Application Note

AN_438

FT260 I2C Example in C#

Version 1.0

Issue Date: 2020-05-12

This document demonstrates using the FT260 as a USB-I^{2}C Master interface to read data from an I^{2}C sensor. The application is written in C# and provides an example of importing and using a subset of the LibFT260 functions in C# GUI applications. The overall application is similar to the USB Power Meter shown in AN_355 but uses the FT260 in place of the FT232H.

Bridgetek Pte Ltd (BRTChip)
178 Paya Lebar Road, #07-03 Singapore 409030
Tel : +65 6547 4827 Fax : +65 6841 6071
Web Site: http://brtchip.com

Table of Contents

1 Introduction

1.1 Overview

The FT260 device offers a versatile bridging solution from USB to I²C Master, UART and GPIO. One popular application is to read data from one or more I²C sensors.

As a HID class device, it is possible to use HID requests directly from the application. FTDI also provide a library libFT260 which can be used from C++ applications and provides a set of functions allowing the FT260 to be easily integrated into the application with familiar functions such as FT260_I2CMaster_Read.

C# is a popular language, especially where a Windows Graphical User Interface (GUI) based interface is desired, and so this application note provides a simple example of calling the functions in libFT260 from a C# application in order to read values from a sensor and display them on a GUI. The application shown here is a small current and voltage measurement device which is also useful to have when debugging USB hardware designs or other low voltage DC circuits. It can be modified to support a range of other sensors and applications.

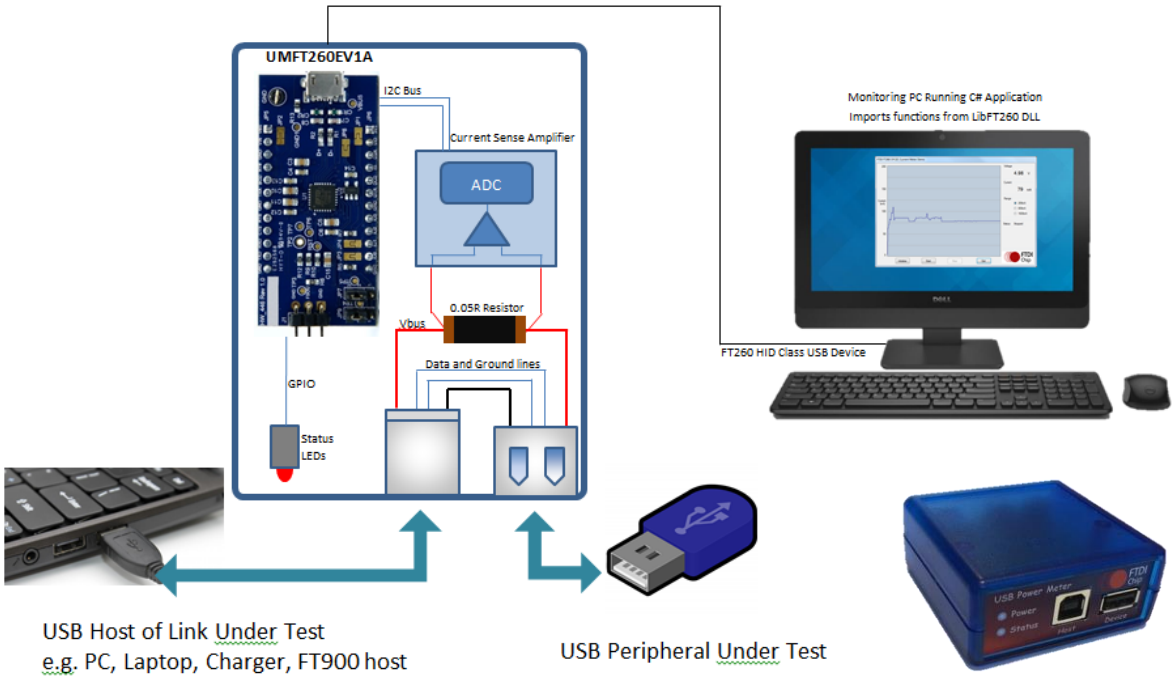


Figure 1 - USB Power Meter connections

Note: The information provided in this document may be subject to change in future. Refer to the latest versions of the datasheets and documentation for all components and libraries used in your design for the latest information. If any content in this application note conflicts with the device documentation, the manufacturer's device documentation should take priority. FTDI are not responsible for the specifications of any third-party devices used here, always consult the latest documentation from the manufacturer as these devices may be subject to change out with the control of FTDI.

2 Hardware

The hardware was constructed using the UMFT260EV1A evaluation module acting as a USB to I²C bridge. The remainder of the circuit consists of the current measurement circuit based around the Texas Instruments INA219, an additional LDO to power the INA219 and some LEDs to indicate status.

