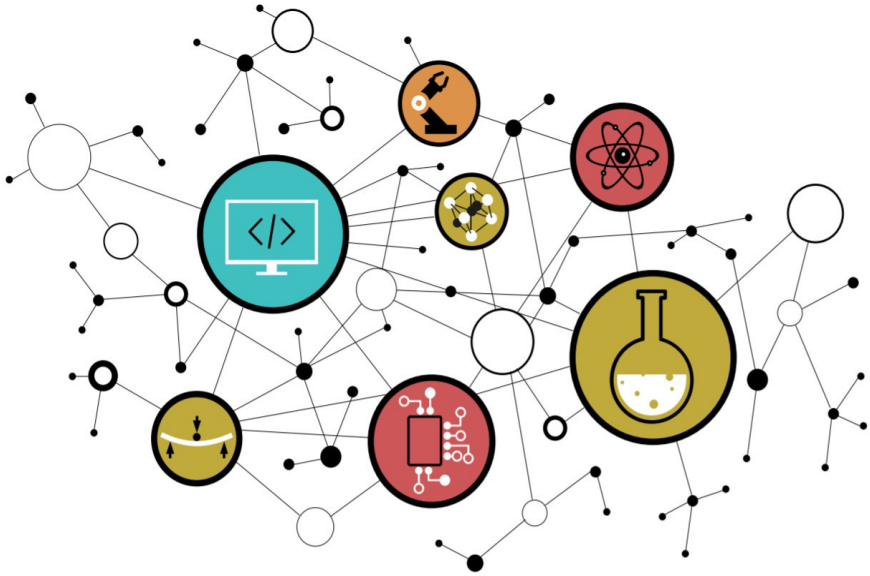
TRAVAUX PRATIQUES

PERIPHERIQUES DE COPIE MÉMOIRE DMA

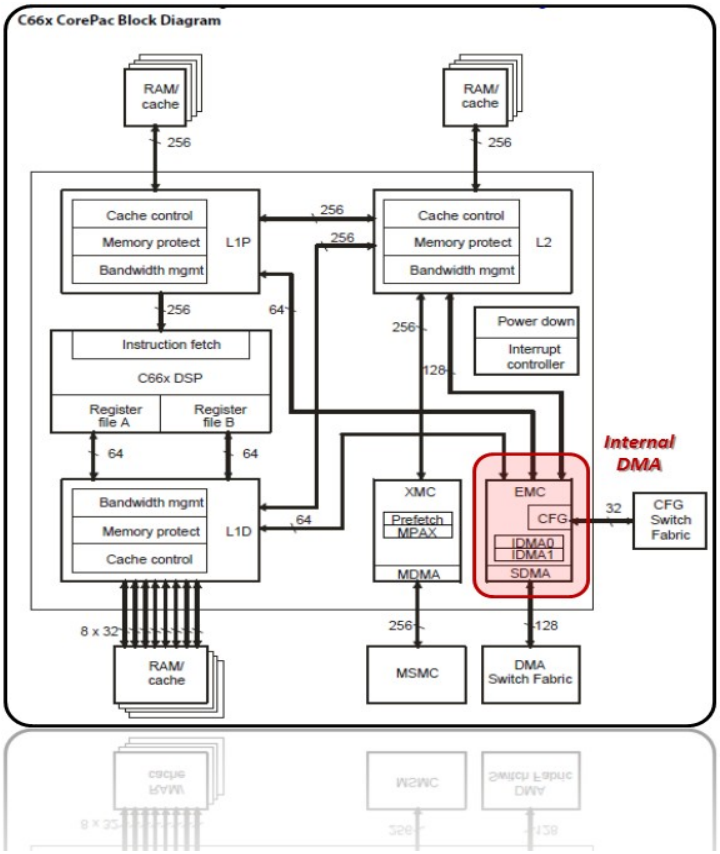


Hugo Descoubes – neku – GNU\Linux - Ubuntu 20.04 LTS – LibreOffice 6.4.6.2 – 2021

