ELECTRONIQUE POUR HESSII

Dernières modifications :
19/06/08 : Ajouts de programmation carte Time Stamp

P. Nayman LPNHE Paris nayman@in2p3.fr

Carte test analogique HESSII 5
Introduction 5
Les messages 7
Commandes internes 7
Les commandes externes 8
Les messages sur le BoxBus 10
La liaison RS485 12
Les signaux utilisés par la mémoire analogique SAM 13
Les signaux de contrôle écriture/lecture des données 13
La liaison SAMSer 13
Résumé des signaux utilisés par la mémoire analogique 15
Le Mode commun de SAM-ADC 15
Exemples de simulation 17
Temps mort 18
Le tiroir 19
Introduction 19
Les améliorations 19
Temps de traitement 19
Connexion carte Analogique/carte Slow Control 19
Bus Externes 21
Connexion du tiroir 23
Chaînage des FPGAs 25
Mode JTAG intégral 25
Mode FlashLoader 26
la carte Slow Control 28
Le bus donnée de la carte Slow Control 28
Le chaînage du Slow Control 28
Interface des signaux de contrôle des cartes analogiques 29
Les signaux du bus série 29
La carte arrière du tiroir 31
Introduction 31
Rappel sur la logique LVDS et le système de sécurité (Fail Safe) 31
Synoptique de la carte arrière 32
Aiguillage des données Slc 36
Chaînage général 39
Les Messages 40
Principe 40
La couche logique 40
Les différents types de message 41
Les messages d'acquisition 41
Les messages de Slow Control 43
Les constantes de conversion 45
Hautes Tensions 45
Thermomètres 46

Seuil Trigger 47
Haute tension HVmon 47
Courant HImon 48
Résumé des différents messages 58
Récapitulatif général des messages 59
Architecture du trigger pour HessII 71
Introduction 71
Description du trigger niveau 1 71
Description du trigger de niveau 2 72
Les contraintes liées au trigger 74
Contrainte liée au temps mort SAM 74
Contrainte liée au temps mort du fpga de lecture des Fifo 75
Transmission L2A/L2R 75
Carte Trigger Proto 76
Introduction 76
La carte Trigger proto 76
Programmation 77
Codage du mot DEVICE_ID fpga cPCI 78
Carte trigger 79
Introduction 79
programmation 81
La fenêtre de comptage 81
Les secteurs touchés et le compteur d'événements 82
Le circuit Delay-PréL2 82
RAZ 82
Mode Test 83
Codage du mot DEVICE_ID fpga cPCI 83
Carte Trigger Management 84
Introduction 84
Fonctionnalités de la carte TrgMngt 85
Les sources de déclenchement L1 85
La mise en temps 85
Le Prescaling 86
Fonctionnalités en service 86
Programmation 86
Codage du mot DEVICE_ID fpga cPCI 88
Exemple d'utilisation du logiciel de base sous Linux 2.6 88
Interface compactPci (cPCI) 90
Introduction 90
L'espace de configuration 91
Détails des registres utilisés dans l'espace de configuration 91
Les autres registres de l'espace de configuration (espace utilisateur) 96
Le bus de commande 98
Les cycles de configuration 98

Écriture mémoire 99
Écriture d'un mot 99
Écriture d'une rafale de mots 100
Écriture d'une rafale de mots avec état d'attente utilisateur 100
Lecture 101
Lecture d'un mot 101
Lecture d'une rafale de mots 102
Lecture d'une rafale de mots avec état d'attente utilisateur et démultiplexage 103
Lecture et écriture en mode 32 bits 103
Arrêt d'un transfert par l'utilisateur 103
Gestion de la parité 104
Le système d'interruptions 104
Le compteur utilisateur 105
Remarque sur les états d'attente 105
L'interface utilisateur 105
Type Interface 106
Données 106
Configuration de l'interface 107
carte fifo Cpci SLC 109
Introduction 109
Gestion des BoxBus-Slc 110
Les ordres en lecture 110
Gestion du taux de remplissage des fifos 112
Les ordres en écriture 112
L'interface Fifos, fpga cPCI et fpga FifoSlc 114
L'interface vers le BoxBus Slc 116
Les signaux sur le BoxBus Slc 117
Exemple de cycles de lecture 118
Programmation des Fpga 119
Codage du mot DEVICE_ID fpga cPCI 120
carte fifo Cpci DATA 121
Introduction 121
Gestion des BoxBus-Data 122
Les ordres en lecture 122
Les ordres en écriture 125
L'interface Fifos, fpga cPCI et fpga DataFifo 126
L'interface vers le BoxBus Data 127
Les signaux sur le BoxBus Data 128
Programmation des Fpga 128
Codage du mot DEVICE_ID fpga cPCI 129
Time Stamping 130
Introduction 130
Principe 130

Le Local Trigger Module (LTM) 131
Les principaux signaux 132
Sérialisation des données vers le Time Stamp esclave 133
Le Compteur synchrone 134
Programmation 134
Codage du mot DEVICE_ID fpga cPCI 135
carte Test PreL2..... 136
Introduction 136
Principe de la carte Test PréL2 138
Les ordres en lecture 138
Les ordres en écriture 139
Programmation des Fpga 140
Codage du mot DEVICE_ID fpga cPCI 141
Système de calibrage caméra..... 142
Introduction 142
Principe 142
Start/Stop 142
Fréquence 142
Délai 143
Amplitude 143
Programmation 144
Mode FlashLoader 144
Drivers cPCI..... 146
Introduction 146

CARTE TEST ANALOGIQUE HESSII

1. Introduction

Cette carte a pour but de tester un certains nombre de concepts pour l'électronique de la future caméra de *HESSII*. La conversion lumière-électrons est effectuée par des photomultiplicateurs. Cette carte est une version limitée de la version finale (2 Photomultiplicateurs au lieu de 8 pour la version finale).

La chaîne de base est composée de 2 voies analogiques (figure 1.1). La gamme dynamique (de 4000 à 5000) est réalisée en 2 gammes, la première de 1 à $100\gamma_e$ donne la précision pour étudier le photoélectron et les petits signaux alors que la seconde permet d'atteindre la dynamique maximale (de 50 à $5000\gamma_e$).

La première des deux voies est dédiée au bas gain et la seconde au grand gain. Les gains sont obtenus par des amplificateurs à sorties différentielles (*AD8351*). Chaque amplificateur pilote une voie de mémoire analogique (développement *CEA*). Les sorties de la mémoire analogique sont converties en numérique par un convertisseur à entrées différentielles (*AD9238*) de 12 bits. Chaque voie numérique est stockée dans une mémoire de type *fifo* synchrone de 1024 mots de profondeur (*IDT72V225*). Un *fpga* (*Cyclone EP1C12Q240C6*) gère l'ensemble des composantes et assure l'interface avec un processeur.

Les mémoires analogiques, *SAM* (Swift Analogue Memory), ont pour rôle de stocker les signaux *PM* en attendant une décision de la logique de trigger de niveau 1. Chaque mémoire est un buffer circulaire (256 cellules) échantillonné en permanence (*f>1GHz*).

Quand le trigger de niveau 1 (*L1A*) est reçu par la carte, l'échantillonnage des mémoires analogiques est arrêté. Le pointeur de lecture (*SAM* interne) est positionné en fonction d'un délai programmable (*Nd*) qui dépend en grande partie de la latence de la logique de niveau 1. A partir de ce pointeur, *Nf* données sont transmises à l'*Adc*. *Nf* est un paramètre programmable du *fpga*. Les données converties sont stockées dans les *fifos*. Les *fifos* permettent de relancer l'échantillonnage des mémoires analogiques dés que celles-ci ont été chargées. La logique de trigger de niveau 2 envoie, au système, pour chaque *L1A*, un *L2A* (Accept) ou un *L2R* (Reject). Dans le cas d'une logique *L2* fonctionnant correctement, le taux de *L2A* doit être environ de *3KHz* et le taux de *L2R* doit être environ de *100KHz*. Quand *L2A* est reçu, le *fpga* lit l'ensemble des données de toutes les *fifos*. Le nombre de mots total à lire est :

$N = Nf \times NbFifos$, pour une fenêtre de 20 mots et pour 16 *fifos*, il faut lire 320 mots. (Le temps total de lecture est inférieur à 10µs soit environ 30ns par mot). Quand un *L2R* est reçu, l'événement, en tête dans les fifos, doit être rejeté. Cette opération est effectuée en avançant le pointeur des *fifos* de *Nf* positions (simultanément sur toutes les *fifos*). Par exemple, pour une fenêtre de 20 mots et pour 16 *fifos*, il faut 645ns pour réaliser cette opération.

L'ensemble des données "*slow control*" est géré par des lignes séries. Une ligne série de type *RS485* (9600 *bauds*) entre le *fpga* et le processeur et une ligne série dédiée entre le *fpga* et chaque mémoire analogique (soit 8 lignes série).

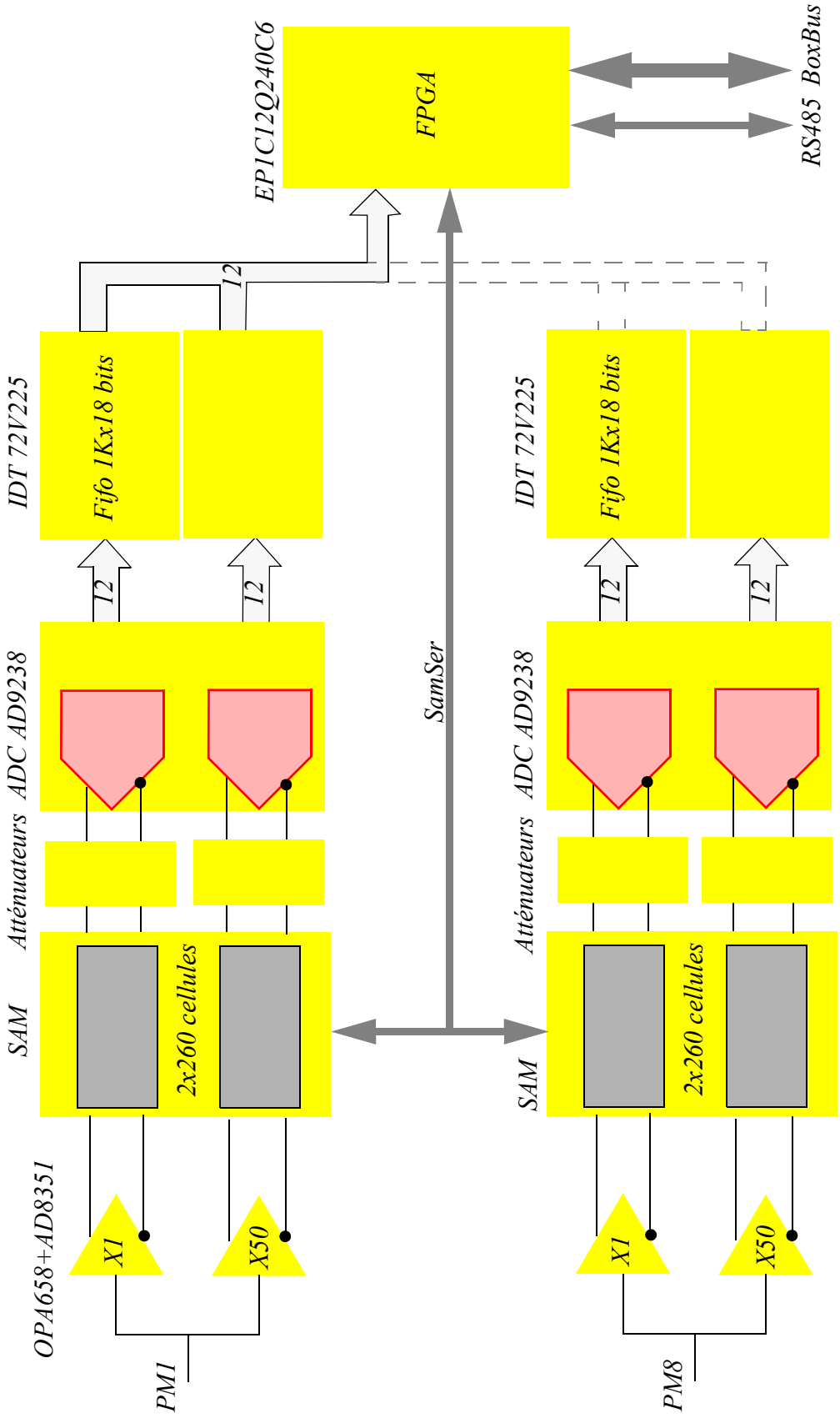


Figure 1.1 : Synoptique

2. Les messages

L'ensemble des commandes du système sont reçues par la ligne *RS485*. Comme la trame asynchrone ne comporte que 8 bits utiles, un message sera composé de plusieurs trames. Les commandes sont séparées en 2 groupes, les commandes internes et les commandes externes. Une commande interne modifie la stratégie du *fpga*. Une commande externe nécessite une action sur le lien série des mémoires analogiques (lecture ou écriture).

Les messages sont encapsulés, ils commencent toujours par un entête 0x"AA"[1] et se terminent toujours par un enqueue 0x"AA".

2.1. Commandes internes

Les commandes internes sont utilisées pour paramètrer le fonctionnement du *fpga*.

AA	*AA*	*AA*	*AA*
cmd = 1	*cmd = 2 à 5*	*cmd = 6*	*cmd = 7*
Nf[4:0]	*AA*	*Vmc[11:8]*	*AA*
AA		*Vmc[7:0]*	
		AA	

Tableau 2.1 : Les commandes internes

Le Tableau 2.1 montre les trois types de commandes internes possibles, selon la valeur de *cmd*.

Cmd	*Action*
1	Chargement *Nf*
2	Envoi message *DaqRdy* (au processeur)
3	Lecture données : mode Charge
4	Lecture données : mode Samples
5	Lecture données : mode Samples + Adresses physiques
6	Chargement de *Vmc*
7	Reset *SAM*

Tableau 2.2 : Le mot de commande

Le premier bloc (*cmd*=1), permet de définir la fenêtre de lecture pour les mémoires

1. 0x"nnnn" fait référence à une donnée en notation hexadécimale

analogiques, c'est à dire le nombre d'échantillons que le *fpga* doit lire pour un signal.

Le second bloc, ne contient pas de données, mais uniquement une commande, qui provoque la génération, par le *fpga*, d'un bloc *DaqRdy* sur le *BoxBus* (*cmd*=2).

Les blocs de données seront des charges (*cmd*=3) ou des échantillons (*cmd*=4). Si les adresses physiques sont disponibles, il est possible de compléter un bloc échantillons (*cmd*=5) par ces adresses. L'*ADC* convertit la valeur analogique sur une gamme de 2V pic-pic. On peut ajouter (ou retrancher) à chaque échantillon de charge une valeur donnée *Vmc*. Par défaut *Vmc*=+1*V* (0x"*7FF*").(*Voir " § 5. page 15 "*). Il est possible de programmer un *Reset* pour les mémoires SAM (*cmd*=7).

2.2. Les commandes externes

Les commandes externes permettent de pré-charger des blocs de données dans le *fpga* puis de le(s) transférer dans une ou plusieurs mémoires analogiques.

2.2.1. Ecriture d'un bloc en mémoire

La commande du Tableau 2.3 permet d'écrire un bloc de données dans la mémoire du *fpga*.

BB
#Bloc [0 à 63]
Data0 [15:8]
Data0 [7:0]

Data7 [15:8]
Data7 [7:0]
BB

Tableau 2.3 : Le bloc de données d'écriture

Il est possible d'écrire, en mémoire, jusqu'à 64 blocs de 8 données de 16 bits. Le numéro du bloc (entre 0 et 63) définit implicitement, son emplacement dans la mémoire interne (figure 2.1). La mémoire de stockage est dans ce cas de 512X16 bits.

Un message de ce type permet d'écrire une zone de 8 mots mémoire consécutifs. La figure 2.1 montre la position des blocs dans la mémoire du *fpga*

.

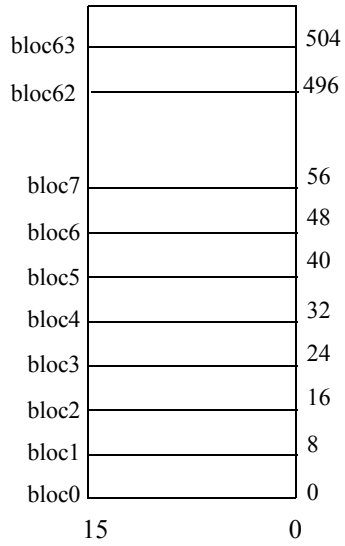


Figure 2.1 : Les blocs en mémoire

2.2.2. Ecriture série d'une mémoire analogique

Le bloc de commande du Tableau 2.4 (commande = 0xA) permet d'écrire, une zone de données de la mémoire du *fpga*, sur la ligne série d'une mémoire analogique de la carte considérée (une carte complète accueille 8 *SAM*).

CC	
A	
#SAM [0 à 7]	*0x[1X]*[a] *en broadcast*
Addr SER	*7 bits*
Start Addr	*StartAd[8]*
	StartAdd[7:0]
Stop Addr	*StopAd[8]*
	StopAd[7:0]
CC	

Tableau 2.4 : Le bloc de commande d'écriture

a. Il suffit que le bit 5 soit à 1

Le numéro de la mémoire *SAM* qui sera écrite est définie par le champ *#SAM[0 à 7]*. Si ce champ est 0x1X, alors toutes les mémoires sont écrites en même temps. *Addr Ser* (6 bits) est l'adresse qui sera envoyée sur la ligne série considérée.

La zone mémoire comprise entre *Start Addr* et *Stop Addr* (incluses) est transmise à la mémoire analogique définie précédemment (figure 2.2). L'adresse fournie sur le lien série sera *Addr Ser*.

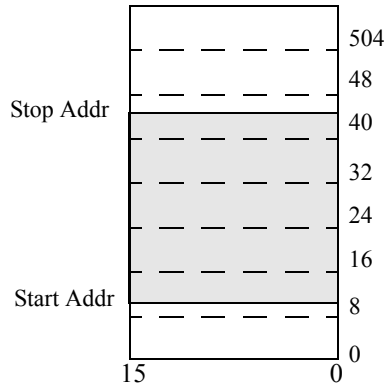


Figure 2.2 : Zone transmise sur le lien série

Si l'on considère que cette mémoire (512X16) gère 16 canaux de mémoire, la mémoire allouée pour chaque canal est donc de 512 bits.

2.2.3. Lecture série d'une mémoire analogique

Il est possible de lire une ligne série, d'une mémoire analogique, grâce au bloc défini dans le Tableau 2.5.

CC	
B	
#SAM [0 à 7]	
Adr SER	*7 bits*
Nb mots	*6 bits*
CC	

Tableau 2.5 : Le bloc de commande de lecture

La trame de donnée de *Nb mots* dont l'adresse est *Adr Ser* est lue de la mémoire analogique spécifiée *#SAM (*1 parmi 8). Il est possible de lire une trame possédant entre 16 et 1024 (16X64) bits.

Les données en provenance de la mémoire analogique sont stockées dans une RAM interne au *fpga* sousforme de mots de 16 bits. La partie correspondante de cette mémoire est ensuite transférée sur la ligne *RS485*. D[15:8] est envoyé dans une première trame, puis D[7:0] dans une seconde.

Les données envoyées sur la ligne *RS485* sont synchronisées par le protocole *XOn*/*XOff*. Pour transmettre sur la ligne série, il est nécessaire que le processeur envoie le caractère *XOn* (0x11). La transmission s'arrête en réception du caractère *XOff* (0x13). La ligne est automatiquement invalidée à la fin de transfert du bloc.

2.3. Les messages sur le BoxBus

Les messages sur le *BoxBus* sont un sous-ensemble de ceux disponibles pour l'expérience *HESSI*. Trois messages différents sont disponibles :

- *DaqSamples* 0x"*FFFE"*
- *DaqCharge* 0x"*FFFC"*
- *DaqRdy* 0x"*FFF8"*

Le message *DaqSamples* permet d'avoir accès à l'ensemble des échantillons de chaque signal de la carte. Le nombre de mots de données est *Nf*x16. Ce message est utile pour une étude détaillée des signaux (forme par exemple).

Le message *DaqCharge* donne la charge intégrée de chaque signal. Ce message perd l'information de forme mais est intéressant pour sa compacité.

Le message *DaqRdy* permet a une carte de s'identifier.

Le message *DaqSamples* transmet l'ensemble des échantillons vers le processeur, selon la requête initiale, ce message peut inclure ou non les adresses physiques des mémoires analogiques (Tableau 2.2, " : Le mot de commande", page 7) :

AAAA	*entête*	
FFFE		
Ident[7:0]		
data (0)	*SAM0*	
--	*SAM0*	
data (Nf-1)	*SAM0*	
data(0)	*SAM1*	
--	*SAM1*	
data(Nf-1)	*SAM1*	
--		
data(0)	*SAM15*	
--	*SAM15*	
data(Nf-1)	*SAM15*	
AD0(7:0)	*FCR0*	*optionnel*
--		*optionnel*
AD15(7:0)	*FCR15*	*optionnel*
AAAA		

Tableau 2.6 : Message DaqSamples

16x*Nf* données sont transmises sur le *BoxBus*. La taille du bloc est 16x*Nf*+*4*.

Le message *DaqCharge* permet de transférer la charge équivalente de chaque signal dans la fenêtre considérée. Ce mode de fonctionnement permet de réduire les données d'un facteur *Nf* (largeur de la fenêtre) , mais dans ce cas on perd l'information temporelle

sur le signal *:*

AAAA	
FFFC	
Ident[7:0]	
data (0)	*SAM0*
--	
data (15)	*SAM15*
AAAA	

Tableau 2.7 : Message DaqCharge

La taille du bloc *DaqCharge* est de 20 mots.

Le message *DaqRdy* :

AAAA
FFF8
Ident[7:0]
AAAA

Tableau 2.8 : Message DaqRdy

La taille du bloc *DaqRdy* est de 4 mots.

3. La liaison RS485

Le support de cette ligne série est une paire torsadée véhiculant des signaux différentiels.

La ligne utilisée doit fonctionner à 9600 *bauds*. La trame série est composée d'un bit de *Start* ("0"), 8 bits de données (*lsb-msb*) et d'un bit de *Stop* ("1").

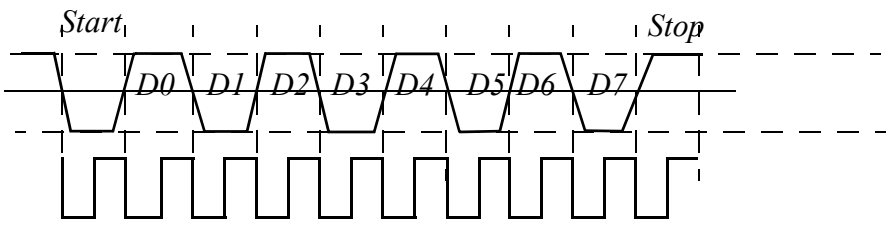


Figure 3.1 : La trame série RS485

La durée d'un bit à 9600 *bauds* est de 1/9600 =104.16µs.

Pour échantillonner correctement un bit, le *fpga* travaille à une fréquence de 16 fois 9600 *bauds* : *Clk16BR*=16*9600=153600Hz.

Si *k* est le rapport entre la fréquence de l'horloge de base (66.66MHz) et *Clk16BR : k=clk/clk16BR.*

Le *fpga* doit échantillonner chaque bit au milieu de sa valeur (à une fréquence double de 9600 *bauds*).

La division de fréquence nécessaire est dans ce cas : *ClkDiv=(k/2)-1*.

Remarque : Le protocole de la ligne série RS485 est essentiellement en mode caractère. Afin d'éviter une mauvaise compréhension par l'interface processeur, il est nécessaire de convertir chaque octet à transférer en 2 caractères. Dans l'autre sens, 2 caractères reçus formeront un octet. La conversion Hexa/Caractère est obtenue en ajoutant 0x30 à chaque code Hexa. De même la conversion Caractère/Hexa est obtenue en retranchant 0x30 à chaque caractère. Ces conversions sont codées dans le *fpga*.

4. Les signaux utilisés par la mémoire analogique SAM

4.1. Les signaux de contrôle écriture/lecture des données

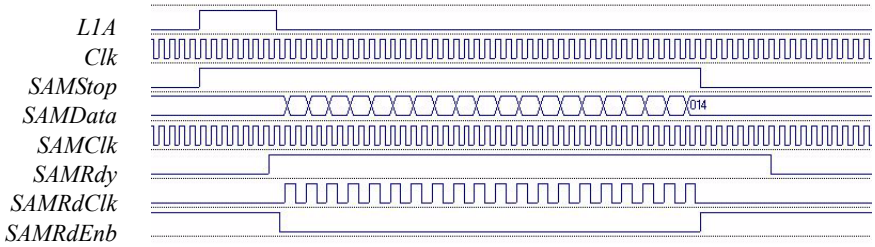


Figure 4.1 : Les signaux de gestion du circuit SAM

La figure 4.1 montre les signaux de contrôle utilisés par la mémoire analogique *SAM*.

Un trigger de niveau 1 (*L1A*) provoque l'arrêt de la mémoire analogique par le signal *SAMStop*. Quand la mémoire est prête à être lue, elle génère le signal *SAMRdy* pour la durée des échanges. Le signal *SAMRdEnb* permet de sortir les données en validant l'horloge de lecture *SAMRdClk*. Les données sont sorties avec le front avant de *SAMRdClk*. L'*ADC* échantillonne les données avec le front arrière de l'horloge. L'horloge de lecture *SAMRdClk* n'est présente que pendant la lecture.

4.2. La liaison SAMSer

La liaison *SAMSer* permet, au *fpga*, d'écrire et de lire des paramètres pour la mémoire analogique. Ce lien série est composé de 4 lignes : *SAMSerIn* pour entrer des données dans la mémoire, *SAMSerOut*, pour lire des données, *SAMSerEnb* pour valider la trame et *SAMSerClk* l'horloge de synchronisation.

Les données sont positionnées sur le front montant de l'horloge et prises en compte avec le front de descente.

En dehors de la trame, l'horloge n'est pas présente.

La trame est composée de la séquence :

[*r/wb*][*Ad6--Ad0*][*D15---D0*]---[*D15---D0*]

Le premier bit définit si l'on a affaire à un cycle de lecture *r/wb*=1 ou d'écriture *r/wb*=0.

[Ad6--Ad0] Adresse d'un registre

[*D15---D0*] [*D15---D0*]---[*D15---D0*] Données

En écriture il est possible de transmettre entre 16 et 512 bits. En lecture, il est possible d'acquérir de 16 à 1024 bits. Dans tous les cas, le nombre de bits des données est un multiple de 16.

Les 8 premiers bits sont toujours transmis sur la ligne *SAMSerIn.*

En écriture, la donnée est transmise, par le *fpga*, sur *SAMSerIn*.

En lecture, la donnée est transmise, par la mémoire, sur *SAMSerOut*.

La figure 4.2 montre un cycle d'écriture.

L'adresse est [*W*][*1010101*] écriture en 0x"55"

La donnée correspond à 3x16 bits soit : "*0006*", "*0007*" et "*0008*".

L'horloge *SAMRdSerCLk* a une période de 90 *ns*. Trois impulsions d'horloge sont présentent après la remontée de *SAMSerEnbi* (nécessaire au fonctionement de la mémoire).

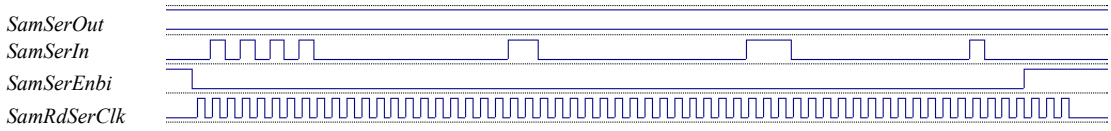


Figure 4.2 : Cycle d'écriture série, vue d'ensemble

La figure 4.3 montre quelques détails du cycle d'écriture. On remarquera que l'horloge d'écriture a une période de 90 *ns*.

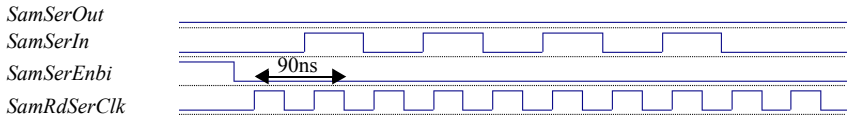


Figure 4.3 : Cycle d'écriture série, détails

La figure 4.4 montre un cycle de lecture, vue d'ensemble. On remarque que l'adresse est écrite sur *SAMSerIn* et que 4 mots de 16 bits sont lus.

Le lien série est actif que lorsque la ligne *SAMSerEnb* est à l'état bas.

L'adresse est [*R*][*1010101*] lecture en *0x"55"*.

La donnée est *0x"AA55D52AEA95754A"*

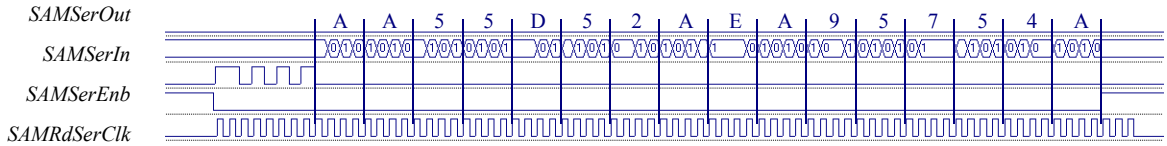


Figure 4.4 : Cycle de lecture série, vue d'ensemble

La figure 4.5 montre quelques détails des signaux de lecture. L'horloge à une période 90 *ns*.

La ligne série est également utilisée pour lire l'adresse physique de chaque canal *SAM*. Quand les données analogiques sont lues, les 8 bits de l'adresse physique sont transmis sur la ligne *SAMSerOut* à chaque impulsion de l'horloge *SAMRdSerCLk*.

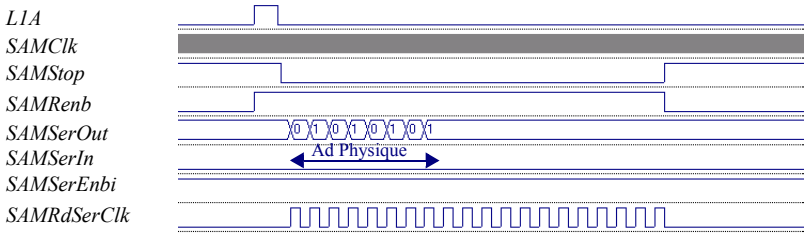


Figure 4.5 : Cycle de lecture de l'adresse physique

4.3. Résumé des signaux utilisés par la mémoire analogique

Le Tableau 4.1 résume les signaux utilisés par la mémoire analogique.

Signal	*Fonction*	*Caractéristique*
SAMClk	Horloge générale	période *15 ns*
SAMRdSerClk	Horloge de lecture, analogique et série	période *90 ns*
SAMRdEnb	Validation de *SAMRdClk*	Actif état bas
SAMStop	Stop et relance l'échantillonnage	Stop si haut Sampling si bas
SAMSerOut	Ligne série donnée sortie de *SAM*	MSB---LSB
SAMSerIn	Ligne série donnée entrée dans *SAM*	MSB---LSB
SAMSerEnbi	Validation de la ligne série i	Actif état bas

Tableau 4.1 : Caractéristiques des signaux SAM

5. Le Mode commun de *SAM-ADC*

Le signal issu de la mémoire analogique (*SAM*) est de type différentiel et varie dans la gamme [0,5V-2V] pour chaque sortie ($\Delta V = 1,5 Volt$). La gamme de variation diffé-

rentielle est donc de $2\Delta V = 3Volts$.

Il est nécessaire que la gamme de la mémoire analogique corresponde à celle de l'*ADC* : $3V \rightarrow 2V$.

Un atténuateur de 2/3 est nécessaire.

Dans ce cas, $[0, 5V–2V] \rightarrow \left[\frac{1}{3}V–\frac{4}{3}V\right]$

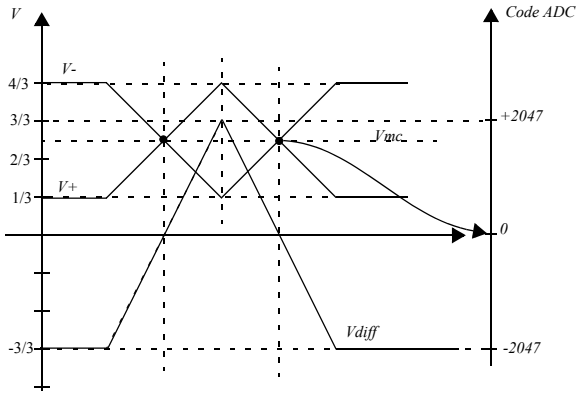


Figure 5.1 : Les signaux différentiels à l'entrée de l'ADC

La tension de mode commun correspond à $Vmc = \frac{1}{2}(V^+ + V^-)$ soit 5/6V.

Pour cette tension, le code fourni par l'*ADC* sera 0. Pour des valeurs inférieurs le code correspond à une tension différentielle négative, pour des valeurs supérieures, la tension différentielle est positive.

Au repos, la tension différentielle est de -1V. Cette valeur pose un problème pour le calcul de la charge du signal. En effet, la somme des lignes de base de chaque échantillon est : (-1V)x*Nf.* Cette valeur atteint environ -40960 pour une fenêtre de 20 échantillons.

Le *fpga* permet d'ajouter ou de retrancher un décalage nommé *Vmc* (voir commandes internes *cmd*=6).

Le cumul de la charge doit donc ramener la ligne de base de chaque échantillon autour de 0V, le *fpga* ajoute par défaut +1V (*0x"7FF"*).

Si un amplificateur de mode commun est utilisé entre la mémoire analogique et l'ADC, il est possible dans ce cas de décaler les niveaux de chaque entrée différentielle.

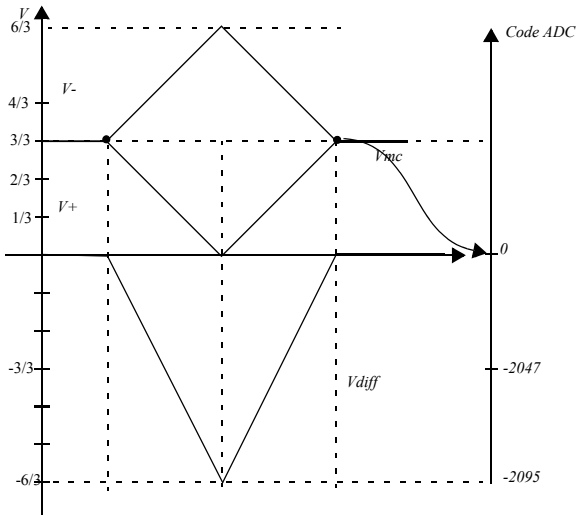


Figure 5.2 : Décalage du mode commun à l'entrée de l'ADC

6. Exemples de simulation

L'ensemble, carte et logique de trigger (*L1* et *L2*) est simulé en *VHDL*[1]. Le *fpga* est décrit en *VHDL* synthétisable de façon à être ciblé dans un *fpga Altera* de la famille *Cyclone*.

La gestion du trigger consiste à générer des *L1A* selon une loi de *Poisson* puis à appliquer une loi uniforme pour générer des *L2A* ou des *L2R*.

Figure 6.1 : Les signaux du trigger

La figure 6.1 montre une séquence avec les signaux du trigger. On remarquera que pour chaque *L1A*, un signal *L2A* ou *L2R* est généré. Ici, la logique du trigger *L2* est efficace, les triggers *L2A* sont rares et les rejets *L2* sont nombreux.

La figure 6.2 montre le chronogramme général des échanges entre les composants de la carte.

1. **V**ery High Speed Integrated Circuits **H**ardware **D**escription **L**anguage

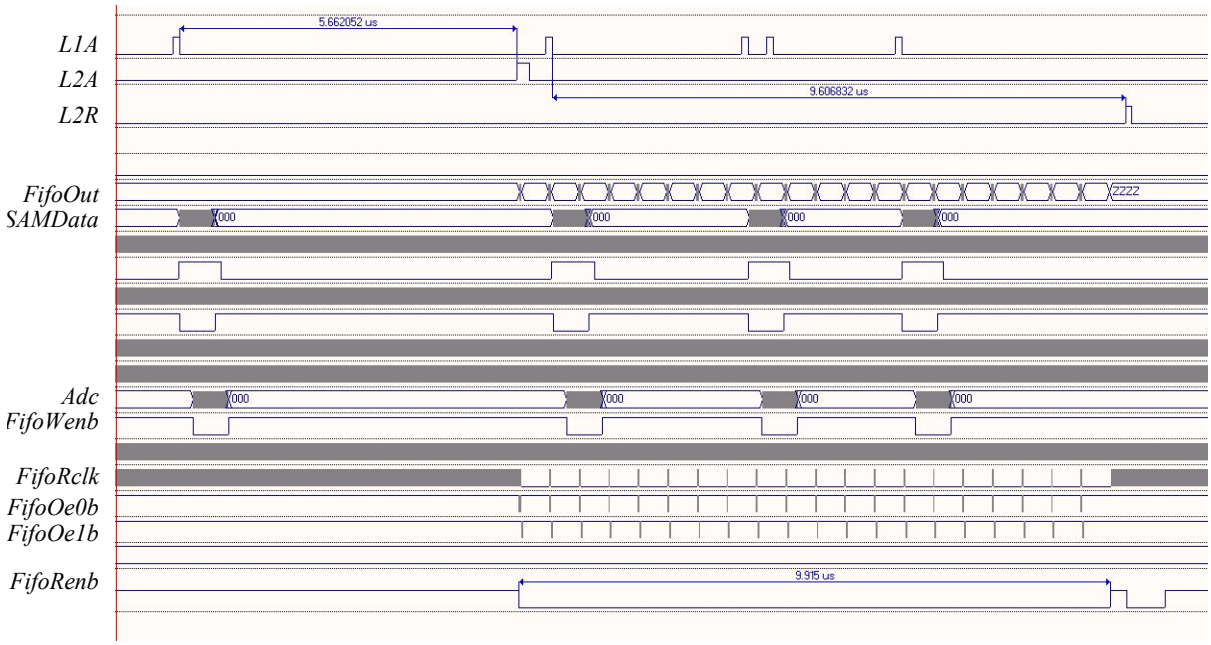


Figure 6.2 : Chronogramme général

Pour chaque *L1A*, la mémoire analogique est arrêtée, 20 données (*Nf* a été programmé à cette valeur) sont lues (*SAMData*), l'*Adc* convertit. On peut remarquer le pipeline de l'*Adc* de 8 coups d'horloge (léger décalage entre *SamData* et *Adc*).

Un *L2A* est généré 5,66 µs après *L1A*. Celui-ci force la lecture des *fifos* (*FifoRclk*).

La durée de la lecture *fifo* (*FifoRenb*) est d'environ 9 µs. Pour chaque *FifoRclk*, 16 mots sont lus (un par *fifo*), seuls *FifoOe0b* et *FifoOe1b* sont représentés. La durée totale de lecture de 9 µs correspond à la lecture de 20X16 mots par le *fpga*.

On peut remarquer que pendant cette lecture, 4 triggers *L1A* sont générés et 3 événements sont convertis et stockés dans les *fifos*. Un *L1A* qui se trouve dans le temps mort de lecture de la mémoire analogique (< 1.8 µs) ne peut pas être pris en compte.

Pour le second *L1A*, un *L2R* est généré 9,6 µs plus tard. En conséquence, le signal *FifoRenb* est activé pour 645 ns. Ce temps est nécessaire pour vider l'événement, simultanément, de toutes les *fifos*.

6.1. Temps mort

Les simulations mettent en évidence des problèmes de perte de trigger lors de certains temps mort :

• Un trigger *L1A* arrive pendant la lecture de la mémoire analogique. Ce trigger n'est pas pris en compte. Le temps d'occupation du couple *SAM-ADC* est de $[N_f \times 90]ns$ (1.8 μs pour une fenêtre de 20 échantillons). La logique de trigger *L1* doit interdire la génération de *L1A* séparé de moins de 2 µs.

• Un trigger *L2R* arrive pendant le vidage des *fifos* ($[N_f \times 16 + 15] \times 30ns$, soit environ 9,9µs pour 320 mots) dans le *fpga*. Ce trigger n'est pas pris en compte. Si un *L2R* doit être généré dans ce temps mort, la logique de trigger doit différer l'envoi de ce signal.

LE TIROIR

1. Introduction

Le tiroir *HESSII* est conceptuellement identique à celui de *HESSI*. Le synoptique du tiroir est représenté figure 2.3. Celui-ci est constitué de deux cartes *Analogiques* et d'une carte *Slow Control*.

Chaque carte analogique permet de traiter 16 canaux analogiques, signaux en provenance de 8 photomultiplicateurs. Pour des raisons de dynamique, chaque voie est gérée en deux gammes, petit gain et grand gain.

Pour chaque voie, le signal est échantillonné par une mémoire analogique[1] de 256 cellules, échantillonnée à 16 fois la fréquence de base (66MHz) soit environ 1GHz. La sortie peut être lue à 33MHz et convertie par un *ADC* de 12 bits. Une *fifo* de 1024 mots permet d'absorber les pointes d'événements en entrée (le flux d'entrée suit une loi de *Poisson*). Un *fpga Altera* de type *Cyclone* gère l'ensemble des composants de la carte.

2. Les améliorations

2.1. Temps de traitement

Les différents éléments de la carte permettent d'améliorer le temps de traitement d'un événement et ainsi limiter le temps mort. Voir " Figure 6.2 page 18 " montre que pour vider un bon événement (16X20 mots), dans le *fpga*, il faut moins de 10µs. Une logique de déclenchement de niveau 2 performante, basée sur le fait qu'il y a au moins 10 fois plus de rejets que d'acceptation, permet de diminuer considérablement le temps mort. Il suffit de 600ns pour supprimer le même événement.

2.2. Connexion carte Analogique/carte Slow Control

La gestion des échanges est améliorée en séparant les bus internes et en utilisant un protocole particulier pour les échanges entre cartes *analogiques* et carte *Slow control*.

De nombreuses informations *Slow control* concernent les cartes analogiques. La figure 2.1 montre le principe des échanges.Une donnée (du bus *DDBus*) peut être chargée dans le *fpga* de la carte 0 et/ou de la carte 1. L'interprétation du mot chargé est donnée par le bus *CmdAna*.

