Application Note

BRT_AN_096

Working with EVE External NOR Flash Memory

Version 1.0

Issue Date: 10-04-2025

Guide to Designing External NOR Flash and Using Flash Utilities in Bridgetek's EVE Asset Builder.

Bridgetek Pte Ltd (BRTChip)
1 Tai Seng Avenue, Tower A, #03-05, Singapore 536464
Tel: +65 6547 4827
Web Site: http://www.brtchip.com

Table of Contents

1 Introduction

This document offers guidance on designing and operating with external Quad-SPI NOR flash memory for Bridgetek's BT81X series Embedded Video Engine (EVE) graphic controller ICs.

External NOR flash memory connected to EVE is used for storing of large graphics assets such as custom fonts, animations and videos. This setup allows EVE to fetch data directly, bypassing the host MCU, which significantly reduces the MCU's workload by eliminating the need for constant display content loading.

EVE graphic controller ICs that support external NOR flash memory include:

- BT815Q
- BT816Q
- BT817AQ
- BT817Q
- BT818Q

The selected external flash memory must be SPI NOR flash with XIP operation support. The BT81X series supports NOR flash memory from major memory manufacturers such as Cypress, Winbond, Micron, Macronix, ISSI, and GigaDevice. A list of supported flash memory devices is provided under Appendix A.

2 Working with EVE via EAB using Programmer Modules

The BT81X series supports NOR flash with power rails ranging from 1.8V to 3.3V, with a maximum capacity of 256MByte.

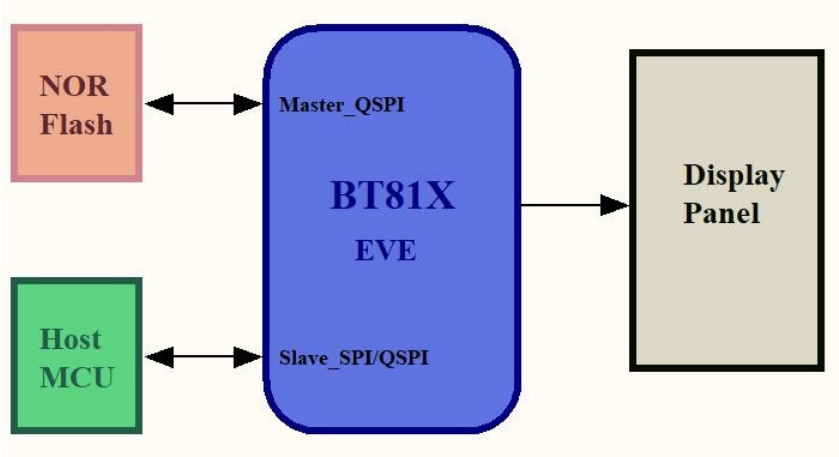


Figure 1 - System Design

The block diagram in Figure 1 illustrates the connections between the host MCU, EVE graphic controller, external flash memory, and output display device. Host microcontrollers and microprocessors connect to EVE SPI/QSPI slave interface for control and communication, while external SPI/ QSPI NOR flash memory connects to EVE QSPI master interface for extended memory access. EVE can display pre-loaded content directly from NOR flash to LCD, bypassing the host MCU.

2.1 Selecting and Configuring Programmer Modules

2.1.1 Choosing a Programmer Module

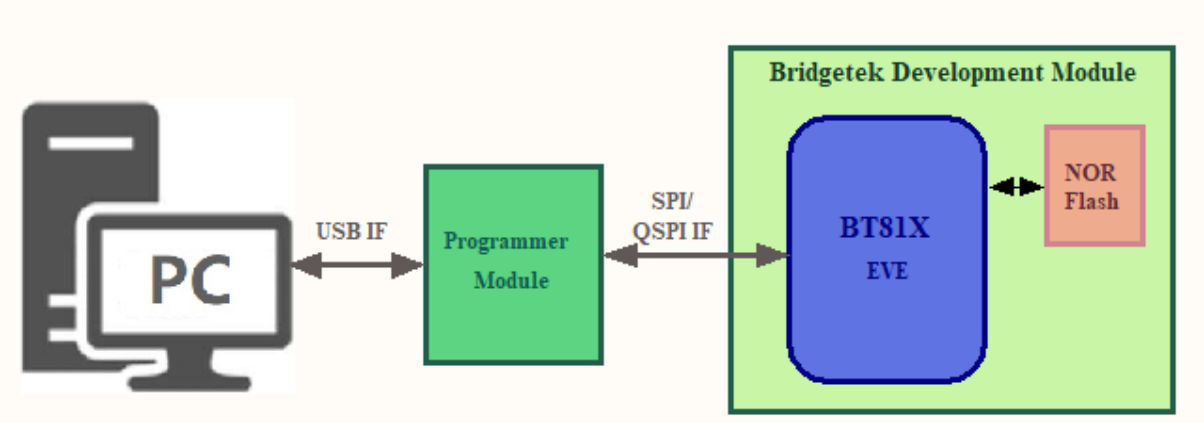


Figure 2 - EVE Communication via Programmer Module

Figure 2 shows how the programmer module facilitates communication between Bridgetek's BT81X EVE development module and a PC using Bridgetek's *Eve Asset Builder* (EAB). EAB includes tools to generate resources for EVE series devices, streamlining EVE-based application development. The *Flash Utilities* tool in EAB allows users to easily generate and download BLOB (see Binary Firmware Driver (BLOB)) and data images into the flash memory.

EAB supports three types of "Programmer Module" that serve as the host MCU for communicating with EVE:

- FT4222
- MPSSE
- Raspberry Pi Pico

2.1.1.1 FT4222

The FT4222 programmer module uses the FT4222H bridge IC from FTDI, a USB2.0 to Quad-SPI/ I2C interface device controller. It supports single, dual and quad data width transfer modes with clock speed up to 30MHz and transfer rate of 53.8Mbps in quad mode. FTDI's royalty-free Direct (D2XX) drivers for Windows eliminate the requirement for USB driver development in most cases.

Designers can either integrate the FT4222 IC into their design or choose a pre-assembled module like the UMFT4222EV-D, available from FTDI online sales and distributors such as Digikey, Mouser, and Element14.

The FT4222 is preferred over MPSSE due to its faster transfer rate, made possible by the quad SPI interface.

2.1.1.2 MPSSE

The MPSSE (Multi-Protocol Synchronous Serial Engine) programmer module uses the FT232H bridge IC from FTDI, a USB 2.0 to JTAG, I2C, SPI (MASTER) or bit-bang interface Device Controller. FTDI's royalty-free Virtual Com Port (VCP) and Direct (D2XX) drivers eliminate the requirement for USB driver development in most cases.

Designers can integrate the MPSSE IC into their design or purchase a ready-made module, such as:

- C232HM-DDHSL-0
- C232HM-EDHSL-0
- UM232H

These are available from FTDI online sales and distributors like Digikey, Mouser and Element14.

2.1.1.3 Raspberry Pi Pico

The Raspberry Pi Pico programmer module utilizes the RP2040 microcontroller chip, which can be programmed in C and MicroPython. The RP2040 features a dual-core Arm Cortex-M0+ processor, 264kB of internal RAM, and supports up to 16MB of off-chip flash. It also offers a variety of flexible I/O options, including I2C, SPI, and Programmable I/O (PIO).

Designers can either integrate RP2040 MCU into their design or use a ready-made evaluation board, such as:

- MM2040EV PICO Adapter Board
- RP2040 Raspberry Pi PICO SC0915 Embedded Evaluation Board

These boards are available from Bridgetek online sales (MM2040EV) and distributors like Digikey, Mouser, and Element14.

2.1.2 Configuring Programmer Modules

The section provides guidelines for hardware connection and configuration of programmer modules and EVE development boards to enable communication with EVE.

There are 3 types of headers used by Bridgetek EVE development boards for power and communication interface to external programmer modules.

- 1*10 header: SPI, power and control interface
- 1*10 header + 1*2 header: QSPI, power and control interface
- 2*8 header: QSPI, power and control interface

These are shown in Figure 3 below:

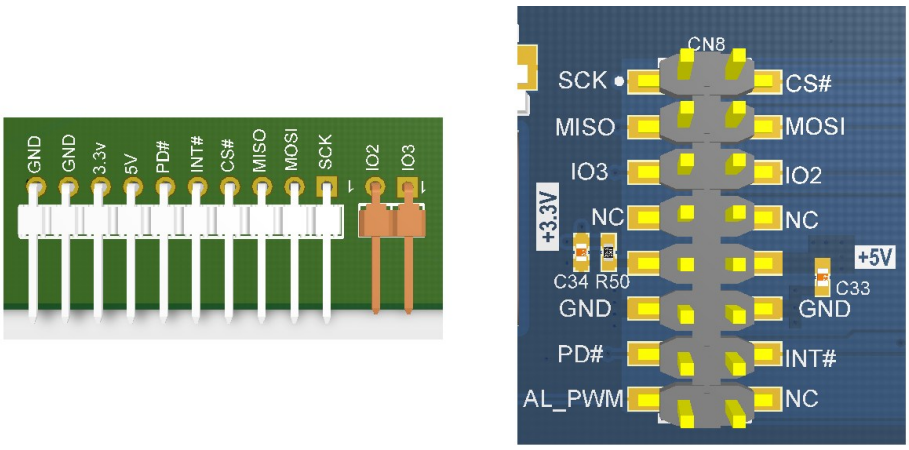


Figure 3 - Bridgetek Interface Headers

The following examples illustrate the setup and configuration of the programming modules using the ME817EV (BT817) development board, which features the 2x8 header.

2.1.2.1 FT4222: UMFT4222EV-D

This section outlines the hardware setup for UMFT4222EV bridge IC and BT817 EVE IC. Figure 4 illustrates the UMFT4222EV programmer module alongside the ME817 development board used in this setup. It is worth noting that the ME817 board includes an onboard FT4222 bridge IC. This example serves to guide developers in understanding how the Quad SPI interface connects to the EVE IC.

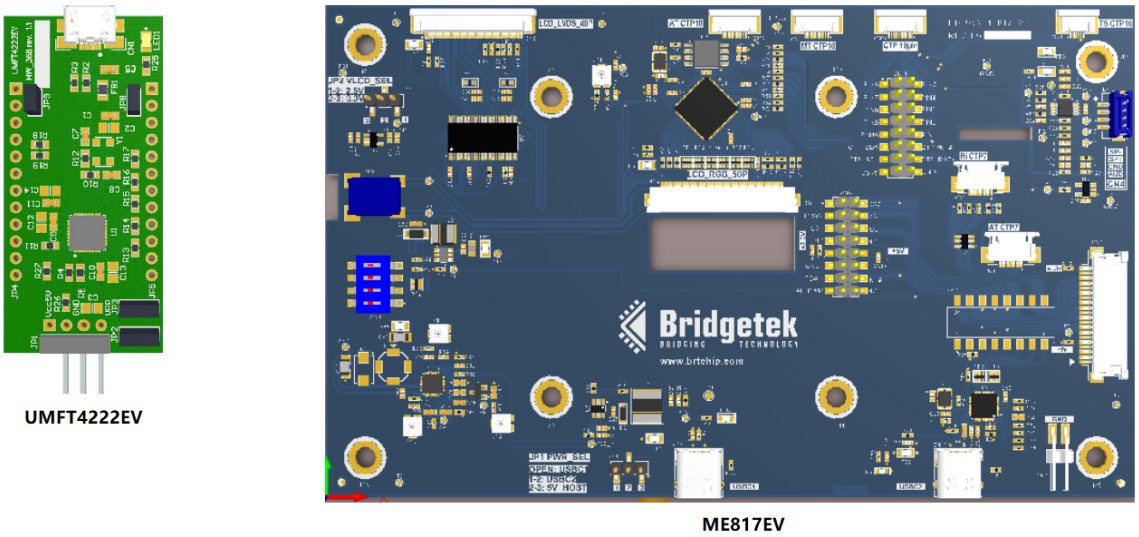


Figure 4 - UMFT4222EV-D Programmer Module/Me817EV BT817 Development Board

To establish communication with EVE, connect the QSPI channel of the UMFT4222EV programmer module to EVE's SPI slave I/O interface. Refer to Figure 5 for pin numbers and names. Use jumper wires to connect and short the QSPI pins as indicated in Table 1 below:

UMFT4222EV-D		ME817EV		Description
Pin No. (Connector)	**Name**	**Pin No. (CN8)**	**Name**	
1 (JP4)	VBUS	10	+5V	+5V Power supply.
9 (JP4)	SS0O	2	CS#	SPI chip select.
4 (JP5)	IO3	5	IO3	SPI data line 3.
5 (JP5)	IO2	6	IO2	SPI data line 2.
6 (JP5)	GND	12	GND	Ground.
7 (JP5)	MOSI	4	MOSI	SPI data line 0.
8 (JP5)	MISO	3	MISO	SPI data line 1.
9 (JP5)	SCK	1	SCK	SPI clock.

Table 1 - QSPI connection for UMFT4222EV and ME817EV

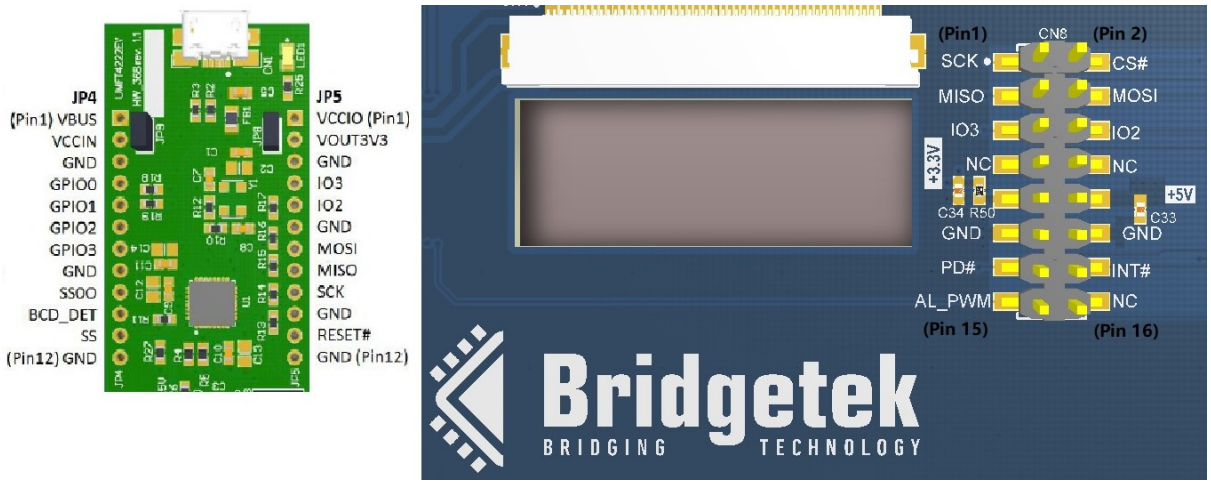


Figure 5 - Connectors Pin Number/Name Assignment

Ensure that the power supply and function jumper settings for the programming modules and EVE development boards are configured as specified in Table 2 below to ensure proper board functionality.

Connector	State	Description
UMFT4222		
JP2	Short 2-3	Chip configuration mode setting, set DCNF0 to GND.
JP3	Short 2-3	Chip configuration mode setting, set DCNF1 to GND.
JP8	Short	VBUS supply to system.
JP9	Short	3V3 supply to IC VCCIO.
ME817		
JP1	Short 2-3	Host (programmer module) supply 5V to system.
SW2	OFF	Switch EVE QSPI interface control to external module via CN8.

Table 2 - Supply and Function Jumper Selections

Connect a USB micro cable from CN1 on UMFT4222EV to PC for communication and control of EVE.

2.1.2.2 MPSSE: C232HM-EDHSL-0

This section provides guidelines for the hardware setup using MPSSE bridge IC and BT817 EVE IC. Figure 6 shows the C232HM-EDHSL MPSSE cable and ME817EV development module.

