My first Raspberry Pi hands-on session

Paolo Burgio
paolo.burgio@unimore.it

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA
High Performance Real Time Lab

Programming is a skill best acquired by practice and example rather than from books.

ALAN TURING

Our guy (Pi4)

General Purpose I/O ports (GPIO)

Eth

USB

HDMI/screen

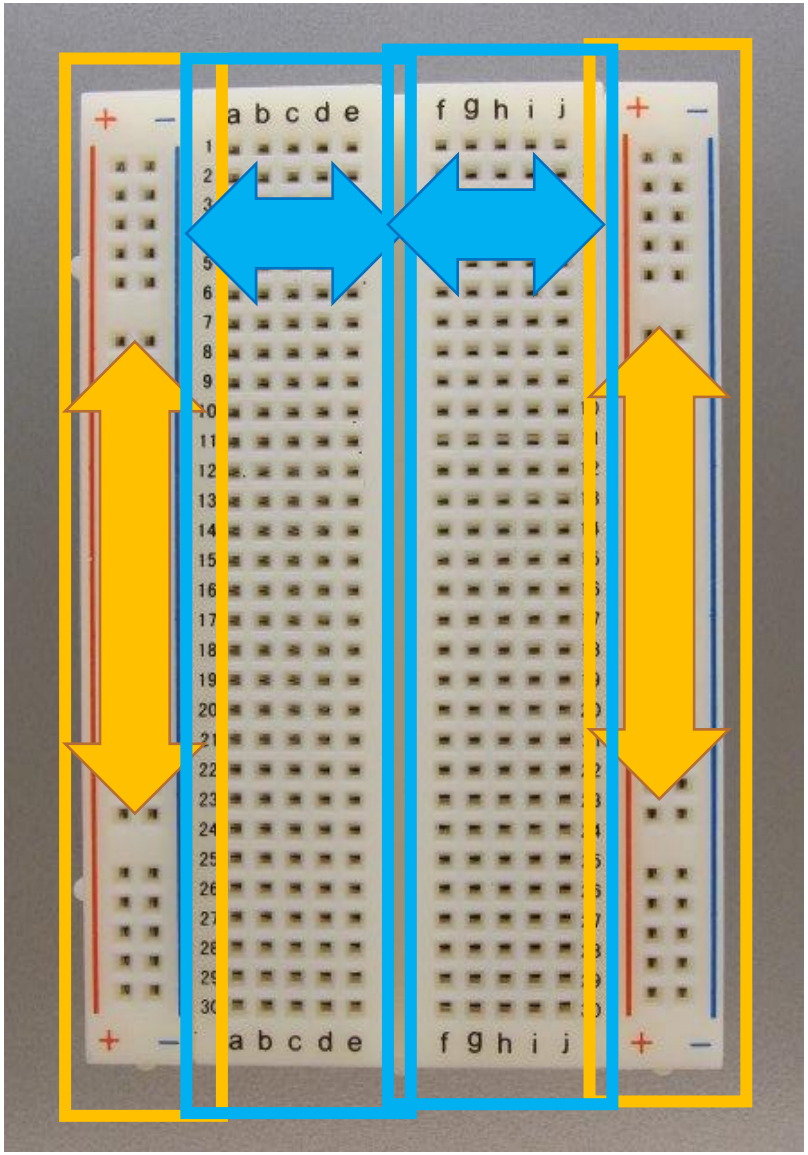
Pi3
Broadcom BCM2837 SoC
- 64-bit ARM Cortex-A53
- @1.2 GHz
- 512 KiB shared L2 $

Pi4
Broadcom BCM2837 SoC
– 64-bit ARM Cortex-A72
- @1.5 GHz
- 1MiB shared L2 $

Breadboard

Provides electrical connectivity

- Vertical vs. horizontal rails
- (Typically, power vs other)
- Can use jumper wires



