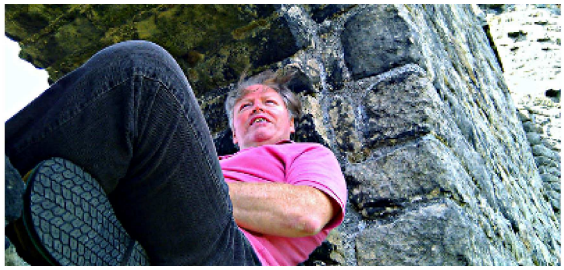
Gordons Projects

Projects, Fun and Games from Gordon @ Drogon



The GPIO utility

WiringPi comes with a separate program to help manage the GPIO. This program, called ***gpio***, can also be used in scripts to manipulate the GPIO pins – set outputs and read inputs. It's even possible to write entire programs just using the ***gpio*** command in a shell-script, although it's not terribly efficient doing it that way… Another way to call it is using the system() function in C/C++ or it's equivalent in other programming languages.

- The ***gpio*** command is designed to be installed as a setuid program and called by a normal user without using the sudo command (or logging in as root).

In addition to using the ***gpio*** utility to control the GPIO pins, you can:

- Export/Unexport pins via the */sys/class/gpio* interface, where they will then be available to user programs (that then do not need to be run as root or with sudo)
- Export pins to enable edge-triggered interrupts via the */sys/class/gpio* interface.
- Control the pins on the **PiFace** peripheral device.
- Load SPI and I2C modules and set /dev/ permissions to enable read/write by the user running the ***gpio*** program.
- Set the SPI buffer size and I2C baud rate (when loading the modules)
- Output values to the Gertboard DAC
- Read inputs from the Gertboard ADC
- Determine your Raspberry Pi board hardware revision.

See the man page for the ***gpio*** program to see what all the features are by typing

```
man gpio
```

at the command prompt.

Usage

From the Linux command line:

- **gpio -v**

This prints the version.

- **gpio -g …**

The optional -g flag causes pin numbers to be interpreted as BCM_GPIO pin numbers rather than standard ***wiringPi*** pin numbers.

Standard input and output commands

- **gpio [-g] mode <pin> in/out/pwm/up/down/tri**

This sets the mode of a pin to be input, output or pwm and additionally can set the internal pull-up/down resistors to pull-up, pull-down or none.

- **gpio [-g] write <pin> 0/1**

This sets an output pin to high (1) or low (0)

- **gpio [-g] pwm <pin> <value>**

Set the pin to a PWM value (0-1023 is supported)

- **gpio [-g] read <pin>**

Reads and prints the logic value of the given pin. It will print 0 (low) or 1 (high).

- **gpio readall**

This reads all the normally accessible pins and prints a table of their numbers (both wiringPi and BCM_GPIO, so makes for a handy cross-reference chart), along with their modes and current values.

Module Load Commands

- **gpio load spi [buffer size in KB]**

This loads the SPI kernel modules and optionally sets the internal buffer to the given size in KB (multiples of 1024). The default is 4KB and is usually more than enough for most application which only exchange a byte or 2 at a time over the SPI bus.

The /dev/spi* entries are set to be owned by the person using the **gpio** program, so there is no need to run subsequent programs as root (unless they use other wiringPi functions)

- **gpio load i2c [baud rate in Kb/sec]**

This loads the I2C kernel modules and optionally sets the baud rate to the given speed in Kb/sec (multiples of 1000). The default is 100Kb/sec.

The /dev/I2c* entries are set to be owned by the person using the **gpio** program, so there is no need to run subsequent programs as root (unless they use other ***wiringPi*** functions)

/sys/class/gpio mode commands

- **gpio export <pin> in/out**

This exports the given pin (BCM-GPIO pin number) as an input or output and makes it available for a user program running as the same user to use.

- **gpio unexport <pin>**

Removes the export of the given pin.

- **gpio unexportall**

Removes all */sys/class/gpio* exports.

- **gpio exports**

This prints a list of all gpio pins which have been exported via the */sys/class/gpio* interface and their modes.

- **gpio edge <pin> rising/falling/both/none**

This enables the given pin for edge interrupt triggering on the rising, falling or both edges. (Or none which disables it)

Note: The pin numbers in the sys mode are *always* BCM-GPIO pin numbers.

Examples

```
gpio mode 0 out
gpio write 0 1
```

This uses the ***wiringPi*** pin numbers to set pin 0 as an output and then sets the pin to a logic 1.

```
gpio -g mode 0 in
gpio -g read 0
```

This uses the BCM_GPIO pin numbering scheme and reads pin 0 (SDA0 on a Rev. 1 Raspberry Pi)

Internal pull up/down resistors

The GPIO lines have internal pull up or pull-down resistors which can be controlled via software when a pin is in input mode.

```
gpio mode 0 up
gpio mode 0 down
gpio mode 0 tri
```

These set the resistors to pull-up, pull-down and none respectively on ***wiringPi*** pin 0.

