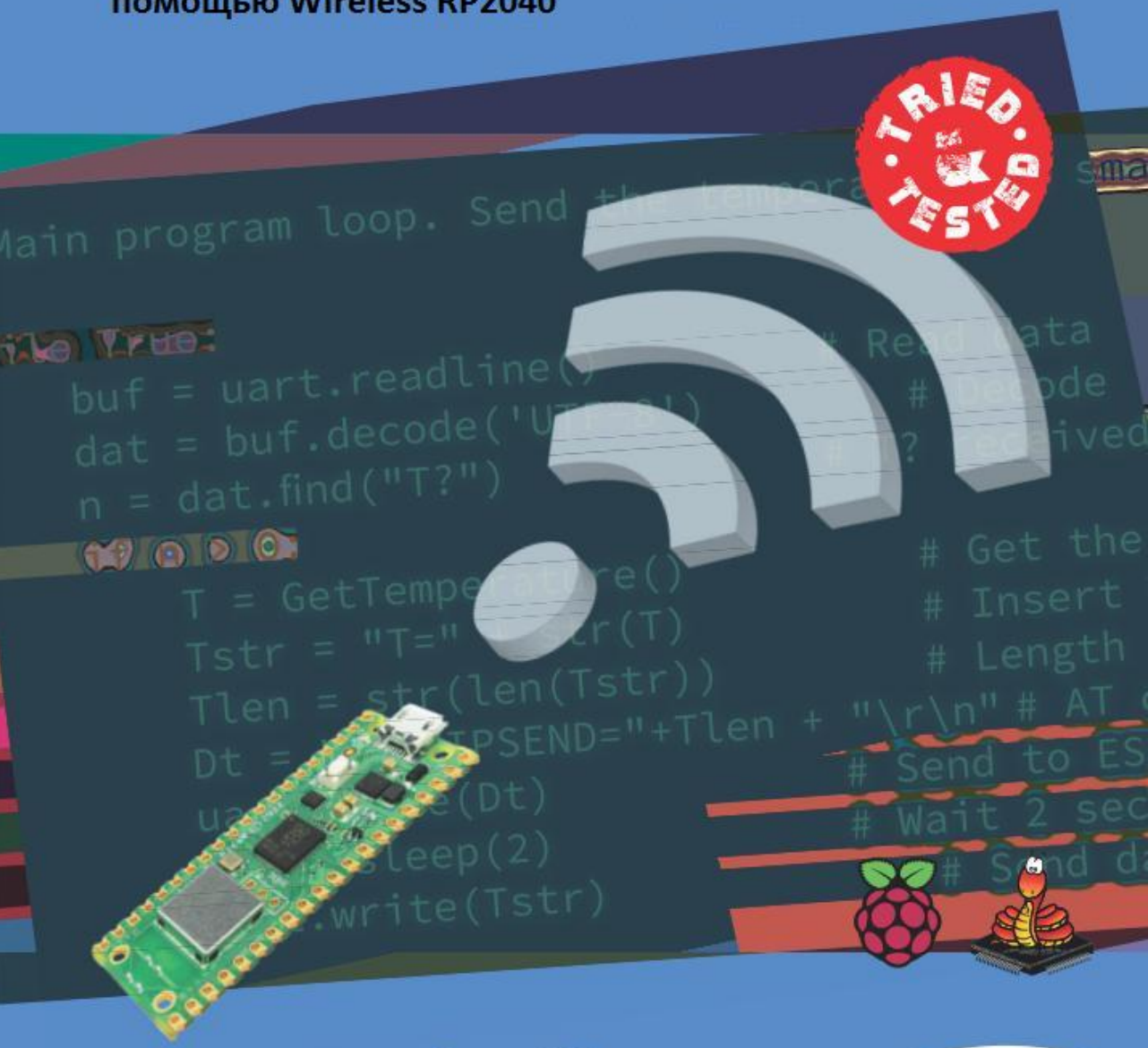
elektor books

60+ проектов на Raspberry Pi Pico W

Программируйте, создавайте и управляйте проектами с помощью Wireless RP2040



Доган Ибрагим

elektor
design > share > earn

Raspberry Pi Pico W

Программируйте, создавайте и управляйте проектами с помощью Wireless RP2040

Доган Ибрагим

Содержание

Предисловие

Микроконтроллер — это, по сути, однокристальный компьютер, включающий ЦП, память, схему ввода-вывода, таймеры, схему прерывания, схему часов и несколько других схем и модулей, размещенных в одном кремниевом кристалле. Ранние МК (МК) были ограничены в своих возможностях и скорости и потребляли значительную мощность. Большинство первых МК представляли собой 8-битные процессоры с тактовой частотой порядка нескольких мегагерц и имели всего сотни байт памяти для программ и данных. Эти МК традиционно программировались с использованием языков ассемблера целевых процессоров. Сегодня 8-битные МК все еще широко используются, особенно в небольших проектах, где большой объем памяти или высокая скорость не являются основными требованиями. С развитием технологии микросхем у нас теперь есть 32-разрядные и 64-разрядные МК со скоростями порядка нескольких гигагерц и несколькими гигабайтами памяти. В настоящее время МК программируются с использованием языков высокого уровня, таких как C, C#, BASIC, PASCAL, JAVA и т. д.

Raspberry Pi Pico — это высокопроизводительный микроконтроллер, разработанный специально для физических вычислений. Читатели должны понимать, что микроконтроллеры сильно отличаются от одноплатных компьютеров, таких как Raspberry Pi 4 (и других членов семейства Raspberry Pi). На Raspberry Pi Pico нет операционной системы. Микроконтроллеры, такие как Raspberry Pi Pico, можно запрограммировать на выполнение одной задачи, и их можно использовать в приложениях быстрого управления и мониторинга в реальном времени.

Raspberry Pi Pico основан на быстром и очень эффективном двухъядерном микроконтроллере ARM Cortex-M0+ RP2040, работающем на частоте до 133 МГц. Чип включает 264 КБ SRAM и 2 МБ флэш-памяти. Что делает Raspberry Pi Pico очень привлекательным, так это большое количество контактов GPIO и широко используемые модули периферийного интерфейса, такие как SPI, I2C, UART, PWM, а также быстрые и точные модули синхронизации.

Выпущенная в 2022 году плата Raspberry Pi Pico **W** является последней моделью семейства плат микроконтроллеров Pico. «Pico W» идентичен стандартному «Pico» с одним существенным отличием: по сравнению с другими членами семейства Pico, «Pico W» имеет встроенный модуль Wi-Fi, что позволяет использовать плату во многих коммуникациях. , контроль и особенно в проектах, основанных на IoT. Плата Pico W также поддерживает аппаратное обеспечение Bluetooth, но прошивка Bluetooth еще не была готова на момент написания этой книги.

Как стандартный Raspberry Pi Pico, так и Raspberry Pi Pico W можно легко запрограммировать с использованием некоторых популярных языков высокого уровня, таких как MicroPython или C/C++. В Интернете доступно множество заметок по применению, учебных пособий и спецификаций по использованию Pico или Pico W.

Эта книга представляет собой введение в использование платы разработки микроконтроллера Raspberry Pi Pico W с языком программирования MicroPython. Среда разработки Thonny (IDE) используется во всех проектах книги, и читателям рекомендуется использовать эту IDE. В книге есть много рабочих и протестированных проектов, охватывающих почти все аспекты Raspberry Pi Pico W. За исключением тем, связанных с Wi-Fi, все проекты в книге также можно использовать со стандартным Raspberry Pi Pico без каких-либо модификаций.

Следующие подзаголовки даны для каждого проекта, где это применимо, чтобы облегчить читателям понимание проектов:

- Заголовок
- Краткое описание
- Цель
- Блок-схема
- Принципиальная схема
- Листинг программы с полным описанием

Я надеюсь, что ваши следующие проекты на базе микроконтроллеров будут использовать Raspberry Pi Pico W, и что эта книга окажется полезной при разработке ваших проектов.

Профессор Доган Ибрагим
Лондон, 2022

Глава 1 • Raspberry Pi Pico W

1.1 Обзор

Raspberry Pi Pico W — это одноплатный модуль микроконтроллера, разработанный Raspberry Pi Foundation. Этот модуль основан на микросхеме RP2040. В этой главе вы подробно рассмотрите аппаратные детали модуля Raspberry Pi Pico W. Отныне этот микроконтроллерный модуль Pico W будет для краткости называться «Pico».

1.2 Аппаратный модуль Pico

Pico — это очень недорогой модуль за 6 долларов, основанный на микросхеме RP2040 с двухъядерным процессором Cortex-M0+. На рис. 1.1 показан вид спереди аппаратного модуля Pico, который представляет собой небольшую плату. В середине платы находится крошечный чип RP2040 размером 7 × 7 мм, помещенный в корпус QFN-56. По двум краям платы имеется 40 металлических контактов GPIO (общий ввод-вывод) золотистого цвета с отверстиями. Вы должны припаять штыревой разъём к этим отверстиям, чтобы можно было легко выполнить внешние подключения к плате. Отверстия отмечены, начиная с номера 1 в верхнем левом углу доски, и числа увеличиваются вниз до номера 40, который находится в верхнем правом углу доски. Плата совместима с макетной платой (т. е. расстояние между контактами 0,1 дюйма), и после пайки контактов плату можно подключить к макетной плате для простого подключения к контактам GPIO с помощью перемычек. Рядом с этими отверстиями вы увидите неровные круглые вырезы, которые можно вставлять поверх других модулей без установки каких-либо физических контактов.

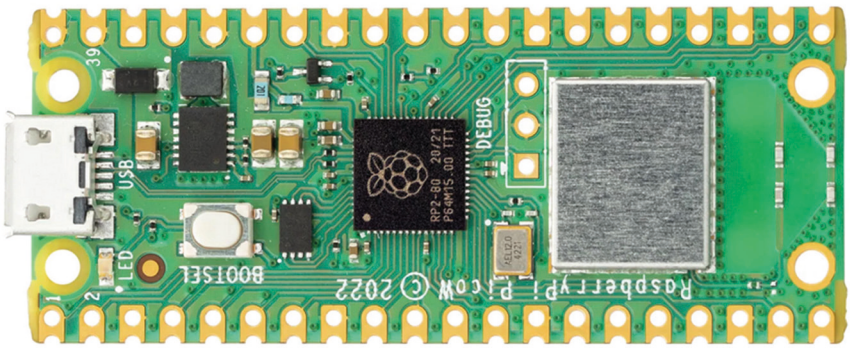


Рис.1.1: Аппаратный модуль Pico, вид спереди.

На одном краю платы находится порт micro-USB B для подачи питания на плату и для программирования платы. Рядом с USB-портом находится встроенный пользовательский светодиод, который можно использовать во время разработки программы. Рядом с этим светодиодом находится кнопка BOOT-SEL, которая используется во время программирования микроконтроллера. Рядом с чипом процессора есть 3 отверстия, к которым можно сделать внешние подключения. Они используются для отладки ваших программ с помощью Serial Wire Debug (SWD). На другом краю платы находится однодиапазонный модуль Wi-Fi 2,4 ГГц (802.11n). рядом с модулем Wi-Fi расположена бортовая антенна.

На рис. 1.2 показан вид сзади аппаратного модуля Pico. Здесь все контакты GPI обозначены буквами и цифрами. Вы заметите следующие типы букв и цифр: -

GND	- земля источника питания (цифровое земля)
AGND	- земля источника питания (аналоговая земля)
3V3	- 3V3 - питание +3,3 В (выход)
GP0 – GP22	- цифровые GPIO
GP26_A0 – GP28_A2	- аналоговые входы
ADC_VREF	- опорное напряжение АЦП
TP1 – TP6	- контрольные точки
SWDIO, GND, SWCLK	- интерфейс отладки
RUN	- вывод RUN по умолчанию. НИЗКИЙ уровень для сброса
3V3_EN	- по умолчанию включает питание +3,3 В.+3,3 В можно отключить, подключив этот контакт LOW
VSYS	- входное напряжение системы (от 1,8 В до 5,5 В), для создания питания +3,3 В для платы.
VBUS	- входное напряжение micro-USB (+5 В)

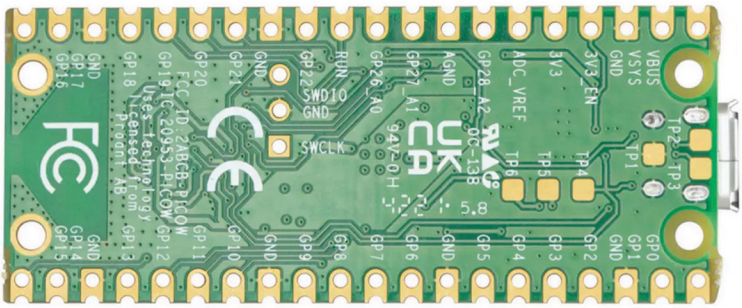


Рис.1.2: Аппаратный модуль Pico, вид сзади.

Некоторые контакты GPIO используются для внутренних функций платы. Это:

GP29 (input)	- используется в режиме АЦП (ADC3) для измерения VSYS/3
GP24 (input)	- GP24 (вход) — VBUS sense — ВЫСОКИЙ, если VBUS присутствует, иначе НИЗКИЙ
GP23 (output)	- управляет встроенным выводом энергосбережения SMPS.

Спецификации аппаратного модуля Pico следующие:

- 32-разрядный двухъядерный процессор RP2040 Cortex-M0+ с частотой 133 МГц
- 2 МБ флэш-памяти Q-SPI
- 264 КБ SRAM-памяти
- 26 GPIO (совместимость с +3,3 В)
- 3 контакта 12-битного АЦП
- Ускоренные встроенные библиотеки вычислений с плавающей запятой

- Встроенный однодиапазонный беспроводной чип Infineon CYW43439, беспроводной интерфейс 2,4 ГГц (802.11b/g/n) и Bluetooth 5.2 (не поддерживается на момент написания книги)
- Порт последовательной проводной отладки (SWD)
- Порт Micro-USB (USB 1.1) для питания (+5 В) и данных
- 2 × UART, 2 × I 2 C, 2 × интерфейс шины SPI
- 16 каналов ШИМ
- 1× Таймер (с 4 будильниками), 1× Счетчик реального времени
- Бортовой датчик температуры
- Встроенный светодиод на GPIO0, управляемый модулем 43439.
- Зазубренный модуль, позволяющий припаивать непосредственно к несущим платам
- 8 программируемых конечных автоматов ввода-вывода (PIO)
- Программирование на MicroPython, C, C++
- Drag & drop programming using mass storage over USB

Обратите внимание, что на плате Raspberry Pi Pico встроенный светодиод подключен к GP25 и доступен пользователю. На плате Raspberry Pi Pico W встроенный светодиод управляется модулем Wi-Fi 43439.

Аппаратное обеспечение Pico GPIO совместимо с +3,3 В, поэтому важно соблюдать осторожность, чтобы не превысить это напряжение при подключении внешних устройств ввода к контактам GPIO. Схема делителя напряжения +5 В в +3,3 В должна использоваться, если необходимо подключить устройства с выходами +5 В к контактам Pico GPIO.

На рис.1.3 показана схема делителя напряжения, которую можно использовать для понижения напряжения с +5 В до +3,3 В.

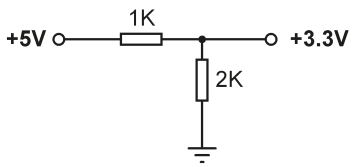


Рис.1.3: Схема делителя напряжения.

1.3: Схема делителя напряжения.

Arduino UNO — одна из самых популярных плат для разработки микроконтроллеров, используемая студентами, практикующими инженерами и любителями. В таблице 1.1 показано сравнение Raspberry Pi Pico W с Arduino UNO. Из этой таблицы видно, что Pico W намного быстрее, чем Arduino UNO, имеет больший объем флэш-памяти и памяти данных, предлагает Wi-Fi, имеет больше контактов цифрового ввода-вывода и имеет встроенный датчик температуры. Arduino UNO работает с +5 В, а его контакты GPIO совместимы с +5 В. Возможно, некоторые преимущества Arduino UNO заключаются в том, что он имеет встроенную память EEPROM, а его АЦП имеет 6 каналов вместо 3.

Feature	Raspberry Pi Pico W	Arduino UNO
Microcontroller	RP2040	Atmega328P
Core and bits	Dual core, 32-bits, Cortex-M0+	Single-core 8-bits
RAM	264Kbyte	2KByte
Flash	2MByte	32KByte
CPU speed	48MHZ to 133MHz	16MHz
EEPROM	None	1KByte
Wi-Fi support	CYW43439 wireless chip	None
Power input	+5V through USB port	+5V through USB port
Alternative power	2 – 5V via VSYS pin	7 – 12V
MCU operating voltage	+3.3V	+5V
GPIO count	26	20
ADC count	3	6
Hardware UART	2	1
Hardware I^2C	2	1
Hardware SPI	2	1
Hardware PWM	16	6
Programming languages	MicroPython, C, C++	C (Arduino IDE)
On-board LED	1	1
Cost	$6	$20

Таб. 1.1: Сравнение Raspberry Pi Pico W и Arduino UNO.

1.4 Условия эксплуатации и питание Pico

Рекомендуемые условия эксплуатации Pico:

- Рабочая температура: от -20 ºC до +85 ºC
- Напряжение VBUS: +5 В ±10 %
- Напряжение VSYS: от +1,8 В до +5,5 В

Встроенный импульсный источник питания +3,3 В используется для питания RP2040 в диапазоне входных напряжений от 1,8 В до +5,5 В. Например, 3 щелочные батареи типа AA могут использоваться для обеспечения +4,5 В для питания Pico.

Пико может питаться несколькими способами. Самый простой способ — использовать USB-порт компьютера или адаптер питания +5 В. Он подаст питание на вход VSYS (см. рис. 1.4) через диод Шоттки. Таким образом, напряжение на входе VSYS равно напряжению VBUS за вычетом падения напряжения на диоде Шоттки (около +0,7 В). Выводы VBUS и VSYS могут быть закорочены, если плата питается от внешнего USB-порта +5 В. Это немного увеличит входное напряжение и, следовательно, уменьшит пульсации на VSYS. Напряжение VSYS подается на SMPS через RT6150, который выдает фиксированные +3,3 В для MCU и других частей платы. VSYS делится на 3 и доступен через аналоговый входной порт GPIO29 (ADC3), который легко контролировать. GPIO24 проверяет наличие напряжения VBUS и имеет высокий логический уровень, если VBUS присутствует.

Другой способ питания Pico — подача внешнего напряжения (от +1,8 В до +5,5 В) непосредственно на вход VSYS (например, с помощью батарей или внешнего источника питания). Вы также можете использовать вход USB и входы VSYS вместе для подачи питания на Pico, например, для работы как от батарей, так и от порта USB. Если используется этот метод, то на входе VSYS следует использовать диод Шоттки, чтобы источники питания не мешали друг другу. Более высокие напряжения будут питать VSYS.

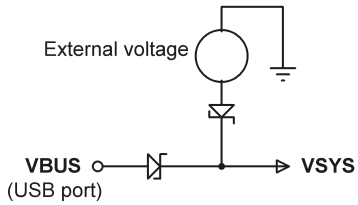


Рис.1.4: Питание Pico.

1.5 Распиновка МК RP2040 и модуля Pico

На рис. 1.5 показана разводка контактов МК RP2040, помещенного в 56-контактный корпус. Распиновка модуля Pico подробно показана на рис. 1.6. Как видно из рисунка, большинство выводов выполняют несколько функций. Например, GPIO0 (контакт 1) также является контактами UART0 TX, I2C0 SDA и SPI0 RX.

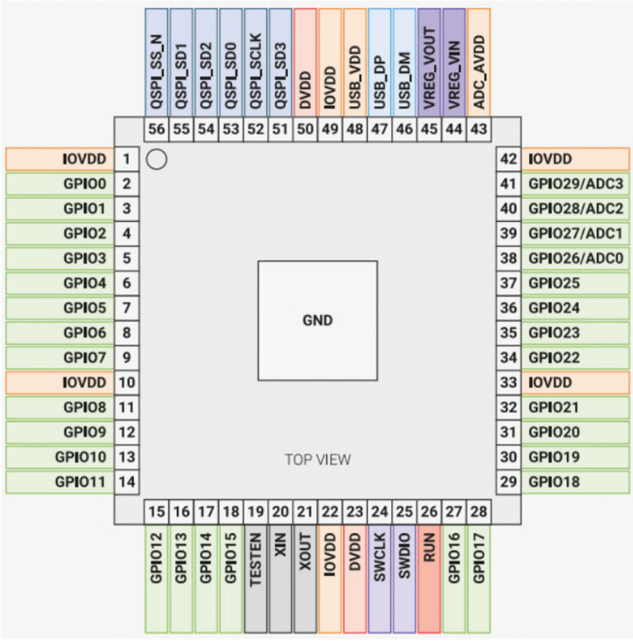


Рис.1.5: Распиновка микроконтроллера RP2040.

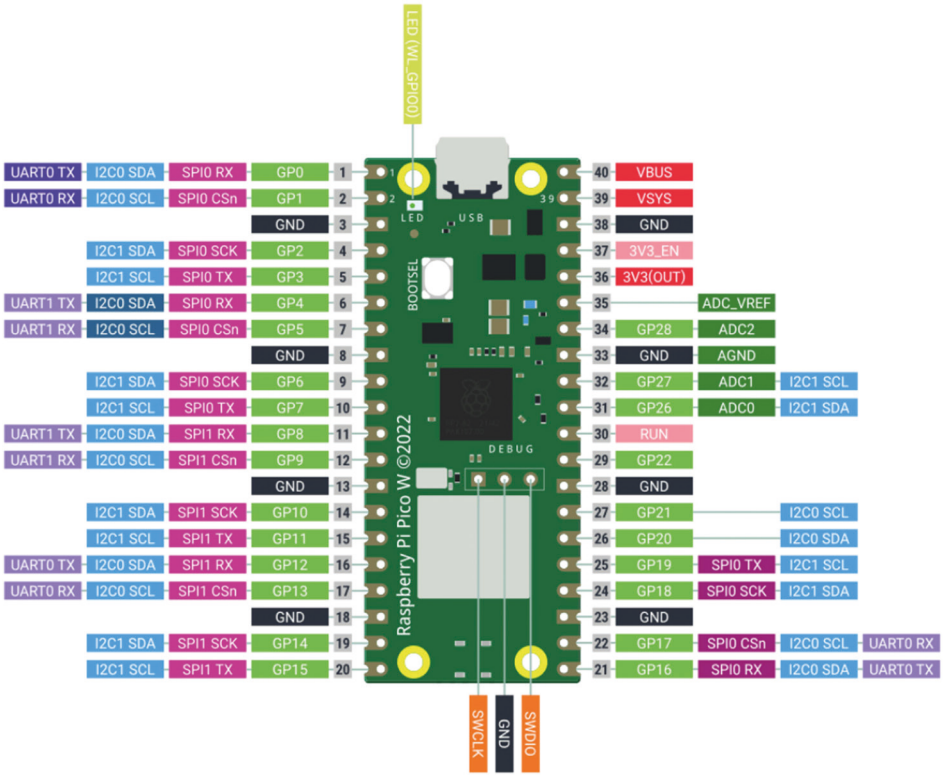


Рис.1.6: Распиновка Pico.

На рис.1.7 показана упрощенная блок-схема аппаратного модуля Pico. Обратите внимание, что контакты GPIO напрямую подключены от микросхемы МК к разъему GPIO. GPIO26-28 можно использовать либо как цифровой GPIO, либо как вход АЦП. Входы АЦП GPIO26-29 имеют обратные диоды на 3В, поэтому входное напряжение не должно превышать 3В3+300 мВ. Еще один момент, который следует отметить, это то, что если RP2040 не питается, подача напряжения на контакты GPIO26-29 может просачиваться через диод в источник питания (нет проблем с другими контактами GPIO, и напряжение может быть подано, когда RP2040 не запитан).

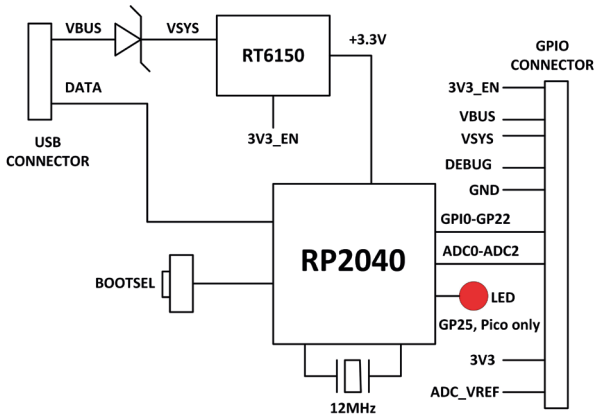


Рис.1.7: Упрощенная блок-схема.

1.6 Другие платы на базе МК RP2040

Существует множество других отладочных плат на базе микроконтроллеров RP2040. На момент написания этой книги представлены:

- Adafruit Feather RP2040
- Adafruit ItsyBitsy RP2040
- Adafruit QT Py RP2040
- Pimoroni PicoSystem
- Pimoroni Tufty 2040
- Arduino Nano RP2040 Connect
- SparkFun Thing Plus RP2040
- Pimoroni Pico Explorer Base
- Pimoroni Pico Lipo
- SparkFun MicroMod RP2040 Processor
- SparkFun Pro Micro RP2040
- Pico RGB Keypad Base
- Pico Omnibus
- Pimoroni Pico VGA Demo Base
- Cytron Maker Pi RP2040 development board
- Technoblogy – Minimal RP2040 board
- Pimoroni Tiny 2040
- Seeed Studio Wio RP2040 Mini development board
- Seeed XIAO RP2040 development board
- И многое другое

Глава 2 • Программирование Raspberry Pi Pico W

2.1 Обзор

На момент написания этой книги Raspberry Pi Pico W допускал программирование на следующих языках программирования:

- C/C++
- MicroPython
- Язык ассемблера

Хотя Pico по умолчанию настроен для использования с мощным и популярным языком C/C++, многим новичкам проще использовать MicroPython — версию языка программирования Python, разработанную специально для микроконтроллеров.

В этой главе вы узнаете, как установить и использовать язык программирования MicroPython. Вы будете использовать текстовый редактор Thonny, разработанный специально для программ на Python.

Многие работающие и полностью протестированные проекты будут представлены в следующих главах с использованием MicroPython с вашим Pico

2.2 Установка MicroPython на Pico W

Перед использованием платы необходимо установить MicroPython на Raspberry Pi Pico W. После установки MicroPython остается на вашем Pico, пока он не будет перезаписан чем-то другим. Для установки MicroPython требуется подключение к Интернету, и это требуется только один раз. Это можно сделать либо с помощью Raspberry Pi (например, Raspberry Pi 4), либо с помощью ПК. В этом разделе вы увидите, как выполнить установку с помощью ПК (например, Windows10).

Шаги следующие:

- Убедитесь, что ваш компьютер подключен к Интернету.
- Загрузите файл Raspberry Pi Pico W MicroPython UF2 в папку (например, Downloads) на вашем ПК по следующей ссылке. На момент написания этой книги файл имел имя: **rp2-pico-w-20220909-unstable-v1.19.1-389-g4903e48e3.uf2**.

https://www.raspberrypi.com/documentation/microcontrollers/micropython.html#drag-and-drop-micropython

- Нажмите и удерживайте кнопку **BOOTSEL** на Pico.
- Подключите Pico к порту USB вашего ПК с помощью кабеля micro-USB, удерживая кнопку.
- Подождите несколько секунд и отпустите кнопку **BOOTSEL**
- Вы должны увидеть, что Pico отображается как съемный диск с именем **RPI-RP2**, как показано на рис. 2.1 (диск **E**: в данном случае).

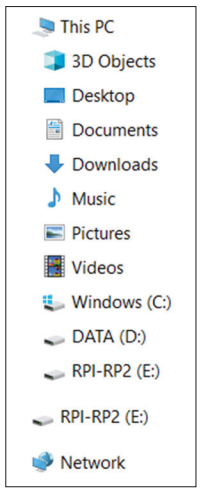


Рис.2.1: Pico как съемный диск RPI-RP2.

- Перетащите загруженный файл MicroPython UF2 на том RPI-RP2. Ваш Pico перезагрузится, и теперь вы используете MicroPython на своем Pico.
- Выключение Pico не удалит **MicroPython** из его памяти.

2.3 Использование текстового редактора Thonny с ПК

В этом разделе вы узнаете, как использовать Thonny на ПК для разработки и запуска ваших программ.

Во-первых, вы должны установить Thonny на свой компьютер (если он еще не установлен). Шаги:

- Перейдите на веб-сайт Thonny.org:

  https://thonny.org/

- Щелкните ссылку в правом верхнем углу экрана, чтобы установить Thonny (см. рис. 2.2).

Рис.2.2: Нажмите, чтобы установить Thonny.

- Вы должны увидеть значок на рабочем столе (рис. 2.3) вашего ПК. Дважды щелкните, чтобы запустить Тонни.

Рис.2.3: Значок Тонни на рабочем столе.

- Стартовый экран Thonny на вашем ПК показан на рис. 2.4.

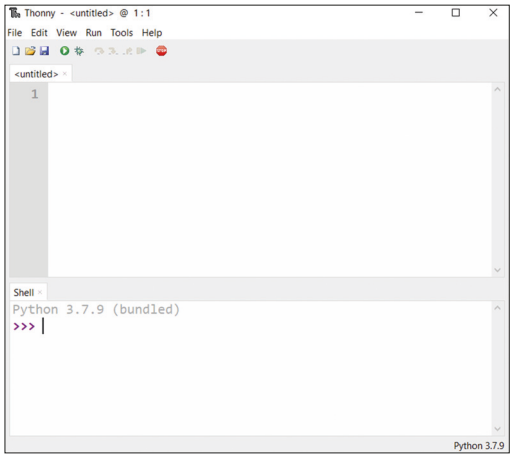


Рис.2.4: Стартовый экран Thonny на ПК.

- Щелкните метку **Python** в правом нижнем углу экрана и выберите **MicroPython** (Raspberry Pi Pico).
- Теперь вы готовы писать свои программы. Давайте напишем очень простое выражение Python для проверки установки MicroPython на вашем Pico.
- Введите следующий оператор в нижней части экрана (в **Shell**):

```
print("hello from Thonny on PC")
```

- Отобразится **hello from Thonny on PC**, как показано на рис. 2.5.

```
Type "help()" for more information.
>>>
>>> print("hello from Thonny on PC")
 hello from Thonny on PC
>>>
```

Рис. 2.5: Отображение сообщения.

В этой книге вы будете использовать Thonny на ПК для написания программ и их выполнения на Raspberry Pi Pico W.

В остальных разделах этой главы приведены примеры простых программ. Цель здесь состояла в том, чтобы рассмотреть основные концепции программирования Python. Эта книга не ставит своей целью обучение языку программирования Python. Заинтересованные читатели могут найти в Интернете множество книг и руководств по изучению языка программирования Python.

2.4 Написание программы с использованием Тонни

В предыдущем разделе вы узнали, как выполнить оператор онлайн с помощью Thonny **Shell**. Почти во всех приложениях вы должны писать программы. В качестве примера шаги по написанию и запуску очень простой однострочной программы для отображения сообщения «**Hello fromprogram... Привет из программы...**» приведены ниже:

- Введите операторы программы в верхней части экрана, как показано на рис. 2.6.

Рис.2.6: Напишите программу в верхней части экрана.

• Нажмите «**File**», затем «**Save As**» и дайте имя вашей программе. например, **FirstProg**. У вас есть возможность сохранить программу либо на вашем ПК, либо на Pico. Нажмите **Raspberry Pi Pico**, чтобы сохранить его на Pico (рис. 2.7). Введите имя вашей программы (**FirstProg**) и нажмите **OK** (обратите внимание, что файл сохранен с расширением **.py**).

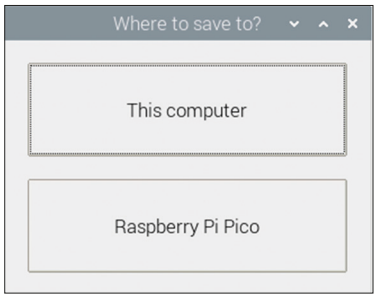


Рис 2.7: Нажмите **Raspberry Pi Pico**, чтобы сохранить вашу программу.

- Щелкните значок с зеленой стрелкой в верхней части экрана, чтобы запустить программу. Вывод программы будет отображаться в нижней части экрана **Shell**, как показано на рисунке 2.8.

```
>>> %Run -c $EDITOR_CONTENT
  Hello from program...
>>>
```

Рис.*2.8:* Вывод программы.

2.5 Только программа MicroPython, использующие Raspberry Pi Pico W

Пример 1: Среднее значение двух чисел, считанных с клавиатуры

В этом примере с клавиатуры считываются два числа и отображается их среднее значение. Цель этого примера — показать, как можно считывать данные с клавиатуры.

Решение 1

Программа называется **Average**, а листинг программы и пример выполнения программы показаны на рис. 2.9. Функция **input** используется для чтения чисел в виде строк с клавиатуры. Затем эти строки преобразуются в числа с плавающей запятой и сохраняются в переменных **n1** и **n2**. Среднее значение рассчитывается путем сложения, а затем деления чисел на два. Результат отображается на экране.

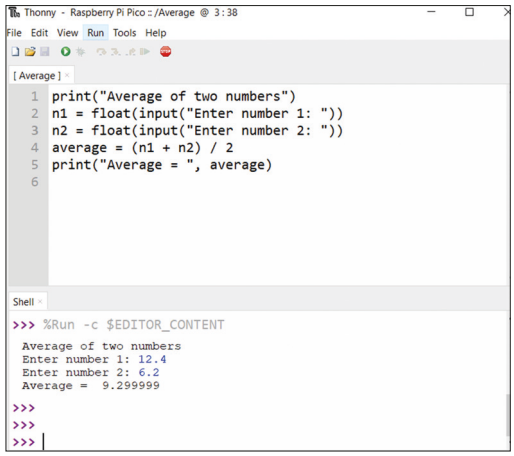


Рис.2.9: Программа: **Average** и пробный запуск

Пример 2: Среднее значение 10 чисел, считанных с клавиатуры

В этом примере с клавиатуры считываются 10 чисел и отображается их среднее значение. Цель этого примера — показать, как можно построить цикл в Python.

Решение 2

Программа называется **Average10**, а листинг программы и пример запуска программы показаны на рис. 2.10. В этой программе строится цикл, который выполняется от 0 до 9 (т. е. 10 раз). Внутри этого цикла числа считываются с клавиатуры, складываются друг с другом и сохраняются в переменной **sum**. Затем вычисляется среднее значение и отображается путем деления **sum** на 10. Обратите внимание, что новая строка не печатается после операторов печати, поскольку внутри оператора печати используется опция **end = ' '**.

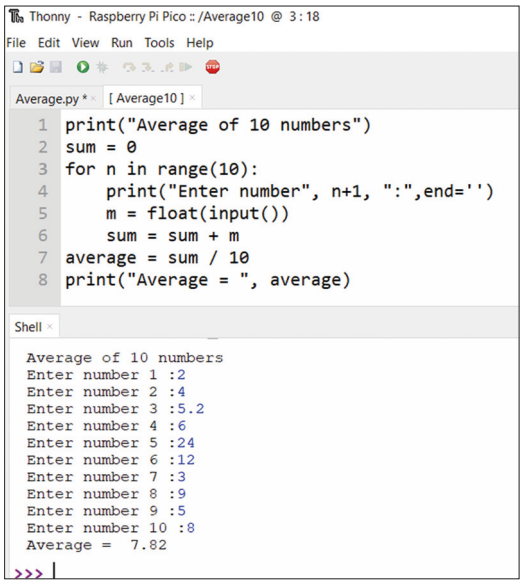


Рис.2.10: Программа: **Average10** и пробный запуск.

Пример 3: Площадь поверхности цилиндра
В этом примере радиус и высота цилиндра считываются с клавиатуры, а площадь его поверхности отображается на экране.

Решение 3

Программа называется **CylArea**, ее листинг и пример запуска программы показаны на рис. 2.11. Площадь поверхности цилиндра находится по формуле:

Площадь поверхности = $2\pi rh$

Где **r** и **h** - радиус и высота цилиндра соответственно. В эту программу импортирована библиотека **math**, поэтому в программе можно использовать функцию **Pi**. Площадь поверхности цилиндра отображается после чтения его радиуса и высоты.

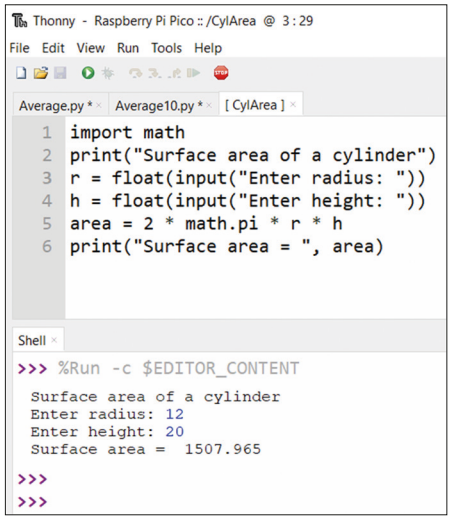


Рис. 2.11: Программа::**CylArea** и пробный запуск.

Пример 4: Преобразование ºC в ºF

В этом примере программа считывает градусы Цельсия с клавиатуры, преобразует и отображает эквивалентные градусы Фаренгейта.

Решение 4

Программа называется **CtoF**, а листинг программы и пример запуска программы показаны на рис. 2.12. Формула для преобразования ºC в ºF:

$$F = 1.8 \times C + 32$$

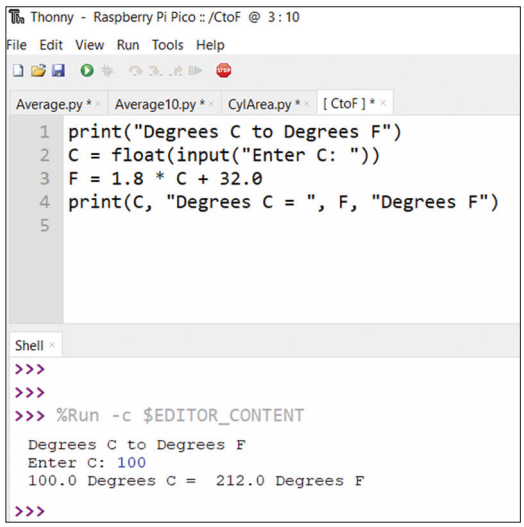


Рис. 2.12: Пример запуска программы **CtoFand**.

Пример 5: Площадь поверхности и объем цилиндра – функция пользователя
В этом примере вычисляются площадь поверхности и объем цилиндра, радиус и высота которого заданы. Программа использует функцию для расчета и возврата площади поверхности и объема.

Решение 5
Программа называется CylAreaSurf, листинг программы и пример запуска программы показаны на рис. 2.13. Площадь поверхности и объем цилиндра находятся по формуле:

Площадь поверхности = $2\pi rh$Volume = $\pi r^2 h$

Где r и h - радиус и высота цилиндра соответственно. Функция Calc используется для получения радиуса и высоты цилиндра. Функция возвращает в основную программу площадь поверхности и объем, которые отображаются на экране.

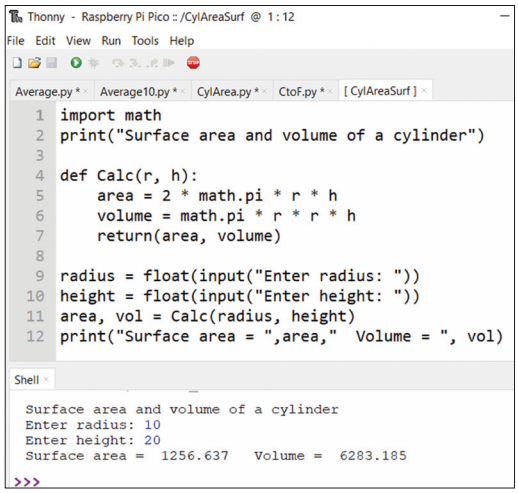


Рис.2.13: Программа: CylAreaSurf и пробный запуск.

Пример 6: Таблица квадратов чисел
В этом примере квадраты чисел от 1 до 11 вычисляются и табулируются.

Решение 6
Программа называется Squares, листинг программы и пример запуска программы показаны на рис. 2.14. Обратите внимание, что \t печатает вкладку, чтобы данные можно было удобно сгруппировать.

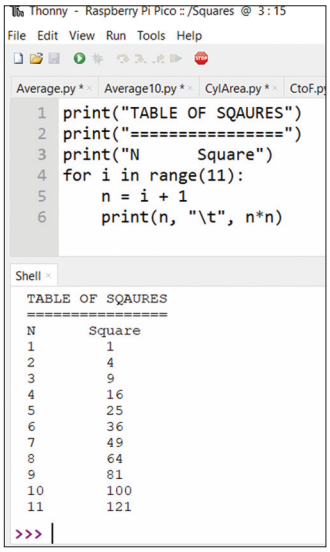


Рис.2.14: Программа: **Squares** и пробный запуск.

Пример 7: Таблица тригонометрического синуса
В этом примере тригонометрический синус табулируется от 0 до 45 градусов с шагом в 5 градусов.

Решение 7
Программа называется **Sines**, листинг программы и пример запуска программы показаны на рис. 2.15. Важно отметить, что аргументы тригонометрических функций должны быть в радианах, а не в градусах.

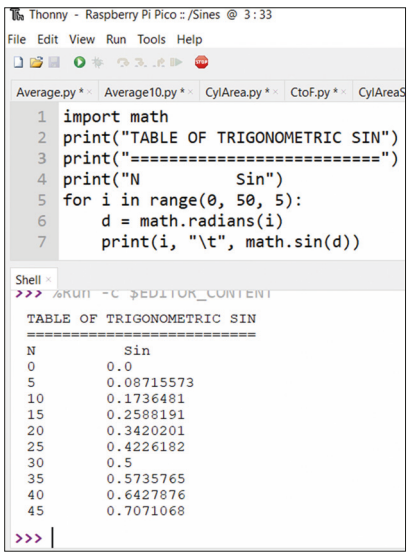


Рис.2.15: Программа: **Sines** и пример вывода.

Пример 8: Таблица тригонометрических синуса, косинуса и тангенса
В этом примере тригонометрические синус, косинус и тангенс сведены в таблицу от 0 до 45 градусов с шагом в 5 градусов.

Решение 8
Программа называется **Trig**, а листинг программы и пример запуска программы показаны на рис. 2.16.

Trig.py

```
1  import math
2  print("          TABLE OF TRIGONOMETRIC FUNCTIONS")
3  print("          ================================")
4  print("N\t   Sin\t\t  Cos\t\t   Tan")
5  for i in range(0, 50, 5):
6      d = math.radians(i)
7      s = math.sin(d)
8      c = math.cos(d)
9      t = math.tan(d)
10     print(i, "\t%9.7f\t%9.7f\t%9.7f" %(s,c,t))
```

Shell

```
          TABLE OF TRIGONOMETRIC FUNCTIONS
          ================================
N          Sin             Cos             Tan
0       0.0000000       1.0000000       0.0000000
5       0.0871557       0.9961946       0.0874887
10      0.1736481       0.9848078       0.1763270
15      0.2588191       0.9659258       0.2679492
20      0.3420201       0.9396926       0.3639702
25      0.4226182       0.9063078       0.4663076
30      0.5000000       0.8660254       0.5773502
35      0.5735765       0.8191521       0.7002075
40      0.6427876       0.7660444       0.8390996
45      0.7071068       0.7071068       1.0000000
```

Рис.2.16: Программа: Trig и образец выходных данных.

Пример 9: Тригонометрическая функция искомого угла
В этом примере угол считывается с клавиатуры. Кроме того, пользователь указывает, требуется ли синус (s), косинус (c) или тангенс (t) угла.

Решение 9
Программа называется **TrigUser**, а ее листинг и пример запуска программы показаны на рис. 2.17.

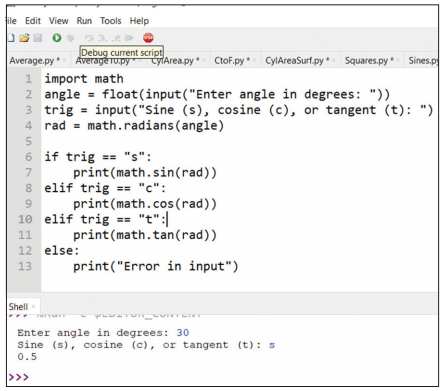


Рис.2.17: Программа: **TrigUser** и пример вывода.

Пример 10: последовательные и параллельные резисторы
Эта программа вычисляет общее сопротивление нескольких последовательно или параллельно соединенных резисторов. Пользователь указывает, будет ли соединение последовательным или параллельным. Кроме того, также указывается количество используемых резисторов в начале программы.

Решение 10
При последовательном соединении нескольких резисторов результирующее сопротивление равно сумме сопротивлений каждого резистора. Когда резисторы соединены параллельно, величина, обратная результирующему сопротивлению, равна сумме обратных сопротивлений каждого резистора.

На рис. 2.18 показан листинг программы (программа: **Serpal**). В начале программы отображается заголовок, и программа входит в цикл **while**. Внутри его цикла пользователю предлагается ввести количество резисторов в цепи и указать, подключены ли они последовательно или параллельно. Функция **str** преобразует число в эквивалентную ему строку. например, число 5 преобразуется в строку «5». Если соединение последовательное (режим равен '**s**'), то значение каждого резистора принимается с клавиатуры, а результирующая вычисляется и отображается на экране. Если, с другой стороны, соединение параллельное (режим равен «**p**»), то снова значение каждого резистора принимается с клавиатуры, а обратное число добавляется к сумме. Когда все значения резисторов введены, результирующее сопротивление отображается на экране.

```
print("RESISTORS IN SERIES OR PARALLEL")
print("==============================")
yn = "y"

while yn == 'y':
  N = int(input("\nHow many resistors are there?: "))
  mode = input("Are the resistors series (s) or parallel (p)?: ")
```

```
  mode = mode.lower()
#
# Прочтите номиналы резисторов и рассчитайте общее сопротивление
#
  resistor = 0.0

  if mode == 's':
    for n in range(0,N):
       s = "Enter resistor " + str(n+1) + " value in Ohms: "
       r = int(input(s))
       resistor = resistor + r
    print("Total resistance = %d Ohms" %(resistor))

  elif mode == 'p':
    for n in range(0,N):
      s = "Enter resistor " + str(n+1) + " value in Ohms: "
      r = float(input(s))
      resistor = resistor + 1 / r
    print("Total resistance = %.2f Ohms" %(1 / resistor))
#
# Проверьте, хочет ли пользователь выйти
#
  yn = input("\nDo you want to continue?: ")
  yn = yn.lower()
```

Рис.*2.18:* Программа*: **Serpal.***

На рис.2.19 показан типичный запуск программы.

```
RESISTORS IN SERIES OR PARALLEL
===============================

How many resistors are there?: 3
Are the resistors series (s) or parallel (p)?: s
Enter resistor 1 value in Ohms: 100
Enter resistor 2 value in Ohms: 150
Enter resistor 3 value in Ohms: 200
Total resistance = 450 Ohms

Do you want to continue?: y

How many resistors are there?: 2
Are the resistors series (s) or parallel (p)?: p
Enter resistor 1 value in Ohms: 100
Enter resistor 2 value in Ohms: 100
Total resistance = 50.00 Ohms

Do you want to continue?: n
```

Рис.2.19. Типичный запуск программы.

Пример 11: Слова в обратном порядке

Напишите программу, которая считывает слово с клавиатуры, а затем отображает буквы этого слова в обратном порядке на экране.

Решение 11

Листинг программы показан на рис. 2.20 (программа: **Letters**). Слово считывается с клавиатуры и сохраняется в строковой переменной **word**. Затем буквы этого слова отображаются в обратном порядке. Пример запуска программы показан на рис. 2.20.

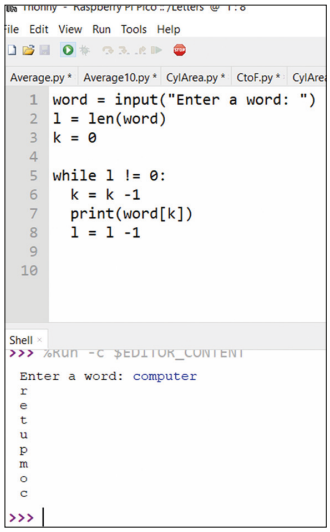


Рис.2.20: Программа: **Letters** и пример вывода.

Пример 12: Калькулятор

Напишите программу-калькулятор для выполнения четырех простых математических операций сложения, вычитания, умножения и деления над двумя числами, полученными с клавиатуры.

Решение 12

Листинг программы показан на рис.2.21 (программа: **Calc**). Два числа принимаются с клавиатуры и сохраняются в переменных **n1** и **n2**. Затем выполняется требуемая математическая операция. Результат, сохраненный в переменной **result**, отображается на экране. Пользователю предоставляется возможность завершить программу.

```
any = 'y'
while any == 'y':
   print("\nCalculator Program")
   print("==================")

   n1 = float(input("Enter first number: "))
   n2 = float(input("Enter second number: "))
   op = input("Enter operation (+-*/): ")

   if op =="+":
```

```
        result = n1 + n2
    elif op == "-":
        result = n1 - n2
    elif op == "*":
        result = n1 * n2
    elif op == "/":
        result = n1 / n2
    print("Result = %f" %(result))
    any = input("\nAny more (yn): ")
```

Рис.2.21: Программа:**Calc**.

Пример запуска программы показан на рис. 2.22.

```
Calculator Program
==================
Enter first number: 25
Enter second number: 2
Enter operation (+-*/): *
Result = 50.000000

Any more (yn): y

Calculator Program
==================
Enter first number: 12
Enter second number: 2
Enter operation (+-*/): /
Result = 6.000000

Any more (yn): n
>>>
```

Рис.2.22: Пример запуска программы.

Пример 13: Обработка файла — запись
В этом примере будет создан текстовый файл с именем **MyFile.txt** и текст **Hello from Raspberry Pi Pico!** будет записано в этот файл.

Решение 13
Программа называется **Filew**, ее листинг и пример выполнения показаны на рис.2.23. Файл открывается в режиме записи (**w**) и в него записывается текст с помощью функции **write**. Обратите внимание, что **fp** — это дескриптор файла.

```
print("Open the file and write the text")
fp = open("MyFile.txt", "w")
fp.write("Hello from Raspberry Pi Pico!\n")
fp.close()
print("End of file operation")
```

```
Shell
Open the file and write the text
End of file operation
```

Рис.2.23: Программа: **Filew**.

Пример 14: Обработка файла – чтение
В этом примере текстовый файл **MyFile.txt**, созданный в предыдущем примере, открывается, и его содержимое отображается на экране.

Решение 14
Программа называется **Filer**, ее листинг и пример запуска показаны на рис. 2.24. Файл открывается в режиме чтения (**r**) и отображается его содержимое.

```
print("Open the file and read its contents")
fp = open("MyFile.txt", "r")
str = fp.read(80)
fp.close()
print(str)
```

```
hell
Open the file and read its contents
Hello from Raspberry Pi Pico!
```

Рис.2.24: Программа:Filer.

Пример 15: Квадраты и кубы чисел
Напишите программу для табулирования квадратов и кубов чисел от 1 до 10.

Решение 15
Программа называется **Cubes**, и ее листинг в качестве примера показан на рис. 2.25.

```
1  print("Squares and cubes of numbers")
2  print(' N  N*N  N*N*N')
3  for i in range(1,11):
4      print('{0:2d} {1:3d} {2:4d}'.format(i, i*i, i*i*i))
5
```

Shell

```
Squares and cubes of numbers
 N  N*N  N*N*N
 1    1    1
 2    4    8
 3    9   27
 4   16   64
 5   25  125
 6   36  216
 7   49  343
 8   64  512
 9   81  729
10  100 1000
```

Рис.2.25: Программа: **Cubes** и пример вывода.

Пример 16: Таблица умножения
Напишите программу, которая считывает число с клавиатуры, а затем отображает результирующую для этого числа от 1 до 12.

Решение 16
Программа называется Times, а ее листинг и пример выполнения показаны на рис.2.26.

```
1  num = int(input("Enter a number: "))
2  print("Timetable of number ", num)
3  for i in range(1, 13):
4     print(num, 'x', i, '=', num*i)
```

Shell

```
Enter a number: 5
Timetable of number  5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
5 x 11 = 55
5 x 12 = 60
```

Рис.2.26: Программа: **Times** и пример вывода.

Пример 17: Нечетное или четное
Напишите программу, которая считывает число с клавиатуры, проверяет и отображает, является ли это число четным или нечетным.

Решение 17

Программа называется **OddEven**, ее листинг и пример выполнения показаны на рис. 2.27.

```
num = int(input("Enter a number: "))
if (num % 2) == 0:
   print("Number {0} is Even".format(num))
else:
   print("Number {0} is Odd".format(num))
```

```
Enter a number: 15
Number 15 is Odd
```

Рис.2.27: Программа: **OddEven** и пример вывода.

Пример 18: Двоичный, восьмеричный и шестнадцатеричный

Напишите программу для чтения десятичного числа с клавиатуры. Преобразуйте это число в двоичное, восьмеричное и шестнадцатеричное и отобразите на экране.

Решение 18

Программа называется Conv, ее листинг и пример запуска показаны на рис. 2.28.

```
dec = int(input("Enter a number: "))
print("The decimal value of", dec, "is:")
print("in binary: ",bin(dec))
print("in octal: ",oct(dec))
print("in hexadecimal: ",hex(dec))
```

```
Enter a number: 254
The decimal value of 254 is:
in binary:  0b11111110
in octal:  0o376
in hexadecimal:  0xfe
```

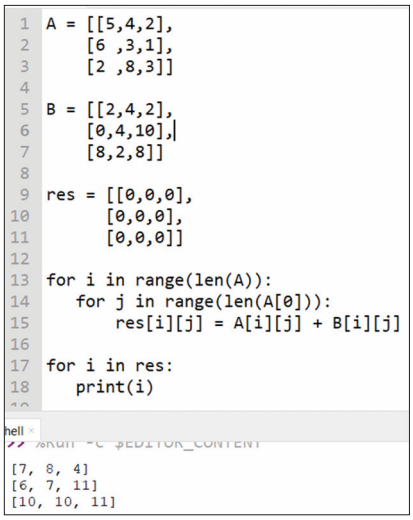
Рис.2.28: Программа: **Conv** и пример вывода.

Пример 19: Добавьте две матрицы

Напишите программу, которая складывает две заданные матрицы и отображает элементы новой матрицы.

Решение 19

Программа называется **AddMatrix**, ее листинг и пример выполнения показаны на рис. 2.29.



```
A = [[5,4,2],
     [6 ,3,1],
     [2 ,8,3]]

B = [[2,4,2],
     [0,4,10],
     [8,2,8]]

res = [[0,0,0],
       [0,0,0],
       [0,0,0]]

for i in range(len(A)):
   for j in range(len(A[0])):
       res[i][j] = A[i][j] + B[i][j]

for i in res:
   print(i)
```

```
[7, 8, 4]
[6, 7, 11]
[10, 10, 11]
```

Рис.2.29: Программа: **AddMatrix** и пример вывода.

Пример 20: Shapes
Напишите программу, которая использует функции для вычисления и отображения площадей фигур: квадрата, прямоугольника, треугольника, круга и цилиндра. Размеры требуемых сторон должны быть получены с клавиатуры.

Решение 20
Области площадей, которые будут использоваться в программе, следующие:

Square: сторона = a	area = a^2
Rectangle: сторона a, b	area = ab
Circle: радиус r	area = πr^2
Triangle: основание b, высота h	area = bh/2
Cylinder: радиус r, высота h	area = 2πrh

Листинг программы показан на рис. 2.30 (программа: **areas**). Для каждой формы используется своя функция, а размеры сторон получаются внутри функций. Основная программа отображает рассчитанную площадь для выбранной формы.

```
#------------------------------------------------------------
#                ПЛОЩАДИ ФИГУР
#                ==============
#
# Эта программа вычисляет и отображает площади
# различных геометрических фигур в виде списка
#
# Автор: Доган Ибрагим
# Файл   : areas.py
# Дата: октябрь 2022 г.
```

```
#-------------------------------------------------------------
import math

def Square(a):                      # квадрат
  return a * a

def Rectangle(a, b):                # прямоугольник
  return(a * b)

def Triangle(b, h):                 # треугольник
  return(b * h / 2)

def Circle(r):                      # круг
  return(math.pi * r * r)

def Cylinder(r, h):                 # цилиндр
  return(2 * math.pi * r * h)

print("AREAS OF SHAPES")
print("===============\n")
print("What is the shape?: ")

shape = input("Square (s)\nRectangle(r)\nCircle(c)\n\
Triangle(t)\nCylinder(y): ")

shape = shape.lower()
if shape == 's':
  a = float(input("Enter a side of the square: "))
  area = Square(a)
  s = "Square"
elif shape == 'r':
  a = float(input("Enter one side of the rectangle: "))
  b = float(input("Enter other side of the rectangle: "))
  area = Rectangle(a, b)
  s = "Rectangle"
elif shape == 'c':
  radius = float(input("Enter radius of the circle: "))
  area = Circle(radius)
  s = "Circle"
elif shape == 't':
  base = float(input("Enter base of the triangle: "))
  height = float(input("Enter height of the triangle: "))
  area = Triangle(base, height)
  s = "Triangle"
elif shape == 'y':
  radius = float(input("Enter radius of cylinder: "))
```

```
  height = float(input("Enter height of cylinder: "))
  area = Cylinder(radius, height)
  s = "Cylinder"

print("Area of %s is %f" %(s, area))
```

Рис.*2.30:* Программа:***areas***

Пример запуска программы показан на рис.2.31.

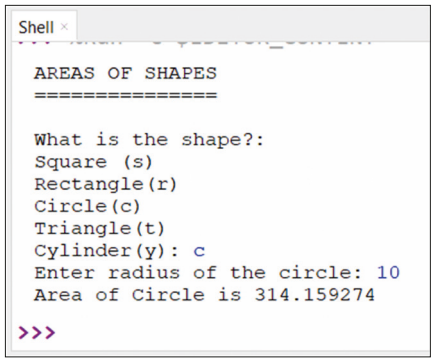


Рис.2.31: Пример запуска программы

Пример 21: Решение квадратного уравнения
Напишите программу для нахождения корней квадратного уравнения вида:

$$ax^2 + bx + c = 0$$

Решение 21
Листинг программы показан на рис. 2.32 (программа: **quadratic**). Коэффициенты **a**, **b** и **c** задаются в начале программы. Два корня вычисляются и отображаются на экране.

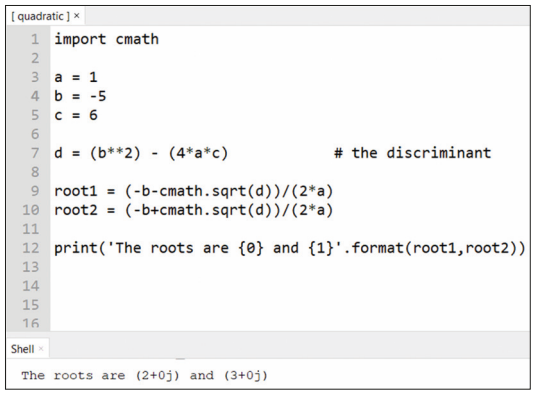


Рис.2.32: Программа: quadratic и пример вывода.

Пример 22: Умножение матриц
Напишите программу для умножения двух матриц.

Решение 22
ТЭлементы двух матриц указываются в начале программы.
Рис.2.33 показывает листинг программы (программа**: multmatrix**).

```
[ multmatrix ] ×
 1  A = [[5,2,3],
 2       [2 ,1,3],
 3       [2 ,4,3]]
 4
 5  B = [[2,2,1,2],
 6       [3,2,3,0],
 7       [0,2,4,1]]
 8
 9  result = [[0,0,0,0],
10            [0,0,0,0],
11            [0,0,0,0]]
12
13  for i in range(len(A)):
14     for j in range(len(B[0])):
15         for k in range(len(B)):
16             result[i][j] += A[i][k] * B[k][j]
17
18  for i in range(3):
19    for j in range(4):
20       print(result[i][j], end="")
21       print(" ", end="")
22    print("\n")
Shell ×
 16 20 23 13

 7 12 17 7

 16 18 26 7
```

Рис.2.33: Программа: multmatrix и пример вывода.

Пример 23: Факториал числа
Напишите программу для вычисления факториала числа, введенного с клавиатуры.

Решение 23
Листинг программы показан на рис. 2.34 (программа: **factorial**).

```
[ factorial ] ×
 1  numbr = int(input("Enter a number: "))
 2  factorial = 1
 3
 4  # check if the number is negative, zero, or positive
 5  if numbr < 0:
 6     print("Negative numbers do not have factorials")
 7  elif numbr == 0:
 8     print("The factorial of 0 is 1")
 9  else:
10     for i in range(1, numbr + 1):
11         factorial = factorial*i
12     print("The factorial of",numbr,"is",factorial)
13
Shell ×
>>> %Run -c $EDITOR_CONTENT
 Enter a number: 5
 The factorial of 5 is 120
>>>
```

Рис.2.34: Программа: **factorial** и пример вывода.

Пример 24: Сложные проценты
Напишите программу для расчета сложных процентов по начальному значению, процентной ставке и количеству лет.

Решение 24
Сложные проценты рассчитываются по формуле:

$$FV = IV(1 + i / 100) ** yr$$

$$\text{Сложные проценты} = FV - IV$$

Где **FV** и **IV** — будущие и начальные значения, **i** — процентная ставка, а **yr** — количество лет.

На рис. 2.35 показан листинг программы (программа: **compound**).

```
[ compound ] ×
 1  def compound_interest(initial, rate, years):
 2
 3      FV = initial * (pow((1 + rate / 100), years))
 4      interest = FV - initial
 5      print("Compound interest is: ", interest)
 6
 7  IV = float(input("Enter initial value: "))
 8  i = float(input("Enter interest rate :"))
 9  yr = float(input("Enter number of years: "))
10
11  compound_interest(IV, i, yr)
12
13
Shell ×
>>> %Run -c $EDITOR_CONTENT
 Enter initial value: 1000
 Enter interest rate :10
 Enter number of years: 2
 Compound interest is:  210.0
>>>
```

Рис. 2.35: Программа: compound и пример вывода.

Пример 25: Угадай число
Напишите программу, которая генерирует секретное число от 1 до 50 и позволяет пользователю угадать это число за 5 попыток.

Решение 25
Листинг программы показан на рис.2.36 (программа: **guess**).

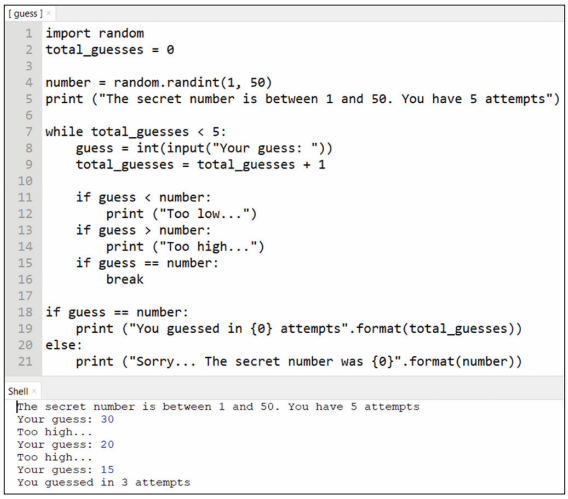


Рис.2.36: Программа: `guess` и пример вывода.

Пример 26: Численное интегрирование
Напишите программу, которая считывает с клавиатуры функцию, вычисляет и отображает интеграл этой функции между двумя заданными точками.

Решение 26

Листинг программы показан на рис. 2.37 (программа: **integrate**). Интеграция выполняется внутри функции **integrate**. Эта функция делится на равные небольшие сегменты между требуемыми нижним и верхним пределами. Вычисляется площадь каждого сегмента. Суммарная площадь равна искомому интегралу функции. Аргументами этой функции являются функция, которую нужно интегрировать, нижний и верхний пределы, а также количество учитываемых точек.

```
def integrate(func, xlow, xhigh, n):
    length = xhigh - xlow
    sect = length / n
    area = 0
    for j in range(n):
        x = xlow + sect * j
        y = eval(func.replace("x", str(x)))
        secarea = y * sect
        area = area + abs(secarea)
    return (area)

function = input("Enter function: ")
lowl = float(input("Low limit:" ))
highl = float(input("High limit:"))
N = int(input("No of points: "))
```

```
r = integrate(function, lowl, highl, N)
print("integral of %s between %5.2f and %5.2f = %6.2f" %(function,lowl,highl,r))
```

Рис.*2.37:* Программа **integrate.**

На рис. 2.38 показан пример запуска программы, в которой следующий интеграл вычисляется с использованием 100 сегментов, а результат отображается на экране Тонни:

$$\int_0^1 (x^2 + 2)dx$$

```
Enter function: x**2 + 2
Low limit:0
High limit:1
No of points: 100
integral of x**2 + 2 between  0.00 and  1.00 =   2.33
```

Рис.2.38: Пример запуска программы.

Пример 27: Практика арифметики
Напишите программу для отображения следующего меню:

1. Addition
2. Subtraction
3. Multiplication
4. Division

1. Сложение
2. Вычитание
3. Умножение
4. Деление

Выбор:
Сгенерируйте два случайных целых числа от 1 до 1000 и попросите пользователя выполнить требуемую операцию. Проверьте ответ пользователя и отобразите CORRECT - ПРАВИЛЬНО или INCORRECT - НЕПРАВИЛЬНО.

Решение 27

На рис. 2.39 показана требуемая программа (Программа:**arithmetic**). Функции используются для сложения, вычитания, умножения и деления. Введенный пользователем номер проверяется, и отображается соответствующее сообщение.

```
import random
def addition():
    n1 = random.randint(1, 1000)
    n2 = random.randint(1, 1000)
    print(n1, "+", n2, "= ")
    user_answer = int(input("Enter answer: "))
    correct_answer = n1 + n2
    return (user_answer, correct_answer)

def subtraction():
    n1 = random.randint(1, 1000)
```

```
    n2 = random.randint(1, 1000)
    print(n1, "-", n2, "= ")
    user_answer = int(input("Enter answer: "))
    correct_answer = n1 - n2
    return (user_answer, correct_answer)

def multiplication():
    n1 = random.randint(1, 1000)
    n2 = random.randint(1, 1000)
    print(n1, "X", n2, "= ")
    user_answer = int(input("Enter answer: "))
    correct_answer = n1 * n2
    return (user_answer, correct_answer)

def division():
    n1 = random.randint(1, 1000)
    n2 = random.randint(1, 1000)
    print(n1, "/", n2, "= ")
    user_answer = int(input("Enter answer: "))
    correct_answer = int(n1 / n2)
    return (user_answer, correct_answer)

def chk(user, correct):
    if user ==correct:
        print("CORRECT")
    else:
        print("INCORRECT")

print("   1. Addition")
print("   2. Subtraction")
print("   3. Multiplication")
print("   4. Integer Division")
ch = int(input("Choice: "))
if ch == 1:
    user,correct = addition()
elif ch == 2:
    user,correct = subtraction()
elif ch == 3:
    user,correct = multiplication()
else:
    user,correct = division()
chk(user,correct)
```

Рис.*2.39:* Программа: **arithmetic**

На рис. 2.40 показан пример запуска программы.

```
    1. Addition
    2. Subtraction
    3. Multiplication
    4. Division
Choice: 3
156 X 298 =
Enter answer: 46488
CORRECT
```

Рис.2.40: Пример запуска.

Глава 3 • Светодиодные проекты Raspberry Pi Pico W

3.1 Обзор

В этой главе вы будете разрабатывать простые аппаратные проекты с Raspberry Pi Pico W, используя текстовый редактор Thonny. Следующие подзаголовки будут даны для каждого проекта, где это применимо:

- Заголовок
- Описание
- Цель
- Блок-схема
- Принципиальная схема
- Листинг программы
- Предложения для будущей работы

Все программы в этой главе были разработаны с использованием Thonny, работающего на ПК, где Raspberry Pi Pico W был подключен к USB-порту ПК. Для создания проектов использовалась беспаечная макетная плата.

3.2 Проект 1: Внешний мигающий светодиод

Описание: В этом проекте к Pico подключен внешний светодиод, который мигает каждую секунду.

Цель: Цель этого проекта — показать, как внешний светодиод можно подключить к Pico.

Блок-схема: На рис. 3.1 показана блок-схема проекта.

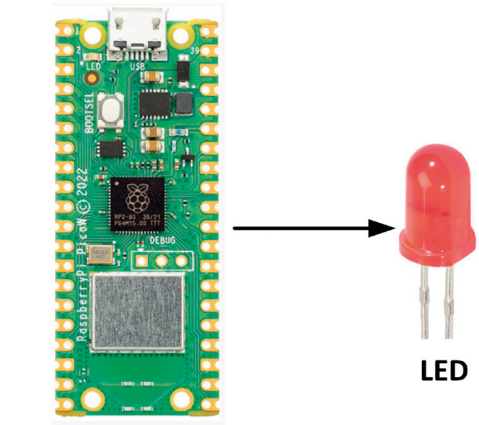


Рис.3.1: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рисунке 3.2, где светодиод подключен к выводу GP1 порта Pico через токоограничивающий резистор на 470 Ом.

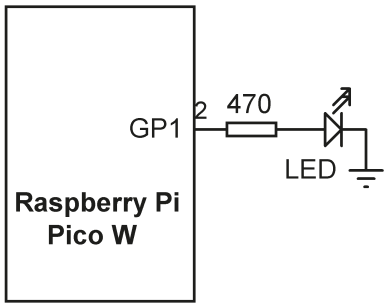


Рис.3.2: Принципиальная схема проекта.

Светодиод может быть подключен как в режиме источника тока, так и в режиме приема тока. В режиме источника тока (рис. 3.3) светодиод включается, когда на вывод порта подается логический HIGH - ВЫСОКИЙ уровень. В режиме стока тока (рис. 3.4) светодиод включается, когда на вывод порта подается логический LOW - НИЗКИЙ уровень.

