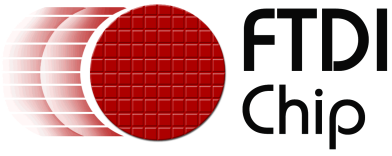

Future Technology Devices International Ltd.

FTCSPI Programmer's Guide

Table of Contents

1 Welcome to the FTCSPI Programmer's Guide

The FT2232C device contains FTDI's multi-protocol synchronous serial engine (MPSSE) controller which may be used to interface to many popular synchronous serial protocols including SPI, JTAG and I2C.

The FTCSPI DLL has been created to allow application developers to use the FT2232C to create a USB to Serial Peripheral Interface (SPI) protocol interface without any knowledge of the MPSSE command set. All of the functions in FTCSPI.DLL can be replicated using calls to FTD2XX.DLL and sending the appropriate commands to the MPSSE as per application note AN2232C-01 Command Processor For MPSSE and MCU Host Bus Emulation Modes.

The latest version of FTDI's FTCD2XX drivers must be installed to use FTCSPI.DLL as several calls are made to a new version of FTD2XX.DLL. The MPSSE is available through channel A of the FT2232C device only; channel B does not support the MPSSE. Channel B may be controlled independently using FTDI's FTCD2XX drivers while channel A is being used for SPI communication.

This document lists all of the functions available in FTCSPI.DLL.

The FTCSPI DLL can be downloaded from the FTCSPI page of the MPSSE section in the Projects area of the site.

2 SPI Interface Functions

The functions listed in this section can be used to set up the FT2232C for SPI communication, write data to an external device using SPI and read data from an external device using SPI.

Please note that the latest version of FTDI's FTCD2XX drivers must be installed to use the FTCSPI DLL for SPI communication.

2.1 SPI_GetNumDevices

Returns the number of available FT2232C devices connected to a system.

FTC_STATUS **SPI_GetNumDevices** (*lpdwNumDevices*)

Parameters

lpdwNumDevices	Pointer to a variable of type DWORD which receives the actual number of available FT2232C devices connected to a system

Return Value

FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_IO_ERROR

Remarks

This function can be used to provide the maximum index for using with SPI_GetDeviceNameLocID 5.

Example

```
FTC_STATUS Status = FTC_SUCCESS;
DWORD dwNumDevices = 0;

Status = SPI_GetNumDevices(&dwNumDevices);
```

2.2 SPI_GetDeviceNameLocID

Returns the device name and location ID of an available FT2232C device.

FTC_STATUS **SPI_GetDeviceNameLocID** (DWORD *dwDeviceNameIndex*, LPSTR *lpDeviceNameBuffer*, DWORD *dwBufferSize*, LPDWORD *lpdwLocationID*)

Parameters

dwDeviceNameIndex	Index of the FT2232C device.
lpDeviceNameBuffer	Pointer to buffer that receives the device name of the specified FT2232C device connected to a system. The string will be NULL terminated.
dwBufferSize	Length of the buffer created for the device name string. Set buffer length to a minimum of 50 characters.
lpdwLocationID	Pointer to a variable of type DWORD which receives the location identifier of the specified FT2232C device.

Return Value

FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_DEVICE_NOT_FOUND
FTC_INVALID_DEVICE_NAME_INDEX
FTC_NULL_ DEVICE_NAME_BUFFER_POINTER
FTC_ DEVICE_NAME_BUFFER_TOO_SMALL
FTC_IO_ERROR

Remarks

The SPI_GetNumDevices 4 function can be used to obtain the number of available FT2232C devices connected to a system. The device index is 0 based. The information returned from this function can be used with SPI_OpenEx 7 to open a specific device.

Example

```
FTC_STATUS Status = FTC_SUCCESS;
char szDeviceName[100];
DWORD dwLocationID = 0;

Status = SPI_GetDeviceNameLocID(0, szDeviceName, 50, &dwLocationID);
```

2.3 SPI_Open

Open the device and return a handle that will be used for subsequent accesses. This function can only be used when there is a single FT2232C device connected.

FTC_STATUS **SPI_Open** (FTC_HANDLE **pftHandle*)

Parameters

**pftHandle*	Pointer to a variable of type FTC_HANDLE where the handle to the open device will be returned. This handle must then be used in all subsequent calls to access this device.

Return Value

FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_DEVICE_NOT_FOUND
FTC_DEVICE_IN_USE
FTC_TOO_MANY_DEVICES
FTC_FAILED_TO_SYNCHRONIZE_DEVICE_MPSSE
FTC_FAILED_TO_COMPLETE_COMMAND
FTC_IO_ERROR
FTC_INSUFFICIENT_RESOURCES

Remarks

If more than one device is attached, this function will return an error code. For multiple devices, use SPI_GetDeviceNameLocID 5 and then SPI_OpenEx 7 instead.

Example

```
FTC_STATUS Status = FTC_SUCCESS;
FTC_HANDLE ftHandle;

Status = SPI_Open(&ftHandle);
```

2.4 SPI_OpenEx

Open the specified device and return a handle that will be used for subsequent accesses. The device must be specified by its device description and location.

FTC_STATUS **SPI_OpenEx** (LPSTR *lpDeviceName*, DWORD *dwLocationID*, FTC_HANDLE **pftHandle*)

Parameters

lpDeviceName	Pointer to a NULL terminated string that contains the name of the specified FT2232C device to be opened.
dwLocationID	Specifies the location identifier of the specified FT2232C device to be opened.
**pftHandle*	Pointer to a variable of type FTC_HANDLE where the handle to the open device will be returned. This handle must then be used in all subsequent calls to access this device.

Return Value

FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_NULL_DEVICE_NAME_BUFFER_POINTER
FTC_INVALID_DEVICE_NAME
FTC_INVALID_LOCATION_ID
FTC_DEVICE_NOT_FOUND
FTC_DEVICE_IN_USE
FTC_FAILED_TO_SYNCHRONIZE_DEVICE_MPSSE
FTC_FAILED_TO_COMPLETE_COMMAND
FTC_IO_ERROR
FTC_INSUFFICIENT_RESOURCES

Remarks

The device name and location ID parameters are returned from the SPI_GetDeviceNameLocID 5 function.

Example

```
FTC_STATUS Status = FTC_SUCCESS;
char szDeviceName[100];
DWORD dwLocationID = 0;
FTC_HANDLE ftHandle;

Status = SPI_OpenEx(szDeviceName, dwLocationID, &ftHandle);
```

2.5 SPI_Close

Close an open device.

FTC_STATUS **SPI_Close** (FTC_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_INVALID_HANDLE
FTC_IO_ERROR

Example

```
FTC_STATUS Status = FTC_SUCCESS;
FTC_HANDLE ftHandle;

Status = SPI_Close(ftHandle);
```

2.6 SPI_InitDevice

Initialise the device.

FTC_STATUS **SPI_InitDevice** (FTC_HANDLE *ftHandle*, DWORD *dwClockDivisor*)

Parameters

ftHandle	Handle of the device.
dwClockDivisor	Specifies a clock divisor which will be used to set the frequency for clocking data in and out of the FT2232C device.

Return Value

FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_INVALID_HANDLE
FTC_INVALID_CLOCK_DIVISOR
FTC_FAILED_TO_SYNCHRONIZE_DEVICE_MPSSE
FTC_FAILED_TO_COMPLETE_COMMAND
FTC_IO_ERROR
FTC_INSUFFICIENT_RESOURCES

Remarks

This function initializes the FT2232C device, by carrying out the following:

- resets the device and purge device USB input buffer
- sets the device USB input and output buffers to 64K bytes
- sets the special characters for the device, disable event and error characters
- sets the device read timeout to infinite
- sets the device write timeout to 5 seconds
- sets the device latency timer to 16 milliseconds
- reset MPSSE controller
- enable MPSSE controller
- synchronize the MPSSE
- set the 8 general purpose pins to output mode and set their output states to high
- set data in and data out clock frequency
- set MPSSE loopback state to off (default)

The valid range for *dwClockDivisor* is 0 to 65535. The highest clock frequency is represented by 0, which is equivalent to 6MHz and the lowest clock frequency is represented by 65535, which is equivalent to 91Hz. This can be calculated using the following formula:

*Clock Frequency = 12MHz / ((1 + dwClockDivisor) * 2)*

Example

```
FTC_STATUS Status = FTC_SUCCESS;
FTC_HANDLE ftHandle;
```

```
DWORD dwClockDivisor = 0;
Status = SPI_InitDevice(ftHandle, dwClockDivisor);
```

2.7 SPI_GetClock

Calculates the actual frequency in Hz for a given clock divisor value.

