

STM 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 (IoT node)

SIM/Wifi

2x USB
(Use this one!)

General Purpose
I/O ports (GPIO)

Reset ;)

Core SoC

- 32-bit ARM Cortex
- 1MiB Flash mem
- 128kB SRAM

General P
I/O ports



Software

Micro-kernel

- › No OS, need to flash all memory regions

ST proprietary

- › STM32 CubeIDE
- › Debug via STLink (won't see this)

How to work

- › No way is to compile our code directly on IoT Node
- › Cross-compilation *via* the CubeIDE
- › Flash the whole OS+program via USB



A simple application



Create a new "Blink" project

- › File -> New Project
- › Then. Select the MCU (or the board)
- › **DO NOT initialize the peripherals in default mode!!!** (for this time..)

IDE STM32 Project

Target Selection

Select STM32 target or STM32Cube example

MCU/MPU SelectorBoard SelectorExample SelectorCross Selector

MCU/MPU Filters

Commercial Part Number

STM32L475

PRODUCT

Package

Core

Coprocessor

MEMORY

Flash From 256 to 1024 (kBytes)

EEPROM = 0 (Bytes)

FeaturesBlock DiagramDocs & ResourcesDatasheet

STM32L4 Series

STM32L475VGT6

Ultra-low-power with FPU Arm Cortex-M4 MCU 80 MHz with 1 Mbyte of Flash memory, USB OTG, DFSDM

ACTIVE

Product is in mass production

Unit Price for 10kU (US\$) : 6.3301

Boards: B-L475E-IOT01A1 - B-L475E-IOT01A2

LQFP 100 14x14x1.4 mm

The STM32L475xx devices are the ultra-low-power microcontrollers based on the high-performance Arm® Cortex® M4 32-bit RISC core operating at 80 MHz. The Cortex-M4 core features a Floating point unit (FPU) single precision which supports all Arm® single-precision data-processing instructions and also implements a full set of DSP instructions and a memory protection unit (MPU) which enhances application security.

MCUs/MPUs List: 14 items

Display similar items

Commercial Part No	Reference	Marketing	Unit Price for 10...	Board	Package	Flash
STM32L475RET6TR	STM32L475R...	Active	4.6939		LQFP 64 10x10x1.4 mm	512 kByt...
STM32L475RGT6	STM32L475R...	Active	5.8839		LQFP 64 10x10x1.4 mm	1024 kB...
STM32L475RGT6TR	STM32L475R...	Active	5.8839		LQFP 64 10x10x1.4 mm	1024 kB...
STM32L475RGT7	STM32L475R...	Active	6.2958		LQFP 64 10x10x1.4 mm	1024 kB...
STM32L475RGT7TR	STM32L475R...	Active	6.2958		LQFP 64 10x10x1.4 mm	1024 kB...
STM32L475VCT6	STM32L475V...	Active	4.4263		LQFP 100 14x14x1.4 mm	256 kB...
STM32L475VET6	STM32L475V...	Active	5.1402		LQFP 100 14x14x1.4 mm	512 kB...
STM32L475VET6TR	STM32L475V...	Active	5.1402		LQFP 100 14x14x1.4 mm	512 kB...
STM32L475VGT6	STM32L475V...	Active	6.3301	B-L475E-IOT01A1 B-L475E-IOT01A2	LQFP 100 14x14x1.4 mm	1024 kB...

STM32L475?? MCU
(or B-L475E-IOT01A1
board)



IDE

workspace_1.10.1 - Blink/Core/Src/main.c - STM32CubeIDE

File Edit Source Refactor Navigate Search Project Run Window Help

The screenshot shows the STM32CubeIDE interface. The Project Explorer on the left displays the project structure for 'Blink'. The 'main.c' file is highlighted, and a blue callout points to it with the text 'Main file (the one with "main")'. Below it, a green box highlights a list of generated files: 'stm32l4xx_hal_msp.c', 'stm32l4xx_it.c', 'syscalls.c', 'systemem.c', and 'system_stm32l4xx.c'. A green callout points to this list with the text 'Some generated files. Do not touch them...'. The main editor window shows the code in 'main.c'. A red box highlights a section of code starting with 'while (1)'. A red callout points to this section with the text 'Initialization/setup + infinite loop (Arduino-like)'. The code in the editor is as follows:

```
83  /* USER CODE END SysInit */
84
85  /* Initialize all configured peripherals
86  MX_GPIO_Init();
87  /* USER CODE BEGIN 2 */
88
89  /* USER CODE END 2 */
90
91  /* Infinite loop */
92  /* USER CODE BEGIN WHILE */
93
94  while (1)
95  {
96      /* USER CODE BEGIN WHILE */
97
98      /* USER CODE END WHILE */
99
100     // write pin state
```



Configure LEDs

We want to create **an alias** for GPIOs

- › So we don't need to change code when we change LEDs

workspace_1.10.1 - Device Configuration Tool - STM32CubeIDE

File Edit Navigate Search Project Run Window Help

Project Explorer

- Blink
 - Binaries
 - Includes
 - Core
 - Inc
 - Src
 - main.c
 - stm32l4xx_hal_msp.c
 - stm32l4xx_it.c
 - syscalls.c
 - systemem.c
 - system_stm32l4xx.c
 - Startup
 - Debug
 - Blink.ioc
 - Blink Debug.launch

