RPi Low-level peripherals

From eLinux.org

Back to the Hub.

Hardware & Peripherals:

Hardware and Hardware History.

Low-level Peripherals *and Expansion Boards.*

Screens, Cases and Other Peripherals.

Contents

Introduction

In addition to the familiar USB, Ethernet and HDMI ports, the R-Pi offers lower-level interfaces intended to connect more directly with chips and subsystem modules. These GPIO (general purpose I/O) signals on the 2x13 header pins include SPI, I²C, serial UART, 3V3 and 5V power. These interfaces are not "plug and play" and require care to avoid miswiring. The pins use a 3V3 logic level and are not tolerant of 5V levels, such as you might find on a 5V powered Arduino. CSI (camera serial interface) can be used to connect the 5 MP camera available. Not yet software-enabled is the flex cable connectors with DSI (display serial interface) and a serial link inside the HDMI connector called CEC. (consumer electronics control)

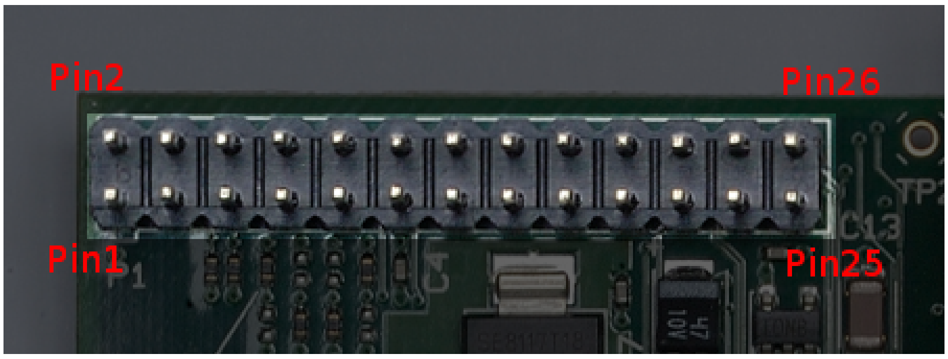
General Purpose Input/Output (GPIO)

General Purpose Input/Output (a.k.a. GPIO) is a generic pin on a chip whose behavior (including whether it is an input or output pin) can be controlled (programmed) through software.

The Raspberry Pi allows peripherals and expansion boards (such as the Rpi Gertboard) to access the CPU by exposing the inputs and outputs.

For further general information about GPIOs, see: the wikipedia article (http://en.wikipedia.org/wiki/GPIO).
For further specific information about the Raspberry Pi's BCM2835 GPIOs, see: this wiki article.
To connect devices to the serial port (UART), see the RPi_Serial_Connection page.
Sample circuits for interfacing the GPIOs with other electronics are shown on the RPi_GPIO_Interface_Circuits page.

The production Raspberry Pi board has a 26-pin 2.54 mm (100 mil)[1] expansion header, marked as P1, arranged in a 2x13 strip. They provide 8 GPIO pins plus access to I²C, SPI, UART), as well as +3.3 V, +5 V and GND supply lines. Pin one is the pin in the first column and on the bottom row. [2]



GPIO voltage levels are 3.3 V and are not 5 V tolerant. There is no over-voltage protection on the board - the intention is that people interested in serious interfacing will use an external board with buffers, level conversion and analog I/O rather than soldering directly onto the main board.

All the GPIO pins can be reconfigured to provide alternate functions, SPI, PWM (http://en.wikipedia.org/wiki/Pulse-width_modulation), I²C and so. At reset only pins GPIO 14 & 15 are assigned to the alternate function UART, these two can be switched back to GPIO to provide a total of 17 GPIO pins[3]. Each of their functions and full details of how to access are detailed in the chipset datasheet [4].

Each GPIO can interrupt, high/low/rise/fall/change.[5][6] There is currently no support for GPIO interrupts in the official kernel, however a patch exists, requiring compilation of modified source tree.[7] The 'Raspbian "wheezy"' [8] version that is currently recommended for starters already includes GPIO interrupts.

GPIO input hysteresis (Schmitt trigger) can be on or off, output slew rate can be fast or limited, and source and sink current is configurable from 2 mA up to 16 mA. Note that chipset GPIO pins 0-27 are in the same block and these properties are set per block, not per pin. See GPIO Datasheet Addendum - GPIO Pads Control (http://www.scribd.com/doc/101830961/GPIO-Pads-Control2). Particular attention should be applied to the note regarding SSO (Simultaneous Switching Outputs): to avoid interference, driving currents should be kept as low as possible.

