APPLICATION NOTE

AN_265

FT_App_MainMenu

Version 1.2

Document Reference No.: FT_000910

Issue Date: 2014-07-09

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.htm

2 Menu 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
```

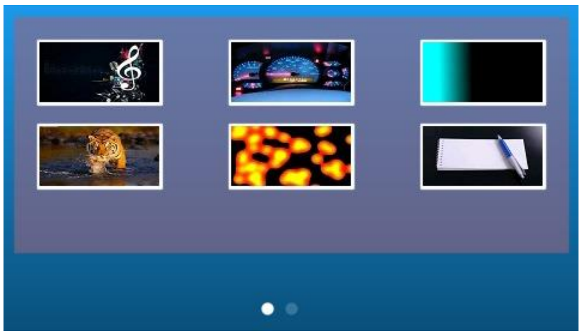


Figure 2.1 Android style

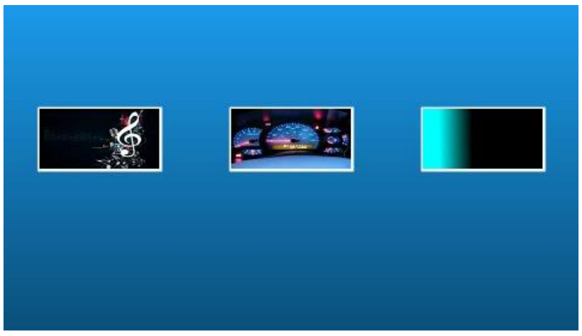


Figure 2.2 Loopback style

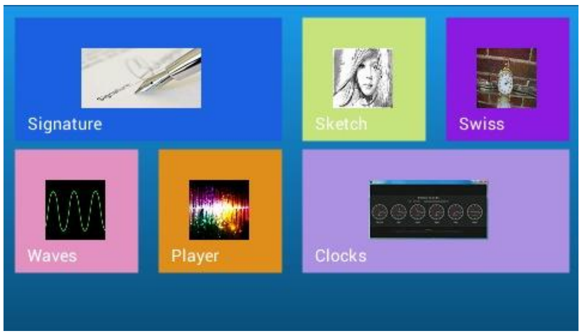


Figure 2.3 Windows 8 style

3 Design Flow

3.1 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.

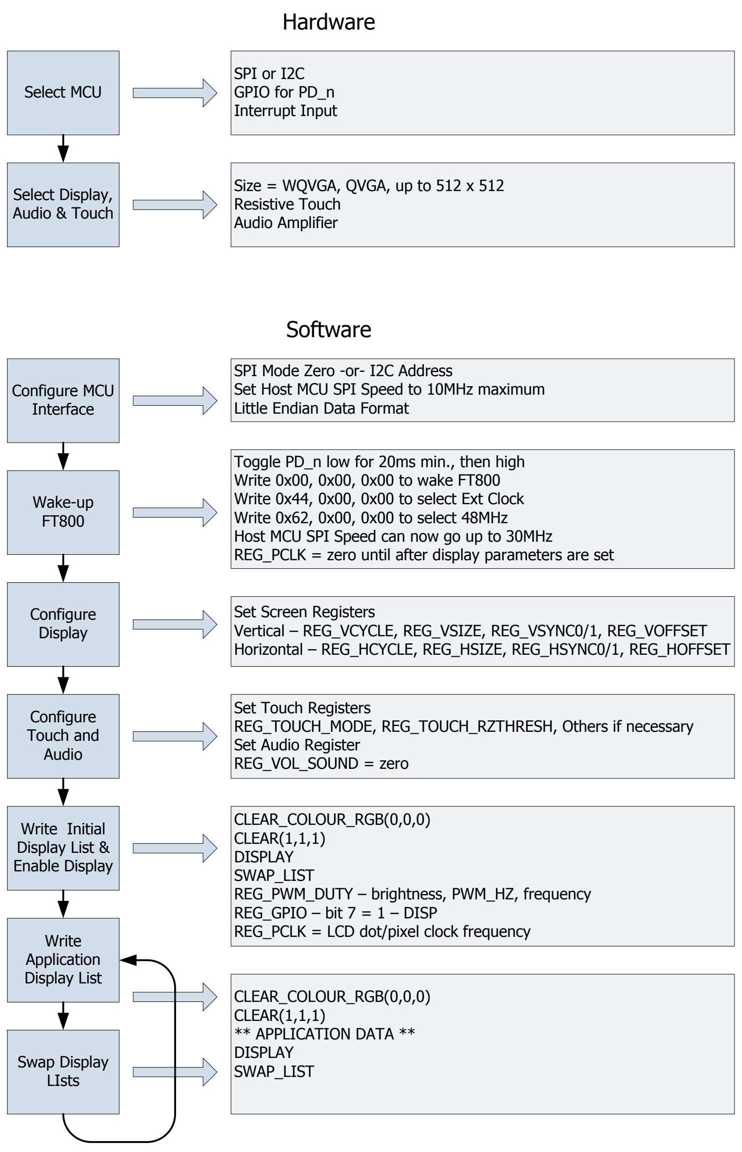

Hardware
Select MCU
SPI or I2C
GPIO for PD_n
Interrupt Input
Select Display, Audio & Touch
Size = WQVGA, QVGA, up to 512 x 512
Resistive Touch
Audio Amplifier
Software
Configure MCU Interface
SPI Mode Zero -or- I2C Address
Set Host MCU SPI Speed to 10MHz maximum
Little Endian Data Format
Wake-up FT800
Toggle PD_n low for 20ms min., then high
Write 0x00, 0x00, 0x00 to wake FT800
Write 0x44, 0x00, 0x00 to select Ext Clock
Write 0x62, 0x00, 0x00 to select 48MHz
Host MCU SPI Speed can now go up to 30MHz
REG_PCLK = zero until after display parameters are set
Configure Display
Set Screen Registers
Vertical – REG_VCYCLE, REG_VSIZE, REG_VSYNC0/1, REG_VOFFSET
Horizontal – REG_HCYCLE, REG_HSIZE, REG_HSYNC0/1, REG_HOFFSET
Configure Touch and Audio
Set Touch Registers
REG_TOUCH_MODE, REG_TOUCH_RZTHRESH, Others if necessary
Set Audio Register
REG_VOL_SOUND = zero
Write Initial Display List & Enable Display
CLEAR_COLOUR_RGB(0,0,0)
CLEAR(1,1,1)
DISPLAY
SWAP_LIST
REG_PWM_DUTY – brightness, PWM_HZ, frequency
REG_GPIO – bit 7 = 1 – DISP
REG_PCLK = LCD dot/pixel clock frequency
Write Application Display List
Swap Display LIsts
CLEAR_COLOUR_RGB(0,0,0)
CLEAR(1,1,1)
** APPLICATION DATA **
DISPLAY
SWAP_LIST

Figure 3.1 Generic EVE Design Flow

3.2 Application Flow

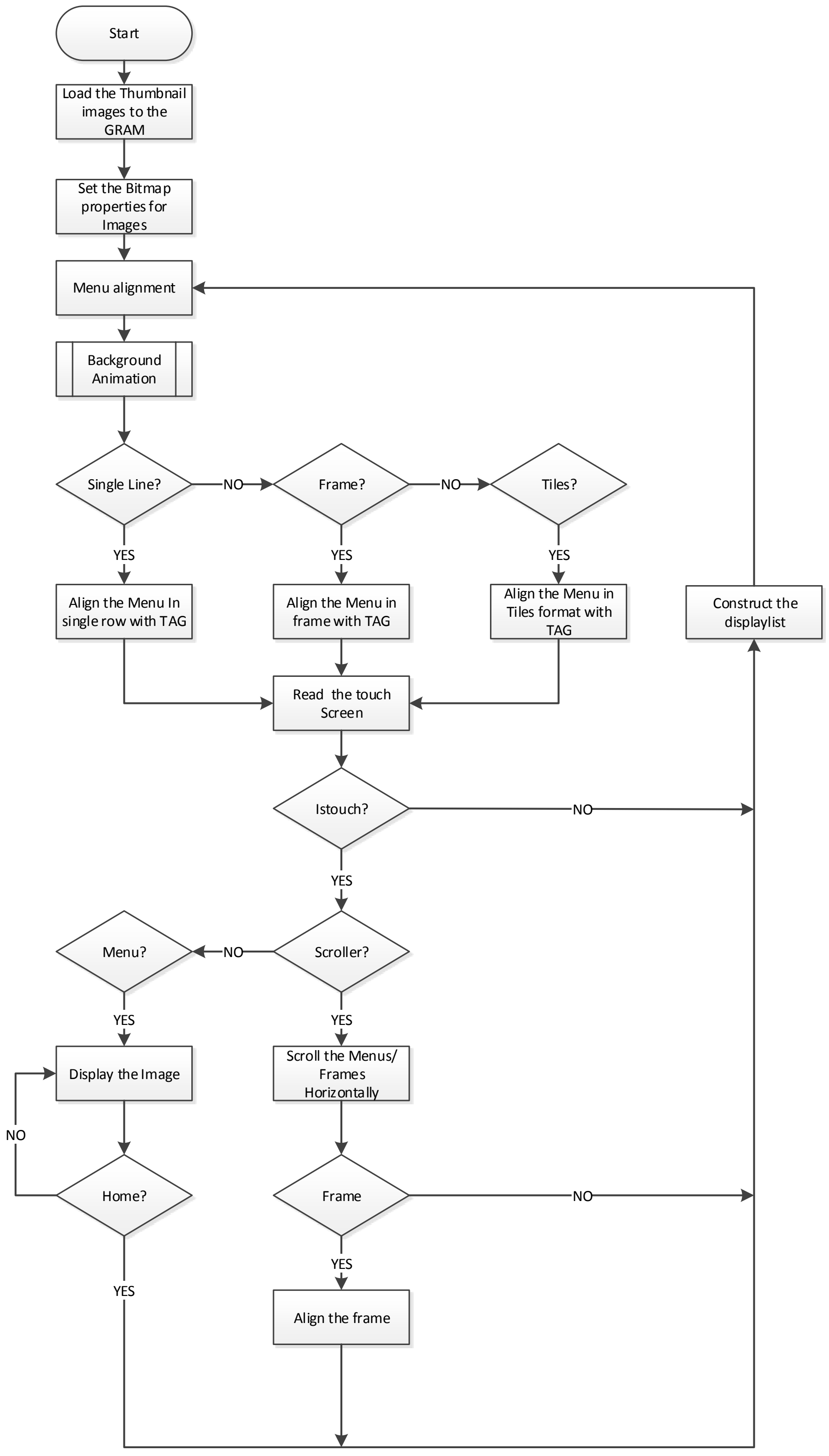

Start
Load the Thumbnail images to the GRAM
Set the Bitmap properties for Images
Menu alignment
Background Animation
Single Line?
NO
Frame?
NO
Tiles?
YES
YES
YES
Align the Menu In single row with TAG
Align the Menu in frame with TAG
Align the Menu in Tiles format with TAG
Construct the displaylist
Read the touch Screen
Istouch?
NO
YES
Menu?
NO
Scroller?
YES
YES
Display the Image
Scroll the Menus/ Frames Horizontally
NO
Home?
Frame
NO
YES
YES
Align the frame

Figure 3.2 Application Flowchart

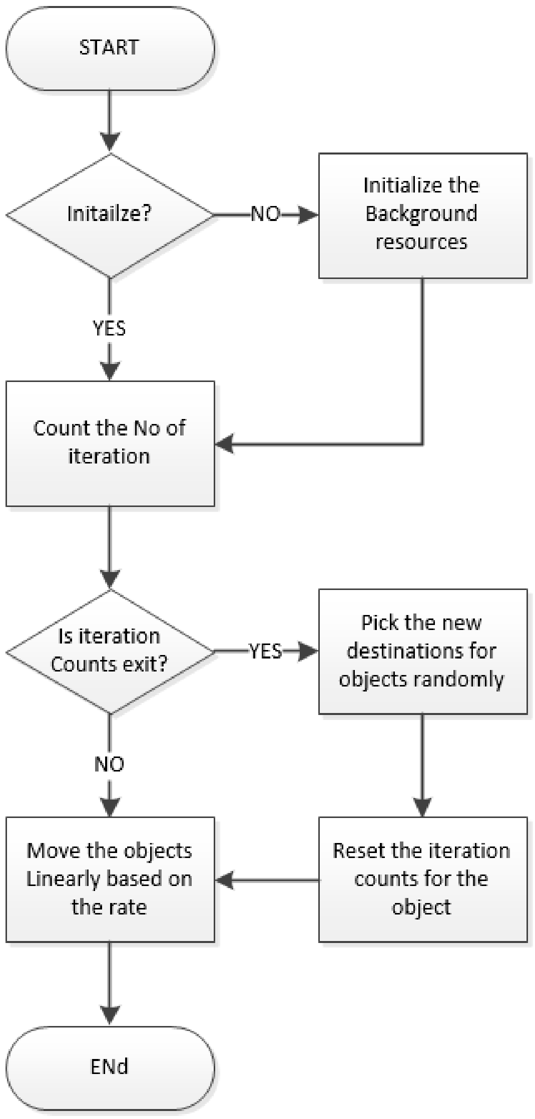

START
Initailze?
NO
Initialize the Background resources
YES
Count the No of iteration
Is iteration Counts exit?
YES
Pick the new destinations for objects randomly
NO
Move the objects Linearly based on the rate
Reset the iteration counts for the object
ENd

Figure 3.3 Background Flowchart

4 Description

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

4.1 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.

4.2 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 secure startup */
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 secure startup */
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 also available 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);
```

