The FT260 is a USB device which supports I²C and UART communication through the standard USB HID interface. This application note is a guide for LibFT260, which provides high-level and convenient APIs for FT260 application development.
Table of Contents

1 Introduction ........................................................................................................... 5
  1.1 Overview ......................................................................................................... 5
  1.2 FT260 HID Interfaces and Endpoints .............................................................. 6
    1.2.1 Interfaces .................................................................................................. 6
    1.2.2 Endpoints .................................................................................................. 6
  1.3 Scope .............................................................................................................. 7

2 Wiring .................................................................................................................. 8
  2.1 I²C ..................................................................................................................... 8
  2.2 UART ............................................................................................................... 8

3 Getting Started ................................................................................................... 9

4 Application Programming Interface (API) ....................................................... 11
  4.1 FT260 General Functions ................................................................................. 11
    4.1.1 FT260_CreateDeviceList .......................................................................... 11
    4.1.2 FT260_GetDevicePath ............................................................................. 11
    4.1.3 FT260_Open .............................................................................................. 12
    4.1.4 FT260_OpenByVidPid ............................................................................. 12
    4.1.5 FT260_OpenByDevicePath ...................................................................... 13
    4.1.6 FT260_Close ............................................................................................. 13
    4.1.7 FT260_GetChipVersion ............................................................................ 13
    4.1.8 FT260_GetLibVersion ............................................................................. 14
    4.1.9 FT260_SetClock ....................................................................................... 14
    4.1.10 FT260_SetWakeupInterrupt ................................................................. 14
    4.1.11 FT260_SetInterruptTriggerType .......................................................... 15
    4.1.12 FT260_SelectGpio2Function .................................................................. 15
    4.1.13 FT260_SelectGpioAFunction .................................................................. 16
    4.1.14 FT260_SelectGpioGFunction .................................................................. 16
    4.1.15 FT260_SetSuspendOutPolarity .............................................................. 17
    4.1.16 FT260_SetUartToGPIOPin ..................................................................... 17
    4.1.17 FT260_EnableDcdRiPin ......................................................................... 18
  4.2 I²C Master Functions ..................................................................................... 18
### 4.2 I2C Master Functions

- **4.2.1** FT260_I2CMaster_Init ............................................................... 19
- **4.2.2** FT260_I2CMaster_Reset ............................................................. 19
- **4.2.3** FT260_I2CMaster_Write ............................................................. 20
- **4.2.4** FT260_I2CMaster_Read ............................................................... 20
- **4.2.5** FT260_I2CMaster_GetStatus ....................................................... 21

### 4.3 UART Functions ................................................................. 21

- **4.3.1** FT260_UART_Init ............................................................... 22
- **4.3.2** FT260_UART_Reset ............................................................... 22
- **4.3.3** FT260_UART_SetBaudRate ....................................................... 22
- **4.3.4** FT260_UART_SetFlowControl .................................................. 23
- **4.3.5** FT260_UART_SetDataCharacteristics ....................................... 23
- **4.3.6** FT260_UART_SetBreakOn .......................................................... 24
- **4.3.7** FT260_UART_SetBreakOff ....................................................... 24
- **4.3.8** FT260_UART_GetConfig ........................................................... 24
- **4.3.9** FT260_UART_SetXonXoffChar ................................................... 25
- **4.3.10** FT260_UART_GetQueueStatus .................................................. 25
- **4.3.11** FT260_UART_Write ............................................................... 26
- **4.3.12** FT260_UART_Read ............................................................... 26
- **4.3.13** FT260_UART_GetDcdRiStatus .................................................. 27
- **4.3.14** FT260_UART_EnableRiWakeup .................................................. 27
- **4.3.15** FT260_GetInterruptFlag ....................................................... 27
- **4.3.16** FT260_CleanInterruptFlag .................................................... 28

### 4.4 GPIO Functions ............................................................... 28

- **4.4.1** FT260_GPIO_Set ............................................................... 28
- **4.4.2** FT260_GPIO_Get ............................................................... 29
- **4.4.3** FT260_GPIO_SetDir ............................................................. 29
- **4.4.4** FT260_GPIO_Read ............................................................... 30
- **4.4.5** FT260_GPIO_Write ............................................................... 30
- **4.4.6** FT260_GPIO_Set_OD ............................................................. 30
- **4.4.7** FT260_GPIO_Reset_OD .......................................................... 31

### 5 Contact Information .......................................................... 32

### Appendix A – References ...................................................... 33
Document References .................................................................33
Acronyms and Abbreviations.....................................................33
Appendix B – List of Tables & Figures ................................. 34
List of Tables..................................................................................34
List of Figures ................................................................................34
Appendix C – FT260_STATUS ........................................ 35
Appendix D – Revision History ............................................. 36
1 Introduction

1.1 Overview

The FT260 is a full speed USB device which supports I²C and UART communication through standard USB HID interfaces. The USB HID class is natively supported by most operating systems. A custom driver is not required to be installed for the FT260. By default, the FT260 has two HID interfaces:

- The first HID interface sends and receives data via the I²C connection.
- The second HID interface sends and receives data via the UART connection.
- The HID interface can be configured by the DCNF0 and DCNF1 pins.

The USB HID class exchanges data between a host and a device by reports. There are three types of reports in USB HID:

1. **Feature report**: Configuration data are exchanged between the host and the HID device through a control pipe. The feature report is usually used to turn on/off a device function.
2. **Input report**: Data content that is sent from the HID device to the host.
3. **Output report**: Data content that is sent from the host to the HID device.

The FT260 device receives output reports from the HID application, decodes the requests, and passes the data to the connected I²C or UART device. Data received from the I²C or UART device is sent to the host by input reports.

![The FT260 System Block Diagram](image)

**Figure 1.1 The FT260 System Block Diagram**
1.2 FT260 HID Interfaces and Endpoints

1.2.1 Interfaces

The FT260 interfaces can be configured as:
- I²C and UART
- I²C only
- UART only

The interfaces can be configured by mode pins: DCNF0 and DCNF1.

