



Application Note

BRT_AN_025

EVE Portable MCU Library

Version 1.0

Issue Date: 30-04-2024

This application note accompanies an example EVE framework which is portable across a variety of MCUs. The framework is based around the PIC MCU files provided in BRT_AN_008 and has been re-structured to ease porting to different MCUs. The accompanying code package is intended to help customers create their own libraries for their EVE applications. It includes projects for the following platforms but can also be ported to other MCUs.

FT9xx	PIC18F	STM32
ESP32	MSP430	BeagleBone
Raspberry Pi	MM2040EV (RPI Pico)	MPSSE (USB-SPI)
	IDM2040 (RPI Pico)	

Use of Bridgetek devices in life support and/or safety applications is entirely at the user's risk, and the user agrees to defend, indemnify, and hold Bridgetek harmless from any and all damages, claims, suits, or expense resulting from such use.

Bridgetek Pte Ltd (BRT Chip)

1 Tai Seng Avenue, Tower A, #03-05, Singapore 536464

Tel: +65 6547 4827 Fax: +65 6841 6071

Web Site: <http://www.brtchip.com>

Copyright © Bridgetek Pte Ltd

Table of Contents

1	Introduction	5
1.1	Overview	5
1.2	Scope	5
2	Application Overview.....	6
2.1	Background Information	6
2.2	Prerequisites	6
2.3	Software Layers	7
2.4	Folder Structure	8
3	Main Application	10
3.1	Main.c.....	10
3.2	EVE_Example.c.....	10
3.2.1	Example Provided with BRT_AN_025	11
3.2.2	Simplified Example.....	13
3.2.3	EVE_calibrate.c.....	15
3.2.4	EVE_fonts.c.....	16
3.2.5	EVE_images.c.....	18
3.2.6	EVE_Helper.c	20
4	Creating your Own Application	21
4.1	Simple Button and Gauge Example.....	21
4.2	Using your EVE Module (Display Settings).....	23
4.2.1	Set the EVE device.....	23
4.2.2	Set the display resolution.....	23
4.2.3	Enable/Disable QSPI.....	24
4.3	Porting to Other MCUs.....	24
4.4	Examples for BRT_AN_025.....	24
5	Library Layers.....	25
5.1	EVE_API Library Layer	25
5.1.1	Initialising EVE	25
5.1.2	Co-Processor Helpers	26
5.1.3	Creating screens and executing commands	26
5.1.4	Writing RAM_G and RAM_CMD.....	28
5.1.5	Writing DL Instructions and Co-Processor Commands.....	29

5.2 EVE_HAL Library Layer	30
5.2.1 EVE SPI Message Structure	30
5.2.2 EVE Supporting Functions	32
5.3 MCU Layer	32
5.3.1 Preprocessor Symbol	32
5.3.2 MCU Initialisation	33
5.3.3 MCU SPI Functions	33
6 Downloading the Code from GitHub	34
7 Setting up the MCU	35
7.1 FT900 MCU Set-up	35
7.1.1 Hardware	35
7.1.2 Software Tools	35
7.1.3 Code	36
7.2 PIC18F MCU Set-up	39
7.2.1 Hardware	39
7.2.2 Software Tools	39
7.2.3 Code	40
7.3 STM32F0 MCU Set-up	41
7.3.1 Hardware	41
7.3.2 Software Tools	42
7.3.3 Code	42
7.4 ESP32 MCU Set-up	43
7.4.1 Hardware	43
7.4.2 Software Tools	44
7.4.3 Code	45
7.5 MSP430 MCU Set-up	46
7.5.1 Hardware	46
7.5.2 Software Tools	47
7.5.3 Code	47
7.6 BeagleBone Black Set-up	48
7.6.1 Hardware	48
7.6.2 Code	49
7.6.3 Running the example	49
7.7 Raspberry Pi Set-up	51

7.7.1	Hardware.....	51
7.7.2	Code	52
7.7.3	Running the example	52
7.8	Raspberry Pi Pico Set-up	53
7.8.1	Hardware.....	53
7.8.2	Running the example	54
7.9	MPSSE Set-up (USB-SPI with Visual Studio code)	57
7.9.1	Hardware.....	57
7.9.2	Software Tools	58
7.9.3	Code	58
8	Conclusion	61
9	Contact Information	62
Appendix A–	References	63
	Document References	63
	Acronyms and Abbreviations.....	63
Appendix B.....		64
Appendix C –	List of Tables & Figures	66
	List of Figures	66
Appendix D –	Revision History	67

1 Introduction

1.1 Overview

This application note shows how to create an EVE library which is portable across a variety of MCUs and host platforms.

It is based around a common set of layers provided as C and Header files which have been arranged to ease portability. Each MCU also has its own source file which includes any MCU-specific items. This allows developers to easily port the code to their chosen MCU type. Full code projects are provided for a range of MCUs and additional types of MCU can be targeted using the same principles explained here.

In addition to the example library framework, the code also includes a simple demo application which uses touch, custom fonts, and an image.

1.2 Scope

This application note builds upon the framework described in earlier application notes in the EVE for MCUs series including [BRT_AN_008](#) and explains how it can be ported to other MCU platforms in addition to the PIC MCU. It focuses on the aspects of making the library easy to use and demonstrating how it can be run on different MCUs.

This document covers the following topics:

- The structure of the library
- The different library layers which are common to all platforms.
- The basic main sample application provided with the library
- How to modify the settings and the main example code to produce your own application
- A separate chapter for each of the target MCUs which contains the GitHub download link and describes the hardware requirements and any MCU-specific considerations.

Note 1: This code is intended to act as a starting point for customers to create their own application rather than being a complete library package. It is necessary that developers of the final application incorporating this library review all layers of the code as part of their product validation. By using any part of this code, the customer agrees to accept full responsibility for ensuring that their final product operates correctly and complies with any operational and safety requirements and accepts full responsibility for any consequences resulting from its use.

Note 2: The library functions are intended to perform a basic set-up of the MCU so that the EVE functionality can be demonstrated. The reader must consult the product documentation provided by the manufacturer of their selected MCU, and the Bridgetek documentation for their EVE device, to confirm that their final code and hardware complies with all recommendations, best practices, and specifications, so that reliable operation of the final product can be assured. The information provided in this document and code is not intended to override any information or specifications in the product datasheets.

Note 3: This document contains links to external sites and software products which are not affiliated with Bridgetek. Bridgetek accept no responsibility whatsoever for any content from 3rd party providers, including the websites, documents, and utilities, or for any consequences of using the 3rd party tools and procedures.

2 Application Overview

2.1 Background Information

When developing a framework for your MCU based on BRT_AN_025, it is recommended to use the code versions provided on GitHub (see section 6 Downloading the Code from GitHub for download links) as these are the latest versions.

The library provided here is based upon the code from several earlier application notes in the series which can be used as a background reference on how the different layers of the code in this application note work.

- [BRT_AN_006](#)

Explains the low-level SPI transfers used and the different interfaces to transfer data to/from EVE (including RAM_DL, RAM_CMD, RAM_G, Registers). This includes the data formatting used in the SPI transfers for addressing and data and the operation of the co-processor FIFO and display list and how these lead to the final screen content.

- [BRT_AN_008](#)

Shows how to create a library framework for an MCU using principles discussed in BRT_AN_006. This application note takes the basic SPI transfers detailed in BRT_AN_006 and adds an additional layer on top to allow the main application to use more user-friendly commands like the EVE Programmers guide.

- [BRT_AN_014](#)

Provides examples of using the framework created in BRT_AN_008 to perform common tasks such as loading images and fonts, loading compressed images, taking screenshots, using touch and tracking, and optimising screen updates.

The framework presented in this application note has been re-structured from BRT_AN_008 to maximise portability across different MCUs and development tools, as well as adding support for the BT81x and many other improvements.

2.2 Prerequisites

This code can be used on a wide range of MCUs. Some key requirements for the MCU are listed below:

- The MCU requires a Single or Quad SPI Master with SPI Mode 0 capability.
- SPI signals used are SCK and MOSI/MISO (or four bi-directional lines for Quad SPI)
- GPIO lines are used for Chip Select and for Power Down inputs to the EVE device
- The code is designed for SPI master routines which can read/write a single byte at a time and have manual control over chip select. This is required to meet the EVE SPI protocol. The provided examples would require significant modification if the MCU uses a SPI API library which sends a buffer of bytes (such as via a DMA transfer) with automatic chip select control and this is out of the scope of this document and sample. Most MCUs can however be programmed at a level which interacts directly with the SPI hardware registers and GPIO for chip select.

The application note includes a code project containing the example framework and sample main application for each of the following MCUs. However, the code can be ported to other MCUs in a similar way by referring to the provided examples.

- [Bridgetek FT900](#)
- Microchip PIC18F
- ST Micro STM32F0
- Espressif ESP32
- TI MSP430
- BeagleBone
- Raspberry Pi
- Raspberry Pi Pico (see Note below)
- MPSSE (USB-SPI using [MPSSE adapters](#) with Visual Studio code) (external link to FTDI)

Note: Bridgetek have the following modules which are based upon the Raspberry Pi Pico / RP2040

- [Bridgetek MM2040EV](#) has an on-board Raspberry Pi Pico, with 10-way and 16-way connectors. This module can attach directly to Bridgetek EVE modules such as ME817EV and VM816C50A-D
- [Bridgetek IDM2040-7A](#) has an on-board RP2040 microcontroller and Bridgetek BT817 EVE Graphics controller with a 7" capacitive touch screen, all in one module

The code project can be used on the following EVE devices by setting the associated device type in the EVE_config.h file (see section 4.2)

- EVE Generation 1 FT800, FT801
- EVE Generation 2 BT880, BT881, BT882, BT883
- EVE Generation 2 FT810, FT811, FT812, FT813
- EVE Generation 3 BT815, BT816
- EVE Generation 4 BT817, BT818

2.3 Software Layers

The software consists of several layers which are shown below. The different layers are discussed in greater detail in the following sections of this document.

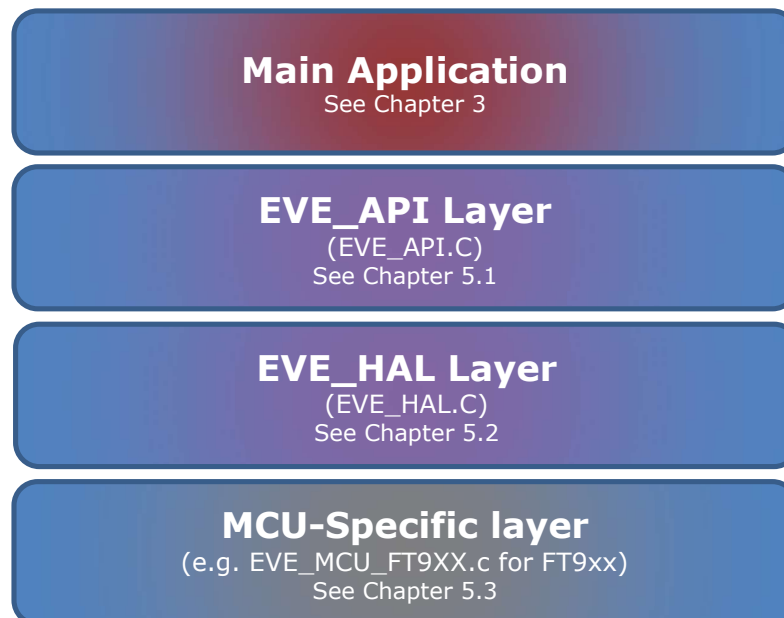


Figure 1 - Layers of the software example

2.4 Folder Structure

The example for each platform uses the files shown below. These include files which are common and can be used across a variety of MCUs and some MCU-specific files.

Filename starting with `EVE_MCU_` indicate files which are MCU-specific and contain the different lower-level functions to talk to the MCU hardware and account for most differences allowing the upper levels to be common. Note that due to MCU and compiler differences, it is possible that even the common files may need to be modified to be used on some other platforms not covered in this document.

The provided code file for each MCU platform (available from GitHub via pull request or downloadable as a zip) contains the full project including all source and header files. The library files can also be ported over to other MCUs. The [BRT_AN_062 porting guide](#) has an example of porting to the NXP K64 which can be used as a reference for this. See section 4.3 for details of the porting guide.

All image and font data needed for the simple example provided with the code package is included in the header files already and will be stored in the MCU's memory and so no separate SD card is required to hold image files etc. If using an MCU with limited Flash, using the BT815 / BT816 / BT817 / BT818 with a flash IC attached would allow all assets to be stored in EVE-attached Flash instead of the MCU and so would save space in the MCU. See the Examples for BRT_AN_025 in section 4.4 for details.

Common Library Files

- `\source\EVE_API.c`
- `\source\EVE_HAL.c` for MCU platforms
- `\source\EVE_HAL_Linux.c` for BeagleBone / RPi platforms
- `\include\FT8xx.h`
- `\include\HAL.h`
- `\include\EVE.h`
- `\include\Platform.h`
- `\include\EVE_Config.h`
- `\include\MCU.h`

Common Application Files

- `\examples\simple\common\eve_calibrate.c`
- `\examples\simple\common\eve_example.c`
- `\examples\simple\common\eve_fonts.c`
- `\examples\simple\common\eve_helper.c`
- `\examples\simple\common\eve_images.c`
- `\examples\simple\common\eve_example.h`

MCU-specific Files

- `\ports\eve_arch_ft9xx\EVE_MCU_FT9XX.c` for FT9xx
- `\ports\eve_arch_pic\EVE_MCU_PIC.c` for PIC18F
- `\ports\eve_arch_stm32\EVE_MCU_STM32.c` for STM32
- `\ports\eve_arch_esp32\EVE_MCU_ESP32.c` for ESP32
- `\ports\eve_arch_msp430\EVE_MCU_MSP430.c` for MSP430
- `\ports\eve_arch_beaglebone\EVE_Linux_BBB.c` for BeagleBone
- `\ports\eve_arch_rpi\EVE_Linux_RPi.c` for RPi
- `\ports\eve_arch_rpi\EVE_MCU_RP2040.c` for IDM2040-7A/RP2040/Pico
- `\ports\ eve_libmpsse\EVE_libmpsse.c` for MPSSE
- `\examples\simple\[MCU specific folder]\main.c` Part of each MCU project

The folder is structured as shown below:

EVE-MCU-BRT_AN_025

```
↳ examples
  ↳ simple
    ↳ common
      ↳ eve_calibrate.c
      ↳ eve_example.c
      ↳ eve_example.h
      ↳ eve_fonts.c
      ↳ eve_helper.c
      ↳ eve_images.c
    ↳ ESP32
      ↳ [toolchain project files for this platform]
    ↳ ft900
      ↳ [toolchain project files for this platform]
    ↳ Libmpsse
      ↳ [toolchain project files for this platform]
    ↳ PIC18F
      ↳ [toolchain project files for this platform]
    ↳ pico
      ↳ [toolchain project files for this platform]
    ↳ raspberrypi
      ↳ [toolchain project files for this platform]
    ↳ STM32
      ↳ [toolchain project files for this platform]
  ↳ include
    ↳ EVE.h
    ↳ EVE_config.h
    ↳ FT8xx.h
    ↳ HAL.h
    ↳ MCU.h
    ↳ Platform.h
  ↳ ports
    ↳ eve_arch_ft9xx
      ↳ EVE_MCU_FT9XX.c
    ↳ eve_arch_pic
      ↳ EVE_MCU_PIC.c
    ↳ eve_arch_stm32
      ↳ EVE_MCU_STM32.c
    ↳ eve_arch_esp32
      ↳ EVE_MCU_ESP32.c
    ↳ eve_arch_msp430
      ↳ EVE_MCU_MSP430.c
    ↳ eve_arch_beaglebone
      ↳ EVE_Linux_BBB.c
    ↳ eve_arch_rpi
      ↳ EVE_Linux_RPi.c
      ↳ EVE_MCU_RP2040.c
    ↳ eve_libmpsse
      ↳ EVE_libmpsse.c
  ↳ source
    ↳ EVE_API.c
    ↳ EVE_HAL.c
    ↳ EVE_HAL_Linux.c
```

3 Main Application

This section describes the general operation of the main demo application provided. The individual sections for each MCU (see Sections 7 to 7.9) discuss any specifics to the associated MCU.

The application displays a logo along with some text at the top of the screen. A counter is also displayed in a custom font (DS-DIGIT). The touch feature is used to detect user presses in the area of the counter and increment the counter.

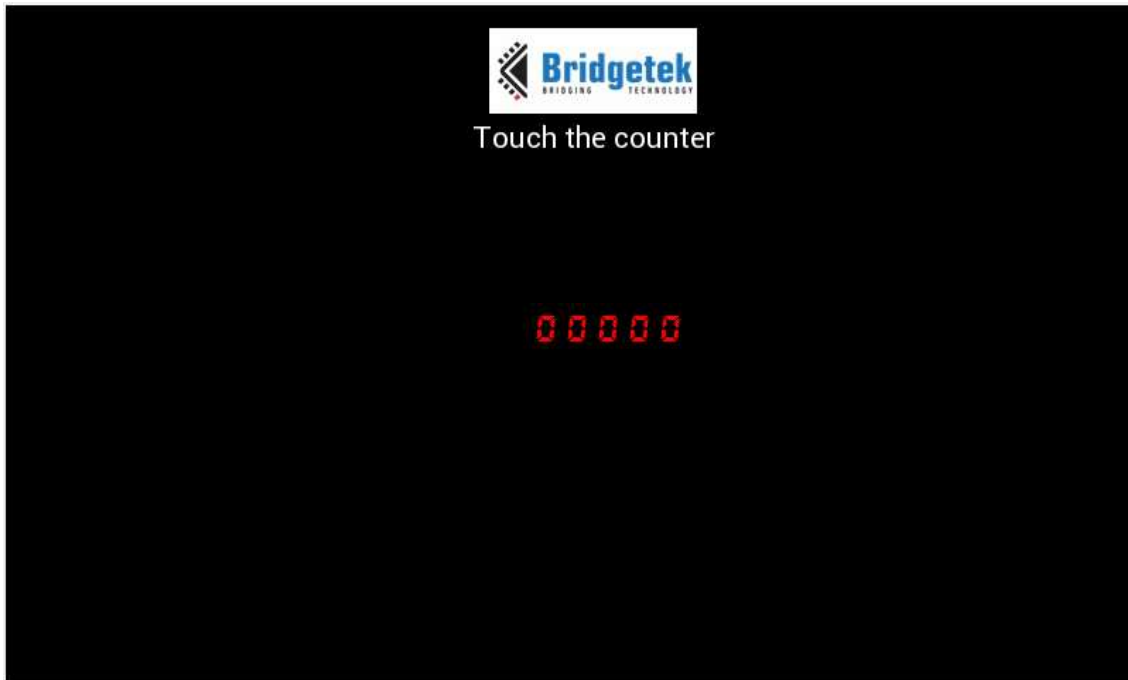


Figure 2 – Main application screen

3.1 Main.c

The application starts up in the file main.c which provides initial MCU configuration and then calls eve_example.c where the remainder of the application will be carried out.

3.2 EVE_Example.c

Function eve_example.c calls EVE_Init() to provide the rest of the EVE configuration.

```
void eve_example(void)
{
    uint32_t font_end;

    // Initialise the display
    EVE_Init();

    // Calibrate the display
    eve_calibrate();

    // Load fonts and images
    font_end = eve_init_fonts();

    eve_load_images(font_end);

    // Start example code
    eve_display();
}
```

```
}
```

Function `eve_calibrate()` is then called which uses the calibration co-processor command to display the calibration screen and asks the user to tap the three dots (see [eve_calibrate.c](#) below).

Once calibration is complete, the font for the counter and the image for the logo are both loaded into the EVE device's RAM_G. This is done within `eve_init_fonts` and `eve_load_images`. (see [eve_fonts.c](#) and [eve_images.c](#) below).

Finally, the main program sits in a continuous loop within `eve_display()`. Each time round the loop, a screen is created using a co-processor list.

3.2.1 Example Provided with BRT_AN_025

This begins in the usual way by starting a new co-processor list, followed by setting a colour and clearing the screen to this colour. This is highlighted in **green** text. The information in chapters 5.1.2 - 5.1.5 have additional guidance on using the co-processor functions and loading data to RAM_G etc.

```
void eve_display(void)
{
    uint32_t counter = 0;
    uint8_t key;
    int8_t i;
    uint32_t units;

    do {
        // Comment this line if the counter needs to increment continuously.
        // Uncomment and it will increment by one each press.
        // while (eve_read_tag(&key) != 0);

        EVE_LIB_BeginCoProList();           // CS low and send address
        EVE_CMD_DLSTART();                  // Sets REG_CMD_DL to 0
        EVE_CLEAR_COLOR_RGB(0, 0, 0);
        EVE_CLEAR(1,1,1);
    }
```

Then, the bitmap of the Bridgetek logo which had been loaded earlier is displayed. The RGB colour is set to white to show the bitmap in its original colours. Some text is also added below the bitmap to inform the user to touch the counter to make it increment.

The FT81x and later (FT81x/BT88x/BT81x) can use the `VERTEX_TRANSLATE` command to offset the coordinates to reach the centre of the screen, as the `VERTEX2II` used to display bitmap or text cells is limited to coordinates of 511. The FT80x uses a smaller screen (e.g. 480x272) and so does not require the `VERTEX_TRANSLATE` command. The FT80x does not support this command and so it is only used when FT81x / BT88x / BT81x series devices are selected.

