The level 2 trigger of HESS 2

Dominique BESIN
Denis CALVET
Christophe COQUELET
Jean-François GLICENSTEIN
Yassir MOUDDEN
Philippe VENAULT
Hamid ZAGHIA

Technical Report - *DRAFT*

Contents

1 Slow control and Configuration

1.1 Introduction

Dans les sections précédentes nous nous sommes concentrés sur le pipeline principal du trigger L2, c'est à dire la chaîne de traitement permettant de passer des données reçues pour un évènement à la réponse L2A ou L2R du trigger. Nous indiquions néanmoins que ce pipeline était paramètré et que la valeur de ces paramètres était fixée par le slow-control. Le système L2 est formé d'une ou plusieurs cartes cPCI dans un châssis piloté par une carte processeur CES / RIOC[1]. C'est cette carte qui est en charge des transactions sur le bus PCI du châssis et donc du slow-control.

Le bus PCI est le seul moyen de communiquer avec les cartes L2 lorsque celle-ci sont embarquées sur la caméra. Ainsi, en plus de fournir au L2 la valeur de certains paramètres instrumentaux (position de la cible dans le plan de la caméra, numéros des lignes preL2 vers L2 actives, etc.), le slow-control doit également permettre de reconfigurer (firmware et software) les différents FPGA du système[2]. D'autres fonctions de maintenance sont confiées au slow-control (suivi de la consommation sur les différentes sources de tension, mesure de température) ainsi que l'acquisition de données brutes issues du preL2 selon le mode de fonctionnement du L2.

Cette section décrit les différentes fonctions du slow control concernant le système L2. Nous décrivons le firmware permettant, sur chaque carte, la communication entre les différentes mezzanines et le composant Spartan 3AN puis vers le bus PCI. Nous décrivons aussi le driver PCI concu pour permettre des échanges entre la logique utilisateur dans le SP3 de chaque carte et la carte RIOC sur le bus cPCI. Enfin, du cote du software utilisateur, nous décrivons les quelques procédures de base permettant d'assurer les fonctions du slow control, qu'il faudra *in fine* intégrer au software global de l'expérience HESS.

L'équipe HESS2 du CEA Saclay DSM/IRFU/SEDI a développé pour l'expérience HESS2, une carte électronique, au format CPCI, chargée de générer un trigger de niveau 2 suite aux déclenchements d'un ou plusieurs télescopes.

Dans la partie suivante, nous allons détailler les composantes de cette carte, ses différents modes de fonctionnement, la façon de l'installer et toutes les fonctions qui vont permettre de l'utiliser.

[1] Operating system ?

[2] Le bridge PCI est logé dans le composant FPGA SP3AN sur chaque carte. Ce dernier n'est pas reconfigurable par slow control. Il est configuré une fois au power on par le contenu de sa prom interne. Le contenu d'une telle PROM ne peut être modifié qu'au sol, par le connecteur JTAG sur la face avant.

La figure 1 montre une vue très simplifiée des différents principaux organes de notre carte.

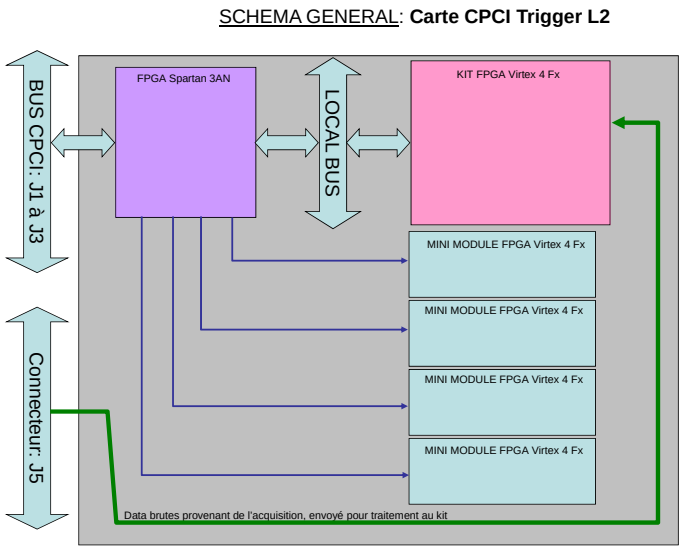

FIG. 1 – Vue générale.

Toute l'architecture de cette carte fonctionne autour d'un axe de communication que nous allons détailler dans les chapitres qui suivent. Cet axe est articulé comme suit :
Bus CPCI $<->$ SPARTAN 3 $<->$ KIT Virtex 4 (Kit V4).

En ce qui concerne les mini-modules, nous en reparlerons dans la partie Evolution. Nous allons donc plus particulièrement regarder l'axe de communication dont nous venons d'évoquer l'existence. Comme le montre La figure 2, chaque Bloc comprend des parties distinctes que nous allons maintenant étudier plus en détail.
Pour le Spartan 3AN (SP3), nous avons 2 parties :
- le Bridge
- la partie User Board

Pour le Kit V4 (KV4), également 2 parties :
- l'Interface
- la partie Processeur

La communication entre le SP3 et le KV4 se fait á travers un bus appelé « LOCAL BUS » sur lequel sont branchés le User Board et L'Interface. Cet ensemble sera appelé dans la suite du document « Partie Utilisateur ».

A noter également que les informations qui transitent entre le bridge CPCI et la partie User Board le font par autre bus de 64 bits appelé : « USER BUS ». Il s'agit lá d'un bus interne au Spartan 3AN.

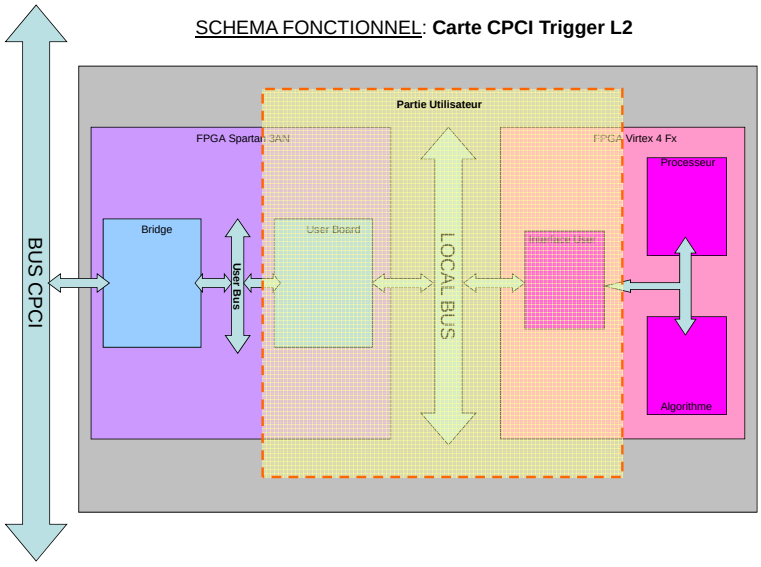

FIG. 2 – Schéma fonctionnel.

1.2 Bridge cPCI

Le bridge nous a été fourni par l'IN2P3/LPNHE, plus particulièrement par Mr P Nayman que nous tenons à remercier. Toutes les informations sont disponibles à l'adresse :
http ://lpnhe-electronique.in2p3.fr/DOCUMENTS/docpci.pdf.

Nous avons dû porter le vhdl du bridge fourni par l'IN2P3 dans autre cible FPGA que celle pour laquelle il a été écrit. Pour des raisons que nous allons décrire ci-après, nous avons décidé d'utiliser un FPGA de la même famille mais plus performant, il s'agit du Xilinx Spartan 3AN 1400. Les principales raison de ce choix sont :

- une même famille pour une homogénéité et une compatibilité du bridge dans tout le châssis du slow control.
- un plus gros chip pour contenir en plus du bridge CPCI, toute la logique de communication entre les mezzanines et le slow control et pour offrir plus de souplesse dans l'écriture du code VHDL gérant les échanges entre tous les organes de la carte TriggerL2.
- une PROM de configuration interne qui simplifie l'utilisation de la configuration par défaut ainsi que l'implémentation hardware du FPGA dans le design de la carte.

1.3 Hardware *utilisateur*

La partie utilisateur se compose de plusieurs modules qui sont implémentés dans le SP3 et dans le KV4. La figure 3 décrit ces différents modules.

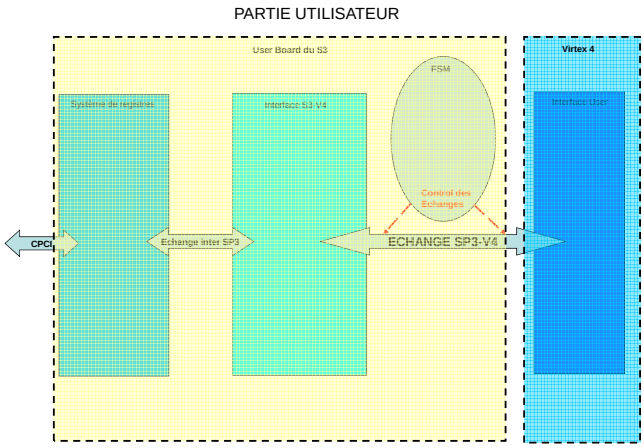

FIG. 3 – Partie utilisateur.

Nous allons voir ces différents modules, leur utilité et leur mode de fonctionnement.

1.3.1 Signaux

Pour commencer, nous allons regarder les signaux de contrôle entrant et sortant de la partie utilisateur ainsi que les signaux de contrôle interne. La plupart des signaux entrant sur la partie utilisateur sont issus du bridge. Ils sont nécessaires pour effectuer les requêtes en lecture ou écriture émises par l'utilisateur par l'intermédiaire de la Carte RIO, carte qui fait office de contrôleur du châssis CPCI. Ces requêtes seront détaillées plus tard dans ce document dans la partie Software-Fonctions utilisateur.
La figure 4 détaille les signaux venant du bridge et également les signaux sortant du SP3 et allant piloter les périphériques placés sur la carte à savoir le KIT V4, les mini-modules V4. Sont présents également les signaux internes à la partie utilisateur ayant un rôle de pilotage et de gestion/contrôle des échanges.

Signaux de contrôle entrant sur le « Système de registre » :

-Reset Hard : Actionné par le bouton poussoir de la face avant (celui du haut), il permet de faire un reset général de la carte *(cf. Gestion des Reset)*.

-User Clock : Horloge 33 MHZ issue du CPCI et traversant le Bridge.

-Address : Adresse du registre à atteindre *(8 bits)*.

-nEnable : Autorisation pour faire un accès en lecture.

-nRead : Lecture d'un registre déterminé par son adresse.

-nWrite : Ecriture dans le registre déterminé par son adresse.

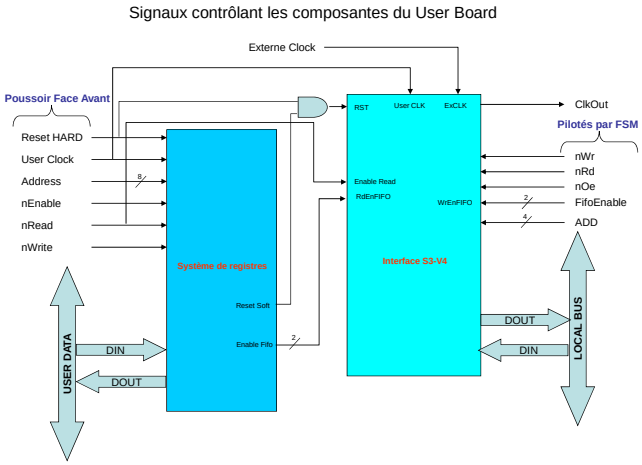

FIG. 4 – Signaux de contrôle de la partie utilisateur

Signaux de contrôle sortant du « Système de registre » :

-Reset Soft : Signal générant un reset de la partie Interface *(cf. Gestion des Reset)*.

-Enable fifo : Signal sur 2 bits activant en lecture les fifos de la partie Interface.

-UserData : Bus IN/OUT de 64 bits servant à communiquer entre le Bridge et la partie utilisateur. Dans notre cas, nous n'utiliserons que les 32 bits de poids faibles de ce bus.

Signaux de contrôle entrant sur « l'Interface SP3-V4 » :

-RST : Signal générant un reset de la partie Interface *(cf. Gestion des Reset)*.

-User Clock : Horloge 33 MHZ issue du CPCI et traversant le Bridge.

-Externe Clock : Horloge externe dont la fréquence est de 50 MHZ.

-ADD : Adresse du registre à atteindre *(4 bits)*.

-nOE : Autorisation pour faire un accès en lecture.

-nRd : Lecture d'un registre déterminé par son adresse.

-nWr : Ecriture dans le registre déterminé par son adresse.

