Application Note

AN_356

FT800 Interfacing I2C Sensor to VM800P

Version Draft 1.0

Issue Date: 2015-06-15

The VM800P boards make an ideal platform for a monitoring and display application, due to their combination of FT800, LCD panel and ATMEGA MCU in a single module. This application note shows how a sensor can be connected over I^2C to the expansion connector on the board, its data read by the ATMEGA MCU, and a scrolling chart displayed showing the results. The example code is based on the FTDI libraries for the Arduino IDE.

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

Table of Contents

1 Introduction

The VM800P boards make an ideal platform for a monitoring and display application, due to their combination of FT800, LCD panel and ATMEGA MCU in a single module. This application note shows how a sensor can be connected over I^{2}C to the expansion connector on the board allowing its data to be read by the ATMEGA MCU.

For measurement applications, there is often a trade-off between providing a basic digital readout which shows only instantaneous values or connecting to a PC or embedded PC if more advanced graphical displays are required. The EVE family of devices provides an excellent solution where the graphical, control and audio advantages of the PC system can be incorporated into the instrument in place of the basic digital display. As demonstrated here, the object-oriented nature of the EVE family means that the amount of software development and the processing power of the MCU handling the measurements can still be comparable to the simple digital display option.

The demonstration takes readings of voltage and current from a Texas Instruments INA219 current monitor IC over I^{2}C, and then presents them as numerical readouts and also plots a scrolling chart of the current values. Three on-screen buttons allow the chart to be set for different ranges so that higher viewing resolution can be achieved at lower current values.

In this case, the sense resistor is connected in the VBUS line between the USB In and USB Out connectors. This allows the demo to be connected in-line with an existing USB connection in order to monitor the Vbus voltage and the current consumed by the USB peripheral.

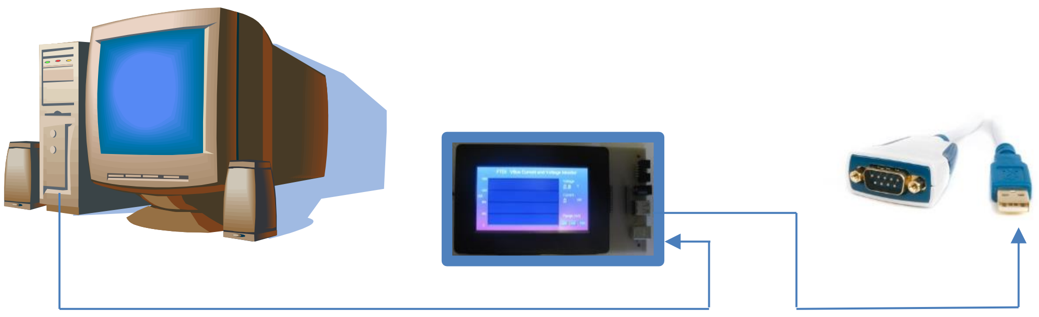


Figure 1-1 Measuring the VBus supply to an FTDI USB-Serial converter

In addition to producing a useful tool for checking the current consumption and bus voltage of a peripheral powered by USB, the application could be modified to work with other types of sensor such as temperature sensors, accelerometers, or ADCs with I^{2}C interfaces.

The techniques used in this example application include:

- Using the Arduino Wire I^{2}C library in conjunction with the FTDI FT800 Arduino libraries
- Using the Append command to make screen updates more efficient
- Plotting a basic scrolling chart on the FT800 screen with the line strip feature
- Adding and using a font to the FT800 application
- Using touch sensing to implement the range buttons

1.1 Scope

This application note is accompanied by a zip file containing the source code and additional supporting information. This document is intended to be read alongside the commented source code provided.

The example code is based on the FTDI libraries for the Arduino IDE for the FT800. The application is compatible with either the VM800P or VM801P by including the associated FT800 or FT801 library.

The supporting files for this example can be found at the following link:

http://www.ftdichip.com/Support/SoftwareExamples/EVE/AN_356.zip

The zip file contains:

Arduino project (tested on 1.0.5 r2)

Fonts folder containing the DS-DIGIT files used in this example

Additional documentation can be found at www.ftdichip.com/EVE.htm

It is recommended to refer to the documents in Appendix A – References of this application note for further information on topics covered in this application note.

This application note contains an example hardware design and software project. The developer of a final solution using any part of this example is responsible for ensuring the safe and correct operation and for any consequences resulting from its use. Developers must consult the full documentation for the I^2C peripheral regarding the full feature set and for recommended input protection and electrical characteristics etc. This software example has reduced levels of error checking in order to ensure the key principles are demonstrated clearly.

2 Demo Overview

The demonstration unit is shown below. The main features are highlighted in the picture.

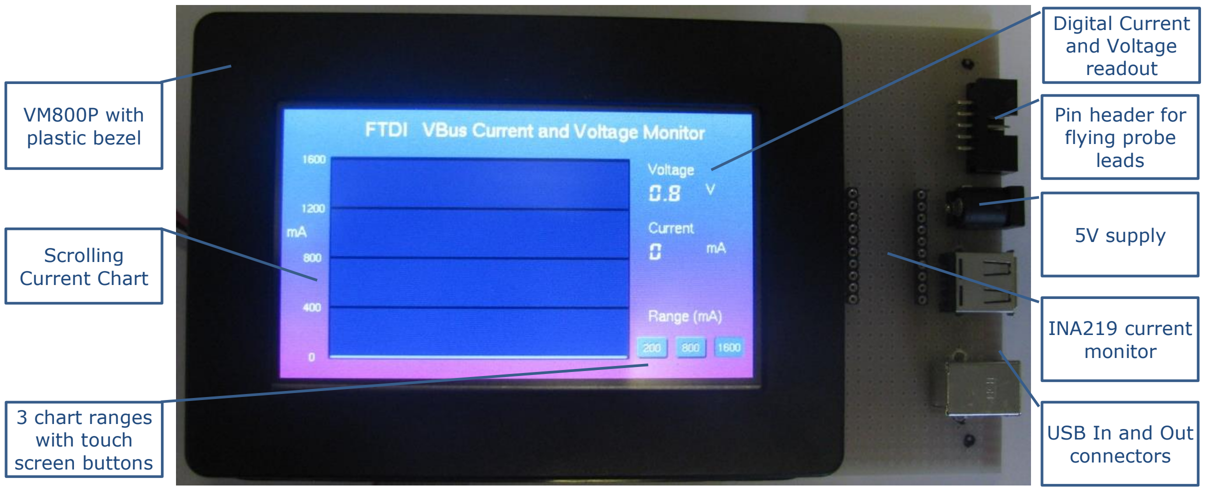

Digital Current and Voltage readout
Pin header for flying probe leads
5V supply
INA219 current monitor
USB In and Out connectors
VM800P with plastic bezel
Scrolling Current Chart
3 chart ranges with touch screen buttons
FTDI VBus Current and Voltage Monitor
Voltage
Current
Range (mA)
200
800
1600

Figure 2-1 VM800P and INA219/connector board

The unit is constructed with a standard VM800P board (in this case the 4.3”) with a small additional board containing the USB in and USB out connectors and the INA219 circuitry. This board connects to the Micro-MaTch™ connector on the rear of the VM800P.

Since the VM800P contains a bootloader, the code can be loaded by connecting the VM800P to the PC via the micro-B USB connector. The example code is loaded into the Arduino IDE from the .ino file in the provided zip package, and programmed into the ATMEGA MCU. The VM800P Datasheet (see Appendix A – References) has further details of connecting and programming the VM800P.

After programming, the sample code will first display the calibration screen. After tapping the three dots, the main application screen will be displayed. With nothing connected on the measurement connectors, the current should be 0mA and the bus voltage typically shows around 800mV as the lines are floating.

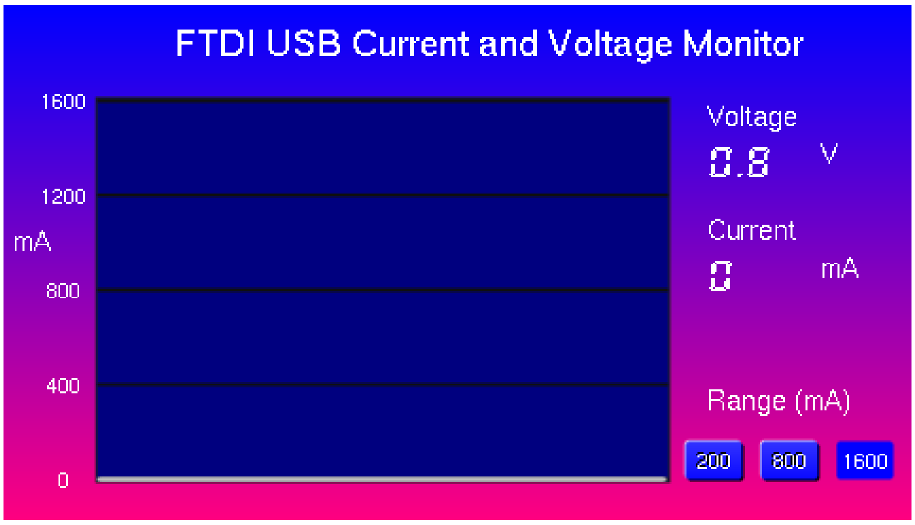


Figure 2-2 Demo after power-on with no USB connections

USB In can then be connected to a USB host, such as a PC USB port, via a standard USB A-B cable. It is recommended to keep this cable short (e.g. 1m or less) to keep the total length of the link as short as possible. The display will now show the voltage of the VBus line from the host (approximately 5V corresponding to the standard USB bus voltage).

