life.augmented

DT0141
Design tip

How to import STMems_Standard_C_drivers in an STM32CubeIDE project

M. Galizzi

Main components	
STM32CubeIDE 1.3.0+	Integrated Development Environment for STM32
SensorTile.box	Wireless multi-sensor development kit (STEVAL-MKSBOX1V1)
STMems_Standard_C_drivers	Platform-independent driver source code for MEMS

Purpose and benefits

This design tip explains how to import and use the platform-independent driver source code for STMicroelectronics MEMS motion and environmental sensors, freely available on STMicroelectronics GitHub.

- The drivers are written in standard C and can run on any microcontroller.
- The implementation shown uses a limited code abstraction in order to maintain the code easy to understand, read and debug.

Description

In this example a set of drivers will be included in a new project for the STEVAL-MKSBOX1V1 platform to show how these drivers can be easily added. In our case both I^2C and SPI buses will be exploited.

On the STEVAL-MKSBOX1V1 **I2C1 bus**, the following sensors are connected:

1. HTS221 humidity and temperature sensor (hts221_STdC driver)
2. LPS22HH pressure and temperature sensor (lps22hh_STdC driver)

On the **I2C3 bus**, one sensor is connected:

1. STTS751 temperature sensor (stts751_STdC driver)

On the **SPI1 bus**, these sensors are connected:

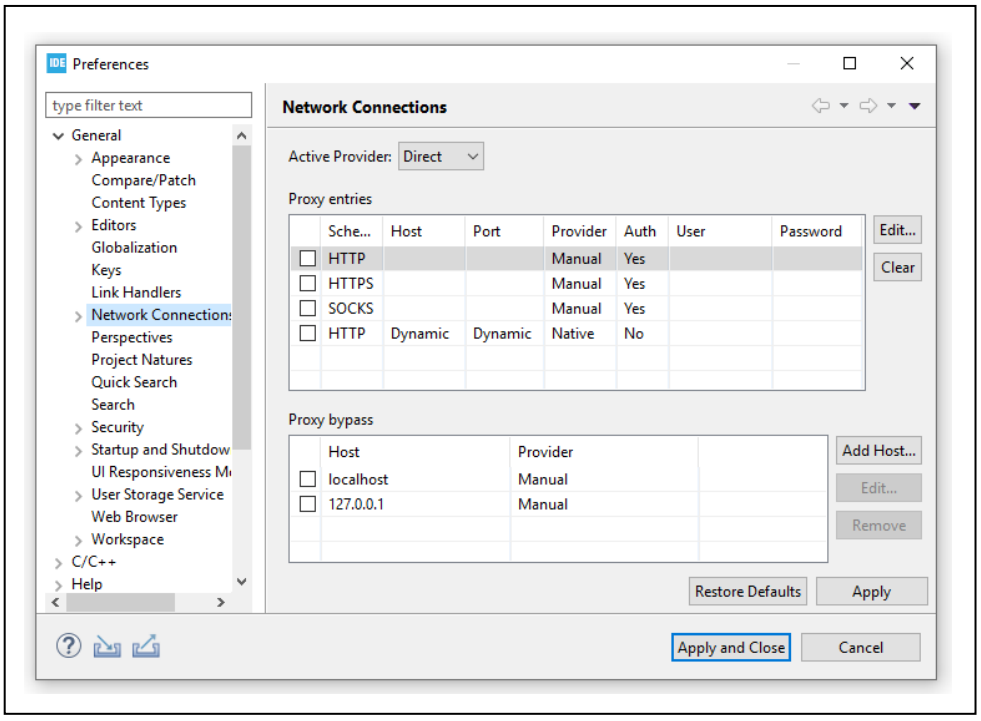
1. LSM6DSOX iNEMO inertial module (lsm6dsox_STdC driver)
2. IIS3DHHC 3-axis industrial digital inclinometer (iis3dhhc_STdC driver)
3. LIS2DW12 3-axis MEMS accelerometer (lis2dw12_STdC driver)

Step 0: Network configuration

The STM32CubeIDE will always try to access the network connection to check for updates. If the network settings are not configured, or if they are configured but the network is not operative, a pop-up dialog will appear, asking to configure the network settings. You can cancel and postpone the configuration until the IDE needs to create the code for the project; that is when you insert the name of the project, press finish, and accept the default configuration for all peripherals; at this point the IDE really needs the correct software package (STM32CubeL4 in our case) to be able to generate the code. Either the package is already installed, or it must be downloaded from the network. In the latter case you need to configure the network settings.

Network settings can be configured at any time: click on *Window* > *Preferences* to open a window as in Figure 1. In the left panel, click on *General* to expand the tree, then *Network Connections*. The default mode is “Native” which means the STM32CubeIDE will use the same settings as the default browser in the system. This is usually fine. If you use a VPN, behind a proxy, you may need to select “Manual”, and configure all three protocols: HTTP, HTTPS, and SOCKS. Select the protocol to be configured, click Edit, insert the IP address of the proxy and the port (usually 80). If needed, select the checkbox for “required authentication” and insert the user name and password. When everything is configured, click Apply and Close.

Figure 1. Configure network connections

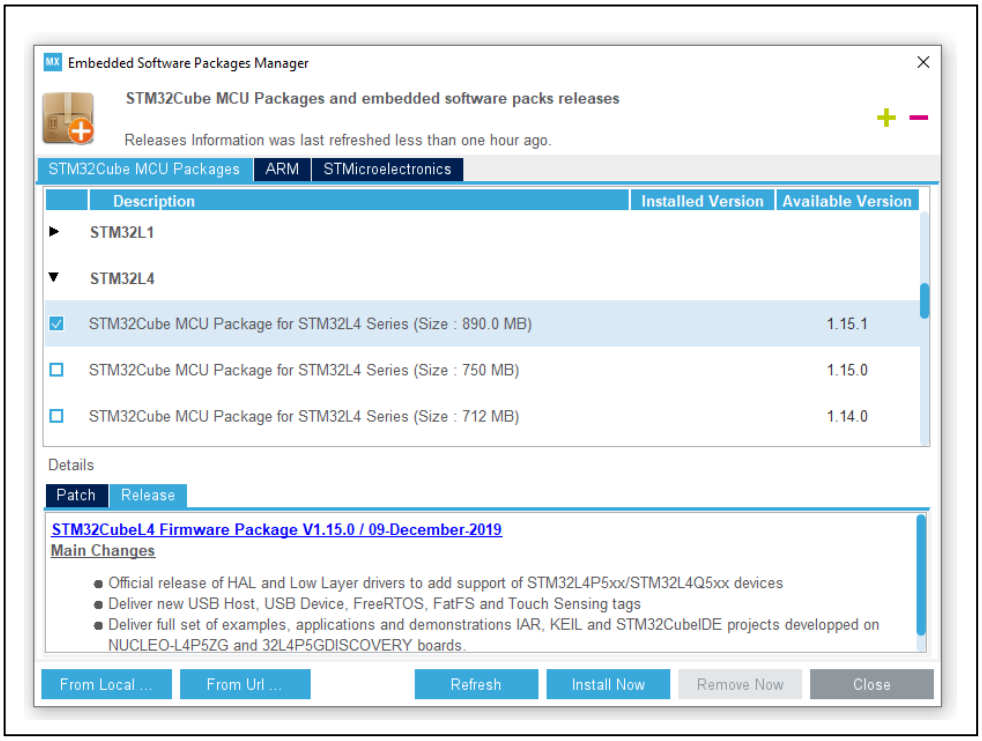


Step 1: Installing the latest Cube library

The first step after setting up the network connection consists of installing or updating the library of the microcontroller used in this project. The microcontroller of the STEVAL-MKSBOX1V1 is an STM32L4R9, so STM32CubeL4 libraries are necessary.

