Manuel de Référence de la carte LLRF_V2

O. Le Dortz, LPNHE

1. Contexte

La carte LLRF_V2 est un module CompactPCI/PXI de hauteur 3U, occupant deux emplacements du châssis et composé d'une carte mère (ASSERVNUM_V2_MERE) et d'une carte fille (ASSERVNUM_V2_MEZZA2). Son rôle est de proposer un système numérique de contrôle radiofréquence bas niveau (Low Level RF) de cavités accélératrices supraconductrices.

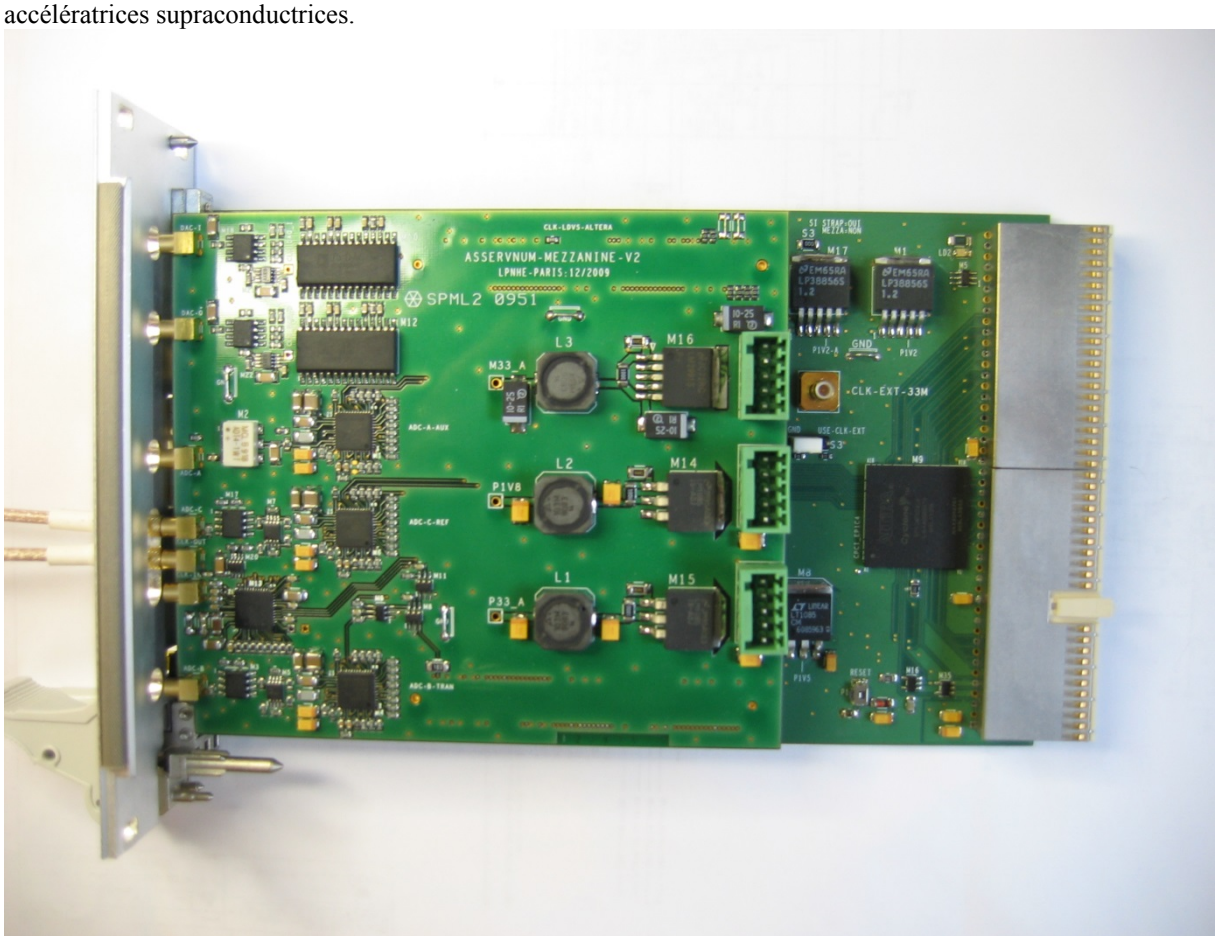


Figure 1 : La carte LLRF_V2 assemblée

Auparavant, une première carte mère avait été développée (ASSERVNUM_V1). Après l'avoir testée, il a été décidé de réaliser un second système (ASSERVNUM_V2) composé d'une carte mère numérique et d'une carte fille analogique. Finalement, la carte mezzanine ASSERVNUM_V2_MEZZA2 est une seconde version de carte fille.
Le présent document se concentre sur la description de l'assemblage ASSERVNUM_V2_MERE / ASSERVNUM_V2_MEZZA2.

2. Description Hardware

La Figure 2 représente un synoptique des différents composants électroniques de l'assemblage. Les éléments en vert sont montés sur la carte mezzanine, le reste se situe sur la carte mère.

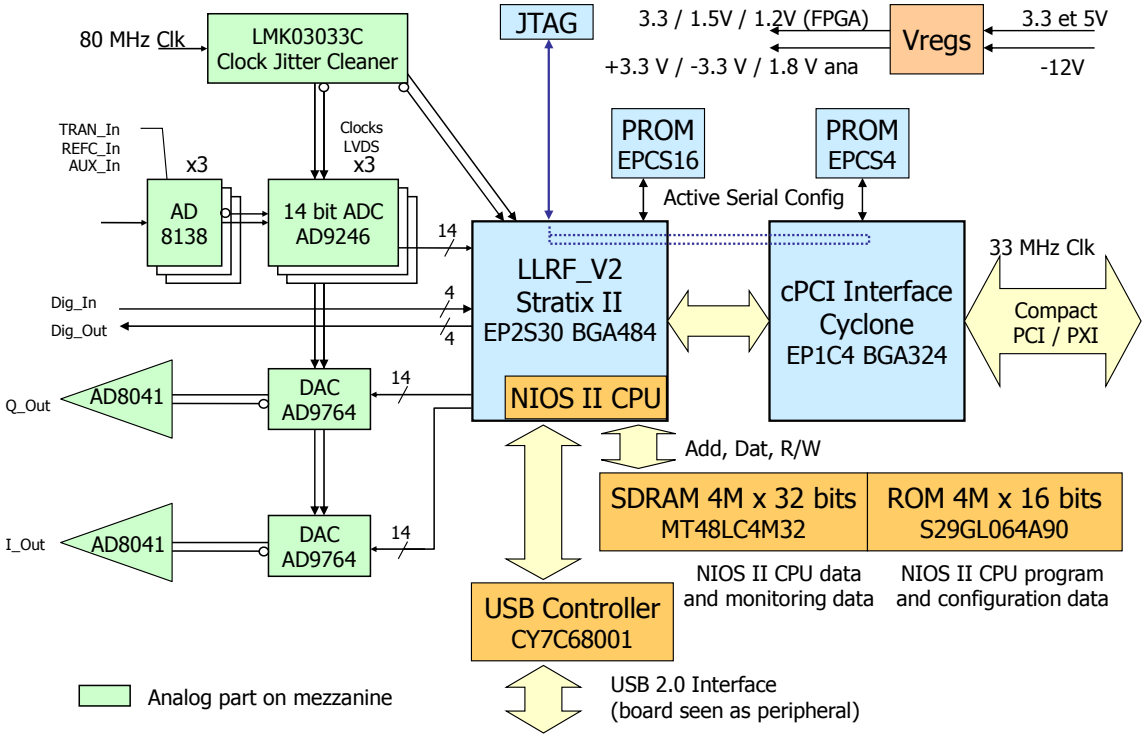


Figure 2: Synoptique de l'assemblage LLRF_V2

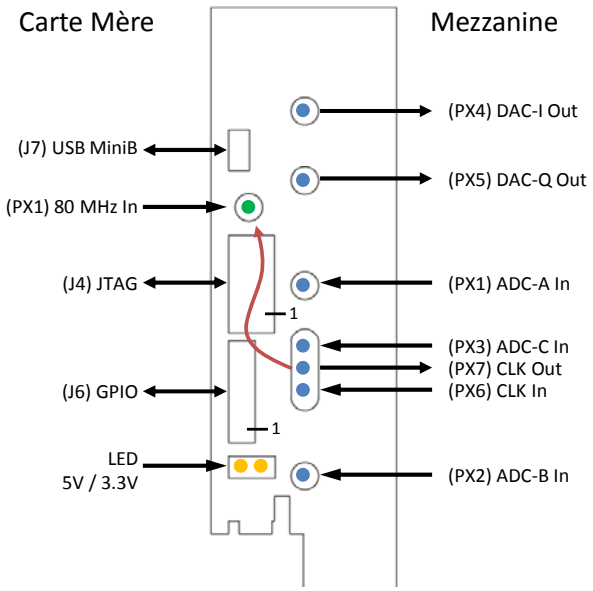


Figure 3 : Eléments de la face-avant de l'assemblage LLRF_V2

a. Carte mère numérique de format CompactPCI

La carte mère contient deux FPGA, un qui gère l'interface CompactPCI, le second effectuant l'asservissement numérique de cavité par démodulation IQ. Les schémas de cette carte sont disponibles en [1]. Une carte mezzanine s'enfiche sur les connecteurs J3 et J5. Par ceux-ci, 8 bus de 14 bits relient le FPGA principal à la carte mezzanine : 5 bus d'ADC (Inci, Refl, Trans, Refc et AuxIn) et 3 bus de DAC (I_OUT,

Q_OUT et AUX_OUT) ainsi que des signaux d'horloge et de contrôle (la carte mezzanine v2 ne gère que 3 ADC parmi ces 5 disponibles et 2 DAC parmi les 3 possibles). Hormis les différentes interfaces et connexions décrites ci-dessous, la carte héberge également deux mémoires (une RAM dynamique de type PC100 de 16Mo et une ROM Flash de 8Mo).

Horloge principale

- PX1, connecteur coudé SMB Jack (Mâle) 50Ω
- 0 à 3,3V (typique) Carré ou Sinus 80 MHz

⇨ Par défaut, la carte mère est câblée pour accepter, en entrée CLK, un signal autour de 2,35V. Les résistances R2 et R3 sont à ajuster pour définir un point milieu adapté à l'entrée attendue[1]. Par exemple, R2 =1.25kΩ et R3 =2.05kΩ définiront un point milieu à 1.25V, adapté à une entrée fournie par la carte mezzanine.

Connexion JTAG

La programmation et/ou le contrôle des deux FPGA de la carte s'effectue en connectant, par exemple, un module USB Blaster sur le connecteur JTAG en face avant (J4, format HE10). La chaîne JTAG est présentée ci-dessous:

1. EP1C4 IntCPCI
2. EP2S30 LLRF_V2
3. - Composants éventuels sur mezzanine (aucun présent sur MEZZA_V2)

En sortie du FPGA LLRF_V2, la chaîne JTAG peut se prolonger sur la mezzanine. Pour ce faire, le strap S3 doit être retiré de la carte.

Connecteur d'entrées-sorties numériques polyvalentes

4 entrées et 4 entrées d'usage général sont disponibles en face avant sur le connecteur 16 contacts J6. Les niveaux sont de type LVCMOS. Ces signaux sont gérés par le FPGA LLRF_V2. Les signaux DigIn sont des entrées de l'extérieur vers la carte, les signaux DigOut des sorties de la carte vers l'extérieur. Le tableau et la figure suivants détaillent le câblage du connecteur et le type de terminaison utilisé.

GND	16	15	DigOut [3]
GND	14	13	DigOut [2]
GND	12	11	DigOut [1]
GND	10	9	DigOut [0]
GND	8	7	DigIn [3]
GND	6	5	DigIn [2]
GND	4	3	DigIn [1]
GND	2	1	DigIn [0]

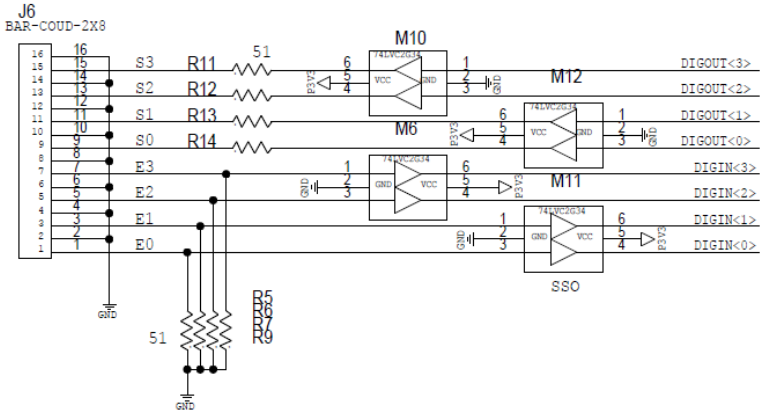


Tableau 1: Signaux du connecteur J6, vu de la face-avant

Interface CompactPCI

La carte s'enfiche dans un châssis PXI 3U par l'intermédiaire des connecteurs J1 et J2. L'interface CompactPCI est prise en charge par le FPGA IntCPCI, qui permet d'accéder aux ressources de la carte par l'intermédiaire de deux espaces mémoire, BAR0 et BAR1. Par simplicité, la carte a été déclarée auprès du CERN[2], qui dispose d'un identifiant normalisé.

[1] [1] page 1
[2] https://ess.web.cern.ch/ESS/boardIDistribution/PHP/script/singleDev.php?id=378

Elle est identifiée, sur le bus CompactPCI, avec les paramètres suivants:

- Vendor_ID 0x10DC (CERN)
- Device_ID 0x017A (LLRF_V2)
- SubSystem_ID 1

La valeur SubSystem_ID (Bits[31:16] du registre $2C de l'espace de configuration PCI) permet de différencier une carte LLRF_V2 équipée d'une mezzanine v1 (=0), d'une mezzanine v2 (=1). Le firmware du FPGA IntCPCI doit être mis à jour en version cpci_v2_rev2 pour pouvoir gérer cette variable [2].

⇨ Avant de pouvoir accéder correctement aux espaces BAR0 et BAR1, il est nécessaire d'écrire le registre $84 de l'espace de configuration PCI à 0. Cette opération, uniquement requise une fois par allumage de châssis, est effectuée, entre autres, dans l'instrument initllrf.vi du logiciel Labview de pilotage.