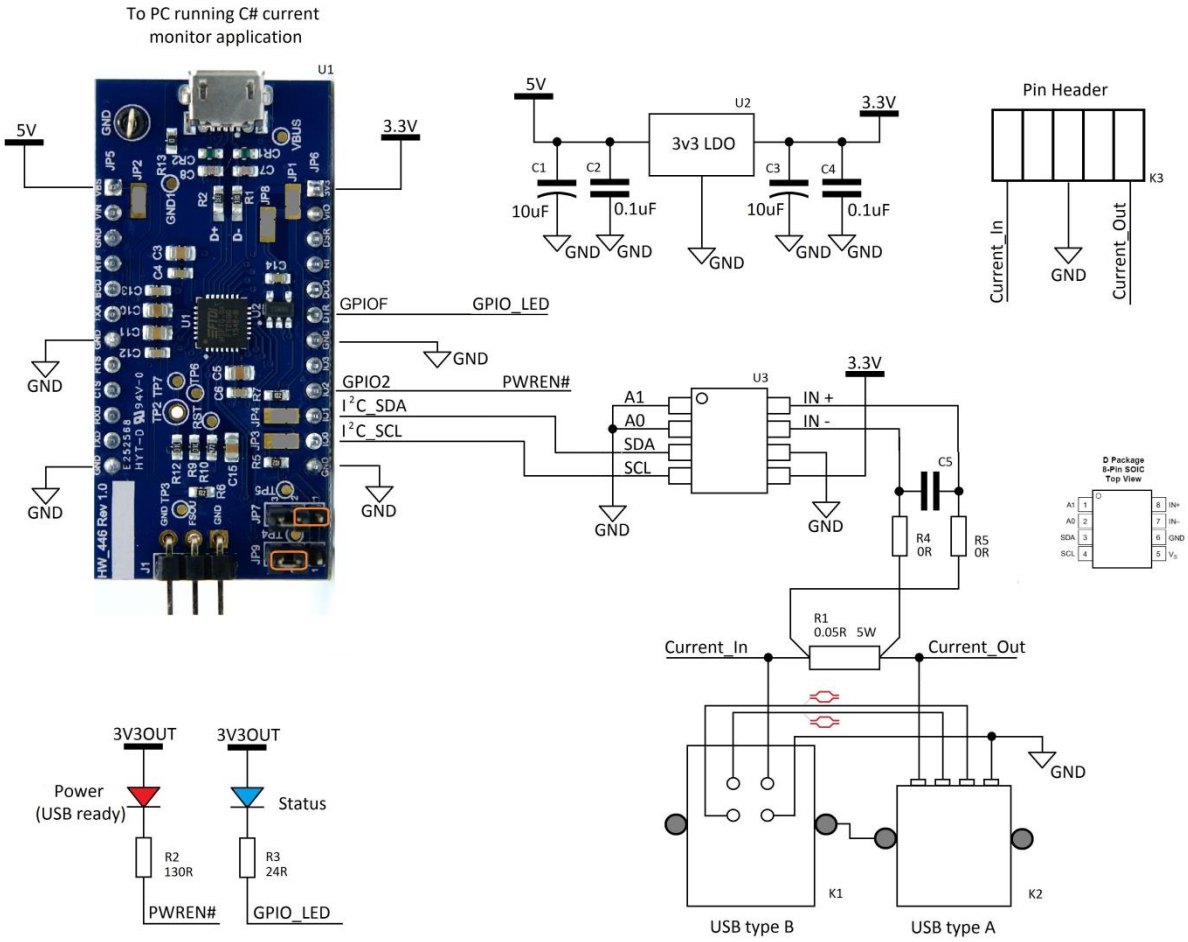


Figure 2 - Schematic

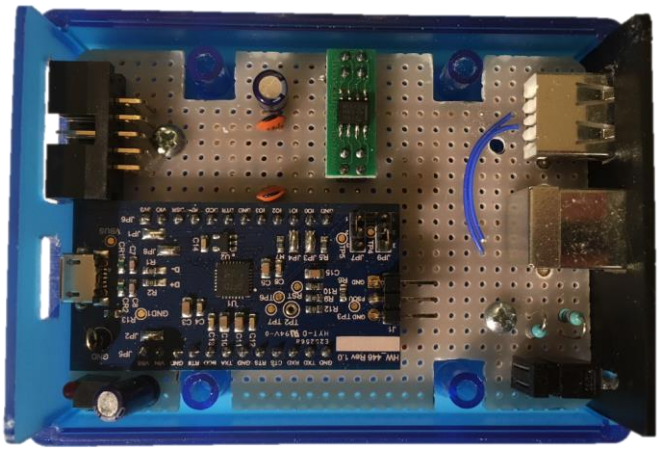


Figure 3 - Prototype hardware unit

2.1 UMFT260EV1A acting as USB to I2C master

The UMFT260EV1A module provides all of the necessary hardware needed to use the FT260 including an external EEPROM fitted on the module PCB for configuration data and the USB connector.

This module has a dual-in-line footprint which can be used with breadboards and prototyping boards making it easy to get started in evaluating and developing with the FT260.

The FT260 IC itself is available in TSSOP and QFN IC packages allowing a compact PCB for a final product design.

The module has a pair of jumpers to select the mode of the FT260, allowing UART + I^2C, UART only or I^2C only. In this case the I^2C only mode was selected by setting the jumpers as follows:

- DCNF0 high Jumper pins 1-2 on JP7
- DCNF1 low Jumper pins 2-3 on JP9

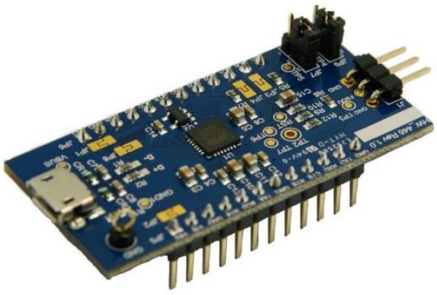


Figure 4 - UMFT260EV1A evaluation module

2.2 Connections

The example hardware includes several USB connectors.

The USB connector on the rear panel is for the FT260 itself, allowing the computer running the C# application to control the FT260 and take readings.

The front panel has two USB connectors which are effectively daisy chained together, passing GND, USB Data + and USB Data - directly through. The Vbus is also connected through with the small value sense resistor in the path. These 2 connectors allow the device to be connected in the middle of the link being tested and to be almost transparent to the link under test.

A box header on the rear panel also allows access to the current in, current out and ground lines via jumper wires.

2.3 Current Sensing

In this example and many other similar user cases, it is preferred to insert a resistor in the 5V Vbus line of the connection under test to measure the current instead of instead of inserting a resistor in the ground line. The latter case can result in the measurement equipment causing ground differences in the circuit under test and the accuracy can also be affected if the circuit under test has other paths to ground. The resistor in the Vbus line will still cause a small drop in voltage but this is often still preferable to inserting the resistor in the ground line. However, measuring the current on the Vbus line (high side current measurement) is significantly more complex than ground referenced measurements and requires significantly more complex analog circuit or a specialized IC.

The INA219 used here is a high-side current shunt monitor device which allows the Vbus current to be measured. It measures the small voltage difference across a low-value current shunt resistor in

order to measure the current flowing and converts this to a ground referenced voltage and then reads this via an ADC. A second ADC channel also converts the voltage present on the downstream side of the shunt resistor. Lines A0 and A1 select the I^2C address of the device.

There are a variety of devices available which can perform high side current measurement at various current levels and accuracies and with different values of sense resistor. The INA219 used here also has various internal registers which can be used to configure the range and features.

2.4 LDO Voltage Regulator

A small 3v3 LDO was used in this case to provide a stable voltage to the INA219 as the UMFT260EV1A's 3v3 out was used to power the blinking status LED etc. However, in many cases the circuit could be simplified even further by powering a low-current sensor from the 3v3 output of the module. The FT260 datasheet has details of the output current capabilities etc.

2.5 FT_Prog Configuration

The FT260 has various configuration options available via its internal one-time-programmable eFUSE memory or an external EEPROM (which shares the I^2C lines and is read on start-up if present). In many cases however, the device's built-in defaults can be used without any programming

Like other FTDI bridging solutions, the settings can be configured if required over the USB interface by connecting the module to a PC and running the free FT_Prog software provided by FTDI.

Programming of the external EEPROM requires no additional hardware, however, in order to program the eFUSE, the FT260 requires an additional programming voltage (3.8V) on its FSOURCE pin. The programming board, UMFTPD3A, provides a connection bridge between the FT260 and a USB host for supplying the power source, for timing control of the eFUSE, and also for communicating with the programming utility FT_Prog. Further details can be found in the UMFTPD3A datasheet.

It is advised to provide an external EEPROM IC (or the footprint for this to be added) where possible. This is especially true during development where the settings may need to be changed several times which would not be possible with the one-time programmable eFUSE.

In this example, the default settings are suitable. The GPIO2 line may be set as PWREN# optionally to indicate enumeration and suspend status. The description may also be changed to identify the device, for example "FT260 Current Meter" in this case.

3 Application

The software was written in C# to provide a GUI-based interface. DLL Imports were used to import the C++ functions provided in the libFT260 library. The source code project can be downloaded using the link in Appendix A – References.

