Implémentation module LCD (BT816/EVE)

Introduction

Les modules LCD EVE (Embedded Video Engine) permettent de développer des interfaces graphiques évoluées sans pour autant avoir besoin derrière d'une grosse puissance de calcul.

Module utilisé : Riverdi 4.3" Resistive TFT

Il est équipé d'une puce FTDI BT816 et d'une mémoire flash de 8Mo (64Mb).

Librairie Arduino

Librairie utilisée pour fonctionner avec un Arduino M0 : Rudolph Riedel FT800-FT813

Les fichiers de la librairie sont répartis de la sorte :

- Dans un dossier "FT800-FT813" placé le dossier "Librairies" de l'Arduino : EVE.h, EVE_commands.cpp, EVE_commands.h, EVE_config.h, EVE_target.c, EVE_target.h
- Les fichiers tft.cpp, tft.h, tft_data.c et tft_data.h sont directement lié au projet Arduino

Le fichier EVE_config.h doit être modifié pour spécifier le type de LCD utilisé (ligne ~119) :

```
#define EVE_RiTFT43
```

Connexion à l'Arduino M0

Le branchement du LCD à l'Arduino M0 se fait via le port SPI matériel dédié.

LCD (RiBUS)		Arduino M0
1	VDD-3.3v	3.3v
2	GND	GND
3	CLK	CLK
4	MISO	MISO
5	MOSI	MOSI
6	CS	D9
8	RESET	D8
17	VDD-5V	5v

Les ports D8 et D9 sont spécifiés dans le fichier EVE_target.h

Pour faciliter les tests, j'utilise un connecteur FFC/FPC 0.5mm 20 pins monté sur un PCB. Attention sur l'exemplaire reçu, la numérotation des pins est inversée selon le coté de lecture du PCB !!

Utilitaires

EVE Asset Builder

Matériel et branchements

Attention au sens de la nappe dans les connecteurs

Attention à la nappe utilisée (connecteurs opposés dans mon cas)

Il faut ensuite envoyer ces fichiers vers la mémoire flash du module LCD. Pour cela il faut un petit module permettant de faire l'interface entre le port USB du PC et le bus SPI.

J'utilise le module "UMFT4222EV-D" de "FTDI" (Mouser.fr).

Coté LCD, il faut se baser sur la documentation RiBUS

LCD (RiBUS)		UMFT4222EV-D
1	VDD-3.3v	JP5.2
2	GND	JP5.3
3	CLK	JP5.9
4	MISO	JP5.8
5	MOSI	JP5.7
6	CS	JP4.9
8	RESET	JP5.11
11	SPI-IO2	JP5.5
12	SPI-IO3	JP5.4
17	VDD-5V	JP4.2

Convertir les images

Aller dans "**IMAGE CONVERTER**", ajouter la ou les images que l'on souhaite utiliser grâce au bouton "**ADD**".

Sélectionner un dossier de destination.

