



# Application Note

## AN\_373

# Bit-Bang Modes for the FT-X Series

**Version 1.0**

**Issue Date: 2017-05-17**

This document describes the three bit-bang modes (asynchronous, synchronous and CBUS) available for the FT-X series of devices. It also gives examples on how to use each mode, including how to read the contents of a serial EEPROM.

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

**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>

Copyright © Future Technology Devices International Limited

## **Table of Contents**

<b>1</b>	<b>Introduction .....</b>	<b>3</b>
1.1	Scope .....	3
1.2	Software Required.....	3
1.3	Hardware Required .....	3
<b>2</b>	<b>Asynchronous Bit-Bang Mode .....</b>	<b>4</b>
2.1	Operation and Available Pins.....	4
2.2	Asynchronous Bit-Bang Driver Commands .....	5
2.3	Asynchronous Bit-Bang Sample Code .....	6
2.4	Read and Write Strokes in Asynchronous Bit-Bang Mode ....	7
<b>3</b>	<b>Synchronous Bit-Bang Mode .....</b>	<b>8</b>
3.1	Operation and Available Pins.....	8
3.2	Synchronous Bit-Bang Driver Commands .....	9
3.3	Synchronous Bit-Bang Sample Code.....	10
3.4	Read and Write Strokes in Synchronous Bit-Bang Mode....	12
<b>4</b>	<b>CBUS Bit-Bang Mode.....</b>	<b>13</b>
4.1	Operation and Available Pins.....	13
4.2	CBUS Bit-Bang Driver Commands .....	15
4.3	CBUS Bit-Bang Sample Code.....	16
<b>5</b>	<b>Synchronous Bit-Bang Mode Application Example ...</b>	<b>17</b>
5.1	Introduction .....	17
5.2	Hardware Connections .....	17
5.3	EEPROM Read Code Overview.....	17
5.4	Synchronous Bit-Bang Code to Read Microchip 93LC66B...20	
<b>6</b>	<b>CBUS Bit-Bang Mode Application Example .....</b>	<b>26</b>
6.1	Introduction .....	26
6.2	Hardware Connections .....	26
6.3	CBUS Example Code Overview .....	26

---

<b>6.4 CBUS Bit-Bang LED Example Code .....</b>	<b>27</b>
<b>7 Contact Information .....</b>	<b>29</b>
<b>Appendix A – References .....</b>	<b>30</b>
Document References .....	30
Acronyms and Abbreviations.....	30
<b>Appendix B – List of Tables &amp; Figures .....</b>	<b>31</b>
List of Tables.....	31
List of Figures .....	31
<b>Appendix C – Revision History .....</b>	<b>32</b>

## 1 Introduction

Bit-bang mode is a special FTDI device mode that either changes the available I/O lines into a bi-directional data bus, or provides up to four additional 'CBUS' bi-directional pins. There are three types of bit-bang mode for the FT-X series:

**Asynchronous Bit-Bang Mode** where any data will be self-clocked via the data pins. This is similar to the FT2232D, Hi-speed series & FT232R/FT245R asynchronous bit-bang mode.

**Synchronous Bit-Bang Mode**, where data is only read from the pins when the device is written to. This is the same as the FT2232D, Hi-speed series & FT232R/FT245R Synchronous Bit-bang mode.

**CBUS Bit-Bang Mode**, a version of bit-bang previously available on the FT232R CBUS pins allowing for additional pins to bit-bang in addition to the primary interface of the device.

Asynchronous and synchronous bit-bang modes are enabled by D2XX driver commands alone, whilst CBUS bit-bang must also be set up in the device's EEPROM before it can be enabled with a driver command. Each bit-bang mode is mutually exclusive, e.g. it is not possible to use both synchronous and CBUS bit-bang simultaneously. For all three bit-bang modes, the high voltage level is equivalent to VCCIO.

### 1.1 Scope

This document covers the basics of the bit-bang feature in FT-X series devices. It describes each of the three bit-bang modes and shows how to enable these in an application.

### 1.2 Software Required

FTDI D2XX driver which can be downloaded from:  
<http://www.ftdichip.com/Drivers/D2XX.htm>

The FTDI EEPROM programming utility FT\_Prog which can be downloaded from:  
[http://www.ftdichip.com/Support/Utilities.htm#FT\\_PROG](http://www.ftdichip.com/Support/Utilities.htm#FT_PROG)

Microsoft Visual Studio Express 2013 for Windows Desktop which can be downloaded from:  
<https://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx>

### 1.3 Hardware Required

PC with Windows OS installed. Visual Studio Express 2013 supports Windows 7 Service Pack 1, Windows 8, Windows 8.1, Windows Server 2008 R2 SP1, Windows Server 2012 & Windows Server 2012 R2. PC hardware requirements are stated on the Visual Studio Express 2013 download page.

The UM231XE module (which features the FT231X), or other module (or PCB) which uses an FT-X series device (FT200XD, FT201X, FT220X, FT221X, FT230X, FT231X, FT234XD or FT240X).

USB 2.0 cable (standard A to mini-B) to connect to UM231XE.

Three LEDs, three 270 ohm resistors (for CBUS bit-bang LED example).

Microchip 93LC66B (or similar EEPROM).

Small breadboard and wire connectors.

## 2 Asynchronous Bit-Bang Mode

### 2.1 Operation and Available Pins

In asynchronous bit-bang mode, any data written to the device in the normal manner using FT\_Write will be self-clocked onto the data pins which have been configured as outputs. Each pin can be independently set as an input or an output.

The rate at which the data is clocked out is controlled by the baud rate generator. For the pin state to change, new data has to be written and the baud rate clock has to tick. If no new data is written to the device, the output pins will hold the last value written.

When asynchronous bit-bang mode is enabled the I/O signal lines are configured as shown in Table 2.1. Note that after first entering asynchronous bit-bang mode, the default configuration for any of the bit-bang pins, if set to be an output, is output low (0V). (Subsequently writing a '1' to any pin will set that pin high to VCCIO.) If it is desired to drive a high value on any outputs when asynchronous bit-bang mode is entered, then it is necessary to first set these pins to be inputs using FT\_SetBitMode, write a '1' to them using FT\_Write, and then set them to be outputs using FT\_SetBitMode.

Data reading from the pins will begin a minimum of one 12MHz clock period (approx. 83ns) after entering asynchronous bit-bang mode and continue at the rate determined by the baud rate setting (16 x selected baud rate) until either asynchronous bit-bang mode is exited (via FT\_SetBitMode), or the driver's 64KB buffer fills. The writing of data to the pins can continue even if this buffer is full and the read strobe has therefore terminated. Note that the EEPROM UART signal inversion option (available on FT230X, FT231X & FT234XD) has no effect on the pins when the device is in asynchronous bit-bang mode.

FT200XD Pin	FT201X Pin	FT220X Pin	FT221X Pin	FT230X Pin	FT231X Pin	FT234XD Pin	FT240X Pin	Pin Type	Bit-bang Data Bus Bit
SCL	SCL	MIOSI0	MIOSI0	TXD	TXD	TXD	DATA(0)	I/O	D0
SDA	SDA	MIOSI1	MIOSI1	RXD	RXD	RXD	DATA(1)	I/O	D1
		MIOSI2	MIOSI2	RTS#	RTS#	RTS#	DATA(2)	I/O	D2
		MIOSI3	MIOSI3	CTS#	CTS#	CTS#	DATA(3)	I/O	D3
			MIOSI4		DTR#		DATA(4)	I/O	D4
			MIOSI5		DSR#		DATA(5)	I/O	D5
			MIOSI6		DCD#		DATA(6)	I/O	D6
			MIOSI7		RI#		DATA(7)	I/O	D7

**Table 2.1 Asynchronous Bit-Bang Mode I/O Configurations**

The FT200XD and FT201X offer two signal lines; the FT220X, FT230X and FT234XD provide four signal lines; while a total of eight signal lines are available on the FT221X, FT231X and FT240X. Any of the FT-X series device CBUS pins can be configured to bring out the internal read (RD#) or write (WR#) strobes when the device is in asynchronous bit-bang mode (note that FT200XD, FT220X, FT221X and FT234XD have only one CBUS pin, therefore only either RD# or WR# can be

selected). CBUS pins can also be used to provide clock signals by setting the appropriate values in the internal EEPROM.

## 2.2 Asynchronous Bit-Bang Driver Commands

All bit-bang modes need to use the D2XX driver – it is not possible to use these functions via the VCP driver, or to use CBUS bit-bang simultaneously with VCP. **Table 2.2** lists the basic commands used for asynchronous bit-bang mode. Full descriptions of these functions are available in the [D2XX Programmer's Guide](#).

