trx_ofdm

1) Fonctionnalités du bloc

Le bloc trx_ofdm est un bloc spécialisé dans le calcul de transformée de fourrier rapide directe (FFT) et inverse (IFFT). Il intègre par ailleurs des fonctionnalités permettant de réaliser une mise en forme des symboles OFDM entrant (framing - insertion des pilotes et des zéros) ainsi qu?une séparation des pilotes et data des symboles sortant (deframing), ce qui lui permet d?être utilisé soit en émission OFDM (tx : framing + IFFT + insertion d?intervalle de garde) soit en réception OFDM (rx : FFT + deframing)

2) Architecture du bloc

L'architecture du bloc trx_ofdm est composée de deux modules, le c?ur et le wrapper. Le c?ur effectue tous les traitements FFT et IFFT. 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 TRX_OFDM

L'interface proposé est compatible avec le module MWMR_controller sans ajouter des modifications ni adaptations. Un rappel du bloc MWMR ainsi que du protocole FIFO est détaillé dans la Section 7.

Le module OFDM dispose des interfaces suivantes:

- Deux FIFOs d'entrée de données.
- Deux FIFOs 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 TRX_OFDM dispose de trois FIFO d'entrée et deux de sortie.

2.2) Le c?ur TRX_OFDM

Le c?ur TRX_OFDM calcule des symboles OFDM, et nécessite en entrée du calcul que les blocs complets (de 32 à 2048 mots) soient présents. Pour des raisons de performance du c?ur, et afin de ne pas pénaliser les temps de calcul par les temps de communication, une solution d?implémentation en double buffering apparaît donc nécessaire. Par ailleurs, les chargements et déchargements des buffers sont configurables.

Le choix fait pour le modèle du c?ur OFDM est donc de séparer clairement les 3 phases de chargement/calcul/déchargement par un pipeline d?exécution. Le traitement d?un symbole OFDM aura donc ces trois phases configurables. Cela signifie également que 3 symboles OFDM pourront coexister dans le c?ur répartis dans chacune des 3 phases. La Figure 2 illustre ce fonctionnement.

Le c?ur trx_ofdm peut réaliser des opérations de Framing et de Deframing à des flux de données pour séparer les données et les pilotes. Pour cela, le c?ur dispose de deux FIFOs en entrée (FIFO_in 0 et 1) et deux FIFOs en sortie (FIFO_out 0 et 1).

Les opérations possibles sont:

- A partir d'un flux de données (FIFO_in 0) on peut faire une FFT et une opération de Deframing pour obtenir des données (FIFO_out 0) et des pilotes (FIFO_out 1).
- A partir d'un flux de données (FIFO_in 0) et d'un flux de pilotes (FIFO_in 1), on peut faire un Framing et ensuite une IFFT pour obtenir un flux de données (FIFO_out 0).
- A partir d'un flux (FIFO_in 0) qui contient des données et puis des pilotes, on peut faire une opération de Framing et ensuite une IFFT pour obtenir un flux de données (FIFO_out 0)

L'utilisation des opérations de Framing et Deframing nécessite la configuration des masques de données et pilotes (voir Paramètres de configuration).

2.3) Anoc_copro_wrapper OFDM

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 autre, 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 (voir Section 7 pou plus de détail).

Le wrapper développé à été conçu pour pouvoir supporter plusieurs coprocesseurs avec plusieurs c?urs de calcul. Par contre, le bloc TRX_OFDM 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

3.1) Liste des paramètres

Les paramètres configurables du bloc trx_ofdm sont les suivants :

