Лабораторная работа № 19

Тема: Программирование Raspberry Pi с использованием Python. Знакомство с GPIO.

Цель: Познакомиться с основами программирования Raspberry Pi с использованием языка Python, получить практические навыки работы с GPIO.

Используемое оборудование: ПК с установленным Proteus.

Raspberry Pi - это небольшой одноплатный компьютер, который можно использовать для множества проектов, начиная от простых автоматов до создания умного дома и робототехники.

Существуют различные модификации Raspberry Pi, а также множество клонов (Orange Pi, Banana Pi, и др.). В данной работе будет использоваться Raspberry Pi 3 с ОС Raspbian.

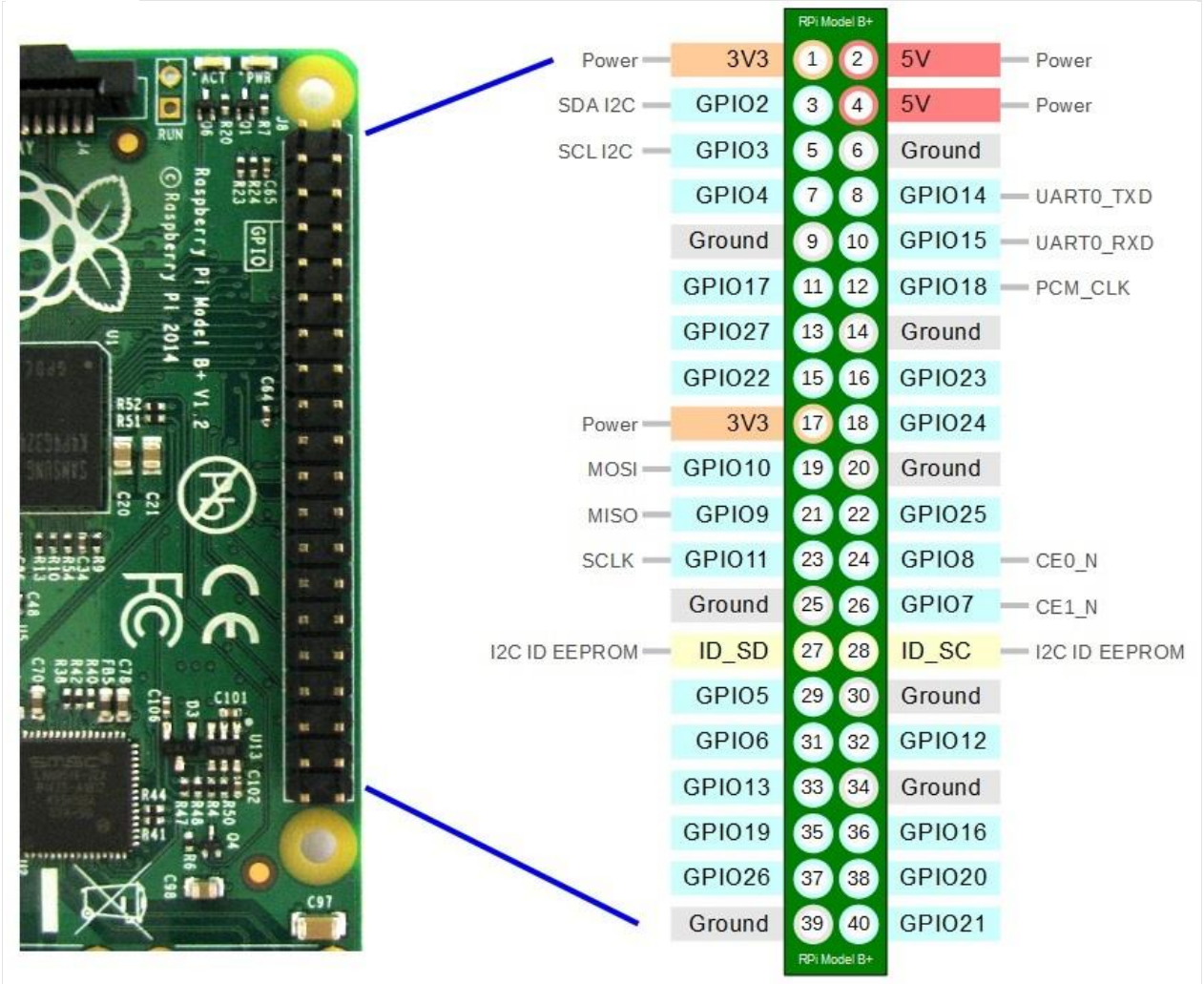
Характеристики Raspberry Pi 3:

Процессор: 4-ядерный ARM Cortex-A53 с тактовой частотой 1,2 ГГц
Оперативная память: 1 ГБ LPDDR2 SDRAM
Видео: Broadcom VideoCore IV
Память: MicroSD
Подключение: HDMI, Ethernet, Wi-Fi, Bluetooth
Питание: 5 В через microUSB
Размеры: 85 x 56 x 17 мм
Операционная система: Raspbian, Ubuntu MATE, Windows 10 IoT Core, OpenELEC и др.
26 контактов GPIO

GPIO (General Purpose Input/Output) — это универсальные входы/выходы на платах Raspberry Pi, которые позволяют взаимодействовать с различными электронными компонентами, такими как светодиоды, кнопки, датчики, моторы и многие другие. Использование GPIO делает Raspberry Pi мощным инструментом для создания различных проектов и устройств.

Основы GPIO на Raspberry Pi

1. **Пины**: Raspberry Pi имеет несколько пинов GPIO, которые расположены на 40-пиновом (или 26-пиновом для старых моделей) разъеме на плате. Эти пины могут быть настроены на вход или выход, и они работают с напряжением 3.3 В.
2. **Библиотеки для работы с GPIO**:
    - **RPi.GPIO**: Это стандартная библиотека для работы с GPIO на Python. Она позволяет контролировать пины GPIO как входы и выходы, а также поддерживает прерывания.
    - **GPIO Zero**: Это более высокоуровневая библиотека по сравнению с RPi.GPIO. Она упрощает многие задачи, такие как контроль светодиодов, кнопок и других компонентов.



Пины в GPIO могут выполнять 3 функции:

- подача электричества определенного напряжения;
- заземление;

- прием/отправка сигналов.

Интересно то, что за вход и выход в интерфейсе могут отвечать одни и те же контакты. По крайней мере это справедливо для RPi. То, как они себя должны вести, определяется программой.

Теперь можно перейти к вопросу, который касается того, какая распиновка GPIO имеется на Raspberry Pi 3. В первую очередь необходимо сказать, что общее количество пинов на соответствующей панели равняется 40. Каждый из них имеет свой номер.

Все они подразделяются на 3 группы. К первой относятся питающие (на англоязычных схемах маркируются как Power) – они подают электричество в 3,3 и 5 Вольт. При этом у разных контактов данного назначения различное напряжение. Это обязательно следует учитывать при подключении модулей.

Ко второй – заземляющие (могут именоваться RND или Ground). Они нужны, чтобы отводить электричество, тем самым обеспечивая безопасное использование.