D2XX Function	Description
<b>FT_SetBitMode</b>	Asynchronous bit-bang mode is enabled using the FT_SetBitMode command. A value of 0x01 will enable it and a value of 0x00 will reset the device mode and return the device to the default interface, e.g. UART. (see Note 1).
<b>FT_GetBitMode</b>	FT_GetBitMode returns the instantaneous value of the pins. A single byte will be returned containing the current values of the pins, both those which are inputs and those which are outputs.
<b>FT_SetBaudRate</b>	The rate of data transfer can be controlled by using the FT_SetBaudRate command. The clock for asynchronous bit-bang mode is actually 16 times the set baud rate. A value of 9600 baud would transfer the data at (9600x16) = 153600 bytes per second, or 1 byte every 6.5 μS. The device's maximum baud rate is 3Mbaud, but to allow time for the data to be setup and held around the WR# strobe, the data's baud rate should be a maximum of 1Mbaud (62500 x 16). The minimum data baud rate is 4.8kbaud (300 x 16).
<b>FT_Write</b>	Data can be written to the device in asynchronous bit-bang mode using the FT_Write command. If multiple bytes are written to the device the values on the pins will change at the rate set by FT_SetBaudRate x 16.
<b>FT_Read</b>	FT_Read will return a buffer of values which have been sampled from the pins at the rate set by FT_SetBaudRate x 16. If the read buffers have filled, data will be lost. Each byte returned contains the values of the pins, both those which are inputs and those which are outputs.

**Table 2.2 Asynchronous Bit-Bang Mode D2XX commands**

Note 1: FT\_SetBitMode command can only enable one mode at a time. When the bit mode is reset to normal operation (mode = 0x00), the data pins will return to their state prior to bit-bang mode.

## 2.3 Asynchronous Bit-Bang Sample Code

// This code is provided as an example only and is not supported or guaranteed by FTDI  
 // It is the responsibility of the user to ensure correct operation of their  
 // overall system.

//  
 // Asynchronous bit-bang Mode Sample Code  
 // Write the 7 bytes 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55 to the data pins

```
#include <Windows.h>
#include <stdio.h>
#include "ftd2xx.h"
```

```
FT_STATUS ftStatus;
FT_HANDLE ftHandle;
DWORD w_data_len = 7; // write 7 bytes
DWORD data_written; // number of bytes written
UCHAR Mask = 0xFF; // all output (output (1) and input (0))
UCHAR Mode = 0x01; // 0x01 = asynchronous bit-bang
```

```
void main()
{
ftStatus = FT_Open(0, &ftHandle);
ftStatus |= FT_SetUSBParameters(ftHandle, 4096, 4096); // Set USB transfer sizes
ftStatus |= FT_SetChars(ftHandle, false, 0, false, 0); // Disable event characters
ftStatus |= FT_SetTimeouts(ftHandle, 5000, 5000); // Set read/write timeouts to 5 sec
ftStatus |= FT_SetLatencyTimer(ftHandle, 16); // Latency timer at default 16ms
ftStatus |= FT_SetFlowControl(ftHandle, FT_FLOW_NONE, 0x11, 0x13);
ftStatus |= FT_SetBaudRate(ftHandle, 62500); //bit rate is x16 this value = 1M
if (ftStatus != FT_OK){
printf("ftStatus not ok %d\n", ftStatus); //check for error
printf("Press Return to Exit\n");
getchar();}
else
```

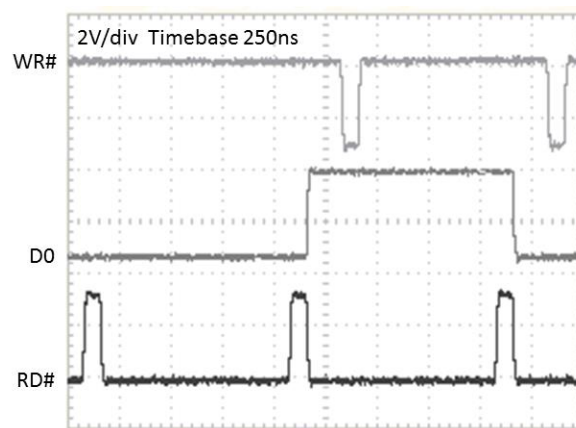
```
{
byte data_out[7]; // data to send 7 bytes of alternate 0x55 and 0xAA
int x;
for (x = 0; x < 7; x = x + 2){ // set even data_out values to 0x55
data_out[x] = 0x55;}
for (x = 1; x < 7; x = x + 2){ // set odd data_out values to 0xAA
data_out[x] = 0xAA;}
// enter asynchronous bit-bang mode
ftStatus = FT_SetBitMode(ftHandle,Mask,Mode);
```

```
// Write 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55 to the pins
// For example 0x55 = 01010101b
// Data pin:      D7      D6      D5      D4      D3      D2      D1      D0
// Value written  0       1       0       1       0       1       0       1
```

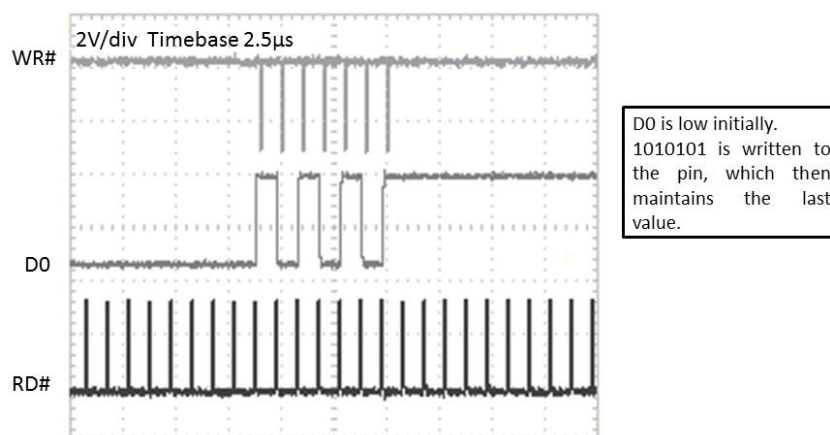
```
ftStatus = FT_Write(ftHandle, data_out, w_data_len, &data_written);
if ((ftStatus == FT_OK)&&(w_data_len == data_written))
printf("FT_Write successful\n");
printf("Press Return to reset bit mode & close device\n");
getchar();
Mode = 0x00; //reset bit mode - pins return to status pre bit-bang mode
ftStatus = FT_SetBitMode(ftHandle, Mask, Mode);
}
ftStatus = FT_Close(ftHandle); //close device
}
```

## 2.4 Read and Write Strokes in Asynchronous Bit-Bang Mode

The read (RD#) and write (WR#) strobes can be output on any available CBUS pin by selecting either BitBang\_RD# or BitBang\_WR# respectively in the EEPROM settings using the utility FT\_Prog. The read strobe is activated a minimum of one 12MHz clock period (approx. 83ns) after the FT\_SetBitMode command to enter asynchronous bit-bang mode is successful. The read buffer will therefore fill to its maximum value of 64KB, unless FT\_Read is used to read the incoming data and prevent buffer overflow. If the buffer is filled, then the read strobe will terminate until the buffer has space available again. Both the read and write strobes have a pulse width of one 12MHz clock period. The read strobe pulses high immediately before the baud rate ticks. The value read from the pins is that sampled during the clock period preceding the read strobe pulse. The write strobe pulses low two clock periods after the baud rate ticks. Figure 2.1 and Figure 2.2 illustrate this behavior.



**Figure 2.1 D0 data output line & read and write strobes in asynchronous bit-bang mode**



**Figure 2.2 D0 data output line & read and write strobes in asynchronous bit-bang mode – 1010101 written to pin.**



## 3 Synchronous Bit-Bang Mode

### 3.1 Operation and Available Pins

Synchronous bit-bang mode synchronizes the read and write operations. When in synchronous bit-bang mode, the data bus pins will be read first before the write (for each data byte) is executed. The value of the pins that is read is therefore always one byte 'behind' the output byte. Unlike asynchronous bit-bang mode, the pins are only read immediately before a write. Therefore to read the response for a byte that has been sent, another byte must be sent. Also, data will only be written to the pins if there is space in the device for data to be read from the pins.

