UM2481

User manual

Getting started with the X-CUBE-EEPRMA1 software expansion for STM32Cube

Introduction

The X-CUBE-EEPRMA1 software expansion for STM32Cube provides an evaluation software example for M24XX I²C and M95XX SPI EEPROMs.

The package is built on STM32Cube software technology to ease portability across different STM32 microcontrollers.

The software comes with sample implementations of the drivers running on the X-NUCLEO-EEPRMA1 expansion board connected to the featured development boards.

1 Acronyms and abbreviations

Table 1. List of acronyms

Acronym	Description
BSP	Board support package
CMSIS	Cortex® microcontroller software interface standard
HAL	Hardware abstraction layer
I2C	Inter integrated circuit
SPI	Serial peripheral interface

2 What is STM32Cube?

STM32Cube™ represents the STMicroelectronics initiative to make developers' lives easier by reducing development effort, time and cost. STM32Cube covers the STM32 portfolio.

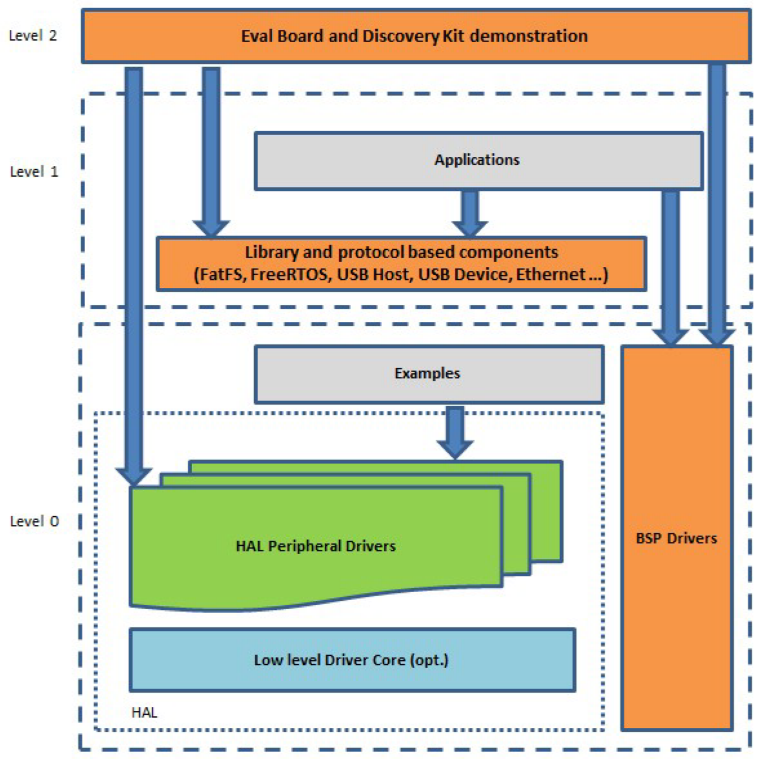
STM32Cube version 1.x includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform specific to each series (such as the STM32Cube for the STM32 series), which includes:
  - the STM32Cube HAL embedded abstraction-layer software, ensuring maximized portability across the STM32 portfolio
  - a consistent set of middleware components such as RTOS, USB, TCP/IP and graphics
  - all embedded software utilities with a full set of examples

2.1 STM32Cube architecture

The STM32Cube firmware solution is built around three independent levels that can easily interact with one another, as described in the diagram below.

Figure 1. Firmware architecture


Level 2
Eval Board and Discovery Kit demonstration
Level 1
Applications
Library and protocol based components
(FatFS, FreeRTOS, USB Host, USB Device, Ethernet ...)
Level 0
Examples
HAL Peripheral Drivers
Low level Driver Core (opt.)
HAL
BSP Drivers

Level 0: This level is divided into three sub-layers:

- Board Support Package (BSP): this layer offers a set of APIs relative to the hardware components in the hardware boards (Audio codec, IO expander, Touchscreen, SRAM driver, LCD drivers. etc…); it is based on modular architecture allowing it to be easily ported on any hardware by just implementing the low level routines. It is composed of two parts:

  - Component: is the driver relative to the external device on the board and not related to the STM32, the component driver provides specific APIs to the external components of the BSP driver, and can be ported on any other board.
  - BSP driver: links the component driver to a specific board and provides a set of easy to use APIs. The API naming convention is BSP_FUNCT_Action(): e.g., BSP_LED_Init(), BSP_LED_On().
- Hardware Abstraction Layer (HAL): this layer provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides generic, multi-instance and function-oriented APIs to help offload user application development time by providing ready to use processes. For example, for the communication peripherals (I^2C, UART, etc.) it provides APIs for peripheral initialization and configuration, data transfer management based on polling, interrupt or DMA processes, and communication error management. The HAL Drivers APIs are split in two categories: generic APIs providing common, generic functions to all the STM32 series and extension APIs which provide special, customized functions for a specific family or a specific part number.
- Basic peripheral usage examples: this layer houses the examples built around the STM32 peripherals using the HAL and BSP resources only.

Level 1: This level is divided into two sub-layers:

- Middleware components: set of libraries covering USB Host and Device Libraries, STemWin, FreeRTOS, FatFS, LwIP, and PolarSSL. Horizontal interaction among the components in this layer is performed directly by calling the feature APIs, while vertical interaction with low-level drivers is managed by specific callbacks and static macros implemented in the library system call interface. For example, FatFs implements the disk I/O driver to access a microSD drive or USB Mass Storage Class.
- Examples based on the middleware components: each middleware component comes with one or more examples (or applications) showing how to use it. Integration examples that use several middleware components are provided as well.

Level 2: This level is a single layer with a global, real-time and graphical demonstration based on the middleware service layer, the low level abstraction layer and basic peripheral usage applications for board-based functions.

3 X-CUBE-EEPRMA1 software expansion for STM32Cube

3.1 Overview

The X-CUBE-EEPRMA1 softeware package key features are:

- Complete software to build applications using M24XX or M95XX-based EEPROM
- Sample implementation available on the X-NUCLEO-EEPRMA1 expansion board plugged to a NUCLEO-F401RE or NUCLEO-L053R8 development board
- Easy portability across different MCU families thanks to STM32Cube
- Free user-friendly license terms

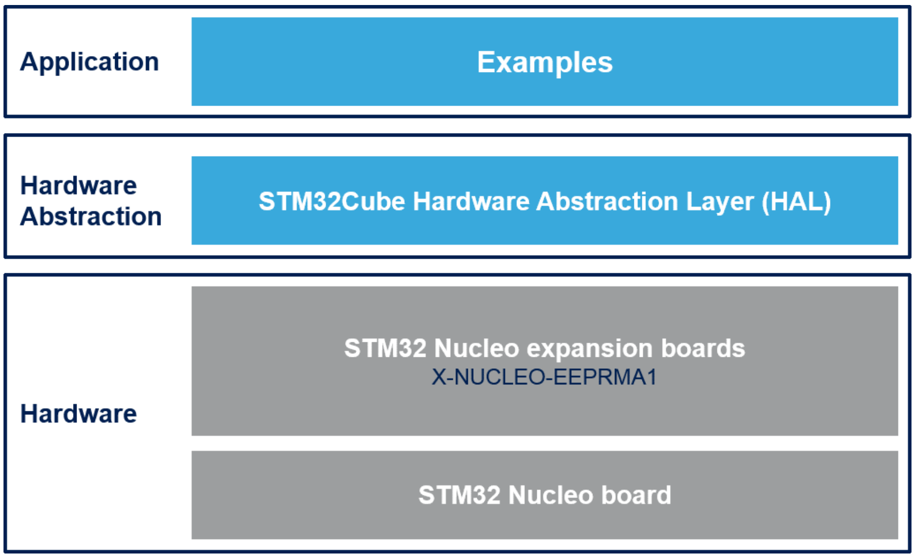
3.2 Architecture

The software is based on the STM32CubeHAL hardware abstraction layer for the STM32 microcontroller and extends STM32Cube by providing a board support package (BSP) for the X-NUCLEO-EEPRMA1 expansion board.