In STM32CubeIDE, click on *Help* > *Manage embedded software package* and a window should appear as in Figure 2. In the *STM32Cube MCU Packages* tab, expand **STM32L4** and check to select the latest **STM32Cube MCU Package for STM32L4 Series**, then click on *Install Now.*

Figure 2. Install the latest STM32CubeL4 libraries

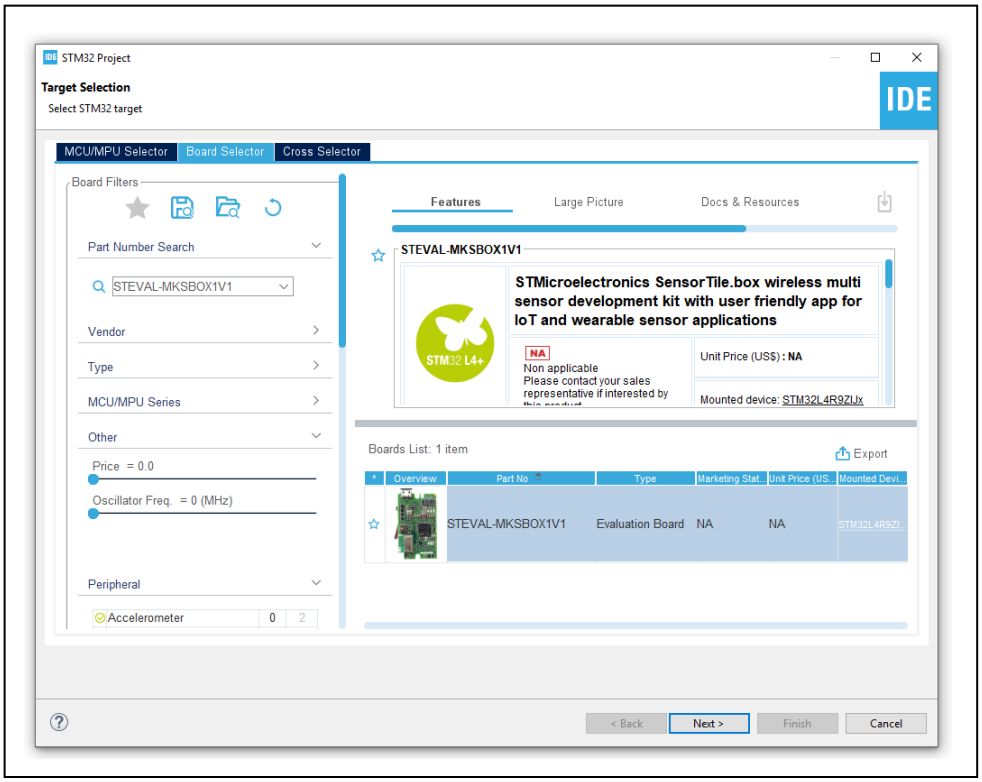


Let the STM32CubeIDE download the necessary libraries and when it is completed, close the **Embedded Software Packages Manager**.

Step 2: Generate an empty project for STEVAL-MKSBOX1V1

Let's begin creating a new empty project for the desired platform. In STM32CubeIDE select *File* > *New* > *STM32 Project* and in the *Board Selector* tab search for the part number **STEVAL-MKSBOX1V1.** Click on the image showing the board and then click on *Next* > as depicted in Figure 3.

Figure 3. Create a new project for STEVAL-MKSBOX1V1 platform



At this point a window will open to set up the new project name. Assign any name and then click on the *Finish* button, as depicted in Figure 4.

Figure 4. New project name for STEVAL-MKSBOX1V1

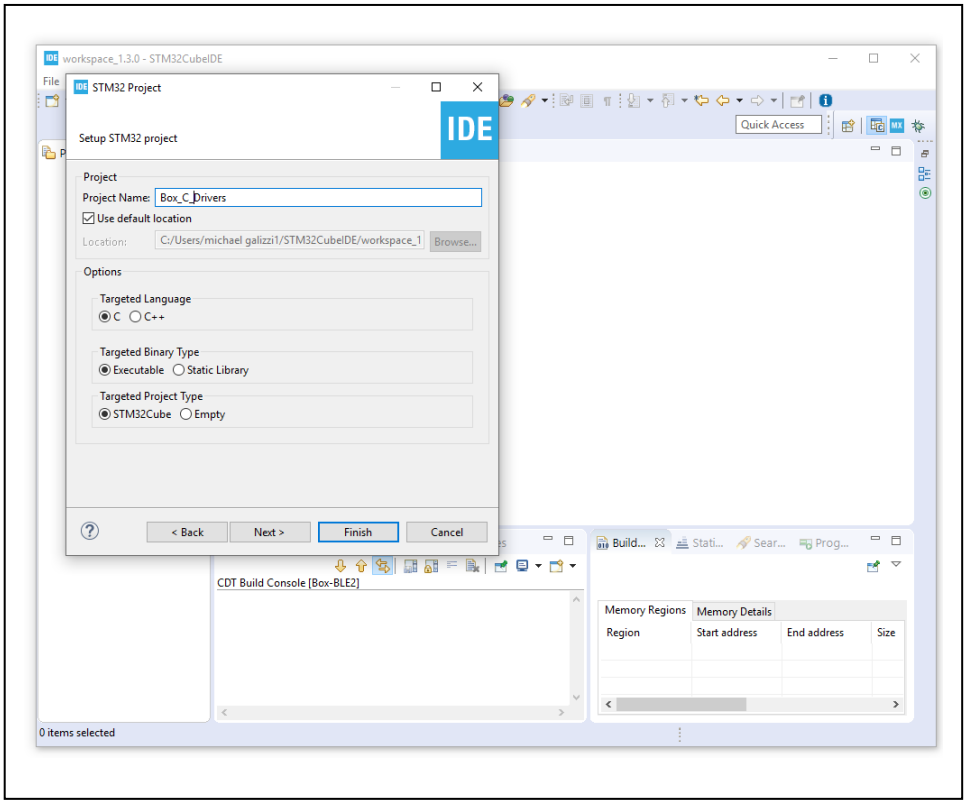
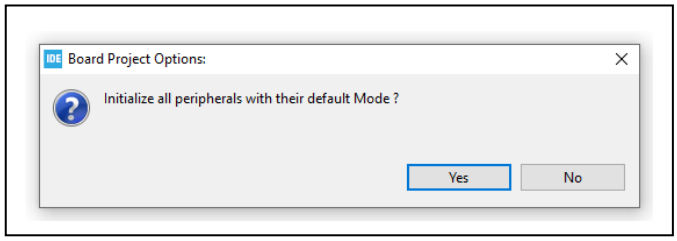
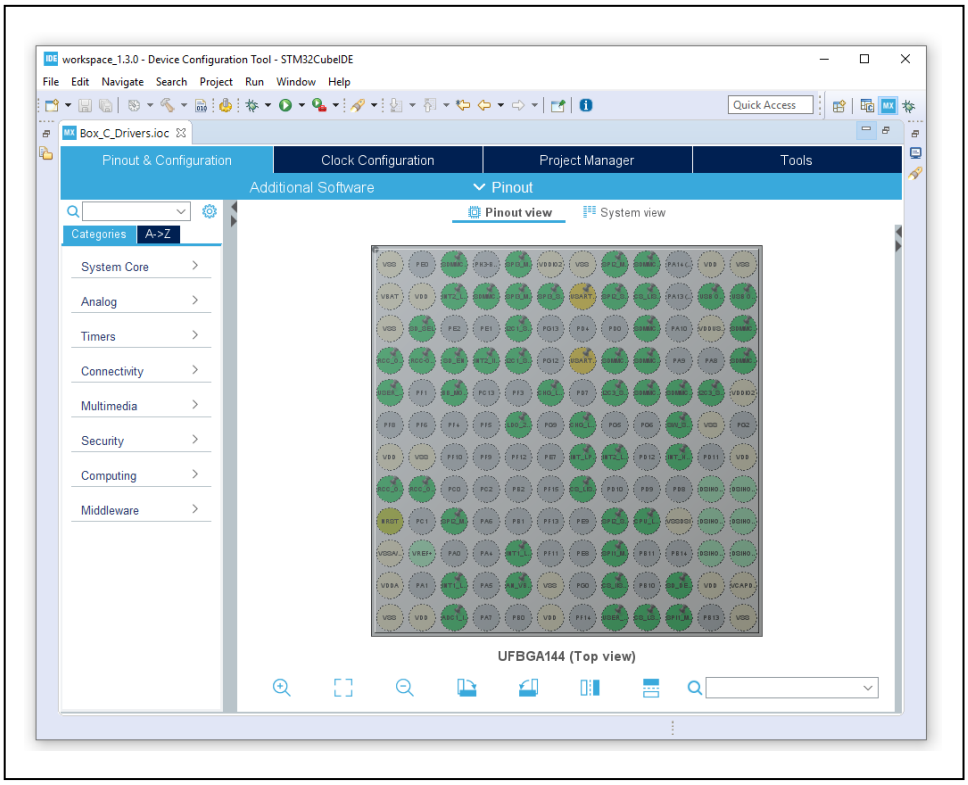


