Министерство образования Республики Беларусь

Учреждение образования
«Полоцкий государственный университет»

А.О. Лукьянов

АРХИТЕКТУРА МОБИЛЬНЫХ ПЛАТФОРМ

Методические указания
по выполнению лабораторных работ
для студентов специальностей
1-40 01 01 «Программное обеспечение информационных технологий»,
1-40 02 01 «Вычислительные машины, системы и сети»

Текстовое электронное издание

Новополоцк
Полоцкий государственный университет
2020

Об издании – 1, 2

1 – дополнительный титульный экран – сведения об издании
УДК 004.4
ББК 32.973

Рекомендовано к изданию
методической комиссией факультета информационных технологий
в качестве методическийх указаний
(протокол № 6 от 26.06.2020)

РЕЦЕНЗЕНТЫ:

канд. техн. наук, доц., доц. каф. технологий программирования Полоцкого государственного университета *А. Ф. ОСЬКИН*;
канд. техн. наук, технический директор ООО «ТриИнком» *К. Я. РАХАНОВ*

2 – дополнительный титульный экран – производственно-технические сведения

Для создания текстового электронного издания «Архитектура мобильных платформ» А. О. Лукьянова использованы текстовый процессор Microsoft Office Word и программа Adobe Acrobat XI Pro для создания и просмотра электронных публикаций в формате PDF.

Технические требования:
1 оптический диск.
Системные требования:
PC с процессором не ниже Core 2 Duo;
2 Gb RAM; свободное место на HDD 2,5 Mb;
Windows XP/7/8/8.1/10
привод CD-ROM/DVD-ROM;
мышь.

Редактор С. Е. Рясова

Подписано к использованию 01.03.2021.
Объем издания: 2,5 Мб. Заказ 138.

Свидетельство о государственной регистрации
издателя, изготовителя, распространителя печатных изданий
№ 1/305 от 22.04.2014.

211440, Ул. Блохина, 29,
г. Новополоцк,
Тел. 8 (0214) 59-95-41, 59-95-44
http://www.psu.by

СОДЕРЖАНИЕ

Введение

Целью выполнения лабораторных работ по дисциплине «Архитектура мобильных платформ» является формирование у студентов знаний об аппаратной и программной архитектуре мобильных платформ, включая инструменты для программирования встраиваемых систем и 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.

Выполнение предлагаемого комплекса лабораторных работ базируется на знаниях, полученных в процессе изучения дисциплин «Системное программное обеспечение вычислительных систем» и «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

1. Интерфейс ввода/вывода общего назначения (GPIO)

Интерфейс ввода/вывода общего назначения (*англ.* general-purpose input/output, GPIO) – это 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 их, как правило, оставляют незадействованными. Это полезно в том случае, когда системному интегратору для построения полной системы, использующей тот или иной чип, может потребоваться несколько дополнительных линий цифрового управления. Это дает возможность организовать дополнительные схемы, не создавая их с нуля. Например, чип Realtek ALC260 (аудиокодек) имеет 8 GPIO-выводов, которые остаются неиспользованными по умолчанию. Некоторые системные интеграторы (к примеру, Acer Inc. в 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, аудиокодеках и видеокартах;

– во встраиваемых системах (например, Arduino, BeagleBone, различных PsoC-комплектах и Raspberry Pi) широко используют GPIO для чтения информации от различных внешних датчиков (ИК, видео, температуры, ориентации по трем осям, ускорения), а также для управления двигателями

постоянного тока (используя широтно-импульсную модуляцию), аудио, ЖК-дисплеями или светодиодами для индикации состояния чего-либо.

2. Одноплатный компьютер Raspberry Pi 3

Raspberry Pi – одноплатный компьютер, созданный сотрудниками 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, 3,5 мм аудиовыход и USB-порты для подключения периферийных устройств. Кроме того, как и на Arduino, на плате Raspberry Pi имеется GPIO-интерфейс (рисунок 1). 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.