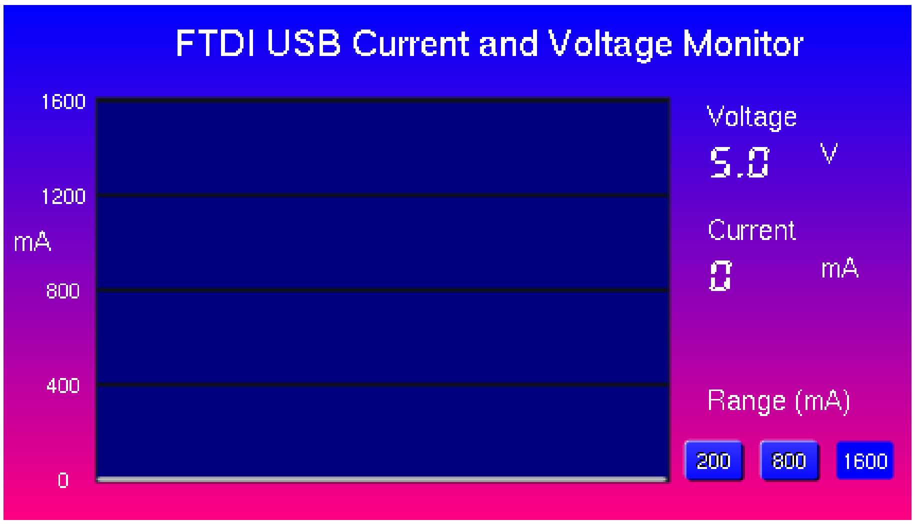


Figure 2-3 Connected to the USB host and showing 5V bus voltage

The USB peripheral can then be connected to USB Out. Since the USB In and USB Out connectors are linked on the demo board, the board is transparent to the operation of the USB peripheral and it will behave as if directly connected to the host PC (see note below). As the peripheral powers up, the current consumption will be shown on the digital readout and on the chart. In the example below, the USB flash drive is connected and enumerates. The current fluctuations correspond to the drive being read as the Windows explorer window opens.

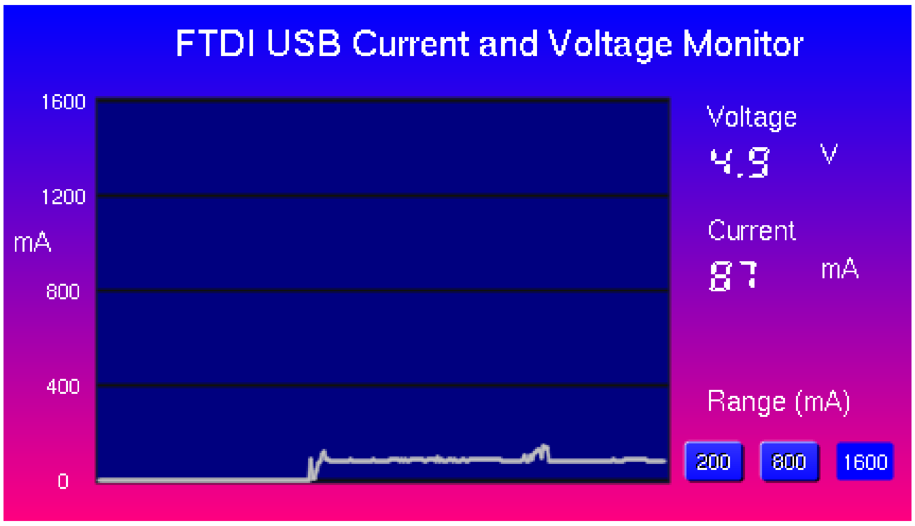


Figure 2-4 Flash drive connected and being read by Windows

Note: USB normally uses a single cable between the host and peripheral with a connector at each end. The two additional connections at the IN and Out connectors and the sense resistor will cause a small additional voltage drop in the link under test when using this unit.

The demo begins with range set for 1600mA full scale. If measuring a much smaller current, the range can be changed by pressing the touch screen buttons (e.g for 200mA full scale). In this example, the scale does not affect the digital readout value or the current measurement accuracy. It acts as a zoom function to allow best viewing resolution for small values.

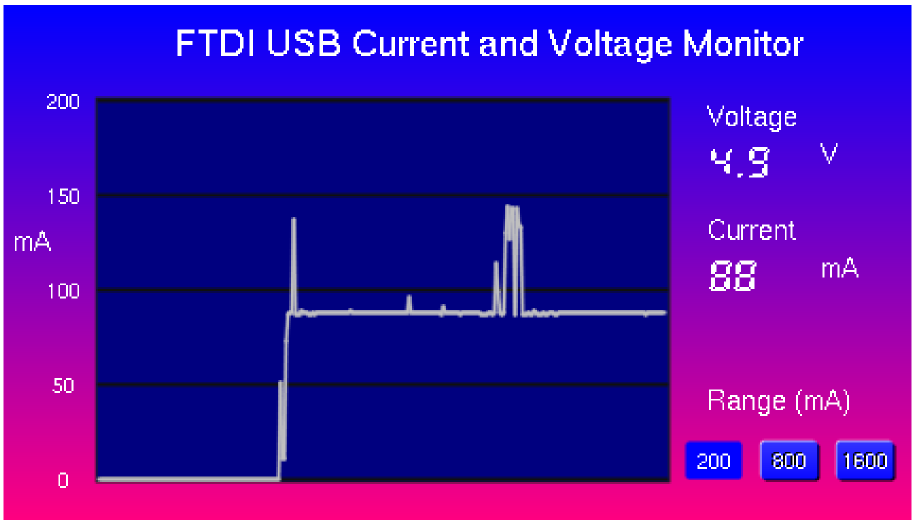


Figure 2-5 200mA range allows a more detailed analysis

If the measured current exceeds the scale set on the chart, the associated points will colored red to show that they are off the scale. The scale buttons can be used to select a larger scale so that the full current graph can be viewed. Here, an FT230X battery charge demo draws 90mA until enumerated but jumps up to over 400mA after enumeration (and therefore when the PC gives permission to do so) in order to charge the battery quickly. Switching to the 800mA scale via the Range buttons will immediately zoom out to show the full waveform.

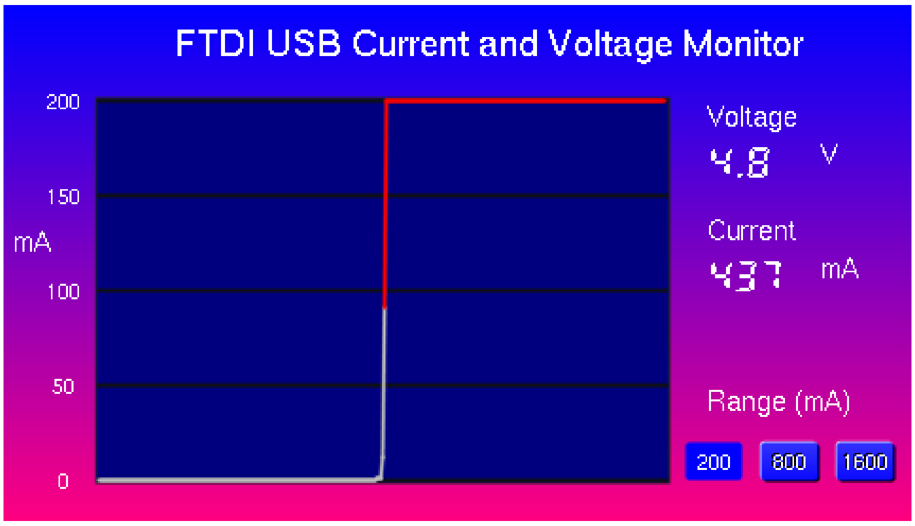


Figure 2-6 Values exceeding the selected range are colored red

In some cases, it may be required to measure current consumption of an MCU or evaluation board and in these cases, the 10-way header provides alternative access to the Current IN, Current OUT and GND lines. The IN and OUT wires can then be used to connect the demo in-line with the current to be measured. For example, the board may have an IDD measurement jumper or another place where the line to be measured can be intercepted. The image below shows a way of measuring the consumption of a VM800P by replacing its power select jumper with the flying leads.

