QSPI Fast Flash Read Library - Programming Guide

Release: 1.0.0
Publication Date: 2023/05/02

Table of Contents

1 Introduction

`lib_qspi_fast_read` is a library for interfacing with a QSPI flash device on an XMOS device. This library achieves higher data transfer bandwidth than the default supplied flash libraries at the cost of flexiblity and write/erase functions.

The SCLK is derived from the system clock, so maximum data rates can only be achieved on the XMOS XS3 architecture.

See *Getting Started* to begin with an XMOS development board.

See *Hardware Selection* for flash part parameters to check for designing your own board.

Find the latest version of `lib_qspi_fast_read` on GitHub.

2 Overview

`lib_qspi_fast_read` is a library for interfacing with a QSPI flash device on an XMOS device to perform quad read operations.

2.1 Capabilities

- Supports SCLK up to 133 MHz versus default lib_quadflash limit of 25 MHz
- Allows support for multiple flash parts, one per IO context
- Quad read 0xEB command only
- Provides startup and shutdown, enabling ports to be reused in some situations
- Supports raw reads and nibble swapped reads to support various flash storage layouts

3 QSPI Fast Read Resource Usage

Each qspi_flash_fast_read peripheral requires ports, clock blocks, compute time, and memory.

This page attempts to capture the requirements for each hardware type with relevant configurations.

Warning: Compute resource usage depends heavily on size of read operations performed.

3.1 Discrete Resources

Resource	Count
1b port	2
4b port	1
clock block	1
thread	<= 1

Note: All operation are performed in place, so thread count usage depends on the application's use case.

3.1.1 Ports

Each `qspi_flash_fast_read` peripheral requires a 1-bit port SCLK, a 1-bit port CS, and a 4-bit port SIO.

3.1.2 Clock Blocks

Each `qspi_flash_fast_read peripheral` requires 1 clock block. This clock block is used to generate SCLK clock from the system frequency clock.

3.1.3 Threads

This library performs all operations in place and does not explicitly require a dedicated thread. The MIPS usage in a given thread varies heavily based on the SCLK frequency, frequency of flash reads, and size of flash reads.

3.2 Compute Requirements

This library has been evaluated to function at a minimum of 75 MIPs, the worst case scenario with a 600 MHz system frequency and all 8 cores scheduling operations.

3.3 Memory

The `qspi_flash_fast_read` library requires approximately 1.8 kB, split between <200 bytes data and <1600 bytes prog.

Each qspi_flash_fast_read peripheral context requires 36 bytes of data.

4 Getting Started

Use of the `qspi_flash_fast_read` peripheral is split into 2 sections below, hardware setup and software use including typical use scenarios.

This document will walk through a use case with the XK-VOICE-L71 board with Winbond W25Q64JW flash part so some steps may need to be adjusted for the particular development board or flash part being used.

4.1 Hardware Setup

To setup the hardware to use this library the flash part must be configured to a drive strength that best suits the applications expected SCLK speed. For the XK-VOICE-L71 1V0A board, the Winbond W25Q64JW flash part the drive strength register must be configured to 50%. Per the datasheet, this can be set by writing the value 0x40 to the status register 3, 0x11.

In the thruput example application a CMake custom target `flash_config_drive_str_50_example_ff_thruput_eval` has been added to wrap the appropriate XFLASH calls. Refer to the XTC Tools Documentation for more information on sending flash commands to the QSPI flash part via XFLASH and a connected XCORE device.

Note: Not all flash parts require this step. For instance, the XCORE_AI_EXPLORER board default drive strength does not need to be changed.

After configuring the flash part, the next step is to flash the appropriate calibration pattern. The calibration pattern is supplied as a binary file, and the user can place this wherever they wish in flash, provided that the address is known when `qspi_flash_fast_read_calibrate()` is called.

In the thruput example application a CMake custom target `flash_calibration_example_ff_thruput_eval_XK_VOICE_L71` has been added to wrap the appropriate XFLASH call to store the pattern at address 0x00000000. Refer to the XTC Tools Documentation for more information on storing in the data partition.

4.2 General Use

The runtime setup of the library consists of initializing the peripheral driver with the XCORE resources to use. Additionally, the user has the ability to setup the driver to populate the data buffer with read data nibble swapped or not. This is an option since depending on how data is stored in flash, the user may need to nibble swap it to be read as expected.