Connexion USB

Une interface USB 2.0 haute vitesse est implantée et prise en charge par un composant Cypress EZ-USB SX2 (Composant M4 CY7C68001). La connexion à un hôte USB s'effectue par la prise en face-avant de format mini-B J7. Le composant doit d'abord être configuré par l'intermédiaire du processeur NIOS II présent dans le FPGA principal. Une fois configuré, la carte LLRF sera identifiée, sur le PC hôte USB, avec les paramètres suivants (choisis par analogie avec les paramètres de l'interface CompactPCI) :

- Vendor_ID 0x10DC
- Device_ID 0x017B

Le composant EZ-USB SX2 communique avec le PC hôte par l'intermédiaire de 4 points de terminaisons USB utilisateurs (Endpoints) :

Endpoint	Type	Paquet Max	Buffers	Description
EP2	Bulk OUT (PC⇨FPGA)	512 octets	Double	Canal de commandes PC vers FPGA
EP4	Bulk OUT (PC⇨FPGA)	512 octets	Double	Réservé
EP6	Bulk IN (FPGA⇨PC)	512 octets	Double	Canal de réponses FPGA vers PC
EP8	Bulk IN (FPGA⇨PC)	512 octets	Double	Réservé

Tableau 2 : « Endpoints » gérés par le composant USB

A l'heure actuelle, seuls 2 de ces endpoints sont exploités : le PC envoie les commandes vers la carte par EP2 et le FPGA renvoie ses réponses par EP6.

Plus de détails sur le fonctionnement de cette interface sont donnés en annexe II.

b. Carte mezzanine analogique

Le schéma de la carte mezzanine est disponible en [3]. Une PLL (M13 LMK03033C) reçoit l'horloge principale de 80 MHz en face-avant et la distribue aux 3 ADC et 2 DAC présents sur la carte mezzanine ainsi qu'en face-avant afin qu'elle puisse être utilisée par la carte mère.

Les échantillons des 3 ADC sont envoyés, à une fréquence de 80 MHz, à la carte mère par les connecteurs d'interface J1 et J2. De même, les 2 DAC reçoivent, à une cadence de 80 MHz, leurs données de la carte mère par ces deux connecteurs.

Connexions en face avant

Des connecteurs de type coudé MMCX Jack sont utilisés pour les 7 entrées-sorties

- (PX6) CLK_IN : Sinus -250 mV/+250 mV 80 MHz. L'entrée de référence de la PLL, en configuration unipolaire.

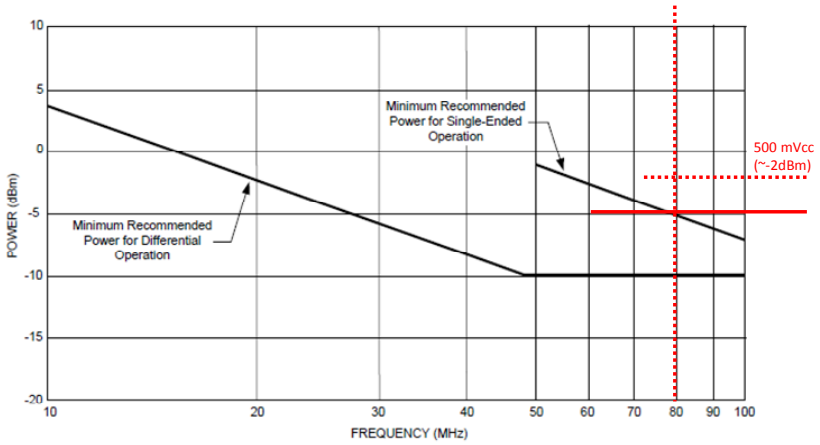


Figure 4 : Puissance minimum recommandée de l'entrée OSCin de la PLL LMK03000 family datasheet [4]

Le fabricant recommande (cf. figure ci-dessus) une puissance minimale en entrée de –5dBm en mode unipolaire à 80 MHz.

- (PX7) CLK_OUT : Carré 0/2.5V 80 MHz pour alimenter la carte mère via son connecteur PX1
- (PX1) ADC_A : Voie de test avec transformateur, 10 MHz, -500 mV/+500 mV typ, inversée. Par défaut, une résistance de 50Ω à la masse a été câblée (R129) mais elle peut être retirée.
- (PX2) ADC_B : 50Ω à la masse, 10 MHz, -500 mV/+500 mV typ, inversée
- (PX3) ADC_C : 50Ω à la masse, 10 MHz, -500 mV/+500 mV typ, inversée
- (PX4) DAC I : Sortie I, 50Ω série + 33 nF à la masse
  Sur 50Ω à la masse, min=-260 mV ; max=260 mV
- (PX5) DAC Q : Sortie Q, 50Ω série + 33 nF à la masse

Fonctionnement des ADC

Nous commentons ici la voie d'ADC B, schéma [3] page 2. Le comportement de la voie d'ADC C est similaire.

L'entrée PX2 est terminée sur 50Ω et entre dans un amplificateur (M3 AD8041) de gain 2. Le driver différentiel d'ADC (M5 AD8138) adapte l'entrée unipolaire en signal différentiel dont le point milieu est adapté à l'ADC (CML_B=1V). Enfin, un réseau RC (R60,R61=33Ω ; C23=20pF) réduit les composantes HF du signal injecté en entrée de l'ADC (M4 AD9246). A chaque période d'horloge, l'ADC renvoie vers la carte mère via le connecteur carte-à-carte J1 les 14 bits de l'échantillon numérique (DRF_B[14:0]) ainsi qu'un signal OVR_B (Out of Range) qui passe à 1 lorsque le signal présent à son entrée dépasse sa gamme.

⇨ Un condensateur de 100nF, absent du schéma initial, doit être ajouté sur le signal CML_B (resp. CML_C) reliant M4.34 et M5.2 (resp. M6.34 et M7.2).

	Entrée Vin	Sortie AD8041	ADC Vin+	ADC Vin-	Sortie Binaire	Compl. à 2
Voie B	PX3	M3.6	M4.30	M4.31	DRF_B[13 :0]	
Voie C	PX2	M17.6	M6.30	M6.31	DRF_C[13 :0]	
Max	+500 mV	+1 V	0.5 V	1.5 V	$0000	**$2000**
Zero +	+ε	+2ε	1-2ε	1+2ε	$1FFF	**$3FFF**
Mid	0 V	0 V	1 V	1 V	$2000	**$0000**
Zero -	-ε	-2ε	1+2ε	1-2ε	$2001	**$0001**
Min	-500 mV	-1 V	1.5 V	0.5 V	$3FFF	**$1FFF**

Tableau 3 : Résumé des niveaux statiques attendus sur les voies d'ADC B et C

Le Tableau 3 résume les niveaux statiques attendus à chaque niveau de la chaîne analogique ainsi que le compte d'ADC attendu lorsque configuré en codage binaire et en complément à 2 sur 14 bits. Il est à noter que, par commodité de routage, les ADC ont été branchés avec une inversion. Un niveau maximum sur l'entrée ADC donnera le compte d'ADC minimal ($2000) alors qu'un niveau minimum donnera le compte d'ADC maximum ($1FFF). Les lignes Zero+ et Zero- correspondent à un signal équivalent à plus ou moins un LSB. Compte tenu du réseau RC en entrée de l'ADC, les niveaux de signaux dynamiques peuvent être réduits, selon la fréquence d'entrée.

Configuration des éléments

Les ADC sont configurables par une interface SPI. Le système est prévu pour qu'ils soient configurés en sortie complément à deux, qui n'est pas leur mode par défaut. Quant à la PLL, elle doit être configurée par son interface microwire pour fonctionner correctement: se verrouiller sur un signal d'entrée de 80 MHz et activer les diverses horloges de sortie à 80 MHz. La configuration de la PLL est décrite en annexe I.

c. Bilan des horloges du système

La carte gère deux domaines d'horloge distincts : un à 33 MHz, issu du châssis PXI et un à 80 MHz, issu du système RF. La Figure 5 résume les distributions et transformations d'horloge réalisées sur les cartes mère et fille. A l'allumage du châssis, l'horloge 33 MHz est disponible; mais les horloges dérivées du 80 MHz ne sont disponibles qu'une fois que la PLL est configurée.

⇨ Dans le cas d'une utilisation « sur table » sans brancher la carte dans un châssis PXI à des fins de débogage, l'horloge 33MHz peut être amenée par le connecteur PX2 sérigraphié CLK-EXT-33M. Il faut alors placer le cavalier S3 en position USE-CLK-EXT (milieu-droite). En fonctionnement normal en châssis, s'assurer que ce cavalier est remis en position GND (gauche-milieu) afin que le FPGA LLRF_V2 utilise l'horloge fournie par le FPGA IntCPCI.

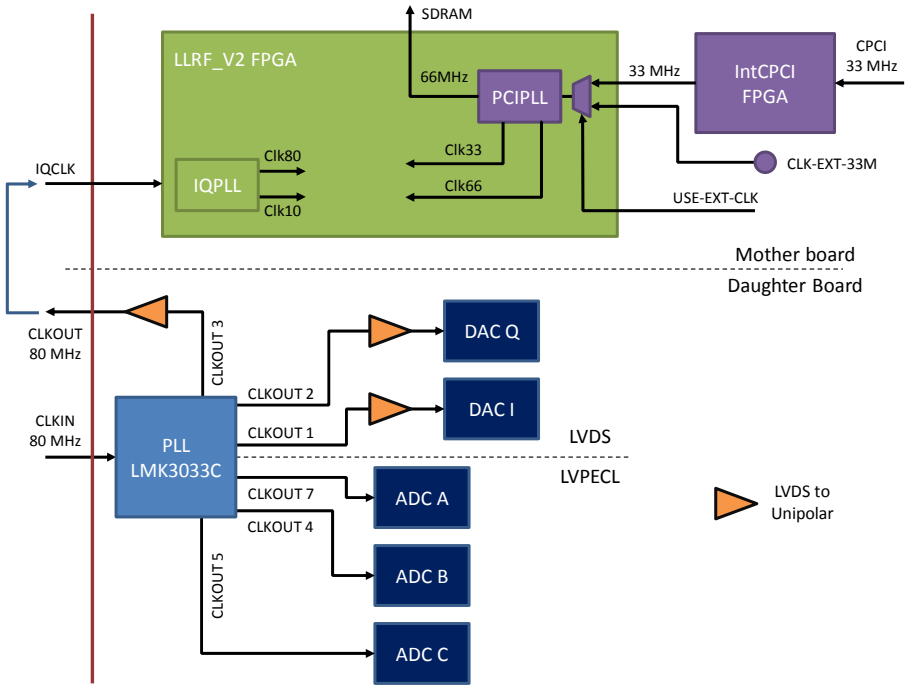


Figure 5 : Distribution d'horloges

d. Bilan des alimentations

Trois tensions d'alimentation différentes issues du châssis PXI sont exploitées : 5V, 3.3V et -12V. La table suivante indique les consommations mesurées sur les alimentations en fonctionnement sur table (33MHz et 80 MHz activés). Seule l'interface IntCPI n'était pas en fonctionnement durant les mesures.

Le logiciel Quartus nous donne une estimation[3] de consommation totale pour le FPGA LLRF_V2 de 1.5W, principalement sur l'alimentation 1.2V du cœur (1.2A). Si cette consommation est largement acceptable pour le bon fonctionnement du FPGA, il apparaît, néanmoins, que le point le plus chaud de la carte mère est le régulateur M1 qui fournit le 1.2V interne du FPGA à partir du 3.3V du châssis PXI (50°C en test sur table, donc sans ventilation du châssis).

Les mesures ont été effectuées avec le FPGA IntCPCI statique. Or, le logiciel Quartus donne une estimation de consommation pour ce FPGA de 225mW dont 70mW statique. Par conséquent, en fonctionnement normal, le FPGA IntCPCI devrait contribuer à **150mW** de consommation supplémentaire répartie sur le 3.3V (entrées-sorties) et sur le 5V (à partir duquel le régulateur M8 fournit le 1.5V interne du FPGA).

Alimentation		Carte Mère	Carte Fille	Courant Total	Puissance Totale
3.3 V	*Statique*	*0.35 A*	1.15 A	*1.50 A*	*4.95 W*
	Dynamique	*0.80 A*		*1.95 A*	*6.44 W*
	Total	1.15 A		**2.30 A**	**7.60 W**
5 V		20 mA	110 mA	**130 mA**	**650 mW**
-12 V		0 mA	90 mA	**90 mA**	**1.08 W**

Tableau 4 : consommations mesurées, hors FPGA IntCPCI

Sur la carte fille, les composants les plus sollicités en termes de consommation sont, d'une part le régulateur qui fournit l'alimentation analogique 1.8V pour les ADC (45°C sur table), et surtout la PLL qui atteint une température en surface de 56°C en situation non-ventilée.

Dans le cas d'une utilisation sur table, seuls le 5V et le 3.3V peuvent être acheminés par le connecteur PXI J1 pour obtenir un fonctionnement du FPGA principal, de la PLL et de tous les éléments numériques.
Le -12V n'est utilisé que pour le front-end analogique de la mezzanine.

[3] Estimation basique effectuée sans fournir de table de fréquence de commutation des entrées

3. Description du FPGA principal LLRF_V2

a. Principe de fonctionnement

Le FPGA principal réalise l'asservissement numérique dédié au contrôle d'une cavité accélératrice. La carte accepte en entrée des signaux, issus d'un châssis RF, qui ont été transposés à une fréquence intermédiaire de 10 MHz. Ces signaux sont échantillonnés sur la carte mezzanine, sur 14 bits à 80 MHz et sont ensuite traités par le FPGA principal : correction d'offset, démodulation IQ. La voie d'ADC correspondant à la mesure de puissance transmise de la cavité est ensuite, après la démodulation IQ, comparée à une consigne ; la mesure de l'erreur entre la puissance transmise et la consigne permet enfin, après filtrage, de calculer deux commandes (I et Q) à appliquer à un modulateur externe pour réduire l'erreur entre la mesure et la consigne.

La Figure 6 représente les éléments du FPGA principal : en haut, les éléments en vert forment la chaîne d'asservissement IQ à proprement parler (IQ Core) ; cette chaîne IQ est contrôlable et observable par l'intermédiaire du « système SOPC », représenté en bas.

