Application Note

APPLICATION NOTE

AN_264

FT_App_Gradient

Version 1.1

Document Reference No.: FT_000209

Issue Date: 2013-11-01

This document describes the operation of the Gradient Demo Application running on Visual Studio. The Gradient example demonstrates the way in which the tracking features of the FT800 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.

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
E-Mail (Support): **support1@ftdichip.com** Web: **http://www.ftdichip.com**

Table of Contents

1 Introduction

This document describes the operation of the Gradient Demo Application running on Visual Studio. The Gradient example demonstrates the way in which the tracking features can be used to manipulate graphics on the screen and produce visual effects. Please refer to the sample code project provided with this application note or at:

www.ftdichip.com/Support/SoftwareExamples/FT800_Projects.htm .

1.1 Overview

This application demonstrates the gradient feature and touch tracking/tagging features of the FT800. 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 FT800'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 by using the FT800 with an SPI host. In this case, a PC running Visual Studio (C++) with a C232HM cable is used as the SPI master.

It covers the following topics:

- Brief overview of the Gradient demonstration
- Flow of the code project including the FT800 initialisation and gradient code
- Description of the Gradient function within the application
- Running the demonstration code

Additional documentation can be found at 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

Note: This document is intended to be used along with the source code project provided in section 4.4 or at : http://www.ftdichip.com/Support/SoftwareExamples/FT800_Projects.htm

2 Gradient Overview

The FT800 includes a Gradient command which uses the FT800'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 FT800 will calculate a smooth colour transition between these coordinates.

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

- The user 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 the user 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 FT800 can identify touches in these areas. The user can drag these points to change the starting and finishing position of the gradient bar.

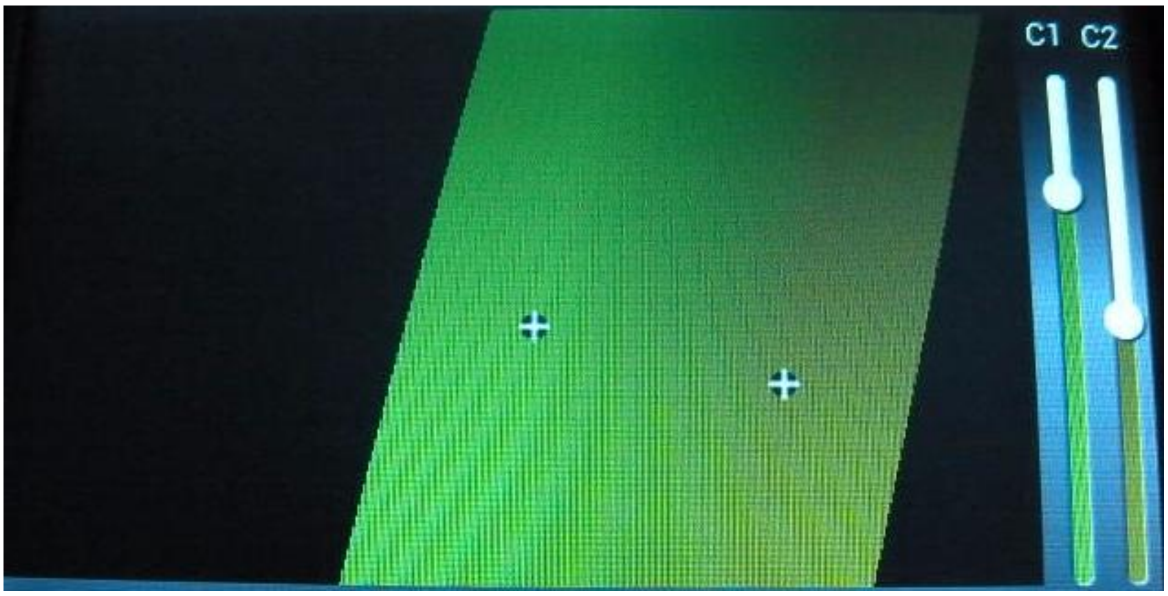


Figure 2.1 The Gradient example application

As described in the following chapters, the SPI Master (in this case a PC running the sample application with a USB to SPI cable) 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 Initilisation

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 FT800 and write to the registers in the FT800 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 gradient 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 FT800, or by writing a series of commands to the Co-Processor FIFO in the FT800 (in which case, the Co-Processor will create a display list in RAM_DL based on the commands). 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 I^{2}C) 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.

