The application note is a guide for LibFT4222 based on D2XX. It provides high-level and convenient APIs for FT4222H application development.

Use of FTDI devices in life support and/or safety applications is entirely at the user’s risk, and the user agrees to defend, indemnify and hold FTDI harmless from any and all damages, claims, suits or expense resulting from such use.
# Table of Contents

1 Introduction .................................................................................................................. 4  
   1.1 Overview .................................................................................................................. 5  
   1.2 Scope ....................................................................................................................... 7  

2 Getting Started ............................................................................................................ 8  

3 Application Programming Interface (API) ................................................................... 11  
   3.1 Typedefs .................................................................................................................. 11  
   3.2 FT4222 General Functions ...................................................................................... 11  
      3.2.1 Open and Close ................................................................................................. 11  
      3.2.2 Un-initialize ...................................................................................................... 12  
      3.2.3 Set Clock .......................................................................................................... 12  
      3.2.4 Get Clock .......................................................................................................... 12  
      3.2.5 Set Suspend Out ............................................................................................... 13  
      3.2.6 Set Wake Up/Interrupt ..................................................................................... 13  
      3.2.7 Set Interrupt Trigger Condition ....................................................................... 14  
      3.2.8 Get Max Transfer Size ..................................................................................... 14  
      3.2.9 Set Event Notification ..................................................................................... 15  
      3.2.10 Get Version ...................................................................................................... 15  
   3.3 SPI Master Functions ............................................................................................... 17  
      3.3.1 SPI Master Init ................................................................................................ 17  
      3.3.2 SPI Master Set Lines ....................................................................................... 18  
      3.3.3 SPI Master Set Driving Strength .................................................................... 19  
      3.3.4 SPI Master Single Read .................................................................................. 19  
      3.3.5 SPI Master Single Write .................................................................................. 20  
      3.3.6 SPI Master Single Read and Write ................................................................. 20  
      3.3.7 SPI Master Multi Read and Write ................................................................... 21  
   3.4 SPI Slave Functions ................................................................................................. 24  
      3.4.1 SPI Slave Init ................................................................................................... 26  
      3.4.2 SPI Slave Get Rx Status ............................................................................... 27  
      3.4.3 SPI Slave Read ................................................................................................. 27
3.4.4 SPI Slave Write ............................................................ 27
3.5 SPI General Functions .................................................. 29
  3.5.1 SPI Reset Transaction ............................................ 29
  3.5.2 SPI Reset .............................................................. 29
3.6 I2C Master Functions .................................................. 30
  3.6.1 I2C Master Init ..................................................... 30
  3.6.2 I2C Master Read .................................................. 30
  3.6.3 I2C Master Write ................................................ 31
  3.6.4 I2C Master Reset ................................................ 31
3.7 I2C Slave Functions .................................................. 33
  3.7.1 I2C Slave Init ..................................................... 33
  3.7.2 I2C Slave Get Address ........................................ 33
  3.7.3 I2C Slave Set Address ........................................ 34
  3.7.4 I2C Slave Get Rx Status ....................................... 34
  3.7.5 I2C Slave Read .................................................. 34
  3.7.6 I2C Slave Write ................................................ 35
  3.7.7 I2C Slave Reset ................................................ 35
3.8 GPIO Functions .......................................................... 36
  3.8.1 GPIO Init .......................................................... 36
  3.8.2 GPIO Read ........................................................ 36
  3.8.3 GPIO Write ........................................................ 37
  3.8.4 GPIO Set Input Trigger ....................................... 38
  3.8.5 GPIO Get Trigger Status ..................................... 38
  3.8.6 GPIO Read Trigger Queue .................................... 39
4 Contact Information ..................................................... 41
Appendix A – Enumeration and Structure Definitions .................. 42
Appendix B – References ................................................. 45
  Document References .................................................. 45
  Acronyms and Abbreviations ........................................ 45
Appendix C – List of Tables & Figures .................................... 46
  List of Tables .......................................................... 46
List of Figures ........................................................................................................................................46
Appendix D – Revision History .............................................................................................................47
1 Introduction

The FT4222H is a USB interface device which supports SPI and I\(^2\)C communication protocol. It is accompanied with the support library “LibFT4222” based on D2XX. It provides high-level APIs to facilitate user application development.

The FT4222H contains SPI/I\(^2\)C configurable interfaces. The SPI interface can be configured as master mode with single, dual, quad bits wide data transfer or slave mode with single bit wide data transfer. The I\(^2\)C interface can be configured as master or slave mode.

![Software Stack Diagram](image)

**Figure 1.1 The software stack**

The LibFT4222 sample code, release notes, and all necessary files can be downloaded from the FTDI website at:

http://www.ftdichip.com/Products/ICs/FT4222H.html

The sample source code contained in this application note is provided as an example and is neither guaranteed nor supported by FTDI.
1.1 Overview

The FT4222H supports 4 modes to allow various I2C/SPI devices to be connected to USB bus. The attachable device configuration for each mode is listed below:

- **Mode 0 (2 USB interfaces):**
  - 1 SPI master, SPI slave, I^2^C master, or I^2^C slave device
  - 1 GPIO device
- **Mode 1 (4 USB interfaces):**
  - SPI master connects up to 3 SPI slave devices
  - 1 GPIO device
- **Mode 2 (4 USB interfaces):**
  - SPI master connects up to 4 SPI slave devices
- **Mode 3 (1 USB interface):**
  - 1 SPI master, SPI slave, I^2^C master, or I^2^C slave device

In mode 0 and 3, the connecting device can be SPI/I^2^C master or slave, depending on how an application developer initializes the FT4222H chip. Mode 1 and mode 2 are designed to connect to multiple SPI slave devices.

The FT4222H can be configured with up to 4 GPIO pins for user applications, but each pin is multiplexed with interrupt/suspend out/SPI slave select/I^2^C functions as listed below:

- gpio0 / ss1o / scl
- gpio1 / ss2o / sda
- gpio2 / ss3o / suspend out
- gpio3 / wakeup/intr

If the FT4222H is initialized as an I^2^C connecting device, as pins shown above, the pins of gpio0 and gpio1 will be switched to scl and sda, and cannot be used as GPIO.

By default the pin for pio2 is configured as suspend out, and the pin for pio3 is configured as wakeup/intr. Only those configured GPIO pins can support GPIO read/set operation through the corresponding endpoint.

The following diagrams show the examples of FT4222H SPI master connections.
Figure 1.2 Mode 0: FT4222H works as SPI master (quad mode)

Figure 1.3 Mode 0: FT4222H works as I²C master
1.2 Scope

The guide is intended for developers who are creating applications, extending FTDI provided applications or implementing FTDI’s applications for the FT4222H.

