Application Note

AN_420

FT_App_Keyboard

Version 1.0

Issue Date: 2016-11-08

This document describes the operation of the Keyboard Demo Application running on an FT81X. The Keyboard example demonstrates how fast and easy it is to use FT81X commands to construct a fully functional keypad GUI.

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

This document describes the operation of the Keyboard Demo Application running on an FT81X. The example demonstrates how fast and easily a fully functional keypad GUI can be constructed using FT81X commands. Please also refer to the sample code project provided on Bridgetek website which demonstrates all the features of FT81x.

http://brtchip.com/eve/

1.1 Overview

This application demonstrates the coprocessor command and touch tracking features of the FT81X.

The example demonstrates that in addition to providing an attractive graphical user interface for an application, the FT81X's tagging features can be used to construct effective, fully interactive GUI operations rapidly.

1.2 Scope

This document can be used by designers to develop GUI applications on a hardware setup that use an FT81X interfaced to a SPI host. The SPI host can be a PC running Visual Studio (C++) with a C232HM cable or an FT900.

The document covers the following topics:

- Brief overview of the Keypad demonstration
- Flow of the program including the FT81X initialisation and keypad code
- Description of the keypad function within the application
- Running the demonstration program

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

1. FT810 datasheet
2. Programming Guide covering EVE command language
3. AN_240 FT800 From the Ground Up
4. AN_245 VM800CB_SampleApp_PC_Introduction
    - Covering detailed design flow with a PC and USB to SPI bridge cable
5. AN_246 VM800CB_SampleApp_Arduino_Introduction
    - Covering detailed design flow in an Arduino platform
6. AN_252 FT800 Audio Primer

2 Keyboard Overview

This example demonstrates keyboard development capabilities of the FT81X.

The application supports a multiline editor area with an alphanumeric keyboard. The layout with numbers also provides some special character buttons. The button labelled "a^" acts like a CAPS LOCK key and switches key entry to capital (block) letters. The button labelled "12*" switches the keyboard layout to numbers and some special characters. The "<-"backspace key, "Clear" key and "Space" spacebar key are all available in both CAPS LOCK and numeric entry mode

Below figures 2.1 and 2.2 show the keyboard layout in alphabets and numbers.

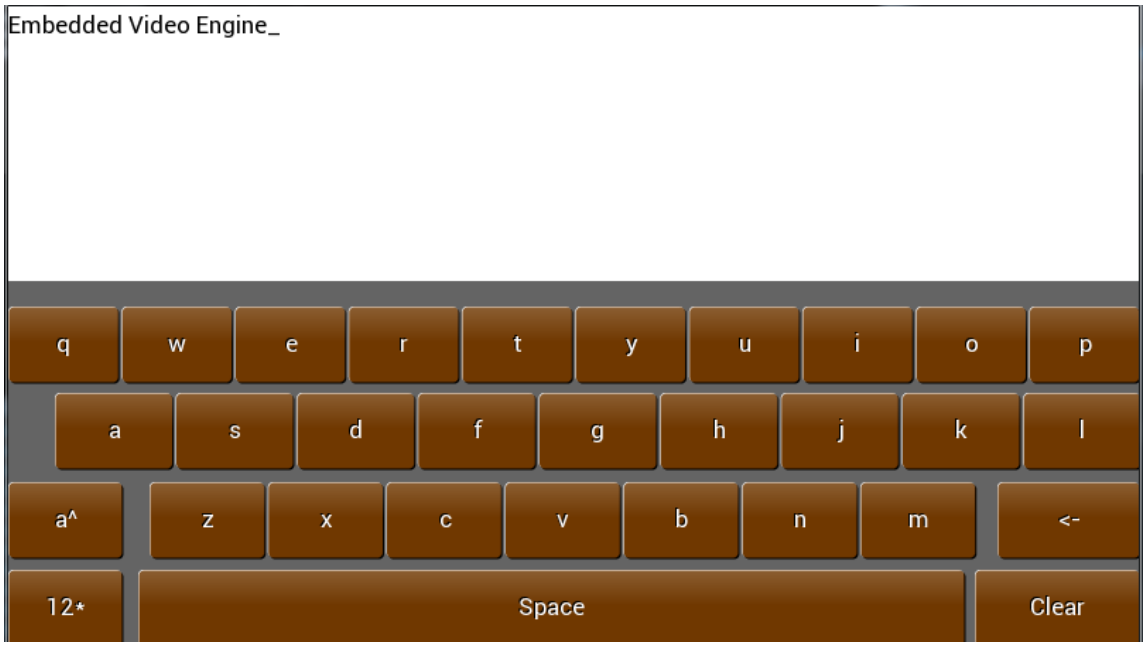

Figure 2-1 The Keyboard example application

Figure 2-2 Keyboard example number layout

This application runs on a host processor which triggers commands for the FT81x over SPI communication. The host processor may be a PC (through an MSVC project application) which uses USB-to-SPI bridge chip from FTDI Chip (e.g. FT232H / FT4222H) or an FT900 which establishes a connection to FT81x over SPI.