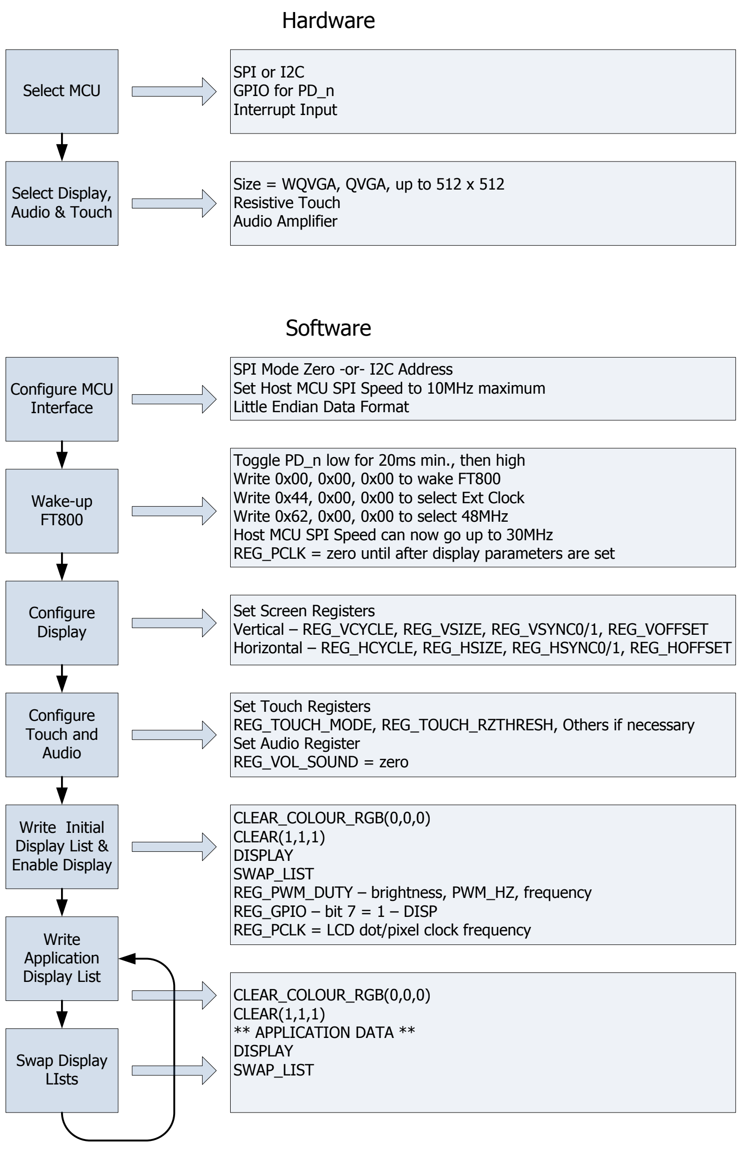

Hardware
Select MCU
SPI or I2C
GPIO for PD_n
Interrupt Input
Select Display, Audio & Touch
Size = WQVGA, QVGA, up to 512 x 512
Resistive Touch
Audio Amplifier
Software
Configure MCU Interface
SPI Mode Zero -or- I2C Address
Set Host MCU SPI Speed to 10MHz maximum
Little Endian Data Format
Wake-up FT800
Toggle PD_n low for 20ms min., then high
Write 0x00, 0x00, 0x00 to wake FT800
Write 0x44, 0x00, 0x00 to select Ext Clock
Write 0x62, 0x00, 0x00 to select 48MHz
Host MCU SPI Speed can now go up to 30MHz
REG_PCLK = zero until after display parameters are set
Configure Display
Set Screen Registers
Vertical – REG_VCYCLE, REG_VSIZE, REG_VSYNC0/1, REG_VOFFSET
Horizontal – REG_HCYCLE, REG_HSIZE, REG_HSYNC0/1, REG_HOFFSET
Configure Touch and Audio
Set Touch Registers
REG_TOUCH_MODE, REG_TOUCH_RZTHRESH, Others if necessary
Set Audio Register
REG_VOL_SOUND = zero
Write Initial Display List & Enable Display
CLEAR_COLOUR_RGB(0,0,0)
CLEAR(1,1,1)
DISPLAY
SWAP_LIST
REG_PWM_DUTY – brightness, PWM_HZ, frequency
REG_GPIO – bit 7 = 1 – DISP
REG_PCLK = LCD dot/pixel clock frequency
Write Application Display List
Swap Display LIsts
CLEAR_COLOUR_RGB(0,0,0)
CLEAR(1,1,1)
** APPLICATION DATA **
DISPLAY
SWAP_LIST