3.1 Main Window

The main window of the application is shown in Figure 5. The application has the following controls on its main window:

- Rolling chart showing the most recent 500 current consumption measurements
- Numeric readouts of the voltage and current measured in the most recent sample
- Radio buttons to scale the data allowing maximum ranges of 200mA, 800mA and 1600mA
- Buttons for initializing, starting and stopping measurements.

This trace below shows an optical mouse which consumes approximately 6mA in a low power state with its LED dimmed when idle. The LED current increases when the mouse is moved as the LED goes to full brightness for accurate position tracking.

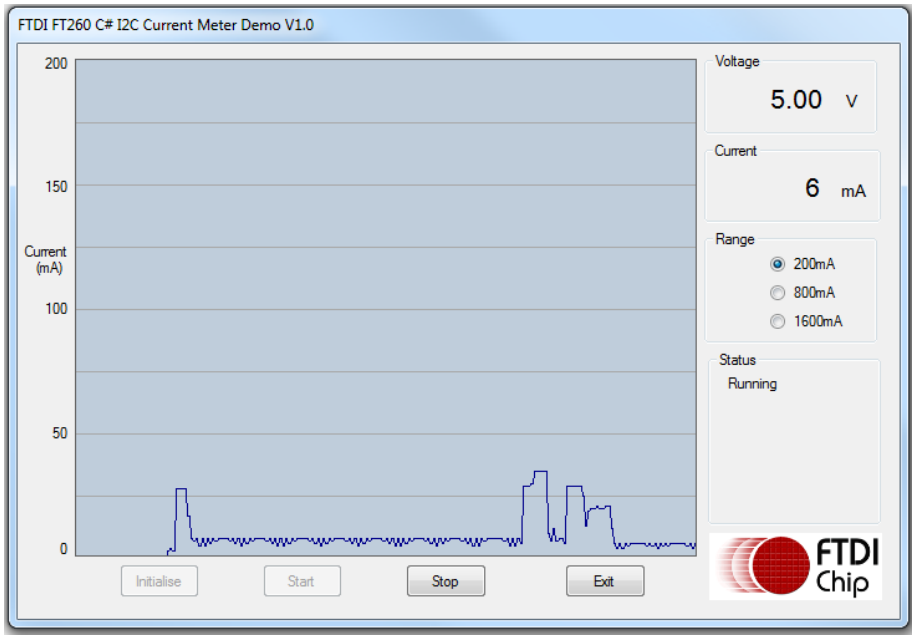


Figure 5 - Main window of application

3.2 Event Handlers

The software is based around the handling of the events resulting from clicking the buttons on the main window. The buttons are enabled or disabled as required to gate the flow of the application. For example, when the application starts, only the Initialize and Exit buttons are enabled.

This sequence is triggered when the Initialise button is clicked by the user. It checks for the presence of the libFT260 DLL using a try-catch and if present, opens the FT260 and configures the INA219 so that it is ready to take measurements.

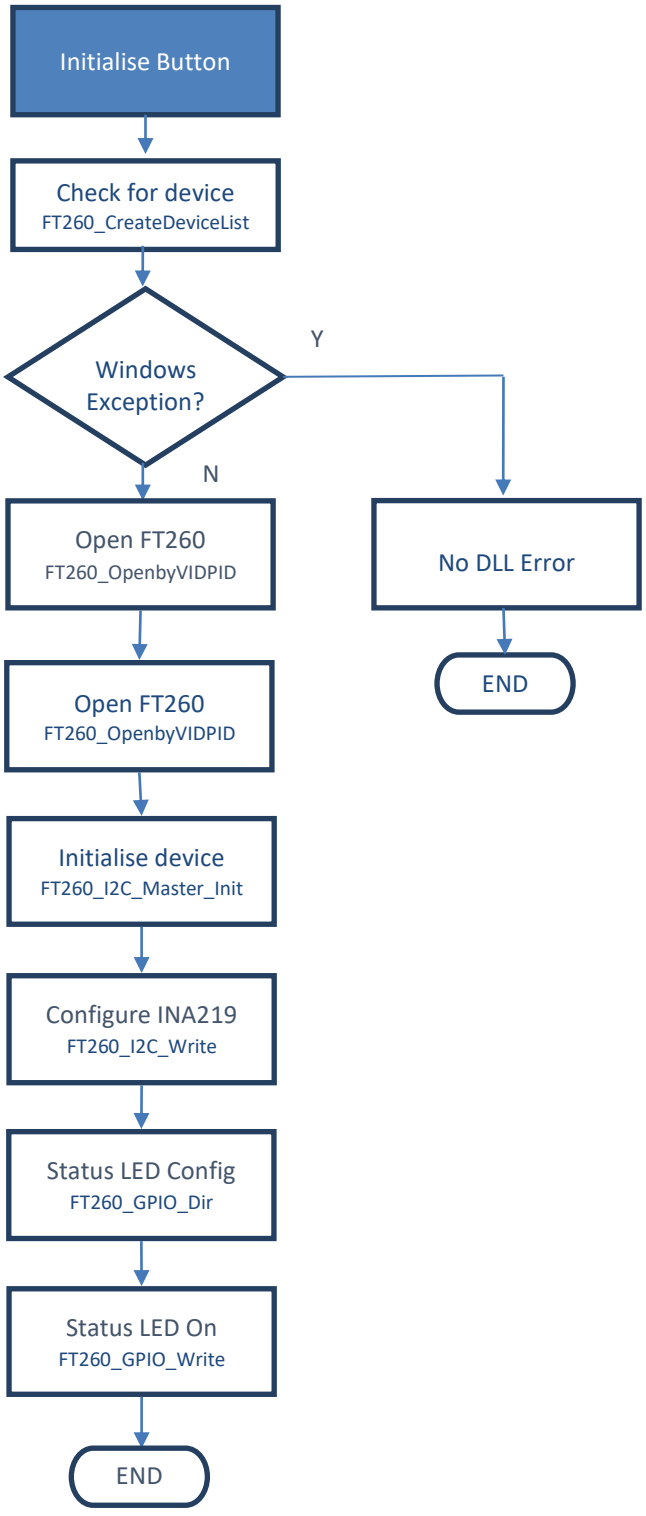


Figure 6 - Flowchart of Initialisation (INIT button)

This sequence is triggered when the Run button is clicked by the user. It runs in a loop, taking measurements continuously from the INA219. A flag 'Running' is set true and the loop runs until this flag is set false by the Stop button.

