# Шины передачи данных

1. I2C

В ESP32 есть возможность использовать любые из выводов GPIO платы для подключения датчиков и устройств, также в ESP32 имеется 2 канала I2C, обратится которым можно через идентификационным номерам 0 и 1.

Но по стандарту адреса GPIO находятся согласно таблице.

|  |  |  |
| --- | --- | --- |
|  | I2C(0) | I2C(1) |
| SCL | 18 | 25 |
| SDA | 19 | 26 |

Пример:

**from** **machine** **import** Pin, I2C

#Обрушение к первой шине I2C по стандартному адресу SCL=18 SDA=19

i2c = I2C(0)

#Обрушение к второй шине I2C с указанными адресами

i2c = I2C(1, scl=Pin(5), sda=Pin(4), freq=400000)

Стандартные функций библиотеки I2C:

1. .init(scl, sda, \*, freq=400000) - Инициализируйте шину I2C с заданными аргументами.

SCL, SDA - Пины, используемые для сигналов шины

freq – скорость передачи данных

1. .deinit() - Выключите шину I2C.
2. .start() – Начать работу с подключённым устройством
3. .stop() – Закончить работу с подключённым устройством
4. .scan() – Сканирование на наличие подключённых устройств
5. .readfrom(0x3a, 4) - Считывание 4 байт с устройства с адресом 0x3a
6. .writeto(0x3a, '12') - Записать '12' на устройство с адресом 0x3a
7. readfrom\_mem(addr, memaddr, nbytes, \*, addrsize=8) –

Считывает nbytes с периферийного устройства, указанного addr, начиная с адреса памяти, указанного memaddr. Аргумент addrsize задает размер адреса в битах. Возвращает объект bytes с прочитанными данными.

1. .writeto\_mem(addr, memaddr, buf, \*, addrsize=8)¶-

Считывание в buf с периферийного устройства, указанного addr, начиная с адреса памяти, указанного memaddr. Количество прочитанных байтов равно длине buf. Аргумент addrsize определяет размер адреса в битах (в ESP8266 этот аргумент не распознается, и размер адреса всегда равен 8 битам).

## Пример работы с датчиком BME280.

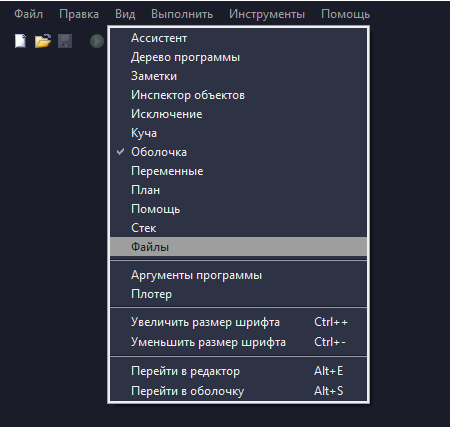
О датчике BME280.

Данный датчик имеет возможность предоставить данные о атмосферно давлений, влажности воздуха и температуры.

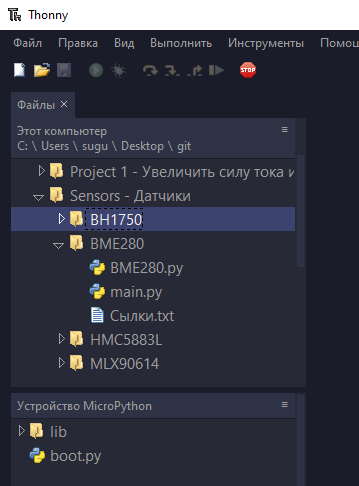
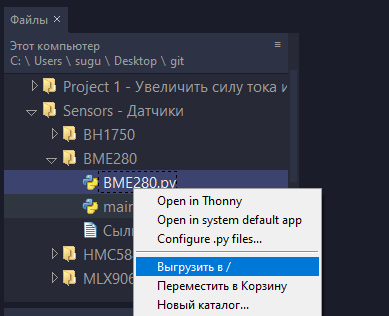
Получения данных осуществляется через шину I2C.

Для получения данных необходимо загрузить библиотеку для работы с датчиком.