4.3 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);
```

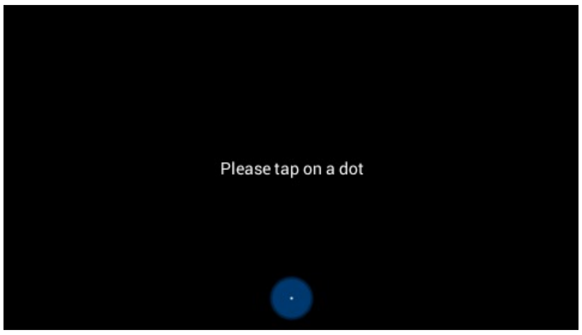


Figure 4.1 Calibration screen

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();
```

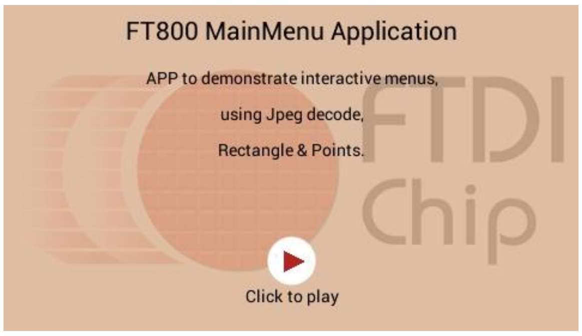
Figure 4.2 Logo screen

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');
```


FT800 MainMenu Application

APP to demonstrate interactive menus,

using Jpeg decode,

Rectangle & Points.

Click to play

Figure 4.3 Start screen

4.4 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.

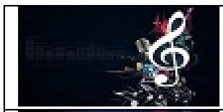
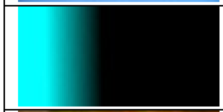
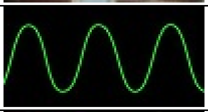
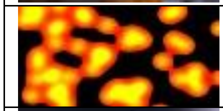
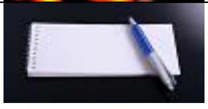
```
#ifdef ANDROID_METHOD
menu();
#endif

#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.

4.4.1 Images into GRAM

12 images are used for the mainmenu application. All the images are common for three types of mainmenu. Image size 100x50 and RGB565 format.So the actual size of each image in the GRAM is 100*2*50 = 10000 bytes.In the application for all the images we are using same bitmap handle but different cells. Because all the images are same size and format, all the images occupy the same amount of memory. Make sure all the addresses of the images in the GRAM are back to back For example, 1st image in the location 0 then the 2nd image location should be 10001 because the size of the image 10000.

Table 1 Icon Cells

	Cell – 0		Cell - 6
	Cell - 1		Cell - 7
	Cell - 2		Cell - 8
	Cell – 3		Cell - 9
	Cell - 4		Cell – 10
	Cell - 5		Cell - 11

4.4.2 Android Menu Style

This style is provided by the menu() function.