Each pin can be independently set as an input or an output. The rate at which the data is clocked out is controlled by the baud rate generator. For the pin state to change, new data has to be written and the baud rate clock has to tick. If no new data is written to the device, the output pins will hold the last value written.

When synchronous bit-bang mode is enabled the I/O signal lines are configured as shown in Table 3.1. Note that after first entering synchronous bit-bang mode, the default configuration for any of the bit-bang pins, if set to be an output, is output low (0V). (Subsequently writing a '1' to any pin will set that pin to VCCIO.) If it is desired to drive a high value on any outputs when synchronous bit-bang mode is entered, then it is necessary to first set these pins to be inputs using FT\_SetBitMode, write a '1' to them using FT\_Write and then set them to be outputs using FT\_SetBitMode. Note that the EEPROM UART signal inversion option has no effect on the pins when the device is in synchronous bit-bang mode.

FT200XD Pin	FT201X Pin	FT220X Pin	FT221X Pin	FT230X Pin	FT231X Pin	FT234XD Pin	FT240X Pin	Pin Type	Bit-bang Data Bus Bit
SCL	SCL	MIOSI0	MIOSI0	TXD	TXD	TXD	DATA(0)	I/O	D0
SDA	SDA	MIOSI1	MIOSI1	RXD	RXD	RXD	DATA(1)	I/O	D1
		MIOSI2	MIOSI2	RTS#	RTS#	RTS#	DATA(2)	I/O	D2
		MIOSI3	MIOSI3	CTS#	CTS#	CTS#	DATA(3)	I/O	D3
		MIOSI4		DTR#			DATA(4)	I/O	D4
		MIOSI5		DSR#			DATA(5)	I/O	D5
		MIOSI6		DCD#			DATA(6)	I/O	D6
		MIOSI7		RI#			DATA(7)	I/O	D7

**Table 3.1 Synchronous Bit-Bang Mode I/O Configurations**

The FT200XD and FT201X offer two signal lines; the FT220X, FT230X and FT234XD provide four signal lines; whilst a total of eight signal lines are available on the FT221X, FT231X and FT240X. Any of the FT-X series device CBUS pins can be configured to bring out the internal read (RD#) or write (WR#) strobes when the device is in synchronous bit-bang mode (note that FT200XD, FT220X, FT221X and FT234XD have only one CBUS pin, therefore only either RD# or WR# can be selected). CBUS pins can also be used to provide clock signals by setting the appropriate values in the internal EEPROM.

## 3.2 Synchronous Bit-Bang Driver Commands

All bit-bang modes need to use the D2XX driver – it is not possible to use these functions via the VCP driver, or to use CBUS bit-bang simultaneously with VCP. Table 3.2 lists the basic commands used for synchronous bit-bang mode. Full descriptions of these functions are available in the D2XX Programmer's Guide.

D2XX Function	Description
<b>FT_SetBitMode</b>	Synchronous bit-bang mode is enabled using the FT_SetBitMode command. A value of 0x04 will enable it and a value of 0x00 will reset the device mode (see Note 1).
<b>FT_GetBitMode</b>	FT_GetBitMode returns the instantaneous value of the pins. A single byte will be returned containing the current values of the pins, both those which are inputs and those which are outputs.
<b>FT_SetBaudRate</b>	The rate of data transfer can be controlled by using the FT_SetBaudRate command. The clock for asynchronous bit-bang mode is actually 16 times the set baud rate. A value of 9600 baud would transfer the data at (9600x16) = 153600 bytes per second, or 1 byte every 6.5 μS. The device's maximum baud rate is 3Mbaud, but to allow time for the data to be setup and held around the WR# strobe, the data's baud rate should be a maximum of 1Mbaud (62500 x 16). The minimum data baud rate is 4.8kbaud (300 x 16).
<b>FT_Write</b>	Data can be written to the device in Synchronous Bit-bang mode using the FT_Write command. If multiple bytes are written to the device the values on the pins will change at the rate set by FT_SetBaudRate x 16.
<b>FT_Read</b>	FT_Read will return a buffer of values which have been sampled from the pins prior to each byte that is written. If the read buffers have filled, data will be lost. Each byte returned contains the values of the pins, both those which are inputs and those which are outputs.

**Table 3.2 Synchronous Bit-Bang Mode D2XX commands**

Note 1: FT\_SetBitMode command can only enable one mode at a time. When the bit mode is reset to normal operation (mode = 0x00), the data pins will return to their state prior to bit-bang mode.

### 3.3 Synchronous Bit-Bang Sample Code

```
// This code is provided as an example only and is not supported or guaranteed by FTDI
// It is the responsibility of the user to ensure correct operation of their
// overall system.
//
// Synchronous Bit-bang Mode Sample Code
// Input pins can be connected to output pins to perform a loopback test.
// For example, if using an FT221X, FT231X or FT240X device, for Mask = 0x0F,
// connect D0 to D4, D1 to D5, D2 to D6 & D3 to D7

#include <Windows.h>
#include <stdio.h>
#include "ftd2xx.h"

FT_STATUS ftStatus;
FT_HANDLE ftHandle;
DWORD w_data_len = 3; //write 3 bytes
DWORD data_written; // number of bytes written
UCHAR Mask = 0x0F; //Set D7-D4 input, D3-D0 output [input (0) and output (1)]
UCHAR Mode = 0x04; //0x04 = synchronous bit-bang
DWORD RxBytes; //number of bytes to be read from the device
DWORD BytesReceived; //number of bytes read from the device
byte RxBuffer[8]; //buffer to receive data from FT-X device
byte data_out[8]; //buffer for data to be sent to FT-X device
unsigned int i;

void main()
{
ftStatus = FT_Open(0, &ftHandle);
ftStatus |= FT_SetUSBParameters(ftHandle, 4096, 4096); // Set USB transfer sizes
ftStatus |= FT_SetChars(ftHandle, false, 0, false, 0); // Disable event characters
ftStatus |= FT_SetTimeouts(ftHandle, 5000, 5000); // Set read/write timeouts to 5 sec
ftStatus |= FT_SetLatencyTimer(ftHandle, 16); // Latency timer at default 16ms
ftStatus |= FT_SetFlowControl(ftHandle, FT_FLOW_NONE, 0x11, 0x13);
ftStatus |= FT_SetBaudRate(ftHandle, 62500); //bit rate is x16 this value = 1M
if (ftStatus != FT_OK)
{
printf("ftStatus not ok %d\n", ftStatus); //check for error
printf("Press Return to Exit\n");
getchar();
}
else
{
//upper nibble is input pins, value (0 here) is 'don't care' for writing
//lower nibble is output pins, D0 is sent 1 0 1, D1 is sent 0 1 1
// D2 is sent 0 0 0 & D3 is sent 0 0 0
data_out[0] = 0x01; // write 0x01 to pins: 0000 0001
data_out[1] = 0x02; // write 0x02 to pins: 0000 0010
data_out[2] = 0x03; // write 0x03 to pins: 0000 0011

//enter synchronous bit-bang mode
ftStatus = FT_SetBitMode(ftHandle, Mask, Mode);
//write data to pins
ftStatus = FT_Write(ftHandle, data_out, w_data_len, &data_written);

do{
//Sleep(1000);
FT_GetQueueStatus(ftHandle, &RxBytes);
//Sleep(10); //delay

