Application Note

AN_252

FT800 Audio Primer

Version 1.0

Issue Date: 2013-08-06

The FT800 provides an inexpensive solution for adding rich graphics, touch and audio to an embedded system. This application note focuses on the FT800 audio controller, related circuitry and programming techniques.

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

What is EVE?

EVE, or the FTDI Embedded Video Engine, is a family of ICs designed to control TFT, resistive touch displays. The first device in this family is the FT800 which in addition to controlling the display also includes embedded support for touch control and audio output.

This document focuses on the audio features available through the FT800, from the audio circuitry to convert the PWM output to the programming techniques for enabling and using the audio controller.

2 Audio Engine

Various audio effects and files can be played by the FT800. The output is provided as a PWM signal on a single pin, AUDIO_L. There are two audio sources, the Sound Synthesizer and the Audio Playback.

2.1 Sound Synthesizer

Sound effects are pre-loaded in a ROM wave library and do not require the use of any of the FT800 RAM space. Most audible user feedback can be provided through these effects, such as a click when an on-screen button is pressed, DTMF tones for telecom or an alarm panel warning. Most effects will play once and then stop. Others will play continuously. The full list of tones and features are shown below:

Value	Effect	Continuous	Pitch adjust
00h	Silence	Y	N
01h	square wave	Y	Y
02h	sine wave	Y	Y
03h	sawtooth wave	Y	Y
04h	triangle wave	Y	Y
05h	Beeping	Y	Y
06h	Alarm	Y	Y
07h	Warble	Y	Y
08h	Carousel	Y	Y
10h	1 short pip	N	Y
11h	2 short pips	N	Y
12h	3 short pips	N	Y
13h	4 short pips	N	Y
14h	5 short pips	N	Y
15h	6 short pips	N	Y
16h	7 short pips	N	Y
17h	8 short pips	N	Y
18h	9 short pips	N	Y
19h	10 short pips	N	Y
1Ah	11 short pips	N	Y
1Bh	12 short pips	N	Y
1Ch	13 short pips	N	Y
1Dh	14 short pips	N	Y
1Eh	15 short pips	N	Y
1Fh	16 short pips	N	Y
23h	DTMF #	Y	N
2Ch	DTMF *	Y	N
30h	DTMF 0	Y	N
31h	DTMF 1	Y	N
32h	DTMF 2	Y	N
33h	DTMF 3	Y	N
34h	DTMF 4	Y	N
35h	DTMF 5	Y	N
36h	DTMF 6	Y	N
37h	DTMF 7	Y	N
38h	DTMF 8	Y	N
39h	DTMF 9	Y	N
40h	harp	N	Y
41h	xylophone	N	Y
42h	tuba	N	Y
43h	glockenspiel	N	Y
44h	organ	N	Y
45h	trumpet	N	Y
46h	piano	N	Y
47h	chimes	N	Y
48h	music box	N	Y
49h	bell	N	Y
50h	click	N	N
51h	switch	N	N
52h	cowbell	N	N
53h	notch	N	N
54h	hihat	N	N
55h	kickdrum	N	N
56h	pop	N	N
57h	clack	N	N
58h	chack	N	N
60h	mute	N	N
61h	unmute	N	N

Table 2.1 FT800 Synthesized Sound Effects

Many of the effects allow pitch control (MIDI note), so various tones can be generated. Standard MIDI note assignments are used:

MIDI note	ANSI note	Freq (Hz)	MIDI note	ANSI note	Freq (Hz)
21	A0	27.5	65	F4	349.2
22	A#0	29.1	66	F#4	370.0
23	B0	30.9	67	G4	392.0
24	C1	32.7	68	G#4	415.3
25	C#1	34.6	69	A4	440.0
26	D1	36.7	70	A#4	466.2
27	D#1	38.9	71	B4	493.9
28	E1	41.2	72	C5	523.3
29	F1	43.7	73	C#5	554.4
30	F#1	46.2	74	D5	587.3
31	G1	49.0	75	D#5	622.3
32	G#1	51.9	76	E5	659.3
33	A1	55.0	77	F5	698.5
34	A#1	58.3	78	F#5	740.0
35	B1	61.7	79	G5	784.0
36	C2	65.4	80	G#5	830.6
37	C#2	69.3	81	A5	880.0
38	D2	73.4	82	A#5	932.3
39	D#2	77.8	83	B5	987.8
40	E2	82.4	84	C6	1046.5
41	F2	87.3	85	C#6	1108.7
42	F#2	92.5	86	D6	1174.7
43	G2	98.0	87	D#6	1244.5
44	G#2	103.8	88	E6	1318.5
45	A2	110.0	89	F6	1396.9
46	A#2	116.5	90	F#6	1480.0
47	B2	123.5	91	G6	1568.0
48	C3	130.8	92	G#6	1661.2
49	C#3	138.6	93	A6	1760.0
50	D3	146.8	94	A#6	1864.7
51	D#3	155.6	95	B6	1975.5
52	E3	164.8	96	C7	2093.0
53	F3	174.6	97	C#7	2217.5
54	F#3	185.0	98	D7	2349.3
55	G3	196.0	99	D#7	2489.0
56	G#3	207.7	100	E7	2637.0
57	A3	220.0	101	F7	2793.8
58	A#3	233.1	102	F#7	2960.0
59	B3	246.9	103	G7	3136.0
60	C4	261.6	104	G#7	3322.4
61	C#4	277.2	105	A7	3520.0
62	D4	293.7	106	A#7	3729.3
63	D#4	311.1	107	B7	3951.1
64	E4	329.6	108	C8	4186.0

Table 2.2 MIDI Note/Pitch Values

Sound synthesis is controlled by the following registers:

- REG_SOUND
  - Bits 31-16 = Don't care
  - Bits 15-8 = MIDI note (pitch)
  - Bits 7-0 Effect
- REG_PLAY
  - Bits 31-1 = Don't care
  - Bit0 = Start Play / Play Status
    - Write = 1 to start playing the selection in REG_SOUND
    - Read = 1 indicates the effect is currently playing
    - Read = 0 indicates the effect has completed
- REG_VOL_SOUND
  - Bits 31-8 = don't care
  - Bits 7-0 = output volume

Reads and writes to FT800 memory space are handled through the "little endian" format, where the first byte will be the least significant. For example, consider a piano effect (0x46) playing A4 (0x45). The data on the SPI or I2C interface would consist of a Host Memory Write sequence with the following data, on order:

0x90 = Host Memory Write transfer (0x80) plus first byte of register address (0x10)
0x24 = second byte of register address
0x84 = third byte of register address
0x46 = piano effect
0x45 = note/pitch = A4 (440Hz)
0x00 = don't care, optional
0x00 = don't care, optional

Note that the last two bytes do not have to be written, so a 16-bit Host Memory Write can actually satisfy the values needed in the register. Further explanation of the Host Memory Write, Host Memory Read and Host Memory Command data transfers are found in AN_240 FT800 From the Ground Up.

2.2 Audio Files

There may be applications where something more than simple tone synthesis is necessary. For example a voice prompt or other announcement may be necessary to give the user specific instructions. The FT800 supports playback of files in the following single-channel (mono) formats:

- 8-bits signed PCM – uncompressed raw audio
- 8-bits µLAW – non-linear compressed audio
- 4-bits IMA-ADPCM – further compressed µLAW where each byte contains two 4-bit samples

The FT800 has 256Kbytes of object ram (RAM_G) to hold video objects (images and fonts) and audio objects (recorded sounds). The space must be managed by the host MCU so that information is not overwritten until after it is no longer required.

Audio files are loaded through the Host Memory Write transaction to available space and are required to be 8-byte aligned (64-bit).

Once an audio file is loaded into the RAM_G memory, playback is controlled by the following registers:

- REG_PLAYBACK_START
  - Bits 31-20 = don't care
  - Bits 19-0 = 20-bit starting address of the file within RAM_G
- REG_PLAYBACK_LENGTH
  - Bits 31-20 = don't care
  - Bits 19-0 = 20-bit length of the file within RAM_G
- REG_PLAYBACK_FREQ
  - Bits 31-16 = don't care
  - Bits 15-0 = 16-bit Playback sampling rate frequency, in Hz