<table>
<thead>
<tr>
<th>DCNF1</th>
<th>DCNF0</th>
<th>HID Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>The default mode. The FT260 will create two HID interfaces: I²C and UART. This mode is the same as mode (1,1).</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>The FT260 will create a HID interface which sends and receives data via the I²C connection.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>The FT260 will create a HID interface which sends and receives data via the UART connection.</td>
</tr>
</tbody>
</table>
| 1     | 1     | The FT260 will create two HID interfaces:  
- The first HID interface sends and receives data via the I²C connection.  
- The second HID interface sends and receives data via the UART connection. |

Table 1.1 FT260 interface configuration

1.2.2 Endpoints

An interface of the FT260 is composed of the following endpoints:

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control In</td>
<td>Input reports, Feature reports sent to the host with a GET_REPORT request</td>
</tr>
<tr>
<td>Control Out</td>
<td>Output reports, Feature reports received from the host with a SET_REPORT request</td>
</tr>
<tr>
<td>Interrupt In</td>
<td>Input reports</td>
</tr>
<tr>
<td>Interrupt Out</td>
<td>Output reports</td>
</tr>
</tbody>
</table>

Table 1.2 FT260 endpoints
1.3 Scope

The guide is intended for developers who are creating applications, extending FTDI provided applications or implementing FTDI's applications for the FT260.

The support library, LibFT260, hides the detail of communicating by HID protocol and provides simple APIs for developers to create their own applications.

The sample source code provided in this application note is an example and is neither guaranteed nor supported by FTDI.
2 Wiring

2.1 I²C

The FT260 I²C is open-drain architecture. It requires a suitable pull-high resistor on the I²C bus.

![Figure 2.1 The FT260 connects with I²C bus](image)

2.2 UART

The FT260 UART supports 3 flow control modes:

- Software flow control (default)
- Hardware flow control by CTS and RTS
- Hardware flow control by DTR and DSR

Software flow control mode is the default flow control mode of the FT260 and it has the simplest wiring. It only requires connecting TXD, RXD and GND. CTS, RTS, and DTR, DSR are optional for hardware flow control.

![Figure 2.2 The FT260 connects to an UART device](image)
3 Getting Started

This example shows how to open the device with the LibFT260 support library. After opening the device, developers need to initialize the FT260 device as either an I²C master or a UART. Different device types require different configurations. For more details refer to chapter 4.

