

Metrology firmware for the STM32F407VG and the STPM32 devices

Introduction

The following document describes a firmware for the STM32F407VG that implements a poly-phase system interfacing with several STPM32 metrology devices. This firmware can be tested using the STM32F4DISCOVERY and EVALSTPM32.

The package includes:

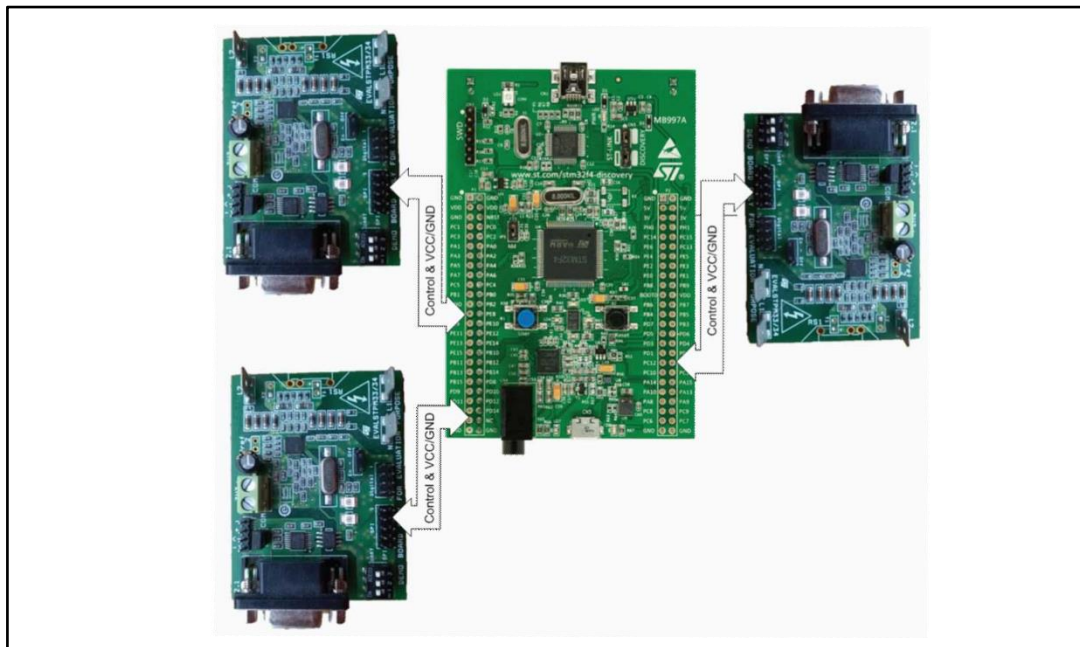
- Complete metrology up to 4 STPM32 devices
- EEPROM support for saving parameters (optional)
- USB virtual COM port (VCP) to interface with PC tools
- Mini-shell (command line parser) support to send commands through USB
- STPM32 communication through UART or SPI
- Simple tasks based on the STM32F407VG timers

The STM32F407VG initialization uses the STM32CubeMX toolchain.

The STPM32 driver integration is based on IAR tool chain with ST-LINK (embedded on-board).

For further reference please refer to the STPM3x and the STM32F407VG datasheets.

Figure 1: Poly-phase application



Contents

1 STM32CubeMX configuration 3

2 Hardware configuration 4

3 STPM32 configuration..... 7

 3.1 Communication mode 7

 3.2 Reset..... 8

 3.3 SPI and UART communication modes..... 8

 3.4 Latch data measure 8

4 Firmware block diagram 9

 4.1 Main file..... 9

 4.1.1 MNSH_task 10

 4.1.2 Metro Task..... 10

 4.1.3 METRO_Get_Measures..... 10

 4.1.4 Metro_UpdateData 10

 4.2 USB virtual COM port..... 10

 4.2.1 CDC_Receive..... 11

 4.2.2 CDC_Send 11


 4.3 EEPROM..... 11

 4.4 Integration with IAR tool chain..... 11

5 Revision history 15

1 STM32CubeMX configuration

The STM32F4DISCOVERY kit is based on the STM32F407VGT microcontroller.