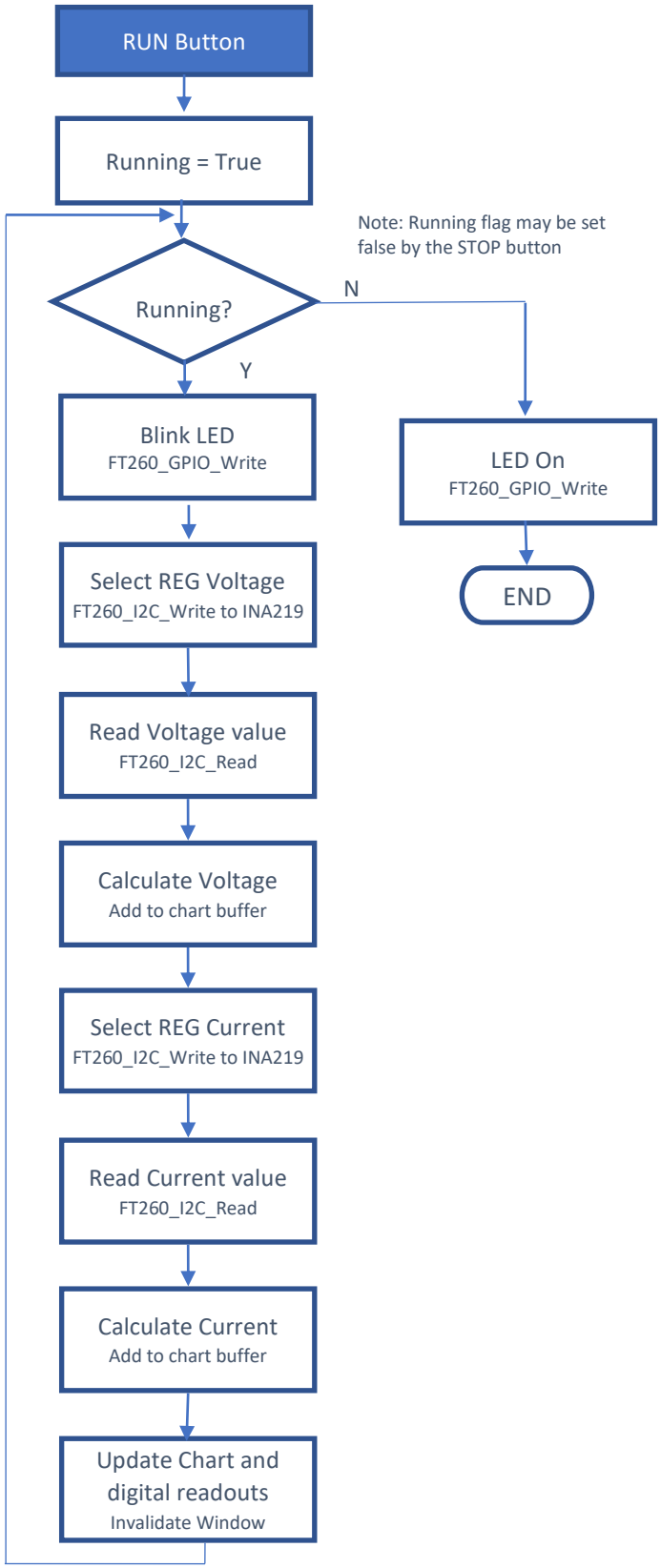


Figure 7 - Flowchart of main measurement loop (RUN button)

This sequence is triggered when the Stop button is clicked by the user. It sets the 'running' flag to false which causes the measurement loop to stop running.

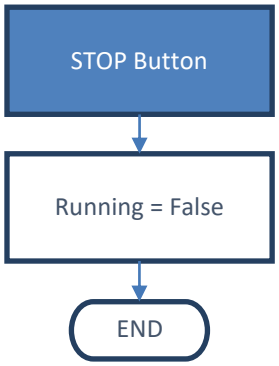


Figure 8 - Flowchart of stopping logging (STOP button)

This sequence is triggered when the Exit button is clicked by the user. If the DLL is present, it closes the handle to the FT260. If the DLL was not present, the application is just closed to avoid a Windows exception.

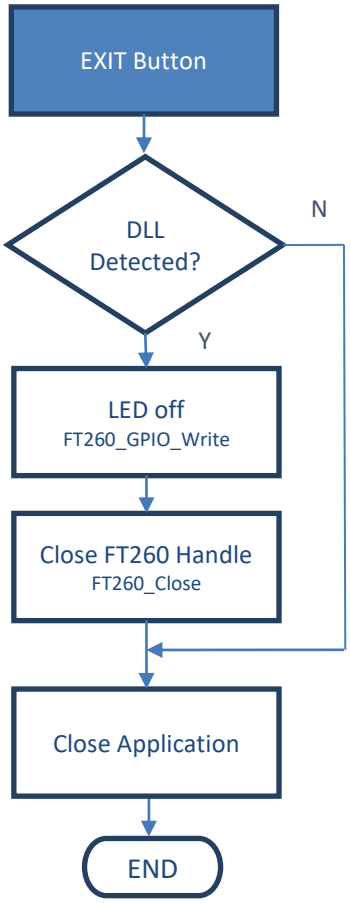


Figure 9 - Flowchart of closing device (EXIT button)

3.3 Chart Code

The application uses a single buffer to store the 500 measurement points (corresponding to the width of the chart) and to plot the resulting chart. The buffer is implemented in software as a circular buffer with write and read pointer.

Adding current values

On the input side, the main measurement loop takes each new current reading from the INA219 and adds it to the next place in the buffer, overwriting the oldest data point. The variable GraphInputPointer is used as the input pointer which is incremented as each new value is added. The buffer therefore contains the most recent 500 data points. The current is stored as a value in milliamps.

Drawing the Chart

The chart is drawn within a PictureBox object. This is re-painted whenever a picturebox1.paint event occurs which can be the loading of the form or when manually trigged by the program loop when it adds a new data point. The entire chart is re-drawn including the scale values such that any change in the scale selected by the user is applied immediately.

After plotting the scales and tick marks etc., the actual chart line is drawn. Chart data is taken directly from the same circular buffer used to store the 500 data points by the routine which reads from the INA219.

The plotting begins at x = 0 with the *oldest* data point (i.e. from index GraphInputPointer + 1) and continues until x = 499 (which will contain the latest data point measured). Each time the chart is updated, the oldest value will be at the left hand side and the newest value will be at the right-hand side, causing the chart to scroll left as new points are added.

The data is scaled depending on the range selected by the user so that the selected range will fit entirely in the 400 pixels height of the chart.

