USER GUIDE

Bridgetek

BRIDGING TECHNOLOGY

EVE Asset Builder 2.11 User Guide

Document Version: 2.4

Issue Date: 10 April 2024

Contents

I. Preface

A. Purpose

This document describes the features and procedures involved in using the **E**VE **A**sset **B**uilder (**EAB**).

B. Related documents

- BT81X Series Programming Guide
- FT81x BT88x Series Programming Guide
- FT80x Series Programming Guide
- BT817/8 Datasheet
- BT817AQ For Automotive Datasheet
- BT815/6 Datasheet
- BT88X Datasheet
- FT81X Datasheet
- FT801 Datasheet
- FT800 Datasheet

C. Acronyms and Abbreviations

Terms	Description
EVE	Embedded Video Engine
EAB	EVE Asset Builder
HAL	Hardware Abstraction Layer
MSVC	Microsoft Visual Studio C++
SPI	Serial Peripheral Interface
UI	User Interface
ASTC	Adaptive Scalable Texture Compression
MPSSE	Multi-Protocol Synchronous Serial Engine
SFDP	Serial Flash Discoverable Parameter

II. Overview

A. Introduction

The EVE Asset Builder (EAB) is a software application designed specifically for Windows operating systems. It includes a variety of utilities that facilitate the generation of resources for EVE series devices. The aim of this document is to provide a detailed guide on effectively leveraging the tools within the EAB.

B. Key Features

The following are the key features of EVE Asset Builder:

- Image Utilities: Convert bitmaps, validate PNG/JPEG, and optimize JPEG
- Video Converter: Convert a video to MJPEG format
- Audio Converter: Convert an audio to EVE-compatible format
- Font Converter: Convert a font to Legacy/Extended format
- Animation Converter: Convert a gif file or a sequence of images to an animation
- Flash Utilities: Generate/Program/Read a flash file
- Asset Compressor: Compress/Validate an asset to EVE-compatible format
- Custom Touch Compiler: Compile a custom touch firmware
- Bin2C: Transform the binary file into a C array
- Disassembler: Translate the EVE instructions into human-readable text
- Display List Optimizer: Optimize display list commands

C. What's new?

Feature Updates:

- ASTC Encoder
  - Users can now select any version of astcenc for improved flexibility.
  - Updated ASTC encoder to version 4.x.
  - Added ASTC Encoder link into the Info tab for quick access.
- Font Converter
  - Removed the output file _charmap.txt for a cleaner output.
- Flash Utilities
  - Added a separated option for BT817A for better usability.
  - Added subtype for animation for enhanced animation support.
- Animation Converter
  - Reversed the arrangement of .data and .object for user convenience.
  - Remove the .anim file in the output folder.
- Bin2C
  - Align the converted value to the left for improved readability.
- SampleApp
  - Improved the generated SampleApp for DXT1 bitmap generation.
- Custom Touch Compiler
  - Add new compiler for BT88X/FT81X
  - Include new macros and pseudo code in the Read Me tab.
- Installer
  - The setup file is signed with a company certificate.

Resolved Issues:

- Animation Converter
  - Resolved the issue associated with 2-frame GIF input.
  - Addressed the bug that occurs when disabling the Tile Size.
- Flash Programmer
  - Corrected the text argument of Eve chip generation.
- Font Converter
  - Fixed the bug related to the wrong baseline in Font Converter.
- SampleApp
  - Resolved the issue of excessive restore_context() command.
- Image Utilities
  - Layout of Input File is no longer cut at the top.
- DXT1
  - Addressed the issue where the SampleApp does not display DXT1 images correctly.
- Session
  - Resolved the issue with loading input files.

D. Credits

Open-Source Software

- Qt: *https://www.qt.io/*
- Python: *https://www.python.org/*
- ASTC Encoder: *https://github.com/ARM-software/astc-encoder*
- FFmpeg: *https://www.ffmpeg.org/*
- OptiPNG: *http://optipng.sourceforge.net/*
- Jpegtran: *http://jpegclub.org/jpegtran/*
- QtProgressCircle: *https://github.com/mofr/QtProgressCircle*
- libraqm: *https://github.com/HOST-Oman/libraqm*

Icons Copyright
Icons made by *Freepik*, *Smashicons*, *Pixel perfect*, *Dave Gandy*, *Royyan Wijaya*, *Recieve* icon by *Icons8*

Audio Copyright
Audio for testing by *Lesfm* from *Pixabay*

III. Setup and Installation

A. System Requirements

To install the application, ensure that your PC meets the requirements recommended below:

- ✓ Recommended Windows 10 or above; alternatively, Windows 8 or 7 with the latest windows updates
- ✓ 64-bit platform
- ✓ 1.6GHz or faster processor
- ✓ 1GB of RAM (1.5GB if running on a virtual machine)
- ✓ A multi-core CPU is highly recommended
- ✓ At least 512MB of hard disk space
- ✓ Display resolution 1080 x 800 pixels or higher
- ✓ "Write" permission to the installation folder

B. Installing EVE Asset Builder

The following steps will guide you through the EVE Asset Builder *Setup/Installation* process.

i. Download the package from *https://brtchip.com/toolchains/#EVEAssetBuilder*.

ii. Extract the zip file contents. Double click on the file **EVE-Asset-Builder-setup.exe**

iii. The EVE Asset Builder Setup Wizard is displayed along with a Welcome message.

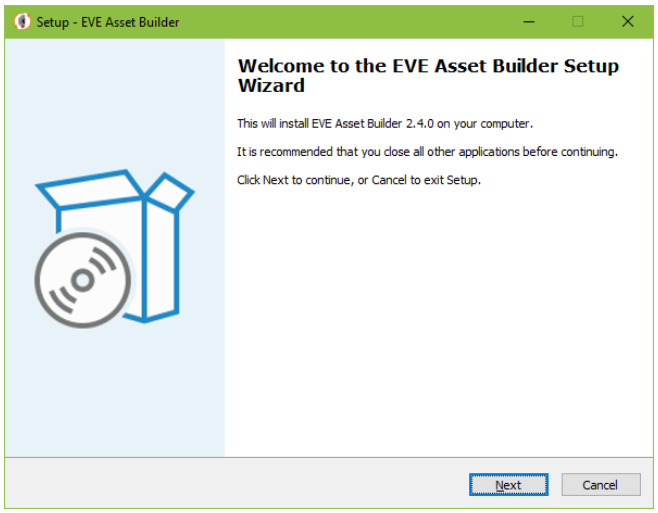


Figure 1 EVE Asset Builder Setup Wizard

iv. Click **Next** to select a "Destination Folder" for installing the files. Accept the default folder or click **Browse** to specify a different location. Click **Next** to confirm the destination folder and continue.

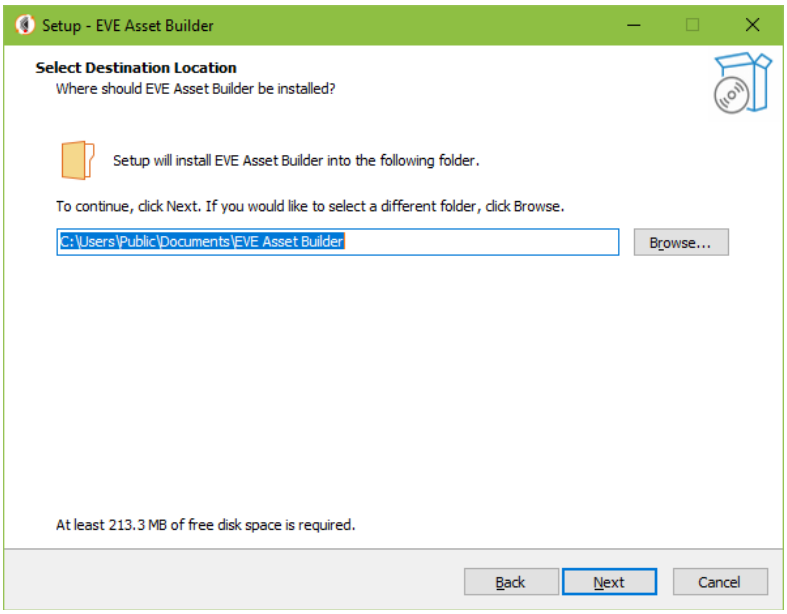


Figure 2 EVE Asset Builder Setup – Select the destination folder

v. In the **Select Additional Tasks** window, check the **"Create a desktop / Create Quick Launch shortcut"** boxes, to have the EVE Asset Builder icon and Quick Launch shortcut displayed on the desktop if required. Click **Next** to prepare for the installation.

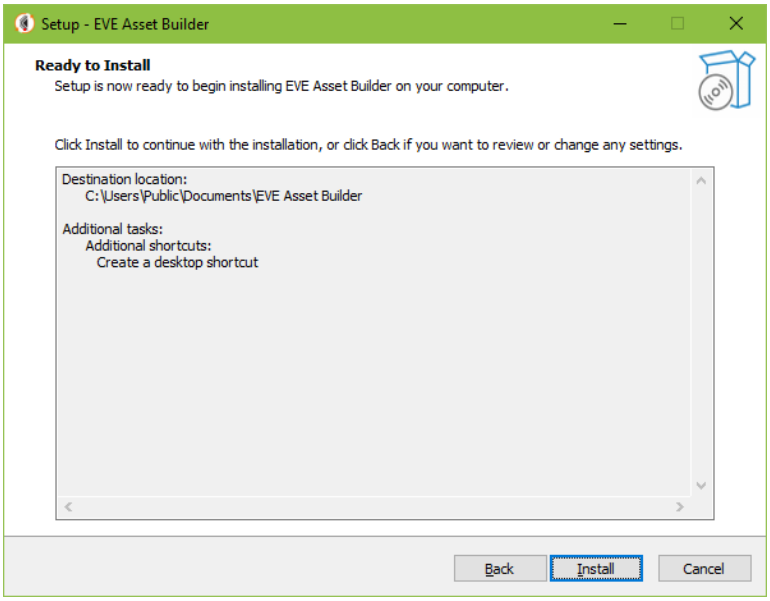


Figure 3 EVE Asset Builder Setup – Ready for Installation

vi. Click **Install** to start the installation. A progress bar indicates that the installation is in progress.

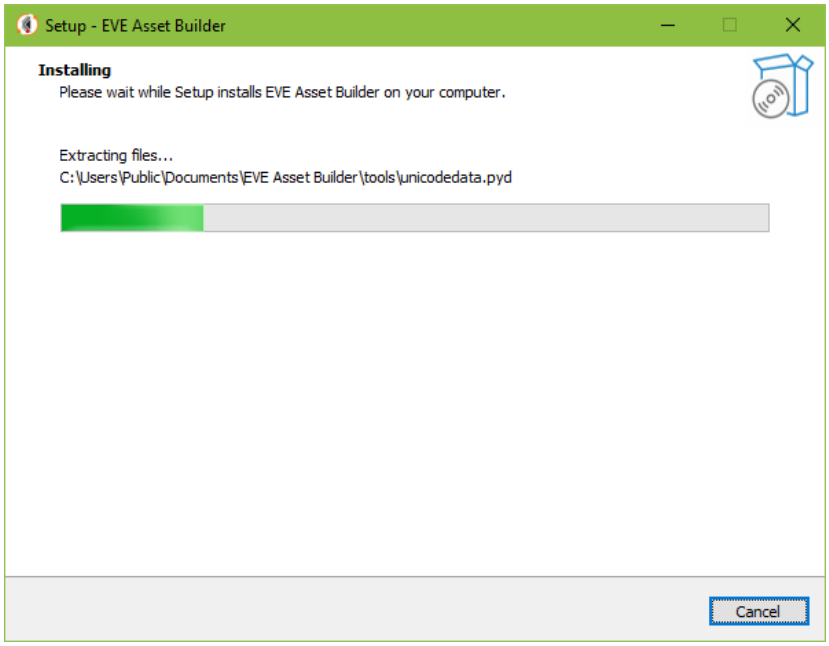


Figure 4 EVE Asset Builder Setup – Installing in Progress

vii. Upon successful installation, click **Finish**. The EVE Asset Builder application UI is displayed.

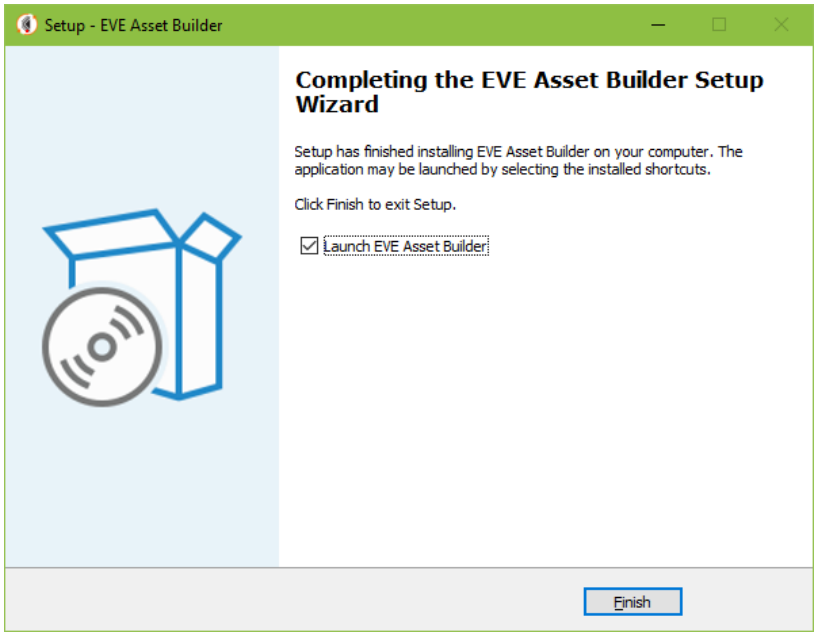


Figure 5 EVE Asset Builder Setup – Finish

IV. Getting Started

A. EVE Chip Series

The available EVE chip generations - BT81X, BT88X/FT81X, and FT80X - can affect which utilities are available once chosen.

BT81X refers to: BT815/6, BT817/8
BT88X refers to: BT880/1/2/3
FT81X refers to: FT810/1/2/3
FT80X refers to: FT800/1

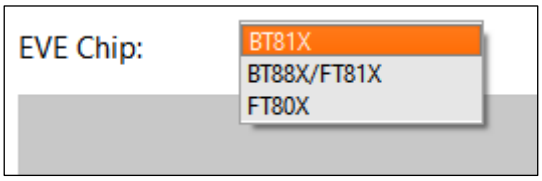


Figure 6 EVE chip series

B. ASTC Encoder

Starting from EAB 2.11, users can download various versions of ASTC encoders. Subsequently, they can select the most suitable version to utilize alongside EAB.

Figure 7 ASTC Encoder

The "ASTC Encoder" button is positioned on the left-side panel. When pressed, a Select Folder Dialog will appear, allowing the user to choose a folder containing ASTC encoders with suffixes such as sse2, sse4.1, and avx2. EAB will subsequently read the CPU specifications and automatically select a compatible encoder. The default version employed in EAB is 4.5.

Note: Official link to ASTC encoder releases:
https://github.com/ARM-software/astc-encoder/releases

C. Session

This functionality allows for the recording of user activities for future reference. Three buttons located on the left edge can be used to load, start, or stop sessions.

START SESSION: Start recording actions from the user, all information is stored in a session file.

STOP SESSION: Stop recording. The user has the option to save or discard the session.

LOAD SESSION: Read a session file and restore all inputs.

Note: Upon starting EAB, a default session will be created. The user can then choose to either retain or discard it.

Figure 8 Session

1. Start Session

Input and output folders will be set as specified in the session dialog below for all utilities of EAB. The output folder must be empty. A session should be named to find easily at the time of loading. Once a session has started, all inputs from users will be recorded and then saved to a file called *<session name>.eab*

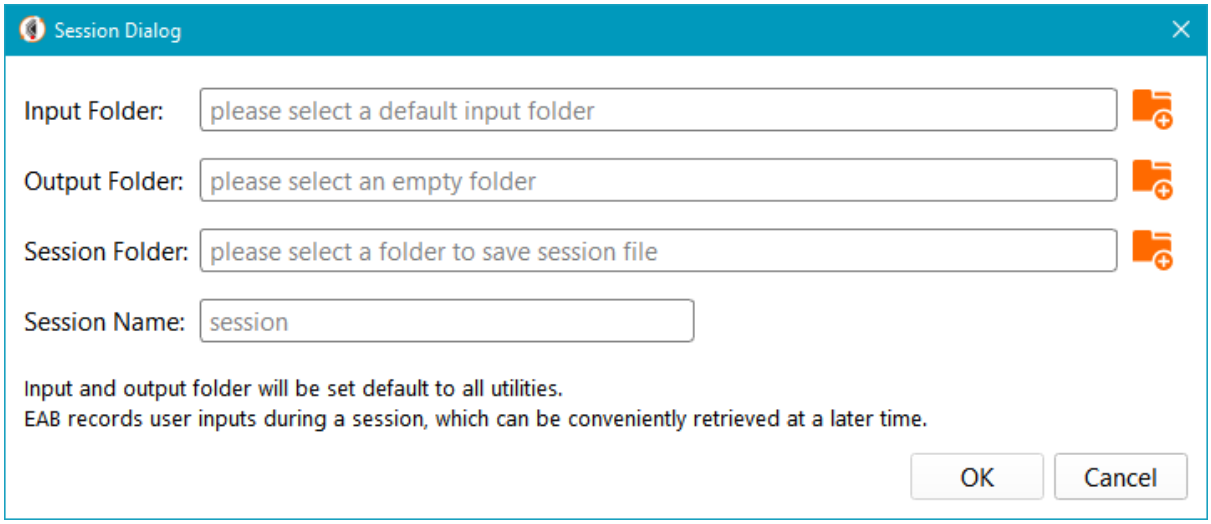


Figure 9 Session Dialog

2. Load Session

Load a session that has previously been saved. Every user input will be filled in again.

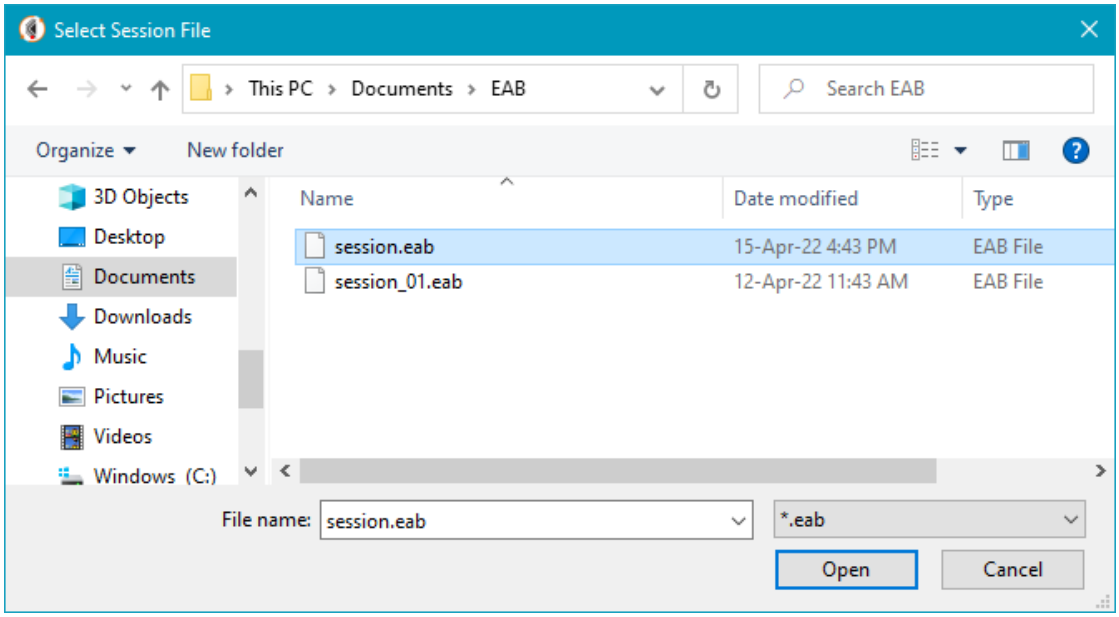


Figure 10 Select session file

3. Stop Session

All user actions are stored in the session file. EAB ceases collecting user inputs upon stopping a session. The "Stop Session" button is enabled only when a session is started or loaded.

D. Command Prompt

Located at the left edge, the *Command Prompt* button will open a command window. The paths to *eab_tools.exe* and ffmpeg.exe are temporarily set to the **PATH** variable so that users can execute these tools without worrying about where they are located.

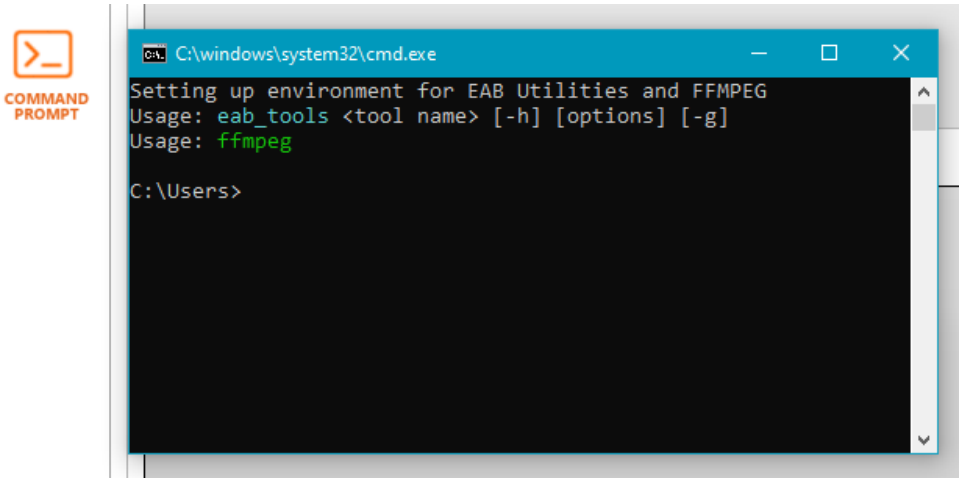


Figure 11 Command Prompt Window

Upon execution, every tool in the EVE Asset Builder (EAB) generates a command-line text in the Log Window. This text can be copied and pasted by users into the command window, allowing them to execute it from there.