- RCC high speed clock HSE (crystal/ceramic resonator) and RCC high speed clock LSE (crystal/ceramic resonator)
 - SYS debug (SWD: serial wire debug)
 - USB in full speed mode
 - Input frequency is 8 MHz with SYSCLK set to 96 MHz
 - SPI1, SPI2, SPI3 are used in full duplex mode with hardware NSS signal disable
 - Frame format: Motorola
 - Data size: 8 bits
 - First bit: MSB first
 - Clock polarity: high
 - Clock phase: 2nd edge
 - CRC calculation: disabled
 - TIM2 and TIM3 clock source is the internal clock (used for USB, mini-shell and metro application)
 - USART2, USART3 and USART6 in asynchronous mode with hardware flow control disable
 - Baud rate: 9600 bits/s
 - Word length: 8 bits
 - Parity: none
 - Stop bits: 1
 - Data direction: receive and transmit
 - Several GPIOs for controlling
 - STPM32 enable
 - STPM32 SYN pin
 - STPM32 SCS pin
-
- 
- SPI and USART configuration depends on the topology used
 - VCP driver from ST must be installed on the PC
 - In the application, three STPM32 devices only are connected but the code has been developed to support up to four STPM32 devices
 - **Isolators must be put between the EVALSTPM32 and the STM32F4DISCOVERY and, in case of tests with connection to mains, on USB as well**

2 Hardware configuration

Figure 2: Hardware configuration UART mode

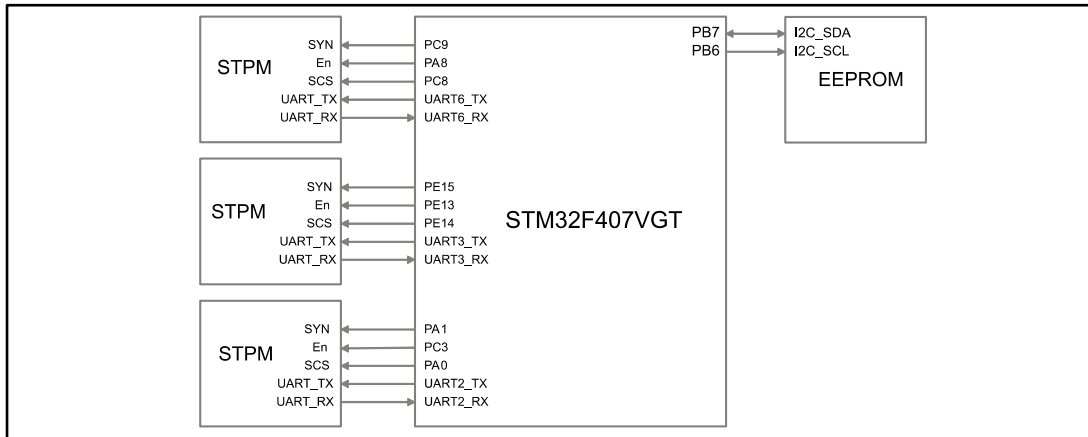
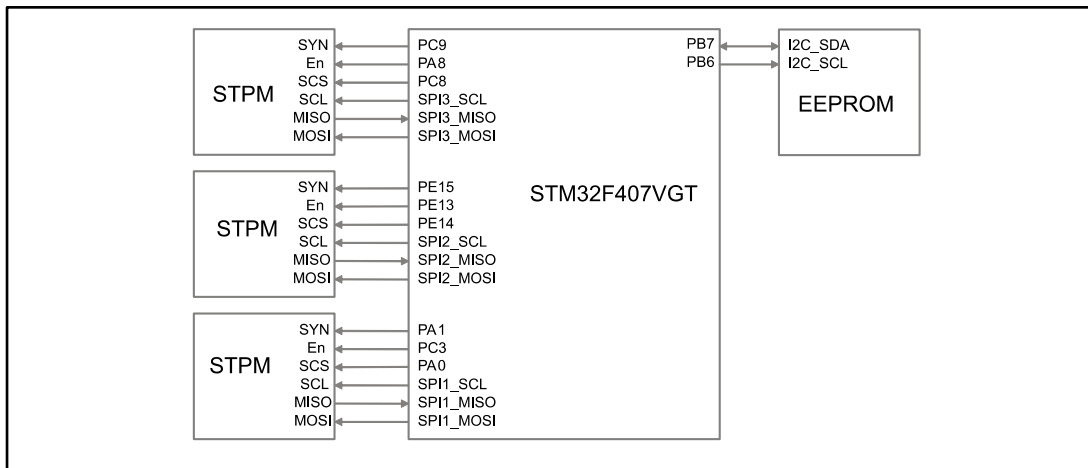


Figure 3: Hardware configuration SPI mode



The configuration of the GPIO output and UART/SPI peripherals is performed by the STM32CubeMX.

