F28HS Hardware-Software Interface: Systems Programming

Hans-Wolfgang Loidl

School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh

Semester 2 — 2024/25

[0]No proprietary software has been used in producing these slides

Outline

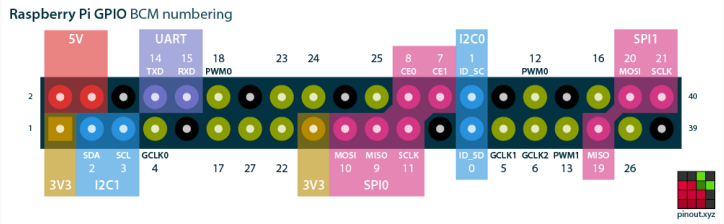
Tutorial 1: Using Python for GPIO Control

- In this first tutorial we will get started with programming the RPi2 to control **output** devices.
- We will use **Python** as programming language and existing libraries for controlling the GPIO pins on the RPi2, which simplifies the programming considerably.
- The main learning objective for this course, however, is to achieve such control by using C and Assembler, and we will focus on these languages for the remaining tutorials.

GPIO pins of the RPi2

- The Raspberry Pi 2 has 40 **General Purpose Input/Output (GPIO)** pins.
- These can be used to control a range of devices, or to receive data from such devices.
- You need to use the jumper cables in the Raspberry Pi2 starter kit to connect devices.
- In this first tutorial we will attach an LED and make it blink

Map of the GPIO pins of a RPi2



[0] Available from http://pinout.xyz/

Electronics basics and wiring diagrams

For a good, introductory discussion on how to wire-up external devices to the Raspberry Pi 2 follow this link.
You can get a small CamJam EduKit, including LEDs, button, resistors and jumper cables, from ThePiHut. These are all include in your RPi2 starter kit, so you don't need these, but they may be useful for experimentation.
The following slides summarise the main steps from this web page.

Connecting an LED to the RPi2

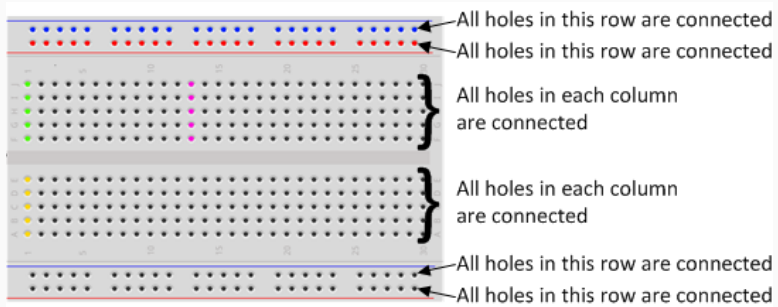
As the first exercise in controlling an external LED we need:

- a Breadboard
- an LED
- a 330 ohm resistor
- two Male-Female jumper wires

[0] For details see this page

How to use a Breadboard

The breadboard is a way of connecting electronic components to each other without having to solder them together.

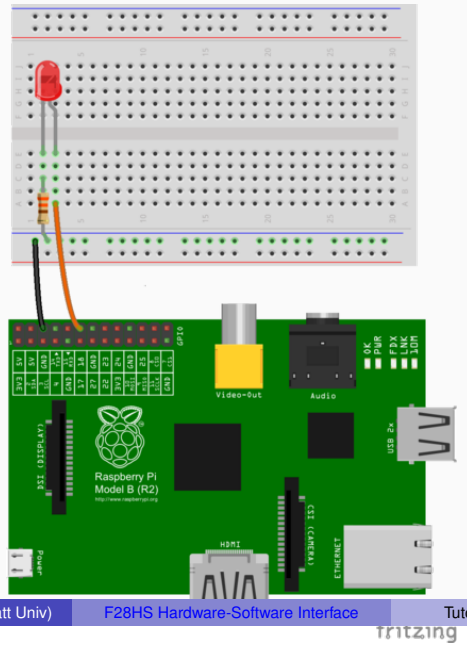


Using a breadboard, like the one above, simplifies the wiring, especially for larger projects (as in CW2).

