Université Lille1
Sciences et Technologies

Master d'informatique
1re année
Module d'architecture
des systèmes d'exploitation

Bibliothèque `hardware`, simulation du matériel

Philippe MARQUET Gilles GRIMAUD

mise à jour d'octobre 2008

Ce document est disponible en ligne à partir de `www.lifl.fr/~marquet/ens/hw/`. Cet accès en ligne autorise des copier/coller... ne vous en privez pas.

1 Une bibliothèque de simulation du matériel

Vous disposez d'une bibliothèque permettant l'émulation de certains composants matériels d'un ordinateur, comme une carte Ethernet, un disque dur, ou plus simplement des interruptions et une horloge.

Cette bibliothèque `hardware` est disponible sur la page web d'ASE ou directement dans `/home/enseign/ASE`. L'encart page suivante précise les fichiers fournis par cette bibliothèque.

L'interface de cette bibliothèque donnée dans `hardware.h` définit un jeu réduit de fonctions C permettant de contrôler le matériel.

Un fichier de configuration paramètre le fonctionnement du matériel émulé. Un tel fichier, nommé `hardware.ini`, vous est fourni.

- Ce paramétrage vous permet d'activer ou de désactiver les différents composants matériels émulés. Par exemple, positionner la valeur `ENABLE_HDA` à 1 permet d'activer un disque dur maître.
- Ce paramétrage définit aussi la configuration du matériel émulé. Par exemple, la valeur `TIMER_TICKS` définit le nombre de tics système par tic d'horloge.
- Ce paramétrage définit également les numéros des registres du matériel. Par exemple, la valeur `HDA_CMDREG` indique le numéro du registre de commande du disque maître ou la valeur `TIMER_PARAM` le numéro du registre de configuration du timer.
- Enfin, ce paramétrage permet de configurer les traces d'exécution qui seront produites par la simulation (paramètre `DEBUG`).

À titre d'illustration, les lignes suivantes sont extraites du fichier `hardware.ini` :

```
#
# Trace de Debug
#
# DEBUG est une somme des valeurs suivantes :
#  DEBUG_SETUP   0x0001 /* trace hardware setup */
#  DEBUG_IT      0x0010 /* trace interruptions generation */
#  DEBUG_REG     0x0100 /* trace hardware register access */
#  DEBUG_WARNING 0x1000 /* trace hardware warning messages */
DEBUG          = 0x1111

# Disque dur IDE maître
ENABLE_HDA      = 1        # ENABLE_HD = 0 => simulation du disque désactivée
HDA_CMDREG      = 0x3F6    # registre de commande du disque maître

# Configuration de l'horologe interne
```

Mise en place des travaux pratiques

Les fichiers suivants sont fournis avec la bibliothèque :

`lib/libhardware.a` la bibliothèque elle-même

`include/hardware.h` contient les déclarations de l'interface de la bibliothèque

`etc/hardware.ini` est le fichier de configuration indiquant les valeurs par défaut

La bibliothèque est installée dans les salles de travaux pratiques sous le répertoire `/home/enseign/ASE/`. Il est inutile de la recopier sur votre compte.
Vous préciserez donc au compilateur C de rechercher les fichiers d'entête de la bibliothèque par l'option de compilation :

```
-I/home/enseign/ASE/include
```

et utiliserez les options

```
-L/home/enseign/ASE/lib -lhardware
```

lors de la phase d'édition de liens.

```
TIMER_PARAM     = 0xF4    # registre de configuration du TIMER
TIMER_TICKS     = 8       # Nombre de SYSTICKS par tick d'horloge
```

Cette configuration du matériel doit en quelque sorte être considérée comme la spécification du matériel émulé. Pour commander ce matériel par programme, il est conseillé de définir un fichier d'entêtes de configuration, mettons `hw_config.h`, qui va reprendre la configuration du matériel en cohérence avec les valeurs du fichier `hardware.ini`. Il comprendrait par exemple

```
#define TIMER_TICKS    8
#define HDA_CMDREG     0x3F6
#define TIMER_PARAM    0xF4
```

Initialisation du matériel

La bibliothèque `hardware` définit la fonction :

```
int init_hardware(const char *config_file);
```

Cette première fonction permet d'initialiser le matériel émulé à partir du fichier de configuration dont le chemin d'accès est fourni en paramètre (typiquement `hardware.ini`). Si cette phase d'initialisation du matériel n'est pas effectuée, le comportement du matériel est imprévisible. Il faut donc appeler cette fonction avant toute autre opération en guise d'initialisation de vos programmes.

Manipulation des registres du matériel

La communication avec le matériel (envoi de commandes et de données, envoi/réception de données) se fait via des registres, soit en écriture soit en lecture. Les numéros de ces registres sont définis dans le fichier de configuration.

On ne peut lire ou écrire qu'un unique octet sur un registre. Pour écrire une valeurs codées sur plusieurs octets, on utilise un premier registre pour l'octet de poids fort et les registres suivants pour les octets de poids plus faible. À l'inverse, si une fonction retourne une valeur de plusieurs octets, l'octet de poids fort sera lu sur dans le premier registre, les octets de poids plus faibles dans les registres suivants.

La fonction

```
int _in(int register);
```

réalise la lecture sur le registre désigné. La valeur retournée correspond à l'octet qui a été lu sur ce registre. La fonction

```
void _out(int register, int value);
```

réalise l'écriture d'une valeur d'un octet sur le port désigné.