The assignment is carried out in the metrology handler, in the structure STPM_com_port[4].

The assignment can be changed by modifying this structure as described below.

```
const STPM_Com_port_t STPM_com_port[4] ={\n  {\n    USART3,    //USART used by device 1\n    GPIOE,     //CS used by device 1\n    GPIO_PIN_14,\n    GPIOE,     //SYN used by device 1\n    GPIO_PIN_15,\n    GPIOE,     //EN used by device 1\n    GPIO_PIN_13\n  },\n  {\n    USART6,    //USART used by device 2\n    GPIOC,     //CS used by device 2\n    GPIO_PIN_8,\n    GPIOC,     //SYN used by device 2\n    GPIO_PIN_9,\n    GPIOA,     //EN used by device 2\n    GPIO_PIN_8\n  },\n  {\n    USART2,    //USART used by device 3\n    GPIOA,     //CS used by device 3\n    GPIO_PIN_0,\n    GPIOA,     //SYN used by device 3\n    GPIO_PIN_1,\n    GPIOC,     //EN used by device 3\n    GPIO_PIN_3\n  },\n  {\n    USART2,    //USART used by device 4\n    GPIOA,     //CS used by device 4\n    GPIO_PIN_0,\n    GPIOA,     //SYN used by device 4\n    GPIO_PIN_1,\n    GPIOC,     //EN used by device 4\n    GPIO_PIN_3\n  }\n};\nconst STPM_Com_port_t STPM_com_port[4] ={\n  {\n    SPI2,      //SPI used by device 1\n    GPIOE,     //CS used by device 1
```

```
GPIO_PIN_14,  
GPIOE,      //SYN used by device 1  
GPIO_PIN_15,  
GPIOE,      //EN used by device 1  
GPIO_PIN_13  
},  
{  
SPI3,      //SPI used by device 2  
GPIOC,     //CS used by device 2  
GPIO_PIN_8,  
GPIOC,     //SYN used by device 2  
GPIO_PIN_9,  
GPIOA,     //EN used by device 2  
GPIO_PIN_8  
},  
{  
SPI1,     //SPI used by device 3  
GPIOA,    //CS used by device 3  
GPIO_PIN_0,  
GPIOA,    //SYN used by device 3  
GPIO_PIN_1,  
GPIOC,    //EN used by device 3  
GPIO_PIN_3  
},  
{  
SPI1,     //SPI used by device 4  
GPIOA,    //CS used by device 4  
GPIO_PIN_0,  
GPIOA,    //SYN used by device 4  
GPIO_PIN_1,  
GPIOC,    //EN used by device 4  
GPIO_PIN_3  
}  
};
```

3 STPM32 configuration

3.1 Communication mode

The STPM32 is configured during the startup in SPI or in UART mode.

If the SCS is set low and EN pin goes from low to high, the STPM32 communicates through SPI. If the SCS is set high and EN pin goes from low to high, the STPM32 communicates through UART.

In the FW, the STM32 enables each of the STPM32 devices through GPIOs. The STPM32 devices are enabled one by one by the signal sequence below, according to the chosen communication peripheral.

