 29. Установка пути SVD в Eclipse.

10.1.1.5.4. Запуск отладчика

Как только конфигурация запуска будет завершена и сохранена, вы можете немедленно запустить ее, нажав кнопку RUN(Выполнить) в правом нижнем углу диалогового окна, или просто применить изменения(Apple) и закрыть диалоговое окно. Затем вы можете запустить приложение, используя опцию debug в меню "RUN".
Это установит Eclipse в режим debug perspective, который отобразит множество различных окон отладки и исходного кода, а также очень полезное представление Peripherals, которое использует данные SVD для обеспечения доступа к периферийным регистрам. С этого момента это стандартный сеанс отладки Eclipse.

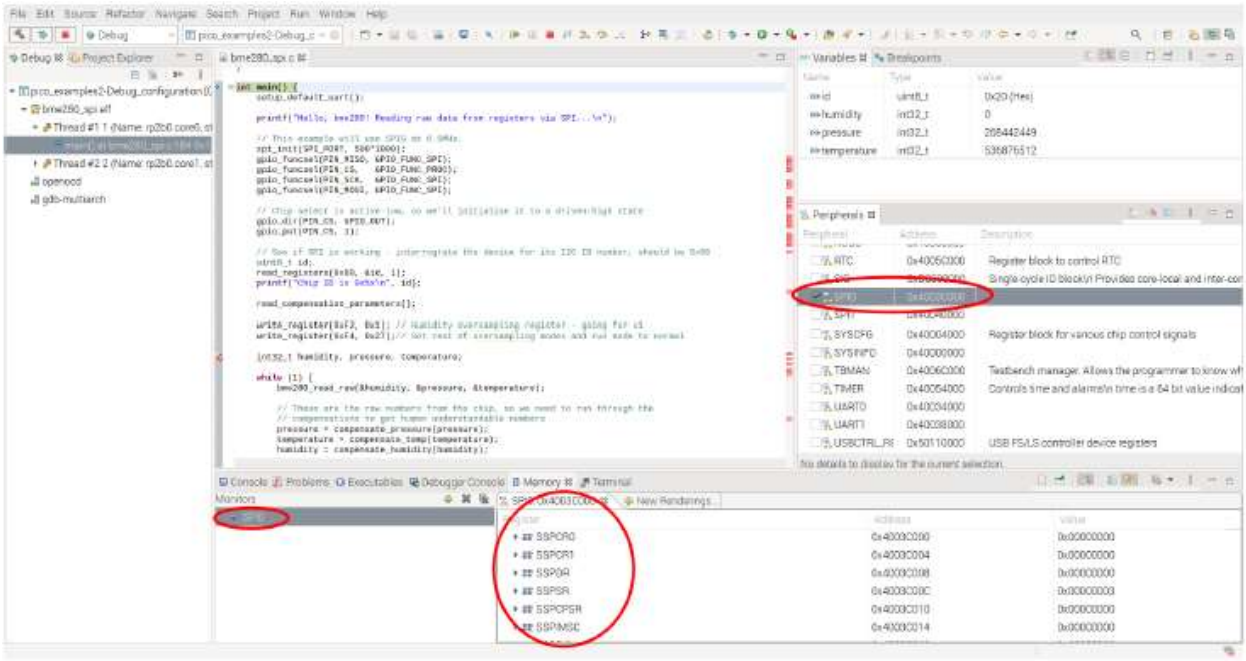


Рисунок 30. Запущенный отладчик Eclipse, показывающий некоторые доступные окна отладки.

10.2. Использование CLion

CLion - это мультиплатформенная интегрированная среда разработки (IDE) от JetBrains, доступная для Linux, Windows и Mac. Это коммерческая среда разработки, которую часто выбирают профессиональные разработчики (или те, кто любит JetBrains IDE), хотя доступны бесплатные лицензии или лицензии по сниженной цене. Он будет работать на Raspberry Pi, однако производительность не идеальна, поэтому ожидается , что вы будете использовать CLion на своем настольном компьютере или ноутбуке.
В то время как настройка проектов, разработка и компоновка - это несложно, настройка debug на данный момент все еще не очень распространена, так что имейте в виду.

10.2.1. Настройка CLion

Если вы планируете использовать CLion, мы предполагаем, что он у вас либо установлен, либо вы можете установить его из https://www.jetbrains.com/clion/

10.2.1.1. Настройка проекта

Здесь мы используем pico-examples в качестве примера проекта. Чтобы открыть проект pico-examples, выберите Открыть... в меню Файл, а затем перейдите в каталог pico- examples, который вы проверили, и выберите его, и нажмите OK.
После открытия вы увидите что-то вроде рисунка 31.

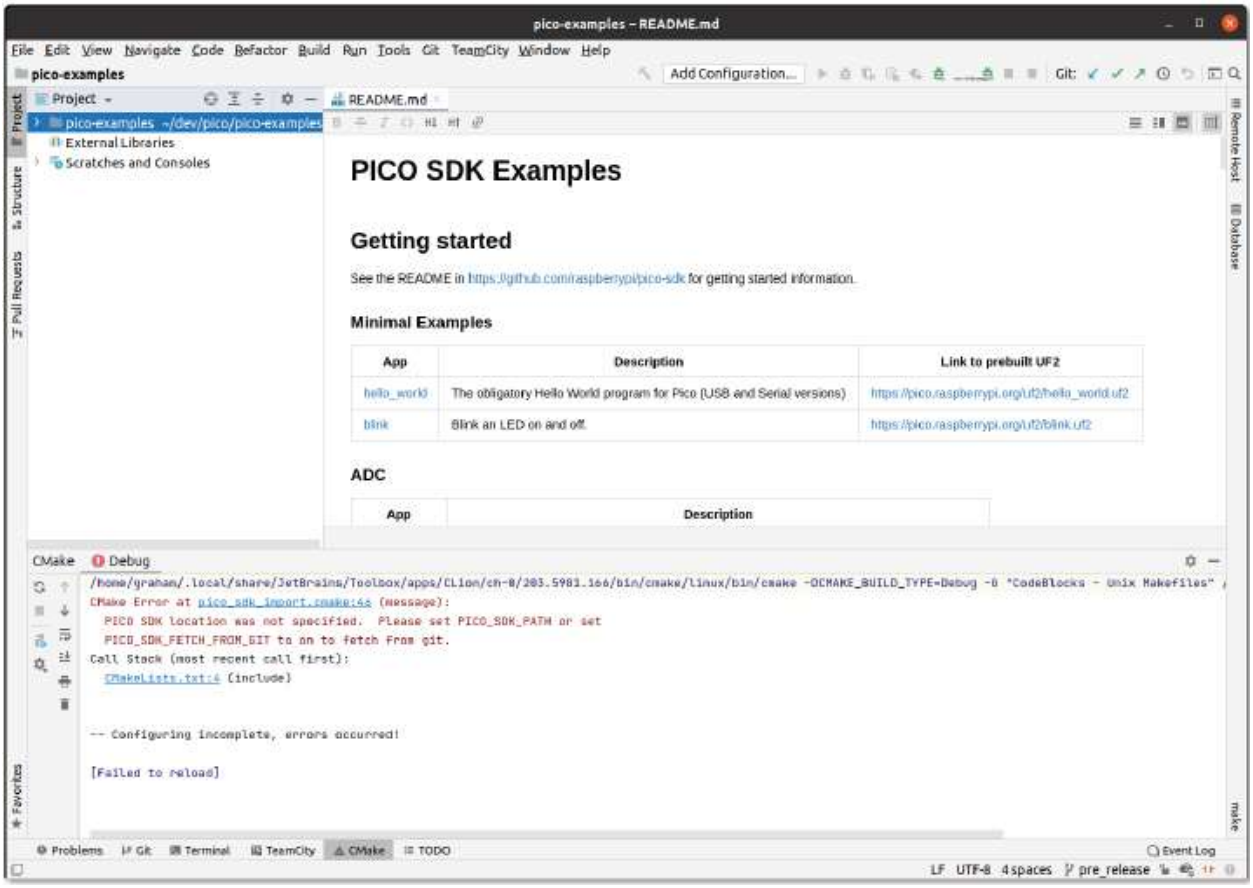


Рисунок 31. Недавно открытый проект CLion picoexamples.

Обратите внимание внизу, что CLion попытался загрузить проект CMake, но произошла ошибка, а именно, что мы не указали PICO_SDK_PATH

10.2.1.1.1. Настройка профилей CMake

Выберите "Настройки..." в меню "Файл", а затем перейдите и выберите "CMake" в разделе "Сборка, выполнение, развертывание". Вы можете установить переменную окружения PICO_SDK_PATH в разделе Environment: как показано на рисунке 32, или вы можете установить ее следующим образом -DPICO_SDK_PATH=xxx в параметрах CMake:. Они точно такие же, как переменные окружения или аргументы командной строки, когда вызываем cmake из командной строки, так что именно здесь вы должны указать параметры CMake, такие как PICO_BOARD, PICO_TOOLCHAIN_PATH и т.д.

