APPLICATION NOTE

AN_265

FT_App_MainMenu

Version 1.1

Document Reference No.: FT_000910

Issue Date: 2013-11-01

This document describes the operation of the Main Menu Demo Application running on Visual Studio. The Main Menu example demonstrates the way in which the FT800 JPEG Decoding feature and Tracker can be used to create user-friendly menus with icons and scrolling capabilities.

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

This example demonstrates the way in which the FT800 JPEG Decoding feature and Tracker can be used to create user-friendly menus with icons and scrolling capabilities. Please refer to the sample code project provided with this application note and available at:

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

1.1 Overview

The application produces a 'desktop' which has twelve icons/tiles on it. Each icon is created by drawing a rectangle with a bitmap image inside. Windows 8 style also adds a text string containing the name of the icon.

In a similar way to the menu systems used on portable touch-screen devices (e.g. Windows 8 and Android), the user can slide their finger on the touch screen in order to scroll along the desktop. They can tap their finger on an icon to open it.

The focus of this application note is the menu system itself. Therefore, tapping on an icon will simply open a screen displaying a single bitmap (a larger version of the same bitmap used in the icon). In a real application, selecting icons on the menu could be used to navigate to screens containing many other images, controls and displays.

Three different menu types can be selected using this sample code. These demonstrate a scrolling menu, an Android-style menu or a Windows 8 style. The same principles can also be used to create other styles of menu.

1.2 Scope

This document can be used by designers to develop GUI applications by using the FT800 with any SPI host, such as an MCU.

It covers the following topics:

- Overview of the menu styles included in the demonstration code
- Flow of the code project including the FT800 initialisation and main menu code
- Description of the menu application code
- Running the demonstration code

Additional documentation can be found at 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
- AN_252 FT800 Audio Primer

Note: This document is intended to be used along with the source code project provided in section 4 or as provided at:

http://www.ftdichip.com/Support/SoftwareExamples/FT800_Projects.htmMenu Overview

The example can display three different styles of menu. Each type provides a different visual user interface but they all work in a similar way. An example screen-shot for each of the menu types is given below.

A #define statement in the source code (FT_App_MainMenu.c file) is used to select the menu type. The project must be compiled after selecting the desired menu type.

#define ANDROID_METHOD

#define LOOPBACK_METHOD

#define WIN8_METHOD

Figure 1.1 Android style

Figure 1.2 Loopback style

Figure 1.3 Windows 8 style

2 Design Flow

2.1 Initilisation

Every EVE design follows the same basic principles as highlighted in Figure 3.1. After configuring the SPI Host itself (such as the PC through the CM232H cable, or an MCU), the application will wake up the FT800 and write to the registers in the FT800 to configure its display, touch and audio settings etc. It then writes an initial display list to clear the screen.

The main application can then create display lists to draw the actual application screens, in this case the menu system. In essence there will be two lists; the active list and the edited list which are continually swapped to update the display. Each screen can be created by either writing a display list to the RAM_DL memory in the FT800, or by writing a series of commands to the Co-Processor FIFO in the FT800 (in which case, the Co-Processor will create a display list in RAM_DL based on the commands). 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.

