REVISION RECORD

<table>
<thead>
<tr>
<th>REVNO.</th>
<th>REVDATE</th>
<th>CONTENTS</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2015-02-09</td>
<td>Initial Release</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>2015-09-30</td>
<td>Adding note about FT81x chip</td>
<td></td>
</tr>
</tbody>
</table>

CONTENTS

REVISION RECORD.................................................................................................................. 2

CONTENTS ................................................................................................................................. 2

1. Introduction...................................................................................................................... 3

2. Basic functionality ............................................................................................................ 3
   2.1 Display initialization ................................................................................................. 3
   2.2 Setting background color and text formatting ......................................................... 6
   2.3 Points drawing ........................................................................................................... 7
   2.4 Lines drawing ............................................................................................................ 8
   2.5 Bitmap drawing – animation ...................................................................................... 9
   2.6 Spinners ..................................................................................................................... 10
   2.7 Keys usage ............................................................................................................... 11
   2.8 Sketch function usage .............................................................................................. 14

3. FT801 and FT813 specific functionality ............................................................................. 17
   3.1 Sketch function in capacitive touch panels ............................................................ 17
   3.2 Multi-touch usage ..................................................................................................... 17

4. Related documents ............................................................................................................. 18
1. Introduction

This paper describes Demo Application for Revelation Board. Demo Application show basic functionality of FT800 and FT801 chips. Thanks to the application user can learn how to prepare his own application using FT80x or FT81x chips and STM32F0 microcontroller in fast way. Riverdi prepared two version of Demo application: one for 3.5” and 4.3” displays and the second for 7.0” models. Both Demo applications consist of 6 sub-applications. Each of sub-applications is designed for different touch panel (resistive – RTP, capacitive – CTP). Demo Application were prepared in CooCox IDE using libraries provided by FTDI, and modified by Riverdi. To start programming, first at all, you must connect Revelation Board to the TFT display via 20 pin 0.5 mm FFC to CN8, MCU programmer to SWD connector (for example ST-LINK) and also the 5V power supply via micro USB connector. Figure 1 shows LCM (Liquid Crystal Module) connected to the Revelation Board and ST-LINK. For more information see Revelation Board datasheet.

![Figure 1. Demo application connection](image)

2. Basic functionality

This section provides basic information about usage of FT80x (FT800, Ft801) and FT81x(FT812, FT813) chips connected to Revelation Board. All demo details you can find in demo application source code. Demo shows only few available functions of FT80x and FT81x chips. For more functions see FT80x/FT81x datasheet.

2.1 Display initialization

Pressing SW1 switch you can chose one of 6 sub-applications. It is important to set appropriate parameters for connected display. **FT81x chips use other library than FT80x controller.** Below you can see parameters for 4.3” CTP display with FT801 and for 7.0” CTP display with FT813.
Parameters for 4.3” CTP display with FT801 chip:

```c
    case CTP35:
        DemoTime=DemoTime_CTP43;
        /*4.3" CTP */
        FT_DispWidth = 480;
        FT_DispHeight = 272;
        FT_DispHCycle = 548;
        FT_DispHOffset = 43;
        FT_DispHSync0 = 0;
        FT_DispHSync1 = 41;
        FT_DispVCycle = 292;
        FT_DispVOffset = 12;
        FT_DispVSync0 = 0;
        FT_DispVSync1 = 10;
        FT_DispPCLK = 5;
        FT_DispSwizzle = 0;
        FT_DispPCLKPol = 1;
        break;
        /*Set requested registers*/
        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); //after this display is visible on
        the LCD
        Ft_Gpu_Hal_Wr16(phost, REG_HSIZE, FT_DispWidth);
        Ft_Gpu_Hal_Wr16(phost, REG_VSIZE, FT_DispHeight);
```
Parameters for 7.0” CTP display with FT813 chip:

```c
case CTP70:
    DemoTime = DemoTime_CTP70;
    /* 7.0” CTP */
    FT_DispWidth = 800;
    FT_DispHeight = 480;
    FT_DispHCycle = 1056;
    FT_DispHOffset = 46;
    FT_DispHSync0 = 0;
    FT_DispHSync1 = 10;
    FT_DispVCycle = 525;
    FT_DispVOffset = 23;
    FT_DispVSync0 = 0;
    FT_DispVSync1 = 10;
    FT_DispPCLK = 1;
    FT_DispSwizzle = 0;
    FT_DispPCLKPol = 0;
    break;

/* Set requested registers */
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); // after this display is visible on the LCD
Ft_Gpu_Hal_Wr16(phost, REG_HSIZE, FT_DispWidth);
Ft_Gpu_Hal_Wr16(phost, REG_VSIZE, FT_DispHeight);
```

