



**Future Technology Devices
International Ltd.**

FTCJTAG Programmer's Guide

Table of Contents

Part I Welcome to the FTCJTAG Programmer's Guide	3
Part II JTAG Interface Functions	4
1 JTAG_GetNumDevices	5
2 JTAG_GetDeviceNameLocID	6
3 JTAG_Open	7
4 JTAG_OpenEx	8
5 JTAG_Close	9
6 JTAG_InitDevice	10
7 JTAG_GetClock	12
8 JTAG_SetClock	13
9 JTAG_SetLoopback	14
10 JTAG_GetGPIOs	15
11 JTAG_SetGPIOs	16
12 JTAG_Write	18
13 JTAG_Read	20
14 JTAG_WriteRead	21
15 JTAG_GenerateClockPulses	23
16 JTAG_ClearCmdSequence	24
17 JTAG_ClearDeviceCmdSequence	25
18 JTAG_AddWriteCmd	26
19 JTAG_AddDeviceWriteCmd	28
20 JTAG_AddReadCmd	30
21 JTAG_AddDeviceReadCmd	31
22 JTAG_AddWriteReadCmd	32
23 JTAG_AddDeviceWriteReadCmd	34
24 JTAG_ExecuteCmdSequence	36
25 JTAG_GetDIIVersion	38
26 JTAG_GetErrorCodeString	39
Part III Appendix	40
1 Type Definitions	40
2 FTCJTAG.H	42

Index	47
--------------	-----------

1 Welcome to the FTCJTAG 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 JTAG, SPI and I2C.

The FTCJTAG DLL has been created to allow application developers to use the FT2232C to create a USB to Joint Test Action Group (JTAG) protocol interface without any knowledge of the MPSSE command set. All of the functions in FTCJTAG.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 FTCJTAG.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 JTAG communication.

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

The FTCJTAG DLL can be downloaded from the [FTCJTAG](#) page of the [MPSSE section](#) in the [Projects](#) area of the site.

2 JTAG Interface Functions

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

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

2.1 **JTAG_GetNumDevices**

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

FTC_STATUS **JTAG_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 [JTAG_GetDeviceNameLocID](#)⁶⁴.

Example

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

Status = JTAG_GetNumDevices(&dwNumDevices);
```

2.2 JTAG_GetDeviceNameLocID

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

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

Parameters

<i>dwDeviceNameIndex</i>	Index of the FT2232C device.
<i>lpDeviceNameBuffer</i>	Pointer to buffer that receives the device name of the specified FT2232C device connected to a system. The string will be NULL terminated.
<i>dwBufferSize</i>	Length of the buffer created for the device name string. Set buffer length to a minimum of 50 characters.
<i>lpdwLocationID</i>	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 [JTAG_GetNumDevices](#)^[5] 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 [JTAG_OpenEx](#)^[8] to open a specific device.

Example

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

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

2.3 JTAG_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 **JTAG_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 [JTAG_GetDeviceNameLocID](#)^[6] and then [JTAG_OpenEx](#)^[8] instead.

Example

```
FTC_STATUS Status = FTC_SUCCESS;  
FTC_HANDLE ftHandle;  
  
Status = JTAG_Open(&ftHandle);
```


2.4 JTAG_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 **JTAG_OpenEx** (LPSTR *lpDeviceName*, DWORD *dwLocationID*, FTC_HANDLE **pftHandle*)

Parameters

<i>lpDeviceName</i>	Pointer to a NULL terminated string that contains the name of the specified FT2232C device to be opened.
<i>dwLocationID</i>	Specifies the location identifier of the specified FT2232C device to be opened.
<i>*pftHandle</i>	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 [JTAG_GetDeviceNameLocID](#) function.

Example

```

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

Status = JTAG_OpenEx(szDeviceName, dwLocationID, &ftHandle);

```

2.5 JTAG_Close

Close an open device.

FTC_STATUS JTAG_Close (FTC_HANDLE *ftHandle*)

Parameters

<i>ftHandle</i>	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 = JTAG_Close(ftHandle);
```

2.6 JTAG_InitDevice

Initialise the device.

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

Parameters

<i>ftHandle</i>	Handle of the device.
<i>dwClockDivisor</i>	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 to the MPSSE controller
- resets the device and purge device USB input buffer
- set data in and data out clock frequency
- set MPSSE loopback state to off (default)
- resets the device and purge device USB input buffer
- reset Test Access Port (TAP) controller of an external device
- set the Test Access Port (TAP) controller of an external device to test idle mode

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:

$$\text{Clock Frequency} = 12\text{MHz} / ((1 + \text{dwClockDivisor}) * 2)$$

Example

```
FTC_STATUS Status = FTC_SUCCESS;  
FTC_HANDLE ftHandle;  
DWORD dwClockDivisor = 0;  
  
Status = JTAG_InitDevice(ftHandle, dwClockDivisor);
```

2.7 JTAG_GetClock

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

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

Parameters

<i>dwClockDivisor</i>	Specifies a clock divisor which will be used to set the frequency for clocking data in and out of the FT2232C device.
<i>lpdwClockFrequencyHz</i>	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 [JTAG_SetClock](#)^[13] or [JTAG_InitDevice](#)^[10].

Example

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

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

2.8 **JTAG_SetClock**

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

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

Parameters

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

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

[JTAG_SetClock](#)^[13] will return the actual frequency in Hz for the current divisor value. The clock frequency in Hz can also be calculated using [JTAG_GetClock](#)^[12].