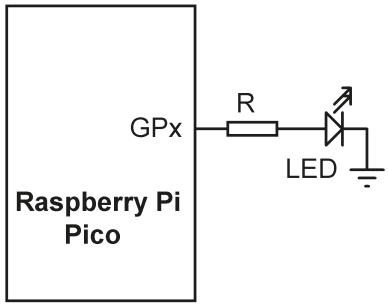


Рис.3.3: Светодиод в режиме источника тока.

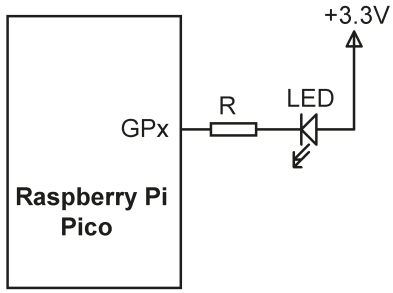


Рис.3.4: Светодиод в режиме потребления тока.

Требуемое значение токоограничивающего резистора можно рассчитать следующим образом:

В режиме источника тока, предполагая, что выходное напряжение HIGH равно +3,3 В, падение напряжения на светодиоде равно 2 В, а ток через светодиод равен 3 мА, требуемое значение токоограничивающего резистора составляет:

R = (3,3 – 2) / 3 = 433 Ом, выбирается ближайший резистор 470 Ом.

Листинг программы: Рисунок 3.5 показывает листинг программы (Программа: ExtFlash.py). В начале программы светодиод на GP1 настроен как выход. Затем формируется цикл while, который выполняется до тех пор, пока пользователь не остановит его. Внутри этого цикла светодиод включается и выключается с задержкой в одну секунду .

```
#------------------------------------------------------------
#            МИГАЮЩИЙ ВНЕШНИЙ СВЕТОДИОД
#            ==============================
#
# В этой программе внешний светодиод подключен к пину порта
# GP1 (вывод 2). Светодиод мигает каждую секунду
#
# Автор: Доган Ибрагим
# Файл   : ExtFlash.py
# Дата: октябрь 2021 г.
#------------------------------------------------------------
from machine import Pin
import utime

LED = Pin(1, Pin.OUT)          # LED на GP1

while True:                    # ДЕЛАТЬ ВСЕГДА
    LED.value(1)               # LED ON
    utime.sleep(1)             # ждать 1 секунду
    LED.value(0)               # LED ВЫКЛ.
    utime.sleep(1)             # ждать 1 секунду
```

Рис.*3.5:* Программа*:* **ExtFlash.py**. .

На рис. 3.6 показана диаграмма Фрицинга проекта, построенного на макетной плате. Pico показан на этом рисунке вместо Пико W.

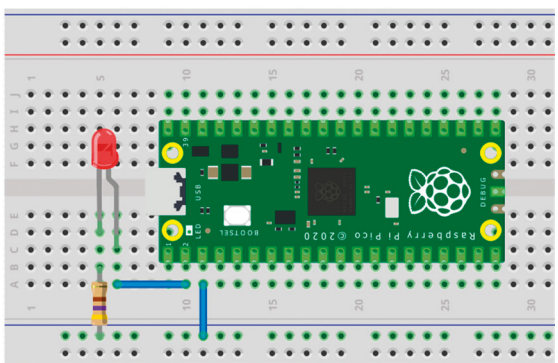


Рис.3.6: Проект, созданный на макетной плате.

Модуль **machine** поддерживает следующие функции (в последующих главах вы увидите, как использовать эти функции):

- Pin
- Timer
- ADC
- I^2C and SoftI2C
- SPI and SoftSPI
- WDT
- PWM
- UART

Общий формат функции machine.Pin (для получения дополнительной информации см. ссылку:https://docs.micropython.org/en/latest/library/machine.Pin.html):

```
machine.Pin(pin, mode, pull, value, alt)
```

Параметры вывода можно повторно инициализировать с помощью следующей функции:

Pin.init (pin, mode, value, drive, at)

Где **pin** это номер пина.

Параметр **mode** может принимать следующие значения:

Pin.IN	- пин сконфигурирован как вход
Pin.OUT	- пин сконфигурирован как выход
Pin.OPEN_DRAIN	- вывод сконфигурирован как выход с открытым стоком
Pin.ALT	- пин настраивается как альтернативная функция

Параметр **pull** может принимать следующие значения:

NONE	- нет внутренних подтягивающих и ограничивающих резисторов
Pin.PULL_UP	- включен внутренний подтягивающий резистор
Pin.PULL_DOWN	- включен внутренний ограничивающий резистор

Параметр **value** допустим только для режимов Pin.OUT и Pin.OPEN_DRAIN и указывает начальное значение выходного пина (если указано)

Параметр **alt** указывает альтернативную функцию для пина (зависит от порта). Этот параметр действителен только для режимов PIN.ALT и Pin.ALT_OPEN_DRAIN.

Пин можно установить/сбросить с помощью одной из следующих функций:

Pin.value(1)	-	установить пин (ножку) на логическую 1
Pin.value(0)	-	установить пин на логический 0

Некоторые другие полезные функции **machine**:

machine.reset()	- перезагрузить устройство (аналогично нажатию внешней кнопки RESET)
machine.reset_cause)	- вернуть причину сброса
machine.disable_irq()	- отключить запросы на прерывание
machine.enable_irq()	- включить запросы на прерывание
machine.freq()	- возвращает частоту процессора

3.3 Проект 2: Мигающий сигнал SOS

Описание: В этом проекте внешний светодиод непрерывно мигает сигналом SOS (три точки, затем три тире, затем три точки). Точка представлена светодиодом, включенным на 0,25 секунды (время точки), а тире — светодиодом, включенным на 1 секунду (время тире). Задержка между точками и тире установлена на 0,2 секунды (время GAP). Этот процесс повторяется непрерывно после 2-секундной задержки.

Блок-схема и принципиальная схема этого проекта такие же, как на рис.3.1 и рис.3.2 соответственно.

Листинг программы. На рис.3.7 показан листинг программы (Программа: **SOS**). В начале программы задаются точки, тире и промежутки времени. Затем формируется цикл с помощью оператора **while**. Внутри этого цикла формируются два цикла **for**, каждый из которых повторяется 3 раза. Первый цикл отображает три точки, а второй цикл — три тире. Этот процесс повторяется после 2-секундной задержки.

```
#-------------------------------------------------------------
#                       LED МИГАЕТ СИГНАЛОМ SOS
#                       ========================
#
# В этой программе внешний светодиод на порту GP1 мигает
# сигналом SOS (... --- ...)
#
# Автор: Доган Ибрагим
# Файл   : SOS.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
from machine import Pin
import utime

Dot = 0.25                          # Время точки
Dash = 1.0                          # Время тире
```

```
Gap = 0.2                       # Время паузы
ON = 1                          # ON
OFF = 0                         # OFF

LED = Pin(1, Pin.OUT)           # Светодиод на GP1

while True:                     # ДЕЛАТЬ ВСЕГДА
    for i in range(0, 3):
        LED.value(ON)           # LED ON
        utime.sleep(Dot)        # Время ожидания точки
        LED.value(OFF)          # LED OFF
        utime.sleep(Gap)        # Время ожидания тире

    utime.sleep(0.5)            # Задержка 0,5 секунды

    for i in range(0, 3):
        LED.value(ON)           # LED ON
        utime.sleep(Dash)       # Время ожидания тире
        LED.value(OFF)          # LED OFF
        utime.sleep(Gap)        # Время ожидания паузы

    utime.sleep(0.5)            # Задержка 0,5 секунды

    for i in range(0, 3):
        LED.value(ON)           # LED ON
        utime.sleep(Dot)        # Время ожидания точки
        LED.value(OFF)          # LED OFF
        utime.sleep(Gap)        # Время ожидания тире

    utime.sleep(2)              # Задержка 2 секунды
```

Рис.3.7: Программа: SOS.

Предложения: Вы можете легко заменить светодиод зуммером, чтобы сигнал SOS был слышен. Есть два типа зуммеров: активные и пассивные. Пассивные зуммеры требуют подачи на них звукового сигнала, а частота выходного сигнала зависит от частоты подаваемого сигнала. Активные зуммеры издают звуковой сигнал при подаче на них напряжения. В этом проекте вы можете использовать активный зуммер с транзистором (можно использовать любой транзистор типа NPN), как показано на рис.3.8.

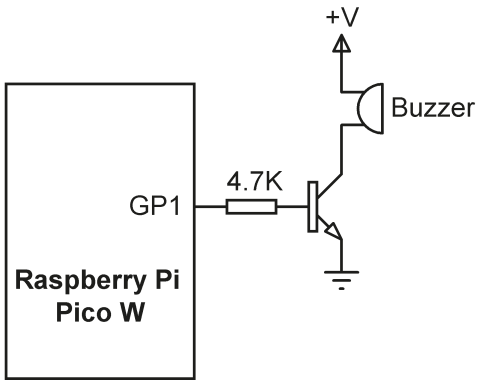


Рис.3.8: Использование активного зуммера.

3.4 Проект 3: Мигающий светодиод – с использованием таймера

Описание: этот проект очень похож на проект 1, где внешний светодиодподключен к порту GP1 порта Pico. В этом проекте используется таймер, которыймигает светодиодом каждые 500 мс.

Цель: Таймеры являются очень важными компонентами всех систем на МК. Цель этого проекта — показать, как таймер можно использовать в программе. Блок-схема и принципиальная схема этого проекта такие же, как на рис.3.1 и рис.3.2 соответственно. Листинг программы. На рис. 3.9 показан листинг программы (Программа: **LEDTimer**). Светодиод на порту GP1 сконфигурирован как выход. Затем инициализируется таймер, который периодически вызывает функцию **Flash_LED** дважды (частота = 2,0 Гц, период = 0,5 секунды) в секунду. Обратите внимание, что светодиод мигает с помощью функции **toggle**.

```
#-------------------------------------------------------
#      МИГАНИЕ СВЕТОДИОДА С ИСПОЛЬЗОВАНИЕМ ТАЙМЕРА
#      ============================================
#
# В этой программе внешний светодиод мигает два раза в секунду
# по таймеру
#
# Автор: Доган Ибрагим
# Файл  : LEDTimer.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------
from machine import Pin, Timer

LED = Pin(1, Pin.OUT)
tim = Timer()

def Flash_LED(timer):
```

```
    global LED
    LED.toggle()

tim.init(freq = 2.0, mode = Timer.PERIODIC, callback = Flash_LED)
```

Рис.3.9: Программа: **LEDTimer**.

3.5 Проект 4: Изменение частоты мигания светодиода – с помощью кнопочных прерываний

Описание: В этом проекте внешний светодиод подключен к выводу GP1 порта Pico, как и в предыдущих проектах. Кроме того, к контактам порта GP2 и GP3 подключены две кнопки. В начале программы светодиод мигает каждую секунду. Кнопка на GP2 называется **Faster** и нажатие этой кнопки приводит к более быстрому миганию светодиода. Точно так же кнопка в GP3 называется **Slower**, и при нажатии на эту кнопку светодиод мигает медленнее.

Цель: Цель этого проекта — показать, как кнопки могут быть подключены к Pico и как можно прочитать состояние кнопки.

Блок-схема: На рис.3.10 показана блок-схема проекта

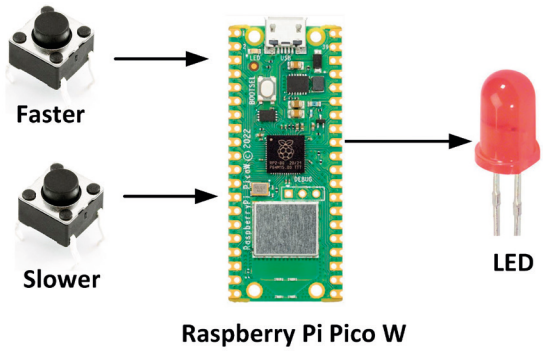


Рис.3.10: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис.3.11. Светодиод подключен через токоограничивающий резистор сопротивлением 470 Ом. Две кнопки соединены через резисторы 10K. Состоянием кнопок по умолчанию является логическая 1, которая подтягивается через резисторы. Нажатие кнопки изменяет состояние ее выхода на логический 0.

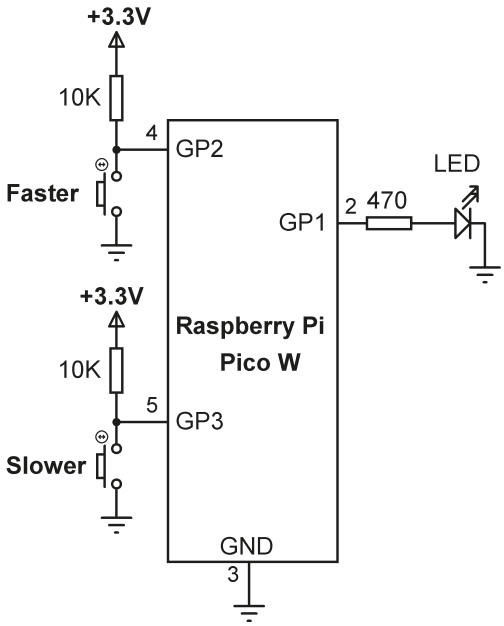


Рис.3.11: Принципиальная схема проекта.

Листинг программы: на рис.3.12 представлен листинг программы (Программа: **LEDrate**). В начале программы светодиод назначен на GP1, а кнопки **Faster** и **Slower** назначены на порты **GP2** и **GP3** соответственно. Частота мигания по умолчанию установлена равной одной секунде и хранится в переменной **dly.** Программа основана на внешнем прерывании. Нажатие любой из кнопок создает прерывание. Например, прерывание для кнопки **Faster** настраивается с помощью следующего вызова функции:

```
Faster.irq(handler=Flash_Faster, trigger=Faster.IRQ_FALLING)
```

Flash_Faster — это имя процедуры обработки прерывания, где **dly** уменьшается. Прерывание сконфигурировано так, чтобы оно происходило по заднему фронту сигнала кнопки, т. е. при нажатии кнопки (нормальное состояние кнопок — логическая 1, подтянутые резисторами). Внутри подпрограммы обслуживания прерываний **Flash_Faster** переменная **dly** объявлена **global** - глобальной, чтобы к ней можно было получить доступ. Затем её значение уменьшается на 100 мс (0,1 секунды). Процедура обслуживания прерывания для кнопки **Slower** выполняется аналогично, при этом задержка увеличивается на 100 мс при каждом нажатии кнопки.

Другие режимы внешнего прерывания:

Pin.IRQ_FALLING	- прерывание по заднему фронту (от высокого к низкому)
Pin.IRQ_RISING	- прерывание по переднему фронту (от низкого к высокому)
Pin.IRQ_LOW_LEVEL	- прерывание по низкому уровню
Pin.IRQ_HIGH_LEVEL	- прерывание на высоком уровне

Приведенные выше значения могут быть объединены по OR (ИЛИ) для запуска нескольких событий. Вы также можете указать приоритет прерывания с помощью ключевого слова **priority**, где более высокие значения представляют более высокие приоритеты.

Параметр пробуждения прерывания **wake** может быть указан со значениями **None**, **machine.IDLE**, **machine.SLEEP** или **machine.DEEPSLEEP**.

Кроме того, можно указать параметр **hard** со значениями **False** или **True**.

Если этот параметр установлен в True, то используются аппаратные прерывания, которые имеют более быструю реакцию.

```
#-----------------------------------------------------------------
#               ИЗМЕНИТЬ ЧАСТОТУ МИГАНИЯ LED
#               ==============================
#
# В этой программе используется внешний светодиод и две кнопки
# При нажатии Faster светодиод мигает быстрее, а при нажатии Slower
# светодиод мигает медленнее
#
# Автор: Доган Ибрагим
# Файл  : LEDrate.py
# Дата: октябрь 2022 г.
#-----------------------------------------------------------------
from machine import Pin
import utime

LED = Pin(1, Pin.OUT)                   # Светодиод на пине GP1
Faster = Pin(2, Pin.IN)                 # Быстрее на пине GP2
Slower = Pin(3, Pin.IN)                 # Медленнее на пине GP3
dly = 1.0                               # Задержка по умолчанию

#
# Это процедура обслуживания прерывания. Всякий раз, когда нажимается кнопка
# Faster, программа переходит сюда и уменьшает задержку,
# чтобы ускорить мигание
#
def Flash_Faster(Faster):
   global dly
   dly = dly - 0.1

#
# Это процедура обслуживания прерывания. Всякий раз, когда нажимается кнопка
# Медленнее, программа переходит сюда и увеличивает задержку,
# чтобы сделать мигание медленнее
#
def Flash_Slower(Slower):
```

```
    global dly
    dly = dly + 0.1

#
# Настроить внешние прерывания
#
Faster.irq(handler=Flash_Faster,trigger=Faster.IRQ_FALLING)
Slower.irq(handler=Flash_Slower,trigger=Slower.IRQ_FALLING)

#
# Основной цикл программы
#
while True:
    LED.value(0)                # LED горит
    utime.sleep(dly)            # Задержка dly
    LED.value(1)                # LED не горит
    utime.sleep(dly)            # Задержка dly
```

Рис.*3.12:* Программа: **LEDrate**.

На рис.3.13 показана диаграмма Фритцинга проекта.

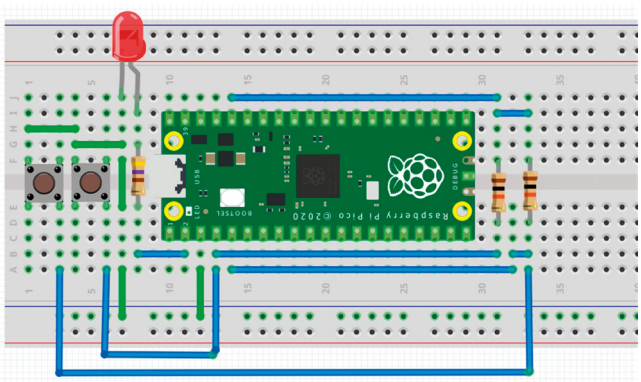


Рис.*3.13:* Проект, созданный на макетной плате.

Использование внутренних подтягивающих резисторов

Вы можете упростить схему на рис.3.11, удалив внешние резисторы и используя внутренние подтягивающие резисторы Pico. Упрощенная принципиальная схема показана на рис.3.14. Листинг модифицированной программы (Program: **LEDrate2**) показан на рис.3.15, где к операторам входной конфигурации добавлена опция **pull = Pin.PULL__UP**.

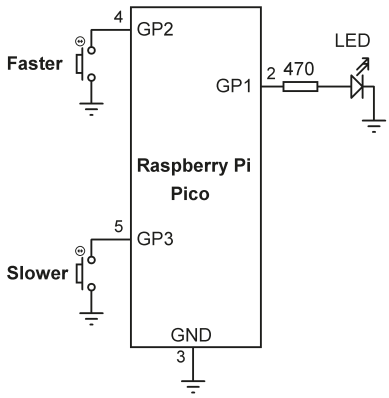


Рис.*3.14:* Модифицированная принципиальная схема.

```
#-----------------------------------------------------------------
#              ИЗМЕНИТЬ ЧАСТОТУ МИГАНИЯ LED
#              ==============================
#
# В этой программе используется внешний светодиод и две кнопки.
# При нажатии Faster светодиод мигает быстрее, а при нажатии Slower
# светодиод мигает медленнее. В этой программе используются внутренние подтягивания.
#
# Автор: Доган Ибрагим
# Файл   : LEDrate2.py
# Дата: октябрь 2022 г.
#-----------------------------------------------------------------
from machine import Pin
import utime

LED = Pin(1, Pin.OUT)                                    # LED на GP1
Faster = Pin(2, Pin.IN, pull=Pin.PULL_UP)                # Faster на GP2
Slower = Pin(3, Pin.IN, pull=Pin.PULL_UP)                # Slower на GP3
dly = 1.0                                                # Задержка по умолчанию

#
# Это процедура обслуживания прерывания. Всякий раз, когда нажимается кнопка
# Faster, программа переходит сюда и
# уменьшает задержку, чтобы ускорить мигание
#
def Flash_Faster(Faster):
   global dly
   dly = dly - 0.1

#
# то процедура обслуживания прерывания. Всякий раз, когда нажимается кнопка
```

```
# Slower,программа переходит сюда и увеличивает
# чтобы сделать мигание медленнее
#
def Flash_Slower(Slower):
    global dly
    dly = dly + 0.1

#
# Настроить внешние прерывания
#
Faster.irq(handler=Flash_Faster,trigger=Faster.IRQ_FALLING)
Slower.irq(handler=Flash_Slower,trigger=Slower.IRQ_FALLING)

#
# Основной цикл программы
#
while True:
    LED.value(0)                # LED горит
    utime.sleep(dly)            # Задержка dly
    LED.value(1)                # LED не горит
    utime.sleep(dly)            # Задержка dly
```

Рис.*3.15:* Программа: **LEDrate2**.

3.6 Проект 5: Случайное мигание красных, зеленых и синих светодиодов — RGB

Описание:В этом проекте к Pico подключен RGB-светодиод. RGB мигает случайным образом, когда красный, зеленый или синий светодиоды случайным образом включаются или выключаются.

Исходная информация: Как показано на рис.3.16, светодиод RGB представляет собой 4-выводное устройство, включающее красный, зеленый и синий светодиоды. Каждому цветному светодиоду назначается вывод, где четвертый вывод является землей. Активируя разные светодиоды с разной яркостью, вы можете генерировать много разных цветов. В этом проекте используется RGB-светодиод с общим катодом. Обратите внимание, что катодный вывод RGB-светодиода является более длинным.

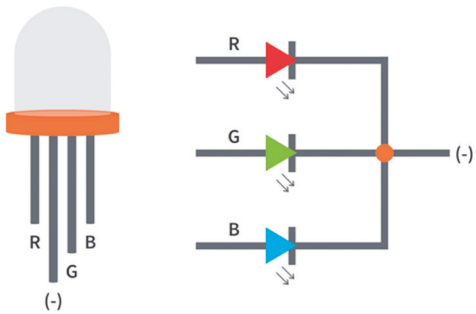


Рис.3.16: RGB-светодиод.

Block Diagram: Figure 3.17 shows the block diagram of the project.

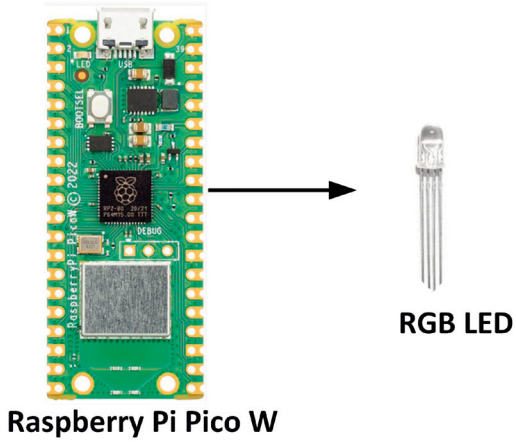


Рис.3.17: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис.3.18. Выводы красного,зеленого и синего LED подключены к пинам порта GP1, GP2 и GP3 соответственно через токоограничивающие резисторы на 470 Ом.

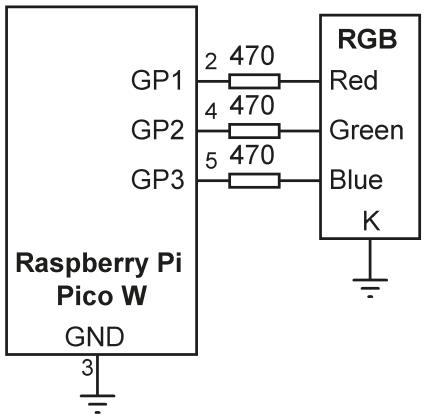


Рис.3.18: Принципиальная схема проекта.

Листинг программы: На рис. 3.19 показан листинг программы (Программа:**RGB**). Случайные числа генерируются как 0 или 1 для каждого цвета, и эти числа используются для включения или выключения цветного светодиода.

```
#------------------------------------------------------------
#                 СЛУЧАЙНО МИГАЮЩИЙ LED RGB
#                 ============================
#
# В этой программе светодиод RGB подключен к Pico. Три
# цветных LED случайным образом мигают каждые 500 мс
#
# Автор: Доган Ибрагим
# Файл   : RGB.py
# Дата: октябрь 2022 г.
#------------------------------------------------------------
from machine import Pin
import utime
import random

Red = Pin(1, machine.Pin.OUT)
Green = Pin(2, Pin.OUT)
Blue = Pin(3, Pin.OUT)

while True:
    r = random.randint(0, 1)
    g = random.randint(0, 1)
    b = random.randint(0, 1)
    Red.value(r)
    utime.sleep(0.2)
    Green.value(g)
    utime.sleep(0.2)
    Blue.value(b)
    utime.sleep(0.2)
```

Рис.*3.19:* Программа:**RGB**.

3.7 Проект 6: LED для двоичного счета

Описание: В этом проекте к Pico подключено 8 LED. Светодиоды ведут счет в двоичном формате от 0 до 255, как показано на рис. 3.20, с задержкой в одну секунду между каждым счетом.

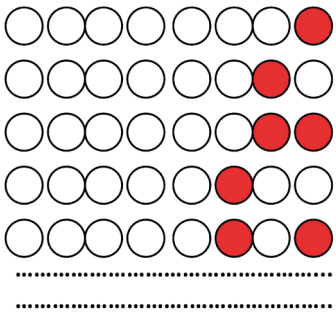


Рис.3.20: Светодиоды двоичного счета.

Цель: Цель этого проекта — показать, как можно комбинировать группу пинов порта и доступ как к параллельному порту.

Блок-схема: На рис. 3.21 показана блок-схема проекта.

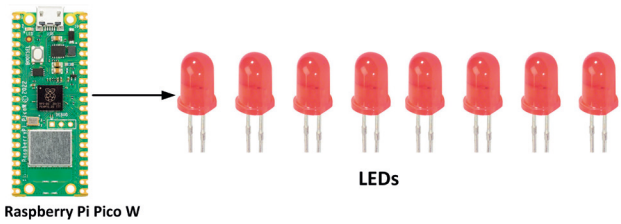


Рис.3.21: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис.3.22. Светодиоды подключены к Pico через токоограничивающие резисторы сопротивлением 470 Ом.

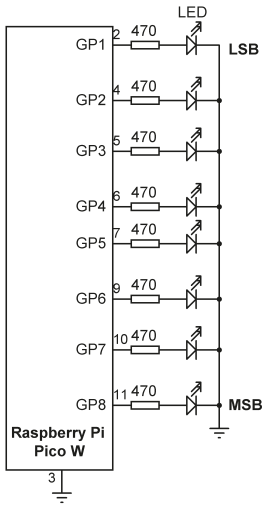


Рис.3.22: Принципиальная схема проекта.

Листинг программы. На рис. 3.23 показан листинг программы (Программа: **LEDCount**). Все 8 портов **GPIO**, используемых в проекте, настроены как выходы с помощью функции **Configure_Port**. Обратите внимание, что функция **Configure_Port** является общей, а list **DIR** устанавливает направления пинов GPIO. «**O**» устанавливается как выход, а «**I**» — как вход. Затем формируется цикл для бесконечного выполнения, и внутри этого цикла светодиоды подсчитываются на единицу в двоичном формате. В качестве счетчика используется переменная **cnt**. Функция **Port_Output** используется для управления светодиодами. Эта функция может принимать целые числа от 0 до 255 и преобразовывать входное число (x) в двоичное с помощью встроенной функции **bin**. Затем из выходной строки **b** удаляются начальные символы «0b» (функция bin вставляет символы «0b» вначало преобразованной строки). Затем преобразованная строка b состоит из 8 символов путем вставки начальных нулей. Затем строка передается в порт побитно, начиная с самой старшей битовой позиции.

```
#------------------------------------------------------------
#              ДВОИЧНЫЙ СЧЕТ 8 LED
#              ======================
#
# В этой программе к Pico подключено 8 светодиодов.
# Светодиоды считают в двоичном формате каждую секунду
#
# Автор: Доган Ибрагим
# Файл   : LEDCount.py
# Дата: октябрь 2022 г.
#------------------------------------------------------------
from machine import Pin
import utime

PORT = [8, 7, 6, 5, 4,  3,  2,  1]             # соединения порта
DIR = ["0","0","0","0","0","0","0","0"]        # направления порта
L = [0]*8

#
# Эта функция настраивает пины порта как выходы ("0") или
# в качестве входных данных ("I")
#
def Configure_Port():
   for i in range(0, 8):
      if DIR[i] == "0":
         L[i] = Pin(PORT[i], Pin.OUT)
      else:
         L[i] = Pin(PORT[i], Pin.IN)
   return

#
#  Эта функция отправляет 8-битные данные (от 0 до 255) в PORT
#
def Port_Output(x):
```

```
    b = bin(x)                          # преобразовать в двоичный
    b = b.replace("0b", "")             # удалить начальный "0b"
    diff = 8 - len(b)                   # найти length
    for i in range (0, diff):
       b = "0" + b                      # вставить начальный os

    for i in range (0, 8):
       if b[i] == "1":
          L[i].value(1)
       else:
          L[i].value(0)
    return

#
#  Настроить PORT на все выходы
#
Configure_Port()

#
# Основной цикл программы. Подсчитать в двоичном формате каждую секунду
#
cnt = 0
while True:
  Port_Output(cnt)                              # отправить cnt на порт
  utime.sleep(1)                                # ждите 1 секунду
  cnt = cnt + 1                                 # приращение cnt
  if cnt > 255:
     cnt = 0
```

Рис.*3.23:* Программа:**LEDCount**.

3.8 Проект 7: Счастливый день недели

Описание: В этом проекте 7 светодиодов расположены в форме круга и подключены к Raspberry Pi Pico. Предполагается, что каждый светодиод представляет день недели. При нажатии кнопки генерируется случайное число от 1 до 7 и загорается только один из светодиодов. Название дня, соответствующее этому индикатору, считается вашим счастливым днем недели.

Блок-схема: На рис. 3.24 показана блок-схема проекта.

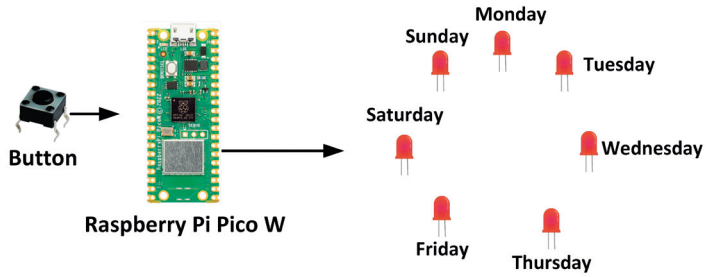


Рис.3.24: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис.3.25, где 7 светодиодов подключены к Pico через токоограничивающие резисторы. Кнопка подключена к GP15. Обычно выход кнопки имеет логическую 1 и переходит в логический 0, когда кнопка нажата.

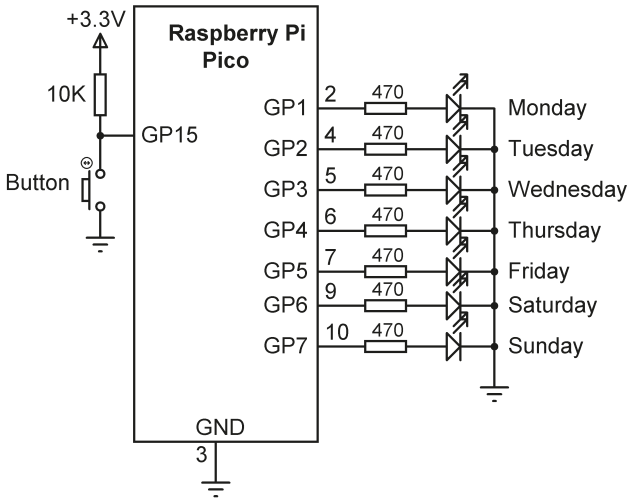


Рис.3.25: Принципиальная схема проекта.

Листинг программы: Рис.3.26 показывает листинг программы (Программа: **LuckyDay**). В начале программы все 8 выводов LED GPIO объединены в один порт и адресуются как один 8-битный порт с помощью функции **PORT_Output**. **utime.ticks_ms()** используется в качестве начального значения для генератора случайных чисел, поэтому при каждом запуске программы будут генерироваться разные последовательности чисел. Эта функция возвращает увеличивающийся счетчик миллисекунд с произвольной точкой отсчета, который возвращается после некоторого значения. Генерируется целое случайное число от 1 до 7, и это число используется для включения одного из светодиодов, соответствующего дню недели.

```
#-------------------------------------------------------------
#                        СЧАСТЛИВЫЙ ДЕНЬ НЕДЕЛИ
#                        ========================
#
#  В этой программе к Pico подключено 7 светодиодов, где
# каждый светодиод представляет день недели. Нажатие кнопки
# случайным образом включает один из светодиодов, что соответствует
# вашему счастливому дню недели.
#
#  Автор: Доган Ибрагим
# Файл   : LuckyDay.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
from machine import Pin
import utime
import random

PORT = [7, 6, 5, 4, 3,  2,  1,  0]      соединения порта
L = [0]*8
Button = Pin(15, Pin.IN)

#
# Эта функция настраивает  пины порта как выходы
#
def Configure_Port():
   for i in range(0, 8):
         L[i] = Pin(PORT[i], Pin.OUT)

#
# Эта функция отправляет 8-битные данные (от 0 до 255) в PORT
#
def Port_Output(x):
   b = bin(x)                           # преобразовать в двоичный
   b = b.replace("0b", "")              # удалить начальный "0b"
   diff = 8 - len(b)                    # найти длину
   for i in range (0, diff):
      b = "0" + b                       # вставить начальный os

   for i in range (0, 8):
      if b[i] == "1":
         L[i].value(1)
      else:
         L[i].value(0)
   return
#
# Настроить PORT на все выходы
```

```
#
Configure_Port()

#
# Основной цикл программы, проверяем, нажата ли кнопка
#
print("Press the Button to display your lucky number...")

random.seed(utime.ticks_ms())

while Button.value() == 1:                # Если кнопка не нажата
   pass
r = random.randint(1, 7)                  # Создать случайное число
r = pow(2, r-1)                           # LED, который нужно включить
Port_Output(r)                            # Отправить на LED-ы
```

Рис.*3.26:* Программа:**LuckyDay**.

3.9 Проект 8: Электронные игральные кости

Описание: В этом проекте 7 светодиодов расположены в виде граней игральной кости и используется кнопка. Когда кнопка нажата, светодиоды включаются, отображая числа от 1 до 6, как на настоящих игральных костях. Дисплей выключается через 3 секунды, готовый к следующей игре.

Цель: Цель этого проекта — показать, как можно построить игральную кость с 7 светодиодами.

Блок-схема: Блок-схема проекта показана на рис.3.27.

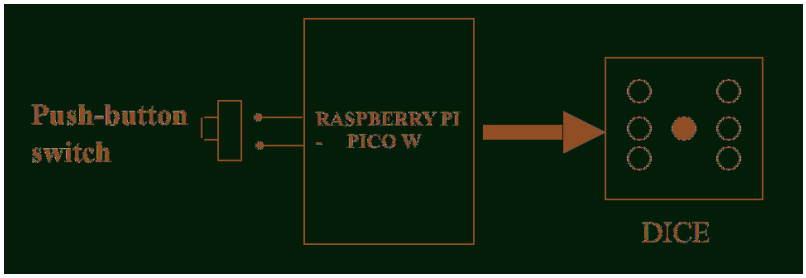


Рис.3.27: Блок-схема проекта.

На рис.3.28 показаны светодиоды, которые должны быть включены, чтобы отобразить 6 номеров игральных костей.

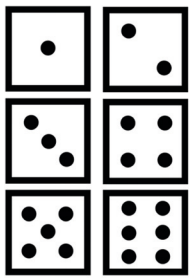


Рис.3.28: Числа светодиодных кубиков.

Принципиальная схема: Принципиальная схема проекта показана на рис.3.29. Здесь 8 пинов GPIO собраны в PORT. Есть 7 светодиодов, но 8 выводов порта используются в виде байта, где позиция старшего бита не используется. Светодиоды установлены таким образом, чтобы можно было отобразить все номера игральных костей, включив соответствующие.

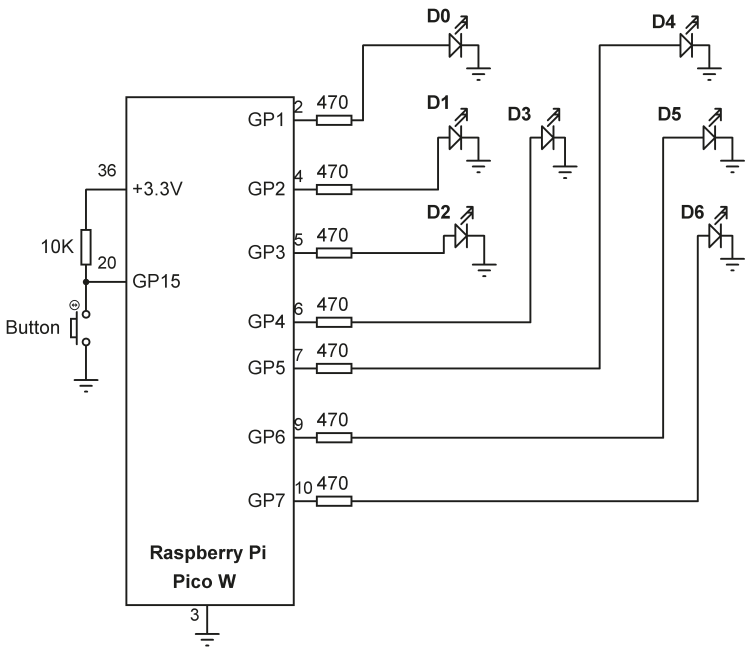


Рис.3.29: Принципиальная схема проекта.

Кнопка подключена к порту GP15.

В таблице 3.1 показано соотношение между номером игральной кости и соответствующими светодиодами, которые должны быть включены, чтобы имитировать лицевые стороны реальных игральных кубиков. Например, чтобы отобразить номер 1 (т. е. горит только средний светодиод), вы должны включить светодиод D3. Точно так же, чтобы отобразить число 4, вы должны включить D0, D2, D4 и D6.

Требуемый номер	Включенные LED
1	D3
2	D0, D6
3	D0, D3, D6
4	D0, D2, D4, D6
5	D0, D2, D3, D4, D6
6	D0, D1, D2, D4, D5, D6

Таблица 3.1: Количество кубиков и светодиоды, которые должны быть включены.

Соотношение между требуемым числом и данными, которые должны быть отправлены в PORT для включения соответствующих светодиодов, приведено в таблице 3.2. Например, чтобы отобразить номер кости 2, вы должны отправить шестнадцатеричное число 0x41 в PORT. Точно так же, чтобы отобразить число 5, вы должны отправить шестнадцатеричный код 0x5D в PORT и так далее.

Требуемый номер	Данные PORT (Hex)
1	0x08
2	0x41
3	0x51
4	0x55
5	0x5D
6	0x77

Таблица 3.2: Требуемый номер и данные PORT.

Листинг программы: Программа называется **DICE**, листинг показан на рис. 3.30. Выводы порта светодиода объявляются в виде списка в переменной **PORT** инастраиваются как выходы с помощью функции **Configure_Port**. Битовая комбинация, которая должна быть отправлена на светодиоды, соответствующие каждому номеру игрального кубика, хранится в шестнадцатеричном формате в списке под названием **DICE_NO** (см. Таблицу 3.2).

GP15 сконфигурирован как вход, и к нему подключена кнопка для имитации «бросания» игрального кубика. Состояние кнопки проверяется в основной программе, и когда кнопка нажата, вызывается функция **DICE** для отображения числа от 1 до 6 в течение 3 секунд. По истечении этого времени все светодиоды гаснут, указывая на то, что программа готова генерировать новый номер кости. Список **DICE_NO** индексируется, чтобы найти светодиоды, которые должны быть включены, и требуемый битовый шаблон отправляется в PORT для отображения количества игральных костей.

```
#------------------------------------------------------------
#                              DICE PROGRAM
#                              ============
#
# В этой программе 7 светодиодов подключены к Pico, чтобы имитировать
# игру в кости. Когда кнопка нажата, светодиоды
# отображают число от 1 до 6
#
# Автор: Доган Ибрагим
# Файл  : DICE.py
# Дата: октябрь 2022 г.
#------------------------------------------------------------
from machine import Pin
import utime
import random

PORT = [8, 7, 6, 5, 4,  3,  2,  1]     # соединения port
DICE_NO = [0, 0x08, 0x41, 0x51, 0x55, 0x5D, 0x77]
L = [0]*8
Button = Pin(15, Pin.IN)

#
# Эта функция настраивает светодиодный порт как выходы
#
def Configure_Port():
   for i in range(0, 8):
         L[i] = Pin(PORT[i], Pin.OUT)

#
# Эта функция отправляет 8-битные данные (от 0 до 255) в PORT
#
def Port_Output(x):
   b = bin(x)                           # преобразовать в двоичный
   b = b.replace("0b", "")              # удалить начальный "0b"
   diff = 8 - len(b)                    # найти длину
   for i in range (0, diff):
      b = "0" + b                       # вставить начальный os

   for i in range (0, 8):
      if b[i] == "1":
         L[i].value(1)
      else:
         L[i].value(0)
   return

#
```

```
# Программа переходит сюда после нажатия кнопки
#
def DICE():
   n = random.randint(1, 6)            # генерирует случайное число
   pattern = DICE_NO[n]                # найти шаблон
   Port_Output(pattern)                # включить необходимые LED
   utime.sleep(3)                      # ждем 3 секунды
   Port_Output(0)                      # выключаем все LED
   return
#
# Настроить PORT как выход
#
Configure_Port()

#
# Основной цикл программы, проверяем, нажата ли кнопка
#
while True:
  if Button.value() == 0:              # Кнопка нажата?
     DICE()                            # Вызов DICE
  pass                                 # ничего не делать
```

Рис.*3.30:* Программа:**DICE**.

3.10 Проект 9: Двоичный счетчик — Использование сдвигового регистра 74HC595

Описание:В этом проекте 8 светодиодов подключены к Raspberry Pi Pico W через сдвиговый регистр и токоограничивающие резисторы. Программа ведет двоичный счет через светодиоды, как показано на рис. 3.20.

Background

Количество выводов GPIO МК обычно ограничено, особенно в сложных проектах управления и мониторинга. Сдвиговые регистры используются для увеличения количества выводов МК. Микросхема сдвигового регистра управляется всего несколькими пинами МК, и микросхема может управлять до 8 выходов. Также можно каскадировать несколько сдвиговых регистров и управлять гораздо большим количеством выходов cМК. Сдвиговые регистры получают последовательные данные, а затем представляют эти данные на своих выходах.

В этом проекте вы будете использовать популярную микросхему регистра сдвига 74HC595. На рис. 3.31 показана микросхема регистра сдвига 74HC595. Пины этой микросхемы:

Pin	Функция
1-7, 15	Q0-Q7 Параллельные выходы
8	GND
9	Последовательный выход Q7 (для каскадирования)
10	MR активен при LOW основной сброс
11	SH_CP тактовый вывод сдвигового регистра

12	ST_CP (или защелка) тактовый вывод регистра хранения
13	Включение активного выхода LOW
14	DS последовательный ввод данных
15	Vcc + источника питания

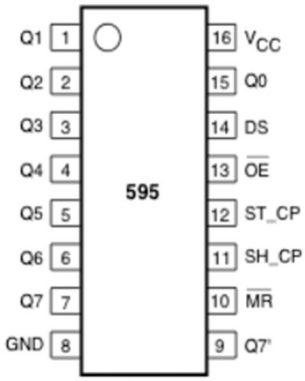


Рис*.3.31:* Микросхема регистра сдвига *74HC595.*

На рис.3.32 показаны основные операции сдвигового регистра 74HC595. Последовательные данные синхронизируются через пин 14 с помощью тактовых импульсов на пине 11 по переднему фронту. Параллельные выходные данные доступны на выходах 1-7 и 15 (пин 15 является младшим битом), когда защелка на контакте 12 установлена в логическую 1.

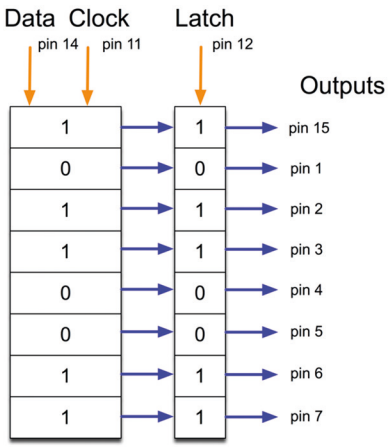


Рис.3.32: Работа сдвигового регистра 74HC595.

Работа сдвигового регистра 74HC595 показана с помощью PDL на рис. 3.33.

```
BEGIN
      Configure DataPin (pin 14) as output
      Configure ClockPin (pin 11) as output
      Configure LatchPin (pin 12) as output
      Set clock = 0, latch = 0
      DO 8 times
             Set clock = 0
Send serial data bit in reverse order
Set clock = 1
      ENDDO
      Set latch = 1
END
```

Рис.3.33: PDL сдвигового регистра 74HC595.

Цель: Цель этого проекта — показать, как сдвиговый регистр можно использовать с Raspberry Pi Pico W.

Блок-схема: На рис. 3.34 показана блок-схема проекта.

Рис.3.34: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис.3.35. Светодиоды подключены к параллельным выходам сдвигового регистра через токоограничивающие резисторы сопротивлением 470 Ом. Контакты тактов, защелки и последовательных данных подключены к контактам Raspberry Pi Pico W. Выводы MR и OE сдвигового регистра в данном проекте не используются.

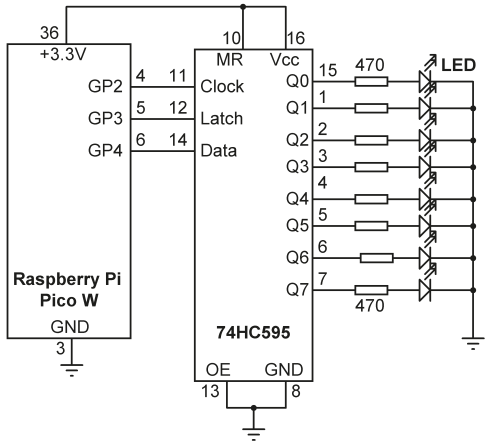


Рис.3.35: Принципиальная схема проекта.

Листинг программы: Рисунок 3.36 показывает листинг программы (Программа: **ShiftUpCnt**). В начале программы пины **Data**, **Clock** и **Latch** настроены как выходы. Функция **ShiftOut** имеет 4 аргумента. Эта функция выводит данные в последовательной форме в сдвиговый регистр. Параллельные данные выводятся на контакты Q0-Q7, когда на защелке регистра сдвига находится 1. Внутри основного цикла программы переменная **Count** увеличивается каждую секунду, а светодиоды отображают двоичный счет от 0 до 255. Обратите внимание, что переменная **Count** преобразуется в 8-битную двоичную строку внутри функции **ShiftOut**.

```
------------------------------------------------------------
#                    LED счетчик на сдвиговом регистре
#                    ================================
#
# В этой программе 8 светодиодов подключены к сдвиговому регистру типа 74HC595
# Программа управляет сдвиговым регистром,
# чтобы  считать  в  двоичном  виде  каждую секунду с помощью LED
#
# Автор: Доган Ибрагим
# Файл   : ShiftUpCnt.py
# Дата: октябрь 2022 г.
#------------------------------------------------------------
from machine import Pin
import utime

DataPin = Pin(4, Pin.OUT)
ClockPin = Pin(2, Pin.OUT)
LatchPin = Pin(3, Pin.OUT)

def ShiftOut(input, DS, CLK, Latch):
    CLK.value(0)
    Latch.value(0)
```

```
    for k in range(8):
        CLK.value(0)
        val = "{0:08b}".format(input)
        DS.value(int(val[k]))
        CLK.value(1)

    Latch.value(1)

Count = 0
while True:
    ShiftOut(Count, DataPin, ClockPin, LatchPin)
    Count = Count + 1
    if Count > 255:
        Count = 0
    utime.sleep(1)
```

Рис.*3.36:* Программа:**ShiftUpCnt**.

Делаем функцию ShiftOut модулем

Поскольку сдвиговый регистр, скорее всего, будет использоваться во многих проектах, имеет смысл сделать функцию **Shift** неимпортируемым модулем. Шаги:

- Создайте новый файл и скопируйте код **ShiftOut** в этот файл.
- Сохраните файл как **ShiftOut.py** на Raspberry Pi Pico W (убедитесь, что расширение .py включено).
- Импортируйте модуль в начало вашей программы
- Настройте пины данных, тактов и защелки в качестве выходов.

На рис.3.37 показан пример программы (Программа: **ShiftUpCnt2**), в которой модуль **ShiftOut** импортируется в программу.

```
#-------------------------------------------------------------
#          LED СЧЕТЧИК НА СДВИГОВОМ РЕГИСТРЕ
#          ===================================
#
# В этой программе 8 светодиодов подключены к сдвиговому регистру типа 74HC595.
# Программа управляет регистром сдвига, чтобы
#  считать в двоичном виде каждую секунду с помощью светодиодов
#  Эта программа использует модуль ShiftOut
#
# Автор: Доган Ибрагим
# Файл   : ShiftUpCnt2.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
```

```
from machine import Pin
import utime
from ShiftOut import ShiftOut

DataPin = Pin(4, Pin.OUT)
ClockPin = Pin(2, Pin.OUT)
LatchPin = Pin(3, Pin.OUT)

Count = 0
while True:
    ShiftOut(Count, DataPin, ClockPin, LatchPin)
    Count = Count + 1
    if Count > 255:
        Count = 0
    utime.sleep(1)
```

Рис.*3.37:* Импорт модуля *ShiftOut.*

3.11 Проект 10: В погоне за светодиодами — Использование сдвигового регистра 74HC595

Описание: В этом проекте 8 светодиодов подключены к Raspberry Pi Pico W через сдвиговый регистр и токоограничивающие резисторы, как и в предыдущем проекте. Программа включает/выключает светодиоды в последовательности каждую секунду, как показано на рис. 3.38.

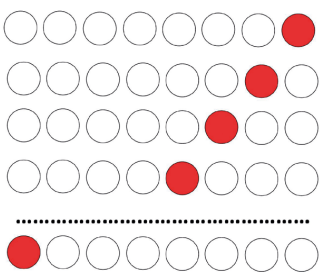


Рис.3.38: Индикаторы погони.

Блок-схема и принципиальная схема проекта представлены на рис. 3.34 и рис. 3.35. соответственно.

Листинг программы: Теперь, когда у вас есть модуль **ShiftOut,** программа очень проста и состоит из нескольких строк, как показано на рис. 3.39 (Программа: **rotate**).

```
#-------------------------------------------------------------
#            ВРАЩЕНИЕ LED СДВИГОВЫМ РЕГИСТРОМ
#            ==================================
#
# В этой программе 8 LED подключены к сдвиговому регистру
# типа 74HC595. Программа вращает LED
# как описано в тексте.
#
# Автор: Доган Ибрагим
# Файл  : rotate.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
from machine import Pin
import utime
from ShiftOut import ShiftOut

DataPin = Pin(4, Pin.OUT)
ClockPin = Pin(2, Pin.OUT)
LatchPin = Pin(3, Pin.OUT)

Count = 1
while True:
    ShiftOut(Count, DataPin, ClockPin, LatchPin)
    Count = Count << 1
    if Count > 128:
        Count = 1
    utime.sleep(1)
```

Рис. *3.39:* Программа: **rotate**.

3.12 Проект 11: Включение выбранного светодиода — использование регистра сдвига 74HC595

Описание: В этом проекте 8 светодиодов подключены к Raspberry Pi Pico W через сдвиговый регистр и токоограничивающие резисторы, как и в предыдущем проекте. Программа включает выбранный светодиод. Светодиоды пронумерованы от 1 до 8, где 1 означает светодиод в позиции **LSB**. Например, при выборе 2 включается второй светодиод с правой стороны. Блок-схема и принципиальная схема проекта представлены на рис. 3.34 и рис. 3.35 соответственно.

Листинг программы: На рис. 3.40 показан листинг программы (Программа: **LEDselect**).

```
#-----------------------------------------------------------
#               ВЫБОРКА СВЕТОДИОДА РЕГИСТРОМ СДВИГА
#               ====================================
#
# В этой программе 8 светодиодов подключены к сдвиговому регистру типа 74HC595
# Программа включает выбранный светодиод
#
# Автор: Доган Ибрагим
# Файл   : LEDselect.py
# Дата: октябрь 2022 г.
#-----------------------------------------------------------
from machine import Pin
import utime
from ShiftOut import ShiftOut

DataPin = Pin(4, Pin.OUT)
ClockPin = Pin(2, Pin.OUT)
LatchPin = Pin(3, Pin.OUT)

while True:
    sel = int(input("Selected LED (1-8): "))
    sel = sel - 1
    r = 2**sel
    ShiftOut(r, DataPin, ClockPin, LatchPin)
    utime.sleep(3)
```

Рис.*3.40:* Программа: **LEDselect**.

Пример запуска программы показан на рис. 3.41.

```
Selected LED (1-8): 2
Selected LED (1-8): 4
Selected LED (1-8): 1
Selected LED (1-8): 7
Selected LED (1-8): 8
Selected LED (1-8): 7
Selected LED (1-8):
```

Рис.3.41: Пример запуска.

3.13 Проект 12: Случайное мигание светодиодов — использование сдвигового регистра 74HC595

Описание: В этом проекте 8 светодиодов подключены к Raspberry Pi Pico W через сдвиговый регистр и токоограничивающие резисторы, как и в предыдущем проекте. Программа случайным образом мигает светодиодами, как в елочных огнях.

Блок-схема и принципиальная схема проекта представлены на рис.3.34 и рис.3.35 соответственно.

Листинг программы: на рис.3.42 представлен листинг программы (Программа: XMAS).

```
#-------------------------------------------------------
#   СЛУЧАЙНО МИГАЮЩИЕ СВЕТОДИОДЫ РЕГИСТРОМ  СДВИГА
#   ======================================================
#
# В этой программе 8 светодиодов подключены к сдвиговому регистру типа 74HC595
# Программа случайным образом мигает светодиодами
#
# Автор: Доган Ибрагим
# Файл  : XMAS.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------
from machine import Pin
import utime
import random
from ShiftOut import ShiftOut

DataPin = Pin(4, Pin.OUT)
ClockPin = Pin(2, Pin.OUT)
LatchPin = Pin(3, Pin.OUT)

while True:
    n = random.randint(1, 255)
    ShiftOut(n, DataPin, ClockPin, LatchPin)
    utime.sleep(0.2)
```

Рис.*3.42:* Программа: ***XMAS.***

3.14 Проект 13: Светофоры

Описание: В этом проекте написана программа для управления светофором на простом перекрестке. Для имитации светофора используются красные, желтые и зеленые светодиоды.

Предполагается, что транспорт движется с левой стороны дороги (т. е. руль автомобилей находится с правой стороны). В этом проекте развязка состоит из двух дорог: NORTH ROAD - СЕВЕРНАЯ ДОРОГА и EAST ROAD - ВОСТОЧНАЯ ДОРОГА. Светофоры расположены на перекрестке, как показано на рис.3.43.

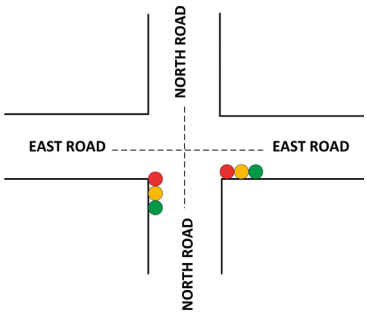


Рис.3.43: Пример расстановки.

Последовательность светофора в Великобритании выглядит следующим образом (рис.3.44). Примечание: желтый цвет - янтарный цвет:

ROAD 1	**ROAD 2**
RED	GREEN
RED+YELOW	YELLOW
GREEN	RED
YELLOW	RED+YELLOW
RED	GREEN

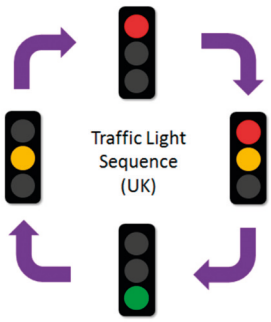


Рис.3.44: Последовательность движения в Великобритании.

Время, используемое в этом проекте, показано в Таблице 3.3, где зеленый цвет на EAST ROAD соответствует 10 секундам, а зеленый цвет на NORTH – 4 секундам. Время цикла освещения составляет 27 секунд и имеет следующие значения:

EAST ROAD	**NORTH ROAD**
10 с зелен	18 с крас.
12 с крас.	2 с крас. + желт.
2 с крас. + желт.	4 с зелен
3 с желт.	3 с желт.

Время (секунды)	ВОСТОЧНАЯ ДОРОГА	СЕВЕРНАЯ ДОРОГА
0	GREEN	RED
1	GREEN	RED
2	GREEN	RED
3	GREEN	RED
4	GREEN	RED
5	GREEN	RED
6	GREEN	RED
7	GREEN	RED
8	GREEN	RED
9	GREEN	RED
10	YELLOW	RED
11	YELLOW	RED
12	YELLOW	RED
13	RED	RED
14	RED	RED + AMBER
15	RED	RED + AMBER
16	RED	GREEN
17	RED	GREEN
18	RED	GREEN
19	RED	GREEN
20	RED	YELLOW
21	RED	YELLOW
22	RED	YELLOW
23	RED	RED
24	RED + AMBER	RED
25	RED + AMBER	RED
26	GREEN	RED
27	GREEN	RED

Таблица 3.3: Время, использованное в проекте.

Блок-схема: Блок-схема проекта показана на рис.3.45. Используется 6 светодиодов, по 3 на каждую дорогу.

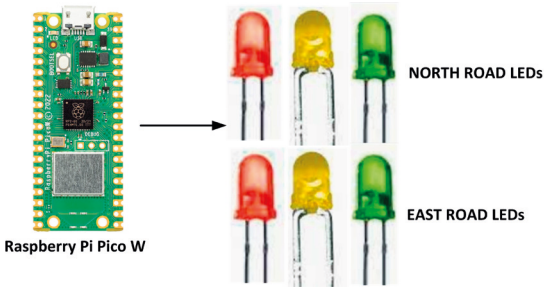


Рис.3.45: Блок-схема проекта.

Принципиальная схема: На рис. 3.46 показано, как светодиоды подключены к Raspberry Pi Pico W. Красный, желтый и зеленый светодиоды СЕВЕРНОЙ ДОРОГИ подключены к GP10, GP11 и GP12 соответственно. Красный, желтый и зеленый светодиоды ВОСТОЧНОЙ ДОРОГИ подключены к GP13, GP14 и GP15 соответственно.

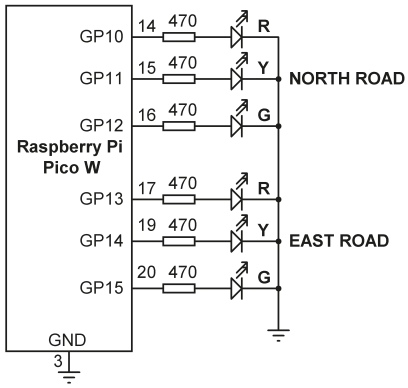


Рис.3.46: Принципиальная схема проекта.

Конструкция: проект был построен на макетной плате.

Листинг программы: На рис.3.47 показан листинг программы (Программа: **traffic**). В начале программы светодиоды назначаются выходным портам, и все светодиоды выключаются. Основная программа работает вечно в цикле **while**. Внутри этого цикла светодиоды на обеих дорогах включаются и выключаются в моменты времени, указанные в Таблице 3.3.

```
#-------------------------------------------------------------
#                          СВЕТОФОР
#                          =========
#
# Это проект светофора. Перекресток моделируется
# с подробностями, указанными в тексте.
#
# Автор: Доган Ибрагим
```

```
# файл  : traffic.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
from machine import Pin
import utime

#
# Конфигурация LED
#
NorthR = Pin(10, Pin.OUT)                   # NORTH ROAD red
NorthY = Pin(11, Pin.OUT)                   # NORTH ROAD yellow
NorthG = Pin(12, Pin.OUT)                   # NORTH ROAD green
EastR  = Pin(13, Pin.OUT)                   # EAST ROAD red
EastY  = Pin(14, Pin.OUT)                   # EAST ROAD yellow
EastG  = Pin(15, Pin.OUT)                   # EAST ROAD green

#
# Все светодиоды выключены
#
NorthR.value(0)
NorthY.value(0)
NorthG.value(0)
EastR.value(0)
EastY.value(0)
EastG.value(0)

#
# Начало последовательности
#
while True:
    NorthG.value(1)
    EastR.value(1)
    utime.sleep(10)

    NorthG.value(0)
    NorthY.value(1)
    utime.sleep(3)

    NorthY.value(0)
    NorthR.value(1)
    utime.sleep(1)

    EastY.value(1)
    utime.sleep(2)

    EastR.value(0)
```

```
    EastY.value(0)
    EastG.value(1)
    utime.sleep(4)

    EastG.value(0)
    EastY.value(1)
    utime.sleep(3)

    EastY.value(0)
    EastR.value(1)
    utime.sleep(1)

    NorthY.value(1)
    utime.sleep(2)

    NorthY.value(0)
    NorthR.value(0)
```

Рис.3.47: Программа: **traffic**.

3.15 Проект 14: Простой логический пробник

Описание: Это простой проект логического зонда. Логический пробник используется для индикации логического состояния неизвестного цифрового сигнала. В типичном приложении для обнаружения неизвестного сигнала используется щуп, а для индикации логического состояния используются два светодиода разного цвета. В этом проекте КРАСНЫЙ светодиод указывает на логический 0, а ЗЕЛЕНЫЙ светодиод указывает на логическую 1.

Принципиальная схема: На рис.3.48 показана принципиальная схема проекта. Неизвестный цифровой сигнал подается на порт GP13, КРАСНЫЙ и ЗЕЛЕНЫЙ светодиоды подключены к GP15 и GP14 соответственно.

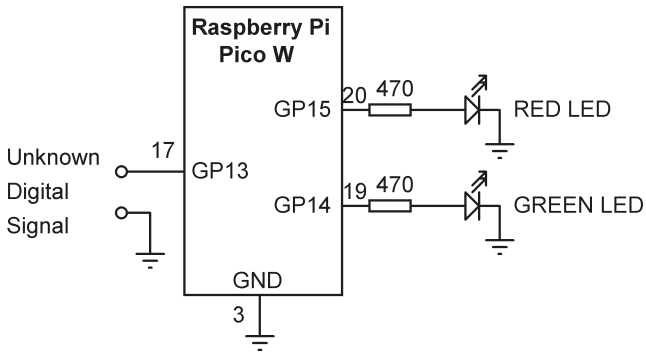


Рис.3.48: Принципиальная схема проекта.

Листинг программы: На рис. 3.49 показан листинг программы (Программа: **probe**).

```
#-------------------------------------------------------
#                ЛОГИЧЕСКИЙ ЗОНД
#               ==================
#
# Это программа логического зонда. Состояние (0 или 1)
# неизвестного цифрового сигнала отображается двумя светодиодами
#
# Автор: Доган Ибрагим
# Файл   : probe.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------
from machine import Pin

RED = Pin(15, Pin.OUT)                          # КРАСНЫЙ LED
GREEN = Pin(14, Pin.OUT)                        # ЗЕЛЕНЫЙ LED
PROBE = Pin(13, Pin.IN)                         # ЗОНД (ПРОБНИК)

while True:
    if PROBE.value() == 1:                      # Логическая 1?
        RED.value(0)                            # КРАСНЫЙ ВЫКЛ.
        GREEN.value(1)                          # ЗЕЛЕНЫЙ ВКЛ.
    else:                                       #  Логический 0
        RED.value(1)                            #  КРАСНЫЙ ВКЛ.
        GREEN.value(0)                          # ЗЕЛЕНЫЙ ВЫКЛ.
```

Рис.*3.49:* Программа:**probe**.

3.16 Проект 15: Расширенный логический пробник

Описание: проблема с этим логическим пробником заключается в том, что один из светодиодов всегда горит, даже если пробник не подключен к какому-либо цифровому сигналу или если сигнал находится в состоянии с высоким импедансом (т. е. в третьем состоянии). Вы можете развить проект дальше, чтобы также можно было обнаружить состояние высокого импеданса, и ни один из светодиодов не был включен.

Принципиальная схема: На рис.3.50 показана измененная принципиальная схема. Обратите внимание, что используется транзистор (BC108 или любой транзистор типа NPN) на входе схемы. Схема работает следующим образом:

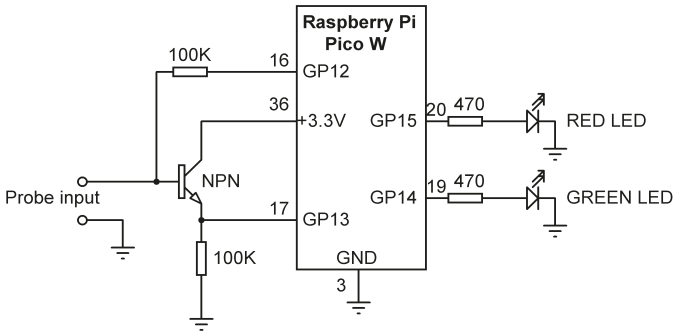


Рис.3.50: Модифицированная принципиальная схема.

Внешний неизвестный цифровой сигнал подается на базу транзистора. Этот сигнал с эммитера транзистора подается на GP13, который и определяет состояние цифрового сигнала.

Состояние зонда	Применяется к GP12	Обнаружен на GP13	ЗЕЛЕН. LED	КРАС. LED
Пробник с высоким импедансом	1	1	OFF	OFF
	0	0		
Пробник на логике 1	1	1	ON	OFF
	0	1		
Пробник на логике 0	1	0	OFF	ON
	0	0		

Таблица *3.4:* Примененные и обнаруженные логические уровни.

Листинг программы: На рис.3.51представлен листинг программы (Программа: probe2). Программа посылает сигналы от **TSTOUT** (GP12) и проверяет входы, как показано в таблице 3.4. КРАСНЫЙ и ЗЕЛЕНЫЙ светодиоды управляются, как описано в таблице.

```
#-----------------------------------------------------------
#                УСОВЕРШЕНСТВОВАННЫЙ ЛОГИЧЕСКИЙ ЗОНД
#                ========================================
#
# Это программа логического зонда. Состояние (0 или 1)
# неизвестного цифрового сигнала отображается двумя светодиодами. В этой
# версии программы также обнаружено высокоимпедансное состояние
#
# Автор: Доган Ибрагим
```

```
# Файл  : probe2.py
# Дата: октябрь 2022
г.#-------------------------------------------------------
--from machine import Pin

RED = Pin(15, Pin.OUT)                          # КРАСНЫЙ LED
GREEN = Pin(14, Pin.OUT)                        #  ЗЕЛЕНЫЙ LED
TSTIN = Pin(13, Pin.IN)                         # ЭММИТЕР ТРАНЗИСТОРА
TSTOUT = Pin(12, Pin.OUT)                       # БАЗА ТРАНЗИСТОРА

while True:                                     # ДЕЛАТЬ ВСЕГДА
    TSTOUT.value(1)
    if TSTIN.value() ==1:
        TSTOUT.value(0)
        if TSTIN.value() == 0:
            GREEN.value(0)
            RED.value(0)
        else:
            RED.value(0)
            GREEN.value(1)
    else:
         TSTOUT.value(0)
         if TSTIN.value() == 0:
             GREEN.value(0)
             RED.value(1)
```

Рис.*3.51:* Программа:**probe2**.

Глава 4 • Проекты Raspberry Pi Pico W с многоразрядным 7-сегментным дисплеем

4.1 Обзор

Многоразрядные 7-сегментные дисплеи требуют постоянного обновления своих цифр, чтобы человеческий глаз видел цифры устойчивыми и не мигающими. Общий используемый метод заключается в том, чтобы активировать каждую цифру на короткое время (например, 10 мс), чтобы человеческий глаз видел обе цифры включенными в любое время. Этот процесс требует, чтобы цифры активировались попеременно и непрерывно. В результате этого процессор не может выполнять никаких других задач и должен быть все время занят, обновляя цифры. Одним из методов, используемых в системах без многозадачности, является использование прерываний таймера и обновление цифр в подпрограммах обслуживания прерывания таймера. В этой главе вы будете использовать многозадачный подход для обновления цифр на дисплее, чтобы процессор мог выполнять другие задачи.

4.2 7-сегментные светодиодные индикаторы

Отображение данных является одним из основных выходных действий любой МК системы. Например, дисплеи используются для отображения данных датчиков, таких как температура, влажность, давление и т. д. Существует несколько типов устройств отображения, которые можно использовать в системах на основе МК. ЖК-дисплеи и 7-сегментные дисплеи, вероятно, являются двумя наиболее часто используемыми устройствами отображения. Существует несколько типов ЖК-дисплеев, таких как текстовые ЖК-дисплеи, графические ЖК-дисплеи, цветные ЖК-дисплеи и т. д. 7-сегментные дисплеи используются для отображения числовых или буквенно-цифровых значений, и они могут иметь одну или несколько цифр. Дисплеи с одной цифрой могут отображать только числа от 0 до 9. Дисплеи с двумя цифрами могут отображать числа от 0 до 99, дисплеи с тремя цифрами отображают числа от 0 до 999 и так далее.

Как показано на рис. 4.1, 7-сегментный LED дисплей в основном состоит из 7 LED, соединенных таким образом, что могут отображаться цифры от 0 до 9 и некоторые буквы. Сегменты обозначаются буквами от a до g. На рис. 4.2 показаны названия сегментов типичного 7-сегментного дисплея.

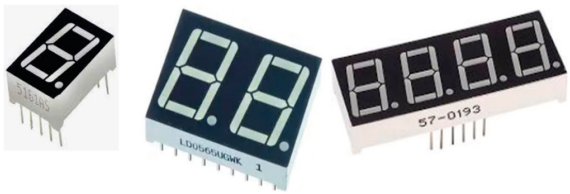


Рис.*4.1:* Некоторые *7*-сегментные индикаторы.

