

---

## Execute in Place (XIP) with Quad SPI Interface (QSPI)

---

### APPLICATION NOTE

## Introduction

---

This application note describes the Execute In Place (XIP) feature of QSPI and its implementation in Atmel® | SMART SAM V71/V70/E70/S70 devices. It also explains how to generate binary to execute in QSPI region and how to execute an application from QSPI.

### QSPI Features

- Master SPI Interface
  - Programmable Clock Phase and Clock Polarity
  - Programmable Transfer Delays Between Consecutive Transfers, Between Clock and Data, Between Deactivation and Activation of Chip Select
- SPI Mode
  - Interface to Serial Peripherals such as ADCs, DACs, LCD Controllers, CAN Controllers and Sensors
  - 8-bit/16-bit/32-bit Programmable Data Length
- Serial Memory Mode
  - Interface to Serial Flash Memories Operating in Single-bit SPI, Dual SPI and Quad SPI
  - Supports “Execute In Place” (XIP) — Code Execution by the System Directly from a Serial Flash Memory
  - Flexible Instruction Register for Compatibility with All Serial Flash Memories
  - 32-bit Address Mode (default is 24-bit address) to Support Serial Flash Memories Larger than 128 Mbit
  - Continuous Read Mode
  - Scrambling/Unscrambling "On-The-Fly"
- Connection to DMA Channel Capabilities Optimizes Data Transfers
  - One channel for the Receiver, One Channel for the Transmitter
- Register Write Protection

## Table of Contents

|   |    |
|---|----|
| Introduction.....   | 1  |
| QSPI Features.....  | 1  |
| 1. Hardware/Software Requirements.....  | 4  |
| 1.1. SAM V71 Xplained Ultra Evaluation Kit.....   | 4  |
| 1.2. SAM V71 / V70 / E70 / S70 Software Package.....  | 5  |
| 1.3. Atmel Studio Version 6.2 SP2 or Later.....   | 5  |
| 1.4. Atmel Software Framework (ASF).....  | 5  |
| 1.5. Atmel SAM-BA In-system Programmer (Version 2.16 or Later).....                         | 6  |
| 2. Introduction to QSPI.....  | 7  |
| 2.1. QSPI – SPI mode.....   | 7  |
| 2.2. QSPI – Serial Memory Mode.....   | 8  |
| 2.3. Instruction Frame Structure.....   | 9  |
| 3. eXecute In Place (XIP).....  | 12 |
| 3.1. XIP in SAM S7/SAM E7/SAM V7 MCUs.....  | 12 |
| 4. MPU Configuration for QSPI .....   | 14 |
| 5. Generating Binary for QSPI Memory Region .....   | 16 |
| 5.1. GCC Linker Script Customization.....   | 16 |
| 5.2. IAR Linker File Customization.....   | 17 |
| 5.3. Compiling an Application in QSPI Region.....   | 18 |
| 6. Executing from External Serial Flash Memory Using QSPI XIP Example.....                  | 19 |
| 6.1. Add QSPI Support to an Existing Application in the Software Package.....               | 20 |
| 6.2. Example Source Code in Software Package.....   | 21 |
| 6.3. Add QSPI Support to an Existing Application in ASF.....                                | 23 |
| 6.4. Example Source Code in ASF.....  | 24 |
| 6.5. Application Example Result.....  | 27 |
| 7. Programming an Application to QSPI Using SAM-BA.....                                     | 28 |
| 8. Performance of QSPI in XIP Mode.....   | 30 |
| 8.1. AutoBench Performance in QSPI.....   | 30 |
| 8.2. Code Structure Impact to QSPI Performance.....   | 30 |
| 8.3. Data Access Way Impact to QSPI Performance.....  | 31 |
| 9. Frequently Asked Questions [FAQ].....  | 32 |
| 9.1. Can we boot from QSPI memory?.....   | 32 |
| 9.2. What is the maximum size of the serial Flash memory connected over QSPI?.....          | 32 |
| 9.3. What is the recommended MPU setting for QSPI region?.....                              | 32 |
| 9.4. What is the maximum data rate of the Quad I/O Serial Peripheral Interface (QSPI)?..... | 32 |
| 10. Reference Documents.....  | 33 |

|  |    |
|--|----|
| 10.1. Atmel   SMART SAM S7/SAM E7/SAM V7 Datasheets..... | 33 |
| 10.2. ARM Documentation on Cortex-M7 .....               | 33 |
| 10.3. Reference Application Notes.....                   | 33 |
| 10.4. ASF User Manual .....                              | 33 |
| 11. Revision History.....                                | 34 |

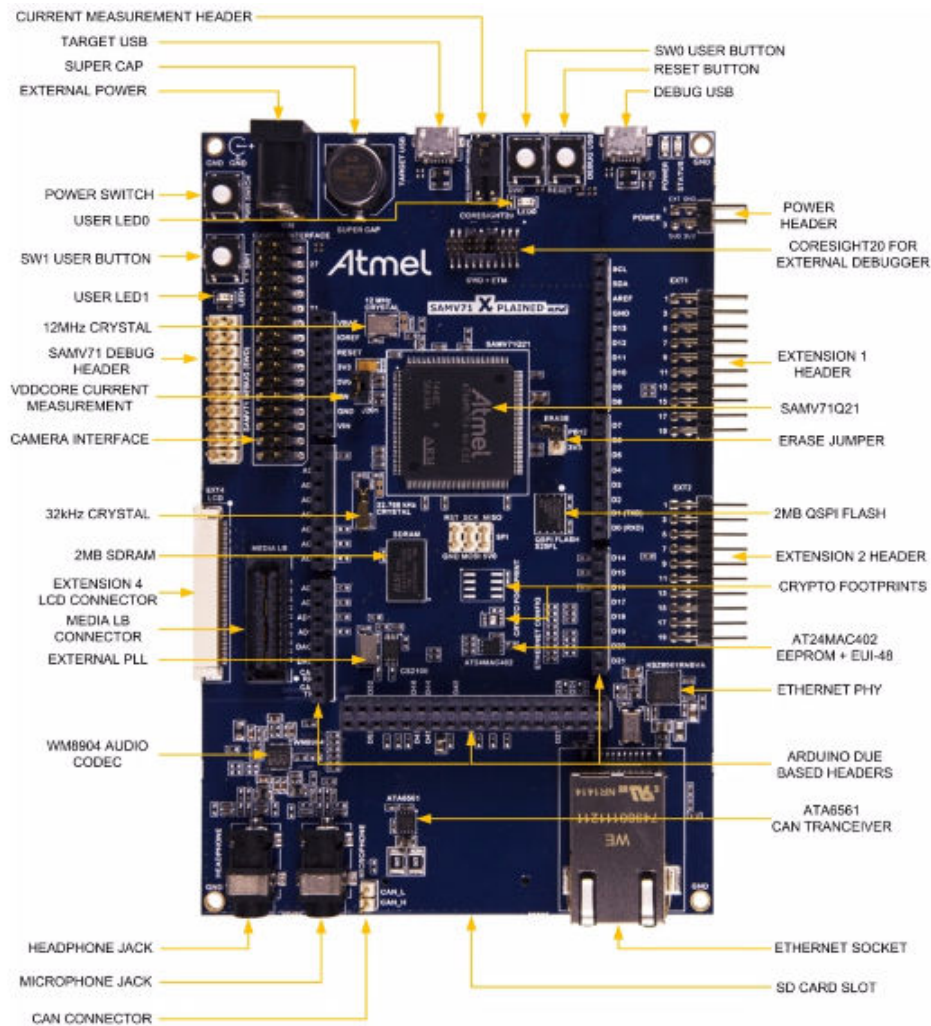
## 1. Hardware/Software Requirements

The following hardware and software environment is needed to evaluate the examples.

### 1.1. SAM V71 Xplained Ultra Evaluation Kit

The Atmel | SMART SAM V71 Xplained Ultra Evaluation Kit is ideal for evaluating and prototyping the Atmel ARM® Cortex®-M7-based microcontrollers in the SAM V71, SAM V70, SAM S70 and SAM E70 series. The SAM V71-XULT evaluation kit does not include extension boards. The extension boards can be purchased individually. Here is a detailed view of the SAM V71 Xplained Ultra Evaluation Kit.

Figure 1-1. SAM V71 Xplained Ultra Evaluation Kit



**Info:** Atmel web page for the SAM V71 Xplained Ultra Evaluation Kit: <http://www.atmel.com/tools/ATSAMV71-XULT.aspx?tab=overview>



**Info:** User Guide: [http://www.atmel.com/Images/Atmel-42408-SAMV71-Xplained-Ultra\\_User-Guide.pdf](http://www.atmel.com/Images/Atmel-42408-SAMV71-Xplained-Ultra_User-Guide.pdf)

---



**Info:** Schematics: [http://www.atmel.com/images/Atmel-42408-SAMV71-Xplained-Ultra\\_User-Guide.zip](http://www.atmel.com/images/Atmel-42408-SAMV71-Xplained-Ultra_User-Guide.zip)

---

## 1.2. SAM V71 / V70 / E70 / S70 Software Package

The software package provides basic drivers, software services, libraries for Atmel | SMART SAM V71, SAM V70, SAM E70, SAM S70 ARM Cortex-M7-based microcontrollers. It contains source code, usage examples, documentation, and ready-to-use projects for Atmel Studio, GNU, IAR EWARM, and Keil MDK.

