Широтно-импульсная модуляция в Raspberry Pi

А. ШИТОВ, г. Амстердам, Нидерланды

Микрокомпьютеры Raspberry Pi имеют возможность генерировать широтно-модулированные сигналы. В этой статье автор рассматривает два подхода к генерированию таких импульсов — использование встроенных аппаратных ШИМ-контроллеров и создание ШИМ-сигналов программными средствами — и описывает два варианта регуляторов (диммеров).

Широтно-модулированные сигналы удобно применить, например, для управления яркостью ламп или шаговым двигателем. Конструкции на основе Raspberry Pi позволяют использовать встроенные средства "системы на кристалле" и получить широтно-модулированный сигнал без каких-либо дополнительных внешних компонентов.

Аппаратная генерация

Микросхемы BCM2835—BCM2837, на основе которых собраны микрокомпьютеры Raspberry Pi, имеют двухканальный генератор ШИМ-сигналов, частота и скважность импульсов которых настраиваются программно. В частности, аудиовыход (выведенный на разъём 3,5 мм) использует оба канала для формирования стереофонического аудиосигнала. Однако ШИМ-генераторы можно использовать и для других нужд.

Дальнейшее описание подразумевает, что в конфигурации Raspberry Pi отключён аудиоканал и включены оба канала ШИМ. В файле /boot/config.txt должны быть, например, такие строки:

```
dtoverlay=pwm-2chan
#dtparam=audio=on
```

Знак # в начале строки означает, что эта строка закомментирована.

Выходы встроенных ШИМ-контроллеров могут быть выведены на какой-либо вывод GPIO, переключённый в один из альтернативных режимов. Особенности коммутации таковы, что только некоторые из выводов GPIO могут служить выходами генераторов ШИМ. В **таблице** показаны все допустимые комбинации и указан альтернативный режим, в который должен быть переведён вывод GPIO. Номера

Выход GPIO	Режим для вывода канала PWM0	Режим для вывода канала PWM1
12	ALT0	—
13	—	ALT0
18	ALT5	—
19	—	ALT5
40	ALT0	—
41	—	ALT0
45	—	ALT0
52	ALT1	—
53	—	ALT1

ШИМ-каналов начинаются с нуля: PWM0 и PWM1 (PWM — pulse width modulation).

Как видно из таблицы, функции аппаратного ШИМ доступны только на выходах GPIO12, GPIO13, GPIO18, GPIO19, GPIO40, GPIO41, GPIO45, GPIO52 и GPIO53. При этом в Raspberry Pi 1—Raspberry Pi 3 и Raspberry Zero доступны первые четыре выхода из этого списка (GPIO12, GPIO13, GPIO18 и GPIO19), а в Compute Module дополнительно ещё три (GPIO40, GPIO41 и GPIO45). На GPIO с чётными номерами может быть выведен только канал PWM0, а на GPIO с нечётными номерами — только PWM1.

Обратите внимание, что, независимо от коммутации GPIO, контроллеров ШИМ всего два. Поэтому, например, если перевести GPIO12 в режим ALT0, а GPIO18 в режим ALT5, то оба выхода окажутся подключёнными к одному и тому же генератору ШИМ.

Рассмотрим минимальную программу на языке C, с помощью которой можно получить на выходе широтно-модулированный сигнал. Программа использует библиотеку libbcm2835 (см. [1]). Ниже приведены фрагменты кода, а полная программа доступна в репозитории [2] в файле pwm0.cpp. Запускать программу следует от имени суперпользователя:

```
sudo ./a.out
```

Прежде всего необходимо перевести выход GPIO в режим работы с каналом PWM0:

```
int pin = 18;
bcm2835_gpio_fsel(pin,
 BCM2835_GPIO_FSEL_ALT5);
```

Далее устанавливают коэффициент деления задающего генератора (1024 в этом примере):

```
int divisor = 1024;
bcm2835_pwm_set_clock(divisor);
```

Теперь необходимо определиться с режимом формирования ШИМ. Их два, мы их рассмотрим чуть позже, а пока активируем режим 1 (этот режим называется Mark-Space). Номер режима указан во втором параметре функции bcm2835_pwm_set_mode; третий параметр разрешает генерацию сигнала:

```
int channel = 0;
bcm2835_pwm_set_mode(channel, 1, 1);
```

Наконец, устанавливаем параметры самого ШИМ-сигнала — длину цикла и длительность высокого уровня:

```
int range = 1000;
int data = 300;
bcm2835_pwm_set_range(channel,
 range);
bcm2835_pwm_set_data(channel, data);
```

В этом примере цикл будет состоять из 1000 отсчётов, в течение 300 из которых на выходе присутствует высокий уровень.