Of course for different displays those parameters have different values. For more details see application source code and FT80x/FT81x datasheet. Setting parameters is done using `FT_Gpu_Hal_Wr16()` or `FT_Gpu_Hal_Wr8()` functions. This functions write appropriate values to FT800/FT801/FT812/FT813 registers.
2.2 Setting background color and text formatting

In this example the method for setting the background color and text drawing is presented. You will see a red string with the name of the sub-application in the middle of the white screen. Result is presented in Figure 2.

Figure 2. Text formatting and background color example

In the

Figure 2 you can see effect of following code:

```c
Ft_Gpu_CoCmd_Dlstart(phost);
/*Setting first screen*/
Ft_App_WrCoCmd_Buffer(phost,CLEAR_COLOR_RGB(0xff,0xff,0xff)); //white color
Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1)); //clear screen
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0x80,0x00,0x00)); //red color
Ft_Gpu_CoCmd_Text(phost,(FT_DispWidth/2), FT_DispHeight/2, 28, OPT_CENTERX,
"Riverdi 4.3\" CTP demo");
Ft_App_WrCoCmd_Buffer(phost,DISPLAY());
Ft_Gpu_CoCmd_Swap(phost);
/* Download the commands into fifo */
Ft_App_Flush_Co_Buffer(phost);
/* Wait till coprocessor completes the operation */
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
```

You can set background color using macro CLEAR_COLOR_RGB and CLEAR, which setting each of red, blue and green component separately.

You can set color of next graphic objects using macro COLOR_RGB. Using function Ft_Gpu_CoCmd_Dlstart(phost) you start display list which will be send to FT800. Using Ft_Gpu_CoCmd_Text(phost,x,y,font,options,string) function you can display text at location (x,y). The font size and type is setting by font parameter(0 – 31), it can be one of embedded font or you can define your own fonts (see ROM and RAM Fonts). The options is responsible for additional options: By default (x,y) is the top-left pixel of the text and the value of options is zero.
OPT_CENTERX centers the text horizontally, OPT_CENTERY centers it vertically, OPT_CENTER centers the text in both directions. OPT_RIGHTX right-justifies the text, so that the x is the rightmost pixel. The value of OPT_RIGHTX is 2048. The string contain displayed text.

Ft_App_WrCoCmd_Buffer (phost, command) function write command into command buffer of FT800/FT801. Ft_Gpu_CoCmd_Swap (phost) function swap current display list.

2.3 Points drawing

In this example, the method for drawing circles (points) in different colors and locations is presented. You will see 4 points (red, green, blue and yellow) in different sizes and positions on the white background. Bellow this points will be drawing text “POINTS” as described in previous sub-section.

In the Figure 3 you can see effect of following code:

```c
Ft_Gpu_CoCmd_Dlstart (phost);
Ft_App_WrCoCmd_Buffer (phost, CLEAR_COLOR_RGB (255, 255, 255)); // color white
Ft_App_WrCoCmd_Buffer (phost, CLEAR (1, 1, 1));
Ft_App_WrCoCmd_Buffer (phost, COLOR_RGB (128, 0, 0));
Ft_App_WrCoCmd_Buffer (phost, POINT_SIZE ((FT_DispWidth / 5) * 16));
Ft_App_WrCoCmd_Buffer (phost, BEGIN (FTPOINTS)); // start drawing points
Ft_App_WrCoCmd_Buffer (phost, VERTEX2F ((FT_DispWidth / 5) * 16, (FT_DispHeight / 2) * 16));
Ft_App_WrCoCmd_Buffer (phost, COLOR_RGB (0, 128, 0));
Ft_App_WrCoCmd_Buffer (phost, POINT_SIZE ((FT_DispWidth / 6) * 16));
Ft_App_WrCoCmd_Buffer (phost, VERTEX2F ((FT_DispWidth * 2 / 5) * 16, (FT_DispHeight / 2) * 16));
Ft_App_WrCoCmd_Buffer (phost, COLOR_RGB (0, 0, 128));
Ft_App_WrCoCmd_Buffer (phost, POINT_SIZE ((FT_DispWidth / 7) * 16));
Ft_App_WrCoCmd_Buffer (phost, VERTEX2F ((FT_DispWidth * 3 / 5) * 16, (FT_DispHeight / 2) * 16));
Ft_App_WrCoCmd_Buffer (phost, COLOR_RGB (128, 128, 0));
Ft_App_WrCoCmd_Buffer (phost, POINT_SIZE ((FT_DispWidth / 8) * 16));
Ft_App_WrCoCmd_Buffer (phost, VERTEX2F ((FT_DispWidth * 4 / 5) * 16, (FT_DispHeight / 2) * 16));
Ft_App_WrCoCmd_Buffer (phost, COLOR_RGB (0, 128, 128));
Ft_App_WrCoCmd_Buffer (phost, END ());
Ft_Gpu_CoCmd_Text (phost, hoffset, voffset, 31, OPT_CENTERX, "POINTS");
Ft_App_WrCoCmd_Buffer (phost, DISPLAY ());
Ft_Gpu_CoCmd_Swap (phost);
/* Download the commands into fifo */
Ft_App_Flush_Co_Buffer (phost);
```
You can set point size using `POINT_SIZE` command, set point color using `COLOR_RGB` and set location using `VERTEX2F`. There is possibility to draw in very similar way round-cornered rectangles (`RECTS` command), curvature of the corners can be adjusted using `LINE_WIDTH` command.