The user then must set up the resources and calibrate. See *Calibration* for more information on the calibration process. After successful calibration, the user may perform reads indefinitely, or until `qspi_flash_fast_read_shutdown()` is called. There is no internal mutual exclusion mechanism and it is up to the user to handle concurrent peripheral use.

```
#include "qspi_flash_fast_read.h"

#define CALIBRATION_PATTERN_ADDR  0x0 // Application specific

...
```

(continues on next page)

(continued from previous page)

```
qspi_fast_flash_read_ctx_t qspi_fast_flash_read_ctx;
qspi_fast_flash_read_ctx_t *ctx = &qspi_fast_flash_read_ctx;

qspi_flash_fast_read_init(ctx,
                          XS1_CLKBLK_1,
                          XS1_PORT_1B,
                          XS1_PORT_1C,
                          XS1_PORT_4B,
                          qspi_fast_flash_read_transfer_raw,
                          3
);

qspi_flash_fast_read_setup_resources(ctx);

uint32_t scratch_buf[QFFR_DEFAULT_CAL_PATTERN_BUF_SIZE_WORDS];
if (qspi_flash_fast_read_calibrate(
        ctx,
        CALIBRATION_PATTERN_ADDR,
        qspi_flash_fast_read_pattern_expect_default,
        scratch_buf,
        QFFR_DEFAULT_CAL_PATTERN_BUF_SIZE_WORDS)
  != 0) {
      printf("Fast flash calibration failed\n");
      /* Error handle here */
}

qspi_flash_fast_read(ctx, /* Some address */, /* buffer to populate */, /* bytes to read */
↪);

qspi_flash_fast_read_shutdown(ctx);
```

4.3 Reuse Resources

The library supports shutting down of the peripheral and a minimal method to reinstantiate flash access. This enables the user to potentially reuse ports or clock blocks when flash access is not required. After shutdown, the resources are free to be used elsewhere. The application is responsible for preventing reads from happening, as there is no internal protection. When flash is needed again the resources can be setup again and calibration re-applied.

```
qspi_flash_fast_read_shutdown(ctx);

/* Re-use XCORE resources here */
{
  ;
}

/* Setup flash fast read */
qspi_flash_fast_read_setup_resources(ctx);
qspi_flash_fast_read_apply_calibration(ctx);
```

Warning: If reusing the ports be aware that there is still a QSPI flash part connected.

4.4 Nibble Swapping

The transfer mode is set during initialization but may be changed during runtime when an operation is not in progress. Here is an example of how to perform one read in nibble swapped mode before returning back to raw.

```
qspi_flash_fast_read_mode_set(ctx, qspi_fast_flash_read_transfer_nibble_swap);
qspi_flash_fast_read(ctx, /* Some address */, /* buffer to populate */, /* bytes to read */
↪);
qspi_flash_fast_read_mode_set(ctx, qspi_fast_flash_read_transfer_raw);
```

4.5 Multiple Peripherals

It is possible to use this library with multiple flash parts. Below is an example of instantiating 2 drivers with separate XCORE resources.

```
qspi_fast_flash_read_ctx_t qspi_fast_flash_read_ctx;
qspi_fast_flash_read_ctx_t *ctx = &qspi_fast_flash_read_ctx;

qspi_fast_flash_read_ctx_t qspi_fast_flash_read_second_ctx;
qspi_fast_flash_read_ctx_t *ctx_2 = &qspi_fast_flash_read_second_ctx;

qspi_flash_fast_read_init(ctx,
                          XS1_CLKBLK_1,
                          XS1_PORT_1B,
                          XS1_PORT_1C,
                          XS1_PORT_4B,
                          qspi_fast_flash_read_transfer_raw,
                          3
);

qspi_flash_fast_read_setup_resources(ctx);

uint32_t scratch_buf[QFFR_DEFAULT_CAL_PATTERN_BUF_SIZE_WORDS];
if (qspi_flash_fast_read_calibrate(
        ctx,
        CALIBRATION_PATTERN_ADDR,
        qspi_flash_fast_read_pattern_expect_default,
        scratch_buf,
        QFFR_DEFAULT_CAL_PATTERN_BUF_SIZE_WORDS)
  != 0) {
      printf("Fast flash calibration failed\n");
      /* Error handle here */
}

qspi_flash_fast_read_init(ctx_2,
```

(continues on next page)