Note that alternatively the `VERTEX2F` (which can reach beyond coordinate 511) can be used. In this case it would be preceded by the `CELL()` command which will select which bitmap cell is to be displayed in a similar way to the `VERTEX2II`.

```
        EVE_COLOR_RGB(255, 255, 255);

        EVE_BEGIN(EVE_BEGIN_BITMAPS);

        #if (defined EVE2_ENABLE || defined EVE3_ENABLE || defined EVE4_ENABLE)
            // Set origin on canvas using EVE_VERTEX_TRANSLATE.
            EVE_VERTEX_TRANSLATE_X(((EVE_DISP_WIDTH/2)-
                (eve_img_bridgetek_logo_width/2)) * 16);
            EVE_VERTEX2II(0, 0, BITMAP_BRIDGETEK_LOGO, 0);
            EVE_VERTEX_TRANSLATE_X(0);
        #else
            // Place directly on canvas EVE_VERTEX_TRANSLATE not available.
            EVE_VERTEX2II((EVE_DISP_WIDTH/2)-(eve_img_bridgetek_logo_width/2),
```

```
0,BITMAP_BRIDGETEK_LOGO, 0);  
#endif
```

A tag of 100 is then assigned and will result in any following items being tagged with tag 100. A VERTEX2II command is used here so that bitmap cells can be used. Each character of the converted font is referred to by its index within the bitmap handle. Since VERTEX2II can only go up to 511, Vertex Translate is used to allow the position to exceed this on the 800x480 screen.

Vertex Translate adds an offset to any subsequent VERTEX commands in the command list, allowing them to be shifted on the screen.

The calculation uses the FontWidthInPixels value from the Metric Block produced when converting fonts. The font conversion with EVE Asset Builder produces the output data file with a 148-byte metric block followed by the font glyph data which contains the bitmap of the font characters.

The metric block contains details on the format and height/width/stride of the font characters, as well as an array of values containing the width of each individual character (see the start of section 3.2.4 for an example) . The individual character widths allow the fonts to be proportionally spaced for applications displaying text strings etc. This helps improve the appearance as some characters such as the number 1 are narrower than others.

In this case, we want the characters to remain at the same spacing regardless of their width so that the counter does not move left and right depending on the numbers displayed in the counter and so we have used the overall FontWidthInPixels value rather than the width of each individual character.

```
EVE_TAG(100);  
EVE_COLOR_RGB(255, 0, 0);  
EVE_BEGIN(EVE_BEGIN_BITMAPS);  
units = 1;  
  
#if (defined EVE2_ENABLE || defined EVE3_ENABLE || defined EVE4_ENABLE)  
EVE_VERTEX_TRANSLATE_Y((EVE_DISP_HEIGHT / 2) * 16);  
for (i = 0; i < 5; i++)  
{  
    EVE_VERTEX_TRANSLATE_X(((EVE_DISP_WIDTH  
        - (font0_hdr->FontWidthInPixels * 5)) / 2)  
        - (font0_hdr->FontWidthInPixels) + (font0_hdr->  
            >FontWidthInPixels * (5 - i))) * 16);  
    EVE_VERTEX2II(0, 0, FONT_CUSTOM, ((counter / units) % 10)+1);  
    // +1 as in the converted font the number '0' is in position 1  
    in the Font table  
    units *= 10;  
}  
#else  
for (i = 0; i < 5; i++)  
{  
    EVE_VERTEX2II(((EVE_DISP_WIDTH  
        - (font0_hdr->FontWidthInPixels * 5)  
        / 2) - (font0_hdr->FontWidthInPixels) +  
        (font0_hdr->FontWidthInPixels * (5 - i))),  
        (EVE_DISP_HEIGHT / 2), FONT_CUSTOM, ((counter / units)  
        % 10)+1);  
    //+1 as in the converted font the number '0' is in position 1  
    in the font table  
    units *= 10;  
}  
#endif
```

The list finishes with the DISPLAY and SWAP commands. In the EVE_LIB_EndCoProList() call, the CS# line is brought high to finish the burst write and the REG_CMD_WRITE pointer is moved to the end of the new list. The EVE_LIB_AwaitCoProEmpty() then awaits the completion of the commands being executed by the co-processor.

```
EVE_DISPLAY();  
EVE_CMD_SWAP();
```

```
EVE_LIB_EndCoProList();
EVE_LIB_AwaitCoProEmpty();
```

The code then checks the tag register to see if any touch had occurred on the counter area (tag 100). If the counter is being touched, a variable increments and will be drawn with the new value next time round the loop.

```
while (eve_read_tag(&key) == 0);

if (key == 100)
{
    counter++;
    if (counter == 100000)
    {
        counter = 0;
    }
}
} while (1);
}
```

3.2.2 Simplified Example

Note that the example above, which is provided with the BRT_AN_025 code, prints the counter by placing each digit individually. This avoids any shifting of the counter by placing each digit at a fixed coordinate regardless of character width.

It is also possible to display a similar counter using the CMD_NUMBER as illustrated below. This requires that the font is converted with the first ASCII character set for ANSI character 48 in EVE Asset Builder, which is the offset of the numbers in the ANSI character set. The SETFONT2 command is used as this supports setting the first character (FT81x or later only). The font is assigned handle 5 in the example below.

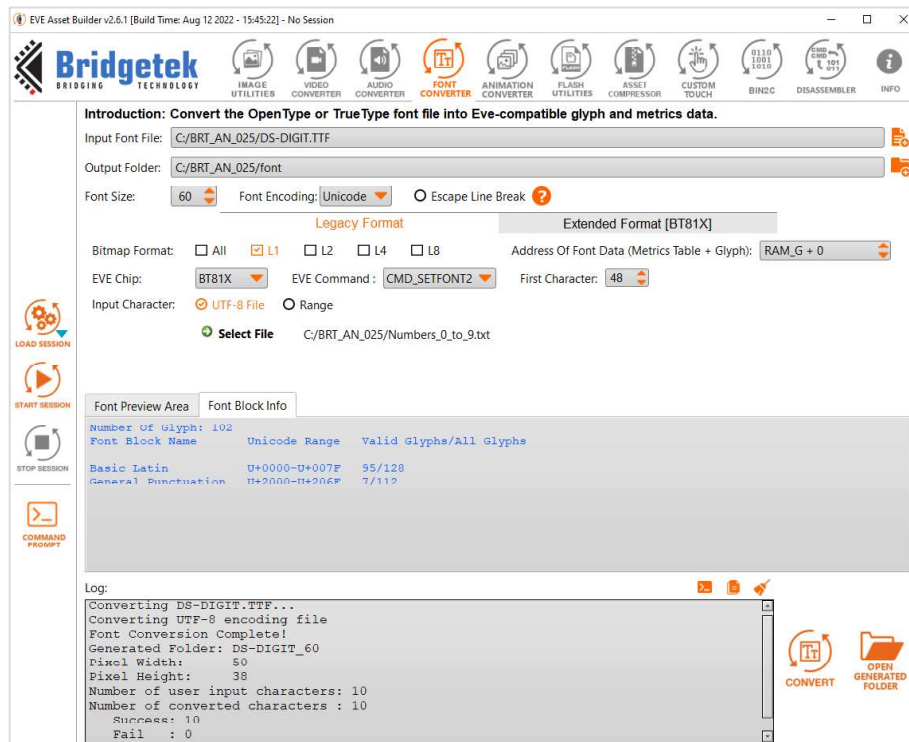


Figure 3 – Converting the DIGITS font for the simplified example


```
#if defined (EVE2_ENABLE) || defined (EVE3_ENABLE) || defined (EVE4_ENABLE)
    EVE_CMD_SETFONT2(5,0,48); // Set the digits font as handle 5
                                // with first character ANSI 48 (number 0)
    EVE_CMD_NUMBER(EVE_DISP_WIDTH/2, 300, 5, EVE_OPT_CENTERX | 5,
        counter); // Print the counter as a number
#else
    EVE_CMD_NUMBER(EVE_DISP_WIDTH/2, 300, 30, EVE_OPT_CENTERX | 5,
        counter); // Print in font 30 if using FT800
#endif

EVE_TAG_MASK(0); // Disable tagging for any items after this

EVE_DISPLAY();
EVE_CMD_SWAP();
EVE_LIB_EndCoProList();
EVE_LIB_AwaitCoProEmpty();

// wait for the user to touch the screen (tag becomes non-zero)
while (eve_read_tag(&key) == 0);

if (key == 10) // If their touch is on the counter (tagged 10)
{
    counter++; // then increment the counter
    if (counter == 100000) // Roll over at 99999
    {
        counter = 0;
    }
} while (1);
}
```

3.2.3 EVE_calibrate.c

This function provides the screen calibration for EVE. It calls the Calibrate command and will block until the user has tapped the three dots to allow EVE to calibrate the touch relative to the screen.

The code in purple below does the actual calibration and EVE automatically stores the resulting six transform values in the REG_TOUCH_TRANSFORM_A to _F registers in EVE. This calibration data is held in volatile memory in EVE and will not be retained across successive power-ups.

```
void eve_calibrate(void)
{
    struct touchscreen_calibration calib;
    uint8_t dummy;

    platform_calib_init();

    // If no store of calibration or current screen touch.
    if ((platform_calib_read(&calib) != 0) || (eve_read_tag(&dummy)))
    {
        // Wait for end of touch.
        while (eve_read_tag(&dummy));

        EVE_LIB_BeginCoProList();
        EVE_CMD_DLSTART();
        EVE_CLEAR_COLOR_RGB(0, 0, 0);
        EVE_CLEAR(1,1,1);
        EVE_COLOR_RGB(255, 255, 255);
        EVE_CMD_TEXT(EVE_DISP_WIDTH/2, EVE_DISP_HEIGHT/2,
            28, EVE_OPT_CENTERX | EVE_OPT_CENTERY, "Please tap on the dots");
        EVE_CMD_CALIBRATE(0);
        EVE_LIB_EndCoProList();
        EVE_LIB_AwaitCoProEmpty();

        calib.transform[0] = HAL_MemRead32(EVE_REG_TOUCH_TRANSFORM_A);
        calib.transform[1] = HAL_MemRead32(EVE_REG_TOUCH_TRANSFORM_B);
        calib.transform[2] = HAL_MemRead32(EVE_REG_TOUCH_TRANSFORM_C);
    }
}
```

```
    calib.transform[3] = HAL_MemRead32(EVE_REG_TOUCH_TRANSFORM_D);
    calib.transform[4] = HAL_MemRead32(EVE_REG_TOUCH_TRANSFORM_E);
    calib.transform[5] = HAL_MemRead32(EVE_REG_TOUCH_TRANSFORM_F);
    platform_calib_write(&calib);
}
Else
{
    HAL_MemWrite32(EVE_REG_TOUCH_TRANSFORM_A, calib.transform[0]);
    HAL_MemWrite32(EVE_REG_TOUCH_TRANSFORM_B, calib.transform[1]);
    HAL_MemWrite32(EVE_REG_TOUCH_TRANSFORM_C, calib.transform[2]);
    HAL_MemWrite32(EVE_REG_TOUCH_TRANSFORM_D, calib.transform[3]);
    HAL_MemWrite32(EVE_REG_TOUCH_TRANSFORM_E, calib.transform[4]);
    HAL_MemWrite32(EVE_REG_TOUCH_TRANSFORM_F, calib.transform[5]);
}
}
```

The FT900, ESP32, BeagleBone Black and Raspberry Pi examples also include code to store the values of the six touch transform registers (REG_TOUCH_TRANSFORM_A - REG_TOUCH_TRANSFORM_F) after the initial calibration within the MCU's non-volatile memory and recall them on start-up. This technique may also be used for other MCUs by using their corresponding EEPROM/Data Flash programming APIs as described by the MCU manufacturer's documentation.

The functions to do so for the MCU can be found in main.c and are shown below. In these cases the functions would contain the MCU-specific code for storing variables to its EEPROM/Data flash.

Note: On the platforms which store the calibration, the IF statement at the beginning of the code shown above will run calibration if either the calibration is not yet stored (e.g. on the first run of the application after programming the MCU) or if the user holds a touch on the screen during start-up. The latter case allows the user to force a re-calibration if required. If calibration values have been stored previously and if no touch is present, the stored values will be loaded.

In the platforms which support storing the calibration data, to force a re-calibration, touch the screen and hold the touch. Reset the MCU (or power-up the MCU/EVE if presently unpowered) whilst still holding the touch. After exiting reset/Power up, lift your finger from the screen and the calibration will begin.

In the other MCUs used here, the values are not stored, and calibration is carried out on each start-up. In these cases, the application can return dummy values as shown below in the following function calls. These functions can be found in main.c.

```
int8_t platform_calib_init(void)
{
    return 1;
}

int8_t platform_calib_write(struct touchscreen_calibration *calib)
{
    return 0;
}

int8_t platform_calib_read(struct touchscreen_calibration *calib)
{
    return -1;
}
```

3.2.4 EVE_fonts.c

This file contains the data array of the font which was produced by the Font Converter tool which is part of [EVE Asset Builder](#).

The data itself was copied from the font converter output and placed in an array as shown below. The output consists of a metric block containing properties of the font and the individual width of each character, followed by the character data itself.


```
// ([location where we load the font in RAM_G] minus 142).
// We load it at RAM_G + 1K to keep this positive (1K - 142 == 858).
// If we loaded the font at RAM_G + 0, it would have been -142.

EVE_BITMAP_SOURCE((font0_hdr->PointerToFontGraphicsData)&(0x3FFFFFF));
EVE_BITMAP_LAYOUT(font0_hdr->FontBitmapFormat,
                  font0_hdr->FontLineStride, font0_hdr->FontHeightInPixels);
EVE_BITMAP_SIZE(EVE_FILTER_NEAREST, EVE_WRAP_BORDER, EVE_WRAP_BORDER,
                font0_hdr->FontWidthInPixels,
                font0_hdr->FontHeightInPixels);
EVE_CMD_SETFONT(FONT_CUSTOM, font0_offset);
EVE_END();
EVE_DISPLAY();
EVE_CMD_SWAP();
EVE_LIB_EndCoProList();
EVE_LIB_AwaitCoProEmpty();
```

Finally, the function returns the end address of the data in RAM_G which can be used when writing the next bitmap or font into RAM_G to avoid any overlap. As the font is in the raw format, the size can be determined easily by the size of the array of data itself. The value can also be adjusted as shown below to provide the desired alignment for the next asset starting address (e.g. 4-byte aligned, or 16-byte aligned).

```
    return ((font0_size + font0_offset) + 16) & (~15);
}
```

3.2.5 EVE_images.c

This file contains the image data of the Bridgetek logo JPEG.

The data was copied by opening the original file in a hex editor and exporting the data. This data was then pasted into an array.

```
static const uint8_t img_bridgetek_logo[] __attribute__((aligned(4))) =
{
    /*000:*/ 0xFF, 0xD8, 0xFF, 0xE0, 0x00, 0x10, 0x4A, 0x46, 0x49, 0x46, 0x00,
    0x01, 0x01, 0x00, 0x00, 0x01,
    /*010:*/ 0x00, 0x01, 0x00, 0x00, 0xFF, 0xDB, 0x00, 0x43, 0x00, 0x03, 0x02,
    0x02, 0x03, 0x02, 0x02, 0x03,
    /*020:*/ 0x03, 0x03, 0x03, 0x04, 0x03, 0x03, 0x04, 0x05, 0x08, 0x05, 0x05,
    0x04, 0x04, 0x05, 0x0A, 0x07,
```

The `eve_load_images` function is then used to decompress and store this image in RAM_G. This uses the `CMD_Loadimage` command which is sent to the co-processor. The data is written to the `RAM_CMD` buffer immediately after the command.

The parameters of the command specify the start address in RAM_G where the inflated data should be put. In this case, the address specified is at the end of the previously loaded font data. The end address for the font data was the one returned by the preceding function which loaded the font.

```
/*uint32_t eve_ui_load_image(const uint8_t __flash__ *img_data, uint32_t
start_addr, uint8_t handle, uint16_t *width, uint16_t *height)*/

uint32_t eve_load_images(uint32_t start_addr)
{
    uint8_t buf[128];
    uint32_t img_width = 0;
    uint32_t img_height = 0;
    uint32_t eve_addr = (uint32_t)img_bridgetek_logo;
    int8_t flag;
    int i;

    flag = 0;
    EVE_LIB_BeginCoProList();
```

```
EVE_CMD_LOADIMAGE(start_addr, 0);
// Send raw JPEG encoded image data to coprocessor. It will be decoded
// as the data is received.
while (flag != 2)
{
    memcpy(buf, (const void *)eve_addr, sizeof(buf));
    for (i = 0; i < sizeof(buf); i++)
    {
        if (buf[i] == 0xff)
        {
            flag = 1;
        }
        else
        {
            if (flag == 1)
            {
                if (buf[i] == 0xd9)
                {
                    flag = 2;
                    i++;
                    break;
                }
            }
            flag = 0;
        }
    }
    EVE_LIB_WriteDataToCMD(buf, (i + 3)&(~3));
    eve_addr += i;
};
EVE_LIB_EndCoProList();
EVE_LIB_AwaitCoProEmpty();
```

The GetProps command can be used to check the properties of the image which was just loaded. In particular, the width and height can be obtained which can then be used when assigning a handle and displaying the image.

It is also important to note that with compressed images, the raw data written to RAM_G by the CMD_LOADIMAGE or CMD_INFLATE command will be larger than the compressed data written to the co-processor. When using compressed files of the types of JPG, PNG (via CMD_LOADIMAGE) or compressed to .bin via the EVE asset Builder (via CMD_INFLATE), EVE does not display them direct from the PNG/JPG/BIN file. Therefore, decompressing to RAM_G is necessary before displaying the image. For this reason, unlike the font, which was loaded as raw in the previous section, the size of the data array is not the final size of the decompressed image.

The GetProps command is one of a small number of commands which return a value. When the co-processor returns a value, the result is written to the RAM_CMD buffer itself by the co-processor. The command is written by the MCU with three dummy values, and these are replaced by the co-processor with the results. A helper function is provided here which sends the command followed by three dummy values, and then looks back in the FIFO to check the result.

```
// Obtain the width and height of the expanded JPEG image.
EVE_LIB_GetProps(&eve_addr, &img_width, &img_height);

eve_img_bridgetek_logo_width = (uint16_t)img_width;
eve_img_bridgetek_logo_height = (uint16_t)img_height;
```

A co-processor list is then used to assign a bitmap handle and specify the properties of the image. This allows the main program loop to display the image easily. FT81x (EVE2) and later has additional EVE_BITMAP_LAYOUT_H and EVE_BITMAP_SIZE_H instructions to define the upper bits of the layout and size values as it supports a larger screen size.

```
EVE_LIB_BeginCoProList();
EVE_CMD_DLSTART();

EVE_BEGIN(EVE_BEGIN_BITMAPS);
```

```
EVE_BITMAP_HANDLE(BITMAP_BRIDGETEK_LOGO);
// Optional to mask the bitmap source here with 0x3FFFFFF to ensure that
// only the valid bits for addressing within RAM_G are set.
// BT81x now supports additional addressing where the source is in
// flash (see bitmap_source in the programmers guide). The mask helps to
// ensure that the address is valid if the source address is negative
// (not the case here but can happen if setting up fonts)
EVE_BITMAP_SOURCE(start_addr & 0x3FFFFFF);
EVE_BITMAP_LAYOUT(EVE_FORMAT_RGB565, img_width * 2, img_height);
#if (defined EVE2_ENABLE || defined EVE3_ENABLE || defined EVE4_ENABLE)
EVE_BITMAP_LAYOUT_H((img_width * 2) >> 10, img_height >> 9);
#endif
EVE_BITMAP_SIZE(EVE_FILTER_NEAREST, EVE_WRAP_BORDER, EVE_WRAP_BORDER,
                img_width, img_height);
#if (defined EVE2_ENABLE || defined EVE3_ENABLE || defined EVE4_ENABLE)
EVE_BITMAP_SIZE_H(img_width >> 9, img_height >> 9);
#endif
EVE_END();

EVE_DISPLAY();
EVE_CMD_SWAP();
EVE_LIB_EndCoProList();
EVE_LIB_AwaitCoProEmpty();
start_addr += ((img_width * 2) * img_height);
start_addr = (start_addr + 3) & (~3);

return start_addr;
}
```

3.2.6 EVE_Helper.c

This file contains any helper functions required by the application. In the example provided, it includes a function `eve_read_tag` which can be used to check for a touch on the screen and determine the tag of the touched area. It is used when detecting a touch on the digital counter in this demo.

```
uint8_t eve_read_tag(uint8_t *key)
{
    uint8_t Read_tag;
    uint8_t key_detect = 0;

    Read_tag = HAL_MemRead8(EVE_REG_TOUCH_TAG);

    if (!(HAL_MemRead16(EVE_REG_TOUCH_RAW_XY) & 0x8000))
    {
        key_detect = 1;
        *key = Read_tag;
    }

    return key_detect;
}
```

4 Creating your Own Application

It is recommended to try out the code with the example provided initially to check that your MCU and screen are all working correctly, if your MCU and module are directly compatible with the code. Once this is up and running, the code can then be modified to produce your final application and to match your intended EVE module. In addition, you may also wish to add support for another MCU platform if your final design will use a different MCU. This section provides guidance on how to do this.

Note The considerations in chapter 5.1.3 (Creating screens and executing commands) should be noted when creating your application, it is recommended to review these before creating the application.

4.1 Simple Button and Gauge Example

The easiest way to create an application is to edit the `eve_example.c` file.

As an illustration, the code below can be pasted into the `eve_example.c` file to replace the code there.

The code displays a numerical counter and a gauge which read from 0 to 100. Buttons marked + and - allow you to increment and decrement the counter. The button options for 3D (un-pressed appearance) and FLAT (pressed in appearance) are used to indicate when a button is pressed.

The code can be extended to interface with other peripherals on the MCU such as GPIO, PWM, serial interfaces etc. to form the real application.

