

Environnement de développement, Exemples de projets, création de projet

Lancement de l'environnement de développement Visual HAM

Le kit de développement est déjà installé sur les postes windows de l'ECE des salles de TP : P317 P347 P429 B421 B321 A506

*Pour ceux et celles souhaitant installer ce kit sur une machine personnelle, voir la rubrique « Kit de développement à télécharger » du site.
Pour les machines en système Windows (XP ou Vista) l'installation est directe : décompresser l'archive dans un répertoire d'accueil c'est tout, le lancement de l'environnement de développement se fait ensuite en double cliquant sur VHAM.exe
Pour les machines en Linux ou MacOSX il y a une procédure un peu plus compliquée à suivre et vous devrez configurer votre propre environnement de développement. Pour l'instant la compilation de projet passe par un script. Il est possible d'adapter le makefile du kit windows pour obtenir une solution plus propre.*

Les projets exemples sur lesquels vous allez travailler durant ce TP sont à télécharger sur le site www.ece.fr/~fercoq et à décompresser dans votre répertoire de travail, par exemple Z : \informatique\arm\

Lancer l'application VHAM (environnement de développement) à partir du menu windows :
-> Démarrer -> Systèmes Embarqués -> Vham -> VHAM

A la première exécution l'éditeur VHAM va peut-être préciser qu'il souhaite accéder à la « base de registres » ce qui nécessite des droits administrateur, vous pouvez valider (ceci n'a pas d'importance pour la suite).

Compilation et exécution d'un 1er exemple

Ouvrir le 1er exemple de projet depuis VHAM : menu -> File -> Open Workspace

Attention il ne faut pas ouvrir uniquement un fichier (menu Open) mais bien un projet complet (menu Open Workspace)

Sélectionner le répertoire des projets exemples, puis `exemple_1_animation\project.vhw`

Les différents fichiers sources du projet apparaissent dans le volet gauche de vham
Double cliquer sur un fichier pour l'ouvrir dans l'éditeur, par exemple main.c

Les commandes de compilation et d'exécution sont dans la barre d'outils : 

Pour compiler le projet à partir des fichiers sources : **Build** ou **F5**
Pour exécuter le projet après compilation : **Run in VBA** ou **F6**
Pour effacer l'exécutable et les fichiers objets : **Clean** ou **F10**

- Compiler le projet (F5)
- Vérifier que la compilation ne génère pas de message d'erreur (panneau du bas : la compilation doit se terminer par le message "ROM fixed!")
- Lancer l'exécutable sur le simulateur VBA (F6) (sur ce projet un ruban multicolore doit balayer l'écran en diagonales...)

Notes :

Pour pouvoir recompiler après une modification du projet, il est nécessaire de fermer d'abord le simulateur VBA
Si la compilation donne <<make.exe: Nothing to be done for `build'.>> c'est que l'exécutable est déjà à jour (pas besoin de re-compiler)
En cas de besoin, pour forcer une re-compilation totale du projet (rebuild all) faire F10 puis F5

Informations sur la structure des projets

Les projets partagent une structure commune :

Fichiers sources principaux (Source Files)

- main.c Contient le point d'entrée du programme (le main) : appelle les autres fonctions, peut servir de test pour de petits algos codés en C
- asm.s Cadre pour les sous programmes assembleurs que vous écrirez
- armgba.c Contient les sous programmes C de la mini-librairie du kit
- asmarmgba.s Contient les sous programmes assembleur de la mini-librairie du kit

Fichiers en-têtes à inclure (Header Files)

- projet.h Contient les déclarations (prototypes en C) pour les sous programmes C ou assembleur que vous ajouterez au projet
- armgba.h Déclarations concernant la mini-librairie du kit : en-tête pour source C, inclus automatiquement avec projet.h
- incarmgba.s Déclarations concernant la mini-librairie du kit : en-tête pour source assembleur

Fichier de compilation

- makefile Décrit le processus et les options de compilation du projet

Les sous programmes de la mini-librairie, utilisables depuis le C, sont décrits dans armgba.h, avec quelques déclarations de types et constantes utiles.

Informations sur l'affichage

La mémoire vidéo de la GBA (VRAM) est accessible directement en lecture et en écriture à partir de l'adresse `0x06000000`.

En mode 3, que nous utiliserons exclusivement en TD/TP et pour le projet, l'écran a une largeur de 240 pixels, une hauteur de 160 pixels, et chaque pixel est représenté par un demi-mot (2 octets) en mémoire vidéo selon le format BGR 15 bits.

Bleu (entre 0 et 31) $b_4 b_3 b_2 b_1 b_0$ Vert (entre 0 et 31) $g_4 g_3 g_2 g_1 g_0$ Rouge (entre 0 et 31) $r_4 r_3 r_2 r_1 r_0$
Le tout tient dans un demi mot de 16 bits : $0 b_4 b_3 b_2 b_1 b_0 g_4 g_3 g_2 g_1 g_0 r_4 r_3 r_2 r_1 r_0$

Les couleurs des pixels de l'écran (2D) sont stockées en mémoire (1D) selon la convention de lecture de gauche à droite et de haut en bas. On peut ainsi calculer l'adresse d'un pixel à lire ou écrire à partir des coordonnées, selon la formule $adresse = 0x06000000 + 2*x + 480*y$. On peut aussi calculer le numéro du pixel à lire ou écrire, à partir des coordonnées, selon la formule $numpixel = x + 240*y$.

