Application Note

AN_329

User Guide for LibFT4222

Version 1.7

Issue Date: 11-10-2023

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

Future Technology Devices International Limited (FTDI)
Unit 1, 2 Seaward Place, Glasgow G41 1HH, United Kingdom
Tel.: +44 (0) 141 429 2777 Fax: + 44 (0) 141 429 2758
Web Site: http://ftdichip.com

Table of Contents

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, which provides high-level APIs to facilitate user application development. At the time of writing support for Windows and Linux OS has been published. Android support uses a different package also available from the FTDI website. 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.

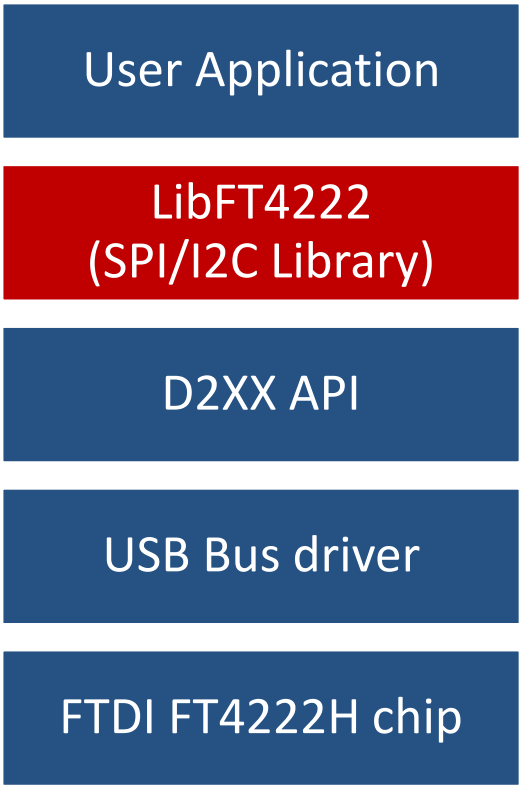


Figure 1.1 The Software Stack

Note that the Window, Linux and MAC OS version of LibFT4222 have D2XX and mini-boost built-in. The LibFT4222 sample code, release notes, and all necessary files can be downloaded from the FTDI website at: https://ftdichip.com/products/ft4222h/

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 operation 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, I²C master, or I²C 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, I²C master, or I²C slave device

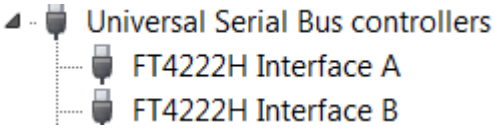
Operation mode is configured by DCNF0 & DCNF1 pins, please see below table for detail

Operation Mode	DCNF0	DCMF1
0	0	0
1	1	0
2	0	1
3	1	1

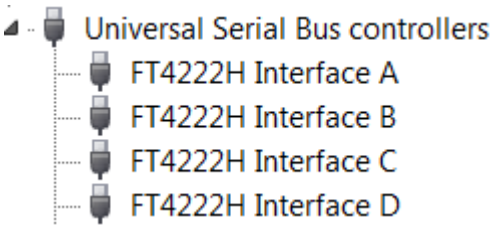
Table 1.1 Chip Mode with DCNF0 and DCNF1

The following shows the interface of different operation mode which displayed in device manager of Windows OS.

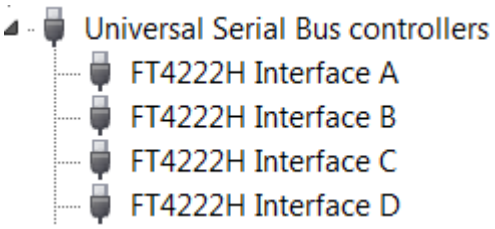
Chip mode 0 USB interfaces:



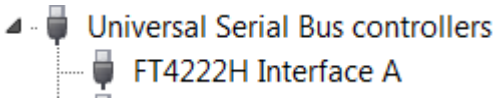
Chip mode 1 USB interfaces:



Chip mode 2 USB interfaces:



Chip mode 3 USB interfaces:



In mode 0 and 3, the connected device can be a 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 in mode 0 and mode 1, 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 device, with pins as 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 gpio2 is configured as suspend out, and the pin for gpio3 is configured as wakeup/intr. Only those configured GPIO pins can support GPIO read/set operation through the corresponding endpoint.

Figure 1.2, **Figure 1.3**, and **Figure 1.4** shows the examples of FT4222H SPI/I^{2}C master connections.

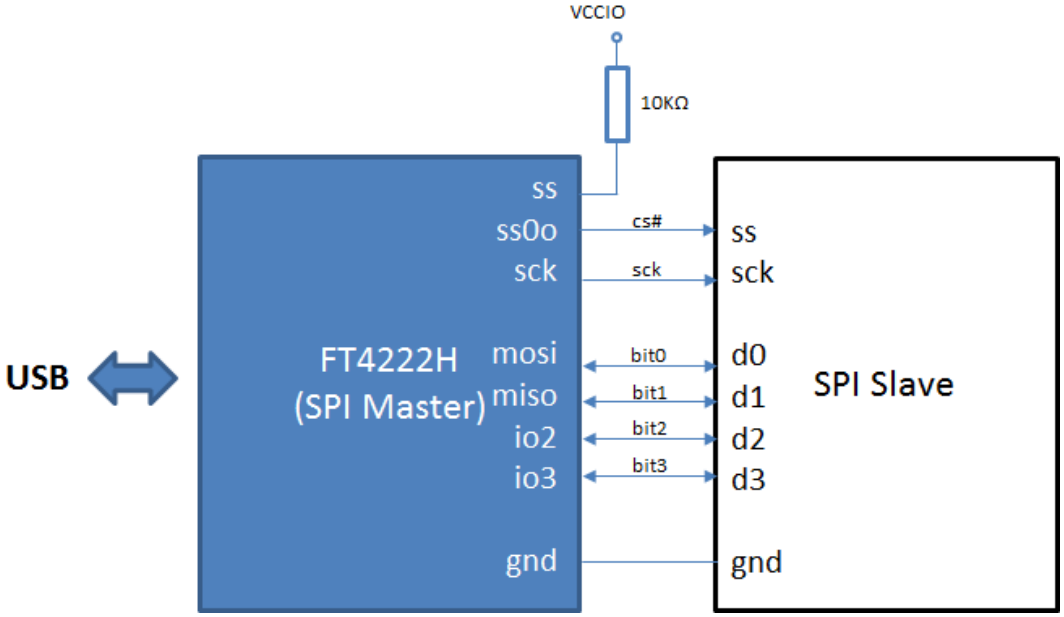


Figure 1.2 Mode 0: FT4222H works as SPI master (Quad Mode)

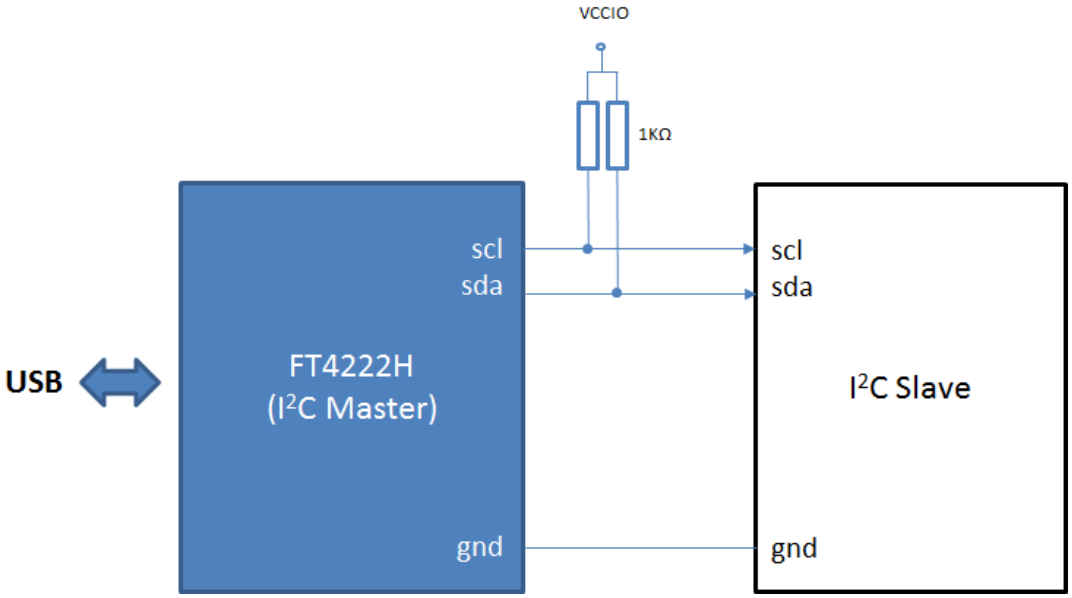


Figure 1.3 Mode 0: FT4222H works as I²C Master

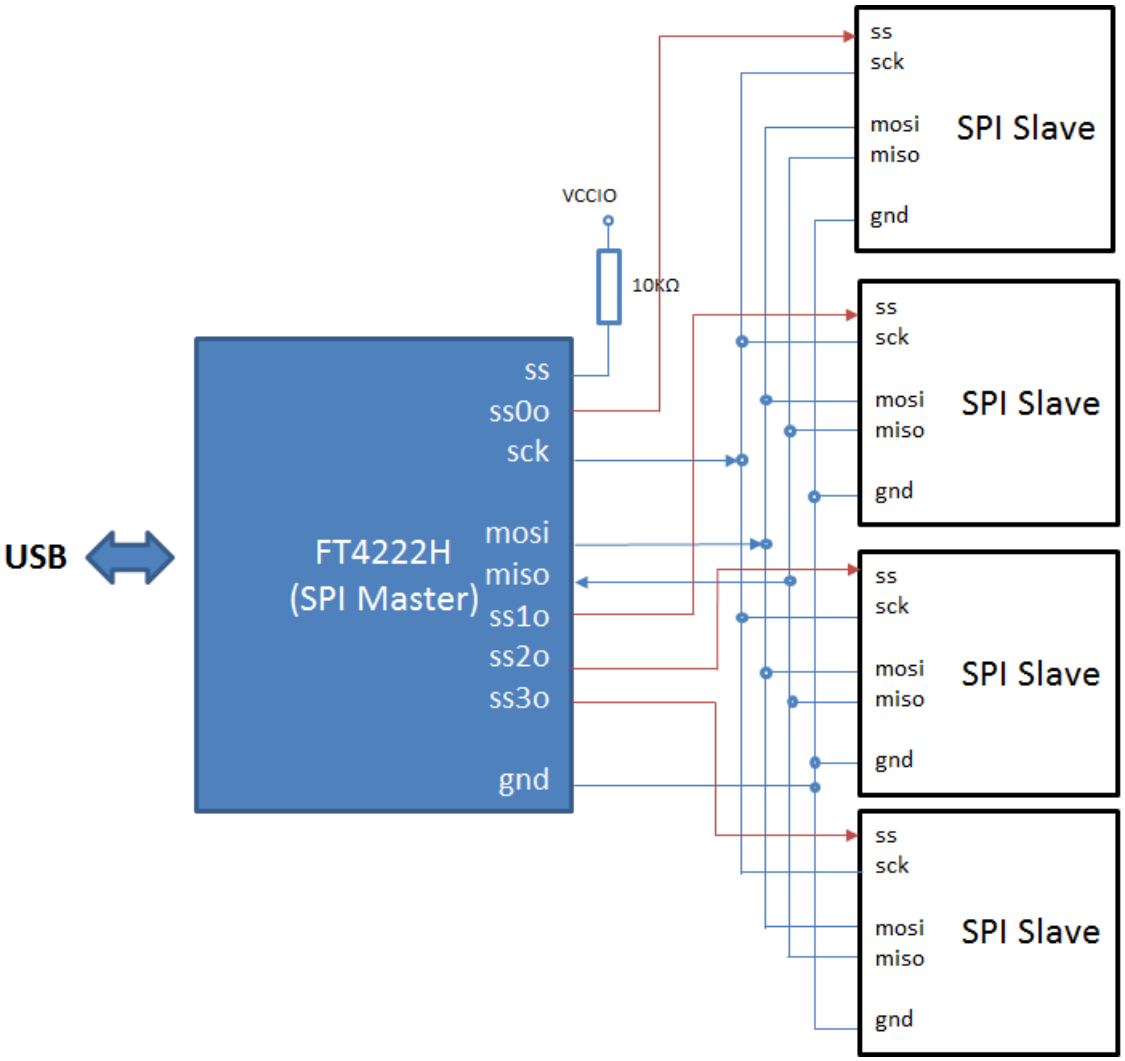


Figure 1.4 Mode 2: FT4222H works as SPI Master

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.

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 **Table 2.1**.

Mode	Number of Interfaces	Device Function
0	2	a. The first interface: it can be one of SPI master, SPI slave, I^2C master, or I^2C 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. It can be one of SPI master, SPI slave, I^2C master, or I^2C 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, I^2C master, or I^2C slave. Different types of devices require different configurations. For more details, please refer the next chapter.