**Note:** The application is demonstrated by using Software Package Version 1.5. It is always recommended to download the latest version given in the link below.



**Info:** Software Package: <http://www.atmel.com/tools/samv71-samv70-same70-sams70-software-package.aspx>

---

## 1.3. Atmel Studio Version 6.2 SP2 or Later

Atmel Studio is the integrated development platform (IDP) for developing and debugging Atmel | SMART ARM-based and Atmel AVR microcontroller applications.



**Info:** Atmel Studio: <http://www.atmel.com/tools/atmelstudio.aspx?tab=overview>

---

## 1.4. Atmel Software Framework (ASF)

The Atmel Software Framework (ASF) is an MCU software library providing a large collection of embedded software for Atmel Flash MCUs: megaAVR, AVR XMEGA, AVR UC3 and SAM devices. ASF is integrated in the Atmel Studio IDE with a graphical user interface or available as standalone for GCC, IAR compilers. ASF can be downloaded for free.



**Info:** ASF: <http://www.atmel.com/tools/avrsoftwareframework.aspx?tab=overview>

---

## 1.5. Atmel SAM-BA In-system Programmer (Version 2.16 or Later)

Atmel SAM-BA<sup>®</sup> software provides an open set of tools for programming the ARM core-based microcontrollers.



**Info:** SAM-BA: <http://www.atmel.com/tools/atmelsam-bain-systemprogrammer.aspx>

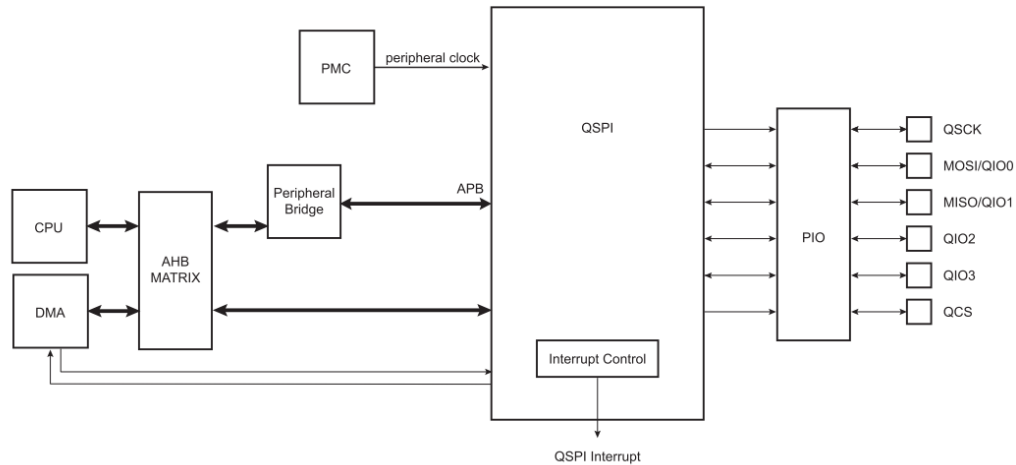
---

## 2. Introduction to QSPI

The Quad SPI Interface (QSPI) is a synchronous serial data link that provides communication with external devices in Master mode. It is similar to SPI protocol except that it has additional data lines.

The normal SPI has four communication lines: Chip Select, Clock, MOSI, and MISO. For QSPI, additional data lines are available. So the command/data/address are sent through single, quad or dual IO based on the mode selected. As data is sent over multiple lines, it helps in increasing bandwidth compared to standard SPI transfer.

**Figure 2-1. QSPI Block Diagram**



In SAM S7/SAM E7/SAM V7 devices, QSPI can be used in normal SPI mode or in Serial Memory mode to connect to external Flash memories. To activate the modes, the SMM bit in the Mode register (QSPI\_MR) needs to be configured accordingly.

QSPI operates on the clock controlled by the internal programmable baud rate generator. Clock phase and polarity can be configured in the Serial Clock register (QSPI\_SCR). The delays listed below are programmable via QSPI\_MR. These delays allow the QSPI to be adapted to the interfaced peripherals based on their speed and timing.

- Transfer Delays between Consecutive Transfers
- Delay between Clock and Data
- Delay between Deactivation and Activation of Chip Select



**Info:** In the following sections, the terms 'serial memory', 'serial Flash memory', 'external Flash memory' refer to the external serial Flash memory connected over QSPI.

### 2.1. QSPI – SPI mode

The QSPI can be used in SPI mode to interface to serial peripherals (such as ADCs, DACs, LCD controllers, CAN controllers and sensors).

In SPI mode, the QSPI acts as a regular SPI Master. To activate this mode, the bit SMM must be cleared in QSPI\_MR.

The QSPI features two holding registers:

- Transmit Data register (QSPI\_TDR)
- Receive Holding register (QSPI\_RDR)

Once enabled, the QSPI\_TDR holds the data to be transferred. After data is written to QSPI\_TDR, it immediately gets shifted to the internal shift register and starts transferring on the MOSI line. When the internal register shifts data on the MOSI line, the MISO line is sampled and stored to QSPI\_RDR. If Receiving mode is not needed, the receive status flag in the Status register (QSPI\_SR) can be discarded.

Interrupts can be generated and DMA can be used for optimized transfers.

Refer to the product datasheet for a detailed description of the operation and configuration of the QSPI in SPI mode.

## 2.2. QSPI – Serial Memory Mode

In Serial Memory mode, the QSPI acts as a serial Flash memory controller. To activate this mode, the SMM bit must be set in the QSPI\_MR. Once enabled, the peripheral appears as memory-mapped device at QSPI memory space 0x8000\_0000. The data is read or written to the address 0x8000\_0000 in Serial Memory mode. In this mode, the data cannot be transferred by the QSPI\_TDR or QSPI\_RDR.

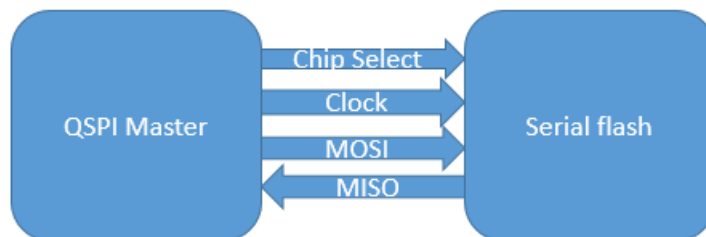
The QSPI can be used to read data from the serial Flash memory allowing the CPU to execute code from it (XIP execute in place). The QSPI can also be used to control the serial Flash memory (Program, Erase, Lock, etc.) by sending specific commands. In Serial Memory mode, the QSPI is compatible with the following modes:

- Single-Bit SPI
- Dual SPI
- Quad SPI

### 2.2.1. Single-Bit SPI

In Single-Bit SPI mode, the communication with external Flash is done via either MOSI or MISO lines only.

**Figure 2-2. Single-Bit SPI**

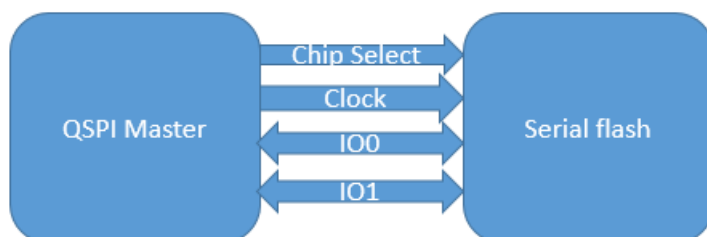


### 2.2.2. Dual SPI

In Dual SPI mode, QSPI master communicates with the external memory through two bidirectional IO lines.



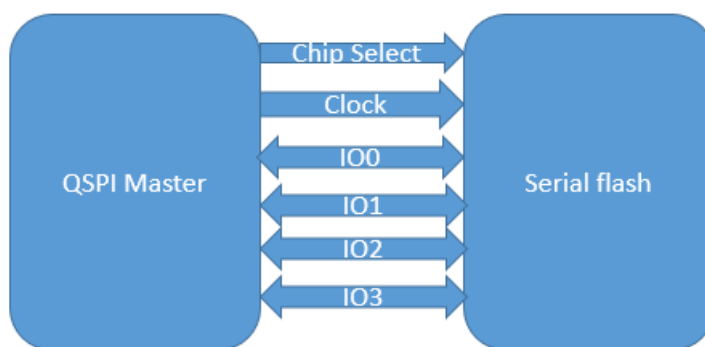
**Figure 2-3. Dual SPI**



### 2.2.3. Quad SPI

In Quad SPI mode, all four IO lines are used for communication with the external memory.

**Figure 2-4. Quad SPI**



In SAM S7/SAM E7/SAM V7 devices, the command, address and data can be sent independently using different modes. The configuration is done in QSPI Instruction Frame register (QSPI\_IFR) based on the serial memory connected over QSPI. The available configurations are listed below.