Figure 4: UART communication peripheral selection

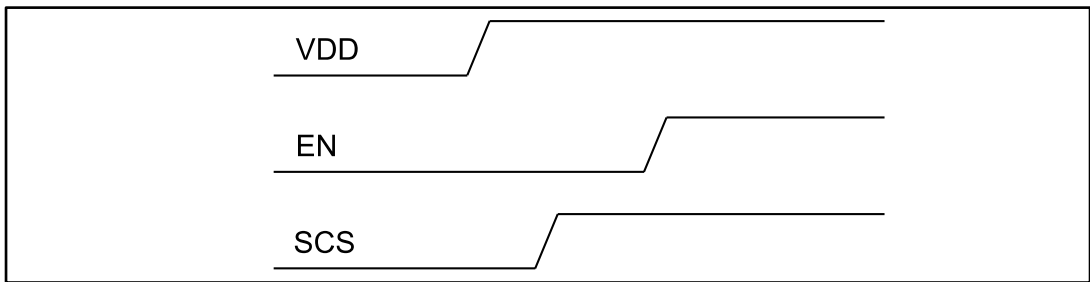
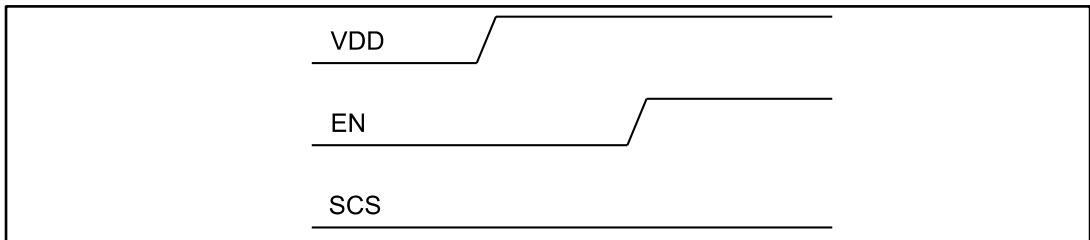


Figure 5: SPI communication peripheral selection

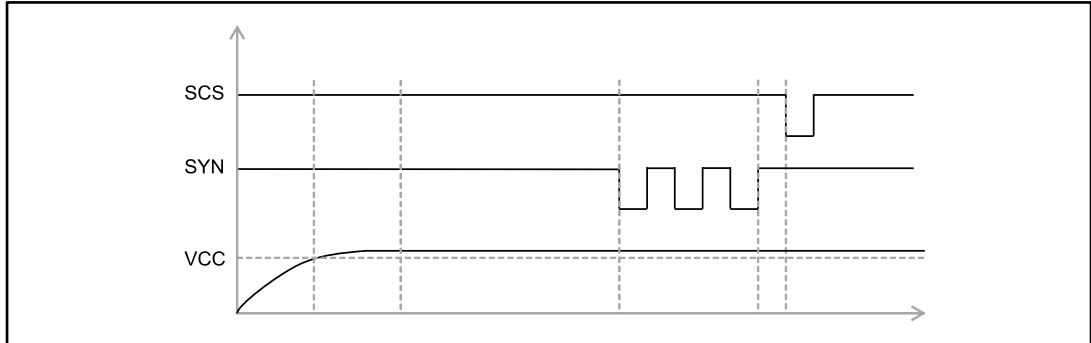


In the FW, this service is included into Metro_Init() and it is called Metro_power_up_device().

3.2 Reset

As mentioned in the STPM3x datasheet, after the POR, the chip must be reset by toggling 3 times SYN pin and once SCS pin.

Figure 6: Reset signal



In the FW, the complete sequence is in the function `Metro_Init()`.

3.3 SPI and UART communication modes

In SPI mode the communication speed is based on the SPI clock.

In UART mode, by default, the STPM32 works at 9600 bauds. In order to speed up the communication, please use `Metro_UartSpeed(uint32_t baudrate)` function to change the baud rate.

