Министерство образования Республики Беларусь
Учреждение образования «Витебский государственный
университет имени П.М. Машерова»
Кафедра инженерной физики

МИКРОПРОЦЕССОРЫ И АППАРАТНЫЕ СРЕДСТВА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Методические рекомендации по выполнению лабораторных работ

Витебск
ВГУ имени П.М. Машерова
2021

УДК 621.382.2/.3(075.8)
ББК 32.97я73
М73

Печатается по решению научно-методического совета учреждения образования «Витебский государственный университет имени П.М. Машерова». Протокол № от .2021.

Составители: старшие преподаватели кафедры инженерной физики ВГУ имени П.М. Машерова **Ю.В. Шиёнок**, **Т.И. Сапелко**

Р е ц е н з е н т :
заведующий кафедрой прикладного и системного программирования
ВГУ имени П.М. Машерова,
кандидат физико-математических наук, доцент *С.А. Ермоченко*

М73 **Микропроцессоры и аппаратные средства вычислительной техники :** методические рекомендации по выполнению лабораторных работ / сост.: Ю.В. Шиёнок, Т.И. Сапелко. – Витебск : ВГУ имени П.М. Машерова, 2021. – 42 с.

Методические рекомендации написаны в соответствии с учебной программой для специальности 1-98 01 01-02 «Компьютерная безопасность». Содержит теорию, методику выполнения, контрольные вопросы защиты лабораторных работ и список литературы. Учебное издание предназначено для изучения теоретического материала и отработки практических навыков при изучении дисциплины «Микропроцессоры и аппаратные средства вычислительной техники».

УДК 621.382.2/.3(075.8)
ББК 53.1я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящие методические указания предназначены для получения практических навыков студентами по профилю подготовки - «Компьютерная безопасность» при изучении дисциплины «Микропроцессоры и аппаратные средства вычислительной техники».

Методические указания содержат 5 работ. Предлагаемые задания охватывают основные разделы рабочей программы, связанные с изучением структуры и функционирования микропроцессора.

Общие методические рекомендации по использованию лабораторных работ и методических указаний:

– к выполнению лабораторной работы следует приступать после ознакомления с теоретической частью соответствующего раздела и рекомендациями, приведенными в конкретной работе;

- лабораторные работы рекомендуется выполнять в порядке их нумерации;

- рекомендуется для экономии времени отчеты о лабораторных работах оформлять в виде протоколов работы с обязательным указанием номера, темы, цели работы и выводов с краткой характеристикой результата;

- дополнительные сведения по лабораторным работам содержатся в прилагаемом списке литературы.

Методические указания рекомендовано студентам для аудиторного и самостоятельного освоения лабораторной части дисциплины «Микропроцессоры и аппаратные средства вычислительной техники».

ЛАБОРАТОРНАЯ РАБОТА № 1
Знакомство с Raspberry Pi

Цель работы: познакомить студентов с Raspberry Pi.
Оборудование: одноплатный компьютер Raspberry Pi

Теория работы

Знакомство с Raspberry Pi

Raspberry Pi несет на борту интерфейс, где реализованы интерфейсы UART, консольный порт, SPI (Serial Peripheral Interface, последовательный периферийный интерфейс) и PC (Inter Integrated Circuit, последовательная шина данных для связи интегральных схем).

Расстояние между выводами – 2,54 мм. Выводы UART, SPI и IC в случае необходимости могут быть настроены как обычные порты ввода/вывода.

	3.3 VDC Power	1	2	5.0 VDC Power			
8	GPIO 8 SDA1 (I2C)	3	4	5.0 VDC Power			
9	GPIO 9 SCL1 (I2C)	5	6	Ground			
7	GPIO 7 GPCLK0	7	8	GPIO 15 TxD (UART)	15		
	Ground	9	10	GPIO 16 RxD (UART)	16		
0	GPIO 0	11	12	GPIO 1 PCM_CLK/PWM0	1		
2	GPIO 2	13	14	Ground			
3	GPIO 3	15	16	GPIO 4	4		
	3.3 VDC Power	17	18	GPIO 5	5		
12	GPIO 12 MOSI (SPI)	19	20	Ground			
13	GPIO 13 MISO (SPI)	21	22	GPIO 6	6		
14	GPIO 14 SCLK (SPI)	23	24	GPIO 10 CE0 (SPI)	10		
	Ground	25	26	GPIO 11 CE1 (SPI)	11		
30	SDA0 (I2C ID EEPROM)	27	28	SCL0 (I2C ID EEPROM)	31		
21	GPIO 21 GPCLK1	29	30	Ground			
22	GPIO 22 GPCLK2	31	32	GPIO 26 PWM0	26		
23	GPIO 23 PWM1	33	34	Ground			
24	GPIO 24 PCM_FS/PWM1	35	36	GPIO 27	27		
25	GPIO 25	37	38	GPIO 28 PCM_DIN	28		
	Ground	39	40	GPIO 29 PCM_DOUT	29		

Рисунок 1.1 Назначение выводов GPIO

Назначение выводов GPIO (так называемая распиновка) представлено на рисунке 1.1.

Для плат «A» и «B» первой ревизии выводы 4, 9, 14, 17, 20, 25 обозначены как DNC (Do Not Connect), и подсоединять к ним что-либо не следует - плата может сгореть. На новых ревизиях платы разведены, но не распаяны, еще четыре GPIO, дополнительно дающие интерфейсы PC и PS (Inter-Integrated Circuit, последовательная шина данных для связи интегральных схем). Первые 26 выводов GPIO платы «B+» полностью идентичны разъемам GPIO плат «A» и «B», остальные 14 содержат 9 выводов GPIO общего назначения, которые могут быть сконфигурированы как вход или выход, 3 вывода GND и 2 вывода не задействованы.

С помощью выводов GPIO к Raspberry Pi можно подключать датчики и внешние исполнительные устройства, что открывает интересные возможности творческого использования Raspberry Pi.

При работе с портами GPIO следует помнить о некоторых их особенностях и соблюдать определенные меры предосторожности, чтобы не повредить Raspberry Pi.

Вот основные из них:

- максимальный суммарный ток обоих выводов 3,3 В равен 50 мА и эти выводы могут использоваться для питания внешних устройств только в том случае, если их потребляемый ток меньше 50 мА;

- максимальный суммарный ток обоих выводов 5 В равен 300 мА, и эти выводы также могут использоваться для питания внешних устройств только в том случае, если их потребляемый ток меньше 300 мА;

- на GPIO нельзя подавать напряжение больше 3,3 В! Цифровые выводы GPIO имеют уровни напряжения 0-3,3 В и не совместимы с традиционными уровнями напряжения 0-5 В. Если подать на вывод GPIO логическую единицу, представляющую собой 5 В (а не·3,3 В), -этот вывод может выйти из строя;

- выводы GPIO 14 и GPIO 15 по умолчанию выполняют альтернативную функцию и являются выводами UART (RXD и TXD), поэтому после включения на них присутствует высокий уровень 3,3 В, однако программно их можно переконфигурировать в обычные выводы. Все остальные выводы GPIO после включения Raspberry Pi выполняют основную функцию и работают как обычные цифровые;

- все настраиваемые выводы GPIO - кроме GPIO 0 (SDA) и GPIO 1 (SCL) - по умолчанию являются входами и поэтому имеют высокое входное сопротивление, при этом подтяжка логического уровня у них не включена, так что после включения Raspberry Pi напряжение на них может «плавать»;

- выводы GPIO 0 (SDA) и GPIO 1 (SCL) по умолчанию «подтянуты» питанию, поэтому после включения Raspberry Pi на них присутствует напряжение логической единицы (3,3 В);

- сигнал на любом из цифровых выводов может служить источником внешнего прерывания.

Нужно помнить, что GPIO - это выводы, непосредственно подключенные к процессору Raspberry Pi, они являются инструментом для взаимодействия с ним.

Поэтому неосторожное обращение с GPIO может привести к необратимым последствиям для процессора. ОС Raspberry представляет собой один из

дистрибутивов Linux, а концепция Linux предполагает, что любой объект является файлом. Именно это позволяет выводить и считывать сигналы с GPIO обычными командами оболочки bash прямо в терминале. Вывод логической единицы при этом выглядит как команда записи «1» в файл, соответствующий нужному выводу:

```
sudo su -
echo"25" > /sys/class/gpio/export
echo"25" > /sys/class/gpio/export
echo"out" > /sys/class/gpio/gpio25/direction
echo"1" > /sys/class/gpio/gpio25/value
echo"0" > /sys/class/gpio/gpio25/value
```

Для чтения входов надо использовать команду cat и путь к файлу:

```
echo "24" > /sys/class/gpio/export
echo "in" > /sys/class/gpio/gpioO/direction
cat /sys/class/gpio/gpio24/value
```

Все операции должны выполняться от имени пользователя root.

Управление GPIO командами языка Python.

Для работы с GPIO на языке Python требуется специальная библиотека RPi.GPIO.

В новом дистрибутиве Raspberry она уже установлена, а если у вас дистрибутив старый, то для установки библиотеки RPi.GPIO выполните команду:

```
sudo apt-get install python-rpi.gpio
```

Чтобы использовать эту библиотеку, необходимо в програм.му на Python добавить строку импорта библиотеки RPi.GPIO:

```
Import RPi.GPIO as GPIO
```

При написании программы можно выбрать один из двух способов нумерации портов GPIO: первый - GPIO.BOARD - использует систему нумерации портов на плате Raspberry Pi. Преимущество этой системы нумерации в том, что ваше оборудование будет работать всегда, независимо от номера ревизии - вам не придется перемонтировать свой разъем или изменить имеющийся код. Вторая система нумерации - GPIO.BCM (номера BCM). Это более низкий уровень работы - с прямым обращением к номерам каналов на процессоре Broadcom. Выбор способа нумерации определяется соответствующими командами:

```
GPIO.setmode(GPIO.BOARD)
GPIO.setmode(GPIO.BCM)
```

Следующие команды устанавливают режим работы контакта на вход или выход:

```
GPIO.setup(channel, GPIO.IN)
GPIO.setup(channel, GPIO.OUT)
```

Если входной канал ни к чему не подключен, его значение может «плыть». Следующие команды устанавливают начальную «подтяжку» вывода к питанию или к «земле»:

```
GPIO.setup(channel, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(channel, GPIO.IN, GPIO.PUD_DOWN)
```

Для выходов out можно установить начальное значение 0 или 1:

```
GPIO.setup(channel, GPIO.OUT, GPIO.LOW)
GPIO.setup(channel, GPIO.OUT, GPIO.HIGH)
```

Для чтения значения контакта GPIO, настроенного как вход IN, служит следующая команда:

```
GPIO.input(channel)
```

Значение контакта, настроенного как выход out, устанавливается следующей командой:

```
GPIO.output(channel, state)
```

```
irnport RPi.GPIO as GPIO#подключаем библиотеку
GPIO.setmode(GPIO.BCM)#устанавливаем режим нумерации
GPIO.setup(7, GPIO.OUT)#конфигурируем GPIO 7 как выход
GPIO.setup(8, GPIO.IN)#конфигурируем GPIO 8 как вход
GPIO.output(7, True)#выводим на GPIO 7 логическую "1" (3.3 V)
GPIO.output(7, False)#выводим 'на GPIO 7 логический "0"
signal = GPIO.input(8)#считываем с GPIO 8 в переменную signal
GPIO. cleanup ( )#завершаем работу с GPIO
```

Библиотека RPi.GPIO позволяет использовать контакты GPIO в качестве выходов ШИМ (сигналов широтно-импульсной модуляции).

