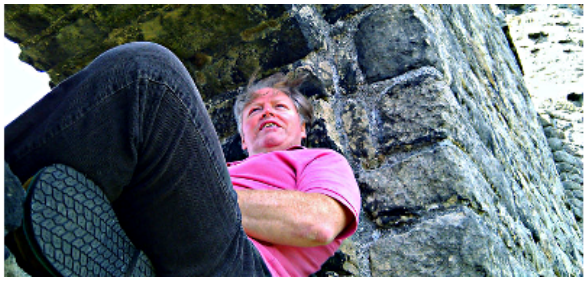
Wiring Pi

GPIO Interface library for the Raspberry Pi



Software PWM Library

WiringPi includes a software-driven PWM handler capable of outputting a PWM signal on any of the Raspberry Pi's GPIO pins.

There are some limitations… To maintain a low CPU usage, the minimum pulse width is 100μS. That combined with the default suggested range of 100 gives a PWM frequency of 100Hz. You can lower the range to get a higher frequency, at the expense of resolution, or increase to get more resolution, but that will lower the frequency. If you change the pulse-width in the driver code, then be aware that at delays of less than 100μS ***wiringPi*** does it in a software loop, which means that CPU usage will rise dramatically, and controlling more than one pin will be almost impossible.

Also note that while the routines run themselves at a higher and real-time priority, Linux can still affect the accuracy of the generated signal.

However, within these limitations, control of a light/LED or a motor is very achievable.

To use:

```
#include <wiringPi.h>
#include <softPwm.h>
```

When compiling your program you must include the *pthread* library as well as the *wiringPi* library:

```
cc -o myprog myprog.c -lwiringPi -lpthread
```

You must initialise ***wiringPi*** with one of *wiringPiSetup()*, *wiringPiSetupGpio()* or *wiringPiSetupPhys()* functions. *wiringPiSetupSys()* is not fast enough, so you must run your programs with sudo.

Some expansion modules may also be fast enough to handle software PWM – it has been tested with the MCP23S17 GPIO expander on the PiFace for example.

The following two functions are available:

- **int softPwmCreate (int pin, int initialValue, int pwmRange) ;**

  This creates a software controlled PWM pin. You can use any GPIO pin and the pin numbering will be that of the ***wiringPiSetup***() function you used. Use 100 for the **pwmRange**, then the value can be anything from 0 (off) to 100 (fully on) for the given pin.

  The return value is 0 for success. Anything else and you should check the global *errno* variable to see what went wrong.

- **void softPwmWrite (int pin, int value) ;**

  This updates the PWM value on the given pin. The value is checked to be in-range and pins that haven't previously been initialised via softPwmCreate will be silently ignored.

Notes

- Each "cycle" of PWM output takes 10mS with the default range value of 100, so trying to change the PWM value more than 100 times a second will be futile.
- Each pin activated in softPWM mode uses approximately 0.5% of the CPU.
- There is currently no way to disable softPWM on a pin while the program in running.
- You need to keep your program running to maintain the PWM output!