PUMPKIN™
INCORPORATED
750 Naples Street • San Francisco, CA 94112 • (415) 584-6360 • http://www.pumpkininc.com

CubeSat Kit SD Card & EFFS-THIN User Manual

This manual describes how to use the EFFS-THIN Embedded Flash File System Library for SD Cards with the CubeSat Kit™.

CHANGELOG

Rev.	Date	Author	Comments
B	20110916	AEK	Updated PIC24 compatibility listings to include PIC24FJ256GB210, as used on PPM E1 & PSPM E.

Introduction

Pumpkin, Inc.'s CubeSat Kit (CSK) is a commercial off-the shelf (COTS) kit designed to radically reduce the time and effort required to build a functional satellite conforming to the current CubeSat (http://www.cubesat.org/) specification.

Starting with Rev C, the CubeSat Kit architecture has supported an SD Card for on-board mass storage. The SD Card interfaces to the CubeSat via a 4-wire interface (SD Card SPI Bus Protocol). This interface is connected to the CubeSat Kit Bus Connector.

Also starting with Rev C, each CubeSat Kit is supplied with a variety of software components. One of those components is a special library-only version of HCC-Embedded's (http://www.hcc-embedded.com/) well-regarded C-language Embedded Flash File System (EFFS), called EFFS-THIN. Because it does not contain source code, the version included with the CubeSat Kit is substantially less expensive than versions of EFFS-THIN that include source code. This "thin build" of EFFS is designed especially for small microcontrollers, and is very well-suited to use in the CubeSat Kit. EFFS-THIN provides a rich API for reading from and writing to FAT-formatted SD cards. By interfacing the CubeSat Kit to the SD Card via industry-standard FAT format, CubeSat Kit users can develop and debug their applications easily, by reading and writing to the SD Card with both CubeSat Kit hardware and with development PCs.

Information

All CubeSat Kit information – including this manual – is available online at http://www.cubesatkit.com/.

Note The version of this manual that accompanies each software release may be newer than that online.

Information for CubeSat Kit customers is available in the Customer Download Area. This area includes CubeSat Kit software, schematics, specification pages, etc.

SD Cards & the CubeSat Kit

Supported targets

The CubeSat Kit's SD Card libraries are compatible with the following targets and compilers:

CubeSat Kit	Target Processor	Compiler(s)
CSK /8051	C8051F120	Keil C51
CSK /MSP430	MSP430F169 MSP430F1611 MSP430F1612	Rowley CrossWorks for MSP430 v1 Rowley CrossWorks for MSP430 v2
	MSP430F2618	Rowley CrossWorks for MSP430 v2
CSK /PIC24	PIC24FJ256GA110 PIC24FJ256GB210	Microchip MPLAB C30 v3
CSK /dsPIC33	dsPIC33FJ256GP710	

Figure 1: Supported Targets and Compilers

SD Card SPI Interface

The CubeSat Kit interfaces to its SD Card via a 4-wire SPI interface.[1] The SD Card is an SPI slave, and the processor that reads from it and writes to it is an SPI Master. Four signals are used: `-CS_SD` (Chip Select for the SD Card), `SCLK` (master clock), `MOSI`/`SIMO` (Master Out Slave In), and `MISO`/`SOMI` (Master In Slave Out).

SD Cards operate in SPI Mode 0 (`CPOL` = 0, `CPHA` = 0). In this mode the base value of the clock[2] is zero (i.e., it idles low, and is high when active). The SD Card is accessed when its chip select line is active.

Incoming data[3] is sampled on the rising edge of the clock. Outgoing data[4] is presented on the falling edge of the clock. This is true for the SPI bus master and every slave on the SPI bus …

An example of the four SPI lines running on a CubeSat Kit with a clock speed of 500MHz is shown in Figure 2, below.[5] The SPI

[1] By 4-wire we mean the three usual SPI signals (`SCLK`, `MOSI` and `MISO`), and a discrete chip select signal. For the SD Card on the CubeSat Kit, the chip select signal is called `-CS_SD` and is present on `IO.0`.

[2] This is the SPI `SCLK` signal. This CubeSat Kit signal is called `SCK0` and is present on `IO.3`.

[3] This is the SPI `MOSI` signal. This CubeSat Kit signal is called `SD00` and is present on `IO.1`

[4] This is the SPI `MISO` signal. This CubeSat Kit signal is called `SDI0` and is present on `IO.2`.

Master is clocking in data from the SPI Slave (i.e., the SD Card). The `MOSI` from the microcontroller line remains high (all 1s) while the `MISO` line from the SD Card contains valid data (`0x00 08 00 0A 00 0C 00` … are shown). You can see that `-CS_SD` (aka `SS`) is low for the entire time of this transfer, `SCK` idles low, and `MISO` data is valid when `SCK` (aka `SCLK`) transitions from low to high.