Example

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

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

2.9 JTAG_SetLoopback

Enables or disables loop back mode.

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

Parameters

<i>ftHandle</i>	Handle of the device.
<i>bLoopbackState</i>	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 = JTAG_SetLoopback(ftHandle, true);
```

2.10 JTAG_GetGPIOs

Reads the values of the 8 general purpose input/output pins of the FT2232C device.

FTC_STATUS JTAG_GetGPIOs (FTC_HANDLE *ftHandle*, BOOL *bControlLowInputOutputPins*, PFTC_LOW_HIGH_PINS *pLowPinsInputData*, BOOL *bControlHighInputOutputPins*, PFTC_LOW_HIGH_PINS *pHighPinsInputData*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>bControlLowInputOutputPins</i>	Enables or disables reading of the lower 4 GPIO pins.
<i>pLowPinsInputData</i>	Pointer to a structure that contains the value of the lower 4 GPIO pins of the FT2232C device.
<i>bControlHighInputOutputPins</i>	Enables or disables reading of the upper 4 GPIO pins.
<i>pHighPinsInputData</i>	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 8 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 [JTAG_SetGPIOs](#)^[16] function.

The definition of the FTC_LOW_HIGH_PINS structure is given in the [Appendix](#)^[40].

Example

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

Status = JTAG_GetGPIOs(ftHandle, true, &LowPinsInputData, true, &HighPinsInputData);
```


2.11 JTAG_SetGPIOs

Controls the use of the 8 general purpose input/output pins of the FT2232C device.

FTC_STATUS **JTAG_SetGPIOs** (FTC_HANDLE *ftHandle*, BOOL *bControlLowInputOutputPins*, PFTC_INPUT_OUTPUT_PINS *pLowInputOutputPinsData*, BOOL *bControlHighInputOutputPins*, PFTC_INPUT_OUTPUT_PINS *pHighInputOutputPinsData*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>bControlLowInputOutputPins</i>	Enables or disables the lower 4 GPIO pins.
<i>pLowInputOutputPinsData</i>	Pointer to a structure that contains the data that is used to control the lower 4 GPIO pins of the FT2232C device.
<i>bControlHighInputOutputPins</i>	Enables or disables the upper 4 GPIO pins.
<i>pHighInputOutputPinsData</i>	Pointer to a structure that contains the data that is used to control 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 provides complete control over the state of the 8 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 [JTAG_GetGPIOs](#)^[15] function.

The definition of the FTC_INPUT_OUTPUT_PINS structure is given in the [Appendix](#)^[40].

Example

```
FTC_STATUS Status = FTC_SUCCESS;
FTC_HANDLE ftHandle;
FTC_INPUT_OUTPUT_PINS LowInputOutputPinsData;
FTC_INPUT_OUTPUT_PINS HighInputOutputPinsData;

// Set low byte pins as output
LowInputOutputPinsData.bPin1InputOutputState = true;
LowInputOutputPinsData.bPin2InputOutputState = true;
LowInputOutputPinsData.bPin3InputOutputState = true;
LowInputOutputPinsData.bPin4InputOutputState = true;

// Set low byte pins as high
LowInputOutputPinsData.bPin1LowHighState = true;
LowInputOutputPinsData.bPin2LowHighState = true;
LowInputOutputPinsData.bPin3LowHighState = true;
LowInputOutputPinsData.bPin4LowHighState = true;
```

```
// Set high pins as output
HighInputOutputPinsData.bPin1InputOutputState = true;
HighInputOutputPinsData.bPin2InputOutputState = true;
HighInputOutputPinsData.bPin3InputOutputState = true;
HighInputOutputPinsData.bPin4InputOutputState = true;

// Set high pins as low
HighInputOutputPinsData.bPin1LowHighState = false;
HighInputOutputPinsData.bPin2LowHighState = false;
HighInputOutputPinsData.bPin3LowHighState = false;
HighInputOutputPinsData.bPin4LowHighState = false;

Status = JTAG_SetGPIOs(ftHandle, true, &LowInputOutputPinsData, true,
&HighInputOutputPinsData);
```

2.12 JTAG_Write

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

FTC_STATUS JTAG_Write (FTC_HANDLE *ftHandle*, BOOL *bInstructionTestData*, DWORD *dwNumBitsToWrite*, PWriteDataByteBuffer *pWriteDataBuffer*, DWORD *dwNumBytesToWrite*, DWORD *dwTapControllerState*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>bInstructionTestData</i>	Selects the instruction register (TRUE) or the test register (FALSE) to write data to.
<i>dwNumBitsToWrite</i>	Number of bits to write to the external device. Valid range is 2 to 524280.
<i>pWriteDataBuffer</i>	Pointer to buffer that contains the data to be written to an external device.
<i>dwNumBytesToWrite</i>	Number of bytes in the write data buffer which contain all the specified bits to be written to the external device. Valid range is 1 to 65535 bytes.
<i>dwTapControllerState</i>	State that the Test Access Port (TAP) controller will be left in.

Return Value

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

```
FTC_INVALID_HANDLE
FTC_INVALID_NUMBER_BITS
FTC_NULL_WRITE_DATA_BUFFER_POINTER
FTC_INVALID_NUMBER_BYTES
FTC_NUMBER_BYTES_TOO_SMALL
FTC_INVALID_TAP_CONTROLLER_STATE
FTC_FAILED_TO_COMPLETE_COMMAND
FTC_IO_ERROR
```

Remarks

This function will write data from the FT2232C to an external device using the JTAG protocol. The data will be clocked at a rate specified by the clock divisor set by calling either the [JTAG_InitDevice](#)^[10] or [JTAG_SetClock](#)^[13] functions.

