# Lab Sheet: LED and Button control in C

This lab sheet deals with controlling a button and an LED, both wired up through a breadboard to a Raspberry Pi 2, using C programs as discussed in the corresponding tutorials. You need to exercise your C programming skills, to implement the logic of the control. You need to draw on the BCM periperals documentation to interact with the external devices. And you are likely to need `gdb` in order to debug your C program.

The tasks should be performed on a Raspberry Pi 2, which you can get on loan from the department, hooked up via a KVM to a monitor and keyboard of one of the machines in the Linux lab (EM 2.50). You can try the same exercise in your own time with the Pi2 directly connected to a keyboard and monitor, e.g. at home. In each case, you need to wire up the external devices (LED and Button) using a breadboard and jumper cables as described below.

Wiring up external devices

An **LED**, as output device, should be connected to the RPi2 using **GPIO pin 12.** You will need a resistor to control the input to the LED. Lookup the wiring diagram in the handout from "Adventures in Raspberry Pi", Section 8, Fig 8-8, but note the difference in pin usage here.

The Fritzing diagram below visualises this wiring.

GPIO12 (yellow) to LED
GPIO16 (yellow) to Button
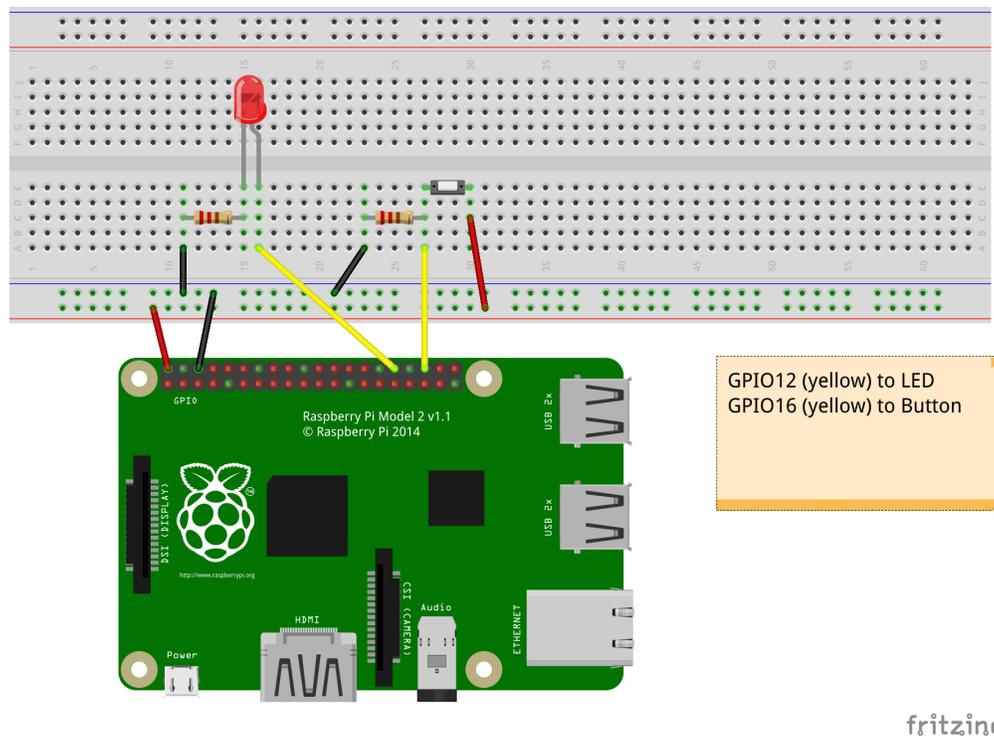
Test the pins

There are several ways how you can test that the wiring is correct. Pick one of the options below.

Option 1: Python scripts

In Tutorial 1 we discussed a short Python script that makes the on-board led blink. It is available from the sample sources section for Tutorial 1. This can be easily modified for other PINs, that are wired to an external LED: you only have to change the pin number. The hand-out chapter in the "Adventures in Raspberry Pi" discusses this in more detail.

Option 2: WiringPi library and interface

Since we are programming in C and Assembler in most parts of the course, it is recommended that you install the `wiringPi` library by Gordon Henderson. Follow these instuctions on how to download and install the wiringPi library. This is also a very useful source of example code, so you may want to look-up the directory `wiringPi/examples`, with several example programs, and `wiringPi/wiringPi` with the core implementation in C.

