Application Note

AN_265

FT_App_MainMenu

Version 1.3

Issue Date: 2018-01-05

This application note describes the operation of the Main Menu Demo Application running on Visual Studio. The Main Menu example demonstrates the way in which the FT8XX JPEG Decoding feature and Tracker can be used to create user-friendly menus with icons and scrolling capabilities.

Bridgetek Pte Ltd (BRTChip)
178 Paya Lebar Road, #07-03, Singapore 409030
Tel: +65 6547 4827 Fax: +65 6841 6071
Web Site: http://www.brtchip.com

Table of Contents

1 Introduction

This example demonstrates the way in which the FT8XX 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://brtchip.com/SoftwareExamples-eve/

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), users 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 using the FT8XX 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 FT8XX initialisation and main menu code
- Description of the menu application code
- Running the demonstration code

Additional documentation can be found at http://brtchip.com/SoftwareExamples-eve/.

Note: This document is intended to be used along with the source code project provided. This can be found at the above link.

2 MainMenu 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

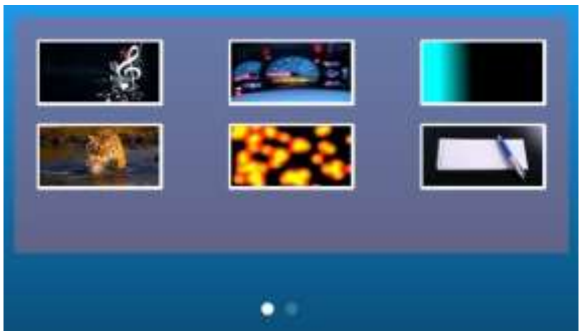


Figure 2.1 Android style

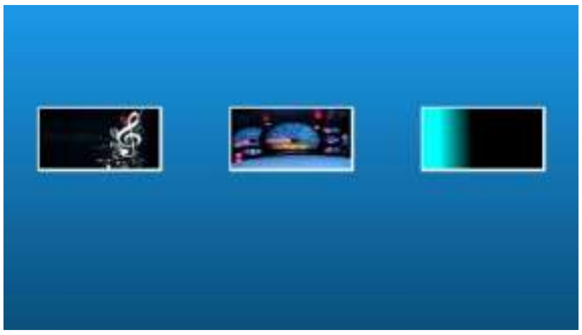


Figure 2.2 Loopback style

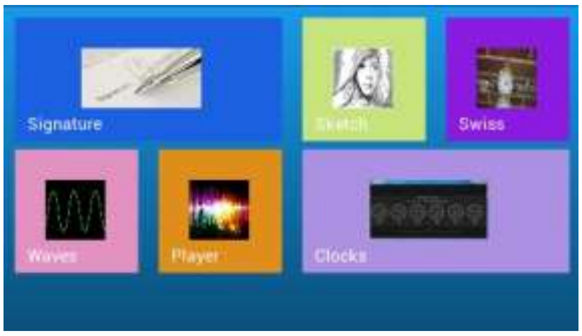


Figure 2.3 Windows 8 style

3 Design Flow

3.1 Initialisation

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

The supplied sample code includes the full flow but this document focuses on the main application in the final step. The earlier steps are generic to the EVE examples and are covered in the application note AN_391 EVE Platform Guide

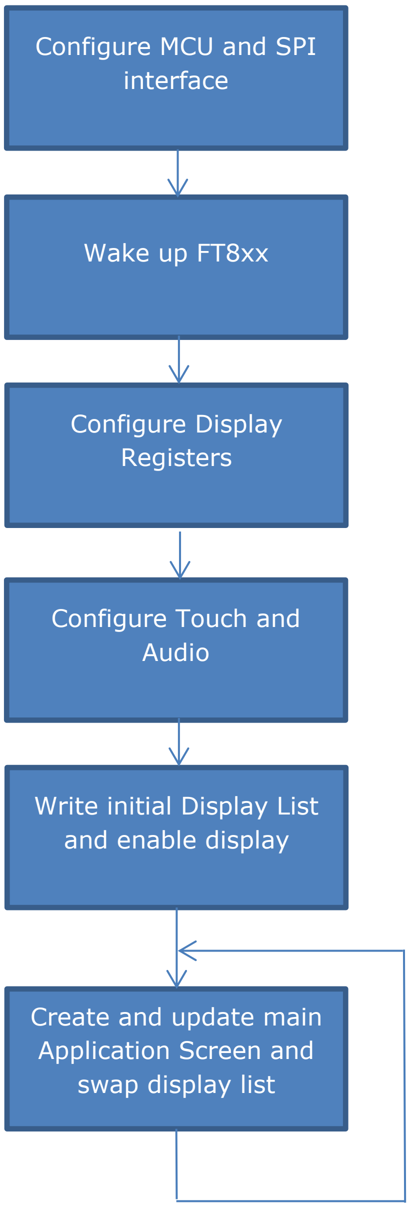

Configure MCU and SPI interface
Wake up FT8xx
Configure Display Registers
Configure Touch and Audio
Write initial Display List and enable display
Create and update main Application Screen and swap display list

Figure 3.1 Generic EVE Design Flow

3.2 Application Flow

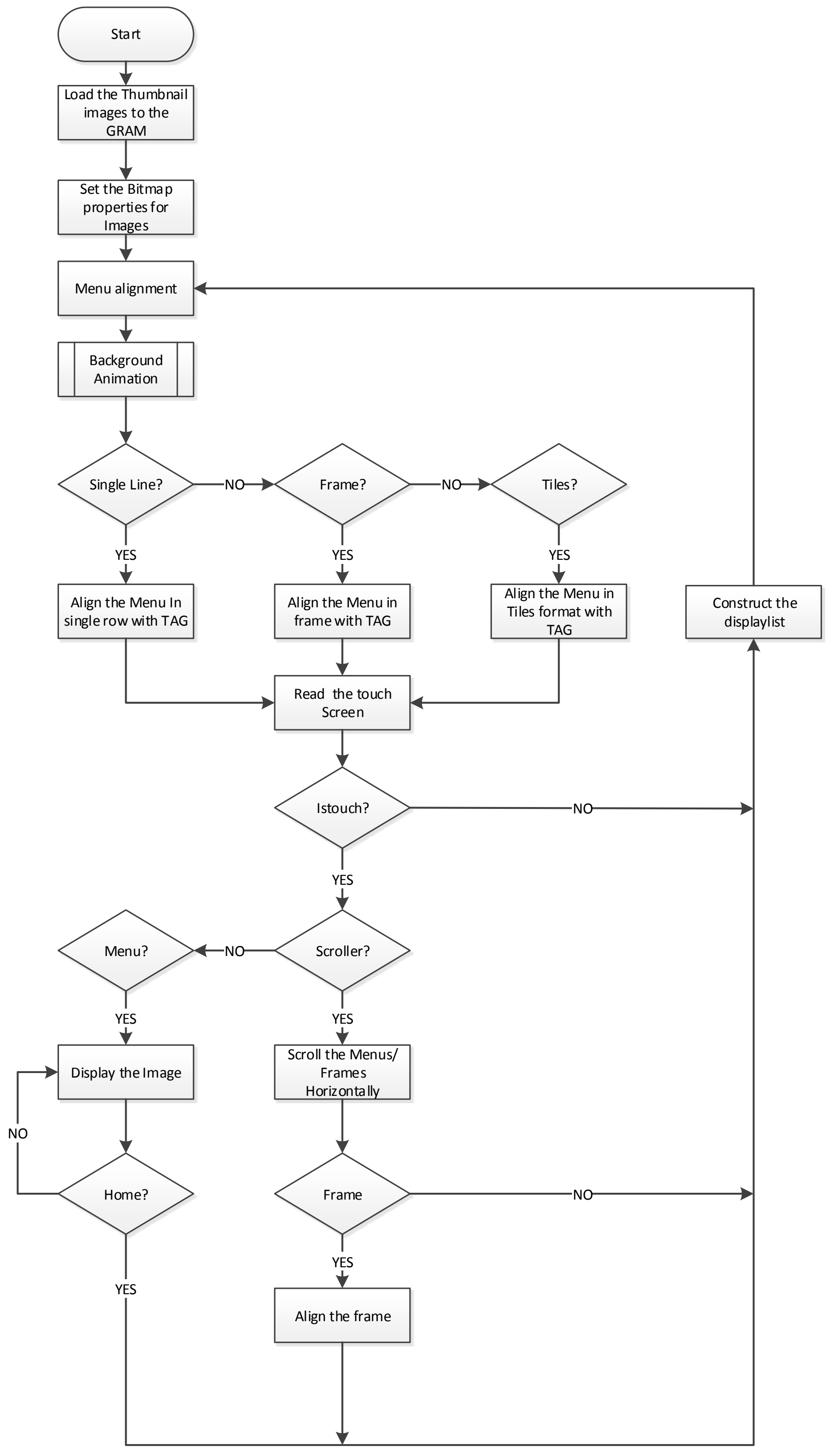

Start
Load the Thumbnail images to the GRAM
Set the Bitmap properties for Images
Menu alignment
Background Animation
Single Line?
NO
Frame?
NO
Tiles?
YES
YES
YES
Align the Menu In single row with TAG
Align the Menu in frame with TAG
Align the Menu in Tiles format with TAG
Construct the displaylist
Read 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

