Application Note

AN_296

FT800 Example with Arduino Over I2C

Version 1.0

Issue Date: 2014-03-04

The FTDI FT800 video controller offers a low cost solution for embedded graphics requirements. In addition to the graphics, resistive touch inputs and an audio output provide a complete human machine interface to the outside world.

This application note will provide a simple example of developing Arduino Sketch code to control the FT800 over I²C. The principles demonstrated can then be used to produce more complex applications.

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

The FT800 operates as an SPI or I²C peripheral to a main system processor providing a low-cost, yet complete, human interface experience by the incorporation of graphics rendering, touch screen sensing and audio capabilities. It is controlled over a low-bandwidth SPI or I²C interface allowing practically any microcontroller to be used.

For this application, an Arduino board is coupled with the MikroElektronika EVE Click board FT800 module and the Arduino Sketch development environment. The target application demonstrates the use of the standard Arduino libraries to initialize and display different elements on the LCD of the VM800C. The design flow follows concepts introduced in application note AN_240 FT800 From the Ground Up.

Once these concepts are mastered, this example can be used as a basis for larger and more complex applications that the FT800 has to offer: graphics primitives (e.g. lines, shapes, text), inbuilt widgets (e.g. sliders, switches, dials), touch events and audio output.

Note: It is recommended to view the code while reading this application note.
2 Software Architecture

This project consists of two files:

- **FT800.h** – a header file containing all of the FT800-specific values for memory locations, command values, etc.
- **AN_296.ino** – the main program file

An "abstraction layer" concept was explicitly avoided in this example. Rather, direct use of the Arduino libraries demonstrates the simplicity of sending and receiving data through the FT800 while producing a graphic output.

Source code for the Arduino Sketch used in this application note can be found here: [http://www.ftdichip.com/Support/SoftwareExamples/EVE/AN_296.zip](http://www.ftdichip.com/Support/SoftwareExamples/EVE/AN_296.zip)

### 2.1 FT800.h

This header file contains all of the information that is specific to the FT800 and assigns meaningful names to each value or address. The file is separated into the following sections:

- **Memory Map** – base addresses of the sections of memory within the FT800
  - RAM_CMD = 4KB ring buffer to place Co-Processor commands
  - RAM_DL = 8KB buffer to place display lists
  - RAM_G = 256KB general element memory for images, fonts and audio data
  - RAM_PAL = 1KB color palette
  - RAM_REG = FT800 registers
- **Register Addresses**
  - Each of the FT800 registers is named and associated with its address. Refer to the **FT800 Datasheet** for register sizes.
- **Graphic Engine Commands**
  - Each command associated with the FT800 graphics engine is assigned a value according to the **FT800 Programmers Guide**. Many of the Co-Processor commands require additional arguments.
- **Display List Commands**
  - Each display list command is assigned a value according to the FT800 Programmers Guide. All display list commands are 4-bytes in length. The first byte is the actual command. The remaining three bytes contain the necessary arguments.
- **Command and register value options**
  - Assorted named values useful for the main program.
- **Useful Macros**
  - Assorted macros to perform basic calculations.

### 2.2 AN_296.ino

This is the main Arduino Sketch program file. In addition to standard C-language, the following standard Arduino libraries are used:

- **Arduino** – This is the base Arduino functionality.
- **Serial** – Serial is used to send debug messages to the UART monitor port within the Arduino environment. Serial is integral to Arduino and does not require a separate header reference. The Serial port is configured for 9600bps, no parity, 8 data bits, one stop bit and no handshake. Use of the serial port is not required, though it may be handy for debug.
- **Digital** – GPIO pins are defined for FT800 reset (PD#) and Interrupt (INT# - not used in this example). Digital is integral to Arduino and does not require a separate header reference.
- **Wire** – The Wire library is used to provide the actual clocked data into and out of the FT800 following the I²C protocol. The default rate of 100Kbps is used. Many Arduino boards can support at least 400Kbps with sufficient pull-ups on the SCL and SDA signals. The FT800 will support up to 3.4Mbps over I²C.
3 User Application

The intent of the user application is to render a circle on the LCD. The initial circle is white, then each time through the loop, the color toggles between red and white. Touch and audio are not covered.

3.1 FT800 Graphics Rendering

There are two methods available for rendering graphics elements, playing audio and sensing touch events.

