Ensimag 3A, filière SLE

2015-2016

Modélisation TLM en SystemC
TP n°2 : Utilisation de composants existants

Placez-vous dans le répertoire `TPs/squelette/tp2`. Comme précédemment, il faudra positionner quelques variables d'environnement avec le fichier `setup.sh` adapté à votre machine, par exemple :

```
source /matieres/5MMMTSP/tlm/setup-ensimag.sh
```

avant toute compilation/exécution.

Dans la suite, on considerra que les adresses (i.e. `ensitlm::addr_t`) et les données (i.e. `ensitlm::data_t`) sur **32 bits**.

Style de codage imposé

Une part de la note sera donnée sur la présentation du code. On vous impose le style suivant (qui est en principe suivi sur le code fourni, si ce n'est pas le cas signalez-le comme un bug) :

- Indentation avec des tabulations et non des espaces
- Accolades ouvrantes sur la même ligne que la ligne de code qui lui correspond (par exemple, `if (toto) {` sans retour à la ligne entre `)` et `{`.
- Placement des espaces autour des opérateurs, parenthèses, ... en suivant le style de codage de Linux (`https://www.kernel.org/doc/Documentation/CodingStyle`, Chapitre 3.1).
- Pas de ligne long (i.e. de plus de 80 colonnes)
- Pas d'espace en fin de ligne.

Merci de respecter ce style *scrupuleusement*. Voir aussi `http://ensiwiki.ensimag.fr/index.php/Ecrire_du_code_de_qualite` pour plus d'information.

Une part de la note sera également accordée à la propreté du rendu : rendez tout ce qui est nécessaire à votre TP et seulement cela. Ne rendez pas de fichiers inutiles (par exemple, ne rendez pas les fichiers binaires ni les fichiers de sauvegardes de votre éditeur de texte). Faites également le ménage dans votre code (supprimez les commentaires « pense-bêtes » par exemple.

Objectif

Dans la plate-forme créée au TP°1, on cherche à rajouter un composant contrôleur d'écran plat (LCD Controller ou LCDC). On souhaite ensuite le tester pour s'assurer de son bon fonctionnement. La documentation du composant LDCD est disponible dans le fichier `LCDC-doc.pdf`.

Question 1 : instanciation du contrôleur LCD

— Les fichiers correspondant au contrôleur LCD et à la ROM (que nous utiliserons plus tard) sont dans votre archive Git, dans le répertoire `squelette/tp2/`. Vous retrouverez le bus, identique à celui du TP1, et le composant `Memory`, similaire à celui que vous avez écrit vous-mêmes pour le TP1, plus d'autres nouveaux composants.
— Vous pouvez recopier `sc_main()` et le composant générateur que vous avez écrit pour le TP1 comme base de travail. Effacer les commandes de test de la mémoire écrites au TP n°1 dans le générateur de transaction (la méthode associée au **`SC_THREAD`** doit être vide).
— Instancier le module LCDC et le connecter au reste de la plate-forme (il faudra ajouter un port d'interruption au générateur, nous l'utiliserons plus tard). Le constructeur du contrôleur prend un paramètre additionnel de type `sc_time` permettant de fixer la fréquence de rafraichissement de l'écran LCD. Passer comme paramètre : `sc_time(1.0 / 25, SC_SEC)` (rafraichissement 25 fois par seconde).
— Compiler et tester.

Question 2 : dimensionnement des plages d'adresses

L'affichage d'une image sur l'écran LCD par le contrôleur nécessite un espace mémoire dédié pour stocker cette image, appelé *mémoire vidéo* ou *video memory*. Nous allons utiliser la mémoire déjà présente dans la plate-forme pour cet espace.

La mémoire est dimensionnée comme suit :

— 10 Kio sont réservés pour les données du logiciel embarqué (cette partie sera inutilisée dans notre plate-forme qui ne modélise pas le logiciel embarqué fidèlement).
— La mémoire vidéo utilise un codage d'1 octet par pixel, donc sa taille sera $320 * 240 = 76800$.

La taille nécessaire pour la mémoire est donc 87040 = 0x15400. Modifiez la taille de la mémoire et la plage d'adresse correspondante (`bus.map()`) dans `sc_main()`.

Le LCDC n'expose qu'un (petit) banc de registres. Calculez la taille nécessaire d'après la documentation, et adaptez la plage d'adresse.

Question 3 : modélisation du registre `START_REG` du LCDC

Dans la version récupérée du contrôleur LCD, il manque l'implémentation du registre `START_REG` (voir documentation).