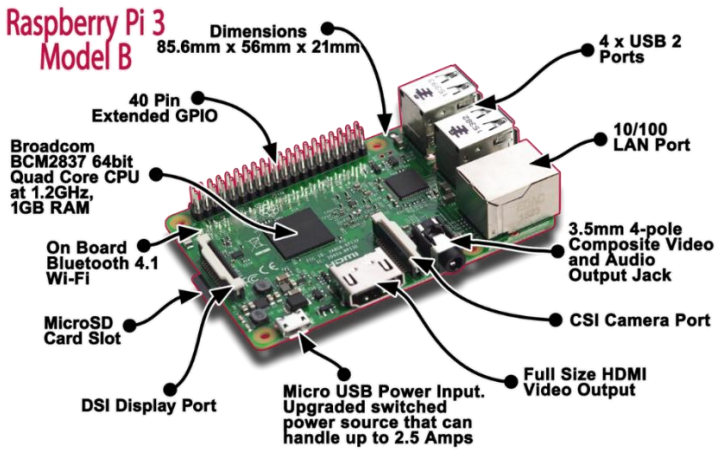

Рисунок 1. – Внешний вид отладочной платы Raspberry Pi 3

3. 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 собой линии питания 3.3 В, 5 В и общий вывод GND – земля (рисунок 2).

Также 27-й (BCM 0) и 28-й (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.

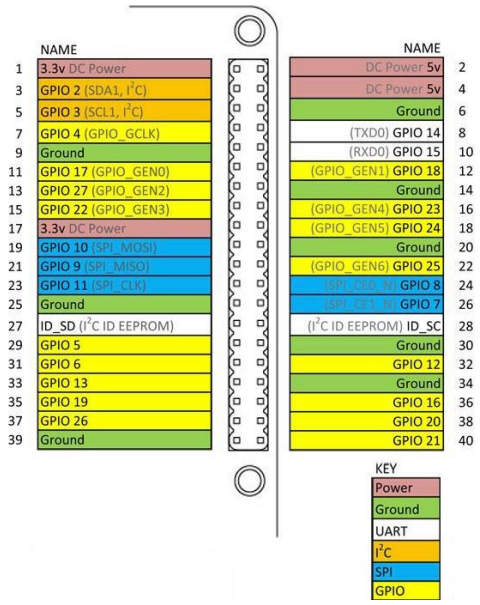

Рисунок 2. – Расположение выводов GPIO отладочной платы Raspberry Pi 3

4. Нумерация выводов GPIO на Raspberry Pi 3

Существует два основных варианта нумерации выводов Raspberry Pi: Board (физическая нумерация по порядку) и BCM (нумерация из чипа). Также некоторые библиотеки, например, WiringPi, используют свою нумерацию (рисунок 3).

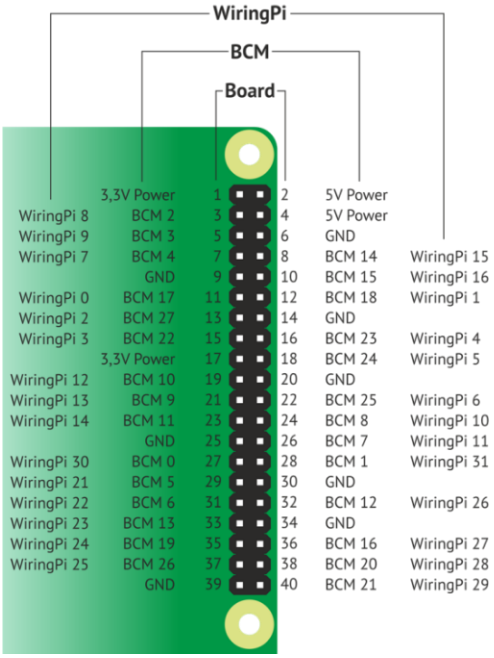

Рисунок 3. – Системы нумерации выводов GPIO отладочной платы Raspberry Pi 3

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

5. 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) резисторами, благодаря которым в режиме *INPUT* они могут быть подтянуты к логической единице, либо стянуты к нулю.

Номиналы сопротивлений непостоянны и находятся в следующих диапазонах:

- для подтягивающего резистора – 50–65 кОм;
- для стягивающего резистора – 50–60 кОм.

Номинал сопротивления подтягивающего/стягивающего резистора для выводов BCM2 и BCM3 составляет 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 в асинхронном режиме – по фронту и по спаду (рисунок 4).