Figure 3.1 Generic EVE Design Flow

3.2 Application Flow

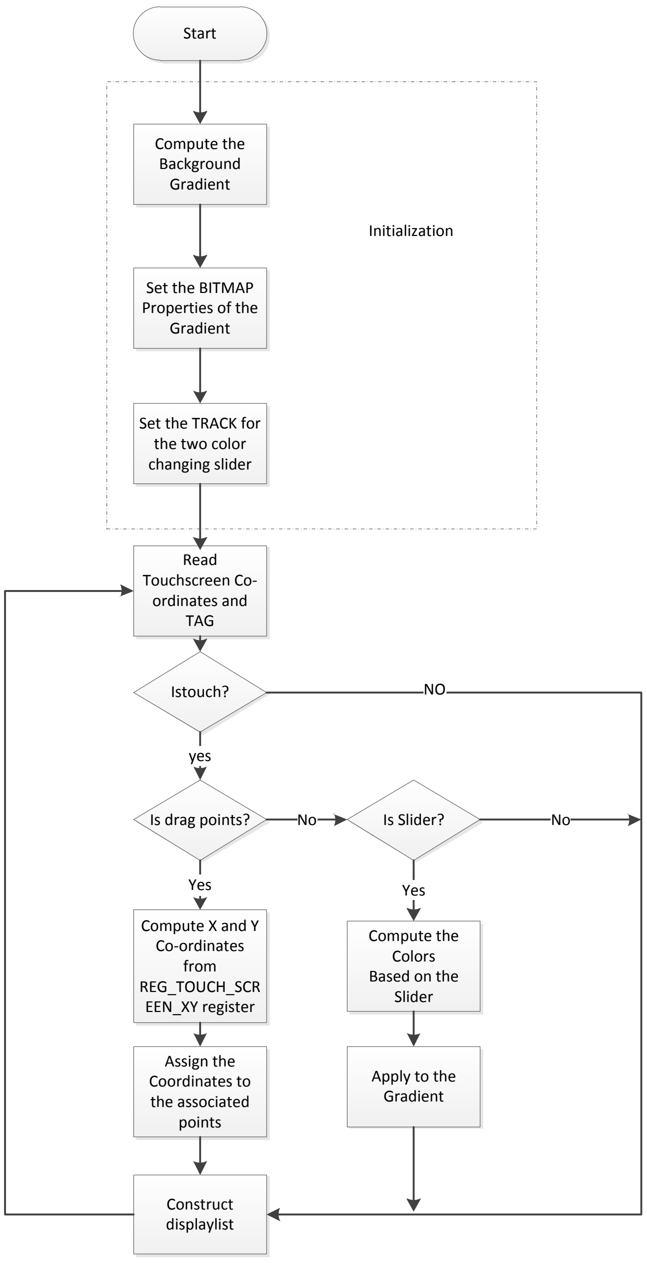

Start
Compute the Background Gradient
Initialization
Set the BITMAP Properties of the Gradient
Set the TRACK for the two color changing slider
Read Touchscreen Co-ordinates and TAG
Istouch?
NO
yes
Is drag points?
No
Is Slider?
No
Yes
Yes
Compute X and Y Co-ordinates from REG_TOUCH_SCREEN_XY register
Compute the Colors Based on the Slider
Assign the Coordinates to the associated points
Apply to the Gradient
Construct displaylist

Figure 3.2 Application Flowchart

4 Description

This section describes the application code used to generate the menu displays.

