M1 SESI 2017-2018
Architecture Multi-Processeurs
TP7 : Contrôleur DMA

Kevin Mambu

April 14, 2018

A) Objectifs

Le premier objectif de ce TP est d'analyser le fonctionnement d'un nouveau contrôleur de périphérique: Le composant IOC (Input Output Controler) peut être utilisé pour effectuer des transferts de données entre la mémoire et un périphérique de stockage externe (disque magnétique, clé USB, etc...).

Le second objectif est d'analyser les problèmes posés par le partage des périphériques quand plusieurs programmes s'exécutent en parallèle sur plusieurs processeurs, et utilisent le même périphérique. L'architecture matérielle est donc l'architecture multi-processeurs générique, déjà utilisée dans les TP5, TP6, et TP7.

B) Contrôleur de disque

- Le composant PibusBlockDevice permet d'acceder à un bloc externe.
- LBA : Logic Block Address, numéro de bloc sur un disque.
- 1 bloc = 512 octets.
- Un accès vers un disque est très long → plusieurs millions de cycles.
- Rôle de l'IOC : déclencher un transfert de données entre le tampon mémoire et le disque.
- Emulation d'un disque : fichier sotcké sur le disque de la machine hôte.
- Fichier `images.raw` :
  - 21 fichiers de 128x128 pixels.
  - 1 octet = 1 pixel codé sur 256 niveaux de gris.

Scénario de communication entre `main` et le contrôleur disque

1. `main` configure l'IOC via un appel système.
2. `main` doit écrire dans les registres mappés en mémoire de l'IOC pour lancer le transfert. Il faut spécifier:
   - L'adresse de base du tampon mémoire.
   - Le nombre de blocs à transferer.
   - Le LBA du 1er bloc du disque.
3. L'IOC effectue le transfert bloc par bloc (une transaction rafale par bloc). **Si le transfert est une écriture sur disque, les transactions sont des lectures sur le PIBUS et inversement.** On parle de coprocesseur Entrées/Sorties car le processeur principal et l'IOC travaille en parallèle pendant ce transfert.
4. Lorsque le transfert est terminé, l'IOC lève une interruption pour signaler la fin du transfert au système d'exploitation, qui peut alors **débloquer** l'application ayant demandé le transfert.

Question B1

- `block_size` correspond au nombre d'octets par bloc au sein du Block Device : 128, 256, 512 ou 1024 octets.
- `latency` correspond au temps d'accès mémoire du Block Device.

Question B2

Une image fait 128×128 pixels, soit 16 Ko. Il a été défini qu'un bloc est taille de 512 octets. Une image prend donc sur un Block Device 32 blocs.

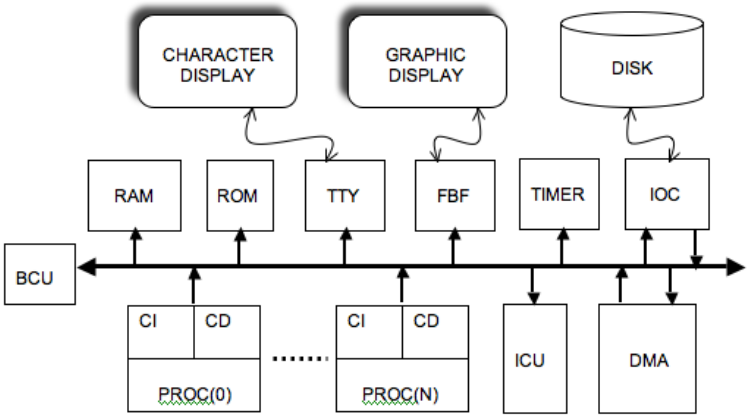
Question B3

Registres adressables de *PibusBlockDevice*		
BLOCK_DEVICE_BUFFER	0x00(RW)	Adresse de base du tampon mémoire.
BLOCK_DEVICE_COUNT	0x04(RW)	Nombre de blocs à transferer.
BLOCK_DEVICE_LBA	0x08(RW)	Numéro du premier bloc sur le disque, e.g. numéro du premier bloc sur le fichier.
BLOCK_DEVICE_OP	0x0C(RW)	Type d'opération à effectuer : • BLOCK_DEVICE_NOOP : Aucune opération. • BLOCK_DEVICE_READ : écriture, du disque vers le tampon mémoire. • BLOCK_DEVICE_WRITE : lecture, du tampon mémoire vers le disque.
BLOCK_DEVICE_STATUS	0x10(RO)	Statut du Block Device, la valeur du statut est défini par l'état de Master FSM : • BLOCK_DEVICE_IDLE : 0 • BLOCK_DEVICE_BUSY : 1 • BLOCK_DEVICE_READ_SUCCESS : 2 • BLOCK_DEVICE_WRITE_SUCCESS : 3 • BLOCK_DEVICE_READ_ERROR : 4 • BLOCK_DEVICE_WRITE_ERROR : 5 Dans tout statut autre qu'IDLE, l'interruption est levée. Une lecture de ce registre remet le statut à IDLE et acquitte l'interruption.
BLOCK_DEVICE_IRQ_ENABLE	0x14(RW)	Active la ligne d'interruption du Block Device si $\neq 0$.
BLOCK_DEVICE_SIZE	0x18(RO)	Nombre de blocs adressables au sein du Block Device
BLOCK_DEVICE_BLOCK_SIZE	0x1C(RO)	Taille du bloc au sein du Block Device en octets.

