APPLICATION NOTE

AN_340

FT800_Optimising screen updates with Macro and Append

Version 1.0

Document Reference No.: FT_001108

Issue Date: 2015-09-02

In many FT800 applications, it is desired to update only part of the screen whilst other items, such as the background, remain unchanged. This application note gives simple examples of using the Macro and Append features of the FT800 to do this without needing to re-send the entire display or co-processor list when only part of the display is to be updated.

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://ftdichip.com

Table of Contents

1 Introduction

In many FT800 applications, a screen is displayed which contains items which are static and other items which must be updated over time. For example, a measurement system may have a background which does not change whilst the gauge or numerical read-out in front of the background is to be constantly updated to show the measurement value.

It is possible to create a co-processor list which displays the background and the gauge and to re-send this entire co-processor command set over SPI/I^2C every time the needle is to be re-drawn.

Whilst the FT800's widgets and object oriented architecture already minimise the overhead on the host MCU, the FT800 has several other features which can be used to achieve the same display result without re-sending the entire co-processor list every time. This further reduces the amount of SPI/I^2C bandwidth needed along with the workload on the host MCU.

This application note introduces two ways of performing screen updates without re-sending the entire screen.

It includes an example code project for Visual Studio 2013 which can be used along with the VM800B/C/BU development modules and can be ported over to other platforms. The samples have been made intentionally simple in order to clearly demonstrate the features themselves but the same principles can be used in more comprehensive applications, where the gain in efficiency of re-sending only part of the screen compared to re-sending the entire screen will be even greater.

2 Macros

The Macro feature can be used where the application is creating a display list where one or two parameters are to be changed dynamically but the remainder of the display remains static.

Whenever the FT800 is given a display list to execute (i.e. the display list is written to RAM_DL and a swap occurs) the graphics engine will continuously work through the list of commands in RAM_DL as it renders each line of the screen. Therefore, the host MCU can execute a display list in this way and the FT800 will continue to display the resulting image until the next swap occurs with no further intervention by the host MCU.

The FT800 has two 32-bit registers, REG_MACRO_0 and REG_MACRO_1. A valid display list instruction can be written into each of these registers. The principle behind Macros is that as the graphics processor executes the current display list, each time it encounters a Macro(0) or Macro(1) instruction, it will instead execute the display list instruction from the corresponding register. Since the MCU can modify the Macro0 and Macro1 registers independently of the display list, the instruction and/or it's parameters can be modified on-the-fly with only a simple register write whilst the original display list continues to run.

Display List without Macro

In the example below, the MCU creates a display list which draws a purple rectangle border around the screen and a large point (filled circle) on the screen.

```
Ft_App_WrDlCmd_Buffer(phost, CLEAR(1, 1, 1));     // Clear the screen

// ####### Draw the border around the screen - we draw a purple rectangle the same size as the screen ########
// ####### and then a black rectangle just inside this so that we get a non-filled rectangle #######
Ft_App_WrDlCmd_Buffer(phost, LINE_WIDTH(2 * 16));           // Set the line width for the following rectangle
Ft_App_WrDlCmd_Buffer(phost, BEGIN(RECTS));                 // Start drawing rectangles
// ### purple rectangle has same dimensions as screen ###
Ft_App_WrDlCmd_Buffer(phost, COLOR_RGB(100, 0, 100));       // Purple colour for rectangle border
Ft_App_WrDlCmd_Buffer(phost, VERTEX2F(0 * 16, 0 * 16));     // top-left corner
Ft_App_WrDlCmd_Buffer(phost, VERTEX2F(FT_DispWidth * 16, FT_DispHeight * 16));// bottom-right corner
// ### black rectangle is 10 pixels in for each side ###
Ft_App_WrDlCmd_Buffer(phost, COLOR_RGB(0, 0, 0));           // Black colour for inner rectangle
Ft_App_WrDlCmd_Buffer(phost, VERTEX2F(10 * 16, 10 * 16));   // top-left corner
Ft_App_WrDlCmd_Buffer(phost, VERTEX2F((FT_DispWidth - 10) * 16, (FT_DispHeight-10) * 16)); // bottom-right corner
Ft_App_WrDlCmd_Buffer(phost, END());                        // Finished drawing rectangles

// ####### Draw the blue circle in the centre of the screen #######
Ft_App_WrDlCmd_Buffer(phost, COLOR_RGB(red, green, blue));  // Set the colour of the circle
Ft_App_WrDlCmd_Buffer(phost, POINT_SIZE(30 * 16));          // Set the point size for subsequent points
Ft_App_WrDlCmd_Buffer(phost, BEGIN(FTPOINTS));              // Start drawing points
Ft_App_WrDlCmd_Buffer(phost, VERTEX2F(xoffset * 16, yoffset * 16)); // Draw the circle at the defined position
Ft_App_WrDlCmd_Buffer(phost, END());                        // End of drawing points
Ft_App_WrDlCmd_Buffer(phost, DISPLAY());                    // Display command finishes off the DL

Ft_App_Flush_DL_Buffer(phost);                              // Download above commands to FT800's DL_RAM
SAMAPP_GPU_DLSwap(DLSWAP_FRAME);                            // Do a swap to make the above DL visible
```

