Application Note

AN_264

FT_App_Gradient

Version 1.2

Issue Date: 2018-01-05

This application note describes the operation of the Gradient Demo Application running on the FT9XX MCU, Microsoft Visual C, Arduino and EVE Screen Editor platforms. The Gradient example demonstrates the way in which the tracking features of the FT8XX can be used to manipulate graphics on the screen and produce visual effects. It displays a gradient object which the user can then rotate and resize by dragging their finger on the screen. Two sliders at the side of the screen can be used to change the colours at each end of the gradient.

Bridgetek Pte Ltd (BRTChip)
178 Paya Lebar Road, #07-03, Singapore 409030
Tel: +65 6547 4827 Fax: +65 6841 6071
Web Site: http://www.brtchip.com

Table of Contents

1 Introduction

This application note describes the operation of the Gradient Demo Application, which demonstrates the way in which the tracking features can be used to manipulate graphics on the screen and produce visual effects.

Figure 1.1 The Gradient Example

Further sample code for the EVE family can be found on the EVE examples page:
http://brtchip.com/SoftwareExamples-eve/

1.1 Overview

This application demonstrates the gradient feature and touch tracking/tagging features of the FT8XX. It displays a colour gradient on the screen which the user can then rotate and resize by dragging their finger on the screen. Two sliders at the side of the screen can be used to change the colours at each end of the gradient.

This example demonstrates that in addition to providing an attractive graphical user interface for an application, the FT8XX's tracking and tagging features can be used to allow this to be manipulated interactively by the user.

1.2 Scope

This document can be used by designers to develop GUI applications using the FT8XX with an SPI host. It covers the following topics:

- Brief overview of the Gradient demonstration
- Flow of the code project including the FT8XX initialisation and gradient code
- More detailed description of the code itself
- Loading and running the demonstration code

Please refer to the sample code package provided with this application note at the following link:
http://brtchip.com/SoftwareExamples-eve/

Note: This document is intended to be used along with the source code project provided. This can be found at the above link.

2 Gradient Overview

The FT8XX includes a Gradient command which uses the FT8XX's internal graphics engine to generate a colour gradient between two specified coordinates. The command specifies the starting and ending coordinates and the colours at the starting and ending coordinates, and the FT8XX will calculate a smooth colour transition between these coordinates.

In this example, a gradient is drawn between two points on the screen. Users can then change the gradient in two different ways:

- Users can select the colours at each end of the gradient. The colour values are taken from two sliders which are positioned at the right-hand side of the screen. The application assigns tags to these sliders so that users can adjust the values using the touch screen. The value of the slider is used to determine the colour of the associated end of the gradient.

- A cross symbol is also drawn (using the Points and Lines primitives) to indicate the location of the two points between which the gradient is drawn. A point drawn under each cross is tagged so that the FT8XX can identify touches in these areas. Users can drag these points to change the starting and finishing position of the gradient bar.

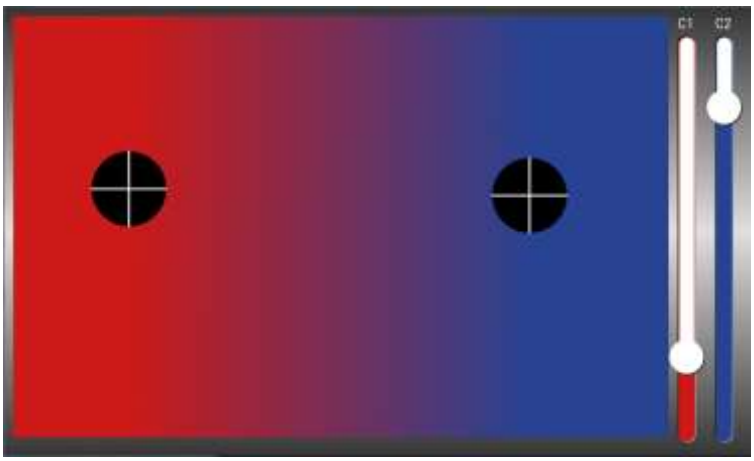

Figure 2.1 The Gradient Example Application

As described in the forthcoming chapters, the SPI Master (which could be a PC running the sample application with a USB to SPI cable, or the Arduino or FT9XX MCUs) will be in a loop; constantly reading the information from the touch screen/tag registers, calculating the new end-colours and position of the gradient bar, and generating a co-processor command list to display the updated gradient.