-fifoenable : Signal sur 2 bits activant en écriture les fifos de la partie Interface.

N.B. : les signaux ADD, nOE, nRD, nWR et fifoenable sont gérés par une machine d'état dont son fonctionnement est expliqué un peu plus loin dans la documentation *(cf. Machine d'état)*.

Signaux de contrôle sortant sur « l'Interface SP3-V4 » :

-ClkOut : Horloge issue de l'horloge externe dont la fréquence est de 50 MHZ. La différence est un déphasage de 180 par rapport à l'horloge entrante.

-Local Bus : Bus IN/OUT de 48 bits servant à communiquer entre « l'interface » de la partie utilisateur et le Kit V4. Une partie de ce bus est utilisée pour les données (32 bits), le reste servant pour le protocole de communication entre le SP3 et le Kit V4.

1.3.2 Module « Système de registre »

Ce module est la partie accessible par l'utilisateur depuis le poste de contrôle via le CPCI et le bridge. C'est en fait un ensemble de registres 32 bits adressables soit en lecture, soit en écriture. Certains sont accessibles en lecture et en écriture. La figure 5 détaille la liste des registres accessibles en mode écriture par l'utilisateur.

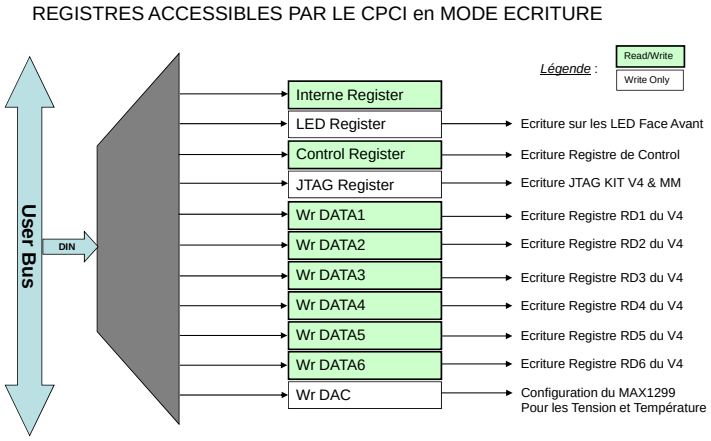

FIG. 5 – Registre en mode Ecriture

Interne Register (R/W) : registre connecté à rien, sert essentiellement au débogage. Il permet à l'utilisateur de contrôler le bon fonctionnement du système en faisant un accès en écriture d'un mot de 32 bits puis accès en lecture pour vérifier la valeur du registre. Cela permet de savoir si l'utilisateur accède correctement à la carte Trigger L2.

JTAG Register (W) : registre dans lequel le slow control, via le cPCI, va écrire les séquences « TDI TCK TMS » pour configurer les mini-modules V4 ou le KIT V4 afin de recréer une configuration type JTAG. Seuls les 3 premiers bits sont utilisés suivant l'ordre :
$Bit < 0 >$: TDI
$Bit < 1 >$: TCK
$Bit < 2 >$: TMS

Wr DAC Register (W) : registre dans lequel sont écrites les valeurs pour configurer le composant MAX1299, composant qui permet de donner la consommation des différentes tensions ainsi qu'une température ambiante de la carte au alentour du composant.

LED Register (W) : registre connecté sur le panneau de led de la face avant. Les 5 bits LSB du registre sont utilisés pour le pilotage des LEDs dont la correspondance « position $< - >$ bit » est donnée figure 6, vue de droite.

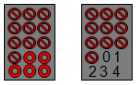

FIG. 6 – LED

Control Register (R/W) : registre très important car c'est ce registre qui pilote bon nombre d'action au sein de la partie utilisateur. La tableau 1 détaille la configuration de ce registre avec les actions résultantes de la position à 0 ou à 1 de ces bits.

Ctrl Registre< 31..0 >	Ecriture du PCI vers Exterieur		Actif	Codage
bit < 0 >	Reset SP3 Interface & FSM	1	H	0x1
bit < 1 >	Enable JTAG	1	1	0x2
bit < 4..2 >	Sélecteur des kits : V4	000	0 00xx	
	MM1	001	0 01xx	0x4
	MM2	010	0 10xx	0x8
	MM3	011	1 00xx	0x10
	MM4	111	1 11xx	0x1C
bit < 5 >	Read Enable pour lecture fifoInfo	1	H	0x20
bit < 6 >	Read Enable pour lecture fifoData	1	H	0x40
bit < 7 >	Start FSM	1	H	0x80
bit < 8 >	Download Request	1	H	0x100
bit < 13 >	Write Enable pour écriture dans Fifo Test	1	H	0x2000
bit < 14 >	Read Enable pour lecture dans Fifo Test	1	H	0x4000
bit < 17 >	Reset 3 MM1 S	1	H	0x00020000
bit < 18 >	Reset 2 MM1 H	1	H	0x00040000
bit < 19 >	Reset 1 MM1 P	1	H	0x00080000
bit < 20 >	Reset 3 MM2 S	1	H	0x00100000
bit < 21 >	Reset 2 MM2 H	1	H	0x00200000
bit < 22 >	Reset 1 MM2 P	1	H	0x00400000
bit < 23 >	Reset 3 MM3 S	1	H	0x00800000
bit < 24 >	Reset 2 MM3 H	1	H	0x01000000
bit < 25 >	Reset 1 MM3 P	1	H	0x02000000
bit < 26 >	Reset 3 MM4 S	1	H	0x04000000
bit < 27 >	Reset 2 MM4 H	1	H	0x08000000
bit < 28 >	Reset 1 MM4 P	1	H	0x10000000
bit < 29 >	Reset 3 KV4 S=Soft sur Processeur	1	H	0x20000000
bit < 27 >	Reset 2 KV4 H=Hard sur Processeur	1	H	0x40000000
bit < 28 >	Reset 1 KV4 P=Périphérique VHDL	1	H	0x80000000

TAB. 1 – Configuration des bits du Control Register

$Bit < 0 >$: sert à reseter la Partie Utilisateur sauf le module « système de registre ».
$Bit < 1 >$: enable les multiplexeurs et démultiplexeurs de la partie JTAG pour le download des configurations dans les différents V4 de la carte. Sert également d'aiguillage pour le RS232.
$Bit < 4..2 >$: sélectionne soit le Kit V4, soit l'un des mini-modules pour soit reconfigurer le FPGA par JTAG, soit atteindre le processeur du FPGA via RS232.
$Bit < 5 >$: autorise la lecture de la fifo « fifoInfo » contenant les informations relatives aux données à lire.
$Bit < 6 >$: autorise la lecture de la fifo « fifoData » contenant les données à lire.
$Bit < 7 >$: démarrage de la machine d'état responsable des échanges entre SP3 et KV4.
$Bit < 8 >$: demande de reconfiguration des V4 par cPCI-JTAG.
$Bit < 13 >$: autorise l'écriture dans la fifo « fifoTest », fifo utile uniquement pour le debug.
$Bit < 14 >$: autorise la lecture de la fifo « fifoTest ».
$Bit < 17..31 >$: bits de Reset pour les différents périphériques de la carte.

N.B. : Pour chaque carte (kit et mini-modules), il existe 3 types de Reset que l'on peut appliquer en fonction des bits du Control Register *(cf. fig. 7)*. Chaque type applique un Reset bien précis sur la cible. Le Reset dit « P ou Périphérique » occasionne uniquement un reset de la partie VHDL de la cible. Le Reset dit « S ou Soft » occasionne un reset simple de la partie processeur de la cible contrairement au Reset « H ou Hard » qui lui aussi resete le processeur mais de façon plus brutale, l'équivalent d'un OFF/ON, tout repart de l'état initial.

Wr DATA1 à Wr DATA 6 Register (W/R) : Ce sont 6 registres à disposition de l'utilisateur pour passer des paramètres au Kit V4.
$WrDATA1$ et $WrDATA2$: description des lignes actives.
$WrDATA3$: Xc (16 bits : 0 à 15) et Yc (16 bits : 16 à 31) :position centre cible.
$WrDATA4$: Seuil1 (8 bits : 0 à 7), Seuil2 (8 bits : 8 à 15) et Coupure COG (16 bits : 16 à 31).
$WrDATA5$: max (16 bits : 0 à 15) et Mode (16 bits : 16 à 31).
$WrDATA6$: LoverS (16 bits : 0 à 15) et LoverW (16 bits : 16 à 31).

La figure 7 détaille la liste des registres accessibles en mode lecture par l'utilisateur.

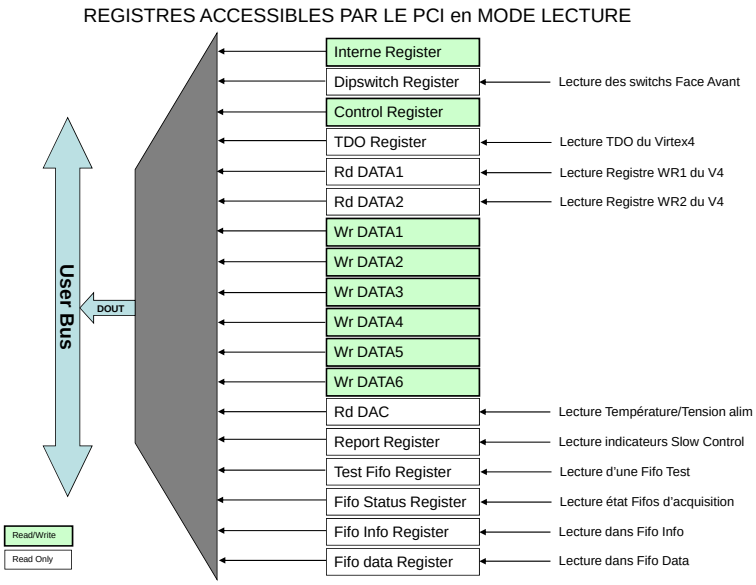

FIG. 7 – Registres en mode lecture

Interne Register (R/W) : registre de lecture pour le test de fonctionnement du système.

Dipswitch Register (R) : registre de lecture des positions des switchs de la face avant.

Control Register (R/W) : permet la relecture de ce registre de contrôle.

TDO Register (R) : registre dans lequel est écrit le « TDO » du JTAG sur le bit$< 0 >$.

Rd DATA1 et Rd DATA2 Register (W/R) : 2 registres qui permettent au kitV4 de remonter des informations vers le poste de pilotage.

Wr DATA1 à Wr DATA 6 Register (W/R) : en mode lecture, permet de vérifier les informations passées au KITV4.

Rd DAC (R) : registre dans lequel sont retournées les informations fournies par le DAC1299.

Test Fifo Register (R) : registre dans lequel se trouvent les données issues de la fifo Test.

Fifo Info Register (R) : registre dans lequel se trouve la data issue de la « fifo Info » lors d'une demande de lecture de cette fifo. Cette data représente le nombre de mots à lire dans la « fifo Data ».

Fifo Data Register (R) : registre dans lequel se trouve la data issue de la « fifo Data » lors d'une demande de lecture de cette fifo.

Fifo Status Register (R) : registre dans lequel se trouvent les données représentant l'état des fifos Info et Data suivant la représentation :
$Bit < 15..0 >$: Information sur l'état de la **Fifo Info**.
-$Bit < 0 >$: indicateur « empty » de la fifo *(vrai si vide).*
-$Bit < 1 >$: indicateur « almost empty » de la fifo *(vrai si au plus 1 valeur dedans).*
-$Bit < 2 >$: indicateur « full » de la fifo *(vrai si pleine).*
-$Bit < 3 >$: indicateur « almost full » de la fifo *(vrai si reste 1 case libre).*
-$Bit < 15..4 >$: indicateur « word count » de la fifo *(nombre de mot contenu dans la fifo).*
$Bit < 31..16 >$: Information sur l'état de la **Fifo Data**.
-$Bit < 16 >$: indicateur « empty » de la fifo *(vrai si vide).*
-$Bit < 17 >$: indicateur « prog empty » de la fifo *(vrai si au plus 100 valeurs dedans).*
-$Bit < 18 >$: indicateur « full » de la fifo *(vrai si pleine).*
-$Bit < 19 >$: indicateur « prog full » de la fifo *(vrai si reste 200 cases libres).*
-$Bit < 31..20 >$: indicateur « word count » de la fifo *(nombre de mot contenu dans la fifo).*

Report Register (R) : registre dans lequel sont retournées les informations de slow control. Le tableau 2 détaille la configuration de ce registre.

