TME 3+4 : Architecture et Applications de PAPR

Daniela Genius

4 Mars 2005

1 Premiers pas

1. Installer le sytème complet chez vous, en recopiant tout ce que se trouve sous le répertoire suivant :
   /users/enseig/genius/NCPU-fifomwmr-distrib/
2. Pour mettre en place les chemins, mettez vous sous bash et lancez le script source `set-env.sh`.
3. Modifier son PATH afin d'y inclure le répertoire /users/enseig/genius/bin.
4. Enfin pour compiler, lancer `make`.
5. Pour faire tourner l'application, lancer ./simulation_systemcass.x 100000 Le paramètre de la ligne de commande donne le nombre de cycles simulés.

Exercice préliminaire : Compiler et exécuter l'application générique et analyser son comportement à l'aide du 4ème cours et de l'annexe de ce document.

Deux plate-formes matérielles sont proposées :

`simulation_ie.cpp` : plate-forme à une Input Engine et une Output Engine

`simulation_2ie.cpp` : plate-forme à deux Input Engines et deux Output Engines

L'application generique a besoin de `simulation_2ie.cpp`
Le `ldscript` est généré automatiquement.
Choisir un parmi des mini-projets suivants (2 trinomes max. par projet). Le but de ces mini-projets est de constituer un benchmark des programmes de petite et moyenne taille afin d'obtenir des temps de parcours typiques et maximaux des applications sur la plate-forme PAPR.

2 Mini-Projets

Le travail demandé est de modifier `generic.c` et de le remplacer par un fichier qui porte un nouveau nom et qui contient des fonctions utiles selon les projets. UN manuel d'utilisateur détaillé de l'interface matériel MWMR (auteur : Etienne Faure) est donné en annexe.
La distribution de NCPU-fifofwmr contient des fichiers d'entrée `file*.txt` qui contiennent des "faux" paquets IP encapsulés Ethernet. Le premier numéro d'une ligne donne la taille du paquet. Il est recommendé de modifier ces fichiers d'entrée et les remplacer par vos propres fichiers contenant des paquets IP générés.
Conseil : Se servir de Click pour générer automatiquement des paquets IP [1].

P1 : Routeur IPv4 à 2 entrées 2 sorties

Ecrire un routeur IP simple à deux entrées deux sorties.
Utiliser le programme principal modifié `simulation_2ie.cpp` qui définit une plate-forme à deux Input Engines et deux Output Engines. Deux fichiers de sortie out1.txt et out2.txt seront donc à générer.
Voici la fonction proposée pour identifier une route IP. Elle reprend l'élément Click et le traduit en langage C.

```
inline int Lookupiprouter(struct anno_p *anno_p_c)
{
    if (anno_p_c->dst_ip_anno[0] == adr3_ip[0]
```

[1] Il n'est évidemment pas interdit de générer les paquets à la main, mais la conversion en chiffres hexadécimales peut être évitée en utilisant les mécanismes de Click.


```
&& anno_p_c->dst_ip_anno[1] == adr3_ip[1]
&& anno_p_c->dst_ip_anno[2] == adr3_ip[2])
dropbroadcast_2(anno_p_c);

    else if (anno_p_c->dst_ip_anno[0] == adr4_ip[0]
     && anno_p_c->dst_ip_anno[1] == adr4_ip[1]
     && anno_p_c->dst_ip_anno[2] == adr4_ip[2])
dropbroadcast_1(anno_p_c);

    else if (anno_p_c->dst_ip_anno[0] == ip1_adr[0]
     && anno_p_c->dst_ip_anno[1] == ip1_adr[1]
     && anno_p_c->dst_ip_anno[2] == ip1_adr[2]) {

    } else
PRINTF("Unknown destination...\n");
    return 0;
}
```

P2 : Pare-Feu

On considère un Pare-feu rudimentaire à 3 ports tel que présenté en TME 1+2 :
- Une entrée de paquets
- Une sortie de paquets autorisés
- Une sortie de paquets non autorisés

Le but du Pare-feu est de trier le flot entrant en 2 parties : un flot *autorisé* et un *non autorisé*. Pour cela, il existent deux méthodes de filtrages :
- **Liste verte** : Seule les transactions passant les règles de filtrages sont autorisées
- **Liste noire** : Seule les transactions passant les règles de filtrages ne sont pas autorisées

Ecrire un Pare-feu de niveau 3 (l'unité étant des paquets IPs) avec les règles de filtrages présentées dans la sous section suivante, et suivant les deux méthodes de filtrages (Liste verte et Liste Noire).

