APPLICATION NOTE

AN_267

FT_App_Player

Version 1.1

Document Reference No.: FT_00912

Issue Date: 2013-11-01

This document is to introduce the Player 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 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 a simple music player using graphic display and audio playback on a FT800 platform.

In the Player application, audio signals are visualized by using the bar graph, graphical primitives. The audio is streamed to a speaker using the audio engine.

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

Loading of graphics and audio, along with the construction of the display list will be 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.
- Display list: directly in DL_RAM
- Audio play: via SPI to GRAM

1.1 Overview

The document will provide information on the image creation, tagging of audio and touch capabilities, and the structure of displays lists. In addition, the application note will outline the general steps of the design flow, including display list creation, and integrating the display list with an external host PC using the FTDI C232HM MPSSE cable

To be read in conjunction with the source code, which is provided 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 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
- AN_252 FT800 Audio Primer

2 Display Requirements

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

2.1 Waveform Display

This application demonstrates the usage of graphics primitive bar graph and audio play. The application constantly streams the audio content into the graphics memory from the storage medium and tracks the read pointer of the audio engine. The application also plots a waveform effect on the screen using bar graph primitives and modifies the values based on the input audio content.

This application does not have any user interaction and plays the entire audio file.

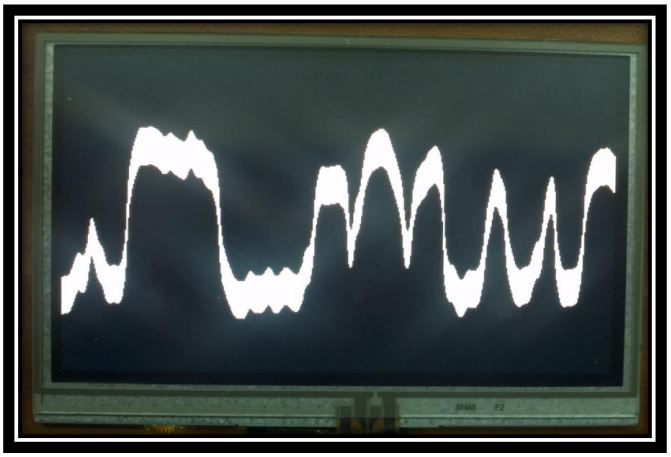

Figure 2.1 App Player Display

2.2 Input File

The FT800 audio player requires a monophonic audio file in uncompressed 8 bit u-Law or 8 bit signed PCM format. The conversion process uses the Audacity audio application and the FTDI aud_cvt utility.

2.3 Menu

The App Player opens with a generic introduction screen that prompts the user to press dots on the touchscreen for calibration. After the FTDI logo appears, pressing the start arrow starts the App Player. No user input is required after this.

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.