К третьей – порты (имеют обозначение BCM). Именно они служат теми контактами, которые могут принимать и отправлять сигналы. Пользователь может подключать модули к любым из них. Самое главное – чтобы было удобно обеспечивать питание подсоединённых компонентов.

Важно знать, что в системе есть пара контактов, которые зарезервированы системой для особых целей. Ими являются BCM 0 и BCM 1 (их номера на обычной схеме – 27 и 28). Они предназначаются специально для установки плат расширения. Поэтому при возможности их не нужно использовать для подключения других модулей.

Raspberyy Pi 3 имеет различные порты ввода-вывода, такие как GPIO, SPI, I2C, UART, USB, Wi-Fi, Ethernet, HDMI CSI (Camera Serial Interface). Некоторые из них выведены на GPIO:

SPI (Serial Peripheral Interface):

На Raspberry Pi 3 интерфейс SPI доступен через следующие пины:

- **SPI0 (SPI библиотека)**:
  - MOSI (Master Out Slave In): GPIO 10 (физический пин 19)
  - MISO (Master In Slave Out): GPIO 9 (физический пин 21)
  - SCLK (Serial Clock): GPIO 11 (физический пин 23)
  - CE0 (Chip Enable 0): GPIO 8 (физический пин 24)
  - CE1 (Chip Enable 1): GPIO 7 (физический пин 26)
- **SPI1 (доступен на GPIO пинах)**:
  - MOSI: GPIO 20 (физический пин 38)
  - MISO: GPIO 19 (физический пин 35)
  - SCLK: GPIO 21 (физический пин 40)
  - CE0: GPIO 18 (физический пин 12)
  - CE1: GPIO 17 (физический пин 11)

I2C (Inter-Integrated Circuit):

Интерфейс I2C также доступен через GPIO пины на Raspberry Pi 3:

- **SDA (Data Line)**: GPIO 2 (физический пин 3)
- **SCL (Clock Line)**: GPIO 3 (физический пин 5)

UART (Universal Asynchronous Receiver-Transmitter):

Raspberry Pi 3 имеет несколько UART портов:

- **UART0 (используется для консоли и отладки)**:
  - TXD (Transmit Data): GPIO 14 (физический пин 8)
  - RXD (Receive Data): GPIO 15 (физический пин 10)
- **Дополнительные UART порты доступны на GPIO пинах**.

Как можно взаимодействовать с GPIO?

Работать с GPIO "Малины" можно посредством языков программирования. Вариантов здесь существует много. Самый лучший для GPIO Raspberry Pi 3 – Python. Это связано с тем, что для этого одноплатника данный ЯП является "родным". Но с не меньшим успехом с интерфейсом возможно взаимодействовать и посредством C/C++ и даже PHP или Basic.

Python - один из самых популярных языков программирования для работы с Raspberry Pi благодаря своей простоте и удобству.

Для работы с GPIO на Raspberry Pi с помощью Python существует несколько основных библиотек, которые делают взаимодействие с аппаратным обеспечением простым и доступным. Эти библиотеки предоставляют функции для контроля различных устройств, подключенных через GPIO пины, таких как светодиоды, кнопки, датчики, моторы и многие другие. Вот основные библиотеки, которые часто используются для этой цели:

1. RPi.GPIO
RPi.GPIO — это стандартная библиотека для работы с GPIO на Raspberry Pi. Она предоставляет базовые функции для настройки пинов GPIO, чтения и записи данных. Библиотека поддерживает как входные, так и выходные режимы, а также работу с прерываниями.

Пример использования RPi.GPIO:

```python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)  # Установка режима нумерации пинов
GPIO.setup(18, GPIO.OUT)  # Настройка пина 18 как выход

try:
    while True:
        GPIO.output(18, GPIO.HIGH)  # Включить светодиод
        time.sleep(1)
        GPIO.output(18, GPIO.LOW)  # Выключить светодиод
        time.sleep(1)
finally:
    GPIO.cleanup()  # Освобождение ресурсов
```

2. GPIO Zero

GPIO Zero — это высокоуровневая библиотека, созданная для упрощения работы с GPIO на Raspberry Pi, особенно для образовательных целей. Библиотека позволяет управлять GPIO с меньшим количеством кода и более интуитивно понятным интерфейсом по сравнению с RPi.GPIO.

Пример использования GPIO Zero:

```python
from gpiozero import LED
from time import sleep

led = LED(17)  # Создание объекта LED на пине 17

while True:
    led.on()  # Включить светодиод
    sleep(1)
    led.off()  # Выключить светодиод
    sleep(1)
```

3. pigpio

pigpio — это библиотека для работы с GPIO, которая особенно хороша для работы с PWM (широтно-импульсной модуляцией) и чтением данных с высокой точностью. Она работает в качестве демона, что позволяет управлять GPIO локально или через сеть.

Пример использования pigpio:

```python
import pigpio
import time

pi = pigpio.pi()  # Подключение к pigpio daemon
pi.set_mode(18, pigpio.OUTPUT)  # Установка пина 18 как выход

try:
    while True:
        pi.write(18, 1)  # Включить светодиод
        time.sleep(1)
        pi.write(18, 0)  # Выключить светодиод
        time.sleep(1)
finally:
    pi.stop()  # Остановить связь с демоном
```

Из-за ограничений Proteus будем использовать одну библиотеку - **RPi.GPIO.**
Вот основные функции и методы, которые предлагает библиотека RPi.GPIO:

Настройка библиотеки и пинов

- **GPIO.setmode(mode)**: Устанавливает схему нумерации пинов. Можно выбрать между **GPIO.BCM** (нумерация по номеру GPIO на чипе Broadcom) и **GPIO.BOARD** (нумерация по физическому расположению пинов на плате).
- **GPIO.setup(channel, direction, pull_up_down=GPIO.PUD_OFF, initial=GPIO.LOW)**: Настраивает пин как входной или выходной. Параметр **direction** может быть **GPIO.IN** или **GPIO.OUT**. Дополнительно можно указать внутренние подтягивающие резисторы (**GPIO.PUD_UP**, **GPIO.PUD_DOWN**) и начальное состояние для выходных пинов (**GPIO.HIGH**, **GPIO.LOW**).
- **GPIO.cleanup()**: Очищает настройки всех пинов или конкретных пинов, которые были настроены ранее. Это хорошая практика вызывать эту функцию в конце скрипта для освобождения ресурсов.