External devices: LED and Resistor

- In this tutorial, we only want to connect an LED to the RPi2, using a breadboard.
- Note that the LED has two legs of different lengths.
- The **longer leg**, is always connected to the **positive** supply of the circuit.
- The **shorter leg** is connected to the **negative** side of the power supply, known as ‘ground’.
- You **must** protect the LED with a resistor, otherwise the LED will try to draw more power than needed and **might burn out the RPi2**.
- Putting the resistors in the circuit will ensure that only this small current will flow and the Pi will not be damaged.

Fritzing Diagrams



The Fritzing Diagram Explained

The Fritzing diagram on the previous slide shows how to wire-up external devices, i.e. which pins to connect to which rows/columns on the breadboard to complete a circuit.

- Use one of the jumper wires to connect a **ground pin** to the rail, marked with **blue**, on the breadboard. The female end goes on the Pi’s pin, and the male end goes into a hole on the breadboard.
- Then connect the **resistor** from the same row on the breadboard to a column on the breadboard, as shown in the diagram.
- Next, push the **LEDs** legs into the breadboard, with the long leg (with the kink) on the right.
- Lastly, complete the circuit by connecting **pin 18** to the **right hand leg of the LED**. This is shown here with the orange wire.

Using Python to control a GPIO pin

Details of the software setup can be found in Chapter 8 of “Adventures in Raspberry Pi”.

- First install the Python library for GPIO support:

```
>  sudo apt-get install python-RPi.GPIO
```

- To test the version of the RPi you have do the following:

```
>>> import RPi.GPIO as GPIO
>>> GPIO.RPI_REVISION
2
```

Python code to control a GPIO pin

First we import all required libraries and set some constants, in particular the pin number that we use for the LED. We need to specify which numbering of the pins to use, and then setup the connection.

```
#!/usr/bin/python

# External module imports
import RPi.GPIO as GPIO
import time

# Pin Definitons:
ledPin = 23 # Broadcom pin 23 (P1 pin 16)

# Pin Setup:
GPIO.setmode(GPIO.BCM) # Broadcom pin-numbering scheme
GPIO.setup(ledPin, GPIO.OUT) # LED pin set as output

# Initial state for LEDs:
GPIO.output(ledPin, GPIO.LOW)
```


Python code to control a GPIO pin (cont'd)

The main part of the program is the loop below, which continously turns the LED on and off, using a delay of 75ms:

```
while True:
  try:
    GPIO.output(ledPin, GPIO.HIGH)
    time.sleep(0.075)
    GPIO.output(ledPin, GPIO.LOW)
    time.sleep(0.075)
  except KeyboardInterrupt: # If CTRL+C is pressed, exit
      cleanly:
    GPIO.cleanup() # cleanup all GPIO
```


Tutorial 1: Using the Linux FS for GPIO Control

- One design principle in Linux is to control and view system information through files.
- We have seen this in class by, e.g. looking up details about the CPU by `cat /proc/cpuinfo`
- This tutorial will demonstrate **how filesystem operations can be used to easily control GPIO pins on the RPi 2**

NB: You need a Linux kernel with support for SysFS. Raspbian 7, as we use it in the kit handed out for this course, provides this.
To check whether SysFS is supported do:

```
> sudo sh -c "cat /lib/modules/`uname -r`/build/.config | fgrep SYSFS"
```

and look for a line like this

```
> CONFIG_SYSFS=y
```


Basics of SysFS

- The Linux kernel provides several RAM based file systems.
- These file systems provide low-level hardware information and in some cases a way to control these.
- The basic programmer API is to use the system-level `read(2)` and `write(2)` commands on the files in these file systems.
- Each file has a special meaning to enable hardware interaction.
- The `read` and `write` function calls, result in callbacks in the Linux kernel which has access to the corresponding value.
- The benefit of using the read and write functions is that the user space has a lot of tools available to send data to the kernel space (e.g. `cat`(1), `echo` (1)).