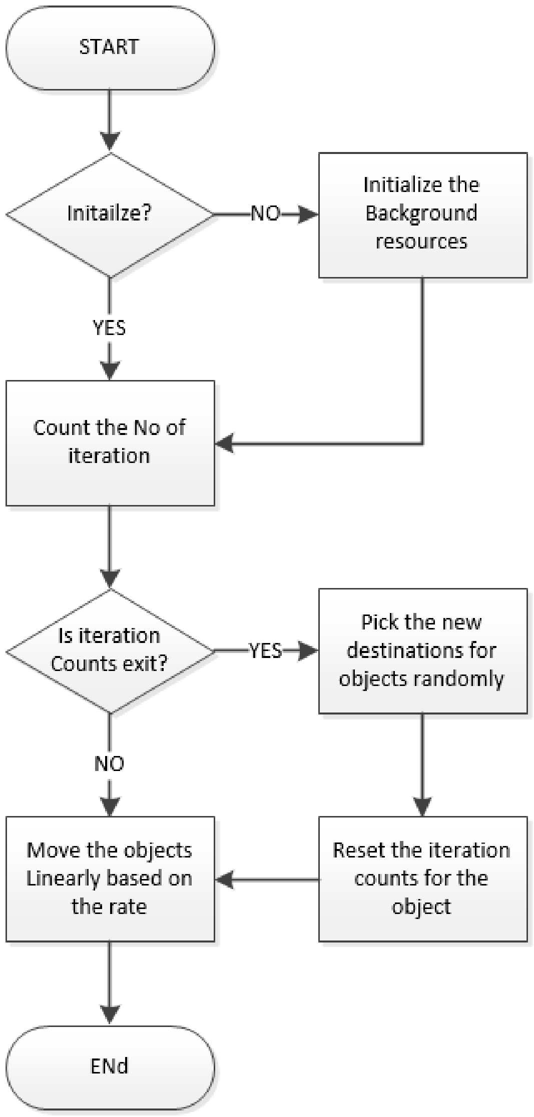

START
Initailze?
NO
Initialize the Background resources
YES
Count the No of iteration
Is iteration Counts exit?
YES
Pick the new destinations for objects randomly
NO
Move the objects Linearly based on the rate
Reset the iteration counts for the object
ENd

Figure 3.3 Background Flowchart

4 Description

Refer to AN 391 EVE Platform Guide for information pertaining to platform setup and the necessary development environment.

4.1 Application Start Screen

Upon completing the setup, the application start screen is displayed.

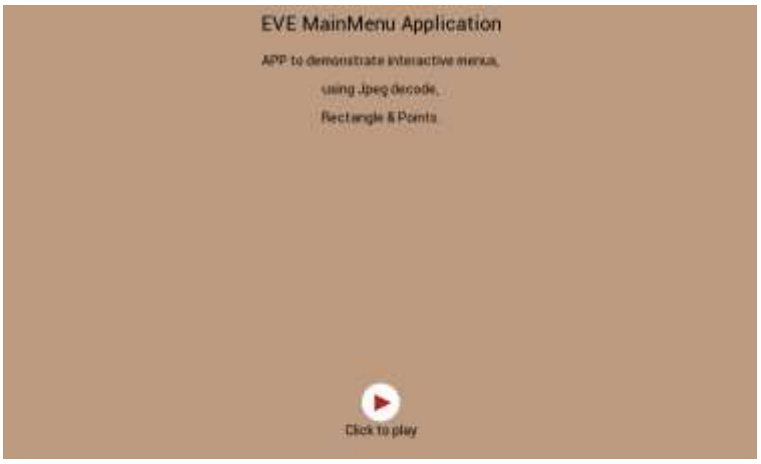


Figure 4.1 Start screen

4.2 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.

4.2.1 Images into GRAM

12 images are used for the main menu application. All the images are common for three types of mainmenu. Image size 100 x 50 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.

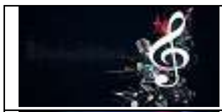
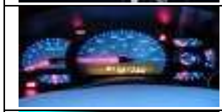
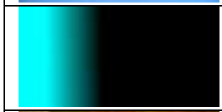
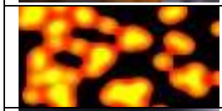
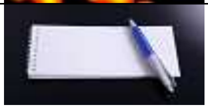
	Cell – 0		Cell - 6
	Cell - 1		Cell - 7
	Cell - 2		Cell - 8
	Cell – 3		Cell - 9
	Cell - 4		Cell – 10
	Cell - 5		Cell - 11

Table 1 Icon Cells

4.2.2 Android Menu Style

This style is provided by the menu() function.

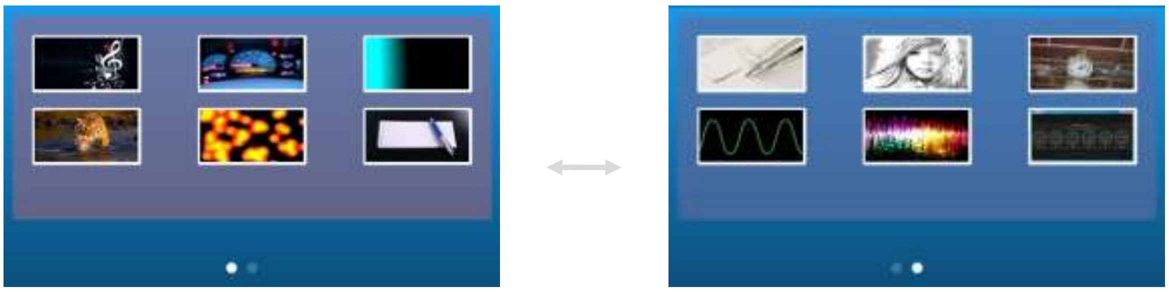


Figure 4.2 Android menu pages

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.

```
        uint8_t image_height = displayImageHeight/*50*/,image_width =
displayImageWidth/*100*/;
        uint16_t dt = 30,dx,dy;
        uint8_t col,row,per_frame,noof_frame,current_frame=0;
        uint8_t i,key_in=0,key_in_counts=0,temp=0;

        int16_t Ox,Oy,sx,drag=0,prev=0,drag_dt=30,dragth=0;
        uint16_t  point_offset,point_dt =15;  // for points
```

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 menu Thumbnails*/
        Load_Thumbnails();
        /*Initialize the scroller*/
        scroller_init((DispWidth*total_frames)*16);
```

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

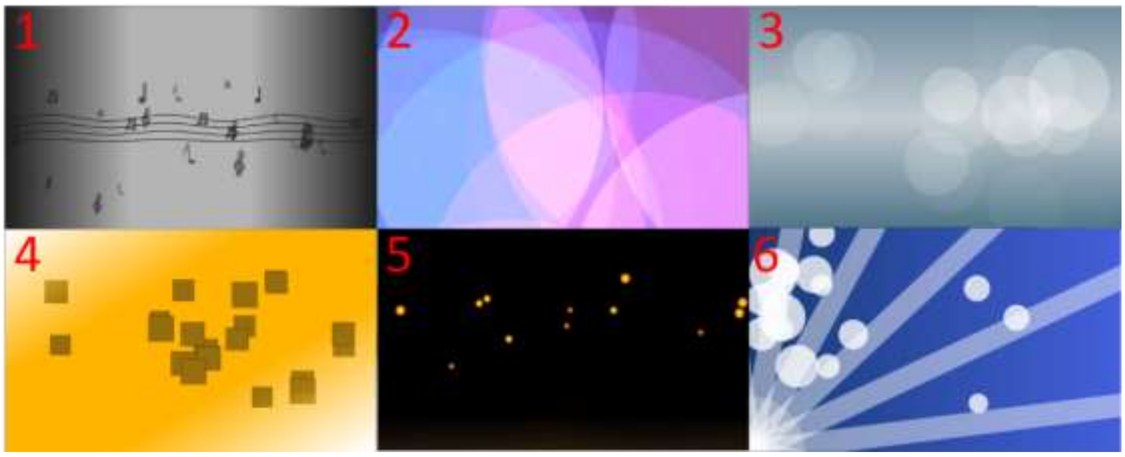
The first section reads the X value of the REG_TOUCH_SCREEN_XY register. The value of this register indicates the coordinates of the point on the screen which is currently being touched (or returns -32768 for the X and Y coordinates if the screen is not being touched). It also checks the REG_TOUCH_TAG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

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 variation and tag in*/
        sx =  Gpu_Hal_Rd16(phost,REG_TOUCH_SCREEN_XY + 2);
        key_in = App_Read_Tag(phost);

        /*Check if any tag in*/
        if(sx!=NOTOUCH)
        keyin_cts++;

        /*Move into the particular frame based on dragdt now 30pixels*/
        if(sx==NOTOUCH)
        {
                keyin_cts = 0;
                if(drag>((current_frame*DispWidth)+drag_dt))
                        drag = MIN((current_frame+1)*DispWidth,drag+15);
                if(drag<((current_frame*DispWidth)-drag_dt))
                        drag = MAX((current_frame-1)*DispWidth,drag-15);
                if(dragth==drag)
                        current_frame = drag/DispWidth;
                dragth = drag;
                scroller.vel = 0;
                scroller.base = dragth*16;                      // 16bit pre
        }
        /*if tag in but still pendown take a scroller basevalue*/
        else if(keyin_cts>5)
        {
                key_in = 0;
                drag = scroller.base>>4;
        }
        if(key_in==0)  scroller_run();
```

The code then begins to draw background animation. There are 6 available backgrounds:



```
        /*Display list start*/
        Gpu_CoCmd_Dlstart(phost);
        App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
        #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
```

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 sets the background for each page with a slightly different colour. The current sample has only two pages.

```
        App_WrCoCmd_Buffer(phost,BEGIN(RECTS));
        App_WrCoCmd_Buffer(phost,COLOR_A(150));
        App_WrCoCmd_Buffer(phost,COLOR_RGB(100,106,156));
        Oy = 10;
        for(i=0;i<=noof_frame;i++)
        {
               Ox = 10;
               Ox+=(i*DispWidth);
               Ox-=drag;
               if(Ox > 1023)
                      Ox = 1023;
               if(i==0) App_WrCoCmd_Buffer(phost,COLOR_RGB(156,100,128));
               if(i==1) App_WrCoCmd_Buffer(phost,COLOR_RGB(100,106,156));
               if(i==2) App_WrCoCmd_Buffer(phost,COLOR_RGB(156,152,100));
```

```
              App_WrCoCmd_Buffer(phost,VERTEX2F((Ox)*16,(Oy)*16));
#if defined(DISPLAY_RESOLUTION_WVGA)
              if((Ox+DispWidth-20) > 1023)

       App_WrCoCmd_Buffer(phost,VERTEX2F((1023)*16,(int16_t)(DispHeight*0.75)*16));
              else
                     App_WrCoCmd_Buffer(phost,VERTEX2F((Ox+DispWidth-
20)*16,(int16_t)(DispHeight*0.75)*16));
#else
              App_WrCoCmd_Buffer(phost,VERTEX2F((Ox+DispWidth-
20)*16,(int16_t)(DispHeight*0.75)*16));
#endif// i pixels wide than image width +1
       }
```