Важно отметить, что ШИМ-генераторы управляются от одного источника, поэтому установка частоты влияет на оба канала. Однако интервал (range) и уровень заполнения (data) могут быть настроены независимо. Например, на **рис. 3** показана ситуация, когда PWM0 формирует импульсы 300/1000 (синяя линяя), а PWM1 — 50/300 (красная линия). Оба генератора находятся в режиме Mark-Space. Второй генератор подключён к выводу GPIO19, полная программа для его настройки находится в файле pwm1.cpp.

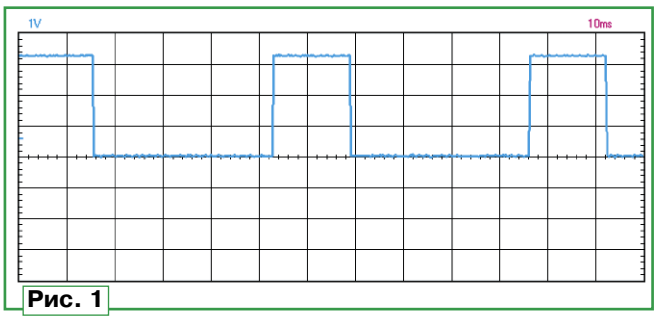

Рис. 1

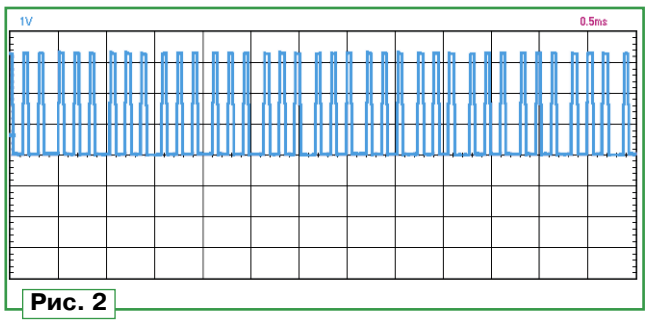

Рис. 2

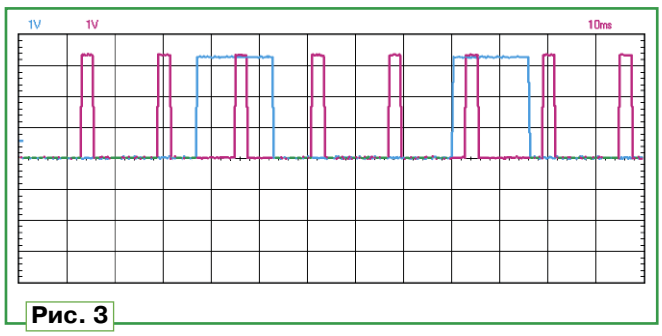

Рис. 3

Задающий генератор работает на частоте 19,2 МГц. При делении на 1024 получается 18,75 кГц, т. е. минимальный временной отсчёт составляет примерно 53,3 мкс. 300 таких отсчётов будут выводить единицу, а 700 — нули. Таким образом, полный цикл повторится через 53,3 мс, что соответствует частоте 18,75 Гц. Осциллограмма сигнала с вывода GPIO18 показана на **рис. 1**. Мерцание светодиода, подключённого (через токоограничительный резистор) к выводу GPIO18, будет заметно невооружённым глазом.

Теперь переключим генератор в режим по умолчанию (режим 0, Balanced), не изменяя при этом остальные параметры:

```
bcm2835_pwm_set_mode(channel, 0, 1);
```

В этом случае рабочий цикл, соответствующий частоте 18,75 кГц, состоит из относительно равномерно распределённых коротких импульсов. Осциллограмма такого сигнала показана на **рис. 2**. Обратите внимание на другой масштаб по оси времени (он указан в верхней правой части графика) и на то, что импульсы не всегда следуют равномерно. Суммарное время, в течение которого на выходе присутствует высокий уровень, по-прежнему равно 300/1000, как и в предыдущем примере, но мерцание светодиода уже не будет заметно.