Gestion des interruptions matérielles

La gestion des interruptions matérielles est réalisée par les éléments suivants fournis par la bibliothèque :

- Seize niveaux d'interruptions, IRQ, sont définis. Chaque niveau correspond à un élément du matériel. Par exemple, le niveau 14 correspond au premier disque dur alors que le niveau 2 correspond à l'horloge ; on trouve les lignes suivantes dans `hardware.ini` :

  ```
  HDA_IRQ         = 14            # Interruption du disque hda
  TIMER_IRQ       = 2             # Niveau d'interruption de l'horologe
  ```

- Un traitant d'interruption doit être associé à chacune des 16 IRQ (0 à 15). Un traitant d'interruption est une fonction de type

  ```
  typedef void (*func_irq_t)();
  ```

  qui sera appelée quand l'interruption correspondante surviendra. Le vecteur

  ```
  extern func_irq IRQVECTOR[16];
  ```

  identifie ces fonctions et se doit d'être initialisé.
- La fonction de masquage des interruptions

  ```
  void _mask(int lvl);
  ```

  positionne le registre de statuts du microprocesseur de tel sorte qu'il ne considère plus les interruptions de niveau inférieur au niveau indiqué par l'argument `lvl` ; seule les interruptions de niveau supérieur ou égale à `lvl` seront délivrées. Ainsi après exécution d'un appel à `_mask(1);` toutes les interruptions de niveau 1 à 15 sont (ré-)activées.
- La fonction d'attente d'une interruption

  ```
  void _sleep(int lvl);
  ```

  gèle l'activité du microprocesseur jusqu'à ce qu'une interruption d'un niveau égal ou supérieur à `lvl` soit reçue. A ce moment le microprocesseur reprends l'exécution du code, après l'appel à `_sleep()`. La fonction attachée à l'interruption dans le vecteur d'interruption est exécuté en premier lieu.

2 Gestion des interruptions basées sur le temps

Pour réaliser les opérations de base de programmation du « timer », vous disposez de trois registres matériels : `TIMER_CLOCK` (`0xF0`), `TIMER_PARAM` (`0xF4`), et `TIMER_ALARM` (`0xF8`).

- `TIMER_CLOCK` est un registre accessible en lecture seule ; il donne la date courante depuis le démarrage du matériel, en milliseconde.
- `TIMER_PARAM` est un registre décomposé en 5 champs de bits comme indiqués figure 1. Il permet d'activer et de configurer un mécanisme d'alarme qui déclenchera une interruption de niveau `TIMER_IRQ` (niveau 2).
- `TIMER_ALARM` est un registre qui contient une valeur périodiquement incrémentée lorsque l'alarme est activée. Lorsque ce compteur atteint la valeur 0, il génère une interruption du microprocesseur, puis continue à s'incrémenter.

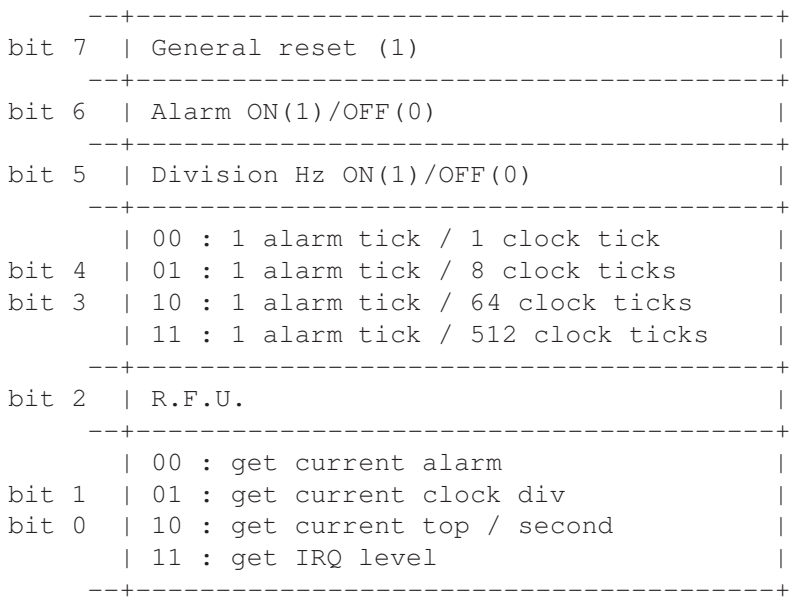

FIGURE 1 – Décomposition du registre `TIMER_PARAM`

Exemple de programmation du timer

L'archive `/home/enseign/ASE/src/hw_tmr.tgz` contient un exemple d'utilisation du timer pour la génération d'une interruption périodique.
Il s'agit
– d'initialiser le matériel ;
– de définir les fonctions associées aux traitants d'interruption, en particulier celle associée à `TIMER_IRQ` ;
– d'armer le timer par une écriture dans le registre `TIMER_PARAM` d'une valeur indiquant :
  – un reset du timer,
  – l'activation du timer,
  – le nombre de tics système par tic timer ;
– de préciser par une écriture dans le registre `TIMER_ALARM` que l'alarme doit survenir au prochain tic ;
– d'activer les interruptions.
À chaque interruption reçue, donc dans la fonction associée au traitant `TIMER-IRQ`, il sera nécessaire de préciser à nouveau le prochain tic auquel doit survenir l'interruption par une écriture ad hoc dans le registre `TIMER_ALARM`.