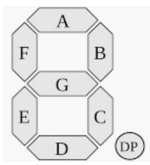


Рис.*4.2:* Распиновка сегментов *7*-сегментного индикаторы.

На рис.4.3 показано, как можно получить числа от 0 до 9, включая или выключая различные сегменты дисплея.

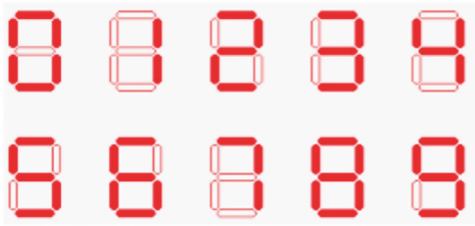


Рис.4.3: Отображение цифр 0–9.

7-сегментные LED дисплеи доступны в двух различных конфигурациях: с общим катодом (**OK**) и с общим анодом (**OA**). Как показано на рис. 4.4, в конфигурации с OK все катоды всех сегментных LED соединены вместе с землей. Затем сегменты включаются путем подачи логической 1 на светодиод необходимого сегмента через токоограничивающие резисторы. В конфигурации с **OK** 7-сегментный LED подключается к микроконтроллеру в режиме источника тока.

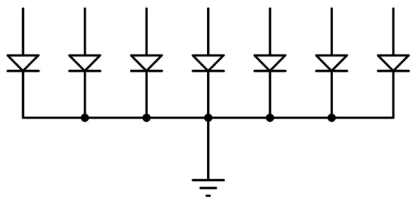


Рис.4.4: 7-сегментный светодиодный дисплей с общим катодом.

В конфигурации с **OA** анодные клеммы всех светодиодов соединены вместе, как показано на рис. 4.5. Затем эта общая точка обычно подключается к напряжению питания. Сегмент включается подключением его катодной клеммы к логическому 0 через токоограничивающий резистор. В конфигурации с **OA** 7-сегментный светодиод подключается к микроконтроллеру в режиме потребления тока.

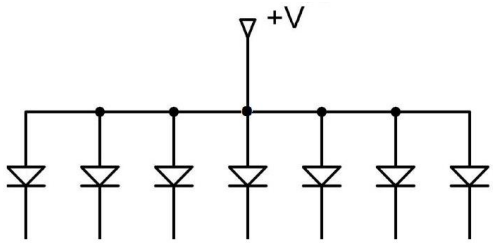


Рис.4.5: 7-сегментный светодиодный дисплей с общим анодом.

В приложениях с мультиплексированием LED (например, см. рис. 4.6 для 2-разрядного мультиплексированного LED дисплея) сегменты LED всех цифр связаны вместе, а общие выводы каждой цифры включаются МК отдельно. Отображая каждую цифру в течение нескольких миллисекунд, глаз не может отличить, что цифры не включены все время. Таким образом, вы можете мультиплексировать любое количество 7-сегментных дисплеев вместе.

Например, чтобы отобразить число 57, вы должны отправить 5 на первую цифру и включить ее общий вывод. Через несколько миллисекунд во вторую цифру отправляется число 7, и включается общая точка второй цифры. Когда этот процесс повторяется непрерывно, пользователь видит, что оба дисплея постоянно включены.

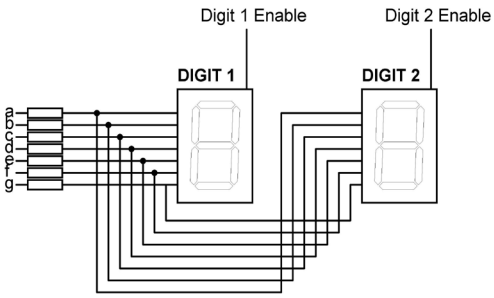


Рис.4.6: 2-разрядный мультиплексированный 7-сегментный светодиодный дисплей.

Некоторые производители предоставляют мультиплексированные многоразрядные дисплеи в отдельных упаковках. Например, вы можете приобрести 2-, 4- или 8-разрядные мультиплексированные дисплеи в одном пакете. В этой главе используется дисплей DC56-11EWA, представляющий собой красный двухразрядный мультиплексный дисплей высотой 0,56 дюйма с общим катодом и 18 контактами, конфигурация контактов которого показана в таблице 4.1. Два таких дисплея используются для создания 4-разрядного дисплея. В принципе, этим дисплеем можно управлять с микроконтроллера следующим образом:

- Отправьте битовый шаблон сегмента для цифры 1 в сегменты от a до g.
- Включить цифру 1.
- Подождите несколько миллисекунд.
- Отключить цифру 1.
- Отправьте битовый шаблон сегмента для цифры 2 в сегменты от a до g
- Включить цифру 2.
- Подождите несколько миллисекунд.
- Отключить цифру 2.
- Подождите несколько миллисекунд.
- Отправьте битовый шаблон сегмента для цифры 3 в сегменты от a до g.
- Включить цифру 3.
- Подождите несколько миллисекунд.
- Отключить цифру 3.
- Отправьте битовый шаблон сегмента для цифры 4 в сегменты от a до g.
- Включить цифру 4.
- Подождите несколько миллисекунд.
- Отключить цифру 4.
- Повторяйте описанный выше процесс непрерывно.

Pin no(s)	Segment
1,5	e
2,6	d
3,8	c
14	digit 1 Enable
17,7	g
15,10	b
16,11	a
18,12	f
13	digit 2 Enable
4	decimal Point1
9	decimal Point 2

Таблица 4.1: Распиновка двойного дисплея DC56-11EWA.

Конфигурация сегмента дисплея DC56-11EWA показана на рис. 4.7. В приложении мультиплексного дисплея выводы сегментов соответствующих сегментов соединены вместе. Например, контакты 11 и 16 соединены как общий сегмент. Точно так же контакты 15 и 10 соединяются как общий сегмент b и так далее.

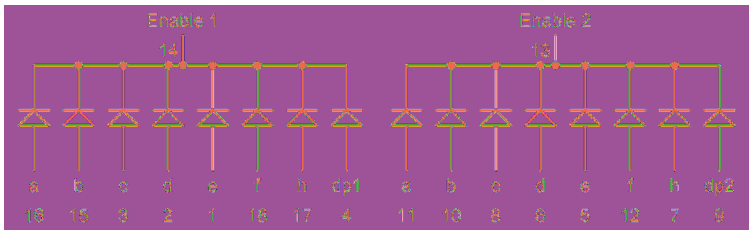


Рис.4.7: Конфигурация сегмента дисплея DC56-11EWA.

4.3 Проект 1: 4-разрядный 7-сегментный счетчик секунд

Описание: В этом проекте 7-сегментный 4-значный мультиплексированный светодиодный дисплей используется в качестве счетчика секунд от 0 до 9999.

Работа 4-разрядного мультиплексированного дисплея такова, что сегменты LED всех цифр связаны вместе, а общие выводы каждой цифры включаются МК отдельно. Отображая каждую цифру в течение нескольких миллисекунд, глаз не может отличить, что цифры не включены все время. Таким образом, вы можете мультиплексировать любое количество 7-сегментных дисплеев вместе. Например, чтобы отобразить число 5734, нужно отправить 5 на первую цифру и включить ее общий вывод. Через несколько миллисекунд во вторую цифру отправляется число 7, включается общая точка второй цифры и так далее. Когда этот процесс повторяется непрерывно, пользователь видит, что оба дисплея постоянно включены.

В этом проекте используется дисплей DC56-11EWA, красный двухразрядный мультиплексный дисплей высотой 0,56 дюйма с общим катодом и 18 контактами, конфигурация контактов которого показана в таблице 4.1. Два таких модуля дисплея используются для построения 4-разрядного дисплея. Каждый модуль имеет свои собственные контакты включения E1 и E2.

В приложении с мультиплексным дисплеем выводы сегментов соответствующих сегментов соединены вместе. Например, контакты 11 и 16 соединены как общий сегмент. Точно так же контакты 15 и 10 соединены как общий сегмент b и так далее.

Блок-схема: На рис. 4.8 показана блок-схема проекта.

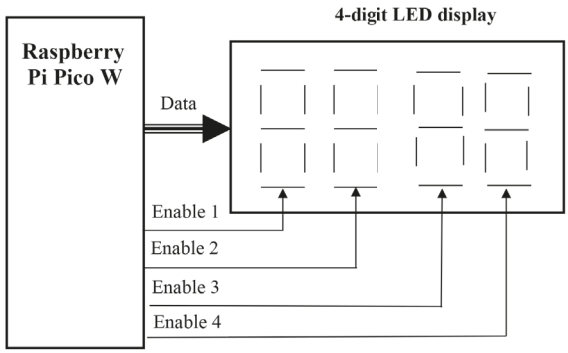


Рис.4.8: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис.4.9. В этом проекте используются следующие контакты Raspberry Pi Pico W для взаимодействия с 7-сегментным светодиодным дисплеем:

7-Segment Display pin	Raspberry Pi Pico W GPIO	Physical pin no
a	0	1
b	1	2
c	2	4
d	3	5
e	4	6
f	5	7
g	6	9
E1	7 (via transistor)	10
E2	8 (via transistor)	11
E1	9 (via transistor)	12
E2	10 (via transistor)	14

Сегменты 7-сегментного дисплея подаются от контактов порта через токоограничивающие резисторы сопротивлением 470 Ом. Разрешение цифр на контакты E1, E2 первого модуля и E1, E2 второго модуля подается от портов GP7, GP8, GP9 и GP10 соответственно через четыре NPN-транзистора типа BC108 (здесь можно использовать любой другой NPN-транзистор). Коллекторы этих транзисторов управляют цифрами сегмента.

Сегменты включаются, когда база соответствующего транзистора установлена в логическую 1. Обратите внимание, что следующие контакты дисплея соединены вместе, чтобы сформировать мультиплексный дисплей:

16 и 11, 15 и 10, 3 и 8, 2 и 6, 1 и 5, 17 и 7, 18 и 12

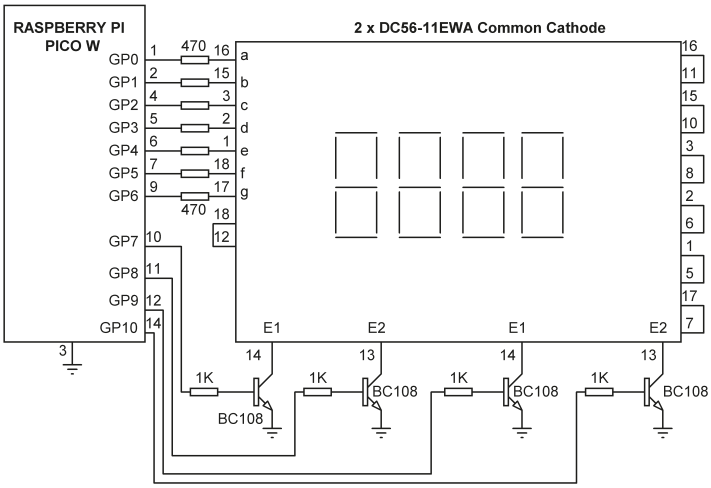


Рис.4.9: Принципиальная схема проекта.

Листинг программы: Перед тем, как управлять дисплеем, вы должны знать взаимосвязь между числами, которые должны отображаться, и соответствующими сегментами, которые должны быть включены, и это показано ниже:

Number to be displayed	**LED bit pattern (a,b,c,d,e,f,g)**
1.	1,1,1,1,1,1,0
2.	0,1,1,0,0,0,0
3.	1,1,0,1,1,0,1
4.	1,1,1,1,0,0,1
5.	0,1,1,0,0,1,1
6.	1,0,1,1,0,1,1
7.	1,0,1,1,1,1,1
8.	1,1,1,0,0,0,0
9.	1,1,1,1,1,1,1
10.	1,1,1,1,0,1,1

На рис.4.10 показан листинг программы (Program:SevenCount4). В начале программы импортируются используемые в программе модули. Здесь списокLED_Digitsсодержит 4 числа, которые являются контактами включения цифр четырех 7-сегментных светодиодных модулей. ФункцияRefresh вызывается прерыванием таймера для обновления дисплея. Если номер состоит менее чем из 4 цифр, то перед ним вставляются пробелы, чтобы скрыть эти цифры на дисплее. Обратите внимание, что количество цифр идет от 0 до 4, а не от 0 до 2, как в случае с 2-разрядным дисплеем. Задержка между включением цифры уменьшена до 5 мс, так как у вас 4 цифры и требуется более высокая частота обновления.

По умолчанию переменная **count** начинается с 0 в начале программы. Он увеличивается на 1 каждую секунду. Когда счетчик достигает 10000, прерывание таймера отключается и подсчет останавливается.

```
#-------------------------------------------------------
#          4-ЗНАЧНЫЙ 7-СЕГМЕНТНЫЙ СЧЕТЧИК СЕКУНД
#         ========================================
#
# В этой программе к Pico подключается 4-разрядный 7-сегментный дисплей.
# Программа считает каждую секунду.
# В этой версии программы ведущие нули опущены
#
# Автор: Доган Ибрагим
# Файл   : SevenCount4.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------
from machine import Pin, Timer
import utime

tim = Timer()
LED_Segments = [0, 1 ,2, 3, 4, 5, 6]
LED_Digits = [7, 8, 9, 10]
L = [0]*7
D = [0, 0, 0, 0]

#
# битовый шаблон светодиода для всех цифр 0-9
#
LED_Bits ={
' ':(0,0,0,0,0,0,0),           # пусто
'0':(1,1,1,1,1,1,0),           # 0
'1':(0,1,1,0,0,0,0),           # 1
'2':(1,1,0,1,1,0,1),           # 2
'3':(1,1,1,1,0,0,1),           # 3
'4':(0,1,1,0,0,1,1),           # 4
'5':(1,0,1,1,0,1,1),           # 5
'6':(1,0,1,1,1,1,1),           # 6
'7':(1,1,1,0,0,0,0),           # 7
'8':(1,1,1,1,1,1,1),           # 8
'9':(1,1,1,1,0,1,1)}           # 9

count = 0                      # Инициализация счетчика

#
# Эта функция настраивает светодиодные порты как выходы
#
def Configure_Port():
```

```
    for i in range(0, 7):
          L[i] = Pin(LED_Segments[i], Pin.OUT)

    for i in range(0, 4):
          D[i] = Pin(LED_Digits[i], Pin.OUT)


#
# Обновить 7-сегментный дисплей
#
def Refresh(timer):                              # Обновление потока
    global count
    cnt = str(count)                             # в строку
    if len(cnt) == 3:                            # 3 цифры?
       cnt = " " + cnt                           # Убедитесь, что 4 цифры
    elif len(cnt) == 2:                          # 2 цифры?
        cnt = "  " + cnt                         # Убедитесь, что 4 цифры
    elif len(cnt) == 1:                          # 1 цифра?
        cnt = "   " + cnt                        # Убедитесь, что 4 цифры
    for dig in range(4):                         # Выполнить для 4 цифр
       for loop in range(0,7):
           L[loop].value(LED_Bits[cnt[dig]][loop])
       D[dig].value(1)
       utime.sleep(0.005)
       D[dig].value(0)

#
# Настроить PORT на все выходы
#
Configure_Port()

#
#  Основной цикл программы. Запустите периодический таймер и подсчет
#
tim.init(freq=50, mode=Timer.PERIODIC, callback=Refresh)

while True:                              # Делать всегда
     utime.sleep(1)                      # ждите секунду
     count = count + 1                   # Увеличение счетчика
     if count == 10000:                  #  Если count = 10000 стоп
       tim.deinit()                      # Остановить таймер
```

Рис.*4.10:* Программа:**SevenCount4**.

4.4 Проект 2: 4-разрядный 7-сегментный дисплей счетчик товаров конвейерной ленты

Описание: В этом проекте подсчитывается количество предметов (например, бутылок), проходящих по конвейерной ленте, и непрерывно отображаете результат на 7-сегментном дисплее.

Блок-схема. На рис. 4.11 показана система. Предполагается, что вы хотите подсчитать количество бутылок, проходящих по конвейерной ленте. Луч света направляется в точку на проходящие бутылки. На другой стороне луча фоторезистор (**LDR**) используется для обнаружения прерывания светового луча. Когда это происходит, на макетную плату посылается сигнал, который затем увеличивает счетчик и отображает общее количество на дисплее.

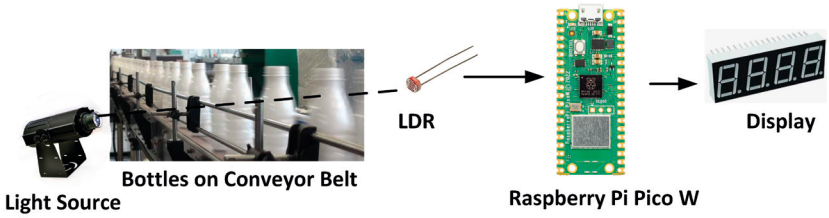


Рис.4.11: Блок-схема проекта.

Фоторезистор

Фоторезистор — это просто резистор, сопротивление которого изменяется при воздействии света на его поверхность (рис. 4.12). Сопротивление LDR увеличивается по мере уменьшения интенсивности света, падающего на устройство.

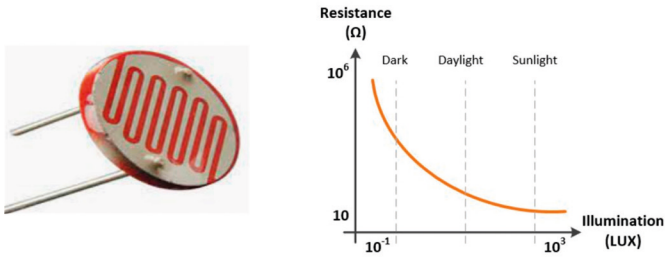


Рис.4.12: LDR и его типичная характеристика.

LDR обычно используются последовательно с резисторами в схеме для создания делителя напряжения. Напряжение на выходе схемы делителя потенциала используется для отправки триггерного сигнала, когда интенсивность света ниже (или выше) установленного уровня. Точка запуска может быть обнаружена МК с использованием аналоговых или цифровых входов.

Принципиальная схема: На рис. 4.13 показана принципиальная схема проекта. Потенциометр на 5 кОм используется последовательно с LDR. Потенциометр отрегулирован таким образом, чтобы выходное напряжение превышало 3 В, когда свет, падающий на LDR, уменьшается из-за проходящей бутылки по конвейерной ленте. Выход схемы делителя напряжения подается на GP11 макетной платы Raspberry Pi Pico W.

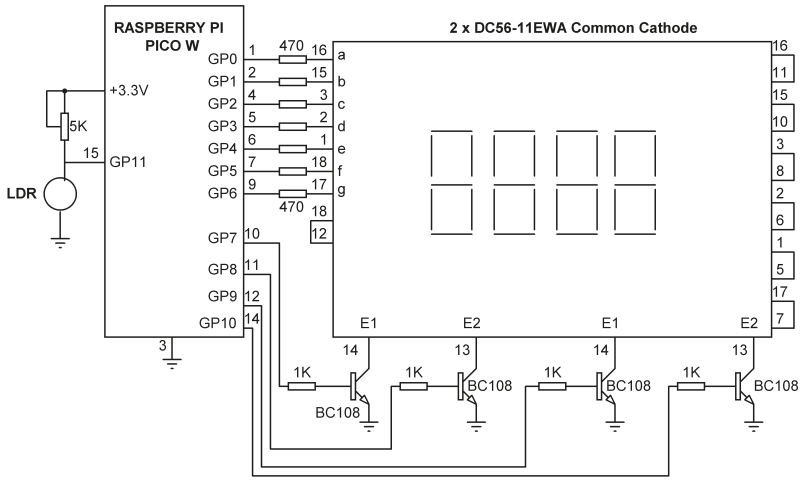


Рис.4.13: Принципиальная схема проекта.

Автором было измерено, что сопротивление поставляемого фоторезистора составляет около 0,5 кОм на свету и увеличивается примерно до 100 кОм в темноте. Предполагая, что потенциометр установлен в точку среднего плеча, т. е. 2,5 кОм:

Напряжение на выходе, когда свет падает на фоторезистор, равно

Vo = 3,3 В × 0,5 кОм / (2,5 кОм + 0,5 кОм) = 0,55 В, что соответствует логическому 0.

Точно так же напряжение на выходе в темноте:

Vo = 3,3 В × 100 кОм / (2,5 кОм + 100 кОм) = 3,21 В, что соответствует логической 1.

Установите ползунок потенциометра чуть ниже его средней точки, чтобы выходное напряжение было даже ниже 0,55 В, когда светло. Точную точку можно легко определить опытным путем.

Листинг программы: Рисунок 4.14 показывает листинг программы (Программа: **Conveyor**). Большинство частей этой программы такие же, как и в предыдущей программе. Здесь фоторезистор назначается порту **GP11**. Внутри основного цикла программы считывается его вывод, и программа ожидает, если он равен 0 (т. е. перед ф/резистором нет бутылок). Когда передним находится бутылка, его выход переходит в логическую 1. Это обнаруживается программой, и переменная **count** увеличивается на 1. Программа ждет, пока бутылка не пройдет мимо ф/резистора.

```
#-------------------------------------------------------
#               СЧЕТЧИК ТОВАРОВ КОНВЕЙЕРНОЙ ЛЕНТЫ
#               ==================================
#
# В этой программе к Pico подключается 4-разрядный 7-сегментный дисплей.
# Также используется ЖК-дисплей. Программа подсчитывает количество товаров,
# прошедших по конвейерной ленте, и отображает общее количество
#
# Автор: Доган Ибрагим
# Файл   : Conveyor.py
# Дата: октябрь 2022 г
#-------------------------------------------------------
from machine import Pin, Timer
import utime

tim = Timer()
LED_Segments = [0, 1 ,2, 3, 4, 5, 6]
LED_Digits = [7, 8, 9, 10]
L = [0]*7
D = [0, 0, 0, 0]
LDR = Pin(11, Pin.IN)

#
# битовый шаблон светодиода для всех цифр 0-9
#
LED_Bits ={
' ':(0,0,0,0,0,0,0),           # пусто
'0':(1,1,1,1,1,1,0),           # 0
'1':(0,1,1,0,0,0,0),           # 1
'2':(1,1,0,1,1,0,1),           # 2
'3':(1,1,1,1,0,0,1),           # 3
'4':(0,1,1,0,0,1,1),           # 4
'5':(1,0,1,1,0,1,1),           # 5
'6':(1,0,1,1,1,1,1),           # 6
'7':(1,1,1,0,0,0,0),           # 7
'8':(1,1,1,1,1,1,1),           # 8
'9':(1,1,1,1,0,1,1)}           # 9

count = 0                      # Инициализация счетчика

#
# Эта функция настраивает светодиодные порты как выходы
#
def Configure_Port():
   for i in range(0, 7):
         L[i] = Pin(LED_Segments[i], Pin.OUT)
```

```
   for i in range(0, 4):
         D[i] = Pin(LED_Digits[i], Pin.OUT)


#
# Обновить 7-сегментный дисплей
#
def Refresh(timer):                              # Обновление потока
   global count
   cnt = str(count)                              # в строку
   if len(cnt) == 3:                             # 3 цифры?
      cnt = " " + cnt                            # Убедитесь, что 4 цифры
   elif len(cnt) == 2:                           # 2 цифры?
       cnt = "  " + cnt                          # Убедитесь, что 4 цифры
   elif len(cnt) == 1:                           # 1 цифра?
       cnt = "   " + cnt                         # Убедитесь, что 4 цифры
   for dig in range(4):                          # Выполнить для 4 цифр
      for loop in range(0,7):
          L[loop].value(LED_Bits[cnt[dig]][loop])
      D[dig].value(1)
      utime.sleep(0.005)
      D[dig].value(0)

#
# Настроить PORT на все выходы
#
Configure_Port()

#
# Основной цикл программы. Запустите периодический таймер и подсчет
#
tim.init(freq=50, mode=Timer.PERIODIC, callback=Refresh)

while True:                                      # Делать всегда
     while LDR.value() == 0:                     # Дождаться наступления темноты
        pass
     count = count + 1                           # Увеличение счетчика
     utime.sleep(0.1)
     while LDR.value() == 1:                     # Подождать, пока снова загорится свет
        pass
```

Рис. *4.14:* Программа: **Conveyor**.

Глава 5 • ЖК-проекты Raspberry Pi Pico W

5.1 Обзор

В системах на основе МК вы обычно должны взаимодействовать с системой, например, чтобы ввести параметр, изменить значение параметра или отобразить вывод измеряемой переменной. Данные обычно вводятся в систему с помощью небольшой клавиатуры или полноразмерной клавиатуры. Данные обычно отображаются с помощью индикатора, такого как один или несколько светодиодов, 7-сегментный дисплей или ЖК-дисплей. ЖК-дисплеи имеют то преимущество, что они могут отображать как буквенно-цифровые, так и графические данные. Некоторые ЖК-дисплеи имеют длину 40 или более символов и могут отображать данные в несколько строк. Некоторые другие ЖК-дисплеи можно использовать для отображения графических изображений (графические ЖК-дисплеи или просто GLCD), таких как анимация. Некоторые дисплеи являются одноцветными или многоцветными, в то время как некоторые другие имеют подсветку, чтобы их можно было просматривать в условиях слабого освещения.

ЖКИ могут быть подключены к МК как параллельно, так и через интерфейс. Параллельные ЖК-дисплеи (например, Hitachi HD44780) подключаются с использованием более одной линии данных и нескольких линий управления, и данные передаются в параллельной форме. Обычно используют 4 или 8 линий данных и две или более линий управления. Использование 4-проводного соединения экономит контакты ввода-вывода, но работает медленнее, поскольку данные передаются в два этапа. С другой стороны, ЖК-дисплеи подключаются к МК с помощью всего двух проводов: данных и тактирования. ЖК-дисплеи, основанные на параллельном подключении, в целом намного проще в использовании и требуют меньше соединении, но они стоят дороже, чем параллельные. В этой главе вы научитесь использовать в проектах и те и другие ЖК-дисплеи.

Программирование ЖК-дисплеев является сложной задачей и требует хорошего понимания внутренних операций контроллеров ЖК-дисплеев, включая знание их точных требований к синхронизации. К счастью, есть несколько библиотек, которые можно использовать для упрощения использования как параллельных, так и последовательных ЖК-дисплеев.

5.2 Параллельный ЖК-модуль HD44780

Хотя существует несколько типов ЖК-дисплеев, в настоящее время тип HD44780 является одним из самых популярных ЖК-модулей, используемых в промышленности, а также среди любителей. Этот модуль представляет собой буквенно-цифровой монохромный дисплей различных размеров. Модули с 16 столбцами популярны в большинстве небольших приложений, но доступны и другие модули с 8, 20, 24, 32 или 40 столбцами. Хотя большинство ЖК-дисплеев стандартно имеют две строки (или строки), можно приобрести модели с 1 или 4 строками. ЖК-дисплеи доступны со стандартными 14-контактными разъемами, хотя также доступны 16-контактные модули с разъемами для подсветки. В Таблице 3.4 приведены распиновка и функции контактов 16-контактного ЖК-модуля. Краткое описание функций штифта приведено ниже:

Пин	Имя	Функция
1	VSS	Земля
2	VDD	+ ve supply
3	VEE	Контрастность
4	RS	Выбор регистра
5	R/W	Read/write
6	E	Включить
7	D0	Бит данных 0
8	D1	Бит данных 1
9	D2	Бит данных 2
10	D3	Бит данных 3
11	D4	Бит данных 4
12	D5	Бит данных 5
13	D6	Бит данных 6
14	D7	Бит данных 7
15	A	Анод подсветки (+)
16	K	Катод подстветки (GND)

Таблица 5.1:Распиновка ЖК-модуля HD44780.

VSS (контакт 1) и VDD (контакт 2) — это контакты заземления и питания. Питание должно быть +5 В.

VEE — это контакт 3, и это контакт управления контрастностью, используемый для регулировки контрастности дисплея. Контрастность дисплея регулируется вращением ручки потенциометра.

Контакт 4 — это выбор регистра (RS), и когда этот контакт имеет LOW уровень, данные, передаваемые на дисплей, обрабатываются как команды. Когда RS HIGH, символьные данные могут передаваться на дисплей и с дисплея.

Контакт 5 — это линия чтения/записи (R/W). На этот контакт подается НИЗКИЙ уровень для записи команд или символьных данных на ЖК-модуль. Когда на этом выводе высокий уровень, из модуля можно считывать символьные данные или информацию о состоянии. Этот контакт обычно постоянно подключен к LOW, чтобы команды или символьные данные могли быть отправлены на ЖК-модуль.

Включение (E) — это контакт 6, который используется для инициации передачи команд или данных между ЖК-модулем и микроконтроллером. При записи на дисплей данные передаются только при переходе этого вывода из ВЫСОКОГО в НИЗКИЙ. При чтении с дисплея данные становятся доступными после перехода с НИЗКОГО на ВЫСОКИЙ уровень вывода разрешения, и эти данные остаются действительными до тех пор, пока на выводе разрешения установлен логический ВЫСОКИЙ уровень.

Контакты с 7 по 14 — это восемь линий шины данных (от D0 до D7). Данные могут передаваться между МК и ЖК-модулем с помощью либо одного 8-битного байта, либо двух 4-битных полубайтов. В последнем случае используются только верхние четыре линии данных (от D4 до D7). Преимущество 4-битного режима состоит в том, что для связи с ЖК-дисплеем требуется на четыре линии ввода-вывода меньше. Однако 4-битный режим медленнее, поскольку данные передаются в два этапа. В этой книге будете использоваться только 4-битный интерфейс.

Контакты 15 и 16 предназначены для управления яркостью.

В 4-битном режиме используются следующие контакты ЖК-дисплея. Линия R/W постоянно соединена с землей. В этом режиме используются 6 контактов порта GPIO микроконтроллера (МК):

VSS, VDD, VEE, E, R/S, D4, D5, D6, D7

В следующих разделах этой главы будут рассматриваться проекты с использованием ЖК-дисплеев. Но прежде , стоит рассмотреть основные детали шиныI2C.

5.3 Шина I2C

Шина I2C является одним из наиболее часто используемых протоколов связи микроконтроллеров для связи с внешними устройствами, такими как датчики и исполнительные механизмы. Шина представляет собой шину с одним ведущим и несколькими подчиненными и может работать в стандартном режиме: 100 Кбит/с, полной скорости: 400 Кбит/с, быстром режиме: 1 Мбит/с и высокой скорости: 3,2 Мбит/с. Шина состоит из двух проводов с открытым заземлителем, подтянутыми резисторами:

SDA: линия данных
SCL: линия синхронизации

На рис.5.1 показана структура шины I2C с одним ведущим и тремя подчиненными.

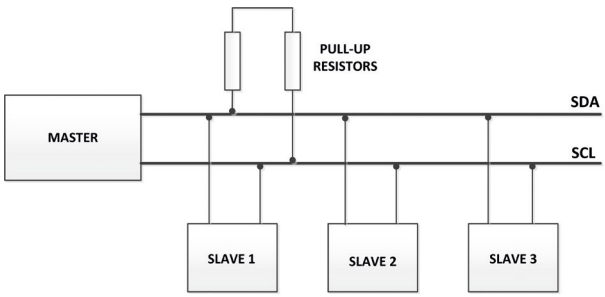


Рис.5.1: Шина I2C с одним ведущим и тремя подчиненными.

Поскольку шина I2C основана всего на двух проводах, должен быть способ адресации отдельного ведомого устройства на той же шине. По этой причине протокол определяет, что каждое ведомое устройство предоставляет уникальный ведомый адрес для данной шины. Этот адрес обычно имеет ширину 7 бит. Когда шина свободна, обе линии имеют ВЫСОКИЙ уровень. Все коммуникации на шине инициируются и завершается мастером, который сначала отправляет бит START

и завершает транзакцию, отправляя бит STOP. Это предупреждает все ведомые устройства о том, что по шине приходят какие-то данные, и все ведомые устройства прослушивают шину. После стартового бита отправляются 7 бит уникального адреса ведомого устройства. Каждое ведомое устройство на шине имеет свой собственный адрес, и это гарантирует, что только адресованное ведомое устройство обменивается данными по шине в любое время, чтобы избежать каких-либо коллизий. Последний отправленный бит является битом области/записи, так что если этот бит равен 0, это означает, что мастер хочет записать на шину (например, в регистр ведомого), если этот бит равен 1, это означает, что мастер желает читать из шины (например, из регистра ведомого устройства). Данные отправляются по шине битом MSB первым. Бит подтверждения (ACK) идет после каждого байта, и этот бит позволяет приемнику сигнализировать передатчику, что байт был успешно принят, и в результате может быть отправлен еще один байт. Бит ACK отправляется на 9-м тактовом импульсе.

Связь по шине I2C выглядит следующим образом:

- Ведущий отправляет по шине адрес ведомого, с которым он хочет установить связь.
- LSB — это бит R/W, устанавливающий направление передачи данных, то есть от ведущего к ведомому (R/W = 0) или от ведомого к ведущему (R/W = 1).
- Отправляются требуемые байты, каждый из которых чередуется с битом ACK, до тех пор, пока не возникнет условие STOP.

В зависимости от типа используемого ведомого устройства для некоторых транзакций может потребоваться отдельная транзакция. Например, шаги для чтения данных с совместимого запоминающего устройства:

- Мастер начинает транзакцию в режиме записи (R/W = 0), отправляя адрес ведомого по шине.
- Извлекаемая ячейка памяти затем отправляется в виде двух байтов (при условии, что память имеет размер 64 Кбит).
- Мастер посылает условие STOP для завершения транзакции.
- Мастер начинает новую транзакцию в режиме чтения (R/W = 1), отправляя адрес ведомого по шине.
- Мастер считывает данные из памяти. При чтении памяти в непоследовательном формате будет прочитано более одного байта.
- Мастер устанавливает на шине состояние STOP.

5.4 Пины Raspberry Pi Pico W

Raspberry Pi Pico и Pico W имеют два вывода I2C, названные I2C0 и I2C1 (см. рис. 5.2). Как показано на рисунке, выводы дублируются и используются совместно с другими выводами. Например, GP0 (контакт 1) — это контакт I2C0 SDA, а GP1 (контакт 1) — это контакт I2C0 SCL. Кроме того, GP16 (вывод 21) — это вывод I2C0 SDA, а GP17 (вывод 22) — вывод I2C SCL.

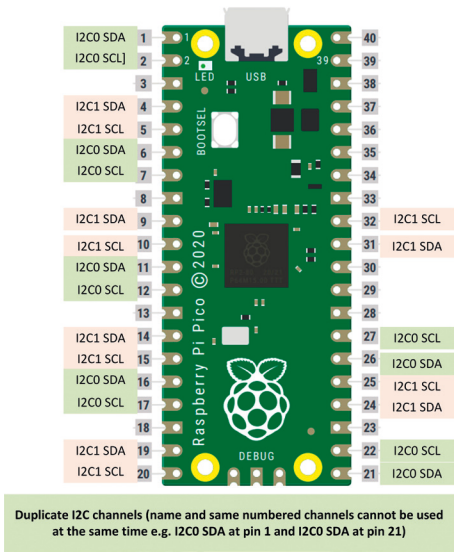


Рис. 5.2: Распиновка Raspberry Pi Pico и Pico W I2C.

Пины по умолчанию:

I2C0 SCL	GP9
I2C0 SDA	GP8
I2C1 SCL	GP7
I2C1 SDA	GP6

5.5 Проект 1: параллельный ЖК-дисплей — отображение текста

Описание: В этом проекте разработаем ряд функций, которые можно использовать для отправки данных и текста на параллельный ЖК-дисплей размером 16×2 символа.

Цель: Целью этого проекта является разработка библиотеки функций, которые можно использовать для управления параллельным ЖК-дисплеем. Эти функции можно использовать в проектах для отправки текста и чисел на ЖК-дисплей.

Блок-схема: На рис. 5.3 показана блок-схема проекта.

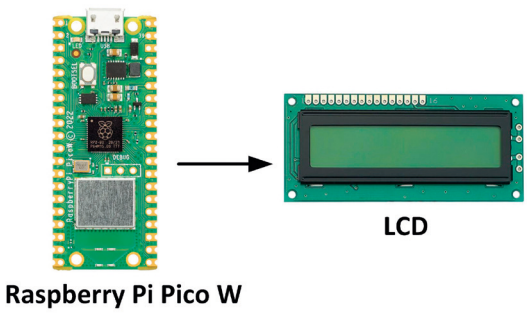


Рис. 5.3: Блок-схема проекта.

Принципиальная схема: Принципиальная схема: Принципиальная схема проекта показана на рис.5.4. ЖК-дисплей подключается к Pico с помощью четырех проводов данных (D4–D7) и двух проводов управления (E и R/S). Соединения между ЖК-дисплеем и Pico следующие:

LCD pin	**Pico pin**
E	GP0
R/S	GP1
D4	GP2
D5	GP3
D6	GP4
D7	GP5

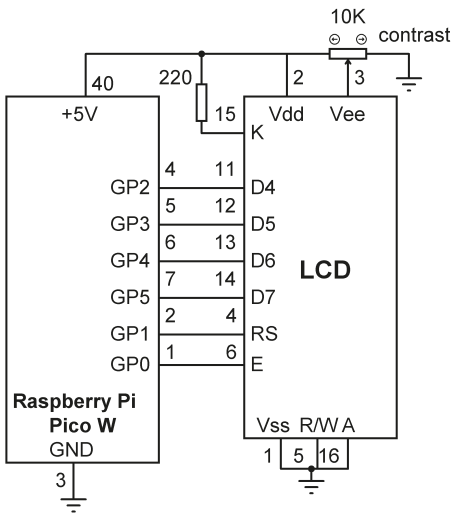


Рис.5.4: Принципиальная схема проекта.

Контрастность ЖК-дисплея регулируется потенциометром на 10 кОм. Обратите внимание, что ЖК-дисплей работает с напряжением +5 В. Порты ввода/вывода Pico не совместимы с +5 В, но это не проблема для Pico, поскольку все контакты ЖК-дисплея находятся в режиме ввода. **(убедитесь, что контакт R/W подключен к GND перед включением схемы)**.

Листинг программы: На рис. 5.5 показан листинг программы (**Программа**: ЖК-дисплей). Соединения между ЖК-дисплеем и Pico определяются в начале, и при желании их можно изменить. Остальные функции не следует изменять для правильного управления ЖК-дисплеем. Эти функции реализуют инициализацию и управление ЖК-дисплеем.

Доступны следующие функции управления ЖК-дисплеем:

lcd_init: это функция инициализации ЖК-дисплея, и она должна вызываться первой, прежде чем будут вызваны любые другие функции.

lcd_clear: очистка экрана

lcd_home: размещает курсор (верхнее левое положение)

lcd_cursor_blink: включает мигающий курсор

lcd_cursor_on: включает видимый курсор

lcd_cursor_off: отключает видимый курсор

lcd_puts(s): отображает строку s

lcd_putch(c): отображает символ c

lcd_goto(col,row): устанавливает курсор в указанный столбец и позицию. (0, 0) — левый угол ЖК-дисплея. Первая строка — это строка 0, вторая строка — это строка 1 и так далее.

```
#-------------------------------------------------------------
#                        ПАРАЛЛЕЛЬНЫЕ  ФУНКЦИИ LCD
#                        ============================
#
# Эти функции инициализируют и управляют параллельным ЖК-дисплеем
#
# Автор: Доган Ибрагим
# Файл   : LCD
# Дата: октябрь 2022 г
#-------------------------------------------------------------
from machine import Pin
import utime

EN = Pin(0, Pin.OUT)
RS = Pin(1, Pin.OUT)
D4 = Pin(2, Pin.OUT)
D5 = Pin(3, Pin.OUT)
D6 = Pin(4, Pin.OUT)
D7 = Pin(5, Pin.OUT)
PORT = [2, 3, 4, 5]
L = [0,0,0,0]

def Configure():
    for i in range(4):
       L[i] = Pin(PORT[i], Pin.OUT)

def lcd_strobe():
    EN.value(1)
    utime.sleep_ms(1)
    EN.value(0)
    utime.sleep_ms(1)
```

```
def lcd_write(c, mode):
    if mode == 0:
       d = c
    else:
       d = ord(c)
    d = d >> 4
    for i in range(4):
        b = d & 1
        L[i].value(b)
        d = d >> 1
    RS.value(mode)
    lcd_strobe()

    if mode == 0:
       d = c
    else:
       d = ord(c)
    for i in range(4):
        b = d & 1
        L[i].value(b)
        d = d >> 1
    RS.value(mode)
    lcd_strobe()
    utime.sleep_ms(1)
    RS.value(1)

def lcd_clear():
    lcd_write(0x01, 0)
    utime.sleep_ms(5)

def lcd_home():
    lcd_write(0x02, 0)
    utime.sleep_ms(5)

def lcd_cursor_blink():
    lcd_write(0x0D, 0)
    utime.sleep_ms(1)

def lcd_cursor_on():
    lcd_write(0x0E, 0)
    utime.sleep_ms(1)

def lcd_cursor_off():
    lcd_write(0x0C, 0)
    utime.sleep_ms(1)
```

```
def lcd_puts(s):
    l = len(s)
    for i in range(l):
       lcd_putch(s[i])

def lcd_putch(c):
    lcd_write(c, 1)

def lcd_goto(col, row):
    c = col + 1
    if row == 0:
        address = 0
    if row == 1:
        address = 0x40
    address = address + c - 1
    lcd_write(0x80 | address, 0)

def lcd_init():
    Configure()
    utime.sleep_ms(120)
    for i in range(4):
        L[i].value(0)
    utime.sleep_ms(50)
    L[0].value(1)
    L[1].value(1)
    lcd_strobe()
    utime.sleep_ms(10)
    lcd_strobe()
    utime.sleep_ms(10)
    lcd_strobe()
    utime.sleep_ms(10)
    L[0].value(0)
    lcd_strobe()
    utime.sleep_ms(5)
    lcd_write(0x28, 0)
    utime.sleep_ms(1)
    lcd_write(0x08, 0)
    utime.sleep_ms(1)
    lcd_write(0x01, 0)
    utime.sleep_ms(10)
    lcd_write(0x06, 0)
    utime.sleep_ms(5)
    lcd_write(0x0C, 0)
    utime.sleep_ms(10)
#================= END OF LCD FUNCTIONS ======================
```

```
lcd_init()
lcd_puts('Hello from PICO')
```

Рис.*5.5:* Программа: **LCD**.

Отображение текста
Следующий оператор отображает на ЖК-дисплее текст **Hello from PICO**:

lcd_init()

lcd_puts('Hello from PICO')

Рис.5.6. Отображение текста

Рис.*5.6.* Отображение текста

Хранение функций LCD в библиотеке модулей
Вы можете легко объединить все функции ЖК-дисплея в библиотеку, а затем импортировать эту библиотеку в начале вашей программы. Шаги приведены ниже:

- Откройте программу **LCD** (см. рис. 5.5).
- Удалите два последних утверждения.
- Нажмите Тонни: **File**, а затем **Save As**.
- Выберите **Raspberry Pi Pico** в качестве пункта назначения.
- Задайте имя файла как LCDFuncs.py и нажмите **OK**.

Теперь вы можете импортировать библиотечные LCDFuncs в свои программы на основе LCD. Например, программу, представленную в этом проекте, можно переписать, как показано на рис. 5.7 (Программа: LCD2). Обратите внимание, что всем функциям LCD должно предшествовать слово LCDFuncs.

```
#-----------------------------------------------------------
#                        ПАРАЛЛЕЛЬНЫЕ  ФУНКЦИИ LCD
#                        ==========================
#
# Эти функции инициализируют и управляют параллельным ЖК-дисплеем
#
# Автор: Доган Ибрагим
# Файл   : LCD2
```

```
# Дата: октябрь 2022 г.
#------------------------------------------------------------
import utime
import LCDFuncs

LCDFuncs.lcd_init()

LCDFuncs.lcd_puts('Hello again...')
```

Figure 5.7: Program: **LCD2**.

5.6 Проект 2: Таймер реакции с параллельным ЖК-дисплеем

Описание: Это игра с таймером реакции. Идея игры заключается в измерении времени реакции пользователя. Игра состоит из ЖК-дисплея, светодиода и кнопки. Игра начинается с того, что пользователь держит руку на кнопке. Светодиод включается в случайное время, и как только он загорается, ожидается, что пользователь нажмет кнопку. Время, прошедшее между включением светодиода и нажатием кнопки, измеряется и отображается на ЖК-дисплее как время реакции пользователя.
Блок-схема: На рис. 5.8 показана блок-схема проекта.

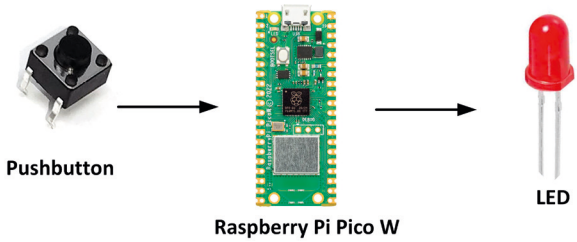


Рис.5.8: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рисунке 5.9. LCD подключен как и в предыдущем проекте. Светодиод и кнопка подключены к **GP16** и **GP17** соответственно. Внутренний подтягивающий резистор используется на выводе **GP17**, так что нет необходимости использовать внешний резистор.

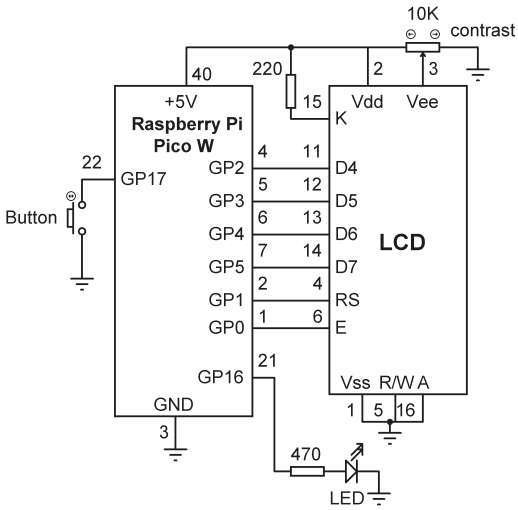


Рис.5.9: Принципиальная схема проекта.

Листинг программы: На рис. 5.10 показан листинг программы (Program: **Reaction**). В начале программы импортируются ЖК-модуль, машина, utime и случайный модуль. Button и LED назначаются GP17 и GP16 соответственно. GP17 внутренне подтягивается, так что состояние этого вывода по умолчанию равно логической 1 и переходит в логическую 0 при нажатии кнопки. Затем библиотека LCD инициализируется вызовом **functionlcd_init**.

В начале основной программы светодиод гаснет, и программа входит в цикл **while**. Внутри этого цикла ЖК-дисплей очищается и генерируется случайное число от 3 до 10. Это значение используется для задержки включения светодиода, чтобы пользователь не знал, когда светодиод загорится. В этот момент загорается светодиод, запускается таймер и включается кнопка прерывания. Прерывание сконфигурировано так, чтобы оно активировалось по заднему фронту выхода кнопки (т. е. при нажатии кнопки). Затем основная программа ожидает, пока прерывание не будет обслужено (т. е. пока **flag** равен 0).

Программа переходит к функции **MyButton**, как только нажимается кнопка. Вначале этой программы дальнейшие прерывания запрещены установкой **handler = None**. Внутри этой функции светодиод гаснет, а таймер останавливается. Прошедшее время рассчитывается путем вычитания текущего времени из времени, когда светодиод был включен. Это время преобразуется в строку и сохраняется в переменной **ReactionStrand** и отображается во второй строке ЖК-дисплея в миллисекундах. Текст **Reaction Time:** отображается в первой строке ЖК-дисплея. Например, если время реакции составляет 500 мс, оно отображается в следующем формате:

Reaction Time:
500

Затем переменная **flags** устанавливается в 1, чтобы основная программа продолжала работать. В то же время ЖК-дисплей очищается и игра перезапускается.

```
#------------------------------------------------------------
#                              ТАЙМЕР РЕАКЦИИ
#                              ===============
#
# Это программа-таймер реакции, которая
# измеряет реакцию пользователя и отображает ее на ЖК-дисплее в миллисекундах.
# Для быстрой реакции пользователь должен нажать кнопку
# только загорится светодиод
#
# Автор: Доган Ибрагим
# Файл  : Reaction.py
# Дата: октябрь 2022 г.
#------------------------------------------------------------
import LCDFuncs
from machine import Pin
import utime
import random

Button = Pin(17, Pin.IN, Pin.PULL_UP)
LED = Pin(16, Pin.OUT)
LCDFuncs.lcd_init()
flag = 0

#
# Это процедура обслуживания прерывания. Программа
# перескакивает сюда, как только нажимается кнопка
#
def MyButton(pin):
    global flag
    Button.irq(handler = None)
    LED.value(0)
    TmrEnd = utime.ticks_ms()
    ReactionTime = utime.ticks_diff(TmrEnd, TmrStart)
    ReactionStr = str(ReactionTime)
    flag = 1
    LCDFuncs.lcd_puts("Reaction Time:")
    LCDFuncs.lcd_goto(0, 1)
    LCDFuncs.lcd_puts(ReactionStr)
    utime.sleep(3)
    LCDFuncs.lcd_clear()

#
# Запуск ГЛАВНОЙ программы
#
LED.value(0)
while True:
```

```
flag = 0
LCDFuncs.lcd_clear()
rnd = random.randint(3, 10)
utime.sleep(2)
utime.sleep(rnd)
LED.value(1)
TmrStart = utime.ticks_ms()
Button.irq(handler=MyButton, trigger = Pin.IRQ_FALLING)
while flag == 0:
   pass
```

Рис.*5.10:* Программа:**Reaction**.

5.7 Проект 3: Вольтметр с параллельным ЖК-дисплеем

Описание: Это простой проект вольтметра, в котором напряжение внешнего источника напряжения измеряется и отображается на ЖК-дисплее в милливольтах.

Перед разработкой программы стоит ознакомиться с аналого-цифровым преобразователем (АЦП) Raspberry Pi Pico W.

АЦП внутри Pico

Большинство датчиков в реальной жизни являются аналоговыми и выдают аналоговые выходные напряжения или токи, которые пропорциональны измеряемой переменной. Такие датчики нельзя напрямую подключать к цифровым компьютерам без использования АЦП. В этой главе вы узнаете, как использовать каналы АЦП Raspberry Pi Pico.

Большинство АЦП для приложений общего назначения имеют разрядность 8 или 10 бит, хотя некоторые профессиональные АЦП более высокого класса имеют разрядность 16 или даже 32 бита. Время преобразования АЦП является одной из его важных характеристик. Это время, необходимое АЦП для преобразования аналогового входа в цифровой. Чем меньше время преобразования, тем лучше. Некоторые более дешевые АЦП выдают преобразованные цифровые данные в последовательном формате, в то время как более дорогие профессиональные обеспечивают параллельный цифровой выход.

Raspberry Pi Pico W имеет 5 каналов АЦП, 4 из которых находятся на контактах GP26, GP27, GP28 и GP29, известных как аналоговые каналы 0, 1, 2 и 3. Первые три канала доступны на контактах GPIO, а четвертый можно использовать для измерения напряжения VSYS платы. Имеется также встроенный канал АЦП 4, который внутренне соединен с датчиком температуры.

АЦП Pico имеет разрешение 12 бит, что позволяет преобразовывать аналоговое входное напряжение в 4096 (от 0 до 4095) уровней. Однако MicroPython преобразует вывод в 16-битное число в диапазоне от 0 до 65535.

Опорное напряжение АЦП, используемого в Pico, составляет +3,3 В. При использовании такого АЦП разрешение составляет 3300 мВ / 65535 = 0,050 мВ на бит или 50 мкВ/бит. Следовательно, аналоговое входное напряжение 0,050 мВ дает на цифровом выходе 00000000 0000001, 0,1 мВ дает 00000000 00000010 и так далее.

Цель: Цель этого проекта — показать, как можно использовать каналы Pico ADC для считывания аналогового входного напряжения.

Принципиальная схема: На рис. 5.11 показана принципиальная схема проекта. В этом проекте измеряемое напряжение подается на аналоговый вход GP26 (вывод 31, канал 0). Необходимо следить за тем, чтобы входное напряжение не превышало +3,3 В. Если требуется измерение более высоких напряжений, то можно использовать схемы резистивного делителя напряжения на входе АЦП.

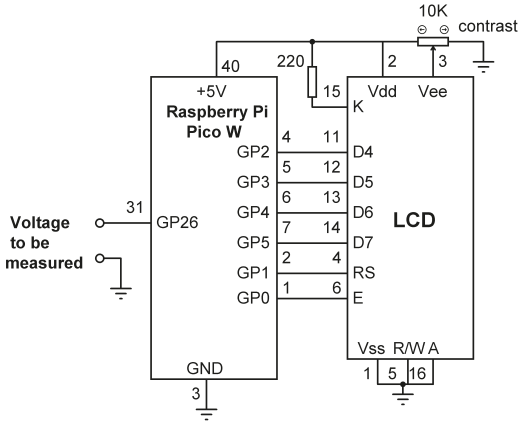


Рис. 5.11: Принципиальная схема проекта.

Листинг программы: На рис. 5.12 показан листинг программы и пример вывода программы (Программа:**VoltLCD**). В начале программного модуля АЦП импортируется в программу, а переменная **AnalogIn** назначается аналоговому входному каналу 0. Затем коэффициент преобразования определяется как 3300 / 65535. Значение, считанное с аналогового канала, должно быть умножено на это число для расчета фактического значения измеренного напряжения. Оставшаяся часть программы выполняется в цикле, в котором входное напряжение считывается, преобразуется в милливольты и затем отображается на ЖК-дисплее.

```
#-------------------------------------------------------
#               ВОЛЬТМЕТР
#               ==========
#
# Это проект вольтметра. Измеряемое напряжение
# подается на GP26 (пин 31) Raspberry Pi Pico W.
# Программа отображает измеренное напряжение на ЖК-дисплее
#
# Автор: Доган Ибрагим
# Файл   : VoltLCD.py
```

```
# Дата: октябрь 2022
г.#-----------------------------------------------------------
--from machine import ADC
import utime
import LCDFuncs

AnalogIn = ADC(0)                       # Канал АЦП 0
Conv = 3300 / 65535                     # Коэффициент преобразования
LCDFuncs.lcd_init()

while True:                             # Делать всегда
    mV = AnalogIn.read_u16()            # Чтение ввода
    mV = mV * Conv                      # Ввод в мВ
    LCDFuncs.lcd_clear()                # Очистить экран
    mVstr = str(mV)                     # Преобразование в строку
    LCDFuncs.lcd_puts(mVstr)            # Отображать
    utime.sleep(1)                      # ждать 1 секунду
```

Рис.*5.12:* Программа: **VoltLCD**.

5.8 Проект 4: Измерение температуры – с использованием внутреннего датчика температуры – с параллельным ЖК-дисплеем

Описание: В этом проекте используется внутренний датчик температуры Pico, а измеренная температура отображается на ЖК-дисплее.

Цель: Цель этого проекта — показать, как внутренний датчик температуры Pico можно использовать для измерения температуры.

Листинг программы: датчик внутренней температуры подключен к каналу 4 АЦП. Считываемые данные преобразуются в градусы Цельсия по следующей формуле:

Температура (в градусах Цельсия) = 27 - ((чтение - 0.706)/0.001721

Листинг программы показан на рисунке 5.13 (Program:**TempInt**). Программа считывает данные с канала 4 АЦП, преобразует их в вольты, а затем применяет приведенную выше формулу для преобразования их в градусы Цельсия. Рассчитанное значение затем отображается на ЖК-дисплее. Программа повторяется каждую секунду. Вы можете убедиться, что температура увеличится, если коснетесь процессора пальцем.

```
#-----------------------------------------------------------
#               ИЗМЕРЕНИЕ ТЕМПЕРАТУРЫ
#               ======================
#
# Эта программа измеряет температуру, используя внутренний
# датчик температуры пико и отображает на ЖК-дисплее
#
```

```
# Автор: Доган Ибрагим
# Файл  : TempInt.py
# Дата: октябрь 2022 г.
#-----------------------------------------------------------
from machine import ADC
import utime
import LCDFuncs

AnalogIn = ADC(4)                                  # канал АЦП 4
Conv = 3.3 / 65535                                 # Коэффициент преобразования
LCDFuncs.lcd_init()

while True:                                        # Делать навсегда
    V = AnalogIn.read_u16()                        # Чтение температуры
    V = V * Conv                                   # Преобразовать в вольты
    Temp = 27 - (V - 0.706) / 0.001721             # Преобразовать в температуру
    LCDFuncs.lcd_clear()                           # Очистить экран
    Tempstr = str(Temp)[:5]                        # Преобразование в строку
    LCDFuncs.lcd_puts(Tempstr)                     # Отображение
    utime.sleep(1)                                 # ждать 1 секунду
```

Рис.*5.13:* Программа:**TempInt**.

5.9 Проект 5: Измерение температуры – с помощью внешнего датчика температуры.

Описание: В этом проекте внешний аналоговый датчик температуры используется для измерения и отображения температуры окружающей среды на ЖК-дисплее.

Цель: Цель этого проекта — показать, как можно использовать внешний аналоговый датчик температуры для измерения внешней температуры.

Блок-схема: На рис. 5.14 показана блок-схема проекта.

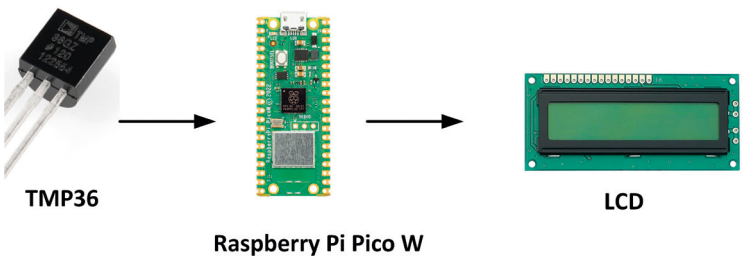


Рис.5.14: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис.5.15. В этом проекте используется микросхема датчика температуры типа TMP36 (рис. 5.16), подключенная к каналу 0 АЦП. Эта микросхема обеспечивает аналоговое выходное напряжение, пропорциональное измеренной температуре. Связь между измеренной температурой и выходным напряжением определяется выражением:

T = (Vo – 500) / 10

Где **T** – измеренная температура в градусах Цельсия, а **Vo** – выходное напряжение датчика в милливольтах.

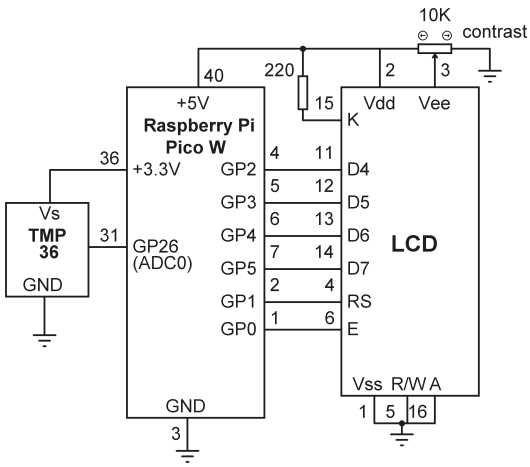


Рис.5.15: Принципиальная схема проекта.

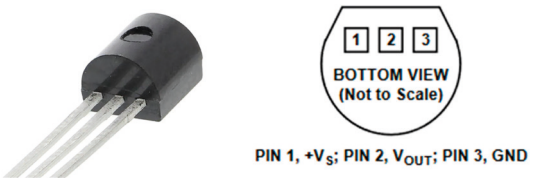


Рис.5.16: Датчик температуры TMP36.

Листинг программы: Рис.5.17 показывает листинг программы (Программа: **TMP36**). Напряжение датчика считывается по каналу 0 АЦП. Затем это напряжение преобразуется в милливольты. Температура рассчитывается в градусах Цельсия и отображается на ЖК-дисплее каждую секунду. Обратите внимание, что выходные данные отформатированы таким образом, что отображаются только первые пять цифр строки Tempare. Например, температура отображается в следующем формате:

nn.mm

```
#-------------------------------------------------------
#                      ИЗМЕРЕНИЕ ТЕМПЕРАТУРЫ
#                      =====================
#
# Эта программа измеряет температуру с помощью внешнего
# чипа датчика температуры типа TMP36
#
# Автор: Доган Ибрагим
# Файл   : TMP36.py
# Дата: октябрь 2022 г
#-------------------------------------------------------
from machine import ADC
import utime
import LCDFuncs

AnalogIn = ADC(0)                       # Канал АЦП 0
Conv = 3300 / 65535                     # Коэффициент преобразования
LCDFuncs.lcd_init()

while True:                             #  Делать всегда
    V = AnalogIn.read_u16()             # Чтение температуры
    mV = V * Conv                       # Преобразовать в вольты
    Temp = (mV - 500.0) / 10.0          # Преобразовать в температуру
    LCDFuncs.lcd_clear()                # Очистить экран
    Tempstr = str(Temp)[:5]             # Преобразование в строку
    LCDFuncs.lcd_puts(Tempstr)          # Отображение
    utime.sleep(1)                      # Ждать 1 секунду
```

Рис.*5.17:* Программа:**TMP36** .

5.10 Проект 6: Терморегулятор ВКЛ/ВЫКЛ с параллельным ЖК-дисплеем

Описание: Контроль температуры важен во многих промышленных, коммерческих и бытовых применениях. Система управления температурой в основном состоит из датчика температуры, нагревателя, вентилятора (дополнительно), исполнительного механизма для управления нагревателем и контроллера. Отрицательная обратная связь используется для управления нагревателем, чтобы температура соответствовала желаемому заданному значению. Точные системы контроля температуры основаны на алгоритме ПИД (пропорционально-интегрально-дифференциальном), который требует некоторых знаний о поведении и передаточной функции системы, которой необходимо управлять. В этом проекте разработана простая система управления типа ВКЛ/ВЫКЛ. В системах контроля температуры ВКЛ/ВЫКЛ обычно используются реле для включения или выключения нагревателя в зависимости от заданной температуры и измеренной температуры. Если измеренная температура ниже заданного значения, то срабатывает реле, которое включает нагреватель. Если, с другой стороны, измеренная температура выше заданного значения, то реле деактивируется, чтобы ВЫКЛЮЧИТЬ нагреватель, чтобы температура понизилась.

В этом проекте датчик температуры типа TMP36 используется вместе с нагревателем и светодиодом для управления температурой в небольшой комнате. Нагреватель включается реле, если измеренная температура в помещении (**RoomTemp)** ниже заданной температуры (**SetTemp**), и выключается, если выше заданного значения. Светодиод загорается, если температура в помещении ниже заданного значения, указывая на то, что обогреватель включен. Этот процесс повторяется каждые 3 секунды.

Цель: цель этого проекта — показать, как можно спроектировать систему контроллера температуры ВКЛ/ВЫКЛ с использованием недорогого чипа датчика температуры с Raspberry Pi Pico W.

Блок-схема: На рис.5.18 показана блок-схема проекта.

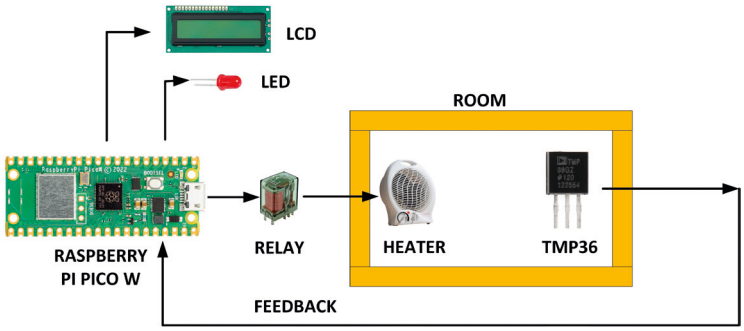


Рис.5.18: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис.5.19. Микросхема датчика TMP36 подключена к аналоговому каналу 0, как и в предыдущем проекте. Светодиод подключен к GP16 через токоограничивающий резистор 470 Ом. Реле подключено к GP17 и активируется при подаче на него логической 1 (+3,3 В). LCD подключается как в предыдущем проекте. Соединения между портами Raspberry Pi Pico W и различными компонентами следующие:

Raspberry Pi Pico W	**Компонент**
GP26 (ADC0)	TMP36 out
GP16	LED
GP17	Relay
GP0-GP5	LCD

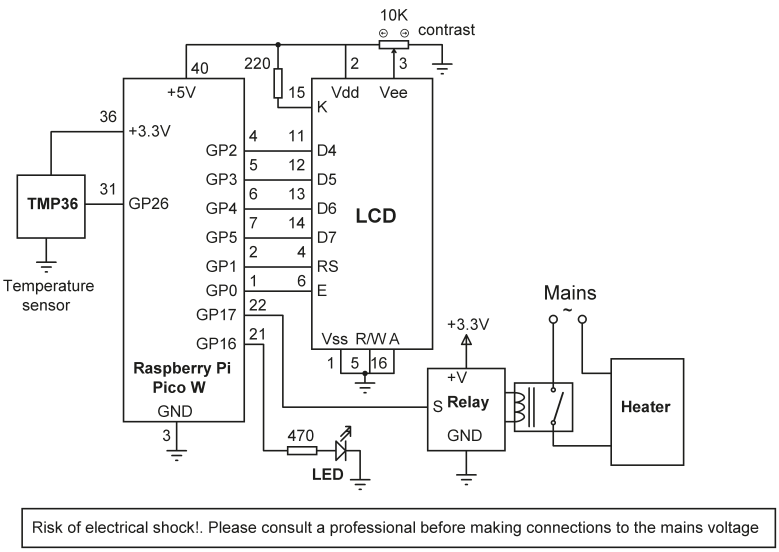


Рис.*5.19:* Принципиальная схема проекта.

Эксплуатация проекта

На ЖК-дисплее отображается желаемая температура (**SetTemp**) и измеренная комнатная температура (**RoomTemp**). Требуемая температура и комнатная температура отображаются в следующем формате, где комнатная температура постоянно обновляется каждые 3 секунды:

Ряд 0: **Set : 23.45**
Ряд 1: **Meas: 22.50**

Листинг программы: На рис. 5.20 показан листинг программы (Программа: **ONOFFLCD**). В начале программы используемые модули импортируются в программу, **SetTemp** устанавливается на 24 ºC, а реле и светодиоды настраиваются как выходы. В основном цикле программы комнатная температура **RoomTemp** считывается и сравнивается с **SetTemp**. Если **SetTemp** больше, чем **RoomTemp**, то и реле и светодиод включены, в противном случае они оба выключаются. Этот цикл повторяется каждые 3 секунды.

```
#------------------------------------------------------------
#              РЕГУЛЯТОР ТЕМПЕРАТУРЫ
#              ======================
#
# Это программа контроллера температуры ВКЛ-ВЫКЛ.
# Проект состоит из датчика температуры, светодиода и обогревателя.
# Нагреватель и светодиод включаются, если температура в помещении
# (RoomTemp) ниже желаемого значения (SetTemp)
#
# ЖК-дисплей используется для отображения желаемой температуры в верхней
# строке и комнатной температуры в нижней строке
```

```
#  Автор: Доган Ибрагим
# Файл  : ONOFFLCD.py
# Дата: октябрь2022 г.
#-------------------------------------------------------------
from machine import ADC, Pin
import utime
import LCDFuncs


LCDFuncs.lcd_init()                     # Инициализировать ЖК-дисплей
AnalogIn = ADC(0)                       # Канал АЦП 0
Conv = 3300 / 65535                     # Коэффициент преобразования

SetTemp = 24.0                          # Желаемая температура
LED = Pin(16, Pin.OUT)                  # LED на GP16
Relay = Pin(17, Pin.OUT)                # Реле на GP17
LED.value(0)                            # Выключить LED
Relay.value(0)                          # Выключение реле

LCDFuncs.lcd_clear()                    # Очистить экран
LCDFuncs.lcd_puts("Set : ")             # Отобразить Set :
LCDFuncs.lcd_puts(str(SetTemp)[:5])  # Отобразить SetTemp

while True:                             # Делать всегда
    V = AnalogIn.read_u16()             # Чтение temp
    mV = V * Conv                       # Преобразовать в вольты
    RoomTemp = (mV - 500.0) / 10.0   # Измеренная температура
    LCDFuncs.lcd_goto(0, 1)             # Курсор на 0,1
    LCDFuncs.lcd_puts("Meas: ")         # Отображение измерений:
    LCDFuncs.lcd_puts(str(RoomTemp)[:5])  # Отображение RoomTemp
    if RoomTemp < SetTemp:                # Если Room temp < требуемой
        Relay.value(1)                    # Включить реле
        LED.value(1)                      # Включить LED
    else:
        Relay.value(0)                    # Выключить реле
        LED.value(0)                      # Выключить LED
    utime.sleep(3)                        # ждите 3 секунды
```

Рис.*5.20:* Программа:**ONOFFLCD**.

5.11 Проект 7: Измерение интенсивности окружающего освещения – с использованием параллельного ЖК

Описание: В этом проекте фоторезистор (LDR) используется для измерения и отображения интесивности окружающего света на параллельном ЖК-дисплее.

Блок-схема: На рис.5.21 показана блок-схема проекта.

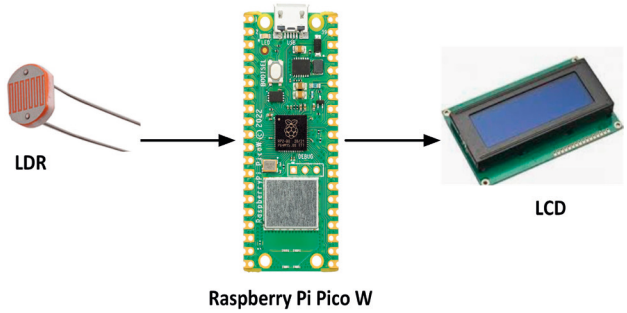


Рис.5.21: Блок-схема проекта.