Управление GPIO

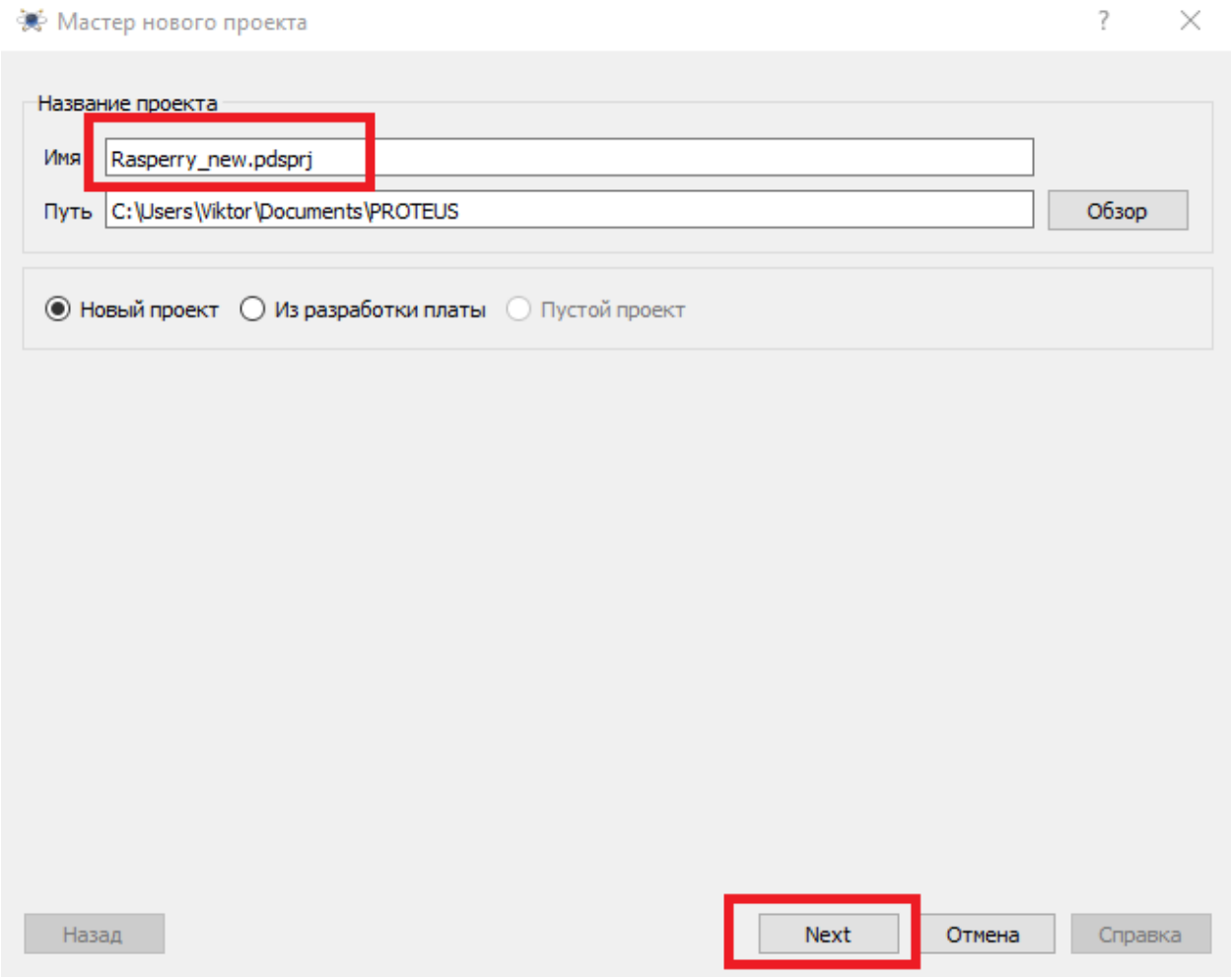
- **GPIO.input(channel)**: Читает текущее состояние входного пина. Возвращает **GPIO.HIGH** или **GPIO.LOW**.
- **GPIO.output(channel, state)**: Устанавливает состояние выходного пина. **state** может быть **GPIO.HIGH** или **GPIO.LOW**.
- **GPIO.add_event_detect(channel, GPIO.RISING, callback=my_callback, bouncetime=200)**: Настраивает прерывание, которое вызовет функцию **my_callback**, когда на пине произойдет изменение с **GPIO.LOW** на **GPIO.HIGH**. Параметр **bouncetime** позволяет игнорировать последующие срабатывания в течение указанного времени в миллисекундах. Можно также использовать **GPIO.FALLING** для срабатывания на переход от высокого к низкому уровню или **GPIO.BOTH** для любого изменения.
- **GPIO.remove_event_detect(channel)**: Удаляет прерывание для пина.

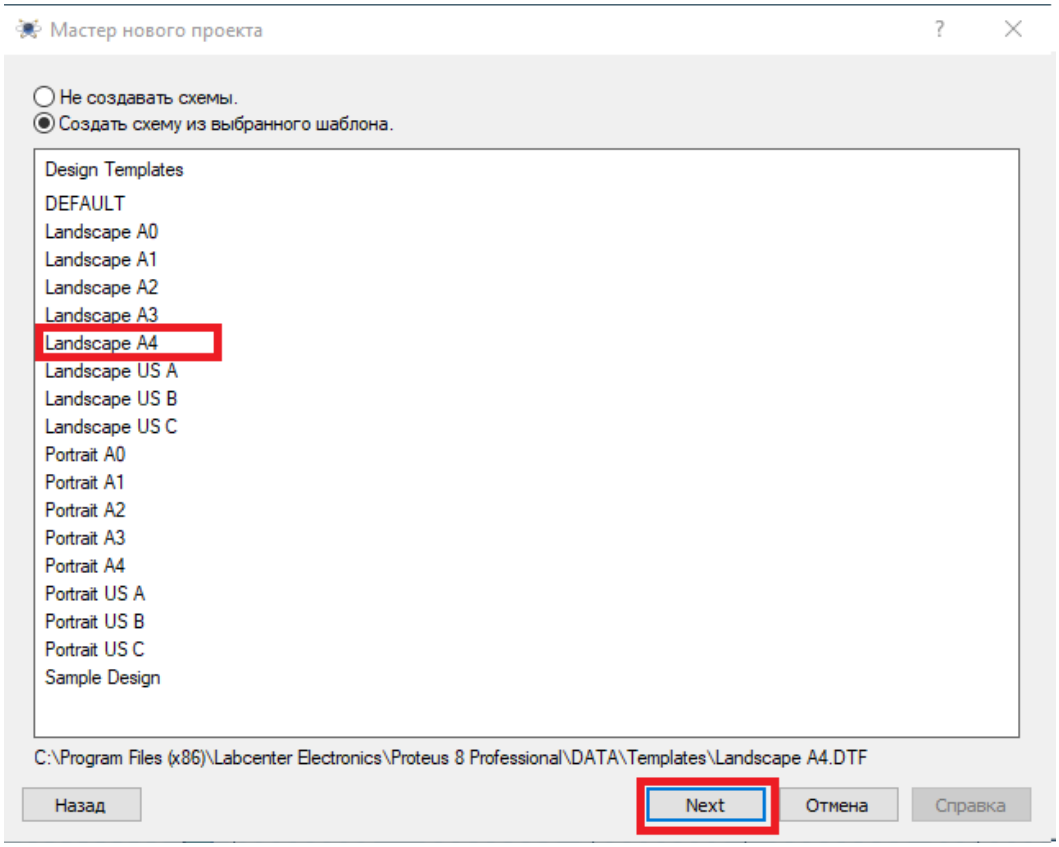
PWM (широтно-импульсная модуляция)

