This document provides details of the function calls required when using the High Speed FTCJTAG DLL.
TABLE OF CONTENTS

1 Introduction ......................................................................................................................... 3
2 Application Programming Interface(API) .............................................................................. 4
  2.1 Public Functions .............................................................................................................. 4
    2.1.1 JTAG_GetNumDevices ............................................................................................... 4
    2.1.2 JTAG_GetNumHiSpeedDevices ............................................................................... 4
    2.1.3 JTAG_GetDeviceNameLocID .................................................................................... 5
    2.1.4 JTAG_GetHiSpeedDeviceNameLocIDChannel ......................................................... 6
    2.1.5 JTAG_Open ................................................................................................................. 7
    2.1.6 JTAG_OpenEx ............................................................................................................. 8
    2.1.7 JTAG_OpenHiSpeedDevice ....................................................................................... 9
    2.1.8 JTAG_GetHiSpeedDeviceType ............................................................................... 10
    2.1.9 JTAG_Close ................................................................................................................ 11
    2.1.10 JTAG_CloseDevice ................................................................................................. 11
    2.1.11 JTAG_InitDevice .................................................................................................... 11
    2.1.12 JTAG_TurnOnDivideByFiveClockingHiSpeedDevice ............................................ 12
    2.1.13 JTAG_TurnOffDivideByFiveClockingHiSpeedDevice ........................................... 13
    2.1.14 JTAG_TurnOnAdaptiveClockingHiSpeedDevice .................................................... 13
    2.1.15 JTAG_TurnOffAdaptiveClockingHiSpeedDevice .................................................. 13
    2.1.16 JTAG_SetDeviceLatencyTimer ............................................................................. 14
    2.1.17 JTAG_GetDeviceLatencyTimer ............................................................................. 14
    2.1.18 JTAG_GetClock ........................................................................................................ 15
    2.1.19 JTAG_GetHiSpeedDeviceClock ............................................................................ 16
    2.1.20 JTAG_SetClock ...................................................................................................... 17
    2.1.21 JTAG_SetLoopback .............................................................................................. 18
    2.1.22 JTAG_SetGPIOs .................................................................................................... 19
    2.1.23 JTAG_SetHiSpeedDeviceGPIOs ......................................................................... 20
    2.1.24 JTAG_GetGPIOs .................................................................................................... 22
    2.1.25 JTAG_GetHiSpeedDeviceGPIOs ......................................................................... 23
    2.1.26 JTAG_Write ........................................................................................................... 23
    2.1.27 JTAG_Read ............................................................................................................. 24
    2.1.28 JTAG_WriteRead ................................................................................................... 26
    2.1.29 JTAG_GenerateClockPulses ................................................................................. 28
    2.1.30 JTAG_ClearCmdSequence ..................................................................................... 29
    2.1.31 JTAG_AddWriteCmd ............................................................................................. 30
    2.1.32 JTAG_AddReadCmd .............................................................................................. 31
    2.1.33 JTAG_AddWriteReadCmd ..................................................................................... 32
    2.1.34 JTAG_ClearDeviceCmdSequence ......................................................................... 34
    2.1.35 JTAG_AddDeviceWriteCmd ................................................................................. 35
2.1.36  JTAG_AddDeviceReadCmd .................................................. 37
2.1.37  JTAG_AddDeviceWriteReadCmd .......................................... 38
2.1.38  JTAG_ExecuteCmdSequence ............................................. 40
2.1.39  JTAG_GetDllVersion ..................................................... 41
2.1.40  JTAG_GetErrorCodeString .............................................. 41

3  FTCJTAG.h ............................................................................. 43
4  JTAG TAP Controller State Diagram ....................................... 50
5  Contact Information ............................................................. 51
Appendix A – Revision History ................................................... 53
1 Introduction

The FT2232D, FT2232H and FT4232H devices 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 FT2232 JTAG API will provide a set of function's to allow a programmer to control the FT2232D dual device MPSSE controller, the FT2232H dual device MPSSE hi-speed controller and the FT4232H quad device MPSSE hi-speed controller, to communicate with other devices using the Joint Test Action Group(JTAG) synchronous serial protocol interface. The FT2232 JTAG API will be contained within the FTCJTAG.DLL.

The FTCJTAG DLL has been created to allow application developers to use the FT2232D, FT2232H and FT4232H devices 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.

The FT2232D MPSSE controller is only available through channel A of the FT2232D device; channel B of the FT2232D device does not support the MPSSE. Channel B may be controlled independently using FTDI's FTD2XX drivers while channel A is being used for JTAG communication.

The FT2232H MPSSE controller is available through channels A and B of the FT2232H device; both channels A and B can be used for JTAG communication.

The FT4232H MPSSE controller is only available through channels A and B of the FT4232H device; channels C and D of the FT4232H device do not support the MPSSE. Channels C and D may be controlled independently using FTDI's FTD2XX drivers while channels A and B are being used for JTAG communication.

This document lists all of the functions available in FTCJTAG.DLL.
2 Application Programming Interface (API)

2.1 Public Functions

The following section contains details of all the functions available in the FTCJTAG dll.

2.1.1 JTAG_GetNumDevices

FTC_STATUS JTAG_GetNumDevices(LPDWORD lpdwNumDevices)

This function must be used, if more than one FT2232D dual device will be connected to a system. This function returns the number of available FT2232D dual device(s) connected to a system.

Parameters

lpdwNumDevices Pointer to a variable of type DWORD which receives the actual number of available FT2232D dual device(s) connected to a system.

Return Value

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

FTC_IO_ERROR

2.1.2 JTAG_GetNumHiSpeedDevices

FTC_STATUS JTAG_GetNumHiSpeedDevices(LPDWORD lpdwTotalNumHiSpeedDevices)

This function must be used, if more than one FT2232H dual/FT4232H quad hi-speed devices will be connected to a system. This function returns the number of available FT2232H dual and FT4232H quad hi-speed device(s) connected to a system.

Parameters

lpdwTotalNumHiSpeedDevices Pointer to a variable of type DWORD which receives the total number of available FT2232H dual and FT4232H quad hi-speed device(s) connected to a system.

Return Value

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

FTC_IO_ERROR
2.1.3 JTAG_GetDeviceNameLocID

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

This function returns the name and the location identifier of the specified FT2232D dual device connected to a system.

Parameters

- **dwDeviceNameIndex**: Index of the FT2232D dual device. Use the FT2232D_GetNumDevices function call, see section 2.1.1, to get the number of available FT2232D dual device(s) connected to a system. Example: if the number of a specific FT2232D dual device returned is 2 then valid index values will be 0 and 1.

- **lpDeviceNameBuffer**: Pointer to buffer that receives the device name of the specified FT2232D dual 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 FT2232D dual device connected to a system.

Return Value

Returns FTC_SUCCESS if successful, otherwise the return value will be one of the following 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
2.1.4 JTAG_GetHiSpeedDeviceNameLocIDChannel

FTC_STATUS JTAG_GetHiSpeedDeviceNameLocIDChannel(DWORD dwDeviceNameIndex, LPSTR lpDeviceNameBuffer, DWORD dwDeviceNameBufferSize, LPDWORD lpdwLocationID, LPSTR lpChannelBuffer)

This function returns the name, location identifier and the channel of the specified FT2232H dual hi-speed device or FT4232H quad hi-speed device connected to a system.

Parameters

dwDeviceNameIndex: Index of the FT2232H dual hi-speed device or FT4232H quad hi-speed device. Use the JTAG_GetNumHiSpeedDevices function call, see section 2.1.2, to get the number of available FT2232H dual and FT4232H quad hi-speed device(s) connected to a system.

Example: if the number of FT2232H dual and FT4232H quad hi-speed device(s) returned is 2 then valid index values will be 0 and 1.

lpDeviceNameBuffer: Pointer to buffer that receives the device name of the specified FT2232H dual hi-speed device or FT4232H quad hi-speed device connected to a system. The string will be NULL terminated.

dwDeviceNameBufferSize: Length of the buffer created for the device name string. Set buffer length to a minimum of 100 characters.

lpdwLocationID: Pointer to a variable of type DWORD which receives the location identifier of the specified FT2232H dual hi-speed device or FT4232H quad hi-speed device connected to a system.

lpChannelBuffer: Pointer to a buffer that receives the channel of the specified FT2232H dual hi-speed device or FT4232H quad hi-speed device connected to a system. The buffer will only return a single character either A or B. The string will be NULL terminated.

dwChannelBufferSize: Length of the buffer created for the channel string. Set buffer length to a minimum of 5 characters.

lpdwHiSpeedDeviceType: Pointer to a variable of type DWORD which receives the actual type of hi-speed device, FT2232H dual hi-speed or FT4232H quad hi-speed.