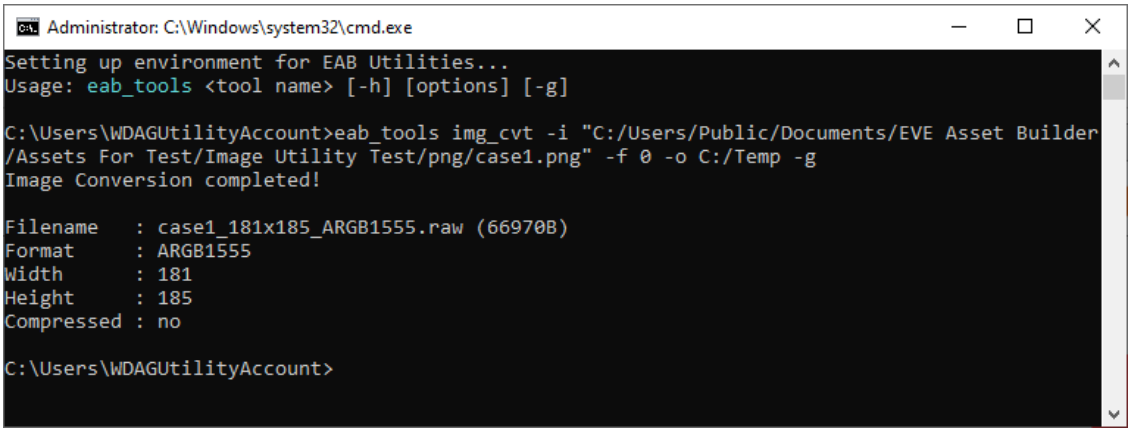


Figure 12 Example of converting an image

E. Log Window

The *Log Window* displays information related to inputs, outputs, warnings, or errors to assist users in effectively utilizing EAB tools.

Figure 13 Log Window

CLEAR: erase all content in Log Window

COPY: if there is any text selected, copy it to the clipboard. Otherwise, copy all the content.

Output Log Tab: show the log output.

Command Line Tab: show the command-line texts which have been executed.

F. Naming Convention and Usage of Output Files

The output files generated by each tool in the EVE Asset Builder (EAB) adhere to a specific naming convention, as outlined in the following table. The convention and its intended use are described in detail. It should be noted that <input> refers to the base name of an input file, such as "lena" for an input file named "lena.png", while <output> is defined by the user.

Tool	Output Naming Convention	Example of input and output file	Usage
Image Converter	Uncompress: <*input*>_ <*width*>x<*height*>_ <*output format*>. <*raw|rawh*>	lena.png -> lena_185x150_L8.raw, lena_185x150_L8.rawh	- Load the .raw or .rawh file to RAM_G - Decode it using cmd_setbitmap
	Compress: <*input*>_ <*width*>x<*height*>_ <*output format*>. <*bin|binh*>	lena.png -> lena_185x150_L8.bin, lena_185x150_L8.binh	- Load the .bin or .binh file to RAM_CMD/Flash - Decompress it using cmd_inflate, cmd_inflate2 - Decode it using cmd_setbitmap
	ASTC: <*input*>_ <*width*>x<*height*>_ <*output format*>.astc	lena.png -> lena_185x150_ASTC_5x5.astc	- Decode by using the ASTC encoder, the output is typically a PNG image.
PNG/JPEG Validator	<*input*>.opt.<*extension*> Note: This tool may not generate any output file if an input image is already good for EVE	lena.png -> lena.opt.png	- Load the output file to RAM_G or RAM_CMD - Decode it using cmd_loadimage
JPEG Optimizer	<*input*>.opt.jpg	lena.jpg -> lena.opt.jpg	- Load the output file to RAM_G or RAM_CMD - Decode it using cmd_loadimage
Video Converter	<*input*>.avi	big_buck.avi -> big_buck.avi	- Load the output file to RAM_G (media FIFO) or RAM_CMD - Decode it using cmd_playvideo

Audio Converter	<*input*>.raw	happy_summer.mp3 -> happy_summer.raw	- Load the output file to RAM_G (media FIFO) - Play it by controlling the appropriate registers
Font Converter	Legacy Format: <input>_<font size>_ <bitmap format>. <raw|rawh>	symbol.ttf -> symbol_25_L8.raw, symbol_25_L8.rawh	- Load the raw file to a predefined offset in RAM_G - Decode it using cmd_setfont, cmd_setfont2
	Extended Format: <*input*>_<font size>_ <bitmap format>. <xfont|glyph>	arial.ttf -> arial_39_ASTC.xfont, arial_39_ASTC.glyph	- Load .xfont file to RAM_G, .glyph file to RAM_G/FLASH - Decode them using cmd_setfont or cmd_setfont2
Animation Converter	FLASH: <*input*>.anim.flash Note: .anim.flash is a flash image. It can be written to flash chip, then play.	bicycle.gif -> bicycle.anim.flash	- Utilize the .anim.flash file for generating a flash image, or alternatively, write the content of the .anim.flash file directly to the flash chip at the predefined offset. - Use cmd_anim* to play the animation.
	RAM_G: <*input*>.anim.ram_g	abstract.gif -> abstract.anim.ram_g	- Load the .anim.ram_g file to RAM_G at the predefined offset - Use cmd_animstartram, and cmd_animframeram to play the animation
Flash Image Generator	<*output*>.bin <*output*>.map <*output*>.edf	abstract.anim.flash, lena.raw -> flash_16MB.bin,	- Write the flash image .bin to flash chip - See the offset of each asset in the map or the edf file
		flash_16MB.map	Each line contains the asset information in the format: *name:offset:length*
		flash_16MB.edf	Refer to section EDF Block
Asset Compressor	Zlib: <*output*>.zlib	arial.xfont, big_buck.avi -> inflate.zlib	- Load compressed file to flash or RAM_G - Use cmd_inflate, cmd_inflate2 to decompress the assets
	Zopfli: <*output*>.zopfli	arial.xfont, big_buck.avi -> inflate.zopfli	
Custom Touch Firmware Compiler	<*input*>.load.bin, <*input*>.load.h	firmware.c -> firmware.load.bin, firmware.load.h	- Write .load.bin file or .load.h file to RAM_CMD - Update the write pointer REG_CMD_WRITE - Wait till EVE completes all the commands
Bin2C Converter	<*input*>.c	img_argb2.bin -> img_argb2.c	Include the C file to use the converted C-array
Disassembler	<*input*>.da.txt	bt817_dump.bin -> bt817_dump.da.txt	Open .da.txt file to see the human-readable EVE commands
Display List Optimizer	<input>.txt	WidgetDL.txt -> WidgetDL_opt.txt	Open _opt.txt file to see the optimized display list commands.

Table 1 Naming convention and usage of output files

G. Image Utilities

Within this tab, there are three sub-tab windows that offer a range of image-related functionalities. The first sub-tab is dedicated to the conversion of PNG, JPEG, and BMP images to the EVE-compatible image format. The second sub-tab is designed for the purpose of assessing the compatibility of **PNG** and **JPG** files with **CMD_LOADIMAGE**. The third sub-tab hosts a JPEG optimizer that allows users to decrease the size of input JPEG files while maintaining their EVE-compatibility.

1. Image Converter

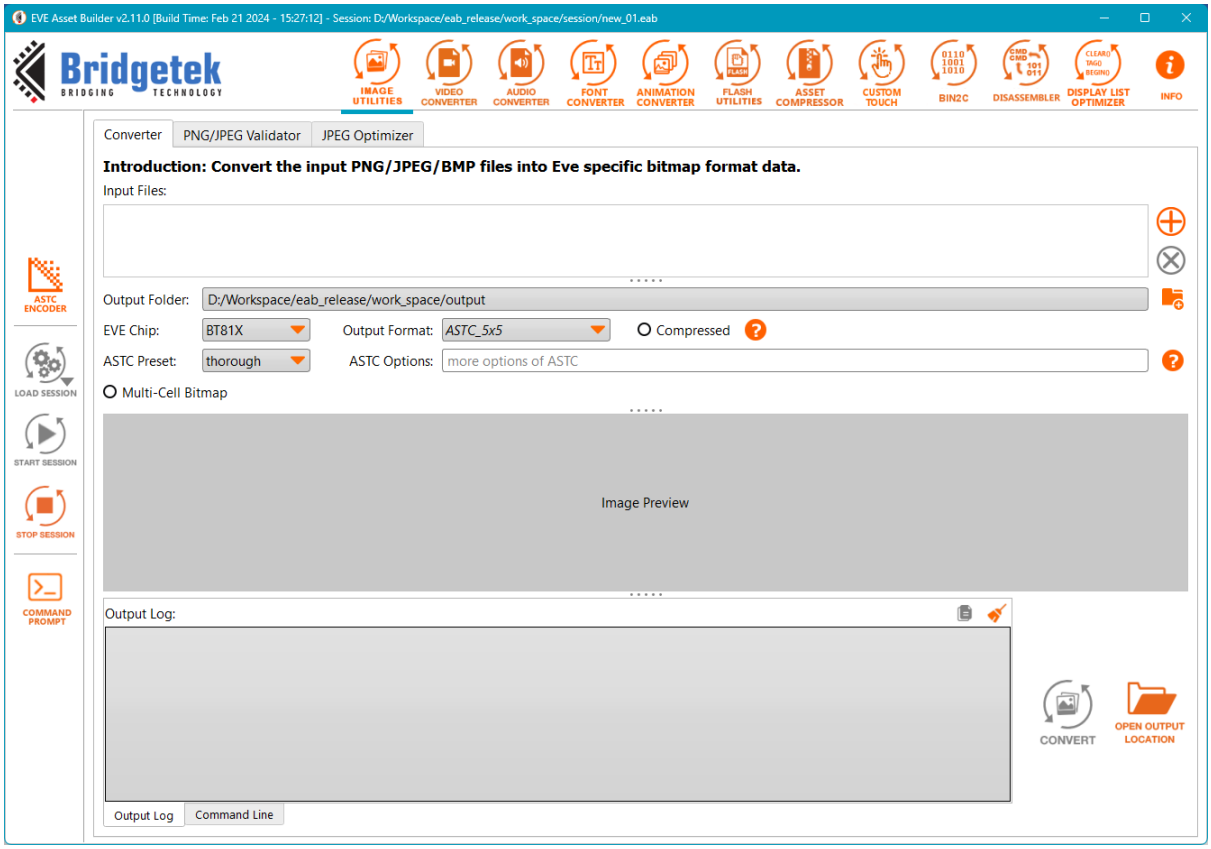


Figure 14 Image Converter tab

Input Files: PNG/JPG/BMP files to be converted. Users can change the order of images by using *Ctrl+Up* or *Ctrl+Down*.

Output Folder: The folder contains the converted files.

EVE Chip: Refer to EVE Chip Series

Compressed: Deflate the output file, which file name is *.bin other than *.raw. It should be decompressed by cmd_inflate or cmd_inflate2 before use.

Output Format: The output format of images, is one of the following:

- ARGB1555, RGB565, ARGB4
- RGB232, ARGB2,
- L1, L2, L4, L8,
- PALETTED565, PALETTED4444, PALETTED8
- ASTC formats
- DXT1_L1_RGB565, DXT1_L1_PALETTED565

- DXT1_L2_RGB565, DXT1_L2_PALETTED565

ASTC Preset: Only available for BT81X. Available options are *veryfast, fast, medium, thorough*, and *exhaustive*. Quality increases from *veryfast* to *exhaustive* while encoding speed decreases as well.

ASTC Options: Users can explore various options to determine the optimal balance for their specific needs, considering factors like memory availability, performance demands, and visual quality.

Multi-Cell Bitmap: Include padding data in ASTC-based multi-cell bitmaps to ensure that each cell is aligned to 64 bytes in memory, enabling proper rendering on the EVE chip.

2. PNG/JPEG Validator

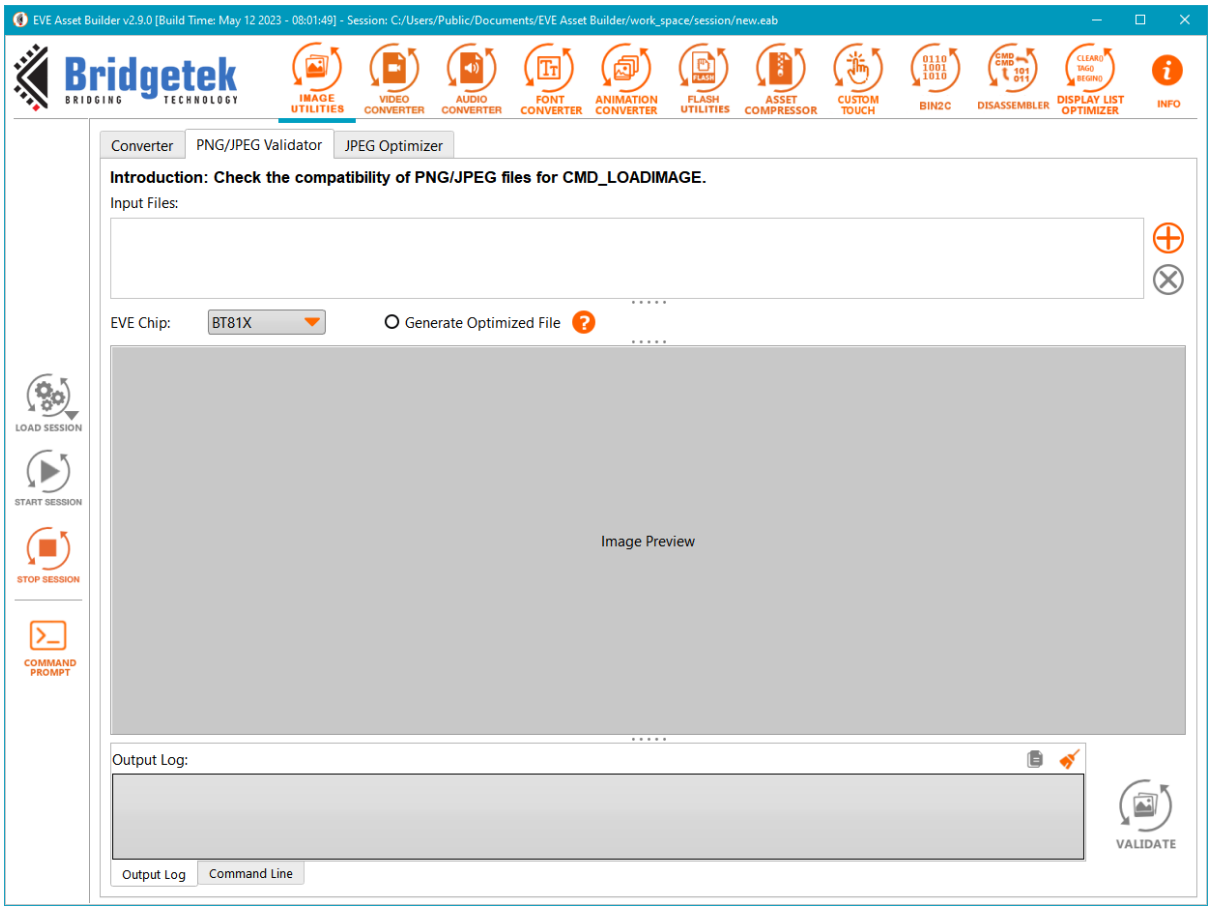


Figure 15 PNG/JPEG Validator tab

EVE Chip: Refer to EVE Chip Series

Generate Optimized File:

- If checked, input images are optimized and saved to the output folder. Then optimized images are validated.

- If unchecked, the original input images will be validated.

3. JPEG Optimizer

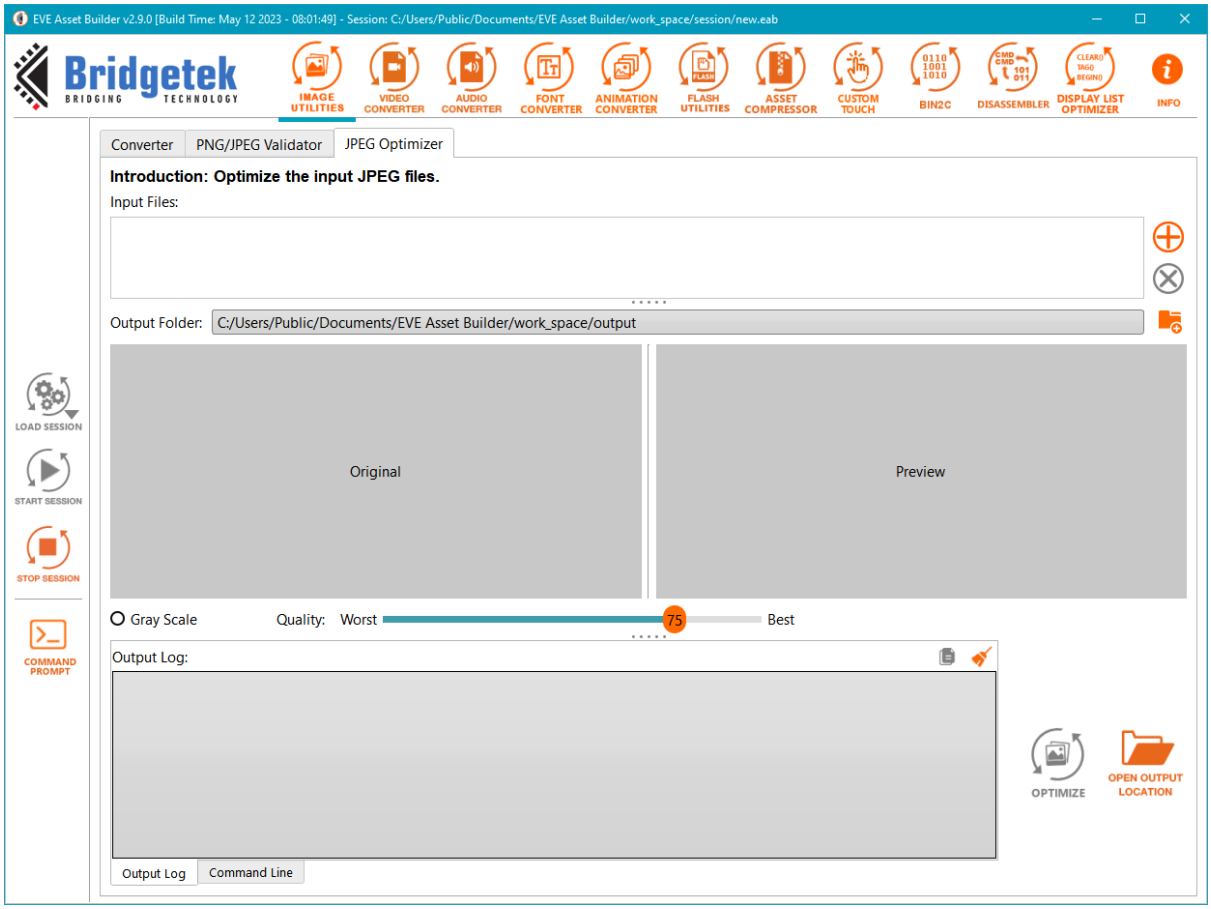


Figure 16 JPEG Optimizer tab

Gray Scale: Convert image to "L" format.

Quality: From 0-95. Reducing image size significantly also results in the worst quality.

H. Video Converter

Convert video file into **MJPEG** format and mono audio channel. Output is an AVI file and a sample project to guide how to playback video on EVE device.

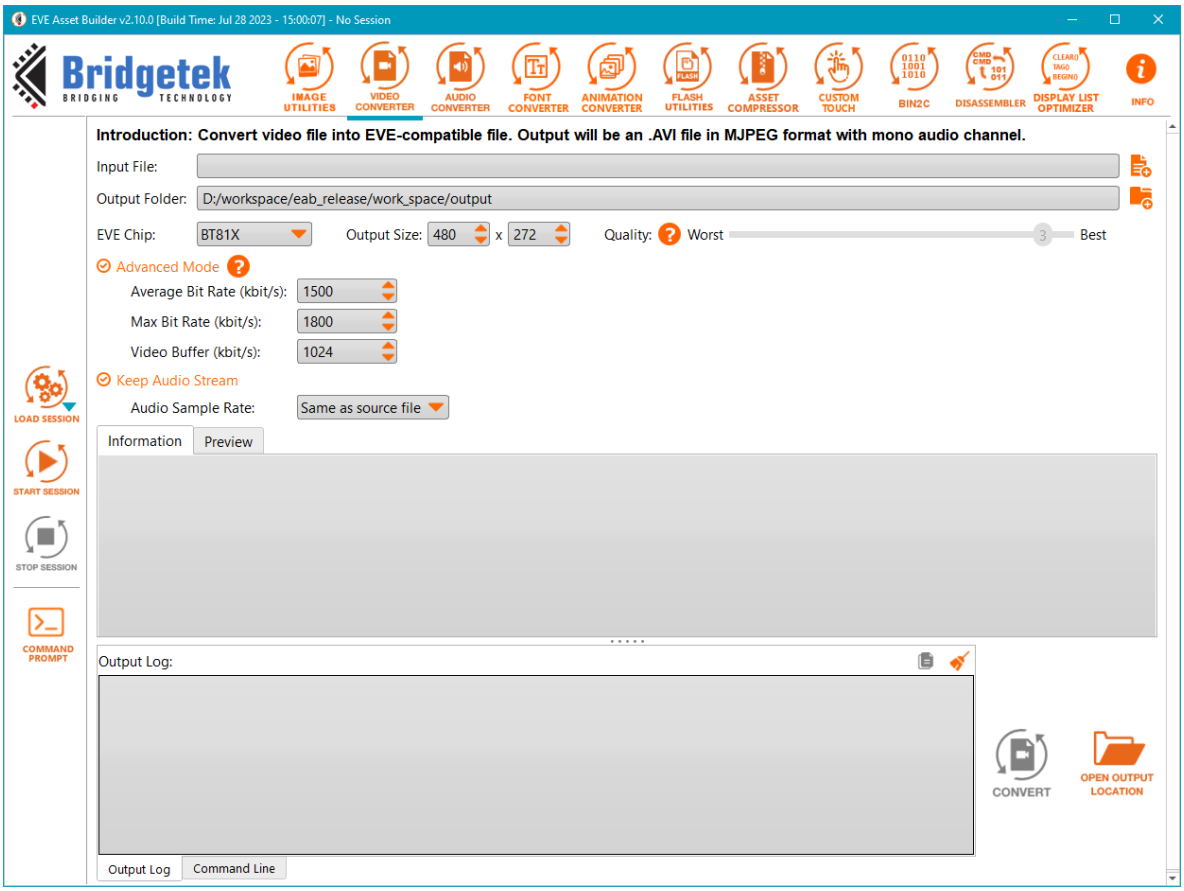


Figure 17 Video Converter tab

Input: The original video will be converted.