SOMMAIRE

5.1. Transferts par IDMA

Dans cette ultime partie, nous allons nous intéresser à des périphériques d'accélération standards sur architectures processeurs évoluées, les DMA (Direct Memory Access). Un DMA est un périphérique de copie d'information de mémoire à mémoire sans passer par le CPU. Un DMA possède ses propres bus et est, après configuration, une entité autonome de l'architecture du processeur. Un DMA classique effectue des copies d'une zone mémoire à une autre et est capable, comme tout périphérique, de prévenir le CPU de la fin d'un transfert par l'envoi d'une interruption matérielle (IRQ ou Interrupt Request). Dans un premier temps, nous travaillerons avec les IMDA (Internal DMA) de notre processeur. Ces IDMA sont présents dans chaque cœur et ne proposent que des services standards de copies (copies linéaires contiguës de mémoire à mémoire L2 SRAM, L1D SRAM ou L1P SRAM) :



Certains DMA plus évolués (exemple des Enhanced DMA chez TI, les Smart DMA chez Freescale, etc) sont capables d'effectuer :

- Des transferts avec modes d'adressages et indexations complexes (manipulation de matrices, de cubes de données, etc)
- De synchroniser des transferts sur événements matériels issus de périphériques
- De capturer et de garder de grands nombres de pré-configurations
- Une étude approfondie de ces types de DMA peut alors prendre des mois

Sur le schéma ci-dessus, nous pouvons observer plus finement l'architecture interne d'un corePac (ensemble CPU/cœur, caches associés et utilitaires matériels spécifiques, exemple des IDMA). Bien faire la distinction entre les mémoires caches (espaces de stockages) et les contrôleurs de caches (DMA autonome travaillant dans notre cas avec des mécanismes de localités temporelles LRU pour le remplacement des lignes de caches). Un contrôleur de cache effectuant des copies de mémoire à mémoire, il n'est donc qu'un DMA autonome.

- Dans le fichier firtest.h, mettre la macro TEST_FIR_L2SRAM_L1DIDMA à 1

```
#define TEST_FIR_ L2SRAM_L1DIDMA   1
```

- Remplacer l'ancien fichier firtest_perf.c par celui présent dans disco/c6678/idma/firtest_perf.c. A l'analysant (cf. ci-dessous), vous constaterez qu'il implémente le même code que celui écrit précédemment (exercice 4.3). Seulement maintenant les copies mémoires de L2 SRAM vers L1D SRAM sont réalisées par périphérique IDMA

```
#if ( TEST_FIR_L2SRAM_L1DIDMA != 0 )
   else if ( memoryModel == UMA_L2SRAM_L1DSRAM ) {
       /* caches levels initializations */
       CACHE_setL2Size(CACHE_32KCACHE);
       CACHE_setL1DSize(CACHE_L1_4KCACHE);
       CACHE_setL1PSize(CACHE_L1_32KCACHE);

       /* prepare coefficients in L1D SRAM */
       memcpy(a_sp_l1d, a_sp, A_LENGTH*sizeof(float32_t));

       start = CSL_tscRead ();

       /* copy part of input array from DDR to L2 */
       for(i=0; i<DDR_ARRAY_LENGTH; i+=L2_ARRAY_LENGTH){

          /* memory copy from DDR to L2 SRAM */
          if( i < (DDR_ARRAY_LENGTH - L2_ARRAY_LENGTH) ){
              memcpy(xk_sp_l2, xk_sp + i, (L2_ARRAY_LENGTH + A_LENGTH - 1)*sizeof(float32_t));
          } else {
              memcpy(xk_sp_l2, xk_sp + i, L2_ARRAY_LENGTH*sizeof(float32_t));
          }

          /* copy part of input array from L2 SRAM to L1D SRAM */
          for(j=0; j<L2_ARRAY_LENGTH; j+=L1D_ARRAY_LENGTH ){

              /* memory copy from L2 SRAM to L1D SRAM */
              idmacpy(xk_sp_l1d, xk_sp_l2 + j, (L1D_ARRAY_LENGTH + A_LENGTH - 1)*sizeof(float32_t));

              (*fir_fct) (xk_sp_l1d, a_sp, yk_sp_l1d, A_LENGTH, L1D_ARRAY_LENGTH);

              /* memory copy from L1D SRAM to L2 SRAM - coherency of output L2 array*/
              idmacpy(yk_sp_l2 + j, yk_sp_l1d, L1D_ARRAY_LENGTH*sizeof(float32_t));
          }

          /* memory copy from DDR to L2 */
          if( i < (YK_LENGTH - L2_ARRAY_LENGTH) ){
              memcpy(output + i, yk_sp_l2, L2_ARRAY_LENGTH*sizeof(float32_t));
          } else {
              memcpy(output + i, yk_sp_l2, (L2_ARRAY_LENGTH - A_LENGTH + 1)*sizeof(float32_t));
          }
       }

       stop = CSL_tscRead ();
       duration += stop-start;
   }
#endif
```

- Ajouter à votre projet la fonction disco/c6678/idmalib/src/idmacopy.c et valider la bonne compilation du projet. Résoudre les erreurs potentielles de compilation.