Config file (.ioc)

Blink.ioc - Pinout & Configuration

Pinout & Configuration

Categories A->Z

- System Core
 - DMA
 - GPIO
 - NVIC
 - RCC
 - SYS
 - TSC
 - WWDG

Configure Pinout

GPIO Mode and Configuration

Configuration

Group By Peripherals

- GPIO

Search Signals

Search (Ctrl+F)

Show only Modified Pins

Pin...	Signal...	GPIO ...	GPIO ...	GPIO ...	Maxi...	Fast ...	User ...	Modified
PA5	n/a	Low	Output...	No pu...	Low	n/a	LED1	✓

PA5 Configuration :

GPIO output level

Low

Here, PA5 pin

Pinout view

System

STM32L475VGTx



Let's configure PA5

GPIO

Search Signals

☐ Show only Modified Pins

Pi...	Signal...	GPIO ...	GPIO ...	GPIO...	Maxi...	Fast ...	User ...	Modifi...
PA5	n/a	Low	Outpu...	No pu...	Low	n/a	LED1	<input checked="" type="checkbox"/>

GPIO output level:

GPIO mode:

GPIO Pull-up/Pull-down:

Maximum output speed:

User Label:

IDE Question

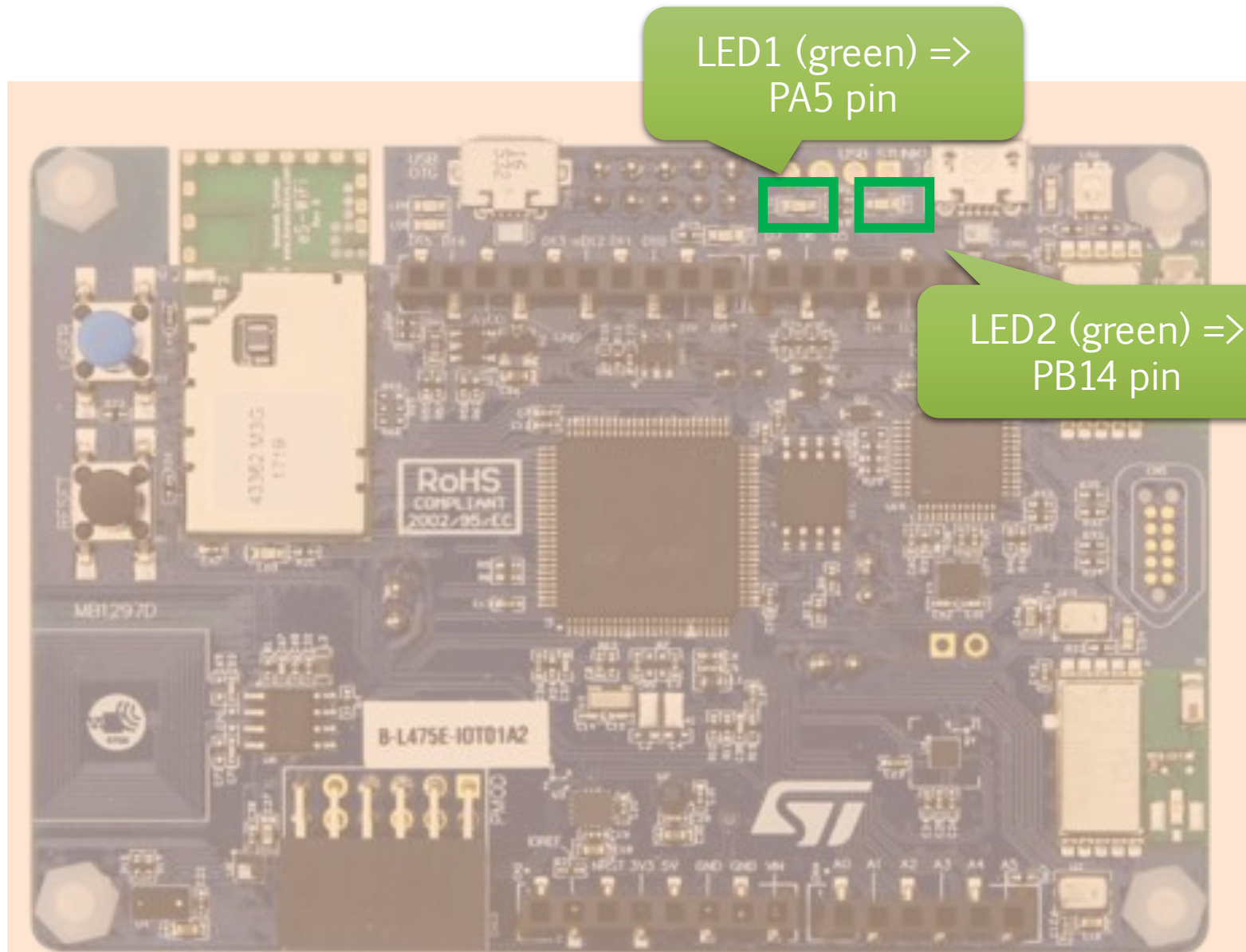
Do you want generate Code?

☐ Remember my decision

That's our Alias!!



