Lesson 3 - HAL and Embedded C

Norman McEntire
norman.mcentire@gmail.com

Contents

- Introduction to HAL and Embedded C
- STM32CubeMX and HAL Code Generation
- Tour of HAL
- TrueStudio and HAL
    - Hello Blinking LED

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

life.augmented

UM1884
User manual

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

Introduction

STMCube™ is STMicroelectronics's original initiative to ease developers' life by reducing development efforts, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows generating C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeL4 for STM32L4 series and STM32L4+ series)
  - The STM32Cube Hardware Abstraction Layer (HAL), an STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio. The HAL is available for all peripherals.
  - The low-layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. The LL APIs are available only for a set of peripherals.

Key Sections From Manual

Introduction to HAL and Embedded C

Introduction to HAL - Part 1

- HAL = Hardware Abstraction Layer
    - Maximize portability across STM32 portfolio
    - Available for all STM32 peripherals
- HAL provides multi-instance set of APIs to interact with upper layers
- HAL drivers are split into two categories
    - Generic APIs - common to complete STM32 portfolio
    - Extension APIs - specific and customized features for given part number
- HAL API is very complete and fully portable between STM32 portfolio
    - Device Initialization
    - Device Communication
        - Polled
        - Interrupt Driven
        - DMA Driven

Introduction to HAL - Part 2

- HAL Drivers are feature oriented instead of IP oriented
    - Example: Timer APIs are split into the following IP functions
        - Basic Timer
        - Capture Timer
        - Pulse Width Modulation
- HAL Drivers include full run-time failure detection
- Strict ANSI C
- Checked with CodeSonar Static Analysis tool
- Fully Documented and MIRSA-C 2004 Compliant

HAL Main Features - Part 1

- Cross-Family Portability across STM32 portfolio
- Three Programming Modes
    - Polling
    - Interrupt
    - DMA Driven
- APIs are RTOS Compliant
    - Fully reentrant
    - Systematic usage of timeouts in polling mode
- Peripheral Multi-Instance
    - USART1, USART2, etc. can be called in parallel

HAL Main Features - Part 2

- All HAL APIs implement user-callback functions
- Object locking mechanism to ensure safe hardware access
- Timeout used for all blocking processes
  - Can be simple timer
  - Can be timebase

HAL Driver Files stm32l4xx_hal.h/.c

- stm32l4xx_hal.h - Header File
- stm32l4xx_hal.c - HAL Initialization including:
  - DBGMCU
  - Remap
  - Time Delay based on SysTick APIs

HAL Driver Files stm32l4xx_hal_def.h

- stm32l4xx_hal_def.h - Header File
  - Common HAL resources such as
  - Common #define statements
  - Enumerations
  - Structures
  - Macros

HAL Driver Files stm32l4xx_hal_ppp.h/.c

- stm32l4xx_hal_ppp.h - Header File
    - Example: stm32l4xx_adc.h
- stm32l4xx_hal_ppp.c - Main peripheral/module driver file
    - Example: stm32l4xx_adc.c

HAL Driver Files
stm32l4xx_hal_ppp_**ex**.h/.c

- stm32l4xx_hal_ppp_**ex**.h - Header File
  - Example: stm32l4xx_adc_**ex**.h
- stm32l4xx_hal_ppp_**ex**.c - Extensions that are specific to part number or override default generic APIs
  - Example: stm32l4xx_adc_**ex**.c

HAL Data Structures

2.2 HAL Data Structures

- Each HAL Driver contains the following data structures
  - Peripheral Handle Structures
  - Initialization and Configuration Structures
  - Specific Process Structures

2.2.1 Peripheral Handle Structures

- PPP_HandleTypeDef *handle
  - Main structure implemented in HAL drivers
  - Handles configuration and registers access
  - Embeds all structures and variables needed to follow the peripheral device flow
  - See example next slide

USART_**Handle**TypeDef

- typedef struct {
  USART_TypeDef *Instance; //Base Address
  USART_InitTypeDef Init; //Com params
  uint8_t *pTxBuffPtr;
  uint16_t TxXferSize;
  __IO uint16_t TxXferCount;
  uint8_t *pRxBuffPtr;
  uint16_t RxXferSize;
  __IO uin16_t RxXferCount;
  DMA_HandleTypeDef *hdmatx;
  DMA_HandleTypeDef *hdmarx;
  HAL_LockTypeDef Lock;
  __IO HAL_USART_StateTypeDef State;
  __IO HAL_USART_ErrorTypeDef ErrorCode;
} **USART_HandleTypeDef**

2.2.2 Initialization and Configuration Structure

- Define values required to configure and initialize a given device

- If config changes from part number to part number, then defined in _ex.h version of file

- See example next slide

UART_**Init**TypeDef

- typedef struct {
    uint32_t BaudRate;
    uint32_t WordLength;
    uint32_t StopBits;
    uint32_t Parity;
    uint32_t Mode;
    uint32_t HwFlowCtl;
    uint32_t OverSampling;
} **UART_InitTypeDef**

2.3 API Classification

- Three Categories
  - Generic APIs - Apply to all STM32 devices
  - Extension APIs - Family Specific
  - Extension APIs - Device Specific

Generic API Example HAL_ADC_…

- HAL_ADC_Init(…)
- HAL_ADC_DeInit(…)
- HAL_ADC_Start(…)
- HAL_ADC_Stop(…)
- HAL_ADC_Start_IT(…)
- HAL_ADC_Stop_IT(…)

Extension API - Family Specific HAL_ADCEx_…

- HAL_ADCEx_Calibration_Start(…)
- HAL_ADCEx_Calibration_GetValue(…)

Extension API - Device Part Number Specific HAL_ADCEx_…

- These will always have #if defined(…)
- #if defined (STM32L475xx) || defined (…)
  HAL_PWREx_EnableVddUSB(void);
  HAL_PWREx_DisableVddUSB(void);
  #endif

2.5.1 HAL Driver Naming Rules - Part 1

- Filenames
  - stm32l4xx_hal_ppp.h/.c - Generic
  - stm32l4xx_hal_ppp_ex.h/.c - Family Specific or Device Specific
- Handle Names
  - PPP_HandleTypeDef
- Int Structure Names
  - PPP_InitTypeDef
- Enum Name
  - HAL_PPP_StructnameTypeDef

2.5.1 HAL Driver Naming Rules - Part 2

- Registers are considered as constants
- In most cases upper-case
- Register names are same as acronyms used in technical documentation

2.5.2 HAL General Naming

- For shared and system peripherals, no handle or instance object used
  - GPIO
    - Example
      - HAL_StatusTypeDef HAL_GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef *Init)
  - SYSTICK
  - NVIC
  - RCC
  - FLASH

Note: GPIOx is GPIOA, GPIOB, etc.

2.5.3 HAL Interrupt Handler and Callback Functions

- HAL peripheral drivers include the following
  - HAL_PPP_IRQHandler()
    - Peripheral interrupt handler that should be called from stm32l4xx_it.c
- User Callback Functions
  - Empty functions with “weak” attribute

Three Types of User Callback Functions

- #1. Peripheral system level init/de-init callbacks
    - These are called by HAL_PPP_Init() function to perform peripheral system-level integration
        - HAL_PPP_MspInit()
        - HAL_PPP_MspDeInit()

MSP = MCU Specific Package

- #2. Process complete callbacks
    - Called by peripheral or DMA interrupt handler when the process completes
        - HAL_PPP_ProcessCpltCallback()
- #3. Error Callback
    - Called by peripheral or DMA interrupt handler when an error occurs
        - HAL_PPP_ErrorCallback()

2.6 HAL Generic APIs

- HAL Generic APIs apply to all STM32 devices
- Four API Groups
    - Init/DeInit
        - HAL_PPP_Init(), HAL_PPP_DeInit()
    - I/O Operations
        - HAL_PPP_Read(), HAL_PPP_Write(), HAL_PPP_Transmit(), HAL_PPP_Receive()
    - Control Functions
        - HAL_PPP_Set(), HAL_PPP_Get()
    - State and Error Functions
        - HAL_PPP_GetState(), HAL_PPP_GetError()

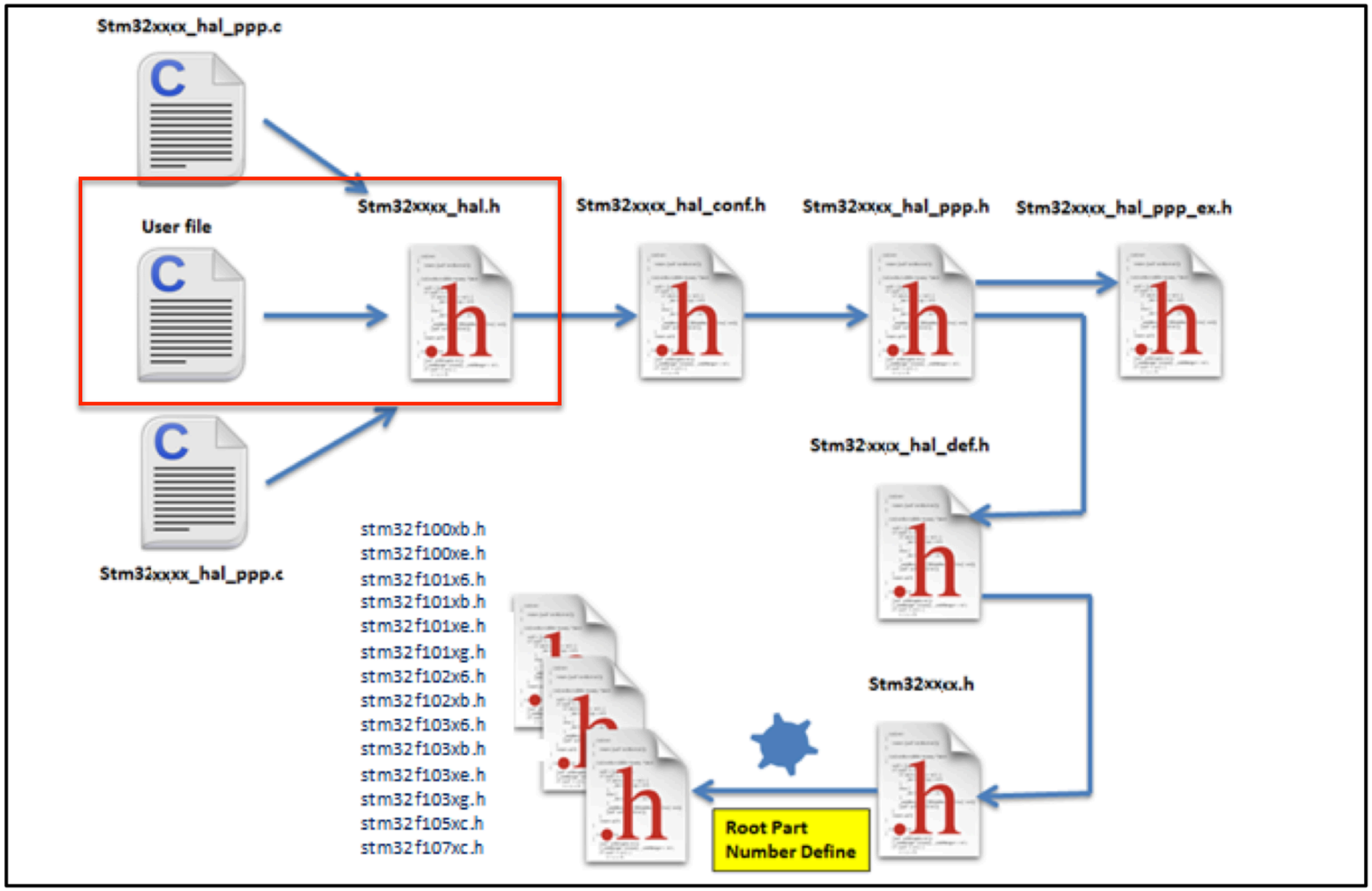
2.7 HAL Extension APIs

- Extension APIs
  - Provide specific functions
  - Override modified APIs for a specific family
- Additional files
  - stm32l4xx_ppp_ex.h/.c
    - Provide specific functions for a given part number

2.8 File Inclusion Model

- Single include file for use in user-mode code
  - stm32l4xx_hal.h
    - Includes common configs for the whole HAL library

File Inclusion Model



To Enable Specific PPP

```
/**************************************************
* @file stm32l4xx_hal_conf.h
* @author MCD Application Team
* @version VX.Y.Z * @date dd-mm-yyyy
* @brief This file contains the modules to be used
**************************************************
(…)
#define HAL_USART_MODULE_ENABLED
#define HAL_IRDA_MODULE_ENABLED
#define HAL_DMA_MODULE_ENABLED
#define HAL_RCC_MODULE_ENABLED
(…)
```

2.9 HAL Common Resources

- HAL_StatusTypeDef

```
Typedef enum
{ HAL_OK = 0x00, HAL_ERROR = 0x01, HAL_BUSY = 0x02, HAL_TIMEOUT = 0x03
} HAL StatusTypeDef;
```

- HAL_LockTypeDef

```
typedef enum
{ HAL_UNLOCKED = 0x00, /*!<Resources unlocked */
HAL_LOCKED = 0x01 /*!< Resources locked */
} HAL_LockTypeDef;
```

- Common Macros

```
#define HAL_MAX_DELAY 0xFFFFFFFF
```

2.10 HAL Configuration

- stm32l4xx_hal_config.h
- Allows customizing the drivers used for given application
  - Uncomment/Comment/Change Values

Selected HAL Config Items Part 1

- HSE_VALUE - Default is 8_000_000 Hz
  - Value of high-speed external oscillator in HZ
- HSE_STARTUP_TIMEOUT - Default is 100ms
  - Timeout for HSE startup in ms

Selected HAL Config Items Part 2

- HSI_VALUE - Default is 16_000_000
    - Value of high-speed internal oscillator in HZ
- MSI_VALUE - Default is 4_000_000
    - Value of multi-speed internal oscillator in HZ
- LSI_VALUE - Default is 32_000 Hz
    - Value of low speed internal oscillator in HZ
- LSE_VALUE - Default is 32768 Hz
    - Value of low speed external oscillator in HZ
- LSE_STARTUP_TIMEOUT - Default is 5000
    - Timeout for LSE start-up, expressed in ms

Selected HAL Config Items Part 3

- VDD_Value - 3300mv
- PREFETCH_ENABLE - Default is FALSE
- INSTRUCTION_CACHE_ENABLE - Default is TRUE
- DATA_CACHE_ENABLE - Default is TRUE

stm32l4xx_hal_conf_template.h

- This is the “template” file
- By default, it has features enabled as needed for STM32 examples and demonstrations
- Copy it to your project and then edit as needed to enable/disable HAL features