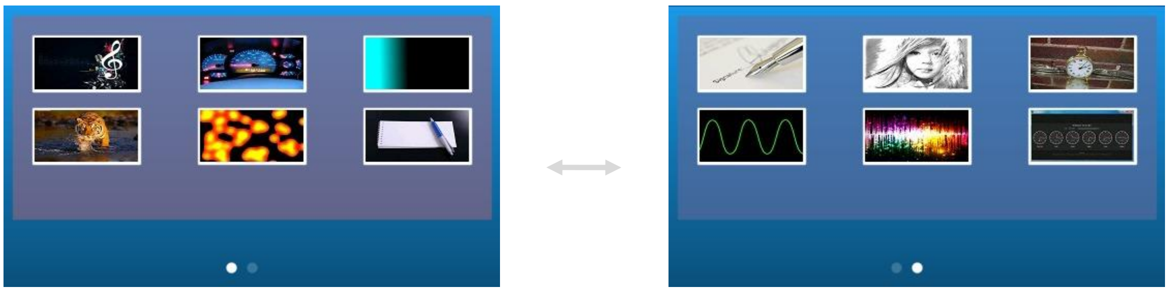


Figure 4.4 Android menu pages

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);
  sd -= 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;
   }
```

4.4.3 Loopback Menu Style

This style is provided by the menu_loopback() function.

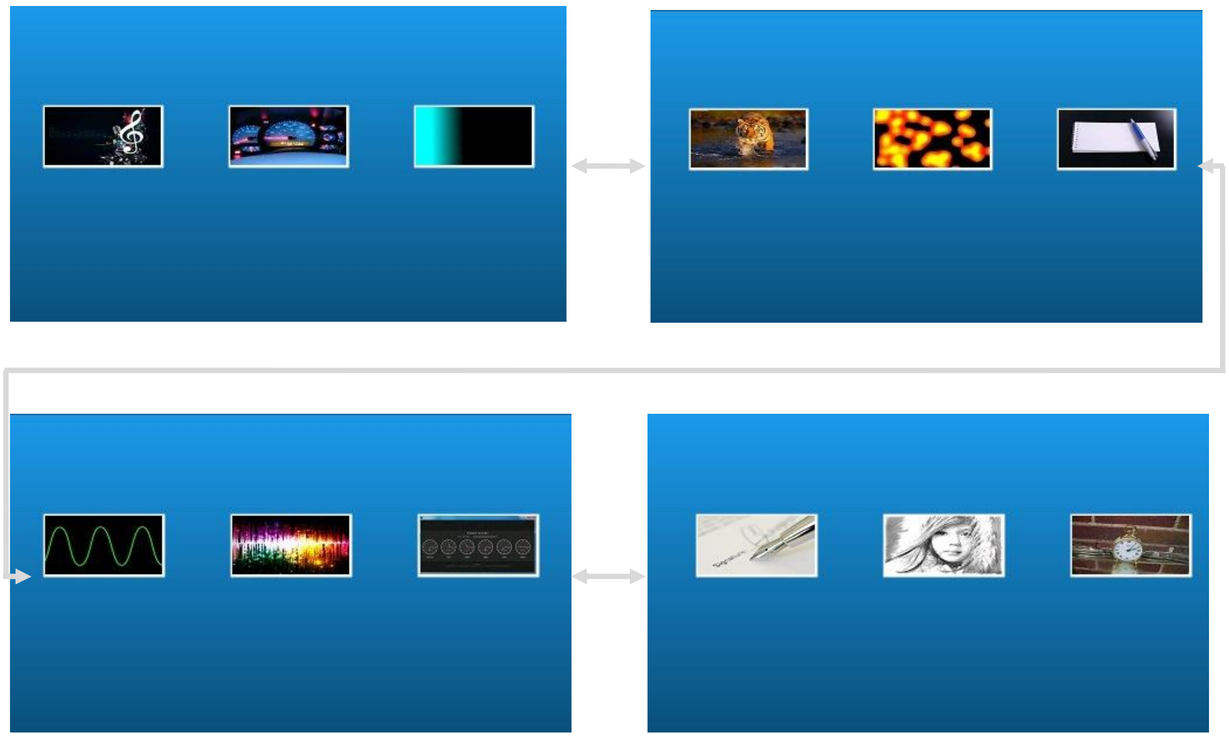


Figure 4.5 Loopback style screenshots

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 keyin_cts = 0,temp_keyin;
ft_uint8_t image_height = 50,image_width = 100;
ft_uint8_t dt = 30,dx,dy;
ft_uint8_t per_frame,no_frames,key_in,current_frame;
ft_int16_t sx,drag,Oy,Ox,dragth,i;
```

```
dx = (dt*2)+image_width;
dy = (10*2)+image_height;
per_frame = FT_DispWidth/dx;
no_frames = (MAX_MENUS-1)/per_frame;
```

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*no_frames)*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.

```
/*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)
{
       keyin_cts++;
       temp_keyin = key_in;
}
/*Move into the particular frame based on dragdt now 30pixels*/
if(sx==NOTOUCH)
{
       keyin_cts = 0;
       drag = scroller.base>>4;
}
/*if tag in but still pendown take a scroller basevalue*/
else if(keyin_cts>5)
{
       key_in = 0;
       temp_keyin = 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_Gpu_CoCmd_Dlstart(phost);
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
```

This section draw the background rectangles for the icons with 1 pixel higher than the bitmap images.

```
Oy = (FT_DispHeight-image_width)/2;
current_frame = drag/dx;
dragth = drag%dx;
Ft_App_WrCoCmd_Buffer(phost,BEGIN(RECTS));
for(i=-1;i<(per_frame+1);i++)
{
  Ox = dt+dx*i;
  Ox-=dragth;
  if(Ox > (FT_DispWidth+dt) || Ox < -dx) 0;
  else
  {
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((Ox-1)*16,(Oy-1)*16));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((image_width+Ox+1)*16,(image_height+Oy+1)*
    16));
  }
}
```

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

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

Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(0));
for(i=-1;i<(per_frame+1);i++)
{
  Ox = dt+dx*i;
  Ox-=dragth;
  if(Ox > (FT_DispWidth+dt) || Ox < -dx) 0;
  else
  {
    Ft_App_WrCoCmd_Buffer(phost,CELL((MAX_MENUS+i+current_frame)%12));
    Ft_App_WrCoCmd_Buffer(phost,TAG((1+i+current_frame)%(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.

4.4.4 Windows 8 Menu Style

This style is provided by the menu_win8() function.

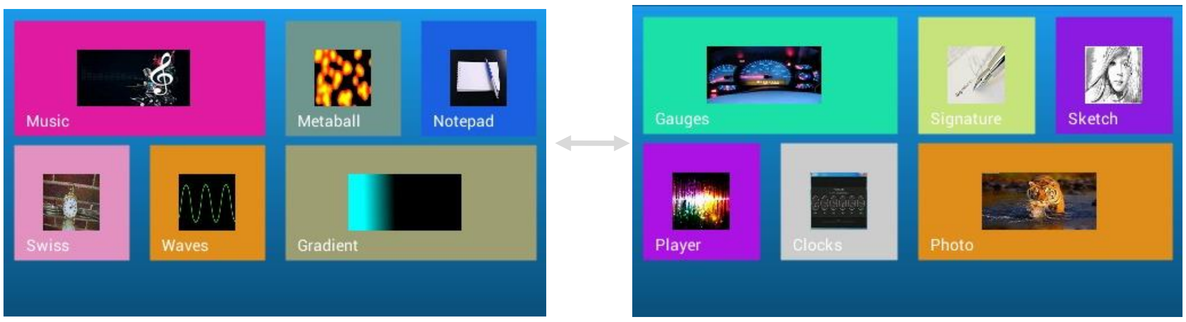


Figure 4.6 Windows 8 style screenshots

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"};

// 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.

```
for(option=0;option<3;option++)
{
 switch(option)
 {
   case 0:
       Ft_App_WrCoCmd_Buffer(phost,LINE_WIDTH(1*16));
       Ft_App_WrCoCmd_Buffer(phost,BEGIN(RECTS));
   break;
   case 1:
       Ft_App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
       Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
   break;
   case 2:
        Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
    break;
  }
  rectangle_width = 220;
  rectangle_height = 100;
  for(i=0;i<4;i+=1)
  {
    if(i<2)
    {
       Ox = 10+FT_DispWidth*i;
       Oy = 10;
    }else
    {
       Ox = 250+FT_DispWidth*(i%2);
       Oy = 120;
    }
    Ox -= frame_xoffset;
    if(Ox > (FT_DispWidth+frame_xoffset_dt) || Ox < -512) 0;
    else
    {
       Ft_App_WrCoCmd_Buffer(phost,TAG(i+1));
       switch(option)
       {
         case 0:
              Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(color[i][0],color[i][1],color[i]
          [2]));
             Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((Ox)*16,(Oy)*16));
              Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((rectangle_width+Ox)*16,(rectangl
              e_height+Oy)*16));
          break;
           case 1:

              Ft_App_WrCoCmd_Buffer(phost,CELL(i));
              Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((55+Ox)*16,(25+Oy)*16));
          break;
          case 2:
              Ft_Gpu_CoCmd_Text(phost,Ox+10,Oy+80,26,0,menudetails[i]);
          break;
         }
       }
     }
     rectangle_width = 100;
     rectangle_height = 100;
```

```
      if(option==1)  Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(512));
      for(i=0;i<8;i+=1)
      {
        if(i<4)
        {
          Ox = 250+FT_DispWidth*(i/2)+(image_width*(i%2))+(20*(i%2));
          Oy = 10;
        }
        else
        {
          Ox = 10+FT_DispWidth*(i/6)+ (((i-4)%2)*image_width)+(((i-4)%2)*20);
          Oy = 120;
        }
        Ox -= frame_xoffset;
        if(Ox > (FT_DispWidth+frame_xoffset_dt) || Ox < -512) 0;
        else
        {
          Ft_App_WrCoCmd_Buffer(phost,TAG(i+5));
          switch(option)
          {
                case 0:
                Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(color[i+5][0],color[i+5][1],colo
                r[i+5][2]));
                Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((Ox)*16,(Oy)*16));
                Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((rectangle_width+Ox)*16,(rectangl
                e_height+Oy)*16));
                break;
                case 1:
                       Ft_App_WrCoCmd_Buffer(phost,CELL(i+4));
                       Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((25+Ox)*16,(25+Oy)*16));
                break;
                case 2:
                Ft_Gpu_CoCmd_Text(phost,Ox+10,Oy+80,26,0,menudetails[i+4]);
                break;
          }
        }
      }
      if(option==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 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_win8() function and re-draw the main menu.

This api is used to show the ICON in larger size by using BITMAP transform properties.

