Министерство образования Республики Беларусь

ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра вычислительных систем и сетей

**АРХИТЕКТУРА МОБИЛЬНЫХ ПЛАТФОРМ**

Методические указания по выполнению   
лабораторных работ для студентов специальностей

1-40 01 01 «Программное обеспечение информационных технологий» и  
1-40 02 01 «Вычислительные машины, системы и сети»  
всех форм обучения

А.О. Лукьянов

Полоцк 2020

Оглавление

[Введение 4](#_Toc58353364)

[1. Лабораторная работа №1 ИСПОЛЬЗОВАНИЕ ИНТЕРФЕЙСА ВВОДА/ВЫВОДА ОБЩЕГО НАЗАНАЧЕНИЯ НА ОДНОПЛАТНОМ КОМПЬЮТЕРЕ RASPBERRY PI 3 5](#_Toc58353365)

[1.1. Теоретическая часть 5](#_Toc58353366)

[1.1.1. Интерфейс ввода/вывода общего назначения (GPIO) 5](#_Toc58353367)

[1.1.2. Одноплатный компьютер Raspberry Pi 3 6](#_Toc58353368)

[1.1.3. GPIO на Raspberry Pi 3 7](#_Toc58353369)

[1.1.4. Нумерация выводов GPIO на Raspberry Pi 3 8](#_Toc58353370)

[1.1.5. Функциональные возможности GPIO-выводов Raspberry Pi 3 8](#_Toc58353371)

[1.1.6. Регистры для работы с GPIO Raspberry Pi 3 10](#_Toc58353372)

[1.1.7. Программирование портов ввода-вывода 10](#_Toc58353373)

[1.2. Задание 13](#_Toc58353374)

[1.3. Контрольные вопросы 14](#_Toc58353375)

[1.4. Содержание отчёта 14](#_Toc58353376)

[2. Лабораторная работа №2 ИСПОЛЬЗОВАНИЕ ШИРОТНО-ИМПУЛЬСНОЙ МОДУЛЯЦИИ НА ОДНОПЛАТНОМ КОМПЬЮТЕРЕ RASPBERRY PI 3 15](#_Toc58353377)

[2.1. Теоретическая часть 15](#_Toc58353378)

[2.1.1. Широтно-импульсная модуляция 15](#_Toc58353379)

[2.1.2. Программное и аппаратное формирование ШИМ-сигнала 16](#_Toc58353380)

[2.1.3. Использование параметров командной строки 19](#_Toc58353381)

[2.2. Задание 21](#_Toc58353382)

[2.3. Контрольные вопросы 21](#_Toc58353383)

[2.4. Содержание отчёта 22](#_Toc58353384)

[3. Лабораторная работа №3 КОНФИГУРИРОВАНИЕ ВЕБ-СЕРВЕРА НА ОДНОПЛАТНОМ КОМПЬЮТЕРЕ RASPBERRY PI 3 23](#_Toc58353385)

[3.1. Теоретическая часть 23](#_Toc58353386)

[3.1.1. Операционная система Raspbian 23](#_Toc58353387)

[3.1.2. Веб-сервер 23](#_Toc58353388)

[3.1.3. Программное обеспечение для работы веб-сервера 26](#_Toc58353389)

[3.1.4. Конфигурирование веб-сервера Apache 27](#_Toc58353390)

[3.1.5. Настройка виртуальных хостов Apache 29](#_Toc58353391)

[3.2. Задание 30](#_Toc58353392)

[3.3. Контрольные вопросы 31](#_Toc58353393)

[3.4. Содержание отчёта 31](#_Toc58353394)

[4. Лабораторная работа №4 ИСПОЛЬЗОВАНИЕ ВЕБ-СЕРВЕРА ДЛЯ ВЗАИМОДЕЙСТВИЯ С АППАРАТНЫМИ РЕСУРСАМИ ОДНОПЛАТНОГО КОМПЬЮТЕРА RASPBERRY PI 3 32](#_Toc58353395)

[4.1. Теоретическая часть 32](#_Toc58353396)

[4.1.1. Механизм клиент-серверного взаимодействия 32](#_Toc58353397)

[4.1.2. Серверные языки программирования 34](#_Toc58353398)

[4.1.3. Синтаксис языка программирования PHP 35](#_Toc58353399)

[4.2. Задание 39](#_Toc58353400)

[4.3. Контрольные вопросы 39](#_Toc58353401)

[4.4. Содержание отчёта 40](#_Toc58353402)

[Литература 41](#_Toc58353403)

# **Введение**

Целью выполнения лабораторных работ является изучение студентами аппаратной и программной архитектуры мобильных платформ, включая инструменты работы для программирования встраиваемых систем, современные аппаратные мобильные решения.

Задачи выполнения лабораторных работ дисциплины:

* изучение процессорных мобильных архитектур;
* изучение аппаратных мобильных архитектур;
* изучение программных мобильных архитектур;
* изучение современных одноплатных компьютеров.

Базовыми учебными дисциплинами для выполнения данных лабораторных работ являются такие дисциплины как «Системное программное обеспечение вычислительных систем», «Схемотехника».

# **Лабораторная работа №1 ИСПОЛЬЗОВАНИЕ ИНТЕРФЕЙСА ВВОДА/ВЫВОДА ОБЩЕГО НАЗАНАЧЕНИЯ НА ОДНОПЛАТНОМ КОМПЬЮТЕРЕ RASPBERRY PI 3**

**Цель:** изучить структуру и аспекты использования интерфейса общего назначения одноплатного компьютера.

## **Теоретическая часть**

### **Интерфейс ввода/вывода общего назначения (GPIO)**

Интерфейс ввода/вывода общего назначения (англ. general-purpose input/output, GPIO) — интерфейс для связи между компонентами компьютерной системы, к примеру микропроцессором и различными периферийными устройствами. Контакты GPIO могут выступать как в роли входа, так и в роли выхода — это, как правило, конфигурируется. GPIO контакты часто группируются в порты.

GPIO контакты не имеют специального назначения и, как правило, остаются неиспользованными. Это используется в том случае, когда системному интегратору для построения полной системы, использующей тот либо иной чип, может оказаться полезным иметь несколько дополнительных линий цифрового управления.

Это даёт возможность организовать дополнительные схемы, не создавая их с нуля. Например, чипы Realtek ALC260 (Аудио кодек) имеют 8 GPIO выводов, которые остаются неиспользованными по умолчанию. Некоторые системные интеграторы (к примеру Acer Inc. на своих ноутбуках), использующие ALC260, задействуют первый GPIO (GPIO0), чтобы включить усилитель, используемый для встроенных динамиков ноутбука и для разъема подключения наушников.

GPIO используются:

* в устройствах с нехваткой выводов (пинов, контактов): интегральных схемах, таких как однокристальные системы (SoC), встраиваемых и специальных системах (embedded и custom hardware) и программируемых логических устройствах (например FPGA);
* в многофункциональных чипах: управляющих питанием, аудио кодеках и видеокартах;
* во встраиваемых системах (например, Arduino, BeagleBone, различные PSoC комплекты и Raspberry Pi) широко используют GPIO для чтения информации от различных внешних датчиков (ИК, видео, температура, ориентации по 3 осям, ускорения), а также для управления двигателями постоянного тока (используя ШИМ), аудио, ЖК-дисплеями, или светодиодами для индикации состояния чего-либо.
  + 1. **Одноплатный компьютер Raspberry Pi 3**

Raspberry Pi – одноплатный компьютер, созданный работниками Кембриджского университета в 2006 году, и запущенный в промышленное производство в 2012 году.

Raspberry Pi представляет собой плату размером чуть больше кредитной карты, на которой распаян ARM-процессор, чипы оперативной памяти, слот под microSD-карту, а также Ethernet-порт, HDMI, 3,5мм аудиовыход и USB-порты для подключения периферийных устройств. Кроме того, как и на Arduino, на плате Raspberry Pi имеется GPIO-интерфейс. Все это работает под управлением адаптированного под ARM-архитектуру дистрибутива \*unix.

В настоящее время выпускается уже третье поколение плат Raspberry Pi – со встроенным адаптером Bluetooth и Wi-Fi.

