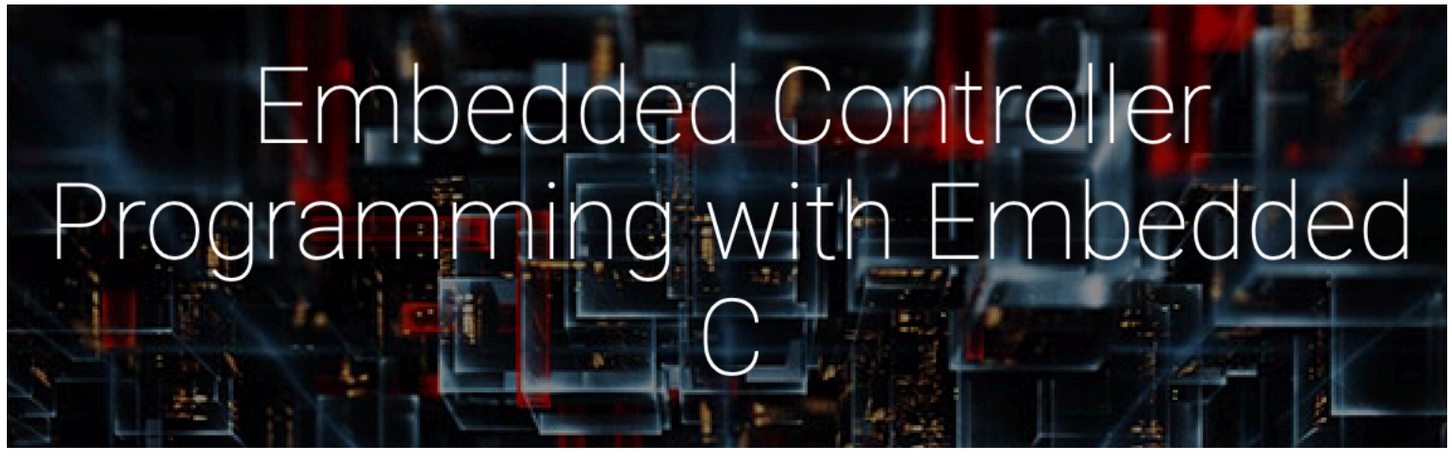


Lesson 8 - I2C and Embedded C

Norman McEntire
norman.mcentire@gmail.com

Contents

- Introduction to I2C
- HAL and I2C
- STM32CubeMX and I2C Code Generation
- Tour of I2C
- TrueStudio and I2C
  - HTS221 Humidity and Temperature Sensor

Introduction to I2C

https://en.wikipedia.org/wiki/I²C

- I2C = Inter-Integrated Circuit (“I-squared C”)
- Serial Protocol
- Synchronous
- Multi-Master
- Multi-Slave
- Widely used for attaching low-speed peripherals to MCUs
- 1982: Invented by Philips Semiconductor (now NXP)
- 1995: SMBus, defined by Intel, is a subset of I2C to promote robustness and interop - hence some I2C systems support both I2C and SMBus

STMicroelectronics 24C08: Serial EEPROM with I²C bus

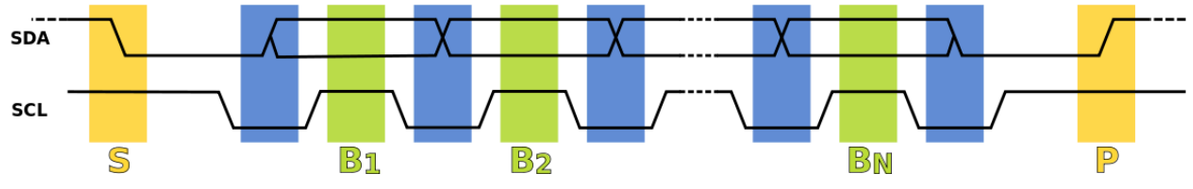
I2C Architecture

- I2C uses two bidirectional open collector (open drain) lines, pulled up with resistors
  - SDA - Serial Data
  - SCK - Serial Clock
- I2C reference design has 7-bit address
  - Also offers rarely-used 10-bit address extension
- I2C Bus Speeds
  - 100kbit/s - Standard Mode
  - 400kbit/s - Fast Mode
  - NOTE: Bus speeds are for transfers between master and slave without clock stretching or protocol overhead - See next slide

I2C Address

Field:	**S**	I^2C address field							R/W'	A	I^2C message sequences...	P
Type	**Start**	**Byte 1**								**ACK**	**Byte X etc... Rest of the read or write message goes here**	**Stop**
Bit position in byte X		**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**			
7-bit address pos		**7**	**6**	**5**	**4**	**3**	**2**	**1**				
Note		MSB						LSB	1 = Read 0 = Write			

I2C Timing



1. Data transfer is initiated with a *start* bit (S) signaled by SDA being pulled low while SCL stays high.
2. SCL is pulled low, and SDA sets the first data bit level while keeping SCL low (during blue bar time).
3. The data are sampled (received) when SCL rises for the first bit (B1). For a bit to be valid, SDA must not change between a rising edge of SCL and the subsequent falling edge (the entire green bar time).
4. This process repeats, SDA transitioning while SCL is low, and the data being read while SCL is high (B2, ...Bn).
5. The final bit is followed by a clock pulse, during which SDA is pulled low in preparation for the *stop* bit.
6. A *stop* bit (P) is signaled when SCL rises, followed by SDA rising.

I2C Protocol Overhead

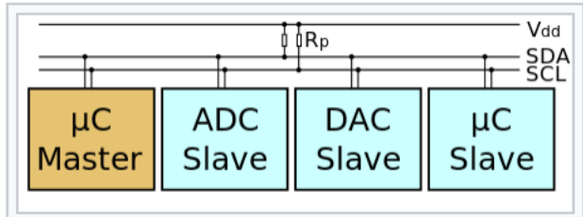
- Bus speeds are for transfers between master and slave without clock stretching or protocol overhead
- Protocol overhead includes
  - Slave Address
  - Register Address (if any)
  - ACK/NACK

I2C Number of Nodes

- Number of Nodes on IC2 bus
  - Limited by address space (7-bits)
  - Limited by total capacitance on the bus - 400pf
  - Must be close together - less than a few meters
  - High impedance and low noise immunity require common ground plane, i.e. on the same PCB

I2C Master and Slave Modes

- Master Mode (e.g STM32L MCU)
  - Generates clock
  - Initiates communication
- Slave Mode (e.g HTS221 Sensor Device)
  - Receives clock
  - Responds to communication



An example schematic with one master (a microcontroller), three slave nodes (an ADC, a DAC, and a microcontroller), and pull-up resistors R_p

I2C Modes of Operation

- Four Modes
    - Master Transmit
    - Master Receive
    - Slave Transmit
    - Slave Receive

I2C Transfer Example

- Master Sends: [START][7-Bit-Address][R/W-Bit]
- Slave Sends: [ACK]
- Master: Either sends or receives based on R/W-Bit
  - 1 = Read, 0 = Write
  - An ACK sent after every byte

I2C Message Protocols

- I2C defines basic transactions
  - All begin with START and end with STOP
  - Single Message (master writes to slave)
  - Single Message (master reads from slave)
  - Combined (master issues two or more operations to slave)

I2C Physical Layer

- SCL and SDA are open drain
    - Requires pull up resistors
- A line is driven low
    - But pull-up resistor make line high (output goes high impedance)

I2C Clock Stretching

- A slave can hold SCL low after receiving (or sending) indicating that it is not yet ready to process more data
    - The master must wait for the clock line to go high
- NOTE: Some masters do NOT support clock stretching
    - These are called “2-wire” interface instead of I2C
        - Clock stretching is a key feature of I2C

HAL and I2C

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

HAL_I2C_StateTypeDef

```
124  typedef enum
125  {
126    HAL_I2C_STATE_RESET             = 0x00U,   /*!< Peripheral is not yet Initialized
127    HAL_I2C_STATE_READY             = 0x20U,   /*!< Peripheral Initialized and ready for use
128    HAL_I2C_STATE_BUSY              = 0x24U,   /*!< An internal process is ongoing
129    HAL_I2C_STATE_BUSY_TX           = 0x21U,   /*!< Data Transmission process is ongoing
130    HAL_I2C_STATE_BUSY_RX           = 0x22U,   /*!< Data Reception process is ongoing
131    HAL_I2C_STATE_LISTEN            = 0x28U,   /*!< Address Listen Mode is ongoing
132    HAL_I2C_STATE_BUSY_TX_LISTEN    = 0x29U,   /*!< Address Listen Mode and Data Transmission
133                                                   process is ongoing
134    HAL_I2C_STATE_BUSY_RX_LISTEN    = 0x2AU,   /*!< Address Listen Mode and Data Reception
135                                                   process is ongoing
136    HAL_I2C_STATE_ABORT             = 0x60U,   /*!< Abort user request ongoing
137    HAL_I2C_STATE_TIMEOUT           = 0xA0U,   /*!< Timeout state
138    HAL_I2C_STATE_ERROR             = 0xE0U    /*!< Error
139
140  } HAL_I2C_StateTypeDef;
```

HAL_I2C_ModeTypeDef

```
164  typedef enum
165  {
166    HAL_I2C_MODE_NONE               = 0x00U,   /*!< No I2C communication on going
167    HAL_I2C_MODE_MASTER             = 0x10U,   /*!< I2C communication is in Master Mode
168    HAL_I2C_MODE_SLAVE              = 0x20U,   /*!< I2C communication is in Slave Mode
169    HAL_I2C_MODE_MEM                = 0x40U    /*!< I2C communication is in Memory Mode
170
171  } HAL_I2C_ModeTypeDef;
```

HAL_I2C_ERROR_...

```
181  #define HAL_I2C_ERROR_NONE      (0x00000000U)    /*!< No error              */
182  #define HAL_I2C_ERROR_BERR      (0x00000001U)    /*!< BERR error            */
183  #define HAL_I2C_ERROR_ARLO      (0x00000002U)    /*!< ARLO error            */
184  #define HAL_I2C_ERROR_AF        (0x00000004U)    /*!< ACKF error            */
185  #define HAL_I2C_ERROR_OVR       (0x00000008U)    /*!< OVR error             */
186  #define HAL_I2C_ERROR_DMA       (0x00000010U)    /*!< DMA transfer error    */
187  #define HAL_I2C_ERROR_TIMEOUT   (0x00000020U)    /*!< Timeout error         */
188  #define HAL_I2C_ERROR_SIZE      (0x00000040U)    /*!< Size Management error */
189  #define HAL_I2C_ERROR_DMA_PARAM (0x00000080U)    /*!< DMA Parameter Error   */
190  #if (USE_HAL_I2C_REGISTER_CALLBACKS == 1)
191  #define HAL_I2C_ERROR_INVALID_CALLBACK  (0x00000100U)    /*!< Invalid Callback error */
192  #endif /* USE_HAL_I2C_REGISTER_CALLBACKS */
193  #define HAL_I2C_ERROR_INVALID_PARAM     (0x00000200U)    /*!< Invalid Parameters error  */
```

I2C_ADDRESSINGMODE_

...

```
314  /** @defgroup I2C_ADDRESSING_MODE I2C Addressing Mode
315    * @{
316    */
317  #define I2C_ADDRESSINGMODE_7BIT         (0x00000001U)
318  #define I2C_ADDRESSINGMODE_10BIT        (0x00000002U)
```

I2C_HandleTypeDef

```
typedef struct __I2C_HandleTypeDef
{
  I2C_TypeDef                *Instance;      /*!< I2C registers base address

  I2C_InitTypeDef            Init;          /*!< I2C communication parameters

  uint8_t                    *pBuffPtr;     /*!< Pointer to I2C transfer buffer

uint16_t                     XferSize;      /*!< I2C transfer size

__IO uint16_t                XferCount;     /*!< I2C transfer counter

__IO uint32_t                XferOptions;   /*!< I2C sequantial transfer options, this
                                                be a value of @ref I2C_XFEROPTIONS */

__IO uint32_t                PreviousState; /*!< I2C communication Previous state

HAL_StatusTypeDef(*XferISR)(struct __I2C_HandleTypeDef *hi2c, uint32_t ITFlags, uint3

DMA_HandleTypeDef            *hdmatx;       /*!< I2C Tx DMA handle parameters
```

...

HAL I2C Functions

Init and DeInit

```
594  /* Initialization and de-initialization functions**********
595  HAL_StatusTypeDef HAL_I2C_Init(I2C_HandleTypeDef *hi2c);
596  HAL_StatusTypeDef HAL_I2C_DeInit(I2C_HandleTypeDef *hi2c);
597  void HAL_I2C_MspInit(I2C_HandleTypeDef *hi2c);
598  void HAL_I2C_MspDeInit(I2C_HandleTypeDef *hi2c);
```

I2C /O Functions
Polling

```
615  /* IO operation functions  *****************************************************/
616  /******* Blocking mode: Polling */
617  HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData,
618  HAL_StatusTypeDef HAL_I2C_Master_Receive(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData,
619  HAL_StatusTypeDef HAL_I2C_Slave_Transmit(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size, uint32
620  HAL_StatusTypeDef HAL_I2C_Slave_Receive(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size, uint32_
621  HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress,
622  HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, u
623  HAL_StatusTypeDef HAL_I2C_IsDeviceReady(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint32_t Trials,
624
```

I2C I/O Functions
Interrupt

```
625  /******* Non-Blocking mode: Interrupt */
626  HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pI
627  HAL_StatusTypeDef HAL_I2C_Master_Receive_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pDa
628  HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size);
629  HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size);
630  HAL_StatusTypeDef HAL_I2C_Mem_Write_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddr
631  HAL_StatusTypeDef HAL_I2C_Mem_Read_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddre
632
633  HAL_StatusTypeDef HAL_I2C_Master_Sequential_Transmit_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress,
634  HAL_StatusTypeDef HAL_I2C_Master_Sequential_Receive_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, u
635  HAL_StatusTypeDef HAL_I2C_Slave_Sequential_Transmit_IT(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16
636  HAL_StatusTypeDef HAL_I2C_Slave_Sequential_Receive_IT(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_
637  HAL_StatusTypeDef HAL_I2C_EnableListen_IT(I2C_HandleTypeDef *hi2c);
638  HAL_StatusTypeDef HAL_I2C_DisableListen_IT(I2C_HandleTypeDef *hi2c);
639  HAL_StatusTypeDef HAL_I2C_Master_Abort_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress);
```

I2C I/O Functions
DMA

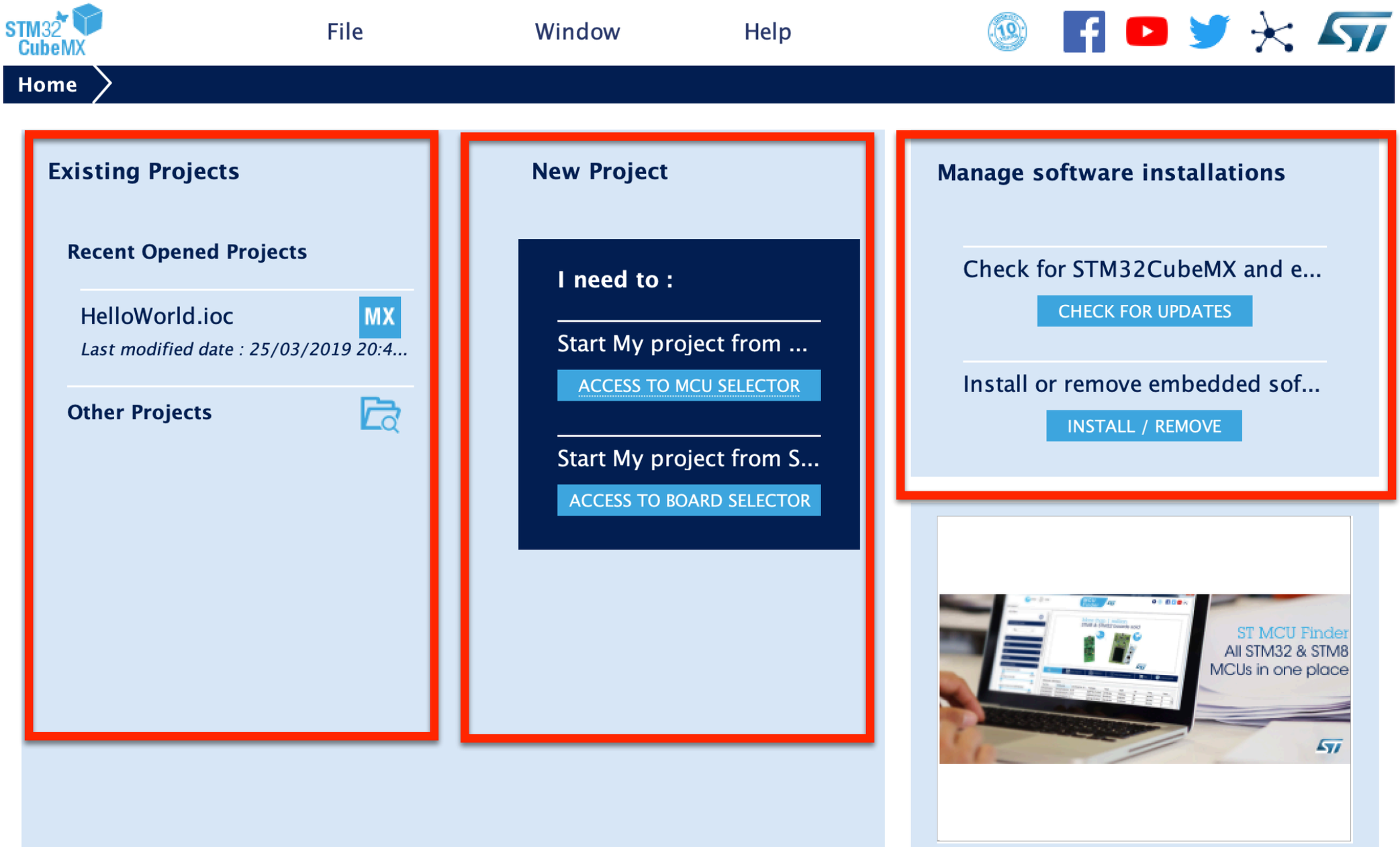
```
641  /******* Non-Blocking mode: DMA */
642  HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *
643  HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *p
644  HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size);
645  HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size);
646  HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAdd
647  HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddr
648
649  HAL_StatusTypeDef HAL_I2C_Master_Sequential_Transmit_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress
650  HAL_StatusTypeDef HAL_I2C_Master_Sequential_Receive_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress,
651  HAL_StatusTypeDef HAL_I2C_Slave_Sequential_Transmit_DMA(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint
652  HAL_StatusTypeDef HAL_I2C_Slave_Sequential_Receive_DMA(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint1
653  /**
```