1. Swift Analogue Memory, conception CEA-Saclay

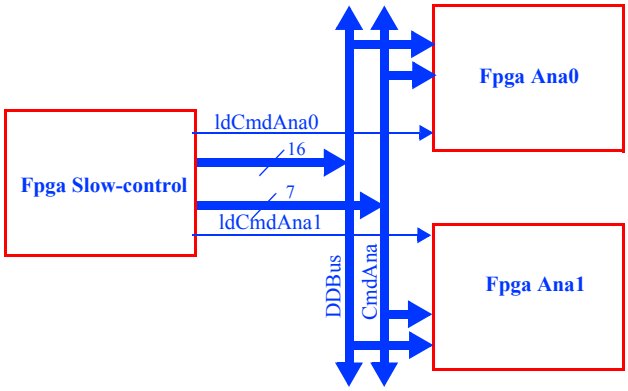


Figure 2.1 : Principe de chargement des paramètres entre les fpga

CmdAna [6:0]	*DDBus [15:0]*	*Commande*	*Message d'origine*
0000000		-	
0[0]10010 (12H) *0[1]10010 (32H)*	*8*	*Nd* *idem + génération DaqRdy*	*CNTRLSamNd*
0[0]10100 (14H) *0[1]10100 (34H)*	*3*	*#SAM* *idem + génération DaqRdy*	*CNTRLSamDac*
0[0]10101 (15H) *0[1]10101 (35H)*	*16*	*Control/Debug Register 1* *idem + génération DaqRdy*	*CNTRLSamReg*
0010110 (16H)	*16*	*Control/Debug Register 2*	*CNTRLSamReg*
0010111 (17H)	*16*	*Test_FCR*	*CNTRLSamReg*
0[0]00001 (01H) *0[1]00001 (21H)*	*8*	*Ident* *idem + génération DaqRdy*	*CNTRLDaq*
0[0]00011 (03H) *0[1]00011 (23H)*	*16*	*[X][X][RC][FAC][X][Nf][C/S][T0(1/0)][TOT(0/1)]* *idem +génération DaqRdy*	*CNTRLDaq*
0001000 (08H)	*12*	*VMC*	*CNTRLSlc*
0001001 (09H)	*12*	*Seuil TOT*	*CNTRLSlc*
1000000 (40H)	*16*	*[Dac1-L0][Dac1+L0]*	*CNTRLSamDac*
1000001 (41H)	*16*	*[Dac2-L0][Dac2+L0]*	*CNTRLSamDac*
....	...	...	...
1011110 (5EH)	*16*	*Dac1-L15][Dac1+L15]*	*CNTRLSamDac*
1011111 (5FH)	*16*	*[Dac2-L15][Dac2+L15]*	*CNTRLSamDac*

Tableau 2.1 : Encodage du bus commande

2.3. Bus Externes

Les bus de transfert des données d'acquisition et de *Slow control* sont séparés. Ce qui permet de simplifier la gestion informatique de la caméra.

Le *BoxBusII* est amélioré en vitesse, en utilisant des transferts parallèles (60ns/16 bits). Le *BoxBusII* utilise d'une part le standard *LVDS* (différentiel) et d'autre part est répété au niveau de chaque tiroir. Chaque tiroir ne voit qu'une partie du bus (distance entre 2 tiroirs). Seul le premier tiroir à un bus plus long (jusqu'à la carte *Fifo cPCI*).

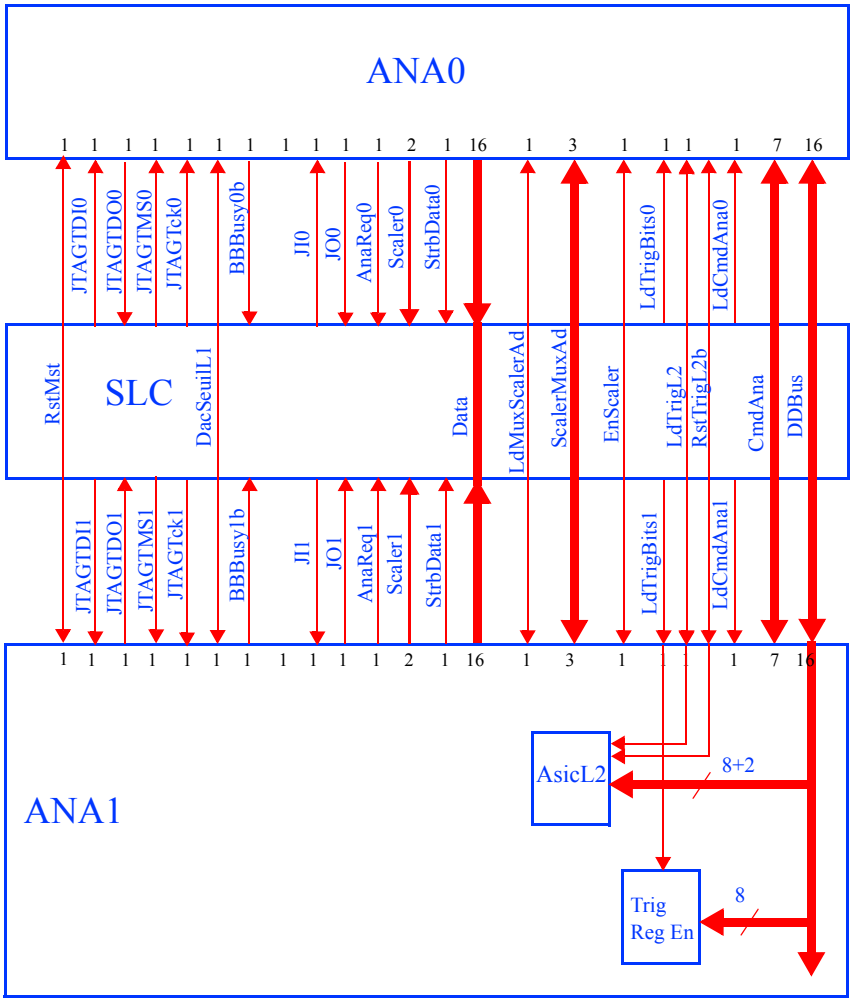


Figure 2.2 : Les différents signaux échangés entres les cartes analogiques et slow-control

Il faut noter que les bus *ScalerMuxAd* et *DDBus* sont concurrents, et peuvent être accédés simultanément.

Signal	*Usage*
JTAGTDI	Programmation Proms + Fpgas
JTAGTDO	Programmation Proms + Fpgas
JTAGTMS	Programmation Proms + Fpgas
JTAGTCK	Programmation Proms + Fpgas
JI	Jeton entrant
JO	Jeton sortant
AnaReq	Requête d'accès au *BoxBus*
Data [15:0]	Bus données vers le *BoxBus*
StrbData	Echantillonnage de *Data*
BBBusyb	Transfert actif sur le *BoxBus*
Scaler[a]	Sortie comparateur vers échelles
ScalerMuxAd [2:0]	Sélection d'une voie échelle parmi 8
LdMuxScalerAd	Chargement de *ScalerMuxAd*
EnScaler	Validation du mode échelle
DDBus [15:0]	Bus données inter-cartes
CmdAna [6:0]	Bus commande vers les fpgas Ana.
LdTrigBits	Chargement des bits de validation trigger à partir de *DDBus[7:0]*
LdTrigL2	Chargement Seuil et délai pour Asic *L2* à partir de *DDBus[9:0]*

Tableau 2.2 : Signaux à l'interface Slc-Ana

a. signal transmis en différentiel

2.4. Connexion du tiroir

Le tiroir est connecté aux bus par ses connecteurs et par la carte *Arrière*.

La carte Arrière accueille les différents connecteurs de bus, les répéteurs de signaux et l'identificateur du tiroir. L'identificateur (*IDENT*-8 bits) est indépendant du tiroir.

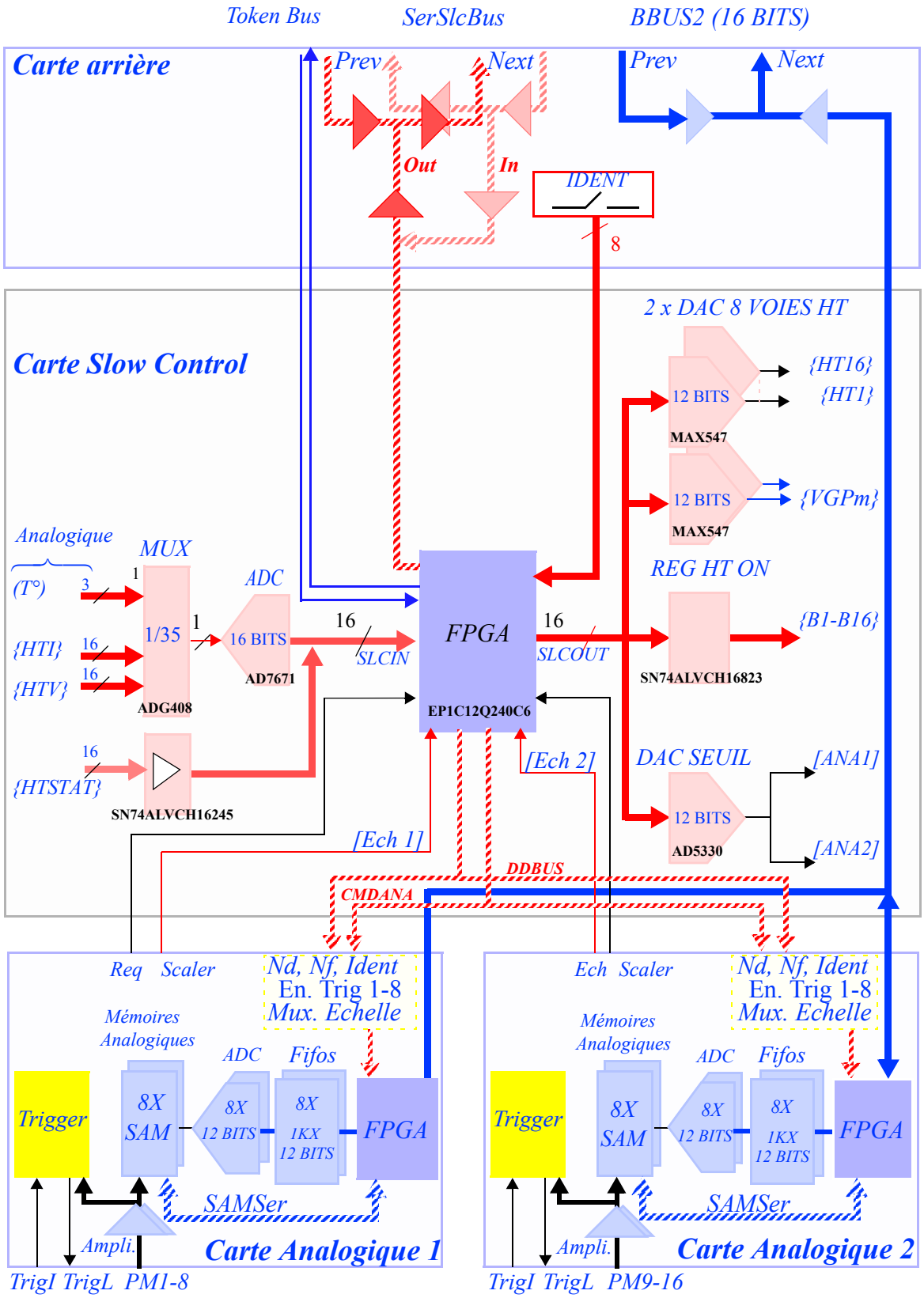


Figure 2.3 : Synoptique du tiroir HESSII

3. Chaînage des FPGAs

Chaque carte du tiroir est contrôlée par un *FPGA* dont le code de programmation est stockée dans une *EPROM*. A la mise sous tension, le contenu de la *EPROM* est vidé dans le *FPGA* qui lui est associé. D'autre part, il est nécessaire de pouvoir, éventuellement, re-programmer facilement une ou plusieurs *EPROM* du tiroir.

Afin d'éviter d'avoir un connecteur de programmation par couple *FPGA/EPROM*, il est préférable de chaîner les composants vers un seul connecteur. Pour faciliter l'accès, ce connecteur est situé sur la carte arrière du tiroir. Il est donc possible de programmer un élément sans démontage préalable du tiroir. Différents schémas de chaînage sont possibles. La programmation par le connecteur externe est prévue selon la norme *JTAG*[1].

Deux architectures différentes sont possibles, la première utilise des *EPROM JTAG* et la seconde des *EPROM* économiques programmables selon le mode *Active Serial*[2].

3.1. Mode JTAG intégral

Les *FPGA* et les *EPROM* associées sont interfacés *JTAG*. Ce mode permet d'accéder indépendamment aux *EPROM* et aux *FPGA*. Ce mode de chargement des *EPROM* est rapide mais nécessite des composants du type *EPC4*[3] (100 pins). Le vidage de L'*EPROM* dans le *FPGA* correspondant est prévu selon le mode *Passive Serial*[4].

Le système présenté sur la figure 3.1 chaîne les 6 composants et permet une programmation individuelle (*JTAG*) par l'intermédiaire du connecteur 10 points (HE10) situé sue la carte arrière du tiroir. Un câble (propriétaire) permet de charger des données d'un ordinateur.

1. Joint Test Action Group
2. Active Serial (standard *Altera*), le *fpga* (ou l'environnement) gère le chargement de l'EPROM.
3. *EPROM Altera*
4. Passive Serial (standard *Altera*), l'EPROM gère son chargement.

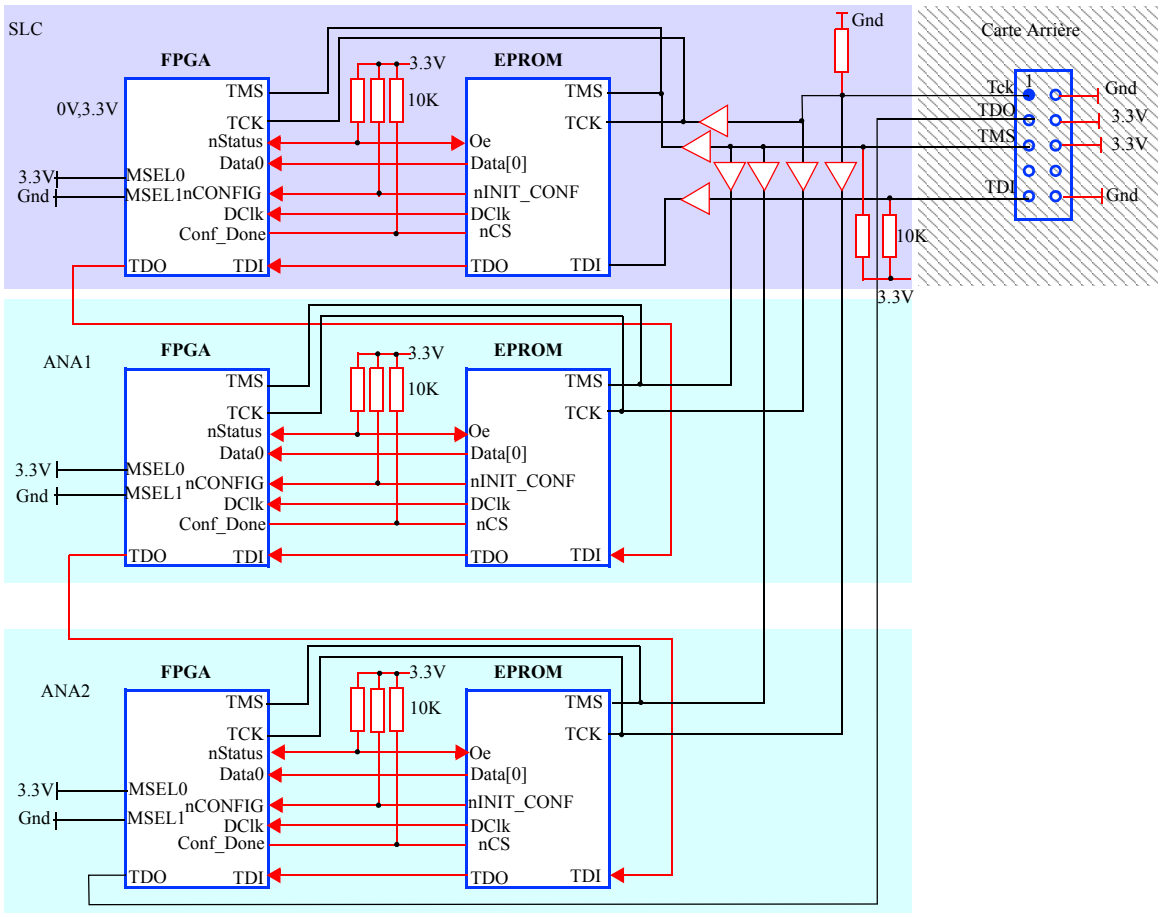


Figure 3.1 : Chaînage FPGA

3.2. Mode FlashLoader

Le mode *FlashLoader* est un mode de programmation des composants *Altera* (*EPROM*) économique. Ce mode autorise l'utilisation des *EPROM EPCS4*[1]. Dans ce mode l'*EPROM* est programmée par le *FPGA*, dans le mode *Active Serial*, et non plus par la chaîne *JTAG*. La figure 3.2 montre la connexion d'une *EPROM Active Serial*. La figure 3.3 explicite les connexions en *JTAG* et en mode *FlashLoader* des 3 *FPGA* du tiroir *HES-SII*.

1. *EPROM Altera* de type *Active Serial*

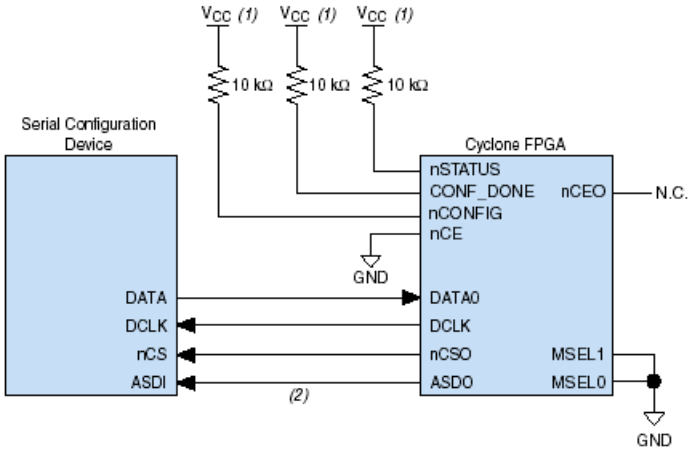


Figure 3.2 : Connexion d'une EPROM (Active Serial)

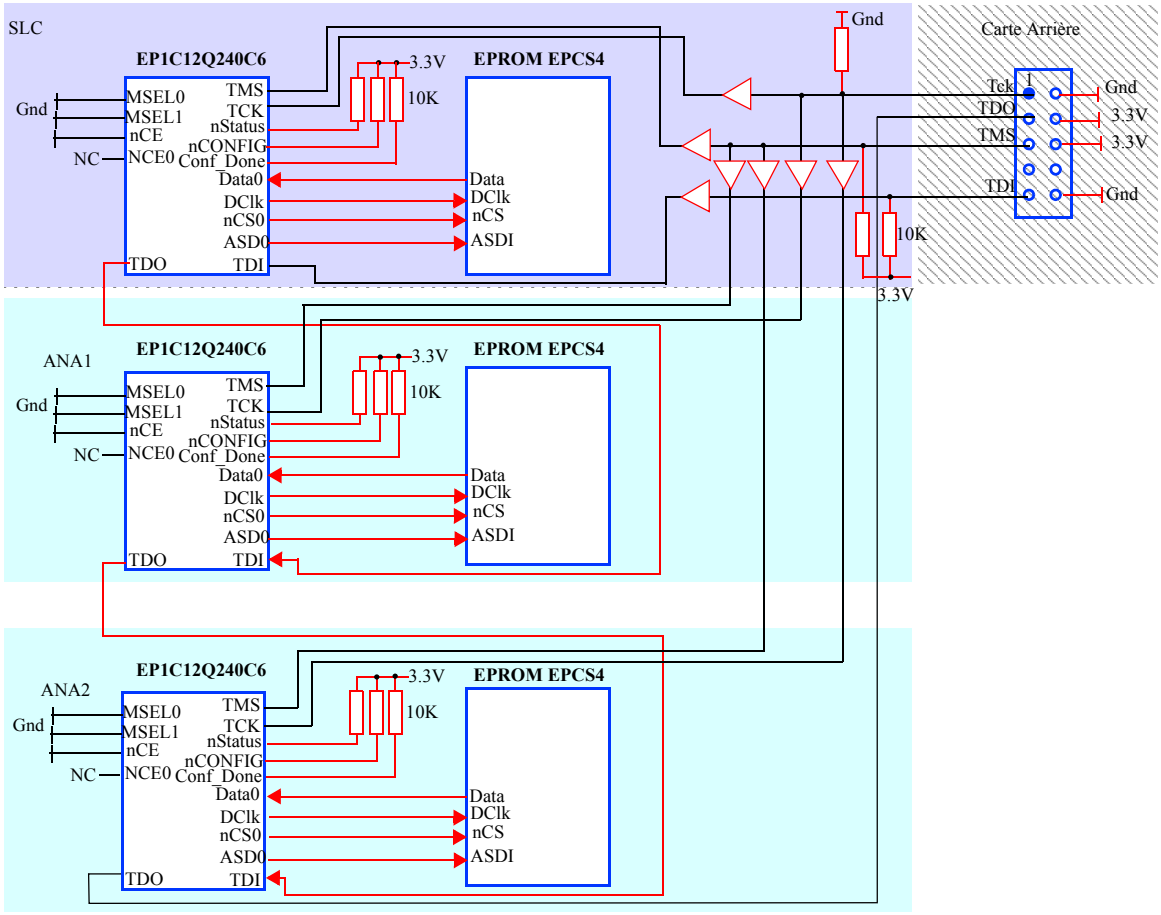


Figure 3.3 : Chaînage FPGA en mode FlashLoader

LA CARTE SLOW CONTROL

1. Le bus donnée de la carte *Slow Control*

La carte *Slow Control* permet de gérer tous les paramètres du tiroir. Ces paramètres peuvent être locaux à la carte *Slow Control* ou situés sur les cartes *Analogiques*.

Le processeur lit ou écrit ces paramètres par la ligne série *SBB* (*Serial BoxBus*).

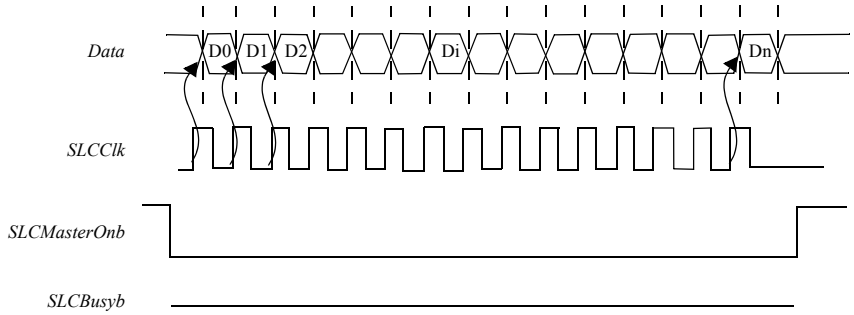


Figure 1.1 : SBB SLC

Les données peuvent être générées par le processeur (*SLCMasterOnb* actif) ou par l'un des tiroirs (*SLCBusyb* actif). Le module qui génère les données génère également les signaux de contrôle correspondants). Les données sont toujours émises avec le front montant de l'horloge et seront capturées avec le front arrière de la même horloge. Tous les signaux sont émis en logique différentielle *LVDS*, seul le signal *SLCReqb* est en logique *open drain*. Afin de limiter le nombre de fils sur le connecteur (*CON78*), la conversion en logique *LVDS* est effectuée sur la carte arrière.

2. Le chaînage du *Slow Control*

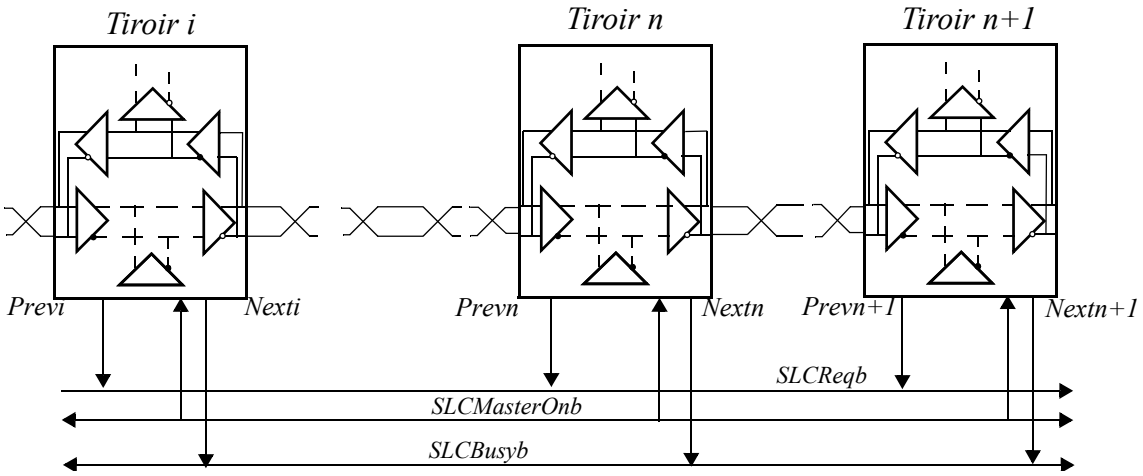


Figure 2.1 : Chaînage des tiroirs slow control

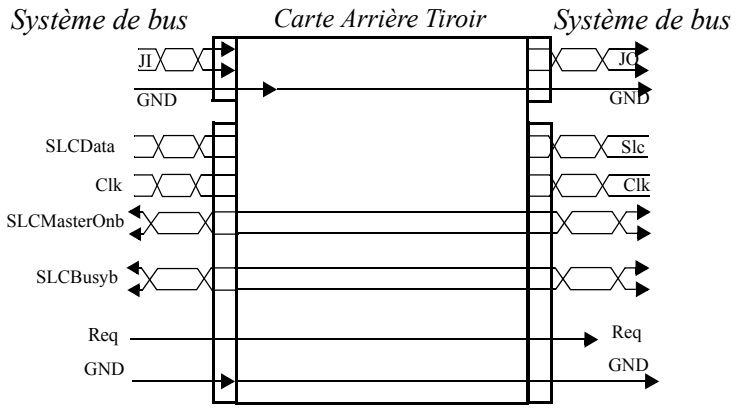


Figure 2.2 : Le tiroir et les connecteurs pour le slow control

3. Interface des signaux de contrôle des cartes analogiques

Les 2 cartes analogiques étant indépendantes, il est nécessaire de conditionner certains signaux. les signaux indiqués dans le tableau 3.1 sont concernés.

Signaux	*Origine*	*Signal conditionné*	*Destination*	*Utilisation*
AnaReq0	*Ana0*	*BBReqb*	*BoxBus*	*Requête de bus*
AnaReq1	*Ana1*			
BBBusy0b	*Ana0*	*BBBusyb*	*BoxBus*	*BoxBus occupé*
BBBusy1b	*Ana1*			
StrbData0	*Ana0*	*BBDClk*	*BoxBus*	*Prise en compte des données*
StrbData1	*Ana1*			
Initb	*BoxBus*	*RstMst*	*Ana0,1*	*RAZ*
JO0, JO1	*Ana0,1*	*BBJO*	*BoxBus*	*Chaînage des jetons*
BBJI	*BBJI*	*JI0, JI1*	*Ana0,1*	

Tableau 3.1 : Reconditionnement de certains signaux

Le reconditionnement de ces signaux est effectué dans le *fpga SLC*.
Le chaînage des jetons est effectué de la manière suivante :
- *BBJI* vers *JI0*,
- *JO0* vers *JI1*,
- *JO1* vers *BBJO*.

4. Les signaux du bus série

La figure 4.1 montre une partie de trame émise par la carte *slow control*. Il est nécessaire d'assurer un temps sufisant entre le front de descente de *SlcBusyb* et le front montant de l'horloge. La donnée (*SlcData*) est positionnée avec le front de montée de

l'horloge (*SlcClk*).

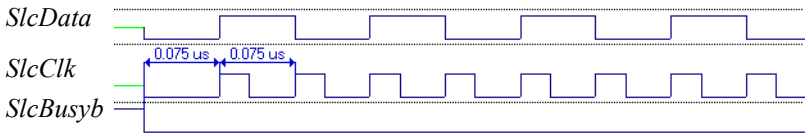


Figure 4.1 : Synchroniston des signaux en émission

LA CARTE ARRIÈRE DU TIROIR

1. Introduction

La carte arrière du tiroir permet à ce dernier d'accéder aux :
• bus de données,
• bus jetons,
• bus d'identification (*Ident*),
• bus JTAG (programmation fpga),
• alimentations.

La figure 1.1 montre le système arrière du tiroir et ses connecteurs.
Le tiroir est relié à la carte arrière par un connecteur *Cannon* 78 contacts.
Le *BoxBus* (données) est scindé en deux parties, *BBPrev* et *BBNext*. Chaque partie est accessible par un connecteur câble plat de 40 contacts.
Le bus dédié au "*Slow control*" est également scindé en *SlcPrev* et *SlcNext* (connecteurs de 20 points). *SlcNext* chaîne les tiroirs en direction de la *Fifo* et *SlcPrev* en s'éloignant de la *Fifo*.

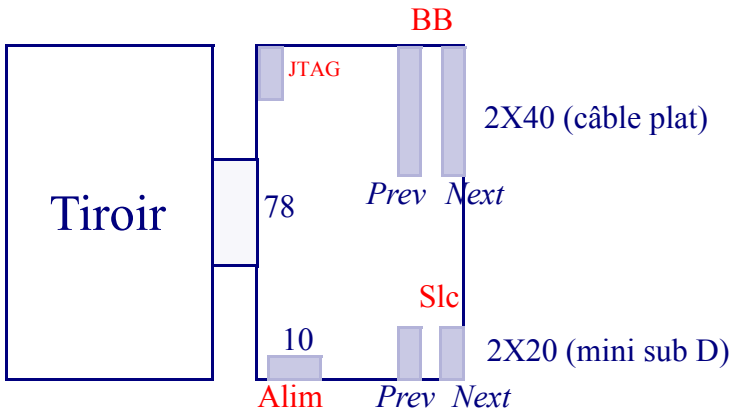


Figure 1.1 : Le système arrière du tiroir et ses connecteurs

2. Rappel sur la logique LVDS et le système de sécurité (Fail Safe)

La figure 2.1 donne le schéma de principe de la logique *LVDS*. Le driver émet un courant différentiel de 3.5mA qui crée une différence de potentiel de 350mV dans 100Ω. Cette tension différentielle est centrée autour de 1.2V (tension de mode commun). Les récepteurs *LVDS* possèdent un réseau de sécurité qui assure une marge de bruit comprise entre 20 et 35mV. Ce système, assure également un niveau haut en sortie du récepteur lorsque l'entrée est débranchée.
Dans le cas d'une application avec un câble, il est possible que le bruit différentiel dépasse la marge de bruit intrinsèque au récepteur.
Il est possible d'augmenter la marge de bruit intrinsèque (jusqu'à 100mV) par un réseau résistif externe.

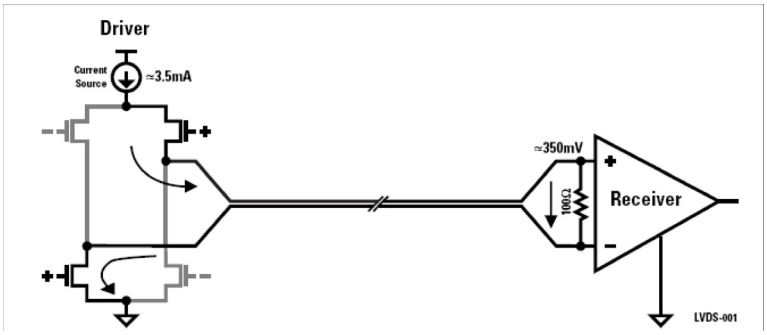


Figure 2.1 : Principe de la logique LVDS (d'après LVDS Owner's Manual - National Semiconductor)

La figure 2.2 montre un réseau de sécurité qui permet d'avoir une marge de bruit de 60mV. Pour ce schéma, le courant dans le réseau est :

$I_{FS} = 3.3V / 16.3K\Omega = 0.202mA$,

$V_{CM} = 0.2mA \cdot 6.15K\Omega = 1.23V$

$V_{FSB} = I_{FS} \cdot 300\Omega = 60mV$

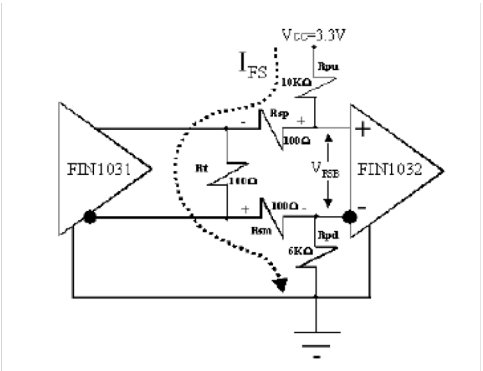


Figure 2.2 : Le réseau de sécurité Fail Safe (AN-5046 Fairchild)

La plupart des récepteurs LVDS, liés au transfert d'information de la carte arrière, sont câblés avec un réseau de sécurité Fail Safe (F.S).

3. Synoptique de la carte arrière

La figure 3.1 montre le synoptique de la carte arrière. Les signaux sont dans le cas général transmis en logique *LVDS*.

Le tableau 3.1 détaille les signaux des connecteurs *SLC*.

Le tableau 3.2 détaille les signaux des connecteurs *BoxBus* données.

Le tableau 3.3 détaille les signaux du connecteur 78 points utilisés pour la liaison avec le tiroir.

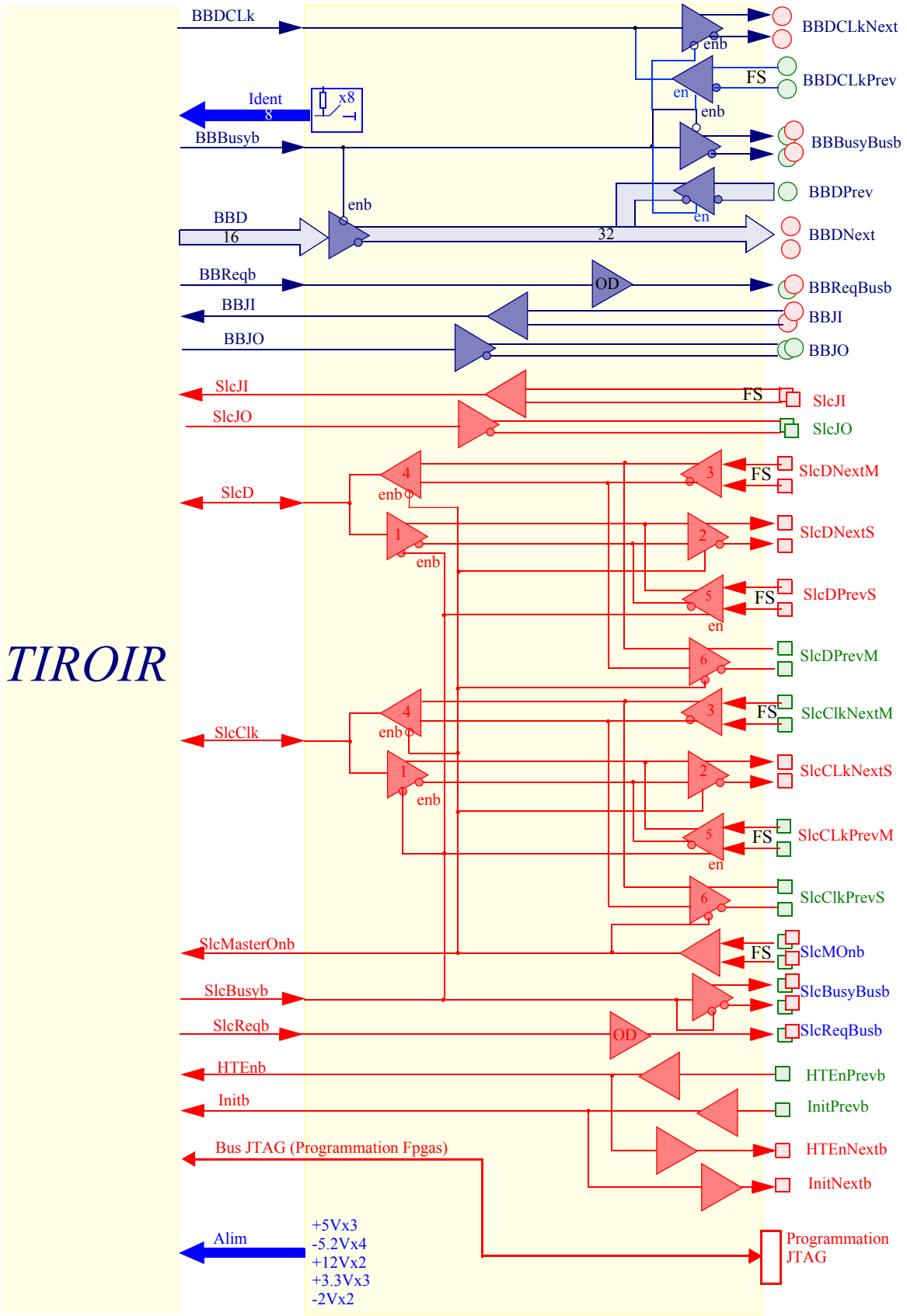


Figure 3.1 : Principe de la carte arrière

SlcNext	*SlcPrev*	*#fils*	
SlcDNextS	*SlcDPrevS*	*2*	
SlcDNextM	*SlcDPrevM*	*2*	
SlcCLkNextS	*SlcCLkPrevS*	2	
SlcCLkNextM	*SlcCLkPrevM*	2	
SlcMOnb	*SlcMOnb*	2	Bus
SlcBusyBusb	*SlcBusyBusb*	2	Bus
SlcReqBusb	*SlcReqBusb*	1	Bus (OD)[a]
SlcJI	*SLCJO*	2	
SLCInitb	*SLCInitb*	1	Bus
SlcHtEnb	*SlcHtEnb*	1	Bus

Tableau 3.1 : Connecteurs SlcNext et SlcPrev

a. Open Drain

BBNext	*BBPrev*		
BBBusyBusb	*BBBusyBusb*	*2*	*Bus*
BBDNext	*BBDPrev*	16x2	
BBDClkBus	*BBDClkBus*	2	Bus
BBReqBusb	*BBReqBusb*	1	Bus (OD)[a]
BBJI	*BBJO*	2	

Tableau 3.2 : Connecteurs BBNext et BBPrev

a. Open Drain

Connecteur 78 contacts		
GND	12	
Ident	8	*Adresse physique du tiroir sur le bus*
BBBusyb	1	*Bus actif*
BBD	16	*Données*
BBDClk	1	*Echantillonnage de la donnée*
BBReqb	1	*Requête du bus*
BBJI	1	*Jeton In BoxBus*
BBJO	1	*Jeton Out BoxBus*
SlcJI	1	*Jeton In Slc*
SlcJo	1	*Jeton Out Slc*
SlcD	1	*Donnée Slc*
SlcClk	1	*Horloge Slc*
SlcMasterOnb	1	*Cycle maître Slc actif*
SlcBusyb	1	*Cycle tiroir Slc actif*
SlcReqb	1	*Requête Slc*
HTEnb	1	*Validation HT*
Initb	1	*Initialisation tiroir*
+5V	3	*Alim*
GND	3	*Alim*
-5.2V	4	*Alim*
GND	4	*Alim*
+12V	2	*Alim*
GND	2	*Alim*
+3.3V	3	*Alim*
GND	3	*Alim*
-2.2V	2	*Alim*
GND	2	*Alim*

Tableau 3.3 : Connecteur liaison tiroir

4. Aiguillage des données Slc

• La figure 4.1 montre comment les données "*Slow control*" sont aiguillées. Trois sources de données sont possibles. Le chiffre entre crochets [*i*] fait état de la validation du buffer *i* de la figure 4.1 :

• Le maître transfère des données dans un ou plusieurs tiroirs (*SlcNextM* [3], *SlcPrevS* [6], *SlcD* [4] actifs),

• Le tiroir (local, *SlcD* [1] et *SlcNext* [2] actifs) transfère des données vers le maître,

• Un autre tiroir (bus *SlcPrevM* [5] et *SlcNextS* [2] actifs) transfère des données vers le maître.

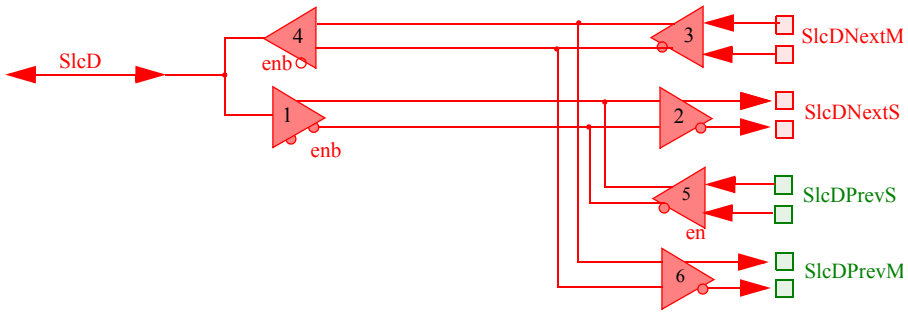


Figure 4.1 : Chemin de données selon l'échange

L'intérêt d'un tel système d'aiguillage est de réduire la charge du bus. Chaque tiroir est chargé par un nombre limité de buffers (2 au plus) et par un câble court (câble entre 2 tiroirs). Seul le premier tiroir est chargé par un câble long (tiroir-fifo).

La validation des buffers se fait selon la logique suivante :

• *Buffer 1* : validation si *SlcBusyb*='0'	Local vers Maître
• *Buffer 2* : validation si *SlcMasterOnb*='1'	Local et non Local vers Maître
• *Buffer 3* : validation permanente	
• *Buffer 4* : validation si *SlcMasterOnb*='0'	Maître vers tiroirs
• *Buffer 5* : validation si *SlcBusyb*='1'	Non local vers Maître
• *Buffer 6* : validation si *SlcMasterOnb*='0'	Maître vers tiroirs

La figure 4.2 illustre ce principe et montre le cheminement des données *Slc* du tiroir *i*-1 vers le maître (3 tiroirs sont représentés). La figure 5.1 montre le cheminement des données du maître vers les tiroirs. Le tiroir adressé (ou les tiroirs en cas de broadcast) prendra en compte les données.

