Mini-projet « Systèmes Numériques Embarqués Spatiaux »

Introduction

Le projet doit être réalisé par groupe de 3 à 4 étudiants.

Un rapport doit être fourni par groupe au format PDF. Ce rapport doit couvrir les 9 points listés dans la partie « Travail demandé ».

Il faudra fournir en annexe du rapport : le code C développé, les sorties TSIM, les feuilles de calculs Excel éventuellement produites, etc.

Spécifications

On souhaite réaliser le dimensionnement HW/SW de la carte DPU (Data Processing Unit) d'un instrument spatial connectée à une caméra équipée d'un détecteur (Large-area photon-counting sensor) de 14.4 millions de pixels fonctionnant dans l'UV.

Le détecteur est numérisé via 8 ADC 16-bits fonctionnant en parallèle à un débit de 4 millions de pixels par seconde. L'électronique de proximité gérant la chaîne de numérisation contient une RAM limitée à 1 MByte.

Le DPU doit permettre d'enchaîner des « observations », chaque observation consistant à :

- Phase n°1 (durée = 10 * 2 secondes) :
    - Faire l'acquisition de 10 images successives au rythme d'une image toutes les 2 secondes.
    - Accumuler les images au fur et à mesure de leur arrivée (sommation pixel à pixel).
- Phase n°2 (durée = 5 secondes) :
    - Compresser l'image accumulée résultante (somme pixel à pixel des 10 images) via un algorithme de compression sans perte de données (utiliser le code C du compresseur « Rice » fourni pour l'estimation).
    - Transmettre l'image compressée vers le module de service du SpaceCraft.
        - La taille maximum des paquets transférés est de 65535 bytes.
        - Les traitements consistent à « découper » l'image compressée en paquets de 65535 bytes : chaque paquet est copié vers une FIFO de transmission alimentant un driver SpaceWire.

Le DPU doit être capable de démarrer une nouvelle « observation » toutes les 25 secondes.

Le lien de communication entre le DPU et le module de service est un lien SpaceWire fonctionnant à 100 Mbps.

Travail demandé

On demande de :

1. (3 points) - Etablir le bilan du flot de données en entrée du DPU (temps de transfert d'une image complète, volumes transférés, débits) et proposer un concept d'interface entre le DPU et la caméra.
2. (3 points) - Prototyper en C les différents algorithmes.
3. (2 points) - Mesurer, à l'aide du simulateur TSIM, les temps d'exécution sur cible LEON des différents traitements.
4. (2 points) - A partir de ces mesures, établir un budget CPU (taux d'occupation CPU) en fonction de la fréquence du processeur et du nombre de cœurs. Ce budget CPU doit être découpé selon les 2 phases, chacune des phases identifiant une cycle temps réel distinct (durée cycle phase n°1 = 2 secondes ; durée cycle phase n°2 = 5 secondes).
5. (2 points) - Etablir un budget mémoire.
6. (2 points) - Identifier les composants de la carte
    a. Processeur (type , fréquence)
    b. Mémoires (type, référence, nombre de boîtiers)
    c. FPGA si nécessaire
    d. Drivers LVDS
    e. Etc.
7. (3 points) - Proposer une architecture de la carte DPU
    a. réaliser un schéma synoptique mettant en évidences les composants, les interfaces externes et les interface internes,
    b. donner une description textuelle.
8. (2 points) - Estimez la puissance consommée (en mW).
9. (1 points) - Références des documents utilisés (datasheets des composants, manuels utilisateurs, etc.).

Note 1 : Prototypage des algorithmes et mesure des temps d'exécution

Acquisition

Pour la réception de l'image, si l'on part de l'hypothèse d'un transfert DMA des pixels par paquet, alors il faut évaluer le temps d'exécution de la routine qui après chaque réception de paquet réarme la réception.

Réarmer la réception, c'est généralement écrire dans un descripteur DMA (un registre) l'adresse à laquelle devra être copié le prochain paquet et positionner un bit d'activation de la réception dans un second registre de l'IP core gérant le lien.

En plus du temps d'exécution de cette routine (qui doit être multiplié par le nombre de fois où elle devra être appelée pour le transfert d'un image, ce qui dépend de la taille du paquet), il faudrait aussi idéalement mesurer l'overhead lié à l'interruption, car la réception de chaque paquet va produire une interruption.

Accumulation

Pour l'accumulation, il ne s'agit pas de seulement copier l'image dans un tableau. Il faut sommer pixel à pixel l'image qui vient d'être reçue avec une image accumulée.