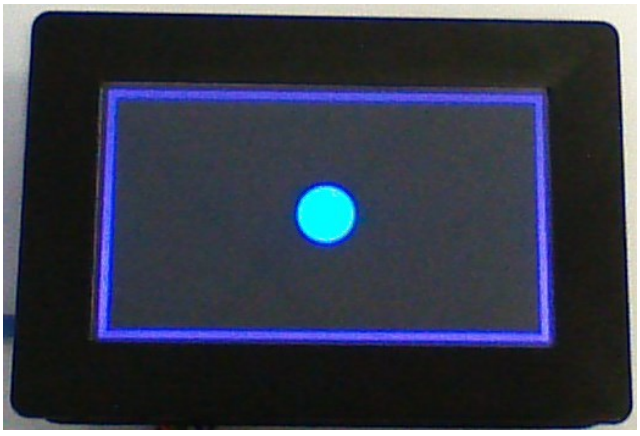

Figure 1 Screen shot from the display list above

In order to make the circle move and change colour, the MCU could re-send the entire display list above with different values for the parameters of the highlighted COLOR_RGB and VERTEX2F instructions.

However, since only two parameters are being changed, it seems inefficient to send the entire display list every time. In this case, macros can be used to reduce the SPI traffic and the workload on the MCU (SPI host)

Display List with Macro

When using the Macros, the principle is to send the initial display list to the FT800 as above but to replace the instructions which will be dynamic with Macro commands. In this case, the COLOR_RGB and the Vertex2F commands are each replaced.

```
// ###############################################################################################################################
// Macro Example
// ###############################################################################################################################

ft_void_t MacroDemo()
{
          ft_int32_t xoffset, yoffset;            // Variables to hold coordinates where the point will be placed
          ft_uint8_t red, green, blue;            // Variables to hold the colour value of the point
          bool go_red = FT_TRUE;                  // Variable used to store direction of colour change between red and blue

          xoffset = 0;                            // X position starts at 0
          yoffset = FT_DispHeight / 2;            // Y position is half-way up the screen

          red = 255;                              // Initial colours are red 100% green 0% blue 0%
          green = 0;
          blue = 0;
```

The REG_MACRO_0 and REG_MACRO_1 must be loaded with valid display list instructions before running the display list since the FT800 will be executing a command from these registers whever it encounters the Macro(0) or Macro(1) instructions.

```
          // ####### Write a valid DL instruction into Macro 0 and Macro 1 registers #######

          Ft_Gpu_Hal_Wr32(phost, REG_MACRO_0, VERTEX2F(xoffset * 16, yoffset * 16));// Write valid instruction into macro0
          Ft_Gpu_Hal_Wr32(phost, REG_MACRO_1, COLOR_RGB(red, green, blue));         // Write a valid instruction into macro1
```