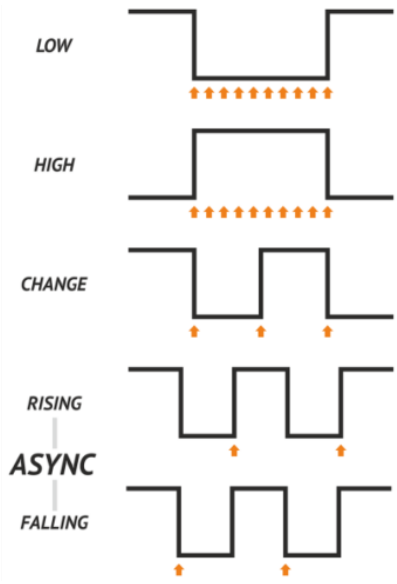

Рисунок 4. – Разновидности прерываний

Также все выводы в режиме *INPUT* имеют входную фильтрацию на триггере Шмитта: преобразовывают аналоговый сигнал в цифровой с резкими переходами между состояниями (рисунок 5).

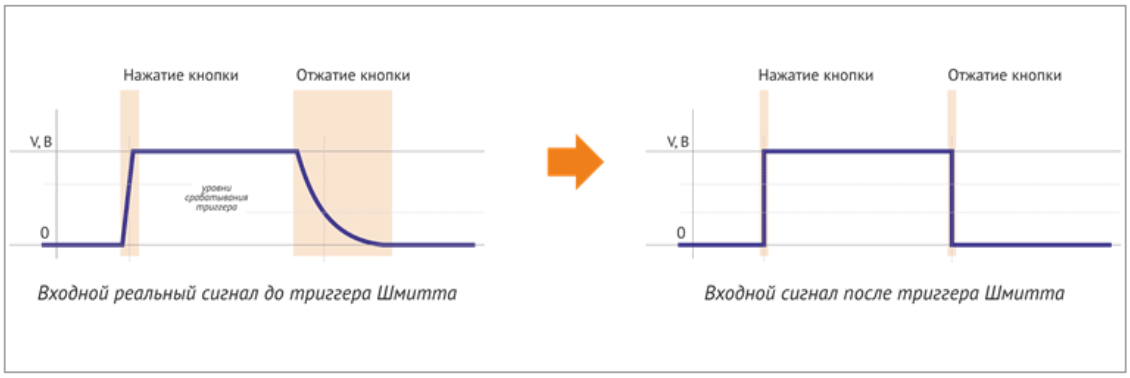

Рисунок 5. – Форма входного сигнала

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

6. 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.

7. Программирование портов ввода-вывода

Программировать поведение GPIO можно на различных языках – Pascal, Ruby, Perl, Java (Pi4J), C, C++, C#, WiringPi, Basic и т.д. Примеры управления GPIO 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 кодом в mian.c (пример).

