Application Note

AN_281

FT800 Emulator Library User Guide

Version 1

Issue Date: 2014-Jul-21

This document describes the interface and usage of FT800 emulator library.

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://www.ftdichip.com

Table of Contents

1 Introduction

The FT800 Emulator is FT800 behavior modeling software targeting to PC. It is designed as a high level (behavior level) emulator other than a low level (clock accurate) emulator. It enables the user to evaluate FT800 features on a PC without hardware.

The FT800 emulator library is the distribution package of the FT800 emulator. This document describes the interface of the FT800 Emulator library and shows one example of how to integrate it into user's project.

The file may be downloaded at:
http://www.ftdichip.com/Support/SoftwareExamples/EVE/setup%20ft800%20emulator.zip

1.1 Overview

The FT800 emulator has the exact SPI/I^{2}C interface configuration and consistent memory map with the FT800 silicon. As such, the users application does not need to write new interface layer for emulator version. In terms of behavior, the FT800 emulator has the maximum similarity, although there are a few limitations, which are mentioned at 1.4.

For touch functionality, the FT800 emulator requires the mouse of the PC to simulate single touch input.

For visual effects, the FT800 emulator employs the OS specific graphics driver to display the output on the PC monitor.

The emulator supports the full set of display list commands and most of the coprocessor commands.

1.2 Scope

This document covers FT800 emulator library interface and introduces how to use it in an application by an example. However, this document does not include the implementation details about FT800 emulator. The FT800 relative information, such as registers , memory map, commands, etc. is also not covered.

1.3 Requirement

Currently, FT800 emulator library is built by Microsoft Visual Studio C++ Express version MSVC 2012. So it may require the same tool chain to link it with users' project successfully.

In addition, the runtime environment "**vcomp110.dll**" and "**msvcr110.dll**" are required to be on Windows to run the emulator project successfully.

1.4 Limitation

The FT800 emulator does NOT support the following functionality:

1. Power management (Host commands)
2. Screenshot (coprocessor command "cmd_snapshot" has no effect)
3. Coprocessor engine reset
4. Interrupt
5. Registers that reflect hardware properties, e.g., the pressure value of touch and ADC related touch registers

2 FT800 Emulator Library Introduction

2.1 FT800 Emulator Library Interface

The interface of FT800 Emulator library is written in C++ and resides in the "FT800EMU::" name space only. Within the "FT800EMU::" name space, there are two modules "SPII2C" and "Emulator" exposing interface. Here is the structure:

Name Space	Module Name	API Name	Parameter	Return	Description
FT800EMU::	SPII2C	begin	None	None	Initialize the SPI I^2C module
FT800EMU::	SPII2C	end	None	None	De-Initialize the SPI / I^2C Module
FT800EMU::	SPII2C	csLow	None	None	Set the Chip Select pin low to start one SPI transfer
FT800EMU::	SPII2C	csHigh	None	None	Set the Chip Select pin low to stop one SPI transfer
FT800EMU::	SPII2C	transfer	One byte	One byte	Make one byte SPI or I^2C transaction on SPI or I^2C bus.
FT800EMU::	Emulator	run	See 2.1.6	None	Start the FT800 emulator. Never return unless the emulator exists.

Table 1 – FT800 Emulator library interface structure

2.1.1 FT800EMU::SPII2C.begin()

- **Prototype**

  void begin();

- **Description**

  Initialize the SPI/I^{2}C module of the FT800 Emulator

- **Return value**

  None

- **Parameter**

  None

2.1.2 FT800EMU::SPII2C.end()

- **Prototype**

  void end();

- **Description**

  De-Initialize the SPI/I^{2}C module of the FT800 Emulator

- **Return value**

  None

- **Parameter**

  None

2.1.3 FT800EMU::SPII2C.csLow ()

- **Prototype**

  void csLow ();

- **Description**

  Call this API to start one SPI/I^{2}C transfer. It is equivalent to pulling down chip select pin on the SPI/I^{2}C bus in the FT800 hardware. For I^{2}C bus, this function is equivalent to start a message with a START.

- **Return value**

  None

- **Parameter**

  None

2.1.4 FT800EMU::SPII2C.csHigh()

- **Prototype**

  void csHigh();

- **Description**

  Call this API to end one SPI/I^{2}C transfer. For SPI bus, it is equivalent to pulling chip select pin high on the SPI/ I^{2}C bus in FT800 hardware. For I^{2}C bus, this function is equivalent to end a message with a STOP.