```

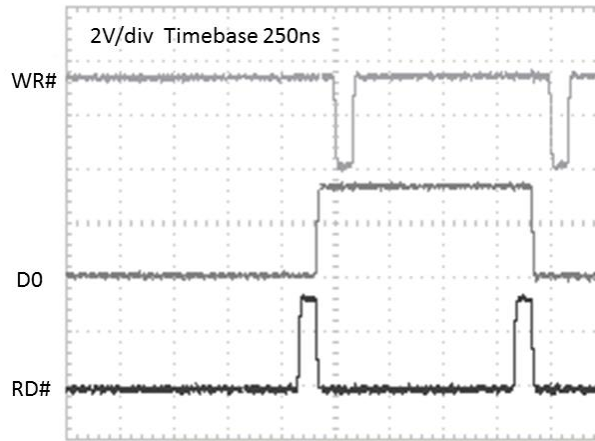
```
} while (RxBytes != (w_data_len)); //check for all bytes to be returned

//once bytes are received, use FT_Read to place in RxBuffer
if (RxBytes > 0) {
    ftStatus = FT_Read(ftHandle, RxBuffer, RxBytes, &BytesReceived);
    printf("%d bytes received\n \n", BytesReceived);
    for (i = 0;i < BytesReceived;i++) printf("RxBuffer %d: %x\n", i,RxBuffer[i]);
    //
    //After enumeration, if device in loopback mode, the RxBuffer values will be
    // RxBuffer 0 = 0x00 (read prior to write of first byte)
    // RxBuffer 1 = 0x11 (read after 1st byte written)
    // RxBuffer 2 = 0x22 (read after 2nd byte written)
}
printf("Press Return to reset bit mode & close device\n");
getchar();
Mode = 0x00; //reset bit mode - pins return to values before setbitmode
ftStatus = FT_SetBitMode(ftHandle, Mask, Mode);
}

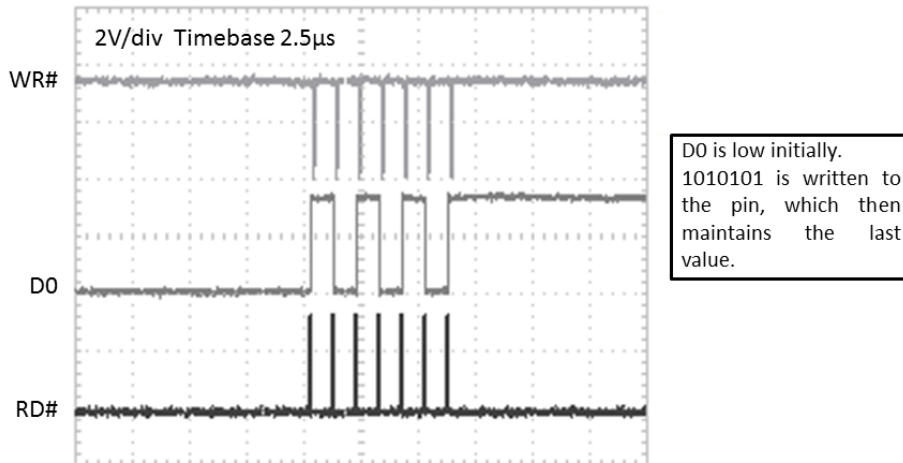
ftStatus = FT_Close(ftHandle); //close device
}
```

### 3.4 Read and Write Strobes in Synchronous Bit-Bang Mode

The read (RD#) and write (WR#) strobes can be output on any available CBUS pin by selecting either BitBang\_RD# or BitBang\_WR# respectively in the EEPROM settings using the utility FT\_Prog. The read strobe only goes high two clock periods (approx. 166ns) before the write strobe goes low. Both the read and write strobes have a pulse width of one 12MHz clock period (approx. 83ns). The read strobe pulses high immediately before the baud rate ticks. The value read from the pins is that sampled during the clock period preceding the read strobe pulse. The write strobe pulses low (goes active) one 12MHz clock period after the baud rate ticks. Figure 3.1 and Figure 3.2 illustrate this behavior.



**Figure 3.1 D0 data output line & read and write strobes in synchronous bit-bang mode**



**Figure 3.2 D0 data output line & read and write strobes in synchronous bit-bang mode – 1010101 written to pin.**

## 4 CBUS Bit-Bang Mode

### 4.1 Operation and Available Pins

Similar to the FT232R, the FT-X series devices (except the FT240X) also support CBUS bit-bang mode. To use this mode the CBUS pins must first be configured for 'GPIO' in the device's EEPROM, for example by using FT\_Prog. The D2XX commands FT\_SetBitMode and FT\_GetBitMode are then used to 'send' and 'receive' data respectively.

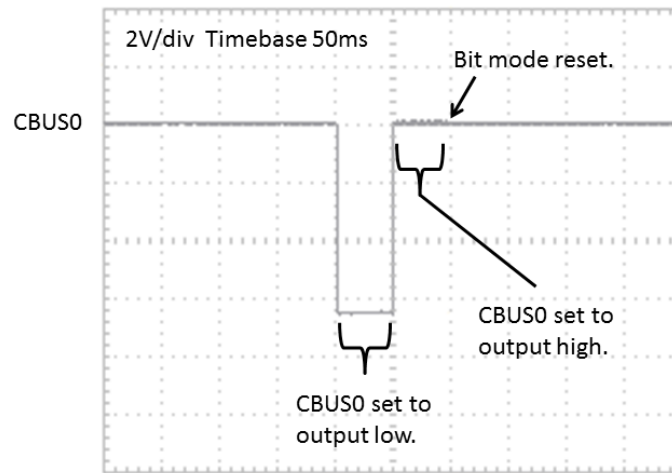
Each pin can be independently set as an input or an output. If no new data is written to the device, the output pins will hold the last value written. Note that after setting any CBUS pin to GPIO mode; it will default to be an input pulled to VCCIO via internal resistors. When the bit mode is reset to normal operation (mode = 0x00), the CBUS GPIO pins will continue to hold their previous configuration and value. (To avoid having any pins continuing to drive out after a bit mode reset, set them to be inputs prior to performing the reset.)

The baud rate has no effect on the CBUS bit-bang pins. The FT\_SetBitMode and FT\_GetBitMode functions are implemented as USB control requests and will be scheduled by the USB host controller accordingly. Since 10% of the bandwidth is reserved for control requests in each full-speed USB frame, it is likely that an FT\_SetBitMode command will travel on the next frame, but this is not guaranteed if there are competing control requests from other applications. CBUS bit-bang is therefore best suited to applications which require intermittent access, e.g. to control an LED, switch operation, to signal an event, etc.

The sample code in Section 0 toggles CBUS0 low, then high before setting all CBUS pins to input and resetting the bit mode. Figure 4.1 shows the resulting 'scope trace for CBUS0. Standard reads and writes to the UART pins using the D2XX driver can also be combined with CBUS bit-bang in an application, but note that since the reads/writes use bulk transfer, in contrast to the control transfer for the SetBitMode command, there can be differences in synchronization. The FT200XD, FT220X, FT221X and FT234XD offer one CBUS pin for bit-bang, and the FT201X, FT230X and FT231X provide up to four CBUS pins for bit-bang – see Table 4.1.

FT200XD Pin	FT201X Pin	FT220X Pin	FT221X Pin	FT230X Pin	FT231X Pin	FT234XD Pin	Pin Type
CBUS0	CBUS0	CBUS3	CBUS3	CBUS0	CBUS0	CBUS0	I/O
	CBUS1			CBUS1	CBUS1		I/O
	CBUS2			CBUS2	CBUS2		I/O
	CBUS3			CBUS3	CBUS3		I/O

**Table 4.1 CBUS Bit-Bang Mode I/O Configurations**



**Figure 4.1 CBUS0 set low then high in CBUS bit-bang mode**

## 4.2 CBUS Bit-Bang Driver Commands

All bit-bang modes need to use the D2XX driver – it is not possible to use these functions via the VCP driver, or to use CBUS bit-bang simultaneously with VCP. Table 4.2 lists the basic commands used for CBUS bit-bang mode.

<b>D2XX Function</b>	<b>Description</b>
<b>FT_SetBitMode</b>	CBUS bit-bang mode is enabled using the FT_SetBitMode command. A value of 0x20 will enable it and a value of 0x00 will reset the device mode. (see note 1)
<b>FT_GetBitMode</b>	FT_GetBitMode returns the instantaneous value of the pins. A single byte will be returned containing the current values of the pins, both those which are inputs and those which are outputs.