Leds and GPIOs





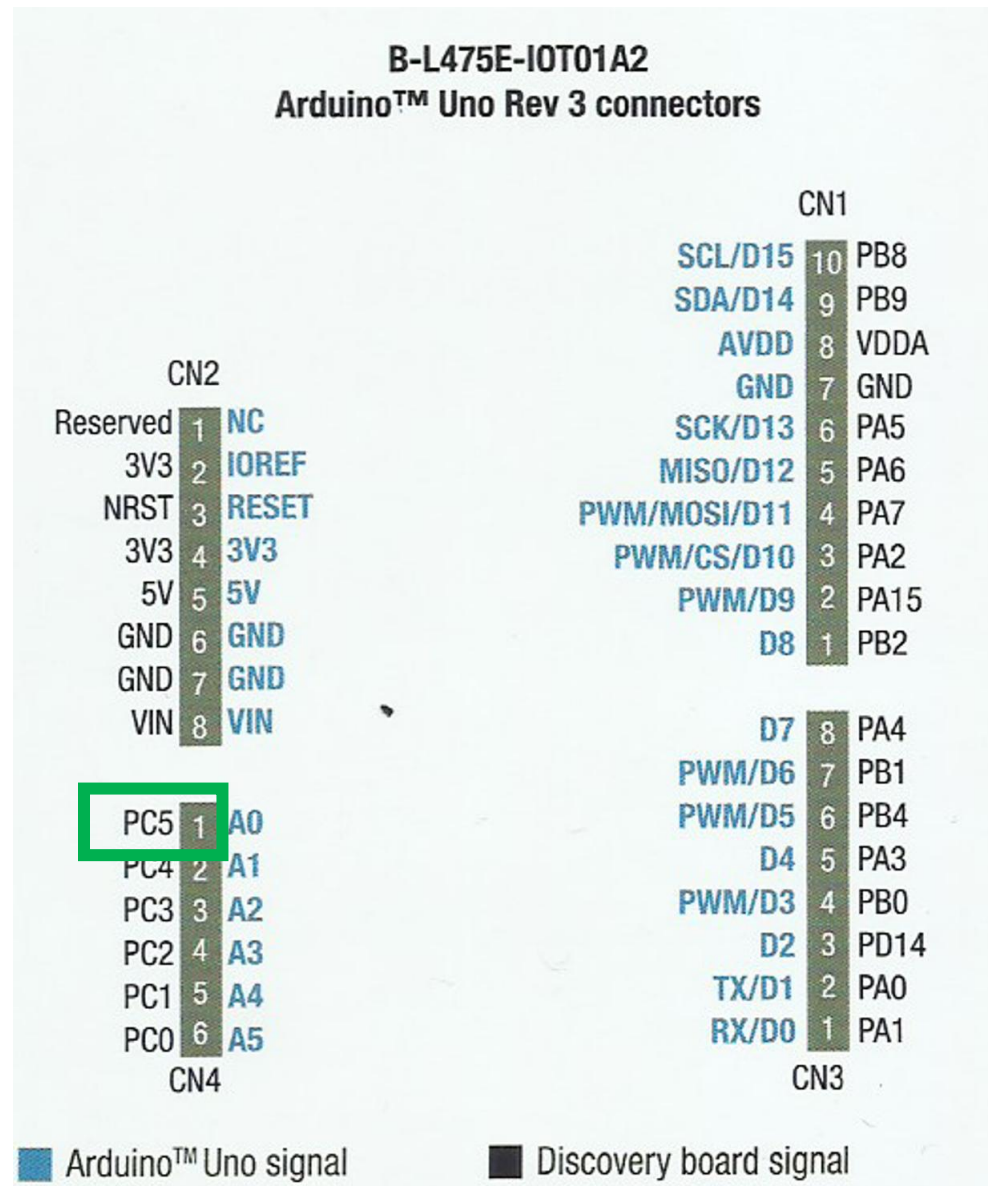
General Purpose I/O Ports

Our interface towards the external world

- › Also supports Arduino Uno R3
- › Let's skip this...

GPIOs are divided into two **board blocks**, and five **SoC ports**

- › CN1,2
- › Port A, B, C, D, E
- › (not all ports are available on the board!!!)