I2C Peripheral State, Mode, and Error

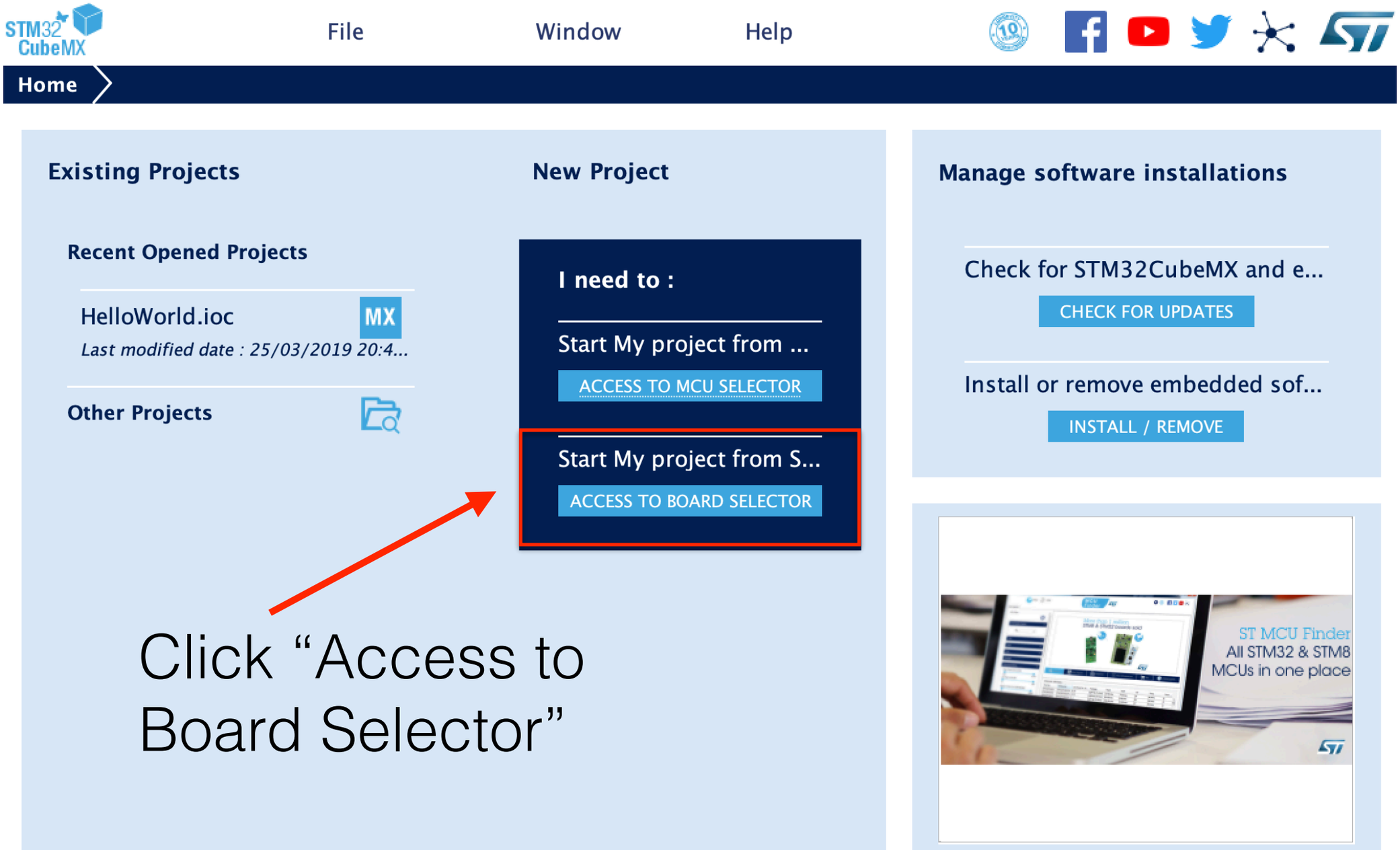
```
/* Peripheral State, Mode and Error functions  ********************
HAL_I2C_StateTypeDef HAL_I2C_GetState(I2C_HandleTypeDef *hi2c);
HAL_I2C_ModeTypeDef  HAL_I2C_GetMode(I2C_HandleTypeDef *hi2c);
uint32_t             HAL_I2C_GetError(I2C_HandleTypeDef *hi2c);
```

STM32CubeMX and Generation of HAL Code

Step: Startup STM32CubeMX

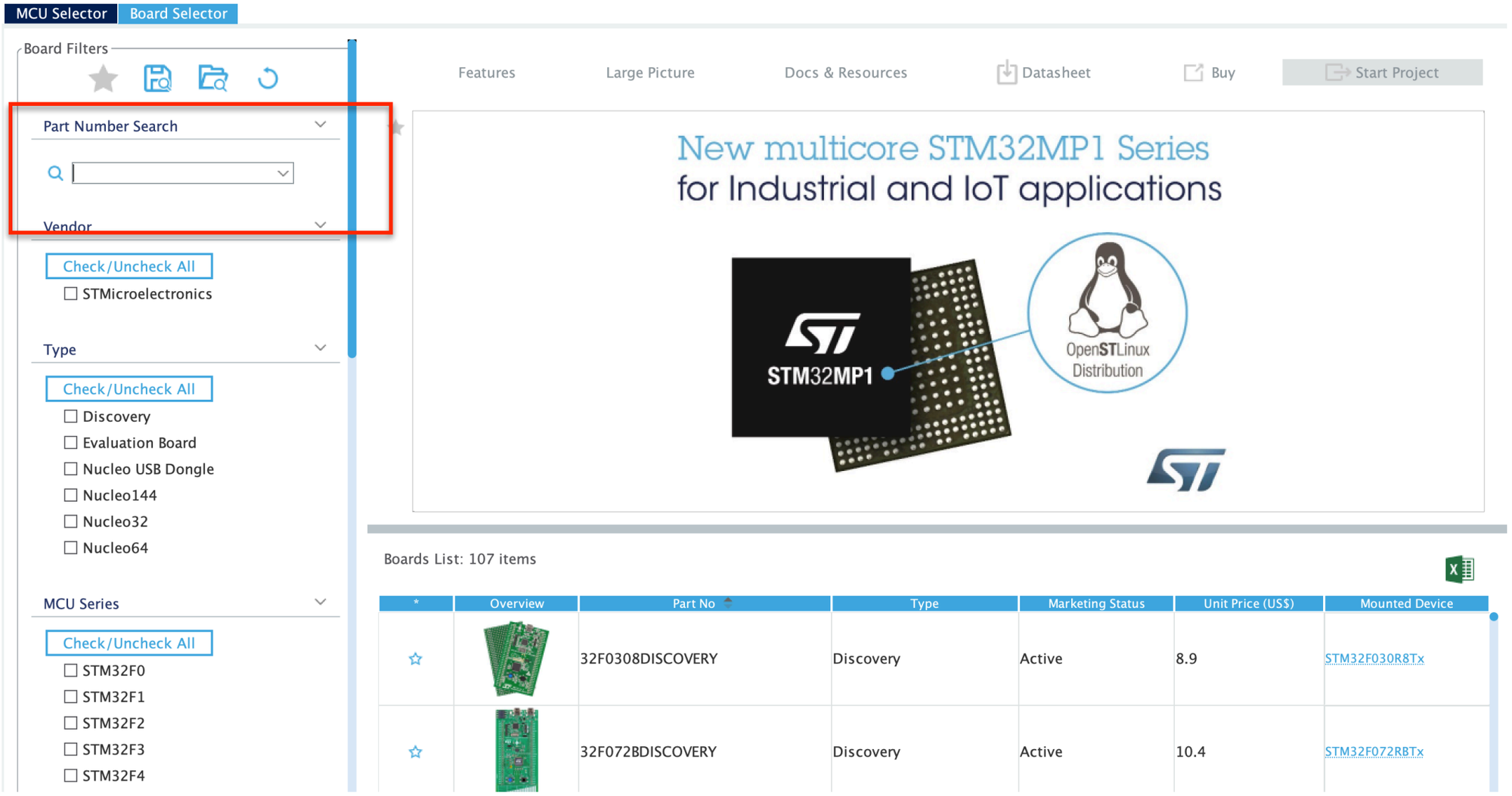


Step: Click on “Access to Board Selector”

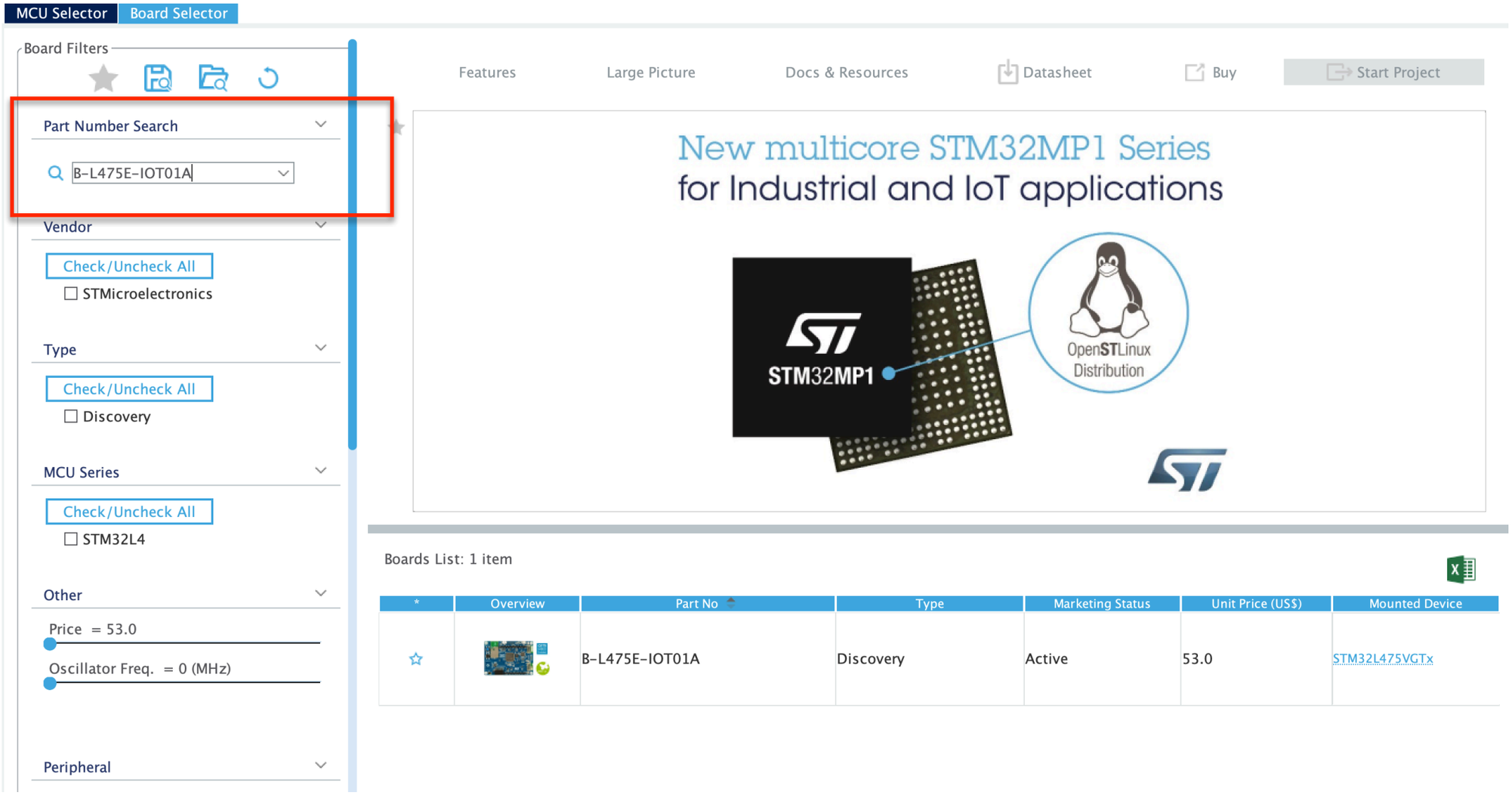


Click “Access to Board Selector”

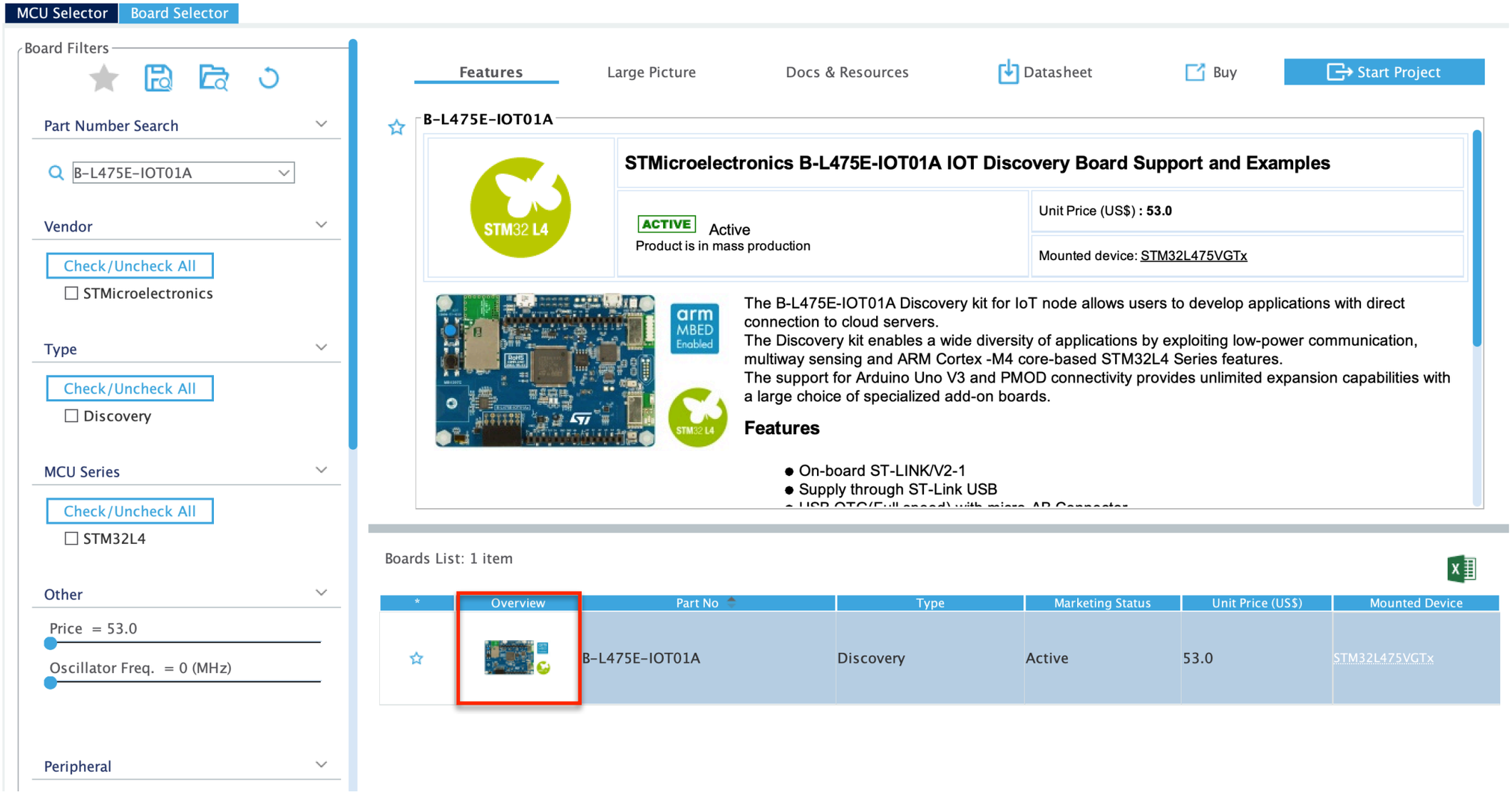
Step: Observe “Part Number Search”



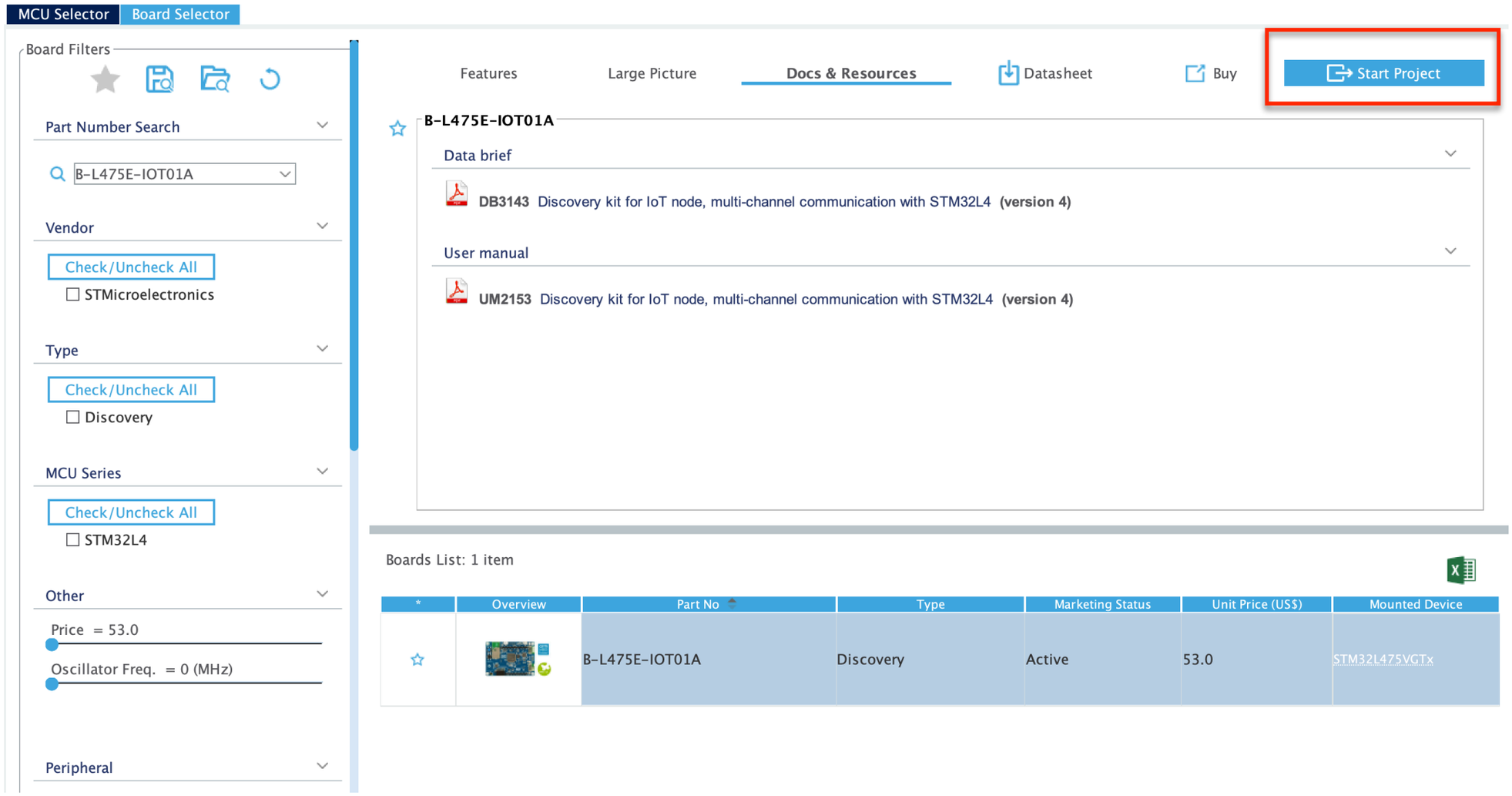
Step: Enter ”B-L475E-IOT01A”



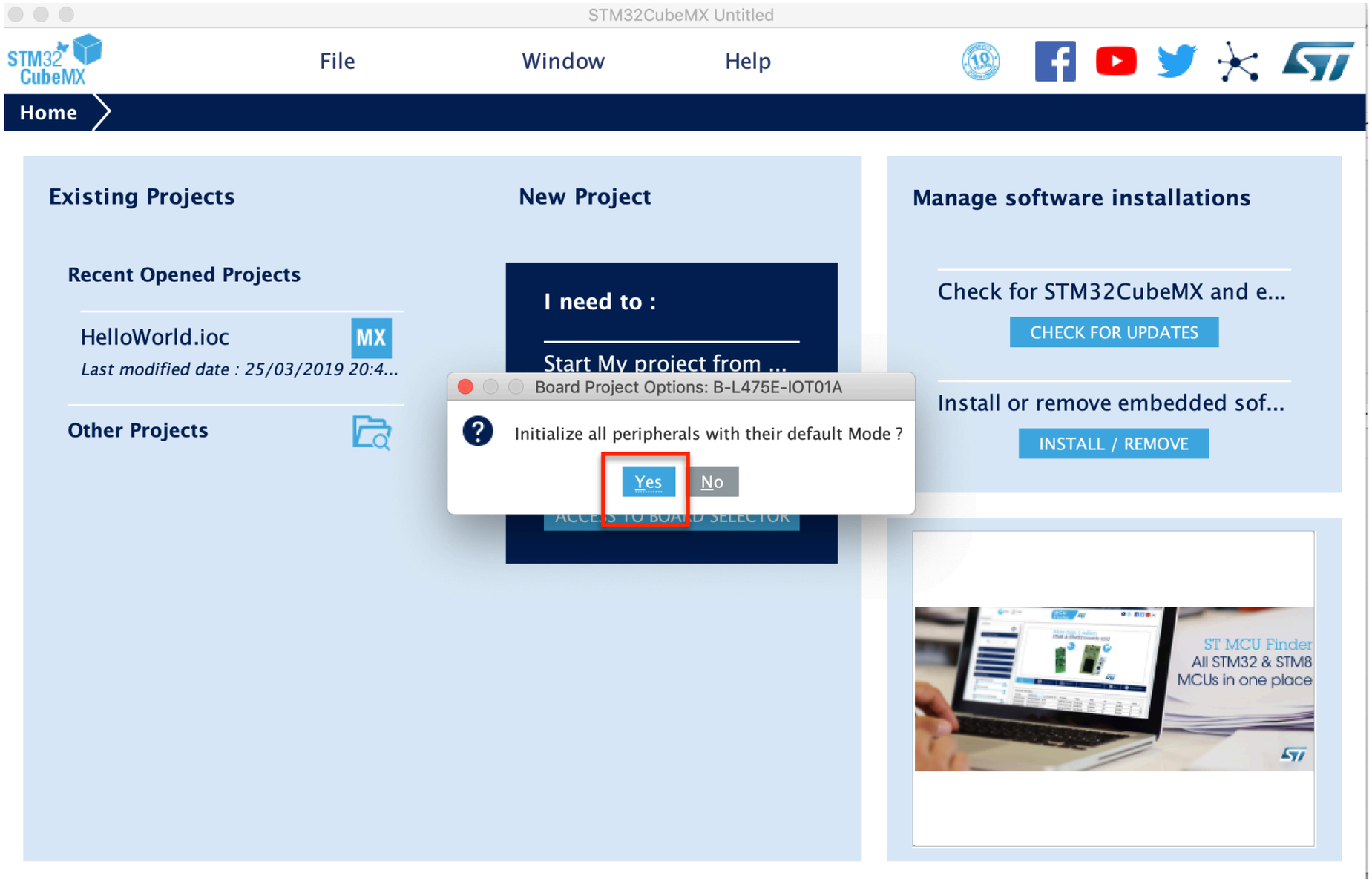
Step: Click on Image, and observe “Features”



