

mbd OS Adding Storage Support



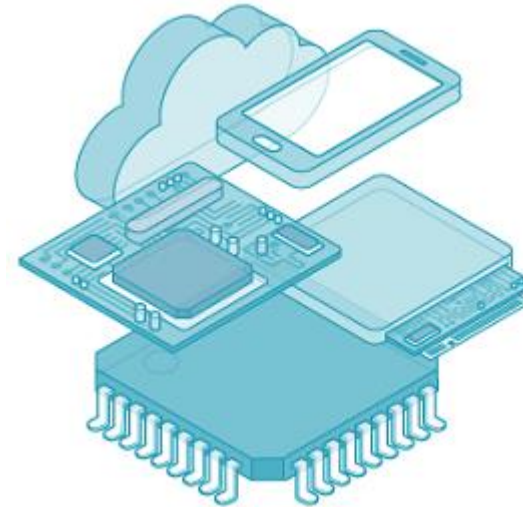
Sam Grove, Chris Haster, Simon Hughes

Silicon Partner Workshop - Wyboston Lakes
March 2017

Agenda

- Mbed OS 5.4 Storage: What's New?
 - Block Device API.
 - FAT32 Filesystem SDCard Support.
 - Block Device examples.
 - Documentation.
- Storage Technical Overview.
 - Filesystem top down view.
 - Memory Technology Device bottom up view.
 - Block Device API.
 - The SDCard Driver (SDBlockDevice).
 - The SPI NOR Flash Driver (SPIFBlockDevice).
 - Resource Usage.
- Conclusions

mbed OS

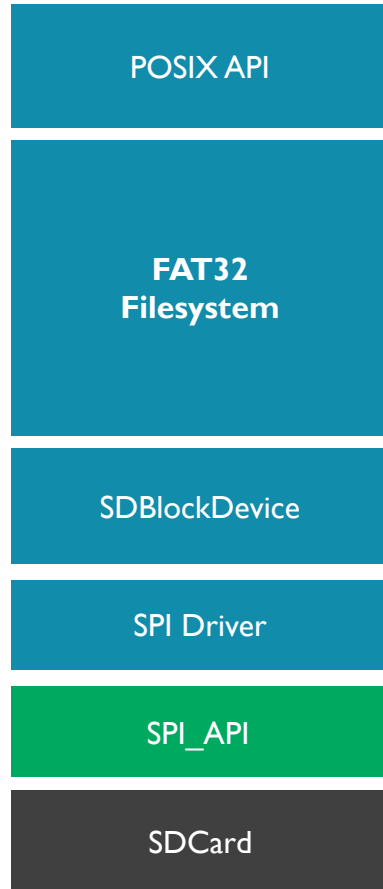


mbled OS 5.4 Storage What's New?

Block Store API: class BlockDevice

```
/** A hardware device capable of writing and reading blocks
 */
class BlockDevice
{
public:
    virtual ~BlockDevice() {};
    virtual int init() = 0;
    virtual int deinit() = 0;
    virtual int read(void *buffer, bd_addr_t addr, bd_size_t size) = 0;
    virtual int program(const void *buffer, bd_addr_t addr, bd_size_t size) = 0;
    virtual int erase(bd_addr_t addr, bd_size_t size) = 0;
    virtual bd_size_t get_read_size() const = 0;
    virtual bd_size_t get_program_size() const = 0;
    virtual bd_size_t get_erase_size() const = 0;
    virtual bd_size_t size() const = 0;
    bool is_valid_read(bd_addr_t addr, bd_size_t size) const;
    bool is_valid_program(bd_addr_t addr, bd_size_t size) const;
    bool is_valid_erase(bd_addr_t addr, bd_size_t size) const;
};
```

sd-driver repository: The SDCard Driver



The SDCard driver

- SDBlockDevice: <https://github.com/armmbed/sd-driver>

ARMmbed / sd-driver

Unwatch 86 Star 0 Fork 1

Code Issues 0 Pull requests 1 Projects 0 Wiki Pulse Graphs Settings

mbed-os SD card driver and associated test cases

Add topics

11 commits 2 branches 3 releases 1 contributor

Branch: master New pull request

Create new file Upload files Find file Clone or download

File	Commit Message	Time Ago
config	Ported mbed 2 FAT32 filesystem test cases to mbed 5 tests in basic.cpp.	25 days ago
docs/pics	Updated README.md to include worked examples and restructuring of inf...	12 days ago
features	Fixing SDBlockDevice::size() and SPIFBlockDevice::size() to implement...	7 days ago
README.md	Updated version of README.md describing how to overcome build problem...	6 days ago

README.md

mbed OS SDCard Driver (sd-driver) for FAT32 Filesystem Support

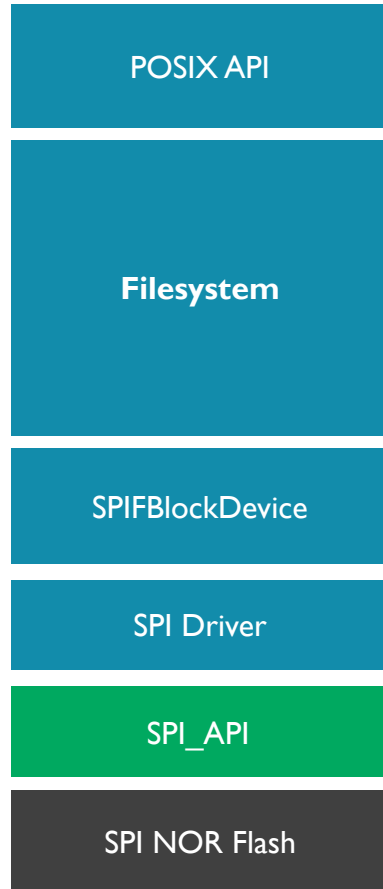
Simon Hughes

20170329

Version 1.00



spiflash-driver repository: SPI NOR Flash Driver



The SPI NOR driver

A screenshot of the GitHub repository page for 'ARMmbed/spiflash-driver'. The page shows the repository name, statistics (6 watches, 0 stars, 1 fork), and navigation tabs (Code, Issues, Pull requests, Projects, Wiki, Pulse, Graphs). Below the repository name, it states 'Block device driver for NOR SPI flash devices that support SFDP'. A table lists recent commits, including updates to match API changes in mbed OS and initial commits for LICENSE, README.md, SPIFBlockDevice.cpp, and SPIFBlockDevice.h. The 'README.md' file is expanded, showing the title 'SPI Flash Driver', a description of the block device driver, and a code example for using the MX25R SPI flash device on the K82F.

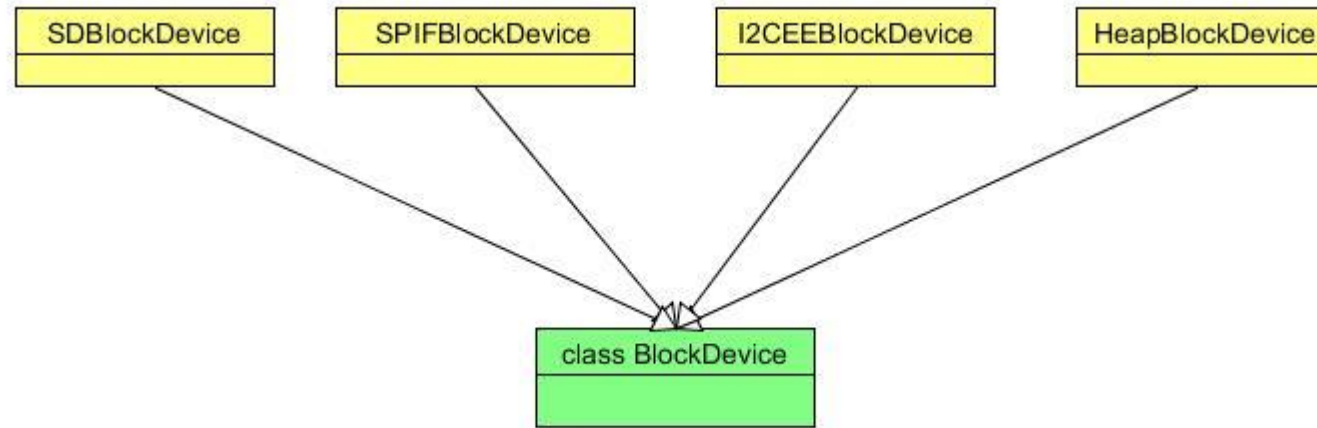
Commit	Message	Time
geky	Updated to match API changes in mbed OS	Latest commit 6e3dc9 14 days ago
TESTS/block_device/spif	Updated to match API changes in mbed OS	14 days ago
LICENSE	Initial commit of SPI flash device	a month ago
README.md	Initial commit of SPI flash device	a month ago
SPIFBlockDevice.cpp	Updated to match API changes in mbed OS	14 days ago
SPIFBlockDevice.h	Updated to match API changes in mbed OS	14 days ago

```
// Here's an example using the MX25R SPI flash device on the K82F
#include "mbed.h"
#include "SPIFBlockDevice.h"
```



- SPIFBlockDevice: <https://github.com/armmbed/spiflash-driver>

Block Device Examples:



- SDBlockDevice example: <https://github.com/armmbed/mbed-os-example-sd-driver>
- SPIFBlockDevice example: <https://github.com/armmbed/mbed-os-example-spiflash-driver>
- I2CEEBlockDevice example: <https://github.com/armmbed/mbed-os-example-i2ceeprom-driver>
- HeapBlockDevice:
<https://github.com/ARMmbed/mbed-os/blob/master/features/filesystem/bd/HeapBlockDevice.h>

Documentation: Storage APIs

The screenshot shows the ARM mbed OS API Reference documentation page for Storage APIs. The page has a teal header with navigation links: Hardware, Documentation, Code, Questions, Forum, and Compiler. Below the header, the breadcrumb trail is 'Docs / mbed OS API References / Storage APIs'. The main title 'Storage APIs' is displayed in a large font. On the left, a sidebar contains a list of API categories: Introduction to the mbed OS API References, Task management APIs, Input and output APIs, Digital interface APIs, Communication APIs, Security APIs, Storage APIs (highlighted), Storage APIs (expanded), File system, Block devices, API documentation, and Other sources. The main content area, titled 'Storage APIs', states: 'The storage APIs present in mbed OS are:' followed by a bulleted list: 'File system: a common interface for using file systems on block devices.' and 'Block device: a common interface for block-based storage devices.' Below the list is a 'Previous' button. At the bottom, the copyright notice reads: '© ARM Ltd. Copyright 2016 – ARM mbed IoT Device Platform'.

Hardware Documentation Code Questions Forum Compiler

Docs / mbed OS API References / Storage APIs

Storage APIs

- Introduction to the mbed OS API References
- Task management APIs
- Input and output APIs
- Digital interface APIs
- Communication APIs
- Security APIs
- Storage APIs**
- ▼ Storage APIs
 - File system
 - Block devices
- API documentation
- Other sources

Storage APIs

The storage APIs present in mbed OS are:

- **File system**: a common interface for using file systems on block devices.
- **Block device**: a common interface for block-based storage devices.

◀ Previous

© ARM Ltd. Copyright 2016 – ARM mbed IoT Device Platform

- https://docs.mbed.com/docs/mbed-os-api-reference/en/latest/APIs/storage/storage_index/

Documentation: File System

The screenshot shows the mbed OS API documentation website. The top navigation bar includes links for Hardware, Documentation, Code, Questions, Forum, and Compiler. The breadcrumb trail indicates the path: Docs / mbed OS API References / Storage APIs. The main heading is 'File system'. On the left, a sidebar menu lists various API categories, with 'Storage APIs' and 'File system' expanded. The 'File system' section includes links for 'Example', 'Declaring a file system', and 'C++ classes'. The main content area contains the following text:

File system

The file system API provides a common interface for implementing a [file system](#) on a [block-based storage device](#). The file system API is a class-based interface, but implementing the file system API provides the standard POSIX file API familiar to C users.

Example

Here is what it looks like to use the file system in mbed OS:

[FAT file system example](#)

Declaring a file system

The `FileSystem` class provides the core API for file system operations. You must provide a block device to back the file system. When you declare a file system with a name, you can open files on the file system through standard POSIX functions (see [open](#) or [fopen](#)).

Existing file systems:

The [FAT file system](#) is a standard file system.

The [BlockDevice](#) class provides the underlying API for representing block-based storage that can be used to back a file system. mbed OS provides standard interfaces for the more common storage media, and you can extend the `BlockDevice` class to provide support for unsupported storage.

