

# Microcontroller programming

## 02 – The basic toolset



# Reading tips



- The keyboard symbol means it's practice time
- Colored text usually carries a meaning :
  - **Blue text** highlights a new concept or an important lesson to take away
  - **Red text** highlights a tricky part or a risk of mistake
  - **Green text** highlights good practices and stuff that is now done
- At the end of each slide deck, you'll find a recap of technologies and concepts



# Agenda

- What's in STM32 Cube ?
- Blinking LED
- Hardware corner: GPIO
- The code behind a blinking LED
- Hardware corner: interrupt
- Configuring pins with STM32 CubeMX
- Software corner: C weak symbol



# What's in STM32 Cube ?

Butterfly sold separately



# What's in STM32 Cube ?



- What is a **development environment for embedded** ?
- Slightly different than programming for a PC !
- Our needs:
  - Writing code
  - Compiling code to a program
  - Storing program on the target hardware
  - Debugging program from the target hardware



# What's in STM32 Cube ?

- Our needs:
  - Writing code → Text editor on console / GUI
  - Compiling code to a program → **Cross-compiler**
  - Storing program on the target hardware → **Flasher** (for “flashing”)
  - Debugging program from the target hardware → **Cross-debugger**
- Some new terms for you !



# What's in STM32 Cube ?

- The **cross-compiler** runs on hardware architecture A, and produces code for hardware architecture B
- A → host
- B → target
  - x86-64 → ARM
  - PowerPC → aarch64 [64-bits ARM]
  - ...
- “gcc” is actually a **native compiler** (A == B)



# What's in STM32 Cube ?

- Native compilers are not practical for MCUs
- Compiling is very CPU intensive
  - You would wait forever
- Compilers are big
  - You can't even store them on the board !
  - On Alpine Linux gcc binary is 1.7MB large (without dependencies)
  - STM32F429ZIT internal flash is 2MB large





# What's in STM32 Cube ?

- Compiler examples !
  - **gcc**
    - Implicitly native for your arch (e.g. x86-64 → x86-64)
  - **arm-none-eabi-gcc**
    - Host : your arch
    - Libc: none
    - ABI : Extended ABI
    - Target: ARM (32-bits)
- (The compiler's full name is called a **tuple**)



# What's in STM32 Cube ?

- **mips32-musl-eabi-gcc**
  - Host : your arch
  - Libc: musb libc ([official website](#))
  - ABI : Extended ABI
  - Target: ARM (32-bits)
- **riscv64-glibc-gnueabi-gcc**
  - Host: your arch
  - Libc: GNU C library (the most common one)
  - ABI: GNU Extended ABI
  - Target: Risc-V 64-bits



# What's in STM32 Cube ?

- This logic works with gcc but also clang, Keil, Microchip etc.
- You may be asking yourself:
  - *Why are there multiple C libraries ?*
  - *What is an ABI ?*
  - *How to choose a cross-compiler ?*
- We'll answer those questions later ;-)



# What's in STM32 Cube ?

- The **simplest toolset for the ARM course** could be:
  - Writing → vim
  - Compiling → arm-none-eabi-gcc
  - Flashing → openocd
  - Debugging → openocd
- All are command-line tools
- All are open-source and require no license



# What's in STM32 Cube ?

- **For ARM course let's use the beginner-friendly STmicro IDE**
- The IDE allows not thinking about :
  - Where to get libraries, and which ones ? (they require configuration)
  - How to handle board-specific code ? (it would requires directly diving into the SMT32F429 reference manual)
  - How to boot my MCU ? (specific initialization code would again require precautious study of the STM32F429)
- *Let's launch STM32 Cube IDE !*



# What's in STM32 Cube ?

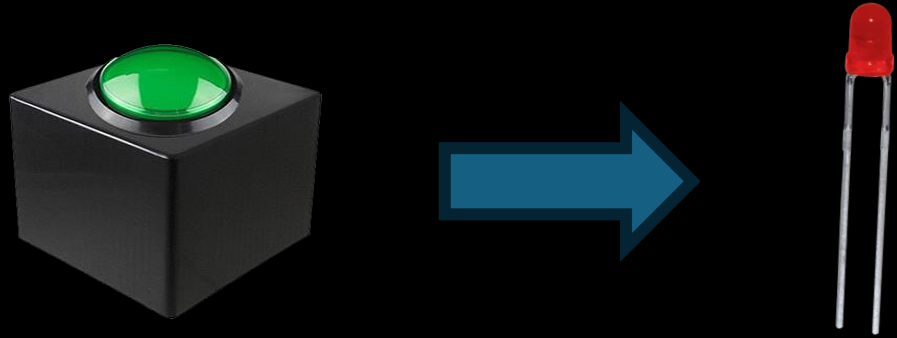
- STM32 Cube IDE is an Eclipse-based IDE, and offers :
  - Text editor
  - C/C++ Compiler
  - Flasher
  - Debugger
  - Board-specific graphical configurator
  - Board-specific code generator
  - Hardware Abstraction Layer (think “library” for now)
  - Example programs



# What's in STM32 Cube ?

- Today we'll be doing the "Hello World" of embedded systems...switching a LED when you press a button
- This simple application will require understanding :
  - How STM32 Cube generates code for us
  - What is the generated code's structure
  - Figuring out why the main loop looks weird





# Blinking LED

Gotta start somewhere





# Blinking LED

- **Please open the GPIO\_EXTI project**
  - You should already have downloaded it before the course
  - Open *gistre26-arm-setup.pdf* on Moodle if not !
- Please compile and run the program on the devkit
- Let's take a moment to make sure everyone is up and running
- The following slides offer tips for debugging



STM32 Project

Target Selection

Select STM32 target or STM32Cube example

MCU/MPU Selector

Board Selector

Example Selector

Cross Selector

Example Filters

★

📁

🔍

🔄

Name

🔍

+

-

Keyword

>

Vendor

>

Board

>

Name

Type

>

MCU / MPU

>

Name

STM32F429ZIT6

Series

>

Project

☐ Device Configuration Tool capable

Projects/STM32F429ZI-Nucleo/Examples/GPIO/GPIO\_IOToggle/

STM32F4

GPIO\_IOToggle

V1.28.0

Required Software Package

STM32Cube\_FW\_F4\_V1.28.0 (size: 611.0 MB)

Vendor

STMicroelectronics

Board

NUCLEO-F429ZI

Mounted device

STM32F429ZIT6

Supported Toolchain/IDE

EWARM, MDK-ARM, STM32CubeIDE

STM32CubeIDE Minimum Compatible Version

NA

Examples List: 111 items

Export

Name	Board	Board Type	Configurable	STM32CubeIDE Version	SW Package Installed
FreeRTOS_MPU	STM32F429I-DISC1	Discovery Kit	NA	NA	
FreeRTOS_ThreadCreation	STM32F429I-DISC1	Discovery Kit	NA	NA	
FWUpgrade_Standalone	STM32F429I-DISC1	Discovery Kit	NA	NA	
Fx_SRAM_File_Edit_Standalone	NUCLEO-F429ZI	Nucleo-144	NA	NA	
GPIO_EXTI	STM32F429I-DISC1	Discovery Kit	NA	NA	
GPIO_EXTI	NUCLEO-F429ZI	Nucleo-144	NA	NA	
GPIO_IOToggle	NUCLEO-F429ZI	Nucleo-144	NA	NA	
HAL_TimeBase_RTC_ALARM	STM32F429I-DISC1	Discovery Kit	NA	NA	
HAL_TimeBase_RTC_ALARM	NUCLEO-F429ZI	Nucleo-144	NA	NA	