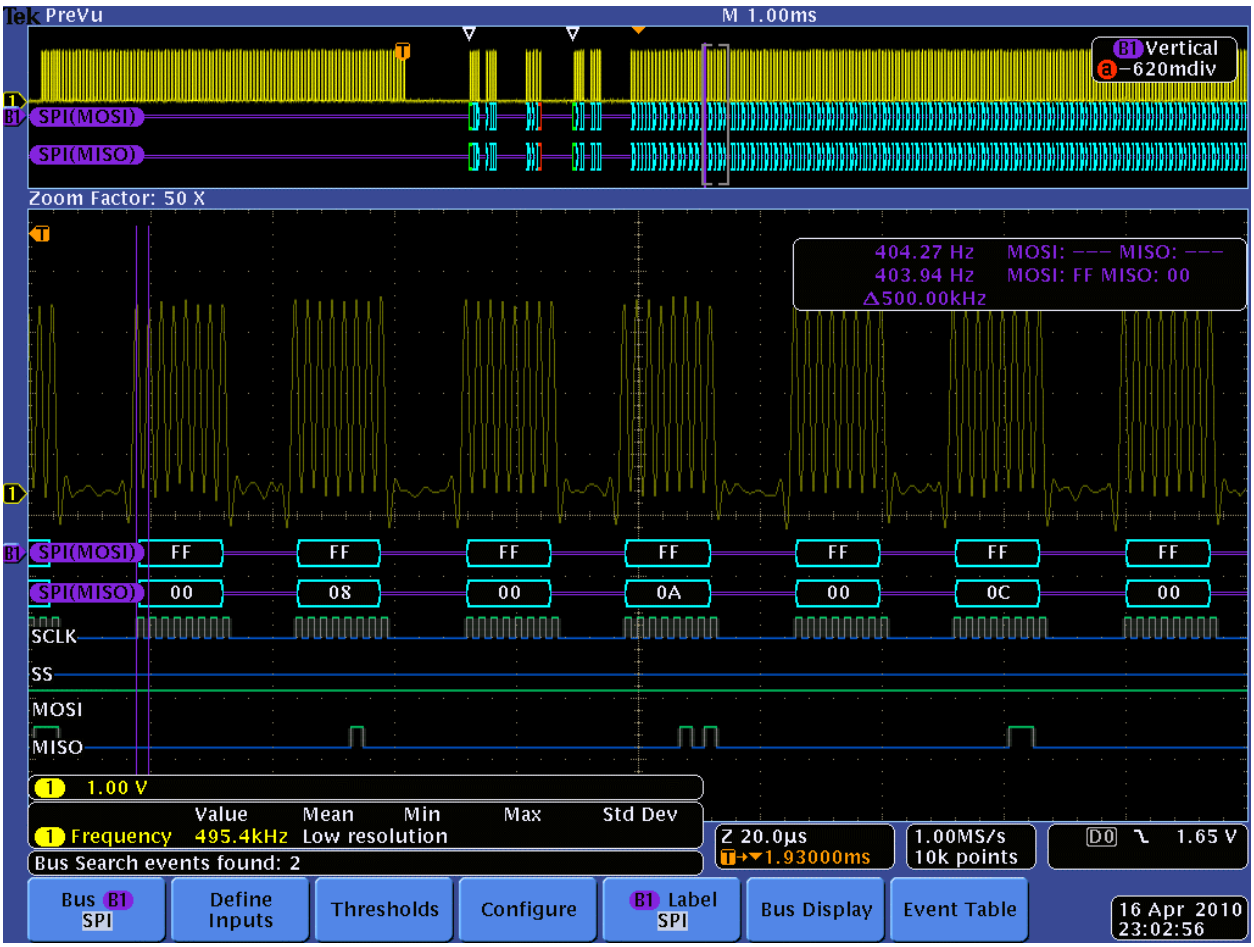

Figure 2: Oscilloscope capture of SD Card signals running at 500kHz on CubeSat Kit hardware

Figure 3 below shows SPI traffic between the microcontroller and the SD Card captured via an SPI analyzer.[6] An analyzer like this can show at-a-glance whether or not the SPI bus is running correctly, and what sort of data is traveling on the `MOSI` and `MISO` lines.

---

[5] Shown on a Tektronix® MSO 4034 with the DPO4EMBD Embedded Serial Triggering and Analysis option. Channel 1 (analog) is probing a buffered version of `SCK`. Bus 1 (digital, D3-D0) are defined as SPI signals.

[6] Captured via a Total Phase® Beagle™ I2C/SPI analyzer. These low-cost analyzers can be connected directly to the CubeSat Kit bus via the CubeSat Kit Breakout Board. The compatible connector on the CubeSat Kit Breakout Board maps `-CS_SD` to the chip select input of the Beagle™ I2C/SPI analyzer.

