EVE2 Module JPEG Viewer

An EVE2 Module example demonstrating how to display JPEGs.

Application Note

Revision 1.0

Introduction

This application note demonstrates how JPEG images can be loaded and displayed on the EVE2 module. The Graphics Ram available on the EVE2 Module is used to store four images, which are then individually displayed on screen. A set of buttons are also included, providing users the ability to cycle through each individual image.

Connections

For this example, an EVE2 module was connected and powered through an EVE2 USB to SPI Bridge. The USB Bridge was directly connected to a computer using a mini USB type B cable.

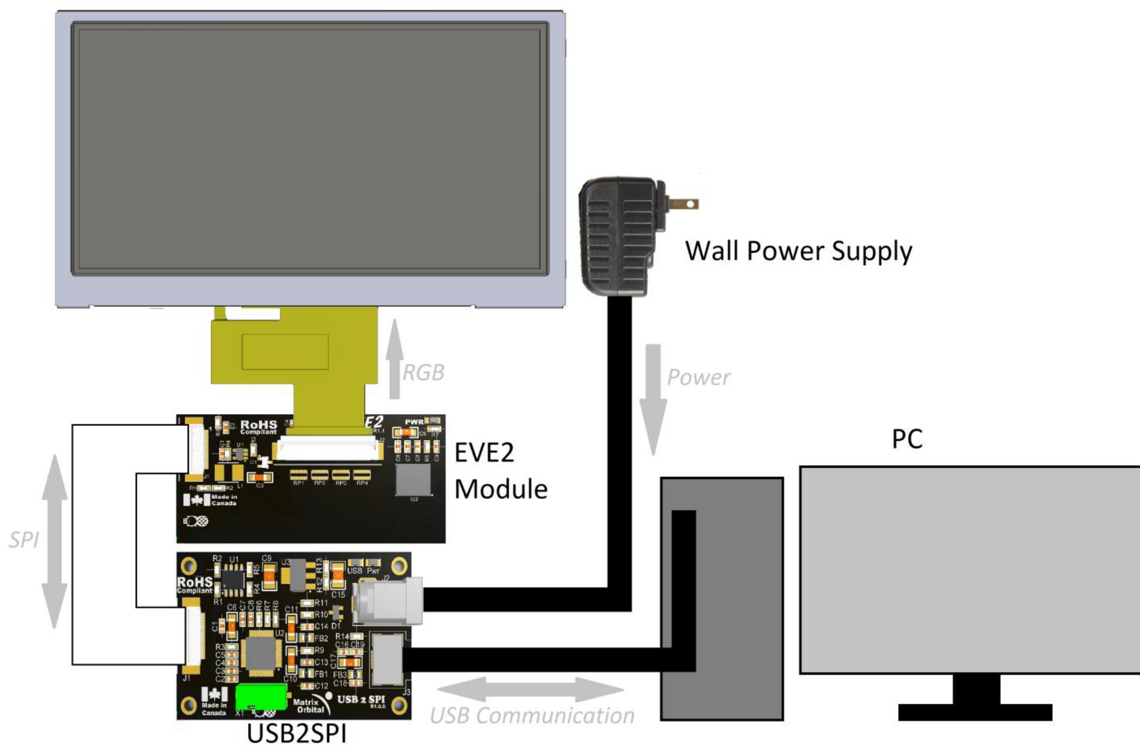

Figure 1: EVE2 Module JPEG Viewer hardware connections.

Table 1: Parts used in the JPEG Viewer Application note

Label	Description	Part Number
EVE2 Module	EVE2 board takes SPI data and sends RGB Information to the display	EVE2-43A-BLM-TPR Module
SPI	20 pin FFC Cable with 0.5mm pitch. Allows communication between a host controller and EVE2	Würth Elektronics 687620050002 or similar
USB2SPI	Convert USB protocol communication to SPI.	Matrix Orbital USB2SPI bridge
USB Communication.	Mini-USB cable to communicate and power smaller displays*	EXTMUSB3FT
Wall Power Supply	5V wall power supply	PWR-ACDC-5V2A

***Note:** Smaller displays can be powered using the USB Communication header. For larger displays such as the EVE2-50A and EVE2-70A, additional power must be supplied via the power jack.

