APPLICATION NOTE

AN_269

FT_APP_SIGNATURE

Version 1.1

Document Reference No.: FT_000914

Issue Date: 2013-11-01

This document is to introduce the Signature Demo. The objective of the Demo Application is to enable users to become familiar with the usage of the FT800, the design flow, and display list used to design the desired user interface or visual effect.

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 3
1.1 Overview 3
1.2 Scope 3
2 Display Requirements 4
2.1 Signature Area 4
2.2 Button 4
2.3 Background 4
3 Design Flow 5
3.1 Signature Flowchart 6
4 Description of the Functional Blocks 7
4.1 System Intialization 7
4.2 Info() 8
4.3 Home_setup() 9
4.4 Signature() 10
4.5 Functionality 12
5 Contact Information 14
Appendix A– References 15
Document References 15
Acronyms and Abbreviations 15
Appendix B – List of Tables & Figures 16
List of Figures 16
Appendix C– Revision History 17

1 Introduction

This design example demonstrates an interactive user interface that provides a signature area, useful for payment terminals. The touch screen is used to capture handwriting inside of the signature area. The touch events are then drawn within the signature area interactively. While the signature area is active, an array of stars is drawn in a moving pattern behind the signature area. A button then clears the signature area, ready for another signature.

Overview

The document will provide information on drawing graphics elements through primitives, tagging of touch capabilities and the structure of display lists. In addition, this application note outlines the general steps of the system design flow, display list creation and integrating the display list with the system host microcontroller.

The source code for this application is provided in section 4 or at:

http://www.ftdichip.com/Support/SoftwareExamples/FT800_Projects.htm

Scope

This document can be used as a guide by designers to develop GUI applications by using FT800 with any MCU via SPI or I^2C. Note that detailed documentation is available on www.ftdichip.com/EVE.htm, including:

- FT800 datasheet
- Programming Guide covering EVE command language
- AN_240 FT800 From the Ground Up
- AN_245 VM800CB_SampleApp_PC_Introduction - covering detailed design flow with a PC and USB to SPI bridge cable
- AN_246 VM800CB_SampleApp_Arduino_Introduction – covering detailed design flow in an Arduino platform

2 Display Requirements

This section describes some of the key components of the design.

Signature Area

The signature area uses the display and touch screen. As the touch screen is written upon, the corresponding pixels in the signature area are rendered out to simulate drawing on a page. The signature area only covers a portion of the screen. Touch events outside of this area are ignored.

Button

A second touch area uses a FT800 button widget. When touched, any information drawn in the signature area is cleared out.

Background

While the signature and button areas are active, an array of animated stars is drawn behind the active touch areas.

3 Design Flow

Every EVE design follows the same basic principles as highlighted in Figure 3.1.

Select and configure your host port for controlling the FT800 then wake the device before configuring the display. The creative part then revolves around the generation of the display list, "** APPLICATION DATA **" in the figure below. There will be two lists. The active list and the updated/next list are continually swapped to render the display. Note, header files map the pseudo code of the design file of the display list to the FT800 instruction set, which is sent as the data of the SPI (or I²C) packet (typically <1KB). As a result, with EVE's object oriented approach, the FT800 is operating as an SPI peripheral while providing full display, audio, and touch capabilities.

