НАБІР ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ RASPBERRY PI

RASPBERRY PI PICO SDK

У статті наведена коротка інформація про склад бібліотек Raspberry Pi Pico, призначених для використання як у невбудованих, так і в вбудованих системах Інтернету речей.

Abstract – The article provides brief information about the composition of Raspberry Pi Pico libraries designed for use in both non-embedded and embedded Internet of things systems.

В. Макаренко

V. Makarenko

Набір для розробки програмного забезпечення Raspberry Pi Pico SDK (Software Development Kit), надалі SDK, надає заголовки, бібліотеки та систему збірки, необхідні для написання програм для мікроконтролерних пристроїв серії RP, таких як Raspberry Pi Pico, на C, C++ або мові асемблера. Пакет SDK створено для надання API (інтерфейсу прикладного програмування) та середовища програмування, яке знайоме як розробникам невбудованих, так і розробникам вбудованих систем [1].

Одна програма одночасно виконується на пристрої за допомогою звичайного методу main(). Стандартні бібліотеки C/C++ підтримуються разом із API для доступу до апаратного забезпечення мікроконтролера, включаючи DMA, IRQ, а також різноманітних периферійних пристроїв з фіксованими функціями та PIO (програмований ввід-вивід).

Крім того, SDK надає бібліотеки вищого рівня для роботи з таймерами, USB, синхронізацією та багатоядерним програмуванням, а також додаткову функціональність високого рівня, побудовану за допомогою PIO, наприклад аудіо. SDK можна використовувати для створення чого завгодно: від простих додатків або повноцінних середовищ виконання, таких як MicroPython, до низькорівневого програмного забезпечення, такого як сам завантажувач мікроконтролера на чіпі.

SDK використовує CMake для керування збіркою. CMake широко підтримується IDE (інтегрованим середовищем розробки) і дозволяє просто вказати збірку (через файли CMakeLists.txt), з яких CMake може створити систему збірки (для використання make, ninja або іншими інструментами збірки), налаштовану для платформи та будь-яких змінних конфігурації, які вибере розробник.

Крім того, що CMake є широко використовуваною системою збирання для розробки на C/C++, CMake є фундаментальною для структури SDK, а також для конфігурації та створення програм.

SDK створює виконуваний файл, який включає весь код, необхідний для роботи на пристрої (окрім конкретного пристрою з плаваючою комою та іншого оптимізованого коду, що міститься в bootrom мікроконтролера).

Розглянемо документацію до SDK для різних застосувань [1]. Структура SDK наведена на рис. 1.

Pico C SDK

Introduction
Hardware APIs
High Level APIs
Third-party Libraries
Networking Libraries
Runtime Infrastructure
External API Headers

Рис. 1. Структура Pico C SDK

Апаратні API-інтерфейси (Hardware APIs)

Ця група бібліотек забезпечує тонкий та ефективний інтерфейс прикладного програмування C (API) / абстракції для доступу до апаратного забезпечення мікроконтролерів серії RP без необхідності прямого читання та запису апаратних регістрів [2]. В табл. 1 наведено склад бібліотек API для доступу до апаратного забезпечення мікроконтролерів

API високого рівня [3]

Ця група бібліотек забезпечує функціональність вищого рівня, не пов'язану з апаратним забезпеченням, або надає більш багатий набір функціональних можливостей, що виходять за рамки базових апаратних інтерфейсів.

Таблиця 1. Склад бібліотек API для доступу до апаратного забезпечення мікроконтролерів