Программная генерация

Второй вариант формирования ШИМ-сигналов — полностью программный. В программе создаётся цикл, который отсчитывает время и переводит соответствующий выход GPIO либо в единичное, либо в нулевое состояние.

Доступные библиотеки способны программно эмулировать широтно-импульсную модуляцию. В частности, — библиотека RPi.GPIO, написанная на языке Python [3].

Выбранный GPIO переводится в режим вывода (здесь номер 12 соответствует физическому номеру контакта GPIO18):

```
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.OUT)
```

Затем устанавливают параметры модуляции — частоту (500 Гц) и процент заполнения (30):

```
p = GPIO.PWM(12, 500)
p.start(30)
```

Для завершения генерации необходимо вызвать следующий метод:

```
p.stop()
```

Полный пример находится в программе soft-pwm.py.

Преимущество такого метода заключается в том, что возможно использовать любой из доступных выводов GPIO для получения независимого канала ШИМ. Кроме того, выводы не используют встроенные ШИМ-контроллеры, поэтому остаётся возможность генерировать звук (см., например, в [4]).

Однако существенным недостатком является невозможность точного контроля частоты формируемого сигнала. Если подключить осциллограф к выходу, можно сразу увидеть нестабильность: длительность импульсов и пауз,

хоть и незначительно, но постоянно меняется даже при отсутствии нагрузки на процессор. Обмен данными по сети или сложные вычисления ещё более ухудшают качество сигнала.

Ещё одно важное отличие — при использовании аппаратных ресурсов задача программы состоит лишь в том, чтобы записать в управляющие регистры соответствующие значения. После этого контроллер продолжает формировать сигнал самостоятельно. Программная же реализация требует постоянно запущенной программы — это может быть и единственный скрипт, как в вышеупомянутом примере, и демон, работающий в памяти отдельно от основного приложения.

Прежде чем мы сможем продолжить, необходимо сделать отступление и рассмотреть малоизвестные особенности работы GPIO в режиме чтения.

Входной гистерезис GPIO

В документации [5] к "системе на кристалле" BCM2835 упоминаются триггеры Шмитта, которые могут быть программно активированы для каждого входа GPIO. Чуть более подробную информацию можно найти в дополнительном документе [6], однако в нём речь идёт не про триггер Шмитта, а про управление гистерезисом. Попробуем разобраться и выяснить подробности.

Активирование режима гистерезиса на входе контролирует третий разряд трёх регистров PADS, отвечающих за входы GPIO0—GPIO27, GPIO28—GPIO45 и GPIO46—GPIO53. Гистерезис на входе включён, если в этот разряд записана единица. Таким образом, все входы GPIO объединены в три группы, в пределах которых изменение режима происходит одновременно.

Чтение из регистров не представляет сложности:

```
uint32_t* gpioBASE =
 bcm2835_regbase
(BCM2835_REGBASE_GPIO);
uint32_t pads = bcm2835_peri_read
(bcm2835_pads +
 BCM2835_PADS_GPIO_0_27 / 4);
cout << bitset<32>(pads) << endl;
```

Нас интересует только третий разряд. Для записи, однако, необходимо в старшие разряды занести значение 0x5A — это своего рода пароль от случайных изменений. В следующем фрагменте показан алгоритм установки третьего разряда в регистре, отвечающего за входы 0—27:

```
uint32_t new_pads = pads;
new_pads |= 1UL << 3;
new_pads &= 0x00FFFFFF;
new_pads |= 0x5A000000;
bcm2835_peri_write(bcm2835_pads +
 BCM2835_PADS_GPIO_0_27 / 4,
 new_pads);
```

Для отключения гистерезиса запишите в третий разряд ноль, для этого строку

```
new_pads |= 1UL << 3;
```

следует заметить такой:

```
new_pads &= ~(1UL << 3);
```

Полная программа находится в файле hysteresis.cpp [2].

Чтобы получить передаточную характеристику, были проведены измерения по простой схеме, показанной на **рис. 4**. Переменным резистором изменяют напряжение на входе GPIO в пределах 0,7...2,6 В. Логическое состояние входа отслеживается с помощью несложной программы (см., например, файл 2-level.cpp из [1]).