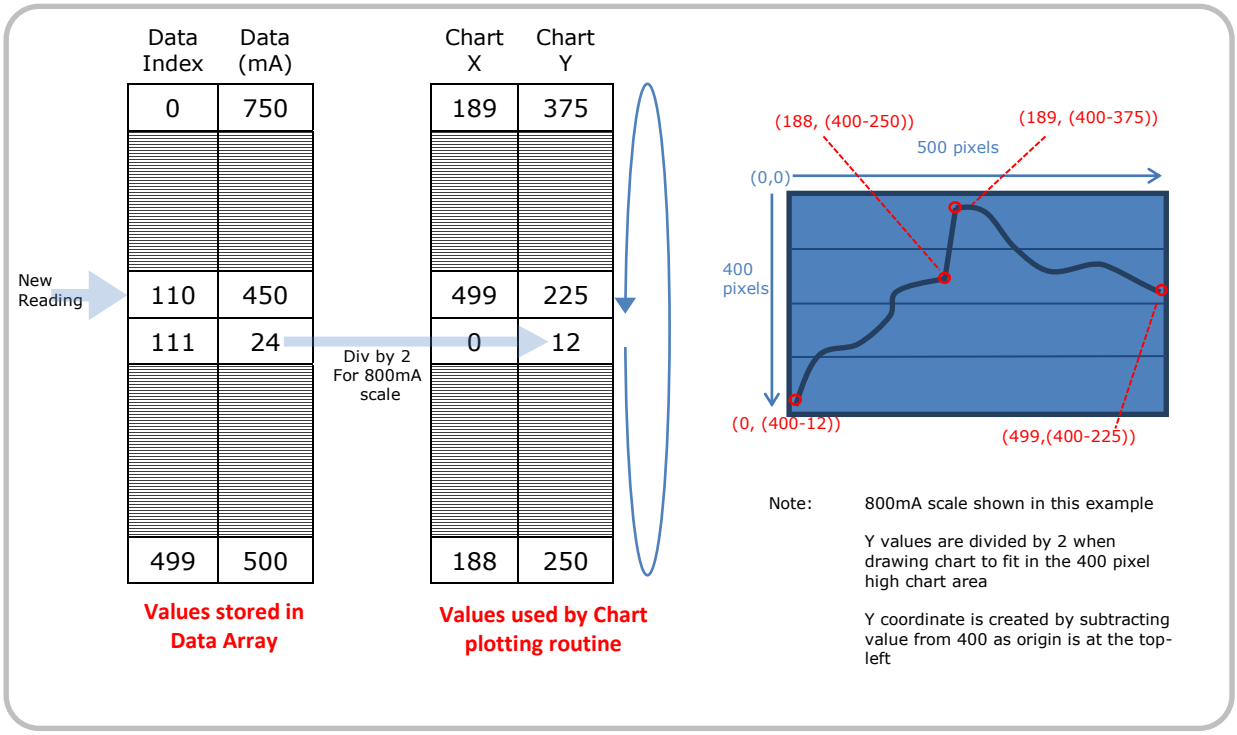


Figure 10 - Chart plotting

If the value exceeds the range set, the y value is truncated at 399 and the corresponding pen colour is set to red to indicate the exceedance. The numerical readout will continue to read the correct value up to 1600mA even if the chart range is set for 200mA etc.

Coordinates of a picture box are taken from the top down, and so each Y coordinate is adjusted by Y = (400 – Y) so that they are now targeted to the bottom of the window with positive Y in the up direction.

The chart is then created by drawing a series of lines between pairs of points to create a single chart line.

4 Library

4.1 libFT260

The libFT260 DLL is provided to make it easy to use the I²C, UART and GPIO features of the FT260 without needing to handle the low-level USB HID requests.

The library is provided for C++ applications but in this case the functions were imported as described in the next section to allow use in a C# graphical user interface.
The latest version can be downloaded from the FT260 product page along with the accompanying programming guide.

LibFT260 release 1.1.2 now includes a 64-bit DLL in addition to the earlier 32-bit DLL. The updated sample code can now be built for both 32-bit and 64-bit. The respective DLLs have been copied into the Debug and Release folders in the supplied code. The build configuration can be set for the required combination of Release/Debug and x86/x64 using the drop-downs shown below.

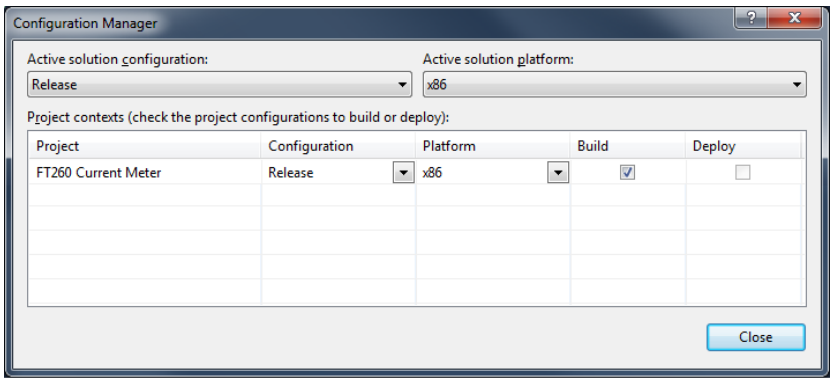


Figure 11 - Build Configurations

4.2 C# Imports

The libFT260 library is provided as a C++ based library and so the functions must be imported in order to use with C#.

A subset of the available DLL functions were imported as necessary for this application example but the other functions could also be imported in a similar way. The function prototypes and associated data types can be found in the header file provided with libFT260.
An example of the import for I²C master Read is shown below:

```
// LIBFT260_API FT260_STATUS WINAPI FT260_I2CMaster_Read(FT260_HANDLE ft260Handle, uint8 deviceAddress, FT260_I2C_FLAG flag, LPVOID lpBuffer, DWORD dwBytesToRead, LPDWORD lpdwBytesReturned, timeout);
[DllImport("LibFT260.dll")]

private static extern FT260_STATUS FT260_I2CMaster_Read(IntPtr ft260Handle, uint deviceAddress, FT260_I2C_FLAG flag, ref byte lpBuffer, UInt32 dwBytesToRead, ref ushort lpdwBytesReturned, UInt32 Timeout);
```

4.3 Using the Imported Functions

This section provides an overview of how the libFT260 functions are used to implement the I²C communication detailed in the I²C peripherals datasheet.

4.3.1 Opening and Initializing

Initially, a call to FT260_CreateDeviceList is used to check how many FT260s are present. In addition, this is placed in a try-catch as it will throw a Windows exception should the LibFT260 DLL not be present (by default expected in the application folder). This allows the application to end gracefully if the DLL is not present rather than crashing.

```
try
{
    Status = FT260_CreateDeviceList(ref NumDev);
}
catch (DllNotFoundException)
{
    SetControls_Error("No DLL Found", "The DLL is not present in the application folder");
    return 1;
}
catch
{
    SetControls_Error_AllowReInit("Stopped", "Capture stopped: Click Initialise and the
Start to begin again");
    return 1;
}
if (Status != FT260_STATUS.FT260_OK)
{
    SetControls_Error("Error", "Please check hardware and re-start application");
    return 1;
}
DLL_Loaded = true;
```

The FT260 is opened by a call to FT_OpenByVidPid which returns a handle which can be used to reference the device thereafter.

A call to FT260_I2CMaster_Init is used to configure the I^2C master in the FT260.
The application can be extended to open a particular FT260 etc. in this part of the code.

4.3.2 Writing and Reading

After opening the device, this example primarily uses the FT260_I2CMaster_Write, FT260_I2CMaster_Read and FT260_GPIO_Write functions during the main application loop.

The INA219 has six registers which provide control settings and results.