Figure 5. Confirm default initialization



After confirming the default peripherals initialization as depicted in Figure 5, the STM32CubeIDE will show the pinout of the microcontroller with all the needed peripherals already configured, as shown in Figure 6.

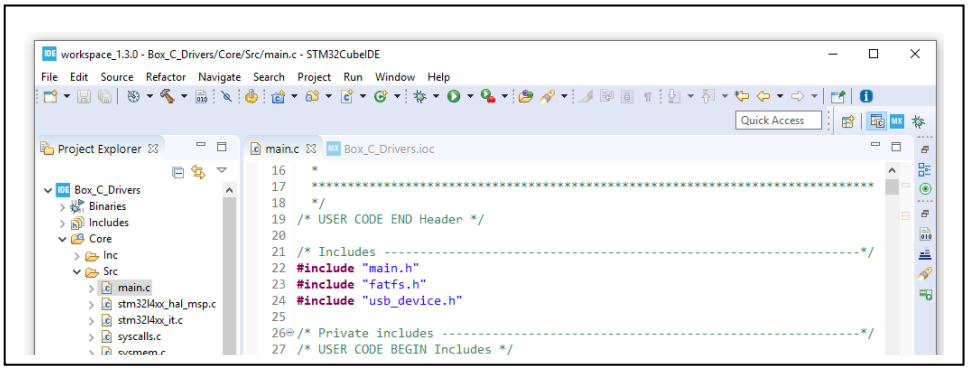
Figure 6. Pinout of the STEVAL-MKSBOX1V1 microcontroller



Nothing else needs to be done to configure the microcontroller for the purpose of this design tip; all the peripherals will work out-of-the-box with the default initialization. Switch to the C/C++ editor perspective, by clicking the  button situated on the top right of the STM32CubeIDE.

In the Project Explorer tab, double click on the “main.c” file located in the folder *Core/Src*, as shown in Figure 7.

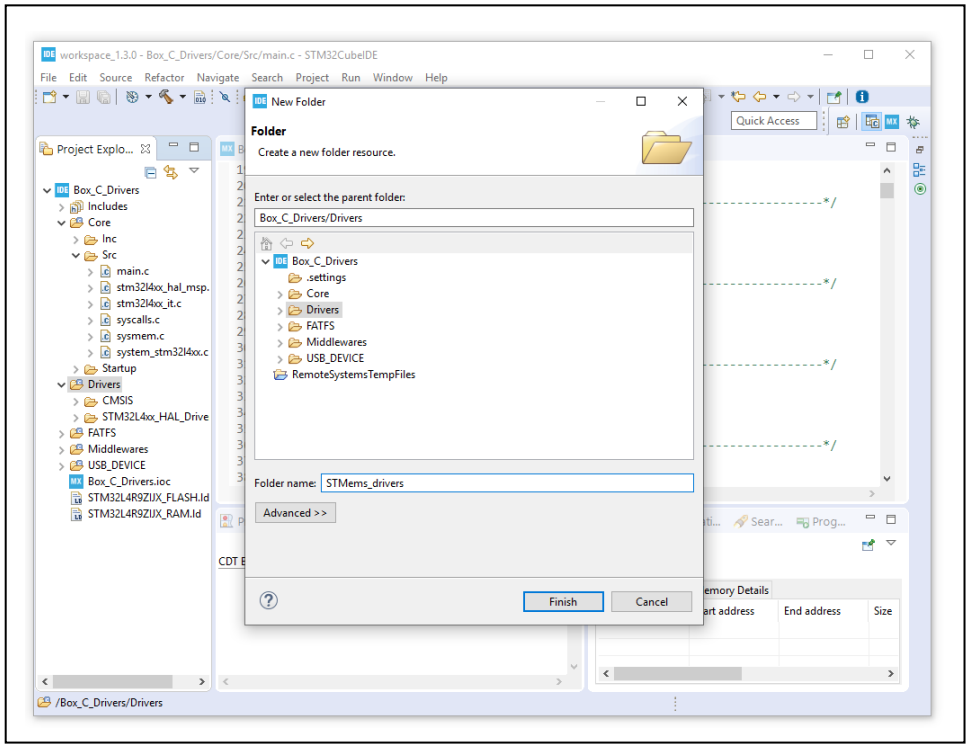
Figure 7. Open main.c file from Project Explorer tab



Step 3: Importing the driver files in the project

Create a new folder to put all the source files by right clicking on the *Drivers* folder and select *New* > *Folder*.
As shown in Figure 8, name the folder as “STMems_drivers” and click on *Finish*.

