RPi GPIO Code Samples

From eLinux.org

The Raspberry Pi GPIOs can be controlled using many programming languages.

Contents

C

Examples in different C-Languages.

Direct register access

Gert van Loo & Dom, have provided (http://www.raspberrypi.org/forum/educational-applications/gertboard/page-4/#p31555) some tested code which accesses the GPIO pins through direct GPIO register manipulation in C-code. (Thanks to Dom for doing the difficult work of finding and testing the mapping.)

Note: For Raspberry Pi 2, change BCM2708_PERI_BASE to 0x3F000000 for the code to work.

Example GPIO code:

```
//
//  How to access GPIO registers from C-code on the Raspberry-Pi
//  Example program
//  15-January-2012
//  Dom and Gert
//  Revised: 15-Feb-2013


// Access from ARM Running Linux

#define BCM2708_PERI_BASE        0x20000000
#define GPIO_BASE                (BCM2708_PERI_BASE + 0x200000) /* GPIO controller */


#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>

#define PAGE_SIZE (4*1024)
#define BLOCK_SIZE (4*1024)

int  mem_fd;
void *gpio_map;

// I/O access
volatile unsigned *gpio;


// GPIO setup macros. Always use INP_GPIO(x) before using OUT_GPIO(x) or SET_GPIO_ALT(x,y)
#define INP_GPIO(g) *(gpio+((g)/10)) &= ~(7<<(((g)%10)*3))
#define OUT_GPIO(g) *(gpio+((g)/10)) |=  (1<<(((g)%10)*3))
#define SET_GPIO_ALT(g,a) *(gpio+(((g)/10))) |= (((a)<=3?(a)+4:(a)==4?3:2)<<(((g)%10)*3))

#define GPIO_SET *(gpio+7)  // sets   bits which are 1 ignores bits which are 0
#define GPIO_CLR *(gpio+10) // clears bits which are 1 ignores bits which are 0

#define GET_GPIO(g) (*(gpio+13)&(1<<g)) // 0 if LOW, (1<<g) if HIGH

#define GPIO_PULL *(gpio+37) // Pull up/pull down
#define GPIO_PULLCLK0 *(gpio+38) // Pull up/pull down clock

void setup_io();

void printButton(int g)
```

```
{
  if (GET_GPIO(g)) // !=0 <-> bit is 1 <- port is HIGH=3.3V
    printf("Button pressed!\n");
  else // port is LOW=0V
    printf("Button released!\n");
}

int main(int argc, char **argv)
{
  int g,rep;

  // Set up gpi pointer for direct register access
  setup_io();

  // Switch GPIO 7..11 to output mode

 /************************************************************************\
  * You are about to change the GPIO settings of your computer.          *
  * Mess this up and it will stop working!                               *
  * It might be a good idea to 'sync' before running this program        *
  * so at least you still have your code changes written to the SD-card! *
 \************************************************************************/

  // Set GPIO pins 7-11 to output
  for (g=7; g<=11; g++)
  {
    INP_GPIO(g); // must use INP_GPIO before we can use OUT_GPIO
    OUT_GPIO(g);
  }

  for (rep=0; rep<10; rep++)
  {
     for (g=7; g<=11; g++)
     {
       GPIO_SET = 1<<g;
       sleep(1);
     }
     for (g=7; g<=11; g++)
     {
       GPIO_CLR = 1<<g;
       sleep(1);
     }
  }

  return 0;

} // main


//
// Set up a memory regions to access GPIO
//
void setup_io()
{
   /* open /dev/mem */
   if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0) {
      printf("can't open /dev/mem \n");
      exit(-1);
   }

   /* mmap GPIO */
   gpio_map = mmap(
      NULL,             //Any adddress in our space will do
      BLOCK_SIZE,       //Map length
      PROT_READ|PROT_WRITE,// Enable reading & writting to mapped memory
      MAP_SHARED,       //Shared with other processes
      mem_fd,           //File to map
      GPIO_BASE         //Offset to GPIO peripheral
   );

   close(mem_fd); //No need to keep mem_fd open after mmap

   if (gpio_map == MAP_FAILED) {
      printf("mmap error %d\n", (int)gpio_map);//errno also set!
      exit(-1);
   }
```

```
   // Always use volatile pointer!
   gpio = (volatile unsigned *)gpio_map;


} // setup_io
```

GPIO Pull Up/Pull Down Register Example

```
   // enable pull-up on GPIO24&25
   GPIO_PULL = 2;
   short_wait();
   // clock on GPIO 24 & 25 (bit 24 & 25 set)
   GPIO_PULLCLK0 = 0x03000000;
   short_wait();
   GPIO_PULL = 0;
   GPIO_PULLCLK0 = 0;
```

WiringPi

Get and install wiringPi: http://wiringpi.com/download-and-install/

Save this, and compile with:

```
  gcc -o blink blink.c -lwiringPi
```

and run with:

```
  sudo ./blink
```

```
/*
 * blink.c:
 *      blinks the first LED
 *      Gordon Henderson, projects@drogon.net
 */

#include <stdio.h>
#include <wiringPi.h>

int main (void)
{
  printf ("Raspberry Pi blink\n") ;

  if (wiringPiSetup () == -1)
    return 1 ;

  pinMode (0, OUTPUT) ;         // aka BCM_GPIO pin 17

  for (;;)
  {
    digitalWrite (0, 1) ;       // On
    delay (500) ;               // mS
    digitalWrite (0, 0) ;       // Off
    delay (500) ;
  }
  return 0 ;
}
```