Example

```c
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include "LibFT260.h"
#define MASK_1 0x0f

void ListAllDevicePaths()
{
    DWORD devNum = 0;
    WCHAR pathBuf[128];

    FT260_CreateDeviceList(&devNum);

    for(int i = 0; i < devNum; i++)
    {
        FT260_GetDevicePath(pathBuf, 128, i);
        wprintf(L"Index:%d
Path:%s
", i, pathBuf);
    }
}

int main(int argc, char const* argv[])
{
    FT260_STATUS ftStatus = FT260_OTHER_ERROR;
    FT260_HANDLE ft260Handle = INVALID_HANDLE_VALUE;
    DWORD devNum = 0;

    ListAllDevicePaths();

    // Open device by index
    ftStatus = FT260_Open(0, &ft260Handle);
    if (FT260_OK != ftStatus) {
        printf("Open device Failed, status: %d\n", ftStatus);
        return 0;
    } else {
        printf("Open device OK\n");
    }

    // Show version information
    DWORD dwChipVersion = 0;
    ftStatus = FT260_GetChipVersion(ft260Handle, &dwChipVersion);
    if (FT260_OK != ftStatus) {
        printf("Get chip version Failed, status: %d\n", ftStatus);
    } else {
        printf("Get chip version OK\n");
        printf("Chip version : %d.%d.%d.%d",
            ((dwChipVersion >> 24) & MASK_1),
            ((dwChipVersion >> 16) & MASK_1),
            ((dwChipVersion >> 8) & MASK_1),
            (dwChipVersion & MASK_1) );
    }
}
}  
// Initialize as an I2C master, and read/write data to an I2C slave  
// FT260_I2CMaster_Init  
// FT260_I2CMaster_Read  
// FT260_I2CMaster_Write  

// Close device  
FT260_Close(ft260Handle);  
return 0;  
}
4 Application Programming Interface (API)

LibFT260 supports I²C, UART and GPIO communication by using high-level APIs. In addition, it provides chip configuration APIs, such as FT260_SetClock. After opening the FT260 device, the FT260 could be initialized by one of the following initial functions:

- FT260_I2CMaster_Init
- FT260_UART_Init

The initialization functions set up the FT260 for the subsequent operations.

Refer to "Appendix C – FT260_STATUS" for the definitions of the error code of following functions.

4.1 FT260 General Functions

The functions listed in this section are configuration functions for the FT260.

4.1.1 FT260_CreateDeviceList

FT260_STATUS  FT260_CreateDeviceList (LPDWORD lpdwNumDevs)

**Summary:**
Create device list and get the number of HID devices.

**Note:** The call creates a list for all HID devices, not only FT260 devices.

**Parameters:**

| lpdwNumDevs | Pointer to a variable for retrieving the number of HID devices. |

**Return Value:**

FT260_OK if successful, otherwise the return value is an error code.

4.1.2 FT260_GetDevicePath

FT260_STATUS  FT260_GetDevicePath(WCHAR* pDevicePath, DWORD bufferLength, DWORD deviceIndex)

**Summary:**
Get device path by index.

**The device path data would be of a format such as shown below:**

\%\?\hid\#vid_0403&pid_6030&mi_00#8&1d5b3f5a&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}

**Parameters:**

| pDevicePath | Pointer to the buffer for getting data. |
| bufferLength | The maximum number of characters to store. Note that the device path is WCHAR. |
| deviceIndex | The index of the device, which is 0 based. |
Return Value:
FT260_OK if successful, otherwise the return value is an error code.

### 4.1.3 FT260_Open

FT260_STATUS FT260_Open(int iDevice, FT260_HANDLE* pFt260Handle)

**Summary:**
Open device by index.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDevice</td>
<td>The index of the device, which is 0 based.</td>
</tr>
<tr>
<td>pFt260Handle</td>
<td>Pointer to a variable of type FT260_HANDLE where the handle will be stored. This handle must be used to access the device.</td>
</tr>
</tbody>
</table>

**Return Value:**
FT260_OK if successful, otherwise the return value is an error code.

### 4.1.4 FT260_OpenByVidPid

FT260_STATUS FT260_OpenByVidPid(WORD vid, WORD pid, DWORD deviceIndex, FT260_HANDLE* pFt260Handle)

**Summary:**
Open device by the given VID, PID and index.

For example, call this function with VID, PID and index:0, 1 and 2 when there are three devices with the same VID and PID.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vid</td>
<td>USB vendor ID.</td>
</tr>
<tr>
<td>pid</td>
<td>USB product ID</td>
</tr>
<tr>
<td>deviceIndex</td>
<td>The index of the device, which is 0 based.</td>
</tr>
<tr>
<td></td>
<td>There might be several devices with the same VID/PID. Use the index to select the device.</td>
</tr>
<tr>
<td>pHandle</td>
<td>Pointer to a variable of type FT260_HANDLE where the handle will be stored. This handle must be used to access the device.</td>
</tr>
</tbody>
</table>

**Return Value:**
FT260_OK if successful, otherwise the return value is an error code.
4.1.5 FT260_OpenByDevicePath

FT260_STATUS FT260_OpenByDevicePath(WCHAR* pDevicePath, FT260_HANDLE* pFt260Handle)

Summary:
Open device by path.

Parameters:

<table>
<thead>
<tr>
<th>pDevicePath</th>
<th>the device path to be opened</th>
</tr>
</thead>
<tbody>
<tr>
<td>pFt260Handle</td>
<td>Pointer to a variable of type FT260_HANDLE where the handle will be stored. This handle must be used to access the device.</td>
</tr>
</tbody>
</table>

Return Value:
FT260_OK if successful, otherwise the return value is an error code.

4.1.6 FT260_Close

FT260_STATUS FT260_Close(FT260_HANDLE ft260Handle)

Summary:
Close the device.

Parameters:

| ft260Handle | Handle of the device. |

Return Value:
FT260_OK if successful, otherwise the return value is an error code.

4.1.7 FT260_GetChipVersion


Summary:
Get the chip version of the FT260 device.

Version 1.0.0.0 is shows as 16777216 in decimal.

Parameters:

<table>
<thead>
<tr>
<th>ft260Handle</th>
<th>Handle of the device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpdwChipVersion</td>
