# STM hands-on session

Paolo Burgio
paolo.burgio@unimore.it

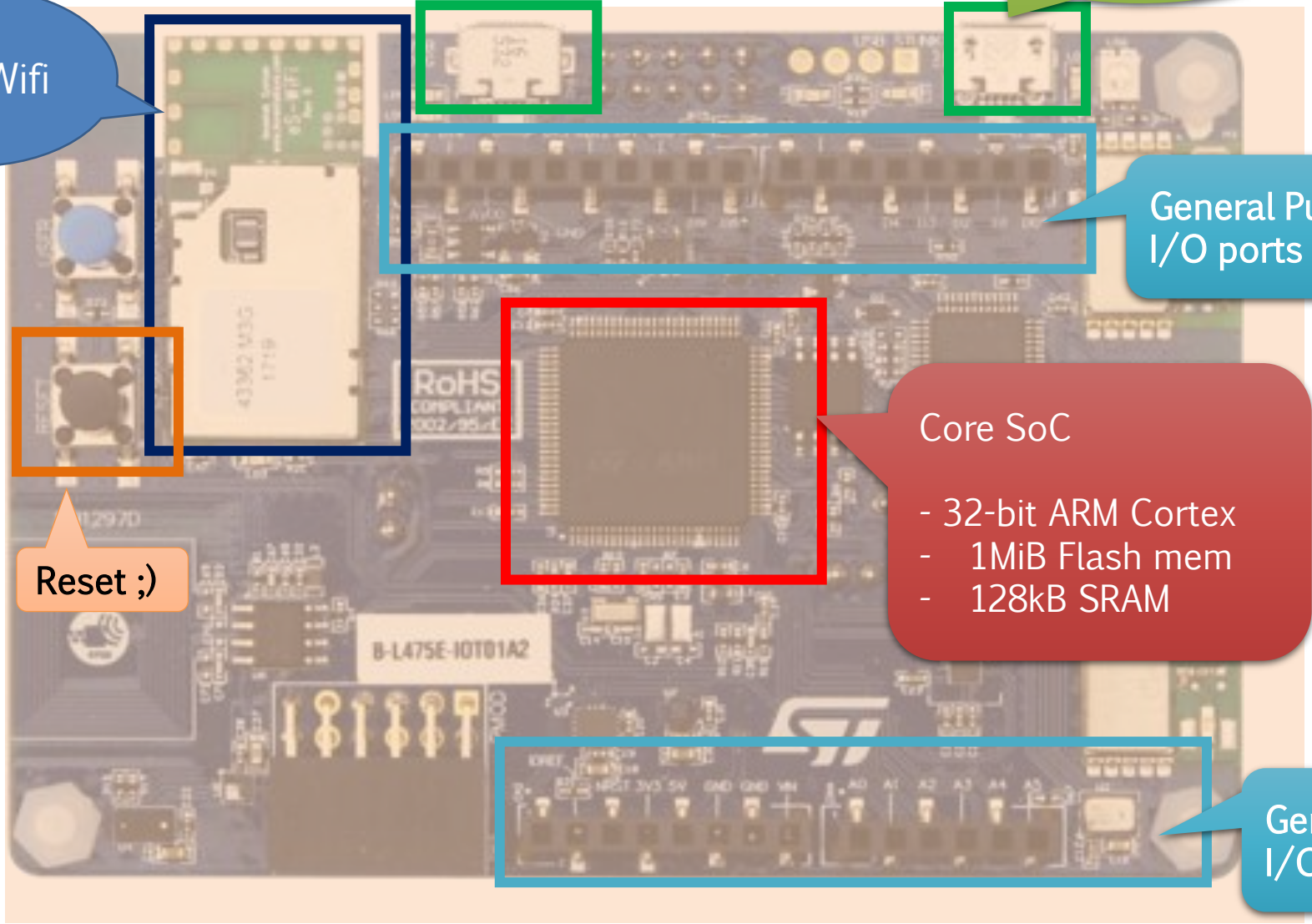

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

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)