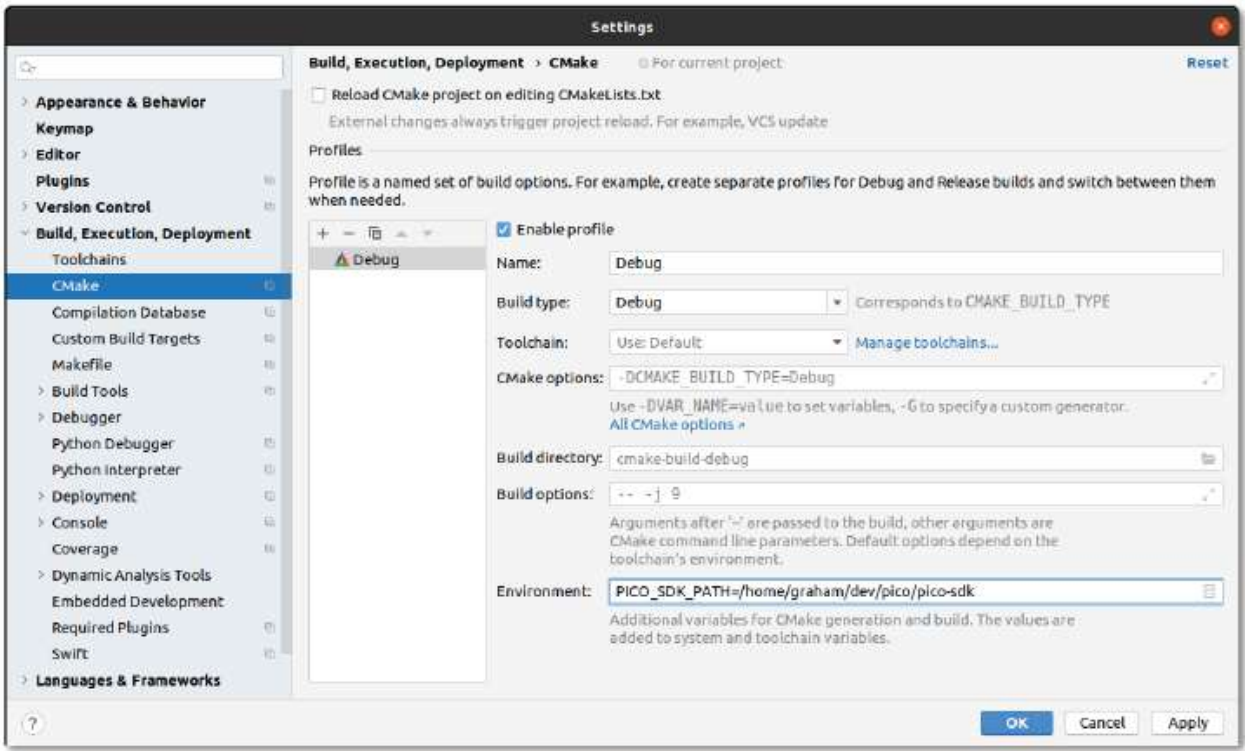


Рисунок 32. Настройка профиля CMake в CLion.

У вас может быть столько профилей CMake, сколько вам нравится, с различными настройками. Вероятно, вы захотите добавить сборку релиза, нажав кнопку +, а затем снова заполнив PICO_SDK_PATH, или нажав вторую кнопку копирования справа и исправив название и настройки (см. рис. 33).

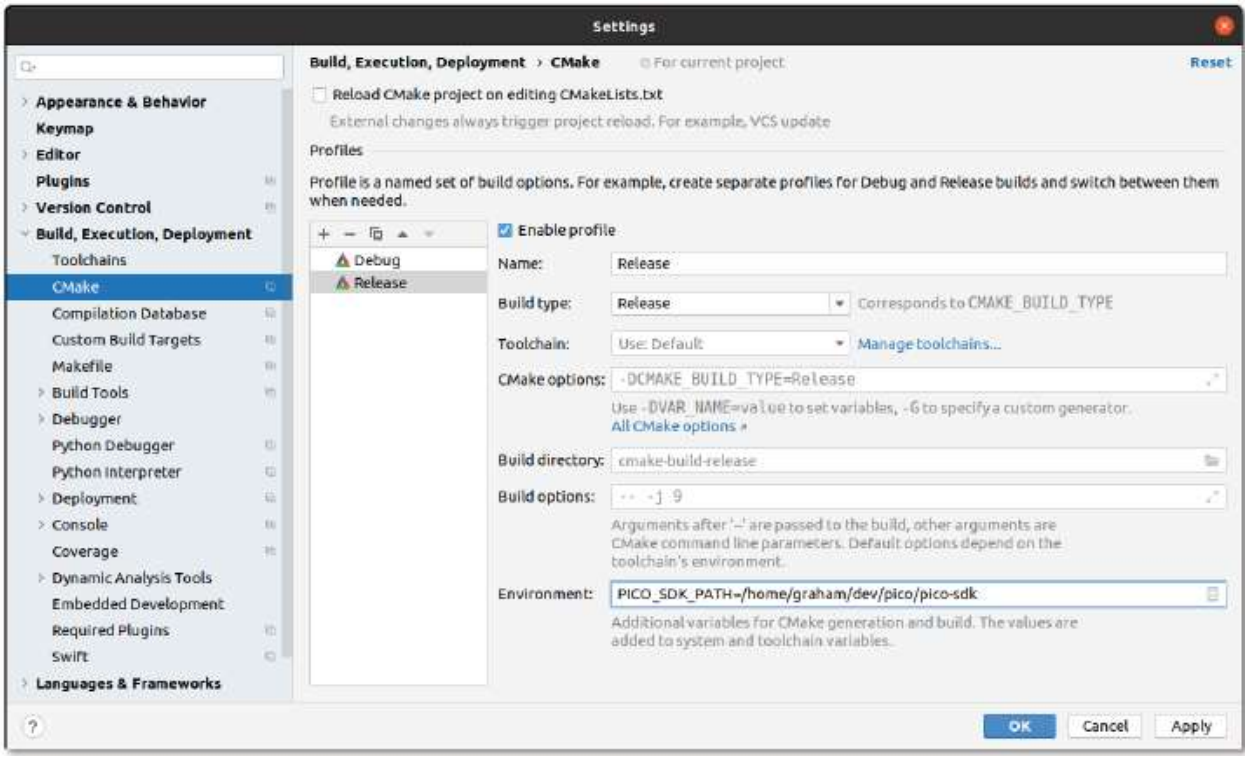


Рисунок 33. Настройка второго профиля CMake в CLion.

После нажатия кнопки ОК вы увидите что-то вроде рисунка 34. Обратите внимание, что для двух профилей есть две вкладки (Debug и Release) в нижней части окна. В этом случае выбран Release, и вы можете видеть, что настройка CMake прошла успешно.

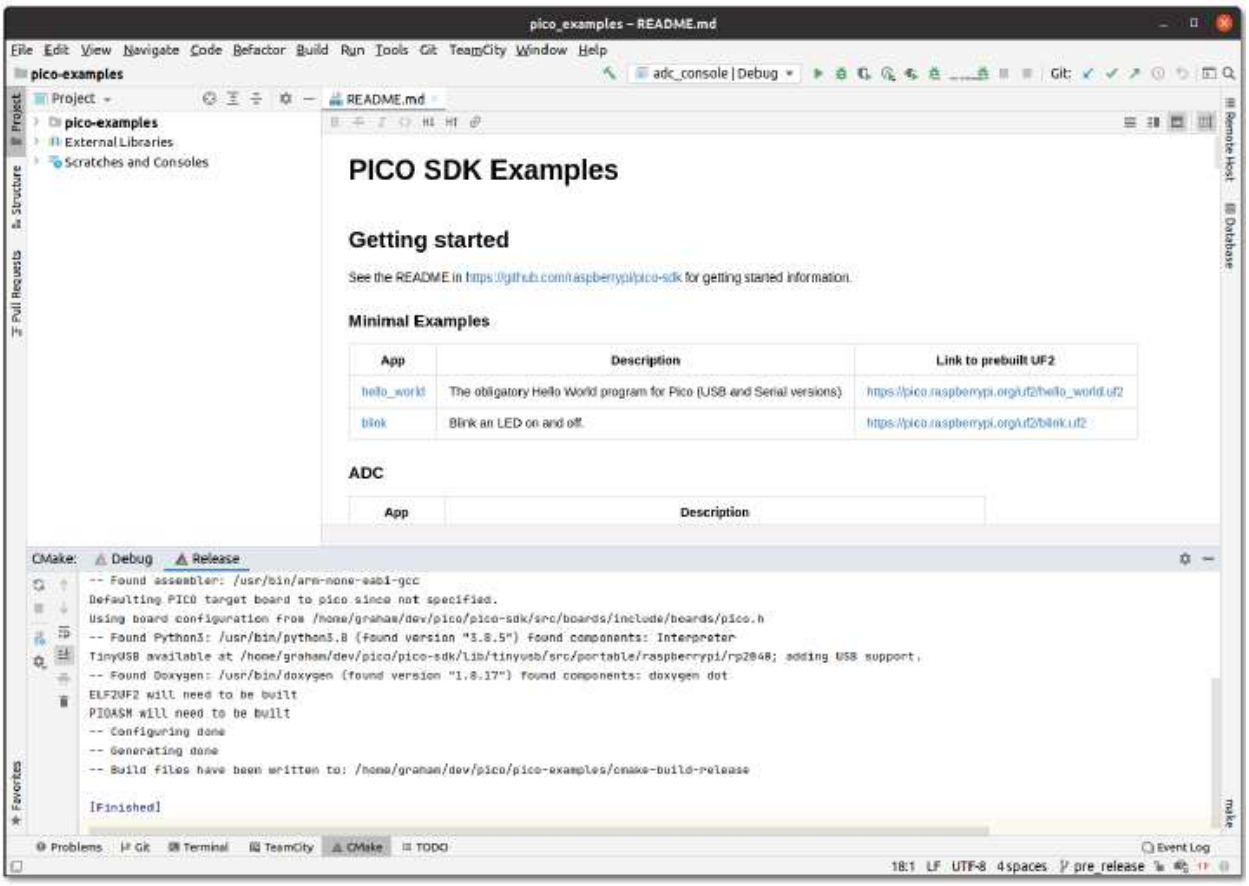


Рисунок 34. Настройка второго профиля CMake в CLion.

10.2.1.1.2. Запуск сборки

Теперь мы можем выбрать для сборки одну или несколько целей. Например, вы можете перейти к выпадающему списку в середине панели инструментов и выберите или начните вводить hello_usb; затем нажмите на значок инструмента слева от него, чтобы выполнить построение (см. рис. 35). В качестве альтернативы вы можете выполнить полную сборку всех целевых объектов или другие типы сборок из меню Сборки.