Пример

```
#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 высо-
кого 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 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 результате 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 учетную 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 из 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 учетом описанных выше электрических параметров портов GPIO на Raspberry Pi 3 для демонстрации работы приведенного примера необходимо подключить светодиод схеме, представленной на рисунке 6.

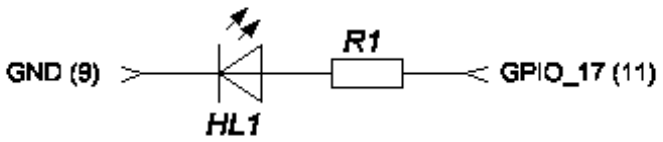

Рисунок 6. – Схема подключения светодиода

При этом номинал резистора R1 должен быть в диапазоне 220–330 Ом.

Задание

Модифицировать код примера так, чтобы последовательность включения и выключения светодиода и количество повторений последовательности соответствовали назначенному варианту (таблица). 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, обозначаемыми соответственно *Квыкл* и *Двыкл*.

Таблица. – Варианты заданий

№ 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

Порядок выполнения общего задания:

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

$$Y = |X| \mathbf{mod} num\_port,$$

где X – число, принятое в *std::cin*;

mod – операция получения остатка от деления нацело;

num_port – общее количество портов, доступных для подключения светодиода.

5. Подать сигнал *HIGH* на номер порта, соответствующий числу Y.

Контрольные вопросы

1. Дайте определение понятию «Интерфейс ввода/вывода общего назначения».
2. Перечислите режимы работы прерываний.
3. Какие электрические характеристики имеют GPIO Raspberry Pi 3?
4. Какими способами выполняется управление GPIO Raspberry Pi 3?
5. Опишите структуру регистров GPIO Raspberry Pi 3.
6. Назовите номиналы подтягивающих и стягивающих резисторов на выводах GPIO Raspberry Pi 3.
7. Охарактеризуйте работу прерываний в различных режимах.
8. Назовите системные файлы, используемые для управления GPIO Raspberry Pi 3.
9. Перечислите регистры управления GPIO Raspberry Pi 3.
10. Перечислите и охарактеризуйте системы нумерации выводов Raspberry Pi 3.

Содержание отчета

Отчет о выполненной лабораторной работе должен содержать следующие сведения:

1) фамилия, имя, отчество студента и его группа;
2) название лабораторной работы;
3) цель работы;
4) краткие теоретические сведения;
5) описание проделанной работы;
6) полученные результаты;
7) выводы.

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

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

Пример названия архива: *AMP-Lab1-11VS-Ivanov.zip*.

Лабораторная работа № 2

Использование широтно-импульсной модуляции (ШИМ) 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.

Теоретическая часть

1. Широтно-импульсная модуляция

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

ШИМ позволяет регулировать яркость подсветки жидкокристаллических дисплеев сотовых телефонов, смартфонов, ноутбуков. Она реализована в сварочных аппаратах, в автомобильных инверторах, в зарядных устройствах и т.д. Любое зарядное устройство в настоящее время 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, 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, 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.

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* (*англ.* Duty cycle (DC). Пример ШИМ-сигнала с различными значениями скважности приведен на рисунке 1.

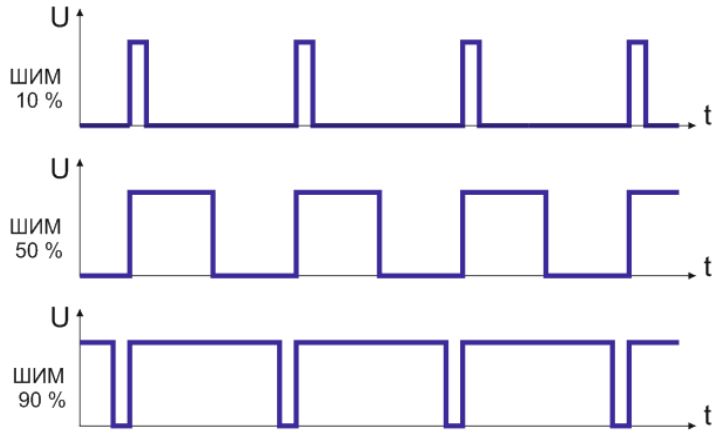

Рисунок 1. – Сигнал широтно-импульсной модуляции

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

2. Программное и аппаратное формирование ШИМ-сигнала

Самый простой способ получить ШИМ на выходе GPIO – это реализовать 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.

Рассмотрим программу, которая заставляет светодиод плавно зажигаться и гаснуть (пример 1).

Пример 1

```
#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 цик-
лов
       //flag- если = 0 светодиод затухает, если = 1 разгора-
ется
       int a=10; // количество полных рабочих циклов
       while (a)
         {
           for (j=100; j!=0; j--) // изменяем коэффициент за-
полнения от 100% до 0%
             {
               t_on=50*d; //находим время включения
               t_off=50*(100-d); //находим время выключения
               // если светодиод затухает, уменьшаем коэффици-
ент заполнения
               if (flag==0) d=d-1;
               // если светодиод разгорается, увеличиваем коэф-
фициент заполнения
               if (flag==1) d=d+1;
               // передаем 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 вклю-
чать
               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, выдает нужный сигнал, не используя дополнительные вычислительные ресурсы на его генерацию. Bcm2837 тоже имеет встроенный аппаратный ШИМ, который является дополнительной функцией портов GPIO 12, 32, 33, 35. Чтобы воспользоваться аппаратными ШИМ, необходимо установить соответствующий порт в режим дополнительных функций (таблица 1) и задать процессору параметры ШИМ-сигнала.

