

Lesson 1

Norman McEntire
norman.mcentre@gmail.com

Contents

- Introduction to Embedded C
- STM32CubeMX
- Tour of Generated Project
- TrueStudio
- Hello Blinking LED

Introduction to Embedded C

Introduction to Embedded C

- Good News: Embedded C is based on Standard C
 - More alike than different
 - Differences relate to lower-level hardware demands on Embedded C
- Classic Example: Hello World
 - Standard C: Display “Hello World” on console
 - Embedded C: Blink an LED

Standard C and Embedded C

How They Are Alike

- Pre-processor
 - #include, #define, #if, etc.
- main() Function as starting point for code
- Primitive Data Types: char, short, int, long, float, double
- User Data Types: typedef, struct, enum
- Operators: +, -, *, /, %, ++, +=, -=, *=, /=, ||, &&, |, &, !, etc.
- Control Structures: if, if-else, switch, break
- Looping: for, while, do, continue, break

Standard C vs Embedded C

main()

- Standard C
 - `int main(int argc, char *argv[])`
 - Command-line arguments
 - Returns when done
- Embedded C
 - `int main(void)`
 - No command-line arguments
 - Never returns (runs forever)

Standard C vs Embedded C

printf() and scanf()

- Standard C
 - Use printf() to send output to console
 - Use scanf() to read from console
- Embedded C
 - printf() may or may not be supported
 - And if supported, goes to serial port
 - scanf() not typically supported

Standard C vs Embedded C

Primitive Data Types

- Standard C
 - Use of char, short, int, long, float, double
 - Issue: Other than char/float/double, size of primitive data types not constant
- Embedded C
 - Use of ISO C99 Data Types
 - `int8_t`, `uint8_t` - 8 bits
 - `int16_t`, `uint16_t` - 16 bits
 - `int32_t`, `uint32_t` - 32 bits
 - `int64_t`, `uint64_t` - 64 bits

ISO C99 Data Types

Data type	Size	Range
<code>int8_t</code>	1 byte	-128 to 127
<code>uint8_t</code>	1 byte	0 to 255
<code>int16_t</code>	2 bytes	-32,768 to 32,767
<code>uint16_t</code>	2 bytes	0 to 65,535
<code>int32_t</code>	4 bytes	-2,147,483,648 to 2,147,483,647
<code>uint32_t</code>	4 bytes	0 to 4,294,967,295
<code>int64_t</code>	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
<code>uint64_t</code>	8 bytes	0 to 18,446,744,073,709,551,615

Standard C vs Embedded C

Memory Addressing

- Standard C
 - Virtual Memory Address Space
 - The address your program uses is not the same as the physical memory address
 - If program does not “fit” into memory, it can be swapped in/out as needed
- Embedded C
 - No virtual memory
 - The addresses your program uses is the same as physical memory space
 - Your code must fit into memory - no swapping in/out of memory

Standard C vs Embedded C

Memory Allocation

- Standard C
 - All memory the same - just one large block of virtual memory
 - Use of malloc(), calloc() to allocate memory
- Embedded C
 - May have different memory banks
 - Flash Memory
 - RAM Bank #1
 - RAM Bank #2
 - etc.

Standard C vs Embedded C

Hardware Initialization

- Standard C
 - Hardware initialize the operating system
- Embedded C
 - Embedded C code initializes the hardware
 - Hardware Clocks
 - Interrupt Vectors

Standard C vs Embedded C

Interrupt Handling

- Standard C
 - No concern for low-level interrupt handling
- Embedded C
 - Interrupt handling required for hardware devices that are interrupt driven

Standard C vs Embedded C

Use of Pointers

- Standard C
 - Use of pointers to point to data structures, etc.
- Embedded C
 - Same as standard C use of pointers
 - PLUS....heavy use of pointers to access physical addresses
 - Example: Pointer to base of GPIO Register

Standard C vs Embedded C

Use of Bit Operations

- Standard C
 - Some use of bit operations
 - Bitwise OR: |
 - Bitwise AND: &
- Embedded C
 - Heavy use of bit operations for I/O register operations
 - Set bits in registers
 - Test bits in registers

Standard C vs Embedded C Libraries

- Standard C
 - Uses “high-level” libraries that are “hardware independent”
- Embedded C
 - Uses “low-level” libraries that are close to the hardware
 - HAL - Hardware Abstraction Layer
 - LL - Lower Layer

Summary so far...

- Embedded C builds upon Standard C
 - Language Syntax the same: if, if-else, switch, for, while, do, continue, break, etc.
- Embedded C differences are due to need for lower-level hardware access
 - C99 Data Types (int8_t, uint8_t, etc.)
 - Use of physical addresses for memory and hardware access
 - Heavy use of pointers to hardware addresses
 - Heavy use of bit operations to set/clear bits and test for bits

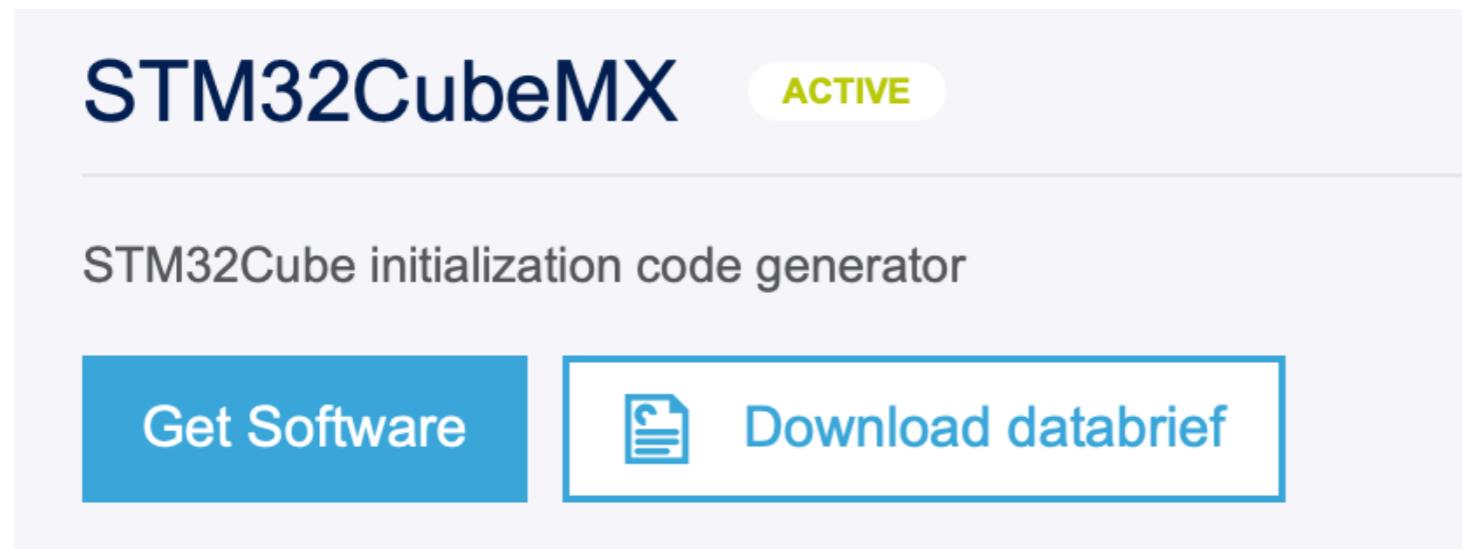
STM32CubeMX

STM32CubeMX

- STM32CubeMX is an Embedded C code generation tool
- We will use the tool to generate our first Embedded C code, then read/study/run the code
- Our approach
 - Install the tool
 - Generate a “Hello Embedded C” project
 - Build/Run on our STM32 Discovery Board

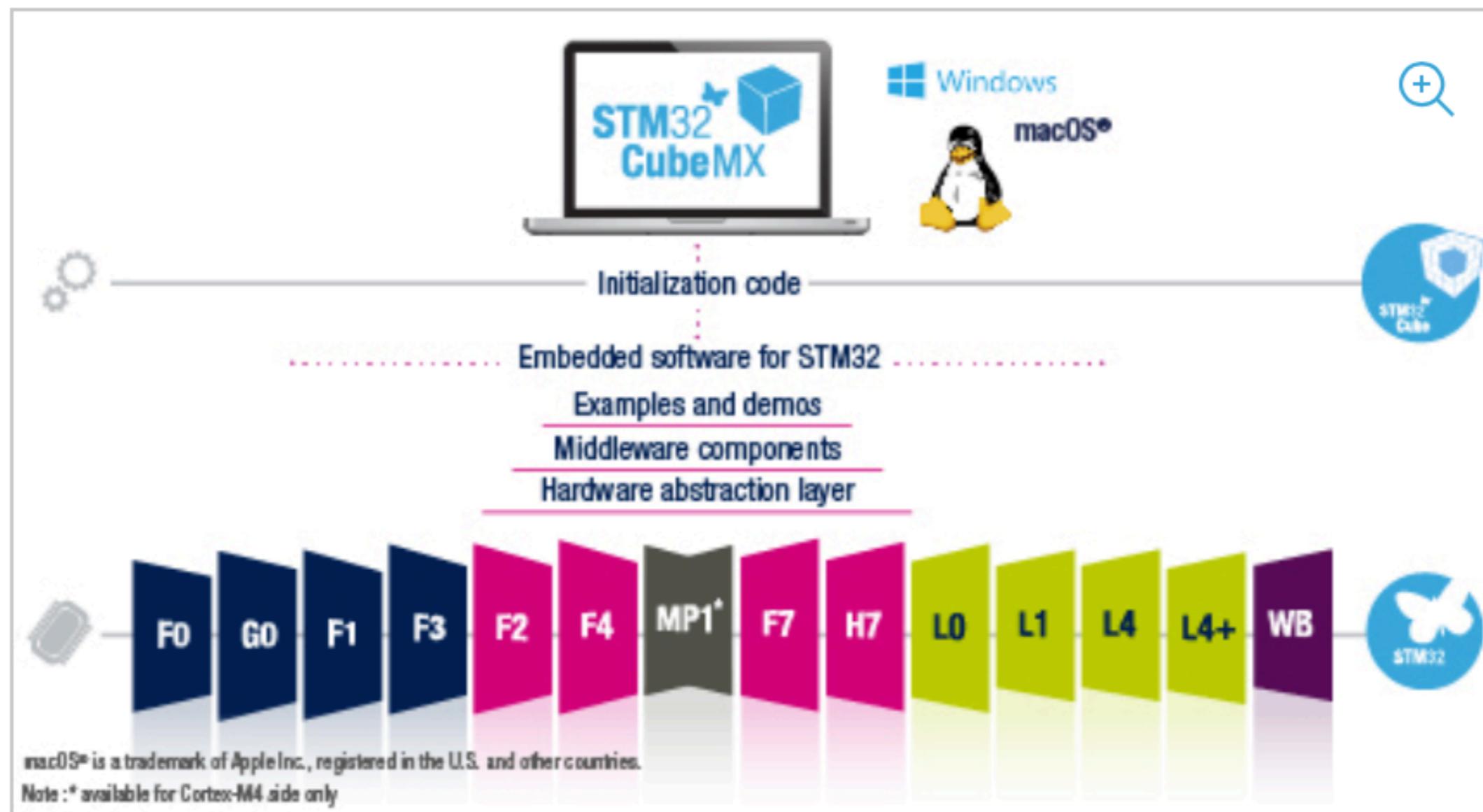
Step: Download and install STM32CubeMX

- <https://www.st.com/en/development-tools/stm32cubemx.html>

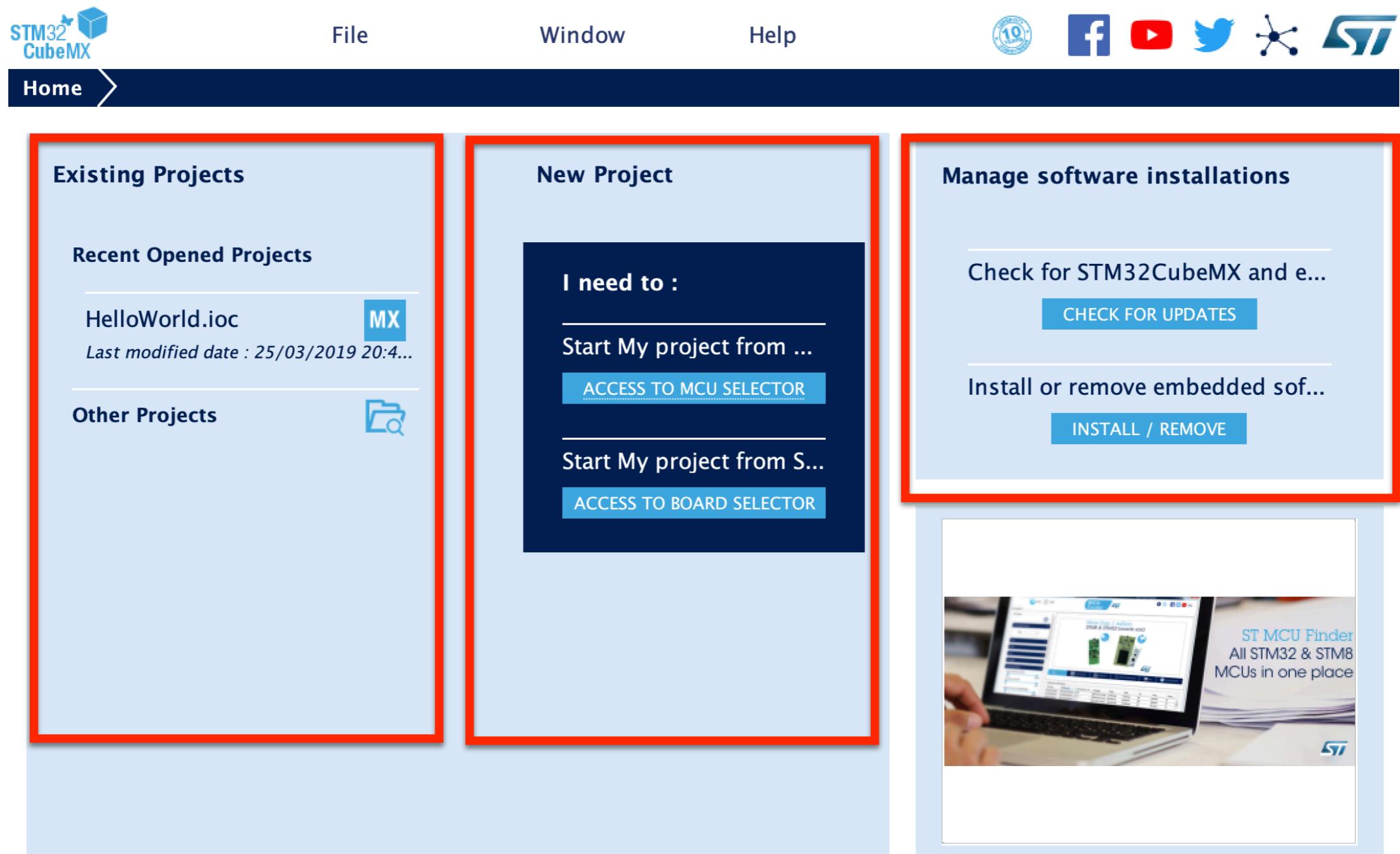


Note: STM32CubeMX works on:
Linux,
Mac,
and Windows
(Java-Based App)