**Table 4.2 CBUS Bit-Bang Mode D2XX commands**



### 4.3 CBUS Bit-Bang Sample Code

```
// This code is provided as an example only and is not supported or guaranteed by FTDI
// It is the responsibility of the user to ensure correct operation of their
// overall system.
// CBUS Bit-Bang Mode Sample Code
// CBUS 0 is toggled low, then high, then all pins set to input
```

```
#include <Windows.h>
#include <stdio.h>
#include "ftd2xx.h"
FT_STATUS ftStatus;
FT_HANDLE ftHandle;
UCHAR Mask;
UCHAR Mode;

void main()

{

ftStatus = FT_Open(0, &ftHandle);
if (ftStatus != FT_OK) printf("ftStatus not ok %d\n", ftStatus); //check for error
Mode = 0x20; //0x20 = CBUS Bit-bang

// Set CBUS3 & CBUS0 to output, Set CBUS3 high, rest low.
//
//      | bit 7 | bit 6 | bit 5 | bit 4 | | bit 3 | bit 2 | bit 1 | bit 0 |
//      | CBUS3 | CBUS2 | CBUS 1 | CBUS 0 | | CBUS3 | CBUS 2 | CBUS 1 | CBUS 0 |
//      | I/O   | I/O   | I/O   | I/O   | | Hi/Lo | Hi/Lo | Hi/Lo | Hi/Lo |
//      | 1     | 0     | 0     | 1     | | 1     | 0     | 0     | 0     |
//
// Mask =          9                      8
//
Mask = 0x98;
// enter CBUS bit-bang mode
ftStatus = FT_SetBitMode(ftHandle, Mask, Mode);
Sleep(50); //delay 50ms

// Set CBUS3 & CBUS0 to output, Set CBUS3 & CBUS0 high, rest low.
Mask = 0x99;
//enter CBUS bit-bang mode
ftStatus = FT_SetBitMode(ftHandle, Mask, Mode);
Sleep(50); //delay 50ms

// Set All CBUS pins to input
Mask = 0x00;

ftStatus = FT_SetBitMode(ftHandle, Mask, Mode);

Sleep (5); //delay 5ms
//reset bit mode & set pins to input
Mode = 0x00;
Mask = 0x00;
ftStatus = FT_SetBitMode(ftHandle, Mask, Mode);
ftStatus = FT_Close(ftHandle); //close device

}
```

## 5 Synchronous Bit-Bang Mode Application Example

### 5.1 Introduction

This example shows how to use synchronous bit-bang mode to communicate with a serial EEPROM, the Microchip 93LC66B. This device has a memory size of 4096 bits, organized as 256 x 16 words. The application will read the memory contents at any address, or perform a sequential read from this address for the required number of locations. The FT231X module UM231XE was used for this example, but any FT-X device or FT-X module with at least four pins which can be configured for synchronous bit-bang can be used.

### 5.2 Hardware Connections

A total of four bit-bang lines are required from the FT-X device, three for output and one for input. In this example D0, D1, D2 & D3 have been chosen. The 93LC66B I/O consists of a clock input, a chip select input, a serial data input and a serial data output. **Figure 5.1** below shows the connections required between the UM231XE module and the 93LC66B eeprom.

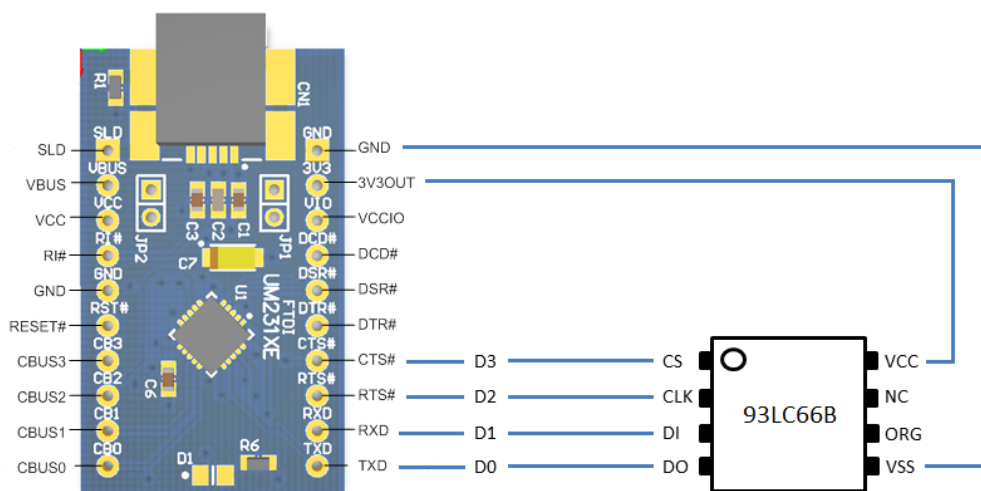
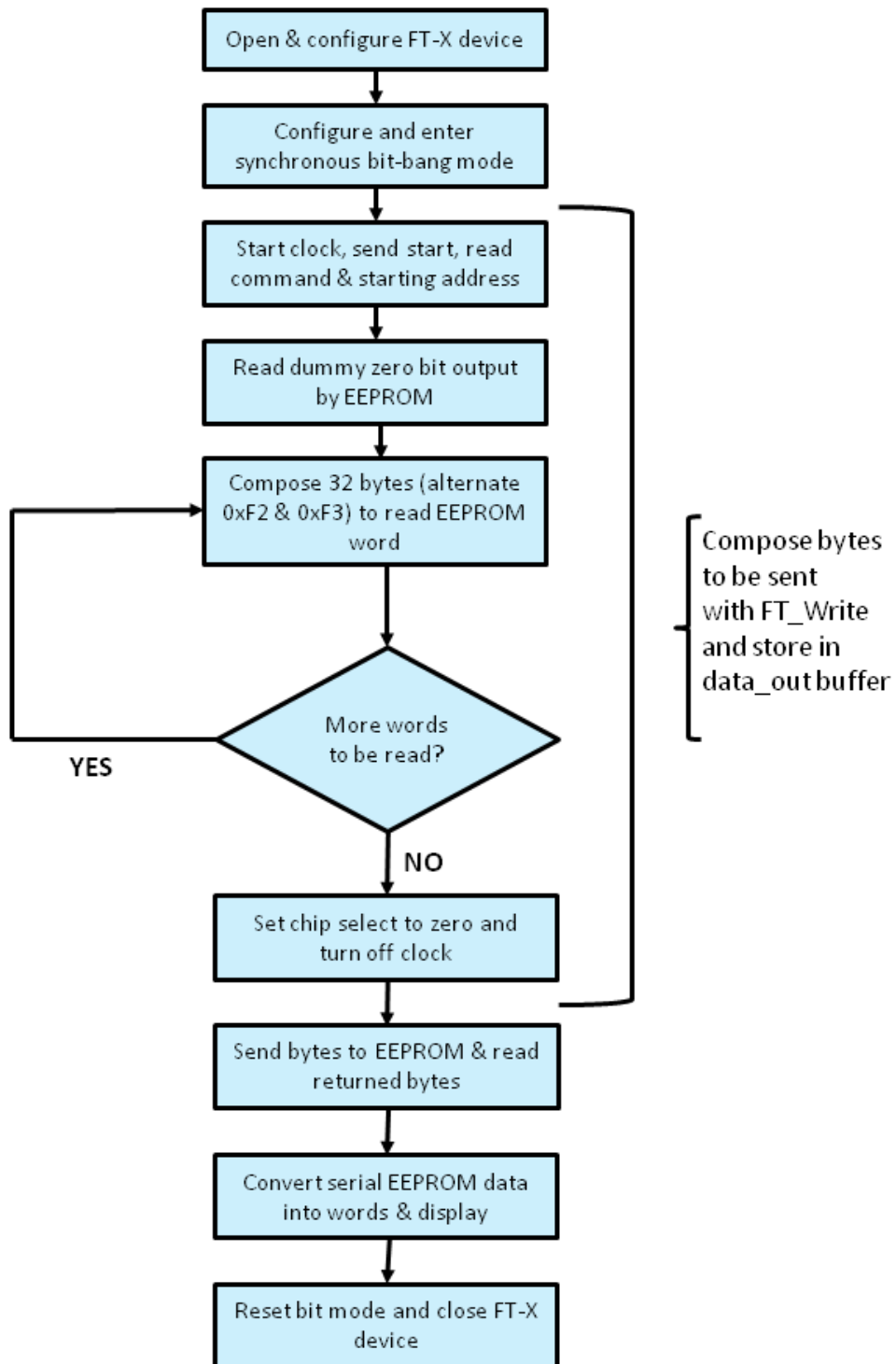


Figure 5.1 Hardware connections between UM231XE and 93LC66B

### 5.3 EEPROM Read Code Overview

The device is opened and then configured for synchronous bit-bang mode by using the command `FT_SetBitMode`. To read from the EEPROM, it is first necessary to clock the device with chip select high. The 'Start' condition then needs to be sent on the serial data input line, followed by the 'Read' command and the starting memory address. The EEPROM will then respond with a dummy bit followed by the 16 bits of the first memory location selected (MSB first). If chip select remains high and the clock line is active then the EEPROM will continue to output the contents of successive memory locations. Figure 5.2 shows the program flow chart. Figure 5.3 shows the program output.