PiFace Commands

The PiFace is somewhat limited in that it has 8 inputs pins and 8 output pins and these are fixed in the hardware, so only the write and read commands are implemented:

- **gpio -p write <pin> 0/1**

Writes the value 0 (off) or 1 (on) to the output pin on the PiFace

- **gpio -p read <pin>**

Reads and prints the value on the given input pin.

- **gpio -p mode <pin> up/tri**

This enables (up) or disables (tri) the internal pull-up resistor on the given input pin. You need to enable the pull-up if you want to read any of the on-board switches on the PiFace board.

Comments

The GPIO utility — 73 Comments

wolfgang schlatter
on August 7, 2012 at 8:19 pm said:

thanks a lot, with your help i got this stepper running:

http://www.youtube.com/watch?v=yzzafSqnk5s&

controlling a stepper with raspberrypi gpio pins

took a schmalzhaus easydriver stepper shield
controlled with a little shell script

Here,my stepper has 1,8° = 200 Steps for 360°
GPIO pin 23 controlls the steps, one high peak per step,
GPIO pin 24 controlls the direction

30 times clockwise and counterclockwise Rounds:

```
#!/bin/sh
pause=0.001
gpio -g mode 23 out
gpio -g mode 24 out

for i in $(seq 30); do

echo Runde $i startet:
sleep 1

gpio -g write 24 1
for i in $(seq 200); do
echo $i
gpio -g write 23 1
sleep $pause
gpio -g write 23 0
sleep $pause
```

done
sleep 1
gpio -g write 24 0
for i in $(seq 200); do
echo $i
gpio -g write 23 1
sleep $pause
gpio -g write 23 0
sleep $pause
done
done

Gordon
on August 7, 2012 at 8:54 pm said:

Impressive! Well done!

-Gordon

Matt Smith
on August 9, 2012 at 3:15 pm said:

what about if i want to write a c program that just toggles an LED on and off

been trying to acomplish this all week (only started c and rasPI on monday)

so far i have installed all your .tz file

and believe i am somewhere close or a million miles away with

/* Blinking LED
* _____
*
* turns on and off a light emitting diode(LED)
*/

#include
#include
#include
#include

int main()
{
pinMode(0, OUTPUT)
digitalWrite(0, HIGH); // sets the LED on
delay(1000); // waits for a second
digitalWrite(0, LOW); // sets the LED off
delay(1000); // waits for a second

return 0;
}

Gordon
on August 9, 2012 at 3:31 pm said:

The program looks fine – it will set output 0 high for a second then set it low again.

compile with:

cc -o prog prog.c -lwiringPi

and run with:

sudo ./prog

To actually make it blink, then:

```
for (;;)
{
digitalWrite(0, HIGH); // sets the LED on
delay(1000); // waits for a second
digitalWrite(0, LOW); // sets the LED off
delay(1000); // waits for a second
}
```

That will loop forever…

To check the hardware is working (and wiringPi is installed OK), you can simply type:

```
gpio mode 0 out
gpio write 0 1
gpio write 0 0
```

and that will check the wiring/led before running the program.

Remember that wiringPi pin 0 is GPIO pin 17.

Remember to use a resistor in-series with the LED – anything from 220 ohms to 330 will do.

-Gordon

Matt Smith
on August 10, 2012 at 8:33 am said:

thanks gordon,

i have that LED blinking, added a printf(" LED blinking\n") so terminal showed a description.

is there a simple way to increment a number (0-255) and output it over pins (17 18 21 22 23 24 25 4) in binary. Not looking for a full solution just a nudge.

possibly assign GPIO0-7 as a port (only ever really done PICs in .BAS)

dont really wanna go with
#define BIT1 0
#define BIT2 1
#define BIT4 2
#define BIT8 3
#define BIT16 4
#define BIT32 5
#define BIT64 6
#define BIT128 7

/code ommited /

pinMode(BIT1, OUTPUT);
pinMode(BIT2, OUTPUT);
pinMode(BIT4, OUTPUT);
pinMode(BIT8, OUTPUT);
pinMode(BIT16, OUTPUT);
pinMode(BIT32, OUTPUT);
pinMode(BIT64, OUTPUT);
pinMode(BIT128, OUTPUT);

//b00000000 d0

digitalWrite(BIT1, LOW); // sets the BIT1 off
digitalWrite(BIT2, LOW); // sets the BIT2 off

digitalWrite(BIT4, LOW); // sets the BIT4 off
digitalWrite(BIT8, LOW); // sets the BIT8 off
digitalWrite(BIT16, LOW); // sets the BIT16 off
digitalWrite(BIT32, LOW); // sets the BIT32 off
digitalWrite(BIT64, LOW); // sets the BIT64 off
digitalWrite(BIT128, LOW); // sets the BIT128 off
delay (1000)

256 times toggling Bits

Also thanks for the compile command, i was using

gcc -o file file.c -L/usr/local/lib -lwiringPi

which is quite a handfull to type

-matt (if my qustions seem a bit too n00b please just post a link a forum or something)

Gordon
on August 11, 2012 at 8:27 am said:

Hi,

So there is an old phrase used by programmers (or is it a phrase used by old programmers!) anyway, it goes like this:

“When in doubt, use brute-force”

Which is what you are doing above, so that’s fine.