3.4 Latch data measure

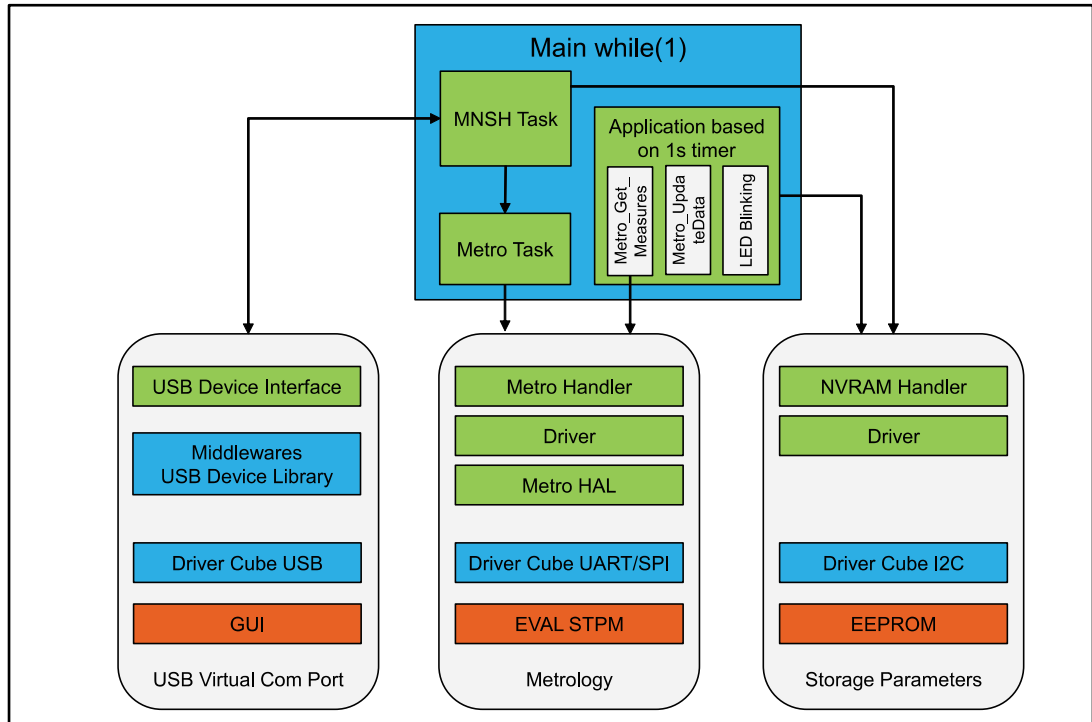
The STPM32 registers can be latched in different ways:

- By FW, by setting the STPM32 DSPCTRL3 register
- By HW, using SYN pin (this method is used in the FW)

The choice of the method depends on the function `uint8_t Metro_Set_Latch_device_type(METRO_NB_Device_t in_Metro_Device, METRO_Latch_Device_Type_t in_Metro_Latch_Device_Type)`.

4 Firmware block diagram

Figure 7: Firmware block diagram



In the figure above:

- In blue : services from Cube environment
- In green : FW package
- In orange : HW components

The package includes:

- The STPM32 management (communication, reset, latch etc.)
- Full metrology measurement
- NVM management with EEPROM storage
- Communication through USB in virtual COM port mode
- Two timers for main state machine

4.1 Main file

The main file includes the complete initialization process:

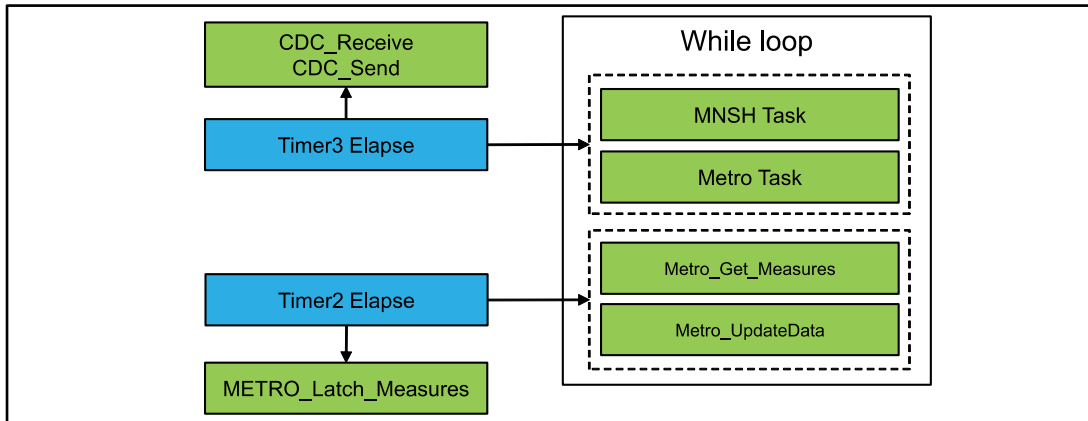
- MCU initialization based on the STM32CubeMX (UART or SPI, SystemClock, GPIOs)
- Timer2 initialization (1 s timer for the metrology latch and measurement)
- Timer3 initialization (1 ms timer for USB Endpoint management, mini-shell task and metro task linked to external requirement)
- NVM_Init : NVRAM initialization which sets default parameters to the STPM32 or parameters coming from the EEPROM
- METRO_Init : metrology initialization configures the different STPM32 devices, and sets the topology (number of phases etc.)