### 2.4 Lines drawing

In this example, the method for drawing lines in different locations is presented. You will see 8 crossed lines which imitates street map. On this map will be text “Main s.”.

![Figure 4. Lines drawing example](image)

In the Figure 4, you can see effect of following code:

```c
/* Wait till coprocessor completes the operation */
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);

You can set point size using `POINT_SIZE` command, set point color using `COLOR_RGB` and set location using `VERTEX2F`. There is possibility to draw in very similar way round-cornered rectangles (`RECTS` command), curvature of the corners can be adjusted using `LINE_WIDTH` command.

### 2.4 Lines drawing

In this example, the method for drawing lines in different locations is presented. You will see 8 crossed lines which imitates street map. On this map will be text “Main s.”.

![Figure 4. Lines drawing example](image)

In the Figure 4, you can see effect of following code:

```c
/* Wait till coprocessor completes the operation */
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);

You can set point size using `POINT_SIZE` command, set point color using `COLOR_RGB` and set location using `VERTEX2F`. There is possibility to draw in very similar way round-cornered rectangles (`RECTS` command), curvature of the corners can be adjusted using `LINE_WIDTH` command.

### 2.4 Lines drawing

In this example, the method for drawing lines in different locations is presented. You will see 8 crossed lines which imitates street map. On this map will be text “Main s.”.

![Figure 4. Lines drawing example](image)

In the Figure 4, you can see effect of following code:

```c
/* Wait till coprocessor completes the operation */
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);

You can set point size using `POINT_SIZE` command, set point color using `COLOR_RGB` and set location using `VERTEX2F`. There is possibility to draw in very similar way round-cornered rectangles (`RECTS` command), curvature of the corners can be adjusted using `LINE_WIDTH` command.

### 2.4 Lines drawing

In this example, the method for drawing lines in different locations is presented. You will see 8 crossed lines which imitates street map. On this map will be text “Main s.”.

![Figure 4. Lines drawing example](image)

In the Figure 4, you can see effect of following code:

```c
/* Wait till coprocessor completes the operation */
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);

You can set point size using `POINT_SIZE` command, set point color using `COLOR_RGB` and set location using `VERTEX2F`. There is possibility to draw in very similar way round-cornered rectangles (`RECTS` command), curvature of the corners can be adjusted using `LINE_WIDTH` command.

### 2.4 Lines drawing

In this example, the method for drawing lines in different locations is presented. You will see 8 crossed lines which imitates street map. On this map will be text “Main s.”.

![Figure 4. Lines drawing example](image)

In the Figure 4, you can see effect of following code:

```c
/* Wait till coprocessor completes the operation */
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);

You can set point size using `POINT_SIZE` command, set point color using `COLOR_RGB` and set location using `VERTEX2F`. There is possibility to draw in very similar way round-cornered rectangles (`RECTS` command), curvature of the corners can be adjusted using `LINE_WIDTH` command.

### 2.4 Lines drawing

In this example, the method for drawing lines in different locations is presented. You will see 8 crossed lines which imitates street map. On this map will be text “Main s.”.

![Figure 4. Lines drawing example](image)

In the Figure 4, you can see effect of following code:

```c
/* Wait till coprocessor completes the operation */
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);

You can set point size using `POINT_SIZE` command, set point color using `COLOR_RGB` and set location using `VERTEX2F`. There is possibility to draw in very similar way round-cornered rectangles (`RECTS` command), curvature of the corners can be adjusted using `LINE_WIDTH` command.

