Introduction to STM32

Dale Wheat

October 2019

Dallas Makerspace

v1.11

Course Outline

- Introduction
- Software installation
- Software configuration
- Exercises
- Conclusion

What will we learn today?

- What is STM32?
- How to use STM32-specific development software
- How to program simple embedded programs in C
- How to debug your programs
- How to add simple peripherals to STM32

What is STM32?

- Product of STMicroelectronics • st.com
- A family of 32-bit Flash microcontrollers
- Based on Arm® Cortex®-M processors • arm.com

ARM Cortex-M series

- Cortex-M3 announced by ARM Holdings in 2004
- ARM Cortex-M uses ARMv7-M microarchitecture
  (not to be confused with ARM7 devices)
- First devices to market were Luminary Micro LM3S101 & LM3S102

Cortex-M series

- Cortex-M0
- Cortex-M0+
- Cortex-M1
- Cortex-M3
- Cortex-M4
- Cortex-M7
- Cortex-M23
- Cortex-M33
- Cortex-M35P

What will you take home?

- A brief introduction to STM32
- An understanding of how it compares to other available solutions
- Access to STM32-specific development software
- Sources for STM32 hardware

- Optional: Your very own STM32 experimenter's starter kit

Is STM32 like Arduino?

- Yes
  - A microcontroller executes user’s programs
  - Small form factor
  - Low cost
  - Modern tools make it simple to learn and use
  - Widely documented on the Internet
- No
  - Arduino is based on Atmel AVR 8-bit architecture at 16 MHz
  - STM32 is based on Arm® Cortex®-M 32-bit architecture at 32 MHz to 480 MHz
  - Not all STM32 software is open source

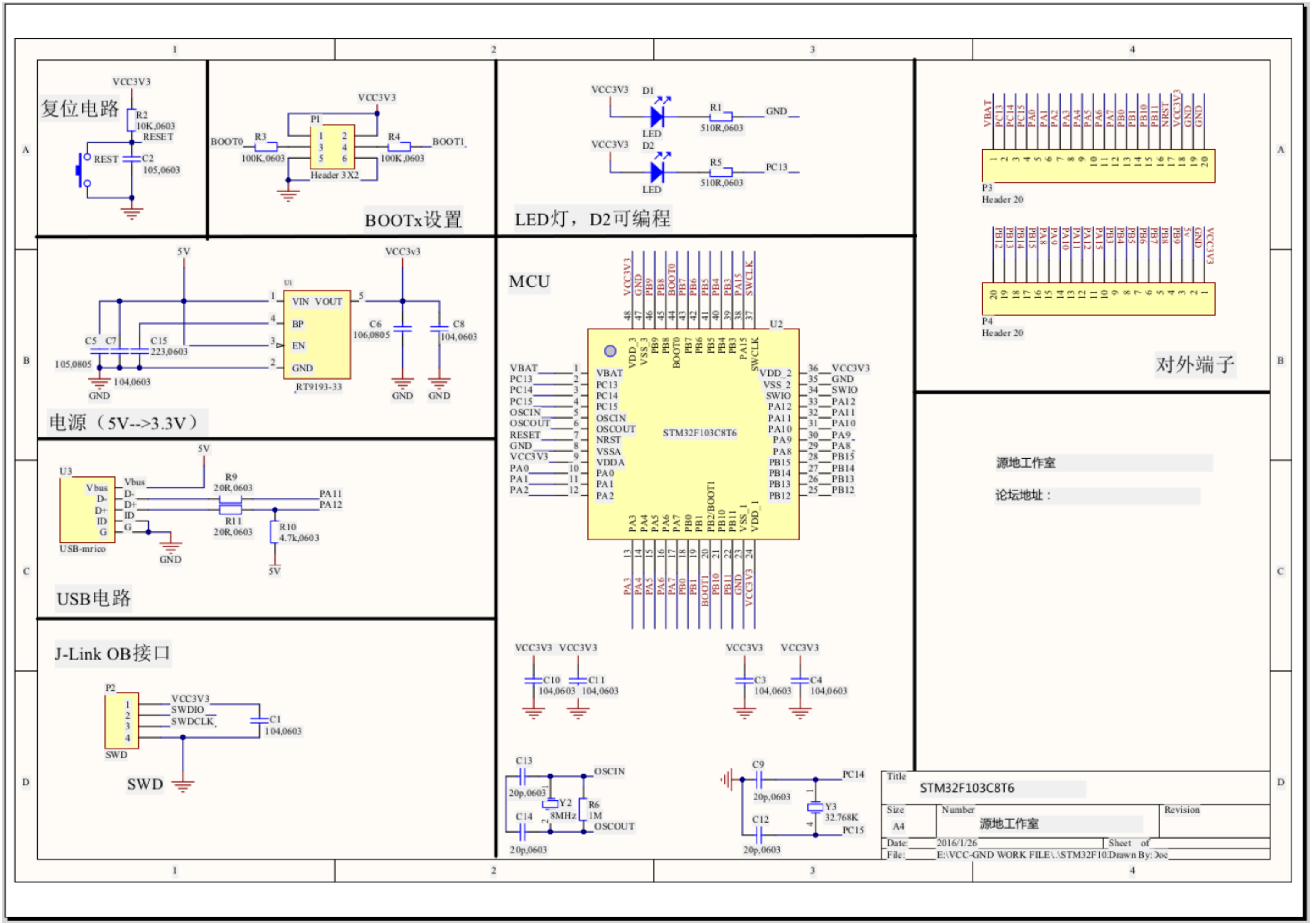
STM32 Experimenter’s Starter Kit

- Check your kit for the following items:
  - Solderless breadboard containing:
    - STM32 “Blue Pill” board
    - ST-LINK V2 USB interface
    - Four push buttons
  - Rainbow-colored jumper wires
  - USB to TTL serial adapter
  - USB cable (A to Micro-B)
  - Bag of LEDs and resistors
  - LCD module

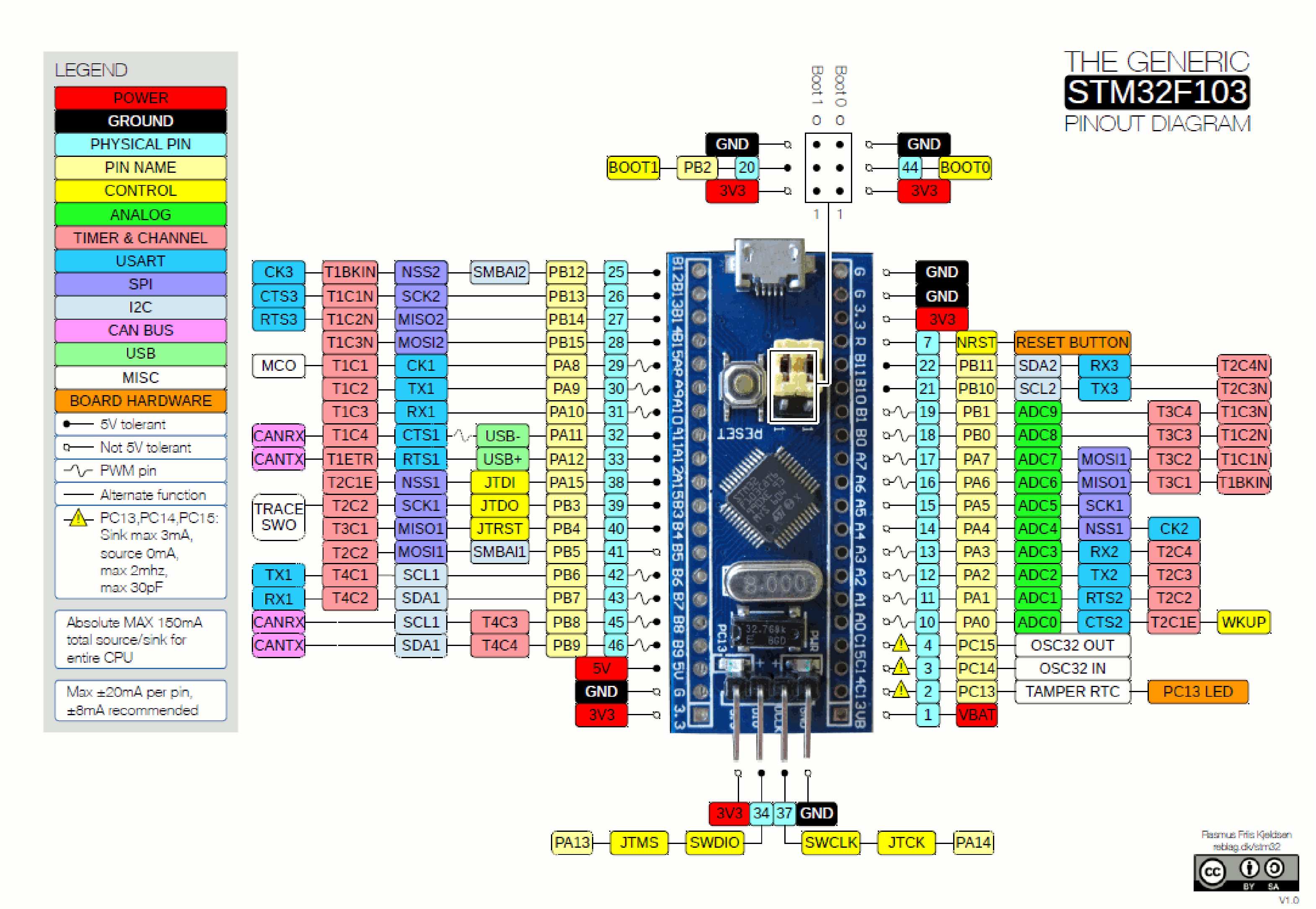
The Blue Pill

- Low cost, generic ARM development module
- Based on STM32F103C8T6
- 40 pin DIP (dual in-line package)
- Headers are usually optional

Blue Pill Schematic



Blue Pill Pin Assignments



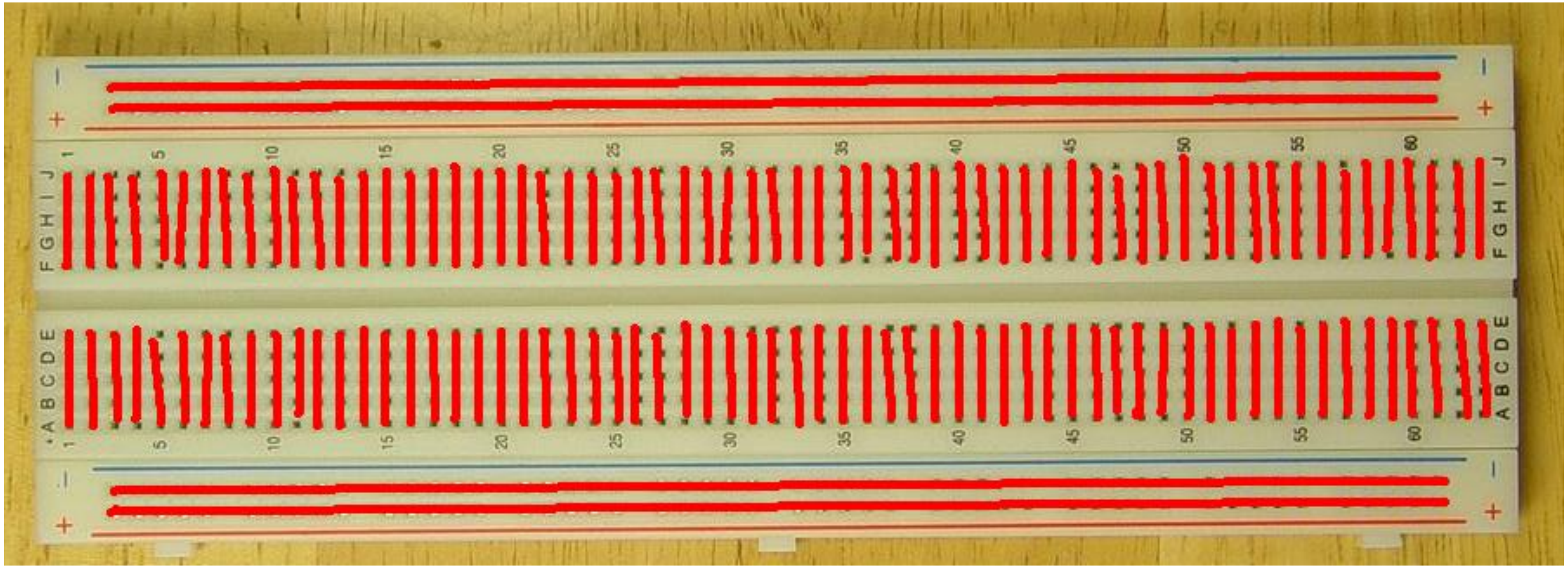
ST-LINK/V2

- Connection to blue pill (4 wires)
  - GND – ground reference
  - SWCLK – clock
  - SWDIO – data
  - 3V3 – 3.3V power

- The ST-LINK/V2 device programmer should already be connected to your Blue Pill
- Don’t disconnect it if you don’t have to!

Solderless Breadboard Basics

- Spring-loaded sockets hold wires in place
- Red lines indicate electrical connections



Important safety information

- Always unplug the ST-LINK/V2 from the USB port when making wiring updates.
- Do not add or remove components when board is powered on.

Software Installation

- STM32CubeIDE application
- Additional source code
  - LCD example code
  - Code from exercises

Software Installation

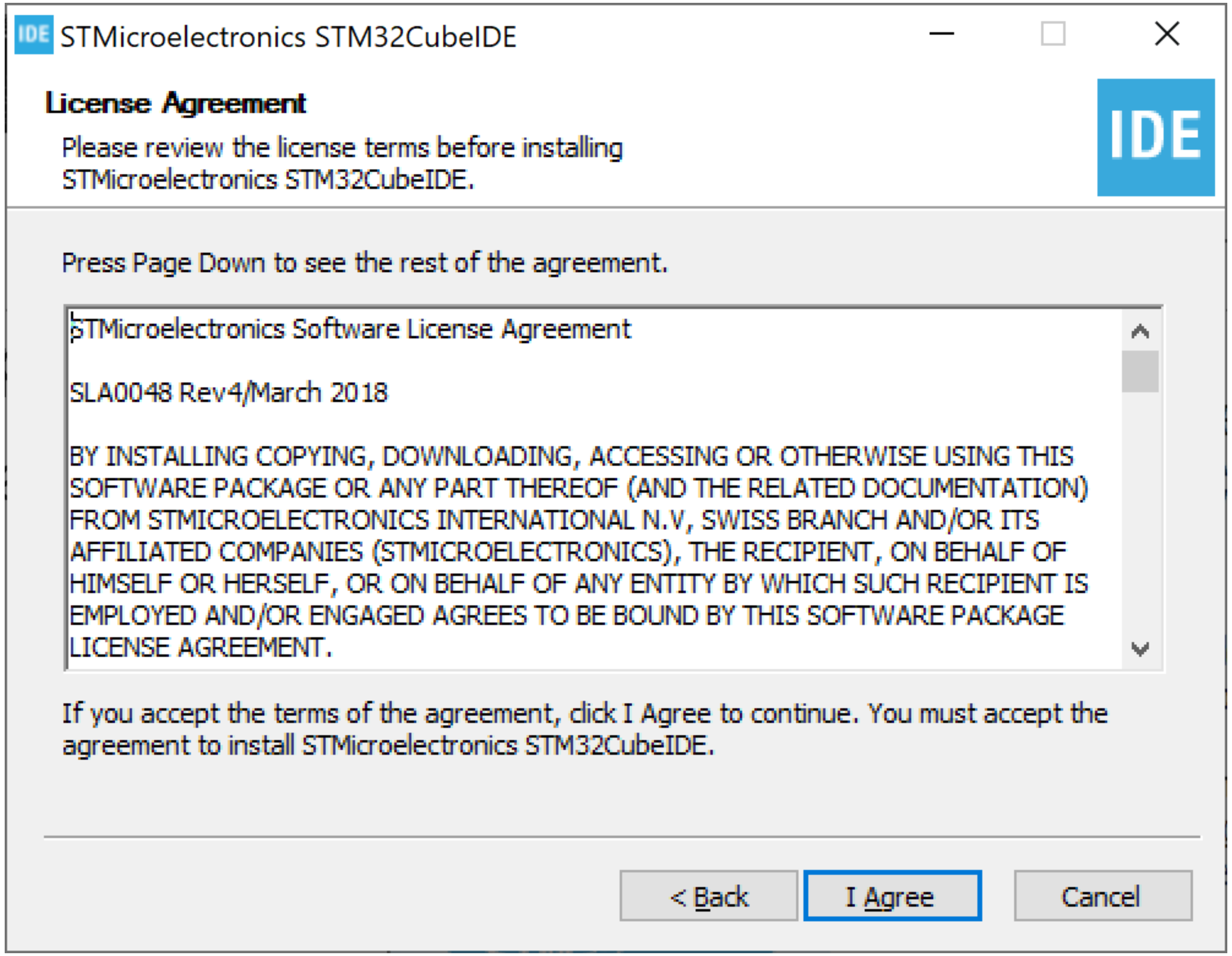
- **Install STM32CubeIDE application**
  - File: en.st-stm32cubeide_1.0.2_3566_20190716-0927_x86_64.exe.zip
  - Webpage: https://www.st.com/content/st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-ides/stm32cubeide.html
  - To download the software from the ST web site, you will need to register
  - Just copy the installation file from the instructor
  - Installation is reported to require 6 GB of available hard drive space

Install STM32CubeIDE

- Execute the installation program
- This “Welcome” screen appears
- Click “Next >”

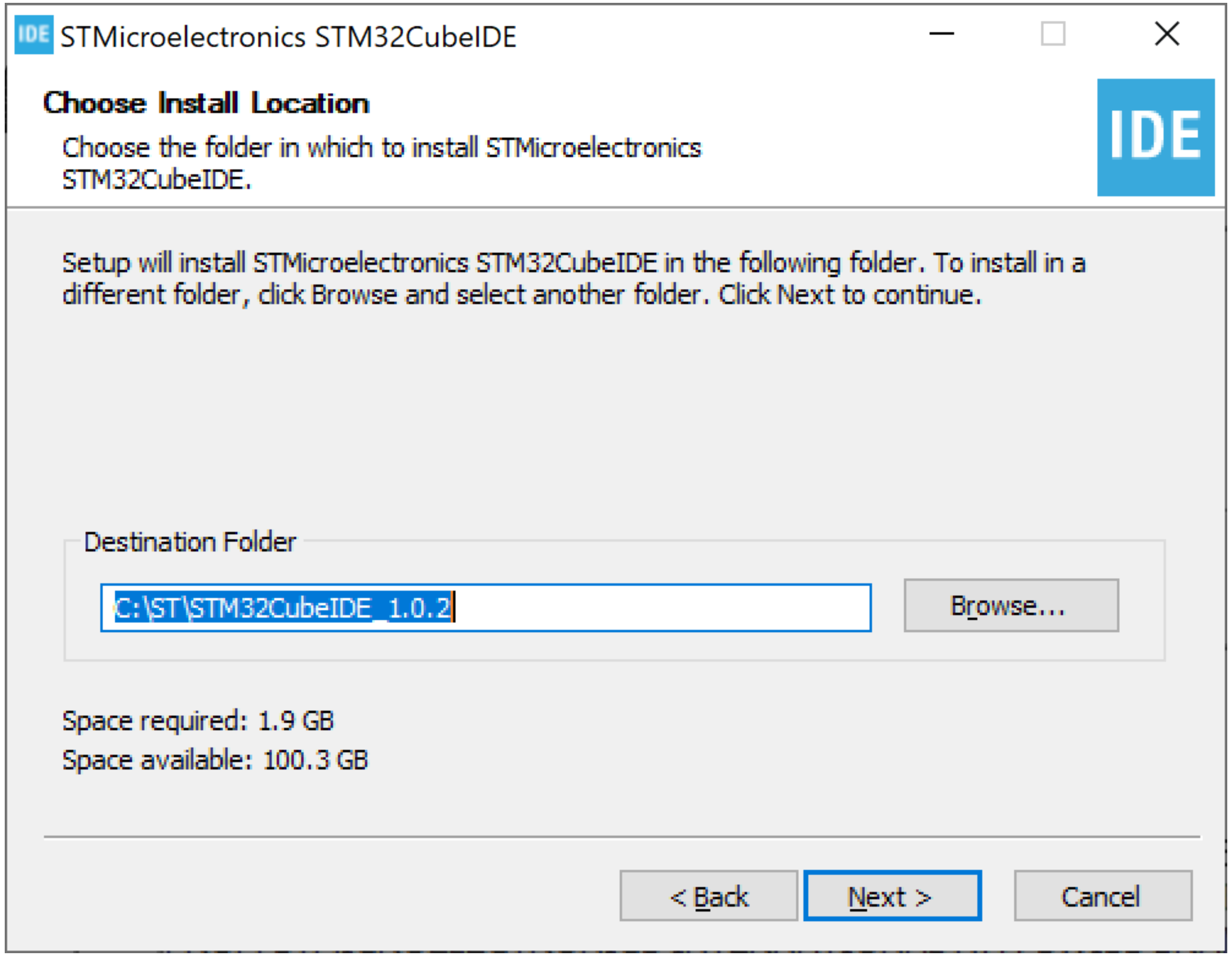
License Agreement

- Click “I Agree”



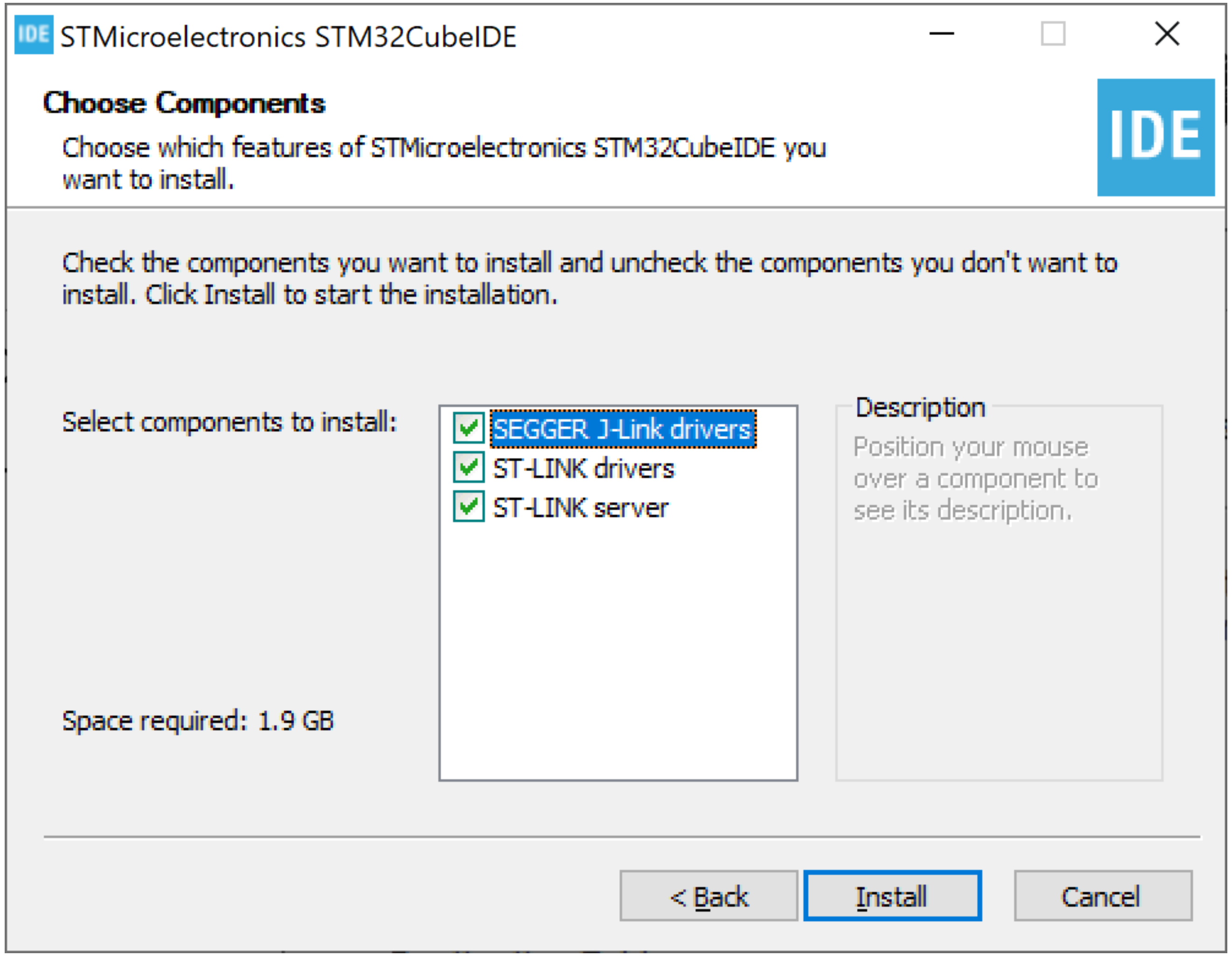
Choose Install Location

- Please use suggested location: “C:\ST\STM32CubeIDE_1.0.2”
- This is the “workspace”
- It doesn’t like spaces ☹
- Click “Next >”



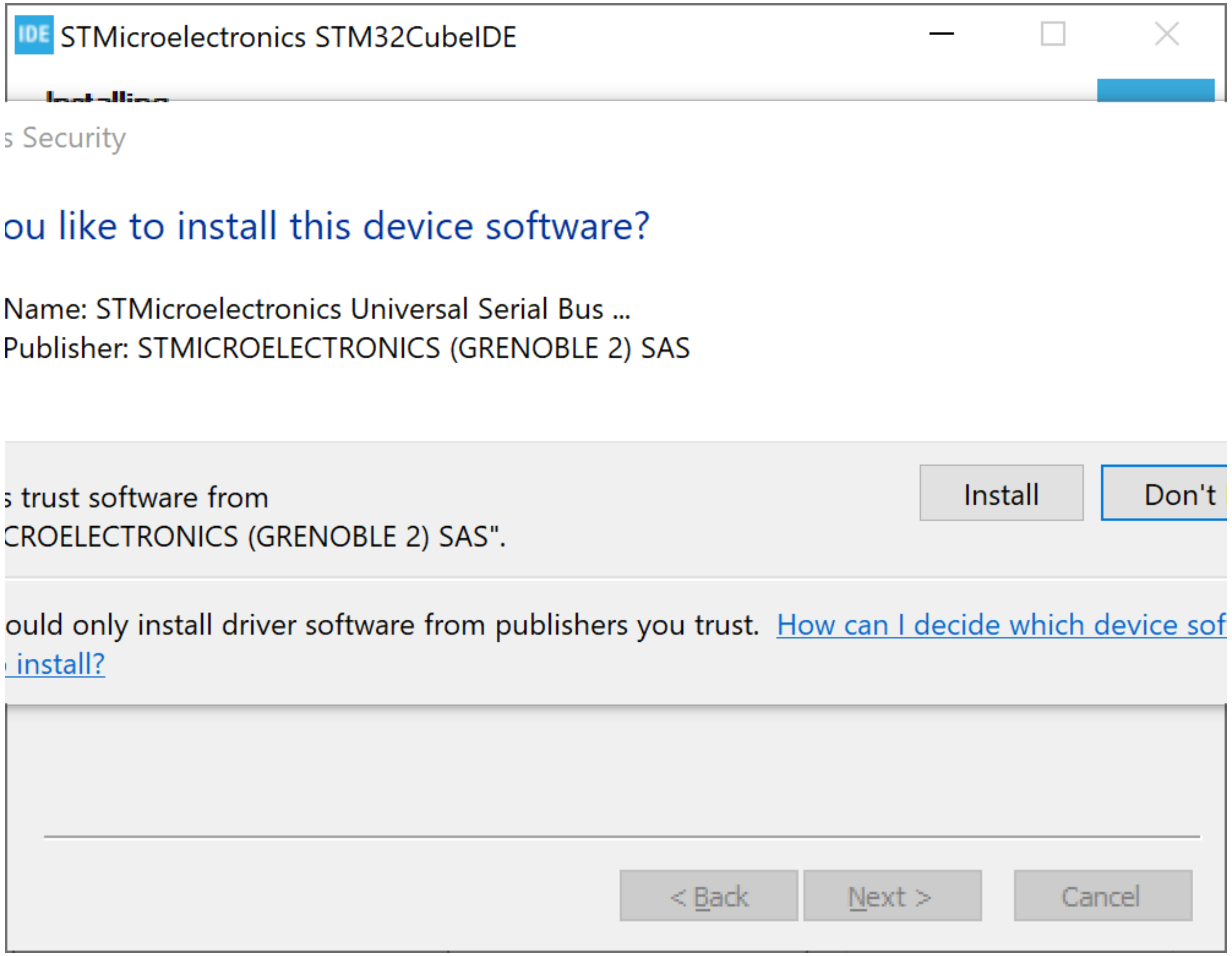
Choose Driver Components

- Leave all three components selected
- Click "Install"