< Back

Next >

Finish

Cancel

Windows

Taper ici pour rechercher

IDE

13°C Nuageux

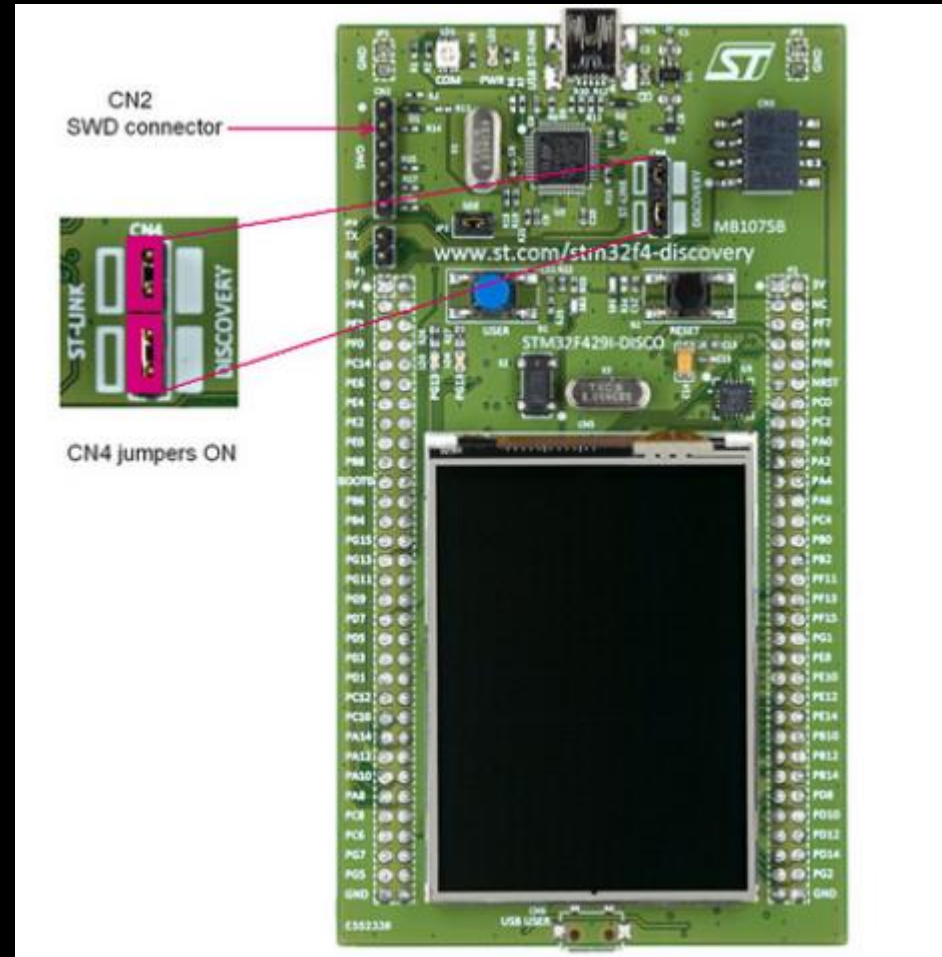
20:08 27/04/2024

The STM32 Cube IDE example code selector

18

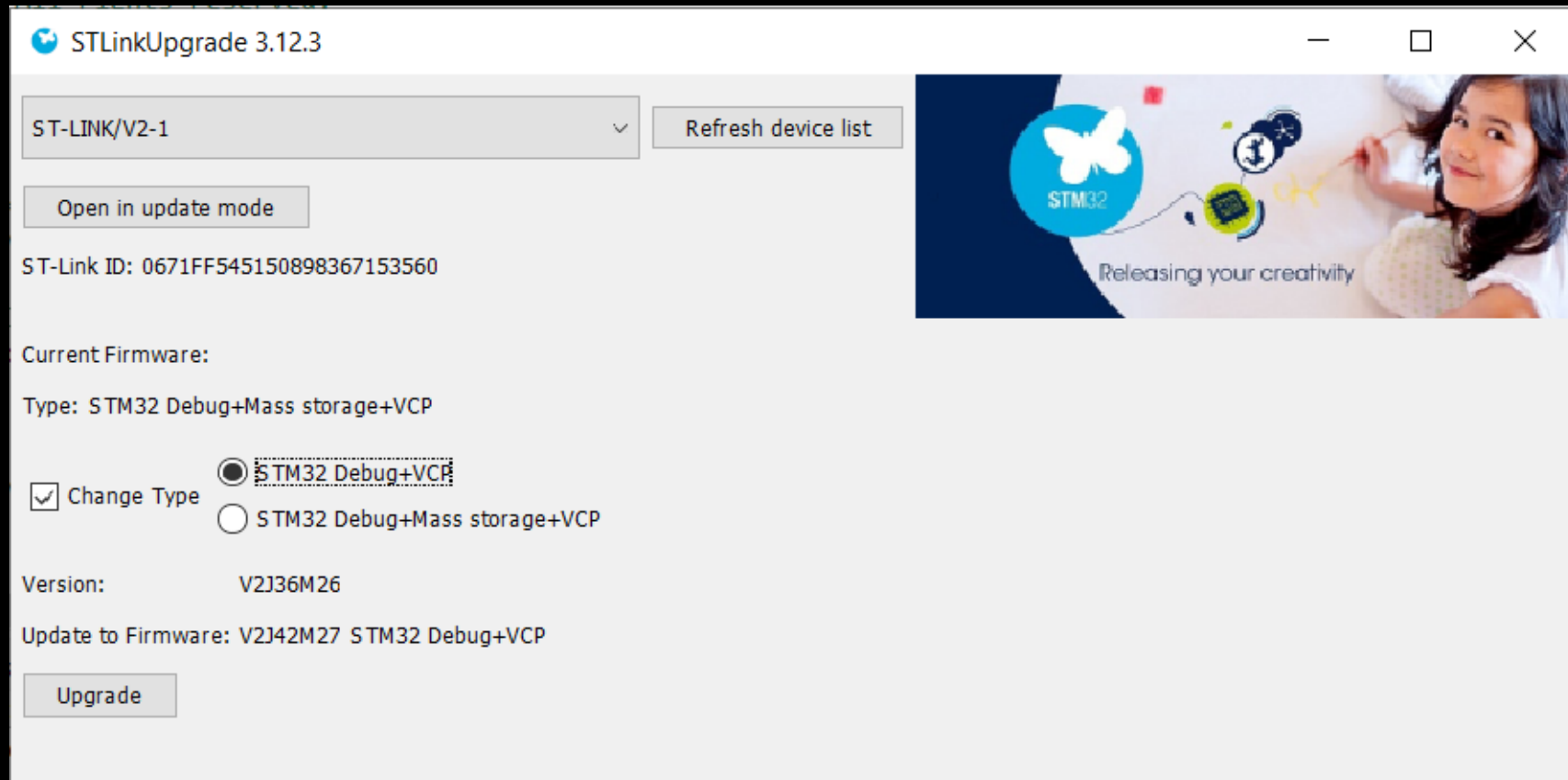
# Blinking LED

Make sure you've put on jumpers on *JP3* (1 jumper) and *CN4* (2 jumpers) in order to be able to connect to the board using USB



# Blinking LED

- Make sure to upgrade the ST link firmware if asked
- The prompt looks like this :





# Hardware corner: GPIO

General Purpose Input/Output



# Hardware corner: GPIO

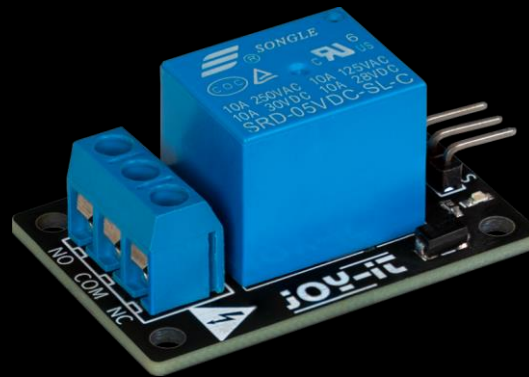
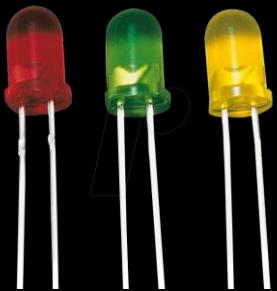
## New Hardware : GPIO

- **GPIOs are digital, 1-wire signals**
- 1 GPIO means 1 pin
- Can generally be in one of 3 **states**:
  - Input (reading)
  - Output (writing)
  - Floating (detached)



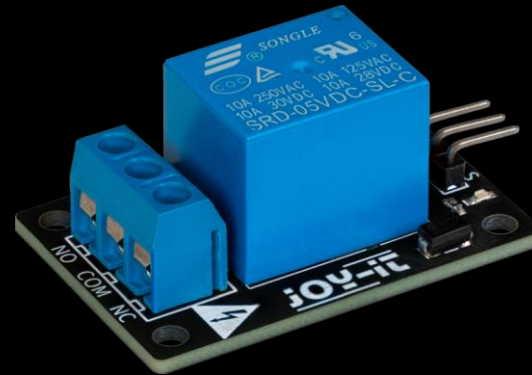
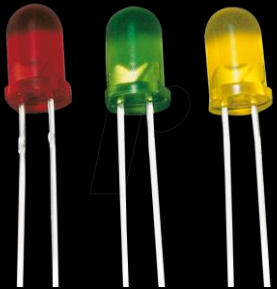
# Hardware corner: GPIO

- GPIOs are used for controlling devices with binary logic
  - Buttons (pressed / not pressed)
  - Lights (on / off)
  - Relays (current flows / current does not flow)
- GPIOs can also be used for manual signal generation



# Hardware corner: GPIO

???





# Hardware corner: GPIO

- Merci *Le Chat* (Mistral AI) ☹️
- Ceci est un signal *analogique* qu'il est bien sûr impossible de générer avec une GPIO !



# Hardware corner: GPIO

- Programming the GPIO is possible thanks to registers
  - Some **register bank** is associated with the GPIO
  - The register bank is linked to the CPU through the I/O bus
- As input...
  - Default value can be set (push/pull : pull-up == VREF, pull-down == GND)
  - Default value can also be left floating/tristate which can be required by the facing component's datasheet
  - CPU reads value from register
- As output...
  - CPU writes value to register



# Hardware corner: GPIO

- Finally, GPIOs are usually organized in GPIO banks
- "Pin A-3" means GPIO bank A, pin number 3
  - Beware 0-based counting ;-)
- Banks are usually powered separately
  - This opens the way to power-efficient designs
- Each GPIO bank has got its register bank
  - Thus each GPIO bank is operated independently

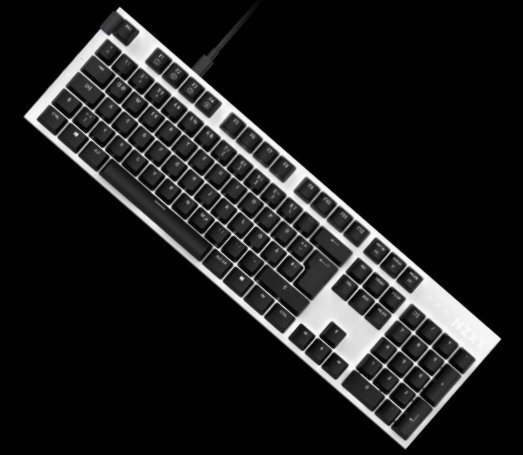


# Hardware corner: GPIO

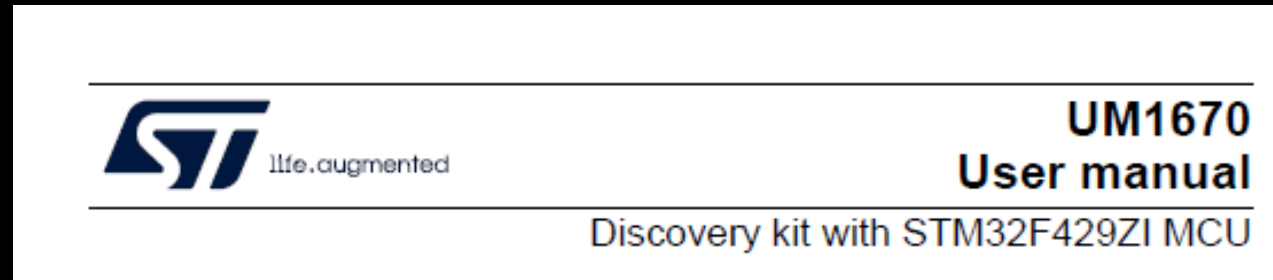
- Let us build an hardware architecture model for the STM32F429I-DISC1 development kit
- During the course, we shall enrich that model
- We begin our architecture study with the green LED
- You will now need to master the documentation needed to get correct information on a microcontroller system
- We begin with the development kit's user manual !