The software layers used by the application software to access and use the board are:

- **STM32Cube HAL layer**: provides a generic, multi-instance set of simple APIs (application programming interfaces) to interact with the upper layers (application, libraries and stacks). It is composed of generic and extension APIs.
- **Board support package (BSP) layer**: supports the peripherals on the STM32 Nucleo board, except for the MCU. This limited set of APIs provides a programming interface for certain board specific peripherals (like the user button, the reset button, etc.) and helps in identifying the specific board version.

Figure 2. **X-CUBE-EEPRMA1 architecture**



3.3 Folder structure

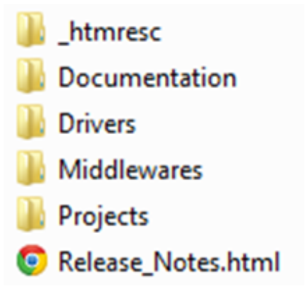


Figure 3. **X-CUBE-EEPRMA1 package folder structure**

The software is packaged in the following main folders:

- **Documentation**: with a compiled HTML file generated from the source code that details the software components and APIs.
- **Drivers**: contains the HAL drivers, the board specific drivers for each supported board or hardware platform, including the on-board components ones and the CMSIS layer which is a vendor-independent hardware abstraction layer for the Cortex-M processor series.
- **Projects**: contains sample application used for the NUCLEO-L053R8 and the NUCLEO-F401RE platforms with three development environments: IAR Embedded Workbench for ARM, RealView Microcontroller Development Kit (MDK-ARM), and System Workbench for STM32 (SW4STM32).

3.4 APIs

Detailed function and parameter descriptions for the user-APIs are compiled in an HTML file in the software package Documentation folder.

4 System setup guide

4.1 Hardware description

4.1.1 STM32 Nucleo platform

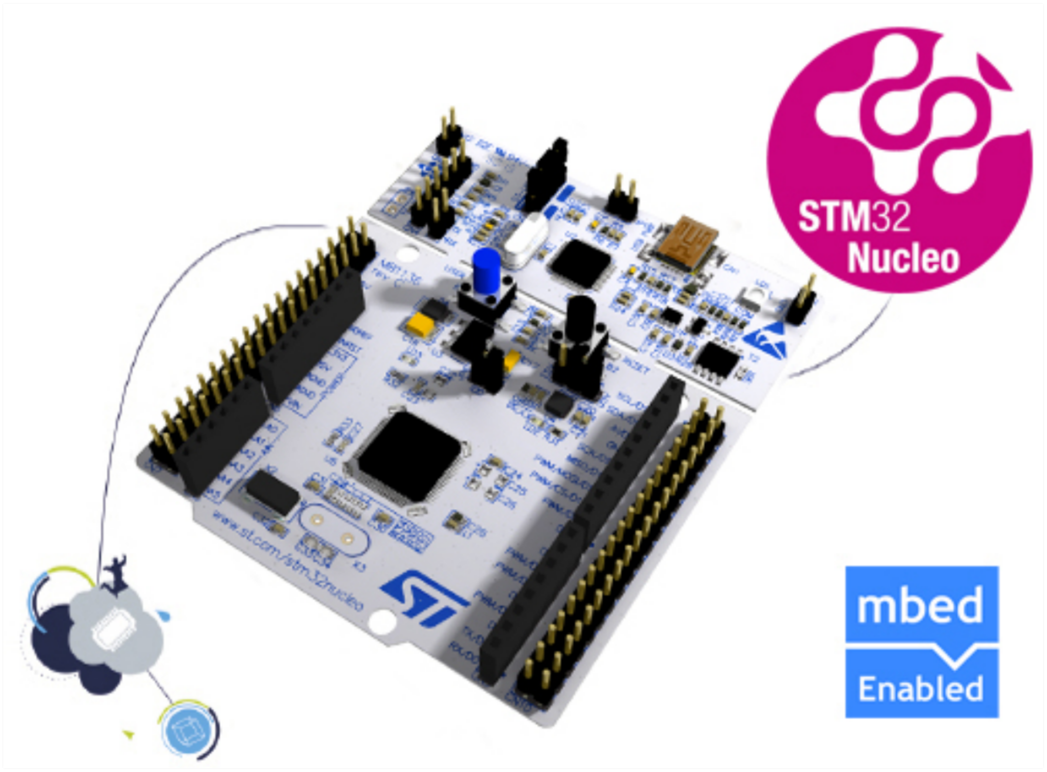
STM32 Nucleo development boards provide an affordable and flexible way for users to test solutions and build prototypes with any STM32 microcontroller line.