Code

The code used in this example was taken from FTDI's EVE2 Sample Application, and modified to highlight how images can be displayed on the EVE2 Module. The code for this example was written in C and makes use of the libraries provided by FTDI, including their write and read functions. Modifications are contained in the SampleApp.c file and all other files remain untouched. FTDI's Sample Application code can be found on their website, www.ftdichip.com.

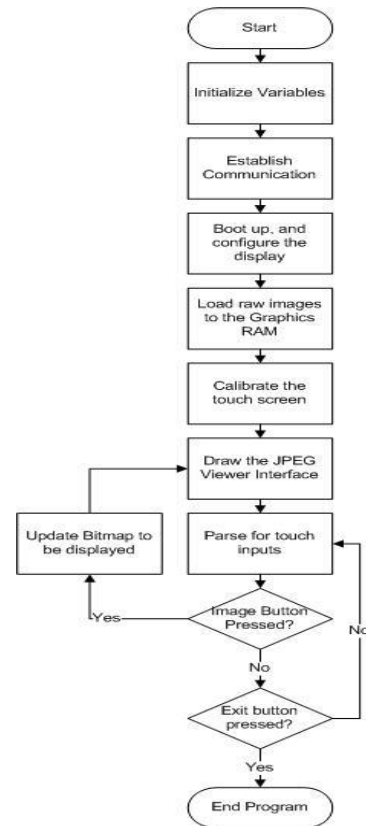

Figure 2: JPEG Viewer flow chart

The code was developed in Microsoft Visual Studio, and can be compiled and run through the Visual Studio compiler. Microsoft Visual Studio can be downloaded for free at https://www.visualstudio.com/vs/community/

Since the code base was taken from one of FTDI's Sample applications, and modified for our purposes, we have taken the liberty to comment certain sections of the code to ensure that the user understands how the code operates, and how FTDI's functions work.

Once running, variables will be initialized, and the EVE2 will be configured for communication with the TFT. All four images are then uploaded to the EVE2 Module's graphics RAM. A calibration sequence will then be called, and the user will be prompted to calibrate the display's touch screen. After calibration, the EVE2 module will draw a gradient, and draw the first bitmap on screen. Next, a set of buttons will be drawn, as well as a white rectangle to highlight which image is currently being displayed. The two arrow buttons can be used to cycle through each image, and the four buttons above can be used to select specific images.

By default, the EVE2 module is configured for a 480 x 272 resolution display in horizontal configuration. If a different display resolution is being used, changes may need to be made to the configuration registers before the display can operate properly.

Once configured, a 'for' loop will initiate, uploading four images to the EVE2 Module's Graphics RAM. In this loop, a raw image is loaded from the file, given a handle, a source, a layout, and a size. At the end of the loop, the bitmap address value gets incremented to ensure that the images do not get overwritten as additional images are uploaded.

```c
// MO: This for loop prepares and loads 4 images into the EVE2's RAM_G. It will also provide a bitmap handle for each image.
for (int i = 0; i < 4; i++){
    Ft_App_LoadRawFromFile(imageNames[i], bitmapAddress[i]);                  // MO: Load an image into Graphics RAM
    Ft_App_WrCoCmd_Buffer(phost, BITMAP_HANDLE(i + 1));                       // MO: Give the image a handle, from 1-4
    Ft_App_WrCoCmd_Buffer(phost, BITMAP_SOURCE(bitmapAddress[i]));            // MO: Determine the source for the image
    Ft_App_WrCoCmd_Buffer(phost, BITMAP_LAYOUT(RGB565, 480 * 2, 272));        // MO: Specify the layout of the image
    Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE(NEAREST, BORDER, BORDER, 480, 272)); // MO: Determine the size of the image
    bitmapAddress[i+1] = bitmapAddress[i] + (480 * 2 * 272);                  // MO: Adjust the bitmap address
}
```

Figure 3: Loading Images into Graphics RAM

The raw image 'for' loop is followed by a touch screen calibration routine. In this routine, a CALIBRATE command will be sent to the EVE2 coprocessor, and three dots will be drawn on screen. The touch screen will be calibrated once all three dots have been pressed.