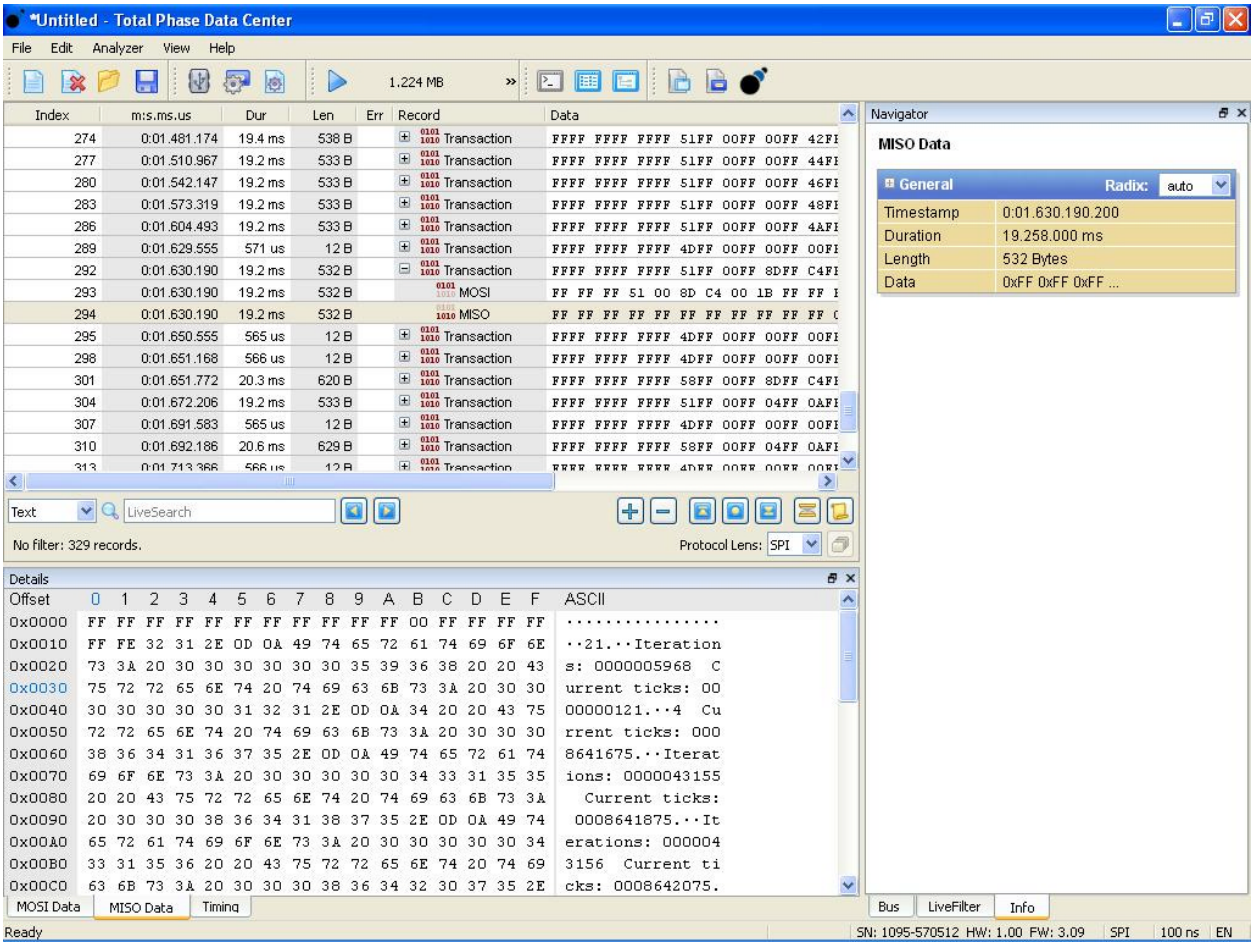

Figure 3: SPI analyzer capture and analysis of SD Card signals running at 500kHz on CubeSat Kit hardware

The CubeSat Kit's SD Card Interface

In CubeSat Kit Rev C, the interface was direct to the SD Card, with pull-up resistors and series current limiting resistors, as shown in Figure 4. The pullup resistors are required by the SD Card specification. The series resistors isolate the microcontroller bus from the SD Card when it was unpowered,[7] or powered but not selected via `-CS_SD`.

[7] `VCC_CARD` is under control of the microcontroller.

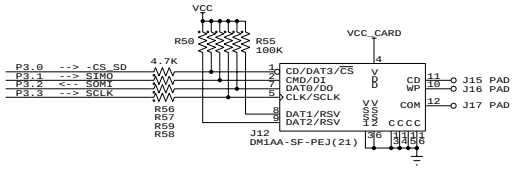

Figure 4: CubeSat Kit Rev C SD Card SPI interface

Additionally, resistor `R59` guaranteed that no matter what happened on the microcontroller end of the `MISO/SOMI` line,[8] there was no possibility of damaging the microcontroller's GPIO pin in the unlikely but still possible case that `P3.2` became an output with a value different from the SD Card's `DO` pin when `-CS_SD` was low (i.e., when the SD Card was selected).

Note Due to parasitic capacitance and output driver capabilities, the series resistors substantially limit the maximum operating speed of the SPI interface. The fastest possible SPI interface speeds to the SD Card can be obtained by replacing the series resistors with smaller value-resistors. Maximum speed is achieved for a value of zero Ohms for each of the series resistors.

Rev D improved on the situation by truly isolating the SD Card from the rest of the microcontroller bus via ultra-low-power bus transceivers that are in the high-impedance state when deselected and/or when either supply voltage is not present. The circuitry is shown in Figure 5.

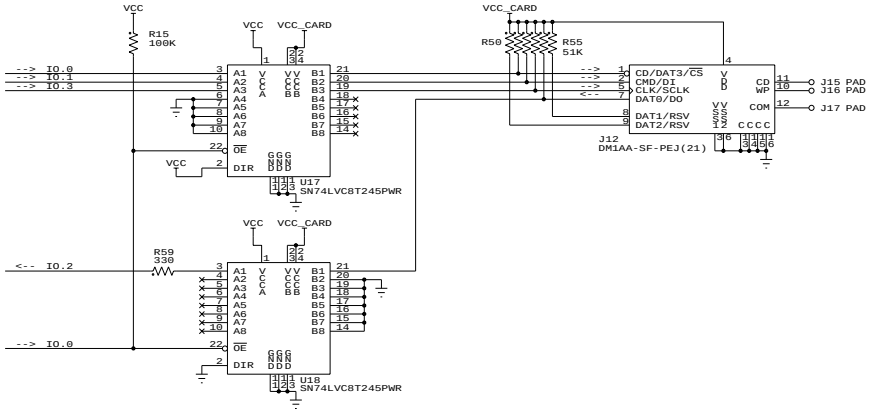

Figure 5: CubeSat Kit Rev D SD Card SPI interface

[8] On the MSP430 (the sole microcontroller used in the CubeSat Kit at the time of Rev C), `P3.2` is a general-purpose I/O pin that can also function as part of the SPI or I2C module of the USART0 peripheral. If the `P3SEL` and/or `P3DIR` registers were to change during an SPI transfer, a conflict between `P3.2` on the MSP430 and `DO` on the SD Card could arise, possibly leading to the destruction of some circuitry due to excessive currents through the output stages.

With the Rev D circuitry, the SD Card is completely isolated (i.e., hi-Z) from the IO.[3..0] bus signals that form the SPI interface to the SD Card whenever VCC_CARD is off or IO.0 (-CS_SD) is inactive.

Note In Rev D a smaller R59 is used to ensure no damage in the case of an output-to-output conflict between IO.2 and the MISO/SOMI signal at U18.3.