3 Design Flow

3.1 Initialisation

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

The supplied sample code includes the full flow but this document focuses on the main application in the final step. The earlier steps are generic to the EVE examples and are covered in the application note AN_391 EVE Platform Guide

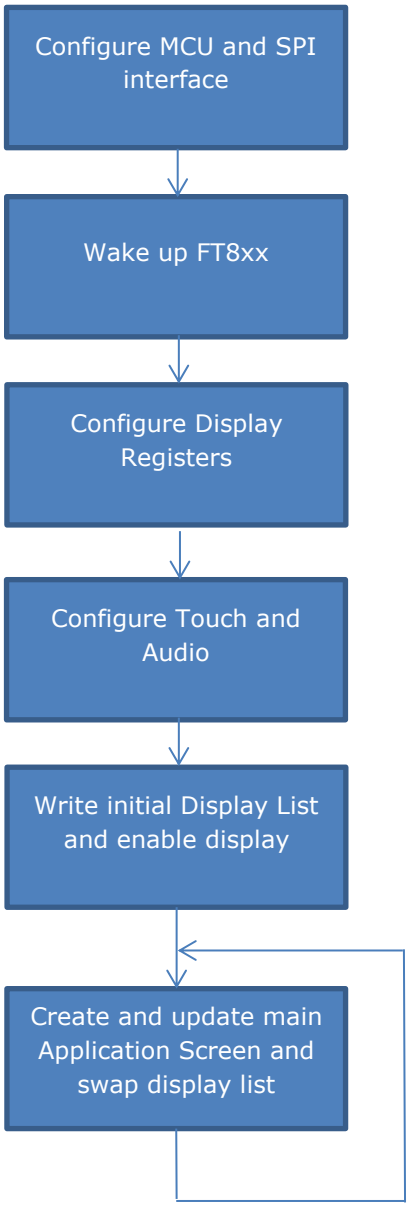

Figure 3.1 Generic EVE Design Flow

3.2 Application Flow

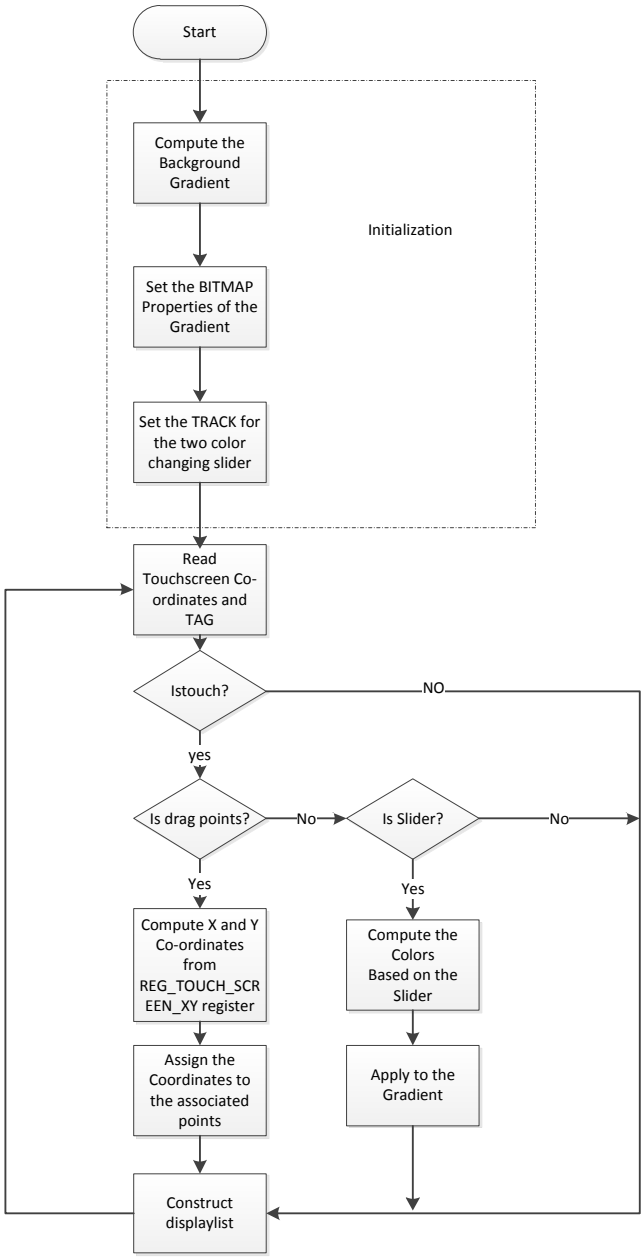

Figure 3.2 Application Flowchart

4 Description

Refer to AN_391_EVE_Platform_Guide for information pertaining to platform setup and the necessary development environment.