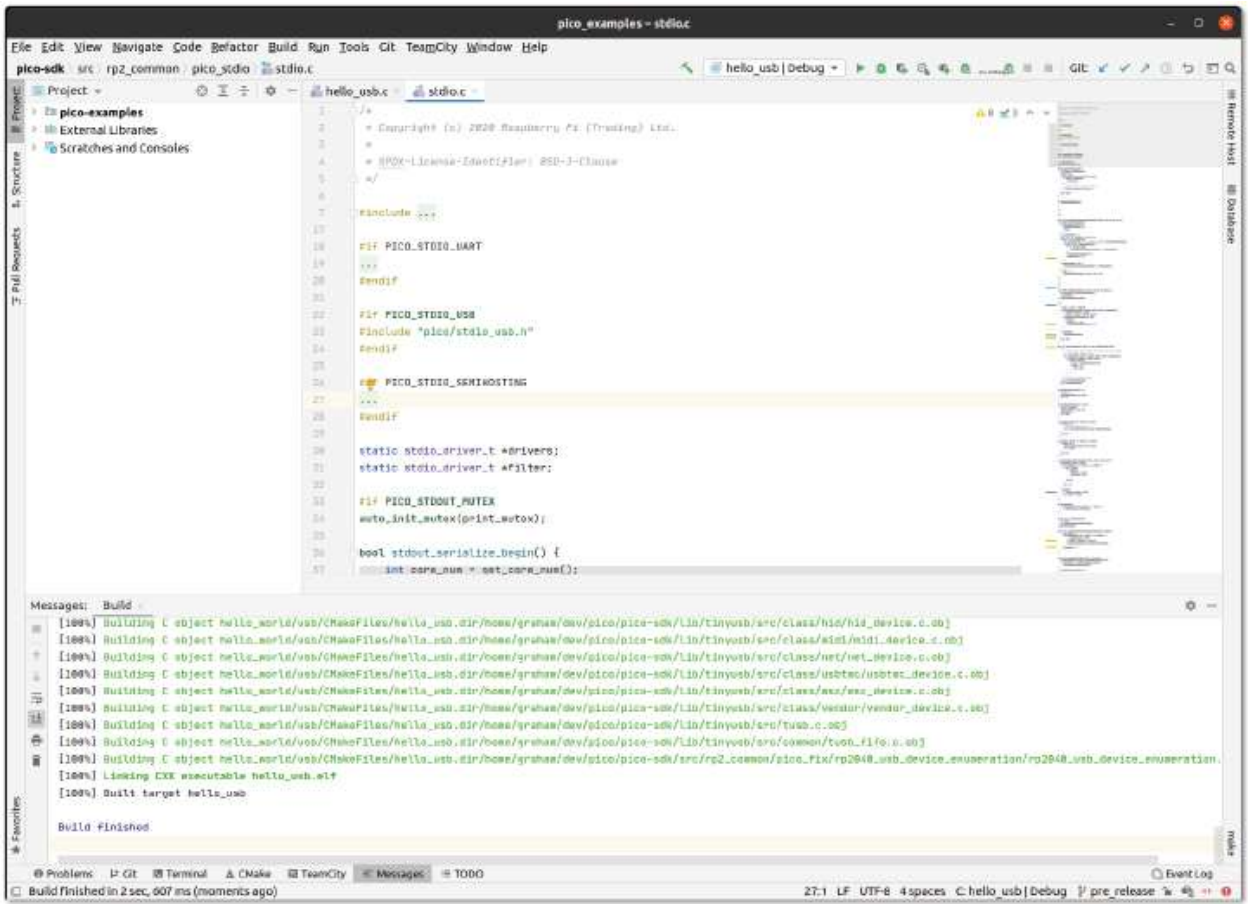


Рисунок 35. hello_usb успешно собран.

Обратите внимание, что выпадающий список позволяет вам выбрать как целевой объект, который вы хотите создать, так и профиль CMake для использования (в данном случае один из Debug или Release).

Еще одна вещь, которую вы заметите на рисунке 35, - это то, что в нижней строке состояния вы можете увидеть hello_usb и снова выполнить отладку. Они показывают вам целевой профиль и профиль CMake, используемые для управления подсветкой синтаксиса и т.д. в редакторе (это было выбрано автоматически, когда вы выбрали hello_usb ранее). Вы можете визуально увидеть в файле stdio.c, который был открыт пользователь, что PICO_STDIO_USB установлен, но PICO_STDIO_UART - нет (которые являются частью конфигурации hello_usb).

Время сборки для каждой двоичной конфигурации библиотек активно используется в SDK, так что это очень приятная функция.

10.2.1.1.3. Создавать артефакты

Артефакты сборки расположены в разделе cmake-build-<профиль> в корневом каталоге проекта (см. рис. 36). В данном случае это каталог cmake-build-debug.

Файл UF2 можно скопировать на устройство RP2040 в режиме загрузки, или ELF можно использовать для отладки.

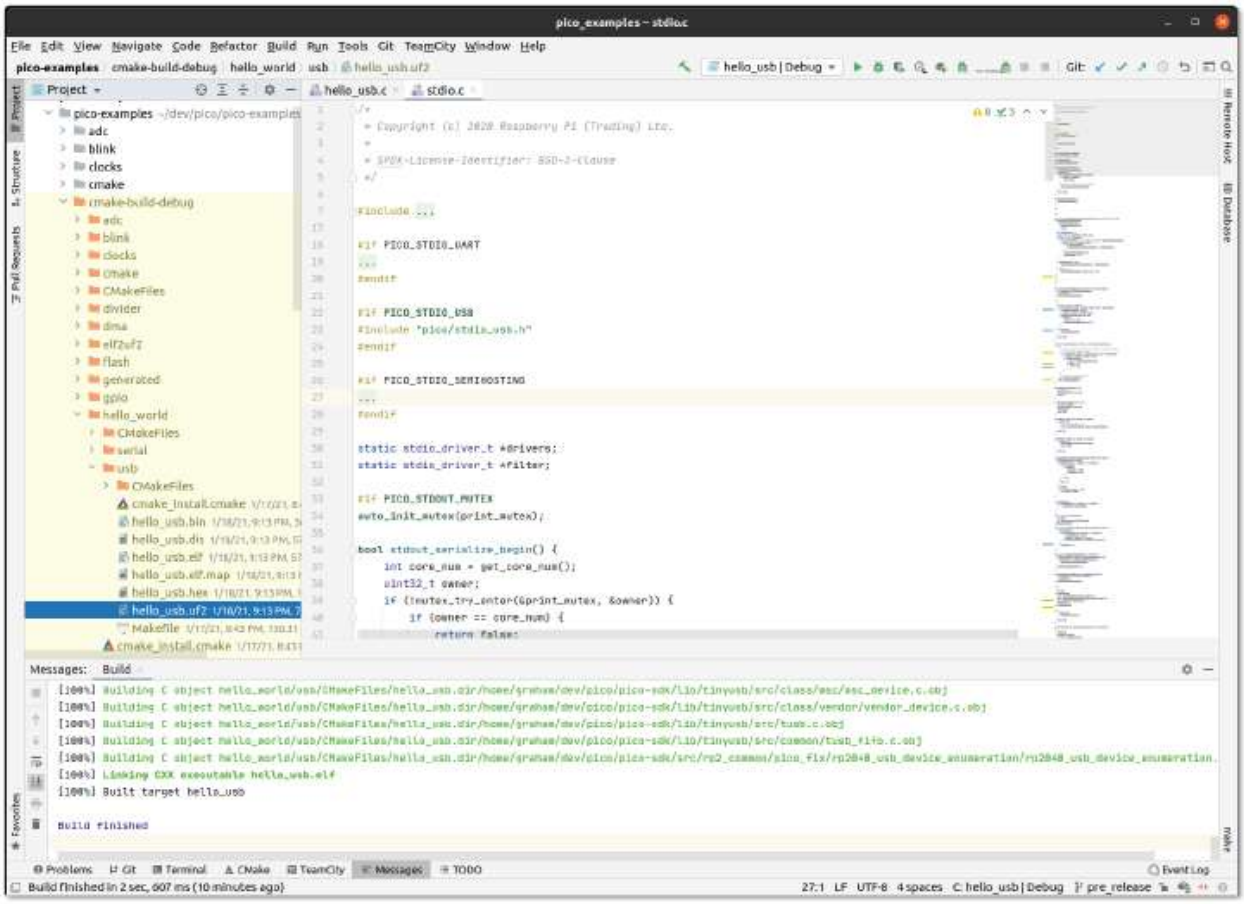


Рисунок 36. Поиск артефактов сборки hello_usb

10.3. Другие среды

Доступно множество сред разработки, и мы не можем описать их все здесь, но вы сможете использовать многие из них с SDK. В вашей среде IDE необходим ряд функций, которые сделают возможной поддержку Raspberry Pi Pico:

- Интеграция с CMake
- Поддержка GDB с удаленными опциями
- SVD. Необязательно, но значительно упрощает считывание состояния периферийных устройств
- Дополнительный модуль разработки ARM embedded. Плагины такого типа часто значительно упрощают поддержку.

10.3.1. Использование openocd-svd

Инструмент openocd-svd - это утилита с графическим интерфейсом на основе Python, которая предоставляет вам доступ к периферийным регистрам ARM MCU через комбинацию OpenOCD и CMSIS-SVD.
Чтобы установить его, вы должны сначала установить зависимости,

```
$ sudo apt install python3-pyqt5
$ pip3 install -U cmsis-svd
```

перед клонированием репозитория git openocd-svd.

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

Убедившись, что ваши Raspberry Pi 4 и Raspberry Pi Pico правильно подключены друг к другу, мы можем подключить OpenOCD к чипу с помощью конфигураций swd и rp2040.

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

⍰ Предупреждения
Если в вашей флэш-памяти есть код бездействующего режима или любой другой код, который останавливает системные часы, отладчику не удастся подключиться, поскольку системные часы остановлены. Хотя это может выглядеть как "замурованная" плата, вы можете без проблем вернуться в режим загрузки с помощью кнопки.

Этот терминал OpenOCD необходимо оставить открытым. Итак, продолжайте и откройте другой терминал, в этом мы прикрепим экземпляр gdb к OpenOCD.

Перейдите к своему проекту и запустите gdb,

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

Подключите GDB к OpenOCD,

```
(gdb) выберите удаленный локальный хост:3333,
```

загрузите его во flash и запустите его.

```
(gdb) load(загрузка)
(gdb) monitor reset init(сброс настроек монитора)
(gdb) continue(продолжить)
```

При запущенных openocd и gdb откройте третье окно и запустите openocd-svd, указав в нем на файл SVD в SDK.

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

Это откроет окно openocd-svd. Теперь перейдите в меню "Файл" и нажмите "Подключить OpenOCD", чтобы подключиться через telnet к запущенному экземпляру openocd.
Это позволит вам проверить регистры кода, запущенного на вашем Raspberry Pi Pico, см. рис. 37.