Paramètre	min val	max val	Description
BYPASS_FFT	0	1	Si 1 bypass de la fonction FFT (mode de test)
FFT_TYPE	0	1	0 = FRAMING+IFFT, 1 = FFT+DEFRAMING
LOG2_SIZE_FFT	5	11	Pour size_fft = 32,64,128,256,512,1024,2048
NORMALIZE POWER FFT	0	1	Si 1, division par sqr(n) après le calcul de FFT/IFFT (n = size_fft)
GI_INSERTION	0	1	Si 1 intervalle de garde inséré
GI_SIZE	0	2047	Taille Interval de garde (pour IFFT uniquement)
SHIFT_CARRIER	0	1	Si 1, décalage des porteuses
SHIFT_PARITY	0	1	Si 1, inversion des porteuses paires/impaires
FLOC	0	2	Type de framing: 0: dp1fifo, data et pilot dans la même FIFO 0. 1: dp2fifo, data FIFO 0, pilotes FIFO 1
MASKIT	0	1	Si 1, IT non masquée
MASK_DATA	64*32 bits		Masque framing ou deframing
MASK_PILOT	64*32 bits		Masque framing ou deframing

3.2) Adresses des paramètres

Le c?ur trx_ofdm peut contenir 3 configurations différentes, chacune rangé dans un SlotID différent. Ces SlotID sont des types de traitement que nous avons enregistré dans le c?ur et que nous pouvons utiliser pour effectuer un traitement (FFT, IFFT, ...). Quand nous voulons effectuer un traitement, nous indiquons le SlotId à utiliser et ensuite nous envoyons les données. Etant donnée que le c?ur trx_ofdm est en pipeline, nous pouvons démarrer un traitement avec une SlotId et ensuite démarrer un nouveau traitement avec une autre SlotId.

Attention, il est interdit de changer la configuration d'un SlotID si celui-ci est en cours d'utilisation par le pipeline.

Les plages d'adresses des SlotId sont:

SlotId 1 0x000 => 0x084
SlotId 2 0x100 => 0x184
SlotId 3 0x200 => 0x284

Les adresses de configuration sont relatives à l'adresse du SlotId.

Relative Address (slotid)	Name	Content
0x0	MASK_DATA_0	mask_data 0 to 64 (32 bits)
...	...	
0x3F	MASK_DATA_64	
0x40	MASK_PILOT_0	mask_pilot 0 to 64
...	...	
0x7F	MASK_PILOT_64	
0x80	FFT_CFG	bypass_fft, fft_type, log2_size_fft, normalize_power_fft, shift_carrier, shift_parity
0x81	GI_CFG	gi_insertion, gi_size
0x82	FRAMING_CFG	floc
0x83	IT_CFG	mask_IT

Le détail des champs de configuration est le suivant:

3.3) Flot des données OFDM

Le flot de données OFDM qui traite le bloc trx_ofdm est composé de données et des pilotes. Les données et les pilotes sont des valeurs complexes composées d'une partie réelle (I) et d'une partie imaginaire (Q). Chaque partie est encodée en 16bits signé et l'ensemble de I+Q forme une valeur du flot de données de 32 bits.

Quand on envoie un flot de donnés au TRX_OFDM, la fréquence centrale se situe au milieu du flot de donnés.

3.4) Masques de données et pilotes

Les masques de données et pilotes sont utilisés pour réaliser les opérations de Framing et Deframing. Un masque de données/pilote contient 2048 bits (étalé en 64 mots de 32bits), un bit par donnée de FFT/IFFT. Si le bit vaut 1, une valeur de donnée/pilote sera prise en considération. Le comportement des masques est différent en fonction si sont utilisés dans le Framing ou le Deframing.

3.4.1) Framing

Dans le module de Framing, on examine les bits à 1 des masques pour savoir dans quelle position on doit ranger les données et les pilotes. Le résultat est un vecteur qui combine les valeurs de données, des pilotes et des 0. Les positions de ce vecteur qui ne continent ni données ni pilotes sont initialisés à 0.

La Figure 3 montre un exemple de Framing avec deux FIFOs d'entrée. La combinaison des données et pilotes obtient les données à envoyer à la IFFT.