**Figure 2-5. Available SPI Modes in Serial Memory Mode**

- **WIDTH:** Width of Instruction Code, Address, Option Code and Data

| Value | Name           | Description   |
|-------|----------------|---|
| 0     | SINGLE_BIT_SPI | Instruction: Single-bit SPI / Address-Option: Single-bit SPI / Data: Single-bit SPI |
| 1     | DUAL_OUTPUT    | Instruction: Single-bit SPI / Address-Option: Single-bit SPI / Data: Dual SPI       |
| 2     | QUAD_OUTPUT    | Instruction: Single-bit SPI / Address-Option: Single-bit SPI / Data: Quad SPI       |
| 3     | DUAL_IO        | Instruction: Single-bit SPI / Address-Option: Dual SPI / Data: Dual SPI             |
| 4     | QUAD_IO        | Instruction: Single-bit SPI / Address-Option: Quad SPI / Data: Quad SPI             |
| 5     | DUAL_CMD       | Instruction: Dual SPI / Address-Option: Dual SPI / Data: Dual SPI                   |
| 6     | QUAD_CMD       | Instruction: Quad SPI / Address-Option: Quad SPI / Data: Quad SPI                   |

## 2.3. Instruction Frame Structure

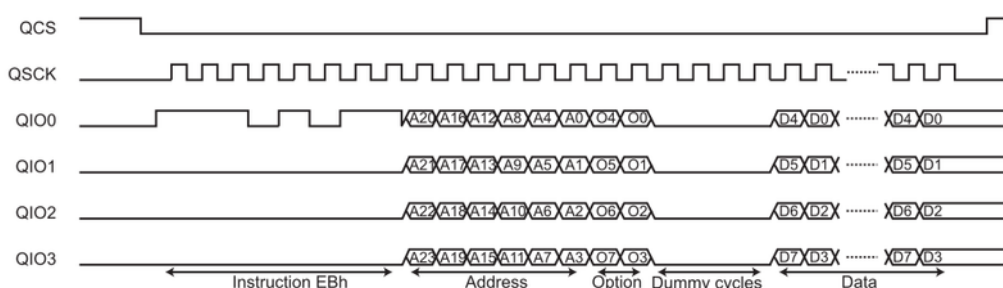
To control the external Flash memories, the QSPI is able to send instructions via the SPI bus for operations such as READ, WRITE, PROGRAM, ERASE, LOCK, etc. The instruction set supported by the serial Flash memories is vendor-dependent.

QSPI includes an Instruction Frame register (QSPI\_IFR) for this purpose which makes it flexible and compatible with all serial Flash memories. An Instruction Frame includes the fields below.

**Figure 2-6. Instruction Frame Structure**

| Instruction Frame Field (Size in bits) | Description  |
|--|--|
| Instruction code (8)                   | The instructions as listed by the serial flash memory. It is optional in some cases.   |
| Address (24 or 32)                     | Address is required for instructions such as READ, PROGRAM, ERASE, LOCK. By default the address is 24 bits long, but it can be 32 bits long to support serial Flash memories larger than 128 Mbit (16 Mbyte) |
| Option Code (1 or 2 or 4 or 8 bits)    | It is useful to activate XIP mode or continuous read mode for READ instruction in some serial flash memory. It improves data read latency.   |
| Dummy Cycles                           | It is required by some READ instructions.  |
| Data bytes                             | Data bytes are present for READ or PROGRAM instructions.   |

A typical instruction frame in Quad SPI mode is illustrated below.



### 2.3.1. Instruction Frame Configuration

The Instruction frame needs to be configured based on the commands to be sent to the external Flash memory. Refer to the datasheet of the respective external Flash memory for the list of supported commands. The registers to be configured are:

- Instruction Frame register (QSPI\_IFR)
- Instruction Address register (QSPI\_IAR)
- Instruction Code register (QSPI\_ICR)

### 2.3.2. Instruction Frame Register

The QSPI\_IFR must be written based on the command to be sent. If the instruction frame does not include any data, writing to this register triggers the instruction transmission over QSPI.

If the instruction frame includes data, the instruction frame will be transferred by the first data access in the QSPI memory space. The QSPI\_IFR includes the following configurable fields.

**Figure 2-7. Instruction Frame Register**

| Field [bits]   | Description   |
|----------------|---|
| WIDTH [2:0]    | Type of QSPI mode to be used for sending instruction, address, option code and data as explained in Figure 2-5. |
| INSTEN [4]     | Enable to send Instruction code.  |
| ADDREN [5]     | Enable to send address after instruction code.  |
| OPTEN [6]      | Enable to send option code after address.   |
| DATAEN [7]     | Enable to receive / send data during READ or PROGRAM instruction.   |
| OPTL [9:8]     | Length of the option code. The length of the option code must be consistent with WIDTH configuration.           |
| ADDRL [10]     | Address length (24 or 32 bits).   |
| TFRTYP [13:12] | Type of data transfer to be performed.  |
| CRM [14]       | Enable continuous read mode.  |
| NBDUM [20:16]  | Number of dummy cycles to be added when reading from serial flash memory.                                       |

### Transfer Types (TFRTYP)

- TFRTYP = 0: To read serial memory such as JEDEC-ID or serial memory status register, but not to read data stored in memory.
- TFRTYP = 1: To read serial memory data. The address of the first instruction frame is the first read access over QSPI memory space (0x80000000). For non-sequential read access, a new instruction frame is sent with the last system read access.
- TFRTYP = 2: To write serial memory such as configuration register or status register, but not to write memory data over QSPI space.
- TFRTYP = 3: To write to serial memory space.

For TFRTYP = 0/2/3: The address sent in the instruction frame is the address of the first system bus accesses. The addresses of the next accesses are not used by the QSPI.

### 2.3.3. Instruction Address Register

If the instruction frame includes only address and no data, the address to send must be written to the Instruction Address register (QSPI\_IAR). For example, BLOCK ERASE command would need only address and does not need any data. When data is present, the address of the instruction is defined by the address of the data accesses in the QSPI memory space, not by QSPI\_IAR.

### 2.3.4. Instruction Code Register

If the instruction frame includes the instruction code and/or the option code, it is written to INST and OPT fields in the Instruction Code register (QSPI\_ICR).

### 2.3.5. End of Instruction Frame

When data transfer is not enabled, the end of the instruction frame is indicated when the INSTRE flag in QSPI\_SR rises. When data transfer is enabled, the user must indicate when data transfer is completed in the QSPI memory space by setting the bit LASTXFR in QSPI\_CR. The end of the instruction frame is indicated when the INSTRE flag in QSPI\_SR rises.

### 3. eXecute In Place (XIP)

Execute in place is the method of executing code directly from serial Flash memory without copying the code to RAM (code shadowing). The serial Flash memory mapping is seen as another memory in the product memory map.

As Quad SPI mode uses four lines for data transfer, it allows the system to use high-performance serial Flash memories which are small and inexpensive, in place of larger and more expensive parallel Flash memories.

#### 3.1. XIP in SAM S7/SAM E7/SAM V7 MCUs

The data of the serial Flash memory can be accessed by sending an instruction with DATAEN = 1 and TFRYP = 1 in QSPI\_IFR. In XIP mode, QSPI is able to read data at random address from the serial Flash memory, allowing the CPU to execute code directly from it (XIP execute-in-place).

##### 3.1.1. Continuous Read Mode

The QSPI is compatible with the Continuous Read mode which is implemented in some serial Flash memories.

Continuous Read mode is used when reading data from the memory (TFRYP = 1). The addresses of the system bus read accesses are often non-sequential and this leads to many instruction frames that have the same instruction code.

When the Continuous Read mode is activated in a serial Flash memory by a specific option code, the instruction code is stored in the memory. For the next instruction frames, the instruction code is not required as the memory uses the stored one. By disabling the send of the instruction code, the Continuous Read mode reduces the access time of the data.

To be functional, Continuous Read Mode must be enabled in both the QSPI and the serial Flash memory. It is enabled in the QSPI by setting the bit CRM in the QSPI\_IFR (TFRYP field value must equal 1). It is enabled in the serial Flash memory by sending a specific option code.



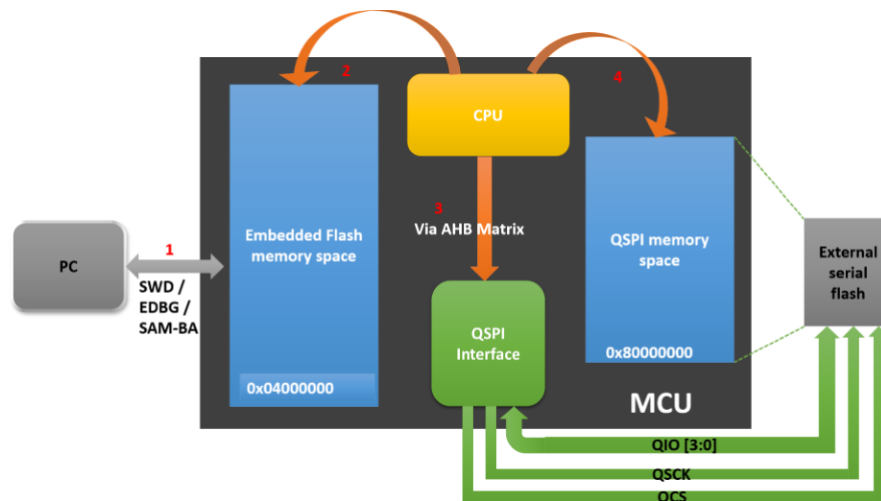
**Caution:** If the Continuous Read mode is not supported by the serial Flash memory or disabled, the CRM bit must not be set. Otherwise data read out from the serial Flash memory is unpredictable.

To enable XIP and execute from serial Flash memory, the application should meet the following criteria:

- MPU for QSPI region should be configured for program and execute mode accordingly.
- Binary should be generated for the application to be stored in the external serial Flash memory.
- A bootloader program is needed in the embedded Flash memory which programs the external serial Flash and directs the CPU to jump to the serial Flash for execution of application.