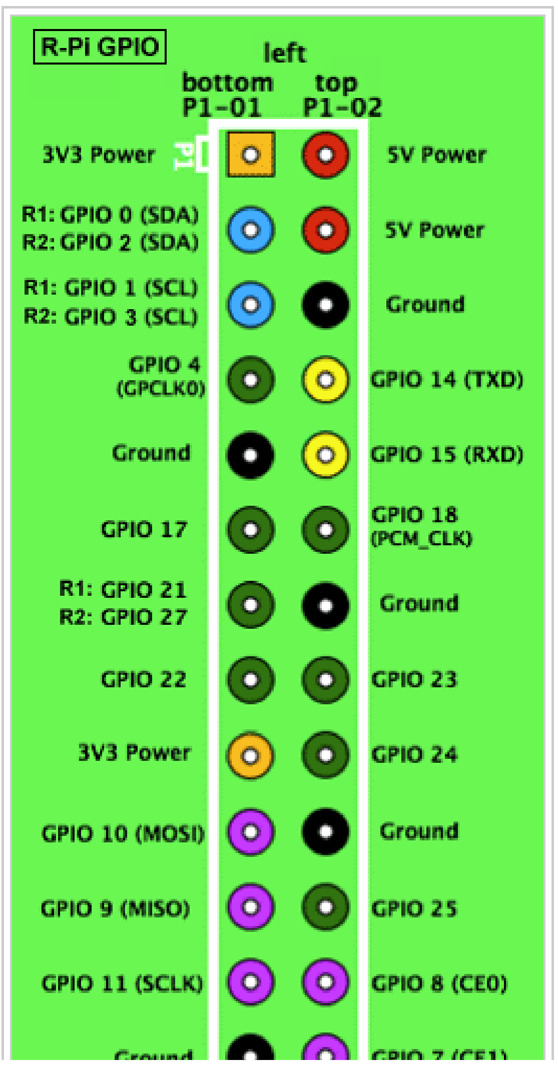
The available alternative functions and their corresponding pins are detailed below. These numbers are in reference to the chipset documentation and may not match the numbers exposed in Linux. Only fully usable functions are detailed, for some alternative functions not all the necessary pins are available for the funtionality to be actually used.

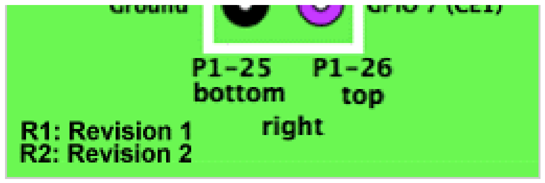
There is also some information on the Tutorial on Easy GPIO Hardware & Software.

Kernel boot messages go to the UART at 115200 bit/s - there are more details on the serial port page.

R-Pi PCB Revision 2 UPDATE: According to Eben at [1] (http://www.raspberrypi.org/archives/1929#comment-31646) the R-Pi Rev.2 board being rolled out starting in September 2012 adds 4 more GPIO on a new connector called P5, and changes some of the existing P1 GPIO pinouts. On Rev2, GPIO_GEN2 [BCM2835/GPIO27] is routed to P1 pin 13, and changes what was SCL0/SDA0 to SCL1/SDA1: SCL1 [BCM2835/GPIO3] is routed to P1 pin 5, SDA1 [BCM2835/GPIO2] is routed to P1 pin 3. Also the power and ground connections previously marked "Do Not Connect" on P1 will remain as connected, specifically: P1-04:+5V0, P1-09:GND, P1-14:GND, P1-17:+3V3, P1-20:GND, P1-25:GND. According to this comment [2] (http://www.raspberrypi.org/archives/2081#comment-33577) (and confirmed in this post [3] (http://www.raspberrypi.org/archives/2233)) the P1 pinout is not expected to change in future beyond the current Rev.2 layout.

P1 Header Pinout, top row:





The layout of the Raspberry Pi P1 pin-header seen from the top, containing pins useable for general purpose I/O. Colour coded to the table. Three pins changed between PCB Rev.1 and Rev.2 Source (https://sites.google.com/site/burngatehouse/h

Pin Number	Pin Name Rev1	Pin Name Rev2	Hardware Notes	Alt 0 Function	Other Alternative Functions
P1-02	5V0		Supply through input poly fuse		
P1-04	5V0		Supply through input poly fuse		
P1-06	GND				
P1-08	GPIO 14		Boot to Alt 0 ->	UART0_TXD	ALT5 = UART1_TXD
P1-10	GPIO 15		Boot to Alt 0 ->	UART0_RXD	ALT5 = UART1_RXD
P1-12	GPIO 18			PCM_CLK	ALT4 = SPI1_CE0_N ALT5 = PWM0
P1-14	GND				
P1-16	GPIO23				ALT3 = SD1_CMD ALT4 = ARM_RTCK
P1-18	GPIO24				ALT3 = SD1_DAT0 ALT4 = ARM_TDO
P1-20	GND				
P1-22	GPIO25				ALT3 = SD1_DAT1 ALT4 = ARM_TCK
P1-24	GPIO08			SPI0_CE0_N	
P1-26	GPIO07			SPI0_CE1_N	

