FHT

Ce fichier en haute qualité: ?soclib_fht_specifications_v1.0.pdf

1) Fonctionnalités du bloc

Le module FHT calcule la transformé de Hartley rapide (Fast Hartley Transform) sur une série de données d'entrée complexes (I et Q). Ces données arrivent à la FHT à travers deux FIFOs d'entrée, chacune avec les valeurs complexes (I et Q).

2) Architecture du bloc

L'architecture du bloc FHT est composée de deux modules, le c?ur et le wrapper. Le c?ur effectue tous les traitements FHT. Le wrapper sert à fournir une interface compatible SoClib CABA avec le module MWMR_controller, disponible dans la librairie SoClib.

2.1) Interface du bloc FHT

L'interface proposé est compatible avec le module MWMR_controller sans ajouter des modifications ni adaptations.

Le module FHT dispose des interfaces suivantes:

- Deux FIFOs d'entrée de données.
- Une FIFO de sortie de données.
- Une FIFO d'entrée de configuration.
- Six registres de configuration (seulement en écriture).
- Cinq registres de status (seulement en lecture).

Les deux FIFO de données et celle de configuration sont du même type. Vis-à-vis du MWMR_controller, on peut considérer que le bloc FHT dispose de trois FIFO d'entrée et une de sortie.

2.2) Architecture de l'unité FHT

Ci dessous l'architecture de l'unité FHT, elle est composé par:

- User mux: multiplexeur d'entrée de donnés à partir de deux FIFOs
- FHT Core: c?ur du calcul FHT
- FIFO out: FIFO de sortie des donnés calculées

Le format de donnés d'entrée complexe a ce format : la partie I (réelle) est sur les 16 bits poids forts et la partie Q (imaginaire) sur les 16 bits poids faibles.

2.2.1) User mux

Le User Multiplexor construit les symboles d?entrées pour la FHT (8, 16 ou 32 complexes) à partir de 2 flots d?entrée. L?origine de chaque complexe est définie par un jeu de 2 masques de 32 bits. Pour chaque bit (i) du jeux des masques, on détermine si la donnée provient de la FIFO 0, FIFO 1 ou elle est nulle.

Mask1(i)	Mask0(i)	Data i
0	0	null
0	1	fifo_in 0
1	0	fifo_in 1
1	1	interdit

2.2.2) FHT

Cette Unité peut exécuter des FHTs sur 8, 16 ou 32 points. La FHT travaille sur des formats internes de scalaires de 20 bits alors qu?en entrée et en sortie, la taille est de 16 bits. Une saturation éventuelle est détectée en sortie sur chacun des parties I et Q. Pour la FHT en Rx, le résultat doit être divisé par le nombre de points (puissance de 2) au moyen d?un décalage droit programmable. Cet operateur se retrouve dans toutes les unités FHT (même unité pour le Tx et le Rx)

2.2.3) Détection de saturation

Toutes les détections de saturation sont comptées dans un seul compteur de 16 bits, I et Q des 2 opérations confondues. Ce compteur est mis à 0 au Reset. Une option de configuration permet de reseter aussi le compteur au démarrage d?une nouvelle configuration.

Lorsque le compteur passe de 0 à une valeur non nulle (première détection d?une saturation). Le compteur continue ensuite à accumuler les saturations. Si le compteur arrive à la valeur maximum (64K) , il reste bloqué à cette valeur.

Le compteur est remis à 0 lorsque le CPU vient dumper le registre.

2.3) Anoc_copro_wrapper

Les interfaces de communication entre les coprocesseurs développés au CEA-Leti ne sont pas directement compatibles avec les interfaces CABA proposés par SoClib. Pour permettre de faire une conversion de protocole en gardant une fonctionnalité équivalente, un wrapper à été développé.

L'interface fournie par le module MWMR controller a certaines limitations. Par exemple, le contenu des registres configuration ne peut pas être modifié par le coprocesseur. En outre, si le coprocesseur fait une copie locale de ce registre, il ne peut pas savoir quand le contenu de ces registres été mise à jour.

Pour résoudre ce problème, nous avons conçu un wrapper qui détecte le changement du contenu des registres de configuration pour détecter un changement dans ces derniers.

Pour écrire dans un registre de configuration et s'assurer que la valeur à été correctement écrite, on doit faire deux écritures avec des valeurs différentes. Une première avec une valeur quelconque et puis une deuxième avec la bonne valeur. De ce fait, le wrapper détecte un changement de la valeur et met à jour la valeur du registre interne.

2.3.1) Registres de configuration

Les registres ici détaillés correspondent aux registres de configuration accessible à travers du bloc MWMR.

Le wrapper développé a été conçu pour pouvoir supporter plusieurs coprocesseurs avec plusieurs c?urs de calcul. Par contre, le bloc FHT contient uniquement un c?ur de calcul, donc tous les registres pour les c?urs 1 à 3 ne sont pas utilisés.

Registre de configuration du MWMR	Description
@0	Execute: Début d'exécution. Bits 0 à 3, un bit par coeur
@1	SlotID pour le coeur 0
@2	SlotID pour le coeur 1 (non utilisé)
@3	SlotID pour le coeur 2 (non utilisé)
@4	SlotID pour le coeur 3 (non utilisé)
@5	Adresse de configuration: Définit à quelle adresse débute la configuration du coeur

2.3.2) Registres de status

Ces registres sont accessibles uniquement en lecture.