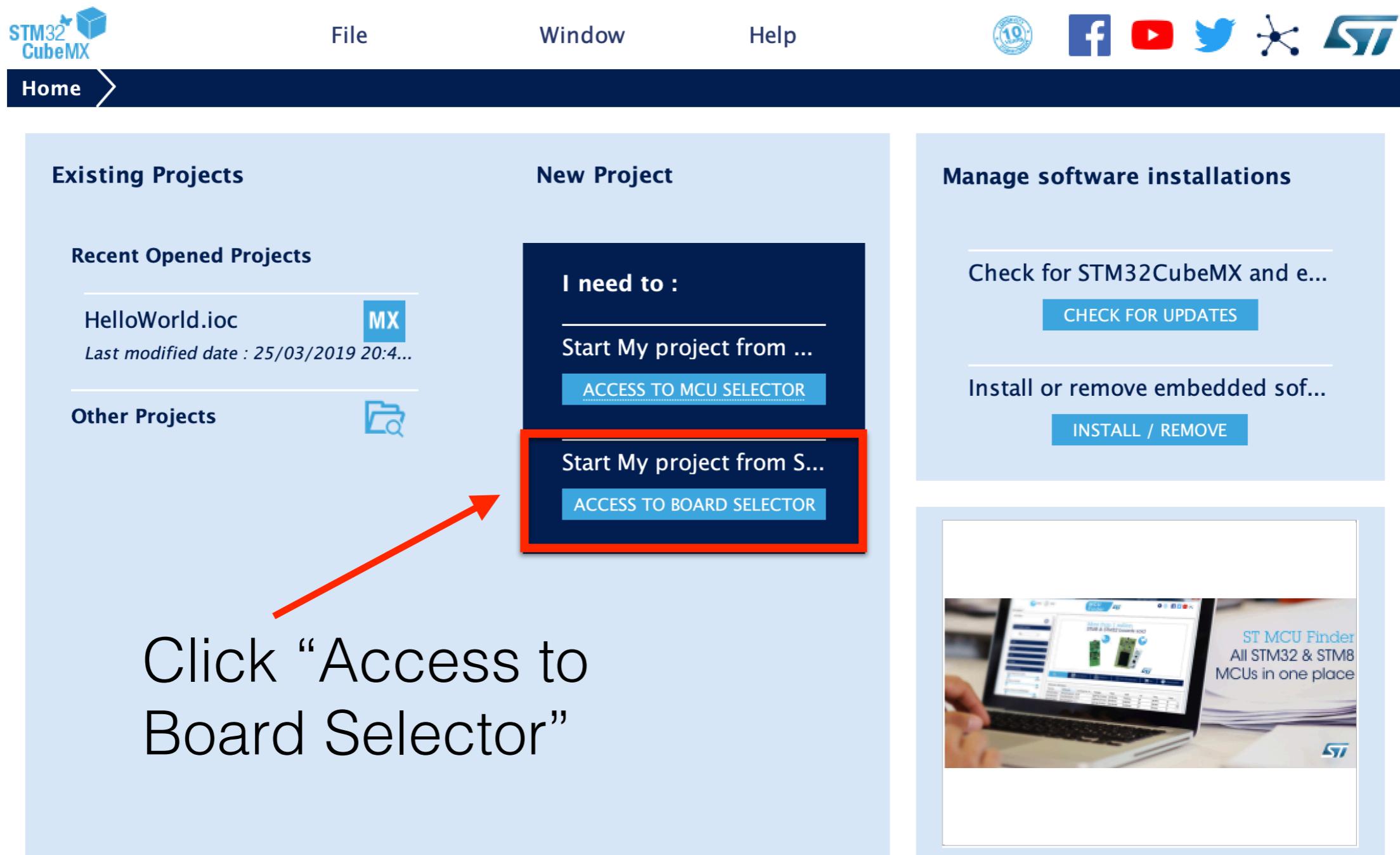
STM32CubeMX



Step: Startup STM32CubeMX



Step: Click on “Access to Board Selector”

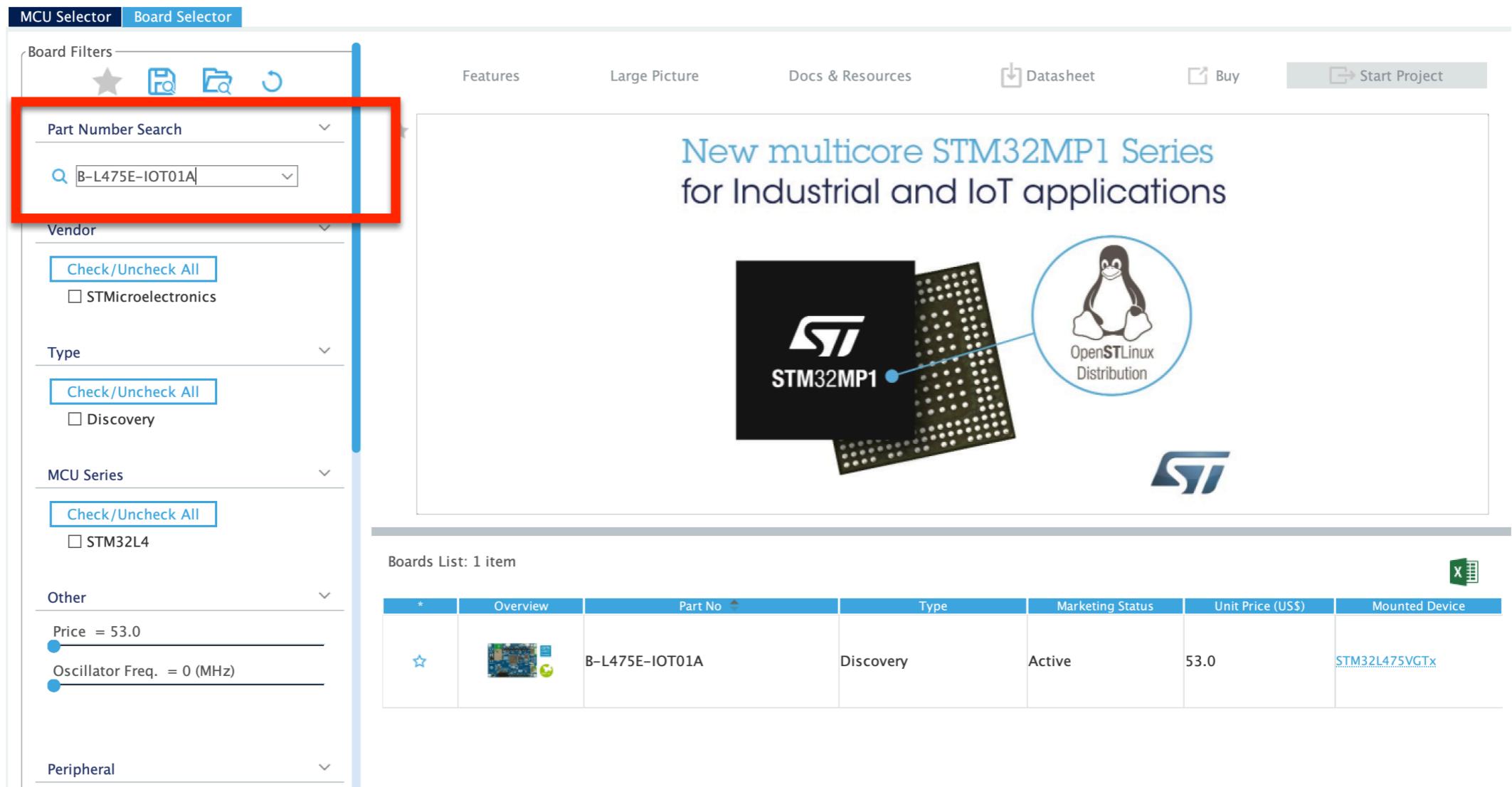


Step: Observe “Part Number Search”

The screenshot shows the STMicroelectronics website interface. On the left, there's a sidebar with filters for 'Board Filters' (Part Number Search, Vendor, Type), 'MCU Series' (Check/Uncheck All for STM32F0, STM32F1, STM32F2, STM32F3, STM32F4), and 'MCU Selector' (Board Selector). The main content area features a large image of an STM32MP1 chip with a Linux logo and the text 'New multicore STM32MP1 Series for Industrial and IoT applications'. Below this is a table titled 'Boards List: 107 items' with columns for Overview, Part No, Type, Marketing Status, Unit Price (US\$), and Mounted Device. Two rows are visible: one for '32F0308DISCOVERY' (Discovery, Active, \$8.9, STM32F030R8Tx) and another for '32F072BDISCOVERY' (Discovery, Active, \$10.4, STM32F072RBTx).

*	Overview	Part No	Type	Marketing Status	Unit Price (US\$)	Mounted Device
☆		32F0308DISCOVERY	Discovery	Active	8.9	STM32F030R8Tx
☆		32F072BDISCOVERY	Discovery	Active	10.4	STM32F072RBTx

Step: Enter "B-L475E-IOT01A"



The screenshot shows the STMicroelectronics Board Selector interface. On the left, there are various filters: Part Number Search (with a red box around it), Vendor (STMicroelectronics selected), Type (Discovery selected), MCU Series (STM32L4 selected), Other (Price = 53.0, Oscillator Freq. = 0 MHz), and Peripheral. The main area displays a banner for the New multicore STM32MP1 Series for Industrial and IoT applications, featuring an STM32MP1 chip and a Linux penguin icon. Below the banner is a table titled 'Boards List: 1 item' with one row containing the information for the B-L475E-IOT01A board.

*	Overview	Part No	Type	Marketing Status	Unit Price (US\$)	Mounted Device
		B-L475E-IOT01A	Discovery	Active	53.0	STM32L475VGTx

Step: Click on Image, and observe “Features”

MCU Selector Board Selector

Board Filters

- Part Number Search: B-L475E-IOT01A
- Vendor: STMicroelectronics
- Type: Discovery
- MCU Series: STM32L4
- Other: Price = 53.0, Oscillator Freq. = 0 (MHz)
- Peripheral:

Features Large Picture Docs & Resources Datasheet Buy Start Project

B-L475E-IOT01A

STM32 L4

STMicroelectronics B-L475E-IOT01A IOT Discovery Board Support and Examples

ACTIVE Active Product is in mass production

Unit Price (US\$) : 53.0

Mounted device: STM32L475VGTx

The B-L475E-IOT01A Discovery kit for IoT node allows users to develop applications with direct connection to cloud servers. The Discovery kit enables a wide diversity of applications by exploiting low-power communication, multiway sensing and ARM Cortex -M4 core-based STM32L4 Series features. The support for Arduino Uno V3 and PMOD connectivity provides unlimited expansion capabilities with a large choice of specialized add-on boards.

Features

- On-board ST-LINK/V2-1
- Supply through ST-Link USB
- USB OTG(Full speed) with micro AB Connector

Boards List: 1 item

*	Overview	Part No	Type	Marketing Status	Unit Price (US\$)	Mounted Device
★		B-L475E-IOT01A	Discovery	Active	53.0	STM32L475VGTx

Step: Click on “Large Picture”

MCU Selector Board Selector

Board Filters

- Part Number Search: B-L475E-IOT01A
- Vendor: Check/Uncheck All (unchecked)
- Type: Check/Uncheck All (unchecked)
- MCU Series: Check/Uncheck All (unchecked)
- Other: Price = 53.0, Oscillator Freq. = 0 (MHz)
- Peripheral

Features: Large Picture (highlighted with a red box)

B-L475E-IOT01A

Large Picture

Docs & Resources

Datasheet Buy Start Project

RoHS COMPLIANT 2002/95/EC

Boards List: 1 item

*	Overview	Part No.	Type	Marketing Status	Unit Price (US\$)	Mounted Device
☆		B-L475E-IOT01A	Discovery	Active	53.0	STM32L475VGTx

Step: Click on “Docs and Resources”

The screenshot shows a web-based board selector interface for the B-L475E-IOT01A. The left sidebar contains various filters:

- Board Filters: Includes a star icon, a file icon, a magnifying glass icon, and a refresh icon.
- Part Number Search: Set to B-L475E-IOT01A.
- Vendor: STMicroelectronics (selected).
- Type: Discovery (selected).
- MCU Series: STM32L4 (selected).
- Other: Price = 53.0, Oscillator Freq. = 0 (MHz).
- Peripheral: (empty dropdown).

The main content area displays the product details for B-L475E-IOT01A:

- Features: Large Picture, Docs & Resources (highlighted with a red box), Datasheet, Buy, Start Project.
- Data brief: DB3143 (Discovery kit for IoT node, multi-channel communication with STM32L4 (version 4)).
- User manual: UM2153 (Discovery kit for IoT node, multi-channel communication with STM32L4 (version 4)).

At the bottom, there is a Boards List table:

*	Overview	Part No	Type	Marketing Status	Unit Price (US\$)	Mounted Device
☆		B-L475E-IOT01A	Discovery	Active	53.0	STM32L475VGTx

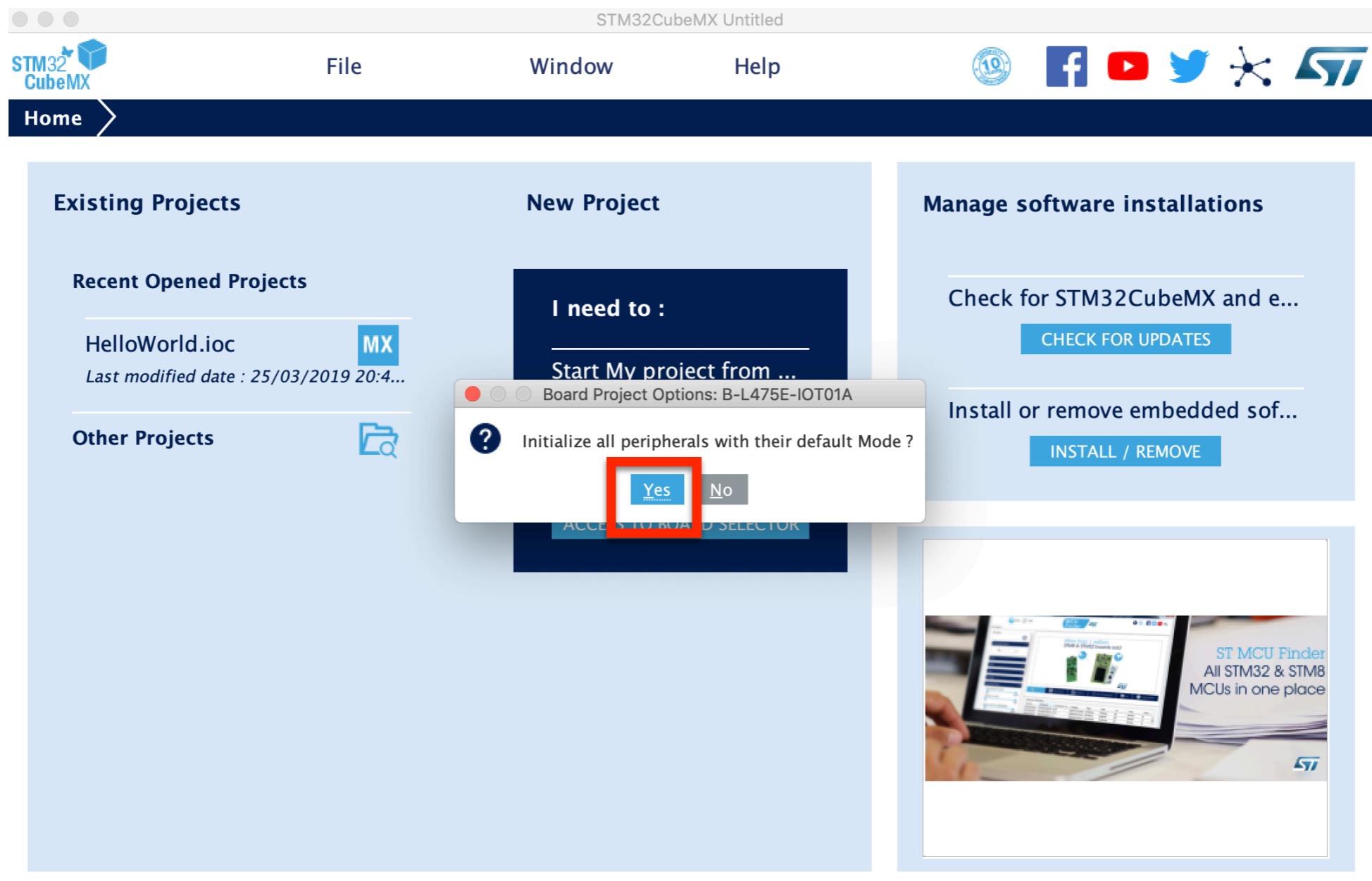
Step: Click on “Start Project”

The screenshot shows a web-based interface for selecting a development board. The top navigation bar includes 'MCU Selector' and 'Board Selector'. On the left, there are several filter sections: 'Board Filters' with icons for star, favorite, search, and refresh; 'Part Number Search' with a dropdown set to 'B-L475E-IOT01A'; 'Vendor' with a 'Check/Uncheck All' button and a checked 'STMicroelectronics' checkbox; 'Type' with a 'Check/Uncheck All' button and an unchecked 'Discovery' checkbox; 'MCU Series' with a 'Check/Uncheck All' button and an unchecked 'STM32L4' checkbox; 'Other' with sliders for 'Price = 53.0' and 'Oscillator Freq. = 0 (MHz)'; and 'Peripheral'.