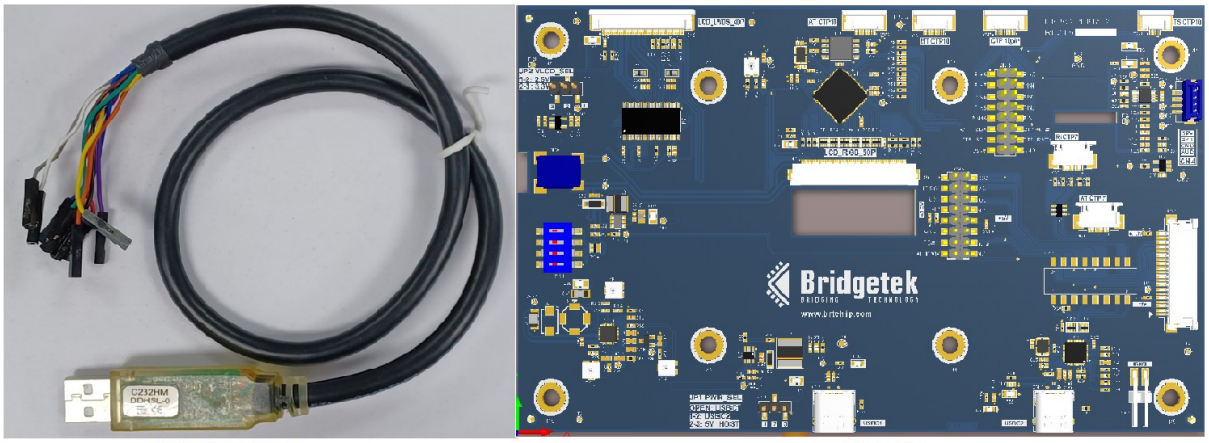


Figure 6 - C232HM-EDHSL-0 Programmer Module/ME817EV BT817 Development Board

To communicate with EVE, connect the SPI channel of C232HM-EDHSL MPSSE cable to EVE's SPI slave I/O interface. Refer to Table 3 for wire connection.

C232HM-DDHSL	ME817EV		Description
Wire Colour	**Pin No. (CN8)**	**Name**	
Orange	1	SCK	SPI clock.
Brown	2	CS#	SPI chip select.
Green	3	MISO	SPI master input slave output.
Yellow	4	MOSI	SPI master output slave input.
Red	10	+5V	+5V Power supply.
Black	12	GND	Ground.

Table 3 - SPI Connection for C232HM and ME817EV

Unlike C232HM-EDHSL, C232HM-DDHSL MPSSE cable only provides 3.3V supply, which is insufficient to power ME817. To address this, the ME817 must tap into external USB power source.

Ensure that the power supply and function jumper settings on ME817 development board are configured as specified in Table 4 below to ensure proper board functionality.

Connector	State	Description
ME817 with C232HM-EDHSL		
SW2	All Off	Switch EVE QSPI interface control to external module via CN8.
JP1	Short 2-3	Host (programmer module) supply 5V to system.
ME817 with C232HM-DDHSL		
SW2	All Off	Switch EVE QSPI interface control to external module via CN8.
JP1 (option 1)	Open	ME817 tap VBUS supply from PC via USB1 connector.
JP1 (option 2)	Short 1-2	ME817 tap VBUS supply from PC via USB2 connector.

Table 4 - Supply and Function Jumper Selections

2.1.2.3 Raspberry Pi Pico: MM2040EV

This section provides guidelines for the hardware setup using Raspberry Pi Pico MCU and BT817 EVE IC. Figure 7 shows the MM2040EV evaluation module and ME817EV development board.

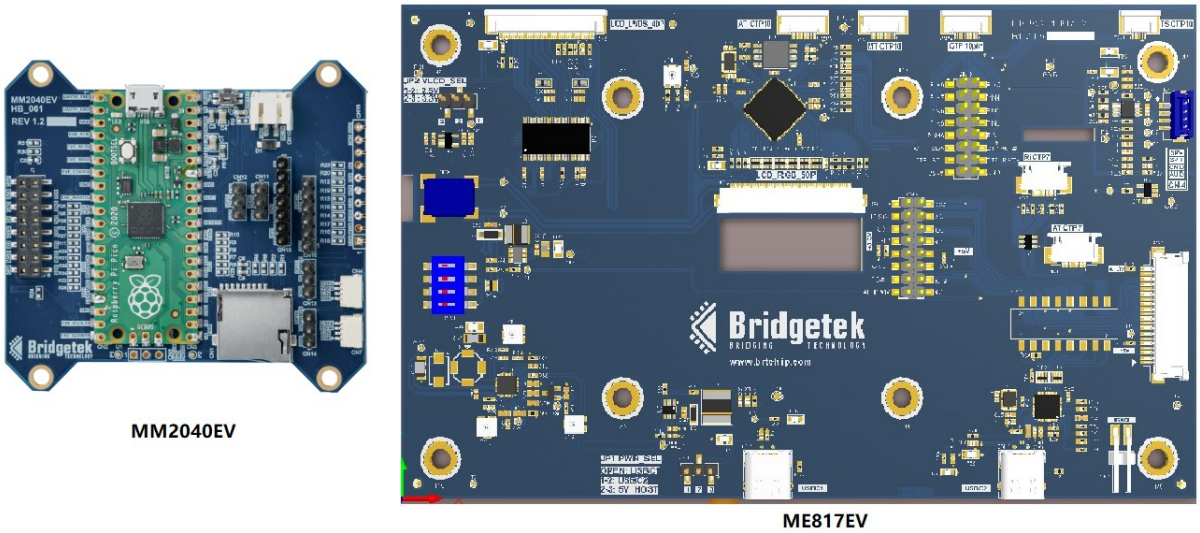


Figure 7 - MM2040EV Evaluation Module/ME817EV BT817 Development Board

To communicate with EVE, connect the SPI channel of the MM2040EV programmer module to the SPI slave I/O interface on the EVE chip. Refer to Figure 8 below for guidance on inserting the programmer module onto the EVE development module with the correct orientation.

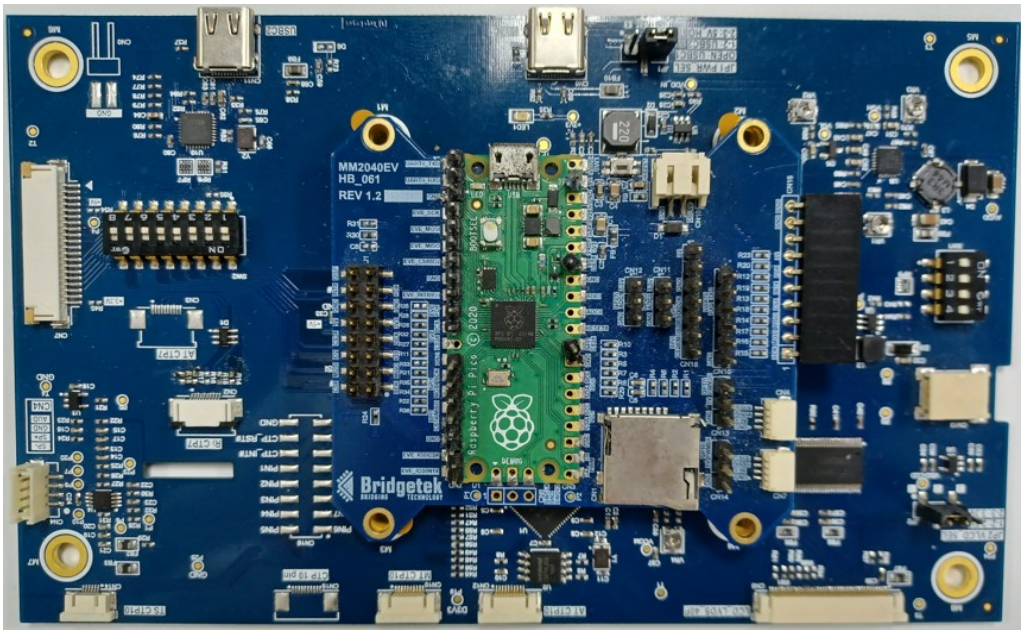


Figure 8 - Connecting MM2040 and EVE Development Board

Short pins 2 and 3 of JP1 on the ME817EV BT817 development board to select power supply from SPI interface connector when using the MM2040EV programmer module.

Make sure selection pins of SW2 on ME817EV development board are set to OFF to configure EVE QSPI interface control through external module via connector CN8.

2.1.2.4 Raspberry Pi Pico: RP2040 MCU Module

This section details the hardware setup for RP2040 MCU and BT817 EVE IC. Figure 9 shows the Raspberry Pi Pico RP2040 MCU module and ME817EV development board.

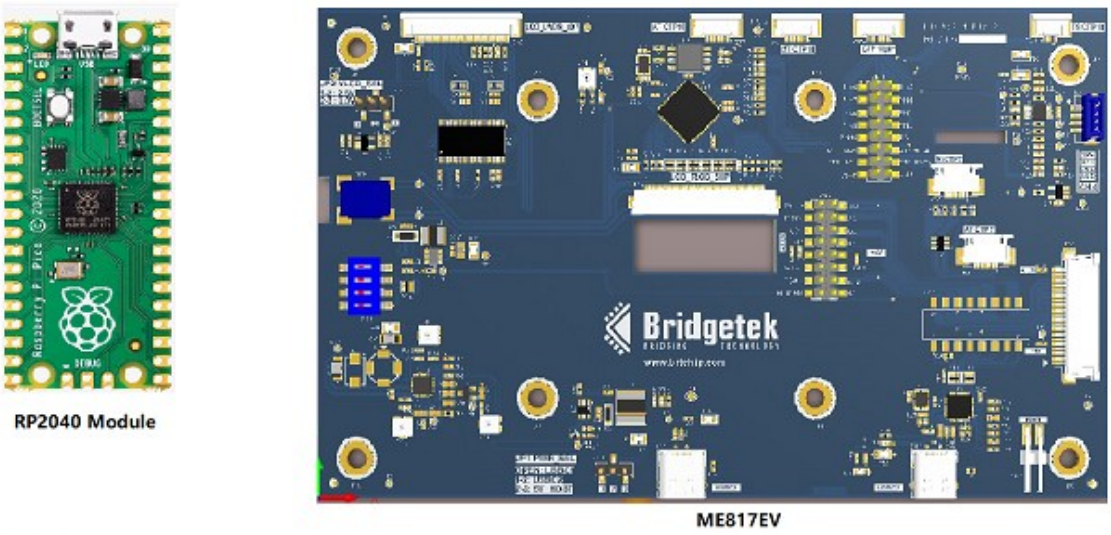


Figure 9 - RP2040 Module/ME817EV BT817 Development Board

To communicate with EVE, connect the SPI channel of RP2040 module to EVE's SPI slave I/O interface. Refer to Table 5 and Figure 10 for pin connections.

RP2040	ME817EV	Description
VBUS	+5V	+5V Power supply.
GP2	SCK	SPI clock.
GP3	MOSI	SPI data line 0.
GP4	MISO	SPI data line 1.
GP5	CS#	SPI chip select.
GND	GND	Ground.

Table 5 - SPI Connection for RP2040 Module and ME817EV

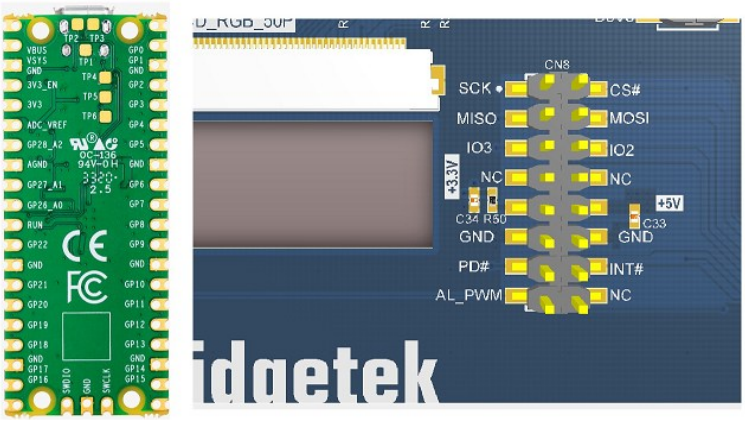


Figure 10 - Connectors Pin Name

Short pins 2 and 3 of JP1 on the ME817EV BT817 development board to select power supply from SPI interface connector when using the RP2040 module.

Make sure selection pins of SW2 on ME817EV development board are set to OFF to configure EVE QSPI interface control through external module via connector CN8.

2.2 Choosing EVE and Development Module

Bridgetek offers several development modules for testing EVE ICs prior to implementation. These modules can be purchased from Bridgetek's website.

2.2.1 Choosing EVE Graphic IC Controller

Bridgetek has developed four generations of EVE chips, but only the third (EVE3) and fourth (EVE4) generations support external NOR flash. Table 6 provides an overview of the available EVE versions.

	Part Number	Resolution	RGB Width (bit)	RAM	Flash Support	Touch Type	Packages (QFN)
EVE							
1.	FT800	480 x 320	18	256KB	No	Resistive	48 Pin
2	FT801	480 x 320	18	256KB	No	Capacitive	48 Pin
EVE2							
3.	FT810Q	800 x 600	18	1M	No	Resistive	48 Pin
4.	FT811Q	800 x 600	18	1M	No	Capacitive	48 Pin
5.	FT812Q	800 x 600	24	1M	No	Resistive	56 Pin
6.	FT813Q	800 x 600	24	1M	No	Capacitive	56 Pin
EVE3							
7.	BT815Q	800 x 600	24	1M	Yes	Capacitive	64 Pin
8.	BT816Q	800 x 600	24	1M	Yes	Resistive	64 Pin
EVE4							
9.	BT817AQ	1280 x 800	24	1M	Yes	Capacitive	64 Pin
10.	BT817Q	1280 x 800	24	1M	Yes	Capacitive	64 Pin
11.	BT818Q	1280 x 800	24	1M	Yes	Resistive	64 Pin
New Products							
12.	BT880Q	480 x 272	18	256KB	No	Resistive	48 Pin
13.	BT881Q	480 x 272	18	256KB	No	Capacitive	48 Pin
14.	BT882Q	480 x 272	24	256KB	No	Resistive	56 Pin
15.	BT883Q	480 x 272	24	256KB	No	Capacitive	56 Pin

Table 6 - Overview of EVE Graphic IC Controllers

2.2.2 Choosing Development Module for EVE Graphic IC Controller

Table 7 lists development modules for EVE3 and EVE4 with external NOR flash support.

	I/O Interface	Bridge IC/ MCU	LCD (Size, Resolution)	Touch Panel
EVE3: BT816Q				
VM816CU50A-D	USB	FT4222HQ	5”, 800 * 480	Resistive
VM816C50A-D	QSPI/SPI	-	5”, 800 * 480	Resistive
VM816CU50A-N	USB	FT4222HQ	-	-
VM816C50A-N	QSPI/SPI	-	-	-
EVE4: BT817Q				
IDM2040-7A	USB	Raspberry Pi 2040	7”, 800 * 480	Capacitive
ME817EV	QSPI/SPI	FT4222HQ	-	-

Table 7 - EVE3/4 Development Modules with External NOR Flash Support

Purchase links from Bridgetek Pte Ltd:

- VM816CU50A-N
- VM816C50A-N
- VM816CU50A-D
- VM816C50A-D
- IDM2040-7A
- ME817EV

Please note that these development modules are also available through distributors like Digikey, Mouser, and Element14.

2.3 Working with EVE Asset Builder

The most recent version of EVE Asset Builder (EAB) can be downloaded from the Bridgetek website using this link EAB. Version 2.12.0 or later includes updated BLOB images offering support for a wider range of flash devices and provides patches addressing the “Display List Overfetch Errata” (BRT_TN_005).

The following section provides a step-by-step guide to:

- Setting up EAB.
- Converting user asset data into a binary file for flash programming.
- Downloading binary file into flash.

2.3.1 Setting up EAB

2.3.1.1 Installing EAB

Download the installer file, "EVE-Asset-Builder-setup-2.12.0," and close all running applications before starting the installation. A default directory, **"C:\Users\Public\Public Documents\EVE Asset Builder"**, will be created to store data and generated files. You can modify the installation location if desired.

2.3.1.2 Running EAB

After connecting and configuring the hardware as illustrated in section 2.1 launch "EVE Asset Builder" from the Windows Start Menu. The EAB user interface will appear, as shown in Figure 11.