POINTER ADDRESS	REGISTER NAME	FUNCTION	POWER-ON RESET		TYPE(1)
HEX			BINARY	HEX	
00	Configuration	All-register reset, settings for bus voltage range, PGA Gain, ADC resolution/averaging.	00111001 10011111	399F	R/$\overline{W}$
01	Shunt voltage	Shunt voltage measurement data.	Shunt voltage	—	R
02	Bus voltage	Bus voltage measurement data.	Bus voltage	—	R
03	Power(2)	Power measurement data.	00000000 00000000	0000	R
04	Current(2)	Contains the value of the current flowing through the shunt resistor.	00000000 00000000	0000	R
05	Calibration	Sets full-scale range and LSB of current and power measurements. Overall system calibration.	00000000 00000000	0000	R/$\overline{W}$

(1) Type: **R** = Read only, **R/$\overline{W}$** = Read/Write.
(2) The Power register and Current register default to 0 because the Calibration register defaults to 0, yielding a zero current value until the Calibration register is programmed.

Figure 12 - INA219 Registers

The waveforms for writing and reading are shown below (please consult the INA219 datasheet for more information). Many different I²C peripherals use a similar scheme.

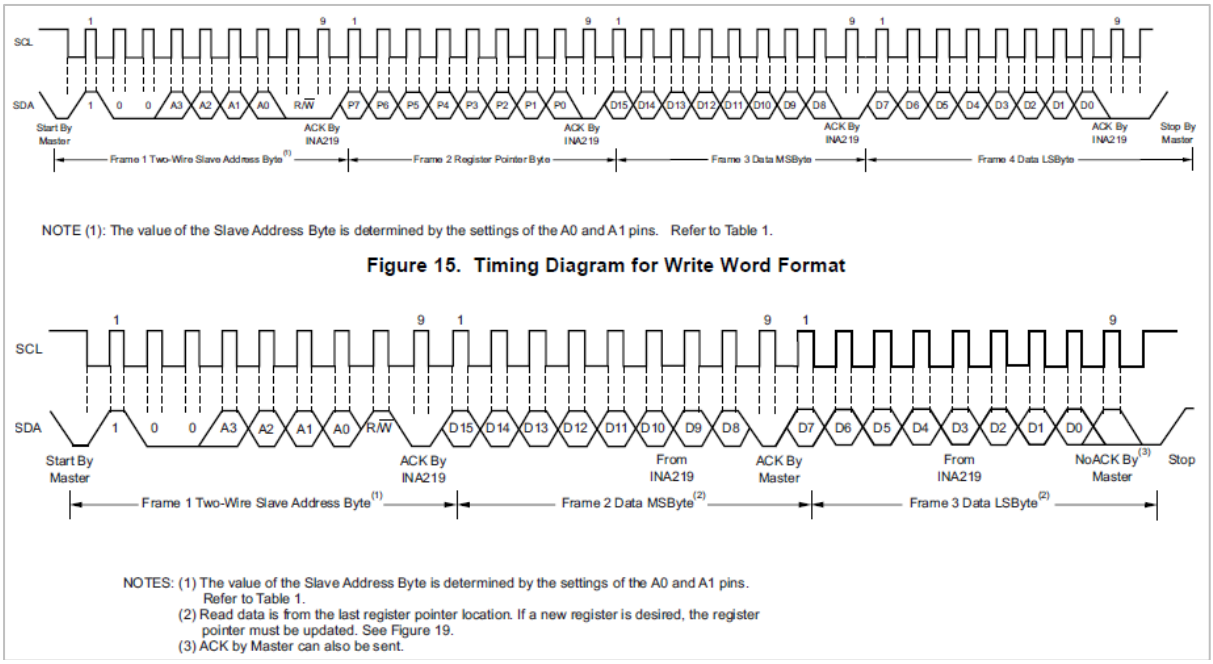


Figure 15. Timing Diagram for Write Word Format

Figure 13 - INA219 Write and Read sequences

During initialization of the application, a write is performed to the INA219's configuration register. Here, an array is created containing the register index within the INA219 (the configuration register has index 0) followed by the values to be written (0x0EEF). A write is then performed to the I²C address of the INA219 (0x40). The three bytes from the array are written. This results in the same sequence of I2C_Address -> Reg_address -> Data_MSbyte -> Data_LSbyte shown in the upper waveform in Figure 13.

```
byte[] writeDataConfig = new byte[10];

writeDataConfig[0] = 0x00;
writeDataConfig[1] = 0x0E;
writeDataConfig[2] = 0xEF;

int size = Marshal.SizeOf(writeDataConfig[0]) * writeDataConfig.Length;
IntPtr pnt = Marshal.AllocHGlobal(size);
Marshal.Copy(writeDataConfig, 0, pnt, writeDataConfig.Length);

numBytesToWrite = 3;

Status = FT260_I2CMaster_Write(ft260handle, 0x40, FT260_I2C_FLAG.FT260_I2C_START_AND_STOP,
pnt, numBytesToWrite, ref writeLength);

if ((Status != FT260_STATUS.FT260_OK) || (writeLength != numBytesToWrite))
{
    SetControls_Error("Error", "Please check hardware and re-start application");
    return 1;
}
```

In the main application loop, the voltage and current registers are read alternately in a continuous loop. The voltage and current reading use the same procedure and the voltage is used as an example here.

First, the register to be read must be specified as a read operation will read data from whichever register was last selected. A write operation is performed to the address 0x40 with the value 0x02 to select the Voltage result register. No data bytes are needed if the register is just being selected and so only the first two bytes of the Write operation from Figure 13 are sent followed by the I²C Stop.

```
byte[] VoltageAddressReg = { 0x02 };
numBytesToWrite = 1;
Marshal.Copy(VoltageAddressReg, 0, pnt, VoltageAddressReg.Length);

Status = FT260_I2CMaster_Write(ft260handle, 0x40, FT260_I2C_FLAG.FT260_I2C_START_AND_STOP,
pnt, numBytesToWrite, ref writeLength);

if ((Status != FT260_STATUS.FT260_OK) || (writeLength != numBytesToWrite))
{
    SetControls_Error("Error", "Please check hardware and re-start application");
    return;
}
```

A read is then performed as shown below. This will read two bytes from address 0x40 (the INA219's I²C address) from register index 0x02 set above (Voltage result). A timeout value of 5 seconds is specified which will ensure the function returns in the event of an error in reading the bytes. The data will be put into buffer Reading[]. The status value is checked to ensure that no errors occurred and the readLength parameter confirms how many bytes were read.

```
byte[] Reading = new byte[100];
numBytesToRead = 2;

Status = FT260_I2CMaster_Read(ft260handle, 0x40, FT260_I2C_FLAG.FT260_I2C_START_AND_STOP,
pnt, numBytesToRead, ref readLength, 5000);

if (Status == FT260_STATUS.FT260_OTHER_ERROR)
{
    SetControls_Error("Error", "Please check hardware and re-start application");
    return;
}

if (Status == FT260_STATUS.FT260_I2C_READ_FAIL)
{
    SetControls_Error_AllowReInit("Stopped", "Capture stopped: Click Initialise and the
Start to begin again");
    return;
}

if ((Status != FT260_STATUS.FT260_OK) || (readLength != numBytesToRead))
{
    SetControls_Error("Error", "Please check hardware and re-start application");
    return;
}
```

The final value is obtained by combining the two bytes into their original 16-bit value.

```
Marshal.Copy(pnt,Reading,0,(int)numBytesToRead);

// Calculate V in Volts
VoltageHighByte = Reading[0] & 0x7F;
VoltageLowByte = Reading[1] & 0xF8;

Voltage = (((VoltageHighByte * 256) + VoltageLowByte) / 2);      // in Volts
```

The same process is repeated to read current values but this time setting the register index to 0x01 for the current result.