Бібліотека	Призначення бібліотеки
hardware_adc	АЦП – Analog to Digital Converter (ADC) API.
hardware_base	Низькорівневі типи і засоби доступу до апаратних регістрів, що відображаються в пам'яті.
hardware_claim	Спрощене управління апаратними ресурсами API.
hardware_clocks	Управління годинником API.
hardware_divider	RP2040 низькорівневий апаратний дільник частоти API. Платформи, що не належать до RP2040, надають версії програмного забезпечення для всіх функцій.
hardware_dcp	Макроси збірки для подвійного співпроцесора RP2350.
hardware_dma	Контролер прямого доступу до па м'яті (DMA) API.
channel_con fig	Конфігурація каналу DMA.
hardware_exception	Методи налаштування обробників винятків процесора.
hardware_flash	Низькорівневе програмування flash і стирання API.
hardware_gpio	Введення / виведення загального призначення (GPIO) API.
hardware_hazard3	Аксесуари для автомобілів RISC-V, що відповідають вимогам стандарту Hazard3, і вбудовані компоненти для призначених для користувача інструкцій по Hazard 3. RP2350
hardware_i2c	I2C контролер API.
hardware_interp	Апаратний інтерполятор API.
interp_con fig	Конфігурація інтерполятора .
hardware_irq	Обробка апаратних переривань API.
hardware_pio	Програмоване введення-виведення (PIO) API.
sm_con fig	Конфігурація кінцевого автомата PIO.
pio_instructions	Кодування команд PIO.
hardware_pll	API-інтерфейси управління контуром фазового автопідстроювання частоти.
hardware_powman	Управління живленням API. RP2350
hardware_pwm	API апаратної широтно-імпульсної модуляції (ШІМ).
hardware_resets	API апаратного скидання.
hardware_riscv	Засоби доступу до стандартного обладнання RISC-V (переважно CSR) RP2350
hardware_riscv_platform_timer	Аксесуари для стандартного таймера платформи RISC-V (mtime/mtimecmp), доступні на мікроконтролерах Raspberry Pi з процесорами RISC-V. RP2350
hardware_rtc	Апаратний API годинника реального часу. RP2040
hardware_rcp	Вбудовані функції і макроси збірки для співпроцесора з надмірністю. RP2350
hardware_spi	Апаратний SPI API.
hardware_sha256	API апаратного прискорювача SHA-256. RP2350
hardware_sync	Низькорівневі апаратні спін-блокування, бар'єрні інтерфейси та інтерфейси обробки подій.
hardware_ticks	Апаратні відмітки API.
hardware_timer	Низькорівневий API апаратного таймера.
hardware_uart	Апаратний UART API.
hardware_vreg	API для регулювання напруги.
hardware_watchdog	API апаратного сторожового таймера.
hardware_xosc	API кварцового генератора (XOSC).

В табл. 2 наведено склад бібліотек API високого рівня.

Таблиця 2. Склад бібліотек API високого рівня

Бібліотека	Призначення бібліотеки
pico_aon_timer	Абстракція високого рівня "Завжди ввімкнено таймер".
pico_async_context	async_context надає логічно однопотоковий контекст для виконання роботи та реагування на асинхронні події. Таким чином, екземпляр async_context підходить для обслуговування сторонніх бібліотек, які не є реентерабельними.
async_context_freertos	async_context_freertos забезпечує реалізацію async_context, яка обробляє асинхронну роботу в окремому завданні FreeRTOS.
async_context_poll	async_context_poll забезпечує реалізацію async_context, яка призначена для використання з простим циклом опитування на одному ядрі. Це небезпечно для потоків.
async_context_threadsafe_background	async_context_threadsafe_background забезпечує реалізацію async_context, яка регулює асинхронну роботу в IRQ з низьким пріоритетом, і користувачеві не потрібно опитувати роботу.
pico_bootsel_via_double_reset	Додаткова підтримка для швидкого подвійного скидання системи при переході в режим початкового завантаження.
pico_flash	Високорівневий flash API.
pico_i2c_slave	Функції, що забезпечують керований перериваннями ведений інтерфейс I2C.
pico_multicore	Додана підтримка запуску коду на другому процесорному ядрі (core 1) і взаємодії з ним.
fifo	Функції для міжядерних FIFO.
doorbell	Функції, пов'язані з дверними дзвінками, які ядро може використовувати для надсилання запитів IRQ собі або іншому ядру.
lockout	Функції, що дозволяють одному ядру змусити інше ядро призупинити виконання у відомому стані.
pico_rand	API генератора випадкових чисел.
pico_sha256	Апаратно прискорена реалізація SHA-256. RP2350
pico_stdlib	Об'єднання основної підмножини бібліотек Raspberry Pi Pico SDK, що використовуються більшістю виконуваних файлів, а також деяких додаткових корисних методів.
pico_sync	Примітиви синхронізації та взаємне виключення.
critical_section	API критичної секції для короткочасного взаємного виключення, безпечний для IRQ і багатоядерних систем.
lock_core	Базова підтримка примітивів синхронізації / блокування.
mutex	Mutex API для взаємного виключення без IRQ між ядрами.
sem	Семафорний API для обмеження доступу до ресурсу.
pico_time	API для точних міток часу, сплячого режиму та зворотних викликів на основі часу.
timestamp	Функції часових міток, що відносяться до моментів часу (включаючи поточний час).
sleep	Функції сплячого режиму для затримки виконання при зниженому енергоспоживанні.
alarm	Функції сигналізації для планування майбутнього виконання.
repeating_timer	Функції повторюваного таймера для простого планування повторного виконання.
pico_unique_id	Унікальний API доступу до ідентифікатора пристрою.