sysfs

The following example requires no special libraries, it uses the available sysfs interface.

```
/* blink.c
 *
 * Raspberry Pi GPIO example using sysfs interface.
 * Guillermo A. Amaral B. <g@maral.me>
 *
 * This file blinks GPIO 4 (P1-07) while reading GPIO 24 (P1_18).
 */

#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define IN  0
#define OUT 1

#define LOW  0
#define HIGH 1

#define PIN  24 /* P1-18 */
#define POUT 4  /* P1-07 */

static int
GPIOExport(int pin)
{
#define BUFFER_MAX 3
        char buffer[BUFFER_MAX];
        ssize_t bytes_written;
        int fd;

        fd = open("/sys/class/gpio/export", O_WRONLY);
        if (-1 == fd) {
                fprintf(stderr, "Failed to open export for writing!\n");
                return(-1);
        }

        bytes_written = snprintf(buffer, BUFFER_MAX, "%d", pin);
        write(fd, buffer, bytes_written);
        close(fd);
        return(0);
}

static int
GPIOUnexport(int pin)
{
        char buffer[BUFFER_MAX];
        ssize_t bytes_written;
        int fd;

        fd = open("/sys/class/gpio/unexport", O_WRONLY);
        if (-1 == fd) {
                fprintf(stderr, "Failed to open unexport for writing!\n");
                return(-1);
        }

        bytes_written = snprintf(buffer, BUFFER_MAX, "%d", pin);
        write(fd, buffer, bytes_written);
        close(fd);
        return(0);
}

static int
GPIODirection(int pin, int dir)
{
        static const char s_directions_str[]  = "in\0out";

#define DIRECTION_MAX 35
        char path[DIRECTION_MAX];
        int fd;

        snprintf(path, DIRECTION_MAX, "/sys/class/gpio/gpio%d/direction", pin);
```

```
        fd = open(path, O_WRONLY);
        if (-1 == fd) {
                fprintf(stderr, "Failed to open gpio direction for writing!\n");
                return(-1);
        }

        if (-1 == write(fd, &s_directions_str[IN == dir ? 0 : 3], IN == dir ? 2 : 3)) {
                fprintf(stderr, "Failed to set direction!\n");
                return(-1);
        }

        close(fd);
        return(0);
}

static int
GPIORead(int pin)
{
#define VALUE_MAX 30
        char path[VALUE_MAX];
        char value_str[3];
        int fd;

        snprintf(path, VALUE_MAX, "/sys/class/gpio/gpio%d/value", pin);
        fd = open(path, O_RDONLY);
        if (-1 == fd) {
                fprintf(stderr, "Failed to open gpio value for reading!\n");
                return(-1);
        }

        if (-1 == read(fd, value_str, 3)) {
                fprintf(stderr, "Failed to read value!\n");
                return(-1);
        }

        close(fd);

        return(atoi(value_str));
}

static int
GPIOWrite(int pin, int value)
{
        static const char s_values_str[] = "01";

        char path[VALUE_MAX];
        int fd;

        snprintf(path, VALUE_MAX, "/sys/class/gpio/gpio%d/value", pin);
        fd = open(path, O_WRONLY);
        if (-1 == fd) {
                fprintf(stderr, "Failed to open gpio value for writing!\n");
                return(-1);
        }

        if (1 != write(fd, &s_values_str[LOW == value ? 0 : 1], 1)) {
                fprintf(stderr, "Failed to write value!\n");
                return(-1);
        }

        close(fd);
        return(0);
}

int
main(int argc, char *argv[])
{
        int repeat = 10;

        /*
         * Enable GPIO pins
         */
        if (-1 == GPIOExport(POUT) || -1 == GPIOExport(PIN))
                return(1);

        /*
         * Set GPIO directions
```

```
         */
        if (-1 == GPIODirection(POUT, OUT) || -1 == GPIODirection(PIN, IN))
                return(2);

        do {
                /*
                 * Write GPIO value
                 */
                if (-1 == GPIOWrite(POUT, repeat % 2))
                        return(3);

                /*
                 * Read GPIO value
                 */
                printf("I'm reading %d in GPIO %d\n", GPIORead(PIN), PIN);

                usleep(500 * 1000);
        }
        while (repeat--);

        /*
         * Disable GPIO pins
         */
        if (-1 == GPIOUnexport(POUT) || -1 == GPIOUnexport(PIN))
                return(4);

        return(0);
}
```

bcm2835 library

This must be done as root. To change to the root user:

```
sudo -i
```

You must also get and install the bcm2835 library, which supports GPIO and SPI interfaces. Details and downloads from http://www.open.com.au/mikem/bcm2835

```
// blink.c
//
// Example program for bcm2835 library
// Blinks a pin on an off every 0.5 secs
//
// After installing bcm2835, you can build this
// with something like:
// gcc -o blink -l rt blink.c -l bcm2835
// sudo ./blink
//
// Or you can test it before installing with:
// gcc -o blink -l rt -I ../../src ../../src/bcm2835.c blink.c
// sudo ./blink
//
// Author: Mike McCauley (mikem@open.com.au)
// Copyright (C) 2011 Mike McCauley
// $Id: RF22.h,v 1.21 2012/05/30 01:51:25 mikem Exp $

#include <bcm2835.h>

// Blinks on RPi pin GPIO 11
#define PIN RPI_GPIO_P1_11

int main(int argc, char **argv)
{
    // If you call this, it will not actually access the GPIO
    // Use for testing
//    bcm2835_set_debug(1);

    if (!bcm2835_init())
```

```
        return 1;

    // Set the pin to be an output
    bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_OUTP);

    // Blink
    while (1)
    {
        // Turn it on
        bcm2835_gpio_write(PIN, HIGH);

        // wait a bit
        delay(500);

        // turn it off
        bcm2835_gpio_write(PIN, LOW);

        // wait a bit
        delay(500);
    }

    return 0;
}
```

pigpio

pigpio provides all the standard GPIO features.