However we can refine it a little:

```
void outputBinary (int value)
{
if ((value & 0x01) == 0) digitalWrite (17, 0) else digitalWrite (17, 1) ;
if ((value & 0x02) == 0) digitalWrite (18, 0) else digitalWrite (18, 1) ;
if ((value & 0x04) == 0) digitalWrite (21, 0) else digitalWrite (21, 1) ;
... etc.
}
```

the next step would be to put the pin numbers into an array, then you can easilly change the pins, then you can also use a loop inside the function.

As for the compiling, yes, Sorry. However you might want to look at using a Makefile. If you get the wiringPi source and look in the examples directory – copy the Makefile from there and adapt as required. Just remember that spaces and TABs are different in Makefiles!

However you can make life easier in other ways – e.g. use the up/down arrow keys on your keyboard to scroll up and down through your command history – remember: Programmers are lazy, so we’ve created many means to make life easy for ourselves 😉

-Gordon

Jeroen
on August 11, 2012 at 5:28 pm said:

Hi Gordon, I’ve been struggling for some time to use PWM from my Java wrapper classes, using the virtual files under /sys/class/gpio. I’ve also looked at your C code, but I realize that my C knowledge is not what it used to be 20 years ago 🙂 Do you think you could give me a simple example of setting a value on a PWM pin using the /sys/class/gpio files?

I would greatly appreciate it!

Jeroen

Gordon
on August 11, 2012 at 5:32 pm said:

Hi Jeroen,

As far as I'm aware, the /sys/class/gpio driver doesn't support PWM. I don't know who actively maintains it though, so I don't know if it's something that's being worked on, or not.

My only suggestion is to use the gpio program called from Java, or write a little C program that does it for you. I've really no idea how to add more functions to Java via C though (or if it's even possible!)

Sorry…

-Gordon

CharlesB
on October 2, 2012 at 2:49 am said:

Your talk about.
gpio edge rising/falling/both/none

We can read the BCM-GPIO pin number 17 in both logic states.

But how do we trap the interrupt triggering .

We tried this on an line and no luck

trap "echo Exit stage right " -17 exit 0

How can we fix this ?

Charles

Gordon
on October 2, 2012 at 6:25 pm said:

I am thinking that you're trying to capture an interrupt in a shell script? If so, then it's not going to work. It really needs a C program to make it work effectively. Look at the wfi.c program in the wiringPi examples directory for an example of how to set it up and make it work.

-Gordon

Lars Kellogg-Stedman
on March 6, 2013 at 2:46 am said:

Maybe add a "wait" subcommand to gpio? Then you could do something like "gpio edge 17 falling" followed by "gpio wait 17", and gpio would block until an interrupt was received.

This would be a simple way of exposing interrupt functionality to shell scripts.

Gordon
on March 6, 2013 at 8:54 am said:

That's in the works. It will be a single command though – maybe just gpio -g wait 17 falling

-Gordon

Rajiv Deo
on December 21, 2013 at 9:57 am said:

Gordon,
When can we get GPIO functionality?
“gpio -g wait 17 falling”

I tried using the command but I get

gpio: unknown command: wait.

Gordon
on December 21, 2013 at 12:21 pm said:

Try:

gpio -g wfi 17 falling

(wfi – wait for interrupt)

also try:

man gpio

-Gordon

Florian
on December 24, 2013 at 1:06 pm said:

Hi there, I am so happy to read the lines below! Thank you Gordon, it is incredible how you engage yourself for wiringPi. Let me say thank you!

I am a not a pure bash-script newbie neither a seniordev, but have done some small sysadmin scripts, .

Could you give me an idea how to catch the interrupt in a bash-script or hint in a way to solve my problem by my self?

My goal is to start and stop some tasks by pressing a button connected to gpio’s.

kind rgeards, flo
any help is verry apreciated

Gordon
on December 25, 2013 at 6:26 pm said:

You can catch a single interrupt using wiringPi’s gpio command:

gpio wfi 5 falling

then it will wait for the falling edge of pin 5 – however the down-side is that it will stall your entire script waiting, but if that’s acceptible, then go for it, but if not, you’ll need to resort to polling the pins – you can use the gpio read command.

Now… If you’re a bash whiz, you could fire-off several bash functions using the & command and have each function wait on a different pin, when wait on one of the functions finishing, but it really starts to get messy – especaily in bash!

-Gordon

Nick
on November 26, 2012 at 10:43 pm said:

Hi Gordon,

Ta for the help – working now I've properly upgraded.
Is there a way or maybe can I propose a new feature to send a parallel byte out to all the 8-pins at once? I am using a fluorescent display's parallel input and it's a bit slow from shell script… Is it possible to set up a pin group and write a byte to that group?

Cheers,

Nick
Dublin

Gordon
on November 26, 2012 at 10:46 pm said:

It's actually something I have in the next version of wiringPi – watch this space as they say!

