APPLICATION NOTE

AN_268

FT_APP_SIGNALS

Version 1.1

Document Reference No.: FT_000913

Issue Date: 2013-11-01

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

Future Technology Devices International Limited (FTDI)
Unit 1, 2 Seaward Place, Glasgow G41 1HH, United Kingdom
Tel.: +44 (0) 141 429 2777 Fax: + 44 (0) 141 429 2758
Web Site: http://ftdichip.com

Table of Contents

1 Introduction

This design example demonstrates an interactive user interface that simulates a device where input signals are drawn on the screen, similar to an oscilloscope or heart monitor. Signals are drawn through the use of Strips, Points, Blend function and Sound play based on the FT800 platform.

In the Signals application, waveforms such as Sine, Triangular, Saw tooth and Electrocardiogram are generated by software using simple math functions. Sound effects are played when the peak values of the signals are reached. Menus are used to select the waveform type and sampling time scale. When not being used, the menu automatically hides.

This application uses all the features of the FT800: touch, audio and display.

Loading of the necessary elements to show and manipulate the graphics elements is as follows:

- Draw graphics primitives directly through the display list
- Incorporate display list commands to access sound and touch events through reads and writes of the FT800 registers.
- Store the display list in DL_RAM

1.1 Overview

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

Source code for this application is presented in section 4 or at:

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

1.2 Scope

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

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

2 Display Requirements

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

2.1 Waveform display

In the Signals application, there are four waveform types that can be displayed: sine, sawtooth, triangle and "electrocardiogram". Values for the waveforms are calculated as the display lists are generated, although these values could easily be read from an actual sensor in the system.

2.2 Menu

The menu is displayed initially, and then hidden after a few seconds. The touch screen is used to activate (unhide) the menu, to select the desired waveform and to select the rate at which the waveform is displayed.