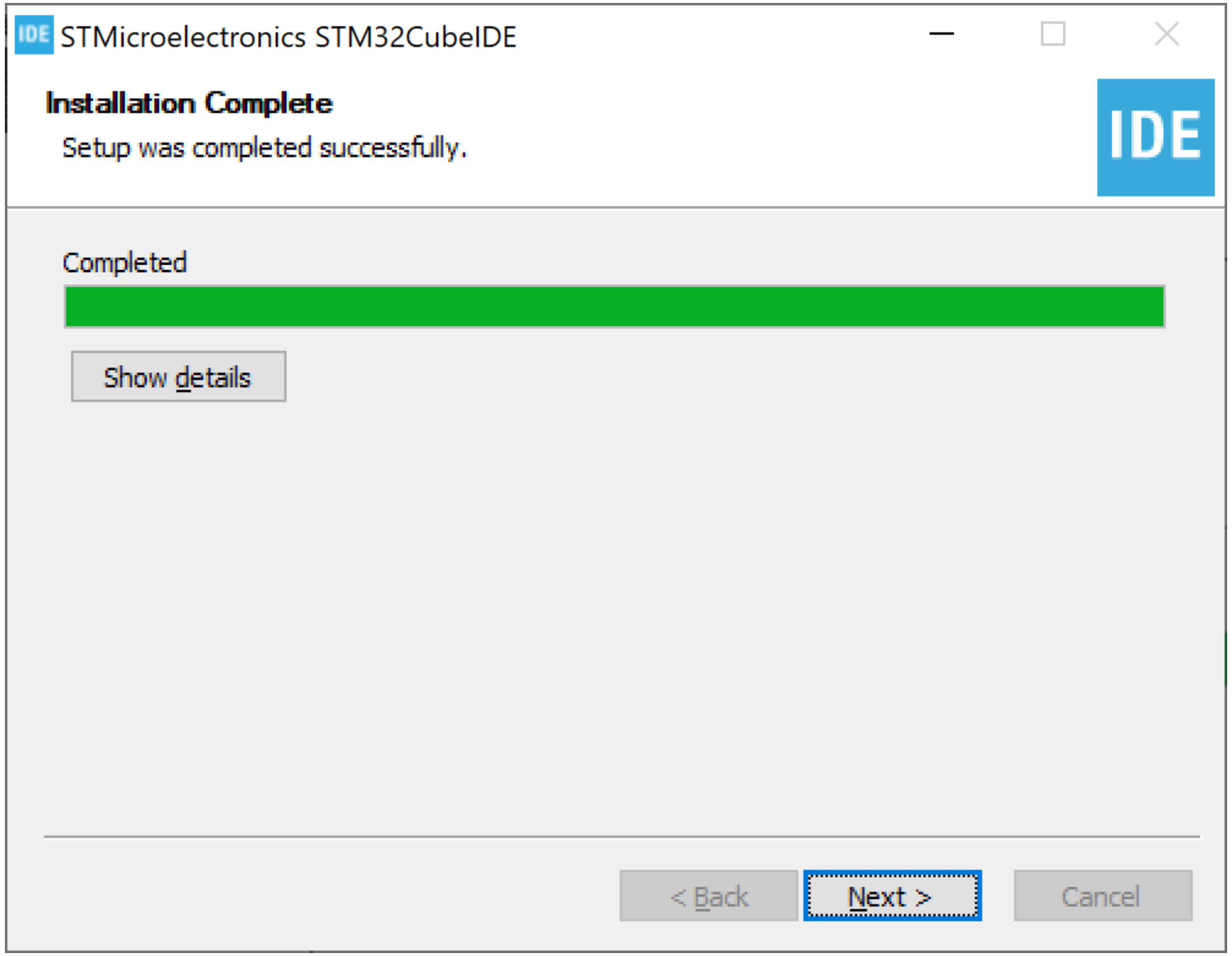
Approve Driver Installation

- Click “Install”
- Repeat two more times
- Installation now begins
- This screen-shot is not right ☹



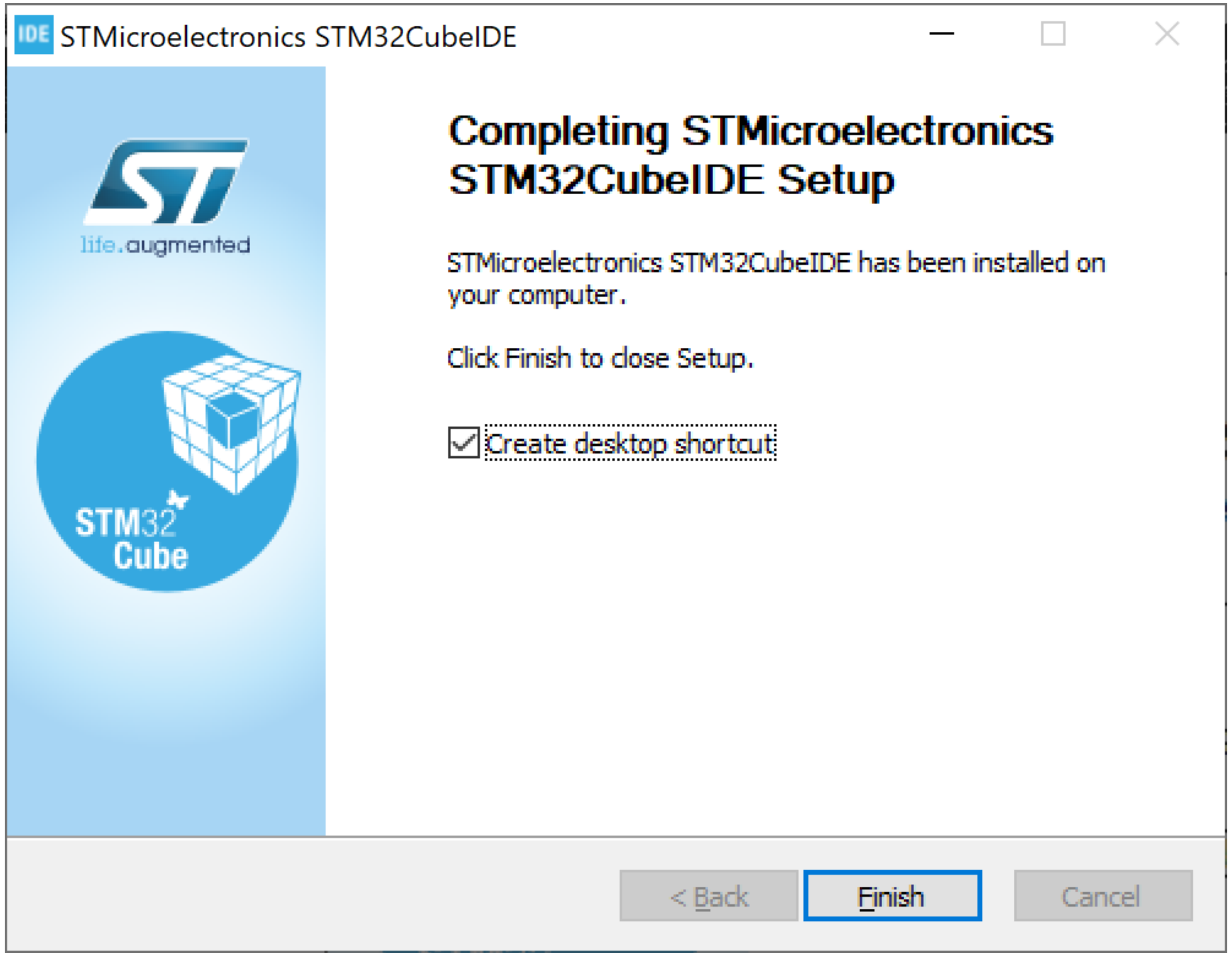
Installation Complete

- Hopefully you get to this screen
- Click “Next >”



Installation is Complete

- Leave option checked:
  - “Create desktop shortcut”
- Click “Finish”

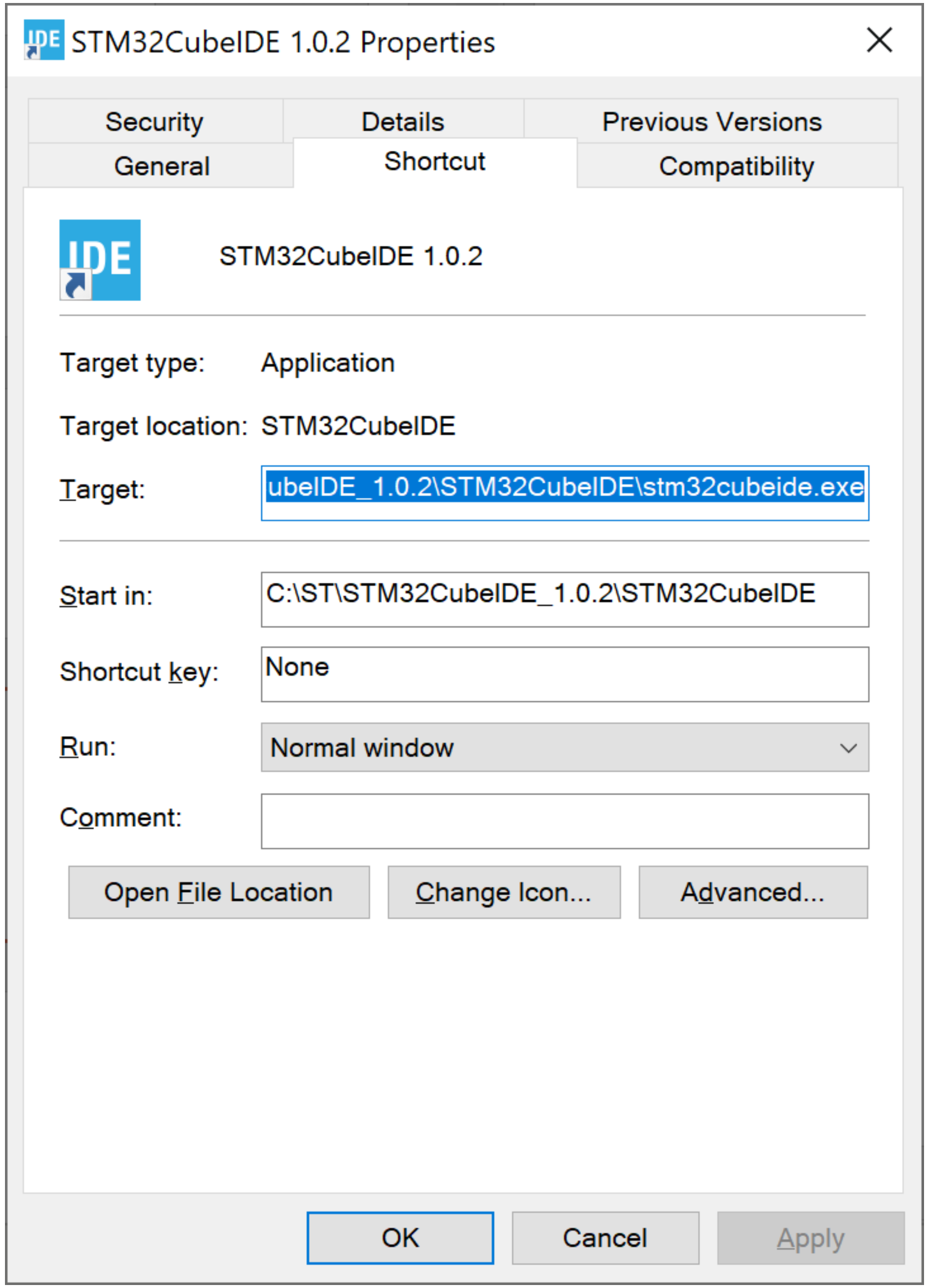


Configure the STM32CubeIDE

- **A brief detour for high DPI monitors**
- Right-click the desktop icon
- Select “Properties” from the context menu

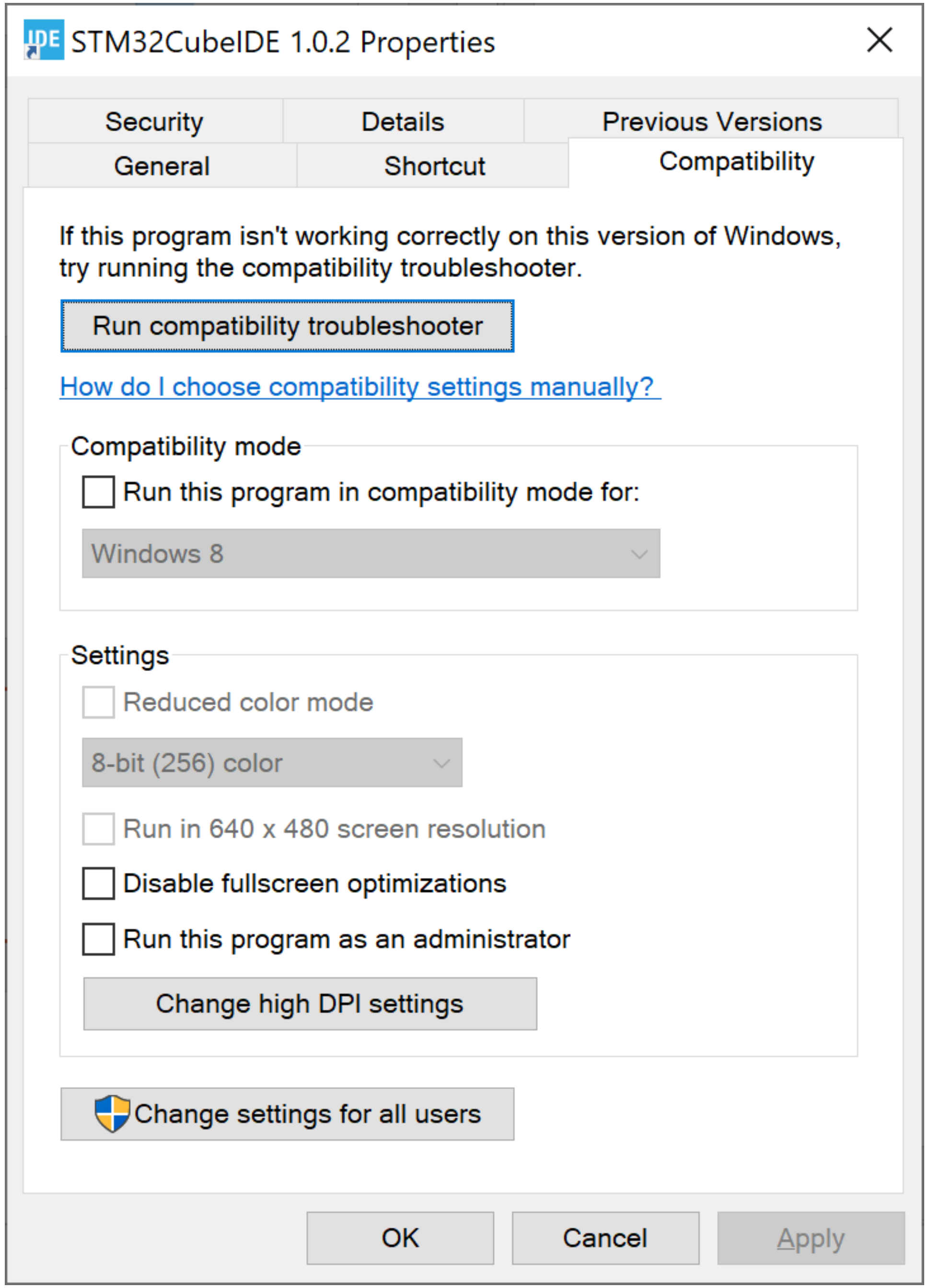
Properties

- Click on the “Compatibility” tab



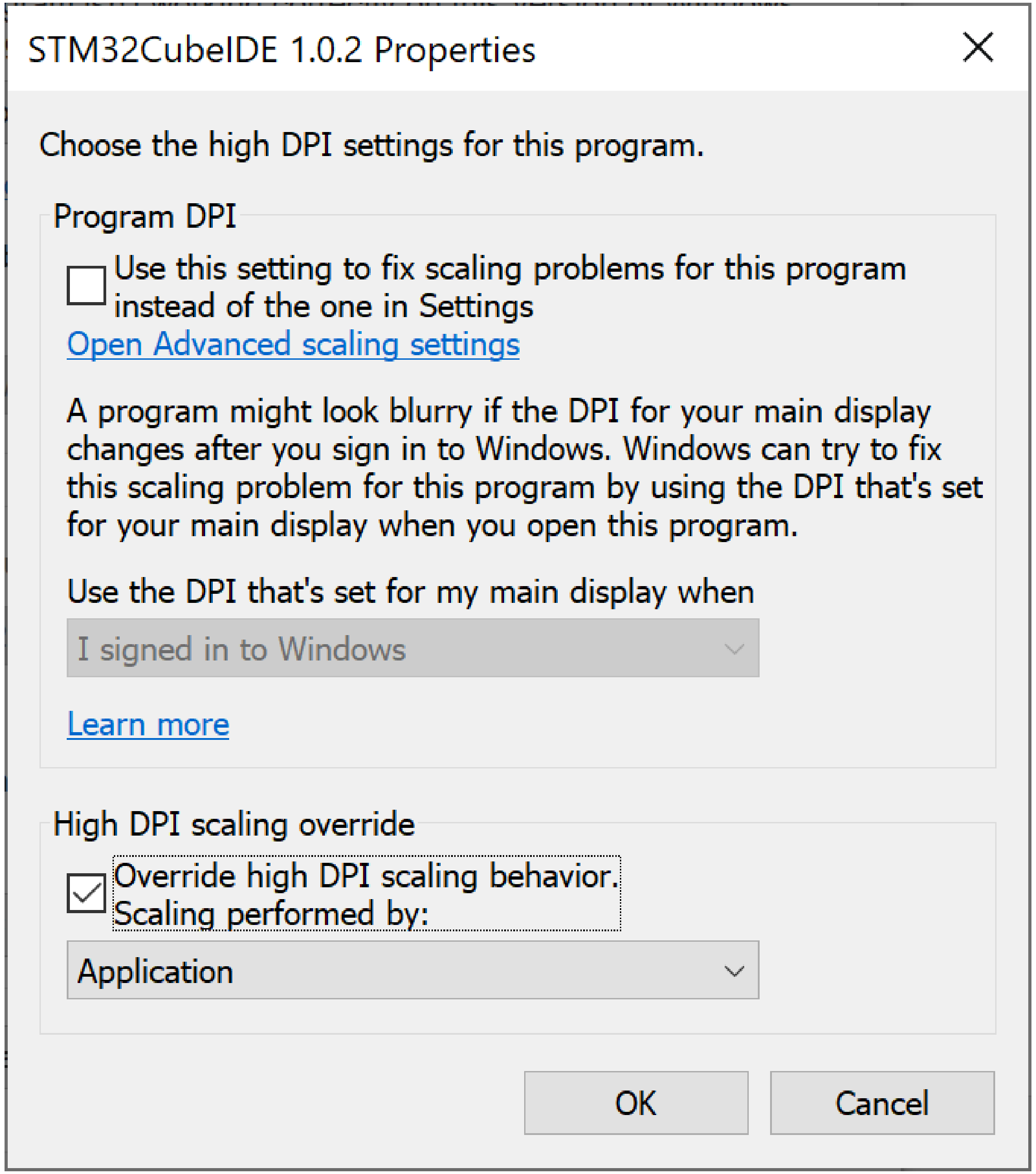
Compatibility

- Click the “Change high DPI settings” button



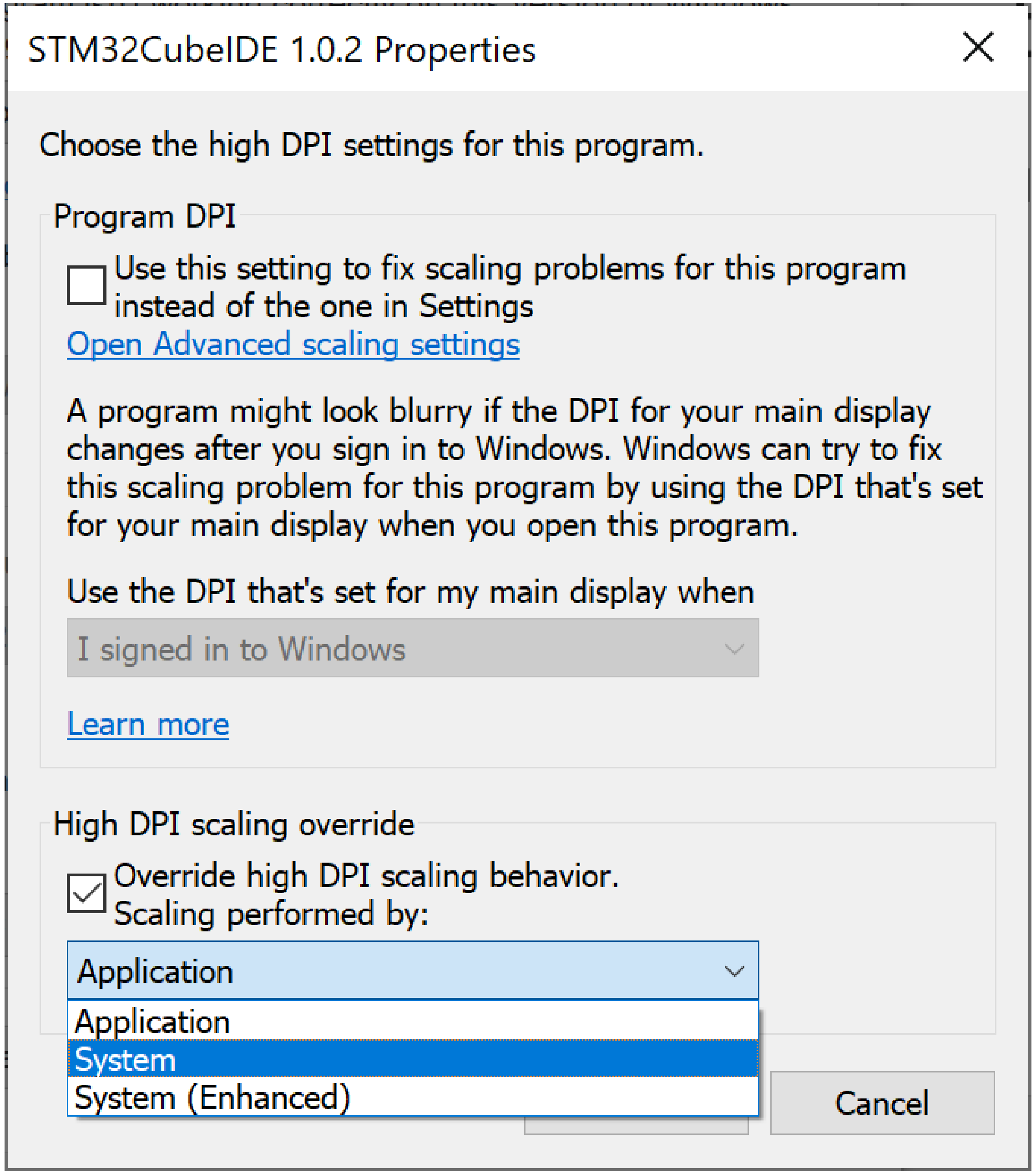
Override DPI

- Check the "Override high DPI scaling behavior" box



Override

- Choose “System” from the drop-down list box.
- Click “OK”
- Click the “OK” button on the “Properties” dialog box.
- <end of high DPI settings detour>

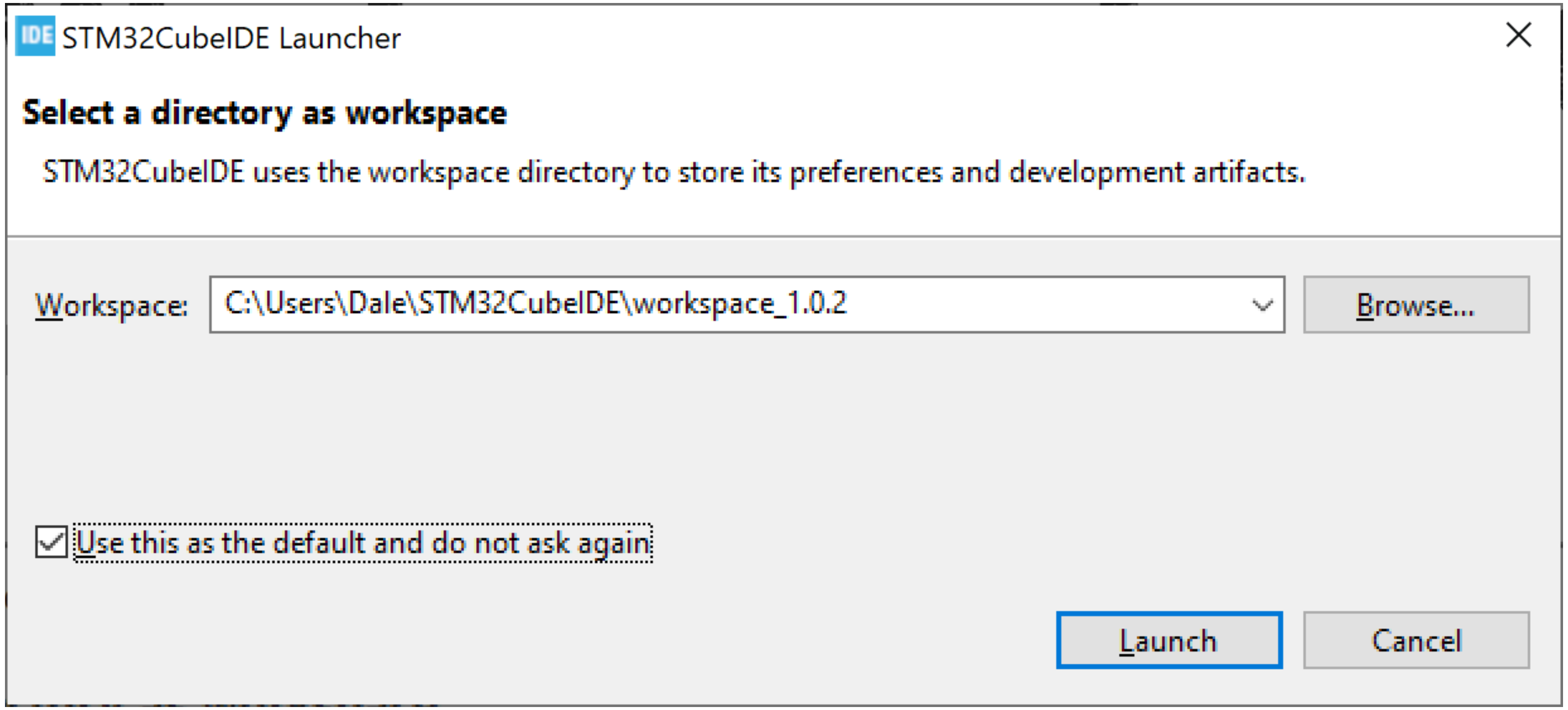


Start the Application

- Double-click on the desktop icon to start the STM32CubeIDE application

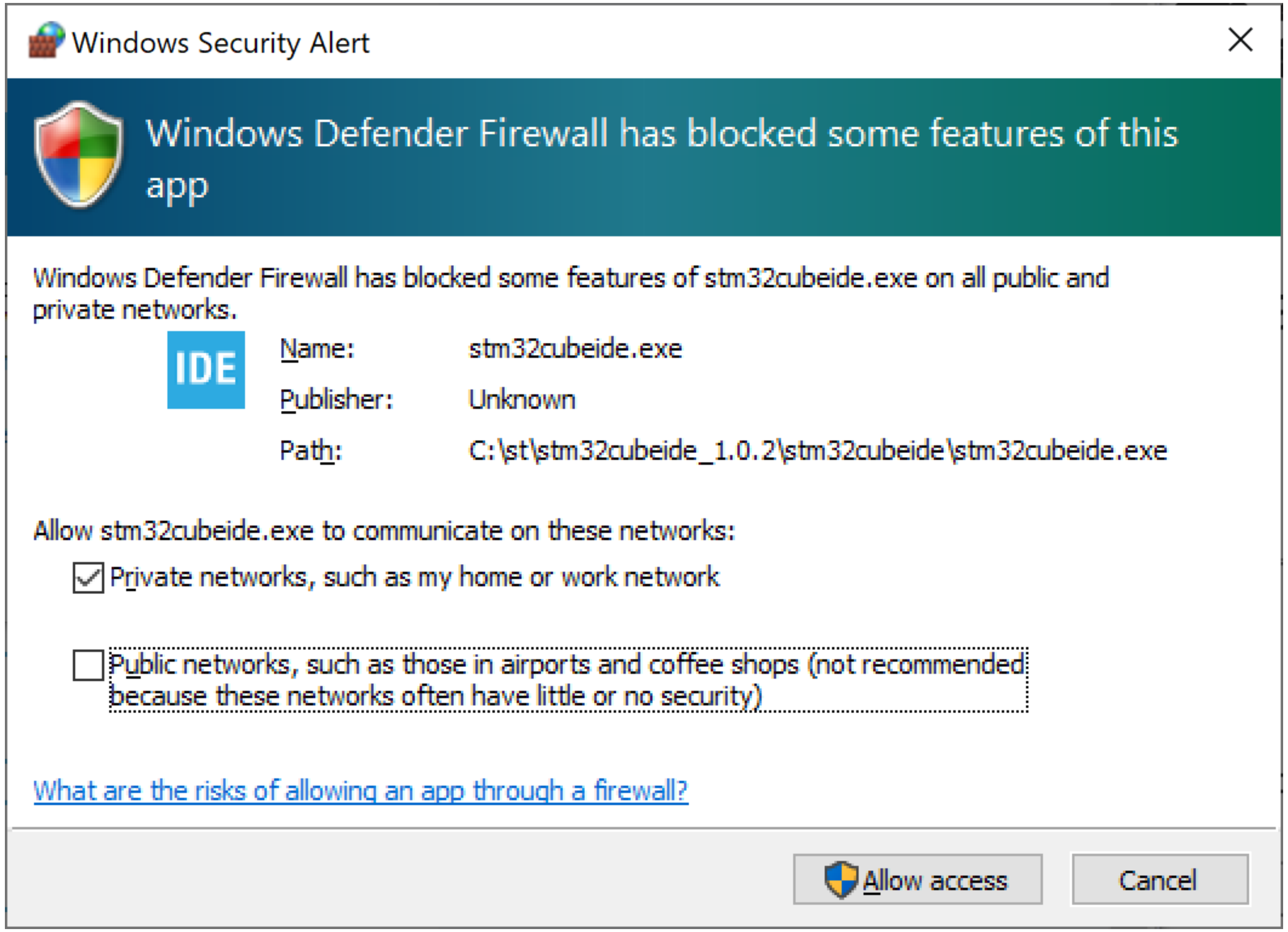
Select Workspace Location

- Check the box labeled "Use this as the default and do not ask again"
- Click "Launch"