The Arduino™ connectivity support and ST morpho connectors make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide range of specialized expansion boards to choose from.

The STM32 Nucleo board does not require separate probes as it integrates the ST-LINK/V2-1 debugger/ programmer.

The STM32 Nucleo board comes with the comprehensive STM32 software HAL library together with various packaged software examples.

Figure 4. STM32 Nucleo board



Information regarding the STM32 Nucleo board is available at www.st.com/stm32nucleo

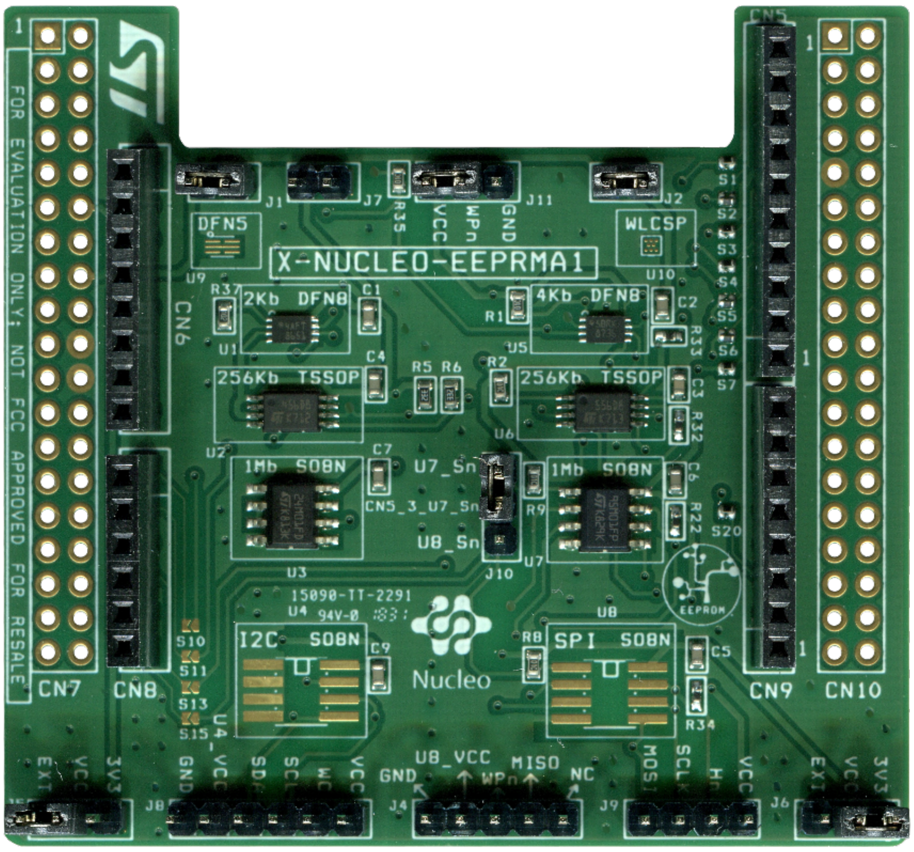
4.1.2 X-NUCLEO-EEPRMA1 expansion board

The X-NUCLEO-EEPRMA1 expansion board is designed for M24xx I²C and M95xx SPI EEPROM for data reading and writing.

The expansion board acts as an external storage device that can be used to store data such as manufacturing traceability, calibration, user setting, error flags, data log and monitoring data to make applications more flexible and accurate.