The display list is now created in the normal way, but with the Macro(0) and Macro(1) instructions in place of the Vertex2F and ColorRGB instructions respectively.

```
          // ------------------------------------------------------------------------------------------------------------------------
          // Write the display list which will run continuously. Macro commands are used where we set colour and position.
          // ------------------------------------------------------------------------------------------------------------------------

          // ####### Load a simple display list into the FT800 which draws a point on the screen #######
          // ####### Macro instructions are used in place of the COLOR_RGB and VERTEX commands #######

          Ft_App_WrDlCmd_Buffer(phost, CLEAR(1, 1, 1));     // Clear the screen

          // ####### Draw the border around the screen - we draw a purple rectangle the same size as the screen ########
          // ####### and then a black rectangle just inside this so that we get a non-filled rectangle #######
          Ft_App_WrDlCmd_Buffer(phost, LINE_WIDTH(2 * 16));           // Set the line width for the following rectangle
          Ft_App_WrDlCmd_Buffer(phost, BEGIN(RECTS));                 // Start drawing rectangles
          // ### purple rectangle has same dimensions as screen ###
          Ft_App_WrDlCmd_Buffer(phost, COLOR_RGB(100, 0, 100));       // Purple colour for rectangle border
          Ft_App_WrDlCmd_Buffer(phost, VERTEX2F(0 * 16, 0 * 16));     // top-left corner
          Ft_App_WrDlCmd_Buffer(phost, VERTEX2F(FT_DispWidth * 16, FT_DispHeight * 16));// bottom-right corner
          // ### black rectangle is 10 pixels in for each side ###
          Ft_App_WrDlCmd_Buffer(phost, COLOR_RGB(0, 0, 0));           // Black colour for inner rectangle
          Ft_App_WrDlCmd_Buffer(phost, VERTEX2F(10 * 16, 10 * 16));   // top-left corner
          Ft_App_WrDlCmd_Buffer(phost, VERTEX2F((FT_DispWidth - 10) * 16, (FT_DispHeight-10) * 16)); // bottom-right corner
          Ft_App_WrDlCmd_Buffer(phost, END());                        // Finished drawing rectangles

          // ####### Draw the circle which will move across the screen and change colour #######
          Ft_App_WrDlCmd_Buffer(phost, MACRO(1));                     // Insert the command stored in Macro 1 (color_rgb)
          Ft_App_WrDlCmd_Buffer(phost, POINT_SIZE(30 * 16));          // Set the point size for subsequent points
          Ft_App_WrDlCmd_Buffer(phost, BEGIN(FTPOINTS));              // Start drawing points
          Ft_App_WrDlCmd_Buffer(phost, MACRO(0));                     // Insert the command stored in Macro 0 (vertex2f)
          Ft_App_WrDlCmd_Buffer(phost, END());                        // End of drawing points
          Ft_App_WrDlCmd_Buffer(phost, DISPLAY());                    // Display command finishes off the DL

          Ft_App_Flush_DL_Buffer(phost);                              // Download above commands to FT800's DL_RAM
          SAMAPP_GPU_DLSwap(DLSWAP_FRAME);                            // Do a swap to make the above DL visible

          // ####### The FT800 will now continuously draw the above screen #######
```

The FT800 will now be continuously displaying the result of the above display list. To make the circle position and colour change, the MCU now needs only to change the contents of the REG_MACRO_0 and _1 registers.

The code below runs in a continuous loop, writing new values to these macro registers. An if-else loop is used to calculate new RGB parameters to fade the colour between blue and red. Another if-else loop is used to move the X position of the circle from the left side of the screen to the right side of the screen gradually, and then snap back to the left again.

```
        // ------------------------------------------------------------------------------------------------------------
        // We can now change the Macro 0 and 1 registers whilst the display list loaded above runs continuously
        // Note that we can change the colour and position below with only two writes to the macro registers
        // and without re-sending the DL
        // ------------------------------------------------------------------------------------------------------------

        while (1)
        {
                // ####### Calculate colour values - Fade colour between blue and red by 1 step each time #######

                if (go_red == FT_TRUE)          // If currently in the 'red increasing' direction...
                {
                        red++;                  // ... increase red ...
                        blue--;                 // ... and decrease blue

                        if (red == 255)         // If we reached red = 255, change direction
                                go_red = FT_FALSE;

                }
                else                            // If currently in the 'blue increasing' direction...
                {
                        red--;                  // ... decrease red ...
                        blue++;                 // ... and increase blue

                        if (blue == 255)        // If we reached blue = 255, change direction
                                go_red = FT_TRUE;
                }

                // ####### Calculate x position - move point from left to right. Jump back to 0 when hits the end #######

                if (xoffset < FT_DispWidth)
                        xoffset ++;
                else
                        xoffset = 0;

                // ####### Now update macro registers with our new colour and position values calculated above #######
                // ####### The two lines below are the only activity over SPI to the FT800 required to change #######
                // ####### the colour and position and so this is more efficient than re-sending the entire DL #######
                // ####### again each time #######

                Ft_Gpu_Hal_Wr32(phost, REG_MACRO_1, COLOR_RGB(red, green, blue));
                Ft_Gpu_Hal_Wr32(phost, REG_MACRO_0, VERTEX2F(xoffset * 16, yoffset * 16));

                Ft_Gpu_Hal_Sleep(5);            // Small delay to avoid the position and colour changing too quickly
        }
}
```

The resulting display has the circle moving across the screen and changing colour, as shown below.

