APPLICATION NOTE

Configuring STMicroelectronics' STM32H7A/B to work with Adesto's EcoXiP Octal NOR Flash Memory

Adesto™

www.adestotech.com

Revision History

Revison Number	Date	Tasks
A	04/2020	Initial Release

3600 Peterson Way, Santa Clara, CA 95054 USA | Phone: +1 (408) 400-0578 | www.adestotech.com | e-mail: info@adestotech.com

Introduction

The STM32H7A / STM32H7B microcontrollers from STMicroelectronics are part of the high-performance STM32H7 family, based on the Cortex M7 CPU core. The STM32H7A/B are the first in the family to support an octal-SPI interface and work with Adesto's EcoXiP family of octal flash devices. This note explains how to configure the STM32H7A/B and its built-in flash host controller, called OctoSPI, to take advantage of these EcoXiP features:

- Octal double-data rate interface (compatible with the JEDEC JESD251 standard)
- High clock speed
- Execute in place (XiP)
- Wrap-and-continue

Figure 1 shows a block diagram of EcoXiP connected to a STM32H7A/B MCU highlighting the blocks which interact directly and indirectly with an external flash.

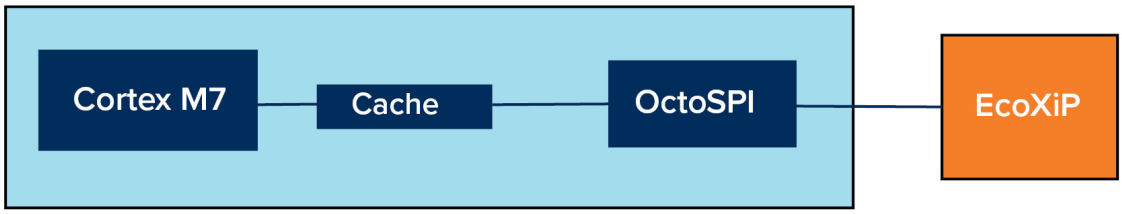

figure 1 - EcoXiP connected to a STM32H7A/B MCU

Attached to this technical note is a code example which runs on top of the STM32Cube SDK and uses its OctoSPI driver and several other drivers.

Note: EcoXiP operates with a voltage supply range of 1.7V-1.95V. If your STM32H7A/B board supports multiple voltage supply options, make sure you select the 1.8V option.

Octal double data-rate interface

EcoXiP comes out of reset in single SPI mode. In order to operate at its highest performance, it should be switched to octal-SPI and double data-rate or DDR mode (sometimes referred to as double transfer rate or DTR mode). In octal-SPI mode, 8 data bits are sent in parallel over 8 I/O signals. In DDR mode, new data is sent on each edge of the clock, falling edge and rising edge. So, in octal-DDR mode, 8 bits are sent each half clock cycle for a total throughput of 16 bits, or 2 bytes, per full clock cycle. Figure 2 below illustrates the signal interface on a typical octal-DDR read transaction.

To configure the system for octal-DDR, these steps are required:

- The host configures the number of dummy cycles (wait states) to be used in read operations between the address phase and data phase. This is done by writing to EcoXiP status/control register 3. The required number of dummy cycles depends on the clock speed. Please refer to the EcoXiP datasheet for specific details.
- The host switches EcoXiP to octal-DDR mode by writing 0x88 to EcoXiP configuration register 2.
- For DDR it is recommended to use an additional signal known as data strobe (also DS or DQS). This is a clock signal used by the slave (flash) to clock out the data it sends when it responds to read commands. EcoXiP always drives DS so no action is required to make it work, however on the host side OctoSPI must be configured to sample the incoming data using the DS signal (by default it doesn't use it – it uses its own clock, the SCK signal, to sample the data).
- At this point each OctoSPI command is configured for octal-DDR mode. OctoSPI allows configuring each element of the command separately: opcode, address, data etc. To work with EcoXiP a command is used in 8-8-8 format (opcode on 8 I/O lines, address on 8 lines, data on 8 lines) as follows:
  - Opcode is configured to be sent on 8 lines; note that the 8-bit opcode element must be sent over a full cycle so essentially it is sent in single data rate (SDR) mode.
  - Address is configured to be sent on 8 lines in DDR format.
  - Data is configured to be received on 8 lines in DDR format.
  - The number of dummy cycles configured earlier on the EcoXiP side must be configured in OctoSPI too so that the master (OctoSPI) and slave (EcoXiP) are in sync.