The write data byte buffer definition and valid TAP controller states are given in the [Appendix](#)^[40].

Example

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

// Add values to write buffer
WriteDataBuffer[0] = '\x00';
WriteDataBuffer[1] = '\x00';
```

```
WriteDataBuffer[2] = '\x00';  
WriteDataBuffer[3] = '\x00';  
  
// Write buffer to device  
Status = JTAG_Write(ftHandle, true, 32, &WriteDataBuffer, 65535, RUN_TEST_IDLE_STATE);
```

2.13 JTAG_Read

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

FTC_STATUS JTAG_Read (FTC_HANDLE *ftHandle*, BOOL *bInstructionTestData*, DWORD *dwNumBitsToRead*, PReadDataByteBuffer *pReadDataBuffer*, LPDWORD *lpdwNumBytesReturned*, DWORD *dwTapControllerState*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>bInstructionTestData</i>	Selects the instruction register (TRUE) or the test register (FALSE) to write data to.
<i>dwNumBitsToRead</i>	Number of bits to be read from the external device. Valid range is 2 to 524280.
<i>pReadDataBuffer</i>	Pointer to buffer that returns the data read from an external device. Size of buffer should be set to 65535.
<i>lpdwNumBytesReturned</i>	Pointer to the actual number of bytes read from the external device.
<i>dwTapControllerState</i>	State that the Test Access Port (TAP) controller will be left in.

Return Value

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

```
FTC_INVALID_HANDLE
FTC_INVALID_NUMBER_BITS
FTC_NULL_READ_DATA_BUFFER_POINTER
FTC_INVALID_TAP_CONTROLLER_STATE
FTC_FAILED_TO_COMPLETE_COMMAND
FTC_IO_ERROR
```

Remarks

This function will read data from an external device to the FT2232C using the JTAG protocol. The data will be clocked at a rate specified by the clock divisor set by calling either the [JTAG_InitDevice](#)^[10] or [JTAG_SetClock](#)^[13] functions.

The read data byte buffer definition and valid TAP controller states are given in the [Appendix](#)^[40].

Example

```
FTC_STATUS Status = FTC_SUCCESS;
FTC_HANDLE ftHandle;
ReadDataByteBuffer ReadDataBuffer;
DWORD dwNumBytesReturned = 0;

Status = JTAG_Read(ftHandle, true, 32, &ReadDataBuffer, &dwNumBytesReturned,
RUN_TEST_IDLE_STATE);
```

2.14 JTAG_WriteRead

Write data from the FT2232C to an external device and then read data from an external device to the FT2232C using the JTAG protocol.

FTC_STATUS JTAG_WriteRead (FTC_HANDLE *ftHandle*, BOOL *bInstructionTestData*, DWORD *dwNumBitsToWriteRead*, PWriteDataByteBuffer *pWriteDataBuffer*, DWORD *dwNumBytesToWrite*, PReadDataByteBuffer *pReadDataBuffer*, LPDWORD *lpdwNumBytesReturned*, DWORD *dwTapControllerState*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>bInstructionTestData</i>	Selects the instruction register (TRUE) or the test register (FALSE) to write data to.
<i>dwNumBitsToWriteRead</i>	Number of bits to be written to and read from an external device. Valid range is 2 to 524280.
<i>pWriteDataBuffer</i>	Pointer to buffer that contains the data to be written to the external device.
<i>dwNumBytesToWrite</i>	Number of bytes in the write data buffer which contains all the specified bits to be written to the external device. Valid range is 1 to 65535 bytes.
<i>pReadDataBuffer</i>	Pointer to buffer that returns the data read from the external device. Size of buffer should be set to 65535.
<i>lpdwNumBytesReturned</i>	Pointer to the actual number of bytes read from the external device.
<i>dwTapControllerState</i>	State that the Test Access Port (TAP) controller will be left in.

Return Value

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

FTC_INVALID_HANDLE
FTC_INVALID_NUMBER_BITS
FTC_NULL_WRITE_DATA_BUFFER_POINTER
FTC_INVALID_NUMBER_BYTES
FTC_NUMBER_BYTES_TOO_SMALL
FTC_NULL_READ_DATA_BUFFER_POINTER
FTC_INVALID_TAP_CONTROLLER_STATE
FTC_FAILED_TO_COMPLETE_COMMAND
FTC_IO_ERROR

Remarks

This function will write data from the FT2232C to an external device using the JTAG protocol. It will then read data back from the external device to the FT2232C using the JTAG protocol. The data will be clocked at a rate specified by the clock divisor set by calling either the [JTAG_InitDevice](#)^[10] or [JTAG_SetClock](#)^[13] functions.

The write data byte buffer definition, read data byte buffer definition and valid TAP controller states are given in the [Appendix](#)^[40].

Example

```
FTC_STATUS Status = FTC_SUCCESS;
FTC_HANDLE ftHandle;
WriteDataByteBuffer WriteDataBuffer;
ReadDataByteBuffer ReadDataBuffer;
DWORD dwNumBytesReturned = 0;

// Add values to write buffer
WriteDataBuffer[0] = '\\x00';
WriteDataBuffer[1] = '\\x00';
WriteDataBuffer[2] = '\\x00';
WriteDataBuffer[3] = '\\x00';