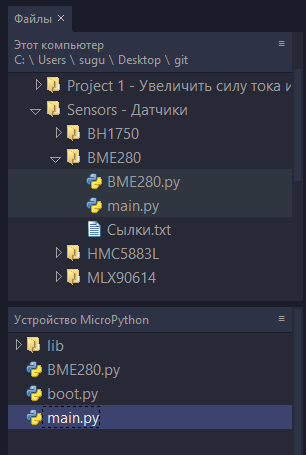
Чтобы загрузить библиотеку вручную требуется открыть путь к файлу в компьютере через встроенный проводник в среде разработки Thonny IDE *вид=>Файлы*.



Откроется проводник в окне Thonny, далее с приложенным отчетом в папке “Sensors\BME280” копируем файлы BME280.py и main.py нажав правой кнопкой мыши в пункте “выгрузить в /”

После этого в Устройстве MicroPython появистя скопированые файлы.



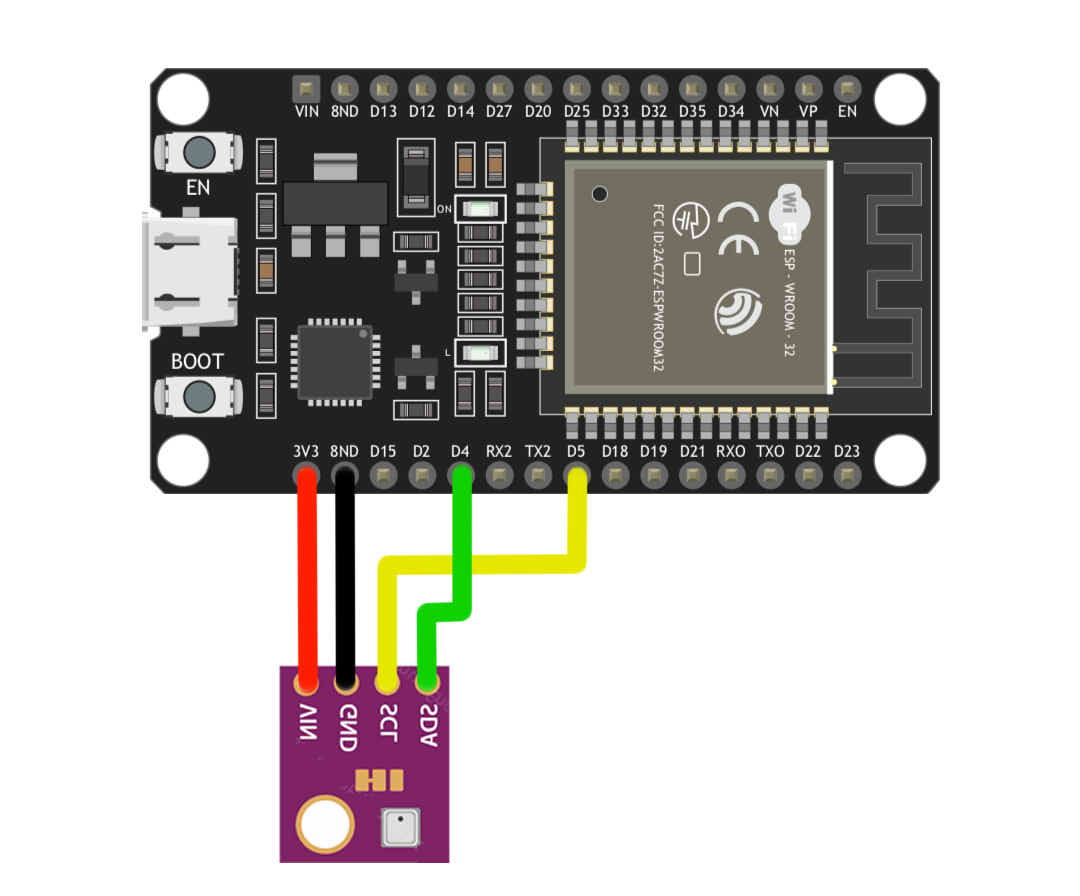
Далее подключаем датчик BME280 к ESP32 согласно схеме ниже.