[0]From: Kernel Space - User Space Interfaces

SysFS filesystem

- `SysFS` was designed to represent the whole device model as seen from the Linux kernel
- It contains information about devices, drivers and buses and their interconnections.
- `SysFS` is heavily structured and contains a lot of links.

The main subdirectories of interest for us are:

- `sys/block/` all known block devices such as hda/ ram/ sda/
- **sys/class/** for each device type there is a subdirectory: for example /printer or /sound
- `sys/device/` all devices known by the kernel, organised by the bus they are connected to

Controlling GPIO pins using SysFS

- We want to control the GPIO pins on the RPi2 using the SysFS interface.
- To this end we need to:
  - tell the system that we want to access a specific pin ("export" this device to the SysFS)
  - configure the mode of the pin, as either `in` or `out`
  - read/write from/to the device using standard tools such as `echo` and `cat`
  - finally, remove the device from the filesystem ("unexport" it from SysFS)
- All of these steps can be done as one-liners from the command-line
- No additional libraries need to be installed

NB: This interface is useful for **testing** a wiring or debugging the hardware. The main learning objective of the course is to learn how to do the above operations directly on the device (in C or Assembler), without involving the operating system or an external library at all.

The Shell Code for Controlling a GPIO

```
# the pin to control
PIN=23

# make this pin available through the SysFS
echo $PIN > /sys/class/gpio/export

# now, set this pin to output
echo out > /sys/class/gpio/gpio${PIN}/direction

# write a value to this pin
echo 1 > /sys/class/gpio/gpio${PIN}/value

# wait for some seconds
sleep 3s

# write a value to this pin
echo 0 > /sys/class/gpio/gpio${PIN}/value

# make this pin unavailable through the SysFS
echo $PIN > /sys/class/gpio/unexport
```

NB: You need to run this as root, i.e. type `sudo sh sysfs_23.sh`
NB: Version with a pin as param: `sudo sh sysfs.sh -p 23`

Other useful information in the SysFS

You can get the information about the model like this:

```
> cat /sys/firmware/devicetree/base/model
```

You can get the information about the cache line size like this (not enabled under Raspbian by default):

```
> cat /sys/devices/system/cpu/cpu0/cache/index0/coherency_line_size
```

On Debian-based systems, such as Ubuntu, you can also get this info by typing:

```
>  getconf -a
```

Useful information in the ProcFS filesystem

The `/proc` filesystem provides information about the **processor**:

```
> cat /proc/cpuinfo
```

gives detailed information about the processor, split by core, eg. each core is an `ARMv7 Processor` and the `neon` instruction set is enabled.
Detailed information about the memory is available via:

```
> cat /proc/meminfo
```

shows that the total memory is `949408 kB`, i. e. ca. 1GB.

```
>  cat /proc/iomem
```

shows the structure of the memory, including the location of the GPIO memory.

The ProcFS filesystem

There is a special subdirectory: `/proc/sys`. It allows to configure a lot of parameters of the running system.

```
> cat /proc/sys/kernel/osrelease
```

tells us that the kernel version is `3.18.11-v7+`.
There are a lot of files in this directory, showing the current state of the kernel. For interacting with the kernel, the `sysctl` interface should be used.
The `sysctl` infrastructure is designed to configure kernel parameters at run time. E. g.

```
> sysctl --all
```

lists all kernel parameters.

Further Reading & Deeper Hacking

- Kernel Space - User Space Interfaces, Ariane Keller
- ProcFS Kernel Docu
- Linux Device Drivers, 3rd ed, Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman
- More detailed documentation on SysFS
- Shell code samples with SysFS