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

```
App_WrCoCmd_Buffer(phost,COLOR_A(255));
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_frame)*DispWidth);
       Ox -= drag;
       Oy = dt+(dy*((i/col)%row));
       if(Ox > (DispWidth+dt)) 0;
       else
       {
          App_WrCoCmd_Buffer(phost,VERTEX2F((Ox-1)*16,(Oy-1)*16));
          App_WrCoCmd_Buffer(phost,
       VERTEX2F((image_width+Ox+1)*16,(image_height+Oy+1)*16));
       }                    // i pixels wide than image width +1
}
```

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

```
for(i=0;i<MAX_MENUS;i++)
{
       Ox = dt+dx*(i%col);                                   // Calculate the
xoffsets
       Ox +=((i/per_frame)*DispWidth);
       Ox -= drag;
       Oy = dt+(dy*((i/col)%row));
       if(Ox > (DispWidth+dt) || Ox < -dx) 0;
       else
       {
              App_WrCoCmd_Buffer(phost,CELL(i));
              App_WrCoCmd_Buffer(phost,TAG(i+1));
              App_WrCoCmd_Buffer(phost,VERTEX2F(Ox*16,Oy*16));
       }
}
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
App_WrCoCmd_Buffer(phost,POINT_SIZE(MENU_POINTSIZE*16));
App_WrCoCmd_Buffer(phost,BEGIN(FTPOINTS));
App_WrCoCmd_Buffer(phost,COLOR_A(50));
Oy = DispHeight - 20;
for(i=0;i<=noof_frame;i++)
{
        Ox = point_offset+(i*(MENU_POINTSIZE+point_dt));
        App_WrCoCmd_Buffer(phost,VERTEX2F(Ox*16,Oy*16));
}

Ox = point_offset+(current_frame*(MENU_POINTSIZE+point_dt));
App_WrCoCmd_Buffer(phost,COLOR_A(255));
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 FT8XX 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 FT8XX. 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.

```
App_WrCoCmd_Buffer(phost,DISPLAY());
Gpu_CoCmd_Swap(phost);
App_Flush_Co_Buffer(phost);
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.

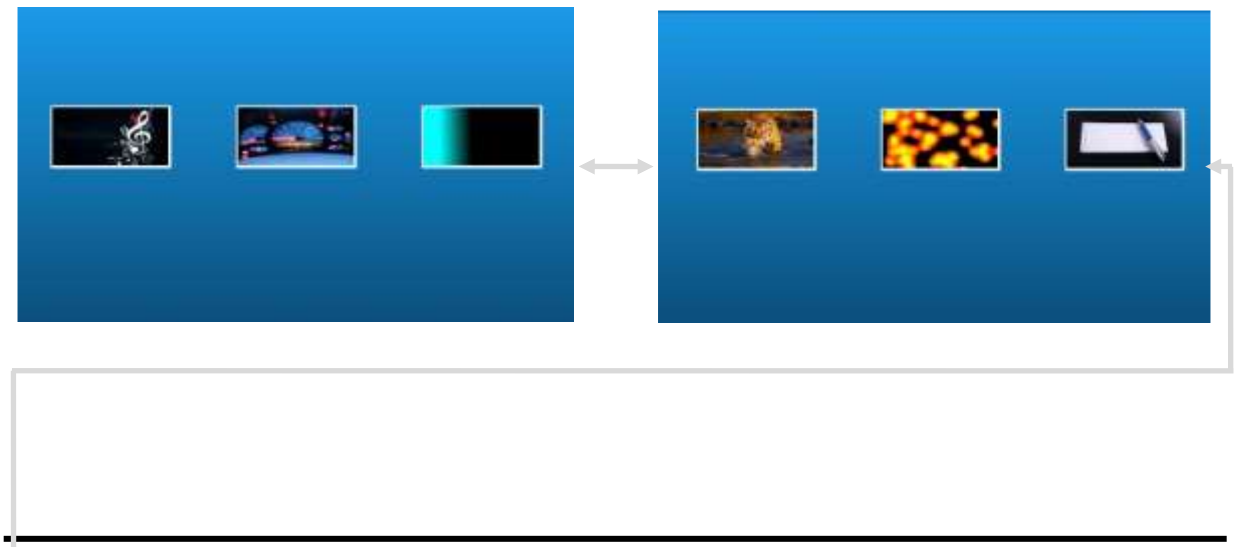
In this case, another Co-Processor list is created which displays a larger version of the bitmap image which had been shown on that icon. It also draws a small 'home' icon at the top-left corner of the screen. The code then waits in a `while` loop for the tag associated with the home button to be detected,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

```
/*API to show the icon with background animation*/
void show_icon(uint8_t iconno)
{
        App_Play_Sound(phost,0x51,100,108);
        do
        {
                Gpu_CoCmd_Dlstart(phost);
                App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
/* Save the graphics context before enter into background animation*/
                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 the graphics context */
            App_WrCoCmd_Buffer(phost,RESTORE_CONTEXT());
            App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
            App_WrCoCmd_Buffer(phost,SAVE_CONTEXT());
            App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(0));

        App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST,BORDER,BORDER,displayImageWidth*2,
displayImageHeight*2));
            App_WrCoCmd_Buffer(phost,CELL(iconno-1));
            App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(128));
            App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_E(128));
            App_WrCoCmd_Buffer(phost,VERTEX2F(((DispWidth-
(displayImageWidth*2))/2)*16,((DispHeight-(displayImageHeight*2))/2)*16));
            App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(256));
            App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_E(256));
            App_WrCoCmd_Buffer(phost,RESTORE_CONTEXT());
            App_WrCoCmd_Buffer(phost,TAG('H'));
            #ifdef BACKGROUND_ANIMATION_4
            App_WrCoCmd_Buffer(phost,COLOR_RGB(0,0,0));
            #endif

         /* Draw a home button a top left screen */
         App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
         App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(START_ICON_HANDLE));
         App_WrCoCmd_Buffer(phost,VERTEX2II(5,5,START_ICON_HANDLE,0));
            App_WrCoCmd_Buffer(phost,DISPLAY());
            Gpu_CoCmd_Swap(phost);
            App_Flush_Co_Buffer(phost);
            Gpu_Hal_WaitCmdfifo_empty(phost);
        }while(App_Read_Tag(phost)!='H');
        App_Play_Sound(phost,0x51,100,108);
        scroller.vel = 0;
}
```

4.2.3 Loopback Menu Style

This style is provided by the menu_loopback() function.



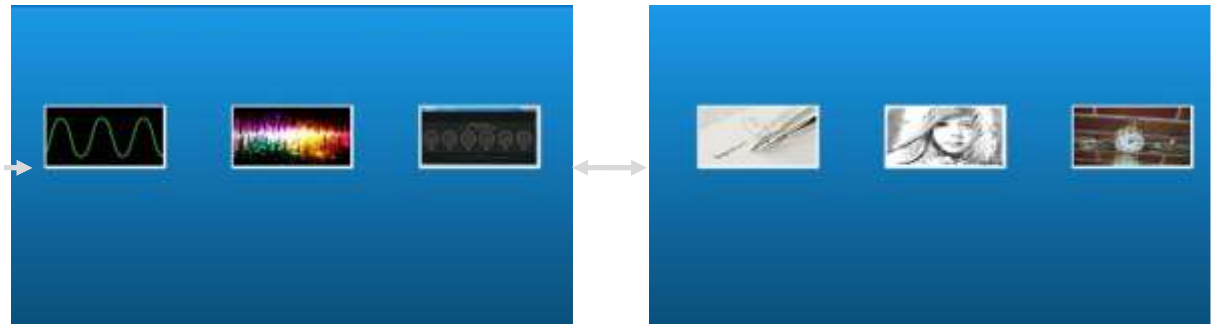


Figure 4.3 Loopback style screenshots

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.

```
uint8_t image_height = displayImageHeight/*50*/,image_width =
displayImageWidth/*100*/;
uint8_t dt = 30,dx,dy;
uint8_t per_frame,no_frames,key_in,current_frame;
int16_t sx,drag,Oy,Ox,dragth,i;
dx = (dt*2)+image_width;
dy = (10*2)+image_height;
per_frame = 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 menu Thumbnails*/
        Load_Thumbnails();
        /*Intilaize the scroller*/
        scroller_init((DispWidth*total_frames)*16);
```