**Figure 5.2 Synchronous bit-bang EEPROM read flow chart**

```

Synchronous bit-bang mode
93LC66B EEPROM read example

1 FTDI device(s) detected

Starting address = 0
No of memory locations to read = 256

Device now in synchronous bit-bang mode
8228 bytes written
8228 bytes received

0000: 0010 0403 6014 0900 3280 000c 0000 0aa0
0008: 32aa 12dc 0000 0000 1000 0001 0000 0066
0010: 0000 0000 0000 0000 0000 0000 0000 0000
0018: 0000 0000 0000 0000 0000 0000 0000 0000
0020: 0000 0000 0000 0000 0000 0000 0000 0000
0028: 0000 0000 0000 0000 0000 0000 0000 0000
0030: 0000 0000 0000 0000 0000 0000 0000 0000
0038: 0000 0000 0000 0000 0000 0000 0000 0000
0040: 0000 0000 0000 0000 0000 0000 0048 0000
0048: 0000 0000 0000 0000 0000 0000 0000 0000
0050: 030a 0046 0054 0044 0049 0332 ffff ffff
0058: 0042 0020 003c 002d 003e 0020 0053 0065
0060: 0072 0069 0061 ffff 0020 0043 006f 006e
0068: 0076 0065 0072 0074 0065 0072 0312 0046
0070: 0054 0058 004d 004a 004f 0044 0038 0000
0078: 0000 0000 0000 0000 0000 0000 0000 3378
0080: ffff ffff ffff ffff ffff ffff ffff ffff
0088: ffff ffff ffff ffff ffff ffff ffff ffff
0090: ffff ffff ffff ffff ffff ffff ffff ffff
0098: ffff ffff ffff ffff ffff ffff ffff ffff
00a0: ffff ffff ffff ffff ffff ffff ffff ffff
00a8: ffff ffff ffff ffff ffff ffff ffff ffff
00b0: ffff ffff ffff ffff ffff ffff ffff ffff
00b8: ffff ffff ffff ffff ffff ffff ffff ffff
00c0: ffff ffff ffff ffff ffff ffff ffff ffff
00c8: ffff ffff ffff ffff ffff ffff ffff ffff
00d0: ffff ffff ffff ffff ffff ffff ffff ffff
00d8: ffff ffff ffff ffff ffff ffff ffff ffff
00e0: ffff ffff ffff ffff ffff ffff ffff ffff
00e8: ffff ffff ffff ffff ffff ffff ffff ffff
00f0: ffff ffff ffff ffff ffff ffff ffff ffff
00f8: ffff ffff ffff ffff ffff ffff ffff ffff

EEPROM read complete, bit-bang mode reset
Press return to end program_
  
```

**Figure 5.3 Synchronous bit-bang program read - screen output when reading entire memory contents.**

## 5.4 Synchronous Bit-Bang Code to Read Microchip 93LC66B

```

//*****
// This code is provided as an example only and is not supported or guaranteed by FTDI
// It is the responsibility of the user to ensure correct operation of their
// overall system.
//
// Sample code to read the contents of a Microchip 93LC66B EEPROM
// using synchronous bit-bang mode.
// The EEPROM contains 256 x 16 = 4096 bits.
// The starting address and number of words to read can be set
// Ensure only 1 FTDI device is connected to the PC and connections to EEPROM are
// as per Figure 5.1 of AN 373
//*****

#include <windows.h>
#include <stdio.h>
#include "ftd2xx.h"

DWORD RxBytes; //number of bytes to be read from the device
DWORD BytesReceived; //number of bytes read from the device
DWORD w_data_len; //number of bytes to sent using FT_Write
DWORD data_written; //number of bytes written using FT_Write
WORD start_address = 0; //starting address - range 0 to 255 (decimal)
WORD no_of_words = 256; //number of words to read from EEPROM (maximum is 256)
WORD memcontents[256]; //array to hold memory contents
byte bytetosend; //used when converting address byte to serial bit stream
byte RxBuffer[8228]; //buffer to receive data from FT-X device
byte data_out[8228]; //buffer for data to send to EEPROM
int byte_multiple; //number of bytes to send via FT_Write
int byte_index; //index number for bytes to send
FT_STATUS ftStatus;
FT_HANDLE ftHandle;
DWORD numDevs;

//*****
// Main Program
//*****
void main()
{

    //disallow addresses>255
    if (start_address> 255) start_address = 0;
    //maximum number of memory locations is 256
    if (start_address + no_of_words > 256) no_of_words = 256 - start_address;

    ftStatus = FT_CreateDeviceInfoList(&numDevs);
    if (ftStatus == FT_OK) {
        printf("Synchronous bit-bang mode\n93LC66B EEPROM read example\n\n");
        printf("%d FTDI device(s) detected\n \n", numDevs);
    }
    else {
        printf("FT_CreateDeviceInfoList failed \n");
    }

    printf("Starting address (decimal) = %d\n", start_address);
    printf("No of memory locations to read = %d\n \n", no_of_words);
}

```

```

ftStatus = FT_Open(0, &ftHandle); // open FT-X device and obtain handle
ftStatus |= FT_SetUSBParameters(ftHandle, 8228, 8228); // Set USB request transfer size
ftStatus |= FT_SetChars(ftHandle, false, 0, false, 0); // Disable event/error chars
ftStatus |= FT_SetTimeouts(ftHandle, 5000, 5000); // Set read/write timeouts to 5 sec
ftStatus |= FT_SetLatencyTimer(ftHandle, 16); // Latency timer at default 16ms
ftStatus |= FT_SetBaudRate(ftHandle, 62500); //bit-bang rate is x16 this value = 1M
ftStatus |= FT_SetFlowControl(ftHandle, FT_FLOW_NONE, 0x11, 0x13);
if (ftStatus != FT_OK) printf("ftStatus not ok %d\n", ftStatus); // report error

//          FT-X D0 (output) connects to EEPROM DO
//          FT-X D1 (output) connects to EEPROM DI
//          FT-X D2 (output) connects to EEPROM CLK
//          FT-X D3 (input)  connects to EEPROM CS

UCHAR Mask = 0x0E; //00001110 - D1, D2, D3 are outputs, D0,D4,D5,D6,D7 are inputs
UCHAR Mode = 0x04; //Set for synchronous bit-bang mode

//Put FT-X device into synchronous bit-bang mode
ftStatus = FT_SetBitMode(ftHandle, Mask, Mode);

if (ftStatus != FT_OK)
    printf("FT_SetBitMode not ok %d\n", ftStatus);
else
{
    printf("Device now in synchronous bit-bang mode\n");
}

//*****
//start clock, compose read instruction and address bytes
//*****
//converts read command and addresses serial bits into output bytes
//
//start clock & send Start (1) & the EEPROM read command (10) on DI
//
//          CS  CLK  DI  DO
data_out[0] = 0xF0; //  1  1  1  1  0  0  0  0  start clock
data_out[1] = 0xF4; //  1  1  1  1  0  1  0  0
//-----
data_out[2] = 0xF0; //  1  1  1  1  0  0  0  0
data_out[3] = 0xF4; //  1  1  1  1  0  1  0  0
//-----
data_out[4] = 0xFA; //  1  1  1  1  1  0  1  0  cs high,
data_out[5] = 0xFE; //  1  1  1  1  1  1  1  0  write 1 to DI
//-----
data_out[6] = 0xFA; //  1  1  1  1  1  0  1  0  cs high,
data_out[7] = 0xFE; //  1  1  1  1  1  1  1  0  write 1 to DI
//-----
data_out[8] = 0xF8; //  1  1  1  1  1  0  0  0  cs high,
data_out[9] = 0xFC; //  1  1  1  1  1  1  0  0  write 0 to DI
//-----

//bit stream for the address
//convert address byte into serial bit stream for DI
//check each bit of the address byte - if high, set DI line to 1
//send most significant bit first
bytetosend = 0x08; //keep chip select high
if ((start_address & 0x01) != 0) { bytetosend = bytetosend + 0x02; }
data_out[24] = bytetosend + 0xF0; //clock low
data_out[25] = bytetosend + 0xF4; //clock high