The figure shown below illustrates the overview of the process flow for executing application from external serial Flash memory. The sections explain each step in detail.

Figure 3-1. Execution of Application from External Flash Memory



1. Load the bootloader code to the embedded Flash (@ 0x04000000) via SWD/EDBG/SAM-BA interface to program the serial Flash memory.
2. The CPU starts executing from the embedded Flash and initializes the QSPI in Serial Memory mode.
3. Program the external serial Flash with binary generated for QSPI memory region via QSPI. Using Quad SPI mode improves the performance.
4. In Serial Memory mode, the serial Flash mapped to 0x80000000 appears as other embedded memory to CPU. The CPU starts executing from QSPI memory space.

## 4. MPU Configuration for QSPI

The Cortex-M7 processor features a Memory Protection Unit (MPU) allowing to divide the memory map into a number of regions with privilege permissions and access rules. It helps in providing fine-grain memory control, enabling applications to utilize multiple privilege levels, separating and protecting code, data and stack on a task-by-task basis.

SAM S7/SAM E7/SAM V7 devices manage up to 16 regions with MPU for safety/critical applications. The following figure summarizes the available MPU attributes in Cortex-M7. Refer to the ARM Cortex-M7 Technical Reference Manual available on [www.arm.com](http://www.arm.com).

Figure 4-1. MPU Attributes

| Memory Type      | Shareability | Attributes   | Description   |
|------------------|--------------|--|---|
| Strongly-ordered | N/A          | N/A  | All access occur in program order. No concurrent access can be done until the current access is completed |
| Device           | Shared       | N/A  | All access occur in program order. Memory-mapped peripherals shared by several masters                    |
|                  | Non-Shared   | N/A  | All access occur in program order. Memory-mapped peripherals used by one single master                    |
| Normal           | Shared       | Non-cacheable<br>Write-through cacheable<br>Write-back cacheable | Normal Memory shared by several masters   |
|                  | Non-Shared   | Non-cacheable<br>Write-through cacheable<br>Write-back cacheable | Normal Memory shared by one single master   |

When accessed by the Cortex-M7 processor for programming operations, the QSPI memory space must be defined in the Cortex-M7 memory protection unit (MPU).

For Programming operations,

- QSPI memory space must be defined in the MPU with the attribute 'Device' or 'Strongly Ordered'.

For Fetch/Read operations,

- QSPI memory space must be defined in the MPU with the attribute 'Normal' in order to benefit from internal cache.

In the software package and ASF example, it is configured as 'Strongly Ordered' by default as below.

In ASF:

`ASF\sam\boards\samv71_xplained_ultra\init.c`

```
/**
 * QSPI memory region --- Strongly ordered
 * START_Addr:- 0x80000000UL
 * END_Addr:- 0x9FFFFFFFUL
 */
dw_region_base_addr =
    QSPI_START_ADDRESS |
    MPU_REGION_VALID |
    MPU_QSPIMEM_REGION;

dw_region_attr =
    MPU_AP_FULL_ACCESS |
    STRONGLY_ORDERED_SHAREABLE_TYPE |
    mpu_cal_mpu_region_size(QSPI_END_ADDRESS - QSPI_START_ADDRESS) |
    MPU_REGION_ENABLE;
```

```
mpu_set_region( dw_region_base_addr, dw_region_attr);
```

In software package:

*\\Atmel\\samv71\_Xplained\_Ultra\\libraries\\libboard\\source\\board\_lowlevel.c*

```
/* *****  
QSPI memory region --- Strongly ordered  
START_Addr:- 0x80000000UL  
END_Addr:- 0x9FFFFFFFUL  
***** */  
dwRegionBaseAddr =  
    QSPI_START_ADDRESS |  
    MPU_REGION_VALID |  
    MPU_QSPIMEM_REGION;          //8  
  
dwRegionAttr =  
    MPU_AP_FULL_ACCESS |  
    STRONGLY_ORDERED_SHAREABLE_TYPE |  
    MPU_CalMPURegionSize(QSPI_END_ADDRESS - QSPI_START_ADDRESS) |  
    MPU_REGION_ENABLE;  
  
MPU_SetRegion( dwRegionBaseAddr, dwRegionAttr);
```



**Info:** Before jumping to QSPI XIP mode, the QSPI memory space should be reconfigured to 'Normal' type to benefit from cache. Example code snippet from software package is shown below.

```
/* Update QSPI Region to Full Access and cacheable*/  
MPU_UpdateRegions(MPU_QSPIMEM_REGION, QSPI_START_ADDRESS, \  
    MPU_AP_FULL_ACCESS |  
    INNER_NORMAL_WB_NWA_TYPE( NON_SHAREABLE ) |  
    MPU_CalMPURegionSize(QSPI_END_ADDRESS - QSPI_START_ADDRESS) |  
    MPU_REGION_ENABLE);
```

## 5. Generating Binary for QSPI Memory Region

To be able to execute from QSPI memory space, the binary file of the application needs to be linked to the QSPI address space. This is controlled by using the linker file.

Once the compiler generates the object files, they need to be correctly linked as per the memory map of the corresponding target device. All the object files use relative addressing, and the final address mapping is performed at link time. A linker combines input files (object file format) into a single output file (executable).

The linker files are different for each compiler. The linkers make use of a linker script/command file to place different code and data sections into appropriate memory.

The linker file can be located in the following locations:

In ASF (GCC/IAR):

`\sam\utils\linker_scripts\`

In software package (MDK), qspi.sct at:

`\Atmel\SAMV71_Xplained_Ultra\libraries\libboard\resources_v71\mdk\`

In software package (Atmel Studio), samv71q21\_qspi.ld at:

`\studio\examples\Atmel\SAMV71_Xplained_Ultra\libraries\libboard\resources_v71\gcc\`

In software package (GCC), samv71q21\_qspi.ld at:

`\Atmel\samv71_Xplained_Ultra\libraries\libboard\resources_v71\gcc\`

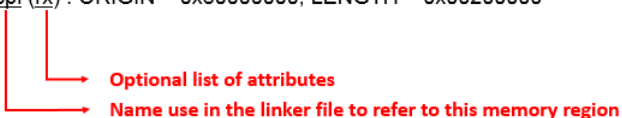
In software package (IAR), samv71q21\_qspi.icf at:

`\arm\examples\Atmel\samv71_Xplained_Ultra\libraries\libboard\resources_v71\iar\`

### 5.1. GCC Linker Script Customization

The default linker file has memory segment for Flash and SRAM region. In addition, to be able to link application to the QSPI memory space, a memory segment with respective address and length needs to be created as below.

```
/* Memory Spaces Definitions */
MEMORY
{
  qspi (rx) : ORIGIN = 0x80000000, LENGTH = 0x00200000
}
```



- r: read-only section
- x: executable section

Once a memory region is defined, the linker script can direct the linker to place the specific output sections into that memory region.



Now the text section, constant data, vector table, etc. need to be linked to the QSPI region. In other words, the Flash region in the linker needs to be replaced with the QSPI region as below.

```
SECTIONS
{
    .text :
    {
        . = ALIGN(4);
        _sfixed = .;
        KEEP(*( .vectors .vectors.*))
        *(.text .text.* .gnu.linkonce.t.*)
        *(.glue_7t) *(.glue_7)
        *(.rodata .rodata* .gnu.linkonce.r.*)
        *(.ARM.exidx* .gnu.linkonce.armexidx.*)

        /* Support C constructors, and C destructors in both user code
           and the C library. This also provides support for C++ code. */
        . = ALIGN(4);
        KEEP(*( .init))
        . = ALIGN(4);
        __preinit_array_start = .;
        KEEP (*( .preinit_array))
        __preinit_array_end = .;

        . = ALIGN(4);
        __init_array_start = .;
        KEEP (*(SORT(.init_array.*)))
        KEEP (*( .init_array))
        __init_array_end = .;

        . = ALIGN(0x4);
        KEEP (*crtbegin.o(.ctors))
        KEEP (*(EXCLUDE_FILE (*crtend.o) .ctors))
        KEEP (*(SORT(.ctors.*)))
        KEEP (*crtend.o(.ctors))

        . = ALIGN(4);
        KEEP(*( .fini))

        . = ALIGN(4);
        __fini_array_start = .;
        KEEP (*( .fini_array))
        KEEP (*(SORT(.fini_array.*)))
        __fini_array_end = .;

        KEEP (*crtbegin.o(.dtors))
        KEEP (*(EXCLUDE_FILE (*crtend.o) .dtors))
        KEEP (*(SORT(.dtors.*)))
        KEEP (*crtend.o(.dtors))

        . = ALIGN(4);
        _efixed = .;          /* End of text section */
    } > qspi

    /* .ARM.exidx is sorted, so has to go in its own output section. */
    PROVIDE_HIDDEN (__exidx_start = .);
    .ARM.exidx :
    {
        *(.ARM.exidx* .gnu.linkonce.armexidx.*)
    } > qspi
}
```

**Note:** The data section is untouched in this example and it is located in internal SRAM by default.

## 5.2. IAR Linker File Customization

The changes below need to be done in IAR linker file (\*.icf), to generate binary for QSPI region.

- Move the vector table to the start of the QSPI memory region 0x80000000.
- Define symbol and memory region for QSPI. Move the read-only, interrupt vector to the QSPI region.