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

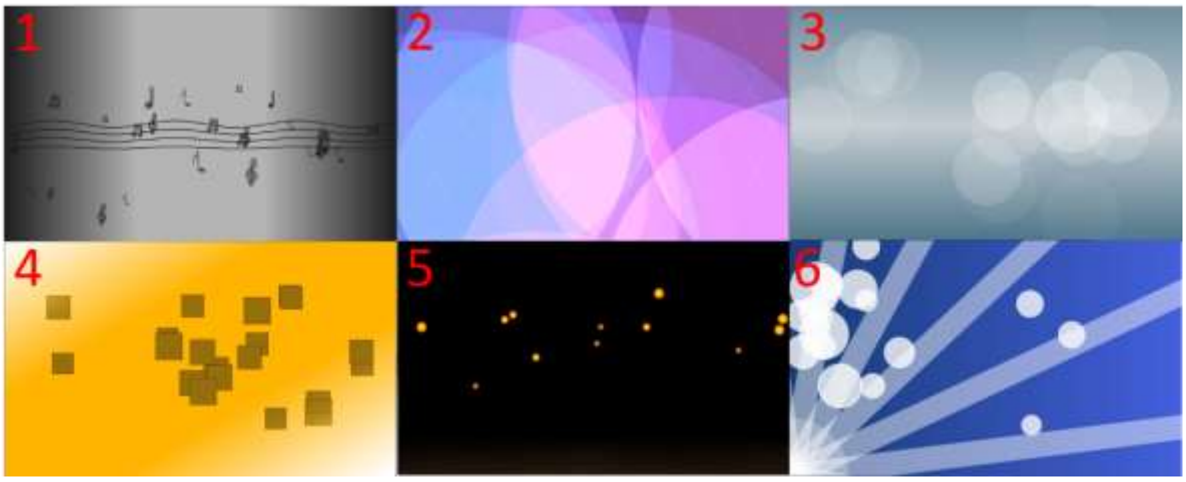
The first section reads the X value of the REG_TOUCH_SCREEN_XY register. The value of this register indicates the coordinates of the point on the screen which is currently being touched (or returns -32768 for the X and Y coordinates if the screen is not being touched). It also checks the REG_TOUCH_TAG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

```
while(1)
{
        /*Read touch screen x variation and tag in*/
         sx =  Gpu_Hal_Rd16(phost,REG_TOUCH_SCREEN_XY + 2);
         key_in =  App_Read_Tag(phost);

         /*Check if any tag in*/
         if(sx!=NOTOUCH)
         keyin_cts++;
         /*Move into the particular frame based on dragdt now 30pixels*/
         if(sx==NOTOUCH)
         keyin_cts = 0;
         /*if tag in but still pendown take a scroller basevalue*/
         else if(keyin_cts>KEYIN_COUNTS)
         key_in = 0;

         if(key_in==0)scroller_run();
         drag = scroller.base>>4;
```

The code then begins to draw background animation. There are 6 available backgrounds:



```
       /*Display list start*/
       Gpu_CoCmd_Dlstart(phost);
       App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
       #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
```

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

```
Oy = (DispHeight-image_width)/2;                         //dt+(dy*((i/col)%row));
current_frame = drag/dx;                          // noof items moved in +/- directions
dragth = drag%dx;

App_WrCoCmd_Buffer(phost,BEGIN(RECTS));
#if defined(DISPLAY_RESOLUTION_WVGA)
```

```
for(i=-1;i<(per_frame+2);i++)
#else
for(i=-1;i<(per_frame+1);i++)
#endif
{
	Ox = dt+dx*i;
	Ox-=dragth;
	if(Ox > (DispWidth+dt) || Ox < -dx) 0;
	else
	{
		App_WrCoCmd_Buffer(phost,VERTEX2F((Ox-1)*16,(Oy-1)*16));

	App_WrCoCmd_Buffer(phost,VERTEX2F((displayImageWidth+Ox+1)*16,(displayImageHeig
ht+Oy+1)*16));  // i pixels wide than image width +1
	}
}
```

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

```
App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));// draw the bitmap
App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(0));
#if defined(DISPLAY_RESOLUTION_WVGA)
for(i=-1;i<(per_frame+2);i++)
#else
for(i=-1;i<(per_frame+1);i++)
#endif
{
	Ox = dt+dx*i;
	Ox-=dragth;
	if(Ox > (DispWidth+dt) || Ox < -dx) 0;
	else
	{
		App_WrCoCmd_Buffer(phost,CELL((MAX_MENUS+i+current_frame)%12));
		App_WrCoCmd_Buffer(phost,TAG((1+i+current_frame)%(MAX_MENUS+1)));
		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 FT8XX 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 FT8XX. 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.

```
App_WrCoCmd_Buffer(phost,DISPLAY());
Gpu_CoCmd_Swap(phost);
App_Flush_Co_Buffer(phost);
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.

4.2.4 Windows 8 Menu Style

This style is provided by the menu_win8() function.

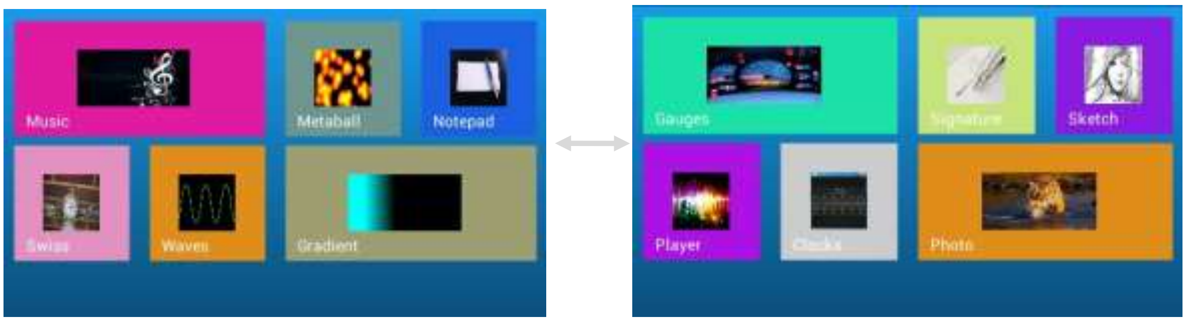


Figure 4.4 Windows 8 style screenshots

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.

```
uint8_t current_frame = 0,total_frames = 0,key_in=0;
int16_t frame_xoffset = 0,frame_xoffset_th= 0;
uint8_t menus_per_frame = 0;
uint8_t col = 3,row = 2,option;
uint16_t image_height = displayImageHeight/*50*/,image_width =
displayImageWidth/*100*/,rectangle_width,rectangle_height;

int16_t Ox,Oy,i,sx;
#if defined(DISPLAY_RESOLUTION_WVGA)
uint8_t frame_xoffset_dt = 30;
#else
uint8_t frame_xoffset_dt = 30;
#endif

#if defined(DISPLAY_RESOLUTION_HVGA_PORTRAIT)
uint8_t blockGap = 40, backgroundBlockHeight = 110;
#elif defined(DISPLAY_RESOLUTION_QVGA)
uint8_t blockGap = 10, backgroundBlockHeight = 110;
#endif

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,0x12,0xE3,
                                          };

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

```
uint8_t  point_offset,frame_point_dt =15;


uint16_t dx = (frame_xoffset_dt*2)+image_width;
uint16_t dy = (10*2)+image_height;
col = DispWidth/dx;
menus_per_frame = col*row;
total_frames = (MAX_MENUS-1)/menus_per_frame;

point_offset = (DispWidth-(total_frames+1)*(MENU_POINTSIZE+frame_point_dt))/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 menu Thumbnails*/
        Load_Thumbnails();
        /*Intilaize the scroller*/
        scroller_init((DispWidth*total_frames)*16);
```

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

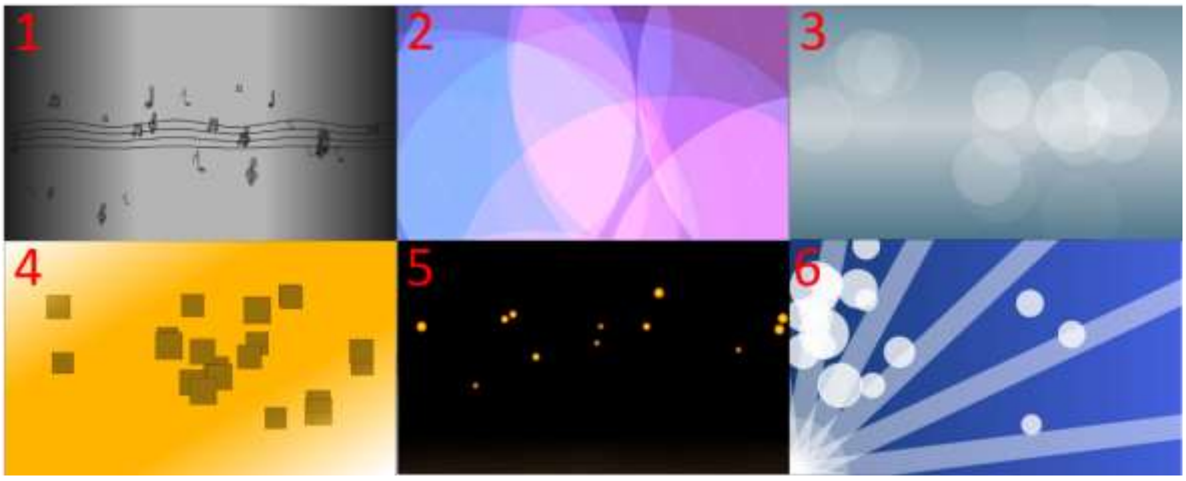
The first section reads the REG_TOUCH_SCREEN_XY register. The value of this register indicates the coordinates of the point on the screen which is currently being touched (or returns -32768 for the X and Y coordinates if the screen is not being touched). It also checks the REG_TOUCH_TAG register to determine 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 variation and tag in*/
        sx =  Gpu_Hal_Rd16(phost,REG_TOUCH_SCREEN_XY + 2);
        key_in =  App_Read_Tag(phost);

        /*Check if any tag in*/
        if(sx!=NOTOUCH)
        keyin_cts++;


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

The code then begins to draw background animation. There are 6 available backgrounds:



```
        /*Display list start*/
        Gpu_CoCmd_Dlstart(phost);
        App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
        #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
```

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. Code for `WQVGA` and `WVGA` screen options:

```
#if defined(DISPLAY_RESOLUTION_WQVGA) || defined(DISPLAY_RESOLUTION_WVGA)
for(option=0;option<3;option++)
{
  switch(option)
  {
    case 0:
      App_WrCoCmd_Buffer(phost,LINE_WIDTH(1*16));
      App_WrCoCmd_Buffer(phost,BEGIN(RECTS));
    break;
    case 1:
      App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
      App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
    break;
    case 2:
      App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
    break;
  }
#if defined(DISPLAY_RESOLUTION_WVGA)
  rectangle_width = 380;
  rectangle_height = 200;
#else
  rectangle_width = 220;
  rectangle_height = 100;
#endif
  for(i=0;i<4;i+=1)
  {
    if(i<2)
    {
      Ox = 10+DispWidth*i;
      Oy = 10;
    }else
    {
#if defined(DISPLAY_RESOLUTION_WVGA)
      Ox = 400+DispWidth*(i%2);
      Oy = 220;
#else
      Ox = 250+DispWidth*(i%2);
      Oy = 120;
#endif
    }
    Ox -= frame_xoffset;
    if(Ox > (DispWidth+frame_xoffset_dt) || Ox < (-DispWidth)) 0;
    else
    {
      App_WrCoCmd_Buffer(phost,TAG(i+1));
      switch(option)
      {
        case 0:
          App_WrCoCmd_Buffer(phost,COLOR_RGB(color[i][0],color[i][1],color[i][2]));
          #if defined(DISPLAY_RESOLUTION_WVGA)
            App_WrCoCmd_Buffer(phost,VERTEX2F((Ox),(Oy)));

App_WrCoCmd_Buffer(phost,VERTEX2F((rectangle_width+Ox),(rectangle_height+Oy)));
// i pixels wide than image width +1
          #else
            App_WrCoCmd_Buffer(phost,VERTEX2F((Ox)*16,(Oy)*16));

App_WrCoCmd_Buffer(phost,VERTEX2F((rectangle_width+Ox)*16,(rectangle_height+Oy)*16));
          #endif
        break;
        case 1:
          App_WrCoCmd_Buffer(phost,CELL(i));
          #if defined(DISPLAY_RESOLUTION_WVGA)
            App_WrCoCmd_Buffer(phost,VERTEX2F((55+Ox),(25+Oy)));
          #else
            App_WrCoCmd_Buffer(phost,VERTEX2F((55+Ox)*16,(25+Oy)*16));
          #endif
        break;

        case 2:
#if defined(DISPLAY_RESOLUTION_WVGA)
          Gpu_CoCmd_Text(phost,Ox+10,Oy+180,26,0,menudetails[i]);
#else
          Gpu_CoCmd_Text(phost,Ox+10,Oy+80,26,0,menudetails[i]);
```

```
#endif
        break;
      }
    }
  }
#if defined(DISPLAY_RESOLUTION_WVGA)
  rectangle_width = 180;
  rectangle_height = 200;
#else
  rectangle_width = 100;
  rectangle_height = 100;
#endif

#if defined(DISPLAY_RESOLUTION_WVGA)
  if(option==1)  App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(512));
#else
  if(option==1)  App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(512));
#endif

  for(i=0;i<8;i+=1)
  {
    if(i<4)
    {
#if defined(DISPLAY_RESOLUTION_WVGA)
      Ox = 400+DispWidth*(i/2)+(rectangle_width*(i%2))+(20*(i%2));
      Oy = 10;
#else
      Ox = 250+DispWidth*(i/2)+(image_width*(i%2))+(20*(i%2));  // 20 is space between
two icon
      //Ox = 150+DispWidth*(i/2)+(image_width*(i%2))+(20*(i%2));
      Oy = 10;
#endif
    }
    else
    {
#if defined(DISPLAY_RESOLUTION_WVGA)
      Ox = 10 + (DispWidth*(i/6)) + (((i-4)%2)*rectangle_width) + (((i-4)%2)*20);
      Oy = 220;
#else
      Ox = 10+DispWidth*(i/6)+ (((i-4)%2)*image_width)+(((i-4)%2)*20);
      Oy = 120;
#endif
    }
    Ox -= frame_xoffset;
    if(Ox > (DispWidth+frame_xoffset_dt) || Ox < -DispWidth) 0;
    else
    {
      App_WrCoCmd_Buffer(phost,TAG(i+5));
      switch(option)
      {
        case 0:

App_WrCoCmd_Buffer(phost,COLOR_RGB(color[i+5][0],color[i+5][1],color[i+5][2]));
            #if defined(DISPLAY_RESOLUTION_WVGA)
              App_WrCoCmd_Buffer(phost,VERTEX2F((Ox),(Oy)));

App_WrCoCmd_Buffer(phost,VERTEX2F((rectangle_width+Ox),(rectangle_height+Oy)));
// i pixels wide than image width +1
            #else
              App_WrCoCmd_Buffer(phost,VERTEX2F((Ox)*16,(Oy)*16));
App_WrCoCmd_Buffer(phost,VERTEX2F((rectangle_width+Ox)*16,(rectangle_height+Oy)*16));
            #endif
        break;

        case 1:
            App_WrCoCmd_Buffer(phost,CELL(i+4));
            #if defined(DISPLAY_RESOLUTION_WVGA)
              App_WrCoCmd_Buffer(phost,VERTEX2F((Ox + (rectangle_width-
displayImageWidth)/2),(25+Oy)));
            #else
              App_WrCoCmd_Buffer(phost,VERTEX2F((25+Ox)*16,(25+Oy)*16));
            #endif
        break;

        case 2:
#if defined(DISPLAY_RESOLUTION_WVGA)
          Gpu_CoCmd_Text(phost,Ox+10,Oy+180,26,0,menudetails[i+4]);
#else
            Gpu_CoCmd_Text(phost,Ox+10,Oy+80,26,0,menudetails[i+4]);
#endif
        break;
      }               // i pixels wide than image width +1
    }
  }

#if defined(DISPLAY_RESOLUTION_WVGA)
  if(option==1)  App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(256));
#else
  if(option==1)  App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(256));
#endif
}
```

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 FT8XX 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 FT8XX. 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.

```
  App_WrCoCmd_Buffer(phost,TAG_MASK(0));
  App_WrCoCmd_Buffer(phost,DISPLAY());
  Gpu_CoCmd_Swap(phost);
  App_Flush_Co_Buffer(phost);
  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.

```
/*API to show the icon with background animation*/
void show_icon(uint8_t iconno)
{
        App_Play_Sound(phost,0x51,100,108);
        do
        {
                Gpu_CoCmd_Dlstart(phost);
                App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
/* Save the graphics context before enter into background animation*/
                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 the graphics context */
                App_WrCoCmd_Buffer(phost,RESTORE_CONTEXT());
                App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
                App_WrCoCmd_Buffer(phost,SAVE_CONTEXT());
                App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(0));

        App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST,BORDER,BORDER,displayImageWidth*2,
displayImageHeight*2));
                App_WrCoCmd_Buffer(phost,CELL(iconno-1));
                App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(128));
                App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_E(128));
                App_WrCoCmd_Buffer(phost,VERTEX2F(((DispWidth-
(displayImageWidth*2))/2)*16,((DispHeight-(displayImageHeight*2))/2)*16));
                App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_A(256));
                App_WrCoCmd_Buffer(phost,BITMAP_TRANSFORM_E(256));
                App_WrCoCmd_Buffer(phost,RESTORE_CONTEXT());
                App_WrCoCmd_Buffer(phost,TAG('H'));
                #ifdef BACKGROUND_ANIMATION_4
                App_WrCoCmd_Buffer(phost,COLOR_RGB(0,0,0));
                #endif

         /* Draw a home button a top left screen */
         App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
         App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(START_ICON_HANDLE));
         App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(START_ICON_ADDR));      // Starting
address in gram
         App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L4, 16, 32));  // format
         App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST, BORDER, BORDER, 32, 32  ));

         App_WrCoCmd_Buffer(phost,VERTEX2II(5,5,START_ICON_HANDLE,0));
                App_WrCoCmd_Buffer(phost,DISPLAY());
                Gpu_CoCmd_Swap(phost);
```

```
            App_Flush_Co_Buffer(phost);
            Gpu_Hal_WaitCmdfifo_empty(phost);
        }while(App_Read_Tag(phost)!='H');
        App_Play_Sound(phost,0x51,100,108);
        scroller.vel = 0;
}
```

5 Background Animation

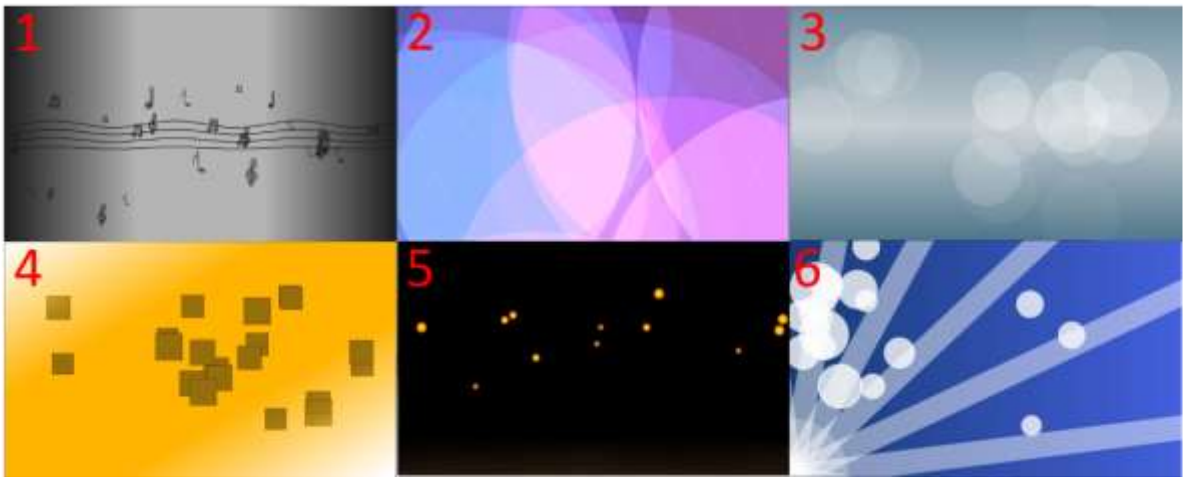
There are six sample background animations implemented in this application. The samples are written in a modular style and the user can add the background animation to their own application. All the background animations are done by using FT8XX graphics primitives.