Продовження табл. 2

Бібліотека	Призначення бібліотеки
pico_util	Корисні структури даних і службові функції.
datetime	Форматування дати і часу.
pheap	Реалізація спряженої пари.
queue	Реалізація багатоядерної та IRQ-безпечної черги.

Сторонні бібліотеки (Third-party Libraries) [4]

Сторонні бібліотеки для реалізації високорівневої функціональності. Склад сторонніх бібліотек наведено в табл. 3.

Таблиця 3. Склад сторонніх бібліотек

Бібліотека	Призначення бібліотеки
tinyusb_device	Підтримка режиму мініатюрного USB-пристрою для RP2040. Веб-сайт із документацією TinyUSB можна знайти у [5].
tinyusb_host	Підтримка режиму хосту TinyUSB для RP2040.

Мережеві бібліотеки (Networking Libraries)

Функції для реалізації мережевої взаємодії наведены в табл. 4 [6].

Таблиця 4. Склад мережевих бібліотек

Бібліотека	Призначення бібліотеки
pico_btstack	Бібліотеки інтеграції / оболонки для BTStack, документація щодо яких знаходиться у [7].
pico_lwip	Бібліотеки інтеграції / оболонки для Lightweight IP (lwIP) стеку, документація щодо яких знаходиться у [8].
pico_lwip_arch	Адаптери компілятора LwIP. За замовчуванням вони не включені в pico_lwip, на випадок, якщо ви хочете реалізувати власні.
pico_lwip_freertos	Бібліотека Glue для інтеграції lwip в режимі NO_SYS=0 з SDK.
pico_lwip_nosys	Бібліотека Glue для інтеграції lwip в режимі NO_SYS=1 з SDK.
pico_cyw43_driver	Оболонка для cyw43_driver нижчого рівня, яка інтегрує його з pico_async_context для виконання фонової роботи.
pico_btstack_cyw43	Низькорівнева підтримка Bluetooth HCI.
pico_cyw43_arch	Архітектура для інтеграції драйвера CYW43 (для безброводової мережі на PicoW) та lwip (для стека TCP/IP) у SDK. Це також потрібно для доступу до вбудованого світлодіода на Pico W.
cyw43_driver	Драйвер, який використовується для безброводового зв'язку Pico W.
cyw43_ll	Низькорівневий інтерфейс драйвера CYW43.

Інфраструктура часу виконання (Runtime Infrastructure) [9]

Бібліотеки, які використовуються для забезпечення ефективної реалізації певних функцій на рівні мови та бібліотеки C, а також бібліотеки інтерфейсу CMake, що абстрагують етапи компіляції та посилання в SDK. Склад бібліотек інфраструктури часу наведено в табл. 5.

Зовнішні заголовки API (External API Headers) [10]

Заголовки для інтерфейсів, які спільно використовуються з кодом за межами SDK. Склад бібліотек заголовків для інтерфейсів, які спільно використовуються з кодом за межами SDK наведено в табл. 6.