Figure 1.4 Mode 2: FT4222H works as SPI master
2 Getting Started

A LibFT4222 application usually starts with FT_CreateDeviceInfoList and FT_GetDeviceInfoList as a traditional D2XX application does. Under different chip modes, FT_CreateDeviceInfoList reports a different number of interfaces as shown in the table below.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Num of Interfaces</th>
<th>Device Function</th>
</tr>
</thead>
</table>
| 0    | 2                 | a. The first interface: SPI master, SPI slave, I²C master, or I²C slave device.  
                  |                   | b. The second interface: GPIO device. |
| 1    | 4                 | a. The first 3 interfaces: SPI master connects up to 3 SPI slaves.  
                  |                   | b. The 4th interface: GPIO device. |
| 2    | 4                 | a. SPI master connects up to 4 SPI slaves. Please refer figure 1.4.  
                  |                   | FT4222H works as SPI master. |
| 3    | 1                 | a. SPI master, SPI slave, I²C master, or I²C slave device. |

Table 2.1 Chip mode and device functions

After opening the device with FT_Open, developers need to initialize the FT4222H device as either SPI master, SPI slave, I²C master, or I²C slave. Different types of device require different configurations. For more details, please refer the next chapter.

Example#

```c
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <string>
#include "ftd2xx.h"
#include "LibFT4222.h"

std::vector< FT_DEVICE_LIST_INFO_NODE > g_FT4222DevList;

inline std::string DeviceFlagToString(DWORD flags)
{
    std::string msg;
    msg += (flags & 0x1)? "DEVICE_OPEN" : "DEVICE_CLOSED";
```
msg += ",",
msg += (flags & 0x2)? "High-speed USB" : "Full-speed USB";
return msg;
}

void ListFtUsbDevices()
{
  DWORD numOfDevices = 0;
  FT_STATUS status = FT_CreateDeviceInfoList(&numOfDevices);

  for(DWORD iDev=0; iDev<numOfDevices; ++iDev)
  {
    FT_DEVICE_LIST_INFO_NODE devInfo;
    memset(&devInfo, 0, sizeof(devInfo));

    status = FT_GetDeviceInfoDetail(iDev,
        &devInfo.Flags, &devInfo.Type, &devInfo.ID, &devInfo.LocId,
        devInfo.SerialNumber, devInfo.Description, &devInfo.ftHandle);

    if (FT_OK == status)
    {
      printf("Dev %d:\n", iDev);
      printf(" Flags= 0x%x, (%s)\n", devInfo.Flags,
          DeviceFlagToString(devInfo.Flags).c_str());
      printf(" Type= 0x%x\n", devInfo.Type);
      printf(" ID= 0x%x\n", devInfo.ID);
      printf(" LocId= 0x%x\n", devInfo.LocId);
      printf(" SerialNumber= %s\n", devInfo.SerialNumber);
      printf(" Description= %s\n", devInfo.Description);
      printf(" ftHandle= 0x%x\n", devInfo.ftHandle);

      const std::string desc = devInfo.Description;
      if(desc == "FT4222" || desc == "FT4222 A")
      {
        g_FT4222DevList.push_back(devInfo);
      }
    }
  }
}
int main(int argc, char const *argv[]) {
    ListFtUsbDevices();

    if(g_FT4222DevList.empty()) {
        printf("No FT4222 device is found!\n");
        return 0;
    }

    FT_HANDLE ftHandle = NULL;

    FT_STATUS ftStatus;
    ftStatus = FT_OpenEx((VOID)*g_FT4222DevList[0].LocId,
                         FT_OPEN_BY_LOCATION, &ftHandle);
    if (FT_OK != ftStatus) {
        printf("Open a FT4222 device failed!\n");
        return 0;
    }

    ftStatus = FT4222_SPIMaster_Init(ftHandle,
                                       SPI_IO_SINGLE,
                                       CLK_DIV_4,
                                       CLK_ACTIVE_LOW,
                                       CLK_LEADING,
                                       0x01);
    if (FT_OK != ftStatus) {
        printf("Init FT4222 as SPI master device failed!\n");
        return 0;
    }

    // TODO:
    // Start to work as SPI master, and read/write data to a
    //    SPI slave
    // FT4222_SPIMaster_SingleWrite
    // FT4222_SPIMaster_SingleRead
    // FT4222_SPIMaster_SingleReadWrite

    FT4222_Uninitialize(ftHandle);
    FT_Close(ftHandle);
    return 0;
}
3 Application Programming Interface (API)

LibFT4222 supports SPI, I²C and GPIO communication using high-level APIs. In addition, it provides chip configuration APIs, such as FT4222_SetClock.

After calling FT_Open, the FT4222H is required to be initialized by one of the following initial functions:

- FT4222_SPIMaster_Init
- FT4222_SPISlave_Init
- FT4222_I2CMaster_Init
- FT4222_I2CSlave_Init
- FT4222_GPIO_Init

The initialization functions help developers to switch the FT4222H into a specific mode.

At the end of the application, FT4222_Uninitialize should be called to release allocated resources, before calling FT_Close.

All the APIs return an FT4222_STATUS, which extends FT_STATUS that is defined in the D2XX driver. FT4222_STATUS defines additional values to report FT4222H specific status.

3.1 Typedefs

The following typedefs have been defined for keeping cross platform portability:

- typedef unsigned long DWORD
- typedef unsigned char uint8
- typedef unsigned short uint16
- typedef unsigned long uint32
- typedef signed char int8
- typedef signed short int16
- typedef signed long int32
- typedef unsigned char bool

Please refer to Appendix A for more enumeration and structure definitions.

3.2 FT4222 General Functions

The functions listed in this section are system-wise configuration functions.

3.2.1 Open and Close

An application of LibFT4222 should open the device and get a handle for subsequent accesses by calling FT_Open or FT_OpenEx. Both are D2XX API. Please refer D2XX for more details. In addition, please note that the FT4222H assigns different functions to different interfaces. For example, under mode 0, interface A is assigned as SPI or I²C interface, and interface B is assigned as GPIO interface.
After finishing using the device, FT_Close should be called to release the device.

### 3.2.2 Un-initialize

FT4222_STATUS **FT4222_UnInitialize**(FT_HANDLE ftHandle)

**Summary:**
Release allocated resources. FT4222_Uninitialize should be called before calling FT_Close.

**Parameters:**

| ftHandle | Handle of the device. |

**Return Value:**
FT4222_OK if successful, otherwise the return value is an FT error code.

### 3.2.3 Set Clock

FT4222_STATUS **FT4222_SetClock**(FT_HANDLE ftHandle, FT4222_ClockRate clk)

**Summary:**
Set the system clock rate. The FT4222H supports 4 clock rates: 80MHz, 60MHz, 48MHz, or 24MHz. By default, the FT4222H runs at 60MHz clock rate.