// Write buffer to device and read data back
Status = JTAG_WriteRead(ftHandle, true, 524280, &WriteDataBuffer, 65535, &ReadDataBuffer,
&dwNumBytesReturned, RUN_TEST_IDLE_STATE);
```

2.15 JTAG_GenerateClockPulses

Generate a specified number of clock pulses. The clock pulses will be generated in the run test idle state. The data written to an external device (i.e. a device attached to a FT2232C) during generation of the clock pulses will be 0.

FTC_STATUS **JTAG_GenerateClockPulses** (FTC_HANDLE *ftHandle*, DWORD *dwNumClockPulses*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>dwNumClockPulses</i>	Specifies the number of clock pulses to be generated by a FT2232C. Valid range 1 to 2000,000,000.

Return Value

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

- FTC_INVALID_HANDLE
- FTC_INVALID_NUMBER_CLOCK_PULSES
- FTC_FAILED_TO_COMPLETE_COMMAND
- FTC_IO_ERROR

Remarks

This function will not normally be required and is recommended for experts only.

2.16 JTAG_ClearCmdSequence

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

FTC_STATUS **JTAG_ClearCmdSequence**

Parameters

None.

Return Value

Always returns FTC_SUCCESS.

Remarks

This function will clear the buffer containing commands which were created by calling [JTAG_AddWriteCmd](#)^[26], [JTAG_AddReadCmd](#)^[30] and [JTAG_AddWriteReadCmd](#)^[32].

Example

```
JTAG_ClearCmdSequence();
```

2.17 JTAG_ClearDeviceCmdSequence

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

FTC_STATUS JTAG_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 [JTAG_AddDeviceWriteCmd](#)^[28], [JTAG_AddDeviceReadCmd](#)^[31] and [JTAG_AddDeviceWriteReadCmd](#)^[34].

2.18 JTAG_AddWriteCmd

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

FTC_STATUS **JTAG_AddWriteCmd** (BOOL *bInstructionTestData*, DWORD *dwNumBitsToWrite*, PWriteDataByteBuffer *pWriteDataBuffer*, DWORD *dwNumBytesToWrite*, DWORD *dwTapControllerState*)

Parameters

<i>bInstructionTestData</i>	Selects the instruction register (TRUE) or the test register (FALSE) to write data to.
<i>dwNumBitsToWrite</i>	Number of bits to write to the external device. Valid range is 2 to 524280.
<i>pWriteDataBuffer</i>	Pointer to buffer that contains the data to be written to an external device.
<i>dwNumBytesToWrite</i>	Number of bytes in the write data buffer which contain all the specified bits to be written to the external device. Valid range is 1 to 65535 bytes.
<i>dwTapControllerState</i>	State that the Test Access Port (TAP) controller will be left in.

Return Value

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

FTC_INVALID_NUMBER_BITS
 FTC_NULL_WRITE_DATA_BUFFER_POINTER
 FTC_INVALID_NUMBER_BYTES
 FTC_NUMBER_BYTES_TOO_SMALL
 FTC_INVALID_TAP_CONTROLLER_STATE
 FTC_COMMAND_SEQUENCE_BUFFER_FULL

Remarks

Do not invoke JTAG_Write^[18], JTAG_READ^[20] or JTAG_WriteRead^[21] functions while constructing a sequence of commands as this will clear the sequence of commands and associated data from the internal command buffer.

This function should be only be used when one device is connected. For multiple devices, use JTAG_AddDeviceWriteCmd^[28].

Calling this function is equivalent to adding the commands and data from a JTAG_Write^[18] call to the internal command buffer.

This function can be used with JTAG_ClearCmdSequence^[24], JTAG_AddReadCmd^[30], JTAG_AddWriteReadCmd^[32] and JTAG_ExecuteCmdSequence^[36] 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;
WriteDataByteBuffer WriteDataBuffer;

// Add values to write buffer
WriteDataBuffer[0] = '\x00';
WriteDataBuffer[1] = '\x00';
WriteDataBuffer[2] = '\x00';
WriteDataBuffer[3] = '\x00';

// Add values to command sequence buffer
Status = JTAG_AddWriteCmd(true, 32, &WriteDataBuffer, 4, RUN_TEST_IDLE_STATE);
```

2.19 JTAG_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, read and write/read before executing the sequence of commands. The internal command buffer has a size of 131070 bytes (128 kB).

FTC_STATUS **JTAG_AddDeviceWriteCmd** (FTC_HANDLE *ftHandle*, BOOL *bInstructionTestData*, DWORD *dwNumBitsToWrite*, PWriteDataByteBuffer *pWriteDataBuffer*, DWORD *dwNumBytesToWrite*, DWORD *dwTapControllerState*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>bInstructionTestData</i>	Selects the instruction register (TRUE) or the test register (FALSE) to write data to.
<i>dwNumBitsToWrite</i>	Number of bits to write to the external device. Valid range is 2 to 524280.
<i>pWriteDataBuffer</i>	Pointer to buffer that contains the data to be written to an external device.
<i>dwNumBytesToWrite</i>	Number of bytes in the write data buffer which contain all the specified bits to be written to the external device. Valid range is 1 to 65535 bytes.
<i>dwTapControllerState</i>	State that the Test Access Port (TAP) controller will be left in.