Following example code shows FT4222H works in SPI master mode.

Example#

```
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;
    FT4222_STATUS ft4222Status;
    ftStatus = FT_OpenEx((PVOID)g_FT4222DevList[0].LocId,
        FT_OPEN_BY_LOCATION, &ftHandle);
    if (FT_OK != ftStatus)  {
        printf("Open a FT4222 device failed!\n");
        return 0;
    }
    ft4222Status = FT4222_SPIMaster_Init(ftHandle,
        SPI_IO_SINGLE, CLK_DIV_4, CLK_ACTIVE_LOW, CLK_LEADING, 0x01);
    if (FT4222_OK != ft4222Status)  {
        printf("Init FT4222 as SPI master device failed!\n");
        return 0;
    }
    // TODO:
    //    Start to work as SPI master, and read/write data to an 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, I^{2}C 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 (see the D2XX Programmer's Guide). FT4222_STATUS defines additional values to report FT4222H specific status.

All definitions with prefix "FT_" are defined in the D2XX driver.

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 to the D2XX Programmers Guide 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 I^{2}C 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)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Release allocated resources. FT4222_Uninitialize should be called before calling FT_Close. FT4222_Uninitialize must be called after one of the following API.

- FT4222_SPIMaster_Init
- FT4222_SPISlave_Init
- FT4222_I2CMaster_Init
- FT4222_I2CSlave_Init
- FT4222_GPIO_Init

Parameters:

ftHandle	Handle of the device.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;

ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
    // open failed
    return;
}
ft4222Status = FT4222_SPIMaster_Init(ftHandle, SPI_IO_SINGLE, CLK_DIV_4, CLK_IDLE_LOW,
CLK_LEADING, 0x01);
if (FT4222_OK != ft4222Status)
{
    // spi master init failed
    return;
}

FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.2.3 Set Clock

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

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

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:

ftHandle	Handle of the device.
clk	FT4222 system clock rate: • SYS_CLK_60 • SYS_CLK_24 • SYS_CLK_48 • SYS_CLK_80

Return Value:

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

Error code:

FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.

Example:

```
    FT_HANDLE ftHandle = NULL;
    FT_STATUS ftStatus;
    FT4222_STATUS ft4222Status;

    ftStatus = FT_Open(0, &ftHandle);
    if (FT_OK != ftStatus)
    {
        // open failed
        return;
    }
     // set system clock to 80MHz
    ft4222Status = FT4222_SetClock(ftHandle, SYS_CLK_80);
    if (FT4222_OK != ft4222Status)
    {
        // set clock failed
        return;
    }
FT_Close(ftHandle);
```

3.2.4 Get Clock

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

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Get the current system clock rate.

Parameters:

ftHandle	Handle of the device.
pClk	Pointer to a variable of type FT4222_ClockRate where the value will be stored.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.
FT4222_INVALID_POINTER: Parameter pClk is NULL.

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
FT4222_ClockRate clk;

ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
    // open failed
    return;
}

ft4222Status = FT4222_GetClock(ftHandle, &clk);
if (FT4222_OK != ft4222Status)
{
    // get clock failed
    return;
}

FT_Close(ftHandle);
```

3.2.5 Set Suspend Out

FT4222_STATUS **FT4222_SetSuspendOut**(FT_HANDLE ftHandle, BOOL enable)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

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. By default, suspend-out function is on.

When suspend-out function is on, suspend-out pin emits signal according to suspend-out polarity. The default value of suspend-out polarity is active high. It means suspend-out pin output low in normal mode and output high in suspend mode. Suspend-out polarity only can be adjusted by FT_PROG.

Parameters:

ftHandle	Handle of the device.
Enable	TRUE to enable suspend out and configure GPIO2 as an output pin for emitting a signal when suspended. FALSE to switch back to GPIO2.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;

ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
    // open failed
    return;
}

ft4222Status = FT4222_SetSuspendOut(ftHandle, TRUE);
if (FT4222_OK != ft4222Status)
{
    // set suspend failed
    return;
}

FT_Close(ftHandle);
```

3.2.6 Set Wake Up/Interrupt

FT4222_STATUS **FT4222_SetWakeUpInterrupt**(FT_HANDLE ftHandle, BOOL enable)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Enable or disable wakeup/interrupt. By default, wake-up/interrupt function is on.

When Wake up/Interrupt function is on, GPIO3 pin acts as an input pin for wakeup/interrupt.

While system is in normal mode, GPIO3 acts as an interrupt pin. While system is in suspend mode, GPIO3 acts as a wakeup pin. An example is provided with the support-lib. The file is located in the following path:

example\samples\interrupt\interrupt.cpp

Parameters:

ftHandle	Handle of the device.
enable	TRUE to configure GPIO3 as an input pin for wakeup/interrupt. FALSE to switch back to GPIO3.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.

Example:

Please refer to the example in FT4222_SetInterruptTrigger.

3.2.7 Set Interrupt Trigger Condition

FT4222_STATUS **FT4222_SetInterruptTrigger**(FT_HANDLE ftHandle, GPIO_Trigger trigger)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Set trigger condition for the pin wakeup/interrupt. By default, the trigger condition is GPIO_TRIGGER_RISING.

This function configures trigger condition for wakeup/interrupt.

When GPIO3 acts as wakeup pin. It means that ft4222H device has the capability to wake up the host. Only GPIO_TRIGGER_RISING and GPIO_TRIGGER_FALLING is valid when GPIO3 act as a wakeup pin. It is not necessary to call FT4222_GPIO_Init to set up wake-up function.

When GPIO3 acts as interrupt pin. All trigger condition can be set. The result of trigger status can be inquired by FT4222_GPIO_ReadTriggerQueue or FT4222_GPIO_Read. This is because the trigger status is provided by the GPIO pipe. Therefore, it is necessary to call FT4222_GPIO_Init to set up interrupt function.

For GPIO triggering conditions, GPIO_TRIGGER_LEVEL_HIGH and GPIO_TRIGGER_LEVEL_LOW, that can be configured when GPIO3 behaves as an interrupt pin, when the system enters suspend mode, these two configurations will act as GPIO_TRIGGER_RISING and GPIO_FALLING respectively.

Parameters:

ftHandle	Handle of the device.
trigger	Trigger condition. One of the following: • GPIO_TRIGGER_RISING • GPIO_TRIGGER_FALLING • GPIO_TRIGGER_LEVEL_HIGH • GPIO_TRIGGER_LEVEL_LOW

Return Value:

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

Error code:

FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.
FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_INTERRUPT_NOT_SUPPORTED: interrupt/wakeup is disabled.
FT4222_INVALID_PARAMETER: parameter trigger is invalid

Example:

```
// example 1: This test code is running in Mode 0.
// sending notification while there is an interrupt happen.

   FT_HANDLE ftHandle = NULL;
   FT_STATUS ftStatus;
   FT4222_STATUS ft4222Status;
   ftStatus = FT_OpenEx("FT4222 B",FT_OPEN_BY_DESCRIPTION, &ftHandle);
   if (FT_OK != ftStatus)
   {
     // open failed
     return;
   }
   HANDLE hRxEvent;
   hRxEvent = CreateEvent(
                NULL,
                false, // auto-reset event
                false, // non-signalled state
                NULL   );        ftStatus  =  FT_SetEventNotification(ftHandle,  FT_EVENT_RXCHAR,
hRxEvent);
   if (FT_OK != ftStatus)
   {
      // FT_SetEventNotification failed
      return ;
```

```
    }
    GPIO_Dir gpioDir[4];
    gpioDir[0] = GPIO_OUTPUT;
    gpioDir[1] = GPIO_OUTPUT;
    gpioDir[2] = GPIO_OUTPUT;
    gpioDir[3] = GPIO_INPUT;

// we must initialize gpio before FT4222_SetInterruptTrigger, because interrupt data is transmitted
by gpio interface.

    FT4222_GPIO_Init(ftHandle, gpioDir);
    // enable interrupt
    FT4222_SetWakeUpInterrupt(ftHandle, true);
    // setup interrupt trigger level
    FT4222_SetInterruptTrigger(ftHandle, GPIO_TRIGGER_RISING);
    while(1)
    {
        BOOL value;
        WaitForSingleObject(hRxEvent,INFINITE);
        //  FT4222_GPIO_Read is a read clear function for interrupt
        if(FT4222_GPIO_Read(ftHandle, (GPIO_Port)GPIO_PORT3, &value) == FT4222_OK)
        {
            if(value == TRUE)
            {
                // got interrupt
            }
            else
            {
                // no interrupt
            }
        }
    }

    FT4222_UnInitialize(ftHandle);
    FT_Close(ftHandle);

// example 2: This test code is running in Mode 0.
// Monitor how many interrupts happen in a period of time.
    FT_HANDLE ftHandle = NULL;
    FT_STATUS ftStatus;
    FT4222_STATUS ft4222Status;

    ftStatus = FT_OpenEx("FT4222 B",FT_OPEN_BY_DESCRIPTION, &ftHandle);
    if (FT_OK != ftStatus)
    {
      // open failed
      return;
    }

    GPIO_Dir gpioDir[4];
    gpioDir[0] = GPIO_OUTPUT;
    gpioDir[1] = GPIO_OUTPUT;
    gpioDir[2] = GPIO_OUTPUT;
    gpioDir[3] = GPIO_INPUT;

// we must initial gpio before FT4222_SetInterruptTrigger, because interrupt data is transmitted
by gpio interface.

    FT4222_GPIO_Init(ftHandle, gpioDir);
    // enable interrupt
    FT4222_SetWakeUpInterrupt(ftHandle, true);
```

```
// setup interrupt trigger level
FT4222_SetInterruptTrigger(ftHandle, GPIO_TRIGGER_RISING);

while(1)
{
   uint16 queueSize;
   // sleep 1s
   Sleep(1000);
   if(FT4222_GPIO_GetTriggerStatus(ftHandle, GPIO_PORT3, &queueSize) == FT4222_OK)
   {
      // got interrupt times in 1s
      if(queueSize > 0)
      {
         BOOL value;
         // clear the interrupt result
         FT4222_GPIO_Read(ftHandle, (GPIO_Port)GPIO_PORT3, &value);
      }
   }
}

FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.2.8 Get Max Transfer Size

FT4222_STATUS **FT4222_GetMaxTransferSize**(FT_HANDLE ftHandle, uint16* pMaxSize)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

This function returns the maximum packet size in a transaction. It will be affected by different bus speeds, chip modes, and functions. The maximum transfer size is maximum size in writing path.

Parameters:

ftHandle	Handle of the device.
pMaxSize	Pointer to a variable of type unit16 where the returning value will be stored.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_INVALID_POINTER: Parameter pMaxSize is NULL

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
uint16 maxSize;
```

```
    ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
    // open failed
    return;
}
ft4222Status = FT4222_I2CMaster_Init(ftHandle, 1000);
if (FT4222_OK != ft4222Status)
{
    // init i2c master failed
    return;
}
ft4222Status = FT4222_GetMaxTransferSize(ftHandle, &maxSize);
if (FT4222_OK != ft4222Status)
{
    // get max transfer size failed
    return;
}

FT_Close(ftHandle);
```

3.2.9 Set Event Notification

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

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Sets conditions for event notification.

An application can use this function to set up 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. The API is only valid when the device acts as SPI slave and SPI slave protocol is not SPI_SLAVE_NO_PROTOCOL.

Parameters:

ftHandle	Handle of the device.
dwEventMask	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: • **FT4222_EVENT_RXCHAR** The event will be set when a data packet has been received by the device.
pvArg	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.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_EVENT_NOT_SUPPORTED: The device must act as SPI slave and protocol is not SPI_SLAVE_NO_PROTOCOL.

Prerequisite:

FT4222_SPISlave_InitEx or FT4222_SPISlave_Init

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
HANDLE hRxEvent;

ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
   // open failed
   return;
}

ft4222Status = FT4222_SPISlave_InitEx(ftHandle, SPI_SLAVE_NO_ACK);
if (FT4222_OK != ft4222Status)
{
   // init spi slave failed
   return;
}

hRxEvent = CreateEvent(
            NULL,
            false, // auto-reset event
            false, // non-signalled state
            NULL );
ft4222Status = FT4222_SetEventNotification(ftHandle, FT4222_EVENT_RXCHAR, hRxEvent);
if (FT4222_OK != ft4222Status)
{
   //set event notification failed
   return;
}

uint16 rxSize;
uint16 sizeTransferred;

while(1)
{
   WaitForSingleObject(hRxEvent, 1000);
   ft4222Status = FT4222_SPISlave_GetRxStatus(ftHandle, &rxSize);
   if(ft4222Status == FT4222_OK)
   {
      if(rxSize>0)
      {
         std::vector<unsigned char> tmpBuf;
         tmpBuf.resize(rxSize);
         ft4222Status = FT4222_SPISlave_Read(ftHandle, &tmpBuf[0], rxSize,
&sizeTransferred);
         // handle receive data
      }
   }
```

```
}

FT_Close(ftHandle);
```

3.2.10 Get Version

FT4222_STATUS **FT4222_GetVersion**(FT_HANDLE ftHandle, FT4222_Version* pVersion)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Get the versions of FT4222H and LibFT4222.