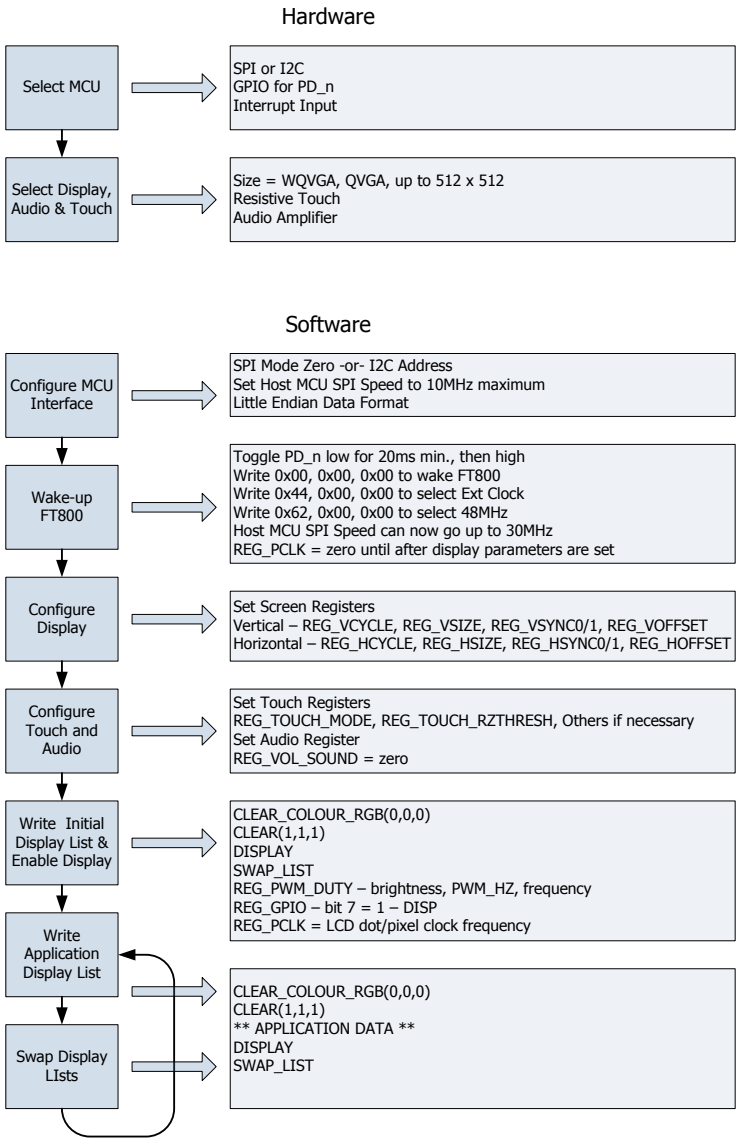

Figure 3.1 Generic EVE Design Flow

3.1 FT App Player Flowchart

The flow chart below is specific to the App Player application. The application uses internal Graphic RAM and SD card storage for storing bitmaps and audio files that may be seen and heard as the application plays.