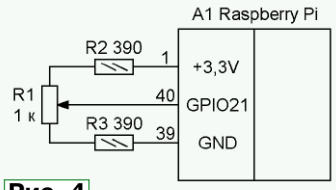

Рис. 4

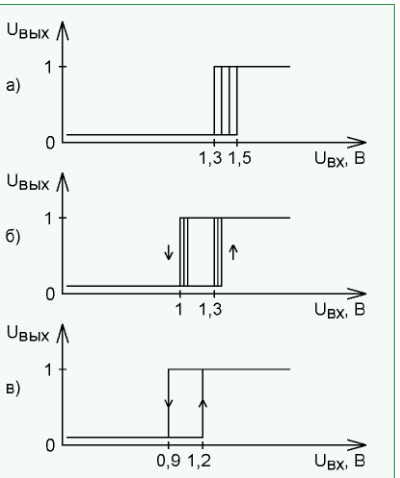

Рис. 5

При выключенном гистерезисе переключение между логическими нулём и единицей происходит при входном напряжении около 1,3...1,5 В (**рис. 5**,а). Отчётливо видно, что при плавном изменении входного напряжения переменным резистором вход переключается несколько раз, получая стабильное значение, когда входной сигнал выходит за пределы указанного интервала.

При включённом же гистерезисе картина несколько меняется (рис. 5,б). Теперь переключение в ноль происходит при снижении напряжения меньше 1 В, а переключение в единицу — когда напряжение превышает 1,3 В. Однако, к сожалению, при обеих сменах состояния по-прежнему наблюдаются нестабильные области, когда состояние входа многократно изменяется.

Чтобы улучшить ситуацию, достаточно добавить на вход настоящий триггер Шмитта, например, по схеме на **рис. 6**. Передаточная характеристика такого варианта показана на рис. 5,в — теперь изменение состояния происходит однозначно и без "дребезга".

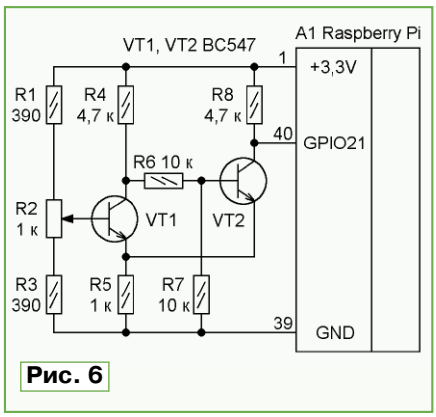

Рис. 6

Регулятор (диммер) для светодиодной ленты

Применим рассмотренную теорию на практике. На **рис. 7** показана схема устройства управления яркостью светодиодной ленты, которая питается от источника постоянного напряжения 24 В. В разрыв плюсового провода питания установлен р-канальный полевой транзистор VT2. Светодиодную ленту подключают с соблюдением полярности к разъёму XS1.

Программа (см. файл led-dimmer.cpp [2]) формирует на выходе GPIO18 импульсы частотой около 300 Гц, скважность которых можно менять кнопками SB1 и SB2.

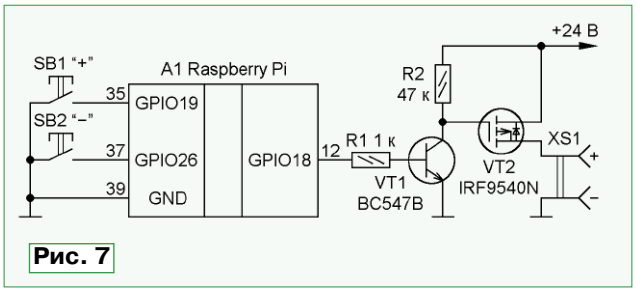

Рис. 7

В этой программе используется аппаратный ШИМ в режиме несбалансированных импульсов (это означает, например, что при коэффициенте заполнения 30 % формируется непрерывный высокий уровень в течение 30 % цикла). Весь цикл разбит на 300 шагов (т. е. значение range равно 300). При нажатии на кнопки "+" и "–" параметр data изменяется в пределах от 0 до 300, причём чем больше значение параметра, тем больше шаг изменения — этим обеспечивается более равномерное регулирование яркости во всём интервале:

```
int delta() {
    if (pwm_data < 15) return 1;
    if (pwm_data < 50) return 3;
    return 10;
}
```

Используя возможности Raspberry Pi как компьютера, несложно создать и веб-интерфейс для управления яркостью светодиодной ленты.