Таблица 1. – Соответствие порта GPIO
и режима дополнительных функций

Порт GPIO	Режим дополнительных функций
GPIO 12 (PWM channel 0) –	Alt5
GPIO 32 (PWM channel 0) –	Alt0
GPIO 33 (PWM channel 1) –	Alt0
GPIO 35 (PWM channel 1) –	Alt5

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

Peripherals	GPIO	Particle	Pin #			Pin #	Particle	GPIO	Peripherals
3.3V			1	X	X	2	5V		
I2C	GPIO2	SDA	3	X	X	4	5V		
	GPIO3	SCL	5	X	X	6	GND		
Digital I/O	GPIO4	D0	7	X	X	8	TX	GPIO14	UART
GND			9	X	X	10	RX	GPIO15	Serial 1
Digital I/O	GPIO17	D1	11	X	X	12	D9/A0	GPIO18	PWM 1
Digital I/O	GPIO27	D2	13	X	X	14	GND		
Digital I/O	GPIO22	D3	15	X	X	16	D10/A1	GPIO23	Digital I/O
3.3V			17	X	X	18	D11/A2	GPIO24	Digital I/O
SPI	GPIO10	MOSI	19	X	X	20	GND		
	GPIO9	MISO	21	X	X	22	D12/A3	GPIO25	Digital I/O
	GPIO11	SCK	23	X	X	24	CE0	GPIO8	SPI (chip enable)
GND			25	X	X	26	CE1	GPIO7	
DO NOT USE	ID_SD	DO NOT USE	27	X	X	28	DO NOT USE	ID_SC	DO NOT USE
Digital I/O	GPIO5	D4	29	X	X	30	GND		
Digital I/O	GPIO6	D5	31	X	X	32	D13/A4	GPIO12	Digital I/O
PWM 2	GPIO13	D6	33	X	X	34	GND		
PWM 2	GPIO19	D7	35	X	X	36	D14/A5	GPIO16	PWM 1
Digital I/O	GPIO26	D8	37	X	X	38	D15/A6	GPIO20	Digital I/O
GND			39	X	X	40	D16/A7	GPIO21	Digital I/O

Рисунок 2. – Выводы GPIO с поддержкой ШИМ

Для использования аппаратного модуля ШИМ в Raspberry Pi 3 воспользуемся примером 2.

Пример 2

```
#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 подключенный к GPIO 12 светодиод плавно загорится и потухнет 5 раз.

3. 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 ее использовании в форме

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

где первый аргумент `argc` – число переданных функции параметров;

второй аргумент `argv` (массив строк) – переданные параметры.

Так как параметры у функции могут быть любыми, то они передаются как строки, а 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.

Чтобы проиллюстрировать использование параметров командной строки, будем передавать состояние вывода GPIO, которое необходимо установить (пример 3).

Пример 3

```
#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.

Задание

Модифицировать пример 3 так, чтобы в качестве параметров командной строки программа принимала два аргумента: команду (установка уровня или скважности ШИМ) и значение (вкл./выкл. или значение скважности). Без указания параметров программа должна выполнить действие, согласно индивидуальному варианту.

Таблица 2. – Варианты заданий

№ 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%

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?

Содержание отчета

Отчет о выполненной лабораторной работе должен содержать следующие сведения:

1) фамилия, имя, отчество студента и его группа;
2) название лабораторной работы;
3) цель работы;

4) краткие теоретические сведения;
5) описание проделанной работы;
6) полученные результаты;
7) выводы.

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

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

Пример названия архива: *AMP-Lab2-11VS-Ivanov.zip*.

Лабораторная работа № 3

Конфигурирование веб-сервера на одноплатном компьютере Raspberry Pi 3

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

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.

Этот 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 управлением *-unix подобной ОС, то это позволяет запускать на нем различные приложения для данной платформы, в том числе и ПО для организации веб-сервера.

2. Веб-сервер

В основе функционирования веб-приложений лежит такое понятие, как веб-сервер. Веб-сервер – это программа, которая принимает входящие HTTP-запросы, обрабатывает их, генерирует HTTP-ответ и отправляет его клиенту. Общий алгоритм работы веб-сервера представлен на рисунке 1 (серым цветом помечены действия, которые обрабатываются веб-сервером).