-Gordon

Throstur Jonsson
on November 27, 2012 at 6:37 pm said:

Hi Gordon,
When I run "gpio load i2c", I get:
gpio: Warning: File not present: /dev/i2c-0
gpio: Warning: File not present: /dev/i2c-1

Is that because I have no i2c devices present?

Second question are there any i2c related functions expected in the WiringPi.

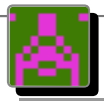
Best regards
TJ

Gordon
on November 27, 2012 at 6:42 pm said:

It could be because you have a rev2 board and I've not tested the i2c loader on a rev 2 board (I know there are i2c differences).

wiringPi will support some i2c functions once I've been able to spend some time with it myself – just some simplified wrappers like I did for SPI.

-Gordon



Throstur Jonsson
on November 28, 2012 at 3:03 pm said:

No actually I'm on a rev1 board.
TJ

Gordon
on November 29, 2012 at 8:48 am said:

OK. The issue is most likely that gpio isn't loading the i2c-dev module. I've fixed this in

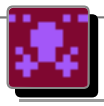
the next release, but for not, try: sudo modprobe i2cdev

-Gordon

jay
on December 3, 2012 at 5:59 pm said:

Hi Gordon, I've been trying to get a rising/falling "interrupt/poll()" using gpio. I set it up with "gpio edge 0 falling". I'm running lastest wheezy on a revision 1 board and gpio 1.4. Do you know of any "C" examples?

Thanks…jay



jay
on December 3, 2012 at 7:11 pm said:

Sorry.. I had looked at wfi.c a while back and forgot about it.. I'll give it a try…

thanks

Gordon
on December 3, 2012 at 7:30 pm said:

Glad you found it. Although I have in in my mind to change how the 'wait for interrupt' mechanism works though as I suspect setting up a concurrent thread is quite an alien concept for people used to microcontrollers…
-Gordon

Justin
on December 7, 2012 at 11:06 pm said:

I am trying out the wfi.c example. Connecting GPIO 17 (P1-11) to switch to GND (P1-06).

Compiled you libs OK. I think. compiled the wfi.c. Ran wfi and get
'Waiting …' never exits. Pushing the switch has no effect. What should happen?

My C is week but I am very interested in this interrupt feature.
Justin

Gordon
on December 7, 2012 at 11:18 pm said:

I'll re-create it tomorrow just to make sure it's still working OK.

However, I'm not that happy with that way of getting interrupts into a program and am working on a Plan B – which will be much easier to use – however that's a week or 2 away…

-Gordon

Bob
on December 16, 2012 at 7:38 pm said:

With the application I'm considering the thing I need is accurate timestamps to go along with the interrupt notification – I seem to recall seeing somewhere an interface that grabbed a timestamp down in the kernel driver and passed it to userspace – does your interface do that or was I seeing that somewhere else?

Thank you BTW for making this level of access possible – the project I'm doing will migrate at some point to a microcontroller but being able to do all the prototyping with Linux and the pi will be a big win.

Gordon
on December 16, 2012 at 9:14 pm said:

You can call gettimeofday() which will give you microsecond accuracy. That ought to be good enough. However there is overhead in that call, and no guarantee that Linux won't deschedule you anyway. You can get good results in Linux, but not as controlled as a microcontroller.

-Gordon

Bob
on December 17, 2012 at 2:33 pm said:

That's why I was thinking that if the gpio kernel driver could grab a timestamp when the interrupt hits the scheduling latency wwouldn't matter. I don't know if that's readily available down at that level though (does the ARM have a RDTSC analog?)

Gordon
on December 17, 2012 at 5:08 pm said:

I'm not that familiar with the architecture of the ARM on the Pi to know if it has a TSC or not – however I do know that there must be something ticking over with microsecond accuracy as the gettimeofday() system call returns it. There is also a spare timer which you can use which is 32-bits wide and counts up (or down) at 1MHz…

However then there is the issue of getting that time stamp back into userland along with the interrupt – and this is where it's going to be a little sticky/tricky… You'll be delving into kernel/module land there, I fear…

So take the interrupt, call gettimeofday() immediately and hope for the best…

-Gordon

Bob
on December 17, 2012 at 5:29 pm said:

I was reading elsewhere that folks see 10s-100s of microsec latency between interrupt and userland code, which is fine for my purposes (event timing, initially to 10ms resolution, eventually to 1ms resolution).

However, I re-found this, which is what I was thinking would be perfect:

http://wiki.gumstix.org/index.php?title=GPIO_Event_Driver

when I get my hardware working I'll take a look at seeing if it is reasonable to port it to the pi.

Gordon
on December 17, 2012 at 5:42 pm said:

It's really the wrong device for this level of accuracy (IMO) However you might want

to look at RISCOS rather than Linux as I understand it's not a pre-emptive multitasking OS, so you may be a step closer to the hardware with a "lighter" OS to help you manage things.

