Dr. Bob Davidov

Построение RT системы управления на базе компьютера Raspberry Pi

Цель работы: Рассмотрение средств для построения простейших RT систем управления на базе компьютера Raspberry Pi.

Задача работы: Построение RT системы термостатирования на базе компьютера Raspberry Pi.

Приборы и принадлежности: Компьютер Raspberry Pi (версия B), основной (удаленный) компьютер, периферия, датчик температуры, твёрдотельное реле и нагреватель (Рис. 1. **Компоненты, задействованные в работе**).

ВВЕДЕНИЕ

Компьютер Raspberry Pi (RPi) мало отличается от компактных контроллеров по стоимости и габаритам, при этом, наличие операционной среды, многочисленных программных приложений и сетевая связь с мощными специальными средами удаленных компьютеров позволяют значительно упростить разработку систем управления на базе вычислительной платформы RPi. Для подобных работ очевидна необходимость понимания задачи построения системы реального времени (RT системы) в Linux среде Raspberry Pi, в среде, которая изначально не проектировалась для RT систем и использует ресурсы компьютера который не имеет собственного аппаратного таймера.

В рамках этой работы затронуты следующие вопросы.

- Установка и проверка компилятора GNAT для Raspberry Pi
- Подключение USB-флеш-накопителя к ОС Raspbian
- Подключение аппаратного ШИМ RPi
- Подключение I^2C датчиков к порту GPIO компьютера Raspberry Pi
- Автономное управление температурой без использования программных средств
- Обеспечение интервалов реального времени Raspberry Pi
- Работа с компьютером Raspberry Pi в режиме терминала.
- Интеграция АДА и С программных модулей.
- Управление с применением аппаратного ШИМ RPi.
- Проверка работоспособности датчика температуры с I^2C интерфейсом.
- Проверка работоспособности канала релейного исполнительного устройства (лампового нагревателя).
- Построение RT релейной системы термостатирования на базе Raspberry Pi.

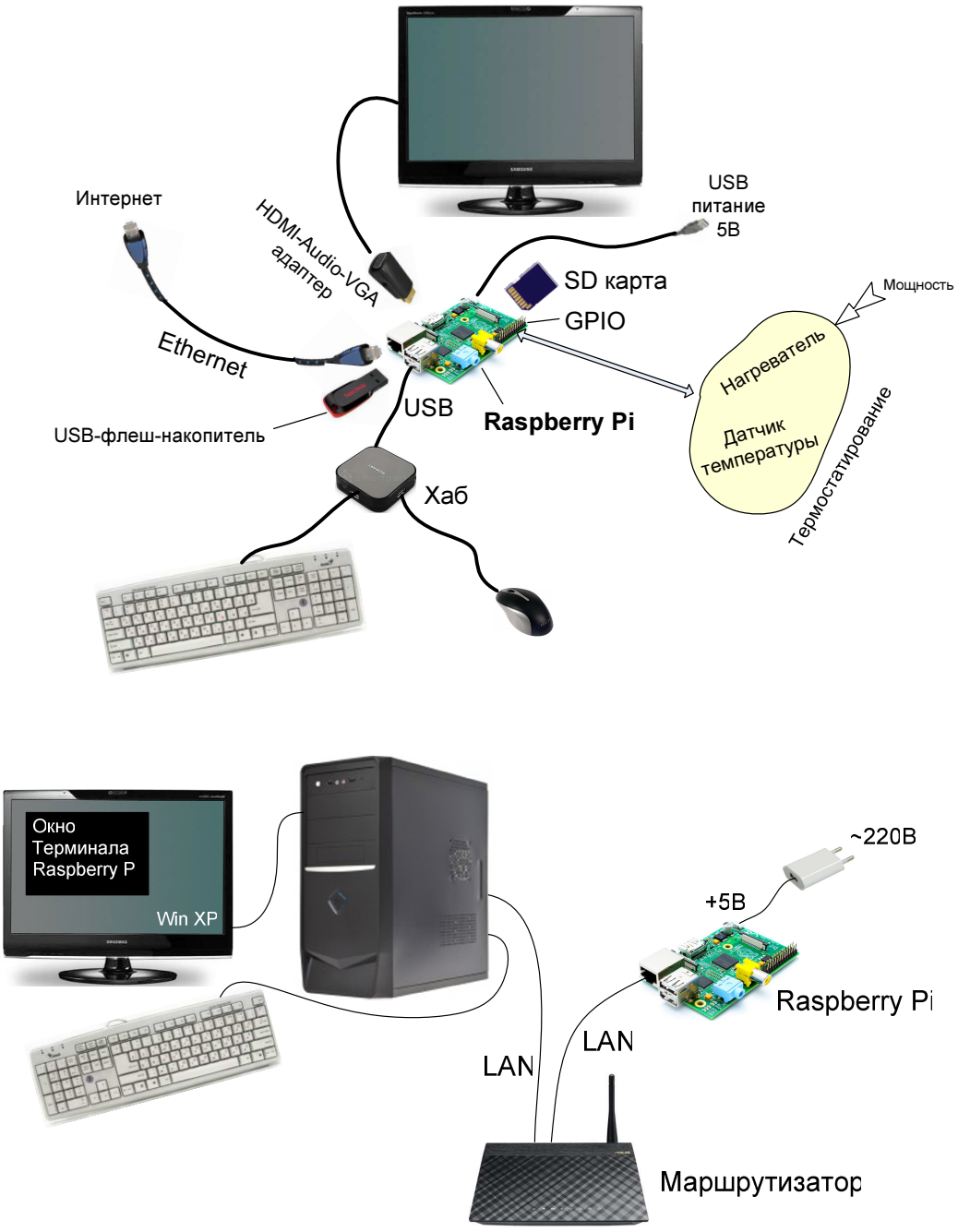


Рис. 1. Компоненты, задействованные в работе

ОБЩИЕ СВЕДЕНИЯ

Установка и проверка компилятора GNAT для Raspberry Pi

Компилятор GNAT можно установить в следующем порядке.

1. На компьютере Raspberry Pi, подключенном к Интернет, обновите список репозиториев.

   $ sudo apt-get update

2. Для установки GNAT введите следующую команду

   $ sudo apt-get install gnat

3. Создайте директорию с именем, например, ada

   $ sudo mkdir /ada

4. Разрешите запись в созданную директорию

   $ sudo chmod 777 /ada

5. Перейдите в папку ada

   $ cd /ada

6. В текстовом редакторе nano откройте файл, например, hello.adb

   $ nano hello.adb

7. Напишите тестовую программу на языке Ada.

```
with Ada.Text_IO;
use Ada.Text_IO;

--  Print a message out to the screen.
Procedure Hello is
Begin
        Put_Line ("Hello from Ada!");
end Hello;
```

8. Закройте редактор с сохранением программы: <Ctrl + X> затем Y

9. Раскройте исходное содержимое текущей папки

   $ ls

10. Откомпилируйте программу.

    $ gnatmake hello

11. Командой ls раскройте содержимое текущего каталога в котором исполняемые файлы выделяются зеленым цветом.

    Дополнительно к hello.adb созданы файлы:

    hello.ali
    hello.o
    hello - загрузочный файл

12. Запустите программу hello на выполнение и проверьте работу программы по выводимому сообщению.

    $ ./hello

Подключение USB-флеш-накопителя к ОС Raspbian

1. Вставьте USB флеш накопитель в USB порт Raspberry Pi. Если оба USB порта компьютера RPi заняты, например, клавиатурой и мышкой, для подключения дополнительных USB портов к компьютеру используйте USB хаб.

Рис. 2. Пример USB хаба который использует один порт и даёт три USB порта.

Примечание:
1. Raspberry Pi рассчитан на подключение USB устройства с током потребления не более 100 мА.
2. Отклонение питающего напряжения RPi от 4.75 .. 5.25 В говорит о проблемах с питанием.
3. В нормальной ситуации красный индикатор платы RPi: PWR не должен мигать поскольку он напрямую подключен к шине питания.

2. Убедитесь, что при подключении накопителя в разделе dev появляются файлы /dev/sda и /dev/sda1.

    sudo ls /dev/sd*

3. Выведите на экран UUID – идентификатор файловой системы накопителя.