The code files which form the other layers of the BRT_AN_025 example can also be ported into an existing application to add the display functionality.



Figure 4 – Creating your own application

```
#include <stdint.h>
#include "EVE.h"
#include "../include/HAL.h"
#include "MCU.h"
#include "eve_example.h"
```

```

void eve_display(void)
{
    uint8_t TagVal = 0;
    uint8_t counter = 0;
    uint16_t ButtonMinus3D = 0;
    uint16_t ButtonPlus3D = 0;

    while(1)
    {
        EVE_LIB_BeginCoProList();           // Begin new screen
        EVE_CMD_DLSTART();                   // Create new Display List
        EVE_CLEAR_COLOR_RGB(0, 0, 0);       // Specify color to clear screen to (black)
        EVE_CLEAR(1,1,1);                   // Clear color, stencil, and tag buffer

        EVE_TAG_MASK(1);                     // Enable tagging

        EVE_TAG(2);                           // Tag following items with tag 2
        EVE_CMD_FGCOLOR(0x008000);           // Deep green foreground color
        // Draw button with a "-" symbol (centered on x = 120)
        EVE_CMD_BUTTON((120-40), 250, 80, 60, 30, ButtonMinus3D, "-");

        EVE_TAG(3);                           // Tag following items with tag 3
        EVE_CMD_FGCOLOR(0x800000);           // Deep red foreground color

        // Draw button with a "+" symbol (centered on x = 280)
        EVE_CMD_BUTTON((280-40), 250, 80, 60, 30, ButtonPlus3D, "+");

        EVE_TAG_MASK(0);                     // Stop following items being tagged

        // Draw a gauge from 0-100 with needle set by 'counter'
        EVE_CMD_GAUGE(200, 100, 80, 0, 10, 1, counter, 100);
        // Display a number with the variable 'counter'
        EVE_CMD_NUMBER(200, 200, 30, EVE_OPT_CENTERX, counter);

        EVE_DISPLAY();                       // Tell EVE that this is end of list
        EVE_CMD_SWAP();                       // Swap buffers in EVE to make this list active
        EVE_LIB_EndCoProList();               // Finish the co-processor list burst write
        EVE_LIB_AwaitCoProEmpty();           // Wait until co-pro has consumed all commands

        // ----- handle the touch on the buttons -----
        TagVal = HAL_MemRead8(EVE_REG_TOUCH_TAG); // Get Tag value

        ButtonMinus3D = 0;
        ButtonPlus3D = 0;

        if(TagVal == 2)                       // If button '-' pressed
        {
            if(counter > 0)
                counter --;

            ButtonMinus3D = EVE_OPT_FLAT;     // Decrement counter if > 0
            // give button pushed-in appearance
        }

        if(TagVal == 3)                       // If button '+' pressed
        {
            if(counter < 100)
                counter ++;

            ButtonPlus3D = EVE_OPT_FLAT;     // Increment counter if < 100
            // give button pushed-in appearance
        }
    }
}

void eve_example(void)
{
    // Initialise the display
    EVE_Init();

```

```
// Calibrate the display
eve_calibrate();
// Start example code
eve_display();
}
```

4.2 Using your EVE Module (Display Settings)

The code provided includes a file EVE_Config.h which has defines for the different EVE family devices as well as the display settings defines. These allow the code to be compiled for a variety of EVE devices and screen sizes.

The defines below can be used to select your platform.

4.2.1 Set the EVE device

Set the EVE device type as shown [below](#). See file FT8xx.h for the list of available devices.

```
// Select the EVE controller type from the supported list in FT8xx.h.
// Note: In FT8xx.h this will lead to the selection of the EVE Programming
// support methods via macros "EVE_x_ENABLE" where 'x' depends on the level of
// the EVE device support. Alternatively, directly set the EVE_x_ENABLE macro
// required. This must be called prior to including FT8xx.h.
// "#define FT8XX_TYPE BT817" is equivalent to having "#define EVE4_ENABLE".
#ifndef FT8XX_TYPE
#define FT8XX_TYPE FT812
#endif
```

4.2.2 Set the display resolution

The code has several pre-defined displays for WQVGA (480x272) and WVGA (800x480) and WSVGA (1024x600) and these cover the modules provided by Bridgetek such as

- | | |
|--------------------------------------------------------|-------|
| • VM80050A, VM801B50A | WQVGA |
| • VM810C50A-D, ME812A-WH50R, ME813A-WH50C, VM816C50A-D | WVGA |
| • ME817EV with 7" (1024x600) screen | WSVGA |
| • ME817EV with 10.1" (1280x800) screen | WXGA |

Set the display resolution as shown [below](#).

```
// Definitions used for target display resolution selection
#define WQVGA 480 //
#define WVGA 800 //
#define WSVGA 1024 //
#define WXGA 1280 //

// Select the display
#ifndef DISPLAY_RES
#define DISPLAY_RES WQVGA
#endif
```

The actual display settings are in the lower part of file EVE_Config.h. Pre-defined settings are given in the file for WQVGA, WVHGA, WSVGA and WXGA. Additional display settings can be added, or the ones provided can be changed. Check the documentation for your chosen module or panel to ensure that the correct settings are defined.

It may also be necessary to edit the EVE_HAL.c file if you want to change the PLL setting to run the EVE device at a different PLL clock rate or if a different oscillator setting is needed (for example to set external crystal mode on some models of EVE). Refer to the function HAL_EVE_Init(void) near the top of the EVE_HAL.c file.

```
#if DISPLAY_RES == WQVGA
#define EVE_DISP_WIDTH 480 // Active width of LCD display
#define EVE_DISP_HEIGHT 272 // Active height of LCD display
#define EVE_DISP_HCYCLE 548 // Total number of clocks per line
#define EVE_DISP_HOFFSET 43 // Start of active line
#define EVE_DISP_HSYNC0 0 // Start of horizontal sync pulse
#define EVE_DISP_HSYNC1 41 // End of horizontal sync pulse
#define EVE_DISP_VCYCLE 292 // Total number of lines per screen
#define EVE_DISP_VOFFSET 12 // Start of active screen
#define EVE_DISP_VSYNC0 0 // Start of vertical sync pulse
#define EVE_DISP_VSYNC1 10 // End of vertical sync pulse
#define EVE_DISP_PCLK 5 // Pixel Clock
#define EVE_DISP_SWIZZLE 0 // Define RGB output pins
#define EVE_DISP_PCLKPOL 1 // Define active edge of PCLK
#define EVE_DISP_CSPREAD 0
#define EVE_DISP_DITHER 1
```

4.2.3 Enable/Disable QSPI

Set the SPI width as shown **below**. Here the QSPI is disabled, and so single SPI will be used.

```
// Explicitly disable QuadSPI
#ifdef QUADSPI_ENABLE
#undef QUADSPI_ENABLE
#endif
```

4.3 Porting to Other MCUs

The code has been designed to be generic so that it can be ported to other MCUs. This will primarily involve editing the MCU-specific file (e.g. the EVE_MCU_ file).

For more information, please refer to application note BRT_AN_062 Porting BRT_AN_025 to NXP MCU. The link can be found in Appendix A– References

4.4 Examples for BRT_AN_025

Examples are also provided on GitHub for the BRT_AN_025 code. These can be found in the examples folder within the EVE-MCU-BRT_AN_025 repository.

[Bridgetek/EVE-MCU-BRT_AN_025 \(github.com\)](https://github.com/Bridgetek/EVE-MCU-BRT_AN_025)

5 Library Layers

5.1 EVE_API Library Layer

This layer is implemented in EVE_API.c and sits just below the main application. Its purpose is to allow the main program above to use the same syntax as the EVE Programmers guide when writing to the co-processor and so make programming of the display simpler and more easily maintained.

The file contains several types of helper function including:

- Functions which are used to begin, finish and check execution of co-processor lists.
- Functions for writing data to RAM_G and RAM_CMD
- A function for calling each Display List instruction and each Co-Processor command from the EVE programmers guide.

5.1.1 Initialising EVE

This layer includes code in EVE_Init() which will write the display settings registers to the values defined (these can be adjusted to suit your display). It then sets up the GPIO and other registers such as PWM (for the backlight) and sound.

A short co-processor list is used to clear the screen.

One additional step is to clear the bitmap handle properties (including BITMAP_LAYOUT_H and BITMAP_SIZE_H) and this is done by the following code. It is important that this code is executed after the GPU is running and rendering the screen and therefore after the REG_PCLK has been set to the required value.

```
#if (defined EVE2_ENABLE || defined EVE3_ENABLE || defined EVE4_ENABLE)
    // ----- Reset all bitmap properties -----
    EVE_LIB_BeginCoProList();
    EVE_CMD_DLSTART();
    EVE_CLEAR_COLOR_RGB(0, 0, 0);
    EVE_CLEAR(1,1,1);
    for (i = 0; i < 16; i++)
    {
        EVE_BITMAP_HANDLE(i);
        EVE_CMD_SETBITMAP(0,0,0,0);
    }
    EVE_DISPLAY();
    EVE_CMD_SWAP();
    EVE_LIB_EndCoProList();
    EVE_LIB_AwaitCoProEmpty();
# else
    // ----- Reset all bitmap properties -----
    EVE_LIB_BeginCoProList();
    EVE_CMD_DLSTART();
    EVE_CLEAR_COLOR_RGB(0, 0, 0);
    EVE_CLEAR(1,1,1);
    for (i = 0; i < 16; i++)
    {
        EVE_BITMAP_HANDLE(i);
        //EVE_CMD_SETBITMAP(0,0,0,0);
        EVE_BITMAP_LAYOUT(0, 0, 0);
        EVE_BITMAP_SIZE(0, 0, 0, 0, 0);
    }
    EVE_DISPLAY();
    EVE_CMD_SWAP();
    EVE_LIB_EndCoProList();
    EVE_LIB_AwaitCoProEmpty();
#endif
```

5.1.2 Co-Processor Helpers

These functions perform the necessary tasks to begin and execute co-processor lists.

```
void EVE_LIB_BeginCoProList(void)
```

Puts Chip Select low and sends the starting address of the RAM_CMD location where the commands will be written. Chip Select remains low.

```
void EVE_LIB_EndCoProList(void)
```

Brings Chip Select high to end the burst and writes the new value to REG_CMD_WRITE so that the co-processor will execute the newly added commands.

```
void EVE_LIB_AwaitCoProEmpty(void)
```

Waits for the read and write pointers to become equal which indicates completion of the command execution.

5.1.3 Creating screens and executing commands

The API Layer provides functions to begin and end lists of co-processor commands. The green and red lines of code will normally be used as shown here with the actual screen content created between these.

5.1.3.1 Beginning and Ending Co-Processor Lists

The green text contains the initial actions to begin the new screen (the colour set for CLEAR_COLOR_RGB can be changed to suit the needs of the application's background). The blue text can be replaced by the required items to be displayed on the screen. The red text shows the end of the list.

```
EVE_LIB_BeginCoProList();           // CS low and send address in RAM_CMD
EVE_CMD_DLSTART();                 // When executed, EVE will begin a new DL
EVE_CLEAR_COLOR_RGB(0, 0, 0);      // Select colour to clear screen to
EVE_CLEAR(1,1,1);                  // Clear

EVE_COLOR_RGB(255, 255, 255);
EVE_CMD_TEXT(100, 100, 28, OPT_CENTERX|OPT_CENTERY, "Hello");

EVE_DISPLAY();                     // Tells EVE that this is the end
EVE_CMD_SWAP();                    // Swaps new list into foreground buffer
EVE_LIB_EndCoProList();            // CS high and update REG_CMD_WRITE
EVE_LIB_AwaitCoProEmpty();         // Wait for FIFO to be empty
// (commands executed)
```

5.1.3.2 Executing a Single Co-Processor Command

When just executing a co-processor command (for example calling CMD_SETROTATE during set-up of the application to set the screen orientation) then the following can be used:

```
EVE_LIB_BeginCoProList();           // CS low and send address in RAM_CMD

EVE_CMD_SETROTATE(2);

EVE_LIB_EndCoProList();            // CS high and update REG_CMD_WRITE
EVE_LIB_AwaitCoProEmpty();         // Wait for FIFO to be empty
// (command(s) executed)
```

5.1.3.3 Co-Processor Lists of more than 4K Size

The examples above use burst writes (CS low, write address, stream data holding CS low, CS high). Therefore, no register writes should be carried out in the middle as this would interrupt the burst. A list can however be created in more than one section as shown below. This is also useful if a list consists of more than (4K-4) bytes. In this latter case the list would be written in smaller sections, each section being executed to create more space in the RAM_CMD FIFO before the next section is sent .

```
// FIRST SECTION OF LIST
EVE_LIB_BeginCoProList();           // CS low and send address in RAM_CMD

EVE_CMD_DLSTART();                  // When executed, EVE will begin a new DL
EVE_CLEAR_COLOR_RGB(0, 0, 0);       // Select colour to clear screen to
EVE_CLEAR(1,1,1);                   // Clear the screen
EVE_COLOR_RGB(255, 255, 255);

EVE_LIB_EndCoProList();             // CS high and update REG_CMD_WRITE
EVE_LIB_AwaitCoProEmpty();          // Wait for FIFO to be empty
// (commands executed)

// **** You can write or read registers here ****

// SECOND SECTION OF LIST
EVE_LIB_BeginCoProList();           // CS low and send address in RAM_CMD

EVE_CMD_TEXT(100, 100, 28, OPT_CENTERX|OPT_CENTERY,"Hello");
EVE_DISPLAY();                       // Tells EVE that this is the end
EVE_CMD_SWAP();                       // Swaps new list into foreground buffer

EVE_LIB_EndCoProList();             // CS high and update REG_CMD_WRITE
EVE_LIB_AwaitCoProEmpty();          // Wait for FIFO to be empty
// (commands executed)
```

The above sequence will create the same set of commands in RAM_DL as the code below.

```
EVE_LIB_BeginCoProList();           // CS low and send address in RAM_CMD

EVE_CMD_DLSTART();                  // When executed, EVE will begin a new DL
EVE_CLEAR_COLOR_RGB(0, 0, 0);       // Select colour to clear screen to
EVE_CLEAR(1,1,1);                   // Clear the screen
EVE_COLOR_RGB(255, 255, 255);

EVE_CMD_TEXT(100, 100, 28, OPT_CENTERX|OPT_CENTERY,"Hello");
EVE_DISPLAY();                       // Tells EVE that this is the end
EVE_CMD_SWAP();                       // Swaps new list into foreground buffer

EVE_LIB_EndCoProList();             // CS high and update REG_CMD_WRITE
EVE_LIB_AwaitCoProEmpty();          // Wait for FIFO to be empty
// (commands executed)
```

The usage is fundamentally the same as the library and examples described in [BRT_AN_008](#) (FT81x Creating a Simple Library For PIC MCU) and [BRT_AN_014](#) (FT81X Simple PIC Library Examples) and so these can be used as a reference when using this library.

5.1.3.4 Limitations in RAM_DL and RAM_CMD

It is important to note that the overall limit of 8K for the generated RAM_DL list still applies, even if lists are sent in multiple sections. It is also important to bear in mind that the size of a co-processor command is not always the same as the size of the resulting RAM_DL instructions which the co-processor generates from the commands.

For example, the CMD_BUTTON uses 16 bytes of RAM_CMD plus the size of the string (plus any string arguments in BT81x) for the command, but the graphic operations in RAM_DL which the co-

processor creates to render the button will be larger than this. The 8K RAM_DL limit does not therefore mean that 8K of co-processor commands can be used in one list.

REG_CMD_DL indicates the next available location in RAM_DL and so after executing a list commands (but before the swap) this register can be used to check how full RAM_DL is. The value read will be between 0 and 8191 with 8191 indicating the RAM_DL is full.

The value of REG_CMD_DL is read after executing the commands above but before the swap is executed. The swap is sent using a separate transaction (beginning with `EVE_LIB_BeginCoProList()` and ending with `EVE_LIB_EndCoProList()` and `EVE_LIB_AwaitCoProEmpty()`) because a register read or write cannot take place whilst an existing SPI transaction (burst write or read) is in progress.

```
EVE_LIB_BeginCoProList();           // CS low and send address in RAM_CMD

EVE_CMD_DLSTART();                 // When executed, EVE will begin a new DL
EVE_CLEAR_COLOR_RGB(0, 0, 0);      // Select color to clear screen to
EVE_CLEAR(1,1,1);                  // Clear the screen
EVE_COLOR_RGB(255, 255, 255);

EVE_CMD_TEXT(100, 100, 28, OPT_CENTERX|OPT_CENTERY, "Hello");
EVE_DISPLAY();                     // Tells EVE that this is end of the list

EVE_LIB_EndCoProList();            // CS high and update REG_CMD_WRITE
EVE_LIB_AwaitCoProEmpty();         // Wait for FIFO to be empty
// (commands executed)

uint16_t RAM_DL_fullness = HAL_MemRead16(EVE_REG_CMD_DL);
// check value in MCU debugger or print to UART etc.

EVE_LIB_BeginCoProList();           // CS low and send address in RAM_CMD

EVE_CMD_SWAP();                    // Swaps new list into foreground buffer

EVE_LIB_EndCoProList();            // CS high and update REG_CMD_WRITE
EVE_LIB_AwaitCoProEmpty();         // Wait for FIFO to be empty
// (commands executed)
```

5.1.4 Writing RAM_G and RAM_CMD

These functions allow burst writes to be performed to RAM_G and RAM_CMD.

```
Void EVE_LIB_WriteDataToRAMG(const uint8_t *ImgData, uint32_t DataSize, uint32_t
DestAddress)
```

This function performs an SPI burst write to RAM_G. The starting address, as well as the source of the data and amount of data are specified. EVE can be written in a similar fashion to an SPI memory device. After asserting CS and sending the address, data can be written as a burst whilst keeping CS low. A similar function performs a read of the selected memory.

```
void EVE_LIB_WriteDataToCMD(const uint8_t *ImgData, uint32_t DataSize)
```

This function allows a block of data to be written to RAM_CMD which is needed when writing data to be inflated for example. This is more complex as the circular nature of the buffer must be handled in addition to splitting data into chunks since the buffer is only 4K in size. This function handles the entire process and so makes writing to RAM_CMD as simple as to RAM_G for the layers above. A flow chart can be found in [BRT_AN_008](#) (FT81x Creating a Simple Library For PIC MCU) for loading data via the co-processor buffer RAM_CMD.

Other helper functions are provided such as for writing strings and for retrieving co-processor results (as some commands such as GetProps return their result via RAM_CMD).

```
Uint16_t EVE_LIB_SendString(const char* string)
```

This function sends a string of characters and is used by commands such as CMD_TEXT, CMD_BUTTON and CMD_TOGGLE which all use text strings. This function takes care of the extra padding which is required as all EVE commands must be 32-bit aligned. Therefore, depending on the length of the string (plus the necessary null character to terminate it) then between one and three extra 00 bytes are added to pad the command to be a multiple of 4 bytes. The main application can therefore send strings without needing to consider the padding.

5.1.5 Writing DL Instructions and Co-Processor Commands

Using EVE commands via the co-processor requires some data formatting to convert the parameters of the command into the correct hex values to be sent as well as keeping track of the number of bytes sent to update the write pointer correctly. Some commands also require padding to make their total size including parameters a multiple of 4 bytes. The functions in EVE_API hide this from the main application.

For example, the CLEAR_COLOR_RGB instruction, and the GRADIENT and TEXT co-processor commands use the following functions.

```
void EVE_CLEAR_COLOR_RGB(uint8_t R, uint8_t G, uint8_t B)
{
    HAL_Write32(EVE_ENC_CLEAR_COLOR_RGB(R, G, B));
    HAL_IncCmdPointer(4);
}

void EVE_CMD_GRADIENT(int16_t x0, int16_t y0, uint32_t rgb0, int16_t x1, int16_t
y1, uint32_t rgb1)
{
    HAL_Write32(EVE_ENC_CMD_GRADIENT);
    HAL_Write32(((uint32_t)y0 << 16) | (x0 & 0xffff));
    HAL_Write32(rgb0);
    HAL_Write32(((uint32_t)y1 << 16) | (x1 & 0xffff));
    HAL_Write32(rgb1);
    HAL_IncCmdPointer(20);
}

void EVE_CMD_TEXT(int16_t x, int16_t y, int16_t font, uint16_t options, const char*
string, ...)
{
    va_list args;
    uint32_t CommandSize;
    uint32_t StringLength;
    uint8_t i, num=0;

    va_start(args, string);

    #if defined (EVE3_ENABLE) || defined (EVE4_ENABLE)
    //Only check % characters if option OPT_FORMAT is set
    num = (options & EVE_OPT_FORMAT) ? (COUNT_ARGS(string)) : (0);
    #endif

    HAL_Write32(EVE_ENC_CMD_TEXT);
    HAL_Write32(((uint32_t)y << 16) | (x & 0xffff));
    HAL_Write32(((uint32_t)options << 16) | (font & 0xffff));
    CommandSize = 12;

    StringLength = EVE_LIB_SendString(string);

    for (i = 0; i < num; i++)
    {
        HAL_Write32((uint32_t)va_arg(args, uint32_t));
    }

    CommandSize = CommandSize + StringLength + (num*4);

    HAL_IncCmdPointer(CommandSize);
```

```

        va_end(args);
    }

```

5.2 EVE_HAL Library Layer

This file provides functions to handle the EVE message structure over SPI. It also has additional helper functions which are called from the API layer above.

5.2.1 EVE SPI Message Structure

These functions provide the correct formatting to match the SPI message structure required by EVE. This includes chip select operation, addressing, and data byte ordering. The functions translate calls such as HAL_MemWrite32 into a series of chip select and byte transfer operations which in turn call the lowest level MCU Specific functions in the MCU-specific file.

