Raspberry gPIo

CONTRIBUTORS: JIMB0, MTAYLOR

FAVORITE 5

C (WiringPi) Example

The intention of WiringPi is to make your I/O code look as Arduino-ified as possible. However keep in mind that we're no longer existing in the comfy confines of Arduino – there's no `loop()` or `setup()`, just `int main(void)`.

Follow along here as we create an example C file, incorporate the WiringPi library, and compile and run that program.

Create blinker.c

Using the terminal, navigate to a folder of your choice and create a new file – "blinker.c". Then open that file in a text editor (Nano or Leafpad are included with Raspbian).

```
pi@raspberrypi ~/code $ mkdir c_example
pi@raspberrypi ~/code $ cd c_example
pi@raspberrypi ~/code/c_example $ touch blinker.c
pi@raspberrypi ~/code/c_example $ leafpad blinker.c &
```

The commands above will open your "blinker.c" file in Leafpad, while leaving your terminal functioning – in-directory – in the background.

Program!

Here's an example program that includes a little bit of everything we talked about on the last page. Copy and paste, or write it yourself to get some extra reinforcement.

```
#include <stdio.h>    // Used for printf() statements
#include <wiringPi.h> // Include WiringPi library!

// Pin number declarations. We're using the Broadcom chip pin numbers.
const int pwmPin = 18; // PWM LED - Broadcom pin 18, P1 pin 12
const int ledPin = 23; // Regular LED - Broadcom pin 23, P1 pin 16
const int butPin = 17; // Active-low button - Broadcom pin 17, P1 pin 11

const int pwmValue = 75; // Use this to set an LED brightness

int main(void)
{
    // Setup stuff:
    wiringPiSetupGpio(); // Initialize wiringPi -- using Broadcom pin numbers

    pinMode(pwmPin, PWM_OUTPUT); // Set PWM LED as PWM output
    pinMode(ledPin, OUTPUT);     // Set regular LED as output
    pinMode(butPin, INPUT);      // Set button as INPUT
    pullUpDnControl(butPin, PUD_UP); // Enable pull-up resistor on button

    printf("Blinker is running! Press CTRL+C to quit.\n");

    // Loop (while(1)):
    while(1)
    {
        if (digitalRead(butPin)) // Button is released if this returns 1
        {
            pwmWrite(pwmPin, pwmValue); // PWM LED at bright setting
            digitalWrite(ledPin, LOW);     // Regular LED off
        }
        else // If digitalRead returns 0, button is pressed
        {
            pwmWrite(pwmPin, 1024 - pwmValue); // PWM LED at dim setting
            // Do some blinking on the ledPin:
            digitalWrite(ledPin, HIGH); // Turn LED ON
            delay(75); // Wait 75ms
            digitalWrite(ledPin, LOW); // Turn LED OFF
            delay(75); // Wait 75ms again
        }
    }

    return 0;
}
```

Once you've finished, **Save** and return to your terminal.

Compile and Execute!

Unlike Python, which is an interpreted language, before we can run our C program, we need to build it.

To **compile our program**, we'll invoke gcc. Enter this into your terminal, and wait a second for it to finish compiling:

```
pi@raspberrypi ~/code/c_example $ gcc -o blinker blinker.c -l wiringPi
```

That command will create an executable file – “blinker”. The “-l wiringPi” part is important, it loads the wiringPi library. A successful compilation won’t produce any messages; if you got any errors, try to use the messages to track them down.

Type this to **execute your program**:

```
pi@raspberrypi ~/code/c_example $ sudo ./blinker
```

The blinker program should begin doing it’s thing. Make sure you’ve set up the circuit just as modeled on the hardware setup page. Press the button to blink the LED, release to have it turn off. The PWM-ing LED will be brightest when the button is released, and dim when the button is pressed.

If typing all of that code in a bland, black-and-white, non-highlighting editor hurt your brain, check out the next page where we introduce a simple IDE that makes your programming more efficient.

← PREVIOUS PAGE
C (WIRINGPI) API

VIEW AS A SINGLE PAGE

NEXT PAGE →
USING AN IDE!