### 2.4 Lines drawing

In this example, the method for drawing lines in different locations is presented. You will see 8 crossed lines which imitates street map. On this map will be text “Main s.”.

![Figure 4. Lines drawing example](image)

In the Figure 4, you can see effect of following code:

```c
/* Wait till coprocessor completes the operation */
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);

You can set point size using `POINT_SIZE` command, set point color using `COLOR_RGB` and set location using `VERTEX2F`. There is possibility to draw in very similar way round-cornered rectangles (`RECTS` command), curvature of the corners can be adjusted using `LINE_WIDTH` command.

### 2.4 Lines drawing

In this example, the method for drawing lines in different locations is presented. You will see 8 crossed lines which imitates street map. On this map will be text “Main s.”.

![Figure 4. Lines drawing example](image)

In the Figure 4, you can see effect of following code:

```c
/* Wait till coprocessor completes the operation */
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);

You can set point size using `POINT_SIZE` command, set point color using `COLOR_RGB` and set location using `VERTEX2F`. There is possibility to draw in very similar way round-cornered rectangles (`RECTS` command), curvature of the corners can be adjusted using `LINE_WIDTH` command.

### 2.4 Lines drawing

In this example, the method for drawing lines in different locations is presented. You will see 8 crossed lines which imitates street map. On this map will be text “Main s.”.

![Figure 4. Lines drawing example](image)

In the Figure 4, you can see effect of following code:

```c
/* Wait till coprocessor completes the operation */
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);

You can set point size using `POINT_SIZE` command, set point color using `COLOR_RGB` and set location using `VERTEX2F`. There is possibility to draw in very similar way round-cornered rectangles (`RECTS` command), curvature of the corners can be adjusted using `LINE_WIDTH` command.
Ft_App_Flush_Co_Buffer(phost);
// Wait till coprocessor completes the operation
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);

You can change line color using COLOR_RGB command. Drawing lines starts after BEGIN(LINES) command. You must only give coordinate of line start and end using VERTEX2F command.

There is possibility to draw in very similar way lines connected head-to-tail using LINE_STRIP command. There is also possibility to draw edge strips thanks to EDGE_STRIP_A/B/L/R command.

2.5 Bitmap drawing – animation

In this example, the method for drawing bitmaps and creating your own animation is presented. You will see on the screen animation of Riverdi logo.

![Logo animation example](image)

Bitmap animation is performing by function Bitmap(). Animation is done in following way: first at all image is load to FT800/FT801 memory as bitmap, then are applied in loop transformations re-sizing (BITMAP_TRANSFORM_A, BITMAP_TRANSFORM_E) an alpha for fading effect (COLOR_A).

For more details see application code.

There is possibility to perform different bitmap transform functionalities such as scaling, rotation and translation using BITMAP_TRANSFORM_A-F coefficients. These are similar to OpenGL transform functionality. In FT80x/FT81x chip is available playing FTDI logo animation thanks to built-in function (CMD_LOGO).
2.6 Spinners
In this example, the method for drawing spinners is presented. The spinner is an animated overlay that shows the user that some task is continuing. You will see on the screen spinner bellow two strings. After some time spinner will change into different type.

![Figure 6. Spinner widget example](image)

In the Figure 6 you can see effect of following code:

```c
Ft_Gpu_CoCmd_Dlstart(phost);
Ft_App_WrCoCmd_Buffer(phost,CLEAR_COLOR_RGB(64,64,64));
Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0xff,0xff,0xff));
Ft_Gpu_CoCmd_Text(phost,(FT_DispWidth/2), 20, 27, OPT_CENTER, "Different types of spinners");
Ft_Gpu_CoCmd_Text(phost,(FT_DispWidth/2), 60, 27, OPT_CENTER, "Please Wait ...");
Ft_Gpu_CoCmd_Spinner(phost, (FT_DispWidth/2), (FT_DispHeight/2), 0, 0); //style 0 and scale 0
Ft_App_WrCoCmd_Buffer(phost,DISPLAY());
Ft_Gpu_CoCmd_Swap(phost);
// Download the commands into fifo
Ft_App_Flush_Co_Buffer(phost);
// Wait till coprocessor completes the operation
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
```

You can use embedded spinner widget thanks to

`Ft_Gpu_CoCmd_Spinner(phost,x,y,style,scale)`.

You can set location of spinner \((x,y)\) and type \((style 0-3)\) and also scaling \((scale)\).

There is possibility to start an animated screensaver using `CMD_SCREENSAVER` command. After the screensaver command, the co-processor engine continuously updates `REG_MACRO_0` with `VERTEX2F` with varying \((x,y)\) coordinates. With an appropriate display list, this causes a bitmap to move around the screen without any MCU work. Command `CMD_STOP` stops the update process. Note that only one of `CMD_SKETCH`, `CMD_SCREENSAVER`, or `CMD_SPINNER` can be active at one time.
2.7 Keys usage

In this example, the method for drawing and handling keys an buttons is presented. You will see on the screen keyboard. After pressing keys the text will appear at the top of the screen. If displayed string is too long it will be cleared.

Figure 7. Keyboard example

In the Figure 7 you can see effect of following code:

```c
ft_void_t Keyboard()
{
    if(FirstTimeThisDemo == 1)
    {
        TextFont = 29;
        ButtonW = 30;
        ButtonH = 30;
        yBtnDst = 5;
        CurrChar = '|';
        CurrTag = 0;
        PrevTag = 0;
        Pendown = 1;
        CurrTextIdx = 0;
        if((tftConfig==CTP35) ||(tftConfig==CTP43))
        {
            Ft_Gpu_Hal_Wr8(phost, REG_CTOUCH_EXTENDED, CTOUCH_MODE_COMPATABILITY); //set mode to compatability,
        }
        FirstTimeThisDemo=0;
    }
    /* Check the user input and then add the characters into array */
    CurrTag = Ft_Gpu_Hal_Rd8(phost, REG_TOUCH_TAG);
    Pendown = (((Ft_Gpu_Hal_Rd32(phost, REG_TOUCH_DIRECT_XY)>>31) & 0x01);
    CurrChar = CurrTag;
    if(CurrTag==249) // Pendown not happened
        { CurrTag=0; }
    if(0 == CurrTag)
        { CurrChar = '|'; }
    /* check whether pendown has happened */
    if((1 == Pendown) && (0 != PrevTag))
    {
```
CurrTextIdx++;  
/* clear all the characters after 24 are pressed */
if (CurrTextIdx > 24)
{
    CurrTextIdx = 0;
}
}

Ft_Gpu_Cmd_DelayStart(phost);
Ft_App_WrCoCmd_Buffer(phost, CLEAR_COLOR_RGB(64, 64, 64));
Ft_App_WrCoCmd_Buffer(phost, CLEAR(1, 1, 1));
Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(0xff, 0xff, 0xff));
/* Draw text entered by user */
DispText[CurrTextIdx] = CurrChar;
DispText[CurrTextIdx + 1] = '\0';
Ft_App_WrCoCmd_Buffer(phost, TAG_MASK(0));
Ft_Gpu_Cmd_Text(phost, FT_DispWidth/2, 40, TextFont, OPT_CENTER, 
DispText);  //text info
Ft_App_WrCoCmd_Buffer(phost, TAG_MASK(1));

yOffset = 80 + 10;
