AN6228

Application note

How to implement XSPI read-while-write feature on STM32 MCUs

Introduction

Given the increasing demand in automotive and IoT (Internet of Things) applications, flash memory featuring read-while-write (RWW) capability is emerging as a significant trend, which offers an ideal solution for over-the-air (OTA) update applications, allowing the execution of read and write operations simultaneously without stalling or waiting. In turn, this meets strict real-time and functionality requirements, reduces latency, and enhances throughput.

This application note explains how to implement the read-while-write using the STM32 devices that embed two XSPI IPs coupled with an IOManager XSPIM.

Table 1. Applicable products

Type	Applicable products
Microcontrollers	STM32HR7/7S7, STM32H7R3/7S3 lines
	STM32N6 series

1 General information

This solution is only valid for products that embed two XSPI IPs with an IOManager (XSPIM).

1.1 Reference documents

Table 2. List of documents

Reference	Name/address	Title
[1]	RM0477	STM32H7Rx/Sx Arm®-based 32-bit MCUs reference manual
[2]	RM0486	STM32N6 Series reference manual
[3]	AN5050	Getting started with Octo-SPI, Hexadeca-SPI, and XSPI interfaces on STM32 MCUs

1.2 List of acronyms

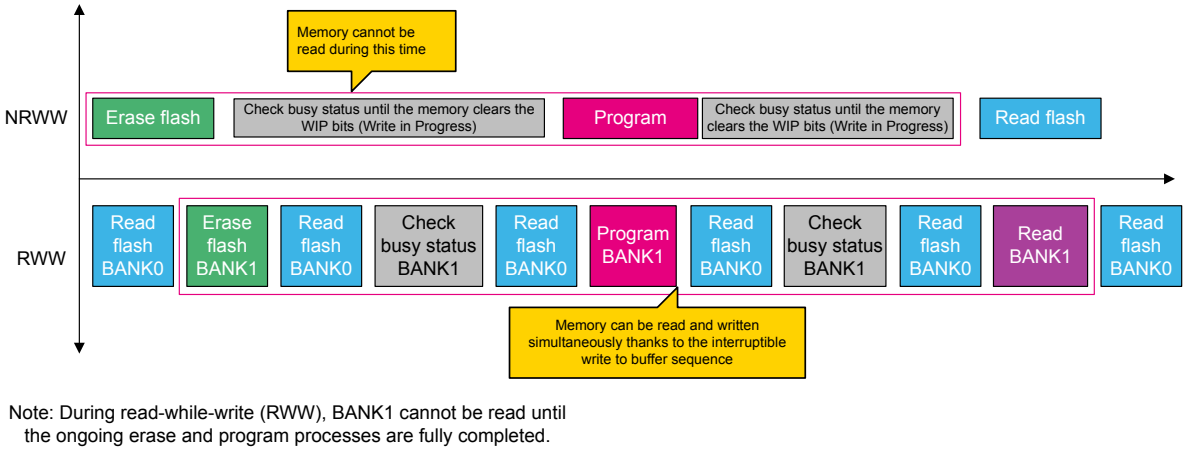
Table 3. List of acronyms

Term	Description
XSPI	Extended-SPI interface
XSPIM	XSPI I/O manager
RWW	Read-while-write
NRWW	Non read-while-write
XIP	Execute in place
WREN	Write enable command (06 h)
WEL	Write enable latch
WRBI	Write buffer initial command (22 h)
WRCT	Write buffer continue command (24 h)
WRCF	Write buffer confirm command (31 h)
RDBUF	Page buffer read command (25 h)
OTA	Over-the-air
IoT	Internet of things

2 Read-while-write feature (RWW)

The read-while-write (RWW) feature can execute the read and write (program or erase) operations simultaneously without stalling the (AXI/AHB) bus. Traditional flash memory typically does not support concurrent read and write, because commands are sent through the XSPI bus in sequence. Erase and program operations require waiting for the busy status to be ready, which usually takes a significant amount of time. This means the host is not able to read or write during that moment. However, with the RWW feature, read and write operations can be performed concurrently. This allows XIP code (fetching instruction) to run from one memory partition while write operations are in progress on another.