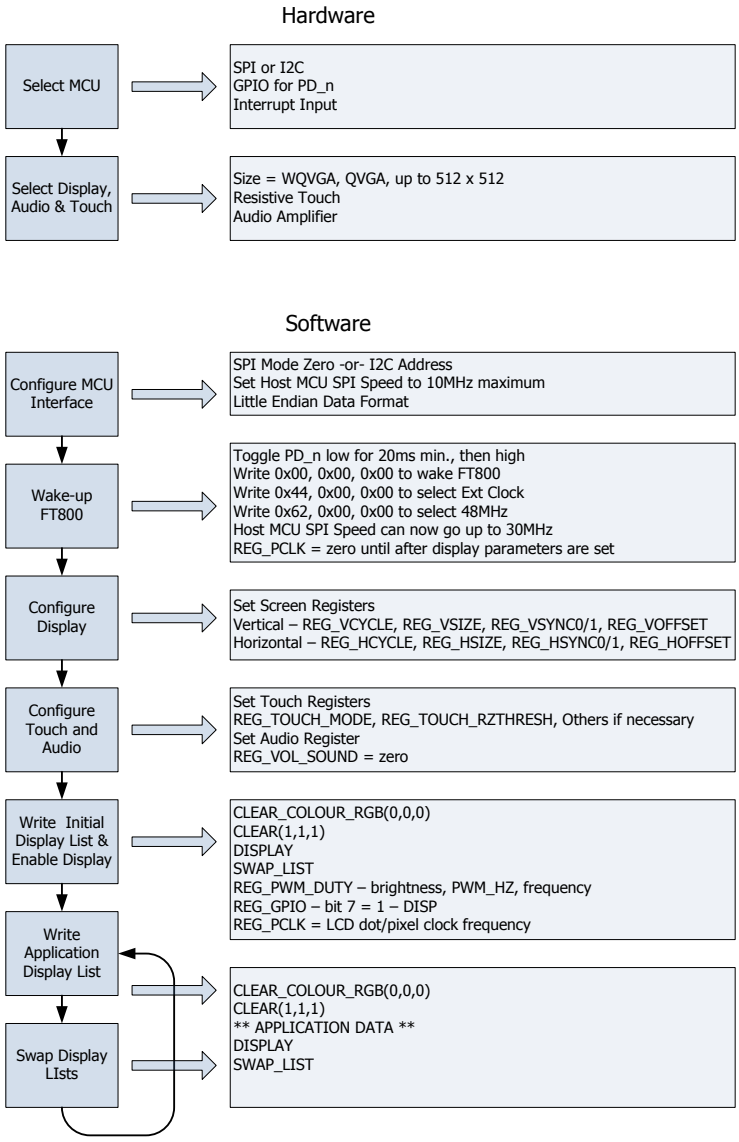

Figure 3.1 Generic EVE Design Flow

Signature Flowchart

The flowchart below is specific to the Signature application.