3 Design Flow

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

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

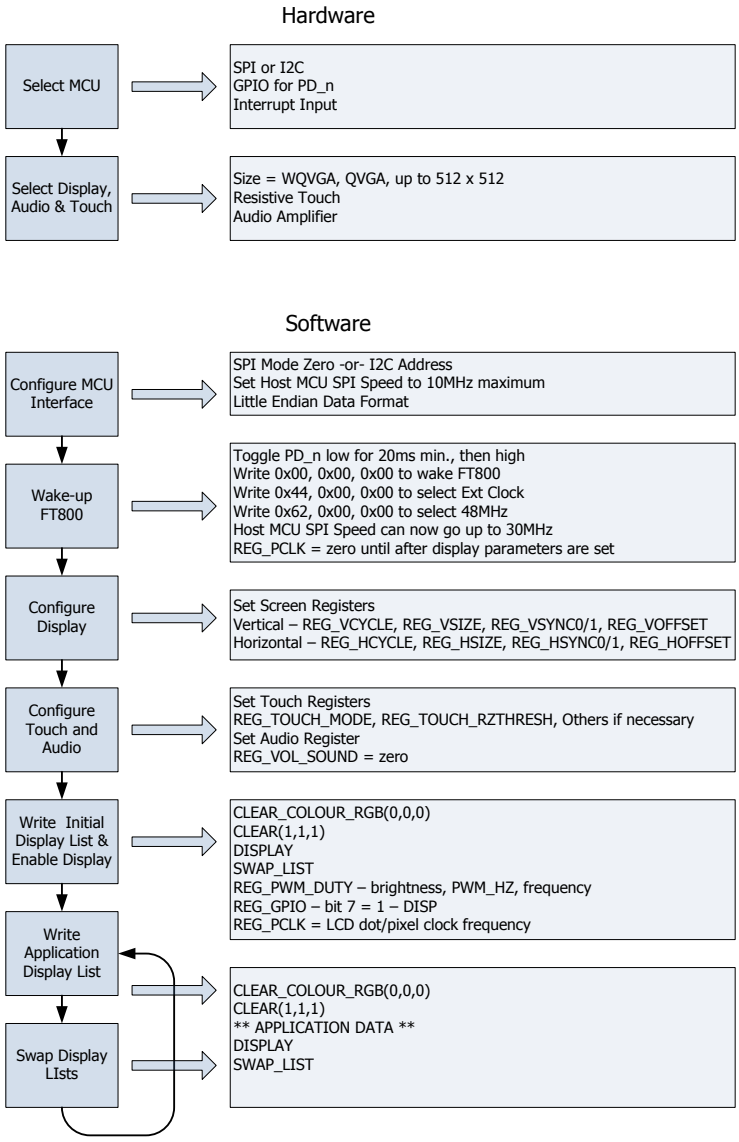


Figure 3.1 Generic EVE Design Flow

3.1 Signals Flowchart

The flowchart below is specific to the Signals application. It begins by displaying a horizontal line and the menu contents. A sine wave is displayed by default, then the menu choices determine which waveform is shown and at what rate.