P1 Header Pinout, bottom row:

Pin Number	Pin Name Rev1	Pin Name Rev2	Hardware Notes	Alt 0 Function	Other Alternative Functions
P1-01	3.3 V		50 mA max (01 & 17)		
P1-03	GPIO 0	**GPIO 2**	1K8 pull up resistor	I2C0_SDA / **I2C1_SDA**	
P1-05	GPIO 1	**GPIO 3**	1K8 pull up resistor	I2C0_SCL / **I2C1_SCL**	

P1-07	GPIO 4			GPCLK0	ALT5 = ARM_TDI
P1-09	GND				
P1-11	GPIO17				ALT3 = UART0_RTS ALT4 = SPI1_CE1_N ALT5 = UART1_RTS
P1-13	GPIO21	**GPIO27**		PCM_DOUT / **reserved**	ALT4 = SPI1_SCLK ALT5 = GPCLK1 / **ALT3 = SD1_DAT3 ALT4 = ARM_TMS**
P1-15	GPIO22				ALT3 = SD1_CLK ALT4 = ARM_TRST
P1-17	3.3 V		50 mA max (01 & 17)		
P1-19	GPIO10			SPI0_MOSI	
P1-21	GPIO9			SPI0_MISO	
P1-23	GPIO11			SPI0_SCLK	
P1-25	GND				

Colour legend
+5 V
+3.3 V
Ground, 0V
UART
GPIO
SPI
I²C

KiCad symbol: File:Conn-raspberry.lib

[9]

Pin 3 (SDA0) and Pin 5 (SCL0) are preset to be used as an I²C interface. So there are 1.8 kilohm pulls up resistors on the board for these pins.[10]

Pin 12 supports PWM (http://en.wikipedia.org/wiki/Pulse-width_modulation) .

It is also possible to reconfigure GPIO connector pins P1-7, 15, 16, 18, 22 (chipset GPIOs 4 and 22 to 25) to provide an ARM JTAG interface.[11] However ARM_TMS isn't available on the GPIO connector (chipset pin 12 or 27 is needed). Chipset pin 27 is available on S5, the CSI camera interface however.

It is also possible to reconfigure GPIO connector pins P1-12 and 13 (chipset GPIO 18 and 21) to provide an I2S (a hardware modification may be required[12]) or PCM interface.[13] However, PCM_FS and PCM_DIN (chipset pins 19 and 20) are needed for I2S or PCM.

A second I²C interface (GPIO02_ALT0 is SDA1 and GPIO03_ALT0 is SCL1) and two further GPIOs (GPIO05_ALT0 is GPCLK1, and GPIO27) are available on S5, the CSI camera interface.

Referring to pins on the Expansion header

The header is referred to as "The GPIO Connector (P1)". To avoid nomenclature confusion between Broadcom signal names on the SoC and pin names on the expansion header, the following naming is highly recommended.

- The expansion header is referred to as "Expansion Header" or "GPIO Connector (P1)"
- Pins on the GPIO connector (P1) are referred to as P1-01, etc.
- Names GPIO0, GPIO1, GPIOx-ALTy, etc. refer to the signal names on the SoC as enumerated in the Broadcom datasheet, where "x" matches BCM2835 number (without leading zero) and "y" is the alternate number column 0 to 5 on page 102-103 of the Broadcom document. For example, depending on what you are describing, use either "GPIO7" to refer to a row of the table, and "GPIO7-ALT0" would refer to a specific cell of the table.
- When refering to signal names, you should modify the Broadcom name slightly to minimize confusion. The Broadcom SPI bus pin names are fine, such as "SPI0_*" and "SPI1_*", but they didn't do the same on the I^{2}C and UART pins. Instead of using "SDA0" and "SCL0", you should use "I2C0_SDA" and "I2C0_SCL"; and instead of "TX" or "TXD" and "RX" or "RXD", you should use "UART0_TXD" and "UART0_RXD".

Power pins

The maximum permitted current draw from the 3.3 V pins is 50 mA.