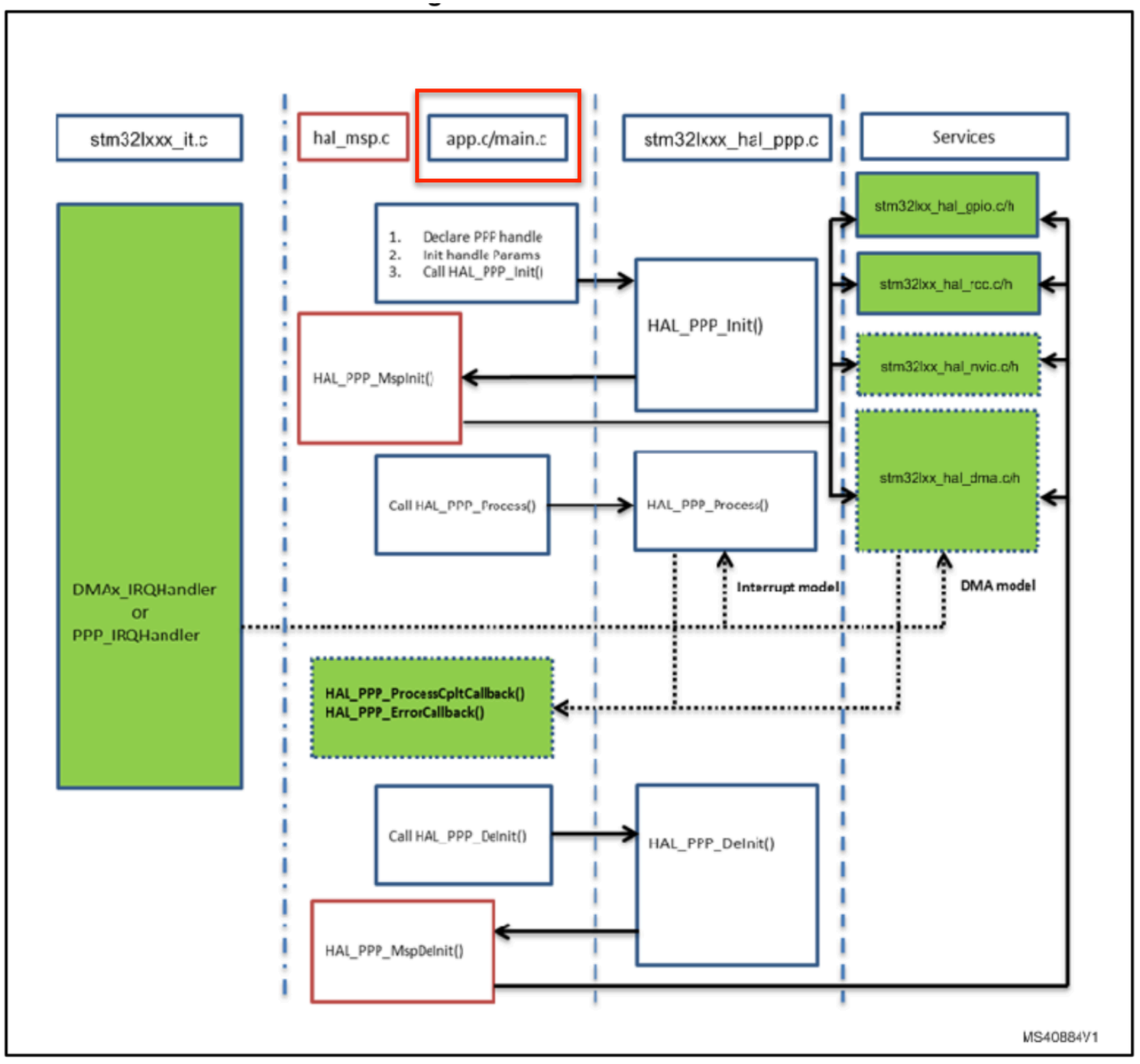
2.11 HAL System Peripheral Handling

- System Peripherals Include the following
  - Clocks
  - GPIOs
  - Cortex NVIC and SysTick Timer
  - PWR - Power
  - EXTI - External Interrupt
  - DMA - Direct Memory Access

2.12 How to use HAL Drivers

- HAL Global Initialization
    - HAL_Init() - Must be called at application startup
        - Calls HAL_MspInit()
            - MSP = MCU Specific Package
- System Clock Initialization

HAL Driver Model



2.12.3 HAL I/O Operations

- Three Data Processing Models
  - Polling Mode
  - Interrupt Mode
  - DMA Mode

Polling Mode

- HAL returns process status when blocking mode complete
- HAL_OK or Error Code
    - If Error Code, call HAL_PPP_GetStatus() for more info

Polling Example

- HAL_StatusTypeDef
  HAL_PPP_Transmit(
    PPP_HandleTypeDef *phandle,
    uint8_t *pData,
    int16_t Size,
    uint32_t Timeout {
    if ((pData == NULL) || (Size == 0)) return
HAL_ERROR;
    …
    if (timeout reached) return HAL_TIMEOUT;
    return HAL_OK;
}

Interrupt Mode

- HAL returns process status after starting data processing and enabling the appropriate interrupt
- End of interrupt indicated by callback
- Four Functions declared in driver
  - HAL_PPP_Process_IT() - launch the process
  - HAL_PPP_IRQHandler() - global PPP interrupt
  - __weak HAL_PPP_ProcessCpltCallback() - callback relative to process completion
  - __weak HAL_PPP_ProcessErrorCallback() - callback relative to process error

Interrupt Mode Example

- main.c
  - int main(void) {
    …
    HAL_PPP_Init(…);
    HAL_PPP_TxIT(…);
    }
    void HAL_PPP_TxCpltCallback(…) { … }
    void HAL_PPP_ErrorCallback(…) { … }

- stm32l4xx_it.c
  - void PPP_IRQHandler(void) {
    HAL_PPP_IrqHandler(…);
    }

DMA Mode

- Returns process status after starting the DMA processing and enabling DMA interrupt
- End of operation indicated by callback
- Four functions declared in driver
  - HAL_PPP_Process_DMA() - launch the process
  - HAL_PPP_IRQHandler() - global PPP interrupt
  - __weak HAL_PPP_ProcessCpltCallback() - callback relative to process completion
  - __weak HAL_PPP_ProcessErrorCallback() - callback relative to process error

DMA Mode Example

- main.c
  - int main(void) {
    …
    HAL_PPP_Init(…);
    HAL_PPP_Tx_DMA(…);
    }
    void HAL_PPP_TxCpltCallback(…);
    void HAL_PPP_TxErrorCallback(…);

- stm32l4xx_it.c
  - void PPPx_IRQHandler(void) {
    HAL_DMA_IRQHandler(…);
    }

2.12.4 Timeout and Error Management

- Polling mode HAL APIs use timeouts
  - Example
    - HAL_StatusTypeDef HAL_DMA_PollForTransfer(DMA_HandleTypeDef *hdma, uint32_t CompleteLevel, uint32_t Timeout)
- Timeout Values
  - 0 = No poll. Immediate process check and exit
  - 1…MAX_HAL_DELAY-1 - Timeout in ms
  - MAX_HAL_DELAY - Infinite poll

2.12.4.2 Error Management

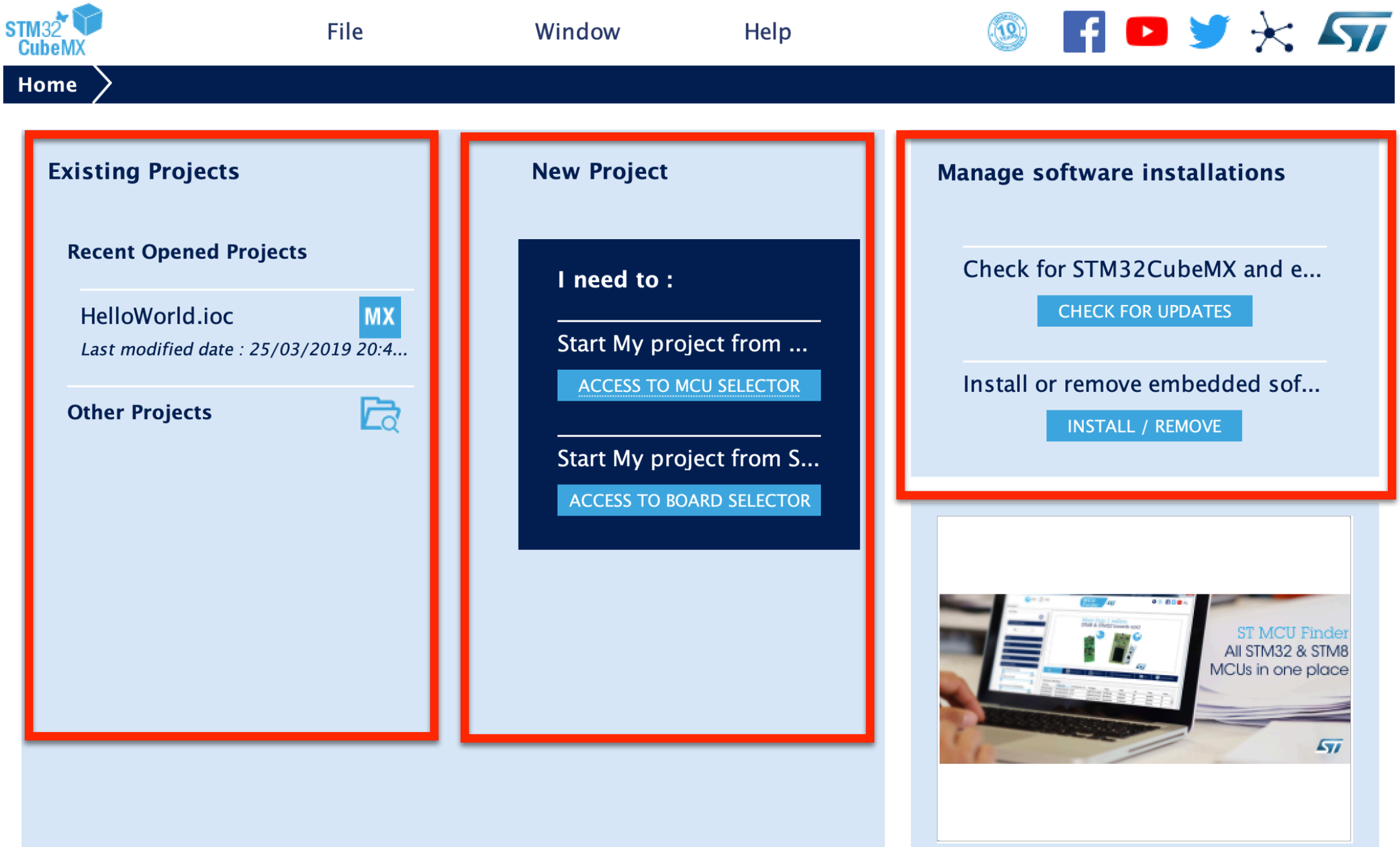
- HAL Drivers check the following
  - Valid Parameters
  - Valid Handle
  - Timeouts

2.12.4.3 Run-time Checking

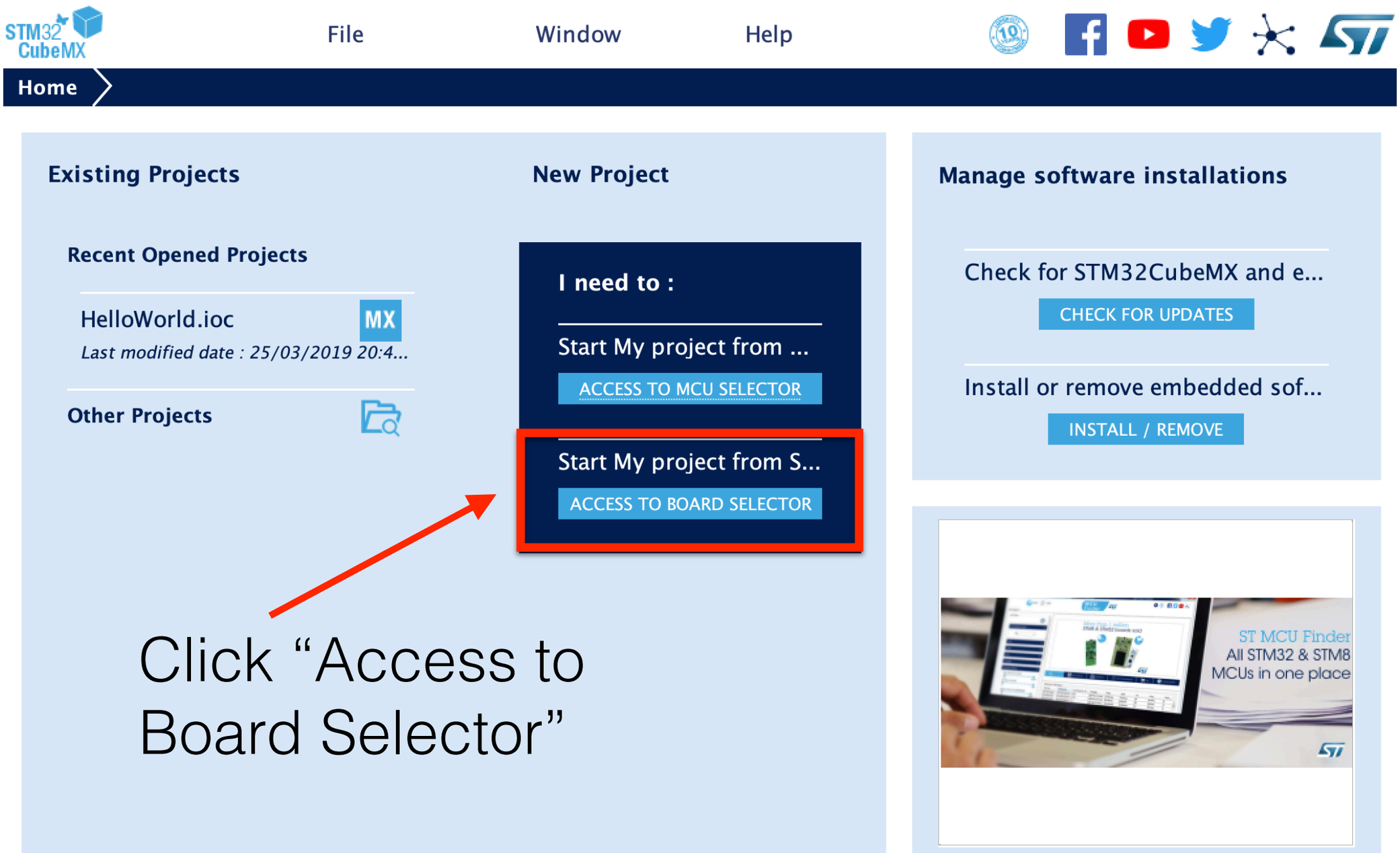
- HAL Drivers implement run-time checking
- Use of assert_param macro
- To use full runtime checking uncomment this define
  - USE_FULL_ASSERT
- NOTE: Due to overhead of run-time checking, use this during development but NOT for production

