M1 SESI 2017-2018
Architecture Multi-Processeurs
TP4 : Partage du Bus dans les architectures multi-processeurs

Kevin Mambu

March 15, 2018

Spécifications

Objectifs :

- Problèmes de performances posés par le partage du bus.
- Goulot d'étranglement lorsque plusieurs processeurs veulent accéder au bus car la bande passante du bus est bornée.
- Temps d'attente du bus proportionnel au nombre de processeurs dans l'architecture.

B) Architecture Matérielle

- Chaque processeur a un TTY attribué
  → N TTYs pour un contrôleur de TTY.
- $|seg\_tty| = NPROC \times 16$. Utilisation d'un framebuffer (256×256) sur 256 niveaux de gris.
- Spécifications du framebuffer
  - Un tampon de luminence, 64Ko
  - Un tampon de chrominance, 64Ko, pas utilisé ici
  - Fréquence de la lecture des tampons : 25 images par seconde
  - Fréquence d'affichage : $f = \frac{1}{40 \times 10^{-3}} = 250Hz$

Question B1

- name : nom de l'instance
- tgtid : index du signal du framebuffer vis-à-vis du PIBUS
- segtab : pointeur vers la table des segments
- latency : temps d'accès au framebuffer
- width : largeur de l'image (en pixels)
- height : hauteur de l'image (en pixels)
- subsampling : fréquence de sous'échantillonage de la chrominance (ici non-utilisé)

Question B2

Le segment associé au Framebuffer doit être aligné et sa taille doit être de $width \times height$ octets (ici $256 \times 256 = 64$Ko).

Dimensionnement des caches

- Caches Write-Through
- Correspondance directe
- 16 lignes de 8 mots

```
#define NPROCS      1 // number of processors
#define FB_NPIXEL  256 // Frame buffer width
#define FB_NLINE   256 // Frame buffer heigth
#define BLOCK_SIZE   512 // IOC block size
#define IOC_LATENCY 1000  // disk latency
#define RAM_LATENCY 0 // ram latency
#define ICACHE_WAYS 1        // instruction cache number of ways
#define ICACHE_SETS 16      // instruction cache number of sets
#define ICACHE_WORDS  8        // instruction cache number of words per line
#define DCACHE_WAYS 1        // data cache number of ways
#define DCACHE_SETS 16      // data cache number of sets
#define DCACHE_WORDS  8        // data cache number of words per line
#define WBUF_DEPTH  8        // cache write buffer depth
#define SNOOP    false // cache snoop activation
#define DMA_BURST 16  // number of words in a DMA burst
```

tp5_top.cpp

C) Compilation de l'application logicielle

Question C1

Le segment associé au contrôleur du Framebuffer fait partie de l'espace privilégié. Tout accès mémoire de l'application utilisateur avec le Framebuffer doit passer par un appel système pour être légal. À cause d'un manque de permissions de la part de l'application utilisateur, un accès direct avec le Framebuffer se traduirait par un Data Bus Error.

Question C2

Écrire ligne par ligne limite le nombre de transactions sur le PIBUS à une seule transaction rafale, une fois le registre intermédiaire plein. Plutôt qu'une transaction par octet, ce qui est plus coûteux.

Question C3

*unsigned int fb_sync_write(unsigned int offset, void *buffer,unsigned int length)*

- offset : le déplacement nécessaire où écrire dans le tampon du Framebuffer.
  *note : sachant qu'on écrit sur le Framebuffer ligne par ligne, l'offset doit être aligné (multiple de 256).
- buffer : addresse de base du tampon intermédiaire
- : longeur en octets de la donnée à écrire sur le framebuffer.

D) Caractérisation de l'application logicielle

Question D1

```
*** proc[0] at cycle 292397
- INSTRUCTIONS        = 205917
- CPI                 = 1.41998
- CACHED READ RATE    = 0.263961
- UNCACHED READ RATE  = 0.00121408
- WRITE RATE          = 0.146399
- IMISS RATE          = 0.00896478
- DMISS RATE          = 0.00943813
- IMISS COST          = 16.0515
- DMISS COST          = 14.386
- UNC COST            = 6
- WRITE COST          = 0
bcu : Statistics
master 0 : n_req = 32754 , n_wait_cycles = 32754 , access time = 1
master 1 : n_req = 0 , n_wait_cycles = 0 , access time = -nan
master 2 : n_req = 0 , n_wait_cycles = 0 , access time = -nan
```

trace.txt.short

Question D2

[TODO]

Question D3

[TODO]

E) Exécution sur architecture multi-processeur

Question E1

```
__attribute__ ((constructor)) void main()
{
    unsigned char   buf[NPIXEL];
    int     n = procid();
    int     nprocs = NB_PROCS;
    unsigned int  line;
    unsigned int  pixel;

    for(line = n ; line < NLINE ; line = line + nprocs)
    {
        for(pixel = 0 ; pixel < NPIXEL ; pixel += 1)
        {
            buf[pixel] = build(pixel, line, 5);
        }
        if (fb_sync_write(256 * line, buf, NPIXEL))
            tty_printf(" !!! wrong transfer to frame buffer for line %d\n", line);
        else
            tty_printf(" - building line %d\n", line);
    }

    tty_printf("\ncycles = %d\n", proctime() );
    exit();

} // end main
```

main.c

Question E2

```
###############################################################################
# File : reset.s
# Author : Alain Greiner
```

```
# Date : 15/12/2011
##############################################################################################
# - It initializes the Status Register (SR)
# - It defines the stack size  and initializes the stack pointer ($29)
# - It initializes the EPC register, and jumps to user code.
##############################################################################################

   .section .reset,"ax",@progbits

   .extern seg_stack_base
   .extern seg_data_base

   .func reset
   .type   reset, %function

reset:
        .set noreorder

# initializes stack pointer
   la   $29,  seg_stack_base
   li   $8, 0x10000     # stack size = 64Kbytes
   mfc0  $27,  $15,  1
   addiu $27,  $27,  1
   mult  $27,  $8
   mflo  $8
   addu  $29,  $29,  $8    # seg_stack_base += 64K

# initializes SR register
        li   $26,  0x0000FF13
        mtc0  $26,  $12     # SR <= 0x0000FF13

# jump to main in user mode
   la   $26,  seg_data_base
        lw   $26,  0($26)      # get the user code entry point
   mtc0  $26,  $14     # write it in EPC register
   eret

   .set reorder

   .endfunc
   .size reset, .-reset
```

reset.s