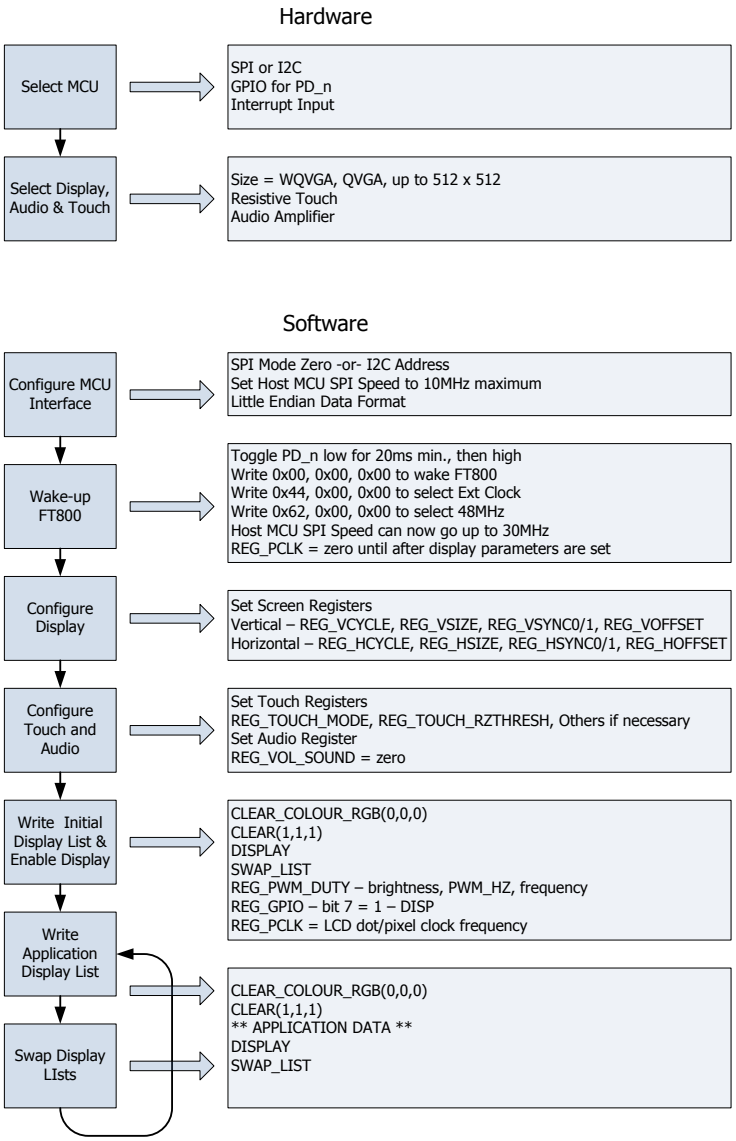

Figure 2.1 Generic EVE Design Flow

2.2 Application Flow

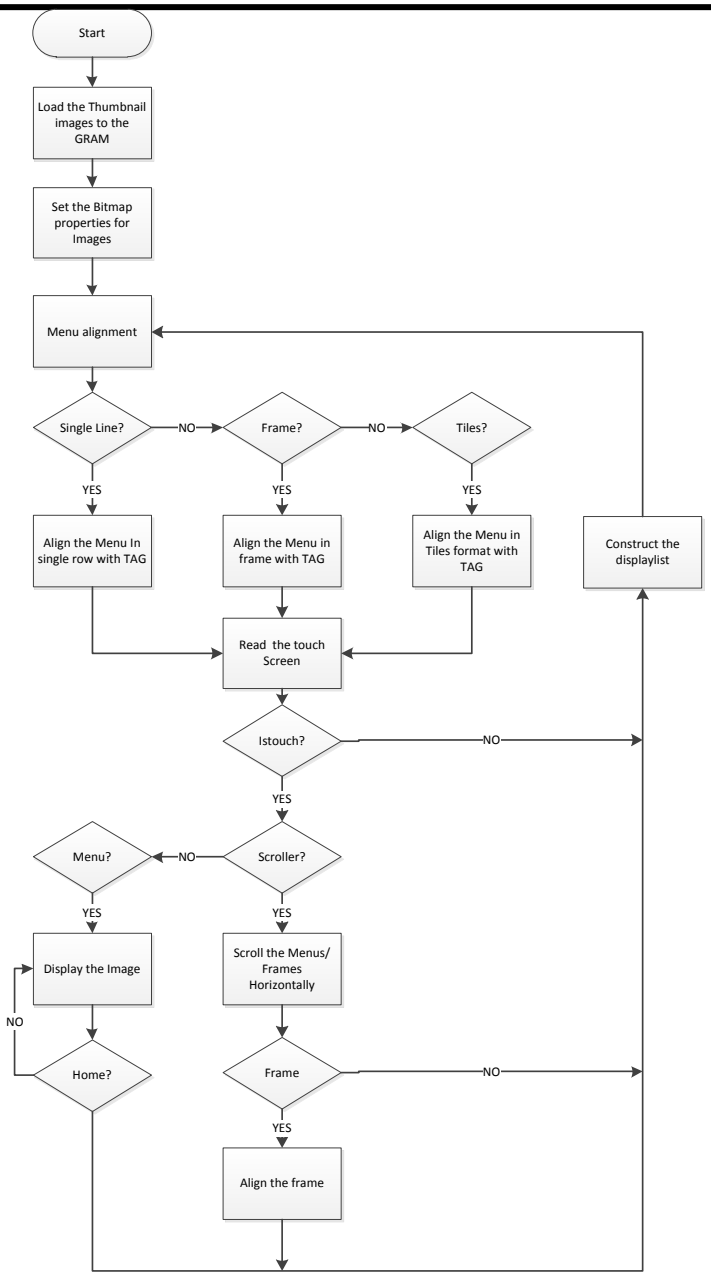

Figure 2.2 Application Flowchart

3 Description

This section describes the application code used to generate the menu displays.

3.1 System Initialisation

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 application uses the same initialisation process as the other application notes in this series.

3.2 Bootup Config

The function labelled Ft_BootupConfig in this project 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 configured by setting the clock register and PLL to 48 MHz. Note that 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 has its own GPIO lines which can be controlled by writing to registers. One of these is connected to the display's enable line and so a write to the FT800 GPIO allows the display to be enabled.

```
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));
```

To confirm that the FT800 is awake and ready to start accepting display list information, the identity register is read continuously until it reports back 0x7C. This register will always read 0x7C if the FT800 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 settings may be configured by writing 13 of the registers inside the FT800 to match the display being used. 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); // display visible on the LCD
Ft_Gpu_Hal_Wr16(phost, REG_HSIZE, FT_DispWidth);
Ft_Gpu_Hal_Wr16(phost, REG_VSIZE, FT_DispHeight);
```

The touch controller can also be configured by setting the resistance threshold.

```
/* 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);
```

An optional step is present back in the `main` program to clear the screen so that no artefacts from boot-up 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);
```

3.3 Info()

This function then provides the initial screens of the application.

A Co-Processor command list is started. The command will clear the display parameters.

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

The following commands set the colour and then print a text message to the user which tells them to tap on the dots during the following calibration routine. The FT800's built-in calibration routine is then called.

```
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
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);
```

Figure 3.1 Calibration screen

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();
```

Figure 3.2 Logo screen

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

Once the 'Click to play' button has been tapped, the application will then call one of the three menu functions, depending on which type has been defined.

```
    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');
```

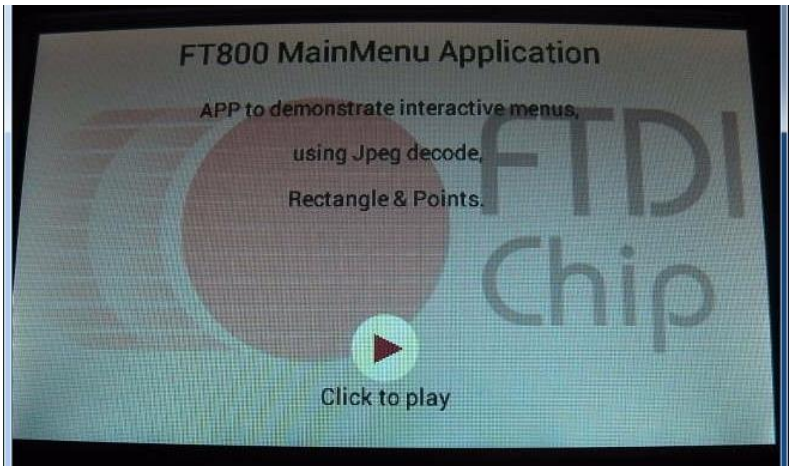

Figure 3.3 Start screen

3.4 Menu Type selection

The main application contains three separate functions, with each one providing a different style of menu. The type of menu required should be selected before compiling the code by enabling one of the three #defines at the top of the source code.

```
#ifdef ANDROID_METHOD
menu();
#endif

#ifdef LOOPBACK_METHOD
menu_loopback();
#endif

#ifdef WIN8_METHOD
menu_win8();
#endif
```

Note: The required menu type should be selected as described above before compiling the project. Although a full user application could use several menu types if required, this application can only use one at a time to avoid unnecessary complexity in the code.

3.4.1 Android Menu Style

This style is provided by the menu() function.

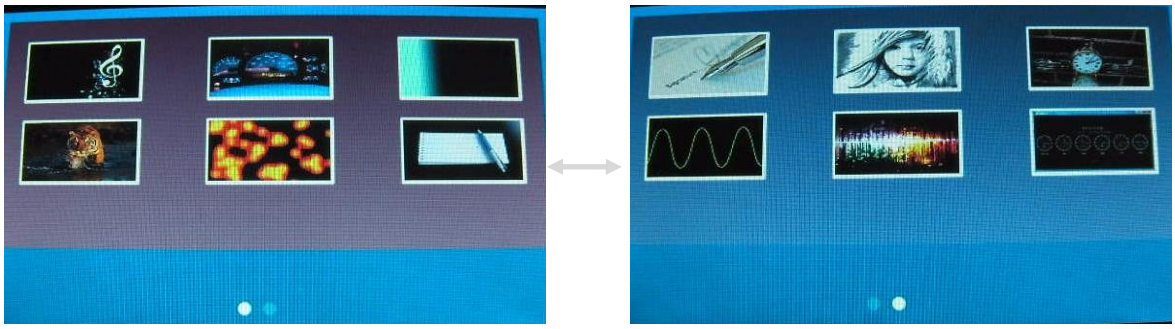

Figure 3.4 Android menu pages

The function begins by initialising the variables used to position the rectangles and bitmaps for each menu icon, and to implement the scrolling (dragging) of the screen when the user slides their finger across the screen.