In addition it provides hardware timed PWM suitable for servos, LEDs, and motors and samples/timestamps GPIOs 0-31 up to 1 million times per second (default 200 thousand).

C documentation: http://abyz.co.uk/rpi/pigpio/cif.html

Download and install pigpio: http://abyz.co.uk/rpi/pigpio/download.html

```
/*
   pulse.c

   gcc -o pulse pulse.c -lpigpio -lrt -lpthread

   sudo ./pulse
*/

#include <stdio.h>

#include <pigpio.h>

int main(int argc, char *argv[])
{
   double start;

   if (gpioInitialise() < 0)
   {
      fprintf(stderr, "pigpio initialisation failed\n");
      return 1;
   }

   /* Set GPIO modes */
   gpioSetMode(4, PI_OUTPUT);
   gpioSetMode(17, PI_OUTPUT);
   gpioSetMode(18, PI_OUTPUT);
   gpioSetMode(23, PI_INPUT);
   gpioSetMode(24, PI_OUTPUT);

   /* Start 1500 us servo pulses on GPIO4 */
   gpioServo(4, 1500);

   /* Start 75% dutycycle PWM on GPIO17 */
```

```
   gpioPWM(17, 192); /* 192/255 = 75% */

   start = time_time();

   while ((time_time() - start) < 60.0)
   {
      gpioWrite(18, 1); /* on */

      time_sleep(0.5);

      gpioWrite(18, 0); /* off */

      time_sleep(0.5);

      /* Mirror GPIO24 from GPIO23 */
      gpioWrite(24, gpioRead(23));
   }

   /* Stop DMA, release resources */
   gpioTerminate();

   return 0;
}
```

Compile

```
gcc -o pulse pulse.c -lpigpio -lrt -lpthread
```

Run

```
sudo ./pulse
```

C#

RaspberryGPIOManager is a very basic C# library to control the GPIO pins via the GPIOPinDriver object. See: https://github.com/AlexSartori/RaspberryGPIOManager

```
using RaspberryGPIOManager;

namespace GPIOtest
{
    class Program
    {
        static void Main(string[] args)
        {
            GPIOPinDriver led1;

            // Create the object.
            led1 = new GPIOPinDriver(GPIOPinDriver.Pin.GPIO15);

            // Set it as an output pin.
            led1.Direction = GPIOPinDriver.GPIODirection.Out

            // Give it power.
            led1.State = GPIOPinDriver.GPIOState.High;
        }
    }
}
```

RaspberryPiDotNet library is available at https://github.com/cypherkey/RaspberryPi.Net/. This more advanced library includes a GPIOFile and GPIOMem class. The GPIOMem requires compiling Mike McCauley's bcm2835 library above in to a shared object.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using RaspberryPiDotNet;
using System.Threading;

namespace RaspPi
{
    class Program
    {
        static void Main(string[] args)
        {
            // Access the GPIO pin using a static method
            GPIOFile.Write(GPIO.GPIOPins.GPIO00, true);

            // Create a new GPIO object
            GPIOMem gpio = new GPIOMem(GPIO.GPIOPins.GPIO01);
            gpio.Write(false);
        }
    }
}
```

Ruby

This example uses the WiringPi Ruby Gem: http://pi.gadgetoid.co.uk/post/015-wiringpi-now-with-serial which you can install on your Pi with "gem install wiringpi"

```
MY_PIN = 1

require 'wiringpi'
io = WiringPi::GPIO.new
io.mode(MY_PIN,OUTPUT)
io.write(MY_PIN,HIGH)
io.read(MY_PIN)
```

Alternatively the Pi Piper Gem (https://github.com/jwhitehorn/pi_piper) allows for event driven programming:

```
require 'pi_piper'
include PiPiper

watch :pin => 23 do
  puts "Pin changed from #{last_value} to #{value}"
end

PiPiper.wait
```

Perl

This must be done as root. To change to the root user:

```
sudo su -
```

Supports GPIO and SPI interfaces. You must also get and install the bcm2835 library. Details and downloads from http://www.open.com.au/mikem/bcm2835 You must then get and install the Device::BCM2835 perl library from CPAN http://search.cpan.org/~mikem/Device-BCM2835-1.0/lib/Device/BCM2835.pm

```
use Device::BCM2835;
use strict;

# call set_debug(1) to do a non-destructive test on non-RPi hardware
#Device::BCM2835::set_debug(1);
Device::BCM2835::init()
  || die "Could not init library";

# Blink pin 11:
# Set RPi pin 11 to be an output
Device::BCM2835::gpio_fsel(&Device::BCM2835::RPI_GPIO_P1_11,
                           &Device::BCM2835::BCM2835_GPIO_FSEL_OUTP);

while (1)
{
    # Turn it on
    Device::BCM2835::gpio_write(&Device::BCM2835::RPI_GPIO_P1_11, 1);
    Device::BCM2835::delay(500); # Milliseconds
    # Turn it off
    Device::BCM2835::gpio_write(&Device::BCM2835::RPI_GPIO_P1_11, 0);
    Device::BCM2835::delay(500); # Milliseconds
}
```

Python

RPi.GPIO

The RPi.GPIO module is installed by default in Raspbian. Any RPi.GPIO script must be run as root.