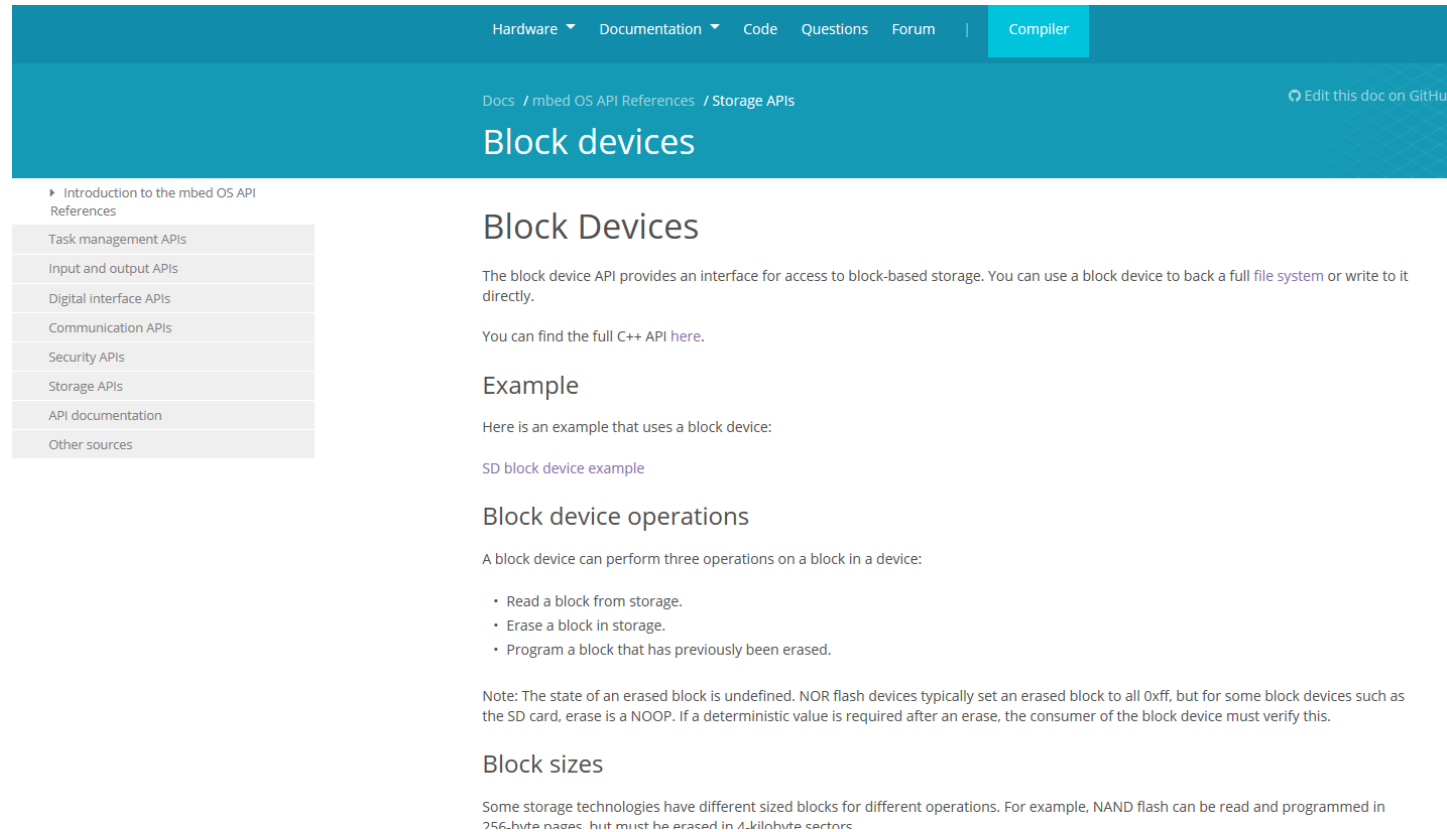
Existing block devices:

- [SDBlockDevice](#) - block device for SD cards.
- [SPIFlashBlockDevice](#) - block device for NOR based SPI flash devices.
- [I2CEEPROMBlockDevice](#) - block device for EEPROM based I2C devices.
- [HeapBlockDevice](#) - block device backed by the heap for quick testing.

A URL box at the bottom left contains the link: <https://github.com/armmbed/mbed-os-example-fat-file-system>

- <https://docs.mbed.com/docs/mbed-os-api-reference/en/latest/APIs/storage/filesystem/>

Documentation: Block Devices



The screenshot shows the mbed OS API Reference documentation page for Block Devices. The page has a teal header with navigation links: Hardware, Documentation, Code, Questions, Forum, and Compiler. Below the header, the breadcrumb trail is 'Docs / mbed OS API References / Storage APIs'. The main title is 'Block devices'. On the left, there is a sidebar with a list of API categories: Introduction to the mbed OS API References, Task management APIs, Input and output APIs, Digital interface APIs, Communication APIs, Security APIs, Storage APIs (highlighted), API documentation, and Other sources. The main content area has a sub-header 'Block Devices' and a paragraph explaining that the block device API provides an interface for access to block-based storage. It mentions that you can use a block device to back a full file system or write to it directly. A link is provided to find the full C++ API. Below this is an 'Example' section with a link to an 'SD block device example'. The 'Block device operations' section lists three operations: Read a block from storage, Erase a block in storage, and Program a block that has previously been erased. A note explains that the state of an erased block is undefined and that NOR flash devices typically set an erased block to all 0xff. The 'Block sizes' section mentions that some storage technologies have different sized blocks for different operations, with an example of NAND flash.

Hardware | Documentation | Code | Questions | Forum | Compiler

Docs / mbed OS API References / Storage APIs

Edit this doc on GitHub

Block devices

► Introduction to the mbed OS API References

- Task management APIs
- Input and output APIs
- Digital interface APIs
- Communication APIs
- Security APIs
- Storage APIs
- API documentation
- Other sources

Block Devices

The block device API provides an interface for access to block-based storage. You can use a block device to back a full file system or write to it directly.

You can find the full C++ API [here](#).

Example

Here is an example that uses a block device:

[SD block device example](#)

Block device operations

A block device can perform three operations on a block in a device:

- Read a block from storage.
- Erase a block in storage.
- Program a block that has previously been erased.

Note: The state of an erased block is undefined. NOR flash devices typically set an erased block to all 0xff, but for some block devices such as the SD card, erase is a NOOP. If a deterministic value is required after an erase, the consumer of the block device must verify this.

Block sizes

Some storage technologies have different sized blocks for different operations. For example, NAND flash can be read and programmed in 256-byte pages, but must be erased in 16kibibyte sectors.

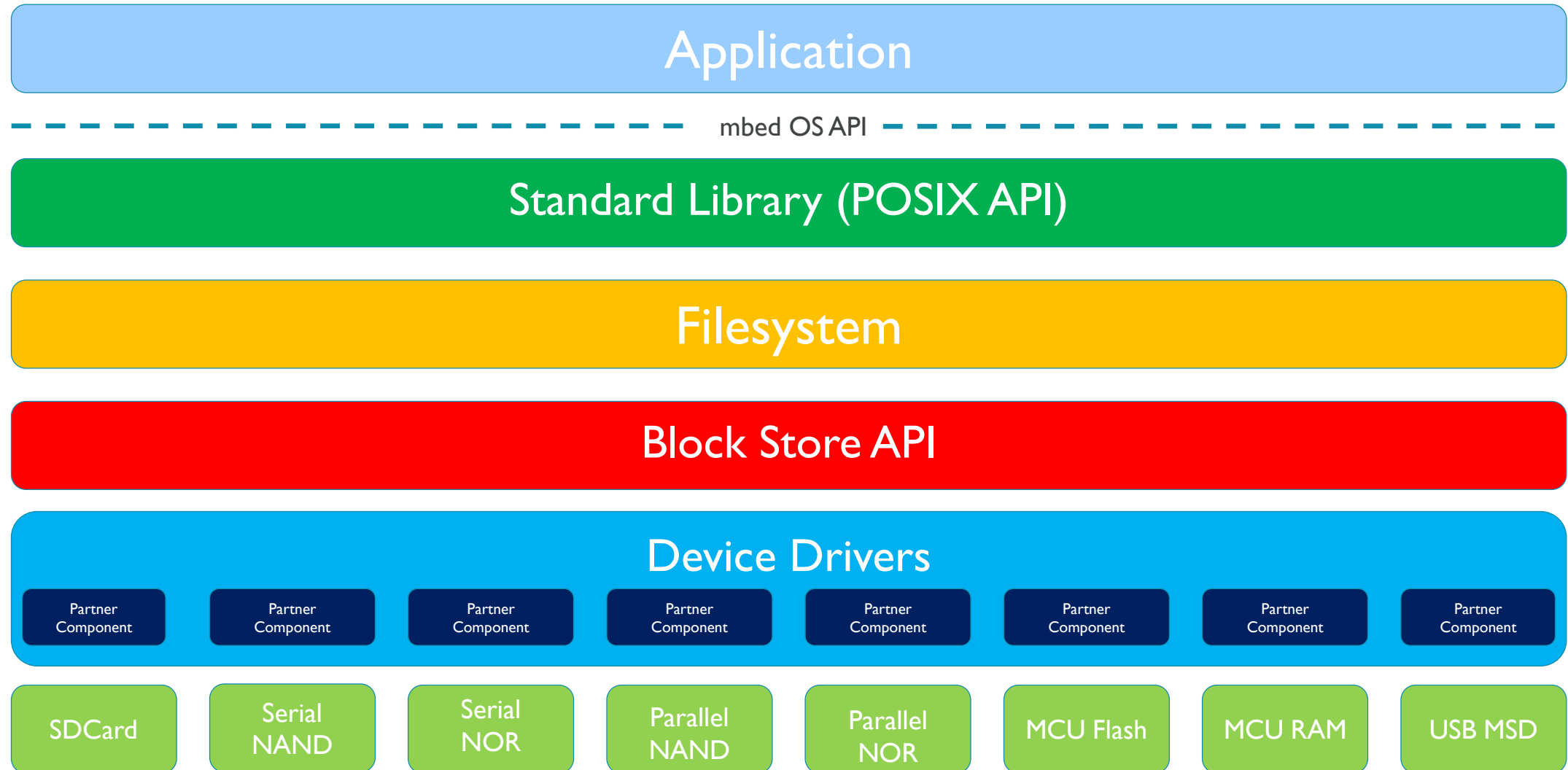
- https://docs.mbed.com/docs/mbed-os-api-reference/en/latest/APIs/storage/block_device/