**Parameters:**

<table>
<thead>
<tr>
<th>ftHandle</th>
<th>Handle of the device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>FT4222 system clock rate:</td>
</tr>
<tr>
<td></td>
<td>• SYS_CLK_60</td>
</tr>
<tr>
<td></td>
<td>• SYS_CLK_24</td>
</tr>
<tr>
<td></td>
<td>• SYS_CLK_48</td>
</tr>
<tr>
<td></td>
<td>• SYS_CLK_80</td>
</tr>
</tbody>
</table>

**Return Value:**
FT4222_OK if successful, otherwise the return value is an FT error code.

### 3.2.4 Get Clock

FT4222_STATUS **FT4222_GetClock**(FT_HANDLE ftHandle, FT4222_ClockRate* pClk)

**Summary:**

Get the current system clock rate.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>pClk</td>
<td>Pointer to a variable of type FT4222_ClockRate where the value will be stored.</td>
</tr>
</tbody>
</table>

**Return Value:**

FT4222_OK if successful, otherwise the return value is an FT error code.

### 3.2.5 Set Suspend Out

FT4222_STATUS `FT4222_SetSuspendOut(FT_HANDLE ftHandle, BOOL enable)`

**Summary:**

Enable or disable suspend out which will emit a signal when FT4222H enters suspend mode. Please note that the suspend out pin is not available under mode 2.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>enable</td>
<td>TRUE to enable suspend out and configure GPIO2 as an output pin for emitting a signal when suspended. FALSE to switch back to GPIO2.</td>
</tr>
</tbody>
</table>

**Return Value:**

FT4222_OK if successful, otherwise the return value is an FT error code.

### 3.2.6 Set Wake Up/Interrupt

FT4222_STATUS `FT4222_SetWakeUpInterrupt(FT_HANDLE ftHandle, BOOL enable)`

**Summary:**

Enable or disable wakeup/interrupt.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>enable</td>
<td>TRUE to configure GPIO3 as an input pin for wakeup/interrupt. FALSE to switch back to GPIO3.</td>
</tr>
</tbody>
</table>
Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.

3.2.7 Set Interrupt Trigger Condition

FT4222_STATUS FT4222_SetInterruptTrigger(FT_HANDLE ftHandle, GPIO_Tigger trigger)

Summary:
Set trigger condition for the pin wakeup/interrupt

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>trigger</td>
<td>Trigger condition. One of the following:</td>
</tr>
<tr>
<td></td>
<td>• GPIO_TRIGGER_RISING</td>
</tr>
<tr>
<td></td>
<td>• GPIO_TRIGGER_FALLING</td>
</tr>
<tr>
<td></td>
<td>• GPIO_TRIGGER_LEVEL_HIGH</td>
</tr>
<tr>
<td></td>
<td>• GPIO_TRIGGER_LEVEL_LOW</td>
</tr>
</tbody>
</table>

Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.

3.2.8 Get Max Transfer Size

FT4222_STATUS FT4222_GetMaxTransferSize(FT_HANDLE ftHandle, uint16* pMaxSize)

Summary:
This function returns the maximum packet size in a transaction. It will be affected by different bus speeds, chip modes, and functions.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>pMaxSize</td>
<td>Pointer to a variable of type unit16 where the returning value will be stored.</td>
</tr>
</tbody>
</table>

Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.
3.2.9 Set Event Notification

FT4222_STATUS FT4222_SetEventNotification(FT_HANDLE ftHandle, DWORD dwEventMask, PVOID pvArg)

**Summary:**
Sets conditions for event notification.

An application can use this function to setup conditions which allow a thread to block until one of the conditions is met. Typically, an application will create an event, call this function, and then block on the event. When the conditions are met, the event is set, and the application thread unblocked. Usually, the event is set to notify the application to check the condition. The application needs to check the condition again before it goes to handle the condition.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>dwEventMask</td>
<td>Conditions that cause the event to be set. It is a bit-map that describes the events the application is interested in. Currently, this function only supports the event below:</td>
</tr>
<tr>
<td></td>
<td>• FT4222_EVENT_RXCHAR</td>
</tr>
<tr>
<td></td>
<td>The event will be set when a data packet has been received by the device.</td>
</tr>
<tr>
<td>pvArg</td>
<td>Interpreted as the handle of an event which has been created by the application. If one of the event conditions is met, the event is set.</td>
</tr>
</tbody>
</table>

**Return Value:**

FT4222_OK if successful, otherwise the return value is an FT error code.

3.2.10 Get Version

FT4222_STATUS FT4222_GetVersion(FT_HANDLE ftHandle, FT4222_Version* pVersion)

**Summary:**

Get the versions of FT4222H and LibFT4222.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>pVersion</td>
<td>Pointer to a variable of type FT4222_Version where the value will be stored. Type FT4222_Version is defined as following:</td>
</tr>
</tbody>
</table>
struct FT4222_Version
{
    DWORD chipVersion; // The version of FT4222H chip
    DWORD dllVersion; // The version of LibFT4222
};

Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.
3.3 SPI Master Functions

The FT4222H can be initialized as an SPI master under all modes. As SPI master, it allows data transfers in three types of bit width:

- **Single SPI transfer** – Standard data transfer format – data is read and written simultaneously
- **DUAL SPI Transfer/Receive** - Data is transferred out or received in on 2 SPI lines simultaneously
- **QUAD SPI Transfer/Receive** – Data is transferred out or received in on 4 SPI lines simultaneously

Please refer DS_FT4222H for more details.

3.3.1 SPI Master Init

FT4222_STATUS FT4222_SPMaster_Init(FT_HANDLE ftHandle, FT4222_SPIMode ioLine, FT4222_SPIClock clock, FT4222_SPICPOL cpol, FT4222_SPICPHA cpha, uint8 ssoMap)

**Summary:**

Initialize FT4222H as an SPI master.

In order to support various types of SPI slave devices, FT4222H SPI master is configurable using the following parameters:

- **IO lines**: SPI transmission lines. FT4222H SPI supports single, dual, or quad transmission mode.
- **Clock divider**: SPI clock rate is subject to system clock. FT4222H SPI clock cloud be 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128, 1/256, or 1/512 system clock rate.
- **Clock polarity**: Active high or active low.
- **Clock phase**: Data is sampled on the leading (first) or trailing (second) clock edge.
- **Slave selection output pins**: Select slave devices by ss0o, ss1o, ss2o, ss3o.