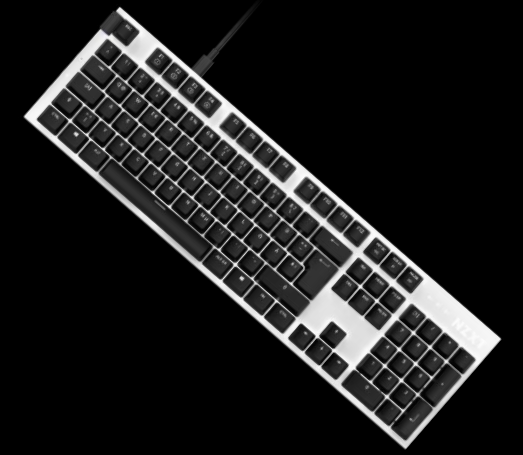
# Hardware corner: GPIO



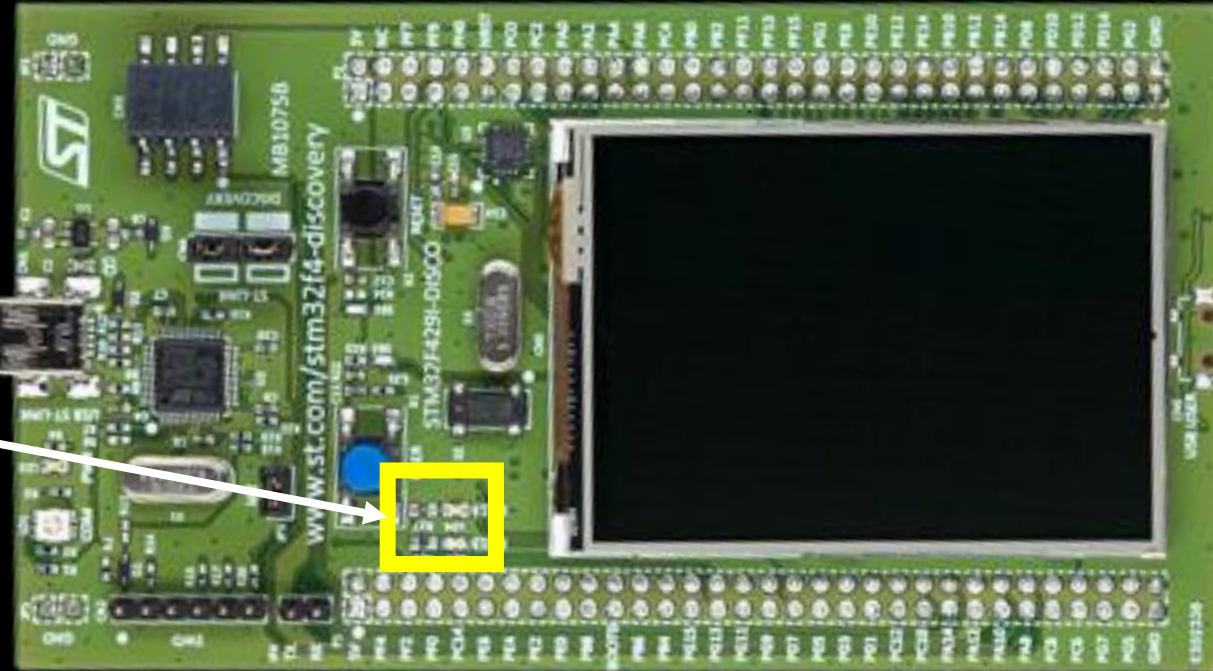
- Find the STM32F429I-DISC1's user manual on st.com
  - Official name: UM1670
  - Target document revision: 5



# Hardware corner: GPIO



LD3



Find LD3 in the STM32F429I-DISC1 user manual.

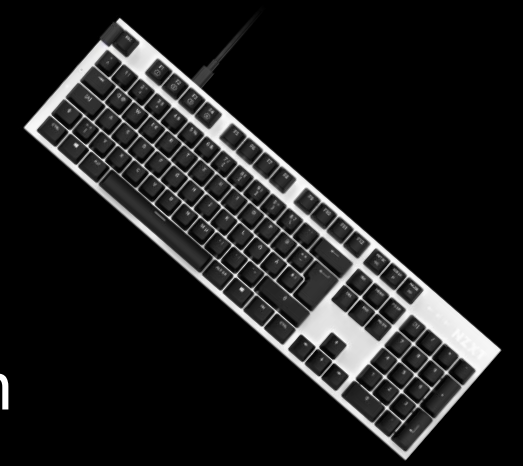


# Hardware corner: GPIO

- The development kit's user manual describes hardware information specific to the board
  - LD3 is routed to some GPIO of the MCU
  - Documented by EE (Electronics Engineers) who built the devkit...
  - ...who are not the EE who built the MCU !
- Information specific to the MCU will be found elsewhere
- Different levels of hardware → different documents
- Our low-level treasure trove: the MCU reference manual !



# Hardware corner: GPIO



- Find the STM32F429 MCU reference manual on st.com
  - Official name: RM0090
  - Target document revision: 20



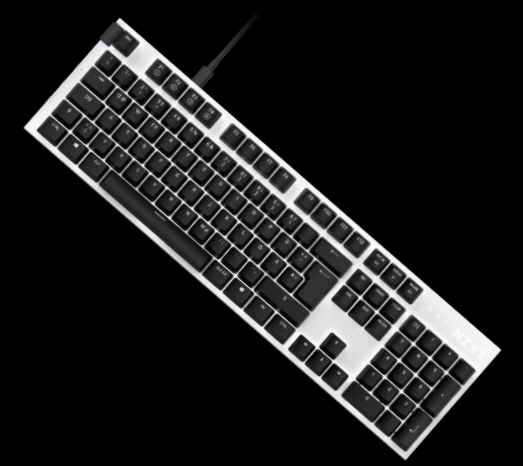
## RM0090 Reference manual

STM32F405/415, STM32F407/417, STM32F427/437 and  
STM32F429/439 advanced Arm<sup>®</sup>-based 32-bit MCUs





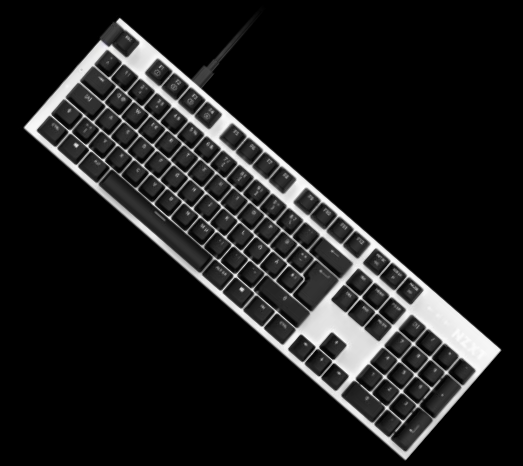
# Hardware corner: GPIO



- Using the two documents in your hands:
  - Find which GPIO bank + pin number is routed to LD3
  - Find which the address in memory of the associated register bank, in the STM32F429 MCU memory system