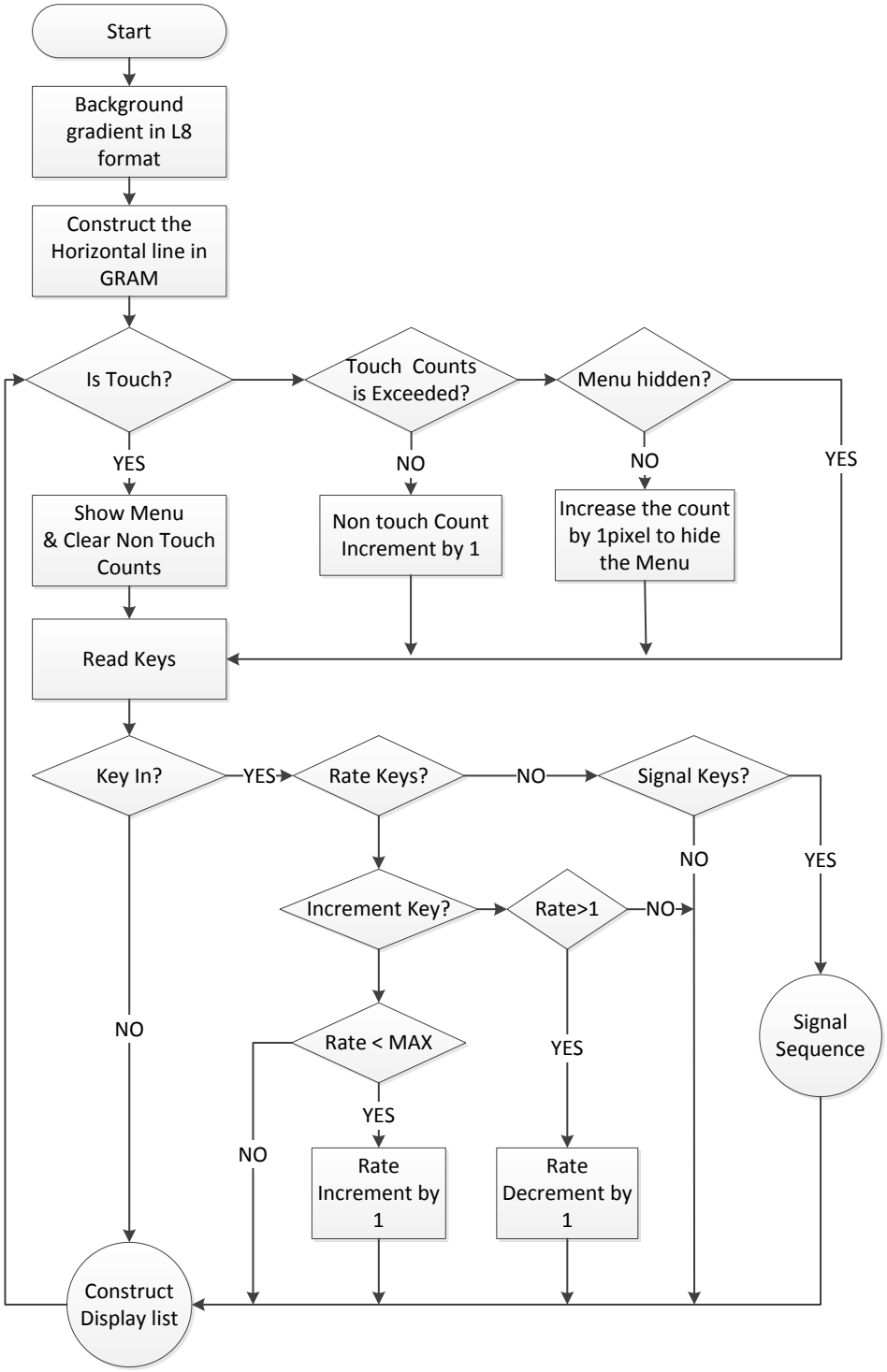
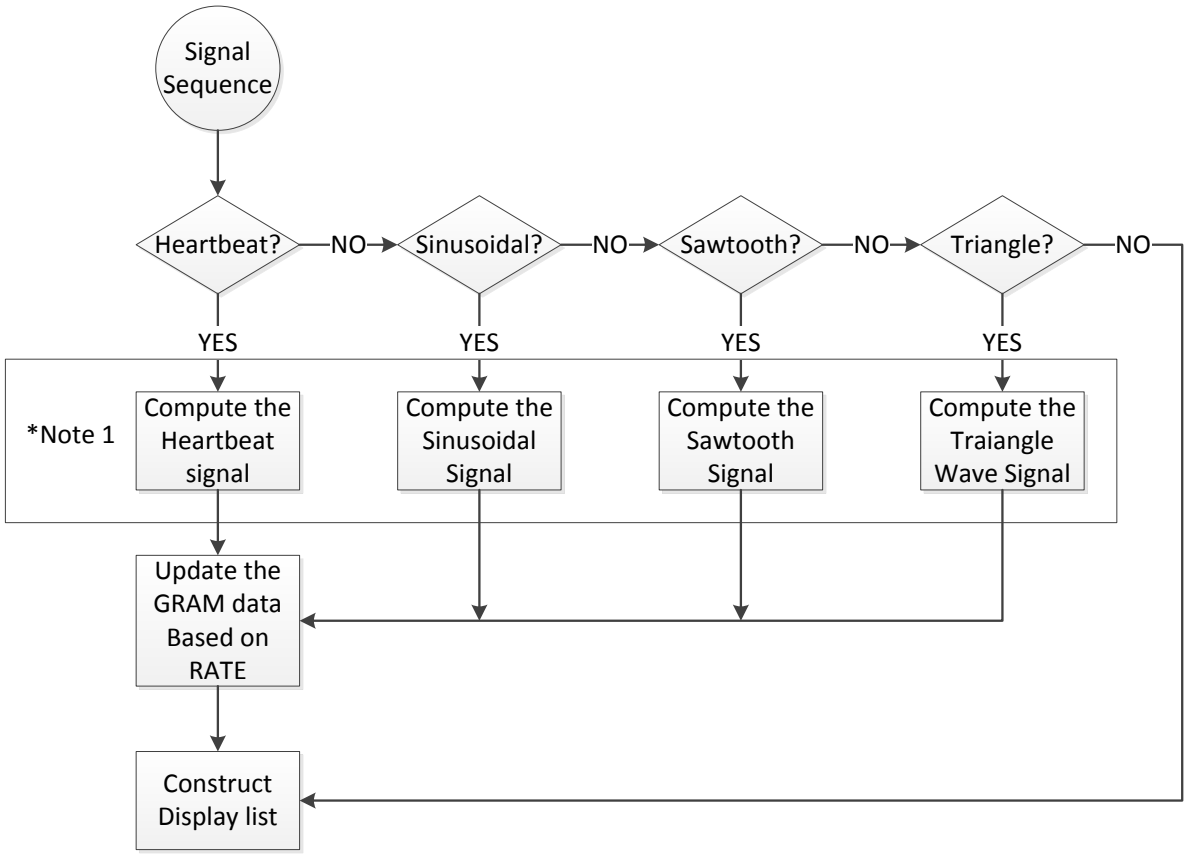


Figure 3.2 Flowchart - Main



NOTE 1: All Signals are hard coded and generated by simple mathematical functions.

Figure 3.3 Flowchart – Signal Selection

4 Description of the Functional Blocks

4.1 System Intialization

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

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

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

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

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

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

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

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

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

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

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

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

And these allow the display to be enabled.