Для создания экземпляра ШИМ служит команда:

```
p = GPIO.PWM(channel, frequency)
```

где frequency – частота, Гц.

Для старта ШИМ на контакте:

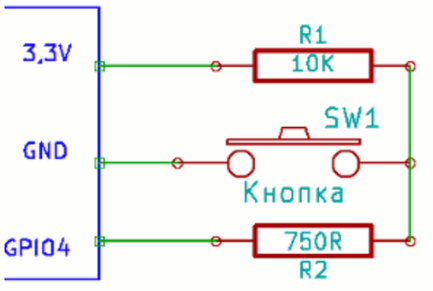
p. start (dc)

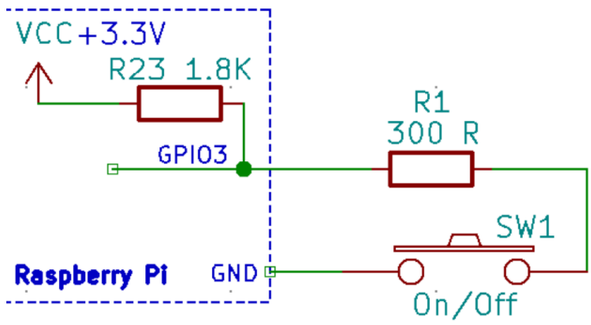
где dc – рабочий цикл ШИМ (0,0-100,0).

Описание работы

1.С помощью команды gpio readall определите режимы работы и уровни на выходах GPIO.

2. Собрать схему 1 и схему 2.





3. Написать программу для считывания нажатия кнопки по первой и второй схеме.

4. Подключить к Raspberry светодиод.

Рассчитать величину гасящего сопротивления.

К примеру, светодиод, рассчитанный на напряжение 2В, красного цвета и для него гасящий резистор при токе 10мА:

R = (3,3В - 2В) / 0.01А = 130 Ом.

5. Написать программу, которая при нажатии кнопки будет изменять частоту моргания светодиода в два раза.

6. Изучить модуль RPIO.

RPIO - это модуль с дополнительными возможностями для работы с GPIO Raspberry Pi. Он содержит весь функционал пакета GPIO(RPi)

rpio-curses - удобный консольный графический интерфейс для мониторинга и установки состояний пинов;

rpio - консольная утилита для установки состояний пинов, мониторинга состояний пинов и их режимов, а также для получения другой полезной информации.

7. Подключите к Respberry восьмисегментный индикатор. Напишите программу для подсчета числа нажатий на кнопку.

8. Подключите к Respberry 4-хразрядный индикатор. Напишите программу секундомер.

ЛАБОРАТОРНАЯ РАБОТА № 2
Управление светодиодами на макетной плате Амперка

Цель работы: научиться управлять светодиодами на макетной плате Амперка

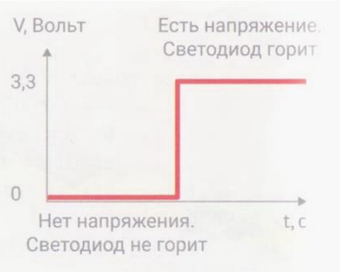
Оборудование: одноплатный компьютер Raspberry Pi

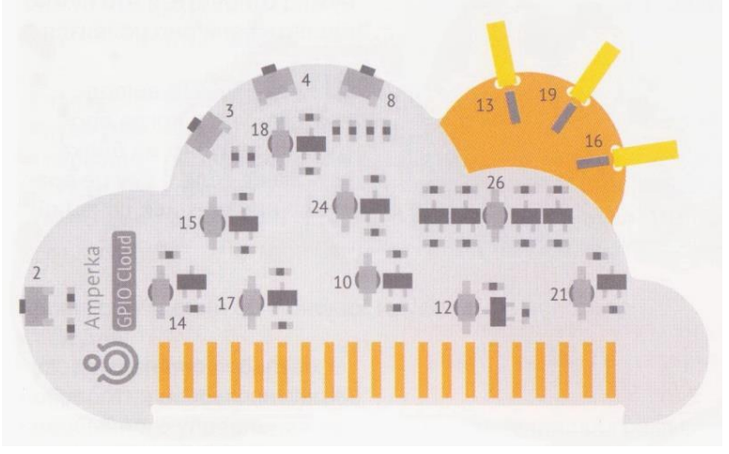
Описание работы

Лампа

Raspberry Pi работает не только как обычный компьютер, но и управляет электронными устройствами, используя пины GPIO (Generall Purpose Input Output – контакты ввода/вывода общего назначения).

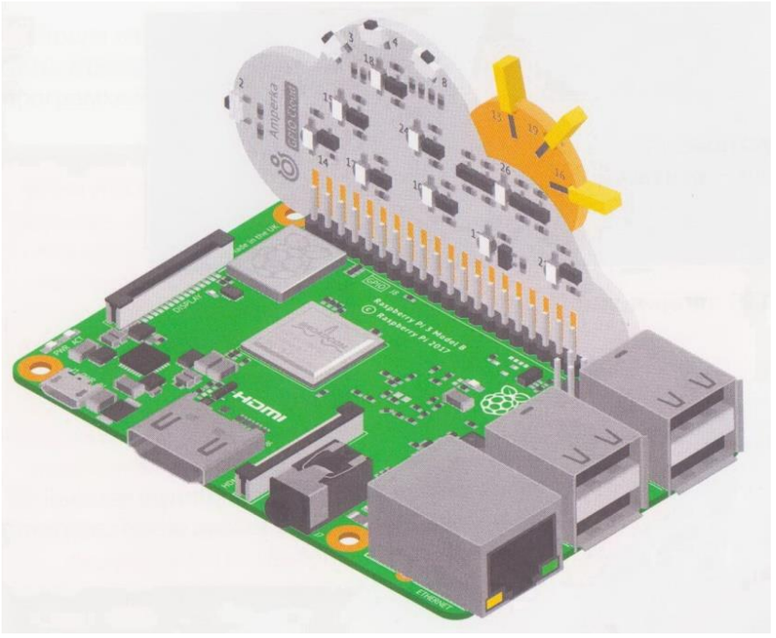
Когда Raspberry задаёт на пине логическую единицу, на нём появляется напряжение (сигнал). Если к пину подсоединен светодиод, он включится. Когда Raspberry задаст логический ноль – выключится. Так работает пин в режиме OUT (цифровой выход).





Номера светодиодов и кнопок указаны рядом с ними.

GPIO Cloud-плата с двенадцатью светодиодами и четырьмя кнопками для управления программами, которых нет на Raspberry Pl. Облако поможет разобраться с управлением пинами GPIO.



ВНИМАНИЕ!
Перед тем как подключить GPIO Cloud, обязательно выключи Raspberry и отключи кабель питания.

1.Подключи GPIO Cloud к Raspberry Pl и снова подай питание.

2.Открой терминал и введи команду python3. Набери строчки:

```
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BCM)
>>> GPIO.setup(21, GPIO.OUT)
>>> GPIO.output(21, GPIO.HIGH)
```

1. Импортируем модуль (набор специфических функций) RPi. GPIO, а обращаться к нему будем чуть короче – просто GPIO.

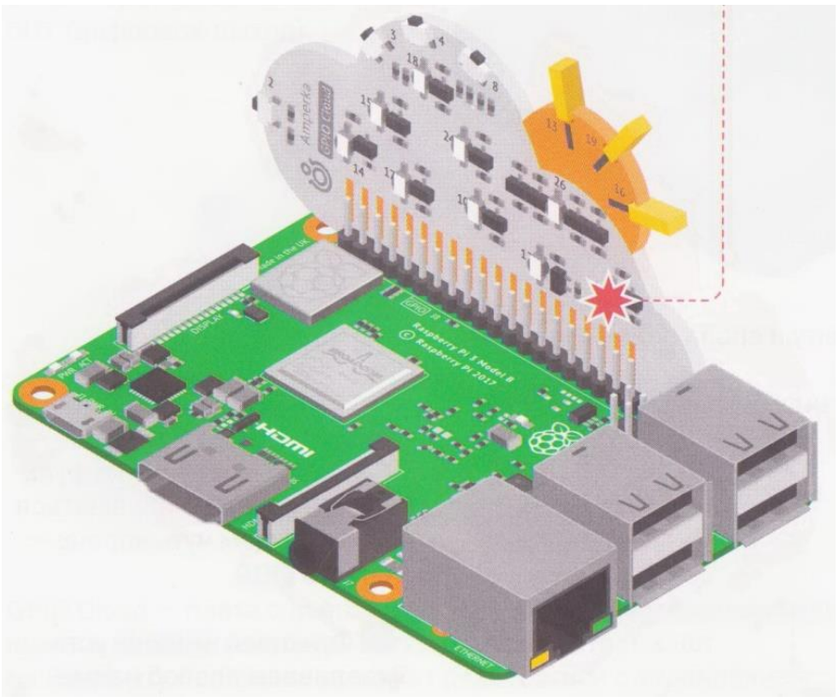
2. Функцией setmode устанавливаем способ наименования пинов. BCM значит «так же, как пины названы на чипе».

3.функция setup задаёт режим работы пина (вход или выход). Пин 21 переводим в режиме выхода.

4.на выводе 21 устанавливаем высокий логический уровень (подаёт напряжение).

```
pi@raspberrypi:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BCM)
>>> GPIO.setup(21, GPIO.OUT)
>>> GPIO.output(21, GPIO.HIGH)
>>>
```

В конце нажми Enter. Светодиод 21 загорится.



ВНИМАНИЕ!
Если в процессе работы увидишь сообщения «Traceback... » или «Error...», значит, где-то в коде есть опечатка. Внимательно всё проверь и просто набери строчку заново.

ТРЮК
В интерпретаторе тоже можно повторять введенные команды стрелками вверх и вниз.

3.Теперь строчкой ниже добавь GPIO.output(21, GPIO.LOW).
На пине 21 установится низкий логический уровень (GPIO. LOW), и светодиод погаснет.

4.После экспериментов со светодиода и вызови функцию GPIO.cleanup(), чтобы освободить все пины от управления программой.

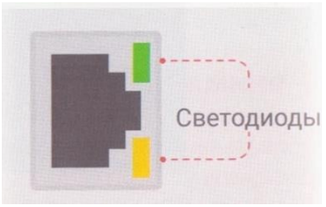
ВНИМАНИЕ!
В конце программ устанавливающих GPIO в режим выхода обязательно выполняй функцию GPIO.cleanup()! Иначе ты рискуешь устроить короткое замыкание, от которого твоя Raspberry обязательно выйдет из строя. В случае, если ты забыл выполнить эту функцию, при старте следующей программы, с использованием GPIO ты увидишь предупреждение о том, что пины всё ещё активны.