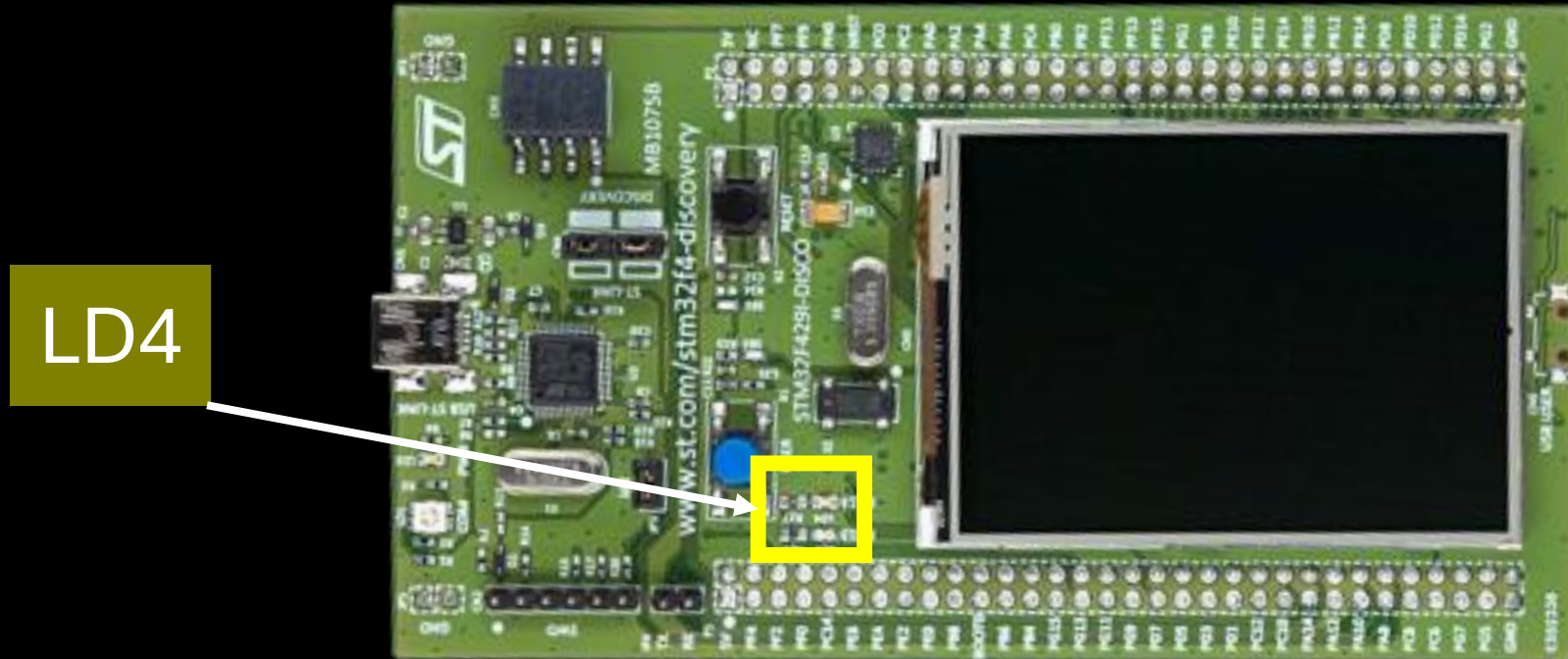
# Hardware corner: GPIO



- Using the two documents in your hands:
  - Find which GPIO bank + pin number is routed to LD3
  - GPIO PG13 → bank G, pin 13 (UM1670 § 6.5)
  - Find which the address in memory of the associated register bank, in the STM32F429 MCU memory system
  - 0x4002\_1800 (RM0090 table 1 “STM32F4xx register boundary addresses”)



# Hardware corner: GPIO



LD3 is actually named PG13 in the STM32F429 documentation  
So GPIO bank G, pin number 13

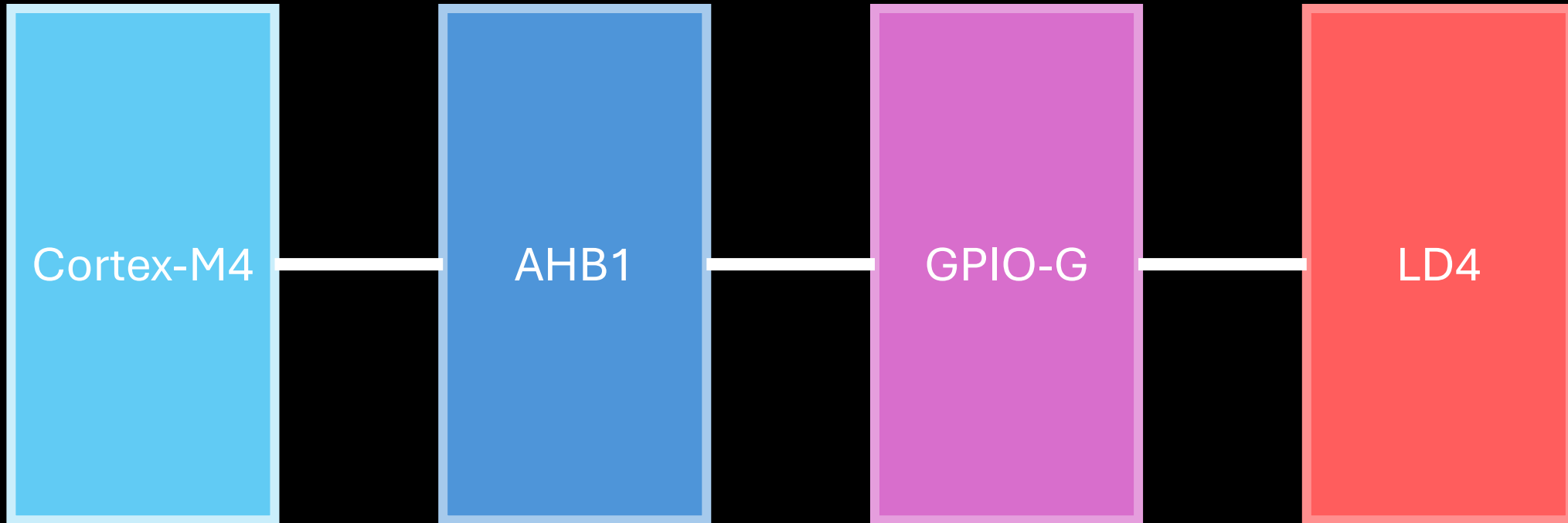


# Hardware corner: GPIO

- Let us build an architecture model for the STM32F429I-DISC1 development kit
- During the course, we shall enrich that model
- We begin our architecture study with how CPU connects with LD4
- You will now discover the documentation needed to get correct information on a microcontroller system



# Hardware corner: GPIO



In this data flow diagram, the CPU (Cortex-M4) is connected to the high-speed AHB bus.

The AHB bus gives access to many peripherals, including GPIO banks.

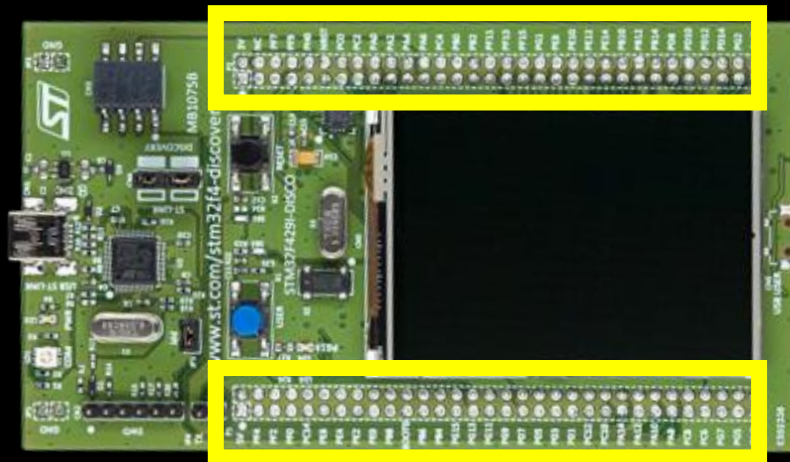
The GPIO bank G controls several pins.

Pin 13 is physically soldered to LD4.



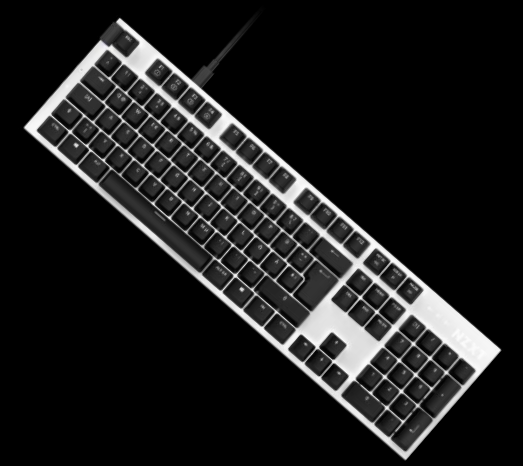
# The code behind a blinking LED

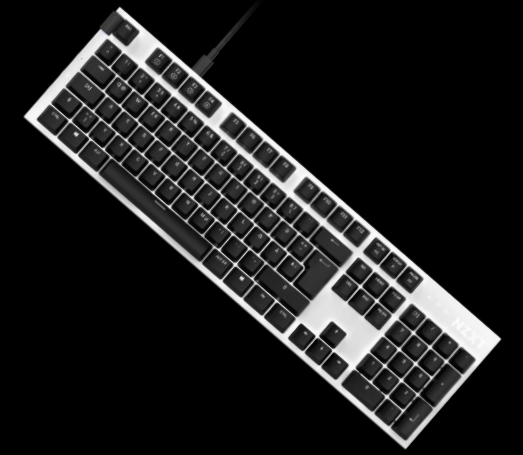
- The pins headers on your devkit are not necessarily GPIOs
- In fact, each pin of the MCU has a fixed set of functions it can be dynamically associated to (“GPIO” is one *function*)
- To know which pin can do what, you have to figure it the pin number in the general pin numbering scheme



# The code behind a blinking LED

- What document can explain MCU pin numbering ?
- What is the pin number of PG13 ?