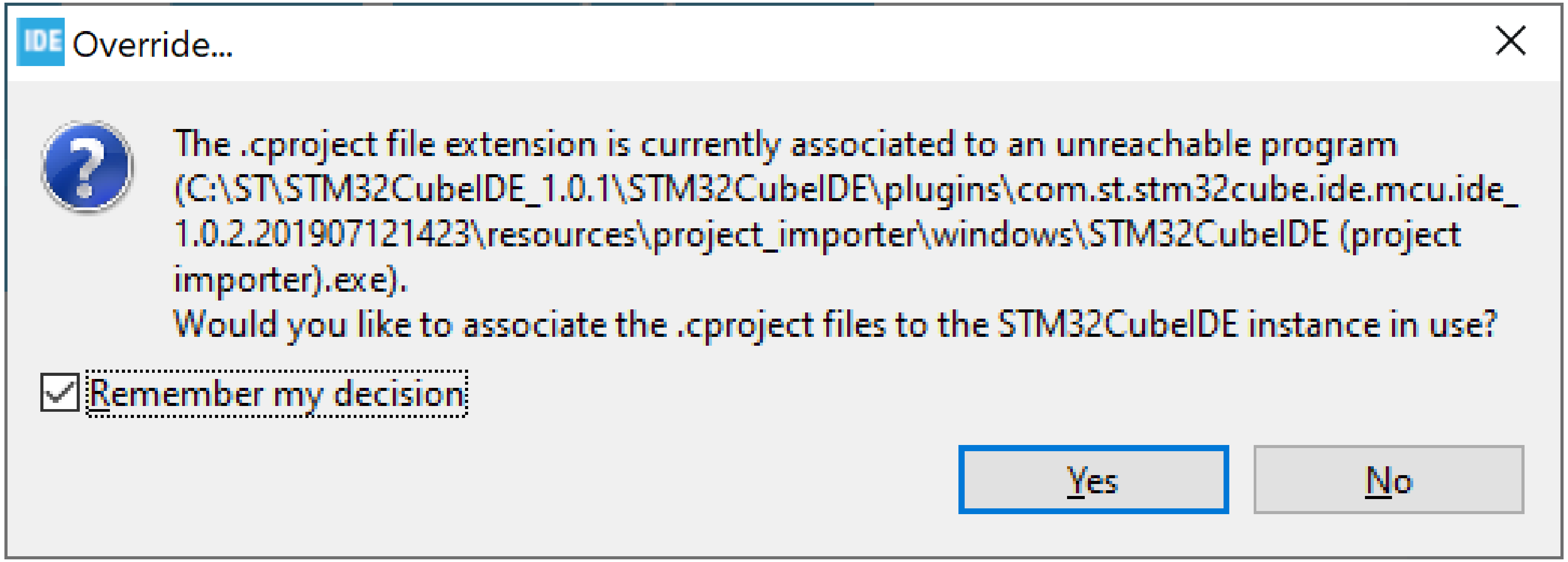
Windows Security Alert

- Adjust as you see fit
- Click the “Allow access” button



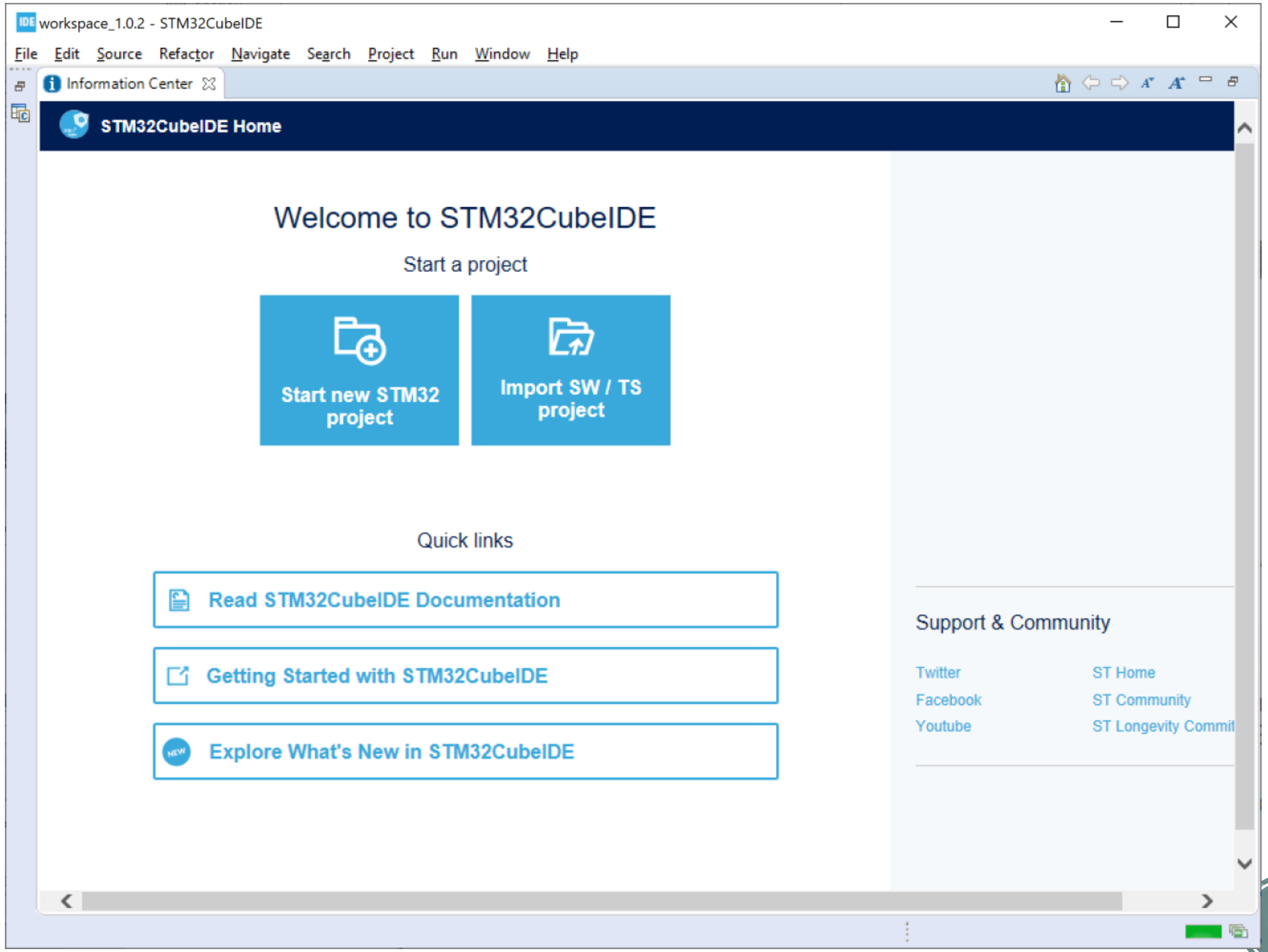
Override File Association

- You might get this message
- Check the “Remember my decision” box
- Click “Yes”



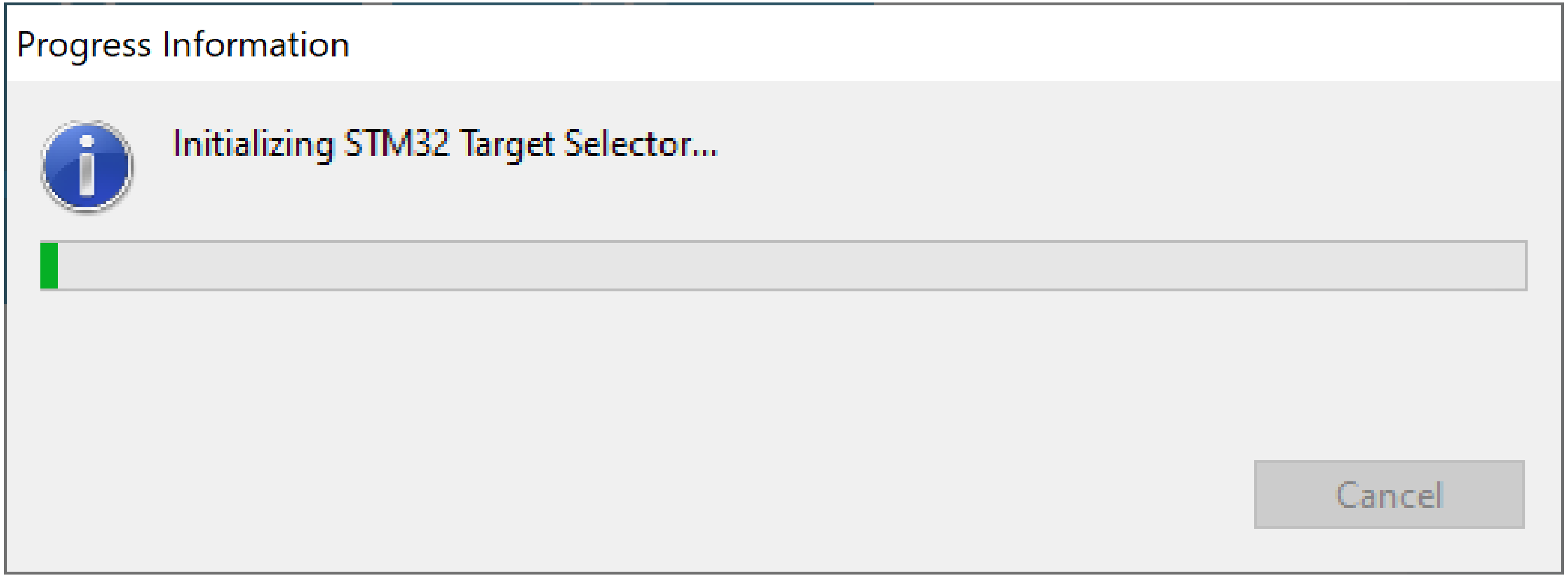
Welcome to STM32CubeIDE

- Maximize the window
- Click “Start New STM32 project”



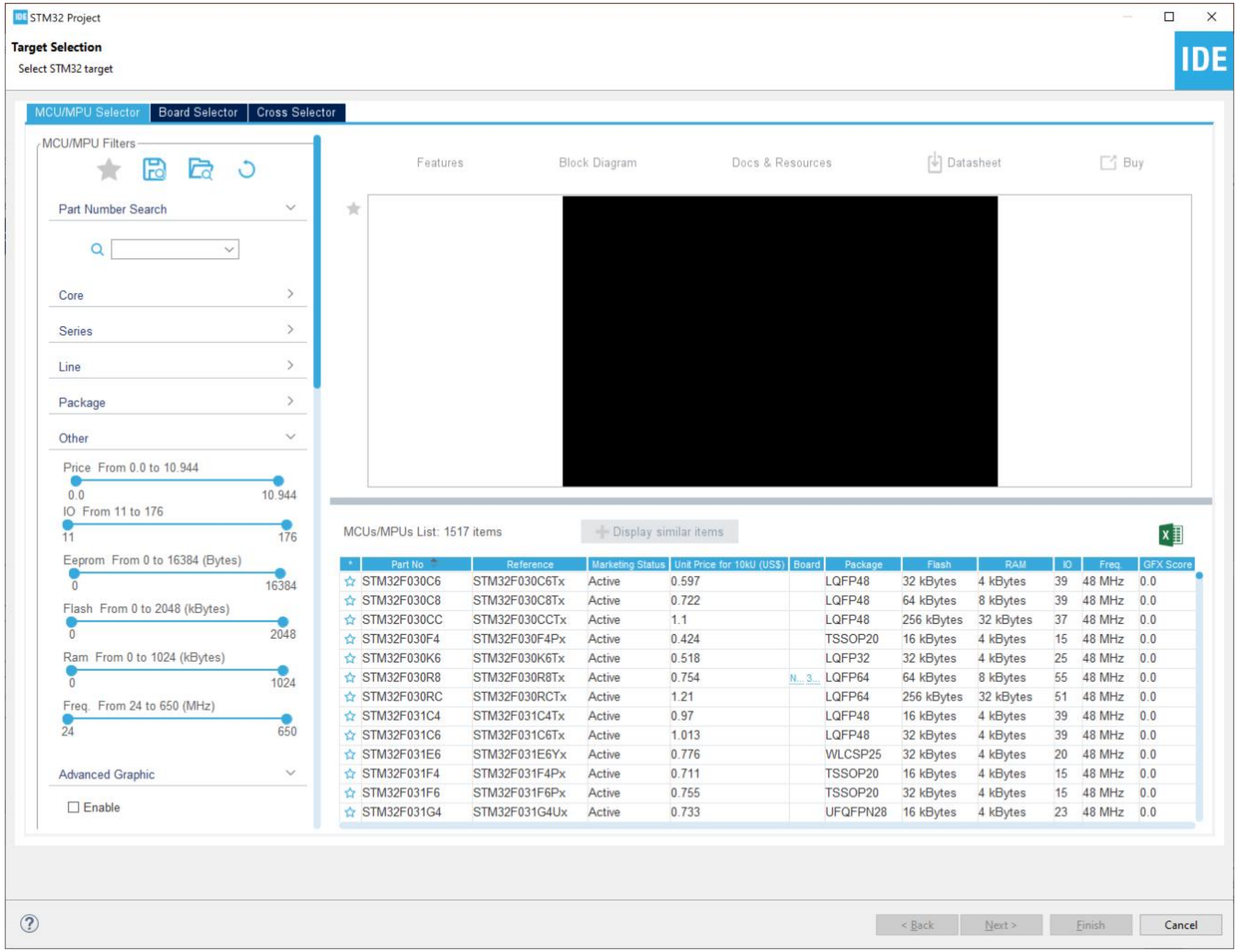
STM32 Target Selector

- Wait for “Target Selection” window to populate
- It can take a while



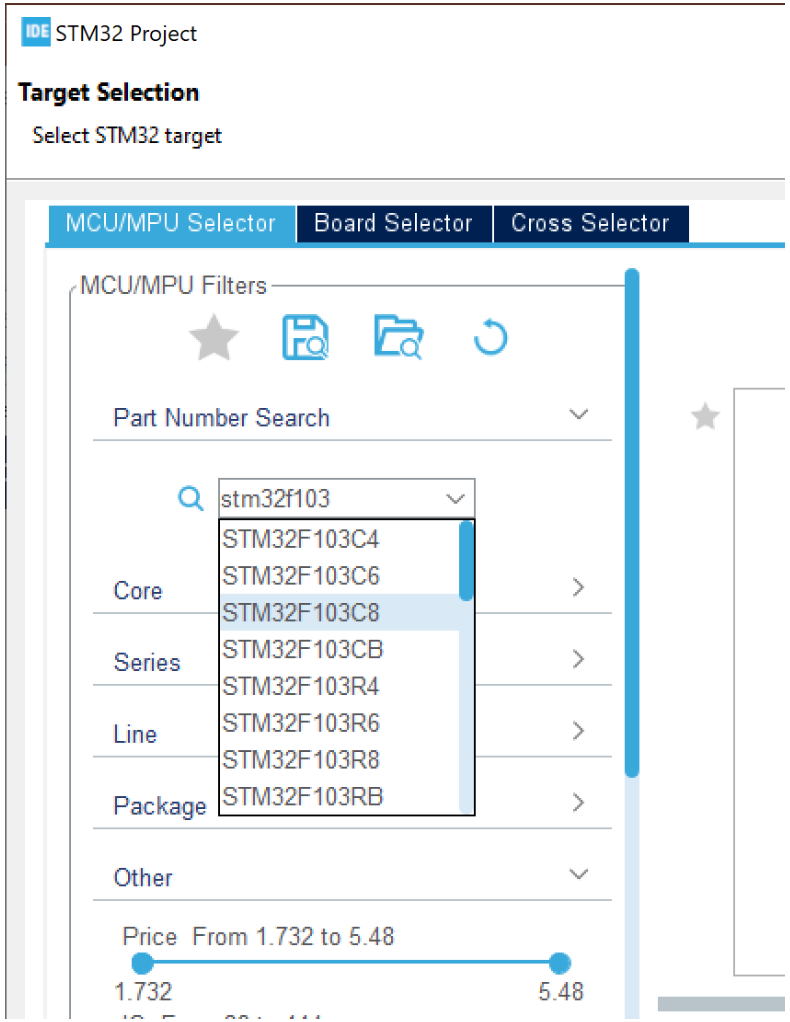
STM32 Target Selector

- In the search field, type “STM32F103”



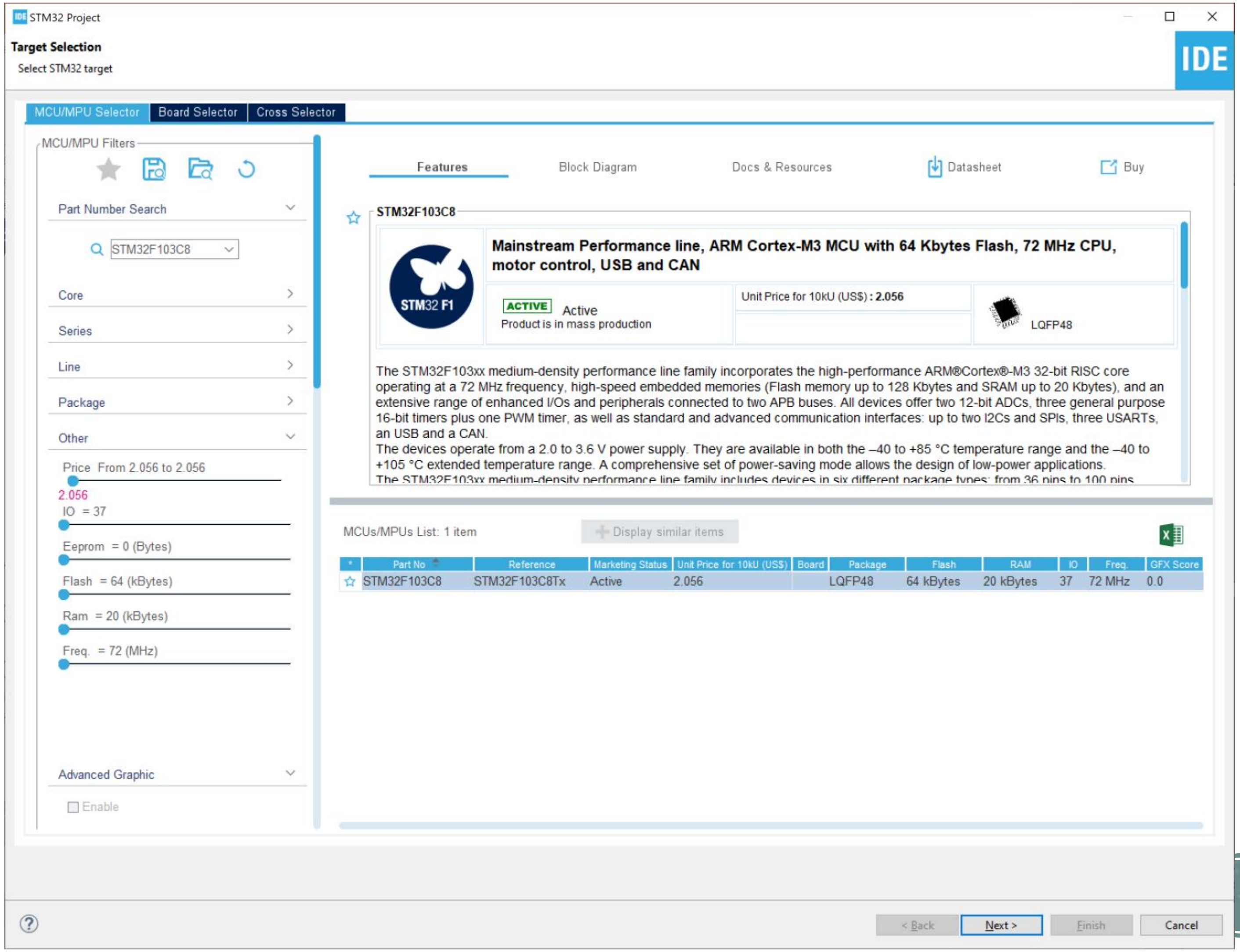
Search for Part Number

- Select “STM32F103C8” from the drop-down list box
- This is the chip on the Blue Pill



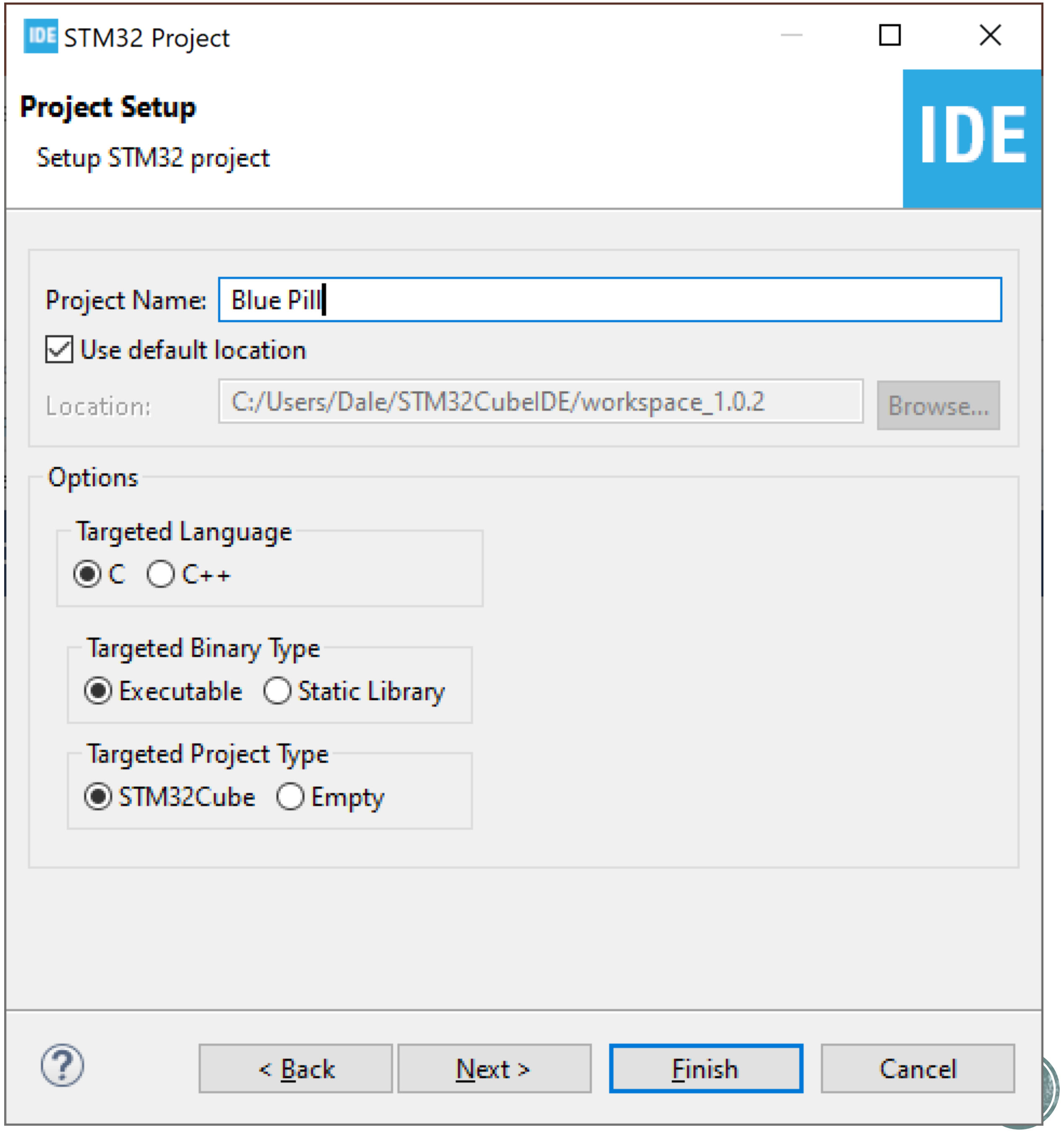
Select Part Number

- Click on the "STM32F103C8" line in the MPU List
- Click "Next >"