Storage Technical Overview

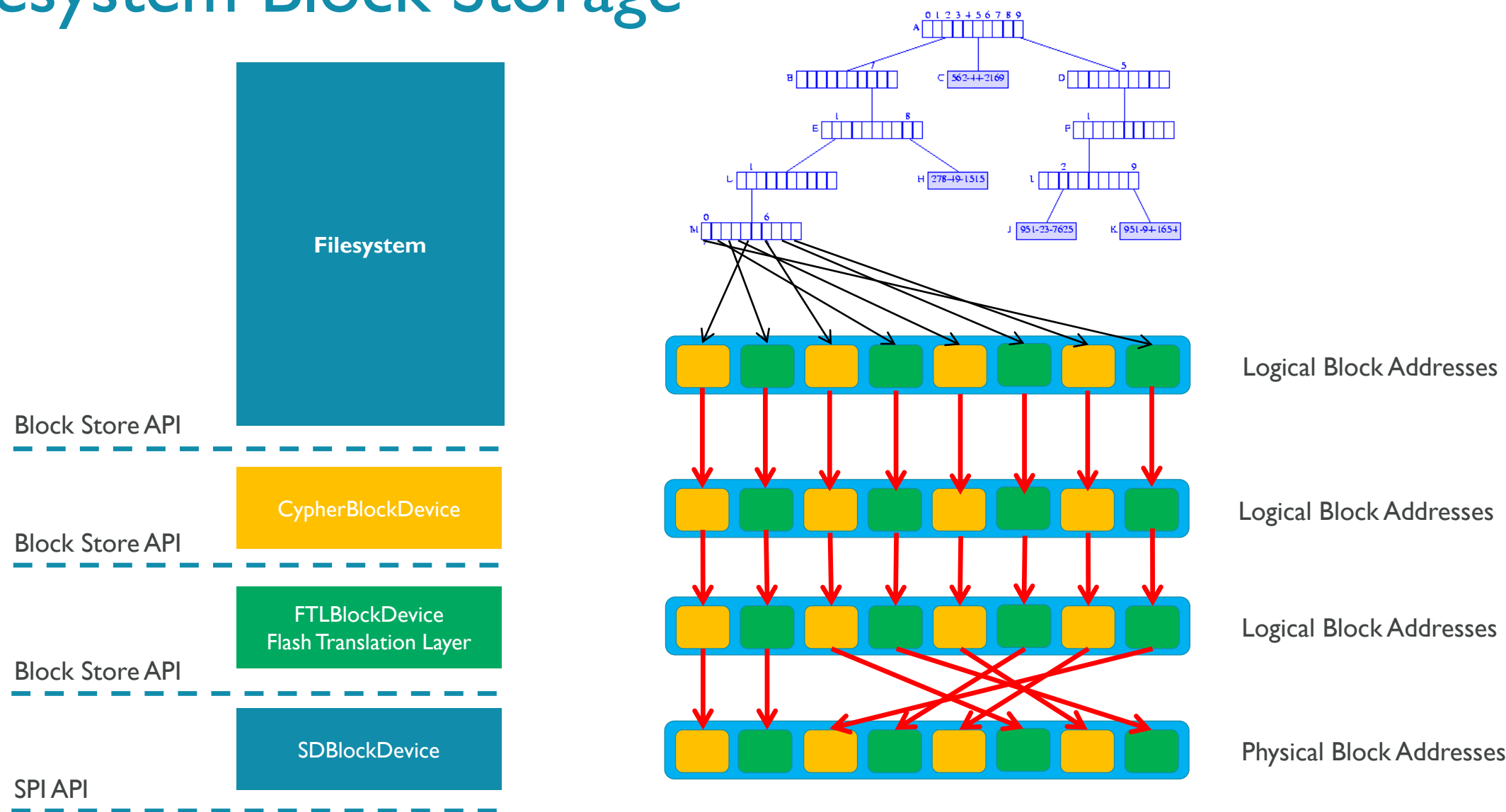
Storage: Overview

- Filesystem top down view:
 - Filesystem software stack.
 - Filesystem block storage and the rationale for the block device API.
- Flash Storage bottom-up view:
 - Memory Technology Devices (MTD) Characteristics.
 - Terminology (pages, sectors, blocks).
 - SPI NOR Flash.
- Block Device API.
- SDBlockDevice: the block device for SDCard.
 - How to use.
- SPIFBlockDevice: the block device for SPI NOR.
 - How to use.
- Filesystem Code size measurements.

File System Software Stack



Filesystem Block Storage



Memory Technology Devices (MTD)

Secure Digital (SD Card)

- Write/read semantic (no erase), page readable and writable.
- On-board MCU to manage NAND properties (bad blocks, ECC). FAT Filesystems often supported.
- Often connected via SPI protocol, or native SDCard protocol (supports 2 modes).

Serial/Parallel NOR FLASH

- Erase before write semantic. Byte readable, sometimes byte writable.
- Having lower cycle endurance than NAND (e.g. 10000-100000 write erase cycles).
- Being slower than NAND.

Serial/Parallel NAND Flash

- Erase before write semantic, being page readable and writable.
- Higher cycle endurance than NOR (e.g. 100000-1000000 write erase cycles).
- Faster than NAND. Requiring bad block management and ECC.

USB MSD (FLASH Disk)

- Used to plug in to devices (e.g. store files, dump logs, load firmware/settings).
- Virtually always runs FAT Filesystem to interoperate with PC.

MCU FLASH

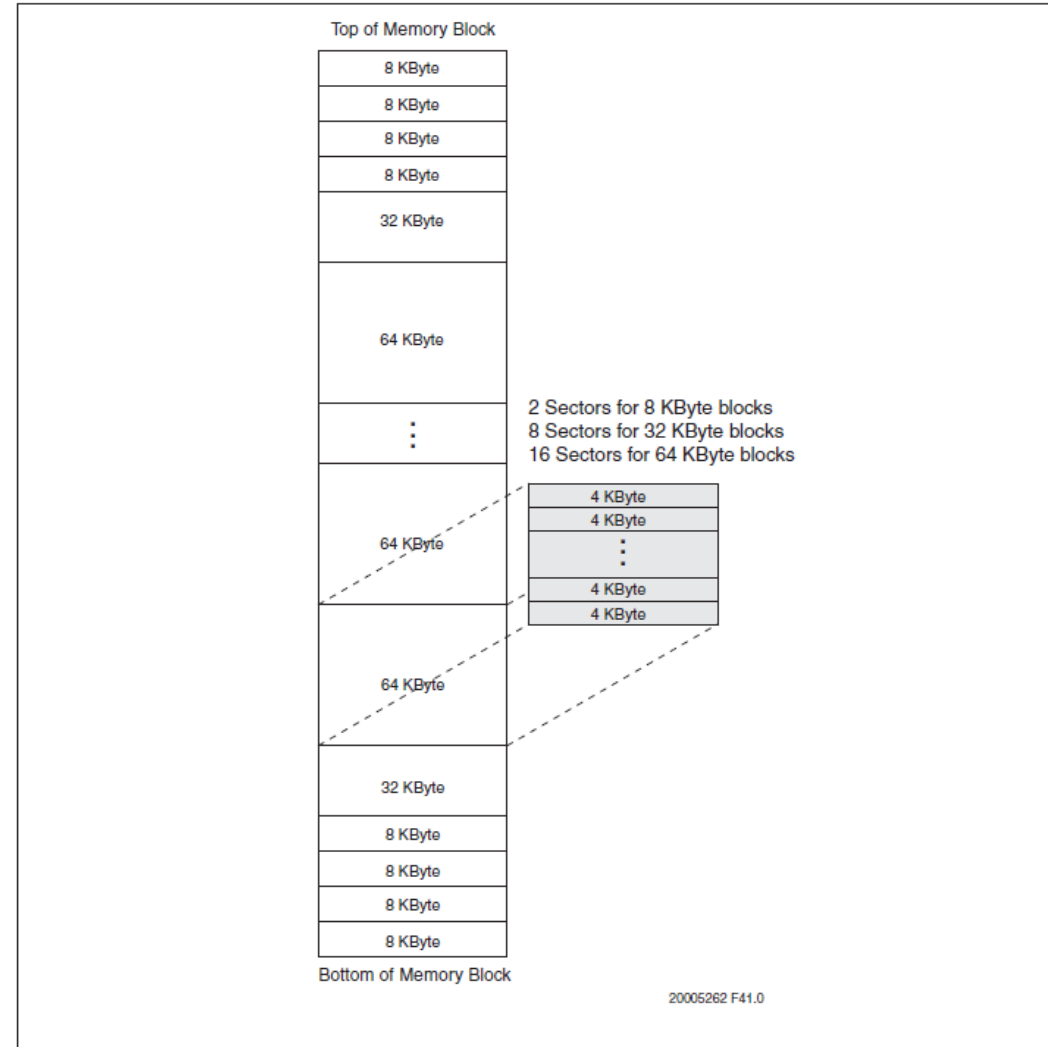
- Erase before write semantic, may have strictly sequential writing of sector after erase.
- Not typically byte writable e.g. K64F program unit e.g. 8bytes.

MCU RAM

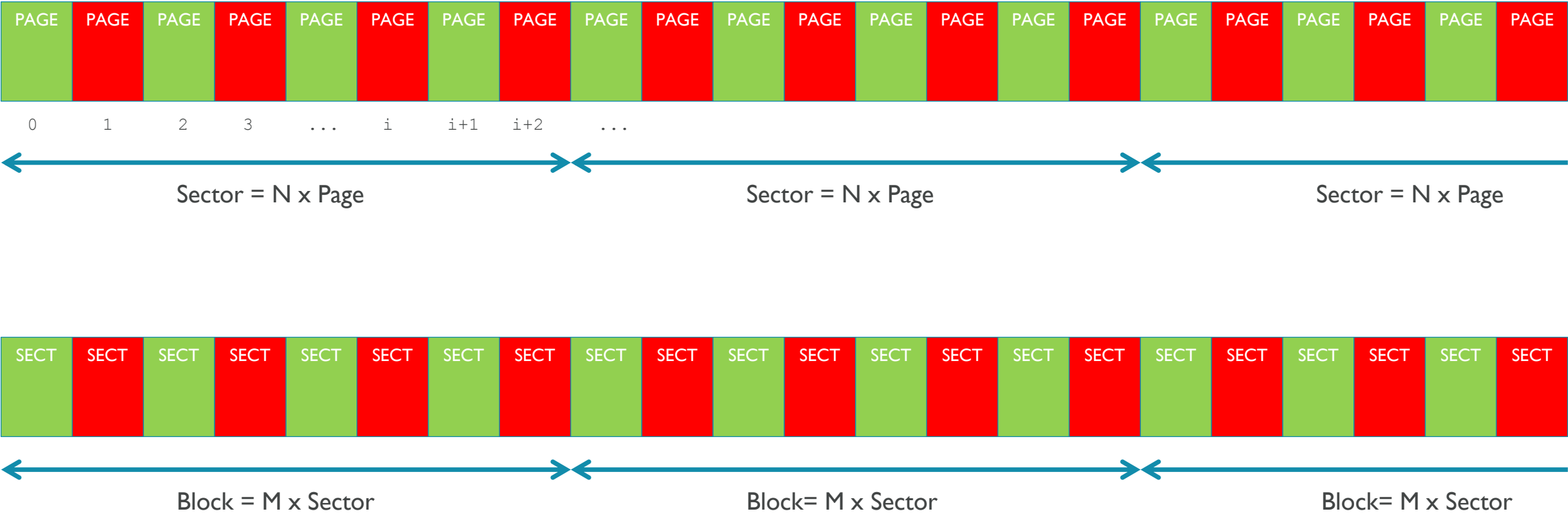
- Read/write semantic (no erase).
- Used for data during run lifetime or for debugging.
- With battery-backup, can be considered a non-volatile store.

SPI NOR Flash: Typical Geometry

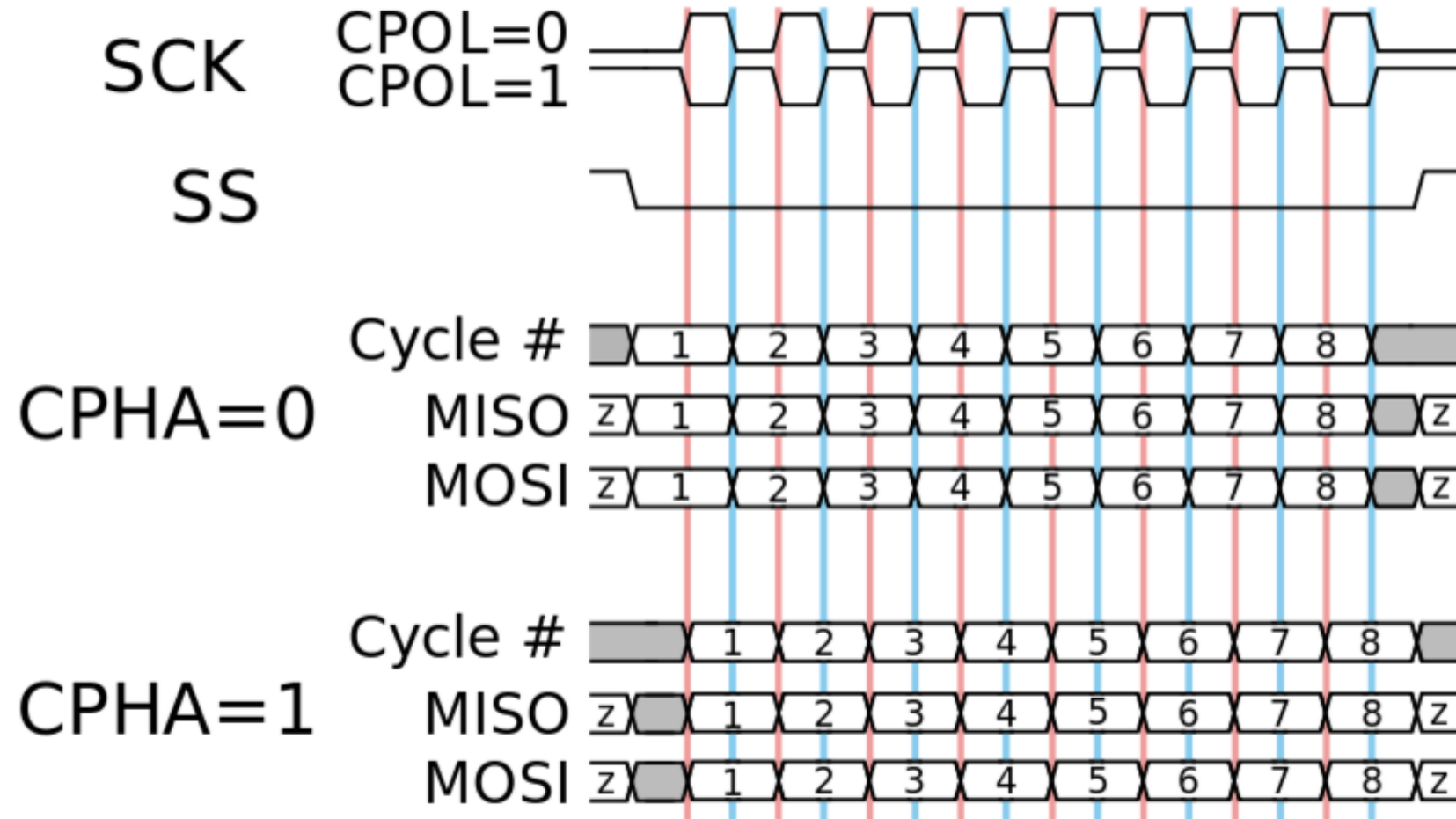
FIGURE 3-1: MEMORY MAP



Flash Terminology



SPI NOR Flash: SPI Bus Review



SPI NOR Flash Protocol: Read and Page Program

FIGURE 5-2: READ SEQUENCE (SPI)