STM32CubeMX and Generation of HAL Code

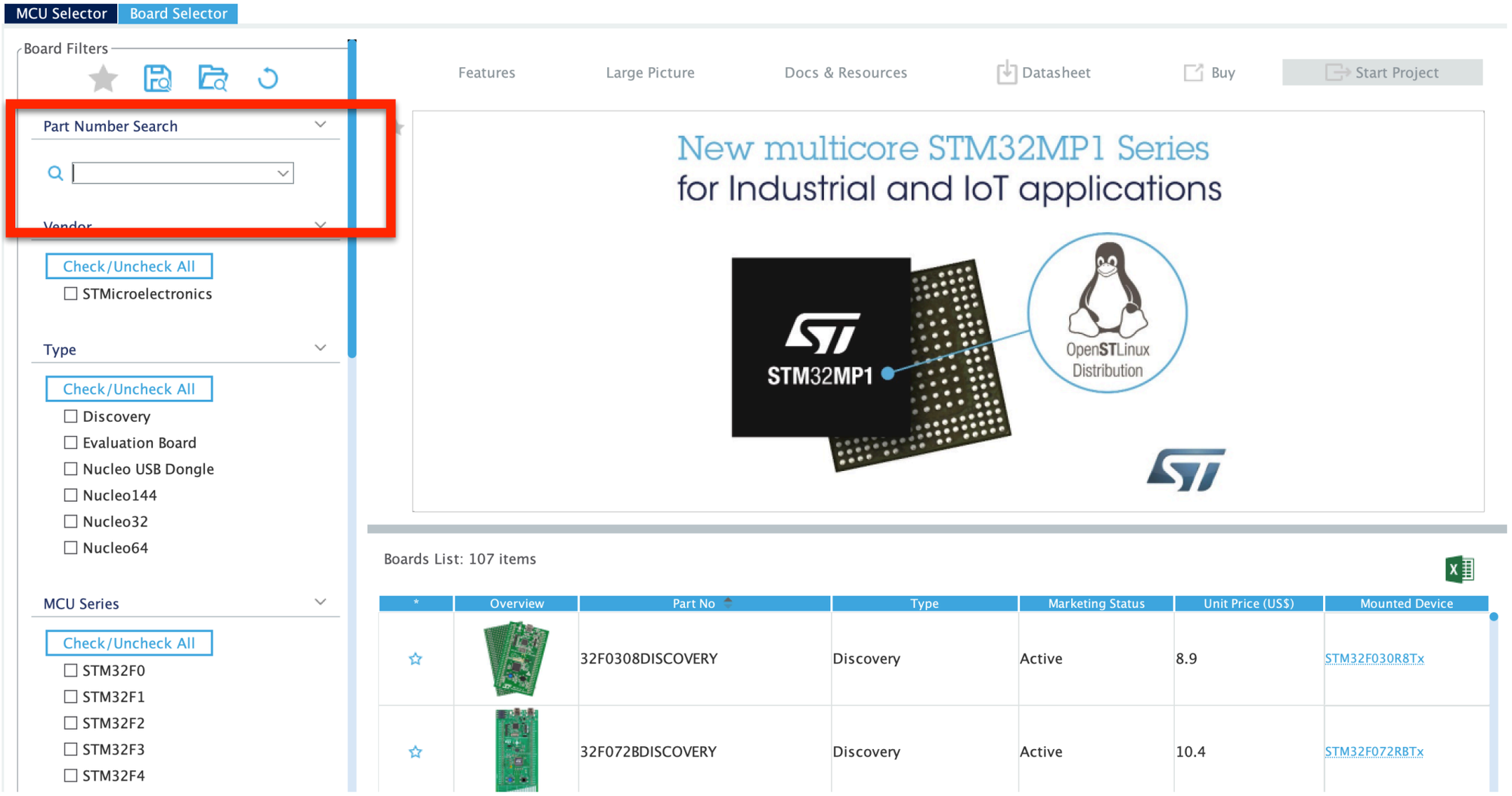
Step: Startup STM32CubeMX



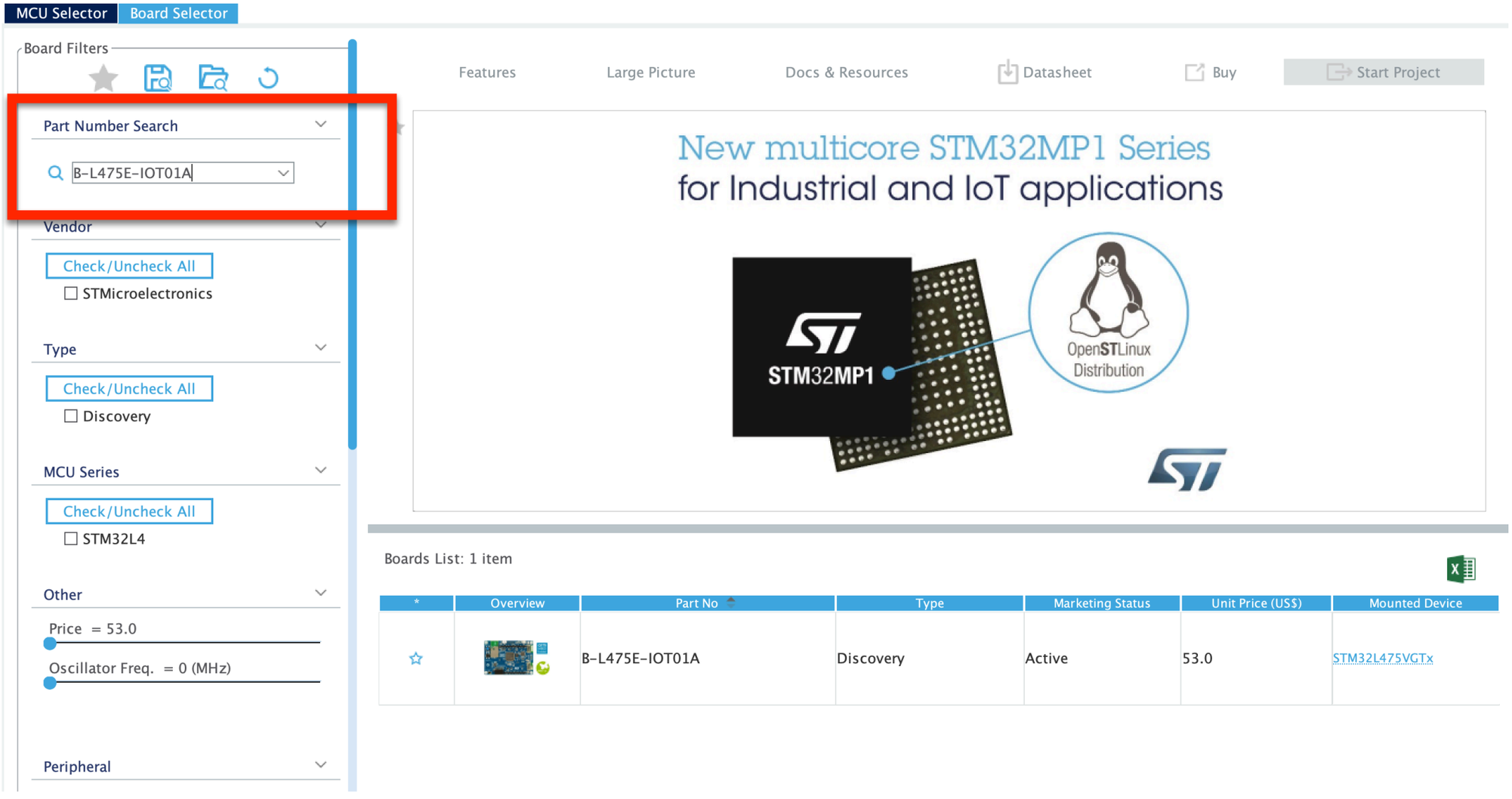
Step: Click on “Access to Board Selector”



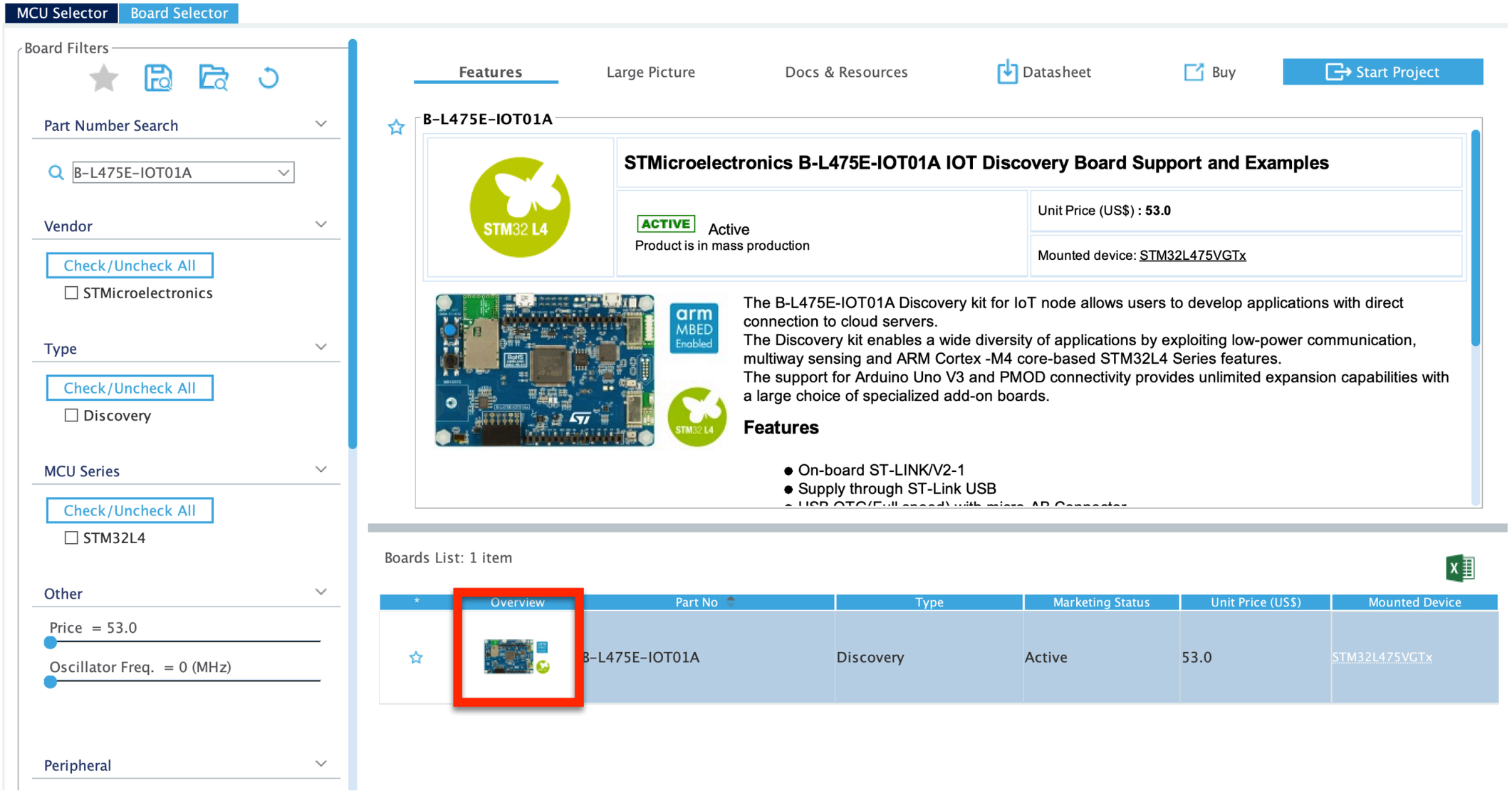
Step: Observe “Part Number Search”



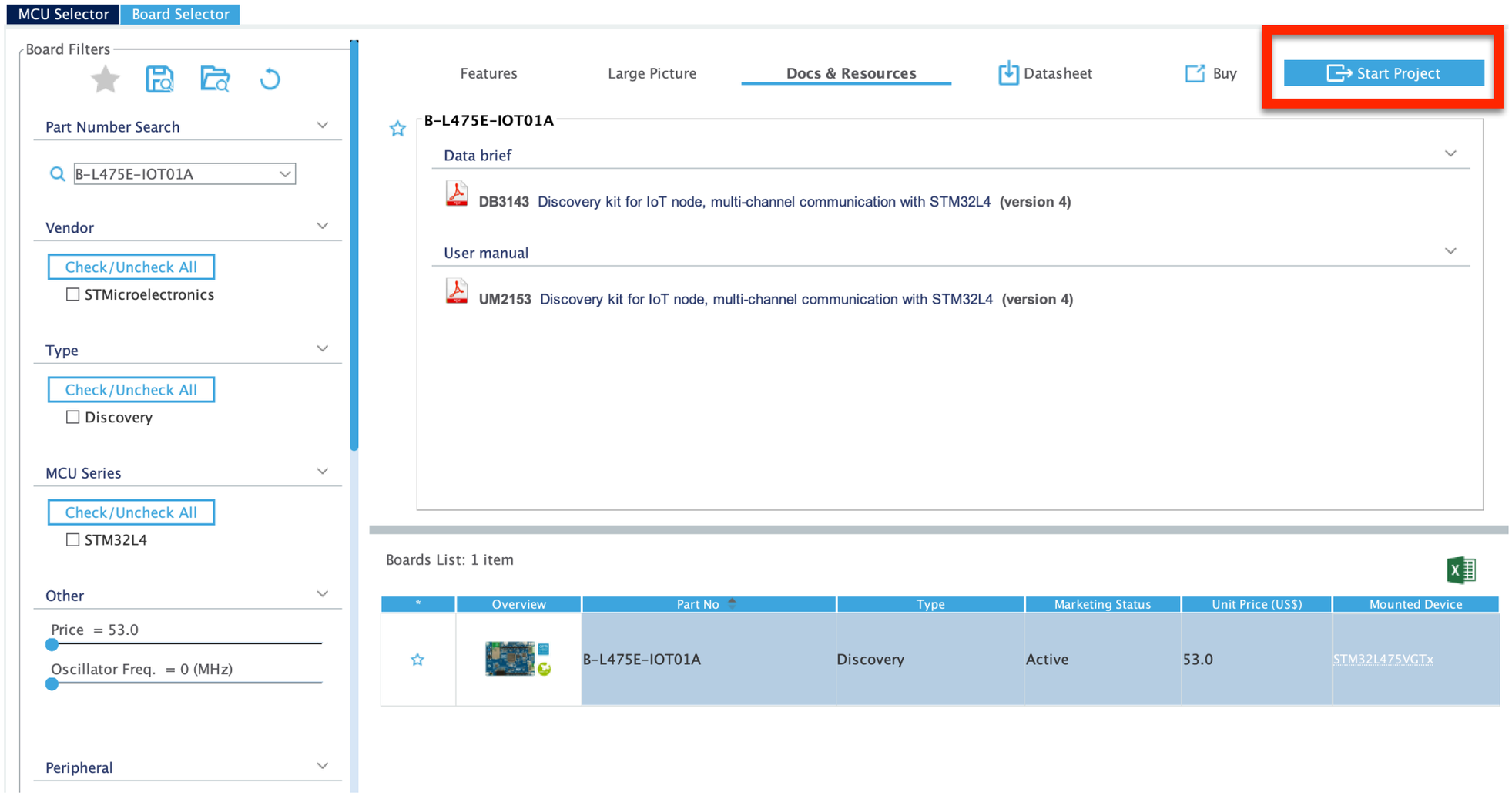
Step: Enter ”B-L475E-IOT01A”