- **Return value**

  None

- **Parameter**

  None

2.1.5 FT800EMU::SPII2C.transfer()

- **Prototype**

  uint8_t transfer(uint8_t data);

- **Description**

  Calling this API is to transfer one bytes from/to emulator.

  The data to be sent is specified as a parameter, while the data to be received is given as a return value.

- **Return value**

  One byte of data received from the FT800 emulator if it is read transfer.

- **Parameter**

  One byte of data sent to FT800 emulator. In case of an SPI read transfer, this byte can be anything.

2.1.6 FT800EMU::Emulator.run ()

- **Prototype**

  void run(const EmulatorParameters &params);

- **Description**

  Calling this function will start the emulator immediately and the application control is transferred to the emulator. The emulator's behavior is configured through the parameters which were passed in. The application's code will be called through two callback functions in the parameter structure. This API shall never return, unless emulator is killed or the application process exists.

- **Return value**

  None

- **Parameters**

  Please check the following code for details about parameter definition.

1) Definition of parameter structure

```
struct EmulatorParameters
{
public:
	EmulatorParameters() :
		Setup(0),
		Loop(0),
		Flags(0),
		Keyboard(0),
		MousePressure(0),
		ExternalFrequency(0),
		ReduceGraphicsThreads(0),
		Graphics(0)
	{ }

	// Microcontroller function called before loop.
	void (*Setup)();
	// Microcontroller continuous loop.
	void (*Loop)();
	// See EmulatorFlags.
	int Flags;

	// Called after keyboard update.
	// Supplied function can use Keyboard.isKeyDown(FT800EMU_KEY_F3).
	void (*Keyboard)();
	// The default mouse pressure, default 0 (maximum).
	// See REG_TOUCH_RZTRESH, etc.
	uint32_t MousePressure;
	// External frequency. See CLK, etc.
	uint32_t ExternalFrequency;

	// Reduce graphics processor threads by specified number, default 0
	// Necessary when doing very heavy work on the MCU or Coprocessor
	// TODO: Maybe possible to automate this based on thread info
	uint32_t ReduceGraphicsThreads;

	// Replaces the default builtin ROM with a custom ROM from a file.
	std::string RomFilePath;
	// Replaces the builtin coprocessor ROM.
	std::string CoprocessorRomFilePath;

	// Graphics driverless mode
	// Setting this callback means no window will be created, and all
	// rendered graphics will be automatically sent to this function.
	// The output parameter is false when the display is turned off.
	// The contents of the buffer pointer are undefined after this
	// function returns.
	// Return false when the application must exit.
	bool (*Graphics)(bool output, const argb8888 *buffer, \
		uint32_t hsize, uint32_t vsize);
};
```

Figure 1 – Definition of structure "EmulatorParameters"

2) Flags to configure the emulator

The enumerate code sample shown below defines the emulator feature to be run with. To enable specific features, you can "OR" these enumerate and assign the result values to "Flags" field in the parameter structure "EmulatorParameters" above.

```
enum EmulatorFlags
{
        // enables the keyboard to be used as input
        EmulatorEnableKeyboard = 0x01,
        // enables audio
        EmulatorEnableAudio = 0x02,
        // enables coprocessor
        EmulatorEnableCoprocessor = 0x04,
        // enables mouse as touch
        EmulatorEnableMouse = 0x08,
        // enable debug shortkeys
        EmulatorEnableDebugShortkeys = 0x10,
        // enable graphics processor multithreading
        EmulatorEnableGraphicsMultithread = 0x20,
        // enable dynamic graphics quality degrading by interlacing
        EmulatorEnableDynamicDegrade = 0x40,
        // enable usage of REG_ROTATE
        EmulatorEnableRegRotate = 0x80,
        // enable emulating REG_PWM_DUTY by fading the rendered display to black
        EmulatorEnableRegPwmDutyEmulation = 0x100,
};
```

Figure 2 – Flags field definition

3) Typical setting

For optimal performance, the settings below are recommended.