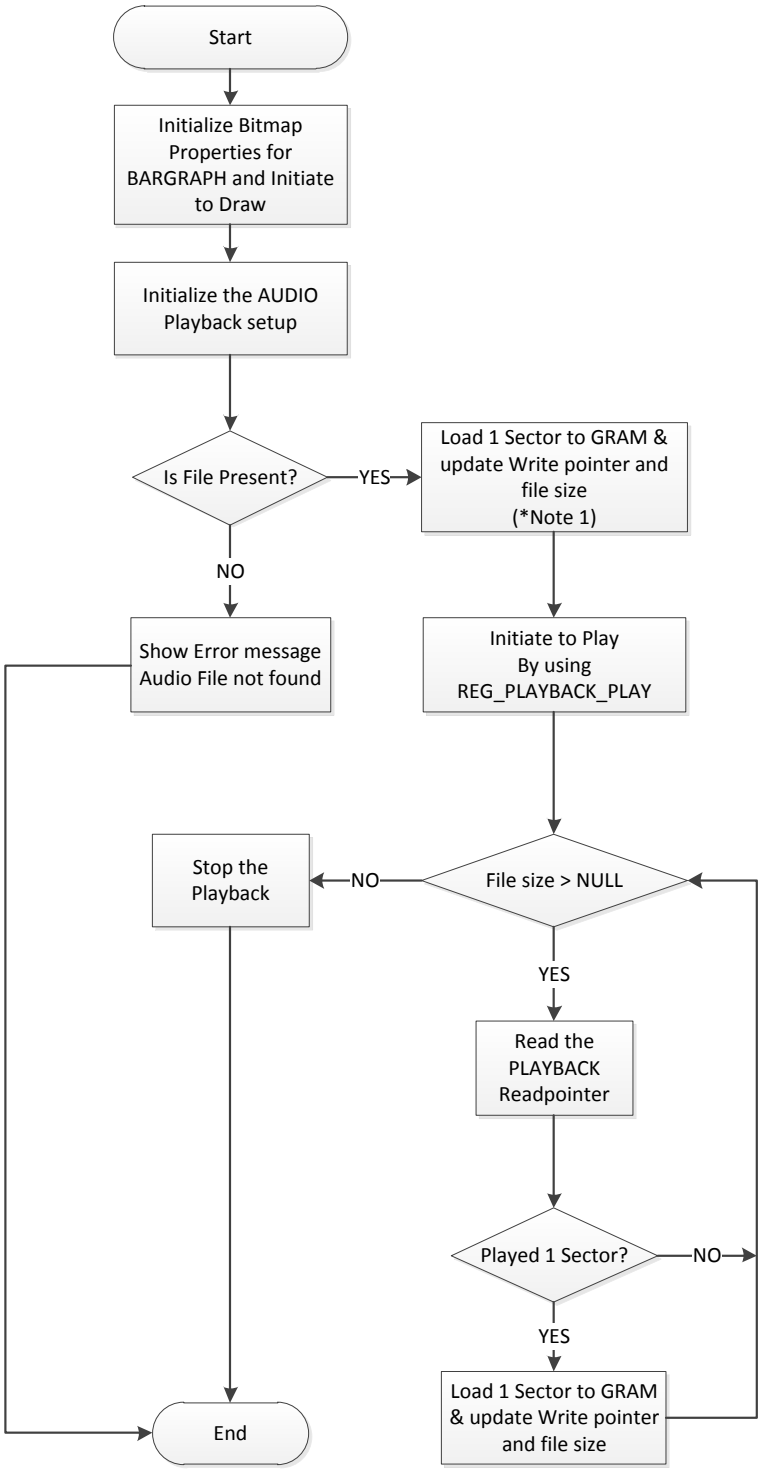

Figure 3.2 App Player Flowchart

4 Description of the Functional Blocks

4.1 System Initilization

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);
```

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 Memory Loading

The on board graphics RAM (GRAM) is used to store the JPEG data and the audio data.

The file is copied from the PC (MPSSE version) or the SD card if present.

e.g. for the audio files, the function Load_afile is called with an address in the 256k Graphics RAM to be written to.

```
void Load_afile(ft_uint32_t add, FILE *afile)
{
  ft_uint8_t pbuff[512],temp[512],tval;
  ft_uint16_t z = 0;
  fread(pbuff,1,512,afile);
  Ft_Gpu_Hal_WrMem(phost,add,pbuff,512L);
```

```
if ((add & 2047L) == 0)                          // every 2kb update the bitmap
  {
    for(z=0;z<512L;z++)
    {
      tval = pbuff[z];
      if (tval & 0x80L)                         // 11 bits of data
      tval ^= 0x7fL;
      temp[z] = tval;
    }
    Ft_Gpu_Hal_WrMem(phost,8192L,temp,512L);
  }
}
```

4.4 Set Properties for Bargraph Function

First step is to initialize bitmap properties:

```
FILE *afile;
ft_uint32_t ftsize=0;
ft_uint16_t wp = 0;
ft_uint32_t rp=0,n,val;

Ft_Gpu_CoCmd_Dlstart(phost);
Ft_Gpu_CoCmd_MemSet(phost,0,0,100L*1024L);
// bit map settings for visualeffect from gram 8192
Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
Ft_App_WrCoCmd_Buffer(phost,BITMAP_SOURCE(8192L));
Ft_App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(BARGRAPH, 256L, 1));

Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(NEAREST, BORDER, BORDER, 256L,     256L));
```

After the initialization of the bitmap properties, tell the processor to draw the bargraph.