The INA219 allows a read to be performed to a register which was already selected whereas some sensors may require a write of the register index followed by a read to be performed in the same I^2C transaction with a repeat start between. One reason for using the separate write and read transaction here is that the INA219 has a timeout which can trigger if the bus is idle in the middle of a transaction for more than 28ms and which is included for SMbus usage and so separating the write and read avoid the risk of timing out. This is not expected to time out due to the 1ms frame rate of the USB link however which is well below the 28ms timeout.

4.3.3 GPIO Operations

GPIO writes are used to blink the status LED. The pin direction can be set during application initialization, specifying the pin to be configured and the direction as shown below.

```
Status = FT260_GPIO_SetDir(ft260handle, FT260_GPIO.FT260_GPIO_F, 1);     // STATUS LED

if (Status != FT260_STATUS.FT260_OK)
{
    SetControls_Error("Error", "Please check hardware and re-start application");
    return 1;
}
```

The pin can then be written using the GPIO_Write call as shown below.

```
Status = FT260_GPIO_Write(ft260handle, FT260_GPIO.FT260_GPIO_F, 0);
if (Status != FT260_STATUS.FT260_OK)
{
    SetControls_Error("Error", "Please check hardware and re-start application");
    return;
}
```

Note: it is necessary to check the status retuned by all calls to the FT260 functions and to check any other returned parameters (e.g. BytesReturned which is returned by reference in a read operation) to ensure that the function has completed successfully.

5 Using the Demo

Connect the FT260's USB port to the PC to be used for taking the measurements. The PC will detect a HID class device and will install the HID driver provided with Windows. No additional driver is required to be loaded. The device will show up as a HID-Compliant Device in Device Manager (see Human Interface Devices section of Device Manager). The red power LED will be illuminated.

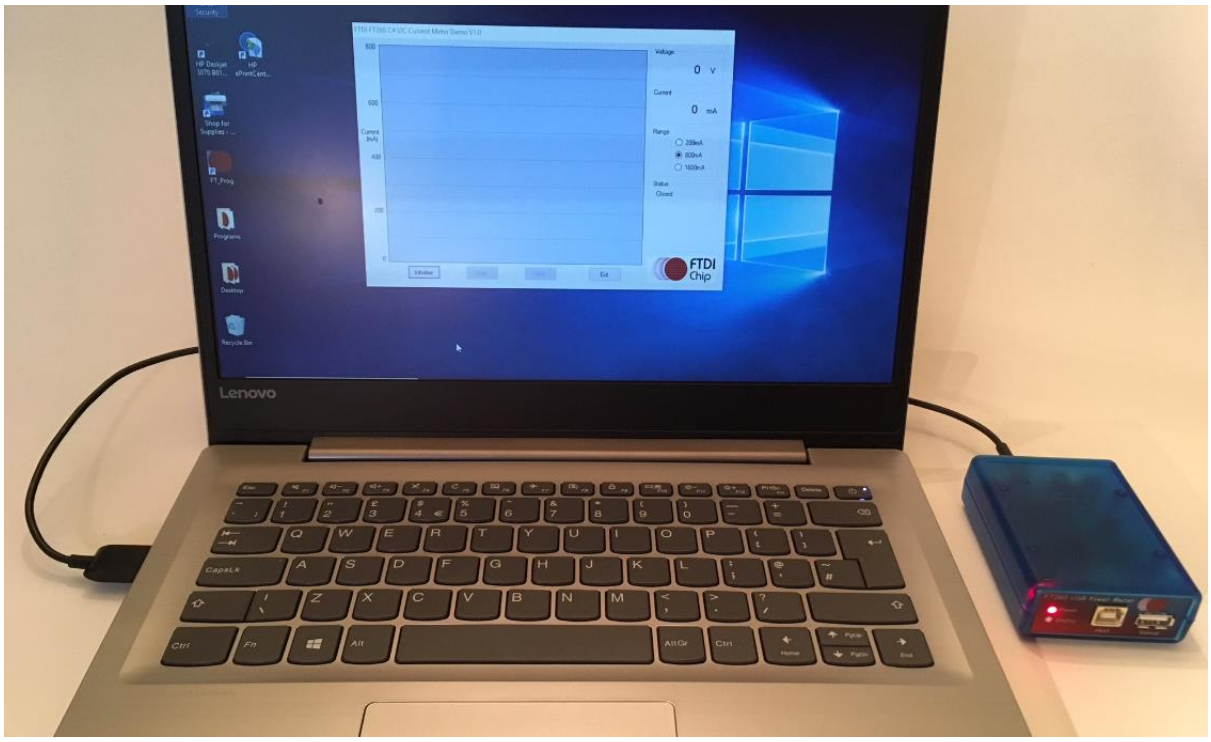
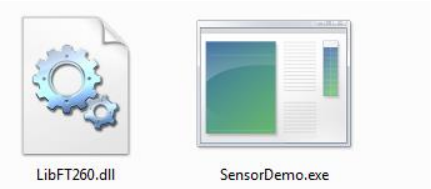


Figure 14 - Connecting the FT260 to the host PC running the application

Open the demo application provided. This can be done by opening the .sln file provided in Visual Studio 2013 or later and running the application from there, or by running the executable from the release folder of the provided file. If running the exe directly, the DLL should be located in the same folder as the final application.



In the application, click the Initialise button and the device will be configured and the Start button will become enabled. The blue Status LED will illuminate to indicate that it is ready. If the hardware isn't connected and enumerated, or if the DLL is missing, an error will show in the status window.

Click the Start button and the device will begin taking measurements. The blue Status LED will blink to indicate readings being taken.

Connect the Upstream port of the meter (USB type B, marked "Host") to the host of the link being measured. For example, to the USB host port of a PC which is to host the device under test (this can be the same PC as used for monitoring). A short cable is recommended to minimize any losses in the cable. The voltage indication will now reflect the voltage of the upstream host (normally 5V). Now, connect a USB peripheral to the downstream port of the meter (USB type A, marked 'Device'). The device should enumerate as if connected directly to the host, as the meter passes

the lines through between upstream and downstream ports on its front panel. The current will be indicated on the rolling chart and the numerical readouts.

