AN5188

Application note

External memory code execution on STM32F7x0 Value line and STM32H750 Value line MCUs

Introduction

There is an increased demand for applications able to support new and complex features, and as a consequence there is an increased demand for devices with a bigger Flash-memory area.

The use of external Flash memory provides higher storage capabilities with comparable performance levels while supplying a cost efficient solution for the demand of an increased Flash-memory area.

The STM32F7x0 Value line and the STM32H750 Value line devices respond to the market demand with a reduced inner Flash-memory area.

This application note describes the steps needed to build an application with code execution from external memory on these Value line devices.

It provides details on how to boot from internal Flash memory, and then jump to user-application execution from an external memory.

Related documents

Available from the STMicroelectronics website at www.st.com:

- *STM32Cube MCU Package for STM32F7 Series with HAL, low-layer drivers and dedicated middleware* databrief (DB2601)
- *STM32Cube MCU Package for STM32H7 Series with HAL and dedicated middleware* databrief (DB3259)
- *STM32F75xxx and STM32F74xxx advanced Arm®-based 32-bit MCUs* reference manual (RM0385)
- *STM32H743/753 advanced ARM®-based 32-bit MCUs* reference manual (RM0433)
- *STM32F7 Series system architecture and performance* application note (AN4667)
- *Quad-SPI (QSPI) interface on STM32 microcontrollers* application note (AN4760)
- *Getting started with STM32H7x3 hardware development* application note (AN4938)
- *Getting started with STM32F7 Series MCU hardware development* application note (AN4661)
- *STM32CubeProgrammer software description* user manual (UM2337)

1 General information

This document applies to Arm®-based devices.

Note: *Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.*

2 External memory code execution overview

2.1 External memory code execution principle

The STM32CubeF7 v1.12.0 and the STM32CubeH7 v1.3.0 firmware packages provide several applications to demonstrate how to boot from internal Flash memory and how to configure the external memories and jump to user application (located on the external memory). Two possible use cases are available: XiP and BootROM.

- The XiP use case is intended for *"eXecute in Place"* from external Flash memory (QSPI or FMC-NOR Flash memory). The user-application code should be linked with the target execution memory-address (external QSPI or FMC-NOR Flash memory).
- The BootROM use case is intended to demostrate how to boot from internal Flash memory, configure the external RAM memories (SDRAM or SRAM), copy user-application binary from the code storage area (an SDCARD or an SPI-Flash memory) to the external SDRAM or external SRAM, and then jump to the user application. The user-application code should be linked with the target execution memory address (external SDRAM or SRAM).

The applications described in the table below are available on the firmware package under \Applications \ExtMem_CodeExecution for the following boards:

- STM32F723E-Discovery board for the STM32F730 devices
- STM32F756G_EVAL board for the STM32F750 devices
- STM32H743I_EVAL board for the STM32H750 devices.

Table 1. **Application details**

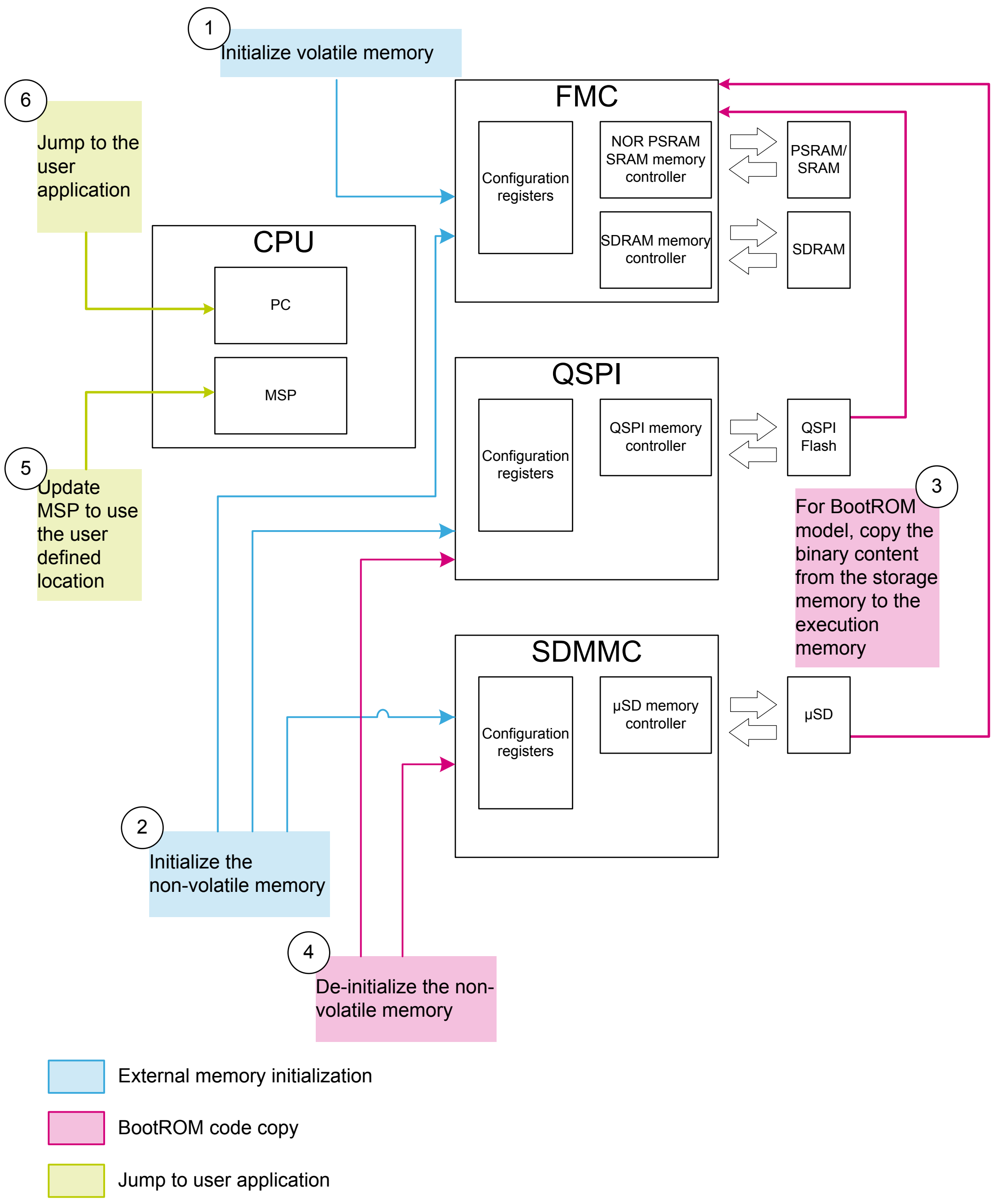
Application	Description
ExtMem_Boot	Shows how to boot from internal Flash memory, configure external memories and then jump to user application located on external memory. The user can select QSPI Flash memory, FMC-NOR Flash memory, external SDRAM or external SRAM for code execution.
ExtMem_Application\LedToggling	Sample application running from external Flash memory (QSPI Flash memory or FMC-NOR Flash memory), external SRAM or external SDRAM
ExtMem_Application\FreeRTOS	Sample FreeRTOS application with execution from external Flash memory (QSPI Flash memory or FMC-NOR Flash memory), external SRAM or external SDRAM