Parameters:

ftHandle	Handle of the device.

pVersion	Pointer to a variable of type FT4222_Version where the value will be stored. Type FT4222_Version is defined as follows:
	
	`struct FT4222_Version`
	`{`
	`  DWORD chipVersion; // The version of FT4222H chip`
	`  DWORD dllVersion;  // The version of LibFT4222`
	`};`
	
	Revision A chips report chipVersion as 0x42220100.
	Revision B chips report chipVersion as 0x42220200.
	Revision C chips report chipVersion as 0x42220300.
	Revision D chips report chipVersion as 0x42220400.
	Revision B chips require version 1.2 or later of LibFT4222, indicated by dllVersion being greater than 0x01020000;
	Revision C chips require version 1.3 or later of LibFT4222, indicated by dllVersion being greater than 0x01030000;
	Revision D chips require version 1.4 or later of LibFT4222, indicated by dllVersion being greater than 0x01040000.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.
FT4222_INVALID_POINTER: Parameter pVersion is NULL.

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
FT4222_Version ver;
ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
```

```
{
    // open failed
    return;
}
ft4222Status = FT4222_GetVersion(ftHandle, &ver);
if (FT4222_OK != ft4222Status)
{
    // get version failed
    return;
}
printf("%x  %x\n",ver.chipVersion,ver.dllVersion);
FT_Close(ftHandle);
```

3.2.11 Chip Reset

FT4222_STATUS **FT4222_ChipReset**(FT_HANDLE ftHandle)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Software reset for device.

This function is used to attempt to recover system after a failure. It is a software reset for device.

Parameters:

ftHandle	Handle of the device.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
    ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
    // open failed
    return;
}
ft4222Status = FT4222_ChipReset(ftHandle);
if (FT4222_OK == ft4222Status)
{
    // chip has been reset
}
else
```

```
{
   // chip reset failed
}
FT_Close(ftHandle);
```

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 to DS_FT4222H for more details.

For SPI Master Single mode, all data packets are terminated with a zero-length packet. Therefore, after one data packet there will be one SOF then follows by the terminating zero-length packet then ends with another SOF. As a result, under normal conditions, these two SOF's will take approximately 250us.

3.3.1 SPI Master Init

FT4222_STATUS **FT4222_SPIMaster_Init**(FT_HANDLE ftHandle,

FT4222_SPIMode ioLine, FT4222_SPIClock clock_div,
FT4222_SPICPOL cpol, FT4222_SPICPHA cpha, uint8 ssoMap)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Initialize the FT4222H as an SPI master.

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

- IO lines: SPI transmission lines. The FT4222H SPI supports single, dual, or quad transmission mode. An application may override this initial selection dynamically using FT4222_SPIMaster_SetLines. For example, commands might be sent in single mode, but data transferred in dual or quad mode.
- Clock divider: SPI clock rate is subject to system clock. The FT4222H SPI clock could 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: Idle high or idle 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. The default slave selection is active low.
- There is only one setting stored in the MCU. If there are multi-SPI masters to be initialized, keep all settings the same, including ssoMap.

Please note that the FT4222H has only one SPI controller. Even though the FT4222H provides up to 4 interfaces for connecting up to 4 SPI slave devices as per Figure 1.4, the 4 slave devices share the same SPI data bus: MOSI, MISO, and SCK. A user can decide how to map the 4 interfaces to the 4 SS signals (ss0o, ss1o, ss2o and ss3o) by the *ssoMap* parameter.

The 4 interfaces cannot work simultaneously because there is only one data bus.

Parameters:

ftHandle	Handle of the device.
ioLine	SPI transmission lines: • SPI_IO_SINGLE • SPI_IO_DUAL • SPI_IO_QUAD
clock_div	Clock divider: • 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)
cpol	Clock polarity: • CLK_IDLE_LOW • CLK_IDLE_HIGH
cpha	Clock phase: • CLK_LEADING • CLK_TRAILING
ssoMap	Slave selection output pins. It's a bitmap: • Bit 0: select device connected with ss0o • Bit 1: select device connected with ss1o • Bit 2: select device connected with ss2o • Bit 3: select device connected with ss3o

Return Value:

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

Error code:

FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.
FT4222_INVALID_PARAMETER: Parameter is not suitable.

Example: single SPI master initialization

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;

ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
    // open failed
    return;
}

ft4222Status = FT4222_SPIMaster_Init(ftHandle, SPI_IO_SINGLE, CLK_DIV_4, CLK_IDLE_LOW,
CLK_LEADING, 0x01);
```

```
if (FT4222_OK != ft4222Status)
{
    // spi master init failed
    return;
}

FT4222_UnInitialize(ftHandle);

FT_Close(ftHandle);
```

Example: multiple SPI master initialization, this sample runs in Mode 1 or Mode 2

```
FT_HANDLE ftHandle1 = NULL;
FT_HANDLE ftHandle2 = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;

ftStatus = FT_Open(0, &ftHandle1);
if (FT_OK != ftStatus)
{
    // open failed
    return;
}
ftStatus = FT_Open(1, &ftHandle2);
if (FT_OK != ftStatus)
{
    // open failed
    return;
}

ft4222Status = FT4222_SPIMaster_Init(ftHandle1, SPI_IO_SINGLE, CLK_DIV_4,
CLK_IDLE_LOW, CLK_LEADING, 0x03);
if (FT4222_OK != ft4222Status)
{
    // spi master init failed
    return;
}
ft4222Status = FT4222_SPIMaster_Init(ftHandle2, SPI_IO_SINGLE, CLK_DIV_4,
CLK_IDLE_LOW, CLK_LEADING, 0x03);
if (FT4222_OK != ft4222Status)
{
    // spi master init failed
    return;
}

FT4222_UnInitialize(ftHandle1);
FT4222_UnInitialize(ftHandle2);

FT_Close(ftHandle1);
FT_Close(ftHandle2);
```

3.3.2 SPI Master Set Lines

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

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Switch the FT4222H SPI master to single, dual, or quad mode. This overrides the mode passed to FT4222_SPIMaster_init. This might be needed if a device accepts commands in single mode, but data transfer is to use dual or quad mode.

Parameters:

ftHandle	Handle of the device.
spiMode	SPI mode could be: • SPI_IO_SINGLE • SPI_IO_DUAL • SPI_IO_QUAD

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_IS_NOT_SPI_MODE: Device does not be configured to SPI Master mode.
FT4222_NOT_SUPPORTED: SPI Master only support single/dual/quad mode, others are not allowed.

Prerequisite:

FT4222_SPIMaster_init

3.3.3 SPI Master Set Mode

FT4222_STATUS **FT4222_SPIMaster_SetMode** (FT_HANDLE ftHandle, FT4222_SPICPOL cpol, FT4222_SPICPHA cpha)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Change the clock polarity and phase of FT4222H SPI master.

- Clock polarity: Idle high or idle low.
- Clock phase: Data is sampled on the leading (first) or trailing (second) clock edge.

Below table shows 4 SPI Mode:

SPI mode	Clock polarity (CPOL)	Clock phase (CPHA)
0	0	0
1	0	1
2	1	0
3	1	1

Table 3.1 SPI Mode with CPOL and CPHA

Parameters:

ftHandle	Handle of the device.
cpol	Clock polarity: • CLK_IDLE_LOW • CLK_IDLE_HIGH
cpha	Clock phase: • CLK_LEADING • CLK_TRAILING

Return Value:

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

Prerequisite:

FT4222_SPIMaster_init

3.3.4 SPI Master Set Chip Select

FT4222_STATUS **FT4222_SPIMaster_SetCS** (FT_HANDLE ftHandle, SPI_ChipSelect cs)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Change chip select of FT4222H SPI master. If this function is not called, the default chip select is active low.

Parameters:

ftHandle	Handle of the device.
cs	Chip select: • CS_ACTIVE_LOW • CS_ACTIVE_HIGH

Return Value:

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

Prerequisite:

FT4222_SPIMaster_init

3.3.5 SPI Master Single Read

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

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Under SPI single mode, read data from an SPI slave.

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.
sizeOfRead	Pointer to a variable of type uint16 which receives the number of bytes read from the device.
isEndTransaction	If TRUE the Slave Select pin will be raised at the end of the read.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_INVALID_POINTER: Pointer is a NULL pointer.
FT4222_IS_NOT_SPI_SINGLE_MODE: Device is not in SPI Master Single mode
FT4222_FAILED_TO_WRITE_DEVICE: Write data timeout or failed. FT_SetTimeouts can be called to extend timeout.
FT4222_FAILED_TO_READ_DEVICE: Failed to read data.

Prerequisite:

FT4222_SPIMaster_init

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
   ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
   // open failed
   return;
}
ft4222Status = FT4222_SPIMaster_Init(ftHandle, SPI_IO_SINGLE, CLK_DIV_4, CLK_IDLE_LOW,
                                     CLK_LEADING, 0x01);
if (FT4222_OK != ft4222Status)
{
   // spi master init failed
   return;
}
```

```
uint8 recvData[10];
uint16 sizeTransferred;
ft4222Status = FT4222_SPIMaster_SingleRead(ftHandle, &recvData[0], 10,  &sizeTransferred,
                                           true);
if (FT4222_OK != ft4222Status)
{
   // spi master read failed
   return;
}

FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.3.6 SPI Master Single Write

FT4222_STATUS **FT4222_SPIMaster_SingleWrite**(FT_HANDLE ftHandle, uint8* buffer, uint16 bytesToWrite, uint16* sizeTransferred, BOOL isEndTransaction)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Under SPI single mode, write data to an SPI slave.

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 written to the device.
isEndTransaction	If TRUE the Slave Select pin will be raised at the end of the write.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_INVALID_POINTER: Pointer is a NULL pointer.
FT4222_IS_NOT_SPI_SINGLE_MODE: Device is not in SPI Master Single mode
FT4222_FAILED_TO_WRITE_DEVICE: Write data timeout or failed. FT_SetTimeouts can be called to extend timeout.
FT4222_FAILED_TO_READ_DEVICE: Failed to read data.

Prerequisite:

FT4222_SPIMaster_init

Example:

```
FT_HANDLE ftHandle = NULL;
```

```
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
  // open failed
  return;
}

ft4222Status = FT4222_SPIMaster_Init(ftHandle, SPI_IO_SINGLE, CLK_DIV_4, CLK_IDLE_LOW,
CLK_LEADING, 0x01);
if (FT4222_OK != ft4222Status)
{
  // spi master init failed
  return;
}

uint8 sendData[10];
uint16 sizeTransferred;

for(int idx=0;idx<10;idx++)
  sendData[idx] = idx;

ft4222Status = FT4222_SPIMaster_SingleWrite(ftHandle, &sendData[0], 10,  &sizeTransferred,
true);
if (FT4222_OK != ft4222Status)
{
  // spi master write failed
  return;
}

FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.3.7 SPI Master Single Read and Write

FT4222_STATUS **FT4222_SPIMaster_SingleReadWrite**(FT_HANDLE ftHandle, uint8* readBuffer, uint8* writeBuffer, uint16 sizeToTransfer, uint16* sizeTransferred, BOOL isEndTransaction)
Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

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 -

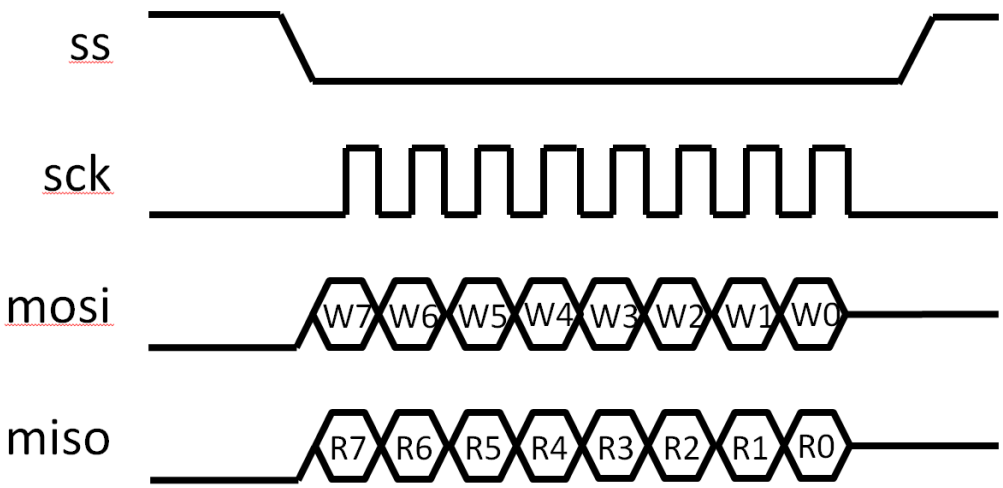


Figure 3.1 SPI full duplex communication

Parameters:

ftHandle	Handle of the device.
readBuffer	Pointer to the buffer that receives data from the device.
writeBuffer	Pointer to the buffer that contains data to be written to the device.
sizeToTransfer	The size of read and write buffer. They must be the same.
sizeTransferred	Pointer to a variable of type uint16 which receives the number of bytes read and written to the device.
isEndTransaction	TRUE to raise the pin of SS at the end of the transaction.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_INVALID_POINTER: Pointer is a NULL pointer.
FT4222_IS_NOT_SPI_SINGLE_MODE: Device is not in SPI Master Single mode
FT4222_FAILED_TO_WRITE_DEVICE: Write data timeout or failed. FT_SetTimeouts can be called to extend timeout.
FT4222_FAILED_TO_READ_DEVICE: Failed to read data.