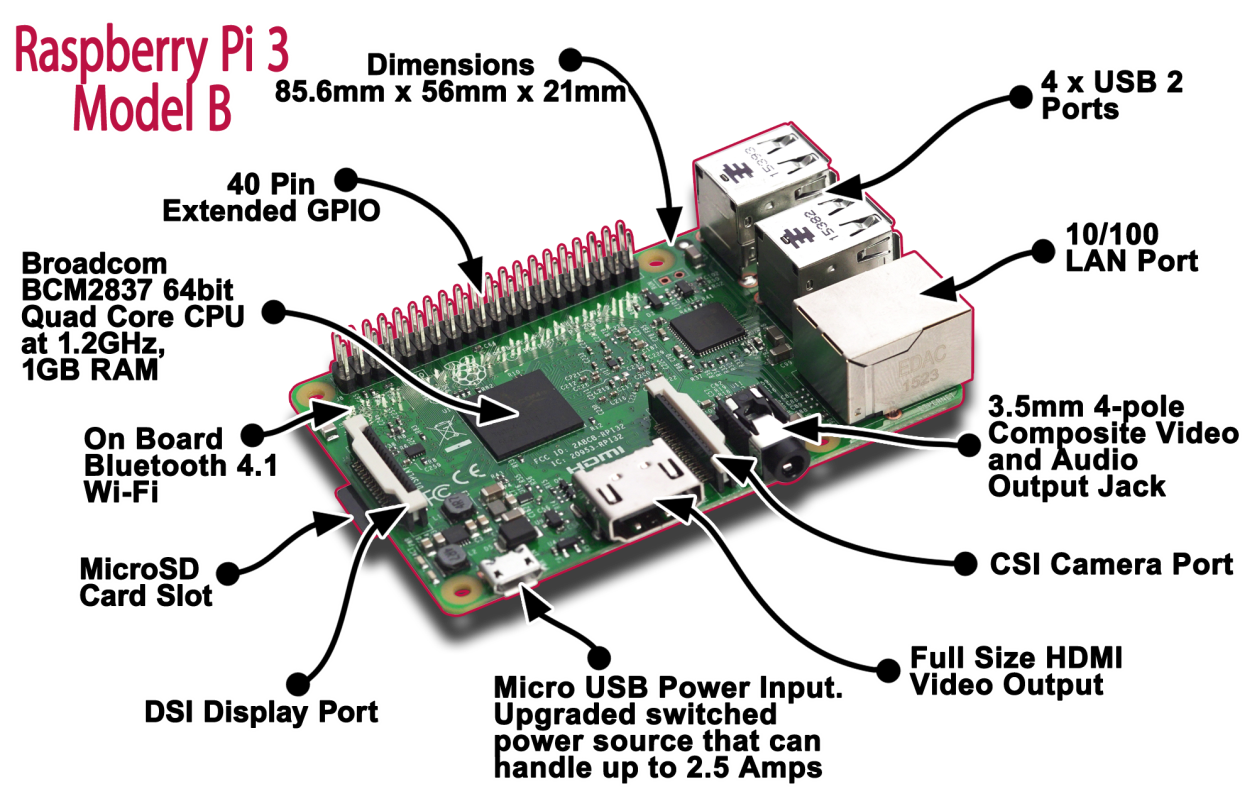


Рисунок 1.1 – Внешний вид отладочной платы Raspberry Pi 3

* + 1. **GPIO на Raspberry Pi 3**

Raspberry Pi 3 имеет 40-пиновую рейку GPIO. Однако называть все 40-пинов выводами GPIO некорректно, так как 12 из них представляют из себя линии питания 3.3 В, 5 В и общий вывод GND (земля):

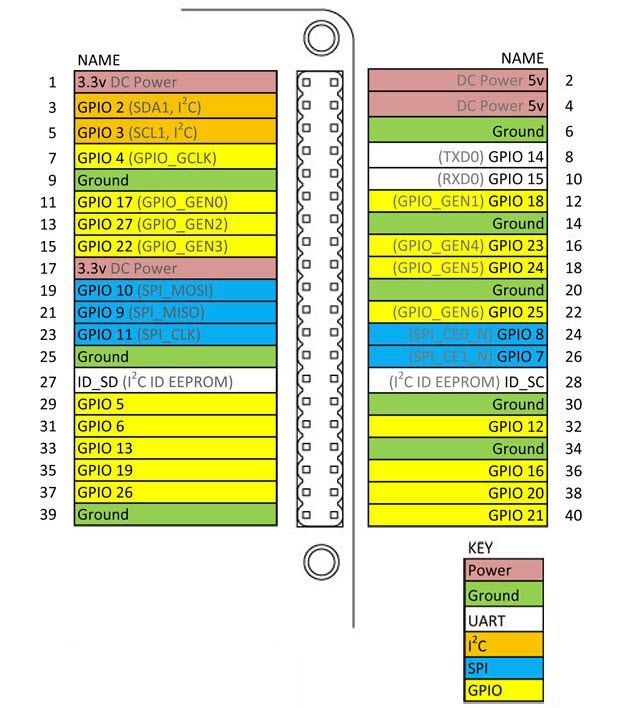


Рисунок 1.2 – Расположение выводов GPIO отладочной платы Raspberry Pi 3

Также 27 (BCM 0) и 28 (BCM 1) выводы используются для конфигурации EEPROM Raspberry Pi для работы с HAT-устройствами (Hardware Attached on Top — устройства поверхностного монтажа, по сути — обычные платы расширения) и использование этих выводов крайне не рекомендуется. Тем не менее, они являются полноценными GPIO-выводами.

Фактически получается, что GPIO-выводов не 40, а 28.

* + 1. **Нумерация выводов GPIO на Raspberry Pi 3**

Существует несколько вариантов нумерации выводов Raspberry Pi: Board (физическая нумерация по порядку) и BCM (нумерация из чипа). Также некоторые библиотеки, например, WiringPi, используют свою нумерацию:

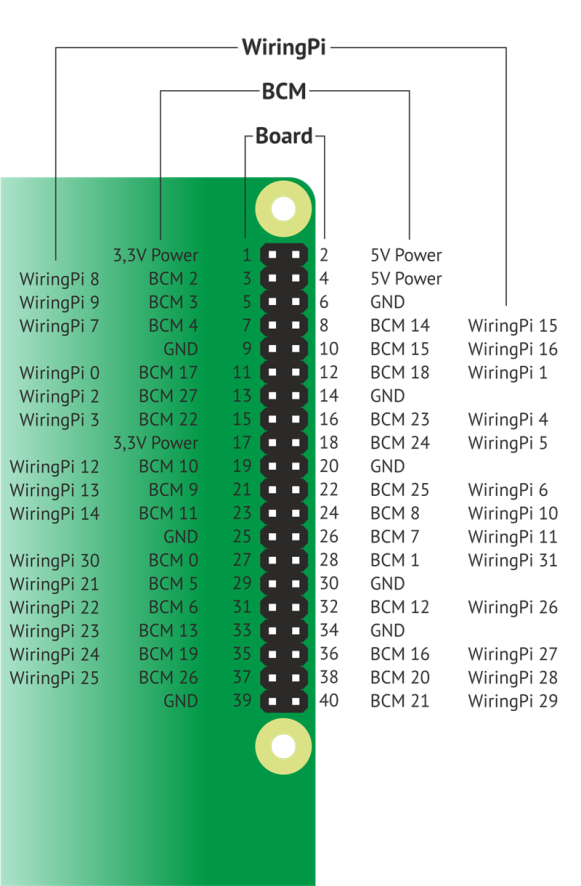


Рисунок 1.3 – Системы нумерации выводов GPIO отладочной платы Raspberry Pi 3

Во избежание проблем, при написании программ, необходимо явно указывать какой режим нумерации будет использован

* + 1. **Функциональные возможности GPIO-выводов Raspberry Pi 3**

Каждый из 28 выводов может быть установлен в режим цифрового выхода *OUTPUT*, и в режим цифрового входа *INPUT*.

Максимальный выходной ток каждого вывода не должен превышать **16 мА**. Суммарный выходной ток всех выводов не должен превышать **50 мА**. 5-вольтовые линии могут давать больший ток, который остается после питания Raspberry Pi 3 и других периферийных устройств (клавиатуры, мыши) — до **500 мА**.

По умолчанию все выводы (кроме BCM14 и BCM15) находятся в режиме *INPUT*, причем выводы BCM0-BCM8 и BCM15 подтянуты к единице подтягивающими резисторами (pullup). Именно по этой причине мультиметр покажет напряжение на этих выводах. Остальные пины стянуты к нулю.

Каждый из 28 пинов снабжен подтягивающим (pullup) и стягивающим (pulldown) резистором, благодаря чему, в режиме *INPUT* может быть подтянут к логической единице, либо стянут к нулю.

Номиналы сопротивлений не постоянны и равны:

* для подтягивающего резистора 50 КОм — 65 КОм
* для стягивающего резистора 50 КОм — 60 КОм

Номинал сопротивления подтягивающего/стягивающего резистора для выводов BCM2 и BCM3 — 1.8 КОм.

Каждый из 28 выводов в режиме *INPUT* может генерировать прерывания — по спаду, по фронту, по единице, по нулю, по изменению сигнала, а также в асинхронном режиме по фронту и по спаду:

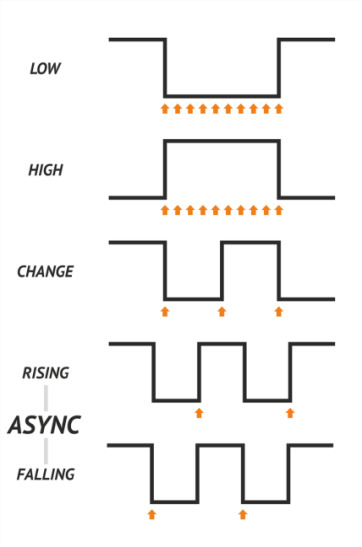


Рисунок 1.4 – Разновидности прерываний

Также все выводы в режиме *INPUT* имеют входную фильтрацию на триггере Шмитта (преобразовывают аналоговый сигнал в цифровой с резкими переходами между состояниями):

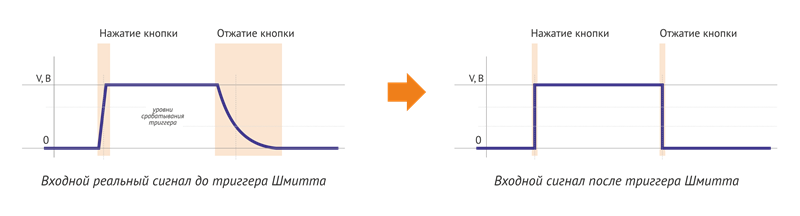


Рисунок 1.5 – Форма входного сигнала

Raspberry Pi 3 не имеет аналоговых входов/выходов. Для реализации подобной функциональности нужно использовать внешние АЦП/ЦАП, например, АЦП ADS1115 (I2C) или АЦП MCP3008.