The *External memory boot* application is in charge of initializing the required resources to make the external memories available and ready to use. This application initializes the required resources as per the user configuration (see Section 3.3 Configuration).

The *External memory boot* application must setup the main stack pointer and configure the application to be executed on external memory. This type of boot schema enables the support of sizable user applications.

The *External memory boot* application ensures that any resources that are no longer needed after the setup phase are reset or free before jumping to the user application. The figure below presents an overview of this boot schema.

Figure 1. **External memory code boot schema**

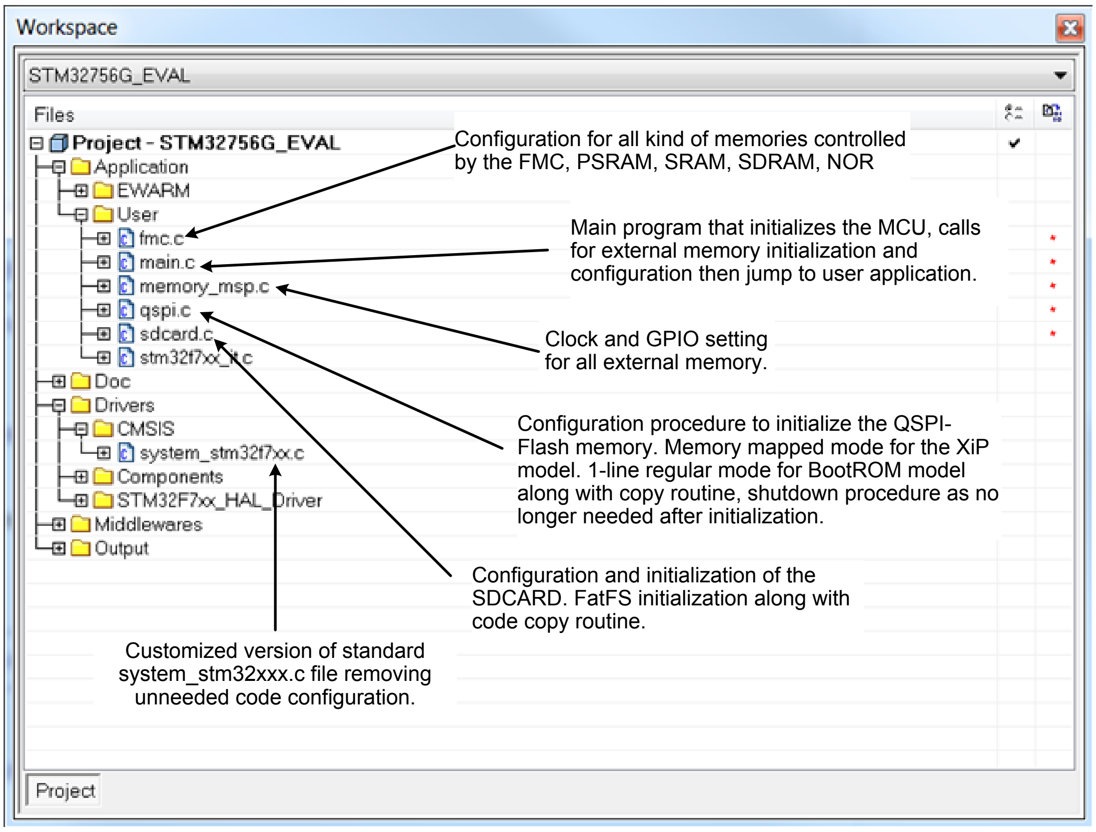

1
Initialize volatile memory
6
Jump to the user application
FMC
Configuration registers
NOR PSRAM SRAM memory controller
SDRAM memory controller
PSRAM/ SRAM
SDRAM
CPU
PC
MSP
QSPI
Configuration registers
QSPI memory controller
QSPI Flash
3
For BootROM model, copy the binary content from the storage memory to the execution memory
5
Update MSP to use the user defined location
SDMMC
Configuration registers
µSD memory controller
µSD
2
Initialize the non-volatile memory
4
De-initialize the non-volatile memory
External memory initialization
BootROM code copy
Jump to user application

2.2 External memory boot application description

The *External memory boot* application contains a set of source files on the STM32CubeF7/H7 package which is tailored to match the supported configuration for each hardware platform.