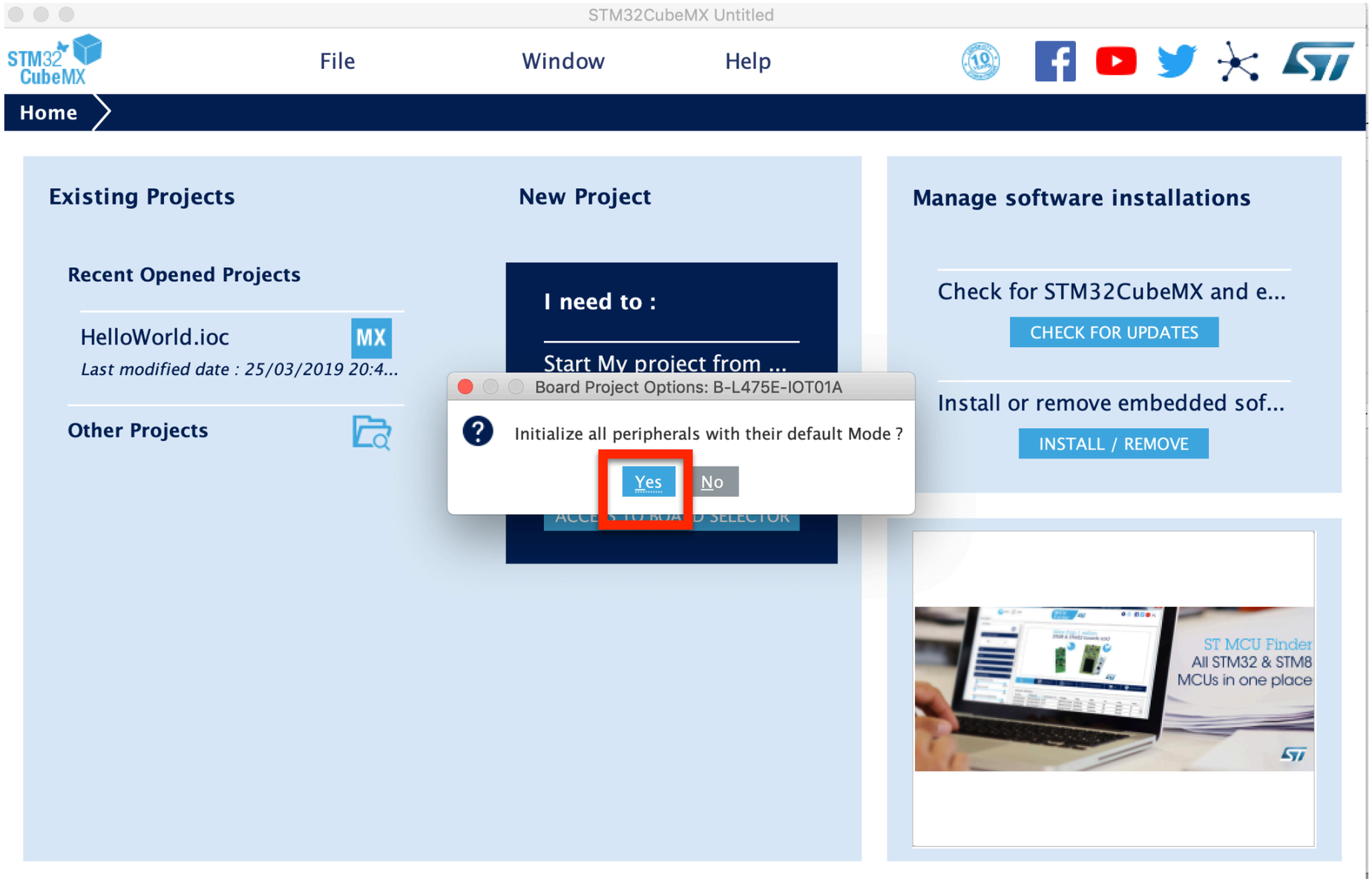
Step: Click on Image, and observe “Features”



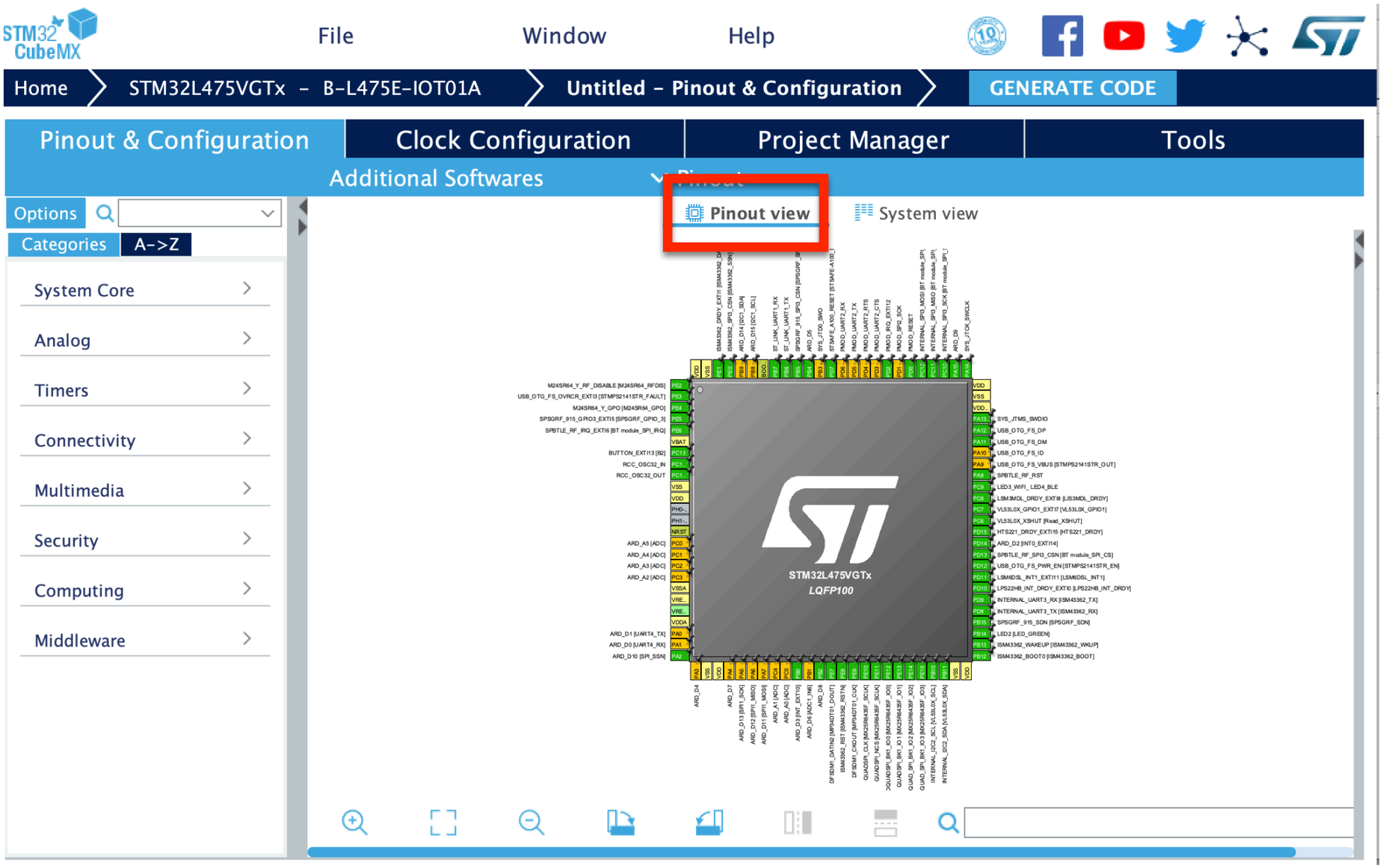
Step: Click on “Start Project”



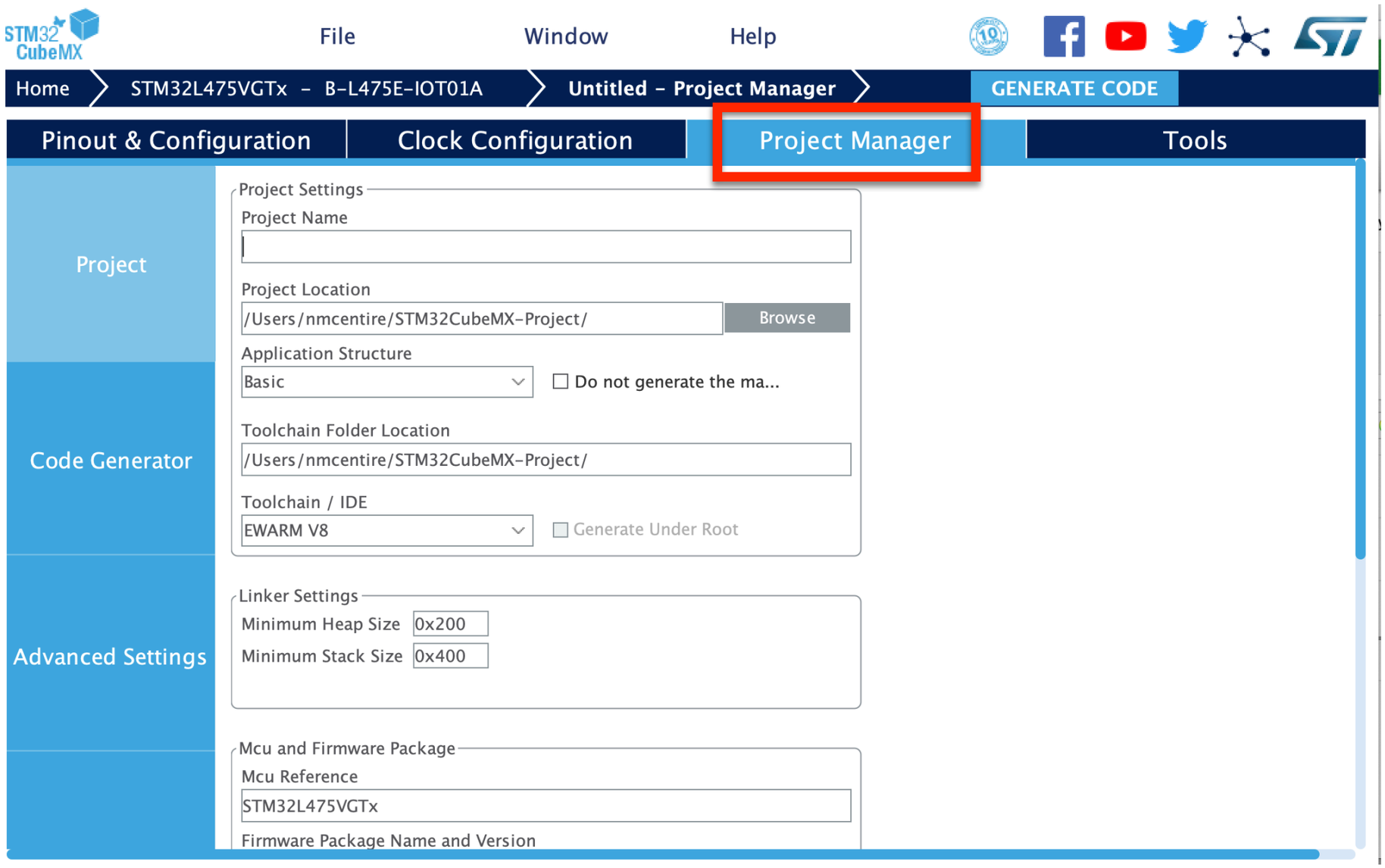
Step: Click on “Yes” (Initialize all …with default mode)



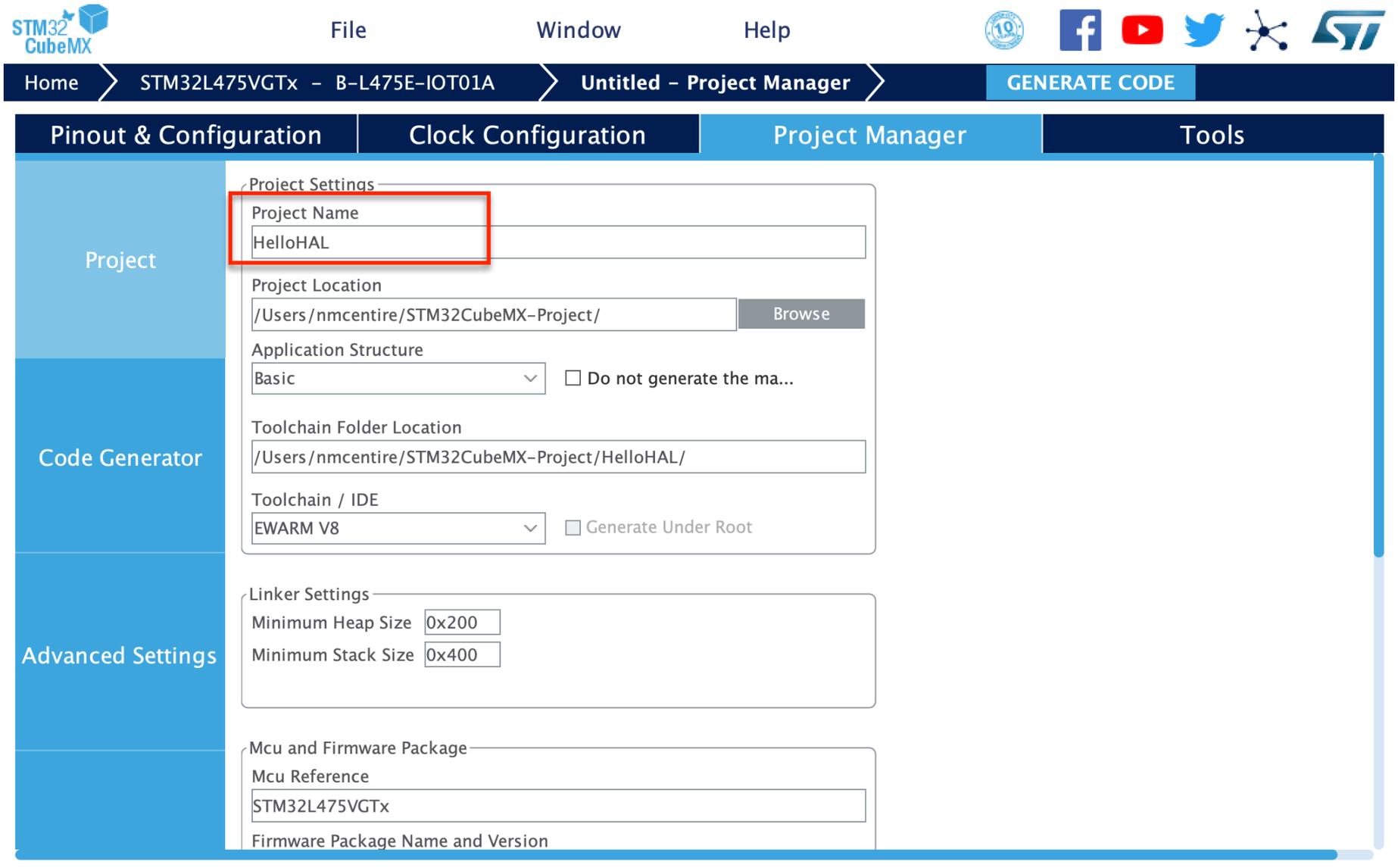
Step: Observe “Pinout View”