Conçevoir soi-même des règles de fitrage plus conséquentes.
Générer des différents types de paquets et les mélanger dans un flux commun.

P3 : Mini passerelle

Afin de réaliser une *passerelle*, nous allons préalablement définir quelques simplifications :
- Il s'agit d'une passerelle de niveau 2 (L'unité étant des paquets ethernets)
- Tous paquets arrivent sur une seule entrée
- La passerelle est reliée à deux réseaux locaux
- Chaque réseau à trois stations de travails dont l'adresse IP et l'adresse MAC sont connus de la passerelle
- Ilexiste une adresse de sortie pas défaut où l'on envoye les paquets dont l'adresse ne correspond pas à l'un des réseaux locaux

Réalisez la mini passerelle. celle-ci devra gérer :
- Dèsencapsulation des trames Ethernet entrantes
- Analyses des en-têtes
  - Quel réseau local ?
  - Quelle station de travail dans le sous réseau ?
- Créer un générateur irrégulier de flux afin de tester votre configuration

P4 : Raccord TCP

Traduire les adresses IPv6 en IPv4 selon le schéma de 5-tuples vu en cours 1 :

```
(s6,sp, protocol, d6, data port) -> (s4, sp, protocol, d4, data port)
```

Pour cela, construire un tableau de traduction d'adresses (ATT Address Translation Table) et l'allouer en mémoire sur plusieurs bancs.

P5 : Classification simplifiée

Le but de ce projet est d'implémenter un classifieur rudimentaire. Plusieurs flux sont multiplexés et arrivent sur l'unique ingress (un seul Input Engine) puis sur l'unique fifo MWMR. Il y aura plusieurs tâches de classification qui font toutes la même chose : analyser l'en-tête d'un paquet pour extraire l'information (p.exemple, adresse source). L'appartenance d'un paquet à un certain flux sera reconnue et le paquet renvoié à une des fifo MWMR en sortie.

Prévoir trois classes de service correspondant à trois fifos MWMR à la sortie des tâches de classification. Puis plusieurs tâches d'ordonnancement prennent en charge ces flux (Projet 6).

Les tâches de classification seront au nombre paramétrable (valeur par défaut : 3). Instancier trois fifos MWMR qui seront écrites pas ces tâches. Ne pas modifier la fonction des tâches consommatrices.

P6 : Ordonnancement/QoS

Les tâches d'ordonnancement accèdent un ensemble des Fifos MWMR en lecture. A cettes Fifos seront alloués des Qualités de service (simplification : des fréquences avec lesquelles ces FIFOS seront lues) différentes. Il faudra pourtant éviter la famine.

Implémenter une stratégie simple réalisant trois classes de service statique :

1. Service expresse : choisir donc 7 fois sur 10.
2. Service standard : cette Fifo est choisi 2 fois sur 10.
3. Service minimum : cette Fifo est choisi 1 fois sur 10.

Les tâches d'ordonnancement seront au nombre paramétrable (valeur par défaut : 2). Instancier les trois fifos MWMR à l'entrée des ces tâches. Ne pas modifier la fonction des tâches productrices.

Les paramètres décrivant la Qualité de Service sont à mettre dans le fichier `appli.h` en forme des lignes `#define` pour faire simple.

Remarque : Les projets P5 et P6 peuvent ere combinés mais restent indépendant l'un de l'autre. On utilise l'ordonnancement générique pour P5, la classification générique pour P6.

A MWMR Manuel de l'utilisateur

Ce manuel suppose que le lecteur est familiarisé avec l'utilisation des canaux de communication logiciel fifomwmr. Il s'adresse aux étudiants du cours PAPR qui veulent en savoir plus sur le fonctionnement en détail. Cette partie n'est pas indispensable pour effectuer le travail demandé dans les mini-projets du TME.