Справочная информация: LDR — это фоторезистор, сопротивление которого изменяется в зависимости от интенсивности падающего на него света. Сопротивление LDR падает с увеличением интенсивности света, падающего на устройство. Как правило, сопротивление при дневном свете может составлять около килоом, а в темноте — несколько мегаом. В результате вы можете использовать LDR для измерения интенсивности света. Типичная характеристика LDR показана на рисунке 5.22. LDR используются в схемах в виде резистивных делителей напряжения. Последовательно с LDR подключается постоянный резистор и измеряется напряжение на этом резисторе. Это напряжение пропорционально сопротивлению LDR и, следовательно, интенсивности света, падающего на LDR.

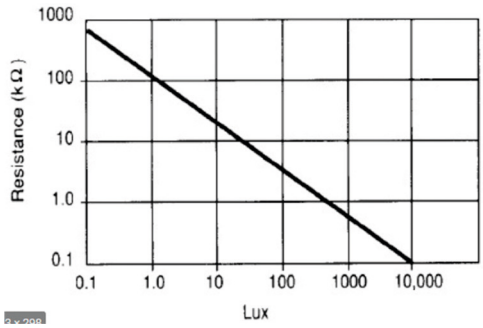


Рис.5.22: Типичная характеристика LDR.

Принципиальная схема: Принципиальная схема проекта показана на рис.5.23. В схеме делителя напряжения используется постоянный резистор сопротивлением 10 кОм. Напряжение на этом резисторе измеряется с помощью канала 0 АЦП.

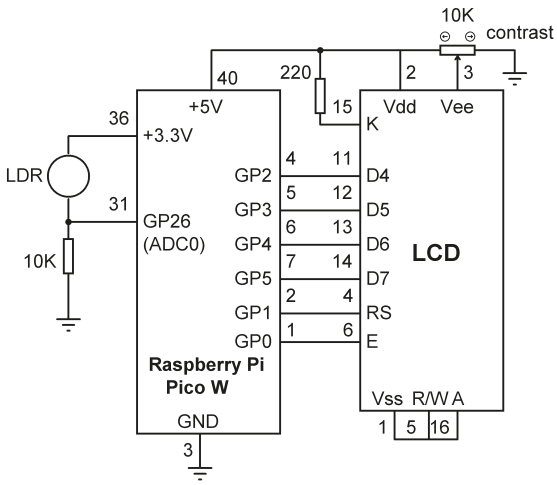


Рис.*5.23:* Принципиальная схема проекта.

Листинг программы: На рис.5.24 представлен листинг программы (Program:**LDR**). Программа считывает аналоговое напряжение на постоянном резисторе. Обратите внимание, что нет необходимости знать абсолютное напряжение. Вы можете просто использовать цифровое значение напряжения (от 0 до 65535), считанное этой программой. Это значение должно быть откалибровано, чтобы перевести интенсивность света в люксы.

```
#-------------------------------------------------------
#              ИЗМЕРЕНИЕ ИНТЕНСИВНОСТИ СВЕТА
#              ================================
#
# В этом проекте LDR подключен к Pico последовательно
# с постоянным резистором. Программа отображает
# напряжение на постоянном резисторе, пропорциональное
# уровню освещенности, падающему на фоторезистор.
#
# Автор: Доган Ибрагим
# Файл   : LDR.py
# Дата: октябрь 2022 г
#-------------------------------------------------------
from machine import ADC
import utime
import LCDFuncs

LDRin = ADC(0)                          # Канал АЦП 0
LCDFuncs.lcd_init()                     # Инициализировать ЖК-дисплей

while True:                             # Делать всегда
    r = LDRin.read_u16()                # Чтение LDR
    Tempstr = str(r)[:5]                # Преобразование в строку
    LCDFuncs.lcd_puts(Tempstr)          # Отображение
```

```
utime.sleep(1)                    # ждать 1 сек
LCDFuncs.lcd_clear()              # Очистка экрана
```

Рис.*5.24:* Программа: **LDR** .

Калибровка
Для калибровки показаний требуется люксметр. Измерения должны быть выполнены при различных уровнях освещенности, и должна быть создана таблица для перечисления показаний прибора в люксах и соответствующих выходных данных АЦП. Затем можно вывести формулу, описывающую взаимосвязь между уровнем освещенности и показаниями АЦП. В качестве альтернативы эта таблица может быть проиндексирована для данного показания АЦП, чтобы найти соответствующий уровень освещенности. Интерполяция может быть выполнена для значений между двумя показаниями.

5.12 Проект 8: Омметр – с параллельным ЖК-дисплеем

Описание: Это проект омметра. Проект измеряет значение неизвестного резистора и отображает его на экране Тонни.

Принципиальная схема: Принципиальная схема проекта показана на рис.5.25. Постоянный резистор (10 кОм) включен последовательно с неизвестным резистором. Программа измеряет напряжение на постоянном резисторе, а затем вычисляет значение неизвестного резистора (Rx).

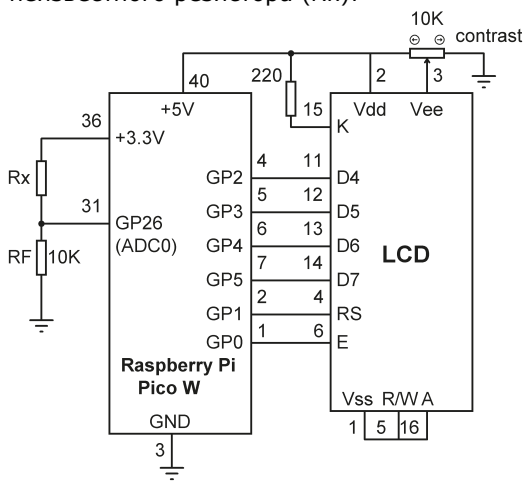


Рис.5.25: Принципиальная схема проекта.

Листинг программы: На рис.5.26 показан листинг программы (Программа: **Ohmmeter**). Если RF — постоянное сопротивление, а Rx — неизвестное сопротивление, при условии, что схема питается от +3,3 В (3300 мВ), напряжение на выходе постоянного резистора будет:

$$V = 3300 \times RF / (RF + Rx)$$

Если вы выберете RF равным 10 кОм, то

$$V = 33000 / (10 + Rx)$$

Где V в милливольтах, а RF и Rx в килоомах. При изменении Rx от,например, 1 кОм до 1 МОм, напряжение на постоянном резисторе изменится с:

$$V = 33000 / 11 = 3000 \text{ mV} \rightarrow V = 33000 / 1001 = 33 \text{ mV}$$

Вы можете легко измерить эти напряжения с помощью АЦП. Следовательно, диапазон измерения сопротивления вашего омметра значительно ниже 1 кОм и выше 1 МОм.

Что вам действительно нужно, так это измерить сопротивление Rx. Вы можете написать:

$$Rx = 3300 \times RF/V - RF$$

Помня, что разрешение АЦП составляет 65535 шагов и RF = 10 кОм, можно записать:

$$Rx = 655350 / Vm - RF$$

Где Vm — цифровое значение, считываемое непосредственно с АЦП.

Напряжение на постоянном резисторе считывается 5 раз и значение усредняется для большей точности.

```
#---------------------------------------------------------------
#                                 ОММЕТР
#                                 =======
#
# В этом проекте измеряется номинал неизвестного резистора
#
# Автор: Доган Ибрагим
# Файл   : Ohmmeter.py
# Дата: октябрь 2022 г
#---------------------------------------------------------------
from machine import ADC
import utime
import LCDFuncs

RF = 10                                 # RF = 10K
LDRin = ADC(0)                          # Канал АЦП 0
LCDFuncs.lcd_init()                     # Инициализировать ЖК-дисплей

while True:                             # Делать всегда
    sum = 0
    for i in range(5):                  # Получить 5 показаний
```

```
        sum = sum + LDRin.read_u16()   # Read voltage
    Vm = sum / 5                       # Average
    Rx = 65535*RF / Vm - RF            # Calculate Rx
    RxOhms = 1000 * Rx                 # Rx in Ohms
    Tempstr = str(RxOhms)              # Convert to string
    LCDFuncs.lcd_puts(Tempstr)         # Display
    utime.sleep(1)                     # Wait 1 second
    LCDFuncs.lcd_clear()               # Clear screen
```

Рис.5.26: Программа: Ohmmeter.

5.13 I²C LCD

Преимущество использования ЖК-дисплея на основе I2C заключается в том, что для управления ЖК-дисплеем требуется всего 2 контакта GPIO. LCD-дисплеи иногда поставляются в виде двух частей: LCD-дисплея и платы контроллера I2C. В большинстве стандартных дистрибутивов плата контроллера припаивается к задней части ЖК-дисплея, как показано на рис. 5.27. ЖК-дисплей представляет собой параллельный ЖК-дисплей типа 1602. Плата контроллера состоит из микросхемы контроллера PCF8574 I2C (от Texas Instruments или NXP semiconductors), небольшого потенциометра для регулировки контрастности, контактов интерфейса ввода-вывода и перемычек выбора адреса.

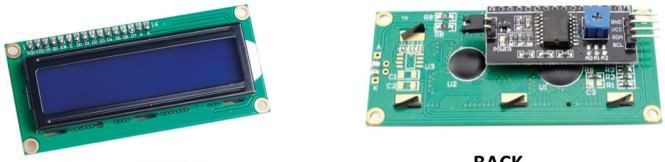


Рис.5.27: Стандартный ЖК-дисплей на базе I2C.

Плата контроллера имеет 4 контакта: SCL, SDA, VCC и GND, и они должны быть припаяны к контактам ввода-вывода макетной платы, как показано на рис. 5.28 (если он еще не припаян).

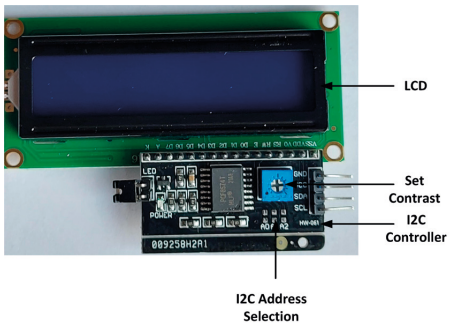


Рис.5.28: Припаяйте плату контроллера I2C к плате ЖК-дисплея.

Как показано на рис. 5.29, адрес I2C микросхемы PCF8574T выбирается тремя перемычками, обозначенными как A0, A1 и A2 на плате контроллера. По умолчанию адрес установлен на 0x27 (т. е. перемычек нет).

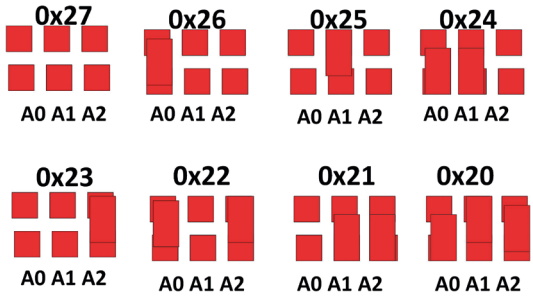


Рис.5.29: выбор адреса.

Прежде чем использовать ЖК-дисплей I2C, вы должны добавить библиотеку I2C в свой Raspberry Pi Pico W. Здесь используется библиотека микроконтроллеров (https://microcontrollerslab.com/i2c-lcd-rasp-berry-pi-pico-micropython- tutorial/), которая состоит из двух библиотечных файлов с именами lcd_api.py и i2c_lcd.py. Эти библиотечные файлы доступны из папки с кодами Program files. Скопируйте эти файлы на Raspberry Pi Pico с точными именами lcd_api.py и i2c_lcd.py. теперь вы можете подключить ЖК-дисплей I2C к Raspberry Pi Pico W и разрабатывать программы.

Аппаратный интерфейс

ЖК-дисплей I2C подключается к Raspberry Pi Pico через два контакта I2C SDA и SCL. ЖК-дисплеи I2C обычно работают с +5 В. Однако Pico не совместим с +5 В. Безопаснее всего использовать двунаправленный модуль преобразователя логических уровней между дисплеем и Pico. В Интернете доступно множество модулей логических преобразователей. Одним из таких модулей преобразователя является TXS0102 (рис. 5.30). Это двухпортовый модуль преобразователя логического уровня.

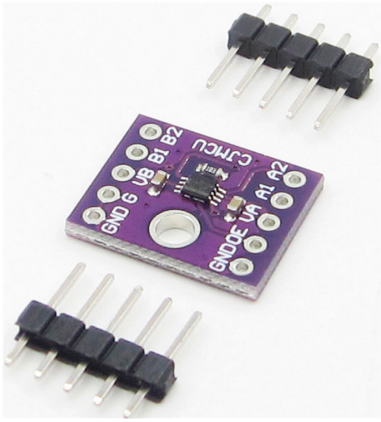


Рис.5.30: Модуль преобразователя логического уровня TXS0102.

На рис. 5.31 показано базовое подключение ЖК-дисплея I2C к Raspberry Pi Pico W через модуль логического преобразователя TXS0102.

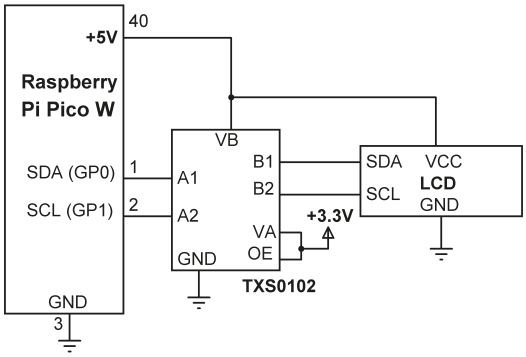


Рис.5.31: Подключение ЖК-дисплея к Pico.

Поиск адреса I2C ЖК-дисплея
Следующая программа отобразит на экране I2C-адреса всех устройств, подключенных к шине I2C:

```
import machine
sda=machine.Pin(0)
scl=machine.Pin(1)
i2c=machine.I2C(0,sda=sda, scl=scl, freq=400000)

devices = i2c.scan()
if len(devices) == 0:
     print("No I2C device found")
else:
     print('I2C devices found:', len(devices))
     for device in devices:
         print("I2C address: ",hex(device))
```

Пример запуска вышеуказанной программы дал автору следующее отображение:

```
I2C devices found: 1
I2C address:       0x27
```

Команда библиотеки I2C LCD:
Библиотека I2C поддерживает следующие команды:

```
clear()
putstr()
backlight_on()
backlight_off()
show_cursor()
```

```
hide_cursor()
blink_cursor_on()
blink_cursor_off()
display_on()
display_off()
move_to(column, row) – left hand corner is (0, 0)
putchar()
custom_char()
```

5.14 Проект 9: ЖК-счетчик секунд I2C

Описание: Это простой проект счетчика секунд с использованием ЖК-дисплея I2C.

Цель: **Цель этого проекта —** показать, как можно разработать простой счетчик секунд с использованием ЖК-дисплея I2C.

Блок-схема: На рис.5.32 показана блок-схема.

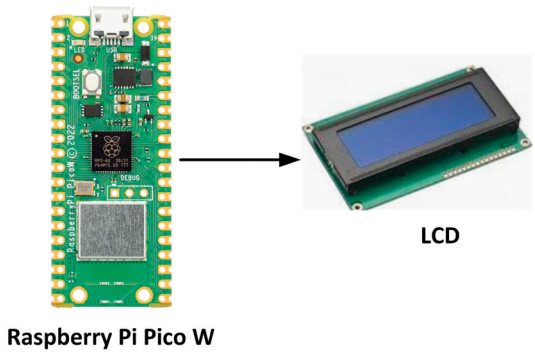


Рис.5.32: Блок-схема проекта.

Принципиальная схема: Принципиальная схема такая же, как на рис.5.31.

Листинг программы: На рис.5.33 представлен листинг программы (Program:**I2CSecons**). В начале программы в программу импортируются ЖК-модули и временной модуль, определяются I2C-адрес ЖК-дисплея, количество строк и количество столбцов. Внутри основной программы переменная count увеличивается каждую секунду и отображается на ЖК-дисплее.

```
#-------------------------------------------------------------
#               I2C ЖК-СЧЕТЧИК СЕКУНД
#               ======================
#
# Это проект счетчика секунд с использованием ЖК-дисплея I2C.
# Количество секунд отображается на ЖК-дисплее в формате:
#     Count=nn
#
# Автор: Доган Ибрагим
```

```
# Файл  : I2CSeconds.py
# Дата: октябрь 2022 г
#-------------------------------------------------------------
import machine
from machine import I2C
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
import utime

I2C_ADDR = 0x27                    # ЖК-адрес I2C
NRows = 2                          # Количество строк
NColumns = 16                      # Количество столбцов

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, NRows, NColumns)


count = 0
while True:
    lcd.clear()
    count = count + 1
    lcd.putstr("Count=")
    lcd.putstr(str(count))
    utime.sleep(1)
```

Рис.*5.33:* Программа:**I2CSeconds**.

5.15 Проект 10: Внутренняя и внешняя температура – с использованием ЖК-дисплея

Описание: В этом проекте используются две микросхемы датчика температуры: одна для измерения внешней температуры окружающей среды, а другая для измерения внутренней температуры окружающей среды. Оба показания отображаются на ЖК-дисплее каждые 2 секунды.

Блок-схема: На рис. 5.34 показана блок-схема проекта.

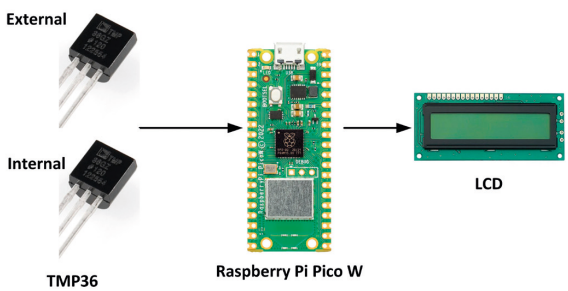


Рис.5.34: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис.5.35. Используются две микросхемы датчиков температуры типа TMP36, одна из которых подключена к каналу 0 (внешний датчик), а другая — к каналу 1 (внутренний датчик) АЦП. ЖК-дисплей подключен к Pico, как и в предыдущих проектах на основе ЖК-дисплея.

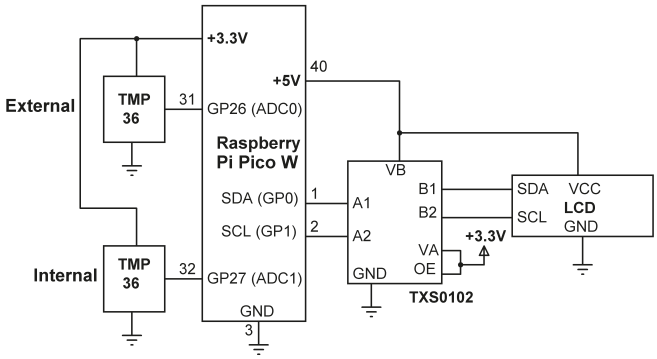


Рис.5.35: Принципиальная схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис.5.35. Используются две микросхемы датчиков температуры типа TMP36, одна из которых подключена к каналу 0 (внешний датчик), а другая — к каналу 1 (внутренний датчик) АЦП. ЖК-дисплей подключен к Pico, как и в предыдущих проектах на основе ЖК-дисплея.

Ряд 0: Ext: nn.mm
Ряд 1: Int: pp.qq

```
#-------------------------------------------------------------
#      ИЗМЕРЕНИЕ ВНЕШНЕЙ И ВНУТРЕННЕЙ ТЕМПЕРАТУРЫ
#      ============================================
#
# Эта программа измеряет внешнюю и внутреннюю температуру
# с использованием двух микросхем датчика температуры типа TMP36. Оба внешние
# и внутренняя температура отображаются на ЖК-дисплее
#
# Автор: Доган Ибрагим
# Файл  : MultiTmp.py
# Дата: октябрь 2022 г
#-------------------------------------------------------------
import machine
from machine import ADC
import utime
from machine import I2C
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
```

```
I2C_ADDR = 0x27                                    # ЖК-адрес I2C
NRows = 2                                          # Количество строк
NColumns = 16                                      # Количество столбцов

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, NRows, NColumns)

ExtTemp = ADC(0)                                   # Канал АЦП 0
IntTemp = ADC(1)                                   # Канал АЦП 1
Conv = 3300 / 65535                                # Коэффициент преобразования

while True:                                        # Делать всегда
    Vext = ExtTemp.read_u16()                      # Чтение канала 0
    mV = Vext * Conv                               # Преобразовать в мВ
    Tempext = (mV - 500.0) / 10.0                  # Внешняя температура
    Vint = IntTemp.read_u16()                      # Чтение канала 1
    mV = Vint * Conv                               # Преобразовать в мВ
    Tempint = (mV - 500.0) / 10.0                  # Внутренняя температура
    lcd.clear()                                    # Очистить экран
    Tempextstr = str(Tempext)[:5]                  # Преобразование в строку
    Tempintstr = str(Tempint)[:5]                  # Преобразование в строку
    lcd.putstr("Ext: ")                            # Заголовок
    lcd.putstr(Tempextstr)                         # Отображение внешней темп.
    lcd.move_to(0, 1)                              # Курсор в строке 1
    lcd.putstr("Int: ")                            # Заголовок
    lcd.putstr(Tempintstr)                         # Отображение внутрен. темп.
    utime.sleep(2)                                 # ждем 2 секунды
```

Рис. *5.36:* Программа: **MultiTmp**.

5.16 Проект 11: Использование термистора для измерения температуры – использование ЖК-дисплей I2C

Описание: В этом проекте будет считываться температура окружающей среды каждую секунду, используя модуль датчика температуры KY-013 (термистор NTC), а затем отображение ее на ЖК-дисплее.

Цель: Цель этого проекта — показать, как можно считывать температуру термисторного датчика температуры NTC, а также как использовать ЖК-дисплей в проекте.

Справочная информация: В этом проекте используется модуль датчика температуры с аналоговым выходом NTC типа KY-013. Термисторы NTC представляют собой полупроводниковые устройства, сопротивление которых обратно пропорционально температуре. Таким образом, при повышении температуры их сопротивление падает и наоборот. Как показано на рис. 5.37, это 3-контактный модуль со следующими контактами:

GND
+V
S (analog output)

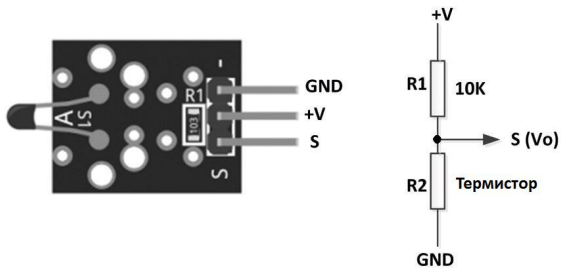


Рис.5.37: Модуль KY-013.

Модуль датчика внутренне подключен к резистору 10 кОм, как показано на рисунке, для создания делителя напряжения. Напряжение на термисторе считывается с помощью аналогового порта процессора. Это напряжение пропорционально температуре, где температура рассчитывается с использованием известного уравнения Штейнхарта-Харта. Различные термисторы имеют разные коэффициенты Штейнхарта-Харта, и необходимо знать эти коэффициенты для расчета температуры используемого термистора. Для KY-013 эти коэффициенты указаны производителем следующим образом (ваши коэффициенты могут немного отличаться!):

c1 = 0.001129148
c2 = 0.000234125
c3 = 0. 0000000876741

Предполагая, что резистор подключен последовательно с термистором, R1 = 10 кОм (т.е. R1 = 10000 Ом), и используется 16-битный АЦП (от 0 до 65535 уровней квантования) для считывания напряжения термистора, температура рассчитывается следующим образом (см. 5.37):

Во-первых, рассчитаем сопротивление термистора. Из схемы делителя потенциала выходное напряжение Vo определяется как:

$$Vo = V \times R2 / (R1 + R2) \quad (1)$$

Где V — приложенное напряжение. Из этого уравнения R2 находится как:

$$R2 = Vo \times R1 / (V - Vo) \quad (2)$$

При использовании 16-разрядного АЦП, если **Raw** является необработанным значением, считываемым АЦП, тогда реальное считываемое физическое напряжение Vo определяется как:

$$Vo = Raw \times V / 65535 \quad (3)$$

Из (3) и (2) получаем:

$$R2 = [R1 \times Raw \times V / 65535] / (V - Raw \times V/65535] \quad (4)$$

Уравнение (4) упрощается и дает:

$$R2 = R1 / (65535/Raw - 1) \quad (5)$$

или,

$$R2 = 10000 / (65535 / Raw - 1) \quad (6)$$

Зная R2, температура определяется уравнением Стейнхарта-Харта:

$$T = \log(R2) \quad (7)$$

$$Tmp = 1 / (c1 + (c2 + (c3 \times T \times T)) \times T) \quad (8)$$

Затем переводится температура из Кельвина в градусы Цельсия:

$$Temp = Tmp - 273.15 \quad (9)$$

Здесь используются приведенные выше уравнения для расчета температуры. Обратите внимание, что выше расчеты только для резистора 10 кОм. Надо убедиться, что есть соответствующий резистор, установленный на модуле KY-013.

Примечание: показания очень чувствительны к резистору, а также к параметрам термистора c1, c2, c3.

Блок-схема: На рис.5.38 показана блок-схема проекта.

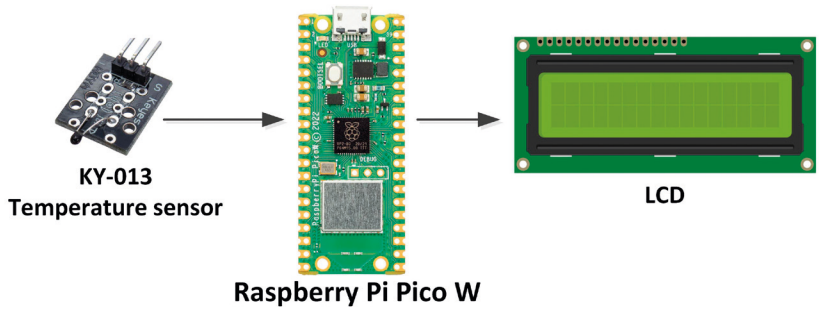


Рис.5.38: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рисунке 5.39. Датчик подключен к каналу 0 АЦП (GP26, контакт 31). LCD подключается как и в предыдущих проектах с помощью LCD.

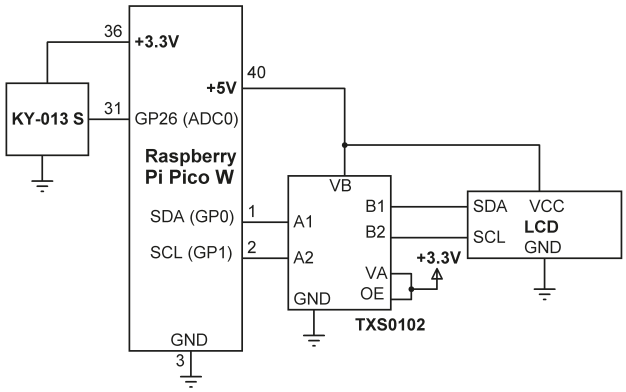


Рис.5.39: Принципиальная схема проекта.

Список программ: На рис.5.40 показан список программ (Программа: **Термистор**). В начале программы переменная **Thermistor** назначается каналу 0 АЦП и инициализируется LCD. Основная программа запускается в цикле каждые 2 секунды. Данные термистора считываются в переменную **Raw**, а затем вызывается функция **Temperature** для расчета температуры. ЖК-дисплей очищается, в верхней строке отображается заголовок **Temperature (C)**, а значение температуры форматируется и отображается в нижней строке ЖК-дисплея в градусах Цельсия в следующем формате:

```
Ряд 0:      Temperature (C)
Ряд 1:      nn.mm
```

```
#-------------------------------------------------------------
#              ИЗМЕРЕНИЕ ТЕМПЕРАТУРЫ ТЕРМИСТОРОМ
#              ====================================
#
# Эта программа измеряет температуру окружающей среды, используя
# недорогой термисторный модуль NTC (KY-013).
# Показания отображаются на ЖК-дисплее
#
# Автор: Доган Ибрагим
# Файл   : Thermistor.py
# Дата: октябрь 2022 г
#-------------------------------------------------------------
import machine
from machine import I2C
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
import utime
from machine import ADC
import math

I2C_ADDR = 0x27                    # ЖК-адрес I2C
```

```
NRows = 2                           # Количество строк
NColumns = 16                       # Количество столбцов

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, NRows, NColumns)

Thermistor = ADC(0)                 # канал АЦП 0

#
# Расчет температуры по уравнению Стейнхарта-Харта
#
def Temperature(RawValue):
    c1 = 0.001129148
    c2 = 0.000234125
    c3 = 0.0000000876741
    R1 = 10000.0
    ADC_Res = 65535.0

    R2 = R1 / ((ADC_Res/RawValue - 1))
    T = math.log(R2)
    Tmp = 1.0 / (c1 + (c2 + (c3 * T * T)) * T)
    Temp = Tmp - 273.15
    return Temp

while True:                                     # Делать всегда
    Raw = Thermistor.read_u16()                 # Чтение канала 0
    temp = Temperature(Raw)                     # Расчет температуры
    lcd.clear()                                 # Очистить экран
    lcd.putstr("Temperature (C)")               # Показать "Temperature (C)"
    lcd.move_to(0, 1)                           # Переместить курсор
    tempstr = str(temp)[:5]                     # Преобразование в строку
    lcd.putstr(tempstr)                         # Отображение температуры
    utime.sleep(2)                              # ждём 2 секунды
```

Рис.*5.40:* Программа:**Thermistor**.

Примечание. Вы также можете использовать модифицированную форму уравнения Стейнхарта-Харта (известную как уравнение параметра B), если параметры c1, c2, c3 неизвестны. Модифицированное уравнение:

1/T = 1/T0 + 1/B (Log(R/R0))

Где:

T — измеренная абсолютная температура (отнимите 273,15, чтобы найти температуру в °C).

T0 — абсолютная комнатная температура (равная 298,15 K) при 25 ºC.

B - термисторный коэффициент (обычно указывается производителями в районе 3960).

R — измеренное сопротивление термистора

R0 — сопротивление термистора при комнатной температуре (обычно указывается производителями в районе 10 кОм).

5.17 Проект 12: Ультразвуковое измерение расстояния – с использованием ЖК-дисплея I2C

Описание: В этом проекте модуль ультразвукового датчика используется для измерения расстояния до препятствия перед датчиком. Расстояние отображается на ЖК-дисплее.

Цель: Цель этого проекта — показать, как модуль ультразвукового датчика можно использовать в проекте для измерения расстояния до препятствия перед датчиком.

Ультразвуковой датчик

В этом проекте используется популярный ультразвуковой приемопередающий модуль типа HC-SR04 (рис. 5.41). Основные характеристики этого модуля:

- Рабочее напряжение: 5 V
- Рабочий ток: 2 mA
- Расстояние обнаружения: 2 cm – 450 cm
- Входной триггерный сигнал: 10-µs TTL
- Угол датчика: не более 15 градусов

Рис.5.41: Модуль ультразвукового датчика HC-SR04.

Распиновка HC-SR04:

Vcc: + питания
Trig: Вход триггера
Echo: Выход эха
Gnd: Земля (- питания)

Основной принцип работы модуля ультразвукового датчика HC-SR04 заключается в следующем (см. рис.5.42):

- На модуль отправляется триггерный импульс длительностью 10 мкс.
- Затем модуль отправляет на цель восемь сигналов прямоугольной формы с частотой 40 кГц и устанавливает на эхо-вывод ВЫСОКИЙ уровень.
- Программа запускает таймер.
- Сигнал достигает цели и эхом возвращается к модулю.
- Когда сигнал возвращается к модулю, вывод эха становится НИЗКИМ.
- Таймер остановливается.
- Рассчитывается длительность эхо-сигнала, пропорциональная расстоянию до цели.

Расстояние до объекта рассчитывается следующим образом:

Расстояние до объекта (в метрах) = (длительность времени эха в секундах × скорость звука) / 2

Скорость звука составляет 343 м/с, или 0,0343 см/мкс при температуре воздуха 20 °C.
Следовательно,

Расстояние до объекта (в см) = (продолжительность эхо-сигнала в мкс) × 0,0343/2

или,

Расстояние до объекта (в см) = (продолжительность эхо-сигнала в мкс) × 0,0171

Например, если длительность эхо-сигнала составляет 294 мкс, то расстояние до объекта рассчитывается следующим образом:

Расстояние до объекта (см) = 294×0,0171 = 5,03 см

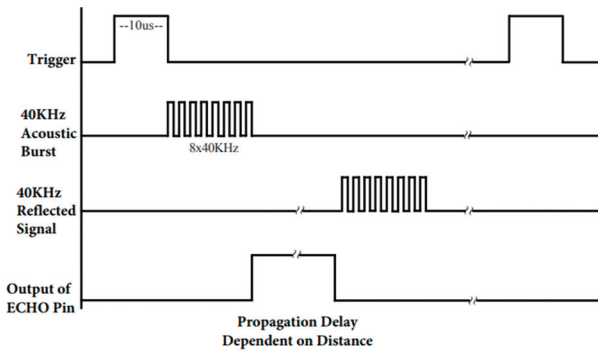


Рис.*5.42:* Работа модуля ультразвукового датчика.

Блок-схема: На рис. 5.43 показана блок-схема проекта.

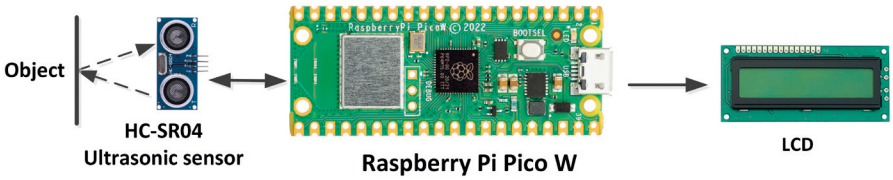


Рис.5.43: Блок-схема проекта.

Принципиальная схема: На рис. 5.44 показана принципиальная схема проекта. Обратите внимание, что датчик работает при +5 В, а его выход несовместим с входом Raspberry Pi Pico W. Для понижения напряжения датчика до +3,3 В используется делитель напряжения.

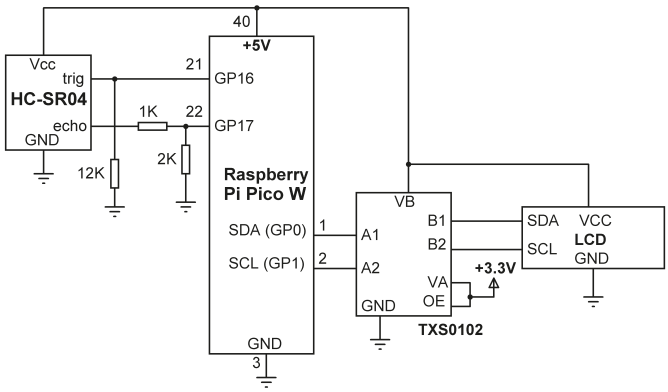


Рис.5.44: Принципиальная схема проекта.

Листинг программы: На рис. 5.45 показан листинг программы (Программа: **Ultrasonic**). В начале программы настраиваются пины триггера и эха. Внутри основного программного цикла импульс триггера отправляется в течение 5 микросекунд, и программа ожидает получения эхо-сигнала (обратите внимание, что требуемая ширина импульса составляет 10 микросекунд, но автор при измерении заметил, что использование 10 мкс создает задержку, намного превышающую 10 микросекунд). Значение между 2 и 5 должно давать правильные результаты). После получения эхо-сигнала его длительность вычисляется и сохраняется в переменной **Duration**. Предполагая, что скорость звука в воздухе равна 343 м/с (при температуре воздуха 20 ºC), расстояние до объекта вычисляется и сохраняется в переменной **Distancecm**. Затем это значение отображается на ЖК-дисплее, как показано на рис. 5.46.

```
#-------------------------------------------------------
#            УЛЬТРАЗВУКОВОЕ ИЗМЕРЕНИЕ РАССТОЯНИЯ
#            ====================================
#
# В этом проекте модуль ультразвукового датчика типа HC-SR04
# подключен к Raspberry Pi Pico. Программа отображает
# расстояние до объекта перед датчиком
#
```

```
# Автор: Доган Ибрагим
# Файл  : Ultrasonic.py
# Дата: октябрь 2022 г
#-------------------------------------------------------------
import machine
from machine import I2C
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
import utime
from machine import Pin

I2C_ADDR = 0x27                         # ЖК-адрес I2C
NRows = 2                               # Количество строк
NColumns = 16                           # Количество столбцов

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, NRows, NColumns)

trig = Pin(16, Pin.OUT)                 # Пин  триггера как выход
echo = Pin(17, Pin.IN)                  # Пин эха как вход

while True:
    trig.value(0)
    utime.sleep_us(5)                   # время на установку

    trig.value(1)                       # Отправить импульс запуска
    utime.sleep_us(5)                   # 5 микросекунд
    trig.value(0)                       # Удалить триггерный импульс

    while echo.value() == 0:            # Ожидание эха 1
        pass
    Tmrstrt = utime.ticks_us()

    while echo.value() == 1:            # Ожидание эха 0
        pass
    Tmrend = utime.ticks_us()

    Duration = utime.ticks_diff(Tmrend, Tmrstrt)
    distancecm = Duration * 0.0171
    lcd.clear()
    D = "Dist = " + str(distancecm)[:6] + " cm"
    lcd.putstr(D)
    utime.sleep(1)
```

Рис.*5.45:* Программа:**Ultrasonic**.

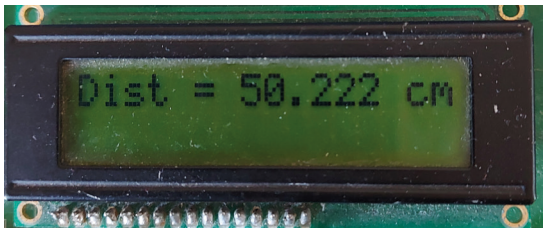


Рис.5.46: Пример отображения на ЖК-дисплее.

5.18 Проект 13: Измерение глубины реки

Описание: Глубина реки непостоянна и меняется в зависимости от количества выпадающих в реке осадков, времени года и приливов. Капитанам лодок или кораблей на реке необходимо знать глубину воды в реке, прежде чем они смогут безопасно плыть по реке. Кроме того, исследователям окружающей среды необходимо знать глубину реки для анализа состояния здоровья живых существ в реке. В этом проекте вы будете использовать ультразвуковой датчик типа HC-SR04 для измерения и отображения глубины воды в реке каждые 60 секунд.

Блок-схема: На рис. 5.47 показана блок-схема проекта. Ультразвуковой датчик устанавливается на известной высоте от вершины реки, глубину воды которой необходимо измерить. Датчик измеряет расстояние между датчиком и поверхностью воды. Предполагая, что расстояние между датчиком и дном реки равно H, а расстояние между датчиком и поверхностью воды равно h, глубина воды d в реке определяется как d = H − h.

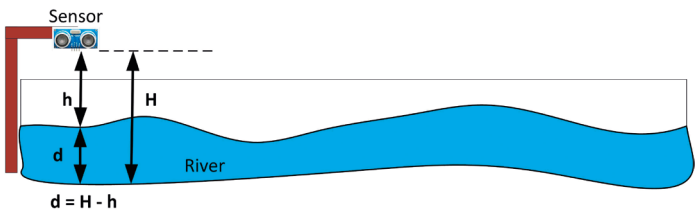


Рис.5.47: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис.5.44.

Листинг программы: На рис.5.48 показан листинг программы (Program:**river**). В этом проекте H принимается равным 300 см. программа похожа на предыдущую, но здесь глубина воды измеряется и отображается каждые 60 секунд.

```
#-------------------------------------------------------------
#            ГЛУБИНА ВОДЫ В РЕКЕ
#            ====================
#
# В этом проекте модуль ультразвукового датчика типа HC-SR04
# подключен к Raspberry Pi Pico W. Программа измеряет
# и отображает глубину реки
#
# Автор: Доган Ибрагим
# Файл  : river.py
# Дата: октябрь 2022 г
#-------------------------------------------------------------
import machine
from machine import I2C
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
import utime
from machine import Pin

I2C_ADDR = 0x27                          # ЖК-адрес I2C
NRows = 2                                # Количество строк
NColumns = 16                            # Количество столбцов

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, NRows, NColumns)

trig = Pin(16, Pin.OUT)                  # Пин триггера как выход
echo = Pin(17, Pin.IN)                   # Пин эха как вход
H = 300                                  # В сантиметрах

while True:
    trig.value(0)
    utime.sleep_us(5)                    # Подождать, пока не будет установлено

    trig.value(1)                        # Отправить импульс запуска
    utime.sleep_us(5)                    # Ждать 5 микросекунд
    trig.value(0)                        # Удалить триггерный импульс

    while echo.value() == 0:             # Ожидание эха 1
        pass
    Tmrstrt = utime.ticks_us()

    while echo.value() == 1:             # Ожидание эха 0
        pass
    Tmrend = utime.ticks_us()
```

```
Duration = utime.ticks_diff(Tmrend, Tmrstrt)
h = Duration * 0.0171
d = H - h
lcd.clear()
D = "d = " + str(d)[:7] + " cm"
lcd.putstr(D)
utime.sleep(1)
```

Рис.*5.48:* Программа: **river**.

5.19 Проект 14: Ультразвуковая система помощи при парковке задним ходом с зуммером

Описание: В этом проекте модуль ультразвукового датчика используется вместе с активным зуммером, чтобы помочь при парковке автомобиля задним ходом. По мере того, как расстояние до объектов уменьшается, зуммер звучит с большей частотой, чтобы предупредить водителя о приближении объектов к задней части автомобиля.

Цель: Цель этого проекта — показать, как модуль ультразвукового датчика можно использовать для парковки автомобиля задним ходом.

Блок-схема: На рис.5.49 показана блок-схема проекта.

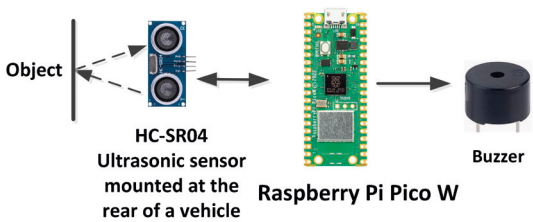


Рис.5.49: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис. 5.50. К GP18 Raspberry Pi Pico W добавлен активный зуммер, который издает звуковой сигнал, когда автомобиль приближается к препятствию. LCD не используется в этом проекте.

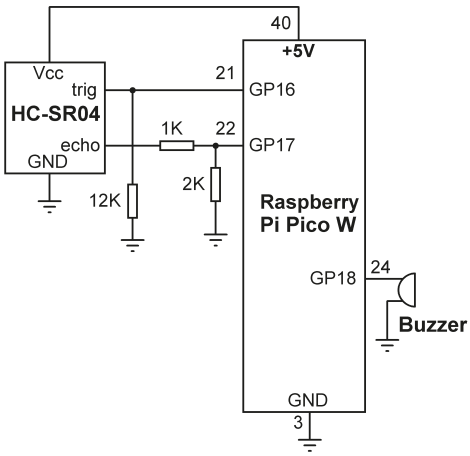


Рис.5.50: Принципиальная схема проекта.

Листинг программы: На рис. 5.51 показан листинг программы (Программа: **Parking**). После определения расстояния до препятствия программа создает значение задержки в зависимости от расстояния до препятствия. Когда транспортное средство приближается к препятствию, это значение задержки уменьшается, так что зуммер звучит чаще, предупреждая водителя о приближении транспортного средства к препятствию. Если, с другой стороны, транспортное средство удаляется от препятствия дальше, то задержка увеличивается, так что зуммер звучит реже, информируя водителя о том, что препятствие находится не очень близко.

```
#------------------------------------------------------------
#  УЛЬТРАЗВУКОВАЯ ПОМОЩЬ ПРИ ПАРКОВКЕ ЗАДНИМ ХОДОМ
#  ================================================
#
# В этом проекте модуль ультразвукового датчика типа HC-SR04
# подключен к Raspberry Pi Pico. Дополнительно подключен зуммер
# Устройство издает звуковой сигнал, когда автомобиль
# приближается к препятствию. Зуммер звучит быстрее, когда
# автомобиль приближается к объекту.
#
# Автор: Доган Ибрагим
# Файл   : Parking.py
# Дата: октябрь 2022 г
#------------------------------------------------------------
import machine
import utime
from machine import Pin

trig = Pin(16, Pin.OUT)                 # Пин триггер как выход
echo = Pin(17, Pin.IN)                  # Пин эха как вход

Buzzer = Pin(18, Pin.OUT)               # Зуммер как выход
```

```
Buzzer.value(0)                          # Выключить зуммер

while True:
    trig.value(0)
    utime.sleep_us(5)                    # Подождать, пока не будет установлено

    trig.value(1)                        # Отправить импульс запуска (триггера)
    utime.sleep_us(5)                    # Ждать 5 микросекунд
    trig.value(0)                        # Удалить триггерный импульс

    while echo.value() == 0:             # Ожидание эха 1
        pass
    Tmrstrt = utime.ticks_us()

    while echo.value() == 1:             # Ожидание эха 0
        pass
    Tmrend = utime.ticks_us()

    Duration = utime.ticks_diff(Tmrend, Tmrstrt)
    distance = Duration * 0.0171

#
# Звучание должно быть частым
# по мере того, как транспортное средство приближается к объекту.
# Это делается путем изменения задержки продолжительности звука.
#

    if distance > 100:
        dely = 0
    elif distance > 70 and distance < 90:
        dely = 600
    elif distance > 50 and distance < 70:
        dely = 400
    elif distance > 30 and distance < 50:
        dely = 300
    elif distance > 10 and distance < 30:
        dely = 200
    elif distance < 10:
        dely = 10

    if distance < 100:
        Buzzer.value(1)
        utime.sleep_ms(dely)
        Buzzer.value(0)
        utime.sleep_ms(dely)
```

Рис.*5.51:* Программа:**Parking**.

5.20 Проект 15: Отображение пользовательских символов на ЖК-дисплее

Описание: В этом проекте вы создадите фигуру со стрелкой вверх и отобразите ее на своем I2CLCD

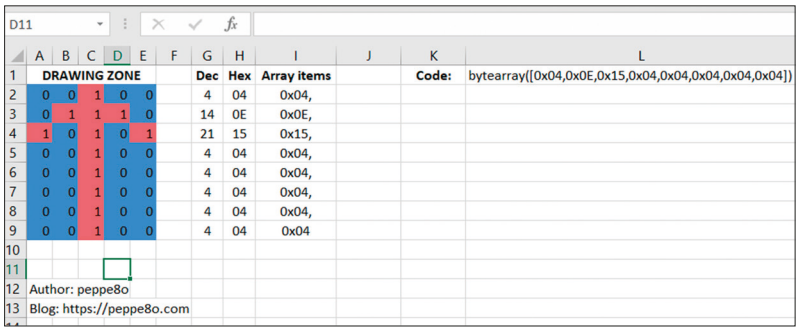
Цель: Цель этого проекта — показать, как можно создавать пользовательские символы и отображать их на ЖК-дисплее. Блок-схема и принципиальная схема проекта показаны на рис. 5.32 и рис. 5.31 соответственно.

Создание пользовательских символов

Пользовательский символ состоит из 5 столбцов и 8 строк, т. е. 40 бит. Существует множество способов создания пользовательского персонажа. Веб-сайт, указанный ниже, предоставляет лист Excel, который поможет создать собственный символ:

https://peppe8o.com/using-i2c-lcd-display-with-raspberry-pi-pico-and-micropython/

По сути, вы манипулировали 1 и 0 для создания нужного вам персонажа. На рис. 5.52 показана стрелка вверх, созданная с помощью листа Excel.



	A	B	C	D	E	F	G	H	I	J	K	L
1	DRAWING ZONE						Dec	Hex	Array items		Code:	bytearray([0x04,0x0E,0x15,0x04,0x04,0x04,0x04,0x04])
2	0	0	1	0	0		4	04	0x04,			
3	0	1	1	1	0		14	0E	0x0E,			
4	1	0	1	0	1		21	15	0x15,			
5	0	0	1	0	0		4	04	0x04,			
6	0	0	1	0	0		4	04	0x04,			
7	0	0	1	0	0		4	04	0x04,			
8	0	0	1	0	0		4	04	0x04,			
9	0	0	1	0	0		4	04	0x04			
10												
11												
12	Author: peppe8o											
13	Blog: https://peppe8o.com											

Рис.5.52: Создание стрелки вверх.

После заполнения необходимых нулей и единиц посмотрите на код справа. В этом примере код:

Код: байтовый массив ([0x04,0x0E, 0x15,0x04,0x04,0x04,0x 04,0x04])

Теперь вы можете написать свою программу.

Листинг программы: На рис. 5.53 показан листинг программы (Программа:**arrow**). В начале программы в программу импортируются ЖК-модули I2C. Затем переменная **arrow** присваивается пользовательскому коду с индексом 0 (может быть от 0 до 7). Символ стрелки отображается на ЖК-дисплее с помощью функции **putstr()**.

```
#------------------------------------------------------------
#      ОТОБРАЖЕНИЕ ПОЛЬЗОВАТЕЛЬСКОГО СИМВОЛА
#      ======================================
#
# Эта программа отображает символ стрелки на ЖК-дисплее
#
# Автор: Доган Ибрагим
# Файл  : arrow.py
# Дата: октябрь 2022 г
#------------------------------------------------------------
import machine
from machine import I2C
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
import utime

I2C_ADDR = 0x27                    # ЖК-адрес I2C
NRows = 2                          # Количество строк
NColumns = 16                      # Количество столбцов

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, NRows, NColumns)

arrow = bytearray([0x04,0x0E,0x15,0x04,0x04,0x04,0x04,0x04])
lcd.custom_char(0, arrow)
lcd.putstr(chr(0))
```

Рис.*5.53:* Программа: **arrow**.

На рис.5.54 показан символ, отображаемый на ЖК-дисплее.

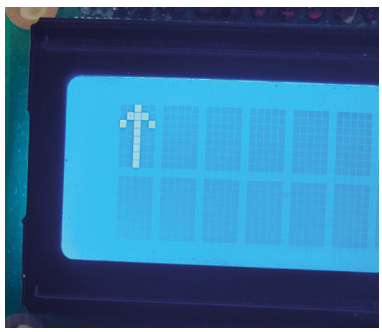


Рис.*5.54:* Отображение пользовательского символа.

5.21 Проект 16: ЖК-игральные кости (кубики)

Описание: Это проект ЖК-игры в кости. Нажатие кнопки отображает два случайных десятичных числа на ЖК-дисплее в течение 3 секунд, после чего дисплей очищается и готов к генерации новых чисел.

Блок-схема: На рис. 5.55 показана блок-схема проекта.

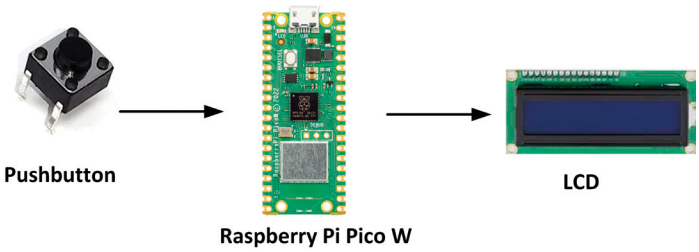


Рис.5.55: Блок-схема проекта.

Принципиальная схема: Принципиальная схема показана на рис. 5.56. Кнопка подключена к порту GP17 Pico.

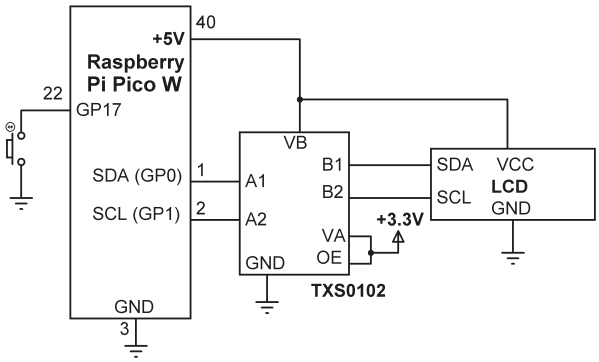


Рис.5.56: Принципиальная схема.

Листинг программы: На рис. 5.57 показан листинг программы (Program:**LCDice**). В начале программы в программу импортируются ЖК-модули I2C. Кнопка настроена как вход с включенным внутренним подтягивающим резистором. Внутри основного цикла программы отображается сообщение «**Ready**», и программа ожидает, пока пользователь не нажмет кнопку. Когда кнопка нажата, генерируются два случайных числа от 1 до 6, которые отображаются на ЖК-дисплее в течение 3 секунд.

```
#-------------------------------------------------------------
#              I2C LCD DICE
#              ============
#
# Эта программа отображает два числа
# на кубиках от 1 до 6 при нажатии кнопки
```

```
#
# Автор: Доган Ибрагим
# Файл  : LCDDice.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
import machine
from machine import I2C, Pin
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
import utime
import random

I2C_ADDR = 0x27                     # ЖК-адрес I2C
NRows = 2                           # Количество строк
NColumns = 16                       # Количество столбцов

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, NRows, NColumns)

Button = Pin(17, Pin.IN, Pin.PULL_UP)
lcd.clear()

while True:
    lcd.putstr("Ready")
    while Button.value() == 1:       # Ждем нажатия кнопки
        pass
    lcd.clear()                      # Очистить экран
    rnd1 = random.randint(1, 6)      # Генерация числа
    rnd2 = random.randint(1, 6)      # Генерация другого числа
    lcd.putstr(str(rnd1))            # Показать первое числ
    lcd.putstr("   ")
    lcd.putstr(str(rnd2))            # Показать второе число
    while Button.value() == 0:       # Ждем отпускания кнопки
        pass
    utime.sleep(3)                   # Отображение на 3 секунды
    lcd.clear()
```

Рис. *5.57:* Программа **LCDDice**.

5.22 Проект 17: Использование модуля часов реального времени (RTC) – установка/отображение дата и время воспроизведения

Описание: В этом проекте модуль RTC используется для установки/получения часов реального времени для вашего Raspberry Pi Pico W. Дата и время модуля RTC устанавливаются с клавиатуры и отображаются на ЖК-дисплее I2C.

Модуль RTC

Модуль RTC — это модуль точных часов, который можно использовать в приложениях на основе микроконтроллеров. Модуль предоставляет информацию о секундах, минутах, часах, дне, дате, месяце и годе. Високосный год устанавливается автоматически, где модуль действует до 210 года.

Модуль RTC (рис. 5.58) основан на синхронной последовательной связи (несовместимой с I2C). Как показано на рисунке, в этом проекте используется модуль RTC на основе чипа DS1302 RTC со встроенным кварцем для синхронизации. Плоская батарейка CR2032 используется для поддержания работы часов, когда модуль отключен от Pico.

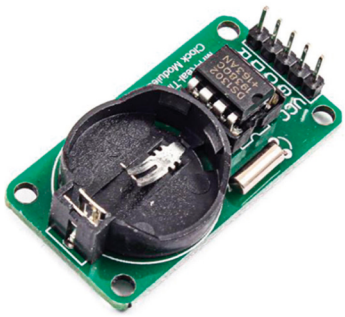


Рис.*5.58:* Модуль *RTC.*

Модуль DS1302 RTC имеет следующие 4 пина:

VCC:	Источник питания (+3,3 В
GND:	GND
CLK:	clock
DAT:	data
RST:	reset

Блок-схема: На рис.5.59 показана блок-схема проекта.

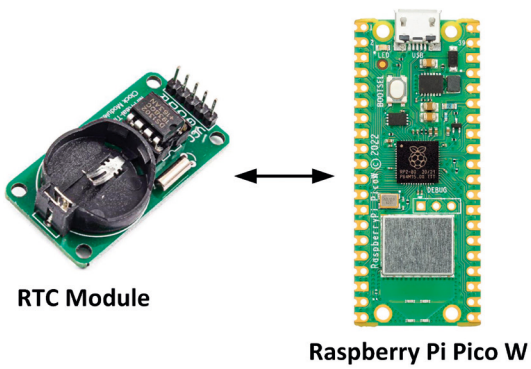


Рис.5.59: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис.5.60. Соединения между модулем RTC и Pico следующие:

RTC module	Raspberry Pi Pico W
CLK	GP16
DAT	GP17
RST	GP18
VCC	+3.3V
GND	GND

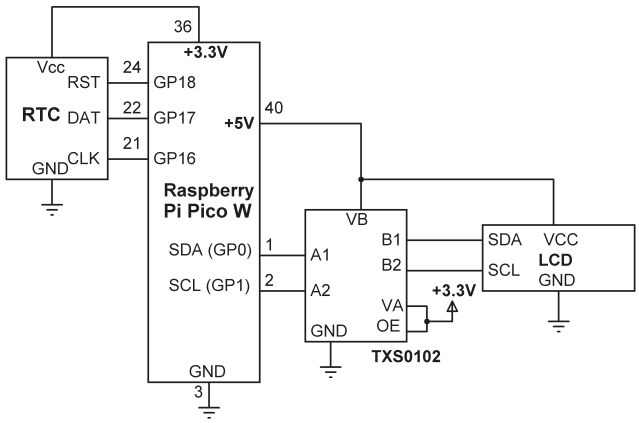


Рис.*5.60:* Принципиальная схема проекта.

Листинг программы: В этом проекте используется следующая библиотека DS1320:

https://github.com/omarbenhamid/micropython-ds1302-rtc

Вы должны извлечь файлы **1302.py** (доступны из папки с кодами), а затем сохранить их на Raspberry Pi Pico W с именем **1302.py**. Эта библиотека предоставляет следующие функции:

start()	запустить RTC
stop()	остановить/приостановить RTC
date_time(DT = None)	получить/установить DateTime. Если параметр не указан, он вернет текущую дату и время
year(year = None)	получить/установить год
month(month = None)	получить/установить месяц
day(day = None)	получить/установить день
weekday(weekday = None)	получить/установить день недели
hour(hour = None)	получить/установить час
minute(minute = None)	получить/установить минуты
second(second = None)	получить/установить секунду
ram(reg, dat = None)	получить/установить данные оперативной памяти (31 байт)

На рис.5.61 показан листинг программы (Программа:**rtc**). В начале программы импортируются ds1302 и другие модули, используемые в программе. ЖК-дисплей инициализируется так же, как и в предыдущих программах на основе ЖК-дисплея I2C. Затем пользователю будет предложено установить дату и/или время RTC. Если ответ **Y** на этот вопрос, то текущие данные о дате и времени считываются с клавиатуры, а затем вызывается функция **date_time()** для установки даты ивремени RTC. Информация о дате и времени отображается на ЖК-дисплее. Обратите внимание, что RTC работает от батареи и продолжает обновлять дату ивремя даже после отключения питания от Pico. Если дата и время правильные, пользователь должен ввести **N** в вопросе, чтобы пропустить обновление даты и/иливремени.

```
#-------------------------------------------------------
#             МОДУЛЬ ЧАСОВ РЕАЛЬНОГО ВРЕМЕНИ
#             ================================
#
# Эта программа использует модуль DS1307 RTC
# для установки и отображения даты и времени на LCD
#
# Автор: Доган Ибрагим
# Файл   : rtc.py
# Дата: октябрь 2022 г
#-------------------------------------------------------
import machine
from machine import I2C, Pin
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
import utime
import ds1302

I2C_ADDR = 0x27                    # ЖК-адрес I2C
NRows = 2                          # Количество строк
NColumns = 16                      # Количество столбцов

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, NRows, NColumns)

ds = ds1302.DS1302(Pin(16),Pin(17),Pin(18))

sel = input("Do you wish to set Date/Time ?: ")
if sel.upper() == "Y":
  print("Date time format is e.g. 2023, 9, 14, 4, 20, 22, 1, 0")
  Y = int(input("Enter Year    : "))
  M = int(input("Enter Month   : "))
  D = int(input("Enter Day     : "))
  day = int(input("Enter day no  : "))
  h = int(input("Enter Hour    : "))
  m = int(input("Enter Minutes : "))
```

```
  s = int(input("Enter Seconds : "))
  ss = int(input("Enter ss      : "))
  ds.date_time([Y,M,D,day,h,m,s,ss])

while True:
    (Y,M,D,day,hr,m,s)=ds.date_time()
    if s < 10:
        s = "0" + str(s)
    if m < 10:
            m = "0" + str(m)
    if hr < 10:
            hr = "0" + str(hr)
    if D < 10:
            D = "0" + str(D)
    if M < 10:
            M = "0" + str(M)
    lcd.move_to(0,0)
    lcd.putstr("Time:")
    lcd.move_to(6,0)
    lcd.putstr(str(hr) + ":" + str(m) + ":" + str(s))
    lcd.move_to(0,1)
    lcd.putstr("Date:")
    lcd.move_to(6,1)
    lcd.putstr(str(D) + "/" + str(M) + "/" + str(Y))
```

Рис.5.61: Программа **rtc**.

На рис.5.62 показано обновление даты и времени до текущих значений с клавиатуры. Пример дисплея показан на рис. 5.63.

```
Do you wish to set Date/Time ?: Y
Date time format is e.g. 2023, 9, 14, 4, 20, 22, 1, 0
Enter Year    : 2022
Enter Month   : 9
Enter Day     : 15
Enter day no  : 5
Enter Hour    : 16
Enter Minutes : 53
Enter Seconds : 15
Enter ss      : 0
```

Рис.*5.62:*Обновление даты и времени.

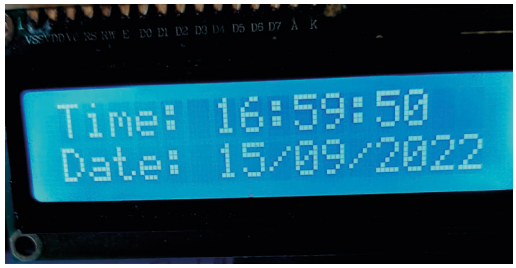


Рис.5.63: Пример отображения даты и времени.

5.23 Проект 18: Сохранение температуры с отметкой времени

Описание: В этом проекте микросхема аналогового датчика температуры TMP36 используется вместе с модулем RTC DS1302. Здесь температура окружающей среды считывается каждые 2 секунды для 10 показаний и сохраняется на Raspberry Pi Pico W с отметкой времени. Затем сохраненные данные отображаются на экране Тонни через 10 секунд.

Цель: Цель этого проекта — показать, как данные датчика могут быть сохранены на Pico с отметкой времени.

Блок-схема: На рис.5.64 показана блок-схема проекта.

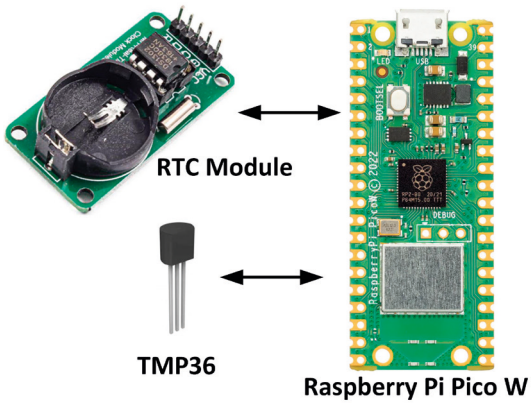


Рис.5.64: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис.5.65. Здесь микросхема датчика температуры TMP36 подключена к одному из аналоговых входов Pico. Модуль RTC подключается к Pico, как и в предыдущем проекте.

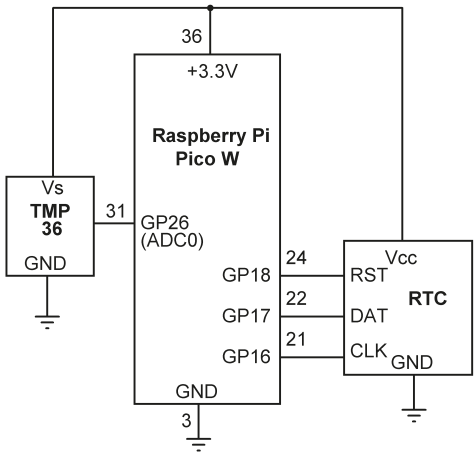


Рис.5.65: Принципиальная схема проекта.

Листинг проекта: Предполагается, что текущая дата и время уже установлены в модуле RTC. На рис.5.66 показан листинг программы (Program:**rtctmp36**). В начале программы используемые модули импортируются, а необходимое количество выборок данных для сбора сохраняется в переменных **samples**. Файл **Temperature.txt** открыт для записи. Цикл формируется для считывания температуры окружающей среды каждую секунду. Температура хранится в переменной Temp. Внутри этого цикла дата и время считываются из модуля RTC и сохраняются в переменной **Cur**. Затем показания температуры сохраняются с отметкой времени в файле. По окончании сбора данных программа ждет 10 секунд, открывает файл для чтения, считывает и отображает собранные данные.

```
#-------------------------------------------------------------
#      СОХРАНИТЬ ТЕМПЕРАТУРУ С ОТМЕТКОЙ ВРЕМЕНИ
#      ============================================
#
# Эта программа использует модуль DS1307 RTC для чтения текущей
# даты и времени. Температура окружающей среды считывается
# и сохраняется на Pico с отметкой времени. Затем сохраненные
# данные отображаются на терминале
#
# Автор: Доган Ибрагим
# Файл   : rtctmp36.py
# Дата: октябрь 2022 г
#-------------------------------------------------------------
import machine
from machine import Pin, ADC
import utime
import ds1302

samples = 10
ds = ds1302.DS1302(Pin(16),Pin(17),Pin(18))
```

```
AnalogIn = ADC(0)                                   # Канал АЦП 0
Conv = 3300 / 65535                                 # Коэффициент преобразования
fp = open("Temperature.txt", "w")

print("Data Collection Started")
print("=======================")
print()

for i in range(samples):
    V = AnalogIn.read_u16()                         # Чтение температуры
    mV = V * Conv                                   # Преобразовать в вольты
    Temp = (mV - 500.0) / 10.0                      # Преобразовать в температуру
#
# COnfigrue date and time
#
    (Y,M,D,day,hr,m,s) = ds.date_time()
    if s < 10:
        s = "0" + str(s)
    if m < 10:
            m = "0" + str(m)
    if hr < 10:
            hr = "0" + str(hr)
    if D < 10:
            D = "0" + str(D)
    if M < 10:
            M = "0" + str(M)

    Cur=str(D)+"/"+str(M)+"/"+str(Y)+" "+str(hr)+":"+str(m)+":"+str(s)
    lin = Cur + " " + str(Temp)[:5] + "\n"
    fp.write(lin)
    utime.sleep(2)
fp.close()

#
# Wait 10 seconds, read the data and display on terminal
#
utime.sleep(10)
print("End Of Data Collection. Collected Data:")
print("=======================================")

fp = open("Temperature.txt", "r")
for i in range(samples):
    lin = fp.readline()
    print(lin)
```

```
fp.close()
print("End of Program")
```

Рис.5.66: Программа: **rtctmp36**.

На рис.5.67 показан пример вывода программы.

```
Data Collection Started
=======================

End Of Data Collection. Collected Data:
=======================================
15/09/2022 20:01:59 23.33

15/09/2022 20:02:01 23.65

15/09/2022 20:02:03 23.65

15/09/2022 20:02:05 24.13

15/09/2022 20:02:07 26.55

15/09/2022 20:02:09 27.84

15/09/2022 20:02:11 27.84

15/09/2022 20:02:13 26.87

15/09/2022 20:02:15 27.19

15/09/2022 20:02:17 26.23

End of Program
```

Рис.5.67: Пример вывода.

5.24 Проект 19: GPS – Отображение географических координат на ЖК-дисплее

Бывают случаи, особенно при мобильной работе, когда вам может понадобиться узнать географические координаты (например, широту и долготу) вашей станции. В этом проекте вы используете GPS для считывания и отображения географических координат вашей станции на ЖК-дисплее.

Описание: Приемники GPS получают географические данные со спутников GPS и предоставляют точную информацию о положении пользователя на Земле. Эти спутники вращаются вокруг Земли на высоте около 20 000 км и каждый день совершают два полных оборота. Чтобы приемник мог определить свое положение, необходимо, чтобы приемник связался как минимум с 3 спутниками. Поэтому, если приемник не имеет четкого обзора неба, может оказаться невозможным определить его положение на Земле. В некоторых приложениях используются внешние антенны, чтобы можно было принимать даже слабые сигналы со спутников GPS.

Данные, отправляемые GPS-приемником, имеют текстовый формат и известны как NMEA Sentenc-es. Каждое предложение NMEA начинается с символа $, а значения в предложении разделяются запятыми. Некоторые из предложений NMEA, возвращаемых приемником GPS, приведены ниже:

$GPGLL: T Это предложение возвращает местную географическую широту и долготу.

$GPRMC: это предложение возвращает местную географическую широту и долготу, скорость, угол пути, дату, время и магнитное склонение.

$GPVTG: это предложение возвращает истинный трек, магнитный трек и путевую скорость.

$GGGA: это предложение возвращает местную географическую широту и долготу, время, качество фиксации, количество отслеживаемых спутников, горизонтальное разбавление положения, высоту, высоту геоида и данные DGPS.

$GPGSV: 4 предложения с этим заголовком. Эти предложения возвращают количество спутников в поле зрения, номер спутника, высоту, азимут и SNR.

В этом проекте используется плата GPS Click (www.mikroe.com). Это небольшой GPS-приемник (см. рис. 5.68), основанный на GPS-приемнике типа LEA-6S. Эта плата работает с напряжением +3,3 В и имеет два типа выходов: I2C или последовательный выход. В этом проекте используется последовательный вывод по умолчанию, который работает со скоростью 9600 бод. Внешняя динамическая антенна может быть прикреплена к плате, чтобы улучшить ее прием для использования внутри помещений или для использования в местах, где не может быть ясного обзора неба.