The X-NUCLEO-EEPRMA1 expansion board is compatible with the Arduino UNO R3 connector pin assignment and can be easily plugged to any STM32 Nucleo development board. You can mount the ST morpho connectors if required.

Figure 5. X-NUCLEO-EEPRMA1 expansion board



4.2 Hardware setup

The following hardware components are needed:

1. One STM32 Nucleo development platform (NUCLEO-F401RE or NUCLEO-L053R8)
2. One X-NUCLEO-EEPRMA1 expansion board
3. One USB type A to Mini-B USB cable to connect the STM32 Nucleo to the PC

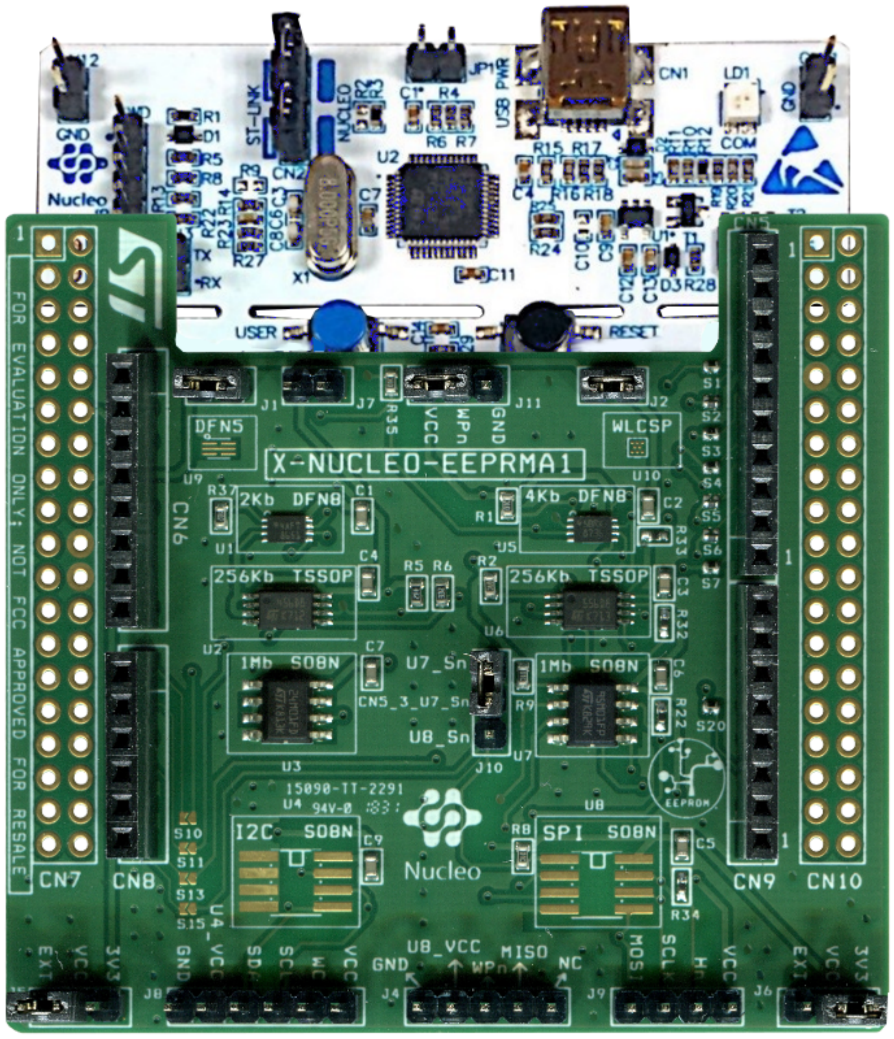
4.2.1 STM32 Nucleo and X-NUCLEO-EEPRMA1 expansion board setup

The STM32 Nucleo board integrates the ST-LINK/V2-1 debugger/programmer.

The developer can download the ST-LINK/V2-1 USB driver by looking for the STSW-LINK008 software on www.st.com.