Report Registre$< 31..0 >$	Lecture via PCI	Actif
$bit < 0 >$	Timeout Check Config V4	1
$bit < 1 >$	Autorisation Download config dans V4	1
$bit < 2 >$	Register Mode	1
$bit < 3 >$	Fifo Mode	1
$bit < 4 >$	FSM Run/Wait	1/0
$bit < 5 >$	Fifo Data full	1
$bit < 14 >$	Not Empty Fifo Test	1
$bit < 25 >$	Config MM1 ok	1
$bit < 26 >$	Config MM2 ok	1
$bit < 27 >$	Config MM3 ok	1
$bit < 28 >$	Config MM4 ok	1
$bit < 29 >$	Config V4 ok	1
$bit < 30 >$	Sequence Reset ON/OFF	1/0
$bit < 31 >$	L2 Ready/Busy	1/0

Tab. 2 – Configuration des bits du Control Register

$Bit < 0 >$: indique s'il y a un problème au niveau de KIT V4. Si un problème est détecté que ce soit sur la présence du KIT, son alimentation, sa configuration alors ce bit passe à 1.
$Bit < 1 >$: suite à une requête de demande de reconfiguration d'un V4 quel qu'il soit, ce bit passera à 1 pour signaler à l'utilisateur que le système est prêt à être reconfigurer.
$Bit < 3..2 >$: indique dans quel mode se trouve la machine d'état *(cf. Machine d'Etat).*
$Bit < 4 >$: indique dans quel état se trouve la machine d'état *(cf. Machine d'Etat).*
$Bit < 5 >$: indique que la fifo « fifoData » est pleine.
$Bit < 14 >$: indique que la fifo « fifoTest » n'est pas vide.
$Bit < 14 >$: autorise la lecture de la fifo « fifoTest ».
$Bit < 25 >$: indique si le mini-module 1 est prêt à fonctionner alors bit à 1.
$Bit < 26 >$: indique si le mini-module 2 est prêt à fonctionner alors bit à 1.
$Bit < 27 >$: indique si le mini-module 3 est prêt à fonctionner alors bit à 1.
$Bit < 28 >$: indique si le mini-module 4 est prêt à fonctionner alors bit à 1.
$Bit < 29 >$: indique si le KIT V4 est prêt à fonctionner alors bit à 1.
$Bit < 30 >$: indique si le système se trouve dans un mode de Reset.
$Bit < 31 >$: donne l'état général de la carte TriggerL2.

La figure 8 donne les différentes connexions qui configurent les bits du Report Register.

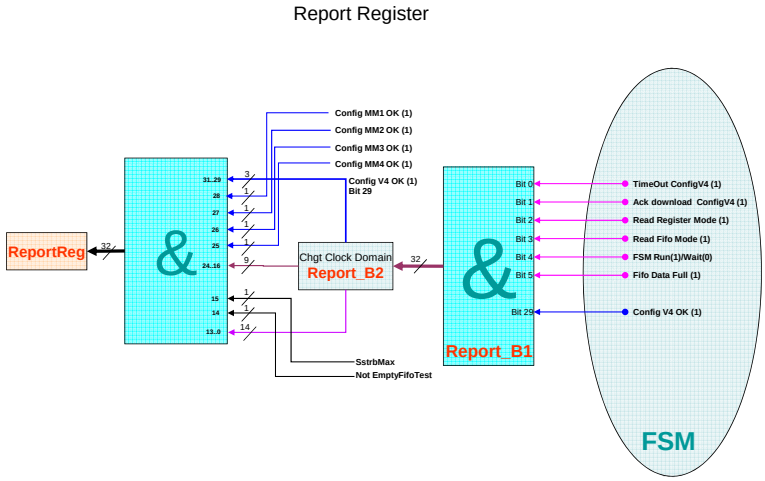

Fig. 8 – Connexion du Report Register

1.3.3 Module « Interface SP3-V4 »

Ce module est une partie tampon entre l'utilisateur et le KIT V4. Il est constitué d'un ensemble de registres 32 bits qui sont connectés d'un coté aux registres du module « Système de registre » et de l'autre à des registres du kit V4 via le « Local Bus » *(cf. Registre du Kit V4)*. Ce module possède des registres accessibles en lecture ou en écriture ainsi que 2 fifos. Les figures 9 - 10 indiquent comment sont connectés les différents organes du système et comment se font les échanges.

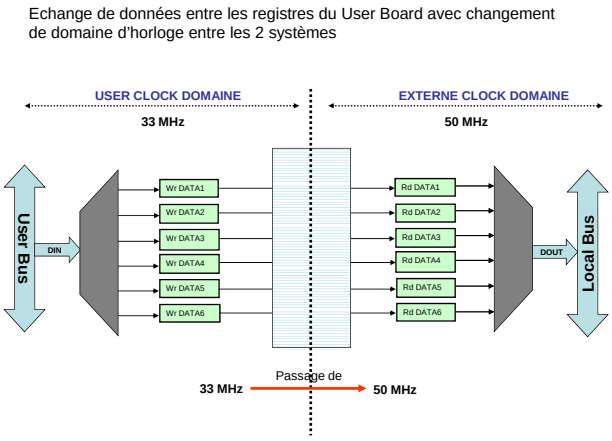

FIG. 9 – Registres en mode écriture

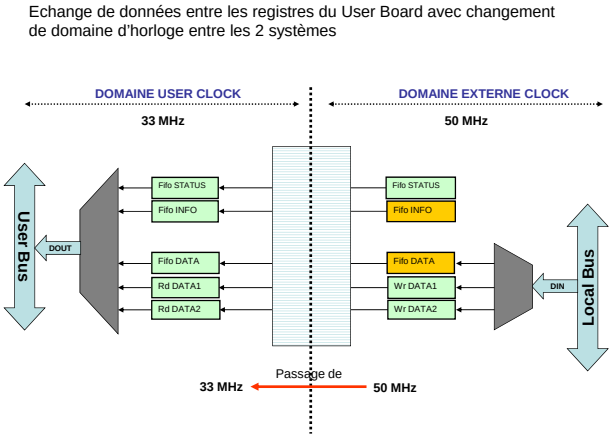

FIG. 10 – Registres en mode lecture

Dans ce module, les registres sont mis à jour en permanence sans intervention de l'utilisateur c'est-à-dire qu'un mot écrit par l'utilisateur via le Bridge cPCI arrive dans un des registres « Wr DATA ». Moyennant un changement de domaine de clock, ce mot sera recopié dans le registre symétrique du module « Interface » afin qu'il soit disponible pour y être envoyé vers le KitV4 via le « Local Bus ».

Une même procédure est établie pour faire le chemin inverse, c'est-à-dire remonter un mot du V4 vers l'utilisateur *(cf. figure 10)*.

Dans cette partie du module, on note la présence de 2 fifos, la « Fifo Info » et la « Fifo Data ». La « Fifo Data » contient les données brutes issues du PréL2, qui auront transitées par le V4. La fifo se remplira lorsque le résultat du traitement des données d'un évènement *(cf. Partie Algorithme)* dans le V4 générera un L2Accept.

Le format du bloc de donnée qui sera contenue dans cette fifo est décrit par figure 11.

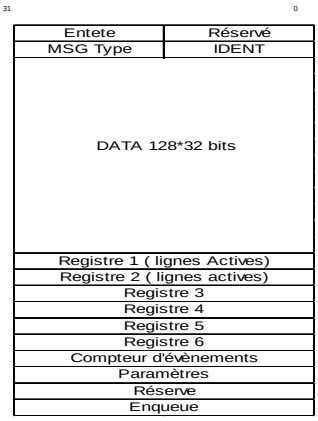

FIG. 11 – Format data

La « Fifo Info » contient 1 seul mot par bloc de donnée enregistré dans la « Fifo Data ». Donc pour chaque L2A, un mot est enregistré automatiquement dans cette fifo. La valeur de ce mot représente le nombre de mot de 32 bits composant le bloc de la figure 11 soit 140 mots de 32 bits d'oú la valeur 0x8C. Ainsi pour savoir combien de blocs sont en mémoire dans la « Fifo Data », il suffira juste pour l'utilisateur de savoir combien de mots sont contenus dans la « Fifo Info ».

Pour ce faire, l'utilisateur n'aura juste qu'à lire la valeur du **Fifo Status Register** qui est la représentation de ce que contiennent nos 2 fifos *(cf. format du registre détaillé précédemment)*.

Nous venons de voir les échanges internes au SP3, nous allons maintenant détailler le fonctionnement des échanges entre le module « Interface SP3-V4 » et la partie « Interface User » du KIT V4. Ces échanges sont arbitrés par une machine d'état qui va contrôler les accès au bus, faire la mise à jour des registres, gérer certaines informations du Report Register.

La figure 12 nous montre une vue d'ensemble des éléments pilotés par cette machine d'état. Certains signaux entre le SP3 et le Kit V4 gérés par la FSM servent de protocole de communication entre les 2 entités. Nous retrouvons les signaux RdV4, WrV4, ADDR et enFifoV4 ayant les mêmes fonctions que celles vues précédemment pour lire et/ou écrire dans les registres et fifo du Kit V4 *(cf. Signaux de contrôle de la partie utilisateur)*.Nous notons également la présence de 3 signaux de Reset, RstV4_1, RstV4_2 et RstV4_2 chargés de reseter la partie Hardware et la partie Software du Kit V4.

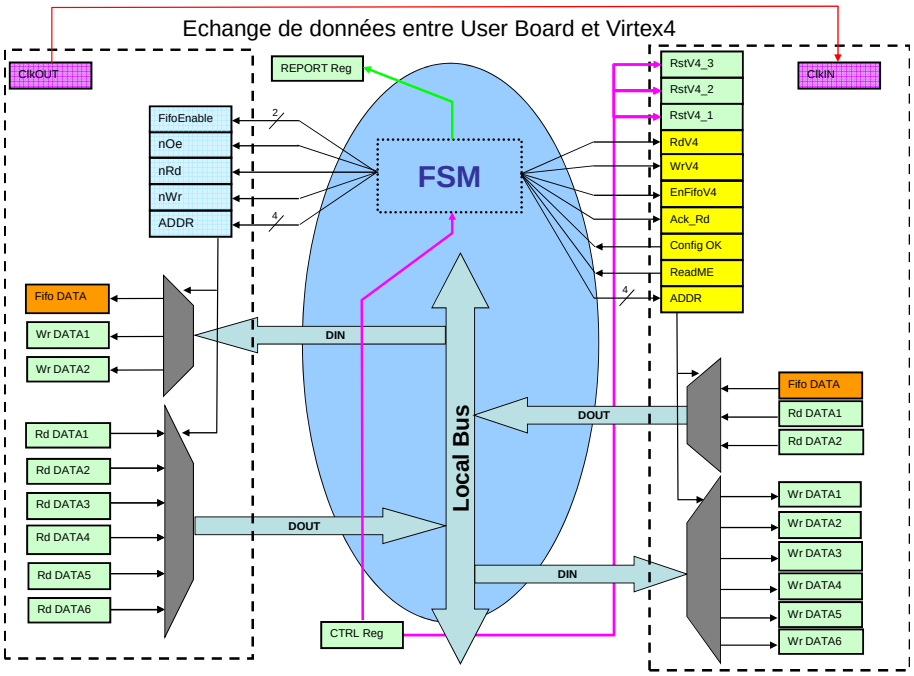

FIG. 12 – Machine d'état

Nous trouvons aussi 3 autres signaux ConfigOK, ReadMe et Ack_Rd ayant les significations suivantes :
- ConfigOK : à 1, indique à la machine d'état si le Kit V4 est bien configuré et prêt.
- ReadMe : à 1, indique à la machine d'état que la « fifoDataV4 » doit être lue.
- Ack_Rd : à 1, indique au KitV4 que la machine d'état a pris en compte le ReadME.

1.3.4 Les Registres et signaux de la partie « Interface User » du KIT V4

Comme le montre La figure 12, dans cette partie, nous trouvons également 6 registres en écriture, Wr DATA1 à Wr DATA 6 dans lesquels sont passés les informations venant du SP3. Pour rappel, ils correspondent à :
$WrDATA1$ et $WrDATA2$: description des lignes actives.
$WrDATA3$: Xc (16 bits : 0 à 15) et Yc (16 bits : 16 à 31).
$WrDATA4$: Seuil1 (8 bits : 0 à 7), Seuil2 (8 bits : 8 à 15) et Coupure COG (16 bits : 16 à 31).
$WrDATA5$: max (16 bits : 0 à 15) et Mode (16 bits : 16 à 31).
$WrDATA6$: LoverS (16 bits : 0 à 15) et LoverW (16 bits : 16 à 31).

Nous avons également 2 registres en lecture, Rd DATA1 et Rd DATA 2, dans lesquels le Kit V4 peut retourner des informations à l'utilisateur.