Return Value

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

FTC_INVALID_HANDLE
 FTC_INVALID_NUMBER_BITS
 FTC_NULL_WRITE_DATA_BUFFER_POINTER
 FTC_INVALID_NUMBER_BYTES
 FTC_NUMBER_BYTES_TOO_SMALL
 FTC_INVALID_TAP_CONTROLLER_STATE
 FTC_COMMAND_SEQUENCE_BUFFER_FULL

Remarks

Do not invoke [JTAG_Write](#)^[18], [JTAG_READ](#)^[20] or [JTAG_WriteRead](#)^[21] 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. If only one device is connected, [JTAG_AddWriteCmd](#)^[28] may be used instead.

Calling this function is equivalent to adding the commands and data from a [JTAG_Write](#)^[18] call to the internal command buffer.

This function can be used with [JTAG_ClearDeviceCmdSequence](#)^[25],

[JTAG_AddDeviceReadCmd](#)^[37], [JTAG_AddDeviceWriteReadCmd](#)^[34] and [JTAG_ExecuteCmdSequence](#)^[36] 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.

2.20 JTAG_AddReadCmd

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

FTC_STATUS **JTAG_AddReadCmd** (BOOL *bInstructionTestData*, DWORD *dwNumBitsToRead*, DWORD *dwTapControllerState*)

Parameters

<i>bInstructionTestData</i>	Selects the instruction register (TRUE) or the test register (FALSE) to write data to.
<i>dwNumBitsToRead</i>	Number of bits to be read from the external device. Valid range is 2 to 524280.
<i>dwTapControllerState</i>	State that the Test Access Port (TAP) controller will be left in.

Return Value

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

FTC_INVALID_NUMBER_BITS
FTC_INVALID_TAP_CONTROLLER_STATE
FTC_COMMAND_SEQUENCE_BUFFER_FULL

Remarks

Do not invoke [JTAG_Write](#)^[18], [JTAG_READ](#)^[20] or [JTAG_WriteRead](#)^[27] 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 is equivalent to adding the commands and data from a [JTAG_Read](#)^[20] call to the internal command buffer.

This function can be used with [JTAG_ClearCmdSequence](#)^[24], [JTAG_AddWriteCmd](#)^[26], [JTAG_AddWriteReadCmd](#)^[32] and [JTAG_ExecuteCmdSequence](#)^[36] 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;

Status = JTAG_AddReadCmd(true, 32, RUN_TEST_IDLE_STATE);
```

2.21 JTAG_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, read and write/read before executing the sequence of commands. The internal command buffer has a size of 131070 bytes (128 kB).

FTC_STATUS **JTAG_AddDeviceReadCmd** (FTC_HANDLE *ftHandle*, BOOL *blInstructionTestData*, DWORD *dwNumBitsToRead*, DWORD *dwTapControllerState*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>blInstructionTestData</i>	Selects the instruction register (TRUE) or the test register (FALSE) to write data to.
<i>dwNumBitsToRead</i>	Number of bits to be read from the external device. Valid range is 2 to 524280.
<i>dwTapControllerState</i>	State that the Test Access Port (TAP) controller will be left in.

Return Value

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

FTC_INVALID_HANDLE
 FTC_INVALID_NUMBER_BITS
 FTC_INVALID_TAP_CONTROLLER_STATE
 FTC_COMMAND_SEQUENCE_BUFFER_FULL

Remarks

Do not invoke JTAG_Write^[18], JTAG_READ^[20] or JTAG_WriteRead^[21] 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. If only one device is connected, JTAG_AddReadCmd^[30] may be used instead.

Calling this function is equivalent to adding the commands and data from a JTAG_Read^[20] call to the internal command buffer.

This function can be used with JTAG_ClearDeviceCmdSequence^[25], JTAG_AddDeviceWriteCmd^[28], JTAG_AddDeviceWriteReadCmd^[34] and JTAG_ExecuteCmdSequence^[36] 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.

2.22 JTAG_AddWriteReadCmd

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

FTC_STATUS **JTAG_AddWriteReadCmd** (BOOL *bInstructionTestData*, DWORD *dwNumBitsToWriteRead*, PWriteDataByteBuffer *pWriteDataBuffer*, DWORD *dwNumBytesToWrite*, DWORD *dwTapControllerState*)

Parameters

<i>bInstructionTestData</i>	Selects the instruction register (TRUE) or the test register (FALSE) to write data to.
<i>dwNumBitsToWriteRead</i>	Number of bits to be written to and read from an external device. Valid range is 2 to 524280.
<i>pWriteDataBuffer</i>	Pointer to buffer that contains the data to be written to the external device.
<i>dwNumBytesToWrite</i>	Number of bytes in the write data buffer which contains all the specified bits to be written to the external device. Valid range is 1 to 65535 bytes.
<i>dwTapControllerState</i>	State that the Test Access Port (TAP) controller will be left in.

Return Value

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

FTC_INVALID_NUMBER_BITS
 FTC_NULL_WRITE_DATA_BUFFER_POINTER
 FTC_INVALID_NUMBER_BYTES
 FTC_NUMBER_BYTES_TOO_SMALL
 FTC_INVALID_TAP_CONTROLLER_STATE
 FTC_COMMAND_SEQUENCE_BUFFER_FULL

Remarks

Do not invoke JTAG_Write^[18], JTAG_READ^[20] or JTAG_WriteRead^[21] 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 is equivalent to adding the commands and data from a JTAG_WriteRead^[21] call to the internal command buffer.

This function can be used with JTAG_ClearCmdSequence^[24], JTAG_AddWriteCmd^[26], JTAG_AddReadCmd^[30] and JTAG_ExecuteCmdSequence^[36] 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;
WriteDataByteBuffer WriteDataBuffer;

// Add values to write buffer
WriteDataBuffer[0] = '\x00';
WriteDataBuffer[1] = '\x00';
WriteDataBuffer[2] = '\x00';
WriteDataBuffer[3] = '\x00';

// Write buffer to device
Status = JTAG_AddWriteReadCmd(true, 32, &WriteDataBuffer, 4, RUN_TEST_IDLE_STATE);
```

2.23 JTAG_AddDeviceWriteReadCmd

Adds a write/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, read and write/read before executing the sequence of commands. The internal command buffer has a size of 131070 bytes (128 kB).