Output Folder: The folder contains converted files.

EVE Chip: Display a warning if the output resolution reaches the RAM_G limit. This happens when the product of width, height, and 2 (2 bytes per pixel) exceeds the capacity of RAM_G. Refer to EVE Chip Series.

Output Size: Must be specified, the video will be converted to this size.

Quality: From 1-31. 1 is best and 31 is the worst.

Advanced Mode: Allow users to control the output bitrate. Enabling this mode will disregard the Video Quality setting.

- **Average Bit Rate (kbit/s):** Set the average bit rate for the output video
- **Max Bit Rate (kbit/s):** Set max bit rate for the output video
- **Video Buffer (kbit/s):** Set buffer size for the output video

Keep Audio Stream: Users can keep or remove the audio stream

Audio Sample Rate: Choose the sample rate for the output audio stream, either keeping it the same as the input video (default) or selecting any value from the provided list. They are: 8000Hz, 11025Hz, 16000 Hz, 22050 Hz, 32000 Hz, 32780 Hz, 44056 Hz, 44100 Hz, and 48000 Hz.

I. Audio Converter

Convert WAV/MP3 file into EVE-compatible file.

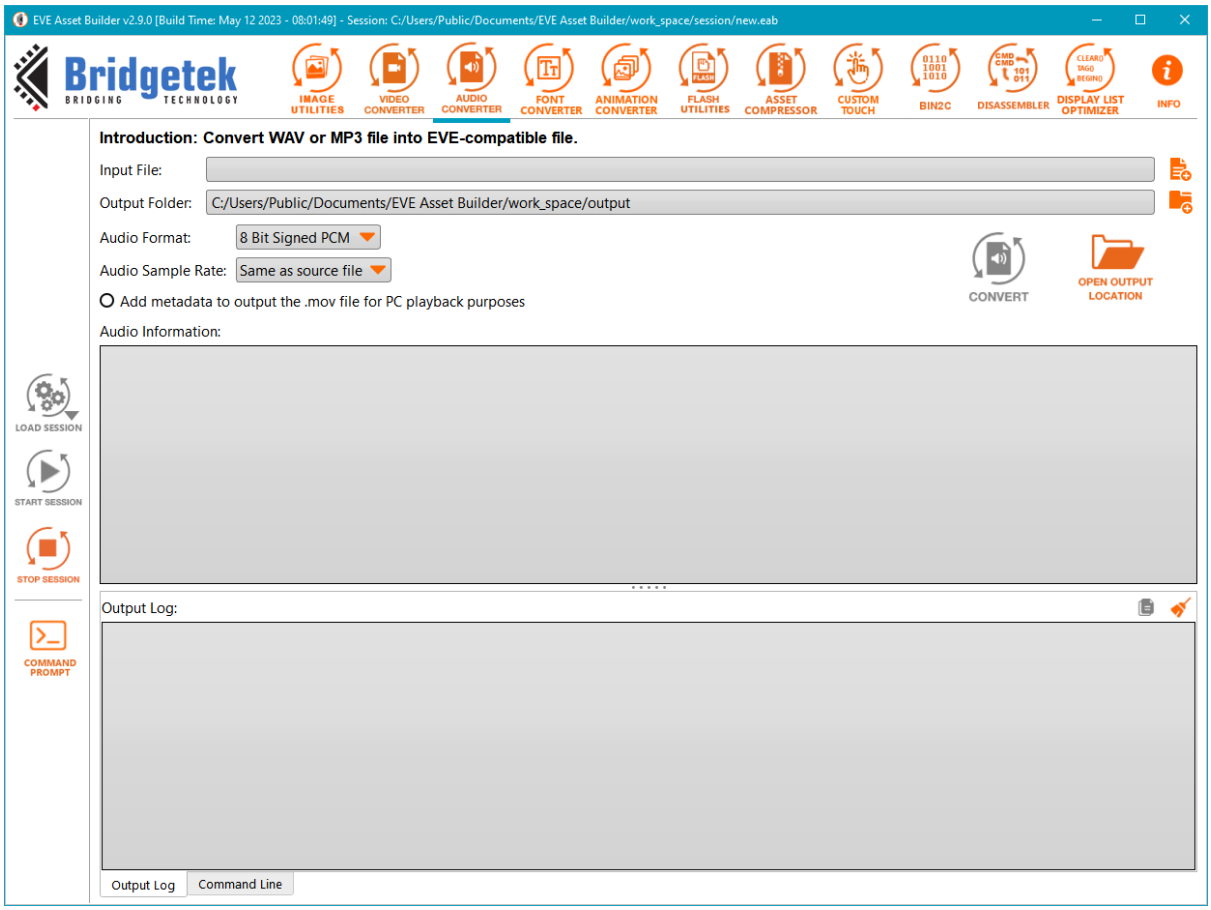


Figure 18 Audio Converter tab

Input File: The audio file will be converted.

Output folder: The folder contains the converted files.

Audio Format: Audio format of the output

- 8-bit signed PCM
- 8-bit u-Law
- 4-bit IMA ADPCM

Audio Sample Rate: Users can keep the same sample rate as the input stream or select any value from the drop-down list.

Add metadata to output the .mov file for PC playback purposes: The converted file requires a header for playback on PCs, but including the header will render it unplayable on EVE devices.

J. Font Converter

Extract characters from the OpenType or TrueType font file into a specific font metric block as well as the raw bitmap data. Unprintable characters will be filtered out.

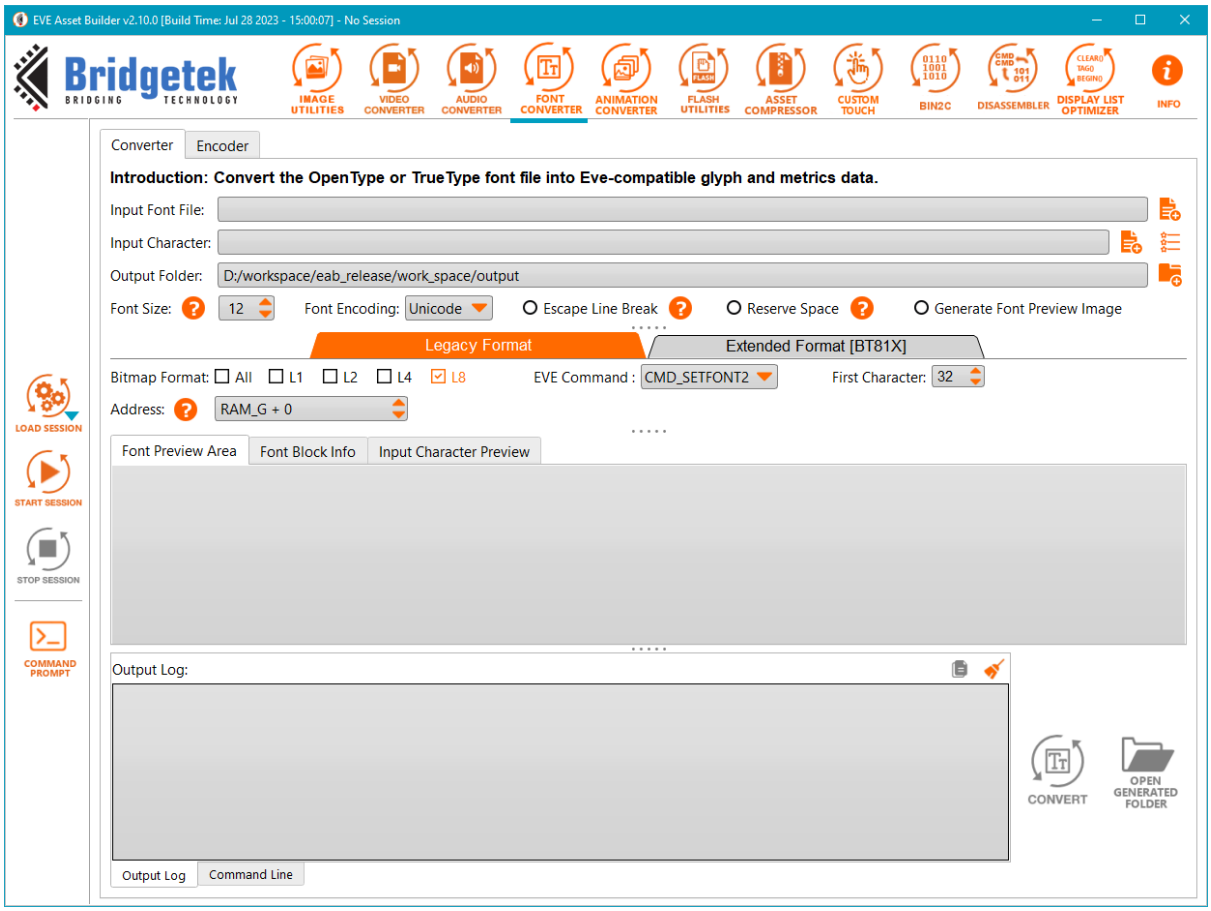


Figure 19 Font Converter tab

Input Font File: Font file to be converted.

Input Character: Choose the character set that will be converted.

Output Folder: The folder contains converted files.

Font Size: Any character whose width exceeds 255 pixels will be removed.

Font Encoding: To ensure proper conversion, the user must select an encoding that is compatible with the input font. Currently, EAB offers support for two encodings: Unicode and Symbol.

Escape Line Break: BT817/8 chip support to overwrite line break. The ASCII 0x0A is interpreted as the index, other than the line break.

Reserve Space: Allocate the code point \x20 for the space character, in order for **CMD_FILLWIDTH** to function properly.

Generate Font Preview Image: These images help users check if the characters are rendered as expected

Font Preview Area: Display sample characters by using the selected font.

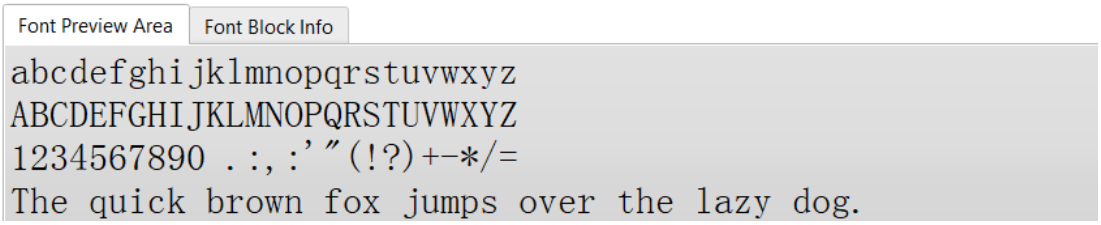


Figure 20 Font Preview Area

Font Block Info:

- Show the number of glyphs that are valid for the selected font.
- Show a statistic of the selected font, including Block Name, Unicode Range, and Valid Glyphs/All Glyphs.

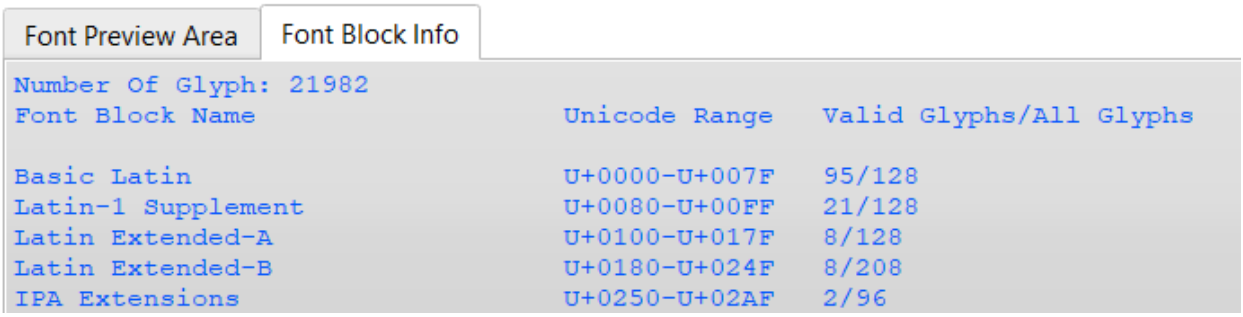


Figure 21 Font Block Info

1. Legacy Format

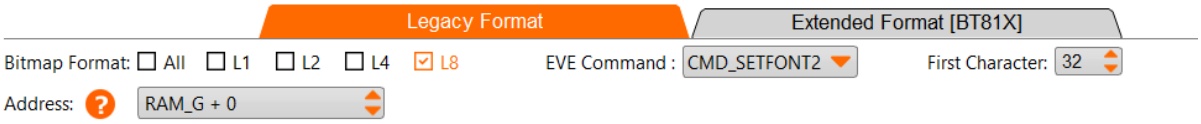


Figure 22 Legacy Format

Bitmap Format: Any combination of bitmap types L1, L2, L4, L8

EVE Command: Select **CMD_SETFONT** or **CMD_SETFONT2**. Converted files will be compatible with the chosen command.

Address: The converted font will be loaded to the offset address in RAM_G, which must be divisible by 4.

First Characters: Set the first index for converted characters. The valid range is 1-127. Apply to CMD_SETFONT2 only.

Output:

- The output consists of a 148-byte metric block along with raw bitmap data.
- In addition, this tool generates example C code to demonstrate usage.
- The output data is formatted for a single bitmap handle.

2. Extended Format [BT81X]

Handle fonts with a full range of Unicode code points. This feature only supports BT81X EVE series.

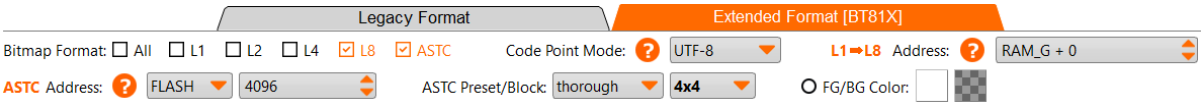


Figure 23 Extended Format

Bitmap Format: Any combination of bitmap types L1, L2, L4, L8 and ASTC

Code Point Mode:

- UTF-8: Addressing the characters with UTF-8 code point.
- Ordinal: Addressing the characters with index 0, 1, 2, …, n

Address: Address of glyph data. For ASTC, the location can be RAM_G or FLASH.
For L1->L8, the location must be in RAM_G.

Note: Due to the size of RAM_G, the maximum number of input characters will be reduced.

- **ASTC**: 8192 characters
- **L8**: 8192 characters
- **L4**: L8 * 2
- **L2**: L8 * 4
- **L1**: L8 * 8

ASTC Preset: fastest, fast, medium, thorough, and exhaustive. The encoding speed decreases while the quality increases from *fastest* to *exhaustive*.

ASTC Block: 4x4, 5x4, 5x5, 6x5, 6x6, 8x5, 8x6, 8x8, 10x5, 10x6, 10x8, 10x10, 12x10, 12x12

Note: For *auto* option, block footprint depends on font size:

- If font size > 52: block footprint = 10x8
- If font size < 19: block footprint = 8x5
- Otherwise: block footprint = 8x8

FG/BG Color: Specify the foreground/background color for the converted characters. These colors are applicable only for ASTC format. The default colors are white/transparent if not set.

Output: .glyph file contains graphic data and .xfont file contains the font metrics block, as well as the sample C code demonstrating the usage.

Limitation: For **CMD_BUTTON** and its relatives, the converted characters are not vertically aligned. This is because the pixel height calculation accommodates all input characters. Thus, it is recommended to employ a limited set of characters for **CMD_BUTTON** text to achieve superior vertical alignment. A potential solution would be to convert only the appropriate number of characters utilized in **CMD_BUTTON** to ensure that the pixel height aligns accordingly.

3. Text Encoder

The Text Encoder serves as a companion utility to the font converter. It encodes user input text using either UTF-8 or ordinal code points, as determined by the .json file generated by the font converter. This tool aids programmers in seamlessly working with EVE coprocessor commands.

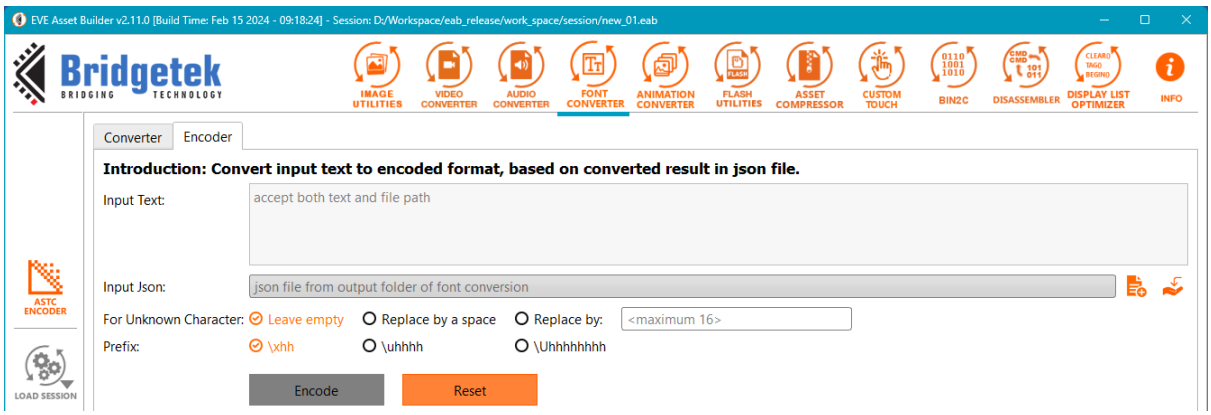


Figure 24 Encoder tab

Input Text: The character set to encode.

Input Json: The file which determines what code points the input text maps to.

For Unknown Character: Options on how to process with unknown characters.

Prefix: The prefix of the code point.

One example is as below:

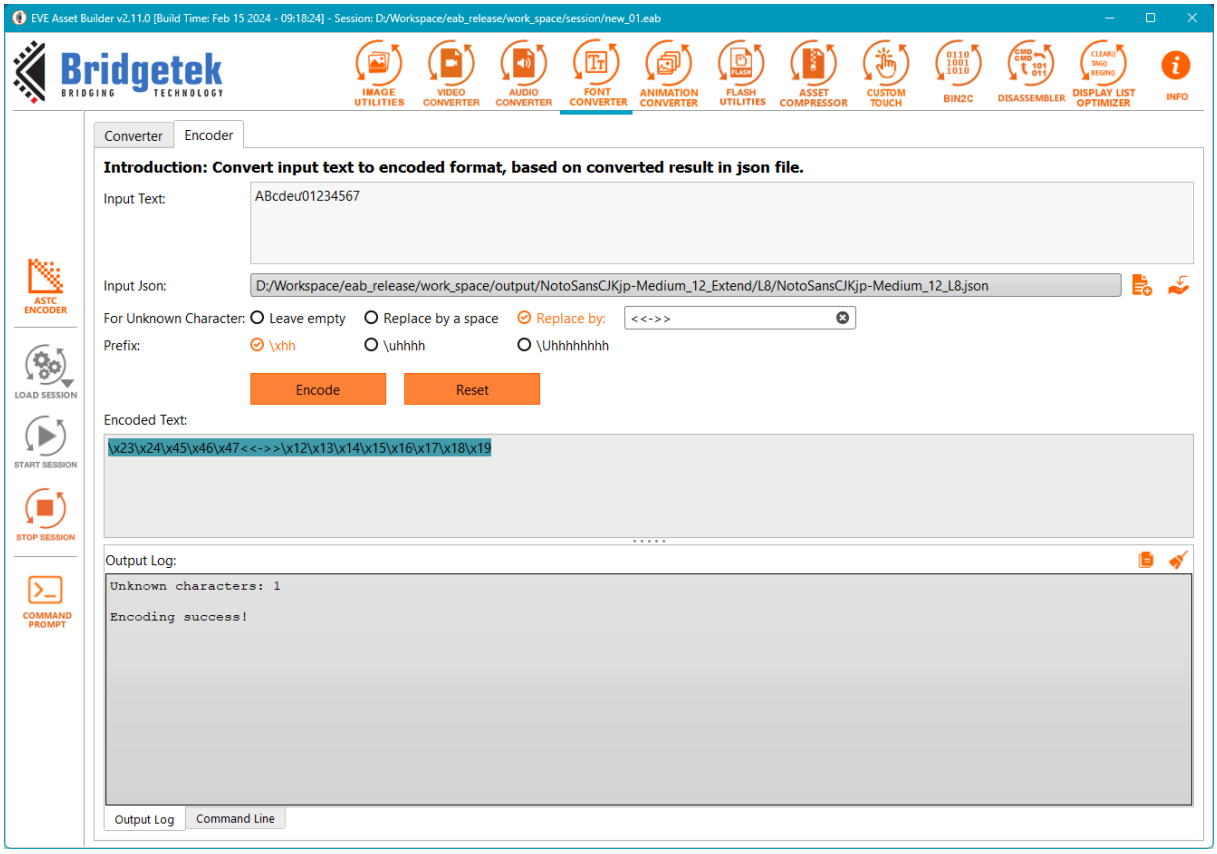


Figure 25 Text Encoder example

K. Animation Converter

Convert a **GIF** file or a series of **PNG/JPEG/BMP** files into EVE-compatible animation files. Animation is supported by the BT81X chip and above.