```
ft_uint8_t  imh = 50, imw = 100, dt = 30, dx, dy, col, row, per_f, n_f, c_f=0,
            i, key_in=0, dg_count=0, temp=0;

ft_int16_t  Ox,Oy,sx,drag=0,prev=0,drag_dt=30,dragth=0,dragpv=0,ddt;

#define NOTOUCH          -32768

// Defining / Calculating the values used when specifying the points

ft_uint8_t  Opt,pdt =15;

dx = (dt*2)+imw;
dy = (10*2)+imh;
col = FT_DispWidth/dx;
row = 2;
per_f = col*row;
n_f = (MAX_MENUS-1)/per_f;

Opt = (FT_DispWidth-(n_f+1)*(MENU_POINTSIZE+pdt))/2;
```

The thumbnail images used within the icons are now loaded from their JPG files, and the structure which is used to hold the scrolling status is also initialised.

```
Load_Thumbnails();
scroller_init((FT_DispWidth*n_f)*16);
```

The code then enters the main `while` loop which will run continuously.

The first section reads the X value of the REG_TOUCH_SCREEN_XY register. The value of this register indicates the coordinates of the point on the screen which is currently being touched (or returns -32768 for the X and Y coordinates if the screen is not being touched). It also checks the REG_TOUCH_TAG register to determine if a tagged area is being touched.

The information obtained here can then be used to determine whether an icon is being touched to select that menu item and whether the screen is being dragged/scrolled.

In the Android-style menu, the scrolling will automatically continue until an entire page is visible even if the user removes their touch from the screen when scrolling.

```
while(1)
{
// Read touch screen x varaiation and tag
sx =  Ft_Gpu_Hal_Rd16(phost,REG_TOUCH_SCREEN_XY + 2);
key_in =  Ft_Gpu_Hal_Rd8(phost,REG_TOUCH_TAG);

// Check if any tag in
if(sx!=NOTOUCH)
{
  dg_count++;
  temp = key_in;
}

// Move into the particular frame based on dragdt
if(sx==NOTOUCH)
{
  dg_count = 0;
  if(drag>((c_f*FT_DispWidth)+drag_dt))
    drag = min((c_f+1)*FT_DispWidth,drag+15);
  if(drag<((c_f*FT_DispWidth)-drag_dt))
    drag = max((c_f-1)*FT_DispWidth,drag-15);
  if(dragth==drag) c_f = drag/FT_DispWidth;
    dragth = drag;

  scroller.vel = 0;
  scroller.base = dragth*16;          // 16bit pre
  }

  // if tag in but still pendown take a scroller basevalue
  else if(dg_count>5)
  {
    key_in = 0;
    temp = key_in;
    drag = scroller.base>>4;
  }

  if(key_in==0)
    scroller_run();
```

The code then begins to create a display list which will draw the menu. It does this by creating a Co-Processor command list which will be sent to the RAM_CMD FIFO in the FT800. The Co-Processor will then use this set of instructions to create a display list in RAM_DL which will be displayed on the screen.

```
// Display list start
Ft_Gpu_CoCmd_Dlstart(phost);
Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
Ft_App_WrCoCmd_Buffer(phost,SCISSOR_XY(0,0));
Ft_App_WrCoCmd_Buffer(phost,SCISSOR_SIZE(FT_DispWidth,FT_DispHeight));
Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
Ft_Gpu_CoCmd_Gradient(phost,0,0,0x1A99E8,0,FT_DispHeight,0x0A4F7A);
Ft_App_WrCoCmd_Buffer(phost,TAG_MASK(1));
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
Ft_App_WrCoCmd_Buffer(phost,LINE_WIDTH(25));
```

The commands below will then draw the coloured background for each 'page' of the screen, on top of which the six icons will be placed later on. The `for` loop makes the background for each page a slightly different colour. The current sample has only two pages.

```
Ft_App_WrCoCmd_Buffer(phost,BEGIN(RECTS));
Oy = 10;

for(i=0;i<=n_f;i++)
{
  Ox = 10;
  Ox+=(i*FT_DispWidth);
  Ox-=drag;
  if(i==0)
    Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(156,100,128));
  if(i==1)
    Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(100,106,156));
  if(i==2)
    Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(156,152,100));

  Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((Ox)*16,(Oy)*16));
  Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((Ox+FT_DispWidth
    -20)*16,(ft_int16_t)(FT_DispHeight*0.75)*16));
  // i pixels wide than image width +1
}
```

The white outlines are drawn for the icons on the screen.

```
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));

for(i=0;i<MAX_MENUS;i++)
{
  Ox = dt+dx*(i%col);           // Calculate the xoffsets
  Ox +=((i/per_f)*FT_DispWidth);
  Ox -= drag;
  Oy = dt+(dy*((i/col)%row));
  if(Ox > (FT_DispWidth+dt))
    0;
  else
  {
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((Ox-1)*16,(Oy-1)*16));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((imw+Ox+1)*16,(imh+Oy+1)*16));}
    // i pixels wide than image width +1
          }
```

The bitmaps are now positioned for the icons on the screen.

```
Ft_App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));

for(i=0;i<MAX_MENUS;i++)
{
  Ox = dt+dx*(i%col);           // Calculate the xoffsets
  Ox +=((i/per_f)*FT_DispWidth);
  Ox -= drag;
  Oy = dt+(dy*((i/col)%row));

  if(Ox > (FT_DispWidth+dt) || Ox < -dx)
    0;
  else
  {
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(i));
    Ft_App_WrCoCmd_Buffer(phost,TAG(i+1));
```

```
        Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(Ox*16,Oy*16));
      }
    }

    Ft_App_WrCoCmd_Buffer(phost,TAG_MASK(0));
```

The two points/dots at the bottom of the screen are now drawn. These allow the user to see which 'page' they are currently on. The colour is still set to white due to the COLOR_RGB command used when framing the bitmaps earlier. The Alpha is set to 50 to draw the semi-transparent dots to indicate the total number of pages available and the alpha level is increased to 255 to draw the solid white dot indicating the current page.

```
// frame_no_points
Ft_App_WrCoCmd_Buffer(phost,POINT_SIZE(MENU_POINTSIZE*16));
Ft_App_WrCoCmd_Buffer(phost,BEGIN(FTPOINTS));
Ft_App_WrCoCmd_Buffer(phost,COLOR_A(50));
Oy = FT_DispHeight - 20;

for(i=0;i<=n_f;i++)
{
  Ox = Opt+(i*(MENU_POINTSIZE+pdt));
  Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(Ox*16,Oy*16));
}

Ox = Opt+(c_f*(MENU_POINTSIZE+pdt));
Ft_App_WrCoCmd_Buffer(phost,COLOR_A(255));
Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(Ox*16,Oy*16));
```

The Co-Processor list is finished in the usual way by adding a Display command followed by a Swap. The Swap command will cause the FT800 to begin displaying the screen which has been created. The Flush function will send the buffer of Co-Processor commands, which have been created within a local buffer in this application by the code above, to the FT800. The WaitCmdFifo_empty will then wait for the Co-Processor to finish executing the command list before the application proceeds to create the next list.

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

The application now checks whether the user has selected any of the icons displayed in the menu. Each of the twelve icons had previously been given tags 1 to 12. If the user touches an icon to select it, then the code described earlier in this section will have recorded the relevant tag number.

In this case, another Co-Processor list is created which displays a larger version of the bitmap image which had been shown on that icon. It also draws a small 'home' icon at the top-left corner of the screen. The code then waits in a `while` loop for the tag associated with the home button to be detected, at which point it will go back to the beginning of the menu() function and re-draw the main menu.

