Document name: ECSD Getting started guide	Version 2.0.0
Internal reference: Products/ECSD/Getting started guide/3548	

ECSD Getting started guide

Version 2.0.0

Table of contents

1 Introduction

The EmbCode SD card driver (EcSD) is a driver to read and write Secure Digital (SD) or MultiMedia Cards (MMC).

This document is intended for developers that are about to start using EcSD.

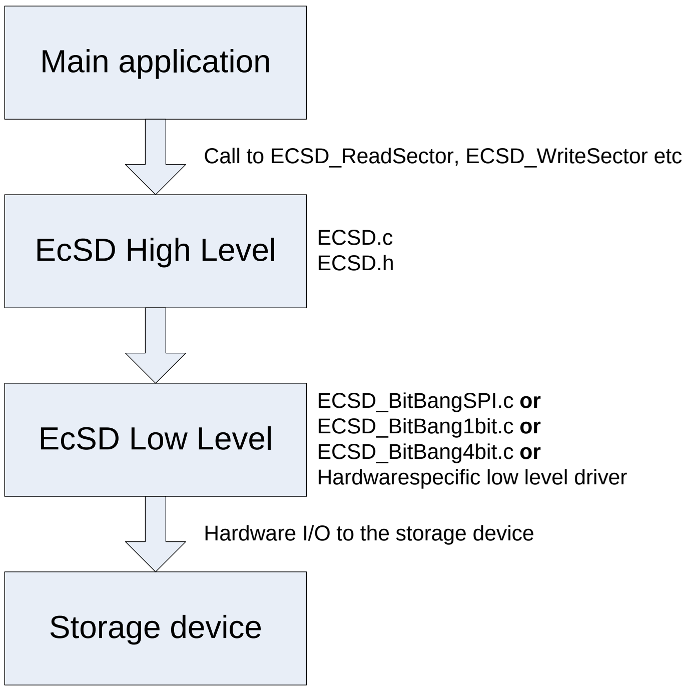
Section 2 Overview describes how EcSD works and how it relates to the rest of your application.

Section 3 Getting started using EcSD guides you through the decisions you need to make before adding EcSD to your code.

Section 4 Using EcSD is a guide on how to use EcSD code in your application.

2 Overview

2.1 Software layers



Main application
Call to ECSD_ReadSector, ECSD_WriteSector etc
EcSD High Level
ECSD.c
ECSD.h
EcSD Low Level
ECSD_BitBangSPI.c **or**
ECSD_BitBang1bit.c **or**
ECSD_BitBang4bit.c **or**
Hardwarespecific low level driver
Hardware I/O to the storage device
Storage device

Figure 1

EcSD consist of two basic layers. The high level layer and the low level layer.

The **high level layer** contains the higher level functions for reading and writing SD cards. The high level layer is always the same and is contained in ECSD.c and ECSD.h.

The **low level layer** communicates with the hardware and can perform basic functions like reading and writing bytes of data. It handles the lowest interaction with the hardware. For the lower layer, you can either use the included Bit-Banging drivers or write one specific to your own hardware.

2.2 Directory structure

```
ECSD 2.0.0/
   Documentation/
   Example Applications/
      IAR/
         AT91SAM7X256/
            ECSD_HardwareTest/
   Source/
      ECSD/
          LowLevel/
             BitBangSPI/
             BitBang1bit/
             BitBang4bit/
```

3 Getting started using EcSD

In order to use EcSD we suggest going through the following steps:

1. Choose which data transfer mode to use. (if you are using EcSD Lite, you must use SPI)
2. Design the hardware
3. Adapt the ECSD_HardwareTest sample to your environment
4. Test the hardware using the ECSD_HardwareTest sample and one of the BitBang drivers.
5. Use EcSD in your main application using one of the BitBang drivers
6. *Optional:* Test the hardware using the ECSD_HardwareTest sample and write your own low level driver.
7. *Optional:* Use EcSD in your real application using your own low level driver.

3.1 Choosing the mode

SD cards basically support 3 different modes of operation. SPI, 1-bit SD and 4-bit SD. Each has its benefits and drawbacks:

SPI:
+ Few pins used in communication.
+ Easy to implement
+ Low load on CPU if it has an SPI controller
- Support in SD cards is not mandatory (in practice, it is supported on most cards)
- Not as fast as 4-bit SD mode

SD 1-bit:
+ Few pins used in communication
+ Can be used in bus mode (several SD cards on the same bus)
+ Always supported
- Not as fast as 4-bit SD mode

SD 4-bit:
+ Fast transfer
+ Always supported
- Uses more pins
- Harder to implement if CPU doesn't have a dedicated controller

3.2 Adapt the ECSD_HardwareTest

To get started you will need to adapt the ECSD_HardwareTest for your environment.