Valid Hi-Speed Device Types
FT2232H_DEVICE_TYPE
FT4232H_DEVICE_TYPE

Return Value

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

FTC_DEVICE_NOT_FOUND
FTC_INVALID_DEVICE_NAME_INDEX
FTC_NULL_DEVICE_NAME_BUFFER_POINTER
FTC_DEVICE_NAME_BUFFER_TOO_SMALL
FTC_NULL_CHANNEL_BUFFER_POINTER
FTC_CHANNEL_BUFFER_TOO_SMALL
FTC_IO_ERROR
2.1.5 JTAG_Open

FTC_STATUS JTAG_Open(FTC_HANDLE *pftHandle)

This function must only be used, if a maximum of one FT2232D dual device will be connected to a system.

This function first determines which attached application is invoking this function. If an attached application invokes this function again and it’s assigned handle is still open then it’s assigned handle will be returned again. If another application attempts to open this device, which is already in use, an error code is returned. This function first then determines if a FT2232D dual device is present then checks that an application is not already using this FT2232D dual device. If another application is not using this FT2232D dual device then an attempt is made to open it. If the open was not successful an error code will be returned. If the open is successful, the FT2232D dual device is initialized to its default state, see section 2.1.14. If the initialization was successful the handle is passed back to the application. If the initialization was not successful an error code will be returned.

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

Returns FTC_SUCCESS if successful, otherwise the return value will be one of the following 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
2.1.6 JTAG_OpenEx

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

This function first determines which attached application is invoking this function. If an attached application invokes this function again and it's assigned handle is still open then it's assigned handle will be returned again. If another application attempts to open this device, which is already in use, an error code is returned. This function first determines if the specified FT2232D dual device is present then checks that an application is not already using the specified FT2232D dual device. If another application is not using the specified FT2232D dual device then an attempt is made to open it. If the open was not successful an error code will be returned. If the open is successful, the specified FT2232D dual device is initialized to its default state, see section 2.1.14. If the initialization was successful the handle is passed back to the application. If the initialization was not successful an error code will be returned.

Parameters

lpDeviceName Pointer to a NULL terminated string that contains the name of the specified FT2232D dual device to be opened.

dwLocationID Specifies the location identifier of the specified FT2232D dual 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

Returns FTC_SUCCESS if successful, otherwise the return value will be one of the following 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
2.1.7 JTAG_OpenHiSpeedDevice

FTC_STATUS JTAG_OpenHiSpeedDevice (LPSTR lpDeviceName, DWORD dwLocationID, LPSTR lpChannel, FTC_HANDLE *pftHandle)

This function first determines which attached application is invoking this function. If an attached application invokes this function again and it's assigned handle is still open then it's assigned handle will be returned again. If another application attempts to open this device, which is already in use, an error code is returned. This function first determines if the specified FT2232H dual hi-speed device or FT4232H quad hi-speed device is present then checks that an application is not already using the specified FT2232H dual hi-speed device or FT4232H quad hi-speed device. If another application is not using the specified FT2232H dual hi-speed device or FT4232H quad hi-speed device then an attempt is made to open it. If the open was not successful an error code will be returned. If the open is successful, the specified FT2232H dual hi-speed device or FT4232H quad hi-speed device is initialized to its default state, see section 2.1.14. If the initialization was successful the handle is passed back to the application. If the initialization was not successful an error code will be returned.

Parameters

lpDeviceName Pointer to a NULL terminated string that contains the name of the specified FT2232H dual hi-speed device or FT4232H quad hi-speed device to be opened.

dwLocationID Specifies the location identifier of the specified FT2232H dual hi-speed device or FT4232H quad hi-speed device to be opened.

lpChannel Pointer to a NULL terminated string that contains the channel of the specified FT2232H dual hi-speed device or FT4232H quad hi-speed device to be opened.

The channel identifier will be a single character either A or B.

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

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

FTC_NULL_DEVICE_NAME_BUFFER_POINTER
FTC_INVALID_DEVICE_NAME
FTC_INVALID_LOCATION_ID
FTC_INVALID_CHANNEL
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
2.1.8 JTAG_GetHiSpeedDeviceType

FTC_STATUS JTAG_GetHiSpeedDeviceType (FTC_HANDLE ftHandle, LPDWORD lpdwHiSpeedDeviceType)

This function returns the high speed device type detected. The type should either be FT2232H or FT4232H.

**Parameters**

- `ftHandle` Handle of the FT2232H dual hi-speed device or FT4232H quad hi-speed device opened.
- `lpdwHiSpeedDeviceType` Pointer to a variable of type DWORD which receives the device type.

**Valid Hi-Speed Device Types**

- FT2232H_DEVICE_TYPE
- FT4232HDEVICE_TYPE

**Return Value**

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

- FTC_INVALID_HANDLE
- FTC_IO_ERROR
2.1.9 JTAG_Close

FTC_STATUS JTAG_Close(FTC_HANDLE ftHandle)

This function closes a previously opened handle to a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

Parameters

ftHandle Handle of the FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device to close.

Return Value

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

FTC_INVALID_HANDLE
FTC_IO_ERROR

2.1.10 JTAG_CloseDevice

FTC_STATUS JTAG_CloseDevice (FTC_HANDLE ftHandle, PFTC_CLOSE_FINAL_STATE_PINS pCloseFinalStatePinsData)

This function closes a previously opened handle to a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

Parameters

ftHandle Handle of the FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device to close.

pCloseFinalStatePinsData Pointer to the structure that contains the data that is used to set the final state of output pins TCK, TDI, TMS

Return Value

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

FTC_INVALID_HANDLE
FTC_IO_ERROR

2.1.11 JTAG_InitDevice

FTC_STATUS JTAG_InitDevice(FTC_HANDLE ftHandle, DWORD dwClockDivisor)

This function initializes the FT2232D dual device, by carrying out the following in the following order:

- 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
- 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 on an external device
set the Test Access Port(TAP) controller on an external device to test idle mode

Parameters

ftHandle Handle of a FT2232D dual device.
dwClockDivisor Specifies a divisor, which will be used to set the frequency that will be used to clock data in and out of a FT2232D dual device. Valid range is 0 to 65535. The highest clock frequency is represented by 0, which is equivalent to 6MHz, the next highest clock frequency is represented by 1, which is equivalent to 3MHz and the lowest clock frequency is represented by 65535, which is equivalent to 91Hz. To obtain the actual frequency in Hz, represented by the specified divisor, see section 2.1.18.

Note: the frequency in Hz, represented by the divisor, is calculated using the following formula:

\[ \text{frequency} = \frac{12\text{MHz}}{((1 + \text{dwClockDivisor}) \times 2)} \]

Return Value

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

### 2.1.12 JTAG_TurnOnDivideByFiveClockingHiSpeedDevice

FTC_STATUS JTAG_TurnOnDivideByFiveClockingHiSpeedDevice (FTC_HANDLE fthandle)

This function turns on the divide by five for the MPSSE clock to allow the hi-speed devices FT2232H and FT4232H to clock at the same rate as the FT2232D device. This allows for backward compatibility.

Parameters

- fthandle Handle of a FT2232H dual hi-speed device or FT4232H quad hi-speed device.

Return Value

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

- FTC_INVALID_HANDLE
- FTC_IO_ERROR
2.1.13 JTAG_TurnOffDivideByFiveClockingHiSpeedDevice

FTC_STATUS JTAG_TurnOffDivideByFiveClockingHiSpeedDevice (FTC_HANDLE fthandle)

This function turns off the divide by five for the MPSSE clock to allow the hi-speed devices FT2232H and FT4232H to clock at the higher speeds. Maximum is 30Mbit/s.

Parameters
- fthandle: Handle of a FT2232H dual hi-speed device or FT4232H quad hi-speed device.

Return Value
- Returns FTC_SUCCESS if successful, otherwise the return value will be one of the following error codes:
  - FTC_INVALID_HANDLE
  - FTC_IO_ERROR

2.1.14 JTAG_TurnOnAdaptiveClockingHiSpeedDevice

FTC_STATUS JTAG_TurnOnAdaptiveClockingHiSpeedDevice (FTC_HANDLE ftHandle)

This function turns on adaptive clocking for a FT2232H dual hi-speed device or FT4232H quad hi-speed device.

Parameters
- ftHandle: Handle of a FT2232H dual hi-speed device or FT4232H quad hi-speed device.