    $ sudo blkid /dev/sda1

    Вывод, например: UUID="A4AD-0A93" TYPE="vfat"

4. В этом случае UUID = A4AD-0A93. Запомните свой UUID.
5. Текстовым редактором nano откройте файл /etc/fstab

    $ sudo nano /etc/fstab

6. Добавьте в файл следующую строку.

    UUID=A4AD-0A93 /mnt/usb auto defaults,auto,umask=000,users,rw,uid=pi,gid=pi 0 0

7. Закройте редактор с сохранением отредактированного файла <Crtl + X> затем Y.
8. Создайте новую папку, например, usb

    $ sudo mkdir /mnt/usb

9. Дайте разрешение на чтение/запись/запуск исполняемых файлов папки usb

    $ sudo chmod 777 /mnt/usb

10. Откройте содержимое флеш накопителя

    $ sudo mount /mnt/usb
    $ ls /mnt/usb

11. Перед отсоединением флеш накопителя введите команду umount.

    $ sudo umount /mnt/usb

12. Чтобы флеш накопитель читался без дополнительных команд сразу после подключения накопителя сделайте следующие настройки (одной строкой).

    $ sudo sh -c 'echo "KERNEL==\"sda\", RUN+=\"/bin/mount /mnt/usb\"" >> /etc/udev/rules.d/99-mount.rules'

13. Перезагрузите систему

    $ sudo reboot

14. Теперь система должна “видеть” USB флеш накопитель после его подключения к RPi. Проверьте это командой

    $ ls /mnt/usb

Подключение аппаратного ШИМ RPi

В Raspberry PI имеется только один вывод ШИМ GPIO 18 (12-й физический контакт разъёма) встроенного аппаратного генератора, доступ к которому для программ на языке Python осуществляется через библиотеки доступа к выводам GPIO, например, WiringPi.

Установка библиотеки WiringPi в каталог /home/pi/ выполняется с использованием следующих команд [1].

```
$ sudo apt-get install python-dev python-setuptools
$ git clone https://github.com/WiringPi/WiringPi-Python.git
$ cd WiringPi-Python
$ git submodule update –init
$ sudo python setup.py install /home/pi/WiringPi-Python
```

Внимание! В библиотеке WiringPi используется своя нумерация выводов GPIO. Так, физический контакт GPIO под номером 12 имеет номер 1 по версии WiringPi.

Звуковой сигнал 3.5 мм аудио разъёма проходит через вывод ШИМ - GPIO 18 (12-й физический контакт разъёма). При необходимости одновременно использовать аппаратный ШИМ и звуковой сигнал используйте звуковой сигнал HDMI интерфейса.

ШИМ можно построить и на программном уровне с использованием библиотеки, например, pizypwm, но такой ШИМ, в сравнении с аппаратным, имеет худшую стабильность и потребляет значительно больше вычислительных ресурсов.

Подключение I²C датчиков к порту GPIO компьютера Raspberry Pi

Датчик температуры AD7416 с встроенным чувствительным элементом, АЦП и I²C интерфейсом имеет следующие характеристики:

- Точность измерения температуры: не хуже +/- 2 С в диапазоне −25 .. +100°С.
  +/- 3 С в диапазоне −55 .. +125°С.
- Разрядность АЦП: 10
- Разрешение датчика: ¼ °С на единицу младшего разряда АЦП
- Время преобразования температуры: не более 40 мкс
- Пространство задания адреса датчика: 1001**000** ... 1001**111** (3 входа)
- Частота I²C интерфейса: 400 кГц
- Напряжение питания: 2.7 .. 5.5 В
- Максимальное потребление датчика: 1 мА
- Индикатор превышения заданной критической температуры (выход **O**ver**T**emperature **I**ndicator): имеется
- Автоматическое переключение в режим пониженного энергопотребления по окончании преобразования: имеется
- AD7416 является функциональным аналогом LM75 с улучшенными характеристиками

Примечание: датчики температуры AD7417 и AD7418 имеют дополнительно 4-е 10-разрядных АЦП для ввода внешних аналоговых сигналов.

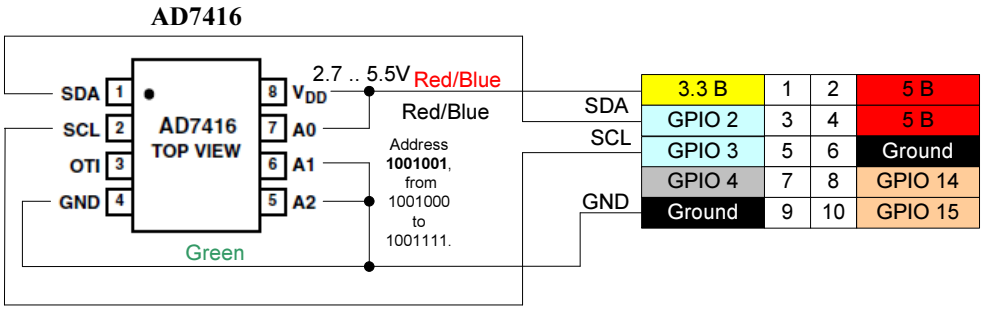


Рис. 3. Схема подключения датчика температуры AD7416 к порту GPIO компьютера Raspberry Pi

В операционной системе Raspbian шина I²C изначально (по дефолту) отключена. Чтобы выводы 3 и 5 разъема GPIO пропускали сигналы SDA и SCL шины I²C (**Рис. 3**), необходимо выполнить следующее.

- Командой sudo nano /etc/modprobe.d/raspi-blacklist.conf откройте файл с записями

  blacklist spi-bcm2708
  blacklist i2c-bcm2708

- Закомментируйте строку blacklist i2c-bcm2708 установкой # в начале строки.

  # blacklist spi-bcm2708.

  Примечание. Таким же образом можно можно обеспечить доступ к интерфейсу SPI:

- Сохраните изменения (Ctrl-x) и перегрузите систему командой консоли sudo reboot

Теперь, после каждого включения компьютера для активизации интерфейса I^2C (0-го и/или 1-го) необходимо вводить через консоль следующие команды.

```
sudo modprobe i2c-dev
sudo chmod o+rw /dev/i2c-0
sudo chmod o+rw /dev/i2c-1
```

Примечание. 1. Эти команды можно записать в файл /etc/rc.local для автоматического активирования I^2C интерфейса при загрузке компьютера.

Для программирования работы I^2C устройств на языке Python необходима библиотека, например, smbus которая устанавливается командой

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

Скорость I^2C шины, например, 400000 Гц можно задать командами

```
cd dev
sudo modprobe -r i2c_bcm2708 && sudo modprobe i2c_bcm2708 baudrate=400000
```

которая удаляет старый драйвер шины и устанавливает новый, с заданной частотой.

Значение частоты I^2C драйвера можно вывести на экран командой dmesg

Автономное управление температурой без использования программных средств

Датчик температуры AD7416 может самостоятельно поддерживать заданную температуру переключением нагревателя подключенного к выходу OTI (см. Рис. 4 и Рис. 5). Уровни включения / выключения нагревателя задаются значениями T_{OTI} и T_{HYST} хранящихся в соответствующих регистрах датчика.

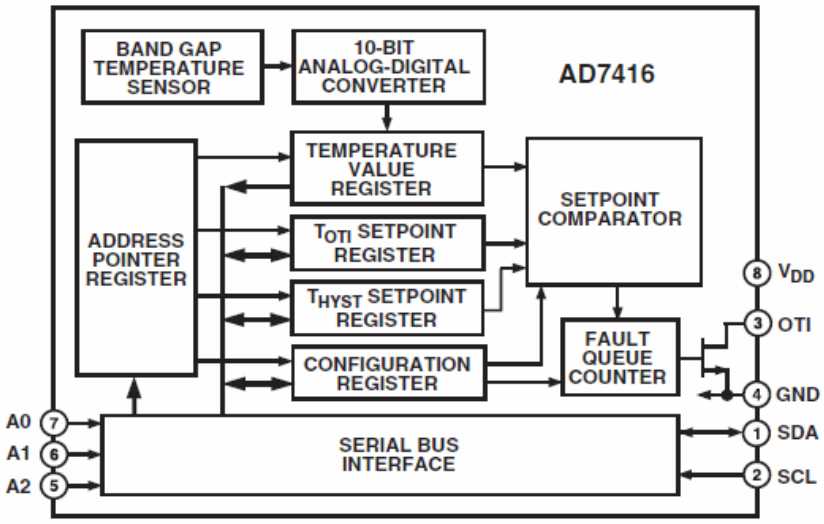


Рис. 4. Функциональная схема датчика температуры AD7416.