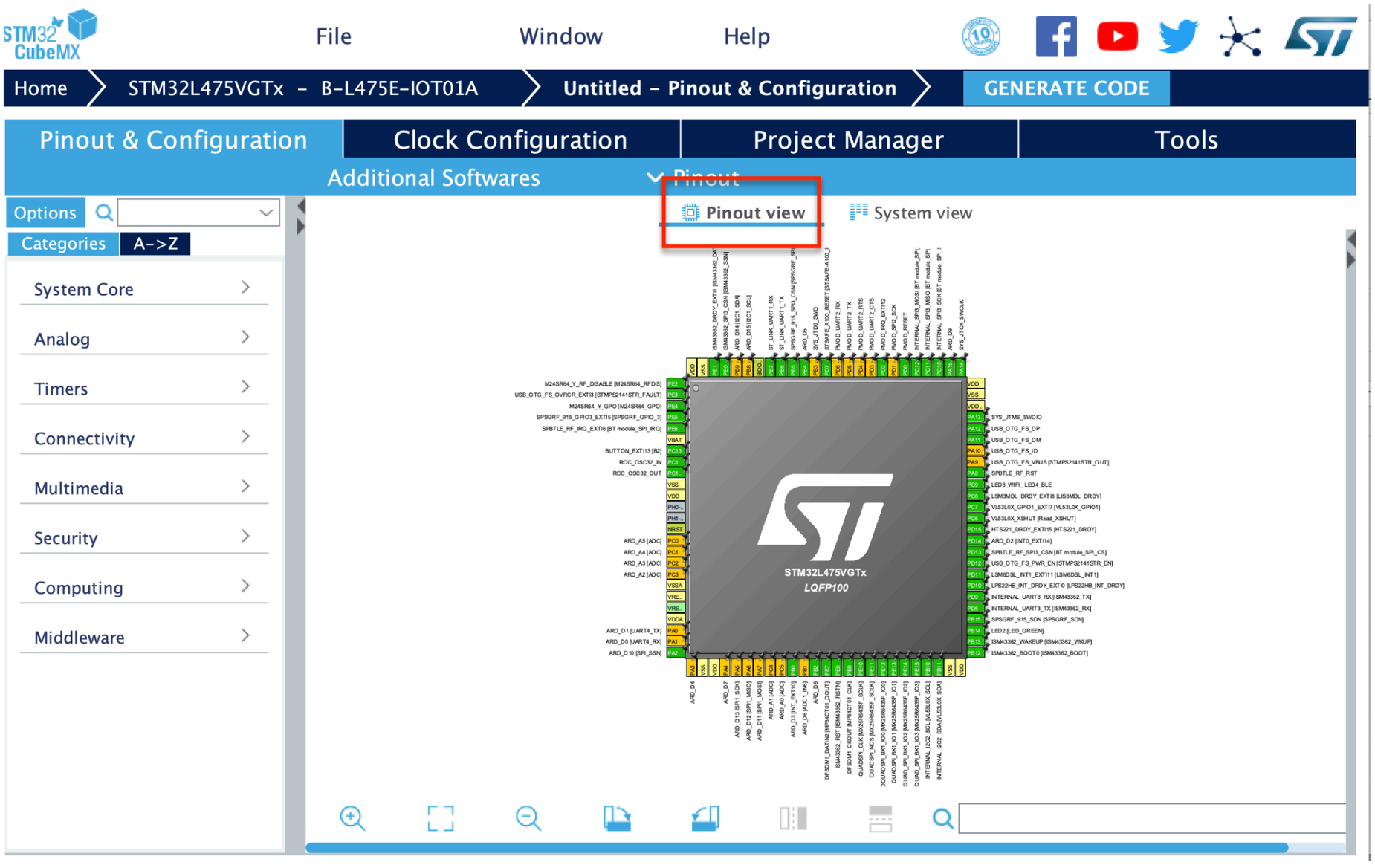
Step: Click on “Start Project”



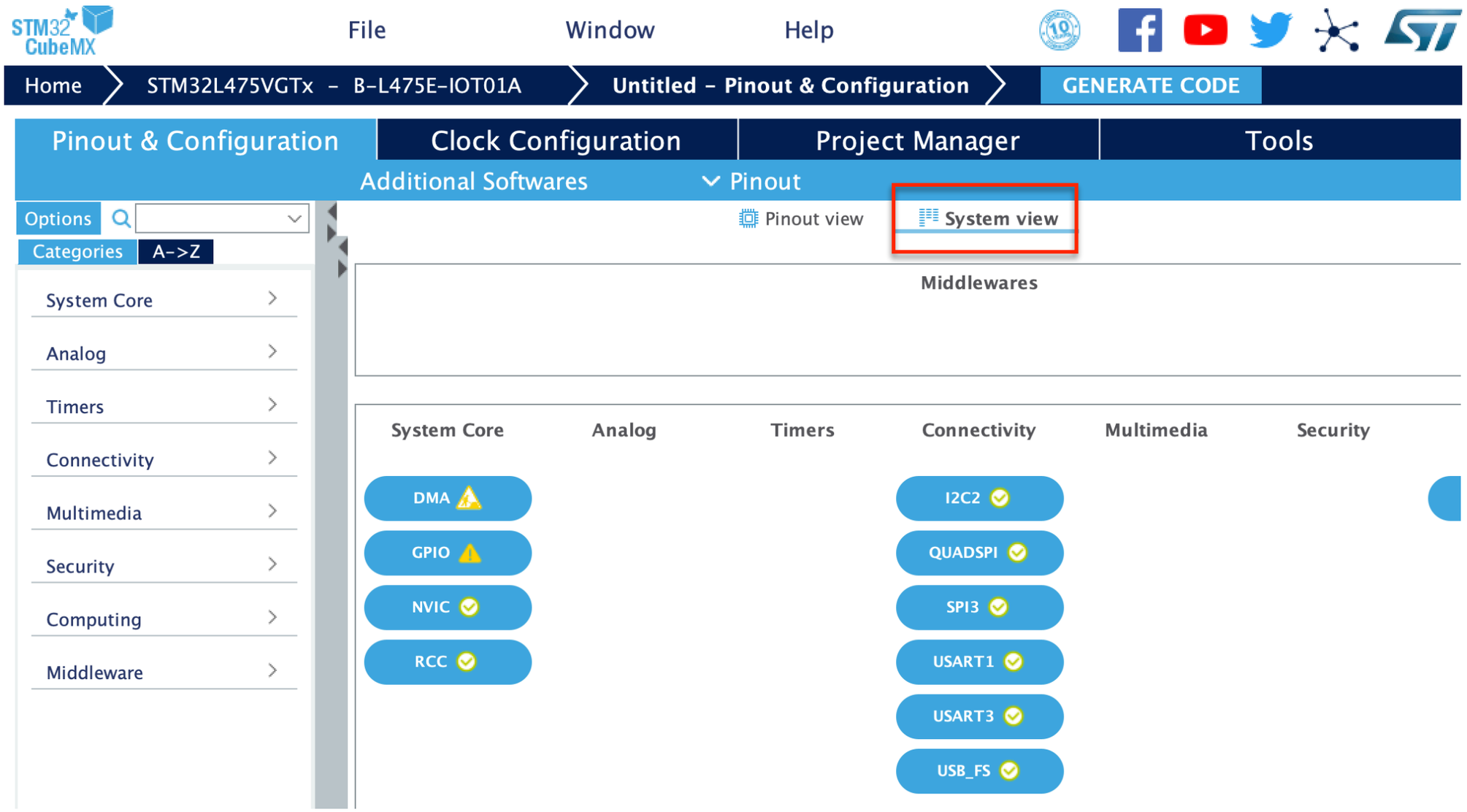
Step: Click on “Yes” (Initialize all …with default mode)



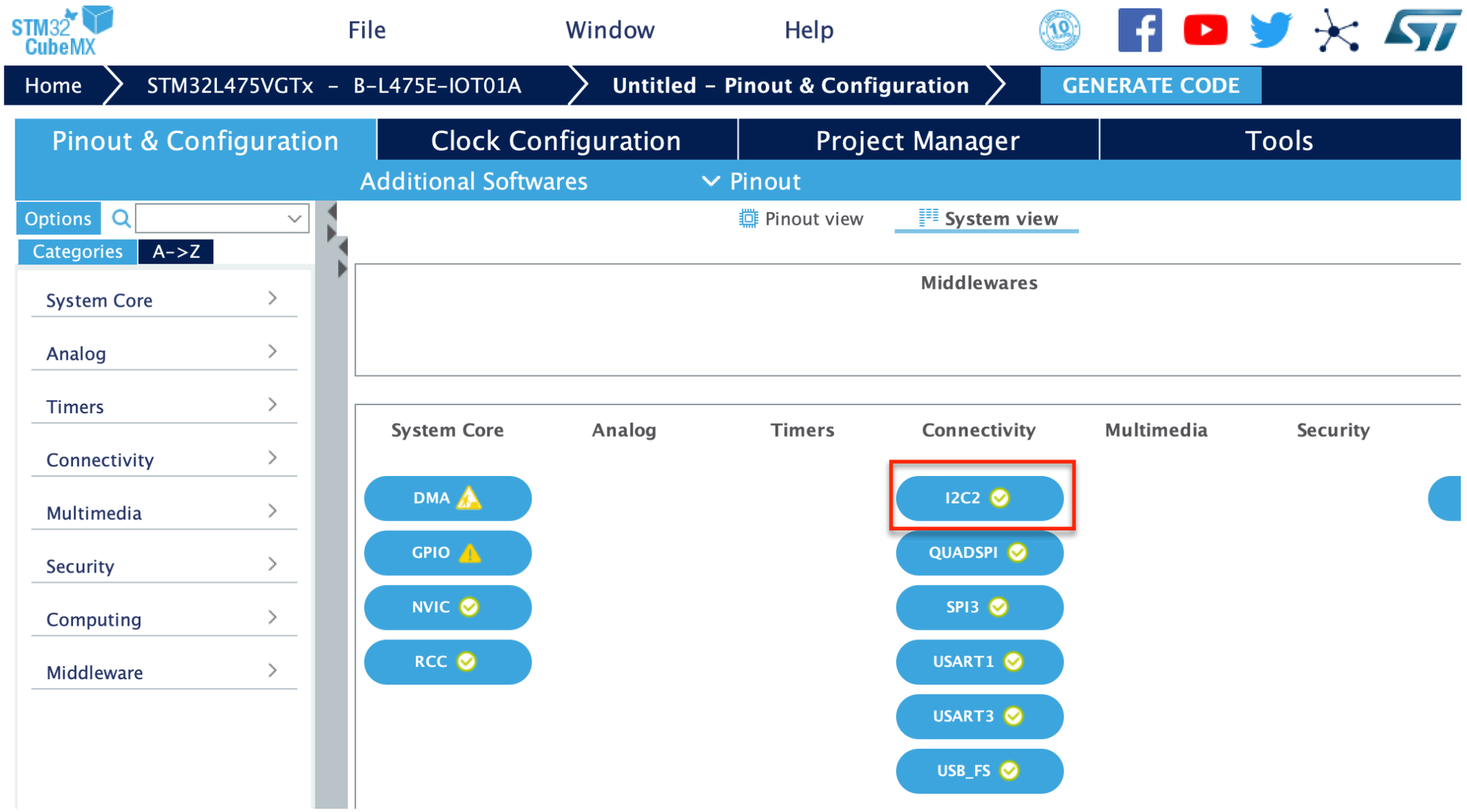
Step: Observe “Pinout View”



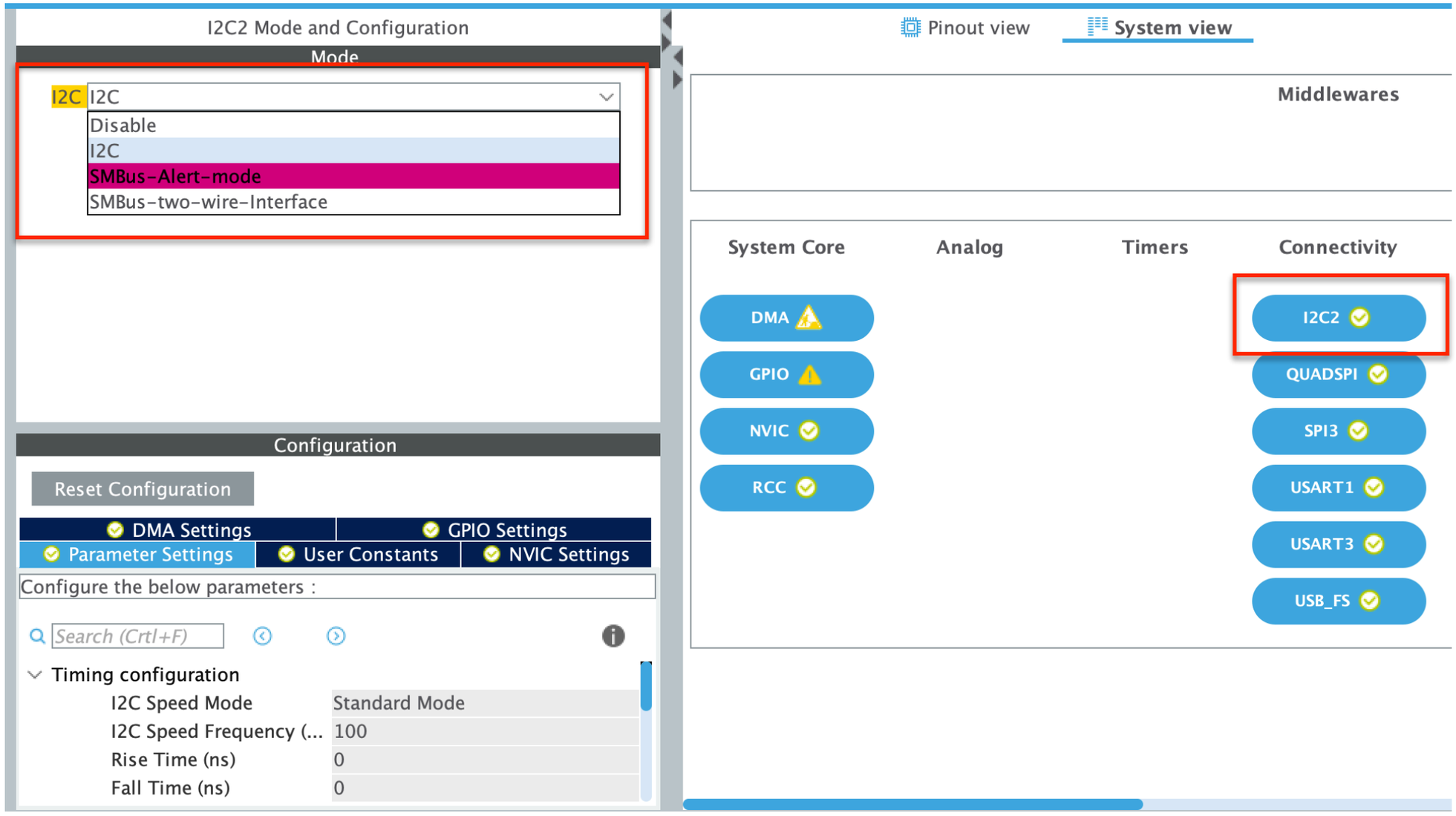
Step: Select “System View”



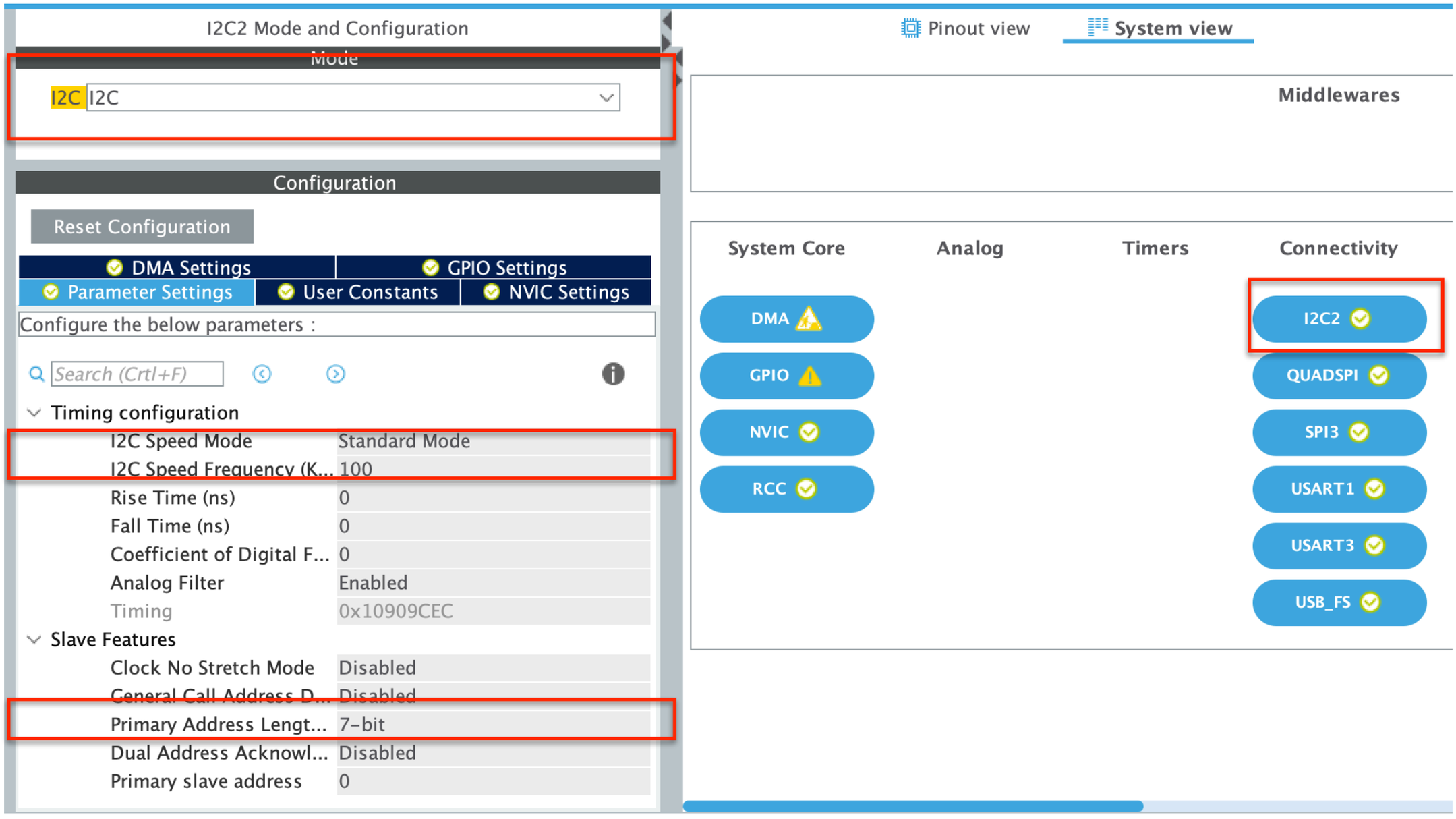
Step: Select “I2C2”