# Software

Micro-kernel

- No OS, need to flash al memory regions

ST proprietary

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

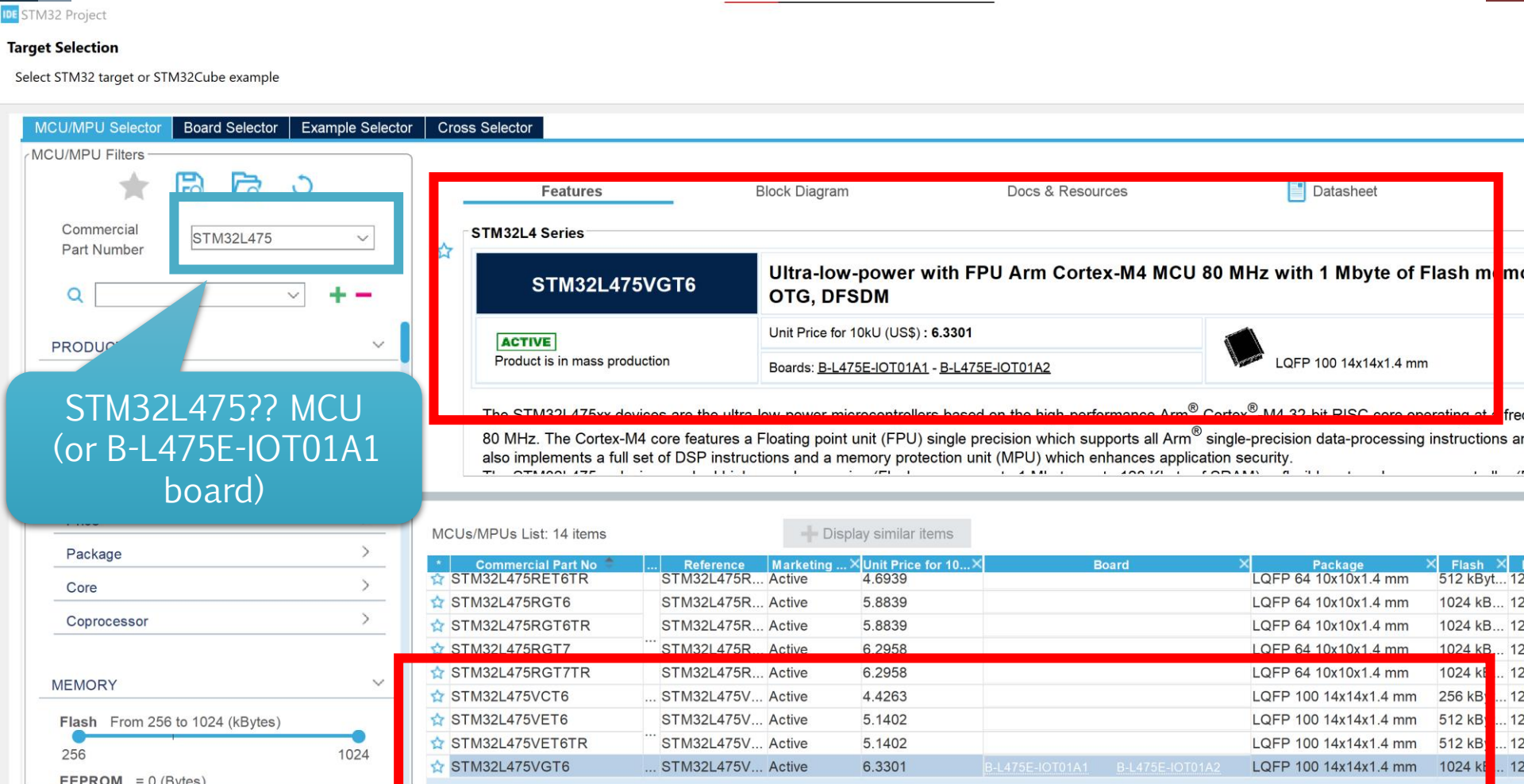
How to work

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

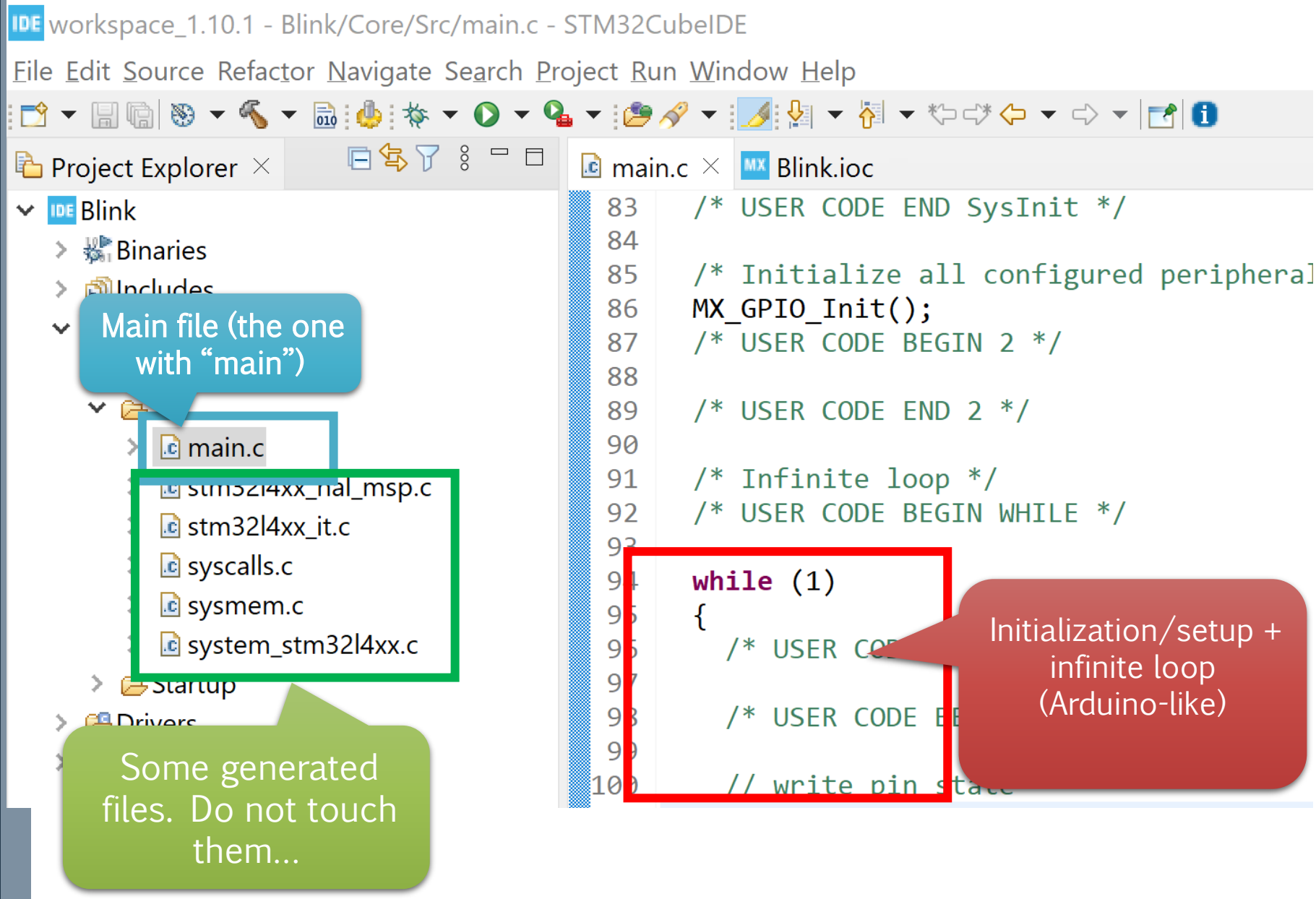
# A simple application

# 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..)



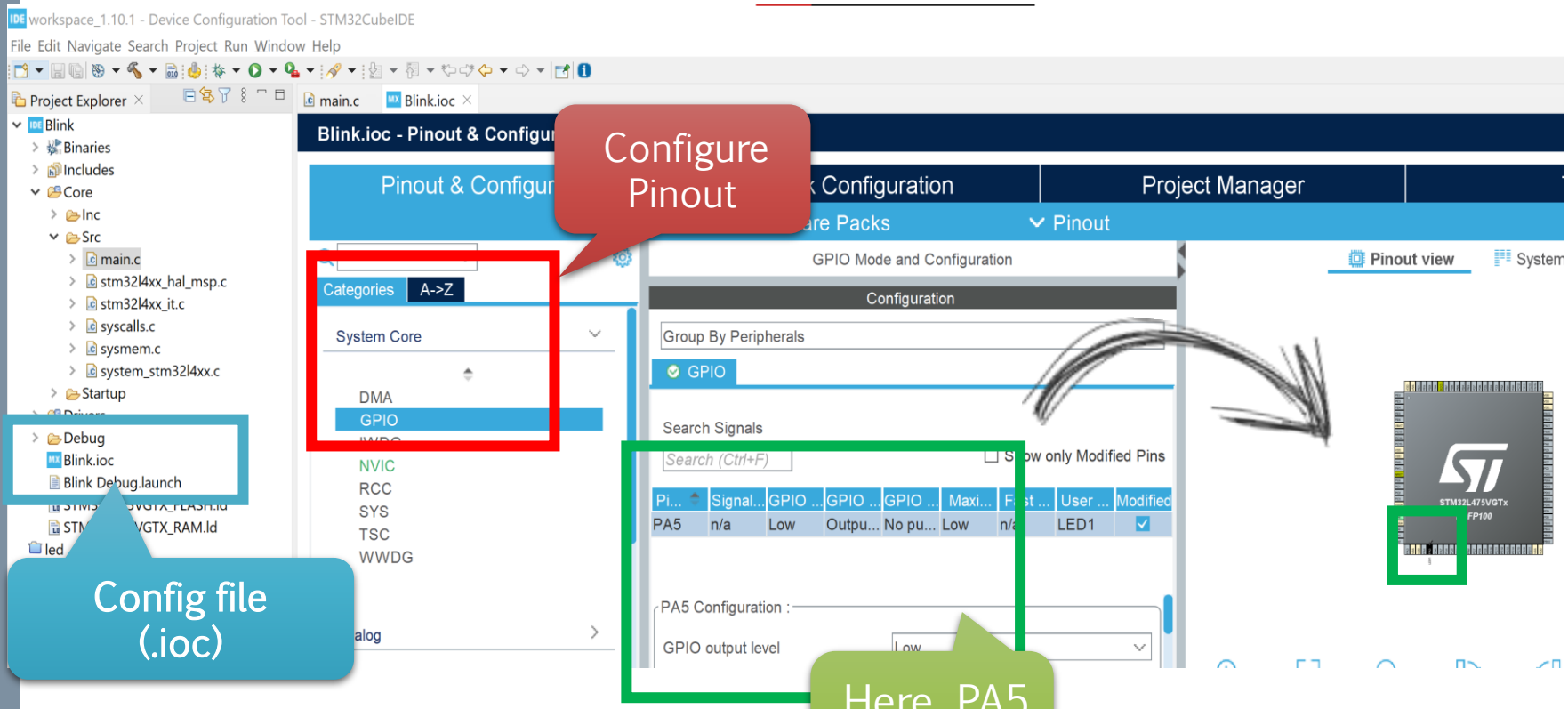
# IDE


workspace_1.10.1 - Blink/Core/Src/main.c - STM32CubeIDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Blink
Binaries
Includes
main.c
stm32l4xx_hal_msp.c