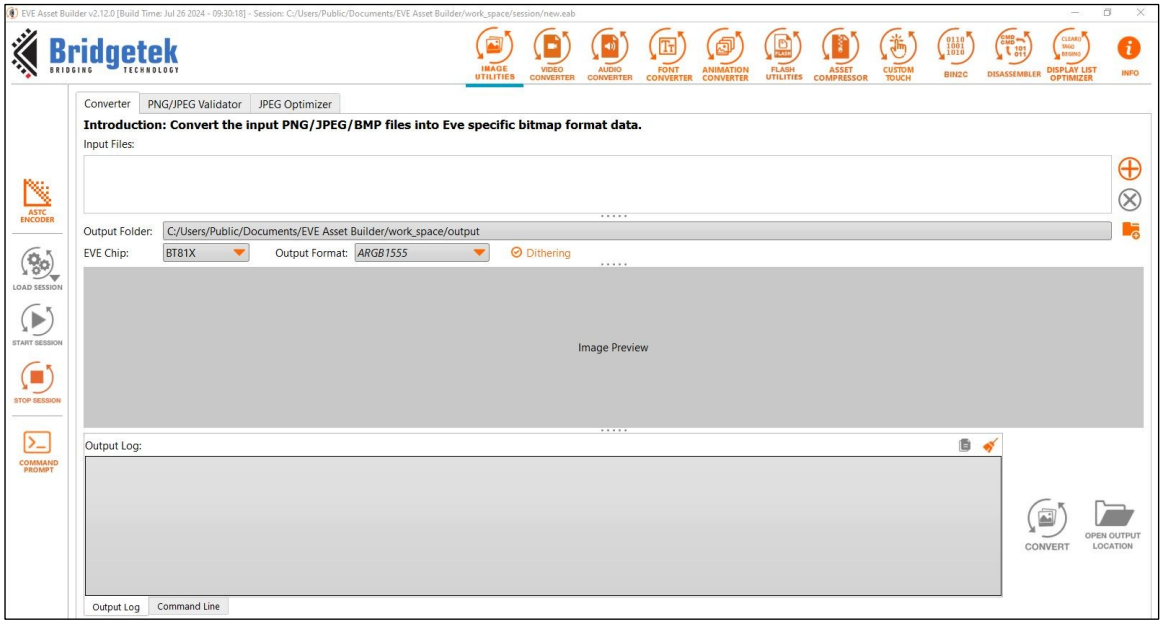


Figure 11 - EAB User Interface Version 2.12.0

2.3.1.3 Setting up EVE

To begin working with flash applications, click on the **FLASH UTILITIES** icon as shown in Figure 12.

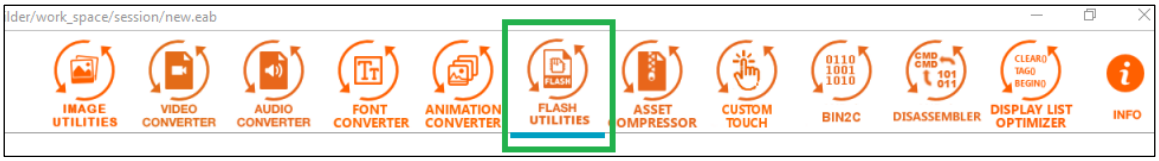


Figure 12 - "FLASH UTILITIES" icon select

Select the EVE chip that corresponds to your setup.

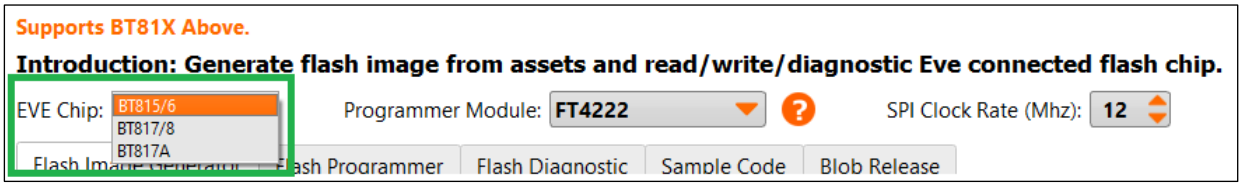
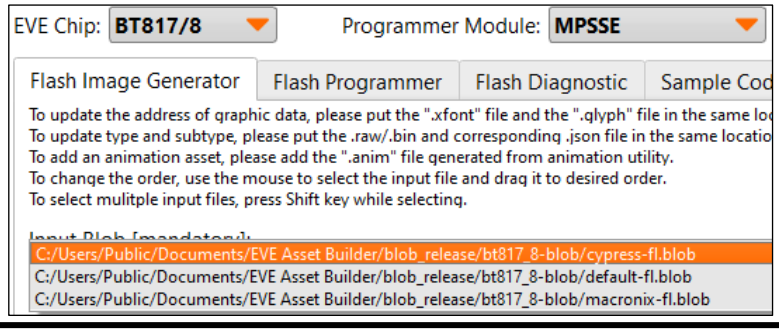


Figure 13 - EVE select

This will load the corresponding BLOB driver (BT815_6-blob or BT817_8-blob). Diagram below shows 3 types of blob drivers being loaded for class BT817/8 EVE graphic controller IC.



Refer to Appendix A to select the correct driver for your NOR flash, ensuring optimal programming speed.

2.3.1.4 Setting up Programmer Module

Select **Programmer Module** that corresponds to your set up as shown in Figure 14.

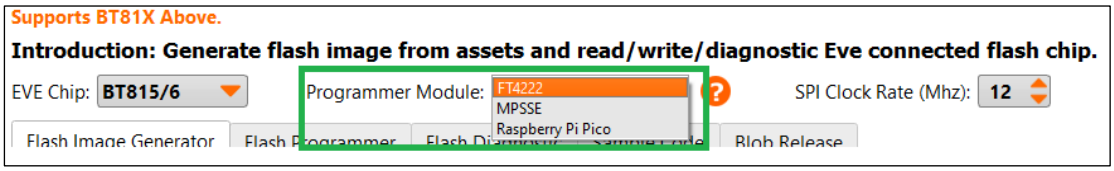


Figure 14 - Programmer Module Selection

Set the 'SPI Clock Rate' (up to 30 MHz) for communication between the programmer module and the EVE chip.

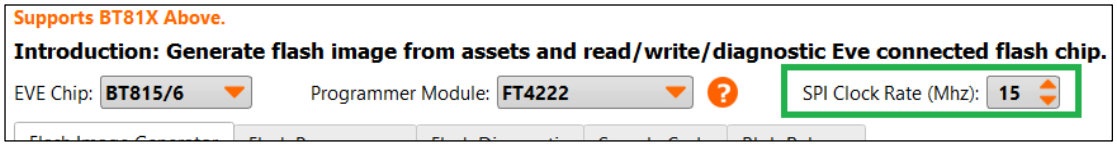


Figure 15 - SPI Clock Rate Selection

2.3.2 Generating Binary Files for Flash

Note that users must convert data files to the appropriate resolution (matching the target display unit) and EVE-specific file format before downloading them into flash memory. This can be accomplished by using data converter tools available in EAB, as shown in Figure 16.

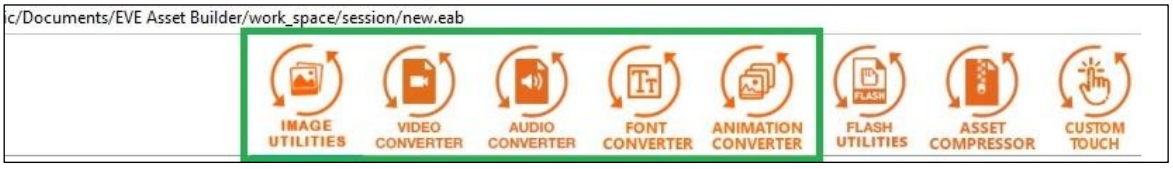


Figure 16 - Data Converter Tools

To generate a flash image binary file that contains both blob driver and display data, start by selecting **Flash Image generator**.

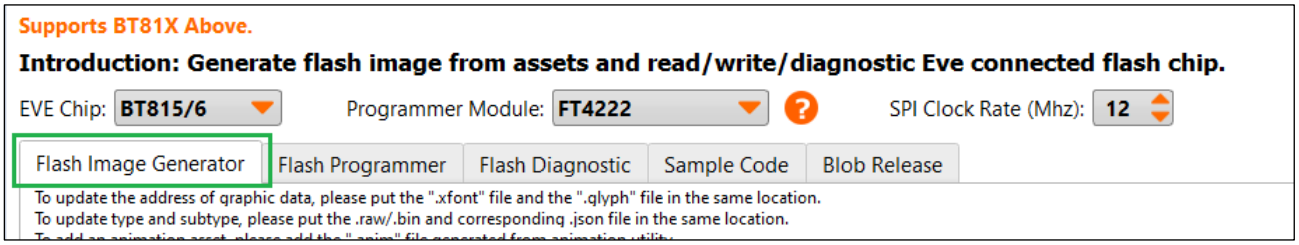


Figure 17 - "Flash Image Generator" Page Selection

Next, click add file icon.

Figure 18 - Insert File Icon

In the file destination navigation window, select the folder that contains the user's converted data.

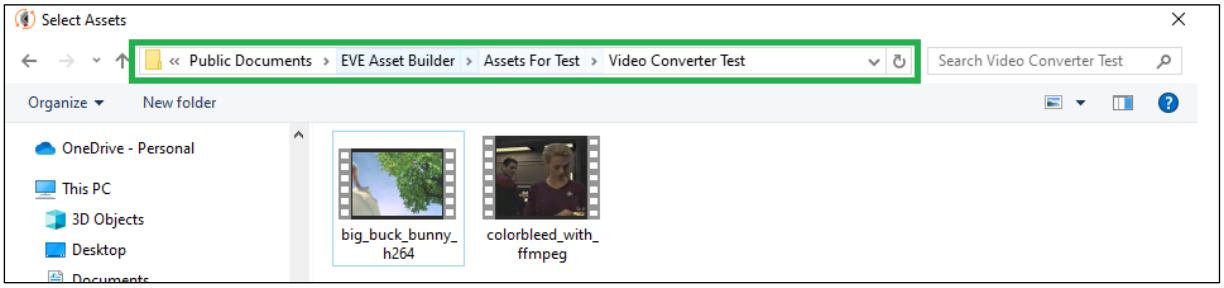


Figure 19 - Storage Folder Selection

Next, choose the target file or files to be downloaded to the flash.

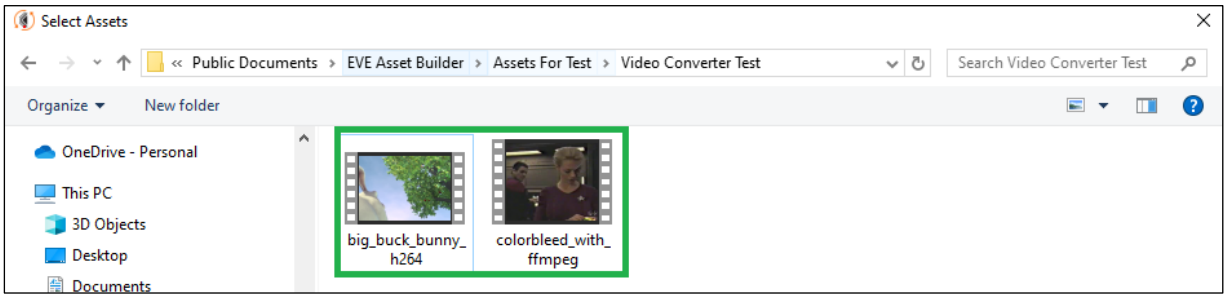


Figure 20 - Upload Target Files

Selected files will be displayed in the 'Input Files' selection window.

Figure 21 - Files Attached

Repeat the process to add all desired asset data.

Click and select "Insert EDF Block" as shown in diagram below. An EDF, or "EVE Flash Description File," containing the asset data (input files) in sequential order will be generated and inserted after the blob driver.

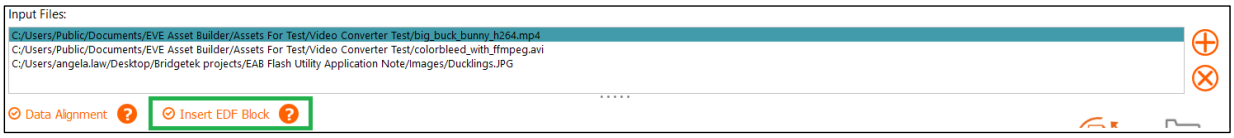


Figure 22 - Enable "Insert EDF Block"

Generated files will be stored under default location "C:\Users\Public\Documents\EVE Asset Builder\work_space\output". User can change output location by clicking "folder" icon.

Figure 23 - Output Folder Location

Next, enter desired file name in the filename entry field and click on **GENERATE** icon.

Figure 24 - "Generate" Icon Selection

Image generation status and generated files information will be shown in "output log" dialog box window as shown in diagram below.

Figure 25 - Output Log

2.3.2.1 Generated Files

The following files as shown in Table 8 are generated after completing the process:

	Format	Extension	Description
1	Binary	.bin	Binary file that will be programmed into flash memory.
2	MAP	.map	File that contains address location and memory size of asset data.
3	EDIF	.edf	File that contains detail information about asset data.
4	Source	.json	File that that correspond to edf file.

Table 8 - Generated Files

Figure 26 below displays information of EDF file, illustrating the content of generated binary file that will be programmed into flash memory.

```
Name                          : ID : Offset   : Size     : Compression : Type : Sub-Type : Width : Height

default-fl.blob               : 0  : 0        : 4096     : 0           : 1    : 0        : 0     : 0
EDF Block                     : 1  : 4096     : 256      : 0           : 253  : 0        : 0     : 0
big_buck_bunny_h264.mp4       : 2  : 4352     : 10852630 : 0           : 254  : 0        : 0     : 0
big_buck_bunny_h264.mp4.padding : 3  : 10856982 : 42       : 0           : 252  : 0        : 0     : 0
colorbleed_with_ffmpeg.avi    : 4  : 10857024 : 1283044  : 0           : 12   : 0        : 0     : 0
colorbleed_with_ffmpeg.avi.padding : 5  : 12140068 : 28       : 0           : 252  : 0        : 0     : 0
Ducklings.JPG                 : 6  : 12140096 : 24455    : 0           : 254  : 0        : 0     : 0
Ducklings.JPG.padding         : 7  : 12164551 : 57       : 0           : 252  : 0        : 0     : 0
red-panda.bmp                 : 8  : 12164608 : 243066   : 0           : 254  : 0        : 0     : 0
red-panda.bmp.padding         : 9  : 12407674 : 6        : 0           : 252  : 0        : 0     : 0
```

Figure 26 - EDF File Content

Table 9 provides a brief overview of EDF content.

Name	Description
ID	Denotes the order of an asset in flash image
Offset	Starting address of the memory location for asset data.
Size	Size of the memory allocated for asset data.
Width	Only for bitmap, video, and animation. Zero for all other data format.
Height	Only for bitmap, video, and animation. Zero for all other data format.
Type	Refer to Appendix B for detail description.
Sub-Type	Refer to Appendix B for detail description.

Table 9 - EDF Content Description

2.3.3 Downloading and Reading Files from Flash

To download generated binary file into flash memory, click on **Flash Programmer**.

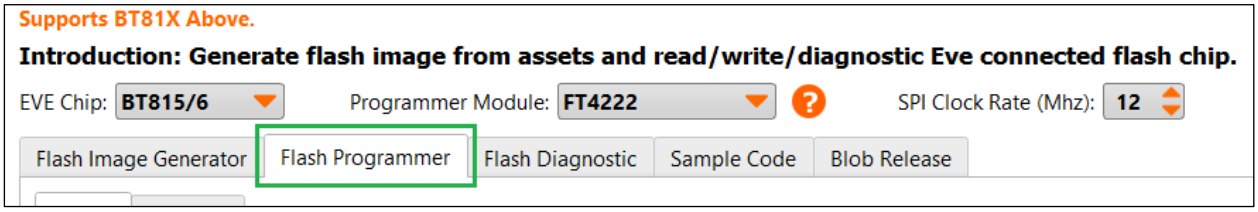


Figure 27 - "Flash Programmer" Page Selection

2.3.3.1 Detecting Flash

Click **Detect** to verify the presence of the flash connected to the EVE.