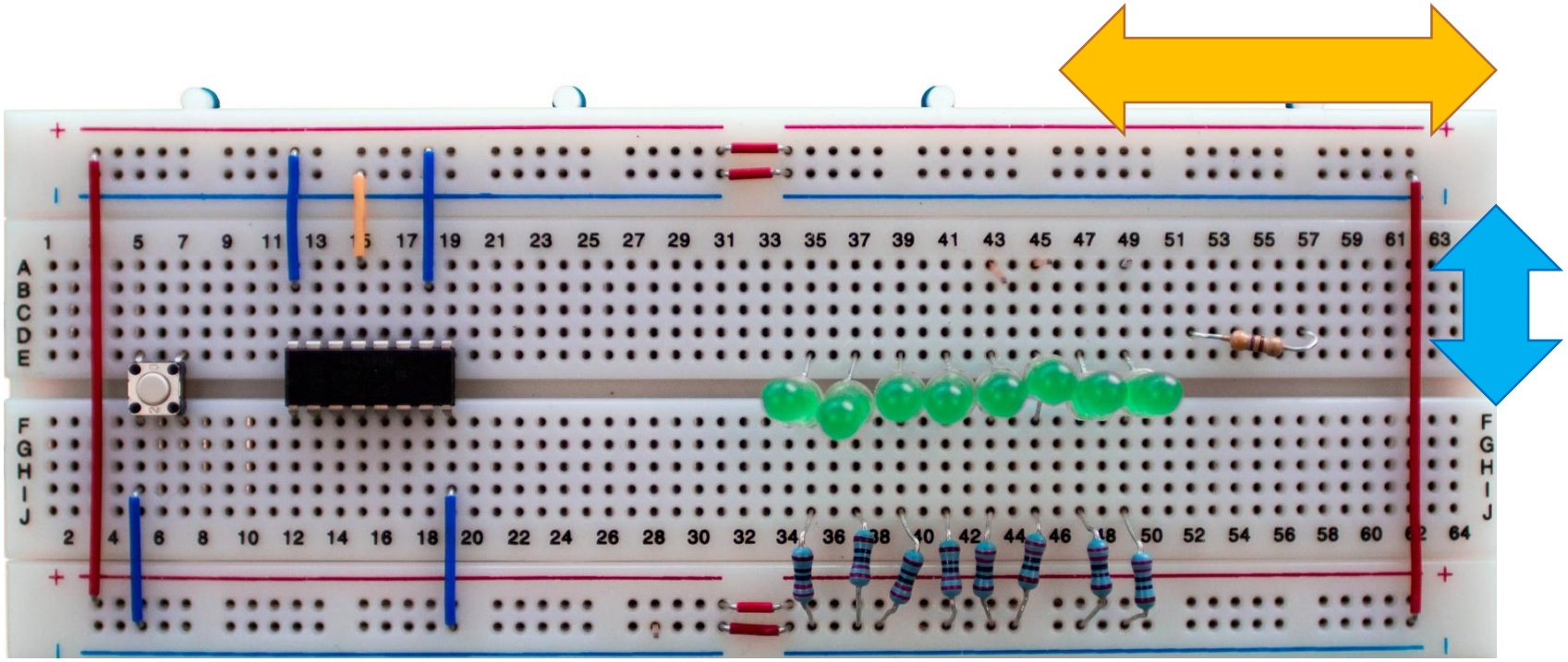
Breadboard

The two sides of the + and - rails are wired together

› Typically, used for power/GND

Brought to the internal rails with jumper wires

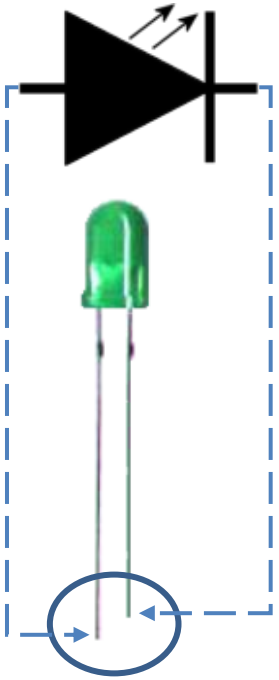
› Where core/chip and other stuff reside