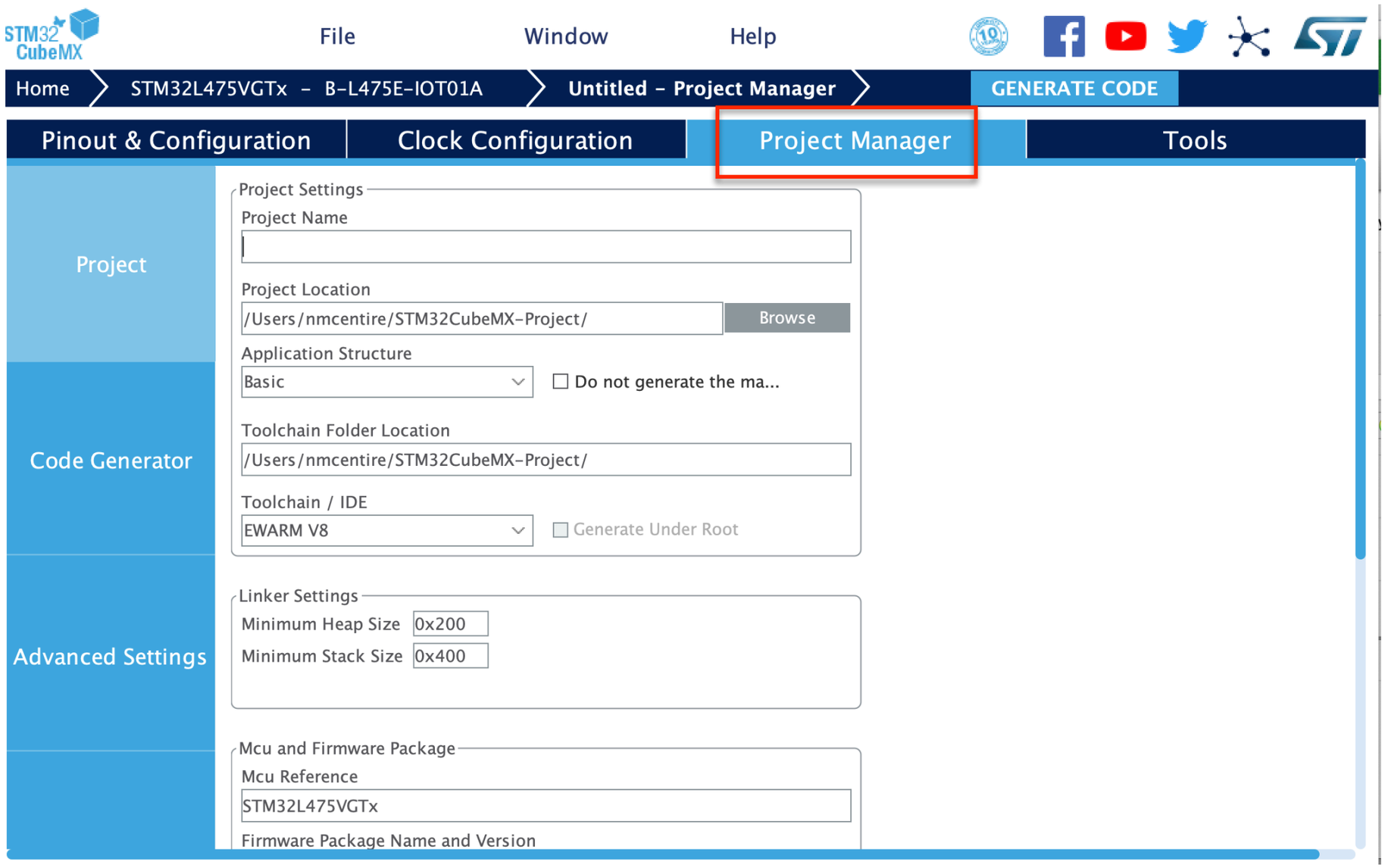
Step: Observe I2C Mode Options



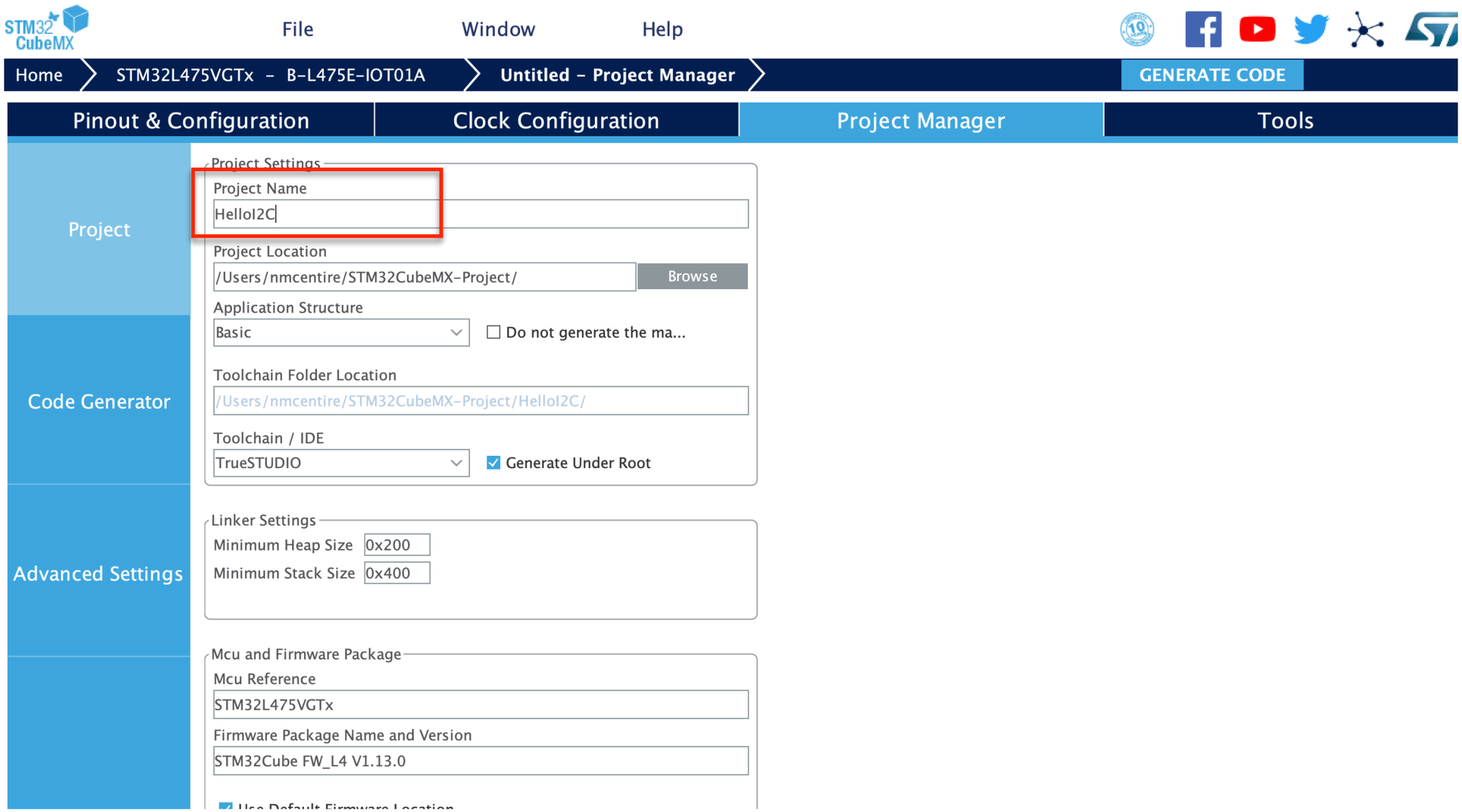
Step: Observe I2C Settings



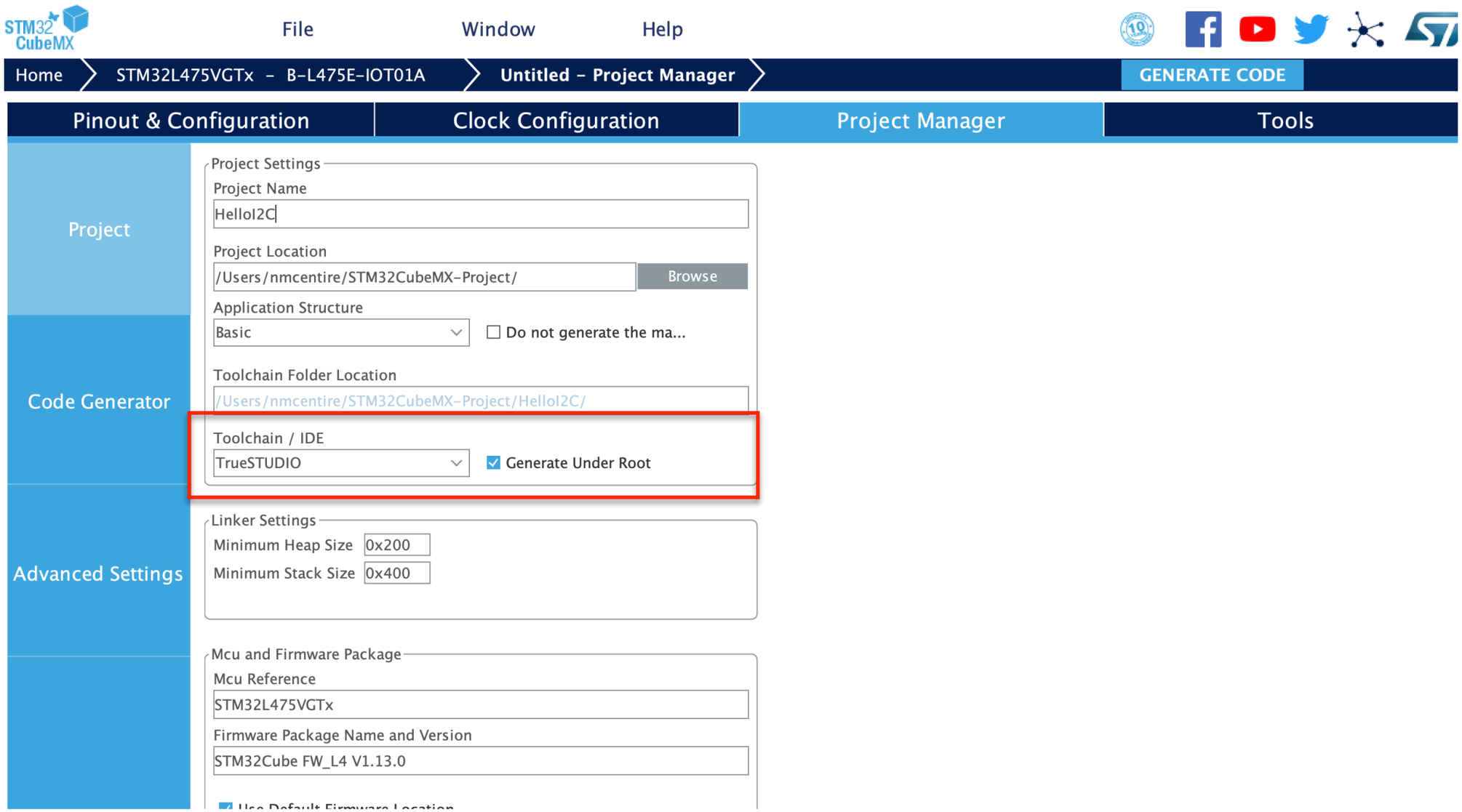
Step: Observe “Project Manager”



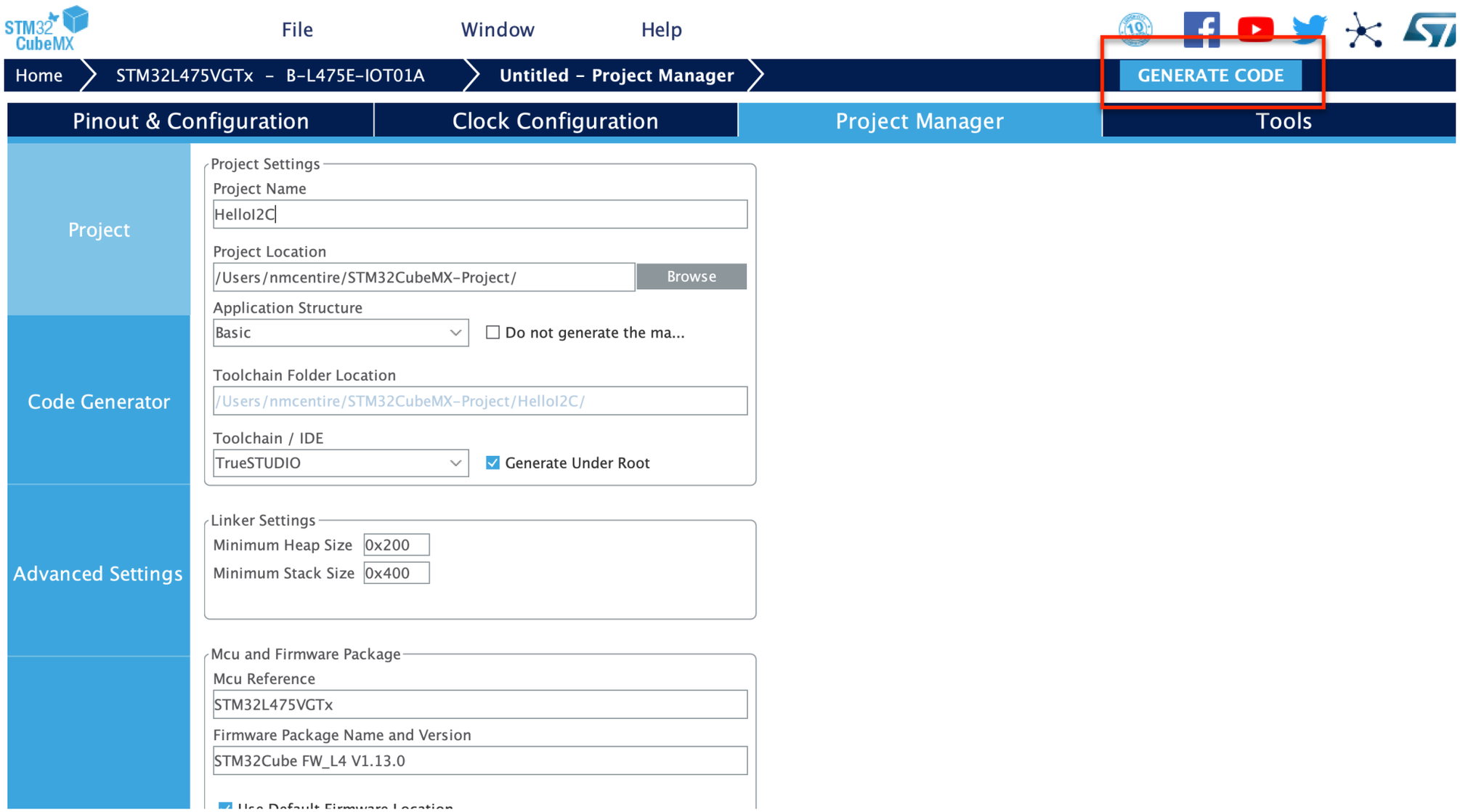
Step: Enter “HelloI2C” for Project Name



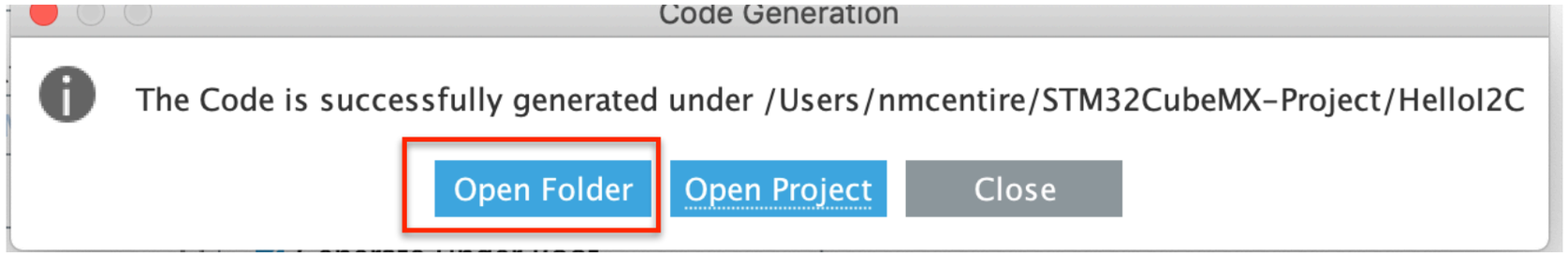
Step: Select “TrueStudio” for Toolchain / IDE



Step: Click on “Generate Code”

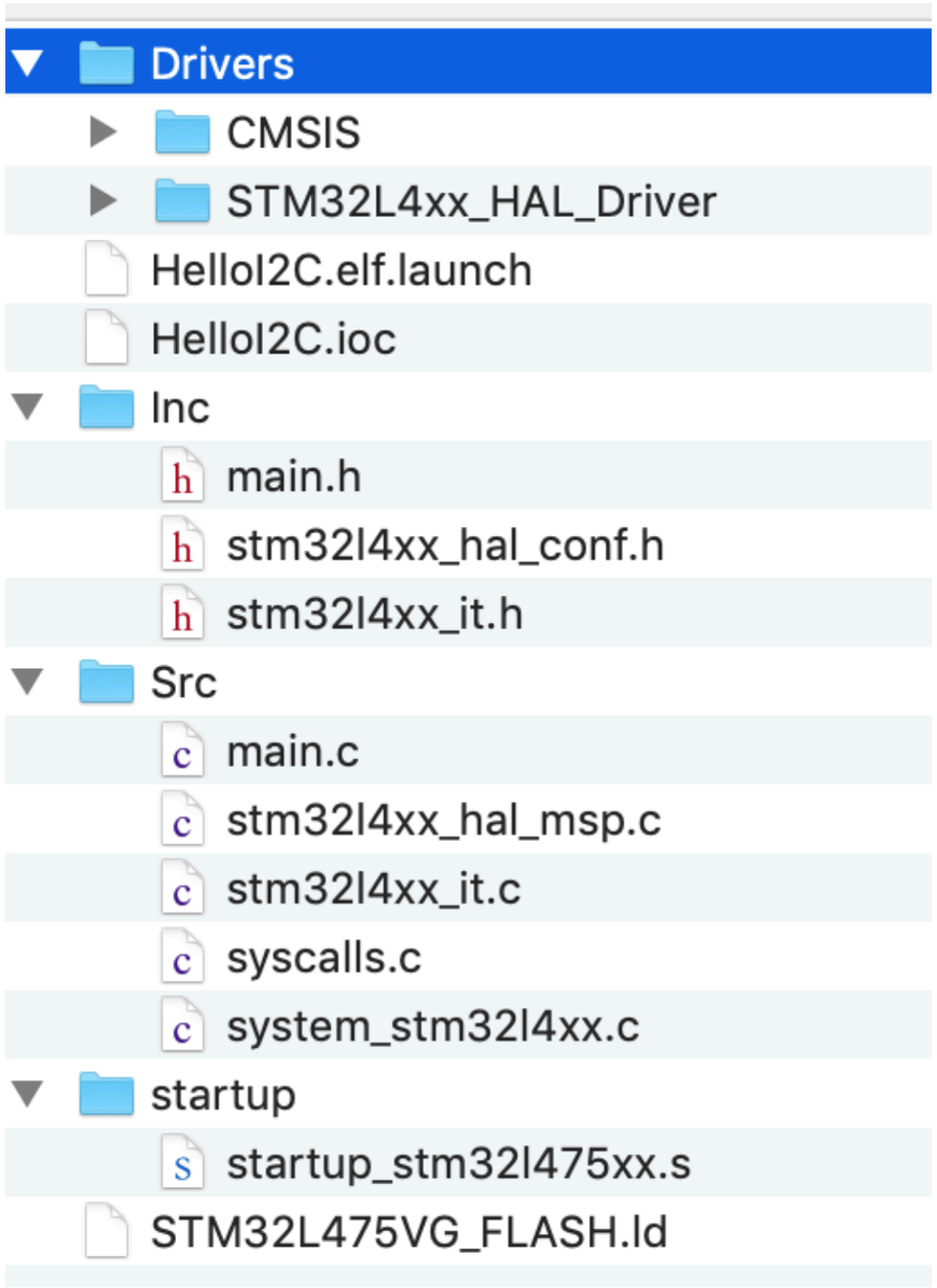


Step: Select “Open Folder”

