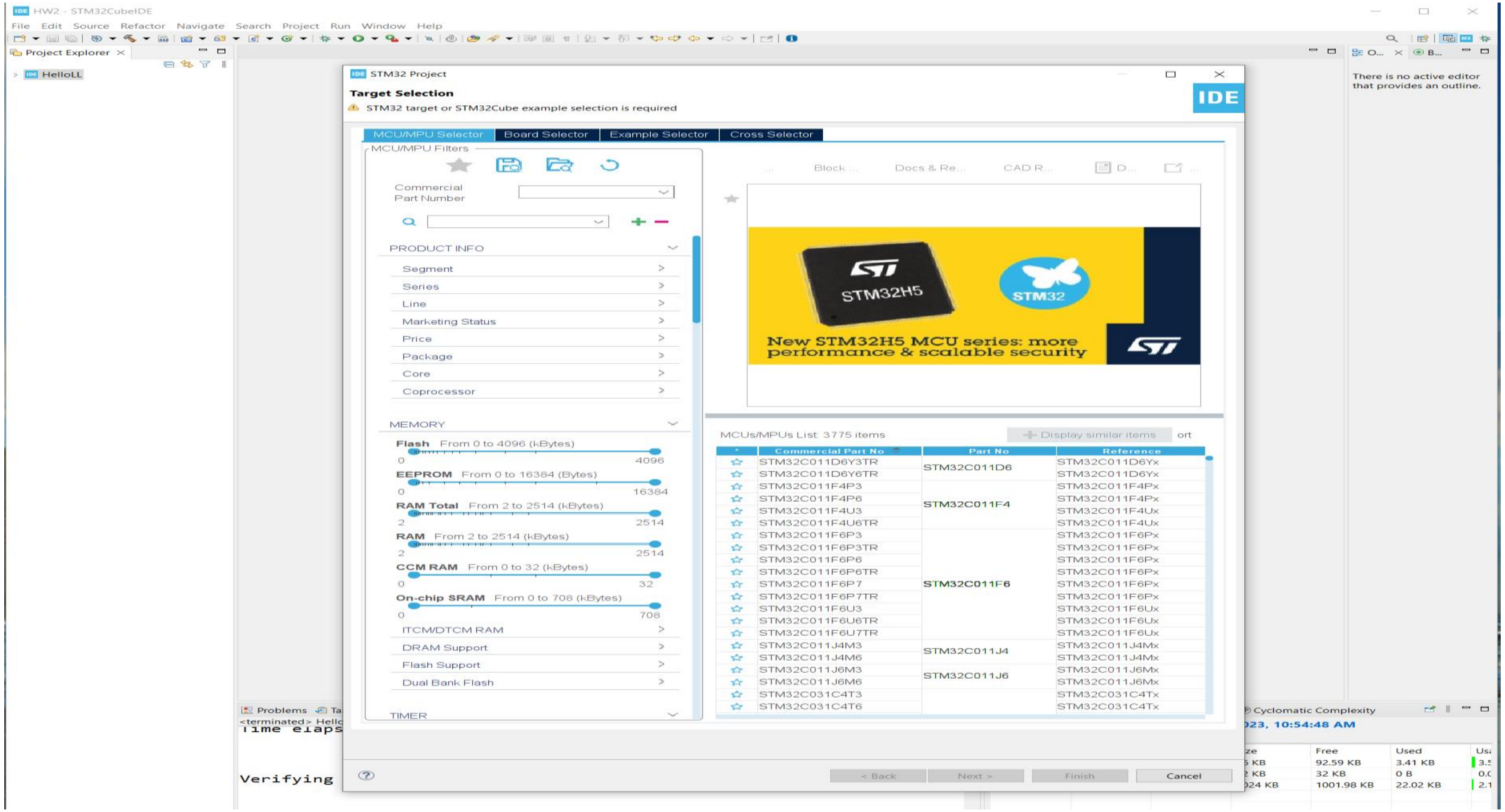
UCSD Embedded C Assignment 4

By

Hsuankai Chang

hsuankac@umich.edu

Step 1. Startup STM32CubeIDE and create new STM32 project



Target Selection

⚠ STM32 target or STM32Cube example selection is required

MCU/MPU Filters

Commercial Part Number

PRODUCT INFO

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

MEMORY

- Flash From 0 to 4096 (kBytes) 0 – 4096
- EEPROM From 0 to 16384 (Bytes) 0 – 16384
- RAM Total From 2 to 2514 (kBytes) 2 – 2514
- RAM From 2 to 2514 (kBytes) 2 – 2514
- CCM RAM From 0 to 32 (kBytes) 0 – 32
- On-chip SRAM From 0 to 708 (kBytes) 0 – 708
- ITCM/DTCM RAM >
- DRAM Support >
- Flash Support >
- Dual Bank Flash >

TIMER

MCUs/MPUs List: 3775 items

*	Commercial Part No	Part No	Reference
☆	STM32C011D6Y3TR	STM32C011D6	STM32C011D6Yx
☆	STM32C011D6Y6TR		STM32C011D6Yx
☆	STM32C011F4P3	STM32C011F4	STM32C011F4Px
☆	STM32C011F4P6		STM32C011F4Px
☆	STM32C011F4U3		STM32C011F4Ux
☆	STM32C011F4U6TR		STM32C011F4Ux
☆	STM32C011F6P3	STM32C011F6	STM32C011F6Px
☆	STM32C011F6P3TR		STM32C011F6Px
☆	STM32C011F6P6		STM32C011F6Px
☆	STM32C011F6P6TR		STM32C011F6Px
☆	STM32C011F6P7		STM32C011F6Px
☆	STM32C011F6P7TR		STM32C011F6Px
☆	STM32C011F6U3		STM32C011F6Ux
☆	STM32C011F6U6TR		STM32C011F6Ux
☆	STM32C011F6U7TR		STM32C011F6Ux
☆	STM32C011J4M3	STM32C011J4	STM32C011J4Mx
☆	STM32C011J4M6		STM32C011J4Mx
☆	STM32C011J6M3	STM32C011J6	STM32C011J6Mx
☆	STM32C011J6M6		STM32C011J6Mx
☆	STM32C031C4T3		STM32C031C4Tx
☆	STM32C031C4T6		STM32C031C4Tx

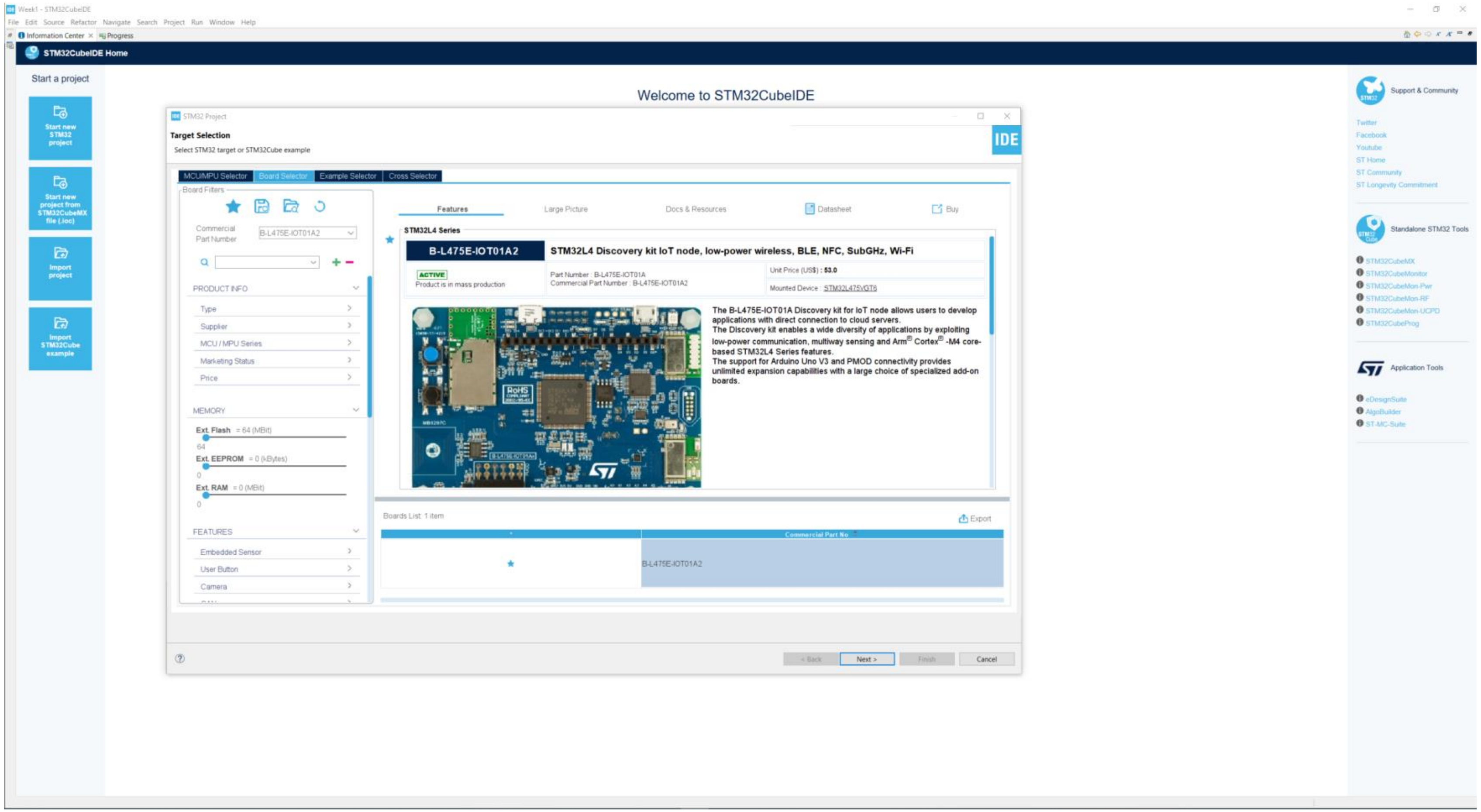
There is no active editor that provides an outline.

```
<terminated> Hello
Time elaps

Verifying
```

023, 10:54:48 AM

ze	Free	Used	Us
5 KB	92.59 KB	3.41 KB	3.5
2 KB	32 KB	0 B	0.0
024 KB	1001.98 KB	22.02 KB	2.1

Step 2. Access board selector and type in the board you use, click Next



Target Selection

Select STM32 target or STM32Cube example

Commercial Part Number: B-L475E-IOT01A2

Ext. Flash = 64 (MBit)

Ext. EEPROM = 0 (kBytes)

Ext. RAM = 0 (MBit)

STM32L4 Series

B-L475E-IOT01A2 STM32L4 Discovery kit IoT node, low-power wireless, BLE, NFC, SubGHz, Wi-Fi

ACTIVE
Product is in mass production

Part Number : B-L475E-IOT01A
Commercial Part Number : B-L475E-IOT01A2

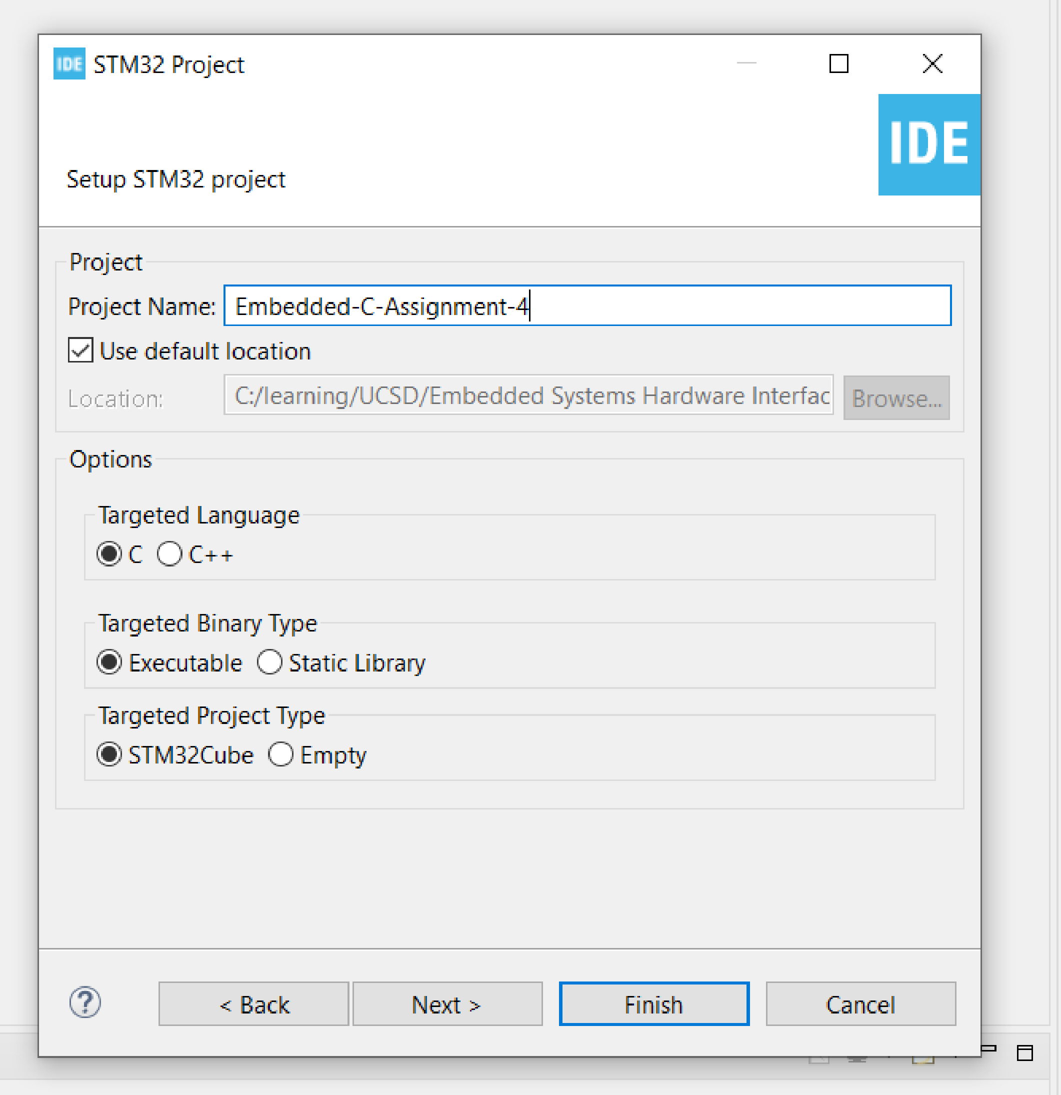
Unit Price (US$) : 53.0
Mounted Device : STM32L475VGT6

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.