Note: BRT_AN_006 (FT81x Simple PIC Example Introduction) has full details on the message structures used by EVE including addressing and reading/writing data and host commands.

5.2.1.1 Reading and Writing Memory

When accessing EVE over SPI to write or read the various memory blocks (REGISTERS, RAM_G, RAM_DL, RAM_CMD etc.) the SPI transfers generally consist of the following sequence. (note that Host Commands are slightly different as explained later in this section)

CS# Low	Address with read/write bits	One or more data bytes	CS# High
---------	---------------------------------	---------------------------	----------

The individual Chip Select (CS#), Addressing and Data functions provided in EVE_HAL.c are used to create the above sequence. This allows both reading/writing of a single memory location, and the creation of SPI burst writes and reads. This is where the CS# goes low, followed by the first address to be read or written, followed by a series of bytes transferred and ending when CS# goes high. EVE increments its internal address counter with each byte allowing bursts of data to be transferred without sending the address for each 8/16/32-bit read or write which is often more efficient.

The functions provided include the following:

Chip Select

This function will set or clear the CS# line depending on the state value passed in. Note that passing in 1 asserts chip select to low and passing in 0 de-asserts chip select to high.

```

    HAL_ChipSelect(state)                      Controls the CS# line to high or low state

```

Addressing

These functions send the 3-byte address with the upper two bits configured accordingly to tell EVE that the address is to be written to or read from

```

    HAL_SetWriteAddress(address)              Send the address to be written to
    HAL_SetReadAddress(address)              Send the address to be read from

```

Data

These functions perform a read and write over SPI of the associated data size (32-bit, 16-bit, 8-bit). Standard 1-bit SPI will always read and write a byte at the same time but when writing the returned value can be ignored or likewise when reading a dummy value can be sent.

HAL_Write32(val32)	Write 32-bit value
HAL_Write16(val16)	Write 16-bit value
HAL_Write8(val8)	Write 8-bit value
Val32 HAL_Read32()	Read 32-bit value
Val16 HAL_Read16()	Read 16-bit value
Val8 HAL_Read8()	Read 8-bit value
HAL_Write(Buffer, length)	Send a stream of values from a buffer

Combined Chip Select, Address and Data (for register writes/reads)

These functions combine calls to the Chip Select, Addressing and Data functions to allow reading and writing of registers via a single call.

HAL_MemWrite32(address, val32)	Write 32-bit register
HAL_MemWrite16(address, val16)	Write 16-bit register
HAL_MemWrite8(address, val8)	Write 8-bit register
Val32 HAL_MemRead32(address)	Read 32-bit register
Val16 HAL_MemRead16(address)	Read 16-bit register
Val8 HAL_MemRead8(address)	Read 8-bit register

For example, when writing a 32-bit value to a register.

```
void HAL_MemWrite32(uint32_t address, uint32_t val32)
{
    // CS low begins the SPI transfer
    MCU_CSLOW();
    // Send address to be written
    HAL_SetWriteAddress(address);
    // Send the data value
    HAL_Write32(val32);
    // CS high terminates the SPI transfer
    MCU_CS_HIGH();
}
```

5.2.1.2 Host Commands

EVE also uses Host Commands which are system commands sent directly to EVE rather than to a memory address. These include selecting internal/external clock source and the active command which wakes up the EVE device. These are sent as a sequence of three bytes containing the command and the parameter values but no addressing. The function for this is shown below:

```
void HAL_HostCmdWrite(uint8_t cmd, uint8_t param)
{
    // CS low begins the SPI transfer
    HAL_ChipSelect(1);
    // Send command
    MCU_SPIWrite8(cmd);
    // followed by parameter
```

```
MCU_SPIWrite8(param);  
// and a dummy 00 byte  
MCU_SPIWrite8(0x00);  
// CS high terminates the SPI transfer  
HAL_ChipSelect(0);  
}
```

For example, in EVE_HAL.c this command is used:

```
HAL_HostCmdWrite(0, 0x00); // Set active
```

The EVE device datasheet has a table of the commands available depending on the device family.

5.2.2 EVE Supporting Functions

This file also contains the lower level implementations of several of the functions provided by the EVE_API layer above.

This includes awaiting the co-processor completion (REG_CMD_READ == REG_CMD_WRITE) and checking free space, both of which are required by the API layer above.

5.3 MCU Layer

The EVE_MCU_[MCU-type] file contains the low-level hardware functions for the MCU being used.

5.3.1 Preprocessor Symbol

The GitHub download for each specific platform contains the MCU layer files for *all* the available platforms. The project for each MCU includes a preprocessor symbol in the project properties to enable only the required file. If porting to other platforms or toolchains, or if adding the code files into an existing project, ensure that this preprocessor symbol is set.

For example, the FT900 file EVE_MCU_FT9XX.c is gated by the following code.

```
#if defined(PLATFORM_FT9XX)
```

The project settings must define this value so that the correct MCU-specific file is used, or errors will occur as several MCU-specific files exist with the same functions in them.

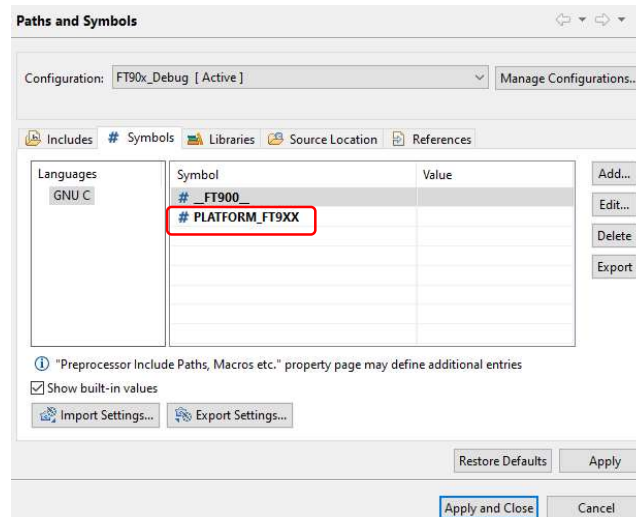


Figure 5 – Enabling the FT9XX platform

5.3.2 MCU Initialisation

The MCU-specific files include a function **MCU_Init** to initialise the MCU (such as setting the clock source, GPIO lines and SPI module up).

5.3.3 MCU SPI Functions

They also include a set of functions which are called from the layers above to handle the GPIO lines and SPI. If porting to a different MCU platform, the **code highlighted** in each function should be replaced with the relevant code for the MCU to set or clear the GPIO line or to clock one byte in and out of the SPI Master.

```
inline void MCU_CSslow(void)          // CS line low
{
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
}
inline void MCU_CShigh(void)         // CS line high
{
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
}
inline void MCU_PDlow(void)          // PD line low
{
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_RESET);
}
inline void MCU_PDhigh(void)        // PD line high
{
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_SET);
}

// ----- SPI Send and Receive -----
uint8_t MCU_SPIReadWrite8(uint8_t DataToWrite)
{
    uint8_t DataRead[4];
    uint8_t TxBuffer[4];

    TxBuffer[0] = DataToWrite;

    switch(HAL_SPI_TransmitReceive(&SpiHandle, (uint8_t*)TxBuffer, (uint8_t
    *)DataRead, 1, 5000))
    {
        case HAL_OK:
            /* Communication is completed _____ */
            break;
        default:
            break;
    }
    return DataRead[0];
}
```

They also include functions which provide 8, 16, 24 and 32-bit reads and writes and translate these operations requested by the layers above to a series of byte transfers which can call the function **MCU_SPIReadWrite8**. This allows the code to be adjusted for the MCU's endianness. The examples for the MCUs provided can be used as a reference for this.

Finally, delay functions are implemented in this file where suitable code for the MCU platform should be added to provide 500ms and 20ms delays.

The porting guide **BRT_AN_062** has additional guidance on this, please refer to Appendix A-References for a link to the porting guide.

6 Downloading the Code from GitHub

The source files and full MCU projects are available on GitHub at [Bridgetek/EVE-MCU-BRT_AN_025 \(github.com\)](https://github.com/Bridgetek/EVE-MCU-BRT_AN_025) (external link). The files can be obtained by doing a Git Clone or by downloading the zip file from the Code menu as shown in Figure 6.

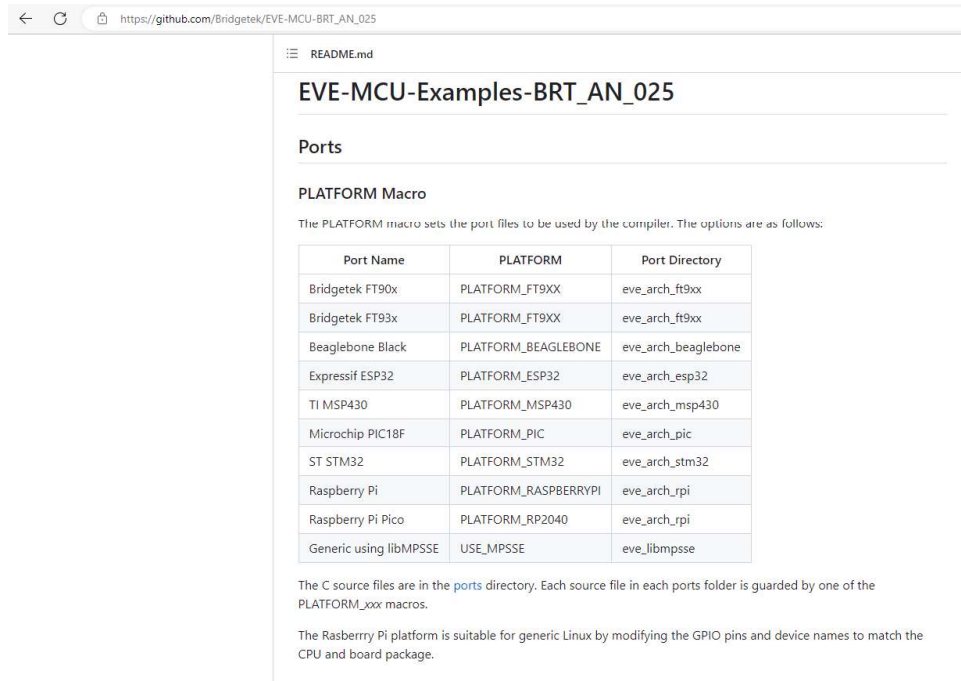


Figure 6 – GitHub Repository for BRT_AN_025 code

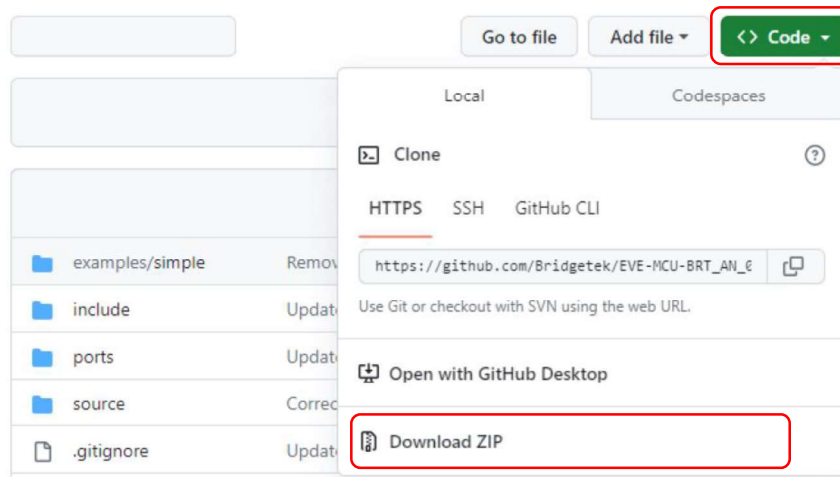


Figure 7 – GitHub Repository - Downloading

7 Setting up the MCU

7.1 FT900 MCU Set-up

7.1.1 Hardware

The provided project was developed using a board from the [MM900EV1B](#) series along with the UMFTPD2A debug and programming module. The MM900EV1B can be attached directly to the rear of some modules (such as [ME812A-WH50R](#), [ME813A-WH50C](#) and [ME817EV](#) modules) as illustrated below or can be connected via short wires to the corresponding signals if using a different EVE module.

Ensure that the power supply to the MM900 is capable of also powering the EVE board. If using third-party modules which may consume more current, a separate power connection to the EVE module could be used, with the grounds of the MM900 and EVE common to both power sources.

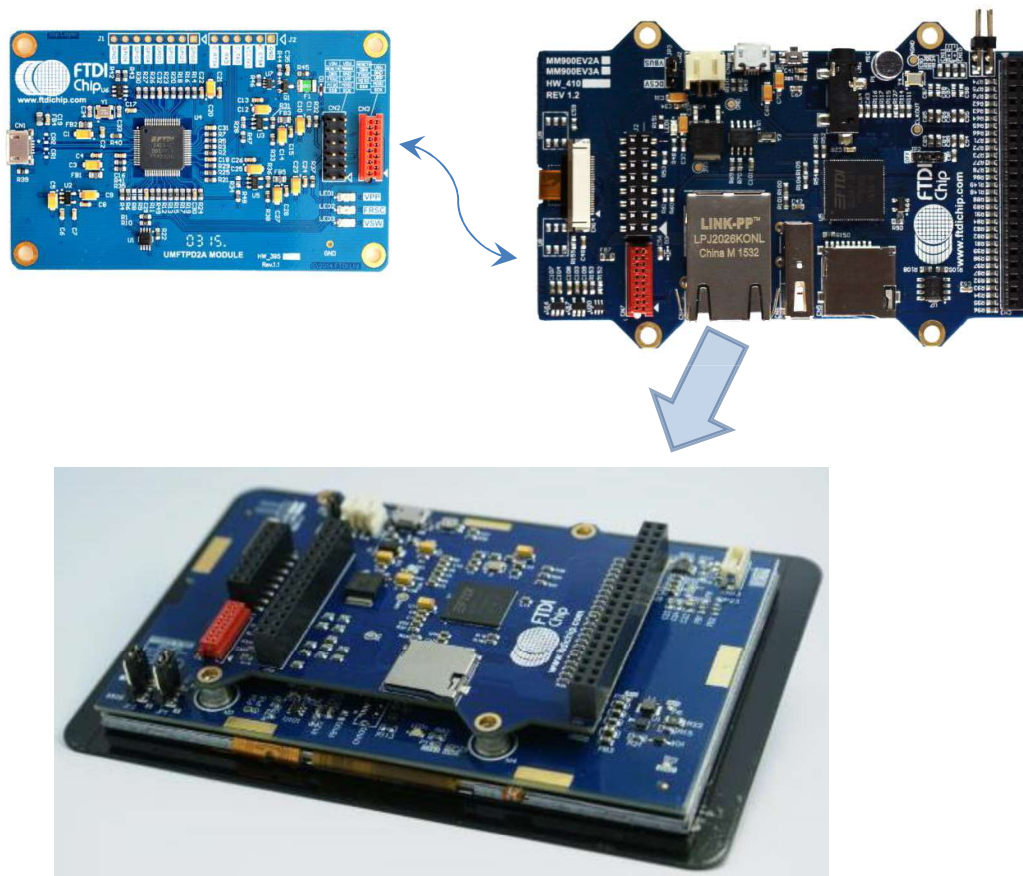


Figure 8 – FT900 hardware

7.1.2 Software Tools

This example can be loaded via the FT900 toolchain which is available from <https://brtchip.com/ic-module/ft9xx-toolchain/>. At the time of writing, version 2.6.0 was used.

Unzip the code which was downloaded from GitHub in section **Error! Reference source not found..** Copy the entire folder into the workspace folder (or another location if preferred).

Then, in the IDE, choose File -> Import. Select "Existing Projects into Workspace" and click next.

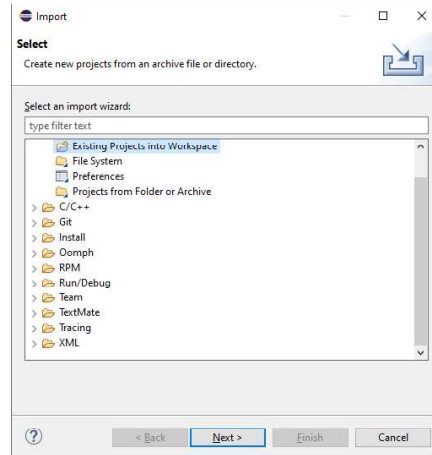


Figure 9 – Importing existing projects into workspace

Browse to the folder containing the project and click “Select Folder”. Click Finish back in the Import window and the project will appear in the Project Explorer in the IDE.

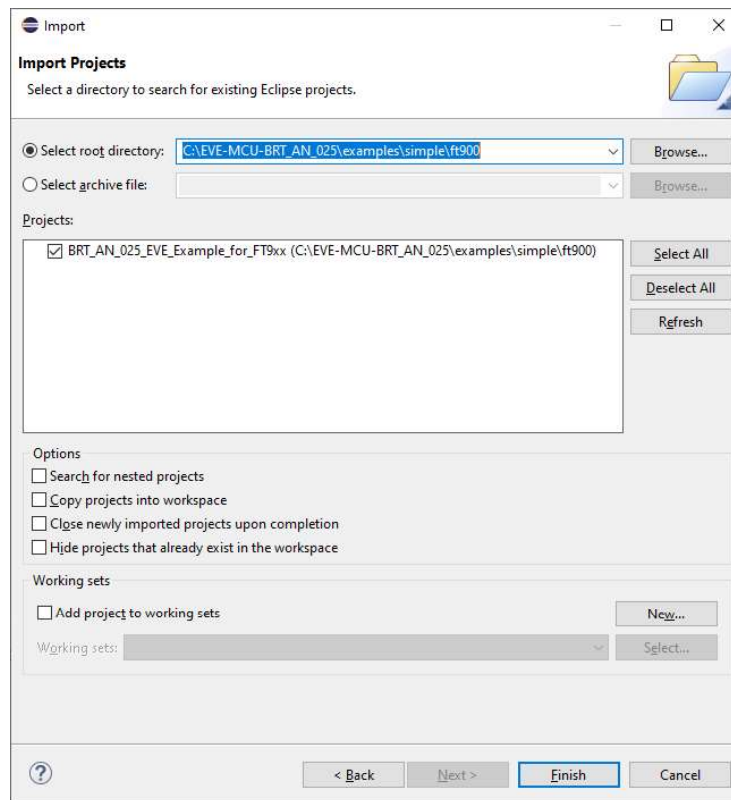


Figure 10 – Selecting the project folder

Full Instructions for loading the project can be found in section 2 of [AN_360](#).

7.1.3 Code

For the FT9xx, the main.c file calls the eve_example function from file eve_example.c. On this platform, it also provides additional functions for storing calibration data and accessing the data stored in the program memory / flash.

After the initial programming of the code onto the MCU and the first run of the code through the calibration steps, the calibration values will be stored, and you will no longer see the calibration screen. To force a re-calibration, touch the screen and hold the touch. Reset the MCU (or power-up the MCU/EVE if presently unpowered) whilst still holding the touch. After existing reset/Power up, lift your finger from the screen and the calibration will begin.

The FT900 supports QSPI and so the option can be enabled if all the lines are connected (for example if connecting an ME813A or ME817EV to the 16-way header on the MM900EV1B) and that the EVE device being used supports QSPI. See section 4.2.3 for details of the setting.

Ensure that the PLATFORM_FT9XX symbol is defined as shown below. It should be set in the provided project but if adding the BRT_AN_025 files to another project then this define should be set in the new project.

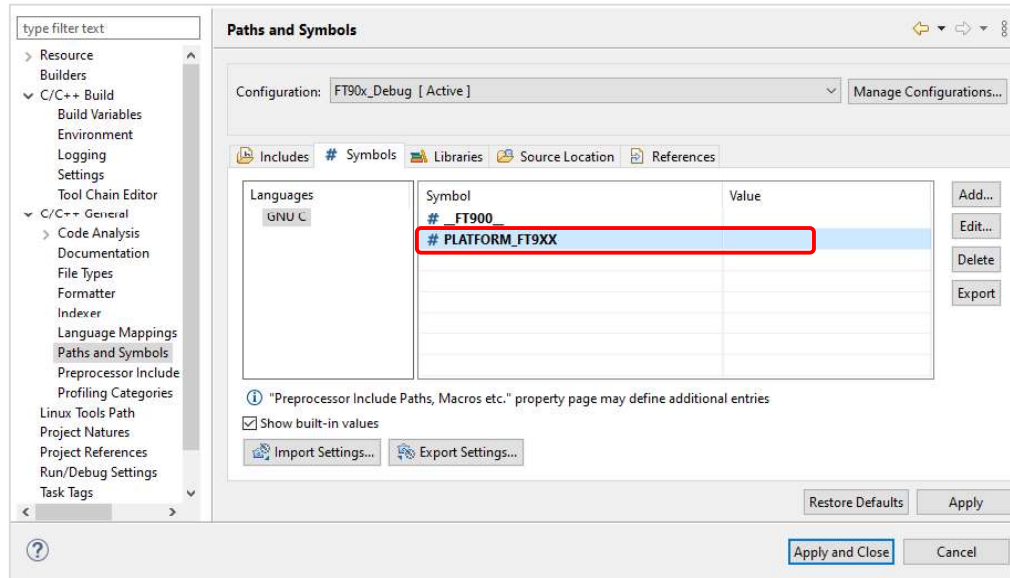


Figure 11 – Platform setting

Note that on some toolchains, the presence of the other project makefiles for the other platforms supported may cause errors when doing a full build. If build errors result from files in other platform folders, these other platform folders such as examples\simple\ESP32 could be removed. If it is desired to keep the other platforms inside the project folder structure (for example if you plan to build the code for several platforms) then these folders can be excluded from the build.