Maximum permitted current draw from the 5 V pin is the USB input current (usually 1 A) minus any current draw from the rest of the board.[14]

- Model A: 1000 mA - 500 mA -> max current draw: 500 mA
- Model B: 1000 mA - 700 mA -> max current draw: 300 mA

Be very careful with the 5 V pins P1-02 and P1-04, because if you short 5 V to any other P1 pin you may permanently damage your RasPi. Before probing P1, it's a good idea to strip short pieces of insulation off a wire and push them over the 5 V pins so you don't accidentally short them with a probe.

GPIO hardware hacking

The complete list of chipset GPIO pins which are available on the GPIO connector is:

> 0, 1, 4, 7, 8, 9, 10, 11, 14, 15, 17, 18, 21, 22, 23, 24, 25

(on the Revision2.0 RaspberryPis, this list changes to: 2, 3, 4, 7, 8, 9, 10, 11, 14, 15, 17, 18, 22, 23, 24, 25, 27, with 28, 29, 30, 31 additionally available on the P5 header)

As noted above, P1-03 and P1-05 (SDA0 and SCL0 / SDA1 and SCL1) have 1.8 kilohm pull-up resistors to 3.3 V.

If 17 GPIOs aren't sufficient for your project, there are a few other signals potentially available, with varying levels of software and hardware (soldering iron) hackery skills:

GPIO02, 03, 05 and 27 are available on S5 (the CSI interface) when a camera peripheral is not connected to that socket, and are configured by default to provide the functions SDA1, SCL1, CAM_CLK and CAM_GPIO respectively. SDA1 and SCL1 have 1K6 pull-up resistors to 3.3 V.

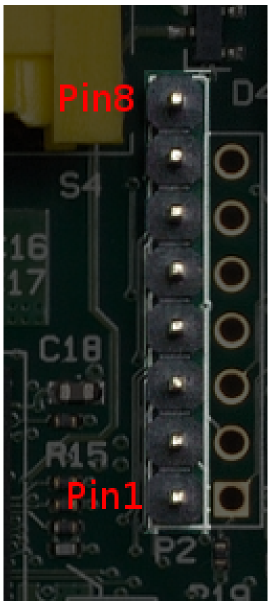
GPIO06 is LAN_RUN and is available on pad 12 of the footprint for IC3 on the Model A. On Model B, it is in use for the Ethernet function.

There are a few other chipset GPIO pins accessible on the PCB but are in use:

- GPIO16 drives status LED D5 (usually SD card access indicator)
- GPIO28-31 are used by the board ID and are connected to resistors R3 to R10 (only on Rev1.0 boards).
- GPIO40 and 45 are used by analogue audio and support PWM (http://en.wikipedia.org/wiki/Pulse-width_modulation). They connect to the analogue audio circuitry via R21 and R27 respectively.
- GPIO46 is HDMI hotplug detect (goes to pin 6 of IC1).
- GPIO47 to 53 are used by the SD card interface. In particular, GPIO47 is SD card detect (this would seem to be a good candidate for re-use). GPIO47 is connected to the SD card interface card detect switch; GPIO48 to 53 are connected to the SD card interface via resistors R45 to R50.

P2 header

The P2 header is the VideoCore JTAG and used only during the production of the board. It cannot be used as the ARM JTAG [15]. This connector is unpopulated in Rev 2.0 boards.

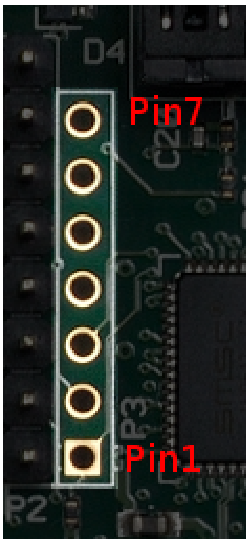


Useful P2 pins:

- Pin 1 - 3.3V (same as P1-01, 50 mA max current draw across both of them)
- Pin 7 - GND
- Pin 8 - GND

P3 header

The P3 header, unpopulated, is the LAN9512 JTAG [16].

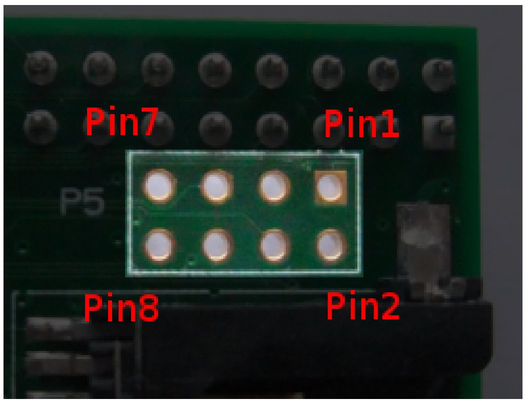


Useful P3 pins:

- Pin 7 - GND

P5 header

The P5 header was added with the release of the Revision 2.0 PCB design.