Figure 15 - Connecting the meter to the link under test

The screenshot shows the USB Flash drive being connected. The current varies as the device enumerates and is enabled. A peak in current occurs when a file is opened on the disk.

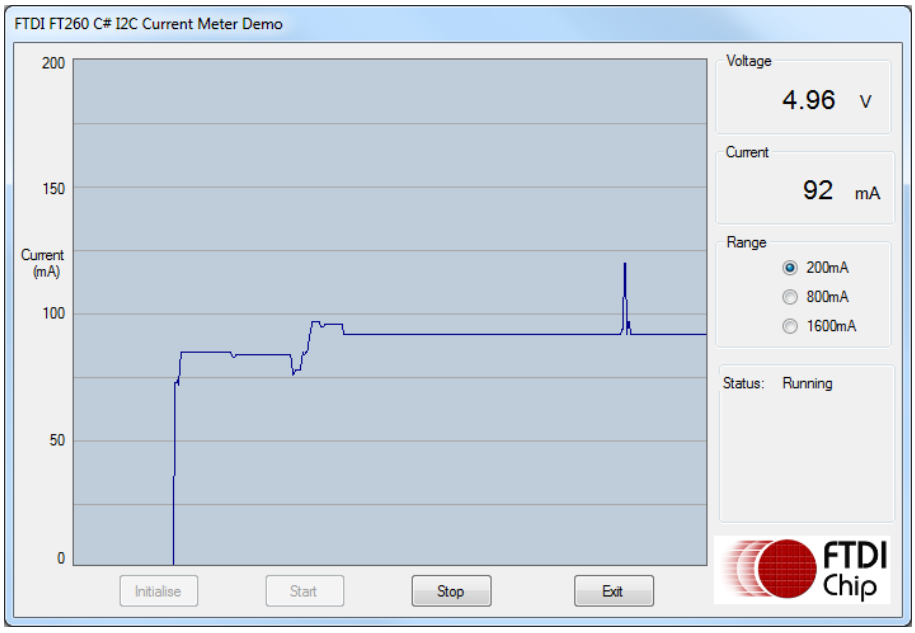


Figure 16 - Current consumed by a Flash drive

The range buttons can be used to select the most suitable current range whilst the application is running. If the current exceeds the maximum value of the range currently set, the associated points will appear red to indicate this. This is demonstrated below:

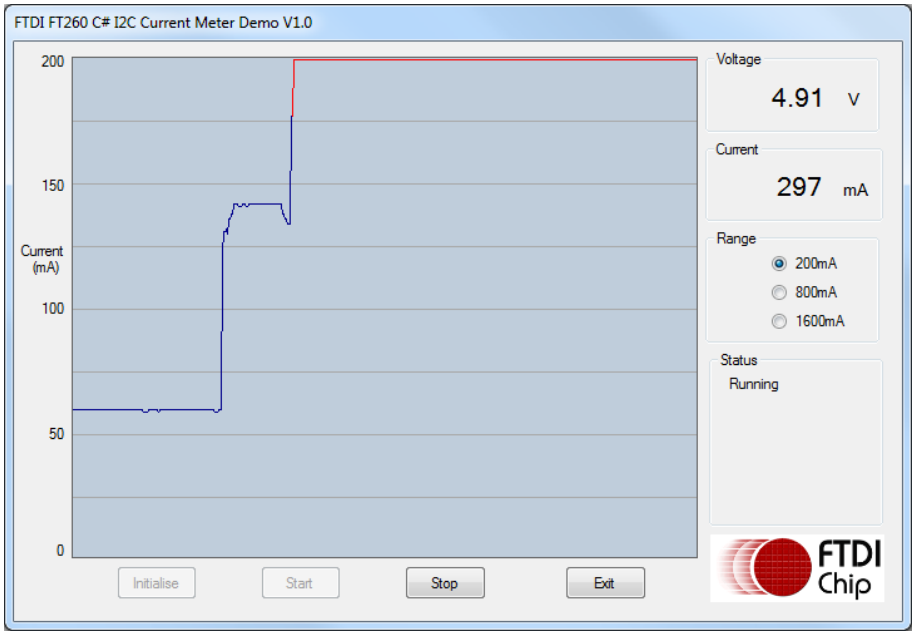


Figure 17 - A high current device exceeds the 200mA range after enumeration

The technical specifications of the current measurement side of the demo are defined by the INA219 and its accompanying components (e.g. sense resistor) and can be found in the INA219 datasheet.

6 Conclusion

This application note has presented an example of how to use the FT260 device as a USB- I^2C bridge in a C# graphical user interface application.

The power meter application is a useful tool to have whilst debugging a variety of low voltage low current DC devices and especially USB devices, where current consumption can provide a good insight into what a device is doing and whether it is working as expected.

The application can be modified and extended to use a variety of different I^2C sensors and peripherals including ADCs, temperature sensors, LED controller ICs, GPIO expanders, MCUs with I^2C Slave ports etc. and can have several different sensors connected on the same I^2C bus provided that they have different I^2C addresses.

Note: The hardware and software examples provided in this package (document and accompanying code sample) are intended as a starting point for developing your own application rather than a finished product. By using any information contained in this package, the user agrees to take full responsibility for ensuring that their final product meets all operational and safety requirements and for any consequences resulting from its use.

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

Branch Office – Tigard, Oregon, USA
Future Technology Devices International Limited
(USA)
7130 SW Fir Loop
Tigard, OR 97223-8160
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales) us.sales@ftdichip.com
E-Mail (Support) us.support@ftdichip.com
E-Mail (General Enquiries) us.admin@ftdichip.com

Branch Office – Taipei, Taiwan
Future Technology Devices International Limited
(Taiwan)
2F, No. 516, Sec. 1, NeiHu Road
Taipei 114
Taiwan , R.O.C.
Tel: +886 (0) 2 8797 1330
Fax: +886 (0) 2 8751 9737

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

Branch Office – Shanghai, China
Future Technology Devices International Limited
(China)
Room 1103, No. 666 West Huaihai Road,
Shanghai, 200052
China
Tel: +86 21 62351596
Fax: +86 21 62351595

E-mail (Sales) cn.sales@ftdichip.com
E-mail (Support) cn.support@ftdichip.com
E-mail (General Enquiries) cn.admin@ftdichip.com

Web Site
http://ftdichip.com

Distributor and Sales Representatives

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

Appendix A – References

Document References

FT260 product page

DS_FT260

Texas Instruments INA219

Source Code

Acronyms and Abbreviations

Terms	Description
GPIO	General-purpose input/output
HID	Human Interface Device
I2C	Inter-Integrated Circuit bus
LDO	Low Drop Out regulator
USB	Universal Serial Bus

Appendix B – List of Tables & Figures

List of Tables

NA

List of Figures

Appendix C – Revision History

Document Title: AN_438 FT260 I2C Example in C#

Document Reference No.: FT_001419

Clearance No.: FTDI#554

Product Page: http://www.ftdichip.com/FTProducts.htm

Document Feedback: Send Feedback

Revision	Changes	Date
1.0	Initial Release	2020-05-12