Il faut donc prévoir une zone mémoire (tableau R) où les images sont écrites par l'IP core lors des transferts DMA, puis une deuxième zone mémoire (tableau A) où l'on va stocker l'image accumulée. Le traitement consiste donc à réaliser la somme de chaque pixel du tableau R avec son homologue accumulé dans le deuxième tableau A : A[i] = A[i] + R[i] où i varie de 0 à nombre de pixels moins 1.

Toutes les 10 images, il faut en plus remettre à 0 A[i].

Compression

Pour la compression, on peut se contenter d'utiliser les algorithmes tels qu'ils sont proposés (d'abord le preprocess, puis le compress).

On peut aussi réécrire une fonction preprocess qui travaillerait avec une image de référence (différence pixel à pixel de l'image à compresser avec cette image de référence). La fonction preprocess fournie fait la différence de 2 pixels successifs au sein de la même image, ce qui n'est pas le plus efficace pour la compression.

Transmission

Pour la transmission de l'image compressée, le traitement consiste à « découper » l'image compressée en paquets de 65535 bytes (un header Spacewire + un second header au format CCSDS doit être ajouté) : chaque paquet est copié vers une FIFO de transmission alimentant un driver SpaceWire. La transmission elle-même (réalisée par le driver SpaceWire) nécessite, comme pour la réception :

- d'écrire dans un descripteur DMA (un registre) l'adresse où l'IP core devra aller chercher le prochain paquet à transmettre
- de positionner un bit d'activation de la transmission dans un second registre de l'IP core gérant le lien

Il faudrait prendre aussi en compte l'overhead lié à l'interruption de fin de transmission du paquet.

Note 2 : Transfert DMA et évaluation de la charge CPU

Le DMA est avant tout un mécanisme hardware permettant de transférer des données d'un périphérique vers la mémoire du processeur sans que le CPU n'intervienne.

Au niveau logiciel, il faut seulement configurer le transfert DMA et cela dépendra de l'interface de configuration de l'IP core en charge du périphérique.

Pour le mini-projet, il faut faire une évaluation très simple, non dépendante d'un IP core précis : configurer un transfert DMA pour une réception de paquet, c'est généralement écrire dans un descripteur DMA (un registre) l'adresse à laquelle devra être copié le prochain paquet et positionner un bit d'activation de la réception dans un second registre de l'IP core gérant le lien. C'est donc extrêmement simple. Il s'agit de faire ici un dimensionnement approximatif et pas de développer une vraie application utilisant de "vrais" transferts DMA.

Si vous voulez voir comment fonctionne un IP Core utilisant des transferts DMA, je vous invite à regarder le chapitre 16 (Spacewire interface) du document https://www.gaisler.com/doc/gr712rc-usermanual.pdf

Supposons que votre image fasse 1000 pixels et que vous décidiez de transférer les pixels par paquets de 100 pixels.

Cela veut dire qu'il y aura transmission en tout de 10 paquets.

On souhaite que l'image soit transférée dans un tableau : uint16_t image[1000];

Avant la transmission de chaque paquet, il faut configurer l'IP core pour donner l'adresse à laquelle devront être copiés les pixels durant le transfert DMA.

Supposons que l'IP core ait un registre appelé rx_address permettant de configurer l'adresse de réception.

Supposons que l'IP core ait un second registre appelé control_reg permettant d'activer la transmission.

Avant chaque transmission de paquet, le logiciel devra appeler une fonction que l'on peut appeler arm_reception(uint16_t* address) :

```
void arm_reception( uint32_t* address) {
  rx_address = address;
  control_reg = 1; // transmission enabled
}
```

Pour modéliser l'acquisition, on peut prototyper une fonction test_acquisition() comme ci-dessous :

```
void test_acquisition() {
  uint16_t* image_ptr = image;
  for (int i=0; i< 10; i++) {
     arm_reception((uint32_t*)image_ptr);
     image_ptr +=100;
  }
}
```

Ce qui compte avec cette modélisation, ce n'est pas de faire quelque chose de fonctionnel, mais d'évaluer les temps d'exécution et la charge CPU.

Pour avoir une modélisation un peu plus fine, il faut aussi prendre en compte l'overhead lié aux interruptions. Une interruption est déclenchée pour chaque paquet reçu.

Pour mesurer cela, vous pouvez déclarer un handler d'interruption que vous déclenchez n fois dans une boucle. Vous mesurez le temps d'exécution des n déclenchements et vous divisez par n, cela vous donne l'overhead pour 1 paquet.