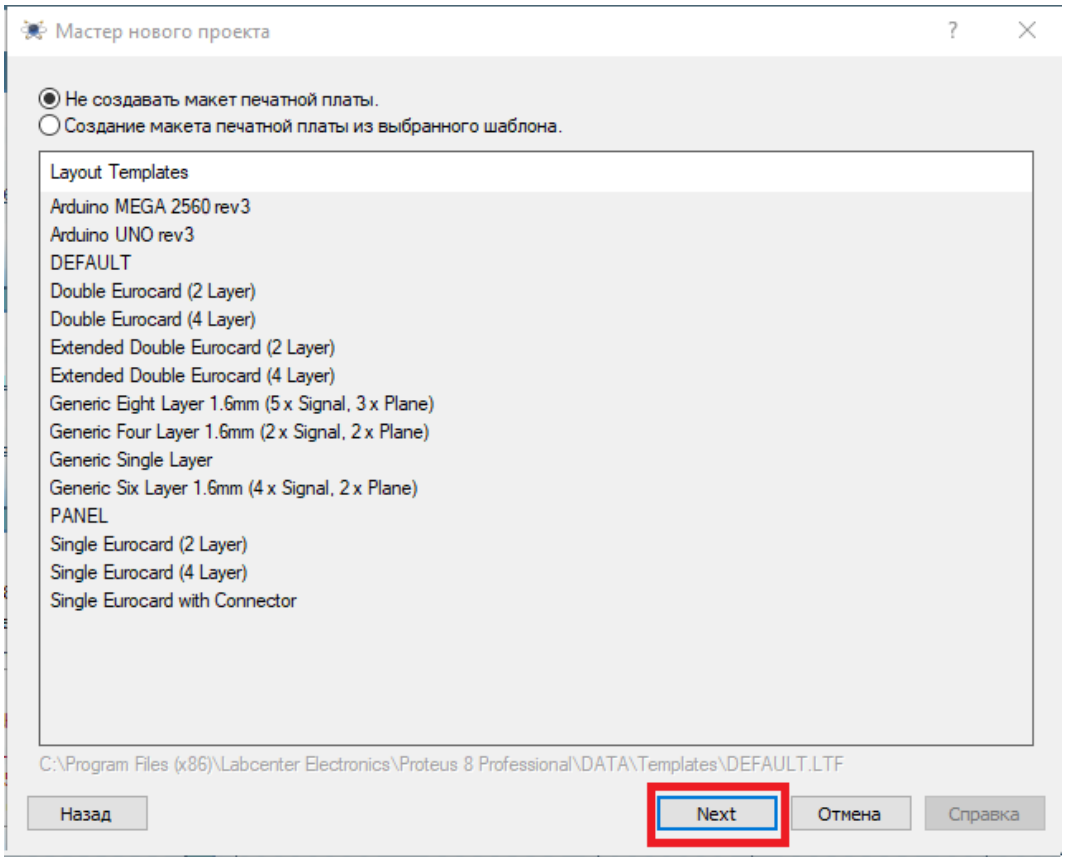
- **GPIO.PWM(channel, frequency)**: Создает объект PWM на указанном пине. **frequency** — частота в герцах.
- **pwm.start(duty_cycle)**: Запускает PWM на пине с заданным скважностью (процент времени, когда сигнал в высоком состоянии).
- **pwm.ChangeDutyCycle(duty_cycle)**: Изменяет скважность сигнала PWM.
- **pwm.ChangeFrequency(frequency)**: Изменяет частоту сигнала PWM.
- **pwm.stop()**: Останавливает PWM.

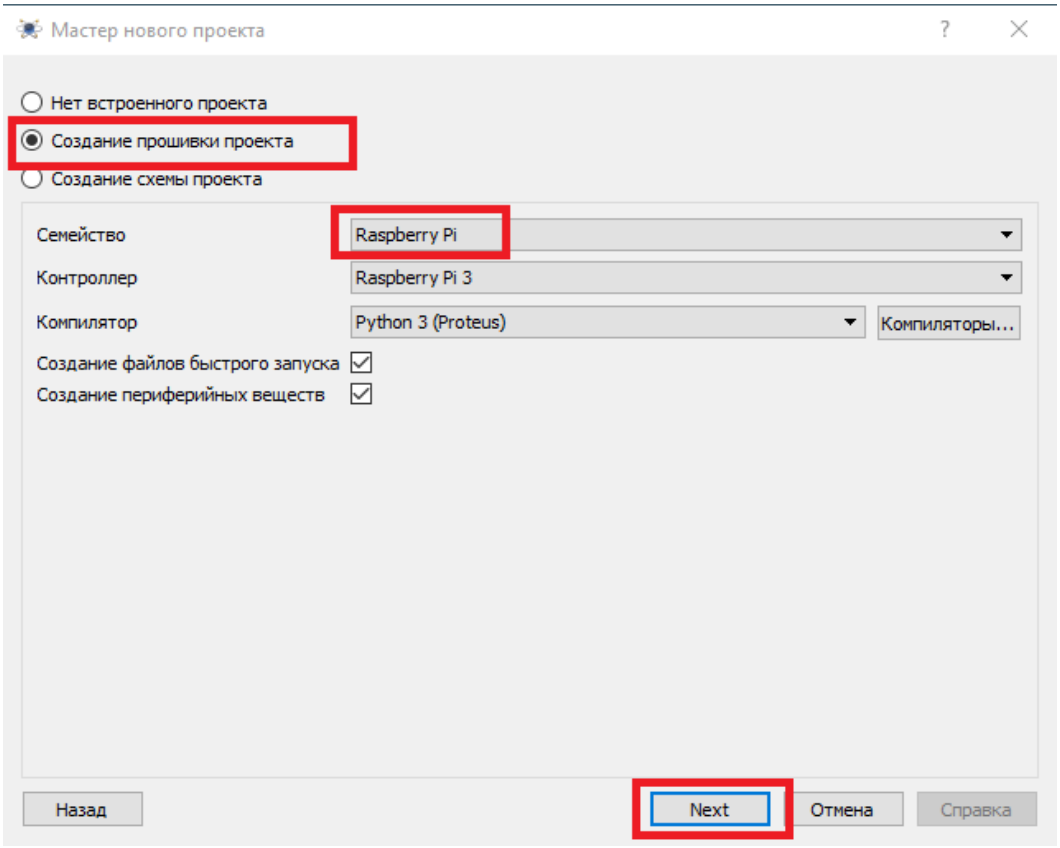
Пример:

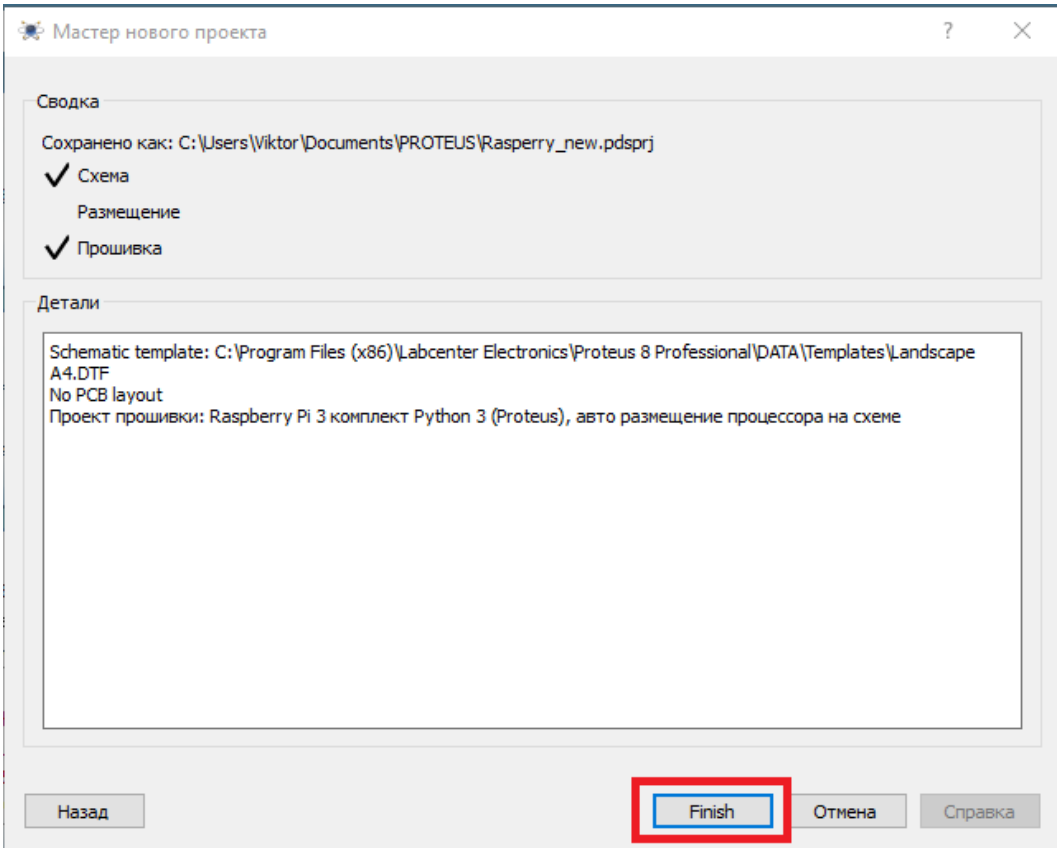
Создадим новый проект в Proteus.



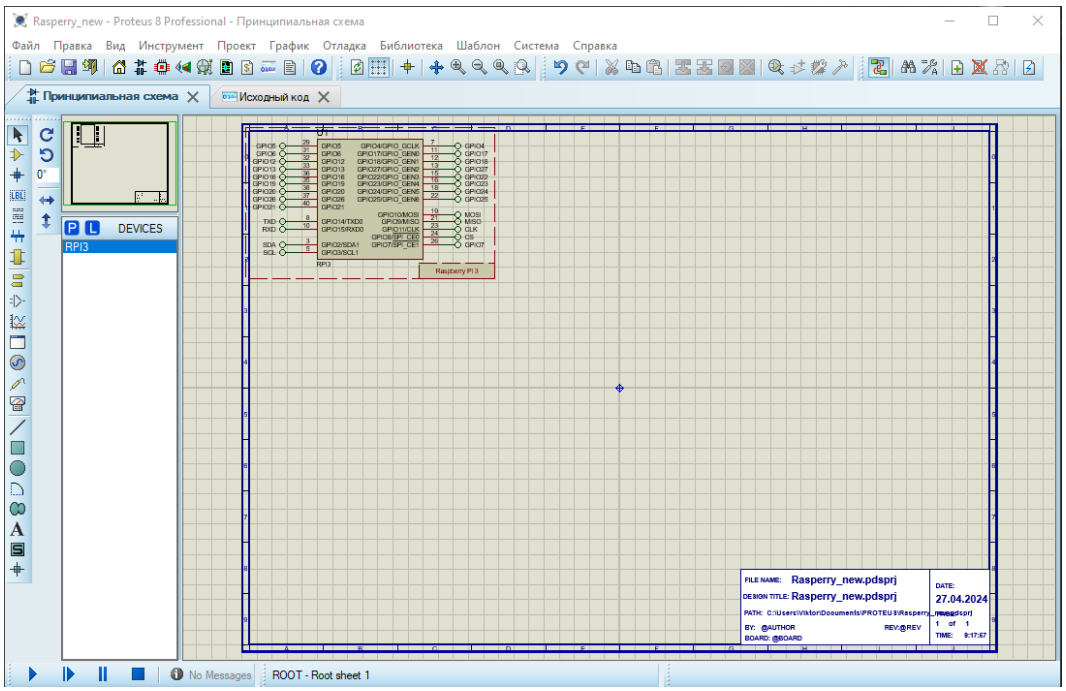

Мастер нового проекта
Не создавать схемы.
Создать схему из выбранного шаблона.
Design Templates
DEFAULT
Landscape A0
Landscape A1
Landscape A2
Landscape A3
Landscape A4
Landscape US A
Landscape US B
Landscape US C
Portrait A0
Portrait A1
Portrait A2
Portrait A3
Portrait A4
Portrait US A
Portrait US B
Portrait US C
Sample Design
C:\Program Files (x86)\Labcenter Electronics\Proteus 8 Professional\DATA\Templates\Landscape A4.DTF
Назад
Next
Отмена
Справка


Мастер нового проекта
Не создавать макет печатной платы.
Создание макета печатной платы из выбранного шаблона.
Layout Templates
Arduino MEGA 2560 rev3
Arduino UNO rev3
DEFAULT
Double Eurocard (2 Layer)
Double Eurocard (4 Layer)
Extended Double Eurocard (2 Layer)
Extended Double Eurocard (4 Layer)
Generic Eight Layer 1.6mm (5 x Signal, 3 x Plane)
Generic Four Layer 1.6mm (2 x Signal, 2 x Plane)
Generic Single Layer
Generic Six Layer 1.6mm (4 x Signal, 2 x Plane)
PANEL
Single Eurocard (2 Layer)
Single Eurocard (4 Layer)
Single Eurocard with Connector
C:\Program Files (x86)\Labcenter Electronics\Proteus 8 Professional\DATA\Templates\DEFAULT.LTF
Назад
Next
Отмена
Справка