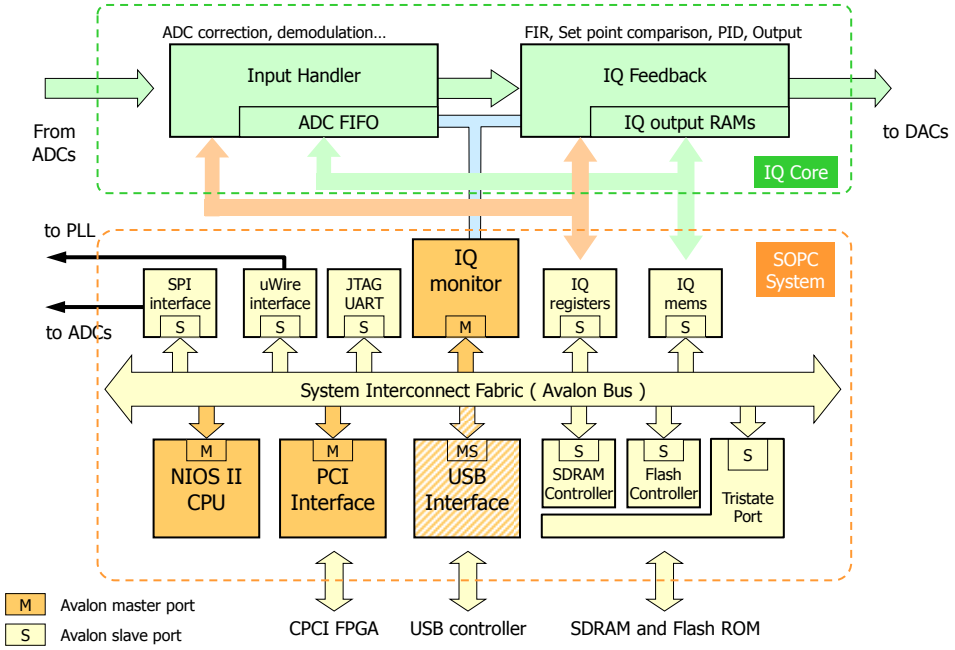


Figure 6 : Synoptique du FPGA principal LLRF_V2

Par défaut, les voies d'ADC ont les fonctions suivantes :

- ADC_B Puissance Transmise Tran_In
- ADC_C Horloge de référence Refc_In
- ADC_A Voie de test avec transformateur Auxi_In

Néanmoins, cette configuration peut être modifiée en recompilant le projet avec des paramètres différents.

b. Chaîne d'asservissement IQ

La chaîne d'asservissement IQ est détaillée sur la figure suivante. La voie d'entrée principale (TRAN) alimente la boucle d'asservissement dont les sorties Iout et Qout sont destinées aux 2 DAC présents sur la carte mezzanine. Différents éléments sont ajustables par l'intermédiaire du système SOPC (éléments repérés en jaune ou orange). De même, les éléments observables par l'intermédiaire de registres de monitoring sont indiqués en rouge. Le module de démodulation IQCalc8 est détaillé en Figure 8. Toute la chaîne fonctionne à une fréquence de 80 MHz.

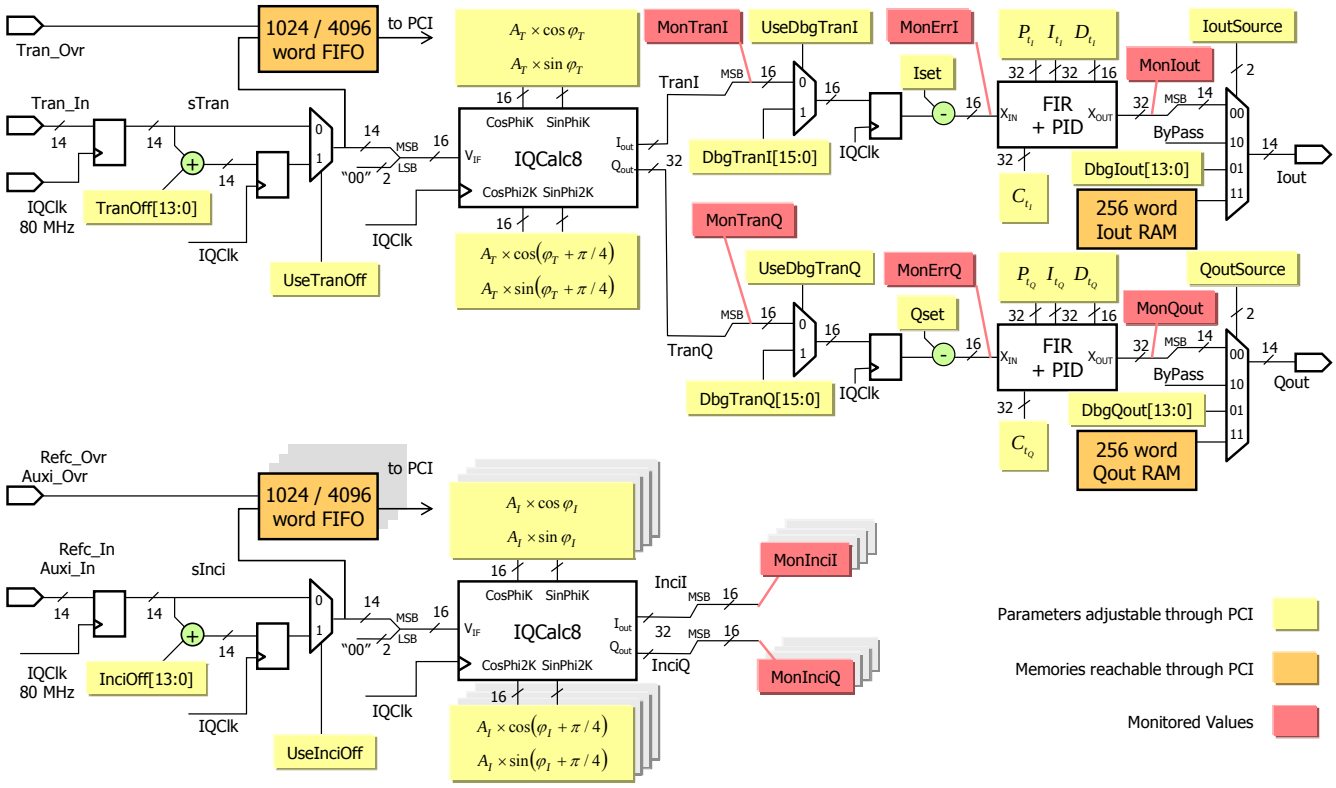
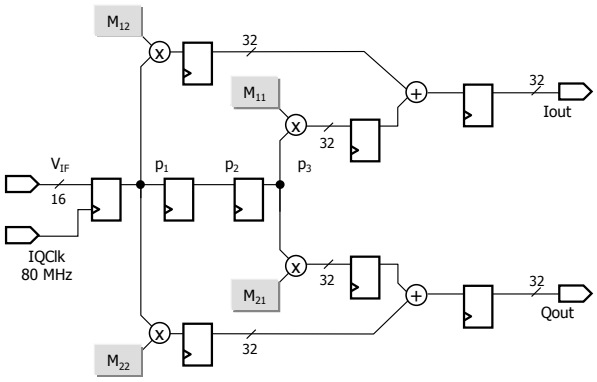


Figure 7 : détail du module IQ Core



Step	M_{11}	M_{12}	M_{21}	M_{22}
0	-CosPhiK	SinPhiK	SinPhiK	CosPhiK
1	-CosPhi2K	SinPhi2K	SinPhi2K	CosPhi2K
2	SinPhiK	CosPhiK	CosPhiK	-SinPhiK
3	SinPhi2K	CosPhi2K	CosPhi2K	-SinPhi2K
4	CosPhiK	-SinPhiK	-SinPhiK	-CosPhiK
5	CosPhi2K	-SinPhi2K	-SinPhi2K	-CosPhi2K
6	-SinPhiK	-CosPhiK	-CosPhiK	SinPhiK
7	-SinPhi2K	-CosPhi2K	-CosPhi2K	SinPhi2K

Figure 8 : module IQcalc8 de démodulation

c. Le système SOPC

Le contrôle des fonctionnalités de l'application est réalisé par l'intermédiaire d'un système SOPC, dont le nœud central est le « bus Avalon » (cf Figure 6, partie inférieure). Le bus est multi-maître, multi-esclave, 32 bits d'adressage et de 8 à 32 bits de données. Chaque esclave ou périphérique SOPC se voit attribué une gamme d'adresses de 32 bits (que nous appellerons « adresses SOPC ») auquel il répondra sur requête d'un maître.

Plusieurs maîtres peuvent effectuer des requêtes d'écriture ou de lecture sur le bus, l'arbitrage s'effectuant de manière transparente pour l'utilisateur. Les différents maîtres sont :

- Un processeur NIOS II, auquel on peut attribuer des tâches d'initialisation à l'allumage du système (activation de la connexion USB, réglage des ADC, de la PLL) ou que l'on peut utiliser pour du débogage. Une section ultérieure est consacrée à son usage.

- L'interface CompactPCI : le FPGA IntCPCI offre à la carte une interface de type cible PCI (*target*). Les requêtes issues du bus CPCI d'écriture ou de lecture aux BAR0 ou BAR1 de la carte sont transférées au module « PCI Interface » du système SOPC. Celui-ci traduit ces requêtes en écriture ou lecture sur le bus Avalon. Le module joue ainsi le rôle de pont CPCI⇒ Avalon ; esclave du bus PCI et maître du bus Avalon.
- IQ Monitor : périodiquement, et de façon paramétrable par l'utilisateur, un bloc de données de monitoring peut-être écrit dans une portion de la mémoire RAM.
- Une interface USB : de manière alternative à l'interface PCI, la carte peut être accédée de l'extérieur par une interface USB 2.0. Ce module est mixte maître-esclave : il peut être configuré via le bus Avalon et jouer le rôle de pont USB⇒Avalon via son interface maître Avalon.

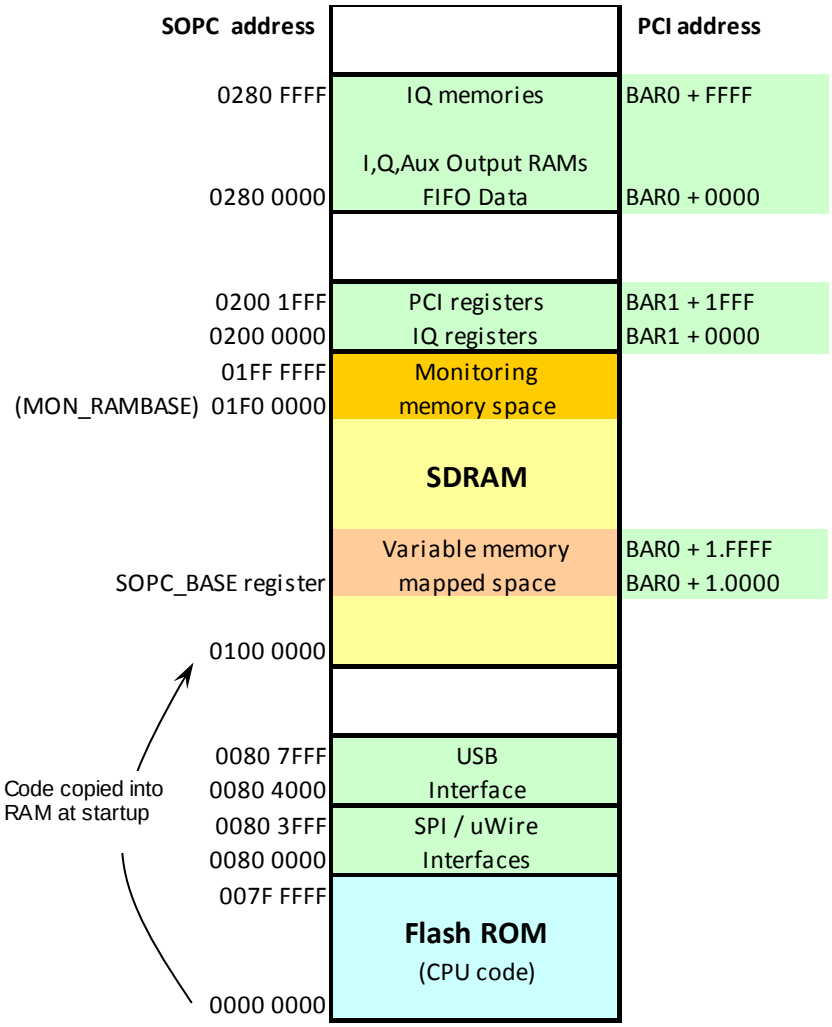


Figure 9 : Adressage des principaux périphériques du système SOPC

Divers périphériques ou esclaves sont implantés autour du bus Avalon. Certains ont été spécifiquement conçus pour l'application afin de paramétrer ou de visualiser les éléments de la chaîne d'asservissement IQ : IQ mems, IQ registers. D'autres permettent de piloter les composants extérieurs au FPGA : contrôleurs de mémoire ROM ou SDRAM, interface SPI pour configurer les ADC et interface microwire pour configurer la PLL. Enfin, d'autres sont principalement exploités par le processeur NIOS : JTAG UART (qui permet de diriger les flux stdin, stdout et stderr du programme exécuté par le processeur vers le bus JTAG du FPGA), timer, etc...

La Figure 9 représente les principaux périphériques du système. A gauche sont indiquées les gammes d'adresses SOPC auxquelles les périphériques répondent à des requêtes issues de maîtres Avalon. A droite sont représentées les gammes d'adresses correspondantes du point de vue du bus CPCI.
Ainsi, l'espace IQ registers/PCI registers, dédié aux registres de la chaîne d'asservissement IQ, est directement

mappé sur le BAR1 de la carte. Le périphérique IQ memories est mappé sur une partie du BAR0. La seconde partie de l'espace BAR0 est une fenêtre de $10000 (65536) octets que l'on peut mapper sur une zone particulière de l'espace SOPC, déterminée par le registre SOPC_BASE du périphérique IQ registers.

Les tables en annexe III (pages 28 et suivantes) donnent les adresses de tous les registres des périphériques principaux du FPGA. Les adresses et offsets sont exprimés en hexadécimal. Les registres et mots de mémoire ne sont accessibles de façon garantie qu'en accès 32 bits de données. Aussi, tous les registres ont des offsets par rapport à l'adresse de base du périphérique multiples de 4.

Exemple

Le registre TEST_PCI est un mot de 32 bit que l'on peut écrire et lire pour vérifier le bon fonctionnement des accès au bus Avalon. Il est accessible à l'offset $1000 de l'espace BAR1 ou de l'adresse de base SOPC du périphérique IQ registers ($0200 0000). Du bus CPCI, on y accédera à l'adresse BAR1+$1000, du bus Avalon, ce sera par l'adresse $0200 1000. La première colonne ($400) indique le numéro de registre 32 bit dans la gamme du périphérique auquel il appartient. Ce paramètre est utilisé essentiellement pour les accès registres par l'intermédiaire du processeur NIOS.

Le registre SOPC_BASE (accessible via PCI à l'adresse BAR1+$1010) permet de définir la fenêtre de l'espace SOPC que l'espace PCI BAR0+$10000 à BAR0+$1FFFF pointe. Par exemple, en écrivant $00800000 dans ce registre, un accès PCI à BAR0+$1wxyz entrainera un accès au mot de 32 bits à l'adresse SOPC $0080wxyz.