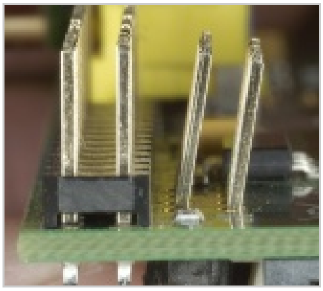
P5 Header Pinout (seen from the back of the board), top row:

Pin Number	Pin Name Rev2	Hardware Notes	Alt 0 Function	Other Alternative Functions
P5-01	5V0	Supply through input poly fuse		
P5-03	GPIO28		I2C0_SDA	ALT2 = PCM_CLK
P5-05	GPIO30			ALT2 = PCM_DIN ALT3 = UART0_CTS ALT5 = UART1_CTS
P5-07	GND			

P5 Header Pinout (seen from the back of the board), bottom row:

Pin Number	Pin Name Rev2	Hardware Notes	Alt 0 Function	Other Alternative Functions
P5-02	3.3 V	50 mA max (combined with P1)		

P5-04	GPIO29		I2C0_SCL	ALT2 = PCM_FS
P5-06	GPIO31			ALT2 = PCM_DOUT ALT3 = UART0_RTS ALT5 = UART1_RTS
P5-08	GND			


Slanted P5 header

Note that the connector is intended to be mounted on the **bottom** of the PCB, so that for those who put the connector on the top side, the pin numbers are swapped. Pin 1 and pin 2 are swapped, pin 3 and 4, etc.

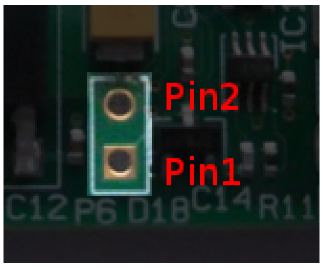
Some people have come to the conclusion that the best way (for them) to attach this header is on top, at a slant away from the P1 header. (http://raspi.tv/2013/the-leaning-header-of-pi5a-how-best-to-solder-a-header-on-p5)

The new header can provide a second I²C channel (SDA + SCL) and handshake lines for the existing UART (TxD and RxD), or it can be used for an I2S (audio codec chip) interface using the PCM signals CLK, FS (Frame Sync), Din and Dout.

Note that the connector is placed JUST off-grid with respect to the P1 connector.

P6 header

The P6 header was added with the release of the Revision 2.0 PCB design.



P6 Pinout:

Pin Number	Pin Name Rev2	Hardware Notes
P6-01	RUN	Short to ground to reset the BCM2835
P6-02	GND	

A reset button can be attached to the P6 header (http://raspi.tv/2012/making-a-reset-switch-for-your-rev-2-raspberry-pi), with which the Pi can be reset. Momentarily shorting the two pins of P6 together will cause a soft reset of the CPU (which can also 'wake' the Pi from halt/shutdown state).

Internal Pull-Ups & Pull-Downs

The GPIO ports include the ability to enable and disable internal pull-up or pull-down resistors (see below for code examples/support of this).

Pull-up is Min. 50K Ohm, Max 65 KOhm.

Pull-down is Min. 50K Ohm, Max 60 KOhm.

Driver support

The Foundation will not include a GPIO driver in the initial release, standard Linux GPIO drivers should work with minimal modification.[17]

The community implemented SPI and I²C drivers [18], which will be integrated with the new Linux pinctrl concept in a later version of the kernel. (On Oct. 14 2012, it was already included in the latest raspbian image.) A first compiled version as Linux modules is available to install on the 19/04/2012 Debian image, including 1-wire support[19]. The I²C and SPI driver uses the hardware modules of the microcontroller and interrupts for low CPU usage, the 1-wire support uses bitbanging on the GPIO ports, which results in higher CPU usage.

GordonH[20] wrote a (mostly) Arduino compatible/style WiringPi library (https://projects.drogon.net/raspberry-pi/wiringpi/) in C for controlling the GPIO pins.

A useful tutorial on setting up I²C driver support can be found at Robot Electronics (http://www.robot-electronics.co.uk/htm/raspberry_pi_examples.htm) - look for the downloadable document rpi_i2c_setup.doc

Graphical User Interfaces

WebIOPi

WebIOPi (http://code.google.com/p/webiopi/) allows you to control each GPIO with a simple web interface that you can use with any browser. Available in PHP and Python, they both require root access, but Python version serves HTTP itself. You can setup each GPIO as input or output and change their states (LOW/HIGH). WebIOPi is fully customizable, so you can use it for home remote control. It also work over Internet. UART/SPI/I²C support will be added later. If you need some computing for your GPIO go to code examples below.