Boards List: 1 item

*	Commercial Part No
★	B-L475E-IOT01A2

Step 3. Enter the project name then click Next



STM32 Project

Setup STM32 project

Project

Project Name: Embedded-C-Assignment-4

☑ Use default location

Location: C:/learning/UCSD/Embedded Systems Hardware Interfac

Options

Targeted Language

● C ○ C++

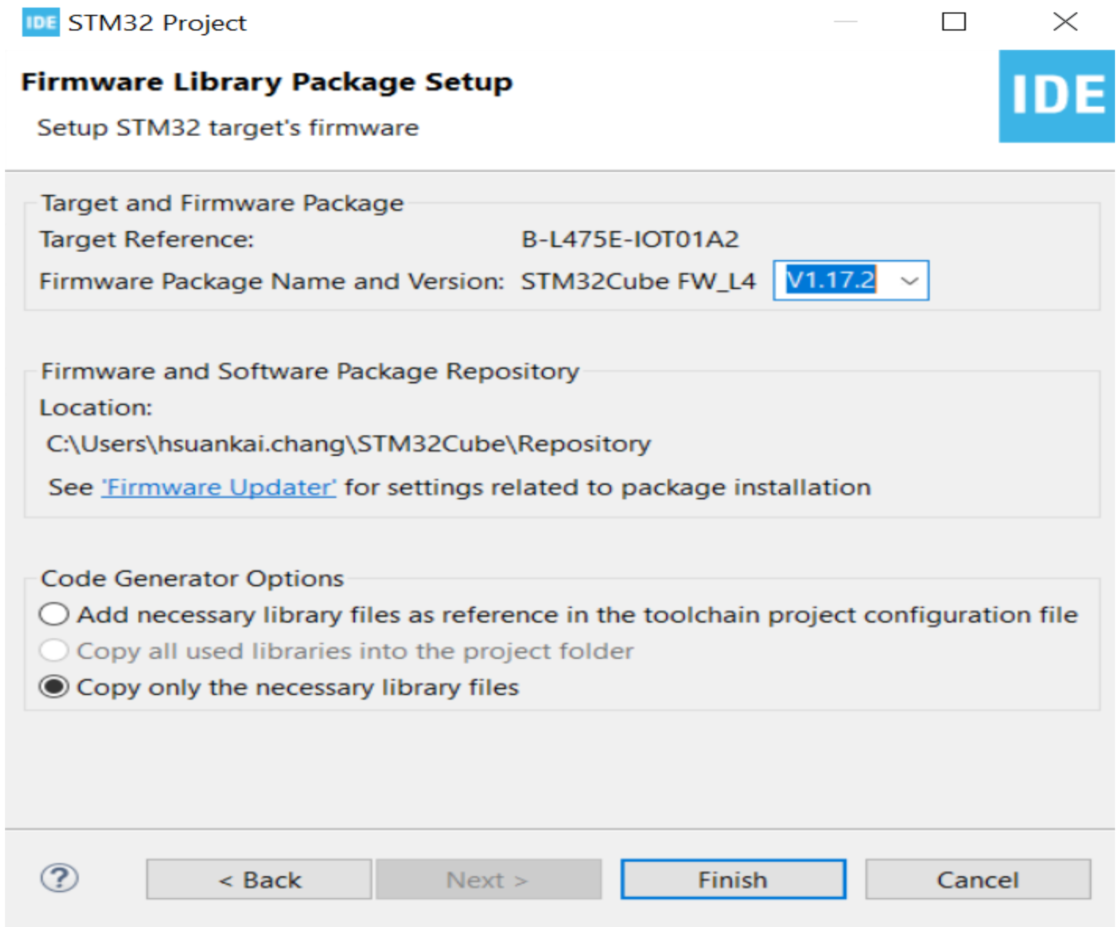
Targeted Binary Type

● Executable ○ Static Library

Targeted Project Type

● STM32Cube ○ Empty

Step 4. See the firmware package name and version



Firmware Library Package Setup

Setup STM32 target's firmware

Target and Firmware Package

Target Reference: B-L475E-IOT01A2

Firmware Package Name and Version: STM32Cube FW_L4 V1.17.2

Firmware and Software Package Repository

Location:

C:\Users\hsuankai.chang\STM32Cube\Repository

See 'Firmware Updater' for settings related to package installation

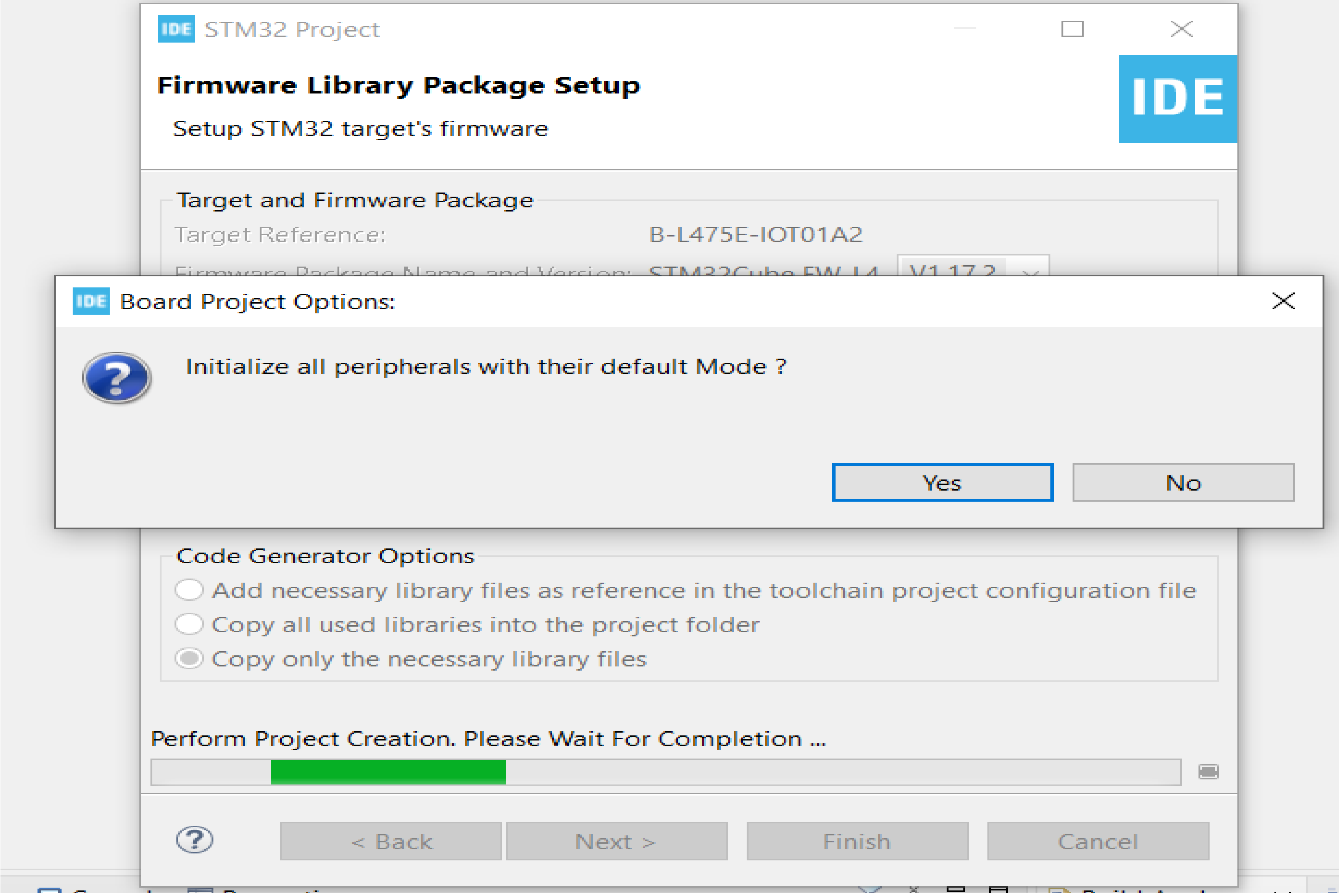
Code Generator Options

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

○ Copy all used libraries into the project folder

● Copy only the necessary library files

Step 5. Click yes to initialize all peripherals to default



STM32 Project

Firmware Library Package Setup

Setup STM32 target's firmware

Target and Firmware Package

Target Reference: B-L475E-IOT01A2

Firmware Package Name and Version: STM32Cube FW_L4 V1.17.2

Board Project Options:

Initialize all peripherals with their default Mode ?

Code Generator Options

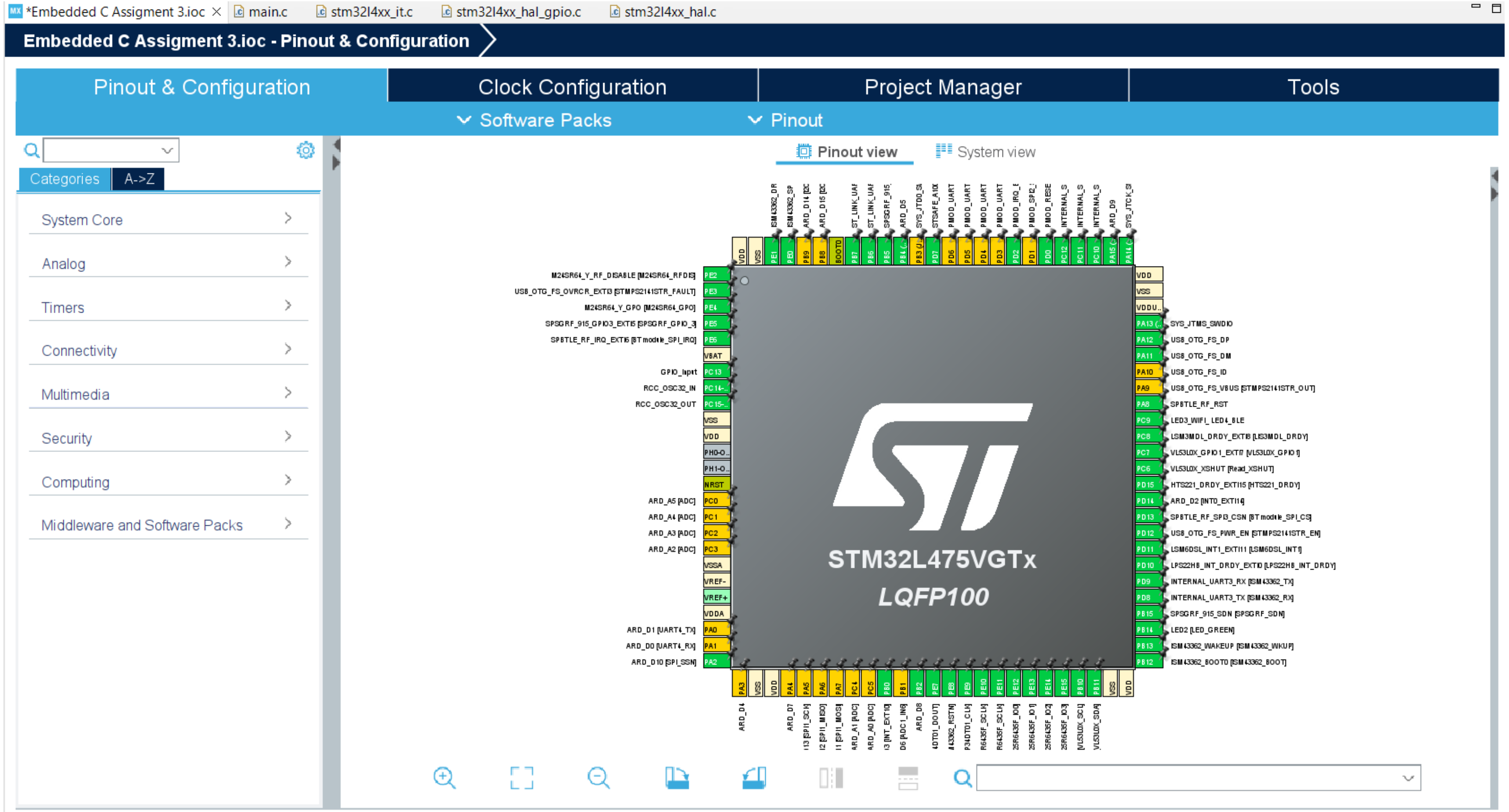
○ Add necessary library files as reference in the toolchain project configuration file

○ Copy all used libraries into the project folder

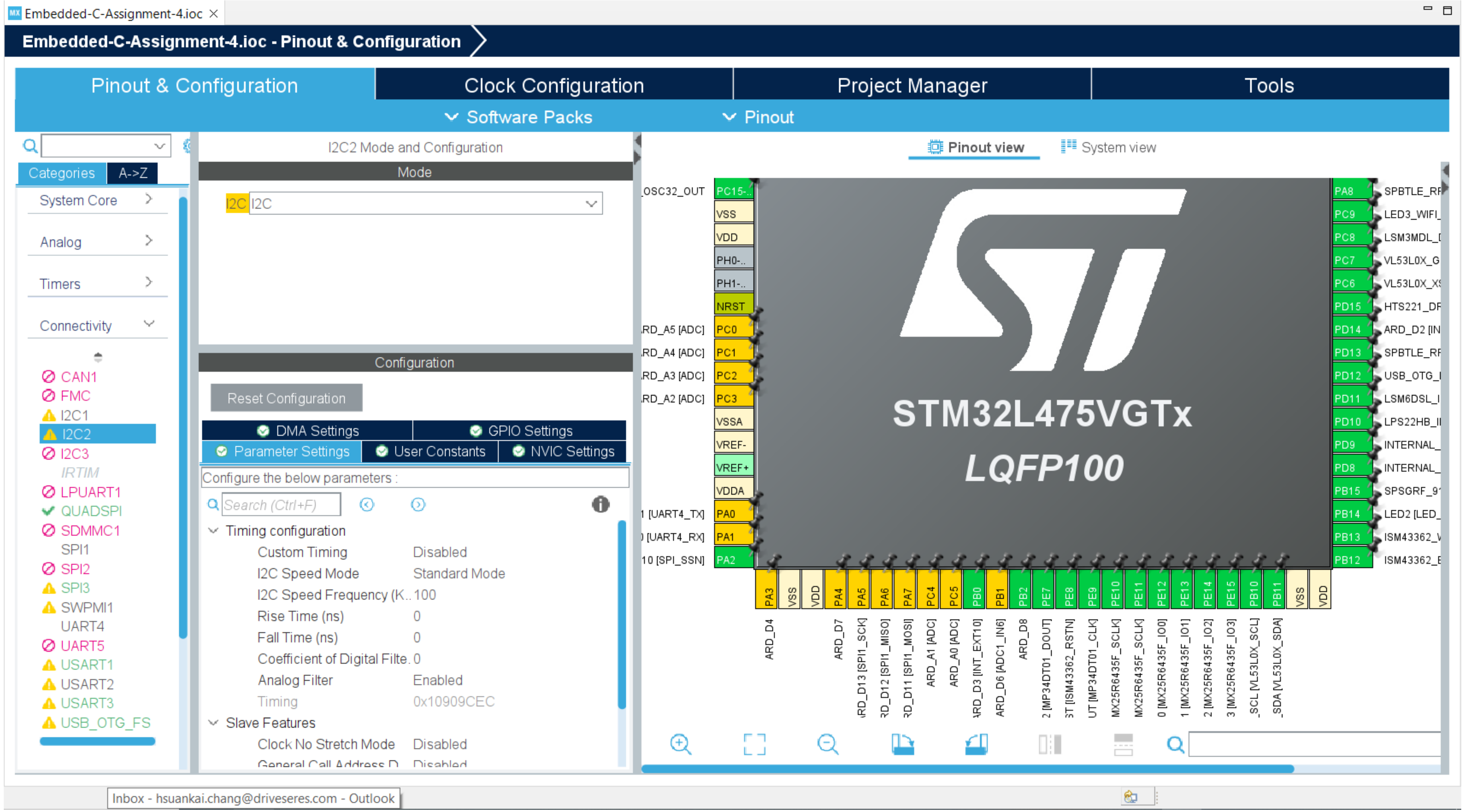
● Copy only the necessary library files

Perform Project Creation. Please Wait For Completion ...