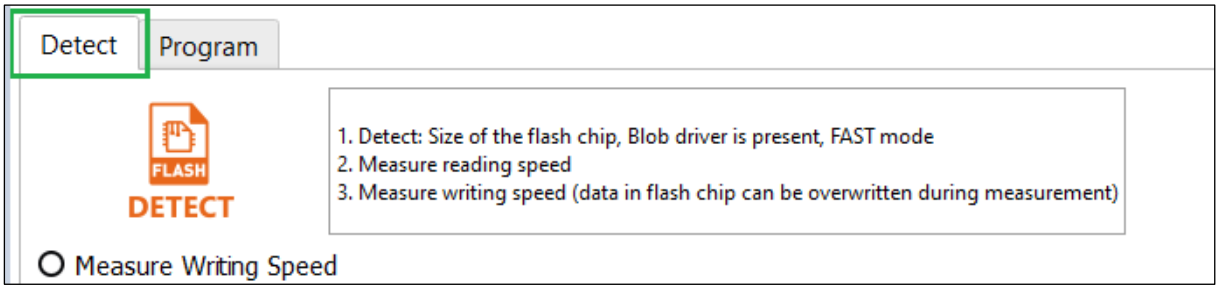


Figure 28 - "Detect" Page Selection

Please be aware that content in flash can be overwritten while measuring writing speed. Avoid selecting "Measure Writing Speed" if users are concerned about data being altered.

To start flash detection, click on **DETECT** icon.

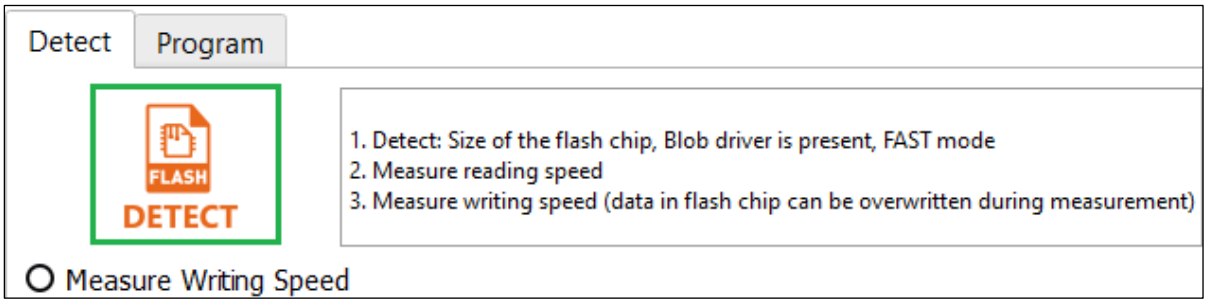
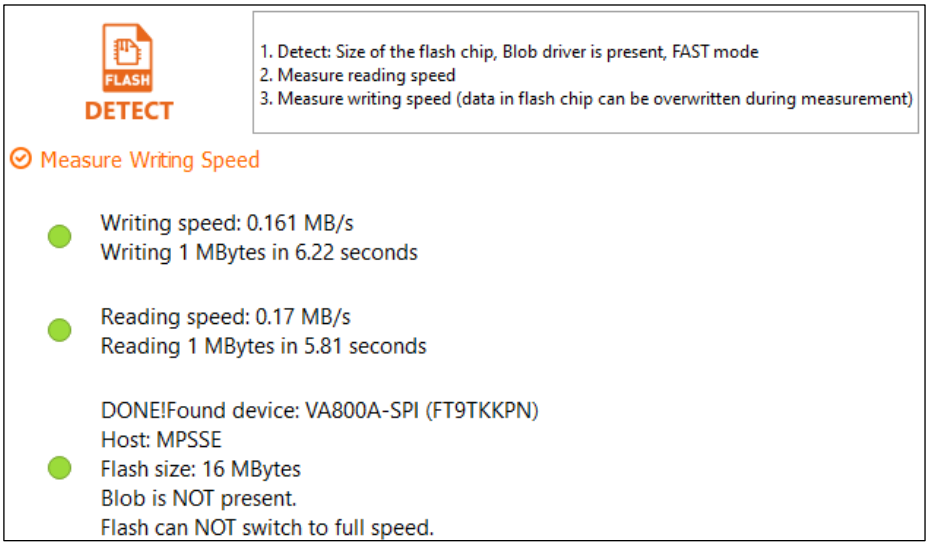
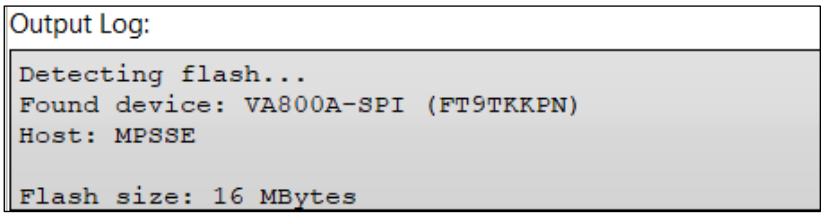


Figure 29 - "Detect" Icon Selection

EVE will initiate communication with the flash and display the data transfer rate. Below is an example of flash detection and the transmission speed. It indicates the presence of a host interface (MPSSE) used for communication with EVE and shows the size of the external flash memory connected to the IC.

If an error occurs or the flash is not detected, users can check the 'Output Log' window for error messages to assist in troubleshooting.

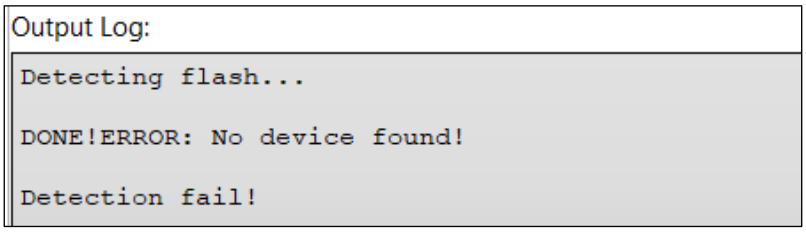




2.3.3.2 Troubleshooting Flash Detection Failure

Example 1:

The error message displayed in the diagram below signifies a communication failure between EAB (PC) and EVE.



Users can run "Device Manager" to verify that the programmer modules connected to the PC are detected and that the drivers are properly installed. To do this, check under "Universal Serial Bus controllers" for "USB Serial Converter" or "USB Serial Device" and ensure that the programmer modules are recognized, with no exclamation marks indicating errors.

Figure 30 to Figure 32 below show examples of proper detection of devices for all 3 types of host-MCU.

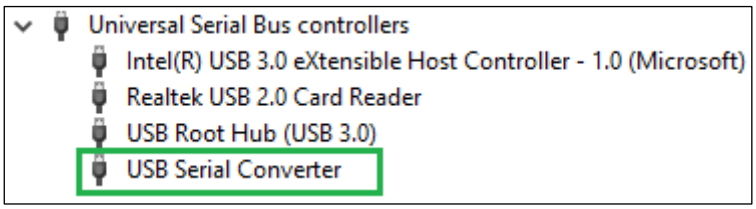


Figure 30 - Proper Detection of MPSSE Bridge IC

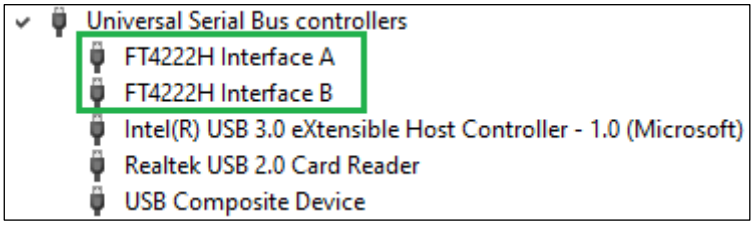


Figure 31 - Proper Detection of FT4222 Bridge IC

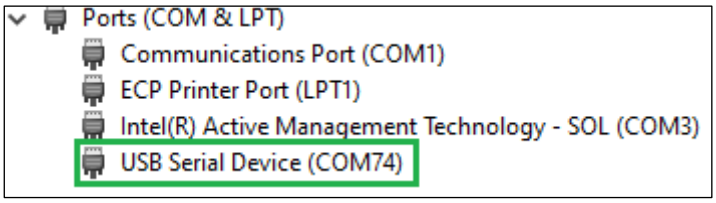
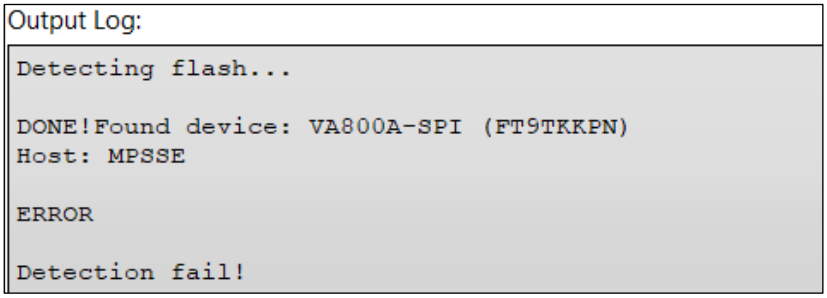


Figure 32 - Proper Detection of Raspberry PI Pico MCU

Example 2:

The error message shown below indicates a failure in communication between EVE and external flash memory.



Ensure that the external flash memory is properly connected to the EVE chip, paying particular attention to the following signals: "CS_N," "SCK," "MISO," and "MOSI." Also, confirm that the correct power supply is used and that the recommended pull-up resistors are in place.

2.3.3.3 Programming Flash

Once proper communication is established and flash is detected, we can proceed to download binary file into the flash memory.

Select the "Program" toolbar as shown in Figure 33.

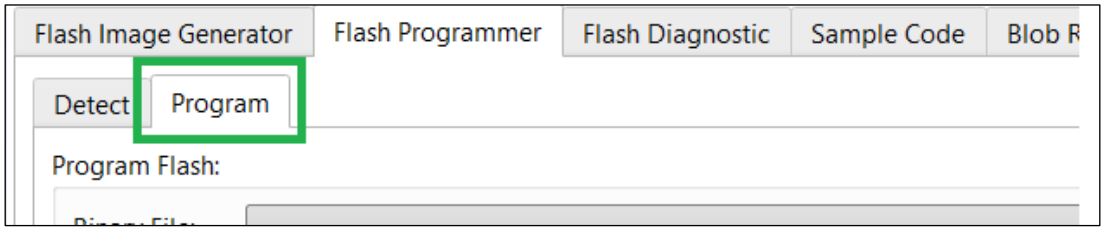


Figure 33 - "Program" Page Selection

Click on add file icon to choose the bin file to be programmed.

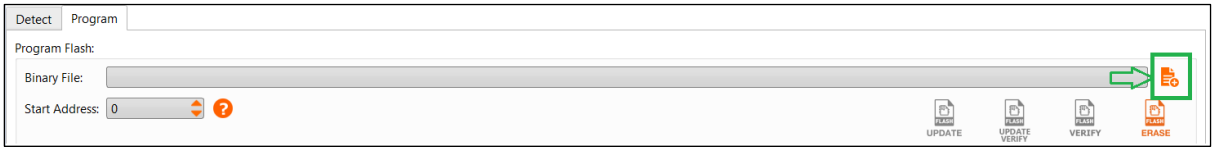


Figure 34 - "Add file" Icon Selection

In the file destination navigation window, select the folder containing the generated binary file to be downloaded to the flash. Double-click to choose the .bin file.

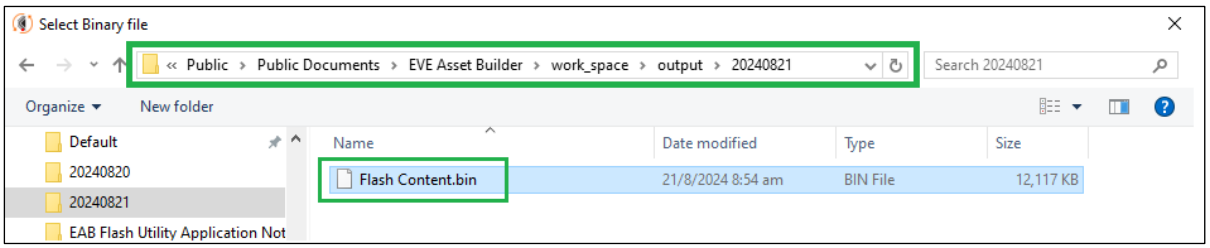


Figure 35 - Binary File Selection

Once a file is selected, the programming tools **PROGRAM**, **UPDATE**, **UPDATE VERIFY**, and **VERIFY** as illustrated in the diagram below, will become available for programming the flash.

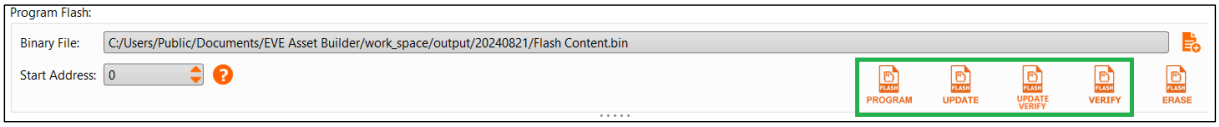
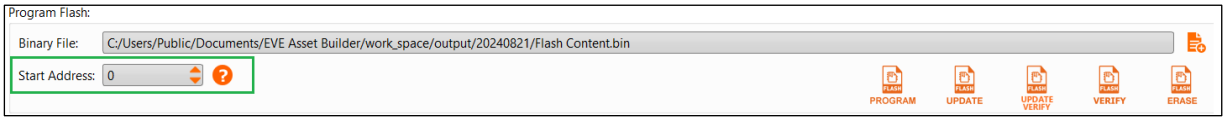


Figure 36 - Programming Tools Enabled

Leave **Start Address** for the binary file set to '0' when downloading. This option is reserved for advanced users who wish to overwrite only a specific part of the flash content and are aware of the exact address location where the data needs to be stored.



2.3.3.3.1 Understanding and operating Programming Modes

Users can choose between two programming modes: **Program** and **Update**. This section provides a brief overview of these modes and explains how to download binary files using each. Select the mode that best suits your needs.

2.3.3.3.1.1 PROGRAM Mode:

This mode is ideal for use with empty flash memory or when significant changes to the flash content are expected. It utilizes the FLASH_PROGRAM command, which offers the fastest data transfer rate compared to other flash write commands like FLASH_WRITE and FLASH_UPDATE.

Please note that programming the flash in this mode requires the flash to be empty, making it necessary to erase the flash content first.

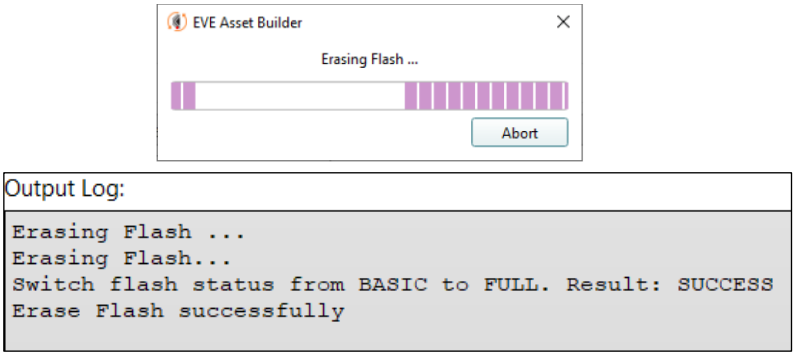
Follow these steps to flash memory using the "PROGRAM" mode.

Step 1: Erase

If you are using fresh memory from the factory, this step can be skipped. To erase the flash content, select the **ERASE** icon, as depicted in the diagram below.

Figure 37 - "ERASE" Icon Selection

A status indicator window and progress updates within the output log window, as shown in the diagram below, will display status of the flash during the erasing process.