```
key_in = temp;

if(key_in!=0 && key_in<=12 && sx==NOTOUCH)
{
  Ft_Gpu_CoCmd_Dlstart(phost);
  Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
  Ft_App_WrCoCmd_Buffer(phost,SCISSOR_XY(0,0));
  Ft_App_WrCoCmd_Buffer(phost,SCISSOR_SIZE(FT_DispWidth,FT_DispHeight));
```

```
    Ft_App_WrCoCmd_Buffer(phost,CLEAR_COLOR_RGB(0x1A,0xE8,55));
    Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
    Ft_Gpu_CoCmd_Gradient(phost,0,0,0x1A99E8,0,FT_DispHeight,0x0A4F7A);
    Ft_App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(key_in-1));
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(128));
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_E(128));
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST,BORDER,BORDER,
      200,100));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(((FT_DispWidth
      200)/2)*16,((FT_DispHeight-100)/2)*16));
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(256));
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_E(256));
    Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(14));
    Ft_App_WrCoCmd_Buffer(phost,TAG('H'));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(5*16,5*16));
    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()!='H');
      key_in = 0;
    temp = key_in;
}
```

3.4.2 Loopback Menu Style

This style is provided by the menu_loopback() function.

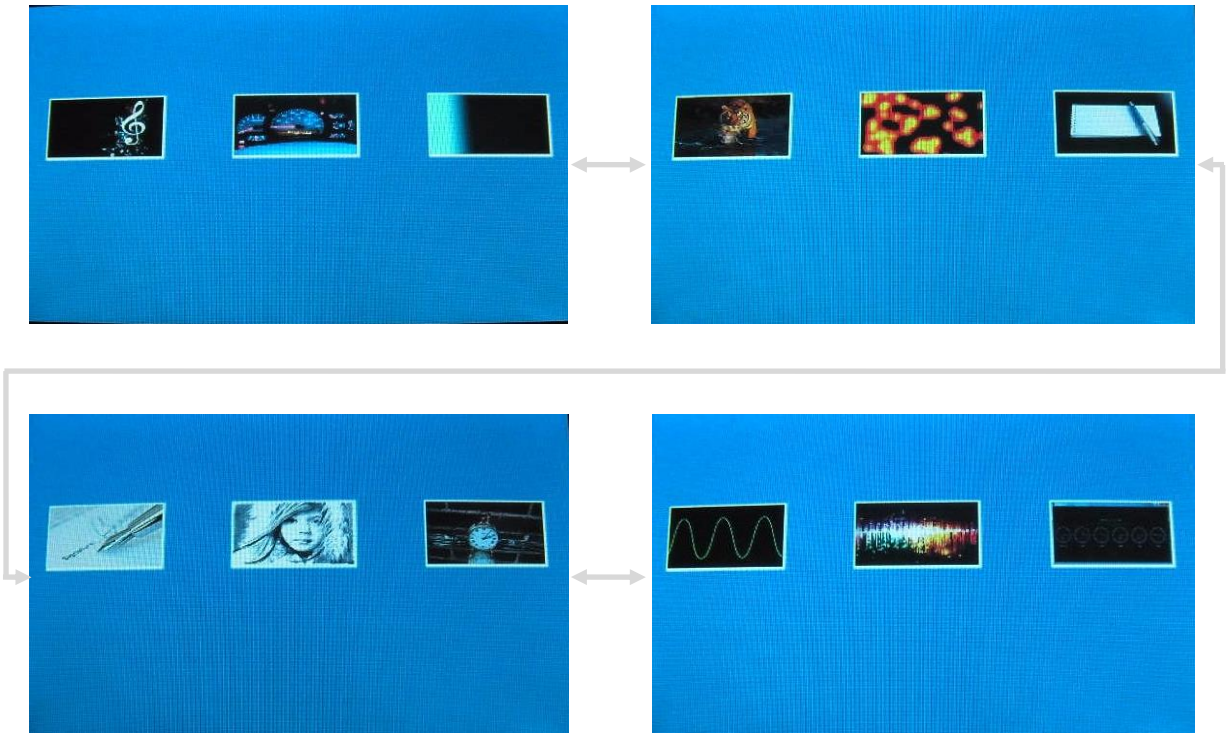

Figure 3.5 Loopback style screenshots

The function begins by initialising the variables used to position the rectangles and bitmaps for each menu icon, and to implement the scrolling (dragging) of the screen when the user slides their finger across the screen.

```
ft_uint8_t  imh = 50, imw = 100, dt = 30, dx, dy, col, row, per_f, n_f, c_f=0,
            key_in=0, dg_count=0, temp=0;

ft_int16_t Ox, Oy, sx, drag=0, prev=0, drag_dt=30, dragth=0, dragpv=0,
           temp_drag=0,i, cts=0;

// Defining / Calculating the values used when specifying the points

ft_uint8_t  Opt,pdt=15;

dx = (dt*2)+imw;
dy = (10*2)+imh;
col = FT_DispWidth/dx;
row = 1;
per_f = col*row;
n_f = (MAX_MENUS-1)/per_f;

Opt = (FT_DispWidth-(n_f+1)*(MENU_POINTSIZE+pdt))/2;
```

The thumbnail images used within the icons are now loaded from their JPG files, and the structure which is used to hold the scrolling status is also initialised.

```
Load_Thumbnails();
scroller_init((FT_DispWidth*n_f)*16);
```

The code then enters the main while loop which will run continuously.

The first section reads the X value of the REG_TOUCH_SCREEN_XY register. The value of this register indicates the coordinates of the point on the screen which is currently being touched (or returns -32768 for the X and Y coordinates if the screen is not being touched). It also checks the REG_TOUCH_TAG register to determine if a tagged area is being touched.

The information obtained here can then be used to determine whether an icon is being touched to select that menu item and whether the screen is being dragged/scrolled.

The screen will slowly stop scrolling if the user removes their touch whilst scrolling. Unlike the Android menu, the Loopback example uses continuous scrolling as opposed to having fixed 'pages' and can stop scrolling at any point. The scroller_run function allows the scrolling speed to decrease gradually so that the scrolling appears to have a smooth action.

```
while(1)
{
  // Read touch screen x varaiation and tag in
  sx =  Ft_Gpu_Hal_Rd16(phost,REG_TOUCH_SCREEN_XY + 2);
  key_in =  Ft_Gpu_Hal_Rd8(phost,REG_TOUCH_TAG);

  // Check if any tag in
  if(sx!=NOTOUCH)
  {
    dg_count++;
    temp = key_in;
  }


  // Move into the particular frame based on dragdt (30pixels)
  // to ensure that the scrolling always stops with full icons visible

  if(sx==NOTOUCH)
  {
    dg_count = 0;
    drag = scroller.base>>4;
  }

  // if tag in but still pendown take a scroller basevalue
  else if(dg_count>5)
  {
    key_in = 0;
    temp = key_in;
    drag = scroller.base>>4;
  }

  if(key_in==0)
    scroller_run();
```

The code then begins to create a display list which will draw the menu. It does this by creating a Co-Processor command list which will be sent to the RAM_CMD FIFO in the FT800. The Co-Processor will then use this set of instructions to create a display list in RAM_DL which will be displayed on the screen.