/\*-Specials-\*/

```
define symbol __ICFEDIT_intvec_start__ = 0x80000000;
```

```
define region QSPI_region = mem:[from __ICFEDIT_region_QSPI_start__ to  
__ICFEDIT_region_QSPI_end__];
```

/\*-Memory Regions-\*/

```
define symbol __ICFEDIT_region_QSPI_start__ = 0x80000000;
```

```
define symbol __ICFEDIT_region_QSPI_end__ = 0x80200000 -1;
```

```
place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };
```

```
place in QSPI_region { readonly };
```

### 5.3. Compiling an Application in QSPI Region

The QSPI memory region can be configured with 'Normal' memory type while generating binary to execute from QSPI. This is because the bootloader application should have it configured as 'Strongly Ordered' type for programming operation of the QSPI memory.

The linker script should be customized as explained in the above sections to link the application to the QSPI region. Now the generated binary file will have the text section placed in the QSPI region and it is ready for programming to the external serial Flash memory.



**Info:** The "getting-started" example in the software package can generate a binary to execute from QSPI, with the configuration of 'qspi' instead of 'sram' or 'flash'.

## 6. Executing from External Serial Flash Memory Using QSPI XIP Example

This section explains the QSPI XIP application that acts as a bootloader to program and execute from serial Flash memory. The example for QSPI XIP mode can be found in ASF and in the software package via the following paths, respectively. The ASF example is also available through Atmel Studio.

In Atmel Studio:

*File -> New -> Example Project -> QSPI Example Memory mode for S25FL1XX – SAMV71-XULTRA*

In standalone ASF:

`\common\components\memory\qspi_flash\s25fl1xx\example\`

In software package:

`\Atmel\samv71_Xplained_Ultra\examples\qspi_xip\`

After generating binary as explained in the section "[Generating Binary for QSPI Memory Region](#)" of this application note, this example extracts the hex code from binary and stores it in the header file as an array of hex values.

The sequence of the QSPI XIP application is as follows:

1. Configure the MPU for all memory regions. The QSPI should be configured as 'Strongly ordered' for programming operation.
2. Initialize system clock and peripheral clock.
3. Enable I/D-Cache.
4. Configure GPIO pins (QSCK, QCS, QIO [3:0]) for QSPI peripheral.
5. Initialize serial Flash memory in Serial Flash Memory mode with clock and polarity settings.
6. Enable Quad SPI mode for better performance.
7. Erase serial memory by executing the appropriate ERASE command.
8. Send the Page program command with the input buffer containing the hex values extracted from binary file.
9. Read the first few bytes to extract the Stack Pointer and reset the handler value of the QSPI binary application.
10. Enable 'Continuous Read Mode' to enter into XIP mode.
11. After reading data, verify the content with the input buffer.
12. If verification passes, configure the Main Stack Pointer and jump to the application in the QSPI based on the value read from the QSPI region.

The serial Flash memory on SAM V71-Xplained Ultra board is S25FL116K. The serial Flash driver implementing the supported commands is located in the directories below.

In ASF:

`\ASF\common\components\memory\qspi_flash\s25fl1xx\`

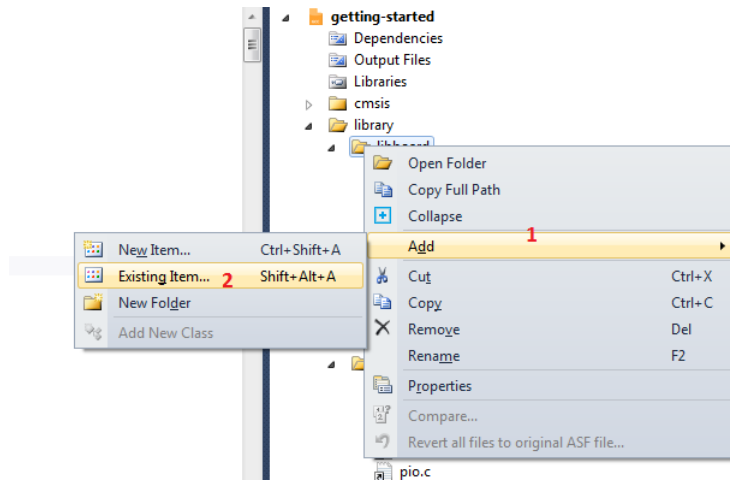
In software package:

`..Atmel\samv71_Xplained_Ultra\libraries\libboard\source\s25fl1.c`

`..Atmel\samv71_Xplained_Ultra\libraries\libboard\include\s25fl1.h`

## 6.1. Add QSPI Support to an Existing Application in the Software Package

In the software package, the driver support files come with the package and need to be added as a link as below to enable support in the application.



To add QSPI XIP support, the following files need to be added as a link to the application as below.

- Serial Flash memory driver file from board

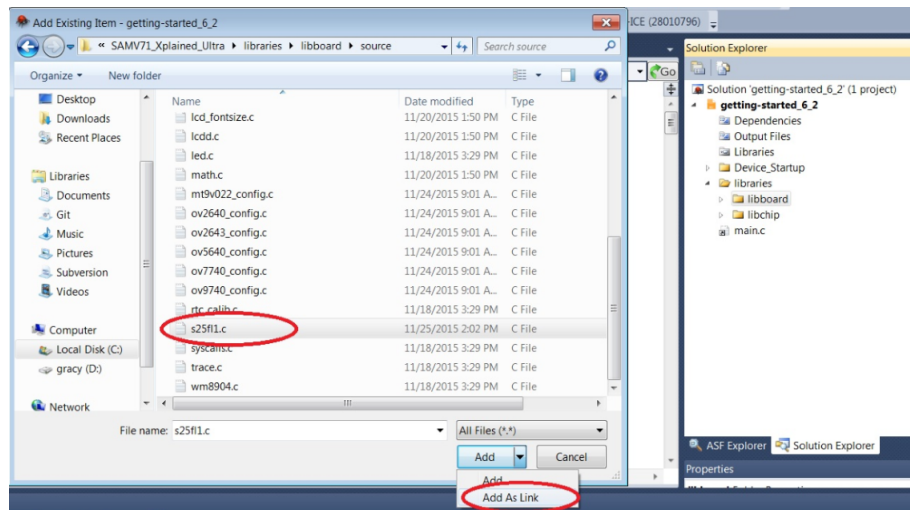
..*Atmel*\samv71\_Xplained\_Ultra\libraries\libboard\source\s25fl1.c

- QSPI driver files

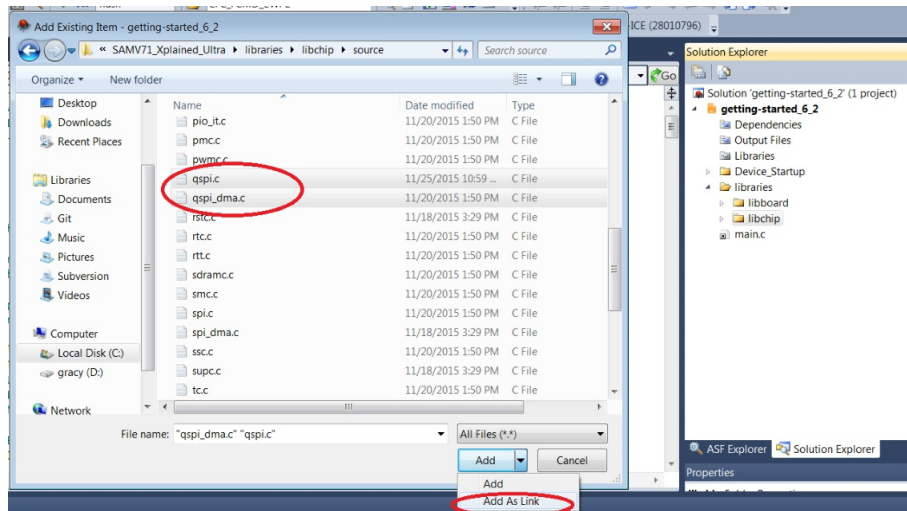
..*Atmel*\samv71\_Xplained\_Ultra\libraries\libchip\source\qspi.c

..*Atmel*\samv71\_Xplained\_Ultra\libraries\libchip\source\qspi\_dma.c

1. Add S25fl1xx driver file to the ..libboard folder.



2. Add QSPI driver files to the ..libchip folder.



## 6.2. Example Source Code in Software Package

```

/* -----
/*                               Atmel Microcontroller Software Support
/*                               SAM Software Package License
/* -----
/* Copyright (c) 2015, Atmel Corporation
/*
/* All rights reserved.
/*
/* Redistribution and use in source and binary forms, with or without
/* modification, are permitted provided that the following condition is met:
/*
/* Redistributions of source code must retain the above copyright notice,
/* this list of conditions and the disclaimer below.
/*
/* Atmel's name may not be used to endorse or promote products derived from
/* this software without specific prior written permission.
/*
/*
/* DISCLAIMER: THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR
/* IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
/* MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
/* DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT,
/* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
/* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
/* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
/* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
/* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
/* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
/* -----
*/

/**
 * \file
 *
 * This file contains all the specific code for the Qspi_serialflash example.
 */

/*-----
 * Headers
 *-----*/

#include <board.h>
#include <stdio.h>
#include <assert.h>
#include <string.h>
#include "stdlib.h"