Prerequisite:

FT4222_SPIMaster_init

Example:

```
// This example is for mxic flash to read out RDSR (read status register)

// bit0 (WIP: write in progress bit). When WIP bit sets to 1, which means the device is

//busy in program/erase/write status register progress

    FT_HANDLE ftHandle = NULL;
    FT_STATUS ftStatus;
    FT4222_STATUS ft4222Status;

    ftStatus = FT_Open(0, &ftHandle);
```

```
    if (FT_OK != ftStatus)
    {
      // open failed
      return;
    }

    ft4222Status = FT4222_SPIMaster_Init(ftHandle, SPI_IO_SINGLE, CLK_DIV_4, CLK_IDLE_LOW,
CLK_LEADING, 0x01);
    if (FT4222_OK != ft4222Status)
    {
      // spi master init failed
      return;
    }

    uint8 sendData[2];
    uint8 readData[2];
    uint16 sizeTransferred;

    // for mxic flash,
    //byte 0: read status command
    //byte 1: status
    sendData[0] = 0x05;  // read status command
    sendData[1] = 0xFF;  // a dummy byte,

    ft4222Status = FT4222_SPIMaster_SingleReadWrite(ftHandle, &readData[0], &sendData[0], 2,
&sizeTransferred, true);
    if((ft4222Status!=FT4222_OK) ||  (sizeTransferred!=2))
    {
       // single read write failed
       return ;
    }

    if ((readData[1] & 0x01) == 0x00)
    {
       // not in writing operation
    }
    else
    {
       // still in writing process
    }

    FT4222_UnInitialize(ftHandle);
    FT_Close(ftHandle);
```

3.3.8 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)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Under SPI dual or quad mode, write data to and read data from an SPI slave.

Figure 3.2 illustrates the dual-SPI protocol supported by the FT4222H SPI master. It is a mixed protocol initiated with a 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.

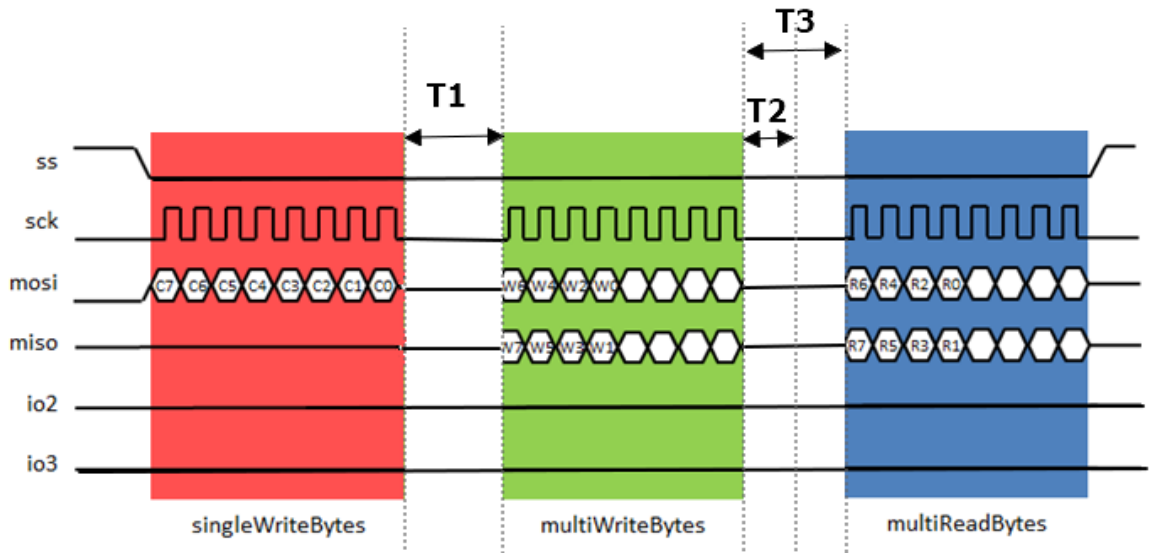


Figure 3.2 Dual SPI communications

Figure 3.3 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 a single write transmission and followed by quad-write and quad-read transmission that use 4 signals in parallel for the data.

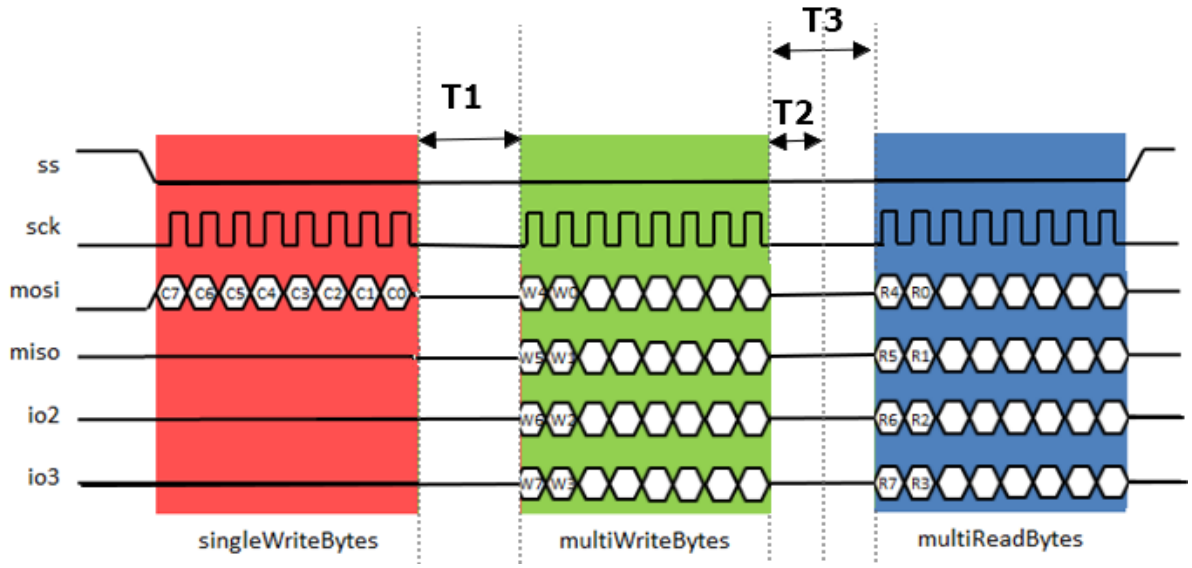


Figure 3.3 Quad SPI communication

Table 3.2 shows the time information for QuadSPI and DualSPI. Avoid I/O conflict from multiWriteBytes state to multiReadBytes state. When SPI finishes multiWriteBytes state, I/O pins (Dual mode:mosi/miso, Quad mode:mosi/miso/io2/io3) will keep the output state for T2 period of time and then change to input state.

This table only takes effect on CLK division >=8. If CLK division is equal to 2 or 4, the time for T2 and T3 is very close and SPI Slave is hard to switch the I/O in such a short time.

Parameter	Min(ns)	Typ(ns)	Max(ns)	Description
T0@48Mhz		20.833		T0 is the period when operating clock=48MHz
T0@60Mhz		16.666		T0 is the period when operating clock=60MHz
T0@80Mhz		12.500		T0 is the period when operating clock=80MHz
Timing for SPI				
T1		288*T0		The timing from singleWrite to multiWrite
T2		90*T0		The time from output pin switch to input pin
T3		160*T0		The timing from multiWrite to multiRead

Table 3.2 SPI Timing for SPI Multi-IO mode

Parameters:

ftHandle	Handle of the device.
readBuffer	Pointer to the buffer that receives the data from the device.
writeBuffer	Pointer to the buffer that contains the data to be written to the device. The data is comprised of both single-write and multi-write parts. It starts with single-write data, whose length is specified by singleWriteBytes, and followed by multi-write data, whose length is specified by multiWriteBytes.
singleWriteBytes	Number of bytes in writeBuffer will be written on single-line. Maximum size is 15.
multiWriteBytes	Number of bytes in writeBuffer will be written on multi-line. Maximum size is 65535.
multiReadBytes	Number of bytes to read on multi-line. Maximum size is 65535.
sizeOfRead	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.

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called
FT4222_INVALID_POINTER: Parameter readBuffer or sizeOfRead is NULL while multiReadBytes is not equal to zero. Parameter writeBuffer is NULL while (singleWriteBytes+ multiWriteBytes) is not equal to zero.
FT4222_FAILED_TO_WRITE_DEVICE: Write data timeout or failed. FT_SetTimeouts can be called to extend timeout.
FT4222_FAILED_TO_READ_DEVICE: Failed to read data.

Prerequisite:

FT4222_SPIMaster_init

Example:

```
// This example it to read 4 IO line mxic flash
    FT_HANDLE ftHandle = NULL;
    FT_STATUS ftStatus;
    FT4222_STATUS ft4222Status;
    uint32 _addr = 0x0;
```

```
ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
  // open failed
  return;
}
ft4222Status = FT4222_SPIMaster_Init(ftHandle, SPI_IO_QUAD, CLK_DIV_4, CLK_IDLE_LOW,
                                     CLK_LEADING, 0x01);
if (FT4222_OK != ft4222Status)
{
  // spi master init failed
  return;
}

uint8 writeData[7];
uint8 readData[16];
uint32 sizeOfRead;

// for mxic flash,
writeData[0] = 0xEB;  // 4 x I/O Read Mode (4READ)
writeData[1] = (unsigned char)((_addr & 0xFF0000) >> 16);  // for addr
writeData[2] = (unsigned char)((_addr & 0x00FF00) >> 8);  // for addr
writeData[3] = (unsigned char)((_addr & 0x0000FF));  // for addr
writeData[4] = 0xFF;  // dummy byte
writeData[5] = 0xFF;  // dummy byte
writeData[6] = 0xFF;  // dummy byte

ft4222Status = FT4222_SPIMaster_MultiReadWrite(ftHandle, readData, &writeData[0], 1, 6, 16,
                                     &sizeOfRead);
if((ft4222Status!=FT4222_OK) ||  (sizeOfRead != 16))
{
   // cannot get correct data
   return;
}

FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.4 SPI Slave Functions

The FT4222H can be initialized as an SPI slave under mode 0 to mode 3. As an SPI slave, the FT4222H only supports the standard single SPI transfer. Please refer to DS_FT4222H for more details. SPI Slave function is not suitable on Android system. Garbage collection is a form of automatic memory management. When garbage collection happens, it does not emit bulk-in packet and RX data may be lost during this period of time.

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.

There are three methods to access FT4222 SPI Slave function.

- SPI_SLAVE_WITH_PROTOCOL
- SPI_SLAVE_NO_ACK
- SPI_SLAVE_NO_PROTOCOL

With all the SPI Slave operational modes listed, the support library will always add a dummy byte of "0x00" as the first byte for every transmission. This is an internal sync byte that is needs to be removed by the SPI Master.

■ SPI_SLAVE_WITH_PROTOCOL

The FT4222H and LibFT4222 design have implemented an SPI slave protocol which must be used to handle the integrity of data transmission. The API "**FT4222_SPISlave_Init**" is used to initialize the slave with this mode.

In this protocol, a master starts an SPI transaction by sending a packet in the format illustrated below. The Sync Word "0x5A" is fixed with this slave mode and user applications do not need to do any operations to add or remove the Sync Word. It is done by the support library.

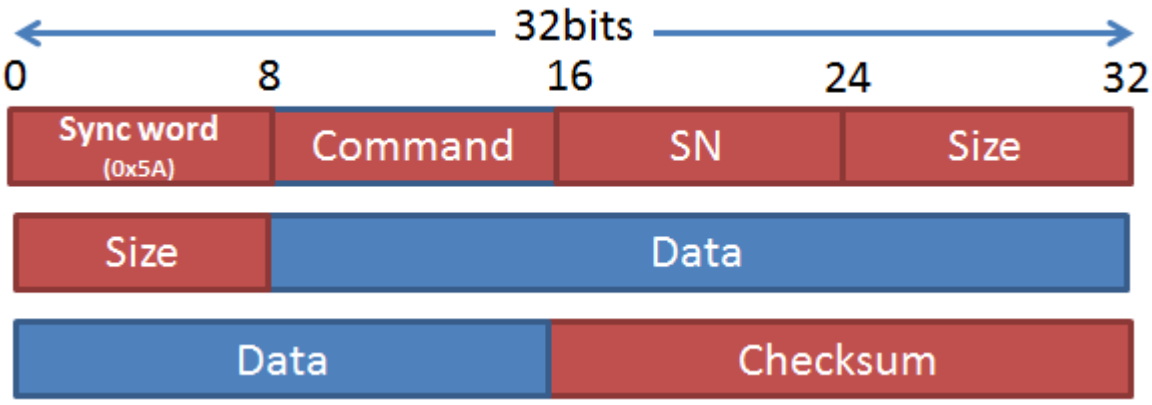


Figure 3.4 SPI Slave Protocol Format

The packet starts with **Sync word**: 0x5A, and followed by a **Command** field:

Command	Value
Master Transfer	0x80
Slave Transfer	0x81
Short master transfer (without checksum)	0x82
Short slave transfer (without checksum)	0x83
ACK	0x84

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.

When SPI Slave receives the Master transfer request, it will check if the format and checksum are correct. If the answer is yes, the support-lib will send the response ACK automatically, grab the data from the packet and send it to application.

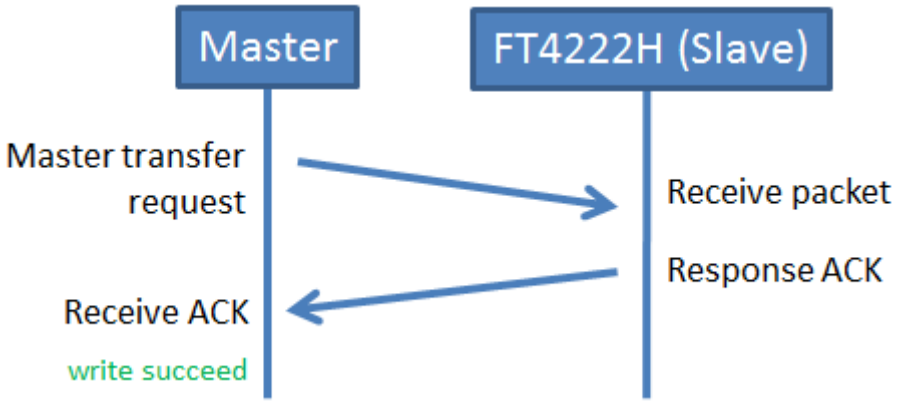


Figure 3.5 SPI Master Transfer Request

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.

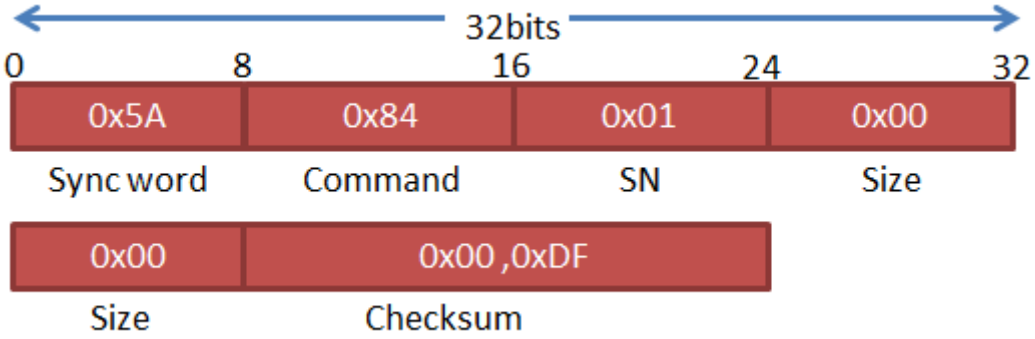


Figure 3.6 An example of the SPI slave responding with ACK

If the SPI master does not receive the ACK response from the slave, it should send its request again.

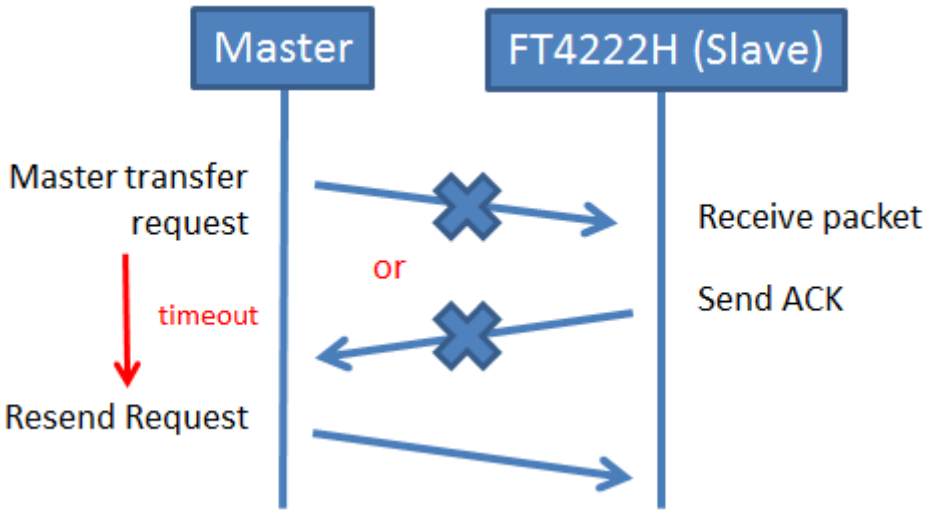


Figure 3.7 An example of when the SPI master doesn't receive ACK

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**.

In this case, it is not necessary to append any header while API FT4222_SPISlave_Write is called.

The encapsulation of header is done by support-lib.

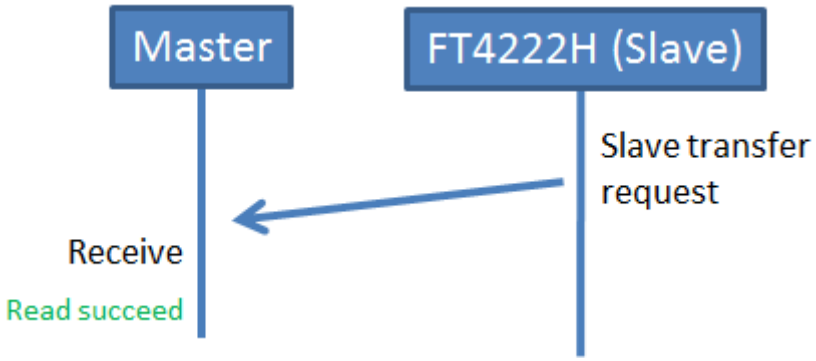


Figure 3.8 Slave sends transfer request

- **SPI_SLAVE_NO_ACK**

This option is to reduce the complication of SPI_SLAVE_WITH PROTOCOL.

It removes the ACK response from the Slave.

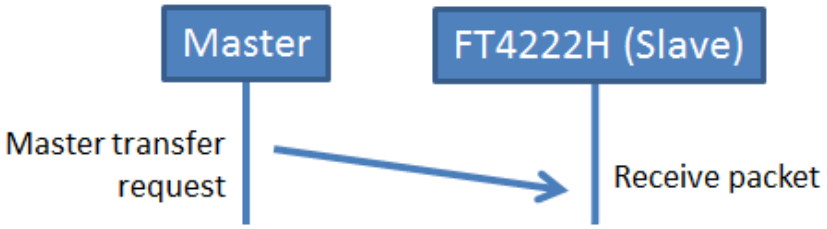


Figure 3.9 SPI Master Transfer Request (NO ACK)

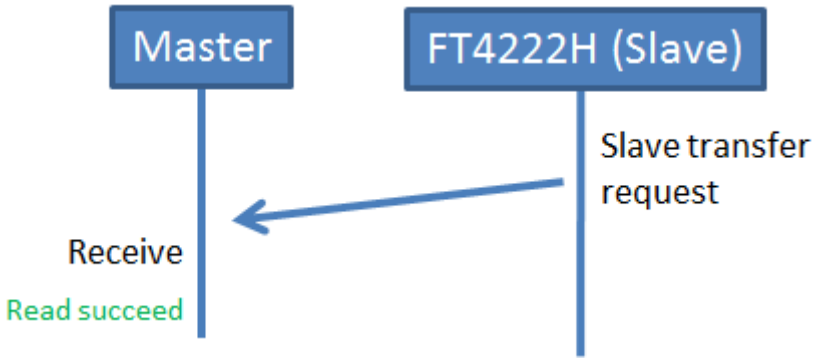


Figure 3.10 Slave sends transfer request (NO ACK)

- **SPI_SLAVE_NO_PROTOCOL**

This option provides no protocol for SPI Slave function, and it is configured and initialized with the API **FT4222_SPISlave_InitEx.**

In this SPI Slave operational mode, the Sync Word "0x5A" is not inserted. And there is no additional process in support-lib.

Users can design own protocol(s) to communicate with a SPI master.

3.4.1 SPI Slave Init

FT4222_STATUS **FT4222_SPISlave_Init**(FT_HANDLE ftHandle)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Initialize the FT4222H as an SPI slave. The default SPI_SlaveProtocol is SPI_SLAVE_WITH_PROTOCOL. The default setting may be replaced with another SPI SLAVE initialization API **FT4222_SPISlave_Init_EX**.

Parameters:

ftHandle	Handle of the device.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.

3.4.2 SPI Slave Init extend function

FT4222_STATUS **FT4222_SPISlave_InitEx**(FT_HANDLE ftHandle , SPI_SlaveProtocol protocolOpt)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Initialize the FT4222H as an SPI slave. It is similar to **FT4222_SPISlave_Init** with parameters to define the SPI Slave Protocol.

Parameters:

ftHandle	Handle of the device.
protocolOpt	SPI SLAVE protocol could be: • SPI_SLAVE_WITH_PROTOCOL With the full SPI SLAVE PROTOCOL supported. Refer to chapter 3.4 • SPI_SLAVE_NO_PROTOCOL Remove SPI SLAVE protocol, users can design their own protocol. • SPI_SLAVE_NO_ACK Retain SPI SLAVE protocol but remove command 'ACK'

Return Value:

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

Error code:

FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.

3.4.3 SPI Slave Set mode function

FT4222_STATUS **FT4222_SPISlave_SetMode**(FT_HANDLE ftHandle ,FT4222_SPICPOL cpol, FT4222_SPICPHA cpha)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Set SPI slave CPOL and CPHA. The Default value of CPOL is CLK_IDLE_LOW, default value of CPHA is CLK_LEADING.

Parameters:

ftHandle	Handle of the device.
cpol	Clock polarity: • CLK_IDLE_LOW • CLK_IDLE_HIGH
cpha	Clock phase: • CLK_LEADING • CLK_TRAILING

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called
FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.
FT4222_IS_NOT_SPI_MODE: The device is not in spi slave mode.

Prerequisite:

FT4222_SPISlave_InitEx or FT4222_SPISlave_Init

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
ftStatus = FT_OpenEx("FT4222 A",FT_OPEN_BY_DESCRIPTION, &ftHandle);
if (FT_OK != ftStatus)
{
  // open failed
  return;
```

```
}

if (FT4222_OK != FT4222_SPISlave_InitEx(ftHandle, SPI_SLAVE_NO_PROTOCOL))
{
   // init spi slave failed
   return;
}

// set spi cpol and cpha to mode 3
if (FT4222_OK != FT4222_SPISlave_SetMode(ftHandle, CLK_IDLE_HIGH, CLK_TRAILING))
{
   // set spi mode  failed
   return;
}
FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.4.4 SPI Slave Get Rx Status

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

Summary:

Get number of bytes in the receive queue.

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Parameters:

ftHandle	Handle of the device.
pRxSize	Pointer to a variable of type uint16 which receives the number of bytes in the receive queue.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called
FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.
FT4222_IS_NOT_SPI_MODE: The device is not in SPI slave mode.
FT4222_INVALID_POINTER: The parameter pRxSize is NULL.

Prerequisite:

FT4222_SPISlave_InitEx or FT4222_SPISlave_Init

Example:

Please refer to the example in FT4222_SPISlave_Read.

3.4.5 SPI Slave Read

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

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Read data from the receive queue of the SPI 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.
sizeOfRead	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.

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called
FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.
FT4222_IS_NOT_SPI_MODE: The device is not in SPI slave mode.
FT4222_INVALID_POINTER: The parameter buffer or sizeOfRead is NULL.
FT4222_INVALID_PARAMETER: Parameter bytesToRead is equal to zero.

Prerequisite:

FT4222_SPISlave_InitEx or FT4222_SPISlave_Init

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
ftStatus = FT_OpenEx("FT4222 A",FT_OPEN_BY_DESCRIPTION, &ftHandle);
if (FT_OK != ftStatus)
{
  // open failed
  return;
}

if (FT4222_OK != FT4222_SPISlave_InitEx(ftHandle, SPI_SLAVE_NO_PROTOCOL))
{
    // init spi slave failed
    return;
}
uint16 sizeTransferred = 0;
```

```
uint16 rxSize;
std::vector<unsigned char> recvBuf;
while(1)
{
    if(FT4222_SPISlave_GetRxStatus(ftHandle, &rxSize) == FT4222_OK)
    {
        if(rxSize>0)
        {
            recvBuf.resize(rxSize);
            if(FT4222_SPISlave_Read(ftHandle,&recvBuf[0], rxSize, &sizeTransferred)==
FT4222_OK)
            {
                // get data
            }
            else
            {
                // get data failed
            }
        }
    }
}

FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.4.6 SPI Slave Write

FT4222_STATUS **FT4222_SPISlave_Write**(FT_HANDLE ftHandle, uint8* buffer, uint16 bytesToWrite, uint16* sizeTransferred)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Write data to the transmit queue of the SPI slave device.

For some reasons, support lib will append a dummy byte (0x00) at the first byte automatically. This additional byte exists at all the three transfer methods.

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 written to the device.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called

FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.
FT4222_IS_NOT_SPI_MODE: The device is not in SPI slave mode.
FT4222_INVALID_POINTER: The parameter buffer or sizeTransferred is NULL.
FT4222_INVALID_PARAMETER: Parameter bytesToWrite is equal to zero.

Prerequisite:

FT4222_SPISlave_InitEx or FT4222_SPISlave_Init

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
ftStatus = FT_OpenEx("FT4222 A",FT_OPEN_BY_DESCRIPTION, &ftHandle);
if (FT_OK != ftStatus)
{
  // open failed
  return;
}

if (FT4222_OK != FT4222_SPISlave_InitEx(ftHandle, SPI_SLAVE_NO_PROTOCOL))
{
    // init spi slave failed
    return;
}
uint16 sizeTransferred = 0;
uint16 rxSize;
std::vector<unsigned char> sendData;
sendData.resize(3);
sendData[0] = 'a';
sendData[1] = 'b';
sendData[2] = 'c';

FT4222_SPISlave_Write(ftHandle, &sendData[0], sendData.size(), &sizeTransferred);
FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.5 SPI General Functions

3.5.1 SPI Reset Transaction

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

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Reset the SPI transaction. It would purge receive and transmit buffers in the device and reset the transaction state. D2XX has a similar function (FT_PURGE) but strongly recommend to use FT4222_SPI_ResetTransaction.

Parameters:

ftHandle	Handle of the device.
spiIdx	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.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_INVALID_PARAMETER: Parameter spiIdx is incorrect. It must depend on mode of chip.

Prerequisite:

FT4222_SPISlave_InitEx or FT4222_SPISlave_Init or FT4222_SPIMaster_Init

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
  ftStatus = FT_OpenEx("FT4222 A",FT_OPEN_BY_DESCRIPTION, &ftHandle);
  if (FT_OK != ftStatus)
  {
    // open failed
    return;
  }
     if (FT4222_OK != FT4222_SPISlave_InitEx(ftHandle, SPI_SLAVE_NO_PROTOCOL))
  {
     // init spi slave failed
     return;
  }
  //clear TX / RX cache
  if (FT4222_OK != FT4222_SPI_ResetTransaction(ftHandle, 0))
  {
     // purge usb tx/rx and SPI FIFO cache
     return;
  }
  // read/write data to a SPI slave
  FT4222_UnInitialize(ftHandle);
  FT_Close(ftHandle);
```