* + 1. **Регистры для работы с GPIO Raspberry Pi 3**

Режим, в котором работает каждый отдельный разряд порта GPIO, управляется полностью программным способом.

Процессор BCM2835 имеет 41 32-разрядный регистр, которые полностью определяют режим и состояние портов GPIO. В частности, для установки единичного значения на выводе, запрограммированном на работу как выход, необходимо записать единичный бит в соответствующий разряд одного из двух регистров установки битов GPIO Pin Output Set Registers (GPSETn). Чтобы установить выход в ноль, следует выставить единичный бит в регистрах сброса битов GPIO Pin Output Clear Registers (GPCLRn). Такая схема позволяет независимо устанавливать и сбрасывать любой бит GPIO без необходимости чтения текущего состояния выводов.

Аналогично, когда разряды GPIO работают на чтение, то узнать уровень входного сигнала можно, прочитав значение одного из двух портов GPIO Pin Level Registers (GPLEVn), каждый бит которого отображает текущее состояние входного разряда.

Регистры, отвечающие за работу с GPIO, расположены по адресам 0x7E200000—0x7E2000B0, которые отображаются на физическую память с адресами, начинающимися с 0x20200000.

* + 1. **Программирование портов ввода-вывода**

Программировать поведение GPIO можно на большом количестве различных языков — Pascal, Ruby, Perl, Java (Pi4J), C, C++, C#, WiringPi, Basic и т.д. Примеры большинства из них можно изучить на странице RPi GPIO Code Samples.

Также можно записать номер GPIO в файл *./export* в подкаталоге */sys/class/gpio/*, и система создаст файл со структурой GPIO согласно номеру входа.

Создание файла доступа GPIO:

*echo 12 > /sys/class/gpio/export*

Настройка направления передачи вывода (вход/выход):

*echo out > /sys/class/gpio/gpio12/direction*

Запись значения для включения светодиода с помощью GPIO12:

*echo 1 > /sys/class/gpio/gpio12/value*

Здесь и далее будем использовать библиотеку на языке C bcm2835 версии 1.52, которая использует нумерацию портов Board. Код программы на C можно также писать в текстовом редакторе командной строки *nano* или загружать исходные файлы по средствам SFTP.

Для проверки возможностей библиотеки воспользуемся следующим кодом в mian.c:

*#include <bcm2835.h> // подключение библиотеки bcm2835*

*#define PIN RPI\_V2\_GPIO\_P1\_12 // определение используемого вывода GPIO*

*int main(int argc, char \*\*argv)*

*{*

*if (!bcm2835\_init()) // инициализация GPIO*

*return 1;*

*bcm2835\_gpio\_fsel(PIN, BCM2835\_GPIO\_FSEL\_OUTP); // установка порта на вывод*

*bcm2835\_gpio\_write(PIN, HIGH); // подача на вывод высокого уровня (+3,3 В)*

*bcm2835\_delay(1000); // ожидание 1000 мс*

*bcm2835\_gpio\_write(PIN, LOW); // подача на вывод низкого уровня (GND)*

*bcm2835\_close(); // завершение работы с GPIO*

*return 0;*

*}*

Для получения исполняемого файла не необходимо воспользоваться компилятором GCC с помощью следующей команды:

*gcc -o main -l rt main.c -l bcm2835*

После выполнения данной команды будет получен исполняемой файл main, для запуска которого необходимо выполнить команду:

*./main*

В результат выполнения программы светодиод, подключенный к 12 выводу GPIO Raspberry Pi 3, включится на 1 секунду и погаснет.

Для выполнения примера необходимо выполнить следующие действия:

* подключиться к Wi-Fi сети Raspberry Pi (SSID: *RPi3-AP*; пароль: *raspberry*);
* подключиться по SSH и, при необходимости, по SFTP к Raspberry Pi по адресу 192.168.1.1 с помощью PuTTY и WinSCP, используя учётную запись пользователя *root/root*.
* перейти в каталог */home/pi/*,используя команду *cd /home/pi/*;
* создать каталог, в названии которого указать свою фамилию и группу, с помощью команды *mkdir* (например: *mkdir ivanov\_10vs*);
* перейти в созданный каталог, использую команду *cd* (например: *cd ivanov\_10vs*);
* создать файл *min.c* с помощью команды *nano main.c*;
* скопировать код с примера и сохранить файл, используя последовательность клавиш *Ctrl+X – Y – Enter*;
* скомпилировать получившийся файл с исходным кодом и запустить полученный исполняемый файл, используя команды, описанные выше.

С учёт описанных выше электрических параметров портов GPIO на Raspberry Pi 3, для наглядной работы примера необходимо подключить светодиод по следующей схеме:

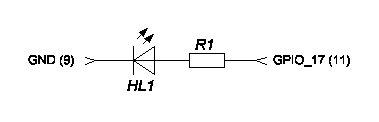


Рисунок 1.6 – Схема подключения светодиода

При этом номинал резистора R1 должен быть в пределе от 220 до 330 Ом.

* 1. **Задание**

Модифицировать код примера так, чтобы последовательность включения и выключения светодиода и количество повторений последовательности соответствовали варианту. Последовательность может состоять из коротких и длинных включений (светодиод горит), 300 мс и 1000 мс соответственно, обозначаемых ***Квкл*** и ***Двкл***, а также коротких и длинных выключений (светодиод не горит), с такими же временными интервалами, обозначаемых соответственно ***Квыкл*** и ***Двыкл***.

|  |  |  |
| --- | --- | --- |
| № варианта | Последовательность | Количество повторений |
| 1 | ***Двкл – Квыкл – Квкл – Двыкл – Двкл – Квыкл*** | 3 |
| 2 | ***Квкл – Квыкл – Двкл – Двыкл*** | 4 |
| 3 | ***Двкл – Двыкл*** | 5 |
| 4 | ***Квкл – Двыкл – Квкл – Двыкл – Двкл – Квыкл*** | 3 |
| 5 | ***Квкл – Двыкл – Двкл – Квыкл*** | 4 |
| 6 | ***Квкл – Двыкл*** | 5 |
| 7 | ***Двкл – Квыкл – Двкл – Квыкл – Квкл – Двыкл*** | 3 |
| 8 | ***Двкл – Квыкл – Квкл – Двыкл*** | 4 |
| 9 | ***Двкл – Квыкл*** | 5 |
| 10 | ***Квкл – Квыкл – Квкл – Двыкл – Двкл – Двыкл*** | 3 |
| 11 | ***Двкл – Двыкл – Квкл – Квыкл*** | 4 |
| 12 | ***Квкл – Квыкл*** | 5 |

**Общее задание:**

1. Реализовать сортировку с выводом наименьшего числа используя интерпретатор *bash.*
2. Полученное число вывести в *std::cout* средствами *bash.*
3. Перенаправить *std::cout* в *std::cin* приложения на *C.*
4. В приложении необходимо принять минимальное число из *std::cin* и преобразовать его по формуле:

*Y = |X|* ***mod*** *num\_port*

где *Х* - число, принятое в std::cin, *mod* - операция получения остатка от деления нацело, *num\_port* - общее количество портов доступных для подключения светодиода.

1. Подать сигнал ***HIGH*** на номер порта, соответствующий числу *Y*.
   1. **Контрольные вопросы**
2. Дайте определение понятию «Интерфейс ввода/вывода общего назначения».
3. Перечислите режимы работы прерываний.
4. Какие электрические характеристики имеют GPIO Raspberry Pi 3?
5. Какими способами выполняется управление GPIO Raspberry Pi 3?
6. Опишите структуру регистров GPIO Raspberry Pi 3.
7. Назовите номиналы подтягивающих и стягивающих резисторов на выводах GPIO Raspberry Pi 3.
8. Охарактеризуйте работу прерываний в различных режимах.
9. Назовите системные файлы, используемые для управления GPIO Raspberry Pi 3.
10. Перечислите регистры управления GPIO Raspberry Pi 3.
11. Перечислите и охарактеризуйте системы нумерации выводов Raspberry Pi 3.
    1. **Содержание отчёта**
12. Ф.И.О., группа, название лабораторной работы.
13. Цель работы.
14. Краткие теоретические сведения.
15. Описание проделанной работы.
16. Результаты выполнения лабораторной работы.
17. Выводы.

Отчет и исходные коды упаковать в архив с названием по следующему шаблону:

AMP-Lab1-«группа, аббревиатуру на латинице»-«Фамилия на латинице».

Пример: **AMP-Lab1-11VS-Ivanov.zip**

# **Лабораторная работа №2 ИСПОЛЬЗОВАНИЕ ШИРОТНО-ИМПУЛЬСНОЙ МОДУЛЯЦИИ НА ОДНОПЛАТНОМ КОМПЬЮТЕРЕ RASPBERRY PI 3**

**Цель:** изучить принципы формирования ШИМ-сигнала. Освоить использование интерфейс ввода/вывода общего назначения на Raspberry Pi 3 в режиме ШИМ.