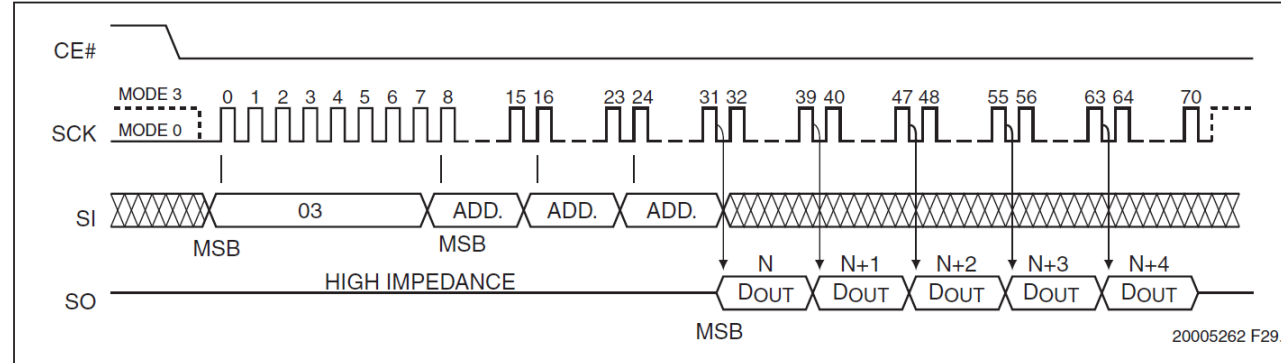
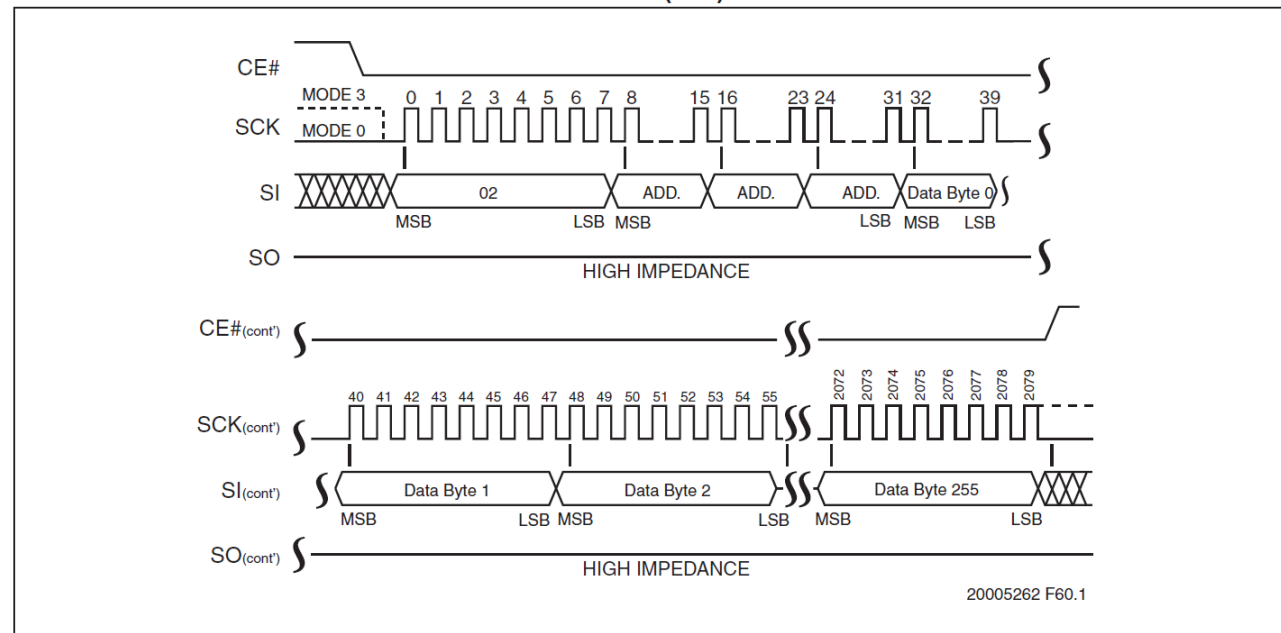


FIGURE 5-26: PAGE-PROGRAM SEQUENCE (SPI)



SPI NOR Flash: Protocol Standardisation

Command	Code (0x)	Description	Implied BlockDevice API
READ	03	Read 1-256 bytes	<code>read(block_store_id, read_buffer)</code>
PAGE PROGRAM	02	Program 1-256 bytes	<code>program(block_store_id, write_buffer)</code>
SECTOR ERASE	20	Reset sector storage to erase value (0xff)	<code>erase(block_store_id, flag=sector)</code> , block_store_id identifies the first block in sector
BLOCK ERASE	D8	Erase set of sectors forming blocks	<code>erase(block_store_id, flag=block)</code> where block_store_id identifies the first block in MT
CHIP ERASE	C7	Erase all storage	<code>erase(block_store_id, flag=chip)</code>
SFDP	5A	Read Serial Flash Discoverable Parameters	<code>get_info(params)</code>

Block Store API: class BlockDevice

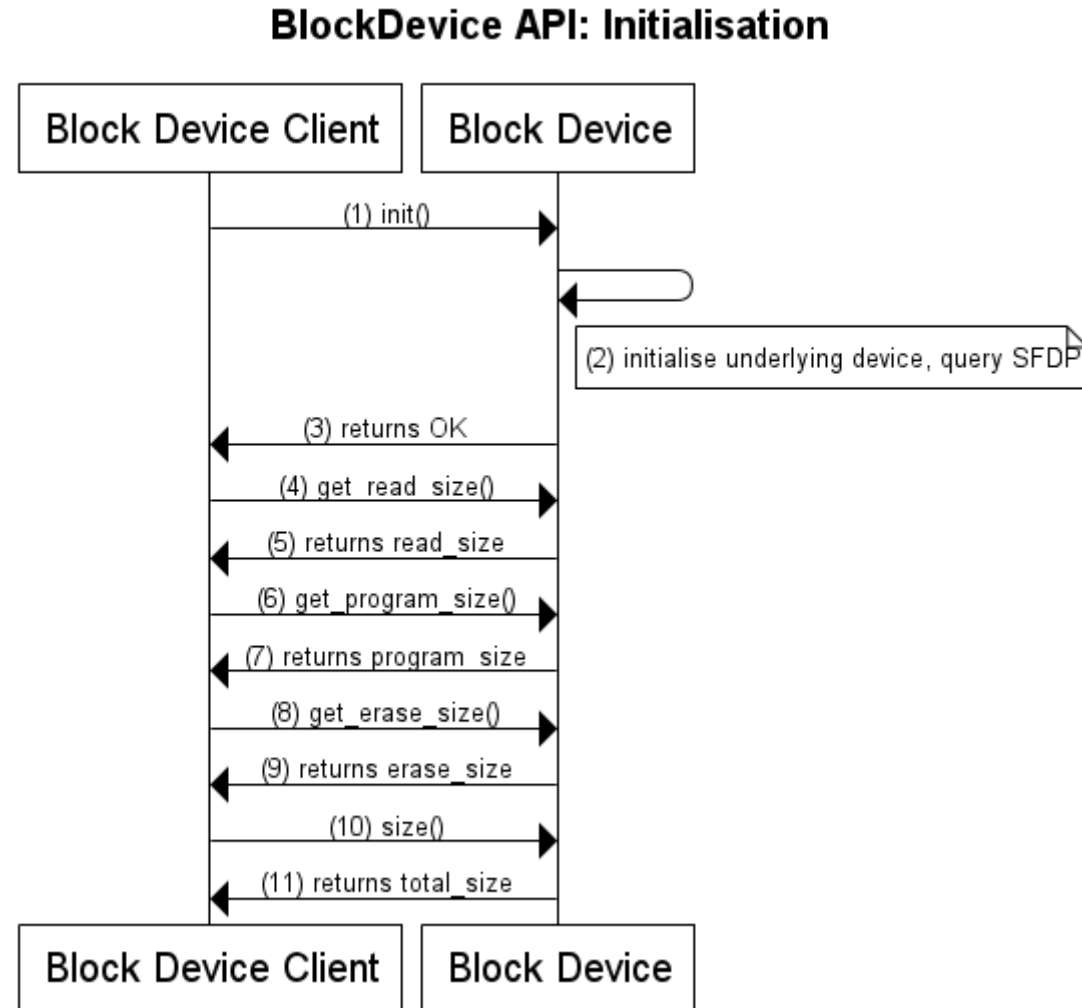
```
/** A hardware device capable of writing and reading blocks
 */
class BlockDevice
{
public:
    virtual ~BlockDevice() {};
    virtual int init() = 0;
    virtual int deinit() = 0;
    virtual int read(void *buffer, bd_addr_t addr, bd_size_t size) = 0;
    virtual int program(const void *buffer, bd_addr_t addr, bd_size_t size) = 0;
    virtual int erase(bd_addr_t addr, bd_size_t size) = 0;
    virtual bd_size_t get_read_size() const = 0;
    virtual bd_size_t get_program_size() const = 0;
    virtual bd_size_t get_erase_size() const = 0;
    virtual bd_size_t size() const = 0;
    bool is_valid_read(bd_addr_t addr, bd_size_t size) const;
    bool is_valid_program(bd_addr_t addr, bd_size_t size) const;
    bool is_valid_erase(bd_addr_t addr, bd_size_t size) const;
};
```

class BlockDevice: Initialisation and De-Initialisation

```
/** Initialize a block device
 *
 * @return      0 on success or a negative error code on failure
 */
virtual int init() = 0;

/** Deinitialize a block device
 *
 * @return      0 on success or a negative error code on failure
 */
virtual int deinit() = 0;
```

