Application Note

AN_113

Interfacing FT2232H Hi-Speed Devices to I2C Bus

Version 1.2

Issue Date: 2020-05-26

.

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
E-Mail (Support): **support1@ftdichip.com** Web: http://www.ftdichip.com

Table of Contents

1 Introduction

The FT2232H and FT4232H are the FTDI's first USB 2.0 Hi-Speed (480Mbits/s) USB to UART/FIFO ICs. They also have the capability of being configured in a variety of serial interfaces using the internal MPSSE (Multi-Protocol Synchronous Serial Engine). The FT2232H device has two independent ports, both of which can be configured using MPSSE while only Channel A and B of FT4232H can be configured using MPSSE.

Using MPSSE can simplify the synchronous serial protocol (USB to SPI, I^{2}C, JTAG, etc.) design. This application note illustrates how to use the MPSSE of the FT2232H to interface with the I^{2}C bus. Users can use the example schematic and functional software code to begin their design. Note that software code is provided as an illustration only and not supported by FTDI.

1.1 I^{2}C Bus Introduction

I2C is a low- to medium-data-rate master/slave communication bus. Two wires, serial data (SDA) and serial clock (SCL), carry information between the devices connected to the bus. Each device is recognized by a unique address and can operate as either a transmitter or receiver, depending on the function of the device. In addition to transmitters and receivers, devices can also be considered as masters or slaves when performing data transfers. A master is the device which initiates a data transfer on the bus. At that time, any device addressed is considered a slave.

The physical layer of I^{2}C bus is a simple handshaking protocol that relies upon open collector outputs on the bus devices and the device driving or releasing the bus lines, so a pull-up resistor is needed on each wire of the bus.

I^{2}C bus is a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer

Serial, 8-bit oriented, bi-directional data transfers can be made at up to 100 Kbit/s in the Standard-mode of I^{2}C bus, up to 400 Kbit/s in the Fast-mode or up to 3.4 Mbit/s in the High-speed mode.

Figure 1 shows typical data transfers on the I^{2}C bus. The master supplies the clock; it initiates and terminates transactions and the intended slave (based upon the address provided by the master) acknowledges the master by driving or releasing the bus. The slave cannot terminate the transaction but can indicate a desire to by a "NAK" or not-acknowledge.

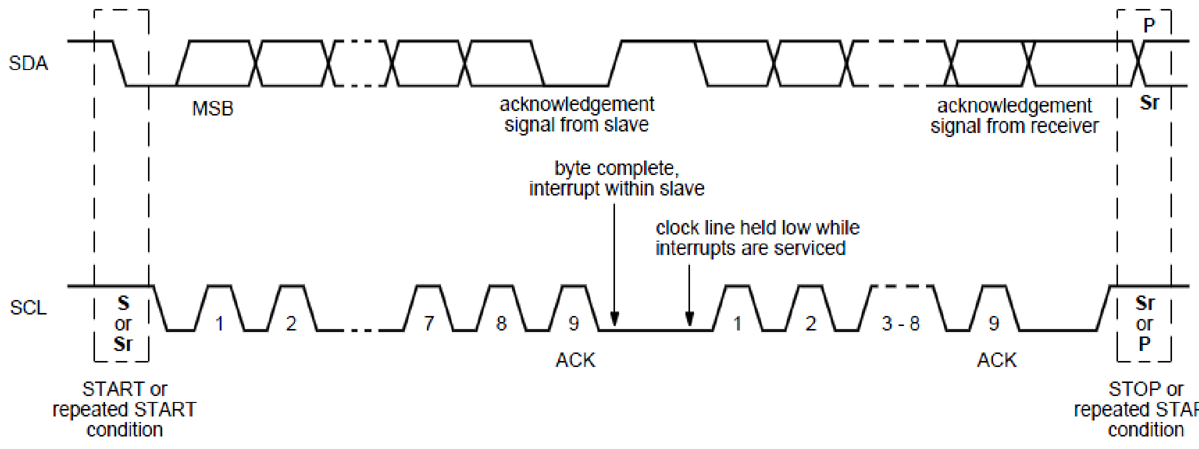
SDA
SCL
MSB
acknowledgement signal from slave
acknowledgement signal from receiver
byte complete, interrupt within slave
clock line held low while interrupts are serviced
P
Sr
S or Sr
1
2
7
8
9
1
2
3 - 8
9
ACK
ACK
Sr or P
START or repeated START condition
STOP or repeated START condition

Figure 1 Data transfer on I^{2}C bus

I^{2}C specification defines unique situations as START (S) and STOP (P) conditions (see Figure 2). A HIGH to LOW transition on the SDA line while SCL is HIGH indicates a START condition. A LOW to HIGH transition on the SDA line while SCL is HIGH defines a STOP condition. START and STOP conditions are always generated by the master.

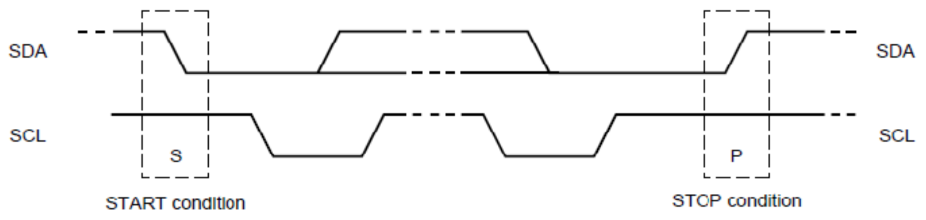

Figure 2 START and STOP conditions

Every byte put on the SDA line must be 8-bits long. The number of bytes can be transmitted per transfer is unrestricted. Each byte is followed by an acknowledge bit. Data is transferred with the most significant bit (MSB) first. In most cases, data transfer with acknowledge is obligatory. The acknowledge–related clock pulse is generated by the master. The transmitter releases the SDA line (HIGH) during the acknowledge clock pulse. The receiver must pull down the SDA line during the acknowledge clock pulse so that it remains stable LOW during the HIGH period of this clock pulse (see Figure 3). Also, set-up and hold times must also be taken into account.

