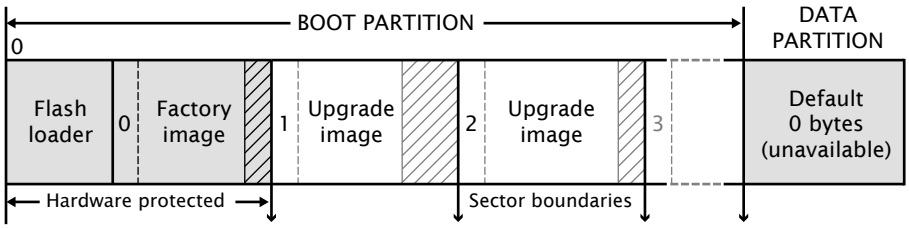
libquadflash API

IN THIS DOCUMENT

- General Operations
- Boot Partition Functions
- Data Partition Functions

The libquadflash library provides functions for reading and writing data to Quad-SPI flash devices that use the xCORE format shown in the diagram below.

Figure 1: Flash format diagram



All functions are prototyped in the header file `<quadflash.h>`. Except where otherwise stated, functions return 0 on success and non-zero on failure.

1 General Operations

The program must explicitly open a connection to the Quad-SPI device before attempting to use it, and must disconnect once finished accessing the device.

The functions `fl_connect` and `fl_connectToDevice` require an argument of type `fl_QSPIPorts`, which defines the four ports and clock block used to connect to the device.

```
typedef struct {
  out port qspiCS;
  out port qspiSCLK;
  out buffered port:32 qspiSIO;
  clock qspiClkblk;
} fl_QSPIPorts;
```


Publication Date: 2016/10/31 Document Number: XM010841C

Function	fl_connect
Description	`fl_connect` opens a connection to the specified Quad-SPI device.
Type	`int fl_connect(fl_QSPIPorts *SPI)`

Function	fl_connectToDevice
Description	`fl_connectToDevice` opens a connection to an Quad-SPI device. It iterates through an array of *n* Quad-SPI device specifications, attempting to connect using each specification until one succeeds.
Type	`int fl_connectToDevice(fl_QSPIPorts *SPI, fl_DeviceSpec spec[], unsigned n)`

Function	fl_getFlashType
Description	`fl_getFlashType` returns an `enum` value for the flash device. The enumeration of devices known to libquadflash is given below.

```
typedef enum {
  UNKNOWN = 0,
  ISSI_IS25LQ016B,
  ISSI_IS25LQ032B,
  ISSI_IS25LQ080B,
  SPANSION_S25FL116K,
  SPANSION_S25FL132K,
  SPANSION_S25FL164K,
} fl_QuadFlashId;
```

	If the function call `fl_connectToDevice(p, spec, n)` is used to connect to a flash device, `fl_getFlashType` returns the parameter value `spec[`*i*`].flashId` where *i* is the index of the connected device.
Type	`int fl_getFlashType(void)`

Function	fl_getFlashSize
Description	`fl_getFlashSize` returns the capacity of the Quad-SPI device in bytes.
Type	`unsigned fl_getFlashSize(void)`

Function	fl_disconnect
Description	`fl_disconnect` closes the connection to the Quad-SPI device.
Type	`int fl_disconnect(void)`

2 Boot Partition Functions

By default, the size of the boot partition is set to the size of the flash device. Access to boot images is provided through an iterator interface.

Function	fl_getFactoryImage
Description	`fl_getFactoryImage` provides information about the factory boot image.
Type	`int fl_getFactoryImage(fl_BootImageInfo *bootImageInfo)`

Function	fl_getNextBootImage
Description	`fl_getNextBootImage` provides information about the next upgrade image. Once located, an image can be upgraded. Functions are also provided for reading the contents of an upgrade image.
Type	`int fl_getNextBootImage(fl_BootImageInfo *bootImageInfo)`

Function	fl_getImageVersion
Description	`fl_getImageVersion` returns the version number of the specified image.
Type	`unsigned fl_getImageVersion(fl_BootImageInfo *bootImageInfo)`

Function	fl_startImageReplace
Description	`fl_startImageReplace` prepares the Quad-SPI device for replacing an image. The old image can no longer be assumed to exist after this call. Attempting to write into the data partition or the space of another upgrade image is invalid. A non-zero return value signifies that the preparation is not yet complete and that the function should be called again. This behavior allows the latency of a sector erase to be masked by the program.
Type	`int` `fl_startImageReplace(fl_BootImageInfo *, unsigned maxsize)`

Function	fl_startImageAdd
Description	`fl_startImageAdd` prepares the Quad-SPI device for adding an image after the specified image. The start of the new image is at least padding bytes after the previous image. Attempting to write into the data partition or the space of another upgrade image is invalid. A non-zero return value signifies that the preparation is not yet complete and that the function must be called again. This behavior allows the latency of a sector erase to be masked by the program.