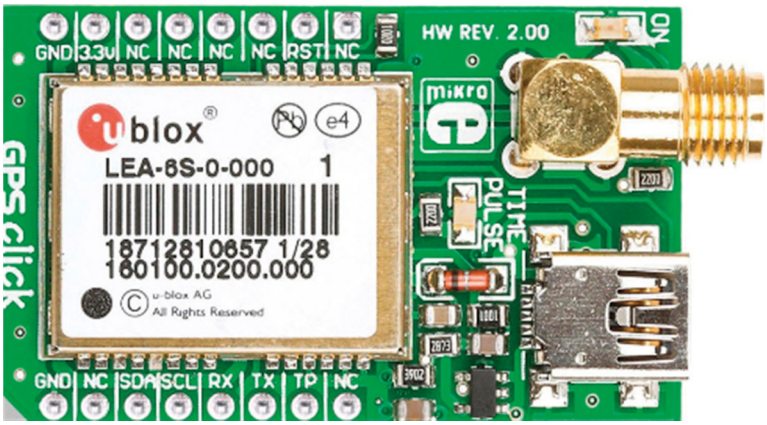


Рис.5.68: Плата GPS Click.

На рис.5.69 показан полный список предложений NMEA, выводимых с панели GPS Click каждую секунду.

```
$GPGLL,5127.3917,N,00003.13141,E,10534.00,A,A*67
$GPRMC,05305.00,A,5127.35909,,0003.13148,E,0.030,,270919,,,A*7E
$GPVTG,,T,,M,0030,N,0.055,K,A*20
$GGGA,105305.00,5127.35909,N,00003.13148,E,1,09,1.18,46.5,M,45.4,M,,*66
$GPSA,A,3,01,32,08,28,18,03,22,14,11,,,,2.12,1.18,1.76*06
$GPGSV,4,1,13,01,7,304,40,03,40,224,31,08,38,165,32,10,05,054,*77
$GPGSV,4,2,13,11,83,217,3,14,39,094,24,17,17,314,22,18,73,091,41*76
$GPGSV,4,3,13,22,63,219,33,24,1,002,,27,05,150,,28,30,284,28*7F
$GPGSV,4,4,13,32,34,063,35*4E
```

Рис.5.69: Предложения NMEA, выводимые с панели GPS Click.

Плата GPS Click представляет собой двухрядный модуль 2x8 контактов со следующей конфигурацией контактов (контакт 1 — это верхний левый контакт модуля):

1: нет соединения	16: нет соединения
2: Reset	15: нет соединения

3: нет соединения	14: TX
4: нет соединения	13: RX
5: нет соединения	12: SCL
6: нет соединения	11: SDA
7: +3.3 V	10: нет соединения
8: GND	9: GND

При последовательной работе требуются только следующие контакты: +3,3 В, GND, TX. В этом проекте к плате GPS Click прикреплена внешняя динамическая антенна, так как она использовалась в помещении.

$GPGLL и $GPGGA — одни из часто используемых предложений NMEA. В этом проекте вы будете использовать $GPGGA, чтобы получить широту и долготу вашего места, а затем отобразить их на ЖК-дисплее. Формат предложения $GPGGA выглядит следующим образом:

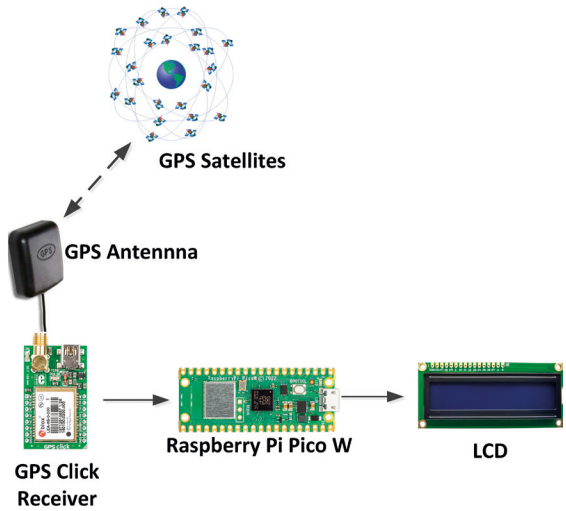
```
$GPGGA, 161229.487, 3723.2475, N, 12158.3416, W, 1, 07, 1.0, 9.0, M, , , , 0000*18
```

Поля в этом предложении можно расшифровать следующим образом:

$GPGGA	Заголовок протокола GGA
Время UTC	161229.487
Широта	3723,2475 (37 градусов, 23,2475 минуты)
Индикатор N/S	N = север, S = юг
Долгота	12158,3416 (121 градус, 58,3416 минут)
Индикатор E/W	E = Восток или W = Запад
Индикатор определения местоположения 1	GPS-предложения Индикатор определения местоположения
Используемые спутники	Диапазон от 0 до 12
HDOP	Горизонтальное снижение точности
Высота над уровнем моря	9,0 метров
Единицы	Метры
Разделение геоида	Метры
Единицы	Метры
Возраст дифф. корр.	Second
Диф. исх. ID	0000
Контрольная сумма	*18
<CR><LF>	Конец завершения сообщения

Обратите внимание, что поля разделены запятыми. Достоверность данных отображается в поле состояния. Значение больше 0 в этом поле указывает на то, что данные действительны.

Блок-схема: На рис. 5.70 показала блок-схему проекта.



Рис*/5.70:* Блок-схема проекта.

Принципиальная схема: Принципиальная схема показана на рис. 5.71. Выход TX GPS-модуля подключен к порту GP9 (UART1 RX) Pico. Модуль работает от +3,3 В. К GPS модулю подключена внешняя антенна, так как проект был построен и испытан в помещении.

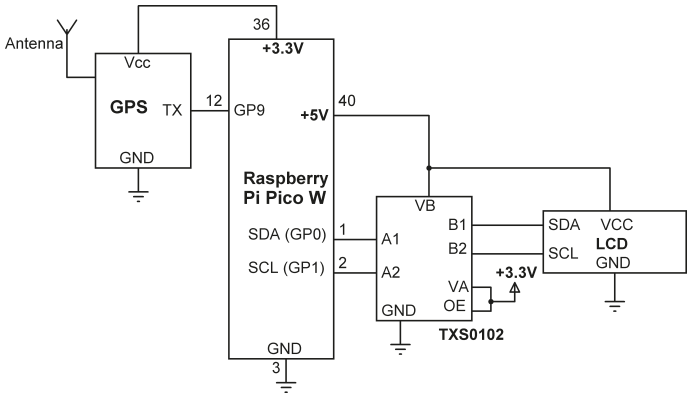


Рис.5.71: Принципиальная схема.

Список программ: На рис.5.72 показан список программ (Program:**gps**). В начале программы импортируются необходимые программе модули. Обратите внимание, что модуль UART также импортируется, так как этот модуль используется для получения последовательных данных от GPS. UART инициализируется со скоростью 9600 бод, которая является скоростью передачи данных по умолчанию для модуля GPS. Параметры **txandrx** установлены на 8 и 9, что соответствует GP8 и GP9 соответственно. Это контакты передачи и приема UART1 Pico. Функция **Get_GPS()** получает последовательные данные от модуля GPS через UART с помощью функции **readline()**.

Полученные последовательные данные сохраняются в байтовом буфере, называемом **buff**. Программа ищет ключевое слово **$GPGGA** в полученных данных. Как только это ключевое слово обнаружено, функция **split()** используется для разделения полученных данных на несколько частей, разделенных запятыми. Затем программа извлекает широту и долготу вместе с их направлениями и возвращает эти данные в основную программу, где они отображаются на ЖК-дисплее. Если данные, полученные от GPS, недействительны, на ЖК-дисплее отображается **NO DATA** - НЕТ ДАННЫХ. Обратите внимание, что широта указана в поле 2, направление широты — в поле 3, долгота — в поле 4, а направление долготы — в поле 5.

```
#-------------------------------------------------------------
#                GPS ГЕОГРАФИЧЕСКИЕ КООРДИНАТЫ
#                ================================
#
# Это проект GPS. Определяются географические координаты
# местоположения, которые отображаются на ЖК-дисплее
#
# Автор: Доган Ибрагим
# Файл  : gps.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
import machine
from machine import I2C, UART, Pin
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
import utime

uart = UART(1,baudrate=9600,tx=Pin(8),rx=Pin(9),
            bits=8,parity=None,stop=1)

I2C_ADDR = 0x27                    # ЖК-адрес I2C
NRows = 2                          # Количество строк
NColumns = 16                      # Количество столбцов

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, NRows, NColumns)

buff=bytearray(255)

#
# Эта функция получает и извлекает широту и долготу
# из предложения NMEA $GPGGA
#
def Get_GPS():
  while True:
       uart.readline()
       buff = str(uart.readline())
       sdata = buff.split(",")                          # Разделить данные
```

```
       if(sdata[0] == "b'$GPGGA" and len(sdata) > 10):
         if sdata[6] == "0":                          # Данные верны?
           return ("No", "DATA")
         else:
          lat = sdata[2]                              # Получить широту
          latdir = sdata[3]                           # Latitude dir
          lon = sdata[4]                              # Получить долготу
          londir = sdata[5]                           # Longitude dir
          deg = lat[0:2]
          min = lat[2:]
          latitude = str(deg) + " " + str(min) + "  " + str(latdir)

          deg = lon[0:3]
          min = lon[3:]
          longitude = str(deg) + " " + str(min) + " " + str(londir)
          return (latitude, longitude)
       else:
          utime.sleep(0.1)

#
# Получите GPS-координаты и отобразите их на ЖК-дисплее
#
try:
  while True:
      lat, lon = Get_GPS()                                    # Декодировать
      lcd.clear()
      lcd.move_to(0, 0)
      lcd.putstr(lat)                                         # Показать широту
      lcd.move_to(0, 1)
      lcd.putstr(lon)                                         # Показать длину
      utime.sleep(0.2)

except KeyboardInterrupt:
   uart.flush()
   uart.deinit()
```

Рис.5.72: Программа: **gps**.

На рис.5.73 показан пример отображения координат GPS. ЖК-экран должен обновляться при получении данных от GPS.

Рис.5.73: Пример отображения.

Глава 6 • Широтно-импульсная модуляция (ШИМ)

6.1 Обзор

Цифровая ШИМ, также пишется как широтно-импульсная модуляция, обычно используется для управления мощными нагрузками, такими как двигатели, приводы, нагреватели и т. д. Как вы увидите в этой главе, ШИМ в основном представляет собой положительную прямоугольную форму, ширина и частота импульса которой могут быть изменены. Изменяя ширину импульса, можно эффективно изменить среднее значение напряжения, подаваемого на нагрузку.

6.2 Основы теории широтно-импульсной модуляции

Широтно-импульсная модуляция (ШИМ) — широко используемый метод управления мощностью, подаваемой на аналоговые нагрузки, с использованием цифровых сигналов. Хотя аналоговые напряжения (и токи) могут использоваться для управления подаваемой мощностью, они имеют несколько недостатков. Для управления большими аналоговыми нагрузками требуются большие напряжения и токи, которые нелегко получить с помощью стандартных аналоговых схем и ЦАП. Прецизионные аналоговые схемы могут быть мощными, большими и дорогими, а также чувствительными к шуму. При использовании метода ШИМ среднее значение напряжения (и тока), подаваемого на нагрузку, управляется путем включения и выключения напряжения питания с высокой скоростью. Чем больше время включения питания, тем выше среднее напряжение, подаваемое на нагрузку.

На рис.6.1 показан типичный сигнал ШИМ, где сигнал представляет собой повторяющийся положительный прямоугольный импульс с периодом T, временем включения T_{ON} и временем выключения T – T_{ON} секунд. Минимальное и максимальное значения напряжения, подаваемого на нагрузку, равны 0 и V_P соответственно. Частота переключения ШИМ обычно устанавливается высокой (обычно порядка нескольких кГц), чтобы она не влияла на потребляющую мощность нагрузку. Основное преимущество ШИМ заключается в том, что нагрузка работает эффективно, поскольку потери мощности в коммутационном устройстве невелики. Когда ключ включен, на нем практически нет падения напряжения, а когда ключ выключен, ток на нагрузку не поступает.

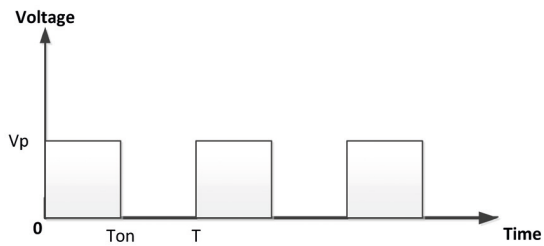


Рис.6.1: Сигнал ШИМ.

Сважность (или D) сигнала ШИМ определяется как отношение времени включения к его периоду. Выражаясь математически,

Скважность (D) = T_{ON} / T

Скважность обычно выражается в процентах, поэтому

$$D = (T_{ON} / T) \times 100\ \%$$

Изменяя скважность от 0 % до 100 %, вы можете эффективно управлять средним напряжением, подаваемое на нагрузку, в диапазоне от 0 до Vp.

Среднее значение напряжения, приложенного к нагрузке, можно рассчитать, рассмотрев общий сигнал ШИМ, показанный на рис.6.1. Среднее значение A сигнала f(t) с периодом T и пиковым значением ymax и минимальным значением ymin вычисляется как:

$$A = \frac{1}{T}\int_0^T f(t)dt$$

или,

$$A = \frac{1}{T}\left(\int_0^{T_{ON}} y_{\max}\ dt + \int_{T_{ON}}^{T} y_{\min\ dt}\right)$$

В ШИМ $y_{min} = 0$, и приведенное выше уравнение становится

$$A = \frac{1}{T}\left(T_{ON}\ y_{\max}\right)$$

или

$$A = D\ y_{\max}$$

Как видно из приведенного выше уравнения, среднее значение напряжения, подаваемого на нагрузку, прямо пропорционально скважности сигнала ШИМ, и, изменяя его, вы управляете средним напряжением нагрузки. На рис. 6.2 показано среднее напряжение при различных значениях скважности.

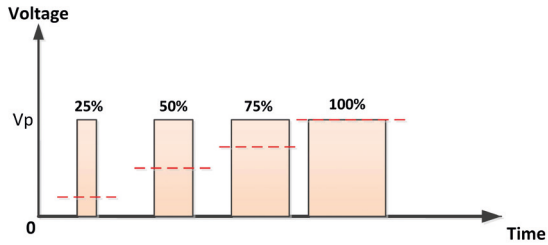


Рис.*6.2:* Среднее напряжение *(*показано пунктирной линией*)*, подаваемое на нагрузку.

Интересно отметить, что при соответствующей фильтрации нижних частот ШИМ можно использовать в качестве ЦАП, если в MCU нет канала ЦАП. Изменяя рабочий цикл, вы можете эффективно изменять среднее аналоговое напряжение, подаваемое на нагрузку.

6.3 ШИМ-каналы Raspberry Pi Pico W

Raspberry Pi Pico и Pico W имеют 16 программируемых каналов ШИМ. На рис. 6.3 показано расположение выводов этих каналов. Каждый канал обозначается буквой и числом, например PWM_A[0]. Некоторые из 16 каналов ШИМ, расположенных на левой стороне, дублируются на правой стороне, и только один из дублированных каналов может использоваться в приложении.

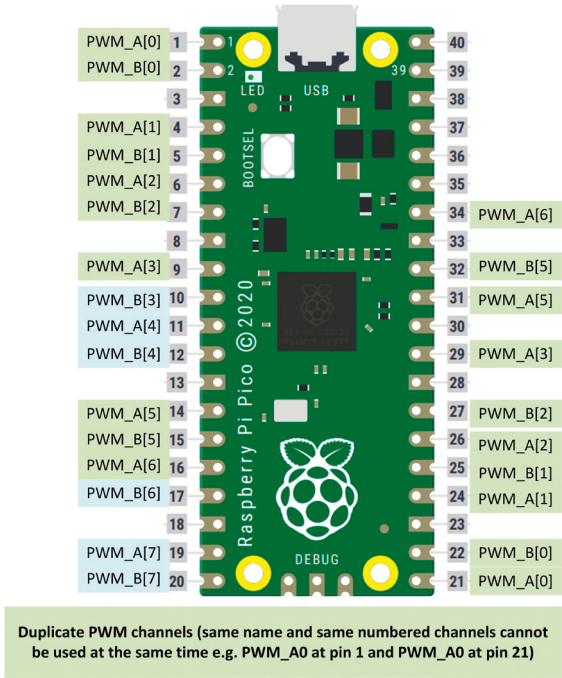


Рис.6.3: ШИМ-каналы Pico/Pico W.

Доступ к ШИМ-каналам можно получить одним из двух способов. Например, к каналу PW-M_A[0], подключенному к GP0, можно получить доступ как:

```
import machine
ch = machine.PWM(machine.Pin(0))
```

или как

```
from machine import PWM, Pinch = PWM(Pin(0))
```

Частоту сигнала ШИМ,например частоту 1000 Гц, можно установить с помощью следующего оператора:

```
ch.freq(1000)
```

Скважность можно установить в диапазоне от 0% до 100%, установив её от 0 до 65535. Скважность, например 50%, можно установить с помощью следующего оператора:

ch.duty_u16(32767)

Примеры проектов приведены в этой главе, чтобы проиллюстрировать, как можно использовать ШИМ-каналы Pico W.

6.4 Проект 1: Генерация сигнала ШИМ с частотой 1000 Гц и скважностью 50 %.

Описание: В этом проекте вы создаете сигнал ШИМ с частотой 1000 Гц и коэффициентом заполнения 50%.

Цель: **ц**ель этого проекта — показать, как можно использовать функции ШИМ Raspberry Pi Pico W.

Принципиальная схема: в этом проекте используется порт GP0 и к нему подключен осциллограф для наблюдения за сигналом ШИМ.

Листинг программы: Листинг программы довольно прост, и он приведен ниже. Обратите внимание, что рабочий цикл установлен на 32767, что соответствует 50%:

```
from machine import Pin, PWM
ch = PWM(Pin(0))                                    # ШИМ на GP0
ch.freq(1000)                                       # Частота = 1000 Гц
ch.duty_u16(32767)                                  # Скважность 50%
while True:
                pass
```

На рис.6.4 показан сгенерированный сигнал на осциллографе. Здесь горизонтальная ось составляла 0,5 мс/деление, а вертикальная ось — 1 В/деление. Ясно, что период генерируемого сигнала составляет 1 мс (частота = 1000 Гц), коэффициент заполнения составляет 50%, а амплитуда составляет около 3 В.

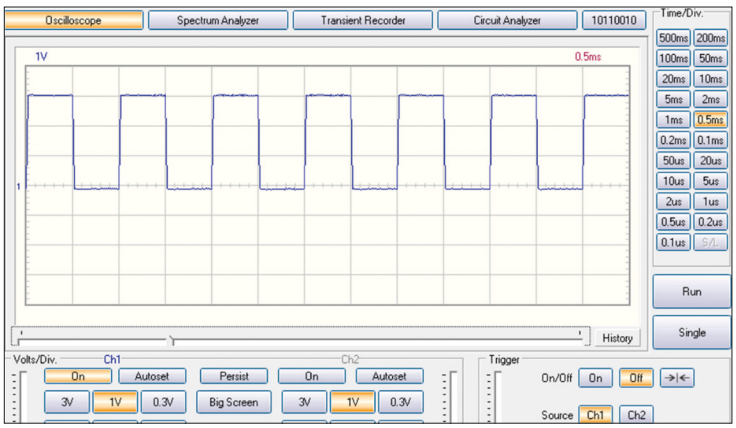


Рис.6.4: Сгенерированный сигнал ШИМ.

Примечание Автор сгенерировал чистые и правильные формы сигналов ШИМ до нескольких МГц без каких-либо шумов.

6.5 Проект 2: Изменение яркости светодиода

Описание: В этом проекте внешний светодиод подключен к выводу GP0 порта через токоограничивающий резистор 470 Ом. Программа изменяет яркость светодиода, изменяя скважность напряжения ШИМ, подаваемого на светодиод.

Цель: Цель этого проекта — показать, как ШИМ можно использовать в проекте. Блок-схема и принципиальная схема этого проекта такие же, как на рис. 3.1 и рис. 3.2 соответственно, где светодиод подключен к выводу GP1 порта.

Листинг программы: Рис.6.5 показывает листинг программы (Программа: LEDfade). Частота установлена на 1000 Гц, чтобы светодиод светился постоянно (т. е. не мигал). По мере увеличения скважности от 0% до 100% яркость светодиода постепенно увеличивается.

```
#-------------------------------------------------------------
#               ИЗМЕНЕНИЕ ЯРКОСТИ СВЕТОДИОДА
#               ================================
#
# В этой программе светодиод подключен к контакту порта GP1.
# Яркость светодиода непрерывно изменяется за счет изменения
# скважности от 0% до 100%.
#
# Автор: Доган Ибрагим
# Файл   : LEDfade.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
from machine import Pin, PWM
import utime

ch = PWM(Pin(1))                                # ШИМ на GP1
ch.freq(1000)                                   # Частота = 1000 Гц

i = 0
while True:                                     # Делать всегда
    ch.duty_u16(i)                              # Изменять скважность
    utime.sleep_ms(300)                         # Задержка 300 мс
    i = i + 5000                                # Увеличение i
    if i > 65535:
        i = 0
```

Рис.6.5: Программа: **LEDfade**.

6.6 Проект 3: Электронная свеча

Описание: В проекте к Pico подключен светодиод. Яркость светодиода изменяется случайным образом, создавая эффект свечи.
Блок-схема и принципиальная схема проекта представлены на рис.3.1 и рис.3.2 соответственно.

Листинг программы: Рис.6.6 показывает листинг программы (Программа: ecandle). Генерируется случайное число от 0 до 65535, и это число используется для изменения рабочего цикла светодиода, создавая эффект настоящей свечи.

```
#------------------------------------------------------------
#                ЭЛЕКТРОННАЯ СВЕЧА
#                ==================
#
# В этой программе светодиод подключен к выводу GP1 порта.
# Яркость светодиода изменяется случайным образом,
# чтобы получить эффект электронной свечи.
#
# Автор: Доган Ибрагим
# Файл  : ecandle.py
# Дата: октябрь 2022 г.
#------------------------------------------------------------
from machine import Pin, PWM
import utime
import random

ch = PWM(Pin(1))                        # ШИМ на GP1
ch.freq(1000)                           # Частота = 1000 Гц

while True:                             # Делать всегда
    ch.duty_u16(random.randint(100, 65535))
    utime.sleep_ms(300)                 # Задержка 300 мс
```

Рис.6.6: Программа: **ecandle**.

6.7 Проект 4: Изменение скорости коллекторного двигателя DC

Описание: Это простой проект, в котором небольшой коллекторный двигатель подключается к PICO через силовой МОП-транзистор. Кроме того, к одному из аналоговых входов микроконтроллера подключен потенциометр. В этом проекте скорость двигателя изменяется перемещением ползунка потенциометра.

Блок-схема: На рис.6.7 показана блок-схема проекта. Драйвер двигателя (MOS-FET транзистор) и потенциометр подключены к микроконтроллеру.

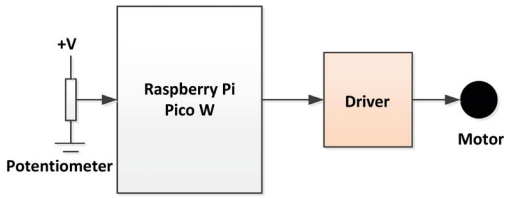


Рис.6.7: Блок-схема проекта.

Двигатель постоянного тока в этом проекте управляется с помощью ШИМ, как и в предыдущем проекте. Изменяя плечо потенциометра, аналоговое напряжение, считываемое микроконтроллером, изменяется, и это, в свою очередь, изменяет рабочий цикл (скважность) ШИМ напряжения, подаваемого на двигатель, что приводит к изменению скорости двигателя.

Принципиальная схема: Принципиальная схема проекта показана на рис.6.8. Потенциометр подключен к каналу 0 АЦП (GP26, пин 31). Мотор подключен к GP17 (вывод 22) через транзистор MOSFET типа IRL540. Для двигателя рекомендуется использовать внешний источник питания.

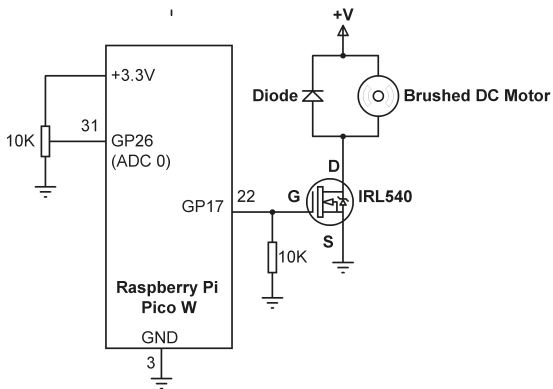


Рис.6.8: Принципиальная схема проекта.

Листинг программы: На рис. 6.9 показан листинг программы (Программа: **Motor**). Данные, считываемые с АЦП, варьируются от 0 до 65535, когда ползунок потенциометра перемещается с одной стороны на другую. Эти данные используются для изменения скважности от 0% до 100%.

```
#-------------------------------------------------------------
#               ИЗМЕНЕНИЕ  СКОРОСТИ  МОТОРА
#               ============================
#
# В этом проекте коллекторный двигатель постоянного тока подключен
# Pico W. Кроме того, потенциометр подключен к каналу № 0 АЦП.
# Перемещение ползунка потенциометра изменяет скорость двигателя.
```

```
## Автор: Доган Ибрагим# File  : Motor.py# Дата: октябрь 2022
г.
#--------------------------------------------------------------
from machine import Pin, PWM, ADC



Pot = ADC(0)                            # Pot  на канале 0
Motor = Pin(17, Pin.OUT)                # Мотор на GP17 как выход

ch = PWM(Pin(17))                       # ШИМ с GP17
ch.freq(1000)                           # Частота = 1000 Гц

while True:                             # Делать всегда
    duty = Pot.read_u16()               # Чтение данных потенциометра
    ch.duty_u16(duty)                   # Изменять скважность
```

Рис.6.9:Программа: **Motor**.

6.8 Проект 5: Генератор частоты с ЖК-дисплеем и потенциометром

Описание: Это проект генератора частоты. Потенциометр и ЖК-дисплей подключены к Raspberry Pi Pico W. Изменяя положение ползунка потенциометра, вы изменяете частоту генерируемого сигнала ШИМ. Частота генерируемого сигнала отображается на ЖК-дисплее. Рабочий цикл (скважность) сгенерированного сигнала установлен на 50%. Частоты примерно до 65 535 Гц можно получить, перемещая ползунок потенциометра в одну сторону.

Блок-схема: На рис.6.10 показана блок-схема проекта.

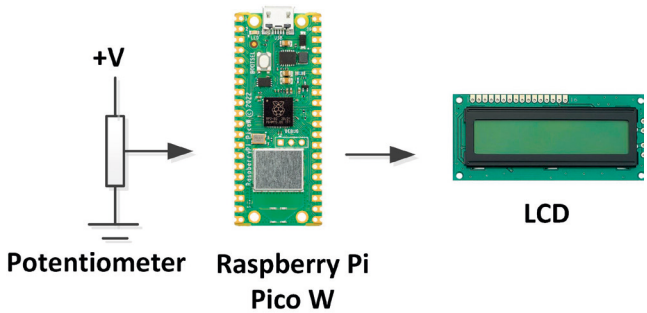


Рис.6.10: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рисунке 6.11. Плечо потенциометра подключено к каналу 0 АЦП (GP0). LCD подключается как в предыдущих проектах LCD. Выходной сигнал ШИМ доступен на GP16, пин 21).

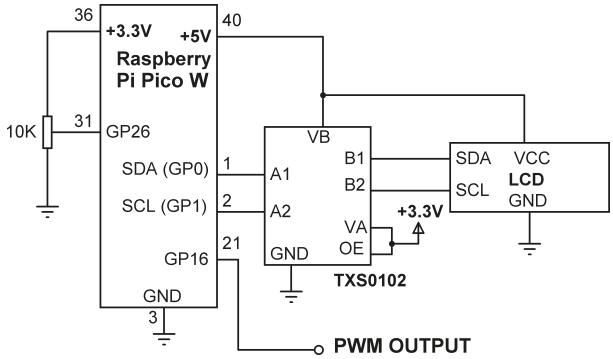


Рис.6.11: Принципиальная схема проекта.

Листинг программы: На рис. 6.12 показан листинг программы (Программа: **FreqGen**). В начале программы **Pot** назначается каналу 0 АЦП, ЖК-дисплей инициализируется, и в верхней строке ЖК-дисплея отображается сообщение **Frequency(Hz)** - Частота (Гц). Внутри цикла программы считывается значение **Pot**, которое используется для установки частоты сигнала ШИМ. Рабочий цикл установлен на 50%.

```
#------------------------------------------------------------
#               ГЕНЕРАТОР ЧАСТОТЫ
#               ====================
#
# В этом проекте потенциометр подключен к каналу 0 Pico.
# Также подключен ЖК-дисплей. Программа генерирует сигналы
# ШИМ различной частоты при перемещении
# ползунка потенциометра
#
# Автор: Доган Ибрагим
# Файл   : FreqGen.py
# Дата: октябрь 2022 г.
#------------------------------------------------------------
import machine
from machine import I2C
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
import utime
from machine import Pin, PWM, ADC

I2C_ADDR = 0x27                         # Адрес I2C ЖК
NRows = 2                               # Количество строк
NColumns = 16                           # Количество столбцов

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, NRows, NColumns)
```

```
Pot = ADC(0)                          # Pot на канале 0

ch = PWM(Pin(16))                     # ШИМ на GP16
ch.freq(100)                          # Частота по умолчанию
lcd.clear()

while True:                           # Делать всегда
    lcd.move_to(0, 0)
    lcd.putstr("Frequency(Hz)")
    frequency = Pot.read_u16()        # Чтение данных потенциометра
    lcd.move_to(0, 1)
    lcd.putstr(str(frequency))
    ch.duty_u16(32767)
    ch.freq(frequency)                # Изменить частоту
    utime.sleep(1)
    lcd.clear()
```

Рис. *6.12:* Программа **FreqGen**.

Частота отображается на ЖК-дисплее в следующем формате:

Строка 1: Frequency(Hz)
Строка 2: нннннн

6.9 Проект 6: Измерение частоты и скважности сигнала ШИМ

Описание: В этом проекте считываются частота и рабочий цикл сигнала ШИМ и отображается на экране Тонни.

Принципиальная схема: сигнал ШИМ, частота и скважность которого должны быть измерены, подается на порт **GP17** (вывод 22). Вы должны убедиться, что напряжение не превышает +3,3 В, иначе вы можете повредить входную схему вашего Pico.

Листинг программы: На рис. 6.13 показан листинг программы (Program:**MeasFreq**). Программа измеряет время метки (т. е. логической 1) и пробела (т. е. логического 0) формы входного сигнала ШИМ в микросекундах. Затем рабочий цикл и частота рассчитываются следующим образом:

Duty cycle (%) = 100 × Mark / (Mark + Space)
Frequency (kHz) = 1000 / (Mark + Space)

Скважность (%) = 100 × метка / (метка + пробел)
Частота (кГц) = 1000 / (метка + пробел)

```
#-------------------------------------------------------------
#                ИЗМЕРЕНИЕ ЧАСТОТЫ И СКВАЖНОСТИ
#                ================================
#
# В этом проекте к PIco применяется сигнал ШИМ.
# частота и скважность этого сигнала измеряются и
# отображается на экране Тонни.
#
# Автор: Доган Ибрагим
# Файл  : MeasFreq.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
from machine import Pin
import utime

PWMin = Pin(17, Pin.IN)                 # Вход сигнала ШИМ

while True:                             # Делать всегда
    while True:
        if PWMin.value() == 1:          # Подождать, пока 0
            break
    Tmr1Strt = utime.ticks_cpu()        # Запустить таймер 1

    while True:
        if PWMin.value() == 0:          # Подождать,пока 1
            break
    Tmr1End = utime.ticks_cpu()         # Конец

    while True:
        if PWMin.value() == 1:          #  Подождать, пока 0
            break

    Tmr2End = utime.ticks_cpu()         # Конец

    Mark = utime.ticks_diff(Tmr1End, Tmr1Strt)
    Space = utime.ticks_diff(Tmr2End, Tmr1End)
    duty = 100.0 * Mark / (Mark + Space)
    freqkHz = 1000.0 / (Mark + Space)
    print("Duty Cycle = %5.2f" % duty)
    print("Frequency (kHz) = ", freqkHz, "\n")
    utime.sleep(2)
```

Рис.*6.13:* Программа:**MeasFreq**.

Пример вывода программы показан на рис. 6.14.

```
Duty Cycle = 39.74
Frequency (kHz) =  1.20048

Duty Cycle = 35.05
Frequency (kHz) =  1.317523

Duty Cycle = 39.59
Frequency (kHz) =  1.203369
```

Рис.6.14: Пример вывода.

6.10 Проект 7: Создатель мелодий

Описание: Этот проект показывает, как тоны ШИМ-типа с разными частотами могут быть сгенерированы и отправлены на устройство с пассивным зуммером. Проект показывает, как простую мелодию Happy Birthday можно сыграть зуммером.

Цель: Цель этого проекта — показать, как можно генерировать различные тона для создания простой мелодии.

Блок-схема: Блок-схема проекта показана на рисунке 6.15.

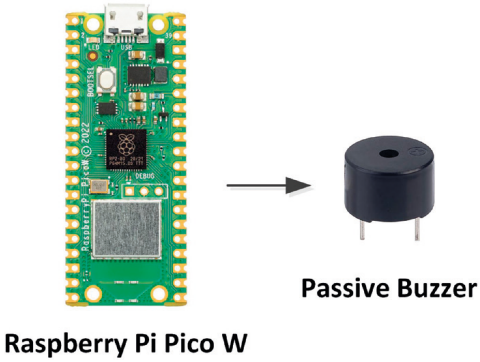


Рис.6.15: Блок-схема проекта.

Принципиальная схема: На рис. 6.16 показана принципиальная схема проекта. Пассивный зуммер подключен к порту GP0 (контакт 1) Raspberry Pi Pico W. Для повышения уровня напряжения зуммера используется транзисторный усилитель (это можно не делать, а при желании зуммер можно напрямую подключить к GP0. Однако, зуммер будет звучать не очень громко. В этом проекте можно использовать любой биполярный транзистор NPN. Вывод + зуммера может быть подключена либо к +3,3 В, либо к +5 В для более высокого выходного сигнала зуммера.

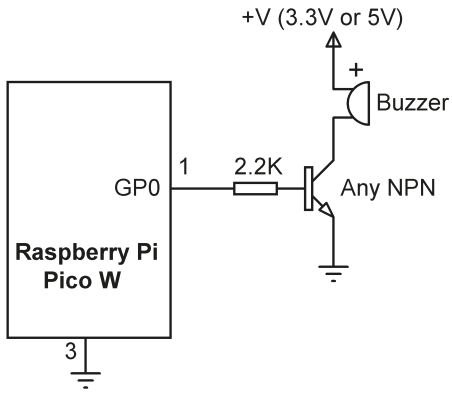


Рис.6.16: Принципиальная схема проекта.

Мелодии

При воспроизведении мелодии каждая нота воспроизводится в течение определенной длительности и с определенной частотой. Кроме того, между двумя последовательными нотами необходим определенный промежуток. Ниже приведены частоты музыкальных нот, начиная со среднего С (т.е. С4). Гармоника ноты получается удвоением частоты. Например, частота С5 составляет 2×262 = 524 Гц.

Нота	C4	C4#	D4	D4#	E4	F4	F4#	G4	G4#	A4	A4#	B4
Hz	261.63	277.18	293.66	311.13	329.63	349.23	370	392	415.3	440	466.16	493.88

Чтобы сыграть мелодию, нужно знать ее ноты. Каждая нота воспроизводится в течение определенной продолжительности, и между двумя последовательными нотами есть определенный промежуток времени. Следующее, что вам нужно, это знать, как генерировать звук с требуемой частотой и длительностью. В этом проекте вы будете генерировать классическую мелодию Happy Birthday, поэтому вам нужно знать ноты и их продолжительность. Они приведены в таблице ниже, где длительность указана в единицах по 400 миллисекунд (т. е. значения, указанные в таблице, следует умножить на 400, чтобы получить фактическую длительность в миллисекундах).

Note	C4	C4	D4	C4	F4	E4	C4	C4	D4	C4	G4	F4	C4	C4	C5	A4	F4	E4	D4	A4#	A4#	A4	F4	G4	F4
Duration	1	1	2	2	2	3	1	1	2	2	2	3	1	1	2	2	2	2	2	1	1	2	2	2	4

Листинг программы: Листинг программы (program:**Melody**) показан на рис. 6.17. Частота и продолжительность мелодии хранятся в двух массивах, называемых **frequency** - частота и **duration** - продолжительность соответственно. Перед основным программным циклом длительность каждого тона вычисляется и сохраняется в массиве **Durations**, так что основному программному циклу не нужно тратить время на эти вычисления. Внутри цикла программы генерируются частоты мелодии с требуемой длительностью. Обратите внимание, что вывод тона останавливается установкой рабочего цикла на 0. Между каждым тоном вводится небольшая задержка (100 мс). Мелодия повторяется через 3 секунды задержки. Вы можете попробовать более высокие гармоники нот для более четкого звучания. Например, на рис. 6.16 частоты умножаются на 2 для воспроизведения вторых гармоник.

```
#-----------------------------------------------------------------------
#              MELODY MAKER - PLAY HAPPY BIRTHDAY
#              ==================================
#
# В этом проекте зуммер подключен к GP0,который настроен как выход PWM.
# В программе звучит мелодия
# Happy Birthday - С Днем Рождения
#
# Автор: Доган Ибрагим
# Файл   : Melody.py
# Дата: октябрь 2022 г.
#-----------------------------------------------------------------------
from machine import Pin, PWM
import utime

ch = PWM(Pin(0))                                    # Выход ШИМ на GP0
MaxNotes = 25
Durations = [0]*MaxNotes
#
# Частота мелодии
#
frequency = [262,262,294,262,349,330,262,262,294,262,
                          392,349,262,262,524,440,349,330,294,466,
                          466,440,349,392,349]
#
# Длительность частоты
#
duration = [1,1,2,2,2,3,1,1,2,2,2,3,1,1,2,2,2,2,
                         2,1,1,2,2,2,3]

for k in range(MaxNotes):
    Durations[k] = 400 * duration[k]

while True:                                         #  Делать всегда
    for k in range(MaxNotes):                       # Сделать для всех нот
        ch.duty_u16(32767)                          # Рабочий цикл
        ch.freq(2*frequency[k])                     # Воспроизведение второй гармоники
        utime.sleep_ms(Durations[k])                #  Продолжительность
        utime.sleep_ms(100)                         # Подождать
    ch.duty_u16(0)                                  # Остановить воспроизведение
    utime.sleep(3)                                  # Остановить на 3 секунды
```

Рис.*6.17:* Программа: **Melody**.

Предложения по дополнительной работе

Измените программу, приведенную на рис. 6.17, изменив продолжительность между нотами, и посмотрите, как это отразится на мелодии. Как сделать так, чтобы мелодия звучала быстрее? Кроме того, замените зуммер усилителем звука и динамиком для более качественного и в то же время более громкого звучания.

Глава 7 • TFT-дисплеи

7.1 Обзор

Тонкопленочный транзисторный дисплей — это тип жидкокристаллического дисплея, в котором используется технология тонкопленочных транзисторов для улучшения таких качеств, как контрастность и адресуемость. TFT также называют жидкокристаллической технологией с активной матрицей. В технологии TFT для управления каждым отдельным пикселем используется отдельный транзистор, что обеспечивает гораздо более быстрое время отклика. TFT в ЖК-дисплее управляет отдельными пикселями на дисплее, устанавливая уровень электрического поля на трех жидкокристаллических конденсаторах (красный, зеленый, синий) в пикселе, чтобы контролировать поляризацию кристаллического материала, которая определяет количество света, которое достигает цветного фильтра от подсветки. TFT-дисплеи доступны во многих различных физических размерах и с разным количеством пикселей.

В этой главе будет использоваться 1,8-дюймовый TFT-дисплей в проектах Raspberry Pi Pico W.

7.2 TFT-дисплей

TFT-дисплей, используемый в проектах этой главы, показан на рис. 7.1. Этот дисплей имеет следующие характеристики:

Размер:	1.8 inch
Разрешение:	128 × 160
Драйвер IC:	ST7735
Интерфейс:	4-проводной SPI
SD:	встроенный адаптер SD-карты

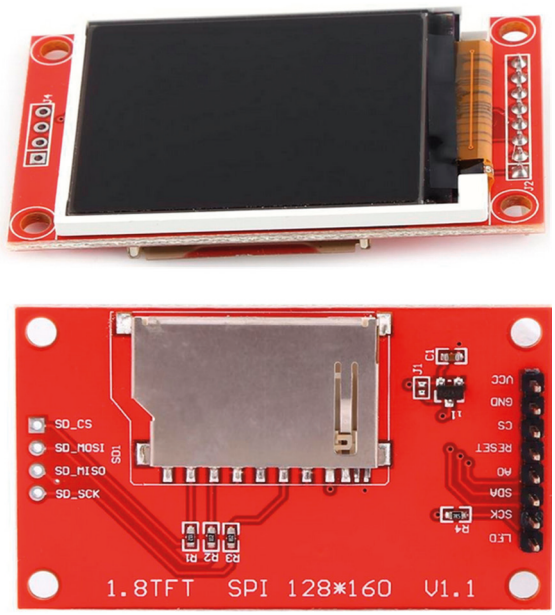


Рис. *7.1: TFT*-дисплей.

Дисплей имеет 8-битный разъем с одной стороны для логики дисплея и 4-битный разъем с другой стороны для интерфейса SD-карты. Определения пинов следующие:

8-контактный разъем	
LED	Управление подсветкой светодиодного дисплея
SCK	Отображение синхротактов SPI
SDA	Отображение данных SPI
A0	Выбор адреса дисплея (данные/команда, DC)
RESET	Сброс дисплея (RST)
CS	Выбор чипа дисплея
GND	Земля источника питания
VCC	+ источника питания

4-контактный разъем	
SD-CS	Выбор чипа SD-карты
SD-MOSI	SD card пин MOSI
SD-MISO	SD card пин MISO
SD-SCK	SD card пин SPI Clock

7.3 Подключение TFT-дисплея к Raspberry Pi Pico W

В проектах из этой главы TFT-дисплей подключается к Pico следующим образом (см. Рис.7.2), где для связи используется SPI0:

TFT display	**Pico port pin**
SCK	GP2 – SPI0 SCK
SDA (or MOSI)	GP3 – SPI0 TX
A0	GP15
RESET	GP14
CS	GP5 - SPI0 CSn
GND	GND
VCC	+3.3V
LED	+3.3V

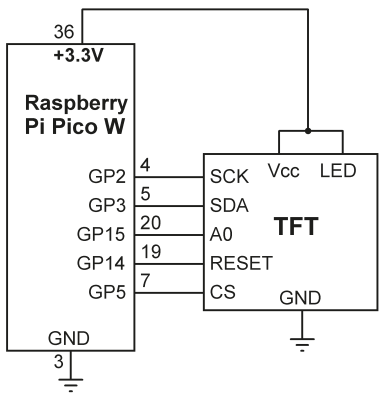


Рис.7.2: TFT-дисплей – интерфейс Pico.

7.4 Библиотека драйверов TFT-дисплея ST7735

Прежде чем использовать 1,8-дюймовый TFT-дисплей с Raspberry Pi Pico, вы должны загрузить в Pico драйвер дисплея ST7735. Шаги следующие:

- Copy file **ST7735.py** from the web site of the book to your Raspberry Pi Pico W with the name **ST7735.py**.
- Copy file **sysfont.py** from the web site of the book to your Raspberry Pi Pico W with the name **sysfont.py**.

Вы также можете найти файлы **ST7735.py** и **sysfont.py** на следующем веб-сайте **Makerfabs** с некоторыми другими полезными файлами, но вам придется изменить размер экрана на 128×160, TF-TRGB на 0x08 и TFTBGR на 0x00 (см. также книгу: **Raspberry Pi Pico Experimenting Kit**, продается Elektor):

https://github.com/Makerfabs/Pico_Primer_Kit/tree/main/example/lib

На рис. 7.3 показаны координаты пикселей используемого TFT-дисплея.

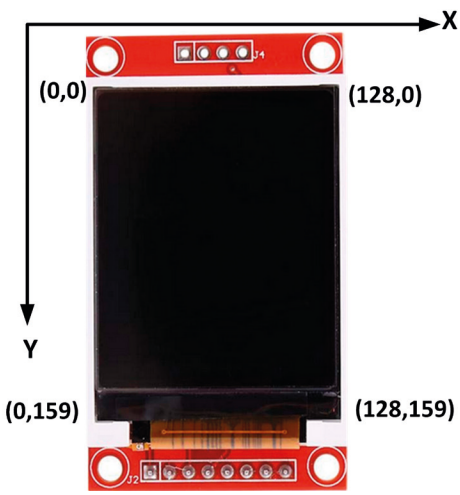


Figure 7.3: Co-ordinates of the display used.

7.4.1 Рисование фигур

Обратите внимание, что все значения указаны в пикселях. Допустимые цвета:

```
TFT.BLACK      - ЧЕРНЫЙ
TFT.RED        -  КРАСНЫЙ
TFT.MAROON  - ТЕМНО-БОРДОВЫЙ
TFT.ORANGE  - ОРАНЖЕВЫЙ
TFT.GOLD      - ЗОЛОТОЙ
TFT.GREEN    - ЗЕЛЕНЫЙ
TFT.FOREST  - ТЕМНО-ЗЕЛЕНЫЙ
TFT.BLUE      - СИНИЙ
```

TFT.NAVY - ТЕМНО-СИНИЙ
TFT.CYAN - ГОЛУБОЙ
TFT.YELLOW - ЖЕЛТЫЙ
TFT.PURPLE - ПУРПУРНЫЙ
TFT.WHITE - БЕЛЫЙ
TFT.GRAY - СЕРЫЙ

Полый прямоугольник
Рисует пустой прямоугольник, где левые верхние координаты
(x вверху слева, y вверху слева) и указаны ширина, высота и цвет.

```
rect((x top left, y top left), (width, height), colour)
```

Закрашенный прямоугольник
Рисует закрашенный прямоугольник с координатами верхнего левого угла
(x вверху слева, y вверху слева), шириной, высотой и цветом.

```
fillrect((x top left, y top left), (width, height), colour)
```

Полый круг
Рисует полый круг, в котором указаны координаты центра x и y, радиус и цвет.

```
circle((x center, y center), radius, colour)
```

Закрашенный круг
Рисует закрашенный круг с заданными координатами центра x и y, радиусом и цветом.

```
fillcircle((x center, y center), radius, colour)
```

Линия
Рисует линию от начальной точки (x начало y начало) до конечной точки
(x конец, y конец) и указанный цвет

```
line((x start, y start), (x end, y end), colour)
```

Вертикальная линия
Рисует вертикальную линию от начальной точки (x start y start) и заданной
длины и цвета.

```
vline((x start y start), length, colour)
```

Горизонтальная линия
Рисует горизонтальную линию от начальной точки (x start y start) и заданной длины
и цвета.

hline((x start, y start), length, colour)

Пример 1
Нарисуйте два концентрических прямоугольника красного цвета по краям дисплея со следующими координатами:

Полый красный прямоугольник с верхним левым углом в точке (0, 0), ширина = 120, высота = 155
Полый красный прямоугольник с верхним левым углом в (10, 10), ширина = 110, высота = 145

Решение 1
Листинг программы показан на рис. 7.4 (Программа: **rectangles)**.

```
#-------------------------------------------------------
#          ДВА КОНЦЕНТРИЧЕСКИХ ПРЯМОУГОЛЬНИКА
#          ======================================
#
# Эта программа рисует два концентрических прямоугольника красного цвета
#
#  Автор: Доган Ибрагим
#  Файл   : rectangles.py
#  Дата: октябрь 2022 г.
#-------------------------------------------------------
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin

spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)

tft.fill(TFT.BLACK)                              # Очистить экран

tft.rect((10, 10), (110, 140), TFT.RED)          # Полый прямоугольник
tft.rect((20, 20), (90, 120), TFT.RED)           # Полый прямоугольник
```

Рис.7.4: программа rectangles.

На рис.7.5 показан дисплей с двумя концентрическими прямоугольниками красного цвета.

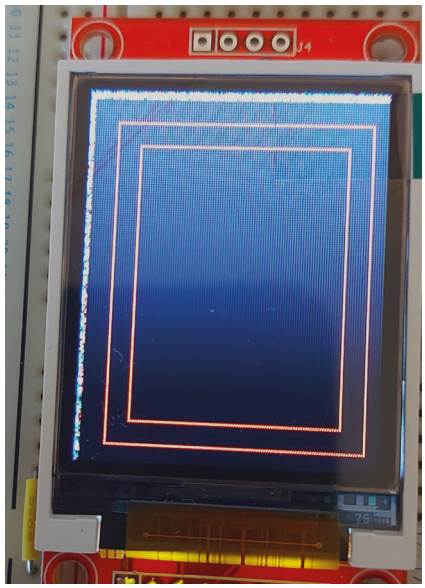


Рисунок 7.5: Дисплей.

Пример 2 – Отображение фигур
Нарисуйте следующие фигуры на TFT-дисплее:

Полый красный прямоугольник с верхним левым углом в точке (0, 0)
Закрашенный синий прямоугольник с верхним левым углом в точке (60, 0)
Полый красный круг с центром в (20, 70) с радиусом = 15
Закрашенный зеленый круг с центром в (60, 70) с рад
Желтая линия от (20, 90) до (60, 100)
Голубая горизонтальная линия, начинающаяся с (20, 110) и имеющая длину 80
Голубая вертикальная линия, начинающаяся с (110, 60) и имеющая длину 70

Решение 2
На рис. 7.6 показан листинг программы (Программа: **TFTBasic**). В начале в программу импортируются необходимые модули и определяется интерфейс SPI TFT-дисплея. Фон экрана устанавливается на черный цвет (т. е. экран очищается), а затем программа рисует необходимые фигуры.

```
#------------------------------------------------------
#               РАЗЛИЧНЫЕ ФИГУРЫ
#               ==================
#
#  Эта программа рисует различные фигуры на TFT-дисплее
#
# Автор: Доган Ибрагим
# Файл   : TFTBasic.py
# Дата: октябрь 2022 г.
```

```
#-------------------------------------------------------
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin

spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)

tft.fill(TFT.BLACK)                              # Очистить экран

tft.rect((0, 0), (50, 50), TFT.RED)              # Полый прямоугольник
tft.fillrect((60, 0), (50, 50), TFT.BLUE)        # Заполненный прямоугольник
tft.circle((20, 70), 15, TFT.RED)                # Полый круг
tft.fillcircle((60, 70), 15, TFT.GREEN)          # Закрашенный круг
tft.line((20, 90), (60, 100), TFT.YELLOW)        # Линия
tft.hline((20, 110), 80, TFT.CYAN)               # Горизонтальная линия
tft.vline((110, 60), 70, TFT.CYAN)               # Вертикальная линия
```

Рис.*7.6:* Программа:**TFTBasic**.

На рис.7.7 показаны фигуры, нарисованные на TFT-дисплее.

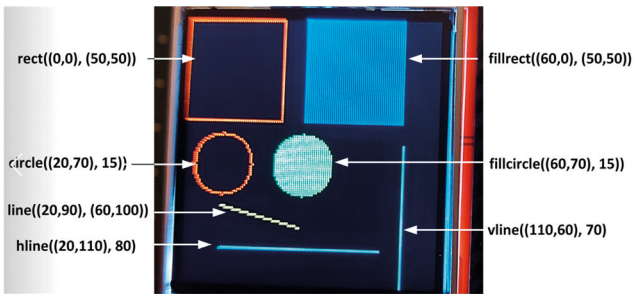


Рис.7.7: Фигуры на TFT-дисплее.

7.4.2 Отображение текста

Для отображения текста на TFT-дисплее требуется начальная (x, y) координата текста,текст сообщения, цвет и шрифт текста, а также дополнительные свойства, такие как обтеканиетекст. Функция text() используется для отображения текста.

Пример 3. Отображение текста

Фон в этом примере установлен белый. В этом примере требуется отобразить следующий текст:

Текст: PICO **Начальная координата**: (10,10) **Цвет**: Red **Шрифт**: 2 **Без обтекания**
Текст: PICO **Начальная координата**: (10,30) **Цвет**: Green **Шрифт**: 3 **Без обтекания**
Текст: PICO **Начальная координата**: (10, 55) **Цвет**: Cyan **Шрифт**: 4 **Без обтекания**
Текст: PICO **Начальная координата**: (10, 90) **Цвет**: Blue **Шрифт**: 5 **Без обтекания**

Решение 3

На рис. 7.8 показан список требуемой программы (Программа:**TFTText**.

```
#-------------------------------------------------------
#               ОТОБРАЖЕНИЕ ТЕКСТА
#               ====================
#
# Эта программа выводит текст на TFT дисплей
#
# Автор: Доган Ибрагим
# Файл   : TFTText.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin

spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)

tft.fill(TFT.WHITE)                    # Очистить экран
tft.text((10, 10), "PICO", TFT.RED, sysfont, 2, nowrap=True)
tft.text((10, 30), "PICO", TFT.GREEN, sysfont, 3, nowrap=True)
tft.text((10, 55), "PICO", TFT.CYAN, sysfont, 4, nowrap=True)
tft.text((10, 90), "PICO", TFT.BLUE, sysfont, 5, nowrap=True)
```

Рис.*7.8:* Программа:**TFTText**

На рис.7.9 показаны тексты, отображаемые на TFT-дисплее.

Рис.7.9: Тексты на TFT-дисплее.

7.4.3 Другие функции TFT

Некоторые другие полезные функции TFT:

вращение: Вы можете вращать свой рисунок. Обратите внимание, что это не повернет то, что вы уже нарисовали, но изменит систему координат для любого нового рисунка. Функция вращения может принимать только 4 значения: 0, 1, 2 и 3. Вы можете поворачивать только на 0º (значение 0), 90º (значение 1), 180º (значение 2) или 270º (значение 3) градусов. При вращении исходная точка (0, 0) меняется.

invertcolor: инвертировать цвет фона. 1, чтобы установить черный цвет, и 0, чтобы установить белый

on: включить дисплей (True) или выключить (False)

pixel((x, y), colour):нарисовать пиксель по заданной координате с указанным цветом

заливка: заливка фона указанным цветом

Пример 4 – Другие функции TFT

В этом примере требуется выполнить следующее:

Установить фон дисплея на белый
Прокрутите отображение, используя значения поворота 1,2,3,0

Решение 4

На рис.7.10 показан листинг программы (Программа: TFTOther).

```
#-------------------------------------------------------
#              ДРУГИЕ ФУНКЦИИ TFT
#              ==================
#
# Эта программа показывает, как работает функция вращения
#
# Автор: Доган Ибрагим
# Файл   : TFTOther.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin
import utime

spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)

tft.fill(TFT.WHITE)                              # Очистить экран

tft.text((10, 30), "PICO", TFT.RED, sysfont, 2, nowrap=True)
utime.sleep(2)

tft.rotation(1)
tft.text((10, 30), "PICO1", TFT.RED, sysfont, 2, nowrap=True)
utime.sleep(2)

tft.rotation(2)
tft.text((10, 50), "PICO2", TFT.RED, sysfont, 2, nowrap=True)
utime.sleep(2)

tft.rotation(3)
tft.text((10, 30), "PICO3", TFT.RED, sysfont, 2, nowrap=True)
utime.sleep(2)

tft.rotation(0)
tft.text((10, 30), "PICO0", TFT.RED, sysfont, 2, nowrap=True)
utime.sleep(2)
```

Рис. *7.10:* Программа: **TFTOther**.

На рис.7.11 показан дисплей во время работы программы. Обратите внимание, что на дисплее отображается только повернутый текст.

Рис.7.11: Отображение повернутого текста.

7.5 Проект 1: Счетчик секунд

Описание: Это проект счетчика секунд. Нажатие кнопки запускает отсчет. Счетчик отображается на TFT-дисплее.

Цель: Цель этого проекта — показать, как текстовые и числовые данные могут отображаться на TFT-дисплее.

Блок-схема: На рис. 7.12 показана блок-схема проекта.

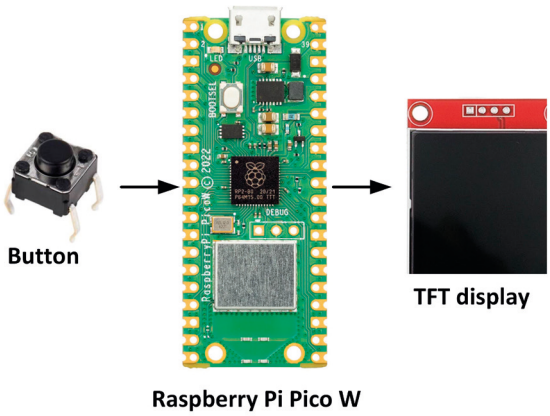


Рис.*7.12:* Блок-схема проекта.

Принципиальная схема: Принципиальная схема показана на рис. 7.13, где кнопка подключена GP16. TFT-дисплей подключается, как показано на рис.7.2.

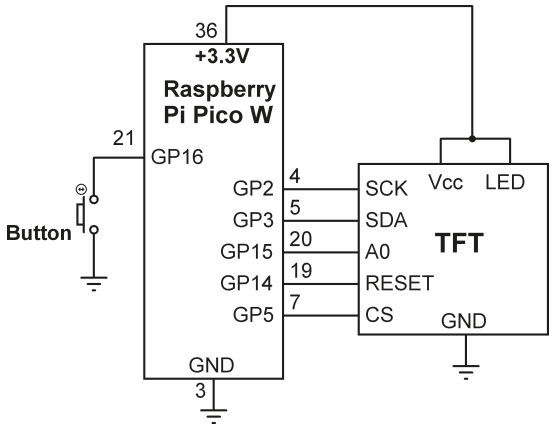


Рис.7.13: Принципиальная схема проекта.

Листинг программы: Рис.7.14 представляет листинг программы (Программа: **TFTCounter**). В начале программы используемые модули импортируются в программу. TFT инициализируется, и кнопка назначается порту **GP16**. Затем программа рисует прямоугольник и отображает заголовок **Seconds Counter** внутри этого прямоугольника. Затем программа ждет, пока кнопка не будет нажата. Подсчет начинается, как только нажимается кнопка, при этом переменная **Count** увеличивается каждую секунду. Счетчик секунд отображается на TFT-дисплее.

```
#-------------------------------------------------------
#                СЧЕТЧИК СЕКУНД
#                ==============
#
# Это программа счетчика секунд. Программа считает каждую
# секунду. Подсчет начинается при нажатии кнопки на GP16.
# is pressed
#
# Автор: Доган Ибрагим
# Файл   : TFTCounter.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin
import utime

spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

Button=Pin(16,Pin.IN, Pin.PULL_UP)      # Кнопка на GP16
```

```
tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)

tft.fill(TFT.BLACK)                    # Очистить экран
Count = 0

#
# Показать заголовок
#
tft.rect((10, 10), (110, 100), TFT.WHITE)
tft.text((15, 40), "SECONDS COUNTER", TFT.WHITE, sysfont,
1.1, nowrap=True)

while Button.value() == 1:             # Ожидание нажатия кнопки
   pass

#
# Увеличение счетчика каждую секунду и отображение на TFT
#
while True:
  tft.text((20, 60), "Count:{:d}".format(Count), TFT.WHITE,
  sysfont, 1, nowrap=True)
  Count = Count + 1
  utime.sleep(1)
```

Рис. *7.14:* Программа: **TFTCounter**.

На рис.7.15 показан пример экрана.

Рис. *7.15:* Пример экрана.

7.6 Проект 2: Таймер реакции

Описание: Это проект таймера реакции. Светодиод подключен к Pico. Этот светодиод включается в произвольное время. Пользователь должен нажать кнопку, как только он/она увидит, что светодиод включен. Время, прошедшее между включением светодиода и нажатием кнопки, считается временем реакции, и это время отображается на TFT-дисплее.

Цель: Цель этого проекта — показать, как можно измерить прошедшее время, как настроить внешнее прерывание и как время реакции можно отобразить на TFT-дисплее.

Блок-схема: На рис. 7.16 показана блок-схема проекта.

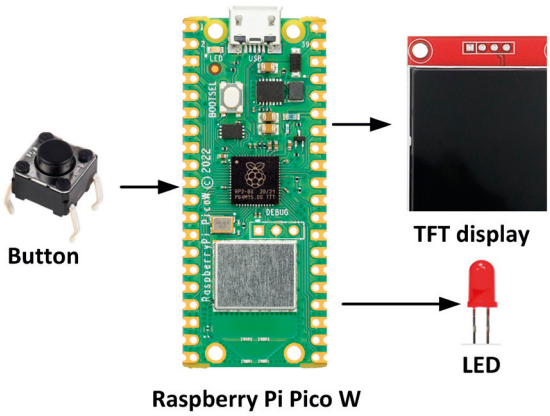


Рис.7.16: Блок-схема проекта.

Принципиальная схема: Принципиальная схема показана на рис.7.17. Кнопка и светодиод подключены к GP16 и GP17 PICO соответственно.

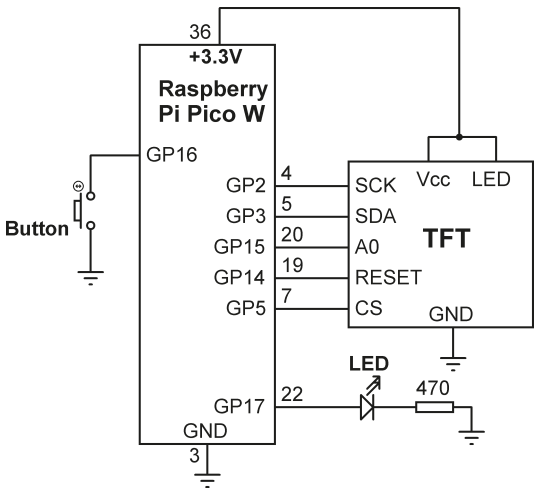


Рис.7.17: Принципиальная схема проекта.

Листинг программы:Рис.7.18 показывает листинг программы (Программа: **TFTReaction**). В начале программы используемые модули импортируются в программу. На TFT отображается заголовок REACTION TIMER.

Основная программа работает в цикле **while**. Внутри этого цикла программа ожидает произвольное время, включает желтый светодиод и сохраняет текущее время процессора в переменной **TmrStart**. В то же время разрешены внешние прерывания, чтобы прерывания можно было распознать при нажатии кнопки. Функция **MyButton** активируется при нажатии кнопки. Внутри этой функции время процессора считывается и сохраняется в переменной **TmrEnd**. Прошедшее время рассчитывается и является переменной **ReactionTime**, а затем отображается на TFT. Описанный выше процесс повторяется после 3-секундной задержки.

```
#-----------------------------------------------------------------
#                                ТАЙМЕР  РЕАКЦИИ
#                                ================
#
# Это программа таймера реакции, которая измеряет реакцию пользователя
# и отображает её на TFT-дисплее в мс.
# Для быстрой реакции пользователь должен нажать кнопку,
# как только загорится светодиод
#
# Автор: Доган Ибрагим
# Файл   : TFTReaction.py
# Дата: октябрь 2022 г.
#-----------------------------------------------------------------
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin
import utime
import random

spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

#
#  Конфигурация кнопки и светодиода
#
Button = Pin(16, Pin.IN)                        # Нажмите кнопку
LED = Pin(17, Pin.OUT)                          # LED
flag = 0

tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)
```

```
#
# Это процедура обслуживания прерывания.
# Программа переходит сюда как только нажимается кнопка
#
def MyButton(pin):
    global flag
    Button.irq(handler = None)
    LED.value(1)                                        # LED не горит
    TmrEnd = utime.ticks_ms()                           # Конец времени
    ReactionTime = utime.ticks_diff(TmrEnd, TmrStart)
    flag = 1
    tft.text((20, 60), "Time:{:.2f} ms".format(ReactionTime), TFT.WHITE,
    sysfont, 1, nowrap=True)
    utime.sleep(3)

#
# Показать заголовок
#
def Heading():
   tft.fill(TFT.BLACK)                                  # Очистить экран
   tft.rect((10, 10), (115, 100), TFT.WHITE)
   tft.text((19, 40), "REACTION TIMER", TFT.WHITE, sysfont,
   1.1, nowrap=True)

#
# Запуск ГЛАВНОЙ программы. Создайте случайную задержку, а затем включите LED
#
LED.value(1)                                           # LED отключен

while True:                                            # ДЕЛАТЬ ВСЕГДА
    Heading()
    flag = 0
    rnd = random.randint(3, 10)                        # Случайное целое число
    utime.sleep(rnd)                                   # случайная задержка
    LED.value(0)                                       # LED горит
    TmrStart = utime.ticks_ms()
    Button.irq(handler=MyButton, trigger = Pin.IRQ_FALLING)
    while flag == 0:
       pass
```

Рис.*7.18:* Программа:**TFTReaction**.

На рис. 7.19 показан пример экрана из программы.

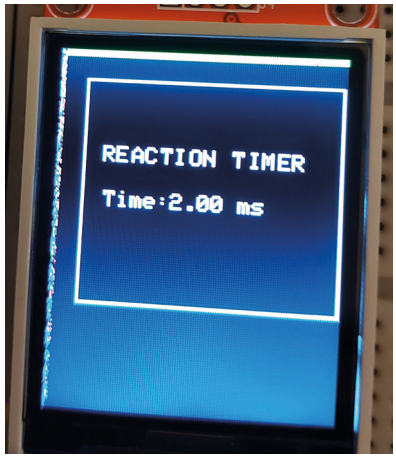


Рис.*7.19:* Пример экрана.

7.7 Проект 3: Температура и влажность – дисплей на TFT

Описание: Эта программа считывает температуру и влажность окружающей среды с модуля датчика DHT11 и отображает их на TFT-экране каждые 5 секунд.

Цель: Эта программа показывает, как сенсорный модуль DHT11 можно использовать с Pico и как показания температуры и влажности могут отображаться на TFT-дисплее.

Блок-схема: DHT11 — недорогой модуль датчика влажности и температуры (рис.7.20). Емкостный датчик измеряет влажность, а термистор измеряет температуру. Основные характеристики DHT11:

- Бюджетный
- Питание от 3 до 5 В и ввод/вывод
- Макс. потребление тока 2,5 мА во время преобразования (при запросе данных)
- Показания влажности 20-80% с точностью 5%
- Показания температуры 0–50 °C с точностью ±2 °C
- Частота дискретизации не более 1 Гц (раз в секунду)

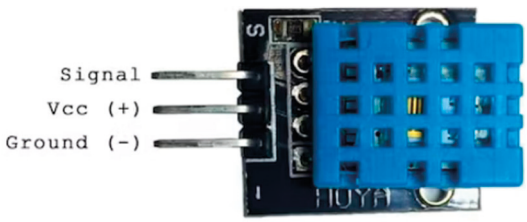


Рис.*7.20:* Датчик температуры и влажности *DHT11.*

Читатели могут предпочесть использовать совместимый DHT22, который более точен, чем DHT11.

.

Блок-схема: На рис. 7.21 показана блок-схема.

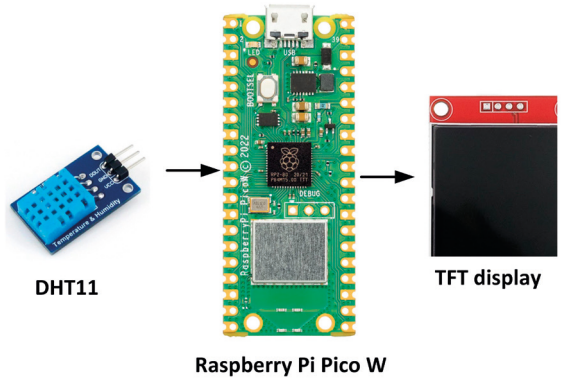


Рис.7.21: Блок-схема проекта.

Принципиальная схема: На рис.7.22 показана принципиальная схема. DHT11 подключен к GP16.

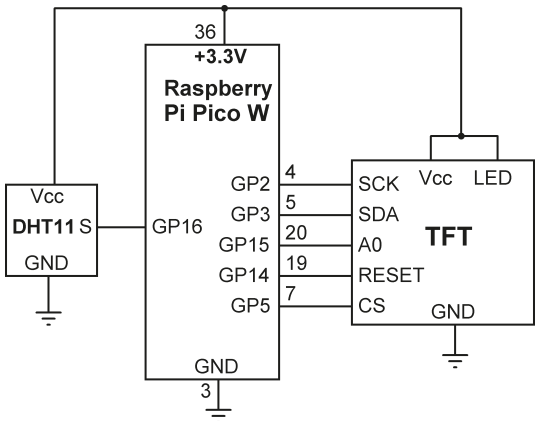


Рис.7.22: Принципиальная схема.

Листинг программы: Перед написанием программы вы должны скопировать драйвер **DHT11/22** на свой Raspberry Pi Pico W. Он указан в папке с кодами под названием **dht.py**. Просто скопируйте этот файл на свой Pico с тем же именем).

На рис.7.23 показан листинг программы (Программа:**TFTDHT11**). После инициализации шины SPI программа запускается в цикле с использованием оператора while. Внутри этого цикла отображается курс, температура и влажность считываются с модуля DHT11 и отображаются на TFT-дисплее. Этот процесс повторяется после 5-секундной задержки.

```
#-------------------------------------------------------
#              ТЕМПЕРАТУРА И ВЛАЖНОСТЬ НА TFT
#              ==============================
#
# В этой программе используется датчик температуры и влажности
# DHT11 и показания отображаются на TFT
#
# Автор: Доган Ибрагим
# Файл   : TFTDHT11.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------
import utime
from dht import DHT11, InvalidChecksum
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin

#
# Инициализировать SPI
#
spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)

#
# Показать заголовок
#
def Heading():
   tft.fill(TFT.WHITE)                                  # Очистить экран
   tft.rect((5, 10), (100, 120), TFT.RED)
   tft.text((15, 40), "TEMP AND HUM", TFT.BLUE, sysfont,
   1.1, nowrap=True)
   tft.hline((15,51),85, TFT.RED)

#
# Считывайте температуру и влажность окружающей среды и
# отображайте на TFT-дисплее каждые 5 секунд
#
while True:
    pin = Pin(16, Pin.OUT, Pin.PULL_DOWN)
    sensor = DHT11(pin)
    t  = (sensor.temperature)
    h = (sensor.humidity)
```

```
    Heading()

    tft.text((15, 60), "Temp:{:.2f}C".format(t), TFT.BLUE,
    sysfont, 1.1, nowrap=True)

    tft.text((20, 75), "Hum:{:.2f}%".format(h), TFT.BLUE,
    sysfont, 1.1, nowrap=True)

    utime.sleep(5)
```

Рис.*7.23:* Программа:**TFTDHT11**.