# The code behind a blinking LED

- What document can explain MCU pin numbering ?
- What is the pin number of PG13 ?
- UM1670 table 7 (repeats pin # information)

Table 7. STM32 pin description versus board functions (continued)																				
STM32 pin		Board functions																		
Main function	LQFP144	System	VCP	SDRAM	LCD-TFT	LCD-RGB	LCD-SPI	I3G4250D	USB	LED	Push-button	I <sup>2</sup> C Ext	Touch panel	Free I/O	Power supply	CN2	CN3	CN6	P1	P2
PG13	128	-	-	-	-	-	-	-	-	Green	-	-	-	-	-	-	-	-	29	-







# The code behind a blinking LED

Like a layered cake



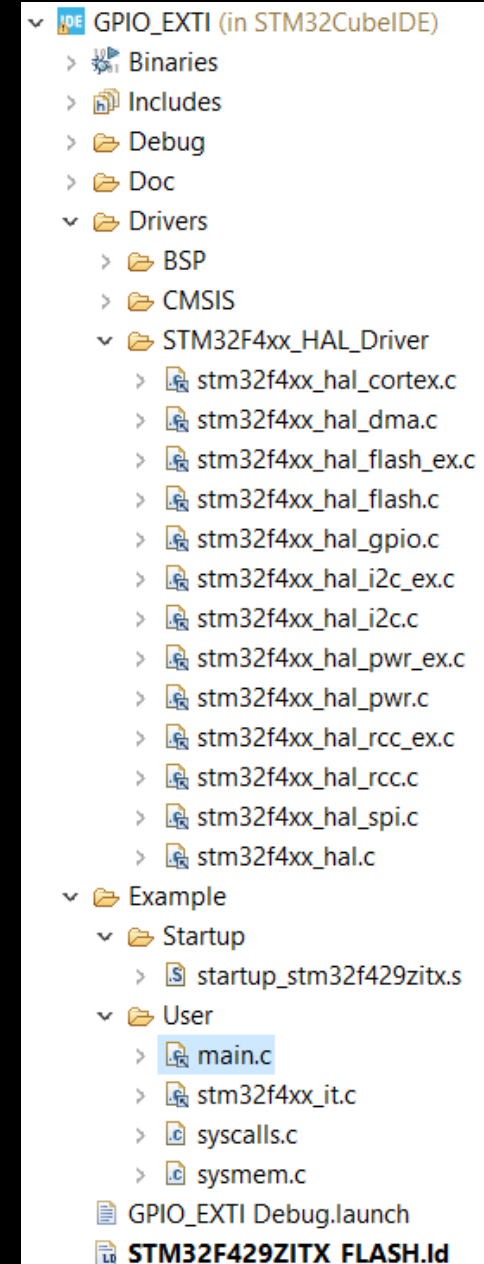
# The code behind a blinking LED

- As software developers, we should **consider each piece of hardware as a new library to learn**
- In fact, hardware manufacturers provide libraries to make our life easier → the **Hardware Abstraction Layer**
- The HAL is, by definition, specific to the target hardware
- In this course we're studying STM32F429 HAL



# The code behind a blinking LED

- Let's read the GPIO\_EXTI project structure...
- Application code :
  - Main function in *Example/User/main.c* (feel free to modify !)
- MCU initialization :
  - Hardware initialization code in *Example/Startup/\** (shouldn't touch)
- Libraries & drivers :
  - Hardware Abstraction Layer in *Drivers/STM32F4xx\_HAL\_Driver/* (vendor API)
  - CMSIS layer in *Drivers/CMSIS/\** (you can ignore)
  - STM32F429-Disco high-level layer in *Drivers/BSP/\** (vendor API)



# The code behind a blinking LED

- All this code has been generated for us...why ?
- This is a good occasion to **build a software architecture model**
- Here, we'll draw a **layer diagram** to focus on dependencies
- **This will not be a . h include tree**



# The code behind a blinking LED



GPIO\_EXTI software architecture, layered view. This simplified view focuses on GPIO control.

The app (the main() function) sits on a vendor “BSP” library.

The BSP sits on the STM32F429 HAL.

The HAL knows what registers to control, and their addresses.



# The code behind a blinking LED

GPIO\_EXTI app

```
173  /* Toggle LED3 */  
174  BSP_LED_Toggle(LED3);
```

STM32F429I Discovery BSP

```
226 void BSP_LED_Toggle(Led_TypeDef Led)  
227 {  
228     HAL_GPIO_TogglePin(GPIO_PORT[Led], GPIO_PIN[Led]);  
229 }  
230
```

STM32F429 HAL GPIO

```
433 void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)  
434 {  
435     uint32_t odr;  
436  
437     /* Check the parameters */  
438     assert_param(IS_GPIO_PIN(GPIO_Pin));  
439  
440     /* get current Output Data Register value */  
441     odr = GPIOx->ODR;  
442  
443     /* Set selected pins that were at low level, and reset ones that were high */  
444     GPIOx->BSRR = ((odr & GPIO_Pin) << GPIO_NUMBER) | (~odr & GPIO_Pin);  
445 }
```

Only the HAL knows about registers (ODR, BSRR...)



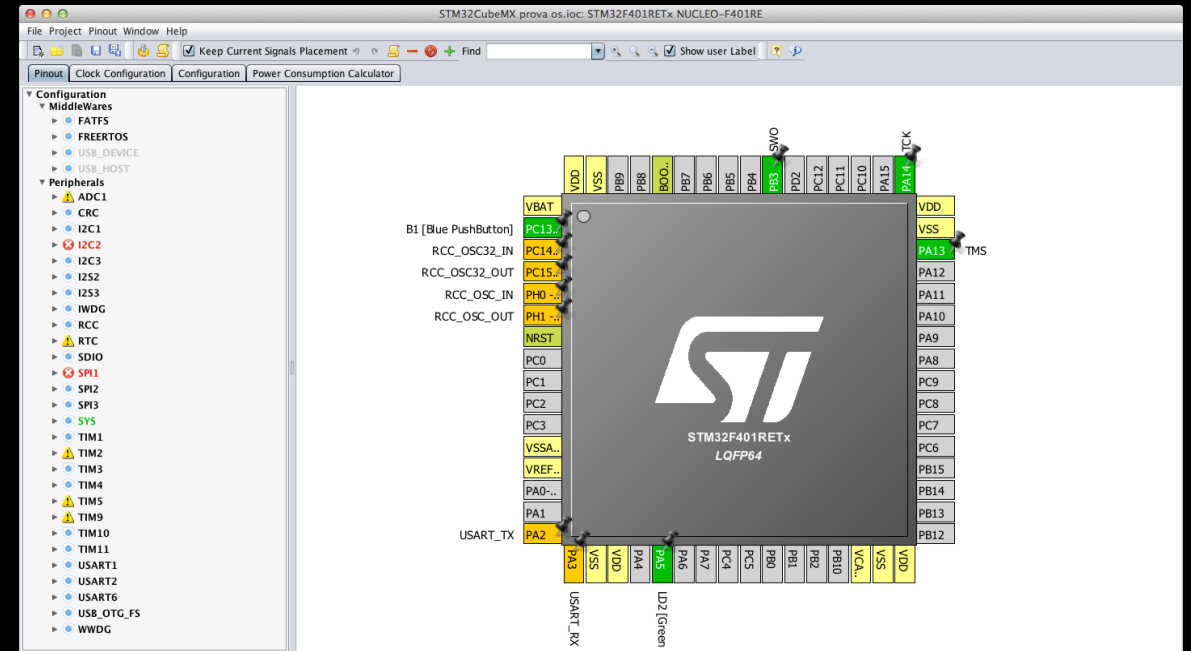
# The code behind a blinking LED

- Looks like the BSP layer just adds symbolic names
  - We should throw it away !
- The HAL is really useful
  - Practical software engineering for MCUs use the HAL
  - Developers build hardware-agnostic libraries on top of that
- Could we have controlled the GPIO by hand ?
- Let's recode this app ourselves



# Configuring pins with STM32CubeMX

# Graphical bonanza





# Configuring pins with STM32CubeMX

- We're going to do the same blinking code but ourselves, without premade code
- Let's create a new STM32 project again, but this time go to the **MCU Selector** and find our STM32F429 reference
- Don't go to example projects



# Configuring pins with STM32CubeMX

- You'll find the following variants :
  - STM32F429ZIT6
  - STM32F429ZIT6TR
  - STM32F429ZIT7
- We wanted "STM32F429ZIT6U"...
  - Can we still use one of these variants ?



# Configuring pins with STM32CubeMX

- You'll find the following variants :
  - STM32F429ZIT6
  - STM32F429ZIT6TR
  - STM32F429ZIT7
- **Any one of these is fine !**
  - Recall MCU datasheet § 8 "Part numbering" : 6 is chip temperature range, and "TR" indicate chip is packaged in tape and reel format
  - **Let's pick STM32F429ZIT6**



# Configuring pins with STM32CubeMX

**STM32 Project**

**Target Selection**  
Select STM32 target or STM32Cube example

MCU/MPU Selector | Board Selector | Example Selector | Cross Selector

MCU/MPU Filters

Commercial Part Number: STM32F429ZIT6

PRODUCT INFO

- Segment
- Series
- Line
- Marketing Status
- Price
- Package
- Core
- Coprocessor

MEMORY

Flash = 2048 (kBytes)

EEPROM = 0 (Bytes)

**STM32F4 Series**

**STM32F429ZIT6**

High-performance advanced line, Arm Cortex-M4 core with DSP and FPU, 2 Mbytes of Flash memory, 180 MHz CPU, ART Accelerator, Chrom-ARTAccelerator, FMC with SDRAM, TFT

Unit Price for 10kU (US\$) : 7.6662

Boards: NUCLEO-F429ZI - STM32F429I-DISC1

LQFP 144 20x20x1.4 mm

The STM32F427xx and STM32F429xx devices are based on the high-performance Arm® Cortex®-M4 32-bit RISC core operating at a frequency of up to 180 MHz. The Cortex-M4 core features a Floating point unit (FPU) single precision which supports all Arm® single-precision data-processing instructions and data types. It also implements a full set of DSP instructions and a memory protection unit (MPU) which enhances application security. The STM32F427xx and STM32F429xx devices incorporate high-speed embedded memories (Flash memory up to 2 Mbyte, up to 256 Kbytes of SDRAM) up to 4