5.Вызови функцию quit() или нажми клавиши ctrl+D на клавиатуре, чтобы выйти в интерфейс терминала

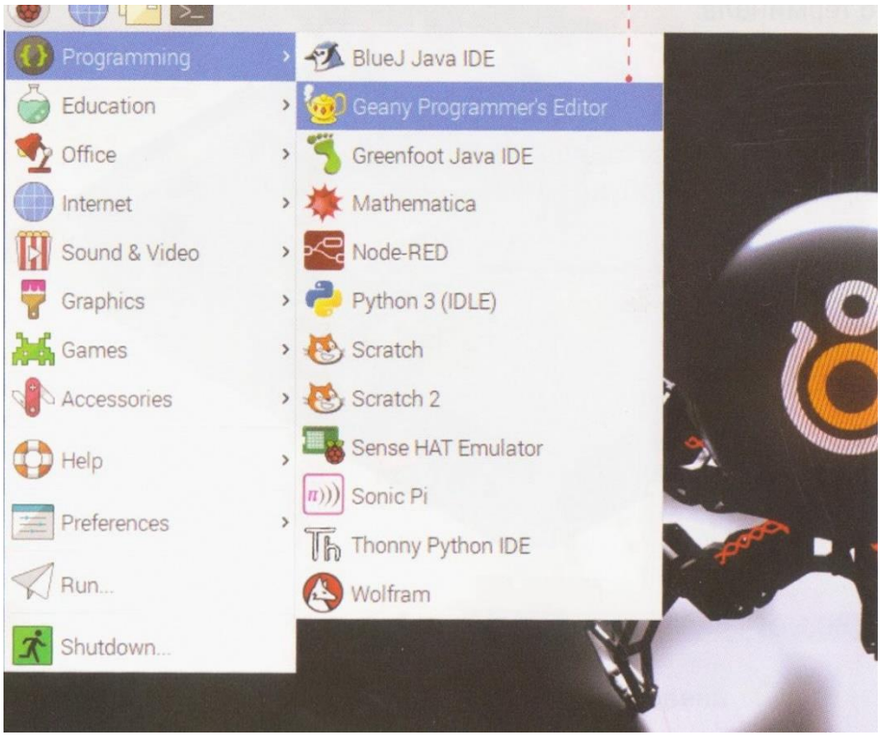
ЗАДАНИЕ
Попробуй включить и выключить остальные светодиоды. Их номера подписаны на плате GPIO Cloud. Не забудь настроить пины на выход.

МАЯЧОК

Светодиоды служат не только для освещения, но и для индикации. Например: в разъёме Enternet один светодиод горит постоянно, сообщая о работе сети. Другой - мигает. Период мигания кодирует скорость передачи данных.
Научись мигать светодиодами на плате GPIO Cloud для индикации состояний в своих проектах.

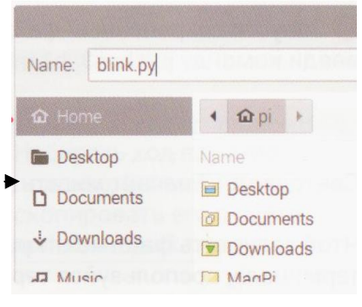


1. Код будем писать в файле. Так гораздо практичнее: он не пропадает после завершения. Открой редактор «Programmer's Editor» (Menu → Programming → Geany Programmer's Editor).



2. В строке меню выбери File → Save As…

3. Введи имя файла «blink.py». Файл сохранится в директории «pi».



4. Сохрани файл, нажав кнопку Save.

5. В окне редактора набери код:

```
1  import RPi.GPIO as GPIO
2  import time
3
4  GPIO.setmode(GPIO.BCM)
5  GPIO.setup(17, GPIO.OUT)
6
7  try:
8      while True:
9          time.sleep(0.5)
10         GPIO.output(17, GPIO.HIGH)
11         time.sleep(0.5)
12         GPIO.output(17, GPIO.LOW)
13 except KeyboardInterrupt:
14     print('The program was stopped by keyboard.')
15 finally:
16     GPIO.cleanup()
17     print('GPIO cleanup completed.')
```

1.Импортируем модуль time для функции задержки время.

2. Константа true всегда истинна, поэтому цикл будет выполняться вечно.

3.Вызываем функцию sleep из модуля time. Параметр 0.5 задает задержку в полсекунды.

4. Отлавливаем исключение сообщающее о прерывании программы клавишами ctrl+C.

5. Поместив внутрь блока finally функцию GPIO. cleanup(), можно быть уверенным, что пины Raspberry сбросятся в начальное состояние.

6.Сохрани изменения в файле нажав на клавиатуре комбинацию клавиш ctrl+S.

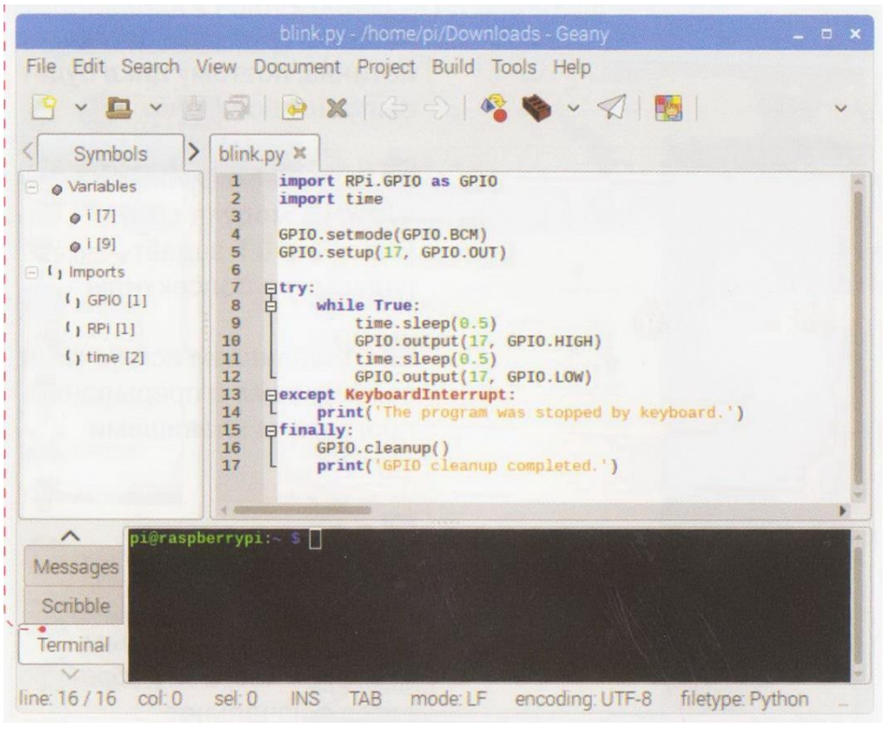
7.Запусти сохраненный файл. Для этого открой терминал, введи команду pyhton3 blink.py и нажми Enter.

```
pi@raspberrypi:~ $ python3 blink.py
```

Светодиод 17 начнёт мигать раз в секунду.

Чтобы запустить файл не открывая дополнительное окно терминала, воспользуйся терминалом встроенным в Geany Programmer’s Editor.

8.Перейди во вкладку Terminal, введи команду запуска файла и нажми Enter.



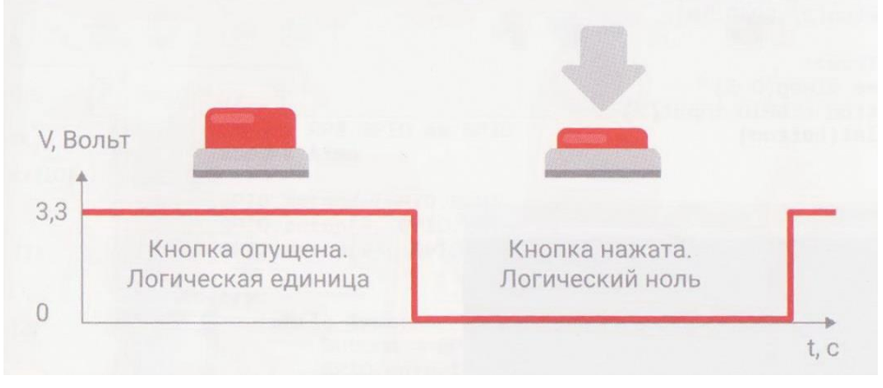
9.Чтобы прервать цикл while и завершить выполнение программы, нажми клавишу ctrl+C. В ответ получишь сообщение из блоков expect и finally.

ЗАДАНИЕ
Измени блок кода с циклом while, сократив период мигания в пять раз.

ВЫКЛЮЧАТЕЛЬ

Взаимодействие человека с устройством не полно без органов управления. Добавим кнопку. Чтобы понять, нажата кнопка или нет, Raspberry считывает состояние пина, настроенного на вход (In).
Пока кнопка отпущена , на пин приходит логическая единица. Если нажать кнопку – на пин придёт логический ноль.

ТРЮК
Набирать код вручную полезно, но можно скопировать его с сайта malina.amperka.ru



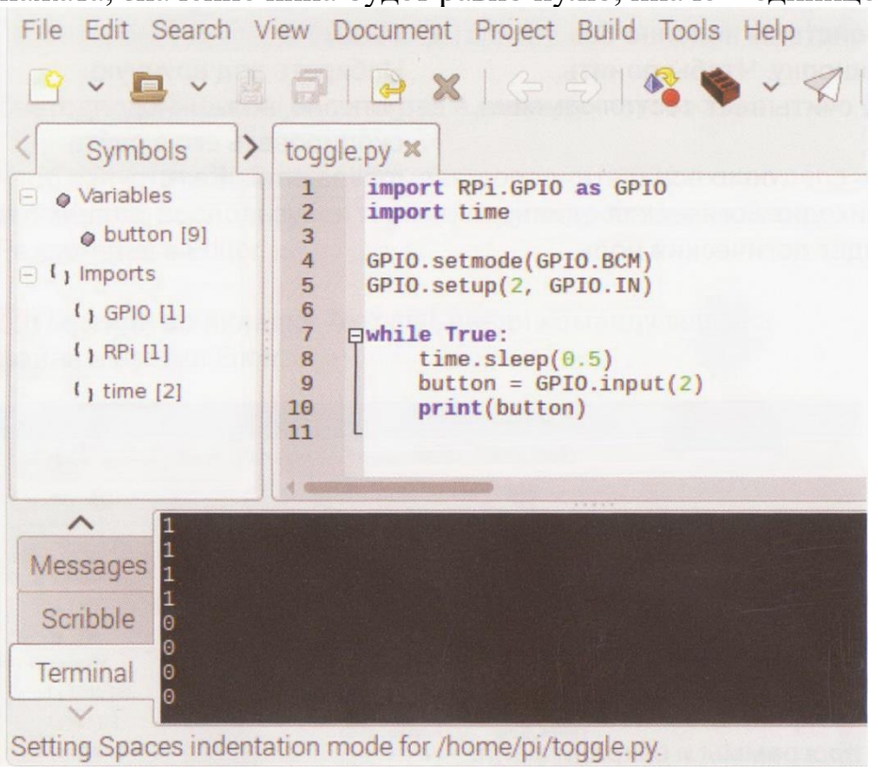
1. Напиши программу для ввода состояния пина. Создай новый файл (File→New) и сохрани его в папке «pi» под именем toggle.py. Вставь код программы и сохрани файл.

```
1  import RPi.GPIO as GPIO
2  import time
3
4  GPIO.setmode(GPIO.BCM)
5  GPIO.setup(2, GPIO.IN)  •----------------------------------
6
7  while True:
8      time.sleep(0.5)
9      button = GPIO.input(2)  •-------------------------------
10     print(button)
```

1. настраиваем пин 2 на вход.