Step 6. When in .ioc file, click Pinout & Configurations



Step 7. Keep the default I2C settings



I2C2 Mode and Configuration

Mode

I2C I2C

Configuration

Reset Configuration

DMA Settings | GPIO Settings

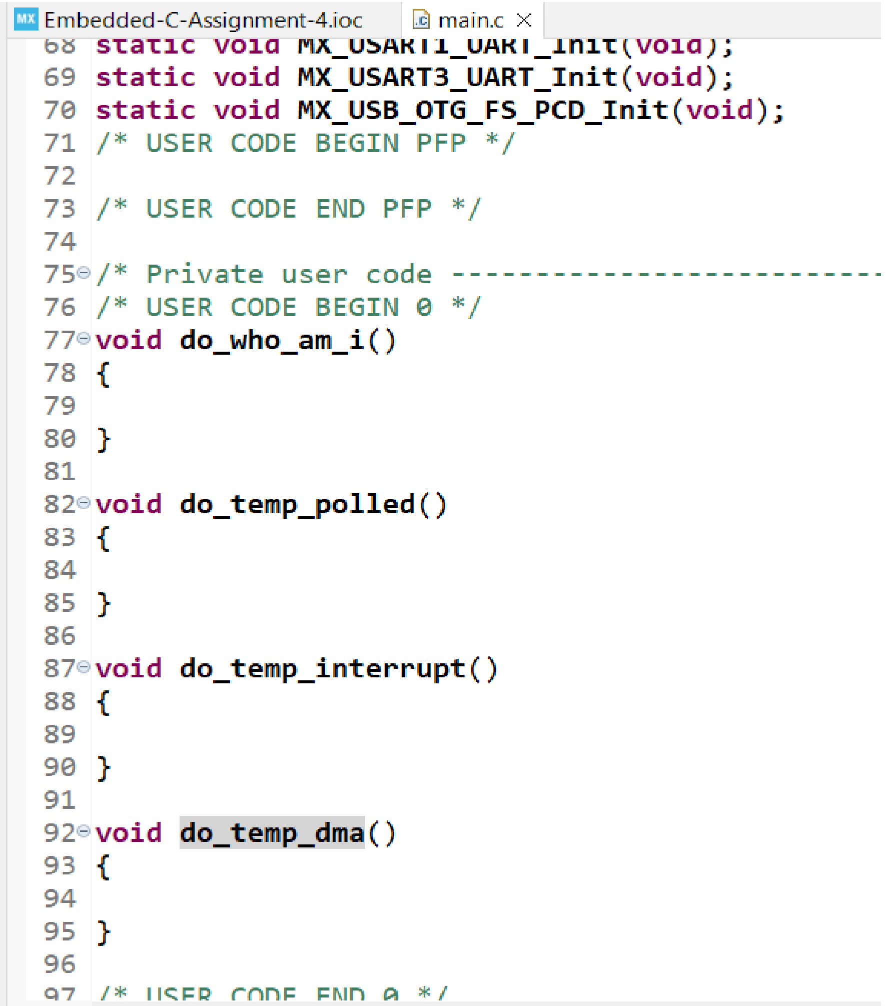
Parameter Settings | User Constants | NVIC Settings

Configure the below parameters :

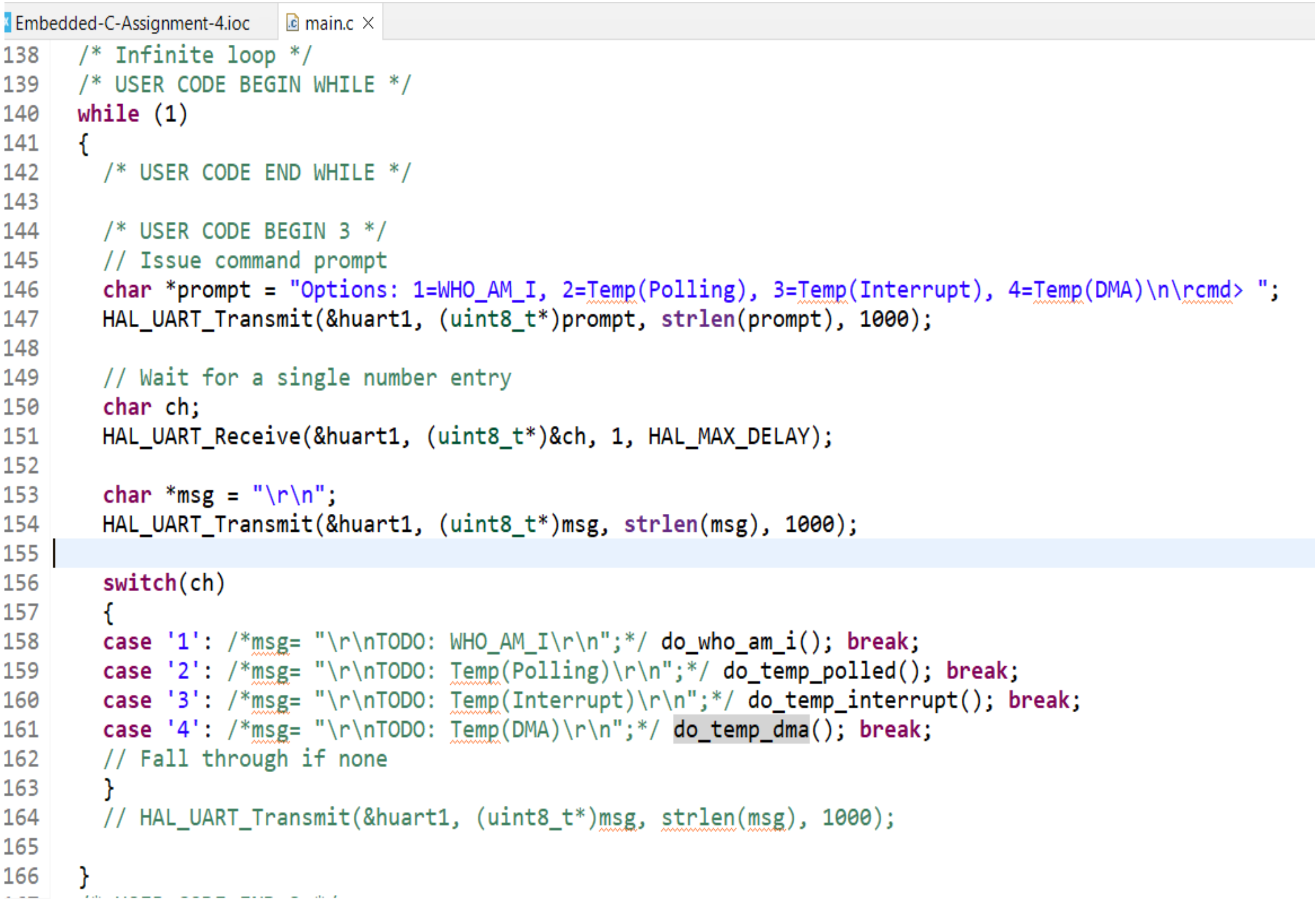
Search (Ctrl+F)

```
Timing configuration
    Custom Timing                  Disabled
    I2C Speed Mode                 Standard Mode
    I2C Speed Frequency (K..       100
    Rise Time (ns)                 0
    Fall Time (ns)                 0
    Coefficient of Digital Filte.. 0
    Analog Filter                  Enabled
    Timing                         0x10909CEC
Slave Features
    Clock No Stretch Mode          Disabled
    General Call Address D..       Disabled
```

Step 8. User Story 1. CLI. Create a CLI (Command Line Interface) on UART1 that prompts you to enter a 1 for WHO_AM_I using polling, 2 temperature read using polling, and 3 for temperature readings using interrupts, and a 4 for temperature read using DMA>

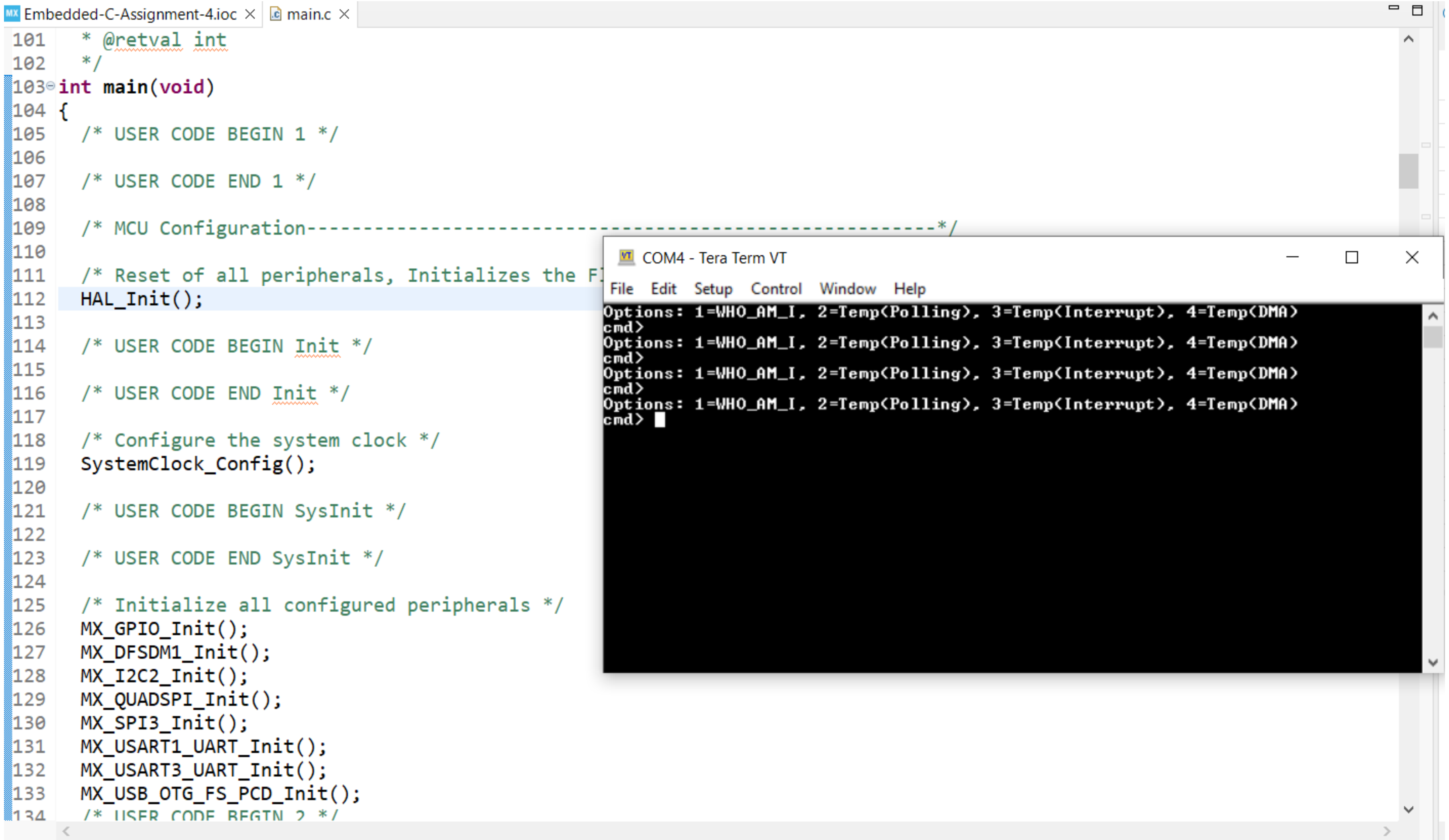


```c
static void MX_USART1_UART_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_USB_OTG_FS_PCD_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------
/* USER CODE BEGIN 0 */
void do_who_am_i()
{

}

void do_temp_polled()
{

}

void do_temp_interrupt()
{

}

void do_temp_dma()
{

}

/* USER CODE END 0 */
```



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

    /* USER CODE BEGIN 3 */
    // Issue command prompt
    char *prompt = "Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)\n\rcmd> ";
    HAL_UART_Transmit(&huart1, (uint8_t*)prompt, strlen(prompt), 1000);

    // Wait for a single number entry
    char ch;
    HAL_UART_Receive(&huart1, (uint8_t*)&ch, 1, HAL_MAX_DELAY);

    char *msg = "\r\n";
    HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), 1000);

    switch(ch)
    {
    case '1': /*msg= "\r\nTODO: WHO_AM_I\r\n";*/ do_who_am_i(); break;
    case '2': /*msg= "\r\nTODO: Temp(Polling)\r\n";*/ do_temp_polled(); break;
    case '3': /*msg= "\r\nTODO: Temp(Interrupt)\r\n";*/ do_temp_interrupt(); break;
    case '4': /*msg= "\r\nTODO: Temp(DMA)\r\n";*/ do_temp_dma(); break;
    // Fall through if none
    }
    // HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), 1000);

  }
```

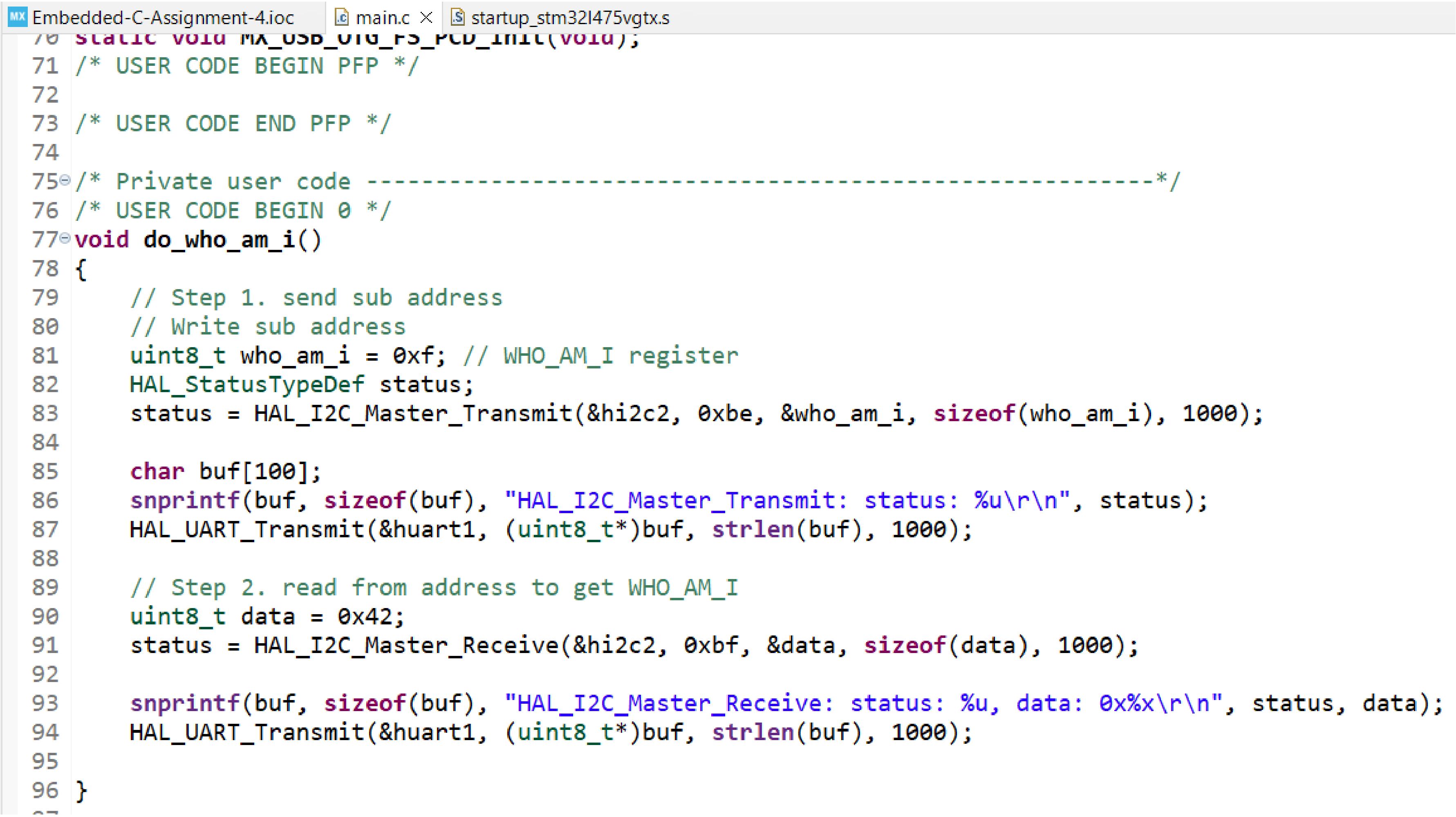
Step 9. User Story 1: Build and run in debug mode. Open tera term and test is successful



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

  /* USER CODE END 1 */

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

  /* Reset of all peripherals, Initializes the Fl
  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 */
```