(continued from previous page)

```
                          XS1_CLKBLK_2,
                          XS1_PORT_1E,
                          XS1_PORT_1F,
                          XS1_PORT_4C,
                          qspi_fast_flash_read_transfer_raw,
                          3
);

qspi_flash_fast_read_setup_resources(ctx_2);

uint32_t scratch_buf[QFFR_DEFAULT_CAL_PATTERN_BUF_SIZE_WORDS];
if (qspi_flash_fast_read_calibrate(
        ctx_2,
        CALIBRATION_PATTERN_ADDR2,
        qspi_flash_fast_read_pattern_expect_default,
        scratch_buf,
        QFFR_DEFAULT_CAL_PATTERN_BUF_SIZE_WORDS)
  != 0) {
      printf("Fast flash calibration failed\n");
      /* Error handle here */
}
```

Note: It may be possible to share SCLK and SIO pins and only use different CS and clock blocks to save on 1-bit ports. This configuration would be heavily dependent on the flash parts being used and their pin states when CS is not asserted and the length of traces run. It would be up to the application programmer to handle mutual exclusion of ports.

5 Hardware Selection

This library has hardware compatibility requirements for the flash part:

- The flash part MUST support the Quad Read command 0xEB.
- The flash part MUST support 3 byte addresses followed by 6 dummy cycles
- The flash part MUST support a minimum of 50 MHz SCLK is using a 600 MHz XCORE system frequency, or 66 MHz SCLK if using an 800 MHz XCORE system frequency.
- The configuration to use the flash part MUST be compatible with `lib_quadflash`. Changing nonvolatile flash parameters for use with `lib_flash_fast_read` could result in incompatibilities with the bootloader. Refer to the XTC Tools Documentation for details on supporting flash parts.
- The flash part MUST be configured to an 8mA IO drive strength to ensure the largest window for successful calibration. The method to do this will vary per flash part and the user must consult the data sheet on how to set this parameter nonvolatile.

It is advised to carefully review the datasheet of any particular flash part as some only support speeds 100+ MHz with additional requirements that may be incompatible with the bootloader. For example some only support higher speeds when a special mode is enabled where the least significant bits of the address are assumed to be 0, and thus all Quad Reads must be word aligned. Others may require additional dummy cycles. If the flash is the same used for the bootloader these values must not be changed, otherwise unexpected power cycling will prevent the application from being able to be loaded.

6 Calibration

This library achieves SCLK speeds higher than 75MHz by utilizing a calibration routine to ensure that the sampling port time has sufficient head and tail room in the SIO data valid window. This is achieved by reading a known pattern from flash with various timing parameters which gradually move the sampling time within the bounds of XCORE port behavour. From these tests, if a sufficient window is found, the timing parameters are saved and the ideal configuration can be set by the library API. This pattern is handcrafted to have the worst case EMI transitions on the wire.

Two calibration pattern binary files intended to be stored in flash. Both contain the same data, but one is nibble swapped, allowing the application programmer to choose the flash read mode that best fits their use case.

7 API Reference

This library can be used to instantiate and read from a Quad SPI flash I/O interface on XCORE.

7.1 Initialization API

The following structures and functions are used to initialize a `qspi_flash_fast_read` peripheral instance.

void `qspi_flash_fast_read_init`(*qspi_fast_flash_read_ctx_t* *ctx, xclock_t clock_block, port_t cs_port, port_t sclk_port, port_t sio_port, *qspi_fast_flash_read_transfer_mode_t* mode, uint8_t divide)

Implements a qspi_flash_fast_read device.

Only supported for use with flash parts with 0xEB Fast Quad read instruction with: Cycles Data Transfer IO 8 0xEB SPI 6 3 byte address QSPI 6 Dummy QSPI (note may be mode byte + 2 dummy bytes) x Data read QSPI

Parameters

- `ctx` – A pointer to the qspi_flash_fast_read context to initialize.
- `clock_block` – The clock block that will get configured for use with sclk.
- `cs_port` – The chip select port. Must be a 1-bit port.
- `sclk_port` – The SCLK port. Must be a 1-bit port.
- `sio_port` – The SIO port. Must be a 4-bit port.
- `mode` – The transfer mode to use for port reads. Invalid values will default to qspi_fast_flash_read_transfer_raw
- `divide` – The divisor to use for QSPI SCLK. Must be between 3 and 6 inclusive. SCLK frequency will be set to CORE_CLOCK / (2 * divide) CORE CLOCK 600MHz 800MHz CLK_DIVIDE SCLK FREQ 3 100MHz 133MHz 4 75MHz 100MHz 5 60MHz 80MHz 6 50MHz 66MHz Values less than 3 will be implicity set to 3. Values greater than 6 will be implicity set to 6

7.2 Core API

The following functions are the core `qspi_flash_fast_read` driver functions that are used after it has been initialized.

enum `qspi_fast_flash_read_transfer_mode_t`

Transfer modes for fast read operations.