MCUs/MPUs List: 2 items

	Commercial...	Part No	Reference	Marketing St...	Unit Price for...	Board	Package	Flash	RAM	I/O	Frequency
☆	STM32F429ZIT6	STM32F429ZI	STM32F429ZI...	Active	7.6662	NUCLEO_STM32	LQFP 144 20x...	2048 kBytes	256 kBytes	114	180 MHz
☆	STM32F429ZI...	STM32F429ZI	STM32F429ZI...	Active	7.6662		LQFP 144 20x...	2048 kBytes	256 kBytes	114	180 MHz

< Back | Next > | Finish | Cancel

Picking the MCU

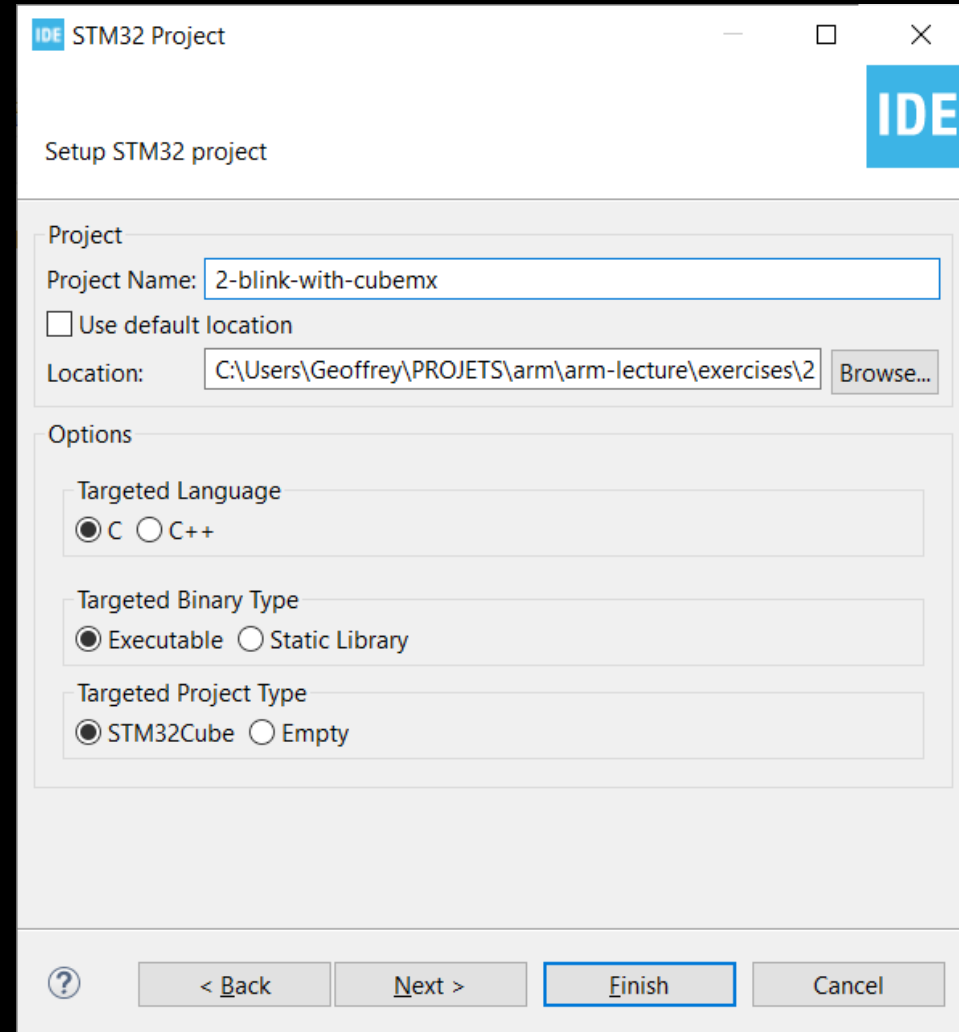


# Configuring pins with STM32CubeMX

Create a new STM32 project.

Make sure to check  
“STM32Cube” as Target Project  
Type to get the CubeMX  
environment.

We'll use it to configure GPIOs  
using a GUI.

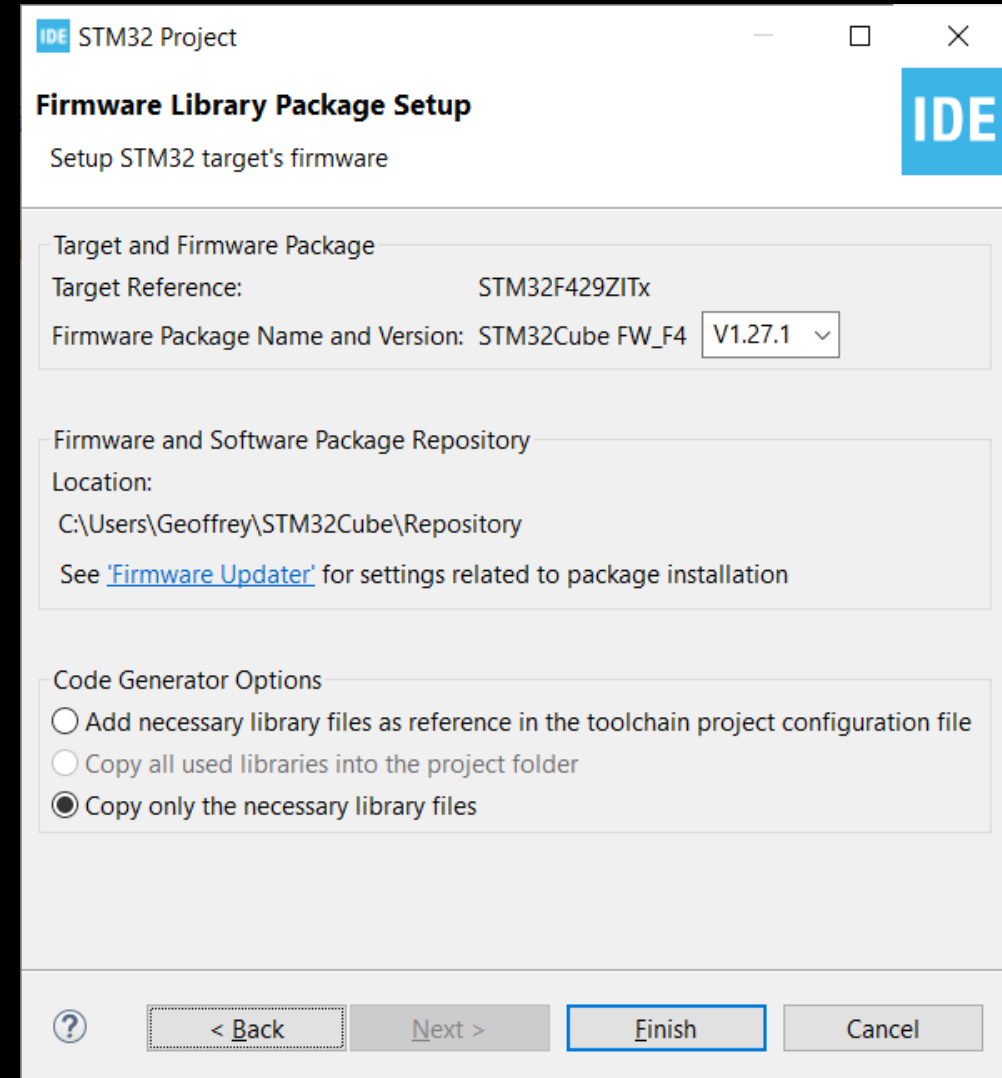


The screenshot shows the 'STM32 Project' setup window in the IDE. The window has a title bar with the IDE logo and the text 'STM32 Project'. The main content area is titled 'Setup STM32 project'. It contains two sections: 'Project' and 'Options'. In the 'Project' section, the 'Project Name' is '2-blink-with-cubemx', and the 'Location' is 'C:\Users\Geoffrey\PROJETS\arm\arm-lecture\exercises\2'. There is a 'Browse...' button next to the location field. In the 'Options' section, there are three groups of radio buttons: 'Targeted Language' with 'C' selected, 'Targeted Binary Type' with 'Executable' selected, and 'Targeted Project Type' with 'STM32Cube' selected. At the bottom of the window, there are four buttons: a help button (question mark), '< Back', 'Next >', and 'Finish' (which is highlighted with a blue border), and a 'Cancel' button.



# Configuring pins with STM32CubeMX

“Initialize peripherals to default ?”  
==> Yes



The screenshot shows the 'Firmware Library Package Setup' dialog box in the STM32CubeMX IDE. The dialog is titled 'STM32 Project' and 'Firmware Library Package Setup'. It contains three main sections: 'Target and Firmware Package', 'Firmware and Software Package Repository', and 'Code Generator Options'. The 'Target and Firmware Package' section shows 'Target Reference: STM32F429ZITx' and 'Firmware Package Name and Version: STM32Cube FW\_F4 V1.27.1'. The 'Firmware and Software Package Repository' section shows 'Location: C:\Users\Geoffrey\STM32Cube\Repository' and a link to 'Firmware Updater'. The 'Code Generator Options' section has three radio buttons: 'Add necessary library files as reference in the toolchain project configuration file', 'Copy all used libraries into the project folder', and 'Copy only the necessary library files' (which is selected). At the bottom, there are buttons for '?', '< Back', 'Next >', 'Finish', and 'Cancel'.