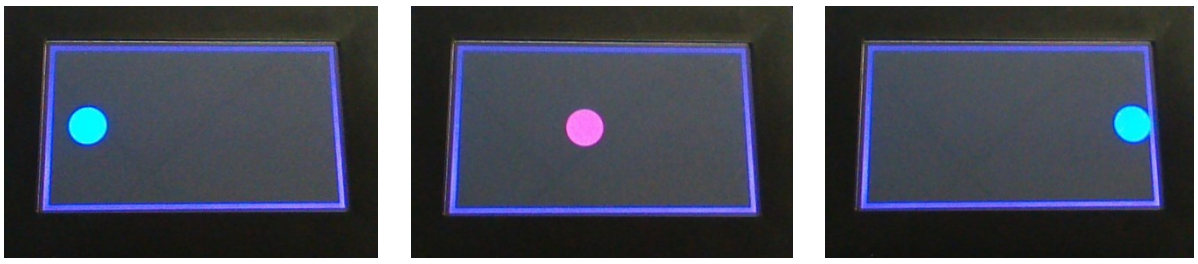

Figure 2 Screen shots from the display list with Macro

This technique has allowed two parts of the display to be dynamic/animated without re-sending the full display list. Even in this simple example, the SPI traffic used to create each step of the animation has gone from sending the entire set of commands shown in the "Display List without Macro" section above, to just sending two 32-bit register writes.

In many real applications, rather than the purple rectangle, there could be many more static items in the display list itself and so the gain in efficiency of writing two registers compared to sending the entire list every time would be even greater.

Summary

In its simplest form, the Macro feature can be used to reduce SPI utilisation and MCU workload in cases where only one or two items on the screen need to be dynamic.

The macro feature is only suitable for use with display lists and not as part of a co-processor list, as the co-processor would regard the Macro command parameters as values rather than pointers to the REG_MACRO_x registers.

Note: Before running a display list containing the Macro command, the associated Macro registers must be written with a valid display list command. Thereafter, the registers must only be updated with a valid display list command as the graphics engine will give undefined results if it tries to execute a non-valid command. This will give unpredictable results on the display.

Other uses of Macro

There are more advanced uses of Macros. For example where a jump instruction could be used in each macro register so that the display list can take one of many different paths, but these are beyond the scope of this application note.

3 Append

In cases where a larger number of items are to be dynamic and/or where a co-processor list is being used, the Append feature can be used.

The principle of this method is to create the static part of the screen as a co-processor or display list and to store it in the graphics RAM. Each time the dynamic part of the screen is to be changed, a new co-processor list is sent which has the commands for the dynamic part but uses the Append command in place of the set of commands which draw the static part. The FT800 will copy the display list components for the static part from grahics RAM over to the display list RAM.

A single Append command may therefore replace a large number of commands sent over SPI which in turn can represent a significant reduction in SPI traffic. This is especially the case where the screen has a lot of static objects and/or where the dynamic parts of the screen are updated at short intervals.

Steps

The following steps are used in the simple example provided:

Creating the static part

- Initialise a block of grahics RAM memory where the static part will be stored. Ensure that this does not overlap with any memory used to store fonts or bitmaps etc. in the application.
- Send a co-processor list containing the static parts of the display to the FT800 using the command FIFO in the normal way. The list does not however terminate with the usual Display and Swap commands since the dynamic part of the list will be appended later.
- The FT800 will now create the display list based on these commands as per normal FT800 operation and so the resulting display list will now be in RAM_DL. This is the display list needed to draw the static part of the display.
- Read the REG_CMD_DL register to determine the end address for this new display list (as an offset from the start of RAM_DL). This equates to the size of the display list and is needed for the following mem copy.
- Use the MemCpy command to copy the display list from RAM_DL to the chosen address in RAM_G

The application can now use this stored display list section by calling the Append command, along with the address in RAM_G (as selected above) and the size (as calculated above), as part of it's main co-processor list.

It can therefore draw a large number of graphics items by calling a single Append command.

Example Code

The simple example provided displays two text strings. The first string "Static" is created by the static part of the list stored in RAM_G. The second string "Dynamic" is created by the main application loop and alternates in colour between Red and Black (which appears invisible as the background is black). The final application therefore displays the word "Static" alongside a flashing word "Dynamic".

It is important to note that whilst this is a very simple example, the static part of the display would often contain large numbers of graphics objects and so the ability to recall these with a single command Append can result in huge savings in SPI traffic and MCU overhead compared to re-sending the entire co-processor list.