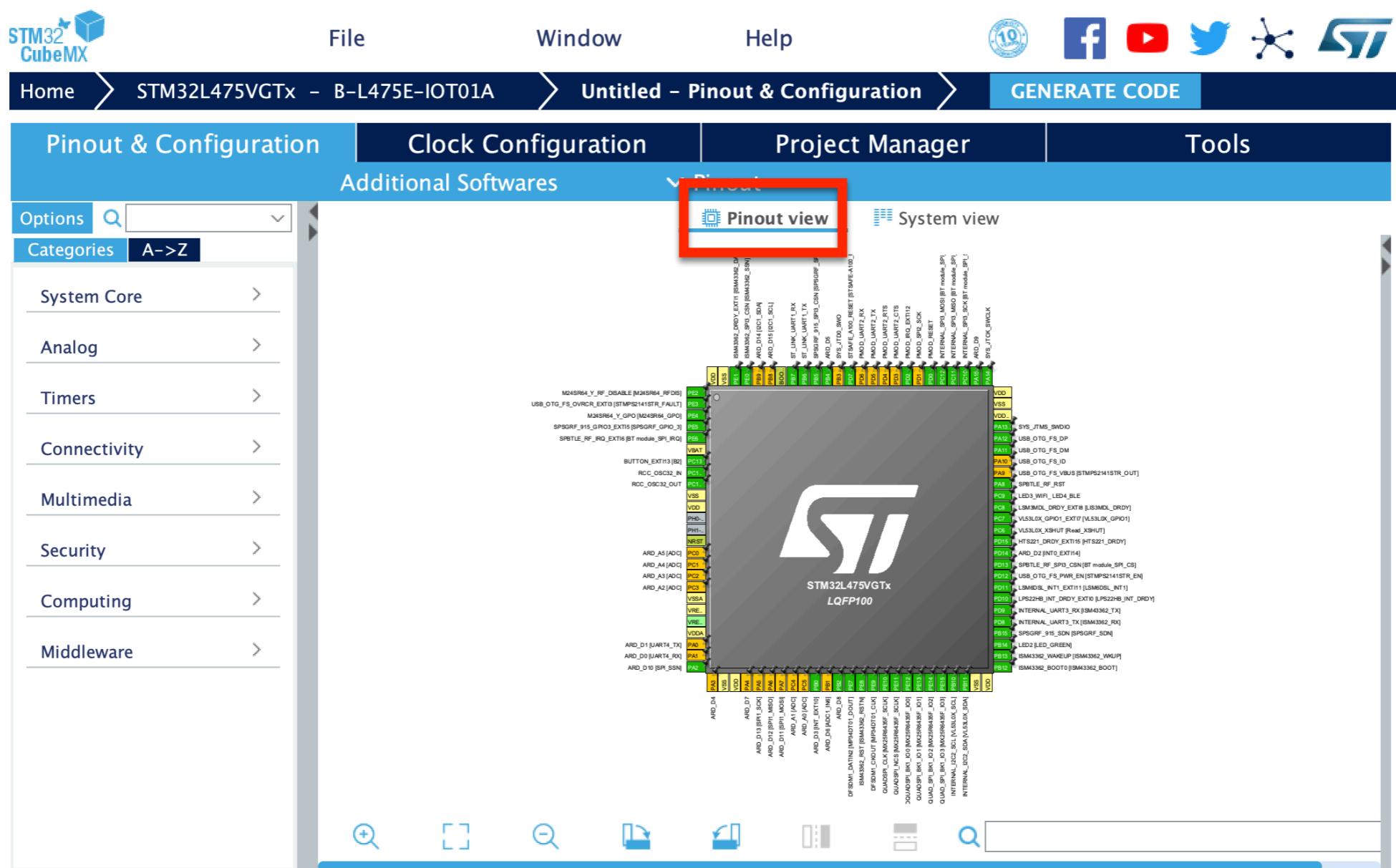
The main content area features tabs for 'Features', 'Large Picture', 'Docs & Resources' (which is underlined in blue), 'Datasheet', 'Buy', and 'Start Project' (which is highlighted with a red box). Below these tabs, the board is identified as 'B-L475E-IOT01A'. It lists 'Data brief' (DB3143) and 'User manual' (UM2153), both described as 'Discovery kit for IoT node, multi-channel communication with STM32L4 (version 4)'.

At the bottom, a 'Boards List: 1 item' section displays a table with one row for the B-L475E-IOT01A. The columns are: * (with a star icon), Overview (with a thumbnail image), Part No (B-L475E-IOT01A), Type (Discovery), Marketing Status (Active), Unit Price (US\$) (53.0), and Mounted Device (STM32L475VGTx).

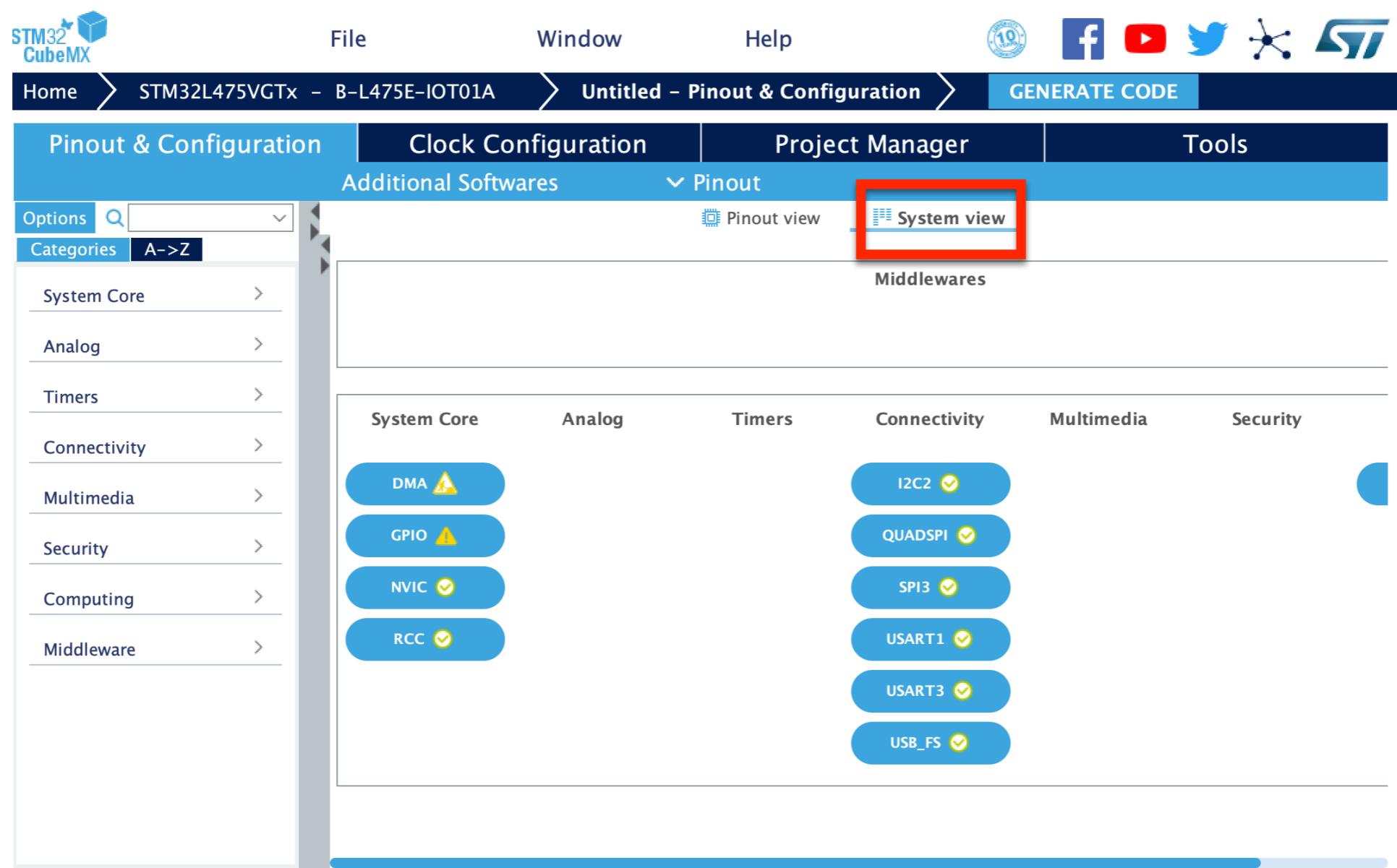
Step: Click on “Yes” (Initialize all ...with default mode)



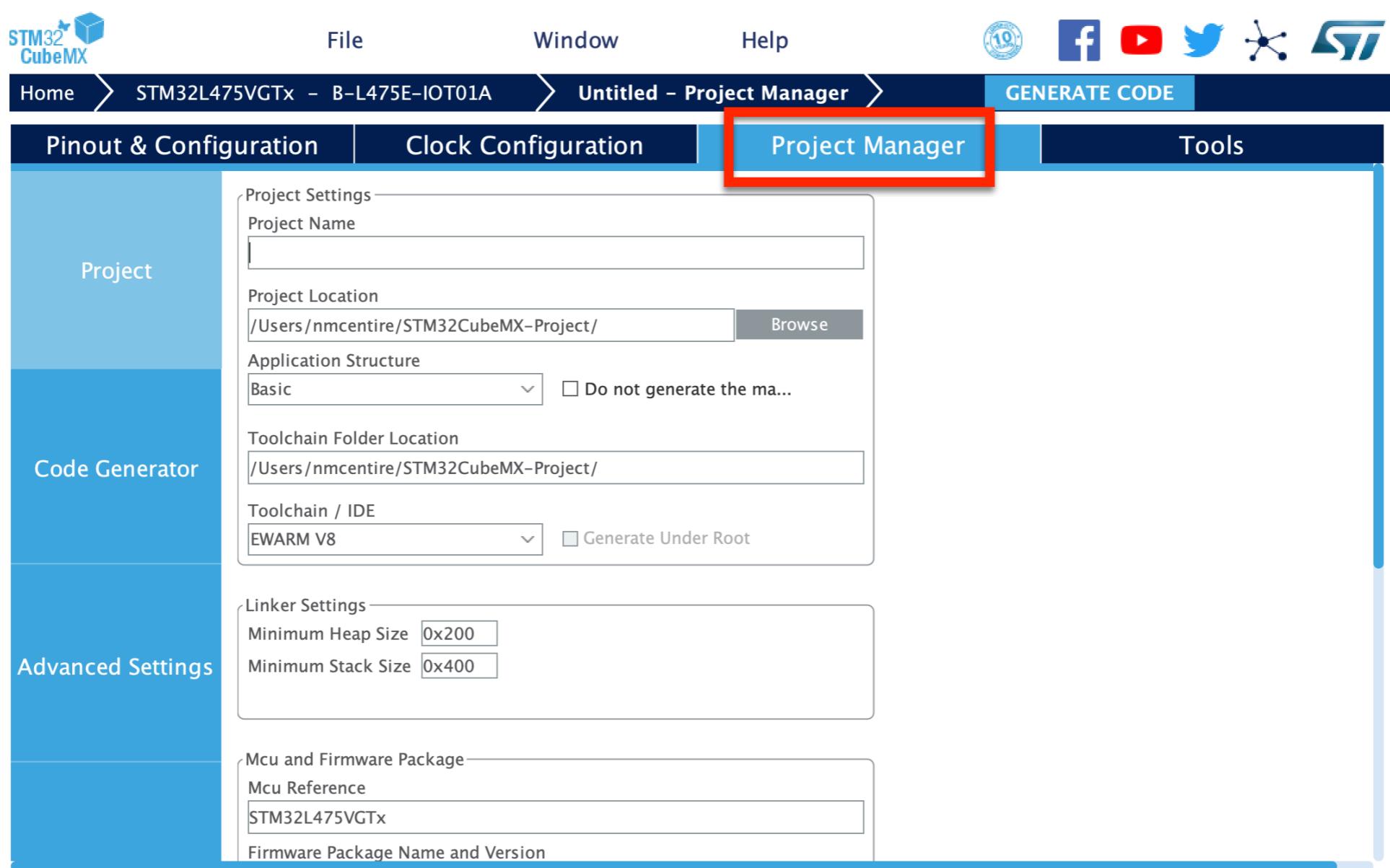
Step: Observe “Pinout View”



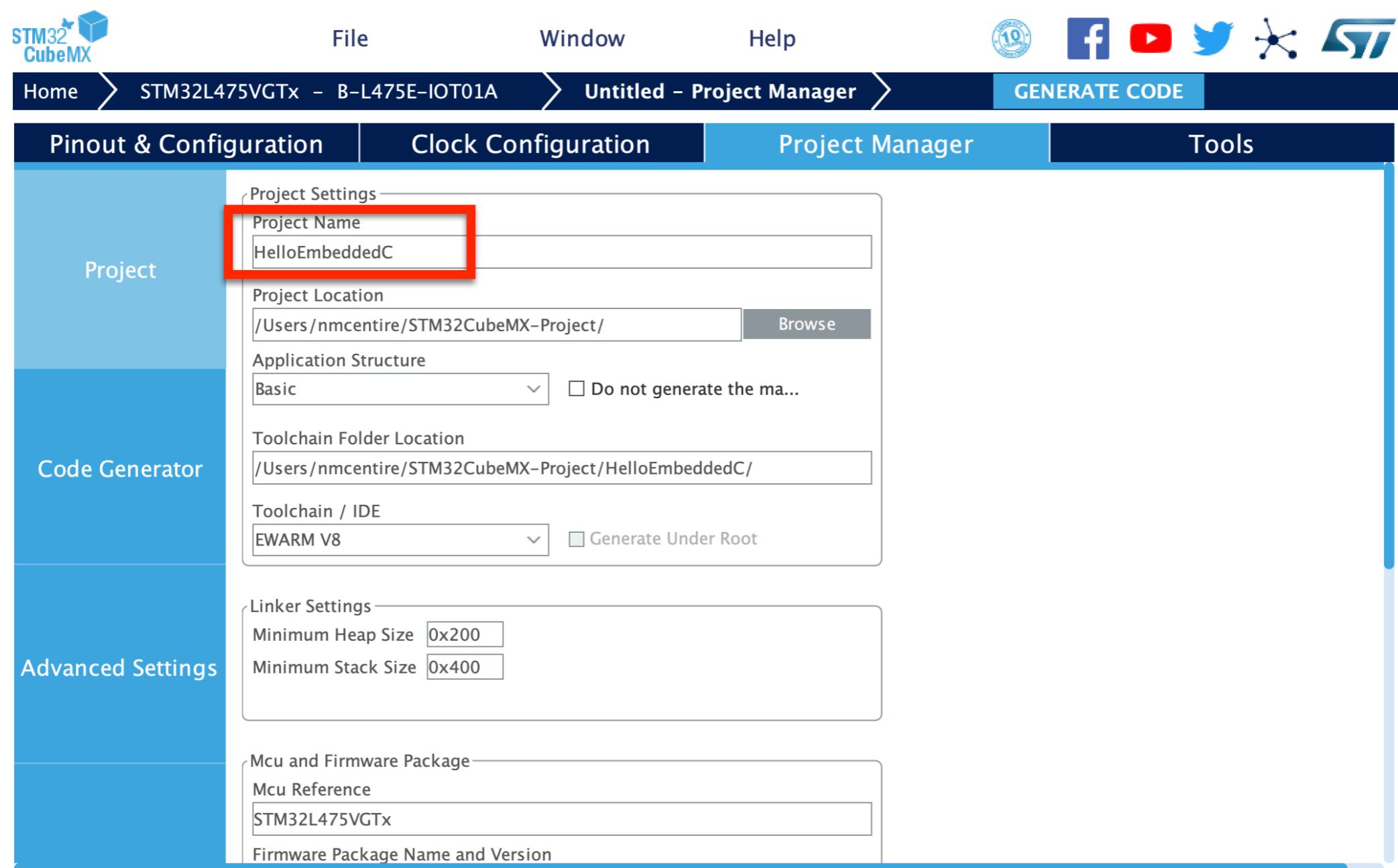
Step: Observe “System View”