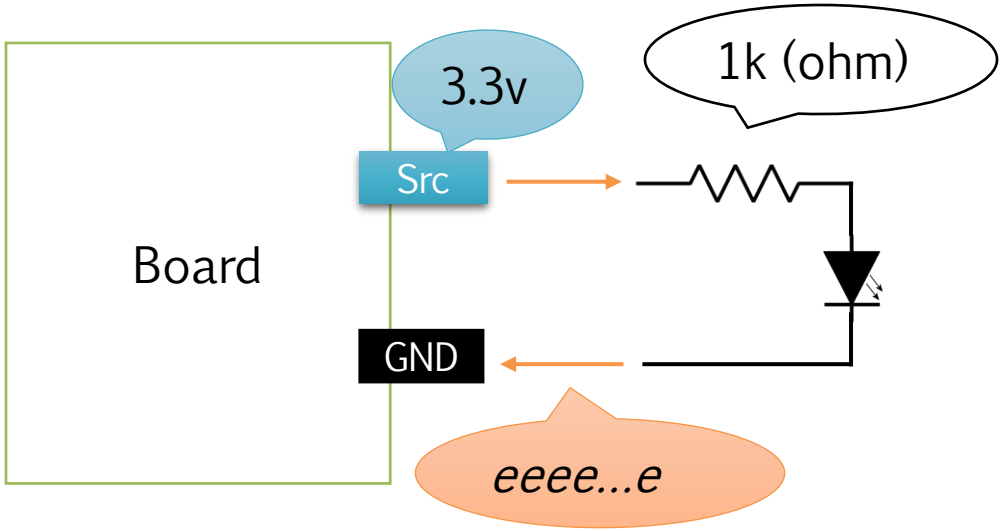
Finally…LEDs

Light Emitting Diodes

- You feed with electrons; they light up
- They have a side!!!!
- They need a resistance to lower the charge



Wrong wiring => you burn them…



General Purpose I/O Ports

Our interface towards the external world

- https://pinout.xyz/pinout/#
- BCM vs. Standard Wiring

Raspberry Pi Pinout

3v3 Power	1	2	5v Power
GPIO 2 (I2C1 SDA)	3	4	5v Power
GPIO 3 (I2C1 SCL)	5	6	Ground
GPIO 4 (GPCLK0)	7	8	GPIO 14 (UART TX)
Ground	9	10	GPIO 15 (UART RX)
GPIO 17	11	12	GPIO 18 (PCM CLK)
GPIO 27	13	14	Ground
GPIO 22	15	16	GPIO 23
3v3 Power	17	18	GPIO 24
GPIO 10 (SPI0 MOSI)	19	20	Ground
GPIO 9 (SPI0 MISO)	21	22	GPIO 25
GPIO 11 (SPI0 SCLK)	23	24	GPIO 8 (SPI0 CE0)
Ground	25	26	GPIO 7 (SPI0 CE1)
GPIO 0 (EEPROM SDA)	27	28	GPIO 1 (EEPROM SCL)
GPIO 5	29	30	Ground
GPIO 6	31	32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33	34	Ground
GPIO 19 (PCM FS)	35	36	GPIO 16
GPIO 26	37	38	GPIO 20 (PCM DIN)
Ground	39	40	GPIO 21 (PCM DOUT)

Software

Operating system

- Debian-based GNU/Linux Distro called **Raspberry Pi OS**
  - Aka *Raspbian*
- Also Ubuntu and Win10 IoT are supported
- (and many more…)

(A number of) dev environments

- **Standard GCC toolchain**
- Arduino IDE (micro-kernel)
- Google's TensorFlow for AI ;)
- ….