The figure below shows an example of the superset of all files for all the supported configurations.

Figure 2. External memory boot application superset of source files



Configuration for all kind of memories controlled by the FMC, PSRAM, SRAM, SDRAM, NOR

Main program that initializes the MCU, calls for external memory initialization and configuration then jump to user application.

Clock and GPIO setting for all external memory.

Configuration procedure to initialize the QSPI-Flash memory. Memory mapped mode for the XiP model. 1-line regular mode for BootROM model along with copy routine, shutdown procedure as no longer needed after initialization.

Configuration and initialization of the SDCARD. FatFS initialization along with code copy routine.

Customized version of standard system_stm32xxx.c file removing unneeded code configuration.

3 Supported boot model

The application supports two types of execution models:

- Execute in place support (XiP support)
- BootROM support

The users must select the configuration matching their needs by tuning the *memory.h* header file.

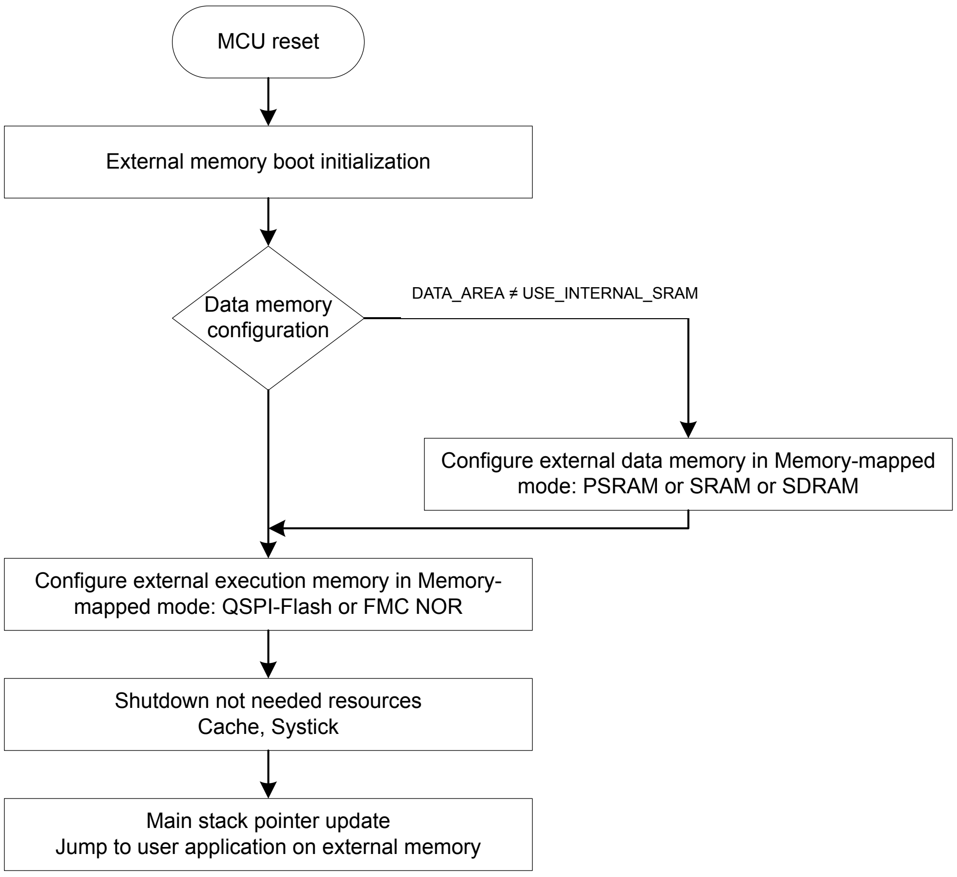
3.1 Execute in place (XiP) support

The XiP model is based on code execution directly from the external non-volatile memory that is used for code storage. This execution model requires memory-mapped support to grant the CPU with direct access to the executed-code user application. The XiP model is available on external NOR/QSPI Flash memory through the FMC/QSPI interfaces.

Based on the user configuration in the *memory.h* file, the *External memory boot* application configures **one** of the following volatile memories: SDRAM, SRAM, PSRAM or internal SRAM. In this model the volatile memory is used for data only.

The following flowchart illustrates the operational flow for the XiP model.