Figure 3 Screen shot with Dynamic text in black colour

Figure 4 Screen shot with Dynamic text in red colour

```
// ###############################################################################################################################
// Append Example
// ###############################################################################################################################

ft_void_t AppendDemo()
{
    ft_int32_t xoffset, yoffset, xoffset2, yoffset2, toggle;

    // ####### Variable initialisation #######

    toggle = 0x00;                                  // Used as a flag when flashing the dynamic text on and off

    xoffset = FT_DispWidth / 2;                     // Variables to hold coordinates where the static text will be placed
    yoffset = FT_DispHeight / 2;

    xoffset2 = FT_DispWidth / 3;                    // Variables to hold coordinates where the dynamic text will be placed
    yoffset2 = FT_DispHeight / 3;
```

The application first prepares an area of RAM_G within which the static part of the list stored. In this application, address 0 has been selected (start of RAM_G). Take care to avoid using or overlapping any areas of memory in which the MCU has stored custom fonts or bitmaps etc. for use by the application.

```
    // ####### Clear the first 10K of RAM_G where we will store the static part of the DL #######

    Ft_Gpu_CoCmd_Dlstart(phost);                    // Start co-pro list
    Ft_Gpu_CoCmd_MemSet(phost, 0, 0, 10 * 1024);    // Set a block of memory to zeros between address 0 and address 10*1024
    Ft_App_Flush_Co_Buffer(phost);                  // Send the above co-pro commands off to the FT800 ...
    Ft_Gpu_Hal_WaitCmdfifo_empty(phost);            // ... and wait for them to be processed
```

Now, the co-processor list for the static part of the display is created. In this simple example, the text command is used to write the word “Static”. The co-processor command list is created in the

usual way when creating a screen to display on the FT800 but one difference is that the Display and Swap commands have not been sent since the intention is to store the screen rather than display it immediately.

The Flush command tells the SPI host (in this case the PC running Visual Studio) to send the buffer of commands commands to the FT800 and execute them. In the normal way, the co-processor will now create the corresponding display list within RAM_DL but it will not appear on the screen since the Display and Swap commands were not sent. The FT800 will continue to display whatever content is currently on the screen.

```
// ------------------------------------------------------------------------------------------------------------------------
// Create the static part of the display and store it within the FT800's RAM_G
// ------------------------------------------------------------------------------------------------------------------------

// ####### Create a co-processor list which displays the Static (non-changing) part of the final screen #######

Ft_Gpu_CoCmd_Dlstart(phost);                            // Start co-pro list
Ft_App_WrCoCmd_Buffer(phost, CLEAR_COLOR_RGB(0, 0, 0)); // Background colour is black
Ft_App_WrCoCmd_Buffer(phost, CLEAR(1, 1, 1));           // Clear the screen
Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(255, 255, 255)); // Set colour to white for the subsequent text
Ft_Gpu_CoCmd_Text(phost, FT_DispWidth / 2, FT_DispHeight / 2, 31, OPT_CENTERX | OPT_CENTERY, "Static");
//Ft_App_WrCoCmd_Buffer(phost, DISPLAY());              // We do NOT want to display this yet since ...
//Ft_Gpu_CoCmd_Swap(phost);                             // ... we will be appending the dynamic part of the screen later
Ft_App_Flush_Co_Buffer(phost);                          // Send the above co-pro commands off to the FT800
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);                    // ... and wait for them to be processed

// ####### The co-pro will now have created a new display list based on our commands above #######
// ####### We can now copy this into RAM_G so that we can use it without sending over SPI each time #######
```

The application now copies the display list from RAM_DL to the chosen area in RAM_G. The start address 1,000 was chosen but any free area (i.e. without overlapping stored bitmaps or fonts or other data) could be used. First, the application reads the REG_CMD_DL register which contains the offset of the end of the current display list in RAM_DL. Since only the static part of the display has been added, this value therefore represents the length of the display list associated with the static part.

The CMD_MEMCPY is then used to copy the data from RAM_DL to the chosen address (1,000 in this example) in RAM_G. The length calculated just before this, is used as a parameter to the MEMCPY command to specify the length of data to be copied.

```
dloffset = Ft_Gpu_Hal_Rd16(phost, REG_CMD_DL); // Reading the REG_CMD_DL tells us where the end of the new DL is in
                                               // RAM_DL and therefore the size of our new 'static' display list

Ft_Gpu_Hal_WrCmd32(phost, CMD_MEMCPY);         // Command to copy a block of memory within the FT800
Ft_Gpu_Hal_WrCmd32(phost, 1000L);              // First parameter is destination, copy to address 1,000 decimal
Ft_Gpu_Hal_WrCmd32(phost, RAM_DL);             // Second parameter is the source, here we copy from start of RAM_DL
Ft_Gpu_Hal_WrCmd32(phost, dloffset);           // Third parameter is length of data to copy, as determined above
```

The static part of the display has now been successfully saved.