The interrupt code was benchamrked (don't have link to hand, sorry), but it did achieve about 20K interrupts/sec. That should be more than enough for 10ms timings. It's also possible to get sub 10uS timings too, but you need to poll the IO pin rather than wait for an interrupt, and even then, delays might be longer due to pre-emption in the kernel, but switching to real-time mode, and high priority will really help here. The down-side is that a busy loop in a high priority real-time process will more or less stall every other user-land program running…

-Gordon

Bob
on December 17, 2012 at 7:25 pm said:

I agree on the wrong device argument – my eventual goal is to move to something like the TI MSP430 + CC2500 wireless or something similar. The pi provides an excellent way for me to get the basic hardware kinks worked out before I fight the entirely new world of microcontroller wireless (and clock sync :-). Going there also lowers the hardware cost significantly as well as reducing power requirements.

Thanks for your input – I appreciate it.

jason
on February 15, 2013 at 2:03 am said:

Hi Gordon,

Thanks for the fine website, code and tutorials. I just got my RPi today and have been tinkering around with it and having a lot of fun. I was hoping you could give me two minutes of advice…

My goal, and I'm not sure I have the skills, is to use the RPi to control an 1991 scrolling LED display I found for $5 at a yard sale. It has a proprietary EZKEY II keyboard that connects with a 14 pin cable. My thought is I can replicate the required keystrokes using the GPIO outputs and WiringPi and/or Python. I've used a multimeter to capture the pin combinations that reflect the letters on the keyboard. However, I think it uses at least 10 of the pins and if I understand right the RPi can only control 8 pins for this. Beyond this, I'm not 100% sure what I'm up against yet but I'd appreciate any tips you might have.

Gordon
on February 15, 2013 at 8:01 am said:

The Pi can control 17 pins (and a further 4 more on the Rev 2 boards), so you've got plenty of pins avalable. The tricky part is working out the codes, and possibly providing some electrical isolation – especially if the display is working at 5v. The Pi is strictly 3.3v only.

To use more than 8 pins, make sure the I2C and SPI kernel modules are not loaded (they're not by default with Raspbian) and just use those pins as GPIO.

-Gordon

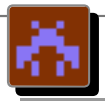
Jason
on February 15, 2013 at 12:35 pm said:

Thanks for the feedback, Gordon. Not being particularly savvy with electrical engineering, if I needed to transmit a particular letter by connecting, say, pin 3 with pin 8, what code would I use in WiringPi? I started to think that I would need to briefly set pin 3 to output with a value of 1 and pin 8 as an input but that doesn't seem to be the right approach. That, and I do need to look at the 5v problem.

Gordon
on February 15, 2013 at 12:53 pm said:

The first thing to work out is the physical wiring. The software protocol appears to be published, but I've not found a pin-out diagram for this 14-pin connector – although googling suggests it might be standard serial (rs232 type) in which case you'd need to use the Pi's serial port, or a USB serial apapter – but without really knowing anything about the display you have, I can't really say much more.

-Gordon



Jason
on February 15, 2013 at 2:25 pm said:

Wow – thanks for looking into this for me. Yeah, info is scarce and some of the docs I've found indicate that it probably is some type of serial, but I don't really know how to determine what pins do what so onto that challenge. I can say that at least 12 of the pins are used based on key presses so it doesn't seem like it is wired like an old keyboard which only uses four or five pins. Thanks again – great work!

Gregor
on February 19, 2013 at 8:39 pm said:

Hi Gordon!

Thank you very much for sharing this very powerful library and gpio tool with us.

Since I am not an C++ programmer I stick to the gpio utility instead of the library and build everything I need around with custom shell scripts. That works well for me.

I miss just one function in gpio. Maybee you can add it in one of the next versions (or as stand-alone tool):

I want to run one (or more) daemons in the background which should trigger a handler script if a pin changes its value to a defined one (edge trigger). Of course I can do that with a polling script, but I am looking for an interrupt based solution to free as many resources of the RPi for other tasks.

I could imagine the following parameters for that function:
– pin number
– edge type (falling/raising/both)
– handler script (eg myhandler.sh, $1=pin, $2=edgetype, $3=pinvalue)
– quit when fired (OPTIONAL, default = yes)
– timeout (OPTIONAL, max. seconds to wait for a trigger before daemon quits, default = -1 / infinite)
– command (set/check/kill): set new trigger, check if a trigger is set, kill trigger (assuming that only one trigger per pin can be defined)

If I understood the library correct it should be able to build such a tool based on the wiringPiISR function. But, as said I am not a C++ programmer ☹

What do you think of such a trigger daemon extension of gpio?

best regards,
Gregor

Gordon
on February 19, 2013 at 9:08 pm said:

What if it were simply:

gpio [-g] wfi [timeout]

and at that point, it just waited for the interrupt then carried on?

Then you could wait on any pin in a script and the script will simply carry on once the interrupt triggers – rather than have the gpio program call your script (which I think gets a bit messy)

You would be able to wait on many different pins in differnet scripts if needed – and examin status to see if it timed out?

That would be relatively easy to implent.. How would it fit in your scenarios?

-Gordon

Gregor
on February 19, 2013 at 9:34 pm said:

wow. fast response 😉
that sounds even better. would that work that way?