— Dans le fichier `LCDC.cpp`, implémenter le registre. Que faut-il faire ? Utiliser les constantes définies dans le fichier `LCDC_registermap.h`.
— Compiler et tester.

Question 4 : premiers tests du LCDC

— Définir dans le générateur des constantes pour les différentes adresses que l'on manipulera par la suite : adresse de base de la mémoire, adresse de base de la partie mémoire vidéo, adresse de base du contrôleur LCD (utiliser des directives **`#define`**, ou **`const`** `ensitlm::addr_t ... = ...;`).
— Dans le générateur, réaliser des écritures dans la mémoire vidéo de façon à afficher une image blanche (c'est à dire, donc la mémoire vidéo est une suite de `0xFFFFFFFF`), avec une boucle simple du type

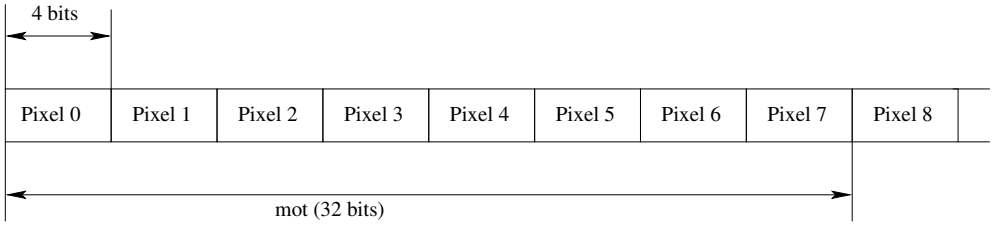
```
for (int i = 0; i < IMG_SIZE; i++) {
```

```
        ...;
}
```

— Compiler et tester.
— Réaliser les écritures appropriées dans les registres du LCDC pour observer effectivement l'image à l'écran (penser à utiliser les constantes définies dans le fichier `LCDC_registermap.h`).
— Compiler et tester.

Question 5 : Ajout d'un composant ROM

Nous allons maintenant afficher une image sur l'écran. L'image est stockée dans un composant ROM, dans un format très légèrement différent de celui utilisé par le contrôleur LCD : l'image est stockée avec 4 bits par pixels, en niveau de gris. Le LCDC utilisant 8 bits par pixels, on considère que les 4 bits stockés dans la ROM correspondent aux bits de poids fort des 8 bits de la mémoire vidéo. Les pixels de 4 bits sont stockés les uns à la suite des autres, en format big-endian :



— Instancier le composant ROM, le connecter au bus dans `sc_main.cpp`.
— Ajouter une plage d'adresse, sachant que l'image fait 38400 octets, et est stockée au début de la ROM.
— Reprendre la boucle d'initialisation de la mémoire à `0xFFFFFFFF`, et remplacer le corps de la boucle par une fonction qui lit un groupe de pixels (un mot de 32 bits) depuis la ROM, et écrit les deux groupes de pixels correspondants dans la mémoire vidéo. Les opérateurs de décalages de bits (`<<` et `>>`) et les masques de bits (`x & 0x00F00000`) devraient vous aider. Cette partie est réalisable en une dizaine de lignes de code.

Question 6 : traitement des interruptions

L'objectif de cette question est de traiter les interruptions du LCDC, de façon à pouvoir réaliser des animations à l'écran. Cela suppose de pouvoir écrire les données en mémoire vidéo au « bon moment », c'est à dire entre deux interruptions.

— Rajouter un port d'entrée d'interruption « `irq` » au générateur et le connecter au signal d'interruption issue du LCDC. Attention, le signal d'interruption va être piloté par deux composants (`LCDC`, et indirectement le générateur de trafic dans son traitement de l'interruption), donc il faudra instancier le signal correspondant comme ceci :

```
sc_signal<bool, SC_MANY_WRITERS> irq_signal("IRQ");
```

— À la suite du code déjà écrit dans le **`SC_THREAD`** du générateur, rajouter une attente sur l'interruption. Vous procéderez en déclarant une **`SC_METHOD`** qui notifiera un événement débloquant le **`SC_THREAD`** principal. Afficher après cette attente un message indiquant la prise en compte de l'interruption.
— Compiler et tester. Comment avez-vous fait ?
— Modifier le code du générateur de façon à avoir une boucle infinie d'attente d'interruptions. Dans cette boucle, rajouter après chaque attente les écritures permettant de générer une image différente à l'écran. On propose de décaler à chaque pas de la boucle

l'image de un pixel vers le bas (en déplaçant la partie de l'image tronquée du bas sur le haut de l'image). On peut aussi afficher un fondu vers le noir ou vers le blanc, mais attention aux manipulations de bits un peu fines !