Figure 8. Drivers folder creation



All the source and header files of the sensors need to be downloaded from STMicroelectronics GitHub, and all those sources need to be copied in the driver folder previously created:

GitHub raw files of the HTS221 sensor are “hts221_reg.c” and “hts221_reg.h”

GitHub raw files of the LPS22HH sensor are “lps22hh_reg.c” and “lps22hh_reg.h”

GitHub raw files of the STTS751 sensor are “stts751_reg.c” and “stts751_reg.h”

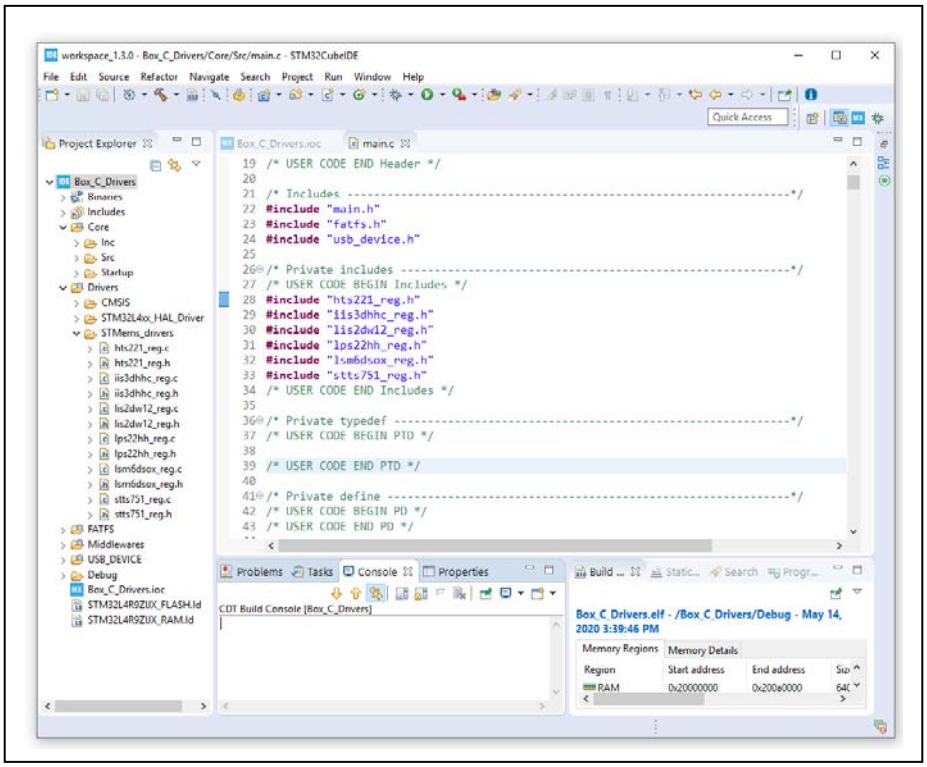
GitHub raw files of the LSM6DSOX sensor are “lsm6dsox_reg.c” and “lsm6dsox_reg.h”

GitHub raw files of the IIS3DHHC sensor are “iis3dhhc_reg.c” and “iis3dhhc_reg.h”

GitHub raw files of the LIS2DW12 sensor are “lis2dw12_reg.c” and “lis2dw12_reg.h”

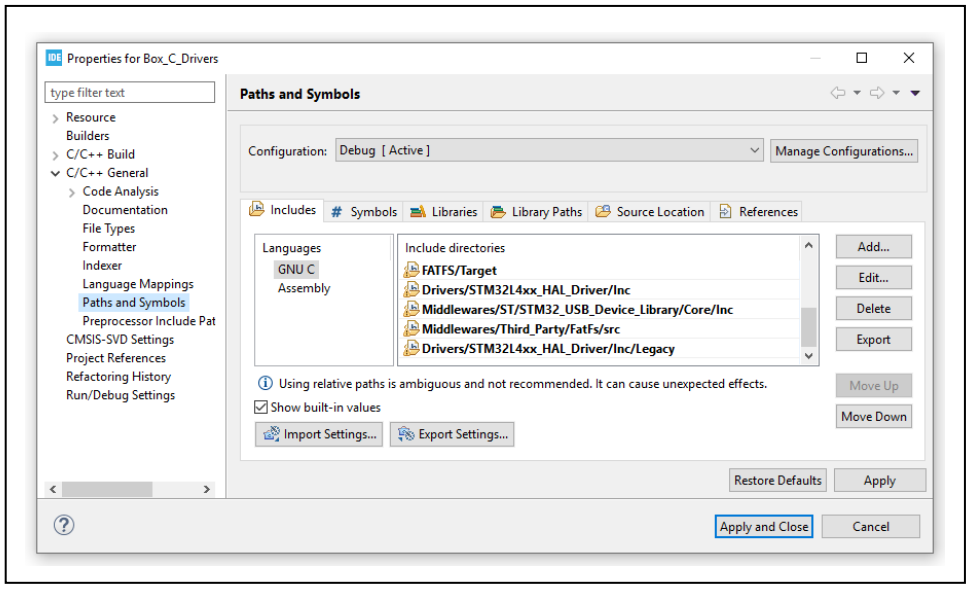
Update the driver folder by clicking on *File* > *Refresh*. The project directory structure should look like the one shown in Figure 9.

Figure 9. Folders containing GitHub ST Mems source drivers



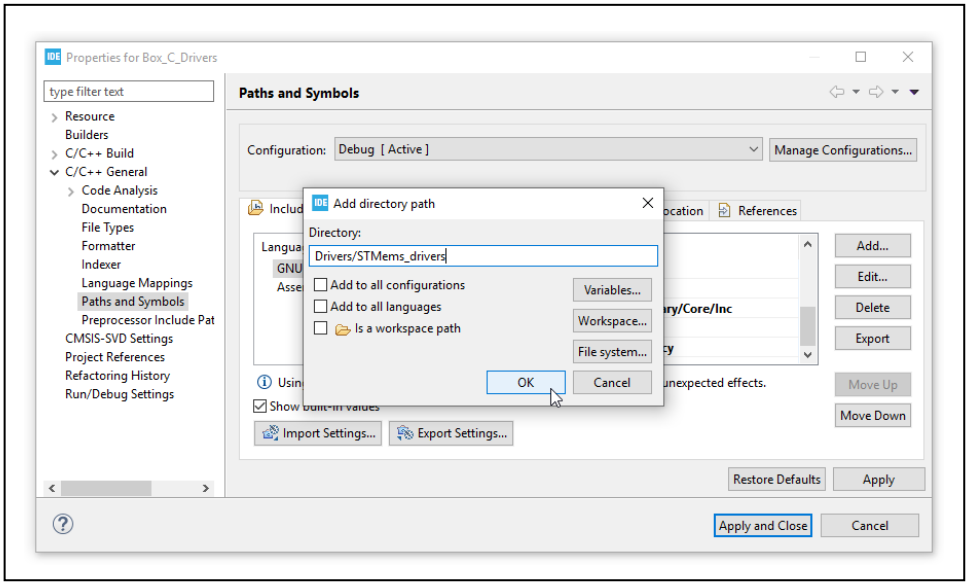
The next step is to add the “STMems_drivers” folder in the included path; click on *Project* > *Properties* and a window as in Figure 10 should pop up.

Figure 10. Add driver folder in included path



In the “C/C++ General > Path and Symbols” tab, click on the “*Add…*” button and insert the directory path “*Drivers/STMems_drivers*”. Click “*Ok*” and “*Apply and Close*”, as shown in Figure 11.