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

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

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

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

To complete the configuration the touch controller should also be calibrated

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

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

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

4.2 Info()

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

A display list is started and cleared:

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

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

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

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

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

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

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

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

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

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

4.3 Waves()

Once the user taps the arrow button, a gradient background is computed and displayed as a backdrop for the displayed signals. This gradient background is then loaded into GRAM. For the Gradient, the height of the display (272 or 240) pixels are used and set the bitmap properties is set to 'Y' while repeated over the width of the display.

```
for(tval=0;tval<=FT_DispHeight/2;tval++)
{
  temp[FT_DispHeight-tval] = temp[tval] = (tval*0.90);
}
```

Here the gradient is written to the GRAM

```
Ft_Gpu_Hal_WrMem(phost,2048L,temp,sizeof(temp));
```

When the display list is constructed, commands to draw the bitmap are given:

```
Ft_App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(2048L));
Ft_App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L8,1,FT_DispHeight));
Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST, REPEAT, BORDER, FT_DispWidth,
FT_DispHeight));
```

Compute the initial zero-line data for the GRAM and load into GRAM

```
for(tval=0;tval<FT_DispWidth;tval+=rate)
        {
          Ft_Gpu_Hal_Wr32(phost,RAM_G+((tval/rate)*4),VERTEX2F(tval*16,y*16));
        }
```

The initial data for the heartbeat is also calculated before entering the loop:

```
y = FT_DispHeight/2;
for(tval=0;tval<10;tval++)
{
  y = y+(beats_Incr[tval]*5);
  beats[tval] = y;
}
```

Note: After this configuration, swap the display list and load into the Graphics Memory, the processing should wait for the Command Processor to be Idle by using REG_CMD_WRITE and REG_CMD_READ registers.

4.3.1 Menu

Once the background and initial data are calculated and loaded into GRAM, the right-side menu is constructed.

This section checks for any touch screen activity. If there is any touch, show the menu, otherwise, hide it after displaying several points of the waveform. The "else" code gives the appearance of sliding the menu off to the right edge.

```
// ========  Menu  ============================================================
    if(istouch())   fg = 1;
    if(fg){ th_to=0; if(hide_x>0)hide_x=0; else
      fg = 0;  }
    else
    { th_to++;
      if(th_to > 250) {
       if( hide_x < 85) hide_x++;   else
       th_to = 0;     }
    }
```

If there is any touch activity, the "Read_keys()" call will check the FT800 for what area of the screen was touched. If it corresponds to a specific item, such as the rate increase/decrease buttons or waveform selection, a "tag" is assigned. These tags are convenient methods of determining which drawn object was touched without having to manually calculate the X-Y position of the touch event and see if it corresponds to the drawn element. The FT800, by assigning tags to a drawn element, automatically knows where the item is and sets the tag value. Later processing can then act on the tag event.

The section of code looks at the tag to see if the rate +/- buttons were tapped. If so, increase or decrease the rate:

```
//==========Option ============================================================
    tag = Read_keys();
    if(tag!=0)
    {
      x = 0;   temp_p = 0;en = 0; temp_x = 0; temp_y = 0; //reset
      if(tag>2)  opt = tag;
      if(tag==1)if(rate>1)rate--;
      if(tag==2)if(rate<6)rate++;
      y = (FT_DispHeight/2);
      for(tval=0;tval<FT_DispWidth;tval+=rate)
      {
        Ft_Gpu_Hal_Wr32(phost,RAM_G+((tval/rate)*4),VERTEX2F(tval*16,y*16));

      }
      add2write = 0;
    }
```

If it was not a rate touch, then check whether if it was a waveform selection:

```
//========= Signals ============================================================
  amp = 100;
  switch(opt)
  {
    case 5:

      Triangle_wave(amp);
    break;

    case 4:
        Sawtooth_wave(amp);
    break;

    case 3:
          Sine_wave(amp);
    break;
```

```
    case 6:
      amp = 50;
      Heartbeat();
    break;
  }
```

4.3.2 Display List

With the rate and waveform selection complete, it's time to construct the display list.