Note: The USB In/Out and the 10-way header must not be used together. Only one of the connection types can be used at any time.

Note: The GND must always be connected to the GND of the development board under test first. This ensures that the board under test has the same ground level as the current measurement demo.

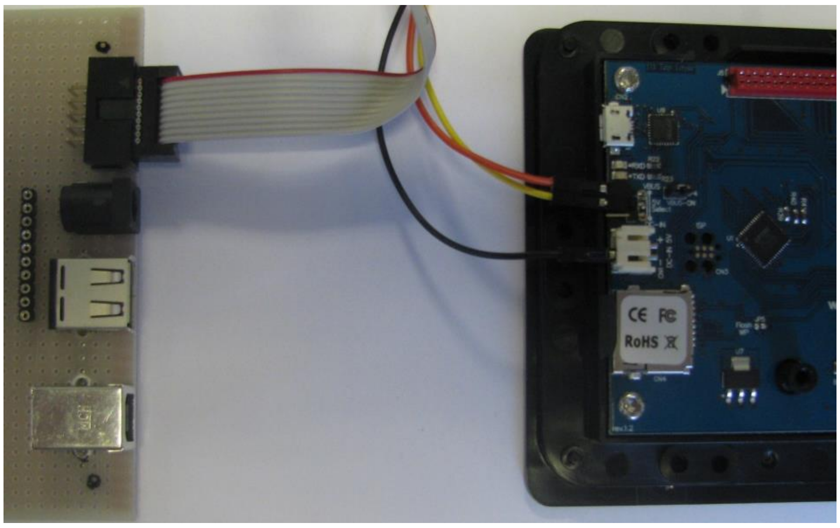


Figure 2-7 Connecting In, Out and Ground from demo to the VM800P board

The demonstration unit is configured primarily as a USB monitor, to measure 0 to 1600mA, with nominal VBUS 5V (max 16V).

However, the INA219 can be configured to measure higher currents by adjustment of the configuration register settings and the shunt resistor, and up to 32V for the VBUS measurement.

3 Hardware

The application is based on the VM800P plus board. This contains an FT800, an ATMEGA 328 with built-in Arduino bootloader, and the LCD panel itself. The VM800P also contains an audio amplifier and all supporting circuitry to make a self-contained display module. This makes it an excellent platform for this measurement and display application.

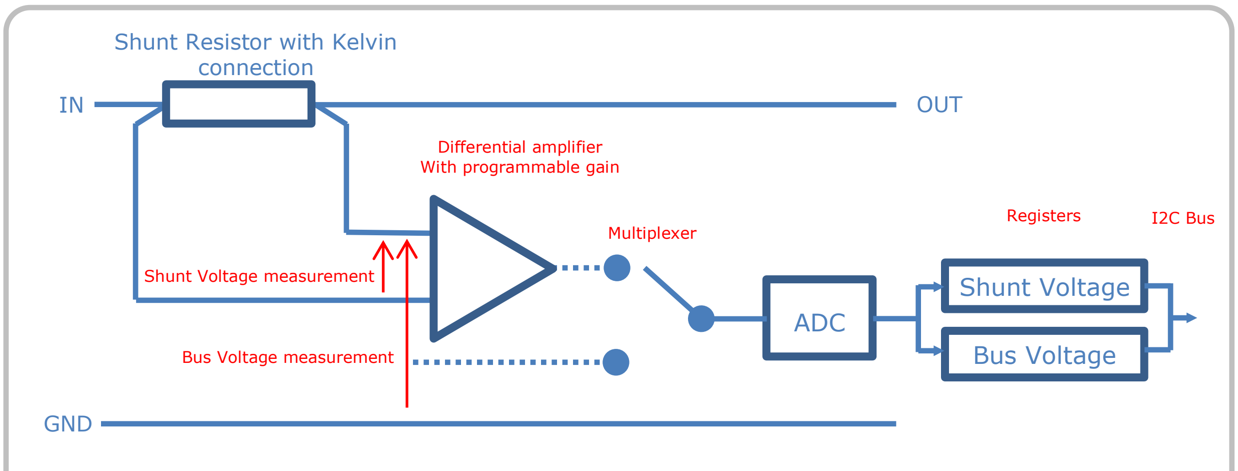
The VM800P uses two red Micro-MaTch connectors which allow access to the GPIO lines of the ATMEGA MCU. The MCU has an I^{2}C Master peripheral which is used in this example via the Wire library provided with the Arduino IDE.

A small external board containing the USB connectors for the link under test, the sense resistor and the INA219 and supporting components attaches via a short ribbon cable to the Micro-MaTch connector of the VM800P.

3.1 Current Sensing

The INA219 is a high-side current shunt monitor. It measures the amount of current flowing through a low-value shunt resistor which would typically be placed in series with a low voltage supply to a device. The INA219 contains an amplifier which senses the differential voltage across the resistor, which will be proportional to the current flowing through it. It converts this to a ground-referenced voltage and measures it with an integrated ADC. The host system can then read the shunt voltage over I^{2}C. The device also measures the voltage of the bus being measured compared to ground after the sense resistor.

The basic operation of the current sensing is shown below.



Note: This diagram is intended to explain the principles used in this example application and not the full feature set or detailed architecture of the INA219

Figure 3-1 Basic operation of the current and bus voltage sensing

3.2 Schematic

The schematic of the measurement board is shown below. The schematic of the VM800P can be obtained from the VM800P datasheet (see Appendix A – References).