Figure 11. Add directory path



Step 4: Configuring the sensors

At this point it is possible to include the drivers in the project source code and use them. For simplicity, all the snippets of code will be added in the “main.c” file. IMPORTANT: While editing the “main.c” source file, make sure to edit it only in between the `/* USER CODE BEGIN … */` and `/* USER CODE END … */` comments in order to preserve the capability of the STM32CubeIDE to regenerate the code without deleting the portions of code manually added.

main.c : Adding included files

```
/* USER CODE BEGIN Includes */
#include "hts221_reg.h"
#include "iis3dhhc_reg.h"
#include "lis2dw12_reg.h"
#include "lps22hh_reg.h"
#include "lsm6dsox_reg.h"
#include "stts751_reg.h"
/* USER CODE END Includes */
```

main.c : Adding private macros

```
/* USER CODE BEGIN PM */
typedef struct {
          float x0;
          float y0;
          float x1;
          float y1;
} lin_t;

float linear_interpolation(lin_t *lin, int16_t x) {
          return ((lin->y1 - lin->y0) * x + ((lin->x1 * lin->y0) - (lin->x0 * lin->y1))) / (lin->x1 - lin->x0);
}
/* USER CODE END PM */
```

main.c : Adding driver private variables

```
/* USER CODE BEGIN PV */
static uint8_t whoamI, rst;
lin_t lin_hum;
lin_t lin_temp;
stmdev_ctx_t hts221Driver;
stmdev_ctx_t lps22hhDriver;
stmdev_ctx_t stts751Driver;
stmdev_ctx_t lsm6dsoxDriver;
stmdev_ctx_t iis3dhhcDriver;
stmdev_ctx_t lis2dw12Driver;
/* USER CODE END PV */
```

main.c : Adding private function prototype of read and write sensors

```
/* USER CODE BEGIN PFP */
HAL_StatusTypeDef Init_HTS221(void);
HAL_StatusTypeDef Init_LPS22HH(void);
HAL_StatusTypeDef Init_STTS751(void);
HAL_StatusTypeDef Init_LSM6DSOX(void);
HAL_StatusTypeDef Init_IIS3DHHC(void);
HAL_StatusTypeDef Init_LIS2DW12(void);
static int32_t hts221_read(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len);
static int32_t hts221_write(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len);
static int32_t lps22hh_read(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len);
static int32_t lps22hh_write(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len);
static int32_t stts751_read(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len);
static int32_t stts751_write(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len);
static int32_t lsm6dsox_read(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len);
static int32_t lsm6dsox_write(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len);
static int32_t iis3dhhc_read(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len);
static int32_t iis3dhhc_write(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len);
static int32_t lis2dw12_read(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len);
static int32_t lis2dw12_write(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len);
/* USER CODE END PFP */
```

main.c : Initializing the driver variables and adding call to initialize functions

```
/* USER CODE BEGIN 2 */
hts221Driver.handle = &hi2c1;
hts221Driver.read_reg = hts221_read;
hts221Driver.write_reg = hts221_write;

lps22hhDriver.handle = &hi2c1;
lps22hhDriver.read_reg = lps22hh_read;
lps22hhDriver.write_reg = lps22hh_write;

stts751Driver.handle = &hi2c3;
stts751Driver.read_reg = stts751_read;
stts751Driver.write_reg = stts751_write;

lsm6dsoxDriver.handle = &hspi1;
lsm6dsoxDriver.read_reg = lsm6dsox_read;
lsm6dsoxDriver.write_reg = lsm6dsox_write;

iis3dhhcDriver.handle = &hspi1;
iis3dhhcDriver.read_reg = iis3dhhc_read;
iis3dhhcDriver.write_reg = iis3dhhc_write;

lis2dw12Driver.handle = &hspi1;
lis2dw12Driver.read_reg = lis2dw12_read;
lis2dw12Driver.write_reg = lis2dw12_write;

Init_HTS221();
Init_LPS22HH();
Init_STTS751();
Init_LSM6DSOX();
Init_IIS3DHHC();
Init_LIS2DW12();
/* USER CODE END 2 */
```

main.c : Implementing the initialization function for each sensor 1/4

```
/* USER CODE BEGIN 4 */
/* Initialize Humidity sensor */
HAL_StatusTypeDef Init_HTS221(void) {
	/* Check device ID for Humidity sensor */
	whoamI = 0;
	hts221_device_id_get(&hts221Driver, &whoamI);
	/* Configure Humidity sensor */
	if (whoamI == HTS221_ID) {
		/* Enable Block Data Update */
		hts221_block_data_update_set(&hts221Driver, PROPERTY_ENABLE);
		/* Set Output Data Rate */
		hts221_data_rate_set(&hts221Driver, HTS221_ODR_12Hz5);
		hts221_humidity_avg_set(&hts221Driver, HTS221_H_AVG_4);
		hts221_temperature_avg_set(&hts221Driver, HTS221_T_AVG_2);
		/* Device Power On */
		hts221_power_on_set(&hts221Driver, PROPERTY_ENABLE);
		/* Read humidity calibration coefficient */
		hts221_hum_adc_point_0_get(&hts221Driver, &lin_hum.x0);
		hts221_hum_rh_point_0_get(&hts221Driver, &lin_hum.y0);
		hts221_hum_adc_point_1_get(&hts221Driver, &lin_hum.x1);
		hts221_hum_rh_point_1_get(&hts221Driver, &lin_hum.y1);
		/* Read temperature calibration coefficient */
		hts221_temp_adc_point_0_get(&hts221Driver, &lin_temp.x0);
		hts221_temp_deg_point_0_get(&hts221Driver, &lin_temp.y0);
		hts221_temp_adc_point_1_get(&hts221Driver, &lin_temp.x1);
		hts221_temp_deg_point_1_get(&hts221Driver, &lin_temp.y1);
		return HAL_OK;
	} else {
		return HAL_ERROR;
	}
}

/* Initialize Pressure sensor */
HAL_StatusTypeDef Init_LPS22HH(void) {
	/* Check device ID for Pressure sensor */
	whoamI = 0;
	lps22hh_device_id_get(&lps22hhDriver, &whoamI);
	/* Configure Pressure sensor */
	if (whoamI == LPS22HH_ID) {
		/* Restore default configuration */
		lps22hh_reset_set(&lps22hhDriver, PROPERTY_ENABLE);
		do {
			lps22hh_reset_get(&lps22hhDriver, &rst);
		} while (rst);
		/* Enable Block Data Update */
		lps22hh_block_data_update_set(&lps22hhDriver, PROPERTY_ENABLE);
		/* Set Output Data Rate */
		lps22hh_data_rate_set(&lps22hhDriver, LPS22HH_200_Hz);
		return HAL_OK;
	} else {
		return HAL_ERROR;
	}
}

/* Initialize Temperature sensor*/
HAL_StatusTypeDef Init_STTS751(void) {
	/* Check device ID for Temperature sensor */
	static stts751_id_t whoamItemp;
	stts751_device_id_get(&stts751Driver, &whoamItemp);
	/* Configure Temperature sensor */
	if (!((whoamItemp.product_id != STTS751_ID_0xxxx) || (whoamItemp.manufacturer_id != STTS751_ID_MAN))) {
		/* Set Output Data Rate */
		stts751_temp_data_rate_set(&stts751Driver, STTS751_TEMP_ODR_16Hz);
		/* Set Resolution */
		stts751_resolution_set(&stts751Driver, STTS751_11bit);
		return HAL_OK;
	} else {
		return HAL_ERROR;
	}
}
```