* 1. **Теоретическая часть**
     1. **Широтно-импульсная модуляция**

ШИМ или PWM (широтно-импульсная модуляция, по-английски pulse-width modulation) – это способ управления подачей мощности к нагрузке. Управление заключается в изменении длительности импульса при постоянной частоте следования импульсов. Широтно-импульсная модуляция бывает аналоговой, цифровой, двоичной и троичной.

Широтно-импульсная модуляция позволяет регулировать яркость подсветки жидкокристаллических дисплеев сотовых телефонов, смартфонов, ноутбуков. ШИМ реализована в сварочных аппаратах, в автомобильных инверторах, в зарядных устройствах и т. д. Любое зарядное устройство сегодня использует при своей работе ШИМ.

Описывая механизм работы ШИМ, можно привести следующую аналогию с механикой. Если при помощи двигателя вращать тяжелый маховик, то поскольку двигатель может быть либо включен, либо выключен, то и маховик будет либо раскручиваться и продолжать вращаться, либо станет останавливаться из-за трения, когда двигатель выключен.

При этом, если двигатель включать на несколько секунд в минуту, то вращение маховика будет поддерживаться, благодаря инерции, на некоторой скорости. И чем дольше продолжительность включения двигателя, тем до более высокой скорости раскрутится маховик. Так и с ШИМ, на выход приходит сигнал включений и выключений (0 и 1), и в результате достигается среднее значение. Проинтегрировав напряжение импульсов по времени, получим площадь под импульсами, и эффект на рабочем органе будет тождественен работе при среднем значении напряжения.

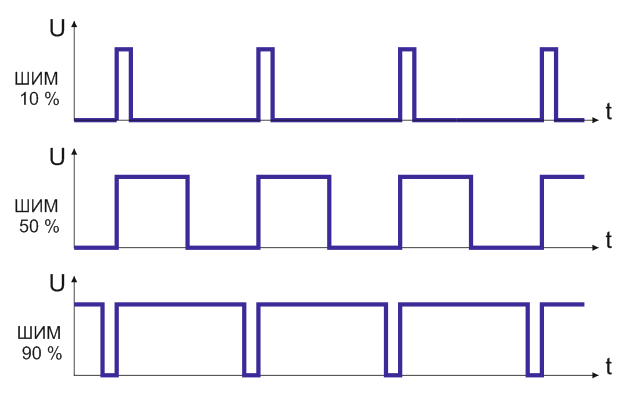


Рисунок 2.1 – Сигнал широтно-импульсной модуляции

Отношение полной длительности периода импульса ко времени включения (положительной части импульса) называется скважностью импульса. Так, если время включения составляет 10 мкс, а период длится 100 мкс, то при частоте в 10 кГц, скважность будет равна 10, и пишут, что S = 10. Величина обратная скважности называется коэффициентом заполнения импульса, по-английски Duty cycle, или сокращенно DC.

Так, для приведенного примера DC = 0.1, поскольку 10/100 = 0.1. При широтно-импульсной модуляции, регулируя скважность импульса, то есть варьируя DC, добиваются требуемого среднего значения на выходе электронного или другого электротехнического устройства, например, двигателя.

* + 1. **Программное и аппаратное формирование ШИМ-сигнала**

Самый простой способ получить ШИМ на выходе GPIO - это программный генератор импульсов. Данный метод хорош из-за отсутствия необходимости установки каких-либо драйверов и разного ПО в ОС. Достаточно просто воспользоваться циклом, который будет каждые N миллисекунд выдавать на нужный GPIO импульс требуемой ширины.

Для примера, рассмотрим программу, которая заставляет светодиод плавно разжигаться и гаснуть.

*#include <bcm2835.h>*

*#define PIN RPI\_V2\_GPIO\_P1\_03*

*int main(int argc, char \*\*argv)*

*{*

*if (!bcm2835\_init())*

*return 1;*

*bcm2835\_gpio\_fsel(PIN, BCM2835\_GPIO\_FSEL\_OUTP); // установка порта на вывод*

*unsigned int t\_on, t\_off; // продолжительность вкл. и выкл. состояния*

*int d = 100, i, j, flag=0;*

*// d- коэффициент заполнения в процентах*

*//i и j, вспомогательные переменные для организации циклов*

*//flag- если = 0 светодиод затухает, если = 1 разгорается*

*int a=10; // количество полных рабочих циклов*

*while (a)*

*{*

*for (j=100; j!=0; j--) // изменяем коэффициент заполнения от 100% до 0%*

*{*

*t\_on=50\*d; //находим время включения*

*t\_off=50\*(100-d); //находим время выключения*

*// если светодиод затухает, уменьшаем коэффициент заполнения*

*if (flag==0) d=d-1;*

*// если светодиод разгорается, увеличиваем коэффициент заполнения*

*if (flag==1) d=d+1;*

*// передаём 10 импульсов на светодиод с рассчитанными параметрами*

*for (i=10; i!=0; i--)*

*{*

*bcm2835\_gpio\_write(PIN, LOW);*

*delayMicroseconds(t\_on);*

*bcm2835\_gpio\_write(PIN, HIGH);*

*delayMicroseconds(t\_off);*

*}*

*// если светодиод выключен, начинаем его включать*

*if (d==0) flag=1;*

*// если светодиод достиг максимума свечения, начинаем его гасить*

*if (d==100) flag=0;*

*}*

*a--;*

*}*

*return (!bcm2835\_close ()); // Выход из программы*

*}*

Минус программных ШИМ в том, что этот самый программный генератор импульсов потребляет вычислительные ресурсы. Другими словами, цикл генерации будет соперничать с прочим кодом, от чего будет страдать и стабильность ШИМ и стабильность выполнения всего остального. Нестабильность ШИМ может выражаться, например, в дергании сервоприводов.

ШИМ настолько часто используется в различных приложениях, что производители процессорного оборудования часто встраивают ШИМ-контроллер непосредственно в процессор. Т.е. процессору задаются параметры требуемого сигнала, а процессор уже сам, без посторонней помощи выдаёт нужный сигнал. При этом, нисколько не тратя программных ресурсов на генерацию этого сигнала. Bcm2837 тоже имеет встроенный аппаратный ШИМ. И этот ШИМ является дополнительной функцией портов GPIO 12, 32, 33, 35. Чтобы воспользоваться аппаратными ШИМ, необходимо установить соответствующий порт в режим дополнительных функций и задать процессору параметры ШИМ-сигнала.

Соответствие порта GPIO и режима дополнительных функций следующее:

* GPIO 12 (PWM channel 0) – режим *Alt5*;
* GPIO 32 (PWM channel 0) – режим *Alt0*;
* GPIO 33 (PWM channel 1) – режим *Alt0*;
* GPIO 35 (PWM channel 1) – режим *Alt5*.

Подробная информация по функциям выводов GPIO приведена на рисунке ниже, а также доступна по ссылке <https://pinout.xyz/>.

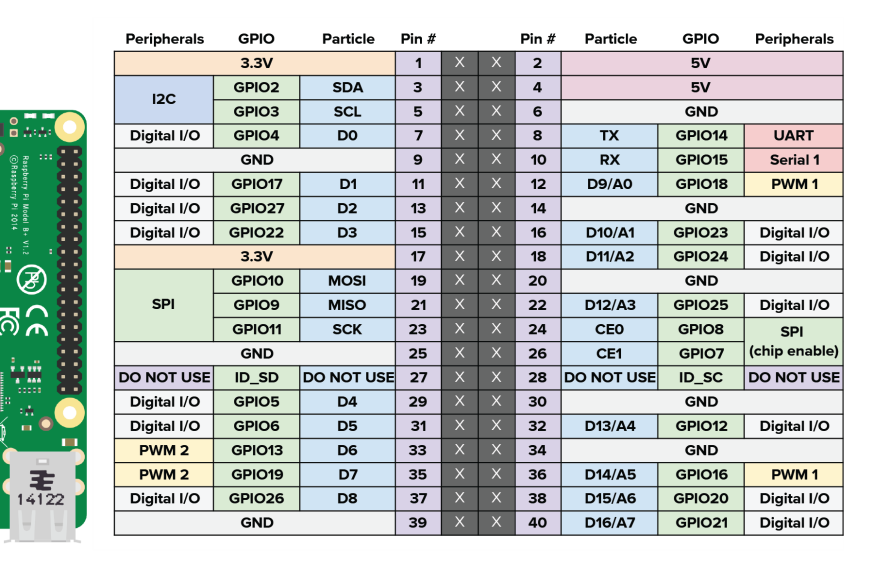


Рисунок 2.2 – Выводы GPIO с поддержкой ШИМ

Для использования аппаратного модуля ШИМ в Raspberry Pi 3 воспользуемся следующим примером:

*#include <bcm2835.h>*

*#define PIN RPI\_GPIO\_P1\_12*

*#define PWM\_CHANNEL 0*

*#define RANGE 1024*

*int main(int argc, char \*\*argv)*

*{*

*if (!bcm2835\_init())*

*return 1;*

*bcm2835\_gpio\_fsel(PIN, BCM2835\_GPIO\_FSEL\_ALT5);*

*bcm2835\_pwm\_set\_clock(BCM2835\_PWM\_CLOCK\_DIVIDER\_16);*

*bcm2835\_pwm\_set\_mode(PWM\_CHANNEL, 1, 1);*

*bcm2835\_pwm\_set\_range(PWM\_CHANNEL, RANGE);*

*int direction = 1;*

*int data = 1;*

*int count = 0;*

*while (count < 5)*

*{*

*if (data == 1)*

*direction = 1;*

*else if (data == RANGE-1)*

*{*

*count++;*

*direction = -1;*

*}*

*data += direction;*

*bcm2835\_pwm\_set\_data(PWM\_CHANNEL, data);*

*bcm2835\_delay(1);*

*}*

*bcm2835\_close();*

*return 0;*

*}*

При выполнении данного примера светодиод, подключённый к GPIO 12 плавно загорится и потухнет 5 раз.