- REG_PLAYBACK_FORMAT
  - Bits 31-2 = don't care
  - Bits 1-0 = 2-bit Playback format
    - 0 = Linear, or uncompressed PCM
    - 1 = μLaw
    - 2 = ADPCM
    - 3 = undefined
- REG_PLAYBACK_LOOP
  - Bits 31-1 = don't care
  - Bit 0 = 1-bit Playback style
    - 0 = play once
    - 1 = play continuous
- REG_PLAYBACK_PLAY
  - Bits 31-1 = don't care
  - Bit 0 = Start Play / Play Status
    - Write = 0 or 1 to start playing the selection in REG_SOUND
    - Read = 1 indicates the file is currently playing
    - Read = 0 indicates the file has completed
- REG_PLAYBACK_READPTR
  - Bits 31-20 = don't care
  - Bits 19-0 = 20-bit pointer of the current playback location
- REG_VOL_PB
  - Bits 31-8 = don't care
  - Bits 7-0 = output volume

2.2.1 Conversion Utility

FTDI provide an audio file conversion utility called "AUD_CVT" to take a common file format and create the three types of files supported by the FT800. A link to this utility is provided in the "Appendix A – References" section of this document.

The source file is assumed to be a raw (uncompressed) 16bit PCM, mono WAV file. Such a file can be created through a commonly available audio editing program, such as Audacity. Once the source file is available, simply run the utility at a command prompt:

```
aud_cvt  -i  inputfilename  -f  format

where "format"    = 0 for 8-bit signed PCM
                  = 1 for 8-bit μLaw
                  = 2 for 4-bit IMA ADPCM
```

A folder with the resulting files will be created. These files can then be used to load into the RAM_G memory area of the FT800.

The audio conversion utility will output a raw and a compressed file for each selected format. It may be helpful at the host MCU to store the compressed files, and then expand them into the RAM_G buffer when needed. See Section 4 for details.

3 Hardware

The audio output provided by the FT800 consists of a single-channel PWM signal, AUDIO_L. This signal needs to be converted to analog so that it may be heard through a speaker. The conversion consists of several stages.

In addition to AUDIO_L, the FT800 interrupt output can be utilized. An interrupt can be generated whenever an effect or file has completed playing. This is useful to queue the host MCU to perform another FT800 task. Interrupt circuitry is not shown here.

3.1 Power Supply

In general, digital video signals tend to produce a bit of noise on the power and ground rails, especially when many signals are transitioning simultaneously. Setting the FT800 "CSPREAD" feature will offset some of the transitions on the video output, but noise will still be present.

The first defense against high frequency noise on the audio signals is a clean power supply. Ferrites and capacitors between power and ground will help reduce the noise so that the output of any buffers or amplifiers using the power supply will not have the noise superimposed on the audio. The circuit below is one example of an audio power supply filter:

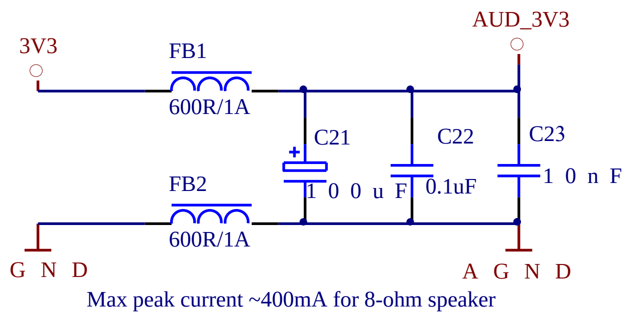

Figure 3.1 Audio Power Supply Filter

3.2 Buffer

With the power supply cleaned up, next in line is a logic-level buffer to provide enough drive current for the subsequent stage. Note that the buffer is powered by the audio power supply:

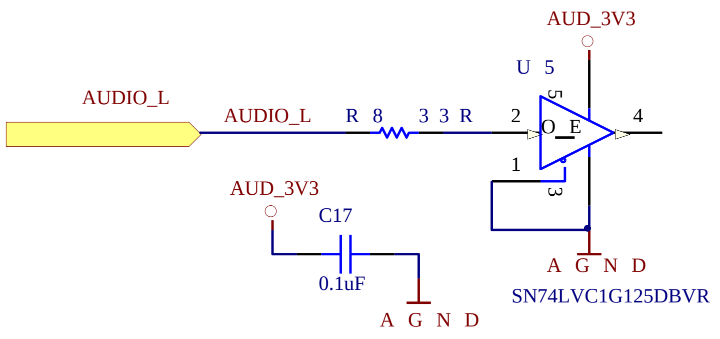

Figure 3.2 Audio PWM Buffer

3.3 3-stage Low-Pass Filter

The output of the buffer is then fed into a 3-stage low-pass filter. This provides 60dB / octave attenuation of frequencies above the cut-off frequency. This steep filter further aids in digital noise suppression. In this example, the cut-off frequency is approximately 34kHz:

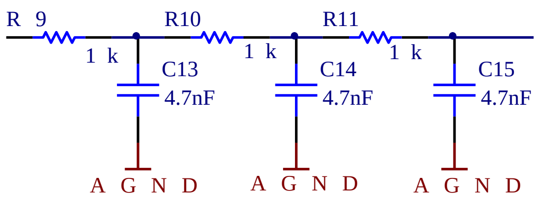

Figure 3.3 Audio Low-Pass Filter (3rd Order)

Since the AUDIO_L output is referenced to the 3.3V VCC power supply, the audio signal is centered on ~1.65V at the R11/C15 junction.

3.4 Audio amplifier

The final stage in the audio circuit is an amplifier. C16 blocks any DC offset before the input. This amplifier will directly drive a speaker. Component values are derived from the amplifier datasheet.

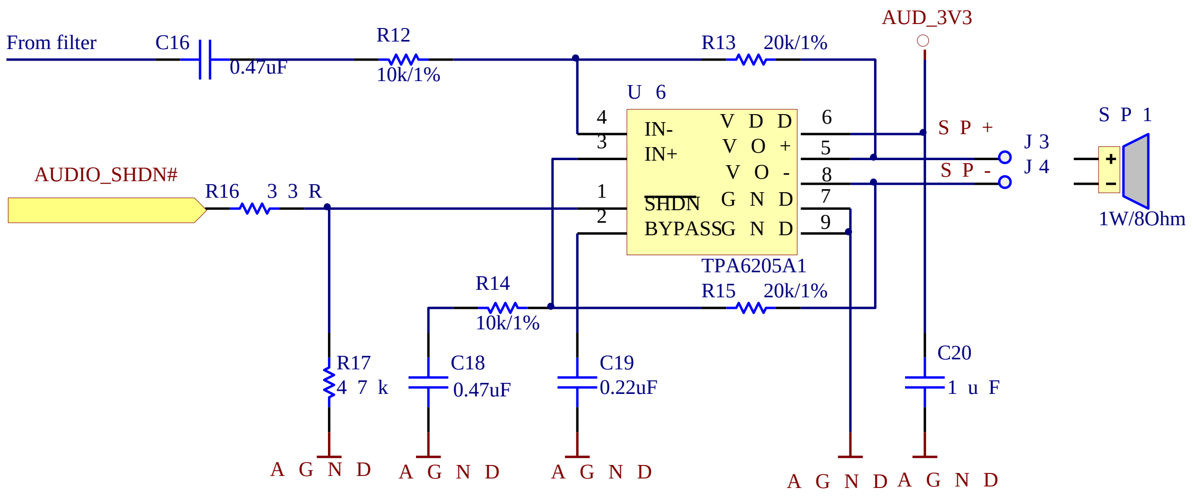

Figure 3.4 Audio Amplifier

3.5 Complete Circuit

The complete audio circuit shows all of the components together:

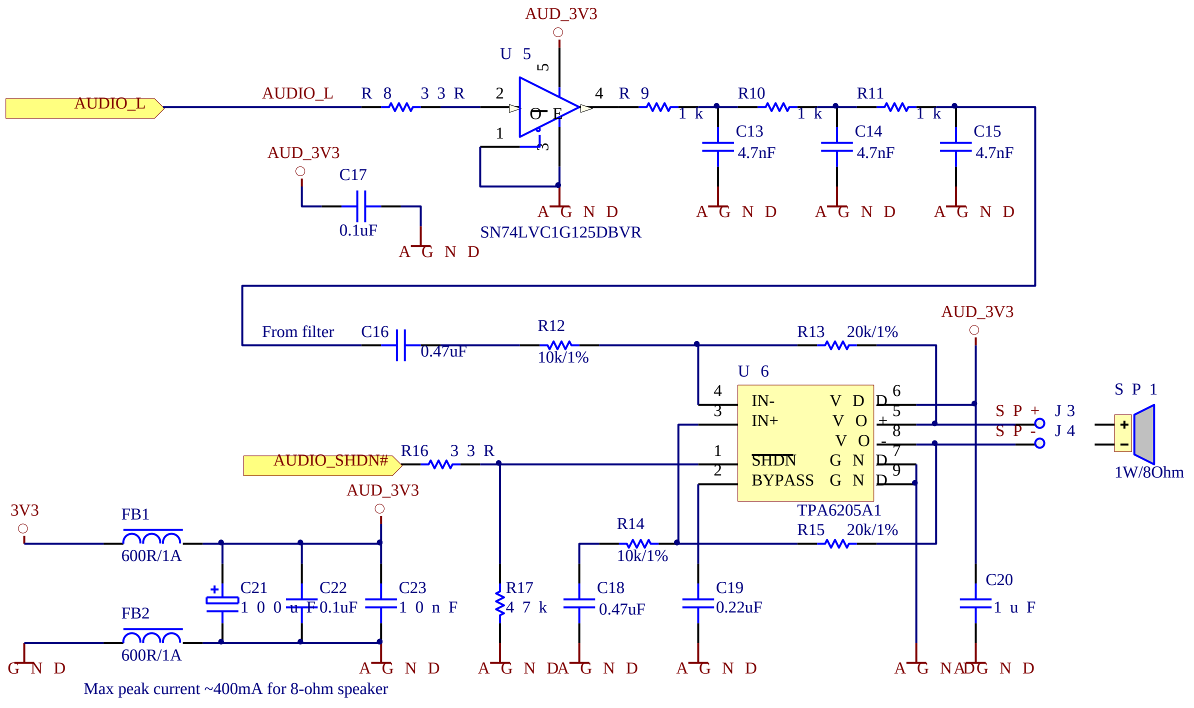

Figure 3.5 FT800 Audio Circuit

3.6 Layout considerations

Careful layout techniques are required:

- The audio circuit should be physically separated as far from the LCD signals as possible.
- Separate audio power and ground planes from the system power and ground.
- Use a single connection point between audio power and system power; likewise, use a single connection point between audio ground and system ground. In this example, the connection points are ferrite beads.

4 Programming

The audio features of the FT800 are accessed through the same Display List, along with the video and touch panel features. For example, every second a clock widget with a second hand could be drawn followed by a "tick" sound. Both the widget and sound reside in the same Display List. New Display Lists would then be sent to the FT800 every second, each containing both the widget with the new second hand position and another audio "tick".

For the examples below, "wr32(address, value)" indicates the MCU would write the value (VOLUME, frequency, etc.) to the address within the display list (RAM_DL + n) or register (e.g. REG_VOL_SOUND). This notation is used throughout other FT800 documents.

4.1 Initial Settings

As part of the FT800 initialization, set the volume to zero to eliminate any pops or clicks while all the initial settings are being sent:

- wr8(REG_VOL_SOUND, 0); // Set Synthesizer volume to zero
- wr8(REG_VOL_PB, 0); // Set Audio File Playback volume to zero
- wr16(REG_SOUND, 0x0062); // Play "mute" sound (optional)

4.2 Play Synthesized Effects

Only a few Display List elements are required in order to play a synthesized effect.

To play an effect:

1. wr8(REG_VOL_SOUND, VOLUME); // set the volume
2. wr16(REG_SOUND, (MIDI_NOTE << 8 + EFFECT)); // set the effect and pitch
3. wr8(REG_PLAY, 1); // start playing
4. rd8(REG_PLAY); // check whether done

An interrupt can also be generated for the host MCU when play has completed.

To stop playing a continuous effect:

1. wr16(REG_SOUND, 0); // set effect to "silence"
2. wr8(REG_PLAY, 1); // start playing silence
3. rd8(REG_PLAY); // check whether done

4.3 Load an Audio File into RAM_G

Audio and video data are treated equally in the RAM_G element memory. The command processor is sent the starting address and data to store, and then the MCU obtains the last used end pointer so another element may be copied in.

There are two methods of loading an audio file into the RAM_G element memory:

Uncompressed Data

Data from the raw audio file is written directly to RAM_G. The MCU, knowing the data_length, knows where to put the next element.

```
1. wr32(cmdBufferWr + n,        0xffffff1a);          // CMD_MEMWRITE
2. wr32(cmdBufferWr + n + 4,    RAM_G + offset);      // An 8-byte aligned location in
                                                      //   RAM_G.  If this is the first
                                                      //   element, then offset = 0
3. wr32(cmdBufferWr + n + 8,    data_length);         // length of audio data
                                                      //  (8-byte aligned / zero-pad)
4. wr8(cmdBufferWr + n + 12,    Byte 0);              // First byte of audio data
5. wr8(cmdBufferWr + n + 13,    Byte 1);              // Second byte of audio data
6. wr8(cmdBufferWr + n + …,     <Bytes 2 through data_length>
                                                      // … the rest of the data
```