stm32l4xx_it.c
syscalls.c
sysmem.c
system_stm32l4xx.c
Startup
Drivers
main.c
Blink.ioc
/* USER CODE END SysInit */
/* Initialize all configured peripheral
MX_GPIO_Init();
/* USER CODE BEGIN 2 */
/* USER CODE END 2 */
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
/* USER CO
/* USER CODE E
// write pin
Main file (the one with “main”)
Some generated files. Do not touch them...
Initialization/setup + infinite loop (Arduino-like)

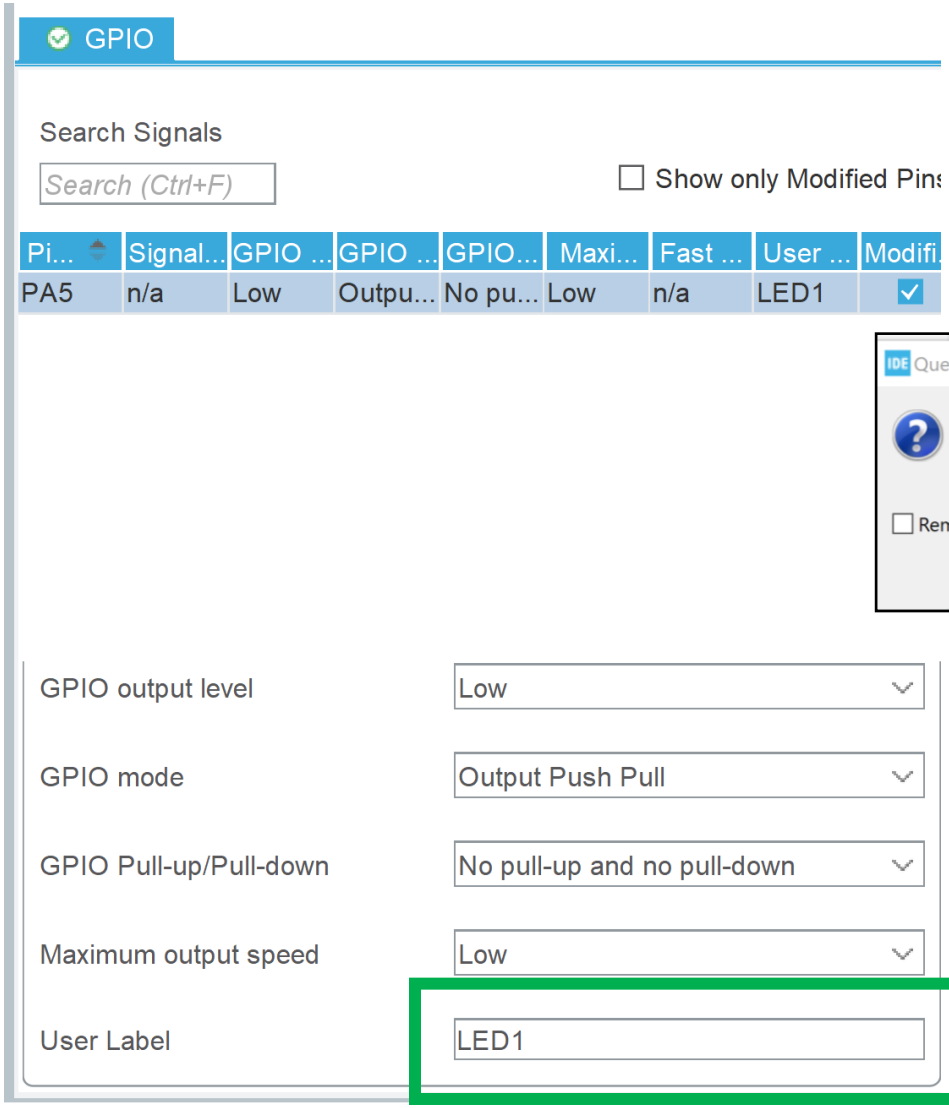
# Configure LEDs

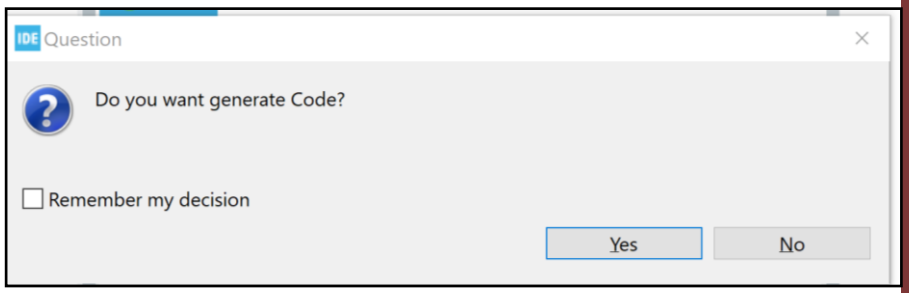
We want to create **an alias** for GPIOs