FTC_STATUS **JTAG_AddDeviceWriteReadCmd** (FTC_HANDLE *ftHandle*, BOOL *bInstructionTestData*, DWORD *dwNumBitsToWriteRead*, PWriteDataByteBuffer *pWriteDataBuffer*, DWORD *dwNumBytesToWrite*, DWORD *dwTapControllerState*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>bInstructionTestData</i>	Selects the instruction register (TRUE) or the test register (FALSE) to write data to.
<i>dwNumBitsToWriteRead</i>	Number of bits to be written to and read from an external device. Valid range is 2 to 524280.
<i>pWriteDataBuffer</i>	Pointer to buffer that contains the data to be written to the external device.
<i>dwNumBytesToWrite</i>	Number of bytes in the write data buffer which contains all the specified bits to be written to the external device. Valid range is 1 to 65535 bytes.
<i>dwTapControllerState</i>	State that the Test Access Port (TAP) controller will be left in.

Return Value

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

FTC_INVALID_HANDLE
 FTC_INVALID_NUMBER_BITS
 FTC_NULL_WRITE_DATA_BUFFER_POINTER
 FTC_INVALID_NUMBER_BYTES
 FTC_NUMBER_BYTES_TOO_SMALL
 FTC_INVALID_TAP_CONTROLLER_STATE
 FTC_COMMAND_SEQUENCE_BUFFER_FULL

Remarks

Do not invoke [JTAG_Write](#)^[18], [JTAG_READ](#)^[20] or [JTAG_WriteRead](#)^[21] 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. If only one device is connected, [JTAG_AddDeviceWriteReadCmd](#)^[34] may be used instead.

Calling this function is equivalent to adding the commands and data from a [JTAG_WriteRead](#)^[21] call to the internal command buffer.

This function can be used with [JTAG_ClearDeviceCmdSequence](#)^[28], [JTAG_AddDeviceWriteCmd](#)^[28], [JTAG_AddDeviceReadCmd](#)^[31] and [JTAG_ExecuteCmdSequence](#)^[36] 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.

2.24 JTAG_ExecuteCmdSequence

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

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

Parameters

<i>ftHandle</i>	Handle of the device.
<i>pReadCmdSequenceBuffer</i>	Pointer to buffer that returns the data read from an external device. Size of buffer should be set to 131071 bytes (128KB).
<i>lpdwNumBytesReturned</i>	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 JTAG_Write^[18], JTAG_READ^[20] or JTAG_WriteRead^[21] 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 JTAG_ClearCmdSequence^[24], JTAG_AddWriteCmd^[26], JTAG_AddReadCmd^[30] and JTAG_AddWriteReadCmd^[32] to buffer a long list of commands and data which can then be sent to the FT2232C in one go. It can also be used with JTAG_ClearDeviceCmdSequence^[25], JTAG_AddDeviceWriteCmd^[28], JTAG_AddDeviceReadCmd^[31] and JTAG_AddDeviceWriteReadCmd^[34] when using multiple devices. 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 = JTAG_ExecuteCmdSequence(ftHandle, &ReadCmdSequenceDataBuffer,
```

```
&dwNumBytesReturned);
```

2.25 JTAG_GetDllVersion

Returns the version number of the current FTC_JTAG DLL.

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

Parameters

<i>lpDllVersionBuffer</i>	Pointer to the buffer that receives the version of this DLL. The string will be NULL terminated.
<i>dwBufferSize</i>	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 = JTAG_GetDllVersion(szDllVersion, 10);
```

2.26 JTAG_GetErrorCString

Provides an explanation of an error code.

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

Parameters

<i>lpLanguage</i>	Pointer to a NULL terminated string that contains the language code. Default for this first version the default language will be English (EN).
<i>StatusCode</i>	Status code returned from a previous FTC_JTAG DLL function call.
<i>lpErrorMessageBuffer</i>	Pointer to the buffer that receives the error code explanation string.
<i>dwBufferSize</i>	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 = JTAG_GetErrorCString("EN", Status, szErrorMessage, 100);
```


3 Appendix

3.1 Type Definitions

For Visual C++ applications, these values are pre-declared in the header file ([FTCJTAG.H](#)^[42]), 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_FREQUENCY_VALUE = 29
FTC_NULL_INPUT_OUTPUT_BUFFER_POINTER = 30
FTC_INVALID_NUMBER_BITS = 31
FTC_NULL_WRITE_DATA_BUFFER_POINTER = 32
FTC_INVALID_NUMBER_BYTES = 33
FTC_NUMBER_BYTES_TOO_SMALL = 34
FTC_INVALID_TAP_CONTROLLER_STATE = 35
FTC_NULL_READ_DATA_BUFFER_POINTER = 36
FTC_NULL_DLL_VERSION_BUFFER_POINTER = 37
FTC_DLL_VERSION_BUFFER_TOO_SMALL = 38
FTC_NULL_LANGUAGE_CODE_BUFFER_POINTER = 39
FTC_NULL_ERROR_MESSAGE_BUFFER_POINTER = 40
FTC_ERROR_MESSAGE_BUFFER_TOO_SMALL = 41
FTC_INVALID_LANGUAGE_CODE = 42
FTC_INVALID_STATUS_CODE = 43