VCC – 3.3v

GND – GND

SCL – D5

SDA – D4

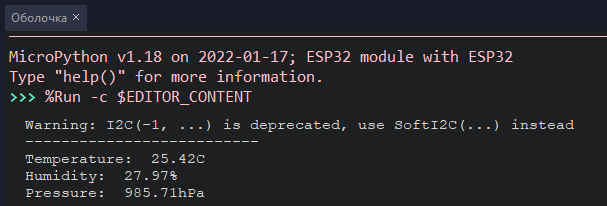


Открываем файл “main.py” в Устройстве MicroPython.

И запускаем программу нажав F5 или на кнопку “Запустить текущий скрипт”



И в результате в терминале Thonny будет выявлена информация о среде.



1. SPI

Существует два аппаратных канала SPI, которые обеспечивают более высокую скорость передачи (до 80 МГц).

Они могут использоваться на любых выводах ввода-вывода, которые поддерживают требуемое направление и в противном случае не используются (см. Контакты и GPIO), но если они не настроены на свои контакты по умолчанию, то им необходимо пройти через дополнительный уровень мультиплексирования GPIO, что может повлиять на их надежность на высоких скоростях. Аппаратные каналы SPI ограничены частотой 40 МГц при использовании на выводах, отличных от перечисленных в таблице по умолчанию.

|  |  |  |
| --- | --- | --- |
|  | HSPI (id=1) | VSPI (id=2) |
| Sck | 14 | 18 |
| mosi | 13 | 23 |
| miso | 12 | 19 |

Пример:

**from** **machine** **import** Pin, SPI

#Обрушение к первой шине SPI c адресами по умолчанию

hspi = SPI(1, 10000000)

#Обрушение к первой шине SPI c указанным адресом

hspi = SPI(1, 10000000, sck=Pin(14), mosi=Pin(13), miso=Pin(12))

#Обрушение к первой шине SPI c указанным адресом, буфером и тд

vspi = SPI(2, baudrate=80000000, polarity=0, phase=0, bits=8, firstbit=0, sck=Pin(18), mosi=Pin(23), miso=Pin(19))

Стандартные функций библиотеки SPI:

1. . init(baudrate=1000000, \*, polarity=0, phase=0, bits=8, firstbit=SPI.MSB, sck=None, mosi=None, miso=None, pins=(SCK, MOSI, MISO))¶

Инициализировать шину SPI с заданными параметрами:

baudrate - Тактовая частота SCK.

polarity - Может быть 0 или 1, и это уровень, на котором находится линия часов простоя.

phase - Может быть 0 или 1 для выборки данных на первом или втором фронте синхронизации соответственно.

bits - Ширина в битах передачи. Только 8 бит гарантированно поддерживается всеми пинами.

Firstbit - Может быть SPI.MSB (установите первый бит как самый значимый бит) или SPI.LSB (установите первый бит в качестве младшего значащего бита).

sck, mosi, miso - Пины, используемые для сигналов шины.

1. . deinit() - Выключите шину SPI
2. . read(nbytes, write=0x00) - Считывает количество байтов, указанное nbytes, при непрерывной записи одного байта, заданного write. Возвращает объект bytes с данными, которые были прочитаны.
3. .readinto(buf, write=0x00) - Считывание в буфер, указанный buf, при непрерывной записи одного байта, заданного write.
4. .write(buf) - Запишите байты, содержащиеся в buf
5. .write\_readinto(write\_buf, read\_buf) - Записывайте байты из write\_buf во время чтения в read\_buf. Буферы могут быть одинаковыми или разными, но оба буфера должны иметь одинаковую длину.

## Пример работы с термодатчиком MAX6672

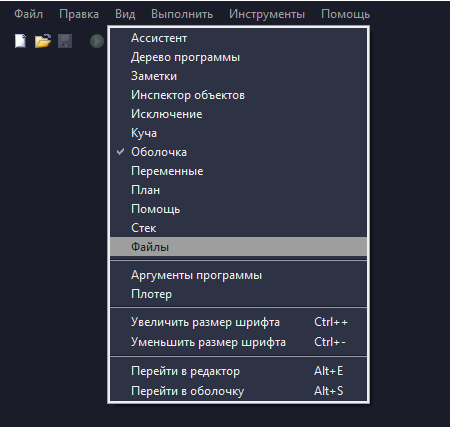
О датчике MAX6672.