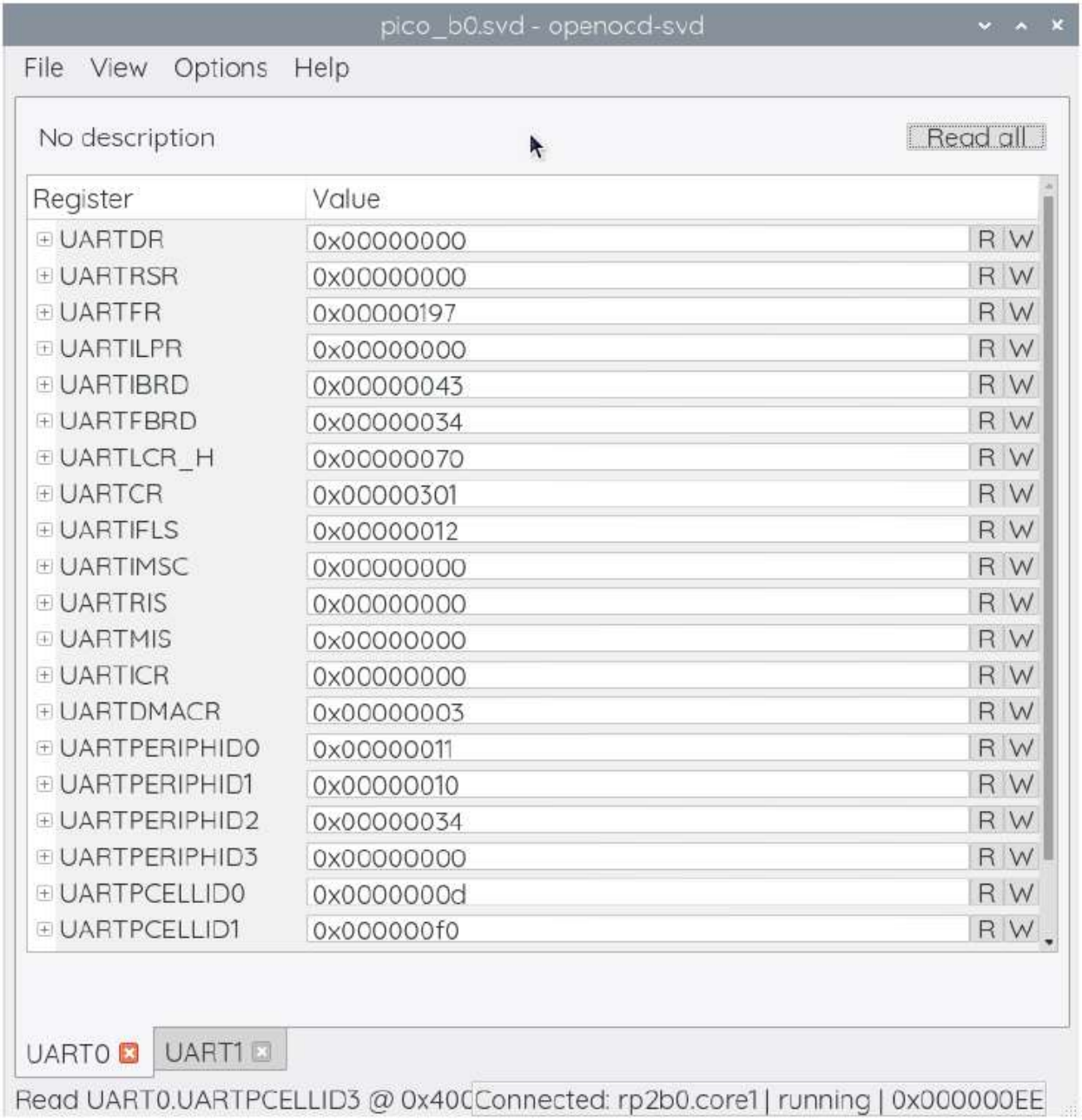


Рисунок 37. OpenOCD SVD запущен и подключен к Raspberry Pi Pico.

Приложение A: Использование пикозонда

Один Raspberry Pi Pico можно использовать для перепрограммирования и отладки другого, используя встроенное ПО picoprobe, которое преобразует подключение Pico к USB → SWD и UART-мосту. Это упрощает использование Raspberry Pi Pico на платформах, отличных от Raspberry Pi , таких как компьютеры Windows, Mac и Linux, где у вас нет GPIO для прямого подключения к UART или SWD, но есть USB-порт.

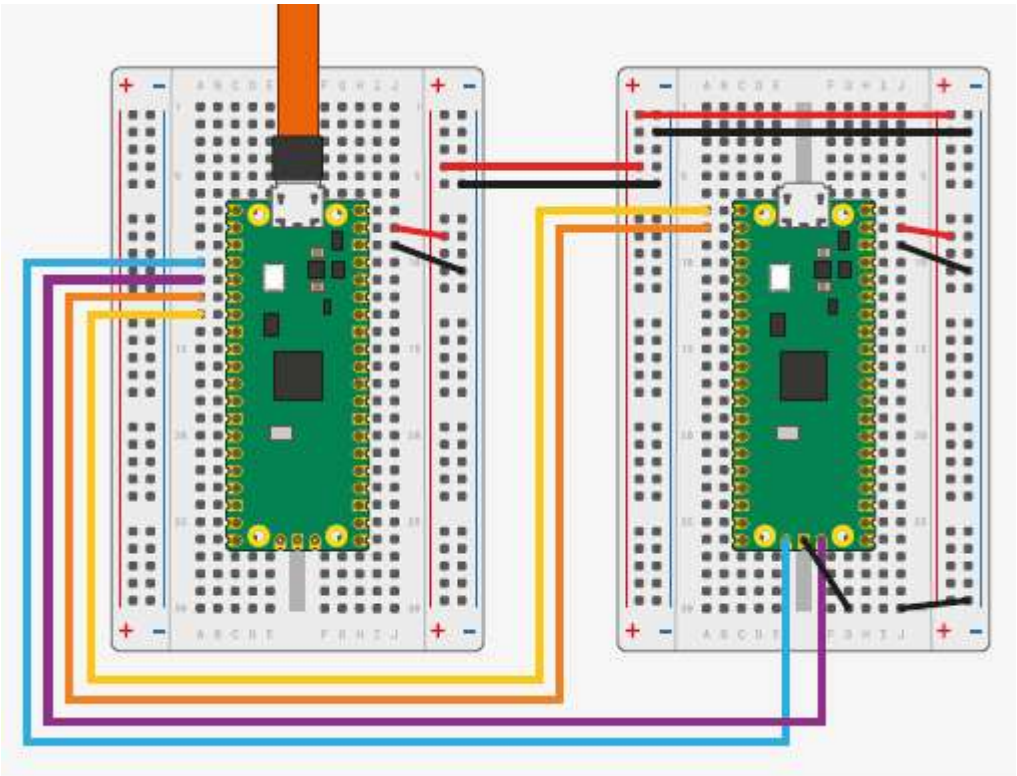


Рисунок 38. Подключение между Pico A (слева) и Pico B (справа) с помощью Пико A, выступающий в качестве отладочного зонда. По крайней мере, заземление и два провода SWD должны быть подключены, для одного Pico, чтобы иметь возможность перепрограммировать и отлаживать другой. На этой схеме также показано, как можно подключить последовательный порт UART, чтобы вы могли видеть последовательный выход UART тестируемого устройства Pico, и как можно соединить источник питания, чтобы обе платы питались от одного USB-кабеля. Подробнее в разделе Подключение пикозонда. Отладочный зонд Raspberry Pi Если вы не хотите использовать второй Raspberry Pi Pico для отладки, мы теперь также производим Raspberry Pi Debug Probe. Raspberry Pi Debug Probe предоставляет: отладку по последовательному проводу (SWD) и универсальный USBto- Последовательный мост.

Несмотря на то, что он был разработан с учетом Raspberry Pi Pico и других целевых программ на базе RP2040, Raspberry Pi Debug Probe можно использовать для отладки любого микроконтроллера на базе Arm, который предоставляет SWD -порт с вводом-выводом 3V3. Полную документацию и дополнительную информацию об отладочном тесте можно найти онлайн на сайте документации.

Создание OpenOCD

Чтобы picoprobe заработал, вам необходимо собрать OpenOCD из репозитория Raspberry Pi на GitHub.

ПРИМЕЧАНИЕ

Предыдущие версии микропрограммы picoprobe требовали специального драйвера протокола OpenOCD, в этом больше нет необходимости.

Линукс

```
$ 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 rp2040 --depth=1
$ cd openocd
$ ./bootstrap
$ ./configure ①
$ make -j4
$ sudo make install
```

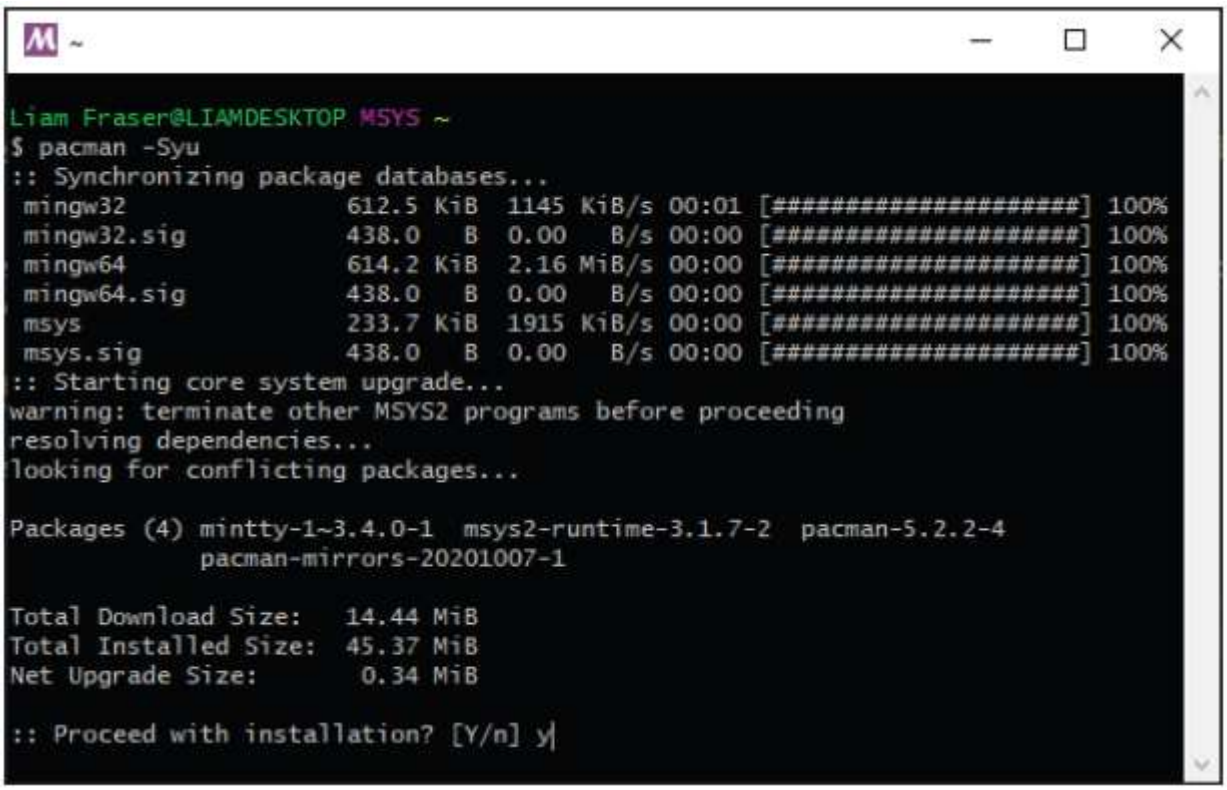
1. Если вы создаете на Raspberry Pi, вы также можете передать --enable-sysfs gpio --enable-bcm2835 gpio, чтобы разрешить изменение битов SWD через контакты GPIO.