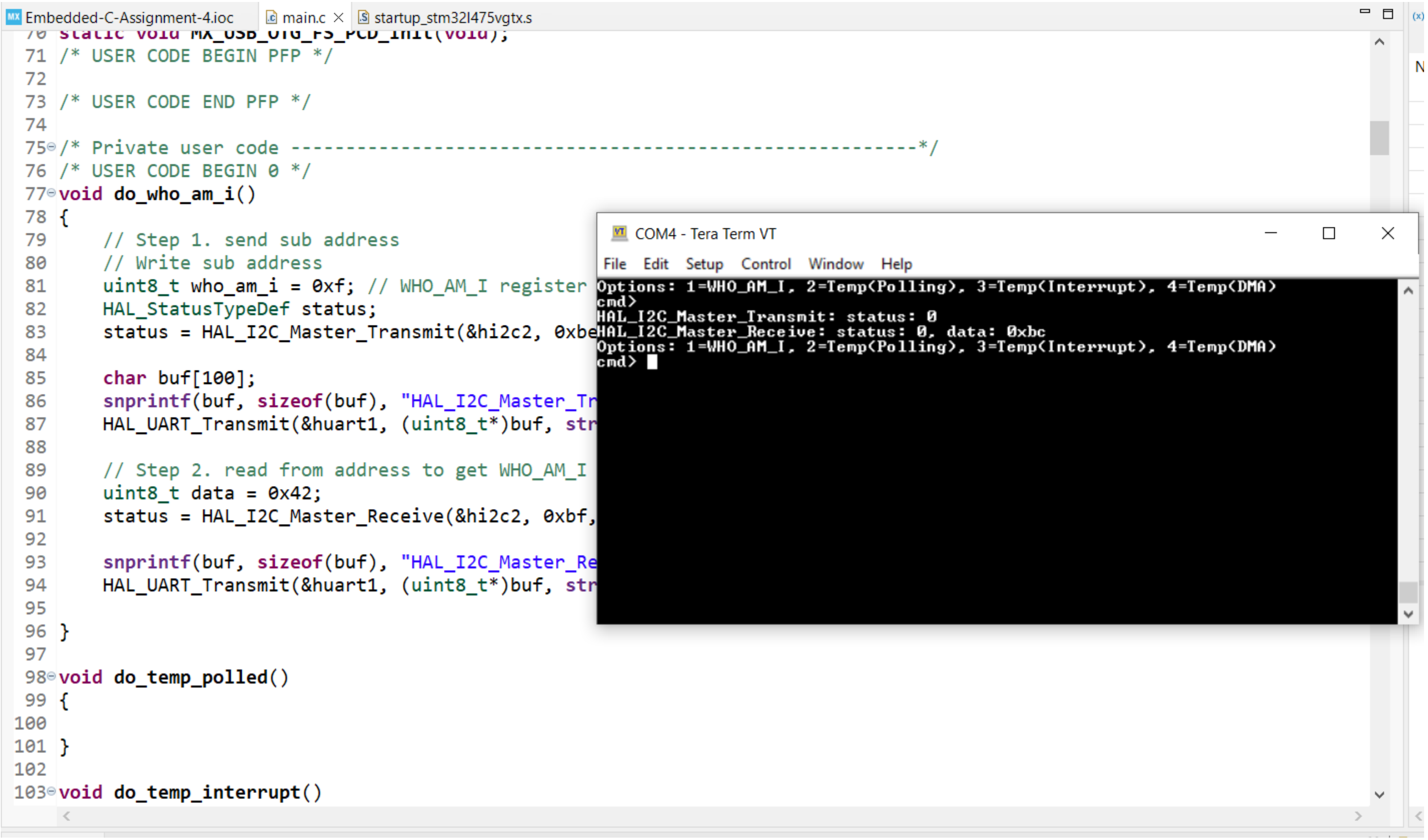
```
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
```

Step 10. User Story 2. WHO_AM_I. Polling. When the user selects 1, display contents of WHO_AM_I register from HTS221



```c
static void MX_USB_OTG_FS_PCD_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void do_who_am_i()
{
    // Step 1. send sub address
    // Write sub address
    uint8_t who_am_i = 0xf; // WHO_AM_I register
    HAL_StatusTypeDef status;
    status = HAL_I2C_Master_Transmit(&hi2c2, 0xbe, &who_am_i, sizeof(who_am_i), 1000);

    char buf[100];
    snprintf(buf, sizeof(buf), "HAL_I2C_Master_Transmit: status: %u\r\n", status);
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);

    // Step 2. read from address to get WHO_AM_I
    uint8_t data = 0x42;
    status = HAL_I2C_Master_Receive(&hi2c2, 0xbf, &data, sizeof(data), 1000);

    snprintf(buf, sizeof(buf), "HAL_I2C_Master_Receive: status: %u, data: 0x%x\r\n", status, data);
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);

}
```

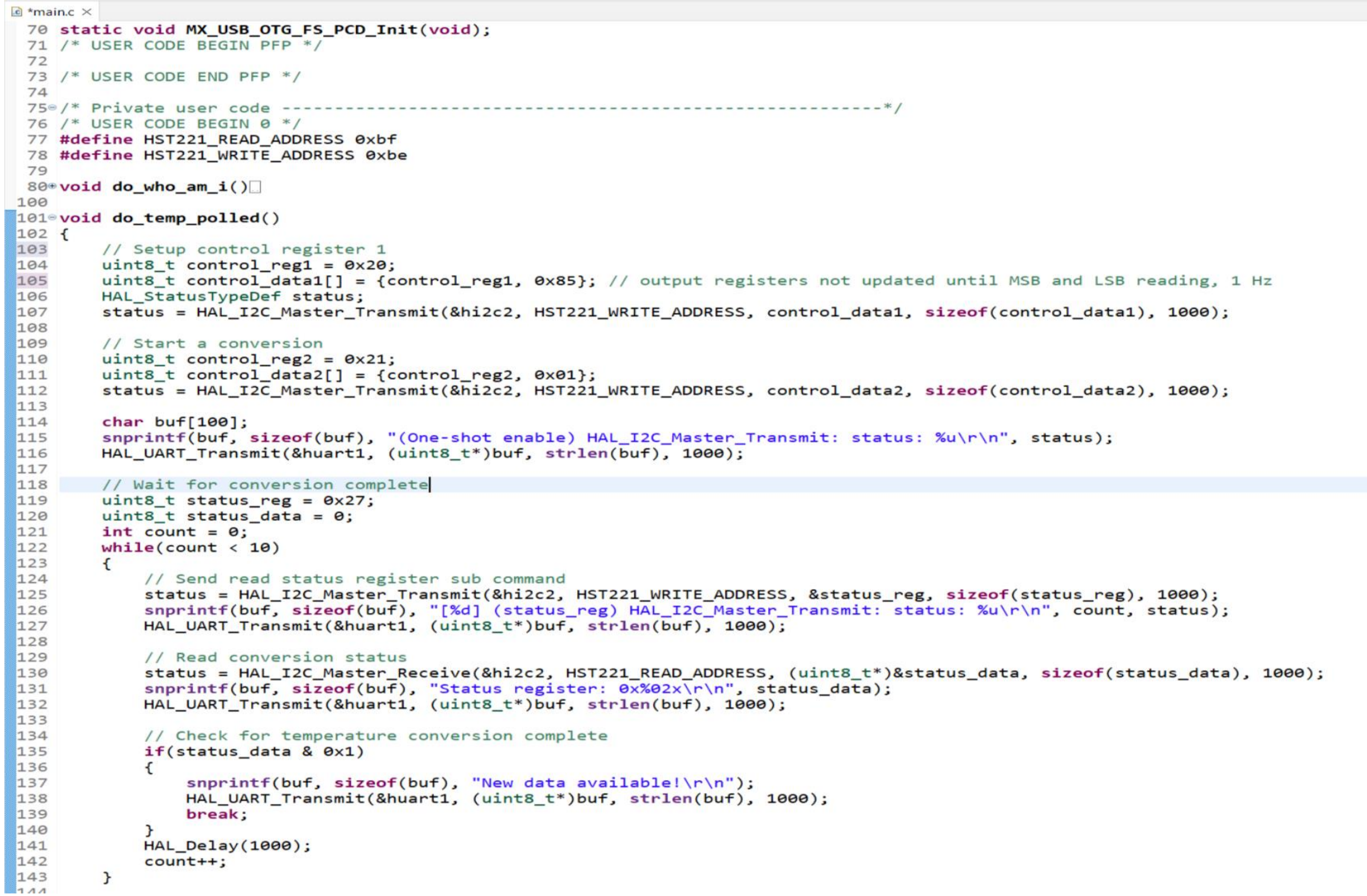
Step 11. User Story 2: Build and run the code, test is successful



```c
static void MX_USB_OTG_FS_PCD_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void do_who_am_i()
{
    // Step 1. send sub address
    // Write sub address
    uint8_t who_am_i = 0xf; // WHO_AM_I register
    HAL_StatusTypeDef status;
    status = HAL_I2C_Master_Transmit(&hi2c2, 0xbe

    char buf[100];
    snprintf(buf, sizeof(buf), "HAL_I2C_Master_Tr
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, str

    // Step 2. read from address to get WHO_AM_I
    uint8_t data = 0x42;
    status = HAL_I2C_Master_Receive(&hi2c2, 0xbf,

    snprintf(buf, sizeof(buf), "HAL_I2C_Master_Re
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, str

}

void do_temp_polled()
{

}

void do_temp_interrupt()
```

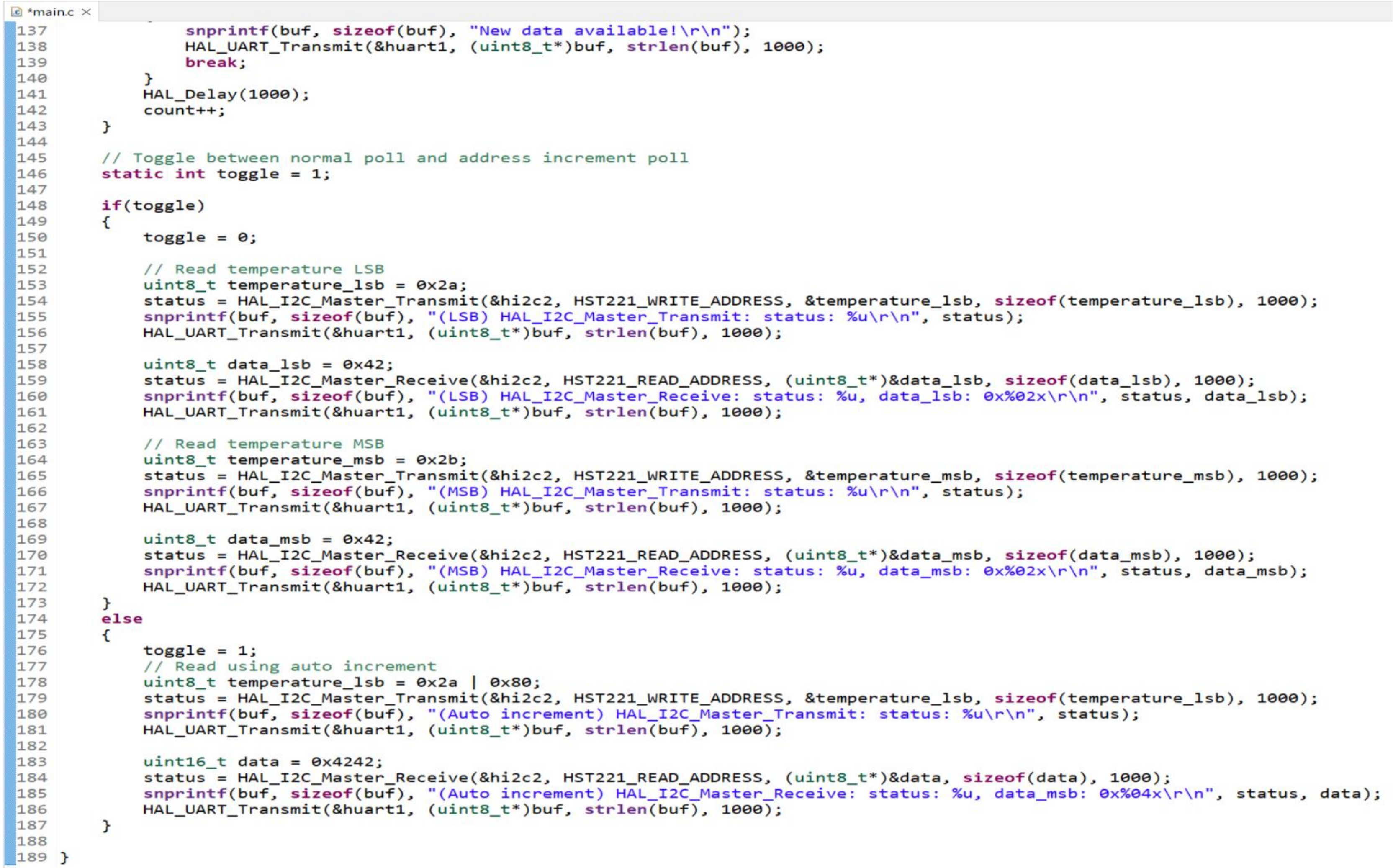
```
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
HAL_I2C_Master_Transmit: status: 0
HAL_I2C_Master_Receive: status: 0, data: 0xbc
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
```

Step 12. User Story 3. Temperature. Polling. The user selects 2, use polling to read HTS221 and display temperature on console.