Figure 1. NRWW versus RWW



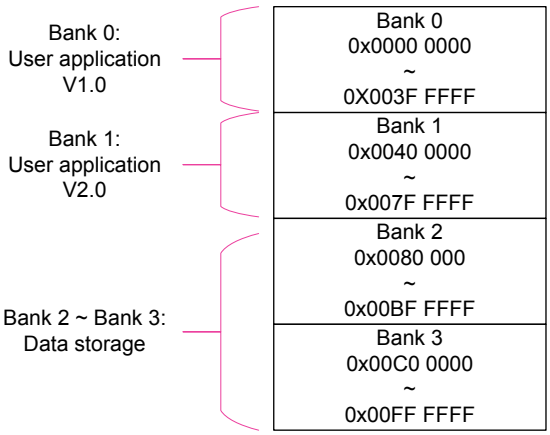
Note: During read-while-write (RWW), BANK1 cannot be read until the ongoing erase and program processes are fully completed.

To implement the read-while-write (RWW) feature, the flash memory must inherently have this capability and be designed with multiple banks. Consequently, some memory vendors have developed advanced flash memory technologies that incorporate RWW capabilities. For example, Macronix™ RWW flash memory offers a multibank architecture which provides simultaneous read-while-write capability that allows read access from one memory while writing to another one.

With this functionality, the flash memory can be accessed even during execute in place (XIP) operations. This allows the system to continue executing the code directly from one region of the flash memory while simultaneously performing read, write, or update operations in a different region. Such capability is especially valuable for applications that must remain responsive and operational while handling tasks like background firmware updates or data logging, ensuring seamless performance without interruptions.

The figure below illustrates an example of the flash partition of the Macronix™ RWW flash memory MX25UW51245G.

Figure 2. MX25UW51425G memory organization example



In such a case, Bank 0 can be used to execute user application V1.0 while Bank 1 is used to store the new user application version, and Banks 2 and 3 are used to store data. This scheme enables the system to continue executing during an OTA update process, which happens in the background. Furthermore, the system can roll back to its original firmware in the event of an OTA failure or data corruption.

The RWW feature enables simultaneous programming and reading of flash memory without causing stalling. There are some principles for the RWW operations:

- All the banks are independent, but the one that is under programming can not read or write until the end of the programming.
- Program and erase operation can be conducted in one bank. However, in the same bank, no read operation is allowed.
- Sequential read operation can occur in any of the three banks which are not programmed or erased.

In addition to the RWW capability, Macronix™ flash offers a sequence called interruptible write-to-buffer during programming. This sequence provides the advantage of optimizing the program operation compared to the classical program command. This can lead to improved overall system performance.

For further details about the interruptible write-to-buffer sequence, refer to the memory datasheet.

In this application note, we demonstrate how to implement the read-while-write (RWW) feature using Macronix™ interruptible write-to-buffer sequence with the STM32H7Rx/7Sx microcontroller.

3 Read-while-write (RWW) implementation on STM32 MCUs

This section explains how to implement the RWW feature using the XSPI controller.