Registre de Status du MWMR	Description
@0	EOC (End of Compute). Bits 0 à 3 (un bit par coeur)
@1	Status du coeur 0
@2	Status du coeur 1 (non utilisé)
@3	Status du coeur 2 (non utilisé)
@4	Status du coeur 3 (non utilisé)

3) Configuration du c?ur

adresse	nom du registre	bloc dest	contenus
128	fht_conf	core fht	configuration du Core
129	user_mask0	user mux	masque de l'utilisateur 0
130	user_mask1	user mux	masque de l'utilisateur 1

format du registre fht_conf

nom	champ	fonction
cfg_sat_rst_at_load	17	le compteur de saturation est remis à 0 à chaque load_cfg
cfg_nb_buf	16:5	nombre de buffers à traiter -1
cfg_nb_shift	4:2	nombre de positions de décalage
cfg_fht_sz	1:0	codage de la taille de la FHT

cfg_fht_sz code la taille effective de la FHT sur 2 bits :

nombre de points de la FHT	8	16	32
cfg_fht_sz	0	1	2

4) Envoi de la configuration du c?ur à travers de MWMR

Les configurations du c?ur ne sont pas visibles directement dans le memory map du système. On ne peut pas faire une lecture/écriture à une adresse pour obtenir la valeur à configurer/lire. Pour configurer ces registres, il faut passer par un mécanisme d'indirection mise en place par l'association MWMR + wrapper du FHT. Ce mécanisme

permet uniquement de faire des écritures de configuration, on ne peut pas faire de lecture.

Pour charger les configurations au c?ur FHT on utilise une FIFO (FIFO d'entrée de configuration) et un registre de configuration (Adresse de configuration: registre @5) de MWMR. La procédure est la suivante:

- On écrit dans le registre @5 l'adresse de début à configurer.
- On écrit dans la FIFO les données de configuration de maniére séquentielle.

Par exemple, si on veut configurer les adresses 10 à 25, on écrit 10 dans le registre @5 et puis on écrit les 15 valeurs de configuration dans la FIFO.

Avant d'envoyer des données au c?ur, on doit s'assurer que tous les paramètres de configuration on été reçus. On doit utiliser la fonction 'mwmr_wait_fifo_empty' pour s'en assurer.

5) Chargement des données dans le c?ur

Le module FHT travaille avec des paquets de données, par exemple pour une FHT de 32 symboles, le paquet de données sont les 32 valeurs nécessaires pour réaliser la FHT.

Pour que le module FHT puisse travailler avec un nouveau paquet de données il faut:

1. Ecrire dans le registre SlotID (registre @1) du c?ur le numéro de configuration à exécuter.
2. Reseter à 0 le registre Excute (registre @0).
3. Ecrire 1 au registre Execute pour démarrer une nouvelle exécution au module.
4. Charger les données du module à travers les FIFOs.
5. Attendre la fin du chargement avant de charger des nouvelles données. Pour cella, faire du polling sur le registre de EOC (End of Compute, registre status @0) et attendre la valeur 1.
6. On peut lire la valeur de Status du c?ur 0 (registre status @1) pour savoir si tout est bien passé. La valeur 0 signifie que tout s'est bien passé.

On peut charger un nouveau paquet de données dès que le registre de EOC du paquet précédent vaut 1. Autrement on pourrait écraser l'exécution courante.

6) Démonstrateur du bloc FHT

Le démonstrateur du bloc FHT est constitué de :

- Un processeur MIPS
- Trois bancs mémoire
  - ♦ Mémoire du programme
  - ♦ Mémoire données
  - ♦ Mémoire des objets logiciels MWMR (non caché)
- Une mémoire de locks
- Un TTY
- Un MWMR connecté au module FHT

Tout le code nécessaire pour faire fonctionner le bloc FHT est embarqué dans le MIPS. Nous avons crée une API (api_fht.h) pour permettre une programmation simple de ce bloc. De même, nous avons crée une API (api_mwmr.h et api_mwmr.cpp) pour le MWMR qui intègre toutes les fonctionnalités requises par la API du FHT.

6.1) Organisation des répertoires

Le répertoire racine du démonstrateur est:

soclib/platform/topcells/caba-vgmn-fht-mipsel

Le répertoire du module FHT est:

soclib/module/streaming_component/fht

Ce répertoire contient deux sous-répertoires:

- soft: contient le logiciel embarqué et les apis de fht et mwmr
- appli/trace: contient des informations internes du module fht. Indispensable pour pouvoir utiliser le module.

6.2) Objets logiciels pour le FHT

Les objets logiciels utilisés par le MWMR sont trois:

- FIFO logicielle
- Etat de la FIFO logicielle
- Locks

Nous avons séparé les objets FIFO logiciel et son état dans une mémoire (RAM MWMR) pour pouvoir définir cette mémoire comme non cachée. Autrement, il ne serait pas possible d'utiliser le MWMR correctement.

Pour pouvoir utiliser le bloc FHT il faut configurer tous les objets logiciels pour le MWMR. Ces objets sont:

- FIFO logicielle
- Etat de la FIFO logicielle
- Lock de la FIFO logicielle

On doit configurer tous ces objets pour chacune des FIFOs utilisés: 2 FIFOs d'entrée, une FIFO de sortie et une FIFO de configuration.