В исходном состоянии в регистре T_{OTI} записана температура 80 С, а в регистре T_{HYST} 75 С.

Адрес регистра (Read/Write) значения T_{OTI}: 00000 011
Адрес регистра (Read/Write) значения T_{HYST}: 00000 010
Адрес 16-разрядного регистра (Read) значения температуры: 00000 000
Адрес регистра (Read/Write) конфигурации: 00000 001

Параметры регистра конфигурации:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0			Полярность OTI выхода	Comparator mode / Interrupt mode	Shut-down

OTI выход датчика имеет открытый коллектор с падением напряжения 0.8 В при токе 4 мА. Подключение нагрузки (нагревателя) к выходу OTI можно организовать через твердотельное реле, например, PF240D25 (Рис. 6).

Характеристики твердотельного реле – ключа для подачи электрической энергии нагревателю:

- Тип: PF240D25
- Выходные параметры

  Напряжение передаваемое нагрузке: от 12 до 280 В (47 .. 63 Гц)
  Рабочий ток: до 10 А
  Пиковый ток: 25 А
  Максимальное время включения/выключения: ½ цикла

- Входные параметры

  Входное напряжение: 3 .. 15 В
  Напряжение включения: > 3 В
  Напряжение выключения: < 1 В
  Номинальный импеданс: 300 Ом

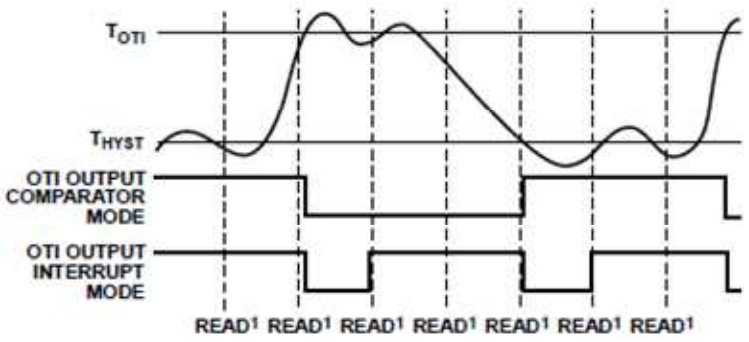
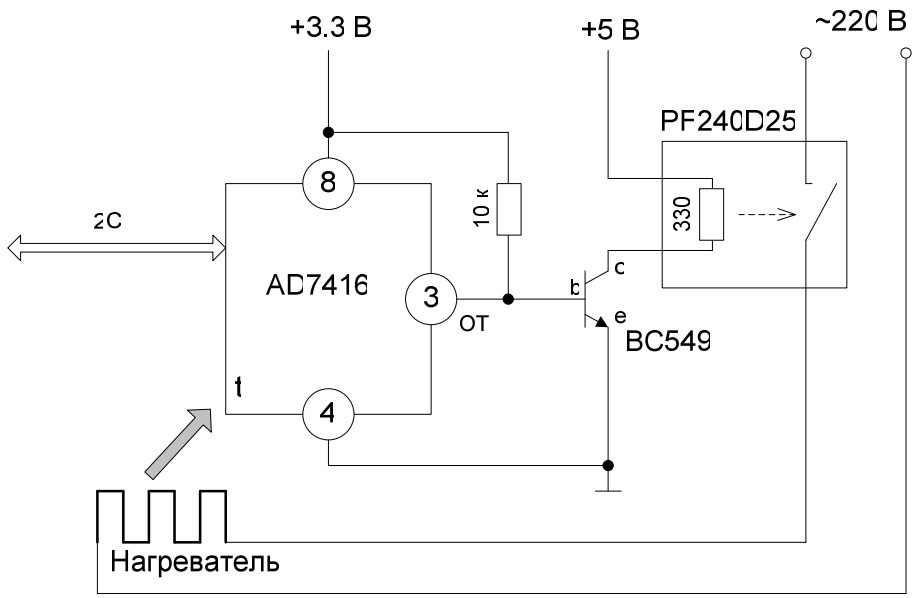


Рис. 5. Временная зависимость выхода OTI микросхемы AD7416 от температуры в режиме COMPARATOR и INTERRUPT.



BC549 прп-транзистор (не оптимальный, избыточен по напряжению, току и усилению по току):

Максимальный ток коллектора: 100 мА
Максимальное напряжение (к-э): 30 В.
Коэф. усиления по току h21 (тип.): 270
Напряжение насыщения (кэ): < 250 mV (10 mA)

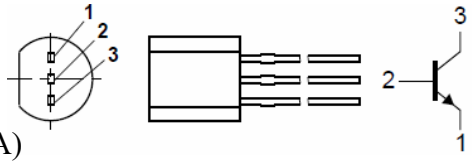


Рис. 6. Пример схемы автономного управления температурой в релейном режиме без использования ресурсов компьютера. От компьютера задаются только уровни T_{OTI} и T_{HYST}.

Обеспечение интервалов реального времени Raspberry Pi

С момента появления ОС Linux не была ориентирована на решение задач реального времени (RT). Она не даёт никаких гарантий по точности времени выполнения задачи. Однако, существуют различные способы позволяющие улучшить временные характеристики процессов. Одним из наиболее интересных способов является PREEMPT_RT - модификация стандартного ядра Linux для решения задач реального времени.

Таблица 1. Список проблем и варианты решения задачи повышения точности такта реального времени.

N	Проблемы RT	Решение
1	Период программы с фиксированной задержкой usleep(delay) зависит от времени выполнения основной программы.	Использовать функцию ожидания нужного момента: sleep_until() (см. Рис. 7)
2	Параллельная работа других процессов приводит к значительным флуктуациям ожидания между двумя тактами.	Запускать процессы реального времени в Linux с приоритетами (scheduling class) отличными от приоритета нормального планирования struct sched_param sp; sp.sched_priority = 30; Это не позволяет другим процессам забирать слишком много процессорного времени у RT задач.
3	RT программа не защищена от использования памяти другими процессами.	Блокировать доступ других процессов к страницам используемой памяти функцией mlockall()
4	Зависимость RT процесса от процессов с более низким приоритетом, например, scanf() или printf()	Одно из решений – использовать так называемый не блокируемый ввод/вывод (non-blocking I/O) Другой, более распространенный подход – передать все интерфейсные пользовательские задачи другими процессам, не имеющим RT приоритета. В последнем случае необходимо защитить общие данные RT и не RT процессов, например, семафорами (semaphores).
5	Задачи низкого приоритета при обновлении общих данных (shared data) блокируют RT процесс	Использовать в RT задачах функцию sem_trywait() которая проверяет значение заданного семафора на положительность, если значение не равно нулю уменьшает его на единицу и немедленно возвращает управление процессу. В отличии от sem_wait() она не приостанавливает процесс. Обе функции возвращают 0

		в случае успешного завершения. В режиме реального времени можно использовать код с переменной которая указывает на структуру, содержащую новое значение задержки записанное интерфейсной пользовательской задачей. По существу, интерфейсная задача использует общие данные до момента уведомления RT процесса при помощи функции sem_post. После этого момента он не может изменять общие данные, пока поток RT не даст соответствующего разрешения.