Question B4

c.f Question B3

C) Architecture matérielle



Larchitecture que nous utiliserons sur ce TP est une à 4 processeurs.

Question C1

L'utilisation du composant PibusBlockDevice impose d'augmenter la valeur du time-out du composant PibusSegBcu à cause de son temps d'accès, celui ci étant de l'ordre du millier, voire du million de cycles. Dans le cas de notre architecture, le tems d'accès est défini par `IOC_LATENCY`, qui est égal à 1000 cycles.

Dans le cas où le time-out n'est pas modifié et garde sa valeur de 100 cycles, tout transfert impliquant l'IOC échouerait. Il faut augmenter le time-out pour le ramener dans l'ordre du millier de cycles.

Question C2

- Le segment `seg_ioc` est à l'adresse `0x92000000` est est de taille 32 octets.
- seg_tty = 64 octets.
- seg_icu = 128 octets.
- seg_tim = 64 octets.

Question C3

- On peut dénombrer 9 composants cibles : bcu, rom, ram, tty, fbf, timer, icu, dma & ioc.
- On peut dénombrer 6 composants maîtres : proc[0..3], dma et ioc.
- L;élement à noter est qu'à l'instar du composant PibusDma, PibusBlockDevice est également un maître et une cible, pouvant recevoir et effectuer des requêtes.

Question C4

- Les composants tty et timer ont une ligne d'interruption par processeur, dma et ioc ont respectivement leur propre ligne d'interruption. Le composant icu recoit donc en entrée 10 lignes d'interruptions :

Entrées IRQ_IN de l'ICU	
IRQ_IN[0]	DMA
IRQ_IN[1]	IOC
IRQ_IN[2]	TIMER[0]
IRQ_IN[3]	TTY[0]
IRQ_IN[4]	TIMER[1]
IRQ_IN[5]	TTY[1]
IRQ_IN[6]	TIMER[2]
IRQ_IN[7]	TTY[2]
IRQ_IN[8]	TIMER[3]
IRQ_IN[9]	TTY[3]

D) Code du boot

Question D1

- Dans le GIET, en l'absence de mémoire virtuelle, l'allocation des processus est faite statiquement. L'adresse de la pile par processeur (i) est alors alors : $seg\_stack[i] = seg\_stack\_base + i * seg\_stack\_size[i]$.

Question D2

Le routage des interruptions entrantes est effectué par le système d'exploitation via le registre ICU_MASK, qui permet de spécifier pour chaque processeur quelles sont les lignes d'interruptions à acquitter.

Question D3

Masques de routages `ICU_MASK[i]`	
IRQ_TIM[0],IRQ_TTY[0],IRQ_DMA, IRQ_IOC ⇒ proc[0]	`0b0000001111`
IRQ_TIM[1],IRQ_TTY[1] ⇒ proc[1]	`0b0000110000`
IRQ_TIM[2],IRQ_TTY[2] ⇒ proc[2]	`0b0011000000`
IRQ_TIM[3],IRQ_TTY[3] ⇒ proc[3]	`0b1100000000`

E) Application logicielle de traitement d'image

Opérations du `main`

- Appel d'ioc_read() : lecture d'une image sur le disque, copie de cette image dans buf_in.
- Appel d'ioc_completed() : attente de la fin du transfert.
- Traitement de l'image pixel par pixel : application d'un seuil sur l'image, stocké dnas un tampon buf_out.
- Appel de fb_sync_write() : affichage sur le framebuffer du tampon de sortie buf_out.

Quesiton E1

La fonction _ioc_read() prend trois arguments :

- `lba` : Adresse du 1er bloc sur le disque.
- `buffer` : Adresse du tampon mémoire où stocker les données lues.

- `count` : Nombre de blocs à transferer.

Cette fonction se bloque en attendant d'avoir accès à l'IOC avec la fonction _ioc_get_lock(). Cette fonction effectue une boucle en assembleur basée sur LL/SC afin de garantir un accès atomique.

Une fois la ressource acquise,déclenche le transfert en écrivant dans le registre BLOCK_DEVICE_OP,

Question E2