main.c : Implementing the initialization function for each sensor 2/4

```
/* Initialize Accelerometer and Gyroscope*/
HAL_StatusTypeDef Init_LSM6DSOX(void) {
	/* Check device ID for LSM6DSOX accelerometer */
	whoamI = 0;
	lsm6dsox_device_id_get(&lsm6dsoxDriver, &whoamI);
	/* Configure LSM6DSOX accelerometer */
	if (whoamI == LSM6DSOX_ID) {
		/* Restore default configuration */
		lsm6dsox_reset_set(&lsm6dsoxDriver, PROPERTY_ENABLE);
		do {
			lsm6dsox_reset_get(&lsm6dsoxDriver, &rst);
		} while (rst);
		/* Disable I3C interface */
		lsm6dsox_i3c_disable_set(&lsm6dsoxDriver, LSM6DSOX_I3C_DISABLE);
		/* Enable Block Data Update */
		lsm6dsox_block_data_update_set(&lsm6dsoxDriver, PROPERTY_ENABLE);
		/* Set Output Data Rate */
		lsm6dsox_xl_data_rate_set(&lsm6dsoxDriver, LSM6DSOX_XL_ODR_6667Hz);
		lsm6dsox_gy_data_rate_set(&lsm6dsoxDriver, LSM6DSOX_GY_ODR_6667Hz);
		/* Set full scale */
		lsm6dsox_xl_full_scale_set(&lsm6dsoxDriver, LSM6DSOX_2g);
		lsm6dsox_gy_full_scale_set(&lsm6dsoxDriver, LSM6DSOX_2000dps);
		/* Configure filtering chain */
		lsm6dsox_xl_hp_path_on_out_set(&lsm6dsoxDriver, LSM6DSOX_LP_ODR_DIV_100);
		lsm6dsox_xl_filter_lp2_set(&lsm6dsoxDriver, PROPERTY_ENABLE);
		return HAL_OK;
	} else {
		return HAL_ERROR;
	}
}

/* Initialize Accelerometer */
HAL_StatusTypeDef Init_IIS3DHHC(void) {
	/* Check device ID for IIS3DHHC accelerometer */
	whoamI = 0;
	iis3dhhc_device_id_get(&iis3dhhcDriver, &whoamI);
	/* Configure IIS3DHHC accelerometer */
	if (whoamI == IIS3DHHC_ID) {
		/* Restore default configuration */
		iis3dhhc_reset_set(&iis3dhhcDriver, PROPERTY_ENABLE);
		do {
			iis3dhhc_reset_get(&iis3dhhcDriver, &rst);
		} while (rst);
		/* Enable Block Data Update */
		iis3dhhc_block_data_update_set(&iis3dhhcDriver, PROPERTY_ENABLE);
		/* Set Output Data Rate */
		iis3dhhc_data_rate_set(&iis3dhhcDriver, IIS3DHHC_1kHz1);
		/* Enable temperature compensation */
		iis3dhhc_offset_temp_comp_set(&iis3dhhcDriver, PROPERTY_ENABLE);
		return HAL_OK;
	} else {
		return HAL_ERROR;
	}
}
```

main.c : Implementing the initialization function for each sensor 3/4

```
/* Initialize Accelerometer */
HAL_StatusTypeDef Init_LIS2DW12(void) {
          /* Check device ID for LIS2DW12 accelerometer */
          whoamI = 0;
          lis2dw12_device_id_get(&lis2dw12Driver, &whoamI);
          /* Configure LIS2DW12 accelerometer */
          if (whoamI == LIS2DW12_ID) {
                    /* Restore default configuration */
                    lis2dw12_reset_set(&lis2dw12Driver, PROPERTY_ENABLE);
                    do {
                              lis2dw12_reset_get(&lis2dw12Driver, &rst);
                    } while (rst);
                    /* Enable Block Data Update */
                    lis2dw12_block_data_update_set(&lis2dw12Driver, PROPERTY_ENABLE);
                    /* Set full scale */
                    lis2dw12_full_scale_set(&lis2dw12Driver, LIS2DW12_2g);
                    /* Configure filtering chain */
                    lis2dw12_filter_path_set(&lis2dw12Driver, LIS2DW12_LPF_ON_OUT);
                    lis2dw12_filter_bandwidth_set(&lis2dw12Driver, LIS2DW12_ODR_DIV_4);
                    /* Configure power mode */
                    lis2dw12_power_mode_set(&lis2dw12Driver, LIS2DW12_HIGH_PERFORMANCE);
                    /* Set Output Data Rate */
                    lis2dw12_data_rate_set(&lis2dw12Driver, LIS2DW12_XL_ODR_1k6Hz);
                    return HAL_OK;
          } else {
                    return HAL_ERROR;
          }
}
```