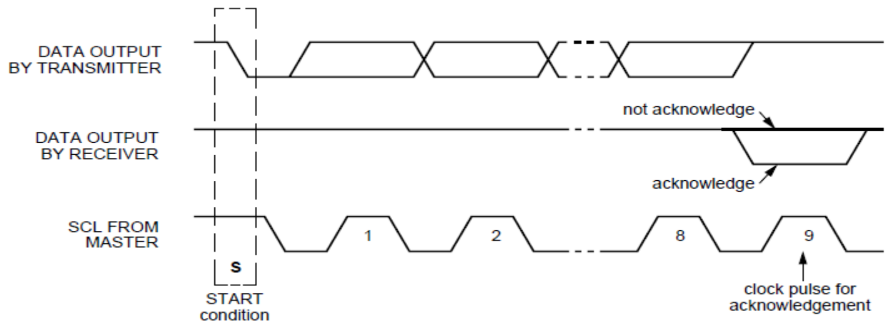

Figure 3 Acknowledge on the I^{2}C-bus

Data transfers of I^{2}C specification should follow the format. After the START condition (S), a slave address should be sent first. This address is 7 bits long followed by an eighth bit which is a data direction bit (R/$\overline{\text{W}}$) – a 'zero' indicates a transmission (WRITE), a 'one' indicates a request for data (READ). After the slave address byte is sent, master can continue its data transfer by writing or reading data byte as defined format. The data transfer is always terminated by a STOP condition generated by the master.

2 Sample Project with FT2232H

2.1 Overview

To demonstrate how to use the Multi-Protocol Synchronous Serial Engine (MPSSE) in a USB to I^{2}C bus interface, a sample project is given. An EEPROM (24LC256) device with I^{2}C serial interface is selected as the typical application. A reference schematic showing the I^{2}C connection between the FT2232H and the 24LC256 is given. Additionally some sample software (C++ listing) is provided which illustrates how to initialize, program and read 24LC256 EEPROM device via the I^{2}C interface.