```
Ft_App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
Ft_App_WrCoCmd_Buffer(phost,VERTEX2II(0,0,0,0));
Ft_App_WrCoCmd_Buffer(phost,VERTEX2II(256,0,0,1));
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0,0,0));
Ft_App_WrCoCmd_Buffer(phost,VERTEX2II(0,32,0,0));
Ft_App_WrCoCmd_Buffer(phost,VERTEX2II(256L,32,0,1));
Ft_App_WrCoCmd_Buffer(phost,DISPLAY());
Ft_Gpu_CoCmd_Swap(phost);
Ft_App_Flush_Co_Buffer(phost);
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
```

Note: After this configuration, swap the display list and flush into the J1 Memory. Code should wait for J1 Idle by using REG_CMD_WRITE and REG_CMD_READ registers.

4.5 Set up Properties for Audio Playback

Set the sampling rate of the input audio file by using the REG_PLAYBACK_FREQ register. The sampling rate may change based on the structure of the audio file.

```
Ft_Gpu_Hal_Wr32(phost,REG_PLAYBACK_FREQ,44100); //16 bit PCM wav sampling rate 44.1kHz
```

Set the starting location of audio in GRAM by using the REG_PLAYBACK_START register.

```
Ft_Gpu_Hal_Wr32(phost,REG_PLAYBACK_START,0);   //begin at start of .raw file
```

Set the size of the GRAM allocated for audio by using the REG_PLAYBACK_LENGTH register.

```
Ft_Gpu_Hal_Wr32(phost,REG_PLAYBACK_LENGTH,8192L);  //default is 8192L
```

In this demo application, the allocated size of GRAM is 8KB.

Set the audio file type to u-Law by using the REG_PLAYBACK _FORMAT register.

```
Ft_Gpu_Hal_Wr32(phost,REG_PLAYBACK_FORMAT,1);  // default is ULAW_SAMPLES
```

Set the speaker volume by using the REG_VOL_PB register. Default volume is 100

```
Ft_Gpu_Hal_Wr8(phost,REG_VOL_PB,100);
```

Read in the audio file from the project subdirectory as follows:

```
afile = fopen("..\\..\\..\\Test\\DARKAGES.ULW","rb"); //read binary (rb) DARKAGES.ULW
```

In this application, the audio file is "DARKAGES.ULW". The size of this file is about 9 MB. The 8KB of GRAM is looped for 9MB audio by setting the register REG_PLAYBACK_LOOP to 1:

```
Ft_Gpu_Hal_Wr32(phost,REG_PLAYBACK_LOOP,1);  //default is 1 loop
```

If the audio file is less than 8KB, set the register REG_PLAYBACK_LOOP to 0.

The 9 MB audio file will be processed by performing multiple read/writes of GRAM memory. The REG_PLAYBACK_READPTR command keeps track of how much of the audio file has been read.

```
fseek(afile,0,SEEK_END);
  ftsize = ftell(afile);
  fseek(afile,0,SEEK_SET);
  Load_afile(0,afile);
  wp = 512L;
  Ft_Gpu_Hal_Wr8(phost,REG_PLAYBACK_PLAY,1);
  while(ftsize > 0)
  {
    rp = Ft_Gpu_Hal_Rd16(phost,REG_PLAYBACK_READPTR);
    val = 8191L & (rp-wp);
    if (val > 512L)
    {
      n = min(512L,ftsize);
      Load_afile(wp,afile);
      wp = (wp +512L) & 8191L;
      ftsize-=n;
    }
  }
   Ft_Gpu_Hal_Wr8(phost,REG_VOL_PB,0);
   Ft_Gpu_Hal_Wr8(phost,REG_PLAYBACK_PLAY,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. Datasheet for VM800C
2. Datasheet for VM800B
3. FT800 programmer guide FT_000793.
4. FT800 Embedded Video Engine Datasheet FT_000792

Acronyms and Abbreviations

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

Appendix B – List of Tables & Figures

List of Figures

Appendix C– Revision History

Document Title: AN_267 FT_App_Player

Document Reference No.: FT_00912

Clearance No.: FTDI# 362

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

Document Feedback: Send Feedback

Revision	Changes	Date
0.1	Initial draft release	2013-07-18
1.0	Version 1.0 updated wrt review comments	2013-08-21
1.1	Version 1.1	2012-11-01