A.1 Principes généraux

Le soclib_vci_mwmr est un composant matériel qui implémente les accès faits par un coprocesseur à une fifo mwmr logicielle. Les fifomwmr sont des structures de données localisées dans la mémoire du sytème. Elles permettent à plusieurs producteurs et consommateurs de partager des données. Leur accès est protégé par un verrou, et de ce fait on ne peut avoir à un instant donné qu'une seule tâche (productrice ou consommatrice) accédant à la fifo. Les données stockées dans la fifo sont des items de un ou plusieurs mots de 32 bits de long. Tous les items contenus dans une fifo doivent avoir la même taille. Une lecture ou une écriture dans cette fifo est celle d'un nombre entier d'items. La taille de ces items est un paramètre de construction de la fifo. L'accès à une fifo logicielle est en fait une succession de 5 accès à la structure correspondante :

- prendre le verrou
- tester le statut de la fifo (non pleine ou non vide) et obtenir l'adresse de lecture ou d'écriture
- effectuer la lecture ou l'écriture (n mots de 32 bits)
- mettre à jour statut et pointeur
- rendre le verrou

chacun de ces accès donne lieu à une requète VCI (de deux paquets pour les 2ème et 4ème étapes).
Il est à noter que si la première étape (la prise de verrou) est un échec, le composant MWMR regarde s'il y a une autre requète en attente avant retenter d'obtenir le verrou. Les différents canaux fifo entre le coprocesseur et le MWMR sont testés chacun leur tour. Si aucune autre requête n'est en attente, alors on re-essaye d'obtenir ce verrou après un certain temps d'attente.
Ce comportement permet à plusieurs automates de se partager un seul contrôleur MWMR, puisque si une ressource est bloquée, les autres canaux n'en souffrent pas.
Le temps d'attente après un essai infructueux a une autre raison. Il s'agit d'éviter de saturer l'interconnect de requêtes pour obtenir le verrou.

Chaque composant mwmr peut effectuer des accès à 8 fifos mwmr au maximum (4 en lecture, 4 en écriture). Ces deux valeurs sont des paramètres de construction du composant.

A.2 Interface matérielle

Le composant dispose de deux types de ports :
du coté VCI, il y a un port initiateur et un port cible. Le port initiateur est celui qui effectue les requètes VCI correspondant à un accès à une fifo. Le port cible permet de configurer le composant.
Du coté coprocesseur, le soclib_vci_mwmr dispose d'un nombre variable de ports fifo (8 au maximum). Chacun de ces ports se compose de trois signaux :
Data, read et read_ok un échange ne se produisant que lorsque read et read_ok sont tous les deux à un.
Le signal data à une largeur fixe de 32 bits. Ce qui signifie que lorsque le coprocesseur veut acceder à une fifo contenant des items de n * 32 bits, il doit s'engager à lire où écrire tous les mots d'un item. (n transferts sur le signal data).

A.3 Configuration

Chacun des ports fifo permet la communication avec une fifo, en lecture ou en écriture. A chacun d'eux est associée une série de registres de configuration qui contiennent les données relatives à cette fifo. Cette série se compose de 8 registres :

- `burst` : la taille de l'item (en mots de 32 bits)
- `size` : la taille de la fifo en nombre d'items
- `fifo_ad` : l'adresse du début de la zone de données de la fifo
- `lock_ad` : l'adresse du verrou associé à la fifo
- `status_ad` : l'adresse du statut de la fifo (le nombre d'items contenus à un instant donné)
- `readp_ad` : l'adresse de la variable contenant le nombre d'items lus
- `writep_ad` : l'adresse de la variable contenant le nombre d'items écrits
- `lock` : drapeau indiquant si le composant soclib_vci_mwmr est actuellement en train d'utiliser la fifo concernée. Ce champ sert à empêcher de reconfigurer une fifo pendant qu'elle sert afin d'éviter de ne pas rendre un verrou. Il sert également à ne pas utiliser un port fifo avant que celui-ci ne soit configuré.