На рис.7.24 показан пример отображения температуры и влажности на TFT.

Читатели должны заметить, что DHT11/22 требует точной синхронизации, и время от времени программа может аварийно завершать работу с ошибками контрольной суммы. В следующем проекте вы увидите, как перехватывать ошибки контрольной суммы и продолжать считывать значения температуры и влажности.

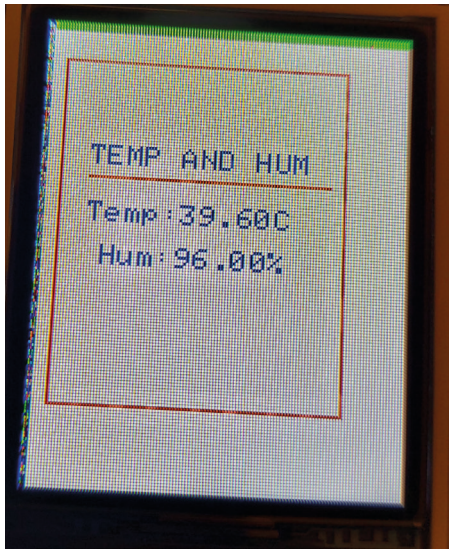


Рис.7.24: Пример экрана.

7.8 Проект 4: Минимальная/максимальная температура и влажность – Отображение на TFT

Описание: Эта программа похожа на предыдущую, но здесь минимальная и максимальная температуры с момента запуска программы отображаются на TFT-дисплее вместе с текущей температурой и влажностью.

Цель: Эта программа показывает, как сенсорный модуль DHT11 можно использовать с Pico и как показания температуры и влажности могут отображаться на TFT-дисплее.

Блок-схема и принципиальная схема проекта такие же, как на рис. 7.22 и рис. 7.22.

Листинг программы: В начале программы в программу импортируются все необходимые модули и инициализируется ЖКИ. Внутри основного цикла программы вызывается функция **Heading()** для отображения фоновых полей, в которых будут отображаться минимальные, максимальные, текущие значения температуры и влажности. Затем основная программа считывает температуру и влажность с DHT11 и отображает их в полях, как показано на рис. 7.26. Текущая температура отображается в средней части дисплея, влажность – в правой части, а минимальная и максимальная температуры отображаются внутри нижнего левого и нижнего правого боковых полей. Обратите внимание, что ошибки контрольной суммы перехватываются с помощью инструкции try-except. Сообщение failed отображается на экране Thonny, и программа продолжает работу через 3 секунды после обнаружения ошибки контрольной суммы. Программа продолжается через 5 секунд, если ошибок не обнаружено.

```
#------------------------------------------------------------
#                               ПОГОДА НА TFT
#                               ==============
#
# В этой программе используется датчик температуры и
# влажности DHT11, а минимальные, максимальные и текущие значения
# температуры и влажности отображаются на TFT-дисплее.
#
# Автор: Доган Ибрагим
# Файл  : weather.py
# Дата: октябрь 2022 г.
#------------------------------------------------------------
import utime
from dht import DHT11, InvalidChecksum
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin

#
# Инициализировать SPI
#
spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)

#
# Показать заголовок
```

```
def Heading():
   tft.fill(TFT.WHITE)                          # Очистить экран
   tft.rect((5, 5), (115, 145), TFT.RED)
   tft.hline((5,115), 113, TFT.RED)
   tft.vline((63, 115), 36, TFT.BLUE)
   tft.vline((75, 5), 112, TFT.BLUE)
   tft.text((85, 10), "HUM", TFT.BLUE, sysfont,
   1.1, nowrap=True)

#
# Считывайте температуру и влажность окружающей среды и
# отображайте на TFT-дисплее каждые 5 секунд.
#
first = 1

while True:
  try:
    pin = Pin(16, Pin.OUT, Pin.PULL_DOWN)
    sensor = DHT11(pin)                         # Чтение с DHT11
    t  = (sensor.temperature)                   # Температура
    h = (sensor.humidity)                       # Влажность
    Heading()                                   Показать заголовок

    if first == 1:                              # мин. и макс.
      minT = t
      maxT = t
      first = 0

    if t > maxT:
       maxT = t

    if t < minT:
       minT = t

    tft.text((9, 60), "{:.2f}C".format(t), TFT.MAROON,
    sysfont, 2, nowrap=True)

    tft.text((79, 50), "{:.1f}%".format(h), TFT.BLUE,
    sysfont, 1.1, nowrap=True)

    tft.text((15, 127), "{:.2f}C".format(minT), TFT.BLUE,
    sysfont, 1.1, nowrap=True)

    tft.text((75, 127), "{:.2f}C".format(maxT), TFT.MAROON,
    sysfont, 1.1, nowrap=True)
```

```
    utime.sleep(5)

  except:
    print("failed")
    utime.sleep(3)
```

Рис. *7.25:* Программа: **weather**.

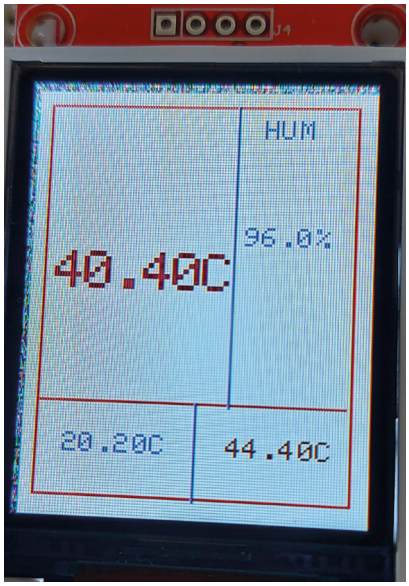


Рис. 7.26: Пример экрана.

7.9 Проект 5: Установка желаемой температуры с помощью кнопок и TFT-дисплея

Описание: Это проект управления температурой ON/OFF - ВКЛ/ВЫКЛ, в котором желаемый SetTempis устанавливается с помощью трех кнопок с названиями UP,DOWN и START. При запуске программы пользователю предлагается ввести заданную температуру. Нажатие UP - ВВЕРХ увеличивает SetPoint. Аналогичным образом, нажатие DOWN - ВНИЗ уменьшает температуру. Когда пользователь удовлетворен желаемой уставкой, он должен нажать кнопку DTART - СТАРТ, чтобы запустить регулятор температуры.

Цель: цель этого проекта — показать, как можно спроектировать систему двухпозиционного регулятора температуры с использованием недорогого модуля датчика температуры, кнопок и TFT-дисплея с Raspberry Pi Pico W.

Блок-схема: Блок-схема проекта показана на рис. 7.27.

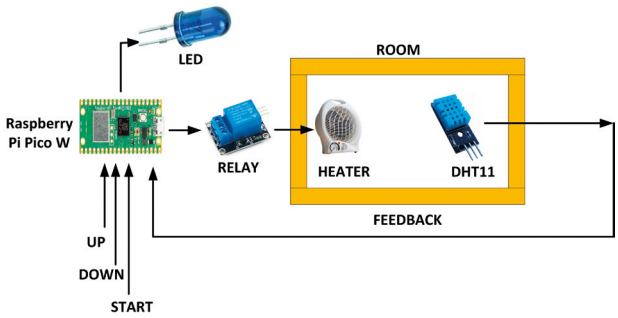


Рис.7.27: Блок-схема проекта.

Принципиальная схема: На рис. 7.28 показана принципиальная схема. Кнопки UP, DOWN и START подключены к GP18, GP19 и GP20 соответственно. Светодиод и РЕЛЕ подключены к GP21 и GP22 соответственно.

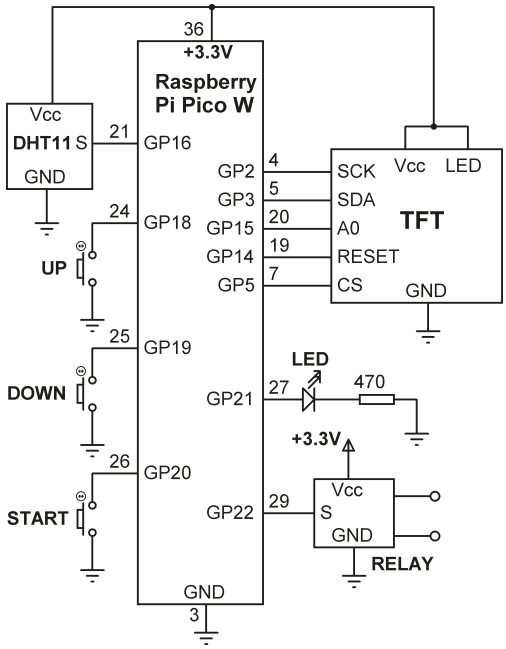


Рис.7.28: Принципиальная схема.

Листинг программы: Листинг программы (Программа:**ONOFFSET**) показанный на рис. 7.29, подобен листингу, приведенному на рис. 7.25. Здесь дополнительно настраиваются три кнопки, которые используются для считывания заданной температуры **SetTemp**. При запуске программы вызывается функция **Desired**. Эта функция отображает заголовок и ожидает нажатия кнопки. При нажатии UP значение **SetTemp** по умолчанию увеличивается на единицу, при нажатии **DOWN** значение **SetTemp** уменьшается на единицу. Если, с другой стороны, нажата кнопка START, то предполагается, что отображаемая **SetTemp** является желаемым значением, и начинается действие регулятора температуры.

Читатели должны заметить, **что** DHT11/22 **не совсем точны**.

```
#-------------------------------------------------------------
#      УПРАВЛЕНИЕ ТЕМПЕРАТУРОЙ  - НАСТРОЙКА ТЕМПЕРАТУРЫ
#      ================================================
#
# В этой программе используется модуль датчика DHT11
# вместе с релейным модулем и светодиодом.
# Комнатная температура измеряется каждые 3 секунды.
# Если температура ниже установленного значения, реле и светодиод активируются.
# С другой стороны, если комнатная температура выше установленного значения,
#  реле и светодиод выключаются.
# В этой программе желаемая температура устанавливается с помощью 3-х кнопок
#
# Автор: Доган Ибрагим
# Файл   : ONOFFSET.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
import utime
from dht import DHT11
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin

#
# Инициализировать SPI
#
spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)

#
# Установите SetTemp, настройте светодиод, реле и кнопки
#
SetTemp = 10.0                                  # Уставка
LED = Pin(21, Pin.OUT)                          # LED
Relay = Pin(22, Pin.OUT)                        # Реле

UP = Pin(18,Pin.IN, Pin.PULL_UP)                # Кнопка
DOWN = Pin(19,Pin.IN, Pin.PULL_UP)              # Кнопка
START = Pin(20,Pin.IN, Pin.PULL_UP)             # Кнопка

LED.value(0)                                    # LED не горит
Relay.value(0)                                  # Реле отпущено
t = 10.0
```

```
#
# Показать заголовок
#
def Heading():
   global SetTemp, t
   tft.fill(TFT.WHITE)                          #  Очистить экран
   tft.rect((5, 10), (110, 100), TFT.RED)
   tft.text((18, 40), "TEMP CONTROL", TFT.BLACK, sysfont,
   1.1, nowrap=True)
   tft.hline((15,51),80, TFT.RED)

   tft.text((15, 60), "SetTemp:{:.2f}C".format(SetTemp), TFT.BLUE,
   sysfont, 1.1, nowrap=True)

   tft.text((15, 75), "   Room:{:.2f}C".format(t), TFT.RED,
   sysfont, 1.1, nowrap=True)

   if Relay.value() == 1:
      tft.text((30, 100), "RELAY ON", TFT.RED, sysfont, 1.1, nowrap=True)
   else:
      tft.text((30, 100), "RELAY OFF", TFT.BLUE, sysfont, 1.1, nowrap=True)

#
# Установите желаемую температуру (SetTemp).
# UP увеличивает, DOWN уменьшает, а СТАРТ запускает управление температурой.
#
def Desired():
   global SetTemp
   while START.value() == 1:
      tft.fill(TFT.WHITE)                       # Очистить экран
      tft.rect((5, 10), (110, 100), TFT.RED)
      tft.text((15, 30), "SET TEMP", TFT.BLACK, sysfont,
      2, nowrap=True)

      tft.text((15, 80), "SetTemp:{:.2f}C".format(SetTemp), TFT.BLUE,
      sysfont, 1.1, nowrap=True)

      while UP.value() == 1 and DOWN.value() == 1 and START.value() == 1:
         pass
      if UP.value() == 0:
         SetTemp = SetTemp + 1
      elif DOWN.value() == 0:
         SetTemp = SetTemp - 1

Desired()                                       # Чтение SetTemp
```

```
#
# Считайте температуру окружающей среды каждые 3 секунды и
# решайте что делать
#
while True:
    try:
       pin = Pin(16, Pin.OUT, Pin.PULL_DOWN)
       sensor = DHT11(pin)
       t  = (sensor.temperature)                    # Чтение температуры
       if SetTemp > t:                              # Если больше
         Relay.value(1)                             # Реле ВКЛ.
         LED.value(1)                               # ON
       else:
         Relay.value(0)                             # Реле ВЫКЛ.
         LED.value(0)                               # LED не горит
       Heading()                                    # Заголовок
    except:                                         # Ошибка чтения
         utime.sleep(10)
         continue
    utime.sleep(10)
```

Рис. *7.29:* Программа: **ONOFFSET**

На рис.7.30 показан начальный экран во время работы программы и экран во время работы алгоритма контроллера.

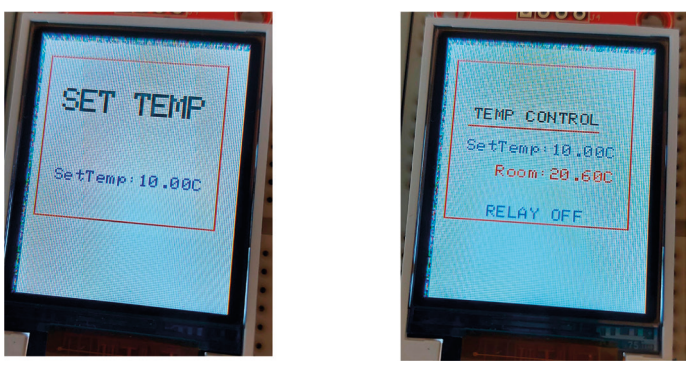


Рис. *7.30:* Начальный экран.

7.10 Проект 6: Управление температурой – установка желаемой температуры с помощью поворотного энкодера и TFT-дисплея

Описание: Этот проект похож на предыдущий, но здесь для установки желаемой температуры используется поворотный энкодер, а не кнопки. Установка желаемой температуры с помощью поворотного энкодера выполняется быстро и легко. Поворот вала энкодера одним щелчком изменяет температуру на 1 ºC.

Блок-схема: Figure 7.31 shows the block diagram of the project. The temperature settings are displayed on the TFT as the rotary encoder is turned.

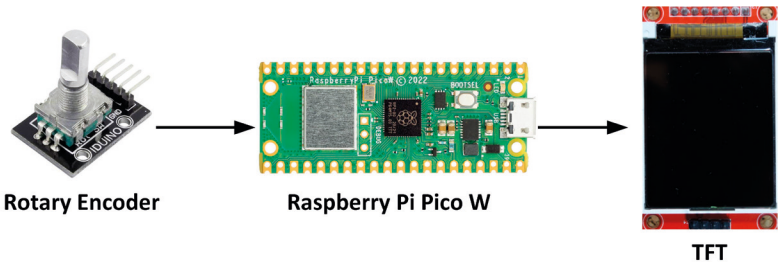


Рис.7.31: Блок-схема проекта.

Поворотный энкодер
Поворотный энкодер (рис. 7.32) представляет собой устройство, похожее на потенциометр, которое определяет вращение и направление своей ручки. Устройство имеет два внутренних контакта, которые замыкают и размыкают цепь при повороте ручки. При повороте ручки ощущается щелчок, указывающий на то, что ручка повернута на одну позицию. С помощью простой логики вы можете определить направление вращения.

Вращающийся энкодер имеет следующие пины:

GND: минус источника питания, земля

Vcc (+): плюс источника питания

CLK: это выходной контакт, используемый для определения количества вращения. Каждый раз, когда ручка поворачивается на один щелчок в любом направлении, выход CLK переходит в состояние HIGH, а затем в LOW.

DT: Это выходной сигнал, похожий на вывод CLK, но он отстает от CLK на 90 градусов. Этот выход используется для определения направления вращения

SW: Это активная кнопка LOW. Когда ручка нажата, напряжение становится LOW

В вашем проекте каждый поворот (т. е. щелчок) ручки будет увеличивать (или уменьшать) желаемую температуру на 1 градус. Поворот ручки в одном направлении увеличивает счетчик на единицу, а поворот в другую сторону уменьшает его на единицу. Когда требуемое значение достигнуто, пользователь должен нажать на ручку, чтобы программа начала управлять температурой.

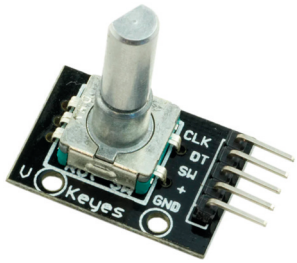


Рис.7.32: Поворотный энкодер.

Принципиальная схема: Принципиальная схема проекта показана на рисунке 7.33. Соединения между Raspberry Pi Pico W и внешними компонентами следующие:

Raspberry Pi Pico W	Номер пина	Подключен к
GP2	4	TFT SCK
GP3	5	TFT SDA
GP15	20	TFT A0
GP14	19	TFT RESET
GP5	7	TFT CS
GP16	21	DHT11
GP21	27	LED
GP22	29	RELAY
GP6	9	CLK енкодер
GP7	10	DT енкодер
GP8	11	SW енкодер

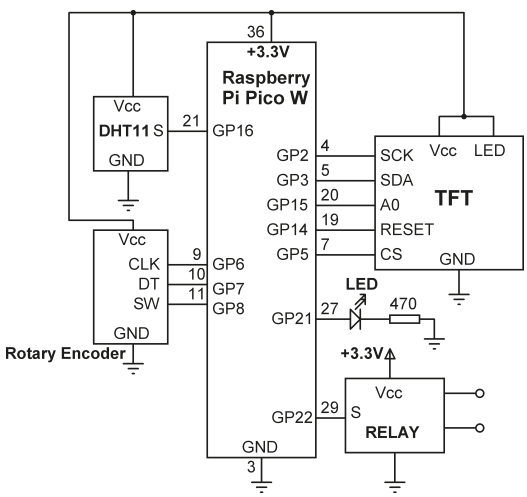


Рис.7.33: Принципиальная схема проекта.

Листинг программы: Листинг программы: В этой программе используется модуль библиотеки **rotate.py**.

На рис. 7.34 показан листинг программы (Программа:**TFTRotary.py**). В начале программы импортируются используемые модули, определяется подключение LCD, настраиваются подключения LED и RELAY, а RELAY и LED выключаются. Функция Heading() отображает заголовок с **SetTemp**, комнатной температурой и статусом RELAY, которые отображаются и обновляются каждые 2 секунды. Функция **rotate_changed()** вызывается всякий раз, когда поворотный рычаг вращается. Вращение по часовой стрелке увеличивает **SetTemp** на 1 градус, а вращение против часовой стрелки уменьшает **SetTemp** на 1 градус. Основная программа считывает температуру в помещении, сравнивает ее с желаемой температурой и затем соответственно управляет светодиодом и RELAY.

Обратите внимание, что поворотная функция управляется прерываниями, так что SetTEmp можно изменить в любое время, даже когда контроллер работает.

```
#-------------------------------------------------------------
#       УПРАВЛЕНИЕ ТЕМПЕРАТУРОЙ - НАСТРОЙКА ТЕМПЕРАТУРЫ
#       ===============================================
#
# В этой программе сенсорный модуль DHT11 используется вместе
# с релейным модулем и светодиодом. Комнатная температура
# измеряется каждые 3 секунды. Если температура ниже установленного
# значения, реле и светодиод активируются. Если, с другой стороны,
# комнатная температура выше установленного значения,
# тогда и реле, и светодиод выключаются. В этой
# программе желаемая температура устанавливается с помощью поворотного энкодера
#
# Автор: Доган Ибрагим
# Файл   : TFTRotary.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
import utime
from dht import DHT11
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin
from rotary import Rotary
#
# Инициализировать SPI
```

```
#
spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)

#
# Установите SetTemp, настройте светодиод и реле
#
global SetTemp
SetTemp = 10.0                                  # Установить точку
LED = Pin(21, Pin.OUT)                          # LED
Relay = Pin(22, Pin.OUT)                        # Relay

rotary=Rotary(7, 6, 8)                          # DT,CLK,SW

LED.value(0)                                    # LED не горит
Relay.value(0)                                  # Relay ВЫКЛ.

#
# Показать заголовок
#
def Heading():
   global SetTemp, t
   tft.fill(TFT.WHITE)                          # Очистить экран
   tft.rect((5, 10), (110, 100), TFT.RED)
   tft.text((18, 40), "TEMP CONTROL", TFT.BLACK, sysfont,
   1.1, nowrap=True)
   tft.hline((15,51),80, TFT.RED)

   tft.text((15, 60), "SetTemp:{:.2f}C".format(SetTemp), TFT.BLUE,
   sysfont, 1.1, nowrap=True)

   tft.text((15, 75), "   Room:{:.2f}C".format(t), TFT.RED,
   sysfont, 1.1, nowrap=True)

   if Relay.value() == 1:
      tft.text((30, 100), "RELAY ON", TFT.RED, sysfont, 1.1, nowrap=True)
   else:
      tft.text((30, 100), "RELAY OFF", TFT.BLUE, sysfont, 1.1, nowrap=True)

#
# Получите желаемую температуру. Каждый щелчок поворотного энкодера
# увеличивает или уменьшает температуру на 1 градус
```

```
#
def rotary_changed(change):
     global SetTemp
     if change == Rotary.ROT_CW:
         SetTemp = SetTemp + 1
     elif change == Rotary.ROT_CCW:
         SetTemp = SetTemp - 1

rotary.add_handler(rotary_changed)

while True:
    try:
       pin = Pin(16, Pin.OUT, Pin.PULL_DOWN)
       sensor = DHT11(pin)
       t  = (sensor.temperature)                    # Чтение темп.
       if SetTemp > t:                              # Если больше
         Relay.value(1)                             # Реле ВКЛ.
         LED.value(1)                               # LED ВКЛ.
       else:
         Relay.value(0)                             # Реле ВЫКЛ.
         LED.value(0)                               # LED ВЫКЛ.
       Heading()                                    # Заголовок
    except:                                         # Ошибка чтения
         utime.sleep(2)
         continue
    utime.sleep(2)
```

Рис. *7.34:* Программа: **TFTRotary.py**.

На рис.7.35 показан пример экрана. Обратите внимание, что дисплей обновляется каждые 2 секунды и **SetTemp** можно изменить в любое время.

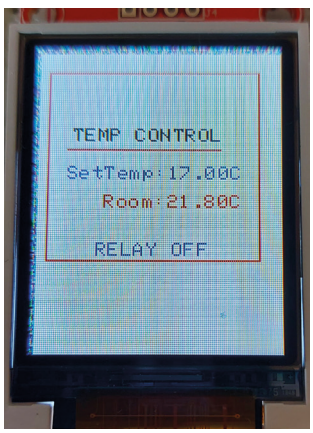


Рис. *7.35:* Проект, созданный на макетной плате.

7.11 Проект 7: растровый TFT-дисплей

Описание: В этом проекте создаются небольшие растровые изображения, которые отображаются на TFT-дисплее. Создаются два растровых изображения: в форме сердца и в форме стрелки. Шесть таких растровых изображений создаются с промежутками между ними и отображаются на экране.

Цель: Цель этого проекта — показать, как можно создавать небольшие растровые изображения и отображать их на TFT-дисплее.

Листинг программы: На рис. 7.36 показан листинг программы (Программа: **bitmaps)**. Отображаемые фигуры создаются с использованием единиц и нулей, как показано в программе. Вид **HEART** создает форму маленького сердца, а вид **ARROW** создает форму маленькой стрелки. Программа отображает 6 фигур на экране с интервалом 20 пикселей между фигурами. Цвета фигур можно легко изменить, указав новый цвет.

```
#------------------------------------------------------------
#               ОТОБРАЖЕНИЕ РАСТРОВЫХ ИЗОБРАЖЕНИЙ
#               ====================================
#
# В этой программе отображаются изображения сердца и изображения стрелок
# на экране TFT с интервалом 20 пикселей между фигурами
#
# Автор: Доган Ибрагим
# Файл   : bitmaps.py
# Дата: октябрь 2022 г.
#------------------------------------------------------------
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin

#
# Инициализировать шину SPI для TFT
#
spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)

tft.fill(TFT.BLACK)                              # Очистить экран

#
# Растровое изображение в форме сердца
#
HEART = [
[ 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

```
[ 0, 1, 1, 0, 0, 0, 1, 1, 0],
[ 1, 1, 1, 1, 0, 1, 1, 1, 1],
[ 1, 1, 1, 1, 1, 1, 1, 1, 1],
[ 1, 1, 1, 1, 1, 1, 1, 1, 1],
[ 0, 1, 1, 1, 1, 1, 1, 1, 0],
[ 0, 0, 1, 1, 1, 1, 1, 0, 0],
[ 0, 0, 0, 1, 1, 1, 0, 0, 0],
[ 0, 0, 0, 0, 1, 0, 0, 0, 0],
]

#
# Растровое изображение формы стрелки
#
ARROW = [
[ 0, 0, 0, 0, 1, 0, 0, 0, 0],
[ 0, 0, 0, 1, 1, 1, 0, 0, 0],
[ 0, 0, 1, 0, 1, 0, 1, 0, 0],
[ 0, 1, 0, 0, 1, 0, 0, 1, 0],
[ 1, 0, 0, 0, 1, 0, 0, 0, 1],
[ 0, 0, 0, 0, 1, 0, 0, 0, 0],
[ 0, 0, 0, 0, 1, 0, 0, 0, 0],
[ 0, 0, 0, 0, 1, 0, 0, 0, 0],
[ 0, 0, 0, 0, 1, 0, 0, 0, 0],
]

#
# Очистить и отобразить 6 фигур на дисплее
#

for i in range(0, 120, 20):
 for y, row in enumerate(HEART):
    for x, c in enumerate(row):
        if c == 1:
           tft.pixel((x+i, y+30), TFT.WHITE)
        else:
           tft.pixel((x+i, y+30), TFT.BLACK)
```

Рис. *7.36:* Программа: **bitmaps**.

На рис.7.37 показаны фигуры сердца и стрелки, отображаемые на экране.

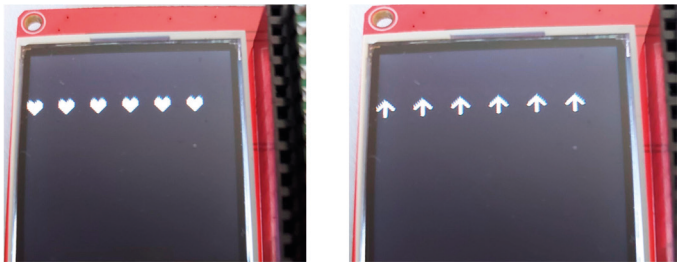


Рис.7.37: Отображение созданных фигур.

7.12 Проект 8: Использование клавиатуры 4×4

Описание: Это программа клавиатуры 4×4. Программа считывает нажатую пользователем клавишу и выводит ее код на экран Тонни.

Цель: цель этого проекта — показать, как клавиатура 4×4 может использоваться вместе с TFT-дисплеем в проекте Raspberry Pi Pico W. Клавиатура 4×4: существует несколько типов клавиатур, которые можно использовать на базе МК проектов. В этом проекте используется клавиатура 4×4 (см. рис. 7.38). На этой клавиатуре есть клавиши для цифр от 0 до 9 и букв A, B, C, D, * и #. Клавиатура соединена с процессором с помощью 8 проводов с именами от R1 до R4 и от C1 до C4, представляющих соответственно ряды и столбцы клавиатуры (см. Рис.7.39).

Рис.7.38: Клавиатура 4×4.

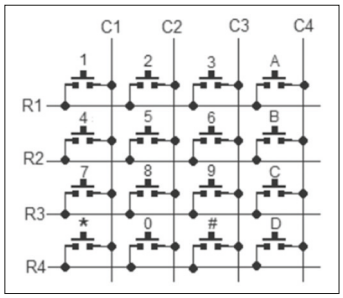


Рис.7.39: Принципиальная схема клавиатуры 4×4.

Работа с клавиатурой довольно проста: столбцы настроены как выходы, а строки как входы. Нажатая клавиша (включение; замыкание) идентифицируется с помощью сканирования столбца. Здесь столбец принудительно устанавливается в 0, а другие столбцы - в единицу. Затем сканируется состояние каждой строки, и если обнаруживается, что строка имеет 0, то клавиша на пересечении строки и столбца, является нажатой. Этот процесс повторяется для всех строк.

Block diagram: Figure 7.40 shows the block diagram

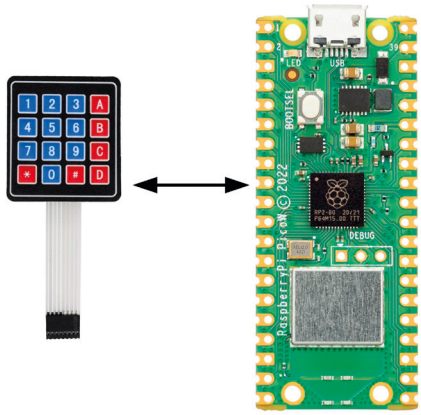


Рис.7.40: Блок-схема.

Принципиальная схема: Принципиальная схема проекта показана на рис.7.41. TFT-дисплей подключен к Pico, как и в предыдущих проектах. Клавиатура 4×4 подключена к следующим контактам GPIO Raspberry Pi Pico W.

Keypad pin	Raspberry Pi pin
R1	P16
R2	GP17
R3	GP18
R4	GP19

Структура модуля матричной клавиатуры 4 × 4

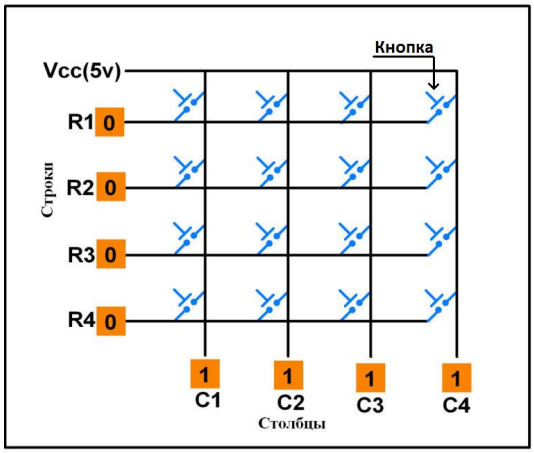


Рис.1. Структура матричной клавиатуры 4×4

Модуль клавиатуры сделан в виде матрицы, где строки и столбцы пересекаются. Когда происходит нажатие кнопки, она замыкает место пересечения строки (ряда) и столбца. Если нажатия не было, то и связи между строкой и столбцом не происходит. Столбцы имеют высокий уровень, а строки - низкий, что и показано на рис.1

Работа с матричной клавиатурой

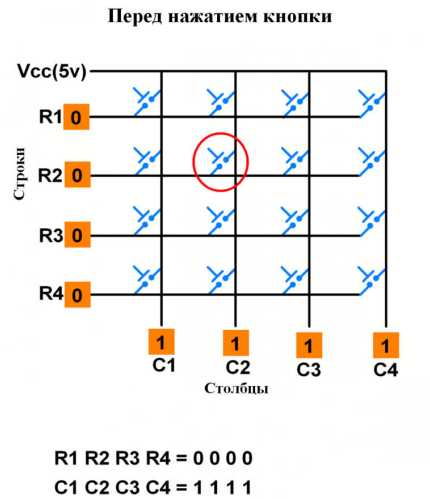


Рис.2. Матричная клавиатура 4×4 перед нажатием кнопки

Для того чтобы обнаружить нажатую клавишу, микроконтроллер все время должен сканировать кнопки клавиш. Сканирование происходит по следующему алгоритму: Микроконтроллер заземляет все строки, устанавливая 0 на выходные контакты (R1, R2, R3 и R4), а затем считывает столбцы (C1, C2, C3 и C4), показанные на рис. 2.

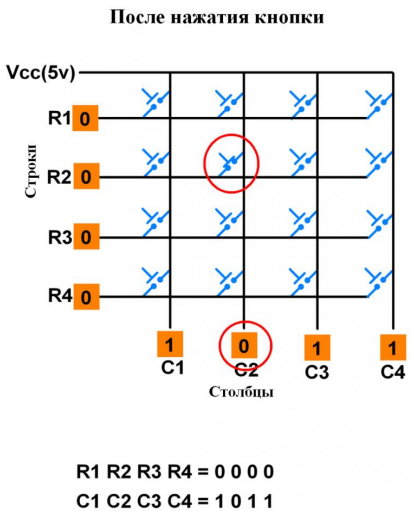


Рис.3. Матричная клавиатура 4×4 после нажатия кнопки

В случае нажатия кнопки в одном из битов столбца установится ноль. Например, если C1: C4 = 1 0 1 1, это означает, что была нажата клавиша в столбце с идентификатором C2.

После того как было зафиксировано нажатие клавиши, микроконтроллер выполнит процесс ее идентификации.

Процесс определения нажатой кнопки

Начиная с верхнего ряда, микроконтроллер (МК) заземлит его, подав низкий уровень только на ряд R1.

Если после чтения столбцов, данные равны 1, ни одна клавиша в этой строке не нажата и МК заземляет следующий ряд, R2.

Далее, как видно из рис.3, нажата кнопка на пересечении R2-C2, откуда C2 равен 0, откуда МК идентифицирует ряд R2, где была нажата кнопка.

Keypad pin	Raspberry Pi pin
R1	P16
R2	GP17
R3	GP18
R4	GP19

C1	GP20
C2	GP21
C3	GP22
C4	GP26

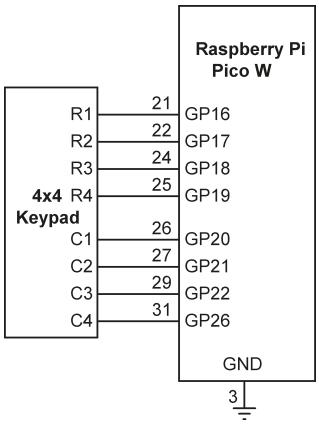


Рис.7.41: Принципиальная схема.

На рис.7.42 показана конфигурация контактов клавиатуры 4×4, используемой в проекте.

Рис.7.42: Конфигурация контактов клавиатуры 4×4.

Листинг программы:Работа алгоритма клавиатуры может быть описана следующим языком описания программы (PDL):

```
Configure all columns as outputs
Configure all rows as inputs
Set all columns to 1
DO for all columns
       Set a column to 0
       DO for all rows
              IF a row is 0 THEN
                     Return the key at this column and row position
              ENDIF
       ENDDO
ENDDO
```

На рис. 7.43 показан листинг программы (программа: **keypad.py**). В начале программы кнопки клавиатуры определяются после импорта необходимых модулей в программу.
Соединения строк и столбцов клавиатуры определяются с помощью списков **ROWS** и **COLS** соответственно. Затем столбцы настраиваются как выходные данные, и им присваивается значение 1. Точно так же строки настраиваются как входные данные. Функция **Get_Key** считывает нажатую клавишу и возвращает ее вызывающей программе. В функции используются два цикла **for**: первый цикл выбирает столбцы и устанавливает их в 0 один за другим. Второй цикл сканирует строки и проверяет, находится ли строка в 0. Основная программа вызывает функцию и отображает нажатую клавишу на экране.

```
#-----------------------------------------------------------------
#                    ПРОГРАММА ПРОВЕРКИ КЛАВИАТУРЫ
#                    ================================
#
# Эта программа показывает, как можно использовать
# клавиатуру для отображения нажатых клавиш
#
# Автор: Доган Ибрагим
# Файл   : keypad.py
# Дата: октябрь 2022 г.
#-----------------------------------------------------------------
from machine import Pin
import utime

C = [0]*4
R = [0]*4

KEYPAD = [                                      # кнопки клавиатуры
    [1,2,3,"A"],
    [4,5,6,"B"],
    [7,8,9,"C"],
    ["*",0,"#","D"]]
```

```
ROWS = [16,17,18,19]                          # Пины рядов
COLS = [20,21,22,26]                          # Пины столбцов

for i in range(4):                            # Конфигурация столбцов
   C[i] = Pin(COLS[i], Pin.OUT)
   C[i].value(1)

for j in range(4):                            # Конфигурация ряда
   R[j] = Pin(ROWS[j], Pin.IN, Pin.PULL_UP)

#
# Эта функция считывает клавишу с клавиатуры
#
def Get_Key():
   while True:
      for j in range(4):
          C[j].value(0)                       # Установить столбец j равным 0
          for i in range(4):                  # Для всех рядов
              if R[i].value() == 0:           # Ряд равен 0?
                    return (KEYPAD[i][j])     # Клавиша Return
                    while R[i].value() == 0:
                       pass
          C[j].value(1)                       # Обратно к 1
          utime.sleep(0.05)                   # ждите 0,05 с

while True:
   key = Get_Key()                            # Получить значение клавиши
   print(key)                                 # Показать значение клавишиы
   utime.sleep(0.5)
```

Рис. *7.43:* Программа: **keypad.py**.

При нажатии клавиши на клавиатуре ее код отображается на экране. На рис. 7.44 показан дисплей после нажатия всех клавиш.

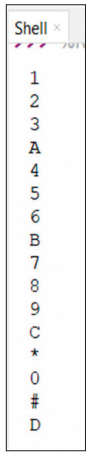


Рис.7.44: Нажатие всех клавиш.

7.13 Проект 9: Элементарное умножение – с помощью клавиатуры 4×4 и TFT

Описание: Это элементарное упражнение на умножение. Программа отображает два случайных числа от 1 до 100 и ждет, пока пользователь введет результат умножения этих чисел. Результат проверяется, и пользователь получает обратную связь, такую как WELL DONE - МОЛОДЕЦ или WRONG - НЕПРАВИЛЬНО... Вышеописанный процесс повторяется с задержкой в 5 секунд.

Цель: Цель этого проекта состоит в том, чтобы дети тренировали свои навыки умножения.

Блок-схема: На рис. 7.45 показана блок-схема добавления в проект TFT-дисплея.

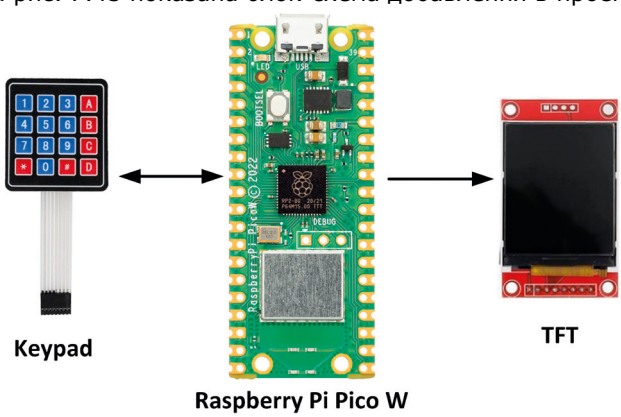


Рис.7.45: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис.7.46. TFT-дисплей подключен к Pico, как и в предыдущих проектах.

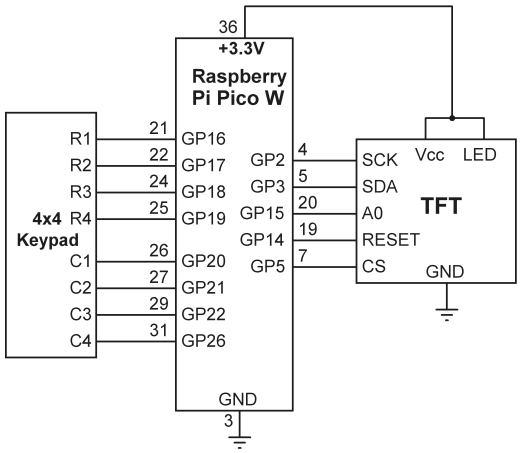


Рис.7.46: Принципиальная схема проекта.

Листинг программы: На рис. 7.47 показан листинг программы (**TFTmultiply.py**). Клавиша D считается клавишей **ENTER**, и ее необходимо нажать после ввода ответа. В начале программы импортируются все необходимые модули. Функция **Heading()** рисует красный прямоугольник и отображает **MULTIPLICATION** - УМНОЖЕНИЕ. Внутри основной программы генерируются два случайных числа от 1 до 100 (**n1** и **n2**). Затем ожидается, что пользователь умножит эти числа вручную и введет результат. Клавишу **D** необходимо ввести после ввода результата. Программа проверяет результат и выводит либо **WELL DONE** - ВЫПОЛНЕНО, либо **WRONG** - НЕПРАВИЛЬНО... Вышеописанный процесс повторяется через 5 секунд, когда генерируется новый набор чисел.

```
#-----------------------------------------------------------------
#                   ЭЛЕМЕНТАРНОЕ УМНОЖЕНИЕ
#                   ==========================
#
# Это элементарная программа умножения.
# Программа отображает два случайных целых числа от 1 до 100
# и ожидает, что пользователь правильно умножит эти числа.
#
# Автор: Доган Ибрагим
# Файл   : TFTmultiply.py
# Дата: октябрь 2022 г.
#-----------------------------------------------------------------
import utime
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin
import random


#
# Инициализировать SPI
#
```

```
spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)

C = [0]*4
R = [0]*4

KEYPAD = [                                  # клавиши клавиатуры
    [1,2,3,"A"],
    [4,5,6,"B"],
    [7,8,9,"C"],
    ["*",0,"#","D"]]

ROWS = [16,17,18,19]                        # Ряд пинов
COLS = [20,21,22,26]                        # пины столбцов

for i in range(4):                          # Конфигурация столбцов
   C[i] = Pin(COLS[i], Pin.OUT)
   C[i].value(1)

for j in range(4):                          # Конфигурация рядов
   R[j] = Pin(ROWS[j], Pin.IN, Pin.PULL_UP)

def Heading():
    tft.fill(TFT.WHITE)                     # Очистить экран
    tft.rect((5, 10), (110, 120), TFT.RED)
    tft.text((17, 30), "MULTIPLICATION", TFT.BLUE, sysfont,
       1.1, nowrap=True)

#
# Эта функция считывает клавишу клавиатуры
#
def Get_Key():
   while True:
      for j in range(4):
          C[j].value(0)                     #  Установить столбец j равным 0
          for i in range(4):                # Для всех строк
              if R[i].value() == 0:         # Ряд равен 0?
                    return (KEYPAD[i][j])   # Клавиша Return
                    while R[i].value() == 0:
                       pass
          C[j].value(1)                     #  Обратно к 1
          utime.sleep(0.05)                 # ждите 0,05 с
```

```
#
# Запуск MAIN программы
#
while True:
   Heading()
   n1 = random.randint(1, 100)                  # Не случайно
   n2 = random.randint(1, 100)                  # Не случайно
   tft.text((15, 60), "{:d} X {:d} = ".format(n1,n2), TFT.BLUE,
    sysfont, 1.1, nowrap=True)

   ans = 0
   key = " "
   r = 15
   while True:
       key = Get_Key()                          #  Нажать клавишу
       if key == "D":
          break
       tft.text((r, 75), "{:d}".format(int(key)), TFT.MAROON,
       sysfont, 1.1, nowrap=True)
       r = r + 6
       ans = 10*ans + int(key)                  # Показать результат нажатия
       utime.sleep(0.5)
       result = n1 * n2
   if result == ans:
       tft.text((10, 90), "WELL DONE",TFT.GREEN,sysfont,2,nowrap=True)
   else:
       tft.text((10, 90), "WRONG...",TFT.MAROON,sysfont,2,nowrap=True)
   utime.sleep(5)
```

Рис.*7.47:* Программа:**TFTmultiply.py**.

На рис.7.48 показан пример экрана с правильным ответом.

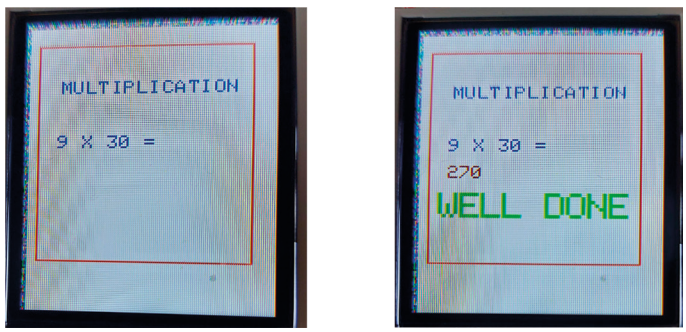


Рис.*7.48:* Пример экрана.

7.14 Проект 10: Калькулятор - с клавиатурой 4×4 и TFT

Описание: Это проект целочисленного калькулятора. Пользователь вводит два целых числа и требуемую операцию (+-*/). Результат вычисляется и отображается на TFT-дисплее. Программа повторяется после 10-секундной задержки.

Блок-схема и принципиальная схема проекта такие же, как на рис. 7.45 и рис. 7.46.

Листинг программы: Keypad key numbering for this project is shown below. Keys ABCs arereplaced with +-*, key * with / and key D with ENTER:

```
1 2 3 +
4 5 6 -
7 8 9 *
/ 0 # ENTER
```

На рис.7.49 показан листинг программы (Программа: **TFTcalc**). Функция **Get_Key()** считывает ключ с клавиатуры, и это то же самое, что и в предыдущем проекте. Программа предлагает пользователю ввести два целых числа. Циклы **while** используются для чтения цифр чисел, введенных пользователем, и эти числа сохраняются в переменных **n1** и **n2**. Эти циклы while завершаются при нажатии клавиши **D** (ENTER). Затем программа считывает требуемую операцию и сохраняет ее в переменной **opkey**. Результат вычисляется в зависимости от значения **opkey**. Например, если выбрано сложение, то добавляются **n1** и **n2**, и результат отображается на TFT.

```
#--------------------------------------------------------------
#                    КАЛЬКУЛЯТОР
#                    ===========
#
# Это программа-калькулятор целых чисел, которая может выполнять
#  4 основные операции +-*/
#
#  Автор: Доган Ибрагим
#  Файл  : TFTcalc.py
#  Дата: октябрь 2022 г.
#--------------------------------------------------------------
import utime
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin
import random

#
# Инициализировать SPI
#
spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
```

```
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)

C = [0]*4
R = [0]*4

KEYPAD = [                                  # Клавиши клавиатуры
    [1,2,3,"+"],
    [4,5,6,"-"],
    [7,8,9,"*"],
    ["/",0,"#","D"]]

ROWS = [16,17,18,19]                        # Ряд пинов
COLS = [20,21,22,26]                        # Пины столбцов

for i in range(4):                          # Конфигурация столбцов
   C[i] = Pin(COLS[i], Pin.OUT)
   C[i].value(1)

for j in range(4):                          # Конфигурация рядов
   R[j] = Pin(ROWS[j], Pin.IN, Pin.PULL_UP)

def Heading():
    tft.fill(TFT.WHITE)                     # Очистить экран
    tft.rect((5, 10), (110, 120), TFT.RED)
    tft.text((26, 30), "CALCULATOR", TFT.BLUE, sysfont,
       1.1, nowrap=True)

#
# Эта функция считывает ключ с клавиатуры
#
def Get_Key():
   while True:
      for j in range(4):
          C[j].value(0)                     # Установить столбец j равным 0
          for i in range(4):                # Для всех строк
              if R[i].value() == 0:         # Строка равна 0?
                    return (KEYPAD[i][j])   # Клавиша Return
                    while R[i].value() == 0:
                       pass
          C[j].value(1)                     # Обратно к 1
          utime.sleep(0.05)                 # ждите 0,05 с
#
```

```
# Запуск  MAIN  программы
#
while True:
   Heading()
#
# Получить первое число
#
   tft.text((10, 50), "First no: ", TFT.MAROON,sysfont, 1.1, nowrap=True)
   n1 = 0
   key = " "
   r = 70
   while True:
       key = Get_Key()                          # Показать включение (замыкание)
       if key == "D":
          break
       tft.text((r, 50), "{:d}".format(int(key)), TFT.MAROON,
       sysfont, 1.1, nowrap=True)
       r = r + 6
       n1 = 10*n1 + int(key)                    # Показать включение
       utime.sleep(0.5)
#
# Получить второе число
#
   r = 70
   tft.text((10, 70), "Second no: ", TFT.MAROON,sysfont, 1.1, nowrap=True)
   n2 = 0
   key = " "
   r = 75
   while True:
       key = Get_Key()                          # Получить замыкание
       if key == "D":
          break
       tft.text((r, 70),"{:d}".format(int(key)),TFT.
MAROON,sysfont,1.1,nowrap=True)
       r = r + 6
       n2 = 10*n2 + int(key)                    # Показать замыкание
       utime.sleep(0.5)

#
# Get operation
#
   tft.text((10, 90),"OP (+-*/): ",TFT.MAROON,sysfont,1.1,nowrap=True)
   key = " "
   while True:
       key = Get_Key()                          # Получить замыкание
       if key == "D":
```

```
            break
        tft.text((74, 90),"{:s}".format(key),TFT.MAROON,sysfont,1.1,nowrap=True)
        opkey = key
        utime.sleep(0.5)
#
# Показать результат
#
    if opkey == "+":
         result = n1 + n2
    elif opkey == "-":
         result = n1 - n2
    elif opkey == "*":
         result = n1 * n2
    elif opkey == "/":
         result = int(n1 / n2)
    tft.text((10, 110),"Result={:d}".format(result),TFT.
BLUE,sysfont,1.1,nowrap=True)

    utime.sleep(10)
```

Рис.*7.34:* Программа:**TFTcalc**.

На рис.7.50 показан пример экрана.

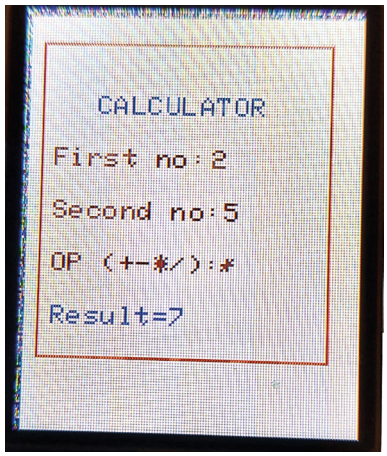


Рис.7.50: Пример экрана.

7.15 Проект 11: игра HiLo — с использованием клавиатуры 4×4 и TFT

Описание: Это классическая игра HiLo. Компьютер генерирует секретное случайное число от 1 до 100, и пользователь пытается угадать это число. Пользователю даются подсказки, когда введенное предположение является близким или отдаленным.

Блок-схема и принципиальная схема проекта такие же, как на рис. 7.45 и рис. 7.46.

Листинг программы: На рис. 7.51 показан листинг программы (Программа: **HiLo.py**). Раскладка клавиатуры такая же, как и в Project 9, где **D** — клавиша ENTER. Программа отображает **TOO HIGH** - СЛИШКОМ БОЛЬШОЕ, если разница между вашим предположением и секретным числом больше 50, отображает **HIGH**, если предположение велико, но разница меньше 50, отображает TOO LOW, если разница между вашим предположением и секретным числом меньше 50, и отображает LOW, если ваша догадка низкая, но разница не меньше 50. Программа также отображает количество попыток, предпринятых при обнаружении секретного числа.

```
#-----------------------------------------------------------------
#                   HiLo GAME
#                   ==========
#
# Генерируется случайное число от 1 до 100, и пользователь
# пытается угадать это число. Программа дает подсказки следующим образом:
#
# TOO HIGH: Угаданное число очень велико (более 50)
# HIGH: угаданное число является HIGH (менее 50)
# TOO LOW: Угаданное число слишком мало (более 50)
# LOW: Угаданное число низкое (менее 50)
#
# Автор: Доган Ибрагим
# Файл  : HiLo.py
# Дата: октябрь 2022 г.
#-----------------------------------------------------------------
import utime
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin
import random

#
# Инициализировать SPI
#
spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)

C = [0]*4
R = [0]*4
```

```
KEYPAD = [                                  # Клавиши клавиатуры
    [1,2,3,"A"],
    [4,5,6,"B"],
    [7,8,9,"C"],
    ["*",0,"#","D"]]

ROWS = [16,17,18,19]                        # пины ряда
COLS = [20,21,22,26]                        # пины столбцов

for i in range(4):                          # конфигурация столбцов
   C[i] = Pin(COLS[i], Pin.OUT)
   C[i].value(1)

for j in range(4):                          # конфигурация рядов
   R[j] = Pin(ROWS[j], Pin.IN, Pin.PULL_UP)

def Heading():
    tft.fill(TFT.WHITE)                     # Очистить экран
    tft.rect((5, 10), (110, 120), TFT.RED)
    tft.text((13, 30), "GUESS MY NUMBER", TFT.BLUE, sysfont,
       1.1, nowrap=True)

#
# Эта функция считывает ключ с клавиатуры
#
def Get_Key():
   while True:
      for j in range(4):
          C[j].value(0)                     # Установить столбец j равным 0
          for i in range(4):                # Для всех строк
              if R[i].value() == 0:         # Строка равна 0?
                    return (KEYPAD[i][j])   # Клавиша возврата
                    while R[i].value() == 0:
                       pass
          C[j].value(1)                     #  Обратно к 1
          utime.sleep(0.05)                 # ждите 0,05 с

#
# Start of MAIN program
#
attempts = 0
n = random.randint(1, 100)

while True:
   Heading()
   tft.text((10, 50), "Your Guess: ", TFT.MAROON,sysfont, 1.1, nowrap=True)
```

```
attempts = attempts + 1

n1 = 0
key = " "
r = 80
while True:
    key = Get_Key()                          # Получить замыкание (включение)
    if key == "D":
       break
    tft.text((r, 50), "{:d}".format(int(key)), TFT.MAROON,
    sysfont, 1.1, nowrap=True)
    r = r + 6
    n1 = 10*n1 + int(key)                    # Показать включение
    utime.sleep(0.5)

if n1 == n:
   tft.text((10, 100),"WELL DONE",TFT.BLUE,sysfont,1.1,nowrap=True)
   tft.text((10, 120), "Attempts:{:d}".format(attempts),TFT.BLUE,
   sysfont,1.1,nowrap=True)
   attempts = 0
   n = random.randint(1, 100)
elif n1 > n + 50:
   tft.text((10, 110),"TOO HIGH",TFT.BLUE,sysfont,1.1,nowrap=True)
elif n1 > n:
   tft.text((10, 110),"HIGH",TFT.BLUE,sysfont,1.1,nowrap=True)
elif n1 < n - 50:
   tft.text((10, 110),"TOO LOW",TFT.BLUE,sysfont,1.1,nowrap=True)
elif n1 < n:
   tft.text((10, 110),"LOW",TFT.BLUE,sysfont,1.1,nowrap=True)
utime.sleep(3)
```

Рис. *7.51:* Программа: **HiLo.py**.

На рис. 7.52 показан пример экрана.

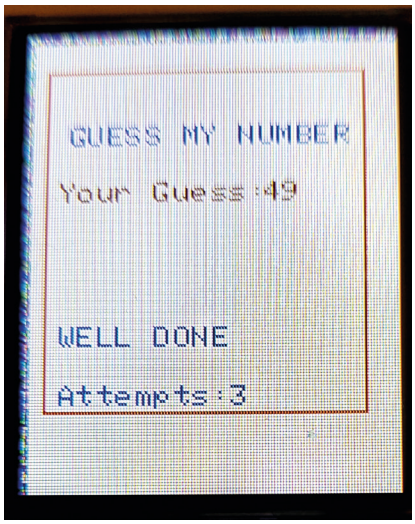


Рис.7.52: Пример экрана.

Глава 8 • Проекты шины I²C

8.1 Обзор

Шина I²C (также пишется как I2C) обычно используется в проектах на основе МК. В этой главе вы рассмотрите использование этой шины на Raspberry Pi Pico. Цель состоит в том, чтобы познакомить читателя с функциями библиотеки шины I2C и показать, как их можно использовать в реальном проекте. Прежде чем рассматривать детали проекта, стоит рассмотреть основные принципы работы шины I2C.

8.2 Шина I²C

Как кратко обсуждалось в разделе 5.3, шина I2C является одним из наиболее часто используемых протоколов МК для связи с внешними устройствами, такими как датчики и исполнительные механизмы. Шина I2C представляет собой одну ведущую и несколько подчиненных шин и может работать в стандартном режиме: 100 Кбит/с, полной скорости: 400 Кбит/с, быстром режиме: 1 Мбит/с и высокой скорости: 3,2 Мбит/с. .Шина состоит из двух проводов с открытым стоком, с резисторами подтяжки:

SDA: линия данных
SCL: тактовая линия

На рис. 8.1 показана структура шины I²C с одним ведущим и тремя ведомыми. Обратите внимание, что для корректной работы шины необходимы подтягивающие резисторы к источнику питания. Это связано с тем, что устройства на шине находятся в режиме открытого стока. Некоторые ведомые устройства уже имеют внутренние подтягивающие резисторы. Важно проверить используемые устройства, и если нет подтягивающих резисторов, то необходимо добавить внешние, когда устройство находится на шине.

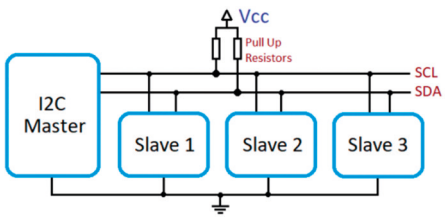


Рис.8.1: Элементарная структура шины I2C.

8.3 Контакты I2C Raspberry Pi Pico W

Как обсуждалось в разделе 5.4, Raspberry Pi Pico и Pico W имеют два вывода I2C, названные I2C0 и I2C1, которые повторяются здесь на рисунке 8.2. Как показано на рисунке, выводы I2C дублируются и используются совместно с другими выводами. Например, GP0 (контакт 1) — это контакт I2C0 SDA, а GP1 (контакт 1) — это контакт I2C0 SCL. Кроме того, GP16 (вывод 21) — это вывод I2C0 SDA, а GP17 (вывод 22) — вывод I2C SCL.

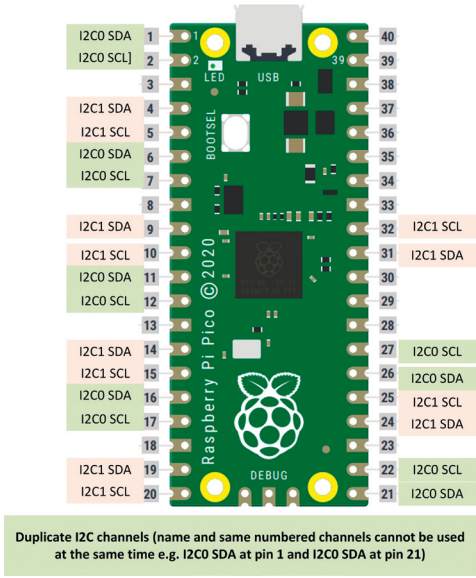


Рис.8.2: Распиновка Raspberry Pi Pico и Pico W I 2 C.

Выводы I2C по умолчанию:

I2C0 SCL	GP9
I2C0 SDA	GP8
I2C1 SCL	GP7
I2C1 SDA	GP6

Ведущий на шине всегда инициирует обмен данными и обычно требует данных от ведомого устройства (например, датчика). Ведомые устройства реагируют на ведущего и не могут инициировать обмен данными по шине. В дополнение к обычной передаче данных по шине бит подтверждения (ACK) используется для сигнализации передатчику об успешном приеме байта.

Преимущества использования шины I^2C:

- Необходимо всего 2 провода.
- На шине поддерживается до 1008 ведомых устройств.
- Поддерживается работа с несколькими мастерами (более одного мастера).

Недостатками шины I^2C являются:

- Скорость передачи данных не очень высока.
- Синхронизация шины сложная.

В оставшихся частях этой главы будут представлены проекты с использованием шины I2C.

8.4 Проект 1: Расширитель портов I2C

Описание:

В этом разделе приведен простой проект, показывающий, как функции I2C можно использовать в программе. В этом проекте микросхема расширителя портов, совместимая с шиной I2C (MCP23017), используется для предоставления дополнительных 16 портов ввода-вывода Raspberry Pi Pico W. Это полезно в некоторых приложениях, где может потребоваться много портов ввода-вывода. В этом проекте светодиод подключен к контакту GPA0 порта MCP23017 (контакт 21), и он мигает каждую секунду, чтобы можно было проверить работу программы. Последовательно со светодиодом используется токоограничивающий резистор сопротивлением 470 Ом.

Цель: цель этого проекта — показать, как можно использовать шину I2C в проектах Raspberry Pi Pico.

Блок-схема: Блок-схема проекта показана на рисунке 8.3.

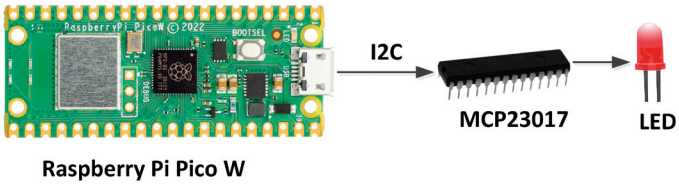


Рис.8.3: Блок-схема проекта.

MCP23017

MCP23017 — это 28-контактный чип со следующими характеристиками. Распиновка выводов показана на рис. 8.4:

- 16 двунаправленных портов ввода/вывода
- Работа на частоте до 1,7 МГц по шине I2C
- Возможность прерывания
- Вход внешнего сброса
- Низкий ток в режиме ожидания
- Работа от +1,8 до +5,5 В
- 3 адресных контакта, так что на шине I^2C можно использовать до 8 устройств.
- 28-контактный DIP-корпус

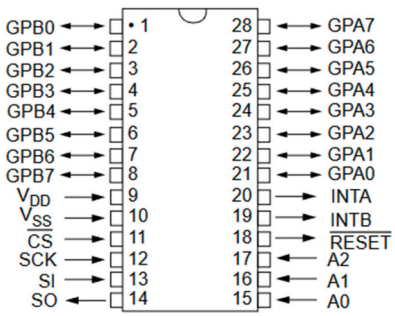


Рис. *8.4:* Распиновка *MCP23017.*

Описание выводов приведено в таблице 8.1.

Pin	Описание
GPA0-GPA7	Port A pins
GPB0-GPB7	Port B pins
VDD	Power supply
VSS	Ground
SDA	I²C data pin
SCL	I²C clock pin
RESET	Reset pin
A0-A2	I²C address pins

Таблица 8.1: Распиновка MCP23017.

Адресация MCP23017 осуществляется с помощью пинов с A0 по A2. Таблица 8.2 показывает выбор адреса. В этом проекте контакты адреса подключены к земле, поэтому адрес чипа равен 0x20. Адрес микросхемы имеет ширину 7 бит, а младший бит установлен или очищен в зависимости от того, хотите ли вы считывать данные с микросхемы или записывать данные на микросхему соответственно. Поскольку в этом проекте вы будете писать в MCP23017, младший бит должен быть равен 0, что делает байтовый адрес микросхемы (также называемый кодом операции устройства) равным 0x40.

A2	A1	A0	Адрес
0	0	0	0x40
0	0	1	0x21
0	1	0	0x22
0	1	1	0x23
1	0	0	0x24
1	0	1	0x25
1	1	0	0x26
1	1	1	0x27

Таблица 8.2: Выбор адреса MCP23017.

TМикросхема MCP23017 имеет 8 внутренних регистров, которые можно настроить для своей работы. Устройство может работать либо в 16-битном режиме, либо в двух 8-битных режимах путем настройки бита IO-CON.BANK. При включении питания этот бит сбрасывается, что по умолчанию выбирает два 8-битных режима.

Направление ввода/вывода выводов порта выбирается регистрами IODIRA (по адресу 0x00) и IODIRB (по адресу 0x01). Сброс бита в 0 в этих регистрах приводит к выходу вывода(ов) соответствующего порта. Точно так же установка бита в 1 в этих регистрах делает вход(ы) соответствующего пина порта. Адреса регистров GPIOA и GPIOB равны 0x12 и 0x13 соответственно. Это показано на рис. 8.5.

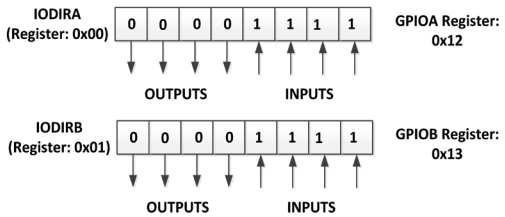


Рис. 8.5: Настройка портов ввода-вывода.

На рис.8.6 показана принципиальная схема проекта. Обратите внимание, что пины I2C расширителя портов подключены к пинам GP8 (I2C0 SDA) и GP9 (I2C0 SCL) Raspberry Pi Pico и подтянуты с помощью резисторов 10 кОм, как того требуют спецификации I2C. Светодиод подключен к выводу GPA0 порта MCP23017 (вывод 21). Все биты выбора адреса MCP23017 подключены к земле.

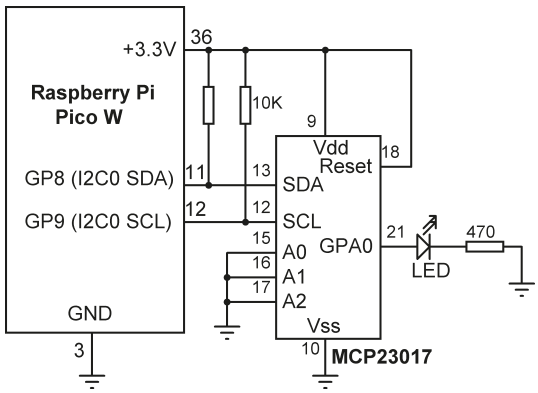


Рис. 8.6: Принципиальная схема проекта.

Более подробную информацию о микросхеме MCP23017 можно получить из даташита:
http://docs-europe.electrocomponents.com/webdocs/137e/0900766b8137eed4.pdf

Листинг программы: На рис. 8.7 показан листинг программы (Программа: MCP23017). RaspberryPi Pico поддерживает следующие функции I2C:

i2c.scan()	сканирование ведомых устройств I2C
i2c.writeto()	запись на шину I2C
i2c.readfrom()	чтение с шины I2C
i2c.writeto_mem()	запись в память ведомого устройства
i2c.readfrom_mem()	чтение из памяти ведомого устройства

Некоторые примеры операций I2C:

`print("I2C address=", i2c.scan())`	вывести адреса ведомых устройств I2C на шине
`i2c.writeto(0x20, b'56')`	записать 56 на адрес I2C 0x20
`i2c.writeto(0x20, bytearray(buff))`	буфер записи на адрес I2C 0x20
`i2c.readfrom(0x20, 3)`	прочитать 3 байта с адреса I2C 0x20
`i2c.writeto_mem(0x20, 0x10, b'\x35')`	записать 0x35 в адрес памяти 0x10 подчиненного устройства, чей адрес I2C равен 0x20
`i2c.readfrom_mem(0x20, 0x10, 3)`	прочитать 3 байта, начиная с адреса регистра 0x10 ведомого устройства I2C, адрес которого 0x20

В начале программы модуль I2C импортируется в программу, а интерфейс I2C определяется путем указания соединений контактов SDA и SCL с Pico. Функция i2c.scan() вызывается для отображения подчиненных устройств I2C на шине, и было отображено следующее:

адрес i2c = [32], что соответствует шестнадцатеричному 0x20

Затем определяются адрес устройства I2C MCP23017, адрес регистра **GPIOA** и адрес **IODIRA** направления ввода-вывода. List **conf** хранит адрес регистра **IODIRA** и 0, чтобы MCP23017 PORTA был установлен в режим вывода. List **buff1** хранит адрес регистра **GPIOA** и то, на что должен быть установлен выход (1 для включения светодиода). Точно так же **buff2** хранит адрес регистра **GPIOA** и то, на что должен быть установлен выход (0, чтобы выключить светодиод). Внутри основного цикла программы светодиод мигает каждую секунду.

```
#-------------------------------------------------------------
#              РАСШИРИТЕЛЬ ПОРТОВ I2C
#              ==========================
#
# В этом проекте используется микросхема расширения портов MCP23017.
# Светодиод подключен к пину порта GP0 расширителя портов
# и этот светодиод мигает каждую секунду
#
# Автор: Доган Ибрагим
# Файл  : MCP23017.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
from machine import Pin,I2C
import utime

i2c = machine.I2C(0, scl=Pin(9), sda=Pin(8), freq=100000)
```

```
print("i2c address=",i2c.scan())

Device_Address = 0x20                   # MCP23017 I2C address
MCP_GPIOA_REG = 0x12                    # MCP23017 GPIOA address
MCP_IODIRA_REG = 0                      # MCP23017 IODIRA Address

conf = [MCP_IODIRA_REG, 0]              # Configure as output
buff1 = (MCP_GPIOA_REG, 0)              # Set GPIOA to 0
buff0 = [MCP_GPIOA_REG, 1]              # Set GPIOA to 1

i2c.writeto(Device_Address, bytearray(conf))

while True:
    i2c.writeto(Device_Address, bytearray(buff1))
    utime.sleep(1)
    i2c.writeto(Device_Address, bytearray(buff0))
    utime.sleep(1)
```

Рис. *8.7:* Программа*:* **MCP23017**

Программа может быть изменена так, чтобы использовались функции памяти I2C. Листинг измененной программы (Программа: **MCP23017-2**) показан на рис. 8.8.