To exclude the ESP32 folder, please refer to the example below:

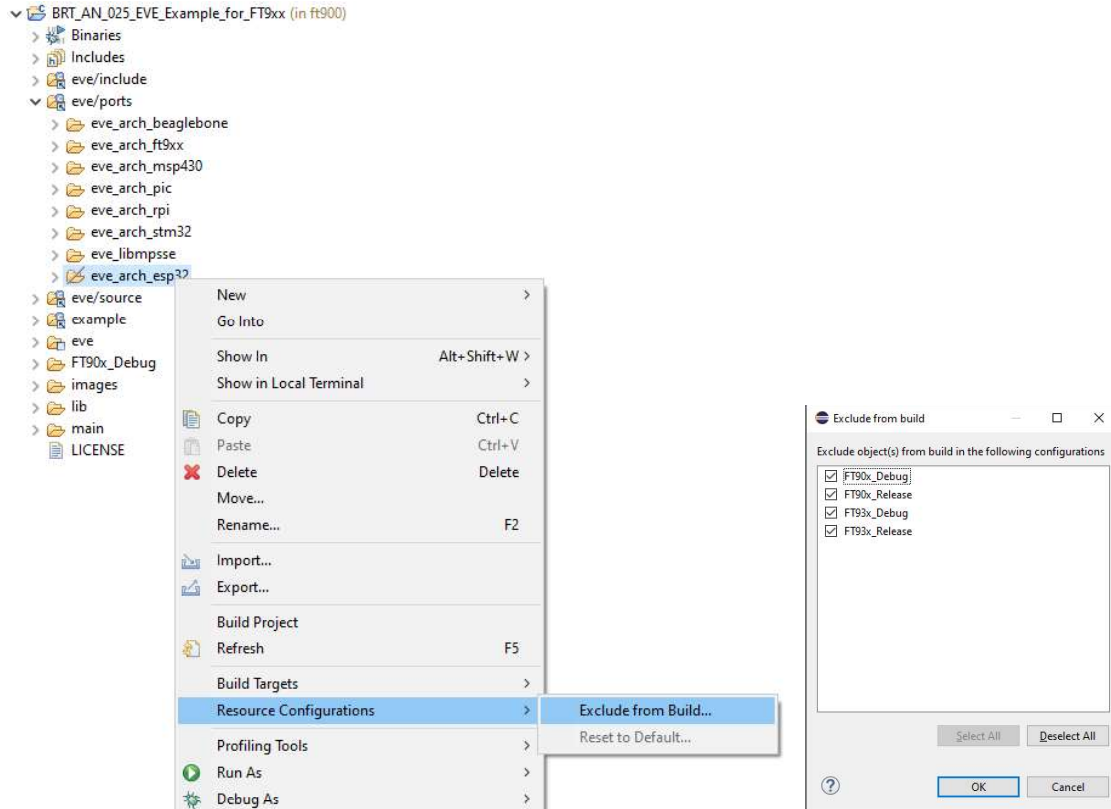


Figure 12 – Excluding ESP32 folder from FT900 build

7.2 PIC18F MCU Set-up

7.2.1 Hardware

The schematic below shows a simple way of connecting EVE (in this case ME812A-WH50R) to the PIC18F.

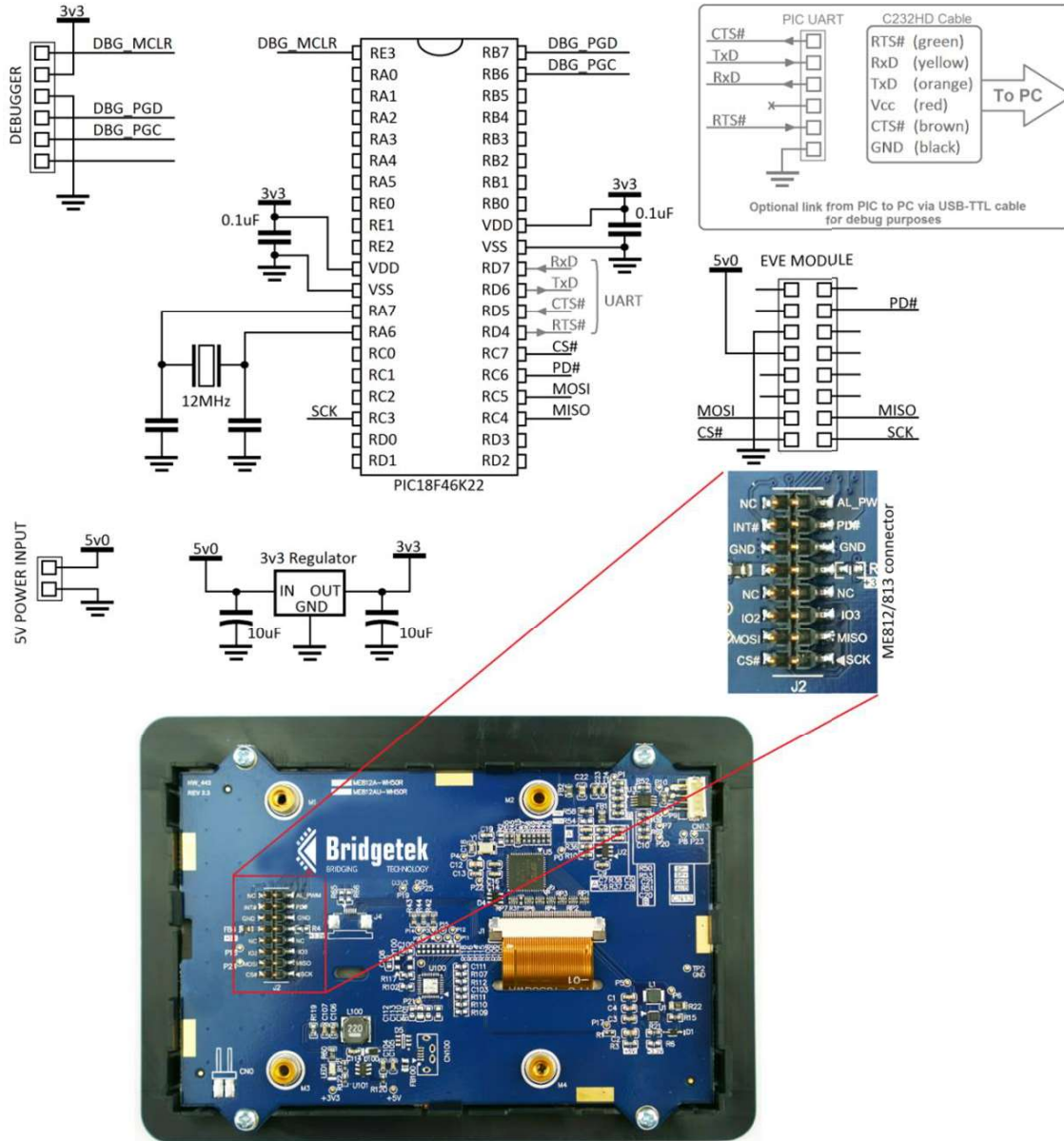


Figure 13 – PIC Hardware using the ME812A-WH50R module

7.2.2 Software Tools

This example can be loaded via the MPLAB X toolchain available from Microchip. The project provided can be opened in MPLAB X and built using the XC compiler. At the time of writing, the version used was MPLABX v6.00 and compiler xc8 v2.36.

Copy the project to your workspace folder and open the project.

7.2.3 Code

For the PIC, the main.c file calls the eve_example function from file eve_example.c which runs the application thereafter. The PIC example does not store the calibration touch transform values in the associated functions in main.c but this could be added.

Ensure that the PLATFORM_PIC symbol is defined as shown below. It should be set in the provided project but if adding the BRT_AN_025 files to another project then this define should be set in the new project.

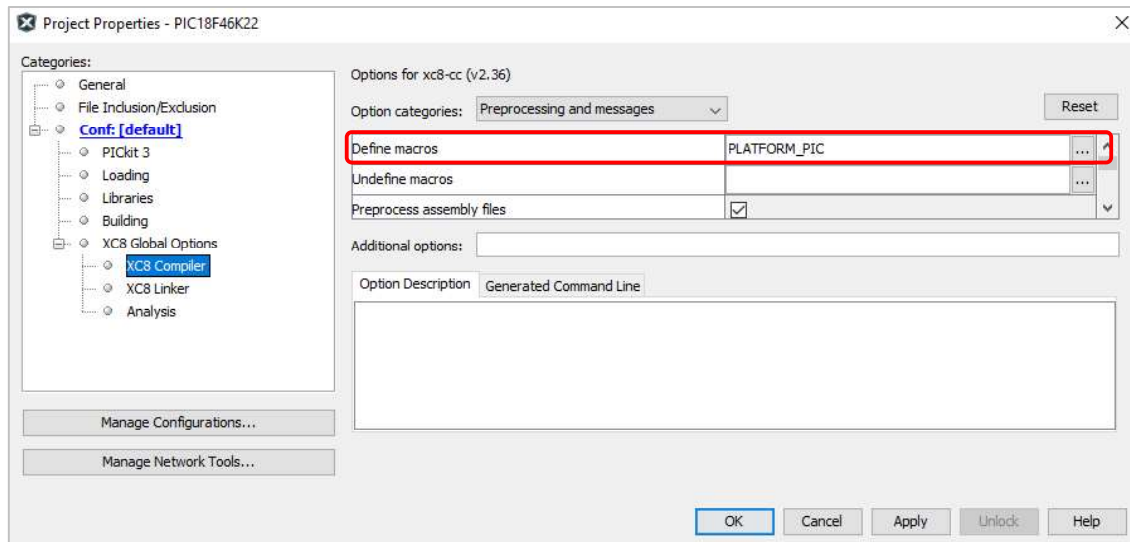


Figure 14 – Platform setting

7.3 STM32F0 MCU Set-up

7.3.1 Hardware

The schematic below shows a simple way of connecting the EVE module (in this case ME812A-WH50R) to the STM32F0 DISCOVERY evaluation module.

Ensure that the overall current drawn by the EVE module is within the limits of the current which the STM32 Discovery can provide especially if powering over USB. If the current exceeds this limit, a separate power connection to the EVE module could be used, with the grounds of the STM32 and

EVE common to both power sources.

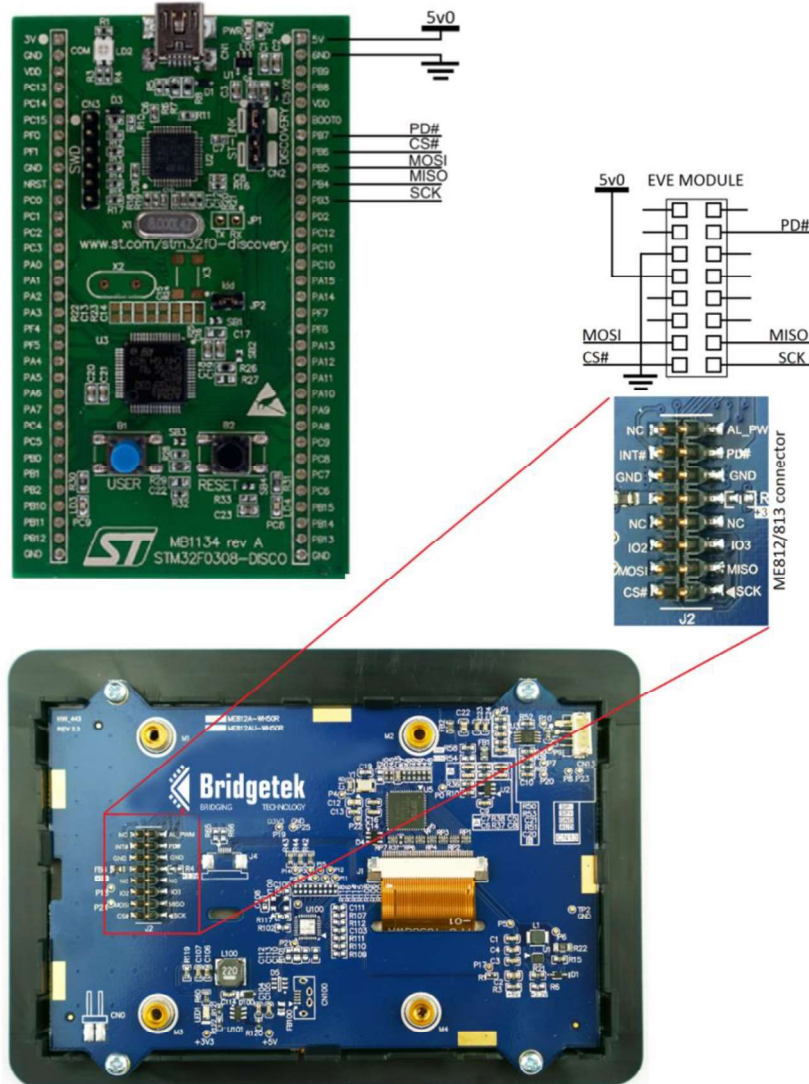


Figure 15 – STM32 Hardware

7.3.2 Software Tools

This example was developed on the Keil uVision 5 toolchain and was based upon an SPI example provided with the toolchain.

At the time of writing, the project is compatible with ARM Compiler 5 (e.g. ARM Compiler v5.06 update 7). If using the latest uVision toolchain(e.g. V5.37) it is necessary to install the ARM compiler v5. The readme for the toolchain has details on how to do this.

The project can be loaded by opening the Project file within the folder \Projects\MDK-ARM_FT81x\
 Note that the BRT_AN_025 files from Github do not include the library files for the MCU itself. One way to obtain these is to download the files from the Keil website for the STM32F051R8T6
[MDK5 - STMicroelectronics STM32F051R8Tx \(keil.com\)](http://www.keil.com/MDK5-STMicroelectronicsSTM32F051R8Tx)

7.3.3 Code

For the STM32, the main.c file calls the eve_example function from file eve_example.c which runs the application thereafter. The STM32 example does not store the calibration touch transform values in the associated functions in main.c but this could be added.

Note that the code in the MCU specific file EVE_MCU_STM32.c is enabled via PLATFORM_STM32. It is recommended to check that this is defined as a preprocessor symbol. Open the project options and check the C/C++ tab. The Preprocessor Symbols section should have PLATFORM_STM32 included within the Define text box (other defines may also be included there too).

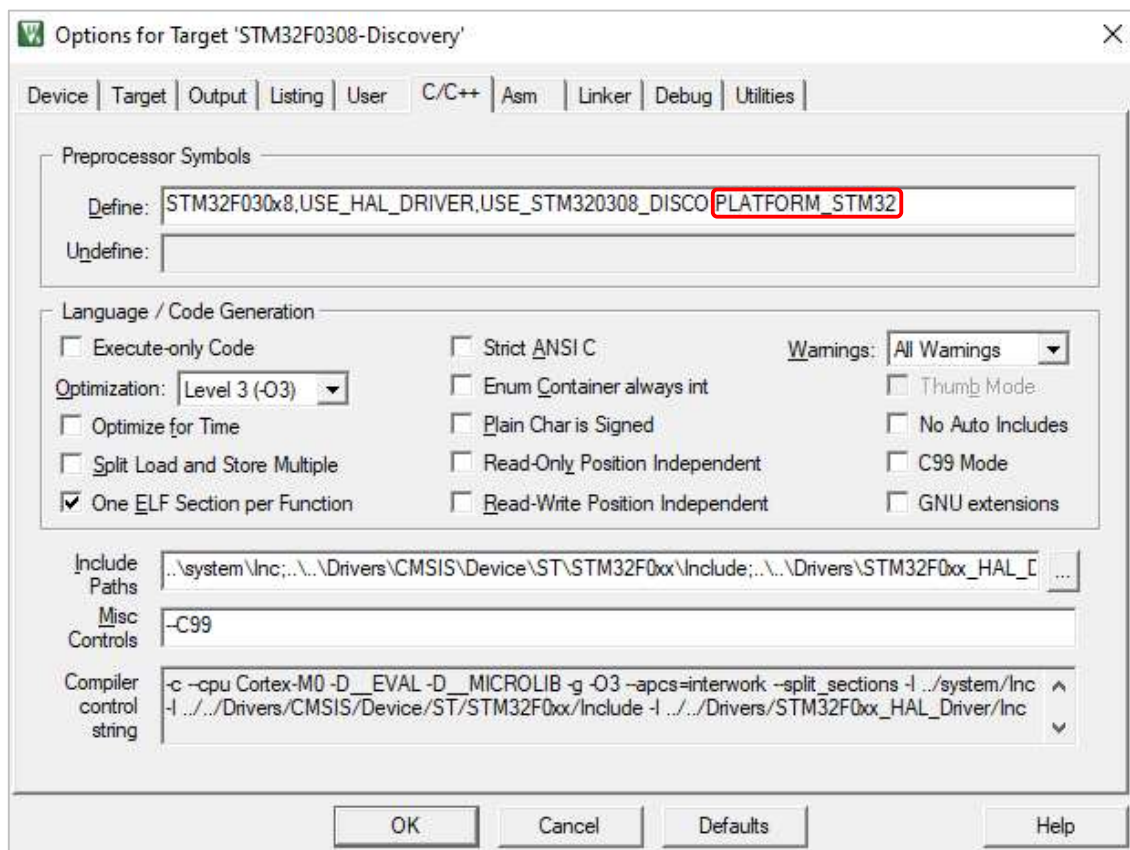


Figure 16 – Platform setting

7.4 ESP32 MCU Set-up

7.4.1 Hardware

An example hardware configuration for initial evaluation of the code is shown below. In this case an ME812A-WH50R is used.

Ensure that the overall current drawn by the EVE module is within the limits of the current which the ESP32 board can provide especially if powering over USB. If the current exceeds this limit, a separate power connection to the EVE module could be used, with the grounds of the ESP32 and EVE common to both power sources.

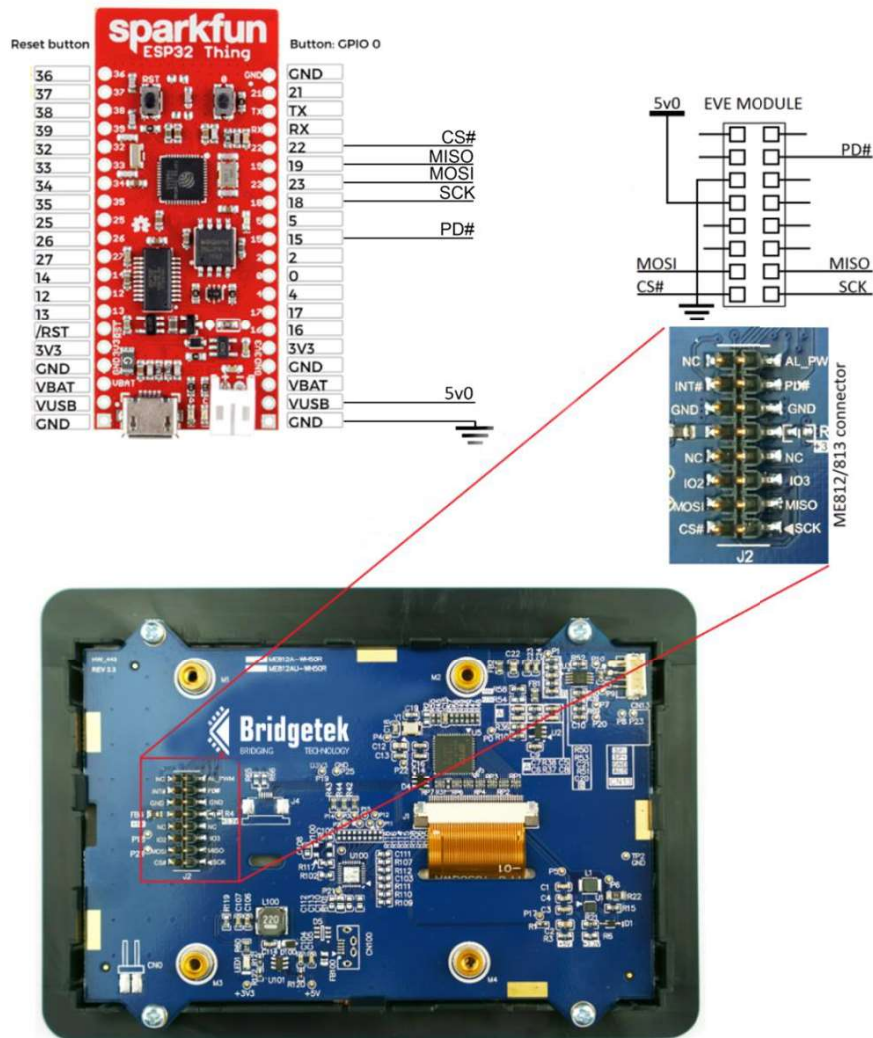


Figure 17 – ESP32 Hardware

7.4.2 Software Tools

This example was created via the ESP-IDF toolchain which can be found at this link:
<https://github.com/espressif/esp-idf>

After running the ESP-IDF command prompt, connect the board to the PC and check that the drivers have been installed (in this case FTDI drivers for the ESP Thing, which provide a Virtual COM Port via the on-board FTDI FT-X series USB – Serial converter). Also check which COM port has been assigned to the board, by checking Device Manager under the Ports section. In the case below it was COM3.

At the time of writing, the ESP-IDF version used was release 4.4.

There are several ways to use the ESP-IDF tools and so the user may wish to use their preferred way, but one example is given below.