2.2 Sample Circuit

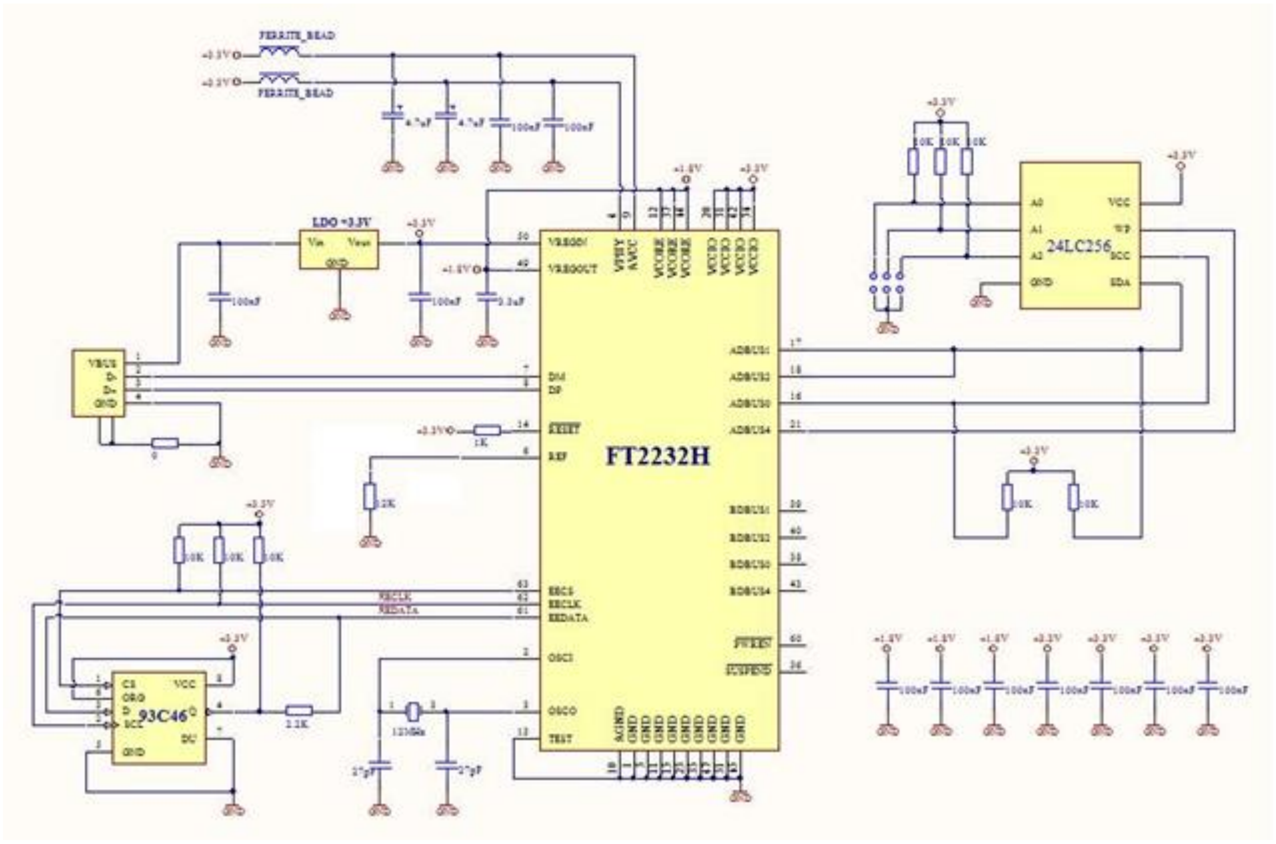

Figure 4 FT2232H interface with 24LC256

Figure 1 illustrates an example of interfacing the MPSSE port of FT2232H with I^{2}C serial EEPROM device. The FT2232H is in USB bus powered design configuration. Please refer to FT2232H datasheet for detailed specifications. The USB VBus (+5V) is regulated to +3.3V to supply VCCIO, VPHY, VPLL and VREGIN of FT2232H, the 93C46 EEPROM and the 24LC256 EEPROM devices. The output of the on-chip LDO regulator (+1.8V) of FT2232H drives the FT2232H core supply (VCORE). The 93C46 EEPROM is an option to allow customization of USB settings such as VID/PID, description, serial number, Remote Wake up, etc.

A Microchip 24LC256 serial EEPROM is used as the target device. The 24LC256 is a 256K bits Electrically Erasable PROM with 2-wire I^{2}C compatible serial interface. It supports a serial clock frequency of up to 400 kHz. Please refer to the 24LC256 datasheet for a detailed specification.

The A0, A1, A2 pins are used by the 24LC256 device for multiple device operation. The chip is selected when the levels on these inputs are identical with slave address. They may be tied to either VCC or VSS in above circuit.

Both channels of FT2232H can be configured to work in MPSSE mode. Channel A is used in **Figure** 1. Detailed pin definitions for the FT2232H I^{2}C connection are described in Table 1. A detailed functional description is given below:

SK – Connect to SCL pin of 24LC256. Serial clock pin to synchronize the data transfer, initiated by FT2232H and output to 24LC256. The clock value is determined by FT2232H internal clock divisor and can be up to 30 MHz, this signal requires a pull-up resistor to VCC (typical 10 KΩ for 100 kHz and 2 KΩ for 400 kHz).

DO/DI – Wired together and connected to SDA pin of 24LC256 for bidirectional data transfer. DO set as output pin to transfer serial data or address from FT2232H to 24LC256 device. DI set as input pin to receive serial data input pin from 24LC256 device to FT2232H. Requires a pull-up resistor to VCC (typical 10 KΩ for 100 kHz and 2 KΩ for 400 kHz).

GPIOL0 – General purpose I/O of MPSSE port of FT2232H. Connect to WP pin of 24LC256 device to control the Write Protect function of 24LC256 device. This is set to a logic low level for normal operation mode of the 24LC256. If set to a logic high level, the write operation of 24LC256 device is prohibited.

Channel A		Channel B		MPSSE Pin Name	Type	Description
Pin#	Pin Name	Pin #	Pin Name			
16	ADBUS0	38	BDBUS0	SK	Output	Serial Clock
17	ADBUS1	39	BDBUS1	DO	Output	Serial Data/Address Output
18	ADBUS2	40	BDBUS2	DI	Input	Serial Data Input
21	ADBUS4	43	BDBUS4	GPIOL0/ GPIOH0	Output	Write Protect control output

Table 1 Pin Description of FT2232H connecting with 24LC256

2.3 Sample Code

The following sample code illustrates the commands required by MPSSE. For further details on these commands, please refer to AN-108 Command Processor for MPSSE and MCU Host Bus Emulation Modes. Additionally, the FTDI D2XX driver is required to be installed on the system. Please refer to D2XX Programmer's Guide for APIs available in D2XX driver. The sample code has been compiled under Visual Studio 2008.