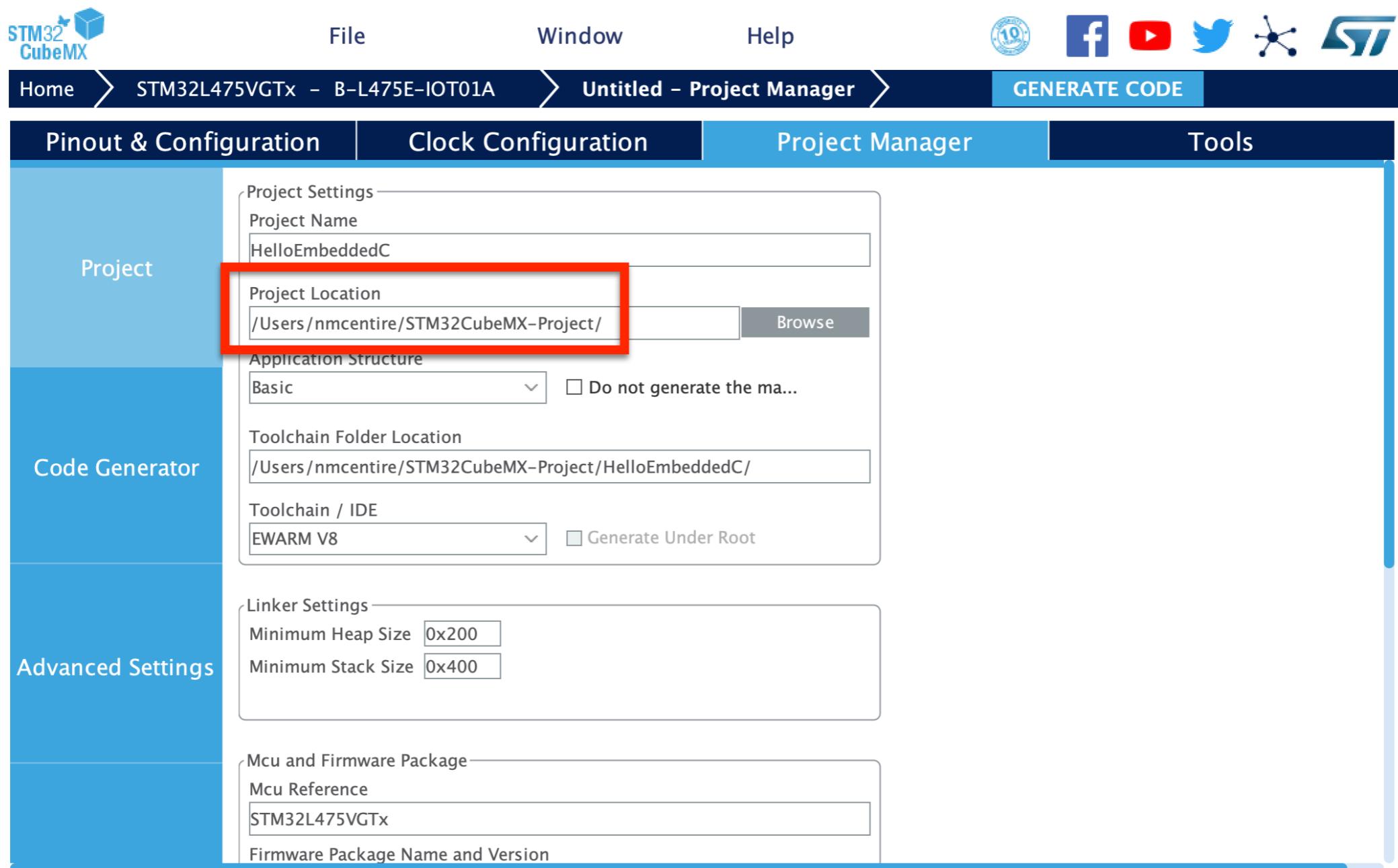
Step: Observe “Project Manager”



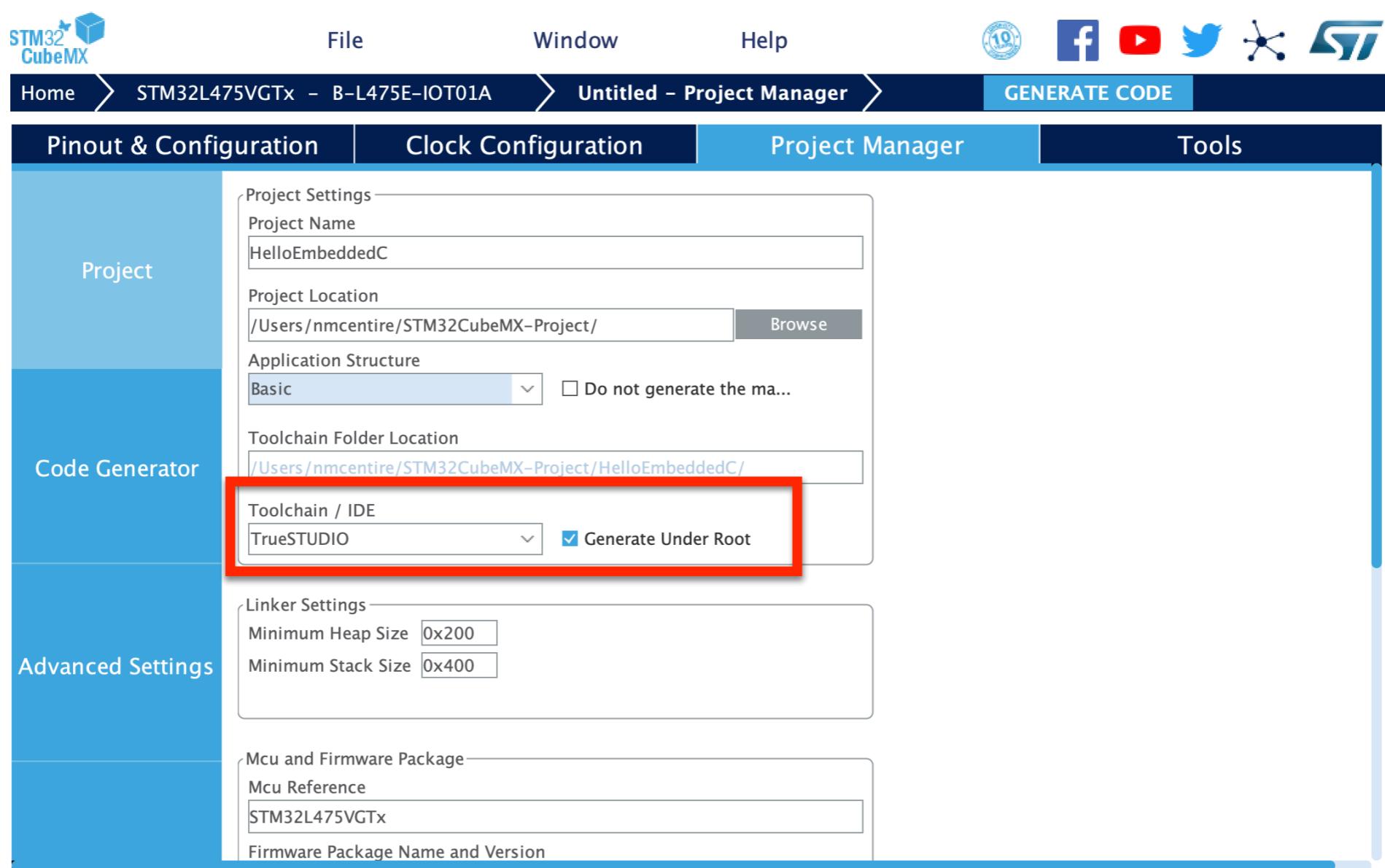
Step: Enter “HelloEmbeddedC” for Project Name



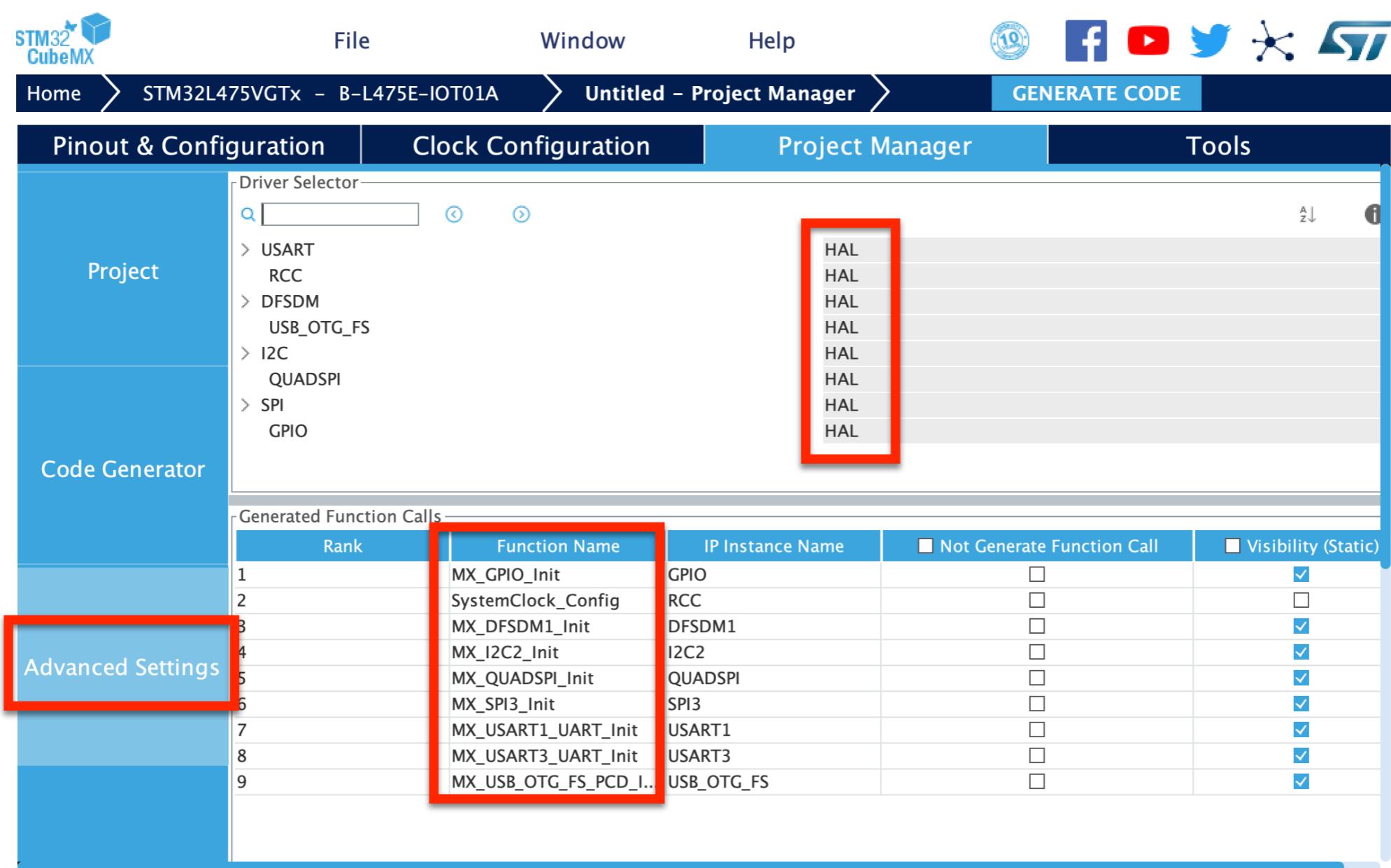
Step: Observe location where project will be stored



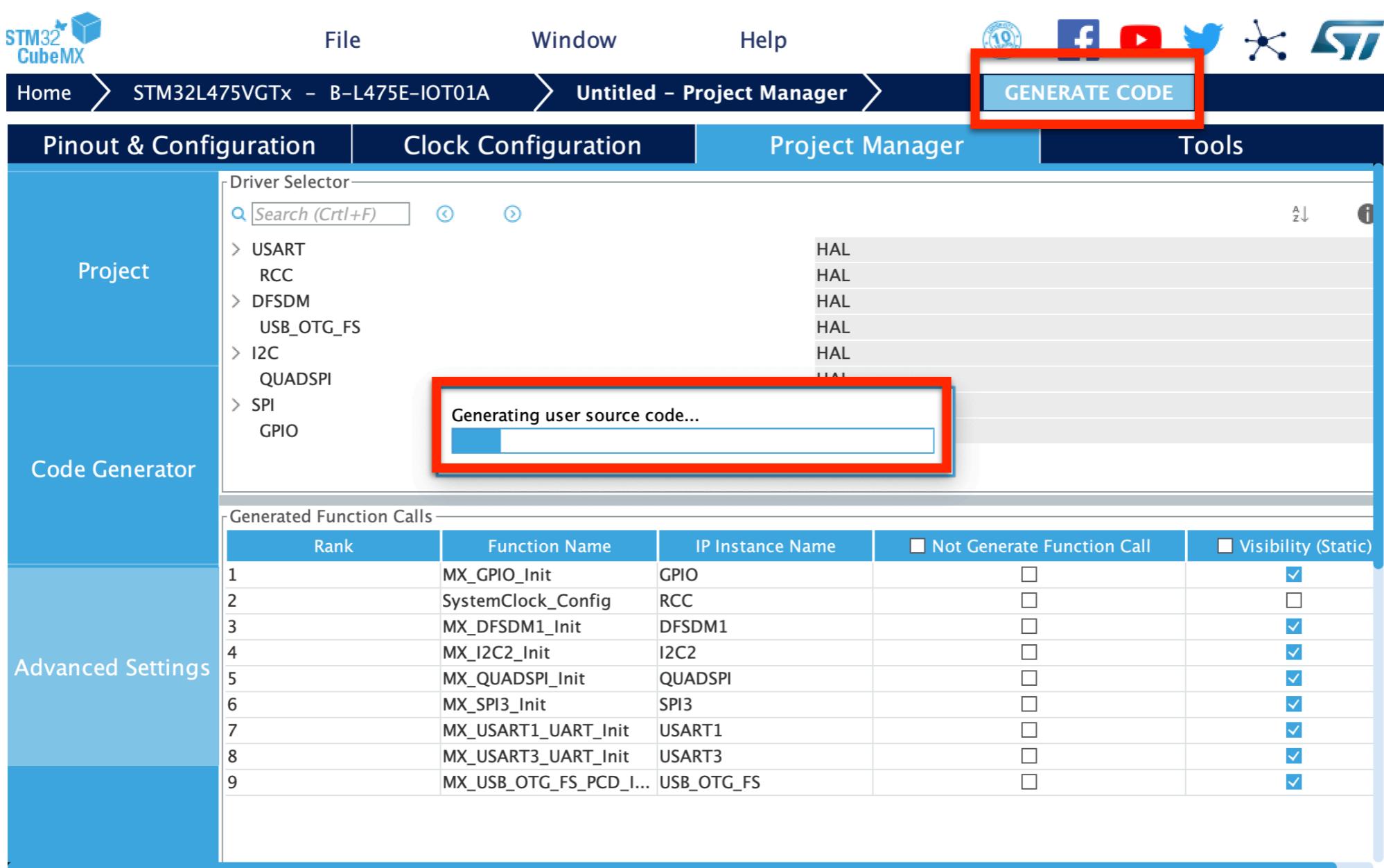
Step: Select “TrueStudio” for Toolchain / IDE



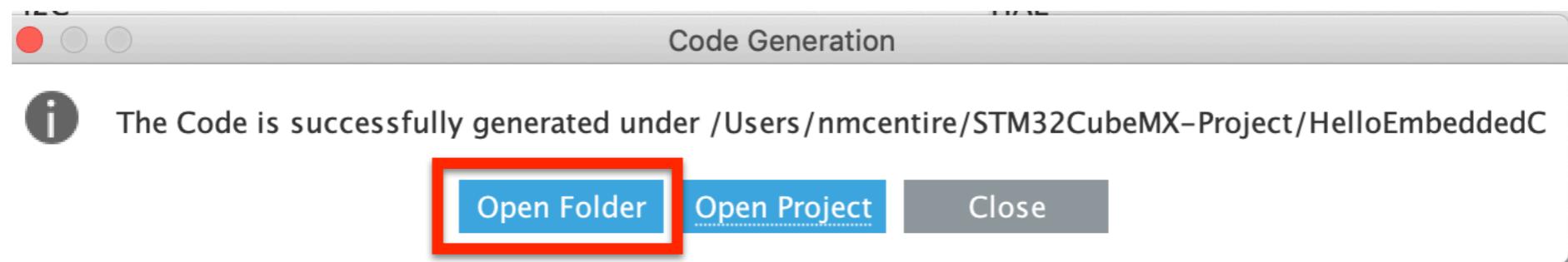
Step: Click on “Advanced Settings”, observing HAL and “Function Name”



Step: Click on “Generate Code”