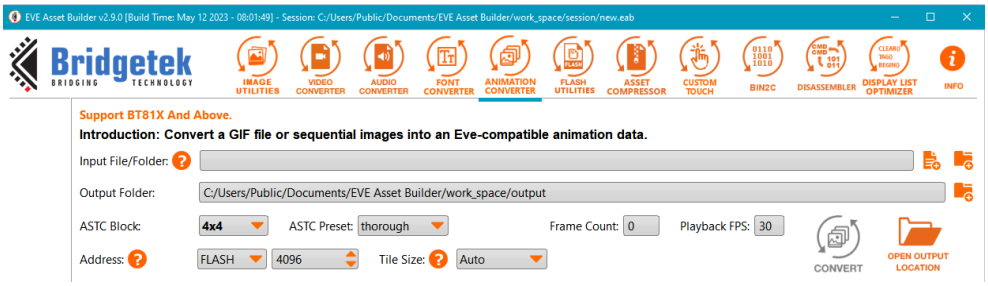


Figure 26 Animation Converter

Input File/Folder: Users can select a **GIF** file or image folder. Image folder must contain PNG/JPEG/BMP files which have the sequential name as regular expression "*.*[0-9]+\.(png|jpeg|jpg|bmp)*" defined. For example: "001.png", "002.png".

Output Folder: The folder contains the converted files.

ASTC Block: Choose one from the following options: 4x4, 5x4, 5x5, 6x5, 6x6, 8x5, 8x6, 8x8, 10x5, 10x6, 10x8, 10x10, 12x10, 12x12

ASTC Preset: fastest, fast, medium, thorough, and exhaustive. The quality increases from *fastest* to *exhaustive*, while the encoding speed decreases as well.

Frame Count: Display the number of available frames.

Playback FPS: For previewing the speed of animation

Address: Select **FLASH** or **RAM_G**. Address must be a multiple of 64.

Tile Size: Tile is the unit to determine the overlap area across frames. The valid options are: Auto, Manual or Disabled

- Auto: The optimal tile size will be automatically chosen by EAB.
- Manual: The user can manually choose both the Tile Width and Tile Height.
  Note: The product of *Tile Width* and *Tile Height* must be a multiple of 4, in compliance with the rendering constraints of ASTC images from flash.
- Disabled: Exclude the application of tiles in the conversion process.

Output

- A *SampleApp* folder illustrates how to showcase an animation.
- An *.anim.flash* file that can be directly written to flash and then used to render animations.
- With BT817/8 onwards, the utilization of **RAM_G** can be leveraged to improve the animation rendering speed. To achieve this, an "*.anim.ram_g"* file is generated.
- An *.anim.json* that contains the conversion information.

L. Flash Utilities

1. Flash Image Generator

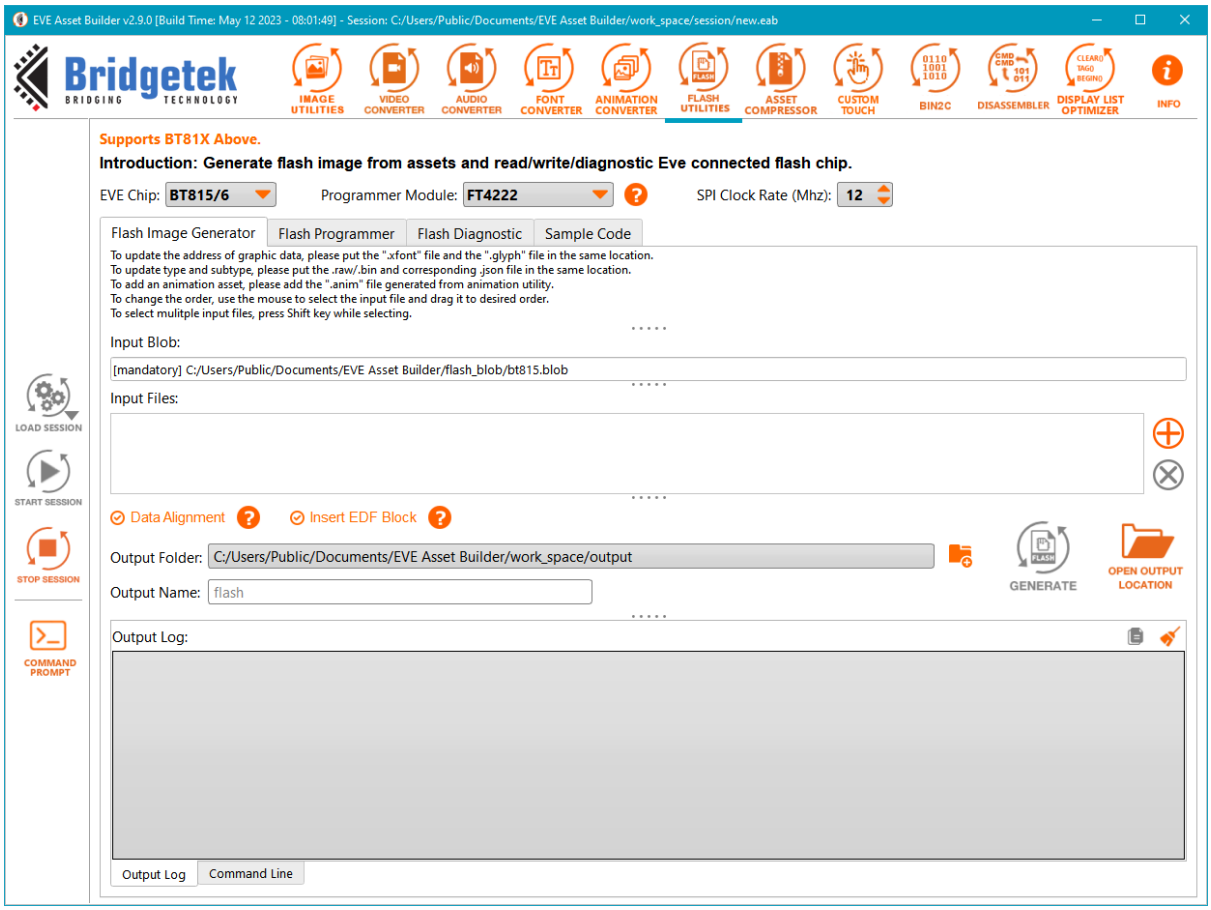


Figure 27 Generate Flash Image

Input Blob: The flash driver used by firmware, always on the first block (4096 bytes).

Input Files: Select the files of flash content.

EVE Chip: Select **BT815/6**, **BT817/8**, or **BT817A** which leads to the respective blob driver in the flash image.

Data Alignment: If unchecked, the alignment will not be applied. However, by default, each asset is aligned to 64 bytes, and the entire flash file is aligned to 256 bytes.

Insert EDF Block: EDF block will be generated and inserted after the blob driver.

Output Folder: Folder containing the generated files.

Output Name: The generated flash image file name. Default is "flash".

Output: A .bin file to write to the flash chip,
A .map file keeps the offset and size of each asset as the format:
<asset name> : <offset> : <size>,
An .edf file has the details of each assets, as described in EDF Block.

Note:

- To update the address of the graphic data in the xfont file, users should place it in the same folder as the glyph file.
- To include xfont in the flash bin, the user should place xfont after the corresponding glyph.
- To modify the type and subtype of a raw/bin file, users need to position the corresponding json file in the same folder.

EDF Block:

The "EVE Flash Description File" (EDF) contains asset data in sequential order and is located immediately after the **blob** driver. In order to obtain detailed information about the asset, a '.json' file is created with the same name as the output flash image file, and it is placed in the same directory.

EDF structure is defined as below:

```
struct EDF {
	uint32_t block_length;                  // the length of this EDF
	EVE_Flash_Asset_Info  assets[0...n-1];  // the information of n assets
};

struct EVE_Flash_Asset_Info {
	uint16_t assetID;                  // the sequential ID of the asset
	uint32_t startAddress;             // the offset of the asset
	uint32_t size;                     // the size of the asset, in byte
	uint8_t  compressionMethod;        // 0 = no compression, 1 = compressed
	uint8_t  type;                     // the type of the asset, see below
	uint16_t subType;                  // the subtype of the asset, see below
	uint16_t width;                    // the width of bitmap/video/animation
	uint16_t height;                   // the height of bitmap/video/animation
};
```

assetID denotes the order of an asset in flash image

width and ***height*** are only meaningful for bitmap, video, and animation. For other types of assets, they will be set to zero.

type and ***subType*** are defined in the following tables

type	subType	Description
0x01	-	Flash Driver (Blob)
0x02	*Table 3*	Bitmap
0x03	-	Bitmap PALETTED
0x04	-	Font Glyph (ASTC)
0x05	-	Extended Font
0x06	-	Legacy Font
0x07	*Table 4*	Animation
0x08	-	Reserved
0x09	-	PNG/JPEG File
0x0A	*Table 5*	DXT1 File
0x0B	*Table 6*	Audio File
0x0C	-	Video File
0xFC	-	Padding Data
0xFD	-	EDF Block
0xFE	-	User Data

Table 2 Asset Type

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

Table 3 subType of Bitmap

subType of Animation	Description
0xaa00	ANIM_FLASH
0xaa01	ANIM_RAMG

Table 4 subType of Animation

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

Table 5 subType of DXT1

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

Table 6 subType of Audio

2. Flash Programmer

Detect Flash

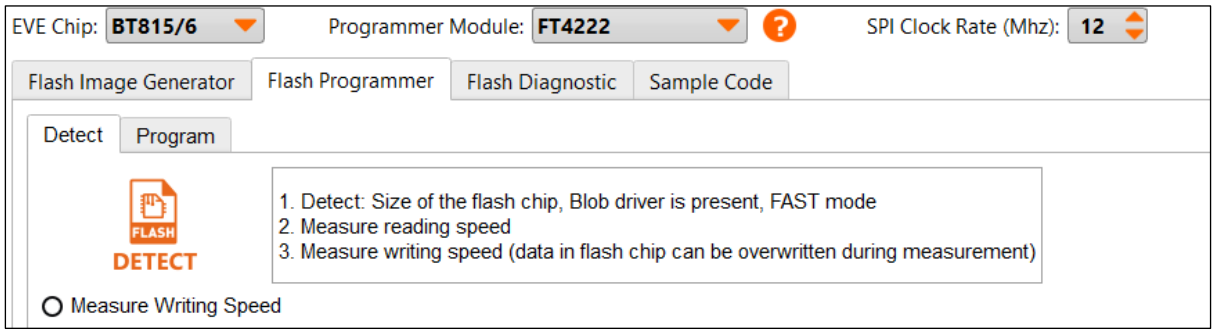


Figure 28 Detect Flash

EVE Chip: Select BT815/6 or BT817/8 based on the board.

Programmer Module: can be ***FT4222***, ***MPSSE***, or **Raspberry Pi Pico.**
Make sure the selected module is present on your device and all required drivers are installed. If your device has no module present, you may need to purchase these modules and connect them with your device properly.

Limitation: At any given time, only a single Programmer Module can be utilized.

SPI Clock Rate (Mhz): set SPI clock rate that is used to communicate from programmer module to EVE chip

Detect: Detect flash information.

Measure Writing Speed: The process will involve the writing of dummy bytes to the flash chip, which will result in the overwriting of any pre-existing data.

Program Flash

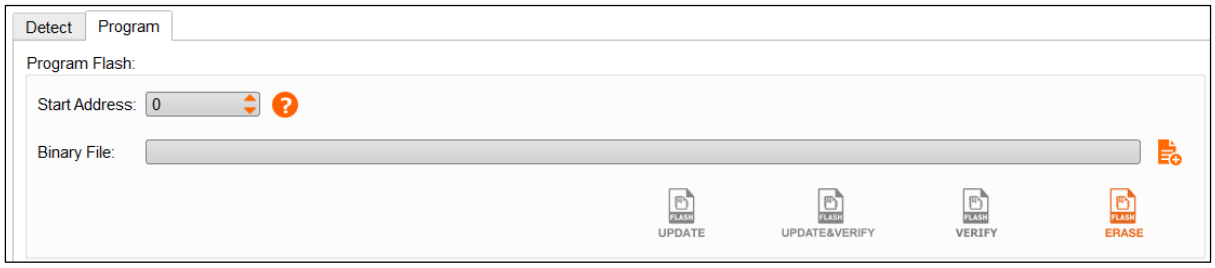


Figure 29 Program Flash

Start Address: Must be 64-byte aligned and within the range 0-256 MBytes

Binary File: Select file to write to flash chip or to verify flash content.

ERASE: Erase all data in the flash chip.

UPDATE: Write to the flash chip, erasing if necessary. Flash is not cleared out completely, only updates the required partitions.

UPDATE & VERIFY: Same as Update, then compare the updated data to source data.

VERIFY: compare the input binary file with flash content.

PROGRAM: Write into factory state flash chip. It is two or three times faster because it does not compare data before writing. This feature is supported for BT817/8 and above.

Read Flash

Read all content of flash chip into .bin file.

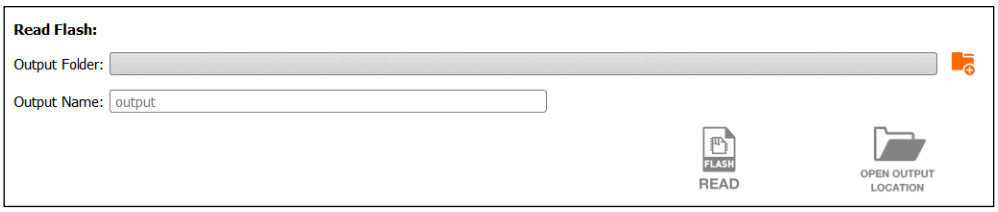


Figure 30 Read Flash

Output Folder: Select a folder to save the output file

Output Name: Select the name for the output file

READ: Start reading all data in the flash chip to the output file

Note: The whole flash chip is read out. As a result, the output file size is the same as the flash chip size.

INSTALL BLOB (for advanced users)

This feature is hidden by default. The following steps below will enable it:

1. Go to the EAB installation folder
2. Open file *ui_config.json*
3. Change the value of "install_blob" from ***false*** to ***true***, then save it
4. Reopen EAB

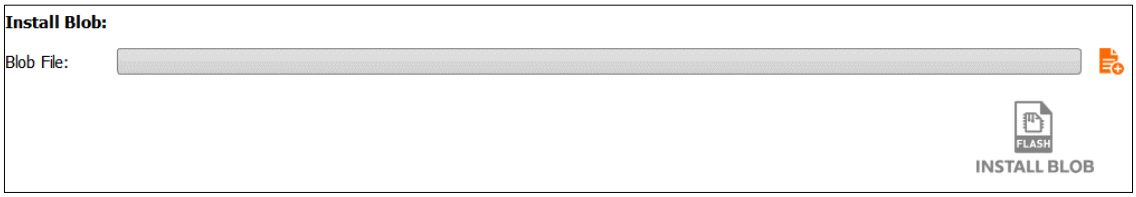


Figure 31 Enable "Install Blob"

Blob File: Select .blob file to install.

INSTALL BLOB: Start installing the blob driver to the flash chip

Programing Flash Using Raspberry Pi Pico

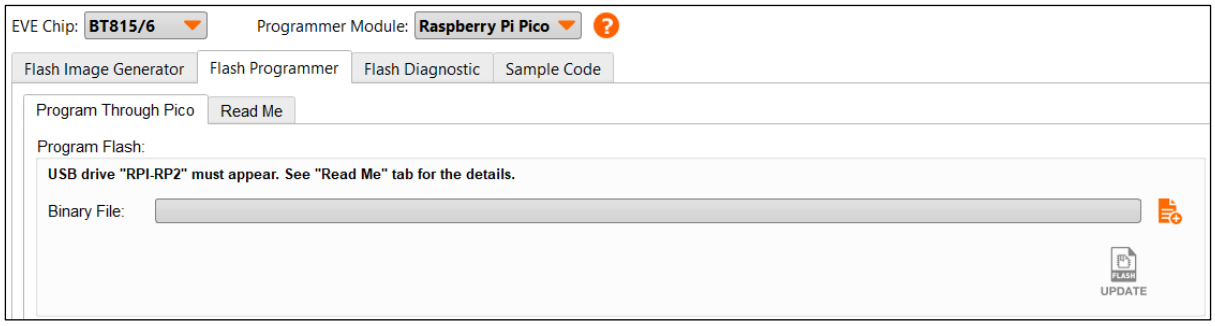


Figure 32 Programming flash using Raspberry Pi Pico

Binary File: select the file to write to the flash chip.

UPDATE: write .bin into the flash chip, erasing if necessary. Flash is not cleared out completely, only update the partition which is being updated.

Read Me Tab: show the steps to program EVE Flash through Pico.

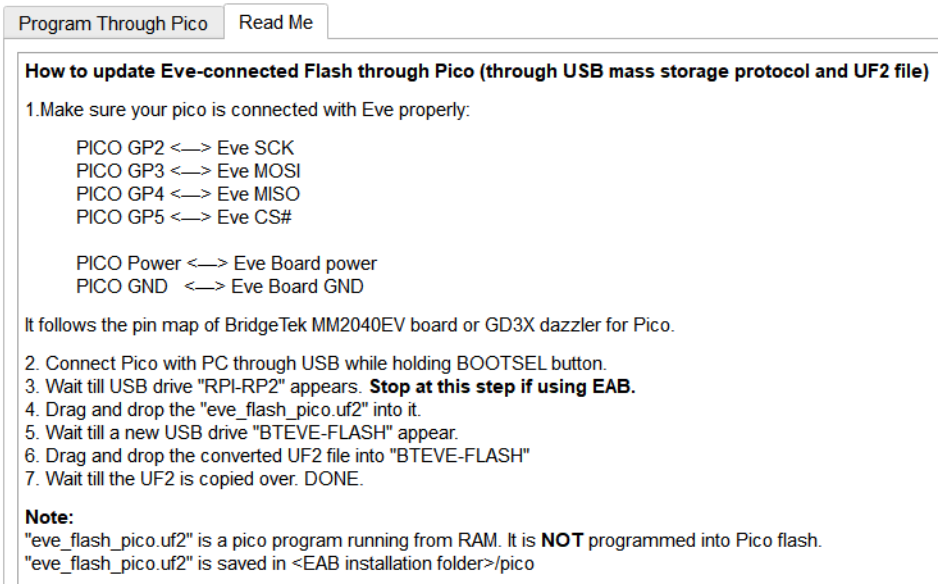


Figure 33 Raspberry Pi Pico Readme

Convert Binary File To Uf2 Format

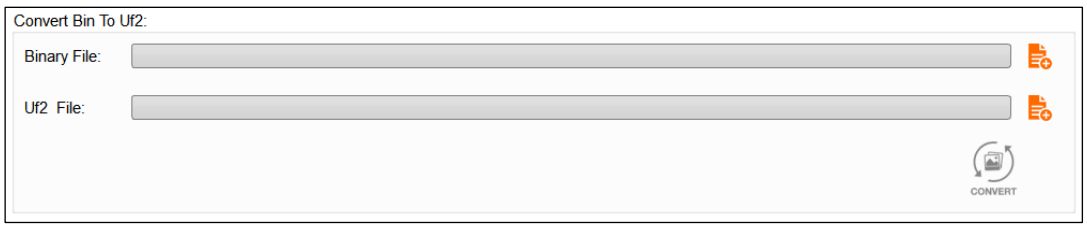


Figure 34 Convert binary file to Uf2 format

Binary File: select a binary file to convert

Uf2 File: choose a file path to save the converted output

CONVERT: start the converting process

3. Flash Diagnostic

The purpose of this feature is to troubleshoot the flash chip. The "Raspberry Pi Pico" cannot currently be used for diagnosis, so you have to use another programmer module like FT4222 or MPSSE.