3.5.2 SPI Reset

FT4222_STATUS **FT4222_SPI_Reset** (FT_HANDLE ftHandle)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Reset the SPI master or slave device. If the SPI bus encounters errors or works abnormally, this function will reset the SPI device. It is not necessary to call SPI init function again after calling this reset function. It remains all original setting of SPI.

Parameters:

ftHandle	Handle of the device.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.

Prerequisite:

FT4222_SPISlave_InitEx or FT4222_SPISlave_Init or FT4222_SPIMaster_Init

3.5.3 SPI Set Driving Strength

FT4222_STATUS **FT4222_SPI_SetDrivingStrength**(FT_HANDLE ftHandle,

SPI_DrivingStrength clkStrength,
SPI_DrivingStrength ioStrength,
SPI_DrivingStrength ssoStrength)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

For the FT4222H SPI, set the driving strength of clk, io, and sso pins. The default driving strength of all SPI pins are 4MA. DS_4MA is adopted mostly. Unless there is some hardware wiring requirement for device, set driving strength to 4MA is enough.

Parameters:

ftHandle	Handle of the device.
clkStrength	The driving strength of the clk pin (SPI master only): • DS_4MA • DS_8MA • DS_12MA • DS_16MA
ioStrength	The driving strength of the io pin: • DS_4MA • DS_8MA • DS_12MA • DS_16MA
ssoStrength	The driving strength of the sso pin (SPI master only): • DS_4MA • DS_8MA

	• DS_12MA • DS_16MA

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_IS_NOT_SPI_MODE: The device is not in SPI slave mode.

Prerequisite:

FT4222_SPISlave_InitEx or FT4222_SPISlave_Init or FT4222_SPIMaster_Init

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.

Refer to DS_FT4222H for more details.

3.6.1 I2C Master Init

FT4222_STATUS **FT4222_I2CMaster_Init**(FT_HANDLE ftHandle, uint32 kbps)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

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

Parameters:

ftHandle	Handle of the device.
kbps	The speed of I²C transmission. It ranges from 60 Kbps to 3400 Kbps. By specified speed, the initialization function helps to setup the bus speed with the corresponding mode. This parameter is used to configure the FT4222H to be either SM, FB, FM+ or HS mode.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.
FT4222_I2C_NOT_SUPPORTED_IN_THIS_MODE: I2C is not supported in mode 1 and mode 2.

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
  // open failed
  return;
}
// initial i2c master with 1000K bps
ft4222Status = FT4222_I2CMaster_Init(ftHandle, 1000);
if (FT4222_OK != ft4222Status)
{
   // i2c master init failed
   return;
}
FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.6.2 I²C Master Read

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

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Read data from the specified I2C slave device with START and STOP conditions.

Parameters:

ftHandle	Handle of the device.
slaveAddress	Address of the target I²C slave.
buffer	Pointer to the buffer that receives 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.

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_IS_NOT_I2C_MODE: The device is not in I2C slave mode
FT4222_INVALID_POINTER: Parameter buffer is NULL
FT4222_INVALID_PARAMETER: bytesToRead is equal to zero
FT4222_FAILED_TO_READ_DEVICE: Failed to read data.

Prerequisite:

FT4222_I2CMaster_Init

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;

ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
  // open failed
  return;
}
// initial i2c master with 1000K bps
ft4222Status = FT4222_I2CMaster_Init(ftHandle, 1000);
if (FT4222_OK != ft4222Status)
{
   // i2c master init failed
   return;
}
const uint16 slaveAddr = 0x22;
uint8 slave_data[4];
uint16 sizeTransferred = 0;

// read 4 bytes data from master
ft4222Status = FT4222_I2CMaster_Read(ftHandle, slaveAddr, slave_data, sizeof(slave_data),
&sizeTransferred);
if (FT4222_OK == ft4222Status)
{
   // read data success
}
else
{
   // read data failed
}

FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.6.3 I²C Master Write

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

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Write data to the specified I^2C slave device with START and STOP conditions.

Parameters:

ftHandle	Handle of the device.
slaveAddress	Address of the target I^2C slave.
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 written to the device.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_IS_NOT_I2C_MODE: The device is not in i2c slave mode
FT4222_INVALID_POINTER: Parameter buffer is NULL
FT4222_INVALID_PARAMETER: bytesToWrite is equal to zero
FT4222_FAILED_TO_WRITE_DEVICE: Failed to write data.

Prerequisite:

FT4222_I2CMaster_Init

Example:

```
FT_HANDLE ftHandle = NULL;
 FT_STATUS ftStatus;
 FT4222_STATUS ft4222Status;
 ftStatus = FT_Open(0, &ftHandle);
 if (FT_OK != ftStatus)
 {
   // open failed
   return;
 }
 // initial i2c master with 1000K bps
 ft4222Status = FT4222_I2CMaster_Init(ftHandle, 1000);
 if (FT4222_OK != ft4222Status)
 {
    // i2c master init failed
    return;
 }
 const uint16 slaveAddr = 0x22;
 uint8 master_data[] = {0x1A, 0x2B, 0x3C, 0x4D};
 uint16 sizeTransferred = 0;

 // write 4 bytes data to master
```

```
    ft4222Status = FT4222_I2CMaster_Write(ftHandle, slaveAddr, master_data,
sizeof(master_data), &sizeTransferred);
    if (FT4222_OK == ft4222Status)
    {
        // write data success
    }
    else
    {
        // write data failed
    }

    FT4222_UnInitialize(ftHandle);
    FT_Close(ftHandle);
```

3.6.4 I²C Master Write Extension

FT4222_STATUS **FT4222_I2CMaster_WriteEx**(FT_HANDLE ftHandle, uint16 deviceAddress, uint8 flag, uint8* buffer, uint16 bytesToWrite, uint16* sizeTransferred)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	NO
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

I²C defines basic types of transactions, each of which begins with a START and ends with a STOP:

- Single message where a master writes data to a slave.
- Single message where a master reads data from a slave.
- Combined format, where a master issues at least two reads or writes to one or more slaves.

In a combined transaction, each read or write begins with a START and the slave address. The START conditions after the first are also called *repeated START* bits. Repeated STARTs are not preceded by STOP conditions, which is how slaves know that the next message is part of the same transaction.

This function is supported by the Revision B FT4222H or later.

Parameters:

ftHandle	Handle of the device.
slaveAddress	Address of the target I²C slave.
flag	The I²C condition will be sent with this I²C transaction • START = 0x02 • Repeated_START = 0x03 Repeated_START will not send master code in HS mode • STOP = 0x04 • START_AND_STOP = 0x06
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 written to the device.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_IS_NOT_I2C_MODE: The device is not in I2C slave mode
FT4222_INVALID_POINTER: Parameter buffer is NULL
FT4222_INVALID_PARAMETER: bytesToWrite is equal to zero
FT4222_FAILED_TO_WRITE_DEVICE: Failed to write data.

Prerequisite:

FT4222_I2CMaster_Init

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
  // open failed
  return;
}
// initial i2c master with 1000K bps
ft4222Status = FT4222_I2CMaster_Init(ftHandle, 1000);
if (FT4222_OK != ft4222Status)
{
   // i2c master init failed
   return;
}
const uint16 slaveAddr = 0x22;
uint8 write_req[1];
uint8 recvBuf[16];
uint16 sizeTransferred = 0;
   write_req[0] = 0x00; // addr
   ft4222Status = FT4222_I2CMaster_WriteEx(ftHandle, slaveAddr, START, &write_req[0], 1,
&sizeTransferred);
if (FT4222_OK == ft4222Status)
{
   // write data success
}
else
{
   // write data failed
}
  ft4222Status = FT4222_I2CMaster_ReadEx(ftHandle,slaveAddr, Repeated_START | STOP,
&recvBuf[0], 16, &sizeTransferred);
if (FT4222_OK == ft4222Status)
{
   // read data success
}
else
{
   // read data failed
}
```

```
FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.6.5 I²C Master Read Extension

FT4222_STATUS **FT4222_I2CMaster_ReadEx**(FT_HANDLE ftHandle, uint16 deviceAddress, uint8 flag, uint8* buffer, uint16 bytesToRead, uint16* sizeTransferred).

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	NO
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Read data from the specified I²C slave device with the specified I²C condition.
This function is supported by the Revision B FT4222H or later.

Parameters:

ftHandle	Handle of the device.
slaveAddress	Address of the target I²C slave.
flag	The I²C condition will be sent with this I²C transaction • START = 0x02 • Repeated_START = 0x03 Repeated_START will not send master code in HS mode • STOP = 0x04 • START_AND_STOP = 0x06
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.

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_IS_NOT_I2C_MODE: The device is not in i2c slave mode
FT4222_INVALID_POINTER: Parameter buffer is NULL
FT4222_INVALID_PARAMETER: bytesToRead is equal to zero
FT4222_FAILED_TO_READ_DEVICE: Failed to read data.

Prerequisite:

FT4222_I2CMaster_Init

Example:

Please refer to the example in FT4222_I2CMaster_WriteEx.

I²C combined message support

In a combined message, each read or write begins with a START and the slave address. After the first START, the subsequent starts are referred to as repeated START bits; repeated START bits are not preceded by STOP bits, which indicate to the slave the next transfer is part of the same message.

Start	7 bit slave address	write	ACK	8 bit data	ACK	**SR**	7 bit slave address	Read	ACK	8 bit data	ACK	8 bit data	NACK	STOP

SR = repeated START condition

Here is an example for typical usage of I²C combined message:

```
// Write to I2C slave with START bit

FT4222_I2CMaster_WriteEx(ftHandle, deviceAddress, START, buffer, bufferSize, sizeTransferred);
// Read from I2C slave with Repeated START and STOP bit
// Use Repeated_START flag instead of START to avoid the FT4222H sending master code
// again in HS mode

FT4222_I2CMaster_ReadEx(ftHandle, deviceAddress, Repeated_START | STOP, buffer, bufferSize, sizeTransferred);
```

3.6.6 I²C Master GetStatus

FT4222_STATUS **FT4222_I2CMaster_GetStatus**(FT_HANDLE ftHandle, uint8 *controllerStatus)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Read the status of the I²C master controller. This can be used to poll a slave after I2C transmission is complete.