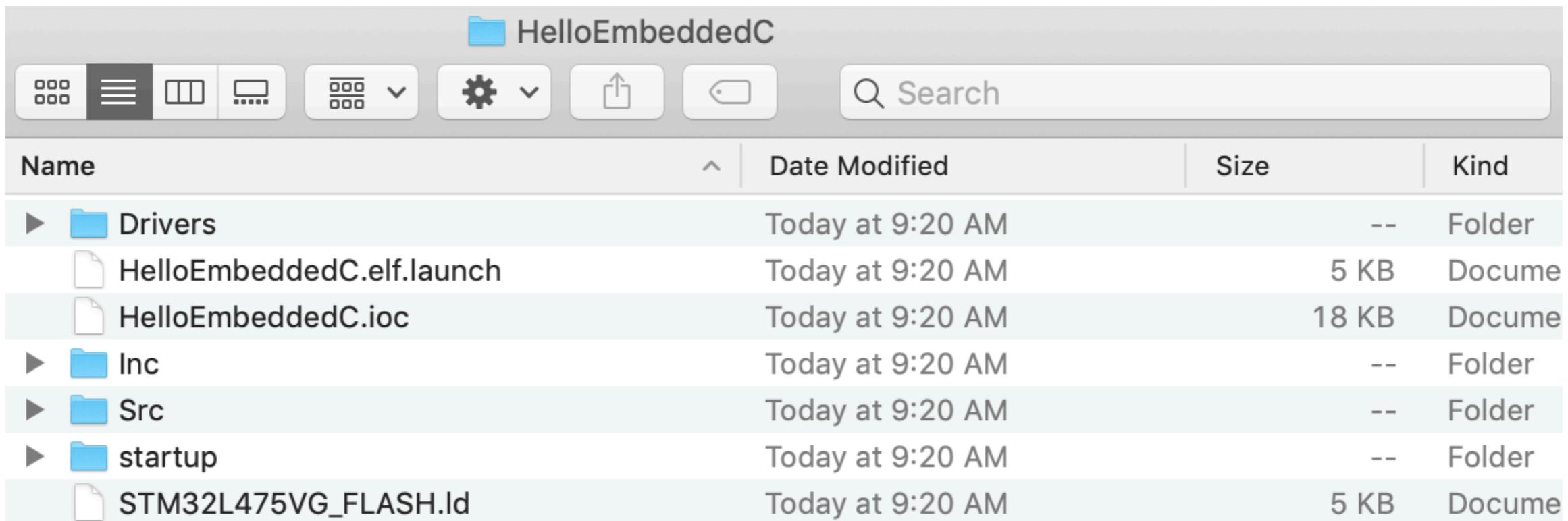


Step: Select “Open Folder”



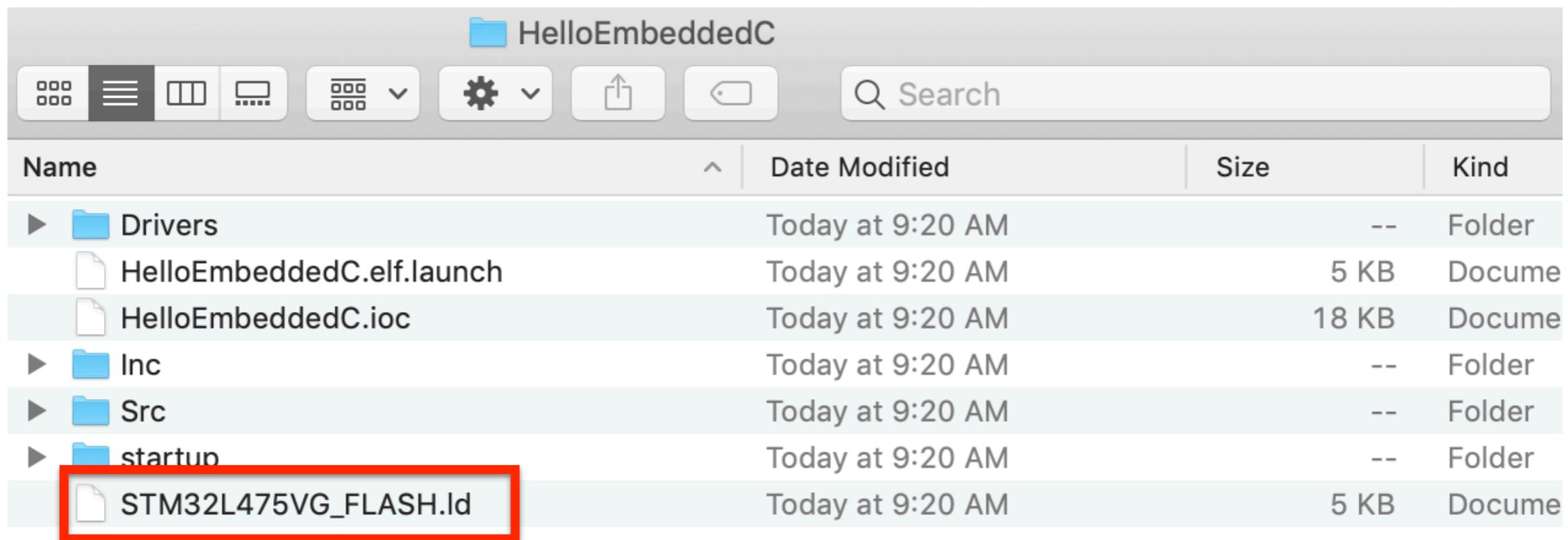
Tour of Generated Project

Step: View Open Folder



Note: The following slides will give a brief tour of key files/directories in the project.

Step: View STM32L475VG_FLASH.ld File (view as text file)



This is the “linker” file. See next slide.

Step: Linker Script - Part 1

```
e  ** File      : stm32_flash.ld
e  **
e  ** Abstract   : Linker script for STM32L475VG Device with
e  **                 1024KByte FLASH, 128KByte RAM
B:  **
B:  ** Set heap size, stack size and stack location according
B:  ** to application requirements.
B:  **
B:  ** Set memory bank area and size if external memory is used.
B:  **
B:  ** Target     : STMicroelectronics STM32
```

Key Points:

- =1= Linker script tied to specific STM32 device.
- =2= Heap Size (where memory allocated from)
- =3= Stack Size (for local variables in functions)
- =4= Notice comment about “Memory Bank”

Step: Linker Script - Part 2

```
/* Entry Point */
ENTRY(Reset_Handler)

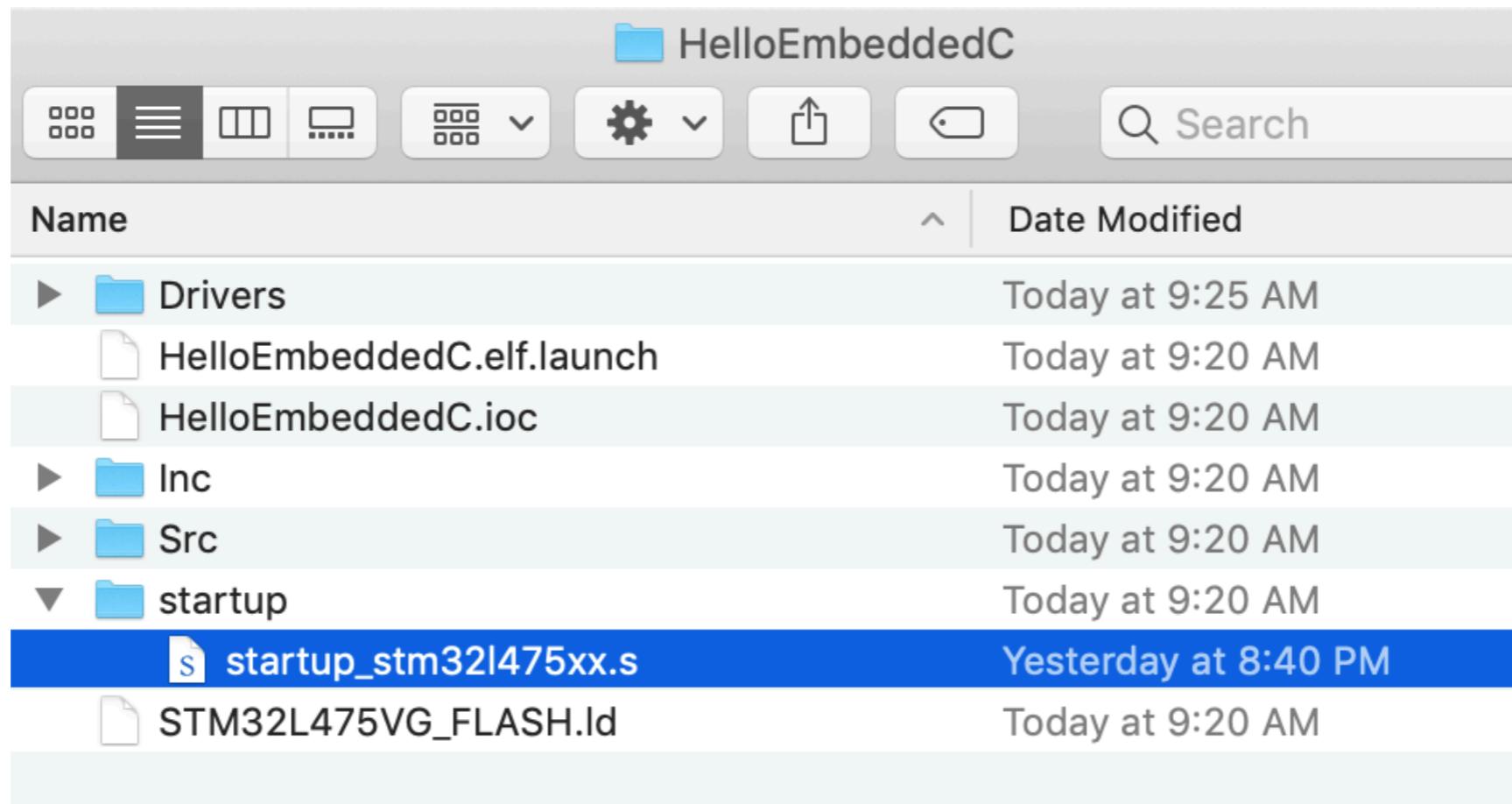
/* Highest address of the user mode stack */
estack = 0x20018000;      /* end of RAM */
/* Generate a link error if heap and stack don't fit into RAM */
_Min_Heap_Size = 0x200;      /* required amount of heap  */
_Min_Stack_Size = 0x400; /* required amount of stack */

/* Specify the memory areas */
MEMORY
{
    RAM (xrw)      : ORIGIN = 0x20000000, LENGTH = 96K
    RAM2 (xrw)     : ORIGIN = 0x10000000, LENGTH = 32K
    FLASH (rx)     : ORIGIN = 0x8000000, LENGTH = 1024K
}
```

Key Points:

- =1= At powerup/reset, ENTRY is Reset_Handler
- =2= 0x2001_8000 as end of RAM for top of stack
- =3= Min Heap and Stack Sizes
- =4= Memory Addresses, RAM, RAM2, FLASH

Step: View “startup_stm32l475xx.s”



Note: .s is assembly language code!
See next slide

Step: View “startup_stm32l475xx.s”

```
Reset_Handler:  
    ldr    sp, =_estack /* Atollic update: set stack pointer */  
  
/* Copy the data segment initializers from flash to SRAM */  
    movs   r1, #0  
    b      LoopCopyDataInit  
  
CopyDataInit:  
    ldr    r3, =_sidata  
    ldr    r3, [r3, r1]  
    str    r3, [r0, r1]  
    adds   r1, r1, #4  
  
LoopCopyDataInit:  
    ldr    r0, =_sdata  
    ldr    r3, =_edata  
    adds   r2, r0, r1  
    cmp    r2, r3  
    bcc   CopyDataInit  
    ldr    r2, =_sbss  
    b      LoopFillZeroBSS  
  
/* Zero fill the bss segment. */  
FillZeroBSS:  
    movs   r3, #0  
    str    r3, [r2], #4  
  
LoopFillZeroBSS:  
    ldr    r3, =_ebss  
    cmp    r2, r3  
    bcc   FillZeroBSS  
  
/* Call the clock system intitialization function.*/  
    bl    SystemInit  
/* Call static constructors */  
    bl    __libc_init_array  
/* Call the application's entry point.*/  
    bl    main
```

Code starts executing here

Code jumps to main()
Embedded C code

Step: View “Src” directory



Step: View “Inc” directory



Step: View “Inc/main.h”

```
/* Private defines -----  
#define M24SR64_Y_RF_DISABLE_Pin GPIO_PIN_2  
#define M24SR64_Y_RF_DISABLE_GPIO_Port GPIOE  
#define USB_OTG_FS_OVRCR_EXTI3_Pin GPIO_PIN_3  
#define USB_OTG_FS_OVRCR_EXTI3_GPIO_Port GPIOE  
#define M24SR64_Y_GPO_Pin GPIO_PIN_4  
#define M24SR64_Y_GPO_GPIO_Port GPIOE  
#define SPSGRF_915_GPIO3_EXTI5_Pin GPIO_PIN_5  
#define SPSGRF_915_GPIO3_EXTI5_GPIO_Port GPIOE  
#define SPSGRF_915_GPIO3_EXTI5_EXTI IRQn EXTI9_5_IRQn  
#define SPBTLE_RF_IRQ_EXTI6_Pin GPIO_PIN_6  
#define SPBTLE_RF_IRQ_EXTI6_GPIO_Port GPIOE  
#define SPBTLE_RF_IRQ_EXTI6_EXTI IRQn EXTI9_5_IRQn  
#define BUTTON_EXTI13_Pin GPIO_PIN_13  
#define BUTTON_EXTI13_GPIO_Port GPIOC  
#define BUTTON_EXTI13_EXTI IRQn EXTI15_10_IRQn  
#define ARD_A5_Pin GPIO_PIN_0  
#define ARD_A5_GPIO_Port GPIOC  
#define ARD_A4_Pin GPIO_PIN_1  
#define ARD_A4_GPIO_Port GPIOC  
#define ARD_A3_Pin GPIO_PIN_2  
#define ARD_A3_GPIO_Port GPIOC  
#define ARD_A2_Pin GPIO_PIN_3  
#define ARD_A2_GPIO_Port GPIOC  
#define ARD_D1_Pin GPIO_PIN_0  
#define ARD_D1_GPIO_Port GPIOA  
#define ARD_DD_Pin GPIO_PIN_1
```

Note: These are
The GPIO pin
definitions

Step: View “Inc/ stm32l4xx_hal_conf.h”

```
/* ##### Module Selection #####
/** @brief This is the list of modules to be used in the HAL driver */
#define HAL_MODULE_ENABLED
/*#define HAL_ADC_MODULE_ENABLED */
/*#define HAL_CRYPT_MODULE_ENABLED */
/*#define HAL_CAN_MODULE_ENABLED */
/*#define HAL_COMP_MODULE_ENABLED */
/*#define HAL_CRC_MODULE_ENABLED */
/*#define HAL_CRYPT_MODULE_ENABLED */
/*#define HAL_DAC_MODULE_ENABLED */
/*#define HAL_DCMI_MODULE_ENABLED */
/*#define HAL_DMA2D_MODULE_ENABLED */
#define HAL_DFSDM_MODULE_ENABLED
/*#define HAL_DSI_MODULE_ENABLED */
/*#define HAL_FIREWALL_MODULE_ENABLED */
```

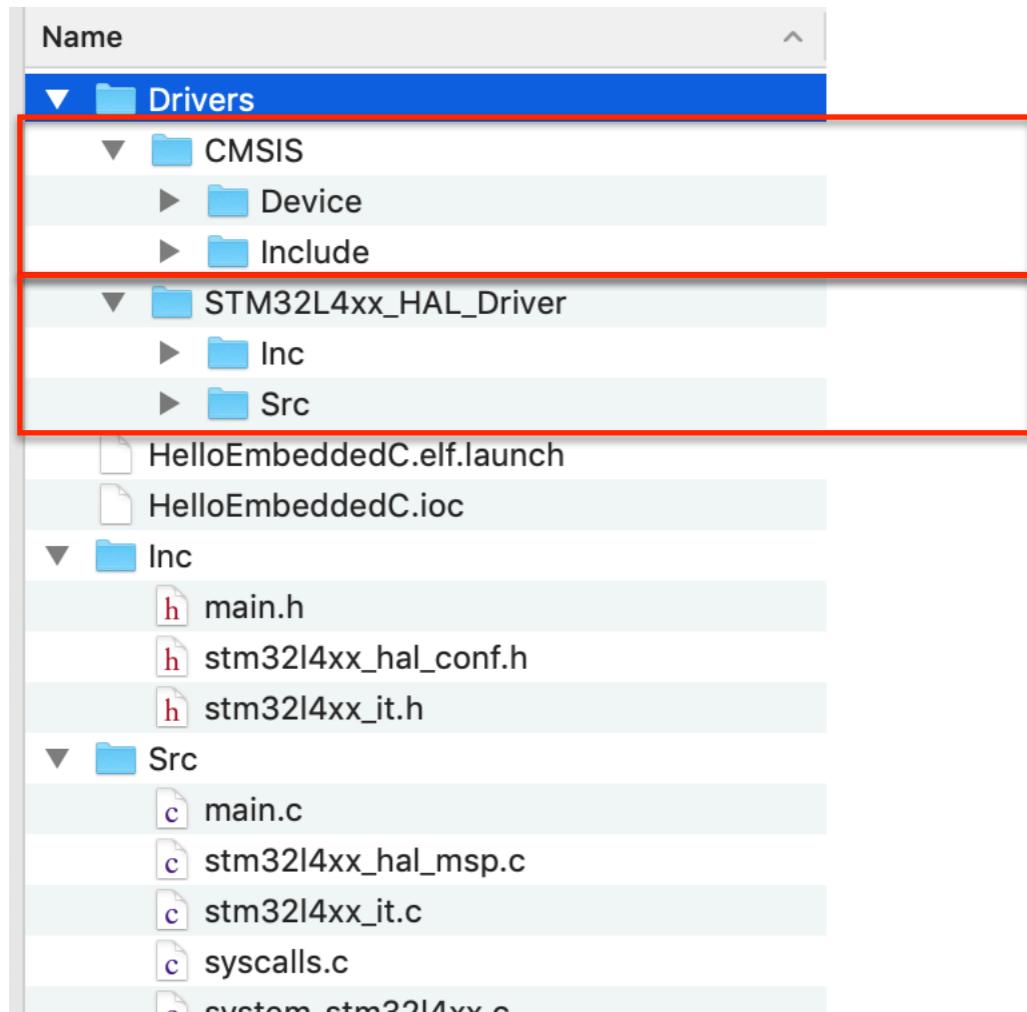
Note: These #defines determine what HAL features are include in build