2.3.1 Definitions and Functions

```
////////////////////////////////////////////////////////////////////
// Define the global variables and const variables
////////////////////////////////////////////////////////////////////
const BYTE MSB_FALLING_EDGE_CLOCK_BYTE_IN = '\x24';
const BYTE MSB_FALLING_EDGE_CLOCK_BYTE_OUT = '\x11';
const BYTE MSB_RISING_EDGE_CLOCK_BIT_IN = '\x22';


FT_STATUS ftStatus;                 //Status defined in D2XX to indicate operation result
FT_HANDLE ftHandle;                 //Handle of FT2232H device port
BYTE OutputBuffer[1024];            //Buffer to hold MPSSE commands and data to be sent to FT2232H
BYTE InputBuffer[1024];             //Buffer to hold Data bytes to be read from FT2232H
DWORD dwClockDivisor = 0x0095;      //Value of clock divisor, SCL Frequency = 60/((1+0x0095)*2) (MHz) = 200khz
DWORD dwNumBytesToSend = 0;         //Index of output buffer
DWORD dwNumBytesSent = 0,  dwNumBytesRead = 0,  dwNumInputBuffer = 0;


////////////////////////////////////////////////////////////////////////////////////
// Below function will setup the START condition for I2C bus communication. First, set SDA, SCL high and ensure hold time
// requirement by device is met. Second, set SDA low, SCL high and ensure setup time requirement met. Finally, set SDA, SCL low
////////////////////////////////////////////////////////////////////////////////////
void HighSpeedSetI2CStart(void)
{
    DWORD dwCount;
    for(dwCount=0; dwCount < 4; dwCount++) // Repeat commands to ensure the minimum period of the start hold time ie 600ns is achieved
    {
        OutputBuffer[dwNumBytesToSend++] = '\x80'; //Command to set directions of lower 8 pins and force value on bits set as output
        OutputBuffer[dwNumBytesToSend++] = '\x03';  //Set SDA, SCL high, WP disabled by SK, DO at bit '1', GPIOL0 at bit '0'
        OutputBuffer[dwNumBytesToSend++] = '\x13';      //Set SK,DO,GPIOL0 pins as output with bit '1', other pins as input with bit '0'
    }
    for(dwCount=0; dwCount < 4; dwCount++) // Repeat commands to ensure the minimum period of the start setup time ie 600ns is achieved
    {
        OutputBuffer[dwNumBytesToSend++] = '\x80'; //Command to set directions of lower 8 pins and force value on bits set as output
        OutputBuffer[dwNumBytesToSend++] = '\x01';      //Set SDA low, SCL high, WP disabled by SK at bit '1', DO, GPIOL0 at bit '0'
        OutputBuffer[dwNumBytesToSend++] = '\x13';      //Set SK,DO,GPIOL0 pins as output with bit '1', other pins as input with bit '0'
    }
    OutputBuffer[dwNumBytesToSend++] = '\x80';  //Command to set directions of lower 8 pins and force value on bits set as output
    OutputBuffer[dwNumBytesToSend++] = '\x00';  //Set SDA, SCL low, WP disabled by SK, DO, GPIOL0 at bit '0'
    OutputBuffer[dwNumBytesToSend++] = '\x13';  //Set SK,DO,GPIOL0 pins as output with bit '1', other pins as input with bit '0'
}

//////////////////////////////////////////////////////////////////////////////////////
// Below function will setup the STOP condition for I2C bus communication. First, set SDA low, SCL high and ensure setup time
// requirement by device is met. Second, set SDA, SCL high and ensure hold time requirement met. Finally, set SDA, SCL as input
// to tristate the I2C bus.
//////////////////////////////////////////////////////////////////////////////////////
void HighSpeedSetI2CStop(void)
{
    DWORD dwCount;
    for(dwCount=0; dwCount<4; dwCount++) // Repeat commands to ensure the minimum period of the stop setup time ie 600ns is achieved
    {
        OutputBuffer[dwNumBytesToSend++] = '\x80';  //Command to set directions of lower 8 pins and force value on bits set as output
        OutputBuffer[dwNumBytesToSend++] = '\x01';  //Set SDA low, SCL high, WP disabled by SK at bit '1', DO, GPIOL0 at bit '0'
        OutputBuffer[dwNumBytesToSend++] = '\x13';  //Set SK,DO,GPIOL0 pins as output with bit '1', other pins as input with bit '0'
    }
    for(dwCount=0; dwCount<4; dwCount++) // Repeat commands to ensure the minimum period of the stop hold time ie 600ns is achieved
    {
        OutputBuffer[dwNumBytesToSend++] = '\x80';  //Command to set directions of lower 8 pins and force value on bits set as output
        OutputBuffer[dwNumBytesToSend++] = '\x03';  //Set SDA, SCL high, WP disabled by SK, DO at bit '1', GPIOL0 at bit '0'
        OutputBuffer[dwNumBytesToSend++] = '\x13';  //Set SK,DO,GPIOL0 pins as output with bit '1', other pins as input with bit '0'
    }
    //Tristate the SCL, SDA pins
    OutputBuffer[dwNumBytesToSend++] = '\x80';  //Command to set directions of lower 8 pins and force value on bits set as output
    OutputBuffer[dwNumBytesToSend++] = '\x00';  //Set WP disabled by GPIOL0 at bit '0'
    OutputBuffer[dwNumBytesToSend++] = '\x10';  //Set GPIOL0 pins as output with bit '1', SK, DO and other pins as input with bit '0'
}

//////////////////////////////////////////////////////////////////////////////////////
// Below function will send a data byte to I2C-bus EEPROM 24LC256, then check if the ACK bit sent from 24LC256 device can be received.
// Return true if data is successfully sent and ACK bit is received. Return false if error during sending data or ACK bit can't be received
//////////////////////////////////////////////////////////////////////////////////////
BOOL SendByteAndCheckACK(BYTE dwDataSend)
{
    FT_STATUS ftStatus = FT_OK;
```

```
        OutputBuffer[dwNumBytesToSend++] = MSB_FALLING_EDGE_CLOCK_BYTE_OUT;  //Clock data byte out on –ve Clock Edge MSB first
        OutputBuffer[dwNumBytesToSend++] = '\x00';
        OutputBuffer[dwNumBytesToSend++] = '\x00';          //Data length of 0x0000 means 1 byte data to clock out
        OutputBuffer[dwNumBytesToSend++] = dwDataSend;      //Add data to be send
        //Get Acknowledge bit from EEPROM
        OutputBuffer[dwNumBytesToSend++] = '\x80';  //Command to set directions of lower 8 pins and force value on bits set as output
        OutputBuffer[dwNumBytesToSend++] = '\x00';  //Set SCL low, WP disabled by SK, GPIOL0 at bit ‘0’
        OutputBuffer[dwNumBytesToSend++] = '\x11';  //Set SK, GPIOL0 pins as output with bit ‘1’, DO and other pins as input with bit ‘0’
        OutputBuffer[dwNumBytesToSend++] = MSB_RISING_EDGE_CLOCK_BIT_IN; //Command to scan in ACK bit , -ve clock Edge MSB first
        OutputBuffer[dwNumBytesToSend++] = '\x0';   //Length of 0x0 means to scan in 1 bit
        OutputBuffer[dwNumBytesToSend++] = '\x87';  //Send answer back immediate command
        ftStatus = FT_Write(ftHandle, OutputBuffer, dwNumBytesToSend, &dwNumBytesSent);      //Send off the commands
        dwNumBytesToSend = 0;                       //Clear output buffer
        //Check if ACK bit received, may need to read more times to get ACK bit or fail if timeout
        ftStatus = FT_Read(ftHandle, InputBuffer, 1, &dwNumBytesRead);     //Read one byte from device receive buffer
        if ((ftStatus != FT_OK) || (dwNumBytesRead == 0))
        {    return FALSE;        /*Error, can't get the ACK bit from EEPROM */          }
        else
            if (((InputBuffer[0] & BYTE('\x1'))  != BYTE('\x0'))      )          //Check ACK bit 0 on data byte read out
            {            return FALSE;        /*Error, can't get the ACK bit from EEPROM */          }
        OutputBuffer[dwNumBytesToSend++] = '\x80';  //Command to set directions of lower 8 pins and force value on bits set as output
        OutputBuffer[dwNumBytesToSend++] = '\x02';  //Set SDA high, SCL low, WP disabled by SK at bit '0', DO, GPIOL0 at bit '1'
        OutputBuffer[dwNumBytesToSend++] = '\x13';  //Set SK,DO,GPIOL0 pins as output with bit ‘1’, other pins as input with bit ‘0’
        return TRUE;
}
```