Practical Maximum Speed of the SPI Interface

SD Cards are rated at a minimum operating speed of 25MHz. However, the series resistor(s) in the CubeSat Kit's SPI interface to the SD Card limit the maximum possible operating speed to approximately 1MHz (`R59` = 330Ω). The effect of R59 on `MISO/SOMI` is shown in Figure 6 below for `SCLK` = 1MHz.[9] Note that `MISO/SOMI`'s rise and fall times are 30x longer than those of `SCLK`.

Note Key to signals in oscilloscope trace (top to bottom):

`-CS_SD`
`SCLK`
`MOSI/SIMO`
`MISO/SOMI`

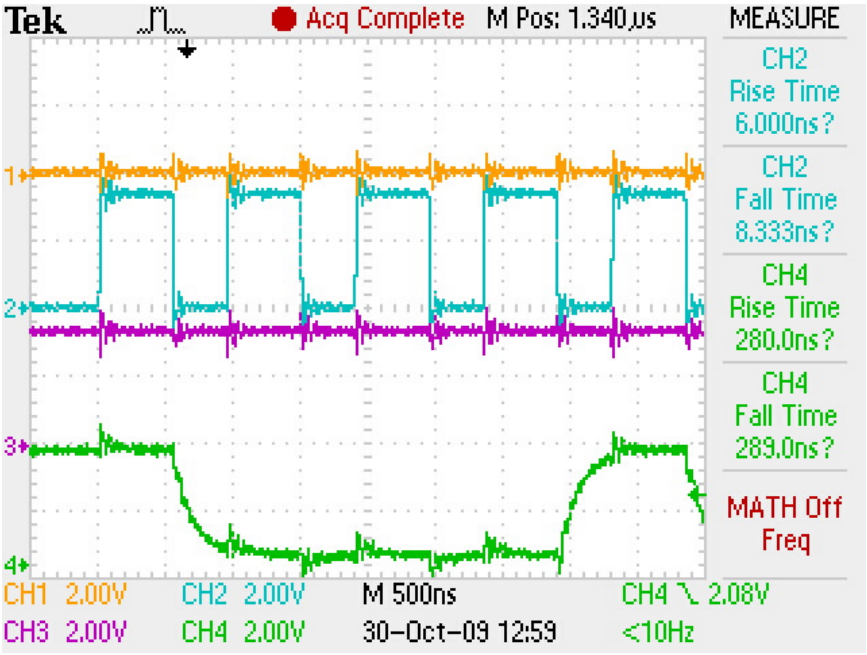

Figure 6: 1MHz SPI interface signals with R59 = 330Ω.

Changing `R59` to 0Ω results in much faster rise and fall times for `MISO/SOMI`, as shown in Figure 7. This shows that higher SPI interface speeds are possible with lower values of `R59`. See also *Running the SPI Interface at Other Speeds*, below.

[9] 1MHz was found empirically to be the maximum SPI interface speed for reliable operations with a SanDisk 512MB SD Card when driven by Pluggable Processor Module (PPM) D3 (MSP430F2618).

Note The maximum SPI interface speed of many microcontrollers is often limited to an integer fraction of the microcontroller's CPU clock. For example, a 7.3728MHz MSP430F1611 has a maximum SCLK speed of 2.458MHz. Therefore for some PPMs it may not be worth changing R59, and the 500kHz default SD Card SPI interface speed may be adequate (and safe) for the application.

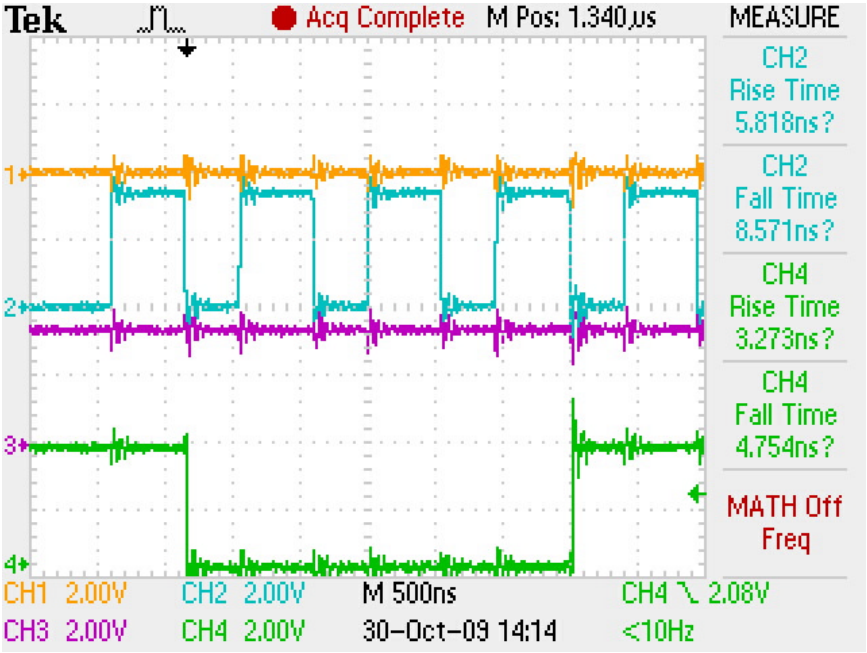

Figure 7: 1MHz SPI interface signals with R59 = 0Ω

Note If your microcontroller has a dedicated MISO/SOMI pin that functions only as an input at all times, or is internally current-limited to prevent damage to the microcontroller in case of an output fault, then R59 can always be safely reduced to 0Ω.

Warning Reducing R59 such that I_{pin} = VCC/R59 exceeds the *absolute maximum ratings* for microcontroller pin current may irreparably damage your microcontroller and/or the SD Card should a conflict arise due to incorrect I/O port settings.