GPIO Code examples

C

Examples in different C-Languages.

C

Gert van Loo & Dom, has 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.) 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

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;
```

C + wiringPi

Get and install wiringPi: https://projects.drogon.net/raspberry-pi/wiringpi/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 ;
}
```

C + 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);
}
```

C

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;
}
```

C#

RaspberryPiDotNet library is available at https://github.com/cypherkey/RaspberryPi.Net/. The 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

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

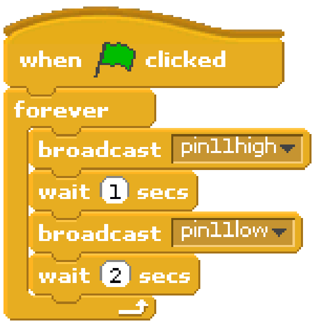
```
import RPi.GPIO as GPIO

# 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/

Also available is RPIO at https://pypi.python.org/pypi/RPIO RPIO extends RPi.GPIO with TCP socket interrupts, command line tools and more.

Scratch

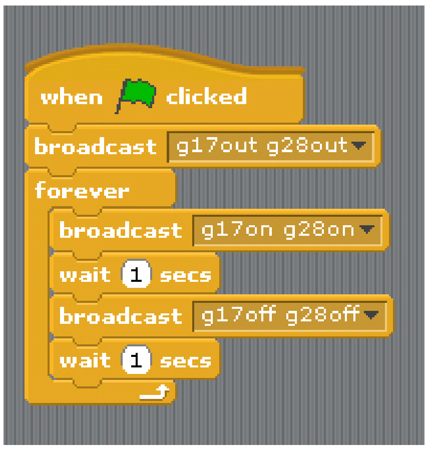
Scratch using the ScratchGPIO



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



```
when [flag] clicked
broadcast g17out g28out
forever
    broadcast g17on g28on
    wait 1 secs
    broadcast g17off g28off
    wait 1 secs
```

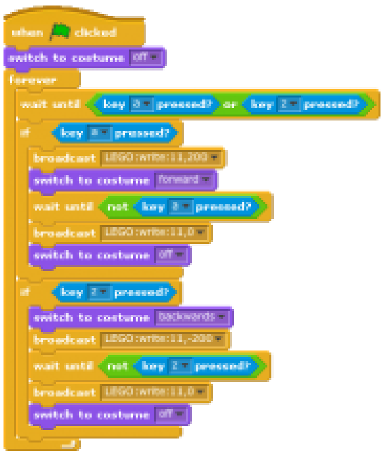
Scratch control GPIO (use GPIO number not P1 pin number can support GPIO 28,29,30,31)

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.



```
when [flag] clicked
switch to costume off
forever
    wait until <key 8 pressed? or key 2 pressed?>
    if <key 8 pressed?>
        broadcast LEGO:write:11,200
        switch to costume forward
        wait until <not <key 8 pressed?>>
        broadcast LEGO:write:11,0
        switch to costume off
    if <key 2 pressed?>
        switch to costume backwards
        broadcast LEGO:write:11,-200
        wait until <not <key 2 pressed?>>
        broadcast LEGO:write:11,0
        switch to costume off
```

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 [21] [22]. 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", Pin

    // 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", Pin

    // 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}
```

Bash shell script, using sysfs, part of the raspbian operating system

The export and unexport of pins must be done as root. To change to the root user see below: To change back, the word exit must be entered.

```
sudo -i
```

Export creates a new folder for the exported pin, and creates files for each of its control functions (i.e. active_low, direction, edge, power, subsystem, uevent, and value). Upon creation, the control files can be read by all users (not just root), but can only be written to by user root, the file's owner. Nevertheless, once created, it is possible to allow users other than root, to also write inputs to the control files, by changing the ownership or permissions of these files. Changes to the file's ownership or permissions must initially be done as root, as their owner and group is set to root upon creation. Typically you might change the owner to be the (non root) user controlling the GPIO, or you might add write permission, and change the group ownership to one of which the user controlling the GPIO is a member. By such means, using only packages provided in the recommended rasbian distribution, it is possible for Python CGI scripts, which are typically run as user nobody, to be used for control of the GPIO over the internet from a browser at a remote location.

```
#!/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
```

Shell script - take 2

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:

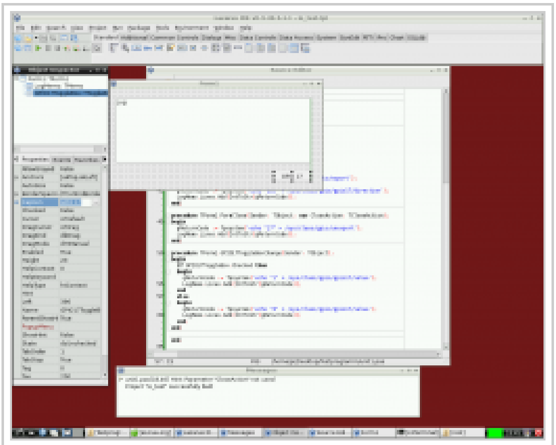
```
  gpio -g export 4 in
  gpio -g mode 4 up
```

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.