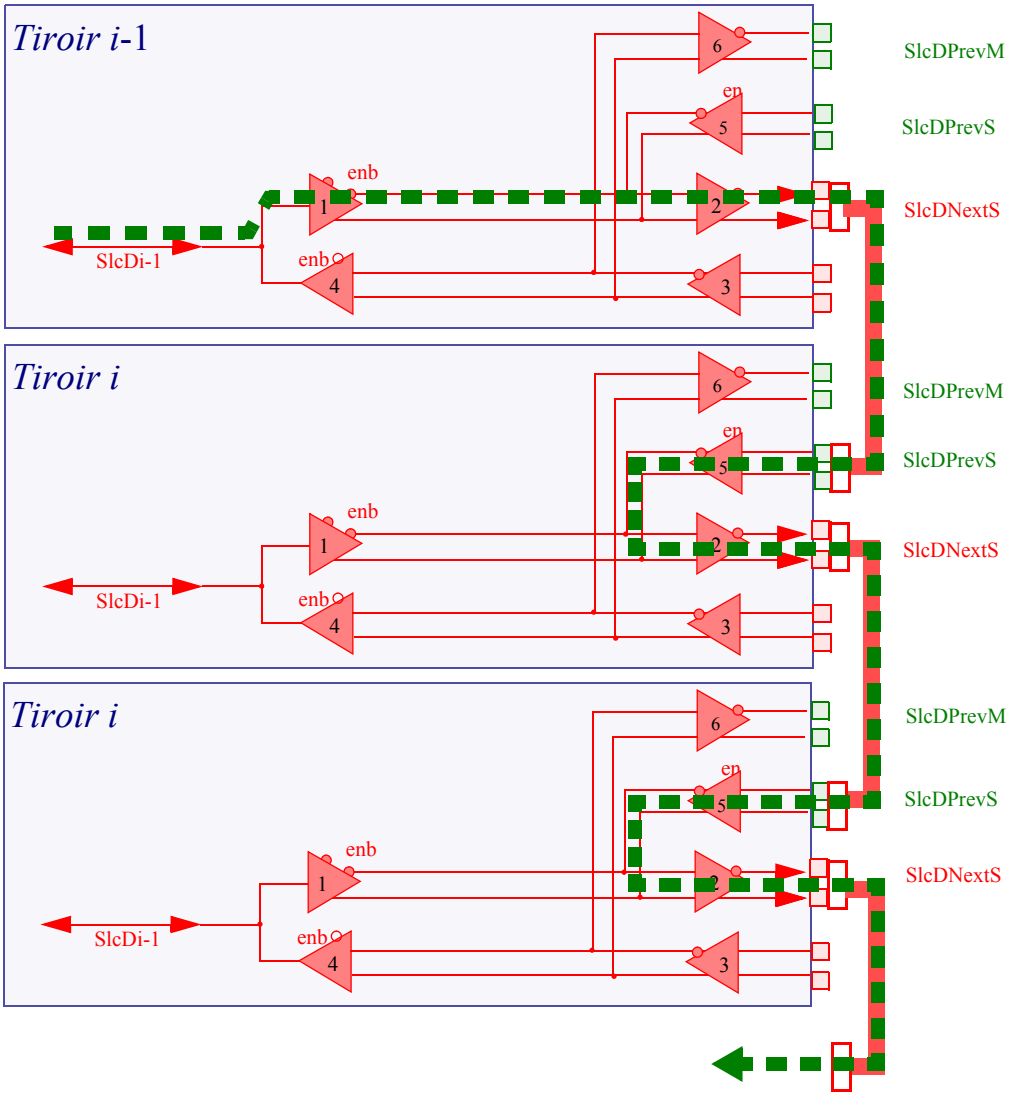


Figure 4.2 : Chaînage des bus Slc (tiroir vers maître)

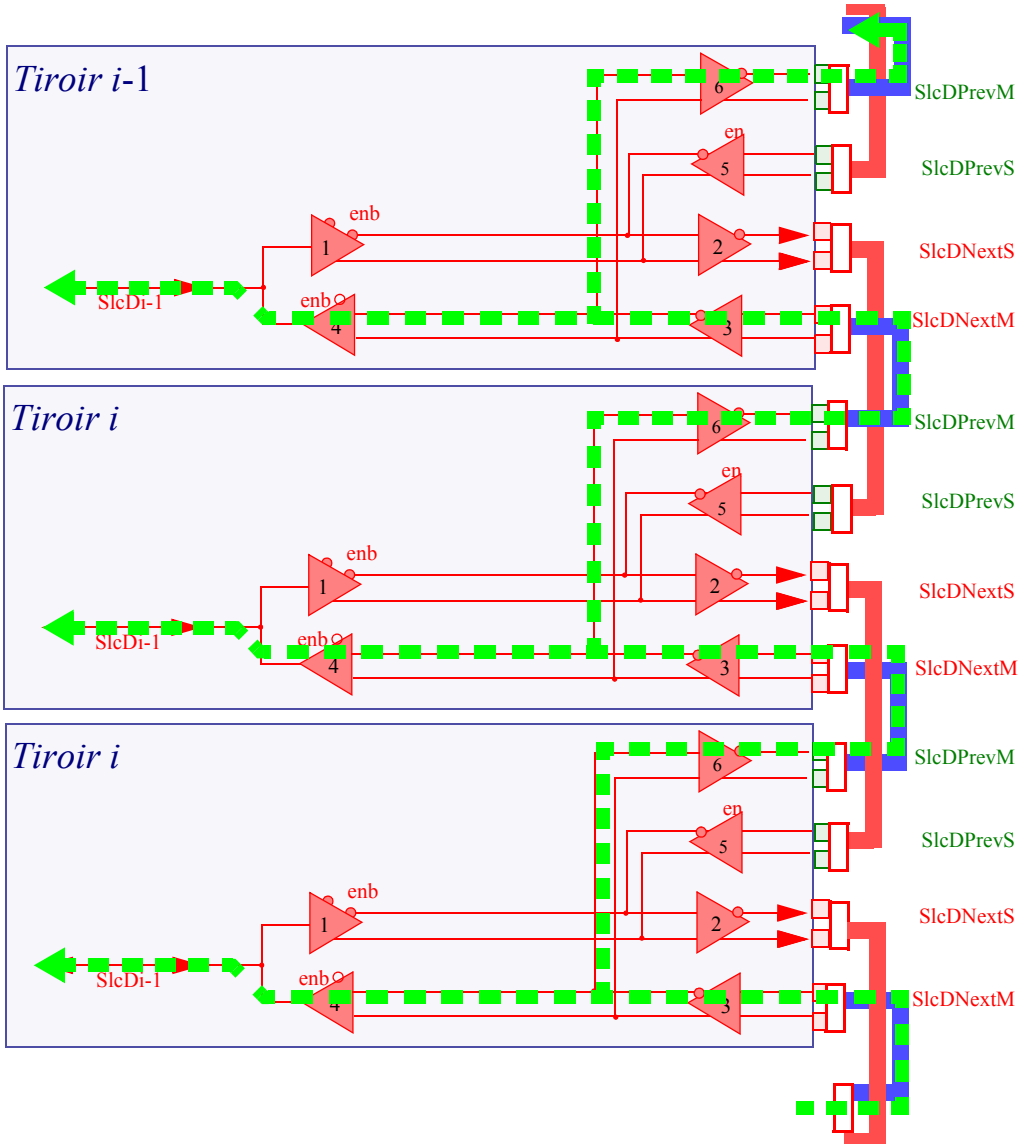


Figure 4.3 : Chaînage des bus Slc (maître vers tiroirs)

5. Chaînage général

La figure 5.1 montre le chaînage général des différents bus des tiroirs ainsi que les signaux reliés par les câbles.

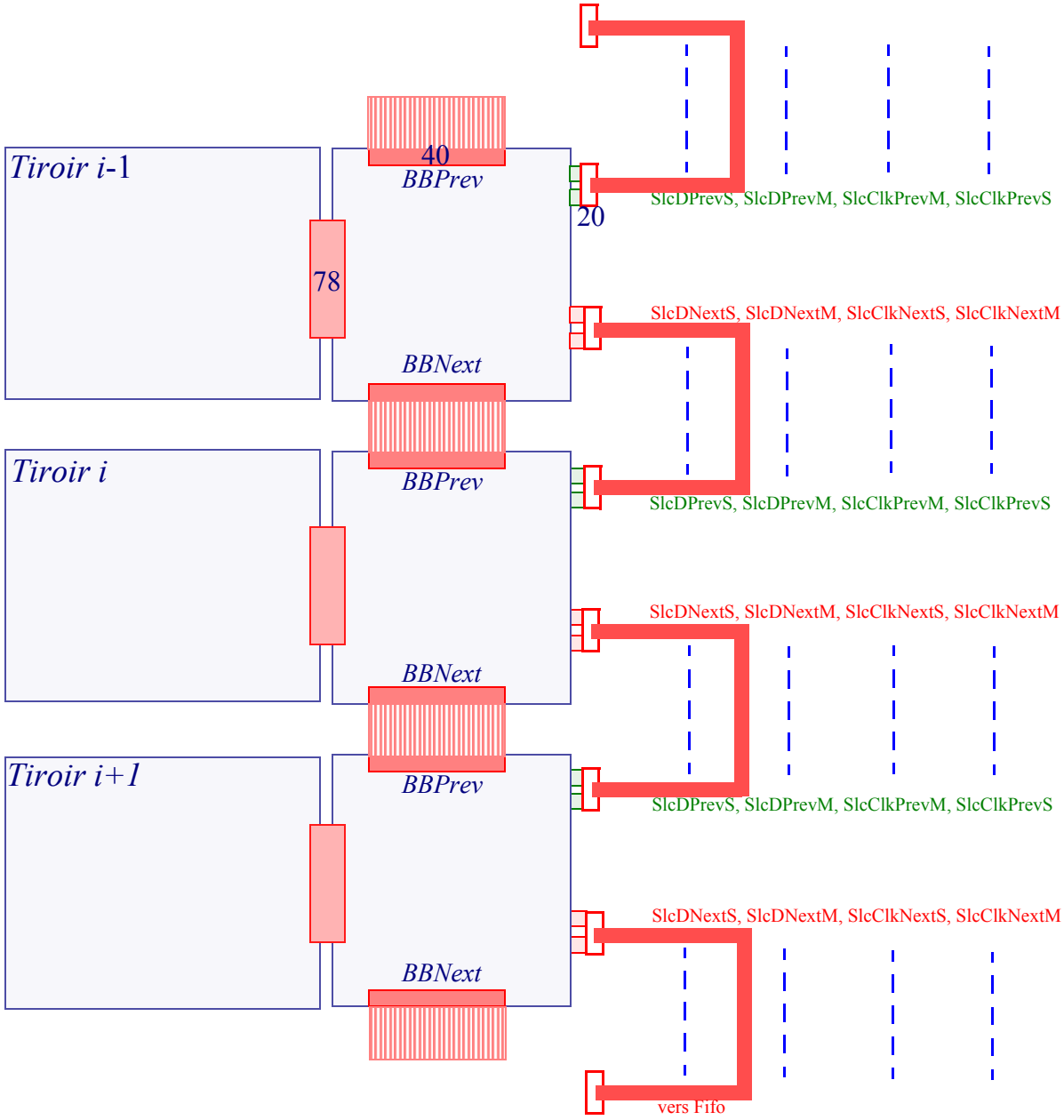


Figure 5.1 : Chaînage des différents bus

LES MESSAGES

1. Principe

L'électronique de la caméra pour l'expérience *HessII* est modulaire. Un module de base est constitué d'un *tiroir* accueillant 2 *cartes.* Chaque carte permet de traiter les informations en provenance de 8 photomultiplicateurs. La grande gamme dynamique de chaque PM nécessite 2 voies d'acquisition différentes. Une *carte* correspond à 16 voies d'acquisition. Le *Slow Control* des 2 *cartes* réside sur la carte mère du *tiroir*.

Le principe de l'interface de la caméra consiste à utiliser un système à 2 couches, l'une physique qui définit la " circuiterie " nécessaire pour les échanges, l'autre logique qui définit la structure des données sur la couche physique.

Ce document a pour but de spécifier un système d'interface qui exploite au mieux les données issues des *cartes analogiques* et du *Slow control*. Les différents concepteurs pourront s'appuyer sur une trame commune, cohérente et performante.

2. La couche logique

La couche logique est basée sur un système de messages pour converser entre un *Maître* et l'ensemble des *tiroirs* d'électronique. Une structure à base de messages présente de nombreux avantages, le principal étant que les données sont parfaitement structurées dés le plus bas niveau. La lisibilité et la manipulation des événements bruts sont donc facilités.

Dans ce concept, un message est un ensemble de mots de 16 bits possédant la structure suivante :

entête
type du message
Bloc de données *La taille du bloc dépend du type du message*
enqueue

Tableau 2.1 : Structure de base d'un message

• *L'entête* est un mot spécial qui permet de définir le début du message.

• Le *type* du message permet d'interpréter le bloc de données.

• *L'enqueue* est un mot spécial qui permet de définir la fin du message (l'enqueue permet essentiellement de vérifier la validité de la structure du message, la taille du message est a priori connue et ne dépend que du type du message).

Tous les échanges, aussi bien en lecture qu'en écriture sont basés sur la structure du message de base du tableau 2.1.

Par la suite, nous appellerons *Maître* le système processeur qui lit la caméra. De même, la *carte* ou le *tiroir* sont des ensembles capables de fournir ou de recevoir des messages du *Maître*.

Quand le *Maître* lit les données, il est en relation avec un élément unique (*carte* ou *tiroir*). Quand le *Maître* écrit des données, il peut le faire dans un ou plusieurs *tiroirs* à la fois (*broadcast*).

3. Les différents types de message

Il est nécessaire de pouvoir envoyer (ou recevoir) 2 catégories de messages :

• Les messages concernant les données d'acquisition. Dans ce cas, la *carte* est impliquée dans l'échange.

• Les messages concernant le " *Slow Control* ". Dans ce cas, le *tiroir* est impliqué dans l'échange.

Chaque catégorie est subdivisée en sous-catégories.

3.1. Les messages d'acquisition

La structure envisagée ne nécessite pas d'écrire dans les *cartes*, de ce fait, les messages d'acquisition sont exclusivement transmis d'une *carte* vers le *Maître*.

3 types de messages d'acquisition sont possibles[1], *DAQCharge*, *DAQSamples* et *DaqRdy* :

• *DAQCharge*, le bloc de données contient l'identificateur de la *carte* puis la charge équivalente (complément à 2) pour chacune des 16 voies de la *carte* (tableau 3.1). Un compteur d'événement (16 bits) permet d'évaluer la cohérence de l'ensemble des blocs lus. Ce compteur est remis à zéro lors du message *CntrlDaq* et incrémenté à chaque *L2A*. Le message peut contenir (selon la programmation de *CntrlDaq*) des informations temporelles (canal du pic du signal, largeur au dessus d'un seuil).

• *DAQSamples*, le bloc de données contient les $N_L \times 16$ échantillons des 16 voies de la *carte* (complément à 2). Une mémoire *SAM* est composée de 2 canaux *Ch1* et *Ch2*. Une carte analogique est équipée de 8 *SAM*. Le bloc est constitué pour chaque *SAM* (1 à 8) des N_L données des canaux *Ch1* et *Ch2*.

• *DAQCal*, Ce bloc de données permet de lire la valeur des DACs utilisée pour la calibration. Le bloc contient 256x16 mots (16 voies par carte et 16 lignes par voie).

• *DaqRdy*, ce message est automatiquement transmis par les cartes lorsqu'elles sont configurées par les messages de *Slow control CNTRLDaq*, *CNTRLSamDac*, *CNTRL-SamNd* (si cette option est validée). Le bloc de données ne contient que l'identificateur de la carte. Ce message permet de reconnaître les cartes présentes et/ou de servir d'accusé de réception à la commande initiale.

1. Par la suite, on fait référence à une structure de message particulière en nommant simplement son type, par exemple : *DAQCharge*

entête
DAQCharge
Identificateur carte
Compteur Evt
Voie 1 Voie 2 Voie 3 .... Voie 16 [T01/T02/T03/T04 T05/T06/T07/T08 T09/T010/T011/T012 T013/T014/T015/T016] [TOT1][TOT2][TOT3][TOT4] .... [TOT13][TOT14][TOT15][TOT16]
enqueue

Tableau 3.1 : Message DAQCharge

entête
DAQSamples
Identificateur carte
Échantillon 1 Échantillon 2 Échantillon 3 .... Échantillon N
enqueue

Tableau 3.2 : Message DAQSamples

entête
DAQCal
Identificateur carte
DAC1 voie 0 SAM0 ligne 0 DAC2 voie 0 SAM0 ligne 0 .... DAC1 voie 0 SAM0 ligne 15 DAC2 voie 0 SAM0 ligne 15 .... DAC1 voie 0 SAM7 ligne 15 DAC2 voie 0 SAM7 ligne 15 .... DAC1 voie 0 SAM7 ligne 15 .... DAC2 voie 15 SAM7 ligne 15
enqueue

Tableau 3.3 : Message DAQCal

entête
DAQRdy
Identificateur carte
enqueue

Tableau 3.4 : Message DAQRdy

Le principe de génération de ce message est le suivant, pour chaque message de *slow control* destiné à l'acquisition (*CNTRLDaq*, *CNTRLSamDac, CNTRLSamNd*) il existe 2 adresses. La première adresse ne demande pas la génération du message *DAQRdy* alors que la seconde implique sa génération.

3.2. Les messages de Slow Control

le *Slow Control* d'un *tiroir* est géré par la carte mère de ce *tiroir*. Les 2 *cartes* du *tiroir* ne sont pas concernées directement par ce type d'échange. Les *cartes* reçoivent, d'une manière interne, des informations programmées par l'intermédiaire du *Slow Control*.

Les informations gérées, par le *Slow Control*, au niveau de chaque *tiroir* sont très diversifiées (tableau 3.5).

On peut noter dans le tableau 3.5, les rubriques suivantes :

• Contrôle (écriture[1]) de la haute tension de chaque *PM*.
• Contrôle (écriture) du seuil Trigger.
• Contrôle (écriture) de la validation des voies du Trigger.
• Contrôle (lecture[2]) des échelles (mesure du fond de ciel).
• Contrôle (lecture) de températures et tensions.
• Configuration (écriture) des paramètres d'acquisition.

Parmi les informations du tableau 3.5, il est important de définir 3 catégories :
• Les informations *Échelles*.
• Les données générales de *Monitoring*.
• Les donnée de *Configuration* de la *DAQ*.

Il est intéressant de définir, respectivement, 2 messages en lecture (*CNTRLCpt* et *CNTRLMon*) et 5 messages de configuration en écriture (*CNTRLSlc*, *CNTRLTrig*, *CNTRLDaq*, *CNTRLSamNd* et *CNTRLSamDac*).

Désignation	*Input*	*Output*	*Nb motsx16 bits/ tiroir*
HT Vset	DAC 12 bits		16
HT Vmon		ADC 16 bits	16
HT Imon		ADC 16 bits	16
HT Status		REG 16 bits	1
HT Cde On/off	REG 16 bits		1
TRIGGER Seuil	DAC 8 bits		1
TRIGGER Enable voies	REG 16 bits		1
ÉCHELLES		CPT 16 bits	16
TEMPÉRATURE + Divers		ADC 16 bits	8
Fenêtre échelles	REG 8 bits		1

Tableau 3.5 : Différentes données gérées par le Slow Control

La représente le synoptique de la carte *Slow Control*. On remarquera que les différentes données analogiques de *Slow Control* (56 données différentes) sont multiplexées et converties par un unique *ADC*.

1. écriture: du *maître* vers le *tiroir*
2. lecture: du *tiroir* vers le *maître*

3.2.1. Lecture du *Slow Control : CNTRLCpt*

Le message *CNTRLCpt* (tableau 3.6) contient les informations liées aux échelles mesurant le taux de comptage du fond de ciel de chaque *PM*. Il est nécessaire de lire 16 mots *Échelle* (1 mot *Échelle* par *PM* du *tiroir*) . Ce bloc de données contient 16 mots. Si le taux de comptage dépasse la capacité du compteur *Échelle*, la valeur *FFFFH* est transmise.

entête
CNTRLCpt
Identificateur tiroir
Échelle 1 Échelle 2 Échelle 3 .... Échelle 16
enqueue

Tableau 3.6 : Message CNTRLCpt

3.2.2. Lecture du *Slow Control : CNTRLMon*

Le message *CNTRLMon* contient les informations liées au monitoring général du *tiroir* (tableau 4.1). Ce bloc de données contient 1+16+16+8=41 mots.

HTStatus indique, pour chaque voie, si la haute tension est conforme à la consigne (1 si conforme, 0 sinon). *HTStatusi*=1 si *HTi=Consignei*.

Le bloc *HTVmon* (16 mots) indique la valeur de la *HT* affichée sur chaque PM du *tiroir*.

Le bloc *HTImon* (16 mots) indique le courant total fourni à chaque PM du *tiroir*.

Le bloc *Température+divers* (5 mots) indique la température à 3 endroits du *tiroir* et la relecture des seuils *trigger L1* et *L2*.

4. Les constantes de conversion

Il est nécessaire de connaître certaines constantes de conversion utilisées afin de pouvoir interpréter les données.

4.1. Hautes Tensions

Les Hautes tensions sont écrites par l'intermédiaire d'une double conversion, un DAC convertit une donnée binaire en tension analogique (carte *slow control*) puis un multiplicateur de tension (carte *ISEG*) permet d'atteindre la valeur souhaitée.

- Pas du *DAC* (12 bits) = 1 *mV.*
- Dynamique du *DAC* = 12 bits (de 0 à 4095 *mV*).
- Facteur de multiplication de la carte *ISEG* = 400.

Le facteur 0,82, dans la formule suivante, correspond à une atténuation de 18% (passage dans les multiplexeurs analogiques).

$$VHTw = [ValeurDAC] \cdot [1mV] \cdot 400 \qquad [mV]$$

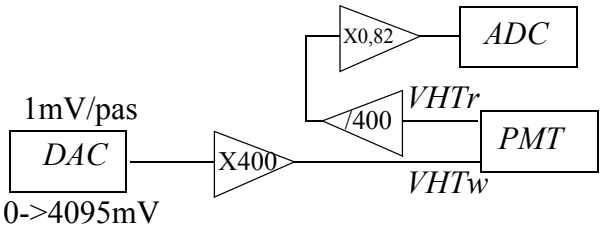


Figure 4.1 : Hautes Tensions

$$VHTr = \frac{[Valeur\ lue] \cdot 400}{0,82} \qquad [V]$$

4.2. Thermomètres

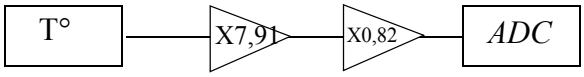


Figure 4.2 : Thermomètres

L'*ADC* est un *AD7671* (1 MSPS, 16 bits) avec une gamme dynamique : 0-5V

Le facteur 0,82 correspond à une atténuation de 18% (passage dans les multiplexeurs analogiques).

• Facteur de conversion des thermomètres : 8 *mV/°C*.

• Pas de l'*ADC* (16 bits) = $\frac{5}{2^{16}} = \frac{5}{65536} = 0,0762939mV$.

• Amplificateur G = 7,91.

$$T = \frac{[Valeur\ lue] \cdot 5/2^{16}}{8 \cdot 10^{-3} \cdot 10 \cdot 0,82} = [Valeur\ lue] \cdot 1,470312 \cdot 10^{-3} \qquad [°C]$$

4.3. Seuil Trigger

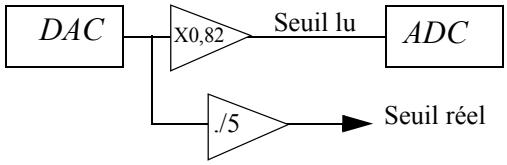


Figure 4.3 : Seuil Trigger

Le facteur 0,82 correspond à une atténuation de 18% (passage dans les multiplexeurs analogiques).

• $Seuil\ réel\ =\ \frac{Seuil\ lu}{5}$

• Pas de l'*ADC* (16 bits) = $\frac{5}{2^{16}}\ =\ \frac{5}{65536}\ =\ 0,0762939 mV$.

$$Seuil\ =\ \frac{[Valeur\ lue]\cdot 5/2^{16}}{5\cdot 0,82}\ =\ [Valeur\ lue]\cdot 1,860827\cdot 10^{-2}\qquad [mV]$$

avec $1\gamma e\ =\ 28 mV$

4.4. Haute tension HVmon

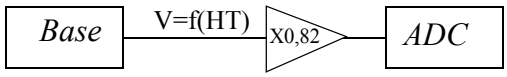


Figure 4.4 : Haute tension HVmon

Le facteur 0,82 correspond à une atténuation de 18% (passage dans les multiplexeurs analogiques).

• Pas de l'*ADC* (16 bits) = $\frac{5}{2^{16}}\ =\ \frac{5}{65536}\ =\ 0,0762939 mV$

• En sortie de la base $HVmon\ =\ \frac{HV\ réelle}{400}$

$$HVréelle\ =\ \frac{[Valeur\ lue]}{400\cdot 0,82}$$

4.5. Courant HImon

La base fournie une tension proportionnelle au courant *HImon.*

Le facteur 0,82 correspond à une atténuation de 18% (passage dans les multiplexeurs analogiques).

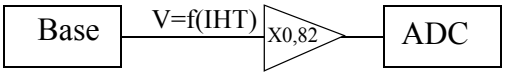


Figure 4.5 : Courant HImon

• Pas de l'*ADC* (16 bits) = $\frac{5}{2^{16}} = \frac{5}{65536} = 0,0762939 mV$.

• Facteur de conversion de la base : 25 $[mV \cdot \mu A^{-1}]$

$$HT\ Imon = \frac{[Valeur\ lue] \cdot 5 / 2^{16}}{25 \cdot 10^{-3} \cdot 0,82} = [Valeur\ lue] \cdot 3,72165 \cdot 10^{-3} \qquad [\mu A]$$

entête
CNTRLMon
Identificateur tiroir
HTStatus (1 si stable) HTVmon 1 .... HTVmon 16 HTImon 1 .... HTImon 16 Température 1 .... Température 3 Seuil Trigger L1 Seuil Trigger L2
enqueue

Tableau 4.1 : Message CNTRLMon

4.5.1. Lecture du *Slow Control : SLCRdy*

Le message *SlcRdy* est généré comme accusé de réception des commandes s*low control*. Le bloc de données ne contient que l'identificateur de la carte.

entête
SLCRdy
Identificateur tiroir
enqueue

Tableau 4.2 : Message SLCRdy

Le principe de génération de ce message est le suivant, pour chaque message de *slow control* (*SETHt*, *CNTRLTrig*, *CNTRLOut*, *CNTRLHt*, *CNTRLSlc*) il existe 2 adresses. La première adresse ne demande pas la génération du message *SLCRdy* alors que la seconde implique sa génération.

4.5.2. Écriture du *Slow Control : SETHt*

Le message *SETHt* (tableau 4.3) permet d'écrire dans un ou plusieurs *tiroirs* des données de type *Slow Control*. Ce bloc de données contient 16 mots.

entête
SETHt
Identificateur tiroir
HT 1 .... HT 16 HTBasse
enqueue

Tableau 4.3 : Message SETHt

SETHt (17 mots) permet de positionner une valeur précise de haute tension, individuellement, pour les 16 *PM* du *tiroir* et une valeur de haute tension basse *HTBasse* qui sera appliquée sur tous les *PMs*. Le passage de la tension nominale à la tension basse se fait par le message *CNTRLOut*.

4.5.3. Écriture du *Slow Control : CNTRLTrig*

Le message *CNTRLTrig* (tableau 4.4) permet de contrôler les voies du trigger dans un ou plusieurs tiroirs. Ce bloc de données contient 1 donnée.

Validation Trigger (1 mot) permet de valider ou d'inhiber une ou plusieurs voies de trigger (*bi*=0 validation voie *i*). En mode *broadcast*, le message *CNTRLTrig* permet de contrôler globalement l'ensemble du trigger.

entête
CNTRLTrig
Identificateur tiroir
Validation Trigger [b15].....[b0]
enqueue

Tableau 4.4 : Message CNTRLTrig

4.5.4. Écriture du *Slow Control : CNTRLHt*

Le message *CNTRLHt* (tableau 4.5) permet de mettre en service ou hors service les hautes tensions des *PM* d'un ou plusieurs *tiroirs*. Ce bloc contient un seul mot.

HTCde (1 mot de 16 bits) permet de valider ou d'inhiber une ou plusieurs voies de haute tension dans le *tiroir* (1 bit par HT).

entête
CNTRLHt
Identificateur tiroir
HTCde [b15].....[b0] (On=1, Off=0)
enqueue

Tableau 4.5 : Message CNTRLHt

4.5.5. Écriture du *Slow Control : CNTRLSlc*

Le message *CNTRLSlc* (tableau 4.6) permet de configurer, dans un ou plusieurs *tiroirs,* des informations liées à l'acquisition. Ce bloc de données contient 5 mots de programmation.

1- La taille de la fenêtre de comptage des échelles (8 bits),

2- *UpdateRate, AutoOff, Scaler/Slc Event, EnScaler, EnSlc* (1 mot) :

- *EnSlc* (bit0) actif si '1'
- *EnScaler* (bit1) actif si '1'
- *Scaler/Slc Event* (bit2) '1'/'0'
- *AutoOff* (bit3) actif si '1'
- *UpdateRate* (bit5-4)

le type de blocs *Slow control* à lire (*Scaler Event* ou *slow control*)

Ces bits définissent le type d'évènement *slow control* à acquérir : échelles ou monitoring général.

La taille de la fenêtre échelle *SW* (8 bits) définit un temps de la manière suivante :

T=[SW+1]x61,44µs

AutoOff est un bit qui affecte la partie *slow control*. Ce mode peut éviter une mono-

polisation du bus *Slow control* par un tiroir. Selon la périodicité des lectures, il est tout à fait possible de lire toujours la première carte *slow control*. En effet, le chaînage des jetons impose une priorité implicite entre les cartes.

Lorsque *AutoOff* est actif (si '1'), la partie *slow control* de la carte se rend automatiquement inactive après une lecture. Quand les cartes désirées ont été lues, l'envoi d'un nouveau message *CNTRLSlc* autorise à nouveau la lecture. Il est possible d'écrire *CNTRLSlc* en mode *broadcast* dans toutes les cartes.

Scaler Event/Slc Event permet de choisir entre une lecture d'un bloc *Scaler* (si '1') ou *Slow control* (si '0'). Deux séquenceurs indépendants scrutent les échelles et les données de *slow control* et remplissent des blocs de mémoire interne au *fpga*. Le message transmis dépendra uniquement du bit *[Scaler Event/Slc Event]*. Il est toutefois possible d'arrêter ces séquenceurs grâce aux bits *[EnScaler]* et *[EnSlc]* (Enable='1', Disable='0').

Le temps entre 2 séquences (*Slc*/*Echelles*) peut être défini par les 2 bits *UpdateRate* (pour un *ADC* convertissant en 1 µs) :

00 Vitesse maximale (42µs) entre 2 séquences.
01 Vitesse élevée, attente d'environ 0.15 s entre 2 séquences.
10 Vitesse moyenne, attente d'environ 1.5 s entre 2 séquences.
11 Vitesse lente, attente d'environ 15 s entre 2 séquences.

Si *Slc Event* n'est pas demandé (*[Scaler Event/Slc Event]=1*) le scanner *Slc* continu de fonctionner mais aucune requête ne sera générée.

3- *[Trigger Seuil]* (8 bits) permet de positionner le seuil du trigger de niveau 1, identique, pour toutes les voies du *tiroir*.

4- *[VMC]* (12 bits) permet d'ajouter (ou retrancher), à chaque échantillon de charge, une valeur fixe. Ceci permet de compenser un niveau continu constant sur chaque échantillon.

5- *[TOT]* (12 bits) est le seuil au dessus duquel il faut déterminer la largeur du signal.

entête
CNTRLSlc
Identificateur tiroir
Fenêtre échelles [UpdateRate][AutoOff][Scaler/SlcEvent][EnScaler][EnSlc] [Trigger Seuil L1] [VMC] [TOT]
enqueue

Tableau 4.6 : Message CNTRLSlc

4.5.6. Écriture du *Slow Control : CNTRLPreL2*

[Trigger delay L2] (6 bits) définit le délai de compensation nécessaire pour remettre

en temps les données *L2* avec le signal de prise en compte (*L1A*).

[Test Reg] (8 bits) définit un "pattern" fixe qui sera envoyé, en série, vers le *L2*, si le bit *UseTest*=1.

Mode=0 définit le mode mono-trame, le "pattern" de 2X8 bits des PMs touchés est sérialisé.

Mode=1 définit le mode multi-trames, dans ce cas, les 5 trames consécutives centrées sur [Trigger Delay L2] sont envoyées.

De façon à limiter le nombre de connexions vers le *L2*, il est possible de chaîner 2 tiroirs (4 cartes analogiques) et d'envoyer les trames de 4 cartes sur la même paire *LVDS*.

Pour cela il est nécessaire que chacune des 4 cartes analogiques soit identifiée :

carte ana 0 tiroir 1 ID=00
carte ana 1 tiroir 1 ID=01
carte ana 0 tiroir 2 ID=10
carte ana 1 tiroir 2 ID=11

Le paramètre *IDBase* définit si le bloc *CntrlPreL2* correspond au tiroir 1 (*IDBase*=0) ou au tiroir 2 (*IDBase*=1).

Le *fpga* chargera *IDBase*&0 dans la carte analogique 0 et *IDBase*&1 dans la carte analogique 1.

Remarque : Le message *CntrlPreL2* est spécifique à chaque tiroir, afin de pouvoir uttiliser le mode *broadcast*, Le bit *Loc*=1 définit d'une part que le mot [*Trigger Delay L2*] ne sera pas chargé et d'autre part, que l'identificateur du tiroir (*IDBase*) sera égal à NT[1] (switches de la carte arrière).

Le bit *Rst* permet d'imposer un *reset* permanent à l'*asic Pré-L2*. Dans ce cas, le *Pré-L2* devient inopérant.

Un reset est envoyé systématiquement à l'*asic Pré-L2* quand le bloc est reçu avec le bit *Loc*=0.

[Trigger Seuil L2] est un mot de 8 bits qui sera chargé dans le *Dac* seuil *pré-L2*.

entête
CNTRLPreL2
Identificateur tiroir
[Rst][Loc][IDBase][Trigger Delay L2] [Test Reg][Mode][UseTest] [Trigger Seuil L2]
enqueue

Tableau 4.7 : Message CNTRLpreL2

4.5.7. Écriture du *Slow Control : CNTRLDaq*

Le message *CNTRLDaq* permet de contrôler certains paramètres de l'acquisition.

entête
CNTRLDaq
Identificateur tiroir
[RstSAM][L2On/Off][RC][FAC][LdId][Nf][Charge/Sample][T0 On/Off][TOT On/Off]
enqueue

Tableau 4.8 : Message CNTRLDaq

RstSAM permet d'envoyer un reset (30 ns) aux circuits *SAM*. Ce bit doit être activé que lors de l'initialisation car celui-ci détruit certains registres.

FAC (Force Auto Cal) permet d'entrer dans le mode d'auto-calibrage des *DACs* des mémoires analogiques (Un *DAC* par ligne). Les *DACs* permettent de compenser les décalages continus (bruit de structure fixe) des amplificateurs de lecture des lignes (16) des mémoires analogiques. Cette calibration automatique est débutée lorsque *FAC=1*. D'autre part, comme ce mode implique la lecture interne des données, il est nécessaire que le " *Trigger soft* " soit validé et que la fréquence d'arrivée des triggers soit d'environ 10KHz (de cette fréquence dépend la durée de l'auto-calibrage). Il est possible de fonctionner avec ou sans L2 (voir l'utilisation du bit *[L2On/Off]*).

Un bloc de données (*DAQCAL*) contenant les données de calibrage est envoyé vers le processeur lorsque le bit *RC=1*.

Selon l'adresse *CNTRLDaq* positionnée (demande de *DaqRdy* ou non) et de l'état des bits *[RC][FAC]* un ou plusieurs messages seront générés par les cartes analogiques.

Le Tableau 4.9 résume les différentes possibilités.

DaqRdy demandé	***[RC]***	***[FAC]***	***Action***
non	*0*	*0*	aucune
non	*0*	*1*	calibration des *DACs*
non	*1*	*0*	*DaqCal* émis immédiatement
non	*1*	*1*	*DaqCal* émis après calibration des *DACs*
oui	*0*	*0*	*DaqRdy* émis immédiatement
oui	*0*	*1*	*DaqRdy* émis immédiatement, calibration des *DACs*
oui	*1*	*0*	*DaqCal* émis immédiatement (pas de *DaqRdy*)
oui	*1*	*1*	*DaqCal* émis après calibration des *DACs* (pas de *DaqRdy*)

Tableau 4.9 : Les messages liés au calibrage des DACs

Remarques :

- Ce message doit être envoyé, au moins une fois, avant toute action sur la carte analogique (CNTRLDaq est utilisé pour initialiser l'ensemble des variables).

- Le fait de lancer un CNTRLDaq avec les bits [RC] et/ou [FAC] lance une action de calibration. Après cette action le tiroir est en attente d'un nouveau CNTRLDaq.

Ceci permet d'avoir une fifo data qui ne contient que les données de calibration ou des messages DaqRdy.

- Après une calibration, les mémoires analogiques sont en adressage fixe. Il est nécessaire d'envoyer, à nouveau,le message CNTRLSAmReg (broadcast possible).

LdId permet de charger l'identificateur d'adresse dans les 2 cartes analogiques du tiroir (*LdId*=1). L'adresse du tiroir est définie par des mini-interrupteurs (8 bits) situés sur la carte arrière du tiroir. Il est nécessaire de demander, au moins une fois, au tiroir de charger l'identificateur *Ident* dans les 2 cartes analogiques. Quand un message *CNTRLDaq* est reçu, par la carte *slow control*, avec *LdId*=1 alors :

Ident[7:1]&0 est chargé dans la carte analogique 0 et

Ident[7:1]&1 est chargé dans la carte analogique 1.

Afin de limiter le nombre d'échantillons à lire des mémoires analogiques, il est possible de définir une fenêtre de lecture (à partir du pointeur de lecture). N_f (8 bits) correspond au nombre d'échantillons à lire dans la fenêtre.

Charge/Sample définit le mode d'acquisition de la carte (*Charge='1'* ou *Samples='0'*).

Le bloc de données (mode *Charge*) peut inclure pour chaque voie, le numéro de la cellule où se trouvait le maximum du signal (la valeur zéro correspond au début de la fenêtre).

[T0 On/Off] (1 bit) spécifie si l'on désire lire, pour chaque voie, la valeur (numéro de l'échantillon dans la fenêtre *[Nf]*) qui correspond au maximum du signal.

La longueur maximale du mot *T0* correspond à la longueur de la fenêtre. Par exemple, pour un échantillonnage à 1 ns et une fenêtre de 16, *T0 max*=4 bits. Dans ce cas, 4 mots sont suffisants pour entrer les 16 valeurs (4 valeurs par mot). Pour une fenêtre supérieure à 16, il faut 8 mots (2 valeurs par mot).

[TOT On/Off] (1 bit) spécifie si l'on désire lire, pour chaque voie, le facteur de forme du signal. *TOT* (*Time Over Threshold*) correspond au nombre de canaux de la fenêtre au dessus d'un seuil. Le seuil *TOT* est chargé par le message *CNTRLSlc*.

La longueur maximale du mot *TOT* correspond à la longueur de la fenêtre. Par exemple, pour un échantillonnage à 1 ns et une fenêtre de 16, *TOT max*=4 bits. Dans ce cas, 4 mots sont suffisants pour entrer les 16 valeurs (4 valeurs par mot). Pour une fenêtre supérieure à 16, il faut 8 mots (2 valeurs par mot). La figure 4.6 montre un exemple de *TOT*. Le signal passe au dessus du seuil à l'échantillon 4 et repasse en dessous du seuil à l'échantillon 10. *TOT*=10-4=6.

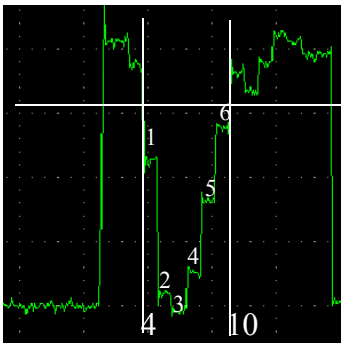


Figure 4.6 : Exemple de TOT

D'une manière plus précise, Les mesures du *T0* et du *TOT* sont représentées par la figure 4.7 (attention les canaux sont numérotés à partir de 1). Le seuil *TOT*=2.

Le signal passe le seuil au canal 4, atteint son maximum (*T0*) au canal 8 et repasse le seuil au canal 14. Les valeurs transmises seront *T0*=8 et *TOT*=10.

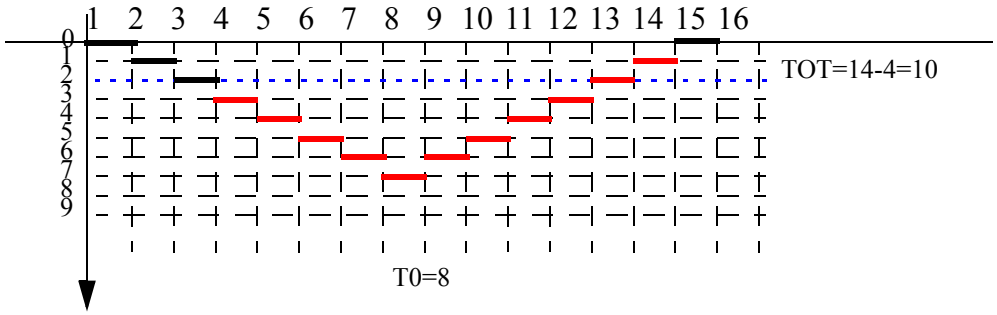


Figure 4.7 : T0 et TOT

Le message *CNTRLDaq* déclenche, automatiquement, le chargement dans les cartes analogiques de l'identificateur de carte. Des interrupteurs situés sur la carte arrière définissent l'identificateur du tiroir sur 8 bits : *Ident*[7:0]. *Ident*[7:1]&'0' est chargé dans la première carte et *Ident*[7:1]&'1' est chargé dans la seconde.

Les cartes analogiques sont synchronisées par les signaux de déclenchement (L1A) et (L2A/L2R). Dans un mode test, le signal *L2A* peut être généré, d'une manière interne, lorsqu'un *L1A* est reçu.

[L2On/Off]=0 permet de générer automatiquement le signal *L2A* à partir du signal *L1A*.

[L2On/Off]=1 lorsque la logique *L2* est utilisée.

Remarque importante : L'envoi du message CNTRLDaq a également pour conséquence de remettre à zéros les Fifos ainsi que de réinitialiser les mémoires analogiques.