Figure 3. **XiP model operational flow**


MCU reset
External memory boot initialization
Data memory configuration
DATA_AREA ≠ USE_INTERNAL_SRAM
Configure external data memory in Memory-mapped mode: PSRAM or SRAM or SDRAM
Configure external execution memory in Memory-mapped mode: QSPI-Flash or FMC NOR
Shutdown not needed resources Cache, Systick
Main stack pointer update Jump to user application on external memory

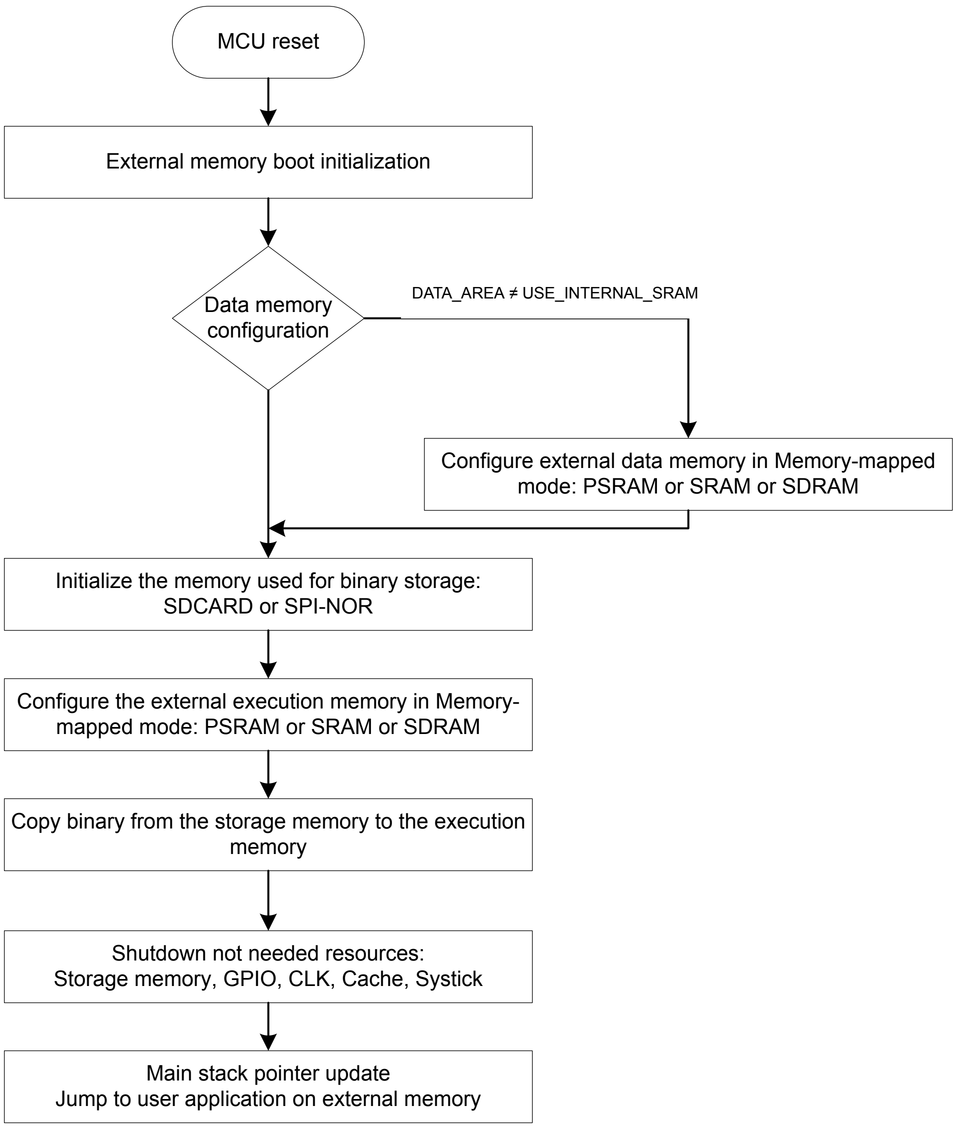
3.2 BootROM support

The BootROM model is based on code execution from a chosen volatile memory. This execution model is suitable when binary data is stored in a memory that has no memory-mapped interface (like for SDCARD). This model is also suitable when binary data is stored in a memory with low throughput (like for SPI-NOR (emulated using QSPI with 1- line)).

Based on the user configuration in the *memory.h* file, the *External memory boot* application configures **two** of the following volatile memories: SDRAM, SRAM, PSRAM or internal SRAM. In this model, binary data is copied from a non-volatile memory to one volatile memory prior to the execution by the *External memory boot* application. The second volatile memory is used for data.

The following flowchart illustrates the operational flow for the BootROM model.

Figure 4. **BootROM model operational flow**


MCU reset
External memory boot initialization
Data memory configuration
DATA_AREA ≠ USE_INTERNAL_SRAM
Configure external data memory in Memory-mapped mode: PSRAM or SRAM or SDRAM
Initialize the memory used for binary storage: SDCARD or SPI-NOR
Configure the external execution memory in Memory-mapped mode: PSRAM or SRAM or SDRAM
Copy binary from the storage memory to the execution memory
Shutdown not needed resources: Storage memory, GPIO, CLK, Cache, Systick
Main stack pointer update
Jump to user application on external memory

3.3 Configuration

The user confoiguration is defined by the following *defines*:

- **DATA_AREA:** it is used to specify the volatile memory that is used for data holding. Supported memories (depending on the board used) are:
  - USE_EXTERNAL_SDRAM: external SDRAM is used for data holding
  - USE_EXTERNAL_SRAM: external SRAM is used for data holding
  - USE_EXTERNAL_PSRAM: external PSRAM is used for data holding
  - USE_INTERNAL_SRAM: internal SRAM is used for data holding
- **CODE_AREA:** it is used to specify the execution location of the user application. This area can be a volatile memory for the BootROM schemas or a non-volatile for the XiP schemas. The supported memories (depending on the hardware used) are:
  - XiP model: BINARY_AREA must be undefined:
    - USE_QSPI: QSPI Flash is used for code execution
    - USE_NOR: FMC-NOR is used for code execution
  - BootROM model: BINARY_AREA must be defined
    - USE_EXTERNAL_SDRAM: external SDRAM is used for code execution
    - USE_EXTERNAL_SRAM: external SRAM is used for code execution
    - USE_EXTERNAL_PSRAM: external PSRAM is used for code execution
    - USE_INTERNAL_SRAM: internal SRAM is used for code execution
- **BINARY_AREA:** is defined in the BootROM model only. It is used to specify the location of the binary containing the user application. Additional *defines* are needed depending on the chosen configuration. Supported memories (depending on the hardware used):
  - USE_SPI_NOR: SPI NOR Flash is used for binary storage
    - BINARY_BASE_OFFSET: offset of the binary within SPI NOR Flash
    - BINARY_SIZE: size of the binary image
  - USE_SDCARD: SDCard is used for binary storage
    - BINARY_FILENAME: name of the binary file to be executed

The user should make sure that the selected memories contain code and data to cover at least a proper user application startup. Afterwards, the user application can initialize any other memory needed.

3.4 Summary of external memories part numbers

The following table summarizes the part numbers of the external memories used versus the board and boot model. As there is not a dedicated board for devices of STM32F7x0 Value line and STM32H750 Value line , the boards (with compatible devices) that are used are:

- STM32F723E-Discovery is used to emulate the STM32F730 devices.
- STM32F756G_EVAL is used to emulate the STM32F750 devices.
- STM32H743I_EVAL is used to emulate the STM32H750 devices.

Table 2. **External memories used on each board by boot model**

Boot model	Memory	STM32F723E-Discovery with STM32F730	STM32756G_EVAL with STM32F750	STM32H743I_EVAL with STM32H750
XiP	QSPI Flash memory	MX25L51245GZ2I-08G *(Bus width : 4 lines)*	N25Q512A13GSF40E *(Bus width : 4 lines)*	Two Quad-SPI Flash MT25QL512ABB8ESF-0SIT *Or* One twin Quad-SPI Flash MT25TL01GHBB8ESF-0SIT *(Bus width : 8 lines)*
	NOR Flash memory *(on FMC)*	-	PC28F128M29EWL A *(Bus width : 16-bit)*	MT28EW128ABA1L PC-0SI T
BootROM	SPI-NOR *(emulated with QSPI 1 line)*	-	N25Q512A13GSF40E *(Bus width : 2 lines)*	Two Quad-SPI Flash MT25QL512ABB8ESF-0SIT *Or* One twin Quad-SPI Flash MT25TL01GHBB8ESF-0SIT *(Bus width : 2 lines)*
	SDCARD	-	*Native support*	Transceiver IP4856CX25/ C_Module_REV
Volatile memory	Internal SRAM	*Native support*	*Native support*	*Native support*
	External SRAM	-	IS61WV102416BL L -10ML I *(Bus width : 16-bit)*	IS61WV102416BL L -10ML I *(Bus width : 16-bit)*
	External SDRAM	-	IS42S32800G-6BL I *(Bus width : 32-bit)*	IS42S32800G-6BL I *(Bus width : 32-bit)*
	External PSRAM	IS66WV51216EBLL-55BLI *(Bus width : 16-bit)*	-	-

4 Resources constraints to be considered

Any resources that are no longer needed after initialization (interruption, ongoing transfers, unused pins) should be released before jumping to the user application. This must be done to avoid an extra power consumption and to limit any interference with the user application. Especially for the BootROM model, as the peripherals used for binary storage are no longer required, they should be reset.

The user should consider the amount of resources used by the *External memory boot* application in order to ensure that the external memory interface remains up and running. The resources constraints are linked to:

- The allocation and configuration of the pins
- The configuration of the interface (QSPI IP register should not be modified, the FMC IP register can be partially updated)
- The RCC configuration to avoid IP reset clock disabling and clock frequency/source update in a harmful way.

The pin allocation table below is provided as reference and is valid for a pin selection according to the used board. Other pin selection could be used based on the available alternate functions.

Table 3. **Pins allocated for each memory by board**

Memory/ board	STM32F723E-Discovery with STM32F730	STM32756G_EVAL with STM32F750	STM32H743I_EVAL with STM32H750
QSPI Flash mmeory	PB(6, 2), PC(9,10), PE2, PD13	PB(6, 2), PF(8, 9, 8, 6)	PB2, PG(6,9,14), PF(6,7,8,9), PH(2,3), PC11

NOR Flash memory *(on FMC)*	-	PD(0,1,4,5,6,7,8,9,10,11,12,13,14,15)
		PE(2,3,4,5,6,7,8,9,10,11,12,13,14,15)
		PF(0,1,2,3,4,5,12,13,14,15)
		PG(0,1,2,3,4,5)
			PD(0,1,4,5,6,7,8,9,10,11,12,13,14,15)
			PE(2,3,4,5,6,7,8,9,10,11,12,13,14,15)
			PF(0,1,2,3,4,5,12,13,14,15)
			PG(0,1,2,3,4,5)