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 be enabled.

```
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));
```

To confirm that the FT800 is awake and ready to start accepting display list information, the identity register is read continuously until it reports back 0x7C. This register will always read 0x7C if the FT800 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 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);
```

The touch controller can also be configured by setting the 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 then 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);
```

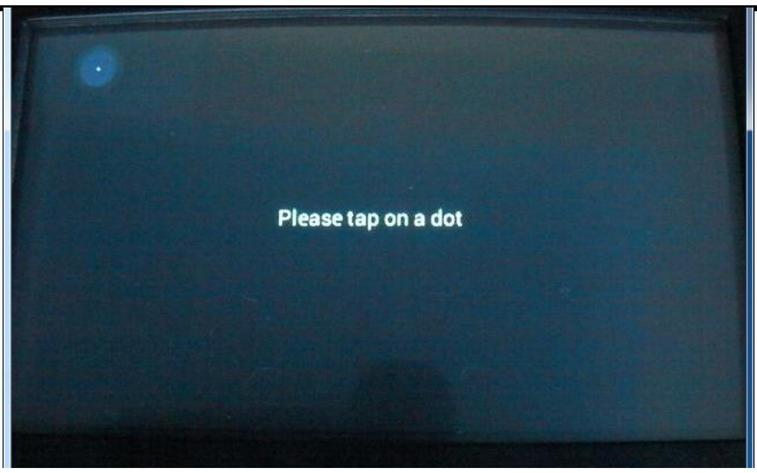


Figure 4.1 Calibration screen

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();
```

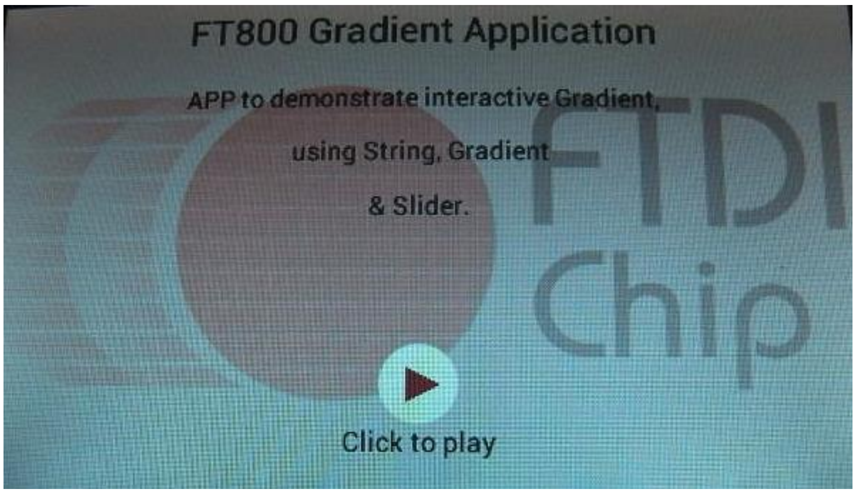
Figure 4.2 Logo screen

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 application will then call one of the three menu functions, depending on which type has been defined.

```
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');
```


FT800 Gradient Application

APP to demonstrate interactive Gradient,

using String, Gradient

& Slider.

Click to play

Figure 4.3 Start screen

4.4 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 FT800 and wait for it to be executed

```
ft_void_t Gradient()
{
  ft_uint32_t  Read_xy = 0, tracker;
  ft_uint16_t  x1,x2,y1,y2, Read_x = 0, Read_y = 0, val1=48568, val2=32765,
  tx = 48, ty = 20, tval;
  ft_uint8_t Read_Tag  = 0,drag, buff[512];

  x1 = 50;y1 = 50;

  x2 = FT_DispWidth-60;
  y2 = FT_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 FT_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.

```
  for(tval=0;tval<(FT_DispHeight/2);tval++)
  {
    buff[FT_DispHeight-1-tval] = buff[tval] = (tval*0.9);
  }
  Ft_Gpu_Hal_WrMem(phost,4096L,buff,FT_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 [FT_DispWidth].It also sets the tracker properties for the two sliders using the Ft_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 Ft_App_Flush_DL_Buffer then causes this buffer to be sent over SPI to the FT800.

A write to the DLSWAP register in the FT800 then causes the FT800 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 FT800. This ensures that the previous command buffer has been executed.