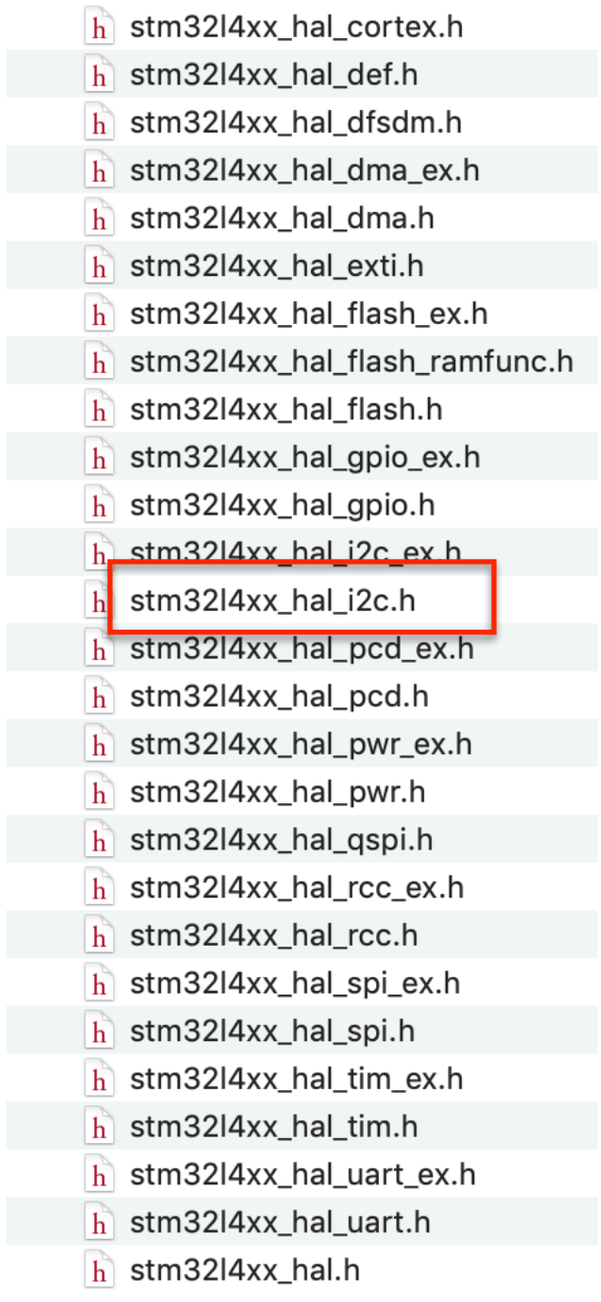


Tour of Generated Project

Step: View Open Folder

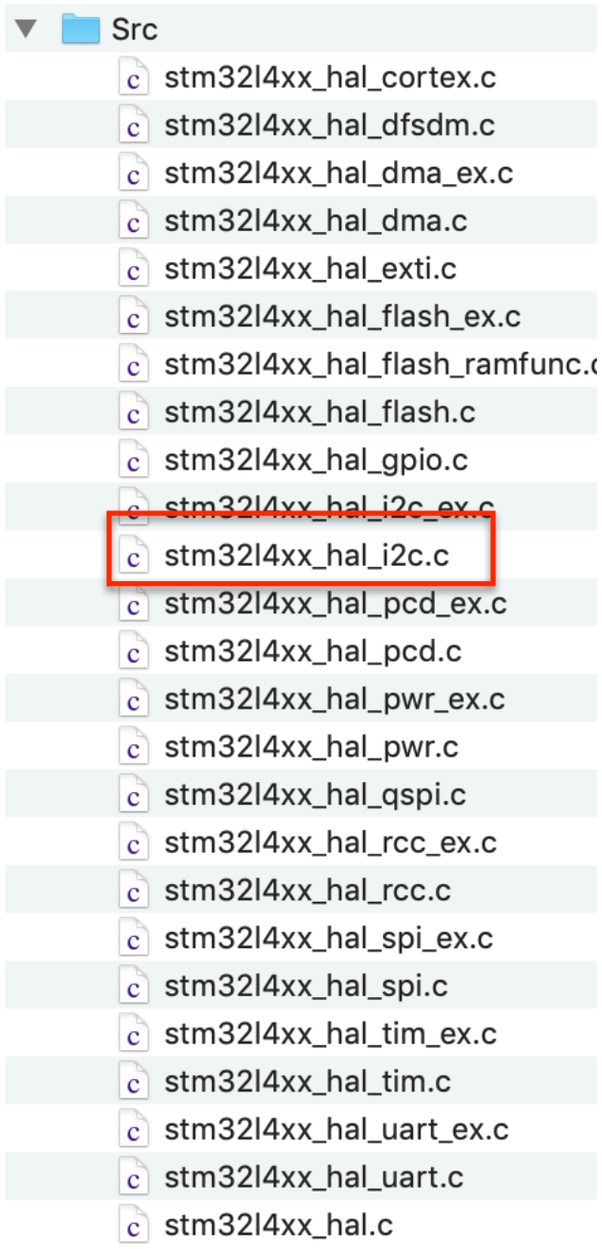


Step: View “Drivers/ STM32L4xx_HAL_Driver/Inc” directory



Observe stm32l4xx_hal_i2c.h

Step: View “Drivers/ STM32L4xx_HAL_Driver/Src” directory



Observe stm32l4xx_hal_i2c.c

Inc/main.h - Part 1

```
128  #define INTERNAL_I2C2_SCL_Pin GPIO_PIN_10
129  #define INTERNAL_I2C2_SCL_GPIO_Port GPIOB
130  #define INTERNAL_I2C2_SDA_Pin GPIO_PIN_11
131  #define INTERNAL_I2C2_SDA_GPIO_Port GPIOB
```

```
157  #define HTS221_DRDY_EXTI15_Pin GPIO_PIN_15
158  #define HTS221_DRDY_EXTI15_GPIO_Port GPIOD
159  #define HTS221_DRDY_EXTI15_EXTI_IRQn EXTI15_10_IRQn
```

Src/main.c - Part 1

```
46
47  I2C_HandleTypeDef hi2c2;
48
```

```
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 */
```

Src/main.c - Part 2

```
176   PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART1|RCC_PERIPHCLK_USART3
177                                       |RCC_PERIPHCLK_I2C2 RCC_PERIPHCLK_DFSDM1
178                                       |RCC_PERIPHCLK_USB;
179   PeriphClkInit.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
180   PeriphClkInit.Usart3ClockSelection = RCC_USART3CLKSOURCE_PCLK1;
181   PeriphClkInit.I2c2ClockSelection = RCC_I2C2CLKSOURCE_PCLK1;
182   PeriphClkInit.Dfsdm1ClockSelection = RCC_DFSDM1CLKSOURCE_PCLK;
183   PeriphClkInit.UsbClockSelection = RCC_USBCLKSOURCE_PLLSAI1;
184   PeriphClkInit.PLLSAI1.PLLSAI1Source = RCC_PLLSOURCE_MSI;
```

Src/main.c - Part 3

```
249  static void MX_I2C2_Init(void)
250  {
251
252    /* USER CODE BEGIN I2C2_Init 0 */
253
254    /* USER CODE END I2C2_Init 0 */
255
256    /* USER CODE BEGIN I2C2_Init 1 */
257
258    /* USER CODE END I2C2_Init 1 */
```

Src/main.c - Part 4

```
259   hi2c2.Instance = I2C2;
260   hi2c2.Init.Timing = 0x10909CEC;
261   hi2c2.Init.OwnAddress1 = 0;
262   hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
263   hi2c2.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
264   hi2c2.Init.OwnAddress2 = 0;
265   hi2c2.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
266   hi2c2.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
267   hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
268   if (HAL_I2C_Init(&hi2c2) != HAL_OK)
269   {
270     Error_Handler();
271   }
```

Base Register Address

Summary so far…

- STM32CubeMX generates I2C Code
- HAL Driver Code
    - Drivers/STM32L4xx_HAL_Driver/Inc
        - stm32l4xx_hal_i2c.h
    - Drivers/STM32L4xx_HAL_Driver/Src
        - stm32l4xx_hal_i2c.c
- Inc/main.h
- Inc/main.c

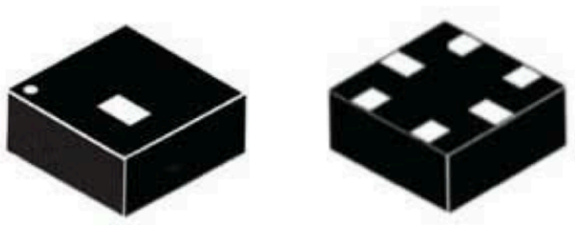
Use TrueStudio to read temperature from HTS221

HTS221
https://www.st.com/resource/en/datasheet/hts221.pdf

HTS221

Capacitive digital sensor for relative humidity and temperature

Datasheet - production data



HLGA-6L
(2 x 2 x 0.9 mm)

Applications

- Air conditioning, heating and ventilation
- Air humidifiers
- Refrigerators
- Wearable devices
- Smart home automation
- Industrial automation
- Respiratory equipment
- Asset and goods tracking

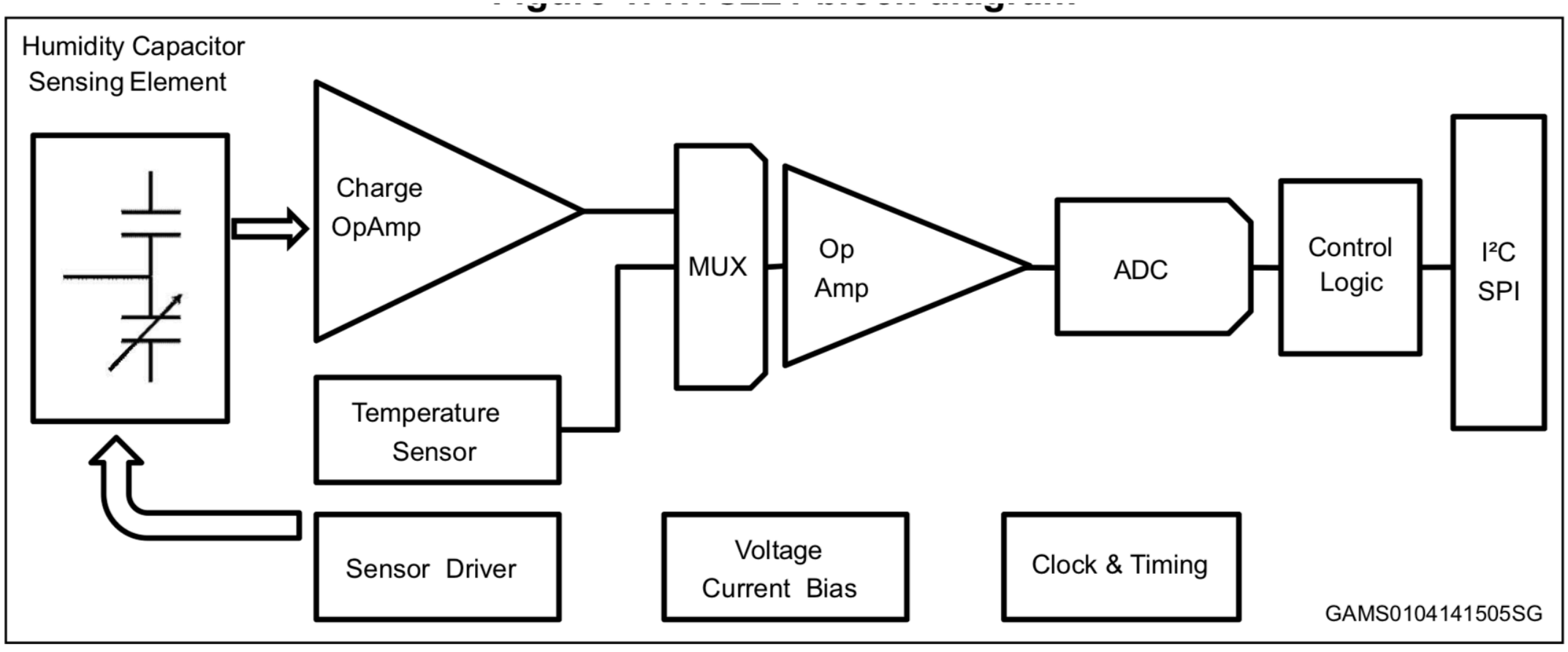
HTS221 - Features

Temperature range [°C]
-40 to +120

Features

- 0 to 100% relative humidity range
- Supply voltage: 1.7 to 3.6 V
- Low power consumption: 2 µA @ 1 Hz ODR
- Selectable ODR from 1 Hz to 12.5 Hz
- High rH sensitivity: 0.004% rH/LSB
- Humidity accuracy: ± 3.5% rH, 20 to +80% rH
- Temperature accuracy: ± 0.5 °C,15 to +40 °C
- Embedded 16-bit ADC
- 16-bit humidity and temperature output data
- SPI and I²C interfaces
- Factory calibrated
- Tiny 2 x 2 x 0.9 mm package
- ECOPACK® compliant

HTS221 Block Diagram



HTS221 Pins

Pin n°	Name	Function
1	V_{DD}	Power supply
2	SCL/SPC	I²C serial clock (SCL) SPI serial port clock (SPC)
3	DRDY	Data Ready output signal
4	SDA/SDI/SDO	I²C serial data (SDA) 3-wire SPI serial data input /output (SDI/SDO)
5	GND	Ground
6	SPI enable	I²C/SPI mode selection (1: SPI idle mode / I²C communication enabled; 0: SPI communication mode / I²C disabled)

Note: DRDY can be used as interrupt

HTS221 I2C Interface

There are two signals associated with the I²C bus: the serial clock line (SCL) and the serial data line (SDA). The latter is a bi-directional line used for sending and receiving the data to/from the interface. Both lines must be connected to V_{DD} through pull-up resistors.

The I²C interface is compliant with fast mode (400 kHz) I²C standards as well as with normal mode.

The 8-bit slave address (SAD) associated to the HTS221 humidity sensor is BEh (write) and BFh (read).

The I²C embedded in the HTS221 behaves like a slave device and the following protocol must be adhered to. After the start condition (ST) a slave address is sent, once a slave acknowledge (SAK) has been returned, an 8-bit sub-address (SUB) will be transmitted: the 7 LSB represents the actual register address while the MSB enables address auto-increment. If the MSB of the SUB field is '1', the SUB (register address) will be automatically increased to allow multiple data read/write.

HTS221 Registers

Name	Type	Register address (hex)	Default (hex)
Reserved		00-0E	Do not modify
WHO_AM_I	R	0F	BC
AV_CONF	R/W	10	1B
Reserved		11-1C	Do not modify
CTRL_REG1	R/W	20	0
CTRL_REG2	R/W	21	0
CTRL_REG3	R/W	22	0
Reserved		23-26	Do not modify
STATUS_REG	R	27	0
HUMIDITY_OUT_L	R	28	Output
HUMIDITY_OUT_H	R	29	Output
TEMP_OUT_L	R	2A	Output
TEMP_OUT_H	R	2B	Output
Reserved		2C-2F	Do not modify
CALIB_0..F	R/W	30-3F	Do not modify

Control Register 1

Control register 1

7	6	5	4	3	2	1	0
PD	Reserved				BDU	ODR1	ODR0

[7]	PD: power-down control (0: power-down mode; 1: active mode)
[6:3]	Reserved
[2]	BDU: block data update (0: continuous update; 1: output registers not updated until MSB and LSB reading)
[1:0]	ODR1, ODR0: output data rate selection (see table 17)

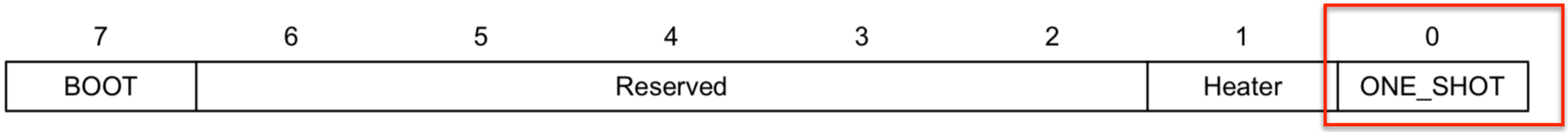
The **PD** bit is used to turn on the device. The device is in power-down mode when PD = ‘0’ (default value after boot). The device is active when PD is set to ‘1’.

The **BDU** bit is used to inhibit the output register update between the reading of the upper and lower register parts. In default mode (BDU = ‘0’), the lower and upper register parts are updated continuously. If it is not certain whether the read will be faster than output data rate, it is recommended to set the BDU bit to ‘1’. In this way, after the reading of the lower (upper) register part, the content of that output register is not updated until the upper (lower) part is read also.

ODR - Output Data Rate

ODR1	ODR0	Humidity (Hz)	Temperature (Hz)
0	0	One-shot	
0	1	1 Hz	1 Hz
1	0	7 Hz	7 Hz
1	1	12.5 Hz	12.5 Hz

Control Register 2



[7]	BOOT: Reboot memory content (0: normal mode; 1: reboot memory content)
[6:2]	Reserved
[1]	Heater (0: heater disable; 1: heater enable)
[0]	One-shot enable (0: waiting for start of conversion; 1: start for a new dataset)

Control Register 3

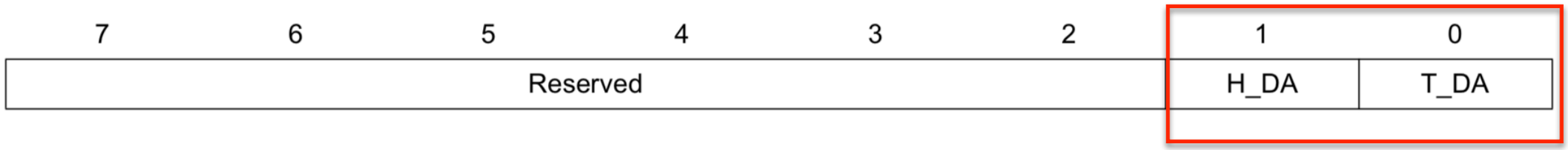
Control register 3

7	6	5	4	3	2	1	0
DRDY_H_L	PP_OD	Reserved			DRDY	Reserved	

Control register for data ready output signal

[7]	DRDY_H_L: Data Ready output signal active high, low (0: active high - default;1: active low)
[6]	PP_OD: Push-pull / Open Drain selection on pin 3 (DRDY) (0: push-pull - default; 1: open drain)
[5:3]	Reserved
[2]	DRDY_EN: Data Ready enable (0: Data Ready disabled - default;1: Data Ready signal available on pin 3)
[1:0]	Reserved

Status Register



Status register; the content of this register is updated every one-shot reading, and after completion of every ODR cycle, regardless of the BDU value in CTRL_REG1.

[7:2]	Reserved
[1]	H_DA: Humidity data available. (0: new data for humidity is not yet available; 1: new data for humidity is available)
[0]	T_DA: Temperature data available. (0: new data for temperature is not yet available; 1: new data for temperature is available)
	H_DA is set to 1 whenever a new humidity sample is available. H_DA is cleared anytime HUMIDITY_OUT_H (29h) register is read.
	T_DA is set to 1 whenever a new temperature sample is available. T_DA is cleared anytime TEMP_OUT_H (2Bh) register is read.

Humidity Out Register

HUMIDITY_OUT_L (28h)

Relative humidity data (LSB)

7	6	5	4	3	2	1	0
HOUT7	HOUT6	HOUT5	HOUT4	HOUT3	HOUT2	HOUT1	HOUT0

Humidity data (see HUMIDITY_OUT_H)

[7:0]	HOUT7 - HOUT0: Humidity data LSB

HUMIDITY_OUT_H (29h)

Relative humidity data (MSB)

15	14	13	12	11	10	9	8
HOUT15	HOUT14	HOUT13	HOUT12	HOUT11	HOUT10	HOUT9	HOUT8

Humidity data are expressed as HUMIDITY_OUT_H & HUMIDITY_OUT_L in 2's complement. Values exceeding the operating humidity range (see *Table 3*) must be clipped by SW.