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 with a TToggleBox named GPIO17ToggleBox and for logging purposes a TMemo with name LogMemo (optional). The program has to be executed with root privileges.


A simple app for controlling GPIO pin 17 with Lazarus

Unit for controlling the GPIO port:

```
unit Unit1;

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

{$mode objfpc}{$H+}

interface

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

type

  { TForm1 }

  TForm1 = class(TForm)
    LogMemo: TMemo;
    GPIO17ToggleBox: TToggleBox;
    procedure FormActivate(Sender: TObject);
    procedure FormClose(Sender: TObject; var CloseAction:
    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 oper

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'
    gReturnCode := fpwrite(fileDesc, OUT_DIRECTION[0], 3)
    LogMemo.Lines.Add(IntToStr(gReturnCode));
  finally
    gReturnCode := fpclose(fileDesc);
    LogMemo.Lines.Add(IntToStr(gReturnCode));
  end;
end;

procedure TForm1.FormClose(Sender: TObject; var CloseActi
var
  fileDesc: integer;
begin
  { Free SoC pin 17: }
  try
    fileDesc := fpopen('/sys/class/gpio/unexport', O_WrOn
    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',
      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',
      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 ways 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_port) that can read the status of a GPIO pin.

BASIC

BASIC - Return to BASIC

RTB or Return to Basic can be found here: https://projects.drogon.net/return-to-basic/

It's a new BASIC featuring modern looping constructs, switch statements, named procedures and functions as well as graphics (caresian and turtle), file handling and more. It also supports the Pi's on-board GPIO without needing to be run as root. (You don't 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. You can download it at. http://packages.debian.org/stable/interpreters/bwbasic

BASIC programming of the I/O

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

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

We need to do this in Linux root. We need to run BASIC in the root too. First we go to the root, then we load bwbasic into root.

```
sudo -1
sudo bwbasic
```

REM Now to export the no4 GPIO pin 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 programme.

Now we can access the file containing the pin direction setting from BASIC

We can set GPIO pin 4 to input or to output by OPENing its pin direction file for output and writing “in” or “out” with a PRINT# command. (ref 2)

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

REM closes the open direction file, whereupon the system performs the action of setting the direction to “out”. NB the system only carries out the action as the file is closed.(ref 3)

We are now able to control the output of the gpio 4 pin from BASIC

We can set the GPIO 4 pin 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
```

REM turns on the output of GPIO pin 4.

REM similarly we can turn off the output of GPIO pin 4. OPEN ”O”,#4, “/sys/devices/virtual/gpio/gpio4/value”,1 PRINT #4,”0” CLOSE #4.