<td>Pointer to a variable for retrieving the chip version.</td>
</tr>
</tbody>
</table>

Return Value:
FT260_OK if successful, otherwise the return value is an error code.
4.1.8 FT260_GetLibVersion

FT260_STATUS FT260_GetLibVersion(LPDWORD lpdwLibVersion)

**Summary:**
Get the library version of the FT260 support library.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpdwLibVersion</td>
<td>Pointer to a variable for retrieving the library version.</td>
</tr>
</tbody>
</table>

**Return Value:**
FT260_OK if successful, otherwise the return value is an error code.

4.1.9 FT260_SetClock

FT260_STATUS FT260_SetClock(FT260_HANDLE ft260Handle, FT260_Clock_Rate clk)

**Summary:**
Set system clock rate. The default clock rate of the FT260 is 48 MHz.

A lower system clock rate will have lower power consumption, and it may also affect maximum transfer rates.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>Clk</td>
<td>System clock rate:</td>
</tr>
<tr>
<td></td>
<td>• FT260_SYS_CLK_12M</td>
</tr>
<tr>
<td></td>
<td>• FT260_SYS_CLK_24M</td>
</tr>
<tr>
<td></td>
<td>• FT260_SYS_CLK_48M</td>
</tr>
</tbody>
</table>

**Return Value:**
FT260_OK if successful, otherwise the return value is an error code.

4.1.10 FT260_SetWakeupInterrupt

FT260_STATUS FT260_SetWakeupInterrupt(FT260_HANDLE ft260Handle, BOOL enable)

**Summary:**
Enable/Disable wakeup interrupt.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>enable</td>
<td>TRUE to enable and switch the pin mode to wakeup/interrupt</td>
</tr>
<tr>
<td></td>
<td>FALSE to disable and switch the pin mode to GPIO3</td>
</tr>
</tbody>
</table>
Return Value:
FT260_OK if successful, otherwise the return value is an error code.

4.1.11  **FT260_SetInterruptTriggerType**

FT260_STATUS  **FT260_SetInterruptTriggerType**(FT260_HANDLE ft260Handle, FT260_Interrupt_Trigger_Type type, FT260_Interrupt_Level_Time_Delay delay)

**Summary:**
Specify edge, level and duration of signals to generate interrupt.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>type</td>
<td>Trigger type:</td>
</tr>
<tr>
<td></td>
<td>• FT260_INTR_RISING_EDGE</td>
</tr>
<tr>
<td></td>
<td>• FT260_INTR_LEVEL_HIGH</td>
</tr>
<tr>
<td></td>
<td>• FT260_INTR_FALLING_EDGE</td>
</tr>
<tr>
<td></td>
<td>• FT260_INTR_LEVEL_LOW</td>
</tr>
<tr>
<td>delay</td>
<td>Specifies the minimum pulse width for level-based interrupts.</td>
</tr>
<tr>
<td></td>
<td>When the voltage at the interrupt pin exceeds the level for the specified duration, the interrupt signal will be generated. This setting only affects trigger types that are level high or level low.</td>
</tr>
<tr>
<td></td>
<td>• FT260_INTR_DELAY_1MS</td>
</tr>
<tr>
<td></td>
<td>• FT260_INTR_DELAY_5MS</td>
</tr>
<tr>
<td></td>
<td>• FT260_INTR_DELAY_30MS</td>
</tr>
</tbody>
</table>

Return Value:
FT260_OK if successful, otherwise the return value is an error code.

4.1.12  **FT260_SelectGpio2Function**

FT260_STATUS  **FT260_SelectGpio2Function**(FT260_HANDLE ft260Handle, FT260_GPIO2_Pin gpio2Function)

**Summary:**
Select the function of GPIO 2.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>gpio2Function</td>
<td>Set the active function of the pin GPIO2:</td>
</tr>
<tr>
<td></td>
<td>• FT260_GPIO2_GPIO</td>
</tr>
<tr>
<td></td>
<td>GPIO 2, General Purpose I/O.</td>
</tr>
<tr>
<td></td>
<td>• FT260_GPIO2_SUSPOUT</td>
</tr>
<tr>
<td></td>
<td>SUSPOUT_N is the default functions to indicate entering the USB suspend state. Active Low. It can be configured as active high.</td>
</tr>
</tbody>
</table>
- FT260_GPIO2_PWREN
  PWREN_N is as the power enable indicator when the FT260 is USB enumerated. Active Low.
- FT260_GPIO2_TX_LED
  TX_LED is the LED driving source when data is transmitted on the UART TX port.

**Return Value:**
FT260_OK if successful, otherwise the return value is an error code.

### 4.1.13 FT260_SelectGpioAFunction

**Summary:**
Select the function of GPIO A.

**Parameters:**
- `ft260Handle` Handle of the device.
- `gpioAFunction` Set the active function of the pin GPIOA:
  - FT260_GPIOA_GPIO
    GPIO A, General Purpose I/O.
  - FT260_GPIOA_TX_ACTIVE
    TX_ACTIVE is the default function to indicate the UART transmitting is active.
  - FT260_GPIOA_TX_LED
    TX_LED is the LED driving source when data is transmitted on the UART TX port.

**Return Value:**
FT260_OK if successful, otherwise the return value is an error code.

### 4.1.14 FT260_SelectGpioGFunction

**Summary:**
Select the function of GPIO G.

**Parameters:**
- `ft260Handle` Handle of the device.
- `gpioGFunction` Set the active function of the pin GPIOG:
• FT260_GPIOG_GPIO
  GPIO G, General Purpose I/O.

• FT260_GPIOG_PWREN
  PWREN_N is the power enable indicator when the FT260 is USB enumerated.
  Active low.

• FT260_GPIOG_RX_LED
  RX_LED is the LED driving source when data is received on the UART RX port.

• FT260_GPIOG_BCD_DET
  BCD_DET is the default function. A battery charger detection indicator output
  when the device is connected to a dedicated battery charger port. Polarity can be
  defined.

**Return Value:**
FT260_OK if successful, otherwise the return value is an error code.

### 4.1.15 FT260_SetSuspendOutPolarity

**FT260_STATUS** `FT260_SetSuspendOutPolarity(FT260_HANDLE ft260Handle, FT260_Suspend_Out_Polarity polarity)`

**Summary:**
Set suspend out polarity.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>polarity</td>
<td>Suspend out level:</td>
</tr>
<tr>
<td></td>
<td>• FT260_SUSPEND_OUT_LEVEL_HIGH</td>
</tr>
<tr>
<td></td>
<td>• FT260_SUSPEND_OUT_LEVEL_LOW</td>
</tr>
</tbody>
</table>

**Return Value:**
FT260_OK if successful, otherwise the return value is an error code.