2. Функцией input считываем значение пина и записываем его в переменную button.

2.Запусти файл командой python3 toggle.py. Когда кнопка нажата, значение пина будет равно нулю, иначе – единице.



3.Останови выполнение программы (ctrl+C)

4.Передай переменную button в функцию output, чтобы управлять светодиодом. Состояние пина 2 будет сразу передаваться на пин 24. При каждом нажатии кнопки светодиод будет гаснуть.

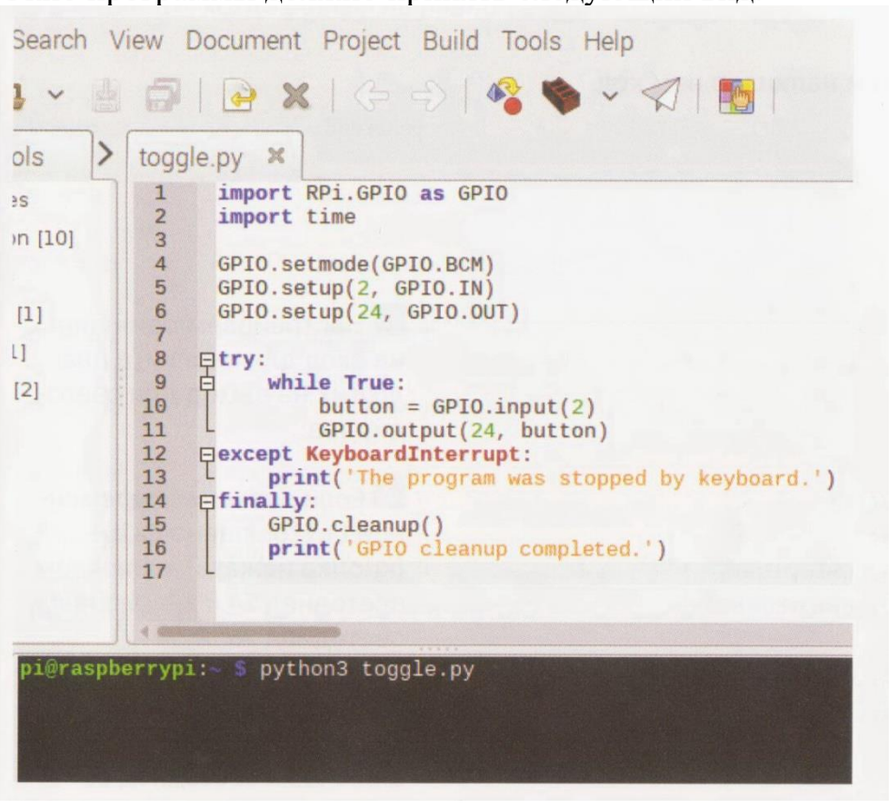
Кнопка	button	Светодиод
нажата	0	не горит
не нажата	1	горит

```
7  while True:
8      button = GPIO.input(2)
9      GPIO.output(24, button)
```

Не забудь добавить строку инициализации пина на выход.

```
6   GPIO.setup(24, GPIO.OUT)
```

Сделай обработку исключений, чтобы сбросить пин 24 в исходное состояние, когда завершишь работу программы. Окно программы должно принять следующий вид:



5.Сохрани файл (ctrl+S) и запусти программу командой python3 toggle.py.

Состояние пина 2 будет сразу передаваться на пин 24. При каждом нажатии кнопки светодиод будет гаснуть.

ЗАДАНИЕ
Пусть светодиод загорится вместе с кнопкой.

ПЕРЕКЛЮЧАТЕЛЬ

Обычно устройство проектируют по принципу «если произошло событие А, сделать сделать действие Б, иначе сделать В» Конструкция «если-то-иначе» называется условиным выражением или ветвлением.
В языке Python условные выражения записываются с помощью операторов if (если) и esle (иначе):

If (условие А):
  действие Б
Esle:
  действие В

Попробуй сделать простую программу с таким выражением: «если кнопка 8 нажата, свети светодиодом 24, иначе свети светодиодом 26»

1. Создай новый файл switch.py и напиши в нем код программы:

```
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
GPIO.setup(8, GPIO.IN)
GPIO.setup(24, GPIO.OUT)
GPIO.setup(26, GPIO.OUT)

try:
    while True:
        button = GPIO.input(8)
        if button == False:
            GPIO.output(24, GPIO.HIGH)
            GPIO.output(26, GPIO.LOW)
        else:
            GPIO.output(24, GPIO.LOW)
            GPIO.output(26, GPIO.HIGH)

except KeyboardInterrupt:
    print('The program was stopped by keyboard.')
finally:
    GPIO.cleanup()
    print('GPIO cleanup completed.')
```

1. Настраиваем один пин на вход для кнопки, а два других на выход для светодиодов.

2. Если значение переменной button равно False (кнопка нажата), включаем светодиод 24 и выключаем светодиод 26.

3. Ветвь «иначе». Если button не равно False, включаем светодиод 26 и выключаем24

2.Сохрани файл, запусти его командой в терминале python3 switch.py.

При нажатии на кнопку один светодиод будет гаснуть, а другой зажигаться и наоборот.

3.Замени знак равенства == на знак неравенства !=, светодиоды начнут работать в обратном режиме. Условие будет срабатывать, когда button не равно False.

Оператор	Значение
a < b	a меньше b
a <= b	a меньше либо равно b
a > b	a больше b
a >= b	a больше либо равно b
a == b	a равно b
a != b	a не равно b
(a < b) & (a >= c)	a меньше b, и a больше либо равно c
(a < b) | (a >= c)	a меньше b, или a больше либо равно c
a	a истинно или не ноль
a | b	a истинно либо не ноль, или b истинно либо не ноль
(not a) & (not b)	a ложно или ноль, и b ложно или ноль

ЗАДАНИЕ

Подключи вторую кнопку с именем stopButton. Пусть цикл while работает, пока она не нажата. После работы цикла погаси все светодиоды, а затем вызови функцию GPIO.cleanup().

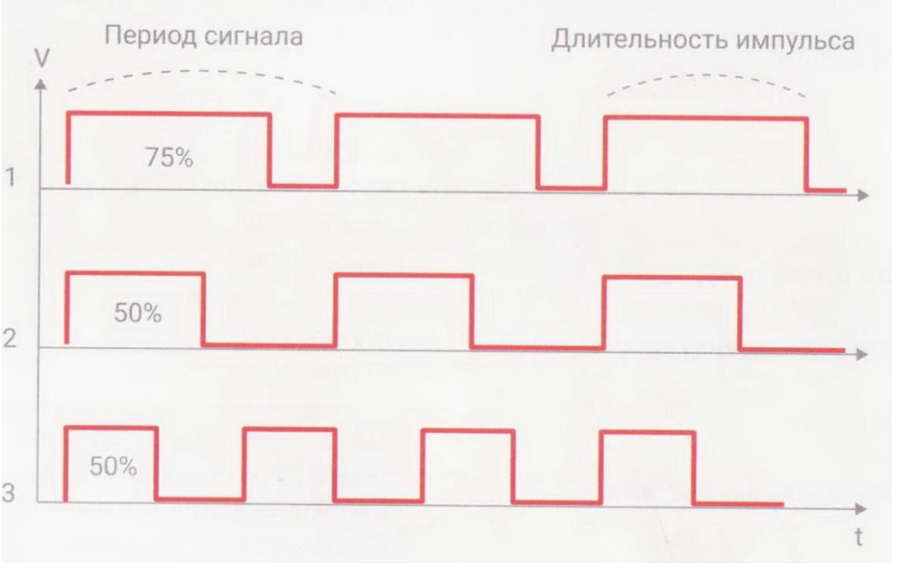
ЛАБОРАТОРНАЯ РАБОТА № 3
ШИМ-модуляция

Цель работы: изучить работу ШИМ-модуляции.
Оборудование: одноплатный компьютер Raspberry Pi

Описание работы

УПРАВЛЕНИЕ ЯРКОСТЬЮ

В обычных условиях человек не может заметить мерцание частотой более 100 Гц. Поэтому, если включать и выключать светодиод чаще, кажется, что он горит не в полную силу.
Для такого быстрого переключения существует специальный механизм — широтно-импульсная модуляция, сокращённо ШИМ (PWM – pulse-width modulation).
У ШИМ есть 2 величины: частота и коэффициент заполнения. Коэффициент заполнения (duty cycle) задает отношение длительности импульса к периоду и выражается в долях или процентах.



1 и 2 - одинаковая частота (период), разные коэффициенты заполнения.
2 и 3 - одинаковые коэффициенты заполнения, разные частоты.

1.Попробуй менять яркость светодиода на пине 18. Создай файл brightnessControl.py и напиши в нем программу.

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
pwm = GPIO.PWM(18, 1000)
dutyCycle = 50
pwm.start(dutyCycle)

try:
    while True:
        time.sleep(0.01)
        dutyCycle = dutyCycle + 1
        if (dutyCycle > 100):
            dutyCycle = 0
        pwm.ChangeDutyCycle(dutyCycle)

except KeyboardInterrupt:
    print('The program was stopped by keyboard.')
finally:
    GPIO.cleanup()
    print('GPIO cleanup completed.')
```

1. Пин 18 будет работать в режиме ШИМ на частоте 1000 герц.

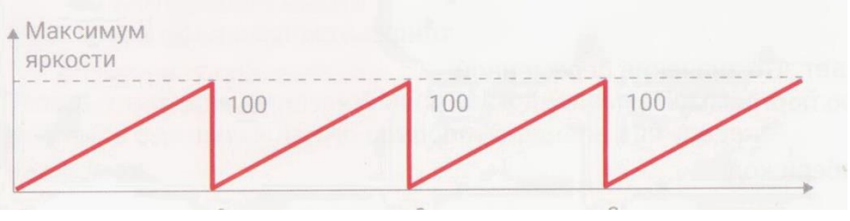
2. Запускаем ШИМ на пине с коэффициентом заполнения 50%, это зажжет светодиод в полсилы.

3. На каждой интерации цикла увеличиваем значение переменной на 1.

4. Коэффициент заполнения принимает значения от 0 до 100. Если вышли за допустимые границы, обнуляем значение.

5. Обновляем коэффициент заполнения новым значением dutyCycle.

2.Сохрани файл, запусти программу. Светодиод 18 начнёт плавно увеличивать яркость и, набрав максимум, сбросится «в ноль».



ЗАДАНИЕ

Увеличь скорость нарастания яркости в 4 раза (это можно сделать двумя способами).

ПАНЕЛЬ УПРАВЛЕНИЯ СВЕТОМ

Вспомни эксперимент «Выключатель»: мы по кнопке зажигали и гасили один светодиод.

```
button = GPIO.input(2)
GPIO.output(24, button)
```

Если создать приборную панель для управления несколькими выключателями, потребуется по две строчки кода на каждую пару кнопка-светодиод (причем код каждой пары будет практически одинаковым). Это не удобно и не красиво.

Чтобы не писать один и тот же код несколько раз, разработчики создают функции. В языке python они оформляются служебным словом def (define - определять).