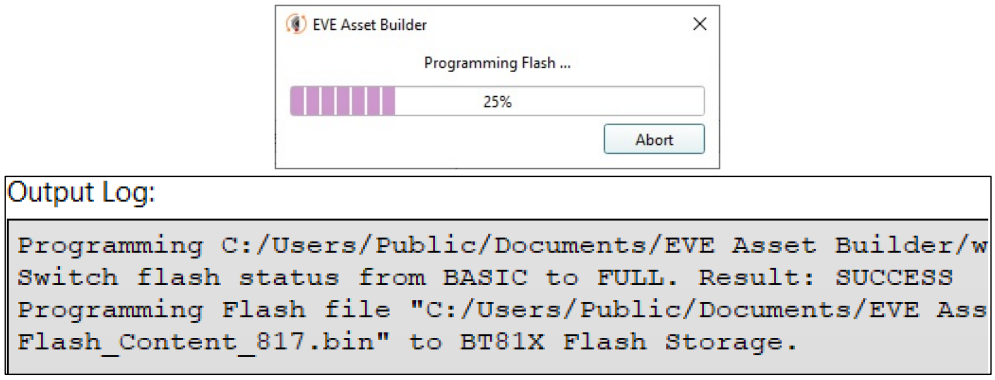


Step 2: Program

User can begin programming the flash when it is empty. To start downloading the binary file into the flash, select the **PROGRAM** icon, as shown in Figure 38 below.

Figure 38 - "PROGRAM" Icon Selection

A status indicator window and progress updates within the output log window, as illustrated in the diagram below, will show status of the flash during the programming process.



Step 3: Verify

Users can choose to verify the flash content after the file has been downloaded. To do this, select the **VERIFY** icon, as shown in the diagram below.

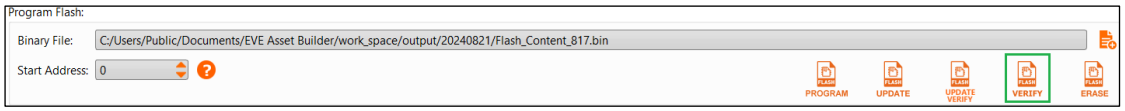
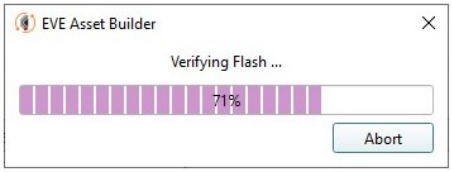


Figure 39 - "VERIFY" Icon Selection

A status indicator window and progress updates within the output log window, as shown in the diagram below, will display status of the flash during the verification process.



Output Log:

```
Verify Flash ...
WARNING: Blob driver will be overwriten!
Switch flash status from BASIC to FULL. Result: SUCCESS
Verifying Flash...
Verifying SUCCESS!
```

2.3.3.3.1.2 ***UPDATE Mode****:*

This mode is ideal for programmed flash when only minor changes are expected. In "UPDATE" mode, EVE checks for discrepancies between the file and the existing flash content, erasing and writing data only when differences are detected. It uses the FLASH_UPDATE command to write data into flash memory.
Follow these steps to flash memory using the "UPDATE" mode.

Step 1: Update

Programming the flash using **Update** mode do not require the flash to be empty, so it is not necessary to erase the flash content. Select the **UPDATE** icon, as depicted in the diagram below to start downloading flash content.

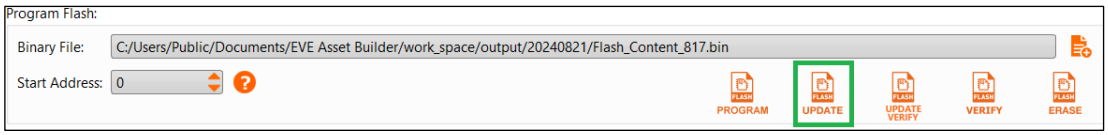
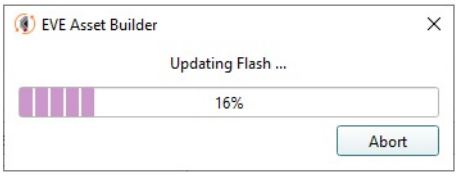
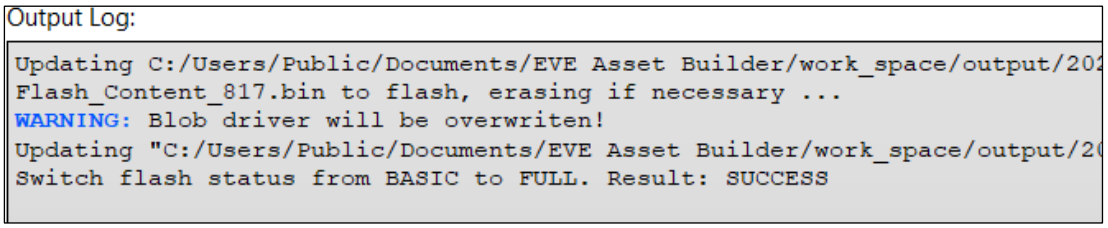


Figure 40 - "UPDATE" icon select

A status indicator window and progress updates within the output log window, will display status of the flash during the flash update process.





Step 2: Verify

Users can choose to verify the flash content after the file has been downloaded. To do this, select the **VERIFY** icon.

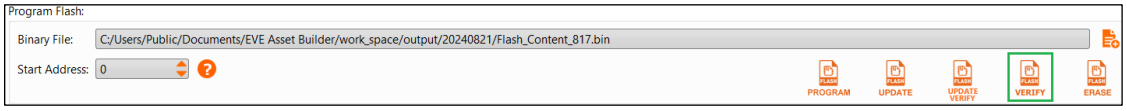
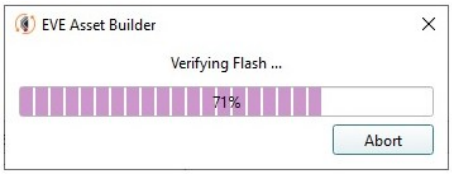
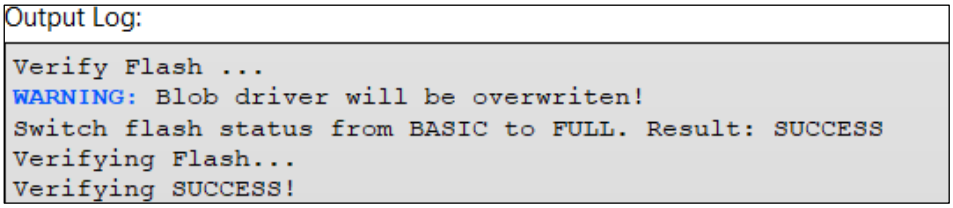


Figure 41 - "VERIFY" icon select

A status indicator window and progress updates within the output log window, will display status of the flash during the verification process.





Please note that users can complete both tasks by selecting the **UPDATE VERIFY** icon.

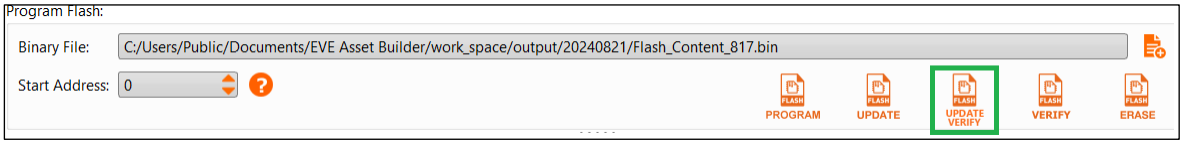


Figure 42 - "UPDATE VERIFY" icon select

2.3.3.4 Reading Flash Content

Users can choose to readback and store contents of programmed flash by using EAB **READ** tool. Before reading flash content user need to choose file destination location and set name of file.

To set file destination, select set folder icon.

Figure 43 - Set destination folder

Next, enter the file name in filename entry field.

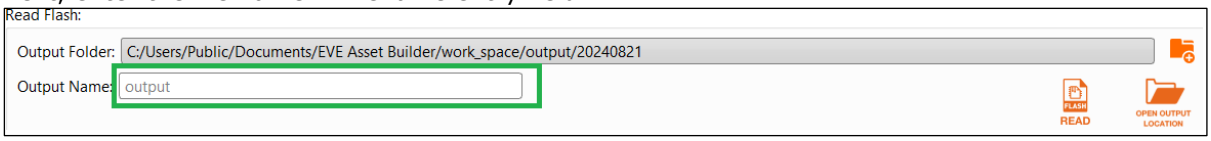


Figure 44 - Input file name

Click on **READ** icon to start reading back contents in flash.

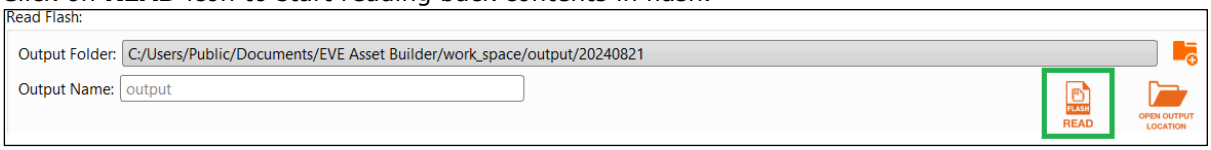
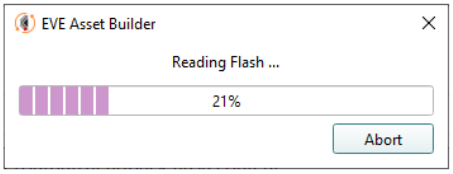


Figure 45 - "READ" icon select

A status indicator window and progress updates within the output log window, will display status of the flash during the reading process.



Output Log:

```
Reading Flash into readback content.bin...
Switch flash status from BASIC to FULL. Result: SUCCESS
Reading BT81X Flash Storage into "readback content.bin"
Read BT81X Flash Storage into "readback content.bin" successfully
```

Please note that **READ** function will retrieve the entire available memory, not just the file content. The output file size will correspond to the full flash memory size.

2.3.3.5 Blob Installation (Hidden tool for advanced users)

The "BLOB installation" tool is reserved for advanced users and is therefore hidden from the default application interface. This tool allows users to write or update only the first 4KB blob image without affecting the rest of the asset data.

Follow these steps to enable this hidden feature:

Step 1: Locate folder

Using Window's explorer, navigate to the EAB installation folder "EVE Asset Builder," typically found under default directory: "C:\Users\Public\Public Documents."

Step 2: Locate file

Locate and open "ui_config.json" file within this folder.

Step 3: File content

The file contains three lines of code as shown in diagram below.

```
1  {
2      "install_blob": false
3  }
```

Figure 46 - "ui_config.json" file content

Step 4: Edit file.

Change value of "install_blob" to true.

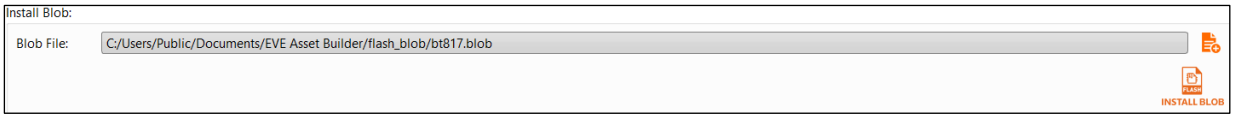
```
1  {
2      "install_blob": true
3  }
```

Figure 47 - Set "install_blob"

Step 5: Relaunch

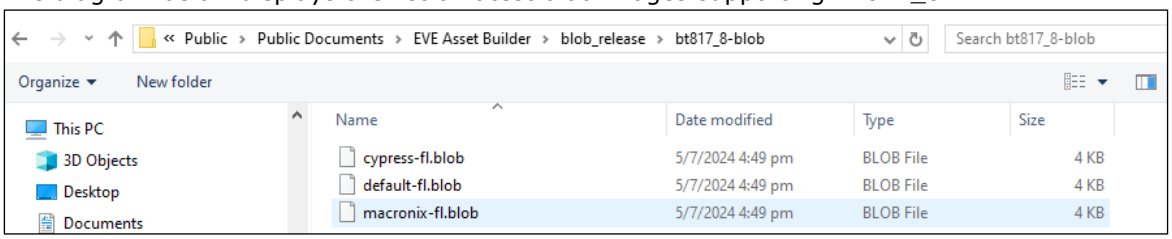
Save edited file and relaunch EAB.

A new tool **Install Blob**, will appear.



To access the latest blob images provided in version 2.12.0, users can find them under folders 'EVE Asset Builder\blob_release\bt815_6-blob' and 'EVE Asset Builder\blob_release\bt817_8-blob.'

The diagram below displays the list of latest blob images supporting BT817_8.



Select the desired blob image and click on **INSTALL BLOB** icon to write blob image.

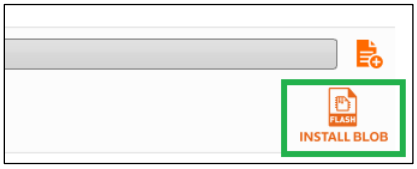


Figure 48 - "INSTALL BLOB" icon

2.3.4 Programming Flash via Raspberry Pi Pico Programmer Module

Loading flash images via Raspberry Pi Pico differs from using FT4222 and MPSSE programmer modules, and there are two methods to achieve this. Follow the steps below for instructions on how to download flash images.

Method 1 is suitable for users with EAB installed, while Method 2 allows for downloading a converted .UF2 file using Windows File Explorer without requiring EAB.

2.3.4.1 Method 1: Loading flash image using RP2040 via EAB

Step 1: Connect RP2040 to EVE as shown in section 2.1.

Step 2: Press and hold "BOOTSEL' button, then connect Pico to PC via USB cable.

Step 3: Wait for the "RPI-RP2" USB drive window to pop up.

Step 4: Launch EAB application from Windows program menu.

Step 5: Select the appropriate EVE chip and set "Programmer Module:" to "Raspberry Pi Pico.

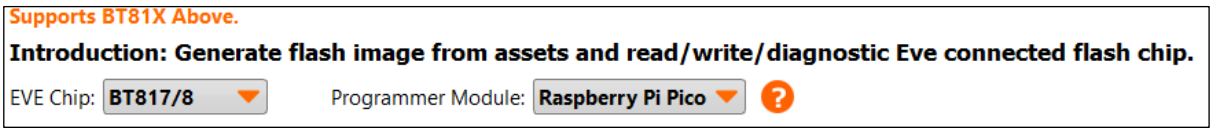


Figure 49 - EVE and programmer module select

Step 6: Click on add file icon to select the binary file (converted .bin file containing blob image and appropriate file format).

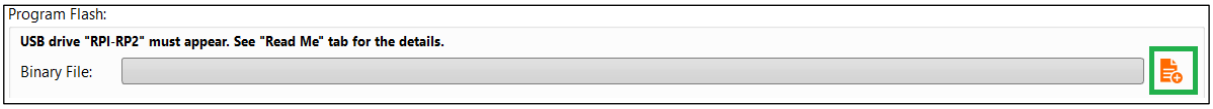
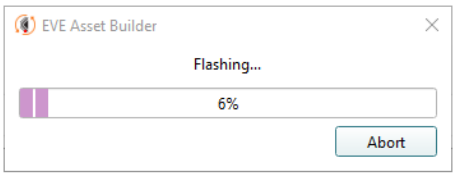


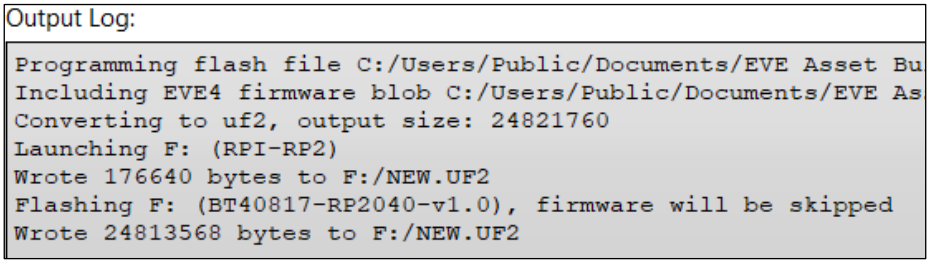
Figure 50 - Binary file select

Step 7: **Update** tool will be enabled after file selection. Click on the tool to start downloading.

Figure 51 - "UPDATE" icon select

Step 8: A status indicator window and progress updates within the output log window will display status of the flash during the flash update process.