### 4.1.16 FT260_SetUartToGPIOPin

**FT260_STATUS** `FT260_SetUartToGPIOPin(FT260_HANDLE ft260Handle)`

**Summary:**
Disable UART mode and switch pins to GPIO B, GPIO C, GPIO D, GPIO E, GPIO F and GPIO H.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
</tbody>
</table>
Return Value:
FT260_OK if successful, otherwise the return value is an error code.

4.1.17  **FT260_EnableDcdRiPin**

**FT260_STATUS FT260_EnableDcdRiPin(FT260_HANDLE ft260Handle, BOOL enable)**

**Summary:**
Set UART DCD, RI function and switch pin function.

**Parameters:**

| ft260Handle | Handle of the device. |
| enable      | FALSE to disable UART DCD, UART RI, and switch the pins modes to GPIO4, GPIO5 | TRUE to enable and switch the pins modes to UART DCD, UART RI |

Return Value:
FT260_OK if successful, otherwise the return value is an error code.

4.2 **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).

**I²C transaction**

All I²C transactions begin with a START condition, a slave address, a single bit representing write (0) or read (1), and are terminated by a STOP condition. All of them are always generated by the master.

<table>
<thead>
<tr>
<th>START</th>
<th>7 bit slave address</th>
<th>READ/ WRITE</th>
<th>8 bit data</th>
<th>8 bit data</th>
<th>8 bit data</th>
<th>8 bit data</th>
<th>STOP</th>
</tr>
</thead>
</table>

I²C defines three basic types of message:
- Single message where a master writes data to a slave;
- Single message where a master reads data from a slave;
- Combined messages, where a master issues at least two reads and/or writes to one or more slaves

For more information on the protocol, refer to the I²C specification.

The FT260 provides flexibility to allow users to decide when to send START and STOP conditions. Here are some examples. The following scenarios are supported by the FT260.

**Send data with START_AND_STOP conditions**

<table>
<thead>
<tr>
<th>START</th>
<th>7 bit slave address</th>
<th>READ/ WRITE</th>
<th>8 bit data</th>
<th>8 bit data</th>
<th>8 bit data</th>
<th>8 bit data</th>
<th>STOP</th>
</tr>
</thead>
</table>
Send the first packet with a START condition, and then send remaining data in the other packet with a STOP condition.

Separate data into three packets.

**I²C combined message**

In a combined message, each read or write begins with a START and the slave address. After the first START, these are called repeated START bits; repeated START bits are not preceded by STOP bits, which is how slaves know the next transfer is part of the same message.

**4.2.1 FT260_I2CMaster_Init**

FT260_STATUS FT260_I2CMaster_Init(FT260_HANDLE ft260Handle, uint32 kbps)

**Summary:**

Initialize the FT260 as an I²C master with the requested I²C clock speed.

**Parameters:**

<table>
<thead>
<tr>
<th>ft260Handle</th>
<th>Handle of the device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>kbps</td>
<td>The speed of the I²C clock, whose range is from 100K bps to 4000K bps.</td>
</tr>
</tbody>
</table>

**Return Value:**

FT260_OK if successful, otherwise the return value is an error code.

**4.2.2 T260_I2CMaster_Reset**

FT260_STATUS T260_I2CMaster_Reset(FT260_HANDLE ft260Handle)

**Summary:**

Reset the FT260 I²C Master controller.

**Parameters:**

| ft260Handle | Handle of the device. |

**Return Value:**

FT260_OK if successful, otherwise the return value is an error code.
4.2.3 FT260_I2CMaster_Write

FT260_STATUS FT260_I2CMaster_Write(FT260_HANDLE ft260Handle, uint8 deviceAddress, FT260_I2C_FLAG flag, LPVOID lpBuffer, DWORD dwBytesToWrite, LPDWORD lpdwBytesWritten);

Summary:
Write data to the specified I2C slave device with the given I2C condition.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>deviceAddress</td>
<td>Address of the target I2C slave.</td>
</tr>
<tr>
<td>flag</td>
<td>I2C condition:</td>
</tr>
<tr>
<td></td>
<td>• FT260_I2C_NONE</td>
</tr>
<tr>
<td></td>
<td>• FT260_I2C_START</td>
</tr>
<tr>
<td></td>
<td>• FT260_I2C_REPEATED_START</td>
</tr>
<tr>
<td></td>
<td>• FT260_I2C_STOP</td>
</tr>
<tr>
<td></td>
<td>• FT260_I2C_START_AND_STOP</td>
</tr>
<tr>
<td>lpBuffer</td>
<td>Pointer to the buffer that contains the data to be written to the device.</td>
</tr>
<tr>
<td>dwBytesToRead</td>
<td>Number of bytes to write to the device.</td>
</tr>
<tr>
<td>lpdwBytesReturned</td>
<td>Pointer to a variable of type DWORD which receives the number of bytes</td>
</tr>
<tr>
<td></td>
<td>read and written to the device.</td>
</tr>
</tbody>
</table>

Return Value:
FT260_OK if successful, otherwise the return value is an error code.

4.2.4 FT260_I2CMaster_Read

FT260_STATUS FT260_I2CMaster_Read(FT260_HANDLE ft260Handle, uint8 deviceAddress, FT260_I2C_FLAG flag, LPVOID lpBuffer, DWORD dwBytesToRead, LPDWORD lpdwBytesReturned, DWORD wait_timer);

Summary:
Read data from the specified I2C slave device with the given I2C condition.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>deviceAddress</td>
<td>Address of the target I2C slave device.</td>
</tr>
<tr>
<td>Flag</td>
<td>I2C condition:</td>
</tr>
<tr>
<td></td>
<td>• FT260_I2C_NONE</td>
</tr>
<tr>
<td></td>
<td>• FT260_I2C_START</td>
</tr>
<tr>
<td></td>
<td>• FT260_I2C_REPEATED_START</td>
</tr>
<tr>
<td></td>
<td>• FT260_I2C_STOP</td>
</tr>
<tr>
<td></td>
<td>• FT260_I2C_START_AND_STOP</td>
</tr>
<tr>
<td>lpBuffer</td>
<td>Pointer to the buffer that receives the data from the device.</td>
</tr>
<tr>
<td>dwBytesToRead</td>
<td>Number of bytes to read from the device.</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>lpdwBytesReturned</td>
<td>Pointer to a variable of type DWORD which receives the number of bytes read from the device.</td>
</tr>
<tr>
<td>wait_timer</td>
<td>A timer counter to check I2C read status from the device. If times up, the I2C will return an error message FT260_I2C_READ_FAIL. The default value is 5000 (5 sec).</td>
</tr>