IDE STM32 Project

**Firmware Library Package Setup**

Setup STM32 target's firmware

Target and Firmware Package

Target Reference: STM32F429ZITx

Firmware Package Name and Version: STM32Cube FW\_F4 V1.27.1

Firmware and Software Package Repository

Location:  
C:\Users\Geoffrey\STM32Cube\Repository

See ['Firmware Updater'](#) for settings related to package installation

Code Generator Options

☐ Add necessary library files as reference in the toolchain project configuration file

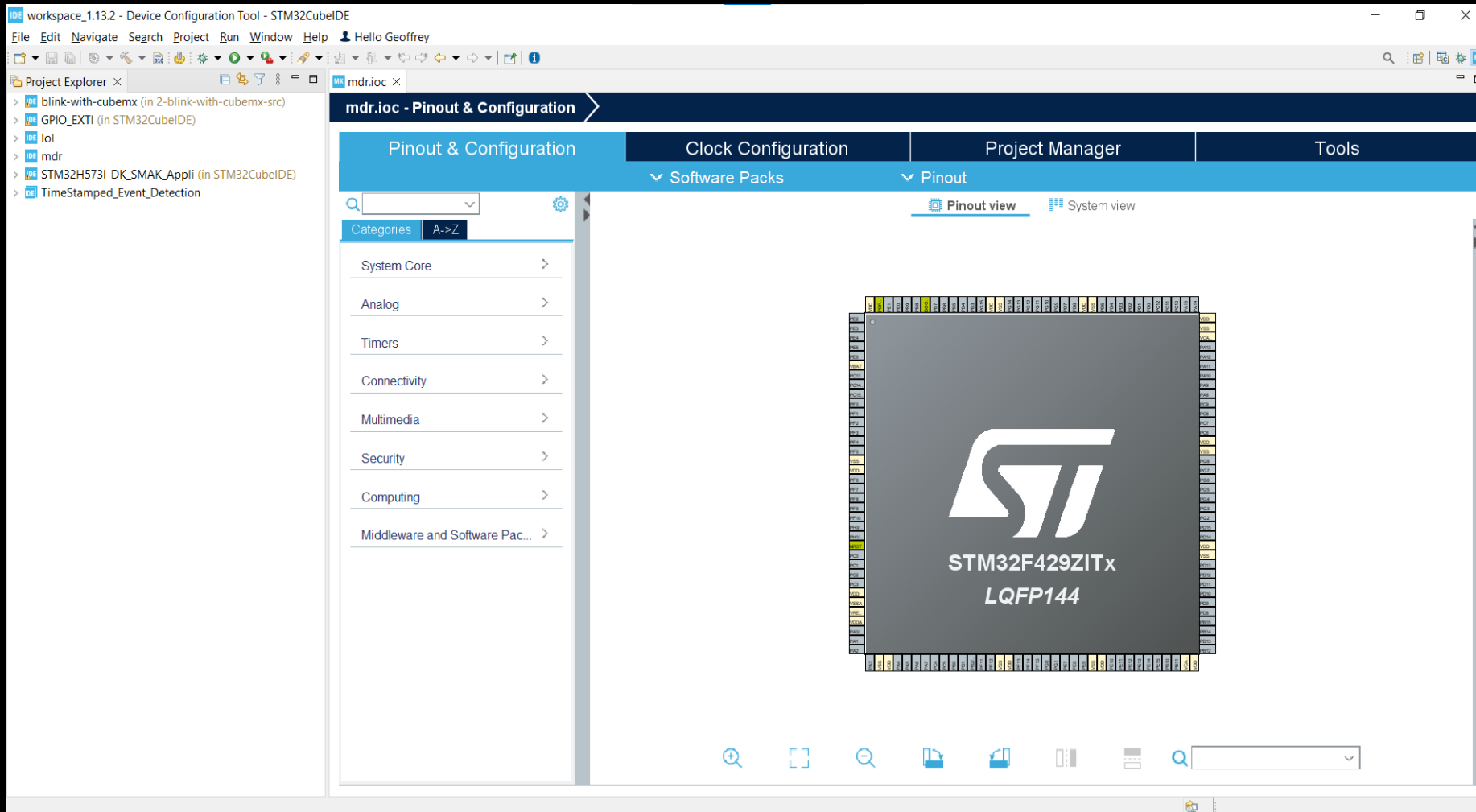
☐ Copy all used libraries into the project folder

☒ Copy only the necessary library files

? < Back Next > Finish Cancel



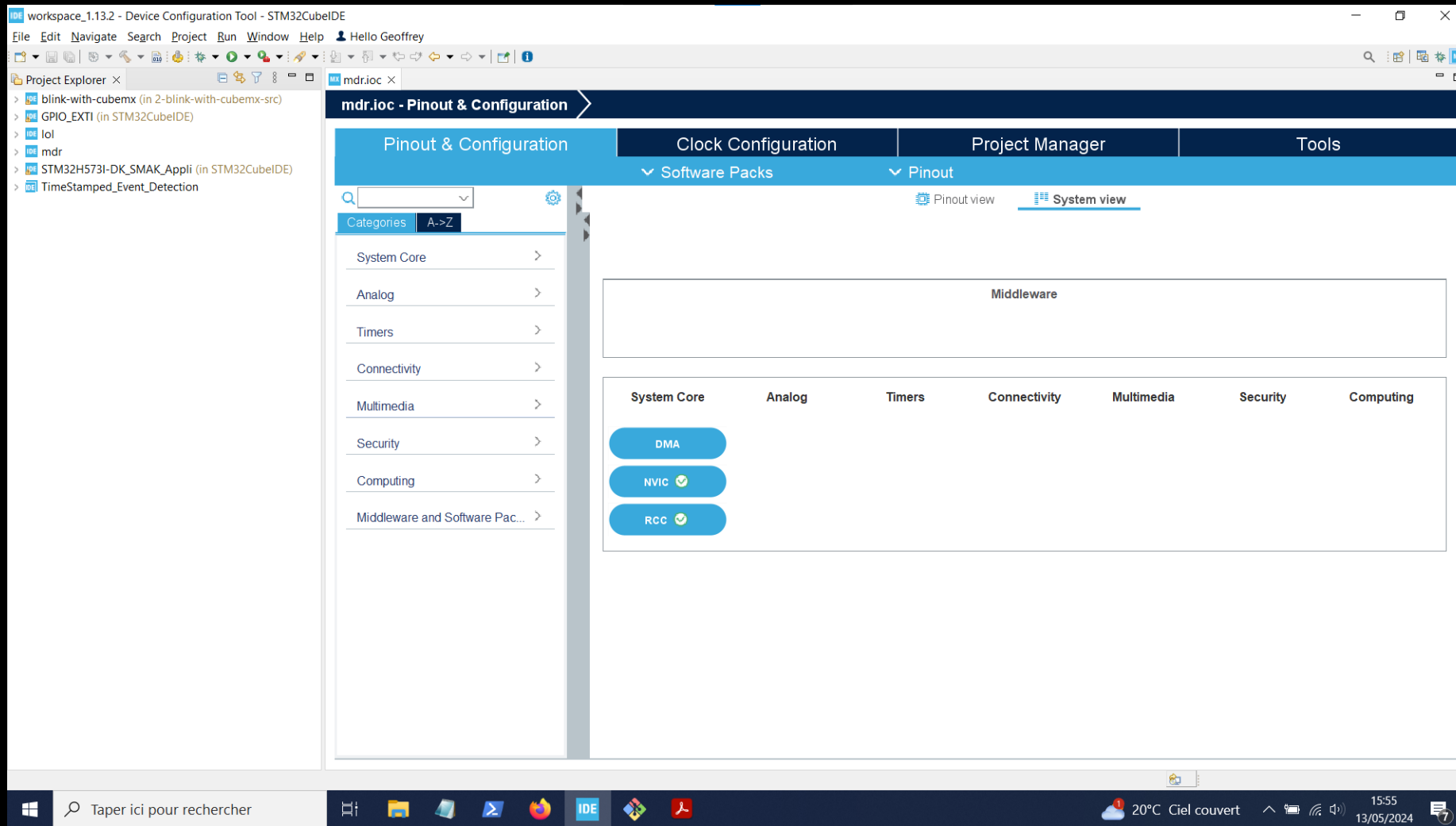
# Configuring pins with STM32CubeMX



Our new project ! Meet CubeMX GUI.



# Configuring pins with STM32CubeMX



The *System view* lists all enabled peripherals. For now we have the bare minimum.