Детектор нуля

Современные модели Raspberry Pi с четырёхъядерным процессором, работающим на частоте более 1 ГГц, — довольно мощные компьютеры, которые весьма привлекательны в системах "умный дом". Однако следует помнить, что операционная система Raspbian не является системой реального времени, поэтому, например, применить Raspberry Pi для регулирования яркости лампы накаливания затруднительно — система будет непредсказуемо останавливать программу, выполняя другие действия. В качестве решения можно полностью запретить программные прерывания, однако тогда компьютер потеряет возможность общаться по сети (хотя способность управлять входами и выходами GPIO полностью сохранится). Библиотека для управления прерываниями находится по адресу [7].

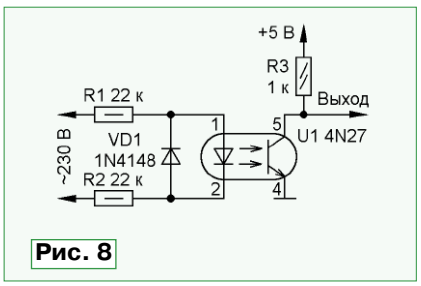

Рис. 8

Несколько лет назад, когда были доступны только Raspberry Pi первой модели, я собрал диммер на двух Raspberry Pi, которые обменивались информацией, используя несколько GPIO. Сегодня есть шанс получить работающий регулятор на одном устройстве (хотя лампы накаливания и выходят из употребления, но диммер сможет управлять и светодиодными лампами).