/* Construct a simple keyboard - note that the tags associated with the keys 
are the character values given in the arguments */
Ft_Gpu_Cmd_PgColor(phost, 0x404080);
Ft_Gpu_Cmd_GradColor(phost, 0x8000ff00);
Ft_Gpu_Cmd_Keys(phost, yBtnDst, yOffset, 10*ButtonW, ButtonH, TextFont, 
(OPT_CENTER | CurrTag), "qwertyuiop");
Ft_Gpu_Cmd_GradColor(phost, 0x8000ff00);
Ft_Gpu_Cmd_Keys(phost, yBtnDst, yOffset, 10*ButtonW, ButtonH, TextFont, 
(OPT_CENTER | CurrTag), "zxcvbnm");  //hilight button z
if (' ' == CurrTag)
{
    Ft_Gpu_Cmd_Button(phost, yBtnDst, yOffset, 10*ButtonW, ButtonH, TextFont, 
OPT_CENTER | OPT_FLAT, " ");  //mandatory to give ' \0' at the end to make sure 
coprocessor understands the string end
}  
else
{
    Ft_Gpu_Cmd_Button(phost, yBtnDst, yOffset, 10*ButtonW, ButtonH, TextFont, 
OPT_CENTER, " ");  //mandatory to give ' \0' at the end to make sure coprocessor 
understands the string end
}

yOffset = 80 + 10;
Ft_Gpu_Cmd_Keys(phost, 11*ButtonW, yOffset, 3*ButtonW, ButtonH, TextFont, (0 
| CurrTag), "789");
Ft_Gpu_Cmd_Keys(phost, 11*ButtonW, yOffset, 3*ButtonW, ButtonH, TextFont, (0 
| CurrTag), "456");
Ft_Gpu_Cmd_Keys(phost, 11*ButtonW, yOffset, 3*ButtonW, ButtonH, TextFont, (0 
| CurrTag), "123");
Ft_App_WrCoCmd_Buffer(phost, COLOR_A(255));
Ft_Gpu_Cmd_Keys(phost, 11*ButtonW, yOffset, 3*ButtonW, ButtonH, TextFont, (0 
| CurrTag), "0.");  //hilight button 0
Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(0xff, 0x00, 0x00));
StringArray[1] = '\0';
strcat(StringArray,"Next demo in ");
Thanks to \texttt{Ft\_Gpu\_Hal\_Rd8(phost,REG\_TOUCH\_TAG)} function you can determine which object on the screen is pressed. \texttt{Pendown} variable stores information that any of objects is pressed. You can tag any drawing object using \texttt{Ft\_App\_WrCoCmd\_Buffer(phost,TAG(249))}, of course value of tag can be different from 249. \texttt{Ft\_App\_WrCoCmd\_Buffer(phost,TAG\_MASK(0))} command disables tagging drawing objects after it. \texttt{Ft\_App\_WrCoCmd\_Buffer(phost,TAG\_MASK(1))} command enables tagging drawing objects after it.

\texttt{FT800} and \texttt{FT812} provide button and key widgets. They can be used in such manner:

\begin{verbatim}
Ft\_Gpu\_CoCmd\_Keys(phost, yBtnDst, yOffset, 10*ButtonW, ButtonH, TextFont, (OPT\_CENTER | CurrTag), "qwertyuiop");
Ft\_Gpu\_CoCmd\_Button(phost,yBtnDst, yOffset, 10*ButtonW, ButtonH, TextFont, OPT\_CENTER | OPT\_FLAT, " ");
\end{verbatim}

You can change foreground color of drawing objects using
\texttt{Ft\_Gpu\_CoCmd\_FgColor(phost,0x404080)}. Using \texttt{Ft\_Gpu\_CoCmd\_GradColor(phost,0x00ff00)} gradient color of objects (only buttons and keys) can be changed.

There is also possibility to draw other widgets such as: analog clocks, gauges, progress bars, scroll bars, sliders, rotary dial controls and toggle switches. For more information please see \texttt{FT80x/FT81x datasheet}. 

2.8 Sketch function usage

In this example, the method for using sketch function is presented. You will see on the screen white area surrounded by the frame with string “Draw test”. After touching white area remains black trace on it. You can clear this area pressing “CLEAR” button.

*Figure 8. Sketch example*

![Sketch example](image)

In the Figure 8 you can see effect of following code:

```c
ft_void_t Sketch()
{
    /*************************************************************************/
    /* Below code demonstrates the usage of sketch function. Sketch function */
    /* draws line for pen movement. Sketch supports L1 and L8 output formats */
    /* Sketch draws till stop command is executed. */
    /*************************************************************************/

    ft_int16_t BorderSz = 40;
    ft_int16_t BorderSz2 = 16;
    ft_uint32_t MemZeroSz;

    MemZeroSz = 1L*(FT_DispWidth - 2*BorderSz2)*(FT_DispHeight - BorderSz - BorderSz2);
    if(FirstTimeThisDemo == 1)
    {
        //Clear memory
        Ft_Gpu_CoCmd_MemZero(phost, (RAM_G+ending_address),MemZeroSz);
    }

    /* Sketch L8 format */
    /* Send command sketch */
    if(tftConfig==RTP35 || tftConfig==RTP43)
    {
        Ft_Gpu_CoCmd_Sketch(phost, BorderSz2,BorderSz, (FT_DispWidth - 2*BorderSz2), (FT_DispHeight - BorderSz - BorderSz2), (RAM_G+ending_address),L8);//sketch in L8 format
    }
    if(tftConfig==CTP35 || tftConfig==CTP43)
    {
        Ft_Gpu_CoCmd_CSketch(phost, BorderSz2,BorderSz, (FT_DispWidth - 2*BorderSz2), (FT_DispHeight - BorderSz - BorderSz2), (RAM_G+ending_address),L8,9000);//sketch in L8 format
    }
}
```
{ Display the sketch */
  Ft_Gpu_CoCmd_Dlistart(phost);
  Ft_App_WrCoCmd_Buffer(phost,CLEAR_COLOR_RGB(0x00, 0x00, 0x00));
  Ft_App_WrCoCmd_Buffer(phost,CLEAR(1, 1, 1));
  Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0xff, 0xff, 0xff));
  Ft_App_WrCoCmd_Buffer(phost,SCISSOR_SIZE((FT_DispWidth, (FT_DispHeight - 2*BorderSz2)), (FT_DispHeight - BorderSz - BorderSz2)));
  Ft_App_WrCoCmd_Buffer(phost,SCISSOR_SIZE((512, 512), (0, 0)));
  Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0, 0, 0));
  /* L8 bitmap display */
  Ft_App_WrCoCmd_Buffer(phost,BEGIN(BITMAPS));
  Ft_App_WrCoCmd_Buffer(phost,BITMAP_SOURCE((RAM_G+ending_address)));///0);
  Ft_App_WrCoCmd_Buffer(phost,BITMAP_LAYOUT(L8, (FT_DispWidth - 2*BorderSz2), (FT_DispHeight - BorderSz - BorderSz2)));
  Ft_App_WrCoCmd_Buffer(phost,BITMAP_SIZE(BILINEAR,BORDER,BORDER, (FT_DispWidth - 2*BorderSz2), (FT_DispHeight - BorderSz - BorderSz2)));
  Ft_App_WrCoCmd_Buffer(phost,VERTEX2F(BorderSz2*16, BorderSz*16));
  Ft_App_WrCoCmd_Buffer(phost,END());
  /* Display the text */
  Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0xff, 0xff, 0xff));
  StringArray[0] = '\0';
  strcat(StringArray,"Next demo in ");
  Ft_Gpu_Hal_Dec2Ascii(StringArray,(ThisDemoEnd+100)+1);
  strcat(StringArray,"s");
  Ft_Gpu_CoCmd_Text(phost,FT_DispWidth, (FT_DispHeight-12), 20, OPT_RIGHTX, StringArray);
  if(tftConfig == RTP35 || tftConfig == RTP43 || tftConfig == CTP35 ||
tftConfig == CTP43)
    if(CurrTag3 == 0)
      if(CurrTag2 == 0)
  { Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(0xff, 0xff, 0xff));
    Ft_App_WrCoCmd_Buffer(phost,TAG(249));
    Ft_Gpu_CoCmd_Button(phost,FT_DispWidth=55,FT_DispHeight=42,60,30,20,0,
      "SKIP->")
    CurrTag2 = Ft_Gpu_Hal_Rd8(phost,REG_TOUCH_TAG);
    Ft_App_WrCoCmd_Buffer(phost,TAG(250));
    Ft_Gpu_CoCmd_Button(phost,FT_DispWidth=55,12,60,30,20,0, "CLEAR");
    CurrTag3 = Ft_Gpu_Hal_Rd8(phost,REG_TOUCH_TAG);
    if(CurrTag3==250)
      Ft_Gpu_CoCmd_MemZero(phost, (RAM_G+ending_address),MemZeroSz); //Clear memory
      CurrTag3=0;
    }
  }
  Ft_App_WrCoCmd_Buffer(phost,DISPLAY());
  Ft_Gpu_CoCmd_Swap(phost);
  /* Download the commands into fifo */
  Ft_App_Flush_Co_Buffer(phost);
  /* Wait till coprocessor completes the operation */
  Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
} /* Send the stop command */
Ft_Gpu_Hal_WrCmd32(phost, CMD_STOP);

/* Update the command buffer pointers - both read and write pointers */
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);

Ft_Gpu_CoCmd_MemZero(phost, (RAM_G+ending_address), MemZeroSz) function clears graphic RAM memory of FT800 chip. Second argument provides information about the start address of memory clearing. Third argument is size of clearing memory area. Sketching is done by Ft_Gpu_CoCmd_Sketch(phost, BorderSz2, BorderSz, (FT_DispWidth-2*BorderSz2), (FT_DispHeight-BorderSz-BorderSz2), (RAM_G+ending_address), L8).

Second and third arguments are coordinates of the screen where function sketch starts. Fourth and fifth arguments are width and height of the sketching area. Sixth argument is graphic RAM address. The last argument is format of storage data.

Ft_App_WrCoCmd_Buffer(phost, SCISSOR_SIZE((512, 512)) and Ft_App_WrCoCmd_Buffer(phost, SCISSOR_XY(0, 0)) specifies location and size of the scissor clip rectangle.

Drawing bitmaps begin after command Ft_App_WrCoCmd_Buffer(phost, BEGIN(BITMAPS)). You can specify location of bitmap in memory using function Ft_App_WrCoCmd_Buffer(phost, BITMAP_SOURCE((RAM_G+ending_address))). Bitmap size is setting with usage of Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE(BILINEAR, BORDER, BORDER, (FT_DispWidth - 2*BorderSz2), (FT_DispHeight - BorderSz - BorderSz2))) command.

There is a possibility to take a snapshot of the current screen thanks to CMD_SNAPSHOT command. This command causes the co-processor engine to take a snapshot of the current screen, and write the result into RAM_G as an ARGB4 bitmap. The size of the bitmap is the size of the screen, given by the REG_HSIZE and REG_VSIZE registers. During the snapshot process, the display should be disabled by setting REG_PCLK to 0 to avoid display glitch. Because co-processor engine needs to write the result into the destination address, the destination address must be never used or referenced by graphics engine.
3. FT801 and FT813 specific functionality

All FT8xx chips share exactly the same graphics feature sets. The touch capabilities of the FT800 and FT812 devices are designed for controlling touch on a resistive panel, while FT801 and FT813 are for capacitive touch and allows simultaneous up to 5 touch points. Therefore, the FT801/FT813 have a different touch engine and touch control register set from the FT800/FT812. All the registers which name starts with “REG_TOUCH” have been assigned to new name “REG_CTOUCH”. By default, the FT800 and FT812 touch engine work in compatibility mode and only one touching point is detected. In extended mode, the FT801 and FT813 touch engine can detect up to 5 touching points simultaneously.

3.1 Sketch function in capacitive touch panels

In FT801 chip you must to use CMD_CSKETCH instead of the CMD_SKETCH. This command has the same functionality as CMD_SKETCH except it has done the optimization for a capacitive touch panel. Because capacitive touch panels have lower sampling frequencies (around 100Hz) to report the coordinates, the sketch functionality updates less frequently compared to resistive touch. CMD_CSKETCH introduces a linear interpolation algorithm to provide a smoother effect when drawing the output line. Please note this command is not applicable to FT800 and FT812 silicon. Code below shows how to use this function. Last parameter is sampling frequency.