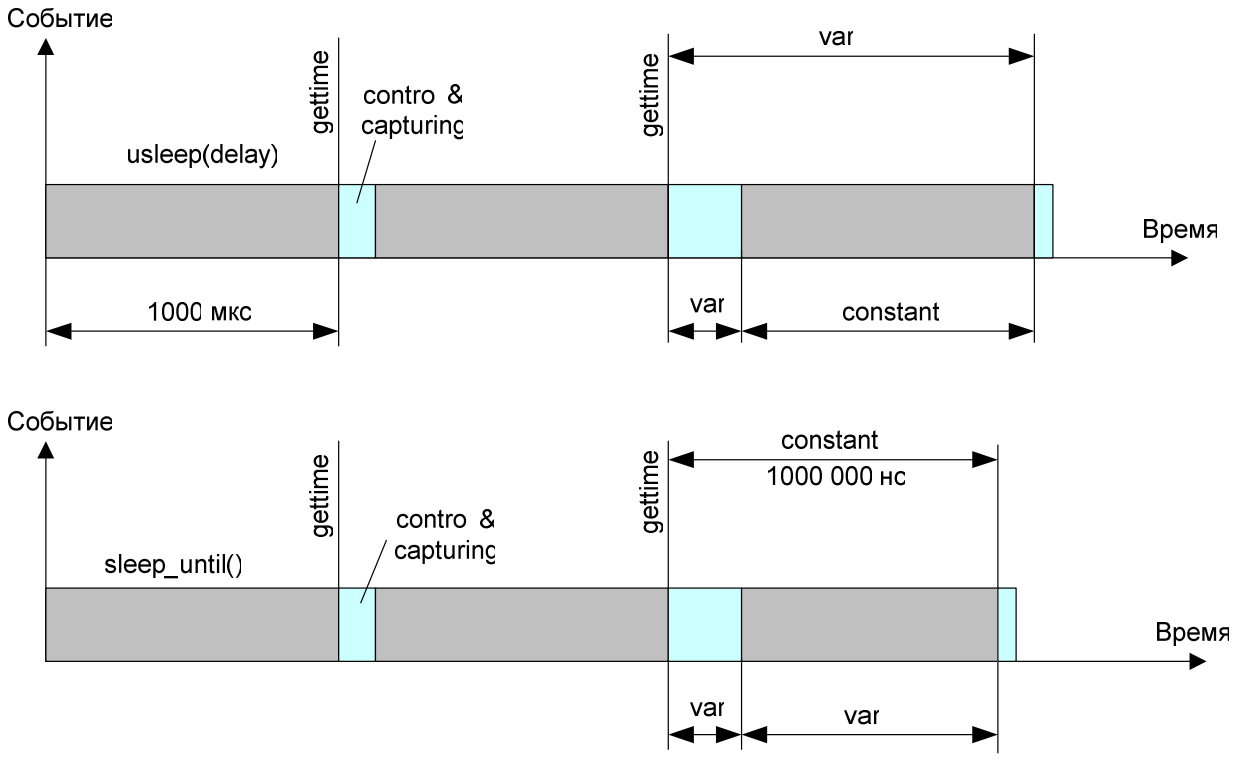


Рис. 7. Использование фиксированной задержки для формирования тактов реального времени (верхняя диаграмма) и применение задержки до наступления события (нижняя диаграмма). Период нижнего процесса менее чувствителен к временным вариациям выполнения основного кода.

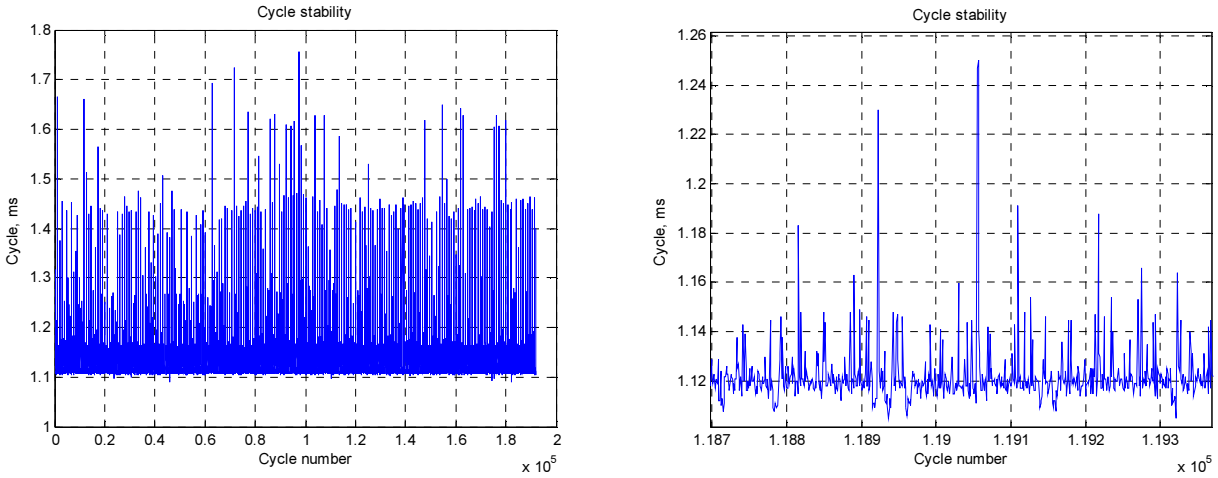


Рис. 8. Пример нестабильности 1 мс периода выполнения задачи RT с использованием фиксированной задержки. Время выполнения основного кода влияет на продолжительность RT периода.

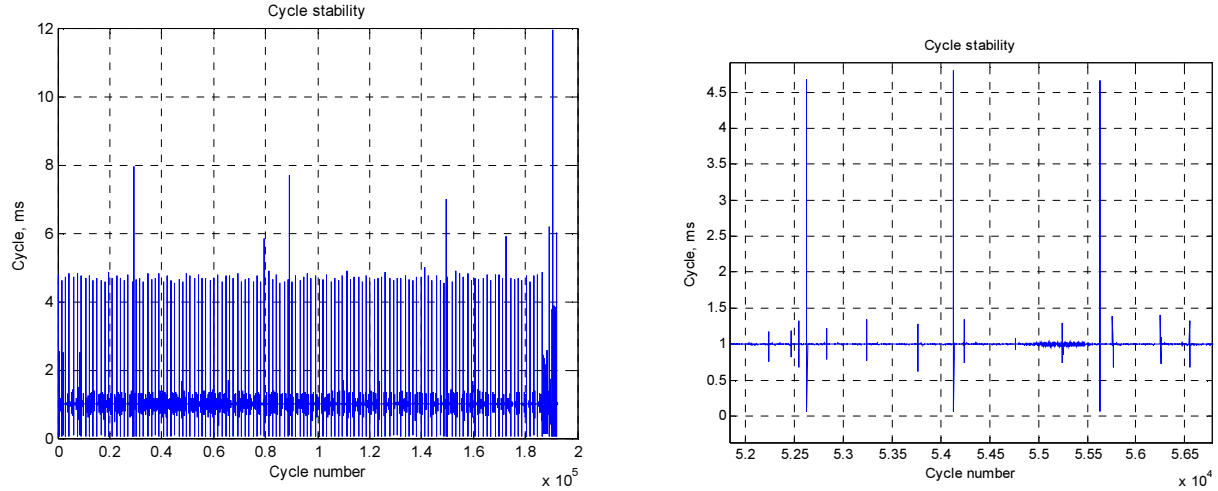


Рис. 9. Нестабильность 1 мс периода RT задачи вызванная параллельным выполнением подобной задачи с таким же уровнем приоритета нормального планирования.

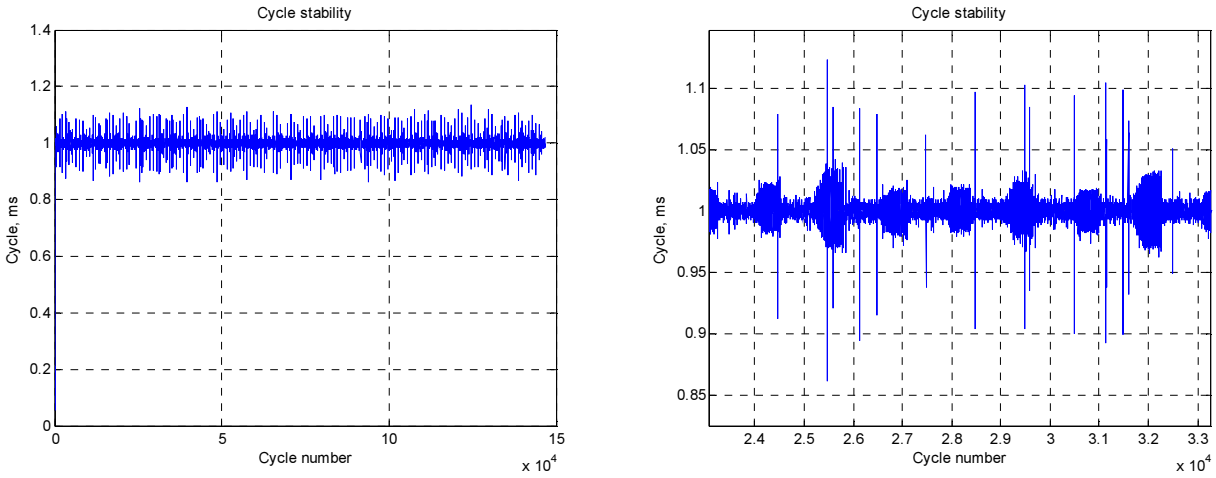


Рис. 10. Увеличенная стабильность 1 мс RT периода управления шаговым двигателем [2] в многозадачном режиме при запуске других процессов. Повышение стабильности достигнуто за счет введения скомпенсированной задержки sleep_until(), изменения приоритета и защиты используемой памяти. Среднее значение периода составляет 0.99995 мс.

