Ensimag 3A, filière SLE

2011-2012

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 ~moy/tlm/setup-ensisun.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**.

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, SC_SEC)` (rafraichissement toutes les secondes).
- Compiler et tester.

Question 2 : dimensionnement de la mémoire

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.

– À partir des informations dont vous disposez (documentation du LCDC), calculer la taille requise pour la mémoire vidéo.
– On souhaite réserver au début de la mémoire une plage de 5 Ko pour les données du logiciel embarqué. La mémoire vidéo sera donc placée à partir de l'adresse 0x10001400. La taille totale de la mémoire est $(5 * 1024) + (320 * 240) = 81920 = 0x14000$.
– Modifier la taille de la mémoire dans le fichier `sc_main.cpp` ainsi que la plage d'adresse correspondante définie par `bus.map()`.
– Quelle est la taille de la plage d'adresse à réserver pour le contrôleur LCD ? Choisir une plage et modifier la plage d'adresse en conséquence.
– Compiler et tester.

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

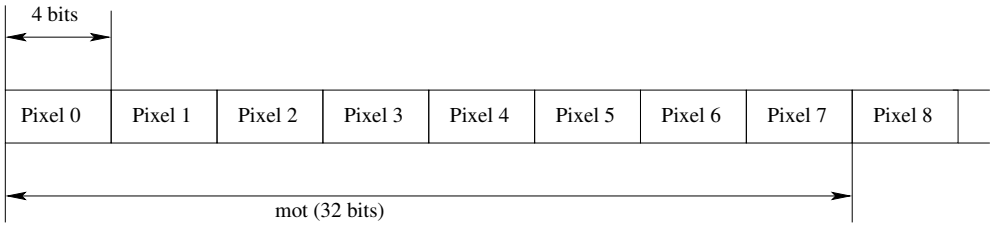
– Définir dans le générateur des constantes pour les différentes adresses que l'on manipulera par la suite : adresse de base de la mémoire, adresse de base de la partie mémoire vidéo, adresse de base du contrôleur LCD (utiliser des directives **`#define`**, ou `const ensitlm::addr_t ... = ...;`).
– Dans 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.
– 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 !