</tbody>
</table>

Return Value:
FT260_OK if successful, otherwise the return value is an error code.

4.2.5 FT260_I2CMaster_GetStatus

FT260_STATUS FT260_I2CMaster_GetStatus(FT260_HANDLE ft260Handle, uint8* status)

Summary:
Read the status of the I2C master controller.

Parameters:

<table>
<thead>
<tr>
<th>ft260Handle</th>
<th>Handle of the device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Point to a variable of type uint8 which saves the status value.</td>
</tr>
</tbody>
</table>

Status:
- 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

Return Value:
FT260_OK if successful, otherwise the return value is an error code.

4.3 UART Functions

UART (Universal Asynchronous Receiver/Transmitter) is a commonly used interface to transfer serial data. Being asynchronous there is no clock signal but the structure of the transmitted data provides for a start and an end to a message. It is also important that both ends of the link decide to operate with the same pulse width defined as the baud rate. The UART of a micro-controller will normally operate at 3V3 or 5V TTL levels. The UART will only connect to one other device in the chain.

The FT260 device can be initialized as a UART. Here is a brief overview of FT260 UART features:
- The UART can support baud rates from 1.2Kbaud to 12MBaud.
- UART data signals: TxD, RxD, RTS, CTS, DSR, DTR, DCD, RI, GND
- Serial Communication Parameters
4.3.1 FT260_UART_Init
FT260_STATUS FT260_UART_Init(FT260_HANDLE ft260Handle);

Summary:
Initialize the FT260 as a UART device.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
</tbody>
</table>

Return Value:
FT260_OK if successful, otherwise the return value is an error code.

4.3.2 FT260_UART_Reset
FT260_STATUS FT260_UART_Reset(FT260_HANDLE ft260Handle)

Summary:
Reset UART controller.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
</tbody>
</table>

Return Value:
FT260_OK if successful, otherwise the return value is an error code.

4.3.3 FT260_UART_SetBaudRate
FT260_STATUS FT260_UART_SetBaudRate(FT260_HANDLE ft260Handle, ULONG baudRate)

Summary:
Set the baud rate for the device.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>baudRate</td>
<td>The speed of UART transmission. It ranges from 1,200 to 12,000,000 bps.</td>
</tr>
</tbody>
</table>

Return Value:
FT260_OK if successful, otherwise the return value is an error code.
4.3.4 FT260_UART_SetFlowControl

FT260_STATUS FT260_UART_SetFlowControl(FT260_HANDLE ft260Handle, FT260_UART_Mode flowControl)

Summary: Set UART flow control for the device.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>flowControl</td>
<td>Flow control:</td>
</tr>
<tr>
<td></td>
<td>• FT260_UART_OFF: Disable UART and switch UART pins to GPIO.</td>
</tr>
<tr>
<td></td>
<td>• FT260_UART_RTS_CTS_MODE</td>
</tr>
<tr>
<td></td>
<td>• FT260_UART_DTR_DSR_MODE</td>
</tr>
<tr>
<td></td>
<td>• FT260_UART_XON_XOFF_MODE</td>
</tr>
<tr>
<td></td>
<td>• FT260_UART_NO_FLOW_CTRL_MODE</td>
</tr>
</tbody>
</table>

Return Value:

FT260_OK if successful, otherwise the return value is an error code.

4.3.5 FT260_UART_SetDataCharacteristics

FT260_STATUS FT260_UART_SetDataCharacteristics(FT260_HANDLE ft260Handle, FT260_Data_Bit dataBits, FT260_Stop_Bit stopBits, FT260_Parity parity);

Summary:

Set UART data characteristics for the device.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>dataBits</td>
<td>Data bits:</td>
</tr>
<tr>
<td></td>
<td>• FT260_DATA_BIT_7</td>
</tr>
<tr>
<td></td>
<td>• FT260_DATA_BIT_8</td>
</tr>
<tr>
<td>stopBits</td>
<td>Stop bits:</td>
</tr>
<tr>
<td></td>
<td>• FT260_STOP_BITS_1</td>
</tr>
<tr>
<td></td>
<td>• FT260_STOP_BITS_2</td>
</tr>
<tr>
<td>parity</td>
<td>Parity:</td>
</tr>
<tr>
<td></td>
<td>• FT260_PARITY_NONE</td>
</tr>
<tr>
<td></td>
<td>• FT260_PARITY_ODD</td>
</tr>
<tr>
<td></td>
<td>• FT260_PARITY_EVEN</td>
</tr>
<tr>
<td></td>
<td>• FT260_PARITY_MARK</td>
</tr>
<tr>
<td></td>
<td>• FT260_PARITY_SPACE</td>
</tr>
</tbody>
</table>

Return Value:
FT260_OK if successful, otherwise the return value is an error code.

### 4.3.6 FT260_UART_SetBreakOn

FT260_STATUS FT260_UART_SetBreakOn(FT260_HANDLE ft260Handle)

**Summary:**
Set the BREAK condition ON for the device.

**Parameters:**
- **ft260Handle**: Handle of the device.

**Return Value:**
FT260_OK if successful, otherwise the return value is an error code.

### 4.3.7 FT260_UART_SetBreakOff

FT260_STATUS FT260_UART_SetBreakOff(FT260_HANDLE ft260Handle)

**Summary:**
Reset the BREAK condition OFF for the device.

**Parameters:**
- **ft260Handle**: Handle of the device.

**Return Value:**
FT260_OK if successful, otherwise the return value is an error code.

### 4.3.8 FT260_UART_GetConfig

FT260_STATUS FT260_UART_GetConfig(FT260_HANDLE ft260Handle, UartConfig* pUartConfig)

**Summary:**
UART get configuration which includes baud rate, data characteristics and break condition.

**Parameters:**
- **ft260Handle**: Handle of the device.
| pUartConfig | Pointer to a variable of type UartConfig where the value will be stored. Type UartConfig is defined as following:
|            | struct UartConfig |
|            | { |
|            |   u8 flow_ctrl;  |
|            |   u32 baud_rate; |
|            |   u8 data_bit;   |
|            |   u8 parity;     |
|            |   u8 stop_bit;   |
|            |   u8 breaking;   |
|            | } |

Please refer to the previous UART setting functions for a description of the fields.

**Return Value:**
FT260_OK if successful, otherwise the return value is an error code.