Another High-Speed SPI Interface Example

Figure 8, below shows a captured set of traces for a Rev D CubeSat Kit running EFFS-THIN code on a 24.5MHz Silicon Labs® C8051F120 with an SPI clock of 1MHz. R69 has been replaced with a 0Ω resistor. Of note are the fast rise times (ca. 30ns) for both the signals from the processor through `U17` to the SD Card (`-CS_SD`, `SCLK`, `MOSI/SIMO`) and those from the SD Card through `U18` to the processor (`MISO/SOMI`).

Note Key to signals in oscilloscope trace (top to bottom):

`-CS_SD`
`SCLK`
`MOSI/SIMO`
`MISO/SOMI`

Figure 8: CubeSat Kit /8051 Rev D SD Card transfers at 1MHz

Note Of interest in Figure 8 is the clock-synchronous noise on the MISO/SOMI line after -CS_SD has gone inactive and the buffers U17 and U18 have been deselected. This is due to coupling between the SCLK (driven for the final eight clock cycles shown) and MISO/MOSI signals (not driven, high-impedance, because U18 is off), which travel together in close proximity on PCB itself.

SD Card formatting

SD Cards of up to 2GB capacity should be formatted as FAT (FAT16), not FAT32.

A precompiled library must be expressly built for FAT32 to support FAT32. FAT32 support requires additional memory.

A library built for FAT32 can also support FAT16.

SD Card compatibility

The CubeSat Kit's SD Card interface is electrically compatible with all SD Cards operating at VCC_CARD = +3.3V.

miniSD Cards are 100% compatible with the CubeSat Kit. They all support the SPI interface, and are therefore compatible with the CubeSat Kit. An SD-to-miniSD adapter must be used with the CubeSat Kit's SD Card socket.

microSD Cards are not necessarily compatible with the CubeSat Kit. This is because the SPI interface is optional on microSD Cards. Therefore the user must verify against a working SD Card whether a particular microSD Card works as well. An SD-to-microSD adapter must be used with the CubeSat Kit's SD Card socket.

Note The Rev D CubeSat Kit permits a VCC different from VCC_CARD (+3.3V). VCC and VCC_CARD voltages are set by the PPM and/or the Motherboard (MB) / Development Board (DB). In nearly all cases, VCC_CARD should be set to +3.3V.

Pumpkin does not validate different SD Card brands and manufacturers. Functional test and validation of SD Cards remains the obligation of the CubeSat Kit user.

Pumpkin offers extended-range industrial SD Cards for use with the CubeSat Kit.

EFFS-THIN & the CubeSat Kit

SD Card compatibility

EFFS-THIN is compatible with SD Cards of up to 2GB capacity.

A future release of EFFS-THIN for the CubeSat Kit may include SDHC Card support, starting at 4GB.

Acquiring EFFS-THIN for the CubeSat Kit

Customers with valid CubeSat Kit licenses can download the CubeSat Kit's EFFS-THIN installer directly from the CubeSat Kit website.

Each installer is built for a particular CubeSat Kit distribution. For example, CubeSat Kits for the MSP430 microcontroller are supplied with a version of the CubeSat Kit EFFS-THIN software that works only with MSP430 microcontrollers, and only with supported compilers.[10] Each library is supplied as a Windows® installer[11] and has a descriptive name, e.g.:

```
cubesatkit-effs-thin-msp430-1.8.9-rc2.exe
```

Installing EFFS-THIN

Obtain and run the CubeSat Kit EFFS-THIN installer on a Windows machine. The installer's default options will place the various required files in the standard CubeSat Kit locations compatible with example projects:

[10] Since EFFS-THIN is provided as a collection of pre-compiled libraries, only the compiler used to generate the libraries at Pumpkin can be used by the user to create applications that use EFFS-THIN.

[11] All Pumpkin installers are created with NSIS, and open-source installer / packager. NSIS archives can be unzipped via the free 7-Zip utility. Thus, the files in an NSIS installer can be extracted on a Linux machine via 7-Zip.

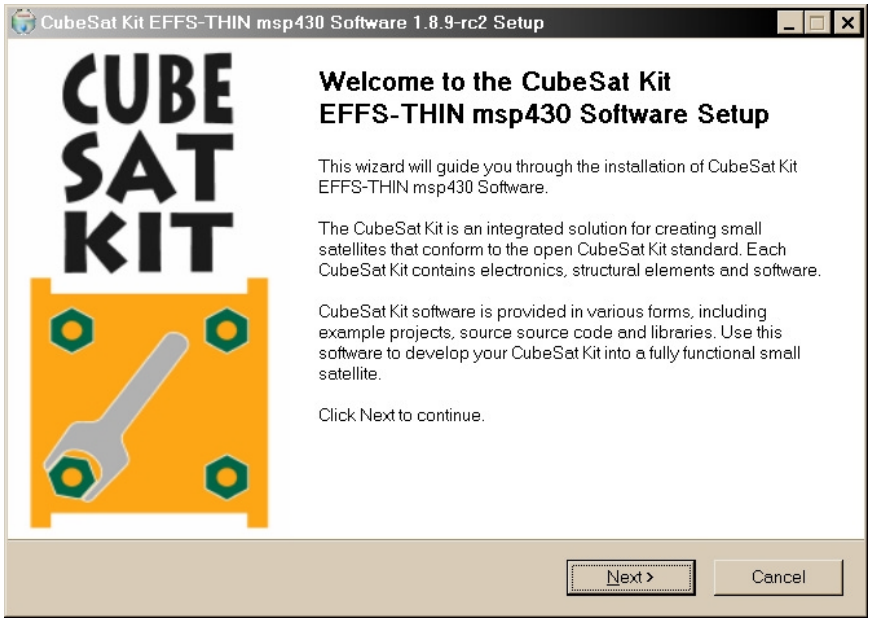

Figure 9: CubeSat Kit EFFS-THIN software installer

Note The CubeSat Kit EFFS-THIN installer simply installs a range of files to particular folders. There are no executables installed by the CubeSat Kit EFFS-THIN installer.