4.5.8. Écriture du *Slow Control : CNTRLOut*

Le message *CNTRLOut* permet de basculer les hautes-tensions *PM* d'une valeur nominale *[HtNom/HtBasse]=1* (initialement chargée par le message *SETHt*) à une valeur basse prédéfinie également dans *SETHt* si *[HtNom/HtBasse]=0*.

entête
CNTRLOut
Identificateur tiroir
[HtNom/HtBasse]
enqueue

Tableau 4.10 : Message CNTRLOut

4.5.9. Écriture du *Slow Control : CNTRLSamDac*

Le message *CNTRLSamDac* permet de charger dans les mémoires analogiques des paramètres de compensation[1].

entête	
CNTRLSamDac	
Identificateur tiroir	
[NbDACs][FirstDAC][#carte][#mémoire]	
DAC1-L0	*DAC1+L0*
DAC2-L0	*DAC2+L0*
-	-
DAC1-L15	*DAC1+L15*
DAC2-L15	*DAC2+L15*
enqueue	

Tableau 4.11 : Message CNTRLSamDac

[# carte]=0/1 définit la carte qui sera chargée.

[# mémoire]=xxx définit le numéro de la mémoire (1/8) sur la carte.

Les 2 voies différentielles d'une mémoire (*DAC1+*, *DAC1-*, *DAC2+*, *DAC2-*) peuvent être compensées par 4 *DACs* de 7 bits. Il est possible de compenser les 16 lignes de la mémoire (notées L0 à L15).

[NbDACs] et [FirstDAC] sont utilisés à des fins de test mais doivent être toutefois correctement programmés.

[NbDACs] définit sur 4 bits (0 à 15) le nombre de registres 32 bits qui seront chargés dans SAM (4 DACs sont globalement chargés).

[FirstDAC] (4bits) définit le premier des DACs à charger (entre 0 et 15).

Pour un mode normal d'acquisition : [NbDACs]=15 et [FirstDAC]=0.

1. voir Spécification de la puce échantillonneuse SAM0, Table VI.5 (E. Delagnes-CEA)

4.5.10. Écriture du *Slow Control : CNTRLSamReg*

Le message *CNTRLSamReg* permet de charger des registres internes des mémoires *SAM*[1]. 3 registres (*Control/Debug Register* 1, *Control/Debug Register* 2 et *Test_FCR*) peuvent être chargés.

entête
CNTRLSamReg
Identificateur tiroir
Control/Debug Register 1
Control/Debug Register 2
Test_FCR
enqueue

Tableau 4.12 : Message CNTRLSamReg

Les tableaux suivants donnent quelques indications sur les bits des registres *Control/Debug register* 1, *Control/Debug register* 2. (le détail complet des régistres est défini dans la documentation *SAM*).

Si *sw_FCR*='1' alors *Test_FCR* est l'adresse 8 bits de la première cellule à lire.

Par exemple, on pourra charger *CDR1* avec : 4800H pour le mode d'acquisition standard, 4808H pour utiliser le registre *FCR*.

D'autre part, on pourra charger *CDR2* avec : 00B0H pour un fonctionnement classique de l'asic.

Bit	Nom	Description
3	*sw_FCR*	Accès en écriture selon *FCR*

Tableau 4.13 : Registre Control/Debug 1

Bit	*Nom*	*Description*
4	*unshort_ref*	Convertir le courant *DAC* en tension
5	*en_dac*	Valide la correction d'offset par les *DACs*
6	*dac_lsb_S1*	Défini *lsb DAC*
7	*dac_lsb_S2*	Défini *lsb DAC*

Tableau 4.14 : Registre Control/Debug 2

1. voir Spécification de la puce échantillonneuse SAM0, Table VI.1 (E. Delagnes-CEA)

dac_lsb_S1	*dac_lsb_S2*	*Description*
0	0	0.5 mV/*LSB*
0	1	0.25 mV/*LSB*
1	0	1 mV/*LSB*
1	1	*LSB* fixé par une résistance externe

Tableau 4.15 : Sélection de la valeur du LSB du DAC

4.5.11. Programmation du délai *Nd*

Nd (8 bits) correpond au délai entre l'arrêt de la mémoire analogique et le pointeur de lecture. Cette valeur permet de compenser le temps nécessaire pour élaborer le *trigger*.

entête
CNTRLSamNd
Identificateur tiroir
Nd
enqueue

Tableau 4.16 : Message CNTRLSamNd

4.6. Résumé des différents messages

Le tableau 4.17 résume les différents messages proposés.

Message	***Fonction***	***Description***
DAQCharge	Lecture des données en charge équivalente d'une *carte*	tableau 3.1
DAQSamples	Lecture des données brutes des mémoires	tableau 3.2
DAQCal	Lecture des *DACs* de calibration pour *SAM*	tableau 3.3
DAQRdy	Identification (Accusé de réception) de carte analogique	tableau 3.4
CNTRLCpt	Lecture des échelles fond de ciel d'un *tiroir*	tableau 3.6
CNTRLMon	Lecture du Slow Control général du *tiroir*	tableau 4.1
SLCRdy	Identification (Accusé de réception) de carte *Slc*	tableau 4.2
SETHt	Ecriture des valeurs des Hautes tensions	tableau 4.3
CNTRLTrig	Validation des voies de trigger d'un *tiroir*	tableau 4.4
CNTRLHt	Mise en service/Hors service des HT du *tiroir*	tableau 4.5
CNTRLSlc	Écriture du *Slow Control* général du *tiroir*	tableau 4.6
CNTRLPreL2	Paramètres pour le circuit *Pré-L2*	tableau 4.7
CNTRLDaq	Paramètres de l'acquisition, *Ident*	tableau 4.8
CNTRLOut	Bascule les HT entre HT nominales et HT Basse	tableau 4.10
CNTRLSamDac	Paramètres de compensation (DACs) *SAM*	tableau 4.11
CNTRLSamReg	Registres internes de *SAM*	tableau 4.12
CNTRLSamNd	Ecriture du délai *Nd*	tableau 4.16

Tableau 4.17 : Résumé des différents messages

5. Récapitulatif général des messages

Ce récapitulatif, permet d'avoir un accès rapide aux différents messages.
Numéro de tiroir : *NT*[7:1] permet de coder 1 parmi 128.
Numéro de carte : *NC*[7:0]=*NT*[7:1][0]
[0]=0 pour la carte analogique 1
[0]=1 pour la carte analogique 2

Messages émis par les cartes analogiques

DAQCharge
DAQSamples
DAQCal
DAQRdy

AAAAH	1
DAQCharge EEE0H	2
Identificateur carte 00xyH	3
Cpt Evt	4
Voie 1 Voie 2 Voie 3 .... Voie 16	
AAAAH	21

Tableau 5.1 : Récapitulatif, message DAQCharge

AAAAH	1
DAQCharge EEE0H	2
Identificateur carte 00xyH	3
Cpt Evt	
Voie 1 Voie 2 Voie 3 .... Voie 16	
[T01][T02][T03][T04] .... [T013][T014][T015][T016]	20 23
[TOT1][TOT2][TOT3][TOT4] .... [TOT13][TOT14][TOT15][TOT16]	24 28
AAAAH	29

Tableau 5.2 : Récapitulatif, message DAQCharge avec T0/TOT sur 4 mots

AAAAH	1
DAQCharge EEE0H	2
Identificateur carte 00xyH	3
Cpt Evt	
Voie 1 Voie 2 Voie 3 .... Voie 16	
[T01][T02] .... [T015][T016]	21 28
[TOT1][TOT2] .... [TOT15][TOT16]	29 36
AAAAH	37

Tableau 5.3 : Récapitulatif, message DAQCharge avec T0/TOT sur 8 mots

AAAAH		1
DAQSamples EEE3H		2
Identificateur carte 00xyH		3
Échantillon 1 ch1 SAM#1 Échantillon 2 ch1 SAM#1 ... Échantillon N_L ch1 SAM#1 Échantillon 1 ch2 SAM#1 Échantillon 2 ch2 SAM#1 ... Échantillon N_L ch2 SAM#1 Échantillon 1 ch1 SAM#2 Échantillon 2 ch1 SAM#2 .... Échantillon N_L ch1 SAM#2 ... Échantillon 1 ch2 SAM#8 Échantillon 2 ch2 SAM#8 ... Échantillon N_L ch2 SAM#8	[11:0] .... [11:0]	
AAAAH		$N_L \times 16 + 4$

Tableau 5.4 : Récapitulatif, message DAQSamples

AAAAH		1
DAQCal EEE5H		2
Identificateur carte 00xyH		3
DAC1 voie 0 ligne 0 DAC2 voie 0 ligne 0 .... DAC1 voie 0 ligne 15 DAC2 voie 0 ligne 15 .... DAC1 voie 15 ligne 0 .... DAC1 voie 15 ligne 15 DAC2 voie 15 ligne 15		
AAAAH		260

Tableau 5.5 : Message DAQCal

AAAAH	1
DAQRdy EEE8H	2
Identificateur carte 00xyH	3
AAAAH	4

Tableau 5.6 : Récapitulatif, message DAQRdy

Messages émis par les cartes *Slow control*

CNTRLCpt
CNTRLMon
SLCRdy

AAAAH	1
CNTRLCpt EEE9H	2
Identificateur tiroir 00xyH	3
Échelle 1 Échelle 2 Échelle 3 .... Échelle 16	
AAAAH	20

Tableau 5.7 : Récapitulatif, message CNTRLCpt

AAAAH	1
CNTRLMon EEEAH	2
Identificateur tiroir 00xyH	3
HTStatus HTVmon 1 .... HTVmon 16 HTImon 1 .... HTImon 16 Température 1 .... Température 3 Seuil Trigger L1 Seuil Trigger L2	
AAAAH	42

Tableau 5.8 : Récapitulatif, message CNTRLMon

AAAAH	1
SLCRdy EEECH	2
Identificateur carte 00xyH	3
AAAAH	4

Tableau 5.9 : Récapitulatif, message SLCRdy

Messages émis par le processeur

SETHt
CNTRLTrig
CNTRLHt
CNTRLSlc
CNTRLPreL2
CNTRLDaq
CNTRLOut
CNTRLSamDac
CNTRLSamReg
CNTRLSamNd

AAAAH	1
SETHt 7E00H FE00H (si *SLCRdy*)	2
Identificateur tiroir 00xyH	3
HT 1 .... HT 16 HT Basse (pour tous les PMs)	
AAAAH	21

Tableau 5.10 : Récapitulatif, message SETHt

AAAAH		1
CNTRLTrig 7E05H FE05H (si *SLCRdy*)		2
Identificateur tiroir 00xyH		3
Validation Trigger (On=0, Off=1)	*bits* [15][14]....[0]	4
AAAAH		5

Tableau 5.11 : Récapitulatif, message CNTRLTrig

AAAAH		1
CNTRLHt 7E0AH FE0AH (si *SLCRdy*)		2
Identificateur tiroir 00xyH		3
HTCde (On=1, Off=0)	*bits* [15:0]	4
AAAAH		5

Tableau 5.12 : Récapitulatif, message CNTRLHt

AAAAH		1
CNTRLSlc 7E0CH FE0CH (si *SLCRdy*)		2
Identificateur tiroir 00xyH		3
Fenêtre échelles	*bits* [7:0]	4
[UpdateRate][AutoOff][Scaler/Slc Event][EnScaler][EnSlc]	*bits* [5:4][3][2][1][0]	5
[Trigger Seuil L1]	*bits* [7:0]	6
VMC	*bits* [11:0]	7
Seuil TOT	*bits* [11:0]	8
AAAAH		9

Tableau 5.13 : Récapitulatif, message CNTRLSlc

AAAAH		1
CNTRLPreL2 7E0EH FE0EH (si *SLCRdy*)		2
Identificateur tiroir 00xyH		3
[Rst][Loc][IDBase][Trigger Delay L2]	*bits* [8][7][6][5:0]	4
[Test Reg][Mode][UseTest]	*bits* [9:2][1][0]	5
[Trigger Seuil L2]	*bits* [7:0]	6
AAAAH		7

Tableau 5.14 : Récapitulatif, message CNTRLPréL2

AAAAH		1
CNTRLDaq 7E30H FE30H (si *DAQRdy*)		2
Identificateur tiroir 00xyH		3
[RstSam][L2(0/1)][RC][FAC][LdId][Nf][Q/Smpl][T0(0/1][TOT(0/1)]	*bits* [15][14][13][12][11][10:3][2][1][0]	4
AAAAH		5

Tableau 5.15 : Récapitulatif, message CNTRLDaq

AAAAH		1
CNTRLOut 7E35H FE35H (si *SlcRdy*)		2
Identificateur tiroir 00xyH		3
[HTNominale/HTBasse]	*bits* [0]	4
AAAAH		5

Tableau 5.16 : Récapitulatif, message CNTRLOut

AAAAH			1
CNTRLSamDac 7E3AH FE3AH (si *DAQRdy*)			2
Identificateur tiroir 00xyH			3
[NbDACs][FirstDAC][Num carte][Num mémoire]		*bits* [15:12][11:8][3][2:0]	4
DAC1-L0	DAC1+L0	[15:0]	5
DAC2-L0	DAC2+L0	[15:0]	6
DAC1-L15	DAC1+L15	[15:0]	35
DAC2-L15	DAC2+L15	[15:0]	36
AAAAH			37

Tableau 5.17 : Récapitulatif, message CNTRLSamDac

AAAAH		1
CNTRLSamNd 7E3CH FE3CH (si *DaqRdy*)		2
Identificateur tiroir 00xyH		3
Nd	*bits* [7:0]	4
AAAAH		5

Tableau 5.18 : Récapitulatif, message CNTRLSamNd

AAAAH		1
CNTRLSamReg 7E3EH FE3EH (si *DaqRdy*)		2
Identificateur tiroir 00xyH		3
Control/Debug Register 1	*bits* [15:0]	4
Control/Debug Register 2	*bits* [15:0]	5
Test_FCR	*bits* [7:0]	6
AAAAH		75

Tableau 5.19 : Récapitulatif, message CNTRLSamReg

ARCHITECTURE DU TRIGGER POUR HESSII

1. Introduction

La caméra HESS II a été largement décrite, rappelons qu'elle est composée de 128 tiroirs. Chaque tiroir a en charge 16 photomultiplicateurs.

2. Description du trigger niveau 1

La figure 2.1 décrit les différents éléments intervenant dans le trigger de niveau 1. Afin d'élaborer le trigger de niveau 1, chacune des 256 cartes analogiques inclus un *pré-trigger L1*[1]. Le *pré-trigger L1* consiste en une somme analogique des 8 voies de la carte. Chaque voie est préalablement discriminée selon un seuil, programmable, défini en photo-électrons. Ces signaux (*pré-trigger L1*) sont transmis par câbles coaxiaux à une logique de *trigger* centrale.

9 cartes *Trigger* (T1 à T9) sont nécessaires pour traîter les 93 secteurs de la caméra (11 secteurs par carte *Trigger*, 1 secteur correspond à 8 *pré-trigger L1*). Le traitement consiste à discriminer la somme des signaux de chaque secteur. Le seuil de comparaison, dans chaque carte *Trigger* est équivalent à une multiplicité définie en pixels.

Les données issues des 9 cartes *Trigger* sont envoyées à une carte de gestion *Trigger Management* (*TM*) qui élabore le signal *L1A*. Le signal *L1A* est re-distribué vers les cartes analogiques par 4 cartes *FanOut* notées F1 à F4. Selon le type de connecteurs utilisé, il est possible de réduire le nombre de cartes *FanOut* à 2.

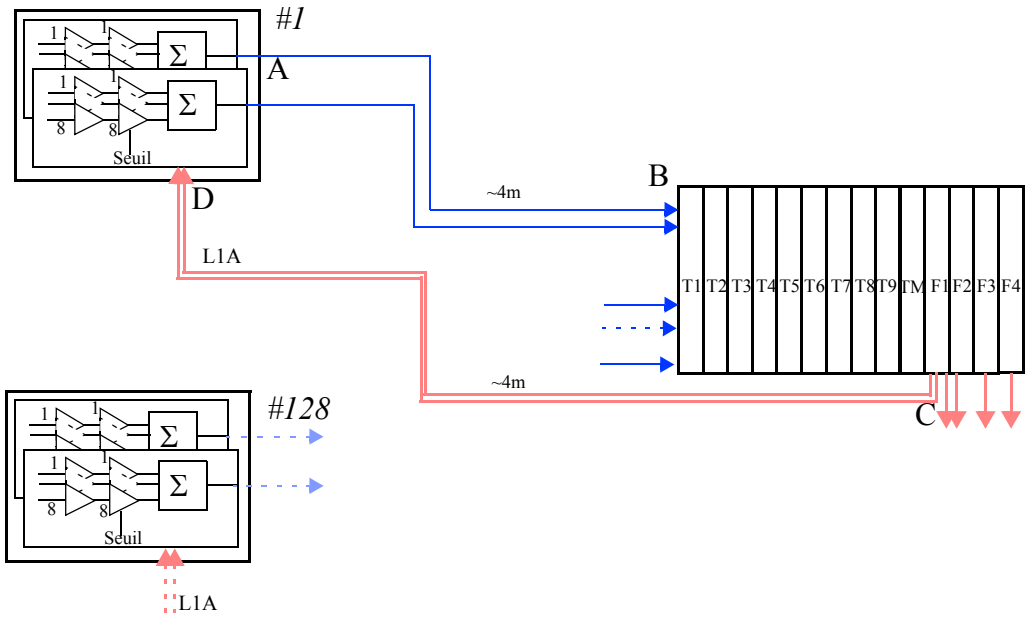


Figure 2.1 : Le trigger de niveau 1

Définissons :

• t_{AB} le temps pour transférer les signaux *pré-trigger L1* dans la carte *Trigger* cor-

1. Level 1

respondante,

• t_{BC} le temps pour élaborer le trigger de niveau 1,

• t_{CD} le temps pour tranférer les signaux *L1A* des cartes *FanOut* aux cartes analogiques.

Du point de vue des cartes analogiques, le temps nécessaire pour élaborer le *trigger L1A* est $t_{AD} = 4 \times (5ns/m) + T_{BC} + 4 \times (5ns/m) \approx 75ns$: avec $T_{BC} = 35ns$, temps de traitement de la logique centrale. Le temps t_{AD} correspond au temps de rétention pour les mémoires analogiques.

2.1. Description du trigger de niveau 2

La logique de *trigger* de niveau 2 est un filtre d'événements qui permet de limiter le flot de données acquis par les processeurs.

La logique de niveau travaille à partir de données *pré-trigger L2* élaborées dans chacune des cartes analogiques (figure 2.2).

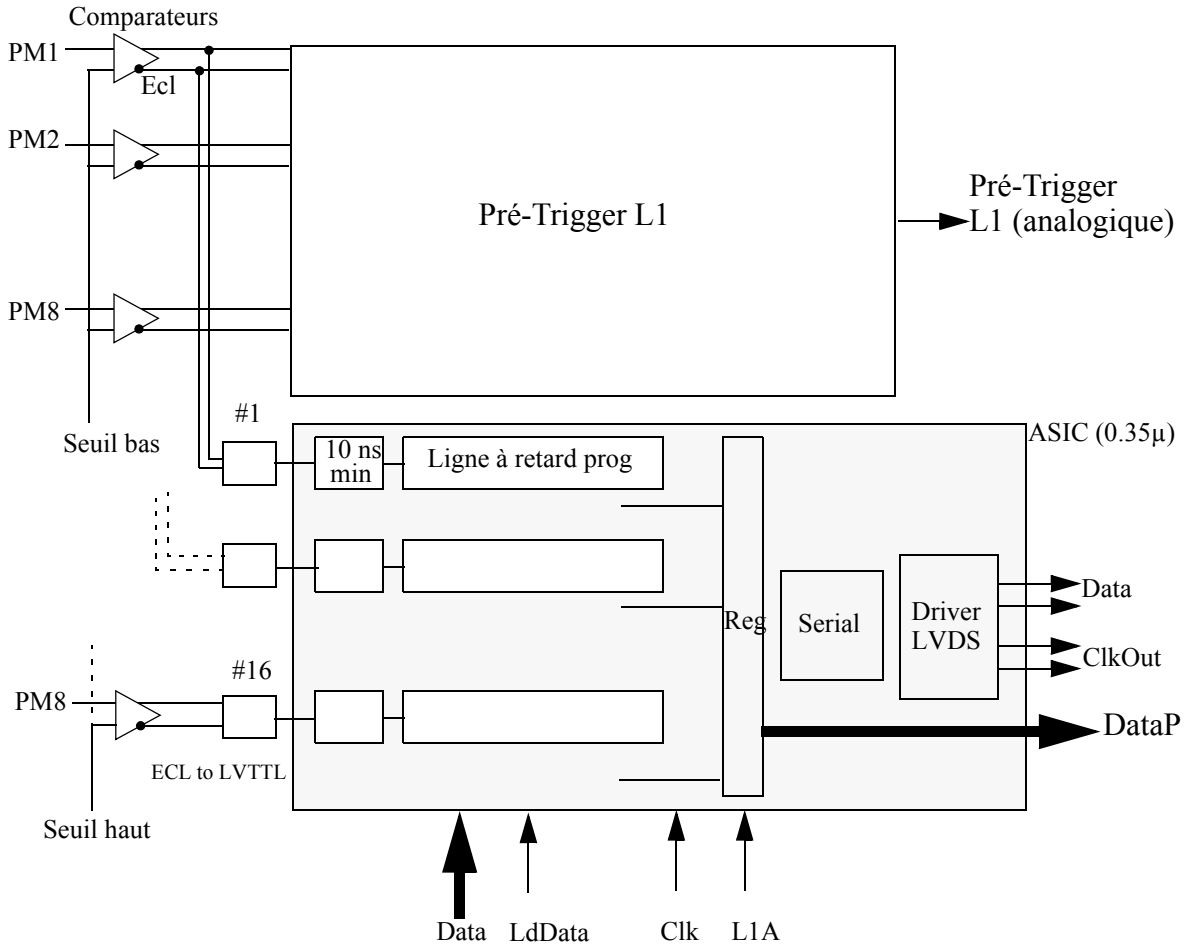


Figure 2.2 : Elaboration des données pré-trigger L2

Le pré-traitement *L2* dans les cartes analogiques est réalisé par un circuit spécifique (Asic, voir http://lpnp90.in2p3.fr:8080/HESS/workspaces/hess-ii/camera/pre-l2). Les 8 signaux analogi-

ques issus des PMs d'une carte analogique sont discriminés selon deux seuils exprimés en photoélectrons (1 *pe*~28 *mV*). Le seuil bas permet d'élaborer le pré-trigger *L1* et *L2*. Le pré-*L2* utilise les 16 informations en sortie des comparateurs seuil bas et seuil haut. Les seuils sont programmables pour tout le tiroir (2 cartes). Par exemple le seuil bas est calé entre 2 et 4 *pe*. Les signaux en sortie des comparateurs sont de largeur variable (entre 2 et 12 *ns* environ).

Pour le pré-*L2* les sorties *ECL* doivent être translatées pour être compatible avec l'ASIC Pré-*L2* (translateur ECL-LVTTL *MC100ETP25*). L'étage d'entrée de l'ASIC remet en forme les signaux pour que leur largeur soit au minimum de 10 *ns*.
Une ligne à retard programmable est utilisée pour resynchroniser le signal issu de la logique de niveau 1 avec le pattern correspondant. Le retard attendu est fixe et inférieur à 100 *ns*.
Le signal *L1* permet de mémoriser le pattern dans un registre puis de déclencher la sérialisation du registre (*Data* + *ClkOut*) dans le standard *LVDS*.

On peut considérer une ligne de 128 *ns* et l'on peut admettre un pas de 2 *ns*. Une horloge rapide *LVDS* (66 *MHz*) est fournie à l'ASIC. Cette horloge est nécessaire pour asservir correctement la ligne à retard (*DLL*) et pour la logique séquentielle du sérialiseur.

Il est possible de demander au circuit de sortir la trame de 8 bits (t_0) ou 5 trames (t_{-2}) (t_{-1}) (t_0) (t_{+1})(t_{+2}). Cette possibilié permet de bien ajuster le délai.
Un bus de 8 bits pris en compte par le signal *LdData*, permet de charger dans l'ASIC Pré-*L2*, le retard (6 bits) permettant de resynchroniser le *L1A* avec le pattern 16 bits qui lui correspond. Un bit de mode définit si la sortie est multi-trames (une ou cinq trames).
Le pattern correspondant au *L1A* est également disponible sous la forme d'un bus de 16 bits (parallèle).

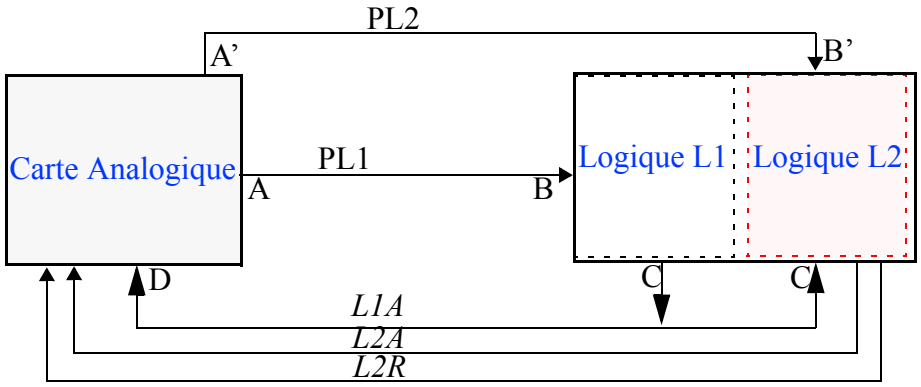


Figure 2.3 : Le trigger L2

La logique *L2* reçoit 512 paires (*pré-trigger L2*) soit 2 paires par carte analogique. La logique *L2* génère, pour chaque *trigger L1*, un signal *L2A* (*Accept*) ou un signal *L2R* (*Reject*).

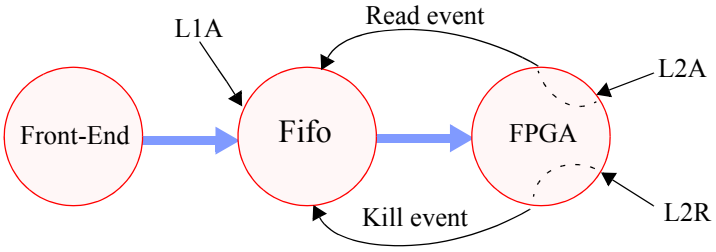


Figure 2.4 : Principe des signaux L2A et L2R

Quand un signal *L2A* est reçu, par le *Fpga* des cartes analogiques, l'événement en tête des *Fifos* est lu, formaté et envoyé vers le processeur. Quand un signal *L2R* est reçu, l'événement en tête des *Fifos* est rejeté.

La figure 2.5 montre le rôle des cartes *TM* (*Trigger Management*) et *Fan Out*, celles-ci n'étant pas représentées sur la figure 2.3.

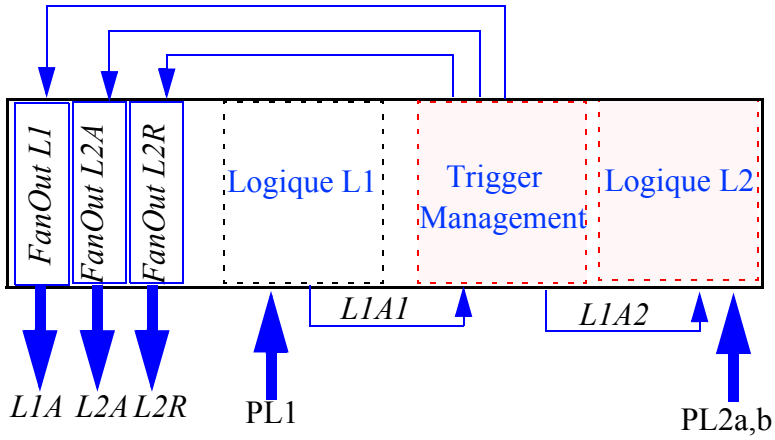


Figure 2.5

3. Les contraintes liées au *trigger*

Certaines contraintes ont été mises en évidence (Voir " § 6.1. page 18 ").

3.1. Contrainte liée au temps mort SAM

Les cartes analogiques ne peuvent pas accepter de *trigger L1A* si les mémoires analogiques sont en mode lecture.

Le temps mort de lecture, dans le cas d'une fenêtre N_f échantillons et d'une horloge de lecture de période *ReadClock*, est donné par la relation suivante :

$t_L = ceil\left[\frac{N_f}{16}\right] + N_f \times ReadClock$ avec *ceil* l'arrondi supérieur (par exemple si $N_f = 17$, $ceil\left[\frac{N_f}{16}\right] = 2$.

Si $N_f = 16$ et $ReadClock = 90ns$ alors $t_L = 1,53\mu s$.

Quand un *trigger L1A* a été généré, la logique centrale de *trigger* doit interdire tout

autre trigger pendant un temps t_L.

3.2. Contrainte liée au temps mort du fpga de lecture des Fifo

Pendant que le *fpga*, de la carte analogique, lit les *fifo* dans sa mémoire interne, il ne peut pas traîter un *L2R*.

Quand un *L2A* est généré, le *fpga* lit la *fifo* pendant $t_F = [N_f \times 16 + 15] \times 30ns$. soit $9,9\mu s$ pour 320 mots.

Quand un *trigger L2R* est généré dans ce temps mort, la logique centrale de *trigger* doit différer l'envoi de ce signal en dehors de l'intervalle t_f.

3.3. Transmission L2A/L2R

Les signaux *L2A* et *L2R* sont générés par la logique centrale du trigger (*Trigger Management*) et doivent être distribués à toutes les cartes analogiques. Ces 2 signaux sont exclusifs et ne peuvent pas être générés en même temps. De ce fait, il est possible d'économiser la moitié des cartes de distributions et des câbles en encodant ces 2 signaux. La figure 3.1 explicite le principe d'encodage du signal *L2AR*.

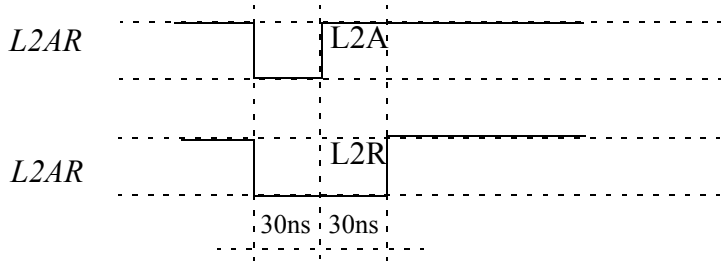


Figure 3.1 : Encodage L2A/L2R

CARTE TRIGGER PROTO

1. Introduction

La carte *Trigger Proto* est une version intermédiaire pour l'expérience *HESSII*.

Cette carte est interfacé avec le bus *cPCI* (le *CUSTOMBUS* de *HESSI* est abandonné.) et tient compte des 99 secteurs de la nouvelle la caméra *HESSII*.

Le *trigger* de niveau 1 est réalisé par 9 cartes de 11 secteurs. Chaque carte accepte en entrée 12 groupes de 4 *pré-trigger L1* (répartis en 3 connecteurs de 16 *pré-trigger L1*).

Cette carte utilise la version *HESSI* de l'interface *cPCI* (circuit *PQFP*), le *fpga Trigger* est quant à lui de la nouvelle génération (*EP1C12 Altera*). Cette carte sera utilisé sur le banc de test PM.

2. La carte Trigger proto

Le synoptique de la carte est représenté par la figure 2.1.

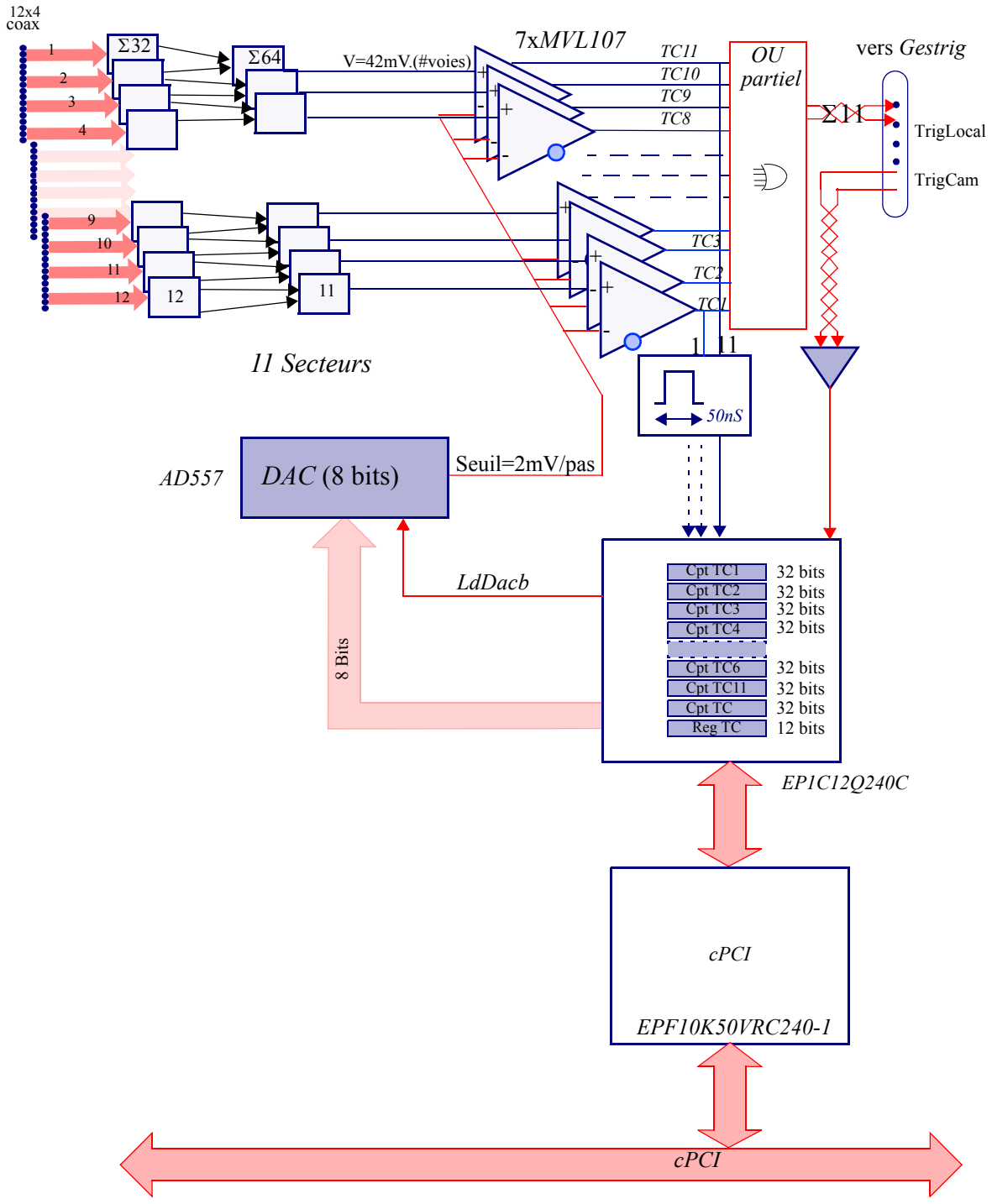


Figure 2.1 : Synoptique de la carte trigger

3. Programmation

La programmation de cette carte est relativement simple et s'effectue au travers l'interface *cPCI*. Le tableau 3.1 donne les commandes pour lire ou écrire les différents fonctions de la carte. Il faut noter que l'interface *cPCI* traîte au minimum des mots de 32 bits.

Adresse (BAR0/BAR1)	BAR	R/W	
5H	BAR0	W	Ecriture *DAC 8 bits*
8H	BAR1	W	RAZ cpt et registre secteurs touchés
0H	BAR0	R	Lecture *(11 bits)* secteurs touchés Reg[j]=secteur(j+1) avec $0 \leq j \leq 10$
$1 \leq i \leq 0BH$	BAR0	R	Lecture compteur secteur *i (32 bits)*
0CH	BAR0	R	Lecture compteur *Trigger* Caméra *(12 bits)*

Tableau 3.1 : Table de programmation

La conception du *fpga* autorise la lecture des registres en mode bloc.

4. Codage du mot *DEVICE_ID fpga cPCI*

Le *FPGA cPCI* possède un espace de configuration dont le contenu du mot *Device_ID* est utilisé par le système pour déterminer le driver logiciel à utiliser.

Pour la carte *Trigger Proto* : *Device_ID=X"2CCC"*.

CARTE TRIGGER

1. Introduction

Le trigger de niveau 1 de l'expérience HESS II est temporel et sectoriel. La partie temporelle située sur les cartes analogiques discrimine chacun des huit signaux de photo-multiplicateurs grâce à huit comparateurs sub-nanoseconde. Le seuil, des comparateurs, définit le nombre de photoélectrons au dessus duquel la voie participera au trigger L1. Pour réaliser la partie sectorielle du niveau 1 il faut fournir aux cartes *Trigger* situées dans le châssis *cPCI Trigger* l'information du nombre de voies touchées (la multiplicité est obtenue par sommation). Le signal de sortie de chaque comparateur ($\Delta V = 800mV$) est converti en courant afin de faciliter la somme des 8 courants de chaque carte analogique. L'impulsion de courant dont la largeur est au moins de 2 *ns* pourra être pris en compte. La somme (analogique) est envoyée, par 2 câbles coaxiaux, vers les cartes *Trigger* situées dans le châssis *cPCI Trigger*.

La caméra est décomposée en 99 secteurs (figure 1.1). Le *trigger* de niveau 1 est réalisé par 9 cartes de 11 secteurs. Chaque carte accepte en entrée 12 groupes de 4 *pré-trigger L1* (répartis en 3 connecteurs de 16 *pré-trigger L1*). Les secteurs se chevauchent verticalement (16 pixels) et horizontalement (32 pixels).

Un signal de déclenchement sera généré s'il y a suffisamment d'énergie dans un secteur, ce qui revient à demander qu'un certain nombre de pixels soient simultanément présents dans le secteur (multiplicité). Ce nombre de pixels au dessus du seuil est programmable dans chaque carte trigger (Le facteur de conversion est de 42mV par pixel). Le synoptique de la carte *Trigger* est représenté sur la figure 1.2. Les sorties des 11 comparateurs (un par secteur) basculent en permanence. Le ou logique des onze comparateurs (*L1i*) va contribuer à l'élaboration du signal *L1A* : $L1A = \sum_{i=1}^{11} L1Ai$. Le temps nécessaire à l'élaboration du signal *L1A* est de l'ordre de 20 à 30 *ns*. Le circuit *Asic Delay* permet de resynchroniser le pattern (11 bits) initial avec le *L1A*. Ce pattern est mémorisé dans une *Fifo* (appelée externe). Le premier mot de la *Fifo* sera soit relu, par le *fpga*, avec le signal *L2A* soit rejeté avec le signal *L2R*.

Afin de donner un peu plus de souplesse au processeur de lecture, un second niveau de *fifo* (appelé locale ou interne) est implémenté dans le *fpga*. Quand une donnée est sortie de la première *fifo* avec le signal *L2A*, elle est entrée dans la fifo interne (256 mots de 11 bits). Cette *fifo* interne est lue par le processeur.

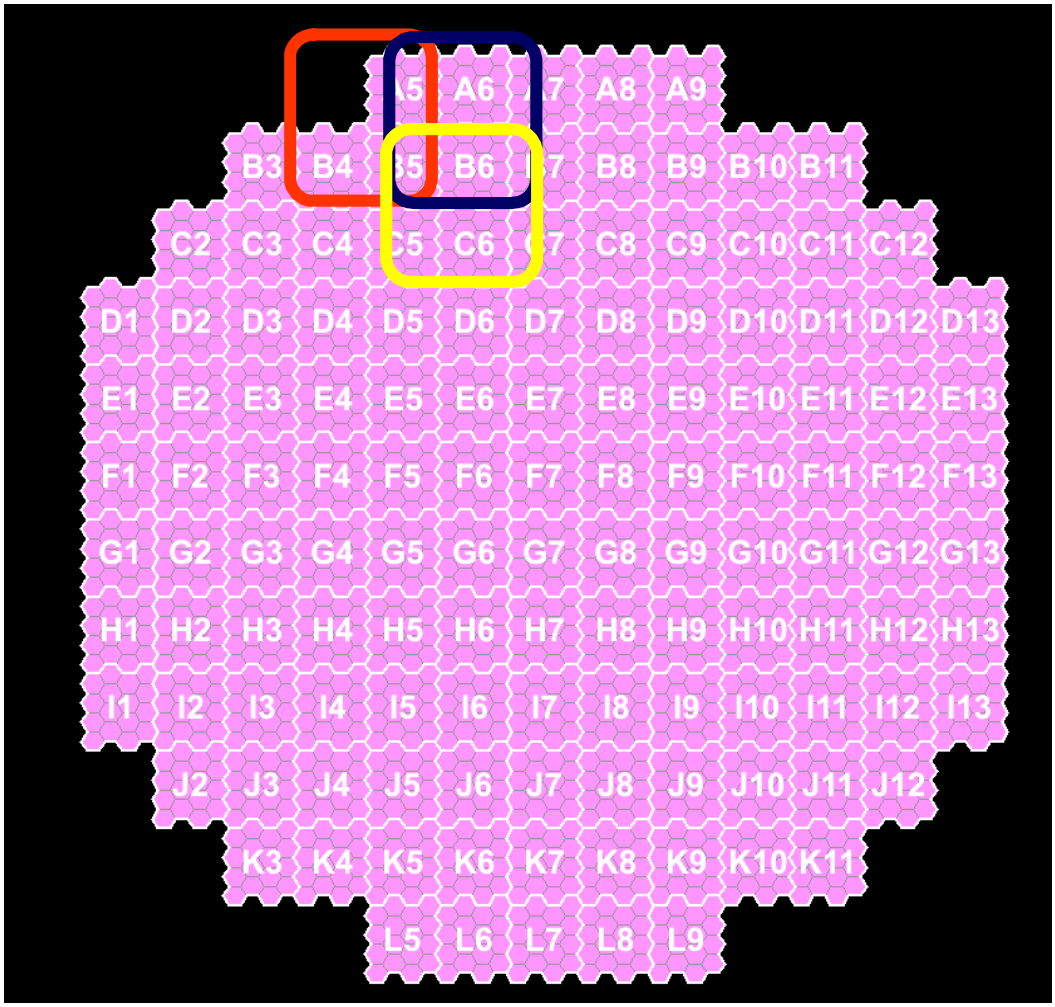


Figure 1.1 : Les 99 secteurs de la caméra HESSII

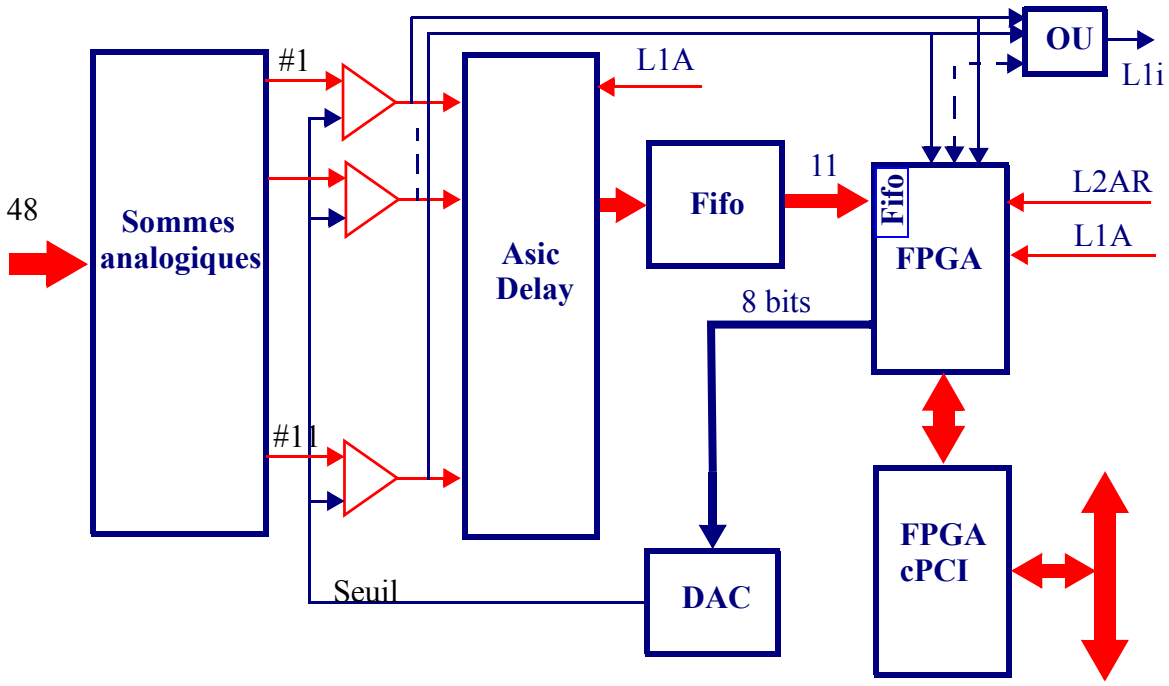


Figure 1.2 : Synoptique de la carte trigger

2. programmation

La carte est équipée de deux *FPGAs*. L'interface avec le bus *compactPCI* est effectuée par un *fpga Altera BGA EP1C4F324C6* de la série *Cyclone*. La fonctionnalité de ce circuit est complètement décrite dans une note[1]. L'interface avec les fonctionnalités de la carte *Trigger* est effectuée par un *fpga Altera* PQFP EP1C12Q240C6 de la série *Cyclone*.