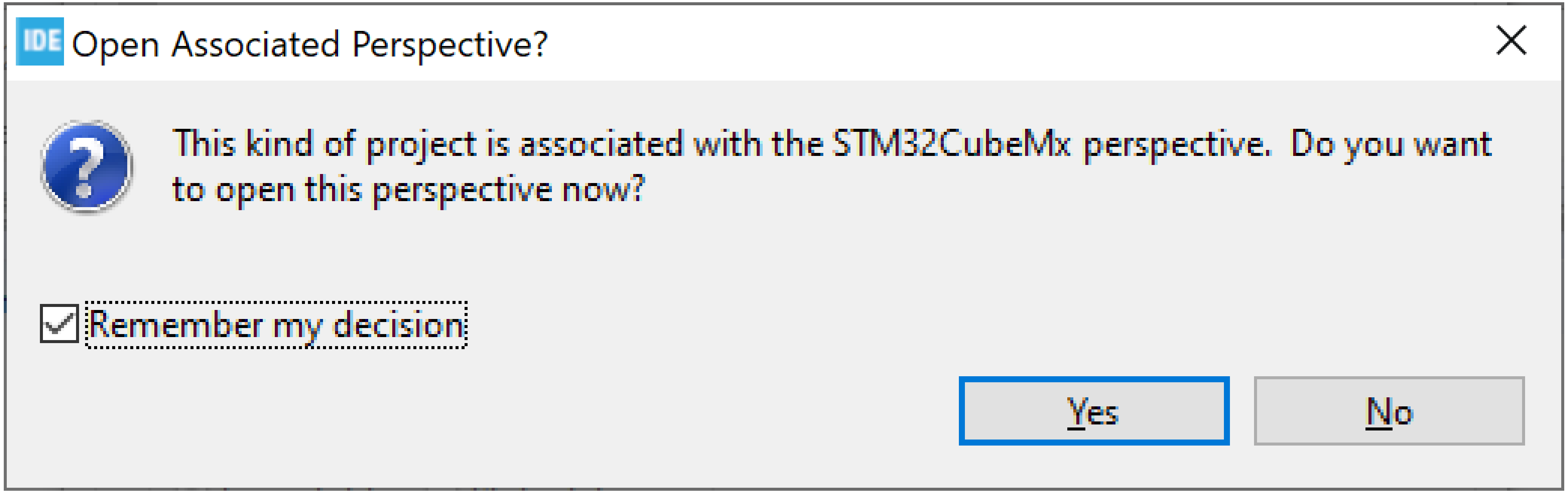
Project Setup

- Type “Blue Pill” in the “Project Name:” field
- We will use this “project” as the basis for other projects
- Click “Finish”



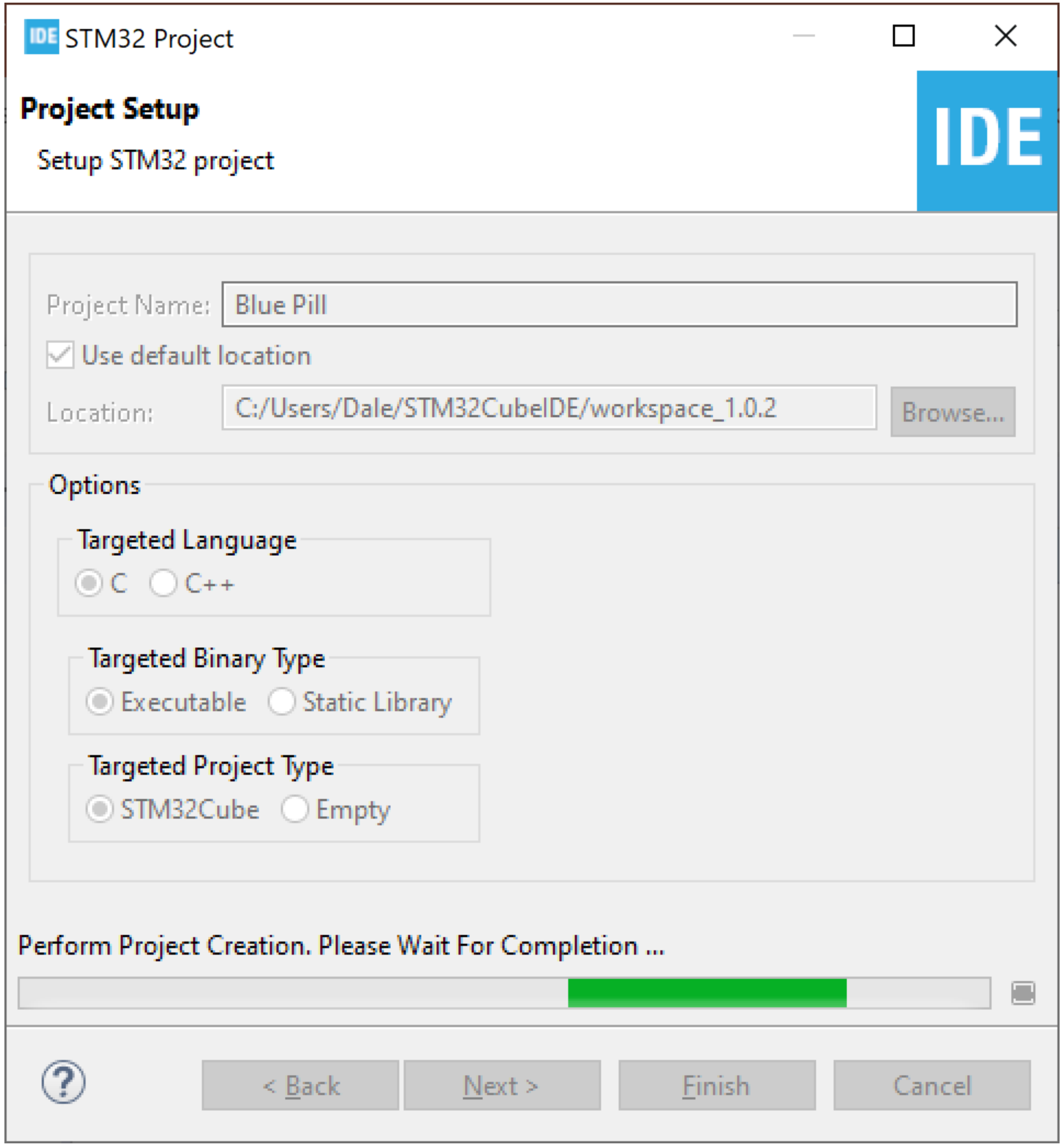
Open STM32CubeMX Perspective

- Check the “Remember my decision” box
- Click “Yes”



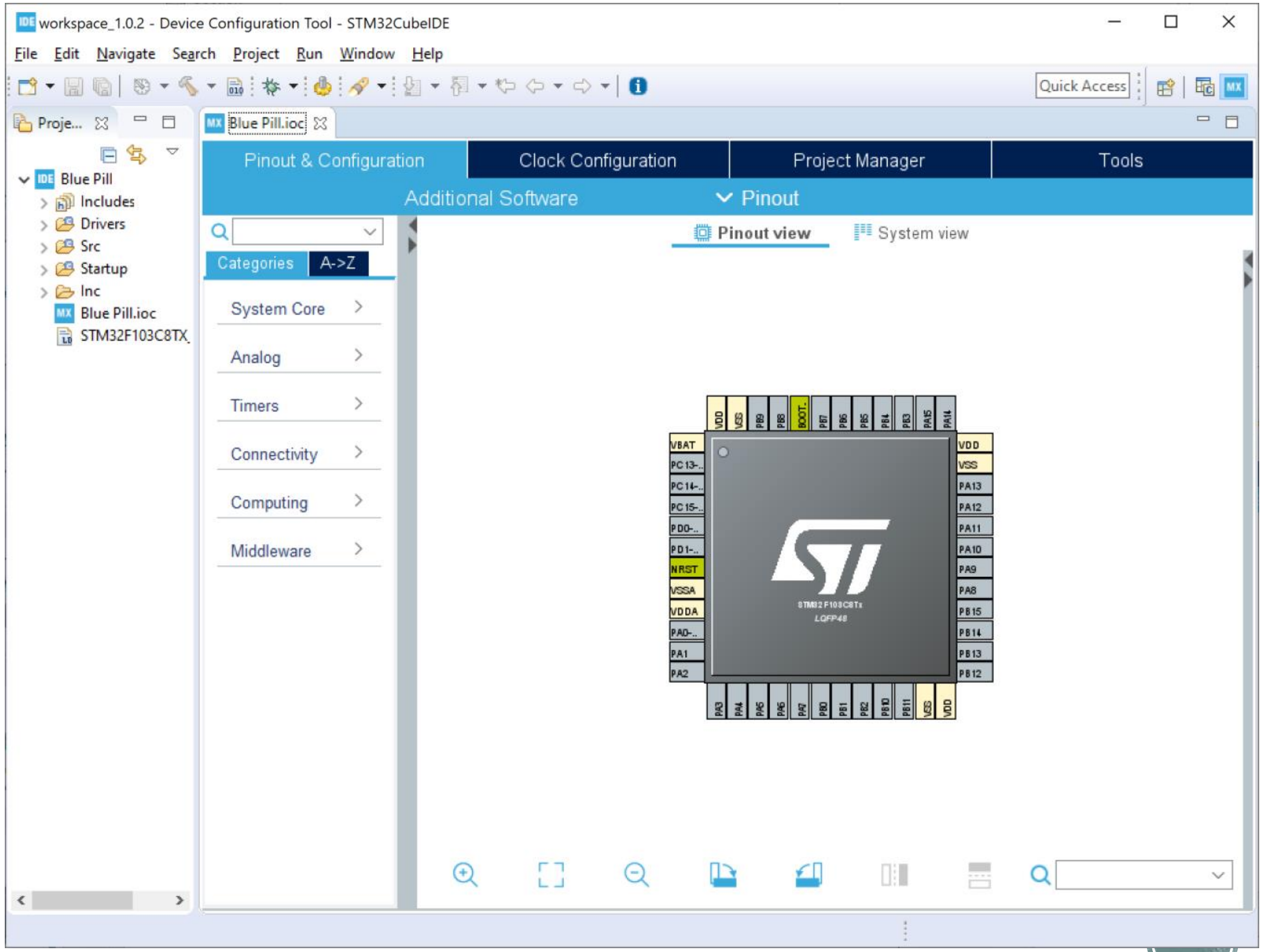
Wait

- Must wait 😐
- It's OK
- It's doing an amazing amount of work for us right now



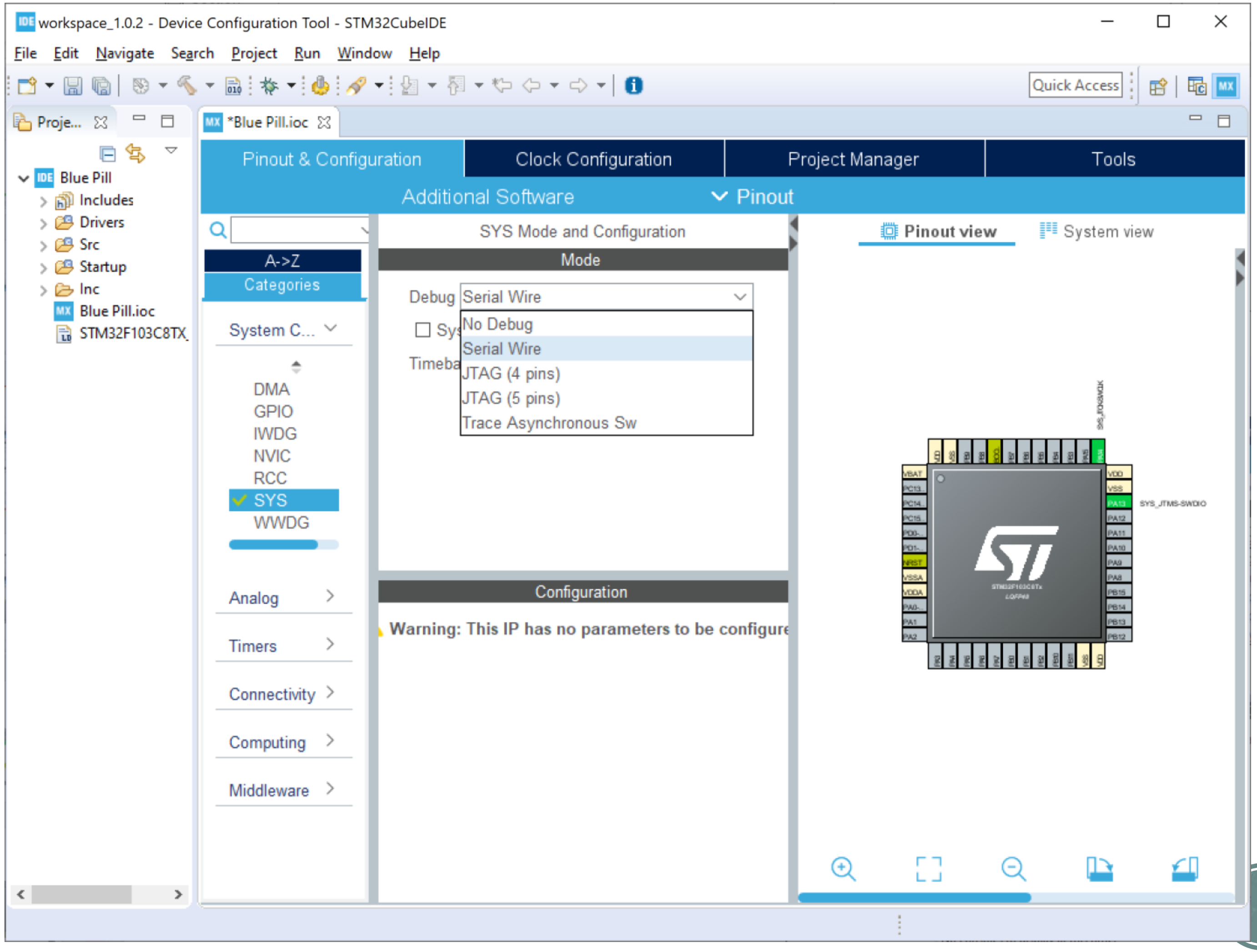
STM32CubeMX

- STM32CubeMX is a device configuration tool
- This is the main STM32CubeMX screen, showing the “Pinout & Configuration” tab
- Click on the “System Core” category
- Select the “SYS” item
- This brings up the “SYS Mode and Configuration” panel



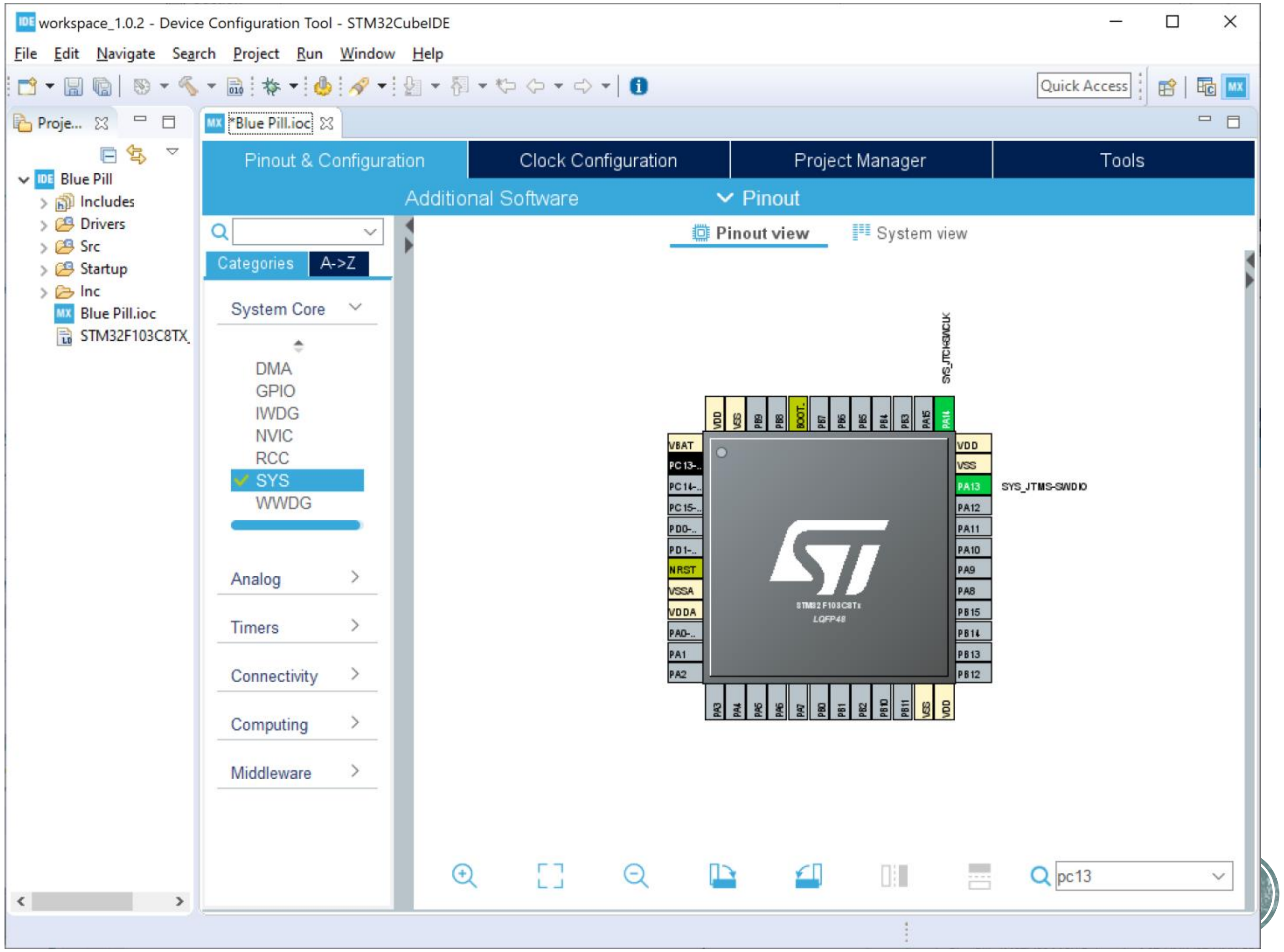
Configure Debug

- From the “Debug” drop-down list, select “Serial Wire”
- Note that some of the pins in the “Pinout view” tab have changed
- We will use Serial Wire Debug (SWD) to program the chip
- Close the “SYS Mode and Configuration” panel by clicking on the left arrow (right under the “v Pinout” tab)



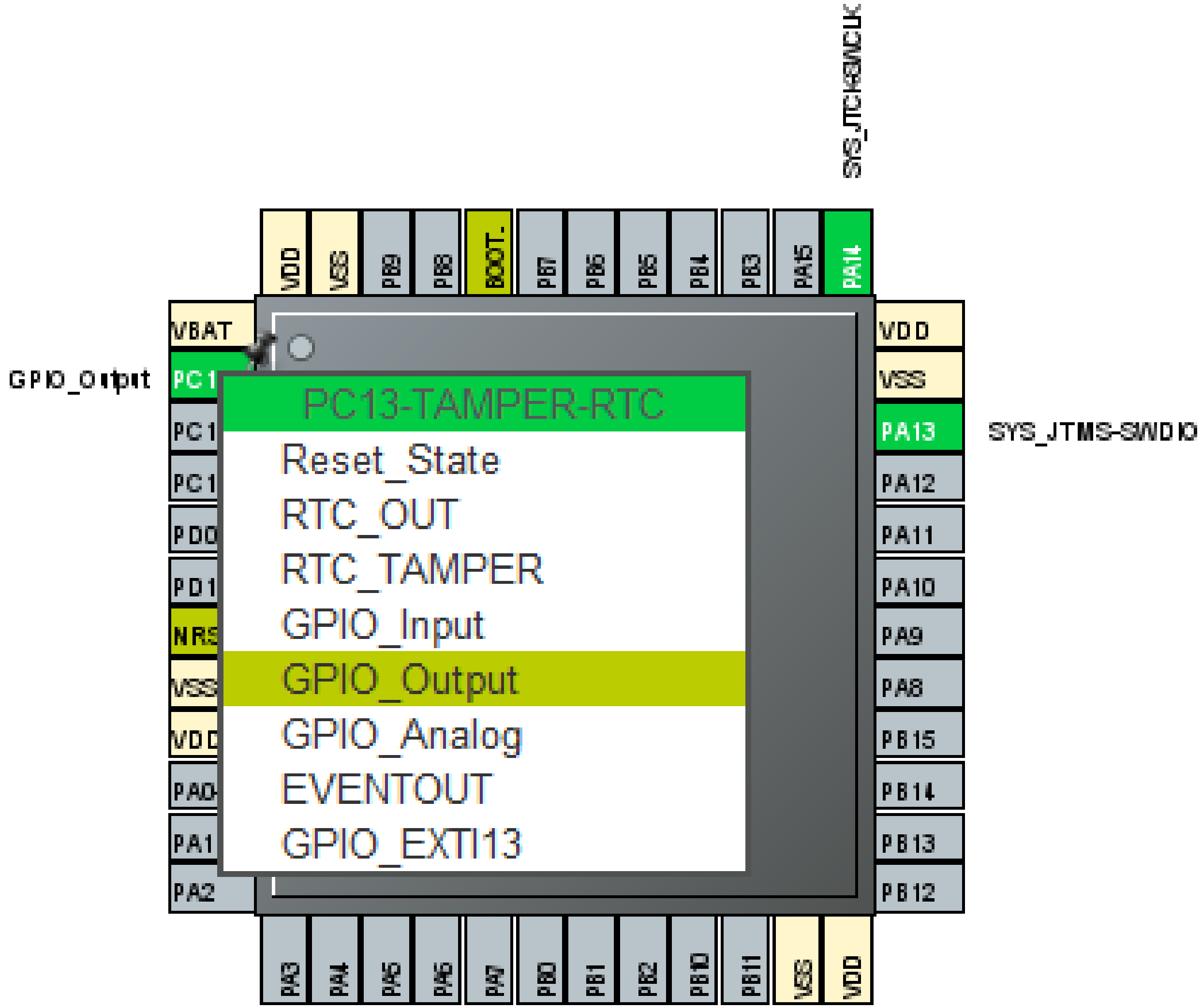
Find LED Driver Pin (PC13)

- The Blue Pill has an LED connected to pin PC13
- “PC13” means Port C, pin 13
- Port C is a GPIO pin, or “General Purpose Input/Output”
- Type “pc13” into the search field in the Pinout view
- Note that the PC13 pin begins to flash on the chip diagram



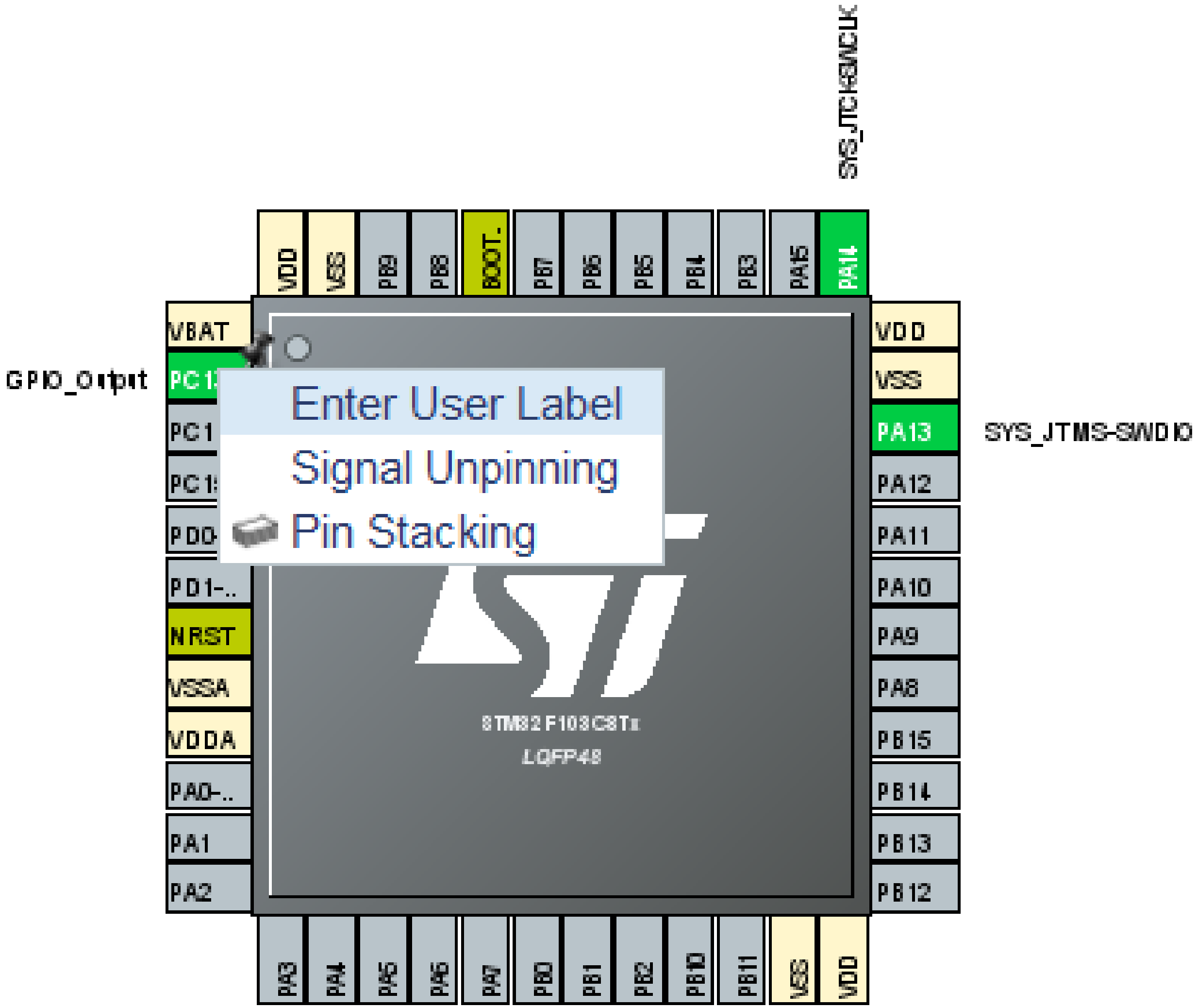
Configure PC13 as Output Pin

- Click on the PC13 pin on the diagram
- Select "GPIO_Output"
  in the context menu
- This will tell the STM32CubeMX
  code generator to initialize this pin
  as a "general purpose output",
  which is what we need to flash
  the LED that is connected to it
- Note that the PC13 pin
  now has a green background,
  indicating that it is now
  properly configured



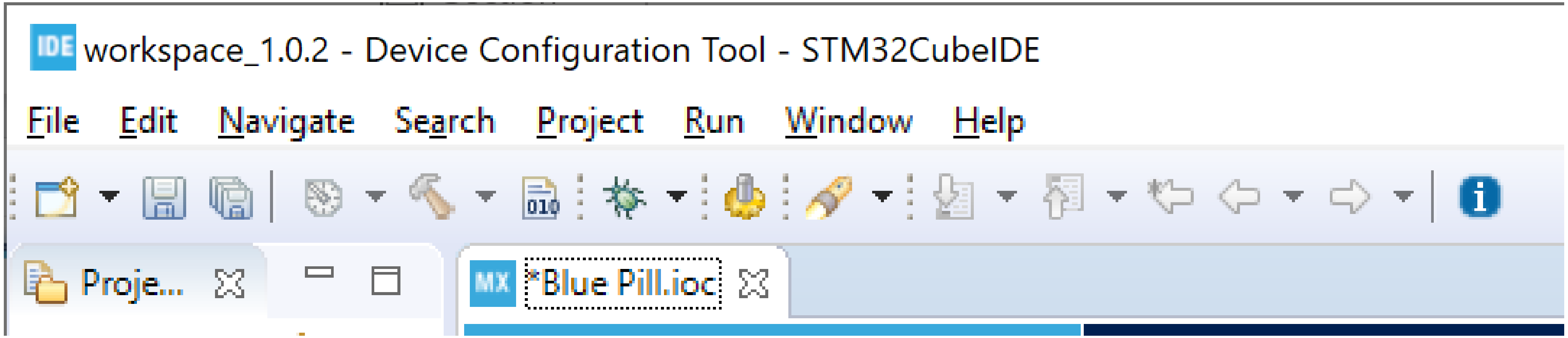
Assign Label to Pin

- Right-click the PC13 pin and select “Enter User label” from the context menu
- Enter “LED” and press [ENTER]
- Note the label next to PC13 has changed to “LED”
- This name will also be reflected in the generated code