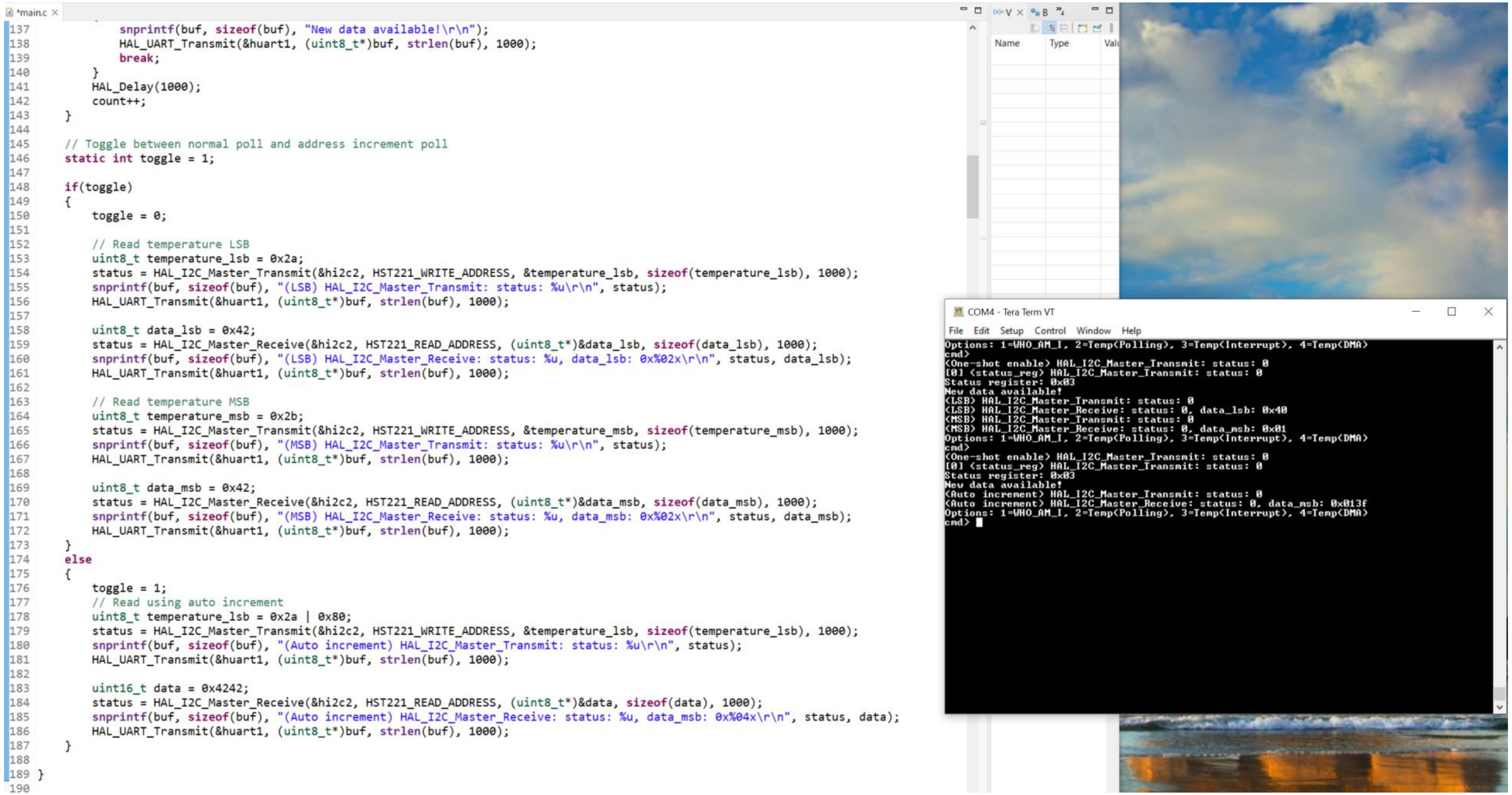
```c
static void MX_USB_OTG_FS_PCD_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
#define HST221_READ_ADDRESS 0xbf
#define HST221_WRITE_ADDRESS 0xbe

void do_who_am_i()

void do_temp_polled()
{
    // Setup control register 1
    uint8_t control_reg1 = 0x20;
    uint8_t control_data1[] = {control_reg1, 0x85}; // output registers not updated until MSB and LSB reading, 1 Hz
    HAL_StatusTypeDef status;
    status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, control_data1, sizeof(control_data1), 1000);

    // Start a conversion
    uint8_t control_reg2 = 0x21;
    uint8_t control_data2[] = {control_reg2, 0x01};
    status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, control_data2, sizeof(control_data2), 1000);

    char buf[100];
    snprintf(buf, sizeof(buf), "(One-shot enable) HAL_I2C_Master_Transmit: status: %u\r\n", status);
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);

    // Wait for conversion complete
    uint8_t status_reg = 0x27;
    uint8_t status_data = 0;
    int count = 0;
    while(count < 10)
    {
        // Send read status register sub command
        status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &status_reg, sizeof(status_reg), 1000);
        snprintf(buf, sizeof(buf), "[%d] (status_reg) HAL_I2C_Master_Transmit: status: %u\r\n", count, status);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);

        // Read conversion status
        status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&status_data, sizeof(status_data), 1000);
        snprintf(buf, sizeof(buf), "Status register: 0x%02x\r\n", status_data);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);

        // Check for temperature conversion complete
        if(status_data & 0x1)
        {
            snprintf(buf, sizeof(buf), "New data available!\r\n");
            HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
            break;
        }
        HAL_Delay(1000);
        count++;
    }
```

Step 13. User Story 3: Toggle between auto increment and direct read from LSB and MSB



```c
            snprintf(buf, sizeof(buf), "New data available!\r\n");
            HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
            break;
        }
        HAL_Delay(1000);
        count++;
    }

    // Toggle between normal poll and address increment poll
    static int toggle = 1;

    if(toggle)
    {
        toggle = 0;

        // Read temperature LSB
        uint8_t temperature_lsb = 0x2a;
        status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb), 1000);
        snprintf(buf, sizeof(buf), "(LSB) HAL_I2C_Master_Transmit: status: %u\r\n", status);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);

        uint8_t data_lsb = 0x42;
        status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data_lsb, sizeof(data_lsb), 1000);
        snprintf(buf, sizeof(buf), "(LSB) HAL_I2C_Master_Receive: status: %u, data_lsb: 0x%02x\r\n", status, data_lsb);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);

        // Read temperature MSB
        uint8_t temperature_msb = 0x2b;
        status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_msb, sizeof(temperature_msb), 1000);
        snprintf(buf, sizeof(buf), "(MSB) HAL_I2C_Master_Transmit: status: %u\r\n", status);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);

        uint8_t data_msb = 0x42;
        status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data_msb, sizeof(data_msb), 1000);
        snprintf(buf, sizeof(buf), "(MSB) HAL_I2C_Master_Receive: status: %u, data_msb: 0x%02x\r\n", status, data_msb);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    }
    else
    {
        toggle = 1;
        // Read using auto increment
        uint8_t temperature_lsb = 0x2a | 0x80;
        status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb), 1000);
        snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Transmit: status: %u\r\n", status);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);

        uint16_t data = 0x4242;
        status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data, sizeof(data), 1000);
        snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive: status: %u, data_msb: 0x%04x\r\n", status, data);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    }

}
```

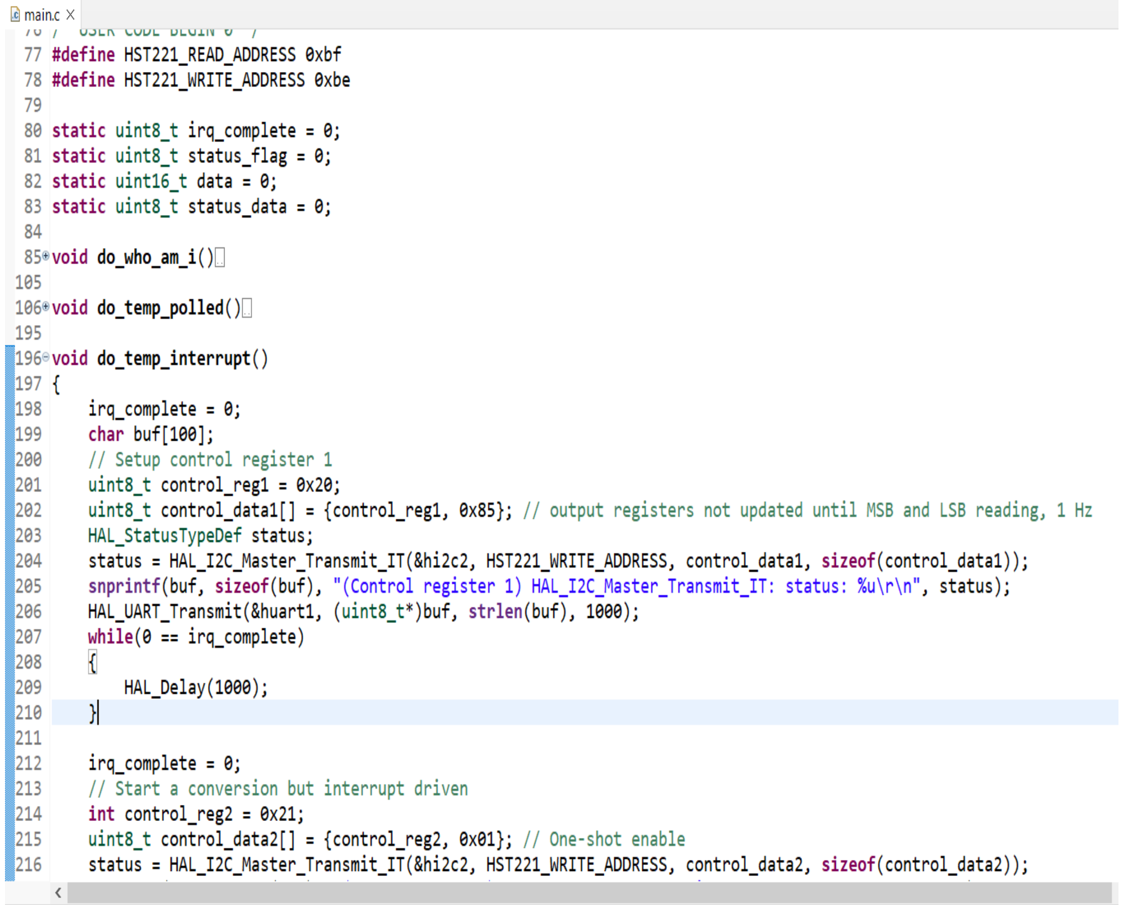
Step 14. User Story 3: Build and run the code, test is successful



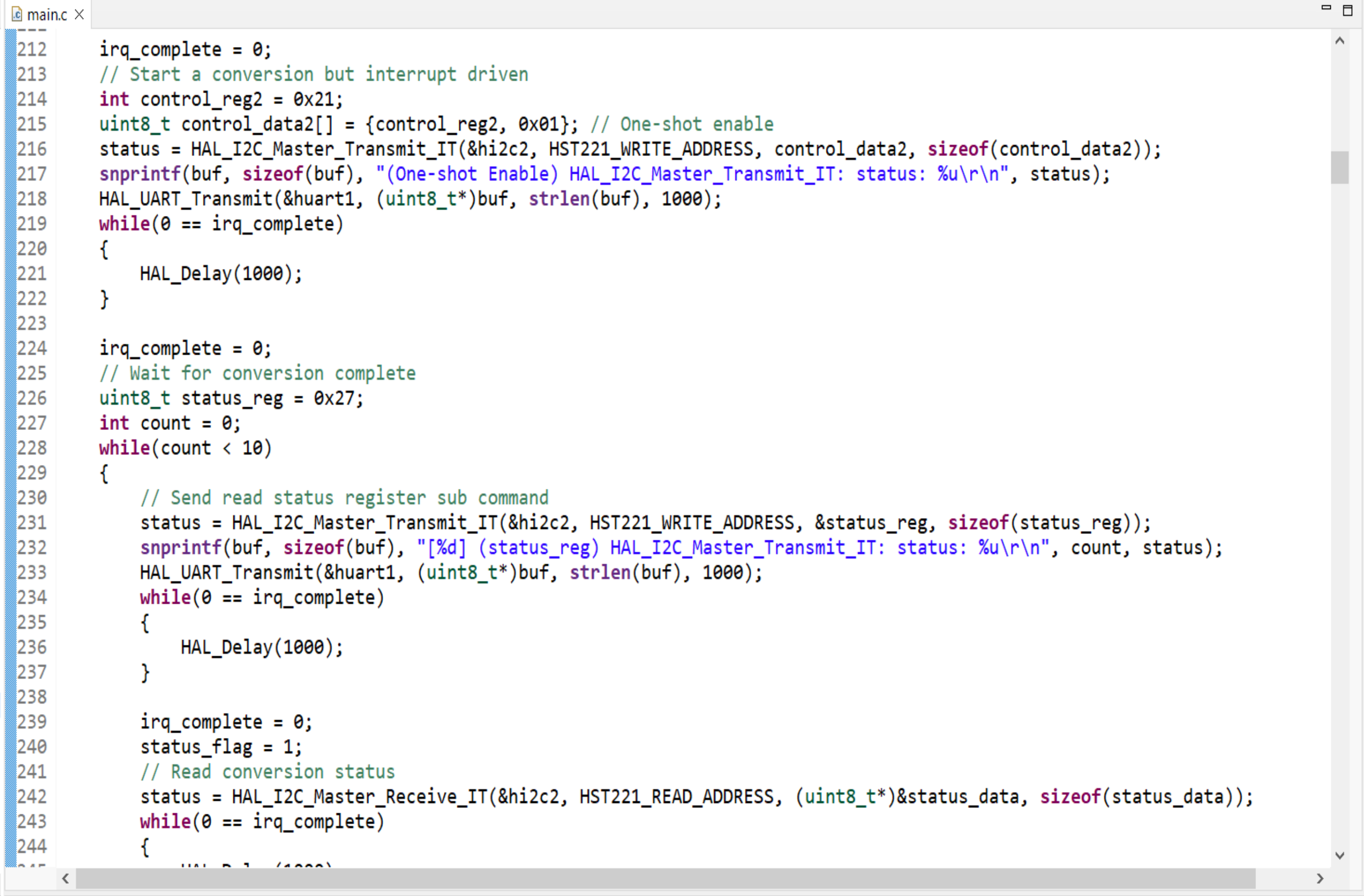
```c
            snprintf(buf, sizeof(buf), "New data available!\r\n");
            HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
            break;
        }
        HAL_Delay(1000);
        count++;
    }

    // Toggle between normal poll and address increment poll
    static int toggle = 1;

    if(toggle)
    {
        toggle = 0;

        // Read temperature LSB
        uint8_t temperature_lsb = 0x2a;
        status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb), 1000);
        snprintf(buf, sizeof(buf), "(LSB) HAL_I2C_Master_Transmit: status: %u\r\n", status);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);

        uint8_t data_lsb = 0x42;
        status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data_lsb, sizeof(data_lsb), 1000);
        snprintf(buf, sizeof(buf), "(LSB) HAL_I2C_Master_Receive: status: %u, data_lsb: 0x%02x\r\n", status, data_lsb);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);

        // Read temperature MSB
        uint8_t temperature_msb = 0x2b;
        status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_msb, sizeof(temperature_msb), 1000);
        snprintf(buf, sizeof(buf), "(MSB) HAL_I2C_Master_Transmit: status: %u\r\n", status);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);

        uint8_t data_msb = 0x42;
        status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data_msb, sizeof(data_msb), 1000);
        snprintf(buf, sizeof(buf), "(MSB) HAL_I2C_Master_Receive: status: %u, data_msb: 0x%02x\r\n", status, data_msb);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    }
    else
    {
        toggle = 1;
        // Read using auto increment
        uint8_t temperature_lsb = 0x2a | 0x80;
        status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb), 1000);
        snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Transmit: status: %u\r\n", status);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);

        uint16_t data = 0x4242;
        status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data, sizeof(data), 1000);
        snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive: status: %u, data_msb: 0x%04x\r\n", status, data);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    }

}
```

```
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
(One-shot enable) HAL_I2C_Master_Transmit: status: 0
[0] (status_reg) HAL_I2C_Master_Transmit: status: 0
Status register: 0x03
New data available!
(LSB) HAL_I2C_Master_Transmit: status: 0
(LSB) HAL_I2C_Master_Receive: status: 0, data_lsb: 0x40
(MSB) HAL_I2C_Master_Transmit: status: 0
(MSB) HAL_I2C_Master_Receive: status: 0, data_msb: 0x01
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
(One-shot enable) HAL_I2C_Master_Transmit: status: 0
[0] (status_reg) HAL_I2C_Master_Transmit: status: 0
Status register: 0x03
New data available!
(Auto increment) HAL_I2C_Master_Transmit: status: 0
(Auto increment) HAL_I2C_Master_Receive: status: 0, data_msb: 0x013f
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
```

Step 15. User Story 4. Temperature. Interrupts. When the user selects 3, use interrupts to read from HTS221 and display temperature on console.