⍰ ПРИМЕЧАНИЕ
Пользователям Ubuntu и Debian может дополнительно потребоваться также установить **pkg-config**.

Windows

Чтобы максимально упростить создание OpenOCD, мы будем использовать MSYS2. Цитирую их веб-сайт: "MSYS2 - это набор инструментов и библиотек, предоставляющих вам простую в использовании среду для создания, установки и запуска собственного программного обеспечения Windows".
Загрузите и запустите программу установки с сайта https://www.msys2.org/.
Начните с обновления базы данных пакетов и основных системных пакетов с помощью:

$ pacman –Syu



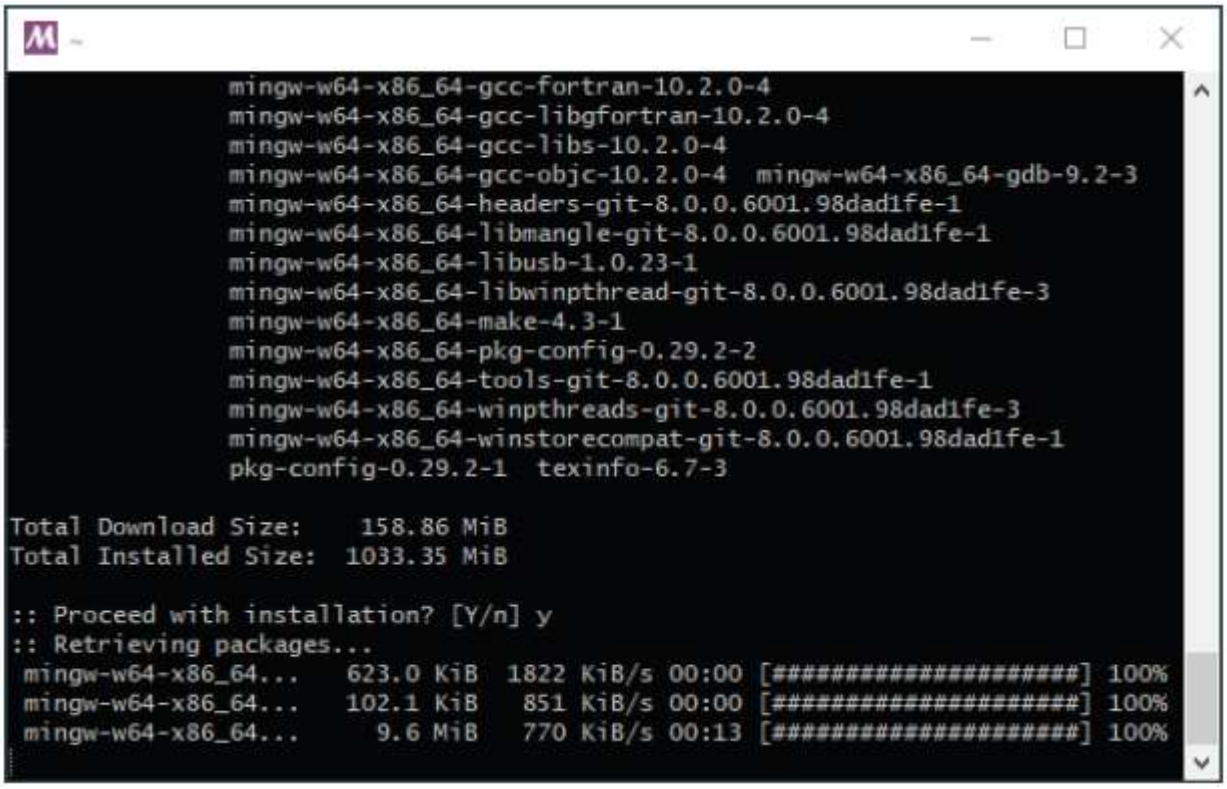
Если MSYS2 закроется, запустите его снова (убедившись, что вы выбрали 64-разрядную версию) и запустите

```
$ pacman -Su
```

для завершения обновления.
Установите необходимые зависимости:

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

Выберите все при установке набора инструментов mingw-w64-x86_64, нажав клавишу enter.



Закройте MSYS2 и снова откройте 64-разрядную версию, чтобы убедиться, что среда поддерживает GCC.

```
$ git clone https://github.com/raspberrypi/openocd.git --branch rp2040 --depth=1
$ cd openocd
$ ./bootstrap
$ ./configure --disable-werror ①
$ make -j4
```

1. К сожалению, disable-werror необходим, потому что не все компилируется чисто в Windows

Наконец, запустите OpenOCD, чтобы проверить, что он собран правильно. Ожидайте, что он выдаст ошибку, поскольку параметры конфигурации не были переданы.

```
$ src/openocd.exe
Open On-Chip Debugger 0.10.0+dev-gc231502-dirty (2020-10-14-14:37)
Licensed under GNU GPL v2
For bug reports, read
http://openocd.org/doc/doxygen/bugs.html
embedded:startup.tcl:56: Error: Can't find openocd.cfg
in procedure 'script'
at file "embedded:startup.tcl", line 56
```

```
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Error: Debug Adapter has to be specified, see "interface" command
embedded:startup.tcl:56: Error:
in procedure 'script'
at file "embedded:startup.tcl", line 56
```

Мак

При необходимости установите brew

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

Установить зависимости

```
$ brew install libtool automake libusb wget pkg-config gcc texinfo ①
```

1. Версия texinfo, поставляемая с OSX, ниже версии, необходимой для создания документов OpenOCD

```
$ cd ~/pico
$ git clone https://github.com/raspberrypi/openocd.git --branch rp2040 --depth=1
$ cd openocd
$ export PATH="/usr/local/opt/texinfo/bin:$PATH" ①
$ ./bootstrap
$ ./configure --disable-werror ②
$ make -j4
```

1.Установите более новую версию texinfo по пути
2. К сожалению, disable-werror необходим, потому что не все компилируется чисто в OSX
Проверьте, запущен ли OpenOCD. Ожидайте, что он выдаст ошибку, поскольку параметры конфигурации не были переданы.

```
$ src/openocd
Open On-Chip Debugger 0.10.0+dev-gc231502-dirty (2020-10-15-07:48)
Licensed under GNU GPL v2
For bug reports, read
http://openocd.org/doc/doxygen/bugs.html
embedded:startup.tcl:56: Error: Can't find openocd.cfg
in procedure 'script'
at file "embedded:startup.tcl", line 56
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

```
Error: Debug Adapter has to be specified, see "interface" command
embedded:startup.tcl:56: Error:
in procedure 'script'
at file "embedded:startup.tcl", line 56
```

Подключение и прошивка пикозонда

Прошивка Picoprobe UF2
Двоичный файл picoprobe UF2 можно загрузить из раздела программных утилит Raspberry Pi
Страница документации Pico. Перейдите в раздел Raspberry Pi Pico, прокрутите вниз до программных утилит и загрузите UF2 в разделе "Отладка с использованием другого Raspberry Pi Pico".
В этих инструкциях по сборке предполагается, что вы работаете в Linux и установили SDK.