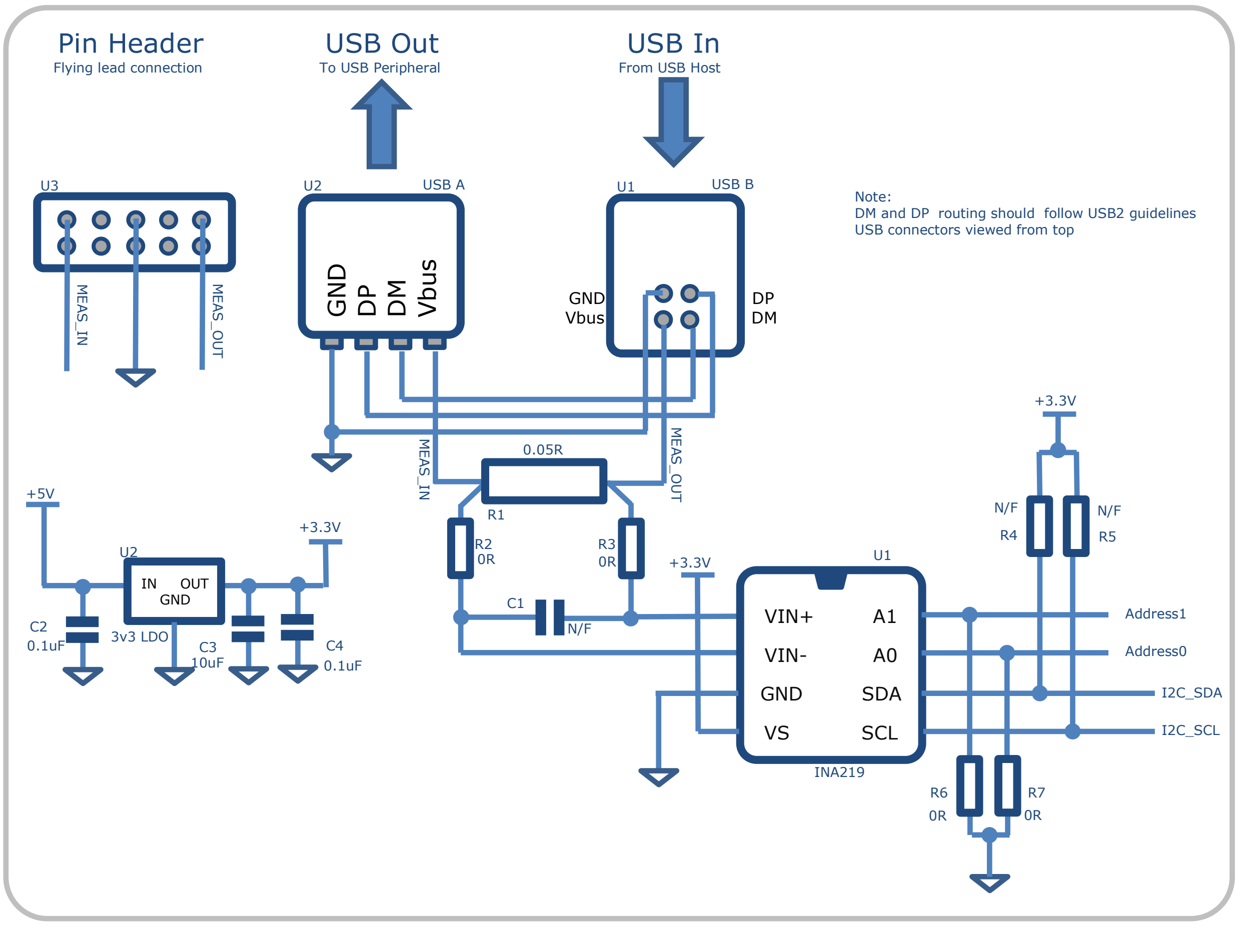


Figure 3-2 Schematic of measurement circuit

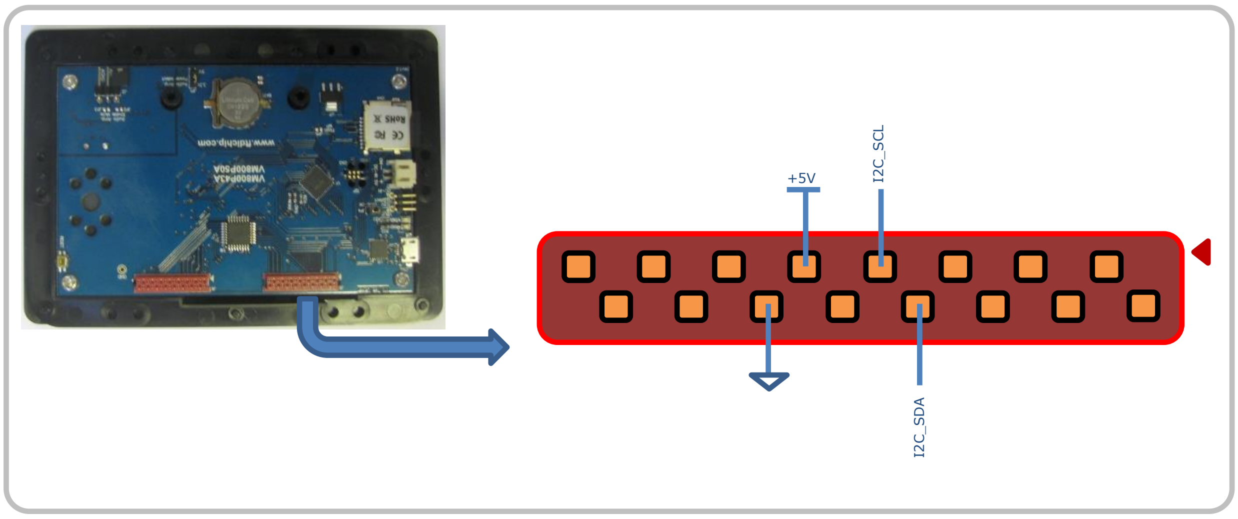


Figure 3-3 MicroMaTch Pinout on VM800P

The USB connectors are wired in a pass-through configuration so that the demo can be connected in-line with the peripheral to be measured. The ground and Data lines of the USB link are simply connected from the type-B socket to the type-A socket so that the demo is transparent to the device under test. USB connectors are viewed from the top in this diagram.

Since the unit may be used to test hi-speed USB 2.0 devices, it is important to keep the Ground and USB data connections between the USB In and USB Out connectors short and of matched length. A 4-layer PCB is recommended following the same guidelines for hi-speed USB design to avoid introducing data errors in the USB link. However, a short pair of twisted wires for USB DM and USB DP has been found to work successfully for testing purposes and was used in the construction of the demonstration unit.

The VBUS line is connected via a low-value shunt resistor to allow the current to be measured. This resistor combined with the two additional USB connectors will lead to some additional resistance and resulting voltage drop for the link under test. It is important that the 0.05R resistance between the two sense lines is as accurate as possible since the derived current values are based on the resistance value. A Kelvin connection, where the sense traces from the INA219 are routed directly to the resistor rather than to the traces carrying the current from the resistor, is recommended to avoid the voltage drop on the traces affecting the accuracy.

A high-power 0.05R 5W surface mount resistor, designed for shunt applications, was selected to ensure the resistor could cope with currents over the measurement range and in the event of overcurrent conditions.

The INA219 can have small value series resistors (e.g. 10R) for R2/R3 to provide additional protection to the inputs although these may slightly reduce accuracy. 0 Ohms was fitted to the demo unit. Capacitor C1 is for an optional filtering capacitor.

A second connection method is provided by the 10-way header, allowing flying leads to be used instead of the USB In and Out connectors. These connectors provide different access to the same sensing resistor and so only one or the other should be used at any time.

A regulator was provided to provide a smooth stable 3v3 supply for the INA219, and reduce dependency on any variation in the main 5V supply to the module. However, the INA219 can be powered directly from the 5V or 3v3 rails of the VM800P to reduce component count.

The pull-up resistors on the I^2C lines are not fitted as the VM800P has on-board resistors.

The demo uses a 0.05R resistor but the value can be optimized depending on the current to be measured. In general, a larger resistor offers better accuracy with small currents. But for larger currents, voltage dropped across the resistor must be considered. The INA219 can support a variety of sense resistors and also has configurable gain set via its configuration register. The gain allows the internal converter to make best use of the range of voltages which will be present over the shunt. A lower value shunt or change to the gain setting could extend the range to 3200mA to include the range used by the new USB charging specifications.

3.3 Connecting Multiple I2C Devices

If more than one I^2C peripheral is to be used on the same I^2C bus, the GND, SDA and SCL lines are connected to each peripheral. Each peripheral must have a different address set. If using multiple INA219 devices, the two address pins can be used to configure them to different addresses. Other I2C peripherals may have different ways to set the address and so please consult the datasheet of the peripheral.

The VM800P has a Real-time Clock (RTC) on the I2C bus at address 0x6F and so this address should not be used for external I2C peripherals.

3.4 Additional Notes

Whilst a PCB is recommended for any final application, a variety of break out boards are available for the INA219 from third parties or alternatively a small SMD adapter can be used if constructing on prototyping board. For example, the Roth Elektronik RE906 adapter converts the 8-pin SOT23 package to a DIP footprint.

The B version of the INA219 was selected as it is specified to have higher accuracy.

I^{2}C opto-isolators are available to provide additional isolation and protection between the VM800P and any I^{2}C peripherals connected.

For connection to the VM800P, ribbon cables are available with Micro-MaTch connectors assembled already or the Micro-MaTch connectors are available as separate items in either IDC or PCB mount versions. The cables are useful when creating custom interfaces to the VM800P such as the board used in this demo.

Note: FTDI cannot guarantee suitability or specification of any third party adapter or component. Please check with the supplier or manufacturer to ensure suitability and latest specification when selecting components as these could be subject to change outwith the control of FTDI.

4 Software

This section describes the example software provided for the Arduino IDE. It assumes that the FTDI FT800 library is installed. The library and installation guide can be found in Appendix A – References.

