EVE2 Module Drawing Primitives

An EVE2 Module example demonstrating how to draw primitives and buttons

Application Note

Revision 1.0

Introduction

This application note demonstrates how basic primitives can be used to draw simple images. In addition, touch functionality is implemented to demonstrate how easy it can be to integrate an interactive user interfaces on the EVE2. Using only primitive shapes, the Matrix Orbital Logo could be recreated on any EVE2 module, and by using the built-in button widget, buttons were added to provide users control.

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.

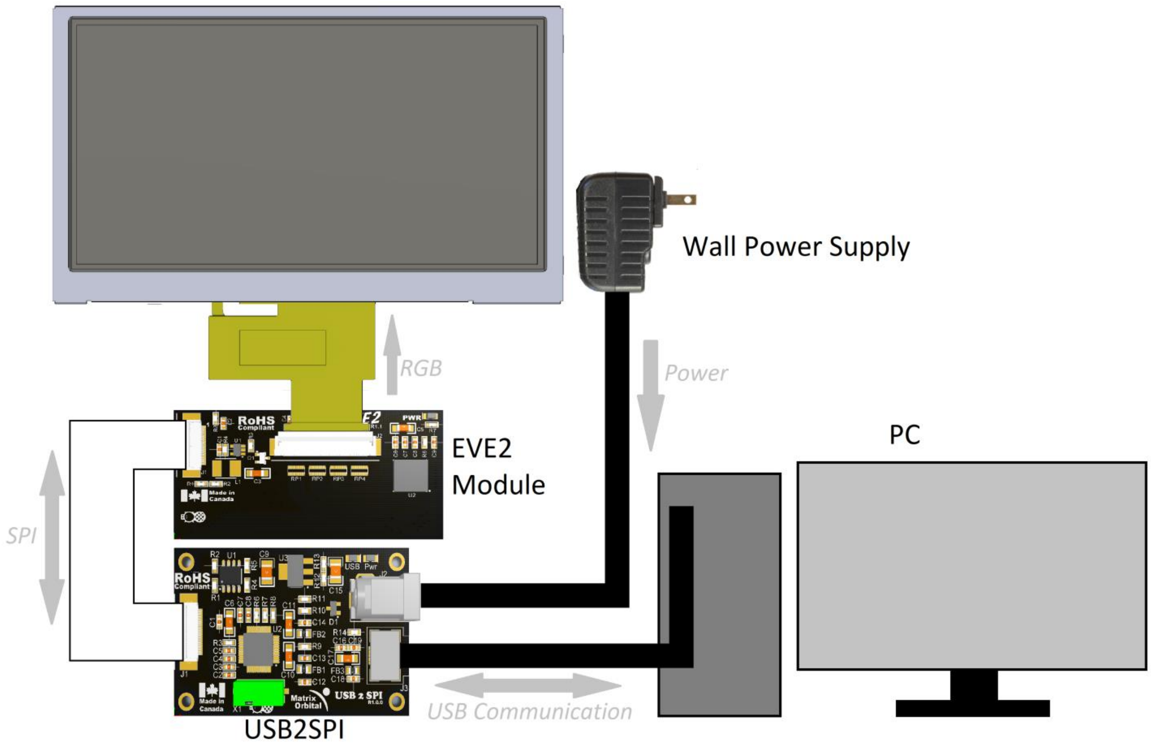


Figure 1: EVE2 Module Drawing Primitives hardware connections.

Table 1: Parts used in the Drawing Primitives 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 primitive commands are sent to 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.

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 many 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. A calibration sequence will then be called, and the user will be prompted to calibrate the display's touch screen. Once calibrated, the EVE2 module will proceed by drawing the Matrix Orbital logo using the drawing primitives available on the EVE2. Once the logo is complete, two buttons will appear, providing the option to redraw the logo, or exit the program

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.