Once calibrated, two calculations occur to scale the bitmaps and buttons to the screen resolution specified in the display configuration. After the scaling calculations, a while loop commences.

Within the 'While' loop the EVE2 module will begin drawing a gradient background, the first bitmap, and a white rectangle. One button will be drawn at either side of the bitmap. A 'for' loop will draw four buttons above the current bitmap. The white rectangle is used to highlight one of the four buttons, and show which bitmap in the list is currently being displayed.

```c
Ft_Gpu_CoCmd_Dlstart(phost); // MO: Start the display list
Ft_Gpu_CoCmd_Gradient(phost,0,0,0,272,480,16777215); // MO: Place a black-white gradient on screen to use as a background

// MO: Draw the current bitmap on screen
Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(255, 255, 255));
Ft_App_WrCoCmd_Buffer(phost, BITMAP_HANDLE(currentBitmapHandle));
Ft_App_WrCoCmd_Buffer(phost, BITMAP_TRANSFORM_A(scaleX)); // MO: Transform the bitmap using the calculated scaleX value
Ft_App_WrCoCmd_Buffer(phost, BITMAP_TRANSFORM_E(scaleY)); // MO: Transform the bitmap using the calculated scaleY value
Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE(NEAREST, BORDER, BORDER, bitmapWidth, bitmapHeight));
Ft_App_WrCoCmd_Buffer(phost, BITMAP_SIZE_H((bitmapWidth >> 9), (bitmapHeight >> 9)));
Ft_App_WrCoCmd_Buffer(phost, BEGIN(BITMAPS));
Ft_App_WrCoCmd_Buffer(phost, VERTEX2F(((FT_DispWidth * 5) / 40) * 16, (FT_DispHeight / 10) * 16));
Ft_App_WrCoCmd_Buffer(phost, END());

// MO: Draw a white rectangle to highlight which bitmap is currently being displayed
x = ((FT_DispWidth * 26) / 160) + ((currentBitmapHandle - 1)* ((FT_DispWidth * 29) / 160)); // MO: Calculate where to place the white rectangle based on tl
Ft_App_WrCoCmd_Buffer(phost, LINE_WIDTH(5 * 16));
Ft_App_WrCoCmd_Buffer(phost, BEGIN(RECTS));
Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(255, 255, 255));
Ft_App_WrCoCmd_Buffer(phost, VERTEX2F(x * 16, (FT_DispHeight / 144) * 16));
Ft_App_WrCoCmd_Buffer(phost, VERTEX2F(((x + ((FT_DispWidth * 24) / 160)) * 16), (FT_DispHeight / 14) * 16));
Ft_App_WrCoCmd_Buffer(phost, END());

// MO: Draw Left and Right Navigation buttons
Ft_App_WrCoCmd_Buffer(phost, TAG(1));
Ft_Gpu_CoCmd_Button(phost, ((FT_DispWidth * 1) / 40), (FT_DispHeight / 10), ((FT_DispWidth * 2) / 40), ((FT_DispHeight * 8) / 10), 25, tagoption1, "<");
Ft_App_WrCoCmd_Buffer(phost, TAG(2));
Ft_Gpu_CoCmd_Button(phost, ((FT_DispWidth * 37) / 40), (FT_DispHeight / 10), ((FT_DispWidth * 2) / 40), ((FT_DispHeight * 8) / 10), 25, tagoption2, ">");

// MO: Draw an Exit button
Ft_App_WrCoCmd_Buffer(phost, TAG(7));
Ft_Gpu_CoCmd_Button(phost, ((FT_DispWidth * 67) / 160), ((FT_DispHeight * 130) / 144), ((FT_DispWidth * 26) / 160), (FT_DispHeight / 14), 25, 0, "Exit");
tagoption1 = tagoption2 = 0;

// MO: Draw four image selection buttons
char buttonNumber[2];
for (int t = 0; t<4; t++)
{
    // MO: Determine where each button will be placed
    x = ((FT_DispWidth * 25) / 160) + ((t)* ((FT_DispWidth * 29) / 160));
    y = (FT_DispHeight / 144);
    w = ((FT_DispWidth * 26) / 160);
    h = (FT_DispHeight / 14);

    Ft_App_WrCoCmd_Buffer(phost, TAG(t + 3));
    sprintf(buttonNumber, "%d", (t + 1));
    Ft_Gpu_CoCmd_Button(phost, x, y, w, h, 28, 0, buttonNumber);
}
```