2.3.4.2 *Method 2: Loading flash image using RP2040 through File Explorer*

Users must first convert the flash image file from binary format (.bin) to UF2, a format recognized by the Raspberry Pi Pico. This can be done using the EAB file conversion tool located under "Program Through Pico."

Follow these steps to convert the Binary File to a UF2 file:

Step 1: Launch EAB and select **FLASH UTILITIES**.

Step 2: Set **Programmer Module** to **Raspberry Pi Pico**.

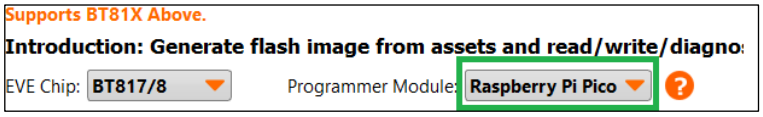


Figure 52 - Programmer module select

Step 3: In the "Convert Binary To UF2" section, click the add file icon next to "Binary File:" to select the binary file (converted .bin file containing blob and correct asset data format). Then, click the add file icon next to "UF2 File:" to choose the output file name and destination for the converted UF2 file.

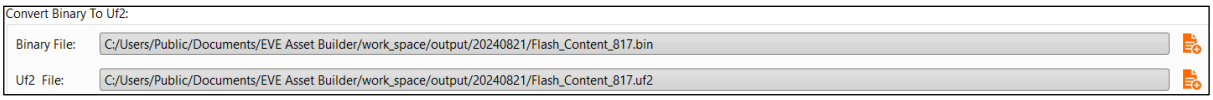


Figure 53 - Binary and UF2 file select

Step 4: **CONVERT** button will be enabled after selecting the files. Click it to start file conversion.

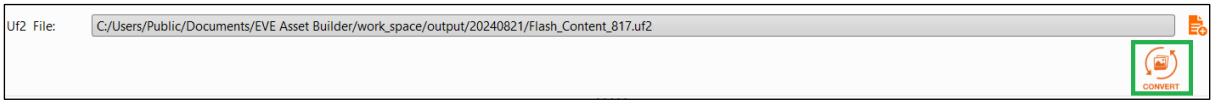


Figure 54 - "CONVERT" icon select

Once the file has been converted, you can download the UF2 file into flash using "File Explorer."

Follow these steps to download the flash image:

Step 1: Connect RP2040 to EVE as shown in section 2.1.

Step 2: Press and hold "BOOTSEL' button, then connect Pico to PC using a USB cable.

Step 3: Wait for the "RPI-RP2" USB drive window to pop up.

Step 4: Locate "eve_flash_pico.uf2" file under EAB working directory under default folder "C:\Users\Public\Documents\EVE Asset Builder\pico".

Step 5: Drag and drop "eve_flash_pico.uf2" file into "RPI-RP2" pop-up window.

Step 6: Wait till a new USB drive "BTEVE-FLASH" appear.

Step 7: Drag and drop converted UF2 file into "BTEVE-FLASH".

Step 8: Wait for the UF2 file to finish copying.

2.3.5 Flash Diagnosis

Flash Diagnosis is a tool provided by EAB to help developers troubleshoot communication errors and speed issues between EVE and the connected flash memory. This debug mode currently supports only the "FT4222" and "MPSSE" programmer modules. Figure 55 below shows the "Flash Diagnostic" tool interface provided by EAB.

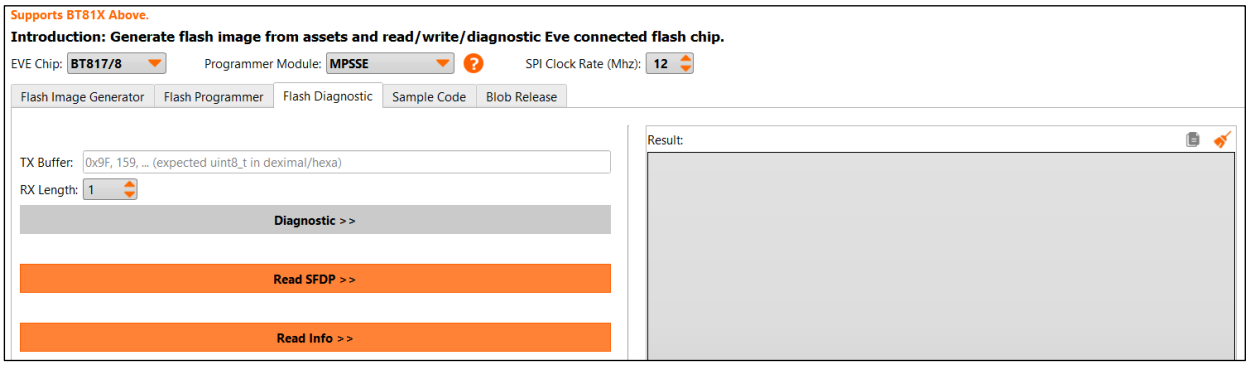


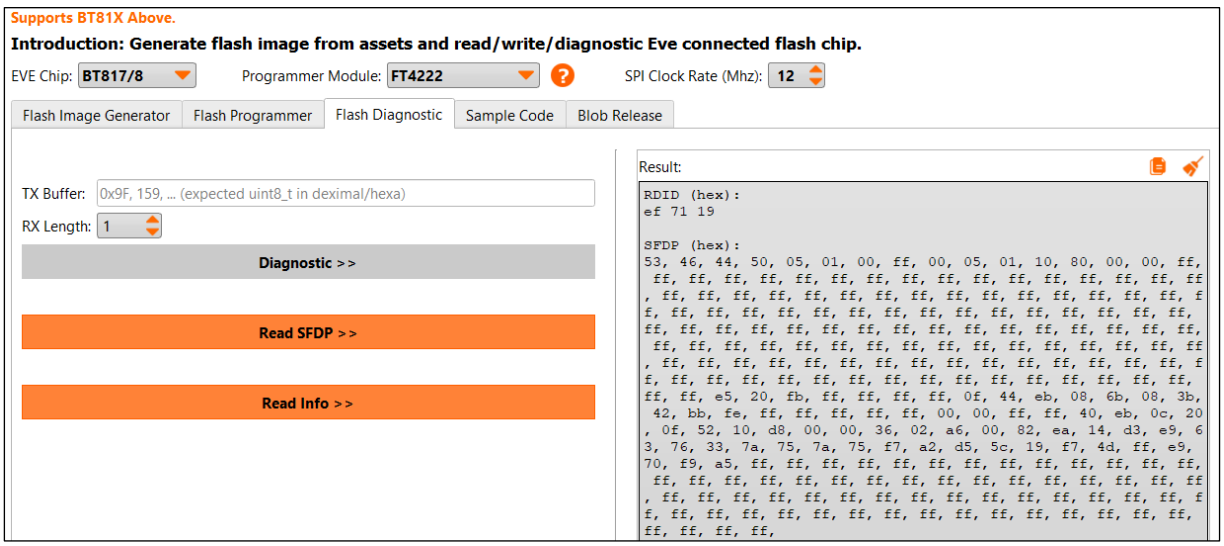
Figure 55 - Flash Diagnosis Tools

2.3.5.1 Diagnostic:

This debug mode helps developers send and receive data from flash.
Tx Buffer: Users can send a string of data with each byte separated by commas. Rx Length: Specify the number of bytes to receive."

2.3.5.2 Read SFDP:

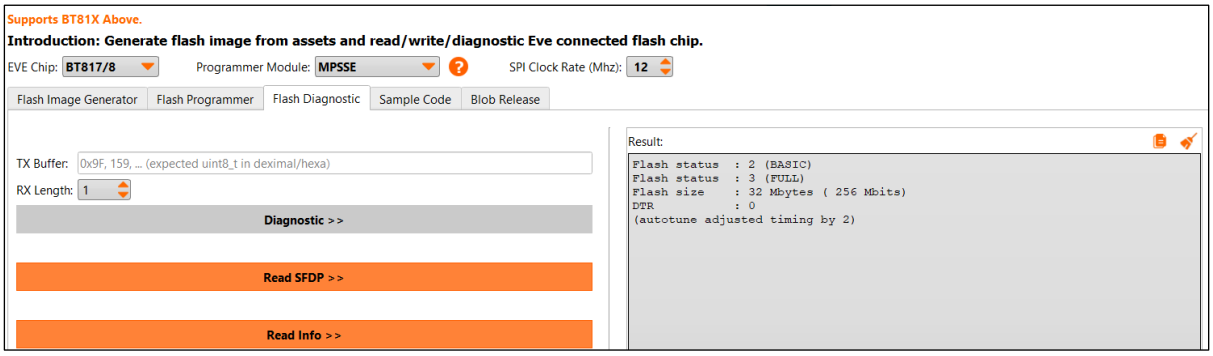
This debug mode reads and displays the RDID (Read Identification) and SFDP (Serial Flash Discoverable Parameters) structures from the flash chip. The diagram below shows information successfully read from the connected flash memory.



2.3.5.3 Read Info:

This debug mode provides information about the flash and the communication status it can establish with EVE. The diagram below illustrates an example of the readback flash information:

1. Flash Status:
    - 2 (Basic): Capable of achieving default or lower-speed communication.
    - 3 (Full): Capable of achieving the maximum speed at which the NOR flash can transfer data.
2. Flash Size: The available memory size is 32 MB (256 Mbits).
3. DTR (Double Transfer Rate): Not supported.



3 Designing in EVE using Developer's MCU

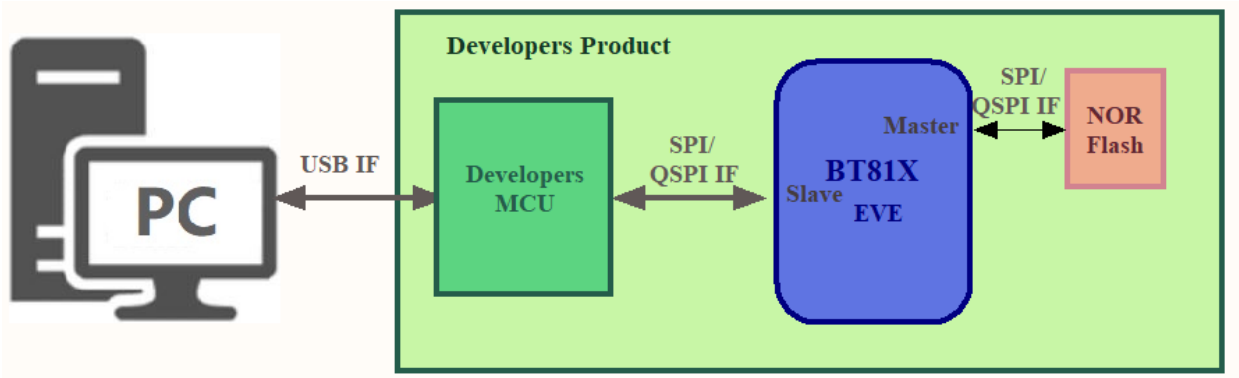


Figure 56 - EVE Communication via Developers MCU

For developers opting to use different microcontrollers in their designs, the following sections offer guidelines to ensure smooth design and implementation.

Figure 56 above illustrates how designer's micro-controller communicates with BT81X EVE.

3.1 Integrating MCU with EVE

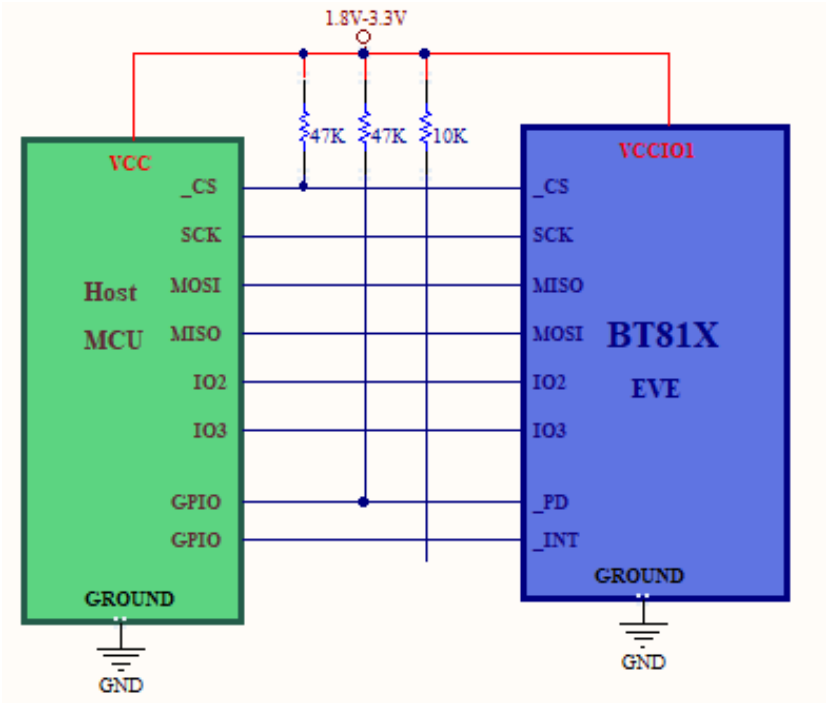


Figure 57 - Hardware Connection for Host MCU with QSPI Interface

Figure 57 above illustrates hardware connection for quad SPI interface between host MCU and EVE.

3.1.1 Pin Description

Host MCU is connected to the SPIS pins of EVE QSPI slave interface. Table 10 below provides a brief description of SPIS pins of EVE BT815 – BT818.

Pin No.	Pin Name	Type	Description
3	SCK	I	SPI clock input Powered from pin VCCIO1
4	MISO	I/O	SPI Single mode: SPI MISO output SPI Dual/Quad mode: SPI data line 1 Powered from pin VCCIO1
5	MOSI	I/O	SPI Single mode: SPI MOSI input SPI Dual/Quad mode: SPI data line 0 Powered from pin VCCIO1
6	CS_N	I	SPI slave select input Powered from pin VCCIO1
7	GPIO0/IO2	I/O	SPI Single/Dual mode: General purpose IO 0 SPI Quad mode: SPI data line 2 Powered from pin VCCIO1
8	GPIO1/IO3	I/O	SPI Single/Dual mode: General purpose IO 1 SPI Quad mode: SPI data line 3 Powered from pin VCCIO1
9	VCCIO1	P	I/O power supply for host interface pins. Support 1.8V, 2.5V or 3.3V.
11	INT_N	OD/O	Interrupt to host, open drain output(default) or push-pull output, active low
12	PD_N	I	Chip power down mode control input, active low. Connect to MCU GPIO for power management or hardware reset function, or pulled up to VCCIO1 through 47kΩ resistor and 100nF to ground. Powered from pin VCCIO1

Table 10 - EVE SPIS QSPI Pin Description.

3.1.2 Power Supply

EVE SPIS pins are powered by the "VCCIO1" power rail, which supports I/O voltage from 1.8V to 3.3V. Connect this pin to the appropriate supply base on the selected host MCU.

3.1.3 Pull-up resistors

Pull-up resistors are recommended on control lines “CS_N”, “INT_N” and “PD_N”.

This will ensure:

- chip select control is set to an inactive state during power-up.
- no false interrupts are triggered during power up.

