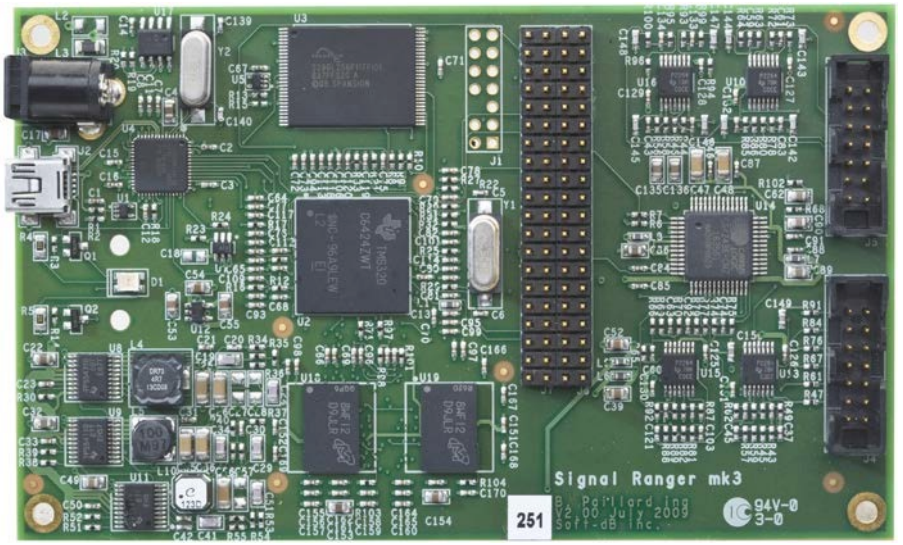
Signal Ranger mk3

Manuel de l'utilisateur



par

Soft dB

En association avec

Convergence Instruments

5 septembre 2011

1040, avenue Belvédère, suite 215
Québec (Québec) G1S 3G3 Canada
Tél. : (418) 686-0993 Fax : (418) 686-2043

1 Avant-propos

Depuis la série originale de cartes DSP *Signal_Ranger*, nous avons adopté une politique consistant à fournir à l'utilisateur toutes les informations nécessaires pour faire fonctionner nos cartes DSP, mais aussi pour comprendre leur fonctionnement dans les moindres détails. Par exemple, nous fournissons tous les schémas de la carte ; nous expliquons en détail le fonctionnement des noyaux DSP... etc. Contrairement aux produits concurrents, cette approche se traduit par une documentation plus vaste et plus détaillée qui peut donner l'impression d'un système trop complexe. Nos systèmes, outils et architectures sont en fait plus faciles à utiliser que de nombreux produits concurrents. Au fil des ans, nous avons appris que nombre de nos clients, dont certains sont des équipementiers, apprécient ce niveau de détail et de transparence dans notre documentation. C'est pourquoi nous avons décidé de poursuivre cette politique, et nous espérons que vous prendrez le temps de parcourir la documentation pour comprendre ce que nos outils et architectures peuvent apporter à vos développements.

2 Caractéristiques principales

SignalRanger_mk3 est une carte DSP à point fixe comprenant un DSP TMS320C6424 cadencé à 590 MHz, une interface analogique 96 kHz/24 bits à 6 entrées/6 sorties conçue pour les applications audio professionnelles et les applications de contrôle à haute performance, ainsi qu'une interface USB 2 à grande vitesse, permettant des communications rapides avec la carte. Le pilote USB Windows permet de connecter un nombre illimité de cartes à un PC.

La carte DSP peut être utilisée lorsqu'elle est connectée à un PC, ce qui permet d'échanger des données et des commandes entre le PC et la carte DSP en temps réel. Elle peut également être utilisée en mode autonome, en exécutant le code DSP intégré.

Les ADC et DAC ont un mode de fonctionnement (par défaut) dans lequel la bande passante descend jusqu'au courant continu. Cela permet à la carte d'être utilisée dans des applications de contrôle à haute performance.

Grâce à ses E/S analogiques de haute qualité, à sa programmabilité et au fait qu'elle peut fonctionner comme une carte autonome, la carte *SignalRanger_mk3* peut être utilisée dans de nombreuses applications :

- Acquisition et traitement multicanal de la parole et du son.
- Contrôle multicanal.
- Mesure et instrumentation.
- Analyse vibro-acoustique.
- Traitement des réseaux acoustiques/formage de faisceaux
- Développement de logiciels DSP.