⇨ Le FPGA est cadencé par deux domaines d'horloge comme décrit Figure 5. Certains registres ou mémoires du bus Avalon sont cadencés par l'horloge IQ de 80 MHz. En fonctionnement standard, en châssis, l'horloge 33 MHz est toujours présente, mais l'horloge IQ peut ne pas l'être (entrée face avant non connectée ou non activée, PLL LMK non configurée ou PLL du FPGA principal non verrouillée). Sur la table des registres en annexe III, seuls les registres dont les lignes apparaissent en bleu clair sont cadencés à 33 MHz. Un accès à un registre cadencé à 80 MHz lorsque l'horloge IQ n'est pas prête engendrera un blocage du système.
Il convient donc à l'utilisateur de s'assurer que l'horloge IQ est bien disponible avant d'accéder aux registres non repérés en bleu (par exemple, en vérifiant la valeur du bit 1=IQPllLocked du registre CLK_CSR).

- **Contrôle des ADC d'entrée**

Comme écrit précédemment, les 3 ADC d'entrée jouent un rôle particulier. Le registre en lecture seule INPUT_CONF indique quelles voies d'ADC correspondent aux signaux Transmis, Horloge de Référence et Auxiliaire.

Sur chaque entrée d'ADC, une FIFO permet, sur requête, d'extraire une trame d'échantillons successifs à 80 MHz. Le registre FIFO_CSR permet de réinitialiser ou de déclencher le remplissage de ces FIFO. La procédure est la suivante pour récupérer une trame d'échantillon d'un ou de plusieurs ADC :

1. Ecrire $1 en FIFO_CSR pour initialiser les FIFOs.
2. Ecrire FIFO_CSR en effaçant le bit Clear (bit 0 à 0) et en mettant à 1 le bit WriteEnable des voies désirées. Cette écriture démarrera le remplissage des FIFO activées.
3. Lire le registre FIFO_CSR jusqu'à ce que toutes les FIFOs activées soient pleines (bit Full à 1).
4. Vérifier la taille de la trame enregistrée de chaque FIFO en lisant FIFO_USEDx (x=A,B,C)
5. Pour chaque FIFO, lire autant de fois que la taille de la trame indiquée à une adresse arbitraire de la gamme FIFO_DAT_x (BAR0+0000 à +0FFC ou $02800000 à $02800FFC pour A). Les comptes d'ADC sont les 14 LSB de chaque mot de 32 bits lu ; le bit Out of Range étant le 15ème.

⇨ Par défaut, la FIFO de la voie A est de profondeur 4096 mots alors que celles des voies B et C ne sont que de 1024. Il vaut donc mieux vérifier chaque registre FIFO_USEDx avant la relecture de chaque FIFO. Par ailleurs, la gamme d'adresse FIFO_DAT_A (0000-0FFC) ne couvre que 1024 mots de 32 bits ; si, pour relire toutes les données de la FIFO, une routine de lecture par bloc est utilisée (avec incrémentation de l'adresse par pas de 4 à chaque mot lu), **il faudra limiter la taille du bloc à 1024 mots consécutifs**.

L'offset des ADC peut être corrigé de deux façons : directement en configurant les ADC par interface SPI et via les registres SOPC OFF_A, OFF_B et OFF_C. Ces 3 registres permettent également d'inverser l'entrée. Si elle est activée, l'inversion a lieu avant l'application de l'offset.

- **Contrôle des sorties DAC I et Q**

Le registre DBG_OUT régit le comportement des sorties Iout et Qout vers les DAC I et Q présents sur la carte mezzanine. Comme indiqué sur la figure page 9, le signal IoutSource[1:0] (resp. QoutSource[1:0]) détermine la valeur de la sortie Iout[13:0] (resp. Qout[13:0]) : la sortie normale issue du bloc PID, une valeur constante de test DbgIout[13:0] (resp. DbgIout[29:16]), une trame d'échantillons prédéfinie écrite en mémoire interne du FPGA ou enfin la valeur d'une des voies d'ADC (mode bypass).

La table ci-dessous résume le comportement des sorties Iout et Qout selon la valeur du registre DBG_OUT. Les bits DBG_OUT[31:16] régissent le comportement de Qout et, indépendamment, les bits DBG_OUT[15:0] régissent la sortie Iout.

Cas particulier : DBG_OUT=C000C0xy pour lequel I et Q reçoivent les données de leur RAM correspondante, de façon synchrone l'une et l'autre.

DBG_OUT	QoutSource	IoutSource	Qout	Iout
4000---- 7FFF----	Debug	-	0000 3FFF	---
----4000 ----7FFF	-	Debug	---	0000 3FFF
8000----	Bypass	-	A_IN	---
8001----	Bypass	-	B_IN	---
8002----	Bypass	-	C_IN	---
----8000	-	Bypass	---	A_IN
----8001	-	Bypass	---	B_IN
----8002	-	Bypass	---	C_IN
C0xy----	RAM	-	QOUTRAM[0] to QOUTRAM[xy]	-
----C0xy	-	RAM	--	IOUTRAM[0] to IOUTRAM[xy]
C000C0xy	RAM	RAM	QOUTRAM[0] to QOUTRAM[xy]	IOUTRAM[0] to IOUTRAM[xy]

Tableau 5 : Registre DBG_OUT

Le contenu des RAM IoutRam et QoutRam est modifiable aux adresses BAR0+$5000 à BAR0+$6FFC.

- **Fonctionnement du monitoring**

Les registres de monitoring MON_TIME, MON_INCI, MON_REFL, MON_TRAN, MON_REFC, MON_OUT, MON_ERR sont mis à jour uniquement si le signal MonRegEnabled du registre MON_REGCTL est à 1.
Pour que MonRegEnabled soit à 1, il faut que les valeurs d'entrées numériques DigIn[3 :0] correspondent à la pattern et au masque MonRegPat et MonRegMask.

Indépendamment, l'enregistrement des données de monitoring dans la RAM externe s'effectue selon la valeur du signal MON_RAMCTL[MonWrite]. Pour démarrer l'enregistrement, régler le champ MonStartCond du registre MON_ RAMCTL. Lorsque la condition est vérifiée, MonWrite passe à 1 et l'enregistrement commence.
Un bloc est enregistré tous les MonPer cycles d'horloge 80 MHz, selon le format décrit dans le Tableau 7.
Pour arrêter l'enregistrement, régler de même le champ MonStopCond. Lorsque la condition est vérifiée, le signal MonWrite redescend, MonPost blocs sont encore écrits dans la RAM de monitoring avant que l'enregistrement ne s'arrête.
Pour effectuer un enregistrement immédiat, régler MonStartCond et MonStopCond à « Now ». Le Tableau 6 énumère les conditions de démarrage ou d'arrêt.

Condition	i=0	i=1	i=2	i=3
Never	$00	$10	$20	$30
Now	$03	$13	$23	$33
When DigIn[i]=0	$04	$14	$24	$34
When DigIn[i]=1	$05	$15	$25	$35
on DigIn[i] fall	$06	$16	$26	$36
on DigIn[i] rise	$07	$17	$27	$37

Tableau 6 : valeurs de MonStartCond et MonStopCond du registre MON_RAMCTL

Les blocs de moritoring sont stockés dans une zone mémoire de $10.0000 octets à partir de la valeur définie dans le registre MON_RAMBASE (par défaut, tout en haut de la SDRAM). En tout, $4000 (16384) blocs de 16 mots peuvent être stockés. Il convient de s'arranger pour que cette zone ne soit pas sur-écrite par un autre processus, et notamment de vérifier que le logiciel embarqué n'y place pas ses données telles que *heap* ou *stack*.
Une fois l'enregistrement stoppé, les blocs de monitoring peuvent être lus à partir de l'offset MON_RAMRADD (par rapport au début de la zone définie par MON_RAMBASE) jusqu'à MON_RAMWADD exclus. D'abord les MonPre blocs précédant la condition d'arrêt sont lus, puis les MonPost blocs suivant la condition d'arrêt.

⇨ Une période minimale d'environ 250 ns (MonPer>20) est nécessaire pour un bon enregistrement des données de monitoring en conditions optimales (quand notamment le processeur est stoppé). Si Mon_Per impose une durée plus faible, elle ne sera pas respectée. La période effective d'échantillonnage des blocs de monitoring est dépendante du taux d'occupation de la RAM, et ne peut pas être garantie. Il est donc conseillé, dans tous les cas, de vérifier la valeur Temps de chaque bloc.

Le dernier mot d'un bloc de monitoring est un mot de parité, calculé en effectuant un xor bit à bit des 15 mots précédents.

Offset	Data [31:0]
0	4D4F4E49 (MEZ2)
1	Time [31:0] period = 12.5 ns, tmax → 53 s
2	Inci Q [31:16] | Inci I [31:16]
3	Refl Q [31:16] | Refl I [31:16]
4	Tran Q [31:16] | Tran I [31:16]
5	Refc Q [31:16] | Refc I [31:16]
6	Aux Q [31:16] | Aux I [31:16]
7	Err Q [15:0] | Err I [15:0]
8	[29:16] = Q out [13:0] [13:0] = I out [13:0]
9	Aux Out [13:0]
10	[14] = IsPreTrig [13:0] = Remaining Post Trigs [13:0]
11	Not specified
12	Not specified
13	Not specified
14	Not specified
15	Parity[31:0]

Tableau 7 : Format du bloc RAM enregistré

d. Projet Quartus

Le projet permettant de compiler le code du FPGA LLRF_V2 est téléchargeable en [5]. La version 12.1 (SP1) de Quartus II est nécessaire pour ouvrir et modifier le projet. Après extraction de l'archive :

- lancer Quartus II et ouvrir le ficher de projet *<base de l'archive>/stratix2s30_mez2_quartus12.1/llrf_v2.qpf*.
- dans Quartus, sélectionner la révision de référence llrf_v2_usb, qui contient les développements les plus récents, et sur laquelle est basé ce manuel. Menu « Project/Révisions », sélectionner llrf_v2_usb et si cette révision n'est pas la révision courante, cliquer le bouton « Set Current ». Cliquer « OK » pour sortir de la fenêtre des révisions

Le projet est principalement décrit en langage VHDL, dont les sources se trouvent dans le répertoire *<base de l'archive>/sources_llrf_v2_q12.1*. Le système SOPC est décrit par l'outil « SOPC Builder » ; il est ajouté au projet sous forme de megacore dont le fichier de base est *<base de l'archive>/stratix2s30_mez2_quartus12.1/sopc_llrf_v2_usb.qip*. Les contraintes temporelles du projet, destinées au module TimeQuest sont écrites dans le fichier *<base de l'archive>/stratix2s30_mez2_quartus12.1/ llrf_v2_usb.sdc*.

Le fichier à télécharger par le lien JTAG dans le FPGA EP2S30 de la chaîne JTAG est *<base de l'archive>/stratix2s30_mez2_quartus12.1/llrf_v2_usb.sof*.
Le fichier permettant de programmer la PROM de configuration EPCS16 attachée au FPGA est *<base de l'archive>/stratix2s30_mez2_quartus12.1/llrf_v2_usb.jic*. Dans le cas d'une recompilation du projet, le fichier .jic ne se remet pas à jour automatiquement. Il faut, après compilation :

- Ouvrir l'outil de conversion de fichier : Menu « File/Convert Programming File... »
- Dans la fenêtre qui s'ouvre, cliquer « Open Conversion Setup Data... » puis ouvrir le fichier llrf_v2_usb.cof dans le répertoire du projet Quartus
- Cliquer le bouton « Generate » et « Yes » pour écraser le fichier .jic précédent.

Avant la première recompilation du projet après extraction de l'archive, il est nécessaire de refaire générer l'arborescence du système SOPC par SOPC Builder. Pour cela, lancer « SOPC Builder » dans le menu « Tools », ouvrir *sopc_llrf_v2_usb.sopc* puis cliquer sur « Generate ». L'IP NIOS II est nécessaire pour engendrer une version du code du processeur NIOS II *cpu.vhd* qui ne soit pas limitée dans le temps[4].

- **Personnalisation du projet**

En cas de modification d'une partie du code du projet, il est conseillé de modifier la valeur que retournera le registre SOPC VERSION. Avant compilation du projet, sélectionner le menu « Assignments/Settings... » puis dans la colonne de gauche de la fenêtre, sélectionner « Default Parameters ».

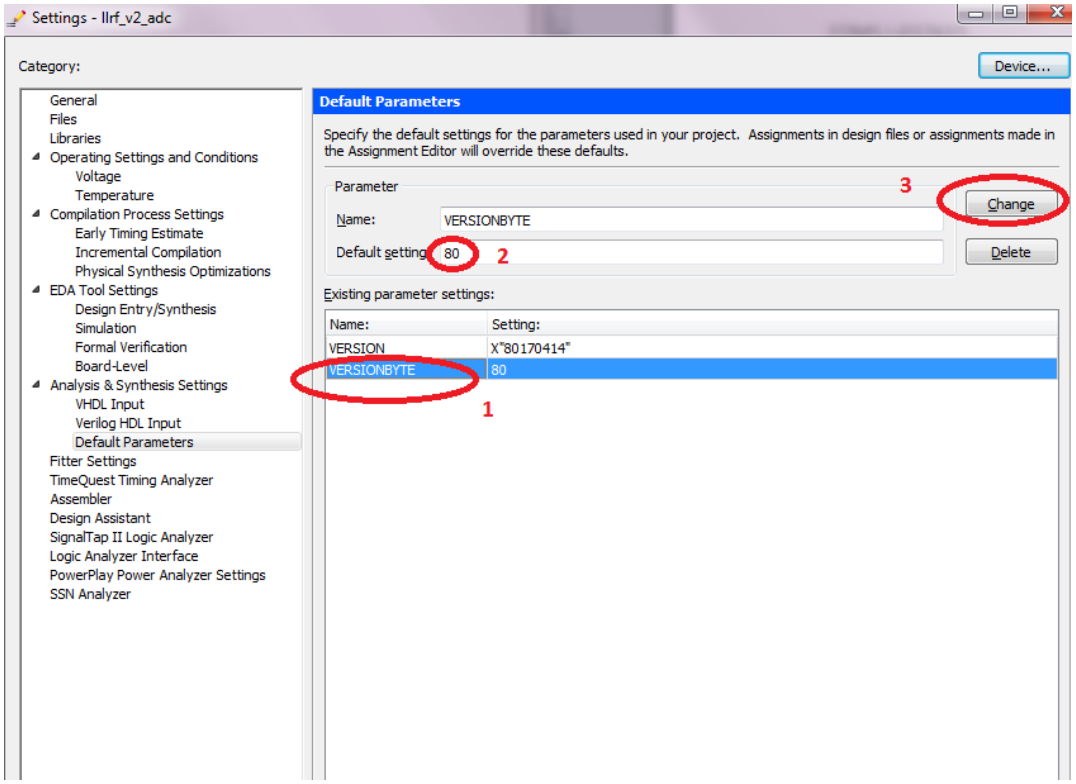


Figure 10 : Réglage de la version du projet

[4] Une licence d'utilisation de l'IP NIOS II est normalement disponible sur le serveur de licence de l'IN2P3 (ccflex*.in2p3.fr). A la génération du module cpu, le message *"encrypted license found. SOF will not be time-limited"* devrait apparaître.