Return Value
- Returns FTC_SUCCESS if successful, otherwise the return value will be one of the following error codes:
  - FTC_INVALID_HANDLE
  - FTC_IO_ERROR

2.1.15 JTAG_TurnOffAdaptiveClockingHiSpeedDevice

FTC_STATUS JTAG_TurnOffAdaptiveClockingHiSpeedDevice (FTC_HANDLE ftHandle)

This function turns off adaptive clocking for a FT2232H dual hi-speed device or FT4232H quad hi-speed device.

Parameters
- ftHandle: Handle of a FT2232H dual hi-speed device or FT4232H quad hi-speed device.

Return Value
- Returns FTC_SUCCESS if successful, otherwise the return value will be one of the following error codes:
  - FTC_INVALID_HANDLE
  - FTC_IO_ERROR
2.1.16  JTAG_SetDeviceLatencyTimer

FTC_STATUS JTAG_SetDeviceLatencyTimer(FTC_HANDLE ftHandle, BYTE timerValue)

This function sets the value in milliseconds of the latency timer for a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device. The latency timer is used to flush any remaining data received from a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device from the USB input buffer, when the latency timer times out.

Parameters

ftHandle Handle of a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

timerValue Specifies the value, in milliseconds, of the latency timer. Valid range is 2 - 255.

Return Value

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

FTC_INVALID_HANDLE
FTC_INVALID_TIMER_VALUE
FTC_IO_ERROR

2.1.17  JTAG_GetDeviceLatencyTimer

FTC_STATUS JTAG_GetDeviceLatencyTimer(FTC_HANDLE ftHandle, LPBYTE lpTimerValue)

This function gets the value in milliseconds of the latency timer for a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device. The latency timer is used to flush any remaining data received from a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device from the USB input buffer, when the latency timer times out.

Parameters

ftHandle Handle of a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

lpTimerValue Pointer to a variable of type BYTE which receives the actual latency timer value in milliseconds.

Return Value

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

FTC_INVALID_HANDLE
FTC_IO_ERROR
2.1.18 JTAG_GetClock

FTC_STATUS JTAG_GetClock (DWORD dwClockDivisor, LPDWORD lpdwClockFrequencyHz)

This function calculates the frequency in Hz, that data will be clocked in and out of a FT2232D dual device.

Parameters

- dwClockDivisor: Specifies a divisor, which will be used to set the frequency that will be used to clock data in and out of a FT2232D dual device. Valid range is 0 to 65535. The highest clock frequency is represented by 0, which is equivalent to 6MHz, the next highest clock frequency is represented by 1, which is equivalent to 3MHz and the lowest clock frequency is represented by 65535, which is equivalent to 91Hz.

- lpdwClockFrequencyHz: Pointer to a variable of type DWORD which receives the actual frequency in Hz, that data will be clocked in and out of a FT2232D dual device.

Note: the frequency in Hz, represented by the divisor, is calculated using the following formula:

\[
\text{frequency} = \frac{12\text{MHz}}{(1 + \text{dwClockDivisor}) \times 2}.
\]

Return Value

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

- FTC_INVALID_CLOCK_DIVISOR
2.1.19 JTAG_GetHiSpeedDeviceClock

FTC_STATUS JTAG_GetHiSpeedDeviceClock (DWORD dwClockDivisor, LPDWORD lpdwClockFrequencyHz)

This function calculates the frequency in Hz, that data will be clocked in and out of a FT2232H dual hi-speed device or FT4232H quad hi-speed device.

Parameters

\( \text{dwClockDivisor} \) Specifies a divisor, which will be used to set the frequency that will be used to clock data in and out of a FT2232H dual hi-speed device or FT4232H quad hi-speed device. Valid range is 0 to 65535. The highest clock frequency is represented by 0, which is equivalent to 30MHz, the next highest clock frequency is represented by 1, which is equivalent to 15MHz and the lowest clock frequency is represented by 65535, which is equivalent to 457Hz.

\( \text{lpdwClockFrequencyHz} \) Pointer to a variable of type DWORD which receives the actual frequency in Hz, that data will be clocked in and out of a FT2232H dual hi-speed device or FT4232H quad hi-speed device.

Note: the frequency in Hz, represented by the divisor, is calculated using the following formula:

\[
\text{frequency} = \frac{60\text{MHz}}{(1 + \text{dwClockDivisor}) \times 2}.
\]

Return Value

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

- FTC_INVALID_CLOCK_DIVISOR
2.1.20 JTAG_SetClock

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

This function sets and calculates the frequency in Hz, that data will be clocked in and out of a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

Parameters

ftHandle Handle of a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

dwClockDivisor Specifies a divisor, which will be used to set the frequency that will be used to clock data in and out of a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device. Valid range is 0 to 65535. The highest clock frequency is represented by 0, which is equivalent to 6MHz for the FT2232D dual device and 30MHz for the FT2232H dual and FT4232H quad hi-speed devices, the next highest clock frequency is represented by 1, which is equivalent to 3MHz for the FT2232D dual device and 15MHz for the FT2232H dual and FT4232H quad hi-speed devices and the lowest clock frequency is represented by 65535, which is equivalent to 91Hz for the FT2232D dual device and 457Hz for the FT2232H dual and FT4232H quad hi-speed devices.

lpdwClockFrequencyHz Pointer to a variable of type DWORD which receives the actual frequency in Hz, that data will be clocked in and out of a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

For the FT2232D dual device the frequency in Hz, represented by the divisor, is calculated using the following formula:

\[\text{frequency} = \frac{12\text{MHz}}{(1 + \text{dwClockDivisor}) \times 2}\]

For the FT2232H dual and FT4232H quad hi-speed devices the frequency in Hz, represented by the divisor, is calculated using the following formula:

\[\text{frequency} = \frac{60\text{MHz}}{(1 + \text{dwClockDivisor}) \times 2}\]

Return Value

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

- FTC_INVALID_HANDLE
- FTC_INVALID_CLOCK_DIVISOR
- FTC_FAILED_TO_COMPLETE_COMMAND
- FTC_IO_ERROR
2.1.21 JTAG_SetLoopback

FTC_STATUS JTAG_SetLoopback(FTC_HANDLE ftHandle, BOOL bLoopbackState)

This function controls the state of the FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device loopback. The FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device is set to loopback for testing purposes.

Parameters

ftHandle Handle of the FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

bLoopbackState Controls the state of the FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device loopback. To switch loopback on(TRUE) or off(FALSE).

Return Value

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

FTC_INVALID_HANDLE
FTC_FAILED_TO_COMPLETE_COMMAND
FTC_IO_ERROR
2.1.22  **JTAG_SetGPIOs**

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

This function controls the use of the 8 general purpose input/output pins (GPIOL1 – GPIOL4 and GPIOH1 – GPIOH4) of the FT2232D dual device.

**Parameters**

- **ftHandle**  Handle of a FT2232D dual device.
- **bControlLowInputOutputPins**  Controls the use of the 4 general purpose lower input/output pins (GPIOL1 – GPIOL4) of the FT2232D dual device. To control the 4 lower input/output pins (TRUE) or to not control the 4 lower input/output pins (FALSE).
- **pLowInputOutputPinsData**  Pointer to the structure that contains the data that is used to control the 4 general purpose lower input/output pins (GPIOL1 – GPIOL4) of the FT2232D dual device.
- **bControlHighInputOutputPins**  Controls the use of the 4 general purpose higher input/output pins (GPIOH1 – GPIOH4) of the FT2232D dual device. To control the 4 higher input/output pins (TRUE) or to not control the 4 higher input/output pins (FALSE).
- **pHighInputOutputPinsData**  Pointer to the structure that contains the data that is used to control the 4 general purpose higher input/output pins (GPIOH1 – GPIOH4) of the FT2232D dual device.

**Return Value**

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

- FTC_INVALID_HANDLE
- FTC_NULL_INPUT_OUTPUT_BUFFER_POINTER
- FTC_FAILED_TO_COMPLETE_COMMAND
- FTC_IO_ERROR

**Example:**

```c
typedef struct FTC_Input_Output_Pins {
    BOOL bPin1InputOutputState;  Set pin1 to input mode (FALSE), set pin1 to output mode (TRUE)
    BOOL bPin1LowHighState;  If pin1 is set to output mode, set pin1 low (FALSE), high (TRUE)
    BOOL bPin2InputOutputState;  Set pin2 to input mode (FALSE), set pin2 to output mode (TRUE)
    BOOL bPin2LowHighState;  If pin2 is set to output mode, set pin2 low (FALSE), high (TRUE)
    BOOL bPin3InputOutputState;  Set pin3 to input mode (FALSE), set pin3 to output mode (TRUE)
    BOOL bPin3LowHighState;  If pin3 is set to output mode, set pin3 low (FALSE), high (TRUE)
    BOOL bPin4InputOutputState;  Set pin4 to input mode (FALSE), set pin4 to output mode (TRUE)
    BOOL bPin4LowHighState;  If pin4 is set to output mode, set pin4 low (FALSE), high (TRUE)
} FTC_INPUT_OUTPUT_PINS *PFTC_INPUT_OUTPUT_PINS
```
2.1.23 JTAG_SetHiSpeedDeviceGPIOs

```
FTC_STATUS JTAG_SetHiSpeedDeviceGPIOs(FTC_HANDLE ftHandle, BOOL bControlLowInputOutputPins, PFTC_INPUT_OUTPUT_PINS pLowInputOutputPinsData, BOOL bControlHighInputOutputPins, PFTH_INPUT_OUTPUT_PINS pHighInputOutputPinsData)
```

This function controls the use of the 12 general purpose input/output pins (GPIOL1 – GPIOL4 and GPIOH1 – GPIOH8) of the FT2232H dual hi-speed device or the 4 general purpose lower input/output pins (GPIOL1 – GPIOL4) of the FT4232H quad hi-speed device.