```
  Ft_DlBuffer_Index = 0;
  Ft_App_WrDlCmd_Buffer(phost,CLEAR(1,1,1));
  Ft_App_WrDlCmd_Buffer(phost,COLOR_A(255));
  Ft_App_WrDlCmd_Buffer(phost,COLOR_RGB(255,255,255));
  Ft_App_WrDlCmd_Buffer(phost,BITMAP_HANDLE(2));
  Ft_App_WrDlCmd_Buffer(phost,BITMAP_SOURCE(4096L));
  Ft_App_WrDlCmd_Buffer(phost,BITMAP_LAYOUT(L8,1,FT_DispHeight));
  Ft_App_WrDlCmd_Buffer(phost,BITMAP_SIZE(NEAREST, REPEAT, BORDER,
    FT_DispWidth, FT_DispHeight));

  Ft_Gpu_CoCmd_Track(phost,(FT_DispWidth-38),40,8,FT_DispHeight-65,3);
  Ft_Gpu_CoCmd_Track(phost,(FT_DispWidth-15),40,8,FT_DispHeight-65,4);

  Ft_App_WrDlCmd_Buffer(phost,DISPLAY());
  Ft_App_Flush_DL_Buffer(phost);
  Ft_Gpu_Hal_Wr8(phost,REG_DLSWAP,DLSWAP_FRAME);
  Ft_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_xy = Ft_Gpu_Hal_Rd32(phost,REG_TOUCH_SCREEN_XY);
    Read_Tag = Ft_Gpu_Hal_Rd8(phost,REG_TOUCH_TAG);
    Read_y = Read_xy & 0xffff;
    Read_x = (Read_xy >> 16) & 0xffff;
    if(Read_xy == 0x80008000)  drag = 0;       // 8000 means no touch
    else if(Read_Tag==1 || Read_Tag==2)drag = Read_Tag;


    if(drag == 1)
    {
       if(Read_x>=(FT_DispWidth-60))0; else     x1 = Read_x;
       if(Read_y>=(FT_DispHeight-20))0; else    y1 = Read_y;
    }
    if(drag == 2)
    {
       if(Read_x>=(FT_DispWidth-60))0; else     x2 = Read_x;
       if(Read_y>=(FT_DispHeight-20))0; else    y2 = Read_y;
    }


    tracker = Ft_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.

```
    Ft_Gpu_CoCmd_Dlstart(phost);
    Ft_App_WrCoCmd_Buffer(phost,CLEAR_COLOR_RGB(55,55,55));
    Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1) );
    Ft_App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2II(0,0,2,0));
    Ft_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.

```
    Ft_App_WrCoCmd_Buffer(phost,TAG_MASK(1));
    Ft_Gpu_CoCmd_BgColor(phost,val2*255);
    Ft_App_WrCoCmd_Buffer(phost,TAG(4));
    Ft_Gpu_CoCmd_Slider(phost,(FT_DispWidth-15),40,8,(FT_DispHeight
      -65),0,val2,65535);
    Ft_Gpu_CoCmd_BgColor(phost,val1*255);
    Ft_App_WrCoCmd_Buffer(phost,TAG(3));
    Ft_Gpu_CoCmd_Slider(phost,(FT_DispWidth-38),40,8,(FT_DispHeight
      -65),0,val1,65535);
    Ft_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.

```
    Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
    Ft_Gpu_CoCmd_Text(phost,FT_DispWidth-45,10,26,0,"C1");
    Ft_Gpu_CoCmd_Text(phost,FT_DispWidth-20,10,26,0,"C2");

    Ft_App_WrCoCmd_Buffer(phost,SCISSOR_XY(0,10));
    Ft_App_WrCoCmd_Buffer(phost,SCISSOR_SIZE(FT_DispWidth-50,FT_DispHeight-30));
    Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));

    Ft_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 are 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.