```
$ cd ~/pico
$ git clone https://github.com/raspberrypi/picoprobe.git
$ cd picoprobe
$ git submodule update --init
$ mkdir build
$ cd build
$ export PICO_SDK_PATH=../../pico-sdk
$ cmake ..
$ make -j4
```

Загрузите Raspberry Pi Pico, который вы хотели бы использовать в качестве отладчика, нажав кнопку BOOTSEL и перетащив picoprobe.uf2.

Подключение пикозонда

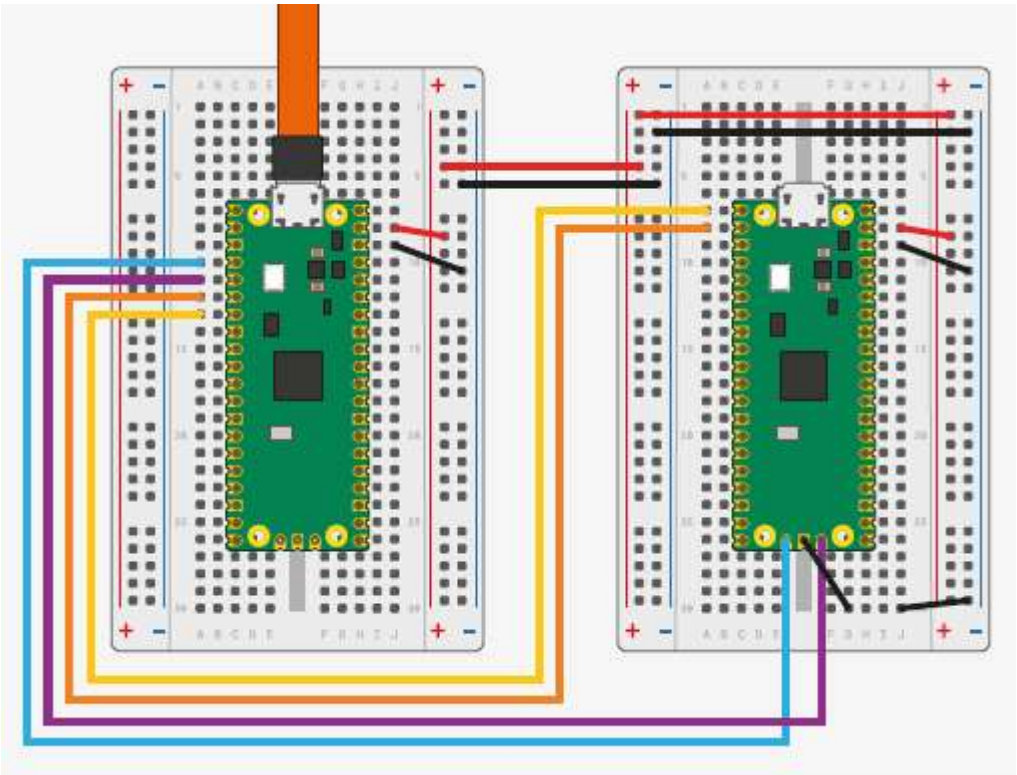


Рисунок 39. Подключение между Pico A (слева) и Pico B (справа) настройка Pico A в качестве отладчика. Обратите внимание, что если Pico B является USB-хостом, то вы хотели бы подключить VBUS к VBUS, чтобы он мог выдавать 5 В вместо VSYS к VSYS.

Схема подключения между двумя платами Pico показана на рисунке 39.

Pico A GND -> Pico B GND
Pico A GP2 -> Pico B SWCLK
Pico 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

Минимальный набор подключений для загрузки и запуска кода через OpenOCD - это GND, SWCLK и SWDIO. Подключение проводов UART также позволит вам взаимодействовать с последовательным портом UART правого Pico через левый Pico USB-соединение. Вы также можете использовать только провода UART для подключения к любому другому последовательному устройству UART, например, к загрузочной консоли на Raspberry Pi.
Необязательно, чтобы запитать Pico A от Pico B, вы также должны подключить Pico A VSYS -> Pico B VSYS

Важно
Если Pico B является USB-хостом, то вы должны подключать VBUS к VBUS, а не WAYS к WAYS, чтобы Pico B мог подавать напряжение 5 В на свой Разъем USB. Если Pico Bios использует USB в режиме устройства или вообще не использует свой USB, в этом нет необходимости.

Интерфейсы Picoprobe

Устройство Picoprobe представляет собой составное устройство, имеющее два интерфейса USB:
1. Совместимый с классом CDC UART (последовательный порт), что означает, что он работает в Windows из коробки
2. Специфичный для производителя интерфейс для данных SWD-зондирования, соответствующий CMSIS-DAP версии 2.

ПРИМЕЧАНИЕ
Предыдущие версии микропрограммы picoprobe требовали ручной установки libusb-win32 для интерфейса конкретного производителя. В этом больше нет необходимости, поскольку WinUSB автоматически назначается интерфейсу отладки.

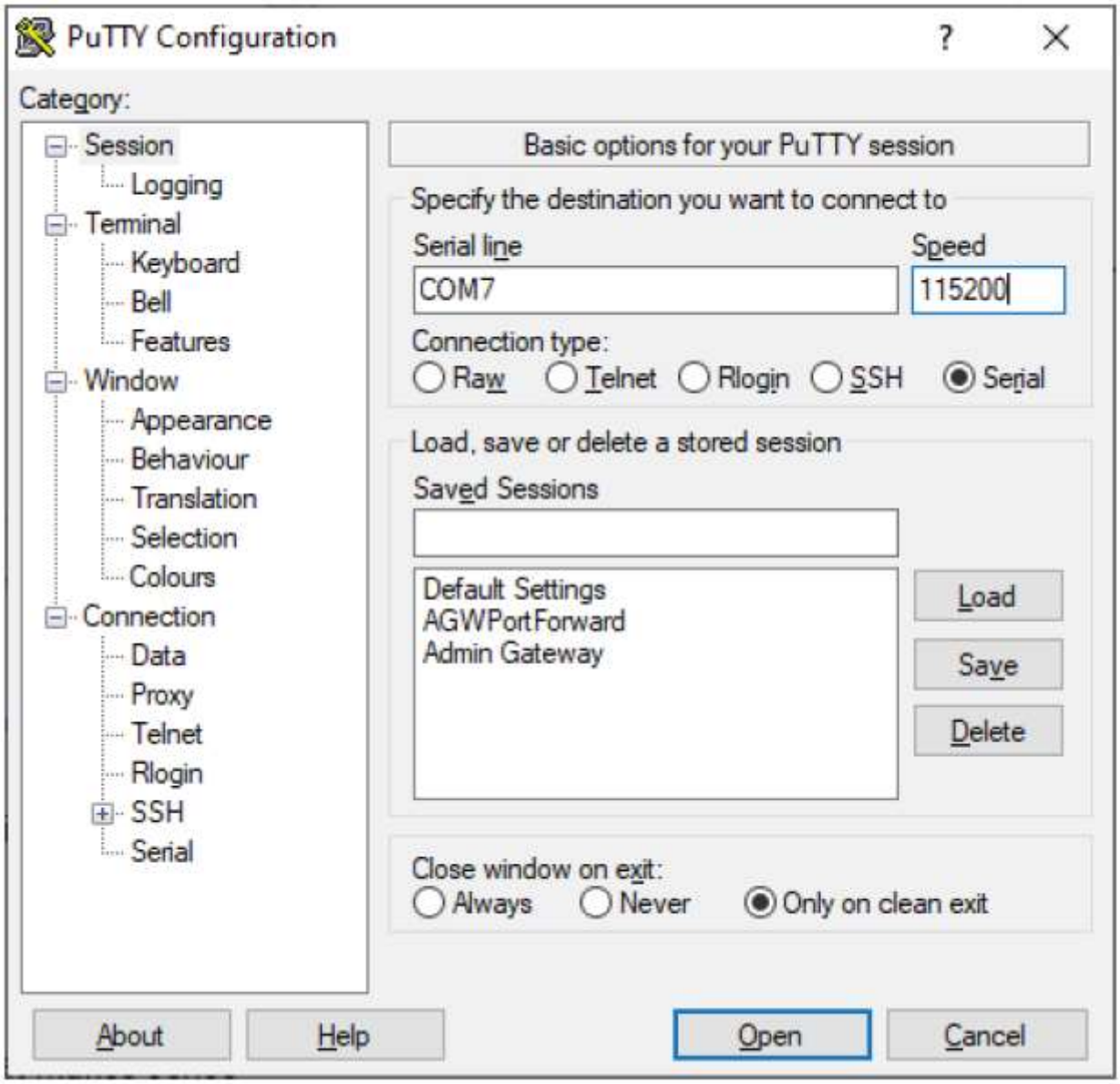
Использование UART от Picoprobe

Линукс

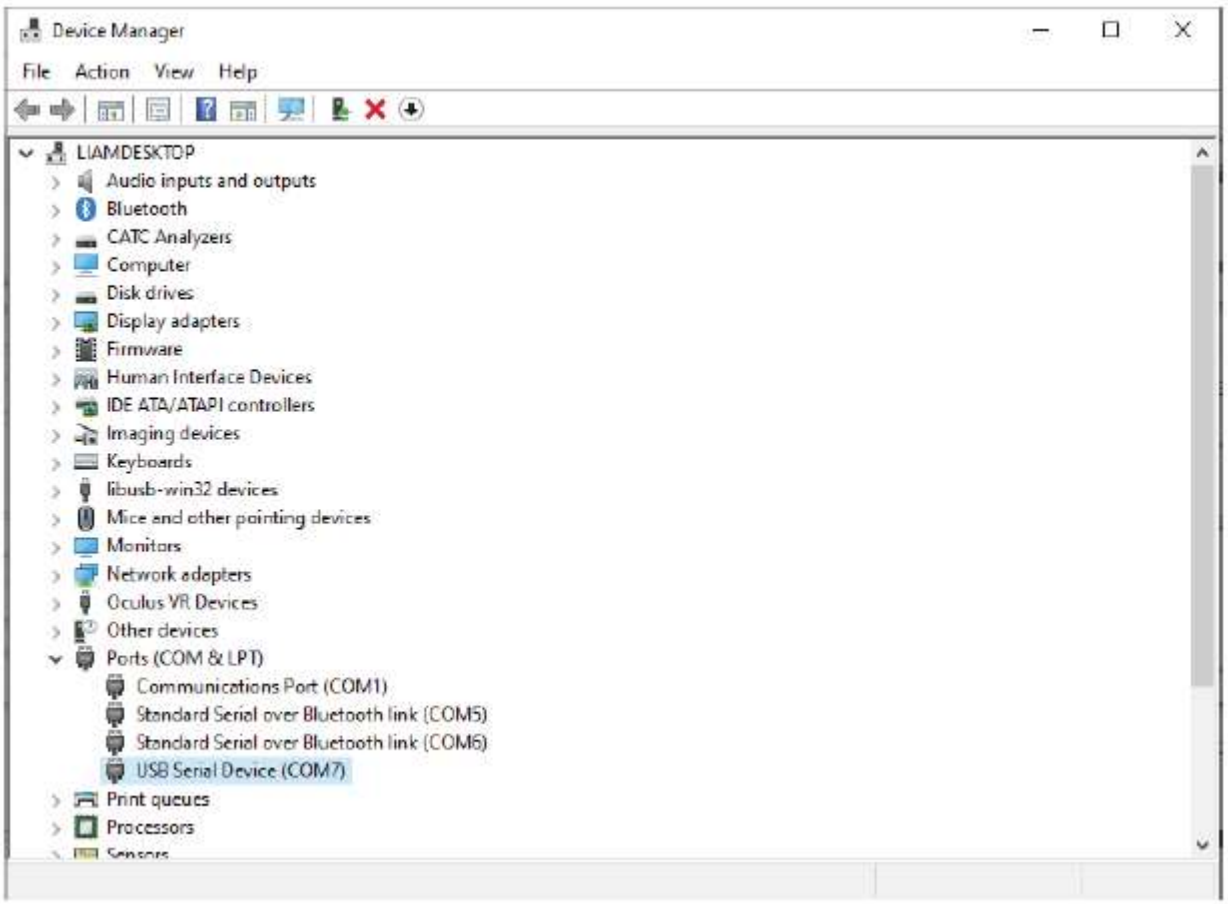
$ sudo minicom -D /dev/ttyACM0 -b 115200

Windows

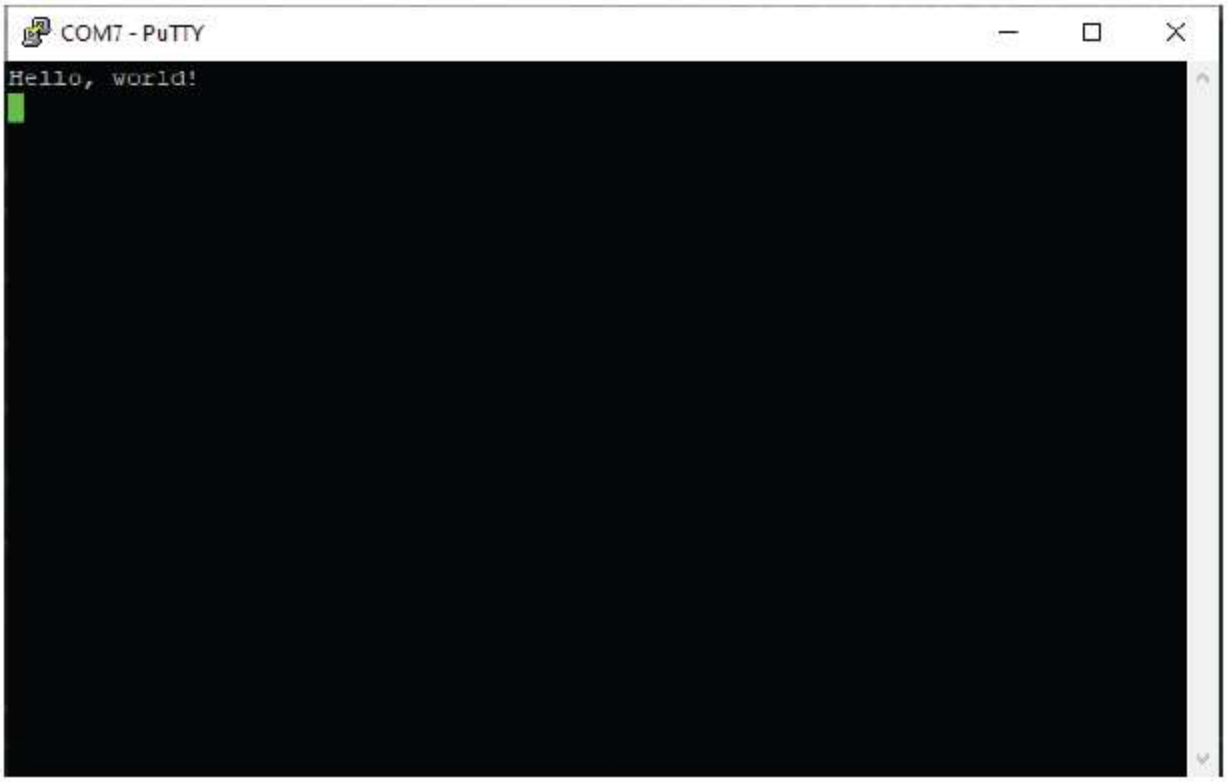
Скачайте и установите PuTTY https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html



Откройте диспетчер устройств и найдите номер COM-порта Picoprobe. В данном примере это COM7.



Получится это:



Мак

Мы можем установить minicom с помощью Homebrew,

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

Использование Picoprobe с OpenOCD

Одинаково для всех платформ,