The application can now create the main co-processor list to draw the actual screen.

A new co-processor list is started. The first item added is the Append command. When the co-processor executes this list of commands, it will add the set of display list instructions saved at the address specified to the RAM_DL. The application had previously stored the list at location 1,000 and had stored the length in variable ‘dloffset’.

The co-processor list then writes the text ‘Dynamic’.

Finally, the Display and Swap commands are used since the resulting screen is to be visible and the commands are then flushed to the FT800.

This co-processor command is sent to the FT800 at 1,000 msec intervals, with the colour of the word “Dynamic” toggling between black and red each time. This gives the impression of red text flashing on and off as the text is invisible against the balck background when coloured black.

```
// ------------------------------------------------------------------------------------------------------------------------
// Sit in a loop which re-calls the static part from RAM_G and adds the dynamic part over SPI
// ------------------------------------------------------------------------------------------------------------------------
```

```
    // ####### Now, we can sit in a loop drawing the screen. Each time, we use our saved DL above which draws the #######
    // ####### static part and then we append the dynamic part which we send each time over SPI #######

    do
    {
          Ft_Gpu_CoCmd_Dlstart(phost);                    // Start co-pro list
          Ft_Gpu_CoCmd_Append(phost, 1000L, dloffset);    // Append command will insert our saved DL which starts @ 1,000
                                                          // and finishes @ 1,000 + length)

          // ####### Now we send the dynamic part over SPI #######

          if (toggle == 0)                        // Alternate colour between Red and Black each time we re-draw the screen
          {
                    Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(255, 0, 0));// Red
                    toggle = 1;
          }
          else
          {
                    Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(0, 0, 0)); // Black (same as background so text not visible)
                    toggle = 0;
          }

          Ft_Gpu_CoCmd_Text(phost, FT_DispWidth / 3, FT_DispHeight / 3, 31, OPT_CENTERX | OPT_CENTERY, "Dynamic");
          Ft_App_WrCoCmd_Buffer(phost, DISPLAY());                  // Display command finished the Display List
          Ft_Gpu_CoCmd_Swap(phost);                                 // Swap to make the new DL active
          Ft_App_Flush_Co_Buffer(phost);                            // Send the above co-pro commands off to the FT800 ...
          Ft_Gpu_Hal_WaitCmdfifo_empty(phost);                      // ... and wait for them to be processed

          Sleep(1000);                                              // Delay so that the text flashes approx every second

    } while (1);    // Keep looping forever
}
```

The do-while loop above runs continuously now. The entire static part of the display can be added with only a single Append command which is much more efficient than sending the entire static part over SPI/ I^2C on every screen update.

Using multiple Append commands

Whilst the example above shows only a single Append command used, more complex applications could save several different sections of static code in different areas of RAM_G and keep a table of starting address and length. The final co-processor list could then contain several pieces of appended code, allowing a screen to be constructed from different static 'building blocks' plus some additional dynamic items added each time the screen is updated.

When planning to build a screen with several appends, each individual section should not start a new display list or clear the screen with the clear(1,1,1) command, as this would clear the contents from the previous appended section. When creating each code section, the REG_CMD_DL register can be written to zero to start the new list instead.

For example, when creating each static section, the following can be used. In this case one section is stored at RAM_G + 0 and the other at RAM_G + 10,000

```
    // STATIC SECTION 1
    Ft_Gpu_Hal_Wr16(phost, REG_CMD_DL, 0)
    Ft_Gpu_CoCmd_Dlstart(phost);                                // Start co-pro list
    Ft_App_WrCoCmd_Buffer(phost, CLEAR_COLOR_RGB(0, 0, 0)); // Background colour is black
    Ft_App_WrCoCmd_Buffer(phost, CLEAR(1, 1, 1));           // Clear the screen
    Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(255, 255, 255)); // Set colour to white for the subsequent text
    Ft_Gpu_CoCmd_Text(phost, FT_DispWidth / 2, FT_DispHeight / 2, 31, OPT_CENTERX | OPT_CENTERY, "Static1");
    [remainder of list as described previously:  store at address 0 and record length as dloffsetStatic1]



    // STATIC SECTION 2
    Ft_Gpu_Hal_Wr16(phost, REG_CMD_DL, 0)
    Ft_Gpu_CoCmd_Dlstart(phost);                                // Start co-pro list
    Ft_App_WrCoCmd_Buffer(phost, CLEAR_COLOR_RGB(0, 0, 0)); // Background colour is black
    Ft_App_WrCoCmd_Buffer(phost, CLEAR(1, 1, 1));           // Clear the screen
    Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(255, 0, 0));     // Set colour to red for the subsequent text
    Ft_Gpu_CoCmd_Text(phost, FT_DispWidth / 2, FT_DispHeight / 2, 31, OPT_CENTERX | OPT_CENTERY, "Static2");
    [remainder of list as described previously:  store at address 10000 and record length as dloffsetStatic2]
```