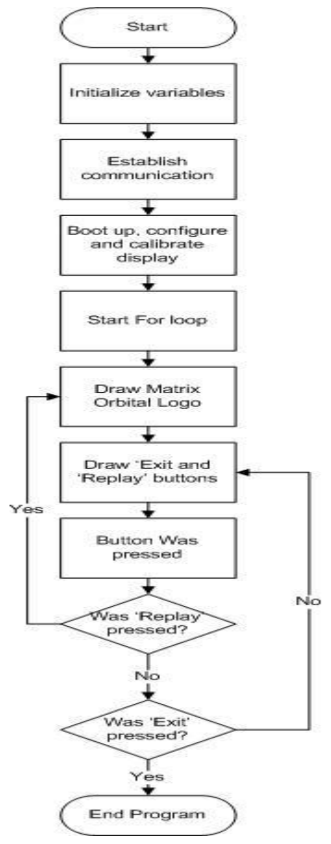
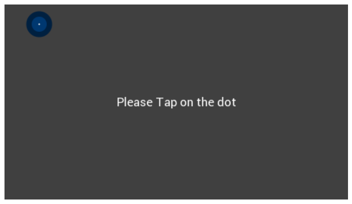


Figure 2: Drawing Primitives flow chart

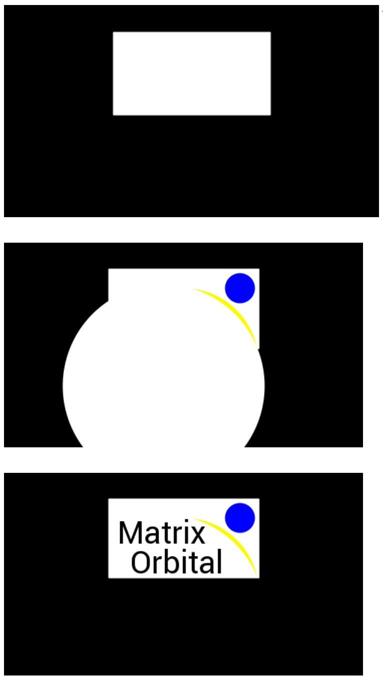
Once configured, a touch screen calibration routine will be called. In this routine, a CALIBRATE command will be sent to the EVE2 coprocessor, and three dots will be drawn on screen. The touch screen will be calibrated once all three dots have been pressed.

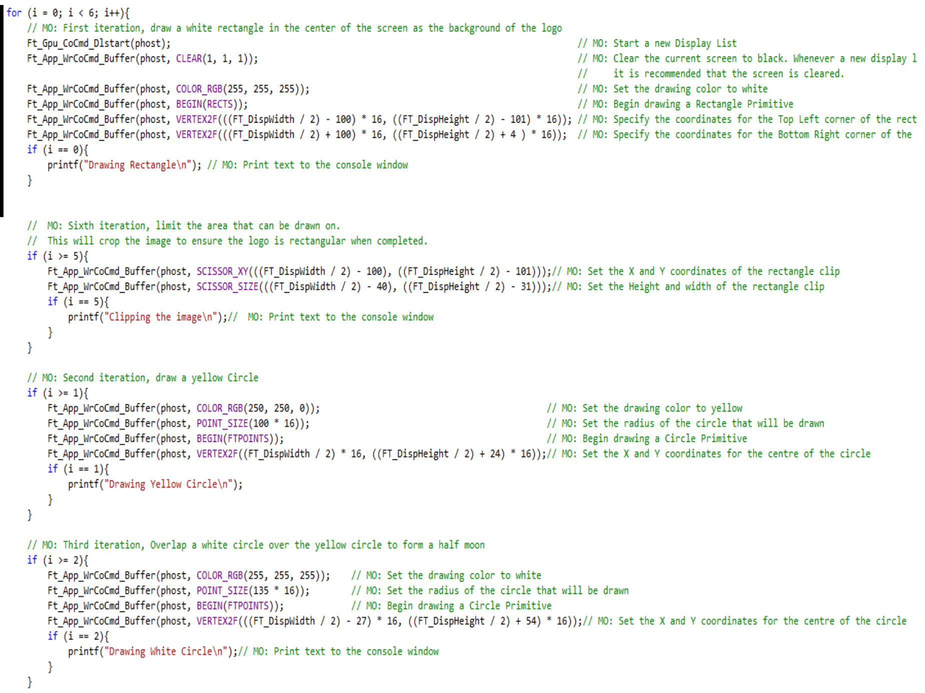


```
Ft_Gpu_CoCmd_Dlstart(phost);
Ft_App_WrCoCmd_Buffer(phost,CLEAR_COLOR_RGB(64,64,64));
Ft_App_WrCoCmd_Buffer(phost,CLEAR(1,1,1));
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(255,255,255));
Ft_Gpu_CoCmd_Text(phost,(FT_DispWidth/2), (FT_DispHeight/2), 27, OPT_CENTER, "Please Tap on the dot");
Ft_Gpu_CoCmd_Calibrate(phost, 0);

/* Download the commands into FIFIO */
Ft_App_Flush_Co_Buffer(phost);
/* Wait till coprocessor completes the operation */
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
```