```
  Ft_CmdBuffer_Index =0;
  Ft_Gpu_CoCmd_Dlstart(phost);
```

```
Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
Ft_App_WrCoCmd_Buffer(phost,SCISSOR_XY(0,0));
Ft_App_WrCoCmd_Buffer(phost,SCISSOR_SIZE(FT_DispWidth,FT_DispHeight));
Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
Ft_Gpu_CoCmd_Gradient(phost,0,0,0x1A99E8,0,FT_DispHeight,0x0A4F7A);
Ft_App_WrCoCmd_Buffer(phost,TAG_MASK(1));
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
Ft_App_WrCoCmd_Buffer(phost,LINE_WIDTH(1*16));          // for rectangles
```

The icons are now drawn on the screen with the bitmap images inside each icon.

```
Oy = (FT_DispHeight-imw)/2;
cts = drag/dx;                  // no of items moved in +/- directions
dragth = drag%dx;

for(i=-1;i<(per_f+1);i++)
{
  Ox = dt+dx*i;
  Ox-=dragth;

  if(Ox > (FT_DispWidth+dt) || Ox < -dx)
    0;
  else
  {
    Ft_App_WrCoCmd_Buffer(phost,BEGIN(RECTS));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((Ox-1)*16,(Oy-1)*16));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((imw+Ox+1)*16,
      (imh+Oy+1)*16));// i pixels wide than image width +1
    Ft_App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
    // draw the bitmap
    Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE
     ((MAX_MENUS+i+cts)%12));
    Ft_App_WrCoCmd_Buffer(phost,TAG((1+i+cts)%(MAX_MENUS+1)));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(Ox*16,Oy*16));
  }
}
```

The Co-Processor list is finished in the usual way by adding a Display command followed by a Swap. The Swap command will cause the FT800 to begin displaying the screen which has been created. The Flush function will send the buffer of Co-Processor commands which have been created within this application above to the FT800. The WaitCmdFifo_empty function will then wait for the Co-Processor to finish executing the command list before the application proceeds to create the next list.

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

The application now checks whether the user has selected any of the icons displayed in the menu. Each of the twelve icons had previously been given tags 1 to 12. If the user touches an icon to select it, then the code described earlier in this section will have recorded the relevant tag number.

In this case, another Co-Processor list is created which displays a larger version of the bitmap image which had been shown on that icon. It also draws a small 'home' icon at the top-left corner of the screen. The code then waits for the tag associated with the home button to be detected, at which point it will go back to the beginning of the menu_loopback() function and re-draw the main menu.

```
      key_in = temp;

      if(key_in!=0 && key_in<=12 && sx==NOTOUCH)
      {
        Ft_Gpu_CoCmd_Dlstart(phost);
        Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
        Ft_App_WrCoCmd_Buffer(phost,SCISSOR_XY(0,0));
        Ft_App_WrCoCmd_Buffer(phost,SCISSOR_SIZE(FT_DispWidth,FT_DispHeight));
        Ft_App_WrCoCmd_Buffer(phost,CLEAR_COLOR_RGB(0x1A,0xE8,55));
        Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
        Ft_Gpu_CoCmd_Gradient(phost,0,0,0x1A99E8,0,FT_DispHeight,0x0A4F7A);
        Ft_App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(key_in-1));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(128));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_E(128));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST,BORDER,BORDER,
          200,100));
        Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(((FT_DispWidth
          200)/2)*16,((FT_DispHeight-100)/2)*16));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(256));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_E(256));
        Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(14));
        Ft_App_WrCoCmd_Buffer(phost,TAG('H'));
        Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(5*16,5*16));
        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()!='H');
          key_in = 0;
        temp = key_in;
      }
    }
}
```

3.4.3 Windows 8 Menu Style

This style is provided by the menu_win8() function.

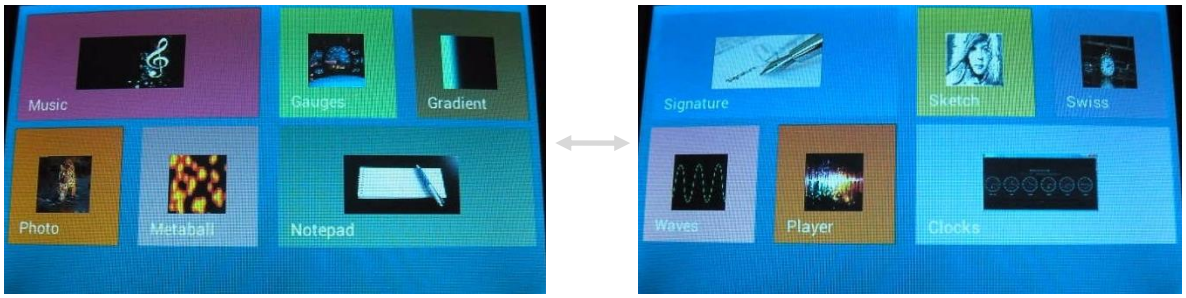

Figure 3.6 Windows 8 style screenshots

The function begins by initialising the variables used to position the rectangles and bitmaps for each menu icon, and to implement the scrolling (dragging) of the screen when the user slides their finger across the screen.

It also creates an array of the colour values which will be used for the icons and an array of the text strings which will be displayed within the icons.

```
ft_uint8_t   imh = 50,imw = 100,dt = 30,dx,dy,col,row,per_f,n_f,c_f=0,
             i,key_in=0,dg_count=0,temp=0;

 ft_int16_t Ox,Oy,sx,drag=0,prev=0,dragth=0,dragpv=0,ddt;

 ft_uint8_t color[12][3] = {     0xE0,0x01B,0xA2,
                                 0x1B,0xE0,0xA8,
                                 0x9E,0x9E,0x73,
                                 0xE0,0x8E,0x1B,
                                 0xB8,0x91,0xB3,
                                 0x6E,0x96,0x8e,
                                 0x1B,0x60,0xE0,
                                 0xC7,0xE3,0x7B,
                                 0x8B,0x1B,0xE0,
                                 0xE3,0x91,0xC1,
                                 0xE0,0x8E,0x1B,
                                 0xAC,0x91,0xE3,      };

 char *mes[20]= { "Music",  "Gauges ",  "Gradient",  "Photo",  "Metaball",
"Notepad",  "Signature",  "Sketch","Swiss","Waves","Player","Clocks"};

 // Defining / Calculating the values used when specifying the points

 ft_uint8_t  Opt,ps = 5,pdt=15;

 dx = (dt*2)+imw;
 dy = (10*2)+imh;
 col = FT_DispWidth/dx;
 row = 2;
 per_f = col*row;
 n_f = (MAX_MENUS-1)/per_f;

 Opt = (FT_DispWidth-(n_f+1)*(ps+pdt))/2;
```

The thumbnail images used within the icons are now loaded from their JPG files, and the structure which is used to hold the scrolling status is also initialised.