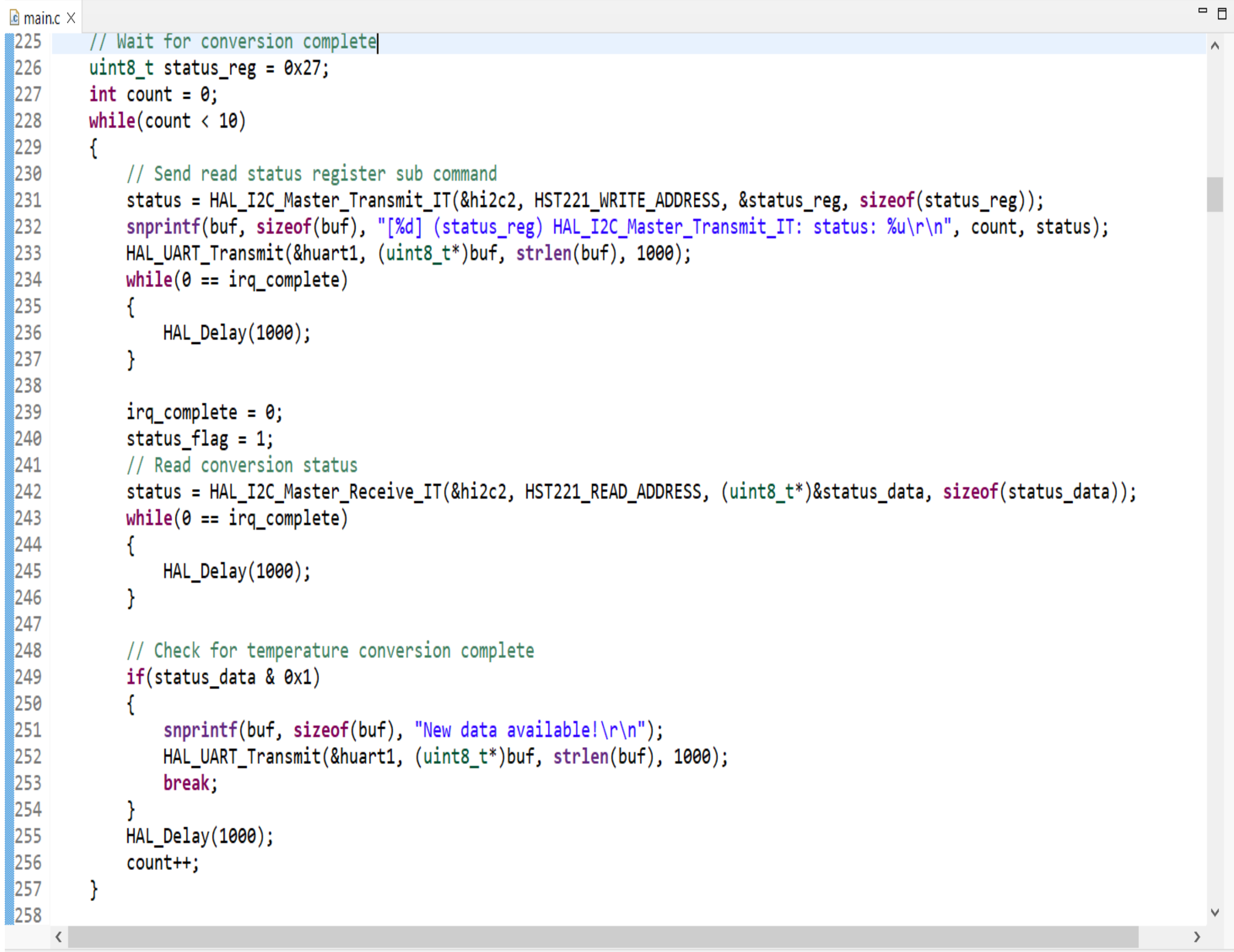


```c
/* USER CODE BEGIN 0 */
#define HST221_READ_ADDRESS 0xbf
#define HST221_WRITE_ADDRESS 0xbe

static uint8_t irq_complete = 0;
static uint8_t status_flag = 0;
static uint16_t data = 0;
static uint8_t status_data = 0;

void do_who_am_i()□

void do_temp_polled()□

void do_temp_interrupt()
{
    irq_complete = 0;
    char buf[100];
    // Setup control register 1
    uint8_t control_reg1 = 0x20;
    uint8_t control_data1[] = {control_reg1, 0x85}; // output registers not updated until MSB and LSB reading, 1 Hz
    HAL_StatusTypeDef status;
    status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, control_data1, sizeof(control_data1));
    snprintf(buf, sizeof(buf), "(Control register 1) HAL_I2C_Master_Transmit_IT: status: %u\r\n", status);
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    while(0 == irq_complete)
    {
        HAL_Delay(1000);
    }

    irq_complete = 0;
    // Start a conversion but interrupt driven
    int control_reg2 = 0x21;
    uint8_t control_data2[] = {control_reg2, 0x01}; // One-shot enable
    status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, control_data2, sizeof(control_data2));
```

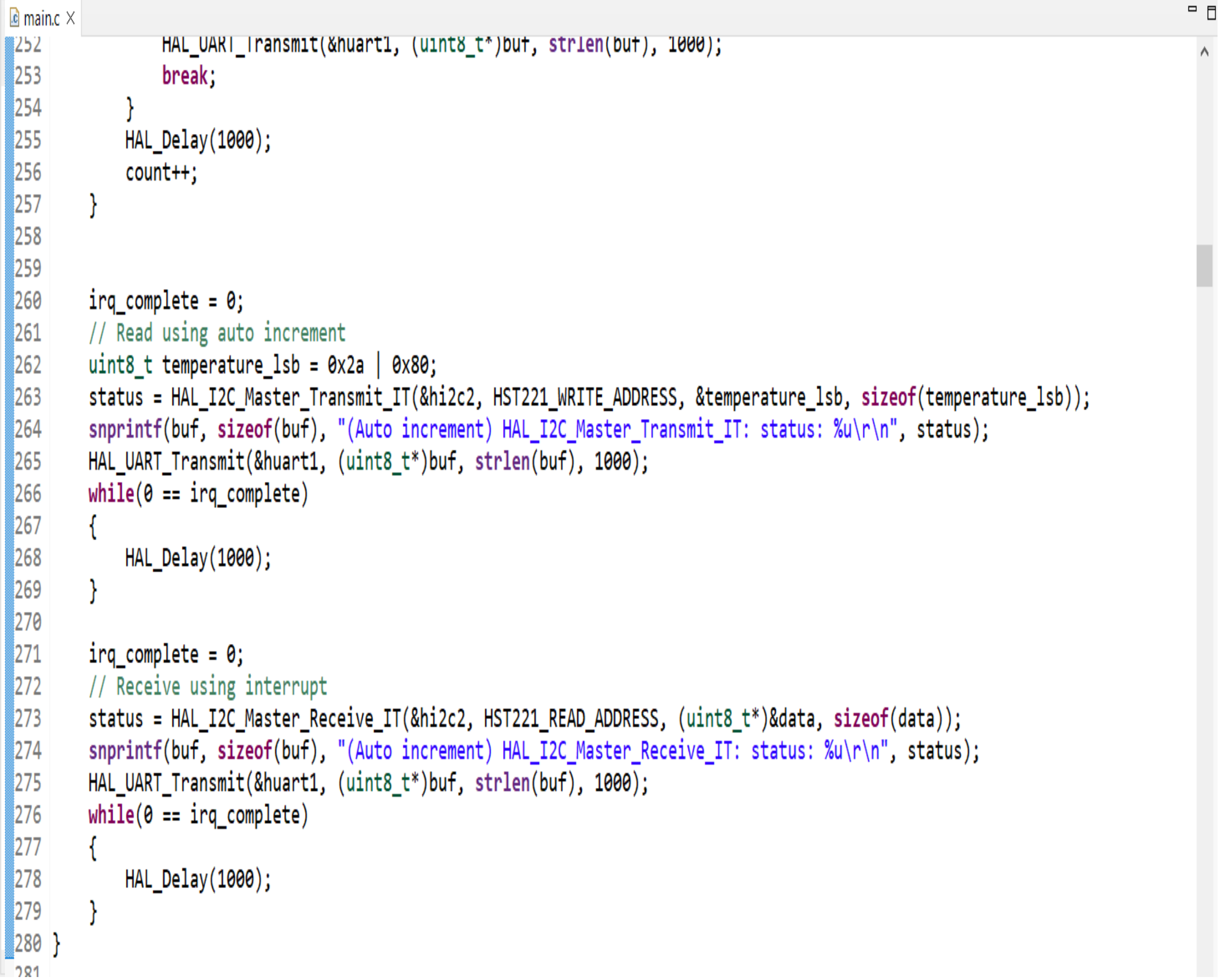


```c
    irq_complete = 0;
    // Start a conversion but interrupt driven
    int control_reg2 = 0x21;
    uint8_t control_data2[] = {control_reg2, 0x01}; // One-shot enable
    status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, control_data2, sizeof(control_data2));
    snprintf(buf, sizeof(buf), "(One-shot Enable) HAL_I2C_Master_Transmit_IT: status: %u\r\n", status);
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    while(0 == irq_complete)
    {
        HAL_Delay(1000);
    }

    irq_complete = 0;
    // Wait for conversion complete
    uint8_t status_reg = 0x27;
    int count = 0;
    while(count < 10)
    {
        // Send read status register sub command
        status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, &status_reg, sizeof(status_reg));
        snprintf(buf, sizeof(buf), "[%d] (status_reg) HAL_I2C_Master_Transmit_IT: status: %u\r\n", count, status);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
        while(0 == irq_complete)
        {
            HAL_Delay(1000);
        }

        irq_complete = 0;
        status_flag = 1;
        // Read conversion status
        status = HAL_I2C_Master_Receive_IT(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&status_data, sizeof(status_data));
        while(0 == irq_complete)
        {
            HAL_Delay(1000);
```

Step 16. User Story 4: I code in the way that when we use Rx or Tx interrupt mode, we check if the irq is complete or not. If it is complete, we can run to the next line of code

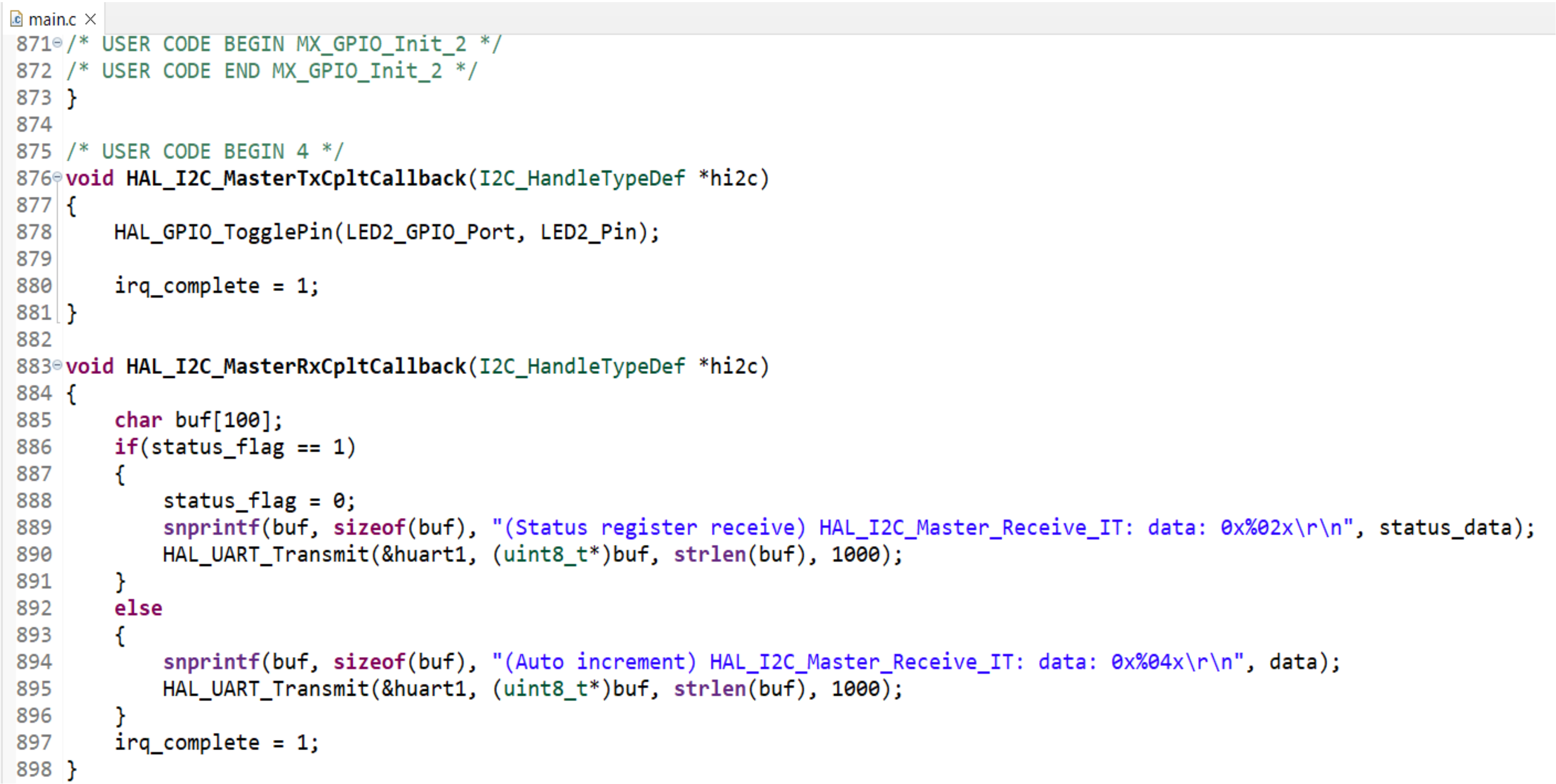


```c
    // Wait for conversion complete
    uint8_t status_reg = 0x27;
    int count = 0;
    while(count < 10)
    {
        // Send read status register sub command
        status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, &status_reg, sizeof(status_reg));
        snprintf(buf, sizeof(buf), "[%d] (status_reg) HAL_I2C_Master_Transmit_IT: status: %u\r\n", count, status);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
        while(0 == irq_complete)
        {
            HAL_Delay(1000);
        }

        irq_complete = 0;
        status_flag = 1;
        // Read conversion status
        status = HAL_I2C_Master_Receive_IT(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&status_data, sizeof(status_data));
        while(0 == irq_complete)
        {
            HAL_Delay(1000);
        }

        // Check for temperature conversion complete
        if(status_data & 0x1)
        {
            snprintf(buf, sizeof(buf), "New data available!\r\n");
            HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
            break;
        }
        HAL_Delay(1000);
        count++;
    }

```



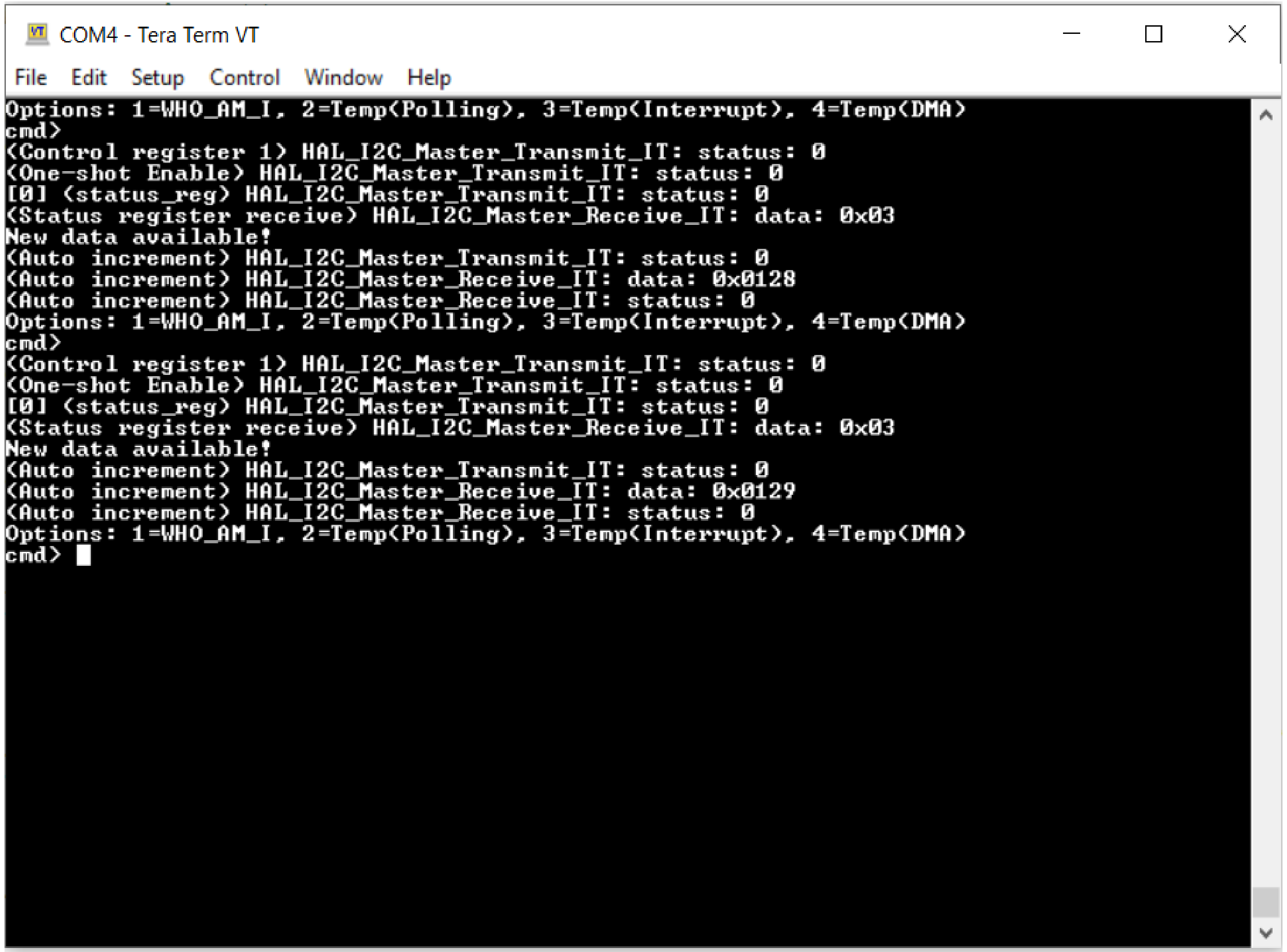
```c
            HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
            break;
        }
        HAL_Delay(1000);
        count++;
    }


    irq_complete = 0;
    // Read using auto increment
    uint8_t temperature_lsb = 0x2a | 0x80;
    status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb));
    snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Transmit_IT: status: %u\r\n", status);
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    while(0 == irq_complete)
    {
        HAL_Delay(1000);
    }

    irq_complete = 0;
    // Receive using interrupt
    status = HAL_I2C_Master_Receive_IT(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data, sizeof(data));
    snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive_IT: status: %u\r\n", status);
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    while(0 == irq_complete)
    {
        HAL_Delay(1000);
    }
}
```