FTC_STATUS **SPI_GetClock** (DWORD *dwClockDivisor*, LPDWORD *lpdwClockFrequencyHz*)

Parameters

dwClockDivisor	Specifies a clock divisor which will be used to set the frequency for clocking data in and out of the FT2232C device.
lpdwClockFrequencyHz	Pointer to a variable of type DWORD which receives the actual frequency in Hz that data will be clocked in and out of the FT2232C device at.

Return Value

FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_INVALID_CLOCK_DIVISOR

Remarks

The valid range for *dwClockDivisor* is 0 to 65535. The highest clock frequency is represented by 0 which is equivalent to 6MHz and the lowest clock frequency is represented by 65535 which is equivalent to 91Hz. This can be calculated using the following formula:

*dwClockFrequency = 12MHz / ((1 + dwClockDivisor) * 2)*

The clock frequency can be set by passing the clock divisor value to SPI_SetClock 12 or SPI_InitDevice 9.

Example

```
FTC_STATUS Status = FTC_SUCCESS;
DWORD dwClockDivisor = 0;
DWORD dwClockFrequencyHz = 0;

Status = SPI_GetClock(dwClockDivisor, &dwClockFrequencyHz);
```

2.8 SPI_SetClock

Sets the clock divisor value and returns the clock frequency in Hz.

FTC_STATUS **SPI_SetClock** (FTC_HANDLE *ftHandle*, DWORD *dwClockDivisor,* LPDWORD *lpdwClockFrequencyHz*)

Parameters

ftHandle	Handle of the device.
dwClockDivisor	Specifies a clock divisor which will be used to set the frequency for clocking data in and out of the FT2232C device.
lpdwClockFrequencyHz	Pointer to a variable of type DWORD which receives the actual frequency in Hz that data will be clocked in and out of the FT2232C device at.

Return Value

FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_INVALID_HANDLE
FTC_INVALID_CLOCK_DIVISOR
FTC_FAILED_TO_COMPLETE_COMMAND
FTC_IO_ERROR

Remarks

The valid range for *dwClockDivisor* is 0 to 65535. The highest clock frequency is represented by 0 which is equivalent to 6MHz and the lowest clock frequency is represented by 65535 which is equivalent to 91Hz. This can be calculated using the following formula:

*dwClockFrequency = 12MHz / ((1 + dwClockDivisor) * 2)*

SPI_SetClock 12 will return the actual frequency in Hz for the current divisor value. The clock frequancy in Hz can also be calculated using SPI_GetClock 11.

Example

```
FTC_STATUS Status = FTC_SUCCESS;
FTC_HANDLE ftHandle;
DWORD dwClockDivisor = 0;
DWORD dwClockFrequencyHz = 0;

Status = SPI_SetClock(ftHandle, dwClockDivisor, &dwClockFrequencyHz);
```

2.9 SPI_SetLoopback

Enables or disables loop back mode.

FTC_STATUS **SPI_SetLoopback** (FTC_HANDLE *ftHandle*, BOOL *bLoopbackState*)

Parameters

ftHandle	Handle of the device.
bLoopbackState	Controls the state of the FT2232C device loopback, to turn loopback on (TRUE) or off (FALSE).

Return Value

FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_INVALID_HANDLE
FTC_FAILED_TO_COMPLETE_COMMAND
FTC_IO_ERROR

Remarks

Loop back mode will simply return any data written to the device. Loop back mode can be enabled by setting *bLoopbackState* to true, or disabled by setting *bLoopbackState* to false. The default state for the loop back mode is disabled.

Example

```
FTC_STATUS Status = FTC_SUCCESS;
FTC_HANDLE ftHandle;

Status = SPI_SetLoopback(ftHandle, true);
```

2.10 SPI_GetGPIOs

Reads the values of the 4 upper general purpose input/output pins of the FT2232C device.

FTC_STATUS **SPI_GetGPIOs** (FTC_HANDLE *ftHandle*, PFTC_LOW_HIGH_PINS *pHighPinsInputData*)

Parameters

ftHandle	Handle of the device.
pHighPinsInputData	Pointer to a structure that contains the value of the upper 4 GPIO pins of the FT2232C device.

Return Value

FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_INVALID_HANDLE
FTC_NULL_INPUT_OUTPUT_BUFFER_POINTER
FTC_FAILED_TO_COMPLETE_COMMAND
FTC_IO_ERROR

Remarks

This function allows the state of the 4 upper general purpose input/output (GPIO) pins to be read as either high or low. The pins can be configured as input, low output or high output using the SPI_SetGPIOs 15 function.

The definition of the FTC_LOW_HIGH_PINS structure is given in the Appendix 31.

Example

```
FTC_STATUS Status = FTC_SUCCESS;
FTC_HANDLE ftHandle;
FTC_LOW_HIGH_PINS HighPinsInputData;

Status = SPI_GetGPIOs(ftHandle, &HighPinsInputData);
```

2.11 SPI_SetGPIOs

Controls the use of the 4 upper general purpose input/output pins of the FT2232C device.

FTC_STATUS **SPI_SetGPIOs** (FTC_HANDLE *ftHandle*, PFTC_CHIP_SELECT_PINS *pChipSelectsDisableStates*, PFTC_INPUT_OUTPUT_PINS *pHighInputOutputPinsData*)

Parameters

ftHandle	Handle of the device.
pChipSelectsDisableStates	Pointer to a structure that contains the disable states for the 5 chip select pins of the FT2232C.
pHighInputOutputPinsData	Pointer to a structure that contains the data that is used to control the upper 4 GPIO pins of the FT2232C.

Return Value

FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_INVALID_HANDLE
FTC_NULL_INPUT_OUTPUT_BUFFER_POINTER
FTC_FAILED_TO_COMPLETE_COMMAND
FTC_IO_ERROR

Remarks

This function provides complete control over the state of the 4 upper general purpose input/output (GPIO) pins. They can be configured to be inputs, low outputs or high outputs. The state of the pins can be read using the SPI_GetGPIOs 14 function.

The definitions of FTC_CHIP_SELECT_PINS and FTC_INPUT_OUTPUT_PINS structures are given in the Appendix 31.

Example

```
FTC_STATUS Status = FTC_SUCCESS;
FTC_HANDLE ftHandle;
FTC_CHIP_SELECT_PINS ChipSelectsDisableStates;
FTC_INPUT_OUTPUT_PINS HighInputOutputPins;

ChipSelectsDisableStates.bADBUS3ChipSelectPinState = false;
ChipSelectsDisableStates.bADBUS4GPIOL1PinState = false;
ChipSelectsDisableStates.bADBUS5GPIOL2PinState = false;
ChipSelectsDisableStates.bADBUS6GPIOL3PinState = false;
ChipSelectsDisableStates.bADBUS7GPIOL4PinState = false;

HighInputOutputPins.bPin1InputOutputState = true;
HighInputOutputPins.bPin1LowHighState = false;
HighInputOutputPins.bPin2InputOutputState = true;
HighInputOutputPins.bPin2LowHighState = false;
HighInputOutputPins.bPin3InputOutputState = true;
HighInputOutputPins.bPin3LowHighState = false;
HighInputOutputPins.bPin4InputOutputState = true;
HighInputOutputPins.bPin4LowHighState = false;
```

```
Status = SPI_SetGPIOs(ftHandle, &ChipSelectsDisableStates, &HighInputOutputPins);
```

2.12 SPI_Write

Write data from the FT2232C to an external device using the SPI protocol.