Write on GPIO PINs

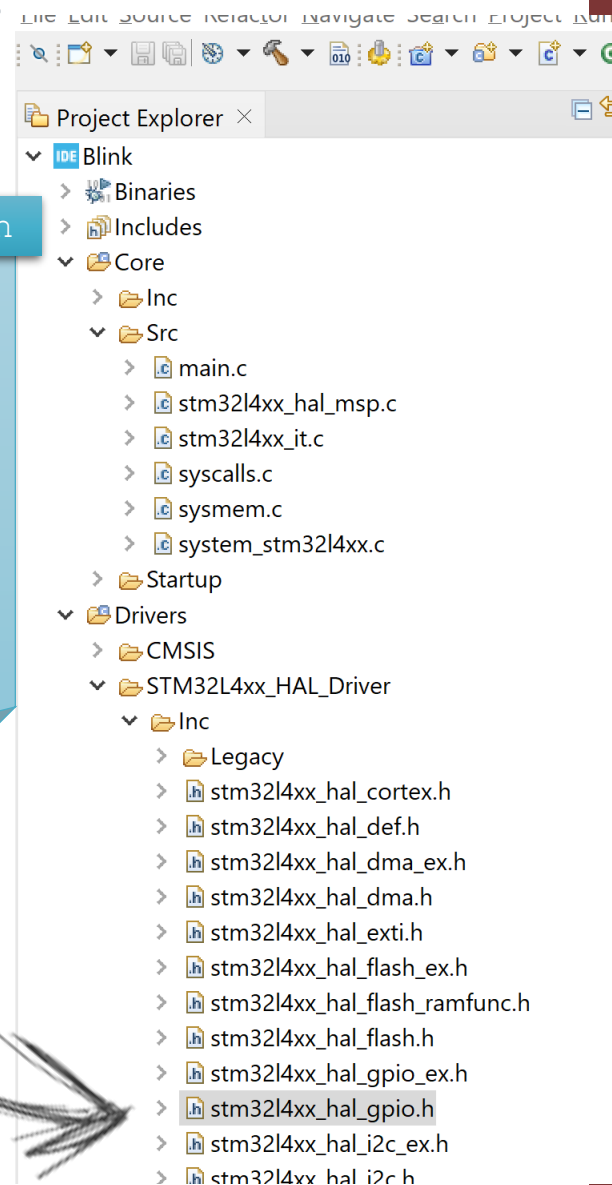
stm32l4xx_hal_gpio.h

```
void HAL_GPIO_TogglePin (GPIO_TypeDef *GPIOx,  
                          int16_t GPIO_Pin );
```

```
GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef *GPIOx,  
                                 int16_t GPIO_Pin );
```

```
void HAL_GPIO_WritePin (GPIO_TypeDef *GPIOx,  
                        int16_t GPIO_Pin,  
                        GPIO_PinState PinState);
```

It's a
generated
file!!





Let's play!