3.2.1 Copy the core of the existing sample

Copy the code in main.c to a project for your own system. Modify the hardware specific parts of the startup, add your own startup setup.

3.2.2 Create Proejct.h

Create your own file called "Project.h" and define the macros needed. See the included Project.h for an example.

3.2.2.1 Timing macros

The following macros are always needed:

ECSD_DELAY_1_25_MICROSECONDS(): Called to cause a delay of 1.25 µs or more.

ECSD_DELAY_1_MILLISECOND(): Called to cause a delay of 1 ms or more.

ECSD_DELAY_100_MILLISECOND(): Called to cause a delay of 100 ms.

The reason for having these as different macros is that they are typically implemented in different ways. A delay of 1.25 µs or 1 ms is typically implemented as a busy wait while a delay of 100 ms can call the OS and let other threads run while waiting.

3.2.2.2 I/O macros

ECSD_INIT_IO(): Called to set up the I/O pins

ECSD_SET_CLK(x): Called with 0 to pull CLK low. Called with 1 to pull CLK high.

ECSD_SET_MOSI(x): Called with 0 to pull MOSI high. Called with 1 to pull MOSI low.

ECSD_GET_MISO(): Called to read MISO. Must return 0 if MISO is low, must return non-0 if MISO is high.

ECSD_SET_CS(x): Called with 1 to set CS high. Called with 0 to set CS low.

ECSD_GET_CP(): Called to see if a card is present. If your card is fixed to the system, always return 1.

ECSD_GET_WP(): Called to see if a card is write protected. If your card is never write protected, you can always return 0.

3.2.3 Testing the timings

You can test your timings by uncommenting TestDelay5Hz(),TestDelay500Hz(),TestDelay400kHz() or TestDelay25MHz() in the beginning of the ECSD_HardwareTest sample. Measure the resulting signal on the CLK pin on an oscilloscope and make sure they are below but close to the specified frequencies.

TestDelay5Hz() will test the ECSD_DELAY_100_MILLISECOND() macro.

TestDelay500Hz() will test the ECSD_DELAY_1_MILLISECOND() macro.

TestDelay400kHz() will test the ECSD_DELAY_1_25_MICROSECONDS() macro.

TestDelay25MHz() will test that ECSD_SET_CLK(x) doesn't go over 25MHz. If it is too fast you need to add a delay to the ECSD_SET_CLK macro.

3.3 Test your hardware

Start out by using the ECSD_HardwareTest sample. The sample will connect to your SD card and read and write blocks to it to verify that the hardware is working properly.

Please do not skip this step since most problems new users have are related to the hardware.

4 Using EcSD

4.1 Initialization

Start by initializing EcSD.

```
ECSD_Init();
```

If your card is not fixed to the system, you will need to check if there is an SD card present in the system.

```
while(ECSD_IsCardPresent() != ECSD_STATUS_OK);
```

Once the card is present, connect to the card.

```
if(ECSD_ConnectToCard() != ECSD_STATUS_OK)
{
   // Handle failure. Normally you would wait a while and
   // try again unless you believe the failure is permanent
}
```

4.2 Get disk and sector size

Obtain the disk and sector size.

```
WORD sectorSize;
DWORD numberOfSectors;

if(ECSD_GetVolumeInformation(NULL, &sectorSize, &numberOfSectors)
    != ECSD_STATUS_OK)
   halt("Unable to get volume information");
```

Note, the first parameter to ECSD_GetVolumeInformation is there for compatability with EcFAT. You can safely pass NULL.

4.3 Reading sectors

To read sector 477:

```
BYTE sectorData[512];

if(ECSD_ReadSector(NULL, 477, sectorData) != ECSD_STATUS_OK)
   halt("Unable to read sector 477");
```

Note, the first parameter to ECSD_ReadSector is there for compatability with EcFAT. You can safely pass NULL.

The size of the sectorData buffer must be at least sectorSize bytes big.

4.4 Writing sectors

To write sector 555:

```
BYTE sectorData[512];

// Fill sectorData with the data to write

if(ECSD_WriteSector(NULL, 555, sectorData) != ECSD_STATUS_OK)
   halt("Unable to write sector 555");
```

Note, the first parameter to ECSD_ReadSector is there for compatability with EcFAT. You can safely pass NULL.

The size of the sectorData buffer must be at least sectorSize bytes big.