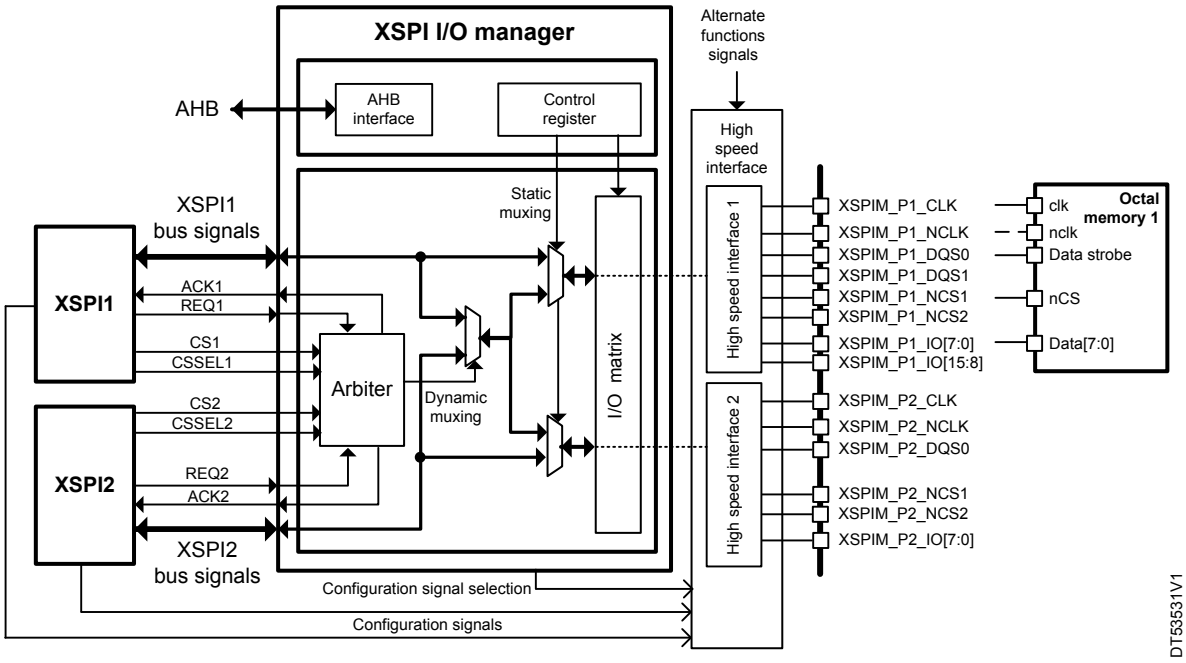
3.1 Hardware requirements

To implement the RWW feature, the hardware requirements are:

- STM32 products equipped with two XSPIs instances and IOManager (XSPIM), for example, the STM32H7Rx/7Sx product lines.
- External octo flash memory supporting RWW feature, multi-bank architecture (for example, MX66UW1G45G or MX25UW12845G).

The hardware implementation is shown in the figure below.

Figure 3. Hardware implementation



The two XSPIs are used to interface with the external octo flash memory in octo mode. The first instance is used to execute the code from Bank 0, and the second instance is used to update data or code in Bank 1 because XSPIM manages the access to the external memory.

The XSPIM is a low-level interface that allows the user to set a fully programmable pre-mapping of the XSPI signals.

Three modes of pre-mapping are supported:

- Direct mode: This is the baseline mode, where each XSPI independently drive the corresponding port (XSPI1 mapped to Port 1, with XSPI2 mapped to Port 2).
- Swapped mode: In this mode, the ports are swapped (XSPI1 mapped to Port 2, with XSPI2 mapped to Port 1).
- Multiplexed mode: When this mode is used, the XSPI are time-multiplexed over the same bus. Ownership of the bus is obtained through a request/acknowledge protocol with REQ/ACK signals.

Two possible configurations are available:

- XSPI1 and XSPI2 both mapped to Port 1, with arbitration.
- XSPI1 and XSPI2 both mapped to Port 2, with arbitration.

Note: To benefit from the RWW feature, the XSPIM must be configured in multiplexed mode.

The arbiter in the IO manager selects XSPI1 or XSPI2 controller to take the ownership of the multiplexed port bus according to the existing transfer requests and status of the two MAXTRAN fairness counters.

There is no notion of priority: each controller with a pending request takes ownership.

The fairness counters (MAXTRAN) of each XSPI controller are used to manage the maximum duration for which a given XSPI uses the bus ownership. This parameter ensures the maximum bus access latency for the other XSPI. When one XSPI releases the bus, a round-robin arbitration phase occurs, during this phase, the bus is given to another XSPI that is requesting the bus. The NCS pin is released every MAXTRAN +1 clock cycles when the other XSPI request the access to the bus.