External SRAM	-	PD(0,1,3,4,5,8,9,10,11,12,13,14,15)
		PE(0,1,3,4,7,8,9,10,11,12,13,14,15)
		PF(0,1,2,3,4,5,12,13,14,15)
		PG(0,1,2,3,4,5,10)
			PD(0,1,3,4,5,8,9,10,11,12,13,14,15)
			PE(0,1,3,4,7,8,9,10,11,12,13,14,15)
			PF(0,1,2,3,4,5,12,13,14,15)
			PG(0,1,2,3,4,5,6,9,10,12,13,14)

External SDRAM	-	PD(0,1,8,9,10,14,15)
		PE(0,1,7,8,9,10,11,12,13,14,15)
		PF(0,1,2,3,4,5,11,12,13,14,15)
		PG(0,1,4,5,8,15)
		PH(2,3,5,8,9,10,11,12,13,14,15)
		PI(0,1,2,3,4,5,6,7,9,10)
			PD(0,1,8,9,10,14,15)
			PE(0,1,7,8,9,10,11,12,13,14,15)
			PF(0,1,2,3,4,5,11,12,13,14,15)
			PG(0,1,2,3,4,5,8,15)
			PH(5,6,7,8,9,10,11,12,13,14,15)
			PI(0,1,2,3,4,5,6,7,9,10)

External PSRAM	PD(0,1,4,5,7,8,9,10,11,12,14,15)
	PE(0,1,7,8,9,10,11,12,13,14,15)
	PF(0,1,2,3,4,5,12,13,14,15)
	PG(0,1,2,3,4,5)
		-	-

The following table summarizes the resources that should be kept unmodified. It describes a list of peripherals (or part of peripheral) that should not be modified in order to avoid the unavailability of external storage. The mentioned peripherals should not be reset or clock disabled, nor reconfigured in a manner that can alter their behavior.

Note: Some elements might change based on the External memory boot application configuration chosen for the selected board. and on the platform's hardware.

Table 4. **Peripherals required by memory type**

	STM32F723E-Discovery with STM32F730	STM32756G_EVAL with STM32F750	STM32H743I_EVAL with STM32H750
QSPI Flash memory	QSPI1 (0x9000000)	QSPI1 (0x9000000)	Dual QSPI Mode QSPI no longer available
NOR Flash (on FMC)	-	FMC-NOR (No FMC-PSRAM / FMC-SRAM)	FMC-NOR (No FMC-PSRAM / FMC-SRAM)
External SRAM	-	FMC-SRAM (No FMC-PSRAM / FMC-NOR)	FMC-SRAM (No FMC-PSRAM / FMC-NOR)
External SDRAM	-	FMC-SDRAM (200 MHz maximum system frequency)	FMC-SDRAM
External PSRAM	FMC-SRAM (No FMC-SRAM / FMC-NOR)	-	-

5 Description of the external memory user application

5.1 Required updates

The external memory application is based on a specific boot schema, which is different from the standard one and which supports a smooth transition from the on-chip application to the off-chip application.

There are two updates that must be done by the user as the location of the application has changed:

- Ensure the usage of the required linker file with memory mapping that corresponds to the selected boot option.
- Update the settings of VTOR to use the right address.

5.2 Load and debug

The three boards STM32F723E-Discovery, STM32756G_EVAL and STM32H743I_EVAL have a loader for external non-volatile memories. Those loaders are provided within the STM32CubeF7/H7 as:

- Patch for EWARM IDE
- Dedicated pack for MDK-ARM IDE

The XiP model provides a seamless load and debug experience similar to an internal Flash debugging. For SW4STM32 IDE, the STM32CubeProgrammer should be used for application loading on external Flash memories.

In BootROM model, the application is compiled and linked for execution from an external volatile memory:

- External SDRAM : linker address 0xD0000000 for STM32H750 Value line and 0xC0000000 for STM32F7x0 Value line
- External SRAM : linker address 0x68000000 for STM32H750 Value line and STM32F7x0 Value line

The application binary shall be then stored either into the SPI_NOR Flash memory or into the SDCARD. It is up to the boot application to copy the user application from the storage area to the execution RAM area.