```
#-------------------------------------------------------------
#              РАСШИРИТЕЛЬ ПОРТОВ I2C
#              ==========================
#
# В этом проекте используется микросхема расширения портов MCP23017.
# Светодиод подключен к контакту порта GP0 расширителя портов
# и этот светодиод мигает каждую секунду.
#
# Эта версия программы использует функции памяти i2c
#
# Автор: Доган Ибрагим
# Файл : MCP23017-2.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
from machine import Pin,I2C
import utime

i2c = machine.I2C(0, scl=Pin(9), sda=Pin(8), freq=100000)
print("i2c address=",i2c.scan())

Device_Address = 0x20                   # MCP23017 I2C-адрес
MCP_GPIOA_REG = 0x12                    # MCP23017 Адрес GPIOA
MCP_IODIRA_REG = 0                      # Адрес MCP23017 IODIRA
```

```
i2c.writeto_mem(Device_Address, MCP_IODIRA_REG, b'0')

while True:
    i2c.writeto_mem(Device_Address, MCP_GPIOA_REG, b'1')
    utime.sleep(1)
    i2c.writeto_mem(Device_Address, MCP_GPIOA_REG, b'0')
    utime.sleep(1)
```

Рис.*8.8:* Модифицированная программа:**MCP23017-2**

8.5 Проект 2: датчик температуры TMP102 с ЖК-дисплеем

Описание: В этом проекте используется микросхема датчика температуры TMP102, совместимая с I2C. Температура окружающей среды считывается каждые 3 секунды и отображается на дисплее I2C.

Цель: Цель этого проекта — показать, как чип датчика температуры TMP102 можно использовать в программе.

TMP102

TMP102 — это I2C-совместимый высокоточный чип датчика температуры со встроенным термостатом, имеющий следующие основные характеристики:

Напряжение питания: от 1,4 В до 3,6 В
Потребляемый ток: 10 мкА
Точность: ±0,5 ºC
Разрешение: 12 бит (0,0625 ºC)
Рабочий диапазон: от –40 ºC до +125 ºC

TMP102 — это 6-контактная микросхема, как показано на рис. 8.9.
описания контактов:

	Название пина	**Описание**
1	SCL	I2C тактовая линия
2	GND	земля источника питания
3	ALERT	Предупреждение о перегреве. Выход с открытым стоком. Нужен подтягивающий резистор
4	ADD0	Выбор адреса
5	V+	питание 5 В+
6	SDA	I2C линия данных

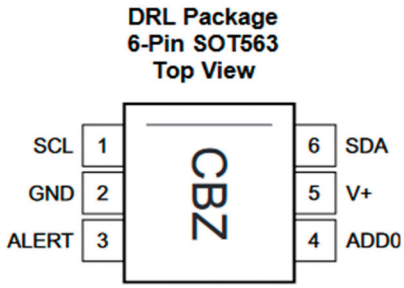


Рис.8.9: Расположение выводов TMP102.

TMP102 имеет следующие режимы работы:

- **Непрерывное преобразование**: по умолчанию внутренний АЦП преобразует температуру в цифровой формат с частотой преобразования по умолчанию 4 Гц и временем преобразования 26 мс. Скорость преобразования можно выбрать с помощью битов CR1 и CR0 регистра конфигурации: 0,25 Гц, 1 Гц, 4 Гц (по умолчанию) и 8 Гц. В этом проекте по умолчанию используется 4 Гц.
- **Расширенный режим**: Бит EM регистра конфигурации выбирает нормальный режим (EM = 0) или расширенный режим (EM = 1). Вобычном режиме (режим по умолчанию) преобразованные данные имеютразмер 12 бит. Расширенный режим используется, если температура выше128 ºC и преобразованные данные имеют размер 13 бит. В этом проекте используется обычный режим.
- **Режим отключения**: этот режим используется для экономии энергии, когда потребление тока снижается до уровня менее 0,5 мкА. Режим выключения вводится, когда бит регистра конфигурации SD = 1. Режим по умолчанию — нормальная работа (SD = 0).
- **Однократное преобразование**: Установка бита регистра конфигурации OS в 1 выбирает однократный режим, который является режимом однократного преобразования. Режим по умолчанию — непрерывное преобразование (OS = 0).
- **Режим термостата**: Этот режим указывает, следует ли работать в режиме компаратора (TM = 0) или в режиме прерывания (TM = 1). По умолчанию используется режим компаратора. В режиме компаратора вывод Alert активируется, когда температура равна или превышает значение в регистре THIGH, и остается активным до тех пор, пока температура не упадет ниже TLOW. В режиме прерывания контакт Alert активируется, когда температура превышает значение THIGH или становится ниже значения TLOW. Вывод предупреждения очищается, когда хост-контроллер считывает регистр температуры.
  Регистр указателя выбирает различные регистры в микросхеме, как показано в таблице 8.3. Старшие 6 бит этого регистра равны 0.

Регистр указателя выбирает различные регистры в микросхеме, как показано в таблице 8.3. Старшие 6 бит этого регистра равны 0.

P1	P0	Выбор регистра
0	0	Регистр температуры (только чтение)
0	1	Регистр конфигурации
1	0	регистр TLOW
1	1	THIGH регистр

Table 8.3: Pointer register bits.

В таблице 8.4 показаны биты регистра температуры в нормальном режиме (EM = 0).

BYTE 1:

D7	D6	D5	D4	D3	D2	D1	D0
T11	T10	T9	T8	T7	T7	T5	T4

BYTE 2:

D7	D6	D5	D4	D3	D2	D1	D0
T3	T2	T1	T0	0	0	0	0

Таблица 8.4: Биты регистра температуры

В таблице 8.5 показаны биты регистра конфигурации. Конфигурация битов по умолчанию при включении питанияпоказано в таблице.

BYTE 1:

D7	D6	D5	D4	D3	D2	D1	D0
OS	**R1**	**R0**	**F1**	**F0**	**POL**	**TM**	**SD**
0	1	1	0	0	0	0	0

BYTE 2:

D7	D6	D5	D4	D3	D2	D1	D0
CR1	**CR0**	**AL**	**EM**	**0**	**0**	**0**	**0**
1	0	1	0	0	0	0	0

Таблица 8.5: Биты регистра конфигурации.

Бит полярности (POL) позволяет пользователю настроить полярность выхода сигнала тревоги. Если установлено значение 0 (по умолчанию), вывод Alert становится активным при низком уровне. Если установлено значение 1, вывод Alert становится активным с высоким уровнем.

Адрес устройства по умолчанию — 0x48. TMP102 доступен в виде модуля (модуля коммутации), как показано на рис. 8.10. Адрес регистра температуры — 0x00, и его следует отправлять после отправки адреса устройства. Затем следует команда чтения, в которой 2 байта считываются из TMP102. Эти 2 байта содержат данные о температуре.

Последовательность считывания температуры следующая:

- Мастер отправляет адрес устройства 0x48 с R/W, установленным на 0.
- Устройство отвечает ACK.
- Мастер отправляет адрес регистра температуры 0x00.
- Устройство отвечает ACK.
- Мастер повторно отправляет адрес устройства 0x48 с битом R/W, установленным на 1.
- Master reads the upper byte of temperature data.
- Устройство отправляет ACK.
- Мастер считывает младший байт данных температуры.
- Устройство отправляет ACK.
- Мастер посылает по шине условие остановки.

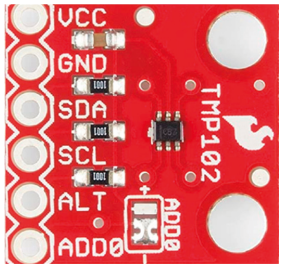


Рис.8.10: TMP102 как модуль.

Блок-схема: На рис. 8.11 показана блок-схема проекта.

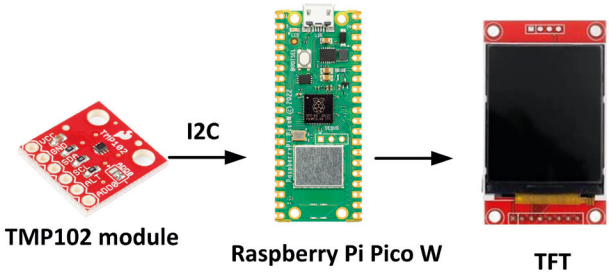


Рис.8.11: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рис.8.12. На линиях шины I2C TMP102 имеются встроенные подтягивающие резисторы, поэтому нет необходимости использовать внешние резисторы.

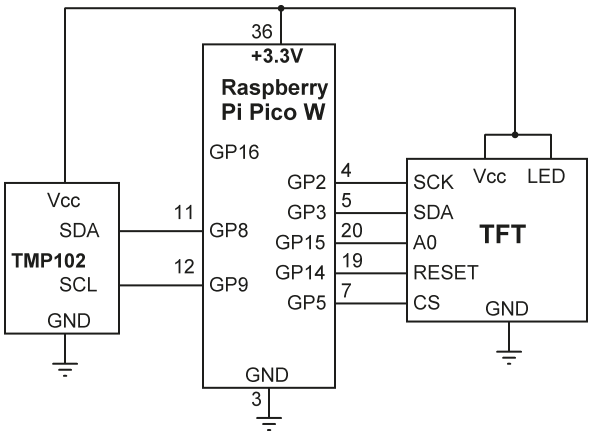


Рис.8.12: Принципиальная схема проекта.

Листинг программы: На рис. 8.13 представлен листинг программы (Программа: **TMP102.py**). В начале программы определяются I2C-адрес TMP102 и адреса регистров указателя. Регистр указателя устанавливается в 0, чтобы выбрать регистр температуры.
Программа выполняется внутри цикла **while** и вызывает функцию **Read()**. Эта функция считывает температуру с TMP102, преобразует ее в положительные(или отрицательные) градусы Цельсия, а затем возвращает показания в основную программу. Два прочитанных байта объединяются для формирования 12-битных данных о температуре в переменной **temp**. Если температура отрицательна, то она находится в форме дополнения до 2, берется ее дополнение и прибавляется 1, чтобы найти истинное отрицательное значение. Умножив **temp** на **LSB**, вы получите температуру в градусах Цельсия. Температура отображается на ЖК-дисплее I2C в виде числа с плавающей запятой. В таблице 8.6 показан формат вывода данных о температуре. Давайте рассмотрим два примера:

Пример 1: Измеренное значение = 0011 00100000 = 0x320 = 800 десятичных разрядов. Это положительная температура, поэтому температура составляет 800 × 0,0625 = +50 ºC.

Пример 2: Измеренное значение = 1110 01110000 = 0xE70. Это отрицательная температура, дополнение равно 0001 10001111, добавление 1 дает 0001 10010000 = 400 десятичных знаков. Температура 400 × 0,0625 = 25 или –25 ºC.

Температура	Цифровой выход (Bin)	Цифровой выход (Hex)
128	011111111111	7FF
100	011001000000	640
50	001100100000	320
0.25	000000000100	004
-0.25	111111111100	FFC
-25	111001110000	E70
-55	110010010000	C90

Таблица *8.6:* Вывод данных для некоторых показаний температуры.

```
#-------------------------------------------------------------
#              TMP102 ДАТЧИК ТЕМПЕРАТУРЫ
#              ============================
#
# В этом проекте микросхема датчика температуры, совместимая с I2C типа TMP102,
#  подключена к Raspberry Pi Pico. Показания температуры
# отображаются на ЖК-дисплее I2C.
#
# Автор: Доган Ибрагим
# Файл   : TMP102.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
import utime
from ST7735 import TFT
from sysfont import sysfont
from machine import SPI, Pin, I2C

#
# Инициализировать SPI
#
spi = SPI(0, baudrate=20000000, polarity=0, phase=0,
sck=Pin(2), mosi=Pin(3), miso=Pin(4))

tft = TFT(spi, 15, 14, 5)
tft.initg()
tft.rgb(True)

#
# Инициализировать SPI
#
i2c = machine.I2C(0, scl=Pin(9), sda=Pin(8), freq=100000)
print("i2c address=",i2c.scan())
```

```
Device_Address = 0x48                  # I2C-адрес TMP102
PointerReg = 0                         # регистр TMP102

#
# Эта функция считывает температуру, извлекает градусы Цельсия
# и возвращает температуру основной вызывающей программе.
#
def Read():
    data = [0, 0]
    LSB = 0.0625
#    data = i2c.readfrom_mem(Device_Address, PointerReg, 2)
    temp = (data[0] << 4) | (data[1] >> 4)
    if temp > 0x7FF:
        temp = (~temp) & 0xFF
        temp = temp + 1
        temperature = -temp * LSB
    else:
        temperature = temp * LSB
    return(temperature)

#
# Основная программа считывает и отображает температуру каждые 3 секунды
#
while True:
    Temperature = Read()
    tft.fill(TFT.WHITE)                                  # Очистить экран
    tft.rect((5, 10), (100, 120), TFT.RED)
    tft.text((17, 30),"TEMPERATURE",TFT.BLUE,sysfont,1.1,nowrap=True)
    tft.text((15, 60), "Temp:{:.2f}C".format(Temperature), TFT.BLUE,
    sysfont, 1.1, nowrap=True)
    utime.sleep(3)
```

Рис.*8.13:* Программа:**TMP102.py**.

Глава 9 • OLED-дисплеи

9.1 Обзор

Данные обычно вводятся в систему с помощью выключателя, небольшой клавиатуры или полноразмерной клавиатуры. Данные обычно отображаются с помощью индикатора, такого как один или несколько светодиодов, 7-сегментные дисплеи, TFT, OLED или простые ЖК-дисплеи. ЖК-дисплеи имеют то преимущество, что они дешевы и могут отображать как буквенно-цифровые, так и графические данные. Некоторые ЖК-дисплеи имеют длину 40 или более символов и могут отображать данные в несколько строк. Некоторые другие ЖК-дисплеи можно использовать для отображения графических изображений (графические ЖК-дисплеи или просто GLCD), таких как анимация. Некоторые дисплеи являются одноцветными или многоцветными, в то время как некоторые другие имеют подсветку, чтобы их можно было просматривать в условиях слабого освещения. Другие дисплеи, такие как TFT и OLED, имеют то преимущество, что они могут отображать цветные графические данные. Кроме того, OLED-дисплеи дешевы и не требуют фонового освещения, поскольку каждый пиксель на дисплее способен излучать свет.

В этой главе используется OLED-дисплей с диагональю 0,91 дюйма, совместимый с интерфейсом I 2 C (рис. 9.1). Это небольшой недорогой дисплей с разрешением 128×32 пикселя. Он имеет следующие основные функции:

- Драйвер: SSD1306
- Интерфейс: I^2C
- Цвет дисплея: белый
- Разрешение: 128×32
- Угол обзора: >160°
- Рабочее напряжение: от +1,65 до +3,3 В.
- Размер: 36 × 12,5 (мм)

Рис.9.1: 0,91-дюймовый OLED-дисплей.

Координаты XY используемого OLED-дисплея показаны на рис. 9.2.

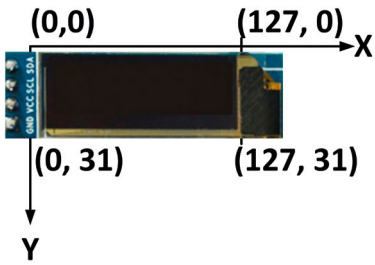


Рис.9.2: Координаты OLED-дисплея.

9.2 Установка программного драйвера SSD1306

Библиотека драйверов SSD1306 должна быть загружена на Raspberry Pi Pico W, прежде чем можно будет использовать функции библиотеки. Файл библиотеки называется **ssd1306.py** и доступен в Интернете. Этот файл также доступен на веб-сайте книги. Скопируйте файл на Raspberry Pi Pico с точным именем **ssd1306.py**.

9.3 Аппаратный интерфейс

OLED-дисплей подключается к Raspberry Pi Pico с помощью следующих контактов (см. рис. 9.3):

OLED дисплей	**Raspberry Pi Pico W**
SCL	GP9 (I2C0 SCL)
SDA	GP8 (I2C0 SDA)
GND	GND
VCC	+3.3V

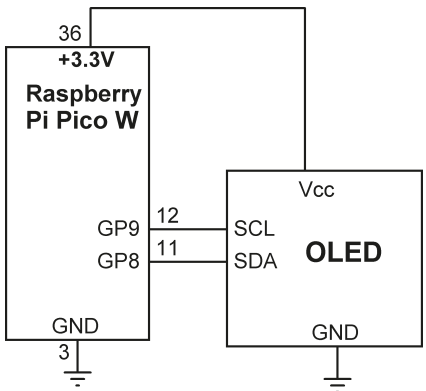


Рис.9.3: OLED — интерфейс Pico.

Теперь вы можете использовать библиотечные функции OLED-дисплея в своих проектах.

9.4 Отображение текста на OLED

Программа (OLEDText.py), показанная на рис. 9.4, отображает двухстрочный текст, начиная с пятого столбца дисплея. Обратите внимание, что в начале программы импортируется библиотечный модуль **ssd1306**. Затем настраиваются соединения I2C SDA и SCL OLED-дисплея.

Программа отображает I2C-адрес дисплея. Хотя эта адресная информация не используется в программе, она показывает, что OLED распознается Pico как устройство I2C. оператор **oled.text()** отображает текст, начиная с заданных координат пикселя. Обратите внимание, что оператор **oled.show()** должен быть включен, чтобы обновить отображение новыми данными.

```
#-------------------------------------------------------------
#                       OLED-ДИСПЛЕЙ
#                      ==============
#
# Это программа для тестирования OLED-дисплеев.
# На OLED-дисплее отображается следующий текст
#
#   Micropython (at 5,5)
#   Пример (на 5,25)
#
# Автор: Доган Ибрагим
# Файл   : OLEDText.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import machine

i2c = I2C(0, scl=Pin(9),sda=Pin(8),freq=100000)
oled = SSD1306_I2C(128,32,i2c)         # Строки и столбцы

#
# Display device address (not used)
#
print("I2C Address: " + hex(i2c.scan()[0]).upper())

oled.fill(0)
oled.text("MicroPython",5,5)                    # MicroPython, в пикселях (5,10)
oled.text("Example",5,25)                      # "Пример" в пикселях (5,25)
oled.show()                                    # Дисплей
print("End")
```

Рис.*9.4:* Программа:**OLEDText.py**.

На рис. 9.5 показан дисплей при запуске вышеуказанной программы.

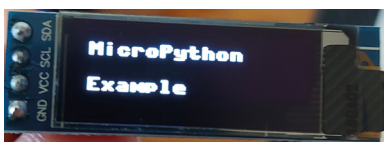


Рис.*9.5:* Дисплей.

При отображении текста проще работать с отображаемыми строками и столбцами. С OLED-дисплеем, использующим набор символов по умолчанию, мы можем иметь до 15 символов по горизонтали и 3 символа по экрану, то есть при отображении текста с помощью 0,91-дюймового OLED-дисплея у вас может быть 3 строки и 15 столбцов. Вы можете изменить приведенный выше код, создав функцию с именем **display_text**, чтобы строки и столбцы вводились для отображения текста, а ненеобработанных чисел в пикселях, как в случае с приведенным выше примером:

```
       def display_text(text, line, column):
              oled.text(text, 8*column, 8*line)
```

Функция отображения **fill(0)** устанавливает все пиксели в черный цвет и может использоваться для очистки экрана, если фон черный. Точно так же **fill(1)** устанавливает все пиксели в белый цвет и может использоваться для очистки экрана, если фон белый.

С помощью функции **oled.text** вы также можете указать цвет текста в качестве третьего аргумента функции, где 0 = черный, а 1 = белый.

Теперь вы можете очистить экран и отобразить текст в строке 1 и строке 3, как показано на рис. 9.6 (программа: **OLEDText2.py**).

```
#-------------------------------------------------------------
#                              OLED-ДИСПЛЕЙ
#                              =============
#
# Это программа для тестирования OLED-дисплеев.
# На OLED-дисплее отображаются следующие тексты:
#
#   Micropython (строка 1, столбец 1)
#   Пример (строка 3, столбец 4)
#
# Автор: Доган Ибрагим
# Файл   : OLEDText2.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import machine


i2c = I2C(0, scl=Pin(9),sda=Pin(8),freq=100000)
oled = SSD1306_I2C(128,32,i2c)      # Строки и столбцы


def display_text(text, line, column):
       oled.text(text, 8*column, 8*line)

oled.fill(0)                        # Очистить дисплей
display_text("MicroPython",1,1)     # "MicroPython" в строке 1 столбец 1
display_text("Program",3, 4)        # "Программа" в строке 3, столбце 4
oled.show()                         # Дисплей
```

Рис. *9.6:* Программа*:* **OLEDText2.py**

В приведенном выше примере текст **MicroPython**отображается в строке 1, начиная с столбца 1, текст **Program**отображается в строке 3, начиная с столбца 4 (рис. 9.7).

Рис.*9.7:* Дисплей.

9.5 Отображение общих фигур

Важно иметь возможность рисовать различные фигуры с помощью графического ЖК-дисплея, такого как OLED.
Вот некоторые из функций рисования (обратите внимание, что цвет равен 1 для белого и 0 для черного):

rect(xtop, ytop, width, height, color): эта функция рисует пустой прямоугольник, где **xtop** и **ytop** — это левые верхние углы прямоугольника в пикселях.

fill_rect(xtop, ytop, width, height, color):эта функция рисует закрашенный прямоугольник, где **xtop** и **ytop** — это левые верхние углы прямоугольника в пикселях.

line(xstrt, ystrt, xend, yend, color): эта функция рисует линию, начиная с позиции пикселя **(xstrt, ystrt)** до **(xend, yend)**.

hline(xstrt, ystrt, length, color): эта функция рисует горизонтальную линию, начиная с позиции в пикселях **(xstrt, ystrt)** и имеющей указанную длину.

vline(xstrt, ystrt, length, color): эта функция рисует вертикальную линию, начиная с позиции в пикселях **(xstrt, ystrt)** и имеющей указанную **length**.

В качестве примера нарисуем пустой прямоугольник, линию, горизонтальную линию и вертикальную линию. Программа для этого примера показана на рис. 9.8 (программа: **OLEDshapes.py**). Рисуются следующие фигуры:

```
oled.rect(2, 2, 120, 30, 1)          # Прямоугольник
oled.line(5, 5, 30, 25, 1)           # Линия
oled.hline(20, 10, 50, 1)            # Горизонтальная линия
oled.vline(50, 15, 12, 1)            # Вертикальная линия
```

```
#-----------------------------------------------------------
#                  ФИГУРЫ  НА OLED
#                  ===============
#
# Эта программа отображает различные фигуры на OLED
```

```
#
# Автор: Доган Ибрагим
# Файл   : OLEDshapes.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import machine
oled = SSD1306_I2C(128,32,i2c)         # Строки и столбцы
i2c = I2C(0, scl=Pin(9),sda=Pin(8),freq=100000)


oled.fill(0)                           # Очистить экран
oled.rect(2, 2, 120, 30, 1)            # Прямоугольник
oled.line(5, 5, 30, 25, 1)             # Линия
oled.hline(20, 10, 50, 1)              # Горизонтальная линия
oled.vline(50, 15, 12, 1)              # Вертикальная линия
oled.show()                            # Показать
```

Рис. *9.8:* Программа: **OLEDshapes.py**.

На рис. 9.9 показан дисплей при запуске вышеуказанной программы

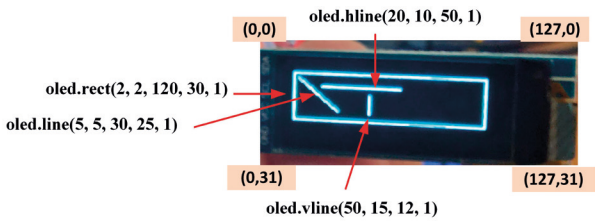


Рис.*9.9:* Дисплей.

9.6 Другие полезные функции

Некоторые другие полезные функции OLED:

pixel(x, y, colour): установите для пикселя **(x, y)** указанный цвет (1=белый, 0=черный)

invert(1): Эта функция инвертирует цвета так, что черный становится белым, а белый наоборот.

invert(0): вернуться к исходным цветам

poweroff(): выключение дисплея (пиксели остаются в памяти)

power(on): Включение дисплея. Пиксели перерисовываются

contrast(0): затемнить экран

contrast(1): яркий экран

rotate(True): эта функция поворачивает экран на 180 градусов.

rotate(False): повернуть экран на 0 градусов

scroll(horiz, vert): эта функция прокручивает пиксели по горизонтали вправо и по вертикали вниз.

Пример — круг

Напишите функцию для рисования круга с помощью функции пикселя. Координаты (x, y) точкицентральная точка, радиус и цвет (черный = 0 или белый = 1) круга должны быть переданыв качестве аргументов. Установите координаты (x, y) на 64, 30-пиксельные позиции соответственно, а радиус до 25 пикселей.

Решение

В этом примере вы должны использовать следующие две формулы для рисования круга:

$$xi+1 = xi + r * \cos(angle)$$
$$yi+1 = yi + r * \sin(angle)$$

Где угол принимает значения от 0 до 359, а r — положение центрального пикселя. Рис.9.10 показывает требуемую программу (программа**: OLEDcircle**). Когда эта программа запускается, центр окружности в точке (64, 16) и радиус 15 пикселей рисуются, как показано на рисунке 9.11.

```
#================================================================
#           НАРИСУЙТЕ КРУГ, ИСПОЛЬЗУЯ ПИКСЕЛИ
#           ====================================
#
# Эта программа использует функцию circle для рисования круга
# с заданными координатами центра и радиусом в пикселях.
#
# Автор: Доган Ибрагим
# Файл   : OLEDcircle.py
# Дата: октябрь 2022 г.
#================================================================
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import machine
import math

i2c = I2C(0, scl=Pin(9),sda=Pin(8),freq=100000)
oled = SSD1306_I2C(128,32,i2c)      # Строки и столбцы
```

```
#
# Эта функция рисует круг, используя пиксели
#
def circle(x, y, r, colour):
    for i in range(360):
       angle = i
       xi = r*math.cos(math.radians(angle))
       yi=r*math.sin(math.radians(angle))
       oled.pixel(int(x+xi),int(y+yi),colour)

#
#  Радиус окружности в точке (64, 10), радиус 15 пикселей
#
oled.fill(0)                              # Очистить экран
circle(64, 16, 15, 1)                     # Circle spev
oled.show()                               # Рисовать
```

Figure 9.10: Program: OLEDcircle.

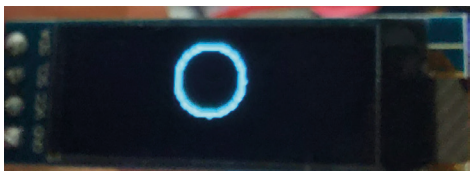


Figure 9.11: Displaying the circle.

Пример — квадрат
Напишите функцию для рисования квадрата. Верхние левые координаты (x, y), длина (или высота) квадрата и его цвет должны передаваться в качестве аргументов функции. Установите верхние левые координаты на (15, 15) и длину на 40 пикселей.

Решение
Программа показана на рис. 9.12 (программа: **OLEDsquare**). Библиотечная функция **rect** используется с одинаковыми значениями длины и высоты, так что рисуется квадрат.

```
#=======================================================================
#                        НАРИСУЙТЕ КВАДРАТ
#                        ==================
#
# Эта программа рисует квадрат, где верхние левые координаты x и y
#  и длина нужного квадрата передаются
#  в качестве аргументов функции квадрата
#
# Автор: Доган Ибрагим
# Файл   : OLEDsquare.py
```

```
# Дата: октябрь 2022 г.
#================================================================
from machine import Pin, I2C
import ssd1306
import machine
import math

i2c = machine.SoftI2C(scl=Pin(9),sda=Pin(8),freq=100000)  #Инициализация I²C
oled=ssd1306.SSD1306_I2C(128,32,i2c)                      # Строки и столбцы

#
# Эта функция рисует квадрат
#
def square(xtop, ytop, length, colour):
    oled.rect(xtop, ytop, length, length, colour)

oled.fill(0)                                    # Очистить экран
square(10, 10, 20, 1)                           # Квадрат
oled.show()                                     # Рисовать
```

Рис.*9.12:* Программа*:* **OLEDsquare.**

Рис.*9.13:* Отображение квадрата*.*

Пример — фигура прокрутки

Напишите программу, которая прокручивает квадрат по горизонтали на 5 пикселей каждые 0,5 секунды. Установите верхние левые координаты на (10, 10) и длину на 20 пикселей.

Решение

Необходимая программа показана на рис. 9.14 (программа: **ScrollShape**). Функция прокрутки OLED используется для прокрутки фигуры по горизонтали.

```
#================================================================
#                        ПРОКРУТКА ФИГУРЫ
#                        ================
#
# Эта программа рисует квадрат, где верхние левые координаты x и y
# и длина требуемого квадрата передаются
# в качестве аргументов функции Square. Эта квадратная форма затем
#  прокручивается вправо по экрану каждые 0,5 секунды.
#
# Автор: Доган Ибрагим
```

```
# Файл : ScrollShape.py
# Дата: октябрь 2022 г.
#=============================================================
from machine import Pin, I2C
import ssd1306
import machine
import utime

i2c = machine.SoftI2C(scl=Pin(9),sda=Pin(8),freq=100000)
oled=ssd1306.SSD1306_I2C(128,32,i2c)           # Строки и столбцы

#
# Эта функция рисует квадрат
#
def square(xtop, ytop, length, colour):
    oled.rect(xtop, ytop, length, length, colour)

oled.fill(0)                                   #  Очистить экран
square(10, 10, 20, 1)                          #  Квадрат
oled.show()

#
# Прокручивать фигуру по горизонтали экрана каждые 0,5 с
#
for i in range(0, 90, 5):
    oled.scroll(5, 0)
    oled.show()
    utime.sleep(0.5)
```

Рис.*9.14:* Программа:**ScrollShape**.

9.7 Проект 1: Счетчик секунд

Описание: Это проект счетчика секунд. Кнопка подключена к одному из портов ввода-вывода. Когда она нажата, программа начинает считать каждую секунду и отображает счет на OLED-дисплее.

Цель: Цель этого проекта — показать, как можно обновлять данные на OLED.

Принципиальная схема:Как показано на рис.9.15, кнопка подключена к контакту порта GP15 (контакт 20). Состояние вывода равно логической 1 и переходит в логическую 0 при нажатии кнопки.

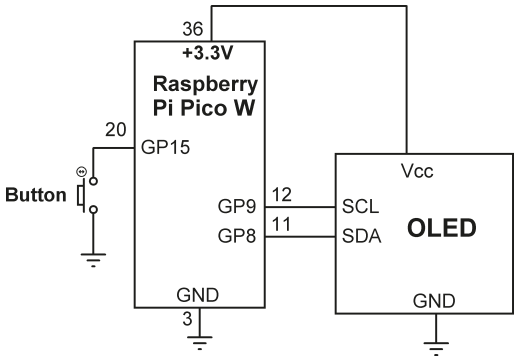


Рис.9.15: Принципиальная схема проекта.

Листинг программы: Рис.9.16 показывает листинг программы (программа: **SecondsCounter**). В начале программы импортируются библиотеки, используемые в программе, определяется функция **display_text** и настраивается OLED. Затем GP15 настраивается как вход, к которому подключается кнопка. Затем программа отображает заголовок «Нажмите кнопку…» и ждет, пока кнопка не будет нажата. Когда кнопка нажата, оставшаяся часть программы запускается в цикле **while**, где переменная **Count** увеличивается каждую секунду и отображается на дисплее.

```
#-----------------------------------------------------------
#               СЧЕТЧИК СЕКУНД
#               ===============
#
# В этой программе кнопка подключена к IO23.
# При нажатии кнопки программа начинает считать
# каждую секунду и отображает счет на OLED-дисплее
#
# Автор: Доган Ибрагим
# Файл   : SecondsCounter.py
# Дата: октябрь 2022 г.
#-----------------------------------------------------------
from machine import Pin, I2C
import ssd1306
import machine
import utime

#
#  Функция для отображения текста в указанной строке и столбце
#
def display_text(text, line, column):
       oled.text(text, 8*column, 8*line)

i2c = machine.SoftI2C(scl=Pin(9),sda=Pin(8),freq=100000)
```

```
oled=ssd1306.SSD1306_I2C(128,32,i2c)          # Строки и столбцы

button=Pin(15,Pin.IN, pull=Pin.PULL_UP)       # Кнопка на GP15
count = 0                                     # количество = 0

oled.fill(0)                                  # Очистить экран
display_text("Press Button...", 2, 1)         # Показать заголовок
oled.show()

while(button.value() == 1):                   # Ждем кнопку
    pass

while True:
    oled.fill(0)                              # Очистить экран
    display_text("Count: ", 2, 1)             # Показать "Количество:-Count:"
    count = count + 1                         # Увеличение счетчика
    display_text(str(count), 2, 7)            # Показать счет
    oled.show()
    utime.sleep(1)                            # ждать 1 секунду
```

Рис. *9.16:* Программа: **SecondsCounter**.

На рис. 9.17 показан дисплей до и после нажатия кнопки.

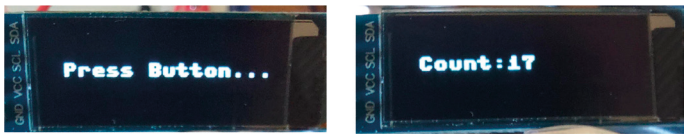


Рис. *9.17:* Дисплей до и после нажатия кнопки.

9.8 Проект 2: Рисование растровых изображений

Описание: Существует множество приложений, в которых вы можете рисовать изображения, например растровые изображения. В этом разделе приведены примеры, показывающие, как растровые изображения можно рисовать на OLED.

Простые фигуры

В качестве примера давайте нарисуем на OLED-дисплее несколько сердечек. Растровое изображение сердечки показано ниже. Например, эту фигуру можно создать с помощью программы Paint:

```
ICON = [
[ 0, 0, 0, 0, 0, 0, 0, 0, 0],
[ 0, 1, 1, 0, 0, 0, 1, 1, 0],
[ 1, 1, 1, 1, 0, 1, 1, 1, 1],
[ 1, 1, 1, 1, 1, 1, 1, 1, 1],
[ 1, 1, 1, 1, 1, 1, 1, 1, 1],
```

```
[ 0, 1, 1, 1, 1, 1, 1, 1, 0],
[ 0, 0, 1, 1, 1, 1, 1, 0, 0],
[ 0, 0, 0, 1, 1, 1, 0, 0, 0],
[ 0, 0, 0, 0, 1, 0, 0, 0, 0],
]
```

Листинг программы: на рис. 9.18 показана программа (программа: **OLEDheart**) которая рисует на дисплее 6 сердечек, как показано на рис. 9.19.

```
#-------------------------------------------------------------
#            ОТОБРАЖЕНИЕ РАСТРОВОГО ИЗОБРАЖЕНИЯ
#            ========================================
#
# В этой программе изображения сердечек отображаются на OLED-дисплее
#
# Автор: Доган Ибрагим
# Файл   : OLEDheart.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
from machine import Pin, I2C, ADC
import ssd1306
import machine

i2c = machine.SoftI2C(scl=Pin(9),sda=Pin(8),freq=100000)
oled=ssd1306.SSD1306_I2C(128,32,i2c)    #  Строки и столбцы

#
# Растровое изображение сердечка
#
ICON = [
[ 0, 0, 0, 0, 0, 0, 0, 0, 0],
[ 0, 1, 1, 0, 0, 0, 1, 1, 0],
[ 1, 1, 1, 1, 0, 1, 1, 1, 1],
[ 1, 1, 1, 1, 1, 1, 1, 1, 1],
[ 1, 1, 1, 1, 1, 1, 1, 1, 1],
[ 0, 1, 1, 1, 1, 1, 1, 1, 0],
[ 0, 0, 1, 1, 1, 1, 1, 0, 0],
[ 0, 0, 0, 1, 1, 1, 0, 0, 0],
[ 0, 0, 0, 0, 1, 0, 0, 0, 0],
]

#
# Очистка и отображение 6 сердечек на дисплее
#
oled.fill(0)
```

```
for i in range(0, 120, 20):
 for y, row in enumerate(ICON):
    for x, c in enumerate(row):
        oled.pixel(x+i, y+20, c)

oled.show()
```

Рис.*9.18:* Листинг программы.

Рис.*9.19:* Отображаемые программой фигурки сердечек.

Создание и отображение растрового изображения

В этом разделе вы создадите растровое изображение размером 128×32, а затем отобразите его на дисплее. Образ, который вы создадите, — это буква A. Шаги приведены ниже:

Создание образа

Вы можете использовать программы для создания изображений, такие как Photoshop, GIMP, Microsoft Windows Paint и так далее. В этом примере вы будете использовать программу Microsoft Paint для создания изображения.

- Запустите программу **Paint**.

- Щелкните **Главная** → **Изменить размер** и щелкните **Пиксели**. Установите **По горизонтали** 128 пикселей и **По вертикали** 32 пикселя как показано на рис.9.20. Нажмите **OK**.

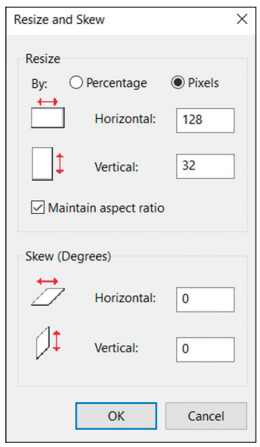


Рис. *9.20:* Установите пиксели на *128×32.*

- Нажмите **Просмотр**, а затем нажмите **Увеличить**, чтобы увеличить сетку.

- Нарисуйте нужную форму с помощью мыши. В этом проекте буква **A** нарисована от руки, как показано на рис. 9.21.

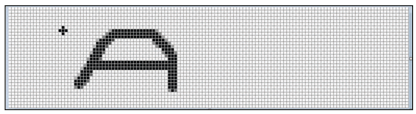


Рис.9.21: Нарисуйте требуемую фигуру.

- Сохраните файл, например, как **LetterA.png** в папке на вашем ПК. Выйдите из программы Paint.

- Теперь вам нужно преобразовать изображение в соответствующий формат. Шаги:
- Запустите программу **GIMP** и откройте файл **LetterA.png** (рис. 9.22).

Рис.9.22: Откройте файл в GIMP.

• Щелкните **Изображение** → **Режим** → **Индексированный** выберите **Использовать** черно-белое (1-битное палитру (рис. 9.23). Щелкните **Преобразовать**.

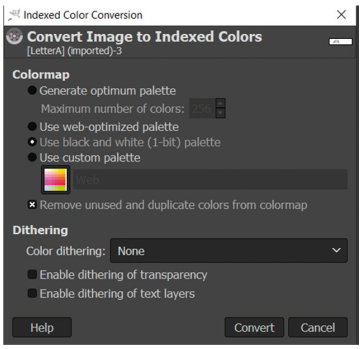


Рис.9.23: Выберите 1-битную палитру с черным и белым цветом.

• Щелкните **Файл** → **Экспортировать как** а затем изменитерасширение файла на **pbm**в левом верхнем углу экрана. Щелкните **Export**.

• Выберите **Raw** в качестве формата данных **Data formatting** и нажмите **Export**. Новый файл под названием **LetterA.pbm** будет находиться в выбранной вами папке. Выйдите из **GIMP**.

• Теперь скопируйте файл изображения с ПК на Pico. Откройте Thonny, щелкните **This Computer** слева и выберите файл **LetterA.pbm** (рис.9.24).

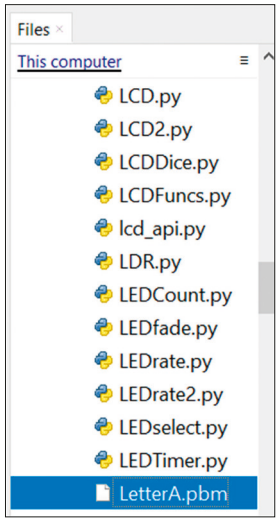


Рис.9.24: Выберите файл **LetterA.pbm**.

• Щелкните правой кнопкой мыши выбранный файл и выберите **Upload to/**- Загрузить в/. Отметьте Thonny и нажмите **File → Open** и выберите **Raspberry Pi Pico,** чтобы убедиться, что файл **LetterA.pbm** был скопирован на ваш Pico.

Листинг программы: Теперь, когда у вас есть файл растрового изображения, следующим шагом будет написание программы для отображения этого изображения на OLED. На рис. 9.25 показан листинг программы (программа: **OLEDbitmap**). В начале программы определяется интерфейс OLED. Затем программа открывает и читает файл изображения **LetterA.pbm**. Обратите внимание, что в этом примере используется **framebuffer** - кадровый буфер OLED. OLED-дисплей SSD1306 имеет оперативную память дисплея, которая имеет тот же размер, что и размер пикселя дисплея, и существует взаимосвязь между битами памяти и пикселями на дисплее. Объект кадрового буфера имеет тот же размер, что и ОЗУ дисплея, и этот объект позволяет программисту загружать изображение из памяти кадрового буфера перед фактической записью данных из изображения памяти в ОЗУ дисплея.
Таким образом, вы можете, например, загрузить в фреймбуфер изображение, которое хотите отобразить, а затем загрузить отображение из фреймбуфера.
Функция OLED **blit()** используется для копирования данных из памяти кадрового буфера на дисплей в виде блочной копии. Обычно на OLED-дисплее отображается белое изображение (см. рис. 9.26 с черным фоном. Вы можете инвертировать изображение, чтобы оно было черным, а фон — таким, как показано на рис. 9.26).

```
#-------------------------------------------------------------
#             ОТОБРАЖЕНИЕ РАСТРОВОГО ФАЙЛА
#             ================================
#
# В этой программе растровый файл отображается на OLED.
# Файл был создан с помощью программы Windows Paint, а затем
# отформатирован с помощью GIMP.
#
# Автор: Доган Ибрагим
# Файл   : OLEDbitmap.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
from machine import Pin, I2C
import ssd1306
import machine
import framebuf

i2c = machine.SoftI2C(scl=Pin(9),sda=Pin(8),freq=100000)
oled=ssd1306.SSD1306_I2C(128,32,i2c)     # Строки и столбцы

#
#  Очистка экрана. Открыть и прочитать файл и отобразить изображение
#
oled.fill(0)
with open("LetterA.pbm", "rb") as bitmap:
    bitmap.readline()
    bitmap.readline()
    bitmap.readline()
    bitdata = bytearray(bitmap.read())

frbuf = framebuf.FrameBuffer(bitdata, 128, 32, framebuf.MONO_HLSB)
#oled.invert(1)
oled.blit(frbuf, 0, 0)
oled.show()
```

Рис.9.25: Программа: **OLEDBitmap.py**.

На рис 9.26 показан дисплей после запуска программы.

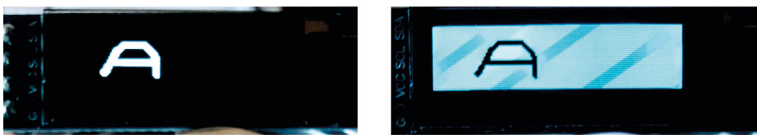


Рис.9.26: Дисплей до и после инвертирования.

9.9 Проект 3: цифровой термометр на основе OLED-дисплеев DS18B20

Описание: В этом проекте вы создадите термометр с использованием микросхемы цифрового датчика температуры DS18B20 и будете каждую секунду отображать температуру окружающей среды на OLED-дисплее.

Цель: Цель этого проекта — показать, как микросхема датчика температуры DS18B20 может использоваться в проекте и как температура может отображаться каждую секунду на OLED-дисплее.

DS18B20: это микросхема цифрового термометра, которая обеспечивает измерение температуры в диапазоне от 9 до 12 бит по Цельсию, а также включает функцию будильника с программируемыми точками срабатывания. Чип обменивается данными по шине 1-Wire. Он имеет 64-битный последовательный код, что позволяет нескольким DS18B20 работать на одной и той же шине 1-Wire.

Основные характеристики микросхемы DS18B20:

- Измеряет температуру от –55 ºC до +125 ºC.
- Точность ±0,5 ºC в диапазоне от –10 ºC до +85 ºC.
- Программируемое разрешение от 9 до 12 бит.
- Не требуются внешние компоненты.
- Имеет уникальный 64-битный серийный код, хранящийся в ПЗУ микросхемы.

Это 3-контактный чип со следующими определениями контактов:

- VCC: плюс питания (+3,3 В)
- DQ: пин ввода/вывода
- GND: Минус источника питания.

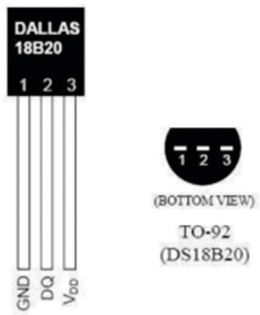


Рис.9.27: Чип DS18B20.

Блок-схема: На рис. 9.28 показана блок-схема.

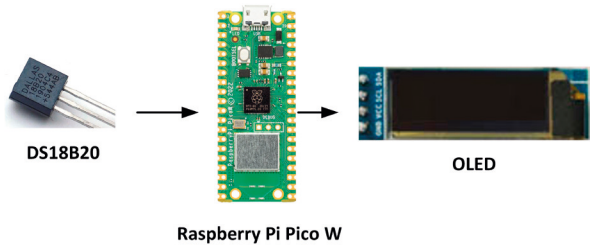


Рис.9.28: Блок-схема проекта.

Принципиальная схема: На рис.9.29 показана принципиальная схема проекта. Вывод DQ модуля DS18B20 подключен к выводу GP15 Pico.

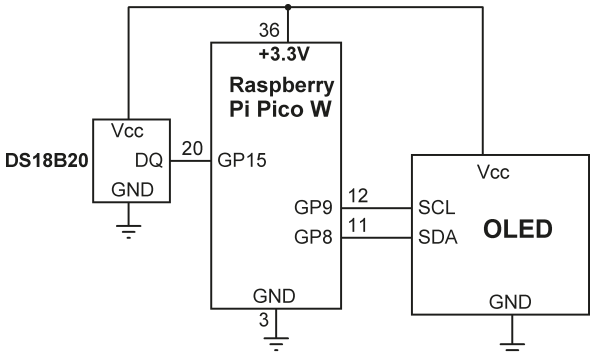


Рис.*9.29:* Принципиальная схема проекта.

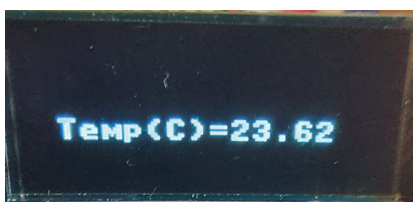
Листинг программы: На рис. 9.30 показан листинг программы (программа: OLEDtemp). В начале программы определяется конфигурация OLED и подключение к микросхеме DS18B20. Затем с помощью оператора **while** формируется бесконечный цикл. Внутри этого цикла температура считывается и отображается на OLED-дисплее каждую секунду.

```
#-----------------------------------------------------------
#                        ЦИФРОВОЙ ТЕРМОМЕТР
#                       ======================
#
#  В этой программе цифровой датчик температуры DS18B20
#  подключается к макетной плате. Температура окружающей среды
#  измеряется каждую секунду и отображается на OLED-дисплее.
#
# Автор: Доган Ибрагим
# Файл   : OLEDtemp.py
# Дата: октябрь 2022 г.
#-----------------------------------------------------------
from machine import Pin, I2C
import ssd1306
```

```
import machine
import utime
import onewire, ds18x20

#
#  Функция для отображения текста в указанной строке и столбце
#
def display_text(text, line, column):
       oled.text(text, 8*column, 8*line)

i2c = machine.SoftI2C(scl=Pin(9),sda=Pin(8),freq=100000)
oled=ssd1306.SSD1306_I2C(128,32,i2c)           # Строки и столбцы

Temp = machine.Pin(15)                         # DS18B20 на GP15
ds = ds18x20.DS18X20(onewire.OneWire(Temp))

#
# Считайте температуру и отобразите на OLED
#
while True:
    roms=ds.scan()                             # Сканировать чип
    ds.convert_temp()                          #  Преобразование температуры
    utime.sleep_ms(750)                        # Ожидание преобразования

    for rom in roms:
        temperature = ds.read_temp(rom)        # Чтение температуры

    oled.fill(0)                               # Очистить дисплей
    display_text("Temp(C)= ", 3, 1)            # Отображение "Count:"
    display_text(str(temperature)[:5], 3, 9)   #  Количество отображаемых значений
    oled.show()
    utime.sleep(1)                             # Подождать 1 секунду
```

Рис.*9.30:* Листинг программы.

Типичный дисплей температуры показан на рис. 9.31, где измеренная температура составляет 23,62 ºC во время измерения.



Ри.9.31: Типичное отображение температуры.

9.10 Проект 4: Измерение частоты сердечных сокращений (пульса)

Описание: важно знать частоту сердечных сокращений, особенно во время упражнений или занятий спортом. Датчик пульса — это небольшое аналоговое устройство, которое можно легко использовать для измерения частоты сердечных сокращений. Устройство включает в себя простой оптический датчик со схемой усиления и шумоподавления, что позволяет получать надежные показания.

Датчик пульса представляет собой небольшое устройство (рис. 9.32) с 3-контактным разъемом и имеет следующие контакты:

-	ЗЕМЛЯ
S	сигнальный выход
Средний	+3.3 V

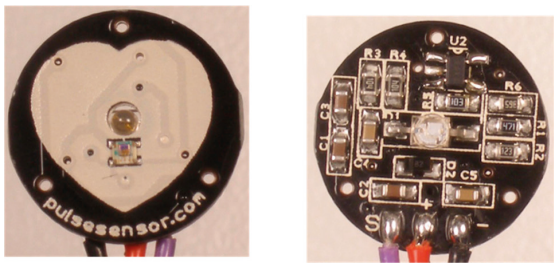


Рис.9.32: Модуль датчика пульса.

Датчик можно держать между пальцами. Кроме того, вы можете приобрести дополнительные детали, такие как клипсы для ушей или виниловые наклейки, чтобы датчик можно было прикрепить к мочке уха. На передней части датчика находится логотип в виде сердца, и это та сторона, которая соприкасается с кожей. Сзади есть небольшое отверстие, через которое светится светодиод. Сбоку размещена небольшая накладка датчика освещенности. Остальные детали монтируются сзади датчика.

В этом проекте определяется частота сердечных сокращений, которая выводится на OLED-дисплей.

Примечание: рекомендуется в целях безопасности не подключать датчик пульса к телу, пока комплект для разработки питается от сети 220В через адаптер.

Блок-схема: На рис. 9.33 показана блок-схема проекта.

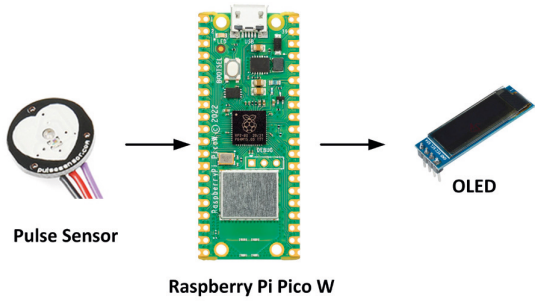


Рис.9.33: Блок-схема проекта.

Принципиальная схема: Принципиальная схема показана на рис. 9.34. Выход (контакт **S**) датчика подключен к аналоговому входу GP26 (ADCON0) Pico. Кроме того, GND и VCC подключены к контактам GND и +3,3 В соответственно.

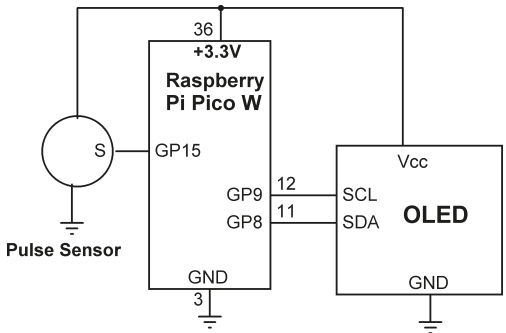


Рис.9.34: Принципиальная схема проекта.

Листинг программы: Листинг программы показан на рис. 9.35 (программа: **OLEDpulse**). В начале проекта определяются интерфейсы OLED и датчика пульса. Затем на OLED-дисплее отображается сообщение, и пользователю предлагается приложить палец к датчику пульса. Программа выполняет калибровку и вычисляет максимальное и минимальное значения, считываемые АЦП. Эти значения хранятся в переменных **maxv** и **minv** соответственно. Оставшаяся часть кода выполняется в бесконечном цикле с использованием оператора **while**. Внутри этого цикла считывается выход датчика пульса, и BPM увеличивается, если сигнал превышает пороговое значение, где этот порог используется для расчета BPM. Затем программа отображает значения, считанные с АЦП, в то время как дисплей прокручивается влево на один пиксель, чтобы можно было построить график данных в режиме реального времени. Затем рассчитывается BPM и отображается в верхней части дисплея. Обратите внимание, что эта часть экрана также прокручивается влево. Описанный выше процесс повторяется до тех пор, пока пользователь его не остановит.

В этом проекте значение BPM было откалибровано с использованием профессионального датчика пульса (например, пульсового оксиметра). Возможно, вам потребуется выполнить повторную калибровку с вашими собственными настройками. Обратите внимание, что полученные здесь результаты могут быть неточными.

```
#------------------------------------------------------------
#               ИЗМЕРЕНИЕ ЧАСТОТЫ ПУЛЬСА
#               ==========================
#
# В этом проекте модуль датчика Pukse подключен к аналоговому входу IO32
# макетной платы. Программа определяет импульсы сердечного ритма
# и отображает график на OLED-дисплее. Кроме того, рассчитывается и
# отображается количество ударов в минуту (BPM).
#
# Автор: Доган Ибрагим
# Файл   : OLEDpulse.py
# Дата: октябрь 2022 г.
#------------------------------------------------------------
from machine import Pin, I2C, ADC
import ssd1306
import machine
import utime
import math

i2c = machine.SoftI2C(scl=Pin(9),sda=Pin(8),freq=100000)
oled=ssd1306.SSD1306_I2C(128,32,i2c)   #  Строки и столбцы

AnalogIn = ADC(0)                      # Канал АЦП 0

minmax = [0]*200
def display_text(text, line, column):
       oled.text(text, 8*column, 8*line)

display_text("Heart Rate Mtr", 1, 1)   # Показать заголовок
display_text("Finger on...", 2, 1)
oled.show()
utime.sleep(3)                         # Даем пользователю время
display_text("Wait...", 3, 1)
oled.show()

#
# Рассчитать максимальное и минимальное значения, полученные АЦП
#
for i in range(200):
    minmax[i] = AnalogIn.read_u16()
    utime.sleep_ms(10)

minv = min(minmax)                     # Минимум
maxv = max(minmax)                     # Максимум
diff = maxv-minv                       # Разница
```

```
oled.fill(0)                              # Очистить дисплей
oled.show()
lasty=30
bpm = 0
cnt = 0

#
# Прочитать частоту сердечных сокращений и отобразить на OLED
#
while True:
    adcValue = AnalogIn.read_u16()     # Чтение температуры
    if adcValue >= maxv - 30000:       # Превышение порога
       bpm = bpm + 1
    y = 30 - int((30*(adcValue - minv))/diff)
    oled.line(125,lasty,126,y,1)
    oled.scroll(-1,0)                  # Прокрутить влево
    lasty=y
    oled.show()
    utime.sleep_ms(20)                 # Подождать 10 мс
    cnt = cnt + 1
    if cnt == 60:
       cnt = 0
       oled.fill_rect(0,0,30,12,0)     # Очистить область
       oled.text("%d BPM" % bpm, 30, 0)        # Показать BPM
       bpm = 0
```

Рис.*9.35:* Листинг программы.

На рис.9.36 показан дисплей при запуске программы. Ожидается, что пользователь поместит палец на датчик, когда отображается текст Finger on…. На рисунке также показан график частоты сердечных сокращений и отображение BPM.

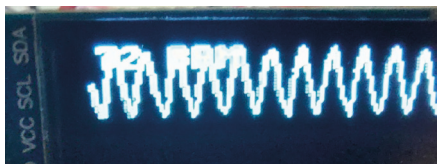


Рис.9.36: Пример экрана.

Глава 10 • Использование Bluetooth с Raspberry Pi Pico W

10.1 Обзор

Bluetooth является одним из популярных средств беспроводного обмена данными на короткие расстояния. В настоящее время многие электронные устройства, такие как смартфоны, ноутбуки, iPad, игры, гаджеты, портативные устройства для мониторинга здоровья и т. д., оснащены модулями Bluetooth. Многие люди используют Bluetooth для обмена изображениями и музыкальными файлами с помощью своих смартфонов.

Bluetooth — это протокол сопряженной связи, при котором оба устройства должны активировать свои соединения Bluetooth, а затем использовать один и тот же ключ для подключения друг к другу. Когда соединение установлено, данные могут быть отправлены в обе стороны. Не нужно беспокоиться о прямой видимости между устройствами, так как связь основана на радиоволнах.

Иногда может произойти сбой сопряжения между устройствами. Для успешного сопряжения устройств следует обратить внимание на следующие моменты:

- Убедитесь, что Bluetooth включен на обоих устройствах.
- Сделайте ваше устройство доступным для обнаружения. На некоторых устройствах вам может потребоваться нажать кнопку, чтобы сделать Bluetooth доступным для обнаружения.
- Убедитесь, что два устройства расположены близко друг к другу.
- Убедитесь, что сопряженные устройства совместимы друг с другом, например, их версии совместимы.
- При появлении запроса введите один и тот же код сопряжения на обоих устройствах.

10.2 Bluetooth-интерфейс Raspberry Pi Pico W

Модуль Wi-Fi на Raspberry Pi Pico W поддерживает связь Bluetooth. К сожалению, на момент написания этой книги прошивка Bluetooth еще не была готова, и в результате поддержка Bluetooth была недоступна. Надеемся, что будущие выпуски будут поддерживать связь Bluetooth.

Поскольку поддержка Bluetooth (пока) недоступна на Raspberry Pi Pico W, вы можете рассмотреть возможность использования внешнего модуля Bluetooth, чтобы позволить Pico обмениваться данными с другими устройствами Bluetooth. Возможно, самым дешевым и простым вариантом является использование последовательного модуля Bluetooth, такого как HC-06. В следующих разделах вы разработаете проекты и узнаете, как подключить недорогой модуль Bluetooth типа HC-06 к вашему Raspberry Pi Pico W.

10.3 Проект 1: Управление тремя светодиодами со смартфона с помощью Bluetooth

Описание: В этом проекте вы будете отправлять команды по каналу Bluetooth со смартфона для управления тремя светодиодами, подключенными к Raspberry Pi Pico W (вы можете легко заменить светодиоды другими компонентами, такими как зуммеры, реле и т. д., чтобы можно было удаленно управлять устройствами). В этом проекте три светодиода названы R, G. и B. Допустимые команды:

R1	Включить R LED ON
R0	Выключить R LED OFF
G1	Turn G LED ON
G0	Turn G LED OFF
B1	Turn B LED ON
B0	Turn B LED OFF

Цель: цель этого проекта — использование недорогого последовательного модуля Bluetooth с Raspberry Pi Pico W для удаленного управления устройствами (в этом проекте 3 светодиода).

Модуль Bluetooth HC-06

HC-06 — недорогой популярный 4-контактный модуль с последовательным управлением и следующими контактами (см. рис. 10.1):

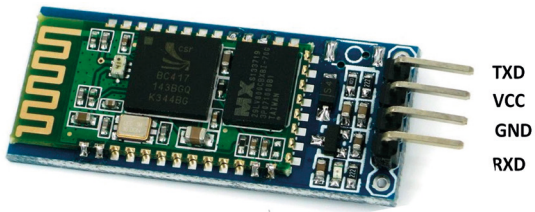


Рис.10.1: Модуль Bluetooth HC-06.

HC-06 имеет следующие основные характеристики:

- Питание от +3,3 В до +6 В
- Потребляемый ток 30 мА (согласованный ток 10 мА)
- Встроенная антенна
- Диапазон: 2,40 ГГц – 2,48 ГГц
- Уровень мощности: +6 дБм
- Связь по умолчанию: 9600 бод, 8 бит данных, без четности, 1 стоповый бит
- Покрытие сигнала около 9 м
- Функция безопасности: аутентификация и шифрование
- Режим модуляции: Гауссова частотная манипуляция

Блок-схема: На рис. 10.2 показана блок-схема проекта.

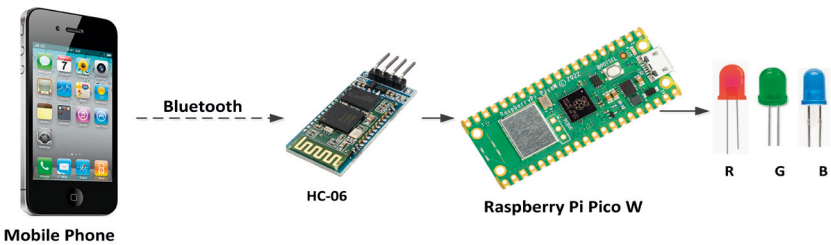


Рис.10.2: Блок-схема проекта.

Принципиальная схема: На рис.10.3 показана принципиальная схема проекта. Контакты RXD и TXD модуля Bluetooth подключены к контактам UART 0 GP0 (TX) и GP1 (RX) Raspberry Pi Pico соответственно. Светодиоды подключены к GP14 (светодиод R), GP15 (светодиод G) и GP16 (светодиод B) через токоограничивающие резисторы сопротивлением 470 Ом.

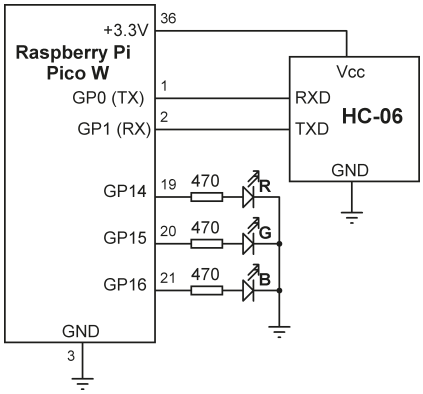


Рис.10.3: Принципиальная схема проекта.

Листинг программы: Листинг программы проекта показан на рис. 10.4 (программа: **BlueLED**). В начале программы для аппаратного интерфейса UART установлена скорость 9600 бод, которая является скоростью по умолчанию для HC-06, а светодиоды настроены как выходы и выключены. Остальная часть программы работает в бесконечном цикле. Внутри этого цикла данные (команды) принимаются от устройства Bluetooth с помощью вызова функции **readline**. Прочитанные данные сохраняются в списке с именем **buf**. Затем программа управляет светодиодами на основе полученных команд.

```
#------------------------------------------------------------
#              СВЯЗЬ ПО BLUETOOTH
#              ==================
#
# В этом проекте последовательный модуль Bluetooth типа HC-06 и
# и 3 светодиода подключены к Raspberry Pi Pico. Светодиоды
# управляются путем отправки команд с Bluetooth-совместимого
# смартфона. Подтверждение отправлено на смартфон
#
# Автор: Доган Ибрагим
# Файл   : BlueLED.py
# Дата: октябрь 2022 г.
#------------------------------------------------------------
from machine import Pin, UART
import utime

uart = UART(0, baudrate=9600,rx=Pin(1),tx=Pin(0))

R = Pin(14, Pin.OUT)                    # R LED
```

```
G = Pin(15, Pin.OUT)                    # G LED
B = Pin(16, Pin.OUT)                    # Y LED

#
# Turn OFF LEDs
#
R.value(0)
G.value(0)
B.value(0)

#
# Основной цикл программы. Получить команду и управлять светодиодом
#
while True:
     buf = uart.read()                          # Чтение данных
     if buf:
      dat = buf.decode('UTF-8')                         # Декодировать
      if dat[0] == 'R' and dat[1] == '1':       # R1?
         R.value(1)                              # LED ON
         uart.write("R LED is ON\n")             # Отправить подтверждение
      elif dat[0] == 'R' and dat[1] == '0':     # R0?
         R.value(0)                             # LED OFF
         uart.write("R LED is OFF\n")            # Отправить подтверждение

      elif dat[0] == 'G' and dat[1] == '1':     # G1?
         G.value(1)                             # LED OFF
         uart.write("G LED is ON\n")             # Отправить подтверждение
      elif dat[0] == 'G' and dat[1] == '0':     # G0?
         G.value(0)                             # LED OFF
         uart.write("G LED is OFF\n")            # Отправить подтверждение

      elif dat[0] == 'B' and dat[1] == '1':     # B1?
         B.value(1)                             # LED OFF
         uart.write("B LED is ON\n")             # Отправить подтверждение
      elif dat[0] == 'B' and dat[1] == '0':     # B0?
         B.value(0)                             # LED OFF
         uart.write("B LED is OFF\n")            # Отправить подтверждение
```

Рис.10.4: Программа: BlueLED.

Тестирование программы

Программу можно протестировать, используя мобильный телефон Android для отправки команд через интерфейс связи Bluetooth. В Play Store есть много бесплатных программ для связи Bluetooth. Автор выбрал **Bluetooth Controller** от mayyIT (it@memighty.com), как показано на рис. 10.5. Вы должны установить это приложение на свой мобильный телефон Android, чтобы вы могли отправлять команды на свое устройство, собранное на макетной плате.

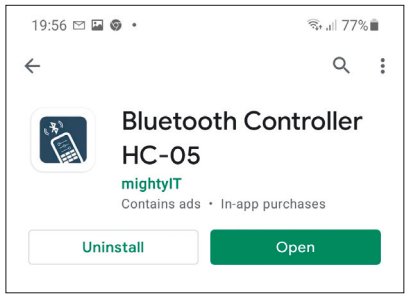


Рис.10.5: Приложения контроллера Bluetooth.

Шаги для тестирования приложения следующие:

- Создать проект.
- Загрузите программу на свой Raspberry Pi Pico W и запустите ее.
- Активируйте приложение **Bluetooth Controller** на своем мобильном телефоне.
- Приложения будут искать ближайшие Bluetooth-устройства. Нажмите на HC-06 при отображении на экране мобильного телефона (возможно, вам придется искать устройства).
- Теперь вас могут попросить ввести пароль для сопряжения мобильного телефона с Pico. Введите пароль по умолчанию 1234.
- Щелкните полукруг со стрелкой, расположенный в верхней правой части экрана, чтобы подключиться к **HC-06**.
- Вы должны увидеть зеленую точку в верхней правой части экрана, когда установлено соединение с HC-06. Кроме того, HC-06 с его адресом (например, HC-06 [98:D3:91:F9:6C:19]) должен отображаться в верхней левой части экрана.
- Чтобы включить светодиод R, введите команду **R1** и щелкните **Send ASCII**. Вы должны увидеть, как загорается светодиод R. Введите команду **R0**, чтобы выключить светодиод. На рис.10.6 показан пример экрана.

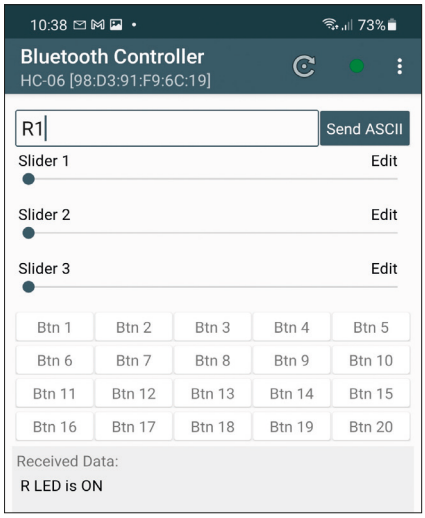
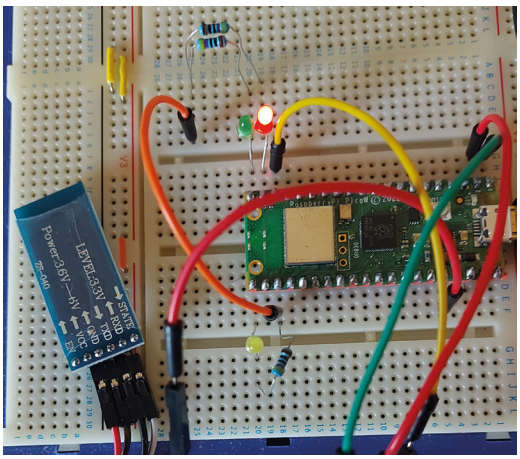


Рис.10.6: Пример команды для включения светодиода R.

На рис.10.7 показан проект, созданный на макетной плате.



На рис.10.7 показан проект, созданный на макетной плате.

10.4 Проект 2: Отправка внутренней температуры Raspberry Pi Pico W на смартфон

Описание: В этом проекте внутренняя температура Raspberry Pi Pico считывается каждые 10 секунд и отправляется на смартфон по каналу Bluetooth.

Цель: цель этого проекта — показать, как можно через равные промежутки времени отправлять данные с Raspberry Pi Pico на смартфон.

Блок-схема: На рис. 10.8 показана блок-схема проекта.

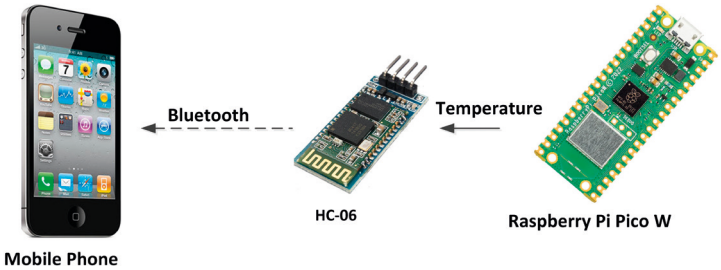


Рис.*10.8:* Блок-схема проекта.

Принципиальная схема: На рис. 10.9 показана принципиальная схема проекта. Контакты RXD и TXD модуля Bluetooth подключены к контактам UART 0 GP0 (TX) и GP1 (RX) Raspberry Pi Pico W соответственно.

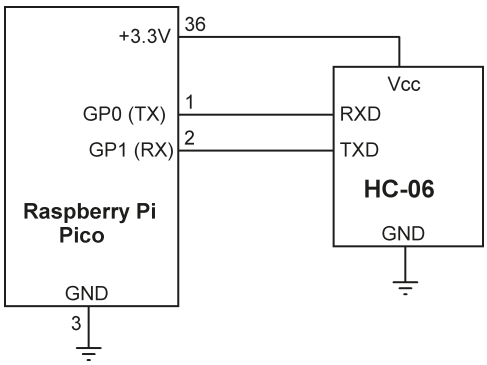


Рис.10.9: Принципиальная схема проекта.

Листинг программы: Листинг программы проекта показан на рис.10.10 (программа: **BlueTemp**). В начале программы для аппаратного интерфейса UART установлена скорость передачи 9600 бод по умолчанию для HC-06. Функция **GetTemperature** возвращает внутреннюю температуру Raspberry Pi Pico W. Внутри основного цикла программы температура считывается и отправляется на смартфон каждые 10 секунд. На рис. 10.11 показана температура, отображаемая на смартфоне с помощью приложений **Bluetooth Controller**, описанных в предыдущем разделе.