2.1. La fenêtre de comptage

Les triggers de chaque secteur sont comptés, dans le *fpga*, pendant un temps programmable défini par le paramètre *Upper* (Compter pendant un temps prédéterminé permet de trouver facilement la fréquence de comptage). Le paramètre *Upper* est la partie supérieure d'un mot de 32 bits : *Win*[31:16]. Les quatre bits de poids faibles de *Win* sont tels que : *Win*[3:0]="F"H.

Winhexa="*Upper000F*" (Upper représente 4 digits en hexadécimal).

La largeur de la fenêtre est donnée par la relation suivante : $T_W = [(WinDec \times 2) - 1] \times 30ns$

Avec *WinDec* la représentation décimale de *Winhexa*.

Par exemple, la plus petite fenêtre est obtenue avec *Upper*=0. Comme WinDec=15 alors $T_W = 870ns$. De même, la plus grande fenêtre est obtenue pour *Upper*=FFFFH. *WinHexa*=FFFF000F, *WinDec*=4294901775 et $T_W = 257s$.

De façon à lire des valeurs cohérentes de compteurs, ceux-ci sont transférés dans des registres de lecture en fin de fenêtre et s'il n'y a pas de lecture en cours.

1. Interface compactPCI (cPCI)

2.2. Les secteurs touchés et le compteur d'événements

Pour chaque *L1A*, l'image des secteurs touchés de la caméra est resynchronisée (circuit *delay-PréL2*) et mémorisée dans une *fifo* externe.

En réception d'un *L2R*, le premier mot de la *fifo* est purgé.

En réception d'un *L2A* le premier mot de la *fifo* externe est lue, un compteur d'événements (16 bits), dans le *FPGA*, est incrémenté. Les 2 mots (compteurs d'événements et secteurs touchés) sont entrés dans une *fifo* interne (256 mots) au *FPGA* dans l'attente d'une lecture. Chaque mot *fifo* a la structure suivante :

FIFO[31:16]= [*Cpt Evt*]
FIFO[15:11]= [000000]
FIFO[10:0]= [*pattern*]

Afin de garder la cohérence de l'association, *Compteur d'événements*, *pattern* et *L1A*, il est nécessaire de lire systématiquement cette *fifo*.

Les compteurs d'événements, de toutes les cartes *Triggers*, sont remis à zéros périodiquement par la carte *TrgMngt* (signal *RstL2Cnt*).

2.3. Le circuit Delay-PréL2

L'*Asic Delay-PréL2* est un circuit qui a été développé pour répondre aux besoins du trigger de niveau 2 de l'expérience HESS II. Ce circuit peut, d'une manière plus générale, être utilisé comme un circuit de 16 lignes à retards programmables. En réception du signal *L1A*, l'image des signaux présents à *DelIn ns* avant est mémorisée et resortie par le circuit. Le tableau 2.1 donne les indications de programmation du circuit *Delay*. Les bits [9:8] de *Param pré-L2* sont considérés comme une adresse. La valeur *DelIn* (6 bits)=*Param pré-L2[5:0]* est programmable par l'utilisateur, dans ce cas les bits [9:8]="00" : *Param pré-L2[9:0]="00xxyyyyyy", yyyyyy* étant les 6 bits du délai (le pas du délai est de 2 *ns*).

Après la programmation, et pour un fonctionnement du mode normal, il est nécessaire de s'assurer que *Param pré-L2[9:0]="11xxxxxx00"*.

Bits : [9:8]	*Bits*	*Param pré-L2*	*Défaut*	
00	[5:0]	*DelIn*	*0*	
01	[7:0]	*TestReg*	*0*	*ne pas utiliser*
10	[1:0]	*ID*	*0*	*ne pas utiliser*
11	[0] [1]	*UseTest* *Mode*	*0* *0*	*doit être 0* *doit être 0*

Tableau 2.1 : Table de programmation du circuit Delay-PréL2

L'accès à la programmation se fait par l'intermédiaire du tableau 2.2.

Par exemple, l'écriture en *BAR1*, userad=03H du mot binaire 0000101010 permet de programmer le circuit délai pour un retard de 42x2 *ns*.

2.4. RAZ

La remise à zéro du *fpga* est réalisée, automatiquement, à l'initialisation par le processeur *cPCI*. Toutefois, il est possible de réinitialiser la plupart des circuits du *fpga* par

la *RAZ programmée*. Cette initialisation est indépendante de la RAZ de la *fifo* et de la RAZ de *l'Asic pré-L2* (delay). La RAZ programmée réinitialise les registres et compteurs des secteurs.

2.5. Mode Test

Ce mode spécial permet de tester la carte quand la logique de *trigger L2* et la carte *Trigger Management* ne sont pas disponibles (il est également possible d'émuler le L2 dans la carte *Trigger Management*). Quand ce mode est activé (*TestMode*) un *L2* est automatiquement généré pour chaque *L1* indépendemment du signal *L2AR*. Le signal *L2* est dans ce cas généré 500 ns après le signal *L1A*.

Adresse (BAR0/BAR1)	BAR	R/W	
01H	BAR1	W	RAZ *programmé*
02H	BAR1	W	RAZ *Asic pré-L2*
03H	BAR1	W	*Param pré-L2* 10 *bits*
04H	BAR1	W	RAZ de la *Fifo*
05H	BAR1	W	Ecriture *DAC 8 bits*
06H	BAR1	W	*Upper=Win*[31:16]
07H	BAR1	W	TestMode L1 donne L2 si Data(0)=1
00H	BAR0	R	*Compteurs d'événements [31:16]* *Pattern Sector*s [10:0] (lecture fifo interne)
$1 \leq i \leq 0BH$	BAR0	R	Lecture compteur secteur *i (16 bits)*
0CH	BAR0	R	Lecture compteur *Trigger* Caméra *(16 bits)*
0DH	BAR0	R	Lecture du word count de la *fifo* interne, *fifofull*, *fifoempty (10 bits)* [*FifoEmpty*][*FifoFull*][*WC*(7:0)]

Tableau 2.2 : Table de programmation

3. Codage du mot *DEVICE_ID fpga cPCI*

Le *FPGA cPCI* possède un espace de configuration dont le contenu du mot *Device_ID* est utilisé par le système pour déterminer le driver logiciel à utiliser.

Pour la carte *Trigger* : *Device_ID=X"2CCD"*.

CARTE TRIGGER MANAGEMENT

1. Introduction

La carte *Trigger Management* (*TrgMngt*) génère, à partir des informations en provenance des logiques de niveau 1, 2, de la carte *TimeStamp* et du *Local Module Trigger* (*LMT*), les signaux nécessaires pour le déclenchement de la caméra *HESS II* (figure 1.1). Le principe de la logique de niveau 1 est décrit dans l'introduction à la carte *Trigger* (Voir " § page 79 ").

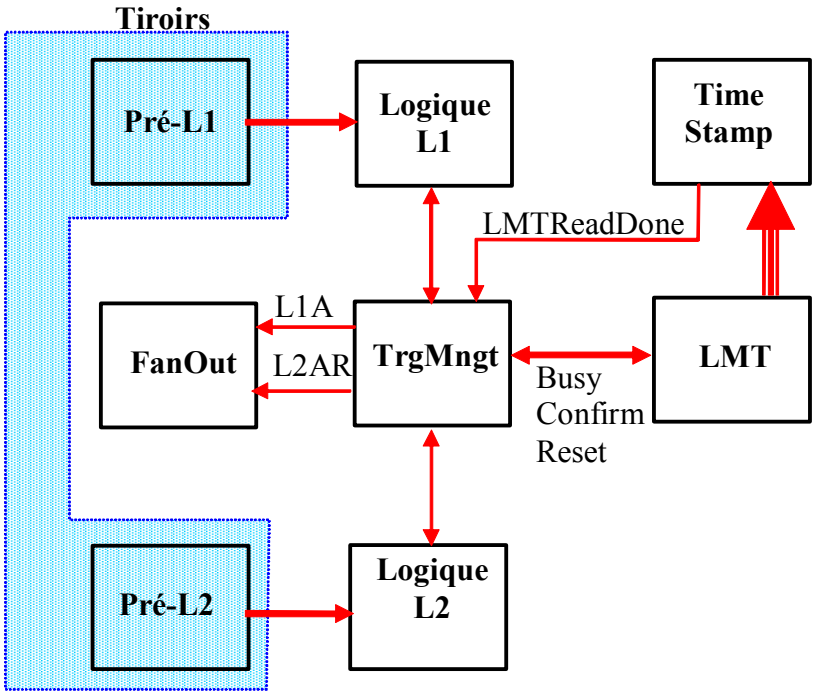


Figure 1.1 : Les différentes cartes en relation avec TrgMngt

Les logiques de trigger *L1* et *L2* travaillent à partir des données *Pré-L1* et *Pré-L2* en provenance des 256 cartes analogiques.

Le *Local Module Trigger* (*LMT*) permet d'interfacer la logique de trigger de la caméra avec le trigger central situé au sol. La liaison entre le *LMT* et le trigger central est réalisée par des fibres optiques. Le *LMT* fournit à la caméra les ordres d'acceptations du trigger ainsi que les données critiques (Numéro d'événement, Numéro de Bunch et Noeud ID) nécessaires pour transmettre l'événement.

La carte *TimeStamp* permet d'une part de synchroniser les différents châssis *cPCI* (*DAQ*, *Trigger*, *Sécurité*) et d'autre part de lire les données critiques du *Local Module Trigger* et de transférer ces données aux différents châssis.

La figure 1.2 donne le timing général des principaux signaux des différentes cartes.

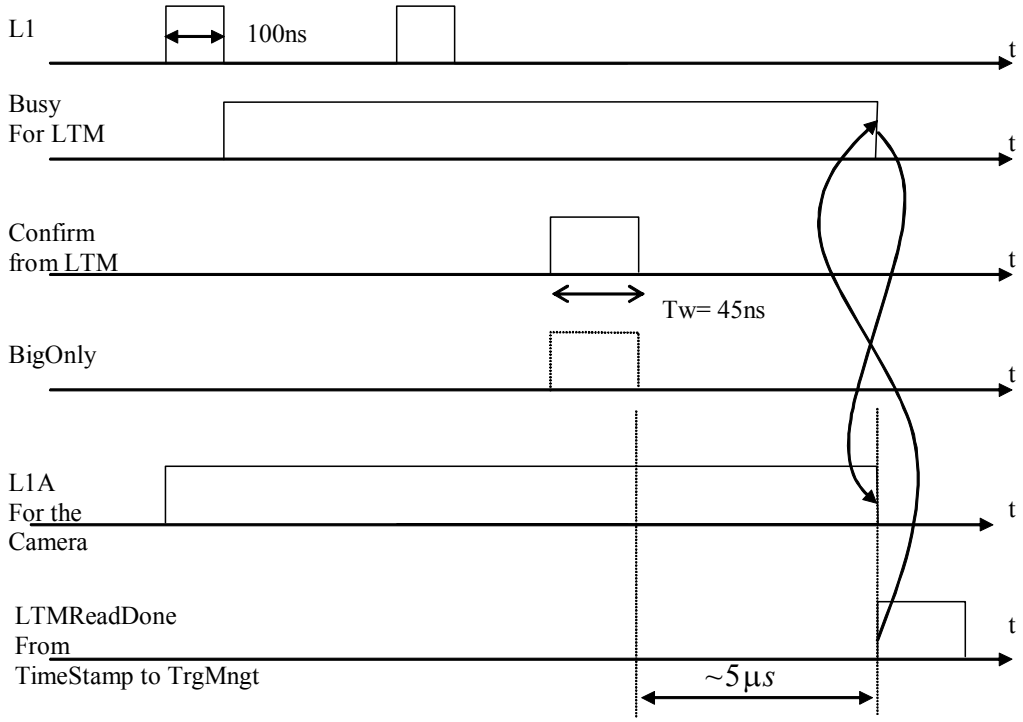


Figure 1.2 : Timing général

2. Fonctionnalités de la carte TrgMngt

La carte **TrgMngt** gère les différentes sources de déclenchement *L1* et *L2* possibles et assure une mise en temps correcte des signaux les uns par rapport aux autres. Cette carte **cPCI** est complètement programmable par l'intermédiaire d'un *FPGA* dédié (circuit *EP1C12Q240C6 Cyclone Altera*).

L'interface avec le bus *cPCI* est réalisé par un *FPGA Altera BGA EP1C4F324C6* de la série *Cyclone*.

D'autre part la carte assure l'interface avec les 9 cartes *cPCI Trigger* et élabore un signal *CaméraTrigger* ('ou' logique des contributions locales).

2.1. Les sources de déclenchement L1

Globalement, 5 sources de déclenchement sont possibles :
- *CaméraTrigger* (généré à partir des cartes Trigger),
- *LED* de calibration,
- *Laser* de calibration,
- *Ext* (générateur par exemple),
- *Logiciel*.

Ces 5 sources peuvent être masquées indépendamment par programmation.

2.2. La mise en temps

L'électronique du *front End* nécessite le respect de temps minimums entre certaines opérations. Par exemple, quand la *FIFO* des cartes analogiques est occupée :

- En lecture, il n'est pas possible d'accepter un nouvel ordre de vidage de la *FIFO*

tant que le précédent n'est pas terminé.

• En écriture, la contrainte est liée au temps de vidage des mémoires analogiques *SAM* qui ne peuvent pas être vidées si elles sont déjà en vidage.

Temps séparant courant-précédent	Temps	Contraintes	Commentaire
L2A et *L2A*	60µs	Lecture **BoxBus-Front End**	Programmable
L2R et *L2R*	1µs	Lecture **RAZ Fifo Front End**	Programmable
L2R et *L2A*	1µs	Lecture **RAZ Fifo Front End**	Programmable
L2A et *L2R*	13µs	Lecture **Fifo Front En**d	Programmable
L1A et *L1A*	1µs	Vidage **SAM**	

Tableau 2.1: Les contraintes temporelles des signaux du trigger

Ces temps sont des contraintes qui ne génèrent pas de temps mort. Par exemple si deux *L2A* sont rapprochés de moins de 60µs, il suffit de retarder le second de façon à être conforme avec le Tableau 2.1.

De façon à pouvoir gérer cette mise en temps, les signaux *L2A* et *L2R* sont entrés dans une *FIFO* de 128 mots de 2 bits gérée par le *FPGA TrgMngt*.

2.3. Le Prescaling

Il est possible de sélectionner, dans le flot de données, les événements qui seront envoyés vers la ferme de processeurs. Cette sélection peut se faire au niveau des signaux de la logique *L1* en ne considérant qu'une partie des triggers ou au niveau de la logique *L2*.

Au niveau de la logique *L2*, il est également possible :

• de réduire le flot de données en rejetant des événement a priori bon,

• d'accepter des événements qui sont considérés comme mauvais ceci afin de tester la qualité des algorithmes de décision de la logique de niveau 2.

2.4. Contrôle de la FIFO de premier niveau

La *fifo* de premier niveau, dans les tiroirs, peut être saturée si le taux de déclenchement *L1* est supérieur au taux de *L2AR*. Dans ce cas, le dernier événement entré peut être incomplet. Pour éviter ce problème, un mécanisme évalue le nombre d'événements dans la *fifo* de premier niveau en fonction des signaux *L1* et *L2AR*. Lorsque le nombre d'événements est supérieur à une valeur programmable *MaxEvt*, la caméra est considérée *Busy* et ne prend plus de déclenchement nouveau tant que les signaux *L2AR* n'ont pas décrémenté le nombre d'événements à une valeur inférieure au seuil *MaxEvt.*

Il est possible de remettre à zéro les compteurs de ce mécanisme par l'ordre *RstEvtCnt*.

L'ordre *EnDisEvtMax* permet de valider ou d'inhiber cette option.

2.5. Fonctionnalités en service

La caméra H.E.S.S. II doit pouvoir fonctionner même si certaines composantes ne

sont pas en service :

• Si la logique de niveau 2 n'est pas en service, la carte *TrgMngt* se substitue à celle-ci en générant un signal *L2A* pour tout *L1* entrant. Cette option est programmable.

• Si le module *LTM* est absent ou non branché (caméra en test par exemple), la carte *TrgMngt* peut générer les signaux nécessaires.

Ces options sont programmables dans le *FPGA TrgMngt*.

3. Programmation

La programmation du *FPGA TrgMngt* se fait par l'intermédiaire du *FPGA cPCI*.

L'espace *BAR1* est utilisé à cet effet.

Les 6 bits de poids faibles, de *UserAddress* sont décodés.

Le Tableau 3.1 résume les ordres, en écriture, acceptés par le *FPGA TrgMngt*.

RstInt : permet de réinitialiser la plupart des éléments du *FPGA TrgMngt*.

RstRS : le signal *L1A* est activé (niveau Haut) par l'une des sources de déclenchement préalablement validées (*Soft*, *LED*, *LASER*, *EXT*) et retourne dans l'état de repos lorsque d'une part le signal *Confirm* a été reçu (du *LMT*) et que les données critiques ont été lues par la carte *TimeStamp* (*LMTReadDone*). Le signal *RstRS* permet de replacer la bascule dans son état de repos. Ce signal ré-initialise la *fifo* de stockage des *L2AR*.

TriggerSoft permet de générer un signal *L1* et de déclencher l'acquisition.

EnDisb (4 bits) permet de valider ou d'inhiber les sources de déclenchement (*LED*, *LASER*, *EXT ou Caméra*). Attention les validations/inhibitions sont individuelles, il est possible de valider plusieurs sources différentes.

Prescale (2 bits) définit le mode de filtrage désiré. Il est possible de filtrer tous les événements *L1* ou encore les événements monoscopiques (mais dans ce cas la *logique L2* ne peut pas être en service). Le mode de filtrage des événements *L2* monoscopiques se fait par programmation dans la *logique L2*. Le filtrage des événements *L2* se fait de la manière suivante : Quand un *L2* est accepté le signal *L2AR* (*Accept TW=30ns*) est généré, quand un signal L2 est filtré alors *L2AR* (*Reject TW=60ns*) est généré.

NFactor définit le facteur de filtrage *N*. Un événement sur *N* sera accepté.

StandAlone permet de placer la caméra *H.E.S.S. II* dans un mode complétement autonome sans relation avec le *LMT*. Ce mode est a priori un mode de test.

EnDisLMTClr : En principe, le *LMT* ne doit pas envoyer de *Reset* à la carte *TrgMngt*. Toutefois, un *Reset* peut être reçu, soit pour rejeter l'événement en cours, soit pour spécifier qu'un problème est survenu pour le trigger courant. Si le signal *Reset* est reçu, la *logique L2* doit générer un *L2R* (Rejet). Un signal de rejet est dans ce cas envoyé à la *logique L2*. L'ordre **EnDisLMTClr** permet de masquer la ligne *Reset*.

L2OnOff permet de spécifier si la *logique L2* est active ou non. Si la *logique L2* n'est pas active, la carte *TrgMngt* se substitue à celle-ci et génère *L2A* pour chaque trigger *L1* entrant.

CntRstL2 permet de charger les 16 bits d'un compteur définissant la période de remise à zéro des compteurs d'événements situés dans les *FPGA* des cartes *Trigger*.

Période minimale : 1,96ms

Période maximale : 128 s

Les compteurs d'événements sont incrémentés avec le signal *L2A*.

MeanCntEvt contient la valeur moyenne du nombre d'événements dans la *fifo* de premier niveau (le moyennage a lieu sur un temps d'environ 2*ms*).

Ordre	*UserAddress [5:0]*	*UserData*	*Commentaire*	*Défaut*
RstInt	000111		Reset programmable	
RstRS	000001		Reset de la bascule génération *L1A* Reset de la *fifo L2AR*	
TriggerSoft	000010		Déclenchement programmé	
EnDisb	000100	 UserData[0] UserData[1] UserData[2] UserData[3]	Validation des sources de déclenchement Trigger LED Trigger LASER Trigger Externe Trigger caméra	 0 0 0 0
RstEvtCnt	000110		Reset des compteurs *L1* & *L2* déterminant le nombre d'événements dans la fifo de premier niveau	
Prescale	001000	UserData[0] UserData[1]	Filtrage de tous les *L1* (*1/N*) Filtrage que des événements mono. (uniquement si *L2* off)	0 0
NFactor	001001	UserData[7:0]	Facteur de filtrage *N* pour l'ordre **Prescale**	
MaxEvt	001010	UserData[7:0]	Nombre maximum d'événements dans la fifo de premier niveau	4
EnDisEvtMax	001011	UserData[0]	1 valide le mécanisme de contrôle	1
StandAlone	001100	UserData[0]	si 1 mode autonome (n'utilise pas le *LMT*)	0
EnDisLMTClr	001101	UserData[0]	1 valide le Clear du *LMT*	0
L2OnOff	001111	UserData[0]	1 valide la logique *L2*	1
L2AL2A	010000	UserData[15:0]	Temps séparant 2 *L2A*	7FF (60µs)
L2AL2R	010001	UserData[15:0]	Temps séparant *L2A* et *L2R*	1F4 (13µs)
L2RL2AR	010010	UserData[15:0]	Temps séparant *L2R* et *L2A* ou *L2R*	32H (1µs)
CntRstL2	100000	UserData[15:0]	Périodicité des RAZ des compteurs d'événements	0x01ff pour 1s
MeanCntEvt	011000	UserData[15:0]	Lecture du nombre moyen d'événement dans la *fifo* 1	

Tableau 3.1 : Les ordres en écriture du FPGA TrgMngt

En lecture *BAR1*, le numéro de version est accessible sur 16 bits :
UserAddress : "000001", *Version*= [*yyxx*] avec *yy* l'année et *xx* un numéro d'ordre.

4. Codage du mot *DEVICE_ID fpga cPCI*

Le *FPGA cPCI* possède un espace de configuration dont le contenu du mot *Device_ID* est utilisé par le système pour déterminer le driver logiciel à utiliser.

Pour la carte *TrgMngt* : *Device_ID=X"2DDD"*.

5. Exemple d'utilisation du logiciel de base sous *Linux* 2.6

Sous *Linux*, se loger comme *root*.

cd /home/huppert/SBig/Driver/TmgTs
./tmgts_mkdev
insmod xpc.ka
insmod tmgts.ko
Pilot/TmgTsPilot

INTERFACE COMPACTPCI (cPCI)

1. Introduction

Ce document décrit l'interface de base *CompactPci* développée pour l'expérience Hess.

Une carte *cPCI* peut être maître, cible ou maître/cible. Le maître est celui qui initie l'échange (carte processeur par exemple). La cible est adressée par le maître. L'interface proposée est une cible *cPCI*.

La norme *PCI* permet de définir 3 espaces différents : mémoire, *E/S* et configuration. L'interface décrite superpose l'espace mémoire et l'espace *E/S*. Un cycle dans l'un ou l'autre des ces espaces accède à la même donnée. A l'initialisation, seul l'accès à l'espace de configuration est possible.

2. L'espace de configuration

L'interface de base *CompactPci* implémente un bloc de 64 registres de 32 bits et quelques registres de contrôle. Dans le cas de périphériques multi-fonctions, plusieurs blocs peuvent être implémentés. Le mécanisme d'accès à ce bloc est indépendant de celui mis en oeuvre pour l'espace mémoire ou pour l'espace entrée-sortie classique. L'accès est défini par le signal *PciIdSel* ='1'.

Adresse	31..24	23..16	15..8	7..0
00H	DEVICE ID		VENDOR ID	
04H	STATUS REGISTER		COMMAND REGISTER	
08H	CLASS CODE			REVISION ID
0CH	BIST	HEADER TYPE	LATENCY TIMER	CACHE LINE SIZE
10H	BASE ADDRESS REGISTER 0			
14H	BASE ADDRESS REGISTER 1			
18H	BASE ADDRESS REGISTER 2			
1CH	BASE ADDRESS REGISTER 3			
20H	BASE ADDRESS REGISTER 4			
24H	BASE ADDRESS REGISTER 5			
28H	CARDBUS CIS POINTER			
2CH	SUBSYSTEM ID		SUBSYSTEM VENDOR ID	
30H	EXPANSION ROM BASE ADDRESS			
34H				CAP POINTER
38H	RESERVE			
3CH	MAX LATENCY	MINIMUM GRANT	INTERRUPT PIN	INTERRUPT LINE
40H-FCH	ESPACE UTILISATEUR[a]			

Tableau 2.1 : L'espace de configuration

a. Certains registres sont utilisés : *Reg80*, *Reg84*

Les parties grisées sont optionnelles et non implémentées.

2.1. Détails des registres utilisés dans l'espace de configuration

Chaque mot (32 bits) du bloc de configuration peut être constitué d'un ou plusieurs registres. L'utilisateur accède à un mot par son adresse.

D'un point de vue interne, chaque mot est représenté par un registre dont le nom est : *RegXX* ou *XX* dénote l'adresse du mot. Par exemple, *Reg10* est le mot de 32 bits localisé à l'adresse 10H du bloc de configuration.

Chaque registre utilisé est décrit ci-dessous, un registre peut être en :

• lecture seule [RO],
• écriture seule [WO],
• lecture/écriture [RW]

Dans certains cas, un registre peut avoir des bits en [RO] et d'autres en [RW].

Les différents registres de l'espace de configuration décrivent et contrôlent les principales parties de l'interface.

Généralement, une fonction est implémentée si le ou les bits correspondants sont en lecture-écriture. L'utilisateur devra lancer une succession de cycles d'écriture et de lecture afin de déterminer les registres et les fonctions implémentés.

2.1.1. Vendor ID (00H) [RO] : *Reg00*(15..0)

Ce registre identifie le constructeur de l'interface. Le groupe *Pci Sig*[1] fournit des numéros valides.

Valeur par défaut : 0H. (temporaire).

2.1.2. Device ID (00H) [RO] : *Reg00*(31..16)

Ce registre identifie le type de périphérique. Ce numéro est fourni par le concepteur et doit être précâblé selon l'application. En effet, en fonction du Vendor ID et du Device ID le logiciel pourra charger automatiquement le bon pilote. Par exemple :

interface BoxBus :	AAAAH
interface CustomBus :	BBBBH
contrôle température :	CCCCH

2.1.3. Command Register (04H) [RW] : *Reg04*(15..0)

Ce registre permet un contrôle général de l'interface *Cpci*.

1. *Pci Sig* est l'organisation chargée de la normalisation et de la promotion du *Pci*.

Bits		Valeur par défaut	
15..10	Réservés		
9	Fast Back-to-Back Enable	0	RO
8	Validation Erreur Système	0	RW
7	Stepping Control	0	RO
6	Validation Erreur Parité	0	RW
5	VGA Palette Snoop	0	RO
4	Memory Write and Invalidate	0	RO
3	Cycles spéciaux	0	RO
2	Bus Maître	0	RO
1	Validation mémoire	0	RW
0	Validation *E/S*	0	RW

Tableau 2.2 : Le registre de commande

Seuls les bits en lecture-écriture (R/W) peuvent être modifiés.

L'interface peut générer des signaux sur les lignes *PciPerr#* et *PciSerr#* en cas d'erreur de parité sur les données et/ou l'adresse :

bit 6 = 1, l'interface peut signaler une erreur de parité sur les données.

bit 8 = 1, l'interface peut signaler une erreur de parité sur l'adresse si le bit 6 =1.

2.1.4. Status Register (04H) [RW] : *Reg04*(31..16)

Les 16 bits de poids forts du registre *Reg04* implémentent un registre d'état de l'interface.

Bits		Valeur par défaut	
31	Erreur parité détectée	0	RW
30	Erreur système signalée (*Serr#*)	0	RW
29	Pour maître uniquement	0	R
28	Pour maître uniquement	0	R
27	*Target Abort* signalé	0	RW
26..25	*DevSel* timing	Slow : 10	R
24	Pour maître uniquement	0	R
23	Possibilité enchaînements rapides	1	R
22	Réservé	0	
21	Possibilité 66 MHz	0	R
20	Liste secondaire	0	R
16..19	Réservé	0000	R

Tableau 2.3

bit 21 : 0 correspond à une horloge de 33 MHz.

Remarque : Pour remettre à 0 les bits 31, 30 et 27 il faut écrire un 1 dans ces bits. La méthode consiste à écrire dans ce registre la valeur précédemment lue.

2.1.5. ClassCode&RevisionID Register (08H) [RO]

Le Registre ClassCode est composé de 3 registres de 8 bits utilisés pour qualifier l'interface :

Class Code : *Reg08*(31..24) = 11H (DAQ & DSP controllers)
Sub Class : *Reg08*(23..16) = 80H (Others DAQ & DSP controllers)
ProgIntByte : *Reg08*(15..8) = 00H (défini le niveau final de granularité)
Revision ID : *Reg08*(7..0) = 00H (défini le numéro de version de l'interface).

2.1.6. Header Type Register (0CH) : Reg0C(23..16) [RO]

Le Tableau 2.1, page 91 défini le type de configuration implémenté dans l'interface (Type 0).

Reg0C(22..16) = 0H (type 0)
Reg0C(23) = 0, l'interface n'est pas multifonctions (un seul *ClassCode*, etc.)

2.1.7. Base Address Register (10H à 28H) : (*BAR*) [RW]

2 registres *BAR* sont implémentés (2 parmi 6 possibles) : *BAR0* et *BAR1*. Chaque

registre permet de définir d'une part un espace mémoire et d'autre part l'adresse de base de cet espace.

	Adresse	Bits	Valeur		
BAR0 *BAR1*	10H 14H	0	0	0 = mémoire 1 = *E/S*	RO
		2..1	00	décodeur adresse 32 bits situé entre 0 et 4GB	RO
		3	1	Prefetch	RO

Tableau 2.4

Les 4 bits de poids faibles n'interviennent pas pour déterminer l'espace mémoire utilisé. Le minimum théorique est donc de 16 octets.

	Bits		Valeur	Espace mémoire
BAR0	16..4	RO	0H	
BAR0	31..17	RW		2^{17}=128KB
BAR1	12..4	RO	0H	
BAR1	31..13	RW		2^{13}=8KB

Tableau 2.5

Principe d'utilisation des registres *BAR*

1- Pour chaque *BAR* possible (0 à 5) l'utilisateur tente de modifier son contenu. Seuls les registres *BAR0* et *BAR1* sont modifiables donc implémentés.

2- Pour chaque *BAR*, l'utilisateur écrit FFFFFFFFH et relit la valeur écrite :

Read (*BAR0*) => FFFE0008H

bit3=1 : Prefetch

Le premier bit à 1 définit l'espace mémoire utilisé par *Bar0* : 17 soit 128KB.

Read (*BAR1*) => FFFFE008H

bit3=1 : Prefetch

Le premier bit à 1 définit l'espace mémoire utilisé par *Bar1* : 13 soit 8KB.

3- L'utilisateur écrit ensuite dans *BAR0*(31..17) et *BAR1*(31..13) les adresses de base des espaces respectifs.

L'interface accepte tous les cycles avec pour adresse de base *BAR0* ou *BAR1*.

2.1.8. SubSystem Vendor ID (2CH) [RO] : *Reg2C*(15..0)

Valeur par défaut : 0H (pas de *SubSystem Vendor ID* associé à l'interface).

2.1.9. SubSystem ID (2CH) [RO] : *Reg2C*(31..16)

Valeur par défaut : 0H (pas de *SubSystem ID* associé à l'interface).

2.1.10. Interrupt Line (3CH) [RW] : *Reg3C*(7..0)

Valeur par défaut : FFH, spécifie que la redirection des interruptions n'a pas encore été appliquée à la fonction de l'agent.

2.1.11. Interrupt Pin (3CH) [RO] : *Reg3C*(15..8)

Valeur par défaut : 01H, spécifie que la ligne *Inta*# est utilisée.

2.2. Les autres registres de l'espace de configuration (espace utilisateur)

2.2.1. Le registre d'interruption (80H) [RW] : *Reg80*(15..0)

Le registre utilisateur *Reg80* est utilisé par l'interface pour la gestion des interruptions utilisateurs. Le système d'interruption est décrit dans le paragraphe §11. page 104. Le système maître peut à partir de ce registre déterminer l'origine de l'interruption, la masquer ou encore la désactiver.

Bits		Valeur par défaut	
15..12	Réservés	0H	RO
11..8	Masque	0H	RW
7..4	Réservés	0H	RO
3..0	*UserInt3..UserInt0*	0H	RW

Tableau 2.6 : Le registre d'interruption

Pour remettre à zéro une interruption il suffit d'écrire un '1' dans le bit correspondant. La méthode préconisée pour désactiver les interruptions éventuelles est de lire *Reg80* puis de réécrire la valeur lue. Par exemple, Si *Reg80*=F01H (*Masque*=FH, *UserInt0*=1). L'écriture de F01H dans *Reg80* aura pour effet de remettre à zéro *UserInt0*.

Le masque agit après le registre, ce qui signifie qu'il est possible de tester les sources d'interruption sans être interrompu.

Chaque Bit à 1 du masque valide l'interruption correspondante :
Reg80(11) valide *UserInt3*,
Reg80(10) valide *UserInt2*,
Reg80(9) valide *UserInt1*,
Reg80(8) valide *UserInt0*.

2.2.2. Le Compteur utilisateur (80H) [RW] : *Reg80*(31..16)

La partie supérieure du registre *Reg80* reflète l'état du compteur utilisateur de 16 bits. Ce registre peut être lu ou écrit. L'utilisateur peut initialiser une valeur de début de comptage. Zéro est une valeur valide.

2.2.3. Le registre de contrôle du pipeline (84H) [RW] : *Reg84*(3..0)

Dans le cas de lecture en mode rafale, l'interface doit anticiper les lectures sur le port *UserData*. Le retard entre la donnée entrante sur le port *UserData* et la donnée écrite sur le bus *Cpci* est de 2 périodes d'horloge.

Quand la fin de cycle *Cpci* est détectée par l'interface, 2 données supplémentaires ont été sorties du dispositif connecté sur le port *UserData*. Pour une mémoire adressée, ceci ne pose pas de problème particulier. Dans le cas d'une *fifo*, il est fondamental de récupérer ces 2 données.

L'interface implémente un pipeline de longueur 2 qui conserve les 2 dernières lectures. Il est possible de lancer la lecture d'une rafale de mots en utilisant ce pipeline. Ce mécanisme permet de lire en priorité ces 2 données.

Bit		Valeur par défaut
Reg84[0]	1 : validation pipeline pour accès Bar0	1
Reg84[1]	1 : validation pipeline pour accès Bar1	1
Reg84[2]	1 : validation automatique pour Bar0	0
Reg84[3]	1 : validation automatique pour Bar1	0

Tableau 2.7 : Le registre Reg84

Pour accéder les données du pipeline, il faut activer le bit 0 (accès sur *Bar0*) et/ou 1 (accès sur *Bar1*).

Dans le cas d'une lecture d'un grand nombre de mots, il est possible que le gestionnaire de rafales sur la carte maître subdivise la grande rafale en plusieurs petites rafales. Dans ce cas, il est préférable d'utiliser la validation automatique (Reg84[2]=1 et/ou Reg84[3]=1) qui entre en action à partir de la deuxième rafale :

Par exemple, l'utilisateur désire initier un cycle *DMA* sur *Bar0* (lecture de *fifos*). L'unicité des échanges n'étant pas garanti, il est prudent de procéder de la façon suivante :

• écriture de *Reg84* : 0100->*Reg84*

• lancement du *DMA*

• Si le cycle *DMA* est subdivisé en plusieurs rafales, la première aura lieu sans pipeline (les premiers mots proviennent directement de la *Fifo*). Après la première rafale, *Reg84* passe automatiquement à la valeur 0101.

• les autres rafales (du même cycle *DMA* initial) utilisent le pipeline (récupération des données prélues).

Cette opération devra être répétée pour chaque cycle *DMA* (en lecture uniquement) initié par l'utilisateur.

Il est très important de programmer correctement *Reg84* avant tout accès sur le port *UserData*.

Reg84[2]=1 à pour effet de modifier dynamiquement *Reg84*[0] alors que *Reg84*[3] modifie *Reg84*[1] .

3. Le bus de commande

Les lignes *PciCbe*[3..0] sont utilisées, lors de la phase d'adressage, pour définir le type de la transaction courante.

Le tableau 3.1 indique les différentes commandes *Pci*. Les commandes grisées ne sont pas implémentées.

PciCbe[3..0]	Commande
0000	Interrupt Acknowledge[a]
0001	Special cycle
0010	I/O Read
0011	I/O Write
0100	Réservé
0101	Réservé
0110	Memory Read
0111	Memory Write
1000	Réservé
1001	Réservé
1010	Configuration Read
1011	Configuration Write
1100	Memory Read Multiple
1101	Dual Address Cycle
1110	Memory Read Line
1111	Memory Write and Invalidate

Tableau 3.1 : Les commandes Pci

a. Ignoré, voir le paragraphe sur la gestion des interruptions.

4. Les cycles de configuration

Un cycle de configuration est défini par le signal issu du maître *PciIdSel*='1'.

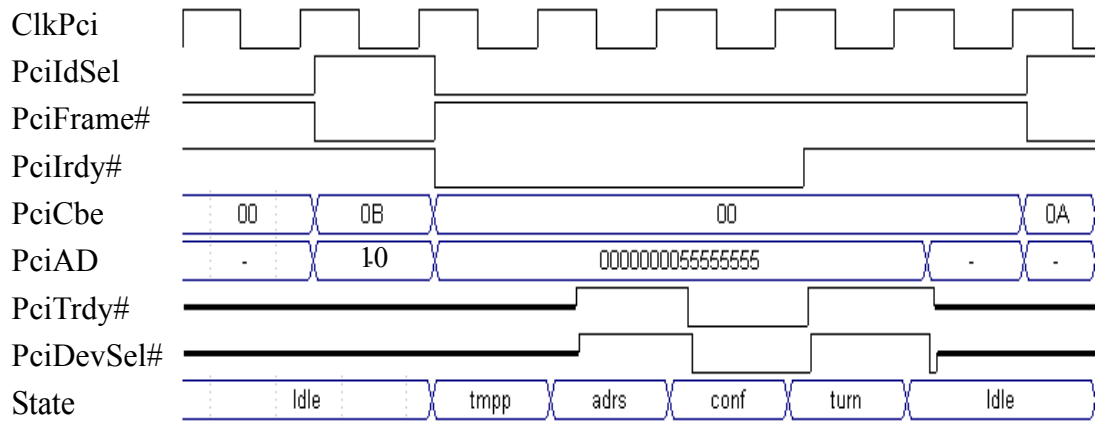


Figure 4.1: Ecriture d'un registre de configuration

Cycle de configuration en écriture (*PciCbe*=0B), le registre de configuration *BAR0* (10H) est écrit avec la donnée 55555555H.

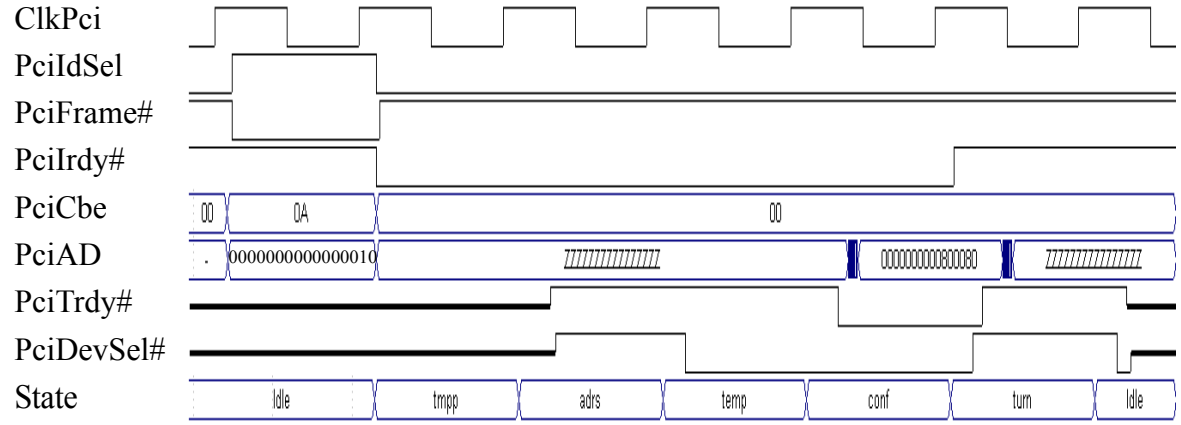


Figure 4.2: Lecture d'un registre de configuration

Cycle de configuration en lecture (*PciCbe*=0A), le registre de configuration *BAR0* (10H) est lu : *PciAD*=00800080H.

5. Écriture mémoire

Un cycle d'écriture est défini par le bus commande *CBE*[3..0]=0111 (tableau 3.1) et par une adresse reconnue par l'un des registres *BAR*.