4.1 Application Start Screen

Upon completing the setup, the application start screen is displayed.

Figure 4.1 Start Screen

4.2 Gradient Function

This is the main function in which the application will now remain. After an initial display/co-processor list which defines the background and tracker settings, the code will sit in a continuous loop, where it will:

- Check the tracking and Tag registers and calculate the gradient's position and start/end colours for this particular frame
- Draw the background, and clear an area with the scissor functions
- Draw two sliders with their current positions defined by the values calculated above
- Add the text labels C1 and C2 above the sliders
- Draw the gradient with a tagged area at the start and finishing coordinates defined above
- Send the command buffer generated by the above steps to the FT8XX and wait for it to be executed

```
void Gradient(Gpu_Hal_Context_t *phost)
{
    uint32_t Read_xy = 0,tracker;
    uint16_t x1,x2,y1,y2,
        Read_x = 0,Read_y = 0,
        val1=48568,  val2=32765,
        tx = 48,ty = 20,tval;
    uint8_t   Read_Tag  = 0, drag,buff[512];
    int32_t TPsize;

  x1 = 50;y1 = 50;

  TPsize = DispWidth*5/100;

  x2 = DispWidth-60;
  y2 = DispHeight-40;
```

```
drag = 0;
```

This loop creates a bitmap which will be the background. It represents a single pixel column on the display (1 wide by DispHeight high). This takes the form of a symmetrical fading effect. The repeat command will be used later when drawing the bitmap on the screen so that the single column gets repeated over the full screen width.

```
// compute the background gradient effect
for(tval=0;tval<(DispHeight/2);tval++)
{
  buff[DispHeight-1-tval] = buff[tval] = (tval*0.9);
}
Gpu_Hal_WrMem(phost,4096L,buff,DispHeight);
```

The code below creates a display list to set the background for the gradient screen.

It displays the bitmap which was created above. The REPEAT parameter causes the single column to be repeated in the X direction until [DispWidth]. It also sets the tracker properties for the two sliders using the Gpu_CoCmd_Track functions, with Tag values 3 and 4 being associated with these sliders.

The display list ends as always with the DISPLAY command. The application builds a buffer of display list commands and the App_Flush_DL_Buffer then causes this buffer to be sent over SPI to the FT8XX.

A write to the DLSWAP register in the FT8XX then causes the FT8xx to swap the display lists so that the list which has just been written becomes the current one and is displayed on the screen. After executing any Co-Processor commands, the application should wait until REG_CMD_WRITE and REG_CMD_READ registers become equal before sending any further commands to the FT8XX. This ensures that the previous command buffer has been executed.

```
// Set the bitmap properties for the background
App_Set_DlBuffer_Index(0);
App_WrDl_Buffer(phost,CLEAR(1,1,1));
App_WrDl_Buffer(phost,COLOR_A(255));
App_WrDl_Buffer(phost,COLOR_RGB(255,255,255));
App_WrDl_Buffer(phost,BITMAP_HANDLE(2));
App_WrDl_Buffer(phost,BITMAP_SOURCE(4096L));
App_WrDl_Buffer(phost,BITMAP_LAYOUT(L8,1,DispHeight));
App_WrDl_Buffer(phost,BITMAP_SIZE(NEAREST, REPEAT, BORDER, DispWidth, DispHeight));
Gpu_CoCmd_Track(phost,(DispWidth-38),40,8,DispHeight-65,3);
Gpu_CoCmd_Track(phost,(DispWidth-15),40,8,DispHeight-65,4);
App_WrDl_Buffer(phost,DISPLAY());
App_Flush_DL_Buffer(phost);
Gpu_Hal_Wr8(phost,REG_DLSWAP,DLSWAP_FRAME);
Gpu_Hal_WaitCmdfifo_empty(phost);
```