The callback functions "setup()" and "loop()" **shall be defined** by the user project and they will be called by the emulator. Function "setup()" is assumed to run once by the emulator for initialization purposes. Function "loop()" will be called periodically by the emulator. These two functions ensure the user project is in the context of the FT800 emulator. The failure of assigning "setup()" and "loop()" to the emulator will result in no input to the FT800 emulator.

Usually, the function "setup()" and "loop()" in users' project defines the main logic and the display list will be sent to FT800 emulator through SPI/I^2C interface.

```
#include <ft800emu_emulator.h>
#include <ft800emu_spi_i2c.h>

/*Application defined function*/
extern "C" void setup();
extern "C" void loop();

ft_int32_t main(ft_int32_t argc,ft_char8_t *argv[])
{
        FT800EMU::EmulatorParameters params;
        params.Setup = setup;
        params.Loop = loop;
        params.Flags =
                   FT800EMU::EmulatorEnableKeyboard
                 | FT800EMU::EmulatorEnableMouse
                 | FT800EMU::EmulatorEnableRegRotate
                 | FT800EMU::EmulatorEnableCoprocessor
                 | FT800EMU::EmulatorEnableGraphicsMultithread;

        FT800EMU::Emulator.run(params);
}
```

Figure 3 – Start FT800 emulator

3 Using the FT800 Emulator library

This chapter will provide an example on how to use the FT800 emulator in the FT800 sample application. Users are encouraged to familiarize themselves with the FT800 sample application before starting this chapter.

The FT800 emulator interface is defined in the following file:

"ft800emu_inttypes.h": the definition for integer type for different platforms.

"ft800emu_spi_i2c.h": the SPI/ I²C interface declaration

"ft800emu_emulator.h": the interface to start the emulator

3.1 Start the FT800 emulator

To make use of the FT800 emulator, the users' project is required to call the API "FT800EMU::Emulator.run" with the specific parameter. The emulator library will be started properly and ready to be accessed through SPI/ I²C interface.

Please see Figure 3 – Start FT800 emulator.

3.2 Working with the SPI/I²C interface.

The SPI/I²C interface is the control interface of FT800. FT800 emulator library provides the APIs to simulate the same interface. Since FT800 sample application is built in C language, instead of C++, one simple C API wrapper is introduced as below to ease the calling convention.

```
#ifdef __cplusplus
extern "C" {
#endif

#ifdef MSVC_FT800EMU
#define BUFFER_OPTIMIZATION
#endif

void Ft_GpuEmu_SPII2C_begin();
void Ft_GpuEmu_SPII2C_csLow();
void Ft_GpuEmu_SPII2C_csHigh();
void Ft_GpuEmu_SPII2C_end();

void     Ft_GpuEmu_SPII2C_StartRead(uint32_t addr);
uint8_t  Ft_GpuEmu_SPII2C_transfer(uint8_t data);
void     Ft_GpuEmu_SPII2C_StartWrite(uint32_t addr);

#ifdef __cplusplus
}
#endif
```

Figure 4 – The C interface API

The implementation is as below:

```
void Ft_GpuEmu_SPII2C_begin()
{
        FT800EMU::SPII2C.begin();
}

void Ft_GpuEmu_SPII2C_csLow()
{
        FT800EMU::SPII2C.csLow();
}

void Ft_GpuEmu_SPII2C_csHigh()
{
        FT800EMU::SPII2C.csHigh();
}

void Ft_GpuEmu_SPII2C_end()
{
        FT800EMU::SPII2C.end();
}

uint8_t Ft_GpuEmu_SPII2C_transfer(uint8_t data)
{
        return FT800EMU::SPII2C.transfer(data);
}


void  Ft_GpuEmu_SPII2C_StartRead(uint32_t addr)
{
        Ft_GpuEmu_SPII2C_csLow();
        Ft_GpuEmu_SPII2C_transfer((addr >> 16) & 0xFF);
        Ft_GpuEmu_SPII2C_transfer((addr >> 8) & 0xFF);
        Ft_GpuEmu_SPII2C_transfer(addr & 0xFF);

        Ft_GpuEmu_SPII2C_transfer(0); //Dummy Read Byte
}

void  Ft_GpuEmu_SPII2C_StartWrite(uint32_t addr)
{
        Ft_GpuEmu_SPII2C_csLow();
        Ft_GpuEmu_SPII2C_transfer(((addr >> 16) & 0xFF) | 0x80);
        Ft_GpuEmu_SPII2C_transfer((addr >> 8) & 0xFF);
        Ft_GpuEmu_SPII2C_transfer(addr & 0xFF);
}
```

Figure 5 – The implementation of C API

3.3 Sample Application adaptation

The FT800 sample application employs a Hardware Abstraction Layer(HAL) to make the application logic independent from the hardware platform. It is defined in "FT_Gpu_Hal.c" and users are assumed to be familiar with it before moving ahead.