The icons shown in the previous screenshots are treated as the foreground in the main menu application. The background animation may be designed separately and ported into the Main menu application.

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

All the background animations images are converted to 'L8'/ 'L4' format by using the Bridgetek's 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_t linear(float_t p1,float_t p2,float_t t,uint16_t rate)
{
	float_t st  = (float_t)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));

```
uint16_t smoothstep(float_t p1,float_t p2,float_t t,uint16_t rate)
{
	float_t dst  = (float_t)t/rate;
	float_t st = SQ(dst)*(3-2*dst);
	return p1+(st*(p2-p1));
}
```

5.1.3 Acceleration Interpolation

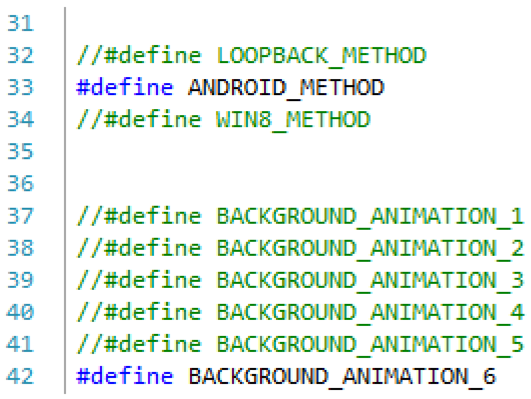
The object moving speed from one point to another is directly proportional to the third parameter time(t).

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

```
float_t acceleration(float_t p1,float_t p2,uint16_t t,uint16_t rate)
{
	float_t dst  = (float_t)t/rate;
	float_t 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 is done by using musical notes bitmaps and gradient bitmap.

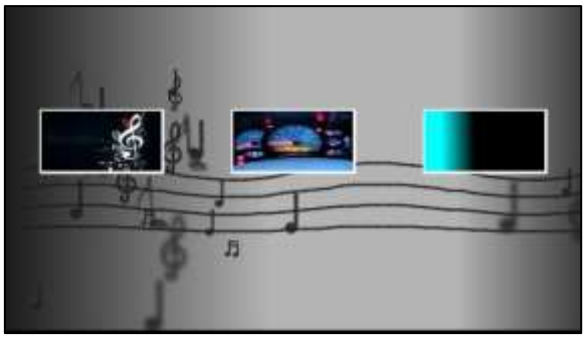


Figure 5.1 Background animation 1

Simple methods are used to construct the background display list.

5.2.2.1 Bitmap Sources

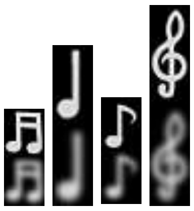


Figure 5.2 Background 1 Bitmap Sources

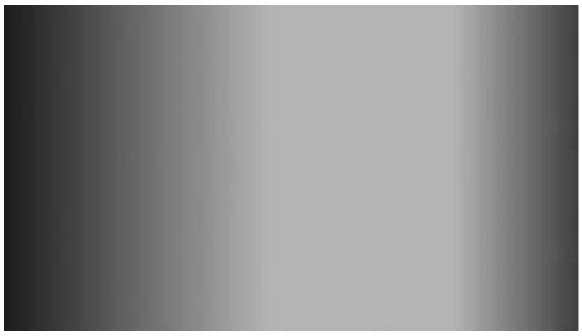


Figure 5.3 Background layer 1

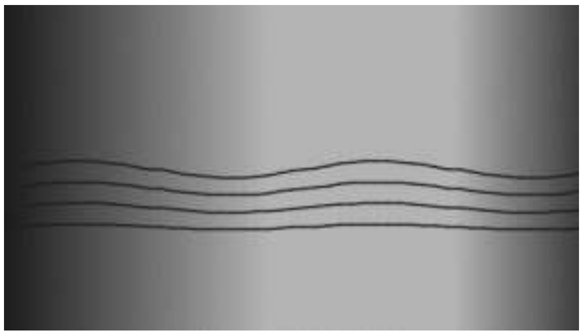


Figure 5.4 Background layer 2

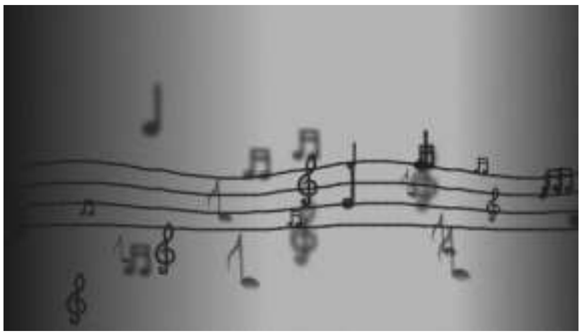


Figure 5.5 Background layer 3

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(uint8_t amp,uint16_t address,uint16_t yoffset)
{
  uint32_t x = 0,y=0;
  for(x=0;x<DispWidth+10;x+=10)
  {
       y = (yoffset) + ((int32_t)amp*qsin(-65536*x/(25*10))/65536);
       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 initialize 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;
       linestripAddress = ((DispWidth+10)/10)*4;
       Sine_wave(15,RAM_G,DispHeight/2);
       Sine_wave(12,RAM_G+linestripAddress,16+(DispHeight/2));
       Sine_wave(9,RAM_G+2*linestripAddress,32+(DispHeight/2));
       Sine_wave(6,RAM_G+3*linestripAddress,48+(DispHeight/2));
       for(i=0;i<30;i++)
       {
              yoffset_array[i] = random(DispHeight);
              bitmap_handle[i] = 4+random(4);
              rate_cts[i] = 300+random(200);
              iteration_cts[i] = random(200);
       }
#if defined(FT81X_ENABLE) && defined(DISPLAY_RESOLUTION_WVGA)
       Gpu_Hal_LoadImageToMemory(phost,"nts1.raw",820*1024L,LOAD);
       Gpu_Hal_LoadImageToMemory(phost,"nts2.raw",822*1024L,LOAD);
       Gpu_Hal_LoadImageToMemory(phost,"nts3.raw",832*1024,LOAD);
       Gpu_Hal_LoadImageToMemory(phost,"nts4.raw",842*1024,LOAD);
       Gpu_Hal_LoadImageToMemory(phost,"hline_H.raw",855*1024L,LOAD);
       App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(4));
       App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(820*1024L));
       App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L4,10,50));
       App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST, BORDER, BORDER, 40, 100));
```

```
        App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(5));
        App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(822*1024L));
        App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L4,25,60));
        App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST, BORDER, BORDER, 100, 120));
        App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(6));
        App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(832*1024L));
        App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L4,10,40));
        App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST, BORDER, BORDER, 40, 80));
        App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(7));
        App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(842*1024L));
        App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L4,10,24));
        App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST, BORDER, BORDER, 40, 48));
        App_WrCoCmd_Buffer(phost,BITMAP_HANDLE(8));
        App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(855*1024L));
        App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L8,800,1));
        App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST, BORDER, REPEAT, 800, 600));
#if (FT81X_ENABLE)
        App_WrCoCmd_Buffer(phost,BITMAP_SIZE_H(800>>9, 600>>9));
#endif
}
```

Note: All the bitmap sources are converted into RAW data by using bridgetek's Image converter utility.

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

```
        wave_cts = DispWidth+10;
        linestripAddress = (wave_cts/10)*4;
        App_WrCoCmd_Buffer(phost,LINE_WIDTH(1*16));
        App_WrCoCmd_Buffer(phost,COLOR_A(255));

        App_WrCoCmd_Buffer(phost,BEGIN(LINE_STRIP));
        Gpu_CoCmd_Append(phost,RAM_G,linestripAddress);
        App_WrCoCmd_Buffer(phost,END());


        App_WrCoCmd_Buffer(phost,BEGIN(LINE_STRIP));
        Gpu_CoCmd_Append(phost,RAM_G+linestripAddress,linestripAddress);
        App_WrCoCmd_Buffer(phost,END());

        App_WrCoCmd_Buffer(phost,BEGIN(LINE_STRIP));
        Gpu_CoCmd_Append(phost,RAM_G+2*linestripAddress,linestripAddress);
        App_WrCoCmd_Buffer(phost,END());

        App_WrCoCmd_Buffer(phost,BEGIN(LINE_STRIP));
        Gpu_CoCmd_Append(phost,RAM_G+3*linestripAddress,linestripAddress);
        App_WrCoCmd_Buffer(phost,END());

        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 FT8XX).

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 is 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

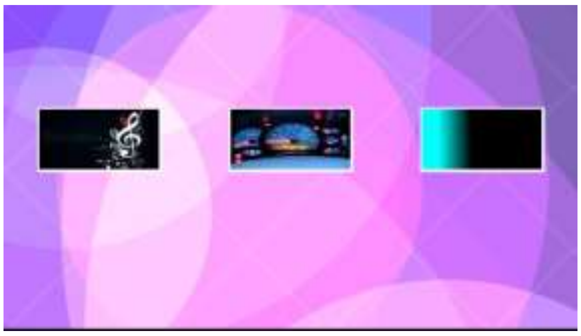


Figure 5.6 Background Animation 2

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 FT8XX 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 required 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 re-entering the application from the animation section, the graphics content then needs to be restored.