2.1 Modes de démarrage et modes de fonctionnement

Le SignalRanger_mk3 comprend une ROM Flash de 32MB, à partir de laquelle le DSP

peut démarrer. Le DSP peut démarrer de deux manières :

- **Démarrage autonome :** Lors de la mise sous tension, si le micrologiciel est présent dans la mémoire Flash, il est chargé et exécuté. En préprogrammant la mémoire Flash avec le code DSP, la carte peut fonctionner en mode autonome, en exécutant une application DSP embarquée directement à la mise sous tension.
- **Démarrage du PC :** Lorsque la carte est connectée à un PC, ce dernier peut forcer le DSP à redémarrer. Dans ce mode, le PC peut forcer le rechargement du nouveau code du DSP. Ce mode peut être utilisé pour "prendre le contrôle" du DSP à tout moment. En particulier, il peut être utilisé pour reprogrammer la mémoire Flash de manière totalement transparente.

Même après le démarrage de la carte DSP en mode autonome, un PC peut être connecté à tout moment pour lire/écrire la mémoire du DSP sans interrompre le code DSP déjà en cours d'exécution. Ce comportement permet une visibilité et un contrôle en temps réel du code DSP intégré déjà en cours d'exécution.

La flexibilité offerte par ces modes de démarrage permet les modes de fonctionnement suivants :

- *SignalRanger_mk3* peut être utilisé dans des applications où il est toujours connecté à un PC. Ces configurations comprennent les applications d'analyse et de traitement des signaux où le PC est utilisé pour l'affichage et le contrôle.
- *SignalRanger_mk3* peut être utilisé dans des applications où il fonctionne dans une configuration autonome. Ces applications comprennent les applications de contrôle embarquées où la carte fournit une boucle de contrôle dédiée entre les entrées et les sorties.
- *SignalRanger_mk3* peut être utilisé dans des applications où il fonctionne dans une configuration autonome, mais peut être connecté à un PC pour des fonctions avancées ou des fonctions qui incluent l'affichage/le contrôle. Ces applications comprennent les applications d'enregistrement de données où la carte effectue l'enregistrement de données de manière autonome, mais peut être connectée à un PC pour télécharger les données ou configurer l'enregistrement de données. Dans ces applications, l'algorithme DSP en temps réel n'a pas besoin d'être interrompu pour permettre la connexion au PC. Cette connexion peut être transparente.

3 Données techniques

3.1 Alimentation électrique

SignalRanger_mk3 est auto-alimenté par un bloc d'alimentation externe de 5V (+-5%). Il peut fonctionner sans connexion à un PC.

La consommation d'énergie typique se situe entre 400 mA et 600 mA, en fonction de l'activité du DSP.

3.2 USB

La connexion PC USB 2.0 à haut débit offre un débit supérieur à 35 Mb/s en lecture et en écriture. Un contrôleur USB autonome décharge le DSP de toutes les tâches de gestion USB.

3.3 DSP

DSP à point fixe TMS320C6424, fonctionnant à 590 MHz.

3.4 Mémoire

- 208 Kbytes de RAM sur la puce (DSP).
- 128 Mbytes DDR2 RAM
- 32 Mbytes Flash Rom.

3.5 Entrées analogiques

- Nombre d'entrées : 6
- Résolution : 24 bits
- Bruit : 25 µV RMS
- Taux d'échantillonnage : jusqu'à 96 kHz
- Largeur de bande de l'entrée analogique : jusqu'à 48 kHz
- Filtre anti-crénelage : Intégré
- Filtre passe-haut : Bypassable
- Type d'entrée : Simple effet. Deux entrées peuvent être configurées pour prendre en charge un microphone électret.
- Gamme dynamique : +-3V
- Retard de groupe : 11 ou 14 échantillons selon le mode d'échantillonnage, y compris la mise en mémoire tampon logicielle

3.6 Sorties analogiques

- Nombre de sorties : 6
- Résolution : 24 bits
- Bruit : 11 µV RMS
- Taux d'échantillonnage : jusqu'à 96 kHz
- Largeur de bande de la sortie analogique : jusqu'à 48 kHz
- Filtre anti-crénelage : Intégré
- Type de sortie : Simple effet
- Gamme dynamique : +-2.1 V
- Capacité Source/Sink : +-2 mA à 25 degC
  +-1 mA à 100 degC
- Retard de groupe : 11 ou 14 échantillons selon le mode d'échantillonnage, y compris la mise en mémoire tampon logicielle

4 Logiciel

4.1 Interface SignalRanger DDCI

La pièce maîtresse de l'architecture *SignalRanger* est son interface *DDCI*. L'*interface DDCI (Development to Deployment Code Instrumentation)* permet à une application de contrôle fonctionnant sur un PC de communiquer avec un dispositif embarqué basé sur la plate-forme *SignalRanger* et de le contrôler.