Changer ensuite la valeur du paramètre VERSIONBYTE à la valeur de version souhaitée tel qu'indiqué sur la Figure 10. Par exemple entrer 83 pour la version 8.3. A la compilation, le paramètre générique VERSION du fichier top du projet (*llrf_v2_mez2_usb.vhd*) sera remis à jour en incluant la date de mise à jour. Pour pouvoir plus facilement discerner les cartes équipées de mezzanine v2 par rapport à celles équipées de mezzanine v1, les versions de firmware pour les cartes LLRF_V2 équipées de mezzanine v2 ont le MSB à 1 (Version 8.0 minimum).
En pratique, le fichier de paramètre du projet *llrf_v2_usb.qsf* contient une directive qui lui permet d'exécuter avant chaque compilation, le script *auto_set_version.tcl* :

```
set_global_assignment -name PRE_FLOW_SCRIPT_FILE "quartus_sh:auto_set_version.tcl"
```

Ce script appelle lui-même *set_version.tcl* qui met à jour le paramètre générique VERSION en fonction de VERSIONBYTE et de la date courante. A noter également que le code de VERSION est recopié dans le paramètre USERCODE du fichier *llrf_v2_usb.sof* destiné à reconfigurer le FPGA, ce qui facilite la gestion de multiples fichiers de programmation.

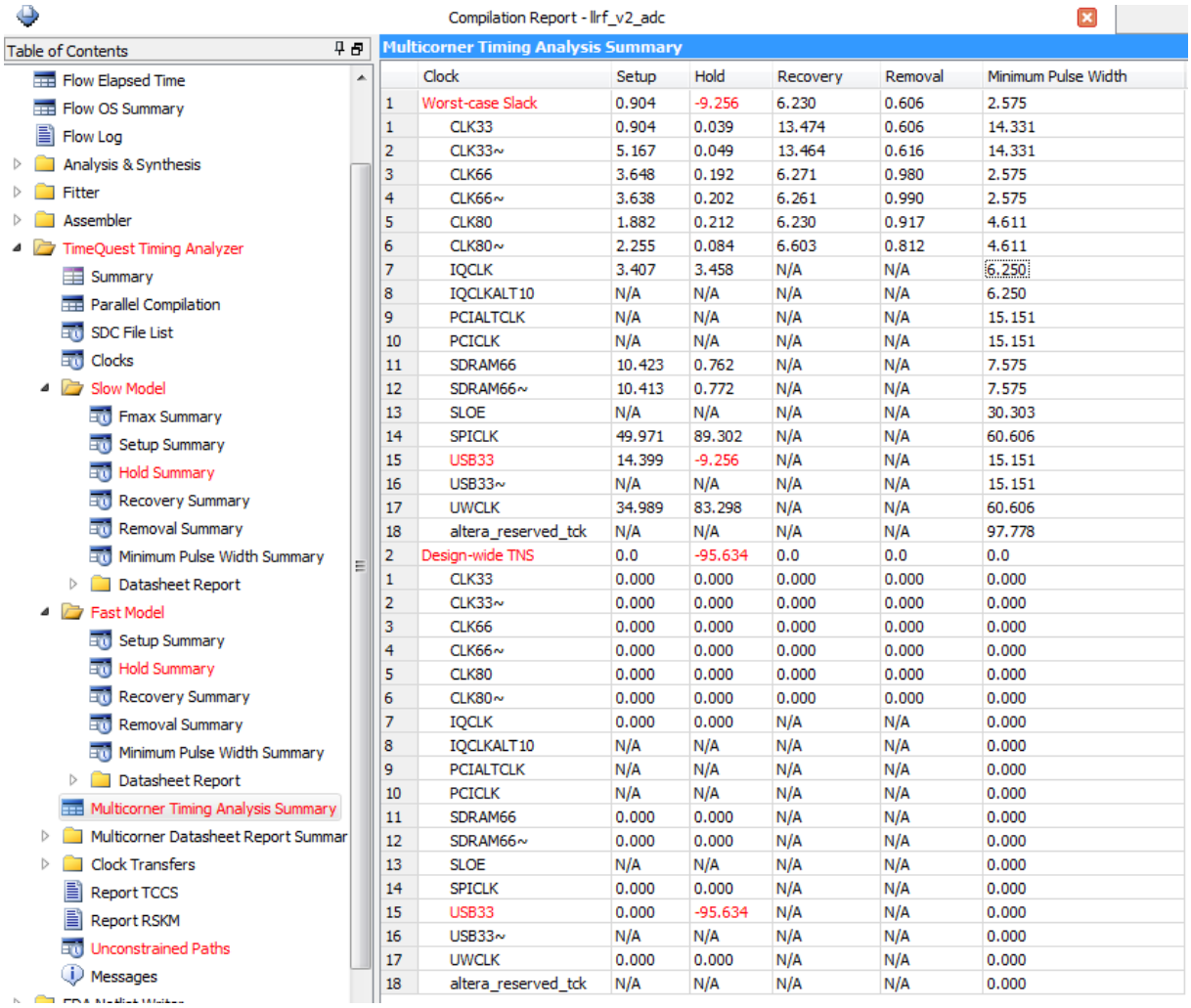
Le rôle des 3 entrées d'ADC A, B et C est défini dans le fichier *llrf_v2_config.vhd*, présenté ci-dessous, qui définit le package VHDL llrf_v2_config. Les éléments surlignés en jaune peuvent être modifiées pour obtenir la configuration souhaitée.

```
------------------------------------------------------------------
--   Title     :  llrf_v2_config
--   Note      :  MEZ2 specific
------------------------------------------------------------------

library ieee;
use     ieee.std_logic_1164.all;

package llrf_v2_config is

  constant A_LINK : integer:=0;
  constant B_LINK : integer:=1;
  constant C_LINK : integer:=2;

  --
  -- MAPPING
  --
  constant TRANS_LINK : integer:=B_LINK;
  constant REFC_LINK  : integer:=C_LINK;
  constant AUXIN_LINK : integer:=A_LINK;

  --
  -- ADC FIFO Config
  --
  constant A_LARGEFIFO : boolean:=TRUE;
  constant B_LARGEFIFO : boolean:=FALSE;
  constant C_LARGEFIFO : boolean:=FALSE;


end llrf_v2_config;
```

Figure 11 : fichier llrf_v2_config.vhd

De la même façon, les constantes x_LARGEFIFO (surlignage bleu) peuvent être modifiées pour définir quel ADC recevra une FIFO de 4096 mots et les autres 1024 mots seulement. Le FPGA n'a de mémoire disponible que pour une seule des trois voies avec une FIFO de 4096 mots. Par conséquent, assurez-vous de n'avoir qu'une constante x_LARGEFIFO à TRUE.

Après compilation, l'analyseur de timing TimeQuest ne doit donner aucune violation à part éventuellement des violations de Hold pour l'horloge USB33. La Figure 12 donne une idée du rapport d'analyse que fournira Timequest. Si d'autres lignes que celles correspondant à USB33 apparaissent en rouge, le FPGA risque de ne pas fonctionner correctement. Quelques entrées-sorties sont également non contraintes mais cela n'influe pas sur la bonne marche de l'application.



Multicorner Timing Analysis Summary

	Clock	Setup	Hold	Recovery	Removal	Minimum Pulse Width
1	Worst-case Slack	0.904	-9.256	6.230	0.606	2.575
1	CLK33	0.904	0.039	13.474	0.606	14.331
2	CLK33~	5.167	0.049	13.464	0.616	14.331
3	CLK66	3.648	0.192	6.271	0.980	2.575
4	CLK66~	3.638	0.202	6.261	0.990	2.575
5	CLK80	1.882	0.212	6.230	0.917	4.611
6	CLK80~	2.255	0.084	6.603	0.812	4.611
7	IQCLK	3.407	3.458	N/A	N/A	6.250
8	IQCLKALT10	N/A	N/A	N/A	N/A	6.250
9	PCIALTCLK	N/A	N/A	N/A	N/A	15.151
10	PCICLK	N/A	N/A	N/A	N/A	15.151
11	SDRAM66	10.423	0.762	N/A	N/A	7.575
12	SDRAM66~	10.413	0.772	N/A	N/A	7.575
13	SLOE	N/A	N/A	N/A	N/A	30.303
14	SPICLK	49.971	89.302	N/A	N/A	60.606
15	USB33	14.399	-9.256	N/A	N/A	15.151
16	USB33~	N/A	N/A	N/A	N/A	15.151
17	UWCLK	34.989	83.298	N/A	N/A	60.606
18	altera_reserved_tck	N/A	N/A	N/A	N/A	97.778
2	Design-wide TNS	0.0	-95.634	0.0	0.0	0.0
1	CLK33	0.000	0.000	0.000	0.000	0.000
2	CLK33~	0.000	0.000	0.000	0.000	0.000
3	CLK66	0.000	0.000	0.000	0.000	0.000
4	CLK66~	0.000	0.000	0.000	0.000	0.000
5	CLK80	0.000	0.000	0.000	0.000	0.000
6	CLK80~	0.000	0.000	0.000	0.000	0.000
7	IQCLK	0.000	0.000	N/A	N/A	0.000
8	IQCLKALT10	N/A	N/A	N/A	N/A	0.000
9	PCIALTCLK	N/A	N/A	N/A	N/A	0.000
10	PCICLK	N/A	N/A	N/A	N/A	0.000
11	SDRAM66	0.000	0.000	N/A	N/A	0.000
12	SDRAM66~	0.000	0.000	N/A	N/A	0.000
13	SLOE	N/A	N/A	N/A	N/A	0.000
14	SPICLK	0.000	0.000	N/A	N/A	0.000
15	USB33	0.000	-95.634	N/A	N/A	0.000
16	USB33~	N/A	N/A	N/A	N/A	0.000
17	UWCLK	0.000	0.000	N/A	N/A	0.000
18	altera_reserved_tck	N/A	N/A	N/A	N/A	0.000

Figure 12: Rapport d'analyse de TimeQuest

4. Programme du processeur embarqué NIOS II

Le programme destiné à être exécuté par le processeur du système SOPC est conçu avec l'environnement logiciel « Nios II Software Build Tools for Eclipse » (Nios II SBT) inclus par défaut dans l'installation de Quartus II 12.1. Il et, normalement, accessible dans le menu Windows Démarrer => <répertoire d'installation de la chaîne Quartus II 12.1> => Nios II EDS 12.1sp1 => Nios II 12.1sp1 Software Build Tools for Eclipse.

Les programmes NIOS II du projet LLRF_V2 sont présents dans le dossier *<base de l'archive>/stratix2s30_mez2_quartus12.1/software*

Pour visualiser, modifier ces programmes ou en créer d'autres :

- Lancer Nios II SBT
- Menu « File/Switch Workspace/Others... » et sélectionner le dossier *<base de l'archive>/workspace 12.1*
- Si besoin, changer la perspective en « NIOS II » en choisissant le menu « Window/Open Perspective/Other » puis « Nios II » et OK

Dans la fenêtre de gauche « Project Explorer », les dossiers entre autres LLRF_V2_MEZ2_USB et LLRF_V2_TEST doivent apparaître. Le dossier LLRF_V2_MEZ2_USB est la librairie système également appelée BSP (Board Support Package) de la carte LLRF_V2, équipée de son processeur embarqué NIOS dans le FPGA LLRF_V2. C'est en quelque sorte le *kernel* qui contient toutes les informations hardware du système embarqué : liste des différents périphériques auxquels peut accéder le processeur, adresses de bases, librairies incluses, etc... Il est fabriqué par Nios II SBT à partir du fichier *sopc_llrf_v2_usb.sopc* du système SOPC. Quant au dossier

LLRF_V2_TEST, il stocke une application qui peut être téléchargée dans la mémoire de la carte LLRF_V2 et exécutée par le processeur NIOS.

L'exécutable par défaut présent dans la ROM Flash externe S29GL064A90 de la carte LLRF_V2 est décrit dans le projet LLRF_V2_TEST. A l'allumage de la carte, et après configuration du FPGA LLRF_V2, le processeur recopie le programme présent dans la ROM vers la RAM externe MT48LC4M32 de la carte. Enfin, il exécute le continu de la routine *main()*. La Figure 13 présente le programme principal de l'application.

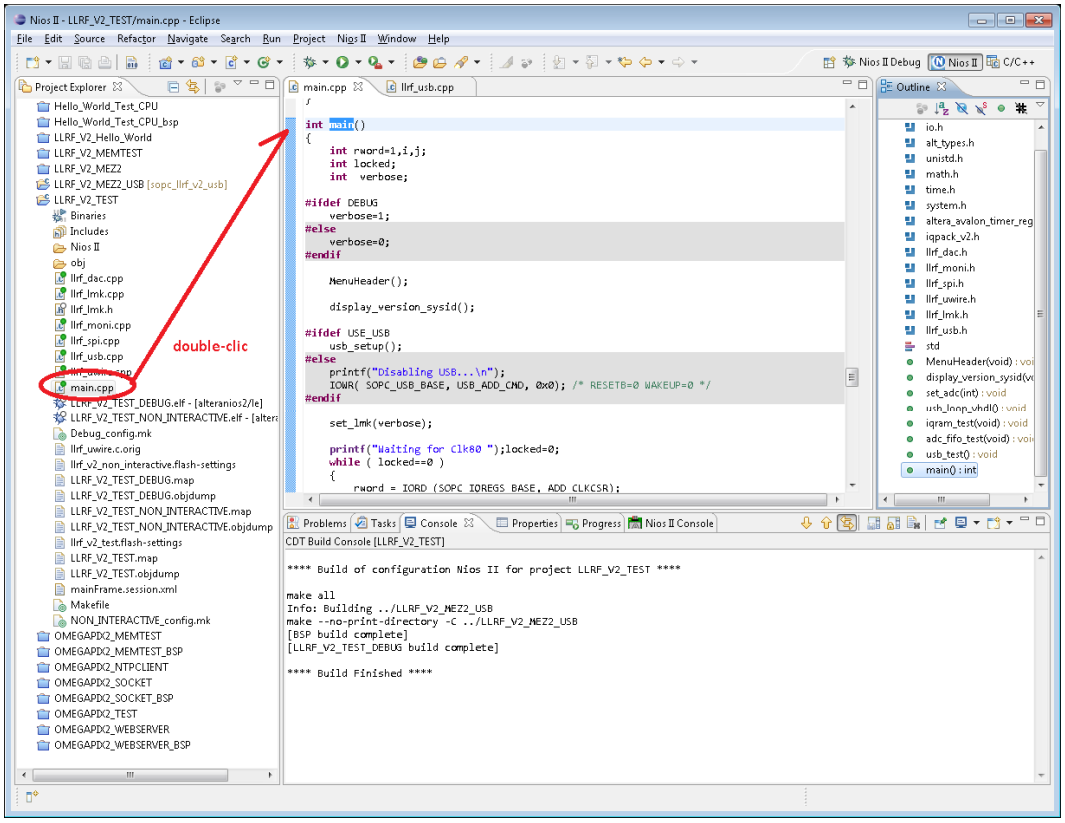


Figure 13 : fichier main.cpp du projet LLRF_V2_TEST

L'application commence par configurer d'abord le contrôleur USB (routine *set_usb()* décrite dans le fichier *llrf_usb.cpp*), ensuite PLL LMK (routine *set_lmk()* décrite dans le fichier *lmk.cpp*) puis attend que la PLL IQ du FPGA soit verrouillée sur le 80 MHz fourni par la PLL. Enfin, les ADC sont configurés (routine *set_adc()*).