### Parameters

- **ftHandle**
  - Handle of the FT2232H dual hi-speed device or FT4232H quad hi-speed device.

- **bControlLowInputOutputPins**
  - Controls the use of the 4 general purpose lower input/output pins (GPIOL1 – GPIOL4) of the FT2232H dual hi-speed device or FT4232H quad hi-speed device. To control the 4 lower input/output pins(TRUE) or to not control the 4 lower input/output pins(FALSE).

- **pLowInputOutputPinsData**
  - Pointer to the structure that contains the data that is used to control the 4 general purpose lower input/output pins (GPIOL1 – GPIOL4) of the FT2232H dual hi-speed device or FT4232H quad hi-speed device.

- **bControlHighInputOutputPins**
  - Controls the use of the 8 general purpose higher input/output pins (GPIOH1 – GPIOH8) of the FT2232H dual hi-speed device. To control the 8 higher input/output pins(TRUE) or to not control the 8 higher input/output pins(FALSE).

- **pHighInputOutputPinsData**
  - Pointer to the structure that contains the data that is used to control the general purpose 8 higher input/output pins (GPIOH1 – GPIOH8) of the FT2232H dual hi-speed device.

  **Note:** the 8 general purpose higher input/output pins (GPIOH1 – GPIOH8) do not physically exist on the FT4232H quad hi-speed device.

### Return Value

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

- FTC_INVALID_HANDLE
- FTC_NULL_INPUT_OUTPUT_BUFFER_POINTER
- FTC_FAILED_TO_COMPLETE_COMMAND
- FTC_IO_ERROR
Example:

typedef struct FTC_Input_Output_Pins {
    BOOL bPin1InputOutputState;  // Set pin1 to input mode(FALSE), set pin1 to output mode(TRUE)
    BOOL bPin1LowHighState;     // If pin1 is set to output mode, set pin1 low(FALSE), high(TRUE)
    BOOL bPin2InputOutputState; // Set pin2 to input mode(FALSE), set pin2 to output mode(TRUE)
    BOOL bPin2LowHighState;     // If pin2 is set to output mode, set pin2 low(FALSE), high(TRUE)
    BOOL bPin3InputOutputState; // Set pin3 to input mode(FALSE), set pin3 to output mode(TRUE)
    BOOL bPin3LowHighState;     // If pin3 is set to output mode, set pin3 low(FALSE), high(TRUE)
    BOOL bPin4InputOutputState; // Set pin4 to input mode(FALSE), set pin4 to output mode(TRUE)
    BOOL bPin4LowHighState;     // If pin4 is set to output mode, set pin4 low(FALSE), high(TRUE)
} FTC_INPUT_OUTPUT_PINS *PFTC_INPUT_OUTPUT_PINS

typedef struct FTH_Input_Output_Pins {
    BOOL bPin1InputOutputState;  // Set pin1 to input mode(FALSE), set pin1 to output mode(TRUE)
    BOOL bPin1LowHighState;     // If pin1 is set to output mode, set pin1 low(FALSE), high(TRUE)
    BOOL bPin2InputOutputState; // Set pin2 to input mode(FALSE), set pin2 to output mode(TRUE)
    BOOL bPin2LowHighState;     // If pin2 is set to output mode, set pin2 low(FALSE), high(TRUE)
    BOOL bPin3InputOutputState; // Set pin3 to input mode(FALSE), set pin3 to output mode(TRUE)
    BOOL bPin3LowHighState;     // If pin3 is set to output mode, set pin3 low(FALSE), high(TRUE)
    BOOL bPin4InputOutputState; // Set pin4 to input mode(FALSE), set pin4 to output mode(TRUE)
    BOOL bPin4LowHighState;     // If pin4 is set to output mode, set pin4 low(FALSE), high(TRUE)
    BOOL bPin5InputOutputState; // Set pin5 to input mode(FALSE), set pin5 to output mode(TRUE)
    BOOL bPin5LowHighState;     // If pin5 is set to output mode, set pin5 low(FALSE), high(TRUE)
    BOOL bPin6InputOutputState; // Set pin6 to input mode(FALSE), set pin6 to output mode(TRUE)
    BOOL bPin6LowHighState;     // If pin6 is set to output mode, set pin6 low(FALSE), high(TRUE)
    BOOL bPin7InputOutputState; // Set pin7 to input mode(FALSE), set pin7 to output mode(TRUE)
    BOOL bPin7LowHighState;     // If pin7 is set to output mode, set pin7 low(FALSE), high(TRUE)
    BOOL bPin8InputOutputState; // Set pin8 to input mode(FALSE), set pin8 to output mode(TRUE)
    BOOL bPin8LowHighState;     // If pin8 is set to output mode, set pin8 low(FALSE), high(TRUE)
} FTH_INPUT_OUTPUT_PINS *PFTH_INPUT_OUTPUT_PINS
2.1.24 **JTAG_GetGPIOs**

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

This function gets the input states (low or high) of the 8 general purpose input/output pins (GPIO1 – GPIO4 and GPIOH1 – GPIOH4) of the FT2232D dual device.

**Parameters**

- **ftHandle**
  Handle of a FT2232D dual device.

- **bControlLowInputOutputPins**
  Controls the use of the 4 general purpose lower input/output pins (GPIO1 – GPIO4) of the FT2232D dual device. To enable the 4 lower input/output pins to be read(TRUE) or to disable the 4 lower input/output pins from being read(FALSE).

- **pLowPinsInputData**
  Pointer to the structure that contains the input states (low or high) of the 4 general purpose lower input/output pins (GPIO1 – GPIO4) of the FT2232D dual device.

- **bControlHighInputOutputPins**
  Controls the use of the 4 general purpose higher input/output pins (GPIOH1 – GPIOH4) of the FT2232D dual device. To enable the 4 higher input/output pins to be read(TRUE) or to disable the 4 higher input/output pins from being read(FALSE).

- **pHighPinsInputData**
  Pointer to the structure that contains the input states (low or high) of the 4 general purpose higher input/output pins (GPIOH1 – GPIOH4) of the FT2232D dual device.

**Return Value**

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

- FTC_INVALID_HANDLE
- FTC_NULL_INPUT_OUTPUT_BUFFER_POINTER
- FTC_FAILED_TO_COMPLETE_COMMAND
- FTC_IO_ERROR

**Example:**

```c
typedef struct FTC_Low_High_Pins {
    BOOL bPin1LowHighState;  // Pin1 input state low(FALSE), high(TRUE)
    BOOL bPin2LowHighState;  // Pin2 input state low(FALSE), high(TRUE)
    BOOL bPin3LowHighState;  // Pin3 input state low(FALSE), high(TRUE)
    BOOL bPin4LowHighState;  // Pin4 input state low(FALSE), high(TRUE)
} FTC_LOW_HIGH_PINS *PFTC_LOW_HIGH_PINS
```
### 2.1.25 JTAG\_GetHiSpeedDeviceGPIOs

FTC\_STATUS JTAG\_GetHiSpeedDeviceGPIOs(FTC\_HANDLE ftHandle, BOOL bControlLowInputOutputPins, PFTC\_LOW\_HIGH\_PINS pLowPinsInputData, BOOL bControlHighInputOutputPins, PFTH\_LOW\_HIGH\_PINS pHighPinsInputData)

This function gets the input states(low or high) of the 12 general purpose input/output pins (GPIOL1 – GPIOL4 and GPIOH1 – GPIOH8) of the FT2232H dual hi-speed device or the 4 general purpose lower input/output pins (GPIOL1 – GPIOL4) of the FT4232H quad hi-speed device.