2.3.2 Initialise EEPROM Device

The following sample code will demonstrate how to open the FT2232H device handle, initialize the device, enable the MPSSE mode and set basic USB related settings based on D2xx APIs. Then a bad command is sent to synchronize the MPSSE channel. This is followed by configuring the MPSSE to communicate with 24LC256 I^{2}C–bus device (sets serial clock and pin direction/values).

```
DWORD dwCount;
//Try to open the FT2232H device port and get the valid handle for subsequent access
char SerialNumBuf[64];
ftStatus = FT_ListDevices((PVOID)0,& SerialNumBuf, FT_LIST_BY_INDEX|FT_OPEN_BY_SERIAL_NUMBER);
ftStatus = FT_OpenEx((PVOID) SerialNumBuf, FT_OPEN_BY_SERIAL_NUMBER, &ftHandle);
if (ftStatus == FT_OK)
{    // Port opened successfully
        ftStatus |= FT_ResetDevice(ftHandle);           //Reset USB device
        //Purge USB receive buffer first by reading out all old data from FT2232H receive buffer
        ftStatus |= FT_GetQueueStatus(ftHandle, &dwNumInputBuffer);        // Get the number of bytes in the FT2232H receive buffer
        if ((ftStatus == FT_OK) && (dwNumInputBuffer > 0))
             FT_Read(ftHandle, &InputBuffer, dwNumInputBuffer, &dwNumBytesRead);   //Read out the data from FT2232H receive buffer
        ftStatus |= FT_SetUSBParameters(ftHandle, 65536, 65535);           //Set USB request transfer size
        ftStatus |= FT_SetChars(ftHandle, false, 0, false, 0);         //Disable event and error characters
        ftStatus |= FT_SetTimeouts(ftHandle, 0, 5000);             //Sets the read and write timeouts in milliseconds for the FT2232H
        ftStatus |= FT_SetLatencyTimer(ftHandle, 16);              //Set the latency timer
        ftStatus |= FT_SetBitMode(ftHandle, 0x0, 0x00);            //Reset controller
        ftStatus |= FT_SetBitMode(ftHandle, 0x0, 0x02);            //Enable MPSSE mode
        if (ftStatus != FT_OK)
```

```
{     /*Error on initialize MPSEE of FT2232H*/    }
Sleep(50);        // Wait for all the USB stuff to complete and work

/////////////////////////////////////////////////////////////////
// Below codes will synchronize the MPSSE interface by sending bad command 'xAA' and checking  if the echo command followed by
// bad command 'AA' can be received, this will make sure the MPSSE interface enabled and synchronized successfully
/////////////////////////////////////////////////////////////////
OutputBuffer[dwNumBytesToSend++] = '\xAA';              //Add BAD command 'xAA'
ftStatus = FT_Write(ftHandle, OutputBuffer, dwNumBytesToSend, &dwNumBytesSent);       // Send off the BAD commands
dwNumBytesToSend = 0;                           //Clear output buffer
do{
      ftStatus = FT_GetQueueStatus(ftHandle, &dwNumInputBuffer);    // Get the number of bytes in the device input buffer
}while ((dwNumInputBuffer == 0) && (ftStatus == FT_OK));            //or Timeout
bool bCommandEchod = false;
ftStatus = FT_Read(ftHandle, &InputBuffer, dwNumInputBuffer, &dwNumBytesRead);  //Read out the data from input buffer
for (dwCount = 0; dwCount < dwNumBytesRead - 1; dwCount++)      //Check if Bad command and echo command received
{
      if ((InputBuffer[dwCount] == BYTE('\xFA')) && (InputBuffer[dwCount+1] == BYTE('\xAA')))
      {
            bCommandEchod = true;
            break;
      }
}
if (bCommandEchod == false)
{     /*Error, can't receive echo command , fail to synchronize MPSSE interface;*/ }

//////////////////////////////////////////////////////////////////
//Configure the MPSSE settings for I2C communication with 24LC256
/////////////////////////////////////////////////////////////////
OutputBuffer[dwNumBytesToSend++] = '\x8A';  //Ensure disable clock divide by 5 for 60Mhz master clock
OutputBuffer[dwNumBytesToSend++] = '\x97';   //Ensure turn off adaptive clocking
OutputBuffer[dwNumBytesToSend++] = '\x8C';  //Enable 3 phase data clock, used by I2C to allow data on both clock edges
ftStatus = FT_Write(ftHandle, OutputBuffer, dwNumBytesToSend, &dwNumBytesSent);       // Send off the commands
dwNumBytesToSend = 0;                           //Clear output buffer
OutputBuffer[dwNumBytesToSend++] = '\x80';  //Command to set directions of lower 8 pins and force value on bits set as output
OutputBuffer[dwNumBytesToSend++] = '\x03';  //Set SDA, SCL high, WP disabled by SK, DO at bit '1', GPIOL0 at bit '0'
OutputBuffer[dwNumBytesToSend++] = '\x13';  //Set SK,DO,GPIOL0 pins as output with bit ', other pins as input with bit ''
// The SK clock frequency can be worked out by below algorithm with divide by 5 set as off
// SK frequency  = 60MHz /((1 +  [(1 +0xValueH*256) OR 0xValueL])*2)
OutputBuffer[dwNumBytesToSend++] = '\x86';                        //Command to set clock divisor
OutputBuffer[dwNumBytesToSend++] = dwClockDivisor & '\xFF';       //Set 0xValueL of clock divisor
OutputBuffer[dwNumBytesToSend++] = (dwClockDivisor >> 8) & '\xFF';          //Set 0xValueH of clock divisor
ftStatus = FT_Write(ftHandle, OutputBuffer, dwNumBytesToSend, &dwNumBytesSent);       // Send off the commands
dwNumBytesToSend = 0;                           //Clear output buffer
Sleep(20);                                      //Delay for a while
//Turn off loop back in case
OutputBuffer[dwNumBytesToSend++] = '\x85';              //Command to turn off loop back of TDI/TDO connection
ftStatus = FT_Write(ftHandle, OutputBuffer, dwNumBytesToSend, &dwNumBytesSent);       // Send off the commands
dwNumBytesToSend = 0;                           //Clear output buffer
Sleep(30);                                      //Delay for a while
```

```
}
```