Tous ces registres sont connectés au « Local Bus », bus de 32 bits IN/OUT partagé également par « l'Interface SP3-V4 ». C'est le bus d'échange de données entre les 2 FPGA. Tous les échanges sont pilotés par la machine d'état qui sert d'arbitre dans les échanges *(cf. Machine d'état)*.

Nous avons aussi des signaux de contrôle qui gèrent cette partie du Kit V4.
Signaux entrants :
-RstV4_1 : reset de la partie Hardware (périphérique VHDL) du Kit V4.
-RstV4_2 : reset hard de la partie Software (Processeur PowerPC) du Kit V4.
-RstV4_3 : reset soft de la partie Software (Processeur) du Kit V4.
-RdV4 : permet la lecture des registres ad-hoc.
-WrV4 : permet l'écriture dans les registres ad-hoc.
-ADDR : détermine le registre à lire ou écrire (4 bits).
-enFifoV4 : autorise la lecture de la fifo.
-Ack_Rd : accusé de réception de la part du SP3 d'un signal ReadME.

Signaux sortants :
-ConfigOK : indicateur de bon fonctionnement du Kit V4.
-ReadMe : indicateur de données disponible à lire dans la « Fifo Data V4 ».

Nous trouvons une fifo qui contient les données brutes issues du pré-L2 : « Fifo Data V4 ».

1.4 Machine d'état (FSM)

La figure 13 nous donne le schéma de principe du fonctionnement de notre machine d'état.
A la mise sous tension ou après un reset, la FSM se met dans un état d'attente *(état idle)* prêt à

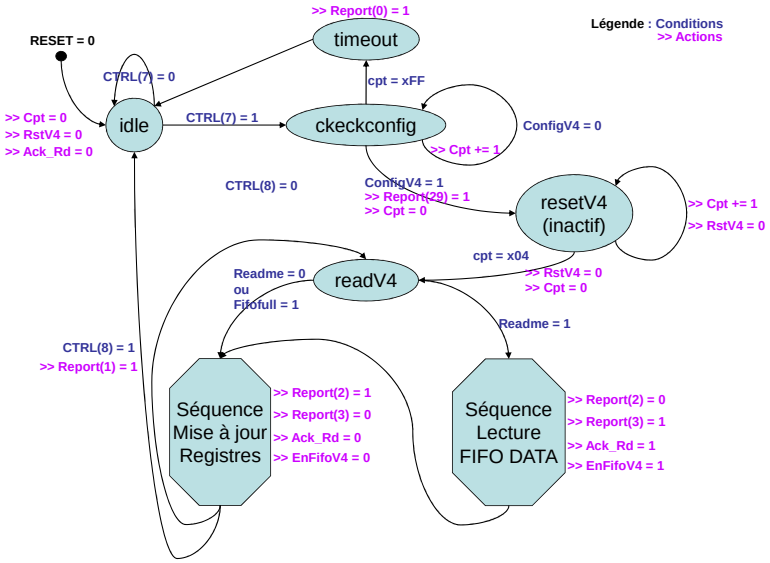

FIG. 13 – Schéma de fonctionnement FSM

fonctionner. L'utilisateur doit alors lui donner l'ordre de démarrer pour commencer les échanges de données entre le SP3 et le Kit V4. Cet ordre est déclenché par la mise à 1 du *bit* $< 7 >$ du Control Register. Une fois le bit à 1, la FSM change d'état et se retrouve dans un état (état checkconfig) où elle scrute le signal ConfigOK. Ce signal piloté par le KitV4 indique s'il vaut 1 que le Kit V4 est présent et bien configurer. Si ce signal est à 0, la FSM le scanne pendant 255 coups d'horloge. Au bout de ce temps, si le signal n'est toujours pas monté à 1, la FSM envoie un avertissement à l'utilisateur *(état timeout)* en positionnant le *bit* $< 0 >$ du Report Register. Ensuite, la FSM se remet en attente *(état idle)*. Si le signal ConfigOK est à 1, la FSM passe alors dans l'état suivant *(état resetV4)* et positionne le *bit* $< 29 >$ du Report Register pour signaler que tout est en ordre au niveau du kit V4. Dans ce nouvel état, la FSM fixe le signal RstV4_1 à 1 pour effectuer sur le KIT V4 un reset de la partie Hardware. Ce signal est haut durant 4 coups de d'horloge puis la FSM passe dans l'état suivant *(état ReadV4)*.

A ce niveau, 2 types de fonctionnement possibles. Soit mise à jour des registres entre le Kit V4 et « l'interface SP3-V4 », soit lecture de la fifoDataV4 + mise à jour des registres. Le choix se fait en fonction de la valeur du signal ReadMe fourni par le Kit V4. Si ce signal vaut 0, alors la FSM entre dans la séquence « mise à jour des registres ». Dans ce cas, les registres en lecture de l'interface sont recopiés dans le V4 et inversement, les registres en lecture du V4 sont recopiés dans les registres de l'interface. A la fin de cette séquence, la FSM revient dans l'état ReadV4 et repart dans un nouveau test du signal ReadMe.

Si le signal ReadMe fourni par le Kit V4 à la FSM vaut 1, cela signifie que la « fifoDataV4 » contient des données pour lesquelles a été généré un Trigger L2 Accept. En principe, ce sont des données intéressantes dont l'utilisateur peut avoir besoin. Pour cela, il faut donc récupérer ces données brutes, issues du préL2 et stockées dans le Kit V4, pour les stocker dans la « Fifo Data », laquelle fifo pourra ensuite être lue par l'utilisateur. La FSM en entrant dans la partie

séquence « lecture Fifo Data » va se charger de faire cela. Une fois le bloc de données récupéré, la FSM écrit dans la « fifo Info » le nombre de mot contenu dans le bloc de donnée de la « fifo Data ». Ensuite, la FSM change d'état pour repasser dans la séquence « mise à jour des registres ».

Pour suivre l'évolution du fonctionnement de la FSM, en fonction de la séquence exécutée, la FSM met à jour certains bits du Report Register :
-si séquence registre : $bit < 2 >= 1$ et $bit < 3 >= 0$
-si séquence fifo : $bit < 2 >= 0$ et $bit < 3 >= 1$

Idem avec certains signaux du protocole de communication entre Kit V4 et interface SP3 :
-si séquence registre : Ack_Rd = 0 et EnFifoV4 = 0
-si séquence fifo : Ack_Rd = 1 et EnFifoV4 = 1

Ack_Rd = 1 signifie que la FSM a pris en compte le signal ReadMe et autorise donc la lecture de la fifoV4 d'où l'enable de cette fifo avec le signal EnFifoV4 qui passe à 1.

N.B.1 : A la fin de la séquence « mise à jour des registres », la FSM revient dans l'état ReadV4 uniquement si l'utilisateur ne demande pas à changer la configuration du V4. Pour se faire, l'utilisateur positionne le bit$< 8 >$ du Control Register à 1 et attend l'autorisation par la FSM de faire la reconfiguration. L'autorisation se matérialise par la mise à 1 par la FSM du bit$< 1 >$ du Report Register. Dans ce cas, la FSM revient dans l'état *idle*. A la fin de la reconfiguration, l'utilisateur devra redémarrer la FSM pour relancer les échanges entre l'interface et le Kit V4.

N.B.2 : Si la « fifo Data » est pleine et que le Kit V4 envoie un signal ReadMe, la FSM ne passera pas dans la séquence « lecture Fifo Data » et les données seront alors perdues.

Les figures 14, 15, et 16, donnent les chronogrammes de cette machine d'état pour les différentes phases de fonctionnement.

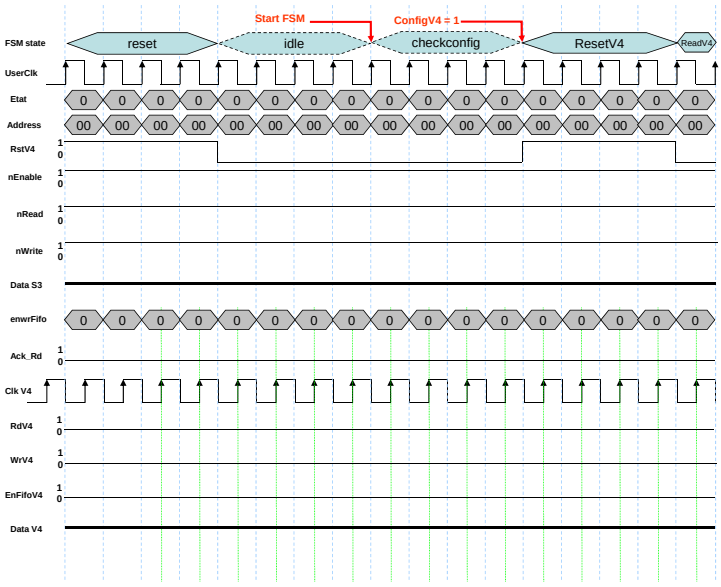

Fig. 14 – Chronogramme FSM

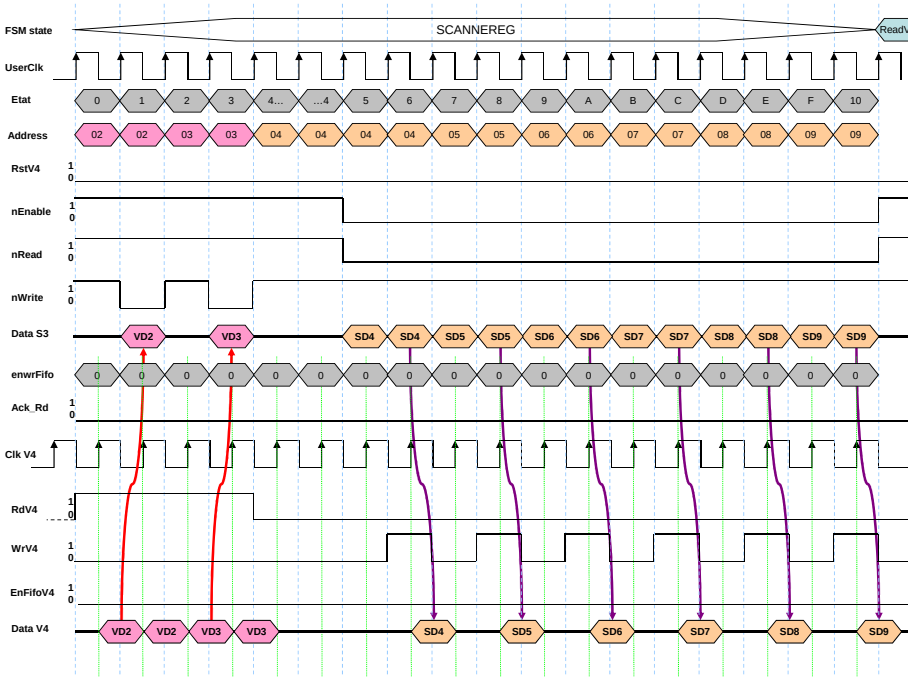

Fig. 15 – Chronogramme « mise à jour des registres »

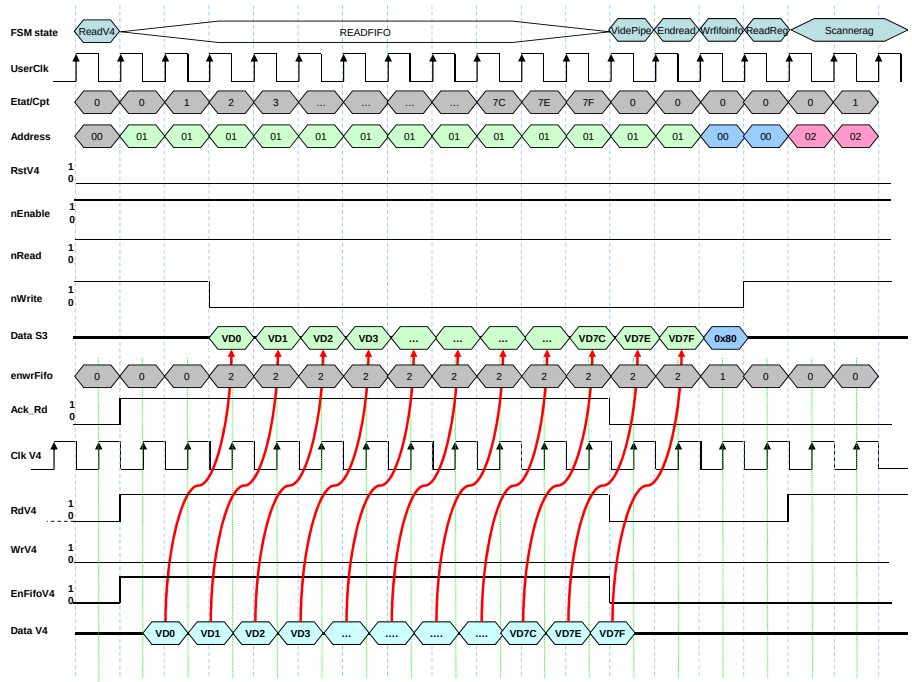

Fig. 16 – Chronogramme « lecture Fifo Data V4 »