4.1 Flow Chart

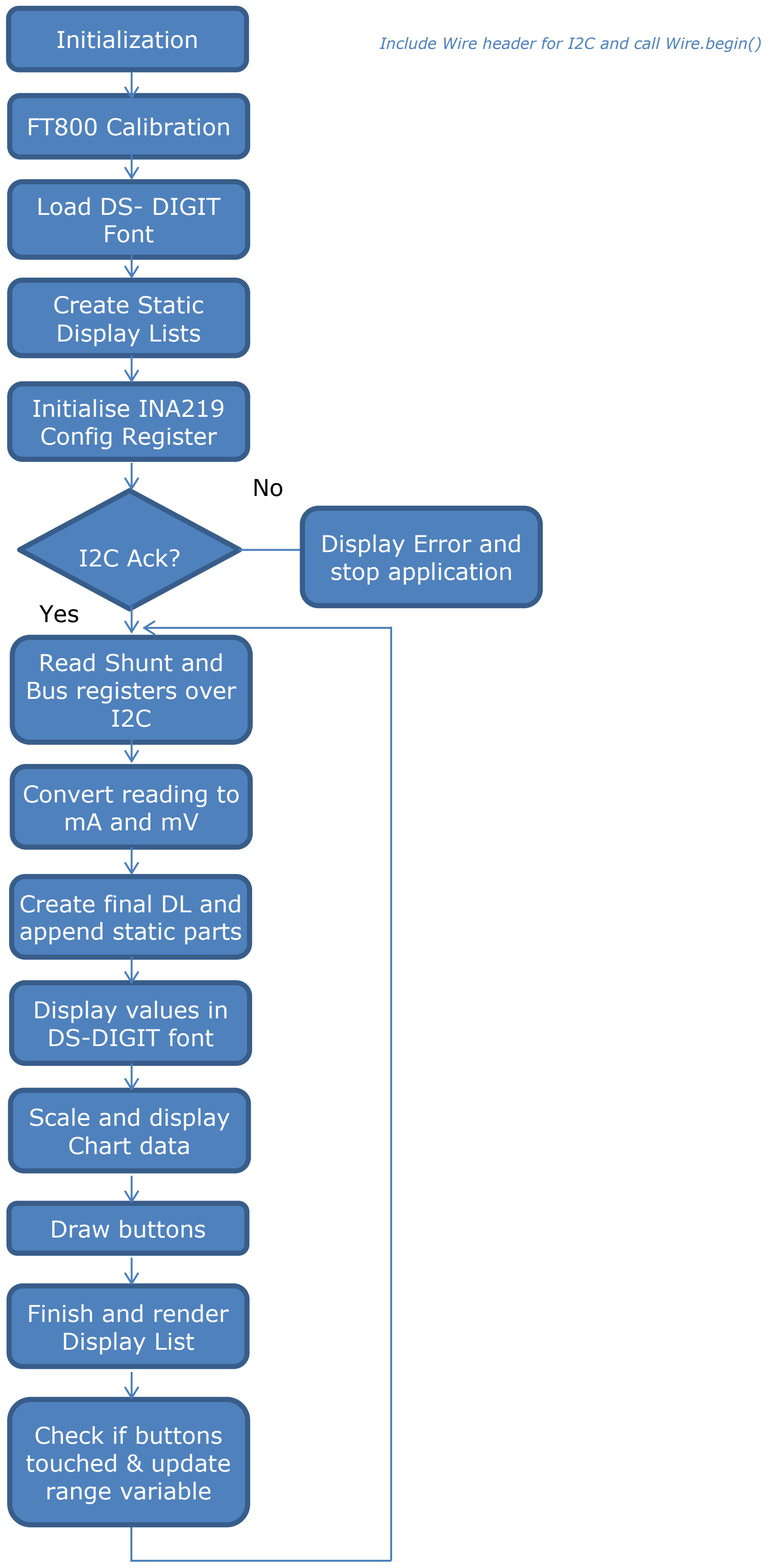

Initialization
Include Wire header for I2C and call Wire.begin()
FT800 Calibration
Load DS- DIGIT Font
Create Static Display Lists
Initialise INA219 Config Register
I2C Ack?
No
Display Error and stop application
Yes
Read Shunt and Bus registers over I2C
Convert reading to mA and mV
Create final DL and append static parts
Display values in DS-DIGIT font
Scale and display Chart data
Draw buttons
Finish and render Display List
Check if buttons touched & update range variable

Figure 4-1 Application flowchart

4.2 Initialization

4.2.1 Device Initialization

The application first performs the typical initialization of the FT800 including reading the ID register to confirm correct communication with the FT800 and configuring display settings via the library functions.

In addition to the FT800 header files, the Wire.h library must be included in order to use the I^{2}C Master functionality. The digit.h file is also included here as it contains the font data for the 7-segment DS_DIGITS font used in the digital voltage and current readouts.

```
/* Arduino standard includes */
#include "SPI.h"
#include "Wire.h"
#include "digit.h"

/* Platform specific includes */
#include "FT_VM800P43_50.h"
/* Global object for FT800 Implementation */
FT800IMPL_SPI FTImpl(FT_CS_PIN,FT_PDN_PIN,FT_INT_PIN);
```

The application must also call the Wire.begin() function to initialize the I^{2}C library. This call can be found within the Setup() function of the sample code.

4.2.2 Calibration

Since this example uses touch for the range selection buttons, the Calibration command is called using a short co-processor list. This allows the FT800 to calculate the transform values to ensure that it can align the touch panel output to the display screen accurately.

Note that the values of the six REG_TOUCH_TRANSFORM A – F registers could be stored in the MCU's EEPROM after completion of this calibration procedure. The values could then be restored to the registers on power-up, avoiding the need to have this calibration routine displayed every time.

For example, a calibration of the finished product may be carried out during factory test and the results loaded on each subsequent power-up.

The code now enters the function Chart() and remains there for the duration of the application. The following sections describe the different code sections within Chart().

4.3 Application initialization

4.3.1 Load the DS-DIGIT Font

A 7-segment font is used in this example for the Voltage and Current read-outs. This is the same font which was used in the Gauges example. However, the font was converted in a smaller font size for this application. The font files are included in the accompanying zip file for this application note.

The font was converted using the FTDI font_cvt tool (see Appendix A – References) using the following command. The parameters specify size 25, all characters and address in RAM_G is 1000.

```
fnt_cvt.exe -i DS-DIGIT.TTF -s 25 -a -d 1000
```

In this case, the entire character set was converted as the font gives good appearance with high compression (L1 in this case) levels and fits easily in RAM_G. Application note AN_277 (see Appendix A – References) has further information on converting fonts.

The resulting output from the rawh file in the L1 folder was pasted into the MetricBlock and FontBmpData sections of the digits.h include file. The Metric Block, followed by the Bitmap Data, is loaded from the MCU into the FT800s RAM_G over SPI.

```
FTImpl.Writefromflash(FT_RAM_G + 1000, MetricBlock, sizeof(MetricBlock)); // Loading data
FTImpl.Writefromflash(FT_RAM_G + 1000 + sizeof(MetricBlock), FontBmpData, sizeof(FontBmpData));
FTImpl.DLStart();
FTImpl.Cmd_SetFont(14,1000);                              // Allocate to font 14
FTImpl.BitmapHandle(14);                                  // Set bitmap handle as 14
FTImpl.BitmapSource(-1252);                               // Set bitmap source (see metric block in digit.h)
FTImpl.BitmapLayout(FT_L1,3,25);                          // Tell the FT800 format, stride and height of font
FTImpl.BitmapSize(FT_NEAREST,FT_BORDER,FT_BORDER,18,25);// Tell the FT800 the bitmap size (width & height)
FTImpl.DLEnd();                                           // End the display list
FTImpl.Finish();                                          // Render and wait for completion of the DL
```

The Font can now be used by selecting font 14 in the Text and Number commands.

4.3.2 Create Static Display Items

There are many parts of the screen which never change during the running of the demonstration. Although the entire screen could be sent to the FT800 every time a change occurs, it is more efficient to store the static parts of the screen as display list sections in RAM_G. These are then recalled using the Append command.

More information on this technique can be found in AN_340 (see Appendix A – References)

First, a display list is stored which draws the chart background and the static text labels. The most important parts of the code are shown below. It is important that the screen is not cleared within the stored display list sections when appending multiple display list sections as this would clear any items added beforehand. Likewise, REG_CMD_DL is written to 0 rather than using DlStart when creating lists to be appended as DlStart clears the screen. The screen will be cleared before calling the appended sections as part of constructing the final display.

```
FTImpl.Write16(REG_CMD_DL,0);      // Make sure next Display List is started at 0 offset
FTImpl.Cmd_Gradient(230, 0, 0xFF0000FF, 230, 272, 0x00FF007F); // Background is a gradient
[Remainder of Display List for background]
FTImpl.End();                                                // End rectangle primitive
FTImpl.Finish();                                             // Render and wait for completion of the DL
```

The co-processor will now execute the commands above and create the resulting display list in RAM_DL. This can now be copied into RAM_G and recalled later. The size of the created display list is required as a parameter of MemCpy and can be determined from a register (REG_CMD_DL). The display list can be copied to an area of RAM_G selected by the user, taking care to avoid any other data loaded in a similar way (such as user fonts, bitmaps and other stored display lists).

```
uint16_t dloffset_backgnd;                                   // This variable will hold size of D/L
dloffset_backgnd =  FTImpl.Read16(REG_CMD_DL);               // This register points to the end of the DL
FTImpl.Cmd_Memcpy(100000L,FT_RAM_DL,dloffset_backgnd);       // Copy amount of data [dloffset_backgnd]
                                                                from RAM_DL into RAM_G memory beginning
                                                                at address 100,000
```

The items created are shown in the image below.

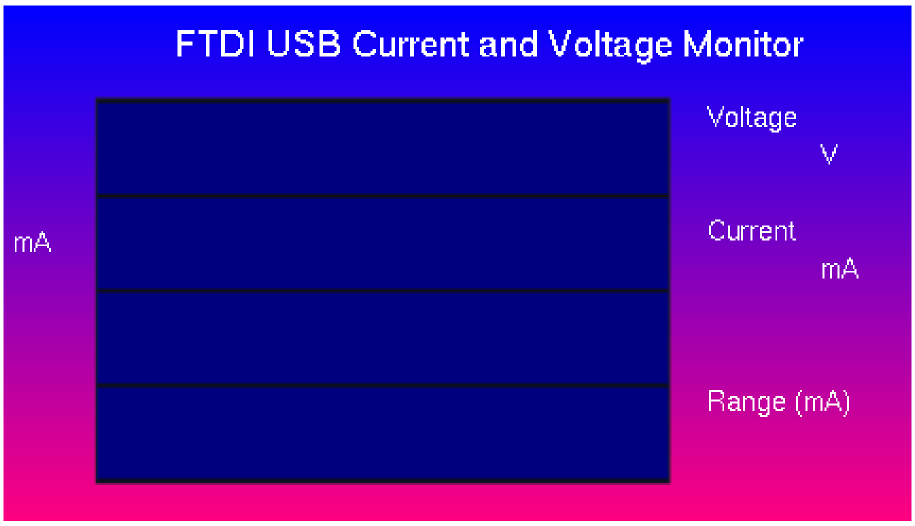


Figure 4-2 Static display items

Since this application offers a fixed set of three chart ranges for the current scale, three smaller display lists were stored which add the scale values to the Y axis. For example, the 200mA one is shown below.

```
FTImpl.Write16(REG_CMD_DL,0);
FTImpl.ColorRGB(255,255,255);
FTImpl.Cmd_Text(Division1_Label_X, ChartBR_Y - (DivisionFontHeight/2), 20, 0, "   0");
FTImpl.Cmd_Text(Division1_Label_X, (ChartBR_Y - DivisionSize - (DivisionFontHeight/2)), 20, 0, "  50");
[Remainder of Display List for background]
FTImpl.End();
FTImpl.Finish();
uint16_t dloffset_Scale200;
dloffset_Scale200 =  FTImpl.Read16(REG_CMD_DL);
FTImpl.Cmd_Memcpy(150000L,FT_RAM_DL,dloffset_Scale200);            // Copy to location 150000 in RAM_G
```