…
#define pins
gpio mode 0 in
gpio mode 0 up
gpio edge 0 falling

gpio mode 1 in
gpio mode 1 up
gpio edge 1 falling

gpio mode 2 in
gpio mode 2 up
gpio edge 2 falling

#wait for interrupt
FIRED=`gpio wfi`

#handle interrupt
case $FIRED in
0) echo “pin 0”;;
1) echo “pin 1”;;
2) echo “pin 2”;;
-1) echo “timeout”;;
*) echo “unkown”;;
esac

…

In this case two process would run in parallel. The shell script waiting for gpio wfi and gpio itself. Would that be efficient, even if the RPi has to wait for days to get an event? (I really do not know). Background: I would like to use the RPi as interface for my alarm system. But in parallel other services like Asterisk shell run.

-Gregor

Gordon
on February 19, 2013 at 9:56 pm said:

that's not quite the way I'd envidaged it, but maybe there's something half way. Let me think about it.

Essentially GPIO is stateless between runs, so it doesn't know what you did last time. So to watch multiple pins at the same time, then you'd (i'd) need to make it do that on the command-line.

gpio [-g] wfi 1:f 2:f 3:r

maybe.

Might just need a bit of command-line parsing…

Might also be the wrong tool for the job, but hey… I do describe it as the swiss army knofe of gpio functions in the man page and it never ceases to amaze me what I see people doing with it!

-Gordon

Gregor
on February 20, 2013 at 8:06 am said:

Maybe my initial approach would than be easier.
– no complex parameter parsing
– no state information required
– each instance of gpio monitors only one interupt
– script is only active for start/stop/fired, not all the time

The shell script could look like this:

```
#! /bin/sh

fnStart()
{

gpio -g mode 17 in
gpio -g mode 17 up
gpio edge 17 falling

#NEW:
gpio wfi 17 "$0 interupt0" &

gpio -g mode 18 in
gpio -g mode 18 up
gpio edge 18 falling

#NEW:
gpio wfi 18 "$0 interupt1" &

}

fnStop()
{
killall gpio
}

fnHelp()
{
echo "…."
}

fnInterupt0()
{
#do what must be done
}
```

```
fnInterupt1()
{
#do what must be done
}

#Main

case $1
start) fnStart();;
stop) fnStop();;
interupt0) fnInterupt0;;
interupt1) fnInterupt1;;
*) fnHelp();;
esac
exit0
```

Gordon
on February 20, 2013 at 9:01 am said:

As I said, there is no state rmemberd between GPIO executions, so 3 calls to gpio to set the directions, etc. isn't visible to the gpio command that waits for the interrupt – but I guess it could just wait for it and hope for the best.

Really you should be writing stuff like this in C though – if someone had asid to me 10 years ago that a shell-script would be able to capture interrupts I'd have laughed… but here we are doing just that 😉

But rather than run the gpio command in the background and have it call another progral (something I'm not keen to do at all), you could simply replace that with the suggestion I made earlier, so you'd write a separate script for each interrupt that went something like:

```
#!/bin/sh
while true; do
gpio -g wfi 17 rising
./dispatcher int17
done
```

and so on. So you break your program down to several smaller scripts and you can then test each one individually too.

And you can haev these as functions inside one big script and still execute the functions with & so it's the function being backgrounded and the function that's calling the script, not the gpio program.

Part of the reason I don't want gpio to call other scripts is that it runs as root (set uid), so I'd need to drop back to the uid of the calling user, then call the script – that's not hard, but it's one more thing to do and I do feel it's getting away from what I wanted gpio to do – it really started off as nothing more than a test program for the wiringPi library, but like all good things, has mutated beyond it's original use into a bit of a little monster, really 🙂

-Gordon

Gregor
on February 20, 2013 at 2:05 pm said:

thanks for your detailed explanation.
Got the point 😉

br
gregor

Dauhee
on February 21, 2013 at 9:33 pm said:

Hi Gordon,

Is there any way to disable interrupts after setting one up i.e.
wiringPiISR (4, INT_EDGE_FALLING, &someFunction)

I was hoping for something quicker than doing a system call

Much appreciated.

Gordon
on February 21, 2013 at 9:38 pm said:

If you run the gpio command:

gpio unexportall

that will turn them all off.

but that's not very quick, so the answer probably is "not really" until your program exists…

-Gordon

killyox
on March 19, 2013 at 3:34 pm said:

Hi, i'm working on a little project to manage my GPIO pins with a PHP interface.
I would like to print the mode of each pins in a table, but i don't find any way to get this information except on the "gpio readall" function.
So, my question is : "Is there a way to get the mode of 1 pin ?"

Thanks !

Gordon
on March 19, 2013 at 3:37 pm said:

There is an un-documented wiringPi C function to return the mode – it's what the gpio program uses. If you study the source of the gpio program you'll see how to use it. You could just extract that and use it stand-alone to return the value or return it back into a php script.