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 обработке, считывании нужного файла с жесткого диска, формировании HTTP-ответа и отправке 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 в зависимости 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 отличаться от одного приложения к другому, а веб-сервер будет один и тот же. Схема взаимодействия веб-сервера с внешним ПО показана на рисунке 2.

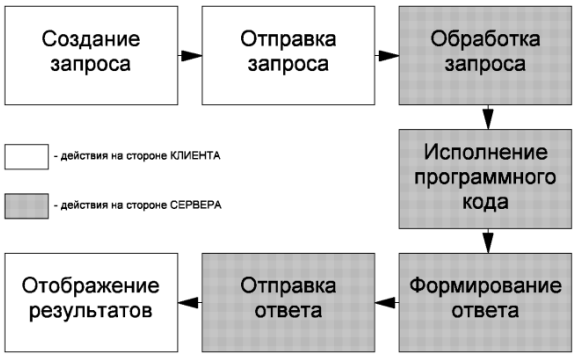

Рисунок 1. – Общий алгоритм работы веб-сервера

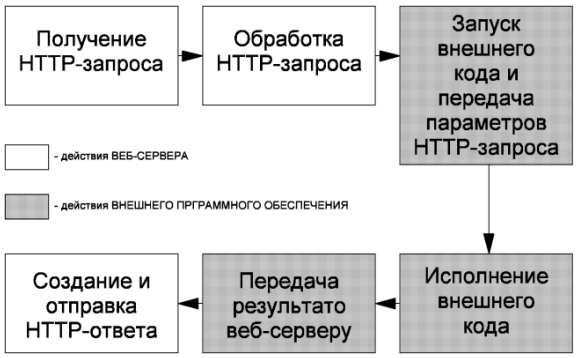

Рисунок 2. – Схема взаимодействия веб-сервера с внешним ПО

Исторически сложилось так, что существует два главных типа интерфейсов 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, состоит 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 с передачей результата обратно веб-серверу. Поскольку веб-сервер и приложение – это разные процессы с точки зрения операционной системы, то для обмена информацией 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.

Однако необходимость создания каждый раз нового процесса влечет за собой дополнительные накладные расходы на его создание (создание процесса – дорогостоящая операция с точки зрения операционной системы) и передачу 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 отличие от CGI взаимодействие в рамках интерфейса ISAPI предполагает, что 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 (т.к. 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 уделять этому вопросу особое внимание.

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

Кроме приведенных выше функций и механизмов, в функции веб-сервера зачастую входят и сопутствующие дополнительные задачи. К этим задачам относится аутентификация 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.

3. Программное обеспечение для работы веб-сервера

ПО для работы веб-сервера позволяет 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 отдает 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.

4. Конфигурирование веб-сервера Apache

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

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

- */etc/apache2/apache2.conf* – 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.

Существует два каталога для *conf*, *mods* и *site*: *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 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. Оптимальное значение этого параметра для большинства задач – 160 секунд;

– *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* – тайм-аут соединения. Обычно для загрузки страницы достаточно 5–10 секунд, соответственно, использовать бóльшие значения этого параметра 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* – устанавливает 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 описания директивы:

<Directory /адрес/в/файловой/системе/>
Параметр значение
</Directory>

Для директив *Directory* доступны следующие 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 и 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* используется только одна директива, *Listen*, которая указывает программе, на каком порте нужно работать.

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

5. 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* (пример 1). Кроме рассмотренных параметров здесь будут использоваться:

– *ServerName* – основное имя домена;

– *ServerAlias* – дополнительное имя, по которому будет доступен сайт;

– *ServerAdmin* – электронная почта администратора;

– *DocumentRoot* – папка с документами для этого домена.

Пример 1

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

Содержимое файла:

*<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 хоста используется 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 самостоятельно указать IP-адреса для доменных имен в файле /etc/hosts. Для этого откроем этот файл для редактирования командой

nano /etc/hosts

и добавим в конец файла 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].

Задание

Создайте персональную страницу (содержащую ФИО и группу), доступную по адресу *www.<фамилия_на_английском>.psu*.

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.