Parameters:

ftHandle	Handle of the device.
controllerStatus	Address of byte to receive status flags: bit 0: controller busy: all other status bits invalid bit 1: error condition bit 2: slave address was not acknowledged during last operation bit 3: data not acknowledged during last operation bit 4: arbitration lost during last operation bit 5: controller idle bit 6: bus busy The header file provides convenience macros (such as I2CM_BUS_BUSY) to test these bits.

Return Value:

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

Prerequisite:

FT4222_I2CMaster_Init

3.6.7 I²C Master Reset

FT4222_STATUS **FT4222_I2CMaster_Reset**(FT_HANDLE ftHandle)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Reset the I²C master device.
If the I²C bus encounters errors or works abnormally, this function will reset the I²C device. It is not necessary to call I2CMaster_Init again after calling this reset function. This function will maintain the original I2C master setting and clear all cache in the device. D2XX has a similar function (FT_PURGE) but strongly recommend to use FT4222_I2CMaster_Reset.

Parameters:

ftHandle	Handle of the device.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_IS_NOT_I2C_MODE: The device is not in I2C slave mode

Prerequisite:

FT4222_I2CMaster_Init

3.6.8 I²C Master Reset Bus

FT4222_STATUS **FT4222_I2CMaster_ResetBus** (FT_HANDLE ftHandle)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

If the data line (SDA) is pulled LOW by slave device, this API will send nine SCK clocks from master to recover I2C bus. The slave device will release data line (SDA) when it receives the nine clocks from master. If data line cannot be released by this API, HW reset or cycle power is another solution.

Parameters:

ftHandle	Handle of the device.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.

Prerequisite:

FT4222_I2CMaster_Init

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
  // open failed
  return;
}
// initial i2c master with 1000K bps
ft4222Status = FT4222_I2CMaster_Init(ftHandle, 1000);
if (FT4222_OK != ft4222Status)
{
   // i2c master init failed
   return;
}

// now slave is blocked by unknown reason
ft4222Status = FT4222_I2CMaster_ResetBus(ftHandle);

FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);

// reopen the i2c master device
ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
  // open failed
  return;
}
// initial i2c master with 1000K bps
ft4222Status = FT4222_I2CMaster_Init(ftHandle, 1000);
if (FT4222_OK != ft4222Status)
{
```

```
    // i2c master init failed
    return;
  }
  // TODO

  FT4222_UnInitialize(ftHandle);
  FT_Close(ftHandle);
```

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_I2CSlave_Init**(FT_HANDLE ftHandle)

Summary:

Initialize FT4222H as an I²C slave. After FT4222_I2CSlave_Init, I2C slave address is reset to 0x40.

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Parameters:

ftHandle	Handle of the device.

Return Value:

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

Error code:

FT4222_I2C_NOT_SUPPORTED_IN_THIS_MODE: I2C is not supported in mode 1 and mode 2.

Example:

```
  FT_HANDLE ftHandle = NULL;
  FT_STATUS ftStatus;
  FT4222_STATUS ft4222Status;
  ftStatus = FT_Open(0, &ftHandle);
  if (FT_OK != ftStatus)
  {
    // open failed
    return;
```

```
}
// initial i2c slave
ft4222Status = FT4222_I2CSlave_Init(ftHandle);
if (FT4222_OK != ft4222Status)
{
  // i2c slave init failed
  return;
}
FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.7.2 I²C Slave Get Address

FT4222_STATUS **FT4222_I2CSlave_GetAddress**(FT_HANDLE ftHandle, uint8* pAddr)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Get the address of the I2C slave device. The default address is 0x40.

Parameters:

ftHandle	Handle of the device.
pAddr	Pointer to a variable of type uint16 which receives the address of the I²C slave device.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_IS_NOT_I2C_MODE: The device is not in I2C slave mode.

Prerequisite:

FT4222_I2CSlave_Init

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
  // open failed
  return;
}
// initial i2c slave
ft4222Status = FT4222_I2CSlave_Init(ftHandle);
```

```
if (FT4222_OK != ft4222Status)
{
  // i2c slave init failed
  return;
}
uint8 i2cAddr;
// set new i2c slave addr
i2cAddr =0x25;
ft4222Status = FT4222_I2CSlave_SetAddress(ftHandle, i2cAddr);
if (FT4222_OK != ft4222Status)
{
  // i2c slave get addr failed
  return;
}
ft4222Status = FT4222_I2CSlave_GetAddress(ftHandle, &i2cAddr);
if (FT4222_OK != ft4222Status)
{
  // i2c slave get addr failed
  return;
}
   FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.7.3 I²C Slave Set Address

FT4222_STATUS **FT4222_I2CSlave_SetAddress**(FT_HANDLE ftHandle, uint8 addr)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Set the address of the I²C slave device.

Parameters:

ftHandle	Handle of the device.
addr	The 7-bit address of the I²C slave device.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_IS_NOT_I2C_MODE: The device is not in I2C slave mode.

Prerequisite:

FT4222_I2CSlave_Init

Example:

Please refer to the example in FT4222_I2CSlave_GetAddress

3.7.4 I²C Slave Get Rx Status

FT4222_STATUS **FT4222_I2CSlave_GetRxStatus**(FT_HANDLE ftHandle, uint16* pRxSize)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Get number of bytes in the receive queue.

Parameters:

ftHandle	Handle of the device.
pRxSize	Pointer to a variable of type uint16 which receives the number of bytes in the receive queue.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_INVALID_POINTER: Parameter pRxSize is NULL.

Prerequisite:

FT4222_I2CSlave_Init

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
  // open failed
  return;
}
// initial i2c slave
ft4222Status = FT4222_I2CSlave_Init(ftHandle);
if (FT4222_OK != ft4222Status)
{
   // i2c slave init failed
   return;
}
ft4222Status = FT4222_I2CSlave_SetClockStretch(ftHandle, TRUE);
if (FT4222_OK != ft4222Status)
{
   // set clock stretch failed
   return;
```

```
    }
     while(1)
    {
        uint16 rxSize;

        if(FT4222_I2CSlave_GetRxStatus(ftHandle, &rxSize) == FT4222_OK)
        {
            if(rxSize>0)
            {
                uint8 *pRead_data = (uint8 *)malloc(rxSize) ;
                uint16 sizeTransferred;

                if(FT4222_I2CSlave_Read(ftHandle,pRead_data,        rxSize,        &sizeTransferred)==
FT4222_OK)
                {
                    // got slave data
                }
                free(pRead_data);
            }
        }
        else
        {
            printf("I2C slave get status error\n");
        }
    }

    FT4222_UnInitialize(ftHandle);
    FT_Close(ftHandle);
```

3.7.5 I²C Slave Read

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

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Read data from the buffer of the 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.

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_IS_NOT_I2C_MODE: The device is not in I2C slave mode.
FT4222_INVALID_POINTER: Parameter buffer or sizeTransferred is NULL.
FT4222_INVALID_PARAMETER: Parameter bytesToRead is equal to zero.

Prerequisite:

FT4222_I2CSlave_Init

Example:

Please refer to the example in FT4222_I2CSlave_GetRxStatus

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 written to the device.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_IS_NOT_I2C_MODE: The device is not in I2C slave mode.
FT4222_INVALID_POINTER: Parameter buffer or sizeTransferred is NULL.
FT4222_INVALID_PARAMETER: Parameter bytesToWrite is equal to zero.
FT4222_FAILED_TO_WRITE_DEVICE: Write data timeout or failed. FT_SetTimeouts can be called to extend timeout.

Prerequisite:

FT4222_I2CSlave_Init

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
ftStatus = FT_Open(0, &ftHandle);
if (FT_OK != ftStatus)
{
  // open failed
  return;
}
```

```
// initial i2c slave
ft4222Status = FT4222_I2CSlave_Init(ftHandle);
if (FT4222_OK != ft4222Status)
{
  // i2c slave init failed
  return;
}
ft4222Status = FT4222_I2CSlave_SetClockStretch(ftHandle, TRUE);
if (FT4222_OK != ft4222Status)
{
  // set clock stretch failed
  return;
}
uint8 sent_data[] = {0x1A, 0x2B, 0x3C, 0x4D};
uint16 sizeTransferred = 0;

ft4222Status = FT4222_I2CSlave_Write(ftHandle, sent_data, sizeof(sent_data),
&sizeTransferred);
if (FT4222_OK != ft4222Status)
{
   // write data error
}
FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.7.7 I²C Slave Reset

FT4222_STATUS **FT4222_I2CSlave_Reset**(FT_HANDLE ftHandle)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Reset the I²C slave device. This function will maintain the original I2C slave setting and clear all cache in the device. D2XX has a similar function (FT_PURGE) but strongly recommend to use FT4222_I2CSlave_Reset.

Parameters:

ftHandle	Handle of the device.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_IS_NOT_I2C_MODE: The device is not in I2C slave mode.

Prerequisite:

FT4222_I2CSlave_Init

Example:

```
FT_HANDLE ftHandle = NULL;
 FT_STATUS ftStatus;
 FT4222_STATUS ft4222Status;
 ftStatus = FT_Open(0, &ftHandle);
 if (FT_OK != ftStatus)
 {
   // open failed
   return;
 }
 // initial i2c slave
 ft4222Status = FT4222_I2CSlave_Init(ftHandle);
 if (FT4222_OK != ft4222Status)
 {
    // i2c slave init failed
    return;
 }
 ft4222Status = FT4222_I2CSlave_Reset(ftHandle);
 if (FT4222_OK != ft4222Status)
 {
    // reset i2c slave failed
    return;
 }
 FT4222_UnInitialize(ftHandle);
 FT_Close(ftHandle);
```

3.7.8 I²C Slave Clock Stretch

FT4222_STATUS **FT4222_I2CSlave_SetClockStretch**(FT_HANDLE ftHandle, BOOL enable)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	NO
FT4222 Rev B	NO
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Enable or disable Clock Stretch. The default setting of clock stretching is disabled.

Clock stretch is as a flow-control mechanism for slaves. An addressed slave device may hold the clock line (SCL) low after receiving (or sending) a byte, indicating that it is not yet ready to process more data. The master that is communicating with the slave may not finish the transmission of the current bit but must wait until the clock line goes high.

Parameters:

ftHandle	Handle of the device.
Enable	TRUE to enable I2C slave clock stretch FALSE to disable I2C slave clock stretch

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_IS_NOT_I2C_MODE: The device is not in I2C slave mode.

Prerequisite:

FT4222_I2CSlave_Init

Example:

Please refer to the example in FT4222_I2CSlave_Write.

3.7.9 I²C Slave Set Response Word

FT4222_STATUS **FT4222_I2CSlave_SetRespWord** (FT_HANDLE ftHandle, uint8 responseWord)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	NO
FT4222 Rev B	NO
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

This function only takes effect when Clock Stretch is disabled. When data is requested by an I²C master and the device is not ready to respond, the device will respond a default value. Default value is 0xFF. This function can be used to set the response word.

Parameters:

ftHandle	Handle of the device.
responseWord	The response word when the device is not ready to send data to master.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_IS_NOT_I2C_MODE: The device is not in I2C slave mode.

Prerequisite:

FT4222_I2CSlave_Init

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 is multiplexed with interrupt/suspend out/SPI slave select/I2C functions as listed 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 I^2C 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 to 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])

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

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:

ftHandle	Handle of the device.
gpioDir	An array defines the directions of 4 GPIO pins. The GPIO direction will be: • GPIO_OUTPUT • GPIO_INPUT

Return Value:

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

Error code:

FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.
FT4222_GPIO_NOT_SUPPORTED_IN_THIS_MODE: GPIO function is not supported in mode 2 and mode 3.

Example:

Please refer to the example in FT4222_GPIO_Read.

3.8.2 GPIO Read

FT4222_STATUS **FT4222_GPIO_Read**(FT_HANDLE ftHandle, GPIO_Port portNum, BOOL* pValue)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Read the status of a specified GPIO pin or interrupt register.

Parameters:

ftHandle	Handle of the device.
portNum	One of the following GPIO ports: • GPIO_PORT0 • GPIO_PORT1 • GPIO_PORT2 • GPIO_PORT3
pValue	Pointer to a variable of type BOOL which receives the value of the GPIO pin. For GPIO: TRUE means voltage level is high now FALSE mean voltage level is low now For Interrupt: TRUE means trigger condition is invoked FALSE means trigger condition is not invoked Interrupt status is cleared after calling this function.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_GPIO_NOT_SUPPORTED_IN_THIS_MODE: GPIO function is not supported in mode 2 and mode 3.

Example:

```
// this is an example for gpio read

    FT_HANDLE ftHandle = NULL;
    FT_STATUS ftStatus;
    FT4222_STATUS ft4222Status;
    ftStatus = FT_OpenEx("FT4222 B",FT_OPEN_BY_DESCRIPTION, &ftHandle);
    if (FT_OK != ftStatus)
    {
      // open failed
      return;
    }
    GPIO_Dir gpioDir[4];
    gpioDir[0] = GPIO_INPUT;
     gpioDir[1] = GPIO_INPUT;
     gpioDir[2] = GPIO_INPUT;
```

```
 gpioDir[3] = GPIO_INPUT;
FT4222_GPIO_Init(ftHandle, gpioDir);
//disable suspend out , enable gpio 2
FT4222_SetSuspendOut(ftHandle, false);
//disable interrupt , enable gpio 3
FT4222_SetWakeUpInterrupt(ftHandle, false);
BOOL value;
if(FT4222_GPIO_Read(ftHandle, (GPIO_Port)GPIO_PORT3, &value) == FT4222_OK)
{
   // got gpio status
}

FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

// for interrupt read, please refer to the example2 in FT4222_SetInterruptTrigger

3.8.3 GPIO Write

FT4222_STATUS **FT4222_GPIO_Write**(FT_HANDLE ftHandle, GPIO_Port portNum, BOOL bValue)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Write value to the specified GPIO pin.