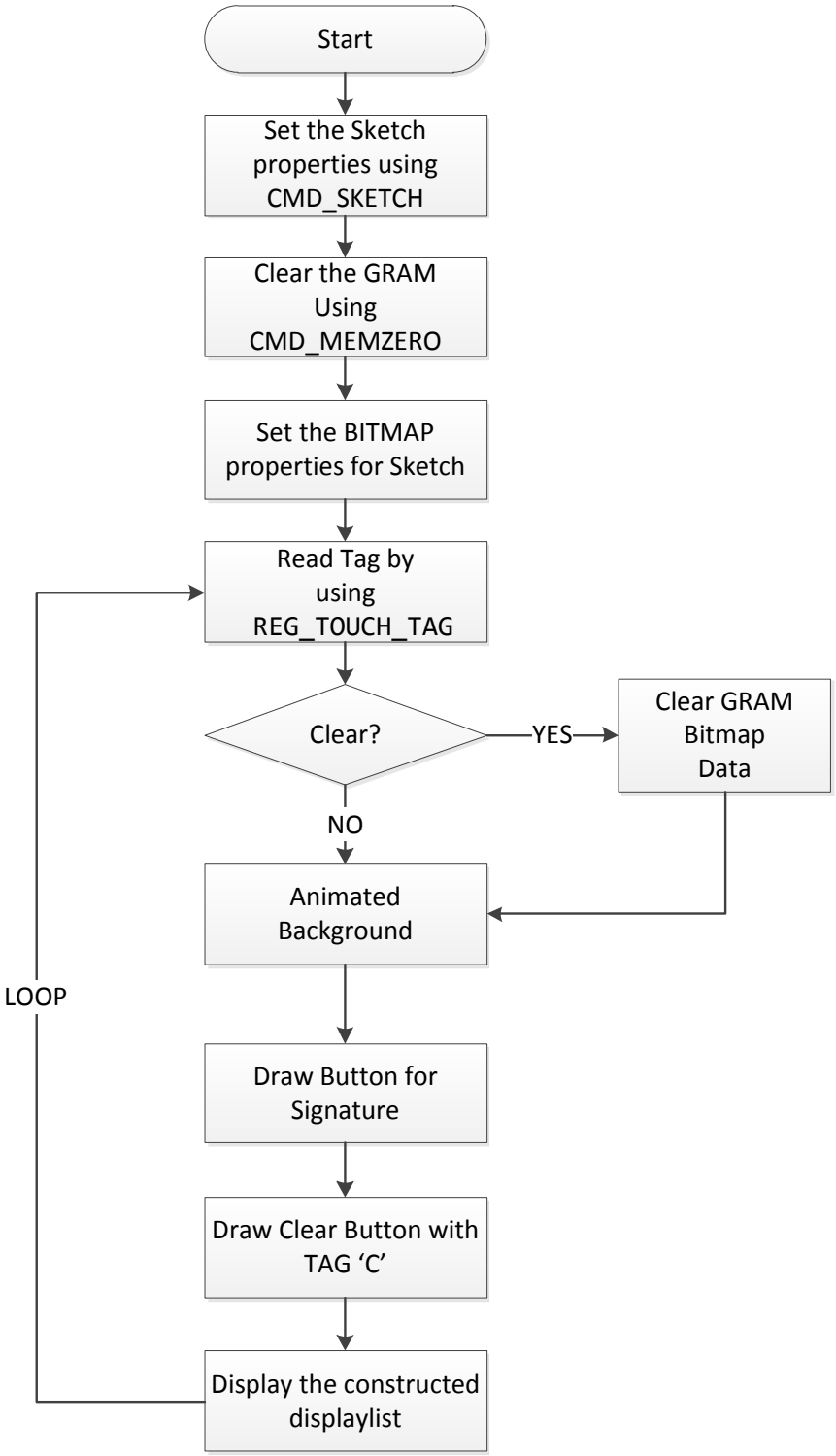

Figure 3.2 Flowchart

4 Description of the Functional Blocks

System Intialization

Configuration of the SPI master port is unique to each controller – different registers etc, but all will require data to be sent Most Significant Bit (MSB) first with a little endian format.

The function labelled Ft_BootupConfig is generic to all applications and will start by toggling the FT800 PD# pin to perform a power cycle.

```
/* Do a power cycle for safer side */
Ft_Gpu_Hal_Powercycle(phost,FT_TRUE);
Ft_Gpu_Hal_Rd16(phost,RAM_G);

/* Set the clk to external clock */
Ft_Gpu_HostCommand(phost,FT_GPU_EXTERNAL_OSC);
Ft_Gpu_Hal_Sleep(10);

/* Switch PLL output to 48MHz */
Ft_Gpu_HostCommand(phost,FT_GPU_PLL_48M);
Ft_Gpu_Hal_Sleep(10);

/* Do a core reset for safer side */
Ft_Gpu_HostCommand(phost,FT_GPU_CORE_RESET);

/* Access address 0 to wake up the FT800 */
Ft_Gpu_HostCommand(phost,FT_GPU_ACTIVE_M);
```

The internal PLL is then given a prompt by setting the clock register and PLL to 48 MHz.

Note 36MHz is possible but will have a knock on effect for the display timing parameters.

A software reset of the core is performed followed by a dummy read to address 0 to complete the wake up sequence.

The FT800 GPIO lines are also controlled by writing to registers:

```
Ft_Gpu_Hal_Wr8(phost, REG_GPIO_DIR,0x80 | Ft_Gpu_Hal_Rd8(phost,REG_GPIO_DIR));
Ft_Gpu_Hal_Wr8(phost, REG_GPIO,0x080 | Ft_Gpu_Hal_Rd8(phost,REG_GPIO));
```

And these allow the display to be enabled.