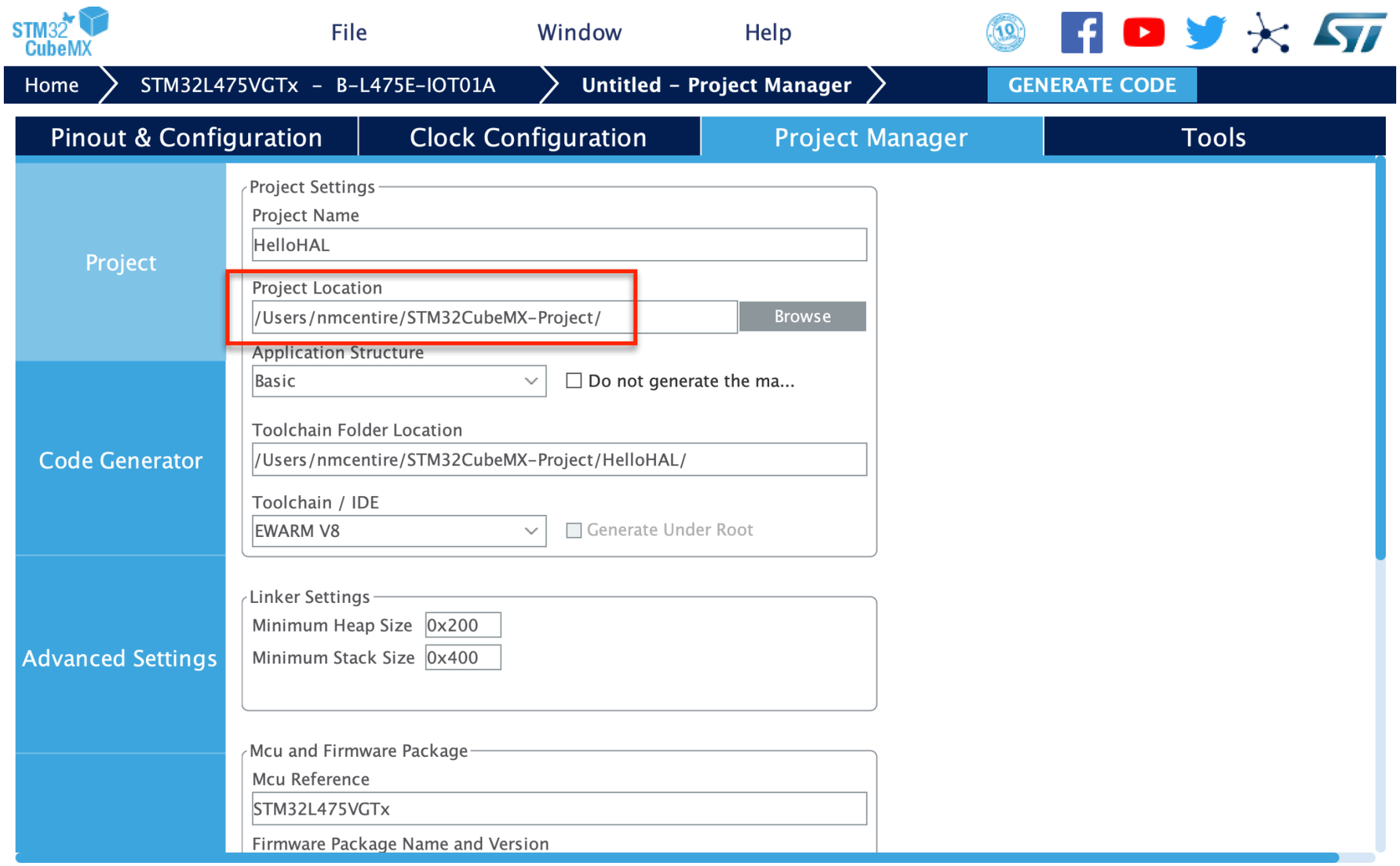
Step: Observe “Project Manager”



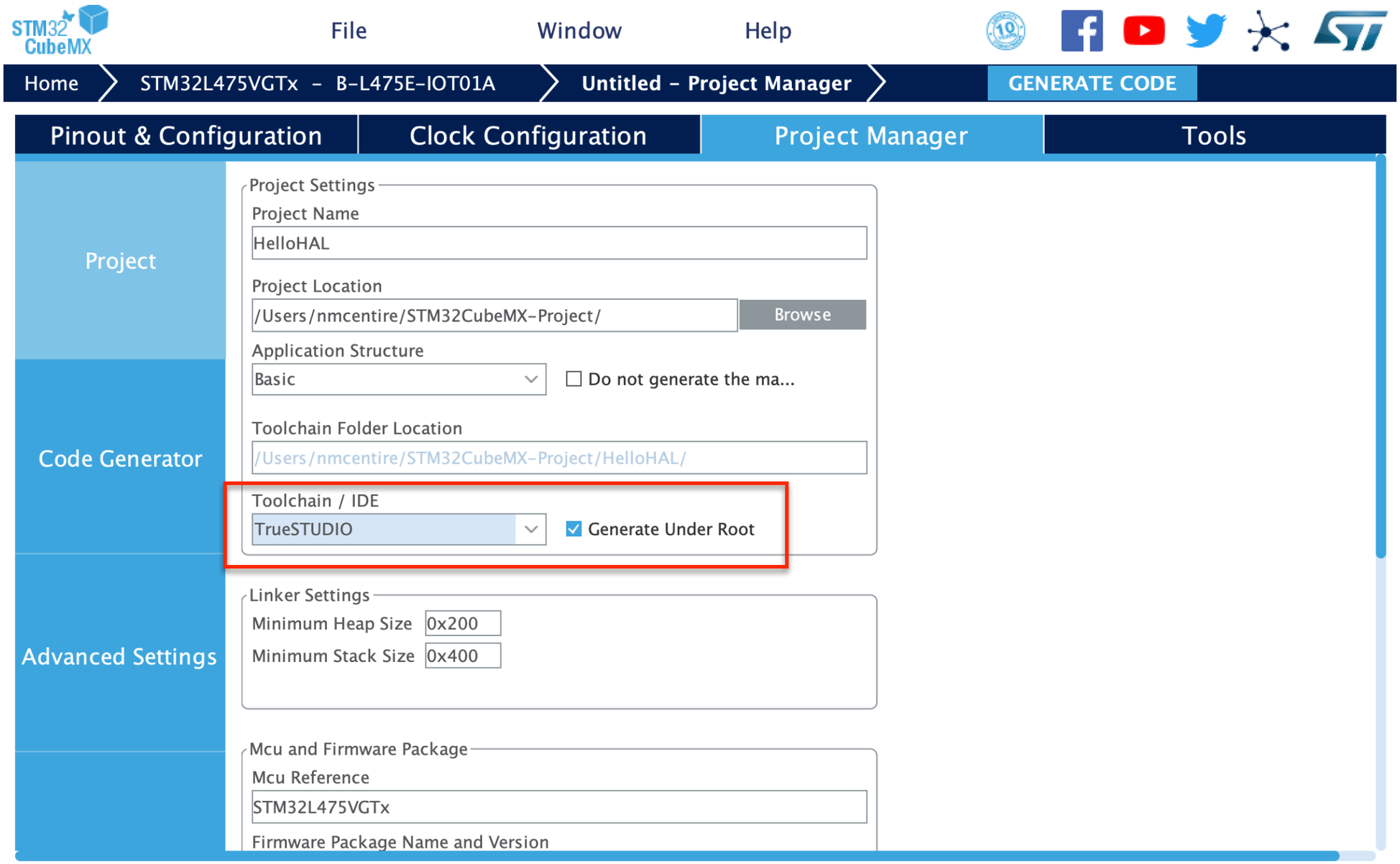
Step: Enter “HelloHAL” for Project Name



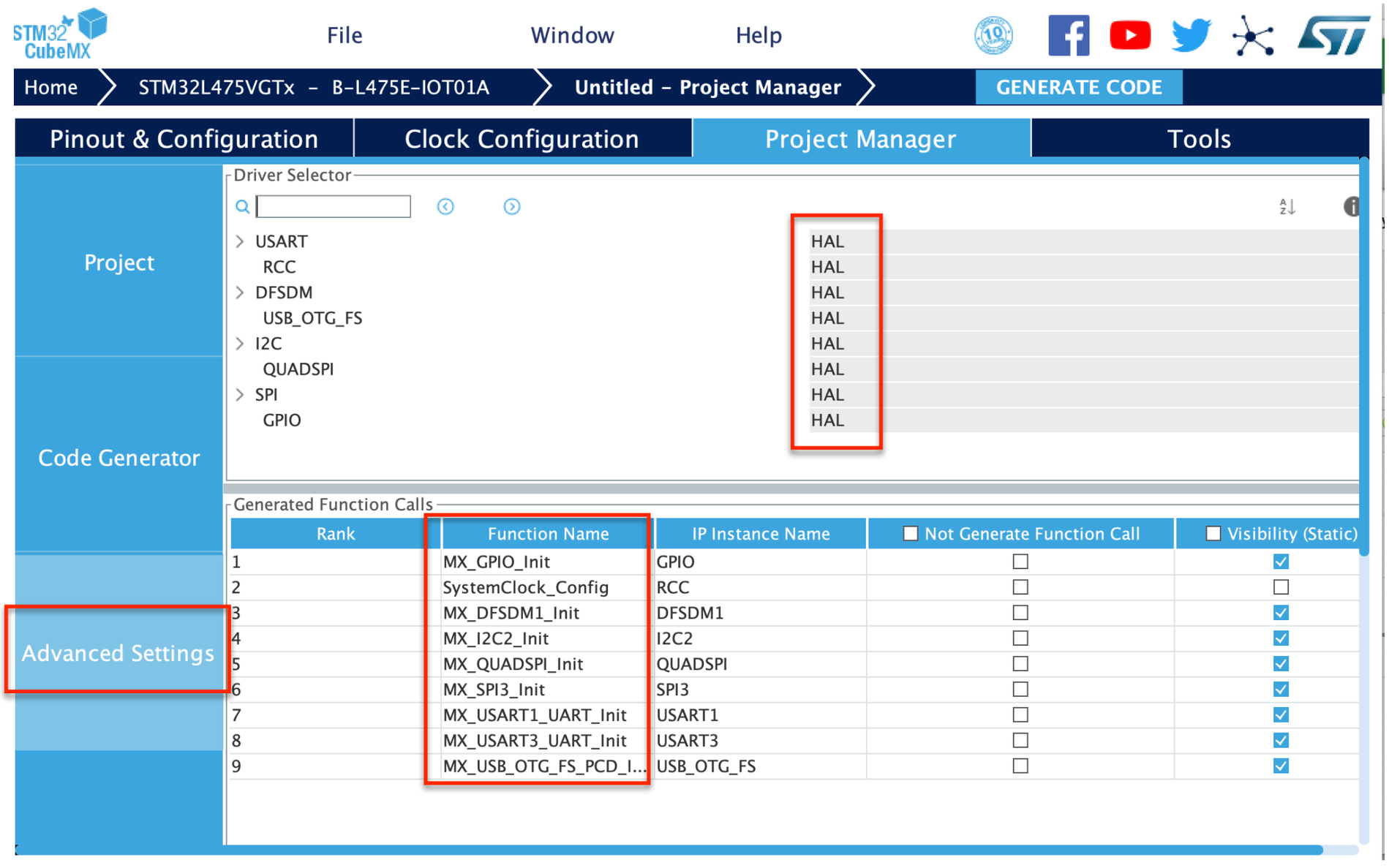
Step: Observe location where project will be stored



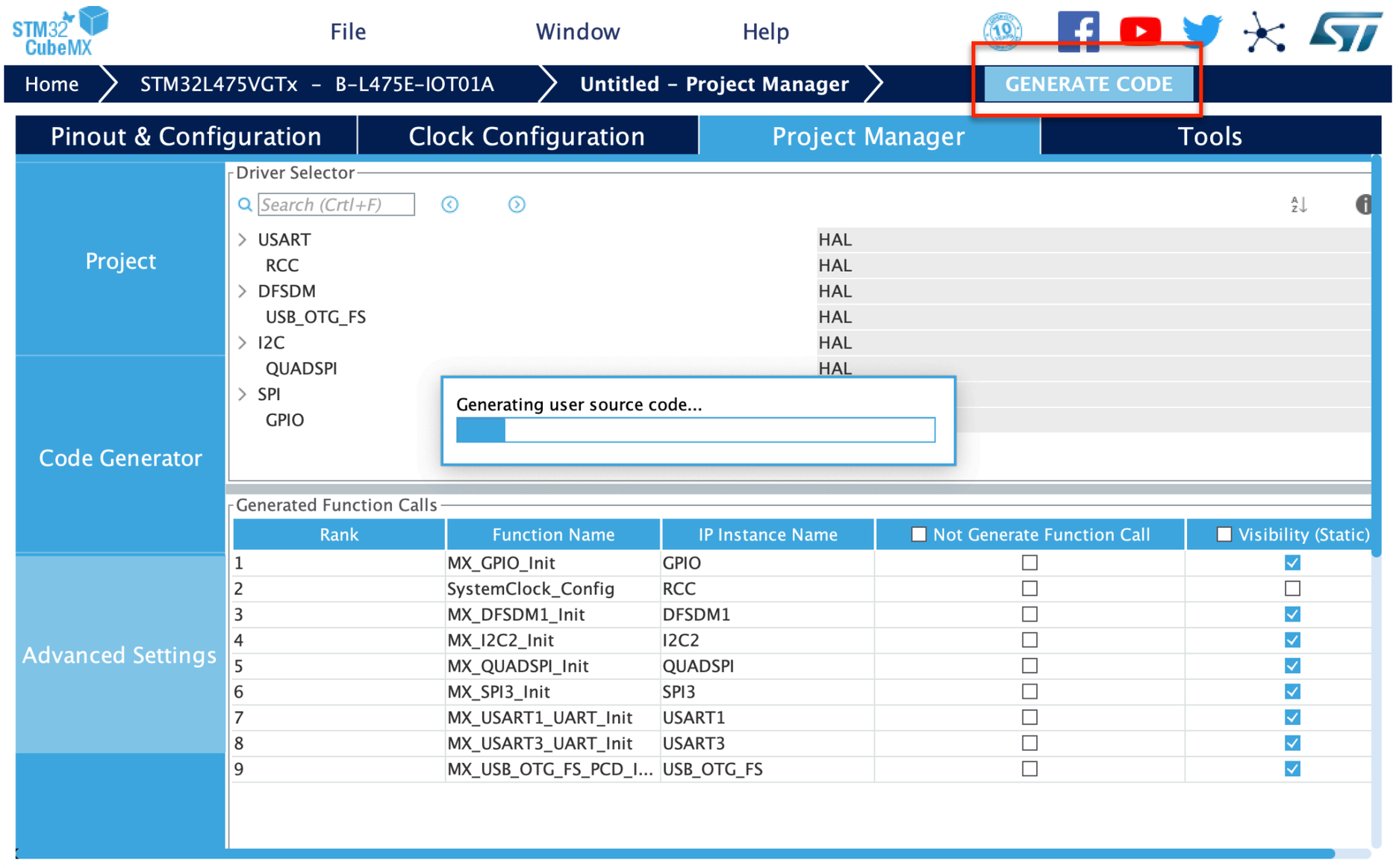
Step: Select “TrueStudio” for Toolchain / IDE



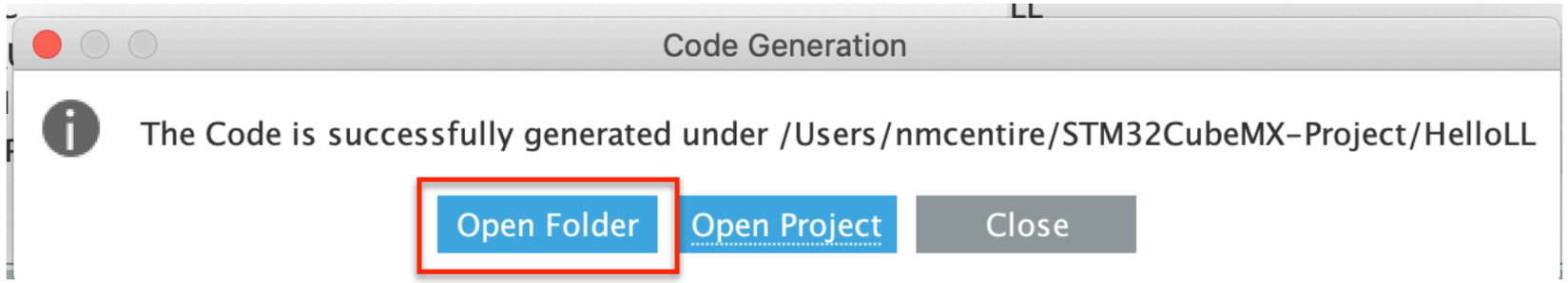
Step: Click on “Advanced Settings”, observing HAL and “Function Name”



Step: Click on “Generate Code”