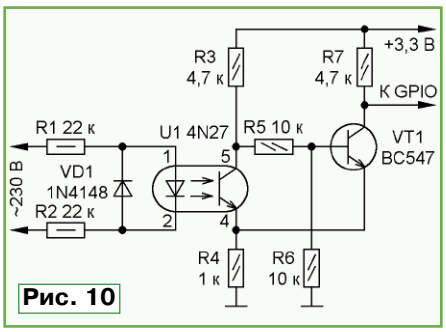

Рис. 10

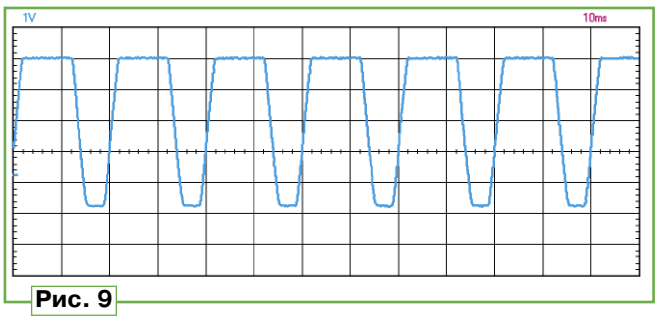

Рис. 9

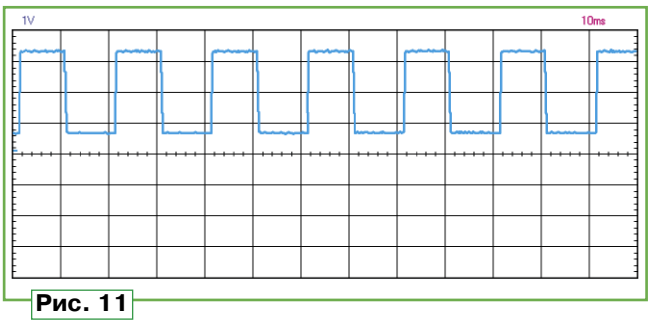

Рис. 11

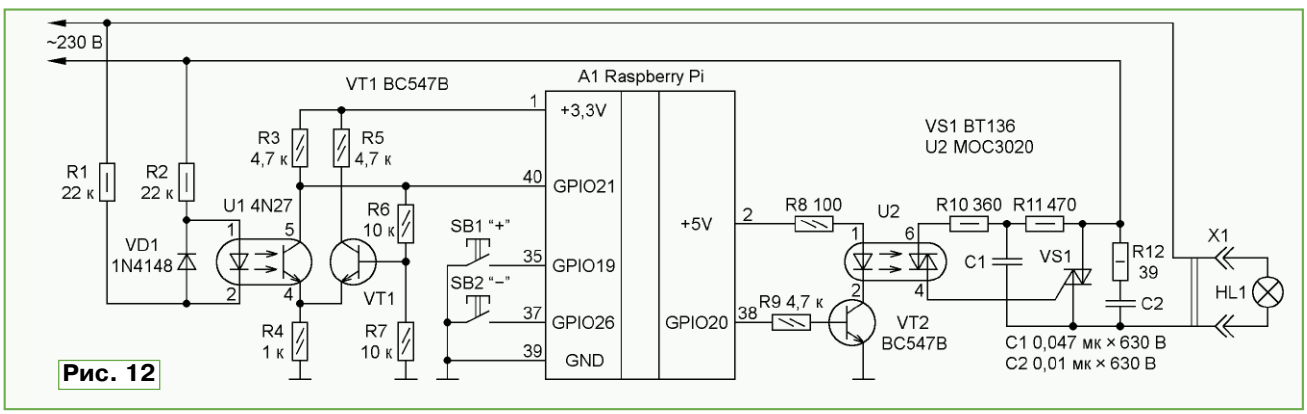

Рис. 12

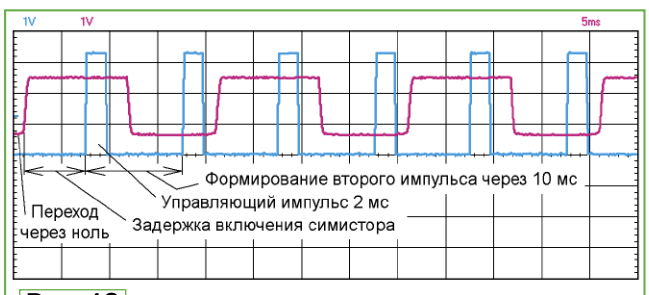

Рис. 13

В устройстве управления яркостью ламп, работающих на переменном токе, часто требуется узел, определяющий момент перехода сетевого напряжения через ноль. На **рис. 8** показана первая версия детектора (не подключайте его к Raspberry Pi, не уменьшив напряжение питания до 3,3 В). Излучающий диод оптрона включается в один из полупериодов сетевого напряжения, открывая транзистор. Диаграмма выходного напряжения показана на **рис. 9**. Как видно, выходные импульсы не имеют прямоугольной формы, поэтому непосредственное управление таким сигналом входами GPIO приведёт к нечётким срабатываниям, как мы видели выше.

Для решения проблемы достаточно собрать триггер Шмитта, причём одним из транзисторов может быть транзистор оптрона. Схема, показанная на **рис. 10**, формирует прямоугольные импульсы частотой 50 Гц (**рис. 11**).

Обратите внимание, что перепады выходного сигнала не обязательно соответствуют переходу сетевого напряжения через ноль. Однако это не является проблемой, поскольку компенсацию сдвига можно переложить на компьютер.

Чтобы убедиться, что устройство работает правильно и все импульсы гарантированно считываются, запустите программу frequency.cpp. Она измеряет входную частоту, измеряя время, за которое на вход приходит 1000 импульсов. Через 20 с программа должна вывести значение частоты, близкое к 50 Гц.

Регулятор (диммер) для лампы накаливания

Используя описанные выше компоненты и выходной каскад, показанный на рис. 13 в [1], можно собрать полноценный диммер для ламп накаливания. Его схема показана на **рис. 12**. Управляющая программа — в файле bulb-dimmer.cpp [2].

Приём статей: mail@radio.ru
Вопросы: consult@radio.ru

Аналогично светодиодному диммеру управление осуществляют двумя кнопками "+" (яркость больше) и "–" (яркость меньше).

В ответ на перепад входного напряжения на входе GPIO21 программа формирует на выходе GPIO20 импульс длительностью 2 мс, необходимый для открывания симистора VS1. Импульс появляется с задержкой 0...8 мс относительно перехода сетевого напряжения через ноль. Интересная особенность этого диммера в том, что оптрон на входе формирует импульсы частотой 50 Гц, а программа формирует в ответ два импульса с интервалом 10 мс. Таким образом, нет необходимости в диодном мосте перед оптроном U1. На **рис. 13** показаны осциллограммы входного (красная линия) и выходного (синяя) сигналов.

Устройство хорошо работает, пока процессор не сильно загружен. Однако, как только Raspberry Pi получает вычислительные задачи, лампа начинает непредсказуемо мигать. Этот недостаток удаётся полностью устранить, если применить Raspberry Pi с многоядерным процессором и выделить одно из ядер исключительно под управляющую программу диммера.