Note: The MAXTRAN parameter is configured through the MAXTRAN field in the XSPI_DCR3 register. its value must be greater than the minimum size of a transaction including, command, address, alternate byte if any, dummy phases and at least one access to the memory. Otherwise there is no transfer from the corresponding host controller. Moreover, it needs to be properly set to optimize the time sharing (to not block the write operation and code execution) depending on application constraints.

The minimum number of cycles needed to switch from an XSPI to another can be configured by the REQ2ACK_TIME field in the XSPIM_CR register. This guarantees a latency between the falling edge of the REQ signal of the active XSPI (the active one release the bus), and the rising edge of the ACK signal to the requesting XSPI (the bus is granted to the requesting one).

In the example project described in Section 4: Read-while-write (RWW) examples, MAXTRAN is set to 22. This means that the maximum time allowed for the XSPI before switching to another one is calculated as follows: (MAXTRANS + 1) × 5ns = 23 × 5 ns = 115 ns. Here, WRBI (Write buffer intial command) consists of a command (2 bytes), an address (4 bytes), and data (16 bytes).

Req2AckTime is set to 10, which requiring at least 8 clock cycles, as indicated by the AC timing in the memory datasheet. This is calculated by the formula below:

Clock Cycles CS Deselect Time = CS Deselect Time (ns)/ Clock Period (ns)

3.2 Pins assignment

When two XSPI instances are used, each XSPI delivers one NCS signal together with a CSSEL control signal. The CSSEL signal selects which of the two outputs (NCS1, NCS2) is active. In this implementation, only one NCS is used in the output.

The table below shows the pins used:

Table 4. XSPIM Input/Output pins

Pin name(1)	Signal type	Description
XSPIM_Px_CLK	Output	XSPI clock
XSPIM_Px_NCS	Output	Chip selection for the memory
XSPIM_Px_DQS	Input/output	Data strobe/write mask signal from/to the memory
XSPIM_Px_IOn (n = 0 to 7)	Input/output	XSPI data pins

1. *X = 1 or 2*

For further details about the XSPI and XSPIM, refer to the STM32 product reference manuals.

3.3 Software requirements

To maintain the XIP + RWW feature, the following routine must be used:

1. Configure XSPI2 in memory mapped mode, which allows the MCUs to execute the code directly from the external flash memory.
2. Configure XSPI1 in indirect write mode.
3. Configure XSPIM in multiplexed mode, then map XSPI1 and XSPI2 both to Port 1 or to Port 2.
4. Configure the HPDMA1 for XSPI1 to manage the write operation.

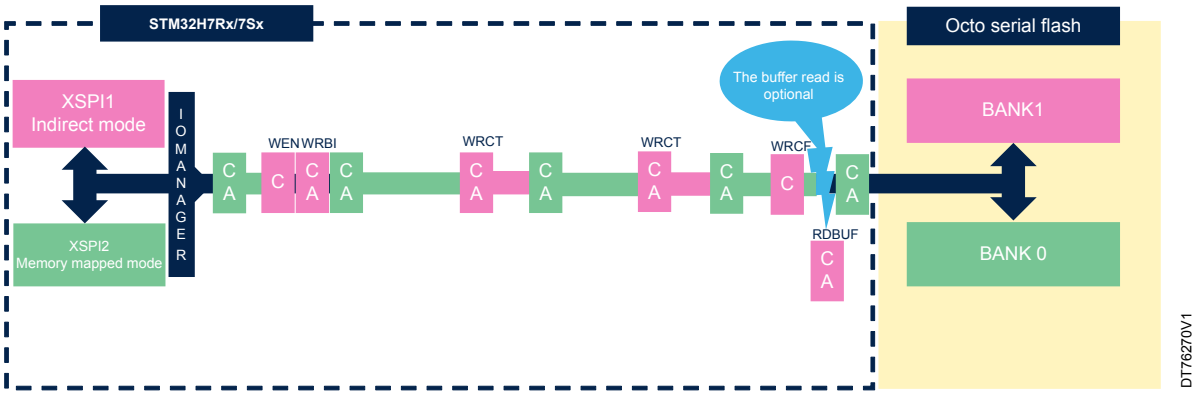
To perform a write operation, the following commands must be managed by the XSPI1:

1. Send a WREN command first to the external flash memory to set the WEL bit in its status register.
2. Send a WRBI command with 0 data to initiate the write buffer to program.
3. Send a WRCT command with all the data using the DMA transfer.
4. Send a WRCF command to initiate an automatic program operation to write the page buffer into the memory.