The main code can then do the clear before calling the Appended sections.

```
do
    {
          Ft_Gpu_CoCmd_Dlstart(phost);                              // Start co-pro list
```

```
        Ft_App_WrCoCmd_Buffer(phost, CLEAR_COLOR_RGB(0, 0, 0));      // Background colour is black
        Ft_App_WrCoCmd_Buffer(phost, CLEAR(1, 1, 1));                // Clear the screen

        Ft_Gpu_CoCmd_Append(phost, 0L, dloffsetStatic1);             // Insert STATIC 1 section
        Ft_Gpu_CoCmd_Append(phost, 10000L, dloffsetStatic2);         // Insert STATIC 2 section

        // ####### Now we send the dynamic part over SPI #######

        if (toggle == 0)                        // Alternate colour between Red and Black each time we re-draw the screen
        {
                Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(255, 0, 0));// Red
                toggle = 1;
        }
        else
        {
                Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(0, 0, 0)); // Black (same as background so text not visible)
                toggle = 0;
        }

        Ft_Gpu_CoCmd_Text(phost, FT_DispWidth / 3, FT_DispHeight / 3, 31, OPT_CENTERX | OPT_CENTERY, "Dynamic");
        Ft_App_WrCoCmd_Buffer(phost, DISPLAY());                     // Display command finished the Display List
        Ft_Gpu_CoCmd_Swap(phost);                                    // Swap to make the new DL active
        Ft_App_Flush_Co_Buffer(phost);                               // Send the above co-pro commands off to the FT800 ...
        Ft_Gpu_Hal_WaitCmdfifo_empty(phost);                         // ... and wait for them to be processed

        Sleep(1000);                                                 // Delay so that the text flashes approx every second

} while (1);    // Keep looping forever
```

Summary

The Append feature can lead to a significant improvement in the efficiency of an EVE application in cases where only part of the screen changes, as the static part can be sent with a single command over SPI/ I^2C on each screen update.

In most applications, the static part would be significantly more complex than the examples here and so the use of the Append command compared to sending the entire display is even more beneficial, especially where the screen is refreshed at a high rate to animate the dynamic parts smoothly.

It also has the advantage that it can be used for both Display Lists and Co-Processor Lists (as these are in turn converted to a display list by the co-processor).

4 Running the Examples

This application note includes a zip file containing a project for Visual Studio 2013, which demonstrates the Macro and Append techniques respectively.

These examples can be used on the following modules (based on those available at the time of writing):

- VM800C Credit card module
- VM800B Basic module
- VM800BU Basic USB module

Note: The VM800C and VM800B will require a separate USB-SPI adapter such as the VA800A-SPI or C232HM-EDHSL-0. The VM800BU includes an on-board USB-SPI interface and so does not require the separate USB-SPI adapter/cable. Please consult the datasheet for the chosen VM800 module for details of the connections required and set-up procedure.

The projects also require an installation of Visual Studio 2013 (or express edition).

Opening the Sample project

This application note is provided with sample code in the AN_340_Files.zip file (see Appendix A–References). Once the module is connected and drivers loaded, the project can be opened by double-clicking on the FT_App_OptimisingScreenUpdates.sln file. This can be found at the following path:

AN_340_Files\Project\Msvc_win32\FT_App_OptimisingScreenUpdates\

In the example below, a VM800B50A-BK module is being used and it is presumed that this is already connected to the PC via a C232HM cable and installed.

Open the FT_App_OptimisingScreenUpdates.c file from the solution explorer toolbar, and scroll to the main function at the bottom (`ft_int32_t main(ft_int32_t argc,ft_char8_t *argv[])`).

Here, the code runs the home_setup and Info functions as with the other FT800 sample applications and then enters one of the three demonstration functions provided for this application note.