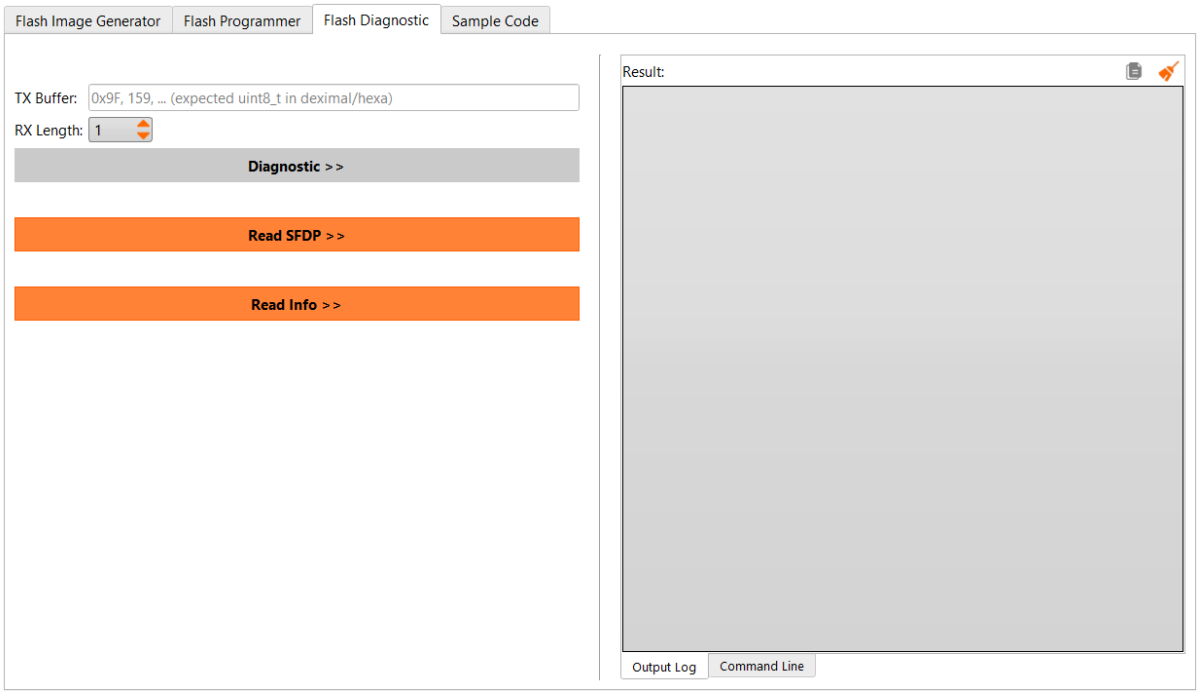


Figure 35 Flash Diagnosis

Diagnostic

TX Buffer: a buffer that will be sent to flash chip, with commas between each byte

RX Length: enter the length that you expect to receive from the flash chip

Note: for the TX and RX commands, please consult the manual of the flash chip that you are using.

Read SFDP

Read **SFDP** (Serial Flash Discoverable Parameter) structure from flash chip.

Read Info

This will report:

1. Flash status
2. Flash size (in Mbytes and Mbits)
3. DTR (double transfer rate) is supported or not

4. Sample Code

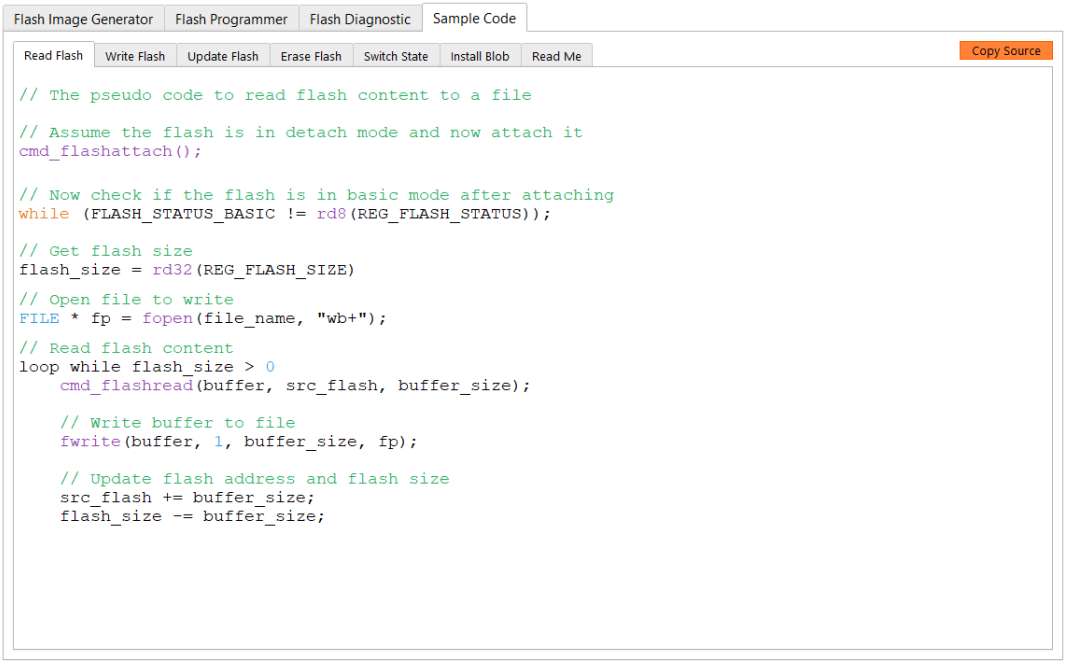


Figure 36 Flash Sample Code

Copy Source: Copy pseudo code in the current tab to the clipboard.

M. Asset Compressor

1. Compressor

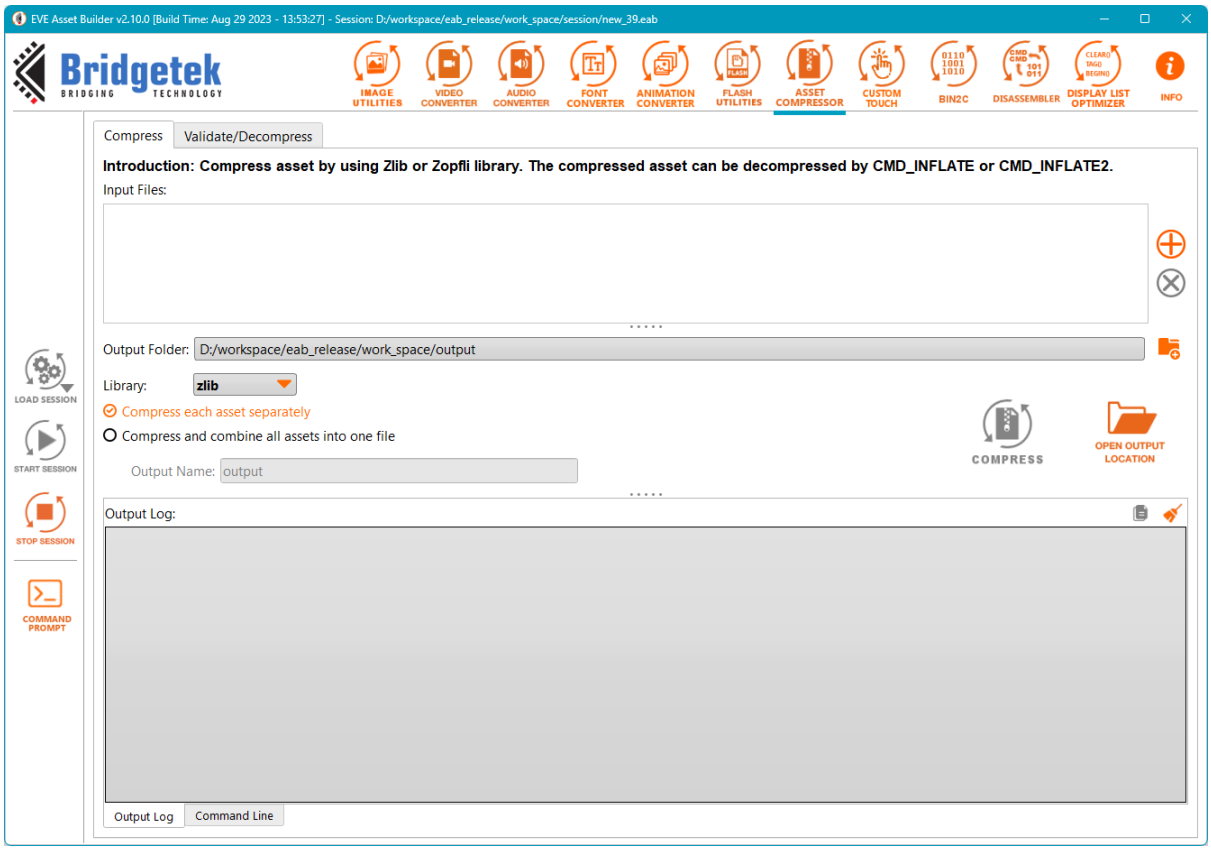


Figure 37 Asset Compressor

Add: Add asset files to compress

Remove: Remove asset file from the list

Output Folder: Folder to save compressed files

Library: Select library to compress. They are **zlib** and **zopfli.**

Compress each asset separately: each input will have one compressed output.

Compress and combine all assets into one file: all inputs will be compressed and combined into only one file.

Note: Zopfli achieves higher compression than other DEFLATE/Zlib implementations but takes much longer to perform the compression. The speed of decompressing Zopfli's output versus zlib's output is practically unaffected.

2. Validator/Decompressor

Check if the input assets are valid for **cmd_inflate** and **cmd_inflate2**. This tool supports to decompress the assets if required.

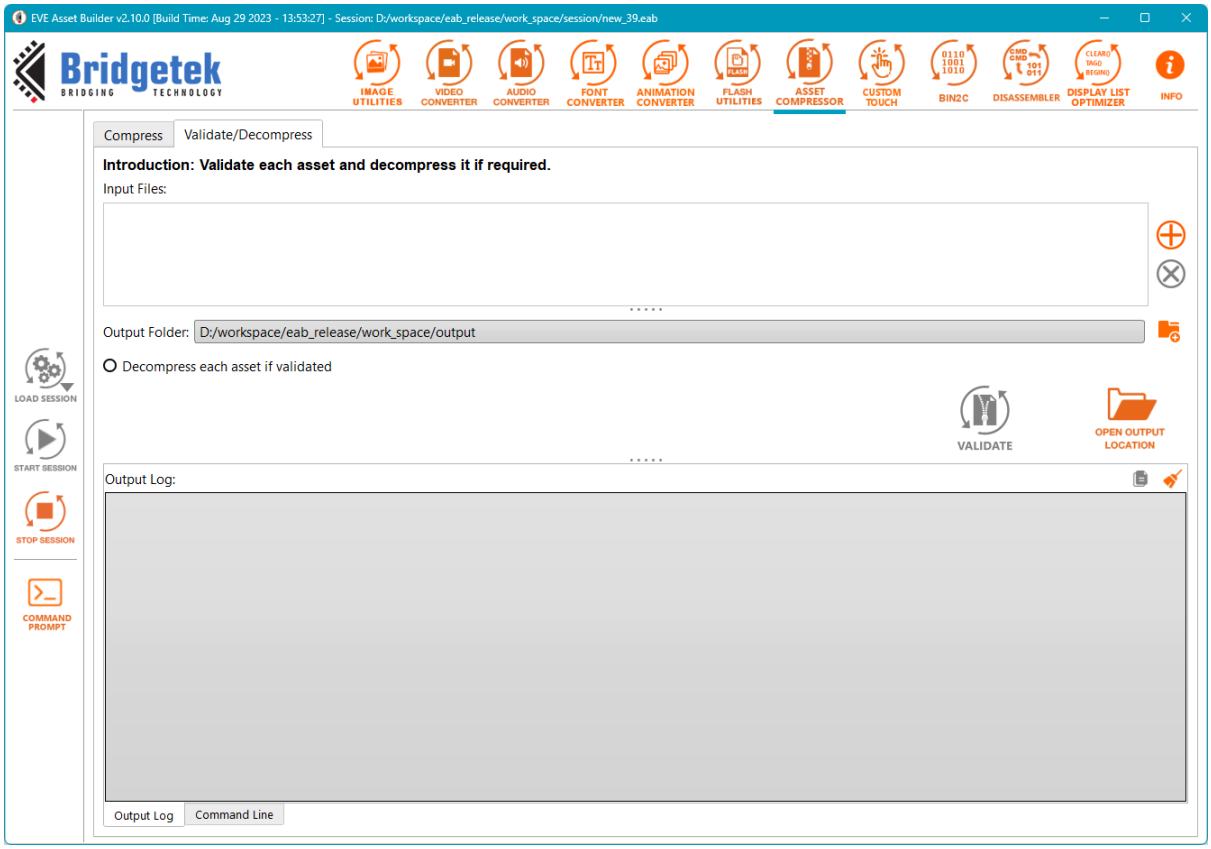


Figure 38 Compressed asset validator

Add: Add files to validate

Remove: Remove files from the list

Output Folder: Folder to save decompressed files

Decompress each asset if validated: Unpack an asset that passed validation when ticked

Validate: Verify the validity of the input files for **CMD_INFLATE** and **CMD_INFLATE2**, and proceed with decompressing them if required

N. Custom Touch Firmware Compiler

Compile a small program in a tiny C-like language and produces a loadable firmware image for touch functionality.

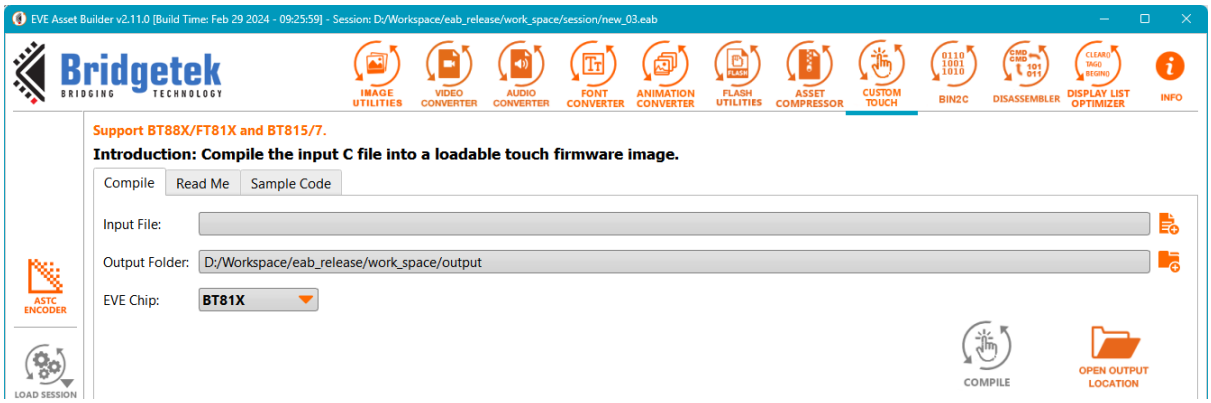


Figure 39 Custom Touch Firmware Compiler

Input File: Specify a small C-like language file for compilation

Output Folder: Indicate a directory to store output files

EVE Chip: Specify the EVE Chip series for which the touch firmware is intended

Read Me Tab: Introduction to the Custom Touch Firmware compiler

Sample Code Tab: Provide source code for some touch controller ICs

Note: There is the underlying command-line tool to compile the input file, which is called *jtc.exe*, located at *<EAB Installation folder>/tools.*

Usage: jtc.exe <input.c>

It produces two output files: <input>.load.bin and <input>.load.h. Both files serve the same purpose and functionality, with the only distinction being that .bin is in binary format, while .h is in text format.

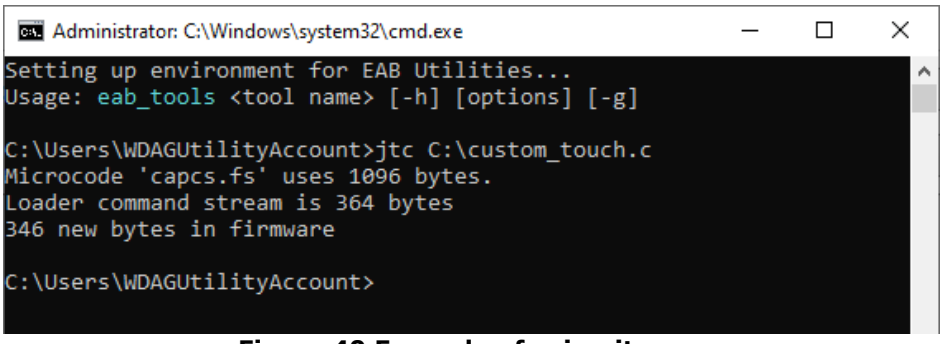


Figure 40 Example of using jtc.exe

Pseudo-code on how to use custom touch firmware

```
/***
Assume that EVE boot-up sequence is properly done before this routine.
The custom firmware is a piece of code that can be executed by the EVE
coprocessor, so the routine will do:
1) Read the firmware .load.bin into local memory or Use the C array from
the .load.h file
2) Write the firmware into RAM_CMD
3) Update the register REG_CMD_WRITE to start execution
4) Wait till the execution is done (REG_CMD_READ==REG_CMD_WRITE)
***/

// Read the custom touch firmware 'output.load.bin' into local memory
custom_touch_content = read("output.load.bin", "b");

// Write the custom touch firmware into RAM_CMD
EVE_Hal_wrMem(RAM_CMD, custom_touch_content,
sizeof(custom_touch_content));

// Update the write pointer register to start the execution
EVE_Hal_wr32(REG_CMD_WRITE, sizeof(custom_touch_content));

// Wait till EVE completes all the commands
while (EVE_Hal_rd32(REG_CMD_READ) == EVE_Hal_rd32(REG_CMD_WRITE));
```

0. Binary To C-Array Converter

Bin2C is a tool to convert a binary file into a C array.

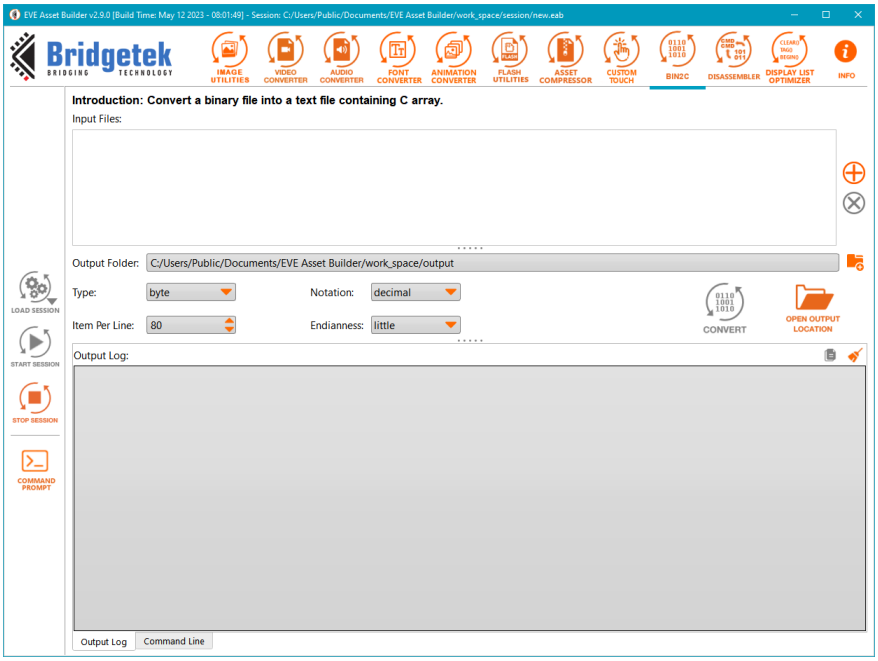


Figure 41 Bin2C Converter

Input Files: List of binary files to be converted

Output Folder: A folder that will contain C array files

Type: Data type of C array, one of byte/word/long

Notation: Set number notation in decimal/hexadecimal

Item Per Line: Number of array items in one line

Endianness: Set endian little/big when reading binary file

P. Disassembler

This tool disassembles a binary file or a text file containing a string of hexadecimal digits to human-readable command text. All display list and coprocessor commands are supported.

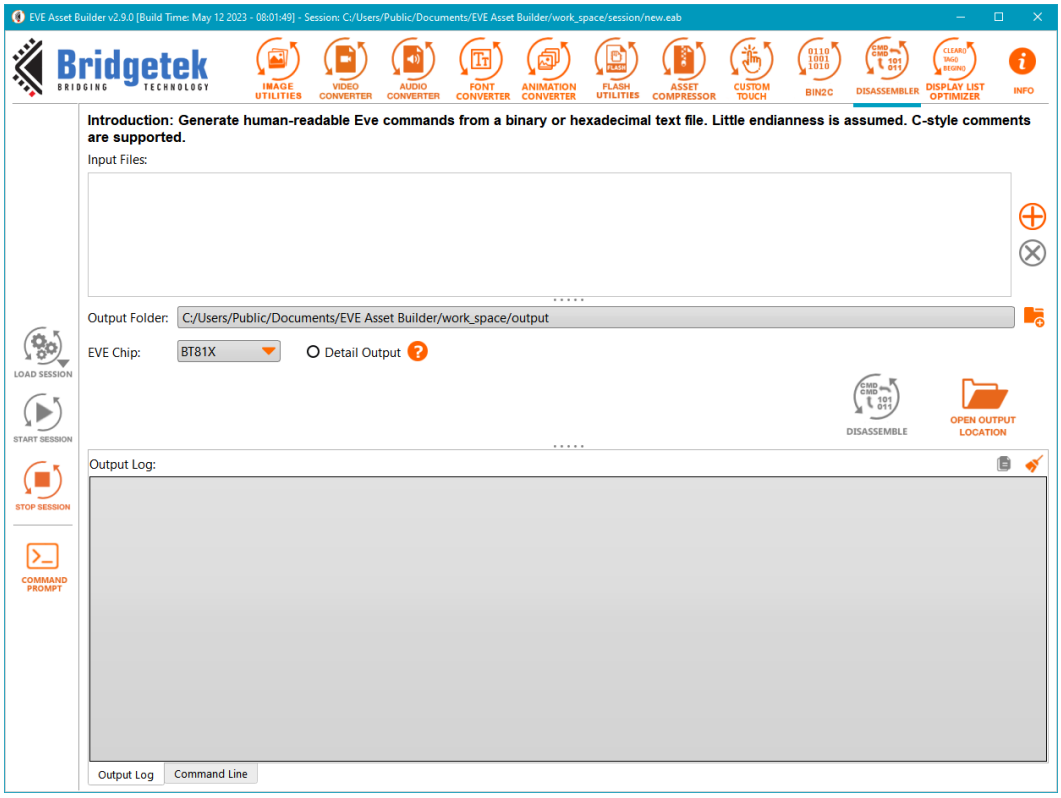


Figure 42 EVE-commands Disassembler

Input Files: List of files to be disassembled. Little-endianness is assumed. C-style comments allowed.

