SPIflash

SPI flash chip Arduino and chipKit library

Manual

Introduction:

The idea for this library came to me long ago when I noticed the empty footprint on some TFT display modules. I thought it would be a good idea to have some extra storage space available for projects.

This library provides basic support for handling SPI flash memory chips. It also supports a very simple, read-only file system that can be used for storing text files and text (string) resource files. The file system also handles image and audio data for add-on libraries.

The included FlashUploader tool (sorry, Windows only) can be used to upload files to the file system on the chip or create files that can be uploaded from a SD card to the flash chip. Note that the FlashUploader tool includes file types that are not supported directly by this library but requires add-on libraries.

You can always find the latest version of the library at **http://www.RinkyDinkElectronics.com/**

For version information, please refer to **version.txt**.

SUPPORTED CHIPS:

Manufacturer	Model	Size (Mbits)	Size (K/Mbytes)	Tested Package
SST / Microchip	**SST25VF020B**	2 Mbits	256 Kbytes	SOIC-8
SST / Microchip	**SST25VF040B**	4 Mbits	512 Kbytes	SOIC-8
SST / Microchip	**SST25VF080B**	8 Mbits	1 Mbyte	SOIC-8
SST / Microchip	**SST25VF016B**	16 Mbits	2 Mbytes	SOIC-8
SST / Microchip	**SST25VF032B**	32 Mbits	4 Mbytes	SOIC-8
SST / Microchip	**SST25VF064C**	64 Mbits	8 Mbytes	SOIC-16
Winbond	**W25Q80BV**	8 Mbits	1 Mbyte	SOIC-8
Winbond	**W25Q16BV**	16 Mbits	2 Mbytes	SOIC-8
Winbond	**W25Q32BV**	32 Mbits	4 Mbytes	SOIC-8
Winbond	**W25Q64FV**	64 Mbits	8 Mbytes	SOIC-8
Winbond	**W25Q128BV**	128 Mbits	16 Mbytes	SOIC-16
Winbond	**W25Q128FV**	128 Mbits	16 Mbytes	SOIC-8
Winbond	**W25Q256FV**	256 Mbits	32 Mbytes	SOIC-16
MXIC	**MX25L1605D**	16 Mbits	2 Mbytes	SOIC-8
MXIC	**MX25L3205D**	32 Mbits	4 Mbytes	SOIC-8
MXIC	**MX25L6405D**	64 Mbits	8 Mbytes	SOIC-16

The library checks the vendor and chip ID on initialization so chips not on this list will not work.

INCLUDED EXAMPLE DATASETS:

These files can be found in the **/tools/FlashUploader/Example Datasets** folder.

Full name	Short name	Minimum Flash Chip Size (Mbits)
Demo Data.*	DEMO.SFD	2 Mbits
Earth_Map.*	EARTH.SFD	32 Mbits
Earth_Map_HR.*	EARTH_HR.SFD	128 Mbits
TestImages_240x320.*	240X320.SFD	8 Mbits
TestImages_240x400.*	240X400.SFD	8 Mbits
TestImages_320x240.*	320X240.SFD	8 Mbits
TestImages_400x240.*	400X240.SFD	8 Mbits
TestImages_480x272.*	480X272.SFD	8 Mbits
TestImages_800x480.*	800X480.SFD	32 Mbits
TestImages_Mono_For_Colordisplays.*	MONO_C.SFD	2 Mbits
TestImages_Mono.*	MONO.SFD	2 Mbits
TestImages_Mono_Large.*	MONO_L.SFD	2 Mbits

If a specific dataset is required by an example sketch it will be noted in the opening comments of that sketch.