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



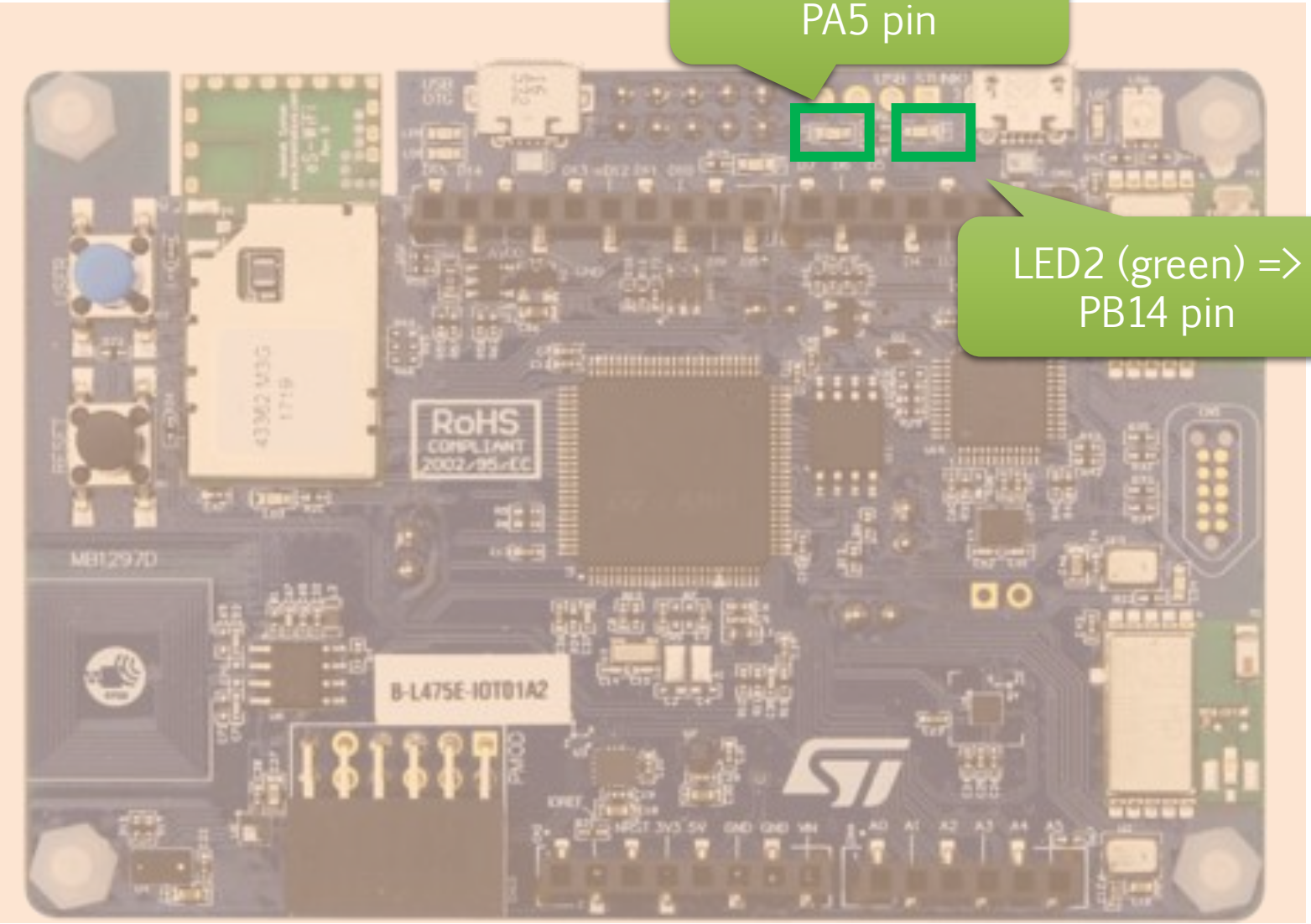
# Let's configure PA5




That's our Alias!!

# Leds and GPIOs



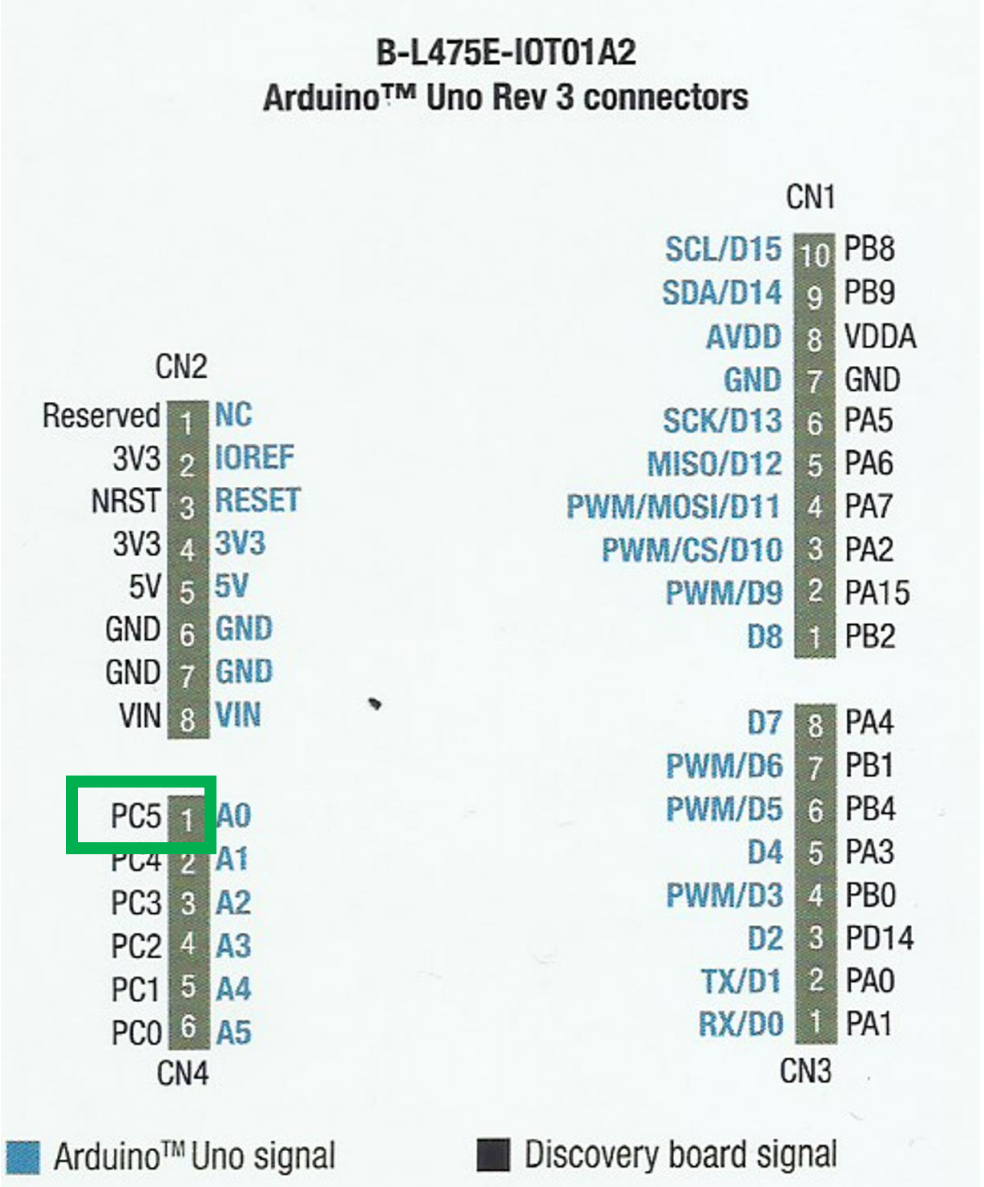
# General Purpose I/O Ports

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!!!)