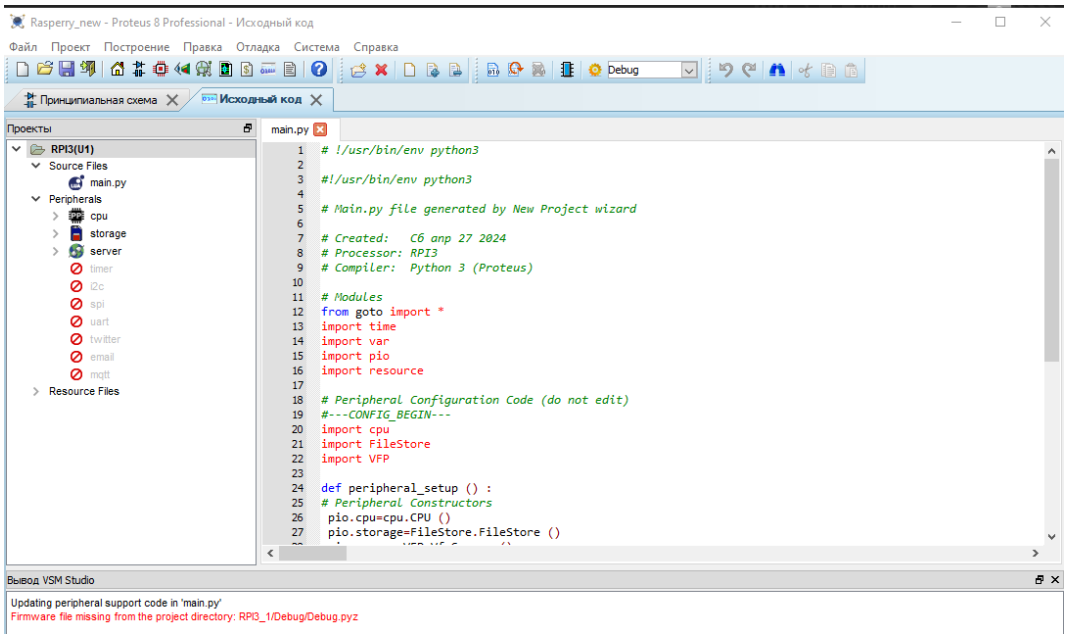




Проект создан, в рабочей области уже расположилась наша плата:

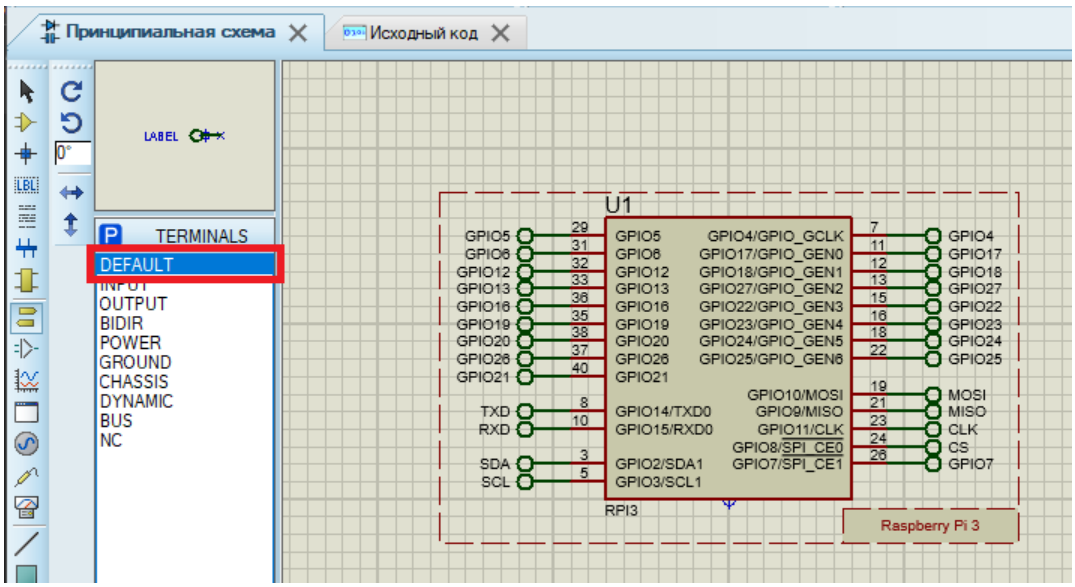


Во вкладке «Исходный код» Исполняемая программа на Python:

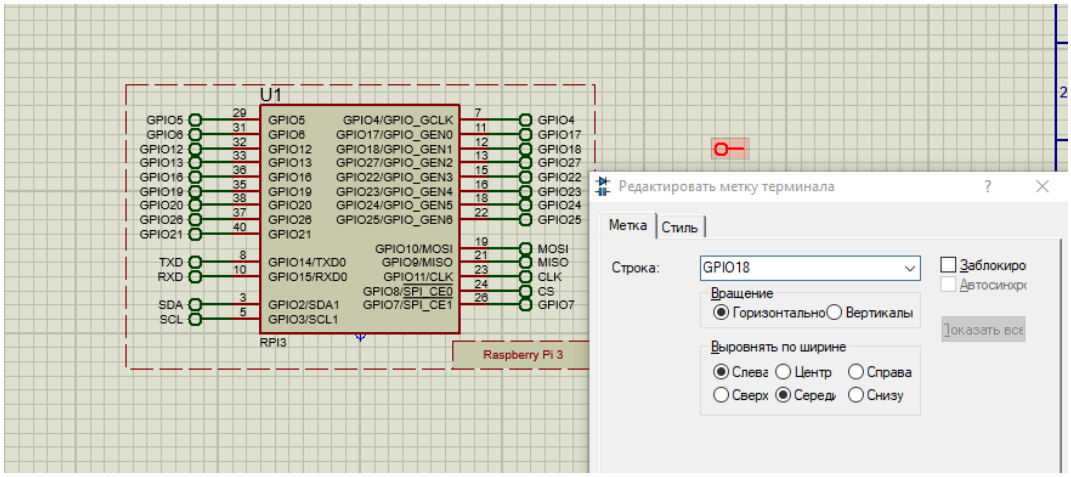


Соберем простую схему:

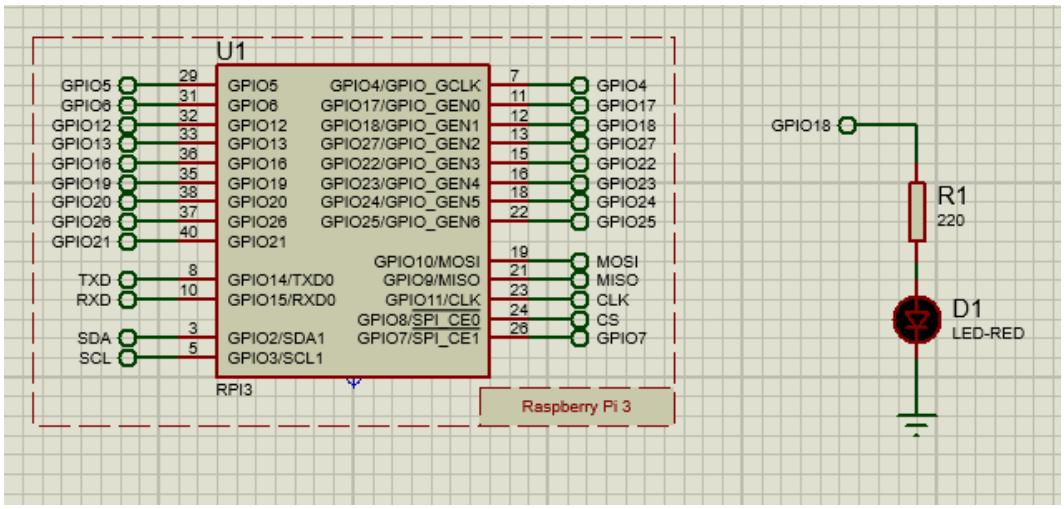
Для подключения к пинам платы используются контакты обозначенные как DEFAULT:



Нужно добавить его в рабочую область, кликнуть и прописать название пина к которому он будет подключен:



Добавим остальные элементы:



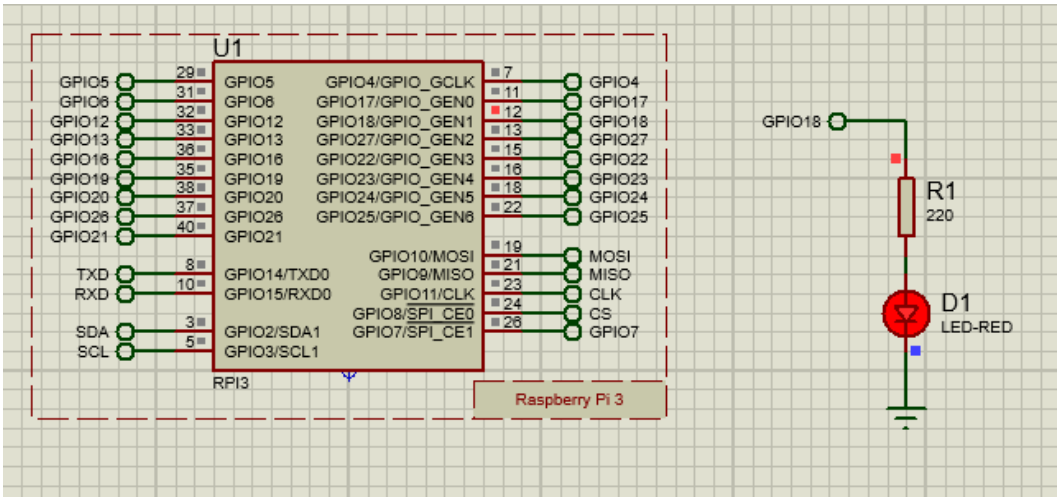
Итак, к GPIO18 подключен светодиод, заставим его мигать. Для этого добавим код:

```
import RPi.GPIO as GPIO
import time
```

```
# Установка схемы нумерации пинов на BOARD
GPIO.setmode(GPIO.BOARD)

# Настройка пина 12 на вывод (физический пин 12 соответствует GPIO18 в режиме BCM)
GPIO.setup(12, GPIO.OUT)

try:
    # Бесконечный цикл для мигания светодиодом
    while True:
        GPIO.output(12, GPIO.HIGH)  # Включение светодиода
        time.sleep(1)               # Пауза в одну секунду
        GPIO.output(12, GPIO.LOW)   # Выключение светодиода
        time.sleep(1)               # Пауза в одну секунду

# этот код выполнится после блока try, независимо от того, произошла ошибка или нет
finally:
    GPIO.cleanup()  # Освобождение всех ресурсов GPIO при завершении работы
```

Основной код программы помещен в конструкцию **try – finaly**, которая обычно используется для гарантии выполнения очистки или завершающих операций. Когда программа завершается, особенно если это происходит из-за ошибки, оставлять GPIO пины в активном состоянии (высоком или низком) нежелательно, т.к. это может привести к непредвиденным результатам, таким как постоянное включение светодиода или активация другого устройства. **GPIO.cleanup()** возвращает все пины в исходное (безопасное) состояние (входы).

Теперь можно запустить симуляцию:

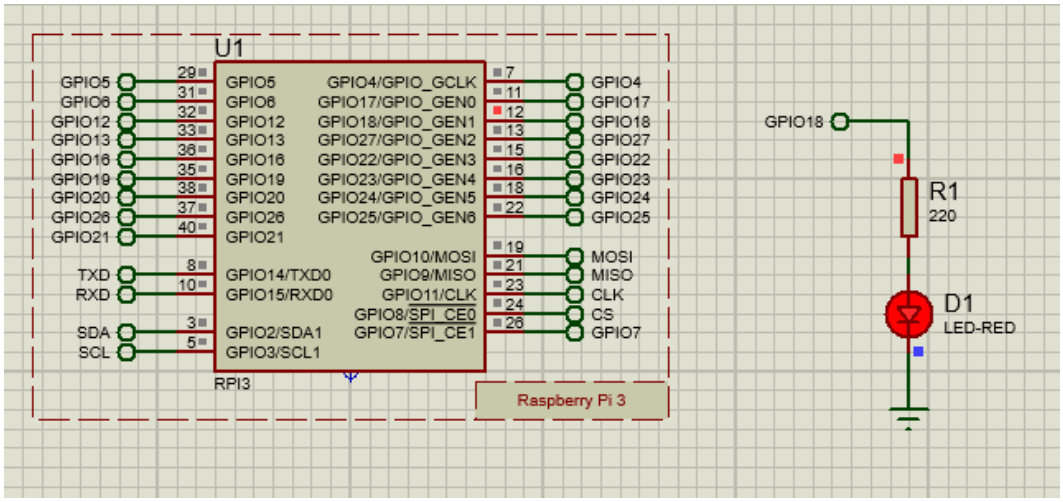


Теперь изменим режим нумерации пинов:

```
import RPi.GPIO as GPIO
import time

# Установка схемы нумерации пинов на BCM
GPIO.setmode(GPIO.BCM)
```