FTC_STATUS **SPI_Write** (FTC_HANDLE *ftHandle*, PFTC_INIT_CONDITION *pWriteStartCondition*, BOOL *bClockOutDataBitsMSBFirst*, BOOL *bClockOutDataBitsPosEdge*, DWORD *dwNumControlBitsToWrite*, PWriteControlByteBuffer *pWriteControlBuffer*, DWORD *dwNumControlBytesToWrite*, BOOL *bWriteDataBits*, DWORD *dwNumDataBitsToWrite*, PWriteDataByteBuffer *pWriteDataBuffer*, DWORD *dwNumDataBytesToWrite*, PFTC_WAIT_DATA_WRITE *pWaitDataWriteComplete*, PFTC_HIGHER_OUTPUT_PINS *pHighPinsWriteActiveStates*)

Parameters

ftHandle	Handle of the device.
pWriteStartCondition	Pointer to the structure that contains the start output states (low or high) of the clock, data out and signal out/chip select pins of the FT2232C.
bClockOutDataBitsMSBFirst	Clock out control and data bits most significant bit (MSB) first (TRUE), clock out control and data bits least significant bit (LSB) first (FALSE).
bClockOutDataBitsPosEdge	Clock out control and data bits on positive clock edge (TRUE), clock out control and data bits on negative clock edge (FALSE).
dwNumControlBitsToWrite	Specifies the number of control bits to be written to an external device. Valid range 2 to 2040. 2040 bits is equivalent to 255 bytes.
pWriteControlBuffer	Pointer to buffer that contains the control data to be written to an external device. Listed below are three examples of control and address bytes: Two Control bytes Control Address byte 1, Control Address byte 2 Two Control bytes, Control Address byte 1, Control Address byte 2
dwNumControlBytesToWrite	Specifies the number of control bytes in the write control buffer, which contains all the specified bits to be written to an external device. Valid range 1 to 255 bytes.
bWriteDataBits	Write data bits to an external device (TRUE), do not write any data bits to an external device (FALSE).
dwNumDataBitsToWrite	Specifies the number of data bits to be written to an external device. Valid range 2 to 524280. 524280 bits is equivalent to 64K bytes.
pWriteDataBuffer	Pointer to buffer that contains the data to be written to an external device.
dwNumDataBytesToWrite	Specifies the number of data bytes in the write data buffer, which contains all the specified bits to be written to an external device. Valid range 1 to 65535 (i.e. 64K bytes).
pWaitDataWriteComplete	Pointer to the structure that contains data that controls whether the FT2232C should wait until all data bytes have been written to an external device before returning.
pHighPinsWriteActiveStates	Pointer to the structure that contains which of the 4 upper general purpose input/output pins of a FT2232C, are to be used during a write to an external device. Each GPIO pin that is to be used during a write to an external device must have been previously configured as an output pin (see SPI_SetGPIOs 15).

Return Value

FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_INVALID_HANDLE
FTC_NULL_INITIAL_CONDITION_BUFFER_POINTER
FTC_INVALID_NUMBER_CONTROL_BITS
FTC_NULL_WRITE_CONTROL_BUFFER_POINTER
FTC_INVALID_NUMBER_CONTROL_BYTES
FTC_NUMBER_CONTROL_BYTES_TOO_SMALL
FTC_INVALID_NUMBER_WRITE_DATA_BITS
FTC_NULL_WRITE_DATA_BUFFER_POINTER

FTC_INVALID_NUMBER_WRITE_DATA_BYTES
FTC_NUMBER_WRITE_DATA_BYTES_TOO_SMALL
FTC_NULL_WAIT_DATA_WRITE_BUFFER_POINTER
FTC_NULL_OUTPUT_PINS_BUFFER_POINTER
FTC_INVALID_INIT_CLOCK_PIN_STATE
FTC_INVALID_FT2232C_CHIP_SELECT_PIN
FTC_INVALID_FT2232C_DATA_WRITE_COMPLETE_PIN
FTC_DATA_WRITE_COMPLETE_TIMEOUT
FTC_INVALID_CONFIGURATION_HIGHER_GPIO_PIN
FTC_FAILED_TO_COMPLETE_COMMAND
FTC_IO_ERROR

Remarks

This function will write data from the FT2232C to an external device using the SPI protocol. The data will be clocked at a rate specified by the clock divisor set by calling either the SPI_InitDevice 9 or SPI_SetClock 12 functions.

The init condition, write control buffer, write data buffer, wait data write complete and high pins write active states definitions are given in the Appendix 31.

Example

```
#define NUM_93LC56B_CMD_CONTOL_BITS 11
#define NUM_93LC56B_CMD_CONTOL_BYTES 2

const SPI_EWEN_CMD = '\x9F'; // set up write enable command

FTC_STATUS Status = FTC_SUCCESS;
FTC_HANDLE ftHandle;
FTC_INIT_CONDITION WriteStartCondition;
WriteControlByteBuffer WriteControlBuffer;
WriteDataByteBuffer WriteDataBuffer;
FTC_WAIT_DATA_WRITE WaitDataWriteComplete;
FTC_HIGHER_OUTPUT_PINS HighPinsWriteActiveStates;

WriteStartCondition.bClockPinState = false;
WriteStartCondition.bDataOutPinState = false;
WriteStartCondition.bChipSelectPinState = false;
WriteStartCondition.dwChipSelectPin = ADBUS3ChipSelect;

WaitDataWriteComplete.bWaitDataWriteComplete = false;

HighPinsWriteActiveStates.bPin1ActiveState = false;
HighPinsWriteActiveStates.bPin1State = false;
HighPinsWriteActiveStates.bPin2ActiveState = false;
HighPinsWriteActiveStates.bPin2State = false;
HighPinsWriteActiveStates.bPin3ActiveState = false;
HighPinsWriteActiveStates.bPin3State = false;
HighPinsWriteActiveStates.bPin4ActiveState = false;
HighPinsWriteActiveStates.bPin4State = false;

// enable writing
WriteControlBuffer[0] = SPI_EWEN_CMD;
WriteControlBuffer[1] = '\xFF';

Status = SPI_Write(ftHandle, &WriteStartCondition, true, false,
NUM_93LC56B_CMD_CONTOL_BITS, &WriteControlBuffer, NUM_93LC56B_CMD_CONTOL_BYTES, false, 0,
&WriteDataBuffer, 0, &WaitDataWriteComplete, &HighPinsWriteActiveStates);
```

2.13 SPI_Read

Read data from an external device to the FT2232C using the SPI protocol.

FTC_STATUS **SPI_Read** (FTC_HANDLE *ftHandle*, PFTC_INIT_CONDITION *pReadStartCondition*, BOOL *bClockOutControBitsMSBFirst*, BOOL *bClockOutControBitsPosEdge*, DWORD *dwNumControlBitsToWrite*, PWriteControlByteBuffer *pWriteControlBuffer*, DWORD *dwNumControlBytesToWrite*, BOOL *bClockInDataBitsMSBFirst*, BOOL *bClockInDataBitsPosEdge*, DWORD *dwNumDataBitsToRead*, PReadDataByteBuffer *pReadDataBuffer*, LPDWORD *lpdwNumDataBytesReturned*, PFTC_HIGHER_OUTPUT_PINS *pHighPinsReadActiveStates*)