```

```

bytetosend = 0x08;
if ((start_address & 0x02) != 0) { bytetosend = bytetosend + 0x02; }
data_out[22] = bytetosend + 0xF0; //clock low
data_out[23] = bytetosend + 0xF4; //clock high
bytetosend = 0x08;
if ((start_address & 0x04) != 0) { bytetosend = bytetosend + 0x02; }
data_out[20] = bytetosend + 0xF0; //clock low
data_out[21] = bytetosend + 0xF4; //clock high
bytetosend = 0x08;
if ((start_address & 0x08) != 0) bytetosend = bytetosend + 0x02;
data_out[18] = bytetosend + 0xF0; //clock low
data_out[19] = bytetosend + 0xF4; //clock high
bytetosend = 0x08;
if ((start_address & 0x10) != 0) bytetosend = bytetosend + 0x02;
data_out[16] = bytetosend + 0xF0; //clock low
data_out[17] = bytetosend + 0xF4; //clock high
bytetosend = 0x08;
if ((start_address & 0x20) != 0) bytetosend = bytetosend + 0x02;
data_out[14] = bytetosend + 0xF0; //clock low
data_out[15] = bytetosend + 0xF4; //clock high
bytetosend = 0x08;
if ((start_address & 0x40) != 0) bytetosend = bytetosend + 0x02;
data_out[12] = bytetosend + 0xF0; //clock low
data_out[13] = bytetosend + 0xF4; //clock high
bytetosend = 0x08;
if ((start_address & 0x80) != 0) bytetosend = bytetosend + 0x02;
data_out[10] = bytetosend + 0xF0; //clock low
data_out[11] = bytetosend + 0xF4; //clock high

//+++++
//end of start clock, compose read instruction and address bytes
//+++++

//*****
//Read the dummy zero bit, output by the EEPROM prior to data
//*****
//To read from the pins they need to be written to
//Keep chip select high, serial data input low and toggle clock
data_out[26] = 0xF8; //RxBuffer[27] stores byte read for this byte written
data_out[27] = 0xFC; //RxBuffer[28] stores byte read for this byte written

//+++++
//End of Read the dummy zero bit output by the EEPROM prior to data
//+++++

//*****
//Read the EEPROM data
//*****

byte_multiple = (no_of_words*16*2); //2 bytes need to be sent for each bit of the word
//compose bytes to be sent to read the data
//output 0 to the EEPROMs input, keep chip select high and toggle clock

for (byte_index = 28; byte_index < byte_multiple + 28; (byte_index = byte_index + 2)){
    data_out[byte_index] = 0xF8; //chip select high and clock low
}
for (byte_index = 29; byte_index < byte_multiple + 28; (byte_index = byte_index + 2)){
    data_out[byte_index] = 0xFC; //chip select high and clock high
}

```

```
//data read complete, now set chip select to zero with clock toggling
data_out[byte_index] = 0xF0;
data_out[byte_index + 1] = 0xF4;
data_out[byte_index + 2] = 0xF0;
data_out[byte_index + 3] = 0xF4;
data_out[byte_index + 4] = 0xF0;
//now turn off clock
data_out[byte_index + 5] = 0xF0;
data_out[byte_index + 6] = 0xF0;
data_out[byte_index + 7] = 0xF0;

// total number of bytes to write (w_data_len) is:
// 28 for start, read command and address +
// byte_multiple = (no_of_words*16*2) +
// 8 for end sequence

w_data_len = 28 + byte_multiple + 8;

//Now toggle the FT-X bit-bang pins by using FT_Write to send data_out contents
FT_Write(ftHandle, data_out, w_data_len, &data_written);

if ((ftStatus != FT_OK)|| (w_data_len!=data_written))
    printf("ftStatus not ok or all bytes not written %d\n", ftStatus);
else
{
    printf("%d bytes written\n", data_written);
}

//check queue status to see when all the bytes are returned
//in synchronous bit-bang, one byte is returned for each byte written

do{
    FT_GetQueueStatus(ftHandle, &RxBytes);
    Sleep(10); //delay
} while (RxBytes != (w_data_len)); //check for all bytes to be returned

//once bytes are received, use FT_Read to place in RxBuffer
if (RxBytes > 0) {
    ftStatus = FT_Read(ftHandle, RxBuffer, RxBytes, &BytesReceived);
    printf("%d bytes received\n \n", BytesReceived);
}

//+++++
//End of Read the EEPROM data
//+++++

//*****
//Convert serial data from EEPROM into words and store in memcontents
//*****

WORD mem;
WORD temp_mem;
INT byte_count = 0;
INT bit_shift = 15;
INT word_count = 1;

for (word_count = 1; word_count < ((byte_multiple / 32) + 1); word_count++)
{
    bit_shift = 15;
    mem = 0x0000;
```



```

temp_mem = 0x0000;
for (byte_count = ((28) + (32 * (word_count - 1))); byte_count <
((28) + (32 * word_count)); (byte_count = byte_count + 2))
{
temp_mem = 0x0000;
if ((RxBuffer[byte_count] & 0x01) != 0) { temp_mem = 0x0001; }
temp_mem = temp_mem << bit_shift;
mem = mem + temp_mem;
if (bit_shift == 0){
//store the reconstructed word in memcontents
memcontents[(start_address + (word_count - 1))] = mem;
}
bit_shift = bit_shift--;
}
}
//+++++
//End of convert serial data from EEPROM into words and store in memcontents
//+++++

//*****
//Print EEPROM contents to screen
//*****
int remainder;
int xcount;
remainder = no_of_words % 8;
xcount = no_of_words - remainder;

int d;
for (d = 0; d < xcount; (d = d + 8))
{
printf("%.4x:  %.4x %.4x %.4x %.4x %.4x %.4x %.4x %.4x\n", d + start_address,
memcontents[d + start_address], memcontents[d + start_address + 1], memcontents[d +
start_address + 2], memcontents[d + start_address + 3], memcontents[d + start_address
+ 4], memcontents[d + start_address + 5], memcontents[d + start_address + 6],
memcontents[d + start_address + 7]);
}

if (remainder != 0)
{
if (xcount<1)printf("%.4x:  ", start_address);
if (xcount>1) printf("%.4x:  ", start_address + xcount);
for (d = (xcount + start_address); (d < (xcount + start_address +
remainder)); d++)
{
printf("%.4x ", memcontents[d]);
}
}
printf("\n \n");

//+++++
//End of Print EEPROM contents to screen
//+++++

//*****
//Leave synchronous bit-bang mode and close the device
//*****
Mode = 0x00; //reset bit mode
ftStatus = FT_SetBitMode(ftHandle, Mask, Mode);

if (ftStatus != FT_OK)

```

```
        printf("FT_SetBitMode not ok %d\n", ftStatus);
    else
        printf("EEPROM read complete, bit-bang mode reset\n");
    ftStatus = FT_Close(ftHandle);
    printf("Press return to end program");
    getchar();
    printf("closed \n");
}
//+++++
//End of leave synchronous bit-bang mode and close the device
//+++++
//
//End of program
//
//*****
```

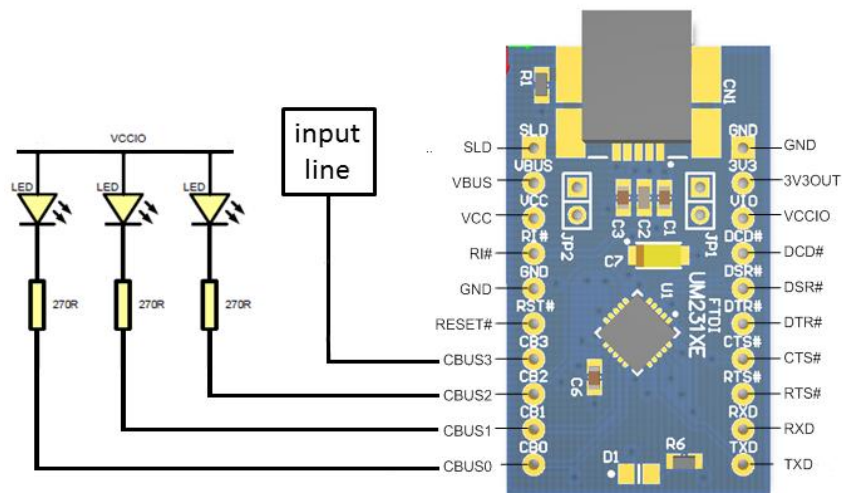
## 6 CBUS Bit-Bang Mode Application Example

### 6.1 Introduction

This example shows how to use CBUS bit-bang mode to control three LEDs and use an input line to select the lighting pattern.

### 6.2 Hardware Connections

A total of four CBUS bit-bang lines are required from the FT-X device, three for output and one for input. Suitable devices with the necessary number of CBUS pins are the FT201X, FT230X and FT231X. Figure 6.1 below shows the hardware connections required when using a UM231XE module.



**Figure 6.1** Hardware connections required for CBUS bit-bang LED example

### 6.3 CBUS Example Code Overview

The device is opened and then configured for CBUS bit-bang mode by using the command `FT_SetBitMode`. One of two lighting patterns for the LEDs is selected by checking the state of the input pin CBUS3. One option (CBUS3 input high) will turn on the LEDs in a traffic-light sequence. The other option (CBUS3 input low) will cause all three LEDs to flash simultaneously.