```

```

#include "getting_started_hex.h"

/*-----
 *          Local definitions
 *-----*/

/** SPI peripheral pins to configure to access the serial flash. */
#define QSPI_PINS      PINS_QSPI

/*-----
 *          Local variables
 *-----*/
/** Pins to configure for the application. */
static Pin pins[] = QSPI_PINS;

COMPILER_ALIGNED(32) static uint32_t Buffer[4];

/*-----
 *          Global functions
 *-----*/

/**
 * \brief Application entry point for QSPI_XIP example.
 * Initializes the serial flash and performs XIP.
 *
 * \return Unused (ANSI-C compatibility).
 */

int main(void)
{
    uint8_t MemVerify = 0;
    uint32_t __Start_SP, idx;
    uint32_t (*__Start_New)(void);
    uint8_t *pMemory = (uint8_t *) ( QSPIMEM_ADDR );

    /* Disable watchdog */
    WDT_Disable(WDT);

    SCB_EnableICache();
    SCB_EnableDCache();

    /* Output example information */
    printf("-- QSPI XIP Example %s --\n\r", SOFTPACK_VERSION);
    printf("-- %s\n\r", BOARD_NAME);
    printf("-- Compiled: %s %s With %s--\n\r", __DATE__, __TIME__, COMPILER_NAME);

    /* Configure systick */
    TimeTick_Configure();

    /* Initialize the QSPI and serial flash */
    PIO_Configure(pins, PIO_LISTSIZE(pins));

    /* Enable the clock of QSPI */
    ENABLE_PERIPHERAL(ID_QSPI);

    S25FL1D_InitFlashInterface(1);
    printf("QSPI drivers initialized\n\r");

    /* enable quad mode */
    S25FL1D_QuadMode(ENABLE);

    /* get the code at the beginning of QSPI, run the code directly if it's valid */
    S25FL1D_ReadQuadIO(Buffer, sizeof(Buffer), 0, 1, 0);
    printf("-I- data at the beginning of QSPI: %08x %08x %08x %08x\n\r",
        (unsigned int)Buffer[0], (unsigned int)Buffer[1],
        (unsigned int)Buffer[2], (unsigned int)Buffer[3]);
    if ((IRAM_ADDR <= Buffer[0]) && (IRAM_ADDR + IRAM_SIZE > Buffer[0]) &&
        (QSPIMEM_ADDR < Buffer[1]) && (1 == (Buffer[1]&0x3))) {
        __Start_New = (uint32_t(*) (void)) Buffer[1];
    }
}

```

```

    __Start_SP = Buffer[0];

    printf("-I- a valid application is already in QSPI, run it from QSPI\n\r");
    printf("===== \n\r");

    __set_MSP(__Start_SP);
    __Start_New();
} else {
    printf("-I- there isn't a valid application in QSPI, program first\n\r");
}

if (S25FL1D_Unprotect()) {
    printf("Unprotect QSPI Flash failed!\n\r");
    while (1);
}

/* erase entire chip */
S25FL1D_EraseChip();

/* Flash the code to QSPI flash */
printf("Writing to Memory\n\r");

S25FL1D_Write((uint32_t *)pBuffercode, sizeof(pBuffercode), 0, 0);

printf("Example code written 0x%x to Memory\n\r", sizeof(pBuffercode));

printf("Verifying \n\r");

/* Update QSPI Region to Full Access and cacheable*/
MPU_UpdateRegions(MPU_QSPIMEM_REGION, QSPI_START_ADDRESS, \
    MPU_AP_FULL_ACCESS |
    INNER_NORMAL_WB_NWA_TYPE( NON_SHAREABLE ) |
    MPU_CalMPURegionSize(QSPI_END_ADDRESS - QSPI_START_ADDRESS) |
    MPU_REGION_ENABLE);

/* Start continuous read mode to enter in XIP mode*/
S25FL1D_ReadQuadIO(Buffer, sizeof(Buffer), 0, 1, 0);

for (idx = 0; idx < sizeof(pBuffercode); idx++) {
    if (*pMemory == pBuffercode[idx]) {
        pMemory++;
    } else {
        MemVerify = 1;
        printf("Data does not match at 0x%x \n\r", (unsigned)pMemory);
        break;
    }
}

if (!MemVerify) {
    printf("Everything is OK \n\r");
    /* set PC and SP */
    __Start_New = (uint32_t(*) (void) ) Buffer[1];
    __Start_SP = Buffer[0];

    printf("\n\r Starting getting started example from QSPI flash \n\r");
    printf("===== \n\r");

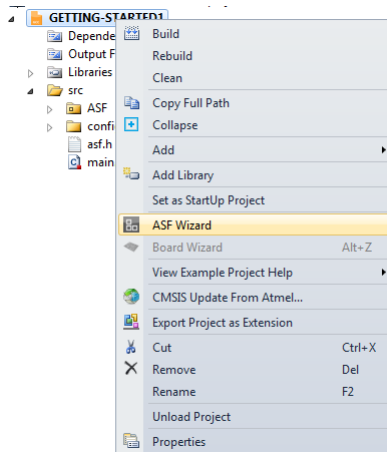
    __set_MSP(__Start_SP);

    __Start_New();
}
while (1);
}

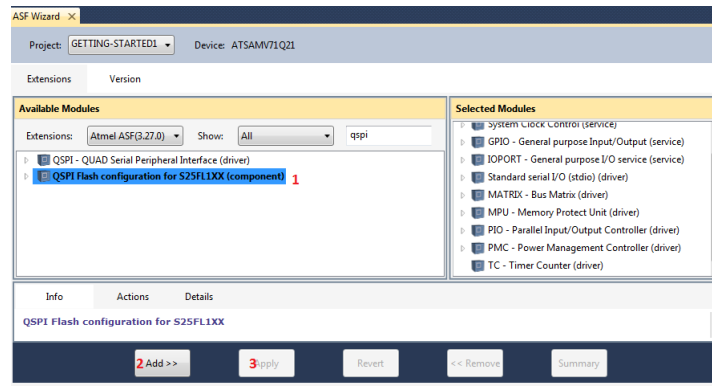
```

### 6.3. Add QSPI Support to an Existing Application in ASF

1. The QSPI serial Flash memory component can be added to the existing application via ASF wizard.



2. Add the QSPI serial Flash memory component. It adds the dependent qspi driver file as well.



## 6.4. Example Source Code in ASF

```

/**
 * \file
 *
 * \brief SDRAMC on QSPI example for SAMV71.
 *
 * Copyright (c) 2015 Atmel Corporation. All rights reserved.
 *
 * \asf_license_start
 *
 * \page License
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. The name of Atmel may not be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * 4. This software may only be redistributed and used in connection with an
 *    Atmel microcontroller product.
 *
 * THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF

```



```

* MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
* EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR
* ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
* ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
* POSSIBILITY OF SUCH DAMAGE.
*
* \asf_license_stop
*
*/

#include "asf.h"
#include "conf_board.h"
#include "getting_started_hex.h"
#include "s25fl1xx.h"

#define STRING_EOL "\r"
#define STRING_HEADER "--QSpi Example --\r\n" \
    "-- "BOARD_NAME" --\r\n" \
    "-- Compiled: " __DATE__ " " __TIME__ " --"STRING_EOL

struct qspid_t g_qspid = {QSPI, 0, 0, 0};

struct qspi_config_t mode_config = {QSPI_MR_SMM_MEMORY, false, false,
QSPI_LASTXFER, 0, 0, 0, 0, 0, 0, 0, false, false, 0};

#define WRITE_SIZE sizeof(pBuffercode)

/**
 * \brief Configure the console uart.
 */
static void configure_console(void)
{
    const usart_serial_options_t uart_serial_options = {
        .baudrate = CONF_UART_BAUDRATE,
#ifdef CONF_UART_CHAR_LENGTH
        .charlength = CONF_UART_CHAR_LENGTH,
#endif
        .paritytype = CONF_UART_PARITY,
#ifdef CONF_UART_STOP_BITS
        .stopbits = CONF_UART_STOP_BITS,
#endif
    };

    /* Configure console UART. */
    stdio_serial_init(CONF_UART, &uart_serial_options);
}

/**
 * \brief Application entry point for sdramc_example.
 *
 * \return Unused (ANSI-C compatibility).
 */
int main(void)
{
    uint8_t mem_verified = 0;
    uint32_t __start_sp, idx;
    uint32_t (*__start_new)(void);
    uint32_t buffer[4];

    uint8_t *memory = (uint8_t *)QSPIMEM_ADDR;
    enum status_code status = STATUS_OK;

    /* Initialize the system */
    sysclk_init();
    board_init();