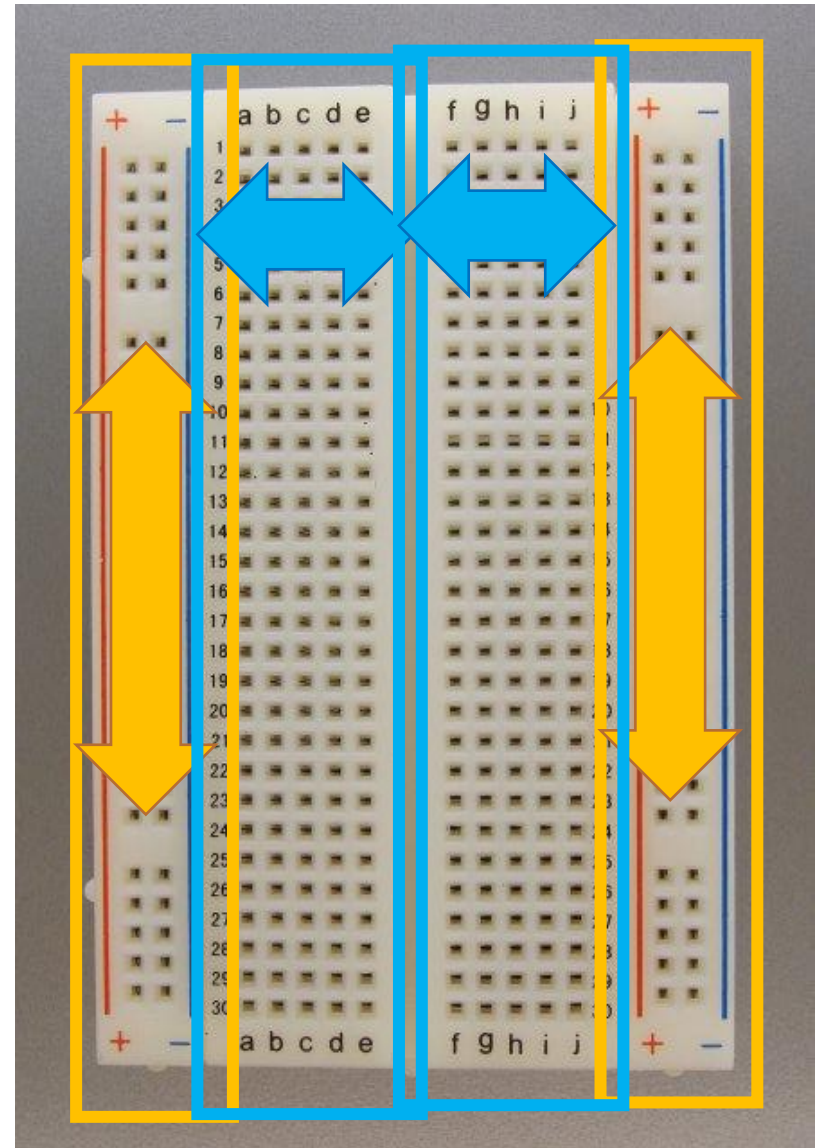




Breadboard

Provides electrical connectivity

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





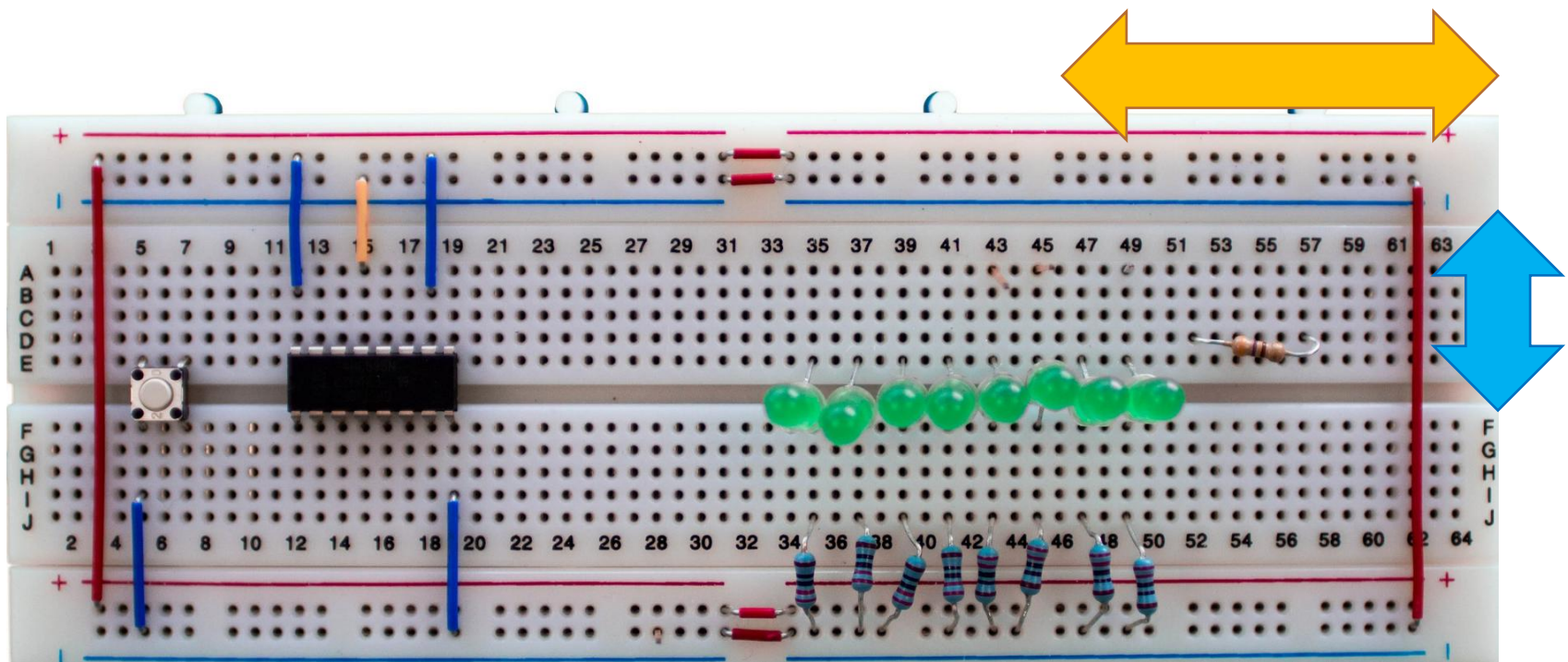
Breadboard

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