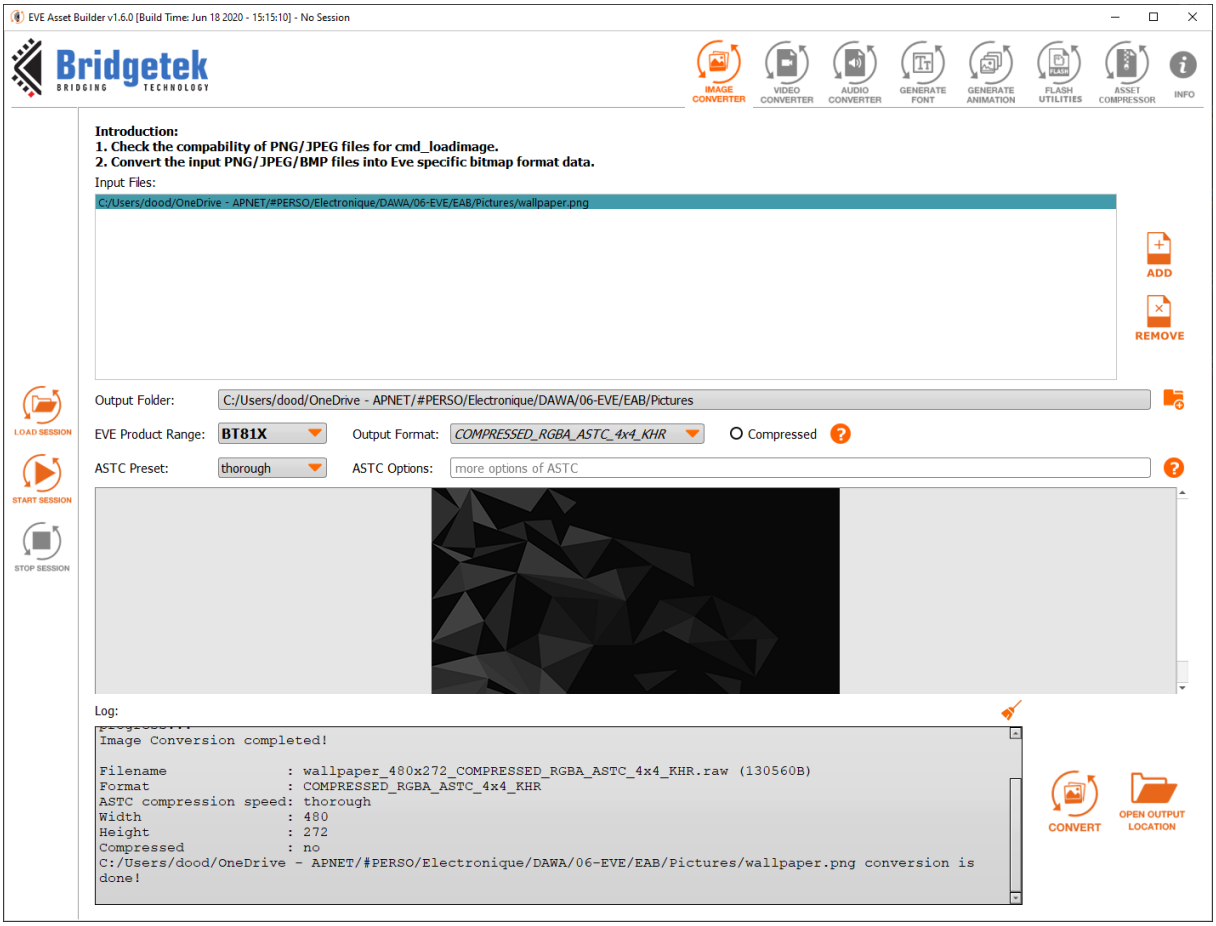
Spécifier le module du LCD (**BT81X** ici)

Spécifier le format de sortie : "**COMPRESSED_RGBA_ASTC_4x4_KHR**"

Ne pas cocher la case "**Compressed**"

Cliquer sur "**CONVERT**"

Un fichier avec l'extension "**.raw**" est généré.



Convertir les polices de caractères

Aller dans "**GENERATE FONT**", ajouter la police que l'on souhaite utiliser grâce au bouton "**ADD**" (champ "**Input Font File**").

Définir la taille de la police souhaitée, si plusieurs tailles sont nécessaires il faut faire une conversion par taille.

Sélectionner un dossier de destination.

Sélectionner l'onglet "**Extended Format [BT81X]**".

Dans "**Bitmap Format**" sélectionner "**ASTC**"

Dans "**Address Of Glyph Data**" sélectionner "**FLASH/4096**"