```

TAP CONTROLLER STATES (DWORD)

```

TEST_LOGIC_STATE = 1
RUN_TEST_IDLE_STATE = 2

```

```
PAUSE_TEST_DATA_REGISTER_STATE = 3
PAUSE_INSTRUCTION_REGISTER_STATE = 4
SHIFT_TEST_DATA_REGISTER_STATE = 5
SHIFT_INSTRUCTION_REGISTER_STATE = 6
```

FTC_INPUT_OUTPUT_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;
```

FTC_LOW_HIGH_PINS

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

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;
```

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;
```

READ COMMAND SEQUENCE DATA BYTE BUFFER

```
#define MAX_READ_CMDS_DATA_BYTES_BUFFER_SIZE 131071    // 128K bytes
typedef BYTE
ReadCmdSequenceDataByteBuffer[MAX_READ_CMDS_DATA_BYTES_BUFFER_SIZE];
typedef ReadCmdSequenceDataByteBuffer *PReadCmdSequenceDataByteBuffer;
```

3.2 FTCJTAG.H

```
/*++
```

Copyright (c) 2005 Future Technology Devices International Ltd.

Module Name:

ftcjtag.h

Abstract:

API DLL for FT2232C Dual Device setup to simulate the Joint Test Action Group(JTAG) synchronous protocol.

FTCJTAG library definitions

Environment:

kernel & user mode

Revision History:

07/02/05 kra Created.

24/08/05 kra Added new function JTAG_GenerateClockPulses and new error code FTC_INVALID_NUMBER_CLOCK_PULSES

```
--*/
```

```
#ifndef FTCJTAG_H
#define FTCJTAG_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 FTCJTAG_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
// FTCJTAG_API functions as being imported from a DLL, whereas this DLL
// sees symbols defined with this macro as being exported.
```

```
#ifdef FTCJTAG_EXPORTS
#define FTCJTAG_API __declspec(dllexport)
#else
#define FTCJTAG_API __declspec(dllimport)
#endif
```

```
typedef DWORD FTC_HANDLE;
typedef ULONG FTC_STATUS;
```

```
#define TEST_LOGIC_STATE 1
#define RUN_TEST_IDLE_STATE 2
#define PAUSE_TEST_DATA_REGISTER_STATE 3
#define PAUSE_INSTRUCTION_REGISTER_STATE 4
#define SHIFT_TEST_DATA_REGISTER_STATE 5
#define SHIFT_INSTRUCTION_REGISTER_STATE 6
```

```
#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_OUTPUT_BUFFER_POINTER 30
#define FTC_INVALID_NUMBER_BITS 31
#define FTC_NULL_WRITE_DATA_BUFFER_POINTER 32
#define FTC_INVALID_NUMBER_BYTES 33
#define FTC_NUMBER_BYTES_TOO_SMALL 34
#define FTC_INVALID_TAP_CONTROLLER_STATE 35
#define FTC_NULL_READ_DATA_BUFFER_POINTER 36
#define FTC_COMMAND_SEQUENCE_BUFFER_FULL 37
#define FTC_NULL_READ_CMDS_DATA_BUFFER_POINTER 38
#define FTC_NO_COMMAND_SEQUENCE 39
#define FTC_INVALID_NUMBER_CLOCK_PULSES 40
#define FTC_NULL_DLL_VERSION_BUFFER_POINTER 41
#define FTC_DLL_VERSION_BUFFER_TOO_SMALL 42
#define FTC_NULL_LANGUAGE_CODE_BUFFER_POINTER 43
#define FTC_NULL_ERROR_MESSAGE_BUFFER_POINTER 44
#define FTC_ERROR_MESSAGE_BUFFER_TOO_SMALL 45
#define FTC_INVALID_LANGUAGE_CODE 46
#define FTC_INVALID_STATUS_CODE 47

#ifdef __cplusplus
extern "C" {
#endif

FTCJTAG_API
FTC_STATUS WINAPI JTAG_GetNumDevices(LPDWORD lpdwNumDevices);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_GetDeviceNameLocID(DWORD dwDeviceNameIndex, LPSTR
lpDeviceNameBuffer, DWORD dwBufferSize, LPDWORD lpdwLocationID);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_OpenEx(LPSTR lpDeviceName, DWORD dwLocationID,
```

```

FTC_HANDLE *pftHandle);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_Open(FTC_HANDLE *pftHandle);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_Close(FTC_HANDLE ftHandle);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_InitDevice(FTC_HANDLE ftHandle, DWORD dwClockDivisor);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_GetClock(DWORD dwClockDivisor, LPDWORD
lpdwClockFrequencyHz);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_SetClock(FTC_HANDLE ftHandle, DWORD dwClockDivisor,
LPDWORD lpdwClockFrequencyHz);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_SetLoopback(FTC_HANDLE ftHandle, BOOL bLoopbackState);

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;

FTCJTAG_API
FTC_STATUS WINAPI JTAG_SetGPIOs(FTC_HANDLE ftHandle, BOOL
bControlLowInputOutputPins,
                                PFTC_INPUT_OUTPUT_PINS pLowInputOutputPinsData,
                                BOOL bControlHighInputOutputPins,
                                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;