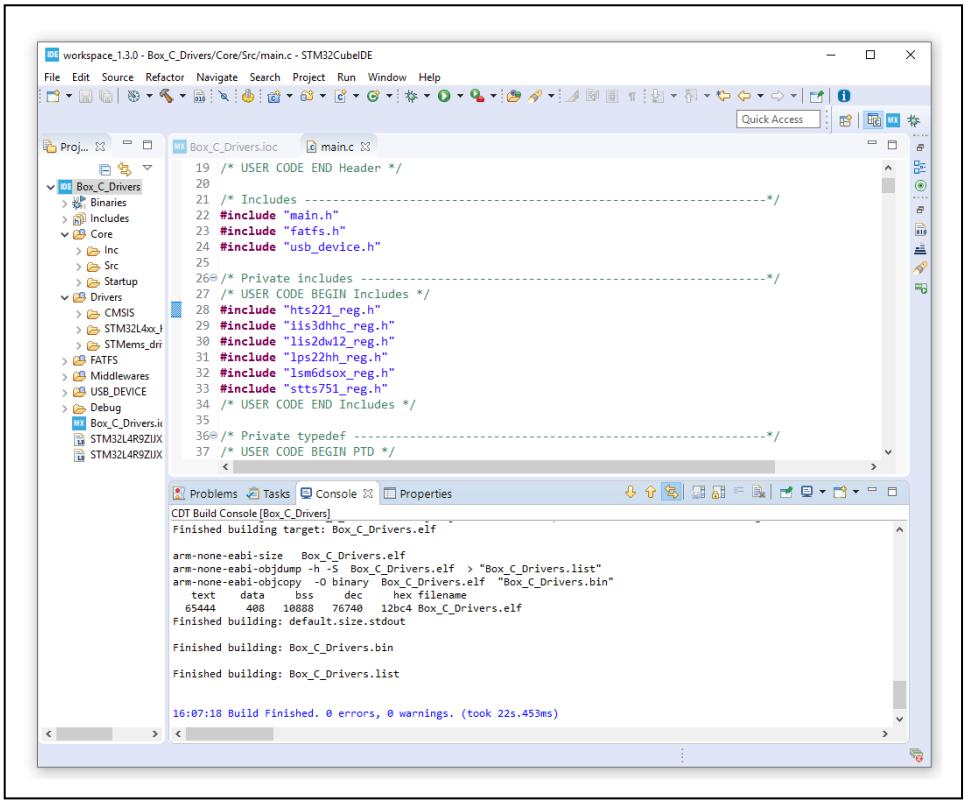
main.c : Implementing the read and write function for each sensor 4/4

```
static int32_t hts221_read(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len) {
          if (len > 1) reg |= 0x80;
          return HAL_I2C_Mem_Read(handle, HTS221_I2C_ADDRESS, reg, I2C_MEMADD_SIZE_8BIT, bufp, len, 1000);
}

static int32_t hts221_write(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len) {
          if (len > 1) reg |= 0x80;
          return HAL_I2C_Mem_Write(handle, HTS221_I2C_ADDRESS, reg, I2C_MEMADD_SIZE_8BIT, bufp, len, 1000);
}

static int32_t lps22hh_read(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len) {
          return HAL_I2C_Mem_Read(handle, LPS22HH_I2C_ADD_H, reg, I2C_MEMADD_SIZE_8BIT, bufp, len, 1000);
}

static int32_t lps22hh_write(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len) {
          return HAL_I2C_Mem_Write(handle, LPS22HH_I2C_ADD_H, reg, I2C_MEMADD_SIZE_8BIT, bufp, len, 1000);
}

static int32_t stts751_read(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len) {
          return HAL_I2C_Mem_Read(handle, STTS751_0xxxx_ADD_20K, reg, I2C_MEMADD_SIZE_8BIT, bufp, len, 1000);
}

static int32_t stts751_write(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len) {
          return HAL_I2C_Mem_Write(handle, STTS751_0xxxx_ADD_20K, reg, I2C_MEMADD_SIZE_8BIT, bufp, len, 1000);
}

static int32_t lsm6dsox_read(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len) {
          int32_t retVal = 0;
          uint8_t regVal = (reg | 0x80);
          HAL_GPIO_WritePin(CS_LSM6DSOX_GPIO_Port, CS_LSM6DSOX_Pin, GPIO_PIN_RESET);
          HAL_SPI_Transmit(handle, &regVal, 1, 1000);
          retVal = HAL_SPI_Receive(handle, bufp, len, 1000);
          HAL_GPIO_WritePin(CS_LSM6DSOX_GPIO_Port, CS_LSM6DSOX_Pin, GPIO_PIN_SET);
          return retVal;
}

static int32_t lsm6dsox_write(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len) {
          int32_t retVal = 0;
          HAL_GPIO_WritePin(CS_LSM6DSOX_GPIO_Port, CS_LSM6DSOX_Pin, GPIO_PIN_RESET);
          HAL_SPI_Transmit(handle, &reg, 1, 1000);
          retVal = HAL_SPI_Transmit(handle, bufp, len, 1000);
          HAL_GPIO_WritePin(CS_LSM6DSOX_GPIO_Port, CS_LSM6DSOX_Pin, GPIO_PIN_SET);
          return retVal;
}

static int32_t iis3dhhc_read(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len) {
          int32_t retVal = 0;
          uint8_t regVal = (reg | 0x80);
          HAL_GPIO_WritePin(CS_IIS3DHHC_GPIO_Port, CS_IIS3DHHC_Pin, GPIO_PIN_RESET);
          HAL_SPI_Transmit(handle, &regVal, 1, 1000);
          retVal = HAL_SPI_Receive(handle, bufp, len, 1000);
          HAL_GPIO_WritePin(CS_IIS3DHHC_GPIO_Port, CS_IIS3DHHC_Pin, GPIO_PIN_SET);
          return retVal;
}

static int32_t iis3dhhc_write(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len) {
          int32_t retVal = 0;
          HAL_GPIO_WritePin(CS_IIS3DHHC_GPIO_Port, CS_IIS3DHHC_Pin, GPIO_PIN_RESET);
          HAL_SPI_Transmit(handle, &reg, 1, 1000);
          retVal = HAL_SPI_Transmit(handle, bufp, len, 1000);
          HAL_GPIO_WritePin(CS_IIS3DHHC_GPIO_Port, CS_IIS3DHHC_Pin, GPIO_PIN_SET);
          return retVal;
}

static int32_t lis2dw12_read(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len) {
          int32_t retVal = 0;
          uint8_t regVal = (reg | 0x80);
          HAL_GPIO_WritePin(CS_LIS2DW12_GPIO_Port, CS_LIS2DW12_Pin, GPIO_PIN_RESET);
          HAL_SPI_Transmit(handle, &regVal, 1, 1000);
          retVal = HAL_SPI_Receive(handle, bufp, len, 1000);
          HAL_GPIO_WritePin(CS_LIS2DW12_GPIO_Port, CS_LIS2DW12_Pin, GPIO_PIN_SET);
          return retVal;
}

static int32_t lis2dw12_write(void *handle, uint8_t reg, uint8_t *bufp, uint16_t len) {
          int32_t retVal = 0;
          HAL_GPIO_WritePin(CS_LIS2DW12_GPIO_Port, CS_LIS2DW12_Pin, GPIO_PIN_RESET);
          HAL_SPI_Transmit(handle, &reg, 1, 1000);
          retVal = HAL_SPI_Transmit(handle, bufp, len, 1000);
          HAL_GPIO_WritePin(CS_LIS2DW12_GPIO_Port, CS_LIS2DW12_Pin, GPIO_PIN_SET);
          return retVal;
}
/* USER CODE END 4 */
```

To verify that all the steps have been implemented correctly, press the  button to start the compilation.

If the compilation succeeds, the build should finish with 0 errors and 0 warnings, as depicted in Figure 12.

Figure 12. Compilation succeeds



Step 5: Reading data from the sensors

The next snippets of code can be added to test the data reading from the sensors whose drivers have been imported. For simplicity, polling mode will be used in the main while loop with a 100 ms delay between consecutive reads.

main.c : Implementing the definition of 1D and 3D axis

```
/* USER CODE BEGIN PTD */
typedef union {
           int16_t i16bit;
           uint8_t u8bit[2];
} axis1bit16_t;

typedef union {
           int32_t i32bit;
           uint8_t u8bit[4];
} axis1bit32_t;

typedef union {
           int16_t i16bit[3];
           uint8_t u8bit[6];
} axis3bit16_t;
/* USER CODE END PTD */
```

main.c : Defining data variables for each sensor

```
/* USER CODE BEGIN 1 */
hts221_reg_t reg_HTS221;
static axis1bit16_t data_raw_humidity_HTS221;         static float humidity_perc_HTS221;
static axis1bit16_t data_raw_temperature_HTS221;      static float temperature_degC_HTS221;

lps22hh_reg_t reg_LPS22HH;
static axis1bit32_t data_raw_pressure_LPS22HH;        static float pressure_hPa_LPS22HH;
static axis1bit16_t data_raw_temperature_LPS22HH;     static float temperature_degC_LPS22HH;

uint8_t reg_STTS751;
static axis1bit16_t data_raw_temperature_STTS751;     static float temperature_degC_STTS751;

uint8_t reg_LSM6DSOX;
static axis3bit16_t data_raw_acceleration_LSM6DSOX;   static float acceleration_mg_LSM6DSOX[3];
static axis3bit16_t data_raw_angular_rate_LSM6DSOX;   static float angular_rate_mdps_LSM6DSOX[3];

iis3dhhc_reg_t reg_IIS3DHHC;
static axis3bit16_t data_raw_acceleration_IIS3DHHC;   static float acceleration_mg_IIS3DHHC[3];

uint8_t reg_LIS2DW12;
static axis3bit16_t data_raw_acceleration_LIS2DW12;   static float acceleration_mg_LIS2DW12[3];
/* USER CODE END 1 */
```