- MNSH_TaskInit: initializes the mini-shell with different modules

The state machine is based on the while loop and two timers.

These two timers are configured in interrupt mode with call back “HAL_TIM_PeriodElapsedCallback”. In case of Timer3, the USB Endpoint in and out are updated and a counter is incremented and used in while loop for Metro_task and MNSH_task. In case of Timer2, the STPM32 is latched and a counter is incremented and used in while loop to read the STPM32 values and compute metrology data.

Figure 8: While loop



4.1.1 MNSH_task

This function stores the message coming from the USB.

When the message is complete, it is parsed and sent to the different mini-shell modules.

“mnsheeprom” parses the message and accesses directly the EEPROM. This module is optional and used for debugging EEPROM access.

In case of message related to metrology, the message is parsed in mnsheeprom and sent to the METRO_task.

4.1.2 Metro Task

This is the access to metrology services, which are addressed according to the command received by mnsheeprom.

4.1.3 METRO_Get_Measures

The data are latched during an interrupt using METRO_Latch_Measures, and the STPM32 devices are read through METRO_Get_Measures. The reading is fulfilled in background since the transfer time depends on the speed used for UART or SPI ports.

4.1.4 Metro_UpdateData

The data computation (energy, power, etc.) is carried out after collecting measures coming from the STPM32.

The values are stored into metroData.

4.2 USB virtual COM port

The USB VCP is implemented in the file usbd_cdc_if.c.

mnshVars.txData and mnshVars.rxData are used to send and receive the data into the Endpoint.

The transmission and reception of data is through packet mode.

The reception is managed into CDC_Receive() and the transmission into CDC_Send() functions, which are called by the Timer2 event.

4.2.1 CDC_Receive

The first byte is compared to 0 to check data availability from the endpoint.

The filtering is implemented to remove noise from the keyboard data reception:

- Check if the packet contains special keys (like arrows)
- Check if the packet received comes from a command
 - from the GUI (terminated by “\n” or “\r”) and all bytes are valid
 - from keyboard terminated by “0” and, in this case only the first byte is valid.

The complete message is managed by the MNSH_UsbRxHandler().

4.2.2 CDC_Send

All data from the mnshVars.txData buffer are sent to the endpoint.

4.3 EEPROM

The EEPROM is used to save the configuration. It is connected to the STM32F407VG using I²C port, configured by the STM32CubeMX tool. The read and the write access is in page mode.

Page size and word definition to access EEPROM are defined into handler_eeprom.h.

EEPROM is optional and enabled by the #define EEPROM_PRESENT.

4.4 Integration with IAR tool chain

The firmware has been developed for the STM32F407VG, but it can be easily ported to other STM32 microcontrollers with minor changes.

Two folders should be added to the STM32CubeMX folders for this application:

- Generic: containing mini-shell, EEPROM services, NVRAM services
- Metrology: containing all metrology functions