To confirm the FT800 is awake and ready to start accepting display list information the identity register is read in a loop until it reports back 0x7C. It will alsways be 0x7C if everything is awake and functioning correctly.

```
ft_uint8_t chipid;
//Read Register ID to check if FT800 is ready.
chipid = Ft_Gpu_Hal_Rd8(phost, REG_ID);
while(chipid != 0x7C)
        chipid = Ft_Gpu_Hal_Rd8(phost, REG_ID);
```

Once the FT800 is awake the display may be configured through 13 register writes according to its resolution. Resolution and timing data should be available in the display datasheet.

```
Ft_Gpu_Hal_Wr16(phost, REG_HCYCLE, FT_DispHCycle);
Ft_Gpu_Hal_Wr16(phost, REG_HOFFSET, FT_DispHOffset);
Ft_Gpu_Hal_Wr16(phost, REG_HSYNC0, FT_DispHSync0);
Ft_Gpu_Hal_Wr16(phost, REG_HSYNC1, FT_DispHSync1);
Ft_Gpu_Hal_Wr16(phost, REG_VCYCLE, FT_DispVCycle);
Ft_Gpu_Hal_Wr16(phost, REG_VOFFSET, FT_DispVOffset);
Ft_Gpu_Hal_Wr16(phost, REG_VSYNC0, FT_DispVSync0);
Ft_Gpu_Hal_Wr16(phost, REG_VSYNC1, FT_DispVSync1);
Ft_Gpu_Hal_Wr8(phost, REG_SWIZZLE, FT_DispSwizzle);
Ft_Gpu_Hal_Wr8(phost, REG_PCLK_POL, FT_DispPCLKPol);
Ft_Gpu_Hal_Wr8(phost, REG_PCLK,FT_DispPCLK);//after this display is visible on the LCD
Ft_Gpu_Hal_Wr16(phost, REG_HSIZE, FT_DispWidth);
Ft_Gpu_Hal_Wr16(phost, REG_VSIZE, FT_DispHeight);
```

To complete the configuration the touch controller should also be calibrated

```
/* Touch configuration - configure the resistance value to 1200 - this value is specific to
customer requirement and derived by experiment */
Ft_Gpu_Hal_Wr16(phost, REG_TOUCH_RZTHRESH,1200);
Ft_Gpu_Hal_Wr8(phost, REG_GPIO_DIR,0xff);
Ft_Gpu_Hal_Wr8(phost, REG_GPIO,0x0ff);
```

An optional step is present in this code to clear the screen so that no artefacts from bootup are displayed.

```
/*It is optional to clear the screen here*/
Ft_Gpu_Hal_WrMem(phost, RAM_DL,(ft_uint8_t *)FT_DLCODE_BOOTUP,sizeof(FT_DLCODE_BOOTUP));
Ft_Gpu_Hal_Wr8(phost, REG_DLSWAP,DLSWAP_FRAME);
```

Info()

This is a largely informational section of code and it starts by synchronising the physical xy coordinates of the display's touch layer with the display's visual layer.

A display list is started and cleared:

```
Ft_Gpu_CoCmd_Dlstart(phost);
Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
```

A text instruction is printed on the display followed by the call to the internal calibrate function:

```
Ft_Gpu_CoCmd_Text(phost,FT_DispWidth/2,FT_DispHeight/2,26,OPT_CENTERX|OPT_CENTERY,"Please tap
on a dot");
Ft_Gpu_CoCmd_Calibrate(phost,0);
```

The display list is then terminated and swapped to allow the changes to take effect.

```
Ft_App_WrCoCmd_Buffer(phost,DISPLAY());
Ft_Gpu_CoCmd_Swap(phost);
Ft_App_Flush_Co_Buffer(phost);
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
```

Next up in the Info() function is the FTDI logo playback:

```
Ft_Gpu_CoCmd_Logo(phost);
Ft_App_Flush_Co_Buffer(phost);
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
while(0!=Ft_Gpu_Hal_Rd16(phost,REG_CMD_READ));
dloffset = Ft_Gpu_Hal_Rd16(phost,REG_CMD_DL);
```

```
dloffset -=4;
Ft_Gpu_Hal_WrCmd32(phost,CMD_MEMCPY);
Ft_Gpu_Hal_WrCmd32(phost,100000L);
Ft_Gpu_Hal_WrCmd32(phost,RAM_DL);
Ft_Gpu_Hal_WrCmd32(phost,dloffset);
play_setup();
```