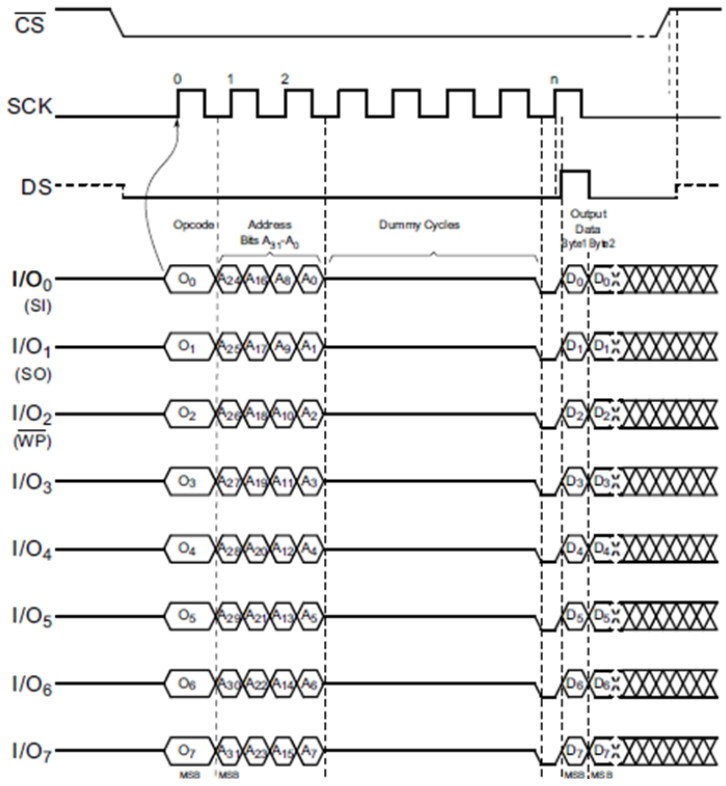

figure 2 - Signal interface on a typical octal-DDR read transaction

High Clock Speed

EcoXiP can operate with a clock frequency of up to 150MHz when it communicates with a host. The OctoSPI host controller on the STM32H7A/B operates at up to 110MHz. Therefore, our example will use 110MHz. To set up the STM32H7A/B to operate in this clock frequency a couple of steps are required:

- Enable I/O speed optimization. In the STM32H7A/B, I/O signals are limited in speed by default. The high-speed optimization option must be enabled to allow those signals to operate at 80MHz and above. Note however that this option is allowed only when the I/O voltage supply is 2.5V or less. Since EcoXiP operates with a voltage range of 1.7V-1.95V, the I/O speed optimization option is applicable to it.
- Modify the OctoSPI source clock. By default, the OctoSPI source clock is the system clock, the same clock which is used by the CPU. This clock reaches a max frequency of 280MHz. It's not possible to divide the 280MHz frequency to generate exactly 110MHz. Therefore, a different clock source is set up using a separate PLL from the STM32H7A/B clock system (PLL2 in our example). Note that it is recommended to divide the clock source by an even number. Dividing by an odd number would generate a non-symmetrical SPI clock.

To enable I/O speed optimization, a couple of option bits must be set. These behave essentially like re-programmable fuses and are physically implemented inside the STM32H7A/B internal flash:

- VDDIO_HSLV: if set it indicates that VDD I/O is below 2.5V and I/O speed optimization at low-voltage feature is allowed.
- VDDMMC_HSLV: if set VDDMMC power rail is operating below 2.5V and I/O speed optimization at low-voltage feature is allowed.

To be able to modify these option bits, options must be unlocked first (they are protected from change by default). This is done by writing special values to the option key register. Once options are unlocked and the above option bits have been set, the option change must be applied by setting the OPTSTART bit of the option configuration register. This essentially programs the fuses associated with these option bits.

All of the above prepares for the final operations of writing the HSLVx bits of the SYSCFG_CCCSR register (compensation cell control/status register). Each of the HSLVx controls a different group of I/Os in the STM32H7A/B.