Pi/ssh setup

- The easiest way is to compile our code directly on Pi
  - Cross-compilation requires to install the custom toolchain

- We might connect Pi to HDMI, USB keyboard + mouse…

…or…

- Setup a ssh daemon, and access it via ssh

HOWTO

- Connect to eth, and log in (just one first time), to identify your IP address
  - Should set static IP? It typically never changes anyhow, if you do point-to-point
- Create `/boot/ssh` empty file to enable ssh daemon at boot
  - Manuall start: `$ sudo service ssh start`
- Username: **pi** - Password: **raspberry**

WiringPi

- Library to interact with I/O
- Uses “progressive” wiring

Raspberry Pi GPIO Header

BCM	WiringPi	Name	Physical		Name	WiringPi	BCM
		3.3v	1	2	5v		
2	8	SDA.1	3	4	5V		
3	9	SCL.1	5	6	0v		
4	7	1-Wire	7	8	TxD	15	14
		0v	9	10	RxD	16	15
17	0	GPIO. 0	11	12	GPIO. 1	1	18
27	2	GPIO. 2	13	14	0v		
22	3	GPIO. 3	15	16	GPIO. 4	4	23
		3.3v	17	18	GPIO. 5	5	24
10	12	MOSI	19	20	0v		
9	13	MISO	21	22	GPIO. 6	6	25
11	14	SCLK	23	24	CE0	10	8
		0v	25	26	CE1	11	7
0	30	SDA.0	27	28	SCL.0	31	1
5	21	GPIO.21	29	30	0v		
6	22	GPIO.22	31	32	GPIO.26	26	12
13	23	GPIO.23	33	34	0v		
19	24	GPIO.24	35	36	GPIO.27	27	16
26	25	GPIO.25	37	38	GPIO.28	28	20
		0v	39	40	GPIO.29	29	21
BCM	WiringPi	Name	Physical		Name	WiringPi	BCM

WiringPi API

Include library header

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

(In desktop environments doesn't exist, so you shall use macro to remove this code, e.g., `NO_PI`)

Init library, and **every** GPio port

```
wiringPiSetup(); // Init lib
pinMode(0, OUTPUT); // GPio 0 is output port
```

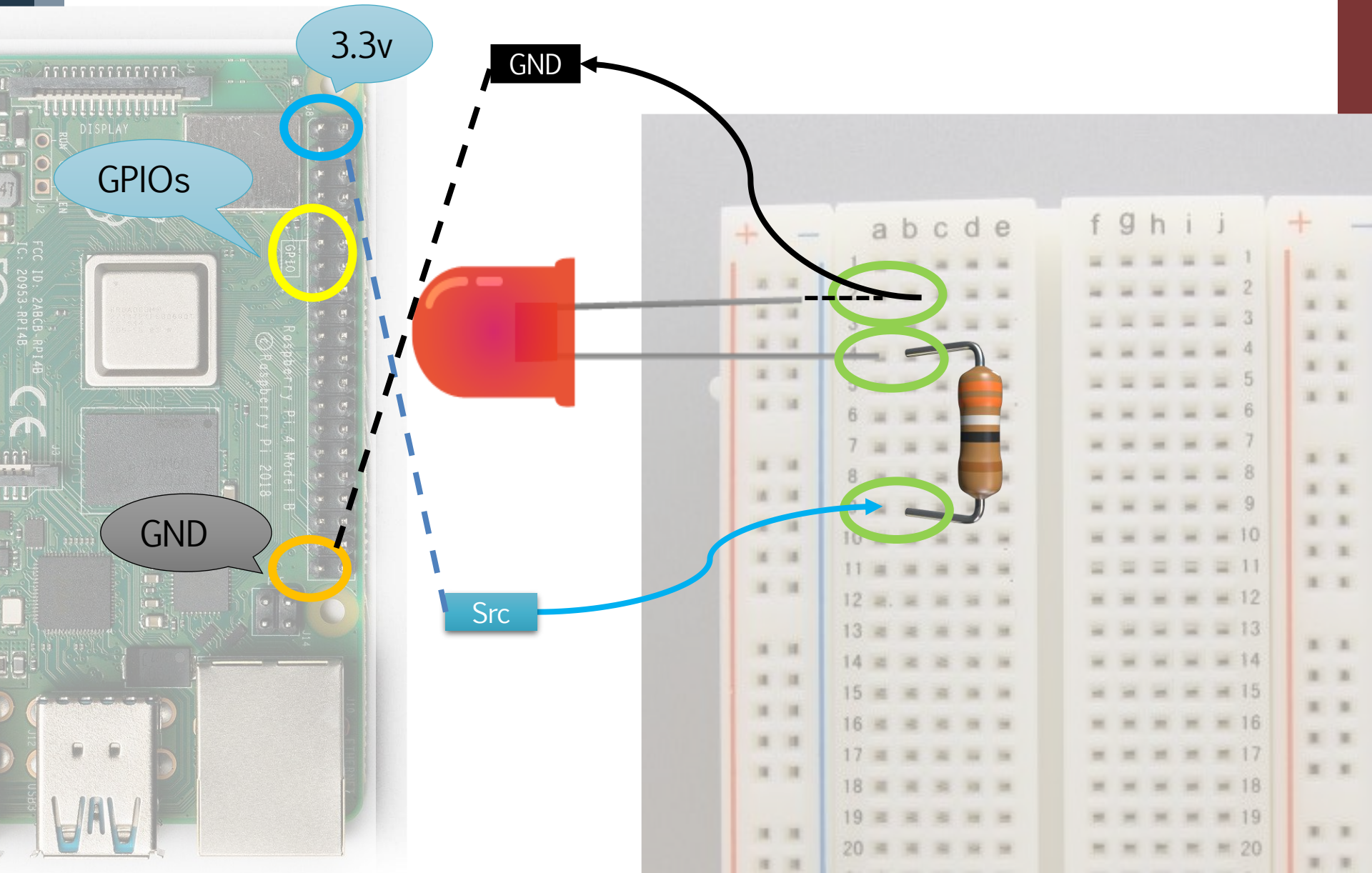
Write to port

```
digitalWrite(0, true); // Set port 0
```