Il est recommandé d'effectuer les tutoriaux disponibles sur le site Altera pour se familiariser avec l'environnement de développement NIOS. Ensuite, le plus simple est de parcourir les différents fichiers de code de l'application LLRF_V2_TEST pour comprendre ce que fait le programme.

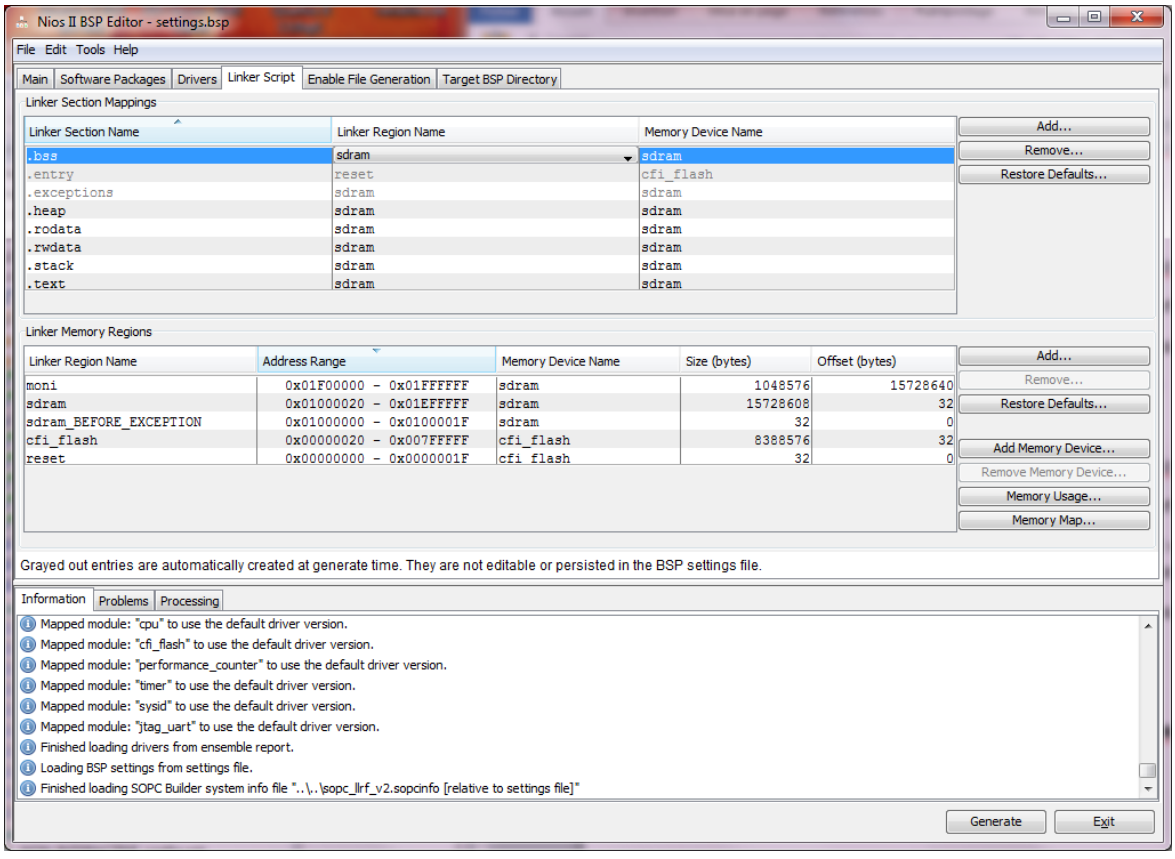
Les commandes pour accéder aux ressources du bus Avalon sont IOWR et IORD. Par exemple, voici une partie de code permettant de lire le registre VERSION du système puis d'écrire le registre TEST_PCI:

```
#include <io.h>
#include "include.h"

int version;

version = IORD(IQ_REGS_BASE, 0x0401);
IOWR(IQ_REGS_BASE, 0x0400, 0xdeadbeef );
```

Le fichier *system.h* est généré automatiquement à la création du BSP de la carte et contient la valeur de IQ_REGS_BASE. Le second paramètre de IOWR et IORD est un offset du mot à accéder par rapport à l'adresse de base du périphérique ; c'est le nombre Local Add, première colonne de la liste des registres donné en annexe III. Le fichier *iqpack_v2.h*, dérivé du fichier VHDL *iqpack_v2_mez2.vhd* contient les valeurs des offsets de tous les registres des périphériques du système SOPC. Ce fichier, contrairement à *system.h*, n'est pas mis à jour automatiquement, les changements doivent être gérés par l'utilisateur.

⇨ Comme expliqué en section « Fonctionnement du monitoring », les blocs de monitoring sont stockés, par défaut, dans la SDRAM, aux adresses $01F0 0000 à $01FF FFFF. Or, par défaut, le compilateur d'exécutable utilisera le haut de la SDRAM pour stocker les pointeurs de pile du processeur et son *heap*. La conséquence sera une interférence entre le module de monitoring IQ monitoring et le processeur qui empêchera le bon fonctionnement des deux entités. Pour éviter ce problème, il faut définir, dans les paramètres du linker de la libraire système utilisée (BSP) que l'espace mémoire de monitoring n'est pas utilisable pour le processeur. Pour cela, dans le cas d'une création d'un nouveau BSP :

- Sélectionner dans la fenêtre de gauche « Project Explorer » le BSP (ici, LLRF_V2_MEZ2_USB) puis clic-droit « Nios II/BSP Editor... »
- dans la fenêtre qui s'ouvre « Nios II BSP Editor », sélectionner l'onglet « Linker Script »
- Dans la section « Linker Memory Regions », sélectionner la région sdram puis « Remove... »
- Puis cliquer sur le bouton « Add... », définir une nouvelle zone avec les paramètres suivants puis cliquer Add :
  - Region Name sdram
  - Region Size 15728608
  - Memory Device sdram
  - Memory Offset 32
- Créer ensuite une seconde zone, nommée moni, avec les paramètres :
  - Region Name moni
  - Region Size 1048576
  - Memory Device sdram
  - Memory Offset 15728640
- Dans la fenêtre du haut « Linker Section Name », affecter la deuxième colonne « Linker Region Name » de toutes lignes disponibles à « sdram »
- Le résultat doit être le suivant :



Une zone « moni » a ainsi été créée spécialement pour le monitoring. Le linker n'utilisera cette zone pour aucun type de donnée qu'il destine à la RAM. Notamment il placera les sections heap et stack en haut de la zone sdram, désormais déclarée à 0x01EF FFFF en interdisant l'espace de monitoring.

- Cliquer sur « Generate » pour régénérer la librairie système ; accepter de sauvegarder les nouveaux paramètres.
- Cette opération n'est à réaliser qu'une seule fois, à la création d'une nouvelle librairie BSP. Une fois le réglage effectué, il est fortement déconseillé de modifier la valeur par défaut du registre MON_RAMBASE.

Le programme *main()* utilise les flux par défaut stdin, stdout et stderr qui sont redirigés vers le périphérique SOPC JTAG_UART. Pour l'utiliser, avec un USB Blaster connecté au port JTAG de la carte en face_avant :

- Lancer l'application « Nios II 12.1sp1 Command Shell » accessible au même endroit que Nios II SBT.
- taper `nios2-terminal` dans la fenêtre. Ce qui est tapé au clavier est dirigé vers le stdin de l'application embarquée, les sorties stdout et le stderr sont dirigées vers la fenêtre.

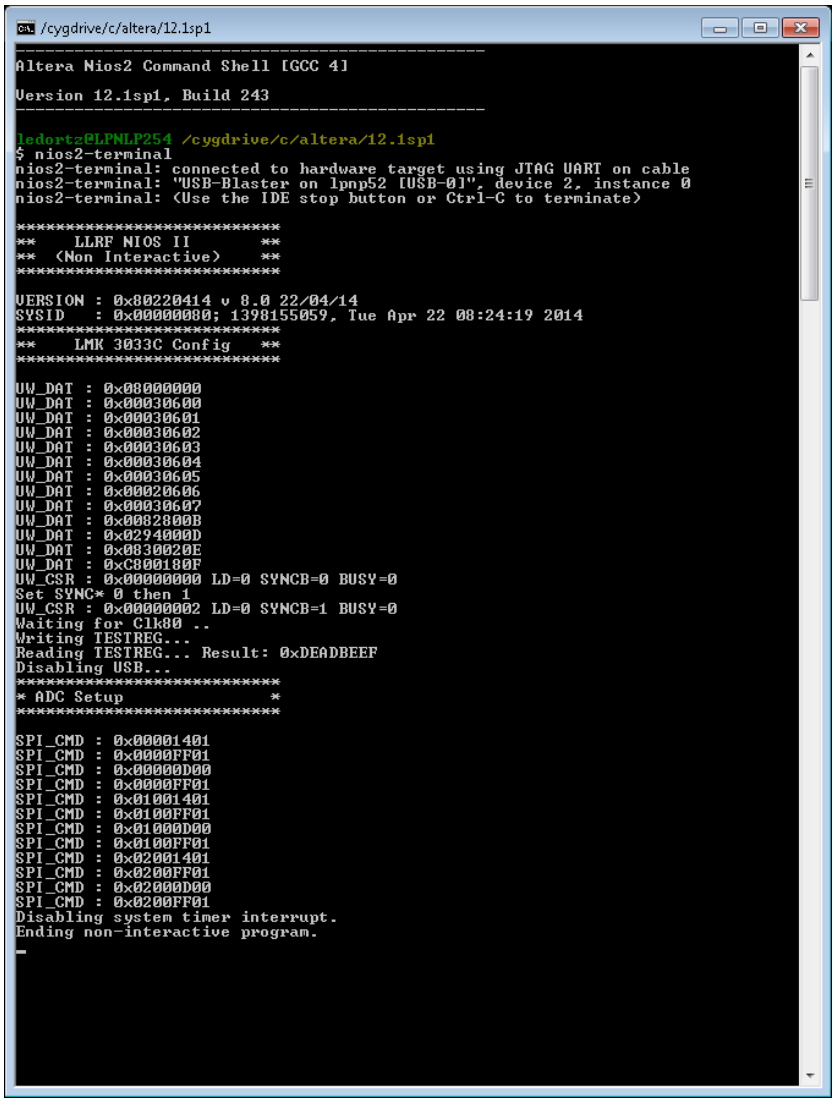


Figure 14 : Utilisation de l'UART JTAG. Programme résidant en PROM

Le programme résidant dans la ROM Flash est une version non-interactive de LLRF_V2_TEST. Sélectionner le projet LLRF_V2_TEST dans le « Project Explorer » puis clic-droit « Properties ». Sélectionner « Nios II Application Properties » à gauche, attendre quelques secondes que la fenêtre se mette à jour. Dans le champ Configuration, il existe deux configurations distinctes :

- NON_INTERACTIVE pour laquelle le symbole `NON_INTERACTIVE` sera défini (cf. ligne Defined Symbols)
- DEBUG pour laquelle le symbole `DEBUG` sera défini

Les codes sources du programme utilisent les directives de compilation `#ifdef NON_INTERACTIVE` ou `#ifdef DEBUG` pour modifier l'exécution du programme selon la version utilisée. La version non-interactive est prévue pour ne pas attendre d'interaction de la part de l'utilisateur qui pourrait être bloquante.

Un autre symbole est défini dans l'application: USE_USB. Pour désactiver la gestion du contrôleur USB, il suffit de supprimer l'option -DUSE_USB dans la ligne Defined Symbols.

Remarque

Le BSP doit être régénéré à chaque fois que le système SOPC a été retouché dans SOPC builder. Dans le cas où le projet FPGA a été recompilé (avec la création d'un nouveau fichier .*sof*) mais que le fichier sopc n'a pas été touché, il est inutile de reconstruire le BSP.

Le système SOPC intègre un périphérique SYS_ID, qui donne un identifiant unique codant le temps universel de génération du système SOPC (*timestamp*). Avant de télécharger par JTAG un code exécutable dans le FPGA ou dans la ROM flash, Nios II SBT vérifie que les identifiants SYS_ID du FPGA connecté par JTAG sont conformes à ceux codés dans le BSP de l'exécutable. Il arrive qu'il y ait des décalages entre les 2 identifiants (de quelques minutes parfois). Pour contourner ce problème, il est possible de forcer le téléchargement du code vers la cible en ignorant le SYS_ID et le timestamp. Une autre piste est de faire un nettoyage du projet et de sa libraire système : clic-droit « Build Clean » sur le projet et/ou le BSP.