* + 1. **Использование параметров командной строки**

При запуске программы из командной строки, ей можно передавать дополнительные параметры в текстовом виде. В программе эти параметры из командной строки можно получить через аргументы функции *main* при использовании её в следующей форме

*void main(int argc, char \*\*argv) { ... }*

Первым аргументом *argc* является число переданных функции параметров. Второй аргумент *argv* – массив строк – переданные параметры. Так как параметры у функции могут быть любыми, то они передаются как строки, и уже сама программа должна их разбирать и приводить к нужному типу.

Первым аргументом (*argv[0]*) всегда является имя программы. При этом имя выводится в зависимости от того, откуда была запущена программа.

Для примера использования параметров командной строки будем передавать состояние вывода GPIO, которое необходимо установить.

*#include <bcm2835.h>*

*#include <stdio.h>*

*#include <stdlib.h>*

*#define PIN RPI\_V2\_GPIO\_P1\_11*

*int main(int argc, char \*\*argv)*

*{*

*int state;*

*if (argc > 1)*

*{*

*if (!bcm2835\_init())*

*{*

*return 1;*

*}*

*bcm2835\_gpio\_fsel(PIN, BCM2835\_GPIO\_FSEL\_OUTP);*

*state = atoi(*argv[1]*);*

*bcm2835\_gpio\_write(PIN, state);*

*bcm2835\_close();*

*} else*

*{*

*printf("No arguments!\n");*

*}*

*return 0;*

*}*

В результате выполнения программы на выходе GPIO 11 будет установлен низкий или высокий уровень в зависимости от того, передадим ли мы 0 или 1 в качестве аргумента при запуске.

* 1. **Задание**

Модифицировать пример так, чтобы в качестве параметров командной строки программа принимала два аргумента: команда (установка уровня или скважности ШИМ) и значение (вкл./выкл. или значение скважности). Без указания параметров программа должна выполнить действие, согласно индивидуальному варианту.

|  |  |  |
| --- | --- | --- |
| № варианта | Используемый GPIO ввод | Заполнение ШИМ |
| 1 | 12 | 20% |
| 2 | 32 | 20% |
| 3 | 33 | 20% |
| 4 | 35 | 20% |
| 5 | 12 | 50% |
| 6 | 32 | 50% |
| 7 | 33 | 50% |
| 8 | 35 | 50% |
| 9 | 12 | 80% |
| 10 | 32 | 80% |
| 11 | 33 | 80% |
| 12 | 35 | 80% |

* 1. **Контрольные вопросы**

1. Дайте определение понятия ШИМ.
2. Как связана скважность с коэффициентом заполнения импульса?
3. Сколько каналов аппаратного ШИМ поддерживает Raspberry Pi 3 и на каких GPIO?
4. Перечислите преимущества и недостатки аппаратного и программного ШИМ.
5. Опишите механизм передачи параметров командной строки.
6. Что называется коэффициентом заполнения импульса?
7. Что называется скважностью ШИМ?
8. В каком виде передаются параметры командной строки при запуске программы?
9. Какие способы реализации ШИМ есть на Raspberry Pi 3?
10. Для чего используется ШИМ?
    1. **Содержание отчёта**
11. Ф.И.О., группа, название лабораторной работы.
12. Цель работы.
13. Краткие теоретические сведения.
14. Описание проделанной работы.
15. Результаты выполнения лабораторной работы.
16. Выводы.

Отчет и исходные коды упаковать в архив с названием по следующему шаблону:

AMP-Lab2-«группа, аббревиатуру на латинице»-«Фамилия на латинице».

Пример: **AMP-Lab2-11VS-Ivanov.zip**

# **Лабораторная работа №3 КОНФИГУРИРОВАНИЕ ВЕБ-СЕРВЕРА НА ОДНОПЛАТНОМ КОМПЬЮТЕРЕ RASPBERRY PI 3**

**Цель:** изучить необходимое программное обеспечение для работы веб-сервера. Научиться конфигурировать веб-сервер на одноплатном компьютере Raspberry Pi 3.

* 1. **Теоретическая часть**
     1. **Операционная система Raspbian**

Основой одноплатного компьютера Raspberry Pi 3 является процессор архитектуры ARM, поэтому большинство используемых на нем операционных систем построены на базе Linux. С 2015 года дистрибутив Raspbian был официально предоставлен Raspberry Pi Foundation в качестве основной операционной системы для семейства одноплатных компьютеров Raspberry Pi.

Это дистрибутив Linux основан на ARM версии Debian 9 Stretch и оптимизирован под используемые аппаратные компоненты. Набор приложений и утилит, собранный в этом дистрибутиве, является базовым и предназначен, в основном, для ознакомительного изучения возможностей компьютера.

Так как Raspberry Pi 3 работает под управление \*-unix подобной ОС, то это позволяет запускать на этом одноплатном компьютере различные приложения для данной платформы, в том числе и ПО для организации веб-сервера.

* + 1. **Веб-сервер**

В основе функционирования веб-приложений лежит такое понятие как веб-сервер. Веб-сервер – это программа, которая принимает входящие HTTP-запросы, обрабатывает эти запросы, генерирует HTTP-ответ и отправляет его клиенту. Общий алгоритм работы веб-сервера можно представить следующим образом (серым цветом помечены действия, которые обрабатываются веб-сервером).

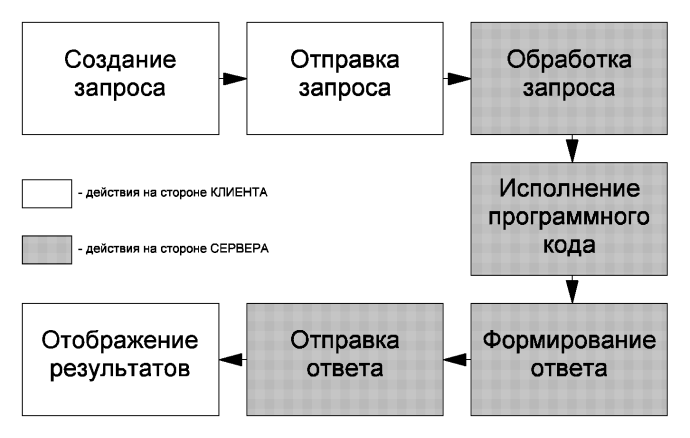


Рисунок 3.1 – Общий алгоритм работы веб-сервера

После того, как пользователь обратился к определенному ресурсу по протоколу HTTP, клиент (обычно браузер) формирует HTTP-запрос к веб-серверу. Обычно указывается символическое имя сервера – в этом случае браузер предварительно преобразует это имя в IP-адрес при помощи сервисов DNS. После этого по протоколу HTTP на веб-сервер отправляется сформированное HTTP-сообщение. В этом сообщении браузер указывает какой ресурс необходимо загрузить и всю дополнительную информацию. Задача веб-сервера – прослушивать определенный TCP-порт (обычно порт 80) и принимать все входящие HTTP-сообщения. Если входящие данные не соответствуют формату сообщения HTTP, то такой запрос игнорируется, а клиенту возвращается сообщение об ошибке.

Самый простой сценарий работы веб-сервера заключается в получении HTTP-запроса, его обработки, считывания нужного файла с жесткого диска, формирование HTTP-ответа и отправки его клиенту. Подобный сценарий является самым простым, однако, в реальности встречается все реже. Дело в том, что при подобном подходе, содержимое, которое передается клиенту, является статическим (т.е. не изменяется от запроса к запросу). Однако если требуется построить веб-приложение, то содержимое HTML-страницы, которое передается клиенту должно изменяться от различных внешних условий (параметров запроса, содержимого базы данных, времени обработки запроса, типа пользователя и т.д.). В этом случае требуется запускать внешний (по отношению к веб-серверу) программный код, реализующий логику веб-приложения. Этот код должен содержаться отдельно от программного кода самого веб-сервера, поскольку код приложения будет различным от одного приложения к другому, а веб-сервер будет один и тот же. Схема взаимодействия веб-сервера с внешним ПО показана ниже.