BlockDevice Initialisation Sequence Diagram



class BlockDevice::Informational Query Methods

```
/** Get the size of a readable block
 * @return      Size of a readable block in bytes */
virtual bd_size_t get_read_size() const = 0;

/** Get the size of a programmable block
 * @return      Size of a programmable block in bytes
 * @note Must be a multiple of the read size */
virtual bd_size_t get_program_size() const = 0;

/** Get the size of a erasable block
 * @return      Size of a erasable block in bytes
 * @note Must be a multiple of the program size */
virtual bd_size_t get_erase_size() const = 0;

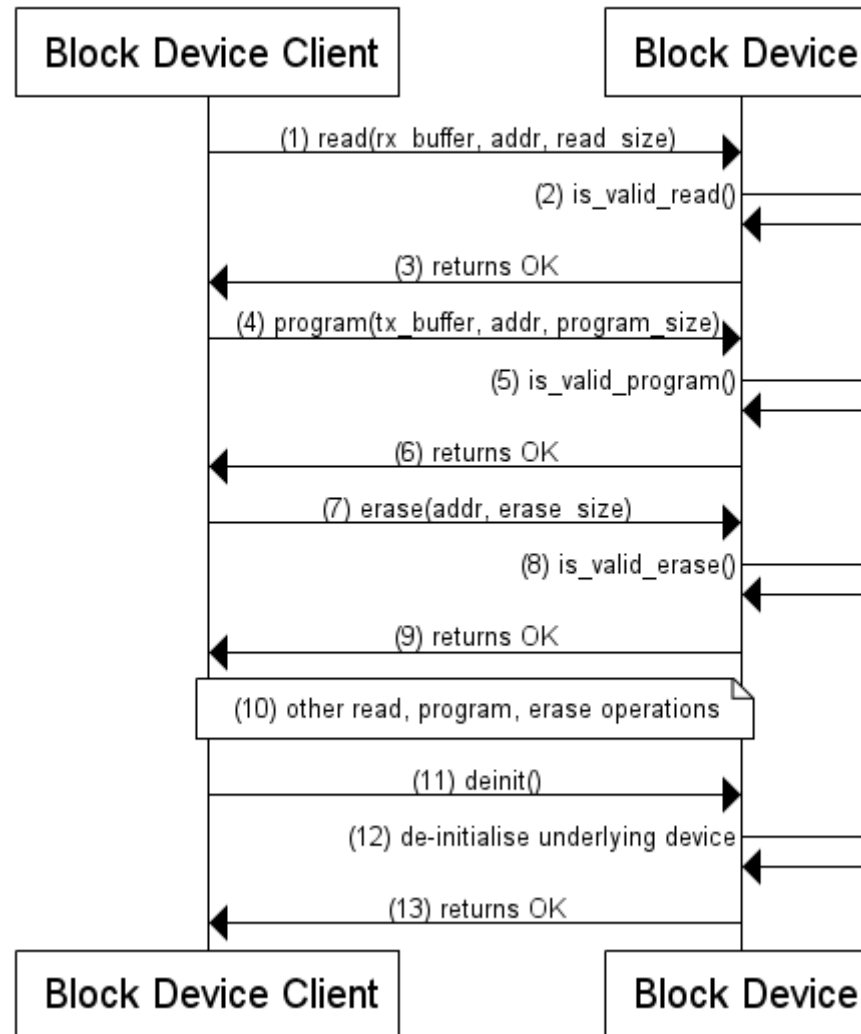
/** Get the total size of the underlying device
 * @return      Size of the underlying device in bytes */
virtual bd_size_t size() const = 0;
```


class BlockDevice::read() method

```
/** Read blocks from a block device
 *
 * If a failure occurs, it is not possible to determine how many bytes succeeded
 *
 * @param buffer    Buffer to write blocks to
 * @param addr      Address of block to begin reading from
 * @param size      Size to read in bytes, must be a multiple of read block size
 * @return          0 on success, negative error code on failure
 */
virtual int read(void *buffer, bd_addr_t addr, bd_size_t size) = 0;
```

BlockDevice Read, Program, Erase Sequence Diagram

BlockDevice API: read(), program(), erase()



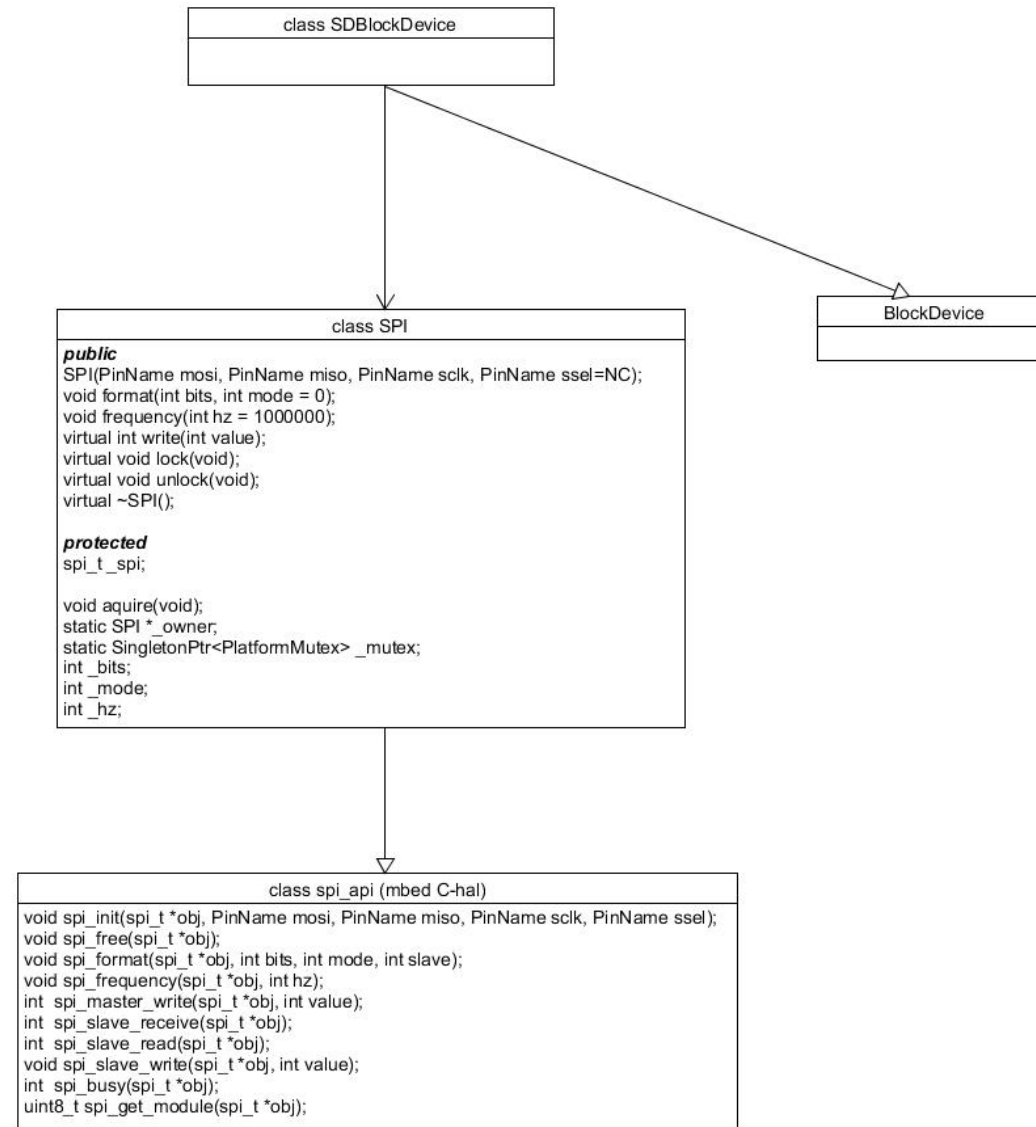
class BlockDevice::program() method

```
/** Program blocks to a block device
 *
 * The blocks must have been erased prior to being programmed
 *
 * If a failure occurs, it is not possible to determine how many bytes succeeded
 *
 * @param buffer    Buffer of data to write to blocks
 * @param addr      Address of block to begin writing to
 * @param size      Size to write in bytes, must be a multiple of program block size
 * @return          0 on success, negative error code on failure
 */
virtual int program(const void *buffer, bd_addr_t addr, bd_size_t size) = 0;
```

class BlockDevice::erase() method

```
/** Erase blocks on a block device
 *
 * The state of an erased block is undefined until it has been programmed
 *
 * @param addr      Address of block to begin erasing
 * @param size      Size to erase in bytes, must be a multiple of erase block size
 * @return          0 on success, negative error code on failure
 */
virtual int erase(bd_addr_t addr, bd_size_t size) = 0;
```

SDBlockDevice UML



SDBlockDevice

SDCard Reference Specification Part I Physical Layer Available for download: Chapter 7. SPI Mode

Available for download: https://members.sdcard.org/downloads/pls/simplified_specs/partI_410.pdf



**SD Specifications
Part 1
Physical Layer
Simplified Specification
Version 4.10
January 22, 2013**

SD Group

Panasonic Corporation
SanDisk Corporation
Toshiba Corporation

**Technical Committee
SD Card Association**

mbed OS File API Example: How To Use

```
#include "mbed.h"
#include "FATFileSystem.h"
#include "SDBlockDevice.h"
#include <stdio.h>
#include <errno.h>
/* mbed_retarget.h is included after errno.h so symbols are mapped to
 * consistent values for all toolchains */
#include "platform/mbed_retarget.h"

SDBlockDevice sd(MBED_CONF_APP_SPI_MOSI, MBED_CONF_APP_SPI_MISO,
MBED_CONF_APP_SPI_CLK, MBED_CONF_APP_SPI_CS);
FATFileSystem fs("sd", &sd);
void return_error(int ret_val);
void errno_error(void* ret_val);

int main()
{
    int error = 0;
    printf("Welcome to the filesystem example.\n");

    printf("Opening a new file, numbers.txt.");
    FILE* fd = fopen("/sd/numbers.txt", "w+");
    errno_error(fd);

    for (int i = 0; i < 20; i++){
        printf("writing decimal numbers to a file (%d/20)\r", i);
        fprintf(fd, "%d\n", i);
    }
}
```

```
printf("writing decimal numbers to a file (20/20) done.\n");
printf("Closing file.");
fclose(fd);
printf(" done.\n");

printf("Re-opening file read-only.");
fd = fopen("/sd/numbers.txt", "r");
errno_error(fd);

printf("Dumping file to screen.\n");
char buff[16] = {0};
while (!feof(fd)){
    int size = fread(&buff[0], 1, 15, fd);
    fwrite(&buff[0], 1, size, stdout);
}
printf("EOF.\n");

printf("Closing file.");
fclose(fd);
printf(" done.\n");

while (true) {}
}

// return_error() and errno_error() not shown to conserve space
```

SDBlockDevice: How To Use

```
#include "mbed.h"
#include "SDBlockDevice.h"
#include <stdio.h>

SDBlockDevice sd(MBED_CONF_APP_SPI_MOSI, MBED_CONF_APP_SPI_MISO, MBED_CONF_APP_SPI_CLK, MBED_CONF_APP_SPI_CS);

int main() {
    printf("sd test\n");

    // Initialize the SDCard device and print the memory layout
    sd.init();
    printf("sd size: %llu\n",      sd.size());
    printf("sd read size: %llu\n",  sd.get_read_size());
    printf("sd program size: %llu\n", sd.get_program_size());
    printf("sd erase size: %llu\n",  sd.get_erase_size());

    // Write "Hello world!" to the first block
    char *buffer = (char*) malloc(sd.get_erase_size());
    sprintf(buffer, "Hello world!\n");
    sd.program(buffer, 0, sd.get_erase_size());

    // Read back what was stored
    sd.read(buffer, 0, sd.get_erase_size());
    printf("%s", buffer);

    // Deinitialize the device
    sd.deinit();
}
```


SPIFBlockDevice: How To Use

```
#include "mbed.h"
#include "SPIFBlockDevice.h"

SPIFBlockDevice spif(D11, D12, D13, D10);

int main() {
    printf("spif test\n");

    // Initialize the SPI flash device and print the memory layout
    spif.init();
    printf("spif size: %llu\n",      spif.size());
    printf("spif read size: %llu\n",  spif.get_read_size());
    printf("spif program size: %llu\n", spif.get_program_size());
    printf("spif erase size: %llu\n",  spif.get_erase_size());

    // Write "Hello world!" to the first block
    char *buffer = (char*) malloc(spif.get_erase_size());
    sprintf(buffer, "Hello world!\n");
    spif.erase(0, spif.get_erase_size());
    spif.program(buffer, 0, spif.get_erase_size());

    // Read back what was stored
    spif.read(buffer, 0, spif.get_erase_size());
    printf("%s", buffer);

    // Deinitialize the device
    spif.deinit();
}
```

FS Code Sizes: measured using GCC_ARM

```
Keil Embedded FS Summary Results
=====
Total size of Keil EFS .text symbols:      6396
Total size of KeilFS wrappers .text symbols:  732

Keil FATFS Summary Results
=====
Total size of Keil FATFS .text symbols:      13836
Total size of Keil FATFS .rodata symbols:      1341
Total size of KeilFS wrappers .text symbols:    741

ChanFS Summary Results
=====
Total size of ChanFS .text symbols:           8960
Total size of sdcard .text symbols:           4830
Total size of wrappers .text symbols:         8479
Total size of rodata:                         1180
Total size of BSS:                           3227
Total size of .text(chanfs+sdcard+wrappers)+.rodata: 23449
```

ChanFS FAT32 RAM requirement ~1.5kB (1kB + size of 1 SDBlockDevice Erase Block (0.5kB)).