```
import RPi.GPIO as GPIO

# use P1 header pin numbering convention
GPIO.setmode(GPIO.BOARD)

# Set up the GPIO channels - one input and one output
GPIO.setup(11, GPIO.IN)
GPIO.setup(12, GPIO.OUT)

# Input from pin 11
input_value = GPIO.input(11)

# Output to pin 12
GPIO.output(12, GPIO.HIGH)

# The same script as above but using BCM GPIO 00..nn numbers
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.IN)
GPIO.setup(18, GPIO.OUT)
input_value = GPIO.input(17)
GPIO.output(18, GPIO.HIGH)
```

More documentation is available at http://sourceforge.net/p/raspberry-gpio-python/wiki/Home/

pigpio

pigpio Python scripts may be run on Windows, Macs, and Linux machines. Only the pigpio daemon needs to be running on the Pi.

pigpio provides all the standard gpio features.

In addition it provides hardware timed PWM suitable for servos, LEDs, and motors and samples/timestamps gpios 0-31 up to 1 million times per second (default 200 thousand).

Python documentation: http://abyz.co.uk/rpi/pigpio/python.html

Download and install pigpio: http://abyz.co.uk/rpi/pigpio/download.html

```
#!/usr/bin/env python

# pulse.py

import time

import pigpio

pi = pigpio.pi() # Connect to local Pi.

# set gpio modes

pi.set_mode(4, pigpio.OUTPUT)
pi.set_mode(17, pigpio.OUTPUT)
pi.set_mode(18, pigpio.OUTPUT)
pi.set_mode(23, pigpio.INPUT)
pi.set_mode(24, pigpio.OUTPUT)


# start 1500 us servo pulses on gpio4

pi.set_servo_pulsewidth(4, 1500)

# start 75% dutycycle PWM on gpio17

pi.set_PWM_dutycycle(17, 192) # 192/255 = 75%

start = time.time()

while (time.time() - start) < 60.0:

      pi.write(18, 1) # on

      time.sleep(0.5)

      pi.write(18, 0) # off

      time.sleep(0.5)

      # mirror gpio24 from gpio23

      pi.write(24, pi.read(23))

pi.set_servo_pulsewidth(4, 0) # stop servo pulses

pi.set_PWM_dutycycle(17, 0) # stop PWM

pi.stop() # terminate connection and release resources
```

pigpio scripts require that the pigpio daemon be running.

```
sudo pigpiod
```

They do not need to be run as root.

```
./pulse.py
```

RPIO

Also available is RPIO at https://pypi.python.org/pypi/RPIO

RPIO extends RPi.GPIO with TCP socket interrupts, command line tools and more.

WiringPi2-Python

There's a Python-wrapped version of Gordon Henderson's WiringPi. See here (https://github.com/WiringPi/WiringPi2-Python).

```
import wiringpi2

#wiringpi2.wiringPiSetup() # For sequential pin numbering, one of these MUST be called before using IO functior
# OR
#wiringpi2.wiringPiSetupSys() # For /sys/class/gpio with GPIO pin numbering
# OR
wiringpi2.wiringPiSetupGpio() # For GPIO pin numbering

wiringpi2.pinMode(6,1) # Set pin 6 to 1 ( OUTPUT )
wiringpi2.digitalWrite(6,1) # Write 1 ( HIGH ) to pin 6
wiringpi2.digitalRead(6) # Read pin 6
```

Scratch

Scratch using the ScratchGPIO

Scratch can be used to control the GPIO pins using a background Python handler available from

http://cymplecy.wordpress.com/2013/04/22/scratch-gpio-version-2-introduction-for-beginners/

Pridopia Scratch Rs-Pi-GPIO driver

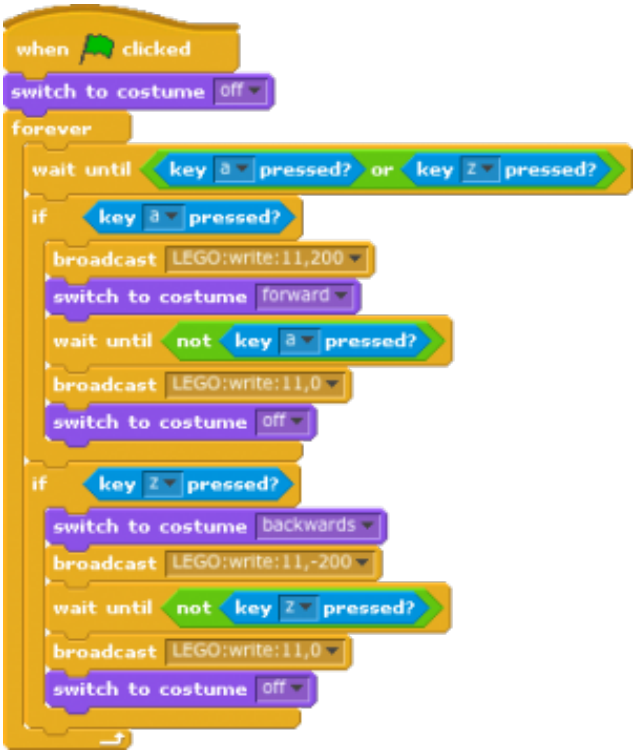
Scratch control GPIO (use GPIO number not P1 pin number can support GPIO 28,29,30,31)

support I²C 23017 8/16/32/64/128 GPIO, I²C TMP102 Temp sensor, I²C RTC DS1307, I²C ADC ADS1015, I²C PWM, I²C EEPROM 24c32, I²C BMP085 Barometric Pressure/Temperature/Altitude Sensor, GPIO input/output, DC motor, Relay, I²C 16x16 LED matrix, I²C 24x16 Matrix, 84x48 pixels LCD, 16x2 character LCD, 20x4 character LCD, 1-Wire 18B20 Temp Sensor, Ultra Sonic distance sensor, available from

http://www.pridopia.co.uk/rs-pi-set-scratch.html

RpiScratchIO

Generic interface for GPIO or other I/O operations. The package allows user modules to be easily added and loaded, to interface with any I/O device. The code is written in Python and uses the scratchpy Python package to interface with Scratch.



RpiScratchIO - Installation

To install the package on the latest Raspbian installation type,