```
void show_icon(ft_uint8_t iconno)
{
        Ft_Play_Sound(0x51,100,108);
        do
        {       Ft_Gpu_CoCmd_Dlstart(phost);
                Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
                Ft_Gpu_CoCmd_Gradient(phost,0,0,0x1A99E8,0,FT_DispHeight,0x0A4F7A);
        /* Save the graphics context before enter into background animatioon*/
                Ft_App_WrCoCmd_Buffer(phost,SAVE_CONTEXT());
                #ifdef BACKGROUND_ANIMATION_1
                                Backgroundanimation_1();
                #endif

                #ifdef BACKGROUND_ANIMATION_2
                        Backgroundanimation_2();
                #endif
                #ifdef BACKGROUND_ANIMATION_3
                                Backgroundanimation_3();
                #endif
                #ifdef BACKGROUND_ANIMATION_4
                                Backgroundanimation_4();
                #endif

                #ifdef BACKGROUND_ANIMATION_5
                                Backgroundanimation_5();
                #endif
                #ifdef BACKGROUND_ANIMATION_6
                                Backgroundanimation_6();
                #endif

                /* Restore th6 graphics context */
                Ft_App_WrCoCmd_Buffer(phost,RESTORE_CONTEXT());
                Ft_App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
                Ft_App_WrCoCmd_Buffer(phost,SAVE_CONTEXT());
                Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(0));

                Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST,BORDER,BORDER,200,100)
        );
                Ft_App_WrCoCmd_Buffer(phost,CELL(iconno-1));
                Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(128));
                Ft_App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_E(128));
                Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(((FT_DispWidth-
        200)/2)*16,((FT_DispHeight-100)/2)*16));
                Ft_App_WrCoCmd_Buffer(phost,RESTORE_CONTEXT());
                Ft_App_WrCoCmd_Buffer(phost,TAG('H'));
                #ifdef BACKGROUND_ANIMATION_4
                Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0,0,0));
                #endif
                Ft_App_WrCoCmd_Buffer(phost,VERTEX2II(5,5,13,0));
                Ft_App_WrCoCmd_Buffer(phost,DISPLAY());
                Ft_Gpu_CoCmd_Swap(phost);
                Ft_App_Flush_Co_Buffer(phost);
                Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
                if(snapflag)
                {
                   FILE *pfile = fopen("M2_6.brga","wb");
                   Screen_Snapshot(pfile,NULL,0);
                   fclose(pfile);
                }
                }while(Read_keys()!='H');
                Ft_Play_Sound(0x51,100,108);
                scroller.vel = 0;}
```

5 Background Animation

There are six sample background animations implemented in this application. The samples are written in a modular style and the user can add the background animation to their own application. All the background animations are done by using FT800 graphics primitives.

From the screen shots treat the icons as a foreground in the mainmenu application. Background animation may be designed separately and ported into the Mainmenu application.

The developer should take care about the GRAM, time comsuming to finish the background display list creation for the background animation.

All the background animations images are converted to 'L8'/ 'L4' format by using FTDI Image Converter utility

Interpolation concepts are used to move from source to destination based on iteration counts.

5.1 Interpolation

Interpolation concepts are implemented in this application for the object movement or image zoom in and zoom out from one point to another.

1. Linear Interpolation
2. Smooth step Interpolation
3. Acceleration and
4. Deceleration interpolations

An interpolation function defines how a variable changes between two values (e.g. starting and ending position of an object) with respect to time.

5.1.1 Linear interpolation

The simplest real interpolation function is Linear interpolation, also known as Lerp. It transits from one value to another at a constant rate, using a straightforward and intuitive formula.

Linear(p0,p1,t,r) = p0 + ((t/r)*(p1-p0));

```
float linear(float p1,float p2,ft_uint16_t t,ft_uint16_t rate)
{
	float st  = (float)t/rate;
	return p1+(st*(p2-p1));
}
```

5.1.2 Smoothstep Interpolation

Either Cosine or SmoothStep interpolation can replace almost all Linear interpolation for a smoother, more polished result.

SmoothStep(p0,p1,t,r) = Linear(p0,p1,((t/r)^2)*(3-2(t/r));

```
ft_int16_t smoothstep(ft_int16_t p1,ft_int16_t p2,ft_uint16_t t,ft_uint16_t rate)
{
	float dst  = (float)t/rate;
	float st = SQ(dst)*(3-2*dst);
	return p1+(st*(p2-p1));
}
```

5.1.3 Acceleration Interpolation

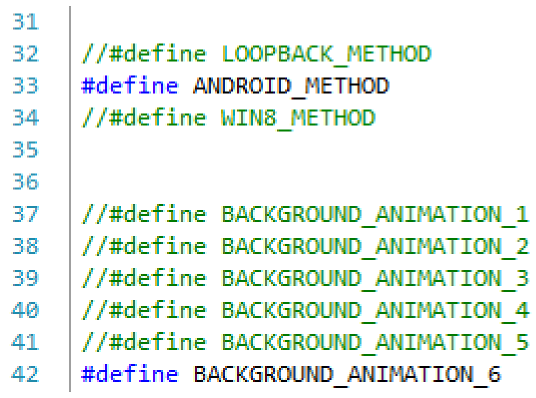
The object moving speed from one point to other is directly proportional to the third parameter time(t).

Acceleration(p0,p1,t,r) = Linear(p0,p1,((t/r)^2));

```
ft_int16_t acceleration(ft_int16_t p1,ft_int16_t p2,ft_uint16_t t,ft_uint16_t rate)
{
	float dst   = (float)t/rate;
	float st = SQ(dst);
	return p1+(st*(p2-p1));
}
```

5.2 Design Description

5.2.1 Background selection on MSVC and ARDUINO



```

//#define LOOPBACK_METHOD
#define ANDROID_METHOD
//#define WIN8_METHOD


//#define BACKGROUND_ANIMATION_1
//#define BACKGROUND_ANIMATION_2
//#define BACKGROUND_ANIMATION_3
//#define BACKGROUND_ANIMATION_4
//#define BACKGROUND_ANIMATION_5
#define BACKGROUND_ANIMATION_6
```

The user can select the Background animation and Application by uncommenting the predefinition and make sure the others are commented out.

5.2.2 Background animation 1

The animation are done by using musical notes bitmaps and gradient bitmap.

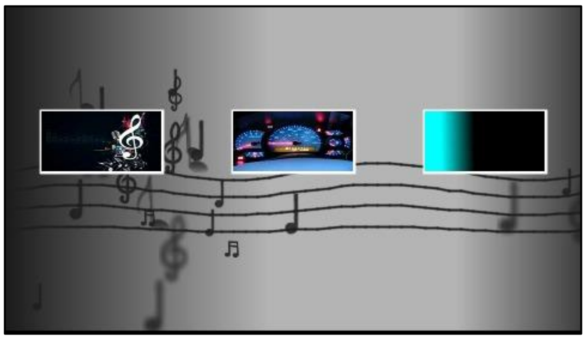


Figure 5.1 Background animation 1

Simple method are used to construct the background display list.

5.2.2.1 Bitmap Sources

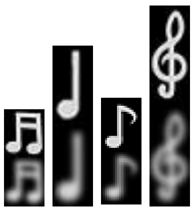


Figure 5.2 Background 1 Bitmap Sources

Figure 5.3 Background layer 1

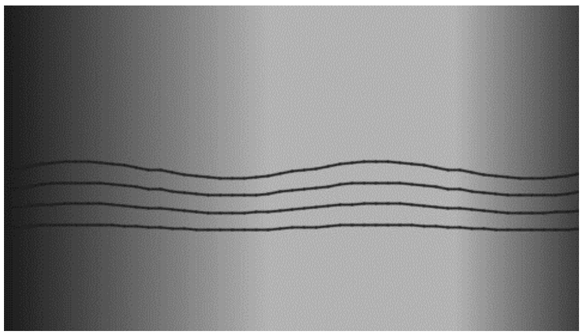


Figure 5.4 Background layer 2

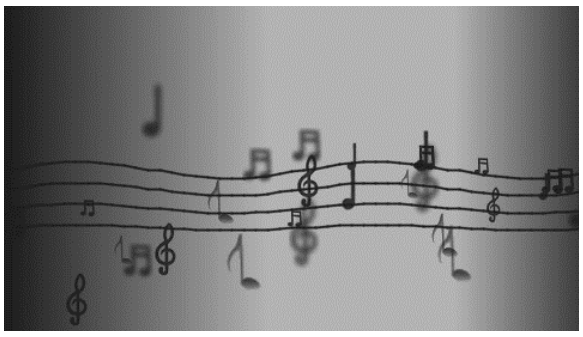


Figure 5.5 Background layer 3

The four lines are constructed once during first iteration and written into **GRAM.** The lines are constructed by using a sine function with different amplitude and vertical offset (yoffset).