A read buffer can be optionally inserted by sending the command RDBUF before the WRCF command to read the internal memory buffer, and verify the data integrity.

The flowchart below illustrates the above process:

Figure 4. RWW flowchart



Using the DMA controller to send the WRCT command is strongly recommended. This allows the hardware to automatically re-execute the command in case the transfer has been interrupted.

Note: Chip selection is released during the switching of the ownership of the bus to XSPI2. A WRBI command must be sent only once, otherwise, it flushes the internal memory FIFO. Any WRCT or WRCF command without a preceding WRBI command would be ignored. A WRCF command must be only sent with a command phase. For read operation, any RDBUF without a WRBI command returns 0xFF as data output.

4 Read-while-write (RWW) examples

This section describes the example XSPI_NOR_ReadWhileWrite_DTR available in the STM32CubeH7RS (https://github.com/STMicroelectronics/STM32CubeH7RS/blob/main/Projects/STM32H7S78-DK/Examples/XSPI/XSPI_NOR_ReadWhileWrite_DTR)

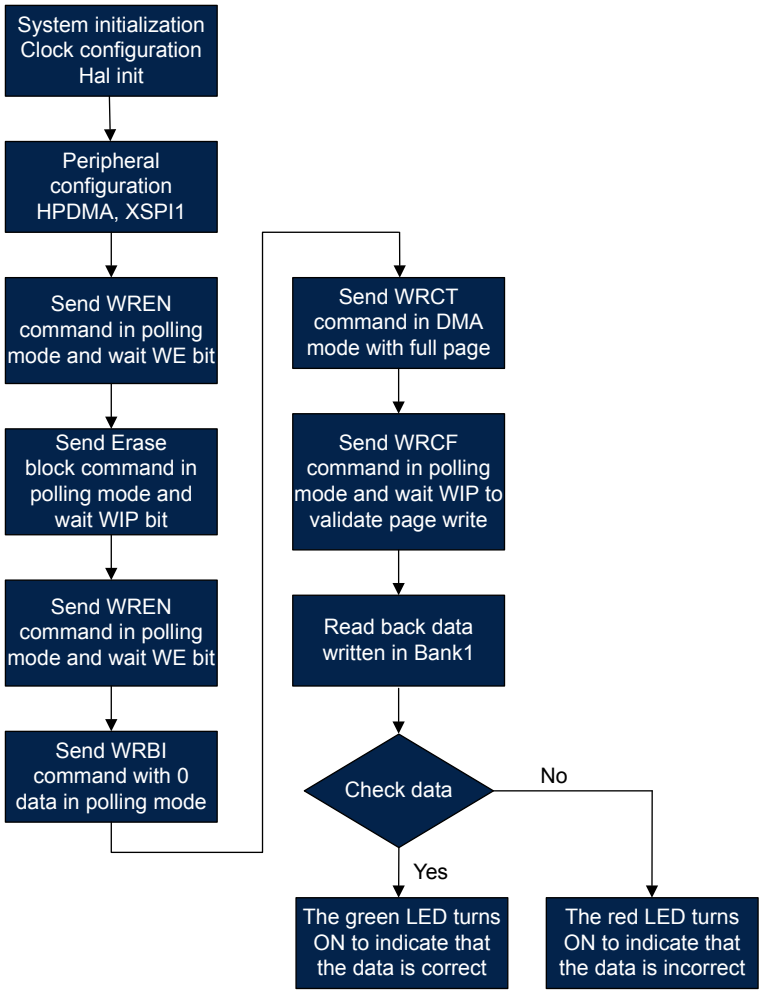
The example demonstrates writing and reading data on XSPI NOR Memory Bank 1 while executing code from Bank 0. This is achieved using Macronix™ interruptible write-to-buffer sequence, which includes the WRBI, WRCT, and WRCF commands.