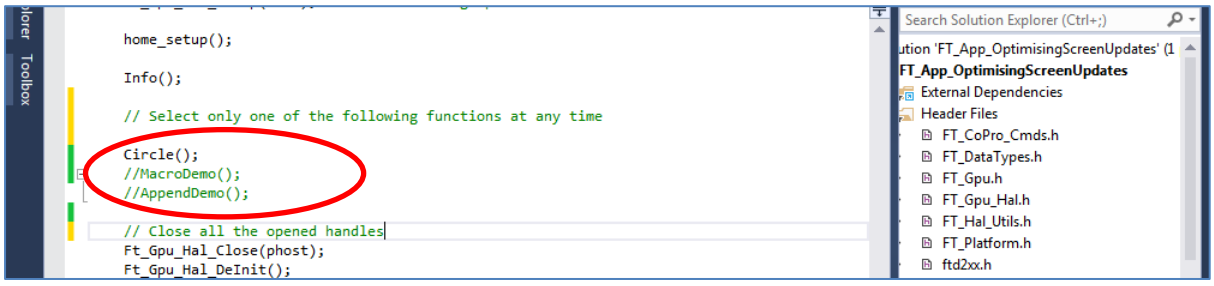

Figure 5 Selecting the demo function to run

Since each of these functions use a while(1) loop, only one of the three should be used at a time and the others should be commented out. Otherwise the program will stay in the first loop which is enabled.

```
Circle();                Code from section Display List without Macro
//MacroDemo();           Code from section Display List with Macro
//AppendDemo();          Code from section Append
```

After un-commenting the desired demonstration function, click the 'Local Windows Debugger' button to run the example.

Running the Demo

Once the project is running, screen of the FT800 module will now display the calibration routine in order to calculate the coefficients which align the touch screen to the LCD panel.

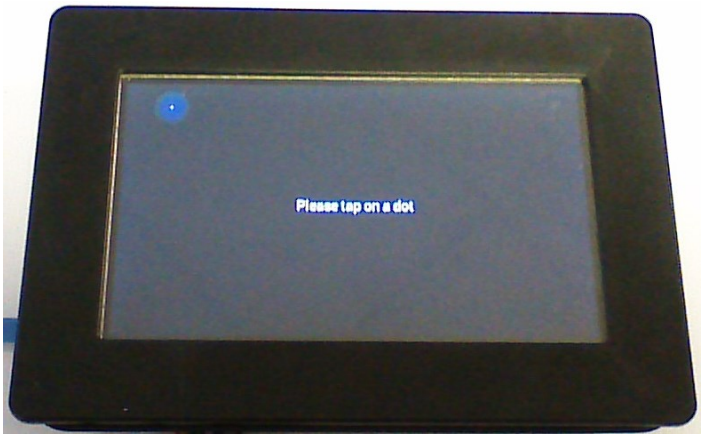

Figure 6 Calibration

After tapping on the three dots as requested, the Info screen appears.

Figure 7 Info screen

Clicking on the play button at the bottom allows the selected demo to begin.

- The Circle() demo will display a static circle only surrounded by the purple rectangle border. This is the code which is subsequently animated by using the Macro example.
- The MacroDemo() demo will display the circle which changes colour and moves across the screen.
- The AppendDemo() will display the word "Static" and the word "Dynamic" will appear to flash on the screen.

Screenshots of each demo are shown in the earlier sections of this application note.

5 Conclusion

This application note has demonstrated two techniques for updating part of the FT800's screen without constantly sending the entire display or co-processor list. It has also provided a simple example of each technique to illustrate the principle. These techniques are used in some of the more advanced FTDI EVE examples, such as the FT_App_Gauges example and the info screen at the beginning of the sample applications available from the FTDI website (see Appendix A– References).

Whilst the object-oriented design of the EVE series in combination with the co-processor and widgets already take much of the workload away from the host processor, techniques such as the Macro and Append allow a further optimisation. This in turn results in less processing on the MCU side and less bandwidth used on the SPI or I^2C link between the EVE device and the MCU whilst still creating the same visual display.

This could free up CPU time and SPI/ I^2C bandwidth for other parts of the application. Or it could allow a higher refresh rate for a given MCU specification, or perhaps even allow a smaller MCU to be used.

This application note has intentionally used simple examples to avoid hiding the key principles of each method. But in real applications with complex screens, where often much of the screen is static, these techniques can provide a significant improvement over refreshing the entire screen.

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. FT800 Datasheet
2. FT800 Programmers Guide
3. FT800 Samples page
4. EVE Product Page
5. Source code file

Acronyms and Abbreviations

Terms	Description
EVE	Embedded Video Engine
LCD	Liquid Crystal Display
SPI	Serial Peripheral Interface

Appendix B – List of Tables & Figures

List of Figures

Appendix C– Revision History

Document Title:	AN_340 FT800_Optimising screen updates with Macro and Append
Document Reference No.:	FT_001108
Clearance No.:	FTDI# 438
Product Page:	http://www.ftdichip.com/EVE.htm
Document Feedback:	Send Feedback

Revision	Changes	Date
1.0	Initial release	2015-02-09