SPI Storm Studio

C library user's guide

Byte Paradigm – info@byteparadigm.com

Table of Contents

References

[]

History

Version	Date	Description
1.00	June, 2nd, 2011	Initial revision
1.02	Dec, 16th, 2011	Updated spis_ScanDev function for SPI Storm Studio 1.1.8
1.03	Jul 9th, 2012	Added version function
1.04	Jan 8th, 2013	Added error code
1.05	Oct 17th, 2014	Re-published document

1 Introduction

SpiStorm.dll is a C DLL used in SPI Storm Studio software. This library contains the functions required to configure and control SPI Storm device.

Users wishing to develop their own interface application, automate tasks or integrate SPI Storm control within other environments are enabled to do so with simple C function calls.

This user's guide describes the functions that are made available in SpiStorm.dll library.

2 SPIStorm.dll Library Description

2.1 Functions Quick Reference Table

Table 1 gives the list of the functions callable from SPIStorm.dll.

Function prototype	Description
int spis_CreateInstance(void);	Creates an instance of the SPI Storm library.
void spis_DeleteInstance(int Handle);	Deletes an instance of the SPI Storm library.
void spis_SelectInstance(int Handle);	Selects an instance of the SPI Storm library.
int spis_ScanDev(unsigned char *pType, unsigned int *pID, unsigned char *pSerNum, bool *pInUse);	Scans the USB bus for available SPI Storm devices.
int spis_Connect(char *pSerNum, unsigned short SupplyVoltage);	Connects to an available SPI Storm device.
int spis_Disconnect(void);	Disconnects from an SPI Storm device.
int spis_SetDisconnectCallback(void *pObj, void *pFct);	Defines a callback for the device disconnection.
int spis_LoadPrjFile(char *pFileName, bool CheckSyntax, bool SetInitial);	Loads a SPI Storm project file.
int spis_ExecProg(bool Blocking);	Executes a program.
int spis_ExecProgBuf(char **pBufOut, char **pBufIn, unsigned int NrBuf, bool Blocking);	Executes a program by passing used buffers.
int spis_ExecMacro(char *pLabel,char *pBufOut, char *pBufIn);	Executes a macro.
int spis_StartSequence(void);	Starts burst of macros.
int spis_EndOfSequence(void);	Finalizes a bursts of macros.
int spis_Abort(void);	Aborts a running program or (burst of) macro(s).
int spis_GetState(void);	Retrieves the current state of the SPI Storm device.
int spis_SetStateCallback(void *pObj, void *pFct);	Defines a callback for the systems status.
int spis_SetSysErrCallback(void *pObj, void *pFct);	Defines a callback for a system error.
int spis_Version(unsigned char *pMajor, unsigned char *pMinor, unsigned short *pPatch)	Returns the DLL version.

Table 1: Functions quick reference

2.2 Functions Details

int spis_CreateInstance(void);

Parameters: none
Returns: A handle to the initialized library instance.
Description: Creates an SPI Storm instance and initializes the SPI Storm library. This function must be called a first time before any other function, to ensure proper operation of the library.
A new SPI Storm library instance is created every time the function is called. The user application can select between the created instances with the spis_SelectInstance function. Multiple instances are useful when controlling multiple SPI Storm devices from one user application.

void spis_DeleteInstance(int Handle);

Parameters: **Handle** : Handle to an instance of a previously created library instance.
Returns: -
Description: Deletes the library instance corresponding to the supplied handle. The library instance must first be created with spis_CreateInstance.

void spis_SelectInstance(int Handle);

Parameters: **Handle** : Handle to an instance of a previously created library instance.
Returns: -
Description: Selects the library instance corresponding to the supplied handle. The library instance must first be created with spis_CreateInstance.

int spis_ScanDev(unsigned char *pType, unsigned int *pID, unsigned char *pSerNum, bool *pInUse);

Parameters: **pType** : Array receiving the type of connection for each SPIStorm device. The type is always USB and is equal to 0.
pID : Array receiving the vendor ID and product ID for each SPIStorm device. The vendor ID is always 0x1CC4 and product ID is always 0x0401.
pSerNum : Array receiving the serial number for each SPIStorm device. Each serial number is 11 bytes wide.
pInUse : Array receiving the connection status of the device. If true, the device is already in use and is not available for connection.
Returns: The number of SPIStorm devices found when the value is between 0 and 127, other values correspond with one of the return codes described in paragraph 2.3.
Description: Scans the USB bus for connected SPIStorm devices. The three parameters are pointers to three buffers.

The connection type is defined by one char (= 1 byte), hence the minimum size of the first buffer must be equal the the number of connected SPIStorm devices times one byte.

The ID is defined by one int (= 4 bytes), hence the minimum size of the second buffer must be equal the the number of connected SPIStorm devices times four bytes.