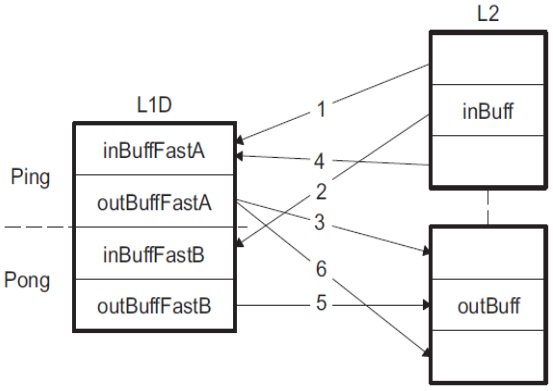
- En vous aidant de la documentation technique de CSL (Chip Support Library ou bibliothèque de fonctions pilotes des DSP C6000) présente dans le répertoire de projet /disco/tp/doc/csl/README.md et de la partie propre à l'IDMA, compléter la fonction idmacopy de façon à remplacer la fonction memcopy précédemment utilisée. *Le programme est très simple, 7 lignes dont une seule ligne de code utile.*

- Après validation en mode Debug, lancer une exécution en mode Release, compiler, tester puis reporter les résultats des tests dans le tableau d'analyse comparative ou Benchmarking présent dans le document de Prélude. Une fois la mesure réalisée, se remettre en configuration initiale en mode Debug.

5.2. Stratégie Ping Pong

Même si la solution précédente reste plus rapide qu'une copie par CPU, elle n'est pas optimale. Notre IDMA possède en réalité 2 canaux de transfert indépendants par cœur. Ces canaux peuvent alors être utilisés en parallèles en manipulant chacun deux vecteurs temporaires de stockage en L1D SRAM. Cette stratégie très générique de copie se nomme **Ping Pong**, lorsque le côté Ping est en cours de traitement (algorithme de traitement du signal), le côté Pong est en cours de chargement/sauvegarde, etc. Rappelons d'ailleurs que nous possédons 24Ko de mémoire L1D SRAM adressable, nous autorisant ainsi à pouvoir définir de nouveaux vecteurs temporaires de stockage. Observons par exemple une séquence de transferts Ping Pong en 6 étapes :

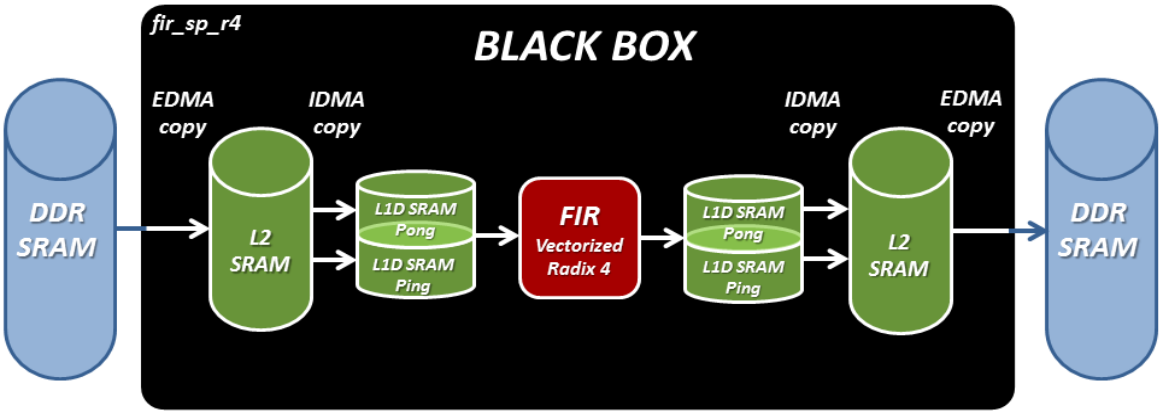
- **Canal 0** : Dédié aux chargements mémoires L2 SRAM vers L1D SRAM
- **Canal 1** : Dédié aux sauvegardes mémoires L1D SRAM vers L2 SRAM