Parameters

ftHandle	Handle of the device.
pReadStartCondition	Pointer to the structure that contains the start output states (low or high) of the clock, data out and signal out/chip select pins of the FT2232C.
bClockOutControBitsMSBFirst	Clock out control bits most significant bit (MSB) first (TRUE), clock out control bits least significant bit (LSB) first (FALSE).
bClockOutControBitsPosEdge	Clock out control bits on positive clock edge (TRUE), clock out control bits on negative clock edge (FALSE).
dwNumControlBitsToWrite	Specifies the number of control bits to be written to an external device. Valid range 2 to 2040. 2040 bits is equivalent to 255 bytes.
pWriteControlBuffer	Pointer to buffer that contains the control data to be written to an external device. Listed below is an example of control and address bytes: Control Address byte 1, Control Address byte 2
dwNumControlBytesToWrite	Specifies the number of control bytes in the write control buffer, which contains all the specified bits to be written to an external device. Valid range 1 to 255 bytes.
bClockInDataBitsMSBFirst	Clock in data bits most significant bit (MSB) first (TRUE), clock in data bits least significant bit (LSB) first (FALSE).
bClockInDataBitsPosEdge	Clock in data bits on positive clock edge (TRUE), clock in data bits on negative clock edge (FALSE).
dwNumDataBitsToRead	Specifies the number of bits to be read from an external device. Valid range 2 to 524280. 524280 bits is equivalent to 64K bytes.
pReadDataBuffer	Pointer to buffer that returns the data read from an external device. Size of buffer should be set to 65535.
lpdwNumDataBytesReturned	Pointer to a variable of type DWORD which receives the actual number of data bytes read from an external device. These bytes contain the specified number of bits read from an external device.
pHighPinsReadActiveStates	Pointer to the structure that contains which of the 4 upper general purpose input/output pins of a FT2232C, are to be used during a write to an external device. Each GPIO pin that is to be used during a read from an external device must have been previously configured as an output pin (see SPI_SetGPIOs 15).

Return Value
FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_INVALID_HANDLE
FTC_NULL_INITIAL_CONDITION_BUFFER_POINTER
FTC_INVALID_NUMBER_CONTROL_BITS
FTC_NULL_WRITE_CONTROL_BUFFER_POINTER
FTC_INVALID_NUMBER_CONTROL_BYTES
FTC_NUMBER_CONTROL_BYTES_TOO_SMALL
FTC_INVALID_NUMBER_READ_DATA_BITS
FTC_NULL_READ_DATA_BUFFER_POINTER
FTC_NULL_OUTPUT_PINS_BUFFER_POINTER
FTC_INVALID_INIT_CLOCK_PIN_STATE
FTC_INVALID_FT2232C_CHIP_SELECT_PIN
FTC_INVALID_CONFIGURATION_HIGHER_GPIO_PIN
FTC_FAILED_TO_COMPLETE_COMMAND
FTC_IO_ERROR

Remarks
This function will read data from an external device to the FT2232C using the SPI protocol. The data will be clocked at a rate specified by the clock divisor set by calling either the SPI_InitDevice 9 or SPI_SetClock 12 functions.

The init condition, write control buffer, read data buffer and high pins read active states definitions are given in the Appendix 31.

Example

```
#define NUM_93LC56B_CMD_CONTOL_BITS 11
#define NUM_93LC56B_CMD_CONTOL_BYTES 2

#define NUM_93LC56B_CMD_DATA_BITS 16

FTC_STATUS Status = FTC_SUCCESS;
FTC_HANDLE ftHandle;
FTC_INIT_CONDITION ReadStartCondition;
WriteControlByteBuffer WriteControlBuffer;
ReadDataByteBuffer ReadDataBuffer;
DWORD dwNumDataBytesReturned = 0;
FTC_HIGHER_OUTPUT_PINS HighPinsWriteActiveStates;

ReadStartCondition.bClockPinState = false;
ReadStartCondition.bDataOutPinState = false;
ReadStartCondition.bChipSelectPinState = false;
ReadStartCondition.dwChipSelectPin = ADBUS3ChipSelect;

dwReadDataWordAddress = 0;

// set up read command and address
dwControlLocAddress1 = 192; //'\xC0';
dwControlLocAddress1 = (dwControlLocAddress1 | ((dwReadDataWordAddress / 8) & '\x0F'));

// shift left 5 bits ie make bottom 3 bits the 3 MSB's
dwControlLocAddress2 = ((dwReadDataWordAddress & '\x07') * 32);

WriteControlBuffer[0] = (dwControlLocAddress1 & '\xFF');
WriteControlBuffer[1] = (dwControlLocAddress2 & '\xFF');

Status = SPI_Read(ftHandle, &ReadStartCondition, true, false, NUM_93LC56B_CMD_CONTOL_BITS,
&WriteControlBuffer, NUM_93LC56B_CMD_CONTOL_BYTES, true, false, NUM_93LC56B_CMD_DATA_BITS,
&ReadDataBuffer, &dwNumDataBytesReturned, &HighPinsWriteActiveStates);
```

2.14 SPI_ClearDeviceCmdSequence

Clears the sequence of commands and associated data from the internal command buffer for the specified device.

FTC_STATUS **SPI_ClearDeviceCmdSequence** (FTC_HANDLE *ftHandle*)

Parameters

ftHandle	Handle of the device.

Return Value

Always returns FTC_SUCCESS.

Remarks

This function will clear the buffer containing commands which were created by calling SPI_AddDeviceWriteCmd 23 and SPI_AddDeviceReadCmd 26.

Example

```
FTC_STATUS Status = FTC_SUCCESS;
FTC_HANDLE ftHandle;

Status = SPI_ClearDeviceCmdSequence(ftHandle);
```

2.15 SPI_AddDeviceWriteCmd

Adds a write command and associated data to the internal command buffer associated with a device. This enables a programmer to build up a sequence of commands i.e. write and read before executing the sequence of commands. The internal command buffer has a size of 131070 bytes (128 kB).

FTC_STATUS **SPI_AddDeviceWriteCmd** (FTC_HANDLE *ftHandle*, PFTC_INIT_CONDITION *pWriteStartCondition*, BOOL *bClockOutDataBitsMSBFirst*, BOOL *bClockOutDataBitsPosEdge*, DWORD *dwNumControlBitsToWrite*, PWriteControlByteBuffer *pWriteControlBuffer*, DWORD *dwNumControlBytesToWrite*, BOOL *bWriteDataBits*, DWORD *dwNumDataBitsToWrite*, PWriteDataByteBuffer *pWriteDataBuffer*, DWORD *dwNumDataBytesToWrite*, PFTC_HIGHER_OUTPUT_PINS *pHighPinsWriteActiveStates*)

Parameters

ftHandle	Handle of the device.
pWriteStartCondition	Pointer to the structure that contains the start output states (low or high) of the clock, data out and signal out/chip select pins of the FT2232C.
bClockOutDataBitsMSBFirst	Clock out control and data bits most significant bit (MSB) first (TRUE), clock out control and data bits least significant bit (LSB) first (FALSE).
bClockOutDataBitsPosEdge	Clock out control and data bits on positive clock edge (TRUE), clock out control and data bits on negative clock edge (FALSE).
dwNumControlBitsToWrite	Specifies the number of control bits to be written to an external device. Valid range 2 to 2040. 2040 bits is equivalent to 255 bytes.
pWriteControlBuffer	Pointer to buffer that contains the control data to be written to an external device. Listed below are three examples of control and address bytes: Two Control bytes Control Address byte 1, Control Address byte 2 Two Control bytes, Control Address byte 1, Control Address byte 2
dwNumControlBytesToWrite	Specifies the number of control bytes in the write control buffer, which contains all the specified bits to be written to an external device. Valid range 1 to 255 bytes.
bWriteDataBits	Write data bits to an external device (TRUE), do not write any data bits to an external device (FALSE).
dwNumDataBitsToWrite	Specifies the number of data bits to be written to an external device. Valid range 2 to 524280. 524280 bits is equivalent to 64K bytes.
pWriteDataBuffer	Pointer to buffer that contains the data to be written to an external device.
dwNumDataBytesToWrite	Specifies the number of data bytes in the write data buffer, which contains all the specified bits to be written to an external device. Valid range 1 to 65535 (i.e. 64K bytes).
pHighPinsWriteActiveStates	Pointer to the structure that contains which of the 4 upper general purpose input/output pins of a FT2232C, are to be used during a write to an external device. Each GPIO pin that is to be used during a write to an external device must have been previously configured as an output pin (see SPI_SetGPIOs 15).