```
Load_Thumbnails();
scroller_init((FT_DispWidth*n_f)*16);
```

The code then enters the main while loop which will run continuously.

The first section reads the REG_TOUCH_SCREEN_XY register. The value of this register indicates the coordinates of the point on the screen which is currently being touched (or returns -32768 for the X and Y coordinates if the screen is not being touched). It also checks the REG_TOUCH_TAG register to determine if a tagged area is being touched.

The information obtained here can then be used to determine whether an icon is being touched to select that menu item and whether the screen is being dragged/scrolled.

The screen will slowly stop scrolling if the user removes their touch whilst scrolling. Unlike the Android menu, the Win8 example uses continuous scrolling as opposed to having fixed 'pages' and can stop scrolling at any point. The scroller_run function allows the scrolling speed to decrease gradually so that the scrolling appears to have a smooth action.

```
while(1)
{
  // Read touch screen x varaiation and tag
  sx =  Ft_Gpu_Hal_Rd16(phost,REG_TOUCH_SCREEN_XY + 2);
  key_in =  Ft_Gpu_Hal_Rd8(phost,REG_TOUCH_TAG);

  if(key_in!=0)
  key_in = key_in;

  if(sx!=NOTOUCH)
  {
    dg_count++;
    temp = key_in;
  }

  if(sx==NOTOUCH)
  {
     dg_count = 0;
     drag = scroller.base>>4;
  }
  else if(dg_count>5)
  {
     key_in = 0;
     temp = key_in;
     drag = scroller.base>>4;
  }

  if(key_in==0)
  {
     scroller_run();
  }
```

The code then begins to create a display list which will draw the menu. It does this by creating a Co-Processor command list which will be sent to the RAM_CMD FIFO in the FT800. The Co-Processor will then use this set of instructions to create a display list in RAM_DL which will be displayed on the screen.

```
Ft_CmdBuffer_Index =0;
Ft_Gpu_CoCmd_Dlstart(phost);
Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
Ft_App_WrCoCmd_Buffer(phost,SCISSOR_XY(0,0));
Ft_App_WrCoCmd_Buffer(phost,SCISSOR_SIZE(FT_DispWidth,FT_DispHeight));
Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
Ft_Gpu_CoCmd_Gradient(phost,0,0,0x1A99E8,0,FT_DispHeight,0x0A4F7A);
Ft_App_WrCoCmd_Buffer(phost,TAG_MASK(1));
```

```
Ft_App_WrCoCmd_Buffer(phost,CLEAR_TAG(0));
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
Ft_App_WrCoCmd_Buffer(phost,LINE_WIDTH(1*16));          // for rectangle
Ft_App_WrCoCmd_Buffer(phost,BEGIN(RECTS));
```

The code then draws the twelve icons on the screen. First of all, the four large ones (220x100) with indexes 0, 6, 5, 11 are drawn. Then the other eight smaller ones (100 x 100) with indexes 1, 2, 3, 4, 7, 8, 9, 10 are drawn.

```
imw = 220;
imh = 100;

for(i=0;i<12;i+=6)
{
  Ox = 10+FT_DispWidth*(i/6);
  Ox -= drag;
  Oy = 10+(110*((i/col)%row));
  Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(color[i][0],color[i][1],color[i][2]));
  Ft_App_WrCoCmd_Buffer(phost,TAG(i+1));
  Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((Ox)*16,(Oy)*16));
  Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((imw+Ox+1)*16,(100+Oy+1)*16));
    // i pixels wide than image width +1
  }
  for(i=5;i<12;i+=6)
  {
     Ox = 250+FT_DispWidth*(i/6);
     Ox -= drag;
     Oy = 10+(110*((i/col)%row));
     Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(color[i][0],color[i][1],color[i]
                 [2]));
     Ft_App_WrCoCmd_Buffer(phost,TAG(i+1));
     Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((Ox)*16,(Oy)*16));
     Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((imw+Ox+1)*16,(100+Oy+1)*16));
       // i pixels wide than image width +1
   }

   imw = 100;
   imh = 100;

   for(i=1;i<3;i+=1)
    {
       Ox = 250+FT_DispWidth*(i/per_f)+120*(i/2);
       Ox -= drag;
       Oy = 10+(110*((i/col)%row));
       Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(color[i][0],color[i][1],color[i]
                   [2]));
       Ft_App_WrCoCmd_Buffer(phost,TAG(i+1));
       Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((Ox)*16,(Oy)*16));
       Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((imw+Ox+1)*16,(100+Oy+1)*16));
         // i pixels wide than image width +1
    }
    for(i=3;i<5;i+=1)
    {
      Ox = 10+FT_DispWidth*(i/per_f)+120*(i/4);
      Ox -= drag;
      Oy = 10+(110*((i/col)%row));
      Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(color[i][0],color[i][1],color[i]
        [2]));
      Ft_App_WrCoCmd_Buffer(phost,TAG(i+1));
      Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((Ox)*16,(Oy)*16));
      Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((imw+Ox+1)*16,(100+Oy+1)*16));
        // i pixels wide than image width +1
    }
    for(i=7;i<9;i+=1)
```

```
    {
      Ox = 250+FT_DispWidth*(i/per_f)+120*(i/8);
      Ox -= drag;
      Oy = 10+(110*((i/col)%row));
      Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(color[i][0],color[i][1],color[i]
        [2]));
      Ft_App_WrCoCmd_Buffer(phost,TAG(i+1));
      Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((Ox)*16,(Oy)*16));
      Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((imw+Ox+1)*16,(100+Oy+1)*16));
       // i pixels wide than image width +1
     }
    for(i=9;i<11;i+=1)
    {
      Ox = 10+FT_DispWidth*(i/per_f)+120*(i/10);
      Ox -= drag;
      Oy = 10+(110*((i/col)%row));
      Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(color[i][0],color[i][1],color[i]
            [2]));
      Ft_App_WrCoCmd_Buffer(phost,TAG(i+1));
      Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((Ox)*16,(Oy)*16));
      Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((imw+Ox+1)*16,(100+Oy+1)*16));
       // i pixels wide than image width +1
    }
```

The text is then added by taking the relevant string from the array which was created earlier. The colour is set to white before drawing the text.