1.4.1 Les différents Mode de Reset

La carte et ses différents périphériques peuvent être reseter tout ou partie par différents moyens soit purement mécanique, soit par software (via CPCI). La figure 17 recense les différents types de Reset.
Pour reseter complètement la carte (tous les composants SP3, Kit V4, mini-modules), l'utilisateur

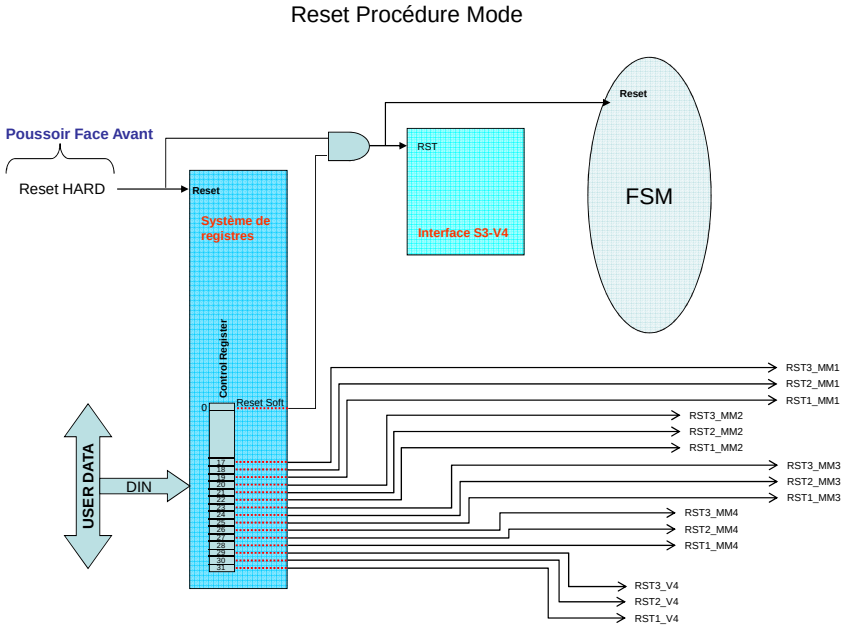

FIG. 17 – Reset Mode

peut le faire en actionnant mécaniquement sur la face avant de la carte le bouton poussoir **J1**. Cela resetera toute la partie utilisateur du SP3 (système de registre et Interface SP3-V4), la machine d'état ainsi que la partie Interface User du V4. Après, il est possible de reseter indépendamment chaque partie soit du SP3, soit des V4. Pour cela l'utilisateur positionnera comme il se doit différents bits du Control Register :
$Bit < 0 >$: Reset des Interfaces SP3−V4 et FSM.
$Bit < 28..17 >$: 3 signaux de reset pour chaque mini-module. Tout comme le Kit V4, chaque mini-module possède les 3 types de Reset : Périphérique, Hard et Soft.
$Bit < 29 >$: Reset SOFT du PowerPC du Kit V4 $->$ signal **RST3_V4**.
$Bit < 30 >$: Reset HARD du PowerPC du Kit V4 $->$ signal **RST2_V4**.
$Bit < 31 >$: Reset du Périphérique VHDL : Interface User Kit V4 $->$ signal **RST1_V4**.

Rappel : Pour chaque carte (kit et mini-modules), il existe 3 types de Reset que l'on peut appliquer en fonction des bits du Control Register *(cf. fig. 7)*. Chaque type applique un Reset bien précis sur la cible. Le Reset dit « P ou Périphérique » occasionne uniquement un reset de la partie VHDL de la cible. Le Reset dit « S ou Soft » occasionne un reset simple de la partie processeur de la cible contrairement au Reset « H ou Hard » qui lui aussi resete le processeur mais de façon plus brutale, l'équivalent d'un OFF/ON, tout repart de l'etat initial.

1.4.2 Configuration par JTAG

Le Kit V4 ainsi que les différents mini-modules comportant eux aussi un FPGA de type Virtex V4 peuvent être configurés de 2 façons :
- soit en utilisant les outils Xilinx (Impact)+ boitier Platform Cble USB branché sur le connecteur JTAG de la face avant de la carte *(cf. fig. 18)*.
- soit par le CPCI, le bridge et la partie utilisateur.

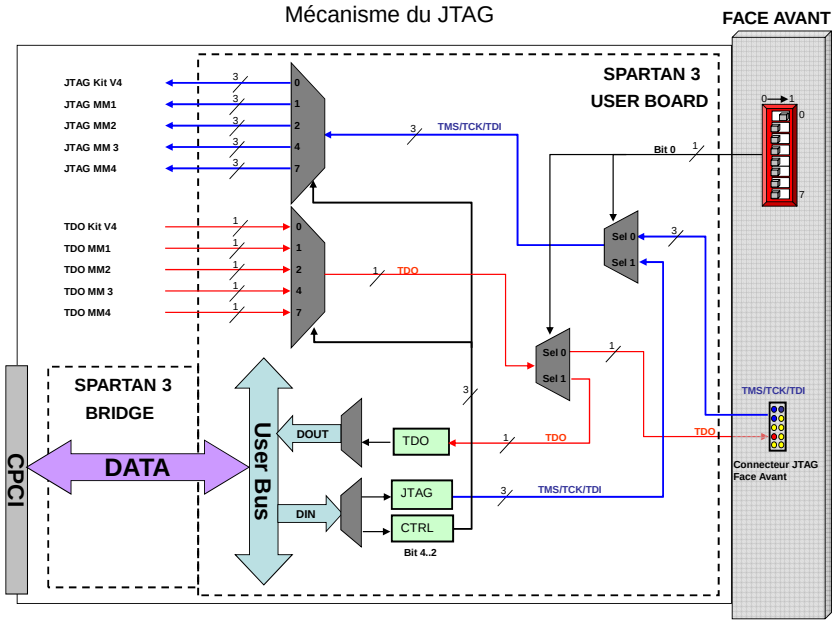

FIG. 18 – JTAG

Le choix entre les 2 modes de download se fait en actionnant le switch 0 *(cf. fig. 18)* :
- en position 0 : JTAG par le connecteur de la Face Avant.
- en position 1 : JTAG par la voie CPCI-Bridge.

Il faut également choisir vers quel V4, l'utilisateur veut downloader la nouvelle configuration. Pour se faire, il doit utiliser certains bits du Control Register, ce sont les bits< 4..2 > *(cf. tableau 1)*. Si le download se fait par CPCI, l'utilisateur doit, une fois le choix du V4 fait, lancer une procédure de download en soft, procédure que nous détaillerons dans la partie software−fonction utilisateur.

1.4.3 RS232

L'utilisateur peut accéder au processeur du Kit V4 ainsi qu'aux processeurs des différents V4 des mini-modules en utilisant la connexion au RS232. La figure 19 détaille le principe de connexion au RS232 :

Pour utiliser cette fonction, il faut que le switch 0 soit en position 0 sinon il sera impossible

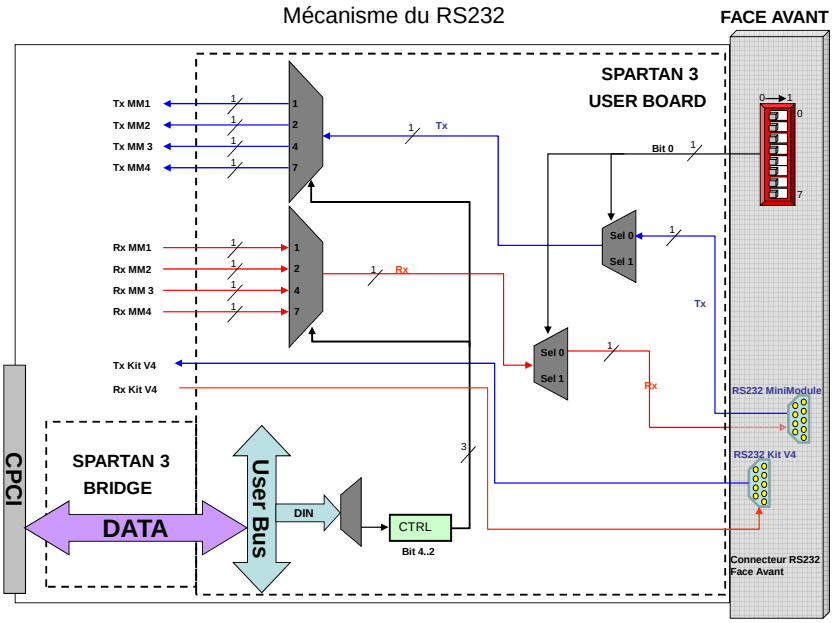

FIG. 19 – RS232

d'accéder aux mini-modules. A noter que l'accès au V4 du Kit se fait en direct peut importe la position du switch. Par contre, pour les mini-modules, cela passe obligatoirement par une partie logique afin de choisir quel module parmi les 4, l'utilisateur désire atteindre. Pour se faire, l'utilisateur doit configurer, tout comme pour le JTAG, certains bits du Control Register, ce sont les bits< 4..2 > *(cf. table 1)*. Le RS232 est utile pour le debugge afin d'accéder aux processeurs des V4 pour tester leurs fonctionnements, réaliser l'affichage des actions des V4. En fonctionnement nominal, cela n'a pas d'utilité.

1.4.4 Registres d'échanges d'information entre le SP3 et l'exterieur

La figure 20 nous montre les autres les registres utilisés entre l'intérieur du SP3 et le reste de la carte Trigger L2.

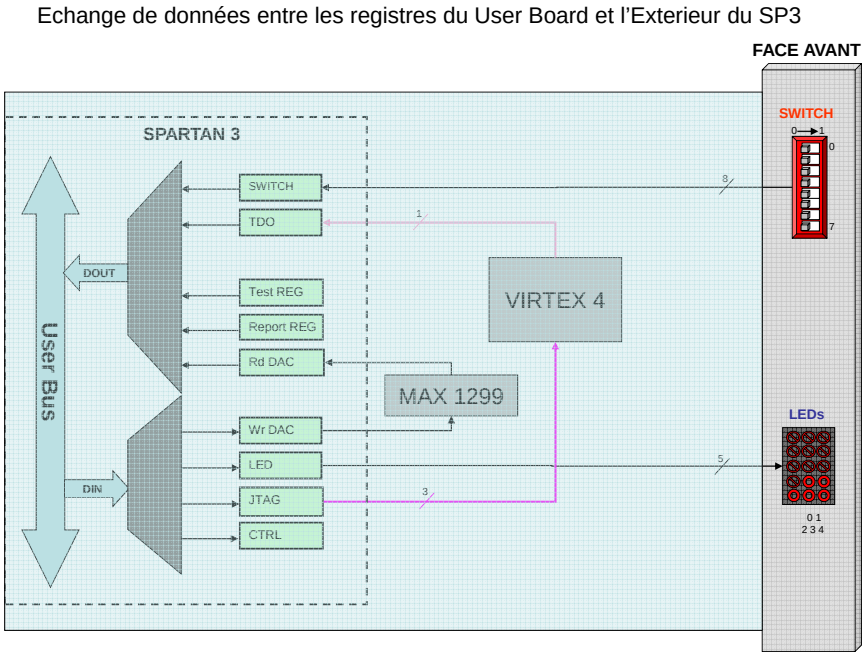

FIG. 20 – Registres vus de l'extérieur du SP3

1.5 Partie Software - Driver cPCI

Nous avons réalisé un driver simple ne permettant pas la fonction « Hot Swap » de notre carte Trigger L2.
Tout d'abord, au niveau du Bridge dans le Spartan 3AN, on a modifié quelques paramètres afin d'identifier correctement notre carte : reconnaissance de la carte dans le châssis $->$ modification du module cpcipack.vhd pour lui intégrer les ID (Device et Vendor) que l'on a défini pour notre carte Trigger L2. Arbitrairement, nous avons choisi 2 mots de 16 bits que sont :
- pour le DeviceID : 0xE552 (visuellement ressemblant à ESS2 = HESS2!!!)
- pour le VendorID : le CEA soit sur 16 bits 0x0CEA

```
library ieee;
use ieee.std_logic_1164.all;
package cpcipack is
-- For each type board use a different device id
constant DeviceId : std_logic_vector(15 downto 0):=X"E552";
-- TrigL2_board
-- Use a valid vendor id, for ex. the cern device id=10DCH
CONSTANT VendorId : STD_LOGIC_VECTOR(15 downto 0):=X"0CEA";
-- CEA VendorID
-- User32_64b defines if the user interface is 32 or 64-bit
constant User32_64b : std_logic := '1'; --0=64bits, 1=32bits
-- Use a valid revision id and change the version number if necessary
CONSTANT RevisionId : STD_LOGIC_VECTOR(7 downto 0) :=X"02";
-- Version number of the interface
end cpciPack;
```