Данный датчик имеет возможность измерять температуры за счет изменения разности потенциалов между двух проводников.

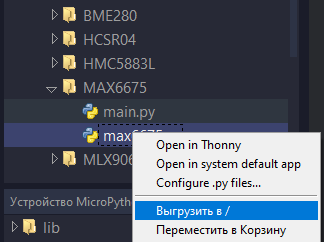
Получения данных осуществляется через шину SPI.

Для получения данных необходимо загрузить библиотеку для работы с датчиком.

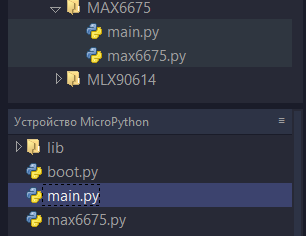
Чтобы загрузить библиотеку вручную требуется открыть путь к файлу в компьютере через встроенный проводник в среде разработки Thonny IDE *вид=>Файлы*.



Откроется проводник в окне Thonny, далее с приложенным отчетом в папке “Sensors\ MAX6675” копируем файлы max6675.py и main.py нажав правой кнопкой мыши в пункте “выгрузить в /”

После этого в Устройстве MicroPython появистя скопированые файлы.



Далее подключаем датчик MAX6672 к ESP32 согласно схеме ниже.

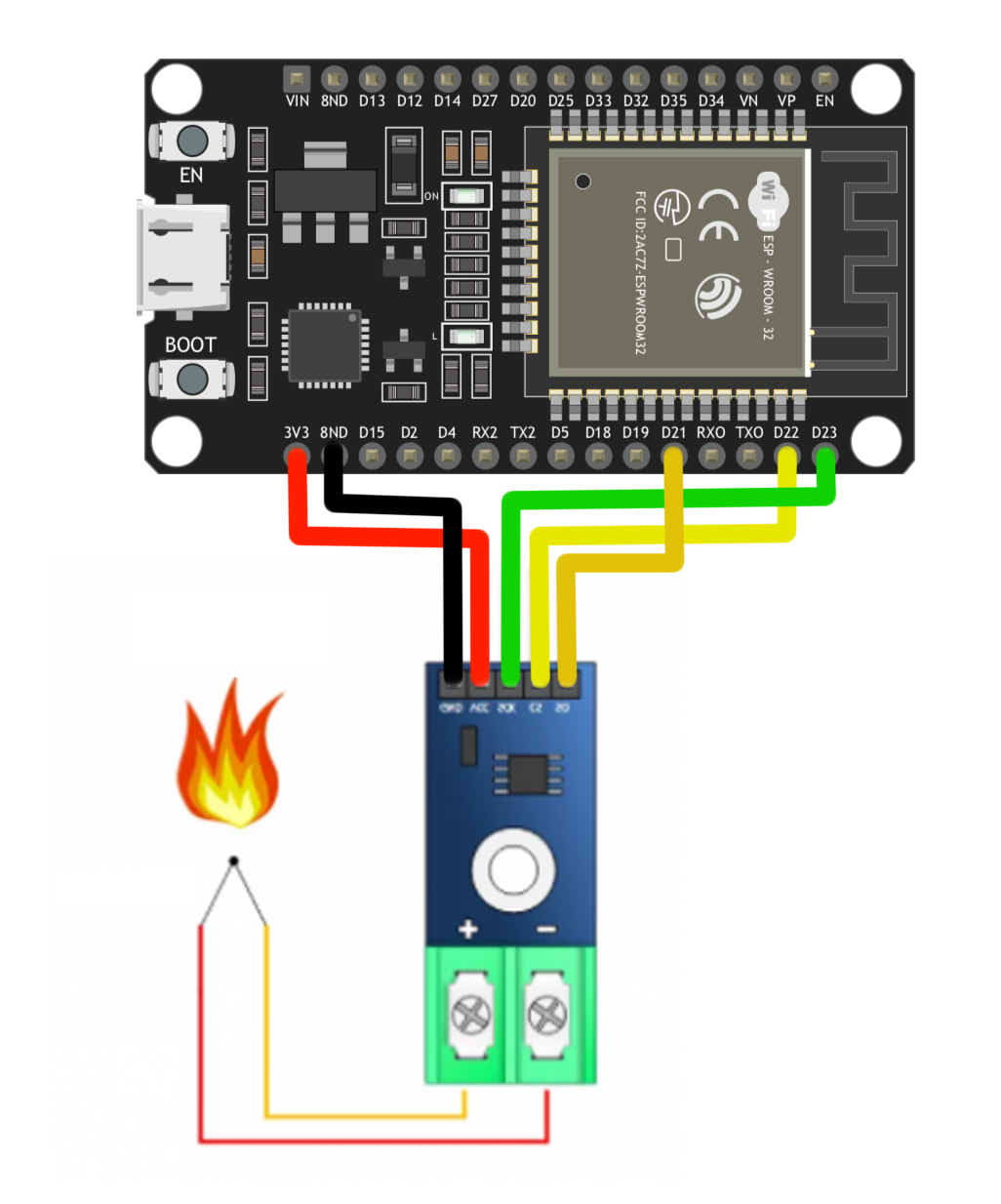
VCC – 3.3v

GND – GND

SCK – D23

CS – D22

SO – D21

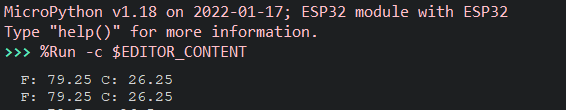


Открываем файл “main.py” в Устройстве MicroPython.

И запускаем программу нажав F5 или на кнопку “Запустить текущий скрипт”



И в результате в терминале Thonny будет выявлена информация о температуре формате фаренгейта и целисиях.



1. URTA

ESP32 имеет три аппаратных интерфейса UART: UART0, UART1 и UART2. Каждому из них назначен GPIO по умолчанию, однако в зависимости от вашего варианта ESP32 может конфликтовать со встроенной flash-памятью, встроенной PSRAM или периферийными устройствами.