**Parameters:**

| ftHandle | Handle of the device. |
| ioLine  | SPI transmission lines: |
|         | - SPI_IO_SINGLE |
|         | - SPI_IO_DUAL |
|         | - SPI_IO_QUAD |
| clock   | Clock divider: |
|         | - CLK_DIV_2  (1/2  System Clock) |
|         | - CLK_DIV_4  (1/4  System Clock) |
|         | - CLK_DIV_8  (1/8  System Clock) |
## 3.3.2 SPI Master Set Lines

**FT4222_STATUS** `FT4222_SPIMaster_SetLines`(`FT_HANDLE ftHandle, FT4222_SPIMode spiMode)`

### Summary:
Switch FT4222H SPI master to single, dual, or quad lines.

### Parameters:

<table>
<thead>
<tr>
<th>ftHandle</th>
<th>Handle of the device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>spiMode</td>
<td>SPI mode could be:</td>
</tr>
<tr>
<td></td>
<td>• SPI_IO_SINGLE</td>
</tr>
<tr>
<td></td>
<td>• SPI_IO_DUAL</td>
</tr>
<tr>
<td></td>
<td>• SPI_IO_QUAD</td>
</tr>
</tbody>
</table>

### Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.
### 3.3.3 SPI Master Set Driving Strength

**FT4222_STATUS** `FT4222_SPIMaster_SetDrivingStrength`(FT_HANDLE ftHandle,
SPI_DrivingStrength clkStrength,
SPI_DrivingStrength ioStrength,
SPI_DrivingStrength ssoStrength)

**Summary:**
For the FT4222H SPI master, set the driving strength of clk, io, and sso pins.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>clkStrength</td>
<td>The driving strength of the clk pin:</td>
</tr>
<tr>
<td></td>
<td>- DS_4MA</td>
</tr>
<tr>
<td></td>
<td>- DS_8MA</td>
</tr>
<tr>
<td></td>
<td>- DS_12MA</td>
</tr>
<tr>
<td></td>
<td>- DS_16MA</td>
</tr>
<tr>
<td>ioStrength</td>
<td>The driving strength of the io pin:</td>
</tr>
<tr>
<td></td>
<td>- DS_4MA</td>
</tr>
<tr>
<td></td>
<td>- DS_8MA</td>
</tr>
<tr>
<td></td>
<td>- DS_12MA</td>
</tr>
<tr>
<td></td>
<td>- DS_16MA</td>
</tr>
<tr>
<td>ssoStregth</td>
<td>The driving strength of the sso pin:</td>
</tr>
<tr>
<td></td>
<td>- DS_4MA</td>
</tr>
<tr>
<td></td>
<td>- DS_8MA</td>
</tr>
<tr>
<td></td>
<td>- DS_12MA</td>
</tr>
<tr>
<td></td>
<td>- DS_16MA</td>
</tr>
</tbody>
</table>

**Return Value:**
FT4222_OK if successful, otherwise the return value is an FT error code.

### 3.3.4 SPI Master Single Read

**FT4222_STATUS** `FT4222_SPIMaster_SingleRead`(FT_HANDLE ftHandle, uint8* buffer,
uint16 bytesToRead, uint16* sizeOfRead, BOOL isEndTransaction)

**Summary:**
Under SPI single mode, read data from an SPI slave.
3.3.5 SPI Master Single Write

**FT4222_STATUS FT4222_SPI_Master_SingleWrite** (FT_HANDLE ftHandle, uint8* buffer, uint16 bytesToWrite, uint16* sizeTransferred, BOOL isEndTransaction)

**Summary:**
Under SPI single mode, write data to a SPI slave.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>buffer</td>
<td>Pointer to the buffer that contains the data to be written to the device.</td>
</tr>
<tr>
<td>bytesToWrite</td>
<td>Number of bytes to write to the device.</td>
</tr>
<tr>
<td>sizeTransferred</td>
<td>Pointer to a variable of type uint16 which receives the number of bytes written to the device.</td>
</tr>
<tr>
<td>isEndTransaction</td>
<td>If TRUE the Slave Select pin will be raised at the end of the write.</td>
</tr>
</tbody>
</table>

**Return Value:**
FT4222_OK if successful, otherwise the return value is an FT error code.

3.3.6 SPI Master Single Read and Write

**FT4222_STATUS FT4222_SPI_Master_SingleReadWrite** (FT_HANDLE ftHandle, uint8* readBuffer, uint8* writeBuffer, uint16 sizeToTransfer, uint16* sizeTransferred, BOOL isEndTransaction)

**Summary:**
Under SPI single mode, read and write data to a SPI slave.
Under SPI single mode, full-duplex write data to and read data from an SPI slave. The standard SPI protocol simultaneously sends data onto the MOSI data line and receives data from the MISO line as shown below.

```
ss ________________________________
sck ________________________________
mosi W7 W6 W5 W4 W3 W2 W1 W0
miso R7 R6 R5 R4 R3 R2 R1 R0
```

**Figure 3.1 SPI full duplex communication**

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>readBuffer</td>
<td>Pointer to the buffer that receives the data from the device.</td>
</tr>
<tr>
<td>writeBuffer</td>
<td>Pointer to the buffer that contains the data to be written to the device.</td>
</tr>
<tr>
<td>sizeToTransfer</td>
<td>The size of read and write buffer. They must be the same.</td>
</tr>
<tr>
<td>sizeTransferred</td>
<td>Pointer to a variable of type uint16 which receives the number of bytes read and written to the device.</td>
</tr>
<tr>
<td>isEndTransaction</td>
<td>TRUE to raise the pin of SS at the end of the transaction.</td>
</tr>
</tbody>
</table>

**Return Value:**

FT4222_OK if successful, otherwise the return value is an FT error code.

### 3.3.7 SPI Master Multi Read and Write

FT4222_STATUS **FT4222_SPIMaster_MultiReadWrite**(FT_HANDLE ftHandle, uint8* readBuffer, uint8* writeBuffer, uint8 singleWriteBytes,uint16 multiWriteBytes,uint16 multiReadBytes, uint32* sizeOfRead)

**Summary:**

Under SPI dual or quad mode, write data to and read data from an SPI slave. The figure below illustrates the dual-SPI protocol supported by the FT4222H SPI master. It is a mixed protocol initiated with single write transmission, which may be an SPI...
command and dummy cycles, and followed by dual-write and dual-read transmission that use 2 signals in parallel for the data. All three parts of the protocol are optional. For example, developers can ignore the multi-read part by setting multiReadBytes=0.

![Dual SPI communication diagram](image)

**Figure 3.2 Dual SPI communication**

The figure below illustrates the quad-SPI protocol supported by the FT4222H SPI master. It is the same as the dual-protocol illustrated above - it is a mixed protocol initiated with single write transmission and followed by quad-write and quad-read transmission that use 4 signals in parallel for the data.