```
            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
            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++)
       {
#if defined(DISPLAY_RESOLUTION_WVGA)
              point_size[pts] = 450 *16 + random(61*16);
              xoffset_array[pts] = random(1024)*16;
              yoffset_array[pts] = random(1024)*16;
              color[pts] = random(5);
              dx_pts[pts] = 240*16+random(240*16);
              dy_pts[pts] = 130*16+random(142*16);
#else
              point_size[pts] = 136 *16 +random(375*16);
              xoffset_array[pts] = random(512)*16;
              yoffset_array[pts] = random(512)*16;
              color[pts] = random(5);
              dx_pts[pts] = 240*16+random(240*16);
              dy_pts[pts] = 130*16+random(142*16);
#endif
```

```
        }
}
```

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

```
/* compute points on top */
for(i=0;i<5;i++)
{
        ptradius = point_size[i];
        colorindex = color[i+0];
        App_WrCoCmd_Buffer(phost,COLOR_RGB(ptclrarray[colorindex][0],ptclrarray[colorin
dex][1],ptclrarray[colorindex][2]));
        yoffset = linear(dy_pts[i+0],yoffset_array[i+0],t,1000);
        xoffset = xoffset_array[i+0];
        App_WrCoCmd_Buffer(phost,POINT_SIZE(ptradius));
        App_WrCoCmd_Buffer(phost,VERTEX2F(xoffset,yoffset));
}
...
```

5.2.4 Background animation 3

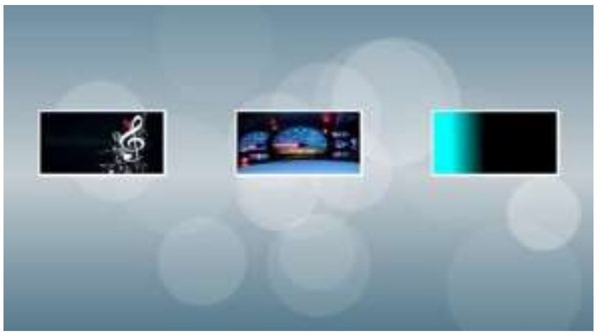


Figure 5.9 Background animation 3

Figure 5.10 Background Layer 1

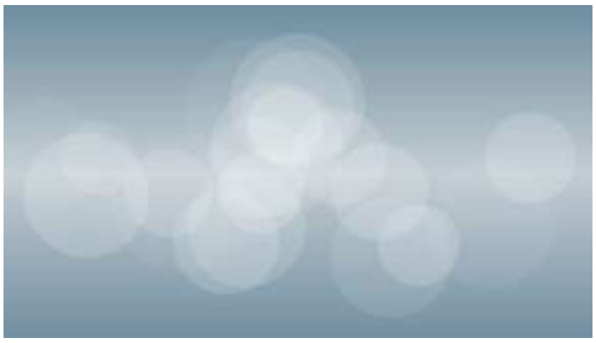


Figure 5.11 Background Layer 2

This background is very easy to do in the FT8XX. We are using two layers for this design; one is a gradient while the second one uses points with additive blending.

Gradient:

```
//draw the cmd gradient with scissors
App_WrCoCmd_Buffer(phost,SCISSOR_SIZE(DispWidth,DispHeight));
App_WrCoCmd_Buffer(phost,SCISSOR_XY(0,0));
Gpu_CoCmd_Gradient(phost,0,0,0x708fa1,0,DispHeight/2,0xc4cdd2);
App_WrCoCmd_Buffer(phost,SCISSOR_XY(0,DispHeight/2));
Gpu_CoCmd_Gradient(phost,0,DispHeight/2,0xc4cdd2,0,DispHeight,0x4f7588);

App_WrCoCmd_Buffer(phost,SCISSOR_XY(0,0));
//reprogram with  default values
App_WrCoCmd_Buffer(phost,SCISSOR_SIZE(DispWidth,DispHeight));
```

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 scissor 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 point’s sizes, offsets are randomly picked,

```
for(i = 0; i<20 ; i++)
{
       if(iteration_cts[i]==0)
       {
              xoffset_array[i] = random(DispWidth);
              yoffset_array[i] = 100+random(DispHeight/4);
              dx_pts[i] = random(DispWidth);
              dy_pts[i] = random(DispHeight);
              rate_cts[i] = 500+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]);
App_WrCoCmd_Buffer(phost,COLOR_A(alpha));
```

5.2.5 Background animation 4

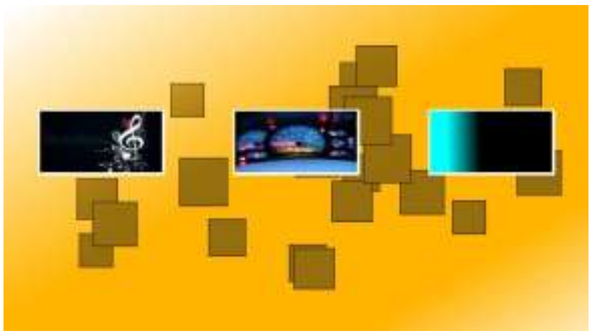


Figure 5.12 Background animation 4

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 FT8XX gradient function is used, here it is a bitmap 512 x 1 pixel. Instead of Circles (POINTS), rectangle is used here. 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 degrees in the code.

```
Gpu_CoCmd_LoadIdentity(phost);
Gpu_CoCmd_Rotate(phost, 60*65536/360);//rotate by 30 degrees clock wise

Gpu_CoCmd_SetMatrix(phost );
```

```
App_WrCoCmd_Buffer(phost,COLOR_MASK(0,0,0,1));
App_WrCoCmd_Buffer(phost,BLEND_FUNC(ONE,ZERO));
App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
App_WrCoCmd_Buffer(phost,COLOR_RGB(0xff,0xb4,0x00));
App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
App_WrCoCmd_Buffer(phost,VERTEX2II(0,0,1,0));

App_WrCoCmd_Buffer(phost,COLOR_MASK(1,1,1,1));
App_WrCoCmd_Buffer(phost,BLEND_FUNC(DST_ALPHA,ONE_MINUS_DST_ALPHA));
App_WrCoCmd_Buffer(phost,COLOR_RGB(0xff,0xb4,0x00));
App_WrCoCmd_Buffer(phost,BEGIN(RECTS));
//App_WrCoCmd_Buffer(phost,VERTEX2II(0,0,0,0));
App_WrCoCmd_Buffer(phost,VERTEX2F(0,0));
//App_WrCoCmd_Buffer(phost,VERTEX2II(DispWidth,DispHeight,0,0));
App_WrCoCmd_Buffer(phost,VERTEX2F(DispWidth*16,DispHeight*16));

App_WrCoCmd_Buffer(phost,BEGIN(RECTS));
App_WrCoCmd_Buffer(phost,BLEND_FUNC(SRC_ALPHA,ONE_MINUS_DST_ALPHA));
App_WrCoCmd_Buffer(phost,COLOR_MASK(1,1,1,0));
App_WrCoCmd_Buffer(phost,COLOR_RGB(0x96,0x6e,0x0d));
App_WrCoCmd_Buffer(phost,LINE_WIDTH(16*1));
for(i=0;i<numBlobs;i++)
{
       int32_t xoffset,yoffset;
       if(0 == i%4)
       {
#if defined(DISPLAY_RESOLUTION_WVGA)
              linesize = 16*(40 + (3*i/4));
#else
              linesize = 16*(25 + (3*i/4));
#endif
       }
       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]);
              App_WrCoCmd_Buffer(phost,VERTEX2F(xoff*16,yoff*16));
              App_WrCoCmd_Buffer(phost,VERTEX2F(xoff*16+linesize,yoff*16+linesize));
       }
}
```

5.2.6 Background animation 5

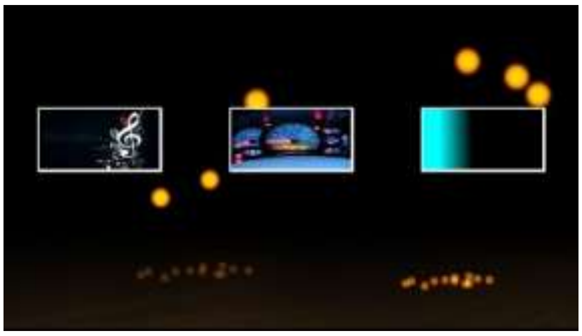


Figure 5.15 Background animation 5

This background animation effect demonstrates a concept of fire balls falling from the sky on to the 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

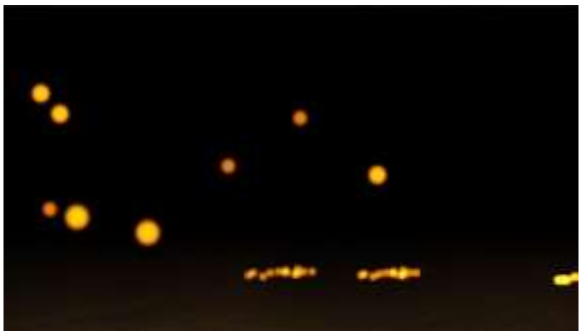


Figure 5.18 Background Layer 3

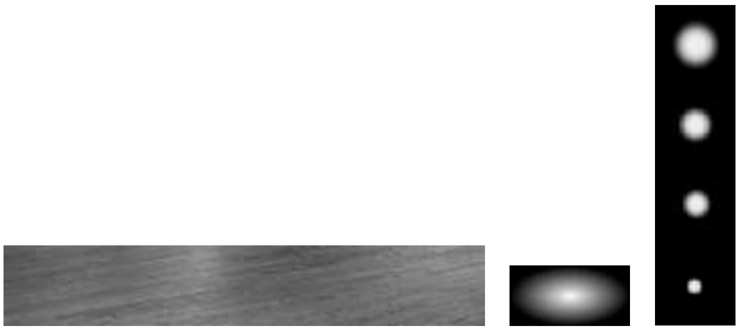


Figure 5.19 Bitmap Sources

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

(c) fireballs when dropping/colliding onto the 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 the 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 collision effect.