Step 17. User Story 4: Tx and Rx call back function implementation



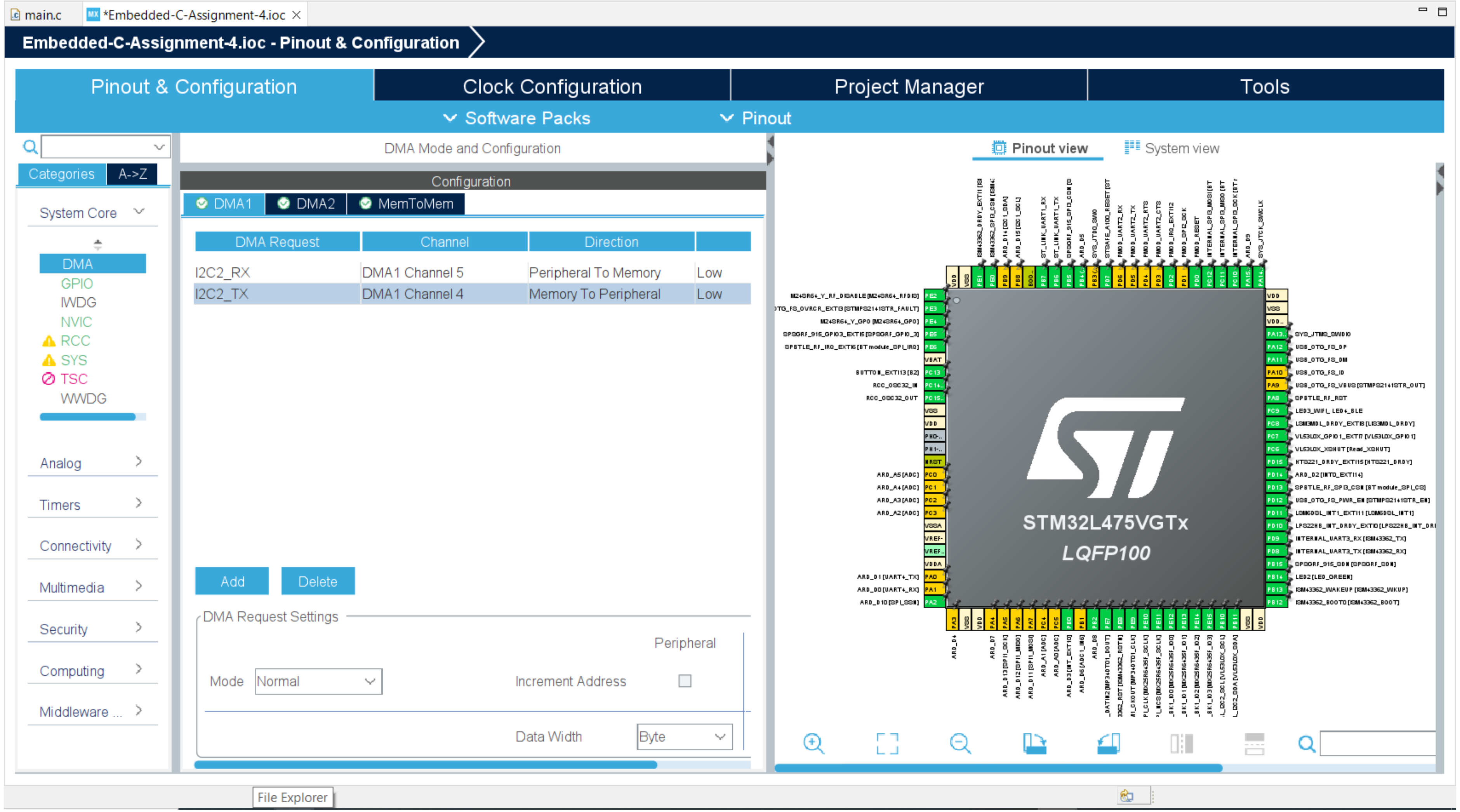
```c
/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void HAL_I2C_MasterTxCpltCallback(I2C_HandleTypeDef *hi2c)
{
    HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);

    irq_complete = 1;
}

void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c)
{
    char buf[100];
    if(status_flag == 1)
    {
        status_flag = 0;
        snprintf(buf, sizeof(buf), "(Status register receive) HAL_I2C_Master_Receive_IT: data: 0x%02x\r\n", status_data);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    }
    else
    {
        snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive_IT: data: 0x%04x\r\n", data);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    }
    irq_complete = 1;
}
```

Step 18. User Story 4: Build and run the code, test is successful. I use auto increment to read the temperature data



```
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
(Control register 1) HAL_I2C_Master_Transmit_IT: status: 0
(One-shot Enable) HAL_I2C_Master_Transmit_IT: status: 0
[0] (status_reg) HAL_I2C_Master_Transmit_IT: status: 0
(Status register receive) HAL_I2C_Master_Receive_IT: data: 0x03
New data available!
(Auto increment) HAL_I2C_Master_Transmit_IT: status: 0
(Auto increment) HAL_I2C_Master_Receive_IT: data: 0x0128
(Auto increment) HAL_I2C_Master_Receive_IT: status: 0
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
(Control register 1) HAL_I2C_Master_Transmit_IT: status: 0
(One-shot Enable) HAL_I2C_Master_Transmit_IT: status: 0
[0] (status_reg) HAL_I2C_Master_Transmit_IT: status: 0
(Status register receive) HAL_I2C_Master_Receive_IT: data: 0x03
New data available!
(Auto increment) HAL_I2C_Master_Transmit_IT: status: 0
(Auto increment) HAL_I2C_Master_Receive_IT: data: 0x0129
(Auto increment) HAL_I2C_Master_Receive_IT: status: 0
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
```

Step 19. User Story 5. Temperature. DMA. When the user selects 4, use DMA to read from HTS221 and display temperature on console. Modify the DMA1 in .ioc file first



DMA Request	Channel	Direction	
I2C2_RX	DMA1 Channel 5	Peripheral To Memory	Low
I2C2_TX	DMA1 Channel 4	Memory To Peripheral	Low

DMA Request Settings

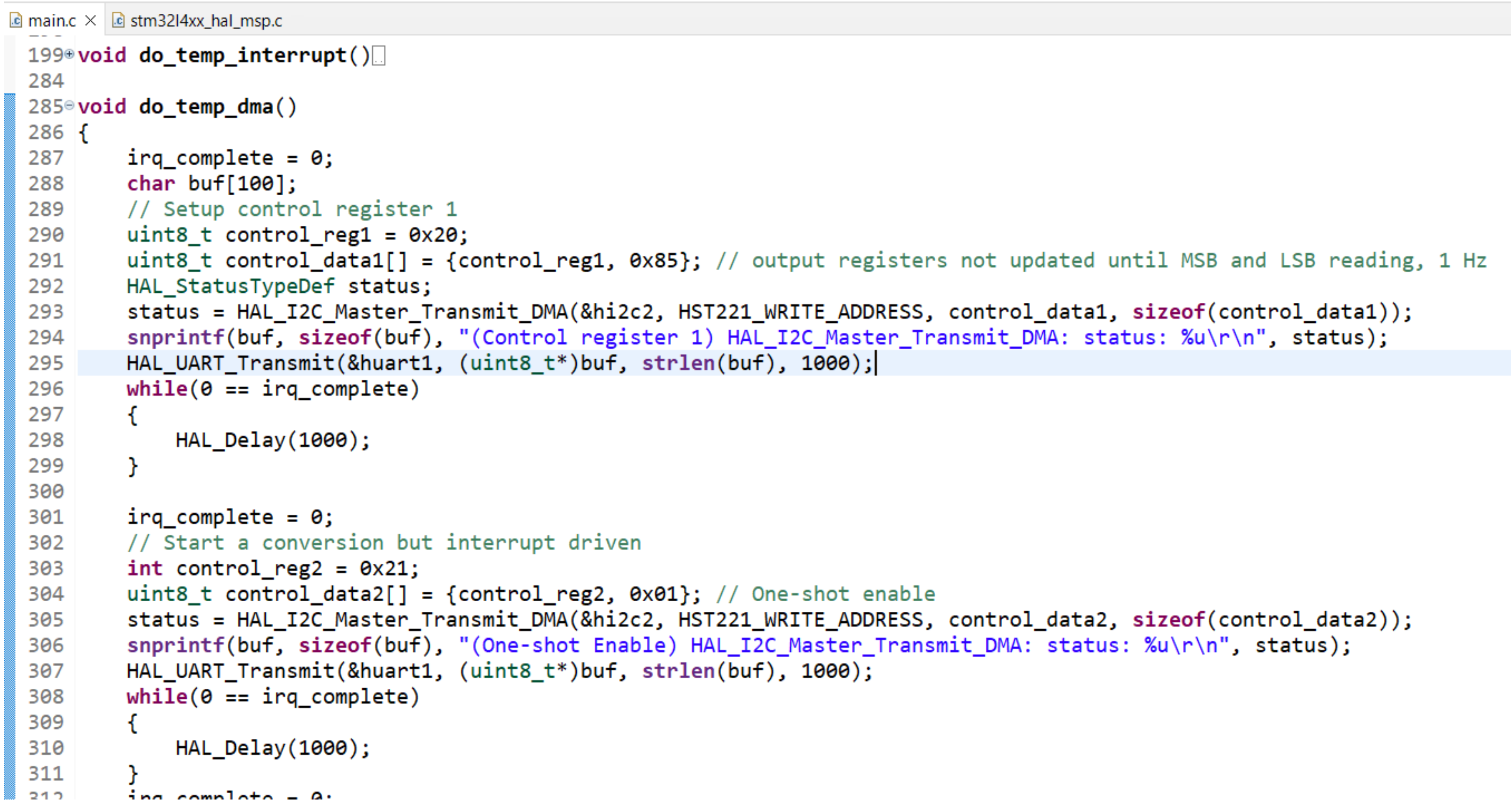
Mode: Normal

Peripheral

Increment Address □

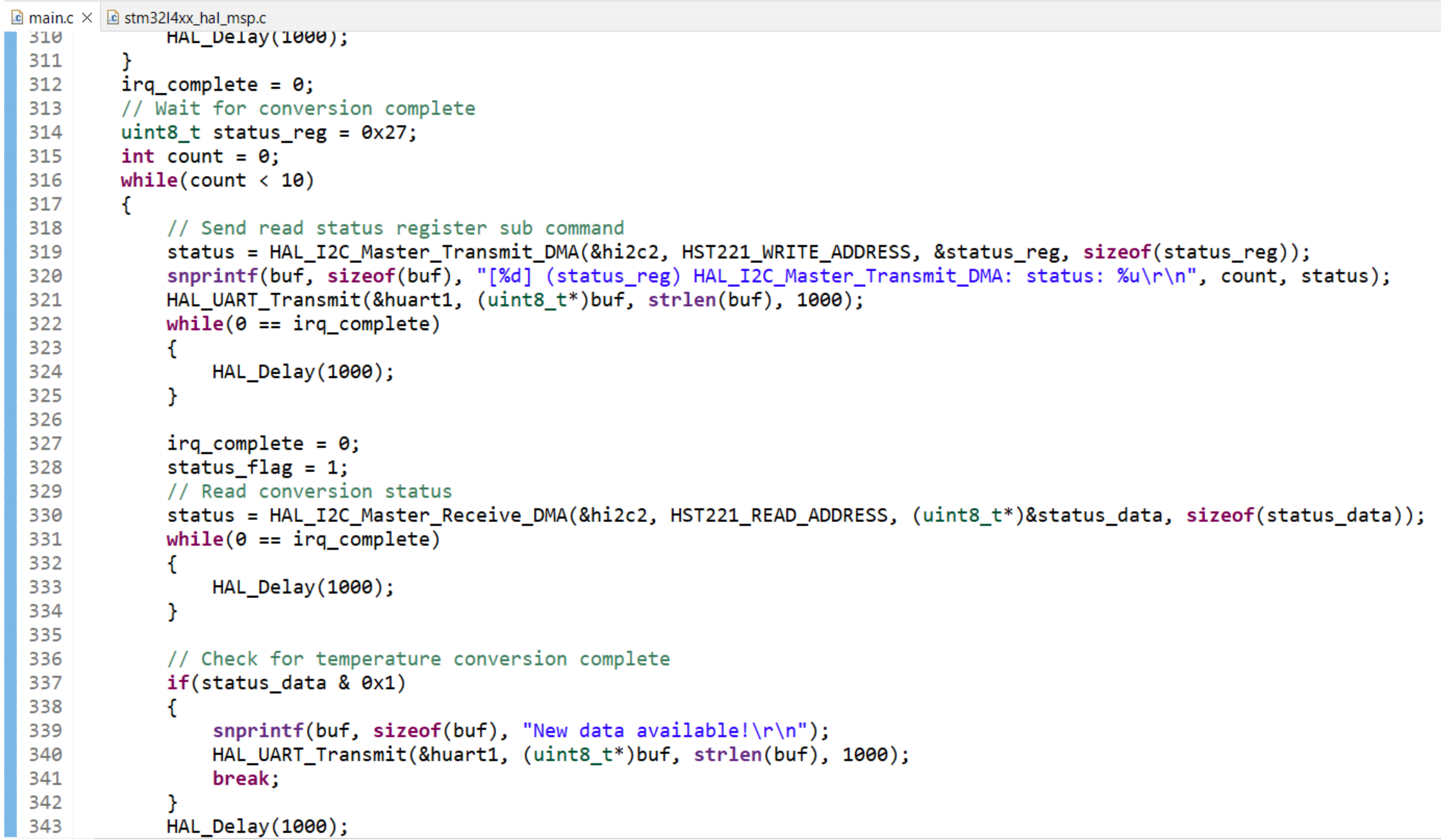
Data Width: Byte

Step 20. User Story 5. Write the code in main.c file, it is very similar to the interrupt version