```
sudo apt-get install -y python-setuptools python-dev
sudo easy_install pip
sudo pip install RpiScratchIO
```

RpiScratchIO - Documentation and examples

More information can be found at https://pypi.python.org/pypi/RpiScratchIO/ The package is also documented in Issues 20 (http://www.themagpi.com/issue/issue-20/) and 22 (http://www.themagpi.com/issue/issue-22/) of The MagPi (http://www.themagpi.com/). RpiScratchIO is the basis of a new BrickPi Scratch handler (https://pypi.python.org/pypi/BrickPi/), which is documented in Issue 23 (http://www.themagpi.com/issue/issue-23/) of The MagPi.

Java

Java using the Pi4J Library

This uses the Java library available at http://www.pi4j.com/. (Any Java application that controls GPIO must be run as root.)

Please note that the Pi4J library uses the WiringPi GPIO pin numbering scheme [1] [2]. Please see the usage documentation for more details: http://pi4j.com/usage.html

```
public static void main(String[] args) {

    // create gpio controller
    GpioController gpio = GpioFactory.getInstance();

    // provision gpio pin #01 as an output pin and turn off
    GpioPinDigitalOutput outputPin = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_01, "MyLED", PinState.LOW);

    // turn output to LOW/OFF state
    outputPin.low();

    // turn output to HIGH/ON state
    outputPin.high();


    // provision gpio pin #02 as an input pin with its internal pull down resistor enabled
    GpioPinDigitalInput inputPin = gpio.provisionDigitalInputPin(RaspiPin.GPIO_02, "MyButton", PinPullResistanc

    // get input state from pin 2
    boolean input_value = inputPin.isHigh();
}
```

More complete and detailed examples are included on the Pi4J website at http://www.pi4j.com/.

The Pi4J library includes support for:

- GPIO Control
- GPIO Listeners
- Serial Communication
- I2C Communication
- SPI Communication

Java

This uses the Java library available at https://github.com/jkransen/framboos. It does not depend on (or use) the wiringPi driver, but uses the same numbering scheme. Instead it uses the default driver under /sys/class/gpio that ships with the distro, so it works out of the box. Any Java application that controls GPIO must be run as root.

```
public static void main(String[] args) {
  // reading from an in pin
  InPin button = new InPin(8);
  boolean isButtonPressed = button.getValue();
  button.close();

  // writing to an out pin
  OutPin led = new Outpin(0);
  led.setValue(true);
  led.setValue(false);
  led.close();
}
```

Java Webapp GPIO web control via HTTP

This uses the Java Webapp available at https://bitbucket.org/sbub/raspberry-pi-gpio-web-control/overview. You can control your GPIO over the Internet. Any Java application that controls GPIO must be run as root.

```
host:~ sb$ curl 'http://raspberrypi:8080/handle?g0=1&g1=0'
{"g1":0,"g0":1}
```

Shell

sysfs, part of the raspbian operating system

For operating system versions prior to the raspbian Jessie release, the export and unexport of pins must be done as root. Since the raspbian Jessie release the pi user is a member of the group "gpio" and so control of the GPIO no longer requires a change to the root user. With Jessie, if using a script as in the code below, you'll need to put a 'sleep 1' command in between your 'export' and 'direction' commands to allow time for the operating system to set up the GPIO number specific direction file.

```
#!/bin/sh

# GPIO numbers should be from this list
# 0, 1, 4, 7, 8, 9, 10, 11, 14, 15, 17, 18, 21, 22, 23, 24, 25

# Note that the GPIO numbers that you program here refer to the pins
# of the BCM2835 and *not* the numbers on the pin header.
# So, if you want to activate GPIO7 on the header you should be
# using GPIO4 in this script. Likewise if you want to activate GPIO0
# on the header you should be using GPIO17 here.

# Set up GPIO 4 and set to output
echo "4" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio4/direction

# Set up GPIO 7 and set to input
echo "7" > /sys/class/gpio/export
echo "in" > /sys/class/gpio/gpio7/direction

# Write output
echo "1" > /sys/class/gpio/gpio4/value

# Read from input
cat /sys/class/gpio/gpio7/value

# Clean up
echo "4" > /sys/class/gpio/unexport
echo "7" > /sys/class/gpio/unexport
```

wiringPi - gpio utility

You need the wiringPi library from https://projects.drogon.net/raspberry-pi/wiringpi/download-and-install/. Once installed, there is a new command **gpio** which can be used as a **non-root** user to control the GPIO pins.

The man page

```
man gpio
```

has full details, but briefly:

```
  gpio -g mode 17 out
  gpio -g mode 18 pwm

  gpio -g write 17 1
  gpio -g pwm 18 512
```

The **-g** flag tells the **gpio** program to use the BCM GPIO pin numbering scheme (otherwise it will use the wiringPi numbering scheme by default).

The gpio command can also control the internal pull-up and pull-down resistors:

```
  gpio -g mode 17 up
```

This sets the pull-up resistor - however any change of mode, even setting a pin that's already set as an input to an input will remove the pull-up/pull-down resistors, so they may need to be reset.

Additionally, it can export/un-export the GPIO devices for use by other non-root programms - e.g. Python scripts. (Although you may need to drop the calls to GPIO.Setup() in the Python scripts, and do the setup separately in a little shell script, or call the **gpio** program from inside Python).