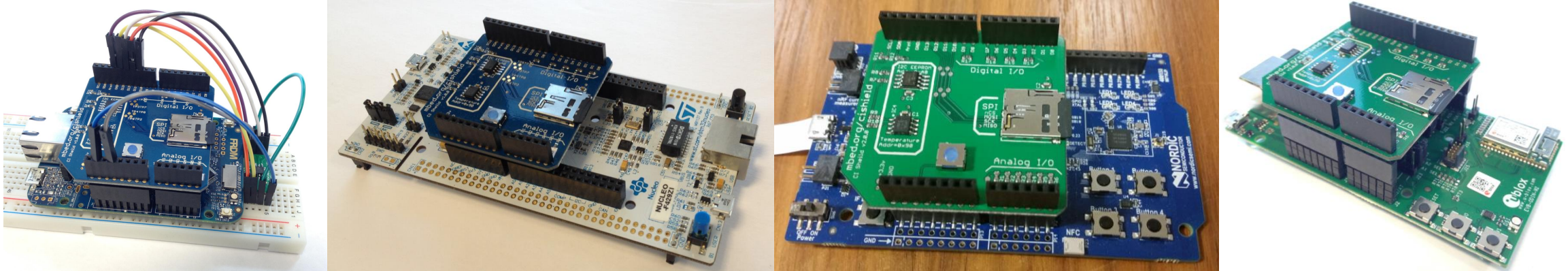
Storage: Conclusion

- Filesystem software stack.
 - Rationale for the block device API.
- Memory Technology Devices (MTD)
 - SPI NOR parts.
- Block Device API.
- How to use the SDBlockDevice, the block device for SDCard
- How to use the SPIFBlockDevice, the block device for SPI NOR.
- Resource Usage Measurements.

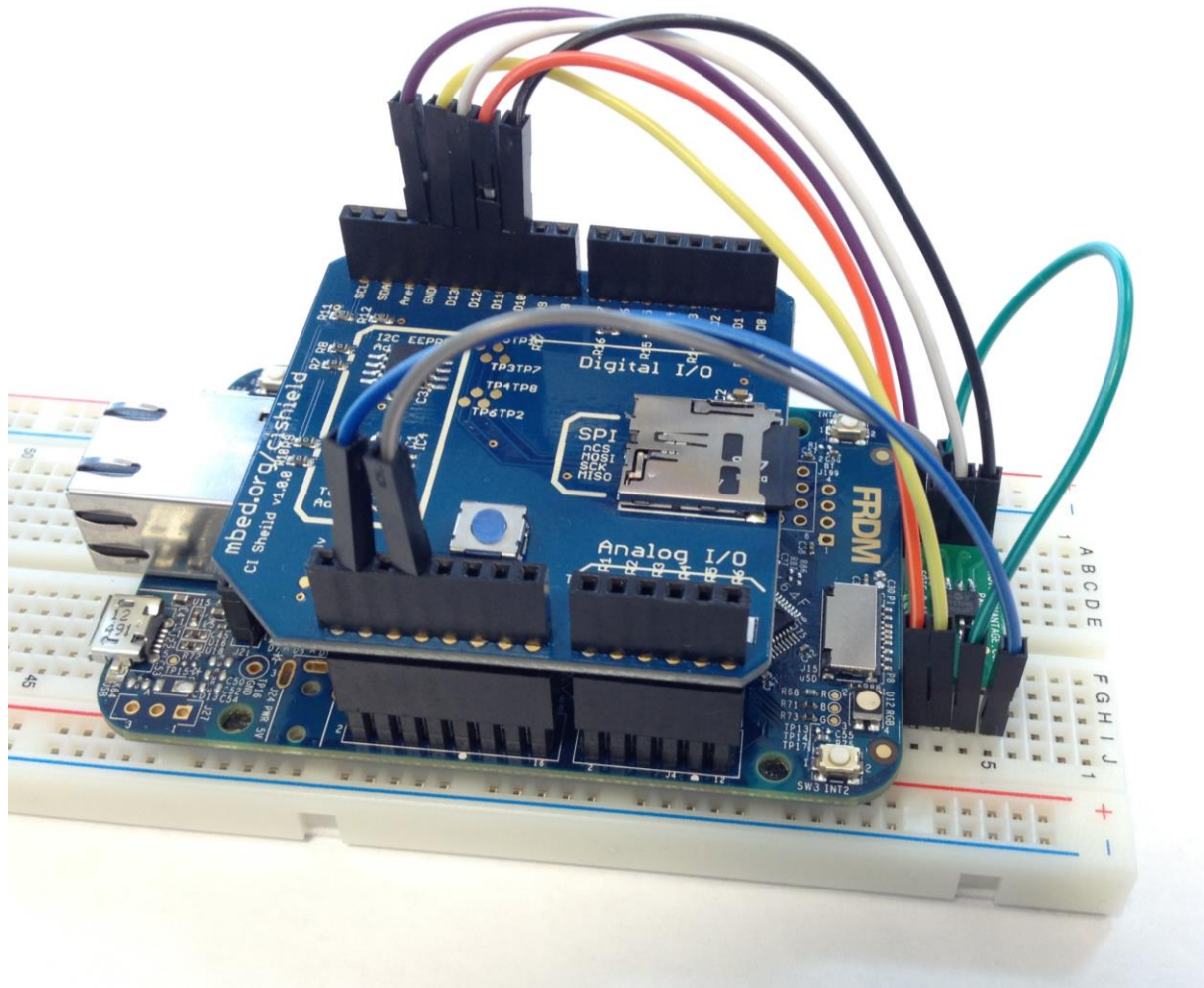
Workshop

Workshop: Overview

- Test Fixtures used as the starting point for examples.
 - Useful Resources.
 - Format of examples.
- Example 1: SDCard Application (class SDBlockDevice) .
- Example 2: SPI NOR Storage (class SPIFBlockDevice).
- Example 3: I2C EEPROM BlockDevice (class I2CEEBlockDevice).



Workshop Test Fixture



We'll use the a test fixture to get the examples up and running, and then replace the devboard with a board of your choice.

Test Fixtures is composed of:

- NXP K64F, or STM F429ZI, or Ublox Odin
- CI Test Shield with SDCard.
- Microchip SST26VF016B 512kB SPI NOR Flash.
- Jumpers from CI Shield Arduino header SPI pins to SPI NOR flash.

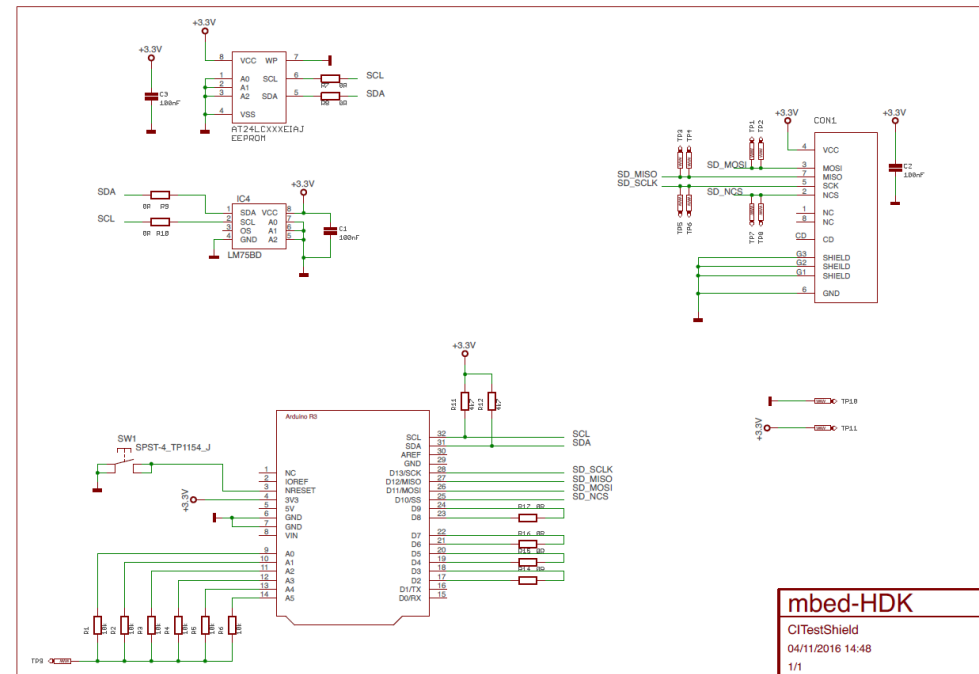
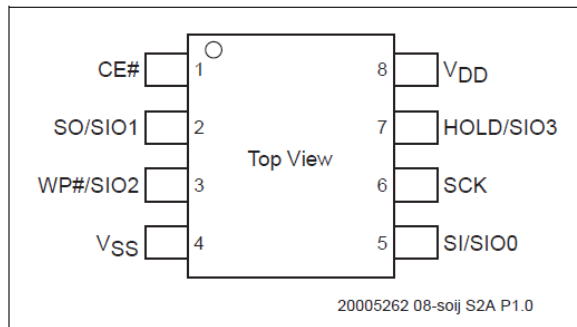
Workshop Test Fixture Resources

- The CI shield wiring diagram: https://github.com/ARMmbed/mbed-HDK/blob/master/Production%20Design%20Projects/CI%20TestShield/v2.0.0/CI%20TestShield%20V_2_0_0%20SCH.pdf
- The Microchip SST26VF016B SPI NOR datasheet: <http://www.microchip.com/downloads/en/DeviceDoc/20005262C.pdf>

SST26VF016B

2.0 PIN DESCRIPTION

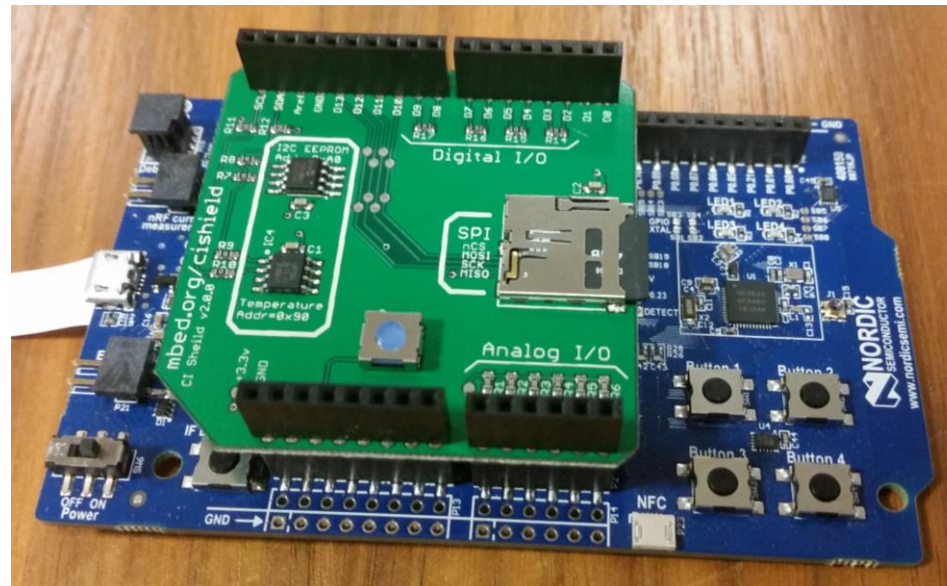
FIGURE 2-1: PIN DESCRIPTION FOR 8-LEAD SOIJ



Example Format:

The examples follow a general format:

- Part 1: Simple application demonstrating key feature using Test Fixtures.
- Part 2: mbed Greentea test cases demonstrating greater complexity examples on Test Fixtures.
- Part 3: Demonstrating Parts 1 and 2 on Your Target.
- Part 4: Suggested exercises for further work.



Example 1: SDCard Application (SDBlockDevice)

This example demonstrates:

- How to use the Test Fixture.
- How to use the POSIX File API and FAT32 Filesystem to read/write data to SDCard.
- How to use the mbed_app.json configuration file to build and test on Your Target.

Complete the example using the following steps:

1. Go to the mbed-os-example-sd-driver repository:
<https://github.com/ARMmbed/mbed-os-example-sd-driver>.
2. In the repository README.md, follow the instructions in the section: “**Getting Started With The sd-driver**” as shown. This will explain how to build and run the example application on the test fixture. The application demonstrates how to use the POSIX API to store data on the SDCard.

Getting Started With The sd-driver

This is the mbed-os example for the SDBlockDevice (SDCard) block device driver. See the [sd-driver](#) repository for more information.

This guide reviews the steps to get the SDCard with FAT filesystem working on an mbed OS platform.