5.1. Écriture d'un mot

La figure 5.1 montre l'écriture de la donnée 12345678H à l'adresse 55555554H. La partie haute de l'adresse est dans le champ du registre *BAR0*. Dans le cas d'un transfert 64 bits, 14 bits sont transmis sur le bus adresse utilisateur, les 3 bits de poids faibles sont supprimés (Il faut 8 octets pour former 1 mot de 64 bits). *UserAd* : 2AAAH. La donnée est

transmise sur le bus donnée utilisateur (*UserData*). Le signal utilisateur *WrBar0* valide un cycle utilisateur en écriture. *WrBar0* doit être utilisé comme validation d'un registre dont la prise en compte est l'horloge utilisateur.

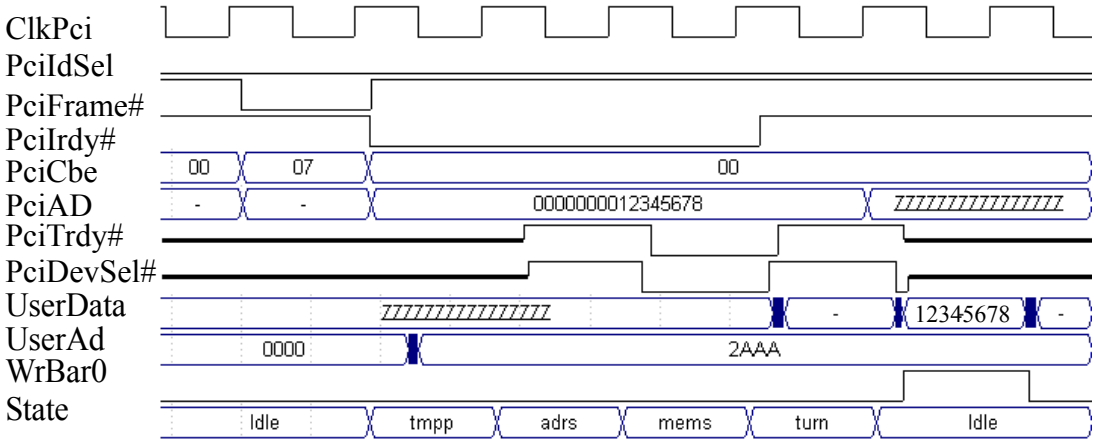


Figure 5.1 : Écriture d'un mot mémoire

5.2. Écriture d'une rafale de mots

La figure 5.2 montre l'écriture de 5 mots à partir de l'adresse 55555554H. La partie haute de l'adresse est dans le champ du registre *BAR0*. Dans le cas d'un transfert 64 bits, 14 bits sont transmis sur le bus adresse utilisateur, les 3 bits de poids faibles sont supprimés (Il faut 8 octets pour former 1 mot de 64 bits). *UserAd* : 2AAAH.

5 données sont transmises sur le bus *UserData*. Pour chaque donnée de 64 bits, le bus *UserAd* est incrémenté : 2AAA, 2AAB, 2AAC, 2AAD et 2AAE. Quand la phase initiale d'adressage *Pci* est terminée, l'interface accepte, en écriture, un mot de 64 bits toutes les 30 ns. Soit un débit de 266 Mo.S^{-1}.

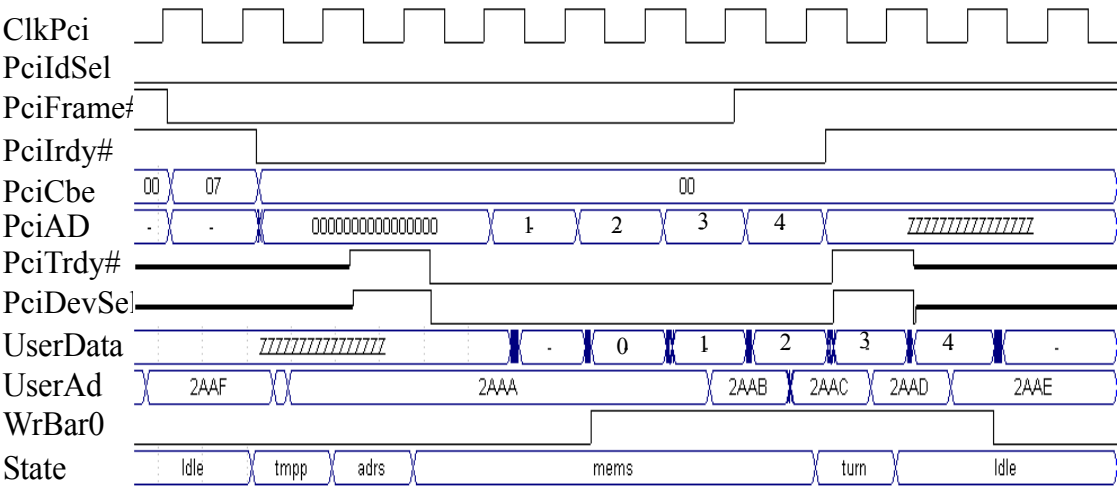


Figure 5.2 : Ecriture d'une rafale de mots

5.3. Écriture d'une rafale de mots avec état d'attente utilisateur

La figure 5.3 montre les cycles d'écriture d'un maître à l'adresse *Cpci* : 55555554H

de 7 mots notés de 1 à 7. L'adresse utilisateur *UserAd* sera 2AAAH pour le premier mot et 2AB0H pour le mot 7. L'utilisateur demande d'insérer un état d'attente (*UserWaitState* activé), en même temps qu'il reçoit la donnée 2. *UserWaitState* est pris en compte, par l'interface, par un front montant de l'horloge. Sur le front d'horloge montant suivant, la cible indique au maître qu'elle n'est plus prête (*PciTrdy#* état haut). En conséquence, le maître maintiendra la donnée 6 un cycle d'horloge supplémentaire avant d'envoyer la donnée 7. Le signal d'écriture utilisateur *WrBar0* devient inactif pendant un cycle d'horloge.

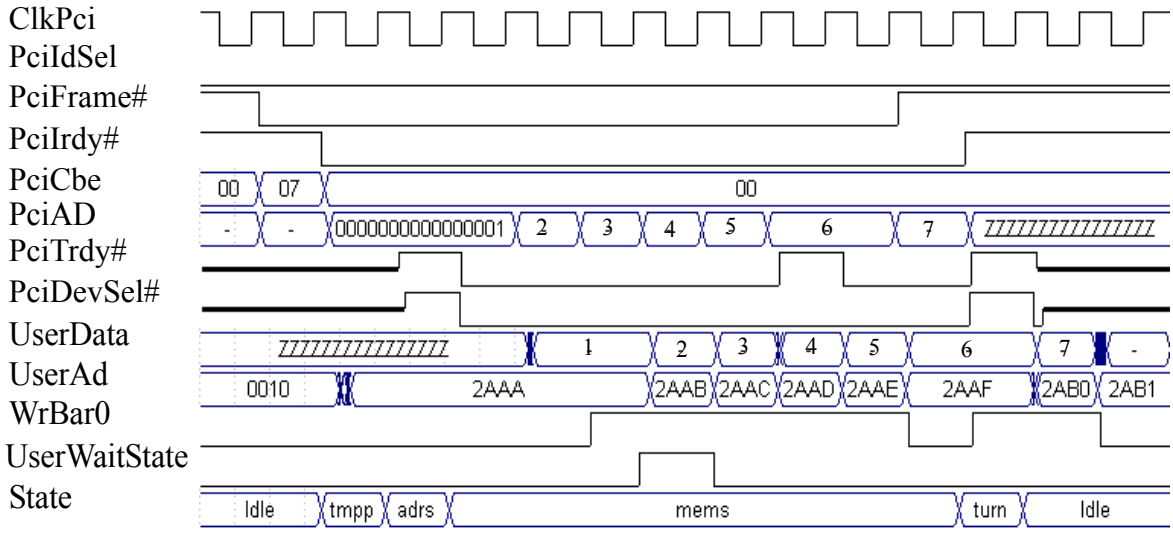


Figure 5.3 : Ecriture en rafale avec état d'attente

6. Lecture

6.1. Lecture d'un mot

La figure 6.1 montre la lecture de la donnée 01234567H à l'adresse *Cpci* : 55555554H. 14 bits sont transmis sur le bus adresse utilisateur, les 3 bits de poids faibles sont supprimés (Il faut 8 octets pour former 1 mot de 64 bits). *UserAd* : 2AABH. La donnée utilisateur est placée sur le bus utilisateur (*UserData*). Le signal utilisateur *RdBar0* valide l'horloge de lecture de la donnée utilisateur.

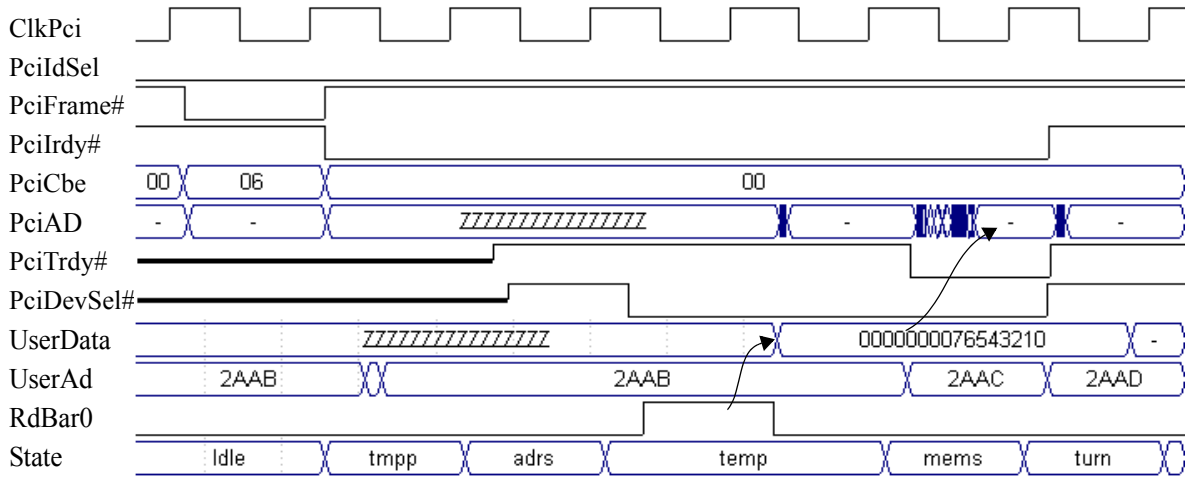


Figure 6.1 : Lecture d'un mot

6.2. Lecture d'une rafale de mots

Une rafale de 9 mots (D0 à D8) est lue à partir du port *UserData* à l'adresse 15554H. Le bus *UserAd* est incrémenté à partir de l'adresse 2AABH.

Du fait de l'anticipation de la lecture et du buffer interne de l'interface, 2 données supplémentaires sont lues à partir du port *UserData*. Ceci ne pose aucun problème dans le cas de lecture d'une mémoire adressée. Par contre, dans le cas d'une *Fifo* toute donnée sortie et non lue est perdue ! A cet effet un mécanisme spécial de l'interface permet de contourner ce problème (Voir " § 2.2.3. page 97 ").

Le signal *RdBar0* valide l'horloge pour lire les données. L'état d'attente se retrouve également sur la ligne *RdBar0*.

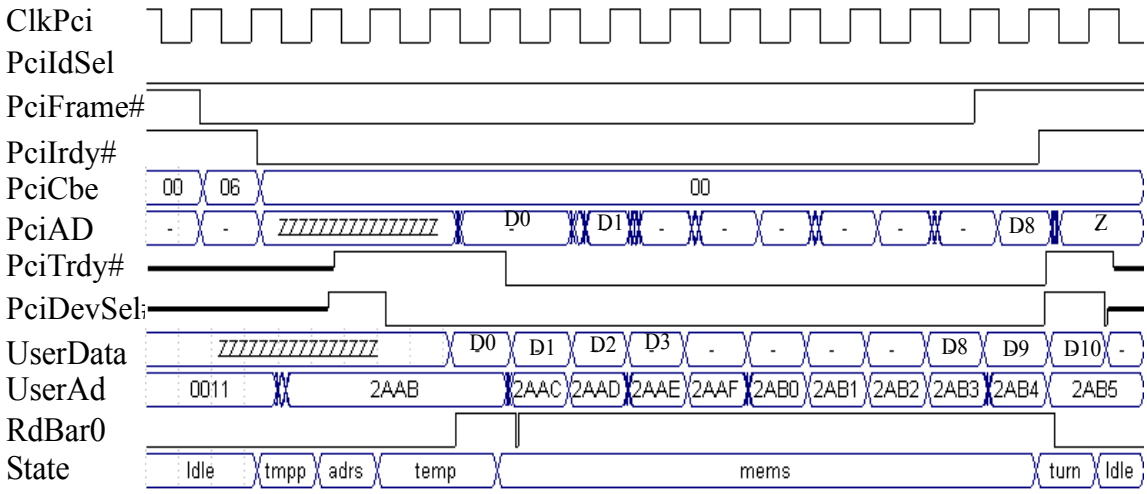


Figure 6.2 : Lecture d'une rafale de mots

7. Lecture d'une rafale de mots avec état d'attente utilisateur et démultiplexage

La figure 7.1 montre comment l'interface démultiplexe les données. La carte est définie pour des transferts 64 bits (ligne *User32_64b* à l'état bas) et le maître lance la lecture d'une rafale de mots en mode 32 bits. Dans ce cas, pour chaque lecture du bus utilisateur (*UserData*), 2 mots seront transférés sur le bus *Cpci*.

7 mots de 32 bits sont transférés sur le bus *Cpci* (D0 à D6), pour cela il est nécéssaire de lire 4 mots de 64 bits sur les lignes *UserData*. Un état d'attente est demandé par l'utilisateur (*UserWaitState*) ce qui provoque le maintient de la donnée D4 une période d'horloge suplémentaire.

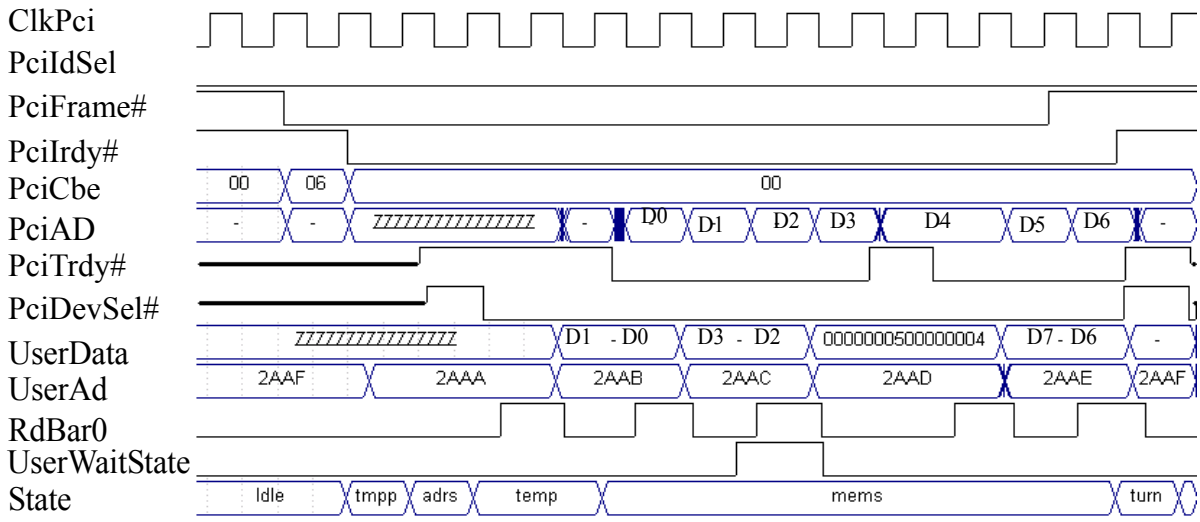


Figure 7.1 : Lecture avec état d'attente et démultiplexage

8. Lecture et écriture en mode 32 bits

Si l'interface est déclarée 64 bits (User32_64b à l'état bas) , l'initiateur peut lancer des cycles de lecture ou d'écriture.

En lecture si le bit PciAD(2)=0 (lecture d'une adresse 32 bits paire), un mot de 64 bits est lu dans l'interface et seule la partie basse de ce mot est transférée sur le bus CPci.

Pour une seconde lecture avec le bit PciAD=1 (lecture d'une adresse 32 bits impaire), aucune lecture n'est demandée sur le bus User mais la partie haute du mot de 64 bits préalablement chargée est transmise sur le bus CPci.

En écriture 32 bits, le mot D(31:0) qui sera écrit sur User(31:0) est automatiquement dupliqué sur User(63:32).

9. Arrêt d'un transfert par l'utilisateur

L'utilisateur peut demander l'arrêt du cycle (uniquement en mode rafale) en activant la ligne *UserStop*. La cible prévient le maître en activant a son tour *PciStop*# (actif bas). Dans l'exemple de la figure 9.1, seules les trois premières données sont transmises.

Le signal *UserStop* peut être activé aussi bien en lecture qu'en écriture. Si cette possibilité n'est pas utilisée, il est nécéssaire de relier la broche *UserStop* à la masse.

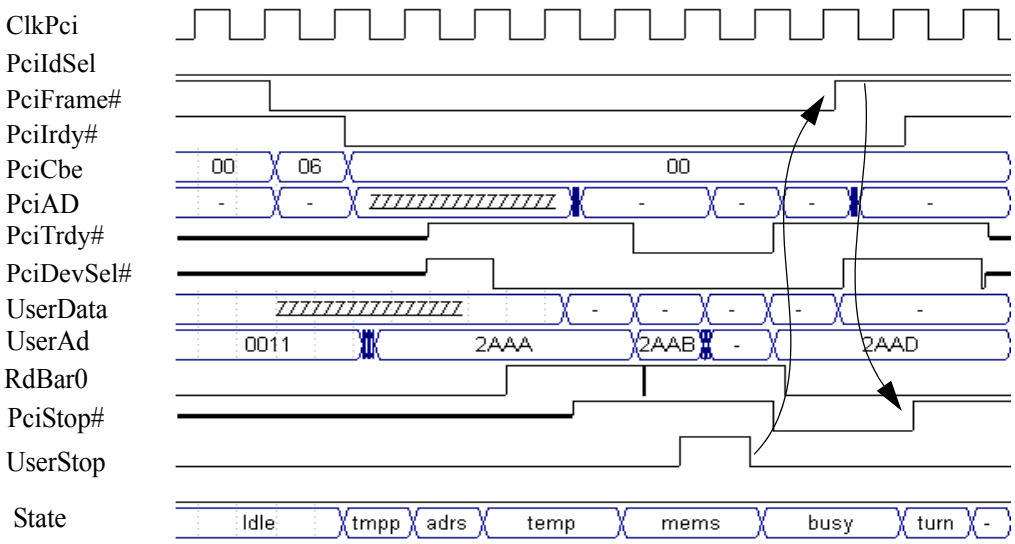


Figure 9.1 : Arrêt d'un transfert par la ligne Stop#

10. Gestion de la parité

Le système qui génère la donnée, crée des bits de parité (parité paire) *PAR* pour couvrir *AD*[31..0] et *Cbe*[3..0] et *PAR64* pour couvrir *AD*[63..32] et *Cbe*[7..4]

Le contrôle de parité est automatiquement géré par le *fpga*.

Une erreur de parité sur les données est indiquée sur la ligne *Perr#*.

Une erreur de parité sur le bus adresse est indiquée sur la ligne *Serr#*.

11. Le système d'interruptions

Un agent *Pci* a accès à 4 lignes d'interruptions : *Inta#*, *Intb#*, *Intc#* et *Intd#* pour interrompre le système maître.

Dans le cas où l'agent n'est pas multi-fonctions (c'est le cas de cette interface), seule la ligne *Inta#* est utilisée.

Plusieurs agents *Pci* peuvent partager les mêmes lignes *Intx#*. (*Inta#* pour les agents non multi-fonctions).

Le système d'interruptions du système peut être complexe, la partie *Pci* n'est qu'un sous ensemble de ce système d'interruptions.

L'organisation générale du système d'interruptions peut varier d'un système à l'autre. Il comprend généralement, un ou plusieurs contrôleurs d'interruptions programmable (*PIC*) associé(s) à un routeur d'interruptions également programmable. La vectorisation des interruptions a lieu à ce niveau.

Compte tenu de ce système, le mécanisme classique d'acquittement d'interruption (cycle *Iack*) où le périphérique ayant positionné l'interruption transmet un vecteur n'est pas applicable à ce niveau. La vectorisation à lieu au niveau système. Ceci est dû au fait que plusieurs agents peuvent être connectés sur la même ligne et que le cycle *Pci Iack* ne donne pas d'adresse spécifique.

Quand le masque d'interruption est bien positionné (Voir " § 2.2.1. page 96 "), l'activation de la ligne *Inta#* interrompra le processeur système. Le système d'interruptions reconnaîtra facilement qu'un agent Pci a activé la ligne *Inta#*. Une routine d'interruptions doit ensuite interroger chaque agent Pci connecté sur *Inta#* pour déterminer la source

exacte de l'interruption.

La procédure pour gérer les interruptions de l'utilisateur est la suivante :

• Ecriture du masque d'interruption dans le registre 80H de l'espace de configuration.

• En cas d'interruption, lire le registre 80H de l'espace de configuration afin de déterminer la source de l'interruption (*Reg80*[3:0]).

• Remettre à zéro l'interruption en réécrivant dans le registre 80H la valeur précédemment lue. Par exemple si *Reg80*[15:0]=F01H, alors les 4 interruptions sont validées et *UserInt0* a été activée. Ecrire F01H dans *Reg80*[15:0], a pour effet de valider à nouveau les interruptions et de remettre à zéro *UserInt0* (Reg80[15:0]=F00H).

Pour générer une interruption, l'utilisateur doit activer (niveau haut) une ou plusieurs lignes : *UserInt0*, *UserInt1*, *UserInt2*, *UserInt3*. Le signal d'interruption doit avoir une largeur *T>TPciClk* pour être certain d'être pris en compte.

Les entrées d'interruptions non utilisées doivent être reliées à la masse.

12. Le compteur utilisateur

Un compteur de 16 bits est implémenté dans l'interface. Il est possible de manipuler ce compteur par programmation (Voir " § 2.2.2. page 97 ") ou par des signaux utilisateur (carte).

Une transition *Bas ->Haut* sur la ligne *UserClkCpt* incrémente le compteur.

Un niveau Haut sur la ligne *UserRstCpt* initialise à 0 ce compteur.

Si cette possibilité n'est pas utilisée, la broche *UserClkCpt* doit être reliée à la masse.

13. Remarque sur les états d'attente

Un état d'attente utilisateur (ligne *UserWaitState*) doit avoir une largeur *T>TPciClk* pour être certain que celui-ci soit pris en compte. La ligne *UserWaitState* est d'abord échantillonnée par l'horloge *PciClk*, le signal *PciTrdy#* est remonté avec le front d'horloge suivant. Il existe donc, un délai à bien maîtriser entre *UserWaitState* et *PciTrdy#*. Voir la *figure 5.3* et la *figure 7.1*.

Si la ligne *UserWaitState* n'est pas utilisée, celle-ci doit être reliée à la masse.

14. L'interface utilisateur

L'interface utilisateur est résumée par la figure 14.1 en 5 groupes de signaux :

• Type Interface,
• Données,
• Compteur,
• Interruptions,
• Arrêt, attente.

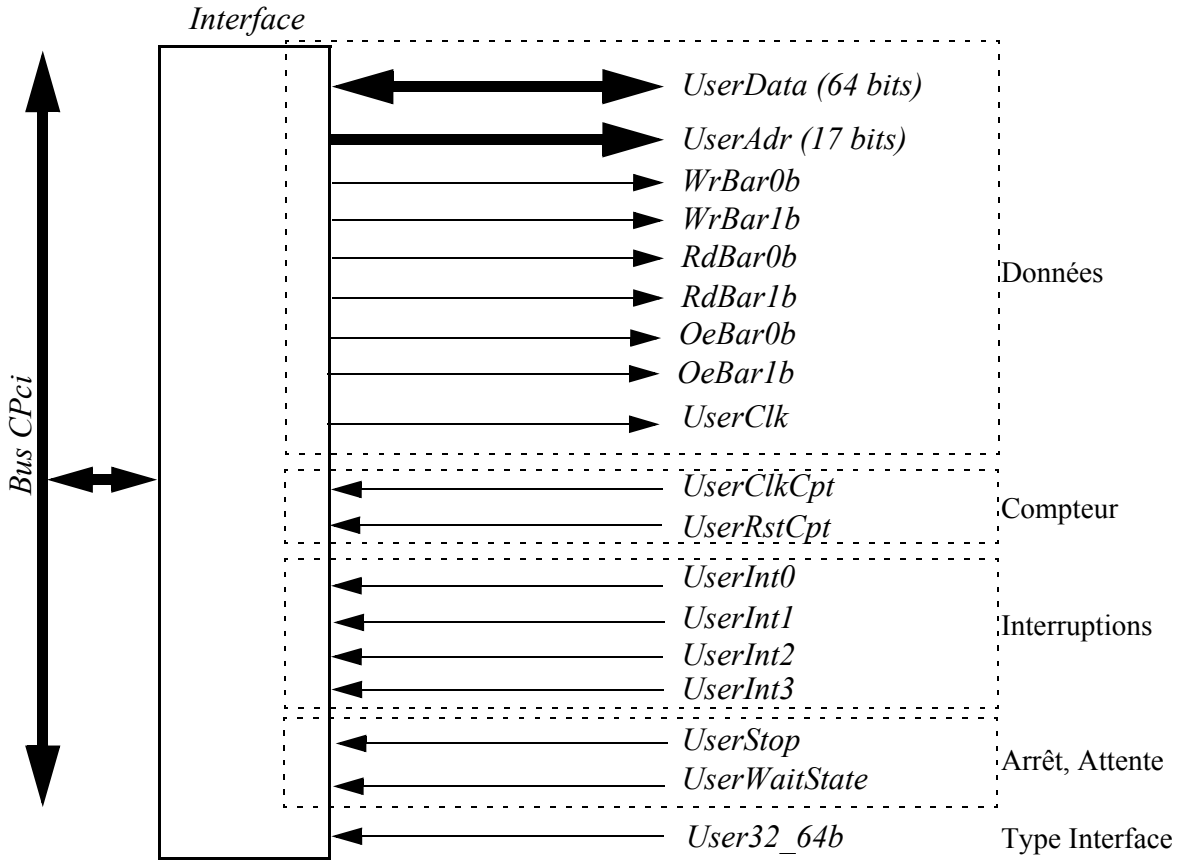


Figure 14.1 : L'interface utilisateur

14.1. Type Interface

Le type de l'interface est défini par un niveau par la ligne *User32_64b*.
• niveau bas : l'interface traite des données 64 bits,
• niveau haut : l'interface traite des données 32 bits.

Il est préférable que les largeurs des bus *Cpci* et interface utilisateur soient compatibles. Toutefois, en lecture uniquement, il est possible de définir l'interface 64 bits avec un *Cpci* de 32 bits. Dans ce cas, l'interface *Cpci* (*fpga*) se charge de transférer successivement 2 mots de 32 bits pour chaque mot de 64 bits utilisateur.

14.2. Données

Il est possible d'accéder à 2 espaces utilisateurs. Ces espaces sont définis par les registres *Bar0* et *Bar1* du *fpga* (Voir " § 2.1.7. page 94 ").

Les accès aux espaces 0 et 1 sont exclusifs, le bus adresse et le bus données sont partagés par ces 2 espaces. Des signaux de contrôle définissent lequel des 2 espaces est utilisé.

Pour chaque espace correspond un signal de validation de lecture et un signal de validation d'écriture (*RdBar0b*, *WrBar0b, RdBar1b*, *WrBar1b*), ainsi qu'une commande de validation des buffers 3-états sur le bus (*OeBar0b*, *OeBar1b*). Les signaux *OeBar0b* et

OeBar1b ne sont gérérés que pour les cycles de lecture.

En lecture, une donnée doit être placée sur le bus *UserData* lors de la transition (*bas-> haut*) du signal *UserClk* lorsque la ligne *RdBar(0,1)b*='0'.

En écriture, une donnée doit être prise du bus *UserData* lors de la transition (*bas-> haut*) du signal *UserClk* lorsque la ligne *WrBar(0,1)b*='0'.

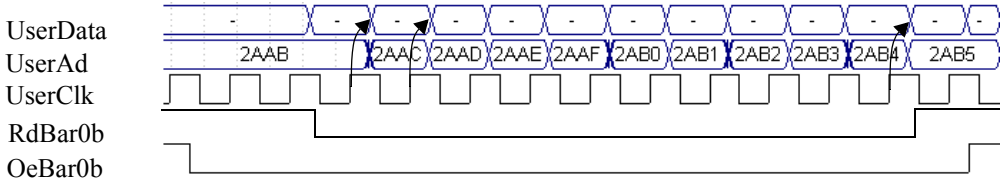


Figure 14.2 : Les signaux de contrôle en lecture

15. Configuration de l'interface

Il est possible de changer les valeurs des champs **DeviceID**, **VendorID** et de définir si l'interface utilisateur est 32 ou 64 bits.

Cette opération nécessite une recompilation du projet sous *QUARTUS* (*Altera*).

La figure 15.1 montre l'arborescence des fichiers *VHDL* sous *QUARTUS*.

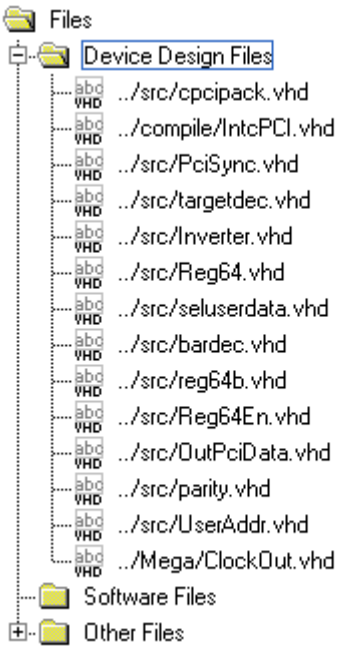


Figure 15.1 : Les différents fichiers utilisés par l'interface

Il est nécessaire d'ouvrir le fichier : *cPCIPackage.vhd* et de modifier les constantes hexadécimales.

Par exemple, dans le fichier ci-dessous le **VendorId** est celui du *CERN*, le **DeviceID** est celui de la carte *FifoSlc* et l'interface utilisateur est de 64 bits.

```
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"2AAA"; -- FifoSlc
    -- Use a valid vendor id, for ex. the cern device id=10DCH
    constant VendorId: STD_LOGIC_VECTOR(15 downto 0):=X"10DC";   -- CERN VendorID
    -- User32_64b defines if the user interface is 32 or 64-bit
    constant User32_64b : std_logic := '0';                                  -- 0=64bits, 1=32bits
end cpciPack;
```

CARTE FIFO CPCI SLC

1. Introduction

La carte *fifo cPCI Slc* est basée d'une part, sur l'utilisation du bus *compactPCI* et d'autre part sur l'utilisation du *BoxBus Slc*.

L'interface avec le bus *compactPCI* est effectuée par un *fpga Altera BGA EP1C4F324C6* de la série *Cyclone*. La fonctionnalité de ce circuit est complètement décrite dans une note[1]. Le *BoxBus Slc* est un bus série, basé sur une structure de messages, élaboré pour l'expérience *HESS*.

La partie *BoxBus Slc* a été décrite dans les notes précédentes.

Un système de *fifos* est utilisé, en lecture, pour transférer des messages vers le *cPCI*. Un autre système de *fifos* est utilisé, en écriture, pour transférer des messages vers le *BoxBus Slc*.

La carte peut gérer 4 *BoxBus Slc*. Les chemins de données de l'interface *fifo cPCI* sont représentés par la figure 1.1.

L'interface avec les *BoxBus Slc* ainsi que le contrôle de la carte est effectué par un *fpga Altera EP1C12Q240C6*. Afin de limiter les temps de transfert avec le processeur, les données sont transférées entre les *fifos* et le *cPCI* sur une largeur de 64 bits. En écriture, le processeur écrit simultanément dans les 4 *fifos* (4x16 bits). En lecture, le processeur lit simultanément les données des 4 *fifos* (4x16 bits).

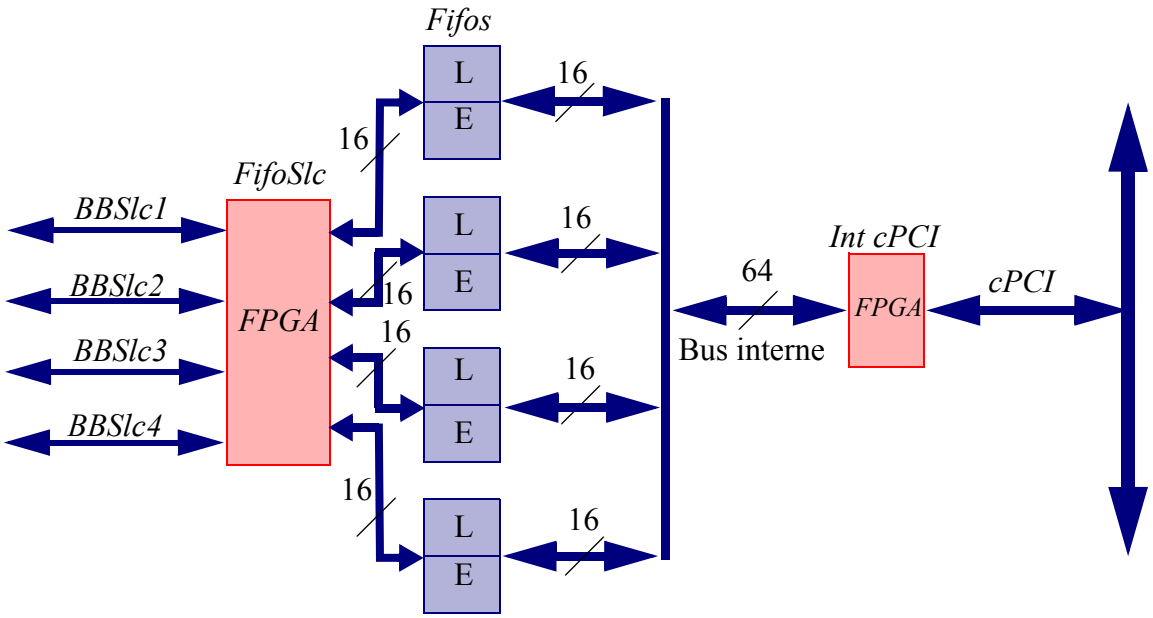


Figure 1.1: Les chemins de données de l'interface fifo cPCI

1. Interface compactPCI (cPCI)

2. Gestion des *BoxBus-Slc*

La gestion des *BoxBus Slc* est basée sur un *fpga Altera EP1C12Q240C6* noté par la suite *fpga FifoSlc*.

Le circuit reconnaît les ordres suivants :

2.1. Les ordres en lecture

Les ordres de lecture sont générés par des lectures *cPCI* dans l'un des 2 espaces, *BAR0* et *BAR1*[1]. L'espace *BAR0* est uniquement utilisé pour des lectures en mode *DMA*. L'espace *BAR1* est affecté à la lecture d'un mot de contrôle spécifique.

Ordre	Espace	Adresse (UserAdr[3:0])[a]	Action
RdStatus	*Bar1*	000	Lecture états *BoxBus* et *Fifos (12 bits)*
RdWC1	*Bar1*	001	Lecture Word Count *Fifo1 (32 bits)*
RdWC2	*Bar1*	010	Lecture Word Count *Fifo2 (32 bits)*
RdWC3	*Bar1*	011	Lecture Word Count *Fifo3 (32 bits)*
RdWC4	*Bar1*	100	Lecture Word Count *Fifo4 (32 bits)*
Version	*Bar1*	111	Lecture du numéro de version *(32 bits)*
RdFifos[b]	*Bar0*		Lecture des *fifos*

Tableau 2.1 : Les ordres en lecture

a. Cette adresse correspond à PciAd[6:3] pour l'interface Cpci en mode User=64 bits et à PciAd[5:2] pour l'interface Cpci en mode User=32 bits

b. Cet ordre est géré directement par les fifos

1. voir la note INTERFACE COMPACTPCI pour l'utilisation des registres BAR

2.1.1. Le mot d'états

Le mot d'états de 12 bits permet de connaître l'état de chaque *BoxBus Slc* et de chaque *fifo* (tableau 2.2).

Bit Status	Signal	Interprétation
0	*SlcBusyb1*	*BoxBus Slc*1 occupé si 0
1	*SlcBusyb2*	*BoxBus Slc* 2 occupé si 0
2	*SlcBusyb3*	*BoxBus Slc* 3 occupé si 0
3	*SlcBusyb4*	*BoxBus Slc*4 occupé si 0
4	*EmptyFifoWrb1*	*Fifo* d'écriture 1 vide si 0
5	*EmptyFifoWrb2*	*Fifo* d'écriture 2 vide si 0
6	*EmptyFifoWrb3*	*Fifo* d'écriture 3 vide si 0
7	*EmptyFifoWrb4*	*Fifo* d'écriture 4 vide si 0
8	*EmptyFifoRdb1*	*Fifo* de lecture 1 vide si 0
9	*EmptyFifoRdb2*	*Fifo* de lecture 2 vide si 0
10	*EmptyFifoRdb3*	*Fifo* de lecture 3 vide si 0
11	*EmptyFifoRdb4*	*Fifo* de lecture 4 vide si 0

Tableau 2.2 : Le mot d'états

2.1.2. Les Compteurs

La lecture des *BoxBus* est un processus automatique et le nombre de mots enregistrés peut varier d'une *fifo* à l'autre. Afin que le processeur puisse lire correctement ces données en mode *DMA*, il est nécessaire de connaître, a priori, le nombre de mots contenu dans chaque *fifo*.

Un compteur de 31 bits est associé à chaque *fifo* de lecture (*WC1*,*WC2*,*WC3*,*WC4*). L'utilisateur accède à ces 4 compteurs par les ordres de lecture : *RDWC1*, *RDWC2*, *RDWC3* et *RDWC4*.

Chaque compteur est incrémenté quand la *fifo* correspondante est écrite et décrémenté à chaque mot lu par le processeur.

Les compteurs sont remis à zéro, en même temps que les *fifos* de lecture, par l'ordre *RstRdFifos* (tableau 2.3).

La figure 2.1 montre les principaux signaux impliqués dans la gestion des compteurs. Ce mécanisme est également décrit dans le paragraphe §6. page 118.

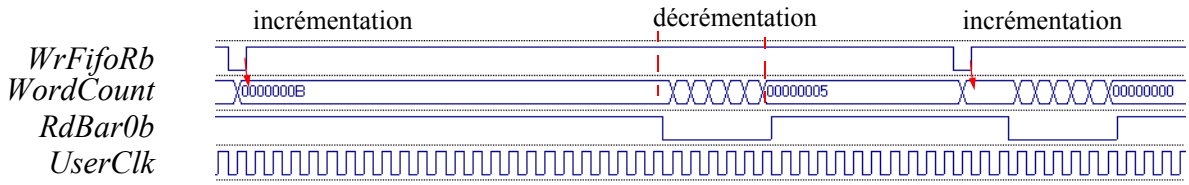


Figure 2.1 : Principe de gestion des compteurs

2.1.3. Numéro de version

Le numéro de version est accessible dans le format suivant :
[yyddmmxx] sur 32 bits.
Par exemple 08040701 correspond à la version 01 du 04/07/08.

2.2. Gestion du taux de remplissage des fifos

Si les *fifos* sont pleines, toutes les données qui vont arrivées seront perdues. De façon à gérer au mieux le remplissage des *fifos* et d'assurer que les fifos ne contiennent que des événements intègres, un mécanisme de blocage et de déblocage des données tiroirs est mis en place.

2 seuils sont programmables dans le *fpga*, *FifoUpperLimit* et *FifoLowerLimit*.

FifoUpperLimit définit le seuil à partir duquel il faut arrêter le remplissage des *fifos*. Quand le seuil est franchi, l'événement en cours continue d'être entré dans les *fifos* avant l'arrêt du remplissage.

Le remplissage des *fifos* reprendra quand le nombre de mots dans les *fifos* sera inférieur à *FifoLowerLimit.*

A l'initialisation, *FifoUpperLimit="3000" et FifoLowerLimit="0FFF"*. L'ordre *RstProg (RAZ du fpga)* ré-initialise ces valeurs.

2.3. Les ordres en écriture

Les ordres d'écriture sont générés par des écritures *cPCI* dans l'un des 2 espaces, *BAR0* et *BAR1*[1]. L'espace *BAR0* est uniquement utilisé pour des écritures en mode *DMA*. L'espace *BAR1* est utilisé pour l'écriture d'un mot de contrôle spécifique.

1. voir la note INTERFACE COMPACTPCI pour l'utilisation des registres BAR

Ordre	Espace	Adresse (UserAdr[3:0][a])	Action
SendFifo(1-4)	*Bar1*	0001	Transfert de(s) *fifo*(s) sur *BoxBus Slc*
Slc Enable(1-4)	*Bar1*	0010	Validation requête *Slc par bus*
Init(1-4)	*Bar1*	0100	Génère un *Initb* sur le(s) *BoxBus Slc*
RstProg	*Bar1*	1000	RAZ du *fpga*
RstWrFifos	*Bar1*	1010	RAZ des *fifos* d'écriture
RstRdFifos	*Bar1*	1100	RAZ des *fifos* de lecture
WrFifos[b]	*Bar0*		Ecriture des *fifos*

Tableau 2.3 : Les ordres en écriture

a. Cette adresse correspond à PciAd[6:3] pour l'interface Cpci en mode User=64 bits et à PciAd[5:2] pour l'interface Cpci en mode User=32 bits

b. Cet ordre est géré directement par les fifos

2.3.1. Transfert des fifos sur les *BoxBus Slc*

Après avoir charger les *fifos* d'écriture, l'utilisateur doit demander, à l'interface, leur vidage sur le *BoxBus*.

L'utilisateur écrit un mot de 4 bits (*UserData*[3:0]) à l'adresse 0001H dans l'espace défini par *BAR1* pour vider une ou plusieurs *fifos*.

Si *UserData*[*i*]=1 alors la *fifo i* est vidée sur le *BoxBus Slci*. Avec : $1 \le i \le 4$.

2.3.2. Validation du système de requête Slc

La commande *SlcEn* (UserAdr[3:0]="0010") permet de valider ou d'interdire le système de requête des bus *slc*. L'utilisateur écrit un mot de 4 bits (*UserData*[3:0]) à l'adresse 0010H dans l'espace défini par *BAR1*.

Si *UserData*[*i*]=1 alors le *bus Slc i* est validé. Avec : $1 \le i \le 4$.