En C l'accès en lecture ou écriture peut se faire

- soit à partir de l'adresse : `*(unsigned short int*)(0x06000000+2*x+480*y)`
- soit comme un tableau : `((unsigned short int*)0x06000000)[x+240*y]`

L'écran ne reflète pas instantanément l'état de la mémoire VRAM. La prise en compte effective des données a lieu 60 fois par seconde selon un balayage de gauche à droite et de haut en bas. On parle de rafraîchissement de l'écran, avec une fréquence de 60Hz. La fin d'un balayage complet de l'écran correspond à un événement que nous appellerons "vSync" (synchronisation avec le balayage vertical).

Analyse de l'exemple 1

Le 1er exemple de projet est programmé entièrement en C au niveau du fichier source main.c

Vérifier que la procédure assembleur `asmDrawPixel` s'utilise de la même manière que la procédure C `drawPixel` en remplaçant les appels (Puis recompiler F5, et exécuter F6) :

```
drawPixel(x,y,color32);    ->    asmDrawPixel(x,y,color32);
drawPixel(x+1,y,color32); ->    asmDrawPixel(x+1,y,color32);
```

Trouver, dans les fichiers du projet, les codes sources de la version C et de la version assembleur : comparer les tailles.

Cela signifie-t-il nécessairement que la version C est plus performante que la version assembleur ?

Dans ce programme, la variable 32 bits `u32 color32` est utilisée comme 3ème argument pour appeler le sous programme d'affichage de pixel. Les déclarations de `drawPixel` et `asmDrawPixel` précisent que le 3ème argument est de 16 bits. Dans ce cas, seuls les 2 octets de poids faibles (bits 0 à 15) de `color32` seront pris en compte.

Quelle est en binaire la valeur initiale de la variable 32 bits `color32` ?

A quelle couleur cela correspond-t-il selon le format BGR 15 bits ?

Quel est le rôle de la ligne de code C suivante ? `color32=(color32>>1) | (color32<<31);`

Ce code répète la même séquence de couleurs au bout de combien d'itérations?

Que se passe-t-il si vous enlevez l'appel à la procédure d'attente de rafraîchissement écran `vSync()` de la boucle d'animation ?

Facultatif : Au lieu de l'appeler à chaque itération ou pas du tout, pouvez-vous appeler `vSync()` une fois sur 32 ?

Exemple 2

L'exemple 2 illustre les accès à l'état des touches de la GBA depuis du code C.

Ouvrir le nouveau projet (confirmer la fermeture du projet actuel) `exemple_2_interaction\project.vhw` puis compiler et exécuter.

Vérifier que chaque touche de la GBA simulée réagit (changement de couleur au niveau des carrés affichés en haut de la fenêtre).

Les touches de la GBA et les touches du simulateur VBA (clavier PC) sont indiquées dans `armgba.h`

Par exemple la touche Start de la GBA correspond à ENTREE pour le simulateur...

Dans le fichier source main.c `REG_KEYINPUT` correspond au demi-mot contenu en mémoire à l'adresse 0x04000130.

```
#define REG_KEYINPUT *(volatile u16 *)0x04000130
```

Cette adresse correspond en fait au périphérique de lecture d'état des touches (1 bit par touche, 0 pour appuyé, 1 pour relâché)

Pour ces adresses particulières qui accèdent à des périphériques on parle de "registre d'entrée/sortie" (ne pas confondre avec registre du processeur)

En utilisant `REG_KEYINPUT` et `asmMakeColor`, compléter proprement le code du main pour que la couleur du carré qui se déplace change dès qu'on appui sur les touches W ou X (clavier PC).

Création d'un nouveau projet à partir du modèle

Avec l'explorateur windows créer un répertoire d'accueil pour le projet sur z: (par exemple `z:\informatique\arm\projet_tp1`)

Depuis VHAM faire `menu -> File -> New -> New Project`

Sélectionner type de projet `armgba` (seul choix possible)

Donner un nom au projet (par exemple `projet_tp1`) et sélectionner l'emplacement du répertoire [...] et enfin confirmer [OK]

VHAM copie l'ensemble des fichiers d'un projet modèle (`vham\templates\project\armgba`) dans le répertoire du nouveau projet.

L'ensemble est un cadre vide qui compile mais ne fait rien.

Créez une procédure en C `testerPixels`, appelée une fois depuis le main, qui dessine quelques pixels à l'écran sans utiliser la procédure `drawPixel` ou `asmDrawPixel` mais qui accède directement à la mémoire vidéo.

Par exemple `*(u16*)(0x060096F0)=0x7FFF; // dessine un pixel blanc au centre`

Notes :

- La calculatrice windows possède un mode affichage scientifique qui permet de faire des calculs et conversions décimal-<->hexa

- `armgba.h` définit différentes constantes symboliques utiles pour améliorer la lisibilité du code :

```
VRAM = 0x06000000  SCREENW = 240  SCREENH = 160  SCREENLINE = 480 (nombre d'octets dans une ligne)
u16 est un typedef (synonyme) pour unsigned short int
```

En modifiant une adresse en mémoire vidéo on se déplace au niveau des pixels, +2 octets = 1 pixel à droite, +480 = 1 pixel en dessous ...

Tester la procédure suivante, et sur le même principe proposer un autre dessin géométrique simple (trait vertical ou diagonal, contour...)

```
void petitSegment() {
    int i,adressePixel=VRAM+2*120+480*80;
    for (i=0;i<10;i++){
        *(u16 *)adressePixel=0x7FFF;
        adressePixel=adressePixel+2;
    }
}
```