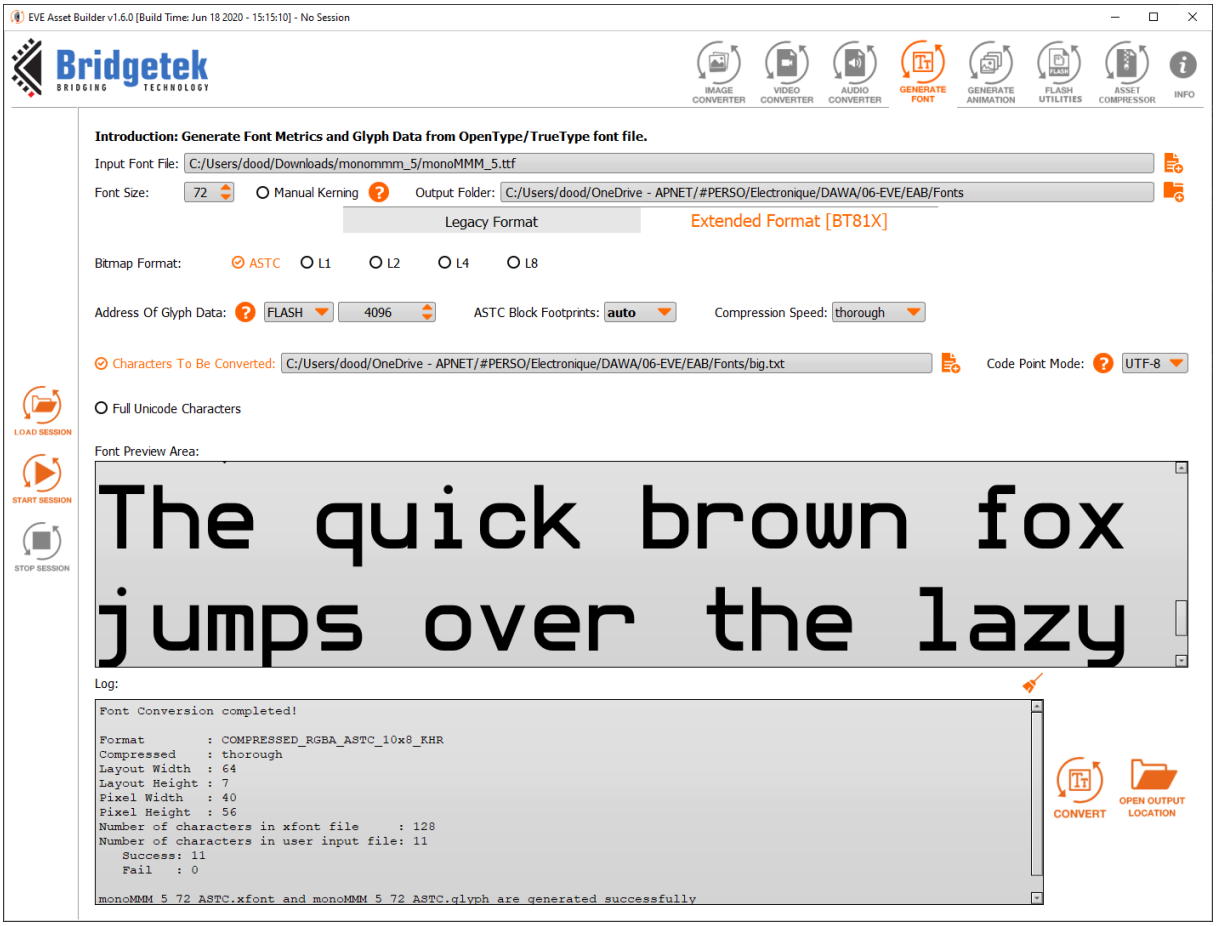
Dans "**ASTC Block Foorprints**" sélectionner "**AUTO**"

Dans "**Compression Speed**" sélectionner "**thorough**"

Sélectionner ensuite les caractères de la police à convertir :

- Tous : "**Full Unicode Characters**"
- Certains d'entres eux : "**Characters To Be Converted**" (spécifier un fichier texte contenant les caractères à convertir)

Cliquer sur "**CONVERT**"



Plusieurs fichiers sont générés, le .glyph et le .xfont nous intéressent.

Générer une image flash

Aller dans "**FLASH UTILITIES**" puis "**Generate Flash Image**", cliquer sur "**ADD**" et ajouter le ou les fichiers .bin générés précédemment.