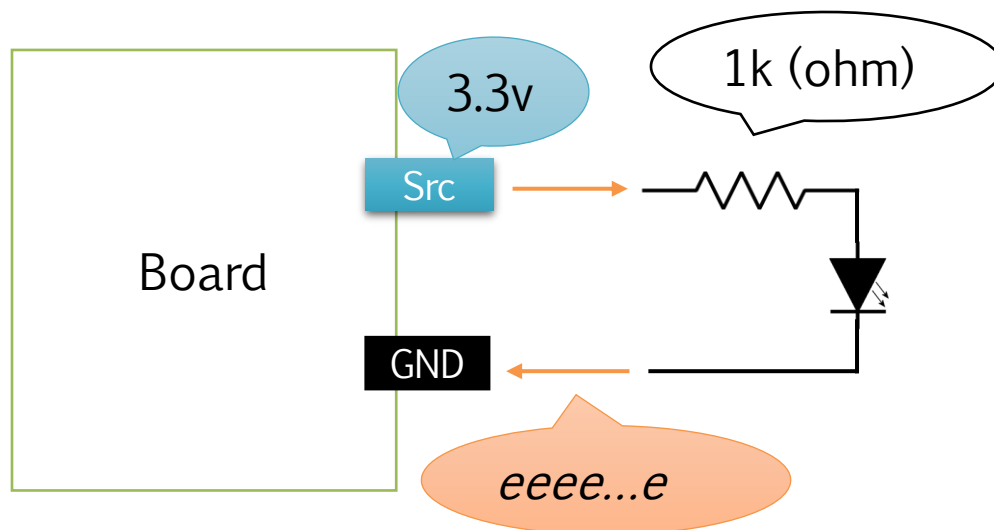


Finally...LEDs

Light Emitting Diodes

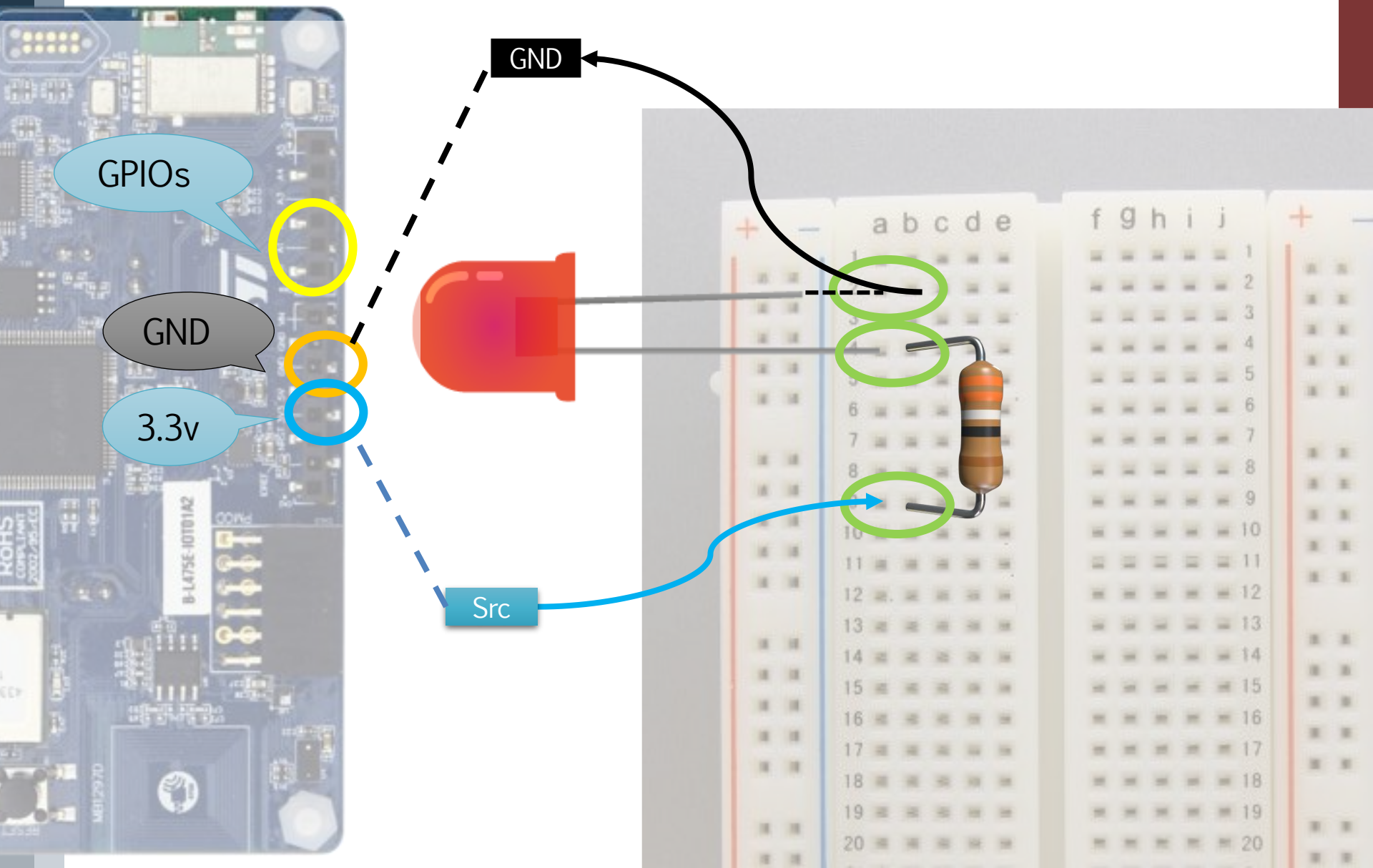
- › You feed with electrons; they light up
- › They have a side!!!!
- › They need a resistance to lower the charge

Wrong wiring => you burn them...