Open a command prompt and browse to the ESP-IDF folder

Run the export.bat

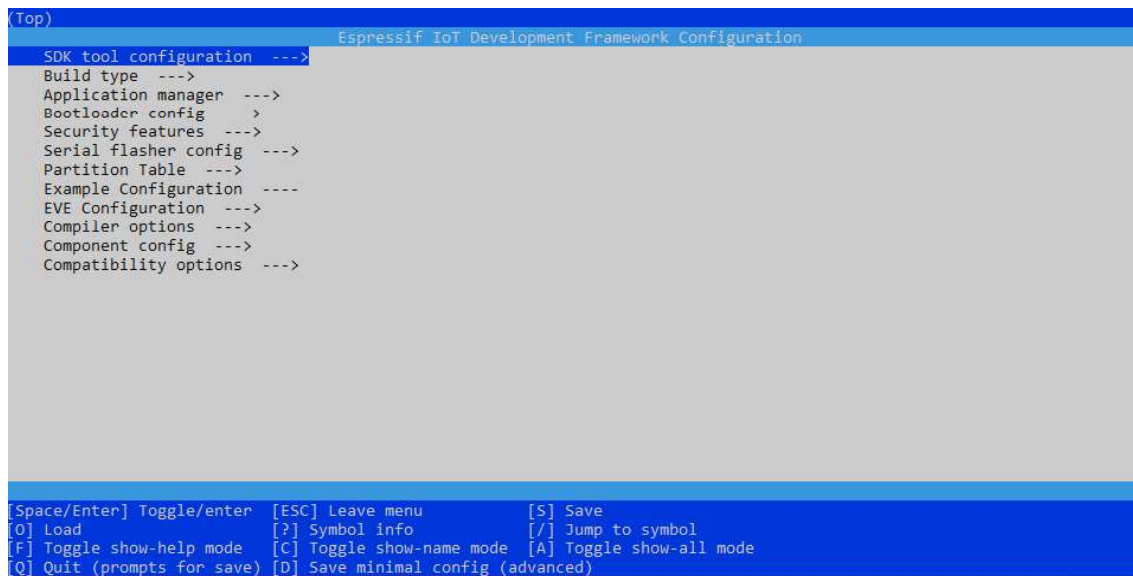
```
C:\Espressif\frameworks\esp-idf-v4.4.1>export.bat
```

Once this completes, browse to the folder containing the example

```
C:\Espressif\frameworks\esp-idf-v4.4.1> cd C:\EVE-MCU-  
BRT_AN_025\examples\simple\ESP32
```

Run the menuconfig to ensure that all settings are applied

```
C:\EVE-MCU-BRT_AN_025\examples\simple\ESP32>idf.py menuconfig
```



```
(Top)
Espressif IoT Development Framework Configuration
SDK tool configuration --->
Build type --->
Application manager --->
Bootloader config >
Security features --->
Serial flasher config --->
Partition Table --->
Example Configuration ----
EVE Configuration --->
Compiler options --->
Component config --->
Compatibility options --->

[Space/Enter] Toggle/enter  [ESC] Leave menu      [S] Save
[O] Load                    [?] Symbol info      [/] Jump to symbol
[F] Toggle show-help mode   [C] Toggle show-name mode [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)
```

Figure 18 – ESP32 menuconfig

For many applications, the default options will be fine and so unless you need to make any changes for your application, press Q to exit.

Then clean and run the build as shown below. Note that you may need to run fullclean as shown, rather than clean, depending on the path where you have downloaded the example code to.

```
C:\EVE-MCU-BRT_AN_025\examples\simple\ESP32>idf.py fullclean
```

```
C:\EVE-MCU-BRT_AN_025\examples\simple\ESP32>idf.py build
```

To program the module, connect it to the USB port of the PC and note the COM port assigned by checking the device manager (see the **Ports (COM & LPT)** section). In this example the module had been assigned COM3.

```
C:\EVE-MCU-BRT_AN_025\examples\simple\ESP32>idf.py -p COM3 flash
```

The module will reset, and the application will begin running.

7.4.3 Code

For the ESP32, the main.c file calls the eve_example function from file eve_example.c. The main.c file also has functions for the storage of the calibration transforms.

Note that the code in the MCU specific file EVE_MCU_ESP32.c is enabled via PLATFORM_ESP32. It is recommended to check that this is defined. The provided CMakeLists.txt has this defined as shown below but if incorporating the BRT_AN_025 code into another project, check if this is defined.

```
C:\EVE-MCU-BRT_AN_025\ports\eve_arch_esp32\CMakeLists.txt
```

7.5 MSP430 MCU Set-up

7.5.1 Hardware

The development board used was the MSP-EXP430G2 LaunchPad Development kit based on the MSP430G2553, the connections between this and an ME812A-WH50R are shown in Figure 7. Ensure that the overall current drawn by the EVE module is within the limits of the current which the MSP board can provide especially if powering over USB. If the current exceeds this limit, a separate power connection to the EVE module could be used, with the grounds of the MSP and EVE common to both power sources.

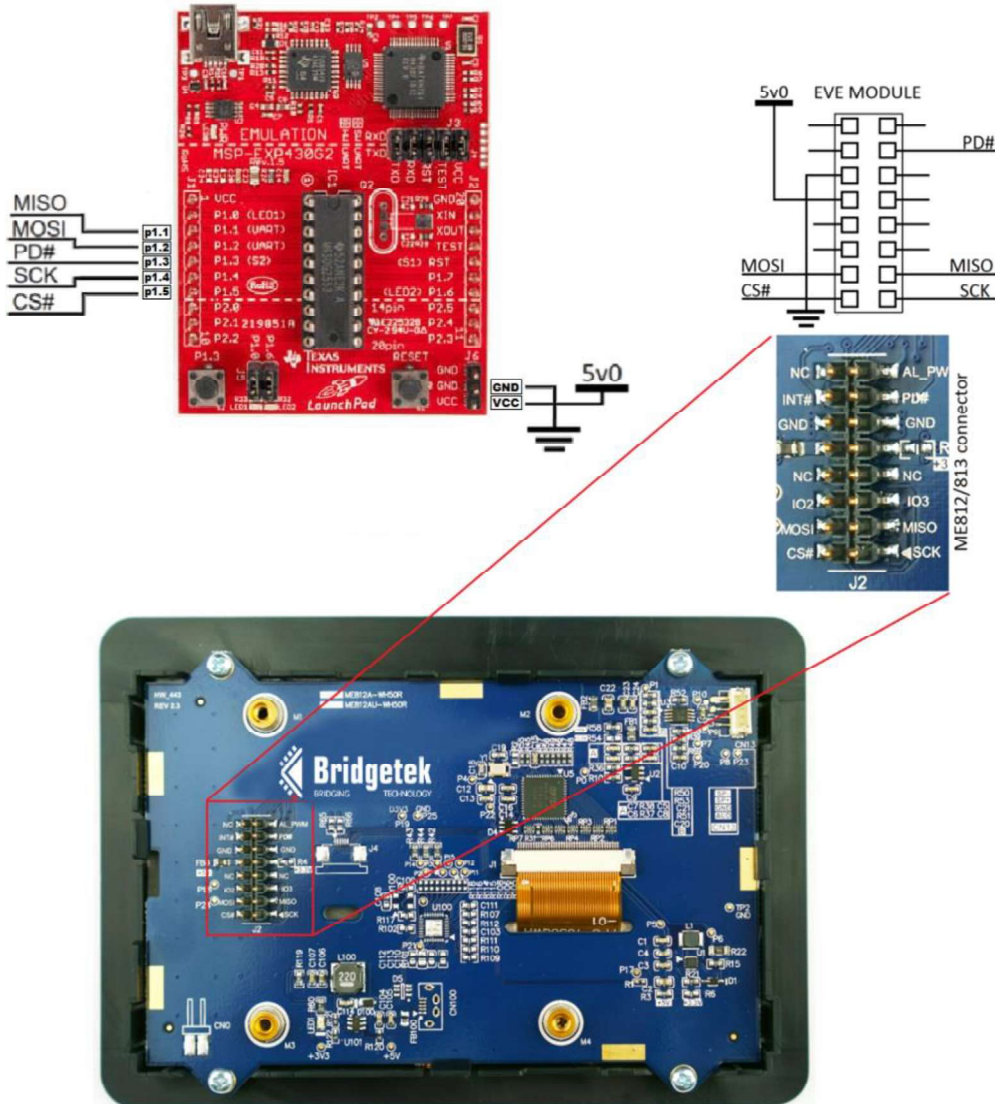


Figure 20 – MSP430 Hardware

7.5.2 Software Tools

This example was created using the Code Composer Studio IDE, which can be found at this link:
<http://www.ti.com/tool/CCSTUDIO#>

This project can be loaded by opening Code Composer Studio, selecting File > Import > CSS Projects and choosing the MSP430 folder.

7.5.3 Code

For the MSP430, the main.c file calls the eve_example function from file eve_example.c which runs the application thereafter.

Note that the code in the MCU specific file EVE_MCU_MSP430.c is enabled via PLATFORM_MSP430. It is recommended to check that this is defined as a preprocessor symbol. Open the project properties and check the Predefined Symbols tab. The Predefined Symbols section should have PLATFORM_MSP430 included as shown below.

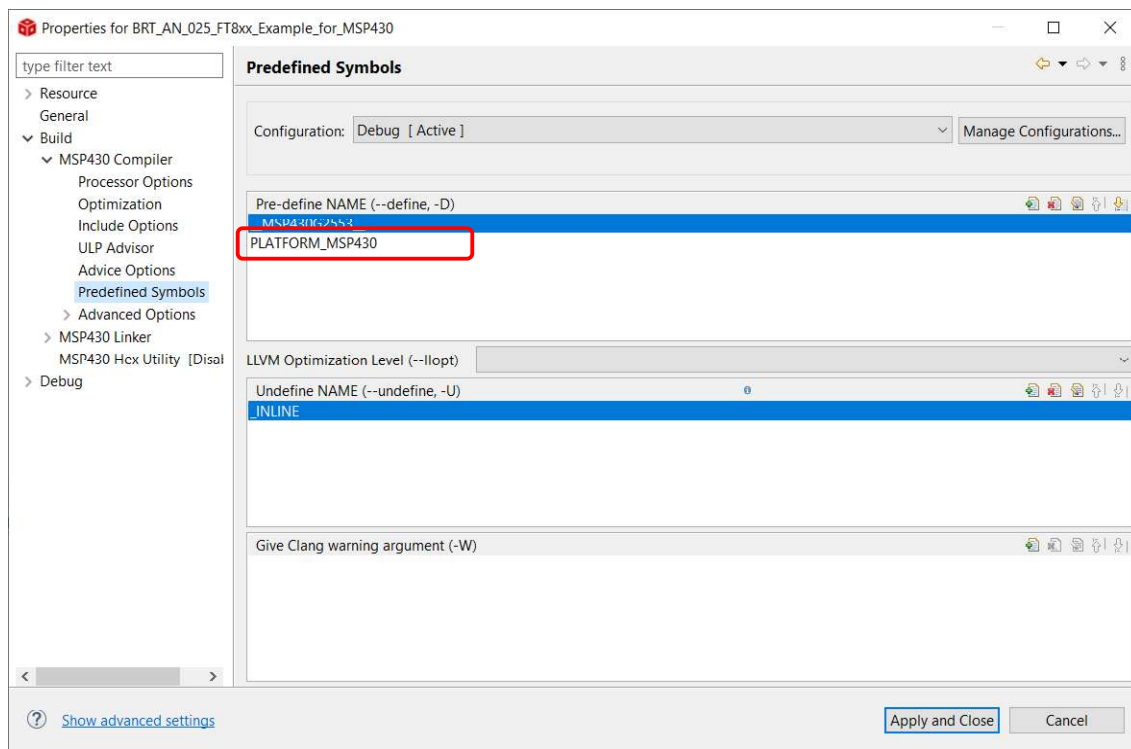


Figure 21 – Platform setting

7.6 BeagleBone Black Set-up

7.6.1 Hardware

The BeagleBone Black board should be connected as shown below. In this example the EVE module is an ME812A-WH50R.

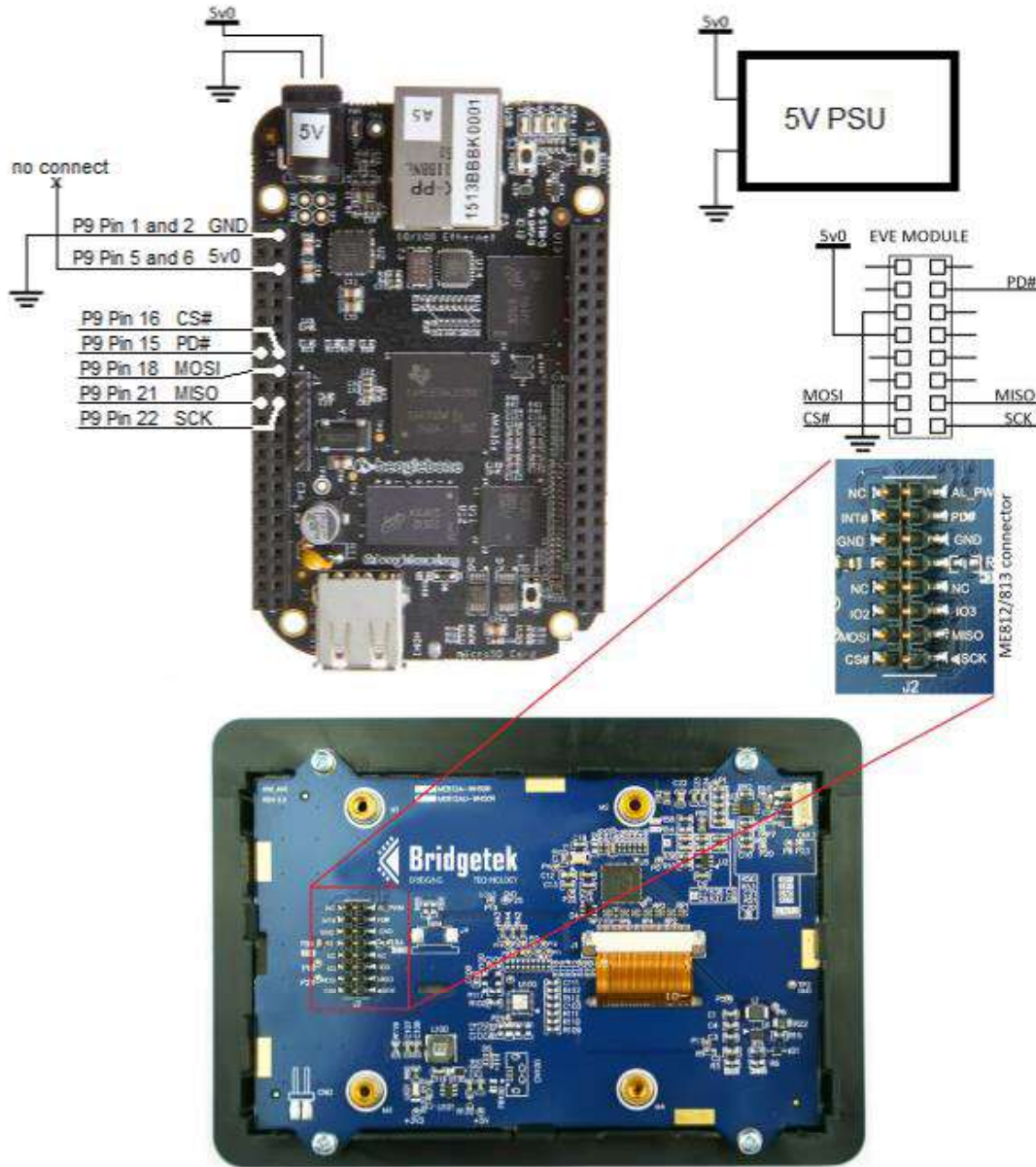


Figure 22 – BeagleBone Black Hardware

Note that the BeagleBone was powered via its 5V DC barrel connector. However, 5V power for the display was taken from a separate connection to the power supply to avoid drawing too much current through the BeagleBone. If using separate power supplies, the BeagleBone and the EVE supplies must have their GND connections common so that both the EVE module and the BeagleBone have the same GND level. This will avoid damage or communication problems which could result if the grounds are not connected and at the same potential.

Beaglebone Pin	BeagleBone signal	EVE Signal
P9 pins 1 and 2	GND	Ground
P9 pins 5 and 6	+5V out	5V supply to EVE module
P9 pin 15	GPIO 48	PD# signal
P9 pin 16	GPIO 51 (see note)	SPI Chip Select
P9 pin 18	SPIO_D1	SPI MOSI
P9 pin 21	SPIO_D0	SPI MISO
P9 pin 22	SPIO_SCLK	SPI SCK

Note: This is a GPIO line and not the SPI0 chip select line

Table 1 – BeagleBone Black connections

7.6.2 Code

The code uses the platform file [EVE_Linux_BBB.c](#) in the folder shown below.

```
EVE-MCU-BRT_AN_025\ports\eve_arch_beaglebone
```

Because the download from GitHub includes platform files for many different hosts and MCUs, the makefile defines the Beaglebone platform as well as the LINUX_SPI_DEV which enables the code in the Linux version of EVE_HAL (EVE_HAL_Linux.c).

```
CFLAGS = -c -Wall -g -Os -DPLATFORM_BEAGLEBONE -DUSE_LINUX_SPI_DEV -
I../../../../include -I../common
```

If modifying the project or incorporating the code into another project for the Beaglebone platform, ensure to define these so that the platform is set up correctly.

7.6.3 Running the example

To run the example code, copy the EVE_MCU_BRT_AN_025 folder to the BeagleBoneBlack. If the overall folder is too large, you can remove the folders for the other MCU platforms within examples\simple leaving only the BeagleBone and the common folders.

Then browse to this folder (assuming the EVE_MCU_BRT_AN_025 folder is on your BeagleBone desktop)

```
debian@beaglebone:~$ cd Desktop/
```

Then, run the makefile

Then run the code as sudo.

The application will then run and appear on the attached EVE screen.

```
debian@beaglebone: ~/Desktop/EVE-MCU-BRT_AN_025/examples/simple/BeagleBone
debian@beaglebone:~/Desktop$ cd Desktop/
debian@beaglebone:~/Desktop$ cd EVE-MCU-BRT_AN_025/examples/simple/BeagleBone/
debian@beaglebone:~/Desktop/EVE-MCU-BRT_AN_025/examples/simple/BeagleBone$ ls
Makefile  main
debian@beaglebone:~/Desktop/EVE-MCU-BRT_AN_025/examples/simple/BeagleBone$ make
gcc -c -Wall -g -Os -DPLATFORM_BEAGLEBONE -DUSE_LINUX_SPI_DEV -I../../../../include -I../common main/main.c -o main/main.o
gcc -c -Wall -g -Os -DPLATFORM_BEAGLEBONE -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../common/eve_fonts.c -o ../common/eve_fonts.o
gcc -c -Wall -g -Os -DPLATFORM_BEAGLEBONE -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../common/eve_helper.c -o ../common/eve_helper.o
gcc -c -Wall -g -Os -DPLATFORM_BEAGLEBONE -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../common/eve_example.c -o ../common/eve_example.o
gcc -c -Wall -g -Os -DPLATFORM_BEAGLEBONE -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../common/eve_calibrate.c -o ../common/eve_calibrate.o
gcc -c -Wall -g -Os -DPLATFORM_BEAGLEBONE -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../common/eve_images.c -o ../common/eve_images.o
gcc -c -Wall -g -Os -DPLATFORM_BEAGLEBONE -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../source/EVE_HAL.c -o ../source/EVE_HAL.o
gcc -c -Wall -g -Os -DPLATFORM_BEAGLEBONE -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../source/EVE_HAL_Linux.c -o ../source/EVE_HAL_Linux.o
gcc -c -Wall -g -Os -DPLATFORM_BEAGLEBONE -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../source/EVE_API.c -o ../source/EVE_API.o
gcc -c -Wall -g -Os -DPLATFORM_BEAGLEBONE -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../ports/eve_arch_beaglebone/EVE_Linux_BBB.c -o ../ports/eve_arch_beaglebone/EVE_Linux_BBB.o
../../../../ports/eve_arch_beaglebone/EVE_Linux_BBB.c:53:9: note: #pragma message: Compiling ../../ports/eve_arch_beaglebone/EVE_Linux_BBB.c for Beaglebone Black
#pragma message "Compiling " _FILE_ " for Beaglebone Black"
gcc -o BeagleBone main/main.o ../common/eve_fonts.o ../common/eve_helper.o ../common/eve_example.o ../common/eve_calibrate.o ../common/eve_images.o ../source/EVE_HAL.o ../source/EVE_HAL_Linux.o ../source/EVE_API.o ../ports/eve_arch_beaglebone/EVE_Linux_BBB.o
debian@beaglebone:~/Desktop/EVE-MCU-BRT_AN_025/examples/simple/BeagleBone$ ls
BeagleBone  Makefile  main
debian@beaglebone:~/Desktop/EVE-MCU-BRT_AN_025/examples/simple/BeagleBone$ sudo ./BeagleBone
Activating...done
^Cdebian@beaglebone:~/Desktop/EVE-MCU-BRT_AN_025/examples/simple/BeagleBone$
```

Figure 23 – BeagleBone

7.7 Raspberry Pi Set-up

7.7.1 Hardware

The Raspberry Pi should be connected as shown below. In this case the EVE module used is an ME812A-WH50R.