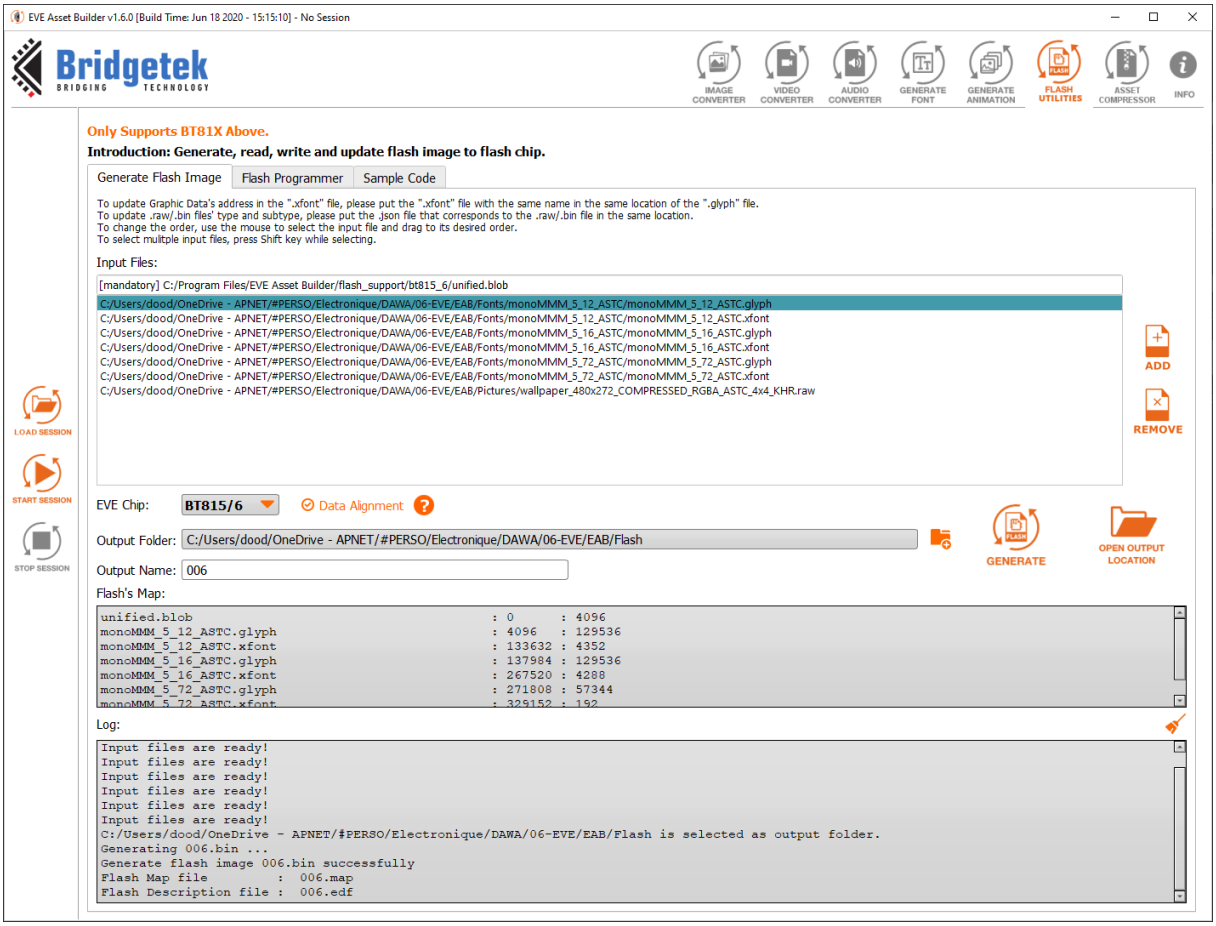
Les polices seront renseignées en premier (.glyph puis .xfont), ensuite spécifier les images (.raw).

Spécifier le module du LCD (**BT815/6** ici)

Cocher la case "**Data Alignement**"

Sélectionner un dossier de destination et un nom de fichier

Cliquer sur "**GENERATE**"



Un fichier avec l'extension "**.bin**" est généré.