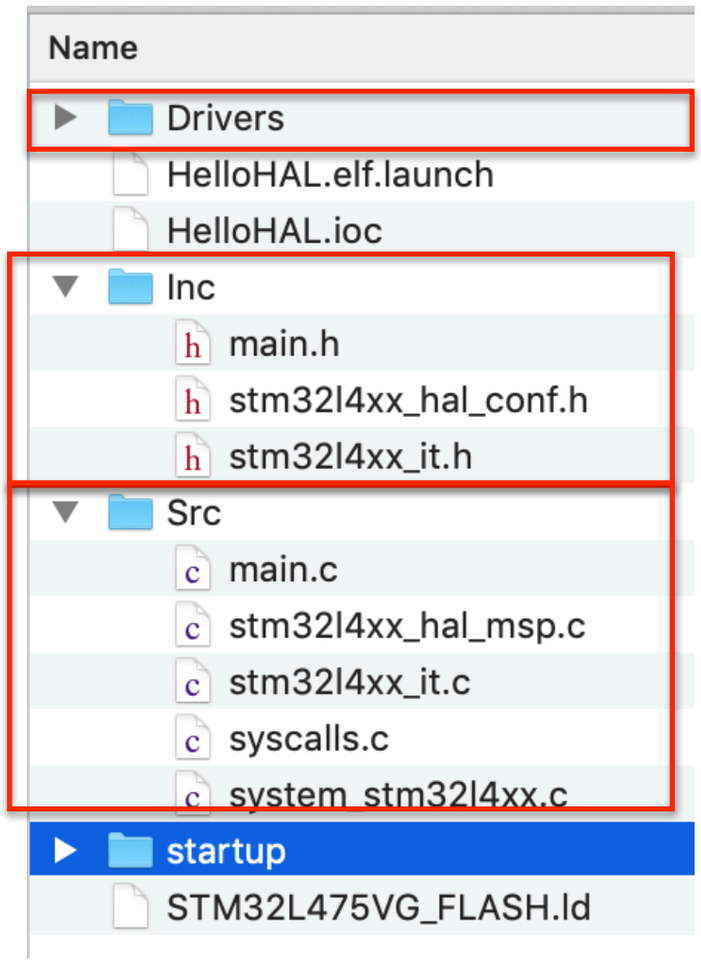


Step: Select “Open Folder”



Tour of Generated Project

Step: View Open Folder



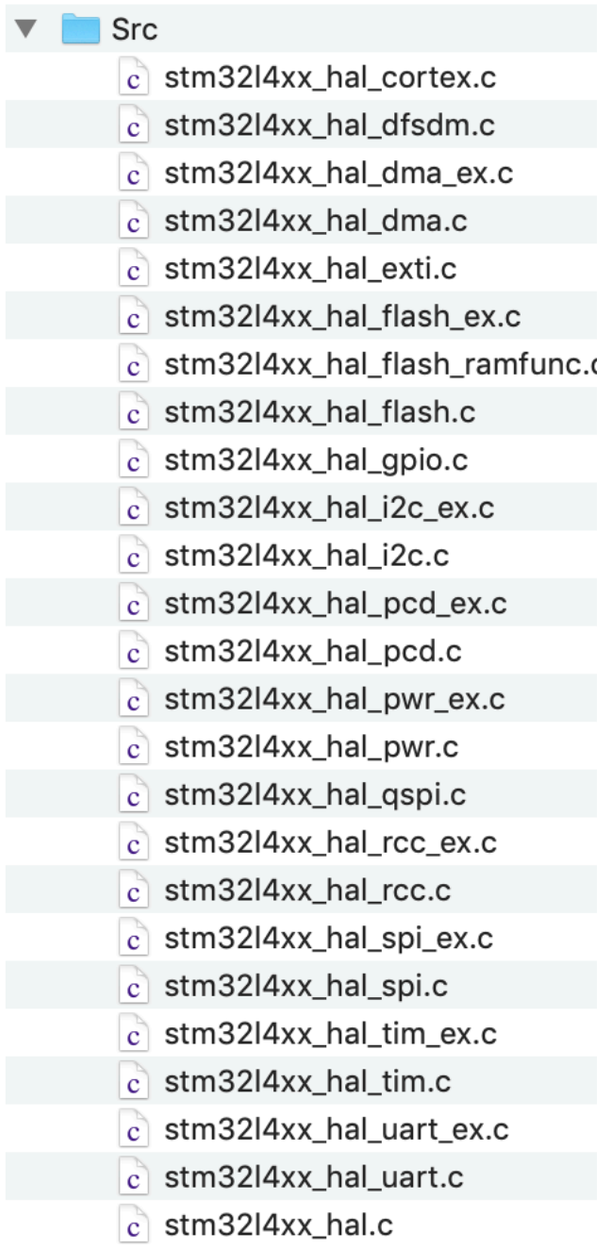
Note: The following slides will give a brief tour of key files/directories in the project.

Step: View “Drivers/ STM32L4xx_HAL_Driver/Inc” directory

- stm32l4xx_hal_cortex.h
- stm32l4xx_hal_def.h
- stm32l4xx_hal_dfsdm.h
- stm32l4xx_hal_dma_ex.h
- stm32l4xx_hal_dma.h
- stm32l4xx_hal_exti.h
- stm32l4xx_hal_flash_ex.h
- stm32l4xx_hal_flash_ramfunc.h
- stm32l4xx_hal_flash.h
- stm32l4xx_hal_gpio_ex.h
- stm32l4xx_hal_gpio.h
- stm32l4xx_hal_i2c_ex.h
- stm32l4xx_hal_i2c.h
- stm32l4xx_hal_pcd_ex.h
- stm32l4xx_hal_pcd.h
- stm32l4xx_hal_pwr_ex.h
- stm32l4xx_hal_pwr.h
- stm32l4xx_hal_qspi.h
- stm32l4xx_hal_rcc_ex.h
- stm32l4xx_hal_rcc.h
- stm32l4xx_hal_spi_ex.h
- stm32l4xx_hal_spi.h
- stm32l4xx_hal_tim_ex.h
- stm32l4xx_hal_tim.h
- stm32l4xx_hal_uart_ex.h
- stm32l4xx_hal_uart.h
- stm32l4xx_hal.h

Observe HAL Drivers

Step: View “Drivers/ STM32L4xx_HAL_Driver/Src” directory



Observe HAL Drivers

Step: View “Inc/ stm32l4xx_hal_conf.h” - Part 1

```
46  /* ########################## Module Selection ##########################
47  /**
48    * @brief This is the list of modules to be used in the HAL driver
49    */
50
51  #define HAL_MODULE_ENABLED
52  /*#define HAL_ADC_MODULE_ENABLED   */
91   /*#define HAL_I2C_MODULE_ENABLED   */
92  #define HAL_UART_MODULE_ENABLED
93  /*#define HAL_USART_MODULE_ENABLED   */
94  /*#define HAL_WWDG_MODULE_ENABLED   */
95  /*#define HAL_EXTI_MODULE_ENABLED   */
96  #define HAL_GPIO_MODULE_ENABLED
97  #define HAL_EXTI_MODULE_ENABLED
98  #define HAL_I2C_MODULE_ENABLED
99  #define HAL_DMA_MODULE_ENABLED
100 #define HAL_RCC_MODULE_ENABLED
101 #define HAL_FLASH_MODULE_ENABLED
102 #define HAL_PWR_MODULE_ENABLED
103 #define HAL_CORTEX_MODULE_ENABLED
```

Observe how Features are enabled/disabled

Step: View “Inc/ stm32l4xx_hal_conf.h” - Part 2

```
105  /* ########################## Oscillator Values adaptation ####################*/
106  /**
107    * @brief Adjust the value of External High Speed oscillator (HSE) used in your application.
108    *        This value is used by the RCC HAL module to compute the system frequency
109    *        (when HSE is used as system clock source, directly or through the PLL).
110    */
111  #if !defined  (HSE_VALUE)
112    #define HSE_VALUE    ((uint32_t)8000000U) /*!< Value of the External oscillator in Hz */
113  #endif /* HSE_VALUE */
114
115  #if !defined  (HSE_STARTUP_TIMEOUT)
116    #define HSE_STARTUP_TIMEOUT    ((uint32_t)100U)   /*!< Time out for HSE start up, in ms */
117  #endif /* HSE_STARTUP_TIMEOUT */
118
```

Observe oscillator values

Step: View “Inc/stm32l4xx_hal_conf.h” - Part 3

```
217  /* Includes ------------------------------
218  /**
219    * @brief Include module's header file
220    */
221
222  #ifdef HAL_RCC_MODULE_ENABLED
223    #include "stm32l4xx_hal_rcc.h"
224    #include "stm32l4xx_hal_rcc_ex.h"
225  #endif /* HAL_RCC_MODULE_ENABLED */
226
227  #ifdef HAL_EXTI_MODULE_ENABLED
228    #include "stm32l4xx_hal_exti.h"
229  #endif /* HAL_EXTI_MODULE_ENABLED */
230
231  #ifdef HAL_GPIO_MODULE_ENABLED
232    #include "stm32l4xx_hal_gpio.h"
233  #endif /* HAL_GPIO_MODULE_ENABLED */
234
```

Observe Pulling in other Header files Based on Configuration options

Step: View “Inc/stm32l4xx_hal_conf.h” - Part 4

```
412  /* Exported macro ------------------------------------------------------*/
413  #ifdef  USE_FULL_ASSERT
414  /**
415    * @brief  The assert_param macro is used for function's parameters check.
416    * @param  expr: If expr is false, it calls assert_failed function
417    *         which reports the name of the source file and the source
418    *         line number of the call that failed.
419    *         If expr is true, it returns no value.
420    * @retval None
421    */
422    #define assert_param(expr) ((expr) ? (void)0U : assert_failed((char *)__FILE__, __LINE__))
423  /* Exported functions ------------------------------------------------------- */
424    void assert_failed(char *file, uint32_t line);
425  #else
426    #define assert_param(expr) ((void)0U)
427  #endif /* USE_FULL_ASSERT */
```

stm32l4xx_hal_msp.c

```
63    */
64  void HAL_MspInit(void)
65  {
66    /* USER CODE BEGIN MspInit 0 */
67
68    /* USER CODE END MspInit 0 */
69
70    __HAL_RCC_SYSCFG_CLK_ENABLE();
71    __HAL_RCC_PWR_CLK_ENABLE();
72
73    /* System interrupt init*/
74
75    /* USER CODE BEGIN MspInit 1 */
76
77    /* USER CODE END MspInit 1 */
78  }
```

main.c - Part 1

```
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DFSDM1_Init();
  MX_I2C2_Init();
  MX_QUADSPI_Init();
  MX_SPI3_Init();
  MX_USART1_UART_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

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

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

main.c - Part 2