#### Parameters

**ftHandle**
Handle of the FT2232H dual hi-speed device or FT4232H quad hi-speed device.

**bControlLowInputOutputPins**
Controls the use of the 4 general purpose lower input/output pins (GPIOL1 – GPIOL4) of the FT2232H dual hi-speed device or FT4232H quad hi-speed device. To enable the 4 lower input/output pins to be read(TRUE) or to disable the 4 lower input/output pins from being read(FALSE).

**pLowPinsInputData**
Pointer to the structure that contains the input states(low or high) of the 4 general purpose lower input/output pins (GPIOL1 – GPIOL4) of the FT2232H dual hi-speed device or FT4232H quad hi-speed device.

**bControlHighInputOutputPins**
Controls the use of the 8 general purpose higher input/output pins (GPIOH1 – GPIOH8) of the FT2232H dual hi-speed device. To enable the 8 higher input/output pins to be read(TRUE) or to disable the 8 higher input/output pins from being read(FALSE).

**pHighPinsInputData**
Pointer to the structure that contains the input states(low or high) of the 8 general purpose higher input/output pins (GPIOH1 – GPIOH8) of the FT2232H dual hi-speed device.

#### Return Value

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

- FTC\_INVALID\_HANDLE
- FTC\_NULL\_INPUT\_OUTPUT\_BUFFER\_POINTER
- FTC\_FAILED\_TO\_COMPLETE\_COMMAND
- FTC\_IO\_ERROR

#### Example:

typedef struct FTC\_Low\_High\_Pins {
  BOOL bPin1LowHighState; // Pin1 input state low(FALSE), high(TRUE)
  BOOL bPin2LowHighState; // Pin2 input state low(FALSE), high(TRUE)
  BOOL bPin3LowHighState; // Pin3 input state low(FALSE), high(TRUE)
  BOOL bPin4LowHighState; // Pin4 input state low(FALSE), high(TRUE)
} FTC\_LOW\_HIGH\_PINS *PFTC\_LOW\_HIGH\_PINS

typedef struct FTH\_Low\_High\_Pins {
  BOOL bPin1LowHighState; // Pin1 input state low(FALSE), high(TRUE)
  BOOL bPin2LowHighState; // Pin2 input state low(FALSE), high(TRUE)
  BOOL bPin3LowHighState; // Pin3 input state low(FALSE), high(TRUE)
  BOOL bPin4LowHighState; // Pin4 input state low(FALSE), high(TRUE)
  BOOL bPin5LowHighState; // Pin5 input state low(FALSE), high(TRUE)
  BOOL bPin6LowHighState; // Pin6 input state low(FALSE), high(TRUE)
  BOOL bPin7LowHighState; // Pin7 input state low(FALSE), high(TRUE)
  BOOL bPin8LowHighState; // Pin8 input state low(FALSE), high(TRUE)
} FTH\_LOW\_HIGH\_PINS *PFTH\_LOW\_HIGH\_PINS
2.1.26 JTAG_Write

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

This function writes data to an external device i.e. a device attached to a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device. A FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device communicates with an external device by simulating the JTAG synchronous protocol.

Parameters

ftHandle Handle of the FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

bInstructionTestData Specifies the type of register, that data is to be written to on an external device. Instruction(TRUE) or test data(FALSE).

dwNumBitsToWrite Specifies the number of 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.

dwNumBytesToWrite Specifies the number of bytes in the write data buffer, which contains all the specified bits to be written to an external device. Valid range 1 to 65535 ie 64K bytes.

dwTapControllerState Specifies the state, the Test Access Port(TAP) controller will be left in after the data has been written to an external device.

Valid TAP Controller States

TEST_LOGIC_STATE
RUN_TEST_IDLE_STATE
PAUSE_TEST_DATA_REGISTER_STATE
PAUSE_INSTRUCTION_REGISTER_STATE
SHIFT_TEST_DATA_REGISTER_STATE
SHIFT_INSTRUCTION_REGISTER_STATE

Return Value

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

Example:

#define MAX_WRITE_DATA_BYTES_BUFFER_SIZE 65536 // 64K bytes
typedef BYTE WriteDataByteBuffer[MAX_WRITE_DATA_BYTES_BUFFER_SIZE];
typedef WriteDataByteBuffer *PWriteDataByteBuffer;
2.1.27  JTAG_Read

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

This function reads data from an external device ie a device attached to a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device. A FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device communicates with an external device by simulating the JTAG synchronous protocol.

Parameters

ftHandle  Handle of a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

bInstructionTestData  Specifies the type of register, that data is to be read from on an external device. Instruction(TRUE) or test data(FALSE).

dwNumBitsToRead  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.

lpdwNumBytesReturned  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.

dwTapControllerState  Specifies the state, the Test Access Port(TAP) controller will be left in after the data has been read from an external device.

Valid TAP Controller States

<table>
<thead>
<tr>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST_LOGIC_STATE</td>
</tr>
<tr>
<td>RUN_TEST_IDLE_STATE</td>
</tr>
<tr>
<td>PAUSE_TEST_DATA_REGISTER_STATE</td>
</tr>
<tr>
<td>PAUSE_INSTRUCTION_REGISTER_STATE</td>
</tr>
<tr>
<td>SHIFT_TEST_DATA_REGISTER_STATE</td>
</tr>
<tr>
<td>SHIFT_INSTRUCTION_REGISTER_STATE</td>
</tr>
</tbody>
</table>

Return Value

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

Example:

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

typedef BYTE ReadDataByteBuffer[MAX_READ_DATA_BYTES_BUFFER_SIZE];
typedef ReadDataByteBuffer *PReadDataByteBuffer;
```
2.1.28 JTAG_WriteRead

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

This function writes then read data to/from an external device ie a device attached to a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device. A FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device communicates with an external device by simulating the JTAG synchronous protocol.

Parameters

- ftHandle: Handle of a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.
- bInstructionTestData: Specifies the type of register, that data is to be written to and read from on an external device. Instruction(TRUE) or test data(FALSE).
- dwNumBitsToWriteRead: Specifies the number of bits to be written to and read from 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.
- dwNumBytesToWrite: Specifies the number of bytes in the write data buffer, which contains all the specified bits to be written to an external device. Valid range 1 to 65535 ie 64K bytes.
- pReadDataBuffer: Pointer to buffer that returns the data read from an external device. Size of buffer should be set to 65535.
- lpdwNumBytesReturned: 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.
- dwTapControllerState: Specifies the state, the Test Access Port(TAP) controller will be left in after the data has been written/read to/from an external device.

Valid TAP Controller States

<table>
<thead>
<tr>
<th>TEST_LOGIC_STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN_TEST_IDLE_STATE</td>
</tr>
<tr>
<td>PAUSE_TEST_DATA_REGISTER_STATE</td>
</tr>
<tr>
<td>PAUSE_INSTRUCTION_REGISTER_STATE</td>
</tr>
<tr>
<td>SHIFT_TEST_DATA_REGISTER_STATE</td>
</tr>
<tr>
<td>SHIFT_INSTRUCTION_REGISTER_STATE</td>
</tr>
</tbody>
</table>
Return Value

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

Example:

```c
#define MAX_WRITE_DATABYTES_BUFFER_SIZE 65536 // 64K bytes

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

#define MAX_READ_DATABYTES_BUFFER_SIZE 65536 // 64K bytes

typedef BYTEReadDataByteBuffer[MAX_READ_DATABYTES_BUFFER_SIZE];
typedef ReadDataByteBuffer *PReadDataByteBuffer;
```
### 2.1.29 JTAG\_GenerateClockPulses

FTC\_STATUS JTAG\_GenerateClockPulses (FTC\_HANDLE ftHandle, DWORD dwNumClockPulses)

This function instructs a FT2232D dual device to 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 ie a device attached to a FT2232D dual device during generation of the clock pulses will be 0. A FT2232D dual device communicates with an external device by simulating the JTAG synchronous protocol.

**Parameters**

- **ftHandle**: Handle of a FT2232D dual device.
- **dwNumClockPulses**: Specifies the number of clock pulses to be generated by a FT2232D dual device. Valid range 1 to 2000,000,000.

**Return Value**

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

- FTC\_INVALID\_HANDLE
- FTC\_INVALID\_NUMBER\_CLOCK\_PULSES
- FTC\_FAILED\_TO\_COMPLETE\_COMMAND
- FTC\_IO\_ERROR
2.1.30 JTAG_ClearCmdSequence

FTC_STATUS JTAG_ClearCmdSequence

This function must only be used, if a maximum of one device will be connected to a system ie FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

This function clears the sequence of commands and associated data from the internal command buffer.

Return Value

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

FTC_TOO_MANY_DEVICES
2.1.31  JTAG_AddWriteCmd

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

This function must only be used, if a maximum of one device will be connected to a system ie FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

This function adds a write command and associated data to the internal command buffer(size 131070 ie 128K bytes). This enables a programmer to build up a sequence of commands ie write, read and write/read, before executing the sequence of commands, see section 2.1.38.

Warning:  While constructing a sequence of commands, do not invoke JTAG_Write, JTAG_Read, JTAG_WriteRead or JTAG_GenerateClockPulses functions, as this will clear the sequence of commands and associated data from the internal command buffer.

Parameters

bInstructionTestData  Specifies the type of register, that data is to be written to on an external device. Instruction(TRUE) or test data(FALSE).

dwNumBitsToWrite  Specifies the number of 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.

dwNumBytesToWrite  Specifies the number of bytes in the write data buffer, which contains all the specified bits to be written to an external device. Valid range 1 to 65535 ie 64K bytes.

dwTapControllerState  Specifies the state, the Test Access Port(TAP) controller will be left in after the data has been written to an external device.

Valid TAP Controller States

<table>
<thead>
<tr>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST_LOGIC_STATE</td>
</tr>
<tr>
<td>RUN_TEST_IDLE_STATE</td>
</tr>
<tr>
<td>PAUSE_TEST_DATA_REGISTER_STATE</td>
</tr>
<tr>
<td>PAUSE_INSTRUCTION_REGISTER_STATE</td>
</tr>
<tr>
<td>SHIFT_TEST_DATA_REGISTER_STATE</td>
</tr>
<tr>
<td>SHIFT_INSTRUCTION_REGISTER_STATE</td>
</tr>
</tbody>
</table>

Return Value

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

FTC_TOO_MANY_DEVICES
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
Example:

```c
#define MAX_WRITE_DATA_BYTES_BUFFER_SIZE 65536 // 64K bytes
typedef BYTE WriteDataByteBuffer[MAX_WRITE_DATA_BYTES_BUFFER_SIZE];
typedef WriteDataByteBuffer *PWriteDataByteBuffer;
```

### 2.1.32 JTAG_AddReadCmd

```c
FTC_STATUS JTAG_AddReadCmd (BOOL bInstructionTestData, DWORD dwNumBitsToRead, DWORD dwTapControllerState)
```

This function must only be used, if a maximum of one device will be connected to a system ie FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

This function adds a read command to the internal command buffer(size 131070 ie 128K bytes). This enables a programmer to build up a sequence of commands ie write, read and write/read, before executing the sequence of commands, see section 2.1.38.

**Warning:** While constructing a sequence of commands, do not invoke JTAG_Write, JTAG_Read, JTAG_WriteRead or JTAG_GenerateClockPulses functions, as this will clear the sequence of commands and associated data from the internal command buffer.

**Parameters**

- `bInstructionTestData` Specifies the type of register, that data is to be read from on an external device. Instruction(TRUE) or test data(FALSE).

- `dwNumBitsToRead` Specifies the number of bits to be read from an external device. Valid range 2 to 524280. 524280 bits is equivalent to 64K bytes.

- `dwTapControllerState` Specifies the state, the Test Access Port(TAP) controller will be left in after the data has been read from an external device.

**Valid TAP Controller States**

- TEST_LOGIC_STATE
- RUN_TEST_IDLE_STATE
- PAUSE_TEST_DATA_REGISTER_STATE
- PAUSE_INSTRUCTION_REGISTER_STATE
- SHIFT_TEST_DATA_REGISTER_STATE
- SHIFT_INSTRUCTION_REGISTER_STATE

**Return Value**

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

- FTC_TOO_MANY_DEVICES
- FTC_INVALID_NUMBER_BITS
- FTC_INVALID_TAP_CONTROLLER_STATE
- FTC_COMMAND_SEQUENCE_BUFFER_FULL
- FTC_INSUFFICIENT_RESOURCES
2.1.33 JTAG_AddWriteReadCmd

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

This function must only be used, if a maximum of one device will be connected to a system ie FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

This function adds a write/read command and associated data to the internal command buffer(size 131070 ie 128K bytes). This enables a programmer to build up a sequence of commands ie write, read and write/read, before executing the sequence of commands, see section 2.1.38.

Warning: While constructing a sequence of commands, do not invoke JTAG_Write, JTAG_Read, JTAG_WriteRead or JTAG_GenerateClockPulses functions, as this will clear the sequence of commands and associated data from the internal command buffer.

Parameters

bInstructionTestData Specifies the type of register, that data is to be written to and read from on an external device. Instruction(TRUE) or test data(FALSE).

dwNumBitsToWriteRead Specifies the number of bits to be written to and read from 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.

dwNumBytesToWrite Specifies the number of bytes in the write data buffer, which contains all the specified bits to be written to an external device. Valid range 1 to 65535 ie 64K bytes.

dwTapControllerState Specifies the state, the Test Access Port(TAP) controller will be left in after the data has been written/read to/from an external device.

Valid TAP Controller States

TEST_LOGIC_STATE
RUN_TEST_IDLE_STATE
PAUSE_TEST_DATA_REGISTER_STATE
PAUSE_INSTRUCTION_REGISTER_STATE
SHIFT_TEST_DATA_REGISTER_STATE
SHIFT_INSTRUCTION_REGISTER_STATE

Return Value

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

FTC_TOO_MANY_DEVICES
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
FTC_INSUFFICIENT_RESOURCES
Example:

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

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

FTC_STATUS JTAG_ClearDeviceCmdSequence(FTC_HANDLE ftHandle)

This function clears the sequence of commands and associated data from the internal command buffer associated with a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

Parameters

ftHandle  Handle of a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

Return Value

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

FTC_INVALID_HANDLE
2.1.35 JTAG_AddDeviceWriteCmd

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

This function adds a write command and associated data to the internal command buffer(size 131070 ie 128K bytes) associated with a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device. This enables a programmer to build up a sequence of commands ie write, read and write/read, before executing the sequence of commands, see section 2.1.38.

Warning: While constructing a sequence of commands, do not invoke JTAG_Write, JTAG_Read, JTAG_WriteRead or JTAG_GenerateClockPulses functions, as this will clear the sequence of commands and associated data from the internal command buffer.

Parameters

ftHandle Handle of a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

bInstructionTestData Specifies the type of register, that data is to be written to on an external device. Instruction(TRUE) or test data(FALSE).

dwNumBitsToWrite Specifies the number of 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.

dwNumBytesToWrite Specifies the number of bytes in the write data buffer, which contains all the specified bits to be written to an external device. Valid range 1 to 65535 ie 64K bytes.

dwTapControllerState Specifies the state, the Test Access Port(TAP) controller will be left in after the data has been written to an external device.

Valid TAP Controller States

<table>
<thead>
<tr>
<th>Test Logic State</th>
<th>Run Test Idle State</th>
<th>Pause Test Data Register State</th>
<th>Pause Instruction Register State</th>
<th>Shift Test Data Register State</th>
<th>Shift Instruction Register State</th>
</tr>
</thead>
</table>

Return Value

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