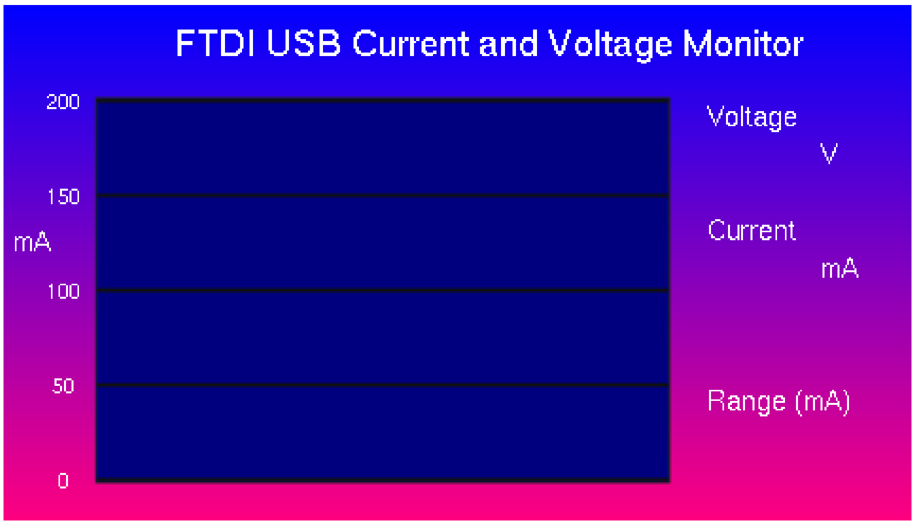


Figure 4-3 Static display items with 200mA scale appended

The final screen will be made up by clearing the screen, appending the background followed by one of the scales and then adding the dynamic part with the Voltage and Current readout and chart data line.

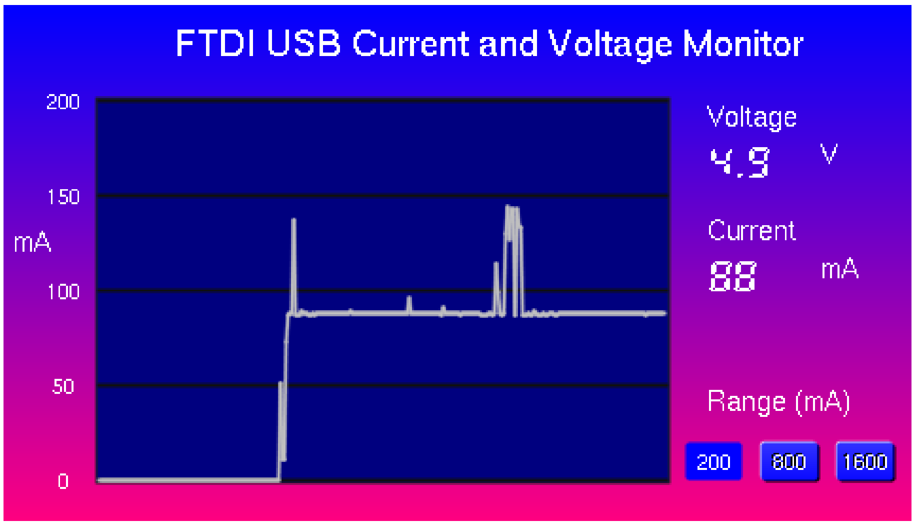


Figure 4-4 Final screen with appended background/scale and dynamic items

4.3.3 Configure the INA219

The final step in the Application Initialization is to configure the INA219 over I^{2}C in preparation for taking voltage and current readings from its result registers.

This application uses the standard Arduino Wire library to implement the I^{2}C Master communication. The library requires the following to be carried out

- Include the library – see #include "Wire.h" at the top of the code
- Call the Wire.Begin() function during initialization - see the setup() function in the code

The following code shows an I^{2}C write transaction.

```
Wire.beginTransmission(I2CADDRESS);      // INA219 has address 0x40 in this example (see INA219 datasheet)
Wire.write((char)(0x00));                // Write the register address - 00 is the INA219's config register
Wire.write((char)(0x0E));                // Write value 0x0EEF (80mV shunt gain, constant conversions)
Wire.write((char)(0xEF));                // The INA219 datasheet explains the config bits in detail
I2C_Status = Wire.endTransmission(true); // Send off these bytes over I2C with an Stop at the end
```

The INA219 has both of its address pins pulled low in the hardware for this example and so its I^{2}C address is 0x40. The define for I2CADDRESS should be set to the appropriate value if a different address is configured in hardware. The I^{2}C peripheral datasheet should be consulted for addressing options.

Note: The VM800P has a Real-time Clock (RTC) on the I^{2}C bus at address 0x6F and so this address should not be used for external I^{2}C peripherals.

When writing a register, three bytes are sent. The first specifies which register within the INA219 is to be accessed. The next two bytes contain the data to be written. The INA219 has a variety of features and settings which can be utilized depending on the application. This example configures the device for a basic continual reading of the shunt and bus voltages with an 80mV sense voltage range.

Configuration register: 0000 1110 1110 1111

0	RST	Not in reset
0		Not used
0	BRNG	Bus voltage range = 0-16V
01	PG1:0	Gain for shunt voltage = 80mV
1101	BADC4:1	Bus ADC settings = Average 32 samples
1101	SADC4:1	Shunt ADC settings = Average 32 samples
111	Mode3:1	Mode setting = measure bus & shunt continuously

Finally, the Wire.endTransmission call begins the I^{2}C transaction and the message goes onto the I^{2}C bus. The return value indicates the success or failure of the call. If the INA219 is not connected correctly, not set for the address matching the application code, or not powered etc. the return value will be non-zero as the INA219 will not ack the address phase of the I^{2}C transaction. In this case, an error message is displayed and the application will enter an indefinite while loop, since no INA219 readings will be able to be taken to support the remainder of the application.

4.4 Main Program Loop

The code now enters the main while() loop and stays here for the remainder of the application.

Each time round the loop, one new sample of the voltage and current values is taken and the chart scrolls left by one pixel, such that the latest sample can be displayed on the right-most pixel. The digital displays update to reflect the latest measured value.

4.4.1 Read INA219 Current and Voltage Registers

The INA219 has already been configured and so the latest voltage and current measurements can be obtained by reading the respective register over I^{2}C.

First, the bus voltage reading is taken.

```
// Select bus voltage register
Wire.beginTransmission(I2CADDRESS);         // I2C Address of INA219 in this example is 0x40
Wire.write((char)(0x02));                   // First byte sent selects the register, 02 is bus voltage
Wire.endTransmission(true);                 // Send this command over I2C

Wire.requestFrom(I2CADDRESS, 2, true);      // Request 2 bytes from the INA219

TimeoutCounter = 0;
while((Wire.available() < 2) && (TimeoutCounter < 1000))     // wait for 2 bytes back, timeout in ~ 1 sec
{
    TimeoutCounter ++;
    delay(1);
}
VoltageHi = (Wire.read());                  // Read high byte of the bus voltage measurement
VoltageLo = (Wire.read());                  // Read low byte of the bus voltage measurement
```

When reading over I^{2}C, the first step is to send a byte to the INA219 to select the register which will subsequently be read. In this case, a transaction to the INA219's I^{2}C address is started, sending the address of the Bus Voltage register 0x02.

The earlier transaction to the config register of the INA219 has confirmed that the INA219 is present and responding but further error handling could be added in the event that the Wire.endTransmission call returns an error (e.g. the ribbon cable to the INA219 becomes disconnected).

Then the Wire.requestFrom call addresses the INA219 again. This causes the device to send the INA219's address with the I^{2}C Read bit set, and then clock in two bytes from the INA219. In this case, two bytes are requested as the result register is 16-bit.

The while loop waits until the two bytes come back (unless they have not come back within a reasonable time in which case the timeout counter will exit the loop) and they can be read by the Wire.read call. The bytes are currently in the Wire libraries buffer. Each Wire.read call retrieves the next byte from the Wire libraries buffer.