### 4.3.9 FT260_UART_SetXonXoffChar

**FT260_STATUS** `FT260_UART_SetXonXoffChar(FT260_HANDLE ft260Handle, UCHAR Xon, UCHAR Xoff)`

**Summary:** Set Xon/Xoff characters for software flow control.

**Software flow control (XON_XOFF)**

This setting uses special characters to start and stop data flow. These are termed XON and XOFF (from "transmit on" and "transmit off", respectively). The XON character tells the downstream device to start sending data. The XOFF character tells the downstream device to stop sending data. Usually it is possible to define these characters in an application. Typical default for XON is 0x11 and for XOFF is 0x13.

**Parameters:**

| ft260Handle | Handle of the device. |
| Xon         | Setting character for transmit on. |
| Xoff        | Setting character for transmit off. |

**Return Value:**
FT260_OK if successful, otherwise the return value is an error code.

### 4.3.10 FT260_UART_GetQueueStatus

**FT260_STATUS** `FT260_UART_GetQueueStatus(FT260_HANDLE ft260Handle, LPDWORD lpdwAmountInRxQueue)`

**Summary:**

Gets the number of bytes in the receive queue.
4.3.11  **FT260_UART_Write**

FT260_STATUS  **FT260_UART_Write**(FT260_HANDLE ft260Handle, LPVOID lpBuffer, DWORD dwBufferLength, DWORD dwBytesToWrite, LPDWORD lpdwBytesWritten)

**Summary:**
UART write data to the device.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>lpBuffer</td>
<td>Pointer to the buffer that contains the data to be written.</td>
</tr>
<tr>
<td>dwBufferLength</td>
<td>The length of the buffer.</td>
</tr>
<tr>
<td>dwBytesToWrite</td>
<td>Number of bytes to write.</td>
</tr>
<tr>
<td>lpdwBytesWritten</td>
<td>Pointer to a variable of type DWORD which receives the number of bytes written.</td>
</tr>
</tbody>
</table>

**Return Value:**
FT260_OK if successful, otherwise the return value is an error code.

4.3.12  **FT260_UART_Read**

FT260_STATUS  **FT260_UART_Read**(FT260_HANDLE ft260Handle, LPVOID lpBuffer, DWORD dwBufferLength, DWORD dwBytesToRead, LPDWORD lpdwBytesReturned)

**Summary:**
UART read data from the device.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>lpBuffer</td>
<td>Pointer to the buffer that contains the data to be read.</td>
</tr>
<tr>
<td>dwBufferLength</td>
<td>The length of the buffer.</td>
</tr>
<tr>
<td>dwBytesToWrite</td>
<td>Number of bytes to read.</td>
</tr>
<tr>
<td>lpdwBytesWritten</td>
<td>Pointer to a variable of type DWORD which receives the number of bytes read.</td>
</tr>
</tbody>
</table>

**Return Value:**
FT260_OK if successful, otherwise the return value is an error code.
4.3.13 FT260_UART_GetDcdRiStatus

FT260_STATUS FT260_UART_GetDcdRiStatus(FT260_HANDLE ft260Handle, uint8* value)

Summary:
Get DCD, RI status.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>Value</td>
<td>Pointer to a variable of type uint8 which saves the status value.</td>
</tr>
<tr>
<td></td>
<td>• BIT 0: DCD status</td>
</tr>
<tr>
<td></td>
<td>• BIT 1: RI status</td>
</tr>
</tbody>
</table>

Return Value:
FT260_OK if successful, otherwise the return value is an error code.

4.3.14 FT260_UART_EnableRiWakeup

FT260_STATUS FT260_UART_EnableRiWakeup(FT260_HANDLE ft260Handle, BOOL enable)

Summary:
UART enable RI wakeup.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>Enable</td>
<td>FALSE to disable.</td>
</tr>
<tr>
<td></td>
<td>TRUE to enable.</td>
</tr>
</tbody>
</table>

Return Value:
FT260_OK if successful, otherwise the return value is an error code.

4.3.15 FT260_GetInterruptFlag

FT260_STATUS FT260_GetInterruptFlag(FT260_HANDLE ft260Handle, BOOL* pbFlag);

Summary:
Get interrupt flag.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>pbFlag</td>
<td>Pointer to a variable of type BOOL which saves the flag value.</td>
</tr>
</tbody>
</table>

Return Value:
FT260_OK if successful, otherwise the return value is an error code.
4.3.16 FT260_CleanInterruptFlag

FT260_STATUS FT260_CleanInterruptFlag(FT260_HANDLE ft260Handle, BOOL* pbFlag);

**Summary:**
Clean the interrupt flag.

**Parameters:**

<table>
<thead>
<tr>
<th>ft260Handle</th>
<th>Handle of the device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbFlag</td>
<td>Pointer to a variable of type BOOL which saves the flag value.</td>
</tr>
</tbody>
</table>

**Return Value:**
FT260_OK if successful, otherwise the return value is an error code.

4.4 GPIO Functions

The FT260 contains 14 GPIO pins. Each GPIO pin is multiplexed with other functions as listed below:

- GPIO0 / SCL
- GPIO1 / SDA
- GPIO2 / SUSPEND OUT / TX_LED / PWREN
- GPIO3 / WAKEUP / INTR
- GPIO4 / UART DCD
- GPIO5 / UART RI
- GPIOA / TX_ACTIVE / TX_LED / PWREN
- GPIOB / UART_RTS_N
- GPIOC / UART_RXD
- GPIOD / UART_TXD
- GPIOE / UART_CTS_N
- GPIOF / UART_DTR_N
- GPIOG / BCD_DET / RX_LED
- GPIOH / UART_DST_N

The LibFT260 support library provides several APIs to set the function of these GPIOs and the GPIO example application shows how to use them.

Please refer to [DS_FT260](#) for more information.

4.4.1 FT260_GPIO_Set

FT260_STATUS FT260_GPIO_Set(FT260_HANDLE ft260Handle, FT260_GPIO_Report report)

**Summary:**
Set directions and values for all GPIO pins with the FT260_GPIO_Report parameter.
### Parameters:

<table>
<thead>
<tr>
<th>ft260Handle</th>
<th>Handle of the device.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>report</th>
<th>The setting values which is a variable of type FT260_GPIO_Report. Type FT260_GPIO_Report is defined as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>struct FT260_GPIO_Report</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>WORD value; // bit0<del>5: GPIO0</del>5 values</td>
</tr>
<tr>
<td></td>
<td>WORD dir; // bit0<del>5: GPIO0</del>5 directions</td>
</tr>
<tr>
<td></td>
<td>WORD gpioN_value; // bit0<del>7: GPIOA</del>H values</td>
</tr>
<tr>
<td></td>
<td>WORD gpioN_dir; // bit0<del>7: GPIOA</del>H directions</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

### Return Value:

FT260_OK if successful, otherwise the return value is an error code.