Link library

```
$ gcc ..... -l wiringPi
```

E/E system



Exercise

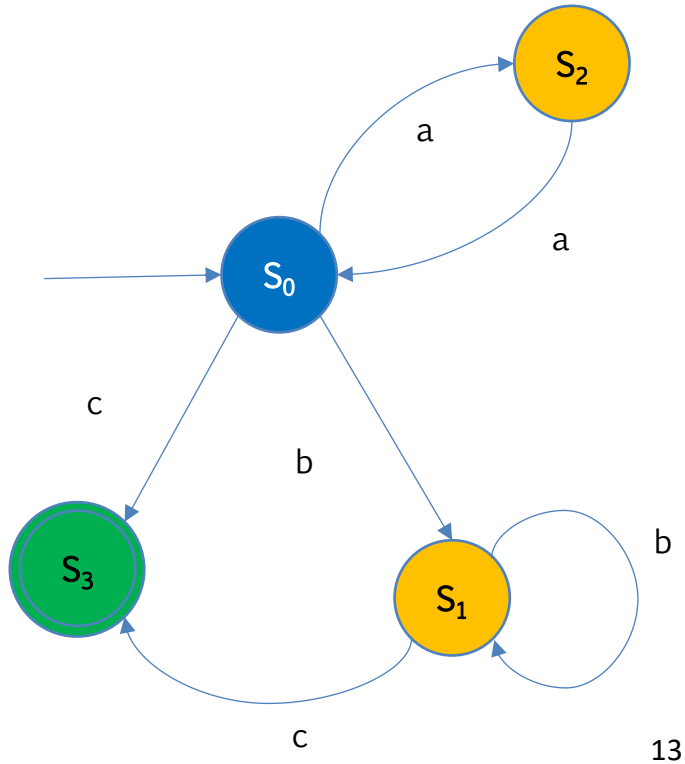
Let's code!