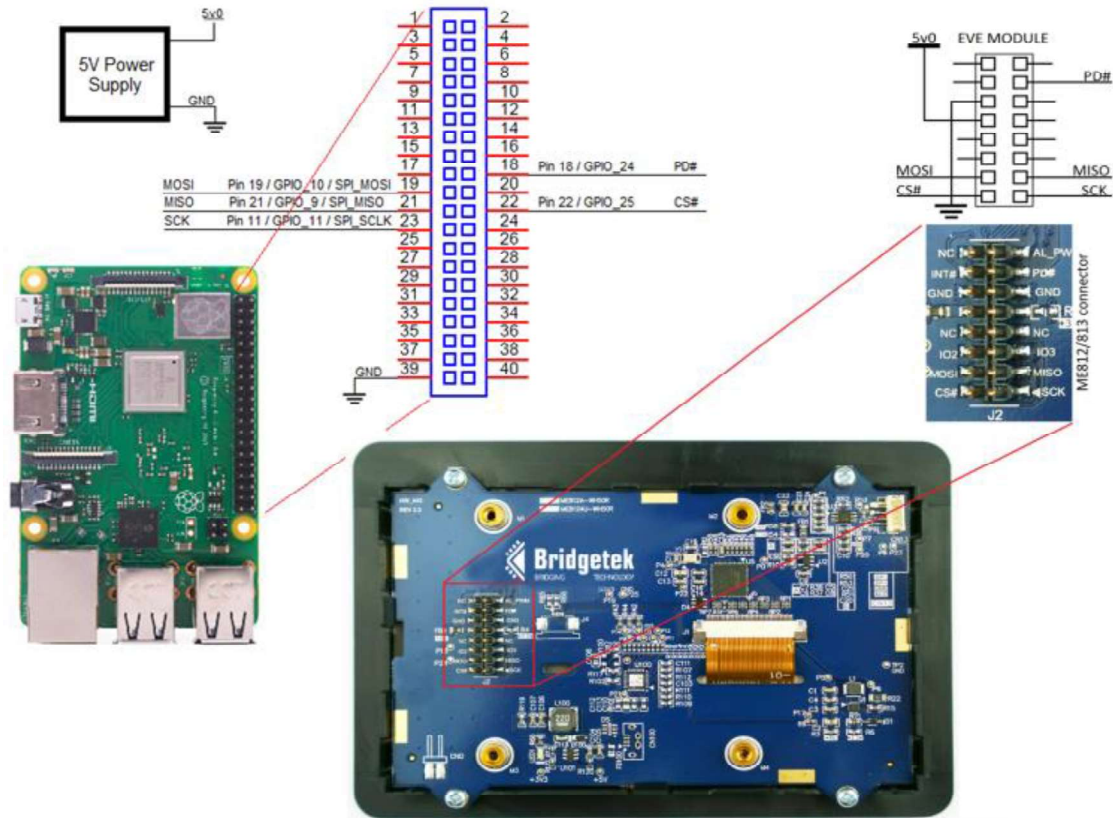


Figure 24 – Raspberry Pi Hardware

Note that the Raspberry Pi was powered via its USB micro-B connector. However, 5V power for the display was taken from a separate power supply to avoid drawing too much current through the Raspberry Pi. The power supply must have its GND common to the EVE module and the Raspberry Pi to avoid damage or communication problems which could result if the grounds are not connected and at the same potential.

Raspberry Pi Pin Header	Raspberry Pi signal	EVE Signal
Pin 39	GND	Ground
Pin 18	GPIO 24	PD# signal
Pin 19	GPIO 10 / SPI_MOSI	SPI MOSI
Pin 21	GPIO 9 / SPI_MISO	SPI MISO
Pin 22	GPIO 25 (see note)	SPI CS#
Pin 23	GPIO 11 / SPI_SCLK	SPI SCK

Note: This is a GPIO line and not the SPI chip select line

Table 2 – Raspberry Pi connections

7.7.2 Code

The code uses the platform file [EVE_Linux_RPi.c](#) in the folder shown below.

EVE-MCU-BRT_AN_025\ports\eve_arch_rpi

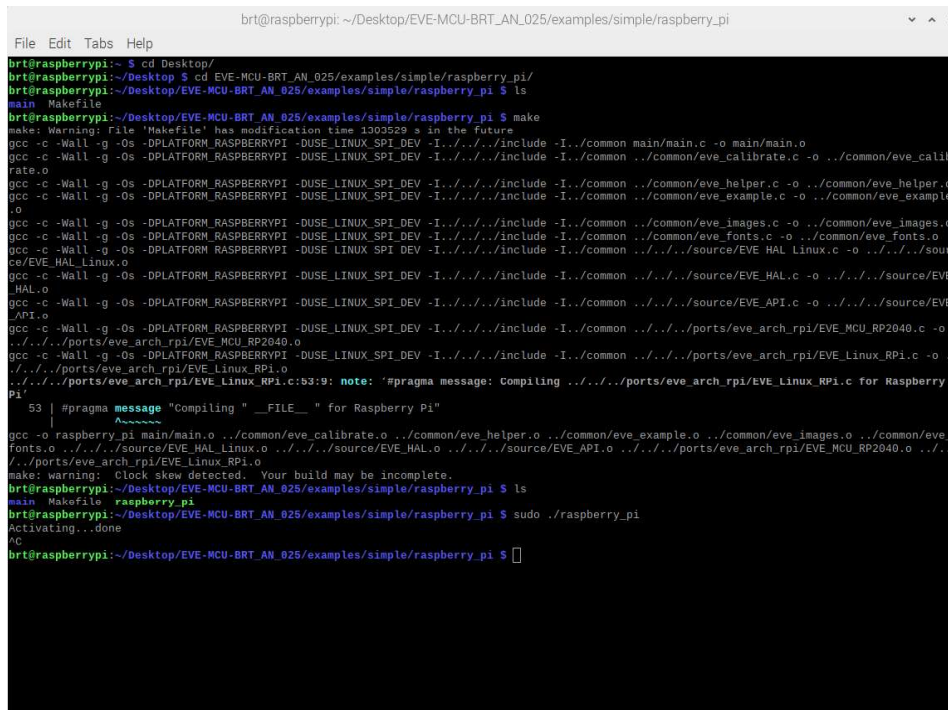
Because the download from GitHub includes platform files for many different hosts and MCUs, the makefile defines the Raspberry Pi platform as well as the LINUX_SPI_DEV which enables the code in the Linux version of EVE_HAL (EVE_HAL_Linux.c).

```
CFLAGS = -c -Wall -g -Os -DPLATFORM_RASPBERRYPI -DUSE_LINUX_SPI_DEV -
I../../../../include -I../common
```

If modifying the project or incorporating the code into another project for the RPi platform, ensure to define these so that the platform is set up correctly.

7.7.3 Running the example

To run the example code, copy the entire EVE-MCU-BRT_AN_025 folder to the Raspberry Pi, run the makefile and then run as sudo. The application will then run and appear on the attached EVE screen. Note that you may wish to remove the folders for the other MCU platforms within examples\simple leaving only the raspberry_pi and the common folders.



```
brt@raspberrypi: ~/Desktop/EVE-MCU-BRT_AN_025/examples/simple/raspberry_pi
brt@raspberrypi:~/Desktop/EVE-MCU-BRT_AN_025/examples/simple/raspberry_pi$ ls
main Makefile
brt@raspberrypi:~/Desktop/EVE-MCU-BRT_AN_025/examples/simple/raspberry_pi$ make
make: Warning: File 'Makefile' has modification time 1300529 s in the future
gcc -c -Wall -g -Os -DPLATFORM_RASPBERRYPI -DUSE_LINUX_SPI_DEV -I../../../../include -I../common main/main.c -o main/main.o
gcc -c -Wall -g -Os -DPLATFORM_RASPBERRYPI -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../common/eve_calibrate.c -o ../common/eve_calibrate.o
gcc -c -Wall -g -Os -DPLATFORM_RASPBERRYPI -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../common/eve_helper.c -o ../common/eve_helper.o
gcc -c -Wall -g -Os -DPLATFORM_RASPBERRYPI -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../common/eve_example.c -o ../common/eve_example.o
gcc -c -Wall -g -Os -DPLATFORM_RASPBERRYPI -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../common/eve_images.c -o ../common/eve_images.o
gcc -c -Wall -g -Os -DPLATFORM_RASPBERRYPI -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../common/eve_fonts.c -o ../common/eve_fonts.o
gcc -c -Wall -g -Os -DPLATFORM_RASPBERRYPI -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../../../../source/EVE_HAL_Linux.c -o ../../../../source/EVE_HAL_Linux.o
gcc -c -Wall -g -Os -DPLATFORM_RASPBERRYPI -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../../../../source/EVE_HAL.c -o ../../../../source/EVE_HAL.o
gcc -c -Wall -g -Os -DPLATFORM_RASPBERRYPI -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../../../../source/EVE_API.c -o ../../../../source/EVE_API.o
gcc -c -Wall -g -Os -DPLATFORM_RASPBERRYPI -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../../../../ports/eve_arch_rpi/EVE_MCU_RP2040.c -o ../../../../ports/eve_arch_rpi/EVE_MCU_RP2040.o
gcc -c -Wall -g -Os -DPLATFORM_RASPBERRYPI -DUSE_LINUX_SPI_DEV -I../../../../include -I../common ../../../../ports/eve_arch_rpi/EVE_Linux_RPi.c -o ../../../../ports/eve_arch_rpi/EVE_Linux_RPi.o
../../../../ports/eve_arch_rpi/EVE_Linux_RPi.c:b3:9: note: '#pragma message: Compiling ../../../../ports/eve_arch_rpi/EVE_Linux_RPi.c for Raspberry Pi'
 53 | #pragma message "Compiling " __FILE__ " for Raspberry Pi"
     | ~~~~~
gcc -o raspberry_pi main/main.o ../common/eve_calibrate.o ../common/eve_helper.o ../common/eve_example.o ../common/eve_images.o ../common/eve_fonts.o ../../../../source/EVE_HAL_Linux.o ../../../../source/EVE_HAL.o ../../../../source/EVE_API.o ../../../../ports/eve_arch_rpi/EVE_MCU_RP2040.o ../../../../ports/eve_arch_rpi/EVE_Linux_RPi.o
make: warning: Clock skew detected. Your build may be incomplete.
brt@raspberrypi:~/Desktop/EVE-MCU-BRT_AN_025/examples/simple/raspberry_pi$ ls
main Makefile raspberry_pi
brt@raspberrypi:~/Desktop/EVE-MCU-BRT_AN_025/examples/simple/raspberry_pi$ sudo ./raspberry_pi
Activating...done
AC
brt@raspberrypi:~/Desktop/EVE-MCU-BRT_AN_025/examples/simple/raspberry_pi$
```

Figure 25 – Raspberry Pi

7.8 Raspberry Pi Pico Set-up

7.8.1 Hardware

The Raspberry Pi Pico should be connected as shown below. In this case the EVE module used is an ME812A-WH50R.

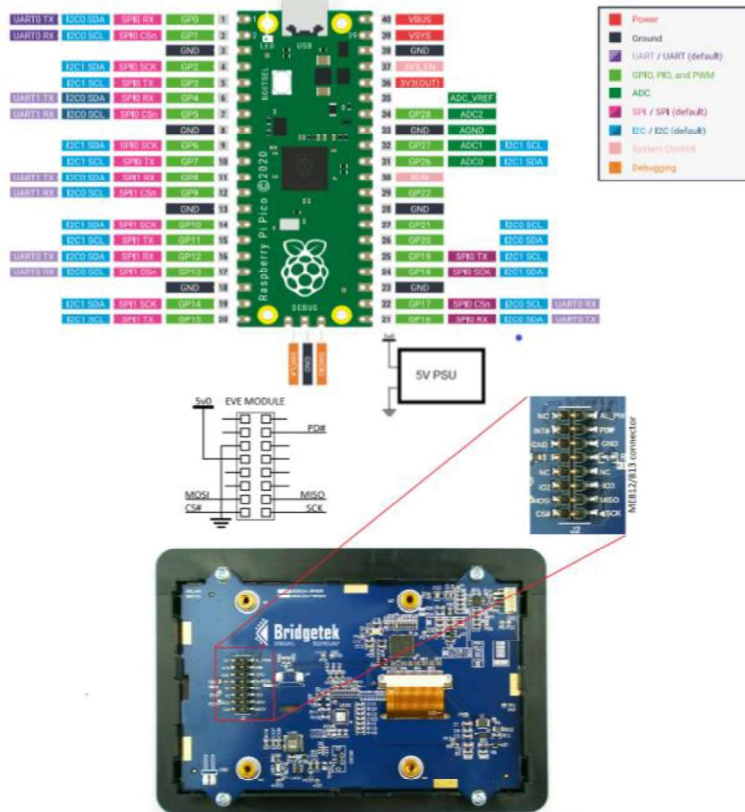


Figure 26 – Raspberry Pi Pico

Note that other RP2040-based boards can also be used. This includes the [Bridgetek MM2040EV](#) which has connectors for the 8x2 and 10x1 headers used on Bridgetek EVE display modules, as well as the [Bridgetek IDM2040-7A](#) which has the RP2040 IC onboard along with a 7" capacitive screen and wide range of interfaces including DMX and BRT Systems LDSBus.

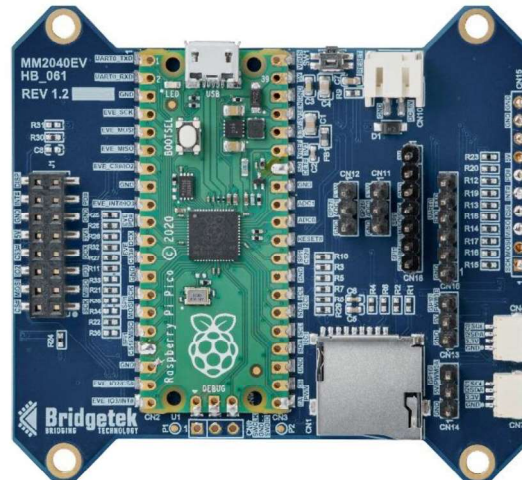


Figure 27 - Bridgetek MM2040EV with RP2040 Pico module onboard

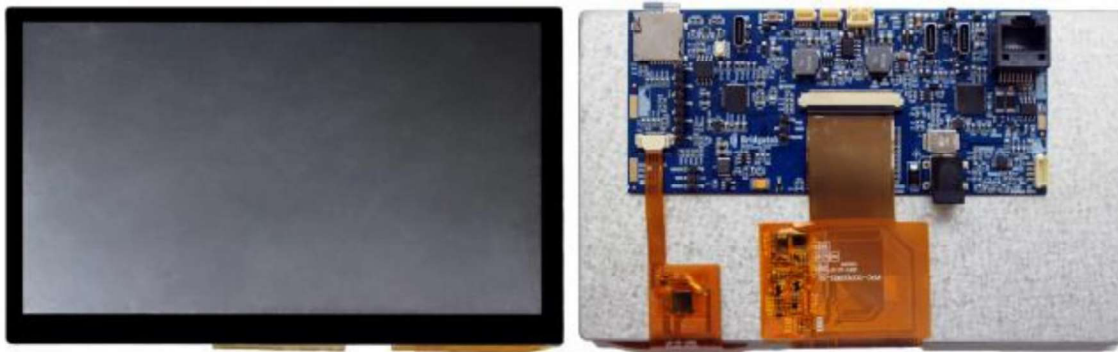


Figure 28 - Bridgetek IDM2040-7A front and back

7.8.2 Running the example

To run the example code, hold down the Boot Select pushbutton on the Pico module whilst connecting the module to the PC, and then release the button once connected. A window will appear like a memory stick being plugged in. The uf2 file can then be dragged into the window like copying a file into a flash drive. The Pico will reset and the code will start to run.

Before the file can be loaded as described above, the uf2 file must first be built. The Raspberry Pi Pico toolchain should be installed to do this. The following guide from the manufacturer of the Raspberry Pi Pico contains details of how to do this on various OS, and the installation procedure should be followed fully. You can use a Raspberry Pi to build the code on Linux or you can use the tools on Windows (at the time of writing this was chapter 9.2 of the guide below). In this case, Windows was used to build the file.

[Getting started with Raspberry Pi Pico](#)

Note that Bridgetek accept no responsibility whatsoever for any content from 3rd party providers, including the websites, documents, and utilities, or for any consequences of using the 3rd party tools and procedures.

In this case, the BRT_AN_025 code was downloaded to the C drive from Github

A folder called `build` was created within the `C:\EVE-MCU-BRT_AN_025\examples\simple\pico` folder

Note: It is important to ensure that the Pico SDK and Toolchain are set up in the environment variables, for example as shown below (the paths set will depend on the locations of your toolchain and SDK)

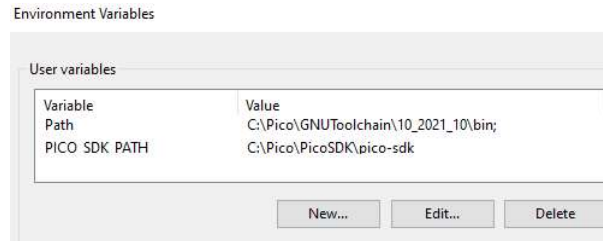


Figure 29 - Pico toolchain / SDK Environment Variables

Open the developer command prompt which was installed as part of the Windows install procedure.

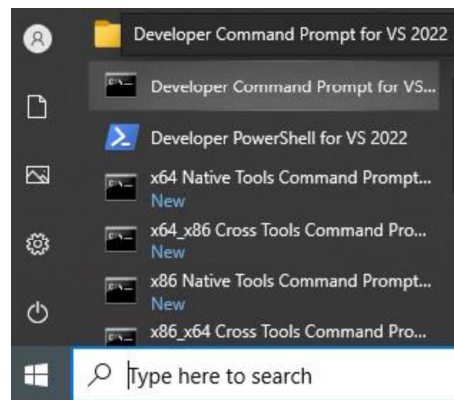


Figure 30 - Open Developer Command Prompt for VS 2022

Use the following commands to build the code.

```
C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools>cd ..
```

```
C:\Program Files (x86)\Microsoft Visual Studio\2022>cd ..
```

```
C:\Program Files (x86)\Microsoft Visual Studio>cd ..
```

```
C:\Program Files (x86)>cd ..
```

Browse to the pico directory on your computer.

```
C:\>cd C:\EVE-MCU-BRT_AN_025\examples\simple\pico
```

Create a folder named build and then browse to the build folder

```
C:\>cd C:\EVE-MCU-BRT_AN_025\examples\simple\pico> mkdir build
```

```
C:\EVE-MCU-BRT_AN_025\examples\simple\pico>cd build
```

Use the command `cmake -G "NMake Makefiles" ..` and you will see an output like below.

```
C:\EVE-MCU-BRT_AN_025\examples\simple\pico\build>cmake -G "NMake Makefiles" ..
```

```
Using PICO_SDK_PATH from environment ('C:\Pico\PicoSDK\pico-sdk')
PICO_SDK_PATH is C:/Pico/PicoSDK/pico-sdk
Defaulting PICO_PLATFORM to rp2040 since not specified.
Defaulting PICO platform compiler to pico_arm_gcc since not specified.
-- Defaulting build type to 'Release' since not specified.
PICO compiler is pico_arm_gcc
-- The C compiler identification is GNU 10.3.1
```

```
-- The CXX compiler identification is GNU 10.3.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Pico/GNUToolchain/10_2021_10/bin/arm-none-eabi-gcc.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Pico/GNUToolchain/10_2021_10/bin/arm-none-eabi-g++.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- The ASM compiler identification is GNU
-- Found assembler: C:/Pico/GNUToolchain/10_2021_10/bin/arm-none-eabi-gcc.exe
Build type is Release
PICO target board is pico.
Using board configuration from C:/Pico/PicoSDK/pico-sdk/src/boards/include/boards/pico.h
-- Found Python3: C:/Users/#####/Programs/Python/Python311/python.exe (found version "3.11.1") found components: Interpreter
TinyUSB available at C:/Pico/PicoSDK/pico-sdk/lib/tinyusb/src/portable/raspberrypi/rp2040; enabling build support for USB.
BTstack available at C:/Pico/PicoSDK/pico-sdk/lib/btstack
cyw43-driver available at C:/Pico/PicoSDK/pico-sdk/lib/cyw43-driver
Pico W Bluetooth build support available.
lwIP available at C:/Pico/PicoSDK/pico-sdk/lib/lwip
mbedtls available at C:/Pico/PicoSDK/pico-sdk/lib/mbedtls
-- Configuring done (4.8s)
-- Generating done (0.5s)
-- Build files have been written to: C:/EVE-MCU-BRT_AN_025/examples/simple/pico/build
```

Now run **nmake** to make the project.

```
c:\EVE-MCU-BRT_AN_025\examples\simple\pico\build>nmake
```

```
Microsoft (R) Program Maintenance Utility Version 14.33.31630.0
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
[ 0%] Building ASM object pico-
sdk/src/rp2_common/boot_stage2/CMakeFiles/bs2_default.dir/compile_time_choice.S.obj
[ 1%] Linking ASM executable bs2_default.elf
[ 1%] Built target bs2_default
[ 1%] Generating bs2_default.bin
[ 2%] Generating bs2_default_padded_checksummed.S
[ 2%] Built target bs2_default_padded_checksummed_asm
[ 3%] Building C object CMakeFiles/eve_library.dir/C_/ftdigdm__EVE-MCU-
BRT_AN_025/source/EVE_API.c.obj
[ 3%] Building C object CMakeFiles/eve_library.dir/C_/ftdigdm__EVE-MCU-
BRT_AN_025/source/EVE_HAL.c.obj
[ 4%] Building C object CMakeFiles/eve_library.dir/C_/ftdigdm__EVE-MCU-
BRT_AN_025/ports/eve_arch_rpi/EVE_MCU_RP2040.c.obj
```

[remainder of log output]

```
[100%] Linking CXX executable pico.elf
      text      data      bss      dec      hex filename
```



```
35808 40 3492 39340 99ac pico.elf
[100%] Built target pico
```

```
C:\EVE-MCU-BRT_AN_025\examples\simple\pico\build>
```

The uf2 file will then be generated in the build folder which was created in the earlier step at this path: C:\EVE-MCU-BRT_AN_025\examples\simple\pico\build>

Note that the folder and path may vary if you have used different names for your folders in the earlier steps above.