The application uses FT81x commands, namely, CMD_KEYS and CMD_BUTTON together with touch controller registers to create this keyboard.

3 Design Flow

3.1 Initialisation

Every EVE design follows the same basic principles as highlighted in Figure 3.1. After configuring the SPI Host itself (such as the PC through the CM232H cable, or an MCU), the application will wake up the FT81X and write to the registers in the FT81X to configure its display, touch and audio settings etc. It then writes an initial display list to clear the screen.

The main application can then create display lists to draw the actual application screens, in this case, the keyboard screen. In essence, there will be two lists; the active list and the edited list which are continually swapped to update the display. Each screen can be created by either writing a display list to the RAM_DL memory in the FT81X or by writing a series of commands to the Co-Processor FIFO in the FT81X (in which case, the Co-Processor will create a display list in RAM_DL based on the commands).

Header files map the pseudo code of the design file of the display list to the FT81X instruction set, which is sent as the data of the SPI packet (typically <1KB). As a result, with EVE's object oriented approach, the FT81X is operating as an SPI peripheral while providing full display, audio, and touch capabilities.

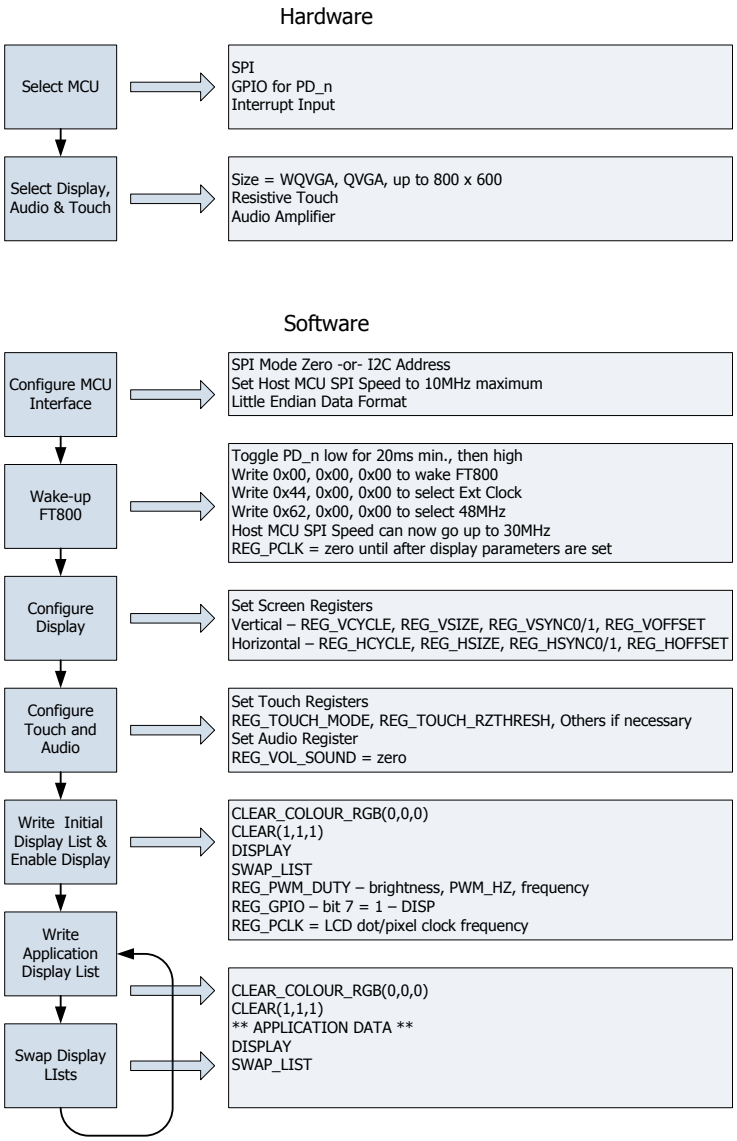

Figure 3-1 Generic EVE Design Flow

3.2 Application Flow

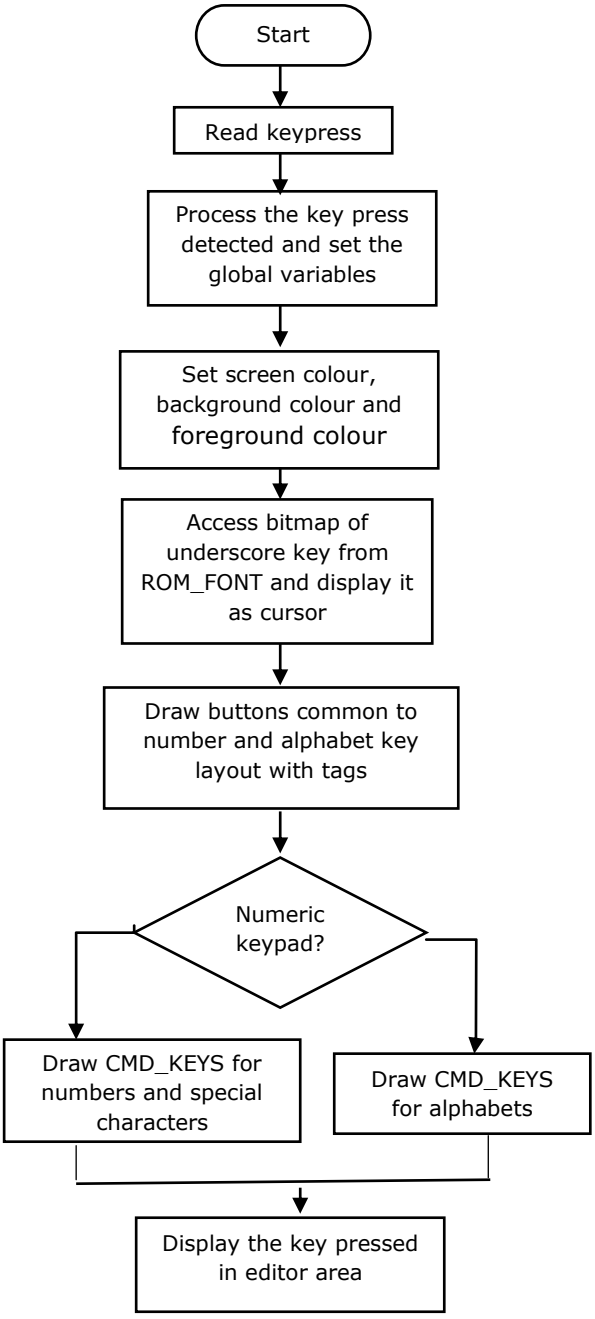

Figure 3-2 Application Flowchart

4 Description

This section describes the application code used to generate the various display elements.

4.1 System Initialisation

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 application uses the same initialisation process as the other application notes in this series.

4.2 Bootup Config