-Gordon

killyox
on March 19, 2013 at 6:30 pm said:

Ok, thank you, i'll try to find it in the source code.
And thank you for your job with wiringPi, and for your quick answer !

killyox
on April 2, 2013 at 11:12 am said:

I found it ! Thank you very much. I modified the GPIO program to add a “gpio readmode pin” command, and it worked fine.
Thank you again !

wu3mi
on April 29, 2013 at 9:17 am said:

I need a little help:
I need a simply script on a raspberry pi …
the “gpio read 6” comment gives me a “1” if high and a “0” if low. so in need a “on” and a “off”. I wrote some scripts, but without any success.
May someone could give me a hint or anything else …
thanks

Crosseye Jack
on April 30, 2013 at 5:06 pm said:

https://github.com/killyox/WiringPi/commit/7b51897971485bbe88823026e6dcdd94fec26ad6

update the gpio.c file from this update and then rebuild. I also made the same change in my own fork before checking the pull requests :-p

Hope it helps.

Crosseye Jack
on April 30, 2013 at 5:20 pm said:

Blahhh Misread your post… Reread it, why not just issue ‘gpio write 0/1’ You will need to set the pin to an output first like ‘gpio mode out’.

Here is a simple read and toggle script i made which uses the modded gpio.c which tells me if the pin is already set as an output http://pi.crosseyejack.com/files/read-n-toggle.txt

Ronnie
on July 10, 2013 at 1:45 pm said:

How do I know what the mode of each pin is? When I do a gpio readall command they come back is IN with the exception of the TxD and RxD which are ALT0. Is there a command to read how they are set or is this just known, then you have to keep up with this in your program? If that is the case, what kind of house keeping should be going on (for instance, if you set the pin modes for program A, then at the end should you set everything back to default)?

Gordon
on July 12, 2013 at 11:51 am said:

The gpio command reads the pin modes – so have a look at the source code of the gpio

command to see how it's doing that… Essentically there are un-documented functions inside wiringPi to return the data needed.

My take on the modes is that EVERY program should explicitly set the pins to the mode it needs. You must never rely on them already being set for you. Just set the mode and be confident that that's the right thing to do.

-Gordon

karim
on July 17, 2013 at 11:16 am said:

Hi, i need a help in my project . i connected the 2 buttons to pi (pin 23 and 24). i have to develop code to detect the event on the buttons.
i used the wiringPiISR (BUTTON_PIN, INT_EDGE_FALLING, &myInterrupt) but function works for one pin only how to do this???i need multipul interrupts..

Gordon
on July 17, 2013 at 12:53 pm said:

You need to write multi-threaded code to handle this. So create a thread for each interrupt you need to sample. At its simplest, each thread can just set a global variable, but this is really no better than just polling the switches every few mS, so the threads can do more – e.g. if the buttons perform different tasks, each thread can do that task independant of the other and the main program.

Read up on pthreads, mutexes and semaphores.

-Gordon

Pieter Zanstra
on August 17, 2013 at 10:22 am said:

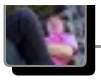
First many thanks for this great product. I intend to use it to drive my Somfy sunshades from OpenRemote (see my post http://www.openremote.org/display/forums/Z-Wave+control+with+Razberry?focusedCommentId=22875010#comment-22875010)

I use the following script:

```
#!/bin/sh
PIN=$1
gpio mode $PIN out
gpio write $PIN 1
sleep 0.1
gpio write $PIN 0
```

This works fine if called from the Raspberry Pi command line.
However if I call it from Javascript (Google V8 engine) with a system() call, I do get error 32512. This error does not appear if I put a # sign before the last line.
Anybody out here who has a clue of what may be wrong?
Thanks,
Pieter

Gordon



on August 21, 2013 at 9:50 pm said:

Make sure the $PATH is correct, or explicitly use /usr/local/bin/gpio …

This may not be the issue though – seems odd about the # on the last line…

-Gordon

Alec
on August 29, 2013 at 11:06 pm said:

I've understood that you can write a value like: gpio pwm 1 512
Does that mean that the duty cycle is 50%?
But how do I specify the cycle time?

Gordon
on August 30, 2013 at 8:00 am said:

The range register is the total number of ticks per cycle – it defaults to 1024, so 512 is 50% – but by default it's running in "balanced" mode – so wil be 0101010… etc. for 1024 bits. Mark:Space ration would be 512 0's followed by 512 1's.

-Gordon

Wynneth
on September 16, 2013 at 7:03 am said:

I'm running into a problem with the gpio wfi command. I have a pin set to input which occasionally receives input, and another pin set to output which is active several hours of the day. Using the gpio wfi command seems to react to both pins. To be clear, from the command line if I do:
gpio wfi 6 rising

Then whether I get input on pin 6 OR if I activate pin 3 – the command interrupts. What I'm trying to do with this is use it in a simple shell script that waits for input on pin 6 then launches a separate script, then waits again for input. I have a python script which does the same thing but it consumes too much CPU. Example script below:

#!/bin/bash

while true
do

gpio wfi 6 rising
python /usr/share/stuff/doathing.py

done

Gordon
on September 16, 2013 at 11:53 am said:

I can't replicate it.

So in onw window, I run

gpio mode 6 input

gpio mode 6 down # Set pull-down
gpio wfi 6 rising

If, in another window I run

gpio mode 6 up # Change down to pull up

then it triggers – this is what I'd expect.