FTCJTAG_API
FTC_STATUS WINAPI JTAG_GetGPIOs(FTC_HANDLE ftHandle, BOOL
bControlLowInputOutputPins,
                                PFTC_LOW_HIGH_PINS pLowPinsInputData,
                                BOOL bControlHighInputOutputPins,
                                PFTC_LOW_HIGH_PINS pHighPinsInputData);

#define MAX_WRITE_DATA_BYTES_BUFFER_SIZE 65536 // 64k bytes

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

```

```
FTCJTAG_API
FTC_STATUS WINAPI JTAG_Write(FTC_HANDLE ftHandle, BOOL bInstructionTestData,
DWORD dwNumBitsToWrite,
    PWriteDataByteBuffer pWriteDataBuffer, DWORD dwNumBytesToWrite,
    DWORD dwTapControllerState);

#define MAX_READ_DATA_BYTES_BUFFER_SIZE 65536 // 64k bytes

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

FTCJTAG_API
FTC_STATUS WINAPI JTAG_Read(FTC_HANDLE ftHandle, BOOL bInstructionTestData,
DWORD dwNumBitsToRead,
    PReadDataByteBuffer pReadDataBuffer, LPDWORD lpdwNumBytesReturned,
    DWORD dwTapControllerState);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_WriteRead(FTC_HANDLE ftHandle, BOOL bInstructionTestData,
DWORD dwNumBitsToWriteRead,
    PWriteDataByteBuffer pWriteDataBuffer, DWORD dwNumBytesToWrite,
    PReadDataByteBuffer pReadDataBuffer, LPDWORD
lpdwNumBytesReturned,
    DWORD dwTapControllerState);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_GenerateClockPulses(FTC_HANDLE ftHandle, DWORD
dwNumClockPulses);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_ClearCmdSequence(void);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_AddWriteCmd(BOOL bInstructionTestData, DWORD
dwNumBitsToWrite,
    PWriteDataByteBuffer pWriteDataBuffer, DWORD dwNumBytesToWrite,
    DWORD dwTapControllerState);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_AddReadCmd(BOOL bInstructionTestData, DWORD
dwNumBitsToRead, DWORD dwTapControllerState);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_AddWriteReadCmd(BOOL bInstructionTestData, DWORD
dwNumBitsToWriteRead,
    PWriteDataByteBuffer pWriteDataBuffer, DWORD dwNumBytesToWrite,
    DWORD dwTapControllerState);

#define MAX_READ_CMDSDATA_BYTES_BUFFER_SIZE 131071 // 128K bytes

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

FTCJTAG_API
FTC_STATUS WINAPI JTAG_ClearDeviceCmdSequence(FTC_HANDLE ftHandle);

FTCJTAG_API
```

```
FTC_STATUS WINAPI JTAG_AddDeviceWriteCmd(FTC_HANDLE ftHandle, BOOL
blInstructionTestData, DWORD dwNumBitsToWrite,
                                     PWriteDataByteBuffer pWriteDataBuffer, DWORD dwNumBytesToWrite,
                                     DWORD dwTapControllerState);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_AddDeviceReadCmd(FTC_HANDLE ftHandle, BOOL
blInstructionTestData, DWORD dwNumBitsToRead, DWORD dwTapControllerState);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_AddDeviceWriteReadCmd(FTC_HANDLE ftHandle, BOOL
blInstructionTestData, DWORD dwNumBitsToWriteRead,
                                     PWriteDataByteBuffer pWriteDataBuffer, DWORD
dwNumBytesToWrite,
                                     DWORD dwTapControllerState);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_ExecuteCmdSequence(FTC_HANDLE ftHandle,
PReadCmdSequenceDataByteBuffer pReadCmdSequenceDataBuffer,
                                     LPDWORD lpdwNumBytesReturned);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_GetDIIVersion(LPSTR lpDIIVersionBuffer, DWORD dwBufferSize);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_GetErrorCodeString(LPSTR lpLanguage, FTC_STATUS
StatusCode,
                                     LPSTR lpErrorMessageBuffer, DWORD dwBufferSize);

#ifdef __cplusplus
}
#endif

#endif /* FTCJTAG_H */
```

Index

- F -

FT2232C 3
FTCJTAG Programmer's Guide 3
FTCJTAG.H 42

- I -

Introduction 3

- J -

JTAG 3
JTAG_AddDeviceReadCmd 31
JTAG_AddDeviceWriteCmd 28
JTAG_AddDeviceWriteReadCmd 34
JTAG_AddReadCmd 30
JTAG_AddWriteCmd 26
JTAG_AddWriteReadCmd 32
JTAG_ClearCmdSequence 24
JTAG_ClearDeviceCmdSequence 25
JTAG_Close 9
JTAG_ExecuteCmdSequence 36
JTAG_GenerateClockPulses 23
JTAG_GetClock 12
JTAG_GetDeviceNameLocID 6
JTAG_GetDIIVersion 38
JTAG_GetErrorCodeString 39
JTAG_GetGPIOs 15
JTAG_GetNumDevices 5
JTAG_InitDevice 10
JTAG_Open 7
JTAG_OpenEx 8
JTAG_Read 20
JTAG_SetClock 13
JTAG_SetGPIOs 16
JTAG_SetLoopback 14
JTAG_Write 18
JTAG_WriteRead 21

- M -

MPSSE 3

- T -

Type Definitions 40

- W -

Welcome 3