2.3.3 Program EEPROM – Random Byte Address

This example illustrates the data transfer sequence on the I^{2}C–bus required to write to the 24LC256 at random byte address. This is shown in **Figure** 5. The sample code below demonstrates how to implement these data transfer sequences based on MPSSE commands. An oscilloscope is used to capture the resultant waveforms. These are shown in **Figure** 6 for reference.

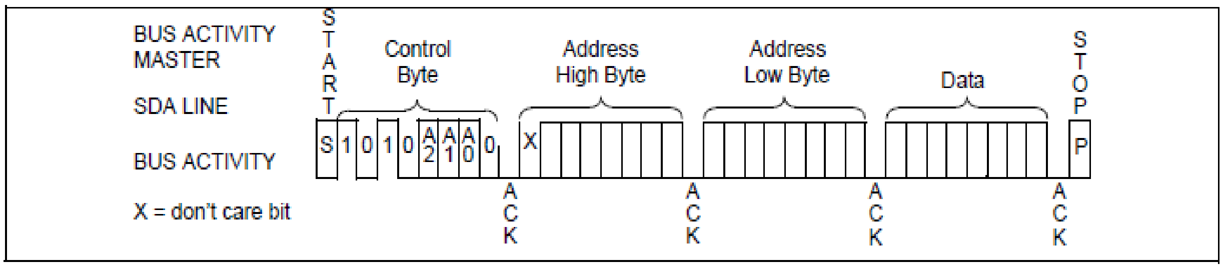

Figure 5 Data transfer sequence for EEPROM random address program

```
BOOL bSucceed = TRUE;
BYTE ByteAddressHigh = 0x00;
BYTE ByteAddressLow = 0x80;             //Set program address is 0x0080 as example
BYTE ByteDataToBeSend = 0x5A;           //Set data byte to be programmed as example
ighSpeedSetI2CStart();                  //Set START condition for I2C communication
bSucceed = SendByteAndCheckACK(0xAE);            //Set control byte and check ACK bit.  bit 4-7 of control byte is control code,
                                                 // bit 1-3 of '111' as block select bits, bit 0 of '0'represent Write operation
bSucceed = SendByteAndCheckACK(ByteAddressHigh);         //Send high address byte and check if ACK bit is received
bSucceed = SendByteAndCheckACK(ByteAddressLow);          //Send low address byte and check if ACK bit is received
bSucceed = SendByteAndCheckACK(ByteDataToBeSend);        //Send data byte and check if ACK bit is received
HighSpeedSetI2CStop();                  //Set STOP condition for I2C communication
//Send off the commands
ftStatus = FT_Write(ftHandle, OutputBuffer, dwNumBytesToSend, &dwNumBytesSent);
dwNumBytesToSend = 0;                   //Clear output buffer
Sleep(50);                              //Delay for a while to ensure EEPROM program is completed
```