ПРИМЕРЫ ПОЛУЧЕНИЯ ПРОВЕРЕННЫХ РЕЗУЛЬТАТОВ И ВАРИАНТЫ ДЛЯ САМОКОНТРОЛЯ

Задание 1. Работа с компьютером Raspberry Pi в режиме терминала.

1. Для работы в режиме терминала подключите Raspberry Pi к компьютеру через маршрутизатор как показано на рисунке.

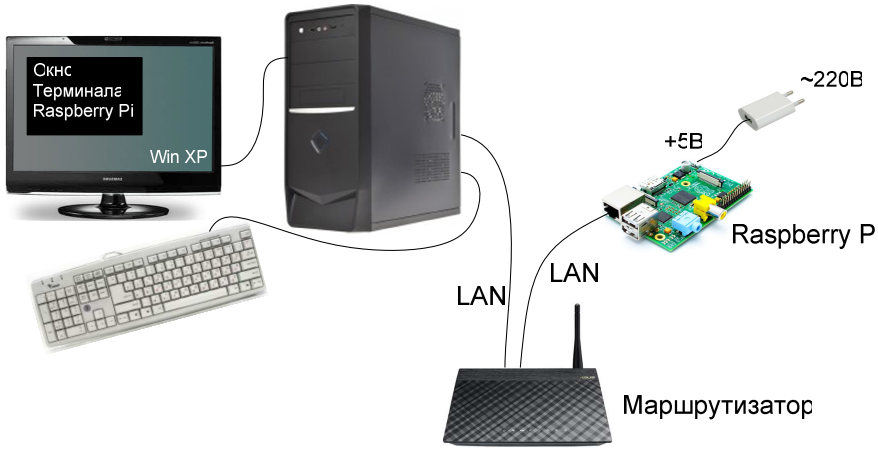
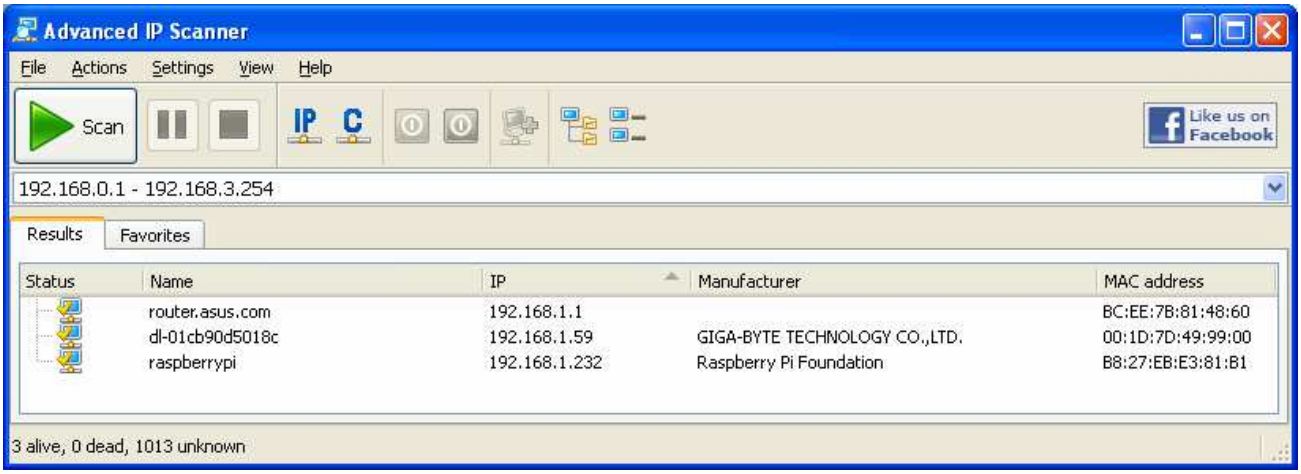


Рис. 11. Подключение к Raspberry Pi в режиме удаленного терминала.

2. Подключите питание 5В к RPi через разъём микроUSB. Наблюдая за светодиодами платы RPi дождитесь окончания загрузки операционной системы: дальний (от края платы) зеленый светодиод должен погаснуть.
3. На стационарном компьютере программой Advanced IP Scanner (ipscan23.exe) найдите IP адрес Raspberry Pi.



Примечание. Файл сетевых интерфейсов Raspberry Pi /etc/network/interfaces содержит указание на автоматическую конфигурацию по DHCP протоколу :

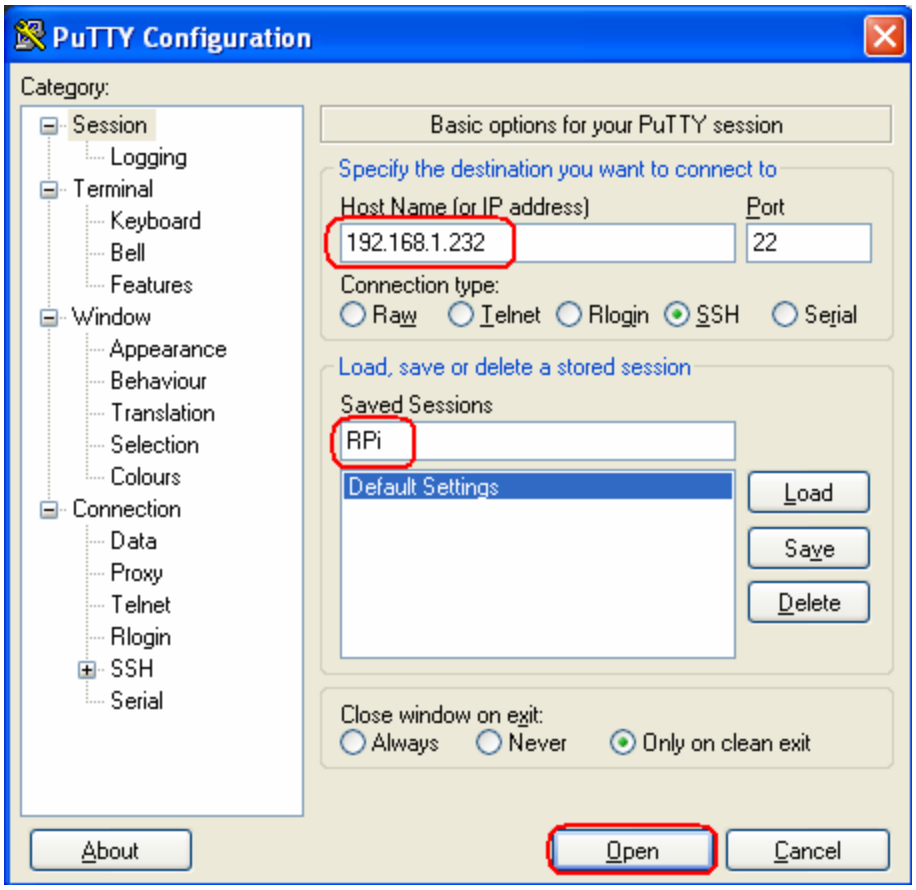
```
iface eth0 inet dhcp
```

Имя сервера в конфигурационном файле /etc/resolv.conf :