```
  gpio export 17 out
  gpio export 18 in
```

These exports GPIO-17 and sets it to output, and exports GPIO-18 and sets it to input.

And when done:

```
  gpio unexport 17
```

The export/unexport commands always use the BCM GPIO pin numbers regardless of the presence of the **-g** flag or not.

If you want to use the internal pull-up/down's with the /sys/class/gpio mechanisms, then you can set them after exporting them. So:

```
  gpio -g export 4 in
  gpio -g mode 4 up
```

You can then use GPIO-4 as an input in your Python, Shell, Java, etc. programs without the use of an external resistor to pull the pin high. (If that's what you were after - for example, a simple push button switch taking the pin to ground.)

A fully working example of a shell script using the GPIO pins can be found at http://project-downloads.drogon.net/files/gpioExamples/tuxx.sh.

pigpio - pigs utility

pigpio provides a command line utility pigs.

pigs provides all the standard gpio features.

In addition it provides hardware timed PWM suitable for servos, LEDs, and motors.

Use man pigs or view the pigs documentation at http://abyz.co.uk/rpi/pigpio/pigs.html

Download and install pigpio: http://abyz.co.uk/rpi/pigpio/download.html

Additional error information may be reported to /dev/pigerr. The following command will display any errors.

```
cat /dev/pigerr &
```

Examples

```
pigs modes 4 w # set gpio4 as output (write)
pigs modes 17 w # set gpio17 as output (write)
pigs modes 23 r # set gpio23 as input (read)
pigs modes 24 0 # set gpio24 as ALT0

pigs servo 4 1500 # start 1500 us servo pulses on gpio4

pigs pwm 17 192 # start 75% dutycycle PWM on gpio17 (192/255 = 75%)

pigs r 23 # read gpio23

pigs w 24 1 # write 1 to gpio 24

# servo may be abbreviated to s
# pwm may be abbreviated to p
# modes may be abbreviated to m

pigs s 4 0 # stop servo pulses

pigs p 17 0 # stop pwm
```

pigs requires that the pigpio daemon be running.

```
sudo pigpiod
```

For an example pigs script see x_pigs in the pigpio archive.

pigpio - /dev/pigpio interface

pigpio provides command line access via the /dev/pigpio pipe.

/dev/pigpio provides all the standard gpio features.

In addition it provides hardware timed PWM suitable for servos, LEDs, and motors.

The command set is identical to that used by pigs. Use man pigs or view the pigs documentation at http://abyz.co.uk/rpi/pigpio/pigs.html

Download and install pigpio: http://abyz.co.uk/rpi/pigpio/download.html

The result of /dev/pigpio commands are written to /dev/pigout. The following command will display the results.

```
cat /dev/pigout &
```

Errors are reported to /dev/pigerr. The following command will display any errors.

```
cat /dev/pigerr &
```

Examples

```
echo modes 4 w >/dev/pigpio # set gpio4 as output (write)
echo modes 17 w  >/dev/pigpio # set gpio17 as output (write)
echo modes 23 r  >/dev/pigpio # set gpio23 as input (read)
echo modes 24 0  >/dev/pigpio # set gpio24 as ALT0

echo servo 4 1500  >/dev/pigpio # start 1500 us servo pulses on gpio4

echo pwm 17 192  >/dev/pigpio # start 75% dutycycle PWM on gpio17 (192/255 = 75%)

echo r 23  >/dev/pigpio # read gpio23

echo w 24 1  >/dev/pigpio # write 1 to gpio 24

# servo may be abbreviated to s
# pwm may be abbreviated to p
# modes may be abbreviated to m

echo s 4 0  >/dev/pigpio # stop servo pulses

echo p 17 0  >/dev/pigpio # stop pwm
```

The /dev/pigpio pipe interface requires that the pigpio daemon be running.

```
sudo pigpiod
```

For an example /dev/pigpio pipe script see x_pipe in the pigpio archive.

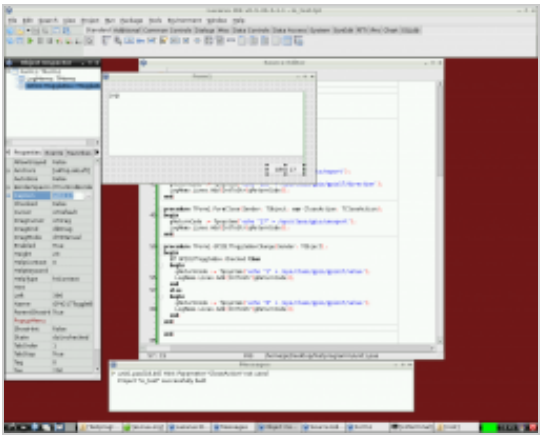
Lazarus / Free Pascal

The GPIO pins are accessible from Lazarus without any third-party software. This is performed by means of the BaseUnix (http://www.freepascal.org/docs-html/rtl/baseunix/index.html) unit that is part of every distribution of Lazarus and Free Pascal or by invoking Unix shell commands with **fpsystem**. The following example uses GPIO

pin 17 as output port. It is assumed that you created a form named GPIO17ToggleBox with a TToggleBox and a TMemo named LogMemo (optional, for logging purposes). The program has to be executed with root privileges.

Unit for controlling the GPIO port:

```
unit Unit1;

{Demo application for GPIO on Raspberry Pi}
{Inspired by the Python input/output demo application by Gareth Hal
{written for the Raspberry Pi User Guide, ISBN 978-1-118-46446-5}