A composite image with the logo and a start arrow is then displayed to allow the user to start the main application

```
do
  {
    Ft_Gpu_CoCmd_Dlstart(phost);
    Ft_Gpu_CoCmd_Append(phost,100000L,dloffset);
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(256));
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(256));
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_B(0));
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_C(0));
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_D(0));
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_E(256));
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_F(0));
    Ft_App_WrCoCmd_Buffer(phost,SAVE_CONTEXT());
    Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(219,180,150));
    Ft_App_WrCoCmd_Buffer(phost,COLOR_A(220));
    Ft_App_WrCoCmd_Buffer(phost,BEGIN(EDGE_STRIP_A));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(0,FT_DispHeight*16));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(FT_DispWidth*16,FT_DispHeight*16));
    Ft_App_WrCoCmd_Buffer(phost,COLOR_A(255));
    Ft_App_WrCoCmd_Buffer(phost,RESTORE_CONTEXT());
    Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0,0,0));
   // INFORMATION
    Ft_Gpu_CoCmd_Text(phost,FT_DispWidth/2,20,28,OPT_CENTERX|OPT_CENTERY,info[0]);
    Ft_Gpu_CoCmd_Text(phost,FT_DispWidth/2,60,26,OPT_CENTERX|OPT_CENTERY,info[1]);
    Ft_Gpu_CoCmd_Text(phost,FT_DispWidth/2,90,26,OPT_CENTERX|OPT_CENTERY,info[2]);
    Ft_Gpu_CoCmd_Text(phost,FT_DispWidth/2,120,26,OPT_CENTERX|OPT_CENTERY,info[3]);
    Ft_Gpu_CoCmd_Text(phost,FT_DispWidth/2,FT_DispHeight-30,26,OPT_CENTERX|OPT_CENTERY,"Click
to play");
    if(sk!='P')
    Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
    else
    Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(100,100,100));
    Ft_App_WrCoCmd_Buffer(phost,BEGIN(FTPOINTS));
    Ft_App_WrCoCmd_Buffer(phost,POINT_SIZE(20*16));
    Ft_App_WrCoCmd_Buffer(phost,TAG('P'));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((FT_DispWidth/2)*16,(FT_DispHeight-60)*16));
    Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(180,35,35));
    Ft_App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2II((FT_DispWidth/2)-14,(FT_DispHeight-75),14,0));
    Ft_App_WrCoCmd_Buffer(phost,DISPLAY());
    Ft_Gpu_CoCmd_Swap(phost);
    Ft_App_Flush_Co_Buffer(phost);
    Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
  }while(Read_Keys()!='P');
```

Home_setup()

The background star image that is rotated in the background is located in line with the code. It is actually compressed data at the MCU or PC:

```
static ft_uint8_t home_star_icon[] = {0x78,0x9C,0xE5,0x94,0xBF,0x4E,0xC2,0x40,0x1C…
```

The home_setup function is run once prior to displaying the FTDI logo through Info(). It takes this data and decompresses it as it's stored in the GRAM:

```
Ft_Gpu_Hal_WrCmd32(phost,CMD_INFLATE);
Ft_Gpu_Hal_WrCmd32(phost,250*1024L);
```

```
Ft_Gpu_Hal_WrCmdBuf(phost,home_star_icon,sizeof(home_star_icon));
```

The initial display list is started by clearing the buffers and setting main colour set to white:

```
Ft_Gpu_CoCmd_Dlstart(phost);        // start
Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255, 255, 255));
```