The application then enters a continuous loop where it checks the X and Y coordinates of a touch and also the tag register. Tags 1 and 2 will be used for the dragging points whilst tags 3 and 4 are used for the sliders.

```
do
{
  /* Read the Touch\drag*/
  Read_xy = Gpu_Hal_Rd32(phost,REG_TOUCH_SCREEN_XY);
  Read_Tag = Gpu_Hal_Rd8(phost,REG_TOUCH_TAG);
  // pick the x& y coordinates of the points
  Read_y = Read_xy & 0xffff;
  Read_x = (Read_xy >> 16) & 0xffff;
  if(Read_xy == 0x80008000)  drag = 0;
```

```
    else if(Read_Tag==1 || Read_Tag==2)drag = Read_Tag;

    /* compute the coordinates of two points x&y*/
    if(drag == 1)
    {
       if(Read_x>=(DispWidth-60))0; else  x1 = Read_x;
       if(Read_y>=(DispHeight-20))0; else y1 = Read_y;
    }
    if(drag == 2)
    {
       if(Read_x>=(DispWidth-60))0; else  x2 = Read_x;
       if(Read_y>=(DispHeight-20))0; else y2 = Read_y;
    }
    tracker = Gpu_Hal_Rd32(phost,REG_TRACKER);
    if((tracker&0xff) > 2)
    {
      if((tracker&0xff)==3)
      {
       val1 = (tracker>>16);
      }
      else if((tracker&0xff)==4)
      {
       val2 = (tracker>>16);
      }
    }
```

The main co-processor command list is now created.

First of all, the background is created by clearing the screen and drawing the bitmap image which was created earlier on. The bitmap was given a handle of 2 when it was created earlier.

```
// start the new displaylist
    Gpu_CoCmd_Dlstart(phost);
    App_WrCoCmd_Buffer(phost,CLEAR_COLOR_RGB(55,55,55));
    App_WrCoCmd_Buffer(phost,CLEAR(1,1,1) );
    App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
    App_WrCoCmd_Buffer(phost,VERTEX2II(0,0,2,0));
    Gpu_CoCmd_FgColor(phost,0xffffff);
```

The following commands then put two sliders on the screen. Tagging is enabled and the sliders are associated with Tag 3 and 4.

```
    App_WrCoCmd_Buffer(phost,TAG_MASK(1));

    Gpu_CoCmd_BgColor(phost,val2*255);
    App_WrCoCmd_Buffer(phost,TAG(4));
    Gpu_CoCmd_Slider(phost,(DispWidth-15),40,8,(DispHeight-65),0,val2,65535);

    Gpu_CoCmd_BgColor(phost,val1*255);
    App_WrCoCmd_Buffer(phost,TAG(3));
    Gpu_CoCmd_Slider(phost,(DispWidth-38),40,8,(DispHeight-65),0,val1,65535);
    App_WrCoCmd_Buffer(phost,TAG_MASK(0));
```

Now, the text ‘C1’ and C2’ is displayed on the screen. The COLOR_RGB before this sets the colour to white. The scissor command is then used to select the area where the gradient will be drawn and this is then cleared. This leaves only a border of the original background under the sliders etc. The following command then draws a gradient between the two current coordinate values, which have been calculated by the Tracking routine earlier on.

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

```
    Gpu_CoCmd_Text(phost,DispWidth-45,10,26,0,"C1");

    Gpu_CoCmd_Text(phost,DispWidth-20,10,26,0,"C2");
    App_WrCoCmd_Buffer(phost,SCISSOR_XY(0,10));
    App_WrCoCmd_Buffer(phost,SCISSOR_SIZE(DispWidth-50,DispHeight-30));
    App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
    Gpu_CoCmd_Gradient(phost,x1,y1,val1*255,x2,y2,val2*255);
```