```
    Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0,0,0));
    Ft_App_WrCoCmd_Buffer(phost,BEGIN(FTPOINTS));
    Ft_App_WrCoCmd_Buffer(phost,COLOR_MASK(0,0,0,0));
    Ft_App_WrCoCmd_Buffer(phost,POINT_SIZE(10*16));
    Ft_App_WrCoCmd_Buffer(phost,TAG_MASK(1));
    Ft_App_WrCoCmd_Buffer(phost,TAG(1));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(x1*16,y1*16));
    Ft_App_WrCoCmd_Buffer(phost,TAG(2));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(x2*16,y2*16));
    Ft_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.

```
    Ft_App_WrCoCmd_Buffer(phost,COLOR_MASK(1,1,1,1));
    Ft_App_WrCoCmd_Buffer(phost,POINT_SIZE(5*16));
    Ft_App_WrCoCmd_Buffer(phost,BEGIN(FTPOINTS));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(x1*16,y1*16));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(x2*16,y2*16));


    Ft_App_WrCoCmd_Buffer(phost,COLOR_MASK(1,1,1,1));
    Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
    Ft_App_WrCoCmd_Buffer(phost,BEGIN(LINES));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((x1-5)*16,(y1)*16));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((x1+5)*16,(y1)*16));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((x1)*16,(y1-5)*16));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((x1)*16,(y1+5)*16));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((x2-5)*16,(y2)*16));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((x2+5)*16,(y2)*16));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((x2)*16,(y2-5)*16));
    Ft_App_WrCoCmd_Buffer(phost,VERTEX2F((x2)*16,(y2+5)*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 FT800, and the CMD_WRITE register updated to the end of this new command sequence, within the Ft_App_Flush_Co_Buffer function. Finally, the program waits until the FT800 confirms that it has completed executing the command list by waiting until CMD_WRITE equals CMD_READ.

```
    Ft_App_WrCoCmd_Buffer(phost,END());
    Ft_App_WrCoCmd_Buffer(phost,DISPLAY());
    Ft_Gpu_CoCmd_Swap(phost);
    Ft_App_Flush_Co_Buffer(phost);
    Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
  }while(1);
}
```

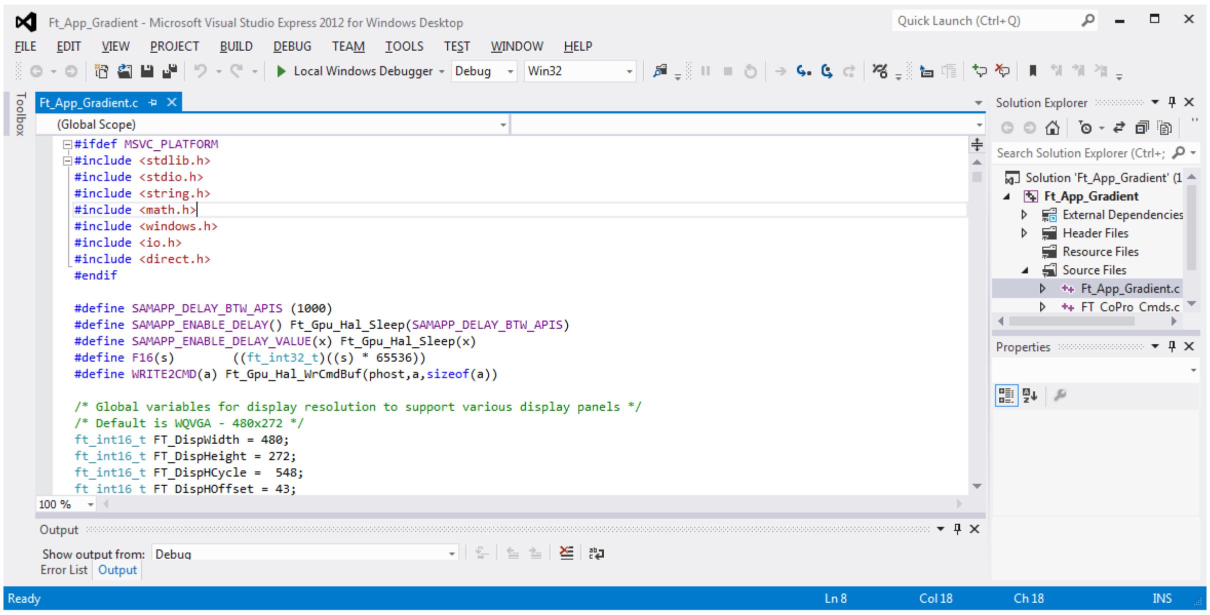
5 Running the demonstration code

This example is shown here when running on a PC with Visual Studio (C++) installed. The FT800 development module (VM800B/VM800C) is connected to the PC using the C232HM cable which acts as a USB to SPI converter.

SCK	ORANGE
MOSI	YELLOW
MISO	GREEN
CS#	BROWN
PD#	BLUE
GND	**BLACK**

Table 1 CM232H Connections to the VM800 pins

The code can now be compiled and run. The debug button can be used to start the application.