```
Ft_App_WrCoCmd_Buffer(phost,TAG_MASK(0));

Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
for(i=0;i<12;i+=6)
{
  Ox = 10+FT_DispWidth*(i/4);
  Ox -= drag;
  Oy = 10+(110*((i/col)%row));
  Ft_Gpu_CoCmd_Text(phost,Ox+10,Oy+80,26,0,mes[i]);
}
for(i=5;i<12;i+=6)
{
  Ox = 250+FT_DispWidth*(i/6);
  Ox -= drag;
  Oy = 10+(110*((i/col)%row));
  Ft_Gpu_CoCmd_Text(phost,Ox+10,Oy+80,26,0,mes[i]);
}
for(i=1;i<3;i+=1)
{
  Ox = 250+FT_DispWidth*(i/per_f)+120*(i/2);
  Ox -= drag;
  Oy = 10+(110*((i/col)%row));
  Ft_Gpu_CoCmd_Text(phost,Ox+10,Oy+80,26,0,mes[i]);
}
for(i=3;i<5;i+=1)
{
  Ox = 10+FT_DispWidth*(i/per_f)+120*(i/4);
  Ox -= drag;
  Oy = 10+(110*((i/col)%row));
  Ft_Gpu_CoCmd_Text(phost,Ox+10,Oy+80,26,0,mes[i]);
}
for(i=7;i<9;i+=1)
{
  Ox = 250+FT_DispWidth*(i/per_f)+120*(i/8);
  Ox -= drag;
  Oy = 10+(110*((i/col)%row));
  Ft_Gpu_CoCmd_Text(phost,Ox+10,Oy+80,26,0,mes[i]);
```

```
  }
  for(i=9;i<11;i+=1)
  {
    Ox = 10+FT_DispWidth*(i/per_f)+120*(i/10);
    Ox -= drag;
    Oy = 10+(110*((i/col)%row));
   Ft_Gpu_CoCmd_Text(phost,Ox+10,Oy+80,26,0,mes[i]);
  }
```

The bitmaps are now positioned within the coloured blocks (icons) on the screen. The transform command is used to halve the size of the images for the smaller blocks.

```
// Draw the bitmaps

Ft_App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
for(i=0;i<12;i+=6)
{
  Ox = 65+FT_DispWidth*(i/6);
  Ox -= drag;
  Oy = 35+(110*((i/col)%row));
  Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(i));
  Ft_App_WrCoCmd_Buffer(phost,CLEAR_TAG(255));
  Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(Ox*16,Oy*16));
}
for(i=5;i<12;i+=6)
{
  Ox = 305+FT_DispWidth*(i/6);
  Ox -= drag;
  Oy = 35+(110*((i/col)%row));
  Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(i));
  Ft_App_WrCoCmd_Buffer(phost,CLEAR_TAG(255));
  Ft_App_WrCoCmd_Buffer(phost,TAG(i+1));
  Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(Ox*16,Oy*16));
}

Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(512));

for(i=1;i<3;i+=1)
{
  Ox = 275+FT_DispWidth*(i/per_f)+120*(i/2);
  Ox -= drag;
  Oy = 35+(110*((i/col)%row));
  Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(i));
  Ft_App_WrCoCmd_Buffer(phost,CLEAR_TAG(255));
  Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(Ox*16,Oy*16));
  // i pixels wide than image width +1
}
for(i=3;i<5;i+=1)
{
  Ox = 35+FT_DispWidth*(i/per_f)+120*(i/4);
  Ox -= drag;
  Oy = 35+(110*((i/col)%row));
  Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(i));
  Ft_App_WrCoCmd_Buffer(phost,CLEAR_TAG(255));
  Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(Ox*16,Oy*16));
   // i pixels wide than image width +1
}
for(i=7;i<9;i+=1)
{
  Ox = 275+FT_DispWidth*(i/per_f)+120*(i/8);
  Ox -= drag;
  Oy = 35+(110*((i/col)%row));
  Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(i));
  Ft_App_WrCoCmd_Buffer(phost,CLEAR_TAG(255));
```

```
      Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(Ox*16,Oy*16));
       // i pixels wide than image width +1
    }
    for(i=9;i<11;i+=1)
    {
      Ox = 35+FT_DispWidth*(i/per_f)+120*(i/10);
      Ox -= drag;
      Oy = 35+(110*((i/col)%row));
      Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(i));
      Ft_App_WrCoCmd_Buffer(phost,CLEAR_TAG(255));
      Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(Ox*16,Oy*16));
       // i pixels wide than image width +1
    }

    Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(256));
```

The Co-Processor list is finished in the usual way by adding a Display command followed by a Swap. The Swap command will cause the FT800 to begin displaying the screen which has been created. The Flush function will send the buffer of Co-Processor commands which have been created within this application above to the FT800. The WaitCmdFifo_empty function will then wait for the Co-Processor to finish executing the command list before the application proceeds to create the next list.

```
    Ft_App_WrCoCmd_Buffer(phost,TAG_MASK(0));

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

The application now checks whether the user has selected any of icons displayed in the menu. Each of the twelve icons had previously been given tags 1 to 12. If the user touches an icon to select it, then the code described earlier in this section will have recorded the relevant tag number.

In this case, another Co-Processor list is created which displays a larger version of the bitmap image which had been shown on that icon. It also draws a small 'home' icon at the top-left corner of the screen. The code then waits for the tag associated with the home button to be detected, at which point it will go back to the beginning of the menu_win8() function and re-draw the main menu.

```
  key_in = temp;
    if(key_in!=0 && key_in<=12 && sx==NOTOUCH)
    {
      Ft_CmdBuffer_Index =0;
      Ft_Gpu_CoCmd_Dlstart(phost);
      Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
      Ft_App_WrCoCmd_Buffer(phost,SCISSOR_XY(0,0));
      Ft_App_WrCoCmd_Buffer(phost,SCISSOR_SIZE(FT_DispWidth,FT_DispHeight));
      Ft_App_WrCoCmd_Buffer(phost,CLEAR_COLOR_RGB(0x1A,0xE8,55));
      Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
      Ft_Gpu_CoCmd_Gradient(phost,0,0,0x1A99E8,0,FT_DispHeight,0x0A4F7A);
      Ft_App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
      Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(key_in-1));
      Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(128));
      Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_E(128));
      Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST,BORDER,BORDER,200,100));
      Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(((FT_DispWidth-
        200)/2)*16,((FT_DispHeight-100)/2)*16));
      Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(256));
      Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_E(256));
      Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
      Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(14));
      Ft_App_WrCoCmd_Buffer(phost,TAG('H'));
```

```
        Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(5*16,5*16));
        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()!='H');
          key_in = 0;
        temp = key_in;
      }
    }
}
```

4 Running the demonstration code

This example is shown here when running on a PC with Visual Studio (C++) installed. The FT800 development module (VM800B/VM800C) is connected to the PC using the C232HM cable which acts as a USB to SPI converter.

SCK	ORANGE
MOSI	YELLOW
MISO	GREEN
CS#	BROWN
PD#	BLUE
GND	**BLACK**

Table 1 CM232H Connections to the VM800 pins

The code can now be compiled and run. As discussed previously, the menu type depends on the #define statement used when compiling the code. After setting this, the debug button can be used to start the application.