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics,
  Dialogs, StdCtrls, Unix, BaseUnix;

type

  { TForm1 }

  TForm1 = class(TForm)
    LogMemo: TMemo;
    GPIO17ToggleBox: TToggleBox;
    procedure FormActivate(Sender: TObject);
    procedure FormClose(Sender: TObject; var CloseAction: TCloseAct
    procedure GPIO17ToggleBoxChange(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;

const
  PIN_17: PChar = '17';
  PIN_ON: PChar = '1';
  PIN_OFF: PChar = '0';
  OUT_DIRECTION: PChar = 'out';

var
  Form1: TForm1;
  gReturnCode: longint; {stores the result of the IO operation}

implementation

{$R *.lfm}

{ TForm1 }

procedure TForm1.FormActivate(Sender: TObject);
var
  fileDesc: integer;
begin
  { Prepare SoC pin 17 (pin 11 on GPIO port) for access: }
  try
    fileDesc := fpopen('/sys/class/gpio/export', O_WrOnly);
    gReturnCode := fpwrite(fileDesc, PIN_17[0], 2);
    LogMemo.Lines.Add(IntToStr(gReturnCode));
  finally
    gReturnCode := fpclose(fileDesc);
    LogMemo.Lines.Add(IntToStr(gReturnCode));
  end;
  { Set SoC pin 17 as output: }
  try
    fileDesc := fpopen('/sys/class/gpio/gpio17/direction', O_WrOnly
    gReturnCode := fpwrite(fileDesc, OUT_DIRECTION[0], 3);
```



A simple app for controlling GPIO pin 17 with Lazarus

```
    LogMemo.Lines.Add(IntToStr(gReturnCode));
  finally
    gReturnCode := fpclose(fileDesc);
    LogMemo.Lines.Add(IntToStr(gReturnCode));
  end;
end;

procedure TForm1.FormClose(Sender: TObject; var CloseAction: TClose
var
  fileDesc: integer;
begin
  { Free SoC pin 17: }
  try
    fileDesc := fpopen('/sys/class/gpio/unexport', O_WrOnly);
    gReturnCode := fpwrite(fileDesc, PIN_17[0], 2);
    LogMemo.Lines.Add(IntToStr(gReturnCode));
  finally
    gReturnCode := fpclose(fileDesc);
    LogMemo.Lines.Add(IntToStr(gReturnCode));
  end;
end;

procedure TForm1.GPIO17ToggleBoxChange(Sender: TObject);
var
  fileDesc: integer;
begin
  if GPIO17ToggleBox.Checked then
  begin
    { Swith SoC pin 17 on: }
    try
      fileDesc := fpopen('/sys/class/gpio/gpio17/value', O_WrOnly);
      gReturnCode := fpwrite(fileDesc, PIN_ON[0], 1);
      LogMemo.Lines.Add(IntToStr(gReturnCode));
    finally
      gReturnCode := fpclose(fileDesc);
      LogMemo.Lines.Add(IntToStr(gReturnCode));
    end;
  end
  else
  begin
    { Switch SoC pin 17 off: }
    try
      fileDesc := fpopen('/sys/class/gpio/gpio17/value', O_WrOnly);
      gReturnCode := fpwrite(fileDesc, PIN_OFF[0], 1);
      LogMemo.Lines.Add(IntToStr(gReturnCode));
    finally
      gReturnCode := fpclose(fileDesc);
      LogMemo.Lines.Add(IntToStr(gReturnCode));
    end;
  end;
end;

end.
```

Main program:

```
program io_test;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Interfaces, // this includes the LCL widgetset
  Forms, Unit1
  { you can add units after this };
```

```
{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Alternative way to access the GPIO port with Lazarus / Free Pascal is by using Lazarus wrapper unit for Gordon Henderson's wiringPi C library (http://www.lazarus.freepascal.org/index.php/topic,17404.0.html) or encapsulated shell calls (http://wiki.lazarus.freepascal.org/Lazarus_on_Raspberry_Pi#2._Hardware_access_via_encapsulated_shell_calls).

The Lazarus wiki (http://wiki.freepascal.org/Lazarus_on_Raspberry_Pi) describes a demo program (http://wiki.freepascal.org/Lazarus_on_Raspberry_Pi#Reading_the_status_of_a_pin) that can read the status of a GPIO pin.

BASIC

BASIC - Return to BASIC

'Return to Basic' (RTB) can be found here (https://projects.drogon.net/rtb/).

It is a new BASIC featuring modern looping constructs, switch statements, named procedures and functions as well as graphics (Cartesian and turtle), file handling and more. It also supports the Pi's on-board GPIO without needing to be run as root. (You do not need any special setup routines either)

Sample blink program:

```
// blink.rtb:
//    Blink program in Return to Basic
//    Gordon Henderson, projects@drogon.net
//
PinMode (0, 1) // Output
CYCLE
  DigitalWrite (0, 1) // Pin 0 ON
  WAIT (0.5) // 0.5 seconds
  DigitalWrite (0, 0)
  WAIT (0.5)
REPEAT
END
```

BASIC

Bywater BASIC Interpreter

The Bywater BASIC Interpreter (bwBASIC) implements a large superset of the ANSI Standard for Minimal BASIC (X3.60-1978) and a significant subset of the ANSI Standard for Full BASIC (X3.113-1987) in C. It also offers shell programming facilities as an extension of BASIC. bwBASIC seeks to be as portable as possible and is downloadable.[3]

BASIC programming of the I/O

Setting up a GPIO pin to be used for inputs or for outputs.

The control words cannot be loaded directly into the 32 bit ARM registers with 32 bit addresses, as bwBASIC has no POKE and PEEK commands and other versions of BASIC only handle 8 bit registers with 16 bit addresses with these commands. So the GPIO pins need to be exported so that they exist in a file structure which can be accessed from basic with the OPEN command (ref 2).

This must be done in Linux root. BASIC must be run in the root too.

```
sudo -1
sudo bwbasic
```

Now to export GPIO pin 4 for example, using a Shell command.

```
echo "4" >  /sys/class/gpio/export
```

Whilst bwbasic can accommodate shell commands, and we can store a set of these commands (eg. to export a number of GPIO pins at the outset) as numbered statements in a file that can be loaded with the basic command LOAD "filename" and RUN (ref 2), the shell commands have to run as a separate file, as they cannot be run from within, as part of a basic program.