Таблиця 5. Склад бібліотек інфраструктури часу

Бібліотека	Призначення бібліотеки
boot_stage2	Завантажувачі другого етапу відповідають за Налаштування зовнішньої флеш-пам'яті.
pico_atomic	Допоміжні реалізації для C11 atomics.
pico_base	Основні типи та макроси для Raspberry Pi Pico SDK.
pico_binary_info	Двійкова інформація призначена для вбудовування машиночитаної інформації у двійковий файл у FLASH.
pico_bootrom	Доступ до функцій та даних у нижній частині екрана.
pico_bit_ops	Оптимізовані функції маніпулювання бітами.
pico_cxx_options	Бібліотека, не пов'язана з кодом, яка керує параметрами компілятора, пов'язаними з C++
pico_clib_interface	Надає необхідний код зв'язку, необхідний для конкретного використовуваного середовища виконання C / C++.
pico_crt0	Надає сценарії компоновщика за замовчуванням і точку входу/виходу з програми.
pico_divider	Оптимізовані функції 32-і 64-розрядного ділення частоти, прискорені апаратним дільником частоти RP2040.
pico_double	Оптимізовані функції з плаваючою комою подвійної точності.
pico_float	Оптимізовані функції з плаваючою комою одинарної точності.
pico_int64_ops	Оптимізовано реалізації заміни вбудованого в компілятор 64-бітного множення.
pico_malloc	Багатоядерний захист для malloc, calloc і free.
pico_mem_ops	Надає оптимізовані реалізації заміни вбудованих в компілятор memcpy, memset і пов'язаних з ними функцій.
pico_platform	Макроси та визначення (і функції, якщо вони не включені в асемблерний код) для пристроїв / архітектури сімейства RP2, щоб забезпечити загальну абстракцію від специфіки компілятора / платформи низького рівня.
pico_printf	Компактна Заміна printf на Marco Paland (info@paland.com)
pico_runtime	Основна підтримка середовища виконання для запуску попередніх основних ініціалізаторів, що надаються іншими бібліотеками.
pico_runtime_init	Основні функції ініціалізації середовища виконання, необхідні для налаштування середовища виконання перед входом в main.
pico_stdio	Індивідуальна підтримка studio, що дозволяє здійснювати введення і виведення даних з UART, USB, напів-хостингу і т. д.
pico_stdio_semihosting	Експериментальна підтримка стандартного виводу з використанням напівгостингу оперативної пам'яті.
pico_stdio_uart	Підтримка стандартного вводу / виводу за допомогою UART.
pico_stdio_rtt	Підтримка стандартного вводу / виводу за допомогою SEGGER RTT.
pico_stdio_usb	Підтримка стандартного введення / виводу по послідовному USB (CDC).
pico_standard_binary_info	Містить інформацію про двійковий файл за замовчуванням, який можна відобразити за допомогою Pico tool.
pico_standard_link	Містить інформацію про двійковий файл за замовчуванням, який CA налаштував для параметрів прив'язки до стандартного виконуваного файлу SDK.

Таблиця 6. Склад бібліотек заголовків для інтерфейсів, які спільно використовуються з кодом за межами SDK

Бібліотека	Призначення бібліотеки
boot_picobin_headers	Константи для формату PICO bin.
boot_picoboot_headers	Файл заголовка для USB-інтерфейсу PICOBOOT, що надається чіпом RP2xxx в режимі початкового завантаження.
boot_uf2_headers	Файл заголовка для формату UF2, підтримуваного мікросхемою RP2xxx в режимі BOOTS EL.
pico_usb_reset_interface_headers	Визначення інтерфейсу скидання, який може бути наданий бібліотекою pico_stdio_usb.

Розглянемо склад однієї з апаратних бібліотек API.

АЦП – Analog to Digital Converter (ADC)

Мікроконтролери серії RP мають внутрішній аналого-цифровий перетворювач послідовних наближень (АЦП) з такими функціями:

- частота дискретизації 500 кГц (з використанням незалежного годинника 48 МГц)
- ефективне число бітів (ENOB) у 12 біт АЦП RP2040 8.7, у RP2350 – 9.2
- 5 мультиплексованих входів у RP2040:
  - 4 входи, які доступні на виводах спільних з GPIO [29:26]
  - 1 вхід призначений для внутрішнього датчика температури
  - 4 елемент отримує вибірку FIFO
- мультиплексор RP2350 з 5 або 9 входами:
  - 4 входи доступні на виводах у корпусі QFN-60, які спільно використовуються з GPIO [29:26]
  - 8 входів, доступних на виводах у корпусі QFN-80, спільних з GPIO [47:40]
  - 8 елемент отримує вибірку FIFO
- генерація переривань
- інтерфейс DMA.

Хоча існує лише один АЦП, ви можете вказати вхідні дані для нього за допомогою функції adc_select_input(). У циклічному режимі (adc_set_round_robin()) АЦП використовуватиме цей вхід і переходить до наступного після читання.

RP2040, RP2350 у корпусі QFN-60:

- входи АЦП користувача 0-3 знаходяться на GPIO 26-29
- датчик температури знаходиться на вході 4.

RP2350 у корпусі QFN-80:

- входи АЦП користувача 0-7 знаходяться на GPIO 40-47
- датчик температури знаходиться на вході 8.