To adapt the sample application on the FT800 emulator, the minimum changes are required: just implement the APIs defined in "FT_Gpu_Hal.c" as below:

```
/* API to initialize the SPI interface */
ft_bool_t  Ft_Gpu_Hal_Init(Ft_Gpu_HalInit_t *halinit)
{
        return TRUE;
}
ft_bool_t    Ft_Gpu_Hal_Open(Ft_Gpu_Hal_Context_t *host)
{
        Ft_GpuEmu_SPII2C_begin();
        host->ft_cmd_fifo_wp = host->ft_dl_buff_wp = 0;
        host->status = FT_GPU_HAL_OPENED;
        return TRUE;
}
ft_void_t  Ft_Gpu_Hal_Close(Ft_Gpu_Hal_Context_t *host)
{
        host->status = FT_GPU_HAL_CLOSED;
        Ft_GpuEmu_SPII2C_end();
}
ft_void_t Ft_Gpu_Hal_DeInit()
{
}
/*The APIs for reading/writing transfer continuously only with small buffer system*/
ft_void_t  Ft_Gpu_Hal_StartTransfer(Ft_Gpu_Hal_Context_t *host,FT_GPU_TRANSFERDIR_T
rw,ft_uint32_t addr)
{
        if (FT_GPU_READ == rw){
                Ft_GpuEmu_SPII2C_StartRead(addr);
                host->status = FT_GPU_HAL_READING;
        }else{
                Ft_GpuEmu_SPII2C_StartWrite(addr);
                host->status = FT_GPU_HAL_WRITING;
        }
}
ft_uint8_t    Ft_Gpu_Hal_Transfer8(Ft_Gpu_Hal_Context_t *host,ft_uint8_t value)
{
        return Ft_GpuEmu_SPII2C_transfer(value);
}
ft_void_t   Ft_Gpu_Hal_EndTransfer(Ft_Gpu_Hal_Context_t *host)
{
        Ft_GpuEmu_SPII2C_csHigh();
        host->status = FT_GPU_HAL_OPENED;
}
ft_void_t Ft_Gpu_HostCommand(Ft_Gpu_Hal_Context_t *host,ft_uint8_t cmd)
{
  //Not implemented in FT800EMU. No Power Management feature in Emulator.
}

ft_void_t Ft_Gpu_Hal_WrMem(Ft_Gpu_Hal_Context_t *host,ft_uint32_t addr,const
ft_uint8_t *buffer, ft_uint32_t length)
{
        ft_uint32_t SizeTransfered = 0;

        Ft_Gpu_Hal_StartTransfer(host,FT_GPU_WRITE,addr);

        while (length--) {
             Ft_Gpu_Hal_Transfer8(host,*buffer);
            buffer++;
        }
        Ft_Gpu_Hal_EndTransfer(host);
}

ft_void_t Ft_Gpu_Hal_RdMem(Ft_Gpu_Hal_Context_t *host,ft_uint32_t addr, ft_uint8_t
*buffer, ft_uint32_t length)
{
        ft_uint32_t SizeTransfered = 0;

        Ft_Gpu_Hal_StartTransfer(host,FT_GPU_READ,addr);

        while (length--) {
           *buffer = Ft_Gpu_Hal_Transfer8(host,0);
           buffer++;
        }

        Ft_Gpu_Hal_EndTransfer(host);
}
```

Figure 6 – Hardware Abstraction Layer implementation in emulator API

3.4 Build and Run

After porting the application to the FT800 emulator according to the instructions above, in order to build the final executable, user project is required to specify the path and name of the FT800 emulator library.

For release build, please specify the FT800 emulator library named "FT800Emu.lib".

For debug build, please specify the FT800 emulator library named "FT800Emud.lib".

Please note that Microsoft Visual Studio 2012 Express version is a must to link with the emulator library and build your application.

The picture below shows a screenshot of when the FT800 logo application is running on top of FT800 emulator.