The function labelled Ft_BootupConfig in this project 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 configured by setting the clock register and PLL to 48MHz. Note that 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 has its own GPIO lines which can be controlled by writing to registers. One of these is connected to the display's enable line and so a write to the FT800 GPIO allows the display to

```
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));
```

enable.

To confirm that the FT800 is awake and ready to start accepting display list information, the identity register is read continuously until it reports 0x7C. Reading 0x7C indicates FT81X initialized successfully and ready to process commands.

```
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 settings may be configured by writing 13 of the registers inside the FT800 to match the display being used. 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);   // display visible on the LCD
Ft_Gpu_Hal_Wr16(phost, REG_HSIZE, FT_DispWidth);
Ft_Gpu_Hal_Wr16(phost, REG_VSIZE, FT_DispHeight);
```

If the underlying EVE chip is an FT810 or FT812 which is interfaced to a resistive display panel then the above configuration list can be extended to set up the touch screen resistance threshold.

```
/* 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);
```

An optional step is present back in the `main` program to clear the screen so that no artefacts from boot-up 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);
```

4.3 Info()

This function provides the initial screens of the application.

A Co-Processor command list is started. The command will clear the display parameters.

```
Ft_Gpu_CoCmd_Dlstart(phost);
Ft App WrCoCmd Buffer(phost,CLEAR(1,1,1));
```

The following commands set the colour and then print a text message to the user which tells them to tap on the dots during the following calibration routine. The FT800's built-in calibration routine is then called.