To set up an alternative OctoSPI clock source with a 110MHz frequency, it is possible to use PLL2 within the RCC (Reset and Clock Control) block of the STM32H7A/B:

- An input clock of 2MHz is used for PLL2.
- The VCO of PLL2 is configured to multiply the frequency by 220 resulting in a 440MHz frequency.
- The 440MHz output is divided by 2 to generate 220MHz frequency for pll2_r_ck clock, one of the derivative clocks of PLL2.
- Finally, pll2_r_ck clock is selected as the source of OctoSPI. Inside the OctoSPI driver the frequency will be divided by 2 to get the desired 110MHz SCK signal to be used for flash communication.

Execute in Place (XiP)

To enable execution of code from the flash, OctoSPI must be configured to operate in memory mapped mode. By default, OctoSPI operates in indirect mode which allows software to send commands to the flash device one by one. In memory mapped mode the STM32H7A/B CPU can read directly from external flash whenever it needs to fetch instructions or data.

In fact, it can read from external flash the same way it reads from any other memory in the system. In contrast to indirect mode, memory mapped mode is fully automated in hardware and no software intervention is required once it's set up.

In more detail, this is how memory mapped mode works. The read request goes on the AHB bus and if it's determined that the address is within the external flash address range, the request is directed to the OctoSPI host controller. OctoSPI, at this point, has already been pre-configured (as part of the transition to memory mapped mode) to use a certain command sequence to implement the read from flash. OctoSPI captures the data returned by the flash during the read transaction and sends it on the bus to complete the CPU read request. This entire process is executed automatically by hardware.

- First, the flash read command to be used for automatic reads in memory mapped mode is configured. As for any command sent by OctoSPI, regardless of its mode of operation, this process configures the command elements: opcode, address, data, and dummy cycles. For best performance we configure octal-DDR mode (as described above) for read commands used in memory mapped mode.
- Note: EcoXiP has a couple of read commands that can be used here, depending on whether the wrap-and-continue feature is enabled. The Read Array command (opcode 0Bh) is the basic read command. The Burst Read with Wrap command (opcode 0Ch) should be used for wrap-and-continue type of read which can speed up read operations and is recommended for the STM32H7A/B as it fully supports this feature. More information about the wrap-and-continue feature is available in the following section.
- Then OctoSPI is switched to memory mapped mode.

Wrap-and-Continue

Wrap-and-continue is a unique feature of EcoXiP which gives it a competitive edge by optimizing read operation from any CPU that uses cache memory, such as the Cortex M7 CPU on the STM32H7A/B.

When a cached CPU fetches instructions or data, it first looks up the address in cache memory. If the requested address is not in the cache, that's a cache-miss event. The request is then directed to the memory bus in order to fetch the instruction/address from one of the system memories. But the CPU doesn't read just one word on a cache miss. It must fill a whole cache line (in the case of Cortex M7, a line is 32 bytes long and starts on a 32-byte boundary). Priority should be given to the word in the address requested by the CPU, the one that caused the cache miss. It's called the critical word. The CPU will perform better if it gets that critical word first. So, if that word falls in the middle of a cache line the optimal read sequence should be implemented as follows:

- Read the critical word first.
- Read the words following the critical word up to the end of the cache line.
- Read the words from the start of the cache line up to, but not including, the critical word.

EcoXiP can do exactly that in one shot with its Burst Read with Wrap command (opcode 0Ch). Not only that, in the likely event that the CPU needs to fill several sequential cache lines, the Burst Read with Wrap command can perform a wrap-and-continue type of read. The first cache line will be read in wrapped mode as described above and the following cache lines will be read sequentially from start to end. All of that with a single read command. This saves the need to send a separate command for each cache line, eliminating the associated overhead of opcode, address and dummy cycles.