The X-NUCLEO-EEPRMA1 expansion board can be easily connected to the STM32 Nucleo through the Arduino UNO R3 extension connector as shown below.

Figure 6. **X-NUCLEO-EEPRMA1 expansion board plugged to an STM32 Nucleo board**



4.3 Software setup

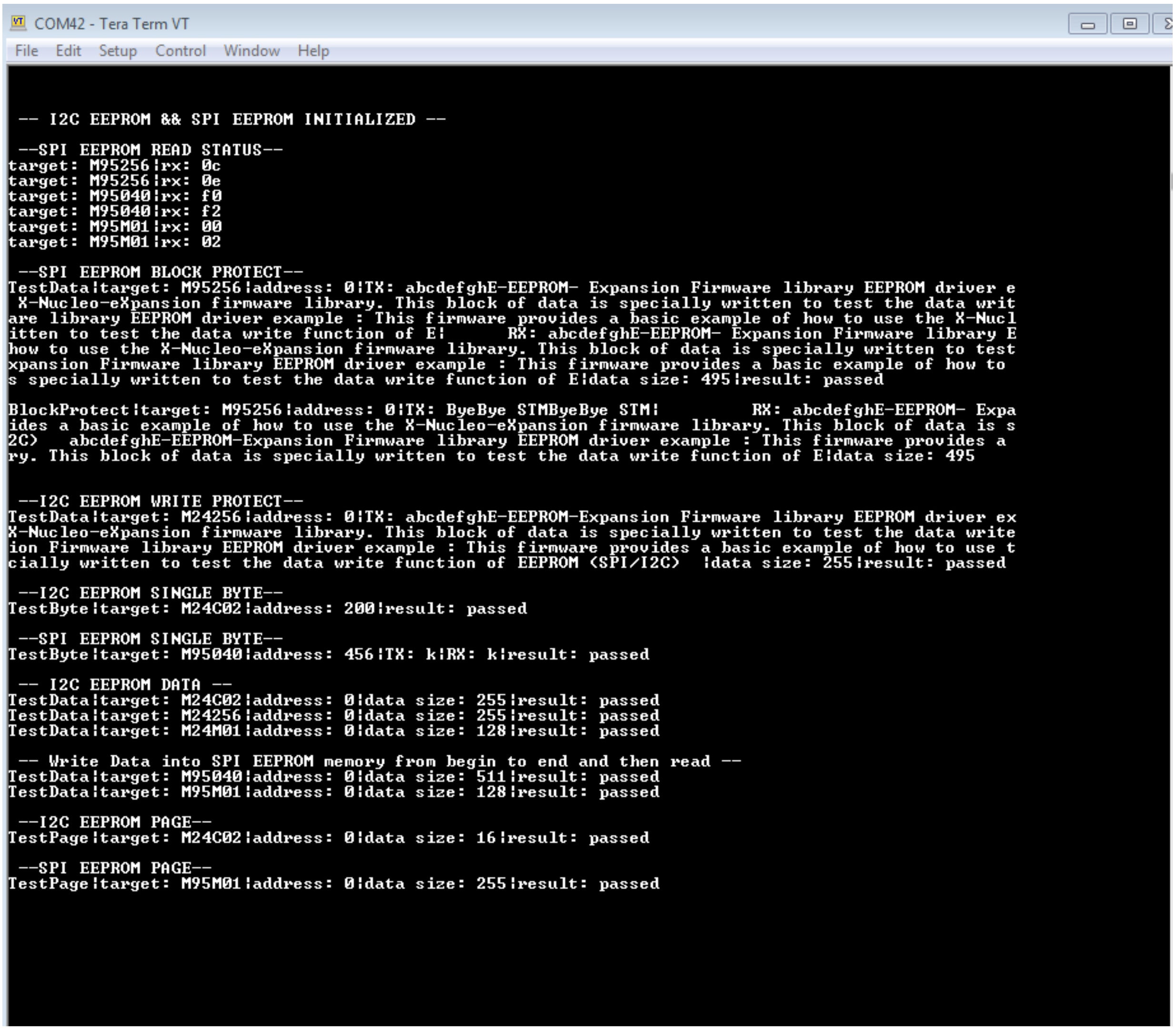
The following software components are needed for a suitable development environment for creating applications for the STM32 Nucleo equipped with the X-NUCLEO-EEPRMA1 expansion board:

- X-CUBE-EEPRMA1 expansion for STM32Cube.
- One of the following development tool-chain and compilers:
  - Keil RealView Microcontroller Development Kit (MDK-ARM-STM32) + ST-LINK
  - IAR Embedded Workbench for ARM (IAR-EWARM) + ST-LINK
  - OpenSTM32 System Workbench for STM32 (SW4STM32) + ST-LINK

The package includes samples the developer can use to start experimenting with the code:

1. a standard initialization of the I²C and SPI EEPROM
2. a sample containing
   - block protect, which reads the status after "write enable" and "write disable" for the SPI EEPROM
   - write protection feature for the I²C EEPROM
3. the third sample writes the complete memory and reads the information received

Figure 7. Console log



```
 -- I2C EEPROM && SPI EEPROM INITIALIZED --

 --SPI EEPROM READ STATUS--
target: M95256¦rx: 0c
target: M95256¦rx: 0e
target: M95040¦rx: f0
target: M95040¦rx: f2
target: M95M01¦rx: 00
target: M95M01¦rx: 02

 --SPI EEPROM BLOCK PROTECT--
TestData¦target: M95256¦address: 0¦TX: abcdefghE-EEPROM- Expansion Firmware library EEPROM driver e
 X-Nucleo-eXpansion firmware library. This block of data is specially written to test the data writ
are library EEPROM driver example : This firmware provides a basic example of how to use the X-Nucl
itten to test the data write function of E¦      RX: abcdefghE-EEPROM- Expansion Firmware library E
how to use the X-Nucleo-eXpansion firmware library. This block of data is specially written to test
xpansion Firmware library EEPROM driver example : This firmware provides a basic example of how to
s specially written to test the data write function of E¦data size: 495¦result: passed

BlockProtect¦target: M95256¦address: 0¦TX: ByeBye STMByeBye STM¦         RX: abcdefghE-EEPROM- Expa
ides a basic example of how to use the X-Nucleo-eXpansion firmware library. This block of data is s
2C)   abcdefghE-EEPROM-Expansion Firmware library EEPROM driver example : This firmware provides a
ry. This block of data is specially written to test the data write function of E¦data size: 495


 --I2C EEPROM WRITE PROTECT--
TestData¦target: M24256¦address: 0¦TX: abcdefghE-EEPROM-Expansion Firmware library EEPROM driver ex
X-Nucleo-eXpansion firmware library. This block of data is specially written to test the data write
ion Firmware library EEPROM driver example : This firmware provides a basic example of how to use t
cially written to test the data write function of EEPROM (SPI/I2C)  ¦data size: 255¦result: passed

 --I2C EEPROM SINGLE BYTE--
TestByte¦target: M24C02¦address: 200¦result: passed

 --SPI EEPROM SINGLE BYTE--
TestByte¦target: M95040¦address: 456¦TX: k¦RX: k¦result: passed

 -- I2C EEPROM DATA --
TestData¦target: M24C02¦address: 0¦data size: 255¦result: passed
TestData¦target: M24256¦address: 0¦data size: 255¦result: passed
TestData¦target: M24M01¦address: 0¦data size: 128¦result: passed

 -- Write Data into SPI EEPROM memory from begin to end and then read --
TestData¦target: M95040¦address: 0¦data size: 511¦result: passed
TestData¦target: M95M01¦address: 0¦data size: 128¦result: passed

 --I2C EEPROM PAGE--
TestPage¦target: M24C02¦address: 0¦data size: 16¦result: passed

 --SPI EEPROM PAGE--
TestPage¦target: M95M01¦address: 0¦data size: 255¦result: passed
```

Revision history

Table 2. **Document revision history**

Date	Version	Changes
04-Oct-2018	1	Initial release.

Contents

List of tables

List of figures