Values:

enumerator `qspi_fast_flash_read_transfer_raw`

Do not nibble swap port ins

enumerator **`qspi_fast_flash_read_transfer_nibble_swap`**

Nibble swap port ins

typedef struct *qspi_flash_fast_read_struct* **`qspi_fast_flash_read_ctx_t`**

Type representing a qspi_flash_fast_read context.

uint32_t
`qspi_flash_fast_read_pattern_expect_default`[*QFFR_DEFAULT_CAL_PATTERN_BUF_SIZE_WORDS*]

Declaration of array which holds the default calibration pattern.

This pattern was selected to maximize potential EMI and crosstalk.

void **`qspi_flash_fast_read_setup_resources`**(*qspi_fast_flash_read_ctx_t* *ctx)

Setup and initialize XCORE resources for a qspi_flash_fast_read device.

Parameters

- `ctx` – A pointer to the qspi_flash_fast_read context to use.

int32_t **`qspi_flash_fast_read_calibrate`**(*qspi_fast_flash_read_ctx_t* *ctx, uint32_t addr, uint32_t *expect_buf, uint32_t *scratch_buf, size_t len_words)

Calibrate port delays and timings for a qspi_flash_fast_read device.

This must be called after *qspi_flash_fast_read_init()* and *qspi_flash_fast_read_setup_resources()*.

On success *qspi_flash_fast_read_apply_calibration()* is called implicitly.

Parameters

- `ctx` – A pointer to the qspi_flash_fast_read context to use.
- `addr` – The address of the calibration pattern.
- `expect_buf` – A pointer to the byte array with the expected calibration pattern.
- `scratch_buf` – A pointer to an application provided scratch buffer. This buffer must be at least len_words words. Contents will be overwritten. After this function call the buffer is owned by the application and no longer needed.
- `len_words` – The length in words of the calibration pattern

Returns

0 on success -1 on failure

void **`qspi_flash_fast_read_apply_calibration`**(*qspi_fast_flash_read_ctx_t* *ctx)

Apply calibration for a qspi_flash_fast_read device.

Parameters

- `ctx` – A pointer to the qspi_flash_fast_read context to use.

void **`qspi_flash_fast_read`**(*qspi_fast_flash_read_ctx_t* *ctx, uint32_t addr, uint8_t *buf, size_t len)

Perform a flash read with a qspi_flash_fast_read device.

Parameters

- `ctx` – A pointer to the qspi_flash_fast_read context to use.
- `addr` – The address to read.
- `buf` – A pointer to the byte array to save the received data. This must begin on a 4-byte boundary.

- `len` – The number of bytes to input.

void `qspi_flash_fast_read_mode_set(`*qspi_fast_flash_read_ctx_t* *ctx, *qspi_fast_flash_read_transfer_mode_t* mode`)`

Set read mode for a qspi_flash_fast_read device.

Parameters

- `ctx` – A pointer to the qspi_flash_fast_read context to use.
- `mode` – The transfer mode to set.

qspi_fast_flash_read_transfer_mode_t `qspi_flash_fast_read_mode_get(`*qspi_fast_flash_read_ctx_t* *ctx`)`

Get read mode for a qspi_flash_fast_read device.

Parameters

- `ctx` – A pointer to the qspi_flash_fast_read context to use.

Returns

The current configured transfer mode.

void `qspi_flash_fast_read_shutdown(`*qspi_fast_flash_read_ctx_t* *ctx`)`

Shutdown a qspi_flash_fast_read device.

Resets the XCORE resources for application to use.

Calibration is stored internally for future use.

Parameters

- `ctx` – A pointer to the qspi_flash_fast_read context to use.

`QFFR_DEFAULT_CAL_PATTERN_BUF_SIZE_WORDS`

Size in words of the default calibration pattern.

struct `qspi_flash_fast_read_struct`

#include <qspi_flash_fast_read.h> Struct to hold a qspi_flash_fast_read context.

The members in this struct should not be accessed directly.

7 Index