```
void collaid_bubbles(uint8_t inc)
{
  int16_t i,j,k,yoff,xoff,temp;
  static uint8_t rate = 50;
  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++)
        {
          App_WrCoCmd_Buffer(phost,COLOR_A(200-firebubbles.disable_cts[j*5+i][k]*10));
          temp =
(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);  //3.14/180=0.01744
          temp =
(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);  //3.14/180=0.01744
          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] = random(DispWidth);
              firebubbles.yoffset_array_source[temp] = -50-random(100);
              if(j==0)
              firebubbles.yoffset_array[temp] = random(20)+(DispHeight-50);
              else if(j==1)
              firebubbles.yoffset_array[temp] = random(20)+(DispHeight-75);
              else if(j==2)
              firebubbles.yoffset_array[temp] = random(20)+(DispHeight-95);

            }
          }
        }
      }
    }
  }
```

5.2.7 Background animation 6

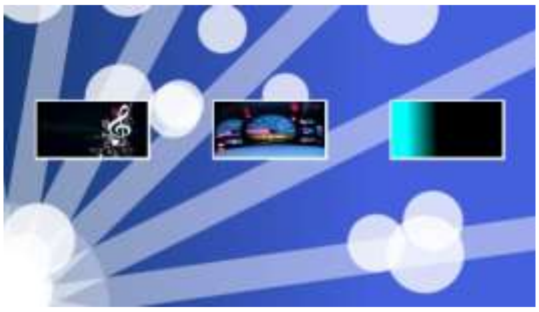


Figure 5.20 Background animation 6

Figure 5.21 Background Layer 1:

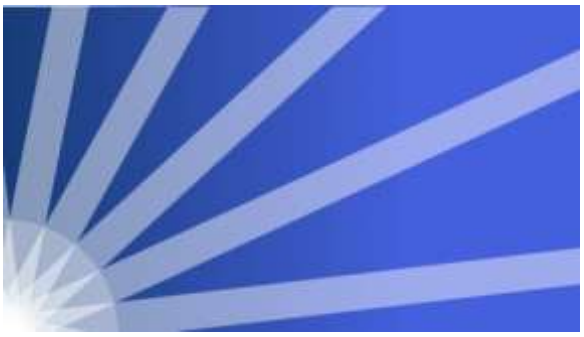


Figure 5.22 Background Layer 2:

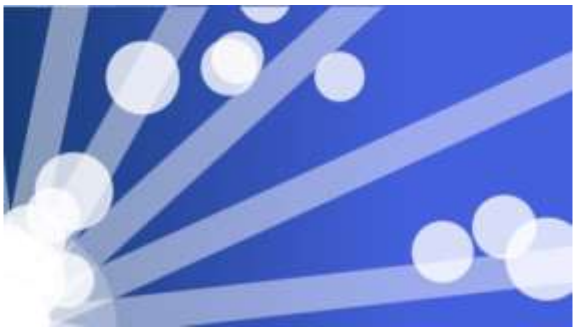


Figure 5.23 Background Layer 3:

In this background there are three layers used, the first layer uses the inbuilt gradient, the second layer uses lines with the origin point and the third layer is displaying bubbles drawn with points all based on FT8XX primitives. All the point sizes and destination are picked randomly during the first iteration. 20 lines are used to create this background 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_draw(int32_t r, float_t th,uint16_t ox,uint16_t oy)
{
  int32_t x, y;
  th = (th * 32768L / 180);
  polarxy(r, th, &x, &y, ox, oy);

#if defined(FT81X_ENABLE)
  App_WrCoCmd_Buffer(phost, VERTEX_FORMAT(0));
  App_WrCoCmd_Buffer(phost,VERTEX2F(x>>4,y>>4));
   App_WrCoCmd_Buffer(phost, VERTEX_FORMAT(4));
#else
  App_WrCoCmd_Buffer(phost,VERTEX2F(x,y));
#endif
```

The lines are drawn by the code below

```
for(z=0;z<20;z++)
{
#if defined(DISPLAY_RESOLUTION_WVGA)
       polar_draw((int32_t)(0),move+z*ANGLE,0,DispHeight);
       polar_draw((int32_t)(950),move+z*ANGLE,0,DispHeight);
#else
       polar_draw((int32_t)(0),move+z*ANGLE,0,DispHeight);
       polar_draw((int32_t)(600),move+z*ANGLE,0,DispHeight);
#endif
}
```

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

```
#if defined(DISPLAY_RESOLUTION_WVGA)
	Gpu_CoCmd_Gradient(phost,0, 0, 0x183c78, DispWidth, 0,0x4560dd);
#else
	Gpu_CoCmd_Gradient(phost,0, 0, 0x183c78, 320, 0,0x4560dd);
#endif
```

For more information about the coprocessor commands refer to the FT8XX series programmer's guide.

6 Running the demonstration code

When running the application, the calibration screen will be displayed first. This uses the FT8XX's built-in calibration routine. It ensures that the FT8XX 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).

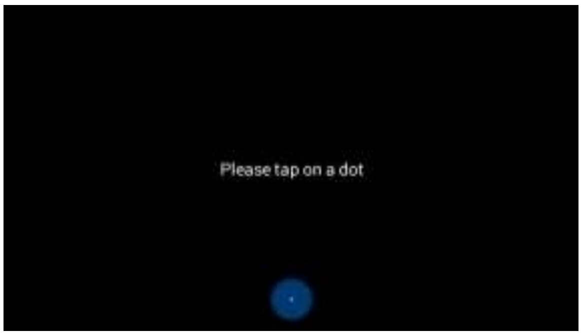


Figure 6.1 Calibration screen

The BRT 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.

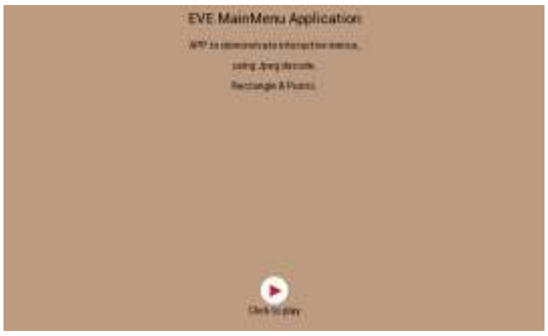


Figure 6.2 Introduction screen

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

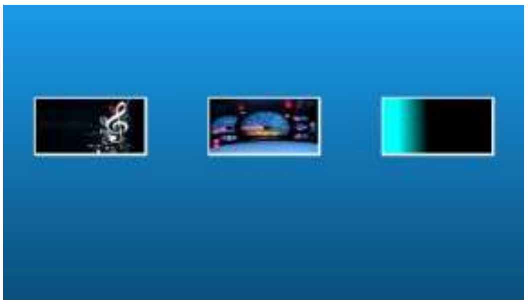


Figure 6.3 Main menu (loopback style)

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.

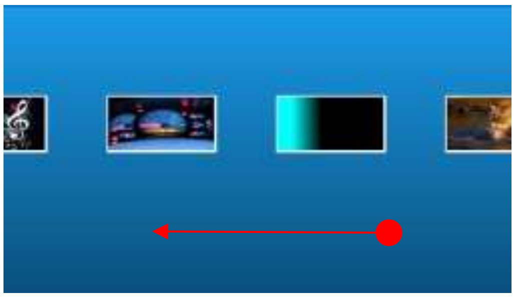


Figure 6.4 Scrolling along the menu

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.

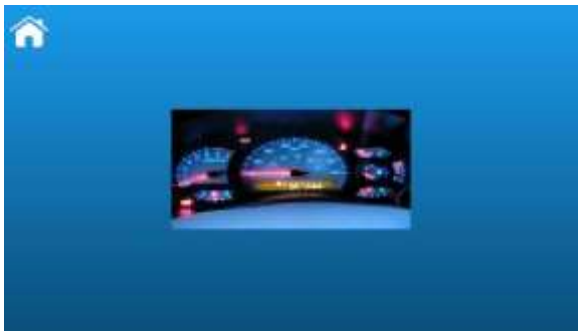


Figure 6.5 After selecting Gauges from the menu

7 Contact Information

Head Quarters – Singapore

Bridgetek Pte Ltd
178 Paya Lebar Road, #07-03
Singapore 409030
Tel: +65 6547 4827
Fax: +65 6841 6071

E-mail (Sales)	sales.apac@brtchip.com
E-mail (Support)	support.apac@brtchip.com

Branch Office – Taipei, Taiwan

Bridgetek Pte Ltd, Taiwan Branch
2 Floor, No. 516, Sec. 1, Nei Hu Road, Nei Hu District
Taipei 114
Taiwan , R.O.C.
Tel: +886 (2) 8797 5691
Fax: +886 (2) 8751 9737

E-mail (Sales)	sales.apac@brtchip.com
E-mail (Support)	support.apac@brtchip.com

Branch Office - Glasgow, United Kingdom

Bridgetek Pte. Ltd.
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)	sales.emea@brtichip.com
E-mail (Support)	support.emea@brtchip.com

Branch Office – Vietnam

Bridgetek VietNam Company Limited
Lutaco Tower Building, 5th Floor, 173A Nguyen Van Troi,
Ward 11, Phu Nhuan District,
Ho Chi Minh City, Vietnam
Tel : 08 38453222
Fax : 08 38455222

E-mail (Sales)	sales.apac@brtchip.com
E-mail (Support)	support.apac@brtchip.com

Web Site

http://brtchip.com/

Distributor and Sales Representatives

Please visit the Sales Network page of the Bridgetek Web site for the contact details of our distributor(s) and sales representative(s) in your country.

Appendix A– References

Document References

- Datasheet for VM800C
- Datasheet for VM800B
- FT8XX Series Programmer Guide
- FT800 Embedded Video Engine Datasheet
- FT81x Embedded Video Engine Datasheet
- Image Conversion Utilities
- AN_391EVE Platform Guide

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 Figures & Tables

List of Figures

List of Tables

Appendix C– Revision History

Document Title:	AN_265 FT_App_MainMenu
Document Reference No.:	BRT_000201
Clearance No.:	BRT#120
Product Page:	http://brtchip.com/product/
Document Feedback:	Send Feedback

Revision	**Changes**	**Date**
1.0	Initial Release	2013-08-21
1.1	Updated version	2013-11-01
1.2	Added a new section on background animation effects	2014-04-10
1.3	Document migrated from FTDI to BRT (Updated company logo; copyright info; contact information; hyperlinks)	2018-01-05