```
$ src/openocd -f interface/cmsis-dap.cfg -c "adapter speed 5000" -f target/rp2040.cfg -s tcl
```

Подключите GDB, как обычно, к целевому удаленному локальному хосту:3333

```
target remote localhost:3333
```

Приложение B: Использование инструмента Picotool

Можно встроить информацию в двоичный файл Raspberry Pi Pico, который можно извлечь с помощью утилиты командной строки под названием picotool.
Получение picotool Утилита picotool доступна в собственном репозитории. Вам нужно будет клонировать и собрать его, если вы еще не запускали скрипт pico-setup.

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

Вам также потребуется установить libusb, если он еще не установлен,

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

ПРИМЕЧАНИЕ
Если вы создаете pico tool на macOS , вы можете установить libusb с помощью Homebrew,

```
$ brew install libusb pkg-config
```

Хотя, если вы создаете на Microsoft Windows, вы можете загрузить и установить двоичный файл libusb для Windows непосредственно с сайта libusb.info.

Создание инструмента picotool

Создание инструмента picotool можно выполнить следующим образом,

```
$ mkdir build
$ cd build
$ export PICO_SDK_PATH=~/pico/pico-sdk
$ cmake ../
$ make
```

это сгенерирует двоичный файл командной строки pico tool в каталоге build/picotool.

ПРИМЕЧАНИЕ
Если вы создаете на Microsoft Windows, вам следует вызвать CMake следующим образом,

```
C:\Users\pico\picotool> mkdir build
C:\Users\pico\picotool> cd build
C:\Users\pico\picotool\build> cmake .. -G "NMake Makefiles"
C:\Users\pico\picotool\build> nmake
```

С помощью picotool

Двоичный файл picotool включает справочную функцию командной строки,

```
$ picotool help
PICOTOOL:
Tool for interacting with a RP2040 device in BOOTSEL mode, or with a RP2040 binary
SYNOPSIS:
picotool info [-b] [-p] [-d] [-l] [-a] [--bus <bus>] [--address <addr>] [-f] [-F]
picotool info [-b] [-p] [-d] [-l] [-a] <filename> [-t <type>]
picotool load [-n] [-N] [-v] [-x] <filename> [-t <type>] [-o <offset>] [--bus <bus>]
[--address <addr>] [-f] [-F]
picotool save [-p] [--bus <bus>] [--address <addr>] [-f] [-F] <filename> [-t <type>]
picotool save -a [--bus <bus>] [--address <addr>] [-f] [-F] <filename> [-t <type>]
picotool save -r <from> <to> [--bus <bus>] [--address <addr>] [-f] [-F] <filename>
[-t
<type>]
picotool verify [--bus <bus>] [--address <addr>] [-f] [-F] <filename> [-t <type>] [-
r
<from> <to>] [-o <offset>]
picotool reboot [-a] [-u] [--bus <bus>] [--address <addr>] [-f] [-F]
picotool version [-s]
picotool help [<cmd>]
COMMANDS:
info Display information from the target device(s) or file.
Without any arguments, this will display basic information for all connected
RP2040 devices in BOOTSEL mode
load Load the program / memory range stored in a file onto the device.
save Save the program / memory stored in flash on the device to a file.
verify Check that the device contents match those in the file.
reboot Reboot the device
version Display picotool version
help Show general help or help for a specific command
Use "picotool help <cmd>" for more info
```

Информация: Отображает информацию с целевого устройства (ов) или файла.
Без каких-либо аргументов, это отобразит основную информацию для всех подключенных Устройства RP2040 в режиме загрузки загружают на устройство программу/диапазон памяти, сохраненный в файле.
Сохранить: Сохраните программу / память, сохраненные во флэш-памяти устройства, в файл.
Проверка: Убедитесь, что содержимое устройства соответствует содержимому файла.

Перезагрузка: Перезагрузите версию устройства
Отобразите: справку о версии picotool.Отобразите общую справку или справку по конкретной команде
Используйте "справка picotool <cmd>" для получения дополнительной информации

ПРИМЕЧАНИЕ

Для выполнения большинства команд требуется подключение устройства RP2040 в режиме загрузки.

ВАЖНО

Если вы получаете сообщение об ошибке, доступные устройства RP2040 в режиме загрузки не найдены. сопровождается примечанием, аналогичным устройству на шине 1, адрес 7, по-видимому, является устройством RP2040 в режиме загрузки, но picotool не удалось подключиться, указывая, что подключен Raspberry Pi Pico, тогда вы можете запустить picotool с помощью sudo, например

. $ sudo picotool info –a

Если вы получите это сообщение в Windows, вам нужно будет установить драйвер.
Загрузите и запустите Zadig, выберите Picotool из выпадающего списка и выберите libusb-win32 в качестве драйвера и нажмите на кнопку "Установить драйвер".

Начиная с версии 1.1 picotool, также возможно взаимодействовать с устройствами RP2040, которые не находятся в режиме загрузки, но используют поддержку USB stdio из SDK, используя аргумент -f picotool.

Отображаемая информация

Таким образом, теперь в SDK есть поддержка двоичной информации, которая позволяет легко хранить компактную информацию, которую может найти picotool (смотрите двоичную информацию ниже). Команда info предназначена для считывания этой информации.
Информация может быть считана либо с одного или нескольких подключенных устройств RP2040 в режиме загрузки, либо из файла. Этот файл может быть файлом ELF, UF2 или BIN.

```
$ picotool help info
INFO:
Display information from the target device(s) or file.
Without any arguments, this will display basic information for all connected RP2040
devices
in BOOTSEL mode
SYNOPSIS:
picotool info [-b] [-p] [-d] [-l] [-a] [--bus <bus>] [--address <addr>] [-f] [-F]
picotool info [-b] [-p] [-d] [-l] [-a] <filename> [-t <type>]
OPTIONS:
Information to display
-b, --basic
Include basic information. This is the default
-p, --pins
Include pin information
-d, --device
Include device information
-l, --build
Include build attributes
-a, --all
Include all information
```

```
TARGET SELECTION:
To target one or more connected RP2040 device(s) in BOOTSEL mode (the default)
--bus <bus>
Filter devices by USB bus number
--address <addr>
Filter devices by USB device address
-f, --force
Force a device not in BOOTSEL mode but running compatible code to reset so the
command can be executed. After executing the command (unless the command itself is
a 'reboot') the device will be rebooted back to application mode
-F, --force-no-reboot
Force a device not in BOOTSEL mode but running compatible code to reset so the
command can be executed. After executing the command (unless the command itself is
a 'reboot') the device will be left connected and accessible to picotool, but
without the RPI-RP2 drive mounted
To target a file
<filename>
The file name
-t <type>
Specify file type (uf2 | elf | bin) explicitly, ignoring file extension
```

Например, подключите Raspberry Pi Pico к компьютеру в режиме массового хранения, нажав и удерживая кнопку загрузки, прежде чем подключить его к USB-порту. Затем откройте окно терминала и введите,

```
$ sudo picotool info
Program Information
name: hello_world
features: stdout to UART
```

или,

```
$ sudo picotool info -a
Program Information
name: hello_world
features: stdout to UART
binary start: 0x10000000
binary end: 0x1000606c
Fixed Pin Information
20: UART1 TX
21: UART1 RX
Build Information
build date: Dec 31 2020
build attributes: Debug build
Device Information
flash size: 2048K
ROM version: 2
```

для получения дополнительной информации. В качестве альтернативы вы можете просто получить информацию об используемых пин-кодах следующим образом,

```
$ sudo picotool info -bp
Program Information
name: hello_world
features: stdout to UART
Fixed Pin Information
```

```
20: UART1 TX
21: UART1 RX
```

Инструмент также можно использовать с двоичными файлами, которые все еще находятся в вашей локальной файловой системе,

```
$ picotool info -a lcd_1602_i2c.uf2
File lcd_1602_i2c.uf2:
Program Information
name: lcd_1602_i2c
web site: https://github.com/raspberrypi/pico-examples/tree/HEAD/i2c/lcd_1602_i2c
binary start: 0x10000000
binary end: 0x10003c1c
Fixed Pin Information
4: I2C0 SDA
5: I2C0 SCL
Build Information
build date: Dec 31 2020
```

Сохранение программы

Сохранение позволяет сохранить область памяти, программу или всю флэш-память устройства целиком в файл BIN или UF2.