EFFS-THIN Example Projects

A working example project that utilizes EFFS-THIN on the CubeSat Kit is invaluable in understanding how to use EFFS-THIN and how to build applications with EFFS-THIN.

Each CubeSat Kit EFFS-THIN distribution contains at least one CubeSat Kit-specific example project that demonstrates the use of EFFS-THIN on CubeSat Kit hardware. After installing EFFS-THIN, find the example project, build it, download it to your CubeSat Kit, and run it.

Note Each example project includes an `abstract.txt` file that explains what the application is doing, and how to observe its operation.

Note CubeSat Kit EFFS-THIN example projects may require other Pumpkin software to be installed. Check the prerequisites outline in the project's abstract.txt file if you are having difficulty successfully building an example project.

Using EFFS-THIN

To use EFFS-THIN, you must select a precompiled library that contains the features that you want. Then you add a few CubeSat Kit and EFFS-THIN components to your project, make a few settings within your project, and make the appropriate calls to the EFFS-THIN API from within your application.

Note To use EFFS_THIN competently, you must familiarize yourself with the included EFFS-THIN documentation.

Selecting a Precompiled Library

The EFFS-THIN supplied with the CubeSat Kit is supplied in a precompiled form – no source code is included. This is done to minimize the additional cost of the EFFS_THIN software, and to simplify the use of this capable software.

Libraries are compiled at Pumpkin with an eye for which types of libraries are most useful for CubeSat Kit users. Some users – e.g. those with MSP430 processors – are likely to want the smallest possible library, due to memory constraints. Others might want the fastest, or one with advanced features (e.g., FAT32 support).

The CubeSat Kit EFFS_THIN libraries are numbered for easy identification.

A brief descriptive overview of the currently available libraries is listed below in Table 3. It is beyond the scope of this document to explain the various features and build options of EFFS-THIN. To understand the build settings used to create each EFFS_THIN library, view the contents of the corresponding CubeSat Kit EFFS-THIN configuration file `(csk_effs_thin-N.h)`.

-N	Type	Like	Notes & Typical Applications
-1	SUPER THIN	-4	1 file handle FAT16 only basic functions included limited functionality via small RAM footprint
-2	THIN	-3	2 file handles FAT12, FAT16 & FAT32 nearly all functions included
-3	THIN	-2	3 file handles FAT12, FAT16 & FAT32 nearly all functions included
-4	SUPER THIN	-1	1 file handle FAT16 only no directory functions limited functionality via small RAM footprint

Table 1: List of available CubeSat Kit EFFS-THIN libraries

Add csk_sd.c

Add your CubeSat Kit's `csk_sd.c` to your project. This source-code module implements a simple API to control the SD Card and is used by the EFFS-THIN SPI driver. It can be found in:

```
/Pumpkin/CubeSatKit/<target>/Src
```

Add EFFS-THIN SPI Driver

Add the EFFS-THIN SPI driver for your CubeSat Kit to your project. The SPI driver is provided in source form (`*.c`, `*.h`).

CubeSat Kit EFFS-THIN SPI drivers follow the naming convention `effs_thin_mmc_drv_xxx.c` and are located in:

```
/Pumpkin/CubeSatKit/<target>/Src
```

Note CubeSat Kit EFFS-THIN SPI drivers are peripheral-dependent and are therefore built for a narrow range of processors that all have the same peripheral. If an existing SPI driver does not match the peripherals on your processor (for instance, if you have designed your own PPM), you can use the source code of the SPI driver as a basis for creating your own, compatible SPI diver that can be linked in your application to the EFFS-THIN library.

Add EFFS-THIN Library

Add the EFFS-THIN library for your CubeSat Kit to your project. The library is provided in object/library form (`*.lib`, `*.a`, `*.hza`, etc.).

CubeSat Kit EFFS-THIN libraries follow the naming convention `lib_effs_thin_<target>[-<options>]-N.lib` and are located in:

```
/Pumpkin/CubeSatKit/<target>/Lib/<compiler>
```

Note CubeSat Kit libraries are peripheral-independent and are therefore built for entire families of processors. If your CubeSat Kit uses a Pumpkin-supplied PPM, then an appropriate library will be available. If you have created your own PPM with a processor that is from the same family, then the CubeSat Kit EFFS-THIN library will probably work with it, too.

Add Include Paths

In order for your application to build correctly, certain files must be located during the C preprocessing stage. Therefore, add the following include paths to your project:

```
/Pumpkin/CubeSatKit/HCC-Embedded/EFFS-THIN/Src
/Pumpkin/CubeSatKit/<target>/Src
```

Note The EFFS-THIN libraries are *precompiled*. Therefore *you cannot and must not* make any changes to the EFFS-THIN header files in the directories above, or your program may not function correctly.

Don't forget to add an include path to your project's home directory as well …

Define the CSK_EFFS_THIN_LIB Symbol

In order for your application to build correctly with an EFFS-THIN library, you must specify which precompiled EFFS-THIN library you wish to use. This is done via the `CSK_EFFS_THIN_LIB` symbol. You define this symbol to be the integer value (i.e., 1, 2, …) that corresponds to the "flavor" of the EFFS-THIN library you wish to use in your application.

For example, if you want to the CubeSat Kit EFFS-THIN library that supports up to three concurrently open file handles, and nearly all of the THIN build options, you would define

```
CSK_EFFS_THIN_LIB=3
```

and apply it via your build environment's IDE to apply to the entire project.

Note The actual settings for each value `N` of `CSK_EFFS_THIN_LIB` can be viewed in the read-only files `csk_effs_thin-N.h`, located in `/Pumpkin/CubeSatKit/HCC-Embedded/EFFS-THIN/Src`.