3.2 Integrating External NOR Flash Memory with EVE

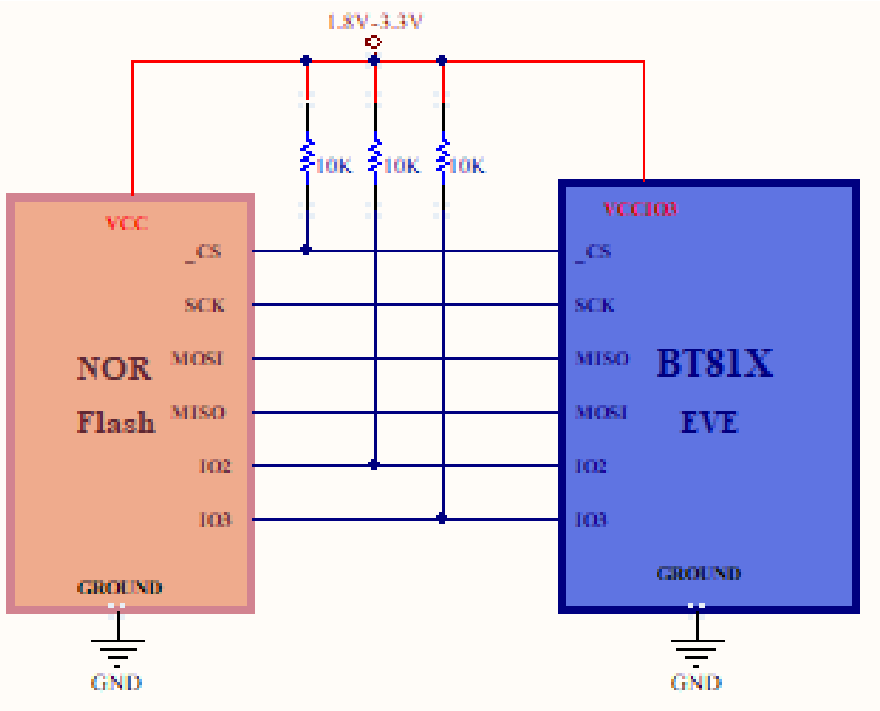


Figure 58 - Hardware Connection for NOR-Flash Memory

Figure 58 Figure 58 above illustrates hardware connection for quad SPI interface between EVE and NOR flash.

It is recommended that designers select their memory from the list of NOR flash options provided in Appendix A, as these have been tested and confirmed to be compatible with the BT81X EVE series.

3.2.1 Pin Description

Extended NOR flash memory are connected to SPIM pins of EVE QSPI master interface. Table 11 below provides brief description of SPIM pins of EVE BT815 – BT818.

Pin No.	Pin Name	Type	Description
14	SPIM_SCLK	O	SPI flash clock output line. Leave floating if not used. Powered from pin VCCIO3.
15	SPIM_SS_N	O	SPI flash chip select output line. Leave floating if not used. Powered from pin VCCIO3.
16	SPIM_MOSI	I/O	SPI flash MOSI line. Leave floating if not used. Powered from pin VCCIO3.
17	VCCIO3	P	I/O power supply for SPIM pins. Support 1.8V, 2.5V or 3.3V.
18	SPIM_MISO	I/O	SPI flash MISO line. Connect to GND if not used. Powered from pin VCCIO3.
19	SPIM_IO2	I/O	SPI flash IO2 line. Leave floating if not used. Powered from pin VCCIO3.
20	SPIM_IO3	I/O	SPI flash IO3 line. Leave floating if not used. Powered from pin VCCIO3.

Table 11 - EVE SPIM QSPI Pin Description.

3.2.2 Power Supply

The EVE SPIM pins are powered by the "VCCIO3" power rail. Connect this pin to the appropriate supply base on the selected flash memory.

3.2.3 Pull-up resistors

Pull-up resistors are recommended on control line “SPIM_SS_N” and data lines “SPIM_IO2” and “SPIM_IO3”.

This will ensure:

- chip select control is set to an inactive state during power-up.
- alternate functions such as HOLD# and RESET#, which are multiplexed with IO2 and IO3 in certain flash memories, are kept in an inactive state.

3.2.4 Layout Guideline

Given the high-speed signaling (up to 72MHz clock speed) between the EVE IC and NOR flash memory, it is important to place the memory close to the EVE IC and avoid running other signals across these communication lines. Ensuring equal-length routing traces, as illustrated in Figure 59, is recommended.

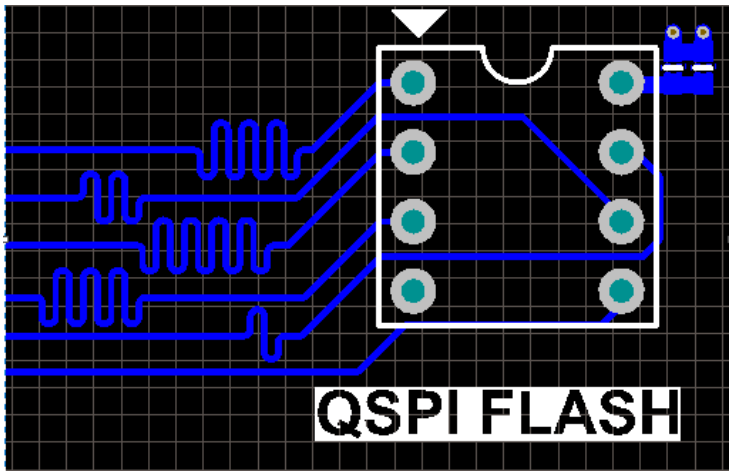


Figure 59 - Equal Trace Length for Signal Traces in QSPI Flash

3.3 Understanding Bridgetek EVE Flash Design Structure

3.3.1 Flash Interface

The BT81X series EVE implements a SPI/ QSPI master to connect to extended NOR flash memory. The interface will work at system clock speed (up to 72MHz) at 4-bit mode, providing maximum data throughput of 288Mbit/s.

3.3.2 Flash Detection

Upon boot-up, the flash is in the 'INIT' state. Once the flash is detected, the RDID of the connected flash will be read, and the appropriate flash driver will be loaded. After detection is complete, the flash will switch to the 'DETACHED' or 'BASIC' state, depending on whether the attached flash device was successfully identified.

Once "BASIC" mode is achieved, the host MCU can call the "FLASHFAST" command to switch the flash system to full-speed mode, setting the flash state to "FULL" and enabling QSPI communication.

For a detailed explanation of flash detection and command options, refer to Appendix A.

3.3.3 Flash Content

Content of EVE's external memory consists of two parts. First 4K of the memory, address 0 to 0xFFF are reserved for flash driver (BLOB). Graphic assets such as images, custom fonts, animation and videos will be stored after address 0x1000.

Designers are strongly encouraged to use Bridgetek's "EVE Asset Builders" for creating data assets and generating flash images. For details on the flash content format required by EVE and its operation, please refer to section 2.3.2.

3.3.4 Binary Firmware Driver (BLOB)

Blob firmware drivers for flash memories are required and loaded during initialization to set up flash communication and enable the QSPI channel. EVE's flash drivers offer BLOB support for flash devices manufactured by major companies such as Gigadevice, Infineon, ISSI, Macronix, Micron, and Winbond.

To enhance the performance of supported flash memories, Bridgetek has released a series of BLOB drivers for EVE BT815/6 and BT817/8 in the latest EAB version 2.12.0. BLOB drivers released are listed in Table 12 below.

	BLOB	Application	Note
1	default-fl.blob	All NOR flash device	Default BLOB image
2	cypress-fl.blob	Cypress NOR Flash with RDID hex code "01 60 XX".	Improved image for Cypress NOR Flash, enables faster programming speed
3	macronix-fl.blob	Macronix NOR Flash with RDID hex code "C2 20 XX", "C2 23 XX" and "C2 25 XX"	Improved image for Macronix NOR Flash, enables faster programming speed

Table 12 - BLOB Release for EAB Version 2.12.0.

Please refer to Appendix A for the list of supported QSPI NOR flash memory and the corresponding BLOB to use for optimize programming speed.

Designers can access the blob drivers in EAB working directory located at "EVE Asset Builder\blob_release".

3.4 Understanding Programming Modes

There are several methods for writing or programming data into flash memory. Selecting the right modes and commands can greatly influence programming time, particularly for high-capacity memory. The following section details the modes and commands designers should use to optimize programming time.

3.4.1 Basic Mode with command 'CMD_FLASHWRITE'

This is the most basic and slowest programming mode for flash memory; it should only be use for small capacity memories.

Step 1: **Attach Flash**: CMD_FLASHATTACH.

Step 2: **Erase Flash**: CMD_FLASHERASE. (Skip step if using fresh memory from factory)

Step 3: **Program data**: CMD_FLASHWRITE (x256 (max 4K) bytes/ command).

3.4.2 Full Mode with command 'CMD_FLASHUPDATE'

This mode is intended for programmed memories and should be chosen when only minor changes to the flash contents are anticipated.

Step 1: **Attach Flash**: CMD_FLASHATTACH.

Step 2: **Program blob**: CMD_FLASHWRITE (4K bytes of blob image).

Step 3: **Load blob**: CMD_FLASHFAST.

Step 4: **Program data**: CMD_FLASHUPDATE (4K bytes/ command).

3.4.3 Full Mode with command 'CMD_FLASHPROGRAM'

This mode offers the fastest programming speed and should be use for large-capacity memories.

Step 1: **Attach Flash**: CMD_FLASHATTACH.

Step 2: **Erase Flash**: CMD_FLASHERASE. (Skip step if using fresh memory from factory)

Step 3: **Program blob**: CMD_FLASHWRITE (4K bytes of blob image).

Step 4: **Load blob**: CMD_FLASHFAST.

Step 5: **Program data**: CMD_FLASHPROGRAM (4K bytes/ command).

Please note that command FLASHWRITE using "RAM_CMD" command buffer can also be used in full mode operation, but using FLASHPROGRAM with "RAM_G" graphics RAM will result in faster programming speed.

3.5 Code development for flash memory communication

EVE comes with a standard set of flash commands embedded in its ROM memory. Developers can consult section 5.3.2 and 5.3.3 for a list of available commands and their descriptions for code development.

Additionally, example codes for flash communication can be found in the "EVE Asset Builder" (EAB) under the "Sample Code" section within the "FLASH UTILITIES" tool. Copies of these codes are also attached under Appendix A.

The list of sample codes available are shown in the Table 13 below.

	Sample Code	Description
1	Flash Read	Read data from flash to main memory in fast mode.
2	Flash Write	Read content from a file and then write it to flash.
3	Flash Update	Update Flash content using command "cmd_flashupdate".
4	Flash Erase	Erase entire flash content.
5	Switch State	Change communication state from 'Basic' mode to 'Full' mode.
6	Blob Installation	Programming blob file (4KB) into the first block of flash.

Table 13 –Sample Codes Provided

4 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.

5 Appendix A - References

5.1 Supported Flash memory devices

Table 14- NOR Flash Memories Supported by BT81X. provides a list of NOR Flash memories supported by BT81X EVE series. Developers can refer to this list to select the appropriate blobs for optimized programming speed.

Vendor Part Number	Size (Mbit)	RDID	BLOB with Optimized Programming Speed
CYPRESS			
S25FL116K0XMFB040	16	01 40 15	default-fl.blob
S25FL132K0XMFA010	32	01 40 16	default-fl.blob
S25FL164K0XMFI011	64	01 40 17	default-fl.blob
S25FL064LABMFA001	64	01 60 17	cypress-fl.blob
S25FS128SAGMFI101*Note	128	01 20 18	default-fl.blob
S25FL128LAGMFA010	128	01 60 18	cypress-fl.blob
S25FL128LAGMFM010	128	01 60 18	cypress-fl.blob
S25FL256LAGNFV010	256	01 60 19	cypress-fl.blob
S25FL256LAGNFM010	256	01 60 19	cypress-fl.blob
ESMT/ EON			
EN35QX512A	512	1c 71 20	default-fl.blob
GIGADEVICE			
GD25Q32CSIGR	32	c8 40 16	default-fl.blob
GD25Q64CSIGR	64	c8 40 17	default-fl.blob
GD25Q64CSIG	64	c8 40 17	default-fl.blob
GD25Q127CSIGR	128	c8 40 18	default-fl.blob
GD25Q256DYIGR	256	c8 40 19	default-fl.blob
ISSI			
IS25LP016D-JBLE	16	9d 60 15	default-fl.blob
IS25LP032D-JBLE	32	9d 60 16	default-fl.blob
IS25WP064A-JBLE*Note	64	9d 70 17	default-fl.blob
IS25LP064A-JBLE	64	9d 60 17	default-fl.blob
IS25LP128-JBLE	128	9d 60 18	default-fl.blob
IS25WP128-JBLE*Note	128	9d 70 18	default-fl.blob
IS25LP256D-JLLE	256	9d 60 19	default-fl.blob
IS25LP512M-RMLE	512	9d 60 1a	default-fl.blob
MACRONIX			
MX25V1635FM2I	16	c2 23 15	macronix-fl.blob

MX25U3235FM2I-10G*Note	32	c2 25 36	macronix-fl.blob
MX25U6435FM2I-10G*Note	64	c2 25 37	macronix-fl.blob
MX25L6445EM2I-10G	64	c2 20 17	macronix-fl.blob
MX25L12845EZNI-10G	128	c2 20 18	macronix-fl.blob
MX25U12835FZ2I-10G*Note	128	c2 25 38	macronix-fl.blob
MX25L12835FM2I-10G	128	c2 20 18	macronix-fl.blob
KH25L12835FM2I-12G	128	c2 20 18	macronix-fl.blob
MX25L25635FZ2I-10G	256	c2 20 19	macronix-fl.blob
MX25U51245GZ4I00*Note	512	c2 25 3a	macronix-fl.blob
MICRON			
MT25QL128ABA8ESF-0AAT	128	20 ba 18	default-fl.blob
MT25QL128ABA1EW7-0SIT	128	20 ba 18	default-fl.blob
MT25QL256ABA1EW9-0SIT	256	20 ba 19	default-fl.blob
MT25QL512ABB8ESF-0AAT	512	20 ba 20	default-fl.blob
MT25QL01GBBB8ESF-0SIT	1024	20 ba 21	default-fl.blob
WINBOND			
W25Q16JVSSIQ	16	ef 40 15	default-fl.blob
W25Q16FWSIQ*Note	16	ef 60 15	default-fl.blob
W25Q32JVSSIQ	32	ef 40 16	default-fl.blob
W25Q32FWSIG*Note	32	ef 60 16	default-fl.blob
W25Q64JVSSIQ	64	ef 40 17	default-fl.blob
W25Q64FVSSIG	64	ef 40 17	default-fl.blob
W25Q128FVSIG	128	ef 40 18	default-fl.blob
W25Q128JVSIQ	128	ef 40 18	default-fl.blob
W25Q128JVSIM	128	ef 70 18	default-fl.blob
W25Q256JVEIQ	256	ef 40 19	default-fl.blob
W25M512JVEIQ	512	ef 71 19	default-fl.blob
W25Q512NWEIM*Note	512	ef 80 20	default-fl.blob
W25Q01NWZEIM*Note	1024	ef 80 21	default-fl.blob

Table 14- NOR Flash Memories Supported by BT81X.