A similar process is repeated for the Current value, by reading the Shunt Voltage register. The same Wire functions are used but in this case Register 0x01 is selected with the initial I^{2}C write, which corresponds to the Shunt Voltage register in the INA219.

4.4.2 Calculate Current and Voltage Values

The I^{2}C operations in the previous section have obtained the two 16-bit values, one for the Bus Voltage and one for the voltage across the shunt resistor. The next step is to turn these into real voltage and current values.

Voltage

For the Bus Voltage, assuming the settings as shown in the previous section (Configure the INA219), the bits are arranged as follows:

```
High byte:  --- B11 B10 B09 B08 B07 B06 B05
Low byte:   B04 B03 B02 B01 B00 --- --- ---     (Bxx = Bus voltage bit)
```

The code masks the received data to ensure that any of the non-value bits are zero and then shifts and combines the value to form a single right-justified value containing all 12 bits.

```
 0   0   0   0  B11 B10 B09 B08 B07 B06 B05 B04 B03 B02 B01 B00
```

The 12-bit resolution gives 4096 steps to 16V and so each step is 4mV. Therefore, the final binary result is multiplied by 4.

The Voltage variable now contains a final bus voltage value in mV

Current

For the Shunt Voltage, assuming the settings as shown in the previous section (Configure the INA219), the bits are arranged as follows:

```
High byte:  SGN SGN SGN S12 S11 S10 S09 S08     (SGN = Sign bit)
Low byte:   S07 S06 S05 S04 S03 S02 S01 S00     (Sxx = Shunt voltage bit)
```

This example was designed to display current in one direction. As a bi-directional current monitor, the INA219 supports both directions (shunt voltage from -80mV to +80mV). This example checks the sign bits and if the negative sign bit is set, it sets the result to 0.

Note: This could be expanded upon to create a bi-directional sensing application, where the chart shows both positive and negative scales, or where a single scale is used with colour coding of the chart trace to show the polarity.

The code then masks the received data to ensure that the sign bits are zero and then combines the value to form a single right-justified value containing all 13 bits.

```
 0   0   0  S12 S11 S10 S09 S08 S07 S06 S05 S04 S03 S02 S01 S00
```

The 13-bit resolution gives 8192 steps. Since the maximum measureable value is

$$I = \frac{V}{R} = \frac{INA219\ Shunt\ Range}{Shunt\ Resistor} = \frac{0.08V}{0.05\ Ohms} = 1600mA$$

Each step is $\frac{1600mA}{8182}$ = approximately 0.2mA per step

Therefore, the final binary result is multiplied by 0.2.

The Current variable now contains a final bus current value in mA

4.4.3 Begin Final Screen and Append Static Items

The application now begins to construct the final screen. It begins by creating a new display list, and then clearing the screen.

```
FTImpl.DLStart();                                   // Start the display list.
FTImpl.ClearColorRGB(0,0,0);                        // Set the background color to black
FTImpl.Clear(1,1,1);                                // Clear the screen
```

Now, the background items previously stored can be added. A single Append command over SPI replaces the overhead of re-sending all the background items on every screen update. The Append command takes the starting address and the length of the stored display list as parameters.

```
FTImpl.Cmd_Append(100000,dloffset_backgnd);         // Append the static background
```

Now, the scale values can be added by selecting one of the three display lists stored previously, using the Append command. These add the values beside each tickmark on the vertical scale.

The variable CurrentRange is initialized to 0 when the program begins and so the 1600mA scale will be selected on the first time through the main program loop. The buttons will be drawn and the touch tags checked later in the main program loop, which in turn lead to the variable CurrentRange being updated, and so the other range options may be selected on subsequent passes through this part of the code.

```
// Put the values beside the Division marks on the Y scale depending on which range is set
if(CurrentRange == RANGE_200)
{
    FTImpl.Cmd_Append(150000,dloffset_Scale200);        // Append the scale
}
else if (CurrentRange == RANGE_800)
{
    FTImpl.Cmd_Append(160000,dloffset_Scale800);        // Append the scale
}
else
{
    FTImpl.Cmd_Append(170000,dloffset_Scale1600);       // Append the scale
}
```

The range/scale values are selected on every run through the main loop rather than being chosen only at start-up. This allows the user to change the graph scale at any time.

4.4.4 Write Current and Voltage values

The Voltage and Current values can now be shown in the meter-style numerical display at the right-hand side of the screen. These values are derived directly from the Current and Voltage variables calculated earlier, whereby the Current variable contains the current in mA and the Voltage variable contains the voltage in mV.

The current value can be directly passed to the Cmd_Number function. Font 14 is selected which is the DS-DIGIT font which was loaded earlier.

For the voltage, the value could be displayed as mV. However, the code converts this to a Voltage value with one decimal place for a more familiar meter-style display. Dividing by 1000 gives the value in Volts and taking the remainder and dividing by 100 leaves only the first digit after the decimal point.

Since the labels and units for the digital readouts never change, they were added as part of the static background via the Append command.

4.4.5 Plot chart

The application then plots the chart of the current value. The chart is 300 pixels wide and 200 pixels high.

The chart uses an array of 300 unsigned integers, which gives one y value for each of the 300 x coordinates on the chart. Although the chart is only 200 pixels high, and so in theory an 8-bit data type would be sufficient, this would require the values to be scaled before storing in the array and would prevent instantaneous range changes without flushing the entire array.

The array ResultBuffer operates as a simple circular buffer, with an input pointer (InputIndex) and an output pointer (OutputIndex). Each time through the main application loop:

- A new current reading is obtained. The InputIndex pointer is incremented by 1 (with wraparound at 300) and the value is stored in this location, overwriting the oldest value.
- The linestrip function draws a series of 300 points (x = 0 to 299) using Y values beginning at (InputIndex + 1). This has the effect that the chart is drawn with oldest value at the left-hand side and working through until the newest value at the right hand side.

Because the new value is always added at the right-hand side, the chart appears to scroll right-to-left by one pixel every time the new measurement is added.

One example is shown below, where the latest measurement goes in at index 110. The chart is drawn beginning from X = 0 and taking the Y value beginning at result 111. As discussed previously, the chart data is scaled depending on user selection and is divided by 8 in the example shown below due to the 1600mA range selection.

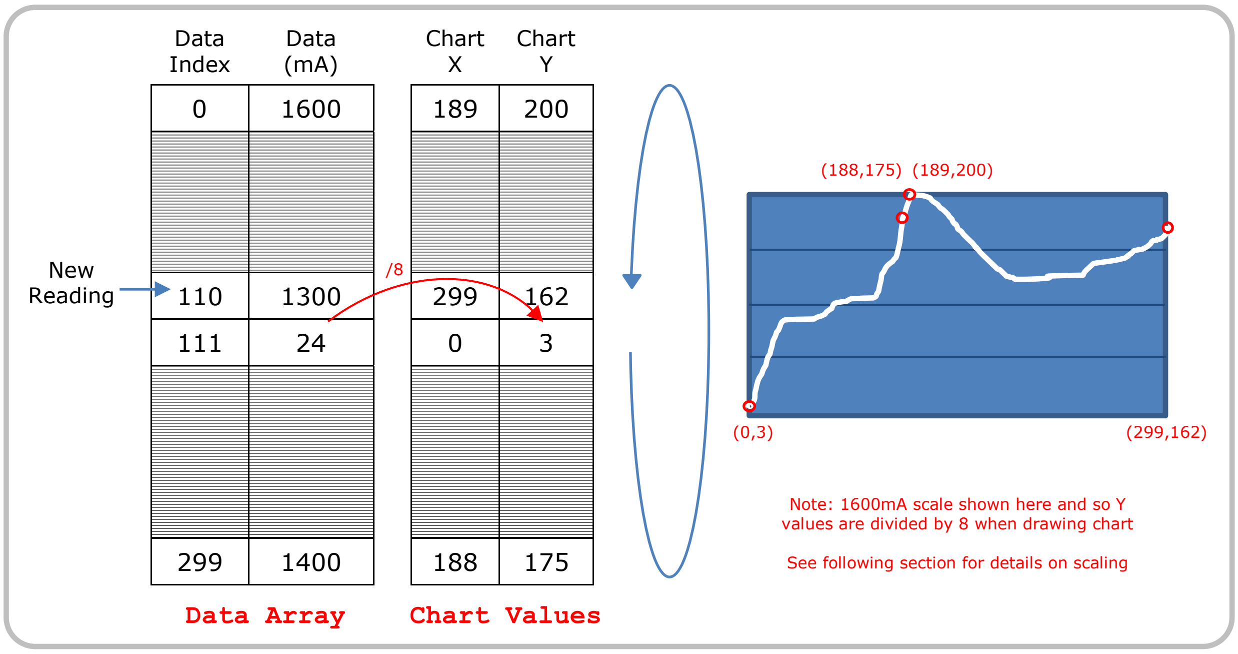


Figure 4-5 Chart plotting

A while( ) loop now plots the 300 points.