The serial number is defined by eleven bytes, hence the minimum size of the last buffer must be equal the the number of connected SPIStorm devices times eleven bytes.

The maximum number of devices allowed on a USB bus is 127. Hence, allocating respectively 1*127 bytes, 4*127 bytes and 11*127 bytes for pType, pID and pSerNum will always work.

The function returns the number of SPIStorm devices found. This also indicates the “fill level” of the three parameters. If 3 devices were found, the buffer will respectively contain 3 bytes, 12 bytes and 33 bytes of valid data.

int spis_Connect(char *pSerNum, unsigned short SupplyVoltage);

Parameters: **pSerNum** : Pointer to an eleven byte buffer containing the serial number of the SPIStorm device to connect to.

SupplyVoltage : Defines the user interface supply voltages. Following values are valid:

- 3300 : supply voltage between 3.30V and 2.91V
- 2500 : supply voltage between 2.90V and 2.16V
- 1800 : supply voltage between 2.15V and 1.66V
- 1500 : supply voltage between 1.65V and 1.39V
- 1250 : supply voltage between 1.38V and 1.25V

Returns: See return codes paragraph 2.3.

Description: Connects to the SPIStorm device defined by the serial number. During connection, the device is loaded with its configuration defined by the supply voltage.

int spis_Disconnect(void);

Parameters: none

Returns: See return codes paragraph 2.3.

Description: Disconnects the library from the SPIStorm device.

int spis_SetDisconnectCallback(void *pObj, void *pFct);

Parameters: **pObj** : Pointer to an instance of a user object.

pFct : Callback function.

Returns: See return codes paragraph 2.3.

Description: Defines a function to be called on the device disconnection. The callback prototype is: void DisconnectCallback(void *pObj)

int spis_LoadPrjFile(char *pFileName, bool CheckSyntax, bool SetInitial);

Parameters: **pFileName** : Pointer to a string containing the file name.

CheckSyntax : Forces to execute a syntax check only, the project file isn't be loaded. To load the project file, this field must be set to false.

SetInitial : Sets the SPIStorm device to its initial state after having loaded the project file. This field is only used when CheckSyntax = false.

Returns: See return codes paragraph 2.3.

Description: Loads a project file. This function must always be called at least once before executing a program or a macro.

int spis_ExecProg(bool Blocking);

Parameters: **Blocking** : When set to true, the function call is blocking till the end of the program execution.

Returns: See return codes paragraph 2.3.

Description: Executes the program as defined in project file. This is fully equivalent to pressing the “Run” button in the graphical user interface.

int spis_ExecProgBuf(char **pBufOut, char **pBufIn, unsigned int NrBuf, bool Blocking);

Parameters: **pBufOut** : Pointer to array of pointers containing the output buffers.

pBufInf : Pointer to array of pointers containing the input buffers.

NrBuf : Number of input/output buffers.

Blocking : The function call is blocking till the end of the program execution if true.

Returns: See return codes paragraph 2.3.

Description: Executes the program defined in the project file, using user provided buffers. A program is an assembly of macros. A macro uses a write and read buffer to send and receive data. A macro can send and receive data simultaneously. Every macro needs a write and a read buffer, hence the length of the array of pointers must be equal to the number of macros executed.

int spis_ExecMacro(char *pLabel,char *pBufOut, char *pBufIn);

Parameters: **pLabel** : Pointer to a string containing a macro label.

pBufOut : Pointer to a buffer containing the data to be sent.

pBufIn : Pointer to a buffer where the received data will be stored.

Returns: See return codes paragraph 2.3.

Description: Executes a macro selected by pLabel. The label must match one of the macros defined in the project file. The function sends data to the device stored in pBufOut and stores the data received from the

device in pBufIn. The output buffer size will for example be three bytes if the macro outputs 17 bits data bits. Identically, the input buffer size will for example be 2 bytes if the macro reads 9 bits data.

int spis_StartSequence(void);

Parameters: none
Returns: See return codes paragraph 2.3.
Description: Marks the beginning of a burst transfer of macros. A burst always starts with a call to spis_StartSequence, one or more calls to spis_ExecMacro and a call to spis_EndOfSequence. The burst transfer has the advantage that the macros are grouped when sent to the SPIStorm device. This is in opposition with a simple macro executions where every macro is executed one after the other with a higher latency.

int spis_EndOfSequence(void);

Parameters: none
Returns: See return codes paragraph 2.3.
Description: Marks the end of a burst transfer of macros. The execution of macros will only start after spis_EndOfSequence is called. All macros are accumulated before this function is called to optimize the burst transfer.

int spis_Abort(void);

Parameters: none
Returns: See return codes paragraph 2.3.
Description: Aborts a running job.

int spis_GetState(void);

Parameters: none
Returns: The current state of SPIStorm. Following states are defined:

- 0x00 : Idle
- 0x01 : Initialising (configuring the library for a program or macro execution)
- 0x02 : Pre-loading (the device for execution)
- 0x03 : SPI waiting trigger
- 0x04 : SPI running
- 0x30 : GPO waiting trigger
- 0x40 : GPO running
- 0x05 : Receiving SPI data
- 0x06 : Done
- 0x07 : Aborted

Description: Retrieves the current state of the SPIStorm device. Note that the low and high nibble of the state can be combined. A state of 0x45 means that the SPI transfers are done but that SPI data is still being sent to the host PC, while the GPO is still running.

int spis_SetStateCallback(void *pObj, void *pFct);

Parameters: **pObj** : Pointer to an instance of a user object.
pFct : Callback function.
Returns: See return codes paragraph 2.3.
Description: Defines a function to be called on a state change (see spis_GetState for the existing states).
The callback prototype is: void StateCallback(void *pObj, unsigned int State)

int spis_SetSysErrCallback(void *pObj, void *pFct);

Parameters: **pObj** : Pointer to an instance of a user object.
pFct : Callback function.
Returns: See return codes paragraph 2.3.
Description: Defines a function to be called when a system error occurs.
The callback prototype is: void SysErrCallback(void *pObj, unsigned int Err)
This function must be called before connecting to a device.

int spis_Version(unsigned char *pMajor, unsigned char *pMinor, unsigned short *pPatch)

Parameters: **pMajor** : pointer to major version number
pMinor : pointer to minor version number
pPatch : pointer to patch version number

Returns: See return codes paragraph 2.3.

Description: Returns SPI Storm Studio's DLL version.

2.3 Functions Return Codes

Return code	Meaning	Action required
0x00000000	No error	
USB Device Driver		
0x80000002	No valid USB device found	Check if your SPI Storm device is connected to the USB port.
0x80000004	Failed to open USB driver	Check if you have properly installed the USB driver. Check if your SPI Storm device is properly connected to the USB port.
0x80050001	Failed to load bin file	Check if your working directory contains all the *.bin files provided with SPI Storm Studio.
Licensing		
0x00080001	No valid license	1) Check if you have received your license file. If you don't have it, mailto: support@byteparadigm.com to request your license file. You MUST provide your SPI Storm unit serial number. It is located at the back of your device. 2) Install your license with SPI Storm Studio GUI. Please click here to know how to install license. 3) If you still receive once of these error codes, please contact Byte Paradigm support (support@byteparadigm.com).
0x00080002	Invalid license features	
0x00080003	License not found in license file	
0x80080001	Failed to open license file	
0x80080002	License decoding failed	
0x80080003	Failed to find license section	
Logging		
0x80200002	Log file not opened	Soft cannot create log file in roaming directory. Check users directory properties: **Windows 7 users**: the roaming directory is located here: **C:\Users\<user>\AppData\Roaming\ByteParadigm\SpiStormStudio** Windows XP users: the 'roaming directory' is located here: **C:\Documents and Settings\<user>\Application Data\ByteParadigm\SpiStormStudio** Contact your IT manager to set your permissions properly.
Configuration File		
0x80400001	Failed to open the project file	The project file is locked or does not exist. Check status.
0x80400002	Failed to create project file log	Check if the log file destination is accessible.
0x80400003	Project file contains errors	Check project file syntax.
0x80400004	Project file not defined	Specified project file is NULL. Please correct.
0x80400005	Project file name exceeds 256 characters	Please shorten project file name (including path).
Application		
0x80500001	No device connected	You attempted to execute an operation with SPI Storm device be you failed to connect it properly. Please first use spis_Connect function to connect a device.
0x80500002	Application already connected to a device	You already connected a device. If you want to connect another device, disconnect it and connect the new device. Use spis_Connect to connect and spis_Disconnect functions.
0x80500003	Unable to scan the USB bus while connected to a device	Disconnect device first (spis_Disonnect) and then scan.
0x80500004	No program defined	You attempted to run a program whereas no program is defined.

0x80500005	Program or macro already running	You attempted to run a program that is already running.
C API		
0x80900001	Unable to find instance on C API	You called a function without having created an instance first. Please use spis_CreateInstance first.
0x80900002	Not enough user buffers provided	There is a mismatch between the number of buffers that you provided when calling ExecProg* and the number of buffer strictly needed for this.
0x80900003	Too many user buffers provided	
0x80900004	Macro reference not found	You called a macro that does not exist. Please check your macro call name and parameters.
0x80900005	Already in a sequence	You called spis_StartSequence multiple times.
0x80900006	Not in a sequence	You called spis_EndSequence without first calling spis_StartSequence.

If you receive an error code that is not listed above, please contact Byte Paradigm support to report it:
support@byteparadigm.com