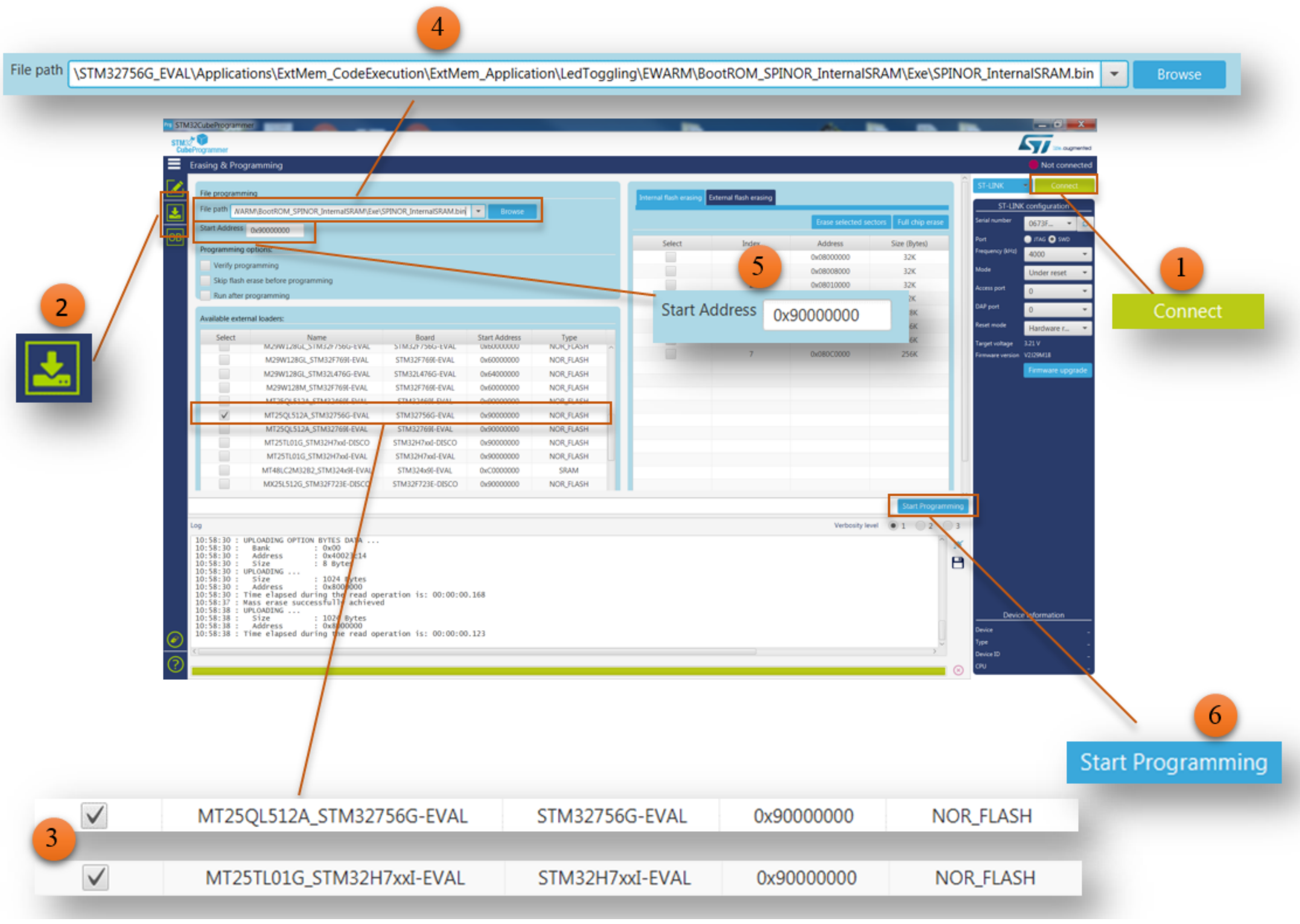
As consequence, the application's load schema cannot be handled by the IDE (MDK-ARM or EWARM) external memory Flash loader (as the application's link address and storage address are different).

Depending on the **BINARY_AREA** define (specified in the “memory.h” file of the boot application), this model requires the use of the two different loading schemas below:

- **SPI_NOR**
  The user application shall be stored into the SPI-NOR Flash memory at the address 0x90000000. It has to be done using the STM32CubeProgrammer. The output of the application shall be in binary format in order to be able to specify a different load address which is the SPI-Flash address. See details in the figure below.
- **SDCARD**
  The user should manually copy the binary file, output of the build, into the SDCARD that is used to store the user application, then plug the SDCARD into the evaluation board.

The figure below shows the steps to be followed to load and debug:

Figure 5. STM32CubeProgrammer


4
File path
\STM32756G_EVAL\Applications\ExtMem_CodeExecution\ExtMem_Application\LedToggling\EWARM\BootROM_SPINOR_InternalSRAM\Exe\SPINOR_InternalSRAM.bin
Browse
2
5
Start Address
0x90000000
1
Connect
6
Start Programming
3

✓	MT25QL512A_STM32756G-EVAL	STM32756G-EVAL	0x90000000	NOR_FLASH
✓	MT25TL01G_STM32H7xxI-EVAL	STM32H7xxI-EVAL	0x90000000	NOR_FLASH