Step: View “Inc/ stm32l4xx_it.h”

```
/* Exported functions prototypes -----  
void NMI_Handler(void);  
void HardFault_Handler(void);  
void MemManage_Handler(void);  
void BusFault_Handler(void);  
void UsageFault_Handler(void);  
void SVC_Handler(void);  
void DebugMon_Handler(void);  
void PendSV_Handler(void);  
void SysTick_Handler(void);  
void EXTI9_5_IRQHandler(void);  
void EXTI15_10_IRQHandler(void);  
/* USER CODE BEGIN EFP */
```

Note: These are function prototypes for the interrupt handlers

Step: View “Drivers” directory



CMSIS

Cortex M Software Interface Standard
(Vendor Independent HAL)

HAL

Hardware Abstraction Layer
(Vendor Dependent HAL)

<http://www.keil.com/pack/doc/CMSIS/General/html/index.html>

The screenshot shows the CMSIS Version 5.5.1 website. At the top, there is a green 'CMSIS COMPLIANT' logo with a checkmark icon. To its right, the text 'CMSIS Version 5.5.1' and 'Cortex Microcontroller Software Interface Standard' is displayed. Below this is a navigation bar with tabs: General, CMSIS-Core(A), CMSIS-Core(M), Driver, DSP, NN, RTOS v1, RTOS v2, Pack, SVD, DAP, and Zone. The 'General' tab is highlighted. Underneath the navigation bar, there are two buttons: 'Main Page' and 'Usage and Description'. The 'Usage and Description' button is currently selected. On the left side, there is a sidebar with a tree view. The 'Introduction' node is expanded, showing sub-options: Motivation, Coding Rules, Validation, License, and ARM::CMSIS Pack. The main content area is titled 'Introduction'. It contains text explaining the purpose of CMSIS and its benefits. The text states: 'The Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for generic tool interfaces. The CMSIS enables consistent device support and simple software interfaces to the processor and reducing the learning curve for microcontroller developers, and reducing the time to market for new devices.' It also mentions that CMSIS is defined in cooperation with various vendors and provides a common approach to systems and middleware components.

Vendor-Independent Hardware Abstraction Layer

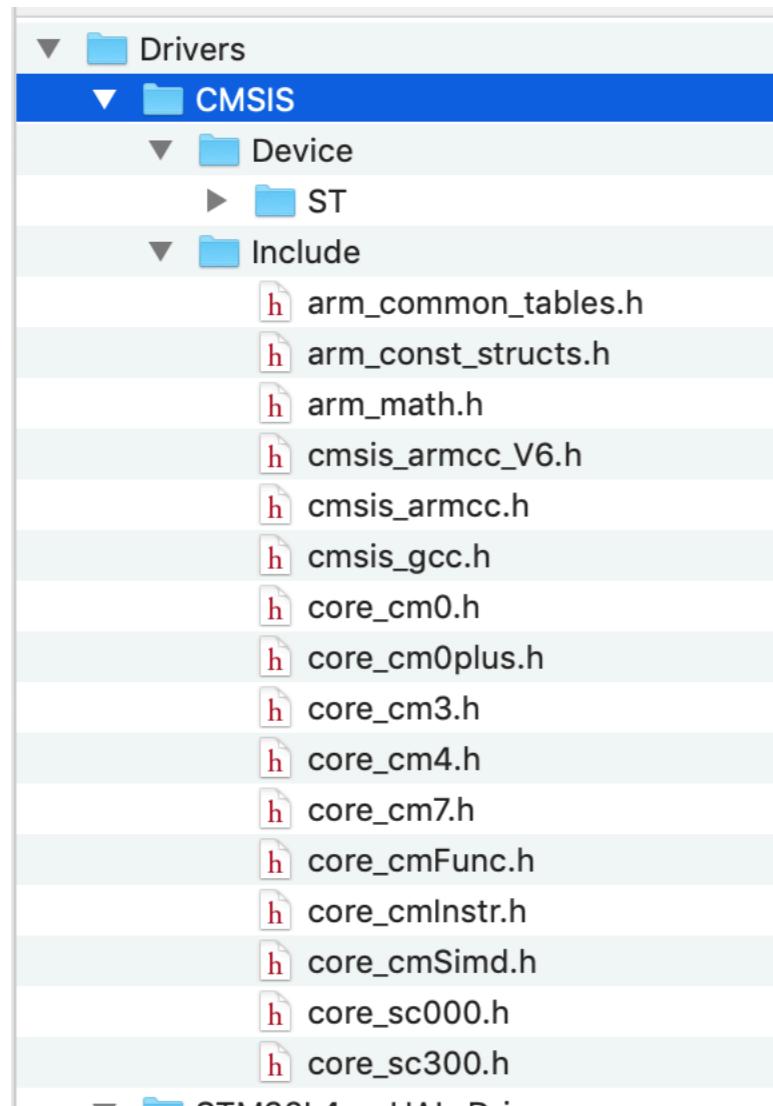
https://www.st.com/content/ccc/resource/technical/document/user_manual/63/a8/8f/e3/ca/a1/4c/84/DM00173145.pdf/files/DM00173145.pdf/jcr:content/translations/en.DM00173145.pdf



UM1884
User manual

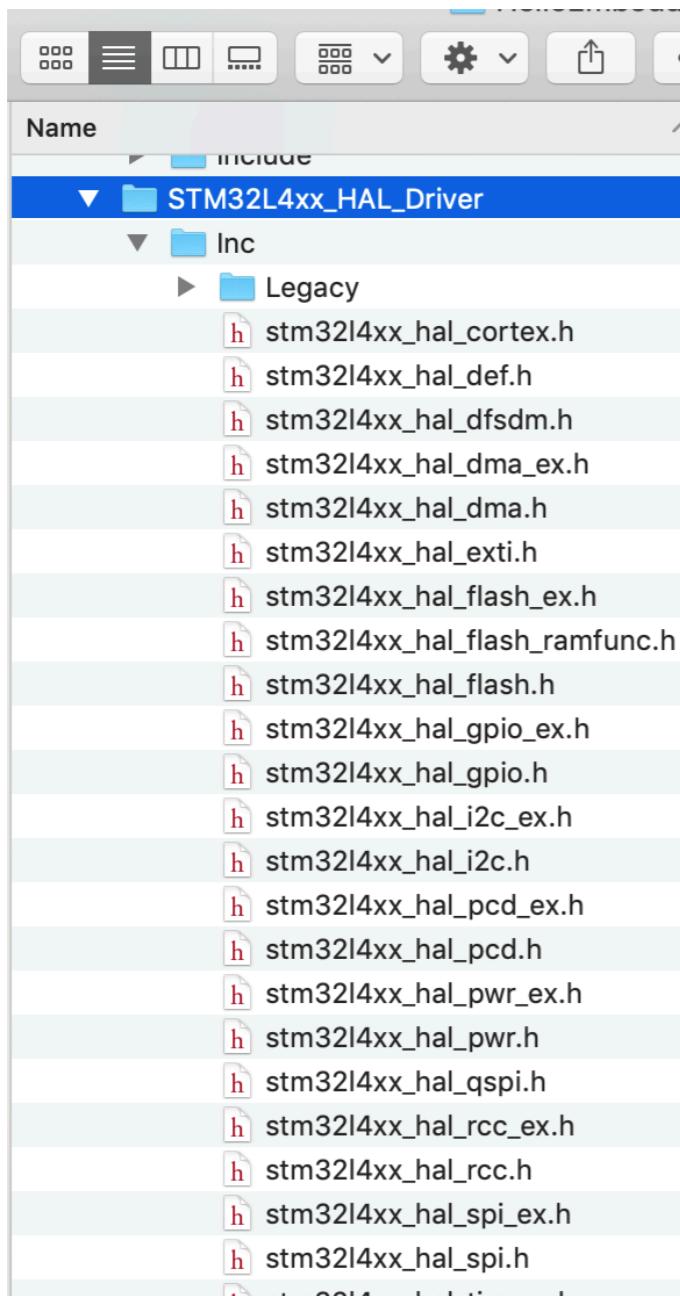
Description of STM32L4/L4+ HAL and low-layer drivers

Step: View “CMSIS API”



Notes:
These are focused on
the Cortex M processor
and not I/O devices.

Step: View “HAL API”



Notes:

- =1= stm32l4xx_hal_...
- =2= stm32l4xx_gpio.h
- =3= stm32l4xx_gpio_ex.h

Summary so far...

- STM32CubeMX generates Embedded C Code
- Code Well organized
 - Drivers - Pre-Written CMSIS and HAL Code
 - Inc - Config Settings
 - Src - You'll begin your coding with main.c
 - startup - The assembly language startup code