Now load this uf2 file using the procedure explained at the top of this section (see 7.8.2)



Figure 31 - uf2 file generated in the build folder

7.9 MPSSE Set-up (USB-SPI with Visual Studio code)

7.9.1 Hardware

The code runs on Visual Studio and requires a USB-MPSSE adapter to connect to the EVE module.

The USB-MPSSE adapter is normally based around the FTDI FT232H USB bridging chip.

A popular solution which makes it easy to get up and running quickly is the range of [C232HM](#) cables (external link) such as C232HM-EDHSL. These can also be purchased from [Connective Peripherals](#) (external link) as a version with Type C USB connector (part USBC-HS-MPSSE-5V-3.3V-500-SPR).



Figure 32 - MPSSE Cables with Type A and Type C USB connectors

The C232HM uses the FT232H chipset and has a 50cm cable with ten single pole receptacles. These can be connected to the associated pins of the EVE module.

For the versions with 5V output on the red wire (C232HM-EDHSL and USBC-HS-MPSSE-5V-3.3V-500-SPR) the following wires can be used in Table 3.

If the version of the cable is used which has the 3.3V power output (or if using the 5V version but the EVE module requires more current than can be provided) then a separate power connection to

the EVE module could be used, with the grounds of the C232HM / USBC-HS-MPSSE and the separate power source common together to EVE.

Wire color	EVE Signal
Orange	SCK
Yellow	MOSI
Green	MISO
Brown	CS#
Blue	PD#
Red	5V
Black	GND

Table 3 – MPSSE connections

The VA800A-SPI module can also be used in the same way as the C232HM. *Note that the VA800A-SPI is now discontinued but the information is retained here for reference.*



Figure 33 - VA800A-SPI

The [VA800A-SPI](#) has a 10-way connector and can be connected directly to some EVE modules such as the VM800B, VM810C50A and VM816C50A. Very short jumper wires can be used to connect it to other EVE modules by connecting the SCK, CS, MOSI, MISO, PD and GND connections. The adapter also provides a 5V power supply via the 5V pin on the header. Ensure that the computer has sufficient power to supply the EVE module when the backlight is on (and if the audio amplifier is used). If the PC cannot provide sufficient power, a separate power connection to the EVE module could be used, with the grounds of the VA800A-SPI and the separate power source common together to EVE.

7.9.2 Software Tools

This example can be opened using Visual Studio 2022, including the Community version.

7.9.3 Code

The LibMPSSE files need to be added, please check on the FTDIchip site for the latest versions. At the time of writing the version 1.0.3 was used. These are added to the folder as shown below.

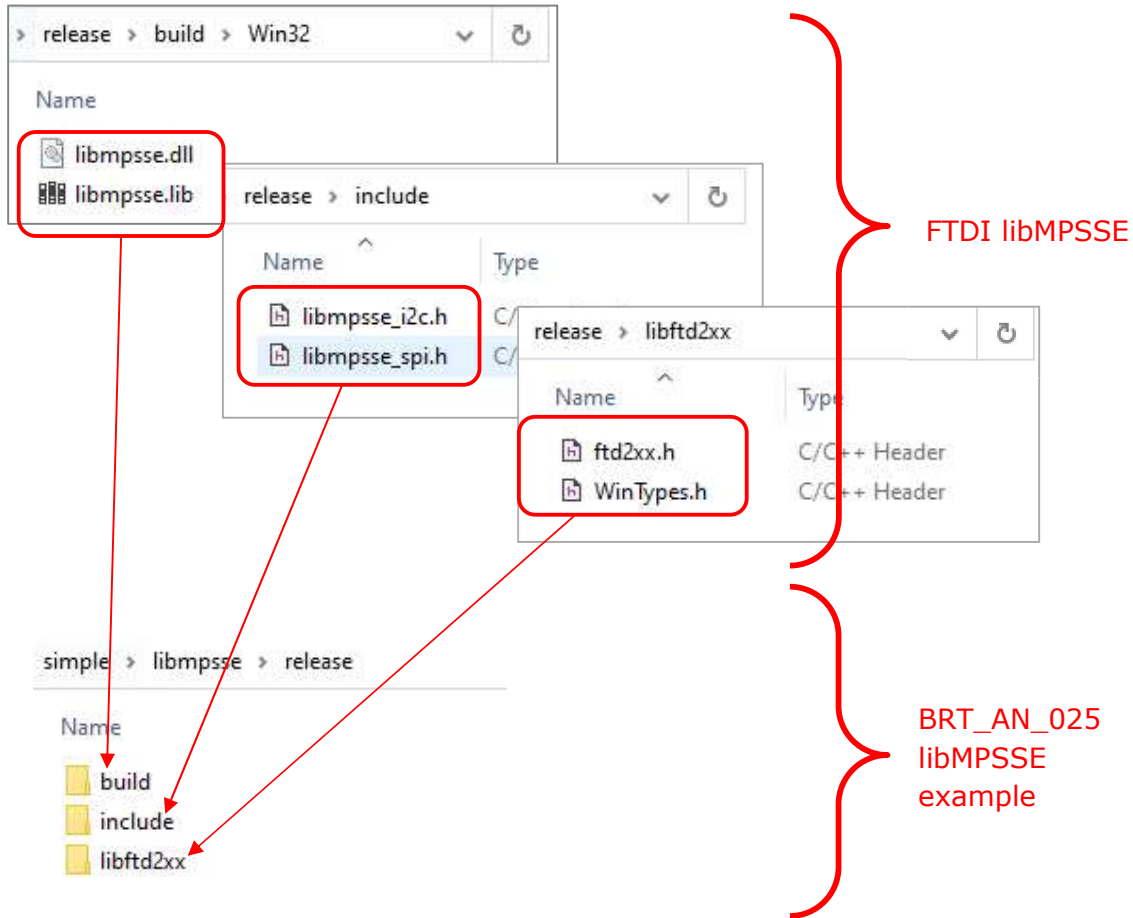


Figure 34 - Copying files from libMPSSE into BRT_AN_025

```

\examples\simple\libmpsse\release\
  build
    libmpsse.dll
    libmpsse.lib
  include
    libmpsse_i2c.h
    libmpsse_spi.h
  libftd2xx
    ftd2xx.h
    WinTypes.h
  
```

To update the libMPSSE files, go to the FTDI Chip site (www.ftdichip.com) and download the latest libMPSSE_Source zip file.

<https://ftdichip.com/software-examples/mpsse-projects/libmpsse-spi-examples/>

Create a folder in your EVE-MCU-BRT_AN_025\examples\simple\libmpsse folder called "release"
 Un-zip the libMPSSE file and copy the three folders and associated files shown above from the FTDI libMPSSE to existing ones in your BRT_AN_025 \examples\simple\libmpsse\release\ folder as illustrated in Figure 34.

To run the example, open the solution using the libMPSSE.sln file, which can be found in the VisualStudio folder.

Check that the USE_MPSSE is defined in the Preprocessor Definitions as shown below. The definition itself is used to enable the platform-specific code for MPSSE and the value assigned here (= 0) will also be used as the channel index (0-based) when opening the port.

Assuming the computer has a single FT232H (VA800A-SPI or C232HM) then there is only one MPSSE and so a value of 0 is used. If the computer had an FT232H-based module acting as USB-SPI then set USE_MPSSE = 0 for channel A or USE_MPSSE = 1 if channel B was to be used. Most applications will use an FT232H based adapter and so a value of 0 is most often used.

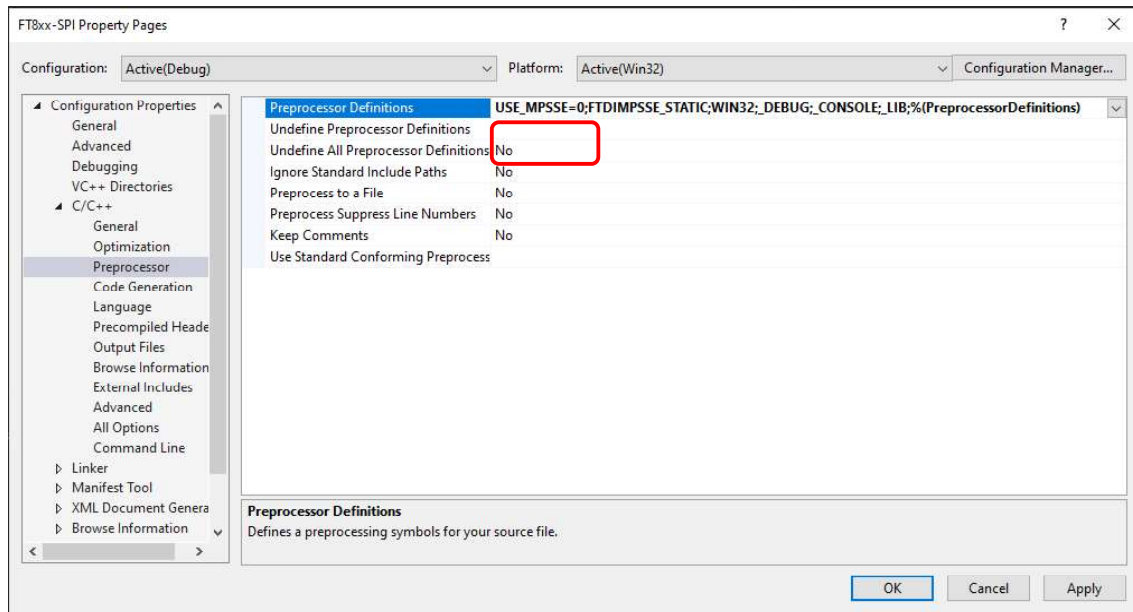


Figure 35 - Selecting Preprocessor Definitions

Then right-click on the solution and select rebuild to build the project.

Select Debug -> Start New Instance to begin running the code.

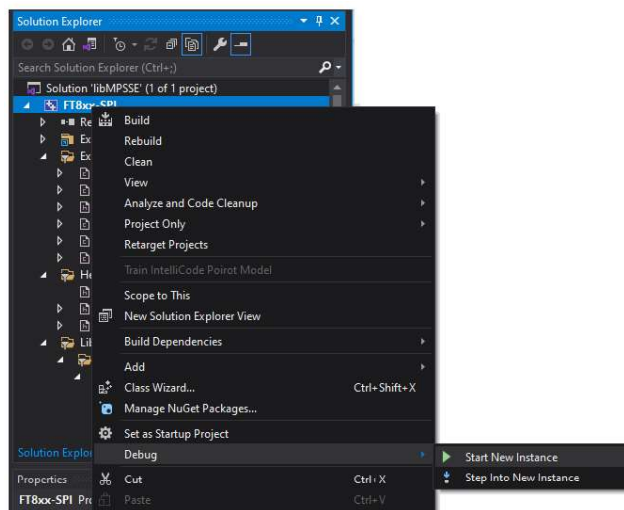


Figure 36 - Running the example

8 Conclusion

This application note is intended to help developers add support for the EVE family of devices to their preferred MCU.

It provides examples for a range of MCUs and includes a sample main application as well as a series of underlying code and header files providing the transport layers to interface to the MCU's SPI Master peripheral.

In addition to the MCU types provided here, the code can be ported to other MCUs using these examples as a reference. The generic library files should be able to be used directly on most toolchains and the MCU-specific files can be modified to suit the compiler and MCU hardware.

After including this framework, the product developer can create content on the EVE screen from their main application using commands in the user-friendly syntax of the EVE Programmers Guide. The provided example demonstrates key techniques such as creating a screen and loading fonts and images and can be extended to create a full application.

9 Contact Information

Refer to <https://brtchip.com/contact-us/> for contact information

Distributor and Sales Representatives

Please visit the Sales Network page of the [Bridgetek Web site](#) for the contact details of our distributor(s) and sales representative(s) in your country.

System and equipment manufacturers and designers are responsible to ensure that their systems, and any Bridgetek Pte Ltd (BRTChip) devices incorporated in their systems, meet all applicable safety, regulatory and system-level performance requirements. All application-related information in this document (including application descriptions, suggested Bridgetek devices and other materials) is provided for reference only. While Bridgetek has taken care to assure it is accurate, this information is subject to customer confirmation, and Bridgetek disclaims all liability for system designs and for any applications assistance provided by Bridgetek. Use of Bridgetek devices in life support and/or safety applications is entirely at the user's risk, and the user agrees to defend, indemnify, and hold harmless Bridgetek from any and all damages, claims, suits, or expense resulting from such use. This document is subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Neither the whole nor any part of the information contained in, or the product described in this document, may be adapted, or reproduced in any material or electronic form without the prior written consent of the copyright holder. Bridgetek Pte Ltd, 1 Tai Seng Avenue, Tower A, #03-05, Singapore 536464. Singapore Registered Company Number: 201542387H.

Appendix A– References

Document References

BRT_AN_025 source on GitHub	Bridgetek repositories on Github
BRT_AN_006	Low level details of EVE SPI transfers
BRT_AN_008	Usage of the MCU library
BRT_AN_014	Examples using the MCU library
EVE ICs and Modules on BRTchip	
BRT_AN_062 - Porting guide	Guide showing how to port BRT_AN_025 to other host platforms which are not already included in BRT_AN_025
EVE Examples	EVE Examples on Bridgetek website

Acronyms and Abbreviations

Terms	Description
EVE	Embedded Video Engine
MCU	Microcontroller
FT80x	First generation of EVE (FT800 and FT801)
FT81x	Second generation of EVE (FT810, FT811, FT812, FT813)
BT88x	New second-generation EVE devices (BT880, BT881, BT882, BT883)
BT81x	Third generation (BT815/6) and fourth generation of EVE (BT817/8)
LCD	Liquid Crystal Display
SPI	Serial Peripheral Interface
QSPI	Quad SPI
CS#	SPI Chip Select signal (Active low)
SCK	SPI Clock
MOSI	SPI Master Out Slave In data line carries data from MCU to EVE
MISO	SPI Master In Slave Out data line carries data from EVE to MCU
UART	Universal Asynchronous Receiver Transmitter for serial data transfer


```
0,0,3,63,255,0,0,0,0,1,63,255,0,0,0,0,0,127,255,192,0,0,0,0,255,255,128,0,0,0,0,127,
255,32,0,0,0,0,63,254,112,0,0,0,0,0,248,0,0,0,0,1,240,0,0,0,0,0,1,248,0,0,
0,0,0,1,240,0,0,0,0,0,3,240,0,0,0,0,0,1,240,0,0,0,0,0,3,240,0,0,0,0,0,3,
240,0,0,0,0,1,240,0,0,0,0,1,240,0,0,0,0,63,252,240,0,0,0,0,127,252,112,0,0,0,
0,255,254,32,0,0,0,1,255,255,16,0,0,0,3,255,255,128,0,0,0,7,255,255,192,0,0,0,0,255,255,248,
0,0,0,0,127,255,240,0,0,0,2,63,255,224,0,0,0,3,31,255,192,0,0,0,3,159,255,128,0,0,0,3,
207,255,0,0,0,0,3,224,0,0,0,0,3,224,0,0,0,0,3,240,0,0,0,0,3,224,0,0,0,
0,0,7,224,0,0,0,0,3,224,0,0,0,0,7,224,0,0,0,0,7,224,0,0,0,0,7,224,
0,0,0,0,7,128,0,0,0,0,3,63,255,0,0,0,1,63,255,0,0,0,127,255,192,0,0,
0,0,255,255,128,0,0,0,2,127,255,32,0,0,0,7,63,254,112,0,0,0,7,128,0,248,0,0,0,15,192,1,
240,0,0,0,15,192,1,248,0,0,0,15,192,1,240,0,0,0,15,192,3,240,0,0,0,15,192,1,240,0,0,0,
15,128,3,240,0,0,0,15,192,3,240,0,0,0,15,128,1,240,0,0,0,15,128,1,240,0,0,0,15,191,252,240,
0,0,0,30,127,252,112,0,0,0,13,255,254,32,0,0,0,27,255,255,16,0,0,0,19,255,255,128,0,0,0,7,
255,255,192,0,0,0,255,255,248,0,0,0,127,255,244,0,0,0,63,255,236,0,0,0,0,31,255,220,0,
0,0,0,31,255,188,0,0,0,15,255,124,0,0,0,0,124,0,0,0,0,252,0,0,0,0,0,
0,252,0,0,0,0,252,0,0,0,252,0,0,0,248,0,0,0,252,0,0,
0,0,0,0,248,0,0,0,0,248,0,0,0,0,120,0,0,0,0,56,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,32,0,0,0,0,0,112,0,0,0,
0,0,0,248,0,0,0,0,1,240,0,0,0,1,248,0,0,0,0,1,240,0,0,0,0,1,240,
0,0,0,0,3,240,0,0,0,0,1,240,0,0,0,0,3,240,0,0,0,0,3,240,0,0,0,
0,1,240,0,0,0,0,1,240,0,0,0,0,240,0,0,0,0,96,0,0,0,0,48,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,248,0,0,0,127,255,246,0,0,0,2,63,
255,236,0,0,0,3,31,255,220,0,0,0,3,159,255,188,0,0,0,3,207,255,124,0,0,0,3,224,0,124,0,0,
0,3,224,0,252,0,0,0,3,240,0,252,0,0,0,3,224,0,252,0,0,0,7,224,0,248,0,0,0,3,224,0,
252,0,0,0,7,224,0,248,0,0,0,7,224,0,252,0,0,0,7,224,0,248,0,0,0,7,128,0,120,0,0,0,
3,63,255,48,0,0,0,1,63,255,144,0,0,0,127,255,128,0,0,0,255,255,128,0,0,0,2,127,255,32,
0,0,0,7,63,254,112,0,0,0,7,128,0,248,0,0,0,15,192,1,240,0,0,0,15,192,1,248,0,0,0,15,
192,1,240,0,0,0,15,192,3,240,0,0,0,15,192,1,240,0,0,0,15,128,3,240,0,0,0,15,192,3,240,0,
0,0,15,128,1,240,0,0,0,15,128,1,240,0,0,0,15,191,252,240,0,0,0,30,127,252,112,0,0,0,13,255,
254,32,0,0,0,27,255,255,16,0,0,0,19,255,255,128,0,0,0,7,255,255,192,0,0,0,255,255,248,0,0,
0,0,127,255,244,0,0,0,2,63,255,236,0,0,0,3,31,255,220,0,0,0,3,159,255,188,0,0,0,3,207,255,
124,0,0,0,3,224,0,124,0,0,0,3,224,0,252,0,0,0,3,240,0,252,0,0,0,3,224,0,252,0,0,0,
7,224,0,252,0,0,0,3,224,0,248,0,0,0,7,224,0,252,0,0,0,7,224,0,248,0,0,0,7,224,0,248,
0,0,0,7,128,0,120,0,0,0,3,63,255,56,0,0,0,1,63,255,128,0,0,0,127,255,128,0,0,0,0,
255,255,192,0,0,0,127,255,32,0,0,0,63,254,112,0,0,0,0,248,0,0,0,1,240,0,
0,0,0,1,248,0,0,0,0,1,240,0,0,0,0,3,240,0,0,0,0,1,240,0,0,0,0,
3,240,0,0,0,0,3,240,0,0,0,0,1,240,0,0,0,0,1,240,0,0,0,63,252,240,0,0,
0,0,127,252,112,0,0,0,0,255,254,32,0,0,0,1,255,255,16,0,0,0,3,255,255,128,0,0,0,7,255,255,
192,0,0,0,
/*Bitmap Raw Data end ---*/
};
```

Appendix C – List of Tables & Figures

List of Figures

Figure 1 - Layers of the software example.....	7
Figure 2 – Main application screen	10
Figure 3 – Converting the DIGITS font for the simplified example	13
Figure 4 – Creating your own application	21
Figure 5 – Enabling the FT9XX platform	32
Figure 6 – GitHub Repository for BRT_AN_025 code	34
Figure 7 – GitHub Repository - Downloading.....	34
Figure 8 – FT900 hardware	35
Figure 9 – Importing existing projects into workspace	36
Figure 10 – Selecting the project folder	36
Figure 11 – Platform setting	37
Figure 12 – Excluding ESP32 folder from FT900 build	38
Figure 13 – PIC Hardware using the ME812A-WH50R module	39
Figure 14 – Platform setting	40
Figure 15 – STM32 Hardware	41
Figure 16 – Platform setting	42
Figure 17 – ESP32 Hardware	43
Figure 18 – ESP32 menuconfig	44
Figure 19 – Platform setting	45
Figure 20 – MSP430 Hardware.....	46
Figure 21 – Platform setting	47
Figure 22 – BeagleBone Black Hardware	48
Figure 23 – BeagleBone	50
Figure 24 – Raspberry Pi Hardware	51
Figure 25 – Raspberry Pi.....	52
Figure 26 – Raspberry Pi Pico	53
Figure 27 - Bridgetek MM2040EV with RP2040 Pico module onboard	54
Figure 28 - Bridgetek IDM2040-7A front and back.....	54
Figure 29 - Pico toolchain / SDK Environment Variables	55
Figure 30 - Open Developer Command Prompt for VS 2022.....	55
Figure 31 - uf2 file generated in the build folder	57
Figure 32 - MPSSE Cables with Type A and Type C USB connectors	57
Figure 33 - VA800A-SPI	58
Figure 34 - Copying files from libMPSSE into BRT_AN_025	59
Figure 35 - Selecting Preprocessor Definitions	60
Figure 36 - Running the example	60

Appendix D – Revision History

Document Title: BRT_AN_025 EVE Portable MCU Library
Document Reference No.: BRT_000429
Clearance No.: BRT#211
Product Page: <http://brtchip.com/product/>
Document Feedback: [Send Feedback](#)

Revision	Changes	Date
1.0	Initial Release	30-04-2024