Calling the EFFS-THIN API

You must include the appropriate header in each `*.c` source file that makes calls to the EFFS-THIN API:

```
#include <thin_usr.h>
```

Then, you can make calls to the EFFS-THIN API, e.g.

```
#include <thin_usr.h>
  …
  F_FILE *file;
  f_initvolume();
  file = f_open("myfile.txt", "a+");
  …
```

Figure 10: Example of EFFS-THIN API calls in C

will open a file called "`myfile.txt`" in append mode on the CubeSat Kit's SD Card.

Running the SPI Interface at Other Speeds

By default, the EFFS-THIN code initially queries the SD Card with the SPI clock speed set to 100kHz. This happens during `f_initvolume()`,[12] when the EFFS_THIN code calls `spi_set_baudrate()` internally with an argument of 100000, i.e. 100kHz. This is the value of `CSK_SD_INIT_SPEED`.[13]

At the end of `f_initvolume()`, once all of the SD Card initialization is complete, the SPI clock speed is raised by the EFFS-THIN code via another internal call to `spi_set_baudrate()`. In the CubeSat Kit implementation, the higher speed SPI baudrate is defined by `CSK_SD_RUN_SPEED`, and is normally 500kHz.[14]

Users who wish to access their SD Cards at different speeds (after initialization) can do so easily by calling the function `spi_set_user_baudrate()` with the desired SPI clock as its argument, in Hz, as shown in Figure 11, below.

Note The resolution at which a particular SPI peripheral's clock can be set varies based on processor architecture. Therefore there are real-world limits on how accurately a desired SPI clock can be set via `spi_set_user_baudrate()`. Consult the appropriate datasheet(s) and the CubeSat Kit SPI driver source files for more information.

A complementary function `spi_get_user_baudrate()` is also available. It returns the desired user baud rate, in Hz.[15]

[12] `f_initvolume()` is part of the EFFS-THIN source code. Therefore, CubeSat Kit users do not have source-code access to it, nor to any other EFFS-THIN functions. These functions are compiled into the EFFS-THIN libraries that are supplied with the CubeSat Kit. The only related functions that CubeSat Kit customers can view as source code are those associated with the SD Card drivers – they are named `spi_xyz()`. These functions are called at various times by the EFFS-THIN library code.

[13] Defined in the SPI driver's header file.

[14] In an off-the-shelf EFFS_THIN implementation, the EFFS-THIN code normally selects an SPI clock rate based on what the particular SD Card reports as its maximum possible operating speed. Because of the issues surrounding the CubeSat Kit's buffering of the SD Card, the maximum reported speed may not be compatible with CubeSat Kit operation. Therefore, a default maximum speed that has been empirically determined is used instead by the CubeSat Kit.

[15] For extremely size-sensitive applications (e.g. MSP430F16xx), defining the symbol `CSK_SD_DISABLE_USER_BAUDRATE` will suppress the two user functions `spi_set_user_baudrate()` and `spi_get_user_baudrate()`.

```
#include <thin_usr.h>
  …
  volatile unsigned int error_code;
  …
  spi_set_user_baudrate(2350000);
  while ((error_code = f_initvolume()) != 0);
  …
```

Figure 11: Configuring SD Card speeds to 2.35MHz.

Thus, it is possible to run the SD Card both faster than the standard 500MHz run speed (e.g., for those users who have replaced `R59` with a 0Ω resistor), and slower than the default initialization speed of 100kHz (e.g., in an attempt to limit power consumption).

Warning `spi_set_user_baudrate()` does not perform any bounds checking on its argument. The argument is of type `unsigned long`.

Examples of Higher SPI Interface Speeds

By using `spi_set_user_baudrate()`, SD Cards at the speeds shown in Table 2, below, have been tested at Pumpkin and appear to operate properly.

Warning These setups have not been tested exhaustively and therefore no guarantee is made as to their suitability for any application. Use at your own risk.

CSK /	SPI Clock Speed	R59	Notes
8051	12.25MHz	0Ω	2011-03-21 CSK DB Rev D CSK PSPM B1 Rev A SYSCLK = 24.5MHz SanDisk 128MB miniSD via adapter

Table 2: Examples of higher SD Card speeds

List of EFFS-THIN Libraries

Table 3 describes the available CubeSat Kit EFFS-THIN libraries, their intended processor targets, and compatible compilers.

CSK /	Name	Intended Targets	Notes
8051	KC51-v8/ lib_effs_thin_c8051-xdata-N	C8051F120	For Keil Cx51 v8.x
MSP430	RA430-v1/ lib_effs_thin_msp430-N	MSP430F169 MSP430F1611 MSP430F1612	For Rowley CW430 v1.4
	RA430-v2/ lib_effs_thin_msp430-N		For Rowley CW430 v2.0
	RA430-v2/ lib_effs_thin_msp430x-N	MSP430F2618	
PIC24	MCC30-v3/ lib_effs_thin_pic24-sm-coff-N	PIC24FJ256GA110 PIC24FJ256GB210	For Microchip MPLAB C30 v3.2.3
dsPIC33	MCC30-v3/ lib_effs_thin_dspic33-sm-coff-N	dsPIC33FJ257GP710	For Microchip MPLAB C30 v3.2.3

Table 3: EFFS-THIN library applications

List of SPI Drivers

Table 4 describes the available SPI drivers, their intended processor targets, and details of their implementations.