Figure 3: Calibration screen

After calibration, the EVE2 will immediately run a loop, drawing primitives, recreating the Matrix Orbital logo, and generating two buttons. Each step will draw a different primitive, and the loop will continue drawing individual primitives until the logo is complete, and the two control buttons are drawn. The primitives demonstrated in the loop include rectangles, points, and text to draw the logo, scissors to crop the logo, and buttons for input.

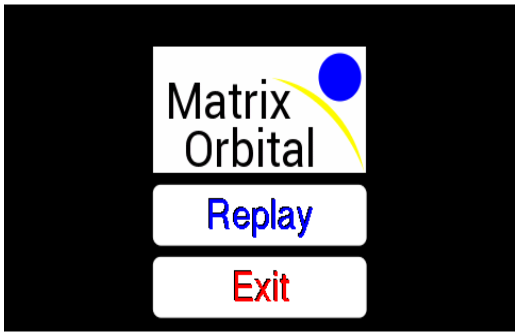


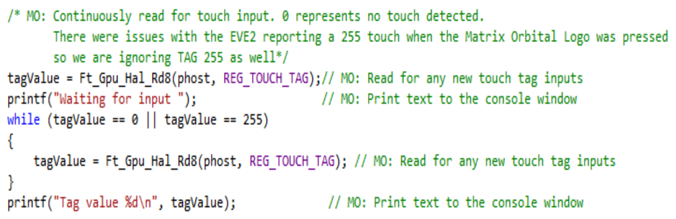


```
for (i = 0; i < 6; i++){
    // MO: First iteration, draw a white rectangle in the center of the screen as the background of the logo
    Ft_Gpu_CoCmd_Dlstart(phost);                                                                              // MO: Start a new Display List
    Ft_App_WrCoCmd_Buffer(phost, CLEAR(1, 1, 1));                                                             // MO: Clear the current screen to black. Whenever a new display l
                                                                                                              //     it is recommended that the screen is cleared.
    Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(255, 255, 255));                                                   // MO: Set the drawing color to white
    Ft_App_WrCoCmd_Buffer(phost, BEGIN(RECTS));                                                               // MO: Begin drawing a Rectangle Primitive
    Ft_App_WrCoCmd_Buffer(phost, VERTEX2F(((FT_DispWidth / 2) - 100) * 16, ((FT_DispHeight / 2) - 101) * 16)); // MO: Specify the coordinates for the Top Left corner of the rect
    Ft_App_WrCoCmd_Buffer(phost, VERTEX2F(((FT_DispWidth / 2) + 100) * 16, ((FT_DispHeight / 2) + 4 ) * 16));  // MO: Specify the coordinates for the Bottom Right corner of the
    if (i == 0){
        printf("Drawing Rectangle\n"); // MO: Print text to the console window
    }


    //  MO: Sixth iteration, limit the area that can be drawn on.
    //  This will crop the image to ensure the logo is rectangular when completed.
    if (i >= 5){
        Ft_App_WrCoCmd_Buffer(phost, SCISSOR_XY(((FT_DispWidth / 2) - 100), ((FT_DispHeight / 2) - 101)));// MO: Set the X and Y coordinates of the rectangle clip
        Ft_App_WrCoCmd_Buffer(phost, SCISSOR_SIZE(((FT_DispWidth / 2) - 40), ((FT_DispHeight / 2) - 31)));// MO: Set the Height and width of the rectangle clip
        if (i == 5){
            printf("Clipping the image\n");//  MO: Print text to the console window
        }
    }

    // MO: Second iteration, draw a yellow Circle
    if (i >= 1){
        Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(250, 250, 0));                                             // MO: Set the drawing color to yellow
        Ft_App_WrCoCmd_Buffer(phost, POINT_SIZE(100 * 16));                                               // MO: Set the radius of the circle that will be drawn
        Ft_App_WrCoCmd_Buffer(phost, BEGIN(FTPOINTS));                                                    // MO: Begin drawing a Circle Primitive
        Ft_App_WrCoCmd_Buffer(phost, VERTEX2F((FT_DispWidth / 2) * 16, ((FT_DispHeight / 2) + 24) * 16));// MO: Set the X and Y coordinates for the centre of the circle
        if (i == 1){
            printf("Drawing Yellow Circle\n");
        }
    }

    // MO: Third iteration, Overlap a white circle over the yellow circle to form a half moon
    if (i >= 2){
        Ft_App_WrCoCmd_Buffer(phost, COLOR_RGB(255, 255, 255));    // MO: Set the drawing color to white
        Ft_App_WrCoCmd_Buffer(phost, POINT_SIZE(135 * 16));        // MO: Set the radius of the circle that will be drawn
        Ft_App_WrCoCmd_Buffer(phost, BEGIN(FTPOINTS));             // MO: Begin drawing a Circle Primitive
        Ft_App_WrCoCmd_Buffer(phost, VERTEX2F(((FT_DispWidth / 2) - 27) * 16, ((FT_DispHeight / 2) + 54) * 16));// MO: Set the X and Y coordinates for the centre of the circle
        if (i == 2){
            printf("Drawing White Circle\n");// MO: Print text to the console window
        }
    }
```