5. Notes de Version

Mars-Avril 2014 :

- Version 8.0 : refonte du projet en version 12.1 de la chaine logicielle Quartus

Juin 2014 :

- Version 8.1 : Révision llrf_v2_usb. Interface USB finalisée

6. Problèmes connus

a. Mécanique

- Les connecteurs d'entrée-sortie MMCX sont trop rentrés dans la carte. Il est difficile d'insérer les câbles. Il aurait fallu, sur le PCB, déplacer les connecteurs vers le bord gauche de quelques millimètres.
- Le câble standard de l'USB blaster s'étend vers la droite du connecteur JTAG (vu de face), rendant l'accès aux autres entrées difficiles.

b. Interface CPCI

- Les accès en mode rafale ne fonctionnent pas correctement sur les espaces mémoire I/Q OUT_RAM. (risque de plantage du châssis). Le logiciel labview en tient compte et fait des accès mot par mot.

c. LMK03033C

- Les sorties horloge LVDS de la PLL LMK03033C ne fonctionnent pas correctement lorsque le récepteur qui leur est relié est de type « fail-safe » (comme le MAX9113). Une intervention sur la carte est indispensable pour un bon fonctionnement (passage en couplage AC ou utilisation d'un récepteur LVDS sans «fail-safe »)

d. Quartus SOPC Builder

- Différences System ID cible FPGA⇔logiciel Nios

7. Références

[1] Schémas de la carte ASSERVNUM_V2
https://edms.in2p3.fr/edms/doc.info? document_id=I-029862
https://edms.in2p3.fr/edms/doc.info? document_id=I-021366

[2] Firmwares du FPGA IntCPCI de la carte LLRF_V2
https://edms.in2p3.fr/edms/doc.info? document_id=I-037589

[3] Schémas de la carte fille MEZZANINE_V2
https://edms.in2p3.fr/edms/doc.info? document_id=I-019569

[4] LMK03000 Family Precision Clock Conditioner with Integrated VCO
http://www.ti.com/product/lmk03033

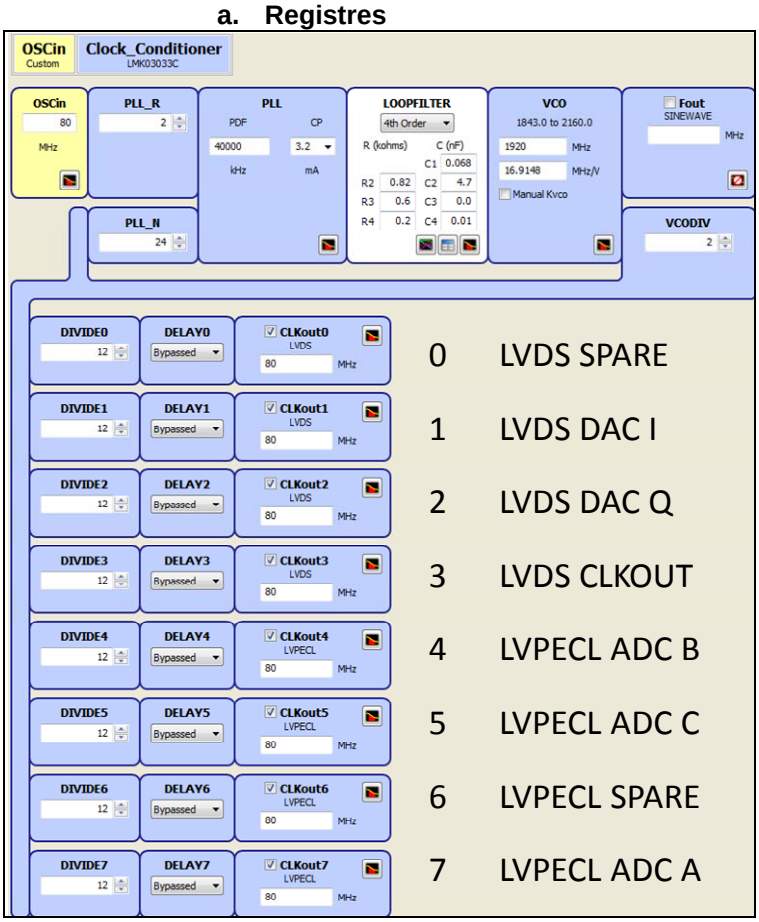
[5] FPGA LLRF_V2
https://edms.in2p3.fr/edms/doc.info?document_id=I-037526

[6] Composant Cypress CY7C68001 EZ-USB SX2
http://www.cypress.com/?id=4242

8. Annexe I: Configuration de la PLL LMK03033C

L'application « Clock Design Tools » de Texas Instruments propose, pour piloter la PLL LMK03033C dans le contexte de la carte mezzanine v2, la configuration suivante :

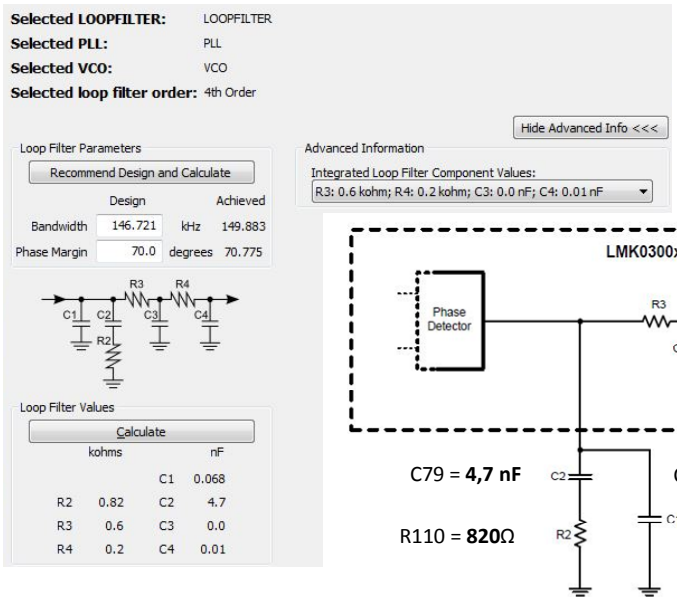
a. Registres



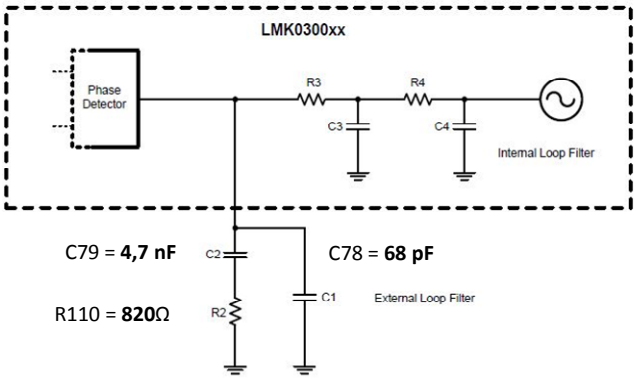
Write Data	Reg	Description
8000 0000	R0	Reset
-		Wait ~1 us
0002 0600	R0	CLK0 | EN=0 | DIVIDED=12
0003 0601	R1	CLK1 | EN=1 | DIVIDED=12
0003 0602	R2	CLK2 | EN=1 | DIVIDED=12
0003 0603	R3	CLK3 | EN=1 | DIVIDED=12
0003 0604	R4	CLK4 | EN=1 | DIVIDED=12
0003 0605	R5	CLK5 | EN=1 | DIVIDED=12
0002 0606	R6	CLK6 | EN=0 | DIVIDED=12
0003 0607	R7	CLK7 | EN=1 | DIVIDED=12
0082 800B	R11	DIV4 = 1 (Phase Det F >20MHz)
0294 000D	R13	OSCin_Freq=80M | Default LF
0830 020E	R14	FOUT=0 | EN_CLK=1 | PWR=0 | PLL_MUX=3 | PLL_R=2
C800 180F	R15	PLL_N=24 | VCODIV=2 | CP=32x + Start Calibration Routine

b. Paramètres du filtre de boucle

Les valeurs des résistances et capacités externes du filtre de boucle sont également à ajuster sur la carte, selon la configuration suivante (R110, C78 et C79):



C3 = 0
C4 = 10 pF
R3 = 600Ω
R4 = 200Ω
} Default Config



c. Programme de démarrage du processeur NIOS

```
void set_lmk(int verbose)
{
{...}
    write_uwire( 0x8000000 ); /* R0 RESET                */
    usleep(1);

    // RX X=0 to 7
    // CLKOutX_Mux = 1 CLKOut_DIV=6 DELAY=0
    // CLKOUTX Disabled => 0x0002060X
    //         Enabled  => 0x0003060X
    //
    write_uwire( 0x00020600 ); /* R0  LVDS   ALTERA (OFF) */
    write_uwire( 0x00030601 ); /* R1  LVDS   I DAC        */
    write_uwire( 0x00030602 ); /* R2  LVDS   Q DAC        */
    write_uwire( 0x00030603 ); /* R3  LVDS   to Mother    */
    write_uwire( 0x00030604 ); /* R4  LVPECL ADC_B        */
    write_uwire( 0x00030605 ); /* R5  LVPECL ADC_C        */
    write_uwire( 0x00020606 ); /* R6  LVPECL SPARE (OFF)  */
    write_uwire( 0x00030607 ); /* R7  LVPECL ADC_A        */

    write_uwire( 0x0082800B ); /* R11   DIV4              */
    write_uwire( 0x0294000D ); /* R13                     */
    write_uwire( 0x0830020E ); /* R14  FOUT=0             */
    write_uwire( 0xC800180F ); /* R15                     */

    wait_uwire_ready();
    print_uwire_csr();

    printf("Set SYNC* 0 then 1\n");

    IOWR(SOPC_UW_BASE, UW_ADD_CSR,0);
    IOWR(SOPC_UW_BASE, UW_ADD_CSR,2);
{...}
}

alt_8 wait_uwire_ready()
{
    alt_u32 i=0;

    while( (IORD(SOPC_UW_BASE, UW_ADD_CSR) & UW_IS_BUSY)!=0 )
    {
       usleep(1);
        if ( i++ >= 20 ) return -1;
    }
    return 0;
}

void write_uwire(alt_u32 dat)
{

    if( wait_uwire_ready() == 0 )
    {
        #ifdef DEBUG
        printf( "UW_DAT : 0x%08X\n", dat);
        #endif
        IOWR(SOPC_UW_BASE, UW_ADD_DAT, dat);
    }
}
```

⇨ A chaque fois que la PLL se déverrouille ou est reconfigurée, elle doit subir une étape de resynchronisation. Pour cela, il faut forcer le signal SYNCB* (contrôlable en agissant sur le bit 1 du registre SOPC UW_CSR) à 0 puis le replacer à 1.

9. Annexe II: Interface USB

Le composant USB Cypress CY7C68001 (appelé également SX2) est accessible, via le FPGA principal, grâce à deux registres de l'espace SOPC, USB_CMD et USB_DAT (cf annexe III), les autres registres n'étant créés qu'à des fins de débogage. Pour pouvoir fonctionner correctement, le SX2 doit tout d'abord être configuré par un maître externe (*external master* tel que décrit en [6]) qui sera, dans le contexte de la carte LLRF, n'importe quel maître du système SOPC et, en pratique, le processeur Nios II.

Du point de vue firmware, c'est le module USB_SX2 du FPGA principal, décrit en VHDL, qui prend en charge la communication d'une part avec le composant physique SX2 et, d'autre part, avec le système SOPC par l'intermédiaire d'une interface esclave (permettant de configurer le SX2) et d'une interface maître SOPC (exploitée en mode pont USB-Bus Avalon).

a. Paramétrage du composant SX2

Le SX2 présente deux interfaces avec le maître externe : une interface FIFO qui permet d'accéder aux entrées et sorties des 4 endpoints gérés par le composant (EP2, 4, 6 et 8) et une interface de commande qui permet d'écrire ou de lire les registres internes du composant. Le choix de l'accès à une des FIFOs ou à l'interface commande s'effectue en agissant sur les valeurs des pattes d'adresses FIFOADR du composant. Ces signaux sont directement accessibles en écrivant les bits [10:8] du registre USB_CMD. Une fois les signaux FIFOADR positionnés, une lecture ou une écriture sur le registre USB_DAT effectuera un cycle d'écriture ou de lecture sur le SX2 (en agissant sur les lignes SLRD, SLWR et SLOE et sur le bus de données FD[15:0]) sur l'interface correspondante.

La routine *usb_setup()* (fichier *llrf_usb.cpp*) du programme résidant configure le composant SX2 en réalisant les opérations suivantes :

- Réinitialisation : écriture de USB_CMD[0] à 0 puis 1, qui agit directement sur la patte RESETB du composant.
- Ecriture du descripteur : le descripteur USB est écrit par l'intermédiaire d'accès en écriture au registre de commande DESC (0x30) du SX2. Y est défini notamment Vendor_ID, Device_ID du périphérique USB ainsi que la configuration des endpoints. Le composant SX2 n'est relié à aucune EEPROM I^2C sur la carte LLRF_V2 ; son descripteur doit donc obligatoirement être écrit par le maître externe.
- Polarité et type d'interface : Registre IFCONFIG (0x01). Réglage de l'interface FIFO en mode synchrone, polarités par défaut (actif niveau bas), et horloge IFCLK fournie par le FPGA, de fréquence 33 MHz dérivée de l'horloge PCI.
- Réinitialisation des FIFOS : Ecriture du registre INPKTEND/FLUSH (0x20) pour vider les FIFOs.
- Fonction des flags : Ecriture de FLAGSAB et FLAGSCD (0x02 et 0x03) pour régler les fonctions des pattes FLAG* ainsi :
  - FLAGA = EP2 Empty*
  - FLAGB = EP6 Full*
  - FLAGC = EP6 Empty*
  - FLAGD = ChipSelect*
- Masquage et initialisation des interruptions : Registre INTENABLE (0x2E) réglé pour que le SX2 ne génère aucune interruption. Ensuite, lecture du composant tant que la ligne INT# est active, afin d'effacer la pile des interruptions encore enregistrées.
- Configuration Endpoints : Registre EP6PKTLENH (0x0E). EP6 Mode WORDWIDE, inhibition de la génération de paquets vides.

Une fois ces opérations effectuées, la connexion USB sera opérationnelle dès lors qu'un PC hôte sera relié à la carte par un câble USB et que l'étape d'énumération USB aura eu lieu. Le PC pourra écrire des octets de données via l'endpoint 2 qui rempliront, du côté SX2, la FIFO de réception EP2 en attente de relecture par le FPGA (par mot de 16 bits en mode WORDWIDE); le FPGA pourra, à l'inverse, écrire des données dans la FIFO d'émission EP6 (par mot de 16 bits en mode WORDWIDE) que le PC pourra récupérer en effectuant une lecture du endpoint 6.

b. Pont USB ⇨Bus Avalon

Lorsque le SX2 est configuré, le module VHDL USB_SX2 réalise la fonction de pont USB-Bus Avalon dès que le signal MasterEnable (USB_CMD[31]) est écrit à 1. Un protocole de communication utilisateur PC⇔Carte a été défini pour permettre d'envoyer, du PC hôte, des données à écrire vers n'importe quel espace mémoire du système SOPC et de retourner vers le PC des données relues du système SOPC.

Le protocole est basé sur des mots de 16 bits par simplicité, car le composant SX2 est configuré en accès aux FIFO EP2 et EP6 en mode WORDWIDE. Seuls des mots de 32 bits peuvent être écrits ou lus sur le bus Avalon.