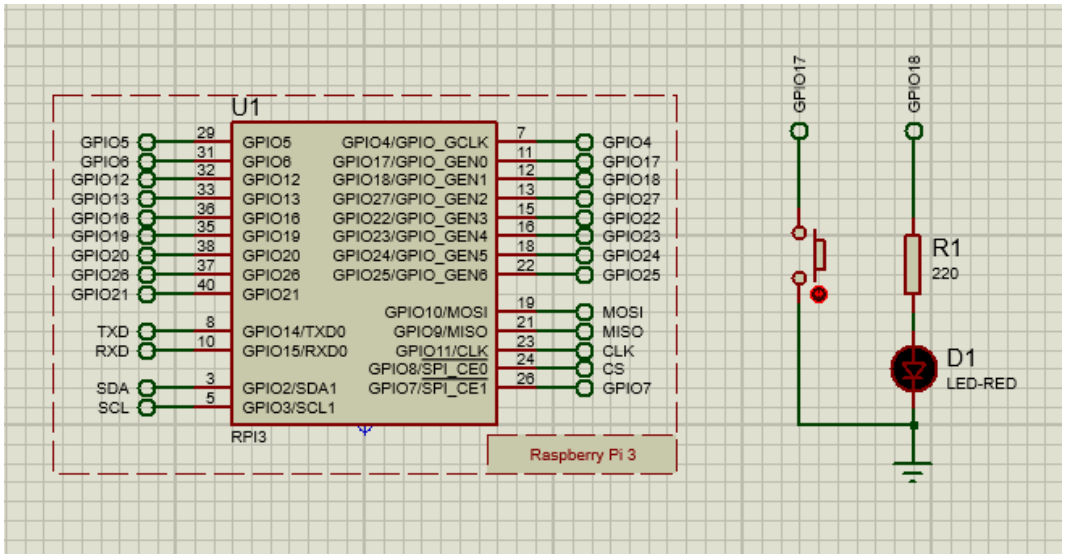
```
# Настройка пина 18 на вывод (пин может использоваться для управления светодиодом)
GPIO.setup(18, GPIO.OUT)

try:
    # Бесконечный цикл для мигания светодиодом
    while True:
        GPIO.output(18, GPIO.HIGH)  # Включение светодиода
        time.sleep(1)               # Пауза в одну секунду
        GPIO.output(18, GPIO.LOW)   # Выключение светодиода
        time.sleep(1)               # Пауза в одну секунду

# этот код выполнится после блока try, независимо от того, произошла ошибка или нет
finally:
    GPIO.cleanup()  # Освобождение всех ресурсов GPIO при завершении работы
```

Результат тот же:



Использование пинов в качестве входа.

Добавим в схему кнопку:



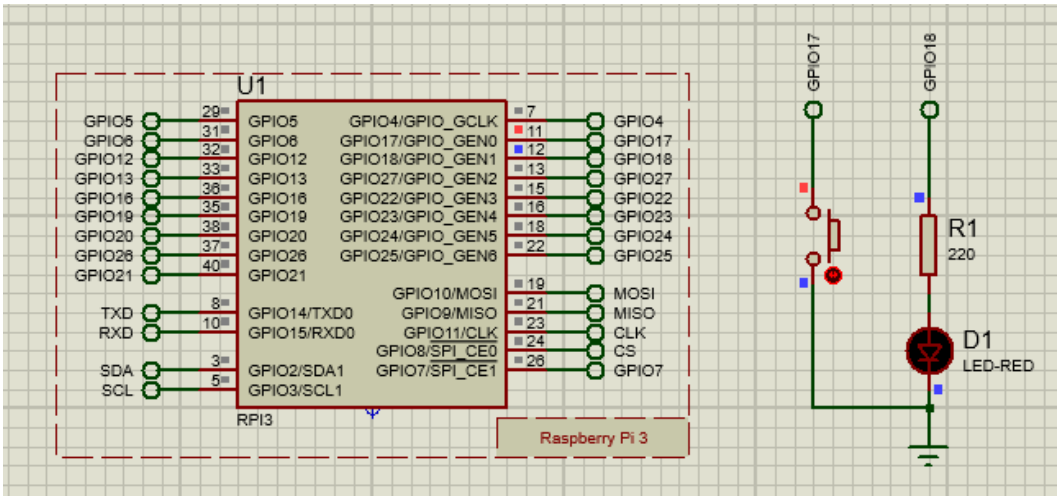
Изменим код таким образом, что бы светодиод загорался только при нажатой кнопке:

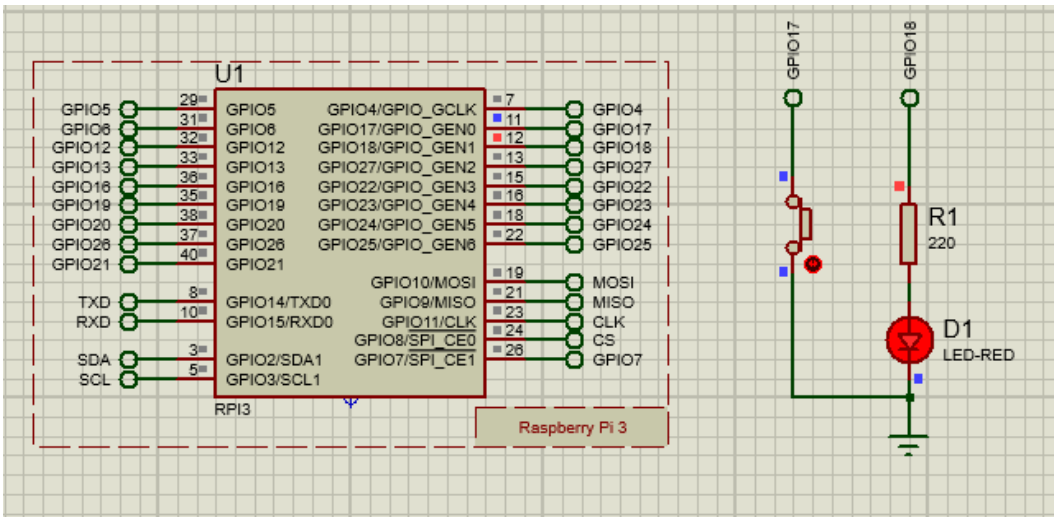
```
import RPi.GPIO as GPIO
import time

# Установка схемы нумерации пинов на BCM
GPIO.setmode(GPIO.BCM)

# Настройка пина 18 на вывод (пин может использоваться для управления светодиодом)
GPIO.setup(18, GPIO.OUT)
# настройки пина 17 в качестве входа
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)

try:
# Бесконечный цикл
    while True:
        button_state = GPIO.input(17)
        time.sleep(0.1)
        if button_state == False:
            GPIO.output(18,GPIO.HIGH)
        else:
            GPIO.output(18,GPIO.LOW)

# этот код выполнится после блока try, независимо от того, произошла ошибка или нет
finally:
    GPIO.cleanup()  # Освобождение всех ресурсов GPIO при завершении работы
```

Конструкция **GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)** используется для настройки GPIO пина на Raspberry Pi в качестве входа с подтяжкой к питанию (pull-up).

- **GPIO.setup(17, GPIO.IN)**: Устанавливает пин 17 в режим входа, что означает, что мы будем читать состояние этого пина (высокий уровень или низкий уровень).
- **pull_up_down=GPIO.PUD_UP**: Включает внутренний подтягивающий резистор на пине. В данном случае, используется подтяжка к питанию (pull-up). Это означает, что в состоянии покоя (когда нажатия на кнопку нет), входной пин будет подтянут к напряжению питания (обычно 3.3V на Raspberry Pi). При нажатии на кнопку, пин будет подключен к земле, что приведет к низкому уровню на входе. Подтяжка к питанию помогает предотвратить появление плавающего состояния на входе и обеспечивает стабильное чтение состояния пина.

Проверка в сумуляторе:





Задание:

1. Создайте схему из Raspberry Pi и 3-х светодиодов. Напишите код, который будет зажигать светодиоды по очереди (эффект бегущего огня).

2. Добавьте в схему кнопку. При нажатии кнопки скорость переключения кнопок должна увеличиваться.