```
void Sine_wave(ft_uint8_t amp,ft_uint16_t address,ft_uint16_t yoffset)
{
        ft_uint16_t x = 0,y=0;
        for(x=0;x<FT_DispWidth+100;x+=10)
        {
                y = (yoffset) + ((ft_int32_t)amp*qsin(-65536*x/(25*10))/65536);
                Ft_Gpu_Hal_Wr32(phost,address+(x/10)*4,VERTEX2F(x*16,y*16));
        }
}
```

5.2.2.2 Displaylist Construction:

In the first iteration before constructing the display list the application must intialze the source and destination offsets of the objects. Upload all the assets for the background animation into the GRAM and set the bitmap properties during first iteration.

```
if(!init)
{
        init = 1;// by set this flag in the second iteration its not execute again
        Sine_wave(15,RAM_G,FT_DispHeight/2);
        Sine_wave(12,RAM_G+232,16+(FT_DispHeight/2));
        Sine_wave(9,RAM_G+2*232,32+(FT_DispHeight/2));
        Sine_wave(6,RAM_G+3*232,48+(FT_DispHeight/2));
        for(i=0;i<30;i++)
        {
                yoffset_array[i] = ft_random(FT_DispHeight); // object destination
                bitmap_handle[i] = 4+ft_random(4);
                rate_cts[i] = 300+ft_random(200);
                iteration_cts[i] = ft_random(200);
        }
        Ft_App_LoadRawFromFile("nts1.raw",220*1024L);
        Ft_App_LoadRawFromFile("nts2.raw",222*1024L);
        Ft_App_LoadRawFromFile("nts3.raw",256506L);
        Ft_App_LoadRawFromFile("nts4.raw",257306L);
        Ft_App_LoadRawFromFile("hline.raw",255*1024L);
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(4));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(220*1024));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L4,10,50));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(BILINEAR, BORDER, BORDER, 40, 100));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(5));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(222*1024));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L4,25,60));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(BILINEAR, BORDER, BORDER,100, 120));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(6));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(256506L));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L4,10,40));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(BILINEAR, BORDER, BORDER, 40, 80));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(7));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(257306L));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L4,10,24));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(BILINEAR, BORDER, BORDER, 40, 48));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(8));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(255*1024L));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L8,512,1));
        Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST, REPEAT, REPEAT, 512, 272);
}
```

Note : All the bitmap sources are converted into RAW data by using FTDI image converter utility.

The static four lines on the design is appended data from GRAM by using the `LINE_STRIP`

```
wave_cts = FT_DispWidth+10;
Ft_App_WrCoCmd_Buffer(phost,LINE_WIDTH(1*16));   //Set the Line Width
Ft_App_WrCoCmd_Buffer(phost,COLOR_A(255));       //Set the COLOR ALPHA

/*First line*/
Ft_App_WrCoCmd_Buffer(phost,BEGIN(LINE_STRIP));
Ft_Gpu_CoCmd_Append(phost,RAM_G,(wave_cts/10)*4);
Ft_App_WrCoCmd_Buffer(phost,END());

/*Second line*/
Ft_App_WrCoCmd_Buffer(phost,BEGIN(LINE_STRIP));
Ft_Gpu_CoCmd_Append(phost,RAM_G+232,(wave_cts/10)*4);
Ft_App_WrCoCmd_Buffer(phost,END());

/*Third line*/
Ft_App_WrCoCmd_Buffer(phost,BEGIN(LINE_STRIP));
Ft_Gpu_CoCmd_Append(phost,RAM_G+2*232,(wave_cts/10)*4);
Ft_App_WrCoCmd_Buffer(phost,END());

/*Fourth line*/
Ft_App_WrCoCmd_Buffer(phost,BEGIN(LINE_STRIP));
Ft_Gpu_CoCmd_Append(phost,RAM_G+3*232,(wave_cts/10)*4);
Ft_App_WrCoCmd_Buffer(phost,END());

Ft_App_WrCoCmd_Buffer(phost,RESTORE_CONTEXT());
```

A total of 30 objects(musical notes) are used in this animation.

This animation is done by layers. Three layers are used to create it and help create a 3D effect.

So in the 3 layers, each layer has 10 objects which are randomly selected. (3x10).

The first layer displays objects with a blurred appearance giving the effect that objects are nearer to the sight. (Blurred objects are done by external tools, not done by FT800).

The second layer displays normal objects and gives the effect normal to the sight.

The third layer displays objects that appear zoomed out giving the effect that the objects are far to the sight.

The combination of the 3 layers provides the 3D effect.

Each of the objects are moving from one location (source) to another location (destination) by using linear interpolation based on the iteration counts. Once the object has reached the destination the iteration counts of the object is reset to ZERO and locations of source/destination are randomly picked. Each object moves at a different rate and the rate is calculated at the init time.

5.2.3 Background animation 2

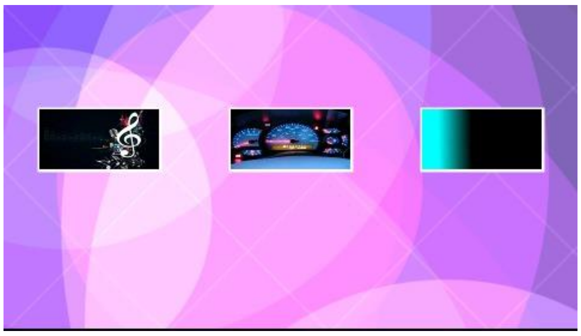


Figure 5.6 Background Animation 2

Figure 5.7 Background layer 1

Figure 5.8 Background layer 2

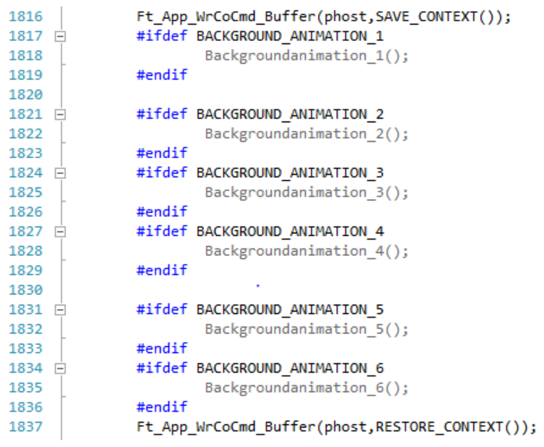
This type of background animation is created with a simple display list by using FT800 primitives. Only points and lines are used, no other resources are used but it gives a very elegant background. In the design only 20 points are used to create the effect with different size and color randomly picked. The point size and colors are picked randomly during the first iteration.

All the points are moved by linear interpolation with a constant rate. When a touch event is detected all the points move faster and when the touch is released the moving speed is gradually reduced.

In this background, an additive blending function used to get the effect.

From the code snippet below the user can find the code need to be added to the display list to create this background.

In the application, before entering into the background animation section, it is necessary to save the current graphics contents. On reentering the application from the animation section, the graphics content then needs to be restore.



```
        Ft_App_WrCoCmd_Buffer(phost,SAVE_CONTEXT());
        #ifdef BACKGROUND_ANIMATION_1
                Backgroundanimation_1();
        #endif

        #ifdef BACKGROUND_ANIMATION_2
                Backgroundanimation_2();
        #endif
        #ifdef BACKGROUND_ANIMATION_3
                Backgroundanimation_3();
        #endif
        #ifdef BACKGROUND_ANIMATION_4
                Backgroundanimation_4();
        #endif

        #ifdef BACKGROUND_ANIMATION_5
                Backgroundanimation_5();
        #endif
        #ifdef BACKGROUND_ANIMATION_6
                Backgroundanimation_6();
        #endif
        Ft_App_WrCoCmd_Buffer(phost,RESTORE_CONTEXT());
```

In the first iteration we need to initialize the source/destination offsets, point sizes.