```
#-------------------------------------------------------------
#               ОТПРАВКА ТЕМПЕРАТУРЫ НА СМАРТФОН
#               ================================
#
# В этом проекте последовательный модуль Bluetooth типа HC-06
# подключен к Raspberry Pi Pico. Показания внутренней температуры
# отправляются на смартфон каждые 10 секунд.
#
```

```
# Автор: Доган Ибрагим
# Файл  : BlueTemp.py
# Дата: октябрь 2022 г.
#------------------------------------------------------------
from machine import Pin, UART, ADC
import utime

uart = UART(0, baudrate=9600,rx=Pin(1),tx=Pin(0))

Conv = 3.3 / 65535
AnalogIn = ADC(4)

def GetTemperature():
    V = AnalogIn.read_u16()
    V = V * Conv
    Temp = 27 - (V - 0.706) / 0.001721
    return Temp

#
# Отправить значение температуры на смартфон
#
while True:
     T = GetTemperature()
     Temp = "T=" + str(T)[:5] + "\r\n"
     uart.write(Temp)
     utime.sleep(10)
```

Рис.*10.10:* Программа: **BlueTemp**.

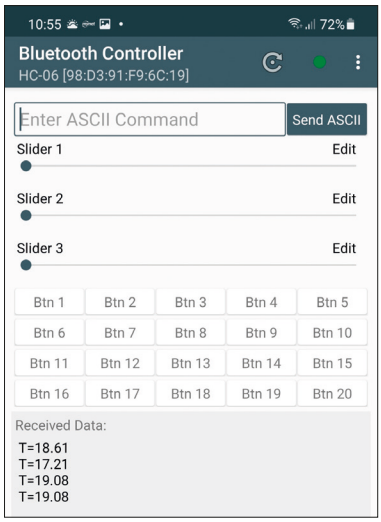


Рис.*10.11:* Температура, отображаемая на смартфоне.

Глава 11 • Использование Wi-Fi с Raspberry Pi Pico W

11.1 Обзор

Raspberry Pi Pico W добавляет возможности Wi-Fi к Raspberry Pi Pico, позволяя подключать устройство к сети Wi-Fi. Новый Pico W добавляет чипсет Infineon CYW43439 Wi-Fi 2,4 ГГц 802,11b/g/n со встроенной антенной. Этот чипсет поддерживает Wi-Fi, Bluetooth и Bluetooth с низким энергопотреблением (BLE). На момент написания этой книги прошивка Bluetooth еще не была доступна.

Возможно, первое, что вам может понадобиться, это попытаться подключиться к беспроводной сети. Как показано в этой главе, вам потребуется создать объект WLAN, использовать его метод подключения для подключения SSID и пароля. Вам должны дать IP-адрес. Затем вы можете разработать на основе Wi-Fiпрограммы, использующие язык программирования Micropython.

11.2 Подключение к беспроводной сети

Шаги для подключения к беспроводной сети описаны ниже с помощью функции connect():

```
import machine
import network
import utime

def connect():
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect("MySSID", "Mypassword")
while wlan.isconnected() == False:
               print("Waiting to be connected")
               utime.sleep(1)
```

После подключения вы можете отобразить IP-адрес вашего подключения с помощью следующего оператора:

```
print(wlan.ifconfig())
```

11.3 Проект 1: Сканирование локальной сети

Описание: Это простой пример Wi-Fi, который сканирует локальный сетевой роутер Wi-Fi и отображает пользователей этого роутера на экране Thonny.

Цель: Цель этого проекта — показать, как сканировать локальную сеть.
Листинг программы: Рис.11.1 показывает листинг программы (Программа: **scanwifi.py**). Обратите внимание, что нет необходимости подключать беспроводной роутер для сканирования сети. Функция **scan()** возвращает следующую информацию о найденных узлах сети:

(ssid, bssid, channel, RSSI, security, hidden)

В этой программе отображается только SSID.

bssid это аппаратный адрес точки доступа в двоичной форме, возвращаемый как объект bytes.Вы можете использовать **binascii.hexlify()**, чтобы преобразовать его в форму ASCII.

Существует пять значений безопасности:

- 0 – open
- 1 – WEP
- 2 – WPA-PSK
- 3 – WPA2-PSK
- 4 – WPA/WPA2-PSK

и два для скрытых:

- 0 – видимый
- 1 – скрытный

Некоторые другие доступные сетевые функции:

wlan.status() Возвращает текущий статус беспроводного соединения.
wlan,config(): получить/установить общие параметры сети
wlan.disconnect(): отключиться от сети

Дополнительную информацию о сетевых функциях можно найти по следующей ссылке:

https://docs.micropython.org/en/latest/library/network.WLAN.html

```
#------------------------------------------------------------
#         СКАНИРОВАНИЕ ЛОКАЛЬНОЙ БЕСПРОВОДНОЙ СЕТИ
#        ==========================================
#
# Эта программа сканирует локальную беспроводную сеть
# и отображает пользователей в этой сети
#
# Автор: Доган Ибрагим
# Файл  : scanwifi.py
# Дата: октябрь 2022 г.
#------------------------------------------------------------
import machine
import network
import utime

def connect():
    global wlan
    wlan = network.WLAN(network.STA_IF)
```

```
    wlan.active(True)
#    wlan.connect("BTHomeSpot-XNH", "49345axwub")
#    while wlan.isconnected() == False:
#                        print("Waiting to be connected")
#                        utime.sleep(1)
#    print("Connected")

connect()
wifis = wlan.scan()

i=0
for w in wifis:
      i+=1
      print(i,w[0].decode())
```

Рис.*11.1:* Программа:**scanwifi.py**.

На рис.11.2 показаны данные, отображаемые на экране Тонни.

```
1 The Hotal
2 BTHub5-6SPN
3 DIRECT-1F-HP Laser 150nw
4 BTHomeSpot-XNH
5
6 BTWi-fi
7 BTB-KNCGPZ
8 Family room TV.b,
9 BTHomeSpot-XNH
10 BTWi-fi
11
12 World Machine
```

Рис.*11.2:* Отображаемые данные.

Вы можете закрыть соединение с помощью функции **wlan.disconnect()** после завершения работы.

11.4 Использование библиотеки сокетов

Библиотека Socket в основном используется в сетевых коммуникациях. UDP и TCP/IP являются два наиболее часто используемых протокола для отправки и получения данных по сети. TCP/IP есть надежный протокол, который включает рукопожатие и, таким образом, гарантирует доставку пакетов до требуемого пункта назначения. UDP, с другой стороны, не так надежен, но является быстрым протоколом.

В таблице 11.1 показано сравнение типов связи UDP и TCP/I

TCP/IP	UDP
Протокол, ориентированный на соединение	Протокол без установления соединения
Медленный	Быстрый
Высоконадежная передача данных	Не так надежно
Пакеты расположены по порядку	Нет заказа пакетов
20-байтовый размер заголовка	8-байтовый размер заголовка
Проверка ошибок и повторная передача	Нет проверки ошибок
Подтверждение получения данных	Нет подтверждения
Используется в HTTP, HTTPS, FTP и т. д.	Используется в DNS, DHCP, TFTP и т. д.

Таблица 11.1: Сравнение UDP и TCP/IP.

Программы, основанные на протоколах UDP и TCP/IP, основаны на клиент-серверном подходе, когда один узел отправляет данные, а другой получает их, и наоборот. Данные передаются через порты, где сервер и клиенты должны использовать одни и те же номера портов.

В следующих разделах вы увидите, как программы на основе протоколов UDP и TCP/IP могут быть написаны с использованием MicroPython для процессора ESP32.

11.4.1 Программы UDP

UDP — это протокол без установления соединения, поэтому нет необходимости устанавливать соединение с узлом назначения, прежде чем на него будет отправлен пакет. Связь между сервером и клиентом в основном выглядит следующим образом:

Сервер
Определите IP-адрес узла назначения и номер порта.
Создайте пакет данных.
Создайте сокет.
Привязать сокет к локальному порту
Получение данных от клиента.
Отправка данных клиенту.
Закройте соединение.

Клиент
Определите IP-адрес узла назначения и номер порта.
Создайте сокет.
Отправить данные на сервер.
Получить данные с сервера.
Закройте соединение.

Обратите внимание, что и сервер, и клиент могут отправлять и получать пакеты данных друг от друга.

11.5 Проект 2: Управление светодиодом со смартфона с помощью Wi-Fi — UDP-связь

Описание: В этом проекте вы будете отправлять команды по каналу Wi-Fi со смартфона для управления светодиодом (светодиод можно заменить реле, например, для управления оборудованием), подключенным к Raspberry Pi Pico W. Допустимые команды (команда должна заканчиваться новой строкой):

LON	Включение LED
LOFF	Выключение LED

Цель: Цель этого проекта — показать, как использовать подключение Wi-Fi на Raspberry Pi Pico W.

Блок-схема: На рис. 11.3 показана блок-схема проекта.

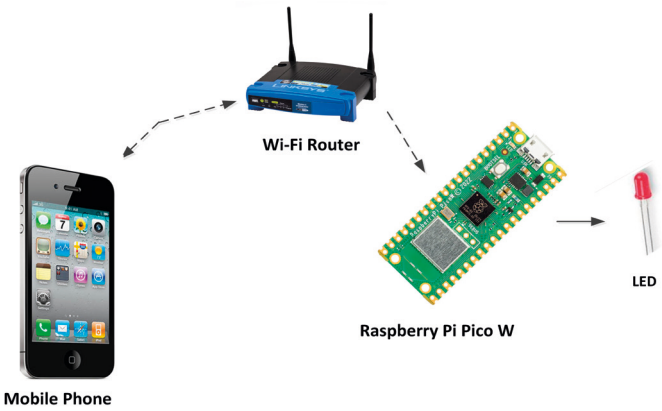


Рис.11.3: Блок-схема проекта.

LED подключен к GP15 (вывод 20) Raspberry Pi Pico W через токоограничивающий резистор 470 Ом.

Листинг программы: перед написанием программы вы должны найти IP-адрес, присвоенный вашему Pico. Это можно отобразить на экране Тонни, включив следующее утверждение:**print(wlan.ifconfig())** после подключения к Wi-Fi. В этом проекте IP-адрес Pico был: 192.168.1.154.

На рис. 11.4 показан листинг программы (программа: **Picowifi**). Функция **Connect()** вызывается для подключения к локальному Wi-Fi роутеру. Оставшаяся часть программы выполняется в бесконечном цикле, сформированном с помощью оператора **while**. Внутри этого цикла данные поступают со смартфона по каналу UD с портом 5000, и светодиод управляется соответствующим образом. Команды **LON** и **LOFF** включают и выключают светодиод соответственно.

```
#------------------------------------------------------------
#              УДАЛЕННОЕ УПРАВЛЕНИЕ LED ПО WI-FI
#              ==================================
#
# В этой программе светодиод (или зуммер, реле и т. д.), подключенный к Pico,
# управляется со смартфона по wi-fi каналу
#
# Автор: Доган Ибрагим
# Файл   : picowifi.py
# Дата: октябрь 2022 г.
#------------------------------------------------------------
from machine import Pin
import network
import socket
import utime

LED = Pin(15, Pin.OUT)               # LED на GP15
LED.value(0)                         # LED  ВЫКЛ.

#
# Эта функция пытается подключиться к Wi-Fi
#
def connect():
    global wlan
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect("BTHomeSpot-XNH", "4933axb")
    while wlan.isconnected() == False:
              print("Waiting to be connected")
              utime.sleep(1)
    print("Connected")

connect()
port = 5000
UDP = ("", port)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(UDP)

#
# Основной цикл программы
#
while True:
     buf, addr = sock.recvfrom(1024)
     if buf:
        buf = buf.decode('utf-8')
        if buf[0]=="L" and buf[1]=="O" and buf[2]=='N':
```

```
        LED.value(1)
    if buf[0]=="L" and buf[1]=='O' and buf[2]=='F' and buf[3]=='F':
        LED.value(0)
```

Рис.*11.4:* Программа:**Picowifi**.

Тестирование программы

Программу можно легко протестировать с помощью программы **PacketSender** (см. рис. 11.5) на ПК или с помощью смартфона после установки приложения UDP.

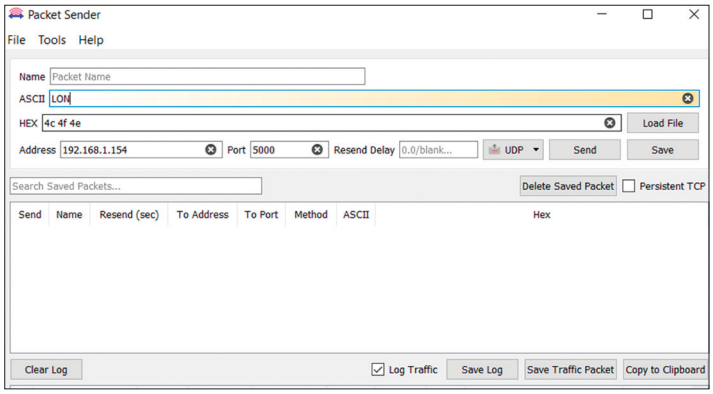


Рис.11.5: Использование PacketSender для тестирования программы.

Вы должны установить приложение **UDP Server** на свой мобильный телефон Android, прежде чем начинать тест со смартфоном. В **Play Store** есть много бесплатных приложений UDP. Тот, который установлен и используется в этом проекте, называется **UDP Terminal**, как показано на рис. 11.6.

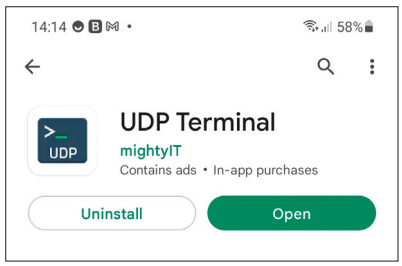


Рис.11.6: UDP-терминал

Шаги для тестирования программы следующие:

- Соберите схему
- Загрузите программу на Raspberry Pi Pico W и запустите ее.
- Запустите приложение **UDP Terminal** на своем смартфоне.
- Установите IP-адрес Pico (в данном примере 192.168.1.154), как показано на рис. 11.7.

• Щелкните **Start Terminal.**
• Введите команду **LON** в нижней части экрана, чтобы включить светодиод, и щелкните **Send ASCII**. Светодиод должен загореться (рис.11.8).

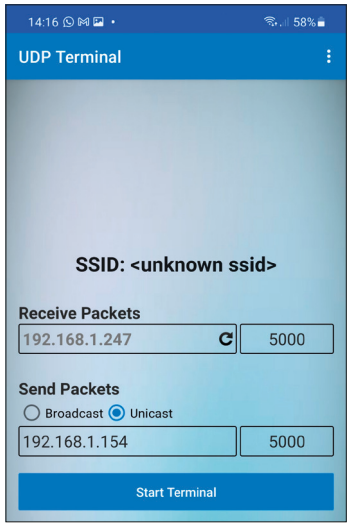


Рис.*11.7:* Запустите *UDP*-терминал.

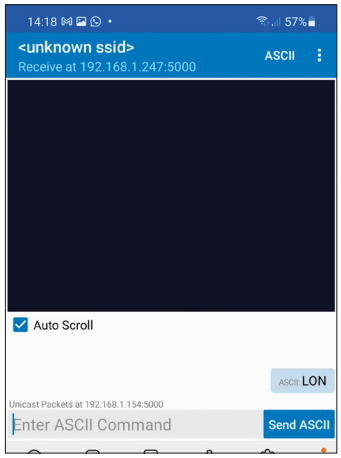


Рис.11.8: Щелкните Send Ascii - Отправить Ascii.

11.6 Проект 3: Отображение внутренней температуры на смартфоне с помощью Wi-Fi

Описание: В этом проекте вы будете считывать внутреннюю температуру Raspberry Pi Pico W, а затем отправлять эти данные на смартфон по каналу Wi-Fi.

Запрос данных осуществляется смартфоном (например, **T?**), когда ему требуются данные от Pico. Этот проект использует двустороннюю связь UDP для получения и отправки данных.

Цель: Цель этого проекта — показать, как можно установить двустороннюю связь со смартфоном по каналу Wi-Fi.

Блок-схема: На рис. 11.9 показана блок-схема проекта.

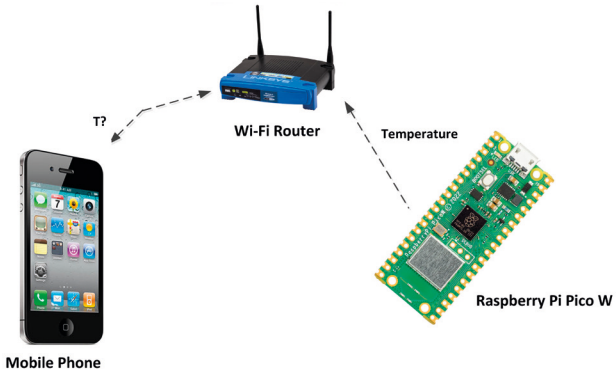


Рис.11.9: Блок-схема проекта.

Листинг программы: Листинг программы показан на рис. 11.10 (program: temptowifi). Функция Connect() вызывается для подключения к локальному роутеру Wi-Fi, как и в предыдущей программе. Внутри основного цикла программы программа ожидает получения команды T? и когда эта команда получена, вызывается функция GetTemperature, и внутренняя температура Pico считывается и сохраняется в переменной T в основной программе. Строковая переменная Tstr хранит строку T=, за которой следует значение температуры в виде строки. Длина этой строки сохраняется в переменной Tlen и затем отправляется на смартфон с помощью сетевой функции sendto().

```
#------------------------------------------------------------
#              ОТПРАВИТЬ ТЕМПЕРАТУРУ НА СМАРТФОН
#              ==================================
#
# В этом проекте внутренняя температура Raspberry Pi Pico W
# отправляется на смартфон
#
# Автор: Доган Ибрагим
# Файл   : temptowifi.py
# Дата: октябрь 2022 г.
#------------------------------------------------------------
from machine import ADC
import network
import socket
import utime

AnalogIn = ADC(4)                       # Канал 4 АЦП
Conv = 3300 / 65535                     # Коэффициент преобразования
```

```
Conv = 3.3 / 65535                    # Фактор общения

#
# Эта функция возвращает внутреннюю температуру
#
def GetTemperature():
    V = AnalogIn.read_u16()
    V = V * Conv
    Temp = 27 - (V - 0.706) / 0.001721
    return Temp

#
#  Эта функция пытается подключиться к Wi-Fi
#
def connect():
    global wlan
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect("BTHomeSpot-XNH", "49345xsdaez")
    while wlan.isconnected() == False:
              print("Waiting to be connected")
              utime.sleep(1)
    print("Connected")

connect()
port = 5000
UDP = ("", port)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(UDP)

#
# Основной цикл программы
#
while True:
     buf, addr = sock.recvfrom(1024)
     if buf:
        buf = buf.decode('utf-8')
        if buf[0]=="T" and buf[1]== '?':
           T = GetTemperature()
           Tstr = "T=" + str(T)[:5]
           Tlen = len(Tstr)
           sock.sendto(Tstr, addr)
```

Рис.*11.10:* Программа:**temptowifi**.

Тестирование программы
Программу можно протестировать либо на ПК с помощью бесплатно доступной программы **PacketSender**, либо на смартфоне с установленным приложением UDP. **PacketSender** позволяет пользователю отправлять и получать UDP- и TCP-пакеты с ПК. Эта программа может быть очень полезна при тестировании UDP- и TCP-программ.

Кроме того, вы можете использовать приложение **UDP Terminal**, как и в предыдущем проекте, для тестирования программы. Пример запуска программы на приложениях показан на рис. 11.11, где команда **T?** отправляется на Raspberry Pi Pico W, а внутренняя температура извлекается и отображается на экране Android. Обратите внимание, что IP-адрес Pico был: 192.168.1.154.

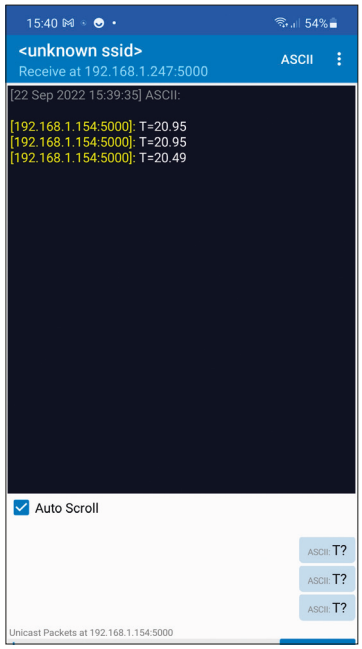


Рис.11.11: Пример запуска программы.

11.7 Проект 4: Дистанционное управление из интернет-браузера (используя смартфон или ПК) - веб-сервер

Описание: В этом проекте два светодиода подключены к Raspberry Pi Pico W и управляются удаленно с устройства, на котором может работать интернет-браузер (например, Firefox и т. д.). Идея заключается в удаленном управлении светодиодами через веб-ссылку. В практических приложениях эти светодиоды можно заменить реле, и проект можно использовать для управления любым устройством, подключенным к макетной плате.

Следующие допустимые команды (любые другие команды игнорируются программой):

0=ON	Включить светодиод 0
0=OFF	Выключить светодиод 0
1=ON	Включить светодиод 1
1=OFF	Выключить светодиод 1

Блок-схема: На рис. 11.12 показана блок-схема проекта.

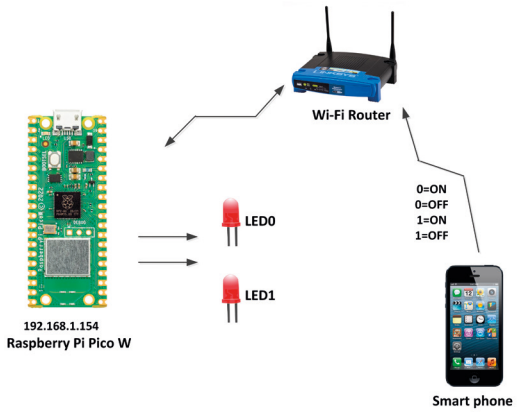


Рис.11.12: Блок-схема проекта.

Принципиальная схема: Принципиальная схема проекта показана на рисунке 11.13. LED0 и LED1 подключены к выводу 19 порта GP14 и GP15 (вывод 20) через токоограничительные резисторы 470 Ом.

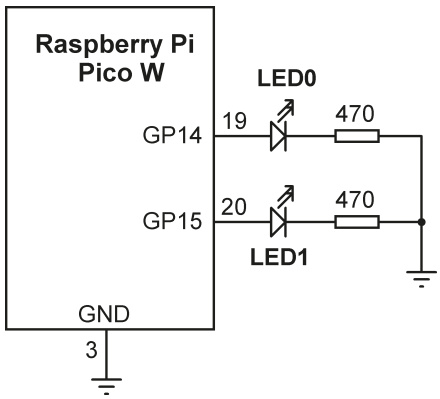


Рис.11.13: Принципиальная схема проекта.

Листинг программы: Рис.11.14 показывает листинг программы (Программа: WebServer.py). В начале программы сетевые библиотеки и библиотеки сокетов импортируются в программу. Затем GP14 и GP15 настраиваются как выходы. Затем программа определяет код HTML, который должен быть отправлен веб-клиенту, когда от него приходит запрос. Этот код хранится в переменной **html**.

Программа веб-сервера отправляет html-файл (операторы **resp=html** и **conn.send (resp)**) клиенту и ожидает команды от клиента. При нажатии кнопки на клиенте на веб-сервер отправляется команда в дополнение к некоторым другим данным. Здесь вас интересует только фактическая отправленная команда. Следующие команды отправляются на веб-сервер для каждой нажатой кнопки:

Кнопка нажата	**Команда отправлена на веб-сервер**
LED0 ON	/?LED0=ON
LED0 OFF	/?LED0=OFF
LED2 ON	/?LED2=ON
LED2 OFF	/?LED2=OFF

Программа веб-сервера на Pico устанавливает TCP-соединение и считывает команду с веб-клиента (**request = conn.recv(1024)**). Затем программа ищет полученную команду, чтобы увидеть, присутствует ли какая-либо из вышеуказанных команд в полученных данных. Помните, что функция **find** ищет в строке подстроку и возвращает начальную позицию подстроки в строке. Если подстрока не найдена, возвращается «-1». Например, рассмотрим следующее утверждение:

```
LEDON0 = request.find("/?LED0=ON")
```

Здесь подстрока **/?LED0=ON** ищется в запросе строки, и если она найдена, дается начальная позиция подстроки в строке. "-1" возвращается, если подстрока **/?LED0=ON** не существует в строковом запросе. Программа ищет совпадения для всех четырех команд. Если команда найдена, то соответствующий светодиод включается или выключается соответственно:

```
LEDON0 = request.find("/?LED0=ON")
       LEDOFF0 = request.find("/?LED0=OFF")
       LEDON2 = request.find("/?LED2=ON")
       LEDOFF2 = request.find("/?LED2=OFF")

       if LEDON0 != -1:                          # Если найдена команда ON0
                       LED0.value(1)             # Включить LED0
       if LEDOFF0 != -1:                         # Если найдена команда OFF0
                       LED0.value(0)             # Выключить LED0
        if LEDON2 != -1:                         # Если найдена команда ON2
                       LED2.value(1)             # Включить LED2
       if LEDOFF2 != -1:                         # Если найдена команда OFF2
                       LED2.value(0)             # Выключить LED2
```

В этом проекте IP-адрес веб-сервера был 192.168.1.154. Процедура проверки системы следующая:

- Запустите программу MicroPython, показанную на рис. 11.14, на Pico.
- Откройте веб-браузер на своем ПК (или любом другом устройстве с интернет-браузером) и введите адрес веб-сайта 192.168.1.154.

- HTML-форма управления будет отображаться на экране вашего ПК, как показано на рис. 11.15. Нажимайте кнопки для включения/выключения нужного светодиода.

```
#--------------------------------------------------------------------------
#                      ВЕБ-СЕРВЕР ДЛЯ УПРАВЛЕНИЯ 2 LED
#                      ================================
#
# Это простая программа веб-сервера. 2 светодиода подключены к GP14 и GP15
#  Raspberry Pi Pico W. Проект управляет этими светодиодами
# (включает или выключает) из приложения веб-сервера. Например,
#  LED можно управлять с любого устройства, которое находится в сети, например
# с ПК, планшета, мобильного телефона и т. д. При активации на устройстве появится
#  форма с кнопками, и нажатие на эти кнопки будет управлять светодиодами
# Светодиоды называются LED0 и LED1
#
# Автор: Доган Ибрагим
# Дата: октябрь 2022 г.
# Файл  : WebServer.py
#--------------------------------------------------------------------------
from machine import Pin
import network
import socket
import utime

#
# Эта функция пытается подключиться к Wi-Fi
#
def connect():
    global wlan
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect("BTHomeSpot-XNH", "49345azse5")
    while wlan.isconnected() == False:
              print("Waiting to be connected")
              utime.sleep(1)
    print("Connected")

#
# Настроить пины как выходы
#
LED0 = Pin(14, Pin.OUT)
LED1 = Pin(15, Pin.OUT)
#
#  Выключите светодиоды, чтобы начать с
#
LED0.value(0)
```

```
LED1.value(0)

#
# HTML-код
#
html = """<!DOCTYPE html>
<html>
<body>
<h1>Raspberry Pi Pico W LED ON/OFF</h1>
<h2>MicroPython Web Server Example with 2 LEDs</h2>
<form>

<button name="LED0" button style="color:green" value="ON" type="submit">LED0 ON</
button>
<button name="LED0" button style="color=red" value="OFF" type="submit">LED0 OFF</
button><br><br>

<button name="LED1" button style="color:green" value="ON" type="submit">LED1 ON</
button>
<button name="LED1" button style="color:red" value="OFF" type="submit">LED1 OFF</
button>
</form>
</body>
</html>
"""

#
# Подключиться к локальному Wi-Fi
#
connect()
addr=socket.getaddrinfo("0.0.0.0",80)[0][-1]
s = socket.socket()
s.bind(addr)
s.listen(1)

while True:
    conn, address = s.accept()
    request = conn.recv(1024)
    request = str(request)
    LEDON0 = request.find("/?LED0=ON")
    LEDOFF0 = request.find("/?LED0=OFF")
    LEDON1 = request.find("/?LED1=ON")
    LEDOFF1 = request.find("/?LED1=OFF")
    if LEDON0 != -1:
        LED0.value(1)
    if LEDOFF0 != -1:
```

```
        LED0.value(0)
    if LEDON1 != -1:
        LED1.value(1)
    if LEDOFF1 != -1:
        LED1.value(0)
    resp = html
    conn.send(resp)
    conn.close()
```

Рис.*11.14:* Программа:**WebServer.py**.

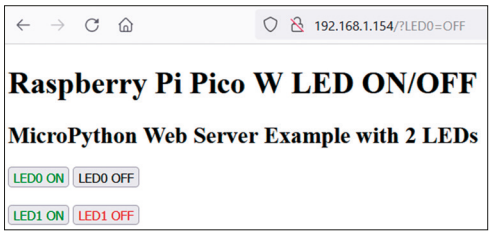


Рис.*11.15:* Форма, отображаемая на ПК.

11.8 Проект 5: Хранение данных о температуре окружающей среды и атмосферном давлении в облаке

Описание: В этом проекте температура окружающей среды и атмосферное давление считываются и сохраняются в облаке. В этом проекте используется сенсорный модуль типа BMP280.

BMP280

BMP280 — это микросхема датчика абсолютного барометрического давления и температуры, разработанная для мобильных приложений. Её небольшие размеры и низкое энергопотребление позволяют использовать его в устройствах с батарейным питанием, таких как мобильные телефоны, модули GPS или часы. BMP280 основан на проверенной технологии пьезорезистивного датчика давления Bosch, отличающейся высокой точностью и линейностью, а также долговременной стабильностью и высокой устойчивостью к ЭМС. Устройство оптимизировано с точки зрения энергопотребления, разрешения и производительности фильтра. Основные характеристики BMP280:

- Диапазон давления: от 300 до 1100 гПа
- Относительная точность: ±0,12 гПа
- Абсолютная точность: ±1 гПа
- Цифровой интерфейс: I2C, SPI
- Диапазон температур: от -40 ºC до +85 ºC
- Потребляемый ток: 2,7 мкА

Блок-схема: Блок-схема проекта показана на рисунке 11.16.

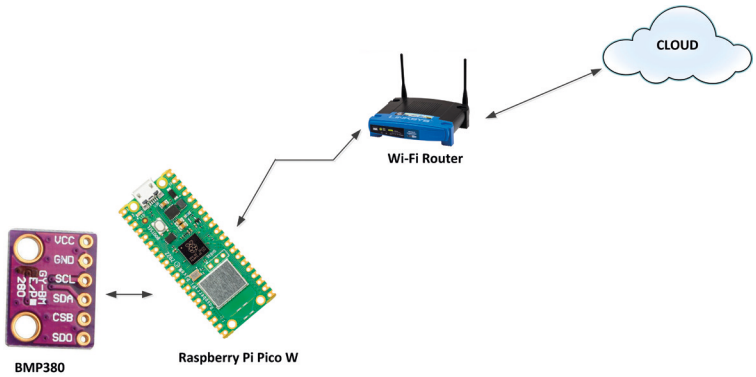


Рис.*11.16:* Блок-схема проекта.

Принципиальная схема: На рис. 11.17 показана принципиальная схема. Выводы SCL и SDA BMP280 подключены к SCL (вывод 2) и SDA (вывод 1) Pico. Датчик питается от +3,3 В.

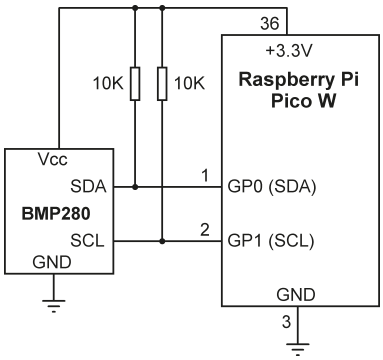


Рис.*11.17:* Принципиальная схема проекта.

Облако

Существует несколько облачных сервисов, которые можно использовать для хранения данных (например, SparkFun, ThingSpeak, Cloudino, Bluemix и т. д.). В этом проекте используется Thingspeak. Это бесплатный облачный сервис, в котором данные датчиков могут храниться и извлекаться с помощью простых HTTP-запросов. Прежде чем использовать Thingspeak, вы должны создать учетную запись на их веб-сайте, а затем войти в эту учетную запись.
Перейдите на веб-сайт ThingSpeak:

https://thingspeak.com/

Перейдите на веб-сайт ThingSpeak:https://thingspeak.com/Нажмите «Started For Free - Начать бесплатно» и создайте учетную запись, если у вас ее еще нет. Затем выдолжны создать новый канал, нажав «**New Channel**». Заполните форму, как показано на рис.11.18. Дайте приложению имя Raspberry Pi Pico W, дайте описание и создайте два поля под названием **Atmospheric Pressure** и Temperature. При желаниивы можете заполнить и другие пункты.

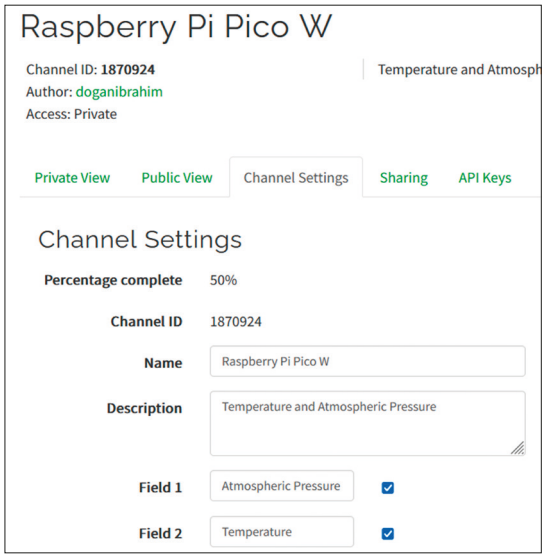


Рис.11.18: Создание нового канала (показана только часть формы).

Щелкните Save Channel в нижней части формы. Теперь ваш канал готов к использованию с вашими данными. Вы увидите вкладки со следующими именами. Вы можете нажать на эти вкладки и просмотреть содержимое, чтобы при необходимости внести исправления:

- **Private View**: на этой вкладке отображается личная информация о вашем канале, которую можете видеть только вы.
- **Public View**: если ваш канал является общедоступным, используйте эту вкладку для отображения выбранных полей и визуализаций канала.
- **Channel Settings**: на этой вкладке отображаются все параметры канала, которые вы установили при создании. На этой вкладке вы можете редактировать, очищать или удалять канал.
- **API Keys**: отображаются ключи API вашего канала. Используйте клавиши для чтения и записи на ваш канал.
- **Data Import/Export**: позволяет импортировать и экспортировать данные канала.

Вы должны щелкнуть вкладку «**API Keys** - Ключи API» и сохранить ключи **Write API** и **Read API**, а также ID каналов в надежном месте, так как они понадобятся вам в вашей программе. Ключи API и ID канала в этом проекте:

Также обратите внимание на номер вашего канала **Channel Number**: 1870924.

Кроме того, выберите «**Public View** - Общий вид» и перейдите к «**Sharing** - Общий доступ». Вы можете выбрать опцию «**Share channel view with everyone** - Поделиться просмотром канала со всеми», чтобы каждый мог получить удаленный доступ к вашим данным.

Листинг программы: на рис.11.19 показан листинг программы (Программа: **Cloud.py**). Прежде чем использовать программу, вы должны загрузить библиотеку BMP280 на свой Pico. Эта библиотека находится на веб-сайте книги под именем **bmp280.py**. Вы должны скопировать этот файл на свой Raspberry Pi Pico W с точным именем **bmp280.py**.
В начале в программу импортируются сетевые библиотеки, библиотека времени, библиотека bmp280 и библиотека I2C. Определены **Write Key** - ключ записи Thingspeak и **Host Address**. Функция **connect()** подключается к локальной сети Wi-Fi. Основной цикл программы начинается с оператора **while**. Внутри этого цикла извлекается IP-адрес веб-сайта **Thingspeak** и устанавливается соединение с этим сайтом через порт 80. Затем из модуля BMP280 получаются показания атмосферного давления и температуры, которые включаются в оператор **path** . Оператор **sock.send** отправляет HTTP-запрос GET на сайт **ThingSpeak** и загружает значения давления и температуры. Этот процесс повторяется каждые 30 секунд.

На рис. 11.20 показаны данные давления и температуры, построенные **ThingSpeak**. Параметры диаграммы можно щелкнуть, чтобы изменить различные параметры диаграмм. Например, на рис. 11.21 температура показана в виде столбца.
На рис. 11.22 давление показано в виде ступенчатого графика. На рис. 11.23 к графику давления добавлены заголовок и метка по оси X. На рис. 11.24 показано атмосферное давление, отображаемое в формате часов (нажмите «**Add Widgets** - Добавить виджеты» для этого типа отображения).

Поскольку канал был сохранен как общедоступный, вы можете просмотреть график из веб-браузера, введя ID канала. В этом проекте ссылка для просмотра графиков данных из веб-браузера:

https://api.thingspeak.com/channels/1870924

На рис.11.25 показаны графики, доступ к которым осуществляется из веб-браузера. Вы также можете экспортировать некоторые или все поля в формате CSV, щелкнув **Export recent data** - Экспорт последних данных, чтобы их можно было проанализировать с помощью внешних статистических пакетов, таких как Excel (рис. 11.26).

```
#-------------------------------------------------------------------------------
#         АТМОСФЕРНОЕ ДАВЛЕНИЕ И ТЕМПЕРАТУРА НА ОБЛАКЕ
#         ============================================
#
# Датчик температуры и давления окружающей среды BMP280 подключен к Pico.
# Проект считывает температуру и атмосферное давление и отправляет в облако,
# где к нему можно получить доступ из любого места. Кроме того, в облаке
# можно отобразить изменение температуры и давления.
#
#
#  Программа использует облачный сервис Thingspeak
#
# Автор: Доган Ибрагим
# Файл   : Cloud.py
# Дата: январь 2021 г.
#-------------------------------------------------------------------------------
import network
import socket
import utime
from machine import Pin,I2C
from bmp280 import *

bus = I2C(0,scl=Pin(1),sda=Pin(0),freq=200000)
bmp = BMP280(bus)

bmp.use_case(BMP280_CASE_INDOOR)
APIKEY = "BUHXZZC718WT0X0P"                    # Ключ API Thingspeak
host = "api.thingspeak.com"                    # Хост Thigspeak

#
# This function attempts to connect to Wi-Fi
#
def connect():
    global wlan
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect("BTHomeSpot-XNH", "49345we34g")
    while wlan.isconnected() == False:
              print("Waiting to be connected")
              utime.sleep(1)

#
#  Отправить данные в Thingspeak. Эта функция отправляет температуру и
#  данные о влажности в облако каждые 30 секунд
#
connect()
```

```
while True:
    connect()
    sock = socket.socket()
    addr = socket.getaddrinfo("api.thingspeak.com",80)[0][-1]
    sock.connect(addr)
    pressure=bmp.pressure
    p=pressure/133.3224                 # Давление в мм вод. ст.
    t=bmp.temperature                   # Температура в градусах Цельсия
    path = "api_key="+APIKEY+"&field1="+str(p)+"&field2="+str(t)
    sock.send(bytes("GET /update?%s HTTP/1.0\r\nHost: %s\r\n\r\n"
%(path,host),"utf8"))
    utime.sleep(5)
    sock.close()
    wlan.disconnect()
    utime.sleep(25)
```

Рис.*11.19:* Программа **Cloud.py**.

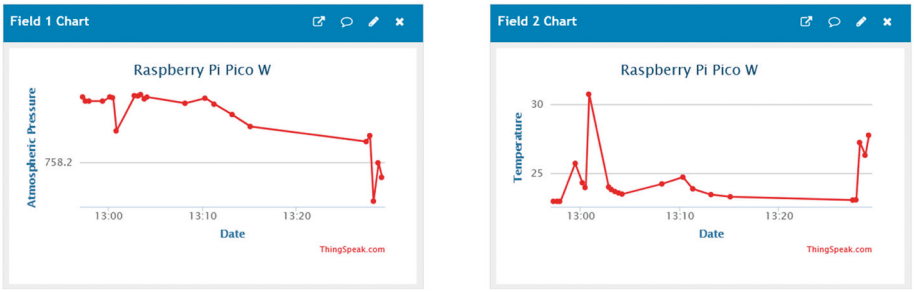


Рис.*11.20:* График давления и температуры.

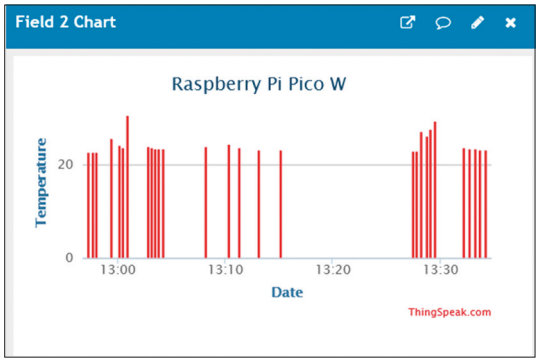


Рис.*11.21:* Отображение температуры в виде столбцов.

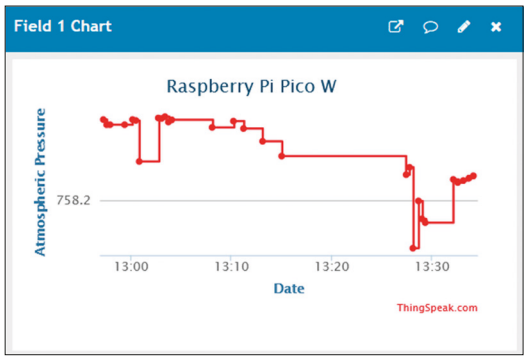


Рис.11.22: Отображение давления в виде шагов.

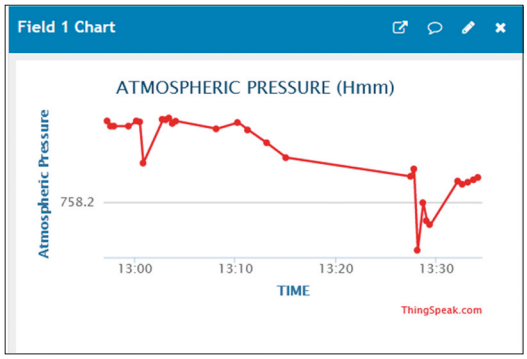


Рис.11.23: Добавление заголовка и метки оси X.

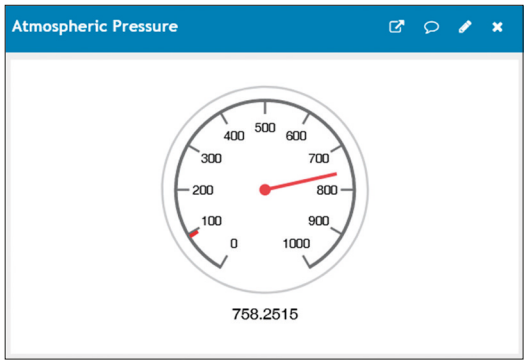


Рис.11.24: Отображение давления в формате часов.

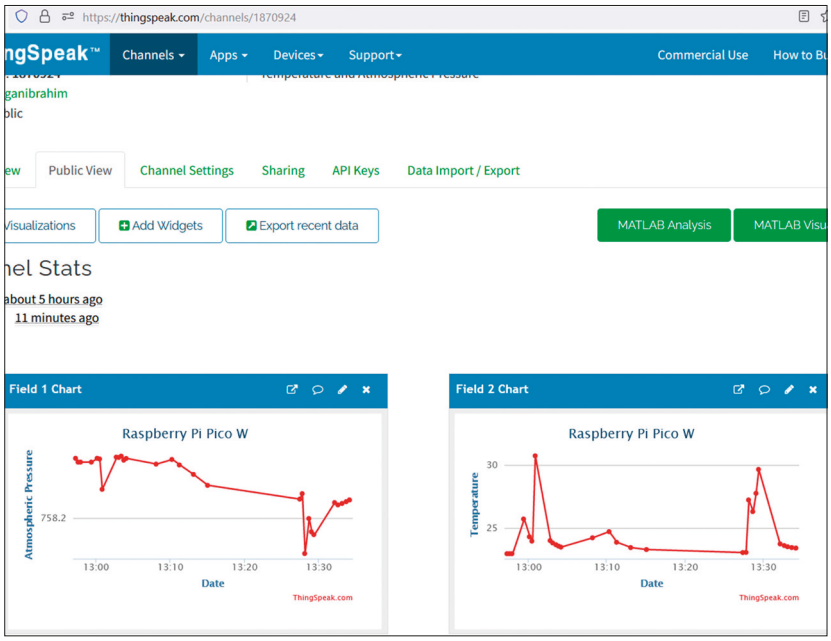


Рис.11.25: Доступ к графику с веб-сервера.

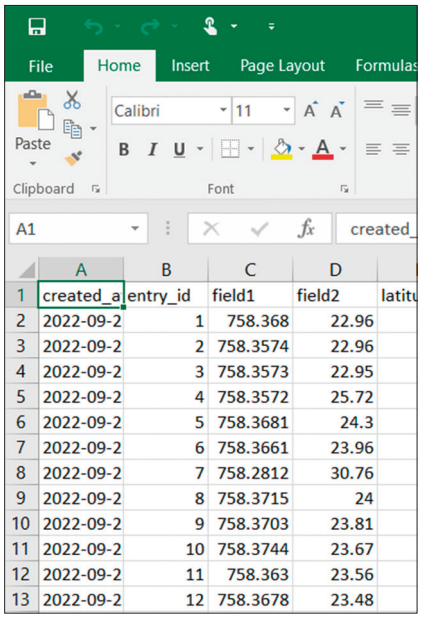


Рис.11.26: Экспорт данных в формате Excel.

Глава 12 • Проекты RFID

12.1 Обзор

RFID — это устройство радиочастотной идентификации, используемое в системах безопасности и отслеживания. Система RFID включает карту считывателя и метку, и обе они входят в комплект.
RFID использует электромагнитные поля для передачи данных на короткие расстояния. Системы RFID в основном используются в приложениях безопасности, например, их можно использовать для открытия двери, где человеку, имеющему правильный тег, разрешено открывать дверь.

Одним из популярных считывателей RFID является модуль RC522 (рис. 12.1) со следующими основными характеристиками:

- Рабочая частота: 13,56 МГц
- Рабочее напряжение: от +2,5 В до +3,3 В.
- Рабочий ток: от 13 до 26 мА
- Диапазон считывания: 5 см
- Работа как с шиной SPI, так и с шиной I^2C

Рис.*12.1:* Считыватель *RFID* и метка.

Считыватель RFID обычно поставляется с штырьками, которые должны быть припаяны к гнездам считывателя, прежде чем его можно будет использовать. Модуль RFID RC522 поставляется с биркой RFID-карты и брелком, что позволяет разработчикам связывать его с любыми другими микроконтроллерами на базе SPI, I2C и UART.

Модуль RC522 RFID работает на частоте 13,56 МГц и связывается с пассивными картами RFID на коротких расстояниях с помощью радиочастотных сигналов (RF). Поставляемая RFID-карта имеет объем памяти 1 КБ (1024 байта), разделенный на 16 секторов, где каждый сектор затем делится на 4 блока. Каждый блок состоит из 16 байт. Следовательно, 4 блока × 16 байт × 16 секторов = 1024 байта.

12.2 Контакты считывателя RFID RC522

Модуль RFID RC522 имеет распиновку, как показано на рис. 2.2 :

MISO: выход SPI от RC522, когда интерфейс SPI включен. Тактовый сигнал при включенном I2C. Последовательный вывод данных, когда UART включен.

MOSI: вход SPI на RC522

SCK: тактовый вход SPI

SDA: чип с поддержкой SPI, когда SPI включен. Последовательные данные при включении I2C. Последовательный ввод данных при включенном UART.

RST: Сброс ввода. При НИЗКОМ уровне модуль переходит в режим пониженного энергопотребления, в котором генератор выключается, а входные контакты разъединяются.

VCC: +3.3 V

GND: земля источника питания

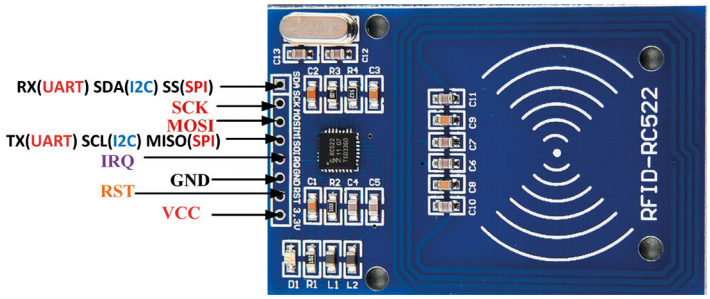


Рис.12.2: Контакты RC522.

12.3 Подключение модуля считывателя RFID RC522 к Pico

Перед подключением модуля RC522 вы должны знать выводы шины SPI Raspberry Pi Pico W, которые совпадают со стандартными выводами Pico, как показано на рисунке 12.3. На разных портах Pico доступны два контакта SPI.

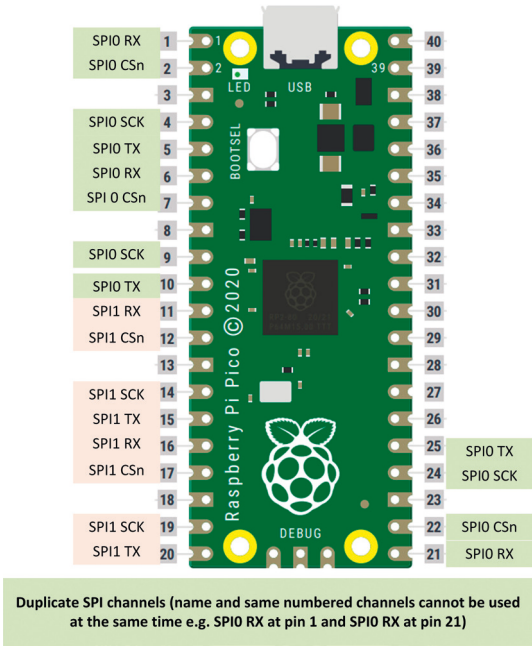


Рис.*12.3:* Распиновка шины *SPI Raspberry Pi Pico W.*

Следующие контакты используются для подключения модуля RC522 к Pico:

RC522 Module	**Raspberry Pi Pico W**
SDA (CS)	GP1
SCK	GP2
MOSI	GP3
MISO	GP4
GND	GND
RST	GP0
VCC	+3.3 V
IRQ не используется	

12.4 Проект 1: Поиск идентификатора тега

Описание: В этом проекте будет отображаться идентификатор метки поставляемой метки RFID на монитор порта.

Блок-схема: На рис. 12.4 показана блок-схема проекта.

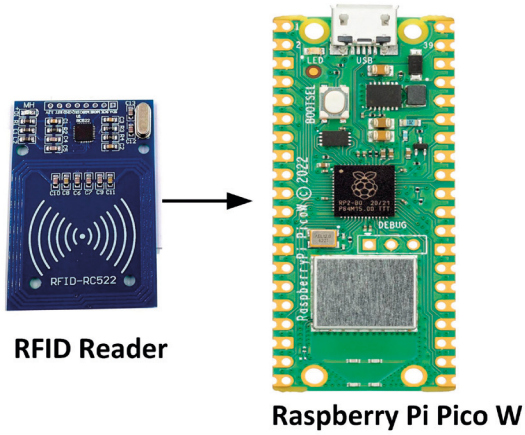


Рис.12.4: Блок-схема проекта.

Принципиальная схема: Соединения между Pico и считывателем RFID следующие (будьте осторожны, чтобы не подключить контакт питания к +5 В). На рис. 12.5 показана принципиальная схема проекта:

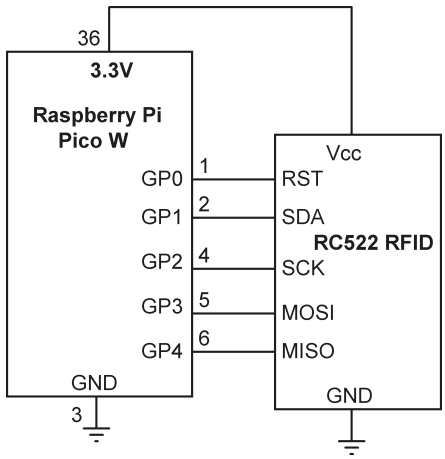


Рис.12.5: Принципиальная схема проекта.

Список программ: Прежде чем использовать считыватель RFID, вы должны добавить библиотеку RFID в свой Pico. Библиотека называется **mfrc522.py** и доступна в Интернете. Копия этой библиотеки размещена на веб-сайте книги. Вы должны скопировать эту библиотеку на свой Pico с точным именем **mfrc522.py**.

Листинг программы показан на рис.12.6 (Программа: **RFID.py**). В начале программы импортируется библиотека **mfrc522**. Затем программа отображает сообщение, чтобы поместить метку ближе к считывателю. Считыватель инициализируется, и на него отправляется запрос. Если тег обнаружен, он выбирается, а идентификатор тега считывается и отображается на экране Thonny.

```
#--------------------------------------------------------------------------------
#        ПРОГРАММА RFID ДЛЯ ОТОБРАЖЕНИЯ ID ТЕГА
#        ======================================
#
# Эта программа отображает ID тега тега, поставляемого со считывателем RFID.
#
# Автор: Доган Ибрагим
# Файл   : RFID.py
# Дата: октябрь 2022 г.
#--------------------------------------------------------------------------------
from mfrc522 import MFRC522

reader = MFRC522(spi_id=0,sck=2,miso=4,mosi=3,cs=1,rst=0)

print("Put Tag near the reader...")

readflag = 0
while readflag == 0:
    reader.init()
    status, tag_type = reader.request(reader.REQIDL)

    if status == reader.OK:
        status, uid = reader.SelectTagSN()

        if status == reader.OK:
            card = int.from_bytes(bytes(uid),"little",False)
            print("Your Card ID (Decial): " + str(card))
            print("Your Card ID (HEX):", hex(card))
            readflag = 1
```

Рис.*12.6:* Программа:**RFID.py**.

На рис.12.7 показан идентификатор тега, отображаемый в виде десятичного числа: 453492048 или шестнадцатеричного числа: 0x1b07bd50

```
Put Tag near the reader...
Your Card ID (Decial): 453492048
Your Card ID (HEX): 0x1b07bd50

>>>
```

Рис.12.7: Отображение идентификатора тега.

12.5 Проект 2: Доступ к дверному замку RFID с реле

Описание: В этом проекте считыватель RFID и реле подключены к Pico. Предполагается, что вход в защищенную дверь управляется реле и исполнительным механизмом и защищен системой RFID. Размещение авторизованной карты тегов рядом со считывателем RFID может только открыть дверь. Реле активируется на 15 секунд. По истечении этого времени он деактивируется, чтобы дверь можно было закрыть.

Блок-схема: На рис. 12.8 показана блок-схема проекта.

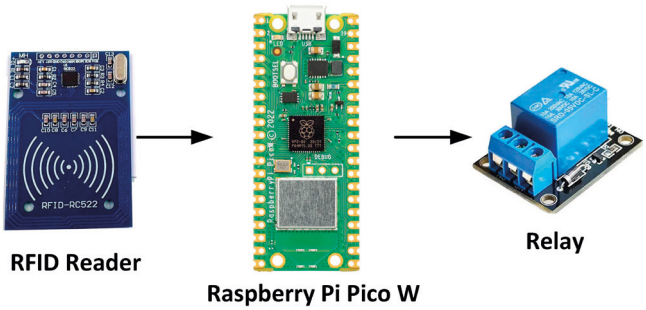


Рис.12.8: Блок-схема проекта.

Принципиальная схема: Принципиальная схема показана на рис. 12.9. Схема в основном то же, что и на рис. 12.5, но здесь к порту GP16 добавлено реле.

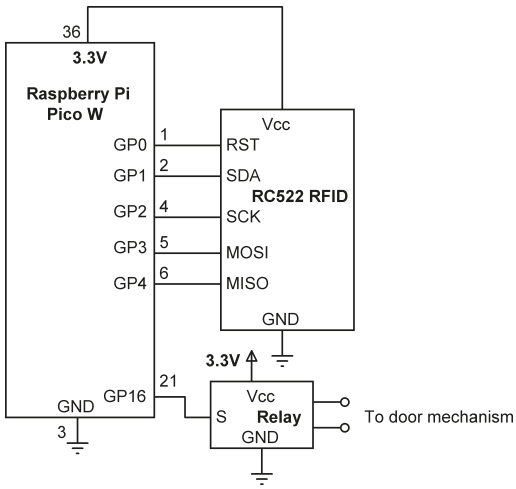


Рис.12.9: Принципиальная схема проекта.

Листинг программы: На рис.12.10 показан листинг программы (Программа: RFIDLock). В начале работы в программу включены библиотеки SPI и MFRC522. Действительный идентификатор карты хранится в строке ValidCard, RELAY назначается порту GP16 и деактивируется. Остальная часть программы выполняется внутри основного цикла программы. Здесь программа ждет, пока метка не будет помещена рядом со считывателем. При размещении тега его идентификатор считывается и сравнивается с ValidCard. Если есть совпадение, предполагается, что метка авторизована, а затем активируется RELAY на 15 секунд, что, в свою очередь, приводит к открытию двери. Если карта недействительна, реле остается деактивированным. Этот процесс повторяется вечно.

```
#-------------------------------------------------------------------------------
#        ДОСТУП К ЗАМКУ RFID
#        ===================
#
#  Он активирует реле, если авторизованная действующая карта находится рядом с
#  RFID-считывателем. Реле остается активным в течение 15 секунд.
#
# Автор: Доган Ибрагим
# Файл   : RFIDLock.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------------------------
from mfrc522 import MFRC522
import utime
from machine import Pin

reader = MFRC522(spi_id=0,sck=2,miso=4,mosi=3,cs=1,rst=0)

ValidCard = 453492048                    # Авторизованный идентификатор карты
RELAY = Pin(16, Pin.OUT)                 #  РЕЛЕ на GP16
RELAY.value(0)                           # Деактивировать РЕЛЕ

while True:
    reader.init()
    status, tag_type = reader.request(reader.REQIDL)

    if status == reader.OK:
        status, uid = reader.SelectTagSN()

        if status == reader.OK:
            card = int.from_bytes(bytes(uid),"little",False)
            if card == ValidCard:
                RELAY.value(1)           #  РЕЛЕ ВКЛ.
                utime.sleep(15)          # ждите 15 секунд
                RELAY.value(0)           # РЕЛЕ ВЫКЛ.
```

Рис.*12.10:* Программа*:***RFIDLock***.

12.6 Проект 3: Многометочная система доступа RFID с ЖК-дисплеем

Описание: В этом проекте считыватель RFID и реле подключены к Pico, как и в предыдущем проекте. Дополнительно подключен LCD на базе I2C. Проект может поддерживать множество пользователей с разными идентификаторами тегов. Имена уполномоченных лиц и их идентификаторы тегов хранятся в программе. Только один из этих уполномоченных лиц может активировать реле, чтобы открыть дверь.

Блок-схема: На рис. 12.11 показана блок-схема проекта.

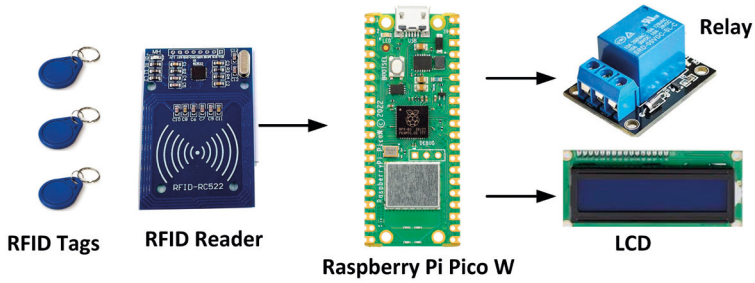


Рис.12.11: Блок-схема проекта.

Принципиальная схема: Принципиальная схема аналогична рисунку 12.9, но здесь к Pico дополнительно подключен I2C, как показано на рисунке 12.12. ЖК-дисплей подключен к контактам I2C1, а RFID подключен к контактам I2C0. Обратите внимание, что микросхема преобразователя напряжения TXS0102 используется между LCD и Pico.

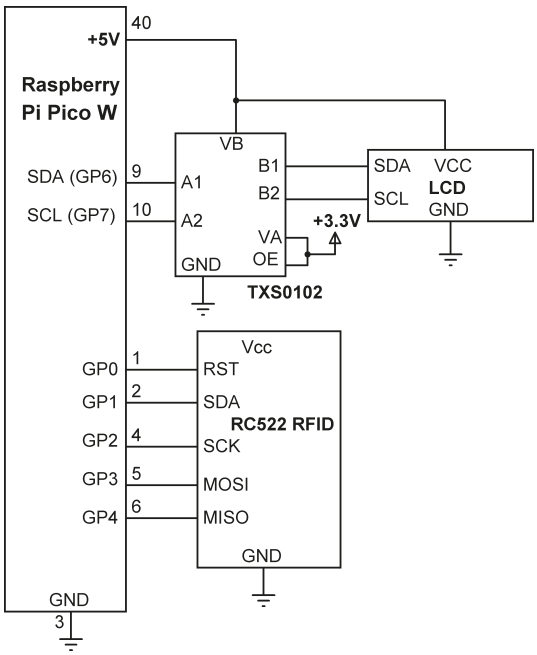


Рис.12.12: Принципиальная схема.

Листинг программы: На рис.12.13 представлен листинг программы (Программа: **members.py**). В начале программы инициализируются ЖК-дисплей и шина RFID I2C, РЕЛЕ настраивается на GP16 как выход. Затем создается словарь с элементами имени. В этом словаре хранятся идентификационные номера тегов и имена людей, которым разрешено иметь теги. Хотя в словаре отображаются только пять имен и идентификаторов тегов, его можно легко расширить. В этом проекте автором идентификатора тега является лицо по имени **Mary Walker**.

```
members={
    123456789: "Jones Smith",
    453492048: "Mary Walker",
    554433765: "Susan Richard",
    545567664: "Mark Beasley",
    455545321: "David Johnson"
}
```

При запуске программы пользователю предлагается поднести свою метку к считывателю, а на ЖК-дисплее отображается сообщение **Touch your Tag** (рис. 12.14). Программа считывает идентификатор тега (card) и сравнивает его с идентификаторами, перечисленными в словаре, используя оператор **member.get (card)**. Если пользователь не авторизован, возвращается **None** и на ЖК-дисплее отображается сообщение **Not Authorized**. Процесс повторяется через 5 секунд, когда пользователи должны коснуться своих тегов. Если, с другой стороны, тег авторизован, то в верхней строке ЖК-дисплея отображается текст **Authorized**. Нижний ряд ЖК отображает имя уполномоченного лица (рис. 12.15). В то же время РЕЛЕ активируется на 10 секунд, чтобы дверь открылась. Через 10 секунд РЕЛЕ деактивируется, и процесс повторяется.

```
#-------------------------------------------------------------
#          МУЛЬТИТЕГИЧЕСКИЙ КОНТРОЛЬ ДОСТУПА С ЖКИ
#          ==========================================
#
#  В этой программе имена и идентификаторы тегов авторизованных пользователей
# перечислены в словаре. Реле активируется, чтобы открыть дверь,
# когда авторизованный пользователь размещает свою метку возле двери.
#  Имя авторизованного пользователя отображается на ЖК-дисплее
## Автор: Доган Ибрагим
# Файл  : members.py
# Дата: октябрь 2022 г.
#-------------------------------------------------------------
import machine
from machine import I2C, Pin
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
import utime
from mfrc522 import MFRC522



#
#  Настройка RFID I2C
#
reader = MFRC522(spi_id=0,sck=2,miso=4,mosi=3,cs=1,rst=0)

#
# Настройка LCD I2C
```

```
#
I2C_ADDR = 0x27                    # I2C LCD адрес
NRows = 2                          # Количество строк
NColumns = 16                      # Количество столбцов

i2c = I2C(1, sda=machine.Pin(6), scl=machine.Pin(7), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, NRows, NColumns)

RELAY = Pin(16, Pin.OUT)           # Реле как выход
RELAY.value(0)                     # Деактивировать реле

#
# Введите допустимые идентификаторы тегов и имена авторизованных участников,
# как в следующем словаре. Вы можете расширить эту таблицу по мере необходимости
#
members={
    123456789: "Jones Smith",
    453492048: "Mary Walker",
    554433765: "Susan Richard",
    545567664: "Mark Beasley",
    455545321: "David Johnson"
}

lcd.clear()
lcd.putstr("Touch your Tag")

#
# Основная программа. Прочтите тег и проверьте, авторизован ли он. Если это так,
# отобразите имя уполномоченного лица и активируйте реле на 10 с.
#
while True:
    reader.init()
    status, tag_type = reader.request(reader.REQIDL)
    if status == reader.OK:
        status, uid = reader.SelectTagSN()

        if status == reader.OK:
            card = int.from_bytes(bytes(uid),"little",False)
            auth = members.get(card)
            lcd.clear()
            lcd.move_to(0, 0)
            if auth == None:
                lcd.putstr("Not Authorized")
                utime.sleep(5)
                lcd.clear()
            else:
```

```
            lcd.putstr("Authorized")
            lcd.move_to(0, 1)
            lcd.putstr(auth)
            RELAY.value(1)
            utime.sleep(10)
            RELAY.value(0)
            lcd.clear()
        lcd.putstr("Touch your Tag")
```

Рис.*12.13:* Программа:**members.py.**

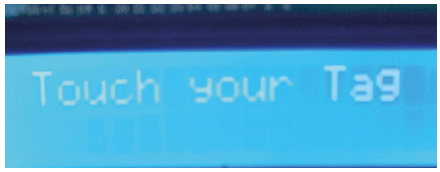


Рис.12.14: Отображение текста Touch your Tag - Коснитесь своего тега.

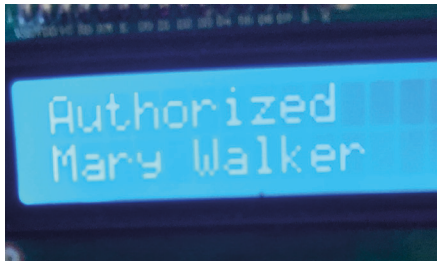


Рис.12.15: Авторизованный пользователь.

Предложение для дальнейшей работы: Программу можно изменить так, чтобы имена авторизованные пользователи и их идентификаторы тегов хранились в файле на Raspberry Pi Pico W.

Raspberry Pi Pico W

Программируйте,создавайте и управляйте проектами с помощью **Wireless RP2040**

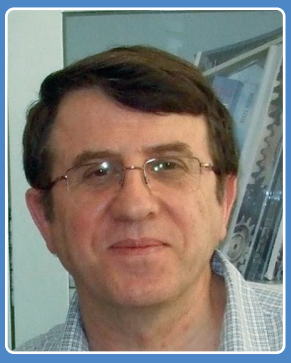
Raspberry Pi Pico и Pico W основаны на быстром, эффективном и недорогом двухъядерном микроконтроллере ARM Cortex M0+ RP2040, работающем на частоте до 133 МГц и имеющем 264 КБ SRAM и 2 МБ флэш-памяти. Помимо большой памяти, Pico и Pico имеют множество контактов GPIO и популярные модули периферийных интерфейсов, такие как ADC, SPI, I2C, UART, PWM, модули синхронизации, интерфейс аппаратной отладки и внутренний датчик температуры.

Raspberry Pi Pico W дополнительно включает в себя встроенный чипсет Infineon CYW43439 Bluetooth и Wi-Fi. На момент написания этой книги прошивка Bluetooth еще не была доступна. Однако Wi-Fi полностью поддерживается на частоте 2,4 ГГц с использованием протоколов 802.11b/g/n.

Эта книга представляет собой введение в использование Raspberry Pi Pico W в сочетании с языком программирования MicroPython. Среда разработки Thonny (IDE) используется во всех более чем 60 рабочих и протестированных проектах, охватывающих следующие темы:

- Установка MicroPython на Raspberry Pi Pico с помощью Raspberry Pi или ПК
- Прерывания по таймеру и внешние прерывания
- Проекты аналого-цифрового преобразователя (АЦП)
- Использование внутреннего датчика температуры
- Использование датчика внутренней температуры и датчика внешней температуры
- Проекты регистрации данных
- Проекты PWM, UART, I²C и SPI
- Использование Bluetooth, WiFi и приложений для связи со смартфонами
- Проекты цифро-аналогового преобразователя (ЦАП)

Все проекты проверены. Они могут быть реализованы как на Raspberry Pi Pico, так и на Raspberry Pi Pico W, хотя объекты на основе Wi-Fi будут работать только на Pico W. Базовый опыт программирования и электроники необходим для выполнения проектов. Для всех проектов даны краткие описания, блок-схемы, подробные принципиальные схемы и полные списки программ MicroPython.



Профессор Доган Ибрагим имеет степень бакалавра наук с отличием, степень в области электронной инженерии, степень магистра в области техники автоматического управления и степень доктора философии в области цифровой обработки сигналов.

Доган работал во многих промышленных организациях, прежде чем вернуться к академической жизни. Он является автором более 70 технических книг и опубликовал более 200 технических статей по электронике, микропроцессорам, микроконтроллерам и смежным областям.

ISBN 978-3-89576-529-2
9 783895 765292

elektor
design > share > earn