В моделях Raspberry Pi, собранных на "системе на кристалле" BCM2837 (например, Raspberry Pi 3), четыре ядра, нумеруемых от 0 до 3. Чтобы выделить одно из ядер, скажем, четвёртое (с номером 3), необходимо изменить настройку операционной системы — в файл /boot/cmdline.txt следует дописать следующий параметр:

```
isolcpus=3
```

После перезагрузки компьютера это ядро не будет использоваться ни операционной системой, ни прикладными программами. В этом можно убедиться, выполнив команду

```
taskset -cp 1
```

Эта команда покажет, на каких ядрах может выполняться процесс с номером 1 (вместо этого процесса можно посмотреть любой другой процесс). Вы должны увидеть значение "0-2".

Теперь следует запустить скомпилированную программу диммера так, чтобы она работала именно на четвёртом ядре. Для этого выполните команду

```
taskset 8 ./bulb-dimmer
```

Здесь параметр "8" — битовая маска 0b1000, в которой единичные разряды соответствуют ядрам (начиная с 0), которые вы разрешаете использовать под данную задачу.

При повторении диммера возможно сделать несколько модификаций. Во-первых, вход GPIO21 подключён к коллектору транзистора оптрона U1, а не к VT1, как требует классическая схема. Это позволило получить перепад напряжения, почти совпадающий с переходом сетевого напряжения через ноль. Однако может потребоваться введение программной задержки 0...10 мс в основном цикле программы.

Во-вторых, можно ещё более усилить обособленность программы, попробовав применить обработчик прерываний и реакцию на перепад напряжения на входе GPIO (см. [1]).

Имитация ПДУ

Микрокомпьютер Raspberry Pi способен справиться ещё с одной интересной задачей — эмуляцией сигналов пультов дистанционного управления с инфракрасным каналом.

Существует готовое решение — библиотека lirc (Linux Infrared Remote Control) [8], которая позволяет записывать с пульта ИК-сигналы и воспроизводить их через один из выходов GPIO. Для записи потребуется ИК-приёмник TSOP1738. Для воспроизведения достаточно подключить через транзисторный усилитель ИК-диод (нужно лишь убедиться, что его излучение находится в диапазоне, воспринимаемом контролируемым устройством).

На странице [9] можно найти готовые файлы с командами распространённых пультов, так что самостоятельно записывать сигнал потребуется только при работе с малораспространёнными устройствами.

В состав ОС Raspbian входит программный модуль ("оверлей") для работы с lirc. Его необходимо активировать, записав следующую строку в файл /boot/config.txt:

```
dtoverlay=lirc-rpi,gpio_in_ ↵
pin=23,gpio_out_pin=22
```

(***Прим. ред.*** *Знак ↵ в конце строки, выделенный красным цветом, обозначает перенос. Это сделано исключительно для удобства вёрстки статьи. Следующую строку следует набрать в предыдущей строке без знака ↵).*

Здесь указано, какие выводы GPIO выбраны для работы с ИК-приёмником (gpio_in_pin) и передатчиком (gpio_out_pin).

ЛИТЕРАТУРА

1. **Шитов А.** Использование портов ввода-вывода GPIO микрокомпьютера Raspberry Pi. — Радио, 2018, № 7, с. 24—28.

2. Исходные коды к статье. — URL: **https://github.com/ash/pwm-pi** (13.06.18).

3. RPi.GPIO. — URL: **https://pypi.org/project/RPi.GPIO/** (13.06.18).

4. **Шитов А.** Микрокомпьютеры Raspberry Pi Zero и Raspberry Compute Module. — Радио, 2018, № 8, с. 28—31.

5. BCM2835 ARM Peripherals, с. 139. — URL: **https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf** (13.06.18).

6. GPIO pads control. — URL: **https://matt.ucc.asn.au/mirror/electron/GPIO-Pads-Control2.pdf** (16.06.18).

7. Библиотека libraspio. — URL: **https://github.com/ash/libraspio** (17.06.18).

8. LIRC. — URL: **http://www.lirc.org** (17.06.18).

9. Remotes Database. — URL: **http://lirc-remotes.sourceforge.net/remotes-table.html** (17.06.18).