Si *UserData*[*i*]=0 alors le *bus Slc i* n'est pas validé. Avec : $1 \le i \le 4$

Avant de transférer son message *slc*, le tiroir concerné envoie une requête (*SlcReqb*). Une autorisation d'émettre (*SlcJO*) est retournée par le *fpga SlcFifo*.

Quand le *BoxBus* est libre, le système de priorité du *fpga SlcFifo* examine les deux sources de requêtes possibles et en autorise une seule :

• *Maître* (le maître désire vider une *fifo* sur le *BoxBus slc*)

• *Slc*

Quand les deux requêtes sont en concurrence, la priorité est donnée à la requête *Maître.*

2.3.3. Génération de *Initb*

L'utilisateur écrit un mot de 4 bits (*UserData*[3:0]) à l'adresse 0100H dans l'espace défini par *BAR1* pour générer un *Initb* sur un ou plusieurs *BoxBus*.

Si *UserData*[*i*]=1 alors *Initbi* est généré sur le *BoxBus i*. Avec : $1 \leq i \leq 4$.
La durée du signal *Initbi* est de 240 *ns* (8**TUserClk*).

2.3.4. RAZ des *Fifos*

L'utilisateur écrit à l'adresse 1010H dans l'espace défini par *BAR1* pour remettre à zéro less *Fifos* d'écriture.

Les *fifos* de lecture sont remises à zéro globalement par l'ordre *RstRdFifos*.

3. L'interface *Fifos*, *fpga cPCI* et *fpga FifoSlc*

Le f*pga cPCI* est bien adapté pour le contrôle de *fifos* synchrones. La figure 3.1 montre les signaux à connecter pour le contrôle des chemins de données. Le tableau 3.1 donne la liste des différents signaux de contrôle ainsi que leurs actions.

Signal	du fpga	vers	Action
RdBar0b	*cPCI*	- *Fifos* lecture - *fpga FifoSlc*	Validation de l'horloge de lecture (REnb)[a] Gestion du compteur WCi
WrBar0b	*cPCI*	*Fifos* écriture	Validation de l'horloge d'écriture (WEnb)
OeBar0b	*cPCI*	*Fifos* lecture	Validation du buffer 3 états de sortie (OEb)
RdBar1b	*cPCI*	*fpga FifoSlc*	Lecture mot d'états[b]
WrBar1b	*cPCI*	*fpga FifoSlc*	Génération d'ordres[c]
RdFifoWib	*FifoSlc*	*Fifos* écriture	Lecture d'un mot de la *fifo* d'écriture
WrFifoRib	*FifoSlc*	*Fifos* lecture	Ecriture d'un mot dans la *fifo* de lecture
UserClk	*cPCI*	*fpga FifoSlc*	Horloge de synchronisation générale dérivée de l'horloge *cPCI*
ClkOut	*FifoSlc*	- *Fifos* lecture - *Fifos* écriture	Horloge de synchronisation générale dérivée de l'horloge *cPCI* et resynthétisé.
BBDir= SlcMasterOn	*FifoSlc*	- *Buffers* bidir. *LVDS* : 1(A->B), 0(B->A)	Contrôle de la direction du *buffer* bidir.

Tableau 3.1 : Les signaux de contrôle

a. Signal pour une *Fifo* type CY7C4285V
b. voir le tableau 2.1
c. voir le tableau 2.3

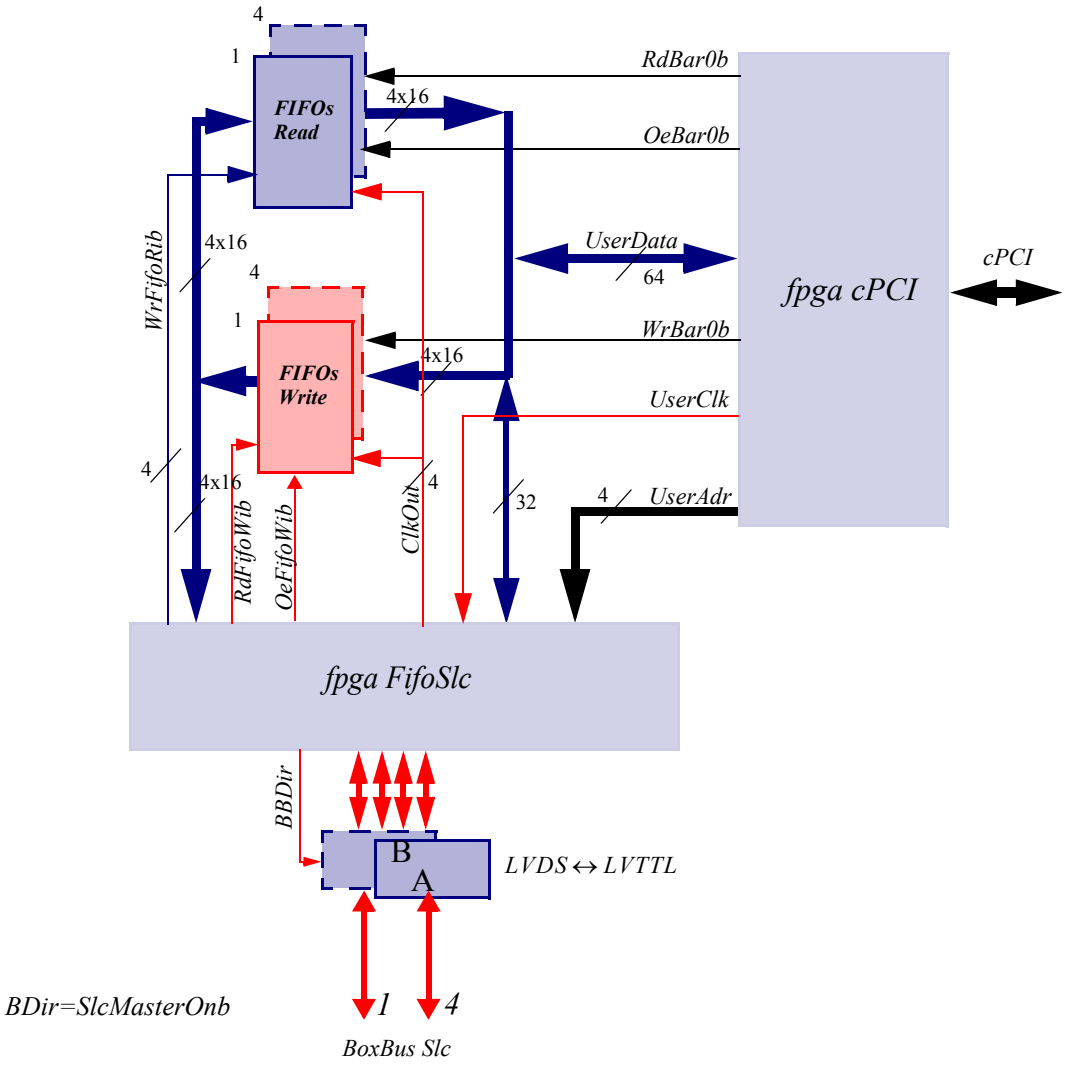


Figure 3.1 : Le contrôle des chemins de données

La figure 3.2 montre les différents signaux à connecter sur des *fifos* synchrones.

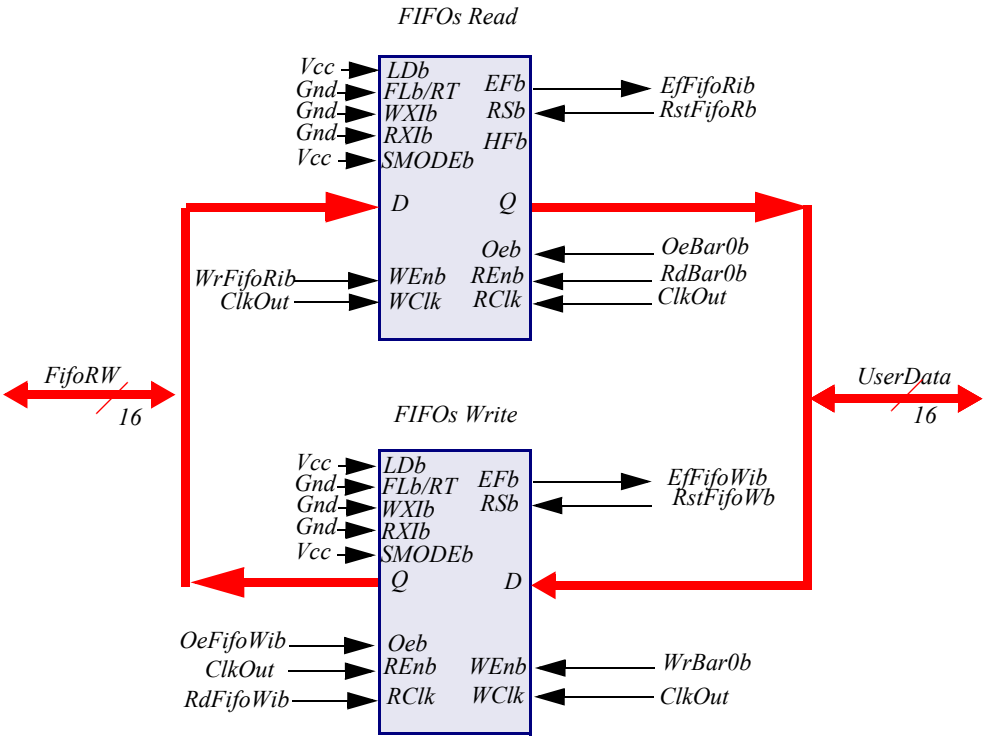


Figure 3.2 : Détails des signaux pour un couple de fifos

4. L'interface vers le *BoxBus Slc*

Les signaux sur le *BoxBus Slc* sont rappelés dans le tableau 4.1.

Signal	***Direction***	***Logique***	***Actif (H ou B)***	
SlcData	*Bidir*	*LVDS* 3-états	H	Donnée
SlcClk	*Bidir*	*LVDS* 3-états	-	Horloge
SlcRequestb_b	*E->M*[a]	LVTTL od[b]	B	Requête de bus *Slc*
SlcMasterOnb	M->E[c]	*LVDS* 3-états	B	Bus Maître Busy
SlcJO	M->E	*LVDS* 3-états	H	Jeton
SlcBusyb_b	*E->M*	*LVDS* 3-états	B	Bus Busy
Initb_b	*M->E*	*LVTTL*	B	Reset

Tableau 4.1 : Les signaux du Box Bus

a. Tiroir vers carte fifo
b. Drain ouvert
c. Carte fifo vers tiroir

5. Les signaux sur le *BoxBus Slc*

Les figures suivantes montrent les chronogrammes des signaux du *BoxBus Slc*.

Le dispositif qui génère les données génère l'horloge.

Les données sont positionnées, par l'émetteur, avec le front montant de l'horloge et échantillonnées par le(s) récepteur(s) avec le front arrière de la même horloge.

Quand un tiroir désire émettre (figure 5.1), il positionne la ligne *SlcReqb*='0'. Si la requête est acceptée, la carte *Fifo cPCI* émet un jeton (*SlcJo*). A la réception du jeton la carte *slow control* prend le contrôle du bus en positionnant *SlcBusyb*='0'. Les données sérialisées sont émises sur la ligne *SlcData* ([1] en premier) et l'horloge sur la ligne *SlcCLk*.

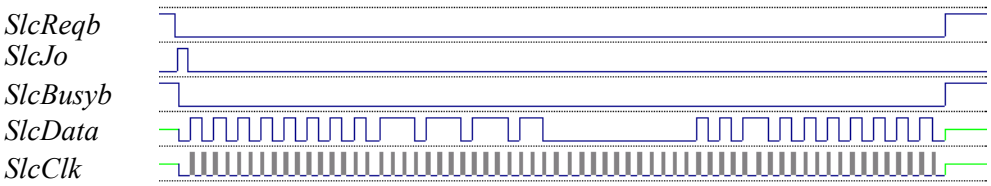


Figure 5.1 : Trame Tiroir vers Fifo (vue globale)

Le signal d'horloge est actif 75 ns après l'activation de *SlcBusyb* (figure 5.2). Sa période est de 75 ns soit un débit d'environ 13 *MBits*/s.

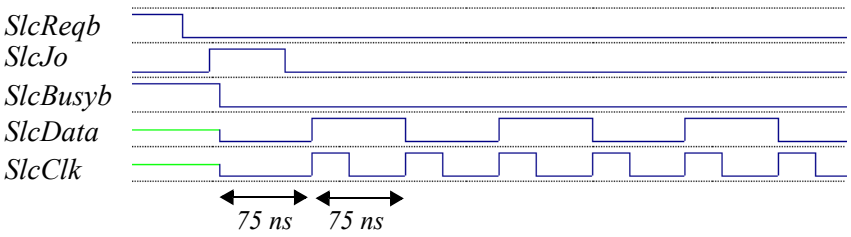


Figure 5.2 : Trame Tiroir vers Fifo (détails 1)

Figure 5.3 : Trame Tiroir vers Fifo (détails 2)

Dans le cas d'une émission vers le(s) tiroir(s), la carte *Fifo* s'assure qu'aucune transaction est en cours puis prend possession du bus en positionnant la ligne *SlcMasterOnb*='0' (figure 5.4). Les données sont ensuite envoyées en série sur la ligne *SlcData* et l'horloge sur la ligne *SlcClk*.

Le signal d'horloge est actif 60 ns après l'activation de *SlcBusyb* (figure 5.5). Sa période est de 90 ns soit un débit d'environ 11 *MBits*/s.

La différence de débit entre les deux modes est liée d'une part à la volonté de ne pas

1. Most Significant Bit (bit de poids fort)

avoir des débits trop rapide et d'autre part aux horloges de base des systèmes (66MHz et 33 MHz respectivement dans les tiroirs et dans la carte *Fifo*).

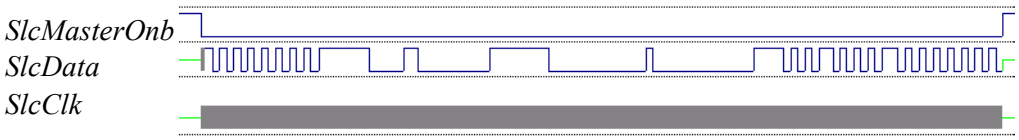


Figure 5.4 : Trame Fifo vers Tiroir (vue globale)

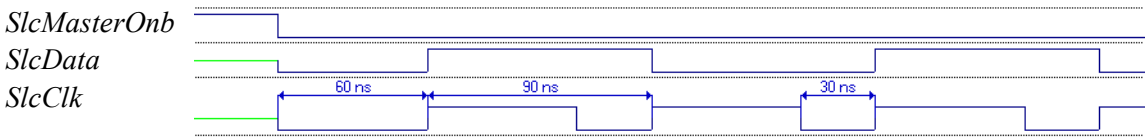


Figure 5.5 : Trame Fifo vers Tiroir (détails)

La figure 5.6 montre un message *CntrlSlc* (X"FE0C") envoyé par le processeur. L'adresse X"FE0C" indique qu'un accusé de réception *SlcRdy* (X"EEEC") est demandé au tiroir. Quand la trame est reçue par le tiroir, une requête d'accès au bus est émise (*SlcReqb*) puis la trame SlcRdy est envoyée lorsque le jeton (*SlcJO*)est reçu. *WrFifoRb* est le signal d'écriture de chaque mot du message (sérialisé) dans la *fifo*.

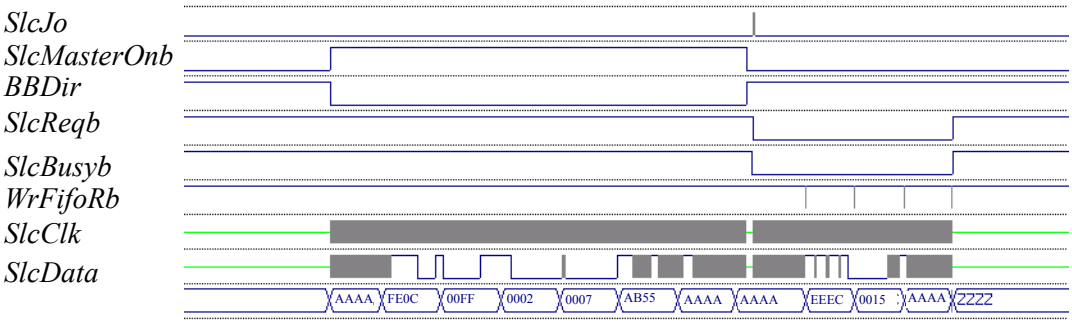


Figure 5.6 : Enchaînement d'une trame maître et d'une trame esclave

6. Exemple de cycles de lecture

Les deux figures suivantes montrent la synchronisation de l'écriture *Fifo*, par des événements échelles, puis une relecture *cPCI*.

Chaque mot de 16 bits désérialisé est écrit dans la Fifo correspondante par le signal *WrFifoRb*.

D'autre part le *Word Count* associé à la *Fifo* est incrémenté.

D'une manière totalement asynchrone, le programme de test *cPCI* boucle sur la lecture du *Word Count* (*RdBAR1b*) et attend une valeur supérieure à 10 (choix de test uniquement). Sur la figure 6.1, le premier test a lieu pour *WC=09H* et le second quand *WC=0CH*.

Pour *WC=0CH*, le programme de test déclenche 2 lectures *DMA*, successives, de 5 mots (*RdBAR0b*). A chaque mot lu, le *Word Count* est décrémenté. La figure 6.2 montre

le détail de l'opération. Lors du premier *RdBAR0b* le *Word Count* est décrémenté de la valeur *0CH* à la valeur *07H*. On peut remarquer que lorsque le *WC=06H* celui-ci reçoit simultanément l'ordre d'incrémentation (*WrFifoRb*) et de décrémentation. Le *Word Count* reste donc à la valeur *06H*.

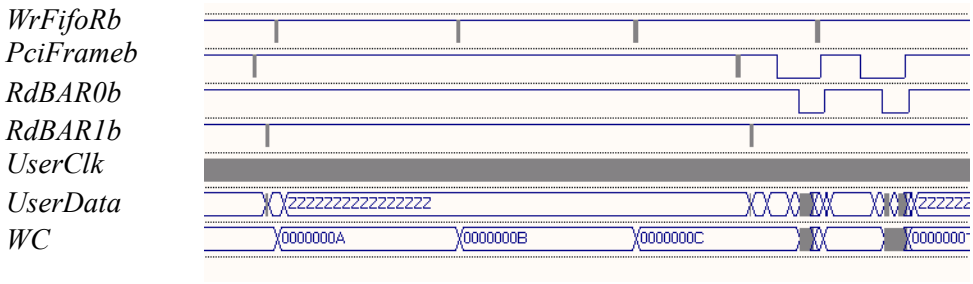


Figure 6.1 : Cycles de lecture vue générale

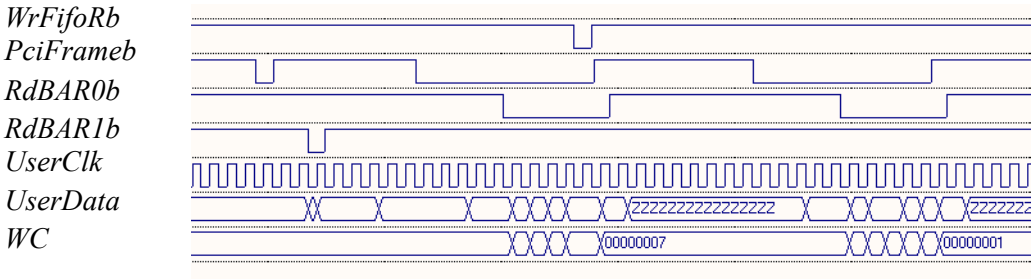


Figure 6.2 : Cycles de lecture (détails)

7. Programmation des *Fpga*

La carte *Fifo cPCI* comporte deux *fpga* qui sont programmés selon la norme *JTAG* d'une part et en utilisant, d'autre part, le mode *Flash Loader* (Voir " § 3.2. page 26 "). La figure 7.1 rappelle le principe de connexion des *Fpga*, des *EPROM* et du connecteur *JTAG*.

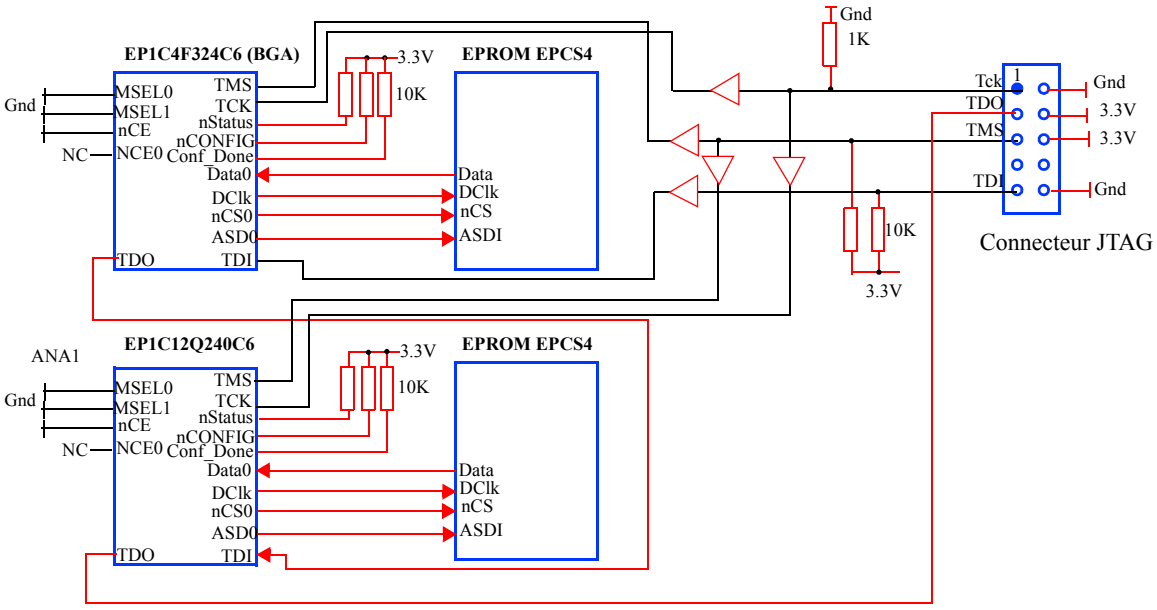


Figure 7.1 : Chaînage FPGA en mode FlashLoader

8. Codage du mot *DEVICE_ID fpga cPCI*

Le *FPGA cPCI* possède un espace de configuration dont le contenu du mot *Device_ID* est utilisé par le système pour déterminer le driver logiciel à utiliser.

Pour la carte *FifocPCI SLC* : *Device_ID=X"2AAA"*.

CARTE FIFO CPCI DATA

1. Introduction

La carte *fifo cPCI Data* est basée d'une part, sur l'utilisation du bus *compactPCI* et d'autre part sur l'utilisation du *BoxBus Data*.

L'interface avec le bus *compactPCI* est effectué par un *fpga Altera BGA EP1C4F324C6* de la série *Cyclone*. La fonctionnalité de ce circuit est complètement décrite dans une note[1]. Le *BoxBus Data* est un bus parallèle de 16 bits, basé sur une structure de messages, élaboré pour l'expérience *HESS*.

La partie *BoxBus Data* a été décrite dans les notes précédentes.

Un système de *fifos* est utilisé, en lecture, pour transférer des messages vers le *cPCI*.

La carte peut gérer 4 *BoxBus Data*. Les chemins de données de l'interface *fifo cPCI Data* sont représentés par la figure 1.1.

Le contrôle de l'interface avec les *BoxBus Data* est effectué par un *fpga Altera EP1C12Q240C6*. Afin de limiter les temps de transfert avec le processeur, les données sont transférées entre les *fifos* et le *cPCI* sur une largeur de 64 bits. En lecture, le processeur lit simultanément les données des 4 *fifos* (4x16 bits).

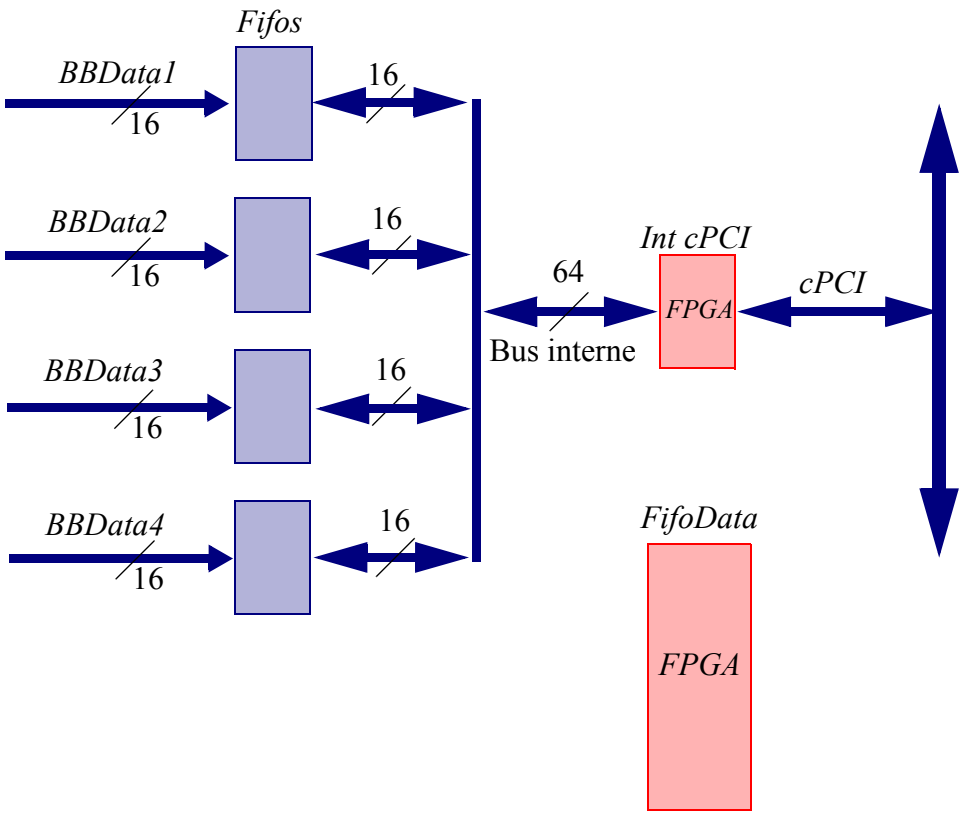


Figure 1.1: Les chemins de données de l'interface fifo cPCI

1. Interface compactPCI (cPCI)

2. Gestion des *BoxBus-Data*

La gestion des *BoxBus Data* est basée sur un *fpga Altera EP1C12Q240C6* noté par la suite *fpga DataFifo*.

Le circuit reconnaît les ordres suivants :

2.1. Les ordres en lecture

Les ordres de lecture sont générés par des lectures *cPCI* dans l'un des 2 espaces, *BAR0* et *BAR1*[1]. L'espace *BAR0* est uniquement utilisé pour des lectures en mode *DMA*. L'espace *BAR1* est affecté à la lecture d'un mot de contrôle spécifique.

Ordre	Espace	Adresse (UserAdr[3:0])[a]	Action
RdStatus	*Bar1*	000	Lecture états *BoxBus* et *Fifos (12 bits)*
RdWC1	*Bar1*	001	Lecture Word Count *Fifo1 (32 bits)*
RdWC2	*Bar1*	010	Lecture Word Count *Fifo2 (32 bits)*
RdWC3	*Bar1*	011	Lecture Word Count *Fifo3 (32 bits)*
RdWC4	*Bar1*	100	Lecture Word Count *Fifo4 (32 bits)*
Version	*Bar1*	111	Lecture du numéro de version *(32 bits)*
RdFifos[b]	*Bar0*		Lecture des *fifos*

Tableau 2.1 : Les ordres en lecture

a. Cette adresse correspond à PciAd[6:3] pour l'interface Cpci en mode User=64 bits et à PciAd[5:2] pour l'interface Cpci en mode User=32 bits

b. Cet ordre est géré directement par les fifos

1. voir la note INTERFACE COMPACTPCI pour l'utilisation des registres BAR

2.1.1. Le mot d'états

Le mot d'états de 12 bits permet de connaître l'état de chaque *BoxBus Data* et de chaque *fifo* (tableau 2.2).

Bit Status	Signal	Interprétation
0	*BBBusyb1*	*BoxBus Data*1 occupé si 0
1	*BBBusyb2*	*BoxBus Data* 2 occupé si 0
2	*BBBusyb3*	*BoxBus Data* 3 occupé si 0
3	*BBBusyb4*	*BoxBus Data* 4 occupé si 0
8	*EmptyFifoRdb1*	*Fifo* de lecture 1 vide si 0
9	*EmptyFifoRdb2*	*Fifo* de lecture 2 vide si 0
10	*EmptyFifoRdb3*	*Fifo* de lecture 3 vide si 0
11	*EmptyFifoRdb4*	*Fifo* de lecture 4 vide si 0

Tableau 2.2 : Le mot d'états

2.1.2. Les Compteurs

La lecture des *BoxBus* est un processus automatique et le nombre de mots enregistrés peut varier d'une *fifo* à l'autre. Afin que le processeur puisse lire correctement ces données en mode *DMA*, il est nécessaire de connaître, a priori, le nombre de mots contenu dans chaque *fifo*.

La gestion des compteurs est délicate du fait que l'horloge d'écriture n'est pas du tout synchrone de l'horloge de lecture. Il est nécessaire d'éviter une lecture et une écriture simultanée.

Pour cela, un compteur de 31 bits *WCR* est incrémenté à chaque lecture, un compteur de 31 bits *WCWi* (un compteur par *Fifo*) est incrémenté à chaque écriture.

Un registre de 31 bits est associé à chaque *fifo* (*WC1*,*WC2*,*WC3*,*WC4*) : $WCi = WCWi - WCR$. L'actualisation a lieu lorsqu'il n'y a pas de transfert *BoxBus* (entre 2 transferts carte par exemple).

Les compteurs *WCWi* sont remis à zéro par l'ordre *RstWcFifos* (tableau 2.3).

La figure 2.1 montre les principaux signaux impliqués dans la gestion des compteurs.

Les compteurs ont pour valeurs initiales : $WCR = 36$ et $WCW = 40$, le registre $WC1 = 4$.

Deux cartes d'un tiroir prennent part à l'échange.

A chaque donnée transmise, le compteur d'écriture *WCW* est incrémenté. A la fin du premier échange (20 mots) celui-ci vaut 60 puis 80 en fin de deuxième échange. Le registre est finalement actualisé qu'au instants ou le *BoxBus* est libre. A la première actualisation, $WC1 = 60 - 36 = 24$ puis $WC1 = 80 - 36 = 44$.

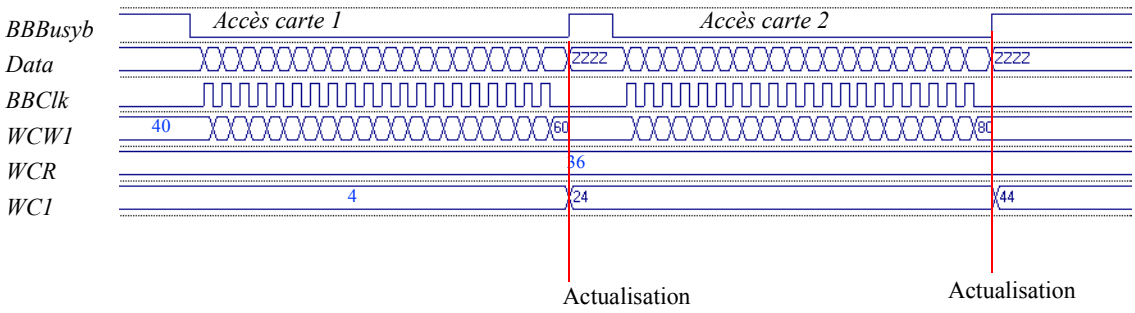


Figure 2.1 : Principe de gestion des compteurs (1)

La figure 2.2 montre la modification des compteurs en cas de lecture par le processeur. Le processeur lance 2 rafales de 5+1 mots, activent lors du signal *RdBAR0b*. Le registre utilisateur est toujours actualisé lorsque *BBBusyb* est inactif.

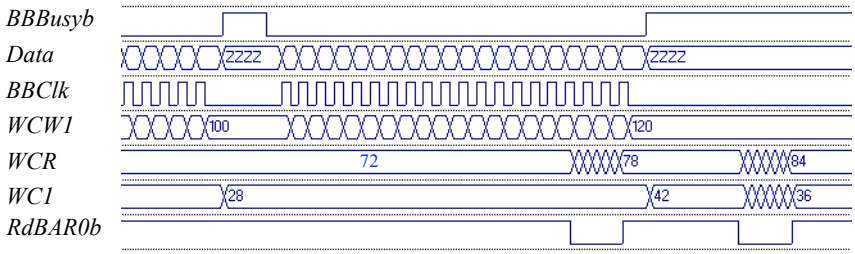


Figure 2.2 : Principe de gestion des compteurs (2)

Remarque : Le système de gestion des compteurs tient compte de tous les mots qui sont effectivement sortis de la fifo. Dans le cas de rafales, il y a toujours un mot (le dernier de la rafale) qui est sorti de la *fifo* mais pas lu par le processeur.

2.1.3. Numéro de version

Le numéro de version est accessible dans le format suivant :
[yyddmmxx] sur 32 bits.
Par exemple 08040701 correspond à la version 01 du 04/07/08.

2.1.4. Gestion du taux de remplissage des *fifos*

Si les *fifos* sont pleines, toutes les données qui vont arrivées seront perdues. De façon à gérer au mieux le remplissage des *fifos* et d'assurer que les fifos ne contiennent que des événements intègres, un mécanisme de blocage et de déblocage des données tiroirs est mis en place.

2 seuils sont programmables dans le *fpga*, *FifoUpperLimit* et *FifoLowerLimit*.

FifoUpperLimit définit le seuil à partir duquel il faut arrêter le remplissage des *fifos*. Quand le seuil est franchi, l'événement en cours continue d'être entré dans les *fifos* avant l'arrêt du remplissage.

Le remplissage des *fifos* reprendra quand le nombre de mots dans les *fifos* sera inférieur à *FifoLowerLimit.*

A l'initialisation, *FifoUpperLimit="3000" et FifoLowerLimit="0FFF"*. L'ordre *RstProg (RAZ du fpga)* ré-initialise ces valeurs.

2.2. Les ordres en écriture

Les ordres d'écriture sont générés par des écritures *cPCI* dans l'un des 2 espaces, *BAR0* et *BAR1*[1]. L'espace *BAR0* est uniquement utilisé pour des écritures en mode *DMA*. L'espace *BAR1* est utilisé pour l'écriture d'un mot de contrôle spécifique.

Ordre	Espace	Adresse (UserAdr[3:0][a])	Action
Data Enable(1-4)	*Bar1*	0010	Validation requête *Data* par bus
FifoUpperLimit	*Bar1*	0110	Stop remplissage *Fifo (16 bits)*
FifoLowerLimit	*Bar1*	0111	Redémarrage remplissage *Fifo (16 bits)*
RstProg	*Bar1*	1000	RAZ du *fpga*[b]
RstWcFifos	*Bar1*	1010	RAZ compteurs de mots des *fifos*
RstRdFifos	*Bar1*	1100	RAZ des *fifos* de lecture[c]

Tableau 2.3 : Les ordres en écriture

a. Cette adresse correspond à PciAd[6:3] pour l'interface Cpci en mode User=64 bits et à PciAd[5:2] pour l'interface Cpci en mode User=32 bits
b. Les machines d'états et les limites Fifo sont réinitialisées, les compteurs restent inchangés
c. Les compteurs associés à chaque Fifo sont réinitialisés

2.2.1. Validation du système de requête Data

La commande *DataEn* (UserAdr[3:0]="0010") permet de valider ou d'interdire le système de requête des bus *Data*. L'utilisateur écrit un mot de 4 bits (*UserData*[3:0]) à l'adresse 0010H dans l'espace défini par *BAR1*.

Si *UserData*[*i*]=1 alors le *bus Data i* est validé. Avec : $1 \le i \le 4$.

Si *UserData*[*i*]=0 alors le *bus Data i* n'est pas validé. Avec : $1 \le i \le 4$

Avant de transférer son message *Data*, le tiroir concerné envoie une requête (*BBReqbusb*). Une autorisation d'émettre (*BBJO*) est retournée par le *fpga DataFifo* quand le *BoxBus* est libre et validé.

2.2.2. RAZ des *Fifos*

L'utilisateur écrit à l'adresse 1100H dans l'espace défini par *BAR1* pour remettre à zéro les *Fifos* de lecture.

Les *fifos* de lecture sont remises à zéro globalement par l'ordre *RstRdFifos*.

1. voir la note INTERFACE COMPACTPCI pour l'utilisation des registres BAR

3. L'interface *Fifos*, *fpga cPCI* et *fpga DataFifo*

Le f*pga cPCI* est bien adapté pour le contrôle de *fifos* synchrones. La figure 3.1 montre les signaux à connecter pour le contrôle des chemins de données. Le tableau 3.1 donne la liste des différents signaux de contrôle ainsi que leurs actions.

Signal	du fpga	vers	Action
RdBar0b	*cPCI*	- *Fifos* lecture - *fpga FifoData*	Validation de l'horloge de lecture (REnb)[a] Gestion du compteur WCi
WrBar0b	*cPCI*	*Fifos* écriture	Validation de l'horloge d'écriture (WEnb)
OeBar0b	*cPCI*	*Fifos* lecture	Validation du buffer 3 états de sortie (OEb)
RdBar1b	*cPCI*	*fpga FifoData*	Lecture mot d'états[b]
WrBar1b	*cPCI*	*fpga FifoData*	Génération d'ordres[c]
WrEnFifoRib	*FifoData*	*Fifos* lecture	Validation de l'écriture d'un mot dans la *fifo* de lecture
FifoWrClki	*FifoData*	*Fifos* lecture	Horloge d'écriture de la *fifo* de lecture
UserClk	*cPCI*	*fpga FifoData*	Horloge de synchronisation générale dérivée de l'horloge *cPCI*
ClkOut	*FifoData*	- *Fifos* lecture - *Fifos* écriture	Horloge de synchronisation générale dérivée de l'horloge *cPCI* et resynthétisé.

Tableau 3.1 : Les signaux de contrôle

a. Signal pour une *Fifo* type CY7C4285V
b. voir le tableau 2.1
c. voir le tableau 2.3

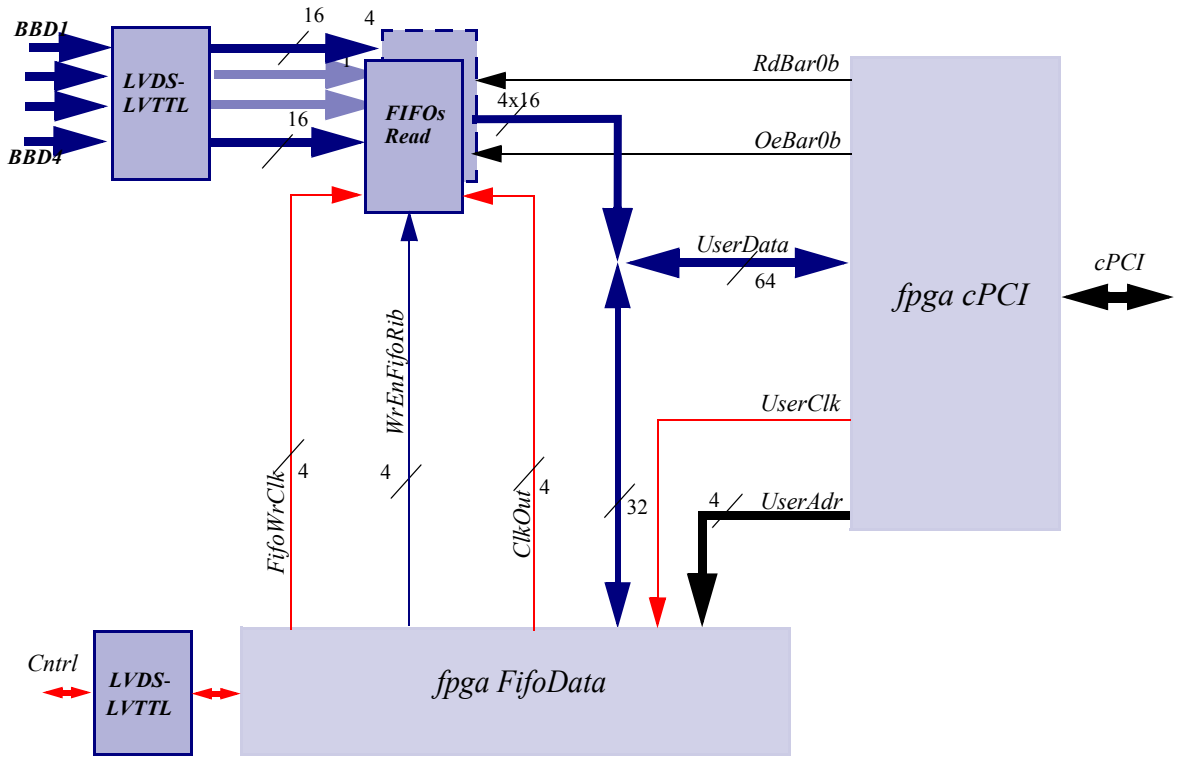


Figure 3.1 : Le contrôle des chemins de données

La figure 3.2 montre les différents signaux à connecter sur des *fifos* synchrones.

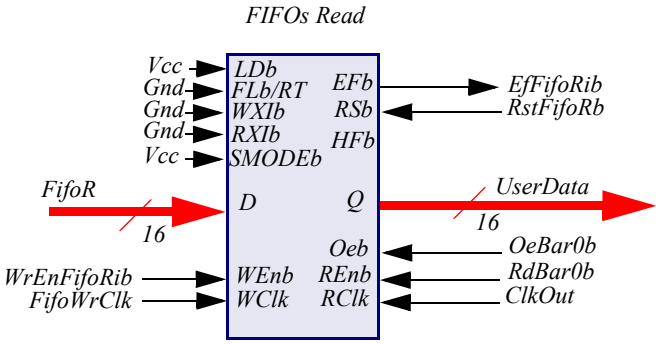


Figure 3.2 : Détails des signaux pour une fifo

4. L'interface vers le *BoxBus Data*

Les signaux sur le *BoxBus Data* sont rappelés dans le tableau 4.1.