TrueStudio

Step: Install TrueStudio

- <https://atollic.com/truestudio/>

Note: TrueStudio works on Windows and Linux but not Mac



HOME TRUESTUDIO STM32 TARGET SUPPORT RESOURCES BLOG ABOUT US

TrueSTUDIO

Home / TrueSTUDIO

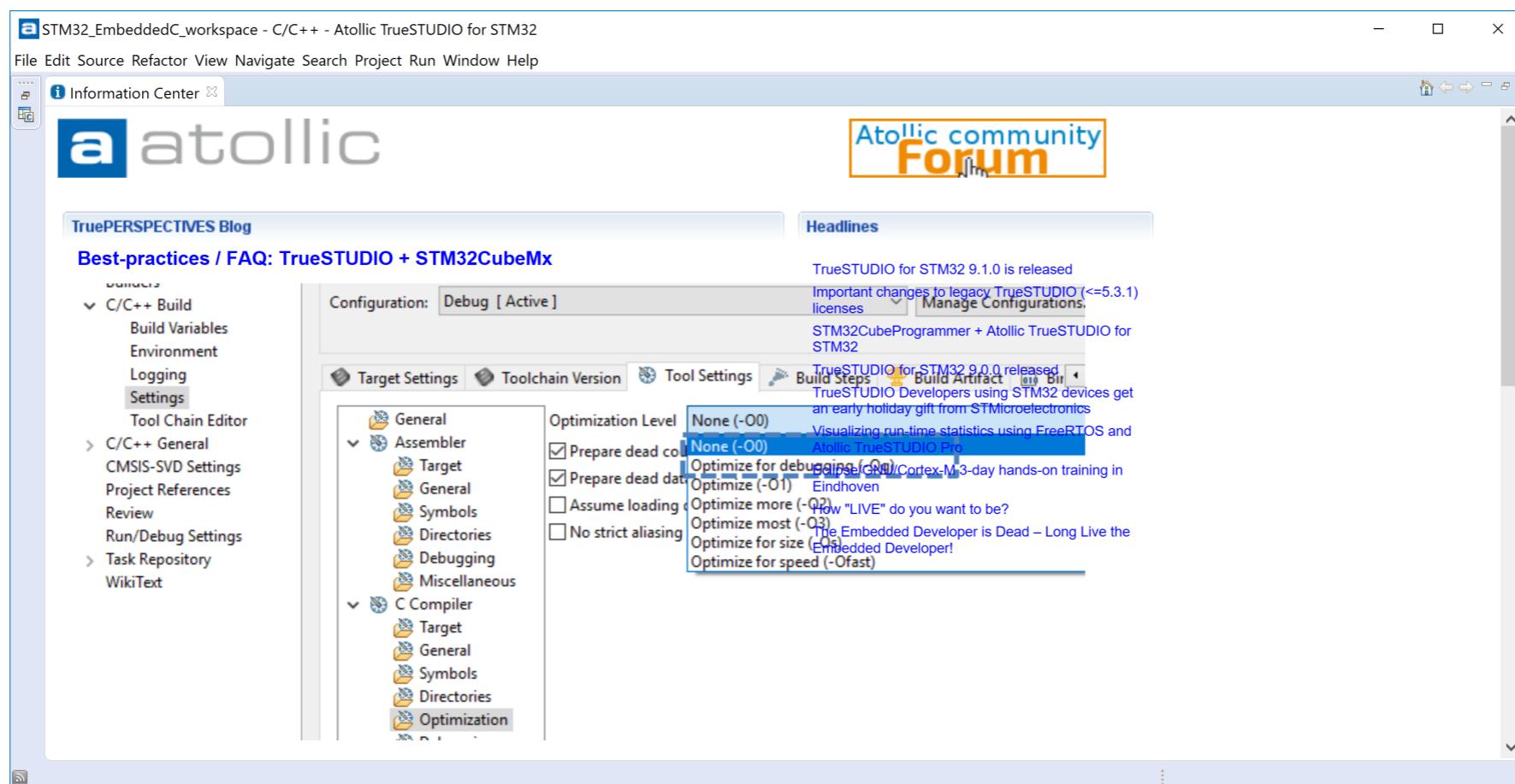


Windows installers

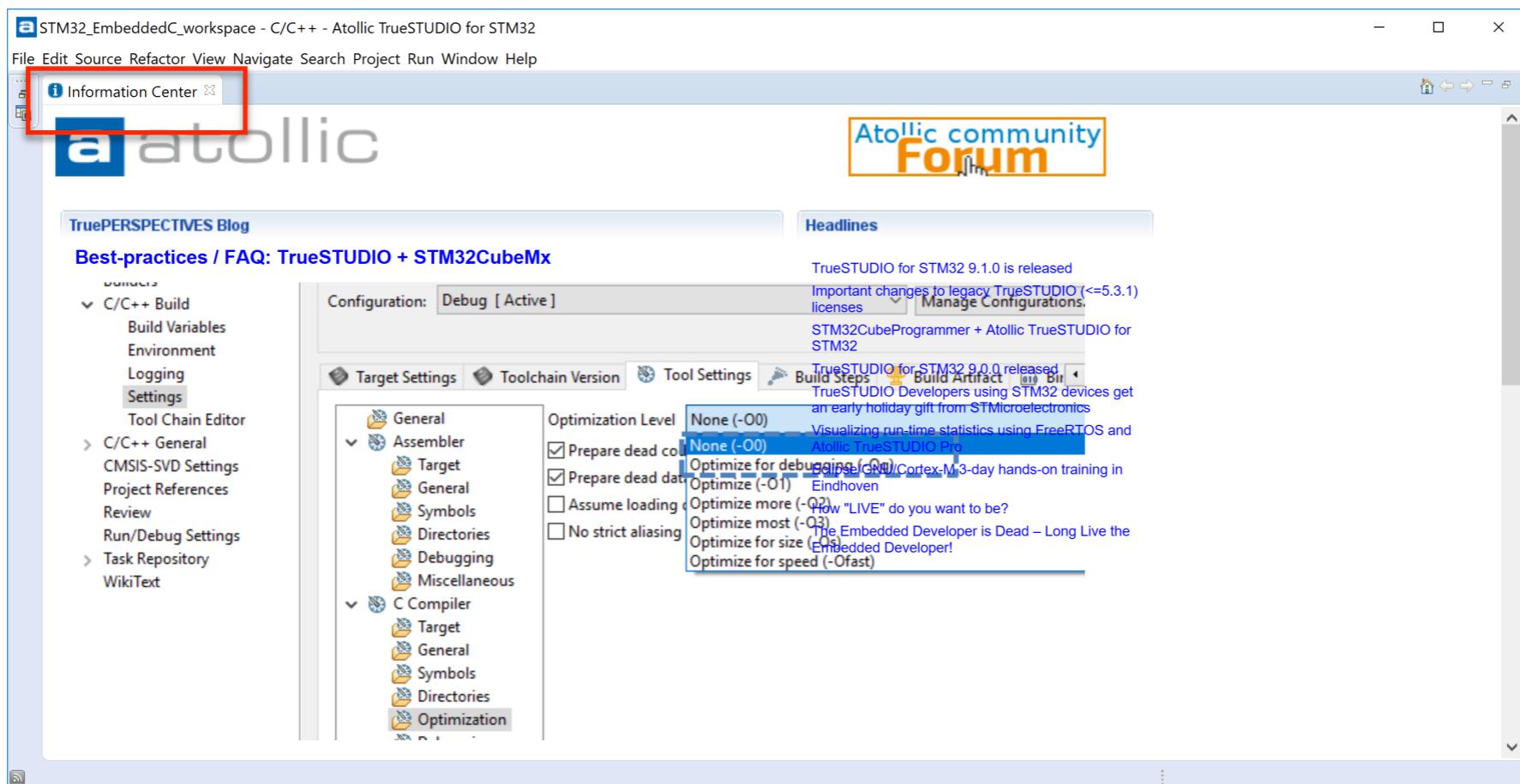


Linux installers

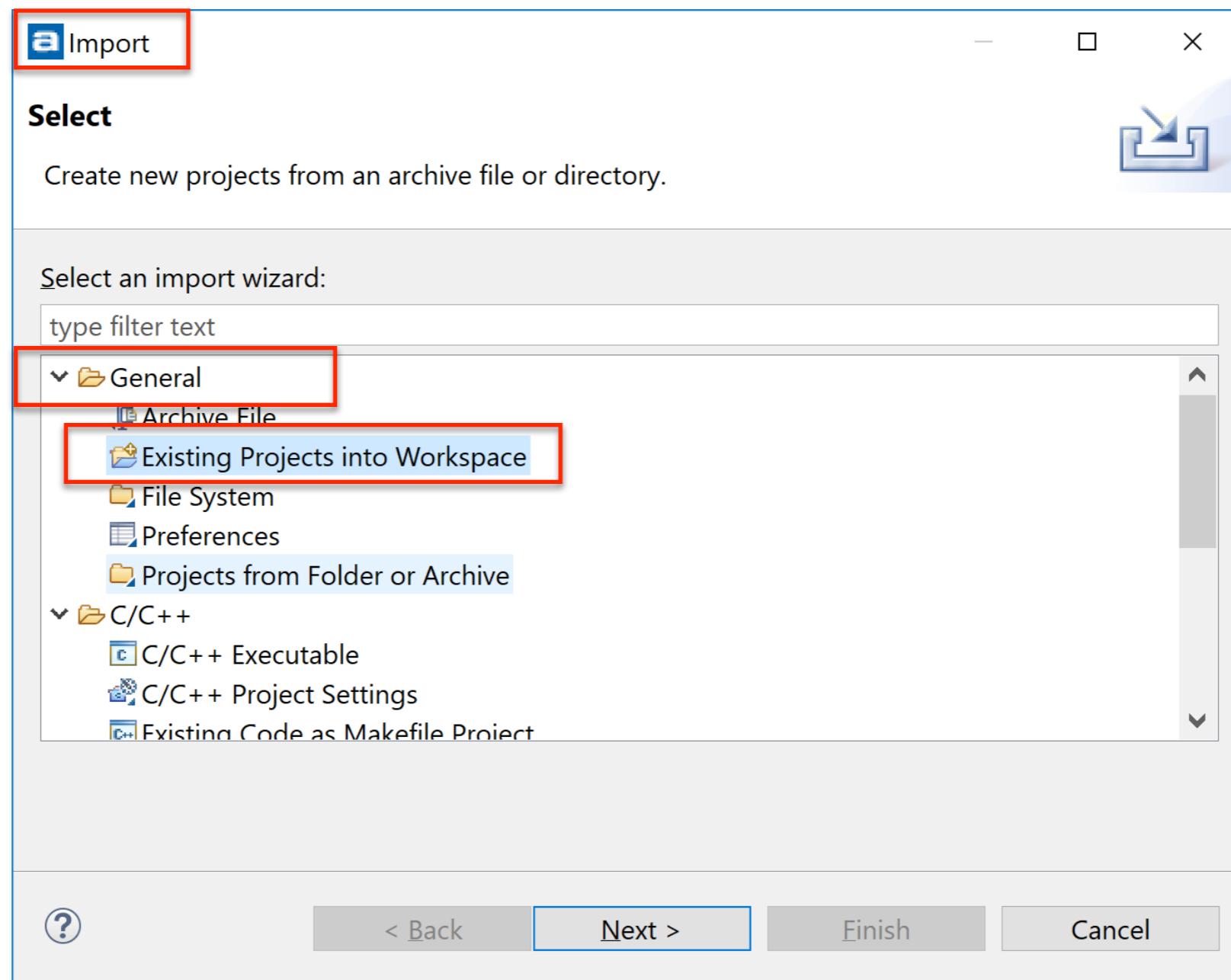
Step: Startup TrueStudio



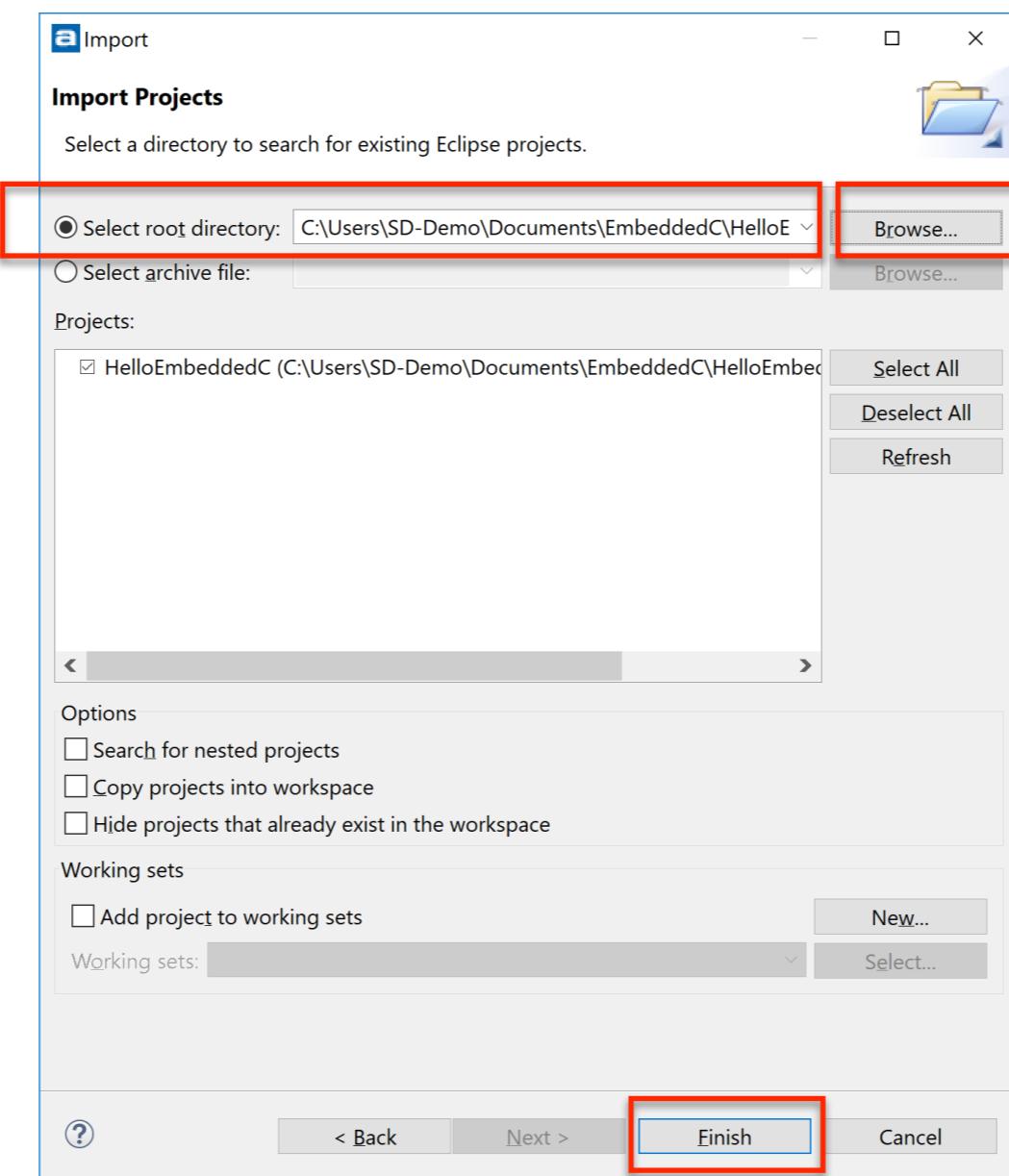
Step: Click to Close “Information Center”



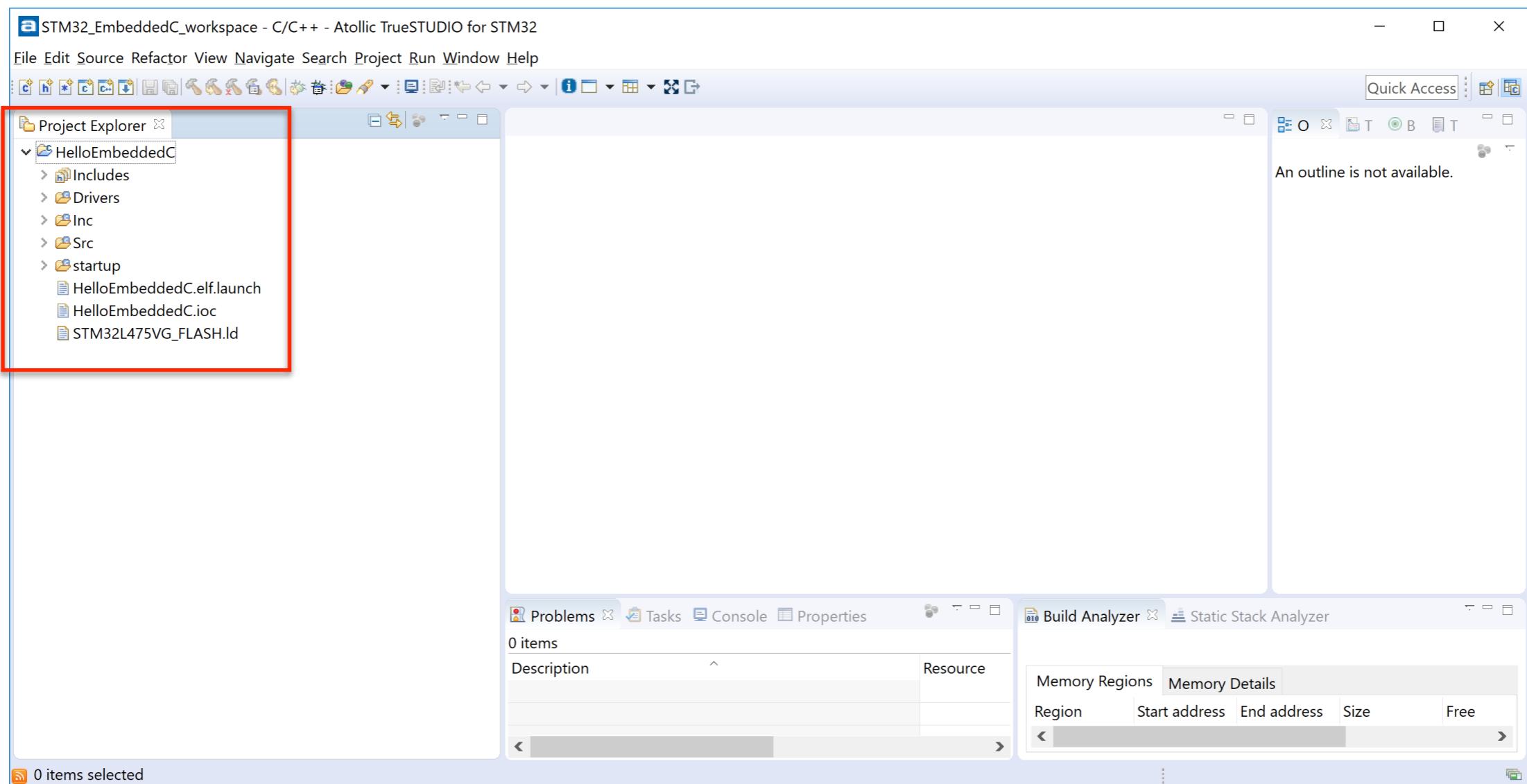
Step: File, Import, Projects, General, Existing Projects into Workspace



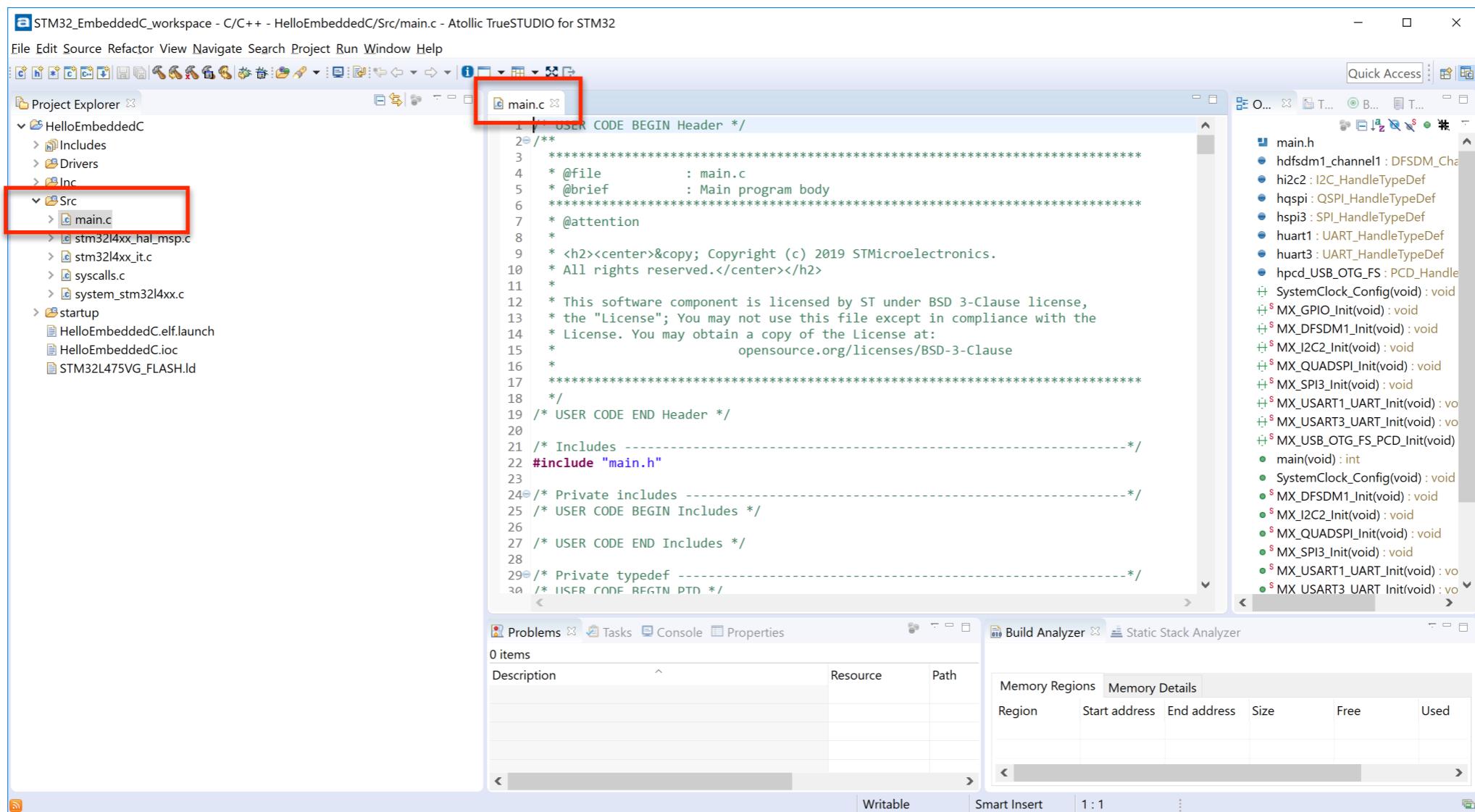
Step: Select Root Directory (Browse to directory as needed)