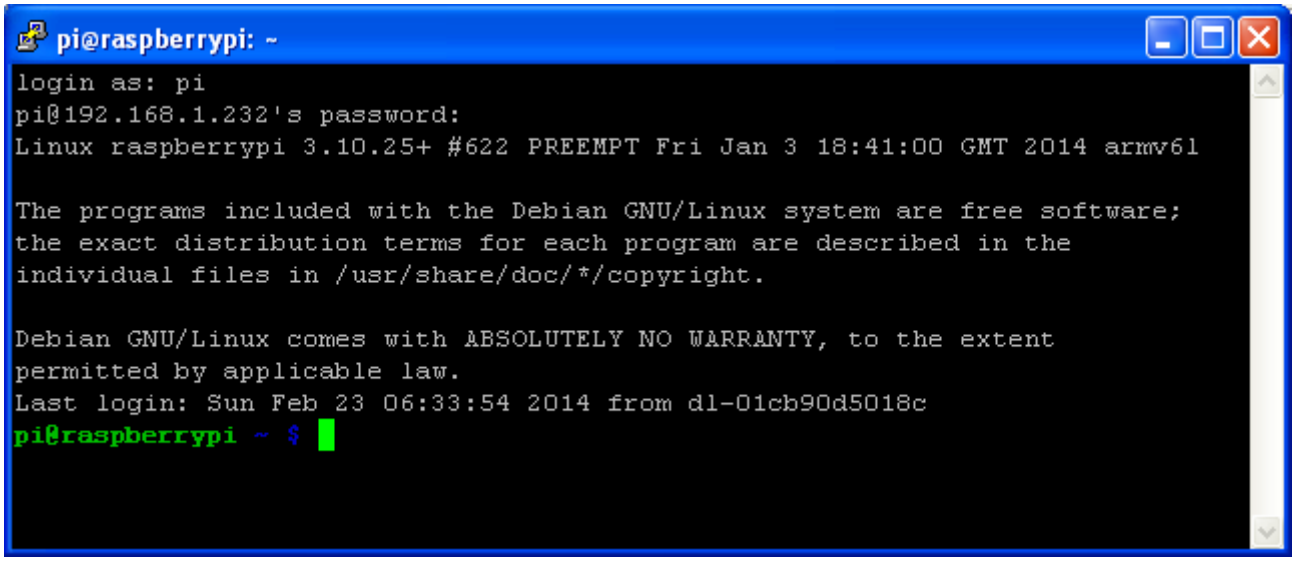
```
nameserver 8.8.8.8
```

При указанных выше параметрах, маршрутизатор самостоятельно назначает сетевые параметры компьютеру Raspberry.

4. Программой PuTTY connection (putty.exe) откройте терминал связи с Raspberry Pi.



5. Введите login: pi и password: raspberry



6. Введите команды операционной среды Raspberry Pi через терминал и наблюдайте за реакцией компьютера Raspberry. Например, рассмотрите выполнение следующих команд.

Команда консоли	Описание
cal	Вывод календаря на текущий месяц
cd	Переход в папку, например, cd /home/pi
Ctrl+C	Выход из открытой консольной программы, например, редактора nano
date	Вывод времени и даты
df -Bm или df -BM	Вывод распределения пространства SD карты
dir	Вывод содержимого текущей папки
help	Вывод списка команд консоли
ls	Вывод содержимого текущей папки и наличия исполняемых файлов
ping cxem.net	Проверка подключения к Интернет
pwd	Вывод пути текущего каталога
sudo	Ставится перед командой для ее выполнения с правами пользователя root
sudo nano	Вызов текстового редактора
sudo python	Запуск интерпретатора Python

7. Выключите компьютер Raspberry через терминал командой

   $ sudo shutdown –h now

8. Дождитесь полной остановки компьютера - когда останется гореть только один красный светодиод питания на плате RPi.

Задание 2. Интеграция АДА и С программных модулей.

1. Установите удаленное соединение с Raspberry Pi в режиме терминала (см. Задание 1).
2. Перейдите в директорию /home/pi
3. В редакторе nano наберите следующие С-программы (подпрограммы модуля АДА).

```
/* file1.c */
#include <stdio.h>

void print_num (int num)
{
```

```
printf ("num is %d.\n", num);
return;
}


/* file2.c */
/* num_from_Ada is declared in my_main.adb */
extern int num_from_Ada;

int get_num (void)
{
return num_from_Ada;
}
```

4. В редакторе nano наберите основной программный модуль на языке АДА.

```
--  my_main.adb
procedure My_Main is

-- Объявить и экспортировать целое число в num_from_Ada
My_Num : Integer := 10;
pragma Export (C, My_Num, "num_from_Ada");

--  Объявить Ada функцию Get_Num для использования C функции get_num
function Get_Num return Integer;
pragma Import (C, Get_Num, "get_num");

--  Объявить Ada процедуру Print_Num для использования C функции print_num
procedure Print_Num (Num : Integer);
pragma Import (C, Print_Num, "print_num");

begin
   Print_Num (Get_Num);
end My_Main;
```

5. Откомпилируйте программы на C чтобы получить объектный файлы file1.o и file2.o.

```
gcc -c file1.c
gcc -c file2.c
```

6. Откомпилируйте модуль Ada чтобы получить объектный файл my_main.o и ALI файл my_main.ali.

```
gnatmake -c my_main.adb
```

7. Запустите компоновщик (binder) на основной программе на Ada.

```
gnatbind my_main.ali
```

8. Редактором связей (Link) создайте исполняемый (exe) файл из объектных модулей Ada и C программы.

gnatlink my_main.ali file1.o file2.o

Примечание: Последние три команды можно заменить одной.

gnatmake my_main.adb -largs file1.o file2.o

9. Запустите программу.

   $ ./my_name

10. Измените аргументы программы АДА с С-модулями, откомпилируйте программу и и проверьте результат.

Задание 3. Управление с применением аппаратного ШИМ RPi.

1. Подключите светодиод к обесточенному компьютеру Raspberry Pi как показано на следующей схеме.

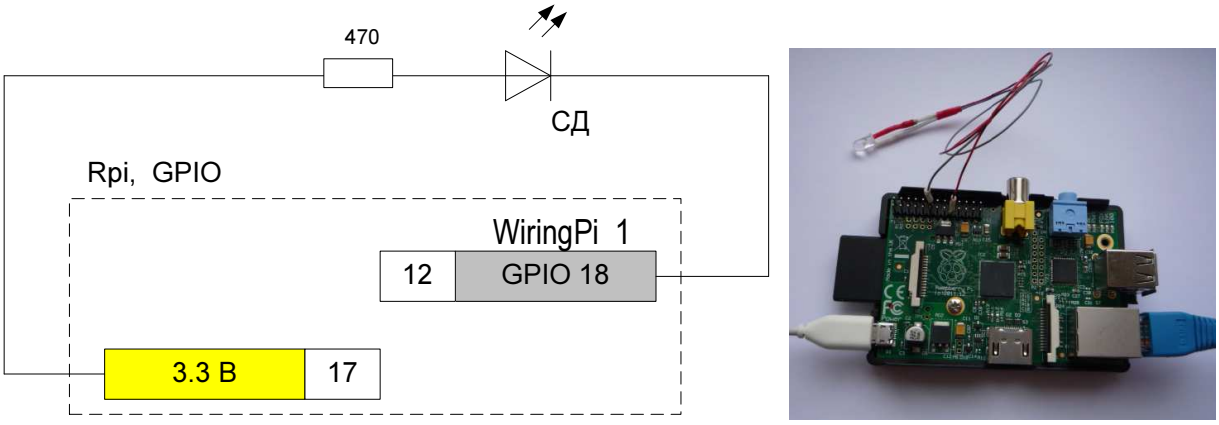


Рис. 12. Подключение светодиода к аппаратному ШИМ.

2. Включите RPi и установите с ним связь через удаленный терминал (см. Задание 1)
3. В редакторе nano напишите Python-программу изменения яркости светодиода подключенного к 12-му контакту разъема GPIO – аппаратному ШИМ.

```
import wiringpi

# GPIO pin 12 = BCM pin 18 = wiringpi pin 1
pwm_pin  = 1

wiringpi.wiringPiSetup()
wiringpi.pinMode(pwm_pin, 2)       # PWM mode

def pwm(pwm_value):
   wiringpi.pwmWrite(pwm_pin, pwm_value)
# value has to be in the range of 0 .. 1024

pwm(512)
```

4. Установите качественно зависимость яркости светодиода от кода ШИМ. Почему при уменьшении ШИМ светодиод горит ярче?

Задание 4. Проверка работоспособности канала измерения температуры.

1. К обесточенному Raspberry Pi подключите датчик температуры AD7416 (с I^2C интерфейсом) как показано на следующей схеме.