The project contains two sub-project:

- Boot sub-project: manages the startup of the application
  - System initialization (MPU, I/D-caches, system clock, and GPIOs)
  - Configuration of the HPDMA controller for XSPI1
  - Configuration of the external memory interface
  - External memory initialization via external memory middleware
  - Jump to application
- Appli sub-project: End user application executed from external NOR Flash bank 0.

The executed code from Bank 0 is illustrated in the below flowchart:

Figure 5. Update Bank1 flowchart



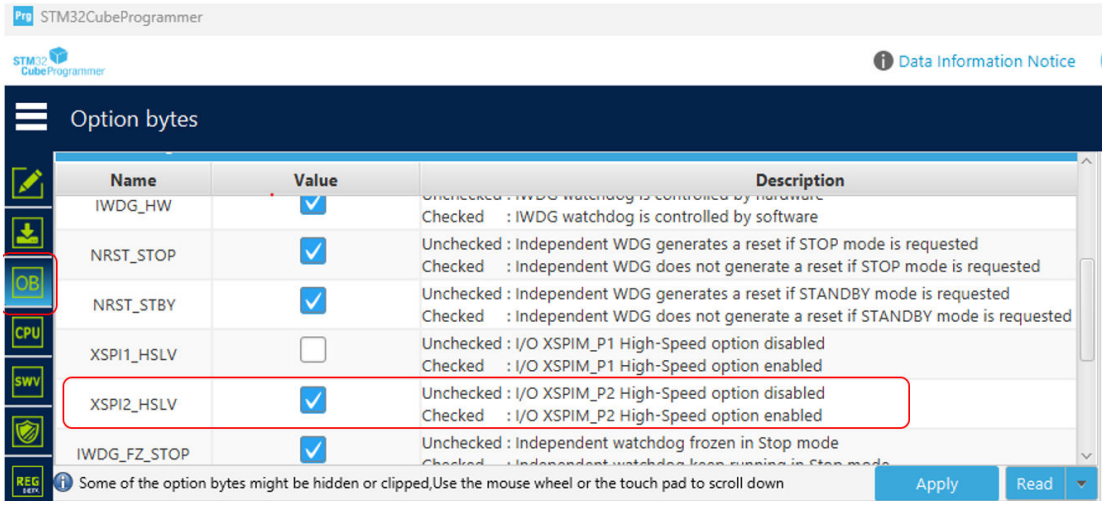
This example has been tested on the STM32H7S78-DK (MB1736) board, which is equipped with MX66UW1G45G flash memory.

In order to make the program work, you must do the following:

1. Configure the Option Bytes to Enable IO HSLV.

Note: The XSPI has an independent power domain, which operates at 1.8 V. Therefore, the HSLV (High-Speed Low-Voltage) feature must be utilized. HSLV is a characteristic of certain I/O ports that allows them to operate at higher speeds while using lower voltage levels.
Open STM32CubeProgrammer and under OB > User Configuration 1, check the XSPI2_HSLV option as shown in the figure below.