Output Folder: A folder to save disassembled files

EVE Chip: Refer to EVE Chip Series

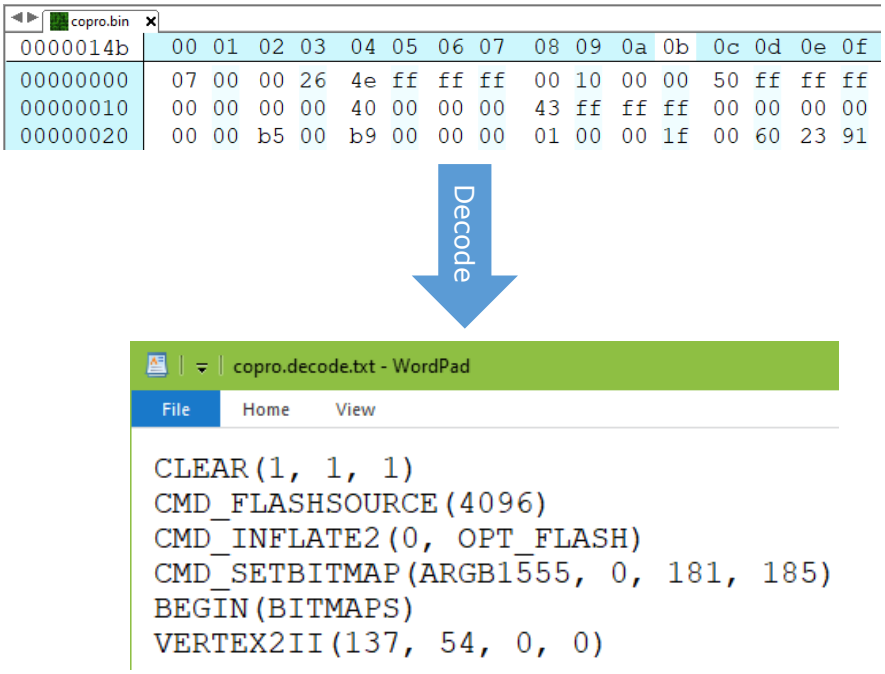
Detail Output: When checked, the output format will be the same as screen-shot below.

Address	Value (Hex)	Instruction	Parameter	Interpretation Of Parameter
0	0x26000007	CLEAR	0x0000 0007	CLEAR(1, 1, 1)
4	0x22000000	SAVE_CONTEXT	-	SAVE_CONTEXT()
8	0x27000002	VERTEX_FORMAT	0x0000 0002	VERTEX_FORMAT(2)
12	0x0500001C	BITMAP_HANDLE	0x0000 001C	BITMAP_HANDLE(28)
16	0x1F000001	BEGIN	0x0000 0001	BEGIN(BITMAPS)
20	0x95C87E54	VERTEX2II	0x15C8 7E54	VERTEX2II(174, 135, 28, 84)
24	0x97887E65	VERTEX2II	0x1788 7E65	VERTEX2II(188, 135, 28, 101)
28	0x98E87E78	VERTEX2II	0x18E8 7E78	VERTEX2II(199, 135, 28, 120)
32	0x9A487E74	VERTEX2II	0x1A48 7E74	VERTEX2II(210, 135, 28, 116)
36	0x23000000	RESTORE_CONTEXT	-	RESTORE_CONTEXT()
40	0x00000000	DISPLAY	-	DISPLAY()

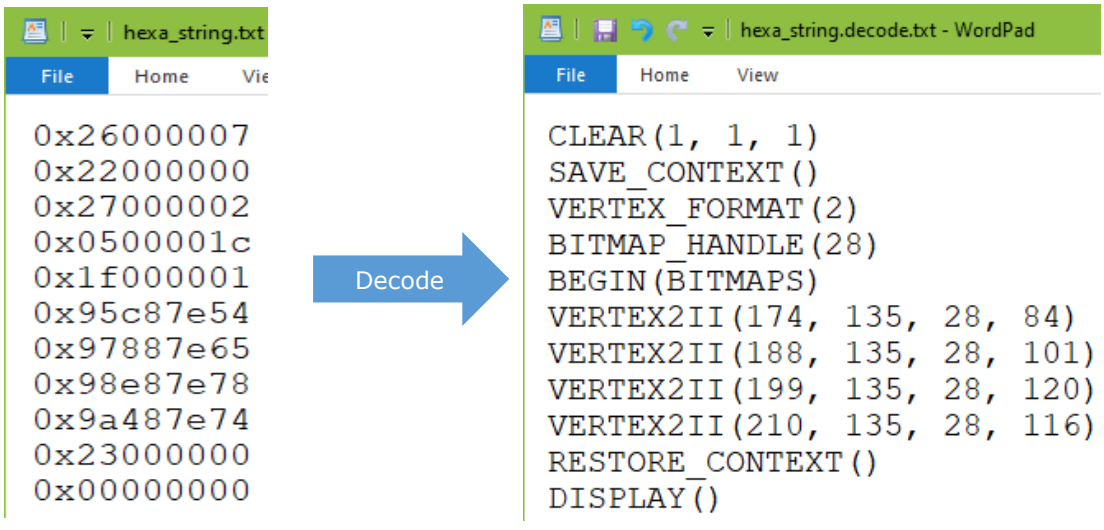
Figure 43 Detail output of Disassembler tool

Sample inputs

- **Binary:**



- **Hexadecimal string:**



Q. Display List Optimizer

This tool helps to optimize display list commands.

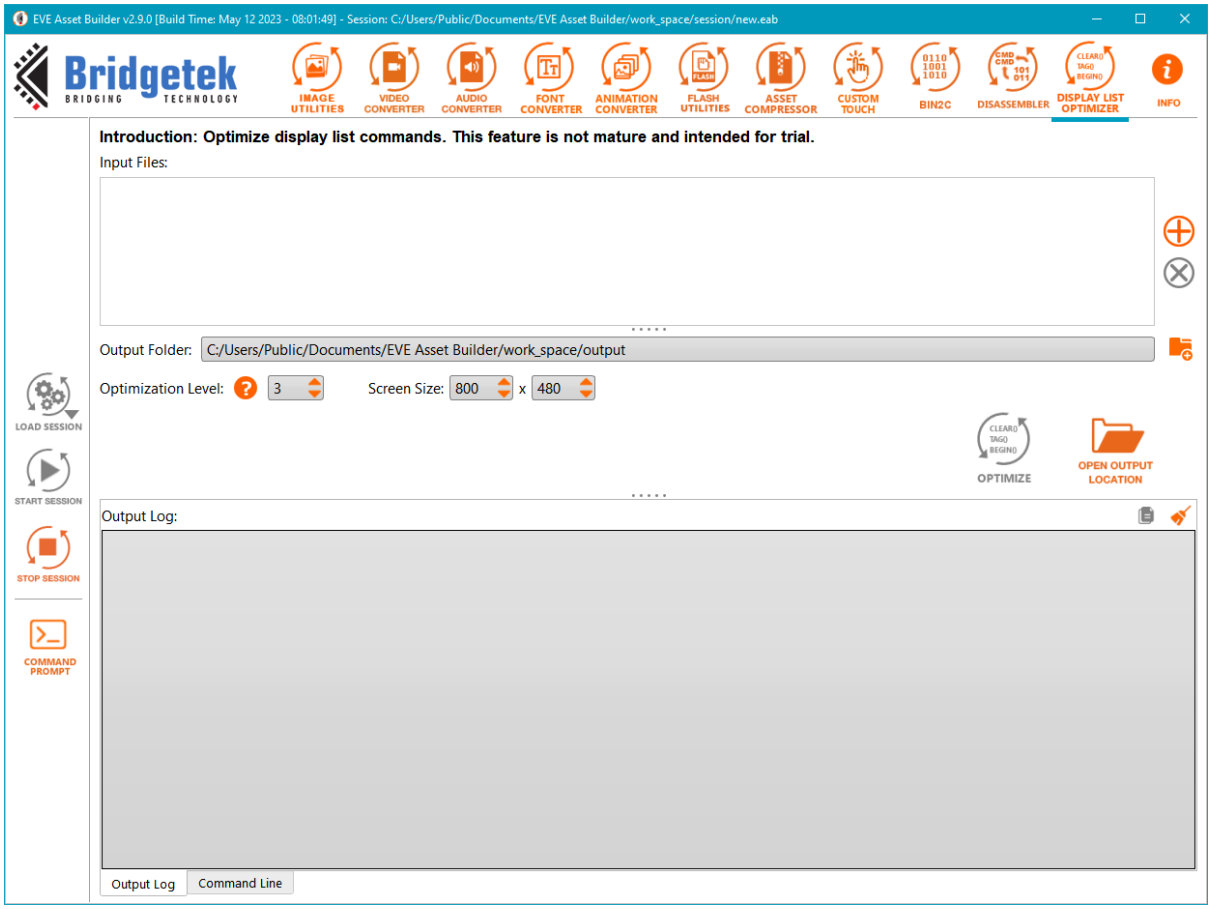


Figure 44 Display List Optimizer

Input Files: List of files to be optimized.

Output Folder: A folder to save optimized files

Optimization Level: Range from 0-3

0 - No optimize
1 - Remove VERTEX_FORMAT, VERTEX_TRANSLATE_X/Y, END
2 - LEVEL_1 + remove the off-screen drawing primitive
3 - LEVEL_2 + remove hidden objects partially/totally

Screen Size: Set screen resolution so that the tool knows which drawing object is off-screen

Optimize: Start optimizing each input file

R. SampleApp

After converting EVE resources, EAB will generate a **SampleApp** project, which is located at the output folder. It contains sample code that demonstrates the usage of converted resources. The **SampleApp** project is a MSVC project which is prepared for the Bridgetek modules VM816CU50A or ME817EV which have built-in FT4222H.

Example: Convert image **lenaface.png** into ARGB1555 format.

```
void Load_Image_lenaface_32x32_ARGB1555(EVE_HalContext * phost)
{

    Gpu_Hal_WaitCmdfifo_empty(phost);

    Gpu_CoCmd_Dlstart(phost);
    App_WrCoCmd_Buffer(phost, CLEAR(1, 1, 1));
    App_WrCoCmd_Buffer(phost, COLOR_RGB(255, 255, 255));

    uint16_t iw = 32;
    uint16_t ih = 32;
    uint16_t format = ARGB1555;

#ifdef USE_BT81X_FLASH
#define BITMAP_ADDRESS_ON_FLASH <address> // address of bitmap file from Flash Map
#define BITMAP_SIZE_ON_FLASH    <size>    // size of bitmap file from Flash Map
    /* Switch Flash to FULL Mode */
    Gpu_CoCmd_FlashHelper_SwitchFullMode(phost);

    if (format >= compressedED_RGBA_ASTC_4x4_KHR && format <= compressedED_RGBA_ASTC_12x12_KHR) {
        Gpu_CoCmd_SetBitmap(phost, (0x800000 | BITMAP_ADDRESS_ON_FLASH / 32), format, iw, ih);
    } else {
        Gpu_CoCmd_FlashRead(phost, RAM_G, BITMAP_ADDRESS_ON_FLASH, BITMAP_SIZE_ON_FLASH);
        Gpu_CoCmd_SetBitmap(phost, RAM_G, format, iw, ih);
    }
#else
    //load bitmap file into graphics RAM
    //RAM_G is starting address in graphics RAM, for example 00 0000h
    Gpu_Hal_LoadImageToMemory(phost, "../../../lenaface_32x32_ARGB1555.raw", RAM_G, LOAD);
    Gpu_CoCmd_SetBitmap(phost, RAM_G, format, iw, ih);
#endif
    //Start drawing bitmap
    App_WrCoCmd_Buffer(phost, BEGIN(BITMAPS));
    App_WrCoCmd_Buffer(phost, VERTEX2II(0, 0, 0, 0));
    App_WrCoCmd_Buffer(phost, END());
    App_WrCoCmd_Buffer(phost, RESTORE_CONTEXT());
    App_WrCoCmd_Buffer(phost, DISPLAY());
    Gpu_CoCmd_Swap(phost);
    App_Flush_Co_Buffer(phost);
    Gpu_Hal_WaitCmdfifo_empty(phost);

}
```

Figure 45 Sample code: load ARGB1555 image

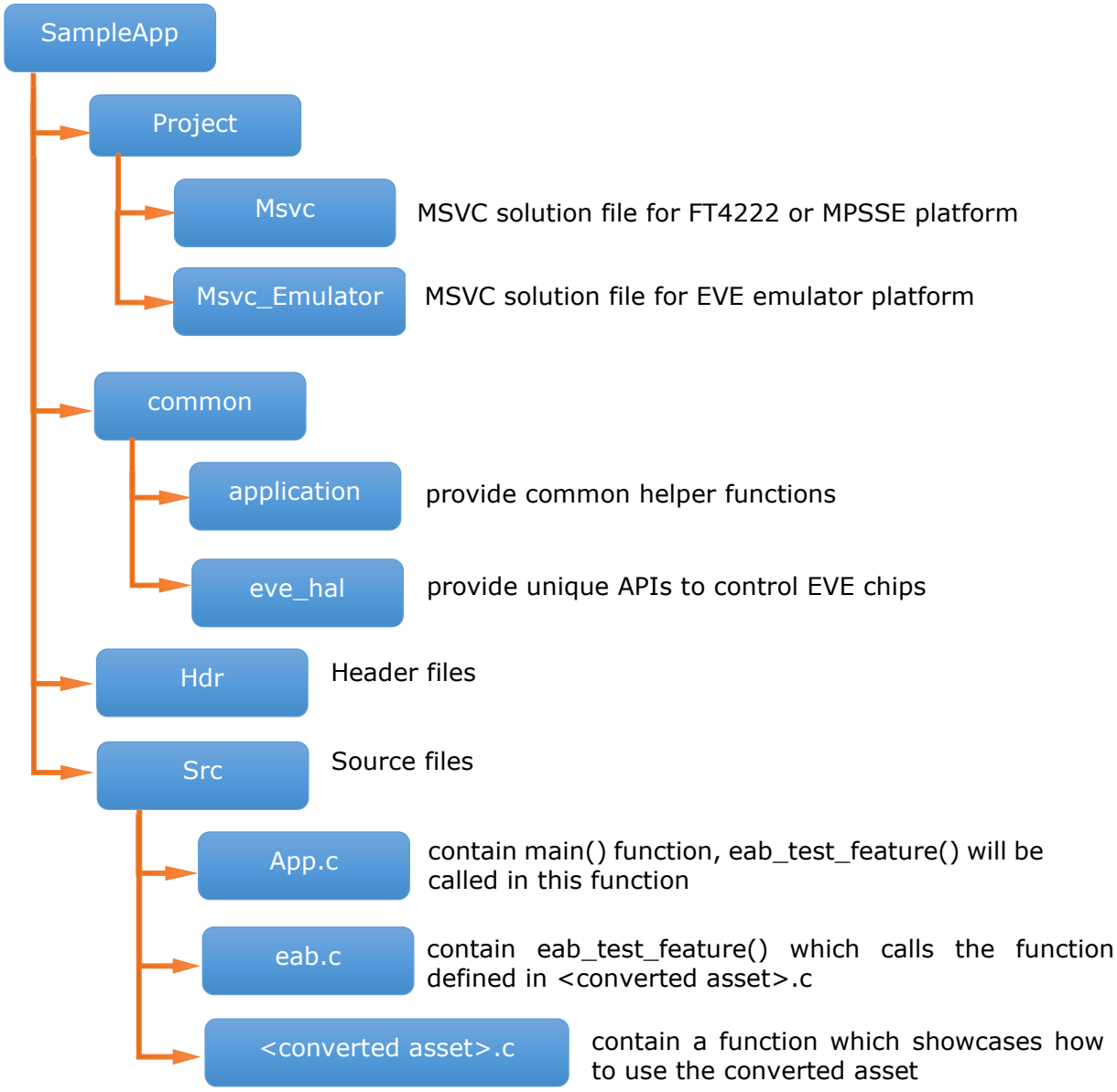
The main() function will call eab_test_feature(). Then eab_test_feature() calls the dedicated Load_Image_lenaface_32x32_ARGB1555().

```
void eab_test_feature(EVE_HalContext * phost)
{
    Load_Image_lenaface_32x32_ARGB1555(phost);
    /* INSERT GENERATED FUNCTION */
}

int32_t main(int32_t argc, char8_t *argv[])
{
    phost = &ph;
    Gpu_Init(phost);
    EVE_Util_clearScreen(phost);

    eab_test_feature(phost);

    system("pause");
    Gpu_Release(phost);
    return 0;
}
```

Figure 46 SampleApp: main() function

Usually, most of the parameters are well-prepared in SampleApp, users just need to open the solution file at *SampleApp/Project/MSVC/ SampleApp_MSVC.sln*, then build and run in the connected EVE board. The converted resource should be rendered in the hardware LCD.

SampleApp folder structure



V. Example

A. Image Converter

Below is an example of converting a **PNG** file into an ASTC format 4x4 file and showcasing how the output file is to be utilized.

Step 1: Convert the input **PNG** file into ASTC

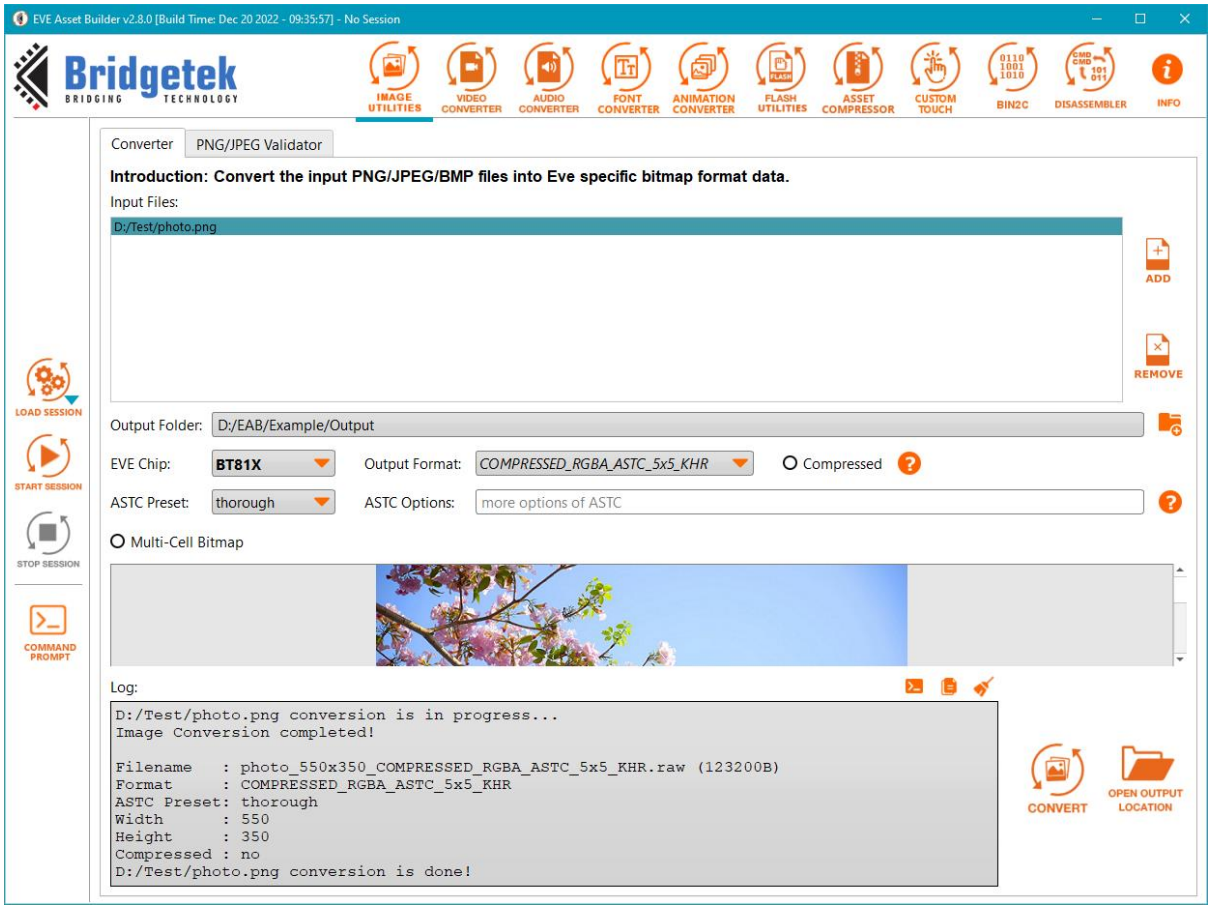


Figure 47 Image Converter example on EAB

Output folder:

Name	Ext	Size	Date
[..]		<DIR>	25/05/2020 11:11
photo_800x480_COMPRESSED_RGBA_A..	c	1,455	25/05/2020 11:10
photo_800x480_COMPRESSED_RGBA_A..	json	234	25/05/2020 11:10
photo_800x480_COMPRESSED_RGBA_A..	raw	245,760	25/05/2020 11:10
photo_800x480_COMPRESSED_RGBA_A..	rawh	878,293	25/05/2020 11:10
photo_800x480_COMPRESSED_RGBA_A..	png	389,331	25/05/2020 11:10

Figure 48 Image Converter example: Output folder

The generated photo_800x480_COMPRESSED_RGBA_ASTC_5x5_KHR.c contains a function Load_Image() to load the generated .raw file:

Step 2: Call that function inside the SampleApp MSVC project, images will be displayed on LCD:

Figure 49 Image Converter example: LCD screen

B. Video Converter

Below is an example to convert an AVI file and shows how the output file is to be utilized.