![Quad SPI communication diagram](image)

**Figure 3.3 Quad SPI communication**

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>readBuffer</td>
<td>Pointer to the buffer that receives the data from the device.</td>
</tr>
<tr>
<td>writeBuffer</td>
<td>Pointer to the buffer that contains the data to be written to the device.</td>
</tr>
</tbody>
</table>

The data is comprised of both single-write and multi-write parts. It starts with the single-write data, whose length is specified...
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>singleWriteBytes</td>
<td>Number of bytes in writeBuffer will be written as single-line.</td>
</tr>
<tr>
<td>multiWriteBytes</td>
<td>Number of bytes in writeBuffer will be written as multi-line.</td>
</tr>
<tr>
<td>multiReadBytes</td>
<td>Number of bytes to read as multi-line.</td>
</tr>
<tr>
<td>sizeOfRead</td>
<td>Pointer to a variable of type uint16 which receives the number of bytes read from the device.</td>
</tr>
</tbody>
</table>

**Return Value:**

FT4222_OK if successful, otherwise the return value is an FT error code.
### 3.4 SPI Slave Functions

The FT4222H can be initialized as an SPI slave under mode 0 and mode 3. As SPI slave, FT4222H only supports the standard single SPI transfer. Please refer to [DS_FT4222H](#) for more details.

A USB-SPI bridge usually faces the challenge that USB cannot guarantee the throughput for each endpoint, but SPI requires data transmission at a steady rate. It is highly possible when an SPI master starts to request data from a USB-SPI slave bridge device, the data has not arrived from the USB host side yet. In addition, SPI does not have a standard protocol to allow the master side to check the status of the slave side. The protocol is usually provided by an SPI slave device on its own, which makes the SPI master device communicate with the slave device by its specified commands.

The FT4222H and LibFT4222 design and implement an SPI slave protocol to handle the integrity of data transmission.

In this protocol, a master starts an SPI transaction by sending a packet in the format illustrated below:

![SPI Slave Protocol Format](#)

It starts with **Sync word**: 0x5A, and followed by **Command** field:

<table>
<thead>
<tr>
<th>Command</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Transfer</td>
<td>0x80</td>
</tr>
<tr>
<td>Slave Transfer</td>
<td>0x81</td>
</tr>
<tr>
<td>Short master transfer (without checksum)</td>
<td>0x82</td>
</tr>
<tr>
<td>Short slave transfer (without checksum)</td>
<td>0x83</td>
</tr>
<tr>
<td>ACK</td>
<td>0x84</td>
</tr>
</tbody>
</table>
SN stands for serial number. It is monotonically increased, and helps to identify packets. Size is a two-byte field, which is the size of the data field in big-endian order. The Checksum is the summation of all data fields’ lower two bytes starting from the first byte, the sync word, to the latest data byte.

The checksum is in big-endian order as well. When the slave, FT4222H, receives the transfer request from the master, it will respond with an ACK. The master can confirm the transaction succeeded when it receives the ACK from the slave.

Here is an example of an ACK packet. The SN field of the ACK packet identifies which request it corresponds to. An ACK packet has no data therefore the Size field should be 0.

If the SPI master does not receive the ACK response from the slave, it should send its request again.
When the FT4222H SPI slave wants to send data to the master, which may be requested by the master, it just sends a transfer request in the same protocol format as shown in figure 3.4.

**3.4.1 SPI Slave Init**

```c
FT4222_STATUS FT4222_SPI_SLAVE_Init(FT_HANDLE ftHandle)
```

**Summary:**

Initialize FT4222H as SPI slave.

**Parameters:**

| ftHandle       | Handle of the device. |

**Return Value:**

FT4222_OK if successful, otherwise the return value is an FT error code.
3.4.2 SPI Slave Get Rx Status

**FT4222_STATUS FT4222_SPISlave_GetRxStatus**(FT_HANDLE ftHandle, uint16* pRxSize)

**Summary:**
Get number of bytes in the receive queue.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>pRxSize</td>
<td>Pointer to a variable of type uint16 which receives the number of bytes in</td>
</tr>
<tr>
<td></td>
<td>the receive queue.</td>
</tr>
</tbody>
</table>

**Return Value:**
FT4222_OK if successful, otherwise the return value is an FT error code.

3.4.3 SPI Slave Read

**FT4222_STATUS FT4222_SPISlave_Read**(FT_HANDLE ftHandle, uint8* buffer, uint16 bytesToRead, uint16* sizeOfRead)

**Summary:**
Read data from receive queue of the SPI slave device.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>buffer</td>
<td>Pointer to the buffer that receives the data from the device.</td>
</tr>
<tr>
<td>bytesToRead</td>
<td>Number of bytes to read from the device.</td>
</tr>
<tr>
<td>sizeOfRead</td>
<td>Pointer to a variable of type uint16 which receives the number of bytes</td>
</tr>
<tr>
<td></td>
<td>read from the device.</td>
</tr>
</tbody>
</table>

**Return Value:**
FT4222_OK if successful, otherwise the return value is an FT error code.

3.4.4 SPI Slave Write

**FT4222_STATUS FT4222_SPISlave_Write**(FT_HANDLE ftHandle, uint8* buffer, uint16 bytesToWrite, uint16* sizeTransferred)
Summary:
Write data to the transmit queue of the SPI slave device.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>buffer</td>
<td>Pointer to the buffer that contains the data to be written to the device.</td>
</tr>
<tr>
<td>bytesToWrite</td>
<td>Number of bytes to write to the device.</td>
</tr>
<tr>
<td>sizeTransferred</td>
<td>Pointer to a variable of type uint16 which receives the number of bytes written to the device.</td>
</tr>
</tbody>
</table>

Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.
3.5 SPI General Functions

3.5.1 SPI Reset Transaction

**FT4222_STATUS FT4222_SPI_ResetTransaction(FT_HANDLE ftHandle, uint8 spiIdx)**

**Summary:**
Reset the SPI transaction.

**Parameters:**

<table>
<thead>
<tr>
<th>ftHandle</th>
<th>Handle of the device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>spiIdx</td>
<td>The index of the SPI transaction, which ranges from 0~3 depending on the mode of the chip. For example, under mode 0 and mode 3 as we mentioned in chapter 1.1, it should be 0 because there is only one SPI master or slave connection, and so forth.</td>
</tr>
</tbody>
</table>

**Return Value:**
FT4222_OK if successful, otherwise the return value is an FT error code.

3.5.2 SPI Reset

**FT4222_STATUS FT4222_SPI_Reset (FT_HANDLE ftHandle)**

**Summary:**
Reset the SPI master or slave device.

**Parameters:**

| ftHandle | Handle of the device. |

**Return Value:**
FT4222_OK if successful, otherwise the return value is an FT error code.
3.6 I²C Master Functions

I²C (Inter Integrated Circuit) is a multi-master serial bus invented by Philips. I²C uses two bi-directional open-drain wires called serial data (SDA) and serial clock (SCL). Common I²C bus speeds are the 100 kbit/s standard mode (SM), 400 kbit/s fast mode (FM), 1 Mbit/s Fast mode plus (FM+), and 3.4 Mbit/s High Speed mode (HS).