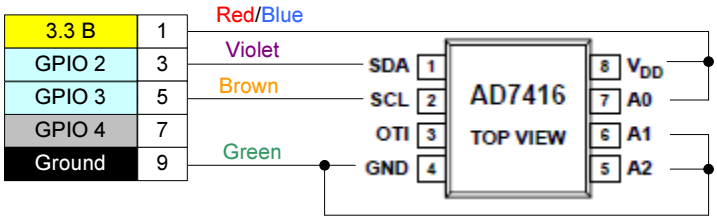


Рис. 13. Электрическая схема подключения датчика температуры AD7416 к RPi.

Рис. 14. Датчик температуры, подключенный к порту GPIO компьютера Raspberry Pi.

2. На удаленном компьютере установите связь с RPi через терминал как показано в Задании 1.
3. Активируйте I^2C интерфейс.

```
$ sudo modprobe i2c-dev
$ sudo chmod o+rw /dev/i2c-0
```

4. Проверьте частоту I^2C канала

```
dmesg
```

5. Установите частоту 100 кГц (если она отличается),

sudo modprobe -r i2c_bcm2708 && sudo modprobe i2c_bcm2708 baudrate=100000

6. Напишите на языке Python программу вывода показаний датчика температуры с интервалом 0, 5 секунды. I^2C адрес датчика температуры: 1001001_2 или 49_{16} или 56_{10}.

```
import smbus
import time

bus = smbus.SMBus(1)

while 1:
   a = bus.read_byte(0x49)
   print (a)
   time.sleep(0.5)
```

7. Проверьте зависимость выводимых показаний от изменения температуры датчика.
8. Остановите программу

   <Ctrl + C>

9. В редакторе nano напишите С-программу (например, tst_4b.c) считывания показаний I^2C температурного датчика (адрес 0x49) и отображения на экране терминала считанного кода датчика.

   $ sudo nano tst_4b.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <linux/i2c-dev.h>
#include <fcntl.h>

#define T_SENS_ADDRESS  (0x49)

int main()
{
   char buf[10];
   const char * devName = "/dev/i2c-1";

   // Open up the I2C bus
   int file = open(devName, O_RDWR);
   if (file == -1)
   {
      perror(devName);
      exit(1);
   }

   // Specify the address of the slave device.
   if (ioctl(file, I2C_SLAVE, T_SENS_ADDRESS) < 0)
   {
      perror("Failed to acquire bus access and/or talk to slave");
      exit(1);
```

```
    }

    // Read a byte from the slave.
    if (read(file,buf,1) != 1)
    {
        perror("Failed to read from the i2c bus");
        exit(1);
    }

    //printf("result: 0x%02X\n", buf[0]);
    printf("result: %d\n", buf[0]);

    return 0;
}
```

10. Выйдите из редактора и откомпилируйте С-программу

    $ sudo gcc -o tsk_4b tsk_4b.c -lrt -lbcm2835

11. Запустите исполняемый файл. Проверьте работоспособность программы.

    $ sudo ./tsk_4b

Задание 5. Проверка работоспособности канала релейного исполнительного устройства (лампового нагревателя).

1. Соберите систему термостатирования как показано на следующем рисунке и электрических схемах Рис. 13 и Рис. 16.

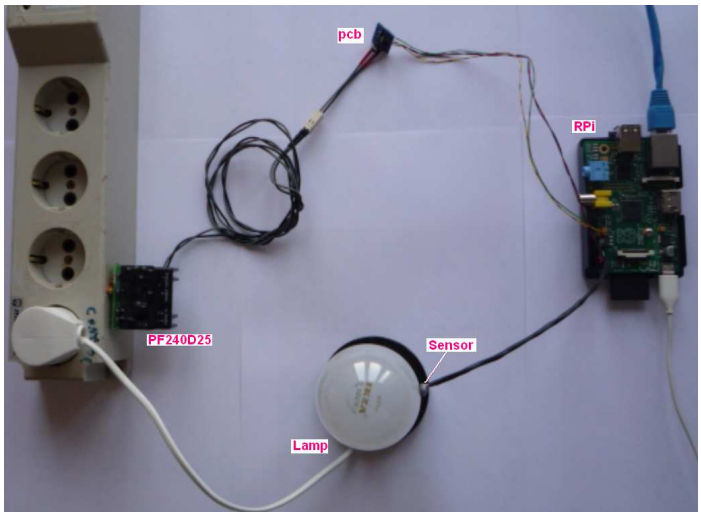


Рис. 15. Система термостатирования.

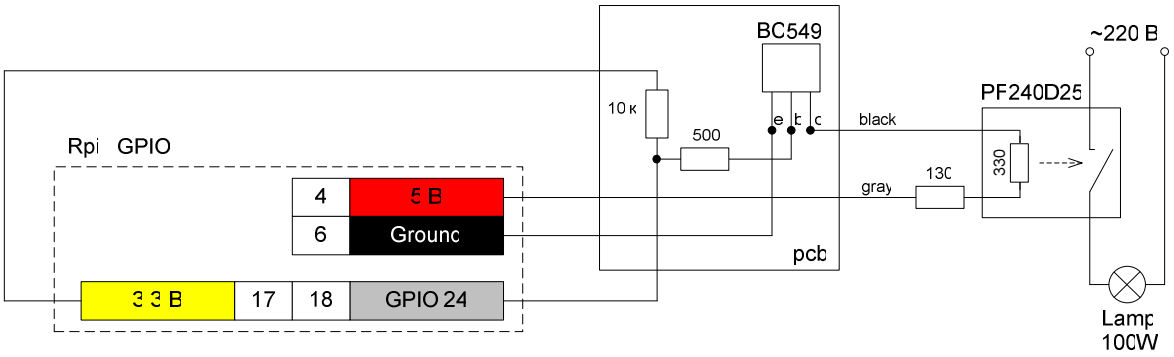


Рис. 16. Схема подключения нагревателя (100 Вт лампы) к выводам порта GPIO компьютера Raspberry Pi

2. Загрузите интерпретатор Python

   $ sudo python

3. Импортируйте библиотеку для работы с GPIO

   >>> import RPi.GPIO as GPIO

4. Установите способ нумерации выводов GPIO: BCM – нумерация выводов по логическим именам (не по номерам контактов разъема GPIO).

   >>>GPIO.setmode(GPIO.BCM)

5. Сконфигурируйте вывод GPIO на вывод – формирование выходного напряжения на контакте

   >>>GPIO.setup(24, GPIO.OUT)

6. Включите / выключите лампу

   >>>GPIO.output(24, GPIO.HIGH)
   или
   >>>GPIO.output(24, GPIO.LOW)

7. Завершите работу с GPIO.

   >>>GPIO.cleanup()

8. Выйдите из интерпретатора Python и остановите работу RPi.

   >>> exit()

   $ sudo shutdown – h now

Задание 6. Построение релейной системы термостатирования на базе Raspberry Pi.

1. В каталоге /home/pi RPi создайте рабочий каталог, например, st_18_02

   $ sudo mkdir st_18_02

2. Перейдите в рабочий каталог

   $ cd st_18_02

3. Напишите RT C-программу термостатирования в релейном режиме. Заданная температура должна на 5 градусов превышать температуру в момент запуска программы. Программа должна обеспечивать формирование RT такта, как показано на нижней диаграмме Рис. 7. Период квантования по времени должен быть нечувствителен к загрузке и выполнению других процессов. Времена начала тактов должны записываться в текстовый файл.

   Схемы подключения датчика температуры и нагревателя представлены на Рис. 13 и Рис. 16. Общий вид системы показан на Рис. 15.