Return Value

FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_INVALID_HANDLE
FTC_NULL_INITIAL_CONDITION_BUFFER_POINTER
FTC_INVALID_NUMBER_CONTROL_BITS
FTC_NULL_WRITE_CONTROL_BUFFER_POINTER
FTC_INVALID_NUMBER_CONTROL_BYTES
FTC_NUMBER_CONTROL_BYTES_TOO_SMALL
FTC_INVALID_NUMBER_DATA_BITS
FTC_NULL_WRITE_DATA_BUFFER_POINTER
FTC_INVALID_NUMBER_DATA_BYTES
FTC_NUMBER_DATA_BYTES_TOO_SMALL
FTC_INVALID_FT2232C_DATA_WRITE_COMPLETE_PIN

FTC_NULL_OUTPUT_PINS_BUFFER_POINTER
FTC_INVALID_INIT_CLOCK_PIN_STATE
FTC_INVALID_FT2232C_CHIP_SELECT_PIN
FTC_DATA_WRITE_COMPLETE_TIMEOUT
FTC_INVALID_FT2232C_DATA_WRITE_COMPLETE_PIN
FTC_INVALID_CONFIGURATION_HIGHER_GPIO_PIN
FTC_COMMAND_SEQUENCE_BUFFER_FULL

Remarks

Do not invoke SPI_Write 17 or SPI_Read 20 functions while constructing a sequence of commands as this will clear the sequence of commands and associated data from the internal command buffer.

This command can be used with multiple devices connected.

Calling this function is equivalent to adding the commands and data from a SPI_Write 17 call to the internal command buffer.

This function can be used with SPI_ClearDeviceCmdSequence 22, SPI_AddDeviceReadCmd 26 and SPI_ExecuteCmdSequence 28 to buffer a long list of commands and data which can then be sent to the FT2232C in one go. This can provide faster data transfer rates in some applications.

The write start condition, write control buffer, write data buffer, wait data write complete and high pins write active states definitions are given in the Appendix 31.

2.16 SPI_AddDeviceReadCmd

Adds a read command to the internal command buffer associated with a device. This enables a programmer to build up a sequence of commands i.e. write and read before executing the sequence of commands. The internal command buffer has a size of 131070 bytes (128 kB).

FTC_STATUS **SPI_AddDeviceReadCmd** (FTC_HANDLE *ftHandle*, PFTC_INIT_CONDITION *pReadStartCondition*, BOOL *bClockOutControBitsMSBFirst*, BOOL *bClockOutControBitsPosEdge*, DWORD *dwNumControlBitsToWrite*, PWriteControlByteBuffer *pWriteControlBuffer*, DWORD *dwNumControlBytesToWrite*, BOOL *bClockInDataBitsMSBFirst*, BOOL *bClockInDataBitsPosEdge*, DWORD *dwNumDataBitsToRead*, PReadDataByteBuffer *pReadDataBuffer*, LPDWORD *lpdwNumDataBytesReturned*, PFTC_HIGHER_OUTPUT_PINS *pHighPinsReadActiveStates*)

Parameters

ftHandle	Handle of the device.
pReadStartCondition	Pointer to the structure that contains the start output states (low or high) of the clock, data out and signal out/chip select pins of the FT2232C.
bClockOutControBitsMSBFirst	Clock out control bits most significant bit (MSB) first (TRUE), clock out control bits least significant bit (LSB) first (FALSE).
bClockOutControBitsPosEdge	Clock out control bits on positive clock edge (TRUE), clock out control bits on negative clock edge (FALSE).
dwNumControlBitsToWrite	Specifies the number of control bits to be written to an external device. Valid range 2 to 2040. 2040 bits is equivalent to 255 bytes.
pWriteControlBuffer	Pointer to buffer that contains the control data to be written to an external device. Listed below is an example of control and address bytes: Control Address byte 1, Control Address byte 2
dwNumControlBytesToWrite	Specifies the number of control bytes in the write control buffer, which contains all the specified bits to be written to an external device. Valid range 1 to 255 bytes.
bClockInDataBitsMSBFirst	Clock in data bits most significant bit (MSB) first (TRUE), clock in data bits least significant bit (LSB) first (FALSE).
bClockInDataBitsPosEdge	Clock in data bits on positive clock edge (TRUE), clock in data bits on negative clock edge (FALSE).
dwNumDataBitsToRead	Specifies the number of bits to be read from an external device. Valid range 2 to 524280. 524280 bits is equivalent to 64K bytes.
pHighPinsReadActiveStates	Pointer to the structure that contains which of the 4 upper general purpose input/output pins of a FT2232C, are to be used during a write to an external device. Each GPIO pin that is to be used during a read from an external device must have been previously configured as an output pin (see SPI_SetGPIOs 15).

Return Value
FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_INVALID_HANDLE
FTC_NULL_INITIAL_CONDITION_BUFFER_POINTER
FTC_INVALID_NUMBER_CONTROL_BITS
FTC_NULL_WRITE_CONTROL_BUFFER_POINTER
FTC_INVALID_NUMBER_CONTROL_BYTES
FTC_NUMBER_CONTROL_BYTES_TOO_SMALL
FTC_INVALID_NUMBER_DATA_BITS
FTC_NULL_OUTPUT_PINS_BUFFER_POINTER
FTC_INVALID_INIT_CLOCK_PIN_STATE
FTC_INVALID_FT2232C_CHIP_SELECT_PIN
FTC_INVALID_CONFIGURATION_HIGHER_GPIO_PIN
FTC_COMMAND_SEQUENCE_BUFFER_FULL
FTC_INSUFFICIENT_RESOURCES

Remarks
Do not invoke SPI_Write 17 or SPI_Read 20 functions while constructing a sequence of commands as this will clear the sequence of commands and associated data from the internal command buffer.

This command can be used with multiple devices connected.

Calling this function is equivalent to adding the commands and data from a SPI_Read 20 call to the internal command buffer.

This function can be used with SPI_ClearDeviceCmdSequence 22, SPI_AddDeviceWriteCmd 23 and SPI_ExecuteCmdSequence 28 to buffer a long list of commands and data which can then be sent to the FT2232C in one go. This can provide faster data transfer rates in some applications.

The read start condition, write control buffer and high pins read active states definitions are given in the Appendix 31.

2.17 SPI_ExecuteCmdSequence

Executes a sequence of commands stored in the internal command buffer.

FTC_STATUS **SPI_ExecuteCmdSequence** (FTC_HANDLE *ftHandle*, PReadCmdSequenceDataByteBuffer *pReadCmdSequenceDataBuffer*, LPDWORD *lpdwNumBytesReturned*)

Parameters

ftHandle	Handle of the device.
pReadCmdSequenceBuffer	Pointer to buffer that returns the data read from an external device. Size of buffer should be set to 131071 bytes (128KB).
lpdwNumBytesReturned	Pointer to the actual number of bytes read from the external device. These bytes contain the total number of bits read as specified in the sequence of read and write/read commands.

Return Value

Returns FTC_SUCCESS if successful, otherwise the return value will be one of the following error codes:

FTC_INVALID_HANDLE
FTC_NO_COMMAND_SEQUENCE
FTC_NULL_READ_CMDS_DATA_BUFFER_POINTER
FTC_FAILED_TO_COMPLETE_COMMAND
FTC_IO_ERROR

Remarks

Do not invoke SPI_Write 17 or SPI_Read 20 functions while constructing a sequence of commands as this will clear the sequence of commands and associated data from the internal command buffer.

Calling this function will send the contents of the internal command buffer to the FT2232C in one go.

This function can be used with SPI_ClearDeviceCmdSequence 22, SPI_AddDeviceWriteCmd 23 and SPI_AddDeviceReadCmd 26 to buffer a long list of commands and data which can then be sent to the FT2232C in one go. This can provide faster data transfer rates in some applications.

Example

```
FTC_STATUS Status = FTC_SUCCESS;
FTC_HANDLE ftHandle;
ReadCmdSequenceDataByteBuffer ReadCmdSequenceDataBuffer;
DWORD dwNumBytesReturned = 0;

Status = SPI_ExecuteCmdSequence(ftHandle, &ReadCmdSequenceDataBuffer,
&dwNumBytesReturned);
```

2.18 SPI_GetDllVersion

Returns the version number of the current FTC_SPI DLL.