Example of an (unstructured) BASIC programme

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

Programme 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 REM clears the export.bas programme 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 v
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 ou
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, we should unexport the GPIO pins, to leave the R-Pi as we found it.(Ref 1.)

```
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.

Ancient Mariner. Dec. 2012

References.

1. This paper RPi Low-level peripherals.

2. Ed Beynon. http://www.ybw.com/forums/showthread.php?t=331320&page=5

3. Arthur Kaletzky. Private communication. 25/10/2012

4. bwbasic manual.

For the two original documents this example has been copied from see:
GPIO_Driving_Example_(BASIC)_.doc

Raspberry_Pi_I-O_viii.doc

SPI

There is one SPI bus brought out to the header: RPi_SPI

I²C

There are two I²C-buses on the Raspberry Pi: One on P1, and one on P5.

Note that there's a bug concerning I²C-clock-stretching, so don't use I²C-devices which use clock-stretching directly with the Raspberry Pi, or use a workaround. Details about this bug can be found at:

- http://www.raspberrypi.org/phpBB3/viewtopic.php?f=44&t=13771
- http://www.advamation.com/knowhow/raspberrypi/rpi-i2c-bug.html

MIPI CSI-2

On the production board[23], the Raspberry Pi Foundation design brings out the MIPI CSI-2 (Camera Serial Interface[24]) to a 15-way flat flex connector S5, between the Ethernet and HDMI connectors. A compatible camera[25] with 5 Megapixels and 1080p video resolution was released in May 2013.

DSI

On the production board, the Raspberry Pi Foundation design brings out the DSI (Display Serial Interface[26]) to a 15-way flat flex connector labelled S2, next to Raspberry Pi logo. It has two data lanes and a clock lane, to drive a possible future LCD screen device. Some smart phone screens use DSI[27].

CEC

HDMI-CEC (Consumer Electronics Control for HDMI) is supported by hardware but some driver work will be needed and currently isn't exposed into Linux userland. Eben notes that he has seen CEC demos on the Broadcom SoC they are using.

libCEC with Raspberry Pi support has been included in OpenELEC and will be included in Raspbmc RC4.[28]

For more information about HDMI-CEC and what you could do with it on the Raspberry Pi please see the CEC (Consumer Electronics Control) over HDMI article.

References

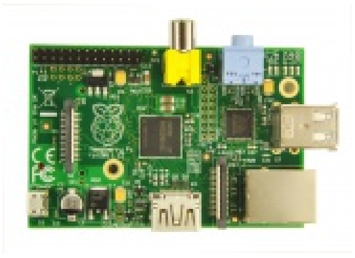
1. ↑ http://www.raspberrypi.org/forum/features-and-requests/easy-gpio-hardware-software/page-3/#p31907
2. ↑ http://www.raspberrypi.org/archives/384
3. ↑ http://www.raspberrypi.org/archives/384
4. ↑ http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf
5. ↑ http://www.raspberrypi.org/archives/384#comment-5217
6. ↑ http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf
7. ↑ http://www.raspberrypi.org/phpBB3/viewtopic.php?f=44&t=7509
8. ↑ http://www.raspberrypi.org/downloads
9. ↑ http://www.raspberrypi.org/forum/projects-and-collaboration-general/gpio-header-pinout-clarification/page-2
10. ↑ http://www.raspberrypi.org/forum/features-and-requests/easy-gpio-hardware-software/page-6/#p56480
11. ↑ http://www.raspberrypi.org/forum?mingleforumaction=viewtopic&t=1288.1
12. ↑ Forum:Sad about removal of I2S. Why was this change made? (http://www.raspberrypi.org/forum/features-and-requests/sad-about-removal-of-i2s-why-was-this-change-made)
13. ↑ http://www.raspberrypi.org/forum?mingleforumaction=viewtopic&t=1288.2
14. ↑ http://www.raspberrypi.org/forum?mingleforumaction=viewtopic&t=1536#postid-21841
15. ↑ http://www.raspberrypi.org/phpBB3/viewtopic.php?f=24&t=5894
16. ↑ http://www.raspberrypi.org/phpBB3/viewtopic.php?f=24&t=5894
17. ↑ http://www.raspberrypi.org/forum?mingleforumaction=viewtopic&t=1278.0
18. ↑ http://www.bootc.net/projects/raspberry-pi-kernel/
19. ↑ http://www.raspberrypi.org/phpBB3/viewtopic.php?p=86172#p86172
20. ↑ http://www.raspberrypi.org/forum/general-discussion/wiring-for-the-raspberry-pis-gpio
21. ↑ http://pi4j.com/usage.html#Pin_Numbering
22. ↑ https://projects.drogon.net/raspberry-pi/wiringpi/pins/

23. ↑ http://www.raspberrypi.org/wp-content/uploads/2012/04/Raspberry-Pi-Schematics-R1.0.pdf
24. ↑ http://www.mipi.org/specifications/camera-interface
25. ↑ http://elinux.org/Rpi_Camera_Module
26. ↑ http://www.mipi.org/specifications/display-interface
27. ↑ http://en.wikipedia.org/wiki/Display_Serial_Interface
28. ↑ http://blog.pulse-eight.com/2012/08/01/libcec-1-8-0-a-firmware-upgrade-and-raspberry-pi-support/

Raspberry Pi

- V
- T
- E (http://elinux.org/index.php?title=Template:Raspberry_Pi&action=edit)

Startup	Buying Guide - SD Card Setup - Basic Setup - Advanced Setup - Beginners Guide - Troubleshooting
Hardware	Hardware - Hardware History - **Low-level peripherals** - Expansion Boards
Peripherals	Screens - Cases - Other Peripherals (Keyboard, mouse, hub, wifi...)
Software	Software - Distributions - Kernel - Performance - Programming - VideoCore APIs - Utilities
Projects	Tutorials - Guides - Projects - Tasks - DataSheets - Education - Communities



Retrieved from "http://elinux.org/index.php?title=RPi_Low-level_peripherals&oldid=345350"
Category: RaspberryPi