The FT4222H device can be initialized as either an I²C master or I²C slave under mode 0 and mode 3. Here is a brief overview of FT4222H I²C features:

- Fully compatible to I²C v2.1 and v3 specification
- 7-bit address support
- Support 4 speed configurations: 100KHz(SM), 400KHz(FM), 1MHz(FM+), and 3.4MHz(HS).
- Clock stretching support in both master and slave mode.

Please refer to DS_FT4222H for more details.

3.6.1 I²C Master Init

FT4222_STATUS FT4222_I2CMaster_Init(FT_HANDLE ftHandle, uint32 kbps)

Summary:
Initialize the FT4222H as an I²C master with the requested I²C speed.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>kbps</td>
<td>The speed of I²C transmission. It ranges from 60K bps to 3400K bps. By specified speed, the initial function helps to setup bus speed with the corresponding mode. This parameter is used to configure FT2222H to be either SM, FB, FM+ or HS mode.</td>
</tr>
</tbody>
</table>

Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.

3.6.2 I²C Master Read

FT4222_STATUS FT4222_I2CMaster_Read(FT_HANDLE ftHandle, uint16 slaveAddress, uint8* buffer, uint16 bytesToRead, uint16* sizeTransferred)

Summary:
Read data from the specified I²C slave device.
### 3.6.3 I2C Master Write

FT4222_STATUS `FT4222_I2CMaster_Write`(FT_HANDLE ftHandle, uint16 slaveAddress, uint8* buffer, uint16 bytesToWrite, uint16* sizeTransferred)

**Summary:**
Write data to the specified I2C slave device.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>slaveAddress</td>
<td>Address of the target I2C slave.</td>
</tr>
<tr>
<td>buffer</td>
<td>Pointer to the buffer that contains the data to be written to the device.</td>
</tr>
<tr>
<td>bytesToWrite</td>
<td>Number of bytes to write to the device.</td>
</tr>
<tr>
<td>sizeTransferred</td>
<td>Pointer to a variable of type uint16 which receives the number of bytes read and written to the device.</td>
</tr>
</tbody>
</table>

**Return Value:**
FT4222_OK if successful, otherwise the return value is an FT error code.

### 3.6.4 I2C Master Reset

FT4222_STATUS `FT4222_I2CMaster_Reset`(FT_HANDLE ftHandle)

**Summary:**
Reset the I2C master device.

**Parameters:**
ftHandle | Handle of the device.

**Return Value:**
FT4222_OK if successful, otherwise the return value is an FT error code.
### 3.7 I²C Slave Functions

The FT4222H device can be initialized as an I²C slave under mode 0 and mode 3. It conforms to v2.1 and v3.0 of the I²C specification and supports all the transmission modes: Standard, Fast, Fast-plus and High Speed.

When the I²C slave receives data from the I²C bus, it will keep the data in its internal receive buffer (256 bytes), and then send the data to the USB host through IN packets.

When data is requested by an I²C master, data will be moved from an OUT packet to the transmit register directly.

#### 3.7.1 I²C Slave Init

FT4222\_STATUS FT4222\_I2CSalve\_Init(FT\_HANDLE ftHandle)

**Summary:**

Initialize FT4222H as an I²C slave.

**Parameters:**

<table>
<thead>
<tr>
<th>ftHandle</th>
<th>Handle of the device.</th>
</tr>
</thead>
</table>

**Return Value:**

FT4222\_OK if successful, otherwise the return value is an FT error code.

#### 3.7.2 I²C Slave Get Address

FT4222\_STATUS FT4222\_I2CSlave\_GetAddress(FT\_HANDLE ftHandle, uint8\* pAddr)

**Summary:**

Get the address of the I²C slave device.

**Parameters:**

<table>
<thead>
<tr>
<th>ftHandle</th>
<th>Handle of the device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pAddr</td>
<td>Pointer to a variable of type uint16 which receives the address of the I²C slave device.</td>
</tr>
</tbody>
</table>

**Return Value:**

FT4222\_OK if successful, otherwise the return value is an FT error code.
3.7.3 I2C Slave Set Address

FT4222_STATUS FT4222_I2CSlave_SetAddress(FT_HANDLE ftHandle, uint8 addr)

Summary:
Set the address of the I2C slave device.

Parameters:

<table>
<thead>
<tr>
<th>ftHandle</th>
<th>Handle of the device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr</td>
<td>The 7-bit address of the I2C slave device.</td>
</tr>
</tbody>
</table>

Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.

3.7.4 I2C Slave Get Rx Status

FT4222_STATUS FT4222_I2CSlave_GetRxStatus(FT_HANDLE ftHandle, uint16* pRxSize)

Summary:
Get number of bytes in the receive queue.

Parameters:

<table>
<thead>
<tr>
<th>ftHandle</th>
<th>Handle of the device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pRxSize</td>
<td>Pointer to a variable of type uint16 which receives the number of bytes in the receive queue.</td>
</tr>
</tbody>
</table>

Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.

3.7.5 I2C Slave Read

FT4222_STATUS FT4222_I2CSlave_Read(FT_HANDLE ftHandle, uint8* buffer, uint16 bytesToRead, uint16* sizeTransferred)

Summary:
Read data from the buffer of I2C slave device.

Parameters:
ftHandle | Handle of the device.
buffer | Pointer to the buffer that receives the data from the device.
bytesToRead | Number of bytes to read from the device.
sizeTransferred | Pointer to a variable of type uint16 which receives the number of bytes read from the device.

Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.

### 3.7.6 I2C Slave Write

FT4222_STATUS **FT4222_I2CSlave_Write**(FT_HANDLE ftHandle, uint8* buffer, uint16 bytesToWrite, uint16* sizeTransferred)

Summary:
Write data to the buffer of I²C slave device.

Parameters:

| ftHandle | Handle of the device. |
| buffer | Pointer to the buffer that contains the data to be written to the device. |
| bytesToWrite | Number of bytes to write to the device. |
| sizeTransferred | Pointer to a variable of type uint16 which receives the number of bytes read and written to the device. |

Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.

### 3.7.7 I2C Slave Reset

FT4222_STATUS **FT4222_I2CSlave_Reset**(FT_HANDLE ftHandle)

Summary:
Reset the I²C slave device.