```
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
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);
```

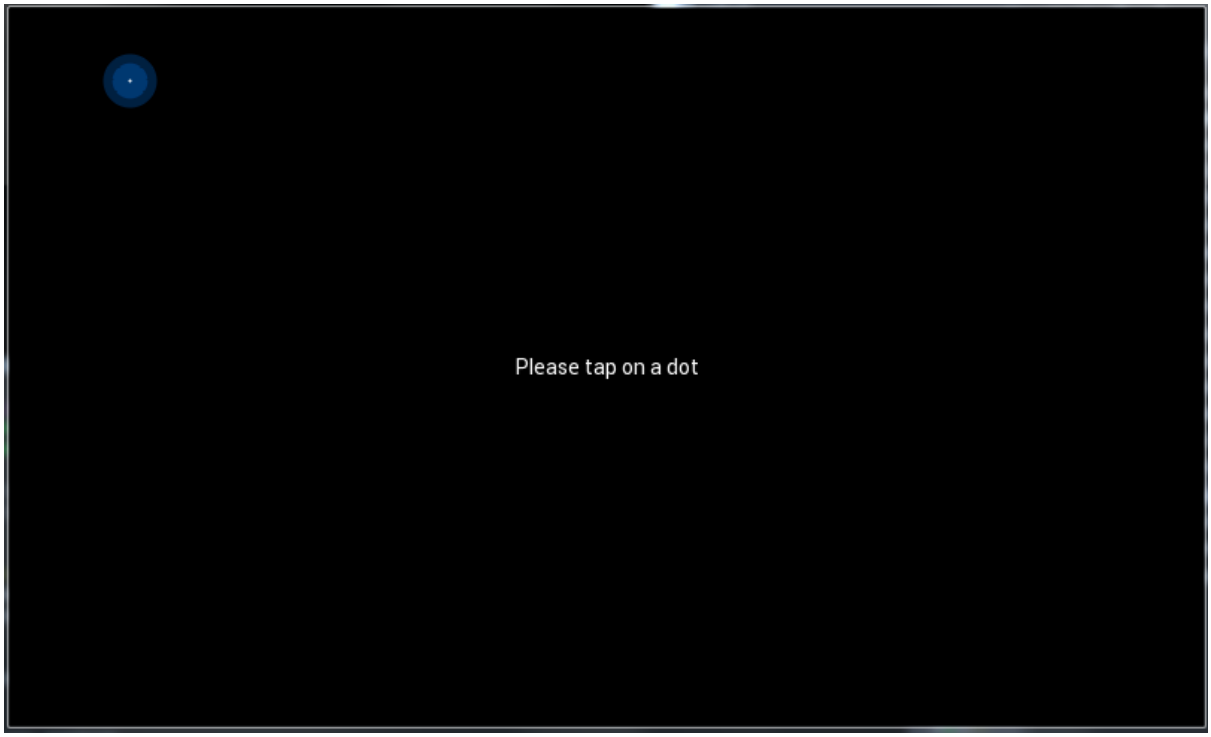

Figure 4-1 Calibration screen

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();
```

Figure 4-2 Logo screen

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

Once the 'Click to play' button has been tapped, the keypad application will be loaded.