On retrouvera ces informations dans l'Espace de Configuration de la carte (voir Norme Compact CPCI). Le reste des valeurs contenues dans cet espace de configuration est implémenté dans le module Bardec.vhd. C'est là que l'on définit entre autre les tailles des différents espaces mémoire adressables (BAR0 et BAR1). Ce module implémente au total un bloc de 64 registres de 32 bits plus quelques registres de contrôle. Le BAR0 a une taille de 128 KB et le BAR1 fait 8 KB.
Pour savoir si la carte Trigger L2 est bien reconnue par la carte RIOC, carte processeur maitre du châssis CPCI et vérifier que les paramètres sont bons, il convient de réaliser les actions suivantes. Tout d'abord, lancer une fenêtre Terminal puis taper après le prompt : **minicom**. Allumer le châssis, la carte RIOC va alors se lancer. Appuyer sur la barre $<$ *espace* $>$ pour éviter le boot complet. Un nouveau promt apparait : **PPC_Mon**. On se trouve dès lors dans la partie monitor de la carte où l'on peut faire certaines procédures de vérification.

```
PPC_Mon>cpci show
CPCI interface [enabled]
   System Controller
   Slave       : A32 Size = 32 Mbytes
                 A32 Base = 0x40000000
                 A64 Base = 0x00000000'00000000  [disabled]
+----+-----------------------------------------+
|              PCI DEVICES  [tree 8 ]           |
+----+------+------+----+----+----+----+----+----+
|idx | vid  | did  |slot|func|bus |b_id|ivec|pmc |
+----+------+------+----+----+----+----+----+----+
|  0 | 10d9 | 4065 | 00 | 00 | 00 |  0 | 00 |  x |
|  1 | 0cea | e552 | 07 | 00 | 00 |  0 | 00 |  7 |   <--- carte TriggerL2
|  2 | 10d9 | 4065 | 08 | 00 | 00 |  0 | 00 |  8 |
+----+------+------+----+----+----+----+----+----+
```

Cela donne la cartographie des devices CPCI que la carte RIOC à trouver dans le châssis. On peut alors vérifier que notre carte est bien reconnue et elle se situe dans le slot **7**.
Si on veut plus de détail sur notre carte Trigger L2, il suffit de taper la commande :
cpci show dev 7.

```
PPC_Mon>cpci show dev 7
Single function device found in slot : 0x07   bus : 0x00
| vid  | did  | type |   addr   |   size   | space| bar  |
+------+------+------+----------+----------+------+------+
| 0cea | e552 |  D   | 42000000 | 00020000 | MEM  |  0   |
                     | 42100000 | 00002000 | MEM  |  1   |
```

On retrouve alors nos 2 espaces mémoires adressables et leur taille ainsi que leurs adresses de bases attribuées dynamiquement par la carte RIOC :
- BAR0 : 0x42000000.
- BAR1 : 0x42100000.

On peut egalement lister la totalité de l'espace de configuration :

```
PPC_Mon>pcc.l 0 ? 7
0x00 [0] [0x07] [0x800] : e5520cea ->
0x04 [0] [0x07] [0x800] : 00800082 ->
0x08 [0] [0x07] [0x800] : 11800002 ->
0x0c [0] [0x07] [0x800] : 00000000 ->
0x10 [0] [0x07] [0x800] : 42000000 ->
0x14 [0] [0x07] [0x800] : 42100000 ->
0x18 [0] [0x07] [0x800] : 00000000 ->
0x1c [0] [0x07] [0x800] : 00000000 ->
0x20 [0] [0x07] [0x800] : 00000000 ->
0x24 [0] [0x07] [0x800] : 00000000 ->
0x28 [0] [0x07] [0x800] : 00000000 ->
0x2c [0] [0x07] [0x800] : 00000000 ->
0x30 [0] [0x07] [0x800] : 00000000 ->
0x34 [0] [0x07] [0x800] : 00000000 ->
0x38 [0] [0x07] [0x800] : 00000000 ->
0x3c [0] [0x07] [0x800] : 00000100 ->
0x40 [0] [0x07] [0x800] : 00000000 ->
```

A noter que la commande **pcc.l** prend pour argument :
- 0 => offset.
- ? => lecture avec modification possible.
- 7 => slot.

A ce moment là, on peut lancer le boot de la carte RIOC avec la commande **boot**.

Le hardware de la carte Trigger L2 est donc bien reconnu. Par contre pour pouvoir l'utiliser, il faut fabriquer un Driver Linux propre à notre carte afin de pourvoir lire ou écrire, dans les registres définis dans la partie précédente, depuis la carte RIOC.

1.5.1 Le driver cote kernel

Tout d'abord, la partie module qui d'écrit les 2 fonctions de base du driver, le point d'entrée et le point de sortie. Ces points d'entrée et de sortie sont nommés **init_module()** et **cleanup_module()**.

Ces 2 fonctions sont automatiquement appelées, lors du chargement et du déchargement du module, avec **insmod** et **rmmod** dans le noyau.

La fonction **init_module()** doit s'occuper de préparer le terrain pour l'utilisation de notre module (allocation mémoire, initialisation matérielle...).
Dans un 1er temps, elle cherche un *PCI_Device* dans le châssis répondant aux paramètres Vendor_ID et Device_ID précisé afin de remplir une structure de donnée de type pci_dev contenant une série d'informations sur ce module :

utilisation de la fonction : pci_get_device(vendor, device, pci_dev) Si un tel module est trouvé, il faut alors que l'utilisateur puisse avoir accès aux adresses des registres de chaque BAR. Sachant que le module est dans le mode noyau, il faut une passerelle entre le mode noyau et le mode utilisateur. Il s'agit de faire un mapping d'adresse entre l'adresse physique des registres au niveau du noyau et des adresses virtuelles de ces registres dans le mode utilisateur. On utilise alors la fonction **ioremap_nocache(phys_addr, size)**.

utilisation de la fonction : ioremap_nocache(phys_addr, size) Il faut utiliser cette commande pour chaque BAR et ainsi sera stockée dans une structure spécifique les 2 adresses de base de nos registres. Elles seront stockées dans la structure *TL2_address* de type *BAR_STRUCT* { *long Bar0 ; long Bar1 ;* }.

Nous avons écrit un driver en mode caractère pour permettre de dialoguer avec le périphérique. L'enregistrement du driver dans le noyau doit également se faire lors de l'initialisation du driver. Cette initialisation se fait lors du chargement du module, donc dans la fonction **init_module())**.

utilisation de la fonction : register_chrdev(major, device_name, ops)
- Major : Ce nombre majeur a pour but d'identifier la classe ou groupe auquel notre driver peut appartenir. Si on fixe la valeur 0, c'est le système qui lui affecte un numéro. Dans notre cas, on fixe sa valeur à 250 ce qui représente un driver en mode char en libre utilisation.
- Device Name : nom du driver : "*Hess2TriggerL2_*".
- fops : la structure de type file_operations où sont définis les méthodes d'appel système utilisable par notre périphérique : open(,,), close(,,), read(,,), write(,,), ioctl(,,)...

La fonction **cleanup_module()** doit quant à elle défaire ce qui a été fait par la fonction init_module(). De même, on supprimera le driver du noyau (déchargement du module dans la fonction cleanup_module())

utilisation de la fonction : unregister_chrdev(major, device_name) A COMPLETER

1.5.2 Le driver cote utilisateur

Les méthodes d'appel système :
Nous avons dû écrire des fonctions d'ouverture et de fermeture du périphérique, des fonctions de lecture, d'écriture et de control de ce périphérique. Toutes ces fonctions sont définies dans notre structure ops de type **file_operations**. Elles ont toutes des prototypes imposés dans lesquels on retrouve 2 nouvelles structures importantes : *struct inode* et *struct file*.
- La structure file définie dans $< linux/fs.h >$ représente un fichier ouvert et est créée par le noyau sur l'appel système **open()**. Elle est transmise à toutes fonctions qui agissent sur le fichier, jusqu'à l'appel de **close()**.
- La structure *inode* définie dans $< linux/fs.h >$ contient elle de nombreux paramètres caractérisant le périphérique.

La méthode open réalise ces différentes opérations :
– Initialisation du périphérique
– Identification du nombre mineur
– Gestion d'une table de réservation si plusieurs périphériques identiques sont utilisés

La méthode release est tout simplement le contraire :
– Libérer ce que open a alloué
– éteindre le périphérique
– Mettre à jour la table de réservation

N.B. : Le nombre mineur sert à différentier des périphériques identiques utilisant le même driver afin d'éviter les conflits.

La méthode read :
- Dans un premier temps, la fonction *read* effectue l'opération suivante : passage à l'espace mémoire du noyau (variable x) la valeur de ce qui est à l'adresse *addr* (mémoire utilisateur) $->$ appel à la routine **get_user(x, addr)** soit x = *contenu de addr*.
- appel à la routine système **y = readl(x)** pour lire la valeur contenue à l'adresse x.
- écrit la valeur de la variable y (espace mémoire du noyau) à l'adresse *addr* (espace utilisateur) – appel à la routine **put_user(y, addr)** pour effectuer cette opération.

La méthode write :
- Dans un premier temps, la fonction write effectue les passages à l'espace mémoire du noyau de l'adresse et de la valeur à écrire depuis la mémoire utilisateur $->$ appel à la routine **get_user (val, addr)** pour les 2 passages de données soit x = *val1* = *contenu de addr1* et y = *val2* = *contenu de addr2*.
- appel à la routine système **writel(x,y)** pour lire la valeur en y la valeur x.

La méthode ioctl :
- Récupération des adresses de base (Bar0 et Bar1) contenu dans la structure *TL2_address* de type *BAR_STRUCT* {*long Bar0 ;long Bar1 ;*}, structure qui a été initialisé et mise à jour lors de l'appel à la fonction **init_module()**.
- copie de ces données entre l'espace mémoire du noyau (kernel) et la mémoire utilisateur (user space) $->$ appel à la routine **put_user(x, addr)** pour effectuer cette opération qui écrit la valeur x à l'adresse *addr*.

Nous avons donc défini les méthodes de base de notre driver. Pour pouvoir l'utiliser, il faut maintenant le charger dans le kernel. Pour cela, nous avons écrit un script de nous exécutons sous le prompt **bash-3.00#**. Dans ce script, on retrouve l'instruction de chargement du module dans le kernel : Script « load »

```
\#!/bin/bash
\#************************************************************
\# file: load
\# install loadable driver in kernel
\#************************************************************
\# set -v
DRIVER=TrigL2driver
MAJNUM=252
insmod ./${DRIVER}.ko
mknod /dev/${DRIVER}_0 c $MAJNUM 0
chmod 666 /dev/${DRIVER}\_0
mknod /dev/${DRIVER}_1 c $MAJNUM 1
chmod 666 /dev/${DRIVER}\_1
mknod /dev/${DRIVER}_2 c $MAJNUM 2
chmod 666 /dev/${DRIVER}\_2
mknod /dev/${DRIVER}_3 c $MAJNUM 3
chmod 666 /dev/${DRIVER}\_3
```

On a le nom du driver *DRIVER=TrigL2driver* , le majeur *MAJNUM=252* et le mineur qui est le nombre variant de 0 à 3 (car on peut avoir j'usqu'à 4 cartes). C'est ce nombre qui va différencier les cartes TriggerL2 si plusieurs sont install´es dans le châssis. En fait, l'instruction **insmod** charge le fichier *nomdriver.ko* obtenu lors d'une compilation spécifique du fichier comprenant toutes les fonctions et méthodes détaillées ci-dessus.
Une fois le driver compilé et chargé, il faut créer son fichier spécial :
mknod /dev/$DRIVER_0 c $MAJNUM 0 avec c = mode charactère et 0 = mineur.
Ensuite, il faut lui donner les droits d'accès au fichier afin de pouvoir, lire, écrire et modifier :
chmod 666 /dev/$DRIVER_0
On recommence la même opération autant de fois qu'il y a de carte en n'oubliant pas de changer le mineur.
A l'exécution de ce script, on obtient :

```
bash-3.00# ./load
Physical Address BAR0 => 0x42000000
Physical Address BAR1 => 0x42100000
Virtual Address BAR0 => 0xe1080000
Virtual Address BAR1 => 0xe1058000
TrigL2driver_0 mapped   I/O_BAR0=0xE1080000 I/O_BAR1=0xE1058000 Rv=0
Slot Position: 7

TrigL2driver_Register device okay...found at least one device
Major Number Found: 252
```

On retrouve toutes les informations, adresse physique, adresse virtuelle (mappée), nombre majeur et même la position de la carte dans le châssis cPCI. Cela nous permet de vérifier que la carte est bien reconnue.