Step: Observe Results



Step: Open main.c

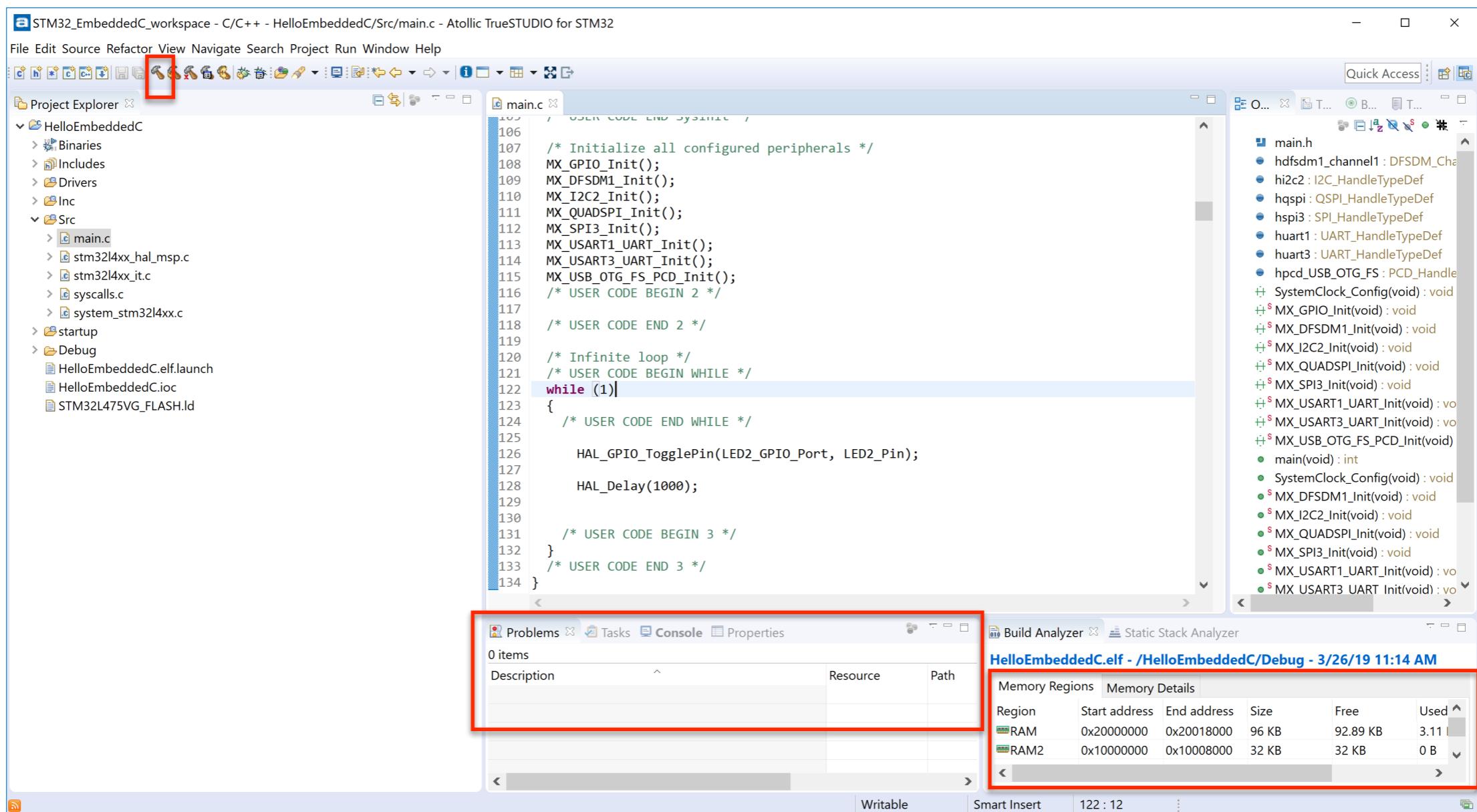


Step: Edit main.c - Add two functions: HAL_GPIO_TogglePin() and HAL_Delay()

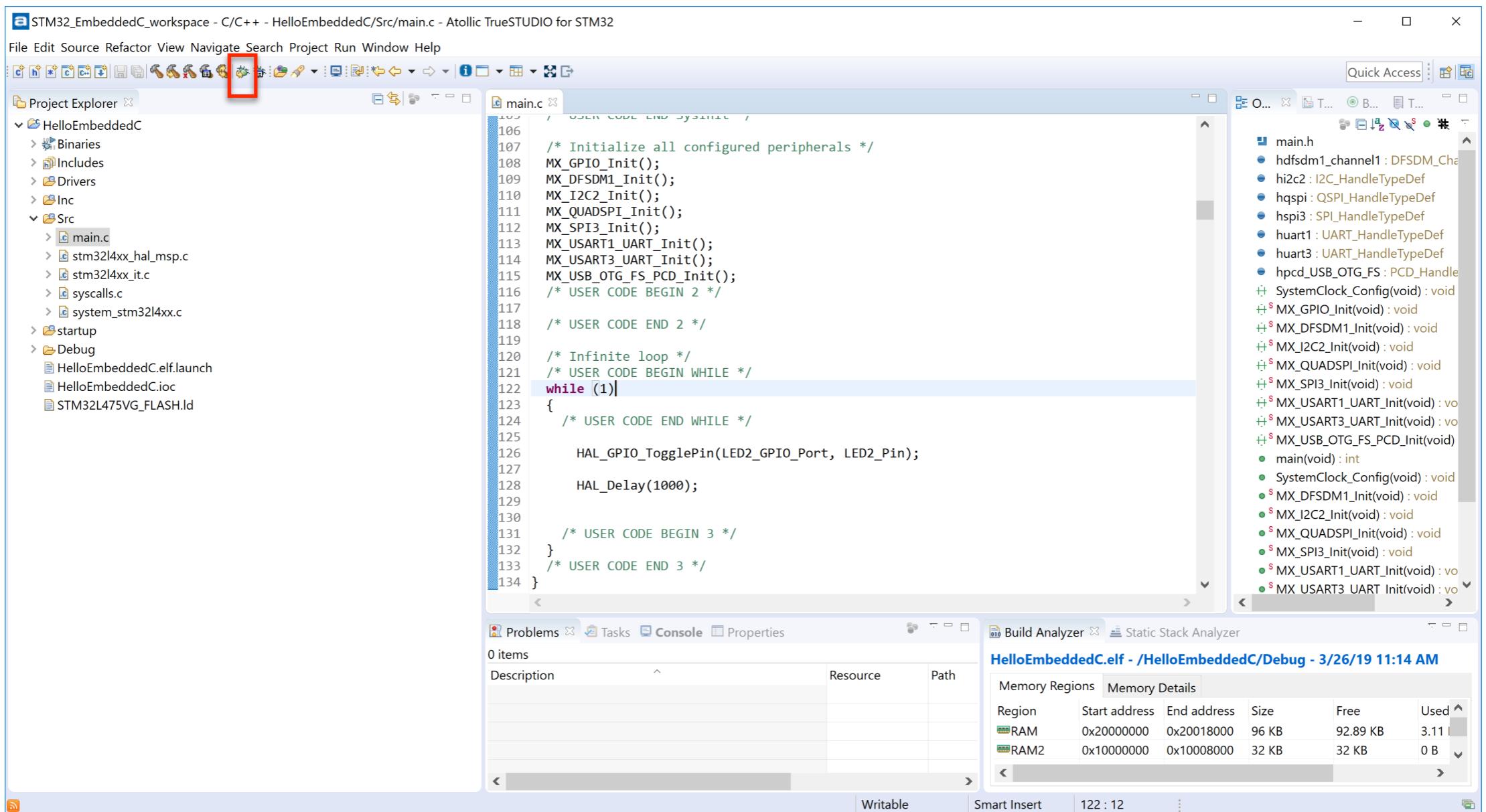
```
STM32_EMBEDDED_C_workspace - C/C++ - HelloEmbeddedC/Src/main.c - Atollic TrueSTUDIO for STM32
File Edit Source Refactor View Navigate Search Project Run Window Help
Project Explorer
HelloEmbeddedC
Includes
Drivers
Inc
Src
main.c
stm32l4xx_hal_msp.c
stm32l4xx_it.c
syscalls.c
system_stm32l4xx.c
startup
HelloEmbeddedC.elf.launch
HelloEmbeddedC.ioc
STM32L475VG_FLASH.ld
main.c
106
107     /* Initialize all configured peripherals */
108     MX_GPIO_Init();
109     MX_DFSDM1_Init();
110     MX_I2C2_Init();
111     MX_QUADSPI_Init();
112     MX_SPI3_Init();
113     MX_USART1_UART_Init();
114     MX_USART3_UART_Init();
115     MX_USB_OTG_FS_PCD_Init();
116     /* USER CODE BEGIN 2 */
117
118     /* USER CODE END 2 */
119
120     /* Infinite loop */
121     /* USER CODE BEGIN WHILE */
122     while (1)
123     {
124         /* USER CODE END WHILE */
125
126         HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
127
128         HAL_Delay(1000);
129
130         /* USER CODE BEGIN 3 */
131     }
132     /* USER CODE END 3 */
133 }
134 }
```

The screenshot shows the Atollic TrueSTUDIO IDE interface. The main window displays the 'main.c' file with the specified code block highlighted by a red rectangle. The code block consists of two function calls: 'HAL_GPIO_TogglePin' followed by 'HAL_Delay(1000)'. The rest of the code in the file is standard peripheral initialization and a main loop. The right side of the interface shows the 'Symbol View' tool window listing various symbols and their definitions.

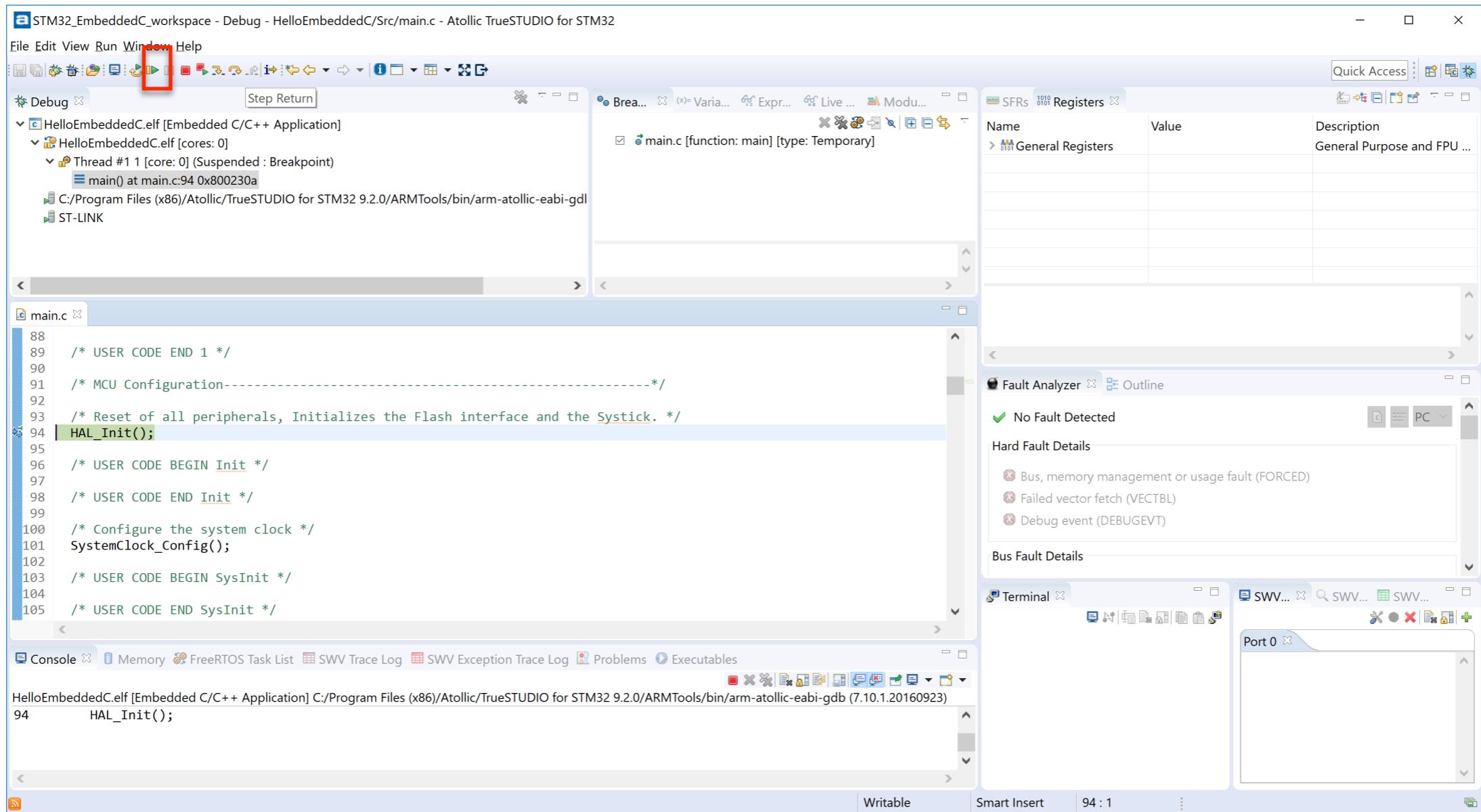
Step: Build Project



Step: Run Project in Debug Mode

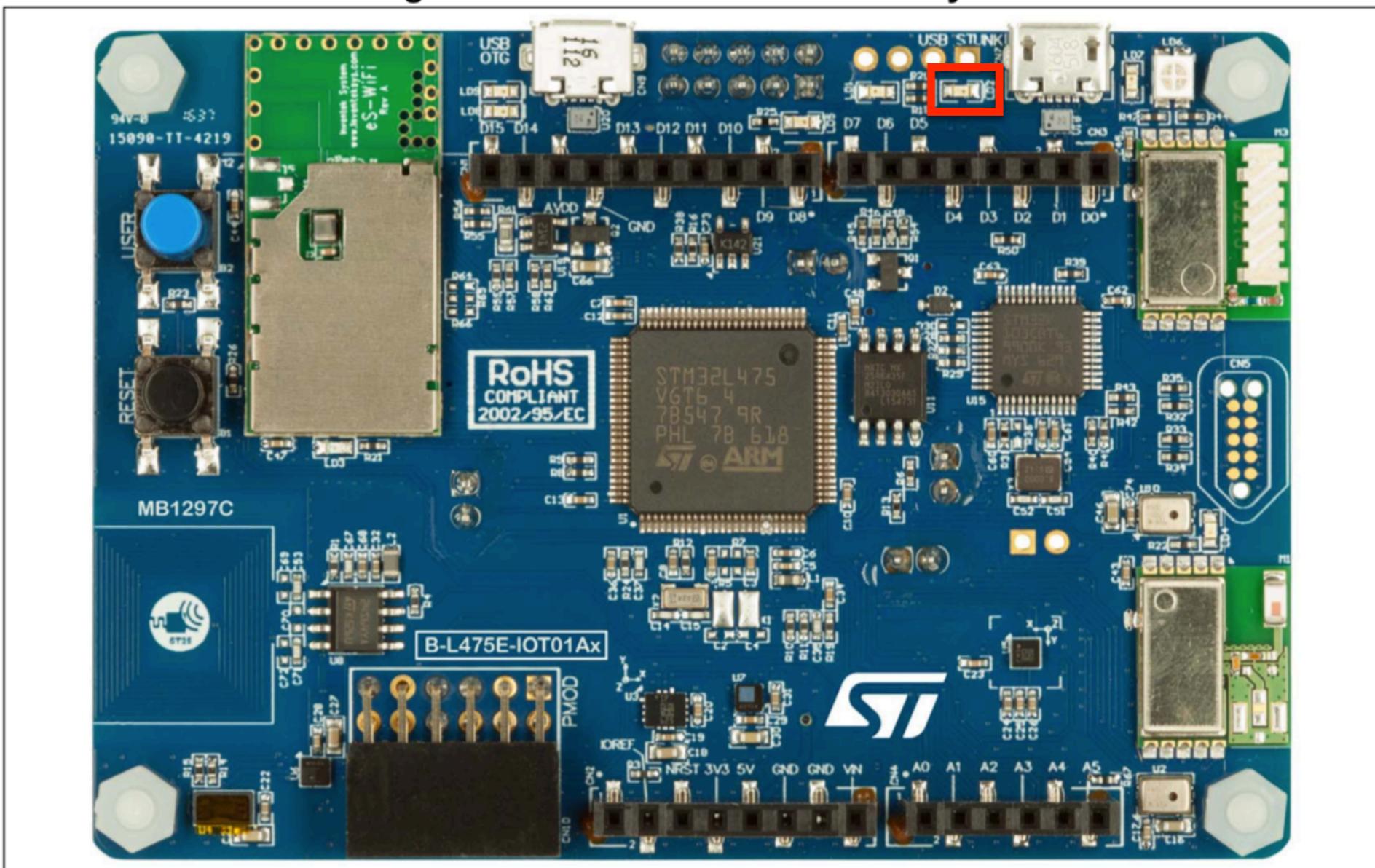


Step: Continue Running (after breakpoint hit)



Step: Look for LED blinking on STM Discovery Board at 1 second on/off rate

Figure 1. B-L475E-IOT01A Discovery kit



Summary

- Introduction to Embedded C
- STM32CubeMX
- Tour of Generated Project
- TrueStudio
- Hello Blinking LED