Clear the screen and set the initial color:

```
Ft_Gpu_CoCmd_Dlstart(phost);
Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0x12,0x4A,0x26));
```

Draw the background:

```
Ft_App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(2048L));
Ft_App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L8,1,FT_DispHeight));
Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST, REPEAT, BORDER, FT_DispWidth,
FT_DispHeight));
Ft_App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
Ft_App_WrCoCmd_Buffer(phost,TAG(0));
Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(0,0));
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0x1B,0xE0,0x67));

Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0x1B,0xE0,0x67));
Ft_App_WrCoCmd_Buffer(phost,LINE_WIDTH(2*16));
```

Begin drawing the waveform

```
Ft_App_WrCoCmd_Buffer(phost,BEGIN(LINE_STRIP));
Ft_Gpu_CoCmd_Append(phost,RAM_G,(x/rate)*4);
Ft_App_WrCoCmd_Buffer(phost,END());

Ft_App_WrCoCmd_Buffer(phost,BEGIN(LINE_STRIP));
if((x/rate)<(FT_DispWidth/rate)-(50/rate))
Ft_Gpu_CoCmd_Append(phost,RAM_G+(x/rate)*4+((50/rate)*4),((FT_DispWidth/rate)*4)-((x/rate)*4)-
((50/rate)*4));
Ft_App_WrCoCmd_Buffer(phost,END());
```

Draw the menu and assign tags to the various buttons. Note that the menu is always drawn. Whether it's "moved" into position or hidden depends on whether a touch event occurred earlier:

```
Ft_App_WrCoCmd_Buffer(phost,POINT_SIZE(6*16));
Ft_App_WrCoCmd_Buffer(phost,BEGIN(FTPOINTS));
Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(x*16,y*16));
Ft_App_WrCoCmd_Buffer(phost,END());
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0xff,0xff,0xff));
Ft_App_WrCoCmd_Buffer(phost,COLOR_A(100));
Ft_App_WrCoCmd_Buffer(phost,BEGIN(EDGE_STRIP_R));
Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((hide_x+FT_DispWidth-80)*16,0));
Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((hide_x+FT_DispWidth-80)*16,FT_DispHeight*16));
Ft_App_WrCoCmd_Buffer(phost,COLOR_A(255));
Ft_Gpu_Radiobutton(hide_x+FT_DispWidth-70,FT_DispHeight-48,0xffffff,0,8,3,opt);
Ft_Gpu_Radiobutton(hide_x+FT_DispWidth-70,FT_DispHeight-28,0xffffff,0,8,4,opt);
Ft_Gpu_Radiobutton(hide_x+FT_DispWidth-70,FT_DispHeight-8 ,0xffffff,0,8,5,opt);
Ft_Gpu_Radiobutton(hide_x+FT_DispWidth-70,FT_DispHeight-68,0xffffff,0,8,6,opt);
Ft_Gpu_CoCmd_Text(phost,hide_x+FT_DispWidth-60,FT_DispHeight-48,26,OPT_CENTERY,"Sine");
Ft_Gpu_CoCmd_Text(phost,hide_x+FT_DispWidth-60,FT_DispHeight-28,26,OPT_CENTERY,"Sawtooth");
Ft_Gpu_CoCmd_Text(phost,hide_x+FT_DispWidth-60,FT_DispHeight-8 ,26,OPT_CENTERY,"Triangle");
Ft_Gpu_CoCmd_Text(phost,hide_x+FT_DispWidth-60,FT_DispHeight-68,26,OPT_CENTERY,"ECG");
Ft_Gpu_CoCmd_Text(phost,(hide_x+FT_DispWidth-60),20,30,OPT_CENTERY|OPT_CENTERX,"-");
Ft_Gpu_CoCmd_Text(phost,(hide_x+FT_DispWidth-20),20,30,OPT_CENTERY|OPT_CENTERX,"+");
Ft_Gpu_CoCmd_Text(phost,(hide_x+FT_DispWidth-80),50,28,0,"Rate:");
Ft_Gpu_CoCmd_Number(phost,(hide_x+FT_DispWidth-30),50,28,0,rate);
```

```
Ft_Gpu_CoCmd_Text(phost,(hide_x+FT_DispWidth-80),80,28,0,"Pk:");
Ft_Gpu_CoCmd_Number(phost,(hide_x+FT_DispWidth-40),80,28,0,amp);
Ft_App_WrCoCmd_Buffer(phost,COLOR_A(50));
Ft_App_WrCoCmd_Buffer(phost,POINT_SIZE(15*16));
Ft_App_WrCoCmd_Buffer(phost,BEGIN(FTPOINTS));
Ft_App_WrCoCmd_Buffer(phost,TAG(1));
Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((hide_x+FT_DispWidth-60)*16,20*16));
Ft_App_WrCoCmd_Buffer(phost,TAG(2));
Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((hide_x+FT_DispWidth-20)*16,20*16));

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

4.4 Functionality

The signals are drawn using strips and additive blending. The signals are computed using a simple software equation and can be replaced with any stored database values from an actual sensor. The application also draws an option menu on the right side of the screen with options to change the rate of the signal and options to select the type of signal to be displayed.

The application constantly monitors the user click on rate buttons (either increase or decrease). According to the values given by the user, the signal is plotted with sound being played when the signal hits the peak.

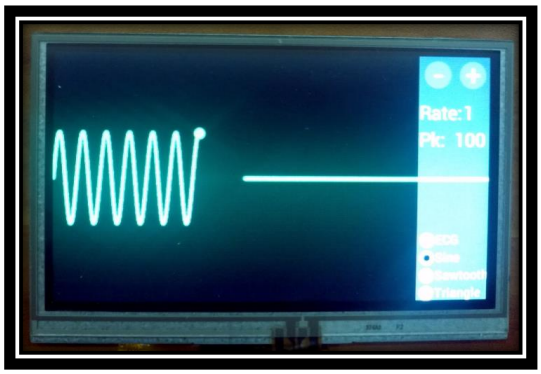


Figure 4.1 Signals Display

4.4.1 Signal Selection

For the demo purpose the signals are generated by using simple functions. The signal selection button is a "Radio Button", drawn by using POINTS. The function Read_keys() will obtain the tag and change the output waveform.