Любой GPIO-пин может использоваться для аппаратных UART, для этого просто укажите контакты tx и rx.

GPIO по умолчанию перечислены в таблице.

|  |  |  |  |
| --- | --- | --- | --- |
|  | UART0 | UART1 | UART2 |
| tx | 1 | 10 | 17 |
| rx | 3 | 9 | 16 |

Пример:

**from** **machine** **import** UART

#Инициализируем первый UART c указанным адресом rx и tx

#со скорость передачи данных 9600

uart1 = UART(1, baudrate=9600, tx=33, rx=32)

# Записываем 5 бит из UART

uart1.write('hello')

# Считываем 5 бит из UART

uart1.read(5)

Стандартные функций библиотеки UART:

1. .init(n, baudrate, rx, tx) –

Инициализация шины UART с параметрами:

n – Номер шины UART

baudrate – Скорость передачи данных

rx, tx – GPIO пины подключения

1. .deinit() - Выключите шину UART
2. .read([nbytes]) - Читать символы. Если указано значение nbytes, то считывается не более этого количества байт, в противном случае считывается как можно больше данных.
3. .readinto(buf[, nbytes]) - Считывание байтов в буфер обмена. Если указано значение nbytes, то считывается не более этого количества байт.
4. .readline() - Прочитайте строку, заканчивающуюся символом новой строки.
5. .write(buf) - Записать буфер байтов.

## Пример на ESP32 и Pico

Задача: отправить данные с ESP32 на Pico через UART.