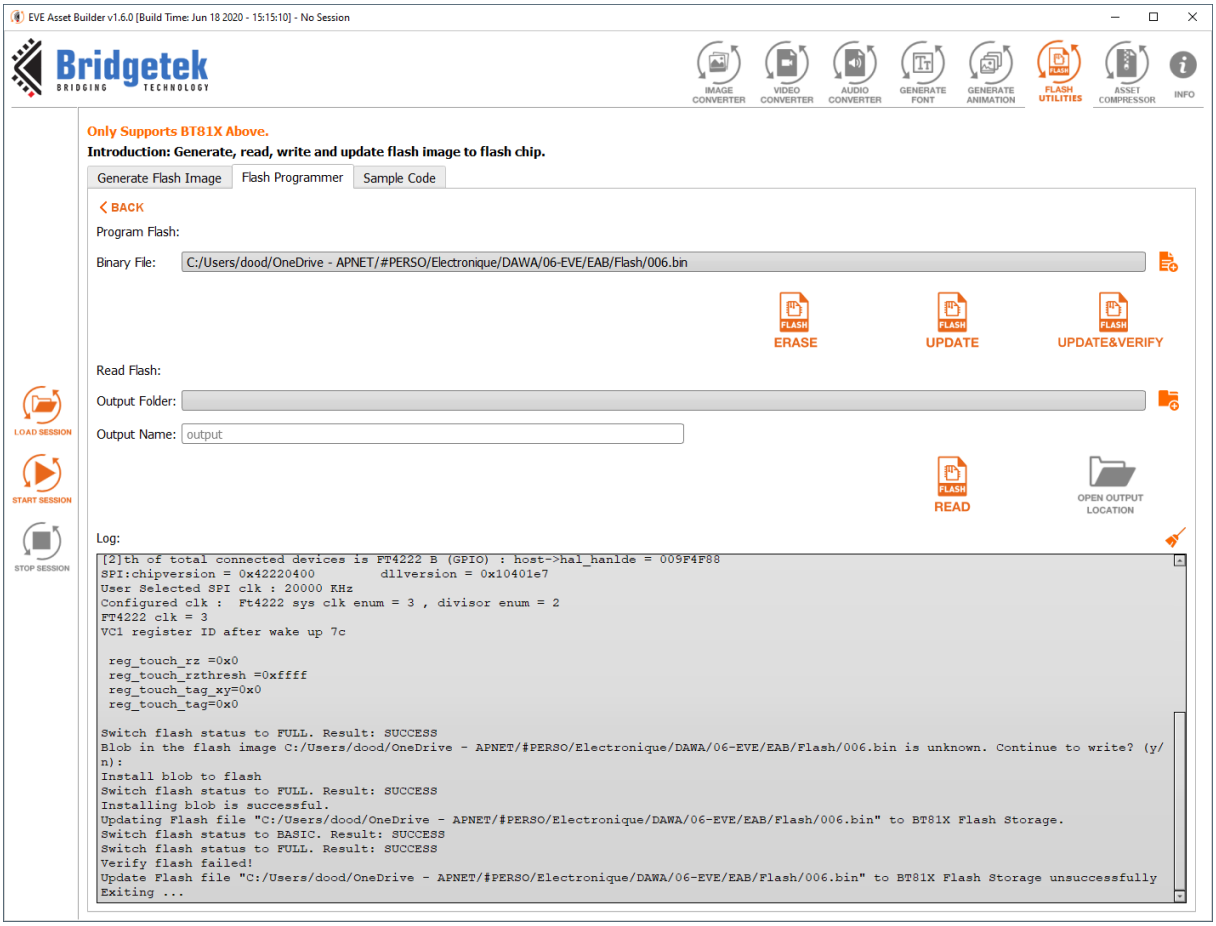
Important, les adresses de début et de fin de chaque fichiers sont spécifiées dans la partie "**Flash's Map**". Ces valeurs seront utilisées par la suite.

unified.blob : 0 : 4096
monoMMM_5_12_ASTC.glyph : 4096 : 129536
monoMMM_5_12_ASTC.xfont : 133632 : 4352
monoMMM_5_16_ASTC.glyph : 137984 : 129536
monoMMM_5_16_ASTC.xfont : 267520 : 4288
monoMMM_5_72_ASTC.glyph : 271808 : 57344
monoMMM_5_72_ASTC.xfont : 329152 : 192
wallpaper_480x272_COMPRESSED_RGBA_ASTC_4x4_KHR.raw : 329344 : 130560

Aller dans "**FLASH UTILITIES**" puis "**Flash Programmer**".