- The first way is to write display list commands directly to the RAM_DL (Display List RAM). Note: This method is illustrated when blanking the screen as part of the initialization of the FT800.

- The second way is to write a series of Co-Processor commands or display list commands to the RAM_CMD (Command FIFO). The Co-Processor then creates the display list in RAM_DL based on the commands which it is given in the RAM_CMD FIFO. This method makes it easier to combine the drawing of graphics objects (lines etc.) and Widgets (slider etc.) on the same screen. Note: This method is illustrated when creating the main screen.

Although it is in theory possible to mix both methods when creating a new display list (screen) it is recommended that only one of the two methods is used in any given screen. This is because the RAM_DL would be written by both the MCU and the Co-Processor within the FT800.

3.1.1 Display List

With the Display List, the FT800 can draw several primitives (points, lines, edge and line strips, rectangles and images), manipulate the screen LCD parameters, read and write registers, etc. It can also play synthesized sounds and audio files and work with the raw touch screen activity (X-Y coordinates, touch pressure).

There is an 8K Display List buffer. All display list commands are 4-bytes in length, allowing for up to 2K commands. A display list command is constructed by logically combining the command byte with the necessary parameters. For example, this application uses the POINTS primitive. A BEGIN command is used coupled with the type of object associated with following commands - in this case, POINTS. The value for BEGIN is 0x1F, located in bits 31 through 24 of the command. Bits 23 through 4 are not used (reserved). Bits 3 through 0 indicate the primitive type. POINTS is type 2. The full command is then 0x1F000002.

Each display list command is constructed in a similar fashion. When sending these values to the FT800, they are sent in "little-endian" format, or least significant byte first: 0x02, 0x00, 0x00, 0x1F.

Commands are sent to successive locations in the RAM_DL (0x100000) memory of the FT800. The last Display List command of every list is "DISPLAY" to instruct the FT800 to process the commands and draw the screen. Processing is started when the register REG_DLSWAP is written with a value of 1 (swap after the current line) or 2 (swap after the current screen).

At this point, the list made "active", and a new list can now be written starting at RAM_DL again.
3.1.2 Co-Processor

While the Display List covers basic screen manipulation, the Co-Processor allows more advanced rendering using inbuilt widgets, memory management and touch tags. The command memory for the Co-Processor is located at RAM_CMD (0x108000). While the display list always starts at the beginning of RAM_DL, the Co-Processor uses a 4K FIFO ring buffer.

![Co-Processor Ring Buffer Diagram]

Note: This FIFO is mapped at FT800 memory addresses 108000h (RAM_CMD) to 108FFFh (RAM_CMD + 4095)

Figure 3.1 - Co-Processor Ring Buffer

The index of the last command executed is held in the register REG_CMD_READ, while the index of the last command written is in REG_CMD_WRITE. The process for adding commands is:

1) Read REG_CMD_READ and REG_CMD_WRITE. Loop here until they’re equal.
2) Write new commands starting at REG_CMD_WRITE.
3) Update REG_CMD_WRITE with the next address following the last command in the list (4-byte aligned).
4) With the new value in REG_CMD_WRITE, the Co-Processor will start executing each command and updating REG_CMD_READ until it reaches REG_CMD_WRITE.

While only one type of rendering is recommended; display list commands can be embedded into the command list. This allows mixing of the primitive graphics elements with widgets allowing the full capabilities of the FT800 to be utilized. The loop() function utilizes this method of embedding display list commands into the command list.

3.2 Arduino Sketch

A standard Arduino Sketch application consists of two standard functions: setup() and loop().

- **void setup(void)**
  - This code is executed once after reset of the Arduino.
- **void loop(void)**
  - This code is continually executed after setup() completes.
### 3.2.1 setup()

All FT800 and Arduino initialization and configuration requirements are handled in setup():

- **MCU Configuration**
  - Configures the Arduino I/O Ports and I2C Interface through the Arduino libraries.

- **Initial FT800 Configuration**
  - Powers up the FT800 and then sends commands over I^2C to configure the FT800’s oscillator settings and reset the FT800. This is followed by reading the FT800’s ID register – reading the expected value of 0x7C confirms that the FT800 is ready and responding correctly.