CSK /	Name	Intended Targets	Notes
8051	effs_thin_mmc_drv_spi0	C8051F120	Uses SPI0 w/NSS. Runs at 500kHz. No interrupts. No DMA.
MSP430	effs_thin_mmc_drv_us0	MSP430F169 MSP430F1611 MSP430F1612	Uses USART0:SPI Runs at 500kHz. No interrupts. No DMA.
	effs_thin_mmc_drv_ucb0	MSP430F2618	Uses USCI_B0:SPI Runs at 500kHz. No interrupts. No DMA.
PIC24 dsPIC33	effs_thin_mmc_drv_spi1	PIC24FJ256GA110 PIC24FJ256GB210 dsPIC33FJ257GP710	Uses SPI1 Runs at 500kHz. No interrupts. No DMA.

Table 4: SPI driver applications

SD Card Database

Using a Rev D Development Board (DB) with PSPM B1, Pumpkin has tested the SD Cards listed in Table 5 for basic functionality.

Note The EFFS-THIN configuration for each card may vary based on card requirements. miniSD and microSD cards were tested using SD Card adapters.

Mfgr	Model	Size	Type	Good?	SPI speed	Format	Notes
Panasonic		32MB	SD	yes	2.5MHz	FAT	
SanDisk		128MB	miniSD	yes	2.5MHz	FAT	
Kingston		2GB	SD	yes	2.5MHz	FAT32	
Motorola		2GB	microSD	yes	2.5MHz	FAT32	
Patriot		2GB	microSD	yes	2.5MHz	FAT32	
Polaroid		4GB	SDHC	no	2.5MHz	FAT32	by PNY
SanDisk	Extreme III	4GB	SDHC	yes	2.5MHz	FAT32	
Kingston		4GB	microSDHC	yes	2.5MHz	FAT32	Class 4
Patriot	Signature Line	4GB	microSDHC	yes	2.5MHz	FAT32	Class 4
Wintek	filemate	4GB	microSDHC	yes	2.5MHz	FAT32	

Table 5: SD Cards tested on the CubeSat Kit using EFFS-THIN

Note SDHC cards require a build of EFFS-THIN that supports SHDC.

Frequently Asked Questions (FAQ)

General

Why can't I change some of the EFFS-THIN configuration options (e.g. F_xxxx)?

Those configuration options require recompiling the EFFS-THIN source code in order to create a new, linkable object module. The CubeSat Kit provides precompiled EFFS-THIN libraries – no source code. Therefore the only valid configuration options are the ones Pumpkin uses to generate the CubeSat Kit's EFFS-THIN libraries.

None of the CubeSat Kit's EFFS-THIN libraries have exactly the configuration(s) I want(need). What can I do?

You can request that Pumpkin create a library that has the configuration options you desire. If approved, we will add it to the precompiled libraries that are supplied as part of each CubeSat Kit EFFS-THIN distribution.

Alternatively, you can purchase an EFFS-THIN source license from HCC-Embedded.

I would like to implement an extension to EFFS-THIN, like TMR (Triple Majority Redundancy). How can I do that?

The SPI-level drivers are included as source code in each CubeSat Kit EFFS-THIN distribution. You are free to modify them as you see fit, for use on a CubeSat Kit. For modifications to the EFFS-THIN core, you may need to acquire your own source-code license for EFFS-THIN. Contact HCC Embedded directly for more information.

Software

How much program and data memory does EFFS-THIN use?

This depends on which EFFS-THIN library you choose (and hence, on the build configuration of EFFS-THIN). You can check the build results and/or map file of the example projects to see how large EFFS-THIN is in a real-world situation.

SD Card transfers are done in blocks of 512 bytes. Therefore the data memory requirements of an EFFS-THIN application will consume around 512 bytes per open file handle.

Why aren't csk_sd.c and the SPI driver included in the EFFS-THIN library?

These two files are driver-level files for the CubeSat Kit. The EFFS-THIN library files are driver-independent files. By keeping the two separate, a more flexible system is created wherein e.g. the user can implement changes and extensions as required.

Hardware

What speeds does the CubeSat Kit's SD Card run at?

The SD Card specification requires that initial discovery of the SD Card be performed at an SPI bus clock speed of less than 400kHz.[16] Thereafter, the SD Card specification permits the interface to run as fast as the card permits. SD Cards are required to support a minimum interface speed of 25MHz. However, this is often much greater than the highest possible speed that the CubeSat Kit can support, due to either the lack of speed of the processor, or interface issues between the processor and the SD Card, or both. Therefore each SPI driver sets a maximum speed that takes effect after the discovery phase.

In the CubeSat Kit, the default discovery phase SD Card clock speed is 100kHz, and the default run phase SD Card clock speed is 500kHz.

[16] The EFFS-THIN source code takes a conservative approach and performs initial discovery at 100kHz or as near to 100kHz as possible.

Users are free to change the SPI clock speed via the `spi_set_user_baudrate()` function.

How is the SD Card interface's speed related to my processor's clock speed?

The SPI driver sets the speed of the SD Card interface. Generally speaking, SPI interfaces are relatively insensitive to clock speeds. During the discovery phase of the SPI protocol must communicate with the SD Card at less than 400kHz. After that, communications can proceed at whatever clock speed the SD Card hardware can handle.

Can I optimize the SPI driver?

Sure – changes to the SPI driver for your particular processor are entirely up to you. The supplied SPI drivers are simply a conservative starting point for more advanced SPI drivers.

Can I run multiple SD Cards in a CubeSat Kit? Can I connect multiple processors to one or more SD Cards in a CubeSat Kit?

By default, the CubeSat Kit's EFFS-THIN libraries interface only to the SD Card that is on the CubeSat Kit Motherboard. This is controlled by the SPI driver. It is relatively simple to expand the driver to support two separate SD Cards from a single processor. You could also run two processors, each to its own card, or you could even share a card across multiple processors. All of this can be done by creating a purpose-built SPI driver.

Licensing

On what targets can I use applications built with the CubeSat Kit's EFFS-THIN?

The EFFS-THIN license that accompanies the CubeSat Kit permits you to use the EFFS-THIN libraries to generate applications exclusively for the CubeSat Kit.