DEFINED LITERALS:

General errors	
Errors returned from most functions if something went wrong.	
ERR_FILETYPE_INCORRECT:	0xFFFF
ERR_FILE_DOES_NOT_EXIST:	0xFFFE
ERR_BUFFER_OVERFLOW:	0xFFFD
ERR_OUT_OF_RANGE:	0xFFFC
ERR_FILE_NOT_OPEN:	0xFFFB
ERR_FILE_ALREADY_OPEN:	0xFFFA
ERR_NO_AVAILABLE_HANDLES:	0xFFF9
ERR_SEEK_PAST_FILE_START:	0xFFF8
ERR_SEEK_PAST_FILE_END:	0xFFF7
ERR_AT_EOF:	0xFFF6
ERR_ACCESS_IS_RESTRICTED:	0xFFF5
ERR_NO_ERROR:	0x0000 (= OK)

getFileSize() errors	
Due to the size of the return variable from getFileSize() it needs its own set of error messages.	
ERROR_FILE_DOES_NOT_EXIST:	0xFFFFFFFE
ERROR_ACCESS_IS_RESTRICTED:	0xFFFFFFF5

INFORMATIONAL VARIABLES:

JEDEC Information	
Can be used to access the JEDEC information from the currently connected flash chip.	
ID_manufacturer:	Manufacturer ID
ID_type:	Chip type ID
ID_device:	Device specific ID

Text Information	
Can be used to access the information from the currently connected flash chip in text form.	
Text_manufacturer:	Contains the name of the chip manufacturer
Text_type:	Device type (Currently only “SPI Serial Flash”)
Text_device:	Model name
Capacity:	Size of the flash chip in Mbits (integer)

DEFINED FILE TYPES:

File types	
Binary:	1
Text:	2
Text Resource:	3
Color Image:	4
Monochrome image (for use on color screens):	5
Monochrome image (for use on monochrome screens):	6
Audio:	7
Custom 1:	32
•	•
•	•
•	•
Custom 16:	47

FUNCTIONS:

SPIflash;

The main class constructor when using the hardware SPI pins with the default SS pin.

Parameters:	none
Usage:	`SPIflash myFlash; // Start an instance of the SPIflash class`
Notes:	Note that there are no parentheses when using this constructor.

SPIflash(SS);

The main class constructor when using the hardware SPI pins with a specific SS pin.

Parameters:	SS: Pin for slave select / chip enable (CE)
Usage:	`SPIflash myFlash(9); // Start an instance of the SPIflash class`

SPIflash(SI, SO, SCK, CE);

The main class constructor when using a software SPI communication protocol (i.e. Not using the hardware SPI pins).

Parameters:	SI: Pin for serial data to the chip
	SO: Pin for serial data from the chip
	SCK: Pin for serial clock signal
	CE: Pin for chip enable / slave select (SS)
Usage:	`SPIflash myFlash(5, 6, 7, 4); // Start an instance of the SPIflash class`
Notes:	Using software SPI is a lot slower than hardware SPI...

begin();

Initialize the instance for use.

Parameters:	None
Usage:	`myFlash.begin(); // Initialize the myFlash object`

readStatus();

Returns the current status byte from the chip.

Parameters:	None
Returns:	(uint8_t) Current status byte
Usage:	`status = myFlash.readStatus(); // Read the status byte`

readPage(page);

Read a complete 256 byte page from the chip into the pre-defined buffer array.

Parameters:	page: Number of the page you wish to read
Usage:	`myFlash.readPage(0x1FF); // Read page 0x1FF into the buffer`
Notes:	Access the buffer through myFlash.buffer[]

writePage(page);

Write a complete 256 byte page from the pre-defined buffer array into the chip.

Parameters:	page: Number of the page you wish to write
Usage:	`myFlash.writePage(0x1FF); // Write the contents of the buffer to page 0x1FF in the chip`
Notes:	Access the buffer through myFlash.buffer[]

waitForReady();

Wait until an already started asynchronous operation has finished.

Parameters:	None
Usage:	`myFlash.waitForReady();  // Wait for the chip to finish the current operation`
Notes:	This function will wait until the BUSY flag (bit 0) of the chip status register clears.

eraseChip();

Erase all the data currently stored in the chip.

Parameters:	None
Usage:	`myFlash.eraseChip(); // Start a chip erase operation`
Notes:	Some chips take quite a while to erase. This function will not return until the erase operation has finished.