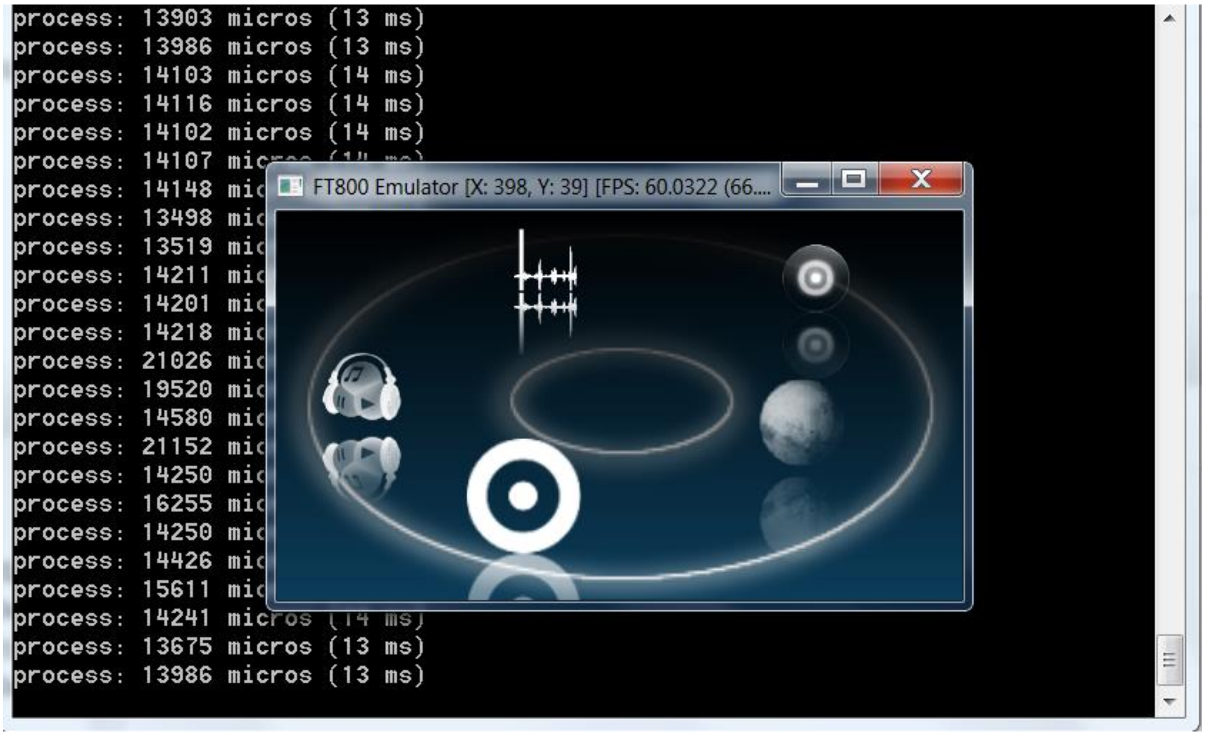

Figure 7 – Logo application running on top of FT800 emulator

4 Contact Information

Head Office – Glasgow, UK

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

7130 SW Fir Loop
Tigard, OR 97223
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

2F, No. 516, Sec. 1, NeiHu Road
Taipei 114
Taiwan, R.O.C.
Tel: +886 (0) 2 8797 1330
Fax: +886 (0) 2 8751 9737

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

Room 1103, No. 666 West Huaihai Road,
Changning District
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

- FT800 data sheet
- FT800 programmer guide
- AN_240_FT800_From_the_Ground_Up
- FT800 Sample Application
- FT800 Emulator

Acronyms and Abbreviations

Terms	Description
API	Application Programming Interface
I^2C	Inter-Integrated Circuit
PC	Personal Computer
SPI	Serial Peripheral Interface
USB	Universal Serial Bus
USB-IF	USB Implementers Forum
Windows	Microsoft Windows Desktop operating system

Appendix B – List of Tables & Figures

List of Tables

List of Figures

Appendix C – Revision History

Document Title: AN_281 FT800 Emulator Library User Guide

Document Reference No.: FT_000944

Clearance No.: FTDI#401

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

Document Feedback: Send Feedback

Revision	Changes	Date
1.0	Initial Release	2014-07-21