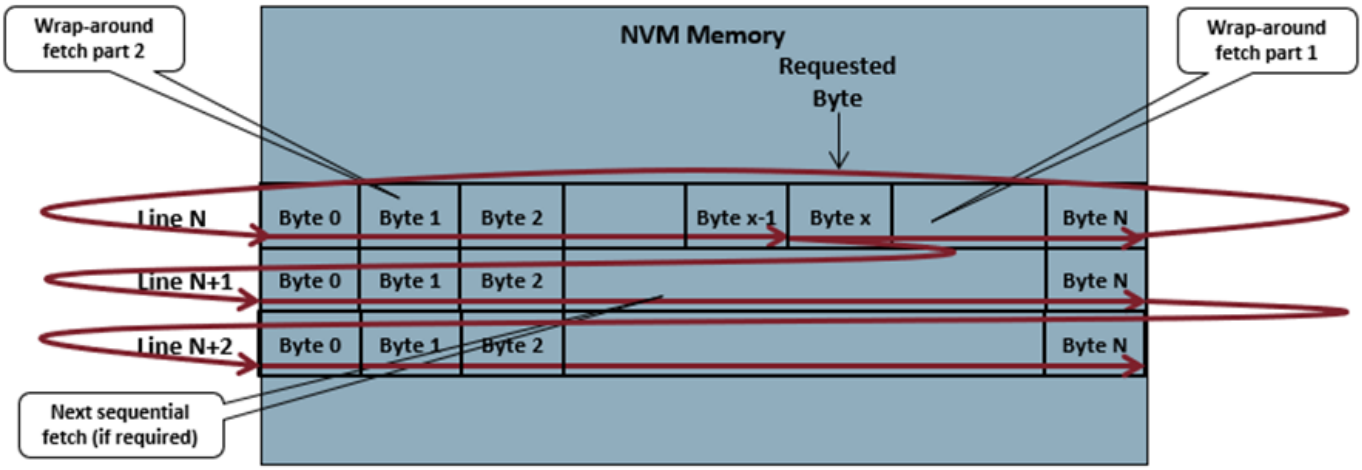

figure 3 - Wrap-and-continue

To enable wrap-and-continue:

- Wrap size should be configured in EcoXiP to 32 bytes by writing to status/control register 3. Note that the bits W[6:5] in register 3 configure the wrap size while bit W7 selects between a wrap read operation and a wrap-and-continue read. For the STM32H7A/B wrap-and-continue option should be selected (bit W7 should be set).
- Wrap size should be configured in OctoSPI to 32 (when wrap size is 0, which is the default, wrapped reads are not allowed by OctoSPI).
- The read command, to be used when the AHB bus sends a wrapped read request, is configured in OctoSPI. As discussed earlier, this should be EcoXiP's Burst Read with Wrap command. Once again, each element of this command can be configured. For best performance, as before, this command should be used in octal-DDR format.
- Note: when wrap-and-continue is enabled, a non-wrapped read option should still be configured in OctoSPI. The wrapped and non-wrapped read commands are configured in two separate registers in OctoSPI.

Notes about Source Code

This note comes with a ZIP file which includes several source files. This is a code example which demonstrates setting up the system for optimal use of EcoXiP on STM32H7A/B as discussed above. The code is implemented on top of the STM32Cube SDK for STM32H7, specifically the STM32Cube_FW_H7_V1.6.0RC3 package. It uses the SDK's API for low-level drivers such as the OctoSPI driver and a few other drivers. The source files do not form a full project: the underlying low-level drivers are not included as they are part of the SDK. The SDK itself can be downloaded from the ST website.

These are the main source files:

- main.c: The main function initializes the system and in particular the flash interface for optimal code execute in place (XiP) from external flash. After this is done it is possible to jump to code located on external flash.
- sysclk.c: includes an initialization of the clock tree including the system clock and other clocks. In particular, it sets up a dedicated clock source for the flash interface (OctoSPI host controller).
- flash.c: includes code for initializing the OctoSPI host controller as well as the EcoXiP flash device and switching to octal-DDR mode and memory mapped mode. It also includes functions for sending the most useful EcoXiP commands in indirect mode.

The code is fully commented, and it should be easy to connect each part of the code with the technical explanations above.

Additional Information

Adesto is a leading provider of innovative, application-specific semiconductors and embedded systems that comprise the essential building blocks of Internet of Things (IoT) edge devices operating on networks worldwide. Our broad portfolio of semiconductor and embedded technologies are optimized for connected IoT devices and systems used in industrial, consumer, communications and medical applications.

Through expert design, unparalleled systems expertise and proprietary intellectual property (IP), our offerings enable customers to differentiate their IoT systems and product designs, leading to improved efficiency, greater reliability and security, integrated intelligence and ultimately lower cost.

Adesto

3600 Peterson Way, Santa Clara, CA 95054 USA | Phone: +1 (408) 400-0578 | www.adestotech.com | e-mail: info@adestotech.com