- **Set display setting registers**
  - Write the display registers of the FT800 to configure it for the particular display which is attached. Each register is configured with a write of a 16-bit value to its address.

- **Set touch screen and audio registers**
  - The touch screen threshold is set here. The touch screen and audio functionality is not used in this application note but some later code examples will use these features.

- **Send Display list for initial blank screen**
  - Create an initial display list which simply blanks the screen. This code writes three 4-byte commands to successive locations in the Display list RAM.
    - [RAM_DL + 0] Specify the colour which will be used when the screen is cleared
    - [RAM_DL + 4] Specify that the Colour, Stencil and Tag buffers should be cleared
    - [RAM_DL + 8] Display command which signifies the end of the Display List
  - Writing to the DL_Swap register then tells the FT800 to render the above display

- **Set FT800 GPIO to enable screen and turn on backlight**
  - The FT800 has its own GPIO port which can be controlled by writing to the FT800’s GPIO_DIR and GPIO registers over I^2C. This part of the code writes to these registers to assert the display’s enable pin which is connected to the FT800 GPIO. The FT800 backlight PWM is also ramped up to turn on the LCD panel’s LED backlight.

### 3.2.2 loop()

The generated images are shown while executing loop().

---

**Figure 3.2 - Sketch Program Output**
The Draw Screen block above performs the following steps:

1. Check the Co-Processor command buffer pointers. Wait until they're equal.
2. Alternate active colors between white and red (start with white).
3. Start the display list.
4. Clear the screen to black to eliminate any artifacts from the previous screen.
5. Clear the color, stencil and tag buffers.
6. Set the active color.
7. Define a point location and size.
8. Display the screen.
9. Swap display lists.

3.2.3 Other functions

Several functions provide basic read and write capabilities for accessing the FT800:

- void ft800memWrite8(unsigned long ftAddress, unsigned char ftData8)
- void ft800memWrite16(unsigned long ftAddress, unsigned int ftData16)
- void ft800memWrite32(unsigned long ftAddress, unsigned long ftData32)
  - This function writes a value from the Arduino through the I2C port to the FT800.
  - Available in three "sizes": 8-bits (unsigned char), 16-bits (unsigned int) and 32-bits (unsigned long).
  - The sequence of events is:
    - Send the I2C Start condition
    - Send the MEM_WRITE command combined with the 3-byte address
    - Send the data, least significant byte first
    - Send the I2C Stop condition
- unsigned char ft800memRead8(ftAddress)
- unsigned int ft800memRead16(ftAddress)
- unsigned long ft800memRead32(ftAddress)
  - This function reads a value from the FT800 through the I2C port to the Arduino.
  - Available in three "sizes": 8-bits (unsigned char), 16-bits (unsigned int) and 32-bits (unsigned long).
  - The sequence of events is:
    - Send the I2C Start condition
    - Send the MEM_READ command combined with the 3-byte address
    - Send the I2C Restart condition
    - Read the data, least significant byte first and construct the return value
    - Send the I2C Stop condition
    - Return the read data
- void ft800cmdWrite(unsigned char ftCommand)
  - This function sends specific commands from the Arduino through the I2C port to the FT800.
  - A command is one of 8 3-byte values as noted in section 4 of the FT800 Datasheet. Since the second and third bytes are always zero, an 8-bit value is passed to the function.
  - The sequence of events is:
    - Send the I2C Start condition
    - Send the command byte
    - Send two bytes with the value 0x00
    - Send the I2C Stop condition
- void incCMDOffset(unsigned int currentOffset, unsigned char commandSize)
  - This function ensures the command offset of the graphics processor command buffer rolls over within the 4kB ring buffer size.
4 Hardware

For this example, the MikroElektronika EVE Click board was used in conjunction with an Arduino Pro. The EVE click board has an LED on the pin used for the I2C Address bit A2, effectively pulling it to zero. The other two I2C address pins are not connected and internally pulled high by the FT800. The I2C base address for this connection is 0 0 1 0 A2 A1 A0, or 0x23.

Note: The VM800C and VM800B kits only support SPI operation. I2C is not available since the MODE pin of the FT800 is connected directly to ground.

Any currently available Arduino can be used to interface with the FT800. The code has been tested with the Arduino Pro.