Step 1: Convert video file into EVE-compatible file by EAB

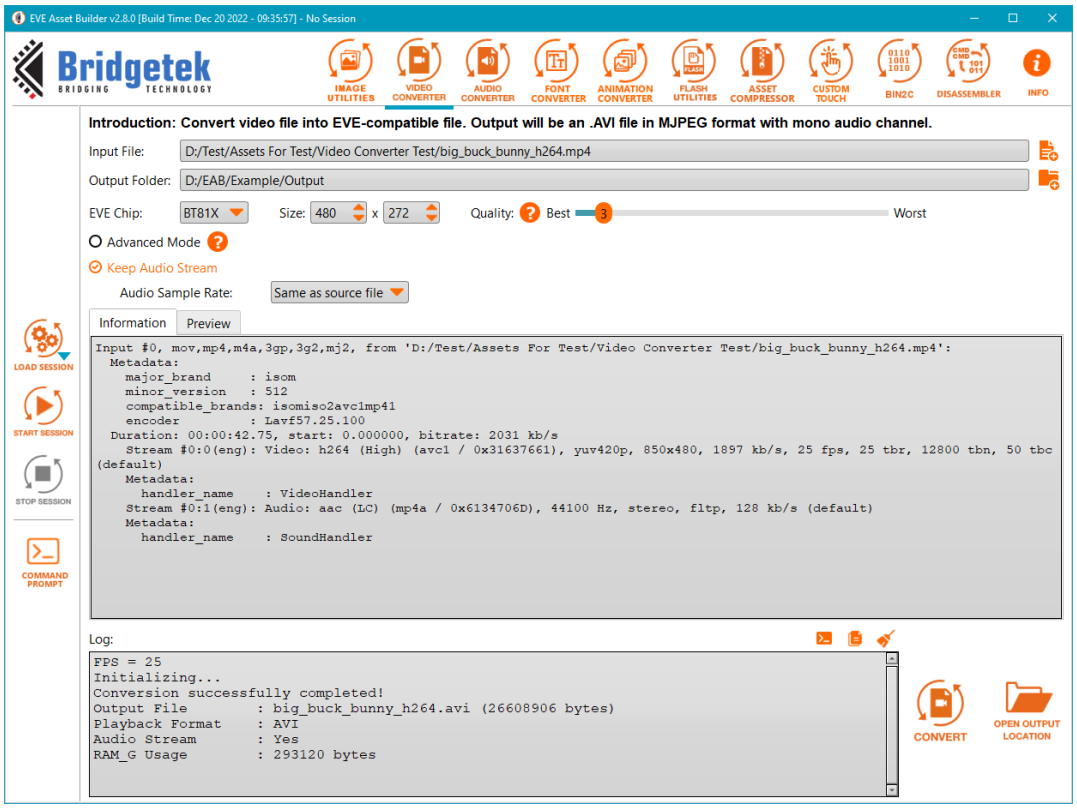


Figure 50 Video Converter example

Output folder:

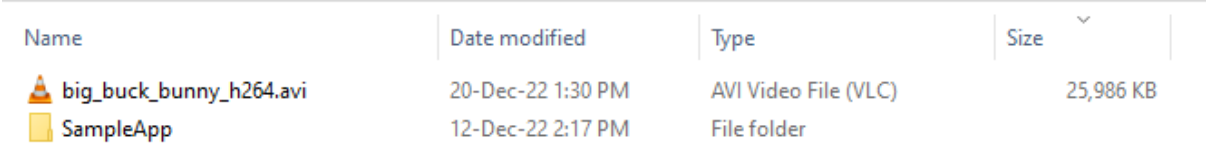


Figure 51 Video Converter example: Output folder

The generated video playback.c contains a C function to load the generated .raw file:

```
void VideoPlayback()
{
#if USE_BT81X_FLASH
    /* Switch Flash to FULL Mode */
    Gpu_CoCmd_FlashHelper_SwitchFullMode(&host);
    Gpu_Hal_Wr32(phost, REG_PLAY_CONTROL, 1);
    Gpu_CoCmd_Dlstart(phost);
    App_WrCoCmd_Buffer(phost, CLEAR(1, 1, 1));
    App_WrCoCmd_Buffer(phost, COLOR_RGB(255, 255, 255));

    Gpu_CoCmd_FlashSource(phost, ADDR_VIDEO);
    App_WrCoCmd_Buffer(phost, CMD_PLAYVIDEO);
    App_WrCoCmd_Buffer(phost, OPT_FLASH | OPT_SOUND | OPT_NOTEAR);
    App_Flush_Co_Buffer(phost);
    Gpu_Hal_WaitCmdfifo_empty(phost);
```

Figure 52 Video Converter example: sample code

Step 2: Call function VideoPlayback from sample app MSVC_Emulator project, video will be played back.

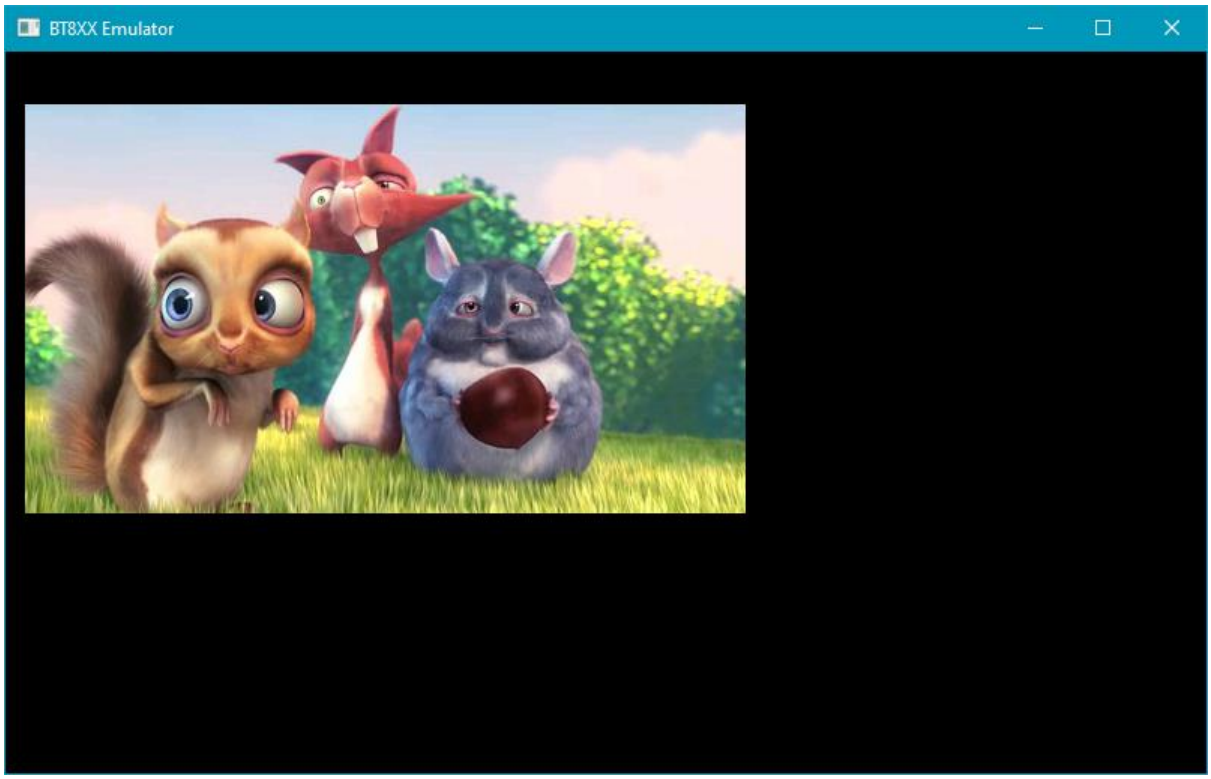


Figure 53 Video Converter example: play back

C. Audio Converter

Below is an example of covert a WAV file and how the output file is to be utilized.

Step 1: Convert the audio file into an EVE-compatible file

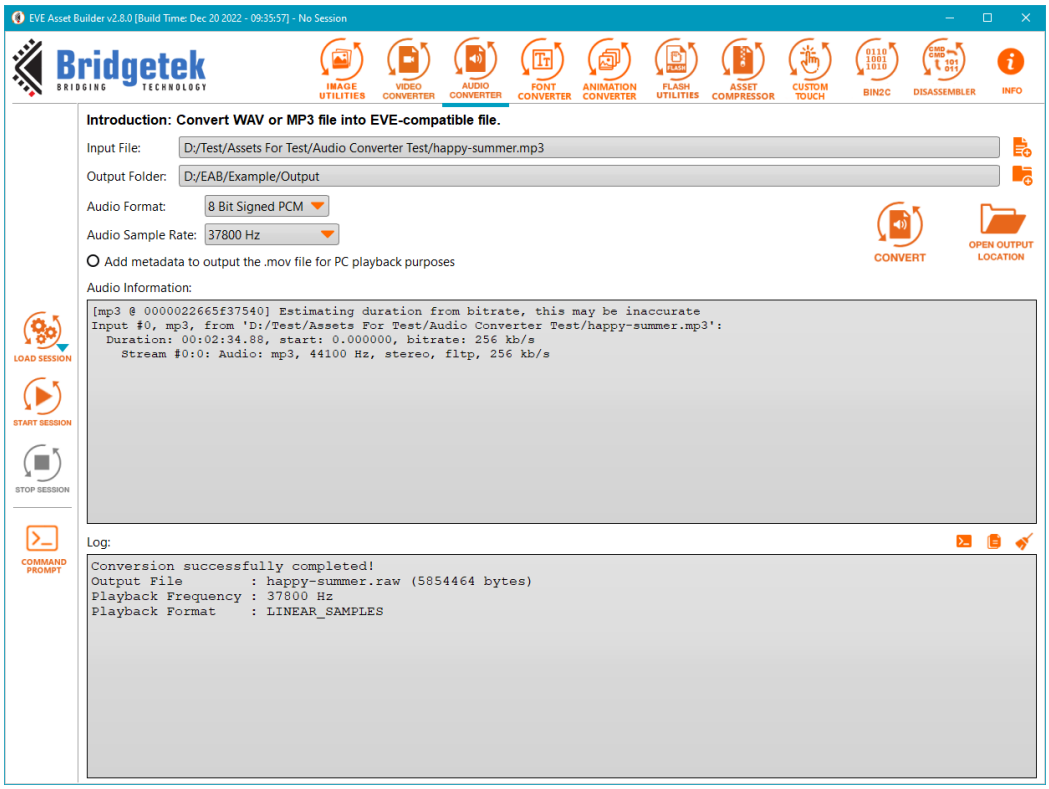


Figure 54 Audio Converter example on EAB

Output folder:

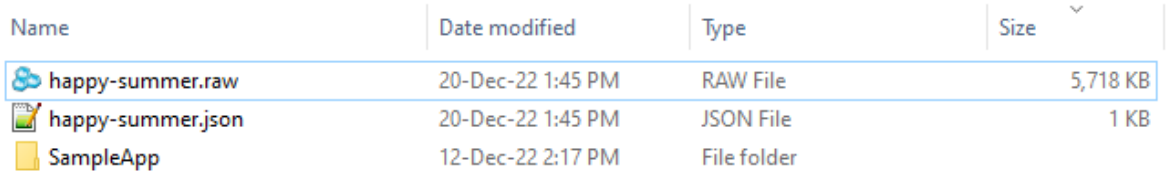


Figure 55 Audio Converter example: Output folder

The generated SampleApp contains a C function to load the generated .raw file:

```
void Audio_Playback(EVE_HalContext *host)
{
    phost = host;
    Fifo_Init(&MediaFifo, MEDIA_FIFO_ADDR, MEDIA_FIFO_BUFFER, REG_MEDIAFIFO_READ, REG_MEDIAFIFO_WRITE);
    Gpu_CoCmd_MediaFifo(phost, MEDIA_FIFO_ADDR, MEDIA_FIFO_BUFFER);
    printf("Mediafifo: Start address and length %d %d\n", MEDIA_FIFO_ADDR, MEDIA_FIFO_BUFFER);
    App_Flush_Co_Buffer(phost);
    Gpu_Hal_WaitCmdfifo_empty(phost);
    MediaFile.pfile = fopen("../../../happy-summer.raw", "rb");
    if (MediaFile.pfile == NULL) {
        return;
    }

    fseek(MediaFile.pfile, 0, SEEK_END);
    MediaFile.FILE_SIZE = MediaFile.file_len = ftell(MediaFile.pfile);
    MediaFile.played_len = 0;
```

Figure 56 Audio Converter example: Sample code

Step 2: Call function Audio_Playback () from SampleApp MSVC project, audio will play LCD.

D. Font Converter

Below is an example of converting a TrueType Font File(.ttf) and how the output file is utilized.

Step 1: Convert font file into EVE-compatible file

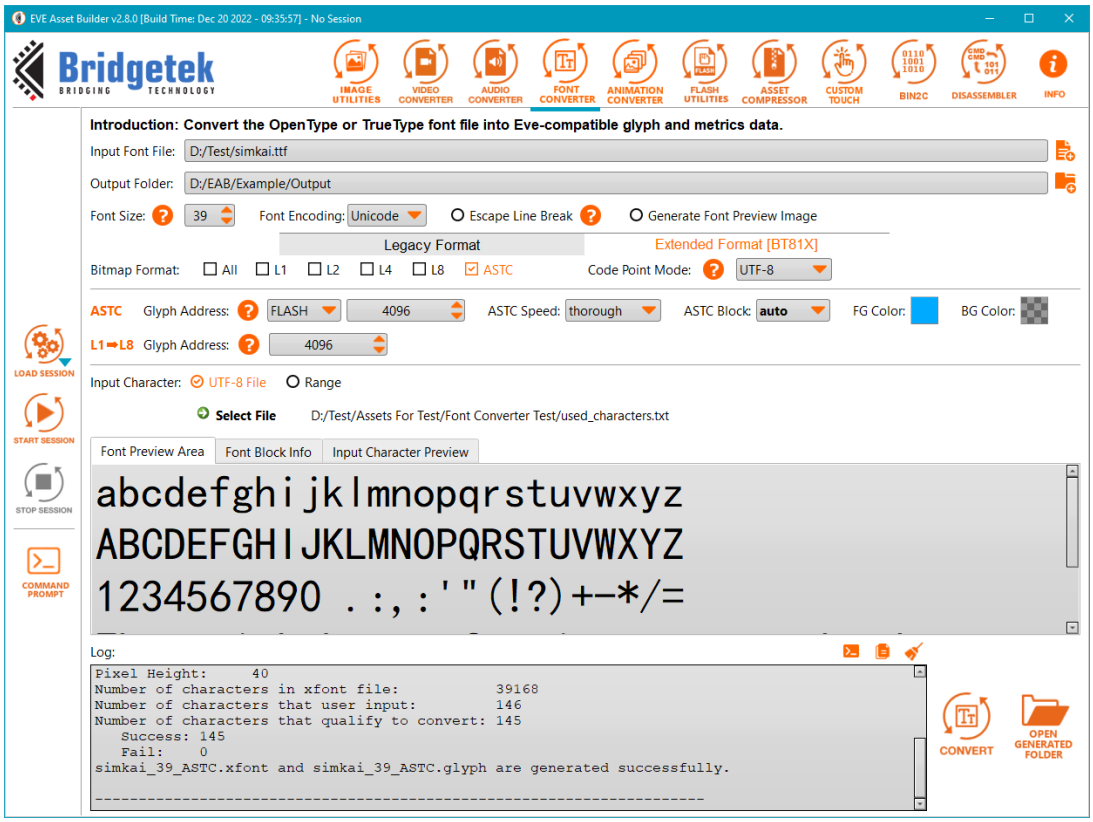


Figure 57 Font Converter example

Output folder:

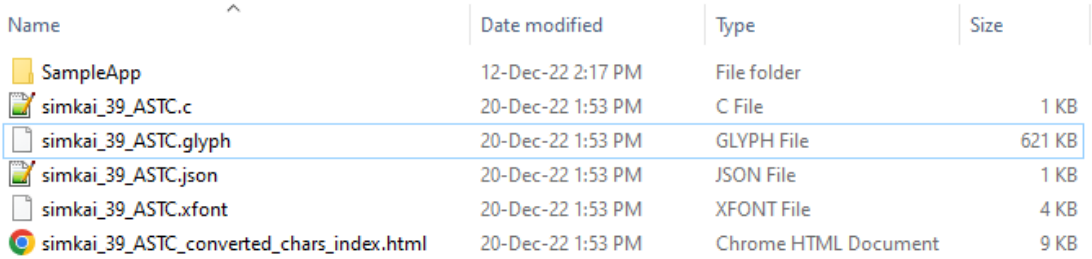


Figure 58 Font Converter example: Output folder

The generated simkai_39_ASTC.c contains a C function to load the generated .raw file:

```
void LoadXfont()
{
    uint32_t fontAddr = RAM_G;

    // NOTE: Remember to write glyph file into BT815's flash at address 4096
    // Switch Flash to FULL Mode
    Gpu_CoCmd_FlashHelper_SwitchFullMode(phost);

    // Load xfont file into graphics RAM
    Gpu_Hal_LoadImageToMemory(phost, "path\\to\\simkai_39_ASTC.xfont", fontAddr, LOAD);
    Gpu_Hal_WaitCmdfifo_empty(phost);
```

Figure 59 Font Converter example: sample code

Step 2: Call function LoadXfont() from SampleApp MSVC project, text will be displayed on LCD:

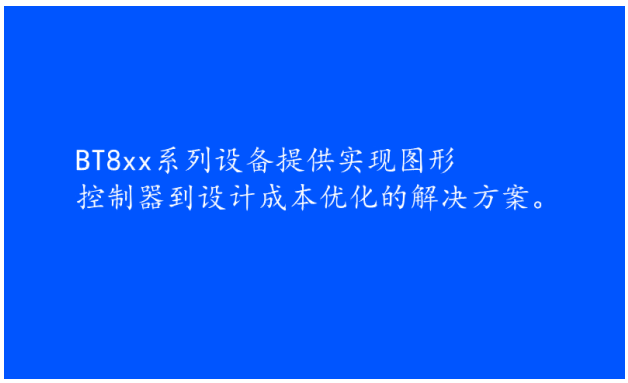
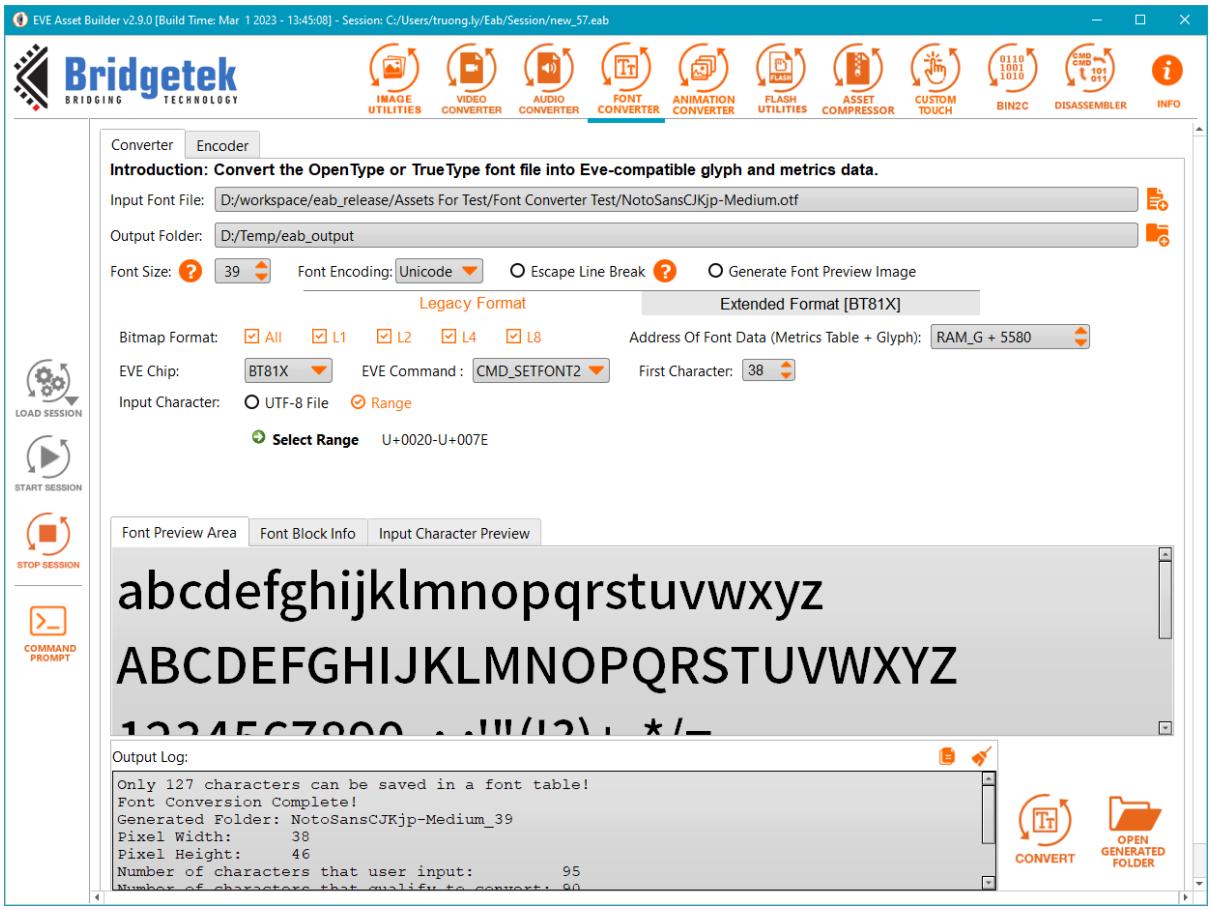