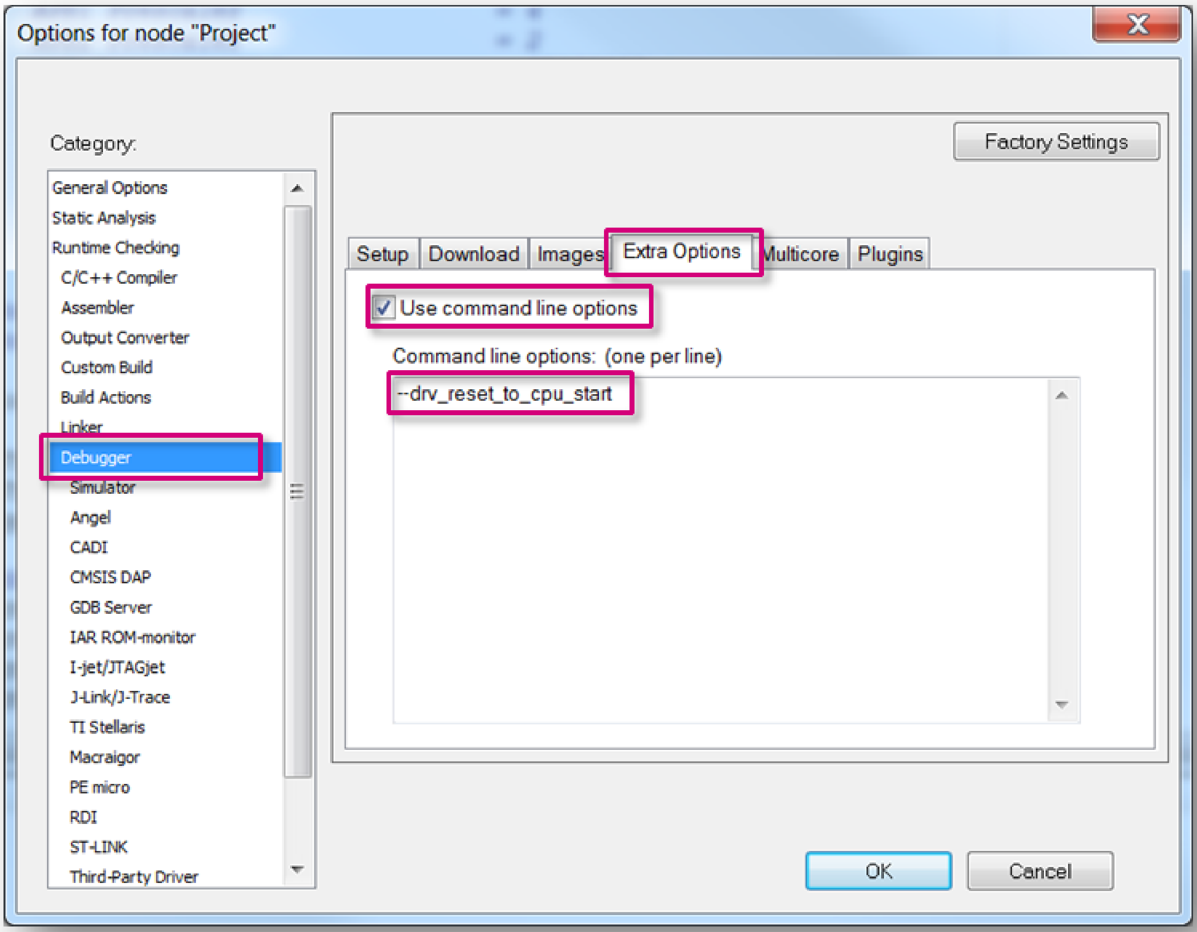
```
10:58:30 : UPLOADING OPTION BYTES DATA ...
10:58:30 :   Bank          : 0x00
10:58:30 :   Address       : 0x40023c14
10:58:30 :   Size          : 8 Bytes
10:58:30 : UPLOADING ...
10:58:30 :   Size          : 1024 Bytes
10:58:30 :   Address       : 0x8000000
10:58:30 : Time elapsed during the read operation is: 00:00:00.168
10:58:37 : Mass erase successfully achieved
10:58:38 : UPLOADING ...
10:58:38 :   Size          : 1024 Bytes
10:58:38 :   Address       : 0x8000000
10:58:38 : Time elapsed during the read operation is: 00:00:00.123
```

5.3 Debug using EWARM IDE

Special precaution is needed with the EWARM IDE when debugging the user application that is running from the external memory. EWARM overrides the default CPU reset value of the PC (program counter) by the one given in the user application (an address value within the external execution memory).

In this boot schema the user application PC address remains inaccessible until the *External memory boot* application is executed (so the external memory is ready and memory mapped via the FMC or QSPI). A hardfault is generated if the EWARM jumps directly to the start point of the user application. To avoid the hardfault, the user shall add the "--drv_reset_to_cpu_start" command line in the debugger options as shown in the figure below. This setting prevents the EWARM from forcing the PC and gives place to the *External memory boot* application to configure the external memory before jumping to the user application.

Figure 6. Debugger command line options



Options for node "Project"

Category: Debugger

Extra Options

[✓] Use command line options

Command line options: (one per line)

```
--drv_reset_to_cpu_start
```

6 Performance characterization

When executing from external memory the performances are impacted due to the external Flash memory latency and the longer instruction/data path. By using the STM32F7x0 Value line and STM32H750 Value line devices, this impact is reduced thanks to the Cortex-M7 L1-cache.

The table below summarizes the EEMBC® CoreMark® scores achieved for each combination of ROM/RAM. The best performances can be achieved when executing from the internal Flash memory. Nevertheless the loss is significantly reduced when execution from an external memory.

These figures illustrate the impact on CPU performance when operating from external memories. The internal Flash configuration score is provided as reference.

Table 5. **EEMBC® CoreMark® score per configuration**

ROM memory	RAM memory	STM32F723E-Discovery with STM32F730 (I/D Cache 8K/8K)	STM32756G_EVAL with STM32F750 (I/D Cache 4K/4K)	STM32H743I_EVAL with STM32H750 (I/D Cache 16K/16K)
QSPI Flash memory	Internal SRAM	1089	948	2020
	External SRAM	-	940	1972
	External SDRAM	-	871	1972
	External PSRAM	1079	-	-
NOR Flash memory	Internal SRAM	-	906	2020
	External SRAM	-	899	1972
	External SDRAM	-	833	1972
External SRAM	Internal SRAM	-	1016	2020
External SDRAM	Internal SRAM	-	989	2020
Internal Flash	Internal SRAM	1092	1082	2020

Revision history

Table 6. **Document revision history**

Date	Version	Changes
11-Jul-2018	1	Initial release.

Contents

List of tables

List of figures