main.c : Reading data from sensors when available and converting to physical unit

```
/* USER CODE BEGIN 3 */
          /* Humidity from HTS221 */
          hts221_status_get(&hts221Driver, &reg_HTS221.status_reg);
          if (reg_HTS221.status_reg.h_da) {
                    memset(data_raw_humidity_HTS221.u8bit, 0x00, sizeof(int16_t));
                    hts221_humidity_raw_get(&hts221Driver, data_raw_humidity_HTS221.u8bit);
                    humidity_perc_HTS221 = linear_interpolation(&lin_hum, data_raw_humidity_HTS221.i16bit);
                    if (humidity_perc_HTS221 < 0) humidity_perc_HTS221 = 0;
                    if (humidity_perc_HTS221 > 100) humidity_perc_HTS221 = 100;
          }
          /* Temperature from HTS221 */
          if (reg_HTS221.status_reg.t_da) {
                    memset(data_raw_temperature_HTS221.u8bit, 0x00, sizeof(int16_t));
                    hts221_temperature_raw_get(&hts221Driver, data_raw_temperature_HTS221.u8bit);
                    temperature_degC_HTS221 = linear_interpolation(&lin_temp, data_raw_temperature_HTS221.i16bit);
          }
          /* Pressure from LPS22HH */
          lps22hh_read_reg(&lps22hhDriver, LPS22HH_STATUS, (uint8_t *)&reg_LPS22HH, 1);
          if (reg_LPS22HH.status.p_da) {
                    memset(data_raw_pressure_LPS22HH.u8bit, 0x00, sizeof(int32_t));
                    lps22hh_pressure_raw_get(&lps22hhDriver, data_raw_pressure_LPS22HH.u8bit);
                    pressure_hPa_LPS22HH = lps22hh_from_lsb_to_hpa(data_raw_pressure_LPS22HH.i32bit);
          }
          /* Temperature from LPS22HH */
          if (reg_LPS22HH.status.t_da) {
                    memset(data_raw_temperature_LPS22HH.u8bit, 0x00, sizeof(int16_t));
                    lps22hh_temperature_raw_get(&lps22hhDriver, data_raw_temperature_LPS22HH.u8bit);
                    temperature_degC_LPS22HH = lps22hh_from_lsb_to_celsius(data_raw_temperature_LPS22HH.i16bit);
          }
          /* Temperature from STTS751 */
          stts751_flag_busy_get(&stts751Driver, &reg_STTS751);
          if (reg_STTS751) {
                    memset(data_raw_temperature_STTS751.u8bit, 0, sizeof(int16_t));
                    stts751_temperature_raw_get(&stts751Driver, &data_raw_temperature_STTS751.i16bit);
                    temperature_degC_STTS751 = stts751_from_lsb_to_celsius(data_raw_temperature_STTS751.i16bit);
          }
          /* Acceleration from LSM6DSOX */
          lsm6dsox_xl_flag_data_ready_get(&lsm6dsoxDriver, &reg_LSM6DSOX);
          if (reg_LSM6DSOX) {
                    memset(data_raw_acceleration_LSM6DSOX.u8bit, 0x00, 3 * sizeof(int16_t));
                    lsm6dsox_acceleration_raw_get(&lsm6dsoxDriver, data_raw_acceleration_LSM6DSOX.u8bit);
                    acceleration_mg_LSM6DSOX[0] = lsm6dsox_from_fs2_to_mg(data_raw_acceleration_LSM6DSOX.i16bit[0]);
                    acceleration_mg_LSM6DSOX[1] = lsm6dsox_from_fs2_to_mg(data_raw_acceleration_LSM6DSOX.i16bit[1]);
                    acceleration_mg_LSM6DSOX[2] = lsm6dsox_from_fs2_to_mg(data_raw_acceleration_LSM6DSOX.i16bit[2]);
          }
          /* Angular Rate from LSM6DSOX */
          lsm6dsox_gy_flag_data_ready_get(&lsm6dsoxDriver, &reg_LSM6DSOX);
          if (reg_LSM6DSOX) {
                    memset(data_raw_angular_rate_LSM6DSOX.u8bit, 0x00, 3 * sizeof(int16_t));
                    lsm6dsox_angular_rate_raw_get(&lsm6dsoxDriver, data_raw_angular_rate_LSM6DSOX.u8bit);
                    angular_rate_mdps_LSM6DSOX[0] = lsm6dsox_from_fs2000_to_mdps(data_raw_angular_rate_LSM6DSOX.i16bit[0]);
                    angular_rate_mdps_LSM6DSOX[1] = lsm6dsox_from_fs2000_to_mdps(data_raw_angular_rate_LSM6DSOX.i16bit[1]);
                    angular_rate_mdps_LSM6DSOX[2] = lsm6dsox_from_fs2000_to_mdps(data_raw_angular_rate_LSM6DSOX.i16bit[2]);
          }
          /* Acceleration from IIS3DHHC */
          iis3dhhc_status_get(&iis3dhhcDriver, &reg_IIS3DHHC.status);
          if (reg_IIS3DHHC.status.zyxda) {
                    memset(data_raw_acceleration_IIS3DHHC.u8bit, 0x00, 3 * sizeof(int16_t));
                    iis3dhhc_acceleration_raw_get(&iis3dhhcDriver, data_raw_acceleration_IIS3DHHC.u8bit);
                    acceleration_mg_IIS3DHHC[0] = iis3dhhc_from_lsb_to_mg(data_raw_acceleration_IIS3DHHC.i16bit[0]);
                    acceleration_mg_IIS3DHHC[1] = iis3dhhc_from_lsb_to_mg(data_raw_acceleration_IIS3DHHC.i16bit[1]);
                    acceleration_mg_IIS3DHHC[2] = iis3dhhc_from_lsb_to_mg(data_raw_acceleration_IIS3DHHC.i16bit[2]);
          }
          /* Acceleration from LIS2DW12 */
          lis2dw12_flag_data_ready_get(&lis2dw12Driver, &reg_LIS2DW12);
          if (reg_LIS2DW12) {
                    memset(data_raw_acceleration_LIS2DW12.u8bit, 0x00, 3 * sizeof(int16_t));
                    lis2dw12_acceleration_raw_get(&lis2dw12Driver, data_raw_acceleration_LIS2DW12.u8bit);
                    acceleration_mg_LIS2DW12[0] = lis2dw12_from_fs2_to_mg(data_raw_acceleration_LIS2DW12.i16bit[0]);
                    acceleration_mg_LIS2DW12[1] = lis2dw12_from_fs2_to_mg(data_raw_acceleration_LIS2DW12.i16bit[1]);
                    acceleration_mg_LIS2DW12[2] = lis2dw12_from_fs2_to_mg(data_raw_acceleration_LIS2DW12.i16bit[2]);
          }
          HAL_Delay(100);
          HAL_GPIO_TogglePin(CPU_LED_GPIO_Port, CPU_LED_Pin);
}
/* USER CODE END 3 */
```

Again, to verify that all the steps have been implemented correctly, press the  button to start the compilation.

If the compilation succeeds, the build should finish with 0 errors and 0 warnings, as depicted in Figure 12.

Support material

Related design support material
STMicroelectronics MEMS Standard C drivers
STEVAL-MKSBOX1V1 - SensorTile.box wireless multi-sensor development kit
STM32CubeIDE - Integrated Development Environment for STM32
Documentation
Data brief STEVAL-MKSBOX1V1
Schematic STEVAL-MKSBOX1V1

Revision history

Date	Version	Changes
05-Mar-2021	1	Initial release

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to *www.st.com/trademarks*. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2021 STMicroelectronics – All rights reserved