Now that the gradient has been drawn, the two tracking areas must be defined, which indicate the points at which the user can drag the gradient shape. Each of the two tagged areas is drawn as a point. The color mask is set to 0/0/0/0 so that the two points drawn here are actually invisible as they do not affect the red/green/blue/alpha properties of the display. The subsequent steps will place a small black circle with white cross which will be visible, but the larger point created here will be the item which is tagged and is larger so that the user can drag it more easily.

```
    App_WrCoCmd_Buffer(phost,COLOR_RGB(0,0,0));
  // Place a dot for drag the color
    App_WrCoCmd_Buffer(phost,BEGIN(FTPOINTS));
    App_WrCoCmd_Buffer(phost,COLOR_MASK(0,0,0,0));
    App_WrCoCmd_Buffer(phost,POINT_SIZE(TPsize*16));
    App_WrCoCmd_Buffer(phost,TAG_MASK(1));
    App_WrCoCmd_Buffer(phost,TAG(1));
    App_WrCoCmd_Buffer(phost,VERTEX2F(x1*16,y1*16));
    App_WrCoCmd_Buffer(phost,TAG(2));
    App_WrCoCmd_Buffer(phost,VERTEX2F(x2*16,y2*16));

    App_WrCoCmd_Buffer(phost,TAG_MASK(0));
```

The tag is now masked so that the following items will not be tagged. The color mask is set so that subsequent drawing operations will be visible again. Now, a small black point is drawn at each of the coordinates. A white cross is subsequently placed on top of the black circle. The size of the black point can be controlled by adjusting the value of TPsize.

```
    /* calculation of touch point size */
    App_WrCoCmd_Buffer(phost,COLOR_MASK(1,1,1,1));
    App_WrCoCmd_Buffer(phost,POINT_SIZE(TPsize*16));
    App_WrCoCmd_Buffer(phost,BEGIN(FTPOINTS));
    App_WrCoCmd_Buffer(phost,VERTEX2F(x1*16,y1*16));
    App_WrCoCmd_Buffer(phost,VERTEX2F(x2*16,y2*16));

    App_WrCoCmd_Buffer(phost,COLOR_MASK(1,1,1,1));
    App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
    App_WrCoCmd_Buffer(phost,BEGIN(LINES));
    App_WrCoCmd_Buffer(phost,VERTEX2F((x1-TPsize)*16,(y1)*16));
    App_WrCoCmd_Buffer(phost,VERTEX2F((x1+TPsize)*16,(y1)*16));
    App_WrCoCmd_Buffer(phost,VERTEX2F((x1)*16,(y1-TPsize)*16));
    App_WrCoCmd_Buffer(phost,VERTEX2F((x1)*16,(y1+TPsize)*16));
    App_WrCoCmd_Buffer(phost,VERTEX2F((x2-TPsize)*16,(y2)*16));
    App_WrCoCmd_Buffer(phost,VERTEX2F((x2+TPsize)*16,(y2)*16));
    App_WrCoCmd_Buffer(phost,VERTEX2F((x2)*16,(y2-TPsize)*16));
    App_WrCoCmd_Buffer(phost,VERTEX2F((x2)*16,(y2+TPsize)*16));
```

The co-processor list is completed in the usual manner with a DISPLAY command and Swap.

The Co-processor buffer created above within the PC's memory is now sent to the FT8XX, and the CMD_WRITE register updated to the end of this new command sequence, within the App_Flush_Co_Buffer function. Finally, the program waits until the FT8XX confirms that it has completed executing the command list (when REG_CMD_WRITE equals REG_CMD_READ).

```
    App_WrCoCmd_Buffer(phost,END());
    App_WrCoCmd_Buffer(phost,DISPLAY());

    Gpu_CoCmd_Swap(phost);
    App_Flush_Co_Buffer(phost);
    Gpu_Hal_WaitCmdfifo_empty(phost);
  }while(1);
}
```

4.3 Running the Demonstration Code

When running the application, the calibration screen will be displayed first. This uses the FT8XX's built-in calibration routine. It ensures that the FT8XX can align inputs from the touch panel to the image on the screen below accurately. The routine will display a dot and ask the user to tap on this dot. It will then repeat this twice more (with the dot at a different location on the screen in each case).

Figure 4.2 Calibration Screen

The BRT logo animation will then appear on the screen (not shown here). The Gradient introduction screen is then displayed and the application waits for the 'Click to play' button to be pressed, before loading the gradient screen.