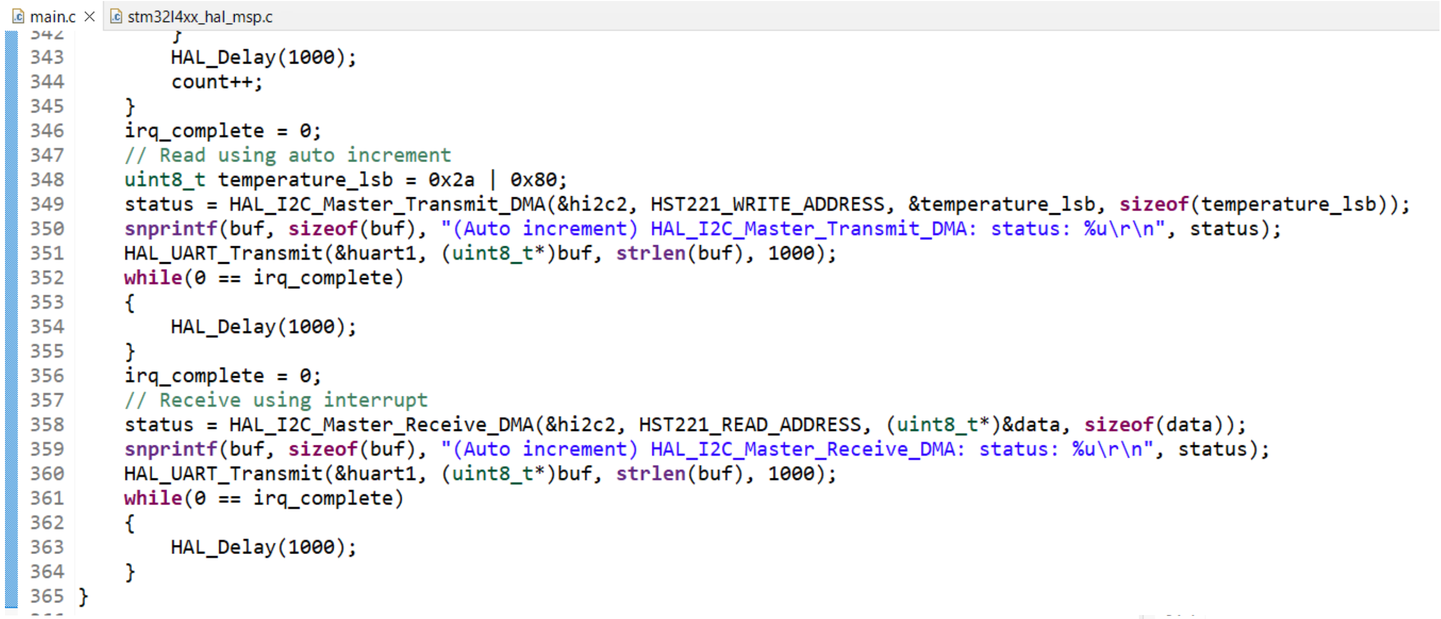
```c
void do_temp_interrupt()

void do_temp_dma()
{
    irq_complete = 0;
    char buf[100];
    // Setup control register 1
    uint8_t control_reg1 = 0x20;
    uint8_t control_data1[] = {control_reg1, 0x85}; // output registers not updated until MSB and LSB reading, 1 Hz
    HAL_StatusTypeDef status;
    status = HAL_I2C_Master_Transmit_DMA(&hi2c2, HST221_WRITE_ADDRESS, control_data1, sizeof(control_data1));
    snprintf(buf, sizeof(buf), "(Control register 1) HAL_I2C_Master_Transmit_DMA: status: %u\r\n", status);
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    while(0 == irq_complete)
    {
        HAL_Delay(1000);
    }

    irq_complete = 0;
    // Start a conversion but interrupt driven
    int control_reg2 = 0x21;
    uint8_t control_data2[] = {control_reg2, 0x01}; // One-shot enable
    status = HAL_I2C_Master_Transmit_DMA(&hi2c2, HST221_WRITE_ADDRESS, control_data2, sizeof(control_data2));
    snprintf(buf, sizeof(buf), "(One-shot Enable) HAL_I2C_Master_Transmit_DMA: status: %u\r\n", status);
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    while(0 == irq_complete)
    {
        HAL_Delay(1000);
    }
    irq_complete = 0;
```

Step 21. User Story 5. Write the code in main.c file, continue

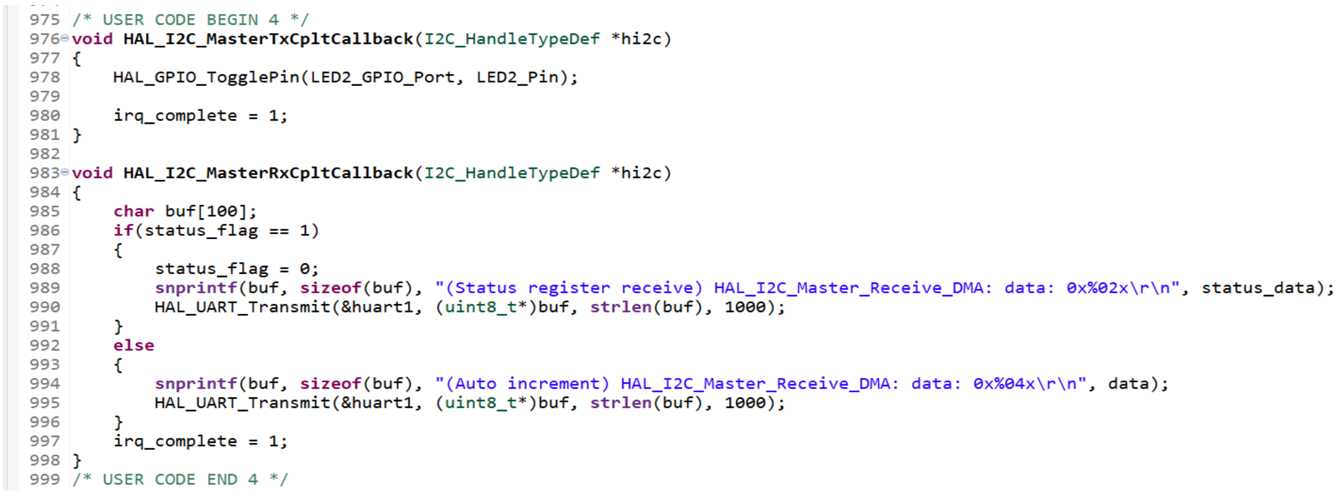


```c
        HAL_Delay(1000);
    }
    irq_complete = 0;
    // Wait for conversion complete
    uint8_t status_reg = 0x27;
    int count = 0;
    while(count < 10)
    {
        // Send read status register sub command
        status = HAL_I2C_Master_Transmit_DMA(&hi2c2, HST221_WRITE_ADDRESS, &status_reg, sizeof(status_reg));
        snprintf(buf, sizeof(buf), "[%d] (status_reg) HAL_I2C_Master_Transmit_DMA: status: %u\r\n", count, status);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
        while(0 == irq_complete)
        {
            HAL_Delay(1000);
        }

        irq_complete = 0;
        status_flag = 1;
        // Read conversion status
        status = HAL_I2C_Master_Receive_DMA(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&status_data, sizeof(status_data));
        while(0 == irq_complete)
        {
            HAL_Delay(1000);
        }

        // Check for temperature conversion complete
        if(status_data & 0x1)
        {
            snprintf(buf, sizeof(buf), "New data available!\r\n");
            HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
            break;
        }
        HAL_Delay(1000);
```

Step 22. User Story 5. Write the code in main.c file, continue

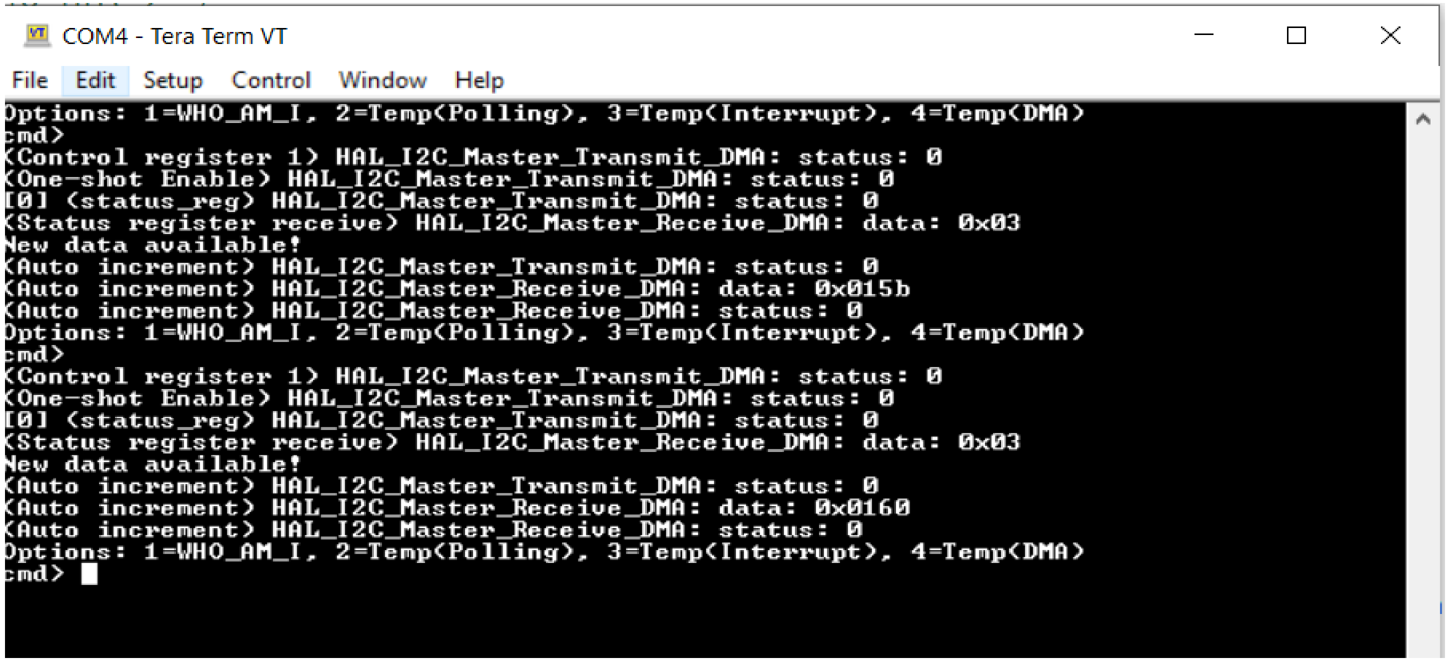


```c
        }
        HAL_Delay(1000);
        count++;
    }
    irq_complete = 0;
    // Read using auto increment
    uint8_t temperature_lsb = 0x2a | 0x80;
    status = HAL_I2C_Master_Transmit_DMA(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb));
    snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Transmit_DMA: status: %u\r\n", status);
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    while(0 == irq_complete)
    {
        HAL_Delay(1000);
    }
    irq_complete = 0;
    // Receive using interrupt
    status = HAL_I2C_Master_Receive_DMA(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data, sizeof(data));
    snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive_DMA: status: %u\r\n", status);
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    while(0 == irq_complete)
    {
        HAL_Delay(1000);
    }
}
```



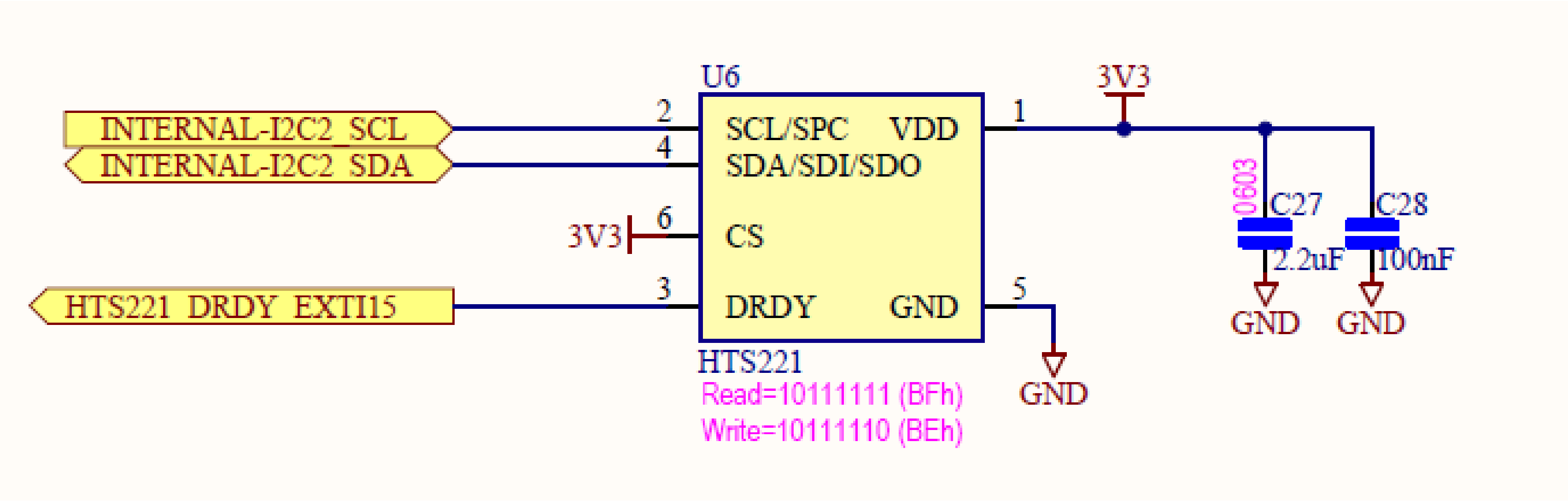
```c
/* USER CODE BEGIN 4 */
void HAL_I2C_MasterTxCpltCallback(I2C_HandleTypeDef *hi2c)
{
    HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);

    irq_complete = 1;
}

void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c)
{
    char buf[100];
    if(status_flag == 1)
    {
        status_flag = 0;
        snprintf(buf, sizeof(buf), "(Status register receive) HAL_I2C_Master_Receive_DMA: data: 0x%02x\r\n", status_data);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    }
    else
    {
        snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive_DMA: data: 0x%04x\r\n", data);
        HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    }
    irq_complete = 1;
}
/* USER CODE END 4 */
```

Step 23. User Story 5. Build and run the code, test is successful



```
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
<Control register 1> HAL_I2C_Master_Transmit_DMA: status: 0
<One-shot Enable> HAL_I2C_Master_Transmit_DMA: status: 0
[0] (status_reg) HAL_I2C_Master_Transmit_DMA: status: 0
<Status register receive> HAL_I2C_Master_Receive_DMA: data: 0x03
New data available!
<Auto increment> HAL_I2C_Master_Transmit_DMA: status: 0
<Auto increment> HAL_I2C_Master_Receive_DMA: data: 0x015b
<Auto increment> HAL_I2C_Master_Receive_DMA: status: 0
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
<Control register 1> HAL_I2C_Master_Transmit_DMA: status: 0
<One-shot Enable> HAL_I2C_Master_Transmit_DMA: status: 0
[0] (status_reg) HAL_I2C_Master_Transmit_DMA: status: 0
<Status register receive> HAL_I2C_Master_Receive_DMA: data: 0x03
New data available!
<Auto increment> HAL_I2C_Master_Transmit_DMA: status: 0
<Auto increment> HAL_I2C_Master_Receive_DMA: data: 0x0160
<Auto increment> HAL_I2C_Master_Receive_DMA: status: 0
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
```

Appendix, schematic for HTS221


INTERNAL-I2C2_SCL
INTERNAL-I2C2_SDA
3V3
HTS221_DRDY_EXTI15
U6
2
4
6
3
SCL/SPC
SDA/SDI/SDO
CS
DRDY
VDD
GND
1
5
HTS221
Read=10111111 (BFh)
Write=10111110 (BEh)
GND
3V3
0603
C27
2.2uF
C28
100nF
GND
GND