```
473  static void MX_GPIO_Init(void)
474  {
475    GPIO_InitTypeDef GPIO_InitStruct = {0};
476
477    /* GPIO Ports Clock Enable */
478    __HAL_RCC_GPIOE_CLK_ENABLE();
479    __HAL_RCC_GPIOC_CLK_ENABLE();
480    __HAL_RCC_GPIOA_CLK_ENABLE();
481    __HAL_RCC_GPIOB_CLK_ENABLE();
482    __HAL_RCC_GPIOD_CLK_ENABLE();
483
484    /*Configure GPIO pin Output Level */
485    HAL_GPIO_WritePin(GPIOE, M24SR64_Y_RF_DIS
486
487    /*Configure GPIO pin Output Level */
488    HAL_GPIO_WritePin(GPIOA, ARD_D10_Pin|SPBT
489
490    /*Configure GPIO pin Output Level */
491    HAL_GPIO_WritePin(GPIOB, ARD_D8_Pin|ISM43
492                             |SPSGRF_915_SDN_P
```

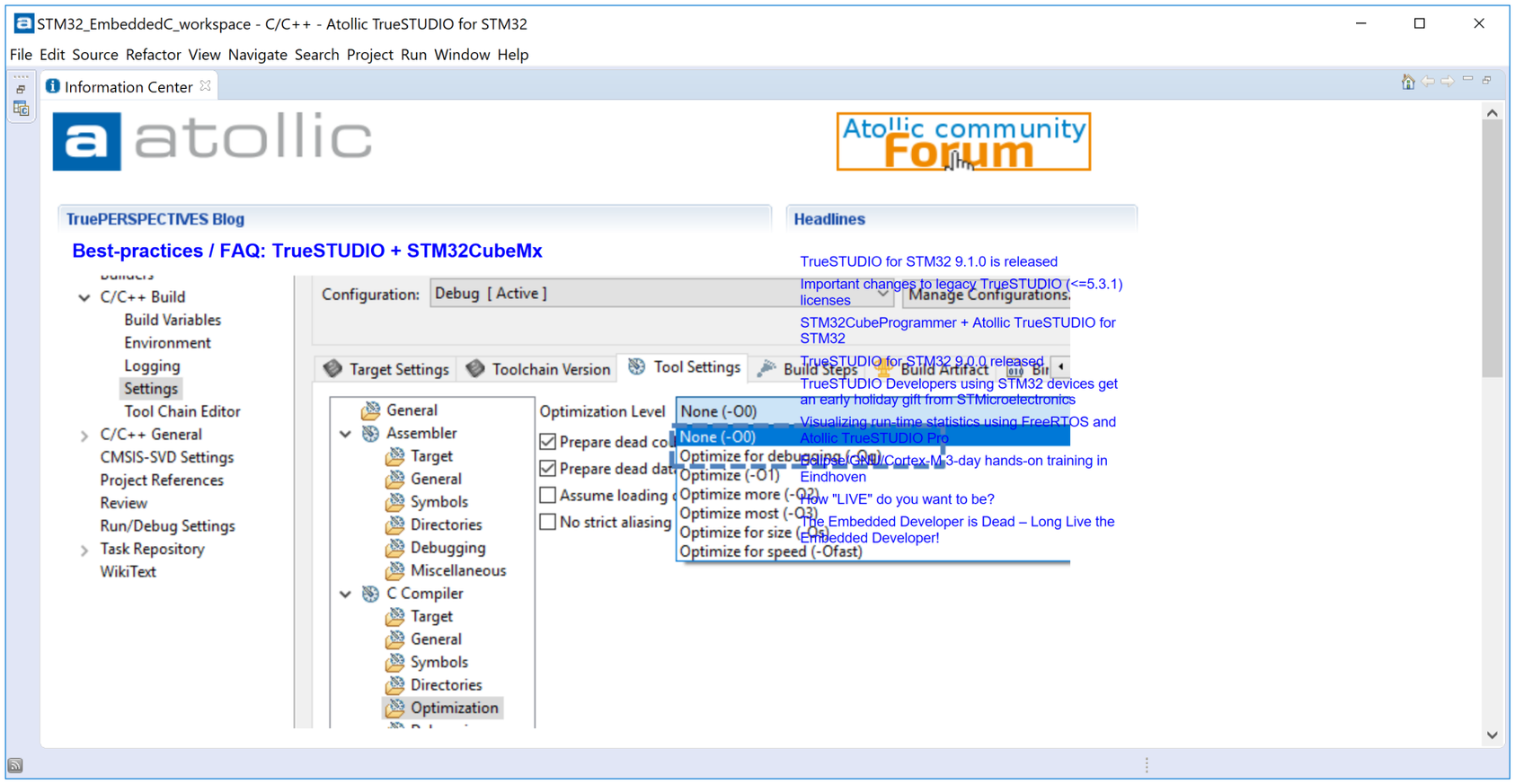
Notice use of HAL_ APIs

Summary so far…

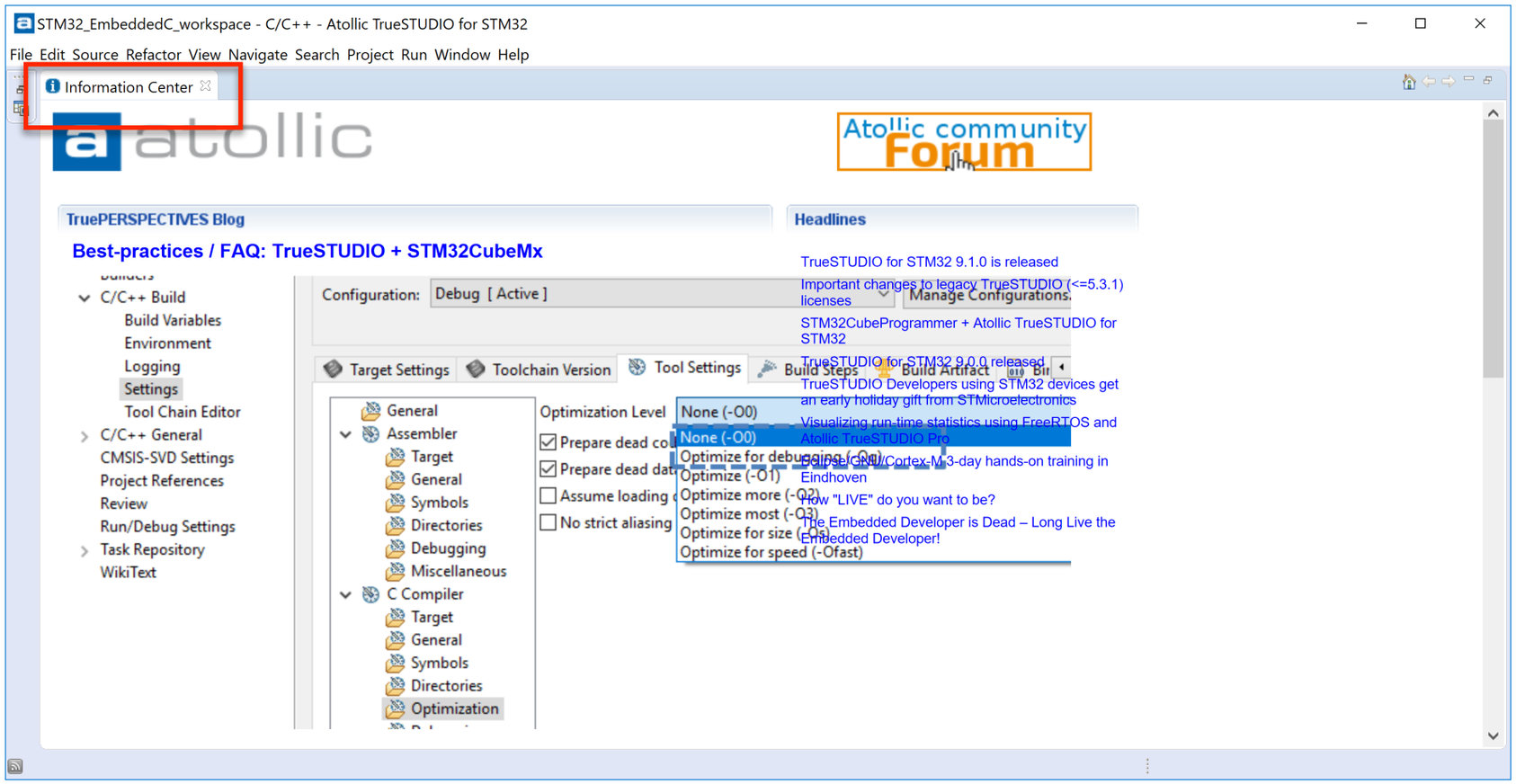
- STM32CubeMX generates HAL Code
- HAL Driver Code
  - Drivers/STM32L4xx_HAL_Driver/Inc
  - Drivers/STM32L4xx_HAL_Driver/Src