FTC_STATUS **SPI_GetDllVersion** (LPSTR *lpDllVersionBuffer*, DWORD *dwBufferSize*)

Parameters

lpDllVersionBuffer	Pointer to the buffer that receives the version of this DLL. The string will be NULL terminated.
dwBufferSize	Length of the buffer created for the device name string. Set buffer length to a minimum of 10 characters.

Return Value

FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_NULL_DLL_VERSION_BUFFER_POINTER
FTC_DLL_VERSION_BUFFER_TOO_SMALL

Example

```
FTC_STATUS Status = FTC_SUCCESS;
char szDllVersion[10];

Status = SPI_GetDllVersion(szDllVersion, 10);
```

2.19 SPI_GetErrorCodeString

Provides an explanation of an error code.

FTC_STATUS **SPI_GetErrorCodeString** (LPSTR *lpLanguage*, FTC_STATUS *StatusCode*, LPSTR *lpErrorMessageBuffer*, DWORD *dwBufferSize*)

Parameters

lpLanguage	Pointer to a NULL terminated string that contains the language code. Default for this first version the default language will be English (EN).
StatusCode	Status code returned from a previous FTC_SPI DLL function call.
lpErrorMessageBuffer	Pointer to the buffer that receives the error code explanation string.
dwBufferSize	Length of the buffer created for the error code explanation string. Set buffer length to a minimum of 100 characters.

Return Value

FTC_SUCCESS if successful, otherwise the return value is one of the following FTC error codes:

FTC_NULL_LANGUAGE_CODE_BUFFER_POINTER
FTC_INVALID_LANGUAGE_CODE
FTC_INVALID_STATUS_CODE
FTC_NULL_ERROR_MESSAGE_BUFFER_POINTER
FTC_ERROR_MESSAGE_BUFFER_TOO_SMALL

Example

```
FTC_STATUS Status = FTC_SUCCESS;
char szErrorMessage[100];

Status = SPI_GetErrorCodeString("EN", Status, szErrorMessage, 100);
```

3 Appendix

3.1 Type Definitions

For Visual C++ applications, these values are pre-declared in the header file (FTCSPI.H 34), which is included in the driver release. For other languages, these definitions will have to be converted to use equivalent types and may have to be defined in an include file or within the body of the code.

DWORD	Unsigned long (4 bytes)
LPDWORD	Long pointer to a DWORD value
BOOL	Boolean value (4 bytes)
LPSTR	Long pointer to a NULL terminated string
FTC_HANDLE	DWORD

FTC_STATUS (DWORD)

FTC_SUCCESS = 0
FTC_INVALID_HANDLE = 1
FTC_DEVICE_NOT_FOUND = 2
FTC_DEVICE_NOT_OPENED = 3
FTC_IO_ERROR = 4
FTC_INSUFFICIENT_RESOURCES = 5

FTC_FAILED_TO_COMPLETE_COMMAND = 20
FTC_FAILED_TO_SYNCHRONIZE_DEVICE_MPSSE = 21
FTC_INVALID_DEVICE_NAME_INDEX = 22
FTC_NULL_DEVICE_NAME_BUFFER_POINTER = 23
FTC_DEVICE_NAME_BUFFER_TOO_SMALL = 24
FTC_INVALID_DEVICE_NAME = 25
FTC_INVALID_LOCATION_ID = 26
FTC_DEVICE_IN_USE = 27
FTC_TOO_MANY_DEVICES = 28
FTC_INVALID_CLOCK_DIVISOR = 29
FTC_NULL_INPUT_BUFFER_POINTER = 30
FTC_NULL_CHIP_SELECT_BUFFER_POINTER = 31
FTC_NULL_INPUT_OUTPUT_BUFFER_POINTER = 32
FTC_NULL_OUTPUT_PINS_BUFFER_POINTER = 33
FTC_NULL_INITIAL_CONDITION_BUFFER_POINTER = 34
FTC_NULL_WRITE_CONTROL_BUFFER_POINTER = 35
FTC_NULL_WRITE_DATA_BUFFER_POINTER = 36
FTC_NULL_WAIT_DATA_WRITE_BUFFER_POINTER = 37
FTC_NULL_READ_DATA_BUFFER_POINTER = 38
FTC_NULL_READ_CMDS_DATA_BUFFER_POINTER = 39
FTC_INVALID_NUMBER_CONTROL_BITS = 40
FTC_INVALID_NUMBER_CONTROL_BYTES = 41
FTC_NUMBER_CONTROL_BYTES_TOO_SMALL = 42
FTC_INVALID_NUMBER_WRITE_DATA_BITS = 43
FTC_INVALID_NUMBER_WRITE_DATA_BYTES = 44
FTC_NUMBER_WRITE_DATA_BYTES_TOO_SMALL = 45
FTC_INVALID_NUMBER_READ_DATA_BITS = 46
FTC_INVALID_INIT_CLOCK_PIN_STATE = 47
FTC_INVALID_FT2232C_CHIP_SELECT_PIN = 48
FTC_INVALID_FT2232C_DATA_WRITE_COMPLETE_PIN = 49
FTC_DATA_WRITE_COMPLETE_TIMEOUT = 50

```
        FTC_INVALID_CONFIGURATION_HIGHER_GPIO_PIN = 51
        FTC_COMMAND_SEQUENCE_BUFFER_FULL = 52
        FTC_NO_COMMAND_SEQUENCE = 53
        FTC_NULL_DLL_VERSION_BUFFER_POINTER = 54
        FTC_DLL_VERSION_BUFFER_TOO_SMALL = 55
        FTC_NULL_LANGUAGE_CODE_BUFFER_POINTER = 56
        FTC_NULL_ERROR_MESSAGE_BUFFER_POINTER = 57
        FTC_ERROR_MESSAGE_BUFFER_TOO_SMALL = 58
        FTC_INVALID_LANGUAGE_CODE = 59
        FTC_INVALID_STATUS_CODE = 60
```

FTC_INIT_CONDITION

```
typedef struct FTC_Init_Condition {
        BOOL  bClockPinState
        BOOL  bDataOutPinState
        BOOL  bChipSelectPinState
        DWORD  dwChipSelectPin
} FTC_INIT_CONDITION *PFTC_INIT_CONDITION
```

FTC_CHIP_SELECT_PINS

```
typedef struct Ft_Chip_Select_Pins{
        BOOL  bADBUS3ChipSelectPinState
        BOOL  bADBUS4GPIOL1PinState
        BOOL  bADBUS5GPIOL2PinState
        BOOL  bADBUS6GPIOL3PinState
        BOOL  bADBUS7GPIOL4PinState
}FTC_CHIP_SELECT_PINS, *PFTC_CHIP_SELECT_PINS;
```

FTC_INPUT_OUTPUT_PINS

```
typedef struct FTC_Higher_Output_Pins {
        BOOL  bPin1State
        BOOL  bPin1ActiveState
        BOOL  bPin2State
        BOOL  bPin2ActiveState
        BOOL  bPin3State
        BOOL  bPin3ActiveState
        BOOL  bPin4State
        BOOL  bPin4ActiveState
} FTC_HIGHER_OUTPUT_PINS *PFTC_HIGHER_OUTPUT_PINS
```

WRITE CONTROL BYTE BUFFER

```
#define MAX_WRITE_CONTROL_BYTES_BUFFER_SIZE         256       // 256 bytes
typedef BYTE WriteControlByteBuffer[MAX_WRITE_CONTROL_BYTES_BUFFER_SIZE];
typedef WriteControlByteBuffer  *PWriteControlByteBuffer;
```

WRITE DATA BYTE BUFFER

```
#define MAX_WRITE_DATA_BYTES_BUFFER_SIZE 65536       // 64K bytes
```

```
typedef BYTE WriteDataByteBuffer[MAX_WRITE_DATA_BYTES_BUFFER_SIZE];
typedef WriteDataByteBuffer  *PWriteDataByteBuffer;
```