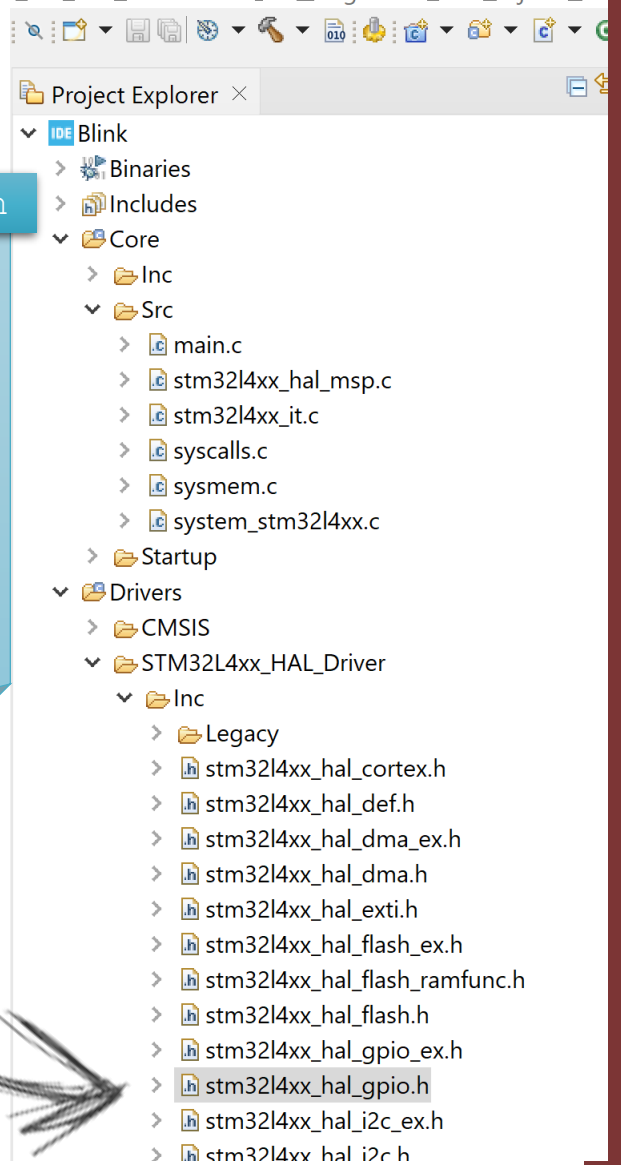
# Write on GPIO PINs

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);
```



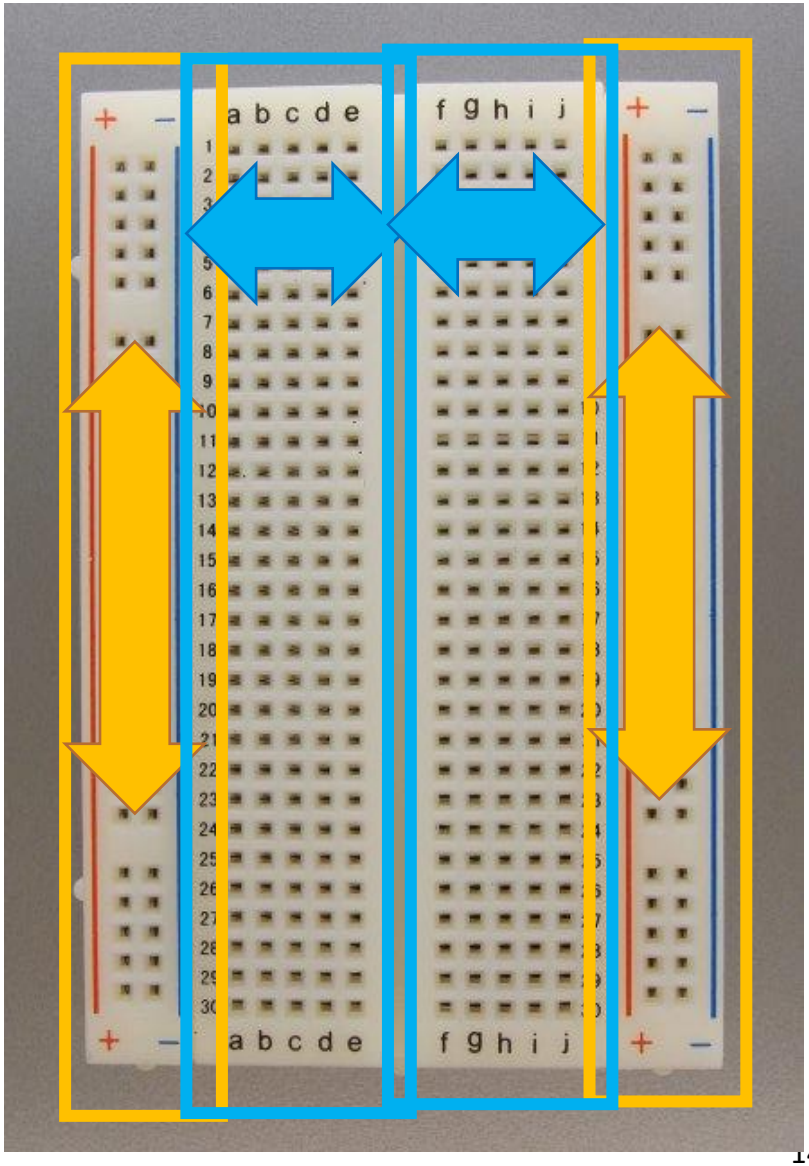

It's a generated file!!

Let's play!

# Breadboard

Provides electrical connectivity

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



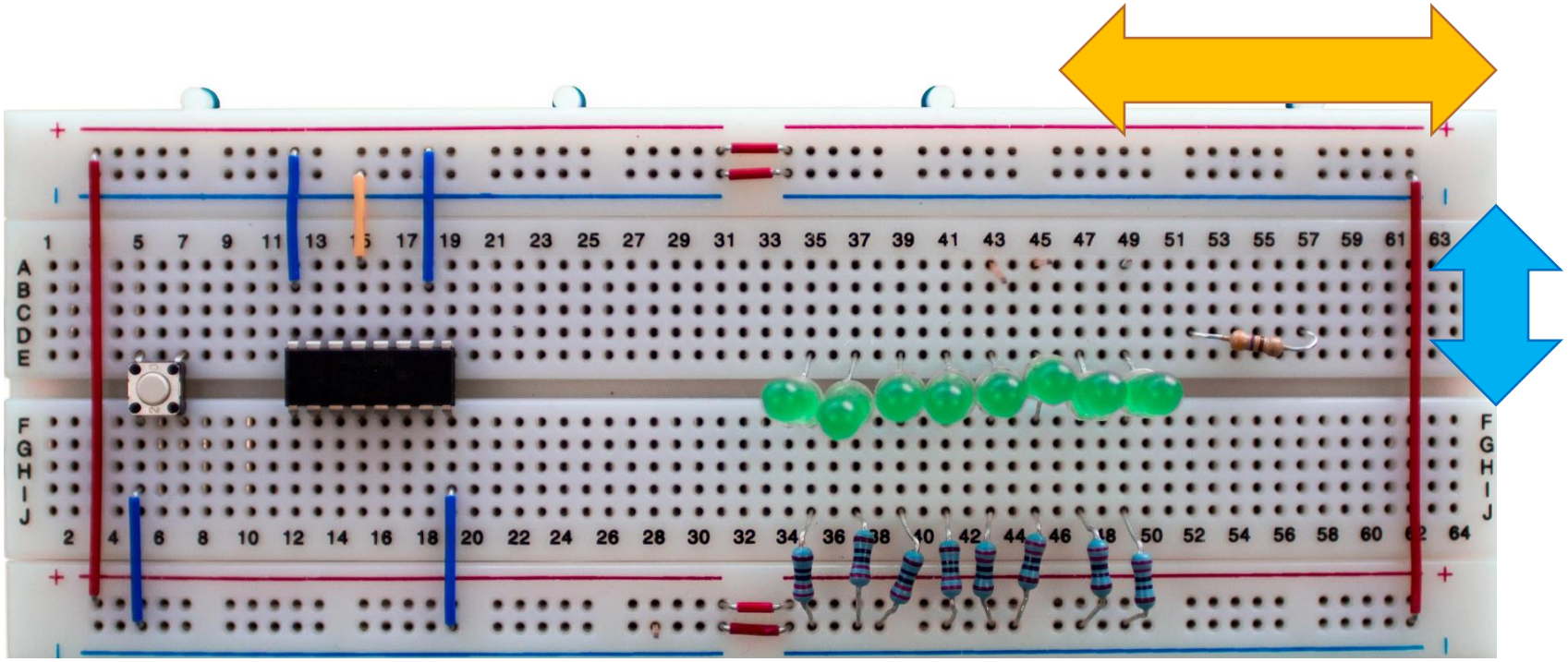
# Breadboard

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

› Typically, used for power/GND

Brought to the internal rails with jumper wires