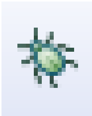


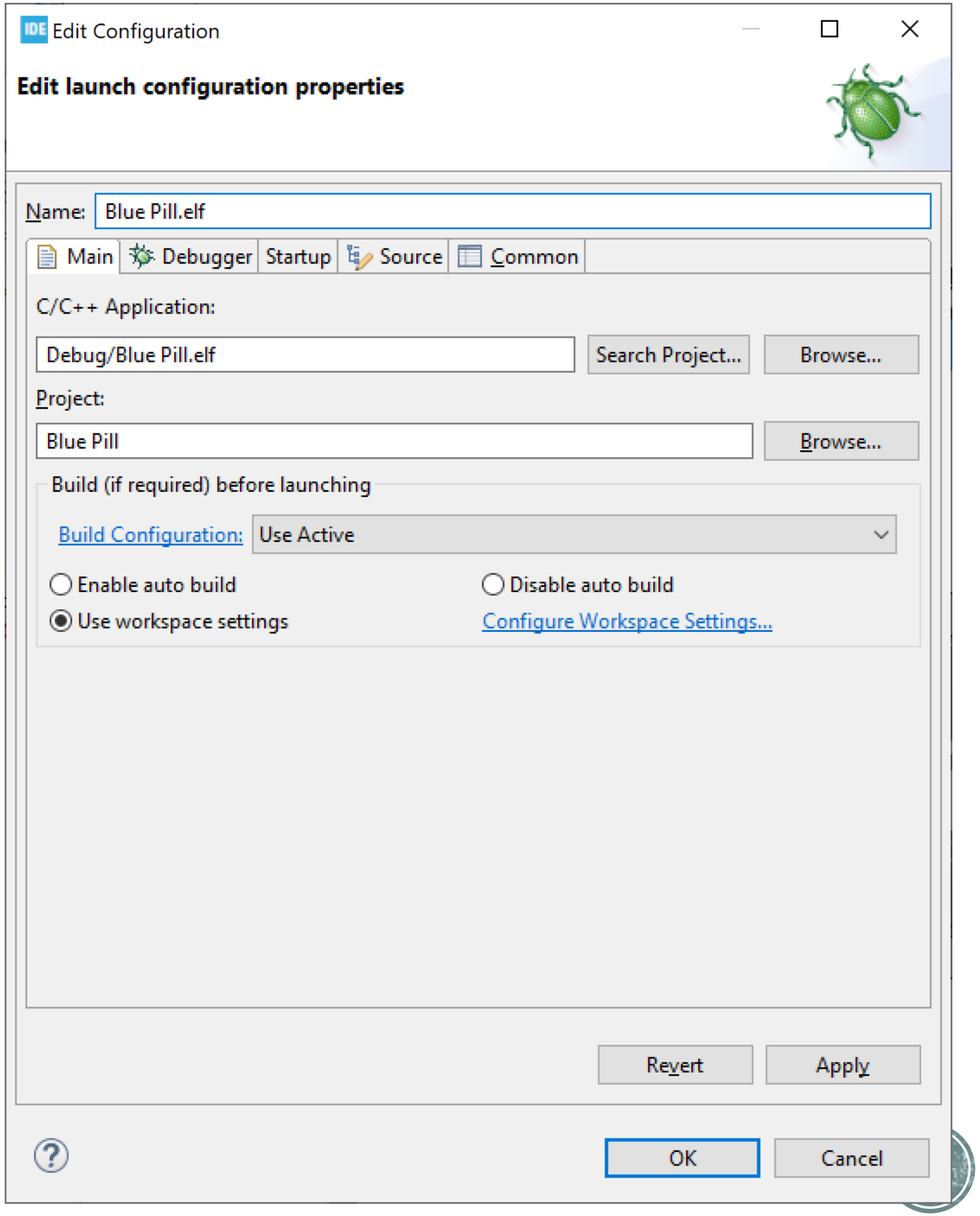
Generate Code

- Click the “Device Configuration Tool Code Generation” toolbar icon (gear icon)
- The STM32Cube MX application now writes all the code for the project
- Now plug the ST-LINK device programmer into your laptop
- The power LED on your Blue Pill should light up
- Another LED might be on or blinking as well



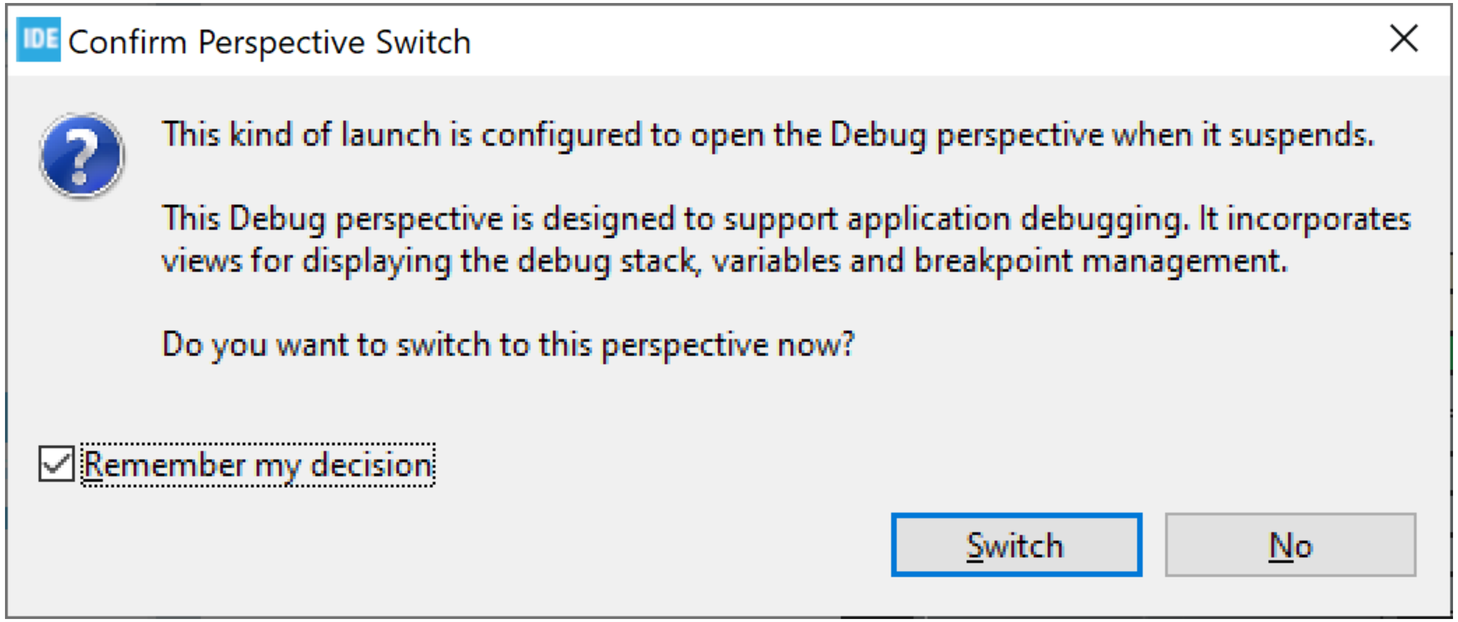
Compile+Debug

- Click the “Debug” toolbar icon 
- The “Edit launch configuration properties” dialog appears
- Click “OK”
- You might get another “Windows Security Alert”
- Click “Allow access”



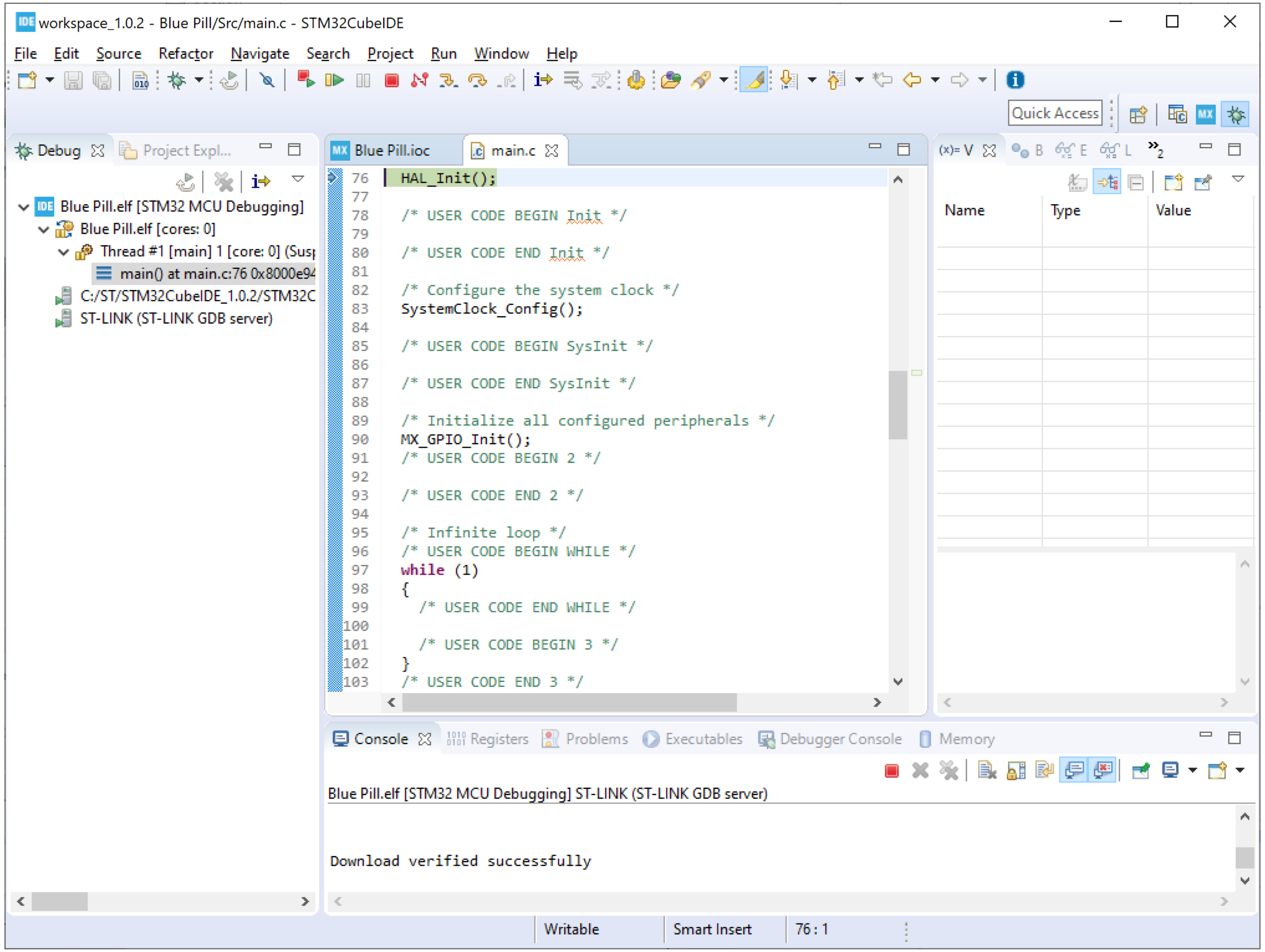
Switch to Debug Perspective

- The “Confirm Perspective Switch” dialog appears
- Check “Remember my decision”
- Click “Switch”



Ready to Debug

- Now we see some code!

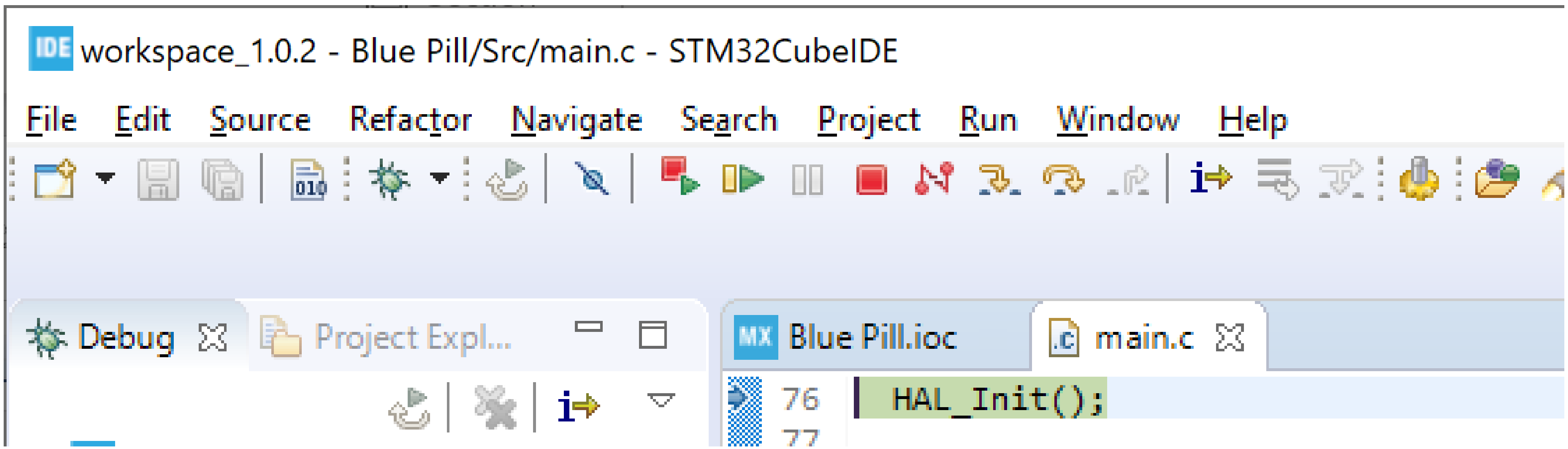


Review

- If all goes well, the application compiled the generated code, downloaded it to the device and started a debug session. The program is now halted at the first executable line in the **main()** function and awaits your command.
- Notice that the line 76 has a green background. That is the next line of code to be executed. It should be a call to the **HAL_Init()** function, to initialize the Hardware Abstraction Layer (HAL).

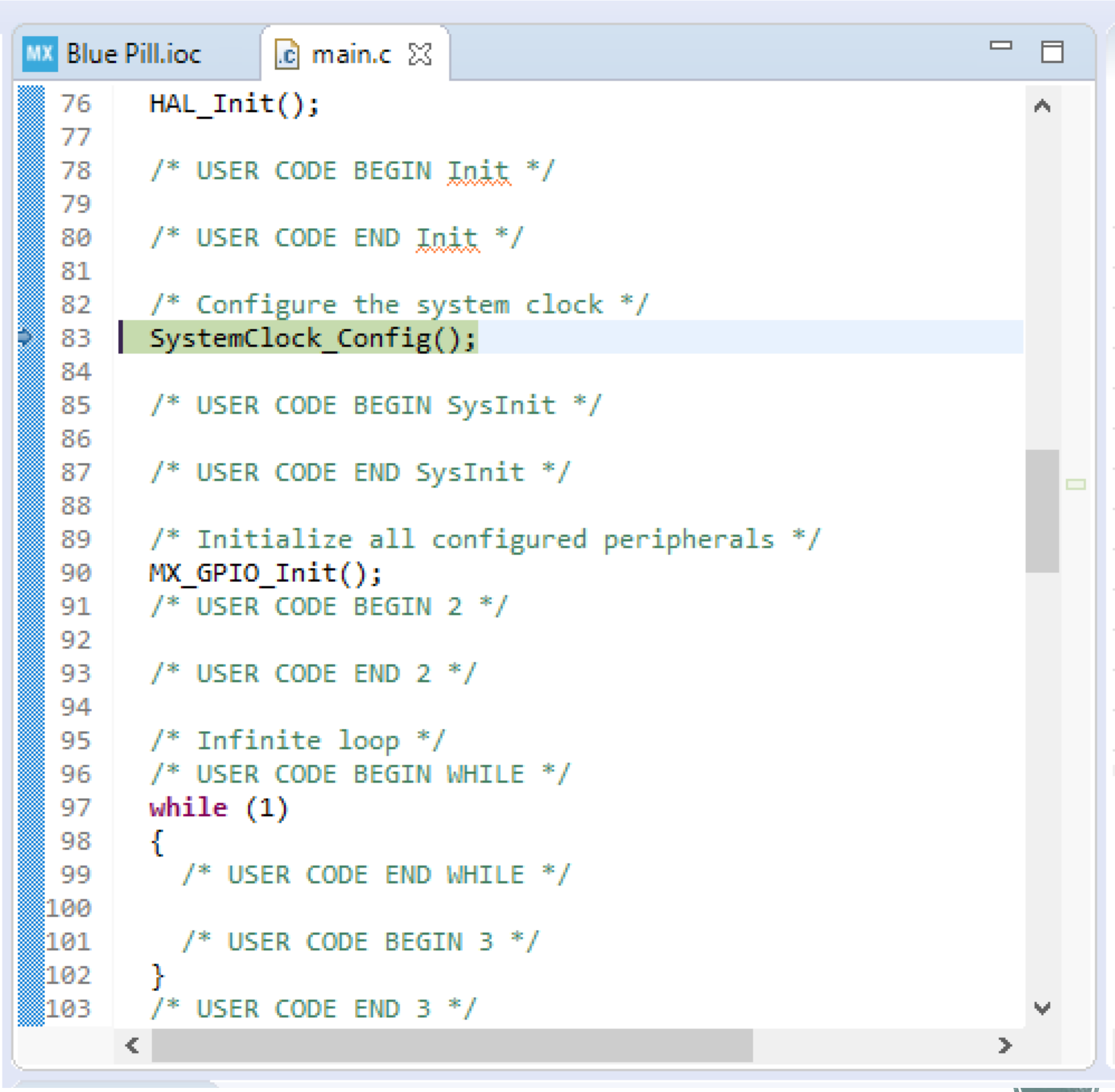
Step Over

- Click the “Step Over (F6)” toolbar icon



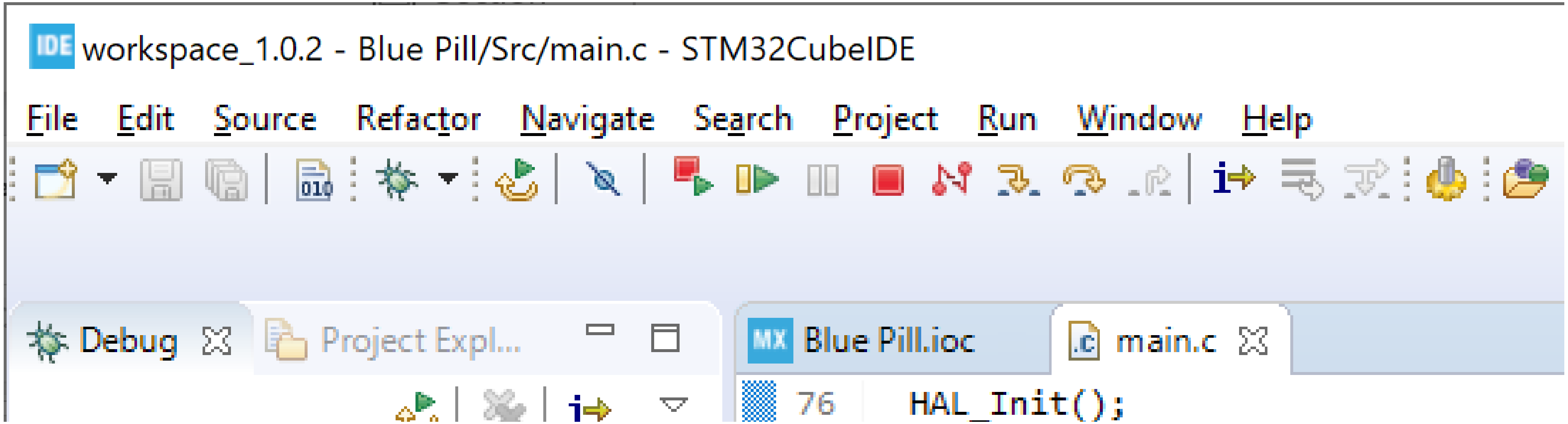
Confirm Step

- The green background should now highlight line 83, a call to the **SystemClock_Config()** function
- This proves that the toolchain is working and properly configured ☺



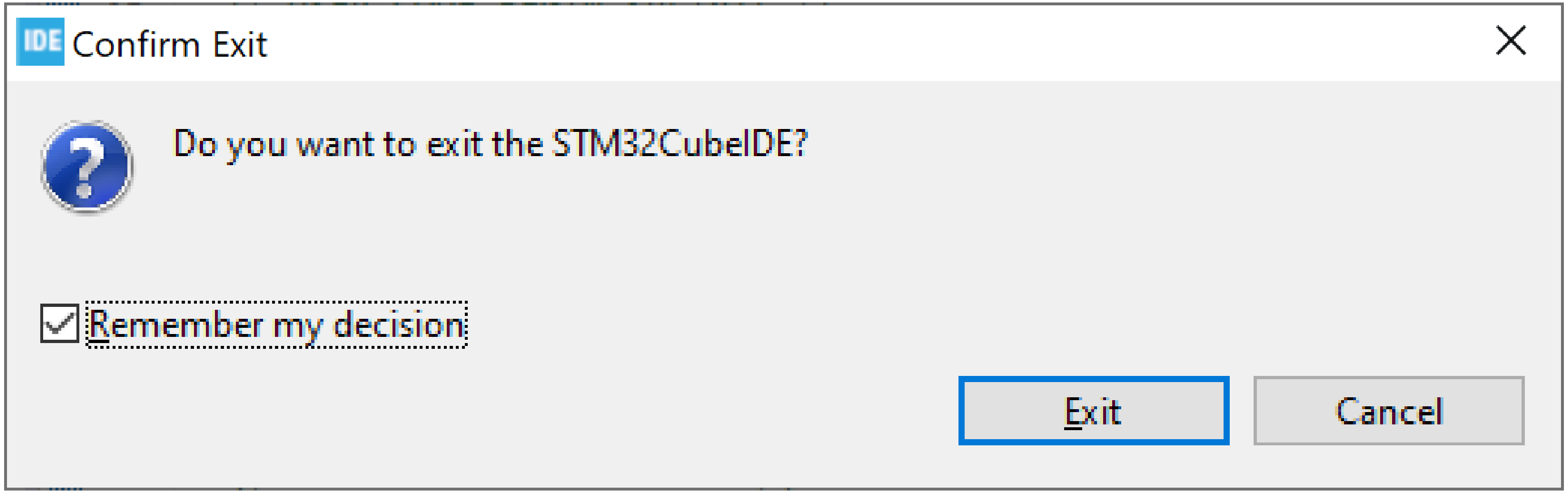
Stop Debugging

- Click the “Terminate (Ctrl+F2)” toolbar icon (red square)
- This stops the debug session



Close the STM32CubeIDE

- Close the STM32CubeIDE application
- Check the “Remember my decision” box
- Click “Exit”



Congratulations!

- Your STM32CubeIDE software is installed and configured correctly
- We now take a short break ☺

Embedded “Hello, world!”

- There is no console to which we can “print” anything… yet
- We will blink the built-in LED on GPIO pin PC13
- No jumpers are required for this experiment
- Connect ST-LINK/V2 device to USB port (if it is not already still connected)
- Observe fast blink rate of PC13 (~5 Hz) – but only on a brand-new Blue Pill

Minimum Code for LED blink

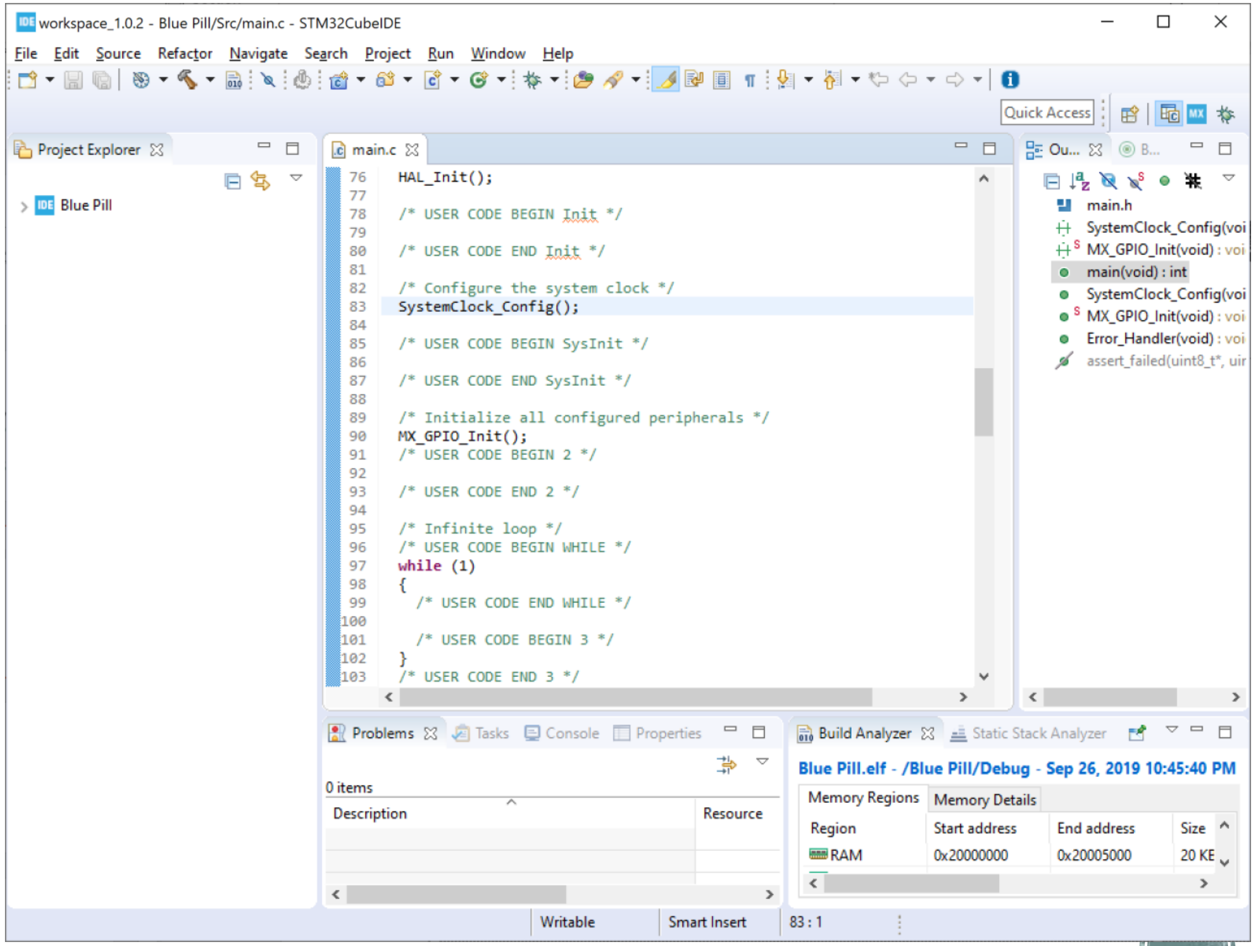
- This is the absolute minimum code needed to blink an LED
- It is quite cryptic at first glance
- You need to have access to the 1,000+ page data sheet to find all the information

```
RCC->APB2ENR = RCC_APB2ENR_IOPCEN; // enable GPIO port C
GPIOC->CRH = GPIO_CRH_MODE13; // PC13 = output, 50 MHz

while(1) {
	GPIOC->BSRR = GPIO_BSRR_BS13; // LED off
	GPIOC->BRR = GPIO_BRR_BR13; // LED on
}
```