Now the file containing the pin direction setting from BASIC can be accessed.

GPIO pin 4 can be set to input or output by OPENing its pin direction file for output and writing "in" or "out" with a PRINT# command.[4]

```
10 OPEN O",#1, "/sys/devices/virtual/gpio/gpio4/direction",2
20 PRINT #1,"out"
30 CLOSE #1
```

This closes the open direction file, whereupon the system performs the action of setting the direction to "out". The system only carries out the action as the file is closed.[5]

Now the output of the gpio 4 pin can be controlled from BASIC.

GPIO 4 pin can be set to 1 or to 0 by OPENing its pin value file for output and writing "1" or "0" with a PRINT# command.

```
40 OPEN "O",#4, "/sys/devices/virtual/gpio/gpio4/value",1
50 PRINT #4,"1"
60 CLOSE #4
```

Similarly the output of GPIO pin 4 can be turned off.

```
OPEN "O",#4, "/sys/devices/virtual/gpio/gpio4/value",1
PRINT #4,"0"
CLOSE #4.
```

Example of an (unstructured) BASIC program

To read the state of a switch and control the power to two LEDs connected to GPIO pins 8,7 and 4 respectively.

Program to set 2 pins as outputs and 1 pin as input and to read the input turning on two different combinations of the two outputs (ie output 0,1 or 1,0) depending on the state of the input (1 or 0).

```
sudo –i
sudo bwbasic
LOAD “export.bas”
LIST
REM a set of Shell statements to export the three GPIO pins.
10 echo “4” > /sys/class/gpio/export
20 echo “7” > /sys/class/gpio/export
30 echo “8” > /sys/class/gpio/export
RUN
```

NEW clears the export.bas program from memory

```
LOAD “demo1.bas”
LIST
10 OPEN ”O”,#1, “/sys/devices/virtual/gpio/gpio4/direction”,2
20 OPEN ”O”,#2, “/sys/devices/virtual/gpio/gpio7/direction”,2
30 OPEN ”O”,#3, “/sys/devices/virtual/gpio/gpio8/direction”,2
REM opens the three pin direction files
40 PRINT #1, “out”
50 PRINT #2, “out”
60 PRINT #3, “in”
REM sets GPIO pins 4 and 7 as outputs and GPIO pin 8 as input.
70 CLOSE #1
80 CLOSE #2
90 CLOSE #3
REM closes all open files, allowing the system to perform the direction settings.
100 OPEN ”I”,#8, “/sys/devices/virtual/gpio/gpio8/value”,1
REM opens the GPIO pin 8 value file
110 INPUT #8,x
REM reads the value of the input pin and stores the value in numerical variable x
120 CLOSE #8
REM closes the open file, allowing the system to read the value of the input pin and store the value in numeric
130 OPEN “O”,#1, “/sys/devices/virtual/gpio/gpio4/value”,1
140 OPEN “O”,#2, “/sys/devices/virtual/gpio/gpio7/value”,1
REM opens the GPIO pins 4 and value files ready for outputting 1s and 0s.
150 IF x<1 THEN GOTO 160 ELSE GOTO 190
REM tests the state of the switch (1 or0) and directs the program to generate the appropriate outputs
160 PRINT #1,”1”
170 PRINT #2,”0”
180 GOTO 210
190  PRINT#1,"0"
200 PRINT #2,"1"
210 CLOSE #1
220 CLOSE #2
REM Closes the files and allows the outputs to light the LED
230 END.
```

When all is done, the GPIO pins should be unexported, to leave the R-Pi as we found it.[6]

```
NEW
LOAD “unexport.bas”
LIST
REM a set of Shell statements to unexport the three GPIO pins.
10 echo “4” > /sys/class/gpio/unexport
20 echo “7” > /sys/class/gpio/unexport
```

```
30 echo "8" > /sys/class/gpio/unexport
RUN
```

A simple circuit to provide the switched input and the two LED outputs.[7]

For the two original documents this example has been copied from, see:

1. GPIO_Driving_Example_(BASIC)_.doc
2. Raspberry_Pi_I-O_viii.doc

Graphical User Interfaces

WebIOPi

WebIOPi (http://code.google.com/p/webiopi/) allows to control each GPIO with a simple web interface that can be used with any browser. Available in PHP and Python, they both require root access, but the Python version serves HTTP itself. Each GPIO pin can be set up as input or output and its LOW/HIGH stae can be changed. WebIOPi is fully customizable, so it can be used for home remote control. It also works over Internet. UART/SPI/I²C support will be added later. See code examples above.

Benchmarks

Toggling gpios

For information about the relative performance at toggling GPIOs for each approach, and a comparison of original Pi vs. Raspberry Pi 2, see:

- http://codeandlife.com/2012/07/03/benchmarking-raspberry-pi-gpio-speed/
- http://codeandlife.com/2015/03/25/raspberry-pi-2-vs-1-gpio-benchmark/

Citations

1. ↑ http://pi4j.com/usage.html#Pin_Numbering
2. ↑ https://projects.drogon.net/raspberry-pi/wiringpi/pins/
3. ↑ packages.debian.org (http://packages.debian.org/stable/interpreters/bwbasic)
4. ↑ Ed Beynon, ybw.com (http://www.ybw.com/forums/showthread.php?t=331320&page=5)
5. ↑ Arthur Kaletzky. Private communication. 25/10/2012
6. ↑ This paper RPi Low-level peripherals.
7. ↑ Ancient Mariner. Dec. 2012