### 4.4.2 FT260_GPIO_Get


**Summary:**

Get directions and values for all GPIO pins with the FT260_GPIO_Report parameter.

### Parameters:

<table>
<thead>
<tr>
<th>ft260Handle</th>
<th>Handle of the device.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>report</th>
<th>Pointer to a variable of type FT260_GPIO_Report where the value will be stored. Type FT260_GPIO_Report is defined as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>struct FT260_GPIO_Report</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>WORD value; // bit0<del>5: GPIO0</del>5 values</td>
</tr>
<tr>
<td></td>
<td>WORD dir; // bit0<del>5: GPIO0</del>5 directions</td>
</tr>
<tr>
<td></td>
<td>WORD gpioN_value; // bit0<del>7: GPIOA</del>H values</td>
</tr>
<tr>
<td></td>
<td>WORD gpioN_dir; // bit0<del>7: GPIOA</del>H directions</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

### Return Value:

FT260_OK if successful, otherwise the return value is an error code.

### 4.4.3 FT260_GPIO_SetDir

**FT260_STATUS FT260_GPIO_SetDir(FT260_HANDLE ft260Handle, WORD pinNum, BYTE dir)**

**Summary:**

Set direction for the specified GPIO pin.
4.4.4 FT260_GPIO_Read

FT260_STATUS FT260_GPIO_Read(FT260_HANDLE ft260Handle, WORD pinNum, BYTE* pValue)

Summary:
Read the value from the specified GPIO pin.

Parameters:
| ft260Handle | Handle of the device. |
| pinNum      | Target GPIO pin number. |
| dir         | 0 for input. |
|             | 1 for output. |

Return Value:
FT260_OK if successful, otherwise the return value is an error code.

4.4.5 FT260_GPIO_Write

FT260_STATUS FT260_GPIO_Write(FT260_HANDLE ft260Handle, WORD pinNum, BYTE value)

Summary:
Write value to the specified GPIO pin.

Parameters:
| ft260Handle | Handle of the device. |
| pinNum      | Target GPIO pin number. |
| value       | The output value. |

Return Value:
FT260_OK if successful, otherwise the return value is an error code.

4.4.6 FT260_GPIO_Set_OD

FT260_STATUS FT260_GPIO_Set_OD(FT260_HANDLE ft260Handle, BYTE pinNum)

Summary:
Function to enable open drain feature. The pins for operation are defined with the parameter `pinNum`.

It is important to note that once this feature is enabled, the `FT260_GPIO_Reset_OD` must be called when the pin is to be configured for other purposes.

### Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
<tr>
<td>pinNum</td>
<td>Target GPIO pins number:</td>
</tr>
<tr>
<td></td>
<td>Ex:</td>
</tr>
<tr>
<td></td>
<td>1. Open drain enable at GPIO 0~5.</td>
</tr>
<tr>
<td></td>
<td>GPIO_0 = 0x01;</td>
</tr>
<tr>
<td></td>
<td>GPIO_1 = 0x01&lt;&lt;1;</td>
</tr>
<tr>
<td></td>
<td>GPIO_2 = 0x01&lt;&lt;2;</td>
</tr>
<tr>
<td></td>
<td>GPIO_3 = 0x01&lt;&lt;3;</td>
</tr>
<tr>
<td></td>
<td>GPIO_4 = 0x01&lt;&lt;4;</td>
</tr>
<tr>
<td></td>
<td>GPIO_5 = 0x01&lt;&lt;5;</td>
</tr>
<tr>
<td></td>
<td>pinNum = (GPIO_0</td>
</tr>
</tbody>
</table>

### Return Value:

FT260_OK if successful, otherwise the return value is an error code.

### 4.4.7 FT260_GPIO_Reset_OD

**FT260_STATUS FT260_GPIO_Reset_OD( FT260_HANDLE handle)**

**Summary:**

To RESET open drain function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft260Handle</td>
<td>Handle of the device.</td>
</tr>
</tbody>
</table>

### Return Value:

FT260_OK if successful, otherwise the return value is an error code.
5 Contact Information

Head Office – Glasgow, UK
Future Technology Devices International Limited
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758
E-mail (Sales) sales1@ftdichip.com
E-mail (Support) support1@ftdichip.com
E-mail (General Enquiries) admin1@ftdichip.com

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.

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

Document References

DS_FT260

Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Terms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>General-purpose input/output</td>
</tr>
<tr>
<td>HID</td>
<td>Humber Interface Device</td>
</tr>
<tr>
<td>I2C</td>
<td>Inter-Integrated Circuit</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver/Transmitter</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>USB-IF</td>
<td>USB Implementers Forum</td>
</tr>
</tbody>
</table>
Appendix B – List of Tables & Figures

List of Tables
Table 1.1 FT260 interface configuration ................................................................. 6
Table 1.2 FT260 endpoints .......................................................................................... 6

List of Figures
Figure 1.1 The FT260 System Block Diagram.......................................................... 5
Figure 2.1 The FT260 connects with I²C bus .......................................................... 8
Figure 2.2 The FT260 connects to an UART device ............................................... 8
Appendix C – FT260_STATUS

FT260_STATUS

FT260_OK = 0
FT260_INVALID_HANDLE = 1
FT260_DEVICE_NOT_FOUND = 2
FT260_DEVICE_NOT_OPENED = 3
FT260_DEVICE_OPEN_FAIL = 4
FT260_DEVICE_CLOSE_FAIL = 5
FT260_INCORRECT_INTERFACE = 6
FT260_INCORRECT_CHIP_MODE = 7
FT260_DEVICE_MANAGER_ERROR = 8
FT260_IO_ERROR = 9
FT260_INVALID_PARAMETER = 10
FT260_NULL_BUFFER_POINTER = 11
FT260_BUFFER_SIZE_ERROR = 12
FT260_UART_SET_FAIL = 13
FT260_RX_NO_DATA = 14
FT260_GPIO_WRONG_DIRECTION = 15
FT260_INVALID_DEVICE = 16
FT260_I2C_READ_FAIL = 17
FT260_OTHER_ERROR = 18
Appendix D – Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Changes</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Initial Release</td>
<td>2016-03-08</td>
</tr>
<tr>
<td>1.1</td>
<td>Updated document for information related to Open drain</td>
<td>2017-10-06</td>
</tr>
<tr>
<td>1.2</td>
<td>Fix editor issue on Code example at Page 9</td>
<td>2018-08-31</td>
</tr>
</tbody>
</table>