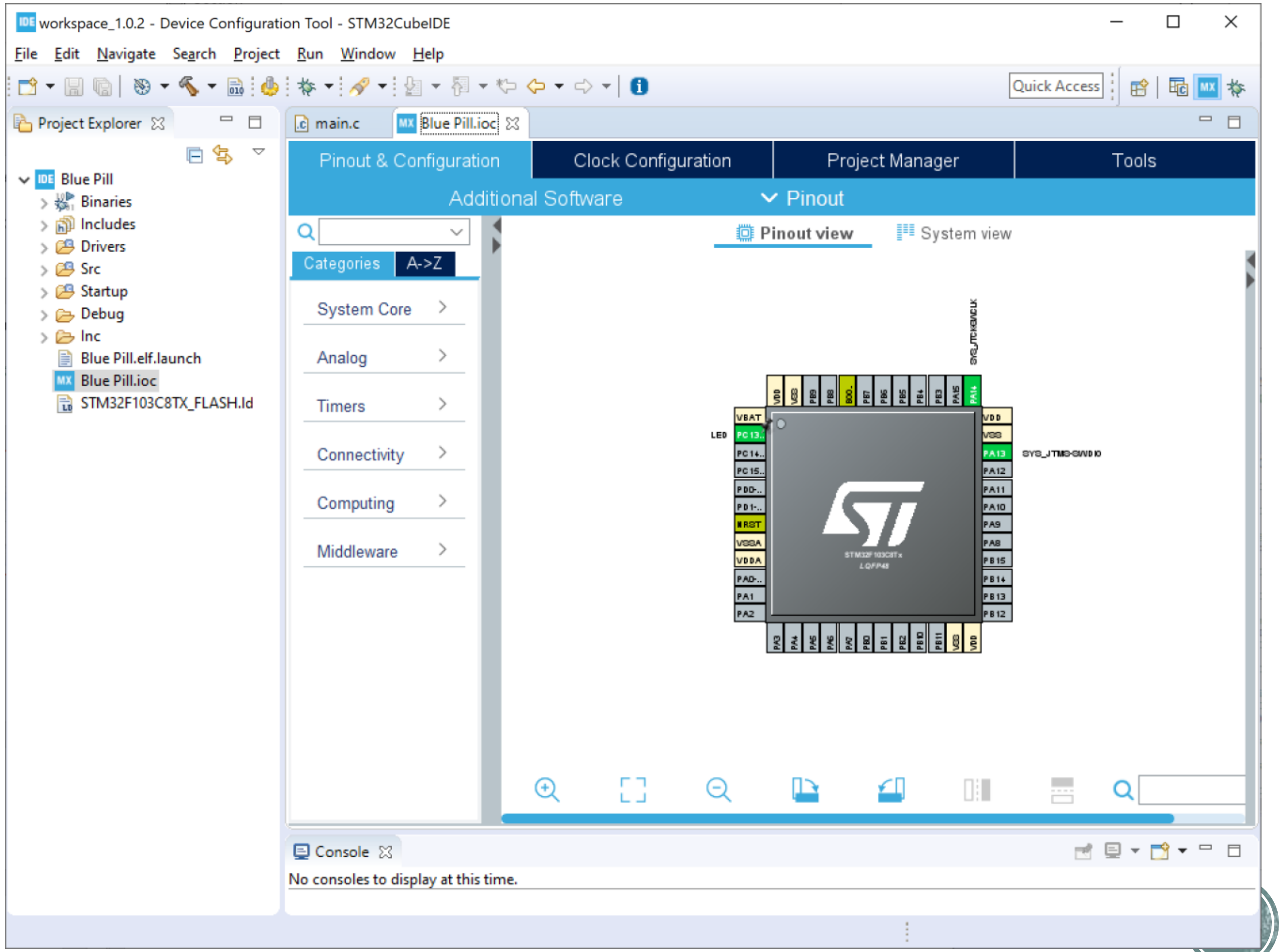
Open STM32CubeIDE

- Restart the STM32CubeIDE application
- It remembers where we left off



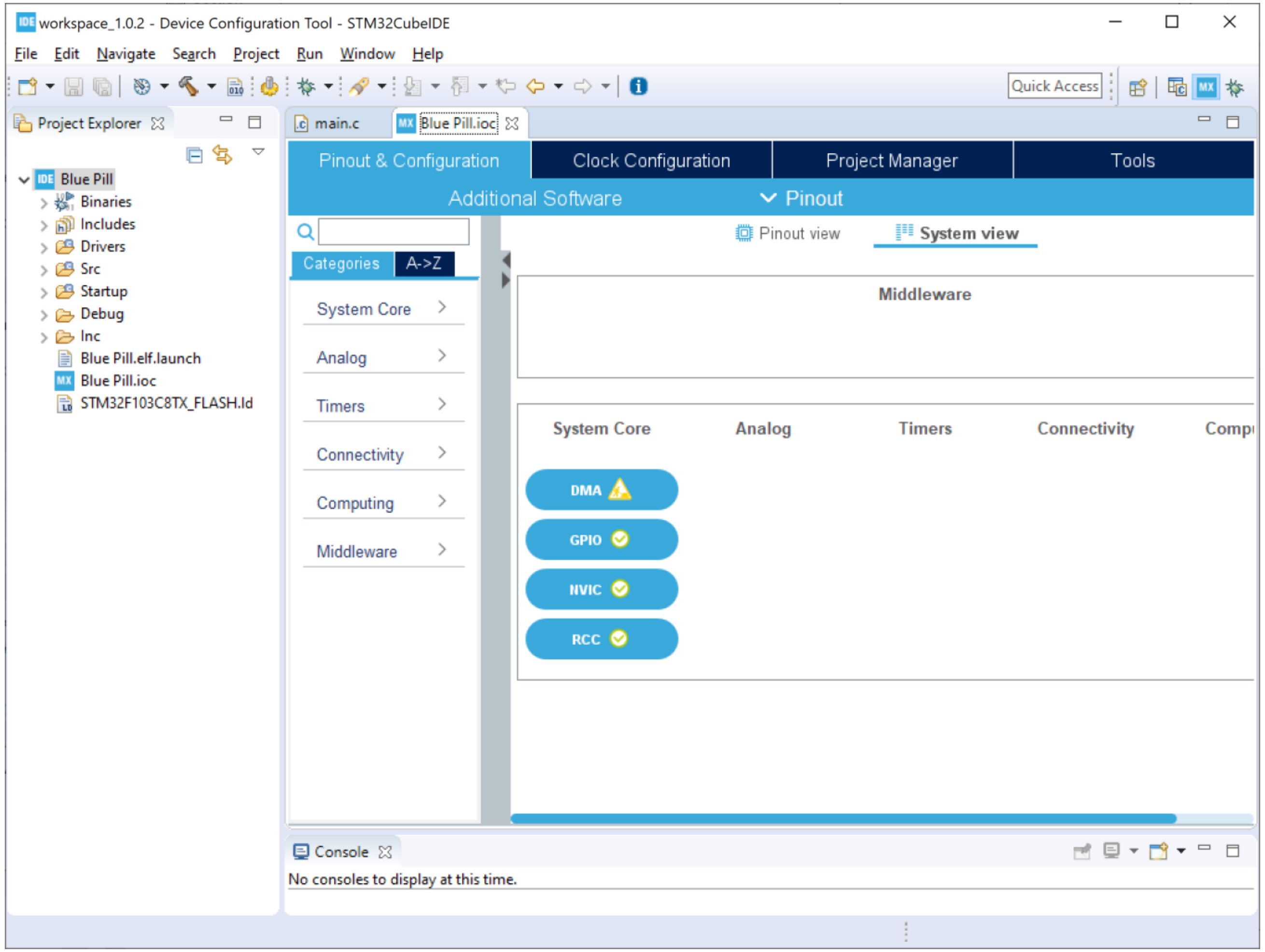
Expand "Blue Pill" Project

- Double-click on the "Blue Pill" project in the Project Explorer panel
- Double-click on the "**Blue Pill.ioc**" file
- This opens the CubeMX perspective
- Let's make one small change here



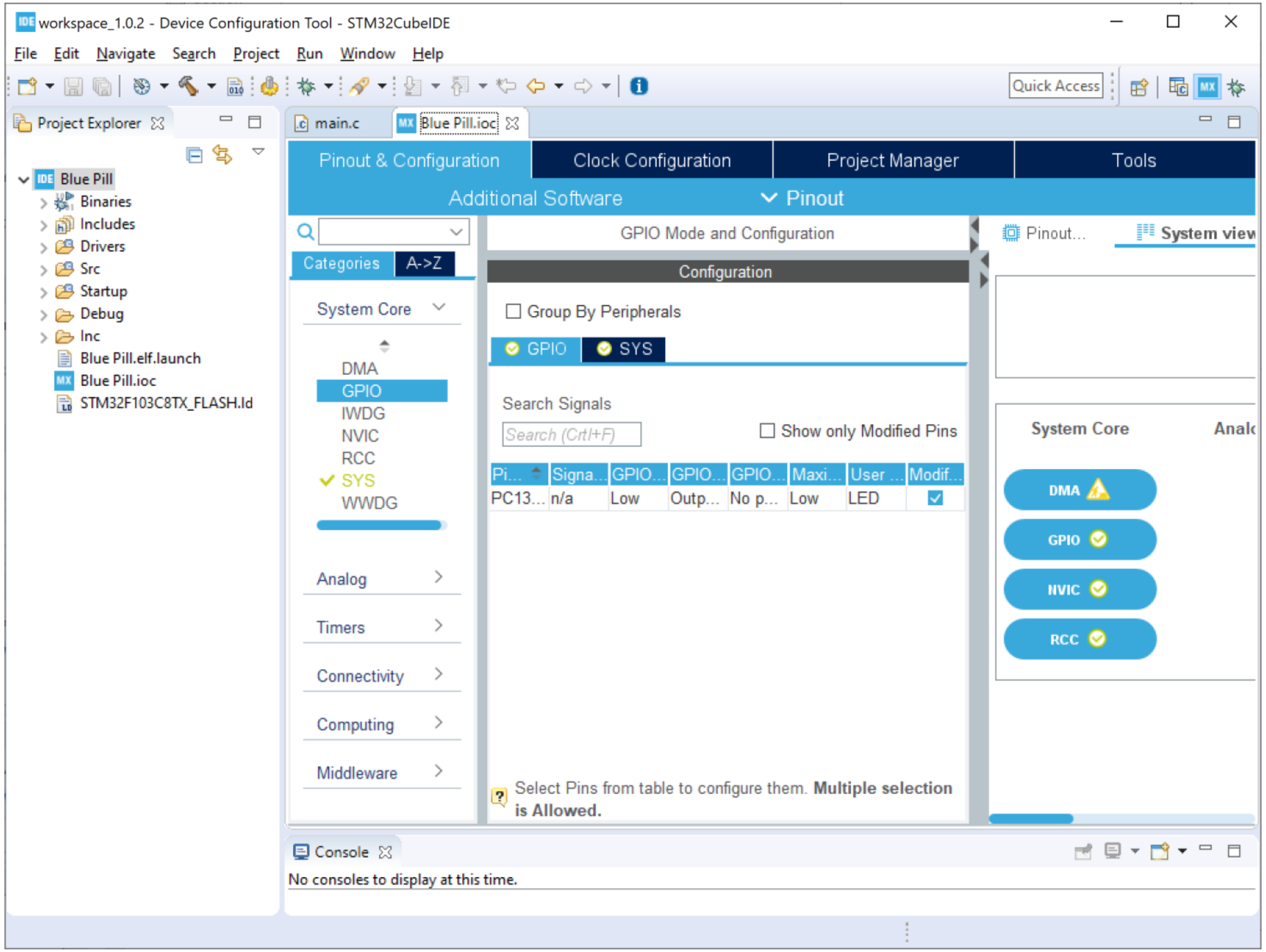
Change to “System View”

- Click on the “System View” tab
  - (instead of the “Pinout View” tab)



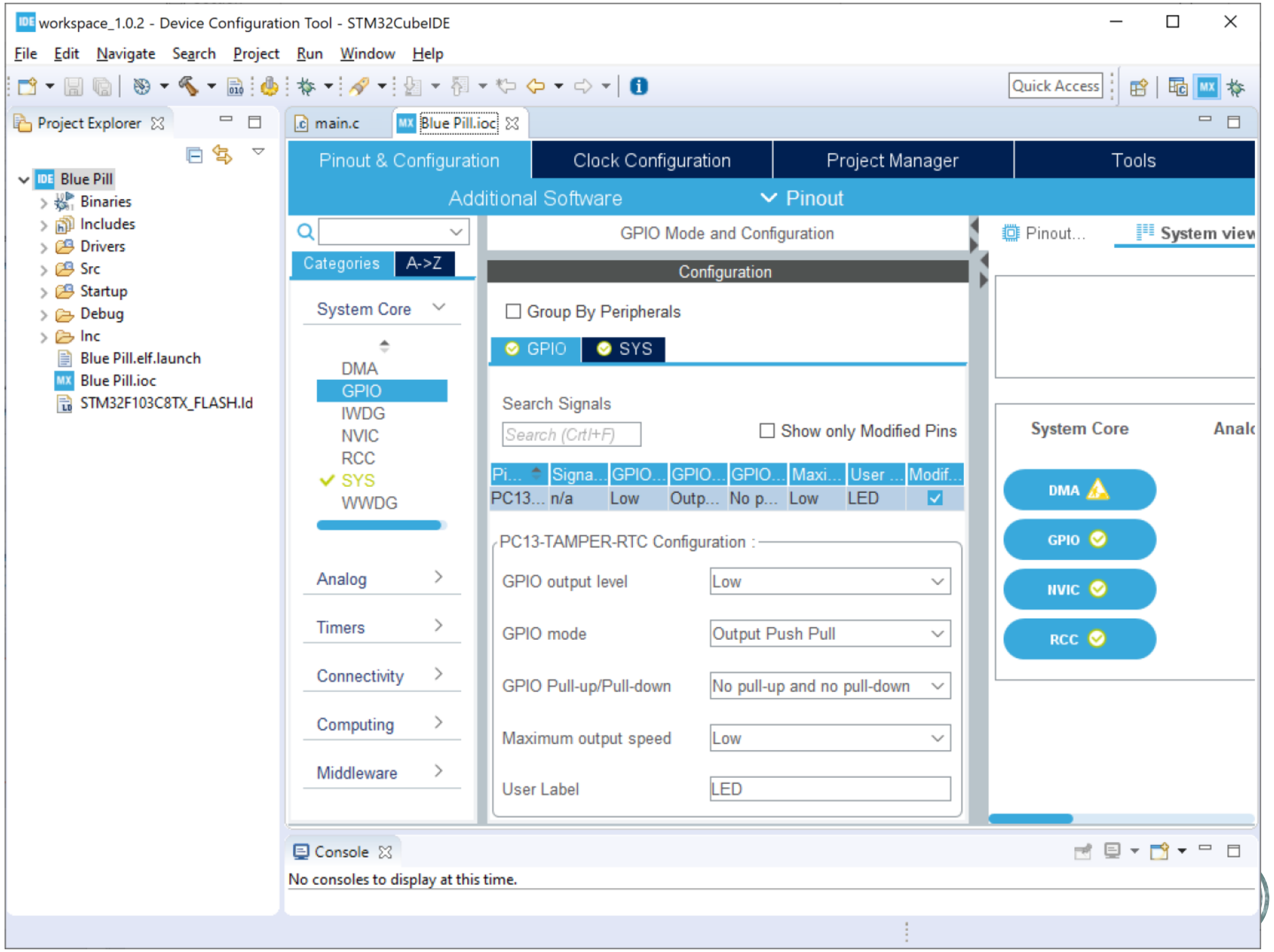
Configure GPIO

- Click on the “GPIO” button
- Now we see the “GPIO Mode and Configuration” panel



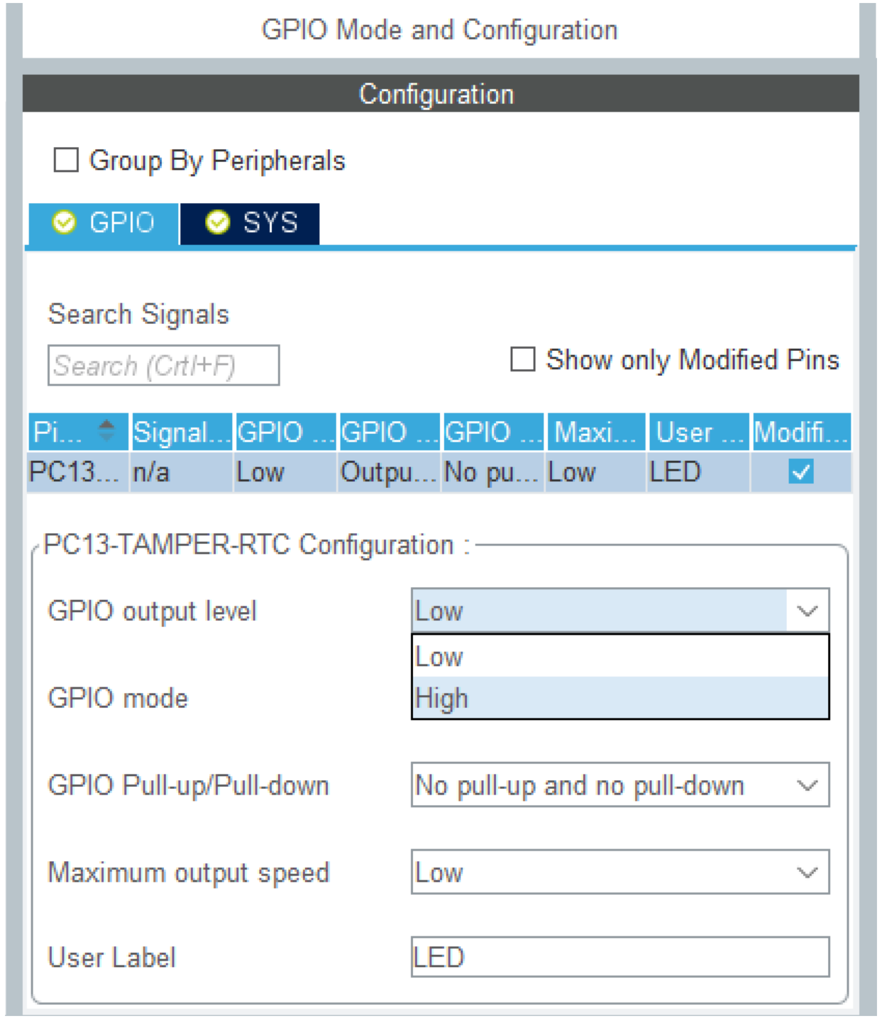
Select “PC13” line

- We have configured a single GPIO pin, PC13
- Its full name is “PC13-TAMPER-RTC”
- Click on its line to reveal options



Change GPIO output level

- Change “GPIO output level” from “Low” to “High”

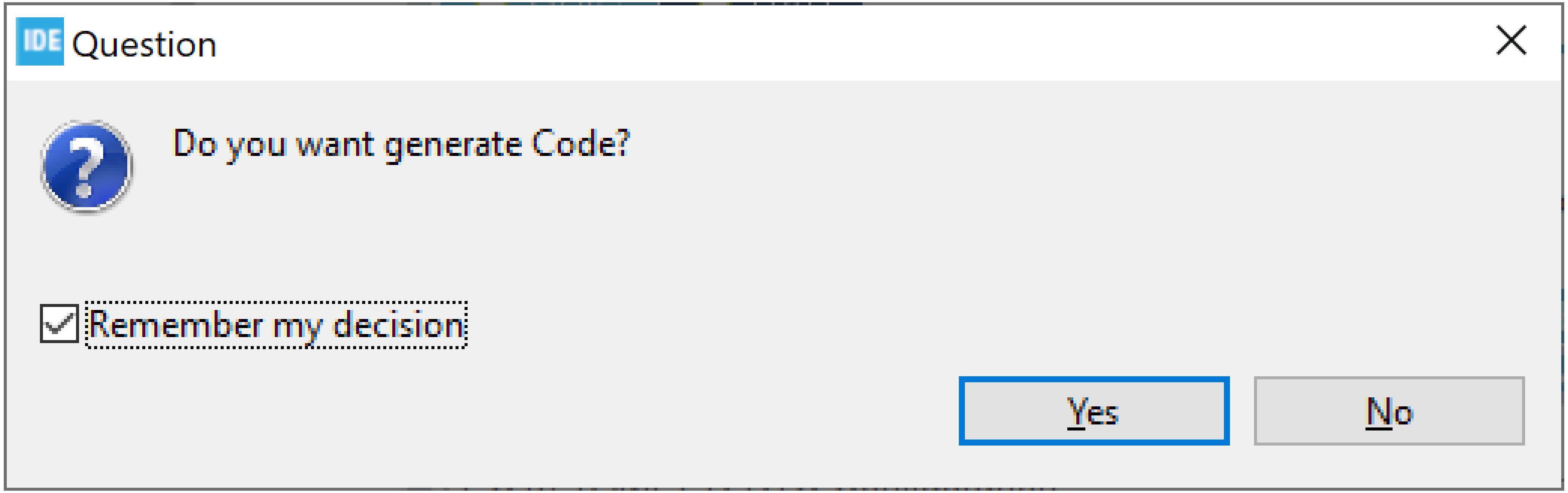


Save the changes

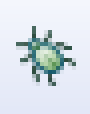
- Click the “Save (Ctrl+S) toolbar button
  - For you youngsters, that’s a “floppy disk”

Generate Code Option

- Check the “Remember my decision” checkbox
- Click “Yes”



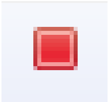
Debug, Observe, Question

- Click the “Debug Blue Pill.ioc” toolbar button 
- Once the “Debug Perspective” loads, click the “Resume (F8)” toolbar icon 
- Observe that the second LED (not the Power LED) is now off
- But why?

Active High vs. Active Low

- A GPIO can be configured as a digital output pin
- It can be programmed to be either “high” or “low”
- Its output state can be changed at any time
- On the Blue Pill, the LED connected to PC13 is configured as “active low”
- This means the LED turns on when the output pin is “low”
- When the output pin is “high”, the LED turns off
- The opposite is true for the Arduino’s Uno’s D13 pin (LED_BUILTIN)
- It is on when the output is “high” and off when the output is “low”
- Both systems work just fine – just be sure you know which one you have!

Stop Debugging

- Click the “Terminate (Ctrl+F2)” toolbar icon 
- The “C/C++” perspective returns
- Now it’s time to add our own code

A Note About User Code

- The STM32CubeMX code generator will preserve “user code” when in the right place
- These areas are indicated by comments in the generated code, e.g.,

```
/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
```

- Code between these two comments will be preserved
- Any code you write elsewhere will be over-written the next time code is generated

A Look at main()

- In the file “**main.c**”, there is a function called “**main()**”
- It is the starting point for all programs written in the C language
- In reality, some initialization stuff happens before the **main()** function is called
- Let’s take a look at out **main()** function in more detail

More About main()

- Here’s what our **main()** function looks like, stripped of all non-executable comments:

```
int main(void) {
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  while (1) {
  }
}
```

- First it calls some initialization functions
- Then goes into an endless loop

The while() Loop in main()

- Let's take a closer look at the **while()** loop in our **main()** function

```
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

- We see two areas for "user code" where we can add some code
  - (that won't get over-written by the code generator!)

Let’s Steal Some Code

- Why write when you can steal? ☺
- Scroll down to line 154 in **main.c**
- It looks like this:

```
HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
```

- This is a call to the function **HAL_GPIO_WritePin()**
  - This comes from ST’s HAL (Hardware Abstraction Layer) library
  - It has a similar function as Arduino’s **digitalWrite()** function
- It takes as parameters a port, a pin and a state
- Highlight this entire line of code and copy it

Paste Code into main()

- Paste the stolen code into the **while()** loop within the **main()** function at line 102:
- Paste it again
- It should look like this:

```
/* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
  }
  /* USER CODE END 3 */
```

Change One Parameter

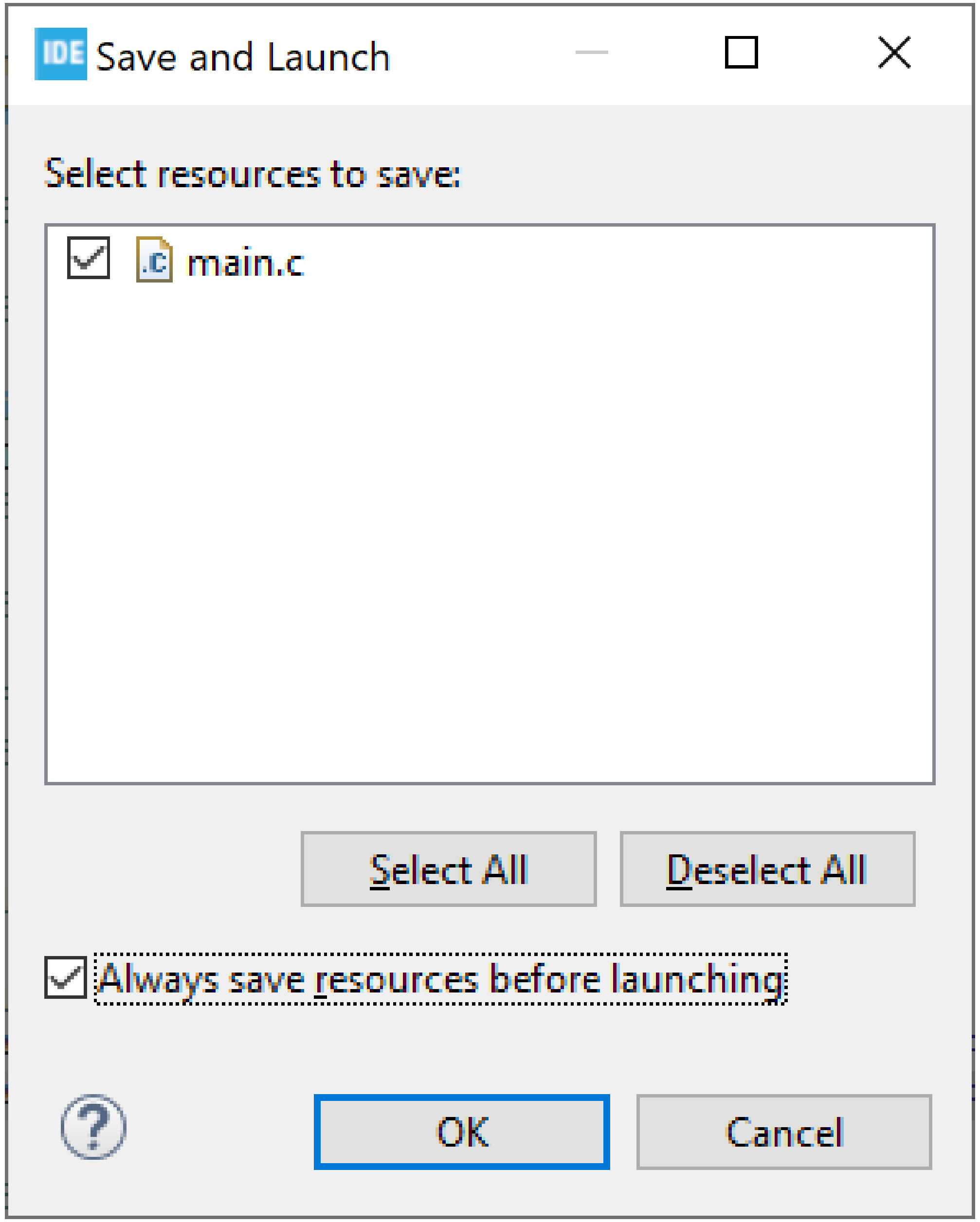
- Change the last parameter (state) in the second **HAL_GPIO_WritePin()** function call