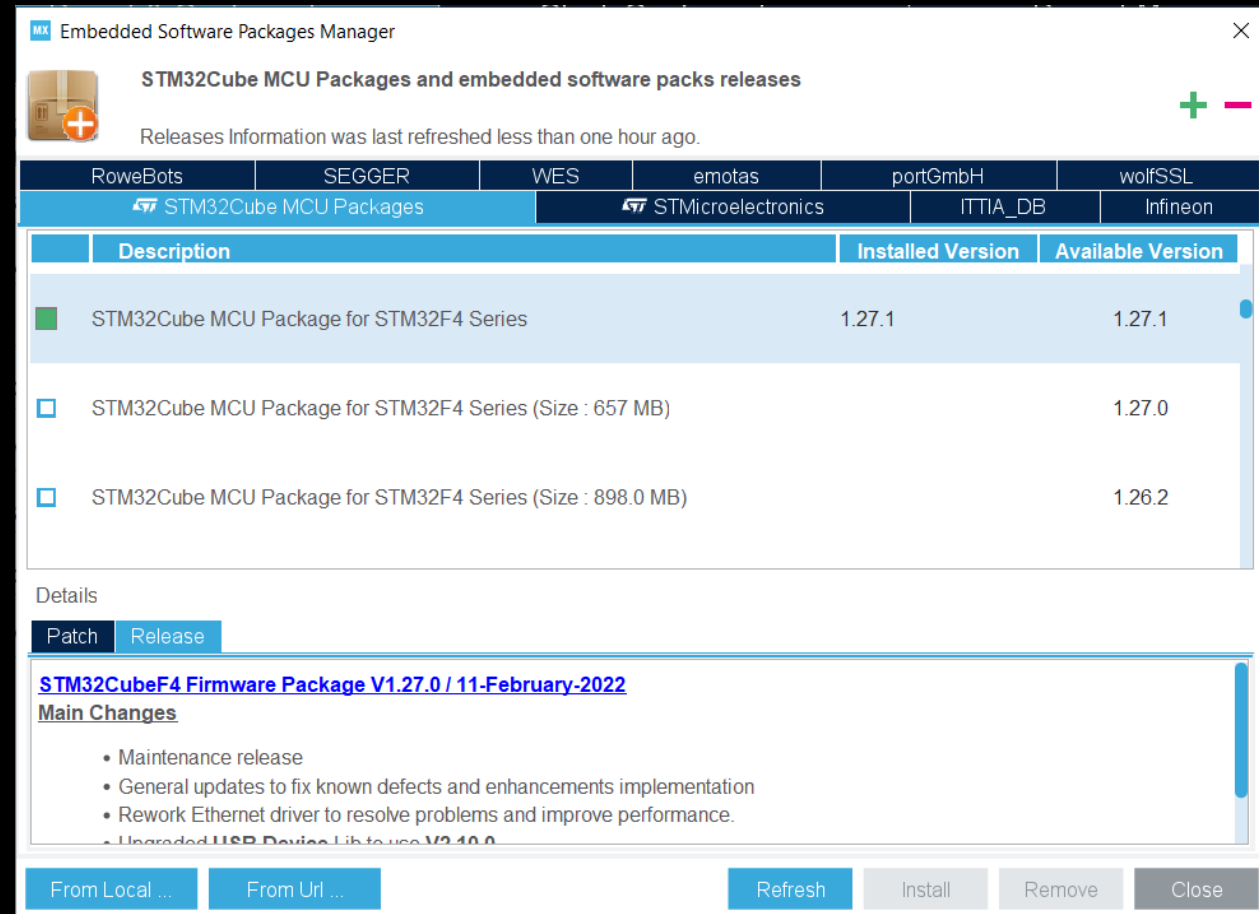


# Configuring pins with STM32CubeMX

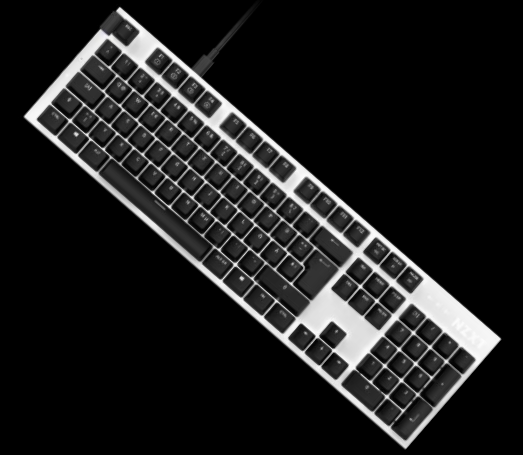
Clicking on « Software Packs » shows that we have installed STM32CubeF4 package.

It contains :

- HAL
- Code generator
- Some of the GUI's « backend »



# Configuring pins with STM32CubeMX



- Build and run default code
- LED does not react to button press : this project only initializes (prepare use of) peripherals but does not change their state
- Compare the code to the previous example
  - What has been added ? Removed ?
  - What missing logic do we need to code ?



# Configuring pins with STM32CubeMX

- We have to code two things :
  1. Lighting on/off the LED (LED3)
  2. Reacting to button press (USER)
- You can get inspiration from previous code
- Don't forget the STM32F4 HAL reference manual for details



# Configuring pins with STM32CubeMX

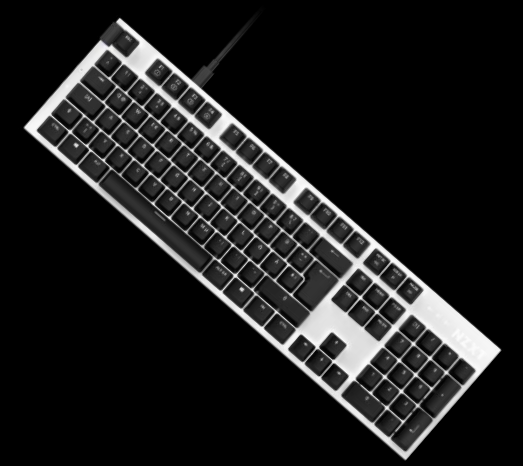
EPITA LED blinking app

STM32F429 HAL GPIO

Manual LED blinking target software architecture



# Configuring pins with STM32CubeMX



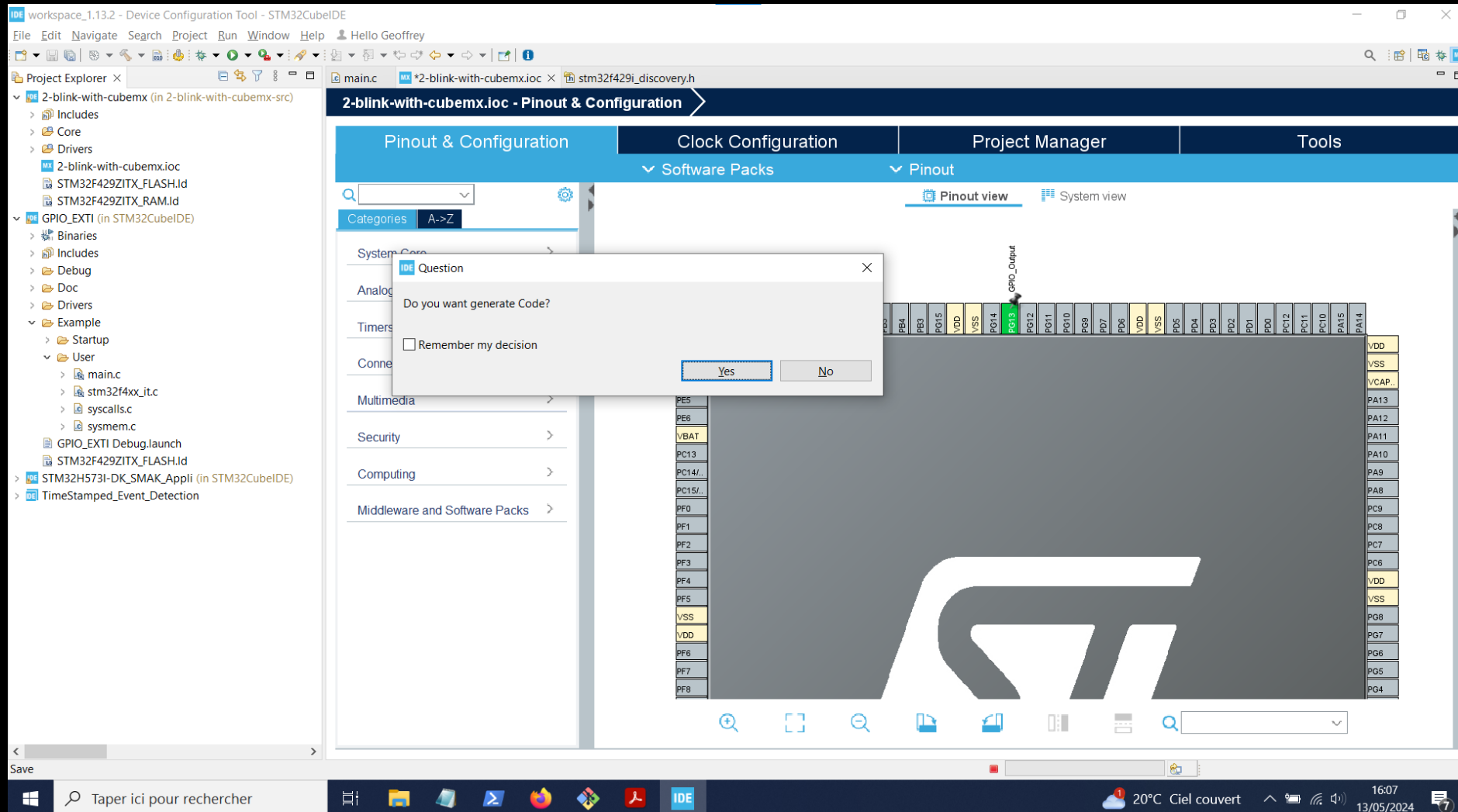
Let's start with the LED



1. Go to "Pinout & Configuration" (if needed, reopen .ioc file) and configure the pin behind LED3 as a GPIO output
2. Regenerate code
3. Code *gistre\_led\_turnon()* and *gistre\_led\_turnoff()* functions
4. Turn on the LED at program startup



# Configuring pins with STM32CubeMX



When you save changes in .ioc, code generator kicks in



# Configuring pins with STM32CubeMX

- Cool, we can light up a LED !
- And we got rid of ST's BSP code...
- Find the exercise with commented code on Moodle
- Now we need to program the button
- *We'll see that in the next chapter !*



# In the next chapter

- New hardware feature: interrupts
- Input GPIO configuration using STM32 CubeIDE
- More about the STM32 HAL, and C weak symbols !
- Until then:
  - Read full code of GPIO\_EXTI
  - Read § 2.1 / 2.2 / 2.3 of RM0090 (global system architecture)
  - Read § 8.1 / 8.2 / 8.4 of RM0090 (GPIO features + registers)





# Quick recap



## Technologies :

- arm-none-eabi-gcc
- openocd

## Concepts :

- Cross-compiler
- Cross-debugger
- Flasher
- Hardware Abstraction Layer
- Host (when compiling)
- Native compiler
- Reference manual (for a MCU)
- Register bank



# Quick recap



## Concepts :

- Target (when compiling)
- Tuple (full compiler name)
- User manual (for a devkit)