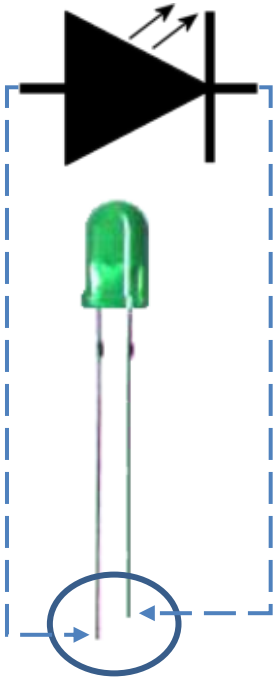
› Where core/chip and other stuff reside



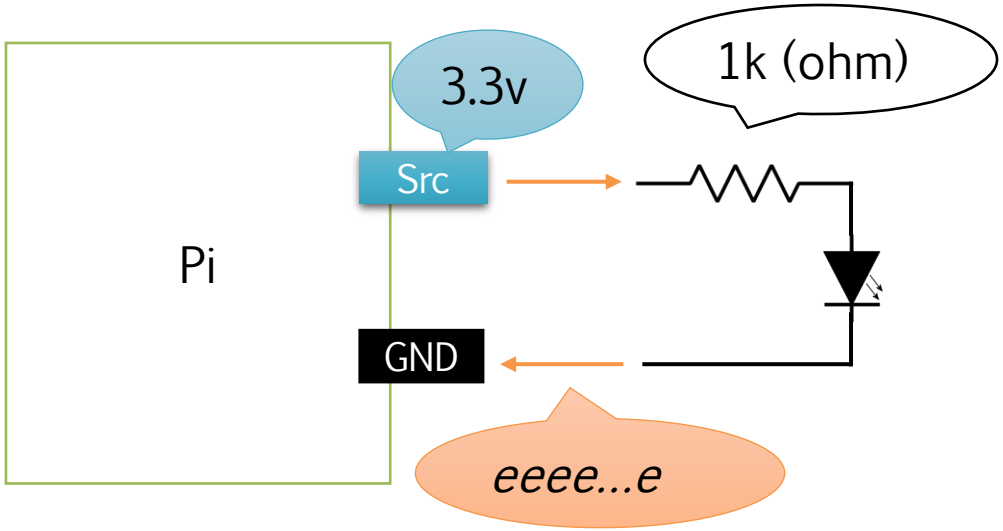
# Finally…LEDs

Light Emitting Diodes

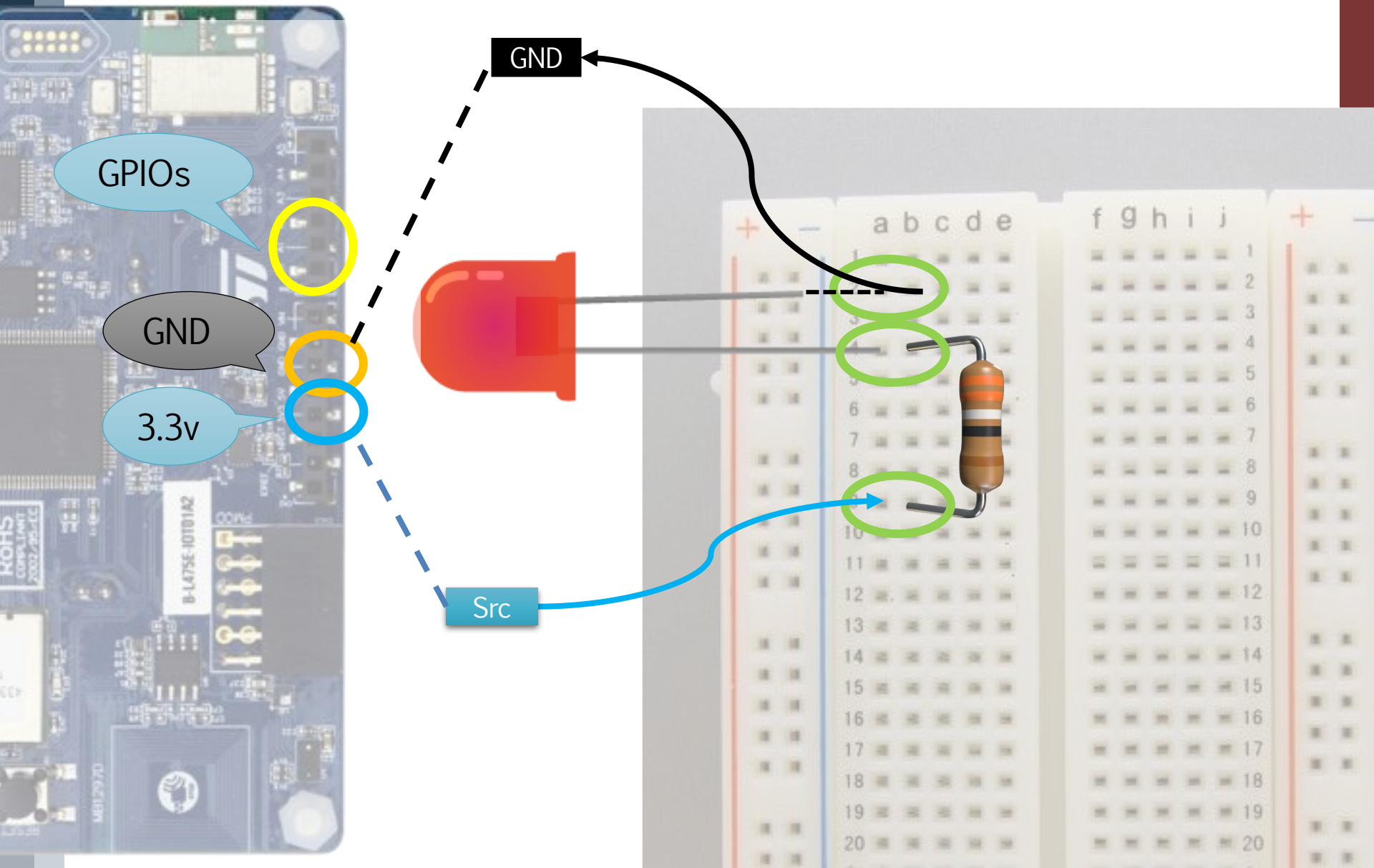
- You feed with electrons; they light up
- They have a side!!!!
- They need a resistance to lower the charge



Wrong wiring => you burn them…



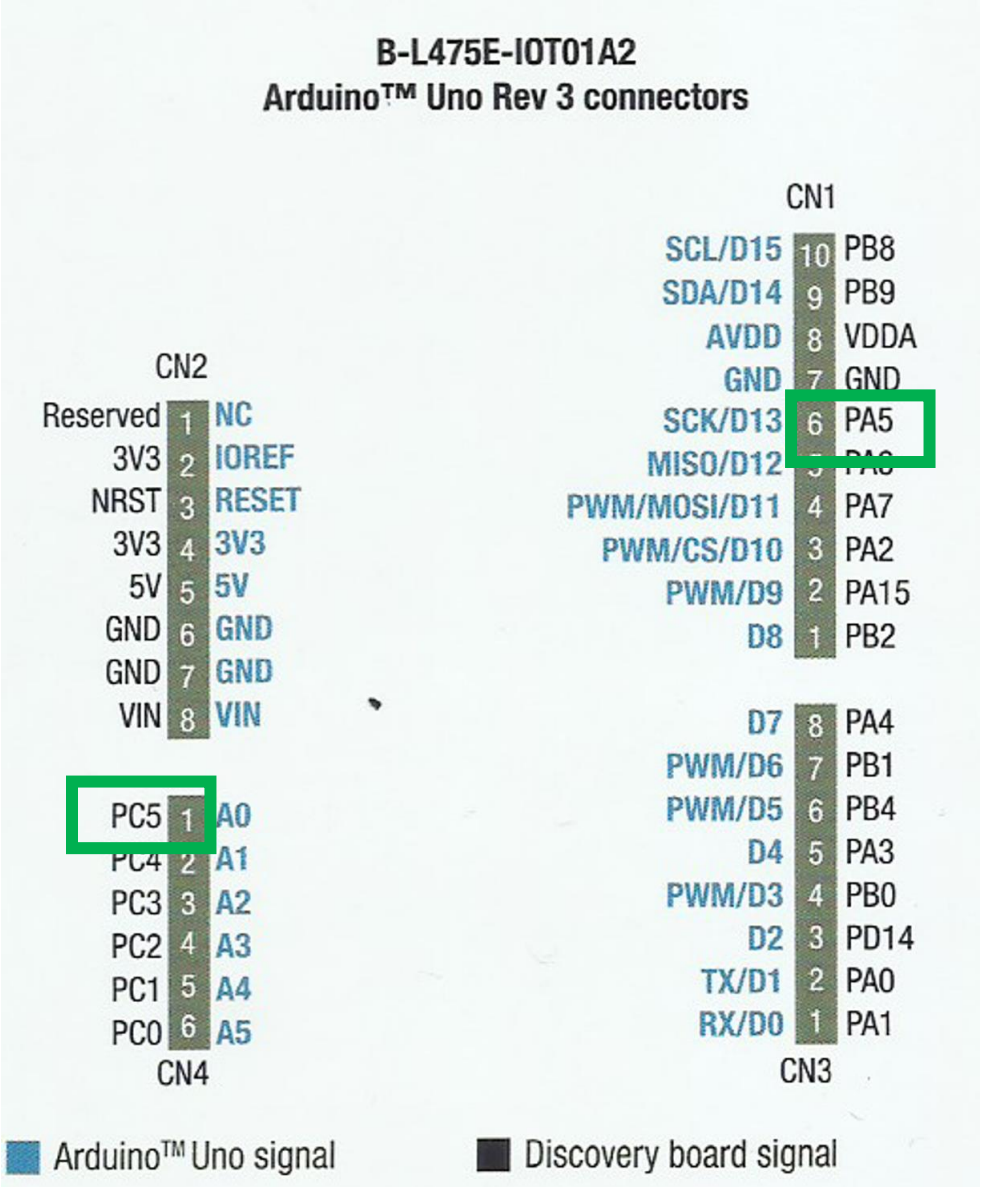
# E/E system



# Let's play with Pins

PA5 is also in board pinout

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