```
tag = Read_keys();
```

4.4.2 Signals

Four example waveforms are calculated on the fly for each point across the screen. The last line in each of the waveform functions is responsible for the new point to be shown.

4.4.2.1 Sine_wave

The sine wave points are calculated using an integer qsin() function based on a stored sine table. If at the peak of the waveform, add in a beep.

```
void Sine_wave(ft_uint8_t amp)
{
  static ft_uint8_t played = 0,change=0;
  x+=rate;
  if(x>FT_DispWidth) x  = 0;
  y = (FT_DispHeight/2) + ((ft_int32_t)amp*qsin(-65536*x/(25*rate))/65536);
  if(played==0 &&  change < y){
  played = 1;
  Play_Sound((108<<8 | 0x10),100); }
  if(change > y)
  played = 0;
  change = y;
  Ft_Gpu_Hal_Wr16(phost,RAM_G+(x/rate)*4,VERTEX2F(x*16,y*16));
}
```

4.4.2.2 Sawtooth_wave

The sawtooth wave is calculated as a simple repeating ramp. As with the sine wave, play a beep at the peak.

```
void Sawtooth_wave(ft_uint8_t amp)
{
  static ft_uint16_t temp=0;
  static ft_uint8_t pk = 0;
  x+=rate;
  if(x>FT_DispWidth){ x  = 0;}
  temp+=2; if(temp>65535L) temp = 0;
  y = (temp % amp);
  pk = y/(amp-2);
  if(pk) Play_Sound((108<<8 | 0x10),100);
  y = (FT_DispHeight/2)-y;
  Ft_Gpu_Hal_Wr16(phost,RAM_G+(x/rate)*4,VERTEX2F(x*16,y*16));
}
```

4.4.2.3 Triangle_wave

The Triangle wave linearly increases amplitude to a maximum, plays a beep, then decreases the amplitude.