N.B. : Cet affichage est donné titre indicatif pour le cas où 1 seul carte TriggerL2 est présente dans le cPCI.
Pour décharger le driver, lancer le script « unload », il exécutera le fichier suivant :

```
#!/bin/bash
#************************************************************
# file: unload
# uninstall loadable driver from kernel
#************************************************************
# set -v
DRIVER=Hess2TriggerL2
FILE=TrigL2driver
MAJNUM=252
rmmod ${DRIVER}
rm /dev/${FILE}_0
rm /dev/${FILE}_1
rm /dev/${FILE}_2
rm /dev/${FILE}_3
```

A l'exécution de ce script, on obtient :

```
bash-3.00# ./unload
TrigL2driver_0 unmapped I/O_BAR0=0xE1080000 I/O_BAR1=0xE1058000
```

Une fois les adresses de bases connues et récupérées, nous avons toutes les informations pour accéder aux différents registres détaillés dans la « partie hardware − utilisateur ». Le tableau 3 donne la liste des registres et les adresses corrspondantes.

Tab. 3 – REGISTRES CONNECTEES AU USERDATA BUS

Addr Base + 0x100	Interne Registre	R/W	0x40
Addr Base + 0x104	Dipswicth Registre	R	0x41
Addr Base + 0x108	Led Registre	W	0x42
Addr Base + 0x10C	Ctrl Registre	R/W	0x43
Addr Base + 0x110	Jtag Registre (TDI,TCK,TMS)	W	0x44
Addr Base + 0x114	TDO Registre	R	0x45
Addr Base + 0x118	RdData1	R	0x46
Addr Base + 0x11C	RdData2	R	0x47
Addr Base + 0x120	WrData1	R/W	0x48
Addr Base + 0x124	WrData2	R/W	0x49
Addr Base + 0x128	WrData3	R/W	0x4A
Addr Base + 0x12C	WrData4	R/W	0x4B
Addr Base + 0x130	WrData5	R/W	0x4C
Addr Base + 0x134	WrData6	R/W	0x4D
Addr Base + 0x138	WrDac	R/W	0x4E
Addr Base + 0x13C	RdDac	R	0x4F
Addr Base + 0x140	Report Registre	R	0x50
Addr Base + 0x144	Test Registre (Fifo Test)	R	0x51
Addr Base + 0x148	FifoStatus	R	0x52
Addr Base + 0x14C	FifoInfo	R	0x53
Addr Base + 0x150	FifoData	R	0x54

1.6 Partie Software - Fonction Utilisateur

Nous allons maintenant voir la partie « fonction utilisateur », c'est-à-dire le programme principal **main()** pour utiliser la carte Trigger L2 ainsi que toutes les routines faisant appel aux méthodes système détaillés précédemment.

Dans un premier temps, ce programme à pour but de récupérer toutes les informations nécessaire pour utiliser notre carte Trigger L2, tout en particulier les adresses de base des registres afin de pourvoir lire et écrire dedans. Nous allons détailler les actions obligatoires que le main() doit réaliser pour un bon fonctionnement des cartes Trigger L2.

Au démarrage du programme, création de 2 variables globales essentielles :
Int gNumberTL2
− > compteur du nombre de carte Trigger L2 trouvées dans le châssis, valeur = -1 au démarrage.

TL2DATA_STRUCT *gpTL2[MAX_TL2]
− >tableau de pointeurs de structure des cartes Trigger L2 potentiellement présentes.
La structure contient les champs suivants :

- int nHandle ; −− > identifiant niveau fonction soft
- int nTL2DeviceHandle ; −− > identifiant niveau fonction kernel type open()
- long lBaseAddress0 ; −− > adresse de base 0
- long lBaseAddress1 ; −− > adresse de base 1
- int nInteruptID ; −− > non utilisé
- int nIntLevel ; −− > non utilisé
- char devname[64] ; −− > nom du device
- BOOL bInitialized ; −− > témoin d'initialisation

– BOOL bIntEnabled ; −− > non utilisé

N.B. : Dans notre cas, on défini la constante **MAX_TL2** qui à pour valeur 4 et qui détermine le nombre maximal de carte Trigger L2 que l'on peut trouver dans le châssis.

On trouve également un variable locale très importante : **Struct cblk**
− >structure contenant informations, paramètres et table des registres accessibles pour une carte Trigger L2, c'est en quelque sorte la carte d'identité locale de la carte Trigger L2.
Dans un premier temps, le programme appelle la routine *InitTL2Lib()* pour initialiser à 0 chaque pointeur « *gpTL2[i] » de chaque carte potentiellement présente dans notre châssis. Cette routine s'exécute uniquement si la globale gNumberTL2 vaut -1 ce qui signifie que l'on n'a encore jamais effectué cette initialisation de tableau. Tous les champs de cette structure sont donc créés et mis à 0.

Pour finir, la routine retourne la valeur 0 si tout c'est bien déroulé et positionne **gNumberTL2 = 0** pour éviter une seconde initialisation non désirée.

Ensuite, le programme appelle la routine *TL2Open(NUMCART, &c_block.nHandle, DEVICE_NAME)* pour ouvrir une instance pour la carte en fonction du nom du device « TrigL2driver_ » (DEVICE_NAME), de son numéro de mineur 0 (NUMCART) et d'un identifiant (handle) qui lui sera retourné et mis à jour.

- Création d'un pointeur ***pTL2** sur une structure type *TL2DATA_STRUCT*.
- Allocation d'une zone mémoire pour cette structure.
- Initialisation à 0 de tous ses champs.
- construction du nom du fichier spécial, celui créé par le scrip « load » : /dev/TrigL2driver_0.
- appel de la méthode système du driver : OPEN(nom fichier, option d'ouverture).
- open(..) retourne 0 si aucun problème détecté
- sauvegarde du nom de la carte dans sa structure type *TL2DATA_STRUCT*.
- récupération des adresses de base Bar0 et Bar1 par appel de méthode système ioctl(..)
- récupère ces adresses contenues dans la structure de type *BAR_STRUC* qui a été remplie à l'initialisation du module
- sauvegarde de ces adresses dans la structure type *TL2DATA_STRUCT* de la carte.
- appelle à la fonction *AddTL2(pTL2)*
  . ajout de l'adresse de la structure de la carte initialisée dans le tableau gérant les pointeurs sur les cartes soit : structure gpTL2{@ structure carte 0, 0, 0, 0}.
  . positionne un identifiant pour la carte trouvée.
  . incrémente le compteur de carte trouvée : **gNumberTL2**.
  . retourne 0 si aucun problème détecté.

L'instanciation de la carte s'est bien déroulée, sa structure est correctement remplie et elle détient toutes les informations nécessaire à son l'utilisation : identifiant et adresse de base en particulier.

Ensuite appel de plusieurs routines pour mettre à jour certains champs de la structure qui n'ont pas encore été correctement mis à jour :
- *TL2Initialize(identifiant de carte)* − > positionne à 1 le champ indiquant que la carte est bien initialisée.
- *GetTL2Address(identifiant de carte, BAR0, BAR1)* − > va chercher dans la structure les adresses de base pour les stocker dans la structure locale cblk.

Dans notre cas, la carte n'utilise que **BAR1** qui servira d'adresse de base à notre table de registre. On associera donc cette adresse de base récupéré par la routine *GetTL2Address()* à une nouvelle structure décrivant tous les registres. Cette structure de type *struct mapTrig* est constitué de tous les registres déclarés dans la « partie hardware − utilisateur », dans un ordre bien déterminé afin que les adresses de chaque registre correspondent au tableau 3. Cette structure *mapTrig* est incluse dans la structure *cblk* et prend pour adresse l'adresse de base bar1.

A ce moment du programme, toute l'initialisation, de la carte Trigger L2 qui a été trouvée, est finie et l'on va pouvoir lire ou écrire dans ses registres. Ces nouvelles fonctions vont être détaillées ci-dessous.

Lors de l'exécution du programme (lancer par la commande « ./Pilot »), l'affichage suivant apparait pour signifier qu'une carte Trigger L2 à bien été trouvée, initialisée et dont on a récupérer l'adresse de base Bar1, adresse qui sera utilisée pour tous les appels aux registres.

```
bash-3.00# ./Pilot
        Open Started....
Trigger L2 library initialized...
OPEN DEVICE OKAY....
Open instance of Trigger L2 Board successfully...
INIT DEVICE OKAY....
New Trigger L2 Bar1 Address Board: val:0xe1058000...
Initialisation ended...
```

L'accès aux registres se fait grâce à 2 routines de base, une fonction qui permet d'écrire dans un registre déterminé et une fonction pour pouvoir lire un registre. Pour écrire, l'utilisateur doit se servir de la routine :
long wrreg(struct cblk *c_blk, int reg, long value)
* struct cblk *c_blk − > structure qui représente la carte Trigger L2.
* int reg − > adresse du registre que l'utilisateur veut atteindre.
* long value − > la valeur à écrire dans le registre.

Pour lire, l'utilisateur doit se servir de la routine :
long rdreg(struct cblk *c_blk, int reg)
* struct cblk *c_blk − > structure qui représente la carte Trigger L2.
* int reg − > adresse du registre que l'utilisateur veut atteindre.

Les adresses des registres sont définies dans le fichier « trigl2.h » :

```
#define TRIGL2_USERREG 0x100
#define TRIGL2_DIPREG 0x104
#define TRIGL2_LEDREG 0x108
#define TRIGL2_CTRLREG 0x10C
#define TRIGL2_JTAG 0x110
#define TRIGL2_TDO 0x114
#define TRIGL2_RDDATA1 0x118
#define TRIGL2_RDDATA2 0x11C
#define TRIGL2_WRDATA1 0x120
#define TRIGL2_WRDATA2 0x124
#define TRIGL2_WRDATA3 0x128
#define TRIGL2_WRDATA4 0x12C
#define TRIGL2_WRDATA5 0x130
#define TRIGL2_WRDATA6 0x134
#define TRIGL2_WRDAC 0x138
#define TRIGL2_RDDAC 0x13C
#define TRIGL2_REPORTREG 0x140
#define TRIGL2_TESTREG 0x144
#define TRIGL2_FIFOSTATUS 0x148
#define TRIGL2_FIFOINFO 0x14C
#define TRIGL2_FIFODATA 0x150
```

Ces 2 routines de base font appel aux méthodes système « read » et « write » précédemment détaillés.

Il existe également plusieurs fonctions permettant de réaliser des actions plus complexes au niveau de la carte Trigger L2 :
* fonction JTAG.
* fonction RS232.
* fonction lecture d'une fifo.
* fonction RESET.
* procédure de démarrage de la carte.

1.6.1 JTAG

Le slow control permet à l'utilisateur de télécharger sans l'aide d'outils Xilinx n'importe quelle configuration dans n'importe lequel des FPGA Virtex 4 disposé sur la carte. Pour se faire, plusieurs conditions doivent être réunies :
- Disposer d'un fichier de type « maconfig.xsvf », ce fichier étant issu et obtenu comme expliqué sur La figure 21.
- Au niveau de la face avant de la carte Trigger L2, il faut que le switch$< 0 >$ (celui du haut) soit en position ON (sur la droite), voir figure 18.
- L'utilisateur doit ensuite choisir dans quel FPGA V4, il veut télécharger une nouvelle configuration. Pour cela, il dispose des bits 2,3 et 4 du Control Register *(cf. tableau1)*.
- L'utilisateur doit également positionner le bit$< 1 >$ á 1 du Control Register pour activer les multiplexeurs responsables du choix du routage vers le V4 séléctionné.
- Enfin, l'utilisateur doit faire une requête pour avoir l'autorisation de faire le téléchargement afin d'éviter d'endommager le système. Pour cela, il doit mettre á 1 le bit$< 8 >$ du Control Register. Avant toute nouvelle action, il doit attendre que le bit$< 1 >$ du Report Register passe á 1, ce qui lui donne l'autorisation de faire le téléchargement de la nouvelle configuration.

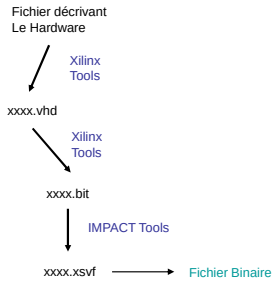

FIG. 21 – Obtention d'un fichier de configuration : il manque la partie .elf ainsi que la description de la chaine JTAG.