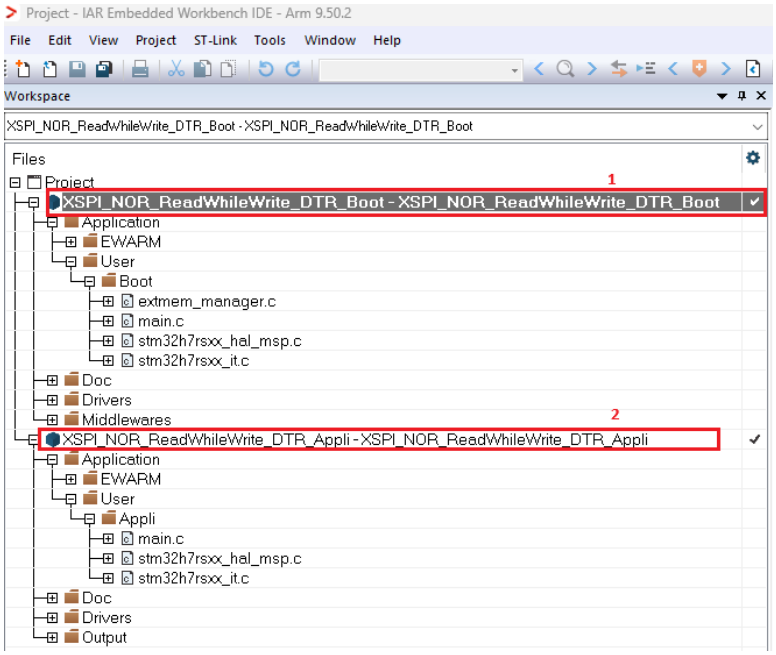
Figure 6. Configure option bytes



DT76289V1

2. Open the XSPI_NOR_ReadWhileWrite_DTR Example with your preferred toolchain , for example, IAR™ (EWARM) v9.50.2. Select first "XSPI_NOR_ReadWhileWrite_DTR_Boot" as an active project and rebuild all the files. Then select "XSPI_NOR_ReadWhileWrite_DTR_Appli" as active project to rebuild all the files.