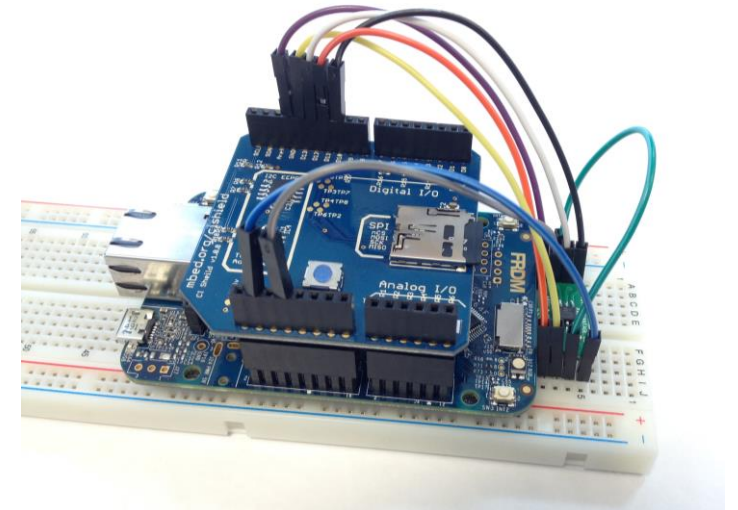
Please install [mbed CLI](#).

Hardware Requirements

This example can be used on an mbedos platform that:

- Has an on-board SDCard slot or,
- Is fitted with a [CI Test Shield](#).

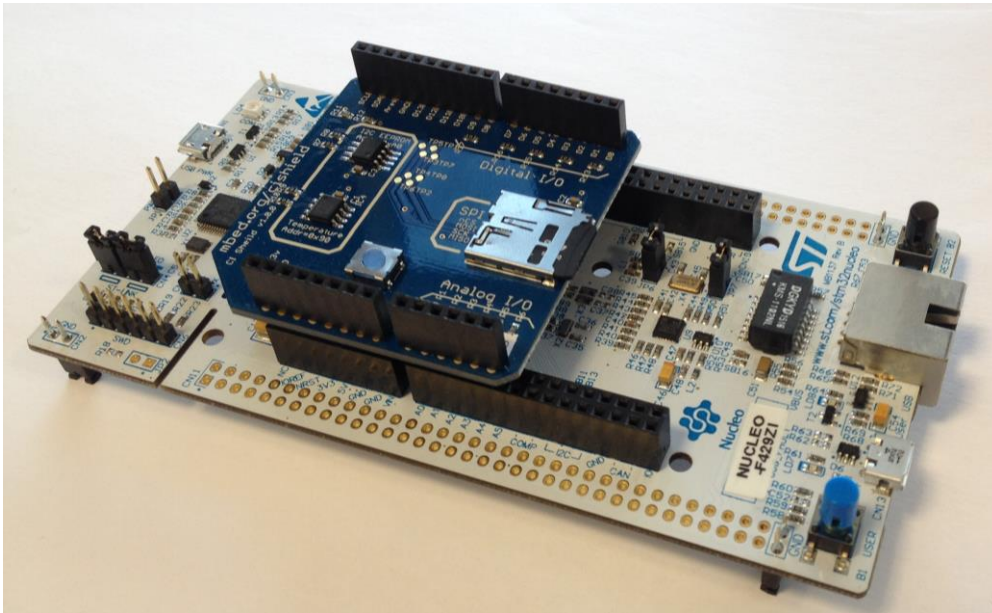
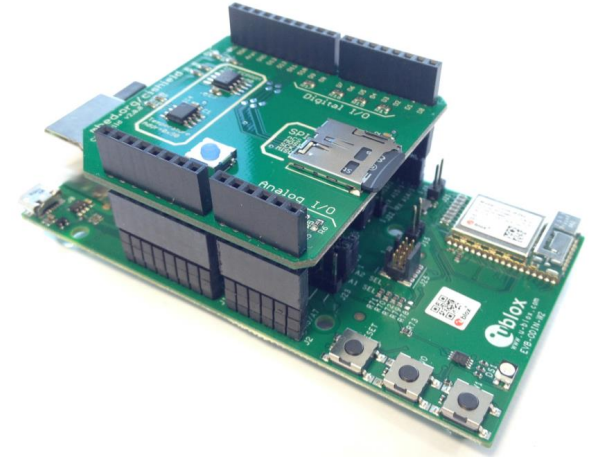
This document uses the K64F as an example. Simply change the relevant options (e.g. -m K64F) to be appropriate for your target.



Example 1: Rebuild for Your Target

Complete the example using the following steps

1. Go to the mbed-os-example-sd-driver repository: <https://github.com/ARMmbed/mbed-os-example-sd-driver>.
2. In the repository README.md, follow the instructions in the section: “**Testing with an SDCard on Target XYZ**” as shown below. This will explain the mbed_app.json changes required (if any) so the example will work for Your Target.
3. Gently remove the shield from the Test Fixture and insert into the Arduino headers of your target, or use another CI test shield.
4. If it doesn’t work.... time to start debugging.



Testing with an SDCard on Target XYZ

The standard way to test is with the mbed CI Test Shield plugged into the target board. This pin mapping for this configuration is parameterised in the `mbed_app.json` file.

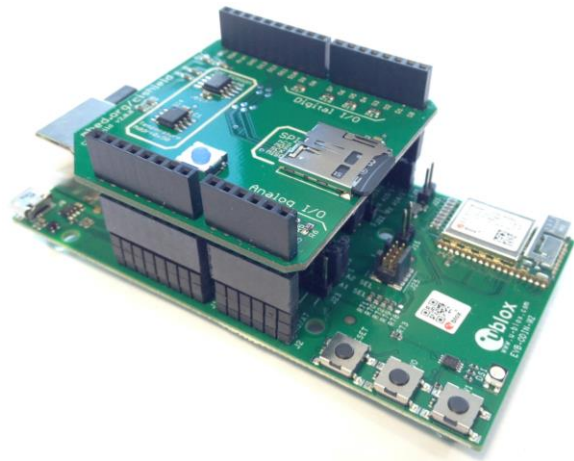
The following is an example of the `mbed_app.json` file available in the repository:

```
{
  "config": {
    "UART_RX": "D0",
    "UART_TX": "D1",
    "DIO_0": "D0",
    "DIO_1": "D1",
    "DIO_2": "D2",
    "DIO_3": "D3",
    "DIO_4": "D4",
    "DIO_5": "D5",
    "DIO_6": "D6",
    "DIO_7": "D7",
    "DIO_8": "D8",
    "DIO_9": "D9",
    "SPI_CS": "D10",
    "SPI_MOSI": "D11",
    "SPI_MISO": "D12",
    "SPI_CLK": "D13",
    "I2C_SDA": "D14",
    "I2C_SCL": "D15",
    "I2C_TEMP_ADDR": "0x90",
  }
}
```

Example I: Run Test Cases on Your Target

Run the test cases by completing the following steps:

1. Go to the sd-driver repo: <https://github.com/armmbed/sd-driver>.
2. In the repository README.md, follow the instructions in the section: “**SDCard POSIX File API mbed Greentea Test Cases**” as shown below. This explains how to build and run the test cases for the K64F, but you should build and run on Your Target.



SDCard POSIX File API mbed Greentea Test Cases

This section describes how to build and run the POSIX file API test cases. The following steps are covered:

- [Create the FAT/SDCard Application Project](#). This section describes how to git clone the mbed OS and sd-driver repositories containing the code and test cases of interest.
- [Build the mbed OS Test Cases](#). This section describes how to build the mbed OS test cases.
- [Insert a microSD Card Into the K64F for Greentea Testing](#). This section describes how to format (if required) a microSD card prior to running the tests.
- [Run the POSIX File Test Case](#). This section describes how to run the POSIX file test cases.

Create the FAT/SDCard Application Project

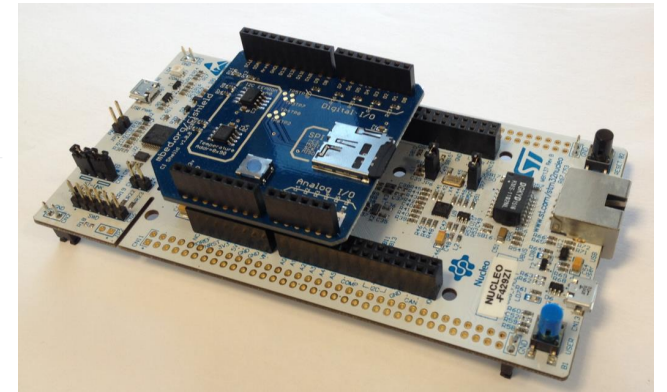
This section describes how to create an application project combining the mbed-os and sd-driver repositories into a single project. In summary the following steps will be covered in this section:

- A top level application project directory is created. The directory name is ex_app1.
- In the ex_app1 directory, the mbed-os repository is cloned.
- In the ex_app1 directory at the same level as the mbed-os directory, the sd-driver repository is cloned.
- The mbed_app.json file is copied from the sd-driver/config/mbed_app.json to the ex_app1 directory.

First create the top level application directory ex_app1 and move into it:

```
simhug01@E107851:/d/demo_area$ mkdir ex_app1
simhug01@E107851:/d/demo_area$ pushd ex_app1
```

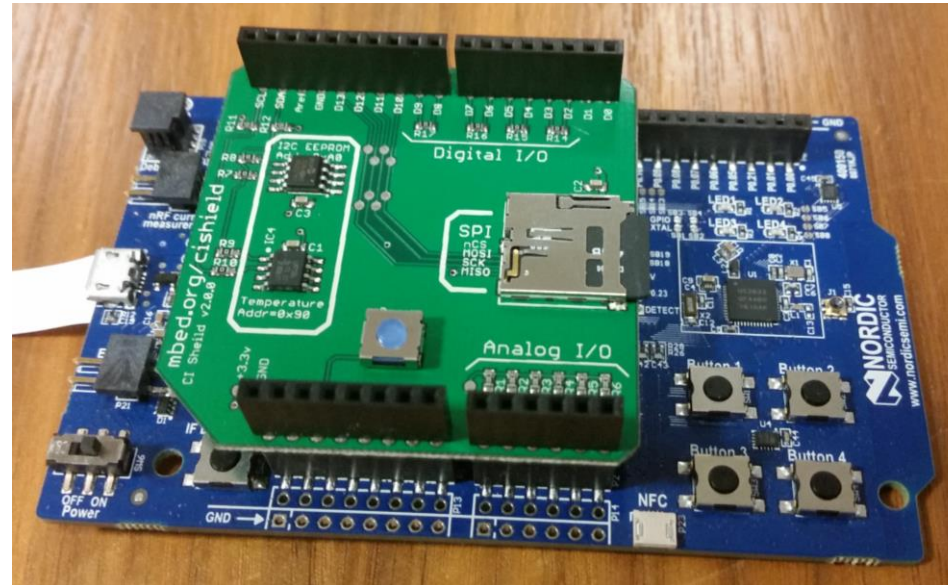
Next, get a clone of public mbed OS repository in the following way:



Example 1: Suggested Exercises

The following are possible exercises for further understanding the SDBlockDevice:

1. If your SoC has an SD IP Block then you could implement a BlockDevice driver to take advantage of the native support.
2. If you have a Quad SPI implementation, you could implement the HAL and wrapper to integrate within mbed OS.
3. Use mbedTLS and the file API to implement file level encryption for files stored on SDCard.