```
#include <linux/i2c-dev.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
//#include <semaphore.h>
#include <sys/mman.h>
#include <signal.h>

#define T_SENS_ADDRESS  (0x49)
#define MAX_LOGENTRIES 200000

static unsigned int logindex;
static struct timespec timestamps[MAX_LOGENTRIES];
static void logtimestamp(void){
  clock_gettime(CLOCK_MONOTONIC, &timestamps[logindex++]);
  if(logindex > MAX_LOGENTRIES){
    logindex = 0;
  }
}
static void dumptimestamps(int unused){
  FILE *fp = fopen("timestamps.txt","w");
  int i;
  for(i=0; i < logindex; i++){
    if(timestamps[i].tv_sec > 0){
      fprintf(fp,"%d.%09d\n", (int) timestamps[i].tv_sec,
        (int) timestamps[i].tv_nsec);
    }
  }
  fclose(fp);
  exit(0);
}
// Initialize a GPIO pin in Linux using the sysfs interface
FILE *init_gpio(int gpioport){
  // Export the pin to the GPIO directory
  FILE *fp = fopen("/sys/class/gpio/export","w");
  fprintf(fp,"%d",gpioport);
  fclose(fp);
  // Set the pin as an output
```

```
  char filename[256];
  sprintf(filename,"/sys/class/gpio/gpio%d/direction",gpioport);
  fp = fopen(filename,"w");
  if(!fp){
    perror("Could not open gpio file");
  }
  fprintf(fp,"out");
  fclose(fp);
  // Open the value file and return a pointer to it.
  sprintf(filename,"/sys/class/gpio/gpio%d/value",gpioport);
  fp = fopen(filename,"w");
  if(!fp){
    perror("Could not open gpio file");
  }
fprintf(fp,"out");
  fclose(fp);
  // Open the value file and return a pointer to it.
  sprintf(filename,"/sys/class/gpio/gpio%d/value",gpioport);
  fp = fopen(filename,"w");
  if(!fp){
    perror("Could not open gpio file");
  }
        return fp;
}
// Given a FP in the stepper struct, set the I/O pin
// to the specified value. Uses the sysfs GPIO interface.
void setiopin(FILE *fp, int val) {
        fprintf(fp,"%d\n",val);
        fflush(fp);
}
// Adds "delay" nanoseconds to timespecs and sleeps until that time
static void sleep_until(struct timespec *ts, int delay)
{
  ts->tv_nsec += delay;
  if(ts->tv_nsec >= 1000*1000*1000){
    ts->tv_nsec -= 1000*1000*1000;
    ts->tv_sec++;
  }
  clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, ts, NULL);
}
// Demo RT program for temperature control
// stable against multitask mode and downloading in memory
// RT clock is 0.1 sec.  Raspberry PI platform.
int main(int argc, char **argv) {
  struct timespec ts;
  unsigned int delay = 100*1000*1000; // Delay in ns
  char buf[10];
  const char * devName = "/dev/i2c-1";

  FILE *pin18 = init_gpio(24);
  signal(SIGINT, dumptimestamps);
```

```
  clock_gettime(CLOCK_MONOTONIC, &ts);
  // Lock memory to ensure no swapping is done.
  if(mlockall(MCL_FUTURE|MCL_CURRENT)){
    fprintf(stderr,"WARNING: Failed to lock memory\n");
  }
  // Set our thread to real time priority
  struct sched_param sp;
  sp.sched_priority = 30;
  if(pthread_setschedparam(pthread_self(), SCHED_FIFO, &sp)){
    fprintf(stderr,"WARNING: Failed to set temperature thread"
       "to real-time priority\n");
  }
  // Open up the I2C bus
  int file = open(devName, O_RDWR);
  if (file == -1){
    perror(devName);
    exit(1);
  }
  // Specify the address of the slave device.
  if (ioctl(file, I2C_SLAVE, T_SENS_ADDRESS) < 0){
    perror("Failed to acquire bus access and/or talk to slave");
    exit(1);
  }

  // Read byte (initial temperature) from the slave.
  if (read(file,buf,1) != 1){
    perror("Failed to read from the i2c bus");
    exit(1);
  }

  char tgt_temp = buf[0]+5; // target temperature
  while(1){
    sleep_until(&ts,delay); logtimestamp();
    if (read(file,buf,1) != 1){
      perror("Failed to read from the i2c bus");
      exit(1);
    }
    if (buf[0] < tgt_temp){
      setiopin(pin18,1);
    }
    else{
      setiopin(pin18,0);
    }
  }
}
```

4. Выйдите из редактора, откомпилируйте С-программу, запустите исполняемый файл и проверьте работоспособность системы термостатирования.
5. Перенесите текстовый файл с накопленными данными из текущего каталога Raspberry Pi на компьютер с МатЛАБ используя USB-флеш карту или сетевое соединение МатЛАБ –

Raspberry Pi. Установка такого соединения показана в [5]. Для копирования файла через сетевое соединение используйте следующие команды МатЛАБ.

```
>> mypi = raspi()
>> openShell(mypi)
>> getFile(mypi,'/home/pi/st_18_02/timestamps.txt','C:\')
```

6. Импортируйте данные скопированного текстового файла в workspace МатЛАБ.
7. Сохраните данные текстового файла в dts.mat файле.

```
>> save('dts','VarName1')
```

8. Используя данные mat-файла напишите m-программу вычисления тактов реального времени системы термостатирования. Постройте график периода RT и вычислите среднее значение периода.

```
load('dts.mat')

figure
plot(diff(VarName1*1000));
grid on
xlabel('Cycle number');
ylabel('Cycle, ms');
title('Cycle stability');

(VarName1(length(VarName1))-VarName1(1))/length(VarName1)

ans = 0.099946
```

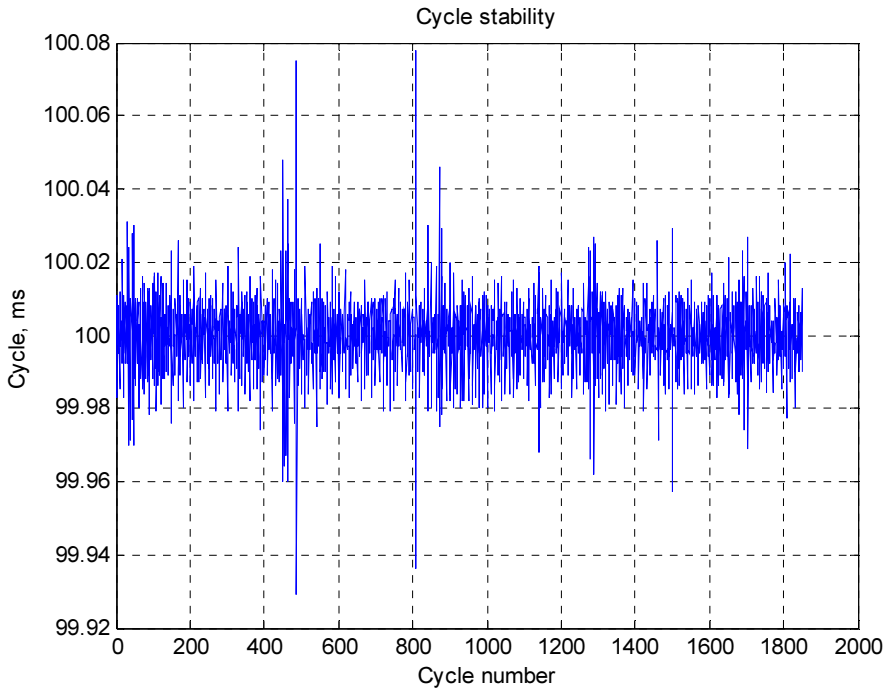


Рис. 17. Значение периода реального времени системы термостатирования на базе компьютера Raspberry Pi.

9. Доработайте программу термостатирования так, чтобы она записывала в текстовый файл текущее значение температуры. Постройте график температуры от времени.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Сравните реализации аппаратного и программного ШИМ raspberry Pi.
2. Почему в Задании 3 при уменьшении ШИМ светодиод горит ярче?
3. Какое преимущество имеет автономная система термостатирования релейного типа реализованная на базе датчика температуры (см. Рис. 5 и Рис. 6).
4. Назовите особенности I^2C интерфейса.
5. Перечислите основные проблемы реализации тактов систем реального времени на базе компьютера Raspberry Pi.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Установка библиотеки WiringPi - доступа к выводам GPIO, включая аппаратный ШИМ. https://github.com/WiringPi/WiringPi-Python
2. Andreas Ehliar. Realtime tasks in Linux: lecture_linux_realtime.pdf.
3. Dr. Bob Davidov. Компьютерные средства систем управления. Raspberry Pi (Computer facilities control systems. Raspberry Pi) http://portalnp.ru/2013/12/1691
4. Dr. Bob Davidov. Подключение периферии к среде разработки систем управления МатЛАБ (Connecting peripherals to Matlab) http://portalnp.ru/2014/03/1783
5. Dr. Bob Davidov. Импорт и экспорт МатЛАБ данных через Raspberry Pi http://portalnp.ru/2014/04/1858
6. Dr. Bob Davidov. Компьютерные технологии управления в технических системах http://portalnp.ru/author/bobdavidov
7. ГЛАВА 10 Семафоры Posix http://www.redov.ru/kompyutery_i_internet/unix_vzaimodeistvie_processov/p4.php