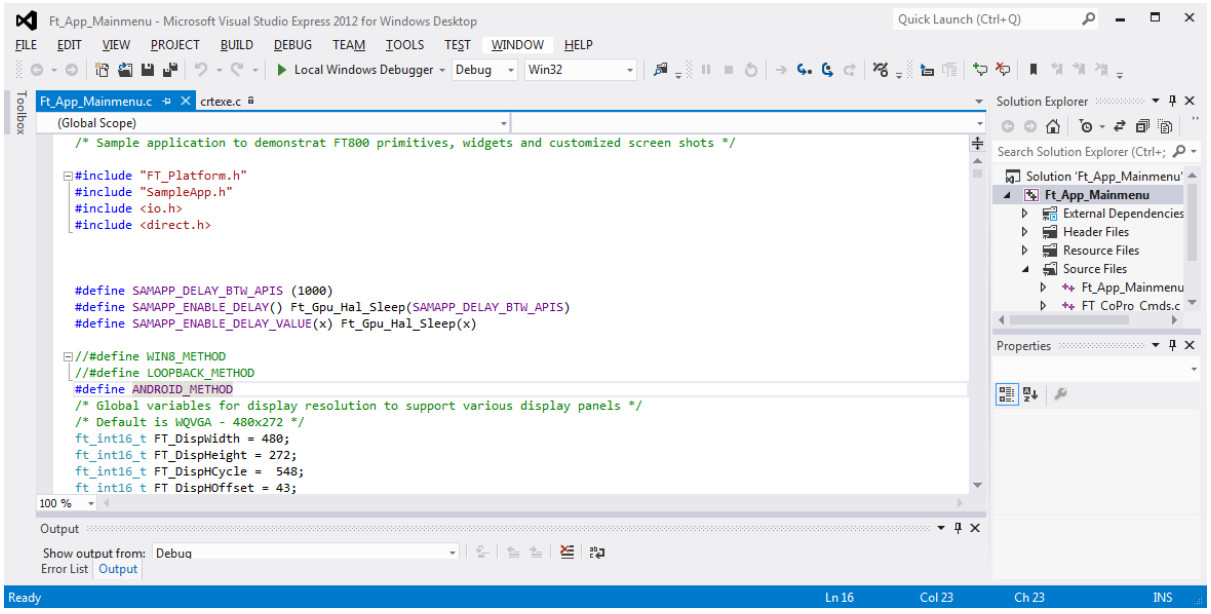

Figure 4.1 Visual Studio screenshot

When running the application, the calibration screen will be displayed first. This uses the FT800's built-in calibration routine. It ensures that the FT800 can align inputs from the touch panel to the image on the screen below accurately. The routine will display a dot and ask the user to tap on this dot. It will then repeat this twice more (with the dot at a different location on the screen in each case).

Figure 4.2 Calibration screen

The FTDI logo animation will then appear on the screen (not shown here).

The MainMenu introduction screen is then displayed and the application waits for the 'Click to play' button to be pressed, before loading the main menu screen.

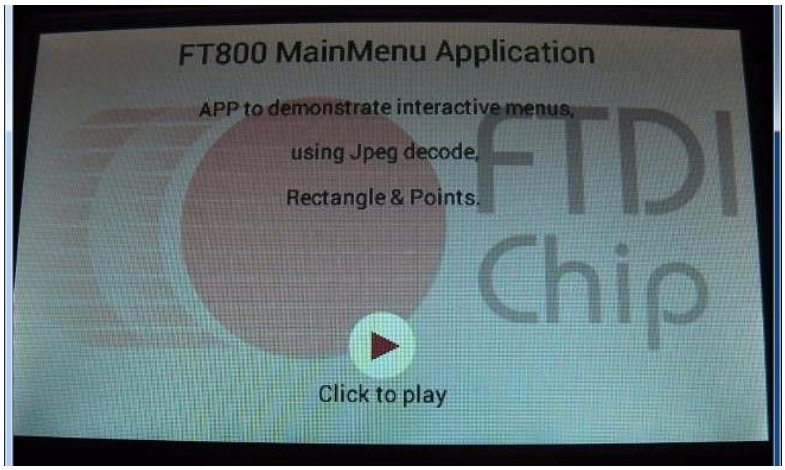

Figure 4.3 Introduction screen

The menu will now be displayed, for example in the LoopBack style below.

Figure 4.4 Main menu (loopback style)

Holding a finger on the screen and dragging the finger to the left or right whilst keeping it down will allow scrolling through the menu, in a similar way to most mobile touch screen devices.

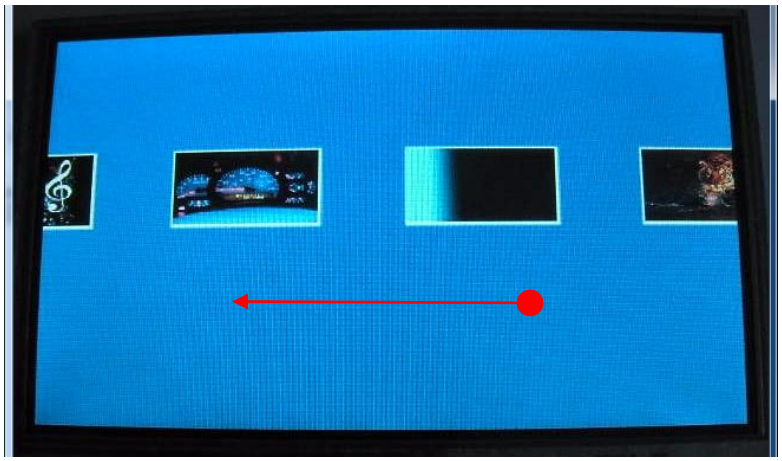

Figure 4.5 Scrolling along the menu

A short tap on an icon will select that menu option and open the associated bitmap file. When the bitmap is open, a home button can be tapped to return to the main screen.

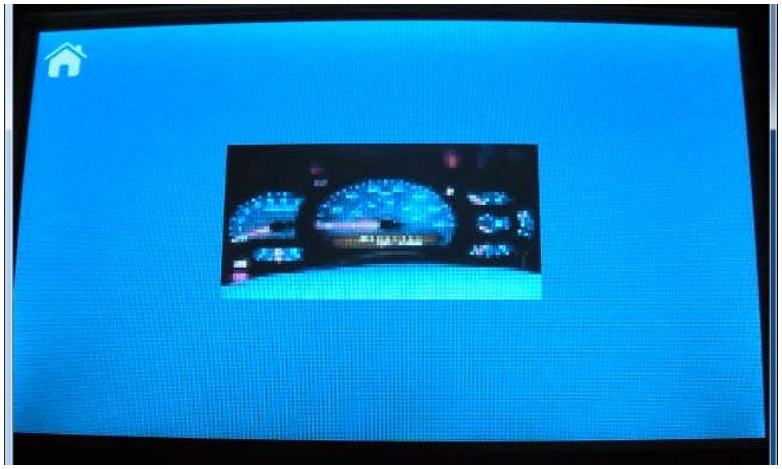

Figure 4.6 After selecting Gauges from the menu

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. Datasheet for VM800C
2. Datasheet for VM800B
3. FT800 programmer guide FT_000793.
4. FT800 Embedded Video Engine Datasheet FT_000792

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
USB	Universal Serial Bus

Appendix B – List of Tables & Figures

List of Figures

Appendix C– Revision History

Document Title: AN_265 FT_App_MainMenu

Document Reference No.: FT_000910

Clearance No.: FTDI# 360

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	Version 1.1	2013-11-01