The data for the star which was loaded with CMD_INFLATE is now assigned two handles so the images can be displayed and manipulated:

```
Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(13));    // handle for background stars
Ft_App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(250*1024L));      // Starting address in gram
Ft_App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L4, 16, 32));  // format
Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST, REPEAT, REPEAT, 512, 512  ));
Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(14));    // handle for background stars
Ft_App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(250*1024L));      // Starting address in gram
Ft_App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L4, 16, 32));  // format
Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST, BORDER, BORDER, 32, 32  ));
Ft_App_WrCoCmd_Buffer(phost,DISPLAY());
```

The cleared display is now shown. While nothing is actually drawn on the screen with this display list, the stars are now in place for later use:

```
Ft_Gpu_CoCmd_Swap(phost);
Ft_App_Flush_Co_Buffer(phost);
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
```

Signature()

The Signature function starts by calculating the size of the signature area based on the screen size:

```
ft_uint16_t w,h,x,y,tag;

ft_int16_t sw = 2 * FT_DispWidth / 3;
ft_int16_t sh = sw / 3;
ft_int16_t ox = ( FT_DispWidth - sw) / 2;
ft_int16_t oy = (2 * FT_DispHeight / 3) - sh;
ft_uint16_t a = 0;

x = FT_DispWidth*0.168;
y = FT_DispHeight*0.317;
w =  FT_DispWidth-(2*x);
h =  FT_DispHeight-(2*y);
```

The display list is started and buffers cleared.

```
Ft_Gpu_CoCmd_Dlstart(phost);        // start
Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
```

Then it clears the GRAM memory covering the size of the display with CMD_MEMZERO, followed by drawing the signature area:

```
Ft_Gpu_CoCmd_MemZero(phost,10*1024L,480L*272L); // Clear the gram from 1024
Ft_Gpu_CoCmd_Sketch(phost,x,y,w,h,10*1024L,L8);
```

At this point, the signature area is just that – there is nothing drawn on the screen to indicate the boundaries. That's done a bit later in the display list. The CMD_SKETCH command continually translates the X-Y touch coordinates within the sketch area and changes those locations in the GRAM from the foreground colour to background colour. By doing this, the image of the signature

is shown on the screen. Once complete, the GRAM corresponding to the sketch area can be read by the MCU and stored as a signature image. This example does not store any sketch data.

After enabling the signature area, the cleared memory is displayed to ensure a blank screen with no artifacts. The display list is swapped to become active and any remaining command buffer is flushed:

```
Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(1));    // handle for background stars
Ft_App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(10*1024L));
Ft_App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L8,w,h));
Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST,BORDER,BORDER,w,h));
Ft_Gpu_CoCmd_Swap(phost);
Ft_App_Flush_Co_Buffer(phost);
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
```

With the visible display initialization complete, the screen elements can now be drawn. A while() loop will create a new display list each time through. The display list will contain the moving stars and display any activity on the touch screen for both the signature area and clear button.

The clear button is assigned a touch tag. Tags remove the need correspond the X-Y touch coordinates with a drawn element. Instead, the entire element is assigned a tag number. The FT800 determines whether the touch event was inside the element boundaries then assigns the appropriate tag. This frees up considerable host MCU time.

The first item in the while() loop is to see if there are any touch events and whether a tag is assigned. If the CLEAR button is tapped, the signature area is cleared by resetting the GRAM to the original zero values:

```
tag = Ft_Gpu_Hal_Rd8(phost,REG_TOUCH_TAG);
if(tag=='0')
{
        Ft_Gpu_CoCmd_Dlstart(phost);
        Ft_Gpu_CoCmd_MemZero(phost,10*1024L,480L*272L); // Clear the gram from 1024
        Ft_App_Flush_Co_Buffer(phost);
        Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
}
```

The display list is started and buffers flushed:

```
Ft_Gpu_CoCmd_Dlstart(phost);        // start
Ft_App_WrCoCmd_Buffer(phost, CLEAR(1,1,1));
```

Next, the background stars are drawn. This is done by repeating a single star image over the entire screen (actually double the size of the screen – as if a large sheet of printed stars were laid over the screen extending beyond the edges). This is repeated a total of three times – one each for red, green and blue coloured stars. The three layers of stars are made semi-transparent and blended to allow all of the items to be seen even if they're on top of each other. Each time through the while() loop, the three layers are rotated to give the appearance of spinning sheets of stars. In addition to the spinning layers, the star images themselves are rotated.