[7:0]	HOUT15 - HOUT8: Humidity data MSB

Temperature Out Register

TEMP_OUT_L (2Ah)

Temperature data (LSB)

7	6	5	4	3	2	1	0
TOUT7	TOUT6	TOUT5	TOUT4	TOUT3	TOUT2	TOUT1	TOUT0

[7:0]	TOUT7 - TOUT0: Temperature data LSB (see TEMPERATURE_OUT_H)

TEMP_OUT_H (2Bh)

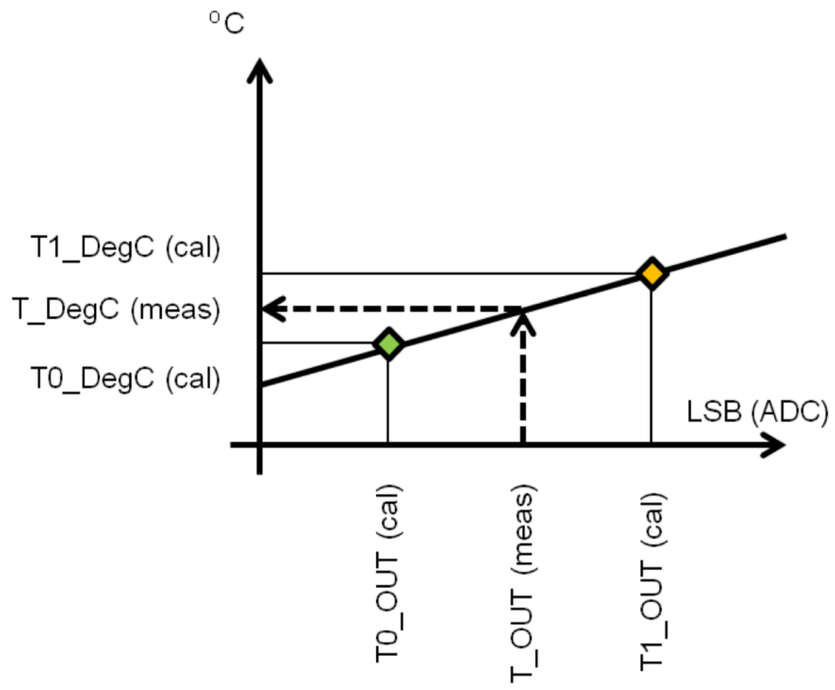
Temperature data (MSB)

15	14	13	12	11	10	9	8
TOUT15	TOUT14	TOUT13	TOUT12	TOUT11	TOUT10	TOUT9	TOUT8

[15:8]	TOUT15 - TOUT8: Temperature data MSB.

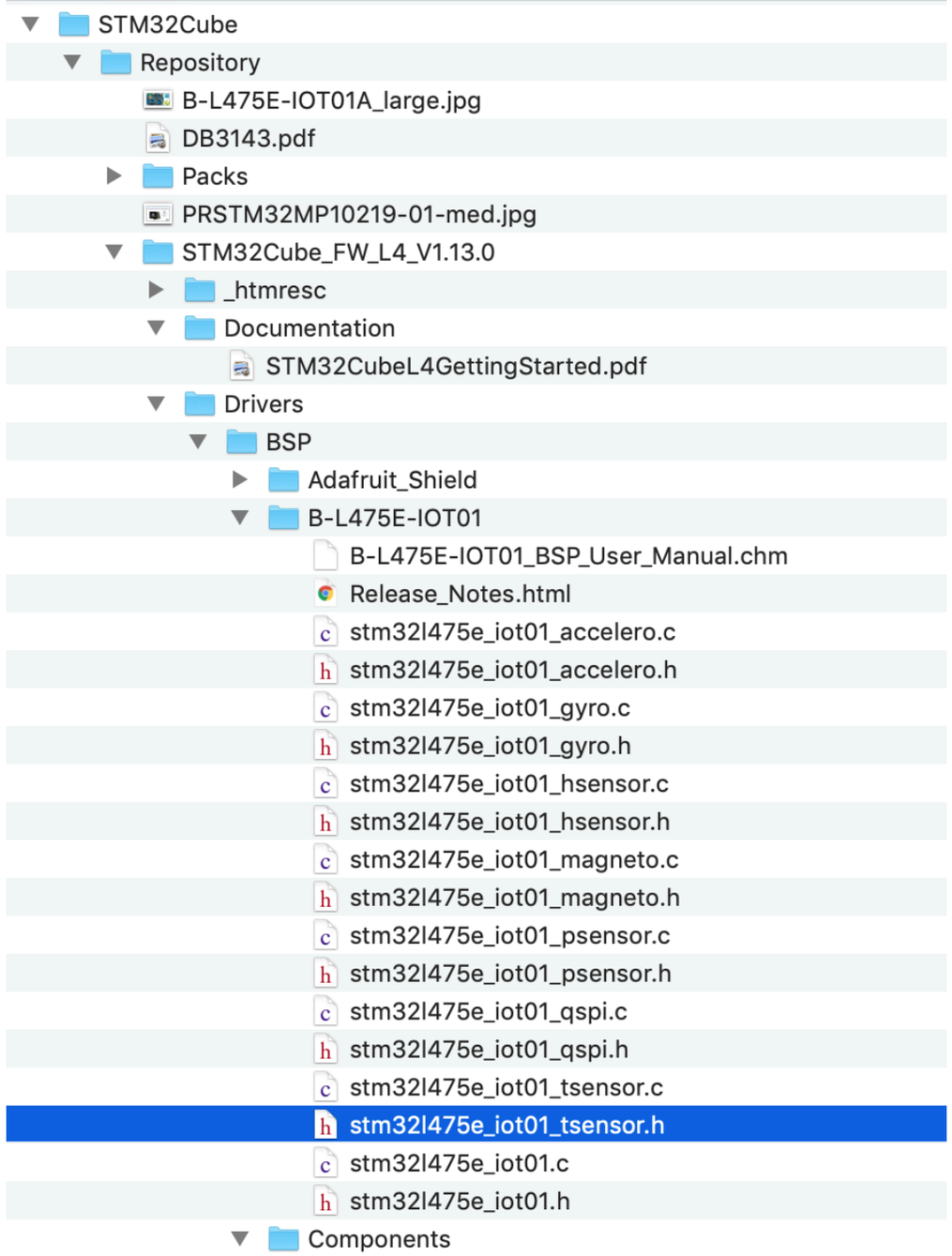
Humidity and Temperature Data Conversion

The Registers in 30h..3Fh address range contain calibration coefficients. Every sensor module has individual coefficients. Before the first calculation of temperature and humidity, the master reads out the calibration coefficients.



Given the details of the HTS221, a BSP interface would be useful…

tsensor.h



tsensor.h - Part 1

```
54  /* Includes ------------------------------------------------------
55  #include "stm32l475e_iot01.h"
56  #ifdef USE_LPS22HB_TEMP
57  #include "../Components/lps22hb/lps22hb.h"
58  #else /* USE_HTS221_TEMP */
59  #include "../Components/hts221/hts221.h"
60  #endif
61
```

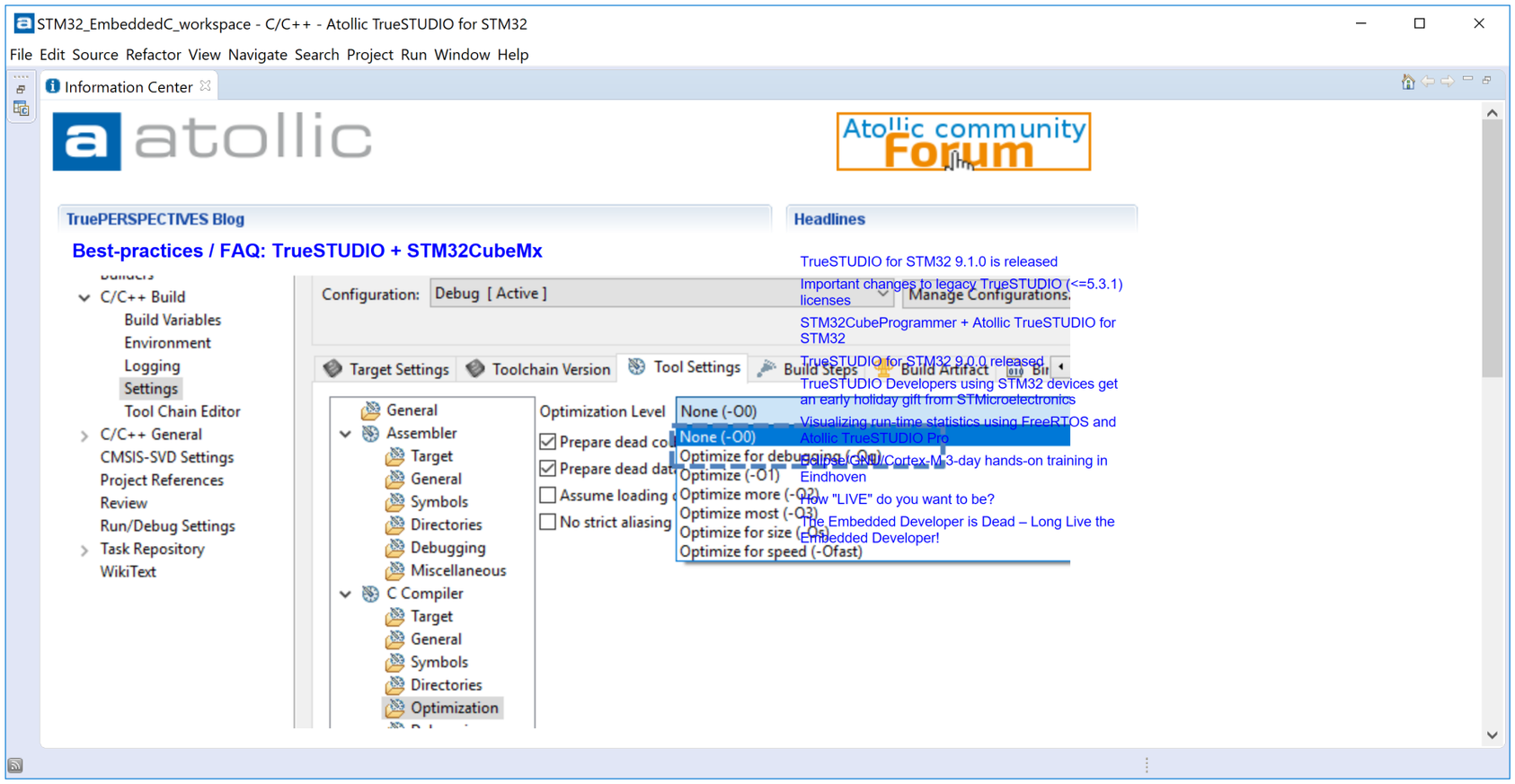
tsensor.h - Part 2

```
81  typedef enum
82  {
83    TSENSOR_OK = 0,
84    TSENSOR_ERROR
85  }TSENSOR_Status_TypDef;
86
```

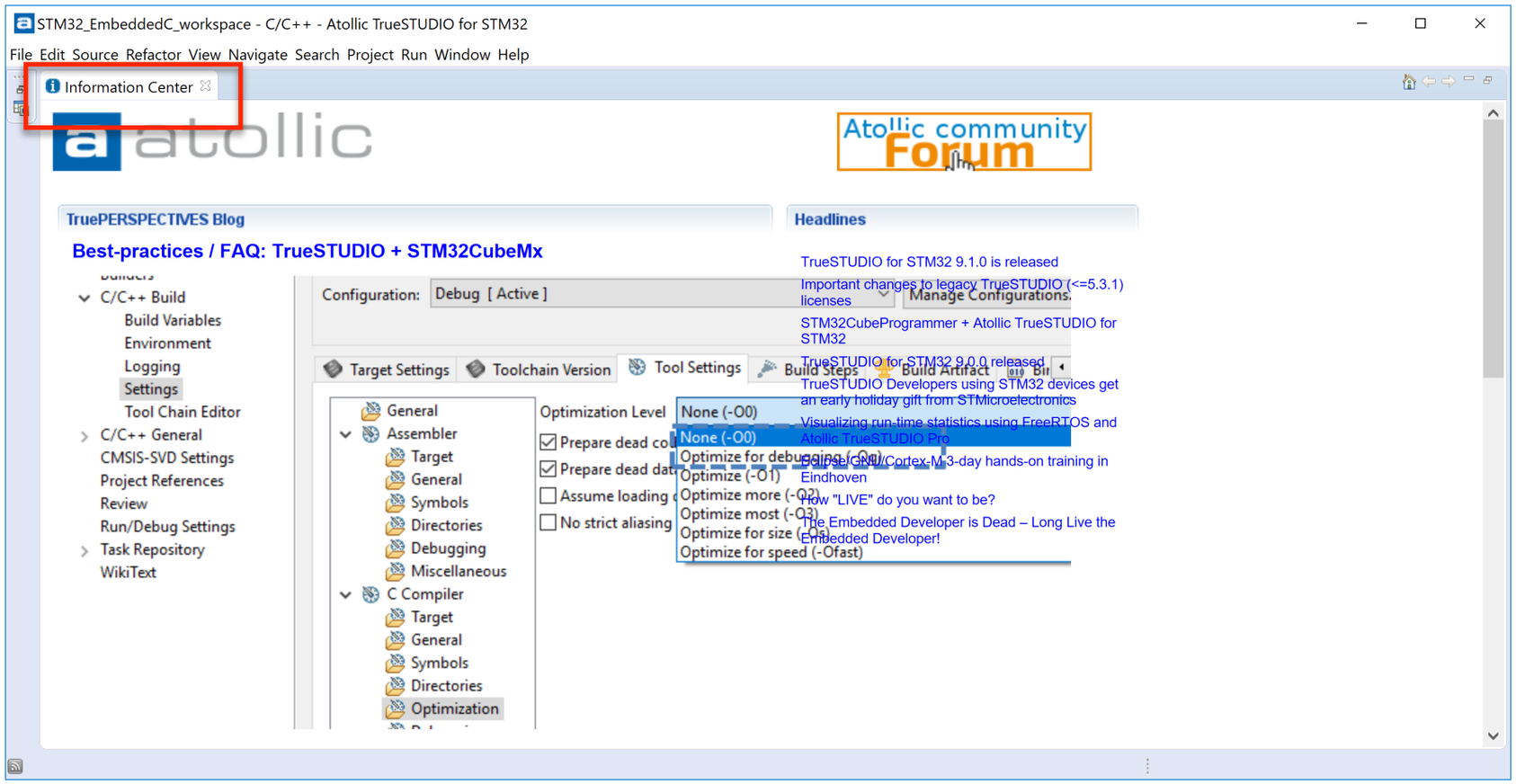
tsensor.h - Part 3

```
98   /* Sensor Configuration Functions */
99   uint32_t BSP_TSENSOR_Init(void);
100  float BSP_TSENSOR_ReadTemp(void);
```

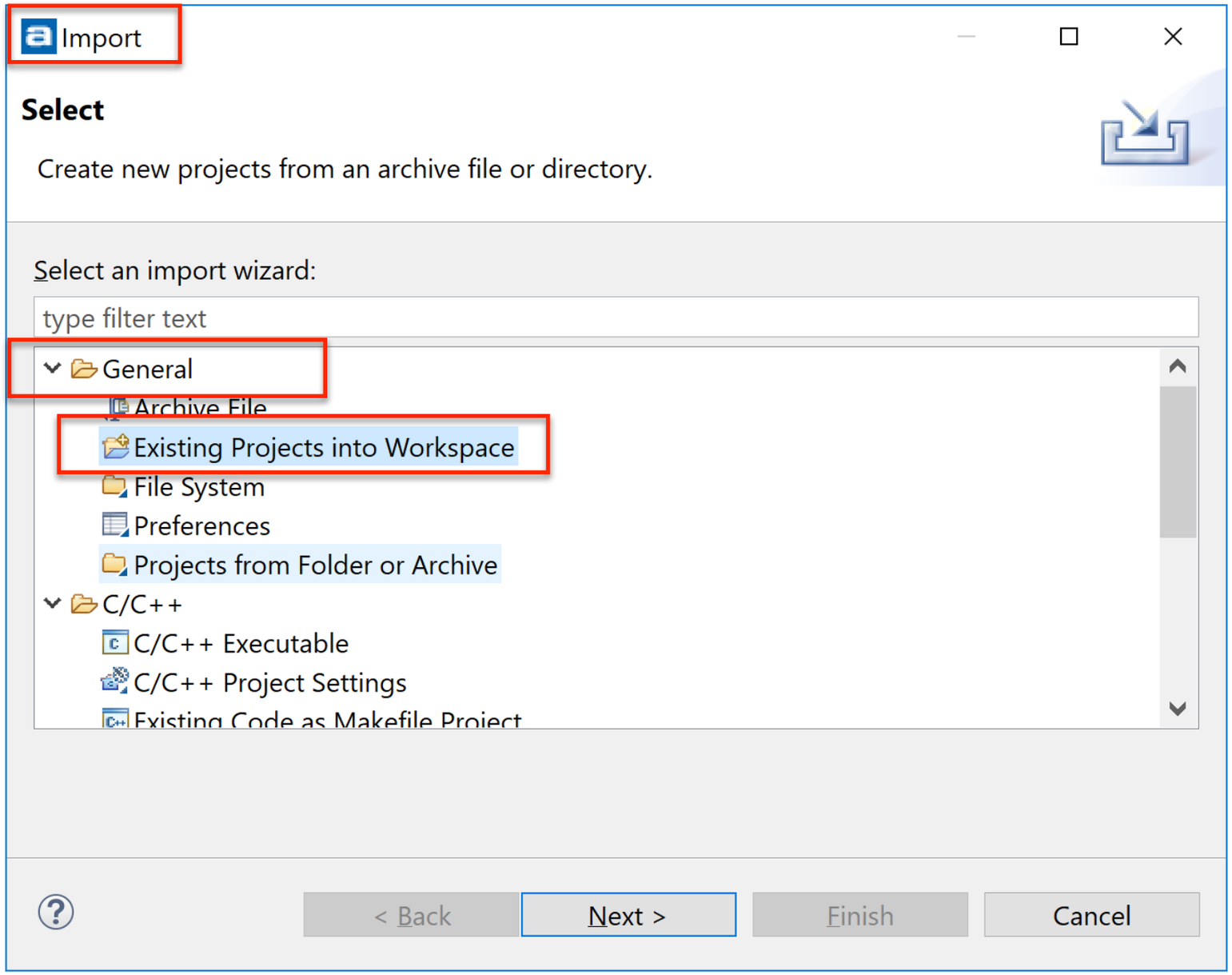
Step: Startup TrueStudio



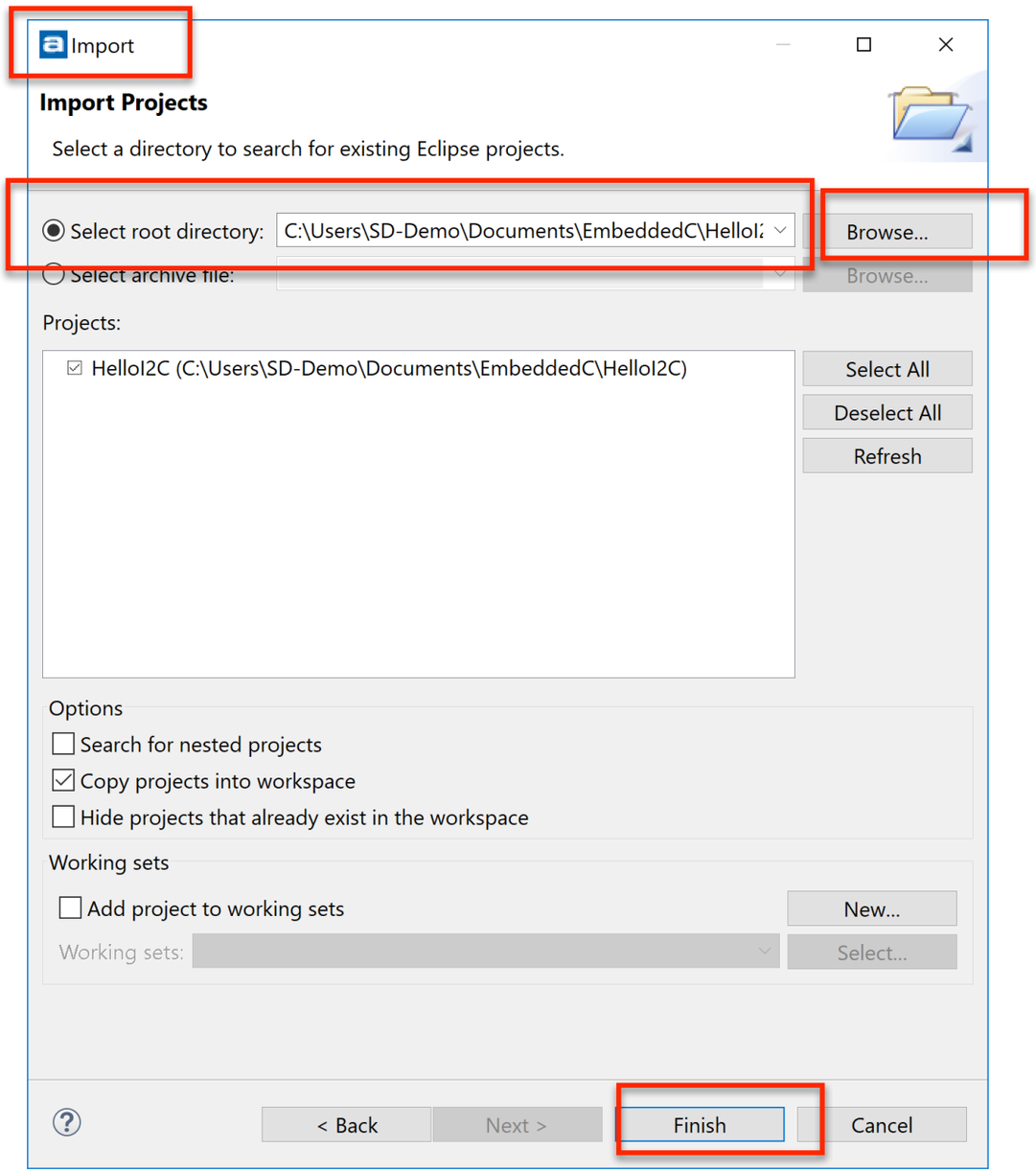
Step: Click to Close “Information Center”