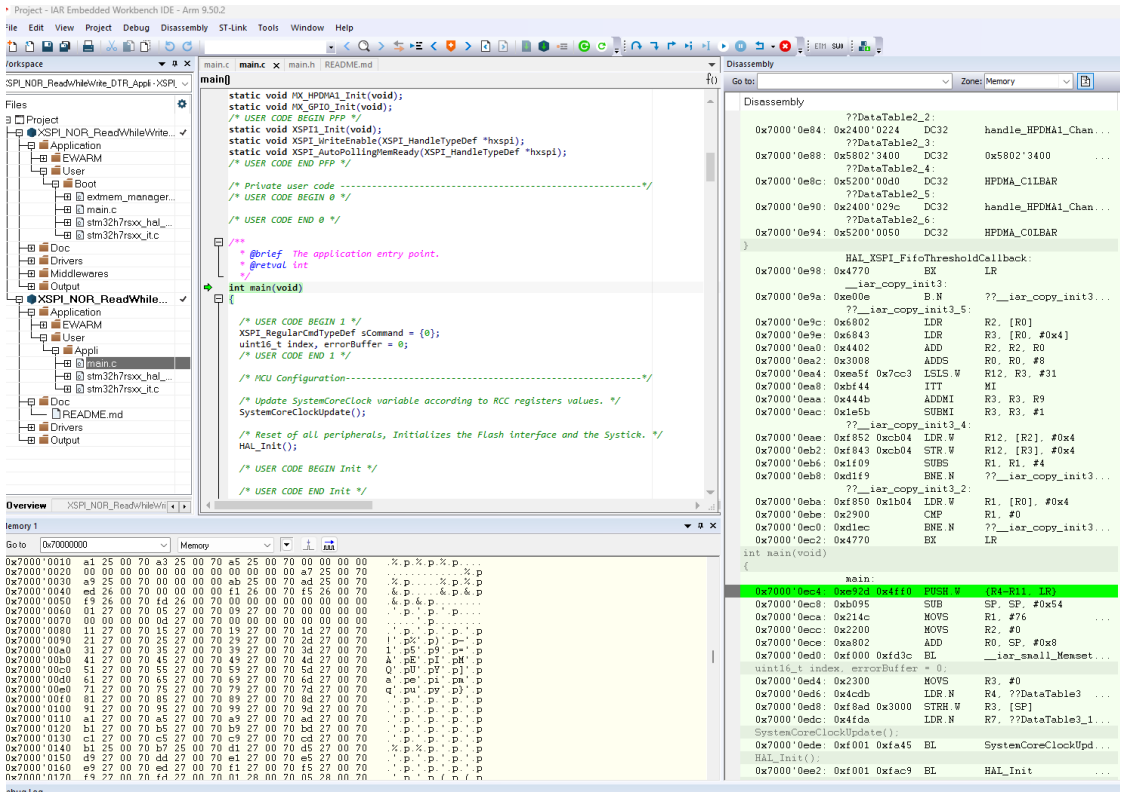
Figure 7. EWARM project structure



DT77873V1

3. Download the "XSPI_NOR_ReadWhileWrite_DTR_Boot" to the target without launching a full debug session. Then, download and debug the "XSPI_NOR_ReadWhileWrite_DTR_Appli" . The workbench window is displayed as shown below:

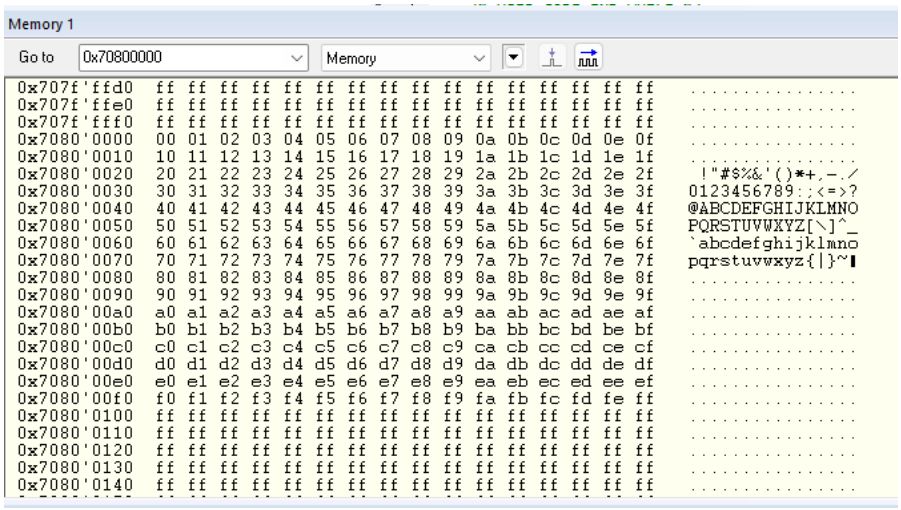
Figure 8. IAR workbench after downloading the project



DT77874V1

The "memory view" below shows that the data has been written to flash memory Bank1.

Figure 9. Memory view



DT77875V1

Revision history

Table 5. **Document revision history**

Date	Version	Changes
25-Jun-2025	1	Initial release.

Contents

1 General information 2
1.1 Reference documents 2
1.2 List of acronyms 2
2 Read-while-write feature (RWW) 3
3 Read-while-write (RWW) implementation on STM32 MCUs 5
3.1 Hardware requirements 5
3.2 Pins assignment 6
3.3 Software requirements 6
4 Read-while-write (RWW) examples 8
Revision history 11
List of tables 13
List of figures 14

List of tables

Table 1. Applicable products . . . 1
Table 2. List of documents . . . 2
Table 3. List of acronyms . . . 2
Table 4. XSPIM Input/Output pins . . . 6
Table 5. Document revision history . . . 11

List of figures

Figure 1. NRWW versus RWW . . . 3
Figure 2. MX25UW51425G memory organization example . . . 4
Figure 3. Hardware implementation . . . 5
Figure 4. RWW flowchart . . . 7
Figure 5. Update Bank1 flowchart . . . 8
Figure 6. Configure option bytes . . . 9
Figure 7. EWARM project structure . . . 9
Figure 8. IAR workbench after downloading the project . . . 10
Figure 9. Memory view . . . 10

IMPORTANT NOTICE – 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 acknowledgment.

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, 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.

© 2025 STMicroelectronics – All rights reserved