The display context is saved and restored, similar to a microcontroller stack push and pop. This allows the foreground image to remain static while the background images are changed:

```
Ft_App_WrCoCmd_Buffer(phost,SAVE_CONTEXT());
Ft_App_WrCoCmd_Buffer(phost,BLEND_FUNC(SRC_ALPHA, ONE));
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(78, 0, 0));
Ft_App_WrCoCmd_Buffer(phost,CMD_LOADIDENTITY);
rotate_around(ox, oy, 47*a);
Ft_App_WrCoCmd_Buffer(phost,VERTEX2II(0, 0, 13, 1));

Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0, 40, 0));
rotate_around(ox + sw, oy, 53*a);
```

```
Ft_App_WrCoCmd_Buffer(phost,VERTEX2II(0, 0, 13, 1));

Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0, 0, 78));
rotate_around(ox, oy + sh, 57*a);
Ft_App_WrCoCmd_Buffer(phost,VERTEX2II(0, 0, 13, 1));
Ft_App_WrCoCmd_Buffer(phost,RESTORE_CONTEXT());
```

After displaying the new star layers and returning to the foreground, the signature area and CLEAR button are displayed. The CLEAR button is assigned a tag of '0', which is checked at the beginning of the while() loop.

```
Ft_Gpu_CoCmd_FgColor(phost,0xffffff);        // Set the fg color
Ft_Gpu_CoCmd_Button(phost,x,y,w,h,31,OPT_FLAT,"");

Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0,0,0));
Ft_App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
Ft_App_WrCoCmd_Buffer(phost,VERTEX2II(x,y,1,0));

if(tag=='0')
        Ft_Gpu_CoCmd_FgColor(phost,0x003300);
else
        Ft_Gpu_CoCmd_FgColor(phost,0x005500);
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255, 255, 255));
Ft_App_WrCoCmd_Buffer(phost,TAG('0'));
Ft_Gpu_CoCmd_Button(phost,FT_DispWidth / 2 - sw / 4, FT_DispHeight - sh / 2 - 3, sw / 2, sh /
2,  28, 0, "CLEAR");
```

The final section of the while() loop is to swap the display list to make it active, flush the buffers and return to the top of the loop. The variable 'a' is used for the star layer rotation angle in the next pass:

```
Ft_App_WrCoCmd_Buffer(phost,DISPLAY());
Ft_Gpu_CoCmd_Swap(phost);
Ft_App_Flush_Co_Buffer(phost);
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
a+=1;
```

Functionality

This application demonstrates the usage of graphics primitives such as bitmaps and inbuilt signature function, text & button widgets.

The application constantly tracks the user touch on the signature area by using sketch function and monitors the user tap on the clear button to clear the signature drawn by the user. The application displays animated star bitmaps in three layers with different colours on the background. These three layers of star bitmaps are rotated with respective to centre axis and rotary rate. The sketch area can be cleared using clear button.