Scaling the chart data

Each time round the loop, the next value from ResultBuffer is read into variable ChartCurrentValue. This variable defines the Y offset above the bottom of the chart for the current point.

Since ResultBuffer values contain the current in mA, up to the maximum of 1600mA, they can be scaled at this point to determine the range.

- If 200mA range is enabled, the value from ResultBuffer is used directly (1mA per pixel)
- If 800mA range is enabled, the value from ResultBuffer is divided by 4 (4 mA per pixel)
- If 1600mA range is enabled, the value from ResultBuffer is divided by 8 (8 mA per pixel)

The value passed as the Y offset is then limited to 200 before plotting the actual point. If for example the 200mA range was selected but the current data point was 500mA, the point would otherwise be off the top of the chart. In this case, the 500mA point will show as 200mA at the very top of the chart. The colour is changed to red for this point to indicate to the user that the value is actually above the range shown.

Because the current values are always stored in the array as the full 1600mA range, the user can immediately change the range to see the correct value on the chart since it is fully re-drawn every time a new measurement is taken.

Note that the digital readouts will still show the 500mA value regardless of the range since they are not scaled.

Note: Implementing auto ranging in this code, or using the INA219's different shunt ranges and power calculating capabilities, could further enhance this example.

Line Strip

The actual chart is drawn using the Line Strip primitive. The line strip begins at the coordinate specified by the first Vertex2F instruction. The code sends a series of 300 Vertex2F commands, one for each X value of the chart, and the FT800 plots these points and links them with a line.

The X coordinates are handled by counter ChartIndexX which runs from 0 to 299.

The Y coordinates for each X value is formed by taking the Y value of the bottom edge of the chart and subtracting the value to be displayed, which has been limited and scaled to the range 0-200 by the previous code. This subtraction moves the point being plotted above the bottom of the chart by the amount to be displayed.

4.4.6 Draw Buttons and Finish Display List

The buttons can now be drawn to allow the user to select the range.

Each button is given a tag value, which is a very useful feature of the FT800 and allows the button presses to be detected very easily and with a minimum of coding. The tag mask is set to enable the tagging and the tag register defines the tag given to subsequent objects (until the tag is changed or masked again). The 200mA, 800mA and 1600mA buttons are given tags 2, 8 and 16 for ease of identification as only three of the available 255 values are required here.

Each button may be drawn with 3D (giving a 'not pressed' effect) or flat (giving a 'pressed in' effect.

```
FTImpl.ColorRGB(0xFF,0xFF,0xFF);        // Ensure text is drawn white
FTImpl.Cmd_BGColor(0x00FF00);           // Set the colours for the buttons
FTImpl.Cmd_FGColor(0x0000FF);

FTImpl.TagMask(1);                      // Ensure tagging is enabled so that any subsequent items are tagged

// Button for 200mA range
FTImpl.Tag(2);                          // Set tag to 2, any subsequent items will be tagged '2'
if(CurrentRange == RANGE_200)
    FTImpl.Cmd_Button(360,230,30,20,20,FT_OPT_FLAT,"200");     // Draw button with 'pressed in' appearance
else
    FTImpl.Cmd_Button(360,230,30,20,20,0,"200");               // Otherwise draw as 'not pressed' 3D

FTImpl.Tag(8);                          // Set tag to 8, any subsequent items will be tagged '8'
if(CurrentRange == RANGE_800)
```

```
    FTImpl.Cmd_Button(400,230,30,20,20,FT_OPT_FLAT,"800");    // Draw button with 'pressed in' appearance
else
    FTImpl.Cmd_Button(400,230,30,20,20,0,"800");              // Otherwise draw 'not pressed' 3D

FTImpl.Tag(16);                        // Set tag to 16, any subsequent items will be tagged '16'
if(CurrentRange == RANGE_1600)
    FTImpl.Cmd_Button(440,230,30,20,20,FT_OPT_FLAT,"1600");   // Draw button with 'pressed in' appearance
else
    FTImpl.Cmd_Button(440,230,30,20,20,0,"1600");             // Otherwise draw 'not pressed' 3D

FTImpl.TagMask(0);                     // Disable Tag, if drawing further items avoids them being tagged

FTImpl.DLEnd();                        // End the display list
FTImpl.Finish();                       // Render the display list and wait for the completion of the DL
```

4.4.7 Check Touch

The final step in the main loop is to check if any of the areas of the screen associated with the buttons is being touched. The tagging assigned previously makes this very easy.

The register REG_TOUCH_TAG indicates if a tagged area is being touched currently. The lower 8 bits indicate the tag number of the area being touched. The code reads this register, and checks for the values 2, 8 or 16 which had been previously assigned to the 200mA, 800mA and 1600mA buttons respectively. If any of these values are present, the CurrentRange variable is set accordingly.

As the code now loops back to the beginning of the main program while() loop, this updated value of CurrentRange will be used when deciding on the chart scale, the chart data scaling and the 3D effect of the buttons the next time the screen is updated.

```
ReadWord = FTImpl.Read( REG_TOUCH_TAG);   // Read the touch tag register

if(ReadWord == 2)
    CurrentRange = RANGE_200;     // If '200mA' button touched, set range variable to show 200mA range
if(ReadWord == 8)
    CurrentRange = RANGE_800;     // If '800mA' button touched, set range variable to show 800mA range
if(ReadWord == 16)
    CurrentRange = RANGE_1600;    // If '1600mA' button touched, set range variable to show 1600mA range
```

4.4.8 Using more than one I2C Peripheral

There may be applications which require more than one I^{2}C peripheral. For example, two INA219's or an ADC plus a real-time clock. Each peripheral should be set to a different I^{2}C address using the method specified by the manufacturer (for example, the two address pins on the INA219). Then, the address of the desired I^{2}C peripheral can be used in the calls to the Wire library.

For example, writing to the config registers of two INA219's

```
I2CADDRESS = 0x40
Wire.beginTransmission(I2CADDRESS);      // INA219 has address 0x40 in this example (see INA219 datasheet)
Wire.write((char)(0x00));                // Write the register address - 00 is the INA219's config register
Wire.write((char)(0x0E));                // Write value 0x0EEF (80mV shunt gain, constant conversions)
Wire.write((char)(0xEF));                // The INA219 datasheet explains the config bits in detail
I2C_Status = Wire.endTransmission(true); // Send off these bytes over I2C with an Stop at the end

I2CADDRESS = 0x41
Wire.beginTransmission(I2CADDRESS);      // INA219 has address 0x41 in this example (see INA219 datasheet)
Wire.write((char)(0x00));                // Write the register address - 00 is the INA219's config register
Wire.write((char)(0x0E));                // Write value 0x0EEF (80mV shunt gain, constant conversions)
Wire.write((char)(0xEF));                // The INA219 datasheet explains the config bits in detail
I2C_Status = Wire.endTransmission(true); // Send off these bytes over I2C with an Stop at the end
```

Note: The VM800P has a Real-time Clock (RTC) on the I^{2}C bus at address 0x6F and so this address should not be used for external I^{2}C peripherals.

5 Conclusion

This application note has presented an example of using an external I^{2}C peripheral with the VM800P board. It has demonstrated the way in which the FTDI Arduino library can be used in conjunction with existing Arduino libraries for other peripherals and the way in which external I^{2}C peripherals can be easily connected via the expansion connectors. It has also demonstrated some useful techniques such as plotting a simple graph, appending display sections to optimize the application, and importing fonts.

The VM800P already provides the key components including display with touch, FT800, MCU with programming interface, and all required supporting circuitry. By connecting external sensors/peripherals to the expansion connector, it is possible to create a compact measurement instrument with touch screen and graphics capabilities. This example could be extended by adding different types of I^{2}C peripheral and making use of the audio and SD card which are also present on the VM800P.

6 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 8791 3570
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

Web Site

http://ftdichip.com

Appendix A – References

Document References

- FT800 datasheet
- Programming Guide covering EVE command language
- AN_240 FT800 From the Ground Up
- VM800P Datasheet
- Optimising screen updates with Macro and Append
- http://www.ti.com/product/ina219
- User guide for FT800 Arduino Library
- EVE Software Examples (see Arduino Specific Libraries)
- FTDI Utilities (Font converter, etc.)
- Custom Font Implementation

- Sample code

Acronyms and Abbreviations

Terms	Description
I^2C	Inter-IC Bus
MicroMatch	A multi-way connector used for the VM800P expansion connectors

Appendix B – List of Tables & Figures

List of Tables

List of Figures

Appendix C – Revision History

Document Title:	AN_356 FT800 Interfacing I2C Sensor to VM800P
Document Reference No.:	FT_001139
Clearance No.:	FTDI# 450
Product Page:	http://www.ftdichip.com/FTProducts.htm
Document Feedback:	Send Feedback

Revision	Changes	Date
1.0	Initial Release	2015-06-15