```
$ picotool help save
SAVE:
Save the program / memory stored in flash on the device to a file.
SYNOPSIS:
picotool save [-p] [--bus <bus>] [--address <addr>] [-f] [-F] <filename> [-t <type>]
picotool save -a [--bus <bus>] [--address <addr>] [-f] [-F] <filename> [-t <type>]
picotool save -r <from> <to> [--bus <bus>] [--address <addr>] [-f] [-F] <filename>
[-t
<type>]
OPTIONS:
Selection of data to save
-p, --program
Save the installed program only. This is the default
-a, --all
Save all of flash memory
-r, --range
Save a range of memory. Note that UF2s always store complete 256 byte-aligned
blocks of 256 bytes, and the range is expanded accordingly
<from>
The lower address bound in hex
<to>
The upper address bound in hex
Source device selection
--bus <bus>
Filter devices by USB bus number
--address <addr>
```

```
        Filter devices by USB device address
        -f, --force
        Force a device not in BOOTSEL mode but running compatible code to reset so the
        command can be executed. After executing the command (unless the command itself is
        a 'reboot') the device will be rebooted back to application mode
        -F, --force-no-reboot
        Force a device not in BOOTSEL mode but running compatible code to reset so the
        command can be executed. After executing the command (unless the command itself is
        a 'reboot') the device will be left connected and accessible to picotool, but
        without the RPI-RP2 drive mounted
        File to save to
        <filename>
        The file name
        -t <type>
  Specify file type (uf2 | elf | bin) explicitly, ignoring file extension
```

Принудительно перезагрузите устройство, не находящееся в режиме загрузки, но выполняющее совместимый код, чтобы команда могла быть выполнена. После выполнения команды (если только сама команда не является "перезагрузкой") устройство будет перезагружено обратно в режим приложения
-F, --принудительная перезагрузка без перезагрузки
Принудительно перезагрузите устройство, не находящееся в режиме загрузки, но выполняющее совместимый код, чтобы команда могла быть выполнена. После выполнения команды (если только сама команда не является "перезагрузкой") устройство останется подключенным и доступным для picotool, но без подключенного диска RPI-RP2
Файл для сохранения в
Укажите тип файла (uf2 | elf | bin) явно, игнорируя расширение файла
Например,
$ sudo picotool info
Информация о программе название: lcd_1602_i2c
веб-сайт: https://github.com/raspberrypi/pico-examples/tree/HEAD/i2c/lcd_1602_i2c
$ picotool экономит ложку.uf2
Сохранение файла: [==============================] 100%
Записал 51200 байт в spoon.uf2
$ picotool info spoon.uf2
Файл spoon.uf2:
Информация о программе
название: lcd_1602_i2c
веб-сайт: https://github.com/raspberrypi/pico-examples/tree/HEAD/i2c/lcd_1602_i2c
Двоичная информация Двоичная информация может быть обнаружена машиной и, как правило, потребляема машиной. Я говорю "в целом", потому что любой человек может включить любую информацию, и мы можем отличить ее от нашей, но от них зависит, сделают ли они свои данные самоописывающимися.

Основная информация

Эта информация действительно полезна, когда вы берете в руки фотопленку и не знаете, что на ней написано! Основная информация включает в себя

- название программы
- описание программы
- строка версии программы
- дата сборки программы
- url программы
- конечный адрес программы
- функции программы, это список, составленный из отдельных строк в двоичном коде, который может быть отображен (например, у нас будет один для UART stdio и один для USB stdio) в SDK
- атрибуты сборки, это аналогичный список строк для вещей, относящихся к самому двоичному файлу (например, отладочная сборка) Штырьки

Это, безусловно, удобно, когда у вас есть исполняемый файл с именем hello_serial.elf, но вы забыли, для какой платы на базе RP2040 он был создан, поскольку на разных платах могут быть разные контакты.

Статические (фиксированные) назначения pin-кодов могут быть записаны в двоичном формате в очень компактной форме:

```
$ picotool info --pins sprite_demo.elf
File sprite_demo.elf:
Fixed Pin Information
Getting started with Raspberry Pi Pico
Binary Information 73
0-4: Red 0-4
6-10: Green 0-4
11-15: Blue 0-4
16: HSync
17: VSync
18: Display Enable
19: Pixel Clock
20: UART1 TX
21: UART1 RX
```

Включение двоичной информации

Двоичная информация объявляется в программе с помощью макросов; в предыдущем примере:

```
$ picotool info --pins sprite_demo.elf
File sprite_demo.elf:
Fixed Pin Information
0-4: Red 0-4
6-10: Green 0-4
11-15: Blue 0-4
16: HSync
17: VSync
18: Display Enable
19: Pixel Clock
20: UART1 TX
21: UART1 RX
```

В функции setup_default_uart есть одна строка:

```
bi_decl_if_func_used(bi_2pins_with_func(PICO_DEFAULT_UART_RX_PIN, PICO_DEFAULT_UART_TX_PIN,
GPIO_FUNC_UART));
```

Два pin-кода и функция UART сохраняются, а затем декодируются в их фактические имена функций (UART1 TX и т.д.) с помощью picotool. bi_decl_if_func_used гарантирует, что двоичная информация включается только в том случае, если вызывается содержащая функция.

Точно так же видеокод содержит несколько строк, подобных этой:
bi_decl_if_func_used(bi_pin_mask_with_name(0x1f << (PICO_SCANVIDEO_COLOR_PIN_BASE + PICO_SCANVIDEO_DPI_PIXEL_RSHIFT), "Красный 0-4"));

Подробности

Все устроено так, чтобы занимать как можно меньше места, но вы можете отключить все с помощью препроцессора var
PICO_NO_BINARY_INFO=1. Кроме того, любой код SDK, который вставляет двоичную информацию, может быть отдельно исключен с помощью его собственного препроцессора var.
Тебе нужно,

```
#include "pico/binary_info.h"
```

В заголовках есть куча макросов bi_

```
#define bi_binary_end(end)
#define bi_program_name(name)
#define bi_program_description(description)
#define bi_program_version_string(version_string)
#define bi_program_build_date_string(date_string)
#define bi_program_url(url)
#define bi_program_feature(feature)
#define bi_program_build_attribute(attr)
#define bi_1pin_with_func(p0, func)
#define bi_2pins_with_func(p0, p1, func)
#define bi_3pins_with_func(p0, p1, p2, func)
#define bi_4pins_with_func(p0, p1, p2, p3, func)
#define 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, label)
#define bi_pin_mask_with_names(pmask, label)
#define bi_1pin_with_name(p0, name)
#define bi_2pins_with_names(p0, name0, p1, name1)
#define bi_3pins_with_names(p0, name0, p1, name1, p2, name2)
#define bi_4pins_with_names(p0, name0, p1, name1, p2, name2, p3, name3)
```

которые используют базовые макросы, например

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

Затем вы либо используете bi_decl(bi_blah(...)) для безусловного включения двоичной информации blah, либо bi_decl_if_func_used(bi_blah(...)) для двоичной информации, которая может быть удалена, если компоновщик не включил заключающую функцию в двоичный файл (подумайте --gc-sections).

Например,

```
1 #include <stdio.h>
2 #include "pico/stdlib.h"
3 #include "hardware/gpio.h"
4 #include "pico/binary_info.h"
5
6 const uint LED_PIN = 25;
7
8 int main() {
9
10 bi_decl(bi_program_description("This is a test binary."));
11 bi_decl(bi_1pin_with_name(LED_PIN, "On-board LED"));
12
13 setup_default_uart();
14 gpio_set_function(LED_PIN, GPIO_FUNC_PROC);
15 gpio_set_dir(LED_PIN, GPIO_OUT);
16 while (1) {
17 gpio_put(LED_PIN, 0);
18 sleep_ms(250);
19 gpio_put(LED_PIN, 1);
20 puts("Hello World¥n");
21 sleep_ms(1000);
22 }
23 }
```

при запросе с помощью picotool,

```
$ sudo picotool info -a test.uf2
File test.uf2:
Program Information
name: test
description: This is a test binary.
features: stdout to UART
binary start: 0x10000000
binary end: 0x100031f8
Fixed Pin Information
0: UART0 TX
1: UART0 RX
25: On-board LED
Build Information
build date: Jan 4 2021
```

отображает наши информационные строки в выходных данных.

Установка общих полей из CMake

Вы также можете задать поля непосредственно из файла CMake вашего проекта, например,

```
pico_set_program_name(foo "не foo") ①
pico_set_program_description(foo "это еда")
pico_set_program_version_string(foo "0.00001a")
pico_set_program_url(foo "www.plinth.com/foo")
```

1.По умолчанию будет использоваться имя "foo".

ПРИМЕЧАНИЕ

Все это передается компилятору в качестве аргументов командной строки, поэтому, если вы планируете использовать кавычки, новые строки и т. д. возможно, вам повезет больше, определив его с помощью bi_decl в коде.