IMPORTANT:
The following functions will only work when the data on the chip has been formatted with the proprietary file system created by the FlashUploader application.
Using these functions on other data may cause unpredictable results and is not supported.
Please note that the file system is currently read-only.

fileOpen(fileID);

Open a file for reading.

Parameters:	fileID: ID of the file you want to open for reading
Returns:	(uint16_t) filehandle *or* a general error (see defined literals)
Usage:	`handle = myFlash.fileOpen(8); // Attempt to open the file with ID #8 for reading`
Notes:	The SPIflash library can handle 5 simultaneously open files. You cannot open Text Resource files with fileOpen(). Use readTextResource() to access those files.

fileClose(filehandle);

Close a previously opened file.

Parameters:	filehandle: Filehandle of the file you want to close
Returns:	(uint16_t) ERR_NO_ERROR (0) *or* a general error (see defined literals)
Usage:	`result = myFlash.fileClose(handle); // Attempt to close a file`

restrictAccess(filehandle);

Restrict access to the file system to a single file. Useful when you need time-critical access to a file.

Parameters:	filehandle: Filehandle of the file you want to have exclusive access to
Returns:	(uint16_t) ERR_NO_ERROR (0) *or* a general error (see defined literals)
Usage:	`result = myFlash.restrictAccess(handle); // Restrict access to the file system`
Notes:	The file must be open before you can restrict access.

unrestrictAccess();

Remove file system restrictions set by `restrictAccess()`.

Parameters:	None
Usage:	`myFlash.unrestrictAccess(); // Remove restrictions`

isRestricted();

Check if there is a file system restriction in place.

Parameters:	None
Returns:	(uint8_t) Filehandle of the file with exclusive access or 0xFF if no restrictions are in place
Usage:	`result = myFlash.isRestricted(); // Check if a restriction is in place`

fileSeek(filehandle, offset);

Change the position for the next read within a file.

Parameters:	filehandle: Filehandle of the file you want to manipulate offset: Number of bytes to change the position by Positive values move the pointer towards the end of the file while negative values Moves the pointer towards the start of the file. 0 will set the position to the start of the file.
Returns:	(uint16_t) ERR_NO_ERROR (0) *or* a general error (see defined literals)
Usage:	`result = myFlash.fileSeek(handle, 10); // Move the pointer 10 bytes towards the end of the file`

fileRead(filehandle, buffer, buffersize);

Read data from a previously opened file.

Parameters:	filehandle: Filehandle of the file you want to read from buffer: Buffer to put the read data into buffersize: Size of the buffer in bytes
Returns:	(uint16_t) number of bytes read *or* a general error (see defined literals)
Usage:	`result = myFlash.fileRead(handle, buf, sizeof(buf)); // Read data into the buf array`
Notes:	This function will read data until the buffer is full or EOF is encountered. If reading from text files the buffer will always contain a string terminator (0 byte) so if the buffer size is 80 bytes you will never get more than 79 characters (+ the terminator) back.

fileReadLn(filehandle, buffer, buffersize);

Read a line of text from a previously opened file.

Parameters:	`filehandle: Filehandle of the file you want to read from`
	`buffer:     Buffer to put the read data into`
	`buffersize: Size of the buffer in bytes`
Returns:	`(uint16_t) number of bytes read` *or* `a general error (see defined literals)`
Usage:	`result = myFlash.fileReadLn(handle, buf, sizeof(buf)); // Read text into the buf array`
Notes:	`This function will read data until the buffer is full, a line break or EOF is encountered.`
	`DOS/Windows (CR+LF), Mac (CR) and Unix (LF) line breaks should all be handled correctly.`
	`When reading from text files the buffer will always contain a string terminator (0 byte) so if the buffer size is 80 bytes you will never get more than 79 characters (+ the terminator) back.`
	`If the buffer was too small to read the entire line the function will return ERR_BUFFER_OVERFLOW.`

getFileType(fileID);

Get the file type of a file.