```
void Triangle_wave(ft_uint8_t amp)
{
  static ft_uint16_t temp=0;
 static ft_uint8_t pk = 0,dc=0,p=0;
  x+=rate;
  if(x>FT_DispWidth){ x  = 0;}
  temp+=2; if(temp>65535L) temp = 0;
  y = (temp % amp);
  pk = (y/(amp-2))%2;
  dc = (temp / amp)%2;
  if(pk) { if(p==0){ p=1; Play_Sound((108<<8 | 0x10),100); } else  p=0;}
  if(dc) y = (FT_DispHeight/2) -(amp-y);  else
  y = (FT_DispHeight/2) - y;
  Ft_Gpu_Hal_Wr16(phost,RAM_G+(x/rate)*4,VERTEX2F(x*16,y*16));
}
```

4.4.2.4 Heartbeat

The heartbeat is a simulation of an ECG waveform. As with the other waveforms, play a beep when the peak is reached.

```
void Heartbeat()
{
    x+=rate; if(x>FT_DispWidth){ x  = 0;temp_p = 0;temp_y=0;
                                 y=FT_DispHeight/2; en=0;temp_x=0;}
   tx = 5*rate;
   tx = ((temp_p+1)*tx) + temp_p*temp_x;
   if(tx<=x){ if(0==en)    en = 1;}
  if(en==1){
  if(y!=beats[temp_y])
  {
    y += beats_Incr[temp_y] * 5;
    if(y==(FT_DispHeight/2)+beats_Incr[4] * 5)
                Play_Sound((108<<8 | 0x10),100);
  }
  else
  {
    temp_y++;
    if(temp_y>9) {
    temp_y = 0;   temp_p++;
    en = 0;  temp_x = x - tx;}
  }}
  Ft_Gpu_Hal_Wr32(phost,RAM_G+(x/rate)*4,VERTEX2F(x*16,y*16));
 }
```

4.4.3 Signal Plotting

With each waveform point calculated as X increases across the screen, the commands of LINE_STRIP and CMD_APPEND are used to actually display the wave. There is also a gap between the end of the previous wave and new points of the current one.

```
Ft_App_WrCoCmd_Buffer(phost,LINE_WIDTH(2*16));
Ft_App_WrCoCmd_Buffer(phost,BEGIN(LINE_STRIP));
Ft_Gpu_CoCmd_Append(phost,RAM_G,(x/rate)*4);
Ft_App_WrCoCmd_Buffer(phost,END());

Ft_App_WrCoCmd_Buffer(phost,BEGIN(LINE_STRIP));
if((x/rate)<(FT_DispWidth/rate)-(50/rate))
Ft_Gpu_CoCmd_Append(phost,RAM_G+(x/rate)*4+((50/rate)*4),((FT_DispWidth/rate)*4)-((x/rate)*4)-
((50/rate)*4));
```

4.4.4 Menu Hiding

The MCU continuously watches the REG_TOUCH_RAW_X. If the touch is not detected, MCU starts the count to hide the menu. If the count is exceeded, the menu bar is slowly moved pixel by pixel.

```
// ========  Menu  ============================================================
    if(istouch())   fg = 1;
    if(fg){ th_to=0; if(hide_x>0)hide_x=0; else
      fg = 0;  }
    else
    { th_to++;
      if(th_to > 250) {
       if( hide_x < 85) hide_x++;   else
       th_to = 0;     }
    }
```

5 Contact Information

Head Office – Glasgow, UK

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

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

Branch Office – Tigard, Oregon, USA

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

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

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited
(Taiwan)
2F, No. 516, Sec. 1, NeiHu Road
Taipei 114
Taiwan , R.O.C.
Tel: +886 (0) 2 8791 3570
Fax: +886 (0) 2 8791 3576

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

Branch Office – Shanghai, China

Future Technology Devices International Limited
(China)
Room 1103, No. 666 West Huaihai Road,
Shanghai, 200052
China
Tel: +86 21 62351596
Fax: +86 21 62351595

E-mail (Sales) cn.sales@ftdichip.com
E-mail (Support) cn.support@ftdichip.com
E-mail (General Enquiries) cn.admin@ftdichip.com

Web Site

http://ftdichip.com

Distributor and Sales Representatives

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

Appendix A– References

Document References

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

Acronyms and Abbreviations

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

Appendix B – List of Tables & Figures

List of Figures

Appendix C– Revision History

Document Title: AN_268 FT_App_Signals

Document Reference No.: FT_000913

Clearance No.: FTDI# 363

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

Document Feedback: Send Feedback

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