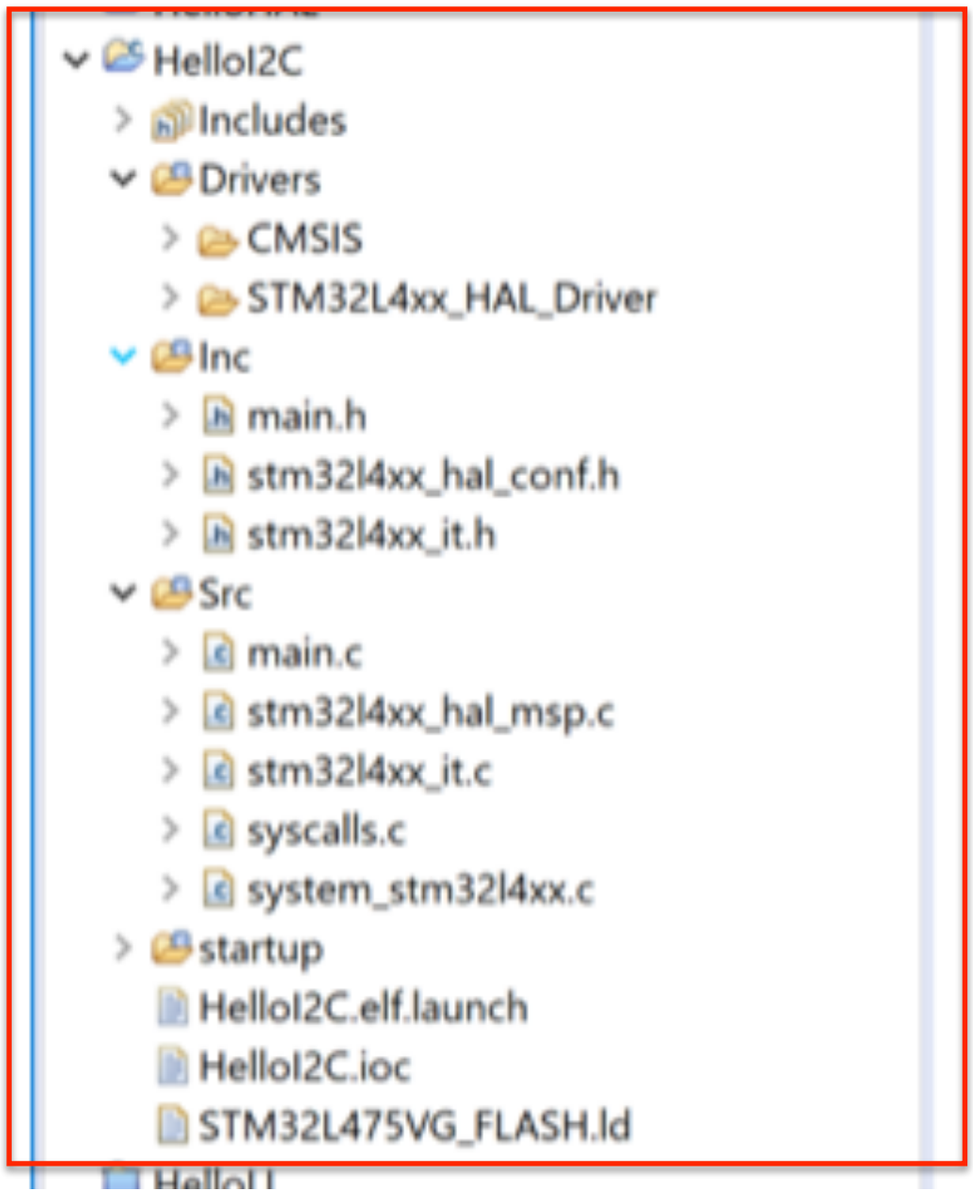
Step: File, Import, General, Existing Projects into Workspace



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

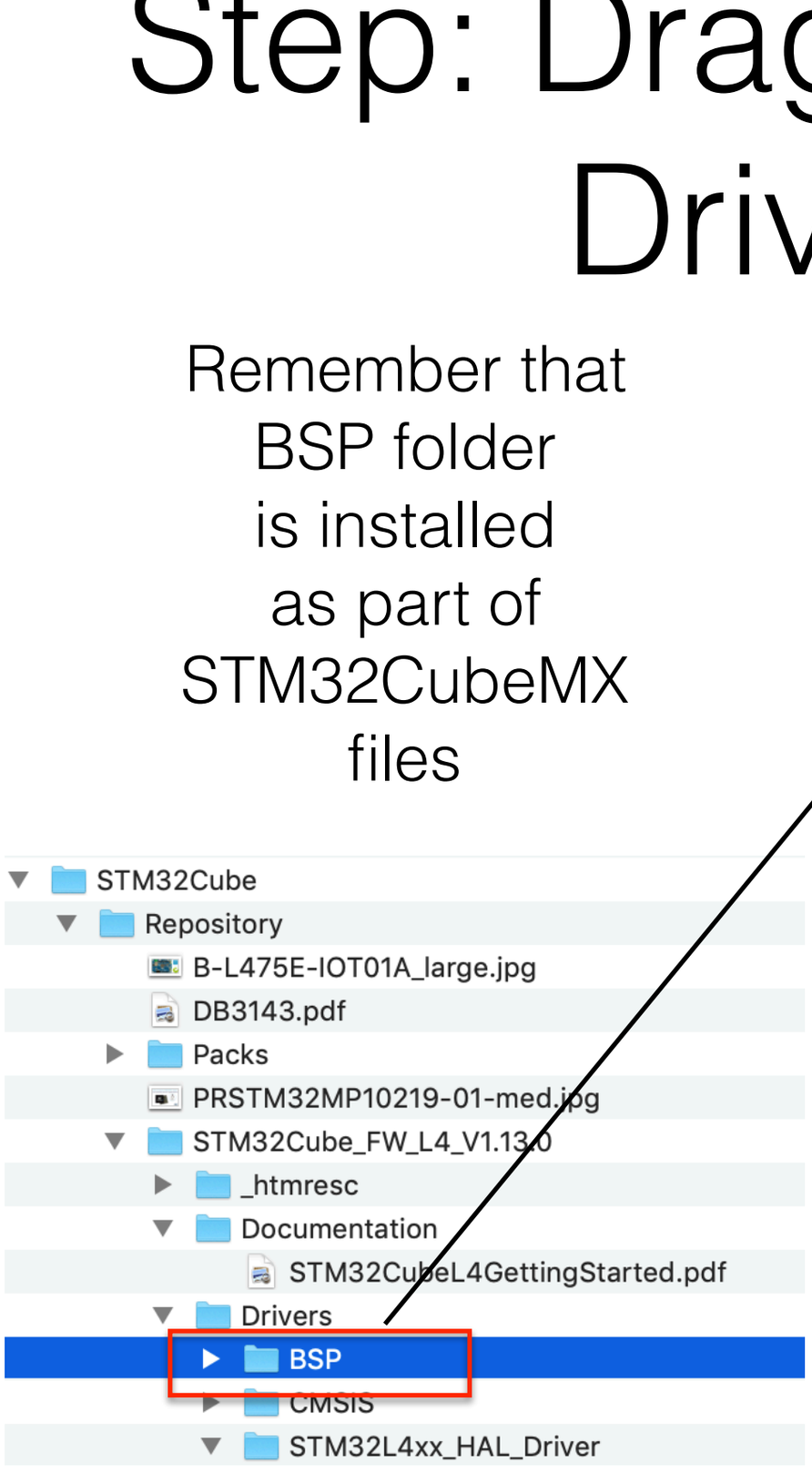


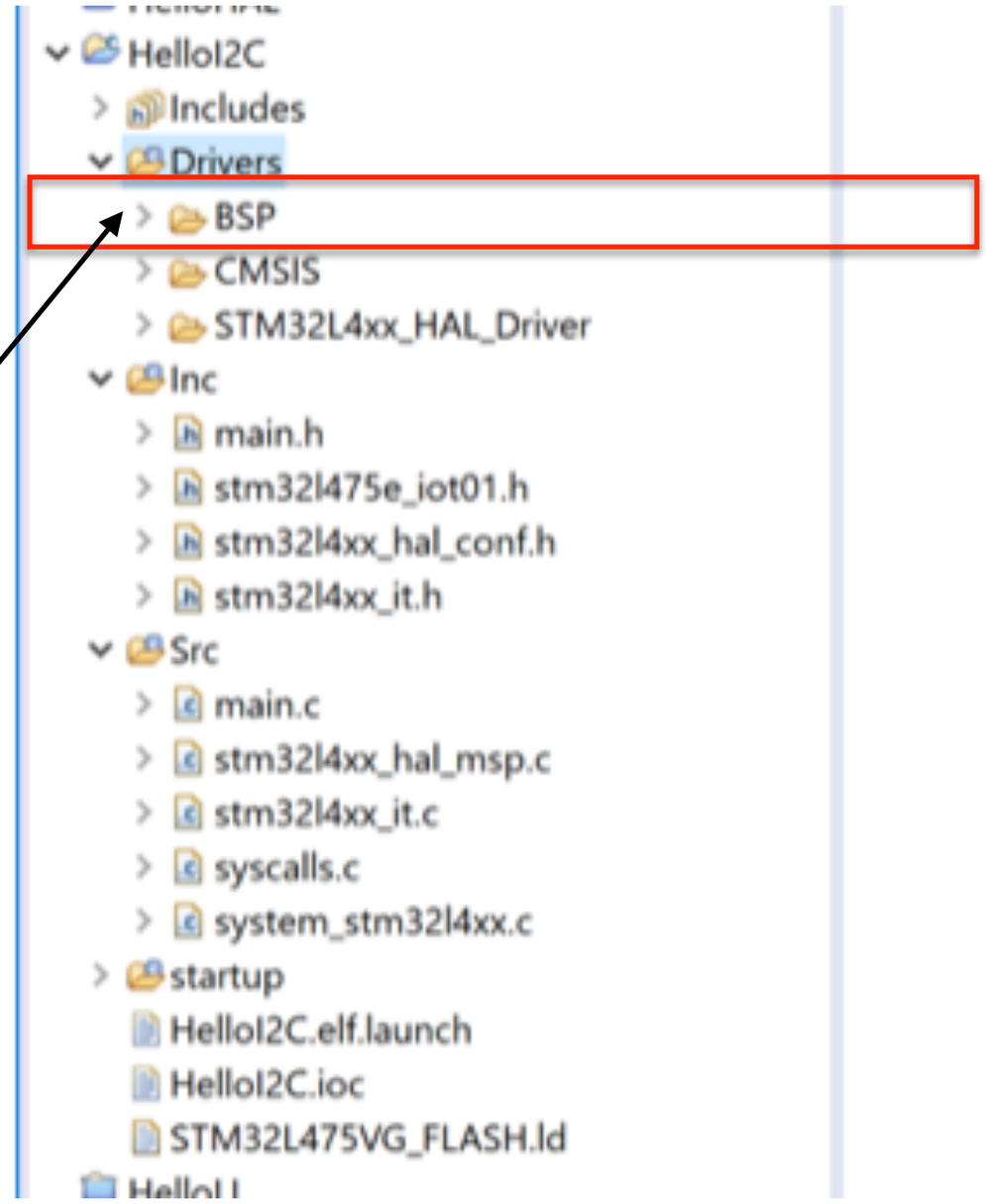
Step: Observe Results



Step: Drag BSP folder into Drivers folder

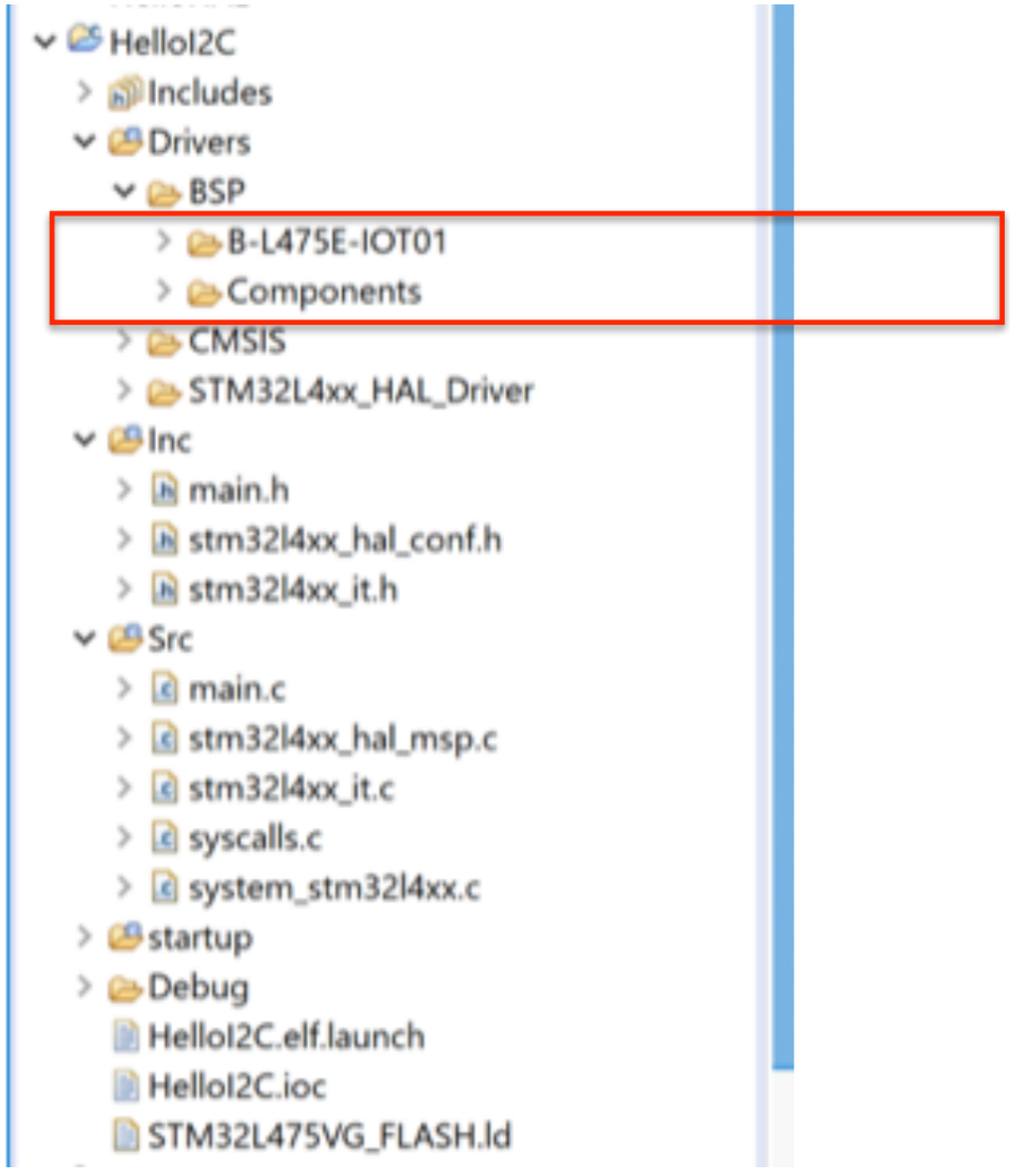
Remember that BSP folder is installed as part of STM32CubeMX files