FTC_WAIT_DATA_WRITE

```
typedef struct FTC_Wait_Data_Write {
        BOOL  bWaitDataWriteComplete
        DWORD  dwWaitDataWritePin
        BOOL  bDataWriteCompleteState
        DWORD  dwDataWriteTimeoutmSecs
} FTC_WAIT_DATA_WRITE  *PFTC_WAIT_DATA_WRITE
```

READ DATA BYTE BUFFER

```
#define MAX_READ_DATA_BYTES_BUFFER_SIZE 65536        // 64K bytes
typedef BYTE ReadDataByteBuffer[MAX_READ_DATA_BYTES_BUFFER_SIZE];
typedef ReadDataByteBuffer *PReadDataByteBuffer;
```

3.2 FTCSPI.H

```
/*++

Copyright (c) 2005 Future Technology Devices International Ltd.

Module Name:

    ftcspi.h

Abstract:

    API DLL for FT2232C Dual Device setup to simulate the Serial Peripheral Interface(SPI)
synchronous protocol.
    FTCSPI library definitions

Environment:

    kernel & user mode

Revision History:

    13/05/05    kra     Created.

--*/


#ifndef FTCSPI_H
#define FTCSPI_H


// The following ifdef block is the standard way of creating macros
// which make exporting from a DLL simpler.  All files within this DLL
// are compiled with the FTCSPI_EXPORTS symbol defined on the command line.
// This symbol should not be defined on any project that uses this DLL.
// This way any other project whose source files include this file see
// FTCSPI_API functions as being imported from a DLL, whereas this DLL
// sees symbols defined with this macro as being exported.

#ifdef FTCSPI_EXPORTS
#define FTCSPI_API __declspec(dllexport)
#else
#define FTCSPI_API __declspec(dllimport)
#endif

typedef DWORD FTC_HANDLE;
typedef ULONG FTC_STATUS;

#define ADBUS3ChipSelect 0
#define ADBUS4GPIOL1 1
#define ADBUS5GPIOL2 2
#define ADBUS6GPIOL3 3
#define ADBUS7GPIOL4 4

#define ADBUS2DataIn 0
#define ACBUS0GPIOH1 1
#define ACBUS1GPIOH2 2
```

```
#define ACBUS2GPIOH3 3
#define ACBUS3GPIOH4 4

#define FTC_SUCCESS 0 // FTC_OK
#define FTC_INVALID_HANDLE 1 // FTC_INVALID_HANDLE
#define FTC_DEVICE_NOT_FOUND 2 //FTC_DEVICE_NOT_FOUND
#define FTC_DEVICE_NOT_OPENED 3 //FTC_DEVICE_NOT_OPENED
#define FTC_IO_ERROR 4 //FTC_IO_ERROR
#define FTC_INSUFFICIENT_RESOURCES 5 // FTC_INSUFFICIENT_RESOURCES

#define FTC_FAILED_TO_COMPLETE_COMMAND 20          // cannot change, error code
mapped from FT2232c classes
#define FTC_FAILED_TO_SYNCHRONIZE_DEVICE_MPSSE 21  // cannot change, error code
mapped from FT2232c classes
#define FTC_INVALID_DEVICE_NAME_INDEX 22           // cannot change, error code mapped
from FT2232c classes
#define FTC_NULL_DEVICE_NAME_BUFFER_POINTER 23     // cannot change, error code
mapped from FT2232c classes
#define FTC_DEVICE_NAME_BUFFER_TOO_SMALL 24        // cannot change, error code
mapped from FT2232c classes
#define FTC_INVALID_DEVICE_NAME 25                 // cannot change, error code mapped from
FT2232c classes
#define FTC_INVALID_LOCATION_ID 26                 // cannot change, error code mapped from
FT2232c classes
#define FTC_DEVICE_IN_USE 27                       // cannot change, error code mapped from
FT2232c classes
#define FTC_TOO_MANY_DEVICES 28                    // cannot change, error code mapped from
FT2232c classes
#define FTC_INVALID_CLOCK_DIVISOR 29
#define FTC_NULL_INPUT_BUFFER_POINTER 30
#define FTC_NULL_CHIP_SELECT_BUFFER_POINTER 31
#define FTC_NULL_INPUT_OUTPUT_BUFFER_POINTER 32
#define FTC_NULL_OUTPUT_PINS_BUFFER_POINTER 33
#define FTC_NULL_INITIAL_CONDITION_BUFFER_POINTER 34
#define FTC_NULL_WRITE_CONTROL_BUFFER_POINTER 35
#define FTC_NULL_WRITE_DATA_BUFFER_POINTER 36
#define FTC_NULL_WAIT_DATA_WRITE_BUFFER_POINTER 37
#define FTC_NULL_READ_DATA_BUFFER_POINTER 38
#define FTC_NULL_READ_CMDS_DATA_BUFFER_POINTER 39
#define FTC_INVALID_NUMBER_CONTROL_BITS 40
#define FTC_INVALID_NUMBER_CONTROL_BYTES 41
#define FTC_NUMBER_CONTROL_BYTES_TOO_SMALL 42
#define FTC_INVALID_NUMBER_WRITE_DATA_BITS 43
#define FTC_INVALID_NUMBER_WRITE_DATA_BYTES 44
#define FTC_NUMBER_WRITE_DATA_BYTES_TOO_SMALL 45
#define FTC_INVALID_NUMBER_READ_DATA_BITS 46
#define FTC_INVALID_INIT_CLOCK_PIN_STATE 47
#define FTC_INVALID_FT2232C_CHIP_SELECT_PIN 48
#define FTC_INVALID_FT2232C_DATA_WRITE_COMPLETE_PIN 49
#define FTC_DATA_WRITE_COMPLETE_TIMEOUT 50
#define FTC_INVALID_CONFIGURATION_HIGHER_GPIO_PIN 51
#define FTC_COMMAND_SEQUENCE_BUFFER_FULL 52
#define FTC_NO_COMMAND_SEQUENCE 53
#define FTC_NULL_DLL_VERSION_BUFFER_POINTER 54
#define FTC_DLL_VERSION_BUFFER_TOO_SMALL 55
#define FTC_NULL_LANGUAGE_CODE_BUFFER_POINTER 56
#define FTC_NULL_ERROR_MESSAGE_BUFFER_POINTER 57
#define FTC_ERROR_MESSAGE_BUFFER_TOO_SMALL 58
```

```
#define FTC_INVALID_LANGUAGE_CODE 59
#define FTC_INVALID_STATUS_CODE 60

#ifdef __cplusplus
extern "C" {
#endif

FTCSPI_API
FTC_STATUS WINAPI SPI_GetNumDevices(LPDWORD lpdwNumDevices);

FTCSPI_API
FTC_STATUS WINAPI SPI_GetDeviceNameLocID(DWORD dwDeviceNameIndex, LPSTR
lpDeviceNameBuffer, DWORD dwBufferSize, LPDWORD lpdwLocationID);

FTCSPI_API
FTC_STATUS WINAPI SPI_OpenEx(LPSTR lpDeviceName, DWORD dwLocationID,
FTC_HANDLE *pftHandle);

FTCSPI_API
FTC_STATUS WINAPI SPI_Open(FTC_HANDLE *pftHandle);

FTCSPI_API
FTC_STATUS WINAPI SPI_Close(FTC_HANDLE ftHandle);

FTCSPI_API
FTC_STATUS WINAPI SPI_InitDevice(FTC_HANDLE ftHandle, DWORD dwClockDivisor);

FTCSPI_API
FTC_STATUS WINAPI SPI_GetClock(DWORD dwClockDivisor, LPDWORD
lpdwClockFrequencyHz);

FTCSPI_API
FTC_STATUS WINAPI SPI_SetClock(FTC_HANDLE ftHandle, DWORD dwClockDivisor,
LPDWORD lpdwClockFrequencyHz);

FTCSPI_API
FTC_STATUS WINAPI SPI_SetLoopback(FTC_HANDLE ftHandle, BOOL bLoopbackState);