So I reset it with:

gpio move 6 down ; gpio wfi 6 rising

then poke the other pins:

for i in 0 1 2 3 4 5 7; do gpio mode $i out ; done
for i in 0 1 2 3 4 5 7; do gpio write $i 1 ; done
for i in 0 1 2 3 4 5 7; do gpio write $i 0 ; done
for i in 0 1 2 3 4 5 7; do gpio write $i 1 ; done

Nothing happens on the input pin.

My thoughts are that maybe you're not using a pull up/pull-down resistor and there there might be noise on the wires as a result.

Also, I think Python resets all GPIOs when it starts, so that might be having an effect too.

-Gordon

Wynneth
on September 18, 2013 at 6:52 am said:

Hmm, let me be more specific in my current setup and see if we can find the problem together. I have a powerswitch tail (ac mains relay unit) connected to the raspberry pi gpio 3 and gnd pin. I have a wireless receiver that outputs ~2v at about 2mA connected to gpio 6 and gnd pin. Gpio 3 is in output mode. Gpio 6 is set to input, pull down. (I've also tried this with a physical 10k resistor tying gpio 6 to gnd, just to see.) I get the expected result when the sensor on gpio 6 is activated. However, when I activate gpio 3 then I also get a reaction from gpio wfi 6 rising. New example tested below:

gpio mode 3 output
gpio mode 6 input
gpio mode 6 down
gpio wfi 6 rising

I would love it if I'm missing something absolutely obvious and you could point it out.
in separate window:
gpio write 3 1
and at that point wfi is interrupted

Marvin
on October 5, 2013 at 9:54 pm said:

Hi Wynneth,

I am still learning this software but I have a good electronics background. It just might be that you are generating a big spike when you switch pin 3 and this is being picked up by your input pin.
Try adding a capacitor, say 0.1 µF to your input pin to suppress noise. Fit it as close to the Pi as possible.

-Marvin

Martin
on November 17, 2013 at 11:30 am said:

Gordon, just for my understanding, is the “gpio” command (as installed and activated by your “wiringPi” git download and build) software that you have written yourself or is “gpio” command a standard software package for raspberry that could be activated quite independently of your “wiringPi” modules?

Gordon
on November 17, 2013 at 3:39 pm said:

The gpio command is part of wiringPi which I wrote – it started off as my test program to test wiringPi functions then changed into a general purpose swiss army knife of a gpio program. I’ve seen people use it entirely in shell scripts to do pretty interesting automation, etc.

-Gordon

Amir
on December 12, 2013 at 1:13 am said:

Hi I’m trying to use a gpio_5 as output in c++.
* Gpio_5 is high when I detect a picture after doing some processing with opencv from an cam.
*Gpio_5 is low when the picture is not detected.

my code is :

```
/*******************************************************/
#include

#define GPIO_output 5
#define Low 0
#define HIGH 1
#define OUTPUT 0

int detect_picture(){
.
.
.
.
}

int main(){
pinMode(GPIO_output, OUTPUT);
digitalWrite(GPIO_output, LOW);
while(1){
if(detect_picture()){
digitalWrite(GPIO_output, high);
}else{
digitalWrite(GPIO_output, LOW);
}
}
return(0);
}
/*******************************************************/
```

befor including the controme of gpio I was compiling my code with this commande : g++ `pkg-config

–cflags opencv` amir.cpp `pkg-config –libs opencv`

after adding the gpio commande function I tried :
g++ `pkg-config –cflags opencv` amir.cpp -lwiringPi `pkg-config –libs opencv`
but it don't compile the wiring pi function still be unknown

Gordon
on December 12, 2013 at 3:05 pm said:

You've not initialised wiringPi.

Put:
wiringPiSetup() ;
in your code before you do any pinMode, etc. calls.

If adding in -lwiringPi causes an error, then it means wiringPi is not installed correctly. Make sure you have downloaded it and run the ./build script.

-Gordon

Matthias L
on December 19, 2013 at 4:09 pm said:

Hello!

I just started testing your WiringPi library, and it's really useful and extremely simple to utilise. I've never tried Wiring on Arduino before, as I'm an old school AVR programmer who likes to set his fuses himself 🙂 But for the RaspberryPi this is really helpful, as most people don't want to fiddle with internal BCM settings anyway, thanks a lot for this excellent tool!
Though I think in the future I'll rather let the Pi communicate over UART or SPI with an external AVR, which can handle time-critical stuff, e.g. communicating with sensors which have their own somehow different from the standard implementation of I²C, better. However I might try creating POSIX threads to simulate AVR Timer ISRs, but I'm not sure about how accurate and reliable this is.
Have you made any experiences with that?

Looking forward to hearing from you!
Best regards from Austria
Matthias

PS: Is there a way to utilise UART from the gpio command line tool?

Gordon
on December 20, 2013 at 12:22 pm said:

experience of posix threads? Yes. Look in the softPwm, softTone libraries as well as the helpers I write to make it slightly easier to use them in wiringPi.

-Gordon