```c
#define MAX_WRITE_DATABYTES_BUFFER_SIZE   65536  // 64K bytes

typedef BYTE WriteDataByteBuffer[MAX_WRITE_DATABYTES_BUFFER_SIZE];
typedef WriteDataByteBuffer *PWriteDataByteBuffer;
```
2.1.36  JTAG_AddDeviceReadCmd

FTC_STATUS JTAG_AddDeviceReadCmd (FTC_HANDLE ftHandle, BOOL bInstructionTestData, DWORD dwNumBitsToRead, DWORD dwTapControllerState)

This function adds a read command to the internal command buffer(size 131070 ie 128K bytes) associated with a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device. This enables a programmer to build up a sequence of commands ie write, read and write/read, before executing the sequence of commands, see section 2.1.38.

Warning: While constructing a sequence of commands, do not invoke JTAG_Write, JTAG_Read, JTAG_WriteRead or JTAG_GenerateClockPulses functions, as this will clear the sequence of commands and associated data from the internal command buffer.

Parameters

ftHandle Handle of a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

bInstructionTestData Specifies the type of register, that data is to be read from on an external device. Instruction(TRUE) or test data(FALSE).

dwNumBitsToRead Specifies the number of bits to be read from an external device. Valid range 2 to 524280. 524280 bits is equivalent to 64K bytes.

dwTapControllerState Specifies the state, the Test Access Port(TAP) controller will be left in after the data has been read from an external device.

Valid TAP Controller States

- TEST_LOGIC_STATE
- RUN_TEST_IDLE_STATE
- PAUSE_TEST_DATA_REGISTER_STATE
- PAUSE_INSTRUCTION_REGISTER_STATE
- SHIFT_TEST_DATA_REGISTER_STATE
- SHIFT_INSTRUCTION_REGISTER_STATE

Return Value

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

- FTC_INVALID_HANDLE
- FTC_INVALID_NUMBER_BITS
- FTC_INVALID_TAP_CONTROLLER_STATE
- FTC_COMMAND_SEQUENCE_BUFFER_FULL
- FTC_INSUFFICIENT_RESOURCES
2.1.37  JTAG_AddDeviceWriteReadCmd

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

This function adds a write/read command and associated data to the internal command buffer(size 131070 ie 128K bytes) associated with a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device. This enables a programmer to build up a sequence of commands ie write, read and write/read, before executing the sequence of commands, see section 2.1.38.

Warning: While constructing a sequence of commands, do not invoke JTAG_Write, JTAG_Read, JTAG_WriteRead or JTAG_GenerateClockPulses functions, as this will clear the sequence of commands and associated data from the internal command buffer.

Parameters

ftHandle Handle of a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

bInstructionTestData Specifies the type of register, that data is to be written to and read from on an external device. Instruction(TRUE) or test data(FALSE).

dwNumBitsToWriteRead Specifies the number of bits to be written to and read from 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.

dwNumBytesToWrite Specifies the number of bytes in the write data buffer, which contains all the specified bits to be written to an external device. Valid range 1 to 65535 ie 64K bytes.

dwTapControllerState Specifies the state, the Test Access Port(TAP) controller will be left in after the data has been written/read to/from an external device.

Valid TAP Controller States

<table>
<thead>
<tr>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST_LOGIC_STATE</td>
</tr>
<tr>
<td>RUN_TEST_IDLE_STATE</td>
</tr>
<tr>
<td>PAUSE_TEST_DATA_REGISTER_STATE</td>
</tr>
<tr>
<td>PAUSE_INSTRUCTION_REGISTER_STATE</td>
</tr>
<tr>
<td>SHIFT_TEST_DATA_REGISTER_STATE</td>
</tr>
<tr>
<td>SHIFT_INSTRUCTION_REGISTER_STATE</td>
</tr>
</tbody>
</table>

Return Value

Returns FTC_SUCCESS if successful, otherwise the return value will be one of the following 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
FTC_INSUFFICIENT_RESOURCES
Example:

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

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

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

This function executes a sequence of commands, stored in the internal command buffer i.e. write, read, write/read data to/from an external device i.e. a device attached to a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device. A FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device communicates with an external device by simulating the JTAG synchronous protocol.

Parameters

- **ftHandle**
  Handle of a FT2232D dual device or FT2232H dual hi-speed device or FT4232H quad hi-speed device.

- **pReadCmdSequenceDataBuffer**
  Pointer to buffer that returns the data read from an external device. Size of buffer should be set to 131071.

- **lpdwNumBytesReturned**
  Pointer to a variable of type DWORD which receives the actual number of data bytes read from an 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

Example:

```c
#define MAX_READ_CMDS_DATABYTES_BUFFER_SIZE 131071 // 128K bytes

typedef BYTE ReadCmdSequenceDataByteBuffer[MAX_READ_CMDS_DATABYTES_BUFFER_SIZE];
typedef ReadCmdSequenceDataByteBuffer  *PReadCmdSequenceDataByteBuffer;
```
2.1.39 JTAG_GetDllVersion

FTC_STATUS JTAG_GetDllVersion(LPSTR lpDllVersionBuffer, DWORD dwBufferSize)

This function returns the version of this DLL.

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

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

FTC_NULL_DLL_VERSION_BUFFER_POINTER
FTC_DLL_VERSION_BUFFER_TOO_SMALL

2.1.40 JTAG_GetErrorCodeString


This function returns the error message for the specified error code, to be used for display purposes by an application programmer. The error code passed into this function must have been returned from a function within this DLL.

Parameters

IpLanguage 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 DLL function call.

IpErrorMessageBuffer Pointer to the buffer that receives the error message. The error message represents the description of the status code. The string will be NULL terminated. If an unsupported language code or invalid status code is passed in to this function, the returned error message will reflect this.

dwBufferSize Length of the buffer created for the error message string. Set buffer length to a minimum of 100 characters.
Return Value

Returns FTC_SUCCESS if successful, otherwise the return value will be one of the following 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
/**
 * Copyright (c) 2005 Future Technology Devices International Ltd.

 Module Name:
   ftcjtag.h

 Abstract:
   API DLL for FT2232H and FT4232H Hi-Speed Dual Device and FT2232D Dual Device setup to simulate the Joint Test Action Group(JTAG) synchronous serial 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
   07/07/08 kra Added new functions for FT2232H and FT4232H hi-speed devices.
   19/08/08 kra Added new function JTAG_CloseDevice.

--*/

#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;

// Hi-speed device types
enum {
   FT2232H_DEVICE_TYPE = 1,
   FT4232H_DEVICE_TYPE = 2
};

#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_SYNCHRONIZEDEVICE_MPSSE 21 // cannot change, error code
mapped from FT2232c classes
#define FTC_INVALIDDEVICE_NAME_INDEX 22 // cannot change, error code
mapped from FT2232c classes
#define FTC_NULLDEVICE_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_INVALIDDEVICE_NAME 25 // cannot change, error code
mapped from FT2232c classes
#define FTC_INVALIDLOCATION_ID 26 // cannot change, error code
mapped from FT2232c classes
#define FTC_DEVCIN_USE 27 // cannot change, error code
mapped from FT2232c classes
#define FTC_TOOMANY_DEVICES 28 // cannot change, error code
mapped from FT2232c classes
#define FTC_NULL_CHANNEL_BUFFER_POINTER 29 // cannot change, error code
mapped from FT2232h classes
#define FTC_CHANNEL_BUFFER_TOO_SMALL 30 // cannot change, error code
mapped from FT2232h classes
#define FTC_INVALIDCHANNEL 31 // cannot change, error code
mapped from FT2232h classes
#define FTC_INVALIDTIMERVERVALUE 32 // cannot change, error code
mapped from FT2232h classes
#define FTC_INVALID_CLOCK_DIVISOR 33
#define FTC_NULL_INPUT_OUTPUT_BUFFER_POINTER 34
#define FTC_INVALIDNUMBER_BITS 35
#define FTC_NULL_WRITE_DATA_BUFFER_POINTER 36
#define FTC_INVALIDNUMBER_BYTES 37
#define FTC_NUMBER_BYTES_TOO_SMALL 38
#define FTC_INVALID_TAP_CONTROLLER_STATE 39
#define FTC_NULL_READ_DATA_BUFFER_POINTER 40
#define FTC_COMMAND_SEQUENCE_BUFFER_FULL 41
#define FTC_NULL_READ_COMMANDS_DATA_BUFFER_POINTER 42
#define FTC_NO_COMMAND_SEQUENCE 43
#define FTC_INVALIDNUMBER CLOCK PULSES 44
#define FTC_INVALIDNUMBER SINGLE_CLOCK PULSES 45
#define FTC_INVALIDNUMBER_TIMES_EIGHT_CLOCK_PULSES 46
#define FTC_CLOSE_FINALSTATE BUFFER POINTER 47
#define FTC_NULL_DLLVERSION BUFFER_POINTER 48
#define FTC_DLLVERSION_BUFFER_TOO_SMALL 49
#define FTC_NULL_LANGUAGE_CODE BUFFER_POINTER 50
#define FTC_NULL_ERRORMESSAGE BUFFER_POINTER 51
#define FTC_ERROR_MESSAGE BUFFER_TOO_SMALL 52
#define FTC_INVALIDLANGUAGE_CODE 53
#define FTC_INVALIDSTATUS_CODE 54
#ifdef __cplusplus
extern "C" {
#endif

FTCJTAG_API
FTC_STATUS WINAPI JTAG_GetNumDevices(LPDWORD lpdwNumDevices);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_GetNumHiSpeedDevices(LPDWORD lpdwNumHiSpeedDevices);

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

FTCJTAG_API
FTC_STATUS WINAPI JTAG_GetHiSpeedDeviceNameLocIDChannel(DWORD dwDeviceNameIndex, LPSTR lpDeviceNameBuffer, DWORD dwBufferSize, LPDWORD lpdwLocationID, LPSTR lpChannel, DWORD dwChannelBufferSize, LPDWORD lpdwHiSpeedDeviceType);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_Open(FTC_HANDLE *pftHandle);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_OpenEx(LPSTR lpDeviceName, DWORD dwLocationID, FTC_HANDLE *pftHandle);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_OpenHiSpeedDevice(LPSTR lpDeviceName, DWORD dwLocationID, LPSTR lpChannel, FTC_HANDLE *pftHandle);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_GetHiSpeedDeviceType(FTC_HANDLE ftHandle, LPDWORD lpdwHiSpeedDeviceType);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_Close(FTC_HANDLE ftHandle);

typedef struct Ft_Close_Final_State_Pins{
    BOOL bTCKPinState;
    BOOL bTCKPinActiveState;
    BOOL bTDIPinState;
    BOOL bTDIPinActiveState;
    BOOL bTMSPinState;
    BOOL bTMSPinActiveState;
}FTC_CLOSE_FINAL_STATE_PINS, *PFTC_CLOSE_FINAL_STATE_PINS;

FTCJTAG_API
FTC_STATUS WINAPI JTAG_CloseDevice(FTC_HANDLE ftHandle, FTC_CLOSE_FINAL_STATE_PINS pCloseFinalStatePinsData);

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

FTCJTAG_API
FTC_STATUS WINAPI JTAG_TurnOnDivideByFiveClockingHiSpeedDevice(FTC_HANDLE ftHandle);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_TurnOffDivideByFiveClockingHiSpeedDevice(FTC_HANDLE ftHandle);

FTCJTAG_API
FTCJTAG_API
FTC_STATUS WINAPI JTAG_TurnOnAdaptiveClockingHiSpeedDevice(FTC_HANDLE ftHandle);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_TurnOffAdaptiveClockingHiSpeedDevice(FTC_HANDLE ftHandle);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_SetDeviceLatencyTimer(FTC_HANDLE ftHandle, BYTE timerValue);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_GetDeviceLatencyTimer(FTC_HANDLE ftHandle, LPBYTE lpTimerValue);

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

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

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

typedef struct Ft_Input_Output_Pins{
    BOOL  bPin1InputOutputState;
    BOOL  bPin1LowHighState;
    BOOL  bPin2InputOutputState;
    BOOL  bPin2LowHighState;
    BOOL  bPin3InputOutputState;
    BOOL  bPin3LowHighState;
    BOOL  bPin4InputOutputState;
    BOOL  bPin4LowHighState;
    BOOL  bPin5InputOutputState;
    BOOL  bPin5LowHighState;
    BOOL  bPin6InputOutputState;
    BOOL  bPin6LowHighState;
    BOOL  bPin7InputOutputState;
    BOOL  bPin7LowHighState;
}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 FTH_Input_Output_Pins{
    BOOL  bPin1InputOutputState;
    BOOL  bPin1LowHighState;
    BOOL  bPin2InputOutputState;
    BOOL  bPin2LowHighState;
    BOOL  bPin3InputOutputState;
    BOOL  bPin3LowHighState;
    BOOL  bPin4InputOutputState;
    BOOL  bPin4LowHighState;
    BOOL  bPin5InputOutputState;
    BOOL  bPin5LowHighState;
    BOOL  bPin6InputOutputState;
    BOOL  bPin6LowHighState;
    BOOL  bPin7InputOutputState;
    BOOL  bPin7LowHighState;
}FTC_INPUT_OUTPUT_PINS, *PFTC_INPUT_OUTPUT_PINS;
BOOL bPin8InputOutputState;
BOOL bPin8LowHighState;
}FTH_INPUT_OUTPUT_PINS, *PFTH_INPUT_OUTPUT_PINS;

FTCJTAG_API
FTC_STATUS WINAPI JTAG_SetHiSpeedDeviceGPIOs(FTC_HANDLE ftHandle, BOOL
bControlLowInputOutputPins,
   PFTC_INPUT_OUTPUT_PINS pLowInputOutputPinsData,
   BOOL bControlHighInputOutputPins,
   PFTH_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);

typedef struct FTH_Low_High_Pins{
   BOOL bPin1LowHighState;
   BOOL bPin2LowHighState;
   BOOL bPin3LowHighState;
   BOOL bPin4LowHighState;
   BOOL bPin5LowHighState;
   BOOL bPin6LowHighState;
   BOOL bPin7LowHighState;
   BOOL bPin8LowHighState;
}FTH_LOW_HIGH_PINS, *PFTH_LOW_HIGH_PINS;

FTCJTAG_API
FTC_STATUS WINAPI JTAG_GetHiSpeedDeviceGPIOs(
   FTC_HANDLE ftHandle, BOOL
   bControlLowInputOutputPins,
   PFTC_LOW_HIGH_PINS pLowPinsInputData,
   BOOL bControlHighInputOutputPins,
   PFTH_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 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_GenerateClockPulsesHiSpeedDevice(FTC_HANDLE ftHandle,
BOOL bPulseClockTimesEightFactor, DWORD dwNumClockPulses, BOOL bControlLowInputOutputPin,
BOOL bStopClockPulsesState);

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_CMDS_DATA_BYTES_BUFFER_SIZE 131071 // 128K bytes

typedef BYTE ReadCmdSequenceDataByteBuffer[MAX_READ_CMDS_DATA_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 bInstructionTestData,
DWORD dwNumBitsToWrite, PWriteDataByteBuffer pWriteDataBuffer,
DWORD dwNumBytesToWrite, DWORD dwTapControllerState);
FTCJTAG_API
FTC_STATUS WINAPI JTAG_AddDeviceReadCmd(FTC_HANDLE ftHandle, BOOL bInstructionTestData, DWORD dwNumBitsToRead, DWORD dwTapControllerState);

FTCJTAG_API
FTC_STATUS WINAPI JTAG_AddDeviceWriteReadCmd(FTC_HANDLE ftHandle, BOOL bInstructionTestData, 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_GetDllVersion(LPSTR lpDllVersionBuffer, DWORD dwBufferSize);

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

#ifdef __cplusplus
#endif /* FTCJTAG_H */
4 JTAG TAP Controller State Diagram

All transitions occur based on the state of TMS on the rising edge of TCK.
5 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom

Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales) sales1@ftdichip.com
E-mail (Support) support1@ftdichip.com
E-mail (General Enquiries) admin1@ftdichip.com
Web Site URL http://www.ftdichip.com
Web Shop URL http://www.ftdichip.com

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited (Taiwan)
2F, No 516, Sec. 1 NeiHu Road
Taipei 114
Taiwan, R.O.C.
Tel: +886 (0) 2 8797 1330
Fax: +886 (0) 2 8751 9737

E-mail (Sales) tw.sales1@ftdichip.com
E-mail (Support) tw.support1@ftdichip.com
E-mail (General Enquiries) tw.admin1@ftdichip.com
Web Site URL http://www.ftdichip.com

Branch Office – Hillsboro, Oregon, USA

Future Technology Devices International Limited (USA)
7235 NW Evergreen Parkway, Suite 600
Hillsboro, OR 97123-5803
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales) us.sales@ftdichip.com
E-Mail (Support) us.support@ftdichip.com
E-Mail (General Enquiries) us.admin@ftdichip.com
Web Site URL http://www.ftdichip.com

Branch Office – Shanghai, China

Future Technology Devices International Limited (China)
Room 408, 317 Xianxia Road,
ChangNing District,
ShangHai, China

Tel: +86 (21) 62351596
Fax: +86(21) 62351595

E-Mail (Sales): cn.sales@ftdichip.com
E-Mail (Support): cn.support@ftdichip.com
E-Mail (General Enquiries): cn.admin1@ftdichip.com
Web Site URL: http://www.ftdichip.com
Distributor and Sales Representatives
Please visit the Sales Network page of the FTDI Web site for the contact details of our distributor(s) and sales representative(s) in your country.

Neither the whole nor any part of the information contained in, or the product described in this manual, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. This product and its documentation are supplied on an as-is basis and no warranty as to their suitability for any particular purpose is either made or implied. Future Technology Devices International Ltd will not accept any claim for damages howsoever arising as a result of use or failure of this product. Your statutory rights are not affected. This product or any variant of it is not intended for use in any medical appliance, device or system in which the failure of the product might reasonably be expected to result in personal injury. This document provides preliminary information that may be subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Future Technology Devices International Ltd, Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow G41 1HH United Kingdom. Scotland Registered Number: SC136640
# Appendix A – Revision History

<table>
<thead>
<tr>
<th>Revision History</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draft</td>
<td>Initial Draft</td>
</tr>
<tr>
<td>1.0</td>
<td>Initial Release</td>
</tr>
<tr>
<td>1.1</td>
<td>Added missing commands</td>
</tr>
<tr>
<td>1.2</td>
<td>Updated JTAG state machine</td>
</tr>
<tr>
<td></td>
<td>Updated US Office support email</td>
</tr>
</tbody>
</table>