Bytes	Value	Name	Description
1-0	$AAAA	SOP	Start of Packet
3-2	$0000	WNI	Write 32b, non-incrementing address
	$0004	WI	Write 32b, incrementing address
	$0010	RNI	Read 32b, non-incrementing address
	$0014	RI	Read 32b, incrementing address
	$007F	NOP	No avalon transaction
5-4		*Size*	N=Number of words to write or read
7-6		ADDRL	LSB of the 32-bit address of the avalon target
9-8		ADDRH	MSB of the 32-bit address of the avalon target
11-10		*Data0L*	LSB of the first data word
13-12		*Data0H*	MSB of the first data word
...		...	...
		Data(n-1)L	LSB of the last data word
		Data(n-1)H	MSB of the last data word
k+1-k	$5555	EOP	End of Packet

Tableau 8 : Protocole USB-Bus Avalon

Les mots de 16 bits sont envoyés dans EP2 et EP6 octet par octet, les LSB d'abord. Dans le cas d'écriture ou de lecture avec incrémentation d'adresse, l'adresse est incrémentée de 4 après chaque accès.

Exemples :

1. Ecriture des mots de 32 bits $deadbeef à l'adresse SOPC $02001000 et $12345678 en $02001004:
   Le PC hôte écrit en EP2 les octets :

```
AA, AA (SOP)
04, 00 (WI)
02, 00 (Size = 2 mots)
00, 10, 00, 02 ($02001000, LSB d'abord)
ef, be, ad, de ($deadbeef, LSB d'abord)
78, 56, 34, 12 ($12345678, LSB d'abord)
55, 55 (EOP)
```

   Le SX2 ne répond rien en EP6

2. Lecture de 5 mots de 32 bits à l'adresse SOPC $02001000 sans incrémenter l'adresse :
   Le PC hôte écrit en EP2 les octets :

```
AA, AA (SOP)
10, 00 (RNI)
05, 00 (Size = 5 mots)
00, 10, 00, 02 ($02001000, LSB d'abord)
55, 55 (EOP)
```

   Le SX2 renvoie sur EP6, en lisant chaque mot sur le bus avalon en $02001000:

```
AA, AA (SOP)
10, 00 (RNI)
05, 00 (Size = 5 mots)
00, 10, 00, 02 ($02001000, LSB d'abord)
5 x [d0, d1, d2, d3] (5 mots de 32 bits LSB d'abord, lus en $02001000)
55, 55 (EOP)
```

Le décodage des commandes émises par le PC, les accès en écriture ou lecture du bus avalon et l'encodage des réponses est réalisé de façon automatique par le module USB_SX2 si MasterEnable=1. C'est le cas pour le programme Nios résidant dans la ROM flash et exécuté à l'allumage de la carte.

Néanmoins, la gestion du protocole utilisateur pourrait également être effectuée en logiciel, au détriment d'une baisse des performances et du débit net de données transférées; un exemple est montré dans la version DEBUG du programme LLRF_V2_TEST (routine *usb_soft_loop()* du fichier *main.cpp*).

10. Annexe III: Registres du FPGA LLRF_V2

IQ Registers / SOPC ADDRESS = $0200 0000 - $0200 1FFF						
Local Add	**BAR1 Offset**	**Name**	**Bits**	**Mode**	**Description**	
0000	0000	TEST_REG	[31 :0]	R/W	Test register	
0001	0004	PID_P_TI	[31:0]	R/W	PID P parameter for V_{t_I}	+1 = 00010000 -1 = FFFF0000
0002	0008	PID_I_TI	[31:0]	R/W	PID I parameter for V_{t_I}	+1 = 00010000 -1 = FFFF0000
0003	000C	PID_D_TI	[15:0]	R/W	PID D parameter for V_{t_I}	+1 = 7FFF -1 = 8000
0004	0010	PID_C_TI	[31:0]	R/W	PID constant for V_{t_I}	+1 = 7FFFFFFF -1 = 80000000
0005	0014	PID_P_TQ	[31:0]	R/W	PID P parameter for V_{t_Q}	+1 = 00010000 -1 = FFFF0000
0006	0018	PID_I_TQ	[31:0]	R/W	PID I parameter for V_{t_Q}	+1 = 00010000 -1 = FFFF0000
0007	001C	PID_D_TQ	[15:0]	R/W	PID D parameter for V_{t_Q}	+1 = 7FFF -1 = 8000
0008	0020	PID_C_TQ	[31:0]	R/W	PID constant for V_{t_Q}	+1 = 7FFFFFFF -1 = 80000000
0010	0040	PHI_A	[15:0] [31:16]	R/W	$A_A \times \cos \varphi_A$ A_ADC $A_A \times \sin \varphi_A$ A_ADC	+1 = 7FFF -1 = 8000
0011	0044	PHI2_A	[15:0] [31:16]	R/W	$A_A \times \cos(\varphi_A + \pi/4)$ A_ADC $A_A \times \sin(\varphi_A + \pi/4)$ A_ADC	
0012	0048	PHI_B	[15:0] [31:16]	R/W	$A_B \times \cos \varphi_B$ B_ADC $A_B \times \sin \varphi_B$ B_ADC	
0013	004C	PHI2_B	[15:0] [31:16]	R/W	$A_B \times \cos(\varphi_B + \pi/4)$ B_ADC $A_B \times \sin(\varphi_B + \pi/4)$ B_ADC	
0014	0050	PHI_C	[15:0] [31:16]	R/W	$A_C \times \cos \varphi_C$ C_ADC $A_C \times \sin \varphi_C$ C_ADC	
0015	0054	PHI2_C	[15:0] [31:16]	R/W	$A_C \times \cos(\varphi_C + \pi/4)$ C_ADC $A_C \times \sin(\varphi_C + \pi/4)$ C_ADC	
0020	0080	OFF_A	[13:0] [16] [17]	R/W	AOff[13:0] UseAOff InvA	+1 = 1FFF -1 = 2000
0021	0084	OFF_B	[13:0] [16] [17]	R/W	BOff[13:0] UseBOff InvB	
0022	0088	OFF_C	[13:0] [16] [17]	R/W	COff[13:0] UseCOff InvC	
0025	0090	INT_TI	[15:0]	R	PID integral for V_{t_I}	
0026	0094	INT_TQ	[15:0]	R	PID integral for V_{t_Q}	
0030	00C0	MON_TIME	[31:0]	R	Time counter (period 12.5 ns)	
0031	00C4	MON_INCI	[15:0] [31:16]	R	Inci I MSB Inci Q MSB	
0032	00C8	MON_REFL	[15:0] [31:16]	R	Refl I MSB Refl Q MSB	
0033	00CC	MON_TRAN	[15:0] [31:16]	R	Tran I MSB Tran Q MSB	

Local Add	BAR1 Offset	Name	Bits	Mode	Description
0034	00D0	MON_REFC	[15:0] [31:16]	R	Refc I MSB Refc Q MSB
0035	00D4	MON_AUX	[15:0] [31:16]	R	Aux I MSB Aux Q MSB
0036	00D8	MON_OUT	[13:0] [29:16]	R	Iout Qout
0037	00DC	MON_ERR	[15:0] [31:16]	R	Ierr Qerr
0040	0100	MON_PER	[15:0]	R/W	MonPer[15:0] Monitoring Sampling period
0041	0104	MON_PREPOST	[13:0] [29:16]	R/W	MonPre[13:0] Monitoring Pre Trigger MonPost[13:0] Monitoring Post Trigger
0045	0114	MON_RAMCTL	[7:0] [15:8] [16] [17] [18] [24]	R/W R/W R/W R R R/W	MonStartCond cf text MonStopCond cf text MonClear MonWrite MonReadyRead TestFifo Default = 0
0046	0118	MON_REGCTL	[3:0] [11:8] [16]	R/W R/W R	MonRegMask MonRegPat MonRegEnabled DigIn & RegMask = RegPat
0047	011C	MON_RAMBASE	[31:20]	R/W	SOPC base address of the monitoring RAM (Default = 01F00000)
0048	0120	MON_RAMRADD	[31:0]	R	Monitoring RAM read pointer offset in bytes
0049	0124	MON_RAMWADD	[31:0]	R	Monitoring RAM write pointer offset in bytes
0050	0140	DBG_TRANI	[15:0] [16]	R/W	DbgTranI Test value for TranI UseDbgTranI '1' = Forces the test value
0051	0144	DBG_TRANQ	[15:0] [16]	R/W	DbgTranQ Test value for TranQ UseDbgTranQ '1' = Forces the test value
0052	0148	DBG_OUT	[13:0] [15:14] [29:16] [31:30]	R/W	$Dbgl_{out}$[13:0] DbgXout :test value IoutSource or StopAdd if Ram $DbgQ_{out}$[13:0] XoutSource : QoutSource 0=Std, 1=Dbg, 2=Byp, 3=Ram
0060	0180	DIG_REG	[3:0] [7:4]	R R/W	DigIn[3 :0] Digital inputs DigOut[3 :0] Digital outputs
0070	01C0	CAV_CSR	[1 :0] [16] [24]	R R/W R/W	Stable Q | Stable I Run8_4 '1'= 8 samples/100ns Clear PID sum '1'= Clear
0071	01C4	STAB_I	[15:0] [31:16]	R/W	StabCptI[15:0] Stability threshold StabMaxI[15:0]
0072	01C8	STAB_Q	[15:0] [31:16]	R/W	StabCptQ[15:0] Stability threshold StabMaxQ[15:0]
0073	01CC	FIFO_CSR	[0] [12] [14:13] [16] [18:17] [20] [22:21]	R/W R/W R R/W R R/W R	Clear Write Enable (A) Full | Empty (A) Write Enable (B) Full | Empty (B) Write Enable (C) Full | Empty (C)
0074	01D0	FIFO_USEDA	[15:0]	R	Fifo Used words (A)
0075	01D4	FIFO_USEDB	[15:0]	R	Fifo Used words (B)
0076	01D8	FIFO_USEDC	[15:0]	R	Fifo Used words (C)
00C0	0300	IQ_SET	[15:0] [31:16]	R/W	Set point I_{set} Set point Q_{set}
400	1000	TEST_PCI	[31 :0]	R/W	Test register
401	1004	VERSION	[31:0]	R	Version code Hex value = Vvddmmyy ex 31281106 is version 3.1 date nov 28 2006

Local Add	BAR1 Offset	Name	Bits	Mode	Description
402	1008	CLK_CSR	[0] [1] [2] [3] [4] [5] [6] [8] [9] [12] [16:17]	R/W R R/W R R R R R/W R R/W R/W	IQPllReset IQPllLocked UseIQClkAlt ‘1’ = Use PXI 10MHz IsIQClkAlt IQClkLoss BadIQClk BadIQAlt IQPllReconf IQPllBusy Update Phi Ref10 Config 00 = ClkInt10 01 = Refc
403	100C	IQPLL_PARAM	[8:0] [15:12] [18:16] [24] [25] [26]	R/W R/W R/W W W R	DataOut/DataIn Counter Type Counter Param Write Request Read Request Busy
404	1010	SOPC_BASE	[31:16]	R/W	Base address of MMAP_SOPC ($yyyy0000)
405	1014	CPU_CSR	[0] [1]	R/W R	CpuResetRequest CpuResetTaken
406	1018	INPUT_CONF	[3:0] [7:4] [11:8]	R	TRAN_LINK REFC_LINK 0=A / 1=B / 2=C AUXIN_LINK
IQ Memories / SOPC ADDRESS = $0280 0000 - $0280 FFFF					
Local Add	**BAR0 Offset**	**Name**	**Bits**	**Mode**	**Description**
0000-03FF 0400-07FF 0800-0BFF 0C00-0FFF 1000-13FF	0000-0FFC 1000-1FFC 2000-2FFC 3000-3FFC 4000-4FFC	FIFO_DAT_A FIFO_DAT_B FIFO_DAT_C - -	[13:0] [14] [29 :16] [30] [31]	R	ADC content Out of Range bit FIFO used words (Channel) Empty Full
1400-17FF 1800-1BFF	5000-5FFC 6000-6FFC	IOUT_RAM QOUT_RAM	[13:0]	R/W	DAC output array when DBG_OUT set to RAM mode 256 values
Memory Mapped Window					
Local Add	**BAR0 Offset**	**Name**	**Bits**	**Mode**	**Description**
4000-7FFF	10000-1FFFC	MMAP_SOPC	[31:0]	R/W	Memory mapped internal SOPC content
Other Peripherals					
Local Add	**SOPC Address**	**Name**	**Bits**	**Mode**	**Description**
SPI and microWire Interface					
0000	0080 0000	SPI_CMD	[7:0] [20:8] [22:21] [23] [27:24] [28]	R/W R/W R/W R/W R/W R	SPI write data SPI address 00 = one byte data READ_write Destination slave (0=A, 1=B , 2=C) Busy
0001	0080 0004	SPI_RDAT	[9] [8] [7:0]	R R R	Busy New data (cleared when read) Data read back
0002	0080 0008	UW_CSR	[0] [1] [2]	R R/W R/W	UW_BUSY UW_SYNCB (Default =1) UW_LD (Lock Detect)
0003	0080 000C	UW_DAT	[3:0] [31:4]	R/W	uWire Write transaction Add[3:0] Data[27:0]

Local Add	SOPC Address	Name	Bits	Mode	Description
USB Slave Interface					
0000	0080 4000	USB_CMD	[0] [1] [3] [10:8] [12] [13] [16] [17] [18] [19] [20] [30:24] [31]	R/W R/W R/W R/W R R R R R R R/W R R/W	ResetB WakeUp (‘1’) PktEndB (‘1’) FifoAdr[2:0] Ready IntB FlagA=EP2 EmptyB FlagB=EP6 FullB FlagC=EP6 EmptyB FlagD_CSB LocCSB (0→ FlagD_CSB=0 else FlagD_CSB=Z) USBState Machine MasterEnable: USB to Avalon Bridge activation
0001	0080 4004	USB_DAT	[15:0] [16] [24] [25] [28] [29] [30] [31]	RW W R R R R R R	FD[15:0] Last (1=> PktEnd=0) IntB Ready FlagA=EP2 EmptyB FlagB=EP6 FullB FlagC=EP6 EmptyB FlagD_CSB
0002	0080 4008	USB_REC	[31:0]	R	Total received 16-bit words from EP2 Cleared when MasterEnable goes low
0003	0080 400C	USB_SENT	[31:0]	R	Total sent 16-bit words through EP6 Cleared when MasterEnable goes low
0004	0080 4010	USB_FRMCNT	[31:0]	R	nCycles spent in USB State Machine non IDLE state
0005	0080 4014	USB_TRTYPE	[31:0]	R	Last Master Transaction Type
0006	0080 4018	USB_TRADD	[31:0]	R	Last Master Transaction Address
0007	0080 401C	USB_TRDAT	[31:0]	R	Last Master Transaction Data
0008	0080 4020	USB_TRSIZE	[31:0]	R	Last Master Transaction Size (should be 0)