This application uses compiler directives to select the platform type and LCD size.

```
// Set LCD display resolution here
#ifndef LCD_QVGA
#define LCD_QVGA  // QVGA = 320 x 240 (VM800B/C 3.5")
#endif

#ifndef LCD_WQVGA
#define LCD_WQVGA  // WQVGA = 480 x 272 (VM800B/C 4.3" and 5.0")
#endif

// Set Arduino platform here
#ifndef ARDUINO
#define ARDUINO  // Arduino Pro, Uno, etc. (CS# on I/O 10)
#endif
```

The connection between the Arduino board and the EVE Click board is made with five wires. 3.3V power was supplied separately to the EVE click 3.3V and GND pins since the LCD panel requires approximately 150mA:

<table>
<thead>
<tr>
<th>Signal</th>
<th>Arduino</th>
<th>EVE Click</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCL</td>
<td>Analog 5 (SCL)</td>
<td>SCLK (I2C_SCL)</td>
</tr>
<tr>
<td>SDA</td>
<td>Analog 4 (SDA)</td>
<td>MISO (I2C_SDA)</td>
</tr>
<tr>
<td>I2C_A0</td>
<td>none</td>
<td>MOSI (I2C_A0) n/c Internally pulled high by FT800</td>
</tr>
<tr>
<td>I2C_A1</td>
<td>none</td>
<td>CS# (I2C_A1) n/c Internally pulled high by FT800</td>
</tr>
<tr>
<td>I2C_A2</td>
<td>none</td>
<td>Pulled low on EVE Click board</td>
</tr>
<tr>
<td>INT#</td>
<td>Digital 3</td>
<td>INT#</td>
</tr>
<tr>
<td>PD#</td>
<td>Digital 2</td>
<td>PD#</td>
</tr>
<tr>
<td>Ground</td>
<td>GND</td>
<td>GND</td>
</tr>
</tbody>
</table>

Table 4.1 - Pin Connections
5 Conclusion

This application note presented a simple example utilizing an Arduino platform with the standard Sketch IDE and libraries to initialize the FT800. This was followed by the creation of different displays using the graphics processor commands. Low-level I2C function calls were created to allow convenient methods of sending and receiving data to the FT800.

Other display screens that contain other graphics objects, images and widgets, as well as audio output and touch events can be implemented through changing the code within the Arduino loop() function.

The I2C transfer calls could also be expanded to include more complex, multi-byte transfers useful for image and audio file transfers to and from the FT800 memory. Interrupts can also be enabled to indicate touch events, completion of display lists, etc. Any Arduino-compatible circuit can be used.
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.sales@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

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

Document References
EVE Product Page
FT800 Datasheet
FT800 Programmers Guide
VM800C Datasheet – Credit Card sized development board with FT800
VM800B Datasheet – Bezel-mounted Display with FT800
AN_240 EVE From the Ground Up
AN_259 FT800 Example with 8-bit MCU
Arduino Sketch IDE
Arduino Sketch Language Reference
Arduino Hardware
Project source code

Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Terms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS#</td>
<td>Chip Select</td>
</tr>
<tr>
<td>GND</td>
<td>Ground</td>
</tr>
<tr>
<td>HMI</td>
<td>Human-Machine Interface</td>
</tr>
<tr>
<td>I²C</td>
<td>Inter-Integrated Circuit</td>
</tr>
<tr>
<td>INT#</td>
<td>Interrupt</td>
</tr>
<tr>
<td>LCD</td>
<td>Liquid Crystal Display</td>
</tr>
<tr>
<td>MISO</td>
<td>Master In / Slave Out</td>
</tr>
<tr>
<td>MOSI</td>
<td>Master Out / Slave In</td>
</tr>
<tr>
<td>PD#</td>
<td>Power Down</td>
</tr>
<tr>
<td>SCLK</td>
<td>Synchronous Clock</td>
</tr>
<tr>
<td>SPI</td>
<td>Synchronous Peripheral Interface</td>
</tr>
</tbody>
</table>
Appendix B – List of Tables & Figures

List of Tables
Table 4.1 - Pin Connections ........................................................................................................ 8

List of Figures
Figure 3.1 - Co-Processor Ring Buffer ......................................................................................... 5
Figure 3.2 - Sketch Program Output ............................................................................................. 6
Appendix C – Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Changes</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Initial Release</td>
<td>04/03/14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>