Содержание отчета

Отчет о выполненной лабораторной работе должен содержать следующие сведения:

1) фамилия, имя, отчество студента и его группа;
2) название лабораторной работы;
3) цель работы;
4) краткие теоретические сведения;
5) описание проделанной работы;
6) полученные результаты;
7) выводы.

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

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

Пример названия архива: *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.

Теоретическая часть

1. Механизм клиент-серверного взаимодействия

Статические HTML-страницы. Рассмотрим схему обработки статических HTML-страниц: есть пользователь, который хочет используя браузер (клиент) найти нужную ему информацию, и есть удаленная машина (сервер), на которой эта информация хранится. На этой машине установлено серверное программное обеспечение и хранится искомая информация в виде HTML-документа (HTML-страницы). Пользователь посредством клиента обращается к серверу за этой HTML-страницей. Сервер выбирает интересующую страницу, обрабатывает ее и отправляет клиенту. Клиент в свою очередь получает страницу и обрабатывает ее по-своему. В результате пользователь видит на экране представленную в удобном виде информацию. На рисунке 1 наглядно показано, как работает описанная схема. Данная схема является упрощенной.

HTML-файл – это статическое содержимое, то есть каждый пользователь, скачавший HTML-документ, увидит одно и то же. Однако необходимо, чтобы пользователи имели различные права доступа к документу и в зависимости от этих прав видели только определенную его часть, но средств, обеспечивающих разделение прав доступа, в HTML нет.

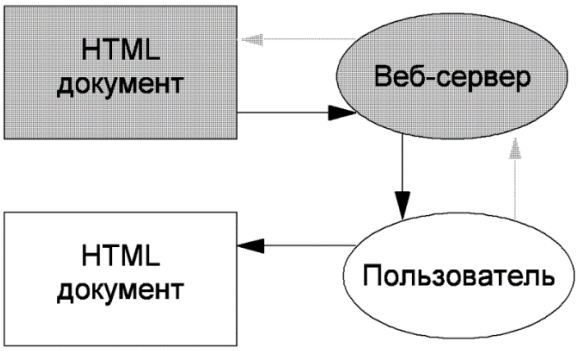

Рисунок 1. – Схема обработки статических HTML-страниц

Динамические страницы. Немного переработаем приведенную выше схему и посмотрим, как она будет выглядеть при использовании языка программирования PHP. Так как вместо PHP можно использовать и ASP.net, и Perl, и Java, можно ввести общее понятие «модуль сервера» (в данном случае – это будет PHP-сервер). Таким образом, если пользователь делает запрос

к web-странице, на которой имеется написанный на PHP или любом другом серверном языке программирования скрипт, выполняются следующие действия: пользователь обращается непосредственно к серверу, указывая путь к нужному файлу (документу) в адресной строке браузера; сервер находит нужный документ и отдает его на обработку серверному модулю (в данном случае – PHP-серверу), который анализирует документ и ищет в нем участки кода, написанные на PHP, чтобы в дальнейшем их обработать. Следует обратить внимание, что PHP-интерпретатор игнорирует все HTML-теги.

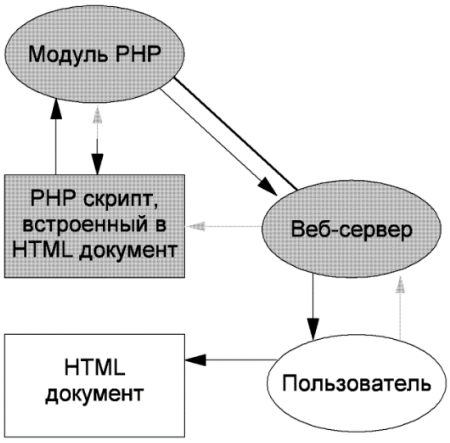

Рисунок 2. – Схема обработки динамических страниц

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

Графически передача данных между сервером и клиентом с использованием PHP-модуля представлена на рисунке 2.

2. Серверные языки программирования

Серверные языки программирования предназначены 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 др.

Серверные языки работают на стороне сервера. Следовательно, на их работе не сказывается то, с какого браузера пользователь выполняет вход: все вычислительные процессы производятся на удаленном компьютере (сервере). Доступ посетителей сайта к коду программы блокируется. Пользователь видит лишь результат его работы – HTML-страницу.