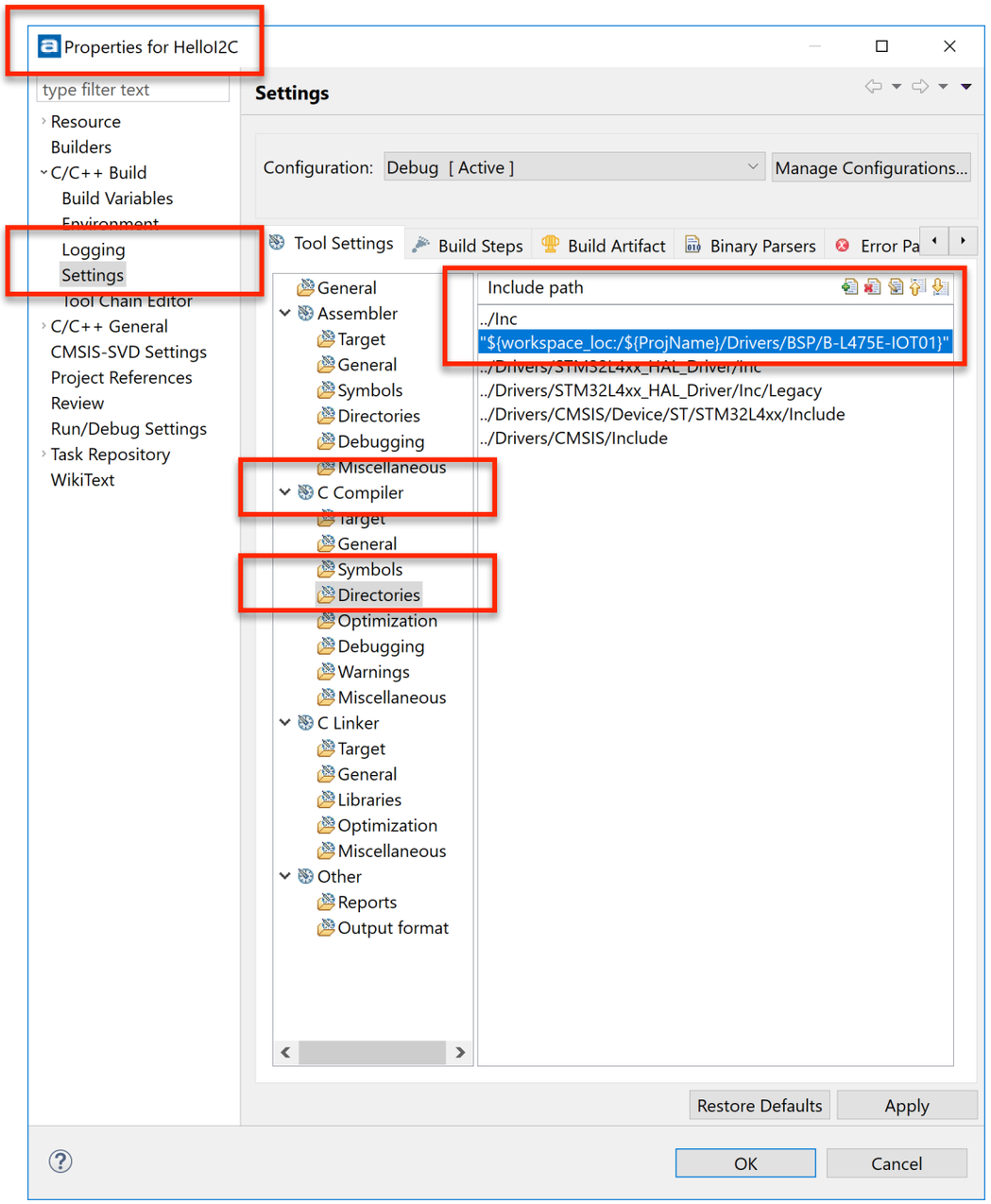


Step: Delete boards we are not using - we are only using B-L475E-IOT01

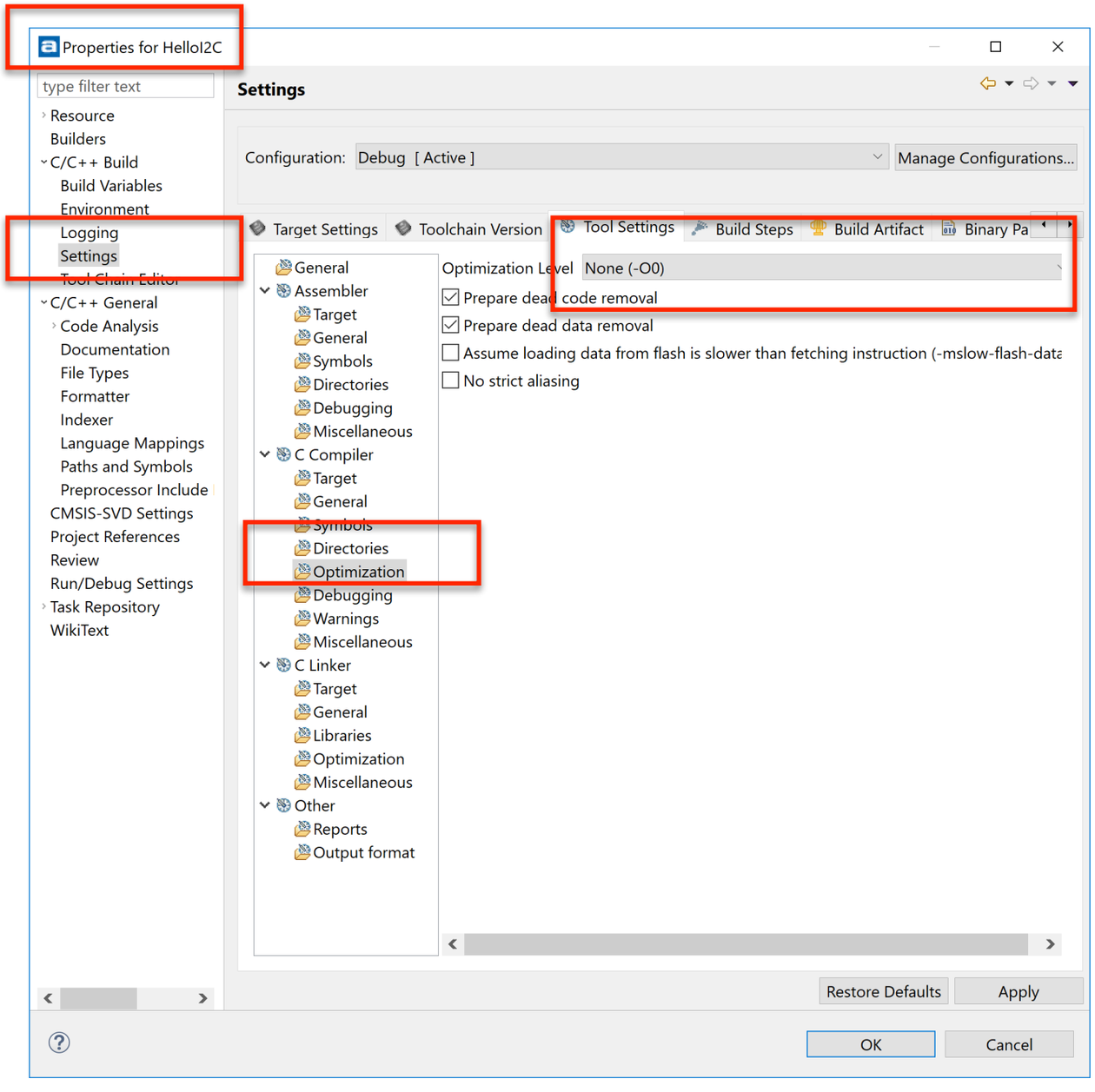
WARNING:
Do not delete “Components” folder!



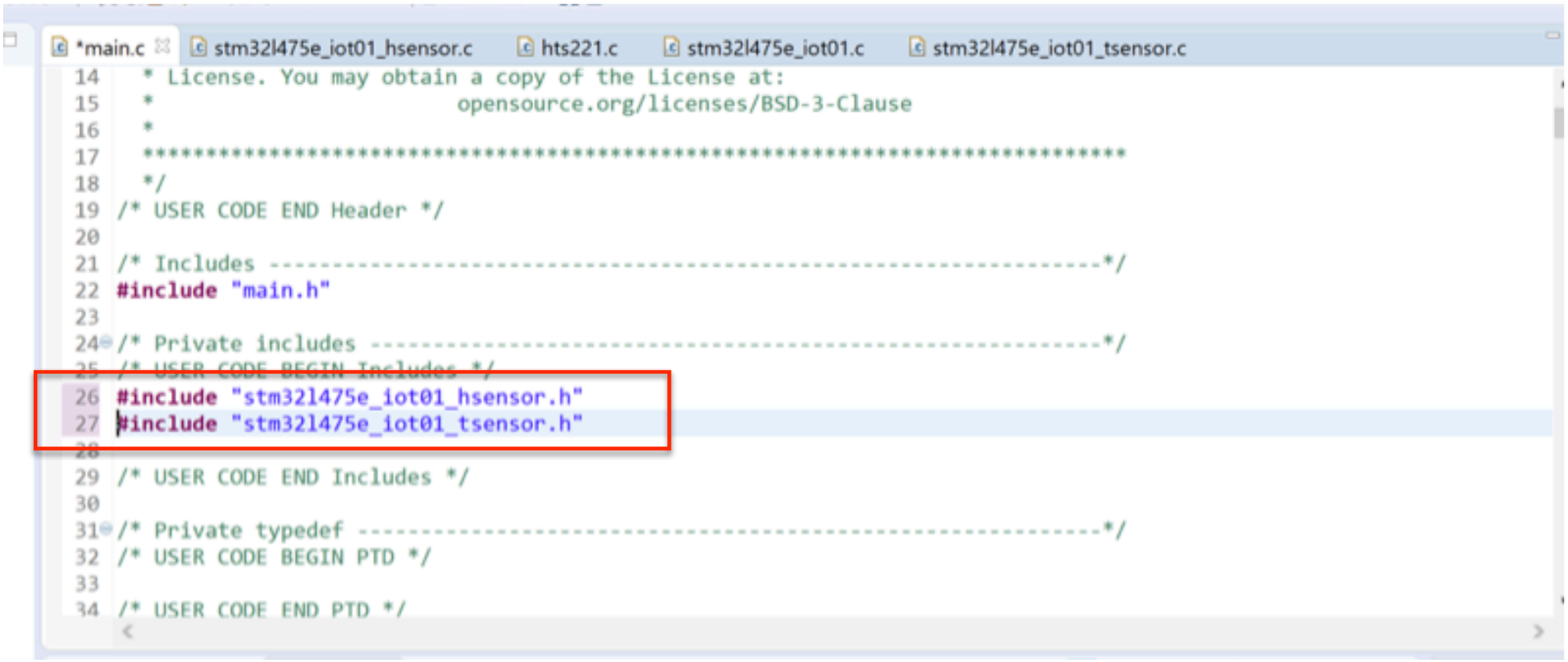
Step: Add BSP to search directory path



Step: Turn OFF optimization to help with debug



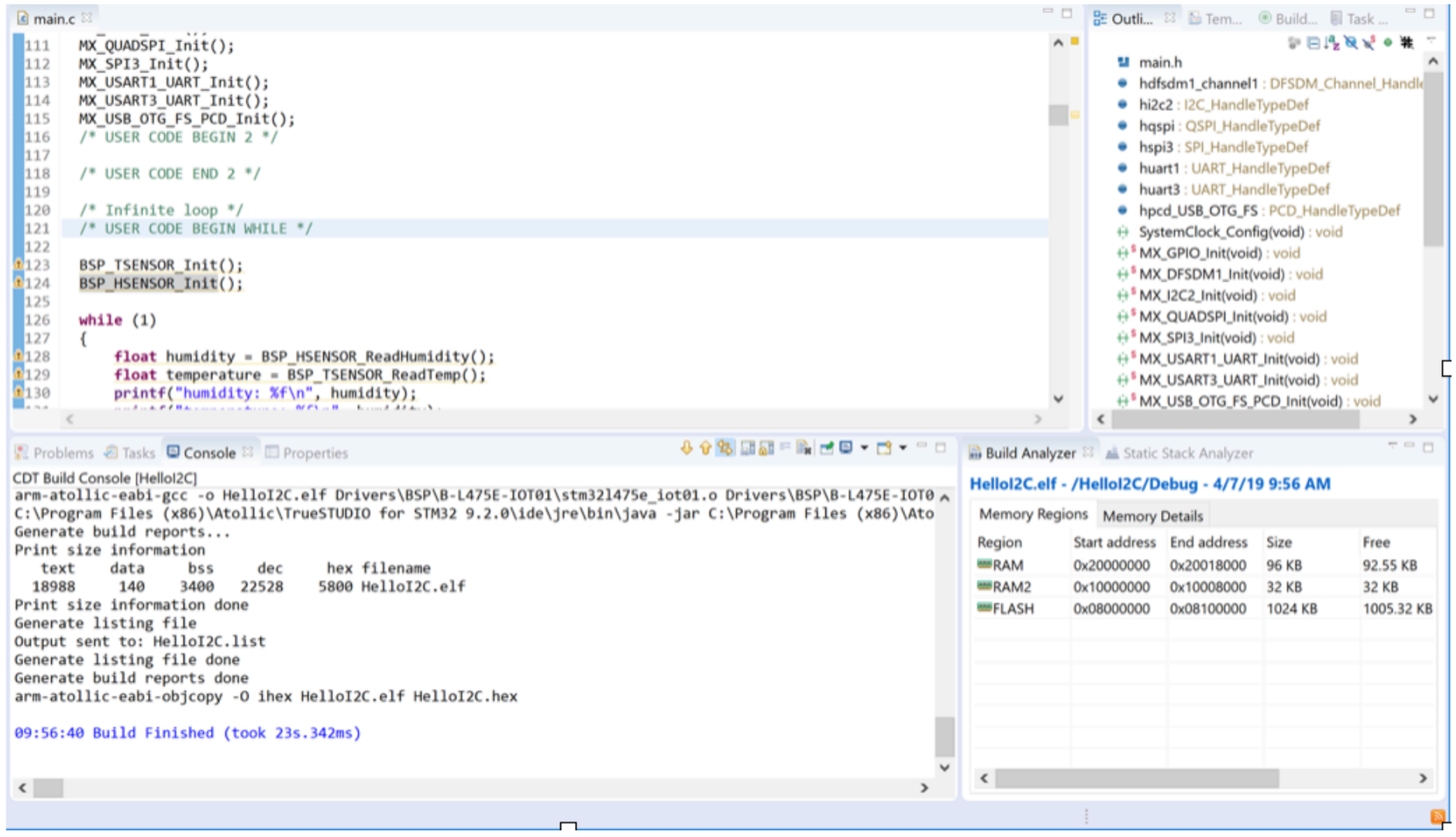
Step: main.c - Add #include files



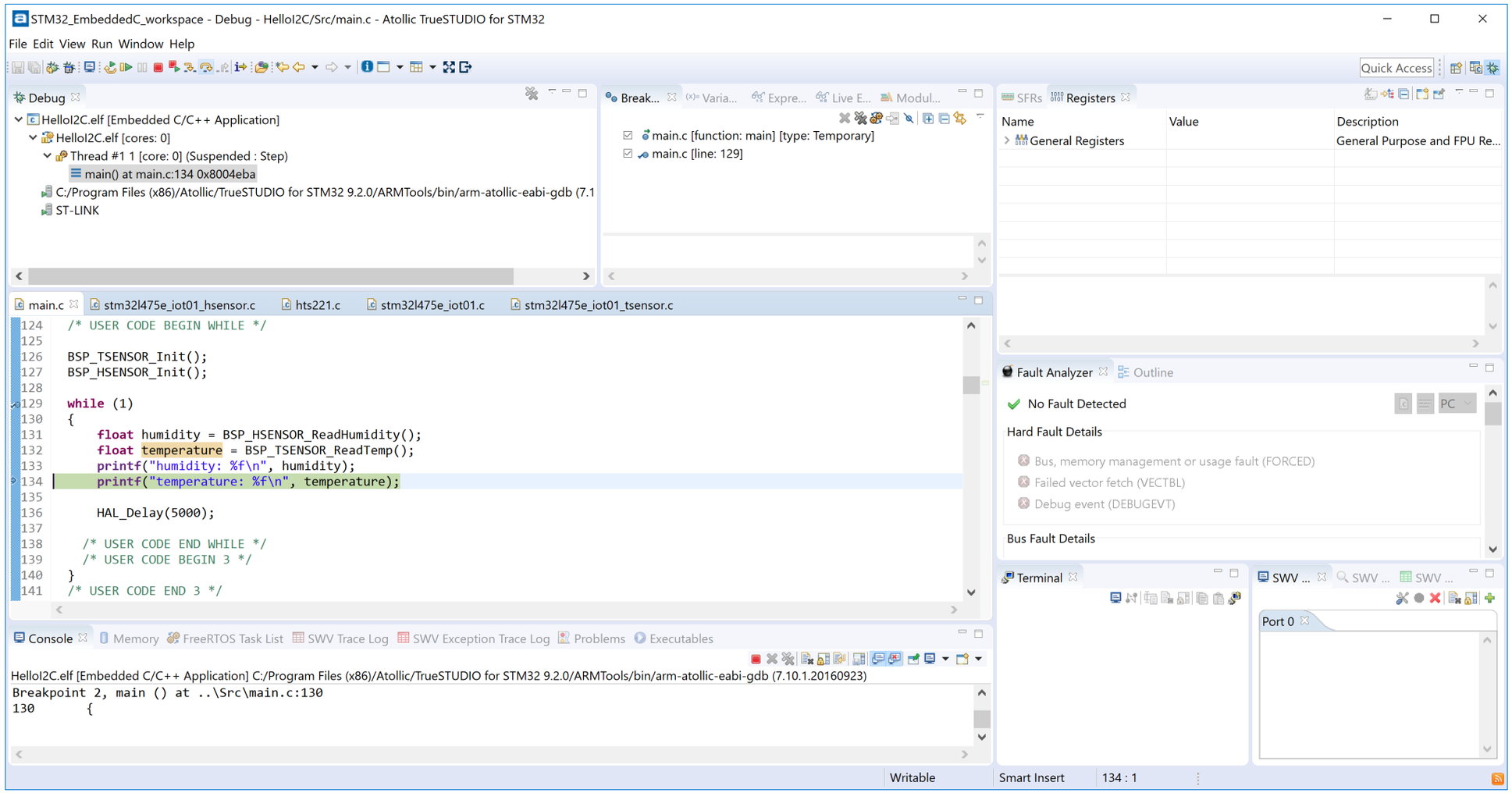
Step: main.c - Add code

```
119
120   /* Infinite loop */
121   /* USER CODE BEGIN WHILE */
122
123   BSP_TSENSOR_Init();
124   BSP_HSENSOR_Init();
125
126   while (1)
127   {
128     float humidity = BSP_HSENSOR_ReadHumidity();
129     float temperature = BSP_TSENSOR_ReadTemp();
130     printf("humidity: %f\n", humidity);
131     printf("temperature: %f\n", humidity);
132
133     HAL_Delay(5000);
134
135     /* USER CODE END WHILE */
136     /* USER CODE BEGIN 3 */
137   }
138   /* USER CODE END 3 */
139 }
140
141 /**
```

Step: Build

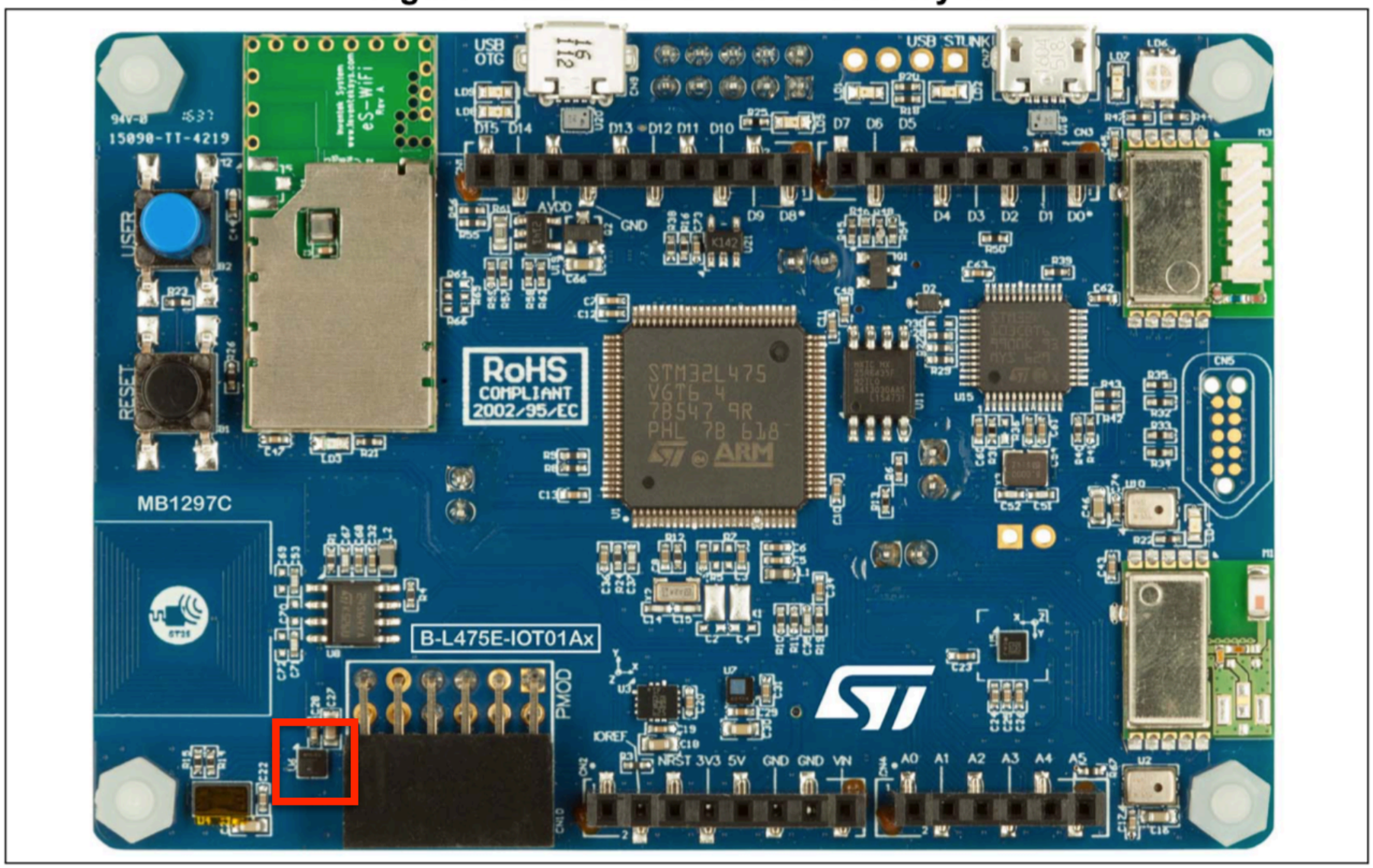


Step: Step Breakpoint, Run, Observe humidity and temperature



Step: Location of HTS221 Humidity and Temperature Sensor

Figure 1. B-L475E-IOT01A Discovery kit



Summary

- Introduction to I2C
- HAL and I2C
- STM32CubeMX and I2C Code Generation
- Tour of I2C
- TrueStudio and I2C
  - HTS221 Humidity and Temperature Sensor