## 6.4 CBUS Bit-Bang LED Example Code

```
// This code is provided as an example only and is not supported or guaranteed by FTDI
// It is the responsibility of the recipient to ensure correct operation of their
// overall system
//
//Sample code which uses CBUS bit-bang mode to control LEDs.
//The LED pattern depends on the value of the CBUS3 bit-bang
//input pin. When pulled high a 'traffic light' sequence is
//displayed & when pulled low all three LEDs flash simultaneously.
```

```
#include <windows.h>
#include <stdio.h>
#include "ftd2xx.h"
```

```
BYTE BitMode; //value read from CBUS pins
UCHAR Mask = 0x77; //Upper nibble is I/O, lower nibble is high/low value
```

```
// Mask = | bit 7 | bit 6 | bit 5 | bit 4 | | bit 3 | bit 2 | bit 1 | bit 0 |
//          | CBUS3 | CBUS2 | CBUS 1 | CBUS 0 | | CBUS3 | CBUS 2 | CBUS 1 | CBUS 0 |
//          | I/O  | I/O  | I/O  | I/O  | | Hi/Lo | Hi/Lo | Hi/Lo | Hi/Lo |
//          | 1 for output, 0 for input  | | 1 for high, 0 for low  |
```

```
UCHAR Mode = 0x20; //Set for CBUS Bit-bang mode
FT_STATUS status;
DWORD numDevs;
FT_HANDLE fthandle;
```

```
void led_flash()
{
    //Mask = 0x70; //All LEDs ON
    status = FT_SetBitMode(fthandle, 0x70, Mode);
    Sleep(1000);
    //Mask = 0x77; //All LEDs OFF
    FT_SetBitMode(fthandle, 0x77, Mode);
    Sleep(1000);
    //Mask = 0x70; //All LEDs ON
    status = FT_SetBitMode(fthandle, 0x70, Mode);
    Sleep(1000);
    //Mask = 0x77; //All LEDs OFF
    FT_SetBitMode(fthandle, 0x77, Mode);
}
```

```
//*****
// Main Program
//*****
```

```
void main()
{
    status = FT_CreateDeviceInfoList(&numDevs);
    if (status == FT_OK) {
        printf("Number of devices is %d\n", numDevs);
        printf("\n");
    }
    else {
        printf("FT_CreateDeviceInfoList failed \n");
    }
}
```

```
status = FT_Open(0, &fthandle);
status = FT_SetBaudRate(fthandle, 9600); //not used by CBUS bit-bang

if (status != FT_OK)
    printf("status not ok %d\n", status);

// For traffic light sequence connect:
//   CBUS0 to a red LED,
//   CBUS1 to an amber LED
//   CBUS2 to a green LED

status = FT_SetBitMode(fthandle, 0xF7, Mode); //turn all LEDs off
status = FT_GetBitMode(fthandle, &BitMode);
if (status == FT_OK)
{
    printf("Bitmode = %x\n", BitMode); // BitMode contains current value
}
else
{
    printf("FT_GetBitMode FAILED!\n");
}
// Check if CBUS3 is high or low, if CBUS3 is low call led_flash()
if ((BitMode & 0x08) != 0x08) {
    led_flash();
}
else
{
    // If CBUS3 is high perform traffic light sequence
    Mask = 0x76; //CBUS0 LED on
    status = FT_SetBitMode(fthandle, Mask, Mode);
    Sleep(3000); //3 second delay
    Mask = 0x74; //CBUS0 & CBUS1 LEDs on
    status = FT_SetBitMode(fthandle, Mask, Mode);
    Sleep(2500);
    Mask = 0x73; //CBUS2 LED on
    status = FT_SetBitMode(fthandle, Mask, Mode);
    Sleep(2000);
    Mask = 0x75; //CBUS1 LED on
    status = FT_SetBitMode(fthandle, Mask, Mode);
    Sleep(1000);
    Mask = 0x76; //CBUS0 LED on
    status = FT_SetBitMode(fthandle, Mask, Mode);
    Sleep(2000);
    Mask = 0x77; //All LEDs off
    status = FT_SetBitMode(fthandle, Mask, Mode);
}

FT_Close(fthandle);
}
//End of program
```

## 7 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](mailto:sales1@ftdichip.com)  
E-mail (Support) [support1@ftdichip.com](mailto:support1@ftdichip.com)  
E-mail (General Enquiries) [admin1@ftdichip.com](mailto: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](mailto:us.sales@ftdichip.com)  
E-Mail (Support) [us.support@ftdichip.com](mailto:us.support@ftdichip.com)  
E-Mail (General Enquiries) [us.admin@ftdichip.com](mailto:us.admin@ftdichip.com)

### Branch Office – Taipei, Taiwan

Future Technology Devices International Limited  
(Taiwan)  
2F, No. 516, Sec. 1, NeiHu Road  
Taipei 114  
Taiwan, R.O.C.  
Tel: +886 (0) 2 8791 3570  
Fax: +886 (0) 2 8791 3576

E-mail (Sales) [tw.sales1@ftdichip.com](mailto:tw.sales1@ftdichip.com)  
E-mail (Support) [tw.support1@ftdichip.com](mailto:tw.support1@ftdichip.com)  
E-mail (General Enquiries) [tw.admin1@ftdichip.com](mailto: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](mailto:cn.sales@ftdichip.com)  
E-mail (Support) [cn.support@ftdichip.com](mailto:cn.support@ftdichip.com)  
E-mail (General Enquiries) [cn.admin@ftdichip.com](mailto: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

FT-X Series Product Page

<http://www.ftdichip.com/FT-X.htm>

UM231XE USB to Full-Handshake UART Development Module

[http://www.ftdichip.com/Support/Documents/DataSheets/Modules/DS\\_UMFT231XE.pdf](http://www.ftdichip.com/Support/Documents/DataSheets/Modules/DS_UMFT231XE.pdf)

<http://www.ftdichip.com/Products/Modules/DevelopmentModules.htm>

[D2xx Programmers Guide](#)

Latest FTDI Drivers

<http://www.ftdichip.com/Drivers/D2XX.htm>

The FTDI EEPROM programming utility FT\_Prog

[http://www.ftdichip.com/Support/Utilities.htm#FT\\_PROG](http://www.ftdichip.com/Support/Utilities.htm#FT_PROG)

### Acronyms and Abbreviations

Terms	Description
EEPROM	Electrically Erasable Programmable Read Only Memory
GPIO	General Purpose Input/output
I/O	Input/output
LED	Light Emitting Diode
MSB	Most Significant Bit
UART	Universal Asynchronous Receiver/Transmitter
VCP	Virtual COM port

## Appendix B – List of Tables & Figures

### List of Tables

Table 2.1 Asynchronous Bit-Bang Mode I/O Configurations.....	4
Table 2.2 Asynchronous Bit-Bang Mode D2XX commands.....	5
Table 3.1 Synchronous Bit-Bang Mode I/O Configurations .....	8
Table 3.2 Synchronous Bit-Bang Mode D2XX commands .....	9
Table 4.1 CBUS Bit-Bang Mode I/O Configurations .....	13
Table 4.2 CBUS Bit-Bang Mode D2XX commands .....	15

### List of Figures

Figure 2.1 D0 data output line & read and write strobes in asynchronous bit-bang mode .....	7
Figure 2.2 D0 data output line & read and write strobes in asynchronous bit-bang mode – 1010101 written to pin. ....	7
Figure 3.1 D0 data output line & read and write strobes in synchronous bit-bang mode .....	12
Figure 3.2 D0 data output line & read and write strobes in synchronous bit-bang mode – 1010101 written to pin. ....	12
Figure 4.1 CBUS0 set low then high in CBUS bit-bang mode.....	14
Figure 5.1 Hardware connections between UM231XE and 93LC66B.....	17
Figure 5.2 Synchronous bit-bang EEPROM read flow chart.....	18
Figure 5.3 Synchronous bit-bang program read - screen output when reading entire memory contents. ....	19
Figure 6.1 Hardware connections required for CBUS bit-bang LED example .....	26



## Appendix C – Revision History

Document Title: AN\_373 Bit-bang Modes for the FT-X Series  
Document Reference No.: FT\_001168  
Clearance No.: FTDI#527  
Product Page: <http://www.ftdichip.com/FTProducts.htm>  
Document Feedback: [Send Feedback](#)

Revision	Changes	Date
1.0	Initial Release	2017-05-17