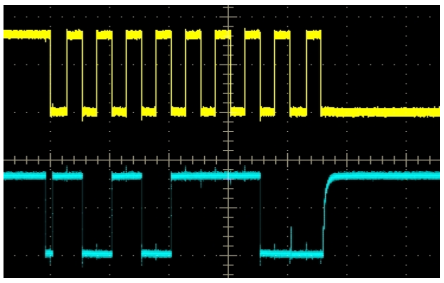
Fig 6-1 Sending control byte 0xAE

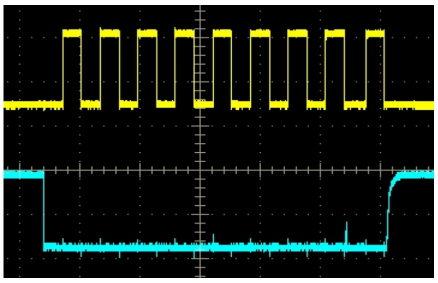
Fig 6-2 Sending high address byte 0x00

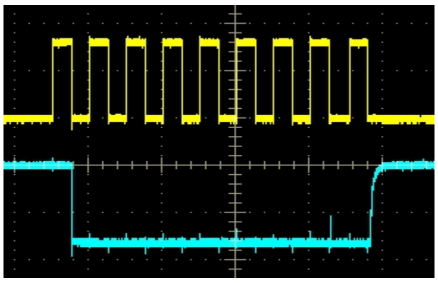
Fig 6-3 Sending low address byte 0x80

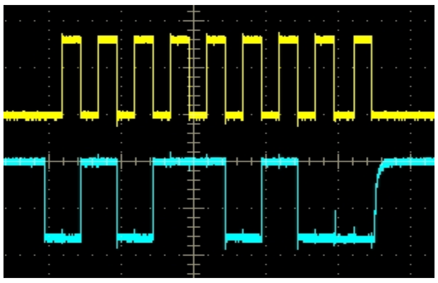
Fig 6-4 Sending data byte 0x5A

Figure 6 Waveforms for EEPROM random address program

2.3.4 Read EEPROM – Random Byte address

This example illustrates the data transfer sequence on the I²C–bus required to read from the 24LC256 at random byte addresses. The data transfer sequence on I²C–bus to read 24LC256 device with random byte address is shown in **Figure** 7. The sample code below demonstrates how to implement these data transfer sequences based on MPSSE commands. An oscilloscope is used to capture the resultant waveforms. These are shown in **Figure** 8 for reference.

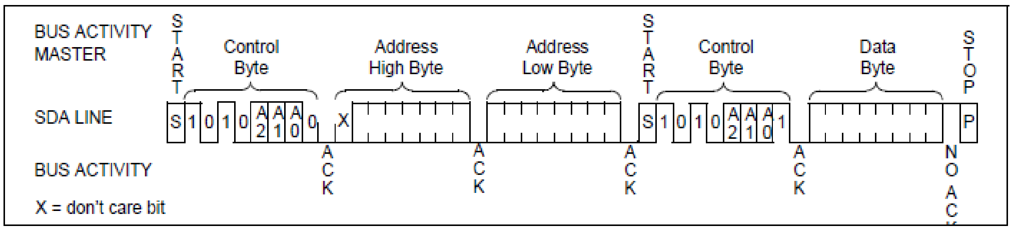

Figure 7 Data transfer sequence for EEPROM random address read

```
BOOL bSucceed = TRUE;
BYTE ByteAddressHigh = 0x00;
BYTE ByteAddressLow = 0x80;                  //Set read address is 0x0080 as example
BYTE ByteDataRead;                           //Data to be read from EEPROM

//Purge USB receive buffer first before read operation
ftStatus = FT_GetQueueStatus(ftHandle, &dwNumInputBuffer);    // Get the number of bytes in the device receive buffer
if ((ftStatus == FT_OK) && (dwNumInputBuffer > 0))
    FT_Read(ftHandle, &InputBuffer, dwNumInputBuffer, &dwNumBytesRead);  //Read out all the data from receive buffer

HighSpeedSetI2CStart();                             //Set START condition for I2C communication
bSucceed = SendByteAndCheckACK(0xAE);               //Set control byte and check ACK bit.  bit 4-7 of control byte is control code,
                                                    // bit 1-3 of '111' as block select bits, bit 0 of '0'represent Write operation
bSucceed = SendByteAndCheckACK(ByteAddressHigh);            //Send high address byte and check if ACK bit is received
bSucceed = SendByteAndCheckACK(ByteAddressLow);             //Send low address byte and check if ACK bit is received
HighSpeedSetI2CStart();                             //Set START condition for I2C communication
bSucceed = SendByteAndCheckACK(0xAF);               //Set control byte and check ACK bit.  bit 4-7 as '1010' of control byte is control code,
                                                    // bit 1-3 of '111' as block select bits, bit 0 as '1'represent Read operation

////////////////////////////////////////////////////////////
// Read the data from 24LC256 with no ACK bit check
////////////////////////////////////////////////////////////
OutputBuffer[dwNumBytesToSend++] = '\x80';          //Command to set directions of lower 8 pins and force value on bits set as output
OutputBuffer[dwNumBytesToSend++] = '\x00';          //Set SCL low, WP disabled by SK, GPIOL0 at bit ''
OutputBuffer[dwNumBytesToSend++] = '\x11';          //Set SK, GPIOL0 pins as output with bit '', DO and other pins as input with bit ''
OutputBuffer[dwNumBytesToSend++] = MSB_FALLING_EDGE_CLOCK_BYTE_IN;  //Command to clock data byte in on –ve Clock Edge MSB first
```

```
OutputBuffer[dwNumBytesToSend++] = '\x00';
OutputBuffer[dwNumBytesToSend++] = '\x00';        //Data length of 0x0000 means 1 byte data to clock in
OutputBuffer[dwNumBytesToSend++] = MSB_RISING_EDGE_CLOCK_BIT_IN;  //Command to scan in acknowledge bit , -ve clock Edge MSB first
OutputBuffer[dwNumBytesToSend++] = '\x0';         //Length of 0 means to scan in 1 bit
OutputBuffer[dwNumBytesToSend++] = '\x87';        //Send answer back immediate command
ftStatus = FT_Write(ftHandle, OutputBuffer, dwNumBytesToSend, &dwNumBytesSent);          //Send off the commands
dwNumBytesToSend = 0;                             //Clear output buffer
//Read two bytes from device receive buffer, first byte is data read from EEPROM, second byte is ACK bit
ftStatus = FT_Read(ftHandle, InputBuffer, 2, &dwNumBytesRead);
ByteDataRead = InputBuffer[0];              //Return the data read from EEPROM

OutputBuffer[dwNumBytesToSend++] = '\x80';        //Command to set directions of lower 8 pins and force value on bits set as output
OutputBuffer[dwNumBytesToSend++] = '\x02';        //Set SDA high, SCL low, WP disabled by SK at bit '0', DO, GPIOL0 at bit '1'
OutputBuffer[dwNumBytesToSend++] = '\x13';        //Set SK,DO,GPIOL0 pins as output with bit '', other pins as input with bit ''

HighSpeedSetI2CStop();                            //Set STOP condition for I2C communication
//Send off the commands
ftStatus = FT_Write(ftHandle, OutputBuffer, dwNumBytesToSend, &dwNumBytesSent);
dwNumBytesToSend = 0;                             //Clear output buffer
```