Figure 60 Font Converter example: LCD screen

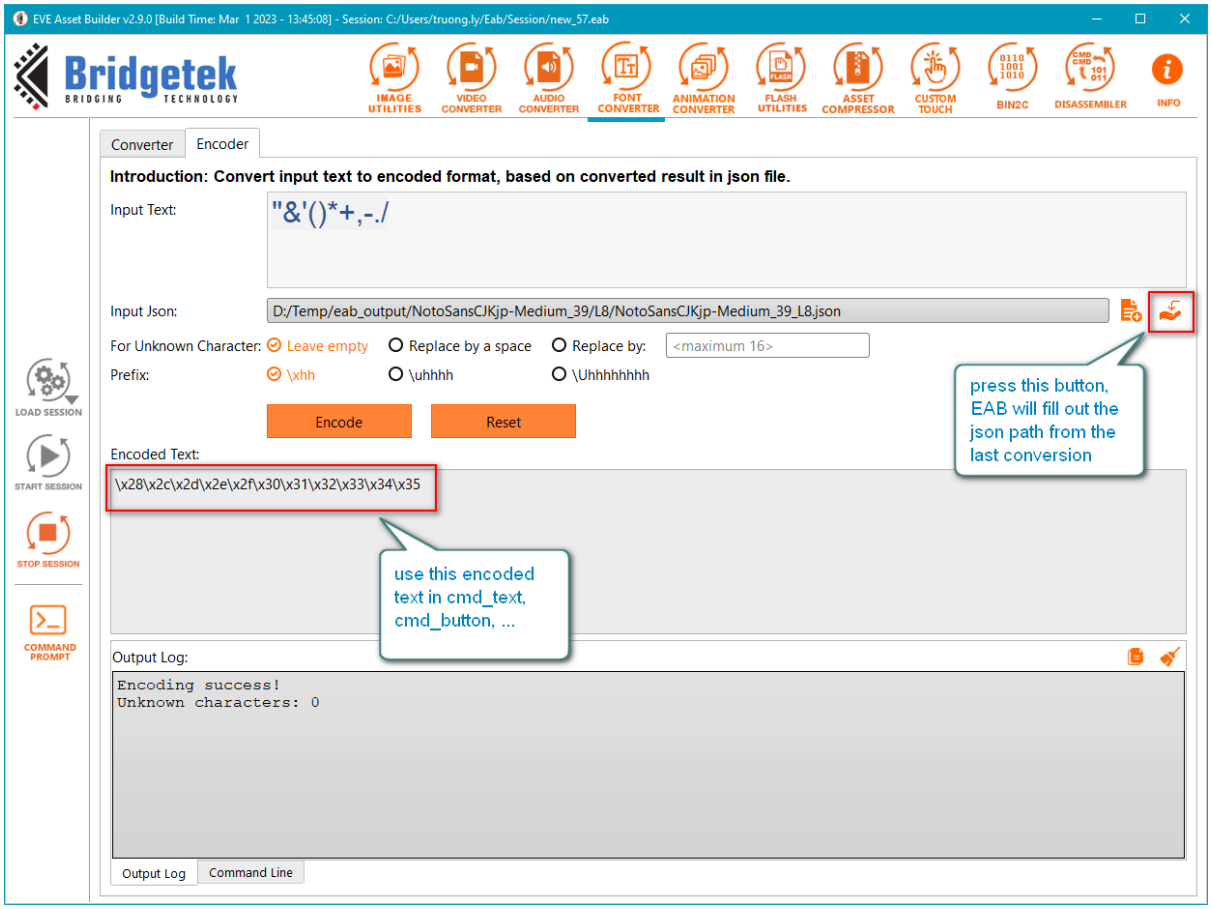
E. Text Encoder

Text Encoder helps to encode an input string into the format of \x*hh*, \u*hhhh*, or \U*hhhhhhhh*. The *hh* value is retrieved from the json file. The encoded string can be used in coprocessor command such as CMD_TEXT, CMD_BUTTON.

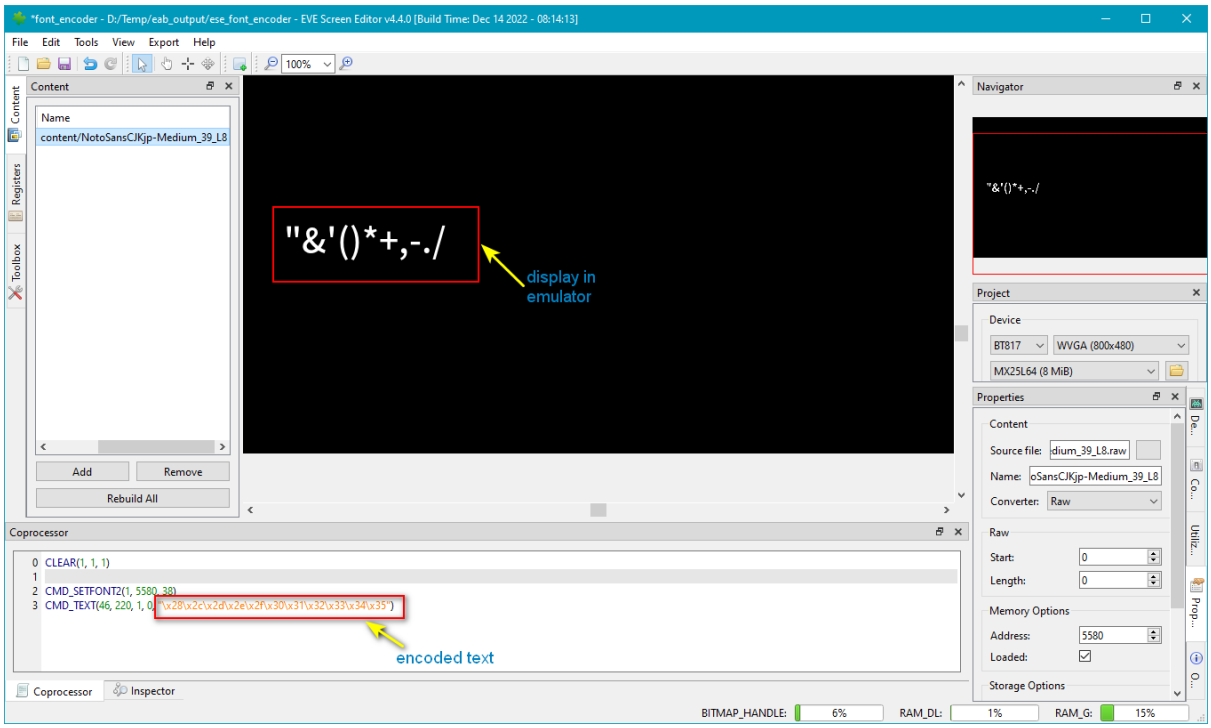
Step 1: Convert a font



Step 2: Switch to Encoder tab, then input some text



Step 3: Use the encoded text in ESE



F. Animation Converter and Generate Flash Image

Below is an example to convert a gif file into an animation, generate a flash image, and display that animation on LCD.

Step 1: Convert a gif file into EVE-compatible file

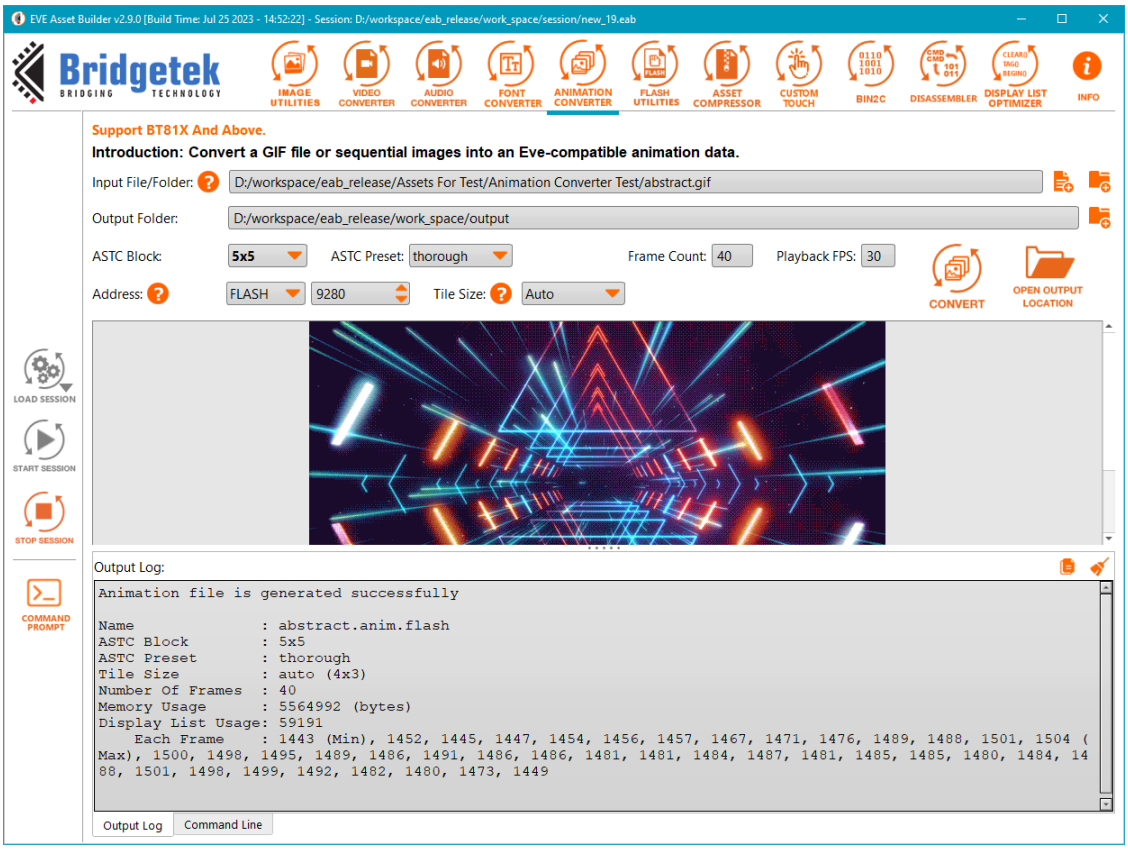


Figure 61 Animation Converter example

Output folder:

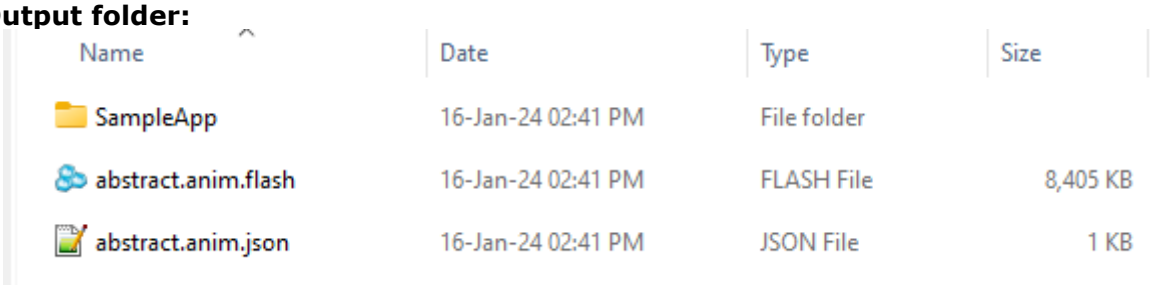


Figure 62 Animation Converter example: Output folder

Note: The .anim.flash file serves as a flash image, while the .anim.ram_g file is specifically designed for loading into RAM_G. Bitmap address are fixed.

The generated "abstract_anim.c" in SampleApp folder contains a function to load the .anim.flash file:

```
// Animation can be loaded from Flash or RAM
// Use "Generate Flash" tool to generate flash file containing the animation

#define ANIM_ADDR          (4096)
#define FRAME_COUNT        (40)

void Load_Animation_abstract(EVE_HalContext *phost)
{
    // Switch Flash to FULL Mode
    Gpu_CoCmd_FlashFast(phost, 0);
    Ftf_Write_File_To_Flash_By_RAM_G(phost, "../../../abstract.anim.flash", ANIM_ADDR);
```

Figure 63 Animation Converter example: Sample code

Step 2: Generate a flash image from .anim.flash file:

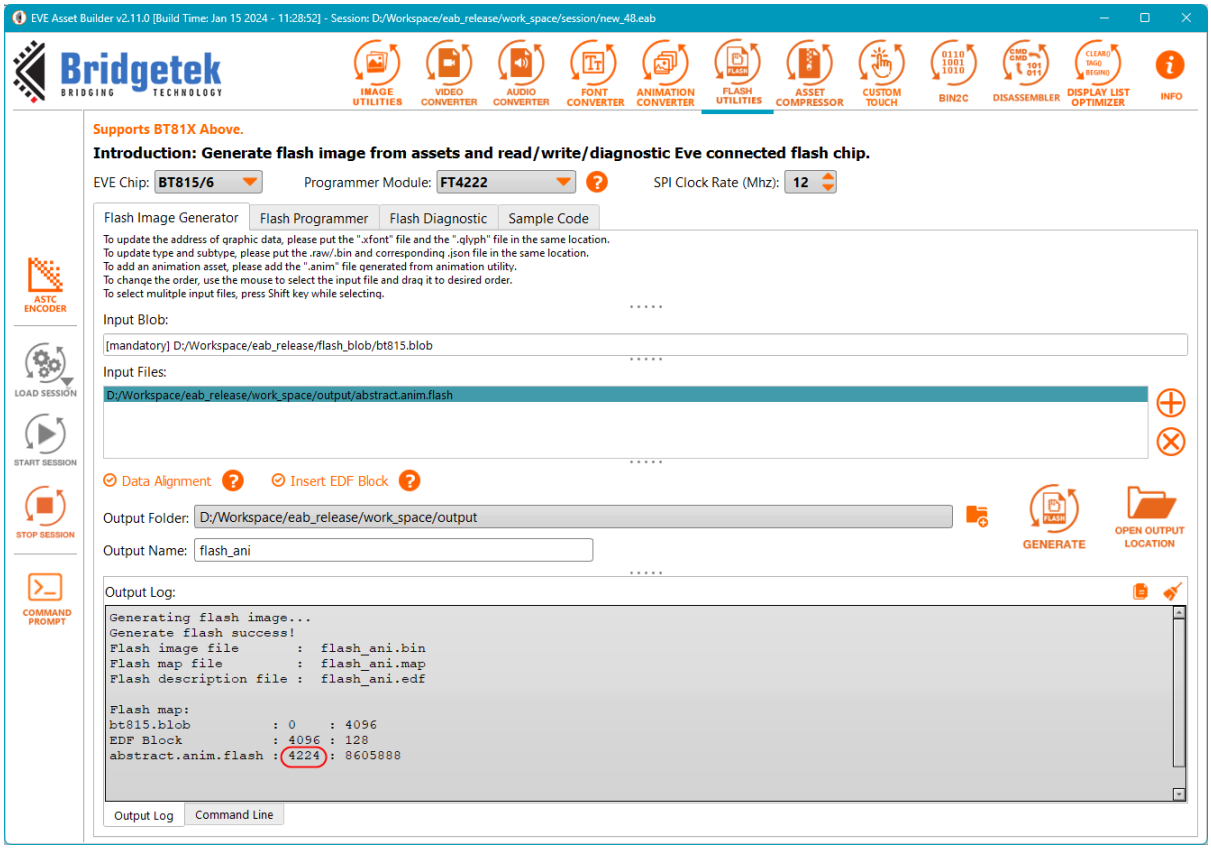


Figure 64 Animation Converter example: Generate flash

Step 3: Write the flash image into EVE's flash chip:

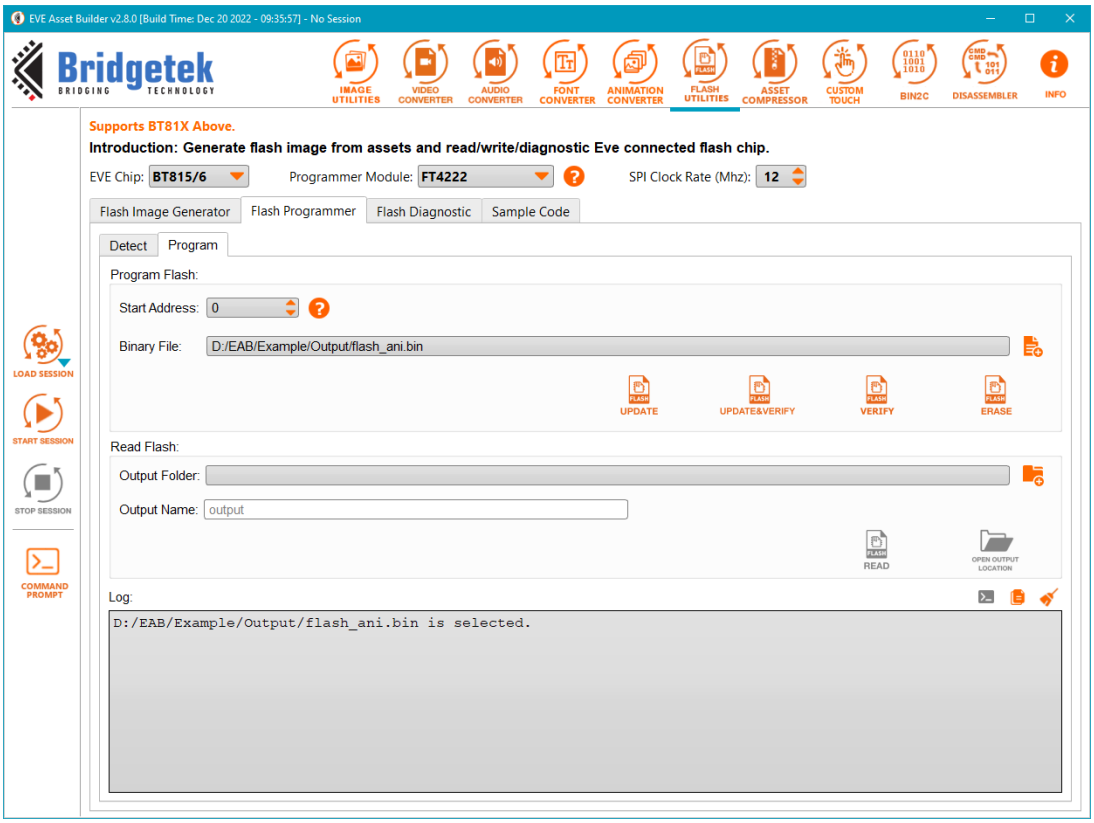


Figure 65 Animation Converter example: Program flash

Step 4: Call function Load_Animation_abstract() from SampleApp->MSVC_Emulator project, and animation will be played.

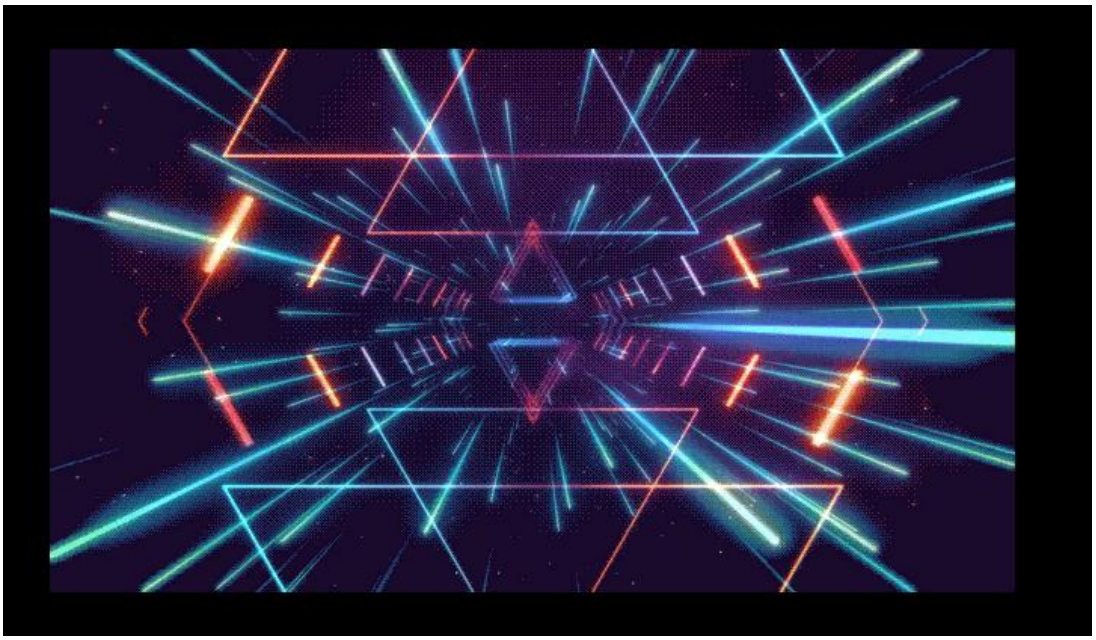


Figure 66 Animation Converter example: play back

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.

Appendix A – List of Figures

Appendix B – List of Tables

Appendix C – Revision History

Document Title	:	BRT_AN_054 EVE Asset Builder 2.11 User Guide
Document Reference No.	:	BRT_000263
Clearance No.	:	BRT#161
Product Page	:	https://brtchip.com/toolchains/
Document Feedback	:	Send Feedback

Revision	Changes	Date
Version 1.5	User Guide for EVE Asset Builder 2.0.0 release	12-08-2020
Version 1.6	Updated User Guide for EVE Asset Builder 2.1.0	17-12-2020
Version 1.7	Updated User Guide for EVE Asset Builder 2.4.0	11-11-2021
Version 1.8	Updated User Guide for EVE Asset Builder 2.5.0	19-01-2022
Version 1.9	Updated User Guide for EVE Asset Builder 2.6.1	20-09-2022
Version 2.0	Updated User Guide for EVE Asset Builder 2.7.1	22-11-2022
Version 2.1	Updated User guide for EVE Asset Builder 2.8.0	10-03-2023
Version 2.2	Updated User Guide for EVE Asset Builder 2.9.0	30-06-2023
Version 2.3	Updated User Guide for EVE Asset Builder 2.10.0	11-10-2023
Version 2.4	Updated User Guide for EVE Asset Builder 2.11.0	10-04-2024