Type	`int fl_startImageAdd(fl_BootImageInfo*,`
	`unsigned maxsize,`
	`unsigned padding)`

Function	fl_startImageAddAt
Description	`fl_startImageAddAt` prepares the Quad-SPI device for adding an image at the specified address offset from the base of the first sector after the factory image. Attempting to write into the data partition or the space of another upgrade image is invalid. A non-zero return value signifies that the preparation is not yet complete and that the function must be called again.
Type	`int fl_startImageAddAt(unsigned offset, unsigned maxsize)`

Function	fl_writeImagePage
Description	`fl_writeImagePage` waits until the Quad-SPI device is able to accept a request and then outputs the next page of data to the device. Attempting to write past the maximum size passed to `fl_startImageReplace`, `fl_startImageAdd` or `fl_startImageAddAt` is invalid.

Continued on next page

Type	int fl_writeImagePage(const unsigned char page[])

Function	fl_writeImageEnd
Description	fl_writeImageEnd waits until the Quad-SPI device has written the last page of data to its memory.
Type	int fl_writeImageEnd(void)

Function	fl_startImageRead
Description	fl_startImageRead prepares the Quad-SPI device for reading the contents of the specified upgrade image.
Type	int fl_startImageRead(fl_BootImageInfo *b)

Function	fl_readImagePage
Description	fl_readImagePage inputs the next page of data from the Quad-SPI device and writes it to the array page.
Type	int fl_readImagePage(unsigned char page[])

Function	fl_deleteImage
Description	fl_deleteImage erases the upgrade image with the specified image.
Type	int fl_deleteImage(fl_BootImageInfo* b)

3 Data Partition Functions

All flash devices are assumed to have uniform page sizes but are not assumed to have uniform sector sizes. Read and write operations occur at the page level, and erase operations occur at the sector level. This means that to write part of a sector, a buffer size of at least one sector is required to preserve other data.

In the following functions, writes to the data partition and erasures from the data partition are not fail-safe. If the operation is interrupted, for example due to a power failure, the data in the page or sector is undefined.

Function	fl_getDataPartitionSize
Description	`fl_getDataPartitionSize` returns the size of the data partition in bytes.
Type	`unsigned fl_getDataPartitionSize(void)`

Function	fl_readData
Description	`fl_readData` reads a number of bytes from an offset into the data partition and writes them to the array dst.
Type	`int fl_readData(unsigned offset, unsigned size, unsigned char dst[])`

Function	fl_getWriteScratchSize
Description	`fl_getWriteScratchSize` returns the buffer size needed by `fl_writeData` for the given parameters.
Type	`unsigned fl_getWriteScratchSize(unsigned offset, unsigned size)`

Function	fl_writeData
Description	`fl_writeData` writes the array `src` to the specified offset in the data partition. It uses the array `buffer` to preserve page data that must be re-written.
Type	`int fl_writeData(unsigned offset, unsigned size, const unsigned char src[], unsigned char buffer[])`

3.1 Page-Level Functions

Function	fl_getPageSize
Description	`fl_getPageSize` returns the page size in bytes.
Type	`unsigned fl_getPageSize(void)`

Function	fl_getNumDataPages
Description	`fl_getNumDataPages` returns the number of pages in the data partition.
Type	`unsigned fl_getNumDataPages(void)`

Function	fl_writeDataPage
Description	`fl_writeDataPage` writes the array data to the *n*-th page in the data partition. The data array must be at least as big as the page size; if larger, the highest elements are ignored.
Type	`unsigned fl_writeDataPage(unsigned n, const unsigned char data[])`

Function	fl_readDataPage
Description	`fl_readDataPage` reads the *n*-th page in the data partition and writes it to the array data. The size of data must be at least as large as the page size.
Type	`unsigned fl_readDataPage(unsigned n, unsigned char data[])`

3.2 Sector-Level Functions

Function	fl_getNumDataSectors
Description	`fl_getNumDataSectors` returns the number of sectors in the data partition.
Type	`unsigned fl_getNumDataSectors(void)`

Function	fl_getDataSectorSize
Description	`fl_getDataSectorSize` returns the size of the *n*-th sector in the data partition in bytes.
Type	`unsigned fl_getDataSectorSize(unsigned n)`

Function	fl_eraseDataSector
Description	`fl_eraseDataSector` erases the *n*-th sector in the data partition.
Type	`unsigned fl_eraseDataSector(unsigned n)`

Function	fl_eraseAllDataSectors
Description	`fl_eraseAllDataSectors` erases all sectors in the data partition.
Type	`unsigned fl_eraseAllDataSectors(void)`