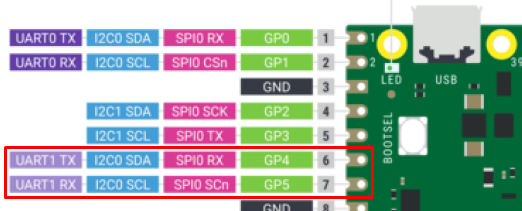
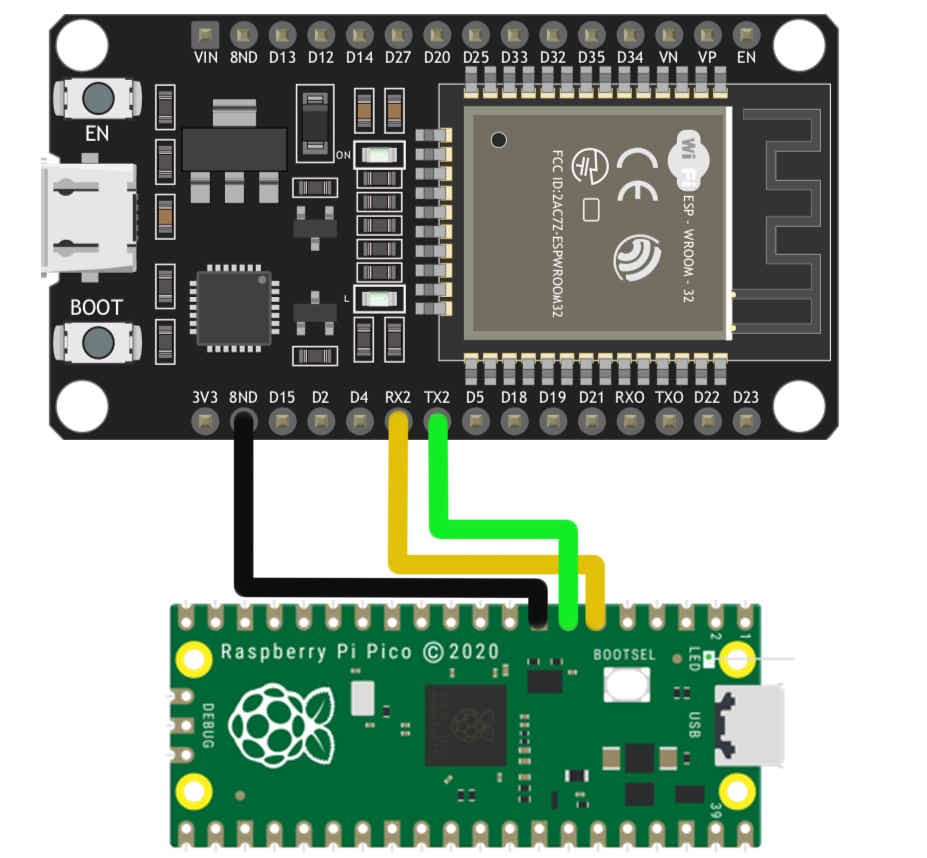
Подключаем платы согласно схеме.

ESP32 – Pico

RX2 – 7 (GPIO5)

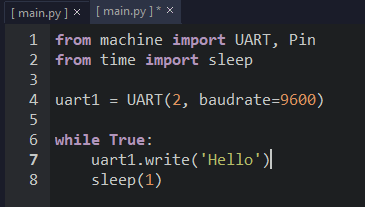
TX2 – 6 (GPIO4)

GND – GND

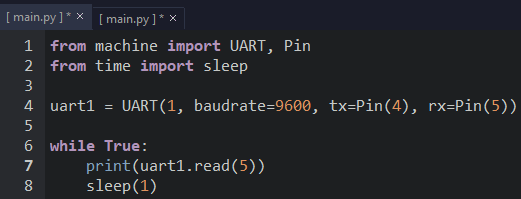


Далее загружаем код программы в платы.

Для ESP32.

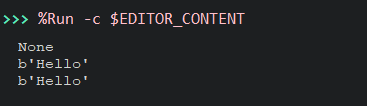


Для Pico



В данном примере в качестве данных ESP32 отправляется текст “Hello”.

И как результат на терминале Pico выводится полученные данные.



Примечание: если от ESP32 нет данных, то перезагрузите ESP32 нажав на кнопку EN на плате или отключив и снова включив питание.