```
#ifdef MSVC_PLATFORM
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <windows.h>
#include <io.h>
#include <direct.h>
#endif

#define SAMAPP_DELAY_BTW_APIS (1000)
#define SAMAPP_ENABLE_DELAY() Ft_Gpu_Hal_Sleep(SAMAPP_DELAY_BTW_APIS)
#define SAMAPP_ENABLE_DELAY_VALUE(x) Ft_Gpu_Hal_Sleep(x)
#define F16(s)        ((ft_int32_t)((s) * 65536))
#define WRITE2CMD(a) Ft_Gpu_Hal_WrCmdBuf(phost,a,sizeof(a))

/* Global variables for display resolution to support various display panels */
/* Default is WQVGA - 480x272 */
ft_int16_t FT_DispWidth = 480;
ft_int16_t FT_DispHeight = 272;
ft_int16_t FT_DispHCycle =  548;
ft_int16_t FT_DispHOffset = 43;
```

Figure 4 Visual Studio screenshot

When running the application, the calibration screen will be displayed first. This uses the FT800's built-in calibration routine. It ensures that the FT800 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).

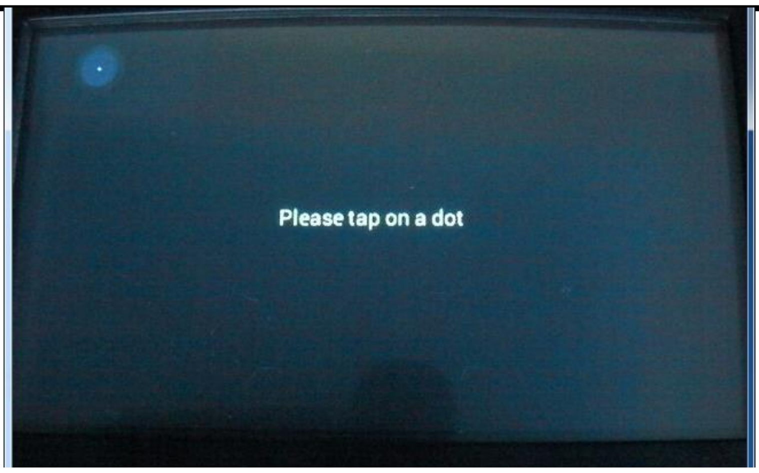


Figure 5.5 Calibration screen

The FTDI 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.