```
1  def functionName(param1, param2):
2      result = param1 + param2
3      return result
```

1. Определение функции.

2. Тело функции.

functionName – имя функции, его задает сам разработчик кода.

ВАЖНО!
Принято давать функциям понятные читаемые имена. Пример плохих имён: a, myFunc, sdf, dothis. Хорошие имена: turnOn, handleButton, setup.

В скобках указываются параметры, с которыми будет работать функция. Параметров может быть много или ни одного. В теле записывают код. Здесь functionName складывает две переменные.

Служебное слово return означает, что значение переменной result вернётся в вызывающую программу.

1.Создай файл function.py и набери код:

```
1  import RPi.GPIO as GPIO
2
3  def isPressed(btn, led):
4      if (GPIO.input(btn) == False):
5          GPIO.output(led, GPIO.HIGH)
6      else:
7          GPIO.output(led, GPIO.LOW)
8
9  button1 = 3
10 button2 = 4
11 led1 = 14
12 led2 = 15
```

1. Создаём функцию с именем isPressed. Говорим, что она принимает 2 параметра: пин кнопки и пин светодиода. В теле функции пишем условие if... else... для управления освещением.

```
13
14  GPIO.setmode(GPIO.BCM)
15  GPIO.setup(button1, GPIO.IN)
16  GPIO.setup(button2, GPIO.IN)
17  GPIO.setup(led1, GPIO.OUT)
18  GPIO.setup(led2, GPIO.OUT)
19
20  try:
21      while True:
22          isPressed(button1, led1)
23          isPressed(button2, led2)
24  except KeyboardInterrupt:
25      print('The program was stopped by keyboard.')
26  finally:
27      GPIO.cleanup()
28      print('GPIO cleanup completed.')
```

2. Сохрани файл и запусти скрипт.

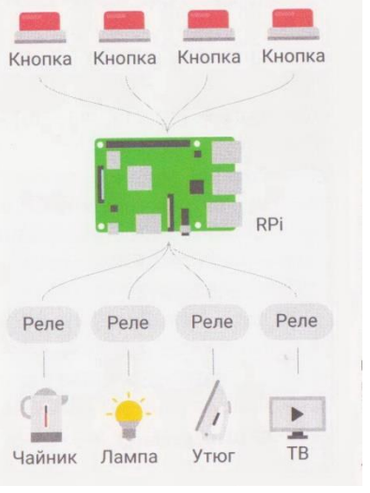
Теперь у тебя две пары кнопок со светодиодами. Несложно добавить 3-ю пару и целую приборную панель для управления домом.

Помни: бытовые приборы, которые питаются от сети, необходимо подключать к Raspberry через реле.

2. В цикле while вызываем функцию isPressed и передаём пары кнопка-светодиод.

ЗАДАНИЕ

Пусть каждый светодиод в панели управления светится вполсилы. Используй ШИМ.



МАССИВНАЯ ОПТИМИЗАЦИЯ

Оптимизируем использование переменных. Так код станет ещё короче и читабильней.

```
led1 = 12
led2 = 13
led3 = 14
led4 = 18
```

Эти кнопки похожи друг на друга. Они отличаются только порядковым номером. Такие переменные удобно объединять в массивы.

```
leds = [12, 13, 14, 18]
```

Такая запись гораздо короче.

1.Создай файл massiv.py и напиши в нем программу для управления сразу четырьмя светодиодами.

```
1  import RPi.GPIO as GPIO
2  import time
3
4  GPIO.setmode(GPIO.BCM)
5
6  leds = [12, 13, 14, 18]
7
8  for led in leds:
9      GPIO.setup(led, GPIO.OUT)
10     GPIO.output(led, GPIO.HIGH)
11
12 time.sleep(3)
13
```

1. создаём массив светодиодов.

2. в цикле for настраиваем пины светодиодов на режим цифрового выхода и сразу включаем их.

```
14 for led in leds:
15     GPIO.output(led, GPIO.LOW)
16
17 GPIO.cleanup()
```

3. через 3 секунды после включения гасим светодиоды.

4. освобождает пины от управления программой.

Теперь количество светодиодов задается только в одной строке.

2.Сохрани файл и запусти. Светодиоды загорится на 3 секунды и погаснут.
3.Сделай панель управления на двух массивах.

```python
import RPi.GPIO as GPIO

def isPressed(btn, led):
    state = 1 - GPIO.input(btn)
    GPIO.output(led, state)

leds = [12, 13, 14, 18]
buttons = [2, 3, 4, 8]

GPIO.setmode(GPIO.BCM)
for i in range(4):
    GPIO.setup(leds[i], GPIO.OUT)
    GPIO.setup(buttons[i], GPIO.IN)