# Exercise

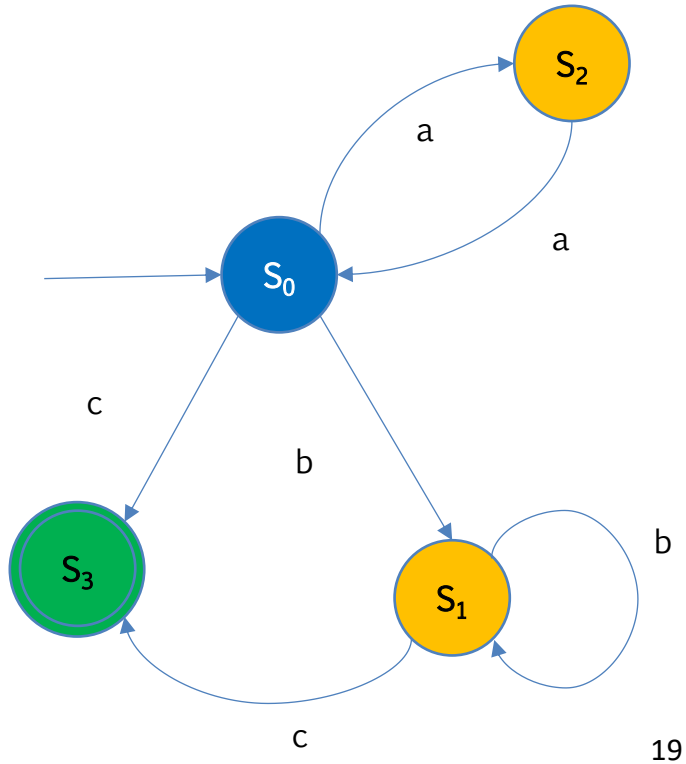
Let's code!

› 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

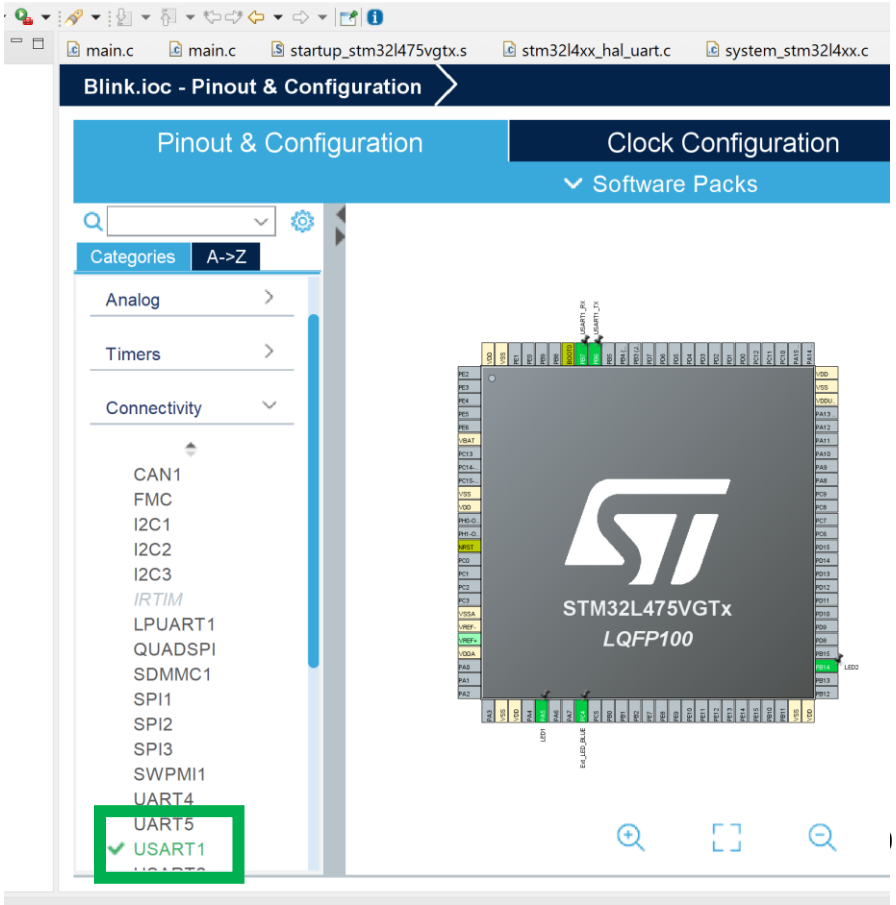
Universal asynchronous receiver-transmitter – UART

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

USART Universal Synchronous/Asynchronous Receiver/Transmitter

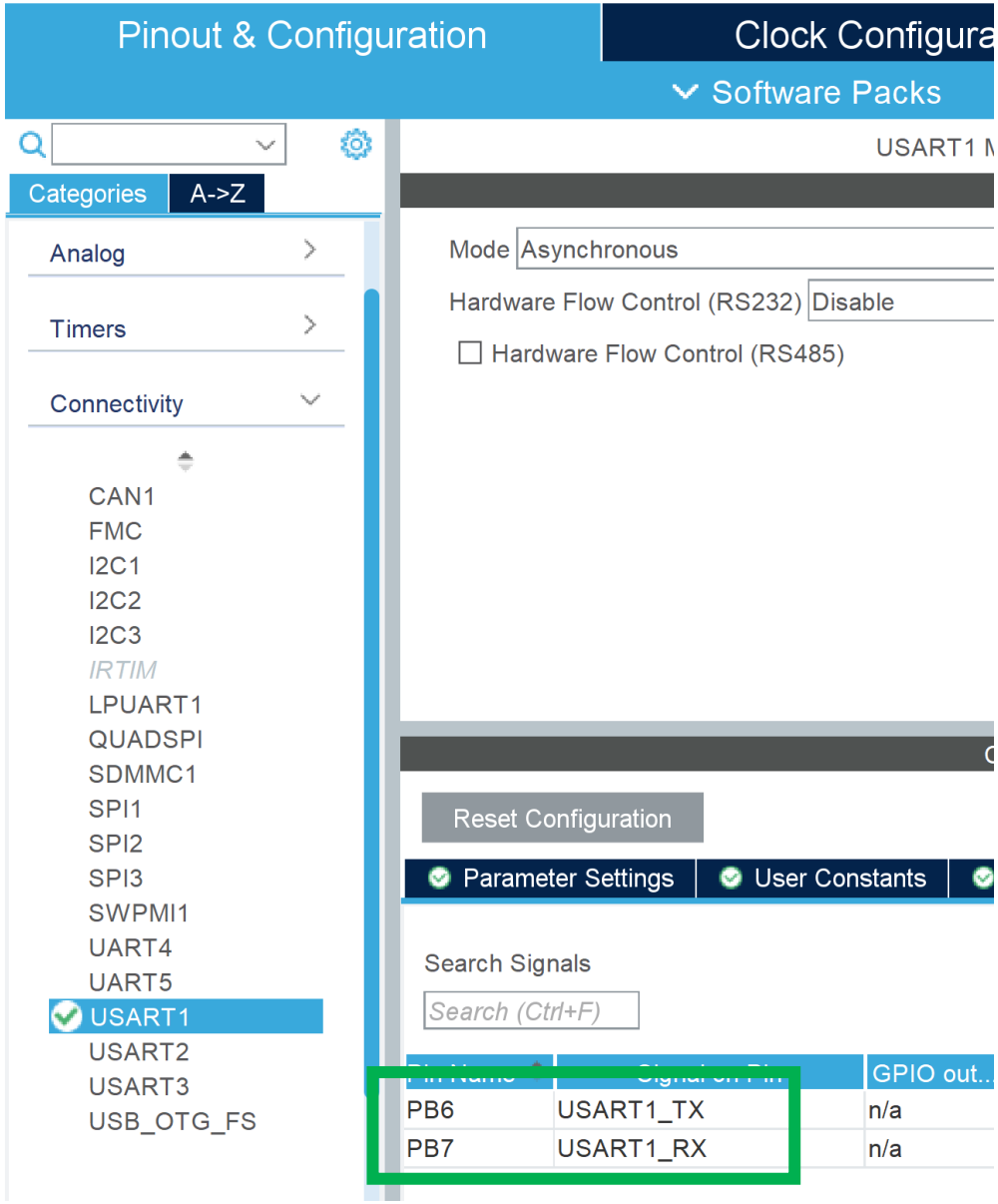
› w/bitstream

› USART1 in our board



# Configure USART1

- PB6 for Tx
- PB7 for Rx



# 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);
```