› Implement the Moore machine of the FSM that understands whether a word is from L

"Identify even sequences of a (even empty),

followed by one, or more, or no, b, ended by c"

› ..and turns on the corresponding led color
  - Blue => GPIO 0
  - Red (error state) => GPIO 1
  - Yellow => GPIO 2
  - Green => GPIO 3



Java on RPi

Set-up

See also “IoT” course from Prof. Picone

Install java using apt tool

```
$ sudo apt update && sudo apt install default-jdk
```

Set up `JAVA_HOME` env var

```
$ export JAVA_HOME="/usr/lib/jvm/default-java"
$ export PATH=$PATH:$JAVA_HOME/bin
```

› (tip: set it in `~/.bashrc`)

Install Maven

› Download, and follow instructions, from http://maven.apache.org/install.html

› (don’t forget to set `M2_HOME` and `PATH` vars in `~/.bashrc`)

Recap: eclipse Paho

Eclipse Paho Java client

› MQTT client lib

Download and install

› https://www.eclipse.org/paho/index.php?page=clients/java/index.php

Dependencies

› SLF4J

PI4J: GPIO with Java

The Pi4J Project

Java I/O library for the Raspberry Pi

What it is

› A library to control GP I/O on a Raspberry Pi

How to install the lib

```
$ curl -sSL https://pi4j.com/install | sudo bash
```

› Target path `/opt/pi41/lib`

Examples and code snippets

```
$ git clone https://github.com/Pi4J/pi4j.git
```

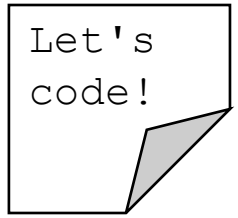
› Compile with Maven

```
$ mvn package
```

› Run (Beware: default example uses GPIO #1)

```
$ java -cp /opt/pi4j/lib/\*:target/??.jar ControlGpioExample
```

Exercise



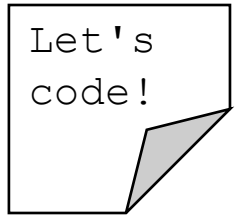
Traffic light

- (Re)implement the traffic light FSMs so that we can dynamically switch among them using a MQTT topic sent by an external entity
  - I use MQTT Explorer, under Win, you can also use `AuthProducer.java`
  - `AuthConsumer.java`
- TLs also publish their status on a MQTT topic, together with their time-to-change
  - In JSON (`JsonProducer.java`)
- You will have to implement your JSON model `TrafficLightDescriptor.java`
  - Can get inspiration by https://github.com/HiPeRT/MASA_protocol/blob/master/include/objects.hpp

Some hints/design rules

- Every TL has a unique ID, and sub-id ("orientation")
- They are paired two-by-two, to implement 4-way crossroads
- You will also need to create a protocol

How to run the examples



Find them in `Code/` folder from the course website

For C++: compile

Use when compiling with desktop/laptop

```
$ gcc code.cpp [-D NO_PI] -o code -Wall -l stdc++ [-l wiringPi]
```

Use when compiling on Raspberry Pi

Run (bash/cygwin)

```
$ ./code[.exe]
```

For Java/Maven: compile

```
$ mvn package
```

Run (command line)

Add folders separated by ':' or';' on Win

Your tar,and the full name of your class

```
$ java -cp /path/to/libs/\*:target/????.tar <main-class>
```

References

Course website

- http://hipert.unimore.it/people/paolob/pub/Industrial_Informatics/index.html

My contacts

- paolo.burgio@unimore.it
- http://hipert.mat.unimore.it/people/paolob/

Resources

- https://www.digikey.com/en/maker/blogs/2019/how-to-use-gpio-on-the-raspberry-pi-with-c
- http://maxembedded.com/2014/07/using-raspberry-pi-gpio-using-python/
- https://www.digikey.com/en/resources/conversion-calculators/conversion-calculator-resistor-color-code
- Pi wires -> https://pinout.xyz/pinout/pin11_gpio17# | http://wiringpi.com/
- Pi4J -> https://pi4j.com/1.2/install.html | https://pi4j.com/1.2/example/control.html | https://github.com/Pi4J/pi4j
- A "small blog -> http://www.google.com