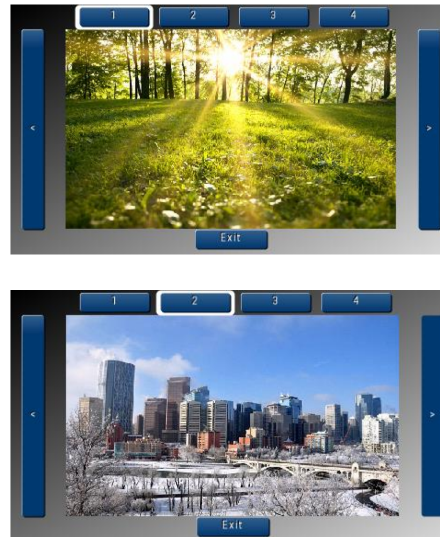

Figure 4: Drawing the JPEG Viewer Interface

Once the JPEG Viewer user interface is drawn, the EVE2 will continuously parse for touch inputs. If the user touches one of the buttons located at the left or right of the bitmap, the current bitmap will be incremented or decremented appropriately. The screen will then be updated with the new bitmap. If either button is held down, the EVE will cycle through the list of bitmaps.

```c
// MO: Read touch tag register. If no button was pressed, reset all touch values.
tagval = Ft_Gpu_Hal_Rd8(phost, REG_TOUCH_TAG);
if ((tagval == 0) || (tagval == 255))
{
    sleepDelay = 30;
    arrowRight = arrowLeft = 0;
    button1HeldCounter = button2HeldCounter = 0;
}

// MO: Wait for button to be pressed
while ((tagval == 0) || (tagval == 255)){
    tagval = Ft_Gpu_Hal_Rd8(phost, REG_TOUCH_TAG);
    printf("tagval = %i\n", tagval);
}
// MO: Report which button was pressed and respond appropriately
printf("tagval = %i\n", tagval);
```

Figure 5: Touch Parser loop

If one of the four buttons above the bitmap is pressed, the current bitmap handle will be changed to a specific value, and the screen will be updated to the corresponding bitmap.

At any point while the code is running, the user will be able to exit the program by pressing the 'Exit' button at the bottom of the screen. When the 'Exit' button is pressed, the screen will clear and the program will close.

Conclusion

This example demonstrates how one can load and display raw images on the EVE2 Module. In addition, user control was implemented, allowing for basic user inputs during operation.

The application note only scratches the surface of what the EVE2 can be programmed to do. By combining the basic primitive commands with more advanced commands, users will be able to unleash the full potential of the EVE2 module.

Figure 6: Screenshot of the program during operation

For more demos and tutorials on the EVE2 module and EVE2 USB-SPI Bridge, check out our forums at lcdforums.com

Stay up to date by subscribing to our Youtube channel, https://www.youtube.com/user/MatrixOrbital

Contact

Sales
Phone: 403.229.2737
Email: sales@matrixorbital.ca

Support
Phone: 403.204.3750
Email: support@matrixorbital.ca

Online
Purchasing: www.matrixorbital.com
Support: www.matrixorbital.ca