Le nombre de 1 dans le masque de données indique le nombre de données qui seront utilisés dans la FIFO de données. De la même manière, le nombre de 1 dans le masque de pilotes indique le nombre de pilotes qui seront lus de la FIFO de pilotes.

La Figure 4 montre un exemple de Framing avec une FIFO d'entrée qui contient les données et ensuite les pilotes. Le nombre de 1 dans le masque de données et masque de pilotes, indique le nombre total de données qui seront utilisés dans le FIFO d'entrée.

3.4.2) Deframing

Le vecteur de sortie de la FFT est analysé pour extraire les valeurs de données et les pilotes. La position des 1 dans les masques indique la position de la valeur à extraire. Si la masque de données et la masque de pilotes contient 0 pour une même adresse, la valeur du vecteur de sortie de la FFT à cette adresse n'est pas utilisé.

Le Deframing permet de séparer les données et les pilotes dans des FIFOs différentes après la FFT. La figure suivante montre un exemple de Deframing avec des masques de données et pilotes.

Le nombre de 1 dans le masque de données indique le nombre de données qui seront extraites. De la même manière, le nombre de 1 dans le masque de pilotes indique le nombre de pilotes qui seront extraites.

3.5) Intervalle de garde

En sortie de l?IFFT seulement, un intervalle dit « de garde » peut être ajouté au symbole OFDM. Le paramètre gi_insertion à 1 indique que cet intervalle de garde sera appliqué. Dans ce cas, le paramètre gi_size est utilisé pour compter le nombre d?échantillons à ajouter AVANT le symbole OFDM. Le premier échantillon est récupéré à l?emplacement (taille_fft-gi_size). Les échantillons suivants sont récupérés séquentiellement jusqu?au dernier échantillon du symbole OFDM. Le symbole OFDM complet est ensuite envoyé. En sortie, nous aurons donc size_fft + gi_size échantillons de 32 bits.

3.6) Shift carrier et shift parity

Deux paramètres supplémentaires permettent de modifier les valeurs des échantillons en entrée de l?IFFT seulement.

- Shift carrier à 1 défini que les échantillons vont être modifiés
- Dans ce cas :
    - si shift parity vaut 0, la valeur des échantillons impairs sont inversés (I et Q indépendamment sur 16 bits signés).
    - si shift parity vaut 1, la valeur des échantillons pairs sont inversés (I et Q indépendamment sur 16 bits signés).

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 le wrapper du trx_ofdm. Ce mécanisme permet uniquement de faire des écritures de configuration, on ne peut pas faire de lecture.

Pour charger les configurations au c?ur TRX_OFDM 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 maniérer 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 OFDM travaille avec des paquets de données, par exemple pour une FFT de 1024 données, le paquet de données sont les 1024 valeurs nécessaires pour réaliser la FFT.

Pour que le module OFDM 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 au module à travers des 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 c'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 TRX_OFDM

Le démonstrateur du bloc TRX_OFDM 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 TRX_OFDM

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 possible d'utiliser le MWMR correctement.

Tout le code nécessaire pour faire fonctionner le bloc TRX_OFDM est embarqué dans le MIPS. Nous avons crée une API (api_trx_ofdm.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 TRX_OFDM.

6.1) Organisation des répertoires

Le répertoire racine du démonstrateur est:

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

Ce répertoire contient deux sous-répertoires:

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

6.2) Objets logiciels pour le MWMR

Pour pouvoir utiliser le bloc TRX_OFDM 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, 2 FIFOs de sortie et une FIFO de configuration.

6.3) Application du démonstrateur

A finir...

7) Rappel du MWMR

La mise à jour par le projet SoClib pour le bloc MWMR se trouve:

?https://www.soclib.fr/trac/dev/wiki/Component/VciMwmrController

Ce module a été conçu suite à la thèse d'Etienne Faure:

"Communications matérielles-logicielles dans les systèmes sur puce orientés télécommunications"

?ftp://asim.lip6.fr/pub/reports/2007/th.lip6.2007.faure.1.pdf