Parameters:

ftHandle	Handle of the device.
portNum	One of the following GPIO port: • GPIO_PORT0 • GPIO_PORT1 • GPIO_PORT2 • GPIO_PORT3
bValue	The output value.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_GPIO_NOT_SUPPORTED_IN_THIS_MODE: GPIO function is not supported in mode 2 and mode 3.
FT4222_GPIO_WRITE_NOT_SUPPORTED: Direction on this port is not writing direction.

Prerequisite:

FT4222_GPIO_Init

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
ftStatus = FT_OpenEx("FT4222 B",FT_OPEN_BY_DESCRIPTION, &ftHandle);
if (FT_OK != ftStatus)
{

  // open failed
  return;
}
GPIO_Dir gpioDir[4];
gpioDir[0] = GPIO_OUTPUT;
gpioDir[1] = GPIO_OUTPUT;
gpioDir[2] = GPIO_OUTPUT;
gpioDir[3] = GPIO_OUTPUT;
FT4222_GPIO_Init(ftHandle, gpioDir);

//disable suspend out , enable gpio 2
FT4222_SetSuspendOut(ftHandle, false);

//disable interrupt , enable gpio 3
FT4222_SetWakeUpInterrupt(ftHandle, false);

// set gpio0/gpio1/gpio2/gpio3  output level high
FT4222_GPIO_Write(ftHandle, GPIO_PORT0, 1);
FT4222_GPIO_Write(ftHandle, GPIO_PORT1, 1);
FT4222_GPIO_Write(ftHandle, GPIO_PORT2, 1);
FT4222_GPIO_Write(ftHandle, GPIO_PORT3, 1);
FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.8.4 GPIO Set Input Trigger

FT4222_STATUS **FT4222_GPIO_SetInputTrigger**(FT_HANDLE ftHandle, GPIO_Port portNum, GPIO_Trigger trigger)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

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 GPIO0 then 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.

This function can only set gpio trigger conditions. For interrupt trigger conditions, please refer to FT4222_SetInterruptTrigger.

Parameters:

ftHandle	Handle of the device.
portNum	One of the following GPIO port: • GPIO_PORT0 • GPIO_PORT1 • GPIO_PORT2 • GPIO_PORT3
trigger	Combination of the following trigger conditions: • GPIO_TRIGGER_RISING • GPIO_TRIGGER_FALLING • GPIO_TRIGGER_LEVEL_HIGH • GPIO_TRIGGER_LEVEL_LOW

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_GPIO_NOT_SUPPORTED_IN_THIS_MODE: GPIO function is not supported in mode 2 and mode 3.
FT4222_GPIO_INPUT_NOT_SUPPORTED: Direction on this port is not reading direction.

Prerequisite:

FT4222_GPIO_Init

Example:

Please refer the example in FT4222_GPIO_ReadTriggerQueue

3.8.5 GPIO Get Trigger Status

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

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Get the size of trigger event queue.

Parameters:

ftHandle	Handle of the device.
portNum	One of the following GPIO port: • GPIO_PORT0 • GPIO_PORT1 • GPIO_PORT2 • GPIO_PORT3
pQueueSize	Pointer to a variable of type unit16 where the returning value will be stored.

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_GPIO_NOT_SUPPORTED_IN_THIS_MODE: GPIO function is not supported in mode 2 and mode 3
FT4222_INVALID_POINTER: Parameter pQueueSize is NULL.

Prerequisite:

FT4222_GPIO_Init

Example:

Please refer the example in FT4222_GPIO_ReadTriggerQueue.

3.8.6 GPIO Read Trigger Queue

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

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	YES
FT4222 Rev B	YES
FT4222 Rev C	YES
FT4222 Rev D	YES

Summary:

Get events recorded in the trigger event queue. Trigger conditions are set by a call to FT4222_GPIO_SetInputTrigger for a GPIO or FT4222_SetInterruptTrigger for an interrupt. After calling this function, all events will be removed from the event queue.

Parameters:

ftHandle	Handle of the device.
portNum	One of the following GPIO port: • GPIO_PORT0 • GPIO_PORT1 • GPIO_PORT2 • GPIO_PORT3
events	Pointer to the buffer that receives the values of the trigger event queue. The value of events will be: • GPIO_TRIGGER_RISING • GPIO_TRIGGER_FALLING • GPIO_TRIGGER_LEVEL_HIGH • GPIO_TRIGGER_LEVEL_LOW
readSize	Number of bytes to read from trigger event queue.
sizeofRead	Pointer to a variable of type uint16 which receives the number of bytes read from the queue. Queue data is cleared after calling this function For GPIO: The trigger condition needs to be set by the function FT4222_GPIO_SetInputTrigger

	For Interrupt: The trigger condition needs to be set by the function FT4222_SetInterruptTrigger

Return Value:

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

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_GPIO_NOT_SUPPORTED_IN_THIS_MODE: GPIO function is not supported in mode 2 and mode 3.

Prerequisite:

FT4222_GPIO_Init

Example:

```
FT_HANDLE ftHandle = NULL;
FT_STATUS ftStatus;
FT4222_STATUS ft4222Status;
ftStatus = FT_OpenEx("FT4222 B",FT_OPEN_BY_DESCRIPTION, &ftHandle);
if (FT_OK != ftStatus)
{
  // open failed
  return;
}
GPIO_Dir gpioDir[4];
gpioDir[0] = GPIO_INPUT;
gpioDir[1] = GPIO_OUTPUT;
gpioDir[2] = GPIO_OUTPUT;
gpioDir[3] = GPIO_OUTPUT;
FT4222_GPIO_Init(ftHandle, gpioDir);
uint16 queueSize;
FT4222_GPIO_SetInputTrigger(ftHandle,
                GPIO_PORT0,
                (GPIO_Trigger)(GPIO_TRIGGER_LEVEL_HIGH |
                               GPIO_TRIGGER_LEVEL_LOW |
                               GPIO_TRIGGER_RISING |
                               GPIO_TRIGGER_FALLING));
while(1)
{
   if(FT4222_GPIO_GetTriggerStatus(ftHandle, GPIO_PORT0, &queueSize) == FT4222_OK)
   {
      if(queueSize>0)
      {
         uint16 sizeofRead;
         std::vector<GPIO_Trigger> tmpBuf;
         tmpBuf.resize(queueSize);
         if(FT4222_GPIO_ReadTriggerQueue(ftHandle, GPIO_PORT0, &tmpBuf[0], queueSize,
&sizeofRead) == FT4222_OK)
         {
            // tmpBuf  store all trigger status of gpio0
         }
      }
   }
}
// monitor gpio trigger status
FT4222_UnInitialize(ftHandle);
FT_Close(ftHandle);
```

3.8.7 GPIO Set WaveForm Mode

FT4222_STATUS **FT4222_GPIO_SetWaveFormMode**(FT_HANDLE ftHandle, BOOL enable)

Supported Chip:

FT4222 chip version	Supported
FT4222 Rev A	NO
FT4222 Rev B	NO
FT4222 Rev C	NO
FT4222 Rev D	YES

Summary:

Enable or disable WaveForm Mode. When WaveForm mode is enabled, the device will record all GPIO status periodically. The peeking time depends on the system clock. The default setting of WaveForm mode is disabled.

Parameters:

ftHandle	Handle of the device.
enable	TRUE to configure GPIO WaveForm mode. FALSE to switch back to GPIO normal mode. In normal mode, it only records the changing status on GPIO pins.

Return Value:

FT4222_OK if successful, otherwise the return value is a FT error

Error code:

FT4222_DEVICE_NOT_OPENED: The initialization API is not called.
FT4222_DEVICE_NOT_SUPPORTED: This device is not a FT4222 chip.

Prerequisite:
FT4222_GPIO_Init

4 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited (UK)
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 97223-8160
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 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

Branch Office – Shanghai, China

Future Technology Devices International Limited (China)
Room 1103, No. 666 West Huaihai Road,
Shanghai, 200052
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

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.

Appendix A – Enumeration and Structure Definitions

Enumeration

FT4222_STATUS

```
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_INVAILD_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_FUN_NOT_SUPPORT
```

FT4222_ClockRate

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

FT4222_SPIMode

```
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_IDLE_LOW  =0
CLK_IDLE_HIGH =1
```

FT4222_SPICPHA

```
CLK_LEADING  =0
CLK_TRAILING =1
```

SPI_DrivingStrength

```
DS_4MA
DS_8MA
DS_12MA
DS_16MA
```

enum GPIO_Port

```
GPIO_PORT0
GPIO_PORT1
GPIO_PORT2
GPIO_PORT3
```

enum GPIO_Dir

```
GPIO_OUTPUT
GPIO_INPUT
```

enum GPIO_Trigger

```
GPIO_TRIGGER_RISING
GPIO_TRIGGER_FALLING
GPIO_TRIGGER_LEVEL_HIGH
GPIO_TRIGGER_LEVEL_LOW
```

enum GPIO_Output

```
GPIO_OUTPUT_LOW
GPIO_OUTPUT_HIGH
```

enum I2C_MasterFlag

```
START          = 0x02
Repeated_START = 0x03 // Repeated_START will not send master code in HS mode
STOP           = 0x04
START_AND_STOP = 0x06  // START condition followed by SEND and STOP condition
```

Structure Definitions

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

struct SPI_Slave_Header
{
    uint8           syncWord;
    uint8           cmd;
    uint8           sn;
    uint16          size;
};
```

Appendix B – D2XX API support

D2XX supported API

FT_CreateDeviceInfoList	
FT_GetDeviceInfoList	
FT_GetDeviceInfoDetail	
FT_ListDevices	
FT_Open	
FT_OpenEx	
FT_Close	
FT_SetTimeouts	
FT_SetLatencyTimer	
FT_GetLatencyTimer	
FT_GetDeviceInfo	
FT_SetBitMode	
FT_SetUSBParameters	
FT_VendorCmdSet	
FT_VendorCmdGet	
FT_VendorCmdGetEx	
FT_Purge	Chip rev must >= D
FT_ResetDevice	Chip rev must >= D
FT_SetEventNotification	This function can be use on SPI Slave (NO protocol), I2C slave, interrupt
FT_GetStatus	This function can be use on SPI Slave (NO protocol), I2C slave
FT_ResetPort	
FT_Rescan	
FT_Reload	
FT_StopInTask	
FT_RestartInTask	
FT_CyclePort	

Other APIs may conflict with FT4222 support-lib. Please inquiry FAE if you would like to use it.

Appendix C – References

Document References

DS_FT4222H

D2XX Programmers Guide

D2XX Drivers

FT_PROG

Acronyms and Abbreviations

Terms	Description
D2XX	FTDI's proprietary "direct" driver interface via FTD2XX.DLL
GPIO	General-purpose input/output
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interconnect
USB	Universal Serial Bus
USB-IF	USB Implementers Forum

Appendix D – List of Tables and Figures

List of Tables

List of Figures

Appendix E – Revision History

Document Title: AN_329 User Guide for LibFT4222

Document Reference No.: FT_001060

Clearance No.: FTDI#406

Product Page: https://ftdichip.com/products/ft4222h/

Document Feedback: Send Feedback

Revision	Changes	Date
1.0	Initial Release.	16-09-2014
1.1	Two new I²C functions are added to support combined message format. Update to FT4222_I2CMaster_WriteEx and FT4222_I2CMaster_ReadEx.	10-09-2015
1.2	Updated description for SPI master ssoMap.	05-10-2016
1.3	Updated FT4222_I2CSlave_SetClockStretch, FT4222_I2CSlave_SetRespWord.	03-08-2017
1.4	Updated FT4222_SPISlave_SetMode, FT4222_GPIO_SetWaveFormMode, FT4222_SPISlave_RxQuickResponse; error message; sample code; D2xx supported API & prerequisite information.	19-04-2018
1.5	Added SPI multi-master example; Removed I2CMaster_GetStatus.	15-04-2020
1.6	Added the following- FT4222_I2Cmaster_ResetBus; SPI_ChipSelect; FT422_SPIMaster_SetMode. In section 1.1, a table showing DCNF0/DCNF1 Settings control operation modes; Screenshots from Windows Device Manager showing how multiple FT422H Interface appear based on DCNF0/DCNF1 Settings. In Section 3.3.3, SPI Mode form. Updated Section 3.3.4. Removed FT422_SPISlave_RxQuickResponse.	06-05-2021
1.7	In Section 3.3.8 added SPI multi-mode timing.	11-10-2023