```
if(!init)
{
        init = 1;
        for(pts=0;pts<20;pts++)
        {
                point_size[pts] = 136 *16 +ft_random(375*16);
                xoffset_array[pts] = ft_random(512)*16;
                yoffset_array[pts] = ft_random(512)*16;
                color[pts] = ft_random(5);
                dx_pts[pts] = 240*16+ft_random(240*16);
                dy_pts[pts] = 130*16+ft_random(142*16);
        }
}
```

This portion in the source code is used to draw the points,

```
for(i=0;i<5;i++)
{
        ptradius = point_size[5+i];
        colorindex = color[i+5];
        Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(ptclrarray[colorindex][0],ptclrarray[c
olorindex][1],ptclrarray[colorindex][2]));
        yoffset = linear(dy_pts[i+5],yoffset_array[i+5],t,1000);
        yoffset = yoffset_array[i+5];
        Ft_App_WrCoCmd_Buffer(phost,POINT_SIZE(ptradius));
        Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(xoffset,yoffset));
}
```

5.2.4 Background animation 3

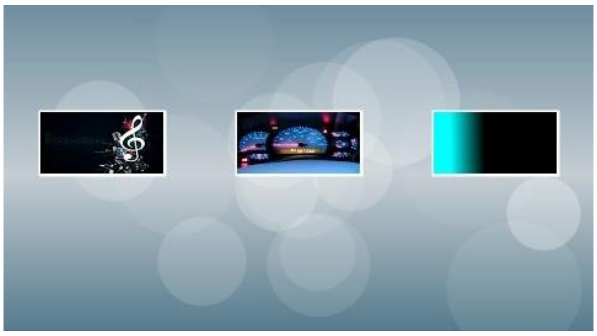


Figure 5.9 Background animation 3

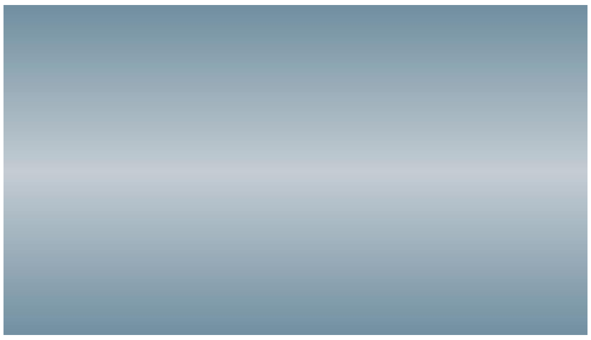


Figure 5.10 Background Layer 1

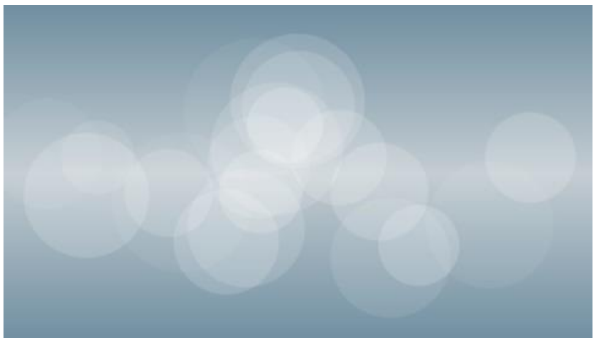


Figure 5.11 Background Layer 2

This background is very easy to do in the FT800. We are using two layers for this design, one is a gradient while the second one uses points with additive blending.

Gradient:

```
Ft_App_WrCoCmd_Buffer(phost,SCISSOR_SIZE(480,136));
Ft_App_WrCoCmd_Buffer(phost,SCISSOR_XY(0,0));
Ft_Gpu_CoCmd_Gradient(phost,0,0,0x4f7588,0,136,0xc4cdd2);
Ft_App_WrCoCmd_Buffer(phost,SCISSOR_XY(0,136));
Ft_Gpu_CoCmd_Gradient(phost,0,136,0xc4cdd2,0,272,0x4f7588);
//reprogram with  default values
Ft_App_WrCoCmd_Buffer(phost,SCISSOR_XY(0,0));
```

```
        Ft_App_WrCoCmd_Buffer(phost,SCISSOR_SIZE(512,512));
```

In the above code snippet the display is essentially split into a top and bottom half and the gradient call is applied in each half. The scissior commands are used to update the display for the particular portion. After the gradient creation we need to reset the default values of scissor commands, in this case to the full screen.

The points sizes, offsets are randomly picked,

```
for(i = 0; i<20 ; i++)
{
        if(iteration_cts[i]==0)
        {
                xoffset_array[i] = ft_random(FT_DispWidth);
                yoffset_array[i] = 100+ft_random(FT_DispHeight/4);
                dx_pts[i] = ft_random(FT_DispWidth);
                dy_pts[i] = ft_random(FT_DispHeight);
                rate_cts[i] = 500+ft_random(500);
        }
        if(iteration_cts[i]<rate_cts[i])iteration_cts[i]+=inc;
        else{ iteration_cts[i] = 0;}
}
```

All the points gradually disappear by using the color alpha value linearly.

```
alpha  = linear(80,0,iteration_cts[i],rate_cts[i]);
Ft_App_WrCoCmd_Buffer(phost,COLOR_A(alpha));
```

5.2.5 Background animation 4

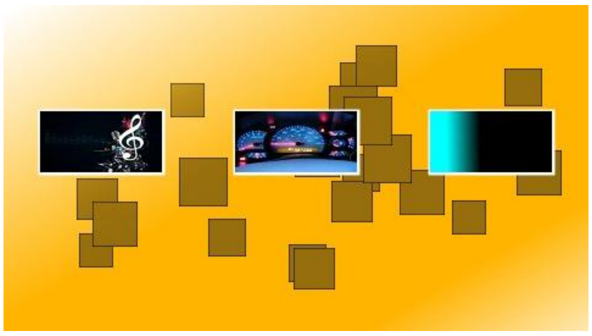


Figure 5.12 Background animation 4

Figure 5.13 Background Layer 1

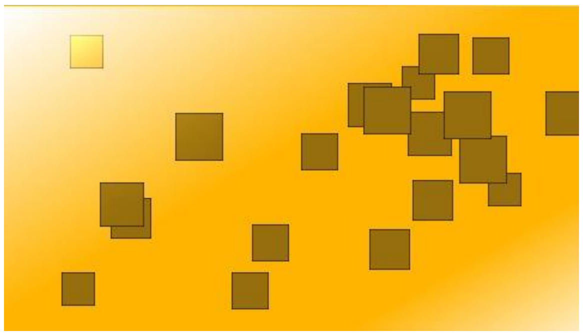


Figure 5.14 Background Layer 2

This example and the previous background are almost similar. In the first background animation the inbuilt FT800 gradient function is used, here it is a bitmap 512x1 pixels. Instead of Circles (POINTS), here rectangle used. Appearance and disappearance is different in this example with objects suddenly appearing and disappearing as opposed to fading in and out.

The gradient is drawn diagonally by using the coprocessor rotation command to rotate the bitmap by 60 degree in the code.

```
Ft_Gpu_CoCmd_LoadIdentity(phost);
Ft_Gpu_CoCmd_Rotate(phost, 60*65536/360);//rotate by 60 degrees clock wise
Ft_Gpu_CoCmd_SetMatrix(phost);

Ft_App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0xff,0xb4,0x00));
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
Ft_App_WrCoCmd_Buffer(phost,VERTEX2II(0,0,1,0));
Ft_App_WrCoCmd_Buffer(phost,COLOR_MASK(1,1,1,1));
Ft_App_WrCoCmd_Buffer(phost,BLEND_FUNC(DST_ALPHA,ONE_MINUS_DST_ALPHA));
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0xff,0xb4,0x00));
Ft_App_WrCoCmd_Buffer(phost,BEGIN(RECTS));
Ft_App_WrCoCmd_Buffer(phost,VERTEX2II(0,0,0,0));
Ft_App_WrCoCmd_Buffer(phost,VERTEX2II(FT_DispWidth,FT_DispHeight,0,0));

Ft_App_WrCoCmd_Buffer(phost,BEGIN(RECTS));
Ft_App_WrCoCmd_Buffer(phost,BLEND_FUNC(SRC_ALPHA,ONE_MINUS_DST_ALPHA));
Ft_App_WrCoCmd_Buffer(phost,COLOR_MASK(1,1,1,0));
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0x96,0x6e,0x0d));
Ft_App_WrCoCmd_Buffer(phost,LINE_WIDTH(16*1));
for(i=0;i<numBlobs;i++)
{
        ft_int32_t xoffset,yoffset;
        if(0 == i%4)
        {
                linesize = 16*(25 + (3*i/4));
        }
        alpha  = linear(80,0,iteration_cts[i],rate_cts[i]);
        if(alpha<75)
        {
        xoff = linear(xoffset_array[i],dx_pts[i],iteration_cts[i],rate_cts[i]);

        yoff = linear(yoffset_array[i],dy_pts[i],iteration_cts[i],rate_cts[i]);
        Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(xoff*16,yoff*16));

        Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(xoff*16+linesize,yoff*16+linesize));
        }
}
Ft_App_WrCoCmd_Buffer(phost,RESTORE_CONTEXT());
```

5.2.6 Background animation 5

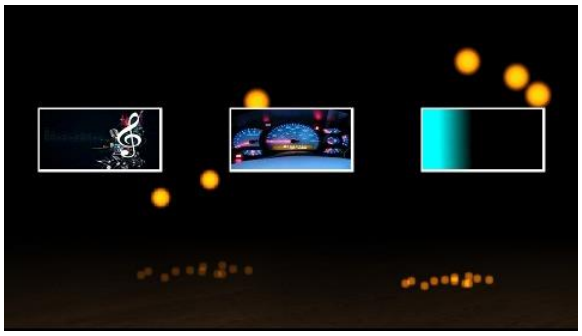


Figure 5.15 Background animation 5

This background animation effect demonstrates a concept of fire balls falling from sky on ground floor and when these fire balls drop/collide onto the floor they break into multiple mini fireballs and disappear.

Figure 5.16 Background Layer 1

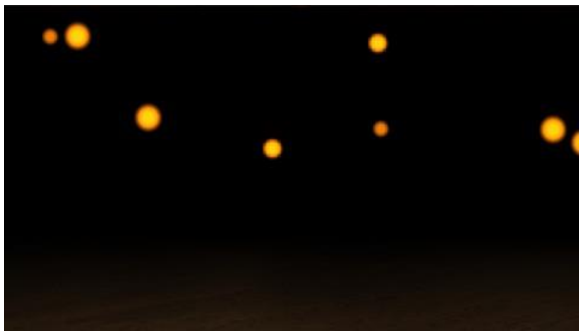


Figure 5.17 Background Layer 2

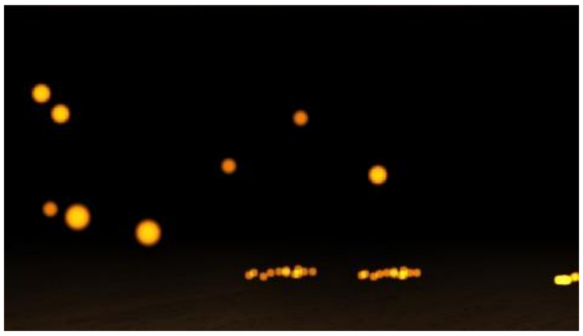


Figure 5.18 Background Layer 3

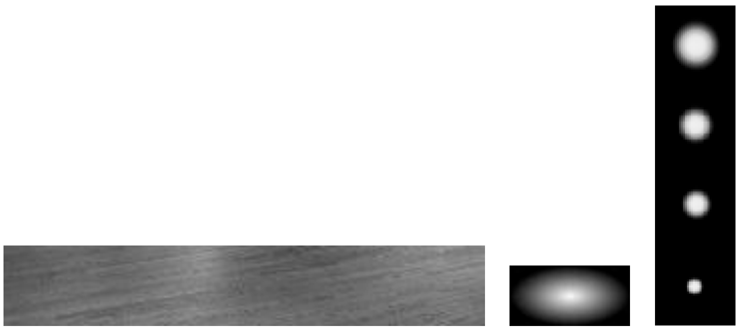


Figure 5.19 Bitmap Sources

The background animation effect can be split into three sub concepts:

(a) Ground floor - constructed by displaying the 1st bitmap of figure 5.19

(b) fireballs falling from sky (before colliding onto ground) - constructed by using the 3rd bitmap of figure 5.19

(c) fireballs when dropping/colliding onto ground floor - constructed by using the 2nd bitmap of figure 5.19 and the 4th cell of the 3rd bitmap of figure 5.19.

The ground floor is constructed by displaying a bitmap (1st bitmap in 5.19) and overlaying with a linear gradient bitmap. Fireballs in the sky are constructed by displaying various bitmaps (3rd bitmap in 5.19) at random locations and moving them from the top of the screen to the bottom of the screen with constant rates. Note that the top 3 cells of the 3rd bitmap in 5.19 are utilized for displaying fireballs in sky.

Fireball collisions onto the ground floor are constructed by displaying the 2nd bitmap of 5.19 and overlaying with many small fireball bitmaps (4th cell of 3rd bitmap in 5.19). This gives an effect of the fireball breaking into multiple mini fireballs. The movement of these mini fireballs are based on an eclipse equation where x axis movement is greater than y axis movement. To give a disappearing effect of the mini fireballs, alpha values of these fireballs are reduced with respect to distance from the collision point till the end of the movement.

Below api is used to create the collition effect.

```
ft_void_t collaid_bubbles(ft_uint8_t inc)
{
 ft_int16_t i,j,k,yoff,xoff,temp;
 static ft_uint8_t rate = 50;
 Ft_App_WrCoCmd_Buffer(phost,CELL(3));
 for(j=0;j<3;j++)
 {
  for(i=0;i<firebubbles.number_of_firebubbles;i++)
  {
   if(firebubbles.iteration_cts[j*5+i]>=firebubbles.yoffset_array[j*5+i])
   {
   for(k=0;k<12;k++)
   {
    Ft_App_WrCoCmd_Buffer(phost,COLOR_A(200-firebubbles.disable_cts[j*5+i][k]*10));

    temp =
(ft_uint8_t)deceleration(0,firebubbles.radius_a[k],firebubbles.disable_cts[j*5+i][k
],20);
    xoff = firebubbles.xoffset_array[j*5+i]+10 +
(temp)*cos(firebubbles.angle[k]*0.01744);
    temp =
(ft_uint8_t)deceleration(0,firebubbles.radius_b[k],firebubbles.disable_cts[j*5+i][k
],20);
    yoff = firebubbles.yoffset_array[j*5+i]+10 +
(temp)*sin(firebubbles.angle[k]*0.01744);
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(xoff*16,yoff*16));
    if(inc)
    {
     temp =  j*5+i;
     if(firebubbles.disable_cts[temp][k]<20)
     firebubbles.disable_cts[temp][k]++;
     else
     {
      firebubbles.disable_cts[temp][k] = 0;
      firebubbles.iteration_cts[temp] = 0;

      firebubbles.xoffset_array[temp] = ft_random(FT_DispWidth);
      firebubbles.yoffset_array_source[temp] = -50-ft_random(100);
      if(j==0)
      firebubbles.yoffset_array[temp] = ft_random(20)+(FT_DispHeight-50);

      else if(j==1)
      firebubbles.yoffset_array[temp] = ft_random(20)+(FT_DispHeight-75);

      else if(j==2)
      firebubbles.yoffset_array[temp] = ft_random(20)+(FT_DispHeight-95);
     }
    }
   }
  }
 }
}
```

5.2.7 Background animation 6

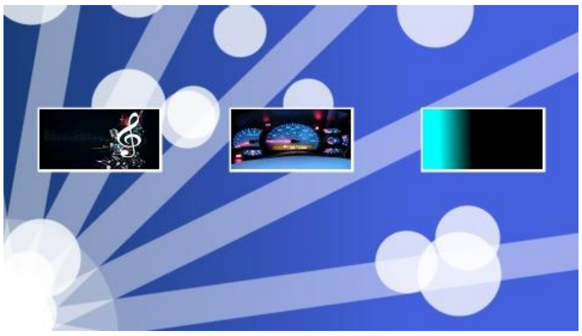


Figure 5.20 Background animation 6

Figure 5.21 Background Layer 1:

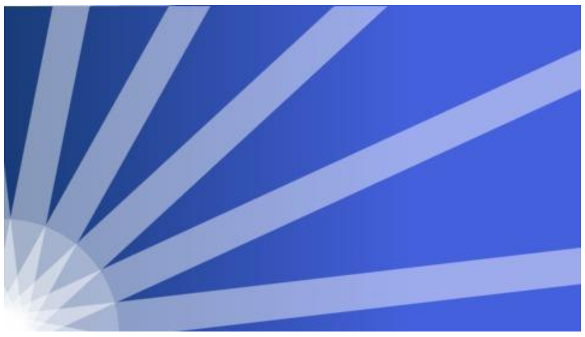


Figure 5.22 Background Layer 2:

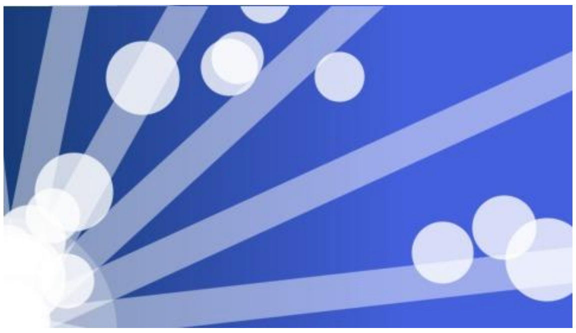


Figure 5.23 Background Layer 3:

In this background there are three layers are used, the first layer uses the inbuilt gradient, the second layer uses lines with the orgin point and the third layer is displaying bubbles drawn with points all based on FT800 primitives. All the point sizes and destination are picked randomly during the first iteration. 20 lines are used to create this backround and the angle between each line is 18 degree so 20x18 = 360.

All the lines are rotating by using a polar function,