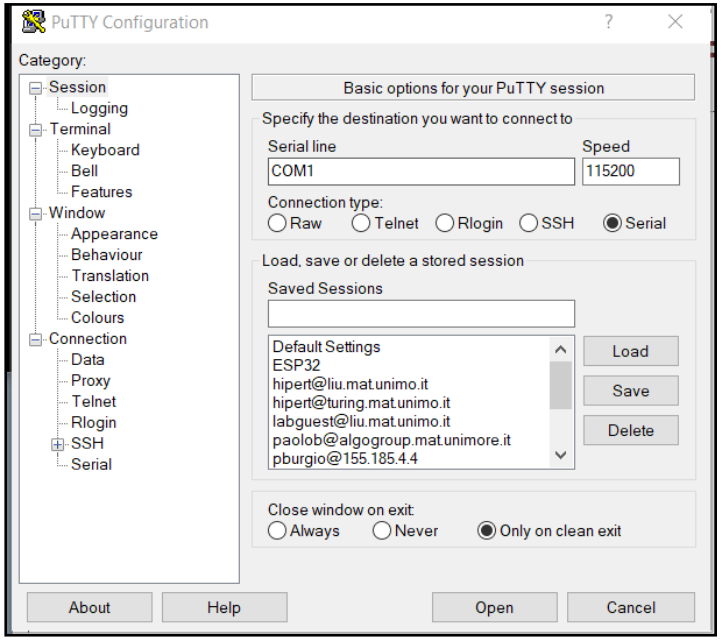
› Returns “check”

# 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