```
Ft_Gpu_CoCmd_CSketch(phost, BorderSz2, BorderSz, (FT_DispWidth - 2*BorderSz2), (FT_DispHeight - BorderSz - BorderSz2), (RAM_G+ending_address), L8, 9000); //sketch in L8 format
```

3.2 Multi-touch usage

In this example, the method for handling multi-touch is presented. You will see on the blue screen white text “Please Tap from 2 to 5 points”. You can touch up to 5 points simultaneously. After touching the appropriate polygon will be displayed.

*Figure 9. Multi touch demo*
Detection of 5 simultaneously touch points is provided by reading appropriate registers (stores touch coordinates) as is shown in the following code:

```c
ft_void_t read_extended(ft_int16_t sx[5], ft_int16_t sy[5])
{
    ft_uint32_t sxy0, sxyA, sxyB, sxyC;
    sxy0 = Ft_Gpu_Hal_Rd32(phost, REG_CTOUCH_TOUCH0_XY);
    sxyA = Ft_Gpu_Hal_Rd32(phost, REG_CTOUCH_TOUCH1_XY);
    sxyB = Ft_Gpu_Hal_Rd32(phost, REG_CTOUCH_TOUCH2_XY);
    sxyC = Ft_Gpu_Hal_Rd32(phost, REG_CTOUCH_TOUCH3_XY);

    sx[0] = sxy0 >> 16;
    sy[0] = sxy0;
    sx[1] = sxyA >> 16;
    sy[1] = sxyA;
    sx[2] = sxyB >> 16;
    sy[2] = sxyB;
    sx[3] = sxyC >> 16;
    sy[3] = sxyC;

    sx[4] = Ft_Gpu_Hal_Rd16(phost, REG_CTOUCH_TOUCH4_X);
    sy[4] = Ft_Gpu_Hal_Rd16(phost, REG_CTOUCH_TOUCH4_Y);
}
```

4. Related documents