```
/* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
  }
  /* USER CODE END 3 */
```

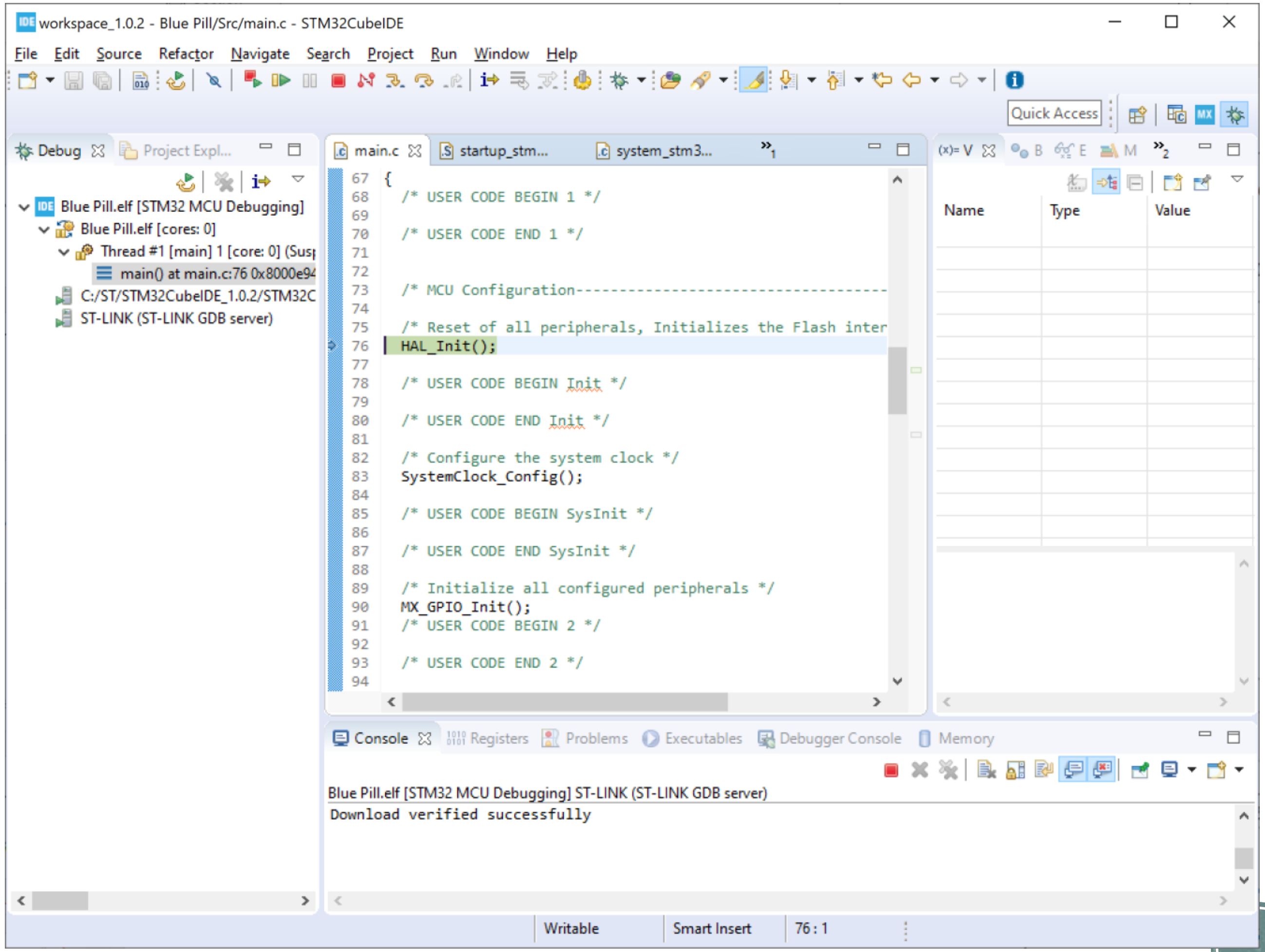
Let’s Debug

- Click the “Debug main.c” toolbar icon
- Since we have unsaved code changes, we are now reminded
- We are also given the opportunity to always automatically save changes before debugging, which is handy
- Check the “Always save…” checkbox
- Click “OK”



Step Through the Code

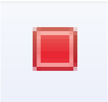
- The Debug perspective appears
- As before, the **HAL_Init()** function is highlighted
- Click on the “Step Over (F6)” toolbar icon 
- Keep clicking until you get into the **while()** loop
- Observe the LED
  - *GPIO_PIN_RESET* = LOW = ON
  - *GPIO_PIN_SET* = HIGH = OFF



Run at Full Speed

- Click on the “Resume (F8)” toolbar icon 
- Observe the LED again
- It appears to be on, but is in fact blinking so fast you can’t see it
- Let’s slow it down a bit

Add Some Delays

- Click the “Terminate (Ctrl+F2)” toolbar icon 
- Type the following line of code between the two **HAL_GPIO_WritePin()** function calls:

```
HAL_Delay(250);
```

  - Note: Capitalization counts!
  - The parameter (250 in this case) is the number of milliseconds to delay
- Type it again (or just copy what you already typed) after the second **HAL_GPIO_WritePin()** function call
- That section should look like this now:

```
HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
HAL_Delay(250);
HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
HAL_Delay(250);
```

Time to Debug

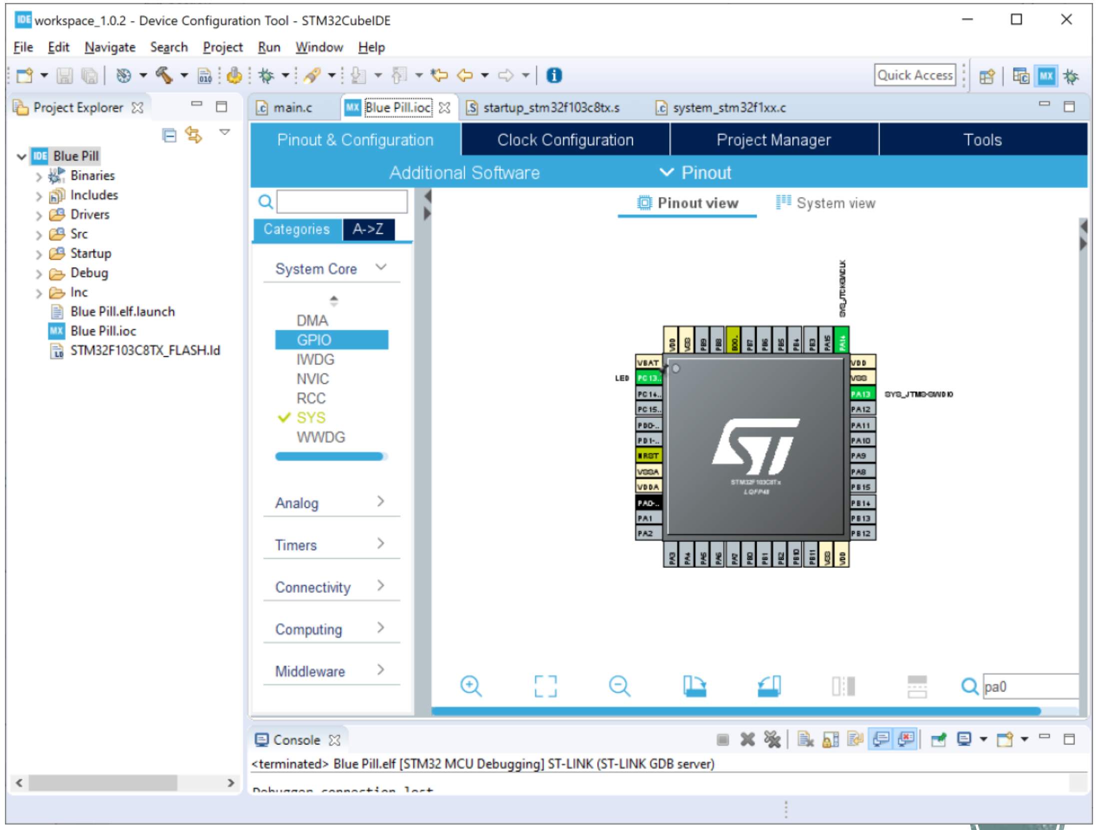
- Click the “Debug main.c” toolbar icon
- Once the Debug perspective loads, click the “Resume (F8)” toolbar icon
- Observe the LED
- It should be blinking about two times per second (~2 Hz)
- Once the novelty fades, click the “Terminate (Ctrl+F2)” toolbar icon

Add Another LED

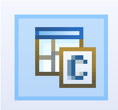
- Select one of the LEDs from the kit package (red, green or blue)
- Also get one of the included resistors from the LED package
  - They are all the same value
- Install the LED and resistor per the live demonstration
- Add a jumper wire from GPIO pin A0 to LED anode
- Use the resistor as a jumper from the LED cathode to the breadboard ground rail

Configure New LED Pin

- Select the “Device Configuration Tool” perspective
- Select the “**Blue Pill.ioc**” tab in the editor panel
- Select the “Pinout View” tab
- Type “PA0” into the search tool
- GPIO pin PA0 begins to blink in the diagram
- Click on the PA0 pin
- Select “GPIO_Output” from the context menu
- Right-click PA0 and select “Enter User Label”
- Enter “LED2” and press the [Enter] key
- Click “Save (Ctrl+S)” toolbar icon



Add Code for LED2

- Select the “C/C++” perspective 
- Select the “**main.c**” tab in the editor
- Copy line 102:
  ```
  HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
  ```
- Paste as line 103
- Change “LED_” to “LED2_” in two places:
  ```
  HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
  ```
- Copy this new line (line 103) and paste as line 106
- Change “_RESET” to “_SET”
  - `HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_SET);`

Verify Code for New LED

- The new code within the **while()** loop should look like this now:

```
HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
HAL_Delay(250);
HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_SET);
HAL_Delay(250);
```

Debug the New LED Code

- Click the “Debug main.c” toolbar icon
- Once the “Debug” perspective loads, click the “Resume (F8)” toolbar icon
- Observe the two LEDs blinking
- We connected the second LED as “active high”
- The new LED turns on when the output level is “high”
- It should be on when the on-board LED is off, and vice versa
- Once this no longer “sparks joy”, click the “Terminate (Ctrl+F2)” toolbar icon

Using Timers to Blink an LED

- We told the STM32 to blink some LEDs and it is doing exactly that
- It is also doing absolutely nothing else at the same time
  - besides taking up space, consuming power and contributing the to heat death of the universe
- The STM32 has many peripheral devices that could help us here
- For example, we could set up a timer to periodically interrupt our "foreground task"
- In the "interrupt handler" routine, or "background task", we put the code to blink LEDs
  - This would leave the foreground task free to do other work without LED timing concerns
- The STM32F103C8 has three general purpose timers and one "advanced" timer
- All of these timers can blink the LEDs directly using pulse-width modulation (PWM)

Pulse-Width Modulation PWM

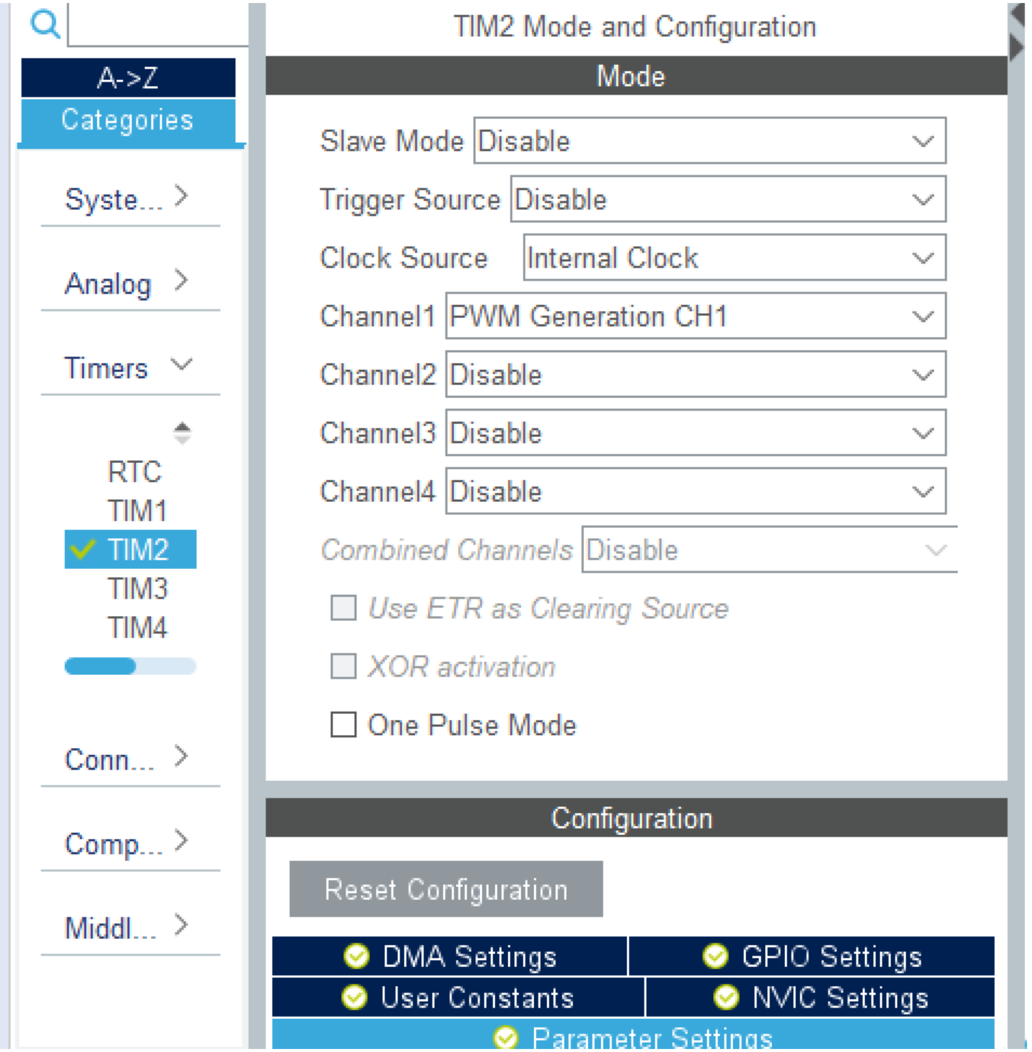
- A pulse-width modulated signal is a periodic (i.e., repeating) signal with a duty cycle
- The duty cycle is the ratio of how long the signal is “on” compared to the period
- For example, a “50% duty cycle” is on half the time, and then off the other half
- PWM signals have many uses in electronics, not just blinking LEDs
- Let’s configure a timer to blink our new LED
    - It’s no coincidence that we chose PA0 to drive the additional LED

Configure Pin to Use PWM

- Select the “Device Configuration Tool” (CubeMX) perspective
- Select the “**Blue Pill.ioc**” tab in the editor
- Select the “Pinout view” tab, if it is not already showing
- Click PA0 pin (LED2) in the diagram
- Select “TIM2_CH1” in the context menu
- This changes the function of pin PA0 (Port A, pin 0)
  - Previously it was GPIO_Output, and we used code to turn the LED on and off
  - Now PA0 is connected to Timer2
  - Timer2 has four independent PWM channels – we have selected channel 1
- Right-click PA0, select “Enter User Label” and type “LED2” again (the name was lost)

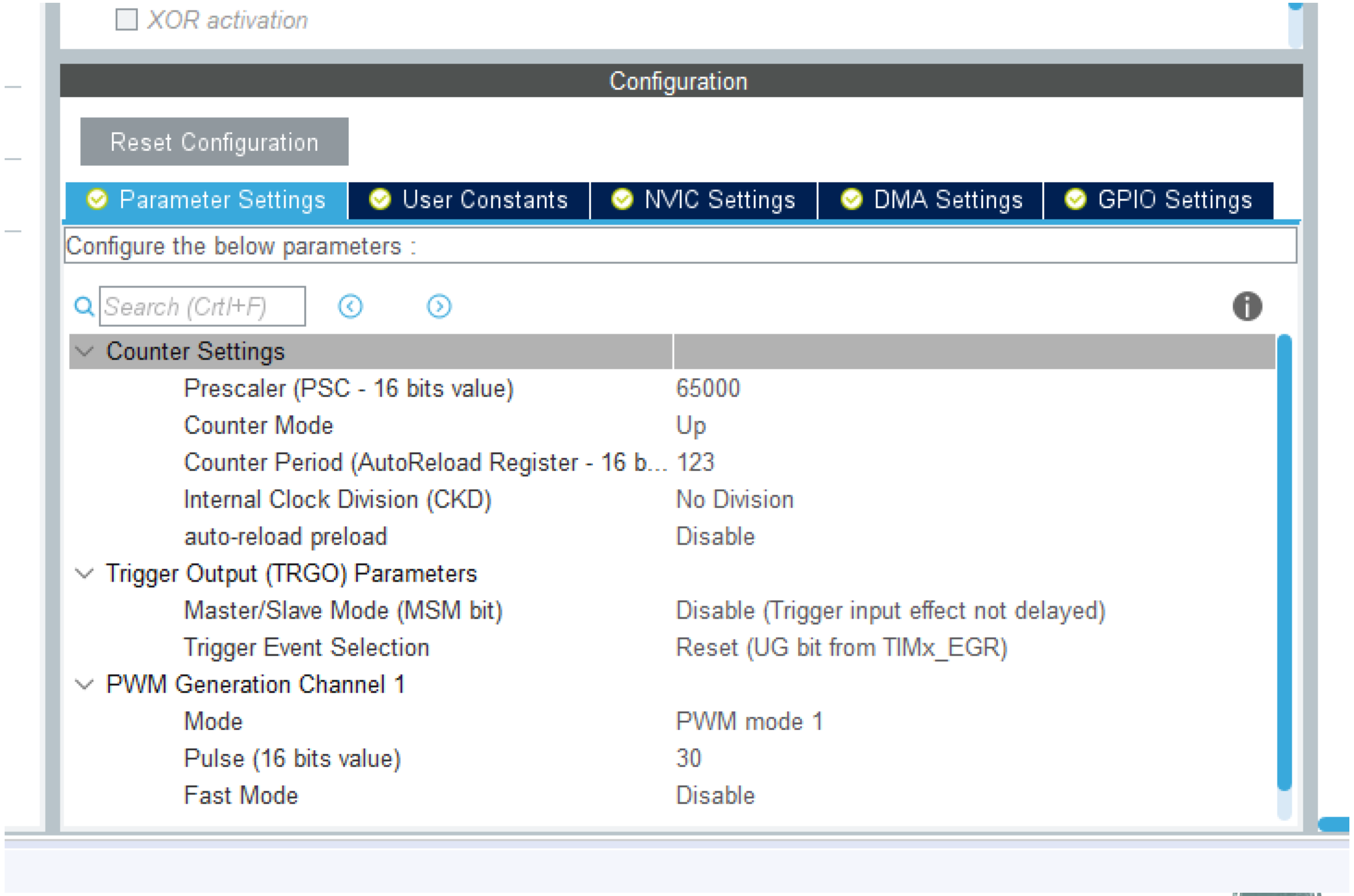
Configure Timer2 for PWM

- Under “Timers”, select “TIM2”
- Configure the “Mode” settings
- For “Clock Source”, select “Internal Clock”
- For “Channel 1”,
  select “PWM Generation CH1”

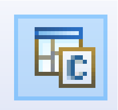


Configure Timer2 Channel 1

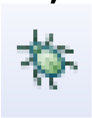
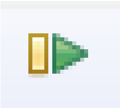
- In the “Configuration” section…
  - Some window adjustments may be required
- For “Prescaler”, enter “65000”
- For “Counter Period”, enter “123”
- For “Pulse”, enter 30
- Click the “Save (Ctrl+S)” toolbar icon 
- This generates the new code



Start Timer2 in PWM Mode

- Timers do not start automatically
- We will add code (one line) to start Timer2's Channel 1 in PWM mode
- Select "C/C++" perspective 
- Select "**main.c**" tab in editor
- Note line 93 was added: `MX_TIM2_Init();`
- On line 95, add this line of code (within the "USER CODE BEGIN 2" section):