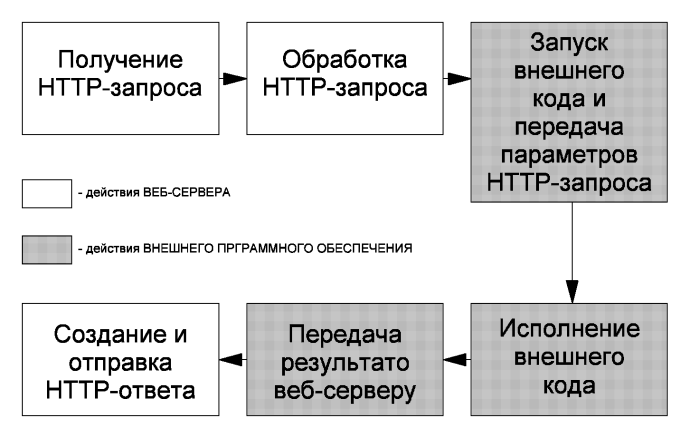


Рисунок 3.2 – Схема взаимодействия веб-сервера с внешним ПО

Исторически сложилось так, что существует два главных типов интерфейс взаимодействия внешнего приложения и веб-сервера – *CGI* и *ISAPI*.

***CGI*** (Common Gateway Interface) – наиболее ранний способ взаимодействия веб-сервера и веб-приложения. Основная идея, которая лежит в основе CGI заключается в том, что при поступлении очередного HTTP-запроса, веб-сервер инициирует создание нового процесса и передает ему все необходимые данные HTTP-запроса. После того, как этот процесс отработает, он завершается, передав при этом результат обратно веб-серверу. Поскольку веб-сервер и приложение – это разные процессы с точки зрения операционной системы, то для обмена информации между ними используются средства межпроцессного взаимодействия (IPC) – зачастую это переменные окружения, именованные каналы и т.д. Основным преимуществом CGI является то, что процесс веб-сервера и приложения изолированы друг от друга и в случае неполадок в веб-приложении, завершится с ошибкой именно процесс приложения, при этом процесс самого веб-сервера будет продолжать функционировать.

С другой стороны, необходимость создания каждый раз нового процесса влечет за собой дополнительные накладные расходы на создание процесса (создание процесса – дорогостоящая операция с точки зрения операционной системы) и передачи данных через границы процессов. Этот факт является серьезным недостатком и оказывает существенное влияние на масштабируемость веб-приложения (возможность обрабатывать большее количество поступающих запросов).

***ISAPI*** (Internet Server API) – альтернативный способ взаимодействия веб-сервера и веб-приложения. В отличии от CGI, при взаимодействии в рамках интерфейса ISAPI, при поступлении очередного запроса, веб-сервер инициирует создание нового потока в рамках основного процесса, в котором работает веб-сервер. Поскольку с точки зрения операционной системы создание потока – это менее дорогостоящая операция, чем создание процесса, то такие приложения на практике оказываются более масштабируемыми. Кроме того, упрощается взаимодействие веб-сервера и веб-приложения, поскольку в этом случае используется единое адресное пространство в рамках операционной системы (поскольку весь код работает в одном и том же процессе). Однако, в случае серьезных неполадок в веб-приложении, которое взаимодействует с веб-сервером в рамках ISAPI, веб-сервер также потенциально подвергается риску быть завершенным. Поскольку веб-сервер и веб-приложение работают в одном и том же процессе, это действительно так. Поэтому разработчикам программного кода веб-сервера, поддерживающего ISAPI следует уделить этому вопросу особое внимание.

На сегодняшний день наиболее распространенным способом взаимодействия веб-сервера и веб-приложения является интерфейс ISAPI, поскольку обеспечивает наиболее оптимальные показатели по накладным расходам и масштабируемости.

Кроме приведенных функций и механизмов веб-сервера, в его функции зачастую входят и сопутствующие дополнительные задачи. К этим задачам относится аутентфикация и авторизация пользователя, ведение серверного лога (для отладки работы веб-сервера), поддержка нескольких веб-сайтов на одном сервере (виртуальный хостинг), поддержка безопасных подключений по протоколу HTTPS и др. Эти функции в каждом конкретном случае зависят от реализации веб-сервера.

* + 1. **Программное обеспечение для работы веб-сервера**

ПО, позволяющее принимать запросы пользователей, обрабатывать их и отправлять пользователям результаты обработки (html-страницы и другие файлы). Наиболее популярный веб-сервер – это Apache, чуть менее распространены Nginx и IIS. Иногда с целью увеличения производительности устанавливается два веб-сервера: быстрый Nginx, который отдаёт пользователям статическое содержимое (физически существующие на сервере документы, не требующие обработки перед отправкой), а остальные запросы переадресовывает мощному Apache, который занимается генерацией динамических документов. Существуют и другие производительные связки (Nginx + FastCGI, например), о рациональности использования той или иной реализации принимают решение разработчики приложения и администраторы серверов.

* + 1. **Конфигурирование веб-сервера Apache**

Для выполнения лабораторной работы будем использовать веб-сервер с открытым исходным кодом Apache.

Все настройки содержатся в папке */etc/apache/*:

* Файл */etc/apache2/apache2.conf* отвечает за основные настройки
* */etc/apache2/conf-available/\** — дополнительные настройки веб-сервера
* */etc/apache2/mods-available/\** — настройки модулей
* */etc/apache2/sites-available/\** — настойки виртуальных хостов
* */etc/apache2/ports.conf* — порты, на которых работает apache
* */etc/apache2/envvars* — переменные окружения

Существуют два каталога для *conf*, *mods* и *site*. Это *available* и *enabled*. При включении модуля или хоста создается символическая ссылка из каталога *available* (доступно) в каталог *enable* (включено). Поэтому настройки лучше выполнять именно в каталогах *available*.

Рассмотрим главный файл конфигурации, выполнив следующую команду:

*nano /eta/apache2/apache2.conf*

Основные параметры для настройки веб-сервера:

***Timeout*** – указывает как долго сервер будет пытаться продолжить прерванную передачу или прием данных. 160 секунд будет оптимально для большинства задач.

***KeepAlive On*** – позволяет передавать несколько файлов, за одно соединение, например, не только саму html страницу, но и картинки и css файлы.

***MaxKeepAliveRequests 100*** – максимальное количество запросов за одно соединение.

***KeepAliveTimeout 5*** – таймаут соединения, обычно для загрузки страницы достаточно 5-10 секунд, так что больше ставить не рекомендуется, но и разрывать соединение раньше чем загрузились все данные тоже не нужно.

***User, Group*** – пользователь и группа, от имени которых будет работать программа.

***HostnameLookups*** – записывать в логи вместо ip адресов доменные имена, можно отключить, чтобы ускорить работу.

***LogLevel*** – уровень логирования ошибок. По умолчанию используется warn, но чтобы логи заполнялись медленнее достаточно включить error

***Include*** – все директивы include отвечают за подключение рассмотренных выше конфигурационных файлов.

Директивы *Directory* отвечают за настройку прав доступа к той или иной директории в файловой системе. Синтаксис здесь следующий:

*<Directory /адрес/в/файловой/системе/>*

*Параметр значение*

*</Directory>*

Здесь доступны такие основные опции:

***AllowOverride*** — указывает нужно ли читать *.htaccess* файлы из этой директории, это такие же файлы настроек и таким же синтаксисом. *All* — разрешать все, *None* — не читать эти файлы.

***DocumentRoot*** — устанавливает из какой папки нужно брать документы для отображения пользователю.

***Options*** — указывает какие особенности веб-сервера нужно разрешить в этой папке. Например, *All* — разрешить все, *FollowSymLinks* — переходить по символическим ссылкам, *Indexes* — отображать содержимое каталога, если нет файла индекса.

***Require*** — устанавливает, какие пользователи имеют доступ к этому каталогу. *Require all denied* — всем запретить, *Require all granted* — всем разрешить. Можно использовать вместо *all* директиву *user* или *group* чтобы явно указать пользователя.

***Order*** — позволяет управлять доступом к директории. Принимает два значения *Allow,Deny* — разрешить для всех, кроме указанных или *Deny,Allow* — запретить для всех, кроме указанных. Можно запретить доступ к директории для всех: *Deny from all*, а затем разрешить только для приложения от *example.com*: *Allow from example.com*.

При выполнении лабораторной работы эти директивы не используются, поскольку нас устраивают значения по умолчанию, но вот в файлах *.htaccess* они могут быть очень полезны.

В файле */etc/apache2/ports.conf* только одна директива, *Listen*, которая указывает программе на каком порту нужно работать.

В файле */etc/apache2/envvars* указаны переменные, которые можно использовать в других конфигурационных файлах.

* + 1. **Настройка виртуальных хостов Apache**

Apache может поддерживать сотни сайтов на одном компьютере и выдавать для каждого из них правильное содержимое. Для этого используются виртуальные хосты. Сервер определяет, к какому домену приходит запрос и отдает нужное содержимое из папки этого домена.