Значення датчика температури можна приблизно виразити в градусах Цельсія як:

$$T = 27 - (ADC\_Voltage - 0{,}706)/0{,}001721.$$

На рис. 2 наведено приклад коду для роботи з АЦП [1]. При роботі з джерелом [2] можна отримати приклад у вигляді тексту. Для цього потрібно натиснути на стрілочку у правому верхньому куті прикладу (рис. 3).

Рис. 3. Натиснувши на стрілочку, отримаємо приклад у вигляді тексту

```c
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "hardware/adc.h"

int main() {
    stdio_init_all();
    printf("ADC Example, measuring GPIO26\n");

    adc_init();

    // Make sure GPIO is high-impedance, no pullups etc
    adc_gpio_init(26);
    // Select ADC input 0 (GPIO26)
    adc_select_input(0);

    while (1) {
        // 12-bit conversion, assume max value == ADC_VREF == 3.3 V
        const float conversion_factor = 3.3f / (1 << 12);
        uint16_t result = adc_read();
        printf("Raw value: 0x%03x, voltage: %f V\n", result, result * conversion_factor);
        sleep_ms(500);
    }
}
```

Рис. 2. Приклад коду для роботи з АЦП

Фрагмент отриманого тектсу наведений нижче.

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "hardware/adc.h"

int main() {
    stdio_init_all();
    printf("ADC Example, measuring GPIO26\n");

    adc_init();
```

Функції АЦП

void adc_init (void)
Ініціалізуємо апаратне забезпечення АЦП.

static void adc_gpio_init (uint gpio)
Ініціалізуємо gpio для використання в якості виводу АЦП.

static void adc_select_input (введення користувачем)
Вибір вхідного каналу АЦП.

static uint adc_get_selected_input (void)
Повертає поточний вибраний канал введення АЦП.

static void adc_set_round_robin (uint input_mask)
Перемикач циклічної вибірки.

static void adc_set_temp_sensor_enabled (bool enable)
Вмикаємо вбудований датчик температури.

static uint16_t adc_read (void)
Виконуємо одноразове перетворення.

static void adc_run (bool run)
Вмикаємо або вимикаємо режим дискретизації в автономному режимі.

static void adc_set_clkdiv (float clkdiv)
Встановлюємо дільник тактової частоти АЦП.

static void adc_fifo_setup (bool en, bool dreq_en, uint16_t dreq_thresh, bool err_in_fifo, bool byte_shift)
Налаштовуємо FIFO АЦП.

static bool adc_fifo_is_empty (void)
Перевіряємо чи стан FIFO порожній.

static uint8_t adc_fifo_get_level (void)
Отримуємо кількість записів в FIFO АЦП.

static uint16_t adc_fifo_get (void)
Отримуємо результат перевірки АЦП з FIFO.

static uint16_t adc_fifo_get_blocking (void)
Очікуємо появу даних в АЦП FIFO.

static void adc_fifo_drain (void)
Розрядіть FIFO АЦП.

static void adc_irq_set_enabled (bool enabled)
Вмикаємо / вимикаємо переривання АЦП.

Детальна інформація про всі бібліотеки наведена у джерелах [2-9]. Приклади використання Raspberry Pi наведені у [11]. У прикладах наведено детальний опис

ЛІТЕРАТУРА

1.Raspberry Pi Documentation. Url: https://www.raspberrypi.com/documentation/pico-sdk/index_doxygen.html#raspberry-pi-pico-sdk
2.Hardware APIs. Url: https://www.raspberrypi.com/documentation/pico-sdk/hardware.html
3.High Level APIs. Url: https://www.raspberrypi.com/documentation/pico-sdk/high_level.html
4.Third-party Libraries. Url: https://www.raspberrypi.com/documentation/pico-sdk/third_party.html
5.TinyUSB. Url: https://docs.tinyusb.org/en/latest/
6.Networking Libraries. Url: https://www.raspberrypi.com/documentation/pico-sdk/networking.html
7.Libraries for BTstack. Url: https://bluekitchen-gmbh.com/btstack/
8.Lightweight IP stack. Url: https://www.nongnu.org/lwip/2_1_x/index.html
9.Runtime Infrastructure. Url: https://www.raspberrypi.com/documentation/pico-sdk/runtime.html
10.External API Headers. Url: https://www.raspberrypi.com/documentation/pico-sdk/misc.html
11.Raspberry Pi tutorials. Url: https://www.raspberrypi.com/tutorials/