Sélectionner le module "**FT4222**" puis cliquer sur "**DETECT**" pour vérifier le bon fonctionnement du module.

Cliquer ensuite sur "**PROGRAM**", sélectionner le dernier fichier .bin généré et cliquer sur "**UPDATE**".



Malgré les erreurs le fichier est généralement bien écrit dans la mémoire flash. Cela peut être vérifié en lisant la mémoire flash et en comparant le fichier obtenu avec le fichier programmé initialement.

Charger l'ensemble dans le programme

Différentes actions sont nécessaires.

Définir les adresses en RAM des différents éléments à copier.

La première police sera copiée à l'adresse 0.

La deuxième police sera copiée à l'adresse 4352 (c'est la taille qu'occupe le premier fichier .xfont)

La troisième police sera copiée à l'adresse 8640 (c'est la taille qu'occupe le premier + le deuxième fichier .xfont)

L'image sera copiée à l'adresse 8832 (somme de la taille des 3 polices : 4352 + 4288 + 192)

Les fichiers .glyph restent en mémoire flash, ils ne sont pas copiés en RAM !

```
/* memory-map defines */
#define MEM_FONT_01 0
#define MEM_FONT_02 4352
#define MEM_FONT_02 8640
#define MEM_PICT_01 8832
```

Il faut ensuite indiquer au programme d'aller lire la mémoire flash de l'écran TFT (dans laquelle on à programmé notre .bin) pour copier le contenu en RAM.

```
/* init */
EVE_init_flash();
EVE_cmd_flashread(MEM_FONT_01, 133632, 4352); /* copy from FLASH (read 4288 bits starting offset 84608) to G-RAM (MEM_FONT_01/.xfont) */
EVE_cmd_flashread(MEM_FONT_02, 267520, 4288);
EVE_cmd_flashread(MEM_FONT_03, 329152, 192);
EVE_cmd_flashread(MEM_PICT_01, 329344, 130560);
```

Enfin, les données peuvent être utilisées, ici à l'initialisation :

```
/* init background */
[...]
EVE_cmd_setbitmap(MEM_PICT_01, EVE_COMPRESSED_RGBA_ASTC_4x4_KHR, 480, 272);
[...]
EVE_cmd_setfont2(12, MEM_FONT_01, 32);
EVE_cmd_setfont2(13, MEM_FONT_02, 32);
EVE_cmd_setfont2(14, MEM_FONT_03, 32);
EVE_cmd_text_var(EVE_HSIZE - 10, EVE_VSIZE - 15, 12, EVE_OPT_RIGHTX, LABEL_VERSION, 0);
```

Les polices personnalisées peuvent être attribuées à des "slots" prévus à cet effet (12, 13, 14 ici).

EVE Screen Editor

Cet utilitaire permet de réaliser le design en faisant des drag&drop des différents éléments souhaités (couleurs, textes, images, etc) puis de générer automatiquement le code associé.

Attention, le code généré convient à l'environnement de développement proposé par Bridgetek et non à la librairie utilisée avec l'Arduino. Néanmoins les commandes sont très similaires et demandent du coup peu d'adaptation.

Exemples

Afficher un bouton

```
EVE_cmd_dl(DL_COLOR_RGB | WHITE);
EVE_cmd_fgcolor(0x00c0c0c0); /* some grey */
EVE_cmd_dl(TAG(10)); /* assign tag-value '10' to the button that follows */
EVE_cmd_button(20, 20, 80, 30, 28, toggle_state, "Touch!");
EVE_cmd_dl(TAG(0)); /* no touch */
```