```

```

/* Configure the console uart */
configure_console();

/* Output example information */
puts(STRING_HEADER);

/* Enable SMC peripheral clock */
pmc_enable_periph_clk(ID_QSPI);

/* QSPI memory mode configure */
status = s25fllxx_initialize(g_qspid.qspi_hw, &mode_config, 1);
if (status == STATUS_OK) {
    puts("QSPI drivers initialized\n\r");
} else {
    puts("QSPI drivers initialize failed\n\r");
    while (1) {
        /* Capture error */
    }
}

/* Enable quad mode */
s25fllxx_set_quad_mode(&g_qspid, 1);

/* Erase 64K bytes of chip */
s25fllxx_erase_64k_block(&g_qspid, 0);

/* Flash the code to QSPI flash */
puts("Writing to Memory\r\n");

s25fllxx_write(&g_qspid, (uint32_t *)pBuffercode, WRITE_SIZE, 0, 0);

printf("\r\nExample code written 0x%x bytes to Memory\r\n", WRITE_SIZE);

puts("Verifying \r\n");
s25fllxx_read(&g_qspid, buffer, sizeof(buffer), 0);
/* Start continuous read mode to enter in XIP mode*/
s25fllxx_enter_continuous_read_mode(&g_qspid);

for (idx = 0; idx < WRITE_SIZE; idx++) {
    if (*(memory) == pBuffercode[idx]) {
        memory++;
    } else {
        mem_verified = 1;
        printf("\nData does not match at 0x%x \r\n", (int)memory);
        break;
    }
}

if (!mem_verified) {
    puts("Everything is OK \r\n");
    /* Set PC and SP */
    __start_new = (uint32_t(*) (void)) buffer[1];
    __start_sp = buffer[0];

    puts("\n\r Starting getting started example from QSPI flash \n\r");
    puts("===== \n\r");

    __set_MSP(__start_sp);

    __start_new();
}

puts("Verified failed \r\n");
while(1);
}

```

## 6.5. Application Example Result

The examples in both software package and ASF display the debug information on a terminal window with the following configurations:

- COM Port: Board dependent (refer to your device manager)
- Connection Type: Serial
- Speed: 115200
- Stop bit: 1
- Data Bit: 8
- Parity bit: None
- Flow Control: None

### 6.5.1. Software Package Example Result Window

```
COM7:115200baud - Tera Term VT
```

|  | File   | Edit  | Setup | Control | Window | Help |
|--|--|---|-------|---------|--------|------|
| -- QSPI XIP Example 1.4 --<br>-- SAM U71 Explained Ultra<br>Compiled: Oct 23 2015 04:41:25 With GCC--<br>-I Configure system tick to get 1ms tick period.<br>QSPI drivers initialized<br>Erasing flash memory done..... 100%<br>Writing to Memory<br>Example code written 0x19d4 to Memory<br>Verifying<br>Everything is OK<br><br>Starting getting started example from QSPI flash<br>===== | -- Getting Started Example 1.0 --<br>-- SAM U71 Explained Ultra<br>Compiled: Feb 27 2015 12:09:11 --<br>-I Configure system tick to get 1ms tick period.<br>Configure LED PIOs.<br>Configure TC.<br>No push buttons., uses DBG key 1 & 2 instead.<br>Press 1 to Start/Stop the blue LED D1 blinking.<br>Press 2 to Start/Stop the red LED D2 blinking. | 1 2 2 2 2 1 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 ■ |       |         |        |      |

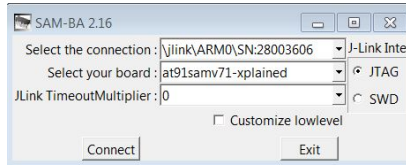
### 6.5.2. ASF Example Result Window

[illegible]

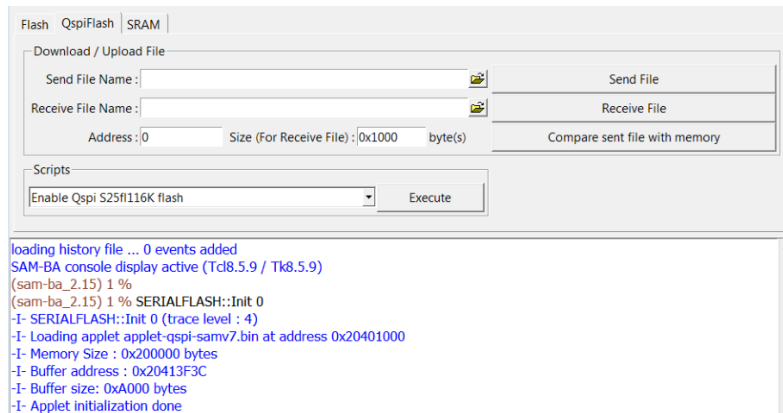
## 7. Programming an Application to QSPI Using SAM-BA

QSPI memory support is available in SAM-BA, through which the generated binary file can be directly loaded to the QSPI via J-Link. The execution sequence is as follows.

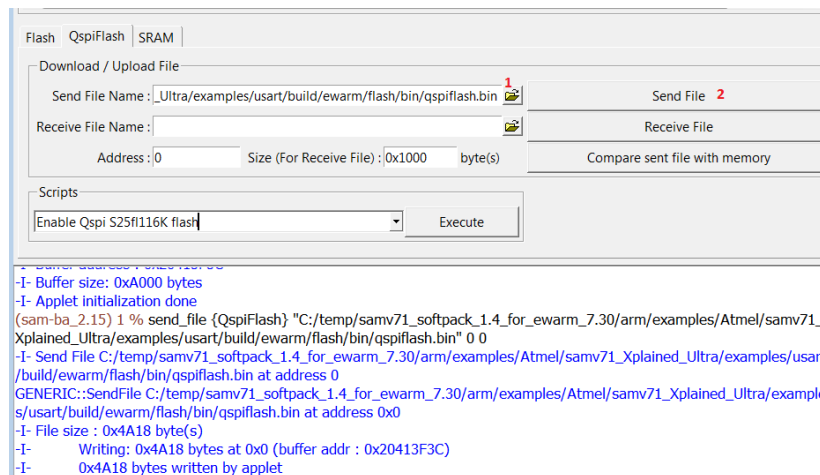
1. Connect to SAM-BA via J-Link.



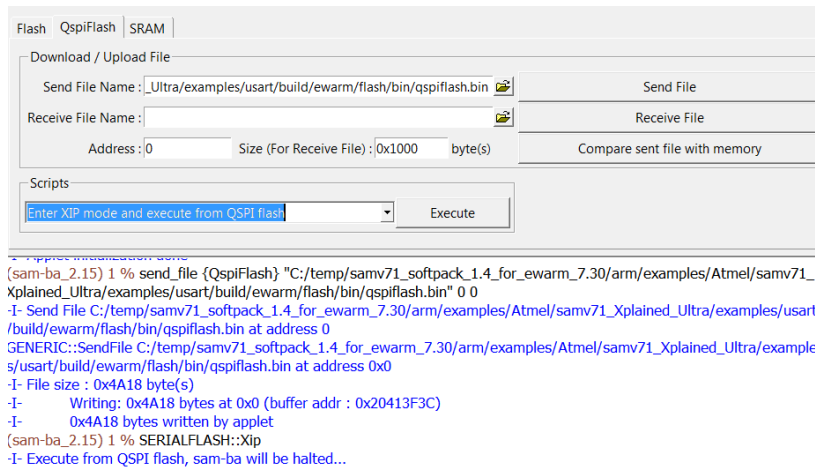
2. Go to the 'QspiFlash' tab. Initialize the QSPI memory by executing the "Enable Qspi S25fl116K flash" command.



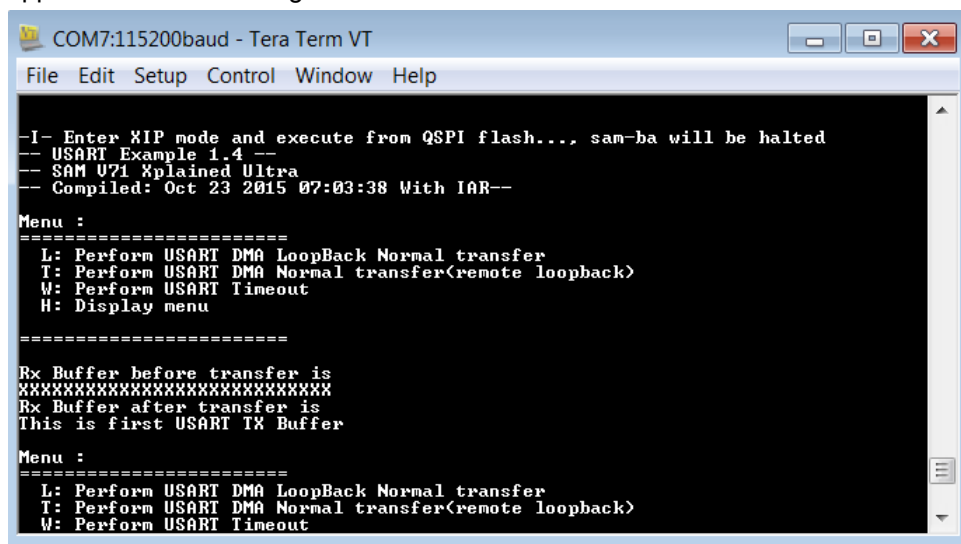
3. Load the binary file generated as explained in the section "[Generating Binary for QSPI Memory Region](#)".



4. Enable XIP mode by executing the "Enter XIP mode and execute from QSPI flash" command.



5. Now the application starts running from QSPI.



## 8. Performance of QSPI in XIP Mode

This section presents various performance numbers, with code/data placed in QSPI against in internal Flash.

### 8.1. AutoBench Performance in QSPI

AutoBench is a suite of benchmarks from EEMBC to predict the performance of MCU or MPU in automotive, industrial, and general-purpose applications. Refer to: [http://www.eembc.org/benchmark/automotive\\_sl.php](http://www.eembc.org/benchmark/automotive_sl.php) for more details.

The figure below displays the results of AutoBench running in QSPI versus running in internal Flash.