```
static void polar(ft_int32_t r, float th,ft_uint16_t ox,ft_uint16_t oy)
{
  ft_int32_t x, y;
  th = (th * 32768L / 180);
  x = (16 * ox + (((long)r * qsin(th)) >> 11) + 16);
  y = (16 * oy - (((long)r * qcos(th)) >> 11));
  Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(x,y));
}
```

The lines are drawn by the below code

```
Ft_App_WrCoCmd_Buffer(phost,LINE_WIDTH(15*16));
Ft_App_WrCoCmd_Buffer(phost,BEGIN(LINES));
for(z=0;z<20;z++)
{
       polar((ft_int32_t)(0),move+z*ANGLE,0,272);
       polar((ft_int32_t)(600),move+z*ANGLE,0,272);
}
```

The inbulit coprocessor command is used to create the gradient effect in the code

```
Ft_Gpu_CoCmd_Gradient(phost,0, 0, 0x183c78, 340, 0,0x4560dd);
```

For more information about the coprocessor commands refer to the FT800 series programmers guide.

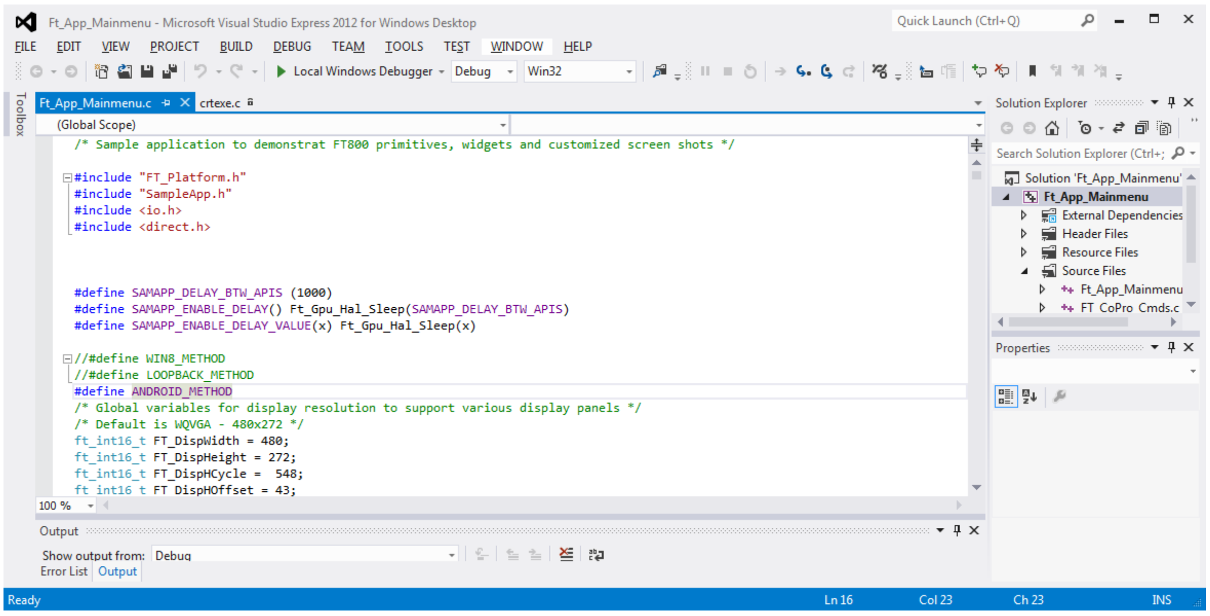
6 Running the demonstration code

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 2 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.



```
/* Sample application to demonstrat FT800 primitives, widgets and customized screen shots */

#include "FT_Platform.h"
#include "SampleApp.h"
#include <io.h>
#include <direct.h>



#define SAMAPP_DELAY_BTW_APIS (1000)
#define SAMAPP_ENABLE_DELAY() Ft_Gpu_Hal_Sleep(SAMAPP_DELAY_BTW_APIS)
#define SAMAPP_ENABLE_DELAY_VALUE(x) Ft_Gpu_Hal_Sleep(x)

//#define WIN8_METHOD
//#define LOOPBACK_METHOD
#define ANDROID_METHOD
/* Global variables for display resolution to support various display panels */
/* Default is WQVGA - 480x272 */
ft_int16_t FT_DispWidth = 480;
ft_int16_t FT_DispHeight = 272;
ft_int16_t FT_DispHCycle =  548;
ft_int16_t FT_DispHOffset = 43;
```

Figure 6.1 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).

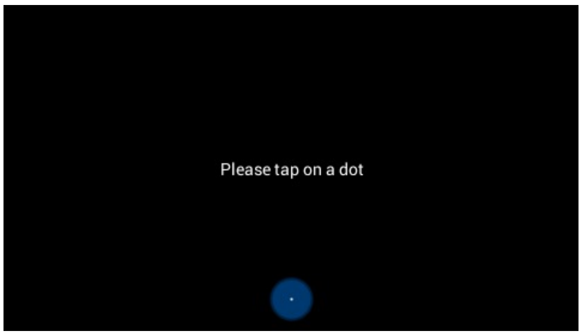


Figure 6.2 Calibration screen

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.

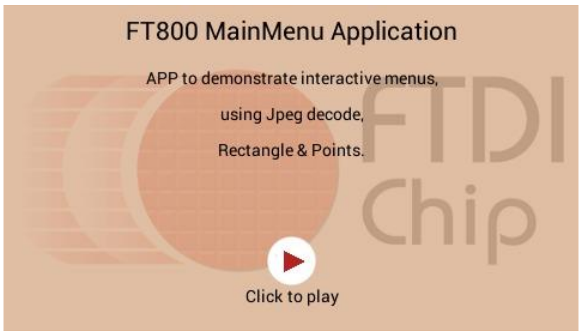

FT800 MainMenu Application
APP to demonstrate interactive menus,
using Jpeg decode,
Rectangle & Points.
FTDI
Chip
Click to play

Figure 6.3 Introduction screen

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

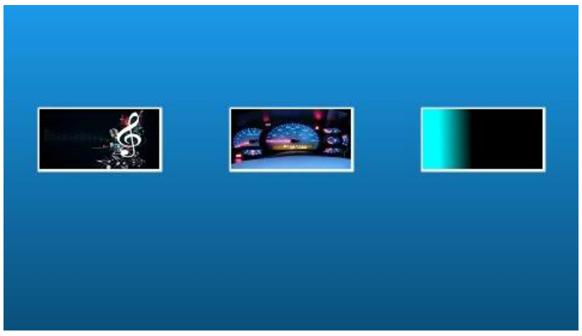


Figure 6.4 Main menu (loopback style)

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.

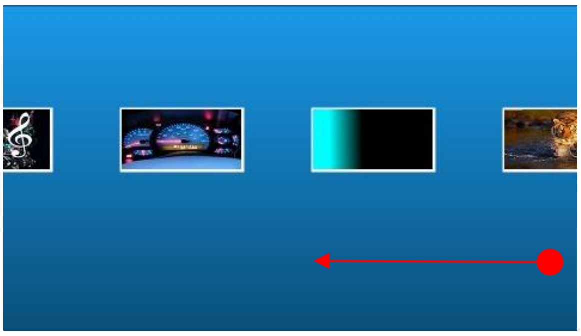


Figure 6.5 Scrolling along the menu

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.

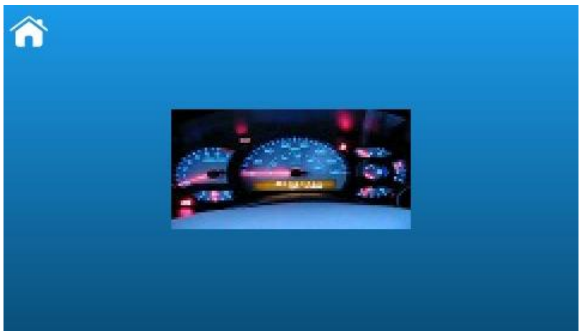


Figure 6.6 After selecting Gauges from the menu

7 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

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

Branch Office – Tigard, Oregon, USA

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

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

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited (Taiwan)
2F, No. 516, Sec. 1, NeiHu Road
Taipei 114
Taiwan , R.O.C.
Tel: +886 (0) 2 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
4. FT800 Embedded Video Engine Datasheet
5. FTDI image conversion utilities

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
1.2	Updated with a new section on background animation effects.	2014-04-10