Signal	***Direction***	***Logique***	***Actif (H ou B)***	
BBD	*E->M*[a]	*LVDS* 3-états	H	Donnée
BBDClkBus	*Bidir*	*LVDS* 3-états	-	Horloge

Tableau 4.1 : Les signaux du Box Bus

Signal	*Direction*	*Logique*	*Actif (H ou B)*	
BBReqBusb	*E->M*	LVTTL od[b]	B	Requête de bus *Data*
BBBusyBusb	M->E[c]	*LVDS* 3-états	B	Bus Busy
BBJO	M->E	*LVDS* 3-états	H	Jeton

Tableau 4.1 : Les signaux du Box Bus

a. Tiroir vers carte fifo
b. Drain ouvert
c. Carte fifo vers tiroir

5. Les signaux sur le *BoxBus Data*

Le dispositif qui génère les données génère l'horloge. La figure 5.1 montre le transfert de données d'un tiroir, constitué de 2 cartes analogiques. Consécutivement à un *trigger L2A* (non représenté sur la figure), Chacune des deux cartes du tiroir émet une requête de bus (le *ou* des deux requêtes est matérialisé par la ligne *BBReqBusb*). Si le *BoxBus* est libre (pas de transfert en cours), le signal *BBBusyb* est inactif, un jeton *BBJO* est émis par la carte *FifoData*. La première carte recevant le jeton et désirant émettre prend possession du bus en activant la ligne *BBBusyb*. La carte *FifoData* désactive l'émission du jeton. En fin de transfert, la première carte relache le bus (*BBBusyb* inactif). Comme la ligne *BBReqBusb* est toujours active (requête de la seconde carte), un nouveau jeton est généré et le transfert peut commencer. La ligne *BBReqBusb* devient inactive losque les deux cartes ont transféré leurs données.

Les données sont positionnées, par l'émetteur, avec le front montant de l'horloge et échantillonnées par le(s) récepteur(s) avec le front arrière de la même horloge.

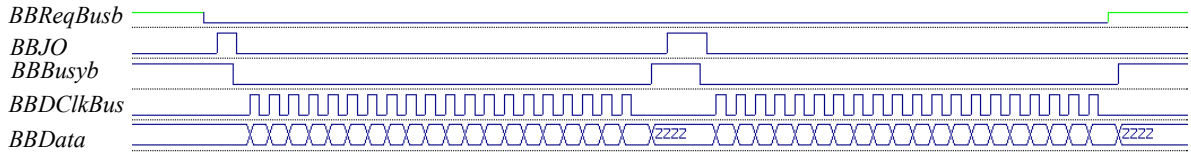


Figure 5.1 : Les signaux sur le BoxBus Data (principe général)

6. Programmation des *Fpga*

La carte *Fifo cPCI Data* comporte deux *fpga* qui sont programmés selon la norme *JTAG* d'une part et en utilisant, d'autre part, le mode *Flash Loader* (Voir " § 3.2. page 26 "). La figure 6.1 rappelle le principe de connexion des *Fpga*, des *EPROM* et du connecteur *JTAG*.

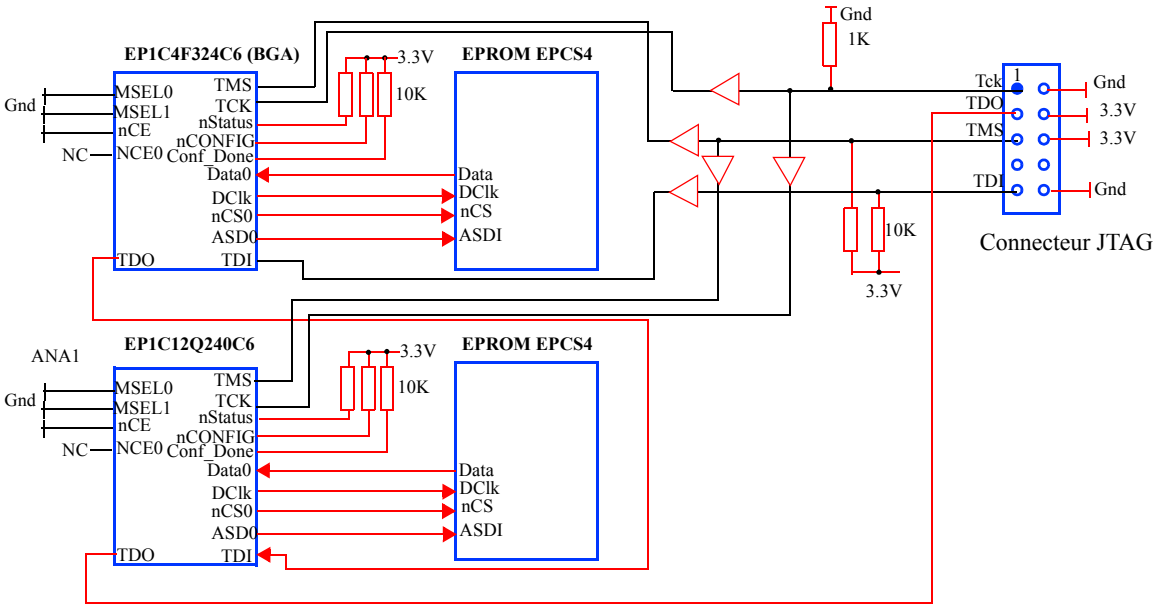


Figure 6.1 : Chaînage FPGA en mode FlashLoader

7. Codage du mot *DEVICE_ID fpga cPCI*

Le *FPGA cPCI* possède un espace de configuration dont le contenu du mot *Device_ID* est utilisé par le système pour déterminer le driver logiciel à utiliser.

Pour la carte *FifocPCI DATA* : *Device_ID=X"2BBB"*.

TIME STAMPING

1. Introduction

Le système de déclenchement de l'expérience HESS-II est constituée de deux niveaux successivement actifs (*L1* et *L2*). Le premier, analogique, très rapide (latence inférieure à 100 *ns*) est basé sur une sectorisation de la caméra. Un minimum d'énergie dans au moins un des secteurs permet de déclencher le niveau 1. Le premier niveau permet d'arrêter l'échantillonnage des 256 mémoires analogiques situées dans les tiroirs. Le second, basé sur des algorithmes pré-programmés, filtre les décisions de niveau 1 afin de limiter le flot de données vers l'acquisition centrale.

D'autre part, le signal de niveau 1 (*L1A*) est transmis au *LTM* (*Local Trigger Module*) interface entre la caméra et la logique de *Trigger* centrale (multi-télescopes).

En retour le *LTM* fournit à la caméra un certain nombre de données (Les données critiques : *Event Number, Bunch Number, Farm Node ID* et des signaux généraux de service).

Les données *DAQ* et *Trigger* de l'expérience HESS-II proviennent de 2 châssis *cPCI* différents. Il est donc fondamental de pouvoir synchroniser ces données.

Les cartes *cPCI Time Stamp* (maître et esclave) permettent d'une part, d'associer un même identificateur aux blocs de données *DAQ* et *Trigger* et d'autre part de distribuer aux châssis correspondants les données critiques lues dans le *LTM* par la carte *Time Stamp* maître.

2. Principe

La figure 2.1 montre le principe du *fpga Time Stamp* maître et la figure 2.2 donne le synoptique général d'interconnexion des différents modules. Les différences entre le maître et l'esclave sont que seul le maître :

- lit le *LTM*. L'esclave reçoit alors les données *LTM* du maître, par un lien série.
- génère l'horloge et la remise à zéro du compteur. L'esclave reçoit les signaux nécessaires *CntClk* et *BunchCntPrst*.

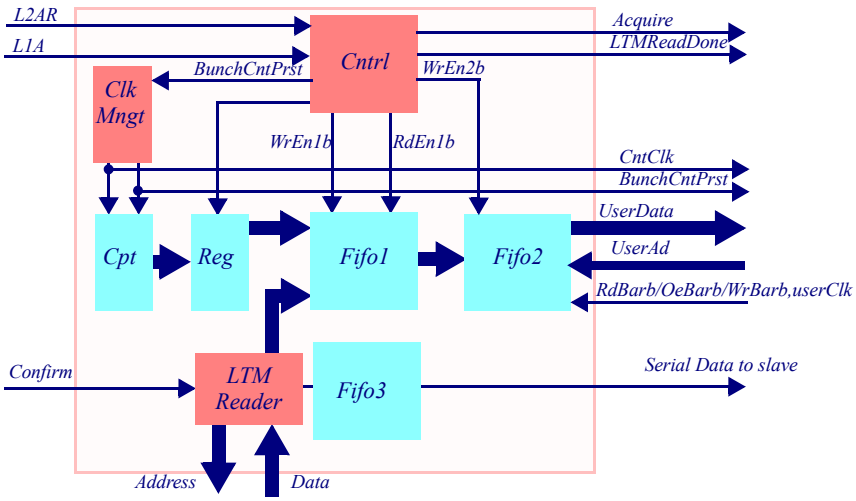


Figure 2.1 : Synoptique du fpga Time Stamp maître

Un compteur est incrémenté par une horloge locale à la carte maître et ré-initialisé régulièrement. Les compteurs des cartes maître et esclave sont synchronisés (les signaux de contrôle sont identiques). Quand le compteur passe par zéro un signal *Acquire* est généré dans le but de déclencher la lecture d'un *GPS*. Connaissant cette valeur *GPS* (heure précise) ainsi que la période de l'horloge des compteurs, il est alors facile de déduire la datation de l'événement en temps absolu.

Quand un signal de déclenchement *L1A* est généré, la valeur du compteur est temporairement stockée dans un registre. Environ 5µs, plus tard, le *LTM Confirm* la validité du déclenchement de premier niveau. Les données critiques peuvent être lues dans le *LTM*. Le registre et les données critiques sont ensuite écrites dans la *fifo1*. (Les données critiques sont également sérialisées et transmises à l'esclave).

En cas de *L2Accept* les données sont transférées de la *fifo1* vers la *fifo2*.

La *fifo2* peut ensuite être lue par le processeur.

En cas de *L2Reject* les données sont supprimées de la *fifo1*.

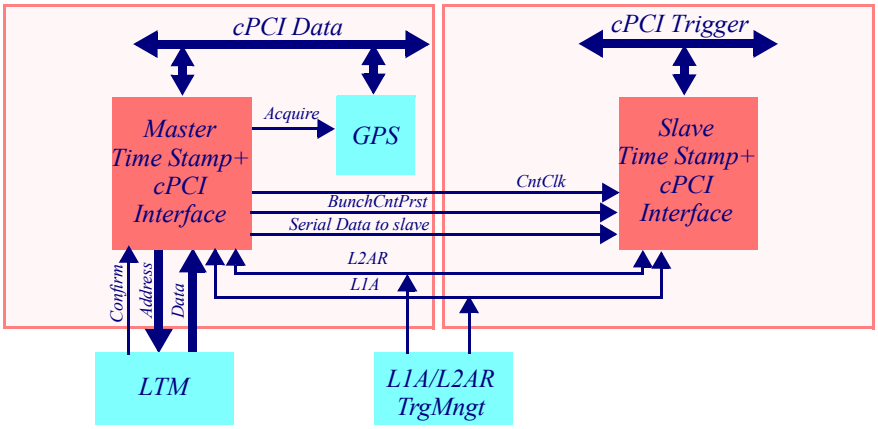


Figure 2.2 : Synoptique général

3. Le Local Trigger Module (*LTM*)

L'interface avec le *LTM* est composé de deux bus : un bus adresse de 8 bits et un bus données de 16 bits (voir figure 2.2.). Quand le signal *Confirm* est généré par le *LTM*, les données critiques sont actualisées et peuvent donc être lues (tableau 3.1). En fin de lecture, la carte *Time Stamp* génère le signal *LTMReadDone,* vers la carte *TrgMngt,* pour relacher les signaux *L1A* et *Busy*. Le chronogramme de lecture de 7 mots dans le *LTM* est représenté par la figure 3.1.

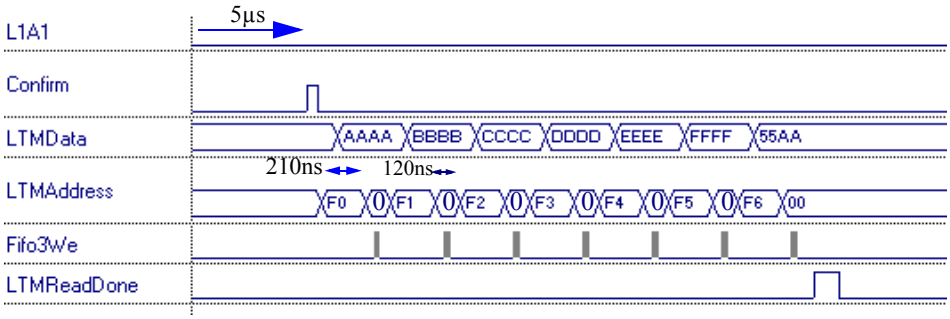


Figure 3.1 : Chronogramme de lecture du LTM par le Time Stamp maître

Lorsqu'un signal *L1* est envoyé au *LTM* celui valide ce déclenchement en générant le signal *Confirm* environ 5 μs après. Pour lire le *LTM*, il suffit de positionner une adresse de 8 bits et d'attendre environ 210 *ns* avant de récupérer la donnée sur 16 bits. L'adresse doit repasser par la valeur 0 pendant 120 *ns* avant de positionner une nouvelle adresse de lecture.

Nom	***Adresse***
Registre A	*Base*[1] *+0*
Registre B	*Base+1*
Bunch Number	*Base+2*
IP Address	*Base+3*
Status Register	*Base+4*
Event Counter Low	*Base+5*
Event Counter High	*Base+6*

Tableau 3.1 : Les données du LTM

1. Par défaut, Base = f0H

La figure 3.1 montre le chronogramme de lecture de 7 mots dans le *LTM*. Afin d'optimiser les temps de lecture il est possible de ne lire que les 4 données fondamentales : [Base+2], [Base+3], [Base+5] et [Base+6]. Cette possibilité est programmable dans l'interface.

4. Les principaux signaux

Les principaux signaux nécessaires pour les cartes *Time Stamp* sont regroupés dans le Tableau 4.1.

Nom	*De la Carte*	*Vers Carte*	*Commentaire*
L1A[1]	*TrgMngt*	*Master/Slave TS*	*Trigger de niveau 1* *Capture le compteur dans le registre*
L2AR[1]	*TrgMngt*	*Master/Slave TS*	*Trigger niveau 2 composite Accept/Reject* *Transfert de fifo1 vers fifo2 si Accept* *Annulation du bloc dans fifo1 si Reject* *TW=30 ns =L2R, 60 ns=L2A*
LTMReadDone[1]	*Master TS*	*TrgMngt*	*Spécifie que le LTM a été lu TW=100 ns*
Confirm[3]	*LTM*	*Master TS*	*LTM actualisé TW=45 ns* *Lancement lecture LTM*
ConfirmForSlv[3]	*Master TS*	*Slave TS*	*identique à Confirm*
LTMAddress[1]	*Master TS*	*LTM*	*TW=330 ns*
LTMData[1]	*Master TS*	*LTM*	
SDToSTS[2]	*Master TS*	*Slave TS*	*Donnée série pour le Time Stamp esclave*
CkSDToSTS[2]	*Master TS*	*Slave TS*	*Horloge de prise en compte de SDToSTS*
Acquire[3]	*Master TS*	*GPS*	*Déclenchement GPS TW=100 ns*
CntClk[2]	*Master TS*	*Slave TS*	*Horloge du compteur Maître/Esclave*
BunchCntPrst[2]	*Master TS*	*Slave TS*	*Preset du compteur Maître/Esclave*

Tableau 4.1 : Les principaux signaux

1. TTL
2. LVDS
3. TTL 50 ohms

5. Sérialisation des données vers le *Time Stamp* esclave

De façon à limiter les connexions entre les cartes *Time Stamp*, les données *LTM* sont sérialisées par le *Time Stamp* maître. Le chronogramme de sérialisation des données est représenté sur la figure 5.1.

Une donnée est lue dans le *LTM* toutes les 330 *ns*[1] et chargée dans une *fifo*. Cette donnée est ensuite sérialisée au rythme d'un symbole toutes les 30 *ns* (@33 *MHz*). Le temps nécessaire pour transmettre un mot de 16 bits est de 30x16=480 *ns*. Ce temps étant supérieur aux 330 *ns* nécessaires pour acquérir un mot du *LTM,* une *fifo* permet d'assurer la synchronisation.

1. Limitation liée au *LTM*

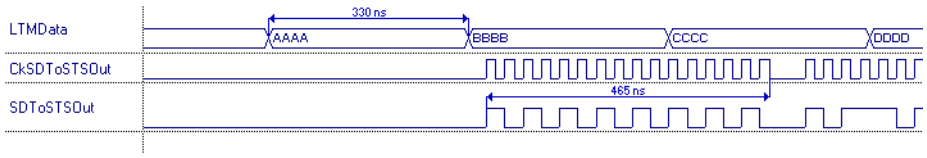


Figure 5.1 : Chronogramme de sérialisation par Time Stamp maître

6. Le Compteur synchrone

Le compteur décrit précédemment (paragraphe §2. page 130) permet de synchroniser la carte esclave avec la carte maître.

Ce compteur est incrémenté par une horloge à 33MHz (30ns de période). La capacité totale du compteur est de 32bits soit une variation du cycle possible entre 30ns et 128,85s. On définit la périodicité en programmant la valeur maximale du compteur. Par exemple, pour une périodicité de 1s il est nécessaire de programmer 25bits du compteur.

7. Programmation

Le tableau 7.1 décrit les différentes possibilités de programmation de la carte *Time Stamp*.

Ordre	***R/W***	***UserAD [3:0]***	***UserData***	***BAR 0/1***	***Commentaire***
Rst	*W*	*1H*		*BAR1*	*Reset programmé*
Cnt	*W*	*2H*	*[31:0]*	*BAR1*	*Compteur horloge* *défaut : "1FFFFFF" (1s)*
BoardType	*W*	*3H*	*[7:0]=BaseLTM* *[8]=0 si esclave* *[8]=1 si maître* *[9]=0 si LTMmin*[1] *[9]=1 si LTMmax*[2]	*BAR1*	*défaut : "f0"* *défaut : 0*
Fifo2Data	*R*	*1H*	*[31:0]*	*BAR0*	
Fifo2Status	*R*	*1H*	*[9]=Empty* *[8]=Full* *[7:0]=WC*	*BAR1*	*Fifo Word count*
Version	*R*	*2H*	*[31:0]=version#*	*BAR1*	*[Année][Mois][Jour][Num]*[3]

Tableau 7.1 : Les ordres de programmation

1. lecture de Base+2,+3,+5,+6 (4mots)
2. lecture de base à base+6 (7 mots)
3. [WW][XX][YY][ZZ] Hexa, par exemple : "08051801" pour 18 mai 2008 #01

8. Codage du mot *DEVICE_ID fpga cPCI*

Le *FPGA cPCI* possède un espace de configuration dont le contenu du mot *Device_ID* est utilisé par le système pour déterminer le driver logiciel à utiliser.

Pour la carte *TimeStamp* : *Device_ID=X"2EEE"*.

CARTE TEST PREL2

1. Introduction

La logique de déclenchement de l'expérience *HESS-II* est composée d'un niveau 1 et d'un niveau 2 (Le niveau 1 a été succintement décrit : Voir " § page 79 "). Pour des raisons d'efficacité, chacun de ces niveaux possède un *PréL1* et un *PréL2* localisé, prés des *photomultiplicateurs*, sur les cartes analogiques des tiroirs.

La carte *Test PréL2 cPCI,* permet de tester les circuits *ASICs PréL2* câblés dans leur environnement (tiroirs). Ce système est indépendant du banc de test utilisé pour caractériser les *ASICs PréL2*.

Cette carte *Test PréL2* permet de relire des trames en provenance des *ASICs PréL2* et d'être interfacé avec le bus *compactPCI.* L'interface avec le bus *cPCI* est effectué par un *fpga Altera BGA EP1C4F324C6* de la série *Cyclone*. La fonctionnalité de ce circuit est complètement décrite dans une note[1]. La fonctionnalité des *ASICs PréL2* est complètement décrite dans une note[2].

Le contrôle de l'interface avec les *ASICs PréL2* est effectué par un *fpga Altera EP1C12Q240C6*.

Le synoptique de l'*ASIC PréL2* est représenté sur la figure 1.1.

Sur une carte analogique, les signaux issus des 8 *photomultiplicateurs* sont comparés à un seuil bas et à un seuil haut pour former un mot de 2X8 bits ou "*pattern"*. Le circuit *PréL2* permet de resynchroniser ces "*patterns*" (Seuil bas=*ChanInL*, Seuil haut=*ChanInH*) avec le signal de déclenchement *L1A* (Retard d'une valeur pré-programmée *DelIn*). L'information resynchronisée est envoyée en série vers la logique de déclenchement de niveau 2.

Différents paramètres sont programmables dans l'*ASIC PréL2*, le tableau 1.1 récapitule ces informations.

AD[1:0]	Param		Défaut
00	[5:0]	*DelIn*	0
01	[7:0]	*TestReg*	0
10	[1:0]	*ID*	0
11	[0] [1]	*UseTest* *Mode*	0 0

Tableau 1.1 : Table de programmation de l'ASIC

DelIn est le retard codé par pas de 2ns nécessaire pour avoir une bonne synchronisation avec le signal *L1A*.

TestReg correspond à un pattern de test que l'on désire sérialisé à la place du pattern d'entrée (utile pour caler le retard *DelIn*).

ID est l'identificateur de l'ASIC concerné.

1. Interface compactPCI (cPCI)
2. Logique du circuit PreL2 (version 2X8 entrées + 2 bits ID) - O. Le Dortz

Mode permet de se placer en multi-trames (*Mode*=1) et d'envoyer 5 trames consécutives centrées autour de la valeur *DelIn*.

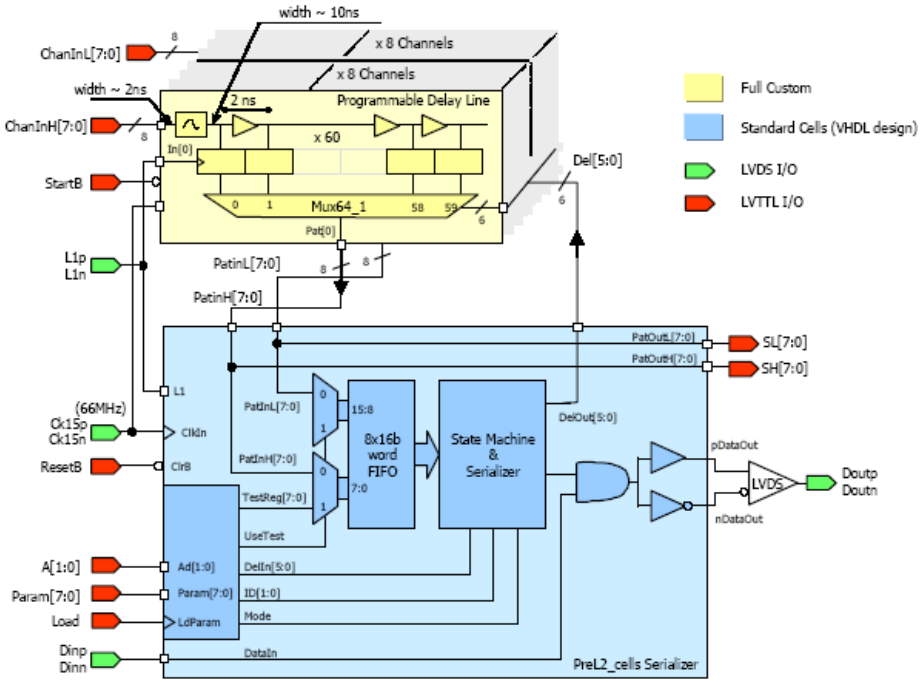


Figure 1.1: Le synoptique de l'ASIC PréL2

Afin de limiter le nombre de câble, entre les tiroirs et la *logique L2*, 4 *ASICs* sont chaînés localement (ce qui correspond à 2 tiroirs). Chaque *ASIC* est identifié, sur les bus séries, par un identificateur *ID* de 2 bits. Cet identificateur doit être préalablement chargé dans chaque *ASIC PréL2* ("00", "01", "10", "11"). Un code ne doit être utilisé qu'une seule fois pour 4 *ASICs* chaînés.

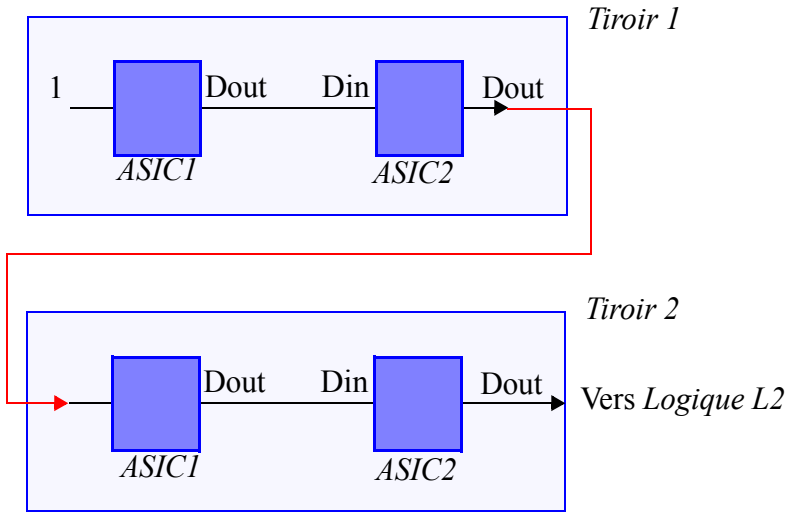


Figure 1.2 : Principe du chaînage des ASICs

2. Principe de la carte Test PréL2

La figure 2.1 donne le principe de la carte *Test PréL2*. La partie en pointillée rouge émule le second tiroir. Dans ce cas, il est possible de définir l'ordre de chaînage des tiroirs ou encore que les deux tiroirs sont chaînés à l'extérieur et reliés à l'entrée *Din*.

En fin de chaîne les données série reçues (mot de 16 bits + 2 bits *ID*) sont désérialisées et mémorisées dans une *fifo* de *256X18 bits* qui peut être relue par le *cPCI*. Il est possible de relire l'état de cette *fifo* (*FifoEmpty*, *FifoFull*) ainsi que le nombre de mots à lire (*WC*).

Le synoptique de l'*ASIC PréL2*, montre 2 parties distinctes. La première, analogique, gère le décalage des "*patterns*" d'entrées et fournit les "*patterns*" synchrones du signal *L1A* (*PatInL* et *PatInH*).

La seconde partie, numérique, gère le chaînage et le stockage temporaire de l'information.

Au niveau de la carte *Test PréL2*, il est possible, pour les 2 *ASICs* émulés, de définir les données qui seront envoyées de 2 manières différentes :

- En programmant le mode test, dans les *ASICs* émulés, soit un mot de commande *Param* (8 bits), (voir le *tableau 1.1*), tel que *Param=TestReg*. Dans ce cas *TestReg* (8 bits) représente le "*pattern*" qui sera envoyé en série sur16 bits (*TestReg* sera utilisé pour la partie basse et pour la partie haute du "*pattern*" série).
- En se substituant à la partie analogique des *ASICs* émulés en programmant les "*patterns*" *PatInL* et *PatInH* pour chaque *ASIC* émulés.

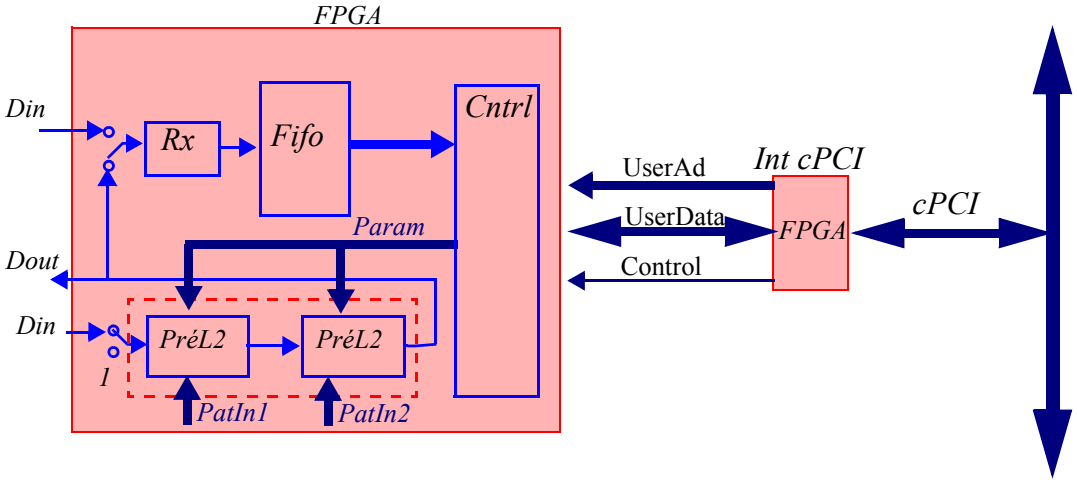


Figure 2.1 : Principe de la carte Test PréL2

2.1. Les ordres en lecture

Les ordres de lecture sont générés par des lectures *cPCI* dans l'un des 2 espaces, *BAR0* et *BAR1*[1]. L'espace *BAR0* est uniquement utilisé pour des lectures en mode *DMA*. L'espace *BAR1* est affecté à la lecture d'un mot de contrôle spécifique.

1. voir la note INTERFACE COMPACTPCI pour l'utilisation des registres *BAR*

Espace	Adresse (UserAdr[3:0])[a]	Action
Bar1	1H	*Lecture du word count fifo interne (10 bits)* [*FifoEmpty*][*FifoFull*][*WC*[7:0]]
Bar1		
Bar1		
Bar1		
Bar1		
Bar0	0H	*Lecture Fifo (18 bits)*

Tableau 2.1 : Les ordres en lecture

a. Cette adresse correspond à PciAd[6:3].

2.2. Les ordres en écriture

Les ordres d'écriture sont générés par des écritures *cPCI* dans l'un des 2 espaces, *BAR0* et *BAR1*[1]. L'espace *BAR0* est uniquement utilisé pour des écritures en mode *DMA*. L'espace *BAR1* est utilisé pour l'écriture d'un mot de contrôle spécifique.

1. voir la note INTERFACE COMPACTPCI pour l'utilisation des registres *BAR*

Espace	Adresse *(UserAdr[3:0][a])*	Action	
Bar1	01H	*Reset programmé*	
Bar1	02H	*userdata[7:0]->Param[7:0]* *userdata[9:8]->AD[1:0]* *si userdata[10]=1 alors LdParam1* *si userdata[11]=1 alors LdParam2*	*Chargement des paramètres des PréL2 émulés.* *Chargement sur 1 ou 2 PréL2*
Bar1	03H	*userdata[7:0]->PatInB1[7:0]* *userdata[15:8]->PatInH1[7:0]*	*Chargement du "pattern" PréL2 #1*
Bar1	04H	*userdata[7:0]->PatInB2[7:0]* *userdata[15:8]->PatInH2[7:0]*	*Chargement du "pattern" PréL2 #2*
Bar1	05H	*userdata[1:0]* *00 Pas de tiroir émulé par le fpga* *01 Tiroir émulé en premier* *10 Tiroir émulé en fin de chaîne*	*Définition de l'ordre de chaînage des tiroirs (connecté et émulé)*
Bar1	06H	*Reset Fifo*	

Tableau 2.2 : Les ordres en écriture

a. Cette adresse correspond à PciAd[6:3]

3. Programmation des *Fpga*

La carte *Fifo cPCI Data* comporte deux *fpga* qui sont programmés selon la norme *JTAG* d'une part et en utilisant, d'autre part, le mode *Flash Loader* (Voir " § 3.2. page 26 "). La figure 3.1 rappelle le principe de connexion des *Fpga*, des *EPROM* et du connecteur *JTAG*.

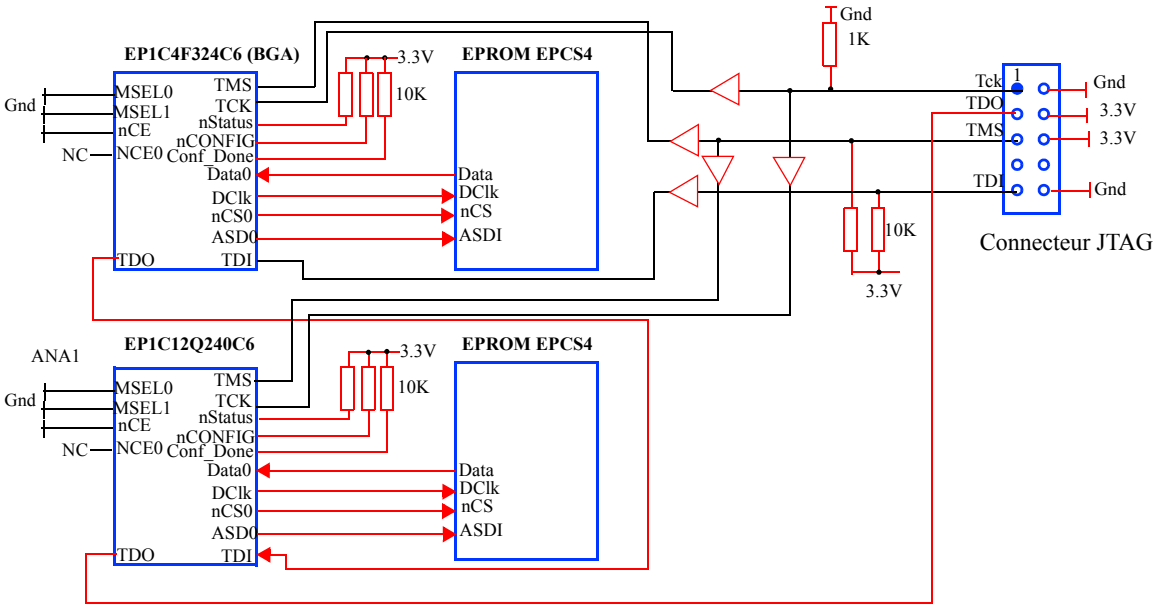


Figure 3.1 : Chaînage FPGA en mode FlashLoader

4. Codage du mot *DEVICE_ID fpga cPCI*

Le *FPGA cPCI* possède un espace de configuration dont le contenu du mot *Device_ID* est utilisé par le système pour déterminer le driver logiciel à utiliser.

Pour la carte *Test PréL2cPCI* : *Device_ID=X"2FFF"*.

SYSTÈME DE CALIBRAGE CAMÉRA

1. Introduction

La caméra HESS II doit être calibrer régulièrement de façon à déterminer le gain de chaque voie d'électronique de façon à pouvoir ajuster les hautes tensions des photomultiplicateurs (*PMs*).

L'idée est d'éclairer le plan focal de la caméra (environ 2 *m* de diamètre) avec une diode électroluminescente ($\lambda = 370nm$) placée à environ 3 *m* du plan focal. La *LED* (Light Emmiting Diode) travaille en impulsionnelle et délivre un flash lumineux de quelques nano-secondes de large. Une roue à filtres permet d'atténuer la lumière émise par la *LED* de façon à n'arracher, en moyenne, qu'un seul photoélectron des photocathodes des *PMs*. Un diffuseur assure que la lumière est uniformément répartie.

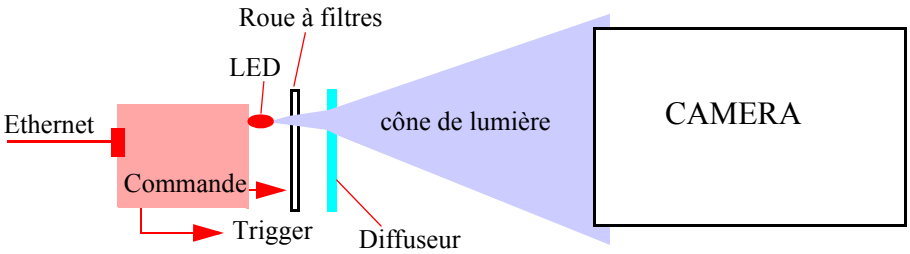


Figure 1.1 : Le système de LED et la caméra

La figure 1.1 montre le synoptique du système.

D'une manière générale, le contrôle du système *LED* est effectué par l'intermédiaire d'Ethernet. Une carte dotée d'un processeur *ARM* permet de tourner un noyau Linux 2.6 et de réaliser l'interface Ethernet. Un bus parallèle permet de commander le générateur de *LED* (commande marche/arrêt, fréquence et délai programmable entre le signal *Trigger* et le flash *LED*). La roue à filtre est commandée par une liaison RS232 issue de la carte processeur.

2. Principe

La connexion du *FPGA* avec la carte processeur *Linux* se fait par un bus parrallèle de 16 bits, sans signaux de contrôle. Les 16 bits sont interprêtés de la manière suivante :

[15] *Strobe*
[14:12] *Ordre*
[11:0] *Donnée*

2.1. Start/Stop

A l'initialisation, le générateur de *LED* est arrêté. Pour le démarrer il suffit d'envoyer l'ordre *Start* (respectivement *Stop* pour l'arrêter).

2.2. Fréquence

Il est possible d'ajuster la fréquence par l'ordre *Freq* qui charge les 12 bits de poids

forts d'un compteur de 20 bits.

Comme l'horloge de base du compteur est de 40 MHz (soit 25 ns de période) :

La valeur minimale (1H) : 1 donne une période de sortie de :
1x25x256=6400 ns (156 KHz).

La valeur maximale (FFFH) : 4095 donne une période de sortie de :
4095x25X256=26208000 ns=26,2 ms (38 Hz).

Pour obtenir 100 Hz (10 ms) il faut charger : (10x10**6)/(25*256)=1562 (61AH)

La formule générale est donnée par :

$$Val = \frac{T_s[ns]}{25 \cdot 256}$$

Remarque : une valeur de 0 est équivalente à un *Stop*.

2.3. Délai

La valeur du délai entre l'impulsion LED et la sortie *Trigger* est programmable par l'ordre *Delay* (8 bits) dans une gamme de 0 à 250 ns environ par pas d'environ 1 ns.

Le délai demandé est effectué par un double mécanisme composé d'un gros délai et d'un délai fin. La formule suivante montre comment est décomposé le délai.

$$Delay = K \cdot 25ns + K' \cdot 1ns$$

Le délai grossier est réalisé en décalant le Trigger d'un certain nombre de périodes d'horloge (25 ns) tandis que le délai fin est effectué en reprogrammant à la volée une *PLL*.

Le tableau suivant montre le délai programmé et le délai mesuré. On remarquera que dans la gamme [|1-255] le délai mesuré varie linéairement avec la valeur chargée. La pente de la droite est de 937 ps/pas

Délai	Mesuré
0	25 ns
1	937 ps
2	1875 ps
3	2812 ps
4	3750 ps
-	-
FE	253,75 ms
FF	254,68 ns

Tableau 2.1

3. Amplitude

L'amplitude du flash de lumière est ajustable au moyen d'un *DAC* de 8 bits.

4. Programmation

A partir du processeur, il est possible de sortir 16 bits de données. Le bit 15 est utilisé comme un strobe (par convention). Cela signifie qu'il faut positionner les bits [14:0] puis faire varier le bit 15 successivement de l'état bas à l'état haut puis à l'état bas. Le passage à l'état Haut du bit 15 permet de prendre en compte les autres bits.

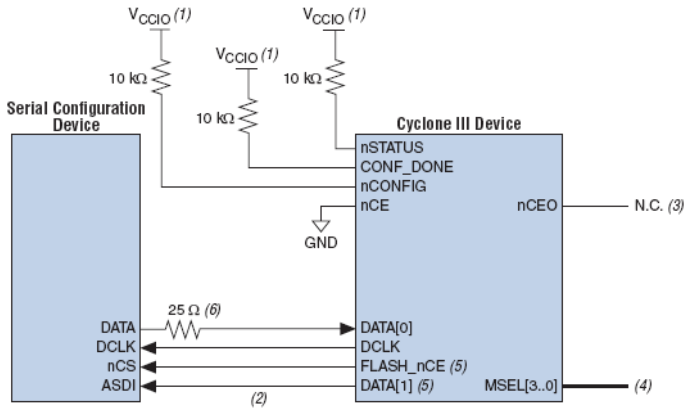
Ordre	Codage	Strobe		Défaut
Start	Data[14:12]=000	Data[15]=b[a]-h[b]-b		inactif
Stop	Data[14:12]=001	Data[15]=b-h-b		actif
Delay	Data[14:12]=010	Data[15]=b-h-b	D[7:0]=Delay	aucun
Freq	Data[14:12]=011	Data[15]=b-h-b	D[11:0]=Freq	100 Hz
DAC	Data[14:12]=100	Data[15]=b-h-b	D[7:0]=DAC	aucun

Tableau 4.1 : Tableau de programmation

a. bas
b. haut

5. Mode FlashLoader

Le mode *FlashLoader* est un mode de programmation des composants *Altera* (*EPROM*) économique. Le mode *Active Serial* autorise l'utilisation des *EPROM EPCS16*[1]. Dans ce mode le *FPGA* est programmé par le lien *JTAG* et l'*EPROM* est ensuite programmée par le *FPGA*. La figure 5.1 montre la connexion d'une *EPROM Active Serial*. La figure 5.2 explicite les connexions en *JTAG* et en mode *FlashLoader*.



[MSEL3=0] pas disponible sur EP3C25
MSEL2=0
MSEL1=1 (3.3V)
MSEL0=0

Figure 5.1 : Connexion d'une EPROM (Active Serial)

1. *EPROM Altera* de type *Active Serial*

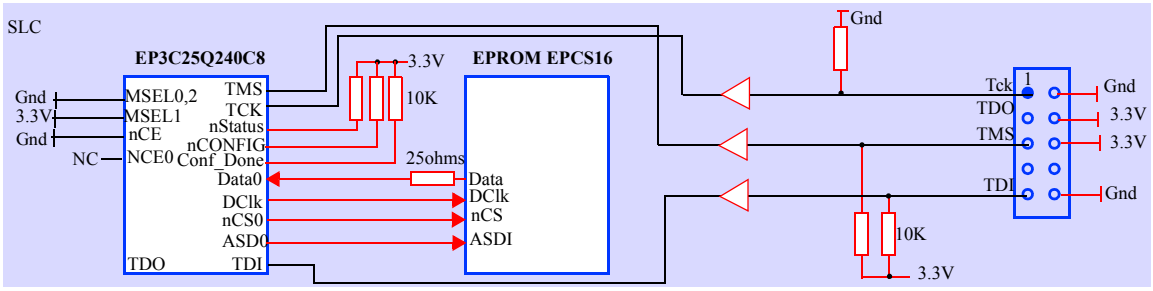


Figure 5.2 : Chaînage FPGA en mode FlashLoader et JTAG

DRIVERS CPCI

1. Introduction

Les cartes accédées par le *cPCI* nécessitent un pilote. Ce pilote est est automatiquement sélectionné par le système en lisant l'espace de configuration des *fpga* concernés.

Le tableau 1.1 donne la valeur (Hexadécimal) du mot *Device ID* pour chaque carte accédée en *cPCI*.

Carte	Device ID
Fifo Slc	2AAA
Fifo Data	2BBB
Trigger	2CCC

Tableau 1.1 : Cartes et Device ID