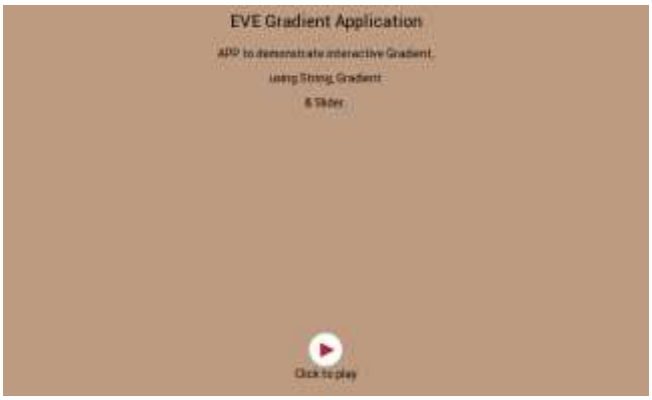

Figure 4.3 Introduction Screen

The main gradient screen will now be displayed. Initially, the display will appear as shown below.

Figure 4.4 Initial Gradient Screen

The gradient can be re-sized and rotated by touching one of the two cursor points (marked with a white cross) and then dragging this point to the new position. The gradients are re-drawn continuously (each time a new co-processor list is sent to the FT8XX) and so appears to smoothly follow the movement of the dragging motion. The screenshot below shows that each of the points have been dragged in to produce a narrower gradient. By dragging one of the points around the screen, the gradient rectangle will also rotate around the other point.

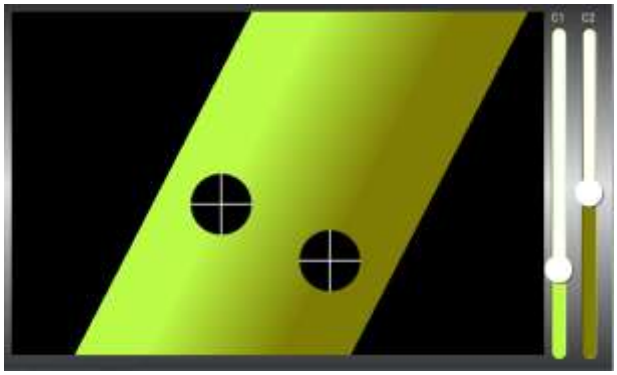

Figure 4.5 Gradient dragged into a narrower shape

Adjusting the sliders on the right-hand side will allow the colours at which the gradient starts and finishes to be changed. In the example below, the sliders have selected red and light-blue. The slider bar itself also indicates the colour selected.

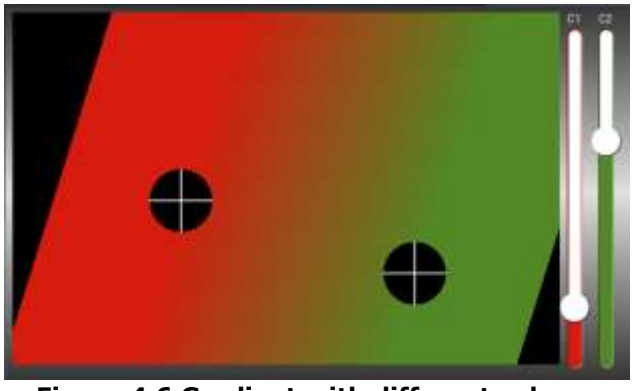

Figure 4.6 Gradient with different colours

5 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 1330
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

- EVE Product page
- 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_391 EVE Platform Guide
- AN_325 FT9XX Toolchain Installation Guide
- AN_281 FT800 Emulator Library User Guide – Covering API Interface for FT800 Emulator
- AN_252 FT800 Audio Primer
- http:/brtchip.com/utilities/ - EVE Screen Editor link
- https://www.ardunio.cc/ - Arduino IDE
- http://brtchip.com/eve/
- http://brtchip.com/mcu/

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 Figures & Tables

List of Figures

List of Tables

NA

Appendix C– Revision History

Document Title: AN_264 FT_App_Gradient
Document Reference No.: BRT_000202
Clearance No.: BRT#121
Product Page: http://brtchip.com/product/
Document Feedback: Send Feedback

Revision	Changes	Date
1.0	Updated Release	2013-08-21
1.1	Updated Release	2013-11-01
1.2	Document migrated from FTDI to BRT (Updated company logo; copyright info; contact information; hyperlinks) Added sections 4.5; 5.1;5.2;5.3;5.4	2018-01-05