Use TrueStudio To Toggle LED2 On/Off Using HAL

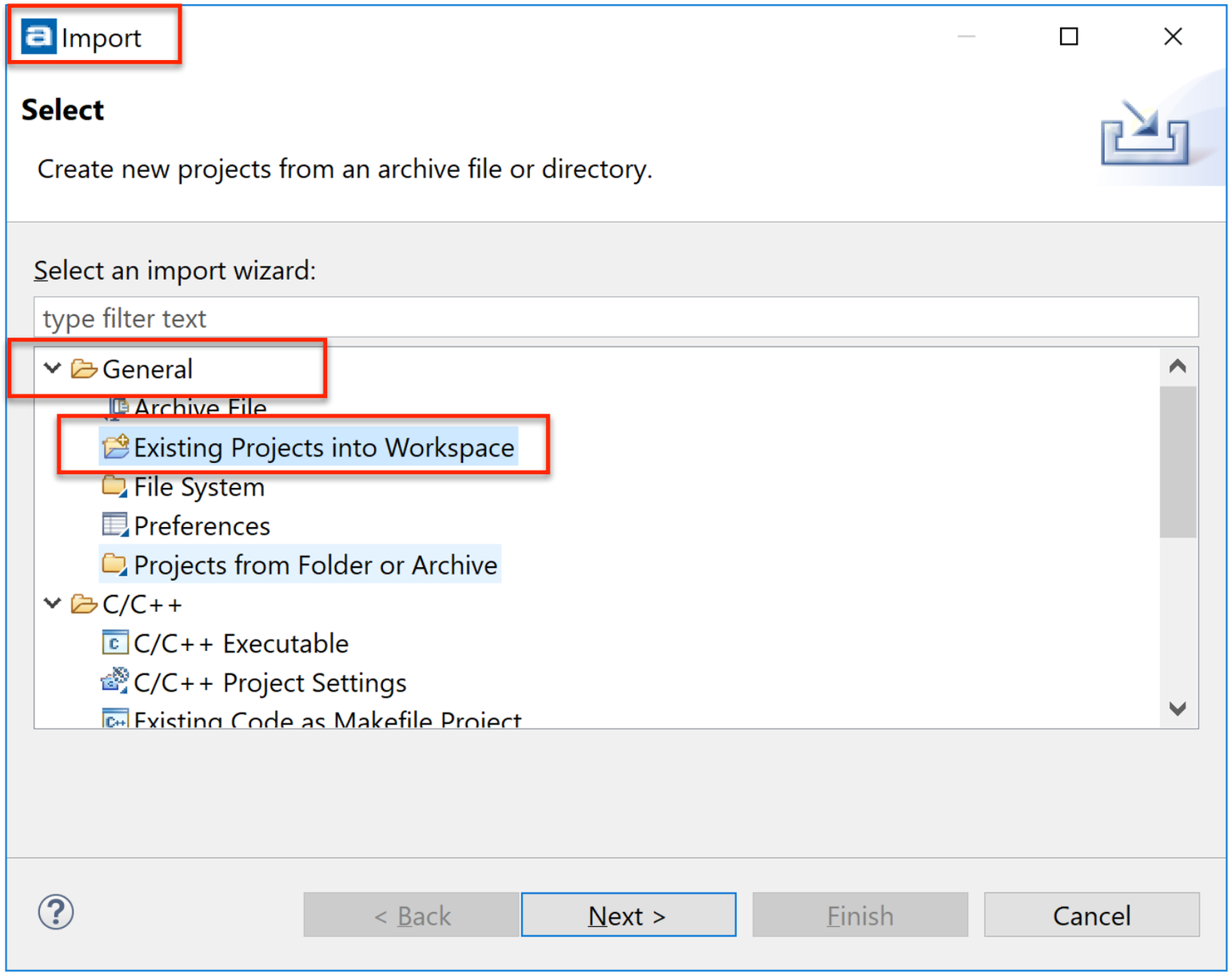
Step: Startup TrueStudio



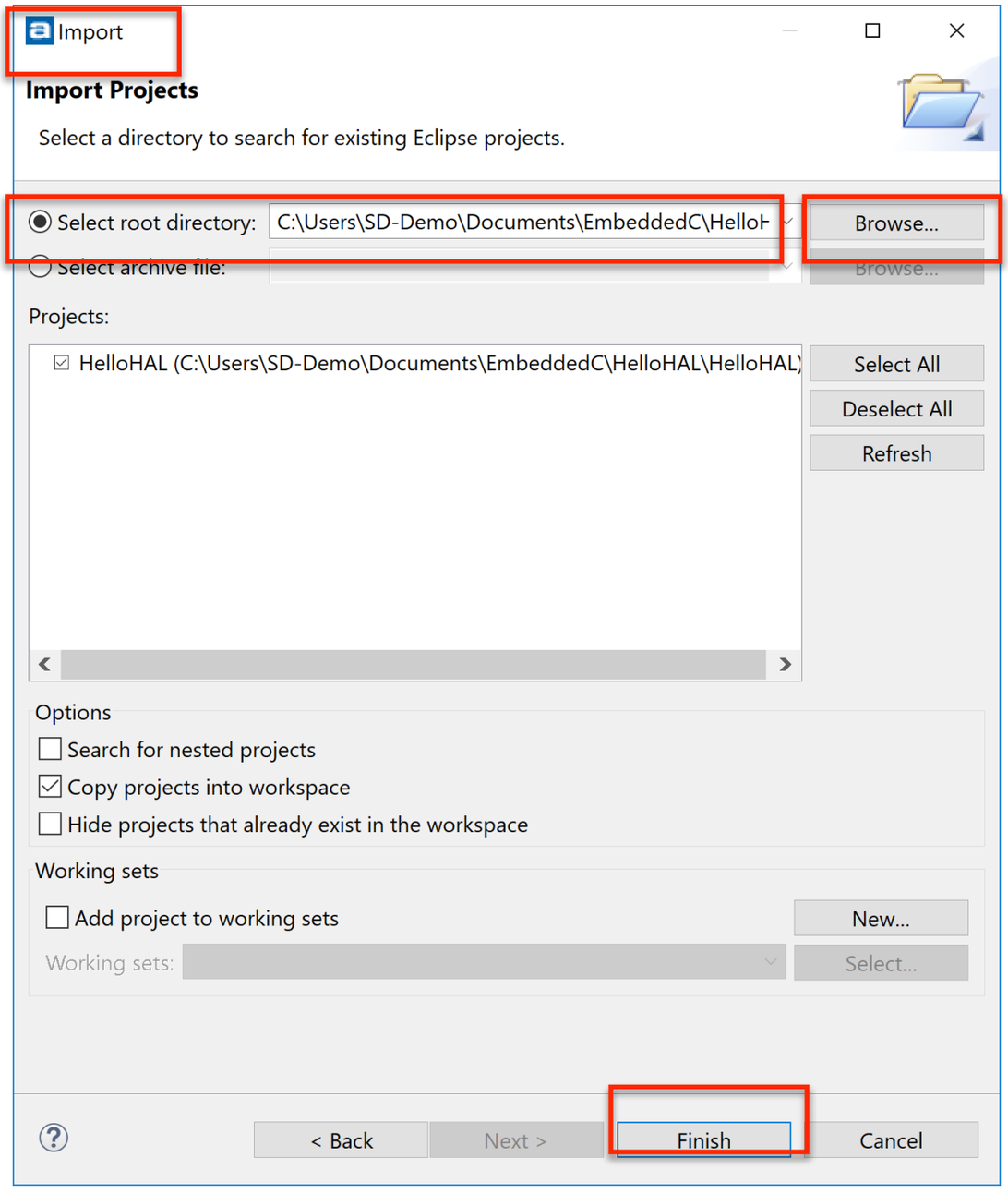
Step: Click to Close “Information Center”



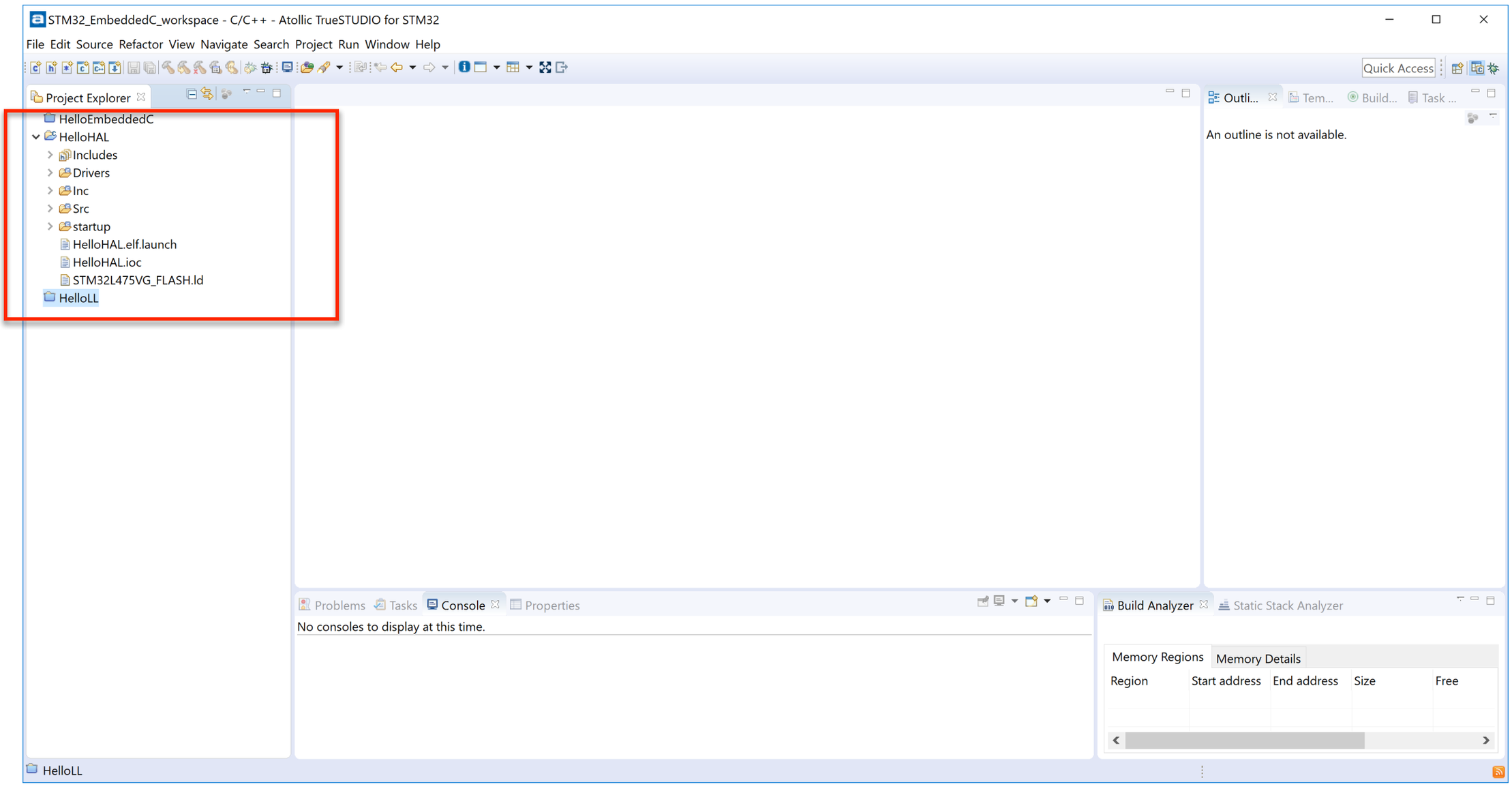
Step: File, Import, General, Existing Projects into Workspace



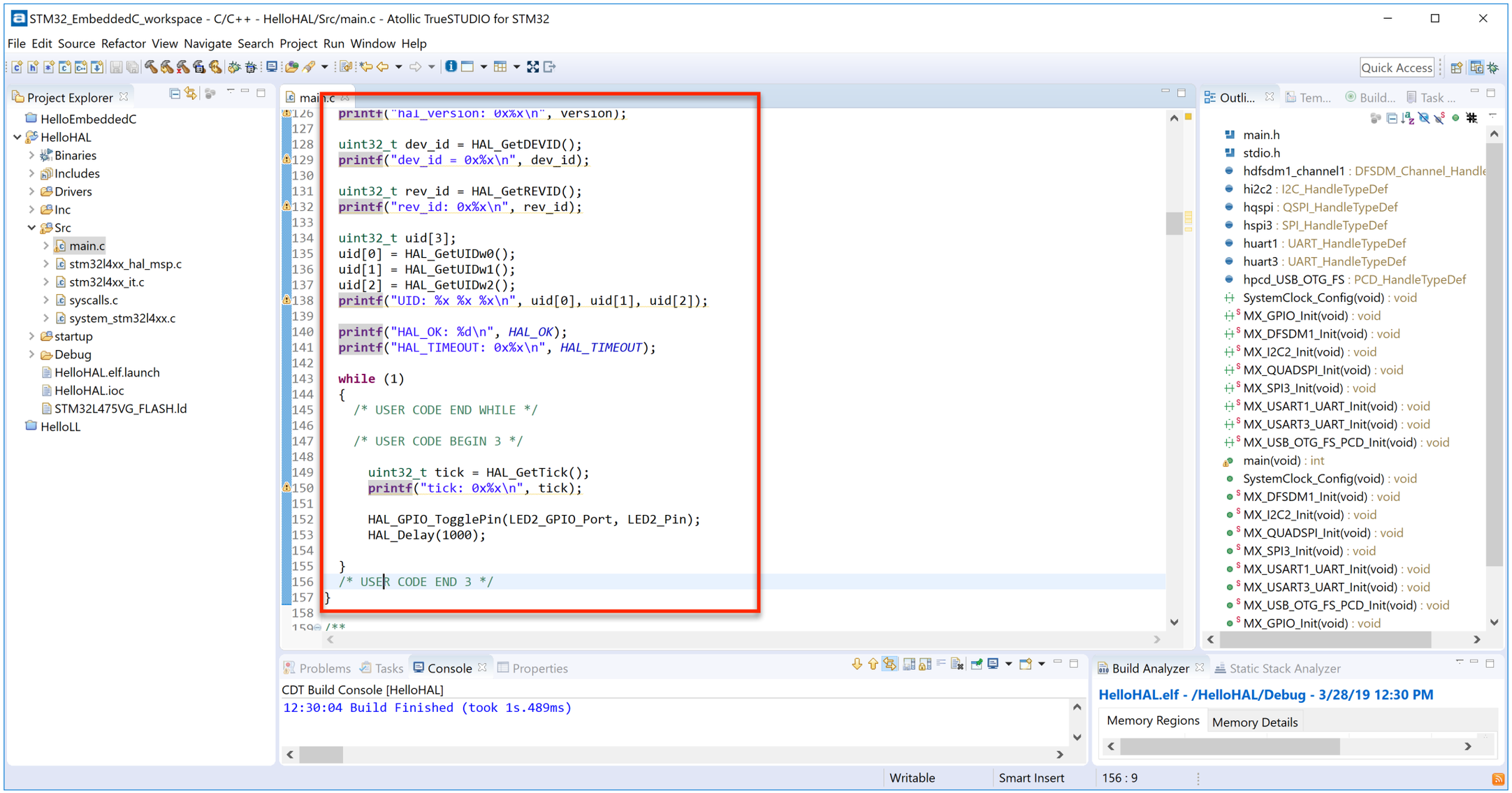
Step: Select Root Directory (Browse to directory as needed)



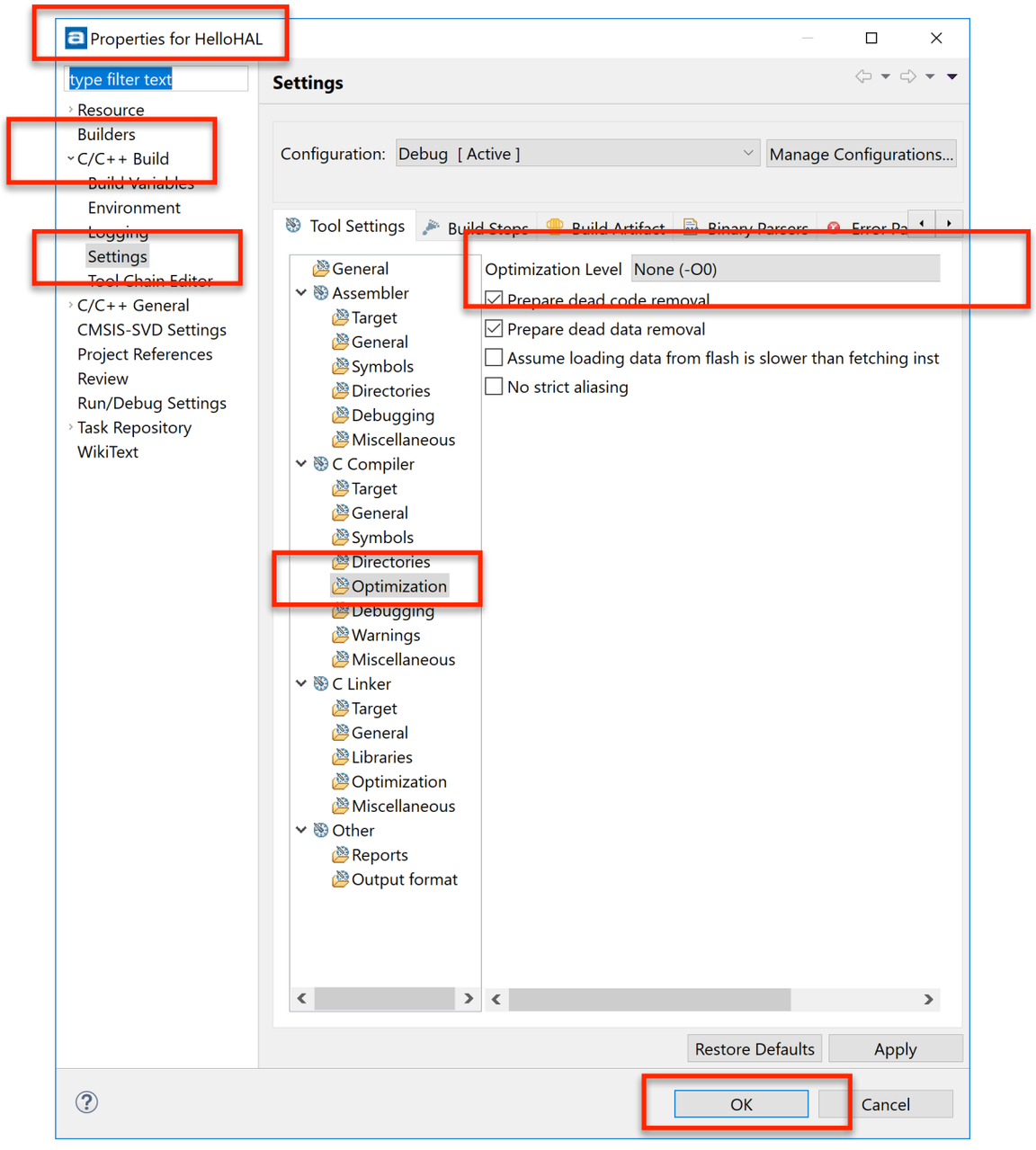
Step: Observe Results



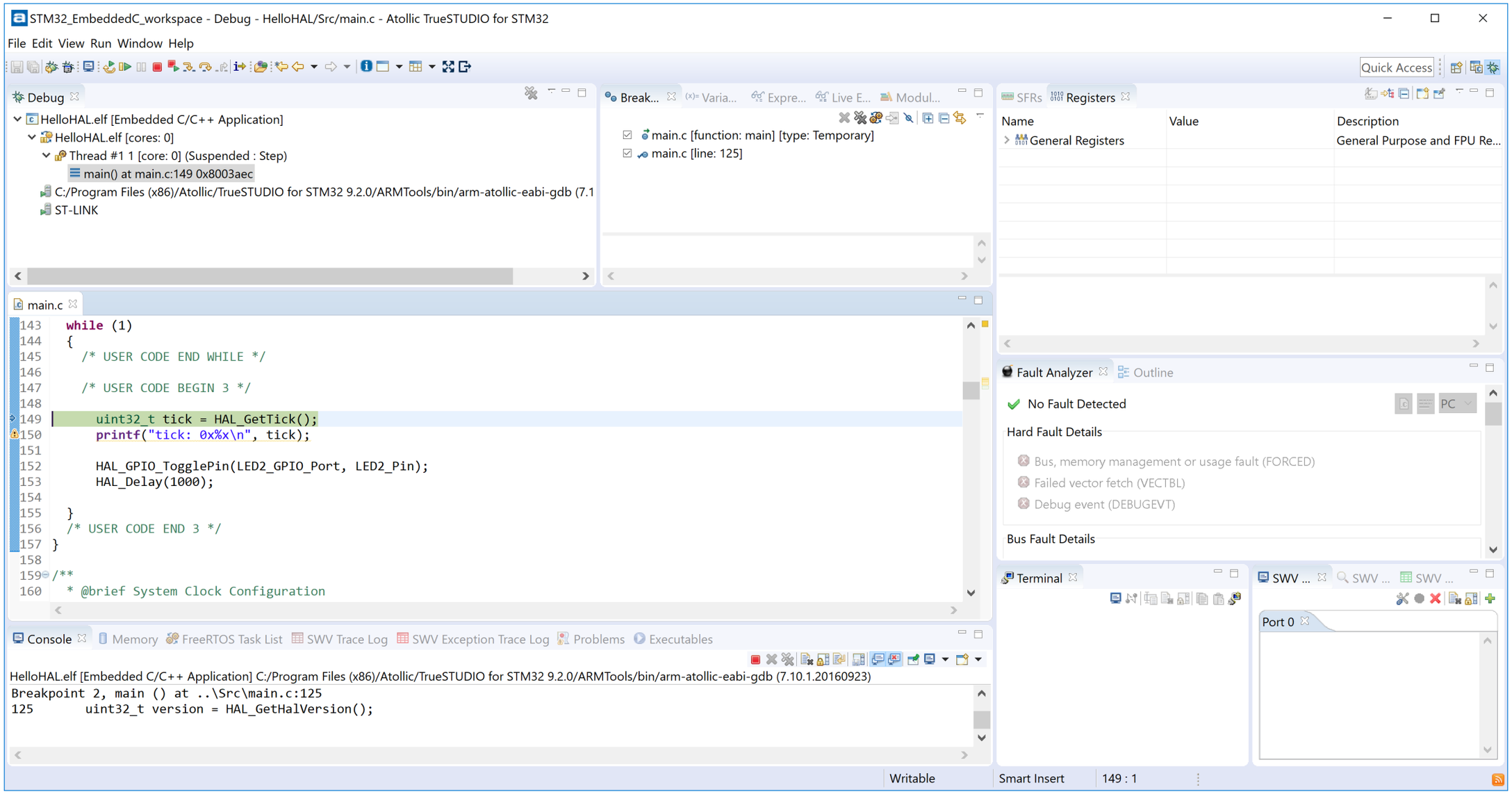
Step: Edit main.c



Step: Project, Properties, C/C++ Build, Settings, C Compiler, Optimization, None

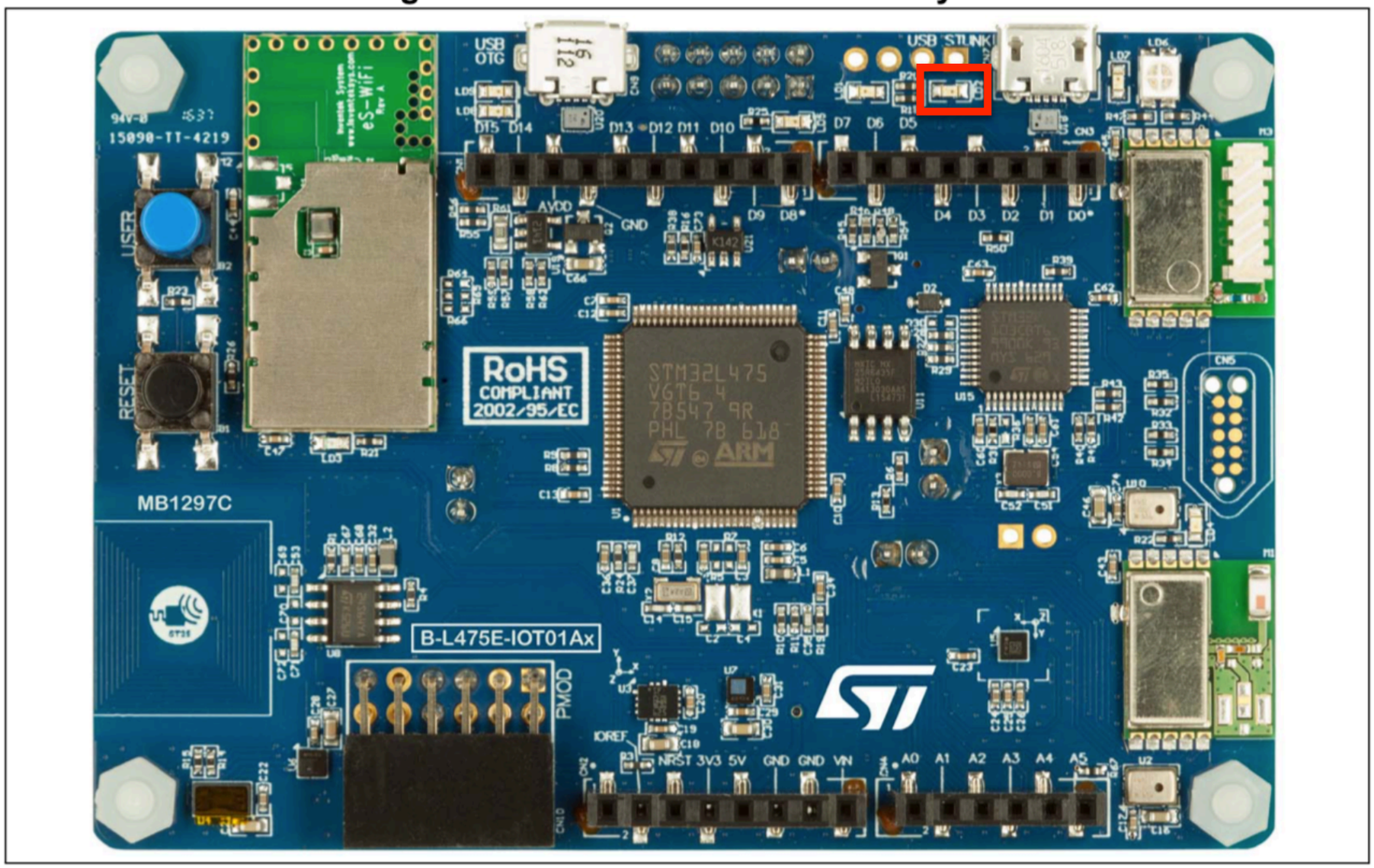


Step: Build and Run Project in Debug Mode



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 HAL and Embedded C
- STM32CubeMX and HAL Code Generation
- Tour of HAL
- TrueStudio and HAL
    - Hello Blinking LED