Parameters:

| ftHandle | Handle of the device. |

Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.
3.8 GPIO Functions

The FT4222H contains 4 GPIO. When the USB GPIO interface is supported, chip mode 0 and mode 1, LibFT4222 helps application developers to control GPIO directly. However, each GPIO pin has multiplexed with interrupt/suspend out/SPI slave select/I2C functions as list below:

- gpio0 / ss1o / scl
- gpio1 / ss2o / sda
- gpio2 / ss3o / suspend out
- gpio3 / wakeup/intr

The number of GPIO pins available depends on the mode of the chip. For example, If the FT4222H is initialized as an I²C device, as shown above, the pins of gpio0 and gpio1 will be switched to scl and sda, and cannot be used as GPIO. If suspend out and remote wakeup are enabled gpio2 and gpio3 cannot be used as GPIO.

The FT4222H supports GPIO on the second USB interface in mode 0 or on the fourth interface in mode 2 (Please refer table 2.1 for chip mode and interface).

3.8.1 GPIO Init

FT4222_STATUS FT4222_GPIO_Init(FT_HANDLE ftHandle, GPIO_Dir gpioDir[4])

Summary:

Initialize the GPIO interface of the FT4222H.

Please note the GPIO interface is available on the 2nd USB interface in mode 0 or on the 4th USB interface in mode 1.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>gpioDir</td>
<td>An array defines the directions of 4 GPIO pins. The GPIO direction will be:</td>
</tr>
<tr>
<td></td>
<td>• GPIO_OUTPUT</td>
</tr>
<tr>
<td></td>
<td>• GPIO_INPUT</td>
</tr>
</tbody>
</table>

Return Value:

FT4222_OK if successful, otherwise the return value is an FT error code.

3.8.2 GPIO Read

FT4222_STATUS FT4222_GPIO_Read(FT_HANDLE ftHandle, GPIO_Port portNum, BOOL* pValue)
Summary:
Read the value from the specified GPIO pin.

Parameters:

<table>
<thead>
<tr>
<th>ftHandle</th>
<th>Handle of the device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>portNum</td>
<td>One of the following GPIO ports:</td>
</tr>
<tr>
<td></td>
<td>• GPIO_PORT0</td>
</tr>
<tr>
<td></td>
<td>• GPIO_PORT1</td>
</tr>
<tr>
<td></td>
<td>• GPIO_PORT2</td>
</tr>
<tr>
<td></td>
<td>• GPIO_PORT3</td>
</tr>
<tr>
<td>pValue</td>
<td>Pointer to a variable of type BOOL which receives the value of the GPIO pin.</td>
</tr>
</tbody>
</table>

Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.

3.8.3 GPIO Write
FT4222_STATUS FT4222_GPIO_Write(FT_HANDLE ftHandle, GPIO_Port portNum, BOOL bValue)

Summary:
Write value to the specified GPIO pin.

Parameters:

<table>
<thead>
<tr>
<th>ftHandle</th>
<th>Handle of the device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>portNum</td>
<td>One of the following GPIO port:</td>
</tr>
<tr>
<td></td>
<td>• GPIO_PORT0</td>
</tr>
<tr>
<td></td>
<td>• GPIO_PORT1</td>
</tr>
<tr>
<td></td>
<td>• GPIO_PORT2</td>
</tr>
<tr>
<td></td>
<td>• GPIO_PORT3</td>
</tr>
<tr>
<td>bValue</td>
<td>The output value.</td>
</tr>
</tbody>
</table>

Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.
3.8.4  GPIO Set Input Trigger

FT4222_STATUS FT4222_GPIO_SetInputTrigger(FT_HANDLE ftHandle, GPIO_Port portNum, GPIO_Tigger trigger)

**Summary:**
Set software trigger conditions on the specified GPIO pin.

This function allows developers to monitor value changes of the GPIO pins. Values that satisfy the trigger condition will be stored in a queue. For example, if GPIO_TRIGGER_RISING is set on GPIO0, and then GPIO0 changes value from 0 to 1, the event GPIO_TRIGGER_RISING will be recorded into the queue. Developers can query the queue status by FT4222_GPIO_GetTriggerStatus, and FT4222_GPIO_ReadTriggerQueue.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>portNum</td>
<td>One of the following GPIO port:</td>
</tr>
<tr>
<td></td>
<td>• GPIO_PORT0</td>
</tr>
<tr>
<td></td>
<td>• GPIO_PORT1</td>
</tr>
<tr>
<td></td>
<td>• GPIO_PORT2</td>
</tr>
<tr>
<td></td>
<td>• GPIO_PORT3</td>
</tr>
<tr>
<td>trigger</td>
<td>Combination of the following trigger conditions:</td>
</tr>
<tr>
<td></td>
<td>• GPIO_TRIGGER_RISING</td>
</tr>
<tr>
<td></td>
<td>• GPIO_TRIGGER_FALLING</td>
</tr>
<tr>
<td></td>
<td>• GPIO_TRIGGER_LEVEL_HIGH</td>
</tr>
<tr>
<td></td>
<td>• GPIO_TRIGGER_LEVEL_LOW</td>
</tr>
</tbody>
</table>

**Return Value:**
FT4222_OK if successful, otherwise the return value is an FT error code.

3.8.5  GPIO Get Trigger Status

FT4222_STATUS FT4222_GPIO_GetTriggerStatus(FT_HANDLE ftHandle, GPIO_Port portNum, uint16* pQueueSize)

**Summary:**
Get the size of trigger event queue.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
</tbody>
</table>
3.8.6 GPIO Read Trigger Queue

FT4222_STATUS FT4222_GPIO_ReadTriggerQueue(FT_HANDLE ftHandle, GPIO_Port portNum, GPIO_Tigger* events, uint16 readSize, uint16* sizeofRead)

Summary:
Get events recorded in the trigger event queue. Trigger conditions are set by a call to FT4222_GPIO_SetInputTrigger. After calling this function all events will be removed from the event queue.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftHandle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>portNum</td>
<td>One of the following GPIO port:</td>
</tr>
<tr>
<td>events</td>
<td>Pointer to the buffer that receives the values of trigger event queue. The</td>
</tr>
<tr>
<td>readSize</td>
<td>Number of bytes to read from trigger event queue.</td>
</tr>
<tr>
<td>sizeofRead</td>
<td>Pointer to a variable of type uint16 which receives the number of bytes</td>
</tr>
<tr>
<td></td>
<td>read from the queue.</td>
</tr>
</tbody>
</table>

Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.
Return Value:
FT4222_OK if successful, otherwise the return value is an FT error code.
4 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

Branch Office – Tigard, Oregon, USA
Future Technology Devices International Limited
(USA)
7130 SW Fir Loop
Tigard, OR 97223
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