Настройки хостов Apache расположены в каталоге */etc/apache2/hosts-available/*. Для создания нового хоста достаточно создать файл с любым именем (лучше кончено с именем хоста) и заполнить его нужными данными. Обернуть все эти параметры нужно в директиву *VirtualHost*. Кроме рассмотренных параметров здесь будут использоваться такие:

* ServerName — основное имя домена
* ServerAlias — дополнительное имя, по которому будет доступен сайт
* ServerAdmin — электронная почта администратора
* DocumentRoot — папка с документами для этого домена

Например:

*nano /etc/apache2/sites-available/test.site.conf*

Со следующим содержимым:

*<VirtualHost \*:80>*

*ServerName test.site*

*ServerAlias www.test.site*

*ServerAdmin webmaster@localhost*

*DocumentRoot /var/www/test.site/public\_html*

*ErrorLog ${APACHE\_LOG\_DIR}/error.log*

*CustomLog ${APACHE\_LOG\_DIR}/access.log combined*

*</VirtualHost>*

Виртуальные хосты, как и модули нужно активировать. Для этого есть специальные утилиты. Чтобы активировать созданный виртуальный хост необходимо выполнить команду:

*sudo a2ensite test.site*

Здесь *test.site* — имя файла виртуального хоста. Для отключения тоже есть команда:

*sudo a2dissite test.site*

Настройка виртуальных хостов Apache завершена и на публичном сервере этого было бы достаточно, но если настройка Apache производится на локальной машине, то новый сайт не откроется в браузере. Браузер не знает такого сайта. DNS службы не могут ничего сообщить об этом доменном имени. Но в системе Linux есть возможность самим указать ip адреса для доменных имен в файле */etc/hosts*. Для этого откроем этот файл для редактирования следующей командой:

*nano /etc/hosts*

И добавим в конец файла такие строки:

*127.0.0.1 test.site*

*127.0.0.1 www.test.site*

После выполнения этих действия сайт откроется в браузере по заданному адресу [4].

* 1. **Задание**

Создайте персональную страницу (содержащую ФИО и группу) доступную по адресу *www.<фамилия\_на\_английском>.psu.*

* 1. **Контрольные вопросы**

1. Назовите официальную операционную систему для Raspberry Pi 3.
2. Дайте определение понятию Веб-сервер.
3. Перечислите и охарактеризуйте интерфейсы взаимодействия веб-сервера и веб-приложения.
4. Перечислите наиболее популярные реализации веб-серверов и их особенности.
5. Перечислите основные конфигурационные файлы Apache и доступные в них параметры.
6. Опишите механизм настройки виртуальных хостов Apache.
7. Перечислите основные особенности операционной системы Raspbian.
8. Назовите команду для активации виртуальных хостов.
9. Перечислите сопутствующие дополнительные задачи, решаемые веб-сервером.
10. Опишите механизм исполнения внешнего ПО веб-сервером.
    1. **Содержание отчёта**
11. Ф.И.О., группа, название лабораторной работы.
12. Цель работы.
13. Краткие теоретические сведения.
14. Описание проделанной работы.
15. Результаты выполнения лабораторной работы.
16. Выводы.

Отчет и исходные коды упаковать в архив с названием по следующему шаблону:

AMP-Lab3-«группа, аббревиатуру на латинице»-«Фамилия на латинице».

Пример: **AMP-Lab3-11VS-Ivanov.zip**

# **Лабораторная работа №4 ИСПОЛЬЗОВАНИЕ ВЕБ-СЕРВЕРА ДЛЯ ВЗАИМОДЕЙСТВИЯ С АППАРАТНЫМИ РЕСУРСАМИ ОДНОПЛАТНОГО КОМПЬЮТЕРА RASPBERRY PI 3**

**Цель:** изучить взаимодействие с аппаратными ресурсами одноплатного компьютера Raspberry Pi 3 используя локальный веб-сервер.

* 1. **Теоретическая часть**
     1. **Механизм клиент-серверного взаимодействия**

***Статические HTML страницы***

Смоделируем такую ситуацию: есть пользователь, который хочет найти какую-то информацию и есть удалённая машина, на которой эта информация хранится. Назовем их соответственно клиент и сервер. На этой машине установлено серверное программное обеспечение. Информация на этом сервере хранится в виде HTML документа. Пользователь обратился к серверу за этой HTML страницей. Сервер выбрал интересующую страницу, обработал ее, отправил клиенту. Клиент (браузер) в свою очередь получил страницу, по своему её обработал, и пользователь увидел на экране своего монитора, представленную в удобном виде информацию.

HTML файл – это статическое содержимое, то есть, каждый пользователь, скачавший HTML документ, увидит одно и то же. Однако возникает необходимость, чтобы пользователи имели различные права доступа к документу и в зависимости от этих прав видели только определенную часть этого документа, однако таких средств в HTML нет.

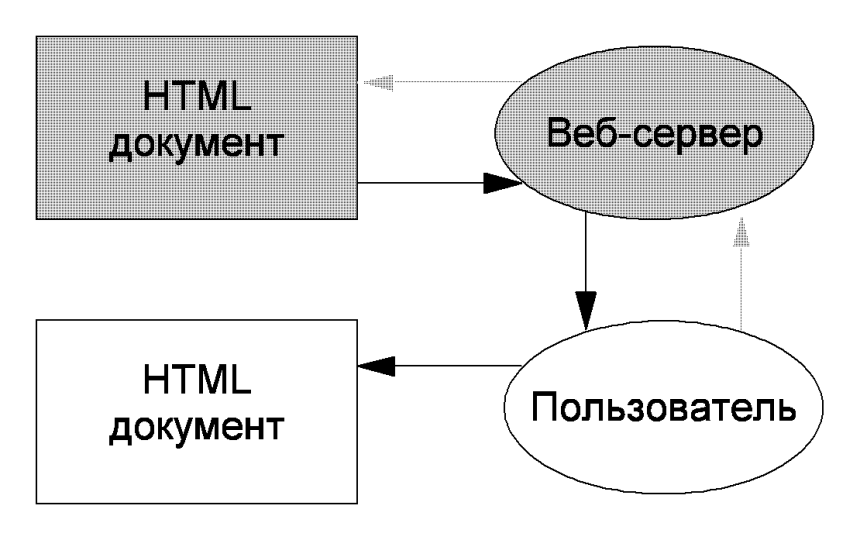


Рисунок 4.1 – Схема обработки статических страниц

На рисунке наглядно показано, как работает выше описанная схема. Данная схема является упрощённой.

***Динамические страницы***

Немного переработаем приведённую схему и посмотрим, как она будет выглядеть при использовании языка программирования PHP. Однако вместо PHP можно использовать и ASP net, и Perl, и Java, поэтому можно использовать нейтральное слово модуль сервера, в данном случае – это PHP сервер. Таким образом, если пользователь делает запрос к web-странице, на которой имеется написанный скрипт на PHP или любом другом серверном языке программирования. Происходит следующая последовательность действий: вначале пользователь обращается непосредственно к серверу, указывая путь к нужному файлу (документу) в адресной строке браузера, сервер находить нужный документ и отдает его на обработку серверному модулю, в данном случае это PHP сервер, который анализирует документ и ищет в нем участки кода, написанные на PHP, чтобы в дальнейшем их обработать. Обратите внимание, что PHP интерпретатор просто отбрасывает все HTML теги.

Затем модуль PHP отдает обработанный документ серверу, после чего сервер отправляет этот документ браузеру, и в итоге пользователь видит у себя на экране HTML страницу, написанную HTML тегами, но при использовании PHP сервера, каждый пользователь, запрашивающий тот или иной документ, на выходе может получить различную информацию.

Для большей наглядности приведён рисунок, на котором изображена передача данных между сервером и клиентом, но уже с использованием PHP модуля.

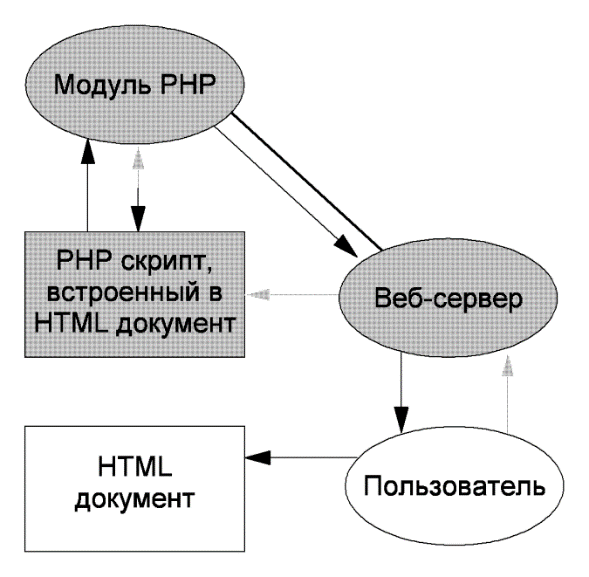


Рисунок 4.1 – Схема обработки динамических страниц

* + 1. **Серверные языки программирования**

Серверные языки программирования нужны для реализации бизнес-логики, то есть разработчик при помощи языка программирования описывает возможные сценарии использования сайта или приложения.

Языков программирования, используемых для серверной веб-разработки, достаточно много: PHP, Ruby, Java, C, Python, Perl и другие.

Серверные языки выполняются непосредственно на самом сервере, специальной программой. Это значит, что для того, чтобы они работали не важно, каким браузером пользуется пользователь, все равно все вычисления будут проходить на удаленном компьютере (сервере).

Увидеть код программы на серверном языке для посетителя сайта вообще невозможно, он видит только результаты работы, которые будут уже представлены в качестве HTML-страницы.

Для выполнения данной лабораторной работы будет использоваться язык PHP.

PHP — скриптовый язык программирования, созданный для генерации HTML-страниц на веб-сервере и работы с базами данных. Ныне поддерживается подавляющим большинством хостинг-провайдеров. Входит в LAMP — «стандартный» набор для создания веб-сайтов.