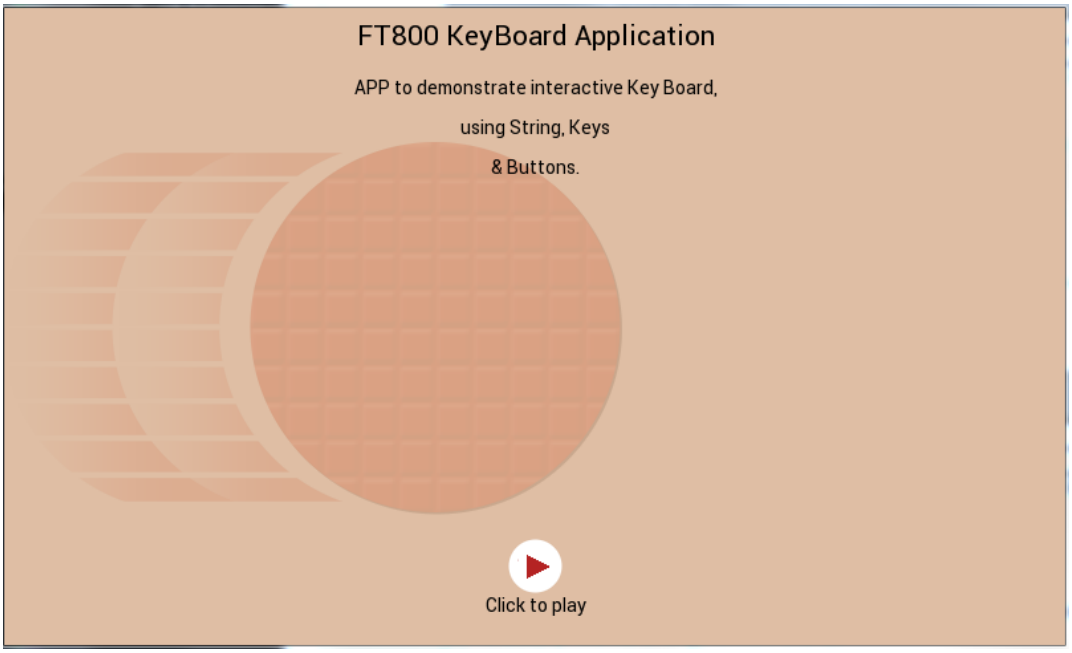

Figure 4-3 Start screen

4.4 Keypad Function

This is the main function in which the application will now remain. After initialization, the code will run in a continuous loop where it will perform the following functions.

- Check the Tag registers and determine the keypress
- Draw the background, and clear an area with the scissor functions
- Draw those buttons common for both number and alphabet keypad layout with labels and tags
- Draw the keypad layout by determining whether number keypad or alphabet with labels and set tags
- Display the key pressed

Following are the crucial global variables and macros which are controlling the keypad operation.

1. `ft_uint8_t Read_Keypad()` is the function to read a keypress. Debounce logic is also included in it. The registers `REG_TOUCH_TAG` and `REG_TOUCH_RAW_XY` are used to detect the key presses.

2. `void Notepad(void)` is the function which holds the entire keyboard functioning logic.

3. The multiline editor area content is held in a global variable called Buffer which is a structure type as shown below.

```
struct Notepad_buffer
{
        ft_char8_t *temp;
        ft_char8_t  notepad[MAX_LINES][80];
}Buffer;
```

4. The following variable Flag maintains the information about the keypad layout to be drawn.

```
struct
{
        ft_uint8_t Key_Detect :1;
        ft_uint8_t Caps :1;
        ft_uint8_t Numeric : 1;
        ft_uint8_t Exit : 1;
}Flag;
```

5. To display the next character input position on the editor area, the current character width is read from the font metric block and added to current display position thereby computing the next character input position.

```
Disp_pos += Ft_Gpu_Rom_Font_WH(Buffer.notepad[Line][0],Font); // Update
the Disp_Pos
noofchars+=1;
```

6. The detected key press is processed in an if/else block based on whether the key pressed is a special function or normal character/number key press. The following special functions are defined for this purpose.

```
#define SPECIAL_FUN     251
#define BACK_SPACE   251                    // Back space
#define CAPS_LOCK    252                    // Caps Lock
#define NUMBER_LOCK  253                    // Number Lock
#define BACK         254                    // Exit
```

7. The TAG display list command along with CMD_BUTTON and CMD_KEYS co-processor commands are used to draw the complete keyboard. The pressed keys are displayed in editor area using CMD_TEXT co-processor command.

5 Running the demonstration code

This package has project solutions to build and execute the keypad application using the following IDEs.

- <...>\FT_App_Keyboard\Project\Msvc_win32\ has a Visual Studio (C++) project in which a PC is used as an SPI host to send FT81X commands. This setup requires a USB-to-SPI bridge chip to establish communication from PC to FT81X. The development modules like ME812AU and ME813AU have this bridge chip mounted on the PCB.

- <...>\FT_App_Keyboard\Project\FT90x has an Eclipse project in which an FT900 is used as the SPI host. The FT90x Toolchain can be downloaded from Bridgetek website.

- <...>\FT_App_Keyboard\Project\MSVC_Emulator has a Visual Studio (C++) project for an Emulated version of the FT81x. This project doesn't need any hardware to launch the application.

The user of this application needs to open the project in one of the IDE listed above, followed by enabling the platform macros in the FT_Platform.h header file. Depending on the underlying hardware, one of the following platform configurations must be chosen.

```
//#define VM800B43_50                    (1)
//#define VM800B35                       (1)
//#define VM801B43_50                    (1)
//#define VM810C50                       (1)
#define ME812AU_WH50R                    (1)
//#define ME813AU_WH50C                  (1)
//#define ME810AU_WH70R                  (1)
//#define ME811AU_WH70C                  (1)
```

Now the project should be able to build and run.

6 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

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_420 FT_App_Keyboard
Document Reference No.: BRT_000054
Clearance No.: BRT#048
Product Page: http://brtchip.com/i-ft8/
Document Feedback: Send Feedback

Revision	Changes	Date
1.0	Initial release	2016-11-08