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 8791 3570
Fax: +886 (0) 2 8791 3576
E-mail (Sales) tw.sales1@ftdichip.com
E-mail (Support) tw.support1@ftdichip.com
E-mail (General Enquiries) tw.admin1@ftdichip.com

Branch Office – Shanghai, China
Future Technology Devices International Limited
(China)
Room 408, 317 Xianxia Road,
Shanghai, 200051
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.admin@ftdichip.com

Web Site
http://ftdichip.com

System and equipment manufacturers and designers are responsible to ensure that their systems, and any Future Technology Devices International Ltd (FTDI) devices incorporated in their systems, meet all applicable safety, regulatory and system-level performance requirements. All application-related information in this document (including application descriptions, suggested FTDI devices and other materials) is provided for reference only. While FTDI has taken care to assure it is accurate, this information is subject to customer confirmation, and FTDI disclaims all liability for system designs and for any applications assistance provided by FTDI. Use of FTDI devices in life support and/or safety applications is entirely at the user’s risk, and the user agrees to defend, indemnify and hold harmless FTDI from any and all damages, claims, suits or expense resulting from such use. This document is subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Neither the whole nor any part of the information contained in, or the product described in this document, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. Future Technology Devices International Ltd, Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow G41 1HH, United Kingdom. Scotland Registered Company Number: SC136640
Appendix A – Enumeration and Structure Definitions

**Enumeration**

**FT4222_STATUS**

```c
FT4222_DEVICE_NOT_SUPPORTED = 1000
FT4222_CLK_NOT_SUPPORTED // spi master do not support 80MHz/CLK_2
FT4222_VENDER_CMD_NOT_SUPPORTED
FT4222_IS_NOT_SPI_MODE
FT4222_IS_NOT_I2C_MODE
FT4222_IS_NOT_SPI_SINGLE_MODE
FT4222_IS_NOT_SPI_MULTI_MODE
FT4222_WRONG_I2C_ADDR
FT4222_INVALID_FUNCTION
FT4222_INVALID_POINTER
FT4222_EXCEEDED_MAX_TRANSFER_SIZE
FT4222_FAILED_TO_READ_DEVICE
FT4222_I2C_NOT_SUPPORTED_IN_THIS_MODE
FT4222_GPIO_NOT_SUPPORTED_IN_THIS_MODE
FT4222_GPIO_EXCEEDED_MAX_PORTNUM
FT4222_GPIO_WRITE_NOT_SUPPORTED
FT4222_GPIO_PULLUP_INVALID_IN_INPUTMODE
FT4222_GPIO_PULLDOWN_INVALID_IN_INPUTMODE
FT4222_GPIO_OPENDRAIN_INVALID_IN_OUTPUTMODE
FT4222_INTERRUPT_NOT_SUPPORTED
FT4222_GPIO_INPUT_NOT_SUPPORTED
FT4222_EVENT_NOT_SUPPORTED
```

**FT4222_ClockRate**

```c
SYS_CLK_60 = 0
SYS_CLK_24
SYS_CLK_48
SYS_CLK_80
```

**FT4222_SPIMode**

```c
SPI_IO_NONE = 0
```
SPI_IO_SINGLE = 1
SPI_IO_DUAL = 2
SPI_IO_QUAD = 4

FT4222_SPIClock

CLK_NONE = 0
CLK_DIV_2 // 1/2 System Clock
CLK_DIV_4 // 1/4 System Clock
CLK_DIV_8 // 1/8 System Clock
CLK_DIV_16 // 1/16 System Clock
CLK_DIV_32 // 1/32 System Clock
CLK_DIV_64 // 1/64 System Clock
CLK_DIV_128 // 1/128 System Clock
CLK_DIV_256 // 1/256 System Clock
CLK_DIV_512 // 1/512 System Clock

FT4222_SPICPOL

CLK_ACTIVE_LOW = 0
CLK_ACTIVE_HIGH = 1

FT4222_SPICPHA

CLK_LEADING = 0
CLK_TRAILING = 1

SPI_DrivingStrength

DS_4MA
DS_8MA
DS_12MA
DS_16MA

denum GPIO_Port

GPIO_PORT0
GPIO_PORT1
GPIO_PORT2
GPIO_PORT3

denum GPIO_Dir

GPIO_OUTPUT
GPIO_INPUT

enum GPIO_Tigger
    {            
        GPIO_TRIGGER_RISING // Change Me!!!
        GPIO_TRIGGER_FALLING
        GPIO_TRIGGER_LEVEL_HIGH
        GPIO_TRIGGER_LEVEL_LOW
    }

enum GPIO_Output
    {            
        GPIO_OUTPUT_LOW
        GPIO_OUTPUT_HIGH
    }

**Structure Definitions**

```c
struct FT4222_Version
{
    DWORD chipVersion; // The version of FT4222H chip
    DWORD dllVersion;  // The version of LibFT4222
};
```

```c
struct SPI_Slave_Header
{
    uint8          syncWord;
    uint8          cmd;
    uint8          sn;
    uint16         size;
};
```
Appendix B – References

Document References

DS_FT4222H
D2XX Programmers Guide
D2XX Drivers

Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Terms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2XX</td>
<td>FTDI’s proprietary “direct” driver interface via FTD2XX.DLL</td>
</tr>
<tr>
<td>GPIO</td>
<td>General-purpose input/output</td>
</tr>
<tr>
<td>I2C</td>
<td>Inter-Integrated Circuit</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interconnect</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>USB-IF</td>
<td>USB Implementers Forum</td>
</tr>
</tbody>
</table>
Appendix C – List of Tables & Figures

List of Tables
Table 2.1 Chip mode and device functions ................................................................. 8

List of Figures
Figure 1.1 The software stack ......................................................................................... 4
Figure 1.2 Mode 0: FT4222H works as SPI master (quad mode) ................................. 6
Figure 1.3 Mode 0: FT4222H works as I²C master ...................................................... 6
Figure 1.4 Mode 2: FT4222H works as SPI master ..................................................... 7
Figure 3.1 SPI full duplex communication .................................................................... 21
Figure 3.2 Dual SPI communication ............................................................................. 22
Figure 3.3 Quad SPI communication ............................................................................ 22
Figure 3.4 SPI Slave Protocol Format .......................................................................... 24
Figure 3.5 SPI Master transfer request ........................................................................ 25
Figure 3.6 An example of the SPI slave responding with ACK ..................................... 25
Figure 3.7 An example of when the SPI master doesn’t receive ACK .......................... 26
Figure 3.8 Slave sends transfer request ....................................................................... 26
Appendix D – Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Changes</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Initial Release</td>
<td>2014-09-16</td>
</tr>
</tbody>
</table>

Document Title: AN_329 User Guide For LibFT4222
Document Reference No.: FT_001060
Clearance No.: FTDI#406
Document Feedback: Send Feedback