В области программирования для Сети PHP — один из популярнейших скриптовых языков (наряду с JSP и языками, используемыми в ASP) благодаря своей простоте, скорости выполнения, богатой функциональности и распространению исходных кодов на основе лицензии PHP. PHP особенен наличием ядра и подключаемых модулей, «расширений»: для работы с базами данных, сокетами, динамической графикой, криптографическими библиотеками, документами формата PDF и т. п. Любой желающий может разработать своё собственное расширение и подключить его. Существуют сотни расширений, однако в стандартную поставку входит лишь несколько десятков хорошо зарекомендовавших себя. Интерпретатор PHP подключается к веб-серверу либо через модуль, созданный специально для этого сервера (например, для Apache или IIS), либо в качестве CGI-приложения.

Кроме этого, он может использоваться для решения административных задач в операционных системах UNIX, Linux, Windows и Mac OS X. Однако в таком качестве он не получил распространение, отдавая пальму первенства Perl, Python и VBScript.

Синтаксис языка подобен тому языка Си. Некоторые элементы, такие как ассоциативные массивы и цикл foreach, заимствованы из Perl [4].

Так как Raspberry Pi 3 работает под управление \*-unix подобной ОС, то это позволяет запускать на этом одноплатном компьютере различные приложения под платформу \*-unix. Поэтому интерпретатор PHP отлично работает на Rasperry Pi 3, что позволяет использовать этот язык для данной платформе.

* + 1. **Синтаксис языка программирования PHP**

***Встраивание РНР кода в HTML***

Скрипты на PHP встраиваются в HTML следующим образом:

*<HTML\_TAGS>*

*<?php*

*# PHP GOES CODE*

*?>*

*<HTML\_TAGS>*

Другой вариант:

*<HTML\_TAGS>*

*<SCRIPT LANGUAGE = “PHP”>*

*# PHP GOES CODE*

*</SCRIPT>*

*<HTML\_TAGS>*

Если в настройках PHP включен флаг short\_open\_tag, то можно использовать конструкцию

*<?*

*# php goes here*

*?>*

Во многом PHP напоминает распространённый язык программирования C. Основные отличия касаются лишь обозначения переменных и функций.

***Переменные***

В отличие от большинства языков программирования, таких как C, C++, C#, Pascal, в РНР нет строгого контроля типов. Переменная объявляется в любом месте и становится глобальной для всего скрипта. Пример работы с переменными в РНР

*<?php*

*# myvarсейчас имеет тип Stringи содержит значение ‘HelloWorld’*

*$myvar= “HelloWorld” ;*

*# сейчас myvarимеет Тип int, потом – floatи массив Stringиз 5ти элементов*

*$myvar = 10;*

*$myvar = 20.56;*

*$myvar[4] = “Hello World”; myvar[1] = “I Said”;*

*#*

*# переменная sumбудет иметь значение 15,*

*# так как PHPинтерпретирует переменную в зависимости от контекста*

*#*

*$mystr = “10”;*

*$myint = 5;*

*$sum = $mystr + $myint;*

*?>*

***Области видимости в PHP***

*Private* – видна только в своем собственном классе.

*Public* – видна для любого другого кода, осуществляющего доступ к классу.

*Protected* – видна только родительских классов и классов, которые расширяют текущий класс.

Все переменные в PHP должны начинаться со знака $. Регистр в имени переменной важен, функции – нет.

***Объявления функций***

Пример объявления и вызова функций PHP:

*<?php*

*functionMyFunction( $var1, $var2 ) {*

*return $var1 \* $var2;*

*}*

*echoMyFunction( 5, 10 );*

*?>*

***Взаимодействие с PHP***

Чаще всего серверные скрипты используются для обработки результатов заполнения форм. Например, гостевой книге посетитель вводит данные в форму, которая затем обрабатывается на сервере. Отвечая либо опрос пользователь, аналогично, устанавливает значение определенных полей формы. Тэги и атрибуты должна содержать форма:

*<FORM NAME="имя\_формы"*

*ACTION="путь\_к\_обработчику"*

*METHOD="метод\_передачи\_переменных">*

*поля ввода...*

*</FORM>*

Обработчик - это скрипт на сервере, в который будут переданы значения полей ввода. Обработчиком так же может быть скрипт, содержащий форму. Каждое поле ввода имеет атрибут *NAME*, которое будет передано в обработчик вместе со своим значением. Существует два метода передачи данных: *GET* и *POST*. Их отличие состоит в том, что при использовании метода *GET* значения полей присоединяются к URL, указанному в атрибуте *ACTION*. Происходит следующим образом: *http://site.domain/action.php?имя=значение&...имя=значение*. Пары "имя=значение" создаются для каждого элемента ввода, для которого указано имя атрибутом *NAME*. В случае использования метода *POST* значения полей передаются в заголовке запроса к серверу.

Предположим, что мы создали форму следующего вида:

*<FORM ACTION="mult.php" METHOD="GET">*

*<INPUT TYPE="text" NAME="first" SIZE="4" MAXLENGTH="4">*

*<INPUT TYPE="text" NAME="second" SIZE="4" MAXLENGTH="4">*

*<INPUT TYPE="Submit" VALUE="Умножить">*

*</FORM>*

Скрипт, содержащийся в файле mult.php может выглядеть следующим образом:

*<?php*

*Header("Content-type: text/html");*

*echo "$first умножить на $second получится", $first\*$second;*

*?>*

Так же существует специальный тип поля *HIDDEN*. Это поле, которое не выводится на экран, но, если ему присвоено имя атрибутом *NAME*, значение его передается в форму. Это бывает полезно, например, когда один обработчик может производить не одно, а несколько действий. С помощью такого поля мы можем задать тип действия, которое мы хотим произвести с данными формы.

Далее рассмотрим пример php-скрипта *led.php*, который формирует html-страницу для управления светодиодом с указанием статуса выполнения команды.

*<!DOCTYPE html>*

*<html lang="ru">*

*<head>*

*<meta charset="UTF-8">*

*<title>LED control</title>*

*</head>*

*<body>*

*<?php*

*$data = $\_REQUEST;*

*if(isset($data['turn\_on']))*

*{*

*echo "<p>LED ON!</p>";*

*exec('/home/pi/samples/arg\_gpio 1');*

*}*

*elseif(isset($data['turn\_off']))*

*{*

*echo "<p>LED OFF!</p>";*

*exec('/home/pi/samples/arg\_gpio 0');*

*}*

*?>*

*<form action="test.php" method="POST">*

*<p><strong>LED control</strong></p>*

*<p>*

*<button type="submit" name="turn\_on">ON</button>*

*<button type="submit" name="turn\_off">OFF</button>*

*</p>*

*</form>*

*</body>*

*</html>*

При переходе на страницу *http://localhost/led.php* будет отображена форма, содержащая две кнопки для включения и выключения светодиода. При нажатии любой из них будет отображено соответствующее сообщение, информирующее о состоянии светодиода.

Взаимодействие с GPIO осуществляется с помощью программы *arg\_gpio*, написанной на C, в качестве аргумента принимающей состояние вывода, к которому подключен светодиод.

* 1. **Задание**

Создайте web-приложение для управления светодиодом, реализующее управление режимами работы светодиода: включение/выключение, установка указанной яркости свечения, воспроизведение уникальной последовательности. Номер вывода GPIO для подключения светодиода и уникальная последовательность выбираются по варианту задания лабораторной работы №2 и №1 соответственно.

* 1. **Контрольные вопросы**

1. Опишите механизм клиент-серверного взаимодействия при отображении статических и динамических страниц.
2. Перечислите и охарактеризуйте наиболее популярные серверные языки программирования.
3. Перечислите и охарактеризуйте методы передачи данных web-приложению.
4. Являются ли регистрозависимыми названия переменных и функций в PHP?
5. Каким образом вставить конструкции PHP в HTML-документ?
6. В чем разница между конструкциями include() и require() в PHP?
7. В чем разница между unset() и unlink() в PHP?
8. Может ли значение константы измениться во время выполнения скрипта PHP?
9. Что означает MVC и что делает каждый компонент?
10. Какие есть 3 области видимости в PHP?
    1. **Содержание отчёта**
11. Ф.И.О., группа, название лабораторной работы.
12. Цель работы.
13. Краткие теоретические сведения.
14. Описание проделанной работы.
15. Результаты выполнения лабораторной работы.
16. Выводы.

Отчет и исходные коды упаковать в архив с названием по следующему шаблону:

AMP-Lab4-«группа, аббревиатуру на латинице»-«Фамилия на латинице».

Пример: **AMP-Lab4-11VS-Ivanov.zip**

# **Литература**

1. Хартов, В.Я. Микропроцессорные системы. – М.: Издательский центр «Академия», 2010.– 352 с.

2. Майер, Рето Android 2. Программирование приложений для планшетных компьютеров и смартфонов / Рето Майер. – М.: Эксмо, 2011. – 406 c.

3. Дэрси Лорен, Кондер Шейн Android за 24 часа. Программирование приложений под операционную систему Google. – М.: Рид Групп – Москва, 2011. – 464 c.

4. Ретабоуил Сильвен Android NDK. Разработка приложений под Android на С/С++. М.: ДМК Пресс – Москва, 2012. – 496 c.

5. Петин, В.А. Микрокомпьютеры Raspberry Pi. – СПб.: БХВ-Петербург, 2015. – 240 с