```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
```

- Click on "Debug main.c" toolbar icon 
- Click "Resume (F8)" toolbar icon 

Observations

- Observe LED blinking pattern
- The on-board LED is still blinking at ~2 Hz
- The off-board LED is now blinking in a different pattern
  - *…even though we did **not** remove the code from the **while()** loop*
- Once we assigned pin PA0 to TIM2_CH1, it is no longer connected to GPIO Port A
- PA0 is entirely controlled by the timing parameters we selected for Timer2 (TIM2)

PWM Timing Parameters

- PWM signals require three main parameters to be specified

1. A clock source (we selected “Internal Clock”) and prescaler (we chose “65000”)
2. A period (we chose “123”)
3. A duty cycle (or “pulse”; we chose “30”)

Clock Source and Prescaler

- The combination of clock source and prescaler determine the speed of the timer
- The default clock speed of the STM32F103 is 8 MHz (8,000,000 cycles per second)
- This is generated internally on the STM32F103 chip using the HSI oscillator
  - HSI stands for “High Speed Internal” oscillator
  - No external components are required
  - Clock accuracy is around ±2% over the entire temperature range (-40°C to 105°C)
- This clock source is then divided by the value contained in the prescaler
  - Since we selected a value of “65000” (it could be anything from 0 to 65535), the resulting clock frequency driving Timer2 is:

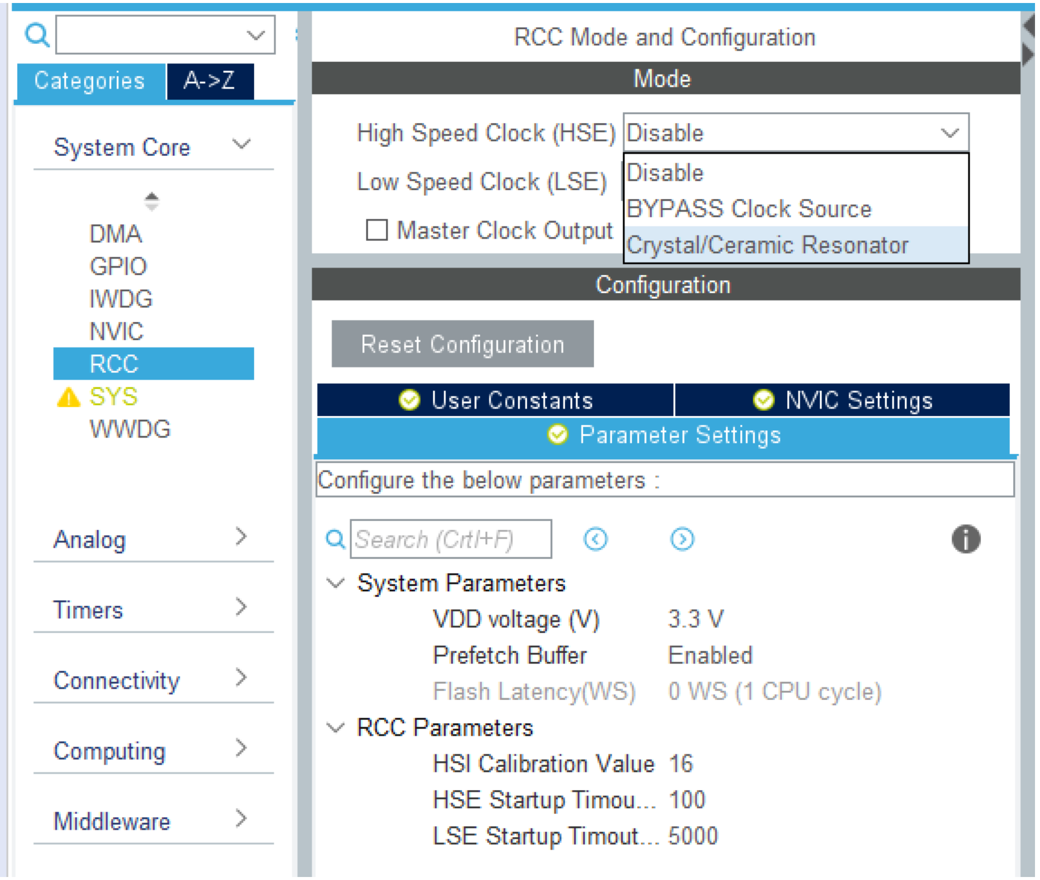
$$8{,}000{,}000\ Hz \div 65{,}000 \approx 123.0769\ Hz$$

PWM Period and Duty Cycle

- We chose a value of “123” for the period of Timer2
- This results in Timer2 completing a cycle approximately once every second
- We chose a value of “30” for the duty cycle, or the “on” portion of the signal
- This means that the LED is on for around a quarter of the time,
  and off for the remaining three-quarters of the cycle
- You should see this reflected in the blinking pattern on LED2

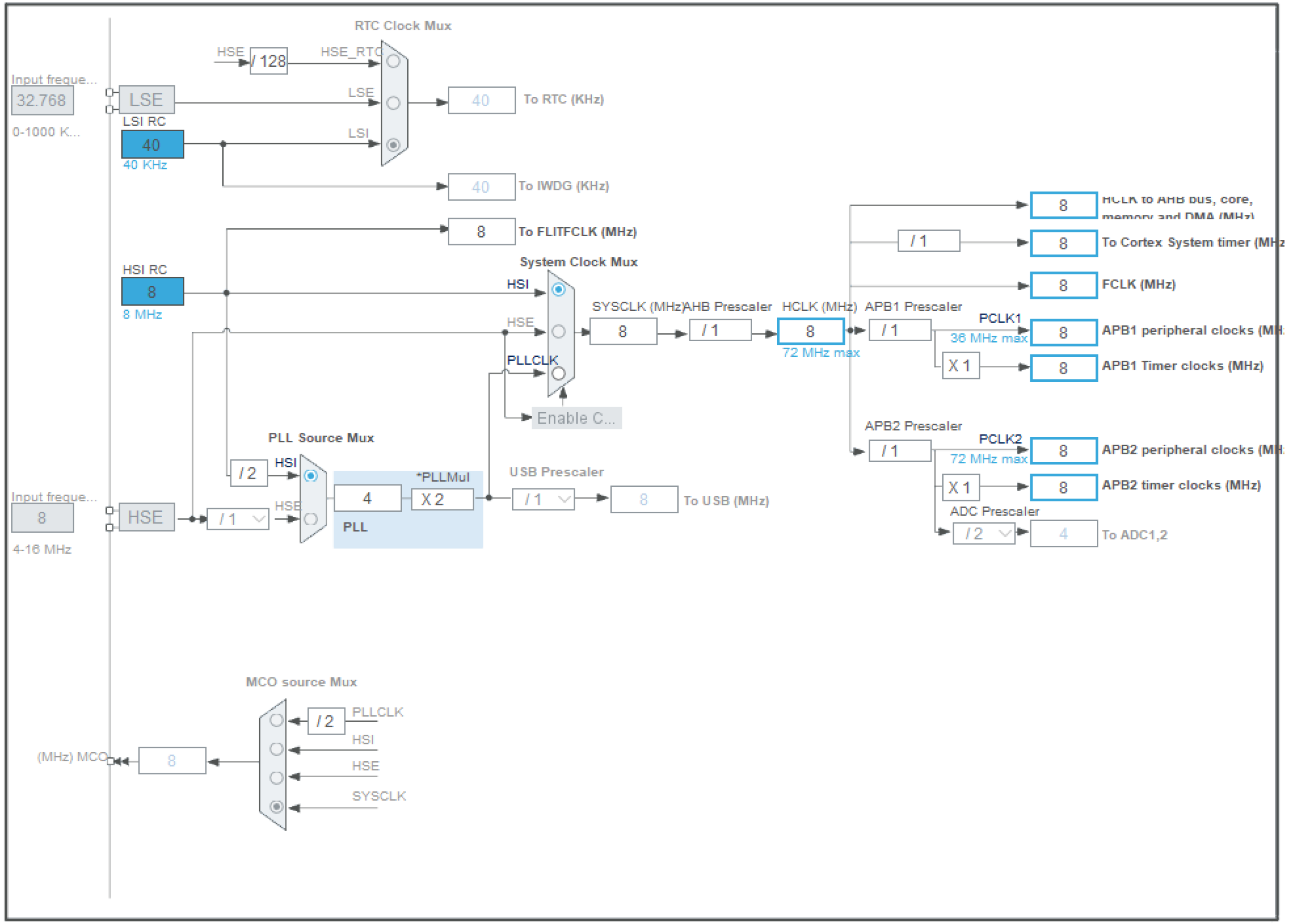
Increase Clock Speed

- The Blue Pill contains an 8 MHz quartz crystal with much better speed tolerance than the HSI
  - Typical tolerances for quartz crystals are measured in “parts per million” instead of percent
- The STM32F103 contains a phase-locked loop (PLL) that can multiply the clock frequency
- Select the “Device Configuration Tool” perspective
- Select the “**Blue Pill.ioc**” tab in the editor
- Under “System Core”, select “RCC”
- For “High Speed Clock (HSE)”, select “Crystal/Ceramic Resonator”



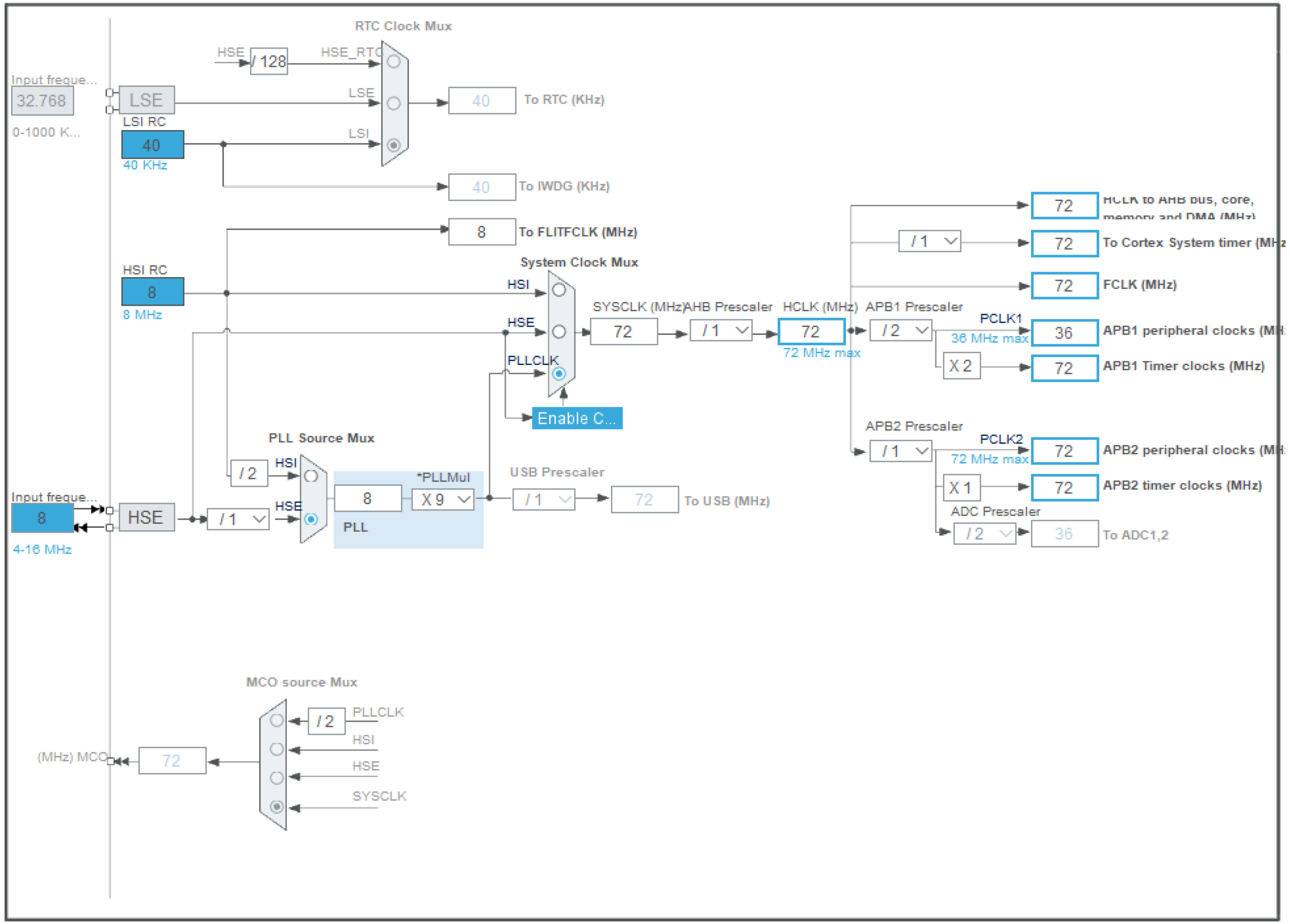
The Clock Tree

- Select the "Clock Configuration" tab
- Behold the "Clock Tree" in its default configuration
- You can zoom in and out
- The possible clock sources are on the left side
- Routing options are in the middle part
- The resulting clock(s) are on the right side (in blue boxes)

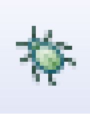


Configure the Clock Tree

- For “PLL Source Mux”, select “HSE”
- For “*PLLMul”, select “X 9”
- For “System Clock Mux”, select “PLLCLK”
- For “APB1 Prescaler”, select “/ 2”
  - APB1 can only run at 36 MHz
- Now the chip will run at 72 MHz

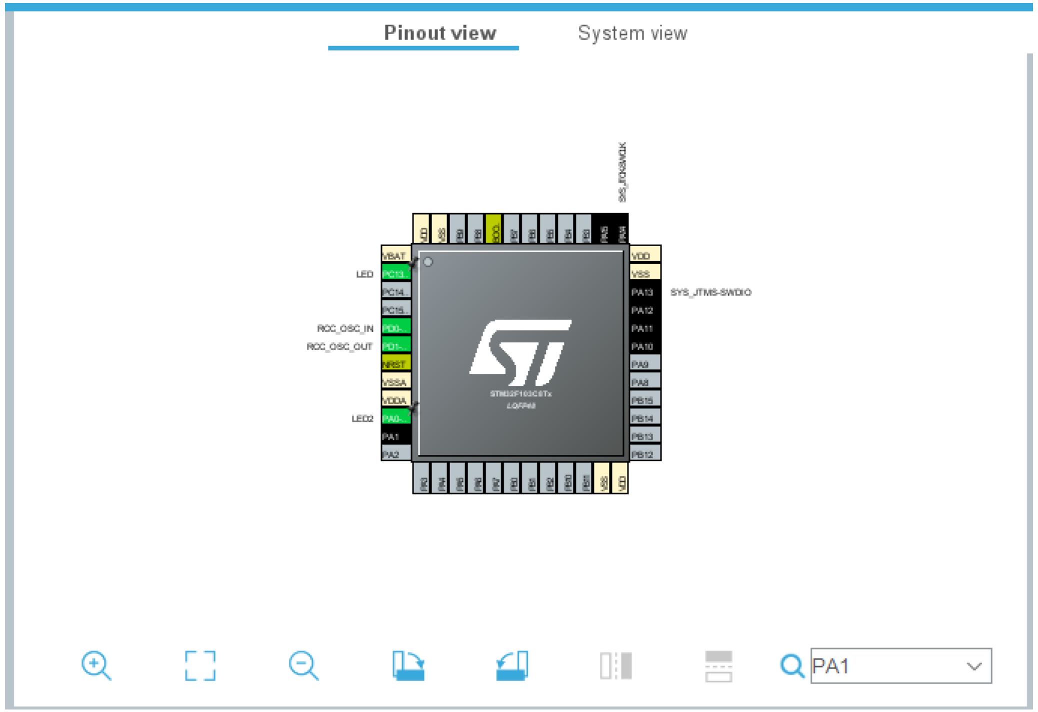


Debug and Observe Results

- Click the “Save (Ctrl+S)” toolbar icon 
- This triggers new code generation
- Once the new code is generated, click the “Debug main.c” toolbar icon 
- Once the Debug perspective loads, click the “Resume (F8)” toolbar icon 
- LED2 is now flashing much faster
  - About nine times faster
- The original LED is still blinking at ~2 Hz
- That’s because the **HAL_Delay()** function knows the current clock frequency
  - The parameter we supplied (250) is number of milliseconds to delay, or about ¼ of a second

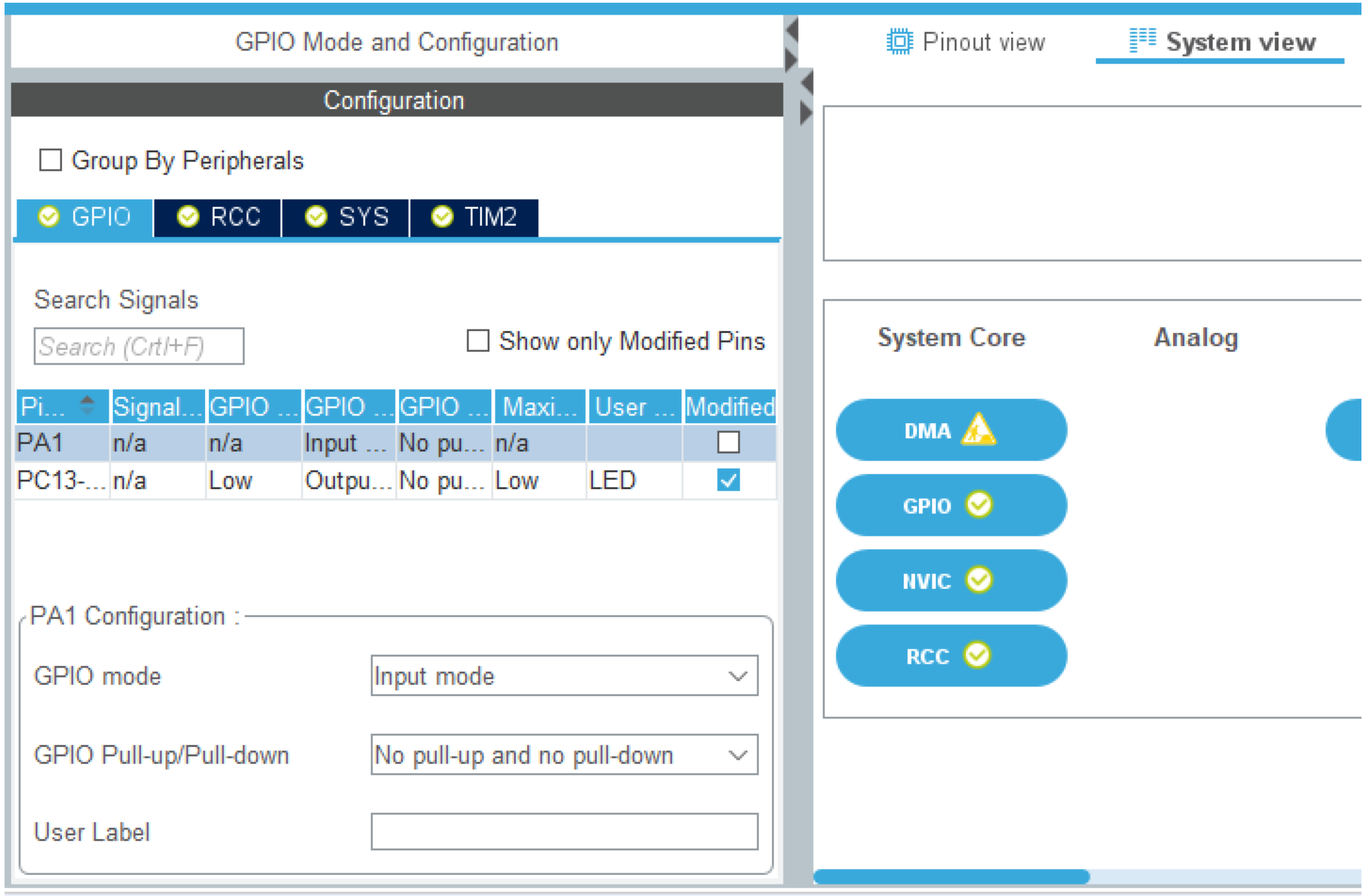
Using Buttons as Inputs

- Connect a jumper wire from GPIO pin A1 to top-left pin on yellow push button
  - There should already be a jumper installed from the other button pin to the ground rail
- Select the “Device Configuration Tool” perspective
- Select the “Blue Pill.ioc” tab in the editor
- Use the search tool to find pin PA1
- All matching pins start to blink
- Click on pin PA1 and select “GPIO_Input”
- Right-click on PA1 and select “Enter User Label”
- Enter “button”

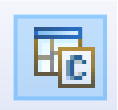


Configure Button Options

- Select "System view" tab
- Click "GPIO" button
- Click on pin "PA1" line
- For "GPIO Pull-up/Pull-down", select ""Pull-up"
- Click the "Save (Ctrl+S)"  toolbar icon
- This generates new code



Add Code for Button Test

- Select the “C/C++” perspective 
- Select the “**main.c**” tab in the editor
- Add the following code after line 110:

```
if(HAL_GPIO_ReadPin(button_GPIO_Port, button_Pin) == GPIO_PIN_RESET) {
  TIM2->CCR1 = 60;
} else {
  TIM2->CCR1 = 0;
}
```

Test New Button Code

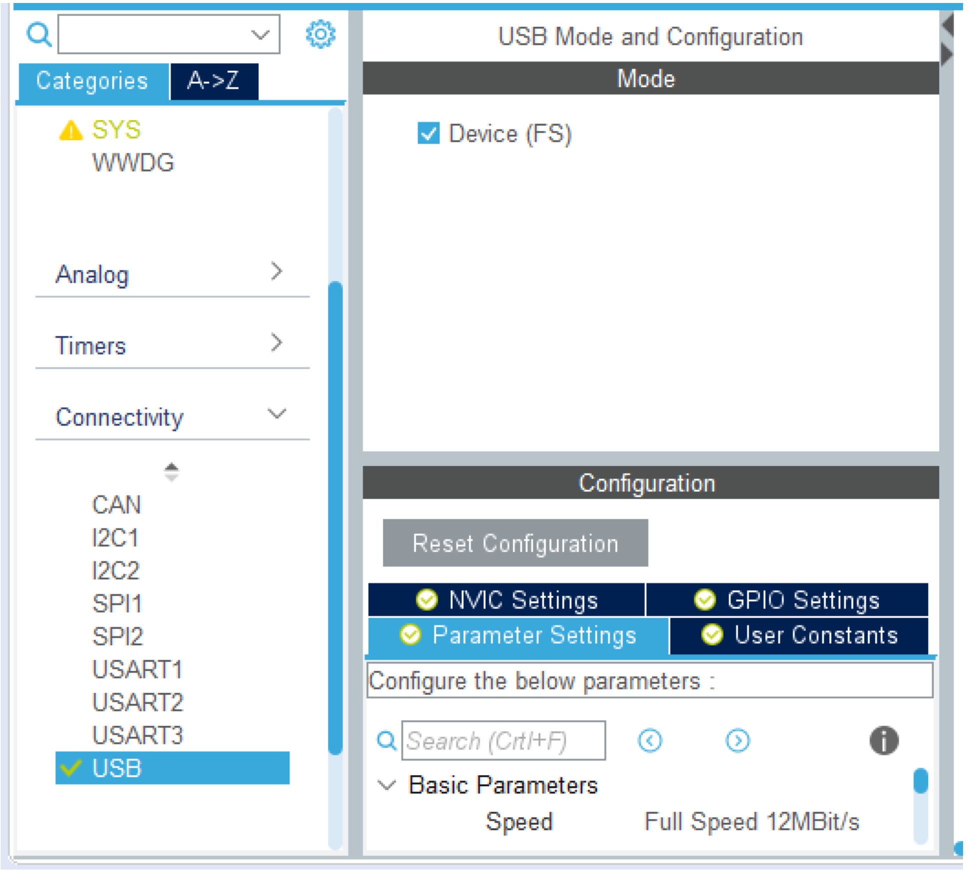
- Click the “Debug main.c” toolbar icon
- Once the Debug perspective loads, click the “Resume (F8)” toolbar icon
- Test the effects of button presses on LED2’s output
- Analysis:
  - When the button is not pressed, no electrical connection is made with input pin PA1
  - Since we configured a “pull-up” resistor to be active on this input, it now reads as “high”
  - When the button is pressed, PA1 is connected to ground, and now reads as “low”
  - The **HAL_GPIO_ReadPin()** function returns the constant value *GPIO_PIN_RESET* (or 0)
    - Hover your mouse pointer over *GPIO_PIN_RESET* to see for yourself
  - This sets the duty cycle of TIM2_CH1 to “60”, which is flashing
  - Otherwise, the duty cycle is set to “0”, or always off
- Click the “Terminate (Ctrl-F2)” toolbar icon when you grok in fullness

USB Device

- The STM32F103C8 device supports a USB Full Speed (FS) interface for devices
  - It does not support “USB host” mode
- STM32CubeMX will supply basic code for six (6) types of device classes:
  - Audio
  - Communication (virtual serial port)
  - Download Firmware Update (DFU)
  - Human Interface Device (HID)
  - Custom Human Interface Device
  - Mass Storage
- Step by step video from “Hugatry's HackVlog” for setting up a USB CDC:
  - https://www.youtube.com/watch?v=YZjnCOun1wU

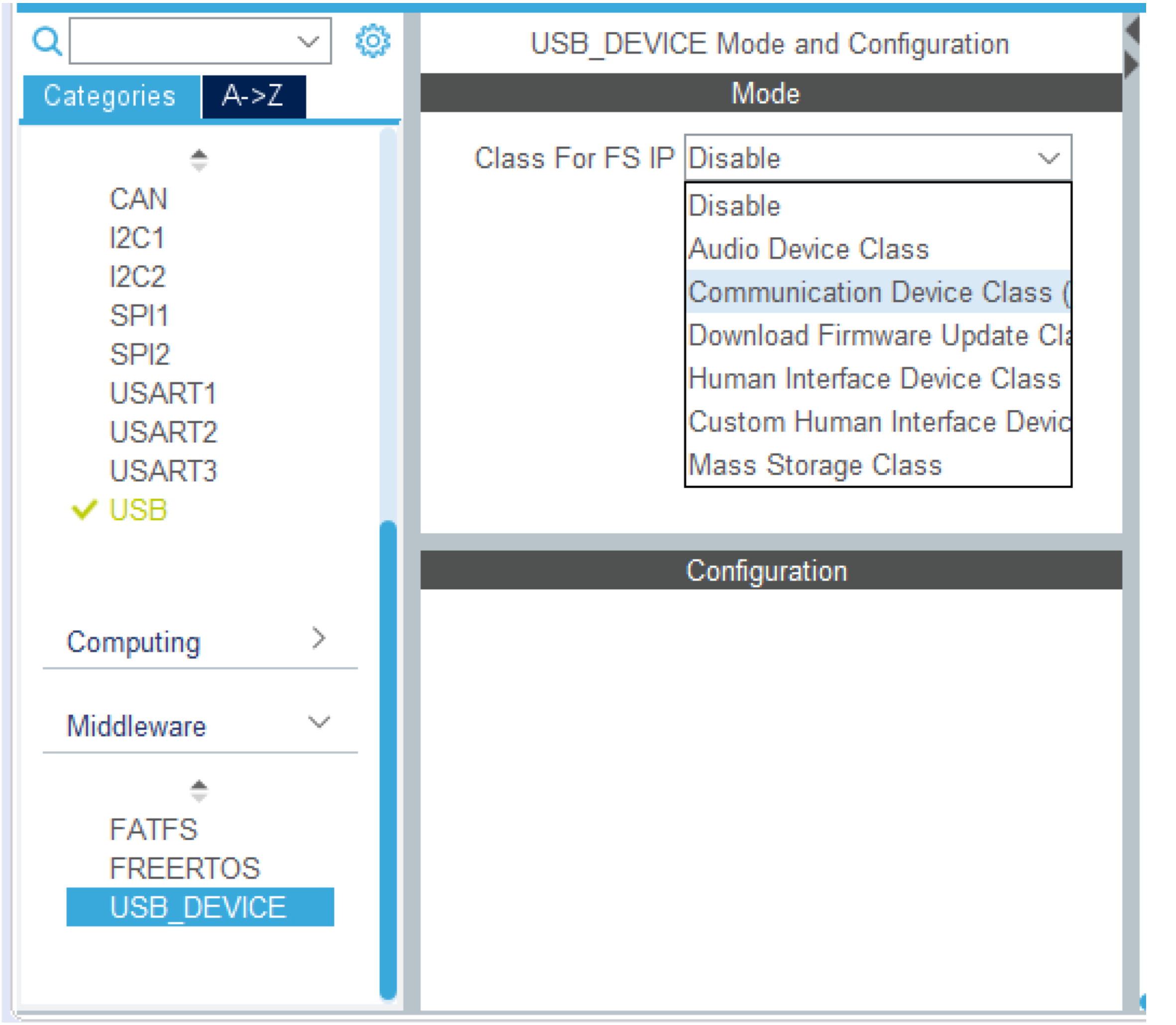
Enable USB Interface

- Select the “Device Configuration Tool” perspective
- Select the “Blue Pill.ioc” tab in the editor
- Under “Connectivity”, click on “USB”
- Check the “Device (FS)” checkbox



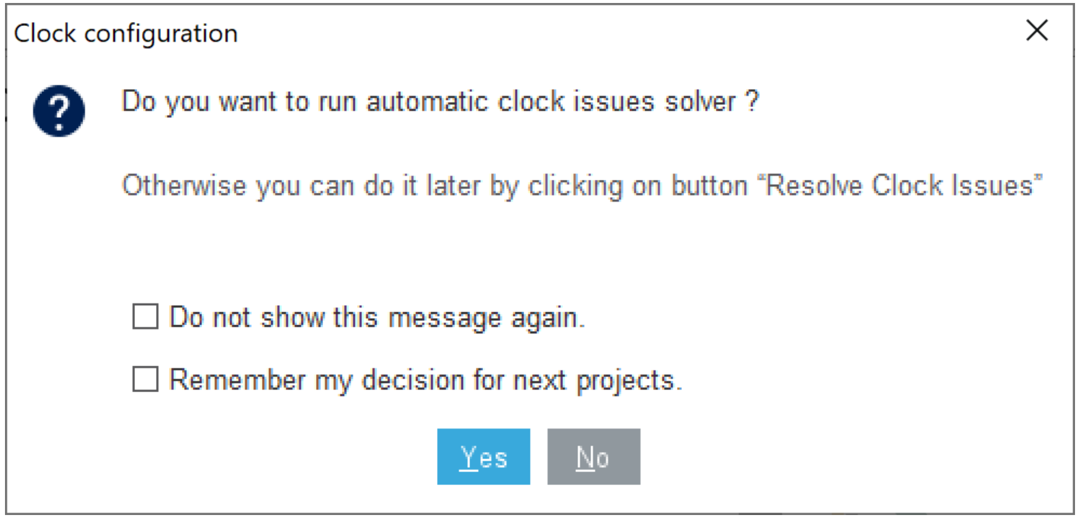
Configure USB Device Class

- Under "Middleware", select "USB_DEVICE"
- For "Class For FS IP", select "Communication Device Class (Virtual Port Com)"



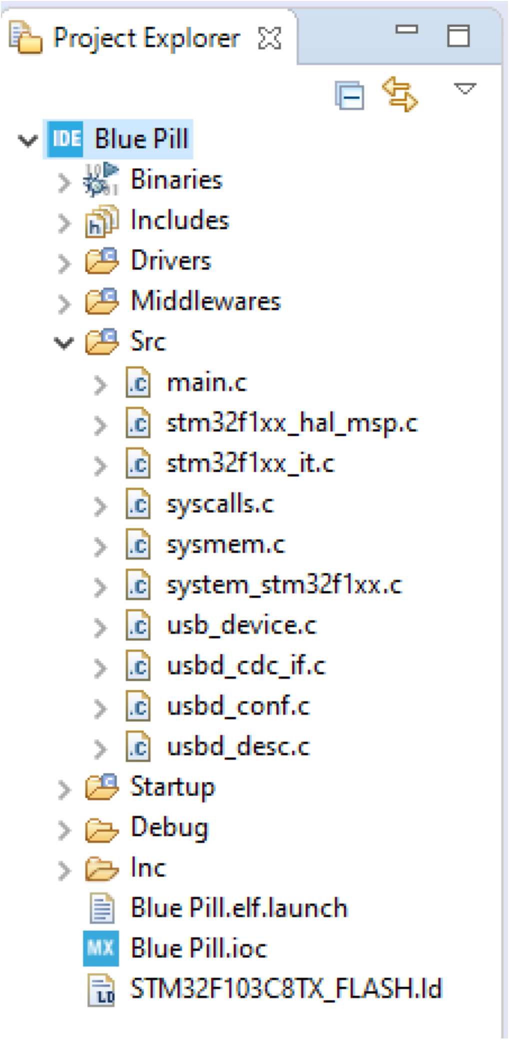
Configure USB Clock

- Select the “Clock Configuration” tab
  - You may have noticed that the tab had a red X on it
  - We will fix that problem right now
- The application will offer to help
- Click “No” for now
- Find the error on the Clock Tree
- For “USB Prescaler”, select “/ 1.5” from the drop-down menu
- USB Clock is now 48 MHz ☺
- Click the “Save (Ctrl+S)” toolbar icon 

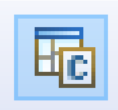


USB Code Added to Project

- The STM32CubeIDE has added several files to your project to support USB
- You can find them using the “Project Explorer” panel
- We will be adding code to both your **main.c** file as well as the **usbd_cdc_if.c** file



Add USB Code to main.c

- Select the “C/C++” perspective 
- Select the “**main.c**” tab in the editor
- After line 26 (`USER CODE BEGIN Includes`), add the following line:

```
#include "usbd_cdc_if.h"
```

- After line 98 (`USER CODE BEGIN 2`), add the following two lines:

```
HAL_Delay(1000);
CDC_Transmit_FS((uint8_t *)"Hello, world\r\n", 14);
```

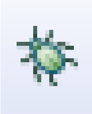
- After line 122 (inside the **while()** loop), add the following line:

```
CDC_Transmit_FS((uint8_t *)"Tick\r\n", 6);
```

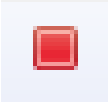
Add Code to usbd_cdc_if.c

- In the “Project Explorer” panel, double-click the **usbd_cdc_if.c** file
- This will open the file in a new tab in the editor
- After line 265 (`USER CODE BEGIN 6`), add the following line:

```
if(Buf[0] == ‘?') CDC_Transmit_FS((uint8_t *)“!", 1);
```

- Click the “Debug usbd_cdc_if.c” toolbar icon 
- Click the “Resume (F8)” toolbar icon 

Connect and Test

- Connect your Blue Pill to your laptop with the USB cable provided
- Allow a device driver for the Virtual Serial Port to be installed
- Open a serial terminal (Tera Term, PuTTY, whatever you like)
- Select the newest or highest numbered COM port
  - Use Device Manager/Ports to determine the correct port to use
  - Baud rate does not actually matter here
- Observe a periodic “Tick” message to scroll down the terminal window
- Type the question mark key “?”
- The STM32 should reply with an exclamation point “!”
- When the novelty fades, click the “Terminate (Ctrl+F2)” toolbar icon 

Add a Character LCD

- Note: This is an extra credit experiment, as class time permits
- Remove the smaller static-dissipative bag containing LCD from the kit
- Note: The smaller bag contains both the LCD and a small resistor
- Install the LCD on the breadboard as shown on screen
- Only twelve (12) of the sixteen (16) LCD connections are required
  - This exercise requires great peace of mind

LCD Power, Ground and Bias

- Connect LCD pin 1 (V_{SS}) to ground rail
- Connect 5.0V pin on ST-LINK/V2 to lower-left power rail
- Connect LCD pin 2 (V_{DD}) to 5.0V rail
- Connect LCD pin 3 (V_O) to ground using 2.0 KΩ resistor
- Apply power to circuit
- Confirm LCD displays a single row of white boxes
- **Important**: Disconnect power before proceeding

LCD Backlight

- Connect LCD pin 15 (A) to 5.0V rail via resistor from LED bag
- Connect LCD pin 16 (K) to ground rail
- Apply power to circuit
- Verify that backlight is illuminated
- **Important**: Disconnect power before proceeding

LCD Control Signals

- Connect LCD pin 4 (RS) to pin PA8
- Connect LCD pin 5 (R/W) to pin PA9
- Connect LCD pin 6 (E) to pin PA10

LCD Data Signals

- Connect LCD pin 11 (D4) to PB12
- Connect LCD pin 12 (D5) to PB13
- Connect LCD pin 13 (D6) to PB14
- Connect LCD pin 14 (D7) to PB15

Add LCD Code to Project

- The code to interface to the LCD is contained in two files:
  - **lcd.c**
  - **lcd.h**
- Copy the **lcd.c** to the “Src” folder
- Copy the **lcd.h** to the “Inc” folder

Modify main.c

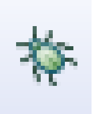
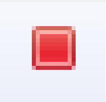
- Select the “C/C++” perspective
- Select the “**main.c**” tab in the editor
- After line 26 (`USER CODE BEGIN Includes`), add the following line:

```
#include "lcd.h"
```

- After line 98 (`USER CODE BEGIN 2`), add the following four (4) lines:
  - Comments are optional

```
LCD_init();
LCD_puts(" This is a test");
LCD_xy(0, 1); // 1st column, 2nd line
LCD_puts(" dalewheat.com");
```

Debug & Observe

- Connect ST-LINK/V2 adapter
- Click the “Debug main.c” toolbar icon 
- Once the Debug perspective loads, click the “Resume (F8)” toolbar icon 
- The LCD should initialize itself and then display a message
- Once you’ve memorized the message, press the “Terminate (F8)” toolbar icon 

Conclusion

- In this class we learned about the tools used to develop code for the STM32
- We used both internal and external devices to demonstrate code operation
- Breadboards and wire jumpers were used to quickly prototype new circuits
- You can learn more about STM32 on the STMicroelectronics' web site:
  - st.com/stm32

Questions and Answers

- What did you learn?
- Did you enjoy this class?
- Would you like to attend similar classes in the future?
- Were your expectations of this class met?
- What other topics would you like to investigate?

Thank you

- Thank you for your participation

Revision History

- February 2019 – v1.0 – original version, using Blue Pill, Atollic TrueSTUDIO, STM32CubeMX, ST-LINK Utility
- October 2019 – v1.1 – Updated for STM32CubeIDE
- October 2019 – v1.11 – Minor typos and corrections