Une fois l'autorisation obtenue, au niveau software pur, l'utilisateur doit appeler la fonction microExecute(file, struct cblk) avec :

- struct cblk $->$ structure qui représente la carte Trigger L2.
- file $->$ nom du fichier « .xsvf, » à télécharger.

Cette fonction est présente dans le fichier « micro.c » qui est le sous-programme, fourni par xilinx et adapté par nos soins pour notre application. Cette partie est dédiée au transfert du fichier .xsvf via le CPCI dans le FPGA sélectionné.

1.6.2 Fonction lecture d'une fifo

Pour pouvoir lire les données contenues dans la fifo Data, l'utilisateur appelle la fonction « rdfifo » dont le prototype est : *long rdfifo(struct cblk c_blk, int ctrl)*
- struct cblk $->$ structure qui représente la carte Trigger L2.
- int ctrl $->$ valeur du Control Register.

Cette fonction s'assure que la fifo Info contient au moins une valeur en regardant le contenu du registre fifostatus. Si la valeur de ce registre indique que la fifo Info est vide, la fonction retourne la valeur 1. Sinon, la sequence suivante est appliquée :
- Enable de la fifo Info.
- Lecture de la valeur contenue dans la fifo Info.
- Disable de la fifo Info.

La valeur obtenue renseigne sur le nombre de mots á lire, c'est-á-dire le nombre de data utiles contenue dans la fifo Data. Il ne reste alors plus qu'á lire x fois (avec x = nombre de mot á lire) la fifo Data pour obtenir le bloc de données á récupérer. Ce bloc est décrit figure 13. Pour se faire, la fonction exécute x fois la séquence suivante :
- Enable de la fifo Data.
- Lecture de la fifo Data.
- Disable de la fifo Data.

La fonction retourne alors la valeur 2 qui signifie que tout s'est bien passé.

1.6.3 RESET

Comme nous l'avons déjá abordé précédement, il existe de nombreux Reset pour remettre dans de bonne condition de fonctionnement toutes les parties de la carte. D'une façon générale, chaque composante supporte 2 types de Reset, un reset destiné á sa partie hardware et un second reset pour sa partie Software. Pour exécuter un ou plusieurs Reset, il suffit d'écrire dans les bits adéquats du Control Register un 1 *(cf fig.7)*. L'utilisateur peut ensuite attendre plusieurs coup d'horloge pour être sur que la partie concernée se mette en « stand by » puis ecriture aux memes bits d'un 0 afin d'obtenir un redémarrage propre :
- Reset General VHDL écrire ***0x00000001*** dans le Registre.
  $->$ FSM du SP3 + interface S3_V4 du SP3
- Reset 3 MM1 écrire ***0x00020000*** dans le Registre.
- Reset 2 MM1 écrire ***0x00040000*** dans le Registre.
- Reset 1 MM1 écrire ***0x00080000*** dans le Registre.
- Reset 3 MM2 écrire ***0x00100000*** dans le Registre.
- Reset 2 MM2 écrire ***0x00200000*** dans le Registre.
- Reset 1 MM2 écrire ***0x00400000*** dans le Registre.
- Reset 3 MM3 écrire ***0x00800000*** dans le Registre.
- Reset 2 MM3 écrire ***0x01000000*** dans le Registre.
- Reset 1 MM3 écrire ***0x02000000*** dans le Registre.
- Reset 3 MM4 écrire ***0x04000000*** dans le Registre.
- Reset 2 MM4 écrire ***0x08000000*** dans le Registre.

- Reset 1 MM4 écrire ***0x10000000*** dans le Registre.
- Reset Soft V4 (PowerPC) écrire ***0x20000000*** dans le Registre.
- Reset Hard V4 (PowerPC) écrire ***0x40000000*** dans le Registre.
- Reset V4 (VHDL) écrire ***0x80000000*** dans le Registre.
- Reset V4 + All MM écrire ***0xB6DA0000*** dans le Registre.
- Reset MM écrire ***0x16DA0001*** dans le Registre.
- ALL Reset écrire ***0xB6DB0001*** dans le Registre.

1.6.4 Procédure de démarrage de la carte

Pour démarrer la carte de façon propre et afin que tous les organes s'initialisent correctement, il est nécessaire de procéder d'une manière bien précise en exécutant une séquence d'action décrite ci-dessous :
- RESET ALL.
- Ecriture des registres WRDATA1 à WRDATA6 avec les bonnes valeurs afin que la carte Trigger L2 soit initialisée correctement.
- Relâche du Reset Général.
- Démarrage de la machine d'état : mise à 1 du bit< 7 > du Control Register.
- Remise à 0 du bit< 7 > du Control Register : évite redémarrage de la machine d'état.
- Démarrage Processeur V4 : relâche du Reset Soft.
- Démarrage VHDL V4 : relâche du Reset Hard.

Dans ces conditions, la carte trigger L2 est prête pour toutes actions venant de l'utilisateur. Seul les mini-modules restent bloqués en état de Reset. Pour les débloquer, il faut relacher tous leurs Reset P et S.

1.6.5 Interpréteur de commande

L'interpréteur de commande est la partie du software qui va analyser les commandes disponibles pour l'utilisateur et exécuter la bonne action en fonction de la commande demandée. Ci dessous, sont listées, détaillées et expliquées toutes les commandes auxquelles l'utilisateur a accès.

SET/RESET FONCTION : *Mise à 1/0 de certains bits dédiés au Reset du Control Register permettant l'initialisation de tout ou partie de la carte TL2.*
Prototype : ...
- TRG TL2_SET SlotCarte Cible Type.
- TRG TL2_RESET SlotCarte Cible Type.

* Paramètres : ...
- SlotCarte : Position dans le châssis de la carte TL2 à piloter.
- Cible : ...
  . ALL −− > SET/RESET de toute la carte.
  . ALLV4 −− > SET/RESET de tous les FPGA de type Virtex4 : Kit et Mini module.
  . SP3 −− > SET/RESET du FPGA Spartan3.
  . KV4 −− > SET/RESET du Kit V4.
  . MM1 / MM2 / MM3 / MM4 −− > SET/RESET du Minimodule choisi.
- Type : ...
  . P −− > SET/RESET de la partie Périphérique : le VHDL
  . H −− > SET/RESET de la partie Software Processeur : Cas d'urgence - Hard RESET
  . S −− > SET/RESET de la partie Software Processeur : Cas normal - Sweet RESET

JTAG FONCTION : *Download par Jtag des configurations dans les V4 et/ou mémoires associées.*
* Prototype : TRG TL2_JTAG SlotCarte Cible File.
* Paramètres : ...
- SlotCarte : Position dans le châssis de la carte TL2 à piloter.

- Cible : ...
  . KV4 $--> $ Download vers le Kit V4.
  . MM1 / MM2 / MM3 / MM4 $--> $ Download vers le Minimodule choisi.
- File : Nom du fichier downloader " $----.xsvf$".

WRITE FONCTION : *Fonctions d'écriture dans les registres accessibles par le CPCI.*
* Prototype : TRG TL2_WRITE $< cmde >$.
* Paramètres $< cmde >$: ...
  - ACTIVELINES valeur1 valeur2 $--> $ descriptions des lignes actives de la camra :
    format [valeur1] : 0Xxxxxxxxx (32 bits).
    format [valeur2] : 0Xxxxxxxxx (32 bits).
  - TARGET Xc Yc coordonnes de la cible :
    format [Xc] : 0Xxxxx (16 bits).
    format [Yc] : 0Xxxxx (16 bits).
    – $--> $ concat des 2 mots en 32 bits ($Yc < 31..16 >, Xc < 15..0 >$).
  - THRESHOLD Seuil1 Seuil2 Coupure_COG format [Seuil1] : 0Xxx (8 bits).
    format [Seuil2] : 0Xxx (8 bits).
    format [Coupure_COG] :0Xxxxx (16 bits).
    – $--> $ concat des 3 mots en 32 bits ($COG < 31..16 >, Seuil2 < 16..8 >, Seuil1 < 7..0 >$).
  - DATA5 : valeur$--> $ en reserve au format 32 bits 0Xxxxxxxxx.
  - DATA6 : valeur$--> $ en reserve au format 32 bits 0Xxxxxxxxx.
  - CTRL : valeur$--> $ valeur du Control Register au format 32 bits 0Xxxxxxxxx.

READ FONCTION : *Fonctions de lecture des registres accessibles par le CPCI.*
* Prototype : TRG TL2_READ $< cmde >$.
* Paramètres $< cmde >$: ...
  - DATA1 $--> $ lecture du Register RDDATA1.
  - DATA2 $--> $ lecture du Register RDDATA2.
  - SWITCH $--> $ lecture du Dipswitch Register.
  - REPORT $--> $ lecture du Report Register.
  - CTRL $--> $ lecture du Control Register.
  - FIFO $--> $ lecture du registre connecté la FIFODATA.
  - ACTIVELINES $--> $ lecture des registres WRDATA1 et WRDATA2.
  - TARGET $--> $ lecture du registre WRDATA3 (format data cf. WRITE FNCT).
  - THRESHOLD $--> $ lecture du registre WRDATA4 (format data cf. WRITE FNCT).
  - DATA5 $--> $ lecture du Register WRDATA5.
  - DATA6 $--> $ lecture du Register WRDATA6.

N.B. : les valeurs lues sont stockées dans la structure *MAP_REG* contenue dans la structure *TL2DATA_STRUCT*.

FSM FONCTION : *Démarrage de la machine d'état gérant les échanges SP3$< -> $V4.*
* Prototype : TRG TL2_FSM.

DAC FONCTION : *Fonctions permettant de lire les diverses tensions d'alimentation ainsi que la température ambiante.*
* Prototype : TRG TL2_DAC $< cmde >$.
* Paramètres $< cmde >$: ...
  - BOARD_5V $--> $ lecture consommation au niveau de l'alimentation de la carte en 5V.
  - BOARD_3.3V $--> $ lecture consommation au niveau de l'alimentation de la carte en 3.3V.
  - KITV4_5V $--> $ lecture consommation au niveau de l'alimentation 5V du kit V4.
  - ALLV4_2.5V $--> $ lecture consommation au niveau de l'alimentation 2.5V des V4.

- ALLV4_1.2V $--> $ lecture consommation au niveau de l'alimentation 1.2V des V4.
- TEMPERATURE $--> $ lecture température.

MASTER FONCTION : *Fonctions permettant de changer la carte TL2 à piloter.*
* Prototype : TRG TL2_MASTER SlotCarte.
* Paramètres : ...
  - SlotCarte : Position dans le châssis de la carte TL2 à piloter.

STARTUP FONCTION :
* Prototype : TRG TL2_STARTUP_CONFIG $--> $ recharge fichiers dans FPGA et SDRAM.
* Prototype : TRG TL2_STARTUP_ACQUIRE $--> $ Exécute un test d'acquisition du système.

REGISTER FONCTION :
* Prototype : TRG TL2_REGISTER_AFFICH $--> $ Affiche le contenu des registres lisibles.
* Prototype : TRG TL2_REGISTER_UPDATE $--> $ Lecture de tous les registres lisibles.

Références

[1] Application notes 717, 901, 738 deal with the apu. the ppc user guide ug018 chapter 4 is also a great reference. *Xilinx Application Notes and user guides*.

[2] Avnet kit user guide and more (schematic, gerbers, mechanical) is included in the l2 documentation folder and are downloadable from the avnet website.

[3] Minimodules and base board user guides and schematics are included in the l2 documentation folder and are downloadable from the avnet website.

[4] Other virtex 4 fx 12 user guides : Ppc, pinout, etc.

[5] Rear io board schematics and ucf files are included in the l2 documentation folder.

[6] Spartan 3an documentation and evaluation board user guides and schematics are included in the trigger l2 documentation folder and are downloadable from the avnet and xilinx websites.

[7] Virtex-4 fpga user guide. ug070 (v2.6). *Communications on Pure and Applied Mathematics*, Dec 2008.

[8] S. Funk, G. Hermann, J. Hinton, D. Berge, K. Bernloehr, W. Hofmann, P. Nayman, F. Toussenel, and P. Vincent. The trigger system of the h.e.s.s. telescope array. 2007.

[9] Kraig Lund. Plb vs. ocm comparison using the packet processor software. *Xilinx Application Notes*, 57(11) :1413–1457, Aug 2004.

[10] P. Nayman. Interface compactpci. February 2001.

[11] P. Nayman. Note trigger camera h.e.s.s. ii. 57, 2007. A copy is included in the L@ documentation folder.

[12] Martin Tluczykont. Studies for a level 2 trigger for h.e.s.s. phase 2. *AIP Conf. Proc. HIGH ENERGY GAMMA-RAY ASTRONOMY : 2nd International Symposium on High Energy Gamma-Ray Astronomy*, 745(1) :785–790, Feb 2005.