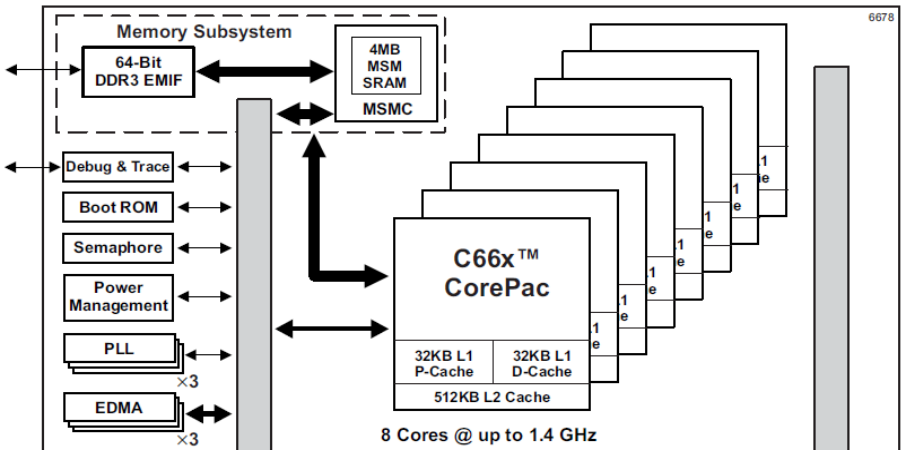
- **Étape 1** : Chargement donnée **xk_sp** du côté Ping de L2 SRAM vers L1D SRAM
- **Étape 2** : Chargement donnée **xk_sp** du côté Pong de L2 SRAM vers L1D SRAM et si chargement du côté Ping terminé, application de l'algorithme de filtrage sur les données précédemment chargées du côté Ping
- **Étape 3** : Si traitement algorithmique terminé, sauvegarde des données de sortie **yk_sp** du côté Ping de L1D SRAM vers L2 SRAM et si chargement du côté Pong terminé, application de l'algorithme de filtrage sur les données précédemment chargées du côté Pong
- **Étape 4** : Chargement donnée **xk_sp** du côté Ping de L2 SRAM vers L1D SRAM
- **Étape 5** : Si traitement algorithmique terminé, sauvegarde des données de sortie **yk_sp** du côté Pong de L1D SRAM vers L2 SRAM
- **Étape 6** : Si traitement algorithmique terminé, sauvegarde des données de sortie **yk_sp** du côté Ping de L1D SRAM vers L2 SRAM

- **Voilà, pour cette année la trame de travaux pratiques s'arrête officiellement ici !** Si vous le souhaitez et si vous avez le temps, vous pouvez tenter d'implémenter la stratégie **Ping Pong**. Bien entendu, cela entraînera un refactoring non négligeable de code.

5.3. Transfert par EDMA



Cette dernière partie dépasse les attentes minimales de la trame d'enseignement et n'est à réaliser que sur volonté propre. Afin de parfaire notre maîtrise de la machine, il resterait également à configurer l'EDMA (DMA système partagé entre cœurs) et enfin pour finaliser l'ensemble, développer un scheduler permettant de paralléliser les copies mémoires (EDMA/IDMA L2SRAM/L1SRAM) des calculs algorithmiques (fir_sp_r4). Il pourrait d'ailleurs s'agir d'un travail de stage et cela nécessiterait des semaines de développement et de test. En revanche, la copie simple (linéaire contiguë) par EDMA peut se réaliser en quelques heures de développement. Voici donc le dernier exercice proposé.



- Ajouter à votre projet la fonction disco/c6678/edmalib/src/edmacopy.c et valider la bonne compilation du projet. Résoudre les erreurs potentielles de compilation.

- En vous aidant de la documentation technique de CSL (Chip Support Library ou bibliothèque de fonctions pilotes des DSP C6000) présente dans le répertoire de projet /disco/tp/doc/csl/README.md et de la partie propre à l'EDMA3, compléter la fonction edmacopy de façon à remplacer la fonction memcopy précédemment utilisée. *Bonne chance !*


notes