typedef struct Ft_Chip_Select_Pins{
  BOOL  bADBUS3ChipSelectPinState;
  BOOL  bADBUS4GPIOL1PinState;
  BOOL  bADBUS5GPIOL2PinState;
  BOOL  bADBUS6GPIOL3PinState;
  BOOL  bADBUS7GPIOL4PinState;
}FTC_CHIP_SELECT_PINS, *PFTC_CHIP_SELECT_PINS;

typedef struct Ft_Input_Output_Pins{
  BOOL  bPin1InputOutputState;
  BOOL  bPin1LowHighState;
  BOOL  bPin2InputOutputState;
  BOOL  bPin2LowHighState;
  BOOL  bPin3InputOutputState;
  BOOL  bPin3LowHighState;
  BOOL  bPin4InputOutputState;
  BOOL  bPin4LowHighState;
}FTC_INPUT_OUTPUT_PINS, *PFTC_INPUT_OUTPUT_PINS;

FTCSPI_API
```

```
FTC_STATUS WINAPI SPI_SetGPIOs(FTC_HANDLE ftHandle, PFTC_CHIP_SELECT_PINS
pChipSelectsDisableStates,
                        PFTC_INPUT_OUTPUT_PINS pHighInputOutputPinsData);

typedef struct Ft_Low_High_Pins{
  BOOL  bPin1LowHighState;
  BOOL  bPin2LowHighState;
  BOOL  bPin3LowHighState;
  BOOL  bPin4LowHighState;
}FTC_LOW_HIGH_PINS, *PFTC_LOW_HIGH_PINS;

FTCSPI_API
FTC_STATUS WINAPI SPI_GetGPIOs(FTC_HANDLE ftHandle, PFTC_LOW_HIGH_PINS
pHighPinsInputData);

#define MAX_WRITE_CONTROL_BYTES_BUFFER_SIZE 256    // 256 bytes

typedef BYTE WriteControlByteBuffer[MAX_WRITE_CONTROL_BYTES_BUFFER_SIZE];
typedef WriteControlByteBuffer *PWriteControlByteBuffer;

#define MAX_WRITE_DATA_BYTES_BUFFER_SIZE 65536    // 64k bytes

typedef BYTE WriteDataByteBuffer[MAX_WRITE_DATA_BYTES_BUFFER_SIZE];
typedef WriteDataByteBuffer *PWriteDataByteBuffer;

typedef struct Ft_Init_Condition{
  BOOL  bClockPinState;
  BOOL  bDataOutPinState;
  BOOL  bChipSelectPinState;
  DWORD dwChipSelectPin;
}FTC_INIT_CONDITION, *PFTC_INIT_CONDITION;

typedef struct Ft_Wait_Data_Write{
  BOOL  bWaitDataWriteComplete;
  DWORD dwWaitDataWritePin;
  BOOL  bDataWriteCompleteState;
  DWORD dwDataWriteTimeoutmSecs;
}FTC_WAIT_DATA_WRITE,  *PFTC_WAIT_DATA_WRITE;

typedef struct Ft_Higher_Output_Pins{
  BOOL  bPin1State;
  BOOL  bPin1ActiveState;
  BOOL  bPin2State;
  BOOL  bPin2ActiveState;
  BOOL  bPin3State;
  BOOL  bPin3ActiveState;
  BOOL  bPin4State;
  BOOL  bPin4ActiveState;
}FTC_HIGHER_OUTPUT_PINS, *PFTC_HIGHER_OUTPUT_PINS;

FTCSPI_API
FTC_STATUS WINAPI SPI_Write(FTC_HANDLE ftHandle, PFTC_INIT_CONDITION
pWriteStartCondition, BOOL bClockOutDataBitsMSBFirst,
                    BOOL bClockOutDataBitsPosEdge, DWORD dwNumControlBitsToWrite,
PWriteControlByteBuffer pWriteControlBuffer,
                    DWORD dwNumControlBytesToWrite, BOOL bWriteDataBits, DWORD
dwNumDataBitsToWrite, PWriteDataByteBuffer pWriteDataBuffer,
                    DWORD dwNumDataBytesToWrite, PFTC_WAIT_DATA_WRITE
```

```
pWaitDataWriteComplete, PFTC_HIGHER_OUTPUT_PINS pHighPinsWriteActiveStates);

#define MAX_READ_DATA_BYTES_BUFFER_SIZE 65536    // 64k bytes

typedef BYTE ReadDataByteBuffer[MAX_READ_DATA_BYTES_BUFFER_SIZE];
typedef ReadDataByteBuffer *PReadDataByteBuffer;

FTCSPI_API
FTC_STATUS WINAPI SPI_Read(FTC_HANDLE ftHandle, PFTC_INIT_CONDITION
pReadStartCondition, BOOL bClockOutControlBitsMSBFirst,
                    BOOL bClockOutControlBitsPosEdge, DWORD dwNumControlBitsToWrite,
PWriteControlByteBuffer pWriteControlBuffer,
                    DWORD dwNumControlBytesToWrite, BOOL bClockInDataBitsMSBFirst, BOOL
bClockInDataBitsPosEdge,
                    DWORD dwNumDataBitsToRead, PReadDataByteBuffer pReadDataBuffer,
LPDWORD lpdwNumDataBytesReturned,
                    PFTC_HIGHER_OUTPUT_PINS pHighPinsReadActiveStates);

FTCSPI_API
FTC_STATUS WINAPI SPI_ClearDeviceCmdSequence(FTC_HANDLE ftHandle);

FTCSPI_API
FTC_STATUS WINAPI SPI_AddDeviceWriteCmd(FTC_HANDLE ftHandle,
PFTC_INIT_CONDITION pWriteStartCondition, BOOL bClockOutDataBitsMSBFirst,
                            BOOL bClockOutDataBitsPosEdge, DWORD dwNumControlBitsToWrite,
PWriteControlByteBuffer pWriteControlBuffer,
                            DWORD dwNumControlBytesToWrite, BOOL bWriteDataBits, DWORD
dwNumDataBitsToWrite,
                            PWriteDataByteBuffer pWriteDataBuffer, DWORD
dwNumDataBytesToWrite,
                            PFTC_HIGHER_OUTPUT_PINS pHighPinsWriteActiveStates);

FTCSPI_API
FTC_STATUS WINAPI SPI_AddDeviceReadCmd(FTC_HANDLE ftHandle,
PFTC_INIT_CONDITION pReadStartCondition, BOOL bClockOutControlBitsMSBFirst,
                            BOOL bClockOutControlBitsPosEdge, DWORD
dwNumControlBitsToWrite, PWriteControlByteBuffer pWriteControlBuffer,
                            DWORD dwNumControlBytesToWrite, BOOL bClockInDataBitsMSBFirst,
BOOL bClockInDataBitsPosEdge,
                            DWORD dwNumDataBitsToRead, PFTC_HIGHER_OUTPUT_PINS
pHighPinsReadActiveStates);

#define MAX_READ_CMDS_DATA_BYTES_BUFFER_SIZE 131071  // 128K bytes

typedef BYTE
ReadCmdSequenceDataByteBuffer[MAX_READ_CMDS_DATA_BYTES_BUFFER_SIZE];
typedef ReadCmdSequenceDataByteBuffer *PReadCmdSequenceDataByteBuffer;

FTCSPI_API
FTC_STATUS WINAPI SPI_ExecuteDeviceCmdSequence(FTC_HANDLE ftHandle,
PReadCmdSequenceDataByteBuffer pReadCmdSequenceDataBuffer,
                                    LPDWORD lpdwNumBytesReturned);

FTCSPI_API
FTC_STATUS WINAPI SPI_GetDllVersion(LPSTR lpDllVersionBuffer, DWORD dwBufferSize);

FTCSPI_API
FTC_STATUS WINAPI SPI_GetErrorCodeString(LPSTR lpLanguage, FTC_STATUS StatusCode,
```

```
                                LPSTR lpErrorMessageBuffer, DWORD dwBufferSize);

#ifdef __cplusplus
}
#endif


#endif  /* FTCSPI_H */
```

Index

- F -

- I -

- M -

- S -

- T -

- W -