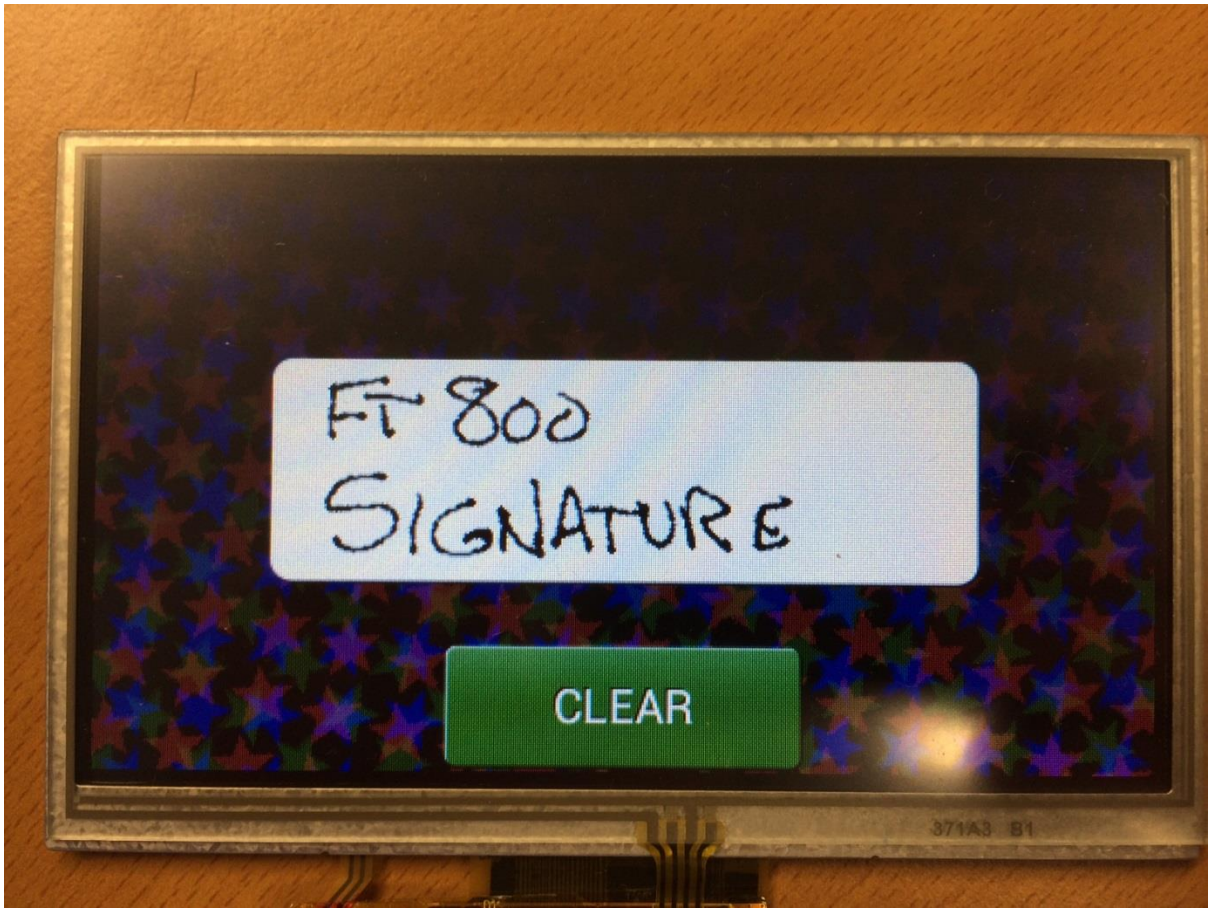

Figure 4.1 Signature Display

5 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

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

1. FT800 datasheet
2. Programming Guide covering EVE command language
3. AN_240 FT800 From the Ground Up
4. AN_245 VM800CB_SampleApp_PC_Introduction - covering detailed design flow with a PC and USB to SPI bridge cable
5. AN_246 VM800CB_SampleApp_Arduino_Introduction – covering detailed design flow in an Arduino platform
6. VM800C datasheet
7. VM800B datasheet

Acronyms and Abbreviations

Terms	Description
Arduino Pro	The open source platform variety based on ATMEL's ATMEGA chipset
EVE	Embedded Video Engine
SPI	Serial Peripheral Interface
UI	User Interface

Appendix B – List of Tables & Figures

List of Figures

Figure 3.1 Generic EVE Design Flow 5
Figure 3.2 Flowchart 6
Figure 4.1 Signature Display 13

Appendix C– Revision History

Document Title: AN_269 FT_App_Signature

Document Reference No.: FT_000914

Clearance No.: FTDI# 364

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

Document Feedback: Send Feedback

Revision	Changes	Date
0.1	Initial draft release	2013-07-18
1.0	Version 1.0 updated wrt review comments	2013-08-21
1.1	Included code discussion	2013-10-08
1.1	Version 1.1	2013-11-01