Ces registres doivent être configurés par logiciel avant que le coprocesseur ne commence à essayer de faire des accès fifo.
La configuration d'une fifomwmr utilisée par un coprocesseur s'effectue en deux étapes :
La première est l'initialisation de la fifomwmr en mémoire. Cela se fait à l'aide de la primitive `fifomwmrinit`

```
struct fifo *fifomwmrinit (unsigned int pool,
                           unsigned int l,
                           unsigned int p)
```

`pool` : numéro du banc mémoire où est localisée la fifo.
`l` : largeur de la fifo (en mots de 32 bits).

p : profondeur de la fifo, en nombre d'items.
Cette première étape permet d'allouer l'espace mémoire nécessaire, et de l'initialiser.
La fifomwmr peut dès ce moment être utilisée par des tâches logicielles à l'aide des primitives `fifomwmrread` et `fifomwmrwrite`. Pour initialiser le composant mwmr, on dispose de la fonction `mwmrHinit`

```
int mwmrHinit(struct fifo *f, unsigned long adr)
```

Le premier paramètre est un pointeur sur une structure fifo déjà initialisée, le second l'adresse où effectuer la configuration. Les bits de poids fort de l'adresse servent à désigner le composant visé. Le nombre exact de bits utilisés pour cet adressage dépend du système dans lequel on se trouve. Les 12 bits de poids faible de cette adresse sont utilisés par le composant pour déterminer le canal et le registre à configurer.
Les 8 canaux sont désignés par les bits 11-8 de l'adresse. Ce décodage s'effectue comme suit :

Canal	adresse
Canal en lecture #0	0x400
Canal en lecture #1	0x500
Canal en lecture #2	0x600
Canal en lecture #3	0x700
Canal en écriture #0	0x800
Canal en écriture #1	0x900
Canal en écriture #2	0xA00
Canal en écriture #3	0xB00

Ces différentes valeurs sont regroupées dans le fichier `mwmrmap.h,` ce qui permet de manipuler des noms plutôt que des adresses :

```
#define FIFOREAD 0x400
#define FIFOWRITE 0x800


#define FIFO_0 0x0
#define FIFO_1 0x100
#define FIFO_2 0x200
#define FIFO_3 0x300
```

Ainsi, pour configurer la fifo 2 en écriture d'un composant, on construira l'adresse en utilisant FIFOWRITE + FIFO_2.
Chacun de ces canaux dispose de 8 registres de configuration, et à chacun d'eux correspond une valeur particulière du dernier octet de l'adresse. Cette correspondance est illustrée par le tableau suivant :

Registre	Adresse
fifo_ad	0x0
lock_ad	0x04
status_ad	0x08
readp_ad	0x0C
writep_ad	0x10
burst	0x14
lock	0x18
size	0x1C

Ces valeurs sont également contenues dans le fichier `mwmrmap.h.` le registre de status de la fifo 1 en lecture se trouve à l'adresse 0xXXXX0508. Ce qui s'écrit aussi FIFOREAD + FIFO_1 + A_STATUS_AD.

A.4 Exemple d'utilisation

```
#include <fifomwmr.h>
/*pour les primitives d'accès aux fifos */
```



```
#include "mwmrmap.h"
/*Pour utiliser les macros FIFOREAD et consort */

fifo *f; // déclaration de la fifo

unsigned int fw_conf_ad=0x76000000 + FIFOWRITE + FIFO_0;
// Adresse du premier canal fifo en écriture
unsigned int fr_conf_ad=0x76000000 + FIFOREAD  + FIFO_0;
// Adresse du premier canal fifo en lecture

f = fifomwmrinit(0, 12, 22); // initialisation de la fifo:
/* La fifo est située dans le premier banc de mémoire locale, */
/* elle fait 12 octets de large, et 22 items de profondeur    */

mwmrHinit(f, fr_conf_ad);
//Initialisation du premier canal de lecture du soclib_vci_mwmr
mwmrHinit(f, fw_conf_ad);
//Initialisation du premier canal d'écriture du soclib_vci_mwmr
```

Le composant MWMR peut également être utilisé pour configurer le coprocesseur auquel il est connecté. Pour cela il dispose de $N \leq 4$ registres de 32 bits qui servent à la configuration, il dispose aussi de $M \leq 4$ registres de 32 bits dans lesquels le coprocesseur peut venir écrire son état interne. Ce sont des registres de statut.
Chacun de ces registres dispose d'une adresse permettant au logiciel d'y accéder. Les registres de configuration ont leurs adresses en `0x400 + N° de registre`, les registres de status en `0x800 + n° de registre`. Le nombre et la fonction de chacun de ces registres dependent du coprocesseur connecté.