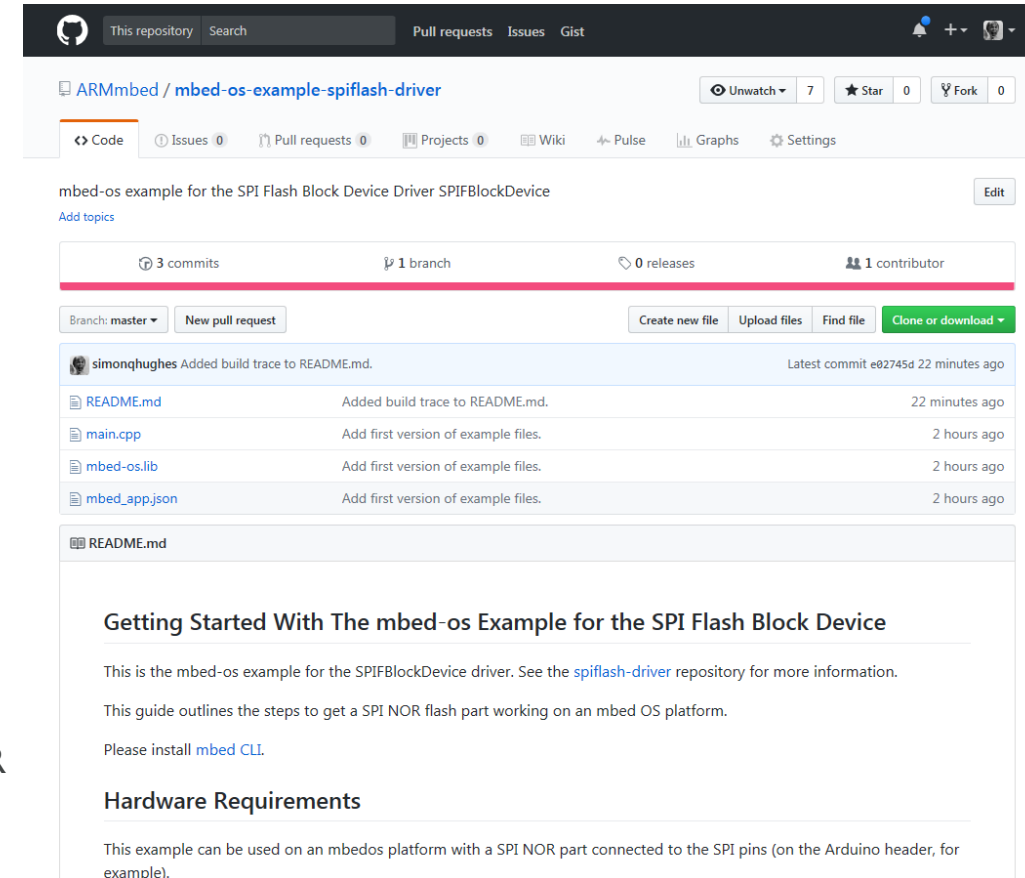
Example 2: SPI NOR Storage Example

This example demonstrates:

- Use of spiflash-driver for reading/writing data using the SPIFlashDevice.
- Use of SPI NOR Flash (Microchip SST26VF016B).
- How to build and run a simple test application.

Complete the example using the following steps:

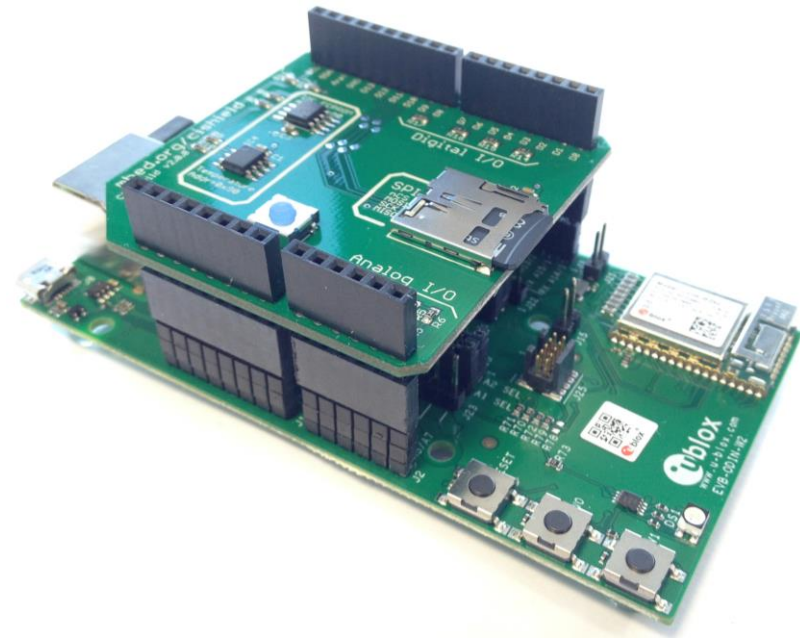
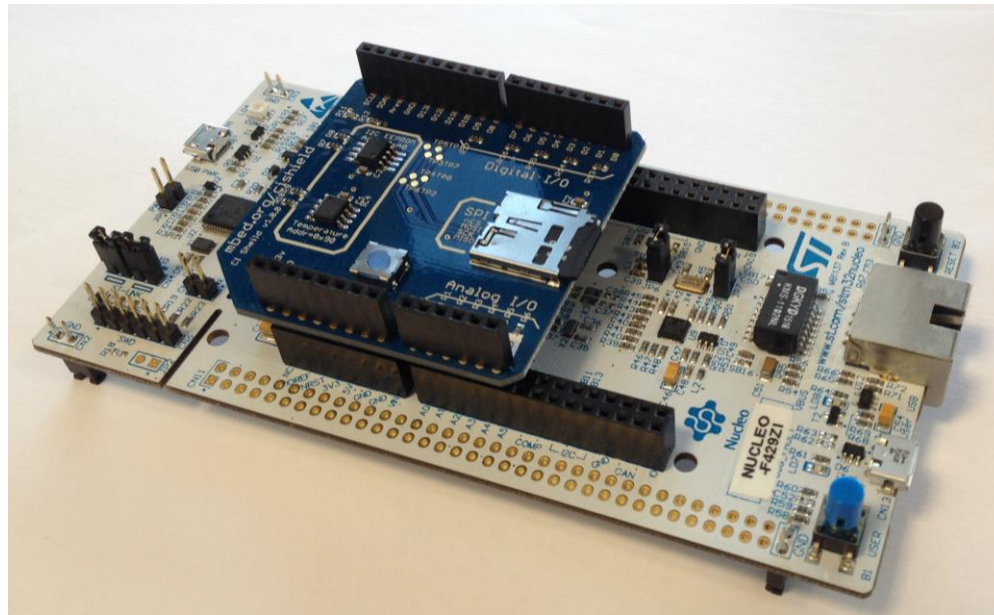
1. Go to the sd-driver repo: <https://github.com/armmbed/mbed-os-example-spif-driver>.
2. In the repository README.md, follow the instructions in the section: “**Getting Started with the mbed-os Example for the SPI Flash Block Device**” as shown. This will explain how to build and run the example application on the test fixture. The application demonstrates how use the SPIFlashDevice class to read and write data to the SPI NOR device.



Example 2: Rebuild for Your Target

Complete the example using the following steps

1. Gently remove the shield from the Test Fixture and insert into the Arduino headers of your target. Rewire the SPI NOR flash if the SPI bus is not presented on the D10-D13 header pins.
2. Repeat the steps described in the first part of this example but this time build and test for your target.
3. If it doesn't work.... time to start debugging.



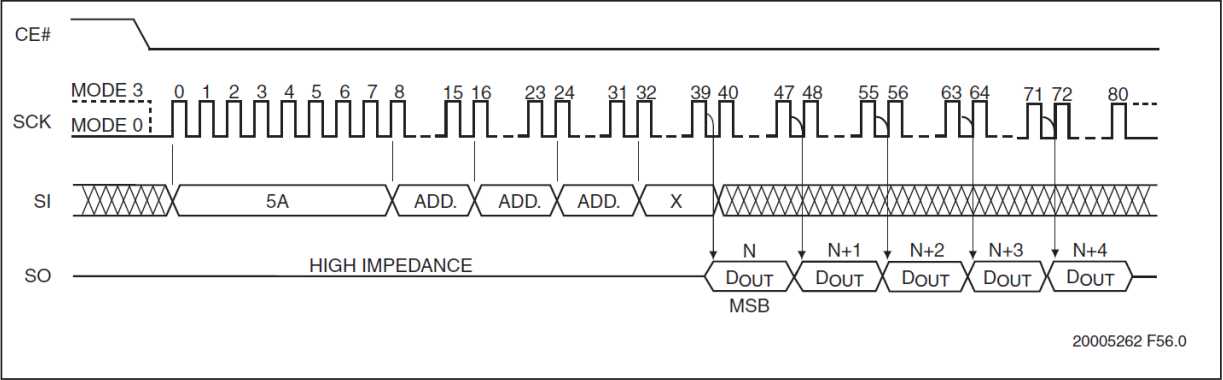
Example 2: Suggested Exercises

The following are possible exercises for further understanding the SPIFlashBlockDevice:

1. If your target has a native SPI flash part then you could implement a BlockDevice driver to integrated within mbedOS.
2. Build and run the examples for the NXP K82F which has the Macronix MX25U3235FZNI SPI NOR flash on-board (1 development board available).
3. Create a tool or test case to read and print out the Serial Flash Discoverable Parameters (SFDP) header and JEDEC Table parameters from a SPI NOR Flash. Decode one or more of the descriptors of interest. This will be a useful diagnostic tool.

Example2: Serial Flash Discoverable Parameters

FIGURE 5-18: SERIAL FLASH DISCOVERABLE PARAMETERS SEQUENCE



6.1 SFDP Overall Header Structure

	[31:24]	[23:16]	[15:8]	[7:0]	Hex Byte Location
SFDP Header	Serial Flash Discoverable Parameters (SFDP) Signature= 0x50444653				◀ [3:0]
	Byte 3 = "P"	Byte 2 = "D"	Byte 1 = "F"	Byte 0 = "S"	
1 st Parameter Header	Unused (set to 0xFF)	Number of Parameter Headers (NPH)	SFDP Major Revision	SFDP Minor Revision	◀ [7:4]
	Parameter Length (in double words)	Parameter Major Revision	Parameter Minor Revision	JEDEC ID (0x00)	◀ [B:8]
	Unused (set to 0xFF)	Parameter Table Pointer (PTP)			◀ [F:C]
2 nd Parameter Header (optional)	Parameter Length (in double words)	Parameter Major Revision	Parameter Minor Revision	JEDEC ID (0x00) or Manufacturer ID	◀ [13:10]
	Unused (set to 0xFF)	Parameter Table Pointer (PTP)			◀ [17:14]
⋮					
N th Parameter Header (optional)	Parameter Length (in double words)	Parameter Major Revision	Parameter Minor Revision	JEDEC ID (0x00) or Manufacturer ID	
	Unused (set to 0xFF)	Parameter Table Pointer (PTP)			

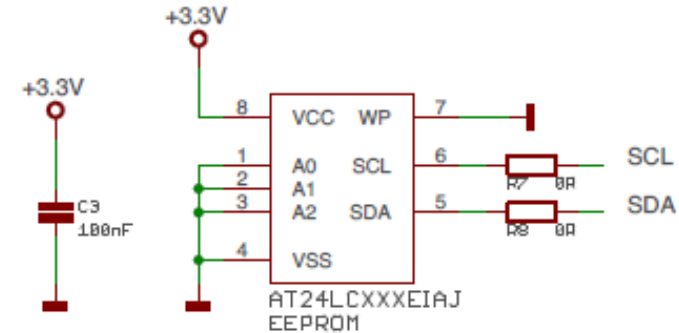
Figure 4 — Overall Header Structure

JEDEC Standard JESD216: Serial Discoverable Parameters (SFDP) for SPI NOR Flash (JESD216.pdf download from <http://www.jedec.org/>)

Example 3: I2CEEPROM BlockDevice Example

This example demonstrates:

- Use of i2ceeprom-driver for reading/writing data using the I2CEEBlockDevice.
- CI Test Shield I2C EEPROM (AT24LC).
- How to build and run a simple test application.



Complete the example using the following steps:

1. Go to the sd-driver repo: <https://github.com/armmbed/mbed-os-example-i2ceeprom-driver>.
2. In the repository README.md, follow the instructions in the section: “**Getting Started With The mbed-os Example for the I2C EEPROM Block Device**” as shown. This will explain how to build and run the example application on the test fixture. The application demonstrates how use the I2CEEBlockDevice class to read and write data to the EEPROM.
3. Gently remove the CI test shield from the Test Fixture and insert into the Arduino headers of your target, or use another shield. Build and run the example on your target.
4. If it doesn't work.... time to start debugging.

The screenshot shows the GitHub repository page for `ARMmbed / mbed-os-example-i2ceeprom-driver`. The repository has 8 stars and 0 forks. The main content area shows the commit history, with the latest commit by `simonhughes` removing `i2ceeprom-driver.lib` 34 minutes ago. The commit list includes `README.md`, `main.cpp`, `mbed-os.lib`, and `mbed_app.json`. The `README.md` file is expanded, showing the section "Getting Started With The mbed-os Example for the I2C EEPROM Block Device". The text in the README states: "This is the mbed-os example for the I2CEEBlockDevice driver. See the [i2ceeprom-driver](#) repository for more information. This guide outlines the steps to get the I2C EEPROM part working on an mbed OS platform fitted with the CI Test Shield. Please install [mbed CLI](#)." The "Hardware Requirements" section is also visible.

Stretch Goal:

- Apply an “over the air update” firmware image to become the new bootable image.

Workshop: Conclusion

- Well Done, have a drink.

Thank you!

ARM

The trademarks featured in this presentation are registered and/or unregistered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

Copyright © 2017 ARM Limited

Confidential ©ARM 2017