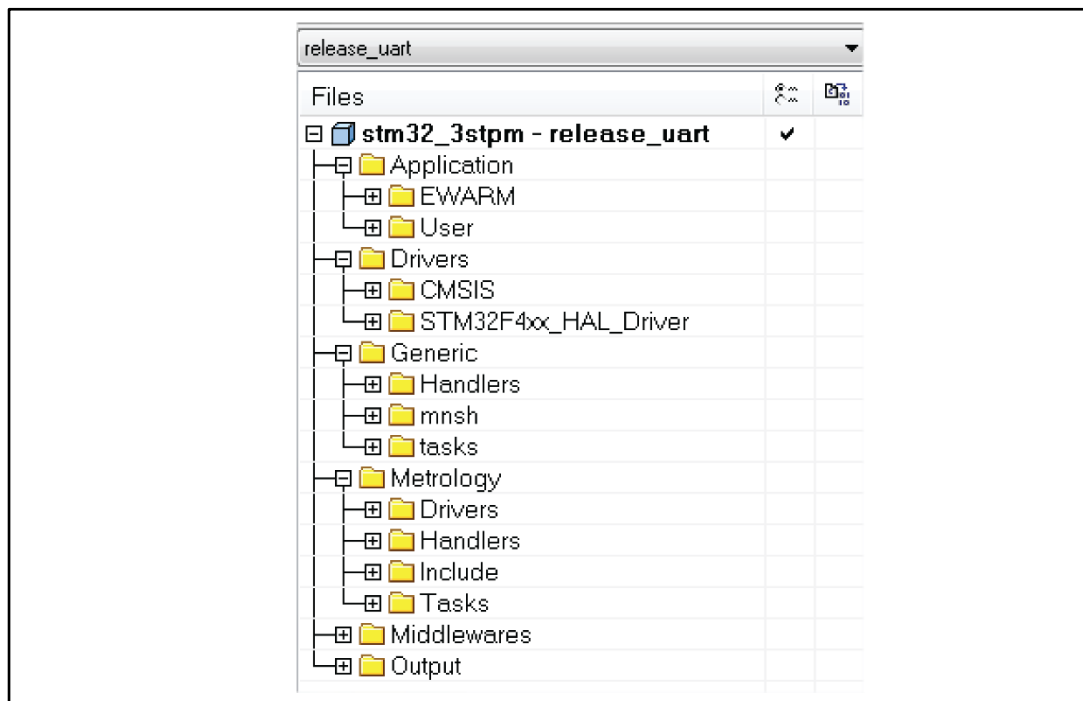
The metrology and generic packages are at the same level as the other folders from the STM32CubeMX.

Figure 9: Folder organization

 Drivers	29/04/2016 14:43	File folder
 EWARM	13/05/2016 10:57	File folder
 Generic	04/05/2016 17:09	File folder
 Inc	10/05/2016 17:28	File folder
 Metrology	29/04/2016 14:43	File folder
 Middlewares	29/04/2016 14:43	File folder
 Src	12/05/2016 16:48	File folder
 .mxproject	21/04/2016 13:52	MXPROJECT File

In the IAR tool chain, folders and files need to be added. The following organization is used:

Figure 10: IAR file organization

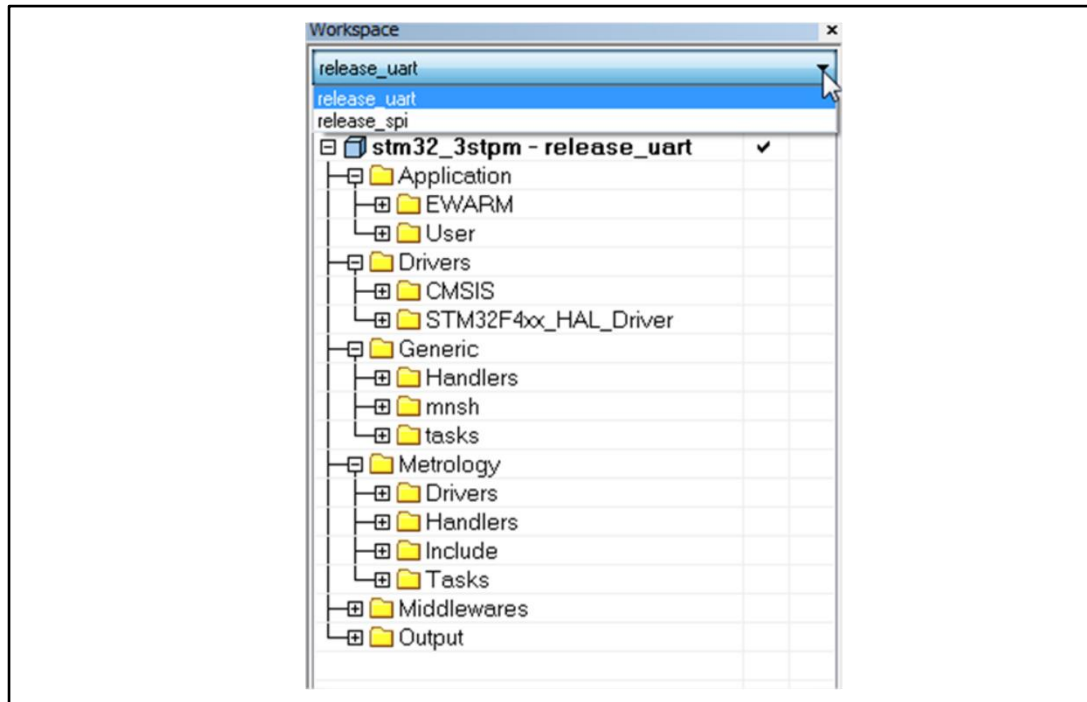


After the integration of files into the project, they need to be included. The following symbols must be defined for the compiler:

- SPI_XFER_STPM3X for SPI communication
- UART_XFER_STPM3X for UART communication
- EEPROM_PRESENT for EEPROM management

It is possible to choose directly the configuration in the project by selecting the right compile flag.

Figure 11: Release SPI/UART



There is a section to update C/C++ compiler options.

The following needs adding

```
$PROJ_DIR$/../Metrology/drivers/inc
```

```
$PROJ_DIR$/../Metrology/handlers/inc
```

```
$PROJ_DIR$/../Metrology/tasks/inc
```

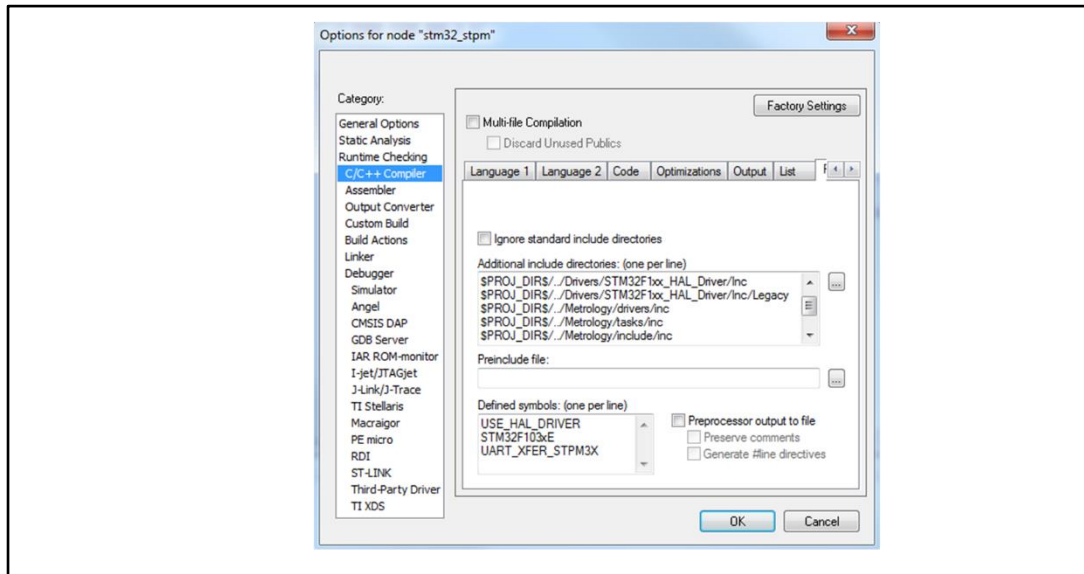
```
$PROJ_DIR$/../Metrology/include/inc
```

```
$PROJ_DIR$/../Generic/mnsh/inc
```

```
$PROJ_DIR$/../Generic/tasks/inc
```

```
$PROJ_DIR$/../Generic/handlers/inc
```

Figure 12: Compiler update



In order to have FW up and running, all changes are set into usdb_cdc_if.c, main.c files, st_device.h and handler_metrology.c.

5 Revision history

Table 1: Document revision history

Date	Revision	Changes
05-Dec-2016	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. 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.

© 2016 STMicroelectronics – All rights reserved