Figure 4: Drawing the Matrix Orbital Logo

Once the logo is complete, a while loop will begin. The while loop will continuously run, reading for input from the user. If the “Replay” button is pressed, the EVE2 will redraw the logo. If the “Exit” button is pressed, the program will exit the loop and the display will be cleared.





```
/* MO: Continuously read for touch input. 0 represents no touch detected.
      There were issues with the EVE2 reporting a 255 touch when the Matrix Orbital Logo was pressed
      so we are ignoring TAG 255 as well*/
tagValue = Ft_Gpu_Hal_Rd8(phost, REG_TOUCH_TAG);// MO: Read for any new touch tag inputs
printf("Waiting for input ");                  // MO: Print text to the console window
while (tagValue == 0 || tagValue == 255)
{
    tagValue = Ft_Gpu_Hal_Rd8(phost, REG_TOUCH_TAG); // MO: Read for any new touch tag inputs
}
printf("Tag value %d\n", tagValue);             // MO: Print text to the console window
```

Figure 5: Completed Matrix Orbital Logo

Conclusion

This example demonstrates how one can use the drawing primitives to design simple images and assets. In addition, simple 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.

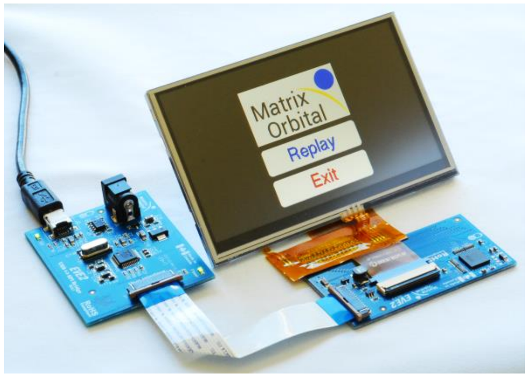


Figure 6: Screenshot of the program during operation

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	Support	Online
Phone: 403.229.2737	Phone: 403.204.3750	Purchasing: www.matrixorbital.com
Email: sales@matrixorbital.ca	Email: support@matrixorbital.ca	Support: www.matrixorbital.ca