Raspberry gPIo

CONTRIBUTORS: JIMB0, MTAYLOR

C (WiringPi) API

On this page we'll discuss some of the most useful functions provided by the WiringPi library. It's tailored to look a lot like Arduino, so if you've done any Arduino programming some of this may look familiar.

Setup Stuff

To begin, you'll need to include the library. At the beginning of your program, type:

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

After you've included the library, your first steps should be to initialize it. This step also determines which **pin numbering scheme** you'll be using throughout the rest of your program. Pick **one of these** function calls to initialize the library:

```
wiringPiSetup(); // Initializes wiringPi using wiringPi's simlified number system.
wiringPiSetupGpio(); // Initializes wiringPi using the Broadcom GPIO pin numbers
```

WiringPi's simplified number system introduces a third pin-numbering scheme. We didn't show it in the table earlier, if you want to use this scheme, check out their pins page for an overview.

Pin Mode Declaration

To set a pin as either an input or output, use the `pinMode([pin], [mode])` function. Mode can be either `INPUT`, `OUTPUT`, or `PWM_OUTPUT`.

For example, to set pin 22 as an input, 23 as an output, and 18 as a PWM, write:

```
wiringPiSetupGpio()
pinMode(17, INPUT);
pinMode(23, OUTPUT);
pinMode(18, PWM_OUTPUT);
```

Keep in mind that the above example uses the Broadcom GPIO pin-numbering scheme.

Digital Output

The `digitalWrite([pin], [HIGH/LOW])` function can be used to set an output pin either HIGH or LOW. Easy enough, if you're an Arduino user.

To set pin 23 as HIGH, for example, simply call:

```
digitalWrite(23, HIGH);
```

PWM ("Analog") Output

For the lone PWM pin, you can use `pwmWrite([pin], [0-1023])` to set it to a value between 0 and 1024. As an example…

```
pwmWrite(18, 723);
```

…will set pin 18 to a duty cycle around 70%.

Digital Input

If you're an Arduino veteran, you probably know what comes next. To read the digital state of a pin, `digitalRead([pin])` is your function. For example…

```
if (digitalRead(17))
    printf("Pin 17 is HIGH\n");
else
    printf("Pin 17 is LOW\n");
```

…will print the status of pin 22. The `digitalRead()` function returns 1 if the pin is HIGH and 0 if it's LOW.

Pull-Up/Down Resistors

Need some pull-up or pull-down resistors on your digital input? Use the `pullUpDnControl([pin], [PUD_OFF, PUD_DOWN, PUD_UP])` function to pull your pin.

For example, if you have a button on pin 22 and need some help pulling it up, write:

```
pullUpDnControl(17, PUD_UP);
```

That comes in handy if your button pulls low when it's pressed.

Delays

Slowing down those blinking LEDs is always useful – assuming you actually want to differentiate between on and off. WiringPi includes two delay functions to choose from: `delay([milliseconds])` and `delayMicroseconds([microseconds])`. The standard delay will halt the program flow for a specified number of milliseconds. If you want to delay for 2 seconds, for example, write:

```
delay(2000);
```

Or you can use `delayMicroseconds()` to get a more precise, microsecond-level delay.

Now that you know the basics, let's apply them to an example piece of code.

← PREVIOUS PAGE
C (WIRINGPI) SETUP

VIEW AS A SINGLE PAGE

NEXT PAGE →
C (WIRINGPI) EXAMPLE