E/E system

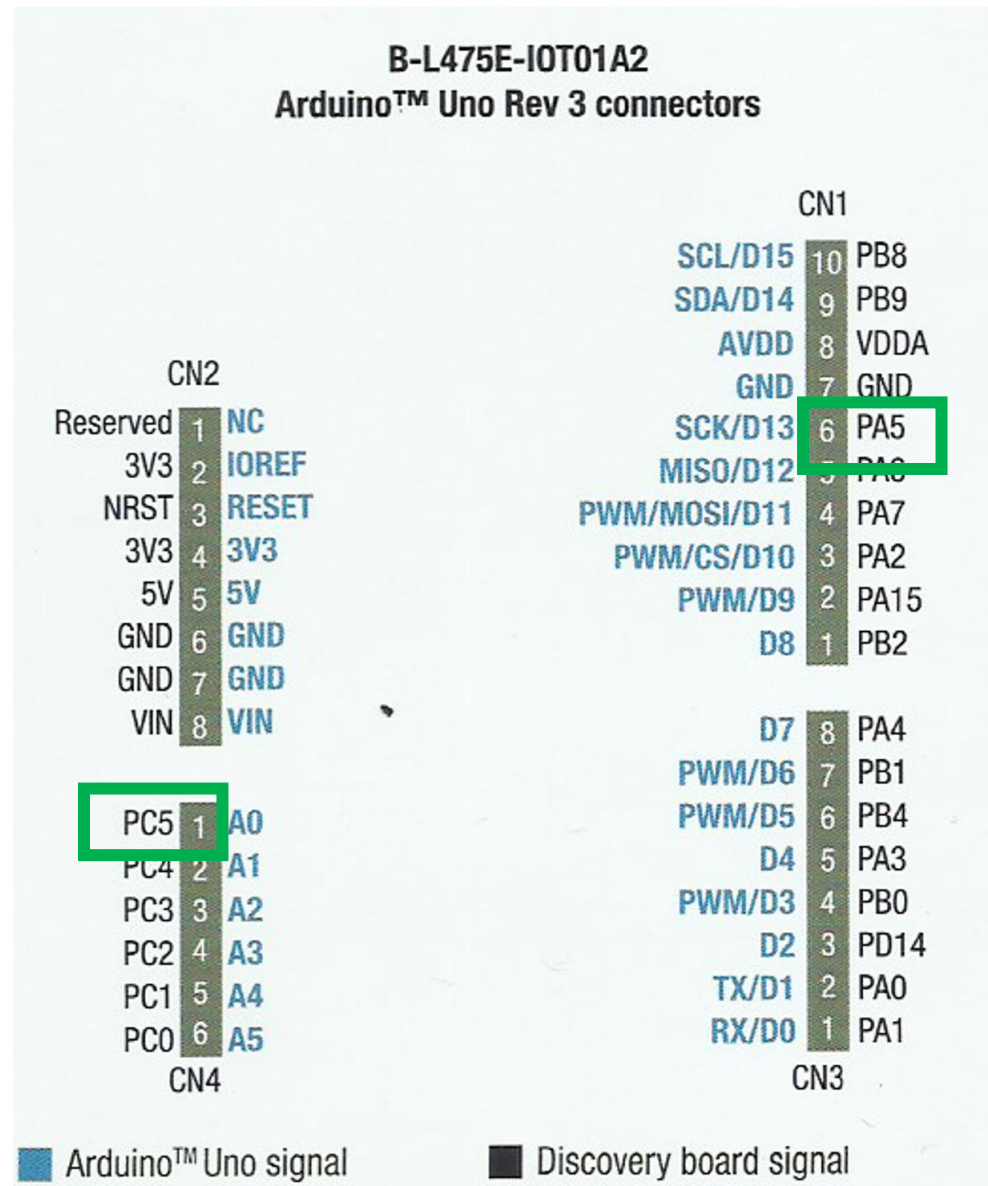




Let's play with Pins

PA5 is also in board pinout

- › Connect our led to them
- › PB14 is not...





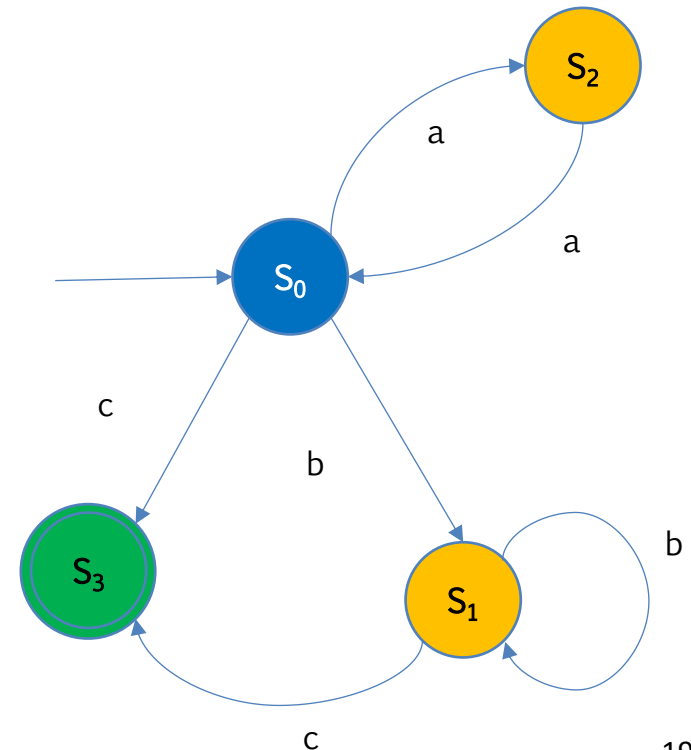
Exercise

Let's
code!

- › Implement the Moore machine of the FSM that understands whether a words 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