7. MCU sets new RAM_G starting offset = offset + data_length for memory management

Compressed Data

Data from the compressed audio file is written to RAM_G through the INFLATE command. This decompresses "ZLIB" compressed data into RAM_G. Once the uncompressed data is written, the new end-pointer is obtained for the MCU to keep track of the RAM_G elements.

```
1. wr32(cmdBufferWr + n,        0xffffff22);      // CMD_INFLATE
2. wr32(cmdBufferWr + n + 4,    RAM_G + offset);  // An 8-byte aligned location in
                                                  //   RAM_G. If this is the first
                                                  //   element, then offset = 0
3. wr8(cmdBufferWr + n + 12,    Byte 0);          // First byte of compressed
                                                  //   audio data
4. wr8(cmdBufferWr + n + 13,    Byte 1);          // Second byte of compressed
                                                  //   audio data
5. wr8(cmdBufferWr + n + …,     <remainder of data>
                                                  // … the rest of the compressed
                                                  //   audio data
6. wr32(cmdBufferWr + n + <zlib_size>, 0xffffff22 // CMD_GETPTR
7. rd32(new RAM_G offset)                         // Assume utility already
                                                  //   padded data
8. MCU stores the new offset for RAM_G memory management
```

Note that if the audio conversion utility is used, both RAW and ZLIB files are created. When writing MCU firmware, the RAW file size can also be used as the end point of the uncompressed data.

4.4 Play an Audio File

Now that audio data is stored in RAM_G, it can be played through the FT800:

```
1. wr32(REG_PLAYBACK_START, RAM_G + offset);   // RAM_G element memory
                                               // offset is managed by host MCU
2. wr32(REG_PLAYBACK_LENGTH, length);          // file length, in bytes
3. wr16(REG_PLAYBACK_FREQ, frequency);         // playback sampling frequency
4. wr8(REG_PLAYBACK_FORMAT, format);           // file format
5. wr8(REG_VOL_PB, volume);                    // playback volume
6. wr8(REG_PLAYBACK_LOOP, play_once);          // play once or forever?
7. wr8(REG_PLAYBACK_PLAY, 1);                  // write a "1" to start playback
8. rd8(REG_PLAYBACK_PLAY)                      // playback complete (optional)
9. rd32(REG_PLAYBACK_READPTR)                  // how far along and how loud?
                                               //   (optional)
```

5 Conclusion

The FT800 provides an easy way to incorporate audio along with graphics displays and touch feedback into products that could not otherwise afford this capability.   A common programming interface provides developers with a single method to manipulate the FT800 regardless of the data types.

With only the FT800 between the MCU and the LCD display, a vivid graphics experience with touch and audio is now possible.

6 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

Appendix A – References

Document References

DS_FT800 FT800 Datasheet

PG_FT800 FT800 Programmer Guide

AN_240 FT800 From the Ground Up

External References

Audacity Open Source audio recording and editing software

Acronyms and Abbreviations

Terms	Description
IMA-ADPCM	Further compressed µLaw data containing two 4-bit samples per byte
MCU	Microcontroller (unit)
PCM	Pulse Coded Modulation
PWM	Pulse Width Modulation
RAW	Uncompressed data
µLaw	mu-law Companding Algorithm
ZLIB	Lossless compressed data algorithm

Appendix B – List of Tables & Figures

List of Tables

List of Figures

Appendix C – Revision History

Document Title:	AN_252 FT800 Audio Primer
Document Reference No.:	FT_000875
Clearance No.:	FTDI# 344
Product Page:	http://www.ftdichip.com/FTProducts.htm
Document Feedback:	Send Feedback

Revision	Changes	Date
1.0	Initial Release	2013-08-06

Revision History

Revision history (internal use only, please clearly state all changes here before saving the file)

Revision	Date YYYY-MM-DD	Changes	Editor
Draft	2013-07-23	Initial Outline	Bob Recny
Draft	2013-08-01	Copy ready for review – Section 3 images need re-drawn	Bob Recny
Draft	2013-08-02	Reviewed. Minor edits.	G Lunn
Draft	2013-08-02	Added external link to Audacity in References Added Table 2 caption Replaced section 3 images	Bob Recny
1.0	2013-08-06	Approved LCE/DS	G Lunn