Le bootloader

Objectif

L'objectif général de ce second TME est également de faire que chaque processeur affiche un message "Hello", mais après que le code ait été préalablement déplacé de la rom vers la ram. Nous allons également gérer la première interruption. Les étapes:

1. Nous allons modifier le programme de boot pour qu'il recopie dans la ram le code des fonctions `__do_init()` et `tty_puts()` avant de d'y sauter pour les exécuter.
2. Nous allons ajouter l'affichage du *timestamp counter* de chaque processeur devant le message "hello" en utilisant une fonction tty_putx() qui affiche un nombre en hexadécimale.
3. Nous allons refaire l'étape 1 en faisant faire la copie du code par le module DMA. Vous noterez la différence de vitesse grâce à l'affichage du *timestamp*. Vous regarderez la documentation de SoCLib pour savoir comment contrôler le DMA.

1. Hello en ram

Le programme exécuté jusque là s'exécutait directement dans la ROM. Cette mémoire n'est pas cachée par les cache L1. Elle n'est donc pas très rapide.

- Nous allons placer le code des fonctions C `__do_init(), tty_puts(), tty_putx, ...` dans une autre section de la mémoire. Ces fonctions seront compilées dans le but d'une exécution dans la section ktext, mais on demandera à l'éditeur de lien de les placer dans la section ktext_lma_base de la ROM.
- Nous allons ensuite modifier le code du programme de boot pour que l'un des processeurs se charge du déplacement des programmes depuis la ROM jusque dans la mémoire RAM. Une fois que le programme est recopié, tous les processeurs saute à leur fonction `__do_init()`.

2. Hello en chiffre

Pour déterminer la durée d'exécution de certaines opération, le MIPS propose un registre dans son coprocesseur 0 qui s'incrémente à cycle d'horloge.

- Nous allons écrire une fonction `unsigned getTimeStamp()` permettant de récupérer la valeur de ce registre en y plaçant le code assembleur nécessaire. Cette fonction sera inlinée.
- Nous allons enuite écrire une fonction `tty_putx(unsigned int tty, unsigned i)}}} qui écrit sur le terminal tty, la chaine de caractère représentant le valeur du nombre i en hexadécimal.
- La fonction `__do_init()` devra écrire le message "Hello at " suivi du nombre de cycle rendu par `getTimeStamp()`

3. Hello avec le dma

Le déplacement en ram est une nécessité pour bénéficier des caches, mais la copie est lente si c'est le processeur qui s'en charge. Or il existe un composant spécialement adapté aux transfert de données, c'est le DMA.

- Nous allons modifier le programme de boot pour que la boucle de copie soit remplacée par une programmation d'une requête DMA. La terminaison de la requête DMA produit normalement une interruption. Nous n'allons pas pouvoir l'utiliser ici car le gestionnaire d'interruption n'est pas encore en place. nous allons donc tester le registre LEN du DMA afin de savoir si le transfert est terminé. Le bus système transfert en moyenne 2 octets par cycles, il est donc possible d'évaluer approximativement la fin de l'opération.

- Vous exécutez les codes `__do_init()` en notant la différence de date.
- Pour info l'ordre des registres de commande du DMA est:

```
enum SoclibDmaRegisters {
   DMA_SRC,
   DMA_DST,
   DMA_LEN,
   DMA_RESET,
   DMA_IRQ_DISABLED,
};
```