Figure 8-1. AutoBench Results in QSPI Vs. Flash

| Auto Mark<br>Note:<br>MPU configuration of QSPI =<br>NON_SHAREABLE | Code : Flash<br>Data : DTCM | Code : QSPI<br>MCK = 150MHz<br>SPCK = 75Mhz<br>Data : DTCM | QSPI Vs Flash |
|--|-----------------------------|--|---------------|
|  | 281                         | 279  | 99.37%        |

### 8.2. Code Structure Impact to QSPI Performance

Assembly code of NOP with different block size and different loop counts was executed, and performance was calculated for QSPI against internal Flash. The results are shared below:

When the code is placed in QSPI, the performance will be affected by the size of the code.

| Code Block Size (byte) | 512   | 2k    | 4k    | 8k    | 12k   | 16k   | 20k   | 24k   | 32k   | 36k   | 40k   |
|------------------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Loop Count             | 48    | 12    | 6     | 3     | 2     | 1     | 1     | 1     | 1     | 1     | 1     |
| Cycle per Instruction  | 0.94  | 1.97  | 3.37  | 6.29  | 11.18 | 17.73 | 17.66 | 17.65 | 17.66 | 17.67 | 17.67 |
| Loop Count             | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 1     |
| Cycle per Instruction  | 18.04 | 17.73 | 17.68 | 17.65 | 17.75 | 17.66 | 17.65 | 17.69 | 17.68 | 17.68 | 17.67 |
| Loop Count             | 20    | 20    | 20    | 20    | 20    | 20    | 20    | 20    | 20    | 20    | 20    |
| Cycle per Instruction  | 1.49  | 1.39  | 1.37  | 1.40  | 1.70  | 1.79  | 8.54  | 12.02 | 13.83 | 15.19 | 14.68 |

#### Result Overview:

- When code of any block size is executed only once, the Cycle per Instruction is similar for all code block size. This is because most part of the time is the time to fetch code from QSPI, and the time of code execution in Cache is relatively a small part.
- As the loop count increases, the Cycle per Instruction reduces. The smaller the code block size is, the more the decrease is.
- The total cycles for executing the code block can be divided into 2 parts:
  - Code fetch from QSPI: the actual fetch is only for 1 time, for all other times from the Cache.
  - Code running from Cache (N-1 times, if the loop is N times).

#### Conclusion:

When code size is smaller than I-Cache:

- The performance in QSPI does not decrease or be affected by QSPI clock. This is due to the fact that with Cache enabled, code fetch from QSPI happens only once. All other times, the code will be fetched from Cache and thus it improves performance, equivalent to embedded Flash.

When code size is larger than I-Cache:

- When a loop or function covers a larger address space over QSPI, it adds more overhead as this will cause frequent access to QSPI.
- In application, keeping the code composed by smaller functions and avoiding loop covering larger space will help reduce the overhead.

### 8.3. Data Access Way Impact to QSPI Performance

A data buffer (around 33KB) was placed in QSPI or SRAM. The number of CPU cycles taken to add all the numbers in a buffer was calculated. The figure below displays the result:

**Figure 8-2. Performance When Data Is Accessed from SRAM Vs. QSPI**

|                             | Code in Flash            |                           |                          |                           | Code in QSPI Flash       |                           |                          |                           |
|-----------------------------|--------------------------|---------------------------|--------------------------|---------------------------|--------------------------|---------------------------|--------------------------|---------------------------|
|                             | Cache Enable             |                           | Cache Disable            |                           | Cache Enable             |                           | Cache Disable            |                           |
|                             | Access by Order (Cycles) | Access by Random (Cycles) | Access by Order (Cycles) | Access by Random (Cycles) | Access by Order (Cycles) | Access by Random (Cycles) | Access by Order (Cycles) | Access by Random (Cycles) |
| Databuff size = 34416 bytes |                          |                           |                          |                           |                          |                           |                          |                           |
| Access Databuff in Flash    | 206412                   | 822683                    | 1282061                  | 1522949                   | 1080283                  | 3854049                   | 6719415                  | 8720564                   |
| Access Databuff in SRAM     | 253958                   | 489889                    | 1204609                  | 1204615                   | 250285                   | 493163                    | 4699657                  | 5251097                   |

#### Result Overview:

- When accessing the data in order, the speed of SRAM is about 4.32 times of the speed of QSPI.
- When accessing the data at random, the speed of SRAM is about 7.81 times of the speed of QSPI.
- When the data buffer is placed in QSPI, the speed of accessing data in order is about 3.57 times of the speed when accessing data at random.
- When the data buffer is placed in SRAM, the speed of accessing data in order is about 1.97 times of the speed when accessing data at random.

#### Conclusion:

With Code in Flash, Cache Enabled and Data buffer in SRAM:

- For access in order, it takes 253958 cycles.
- For access at random, it takes 489889 cycles.

With Code in QSPI Flash, Cache Enabled and Data buffer in SRAM:

- For access in order, it takes 250285 cycles.
- For access at random, it takes 493163 cycles.

In both cases, the number of cycles taken is similar when taking the advantage of Cache. When Cache is not used, if data is placed in QSPI, the following points will help mitigate the overhead:

- Avoid frequent data access to QSPI.
- Avoid random data access to QSPI.

## 9. Frequently Asked Questions [FAQ]

This section covers some of the frequently asked questions regarding QSPI.

### 9.1. Can we boot from QSPI memory?

No, it is not possible to boot from QSPI memory. The system can boot either from Flash or ROM depending on the value of GPNVM1 bit.

### 9.2. What is the maximum size of the serial Flash memory connected over QSPI?

QSPI is mapped at address 0x80000000 when in Serial Memory mode. By default the address is 24 bits long. It can be extended to 32 bits for memory larger than 16MB.

The maximum size of external memory that can be connected over QSPI is 512MB, which is the size of the QSPI memory space area. Refer to the section "Product Mapping" of the product datasheet.

### 9.3. What is the recommended MPU setting for QSPI region?

When accessed by the Cortex-M7 processor for programming operations, the QSPI address space must be defined in the Cortex-M7 Memory Protection Unit (MPU) with the attribute 'Device' or 'Strongly Ordered'. For fetch or read operations, the attribute 'Normal memory' must be set to benefit from the internal cache.

### 9.4. What is the maximum data rate of the Quad I/O Serial Peripheral Interface (QSPI)?

From the "QSPI Characteristics" of "Electrical Characteristics" section of the product datasheet:

$$f_{SCK}^{max} = \frac{1}{QSPI_0(\text{or } QSPI_3) + t_{VALID}}$$

$t_{VALID}$  is the slave time response to output data after detecting a QSCK edge.

The slave response time is available in the respective serial Flash memory datasheet. The example used in this application note demonstrates using "S25FL116K" on SAMV71-XULT board which has a response time of 6-7ns. With the above formula, in this case,  $f_{SCK}$  can reach about ~100MHz. The accurate number is based on the conditions used in the application, as mentioned in the "QSPI Characteristics" of the device datasheet.

For example, for a non-volatile memory with  $t_{VALID}(\text{or } t_v) = 12 \text{ ns}$ , at  $VDDIO = 3.3V$ :

- With 1-bit serial Flash,  $f_{SCK} \text{ max} = 67 \text{ MHz}$ , i.e. 67 Mbps data rate
- With 4-bit serial Flash, 268 Mbps (67\*4) data rate



## 10. Reference Documents

### 10.1. Atmel | SMART SAM S7/SAM E7/SAM V7 Datasheets

Each device datasheet contains block diagrams of the peripherals and details about implementing firmware for the device. It also contains the electrical specifications and expected characteristics of the device.

The datasheets are available in the Documents section of the product pages:

- SAM S70: <http://www.atmel.com/products/microcontrollers/arm/sam-s.aspx?tab=overview#sams70>
- SAM E70: <http://www.atmel.com/products/microcontrollers/arm/sam-e.aspx#same70>
- SAM V70: <http://www.atmel.com/products/microcontrollers/arm/sam-v-mcus.aspx#samv70>
- SAM V71: <http://www.atmel.com/products/microcontrollers/arm/sam-v-mcus.aspx#samv71>

### 10.2. ARM Documentation on Cortex-M7

1. Cortex-M7 Devices Generic User Guide (ARM DUI 0646A)
2. Cortex-M7 Technical Reference Manual (revision r0p2)

These documents are available at <http://www.arm.com/> in the Info Center section.

### 10.3. Reference Application Notes

1. Getting Started with SAM V71 Microcontrollers
2. Atmel AT12874: Getting Started with SAM S70/E70
3. Atmel AT02346: Using the MPU on Atmel Cortex-M3 / Cortex-M4 based Microcontrollers

The application notes can be downloaded from <http://www.atmel.com/products/microcontrollers/arm/default.aspx?tab=documents>

### 10.4. ASF User Manual

1. Atmel AVR4029: Atmel Software Framework - Getting Started
2. Atmel AVR4030: AVR Software Framework - Reference Manual

The above documents can be downloaded from <http://www.atmel.com/tools/avrsoftwareframework.aspx?tab=documents>

## 11. Revision History

Table 11-1. Execute in Place (XIP) with Quad SPI Interface (QSPI) - Rev. 44065A - Revision History

| Date     | Changes      |
|----------|--------------|
| 5-Jan-16 | First issue. |

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. ARM®, ARM Connected® logo, and others are the registered trademarks or trademarks of ARM Ltd. Other terms and product names may be trademarks of others.

**DISCLAIMER:** The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

**SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER:** Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death ("Safety-Critical Applications") without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.