Parameters:	`fileID: ID of the file you want to find the file type of`
Returns:	`(uint16_t) file type` *or* `a general error (see defined literals)`
Usage:	`ftype = myFlash.getFileType(4); // Get the file type for file ID #4`

getFileSize(fileID);

Get the size of a file.

Parameters:	`fileID: ID of the file you want to find the file size of`
Returns:	`(uint32_t) file size in bytes` *or* `a getFileSize() specific error (see defined literals)`
Usage:	`fsize = myFlash.getFileSize(0); // Get the file size for file ID #0`

readFileNote(fileID, buffer);

Get the note associated with a file.

Parameters:	`fileID: ID of the file you want to get the file note for`
	`buffer: Buffer to store the file note in`
Returns:	`(uint16_t) ERR_NO_ERROR (0)` *or* `a general error (see defined literals)`
Usage:	`result = myFlash.readFileNote(100, buf); // Get the file note for file ID #100 and store it in buf`
Notes:	The buffer *must* be at least 17 bytes long. File notes can be up to 16 characters and will be terminated with a string terminator (0 byte).

getImageXSize(fileID);

Get the X size of an image.

Parameters:	`fileID: ID of the file containing the image you want to get the X size for`
Returns:	`(uint16_t) X size in pixels` *or* `a general error (see defined literals)`
Usage:	`Xsize = myFlash.getImageXSize(100); // Get the X size for file ID #100`
Notes:	This function will return **ERR_FILETYPE_INCORRECT** if you try to get the size of a non-image file.

getImageYSize(fileID);

Get the Y size of an image.

Parameters:	`fileID: ID of the file containing the image you want to get the Y size for`
Returns:	`(uint16_t) Y size in pixels` *or* `a general error (see defined literals)`
Usage:	`Ysize = myFlash.getImageYSize(100); // Get the Y size for file ID #100`
Notes:	This function will return **ERR_FILETYPE_INCORRECT** if you try to get the size of a non-image file.

getAudioSamplerate(fileID);

Get the sample rate for an audio sample.

Parameters:	`fileID: ID of the file containing the audio sample you want the sample rate for`
Returns:	`(uint16_t) sample rate in Hz` *or* `a general error (see defined literals)`
Usage:	`bitrate = myFlash.getAudioSamplerate(14); // Get the sample rate for file ID #14`
Notes:	This function will return **ERR_FILETYPE_INCORRECT** if you try to get the size of a non-audio file.

getAudioBPS(fileID);

Get the bits per sample for an audio sample.

Parameters:	`fileID: ID of the file containing the audio sample you want the bits per sample for`
Returns:	`(uint16_t) bits per sample` *or* `a general error (see defined literals)`
Usage:	`bps = myFlash.getAudioBPS(14); // Get the bits per sample for file ID #14`
Notes:	`This function will return` **`ERR_FILETYPE_INCORRECT`** `if you try to get the size of a non-audio file.`

getAudioChannels(fileID);

Get the number of channels for an audio sample.

Parameters:	`fileID: ID of the file containing the audio sample you want the number of channels for`
Returns:	`(uint16_t) number of channels` *or* `a general error (see defined literals)`
Usage:	`channels = myFlash.getAudioChannels(14); // Get the number of channels for file ID #14`
Notes:	`This function will return` **`ERR_FILETYPE_INCORRECT`** `if you try to get the size of a non-audio file.`

readTextResource(fileID, resID, buffer, buffersize);

Get a string from a text resource file.

Parameters:	`fileID:     ID of the file you want to get the string from`
	`resID:      Resource ID within the file`
	`buffer:     Buffer to put the read data into`
	`buffersize: Size of the buffer in bytes`

Returns:	`(uint16_t) ERR_NO_ERROR (0)` *or* `a general error (see defined literals)`
Usage:	`result = myFlash.readTextResource(2, 4, buf, sizeof(buf)); // Read string #4 from file #2`

Note:	`Text Resource files can only be accessed with this function.`
	`If the buffer is too small for contain the entire string the result will be ERR_BUFFER_OVERFLOW.`
	`Remember that strings are terminated with a 0 byte so the buffer should be at least 1 byte longer than the expected length of the text.`