Using `wiringPi` you can test your LED like this:

```
> gpio -g mode 12 out
> gpio -g write 12 1
```

and your LED (on Pin 12) should turn on. But remember this is just for testing. The Learning Objective of the lab is to achieve this kind of control through your own C program, without using external libraries. To close the session, turn the LED off again, like this

```
> gpio -g write 12 0
```

Another useful command is

```
> gpio readall
```

which shows the settings for all GPIO pins, and whether they are configured for input or output.

Option 3: Linux SysFS

The second half of Tutorial 1 discusses how you can use the Linux SysFS filesystem to control the LED directly from the command-line. You don't need any special libraries or packages for this. Just type (you need to do this as root, therefore the first command `sudo su` to give you a root shell):

```
> sudo su
> echo 12 > /sys/class/gpio/export
> echo out > /sys/class/gpio/gpio12/direction
> echo 1 > /sys/class/gpio/gpio12/value
```

and your LED (on Pin 12) should turn on. See the the sample sources section for Tutorial 1 to download a shell script doing this in one go. Once finished, close your session like this

```
echo 0 > /sys/class/gpio/gpio12/value
echo 12 > /sys/class/gpio/unexport
```

Step 1: Developing a C program for a blinking LED (Week 3)

The first task of this lab is to write a C program to make an LED blink in the same way as discussed in class for the on-chip ACT LED (pin 47). The LED should be connected via the breadboard to **pin 12**. The code needs to configure this pin for output, and then in a loop turn the LED on/off, with a fixed delay between state changes.

Start from the sample code discussed in class: tut_led.c, which implements this control for the ACT LED (pin 47). You will have to change the code to work with pin 12, as discussed in the Tutorial 2 slides, discussing LEDs

Step 2: Developing a C program to control the LED using a button (by Week 5)

The main task of this lab is to write a C program to read input from a button and let the LED blink, whenever you press the button. The button should be connected via the breadboard to **pin 16** of the RPi2 (see the Fritzing diagram on the first page for the complete wiring needed in this lab).

The C code for controlling a button is discussed in Tutorial 3. Start from the sample code discussed in class: tut_button.c, which implements a button-controlled LED and adjust it to use the pins specified above.

Note, that this sample source has a "bug" in that the LED is on at the start and is turned off by pressing the button. The desired behaviour should be the inverse: the LED should be off at the start and turned off when pressing a button. It's suggested that you first make the small change to the code, to achieve this behaviour, and then move on to the main task, of a button-controlled LED, with pin usage discussed above.

Tips on developing the code

For debugging it is recommended that you work with print messages to stderr to get information about the behaviour of the program, printed to the screen. One big advantage of using the RPi2 with Raspbian 7 is that you can use a familar development environment and use the programming techniques that you are familar with already. To debug problems with the interface to the devices, you will most likely need to use gdb to see what exactly is going on. Look-up the earlier slides on gdb usage from Greg's part of the course, and check the "External Links" section on the main Course Information page:

http://www.macs.hw.ac.uk/~hwloidl/Courses/F28HS/index.html#external

At the end of this exercise, you should be able to demo that with your program running, a button press will make the LED blink. This exercise builds up to the final CW, where you will need similar functionality.

Different forms of pin numberings

Sadly, there are 3 different ways to number the GPIO pins, and this can be confusing when starting to wire-up a configuration. In all exercises, we will be using the **BCM numbering**. For picture, covering all 3 numberings on the RPi, see this page.

- The *physical numbering* is the most intuitive one: it numbers the pins by physical location on the device, starting in a corner, with even numbers on the out- and odd numbers on the in-side;

- The *BCM numbering* uses the numbers given to the pins in Broadcom's technical manuals for the BCM2835/2836 chips. It's the most widely used numbering and somtimes identified by putting the BCM in front of the pin number, e.g. BCM 12.

- The *wiringPi numbering* numbers the data pins from 1 onwards, and is used per default in the wiringPi library.

For this exercise, the pin numbers, in the different numberings are:
LED on pin 12 (BCM), 26 (wiringPi), 32 (physical)
Button on pin 16 (BCM), 27 (wiringPi), 36 (physical)