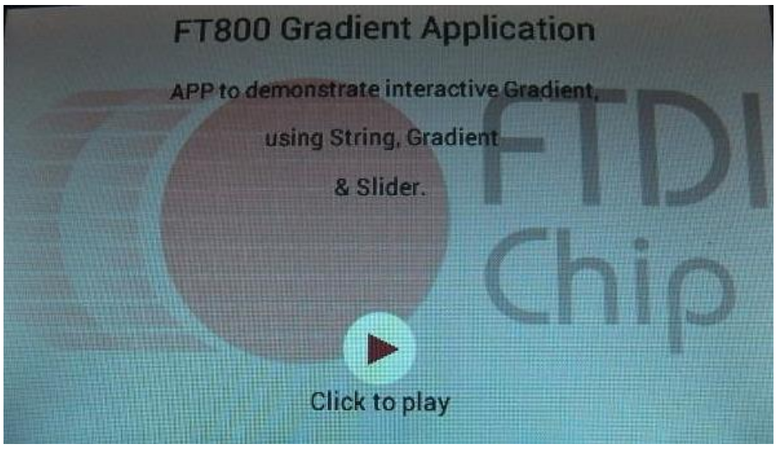

FT800 Gradient Application
APP to demonstrate interactive Gradient,
using String, Gradient
& Slider.
Click to play

Figure 5.6 Introduction screen

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

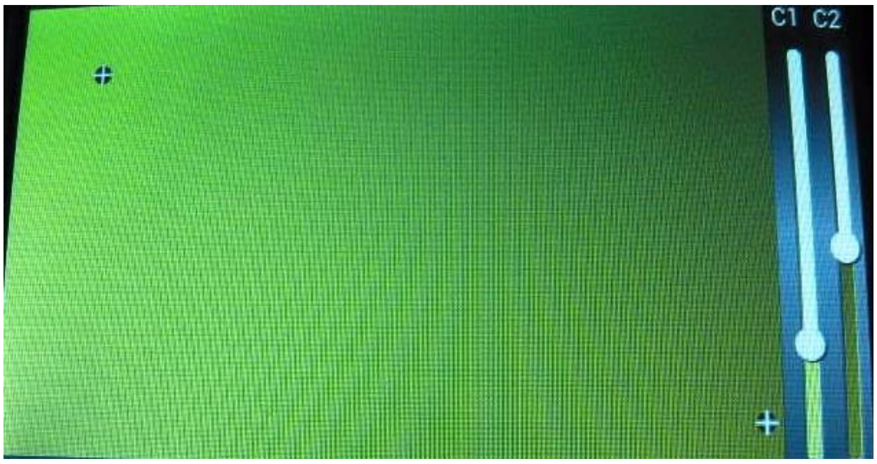


Figure 5.7 Initial gradient screen

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 gradient is re-drawn continuously (each time a new co-processor list is sent to the FT800) 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.

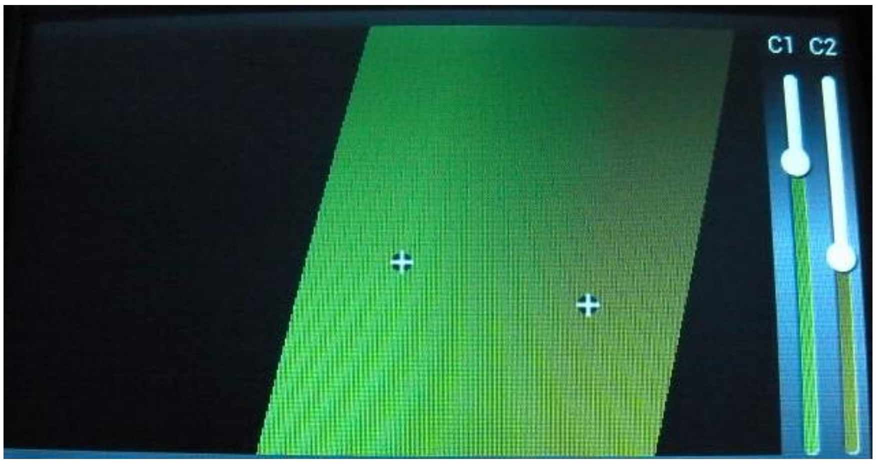


Figure 5.8 Gradient dragged into a narrower shape

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.

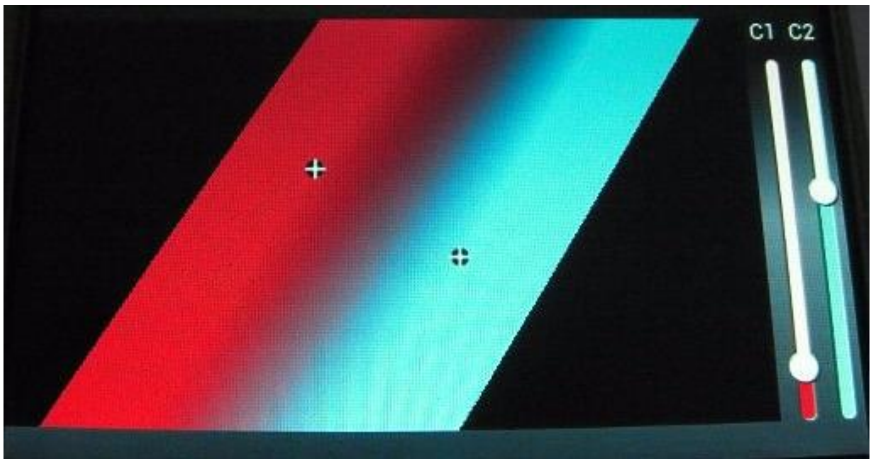


Figure 5.9 Gradient with different colours

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

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_264 FT_App_Gradient
Document Reference No.:	FT_000209
Clearance No.:	FTDI# 359
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	2013-11-01