При выполнении лабораторной работы будет использоваться язык PHP – скриптовый язык программирования, созданный для генерации HTML-страниц на веб-сервере и работы с базами данных. Этот язык поддерживается подавляющим большинством хостинг-провайдеров и 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. Особенность PHP состоит в наличии 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 свое 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 того, он может использоваться для решения административных задач в операционных системах UNIX, Linux, Windows и Mac OS X. Однако в таком качестве он не получил распространения, отдав пальму первенства Perl, Python и VBScript.

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

Raspberry Pi 3 функционирует под управлением *-unix подобной ОС, что позволяет запускать на этом одноплатном компьютере различные приложения для платформы *-unix, поэтому интерпретатор PHP отлично работает на Rasperry Pi 3. Очевидно, это позволяет использовать PHP на данной платформе.

3. Синтаксис языка программирования PHP

3.1. Встраивание PHP-кода в HTML

Скрипты на PHP встраиваются в HTML несколькими способами.

Вариант 1:

```
<HTML_TAGS>
      <?php
            # PHP GOES CODE
      ?>
<HTML_TAGS>
```

Вариант 2:

```
<HTML_TAGS>
      <SCRIPT LANGUAGE = “PHP”>
```

```
            # PHP GOES CODE
        </SCRIPT>
<HTML_TAGS>
```

Вариант 3. Если в настройках PHP включен флаг short_open_tag, то можно использовать конструкцию

```
<?
      # php goes here
?>
```

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

3.2. Переменные

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

```
<?php
# myvar сейчас имеет тип String и содержит значение
'HelloWorld'
$myvar= "HelloWorld" ;
# сейчас myvar имеет Тип int, потом – float и массив
Stringиз 5-ти элементов
$myvar = 10;
$myvar = 20.56;
$myvar[4] = "Hello World"; myvar[1] = "I Said";
#
# переменная sum будет иметь значение 15,
# так как PHP интерпретирует переменную в зависимости
от контекста
#
$mystr = "10";
$myint = 5;
$sum = $mystr + $myint;
?>
```

3.3. Области видимости переменных

Private – переменная видна только в своем собственном классе.

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

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

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

3.4. Объявление функций

Пример объявления и вызова функций PHP:

```
<?php
functionMyFunction( $var1, $var2 ) {
      return $var1 * $var2;
}
echoMyFunction( 5, 10 );
?>
```

3.5. Взаимодействие с PHP

Чаще всего серверные скрипты используются для обработки результатов заполнения форм. Например, в гостевой книге посетитель вводит данные в форму, которая затем обрабатывается на сервере. Отвечая на какой-либо вопрос, пользователь аналогично устанавливает значение определенных полей формы. Тэги и атрибуты, которые 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 также может быть скрипт, содержащий форму. Каждое поле ввода имеет атрибут *NAME*, который будет передан 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*. Это происходит следующим образом:

http://site.domain/action.php?имя=значение&...имя=значение

Пары «имя = значение» создаются для каждого элемента ввода, для которого имя указано 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;
?>
```

Также существует специальный тип поля *HIDDEN*. Это поле, которое не выводится на экран, но, если ему присвоено имя атрибутом *NAME*, его значение 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 необходимо 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</but-
ton>
        </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.

Задание

Создайте веб-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.

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 веб-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 конструкциями 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. Назовите три области видимости в PHP.

Содержание отчета

Отчет о выполненной лабораторной работе должен содержать следующие сведения:

1) фамилия, имя, отчество студента и его группа;
2) название лабораторной работы;
3) цель работы;
4) краткие теоретические сведения;
5) описание проделанной работы;
6) полученные результаты;
7) выводы.

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

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

Пример названия архива: *AMP-Lab4-11VS-Ivanov.zip*.

Литература

1. Хартов, В.Я. Микропроцессорные системы / В.Я. Хартов. – М. : Академия, 2010.– 352 с.

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

3. Дэрси, Л. Android за 24 часа. Программирование приложений под операционную систему Google / Л. Дэрси, Ш. Кондер. – М. : Рид Групп, 2011. – 464 с.

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

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