try:
    while True:
        for i in range(4):
            isPressed(buttons[i], leds[i])
except KeyboardInterrupt:
    print('The program was stopped by keyboard.')
finally:
    GPIO.cleanup()
    print('GPIO cleanup completed.')
```

1.GPIO. input() возвращает либо 0 либо 1. Вычитаем из единицы значение, получаем обратное.

2.Функция range(n) создаёт диапазон от 0 до n для перебора в цикле for. Число 4 равно длине массивов кнопок и светодиодов.

ЗАДАНИЕ

Добавь в тело цикла while цикл for, который будет по очереди зажигать и гасить светодиоды на облачке. Получится бегущий огонек.

4.Запусти скрипт и увидишь, как светодиоды отзываются на кнопки.

5.Заверши выполнение программы клавишами ctrl+C.

ЛАБОРАТОРНАЯ РАБОТА № 4
Передача данных с помощью последовательных интерфейсов SPI

Цель работы: изучить передачу данных с помощью последовательных интерфейсов SPI

Оборудование: одноплатный компьютер Raspberry Pi

Теория работы

Последовательный периферийный интерфейс SPI (Serial Peripheral Interface) предназначен для связи МК с периферийными устройствами МП-системы, основой которой он является. Наиболее часто эти устройства расположены на одной плате с МК, реже – это вынесенные пульты управления, индикаторные панели и т.п.

В качестве периферийных устройств могут использоваться как простейшие сдвиговые регистры, так и сложные периферийные ИС со встроенными контроллерами управления, такие, как ЦАП, сигма-дельта АЦП с цифровой фильтрацией, последовательные запоминающие устройства типа FLASH или EEPROM, энергонезависимые ОЗУ и т.д.

Каждая из периферийных ИС является ведомым устройством. SPI-шина представлена тремя общими линиями связи (MISO, MOSI, SCK) и двумя линиями выбора ведомого устройства (SS1, SS2), которые индивидуальны для каждой периферийной ИС:

• MOSI – линий передачи данных от ведущего к ведомому (Master Output Slave Input).

• MISO – линия передачи данных от ведомого к ведущему (Master Input Slave Output).

• SCK – линия сигнала стробирования данных.

• SS1 и SS2 – линии сигналов выбора ведомого устройства.

Линии передачи данных и линия синхронизации являются примером шинной организации, а линии выбора ведомого устройства – элементом системы радиального типа. Перед началом обмена ведущее устройство отмечает одно ведомое устройство, с которым будет производиться обмен. Для этого на линии выбора устройства SSi устанавливается низкий активный уровень сигнала. Затем ведущее устройство последовательно выставляет на линию MOSI восемь бит информации, сопровождая каждый бит сигналом синхронизации SCK. Ведомое устройство дешифрирует переданный байт информации и определяет, в каком направлении будет производиться дальнейший обмен.

Если ведомое устройство должно принимать информацию, то ведущее устройство, не снимая сигнала выбора ведомого SSi, продолжит передачу по линии MOSI. Если ведомое устройство должно передавать информацию, то оно активизирует линию MISO и в ответ на каждый импульс синхронизации от ведущего будет выставлять один бит информации. Длина посылки обмена в общем случае не ограничена и может составлять даже не целое число байтов. Завершение обмена также инициируется ведущим посредством установки в неактивное состояние сигнала выбора ведомого SSi.

Значительно реже SPI-интерфейс используется для организации межпроцессорной связи. По мнению автора, чрезвычайно удобным может оказаться его применение в достаточно простых, но многопроцессорных системах управления устройствами силовой электроники, которые конструктивно требуют смещения устройств управления разными потенциалами. И именно полнодуплексный SPI-интерфейс позволяет организовать индивидуальную потенциальную развязку для каждого МК системы.

Для подключения к SPI-шине встроенный контроллер SPI имеет четыре вывода: MOSI, MISO, SCK, SS. Встроенный контроллер SPI может работать как в ведущем, так и в ведомом режиме. Скорость приема и передачи определяется частотой тактирования межмодульных магистралей МК busf: в ведущем режиме скорость обмена не может превышать 2busf, в ведомом режиме максимальная скорость обмена равна busf. Поэтому для большинства МК максимальная скорость обмена в ведущем режиме составляет 1 Мбит/с, в ведомом – 2 Мбит/с. При работе встроенного контроллера в ведущем режиме к выводу MOSI подключается выходная линия данных, а к MISO – входная. При работе в ведомом режиме выводы меняются ролями. Вывод SCK является выходом, если контроллер SPI работает в ведущем режиме, и входом, если в ведомом. В системах с несколькими ведущими устройствами все выводы SCK соединяются вместе. То же делается с выводами MOSI и MISO. На время отсутствия связи буферы выводов встроенного контроллера SPI переводятся в высокоимпедансное состояние. Последнее позволяет избежать конфликтов на шине SPI. В противном случае несколько выводов MISO ведомых устройств одновременно были бы активными, что не позволило бы ведущему устройству произвести прием достоверной информации. Только одно из устройств системы в каждый момент времени может работать в ведущем режиме, остальные – только в ведомом.

Ведущее устройство формирует сигналы на выводах MOSI и SCK, которые поступают на одноименные выводы ведомых устройств. Одно из выбранных ведомых устройств передает данные через вывод MISO на вывод MOSI ведущего устройства. Автоматическое управление направлением передачи выводов MOSI, MISO и SCK позволяет обойтись без изменений во внешних схемах управления, когда новое устройство начинает работать в ведущем режиме. Вывод SS встроенного контроллера SPI используется в зависимости от того, в каком режиме работает данное устройство. При работе в ведомом режиме при подаче высокого уровня сигнала на вход SS устройство игнорирует сигналы SCK и удерживает вывод MISO в высокоимпедансном состоянии. Если же в ведомом режиме работы на входе SS установлен низкий логический уровень, то буферы линий MOSI и SCK разворачиваются на ввод, линия MISO – на вывод.

При работе в ведущем режиме вывод SS может быть использован как обычная линия вывода. В системах со сложной логикой работы этот вывод может использоваться как вход сигнала обнаружения ошибки для индикации состояния шины в случаях, если более чем одно устройство пытается стать ведущим. Схема управления контроллера SPI-интерфейса позволяет выбрать один из двух протоколов обмена и полярность импульсов синхронизации SCK. При работе в ведущем режиме возможно также программно выбрать частоту импульсов синхронизации. Протоколы связи SPI. Два бита регистра управления любого контроллера SPI-интерфейса определяют временную диаграмму обмена по шине

SPI: • Бит CPHA назначает протокол обмена. • Бит CPOL определяет полярность сигнала синхронизации SCK. В соответствии с комбинацией битов CPHA:CPOL принято различать четыре режима работы интерфейса SPI. Комбинации CPHA:CPOL = 00 соответствует режим 0, комбинации CPHA:CPOL = 11 – режим 3. Встроенный контроллер SPI МК позволяет программно настраивать режим SPI в процессе инициализации, в то время как периферийные ИС реализуют один или два режима SPI, которые определяются их техническим описанием.

Наиболее часто это режимы 0 и 3. На рис представлены временные диаграммы сигналов для протокола передачи CPHA = 0. Для сигнала SCK приводятся две диаграммы, различающиеся полярностью сигнала. Первая соответствует режиму 0, вторая – режиму 1. Диаграммы относятся как к ведущему, так и к ведомому устройству, поскольку выводы MISO и MOSI ведущего соединены с аналогичными выводами ведомого.

Сигнал SS подается только на ведомое устройство. Поэтому вывод SS у ведущего остается незадействованным и его диаграмма не представлена, но имеется в виду, что она соответствует неактивному состоянию. Встроенные контроллеры SPI выполнены таким образом, что длина посылки составляет один байт, что и отражено на временных диаграммах.

Начало обмена рассматриваемого протокола определяется установкой сигнала выбора ведомого SS в активное состояние SS = 0. При направлении передачи от ведущего к ведомому первый перепад сигнала синхронизации SCK используется ведомым устройством для запоминания очередного бита во внутреннем сдвиговом регистре контроллера SPI. Ведущий выставляет очередной бит посылки на линии MOSI по каждому четному фронту сигнала SCK. При передаче данных от ведомого к ведущему старший бит передаваемого байта должен быть выставлен ведомым на линию MISO сразу после изменения уровня сигнала SS = 0.

По первому фронту SCK уровень сигнала на линии MISO будет запомнен в младшем разряде сдвигового регистра ведущего устройства. По этой причине сигнал на линии выбора ведущего должен быть возвращен в неактивное состояние SS = 1 после передачи каждого байта в любом направлении. Тогда передача каждого нового байта будет сопровождаться предварительной установкой SS в 0.

Начало обмена для протокола опции CPHA = 1 определяет первое изменение уровня сигнала на линии SCK после установки сигнала выбора ведомого SS в активное состояние SS = 0. При передаче данных от ведущего к ведомому и в обратном направлении все нечетные перепады SCK вызывают выдвижение очередного бита посылки из сдвигового регистра передатчика на линию. Каждый четный перепад используется для записи этого бита в сдвиговый регистр приемника. Сигнал выбора ведомого может оставаться в активном состоянии S3 = 0 в течение передачи нескольких байт информации. Это несколько упрощает логику программного драйвера SPI.

Системные ошибки SPI. Предусмотрено обнаружение двух системных ошибок в системах, использующих SPI-стандарт. Первая ошибка носит название "ошибка режима", вторая – "ошибка записи".

Ошибка режима. Эта ошибка возникает при работе SPI в ведущем режиме, если сигнал на выводе SS изменяется с 1 на 0. Это происходит, когда какое-то из ведомых устройств пытается стать ведущим. Для буферных устройств

микроконтроллеров это может привести к катастрофическим последствиям. В некоторых случаях защита не может быть осуществлена в полной мере. Например, если второе устройство перешло в ведущий режим, но не изменило сигнал SS сразу. Или два ведомых устройства пытаются одновременно использовать линию MISO. В этом случае может быть состязание их выходных буферов, но сигнала " ошибки системы" не будет и буферы останутся незащищенными. Можно дать несколько советов разработчику по предотвращению необратимых последствий. Полезный совет. Включайте гасящий резистор от 1 до 10 кОм последовательно с выводами SPI. (При нормальной работе резистор не оказывает заметного воздействия на сигнал, но при состязании буферов он ограничивает ток и предотвращает возможность их повреждения.) Или используйте режим " монтажное ИЛИ". Следует отметить, что оба совета связаны с некоторым снижением скорости передачи, которая начинает зависеть от емкости соединительных проводов. Ошибка записи. Эта ошибка возникает при записи в регистр данных в процессе передачи данных и определяется особенностями логики встроенных систем, реализующих протокол связи в стандарте SPI. Поскольку регистр данных не имеет буфера для передаваемых данных, запись производится непосредственно в сдвиговый регистр. Для предотвращения искажения передаваемых данных предусматривается сигнал ошибки записи.

Передача данных продолжается без нарушений, но новые данные в сдвиговый регистр не записываются. Эта ошибка обычно возникает в ведомых устройствах, так как они не располагают точной информацией о том, когда ведущее устройство выйдет на связь. Ведущее устройство само инициирует передачу и знает, когда работает его передатчик, поэтому ошибка такого рода для него маловероятна, хотя она обнаруживается так же, как и в ведомом устройстве.

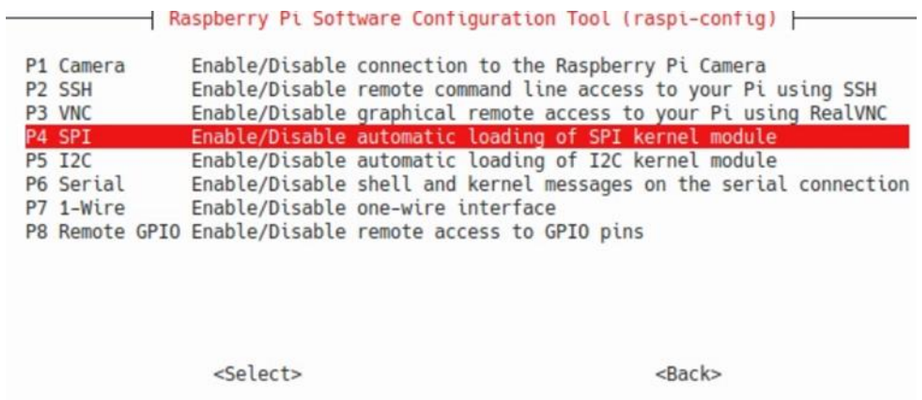
Описание работы

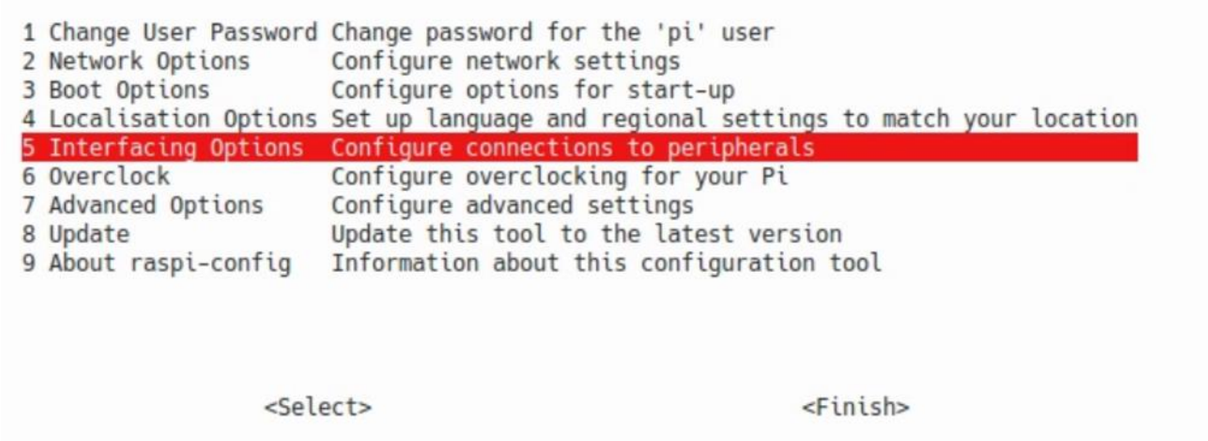
1. Активировать интерфейс SPI платы Raspberry Pi.

Для того, чтобы активировать интерфейс SPI платы Raspberry Pi можно воспользоваться утилитой *raspi-config*. Вводим команду:

sudo raspi-config

После запуска утилиты идем по пути *Interfacing options — SPI* и включаем нужный нам интерфейс:





Другим способ активации SPI является непосредственное редактирование файла */boot/config.txt*. В нем нужно раскомментировать строку:

dtparam=spi=on

В обоих случаях после изменения конфигурации необходимо перезапустить плату:

sudo reboot

Не важно, каким именно способом пользоваться, результат будет одинаков. Для проверки вводим команду:

ls /dev

Наличие в списке *spidev0.0* и *spidev0.1* сигнализирует нам о том, что интерфейс SPI успешно активирован. Первая цифра (**0**) в наименовании указывает, что используется SPI**0**, к которому и подключено подчиненное устройство.

А вторая цифра (**0** или **1**) определяет, какой вывод платы будет использован в качестве Chip Select'а. Здесь есть два варианта:

- SPI0_CE**0**_N
- SPI0_CE**1**_N

В случае, если использован SPI0_CE**0**_N, нас интересует spidev0.**0**. Два вывода Chip Select нужны для того, чтобы реализовать возможность подключения двух разных устройств к SPI0, каждому из которых будет соответствовать свой сигнал Chip Select (**CS**), а линии данных и тактирования (**MOSI**, **MISO**, **SCLK**) будут общими.

При таком подключении, если мы собираемся обмениваться данными с одним из устройств, то следует подать низкий уровень (0) на его вывод CS. А на CS второго устройства — высокий уровень (1). Таким образом и осуществляется переключение между устройствами, использующими одни и те же линии данных.

2. Изучить работу с интерфейсом SPI с помощью модуля *python-spidev*.

Модуль *python-spidev* можно установить командой:

sudo apt-get install python-spidev

Если при запуске написанного кода будут возникать ошибки, то поможет принудительная установка версии 3.4 пакета:

pip3 install spidev==3.4 —force-reinstall

Для использования SPI в программме делаем *import*:

import spidev

Далее нужно создать объект для работы с **SPI** и произвести подключение:

```
spi = spidev.SpiDev()
spi.open(0, 0)
```

Аргументы функции *open()* как раз и позволяют выбрать использующийся модуль и сигнал Chip Select, который мы обсудили ранее. В нашем примере — SPI**O** и SPIO_CE**O**_N. В целом, после вызова этих функций уже можно начинать работу с SPI, но давайте рассмотрим помимо прочего доступные варианты конфигурации:

- **bits_per_word** — количество битов в слове.
- **cshigh** — в активном режиме сигнал на выходе Chip Select может быть либо высокого, либо низкого уровня. Для первого случая ставим *cshigh = True*, для второго, соответственно — *cshigh = False*.
- **loop** — включение/отключение loopback режима.
- **no_cs** — записываем в параметр *True*, если не используем Chip Select.
- **lsbfirst** — выбираем, наиболее или наименее значимый бит будет первым в данных.
- **max_speed_hz** — максимальная частота интерфейса.
- **mode** — в этот параметр записываем два бита, например 0b01:

Бит	Сигнал	Описание	Значения
Старший	CPOL	Исходный уровень сигнала синхронизации	0: низкий уровень 1: высокий уровень
Младший	CPHA	Фаза синхронизации	0: по переднему фронту — выборка данных, по заднему — установка 1: по переднему фронту — установка данных, по заднему — выборка

- **threewire** — режим, при котором вместо двух линий MOSI и MISO используется одна, объединяющая в себе их функции.

В коде установка параметров может происходить следующим образом:

```
spi.lsbfirst = False
spi.cshigh = False
spi.mode = 0b01
spi.bits_per_word = 8
```

Основные функции для приема и передачи данных:

•**xfer**(list of values[, speed_hz, delay_usec, bits_per_word])

Функция осуществления SPI транзакции. В квадратных скобках указаны необязательные аргументы, среди которых последовательно: частота, величина задержки между данными в микросекундах и количество битов в слове. Обязательный аргумент *list of values* — непосредственно передаваемые данные. Возвращает же функция данные, принятые от подчиненного устройства.

•**xfer2**(list of values[, speed_hz, delay_usec, bits_per_word])

Аналогичная функция, отличие от *xfer()* заключается в том, что сигнал Chip Select остается активным между транзакциями. В случае же с *xfer()* после каждой отправки CS переходит в неактивное состояние, а затем снова активируется.

•**xfer3**(list of values[, speed_hz, delay_usec, bits_per_word])

И еще одна функция из этой группы. Ее отличие заключается в том, что если размер передаваемых данных превышает максимально допустимый, то данные будут разбиты на порции и отправлены по частям. Максимальный размер передаваемых данных задается через */sys/module/spidev/parameters/bufsiz*.

•**readbytes**(n)

Функция приема данных, аргумент — количество байт, которые необходимо прочитать.

•**writebytes**(list of values)

•

Передача данных, задаваемых аргументом функции.

•**writebytes2**(list of values)

•

И похожая на предыдущую функция, отличающаяся тем, что разбивает данные на части в случае превышения максимального размера. В общем, аналогично функции *xfer3()*.

Для того, чтобы запросить значение регистра с адресом *0x0F*, нам нужно отправить **подчиненному** (датчику) команду *0x8F*. В ответ на это гироскоп отправит значение регистра. Нюанс заключается в том, что, отправив команду, **ведущий** (Raspberry Pi) должен обеспечивать тактирование на линии в то время, как идет отправка данных **подчиненным**. Для того, чтобы это осуществить, отправим 2 байта — *0x8F* и *0x00*. Второй байт не несет никакой смысловой нагрузки, он нужен только для того, чтобы **master** продолжал выдавать сигнал тактирования в то время, пока **slave** отправляет данные.

Создаем файл *spi_test.py*. Итоговый код будет выглядеть так:

```
import spidev
spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 4000000
txData = [0x8F, 0x00]
rxData = spi.xfer(txData)
for i in range(len(rxData)):
print (hex(rxData[i]))
spi.close()
```

Запускаем:

```
python spi_test.py
```

Отправляем два байта и в ответе от *slave* получаем также два байта, первый из которых не несет полезной информации. Он соответствует тому периоду времени, когда *master* отправлял запрос (0x8F), соответственно на шине было тактирование, а на MISO в этот момент был низкий уровень, который и был прочитан как 0x00.

ЛАБОРАТОРНАЯ РАБОТА № 5
Передача данных с помощью последовательных интерфейсов I2C

Цель работы: изучить передачу данных с помощью последовательных интерфейсов I2C

Оборудование: одноплатный компьютер Raspberry Pi

Теория работы

Синхронный последовательный интерфейс I2C Интерфейс I2C (синхронный последовательный интерфейс I2C) является двухпроводным последовательным интерфейсом, разработанным фирмой Philips. Изначально в стандартном режиме шина была предназначена для скорости передачи данных до 100 кбит/с. В усовершенствованном быстром режиме поддерживается передача данных со скоростью до 400 кбит/с. На одной шине одновременно могут работать устройства и стандартного, и быстрого режима. Основными свойствами шины являются следующие: • Двунаправленная передача данных между главными и подчиненными устройствами. • Многоабонентская связь (нет центрального главного узла). • Арбитраж между одновременно передающими устройствами без разрушения целостности передаваемых данных. • Последовательная тактовая синхронизация позволяет приборам с различными скоростями передачи битов осуществлять связь через одну последовательную шину. • Последовательная тактовая синхронизация может использоваться в качестве механизма квитирования установления связи, чтобы приостанавливать и возобновлять последовательную передачу. I2C-шина может использоваться в целях тестирования и диагностики. I2C-шина использует два провода (SDA и SCL) для передачи информации между приборами, соединенными этой шиной.

Протокол связи I2C. Интерфейс I2C использует протокол с битом подтверждения ASK для обеспечения надежной передачи и приема. При передаче одно устройство – ведущее (выдает тактовый сигнал), а другое – ведомое. Все составляющие протокола ведомого, включая поддержку общего вызова, а также протокол ведущего реализованы аппаратно в модуле I2C

Адресация на шине I2C. В протоколе интерфейса I2C каждое устройство имеет адрес. Когда ведущий хочет инициировать передачу данных, он сначала передает адрес устройства SLA, к которому хочет обратиться. Все устройства на шине следят за выставляемым на шину адресом и сравнивают его с собственным. Вместе с адресом передается бит направления передачи R/W, который определяет, будет ли ведущий читать из ведомого или писать в него. Существуют два формата адреса. Простейший – 7-битный формат с битом R/W. Адрес – это 7 старших битов байта. Более сложным является 10-битный формат с битом R/W. В 10-битном адресе первые пять битов определяют, что это 10-битный адрес.

Основные типы передачи данных. В зависимости от направления передачи битов (R/W) для I2C-шины возможны два типа передачи данных:

Передача данных от главного передатчика к подчиненному приемнику. Первый байт, передаваемый передатчиком, является адресом подчиненного приемника. Затем следует несколько байтов данных. Подчиненный приемник возвращает бит подтверждения после каждого принятого байта. • Передача данных

от подчиненного передатчика к главному приемнику. Первый байт (адрес подчиненного передатчика) передается главным устройством. Затем подчиненный передатчик возвращает бит подтверждения. Следующие несколько байтов данных передаются подчиненным устройством к главному. Главное устройство возвращает бит подтверждения после всех принятых байтов, кроме последнего байта. В конце последнего принятого байта возвращается " нет подтверждения". Инициализация и прекращение передачи данных. Когда нет передачи данных (режим ожидания), линии тактирования SCL и данных SDA приведены подтягивающим резистором к высокому уровню. Главное устройство генерирует все последовательные синхроимпульсы и условия START и STOP, определяющие начало и конец передачи данных. Условие START определяется как переход SDA из высокого уровня в низкий при высоком уровне SCL, а условие STOP – как переход SDA из низкого уровня в высокий при высоком уровне SCL. Ввиду такого способа определения условий START и STOP при передаче данных, линия SDA может изменять свое состояние только при низком уровне SCL. Когда ведущий не желает освобождать шину (выставив STOP), он должен выставить повторное условие START, которое идентично START (переход SDA из высокого уровня в низкий при высоком уровне на SCL), но выдается вслед за подтверждением, являясь началом следующей последовательной передачи, таким образом шина не освобождается. На рис. 20 показано, как осуществляется передача данных по шине в соответствии с принятым стандартом протокола связи.

Режимы работы I2C-логики. I2C-логика устройств связи, обеспечивающая последовательный интерфейс, который удовлетворяет спецификации I 2C-шины и поддерживает два вышеперечисленных типа передачи данных, может работать в следующих четырех режимах.

Режим главного передатчика. Последовательный вывод данных через SDA-линию, в то время как SCL выдает последовательные синхроимпульсы. Первый переданный байт содержит адрес подчиненного приемного устройства (7 бит) и бит направления данных. В этом случае бит направления данных (R/W) будет логическим 0, мы говорим, что передается "W". Таким образом, первый переданный байт представляет собой SLA+W. Последовательные данные передаются по 8 бит. Условия START и STOP выводятся для указания начала и конца последовательной передачи. Режим главного приемника. Первый переданный байт содержит адрес подчиненного передающего устройства (7 бит) и бит направления данных. В этом случае бит направления данных (R/W) будет логической 1 и мы говорим, что передается "R". Таким образом, первый переданный байт представляет собой SLA+R. Последовательные данные передаются через линию SDA, в то время как линия SCL выдает последовательные синхроимпульсы. Последовательные данные передаются по 8 бит. После того как принят каждый байт, передается бит подтверждения. Условия START и STOP выводятся для индикации начала и конца последовательной передачи. Режим подчиненного приемника. Последовательные данные и последовательные синхроимпульсы передаются через линии SDA и SCL. После того как принят каждый байт, передается бит подтверждения. Условия START и STOP распознаются как начало и конец последовательной передачи. Распознавание адреса выполняется аппаратным обеспечением после приема адреса подчиненного устройства и бита направления. Режим подчиненного передатчика. Первый байт

принимается и обрабатывается как в режиме подчиненного приемника. Однако в этом режиме бит направления будет указывать, что направление передачи изменено на обратное. Последовательные данные передаются через линию SDA, в то время как последовательные синхроимпульсы вводятся через SCL. Условия START и STOP выводятся для индикации начала и конца последовательной передачи. В подчиненном режиме аппаратные средства I2C-интерфейса осуществляют поиск своего собственного подчиненного адреса и адреса общего вызова. Если детектируется один из этих адресов, запрашивается прерывание. Когда МК желает, чтобы шина стала главной, аппаратные средства ожидают, пока шина освободится, прежде чем ввести главный режим, так что возможное действие в качестве подчиненного не прерывается. Если арбитраж шины потерян в главном режиме, то соответствующий передатчик переключается в подчиненный режим немедленно и может детектировать свой собственный подчиненный адрес в той же последовательной передаче. Работа при конкуренции. Протокол I2C допускает наличие более одного ведущего в системе. Это называется конкуренцией. Когда два или более ведущих пытаются передать данные одновременно, необходимы арбитраж и синхронизация. Арбитраж. Арбитраж осуществляется на шине данных SDA, пока на SCL высокий уровень. В режиме главного передатчика арбитражная логика проверяет, чтобы каждая переданная логическая единица действительно появлялась в виде логической 1 на I2C-шине. Если другой прибор на шине отменяет логическую 1 и переводит линию SDA в низкий уровень, арбитраж теряется и соответствующий ведущий немедленно переходит из режима главного передатчика в режим подчиненного приемника, поскольку устройство, выигравшее арбитраж, может обратиться к нему. Устройство будет продолжать выдавать синхроимпульсы (на SCL) до тех пор, пока передача текущего последовательного байта не будет завершена. Арбитраж может также быть потерян в режиме главного приемника. Потеря арбитража в этом режиме может произойти только в то время, когда ведущее устройство возвращает сигнал " нет подтверждения" (логическая 1) на шину. Арбитраж теряется, когда другой прибор на шине изменит этот сигнал на низкий уровень. Поскольку это может произойти только в конце последовательного байта, устройство не генерирует в дальнейшем тактовые импульсы. Арбитраж не допускается между повторными сигналами START, условием STOP и битом данных, повторным START и STOP.

Ведущий должен обеспечить невозможность возникновения приведенных условий.

Синхронизация тактовых сигналов. Синхронизация тактовых сигналов возникает, когда устройства начинают арбитраж, и реализуется благодаря использованию соединения "монтажного И" с SCL. Переход SCL из высокого уровня заставляет все устройства, вовлеченные в арбитраж, начать отсчет длительности низкого уровня. После того как тактовый сигнал устройства перешел, а низкий уровень, оно будет удерживать этот уровень на SCL, до тех пор, пока тактовый сигнал не перейдет в высокий уровень, но при этом на SCL может по-прежнему сохраняться низкий уровень, если уровень тактового сигнала другого устройства все еще низкий. Низкий уровень на SCL удерживается устройством с самым длинным полупериодом низкого уровня. Устройства с более коротким полупериодом выдают высокий уровень на SCL и переходят в состояние ожидания, до тех пор, пока на SCL не появится высокий уровень. После этого все устройства

начинают отсчет длительности высокого уровня. Устройство с самым коротким полупериодом высокого уровня по завершении его переведет SCL в низкий уровень. Длительность высокого уровня на SCL определяется устройством с самым коротким полупериодом высокого уровня.

Коллизии на линиях шины и их обработка PC-логикой. Аппаратные средства I2C-интерфейса имеют возможность обрабатывать некоторые специальные случаи, которые могут возникать во время последовательной передачи.

Одновременно повторяющиеся условия START от двух главных устройств. Повторяющиеся условия START могут генерироваться в режиме главного передатчика или главного приемника. Особая ситуация возникает, если два главных устройства одновременно генерируют повторяющееся условие START. Пока это происходит, арбитраж не теряется ни одним из главных устройств, так как они оба передают одни и те же данные. Если аппаратные средства I 2C детектируют повторяющееся условие START на I2C-шине до того, как они сами генерируют повторное условие START, то они освободят шину и запрос на прерывание генерироваться не будет. Если другое главное устройство " освобождает" шину путем подачи условия STOP, исходное устройство будет передавать нормальное условие START и пытаться повторно начать прерванную передачу последовательных данных

исходное устройство будет передавать нормальное условие START и пытаться повторно начать прерванную передачу последовательных данных. Принудительный доступ к I2C-шине. В некоторых случаях неконтролируемый источник может вызвать неожиданный останов (зависание) шины. В подобных ситуациях проблема может быть вызвана взаимными помехами, временным прерыванием шины или временным коротким замыканием между SDA и SCL.

Если неконтролируемый источник генерирует ненужный START или маскирует условие STOP, тогда I2C-шина остается занятой неопределенное время. Если доступ к шине не обеспечивается в течение разумного промежутка времени, возможен принудительный доступ. Никакого условия STOP в этом случае не передается. Аппаратные средства I2C ведут себя так, как если бы условие STOP было принято, и способны передавать условие START. Блокировка I2C-шины низким уровнем SCL или SDA. Неожиданный останов I2C-шины происходит, если на линии SDA или SCL неконтролируемый источник установит логический 0. Если линия SCL блокируется (подключается к низкому уровню) прибором на шине, дальнейшая последовательная передача невозможна и аппаратные средства I2C не могут решить проблему этого типа. Когда это происходит, проблема должна быть решена со стороны того прибора, который подключил линию SCL к низкому уровню. Если линия SDA блокируется другим прибором на шине (например, подчиненным прибором вне бита синхронизации), проблема может быть решена путем передачи дополнительных тактовых импульсов на линию SCL (рис.). Аппаратные средства интерфейса пытаются генерировать условие START после каждых двух дополнительных тактовых импульсов на линии SCL. Когда линия SDA в конце концов освобождается, передается нормальное условие START и последовательная передача продолжается. 24Если происходит принудительный доступ к шине или передается повторное условие START, в то время когда линия SDA заблокирована (подключена к низкому уровню), аппаратные средства выполняют действия, аналогичные вышеописанным. Ошибка шины. Ошибка шины

возникает в случае, если условия START или STOP устанавливаются в неправильной позиции по отношению к разрядной сетке. Неправильные позиции возникают во время последовательной передачи адресного байта, данных или бита подтверждения. Аппаратные средства I2C реагируют на ошибку шины только тогда, когда они участвуют в последовательной передаче в главном или в адресуемом подчиненном режимах. Когда детектируется ошибка шины, интерфейс сразу же переключается в неадресуемый подчиненный режим, освобождает линии SDA и SCL и устанавливает флаг прерывания. Далее производится попытка вновь повторить прерванную последовательную передачу.

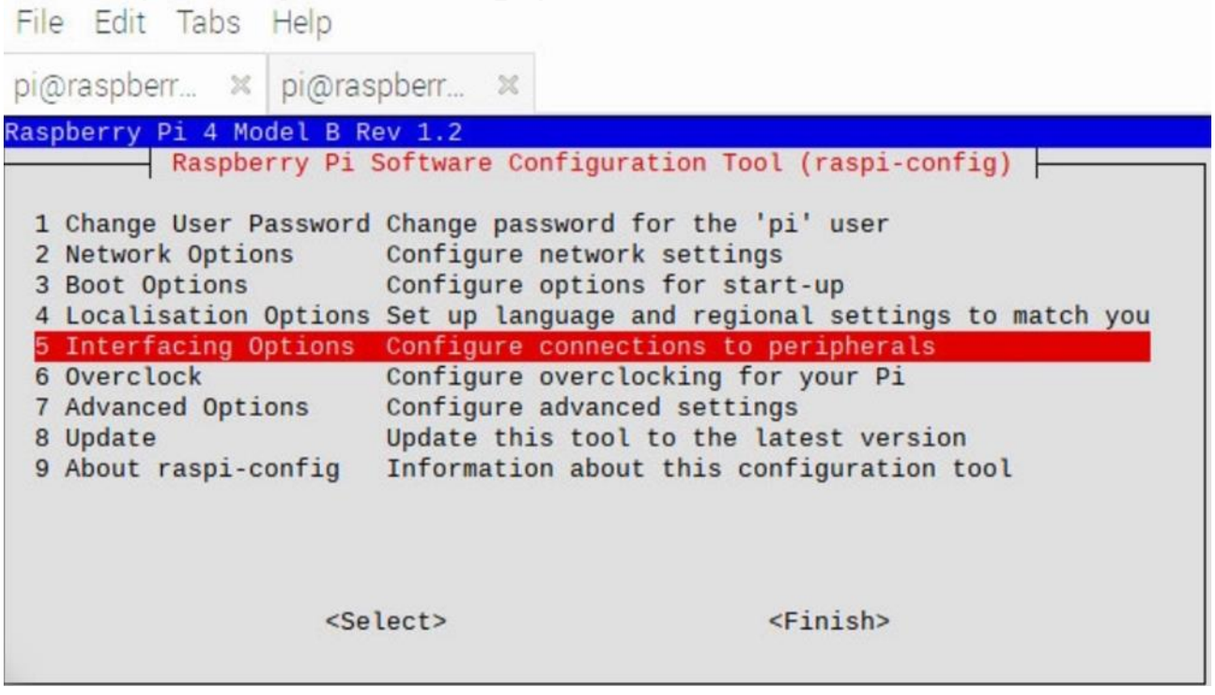
Описание работы

1. Включение I2C на Raspberry Pi.

а. Включить интерфейс I2C. Для этого воспользуемся утилитой для конфигурации Raspberry Pi — raspi-config. Вводим в терминале команду:

```
sudo raspi-config
```

б. Перейти в пункт Interfacing options и включаем I2C:



в. Перезапустить плату:

```
sudo reboot
```

Альтернативным способом включения I2C является редактирование файла /boot/config.txt. В файле находим и раскомментируем строку:

```
dtparam=i2c_arm=on
```

После чего аналогичным образом перезапускаем плату. При следующем включении модуль I2C будет активирован. Оба этих способа можно использовать, подключившись к плате по SSH.

Для I2C, как и для других модулей платы, существуют разные варианты и методы использования.

2. Устанавливаем пакет i2c-tools командой:

```
sudo apt-get install -y i2c-tools
```

3. Изучить работу утилиты i2ctools. Для этого выполним ряд команд. Для просмотра доступных шин I2C используем команду i2cdetect:

```
i2cdetect -l
```

Для работы с шиной i2c-1 выполним команду

```
sudo i2cdetect -y 1
```

Цифра 1 здесь как раз и относится к тому, что мы будем работать с шиной i2c-1. Кроме того, используем ключ -y. По умолчанию команда i2cdetect ожидает подтверждения от пользователя прежде, чем вмешиваться в работу шины, а ключом -y мы даем ей добро на работу с I2C без дополнительного разрешения.

Кроме того, пакет i2c-tools включает в себя ряд других команд для работы с шиной, в частности:

i2cdump — позволяет прочитать значения всех регистров подключенного к шине устройства. Например:

```
i2cdump -y 1 0x68
```

2cget — чтение значения конкретного регистра определенного устройства. Конечно, же рассмотрим пример:

```
i2cget -y 1 0x68 0x75
```

i2cset — запись определенного значения в регистр устройства. И снова не обходимся без примера:

```
i2cset -y 1 0x68 0x1A 0x03
```

4. Изучим работы шины I2C с помощью пакета python-smbus.

Для установки пакета выполним команду:

```
sudo apt-get install -y python-smbus
```

По традиционной схеме создаем файл, в котором и будем писать код для работы с I2C, назовем файл, к примеру, i2c_test.py. Ну и для наглядной демонстрации прочитаем значение все того же регистра с адресом 0x75 и сравним результаты. Собственно, пишем минимально необходимый для этого код:

```
from smbus import SMBus
```

```
bus = SMBus(1)
data = bus.read_byte_data(0x68, 0x75)
print(hex(data))
bus.close()
```

Давайте разберем, что тут происходит, поэтапно. Делаем import библиотеки:

```
from smbus import SMBus
```

Открываем использующуюся у нас шину i2c-1:

```
bus = SMBus(1)
```

Читаем значение регистра, выводим его на экран и заканчиваем работу с I2C:

```
data = bus.read_byte_data(0x68, 0x75)
print(hex(data))
bus.close()
```

Первый аргумент функции — адрес устройства, второй — адрес регистра. Запускаем выполнение нашего кода:

```
python i2c_test.py
```

ЛИТЕРАТУРА

1. Петин, В.А. «Микрокомпьютеры Raspberry Pi» / В.А. Петин. БХВ-Петербург, 2015. – 240 с.
2. Красиков, Р.В. Использование исполнительного модуля «CoDeSys Control» совместно с «Raspberry Pi» / Р.В. Красиков, В.Х. Аль-Тибби // Молодой исследователь Дона. – 2017. – № 3(6). – С. 45–51.
3. Барабанов, Ю.А. Микропроцессорные устройства релейной защиты и автоматики распределительных сетей / Ю.А. Барабанов. – Вологда: Инфра-Инженерия, 2015. – 172 с.
4. Хартов, В.Я. Микропроцессорные системы: учеб. пособие / В.Я. Хартов. – М.: Academia, 2017. – 320 с.

Учебное издание

МИКРОПРОЦЕССОРЫ И АППАРАТНЫЕ СРЕДСТВА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Методические рекомендации по выполнению лабораторных работ

Составители:
ШИЁНОК Юрий Владимирович
САПЕЛКО Татьяна Ивановна

Технический редактор *Г.В. Разбоева*
Компьютерный дизайн *Л.Р. Жигунова*

Подписано в печать 29.11.2021. Формат 60x84 $^{1}/_{16}$. Бумага офсетная.
Усл. печ. л. 2,44. Уч.-изд. л. 1,86. Тираж 40 экз. Заказ 180.

Издатель и полиграфическое исполнение – учреждение образования
«Витебский государственный университет имени П.М. Машерова».

Свидетельство о государственной регистрации в качестве издателя,
изготовителя, распространителя печатных изданий
№ 1/255 от 31.03.2014.

Отпечатано на ризографе учреждения образования
«Витебский государственный университет имени П.М. Машерова».
210038, г. Витебск, Московский проспект, 33.