*Note: Low operating voltage flash

5.2 Classification of EDF Type/Sub-Type Asset Data

5.2.1 Asset Type

type	subType	Description
0x01	-	Flash Driver (BLOB)
0x02	5.2.2	Bitmap
0x03	-	Bitmap PALETTED
0x04	-	Font Glyph (ASTC)
0x05	-	Extended Font
0x06	-	Legacy Font
0x07	5.2.3	Animation
0x08	-	Reserved
0x09	-	PNG/JPEG File
0x0A	5.2.4	DXT1 File
0x0B	5.2.5	Audio File
0x0C	-	Video File
0xFC	-	Padding Data
0xFD	-	EDF Block
0xFE	-	User Data

5.2.2 Sub-type: Bitmap

subType of Bitmap	Description
0	ARGB1555
1	L1
2	L4
3	L8
4	RGB332
5	ARGB2
6	ARGB4
7	RGB565
9	TEXT8X8
10	TEXTVGA
11	BARGRAPH
14	PALETTED565
15	PALETTED4444
16	PALETTED8
17	L2
37808	COMPRESSED_RGBA_ASTC_4x4_KHR
37809	COMPRESSED_RGBA_ASTC_5x4_KHR
37810	COMPRESSED_RGBA_ASTC_5x5_KHR
37811	COMPRESSED_RGBA_ASTC_6x5_KHR
37812	COMPRESSED_RGBA_ASTC_6x6_KHR
37813	COMPRESSED_RGBA_ASTC_8x5_KHR
37814	COMPRESSED_RGBA_ASTC_8x6_KHR
37815	COMPRESSED_RGBA_ASTC_8x8_KHR
37816	COMPRESSED_RGBA_ASTC_10x5_KHR
37817	COMPRESSED_RGBA_ASTC_10x6_KHR
37818	COMPRESSED_RGBA_ASTC_10x8_KHR
37819	COMPRESSED_RGBA_ASTC_10x10_KHR
37820	COMPRESSED_RGBA_ASTC_12x10_KHR
37821	COMPRESSED_RGBA_ASTC_12x12_KHR

5.2.3 Sub-type: Animation

subType of Animation	Description
0xaa00	ANIM_FLASH
0xaa01	ANIM_RAMG

5.2.4 Sub-type: DXT1

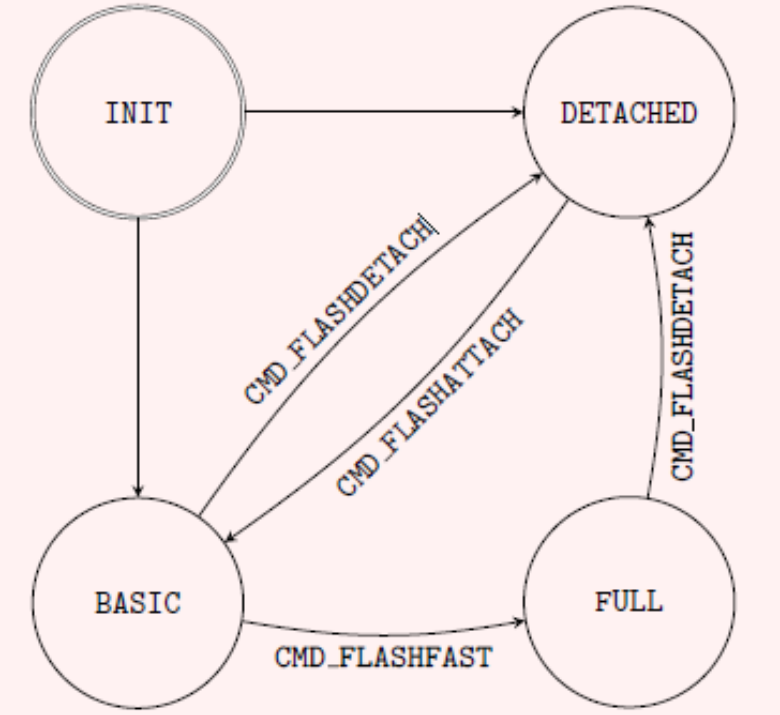
subType of DXT1	Description
0xdd00	DXT1_L1_RGB565
0xdd01	DXT1_L1_PALETTED565
0xdd02	DXT1_L2_RGB565
0xdd03	DXT1_L2_PALETTED565

5.2.5 Sub-type: Audio

subType of Audio	Description
0	Linear Sample format
1	uLaw Sample format
2	4-bit IMA ADPCM Sample format

5.3 Flash Interface and Commands

5.3.1 Flash Mode Transition State



The register REG_FLASH_STATUS indicates the state of the flash subsystem. At boot the flash state is FLASH_STATE_INIT. After detection has completed flash is in state FLASH_STATE_DETACHED or FLASH_STATE_BASIC, depending on whether an attached flash device was detected. If no device is detected, then all SPI output signals are driven low. When the host MCU calls CMD_FLASHFAST, the flash system attempts to go to full-speed mode, setting state to FLASH_STATE_FULL.

At any time, the user can call CMD_FLASHDETACH in order to disable flash communications. This tri-states all flash signals, allowing a suitably connected MCU to drive the flash directly. Alternatively, in the detached state, commands CMD_FLASHSPIDESEL, CMD_FLASHSPITX and CMD_FLASHSPIRX can be used to control the SPI bus.

If detached, the host MCU can call CMD_FLASHATTACH to re-establish communication with the flash device. Direct rendering of bitmaps from flash is only possible in FLASH_STATE FULL. After modifying the contents of flash, the MCU should clear the on-chip bitmap cache by calling CMD_CLEARCACHE.

5.3.2 Flash Commands

Command	DETACHED	BASIC	FULL	Operation
CMD_FLASHERASE		✓	✓	Erase all of flash
CMD_FLASHWRITE		✓	✓	Write data from **RAM_CMD** to blank flash
CMD_FLASHUPDATE		✓	✓	Read the flash and update to flash if different
CMD_FLASHPROGRAM			✓	Write data from **RAM_G** to blank flash
CMD_FLASHREAD		✓	✓	Reads data from flash to main memory
CMD_FLASHDETACH		✓	✓	Detach from flash
CMD_FLASHATTACH	✓			Attach to flash
CMD_FLASHFAST		✓		Enter full-speed(fast) mode
CMD_FLASHSPIDESEL	✓			SPI bus: deselect device
CMD_FLASHSPITX	✓			SPI bus: write bytes
CMD_FLASHSPIRX	✓			SPI bus: read bytes

Table 15 - Flash Interface States and Commands

5.3.3 Commands Usage

5.3.3.1 CMD_FLASHUPDATE

This command writes the given data to flash. If the data matches the existing contents of flash, nothing is done. Otherwise, the flash is erased in 4K units, and the data is written.

C prototype

void cmd_flashupdate (uint32_t dest,
uint32_t src,
uint32_t num);

Parameters

dest
Destination address in flash memory. Must be 4096-byte aligned.
Start address of first block is from **zero**.

src
source address in main memory **RAM_G**. Must be 4-byte aligned.

num
number of bytes to write, must be multiple of 4096

Command layout

+0	CMD_FLASHUPDATE (0xFFFF FF47)
+4	dest
+8	src
+12	num

Example

```
// The pseudo code below shows how to program the blob file to first block of flash
// Assume the flash is in detach mode and now attach it
cmd_flashattach();

// Now check if the flash is in basic mode after attaching
while (FLASH_STATUS_BASIC != rd8(REG_FLASH_STATUS));

//Write the BLOB file into the first block of flash
//Assume the BLOB file is in RAM_G
cmd_flashupdate(0, RAM_G, 4096);

// To check if the blob is valid, try to switch to full mode
cmd_flashfast(0);

while (FLASH_STATUS_FULL != rd8(REG_FLASH_STATUS));
```

5.3.3.2 CMD_FLASHDETACH

This command causes **EVE** to put the SPI device lines into hi-Z state. The only valid flash operations when detached are the low-level SPI access commands as following:

- CMD_FLASHSPIDESEL
- CMD_FLASHSPITX
- CMD_FLASHSPIRX
- CMD_FLASHATTACH

Refer to the section - Flash interface in BT817/8 datasheet.

C prototype

void cmd_flashdetach( );

Command layout

+0	CMD_FLASHDETACH (0xFFFF FF48)

5.3.3.3 CMD_FLASHATTACH

This command causes **EVE** to re-connect to the attached SPI flash storage. After the command, register **REG_FLASH_STATE** should be **FLASH_STATUS_BASIC**. Refer to the section - Flash interface in BT817/8 datasheet.

C prototype

void cmd_flashattach( );

Command layout

+0	CMD_FLASHATTACH (0xFFFF FF49)

5.3.3.4 CMD_FLASHFAST

This command causes the BT81X chip to drive the attached flash in full-speed mode, if possible. Refer to the section - Flash interface in BT817/8 datasheet.

C prototype

void cmd_flashfast (uint32_t result);

Parameters

result

Written with the result code. If the command succeeds, zero is written as a result. Otherwise, an error code is set as follows:

Error Code	Meaning
0xE001	flash is not supported
0xE002	no header detected in sector 0 – is flash blank?
0xE003	sector 0 data failed integrity check
0xE004	device/blob mismatch – was correct blob loaded?
0xE005	failed full-speed test – check board wiring

Command layout

+0	CMD_FLASHFAST (0xFFFF FF4A)
+4	result

Note: To access any data in flash by EVE, host needs send this command at least once to EVE in order to drive flash in full-speed mode. In addition, the flash chip is assumed to have correct blob file programmed in its first block (4096 bytes). Otherwise, it will cause the failure of this command.

5.3.3.5 CMD_FLASHSPIDESEL

This command de-asserts the SPI CS signal. It is only valid when the flash has been detached, using **CMD_FLASHDETACH**.

C prototype

void cmd_flashspidesel ();

Command layout

+0	CMD_FLASHSPIDESEL (0xFFFF FF4B)

5.3.3.6 CMD_FLASHSPITX

This command transmits the following bytes over the flash SPI interface. It is only valid when the flash has been detached, using **CMD_FLASHDETACH**.

C prototype

void cmd_flashspitx (uint32_t num);

Parameters

num

number of bytes to transmit

Command layout

+0	CMD_FLASHSPITX (0xFFFF FF4C)
+4	num
byte1…byten	the data to transmit

Example

Transmit single-byte 06:

```
cmd_flashdetach();
cmd_flashspidesel();
cmd_flashspitx(1);
cmd(0x00000006);
```

5.3.3.7 CMD_FLASHSPIRX

This command receives bytes from the flash SPI interface, and writes them to main memory. It is only valid when the flash has been detached, using **CMD_FLASHDETACH.**

C prototype

void cmd_flashspirx (uint32_t ptr,
uint32_t num);

Parameters

ptr

destination address in **RAM_G**

num

number of bytes to receive

Command layout

+0	CMD_FLASHSPIRX (0xFFFF FF4D)
+4	ptr
+4	num

Example

Read 3 bytes from SPI flash to main memory locations 100,101,102:

```
cmd_flashdetach();
cmd_flashspidesel();
cmd_flashspirx(100, 3);
```

5.3.3.8 CMD_CLEARCACHE

This command clears the graphics engine's internal flash cache. It should be executed after modifying graphics data in flash by **CMD_FLASHUPDATE** or **CMD_FLASHWRITE**, otherwise bitmaps from flash may render "stale" data. It must be executed when the display list is empty, immediately after a **CMD_DLSTART** command. Otherwise, it generates a coprocessor fault ("display list must be empty") and sets **REG_PCLK** to zero.

C prototype

void cmd_clearcache ( );

Command layout

+0	CMD_CLEARCACHE (0xFFFF FF4F)

Example

```
//Flash is in Full mode and has the right content working with EVE
//Update the 4th block of flash chip with new bitmap data located at RAM_G+1024
cmd_flashupdate(4*4096, RAM_G+1024, 4*4096);

//To continue rendering the bitmap data in flash, need to call cmd_clearcache
cmd_dlstart();
cmd_clearcache();
cmd_swap();
```

5.4 Sample Code for Flash Communication

5.4.1 Flash Read

```
// The pseudo code to read flash content to a file
// By default, the flash device operates in basic mode,
// which is slow. However, after programming the blob,
// it can be switched to fast mode by issuing the
// cmd_flashfast command. Once the flash is in full mode,
// the speed will be significantly improved.

// Assume the flash is in detach mode and now attach it

cmd_flashattach();
// Switch the flash into FAST mode
cmd_flashfast();
// Now check if the flash is in FAST mode
while (FLASH_STATUS_FULL != rd8(REG_FLASH_STATUS));

// Get flash size
flash_size = rd32(REG_FLASH_SIZE);
// Open file to write
FILE * fp = fopen(file_name, "wb+");
// Read flash content
loop while flash_size > 0
    cmd_flashread(buffer, src_flash, buffer_size);

    // Write buffer to file
    fwrite(buffer, 1, buffer_size, fp);

    // Update flash address and flash size
    src_flash += buffer_size;
    flash_size -= buffer_size;
```

5.4.2 Flash Write

```
// The pseudo code to read content from a file then write to flash

// Open file to read its content
FILE * fp = fopen(file_name, "rb+");
// Then get file length
file_length = get_file_length(fp);

// Now write to flash chip
loop while file_length > 0:
    // Read from file to buffer
    fread(buffer, 1, buffer_size, fp);

   // Write buffer to flash
   cmd_flashwrite(flash_addr, buffer_size, buffer);

   // Update flash address
   flash_addr += buffer_size;
   file_length -= buffer_size;
```

5.4.3 Flash Update

```
// The pseudo code to update flash content

// Open file to read its content
FILE * fp = fopen(file_name, "rb+");
// Then get file length
file_length = get_file_length(fp);

// Now update flash chip
loop while file_length > 0:
    ram_length = 0;
    ram_addr = RAM_G;  // RAM_G is starting address of Graphic RAM
    loop while ram_length < RAM_SIZE and file_length > 0:
        // Read from file to buffer
        fread(buffer, 1, buffer_size, fp);
        // Write buffer to RAM
        wrMem(ram_addr, buffer, buffer_size);
        file_length -= buffer_size;
        ram_addr += buffer_size;
        ram_length += buffer_size;
    // Write RAM content to flash
    cmd_flashupdate(flash_addr, RAM_G, RAM_SIZE);
    // Update flash address
    flash_addr += RAM_SIZE;
```

5.4.4 Flash Erase

```
// The pseudo code to erase flash content
// Erase entire flash
cmd_flasherase();

// Check first 4KB to ensure flash is erased
BUF_SIZE = 4 * 1024; // 4 KB
// Read first 4KB into RAM_G
cmd_flashread(RAM_G, 0, BUF_SIZE);
// Read from RAM_G to buffer
rdMem(RAM_G, buffer, BUF_SIZE);
// Loop to check buffer content
for i in [0, BUF_SIZE]:
    if buffer[i] != 0xFF:
        print("Flash is not erased!");
        break;
print ("Flash is erased!");
```

5.4.5 Switch State

```
// The pseudo code to switch flash state
Error code:
- 0x0    command succeeds
- 0xffff command fails (invalid transition state)
- 0xe001 flash is not attached
- 0xe002 no header detected in sector 0 - is flash blank?
- 0xe003 sector 0 data failed integrity check
- 0xe004 device/blob mismatch - was correct blob loaded?
- 0xe005 failed full-speed test - check board wiring

// Read current flash state
current_state = rd8(REG_FLASH_STATUS);
error_code = 0
```

```
if FLASH_STATUS_DETACHED == next_state:
    cmd_flashdetach();
else if FLASH_STATUS_BASIC == next_state:
    if FLASH_STATUS_FULL == current_state:
        while (FLASH_STATUS_DETACHED != rd8(REG_FLASH_STATUS)):
            cmd_flashdetach();
    cmd_flashattach();
else if FLASH_STATUS_FULL == next_state:
    if FLASH_STATUS_BASIC != current_state:
        while (FLASH_STATUS_BASIC != rd8(REG_FLASH_STATUS)):
            cmd_flashattach();
    cmd_flashfast();

    // Read the return code in cmd_buffer
    #define FIFO_SIZE_MASK              (4095)   // FIFO valid range from 0 to 4095
    #define FIFO_BYTE_ALIGNMENT_MASK    (0xFFC)
    ret_addr = (cmd_fifo_wp - 4) & FIFO_SIZE_MASK;
    ret_addr = (ret_addr + 3) & FIFO_BYTE_ALIGNMENT_MASK;

    error_code = rd32(RAM_CMD + ret_addr);
```

5.4.6 Blob Installation

```
// The pseudo code below shows how to program the blob file to first block of flash

// Assume the Blob file(4096 bytes) has been downloaded to RAM_G
// Assume the flash is in detach mode and now attach it
cmd_flashattach();

// Now check if the flash is in basic mode after attaching
while (FLASH_STATUS_BASIC != rd8(REG_FLASH_STATUS));

cmd_flashupdate(0, RAM_G, 4096);

// To check if the blob is valid , try to switch to full mode
cmd_flashfast();
while (FLASH_STATUS_FULL != rd8(REG_FLASH_STATUS));
```

Acronyms and Abbreviations

Terms	Description
BLOB	Binary Large Object
DTR	Double Transfer Rate
EAB	EVE Asset Builder
EDF	EVE Flash Description File
EVE	Embedded Video Engine
I2C	Inter-Integrated Circuit
MCU	Micro Controller Unit
MPSSE	Multi-Protocol Synchronous Serial Engine
PC	Personal Computer
QSPI	Quad Serial Peripheral Interface
Rx	Receive
SFDP	Serial Flash Discoverable Parameters
SPI	Serial Peripheral Interface
SPIM	Serial Peripheral Interface Master
SPIS	Serial Peripheral Interface Slave
Tx	Transmit
USB	Universal Serial Bus
VCP	Virtual COM Port

Appendix B – List of Tables & Figures

List of Figures

List of Tables

Appendix C – Revision History

Document Title: BRT_AN_096 Working with EVE External NOR Flash Memory
Document Reference No.: BRT_000438
Clearance No.: BRT#233
Product Page: https://brtchip.com/product-category/products/
Document Feedback: Send Feedback

Revision	Changes	Date
1.0	Initial Release	10-04-2025