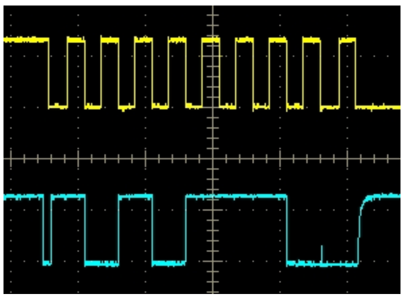
Fig 8-1 Sending control byte 0xAE

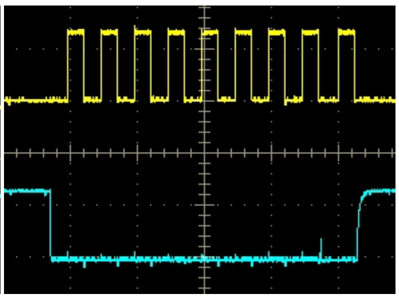
Fig 8-2 Sending high address byte 0x00

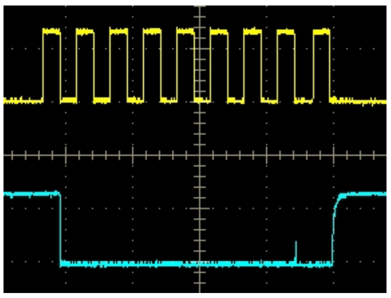
Fig 8-3 Sending low address byte 0x80

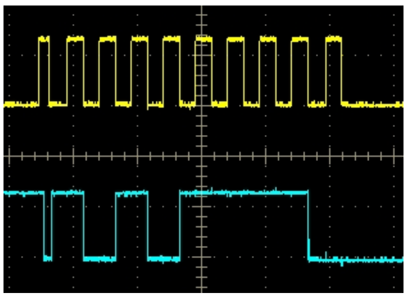
Fig 8-4 Sending control byte 0xAF

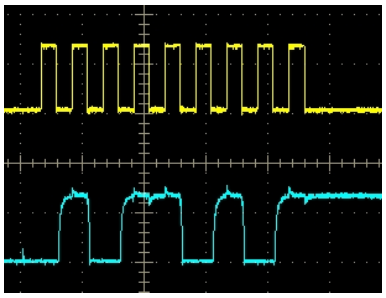
Fig 8-5 Read data byte 0x5A

Figure 8 Waveforms for EEPROM random address read

2.4 Using Channel B Requirements

If it is necessary to use Channel B of FT2232H MPSSE interface to connect with I^{2}C-bus 24LC256 device, the following changes are necessary:

- Re-map connection pins from ADBUS 0,1,2,4 to BDBUS 0,1,2,4 accordingly (refer to Table 1).
- Open the MPSSE port with device index and serial number especial for Channel B and get according handle.

3 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 8791 3576

E-mail (Sales)	tw.sales1@ftdichip.com
E-mail (Support)	tw.support1@ftdichip.com
E-mail (General Enquiries)	tw.admin1@ftdichip.com

Branch Office – Shanghai, China

Future Technology Devices International Limited (China)
Room 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

Distributor and Sales Representatives

Please visit the Sales Network page of the FTDI Web site for the contact details of our distributor(s) and sales representative(s) in your country.

Appendix A - References

Document References

AN_108_Command_Processor_for_MPSSE_and_MCU_Host_Bus_Emulation_Modes

AN2232C-02_FT2232CBitMode

D2XX Programmer's Guide

Datasheet for FT2232H V202

Datasheet for Microchip 24LC256 – 2K I2C Serial EEPROM

Acronyms and Abbreviations

Terms	Description
EEPROM	Electrically Erasable Programmable Read-Only Memory
FIFO	First In First Out
FTDI	Future Technology Devices International
IC	Integrated Circuit
I^2C	Inter-Integrated Circuit
JTAG	Joint Test Action Group
MPSSE	Multi-Protocol Synchronous Serial Engine
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

Appendix B - List of Figures and Tables

List of Figures

List of Tables

Appendix C - Revision History

Document Title:	AN_113 Interfacing FT2232H Hi-Speed Devices to I2C Bus
Document Reference No.:	FT_000137
Clearance No.:	FTDI# 90
Product Page:	http://www.ftdichip.com/FTProducts.htm
Document Feedback:	Send Feedback

Revision	Changes	Date
Version 1.0	First Release	2009-05-08
Version 1.1	Updated section 2.4 to remove third bullet point as this step was not required when changing from Channel A to Channel B	2011-02-25
Version 1.2	Corrected command used to enable 3-phase-clocking on page 8. Correct command is 0x8C instead of 0x8D.	2020-05-26