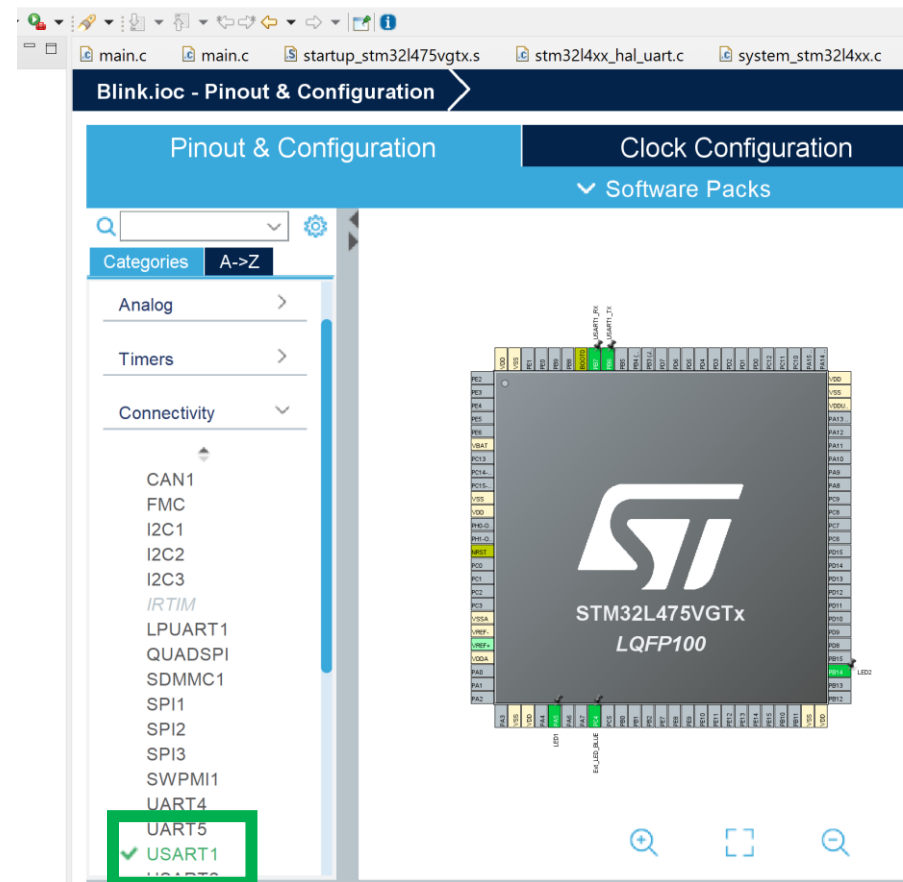
Serial communication

Universal asynchronous receiver-transmitter – UART

- › “Asynchronous” -> One channel for TX, one channel for RX

USART Universal Synchronous/Asynchronous Receiver/Transmitter

- › w/bitstream
- › USART1 in our board





Configure USART1

- › PB6 for Tx
- › PB7 for Rx

Pinout & Configuration

Clock Configuration

Software Packs

USART1 M

Categories A->Z

Analog >

Timers >

Connectivity >

CAN1

FMC

I2C1

I2C2

I2C3

IRTIM

LPUART1

QUADSPI

SDMMC1

SPI1

SPI2

SPI3

SWPMI1

UART4

UART5

✔ USART1

USART2

USART3

USB_OTG_FS

Mode Asynchronous

Hardware Flow Control (RS232) Disable

☐ Hardware Flow Control (RS485)

Reset Configuration

✔ Parameter Settings

✔ User Constants

✔

Search Signals

Search (Ctrl+F)

Pin Name	Signal on Pin	GPIO out..
PB6	USART1_TX	n/a
PB7	USART1_RX	n/a



Write code

```
/* USER CODE BEGIN 3 */

uint8_t Test[] = "Hello World !!!\r\n"; //Data to send
HAL_StatusTypeDef retval =
    HAL_UART_Transmit(&huart1,Test,sizeof(Test),10);

//Let's add some delay
HAL_Delay(1000);

/* USER CODE END 3 */
```

- › Copy-paste this in your main loop



System header

```
/**
 * @brief Send an amount of data in blocking mode.
 * [...]
 * @param huart UART handle.
 * @param pData Pointer to data buffer (u8 or u16 data elements).
 * @param Size Amount of data elements (u8 or u16) to be sent.
 * @param Timeout Timeout duration.
 * @retval HAL status
 */
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart,
const uint8_t *pData, uint16_t Size, uint32_t Timeout);
```

› Returns "check"



On your machine... (1)

First, test with a “standard” serial Monitor

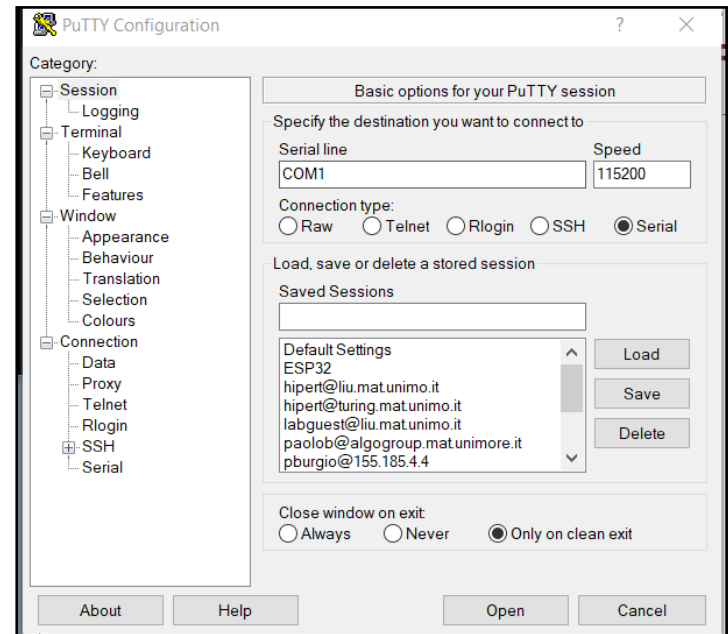
› Linux

- `sudo apt install minicom`
- Serial/USB ports are typically `/dev/ttySOMETHING`

› Windows

- Putty
- Serial/USB ports are COMx

115200 Baud, no parity, 8 bit





On your machine... (2)

Programmatically read from serial/USB

- › C++
 - <https://github.com/imabot2/serialib>
- › Python
 - pySerial



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

- › A "small blog -> <http://www.google.com>
- › Serial comms
 - https://wiki.st.com/stm32mcu/wiki/STM32StepByStep:Step3_Introduction_to_the_UART
 - <https://github.com/imabot2/serialib>