Contrairement aux techniques traditionnelles de débogage et d'émulation, l'interface *DDCI* ne repose pas sur un pod émulateur ou sur l'environnement de développement *Code Composer Studio*™ pour communiquer avec la carte *SignalRanger_mk3*. Elle s'appuie plutôt sur la connexion USB existante et les bibliothèques logicielles déployables. Grâce à cela, le développeur travaille dans les mêmes conditions pendant le développement et après le déploiement de l'application.

L'interface fournit en effet une visibilité et un contrôle en temps réel du code qui s'exécute dans le dispositif embarqué. Elle est utile à deux stades du cycle de vie de l'application :

- Pendant le développement, il est utilisé pour fournir un débogage en temps réel au niveau de l'application, ce qui n'est généralement pas possible avec les techniques de débogage et d'émulation standard. En particulier, les fonctions de lecture, d'écriture et de contrôle du code ne nécessitent pas l'arrêt de l'unité centrale. L'interface permet d'instrumenter le code en temps réel et dans les conditions réelles de fonctionnement.
- Après le déploiement de l'application, lorsque la même interface est utilisée pour prendre en charge le contrôle par l'utilisateur et les communications avec le dispositif embarqué par le biais d'une application PC spécifique développée à cet effet.

Les trois caractéristiques essentielles de l'interface *DDCI* sont les suivantes :

- La même interface physique USB et les mêmes bibliothèques et fonctions hôtes sont utilisées pour prendre en charge l'interface aux deux stades du cycle de vie de l'application.
- Le fonctionnement de l'interface *DDCI* ne nécessite pas d'ajout ou d'adaptation de code DSP, et n'exige qu'un temps CPU ou une surcharge de mémoire minimes.
- L'interface physique USB peut être connectée et déconnectée en cours de fonctionnement, sans perturber le code DSP exécuté sur la plate-forme *SignalRanger.*

Le principal résultat de l'utilisation de l'interface est de condenser les deux étapes du cycle de vie de l'application en une seule étape plus courte. En effet, l'application est généralement déployée "telle quelle" juste après la phase de débogage.

Un autre avantage important de l'utilisation de l'interface est qu'elle permet, avec très peu d'efforts, d'avoir une bien meilleure visibilité en temps réel du fonctionnement du code intégré. Cette meilleure visibilité pendant le développement se traduit directement par un code intégré plus fiable.

Dans de nombreuses applications où il est nécessaire d'exécuter des simulations de l'algorithme de traitement du signal pour valider son fonctionnement, l'instrumentation en temps réel fournie par l'interface *DDCI* permet d'analyser facilement le comportement de haut niveau du code de traitement du signal, dans des conditions et avec des données réelles. Souvent, l'étape de simulation peut être complètement évitée, avec de meilleurs résultats basés sur des données réelles.

Lorsque l'on utilise l'interface LabVIEW, par opposition à l'interface C/C++, un troisième avantage de l'interface est que les vastes bibliothèques LabVIEW sont disponibles pour ajouter de puissantes capacités de traitement, d'analyse et d'affichage des signaux en temps réel, à la fois dans les phases de débogage et dans les phases de déploiement de l'application.

Les fonctions de l'interface peuvent être exercées pendant que le code intégré est en cours d'exécution. Toutes ces fonctions supportent l'*accès symbolique*, où le nom des variables et des fonctions DSP peut être utilisé pour y accéder, au lieu de leurs adresses absolues. L'avantage de cette fonction est que le code DSP intégré peut être modifié et relié à nouveau sans qu'il soit nécessaire de mettre à jour l'application d'accès utilisateur associée qui tourne sur un PC. Tant que les noms des variables et des fonctions restent les mêmes, l'accès reste opérationnel même après les mises à jour du code DSP.

Les fonctions en temps réel suivantes sont prises en charge directement par l'interface :

- Lecture et écriture de la RAM
- Lecture et écriture de la mémoire flash
- Lecture et écriture des périphériques
- Forcer l'exécution du code
- Réinitialisation de l'unité centrale
- Gestion de la mise à jour du micrologiciel en service
- Détection et gestion automatiques des dispositifs cibles.

L'interface est composée de plusieurs parties :

- **Conducteur signé pour :**
    - **Windows XP (plate-forme x86)**
    - **Windows Vista (plateformes x86 et x64)**
    - **Windows 7 (plateformes x86 et x64)**
    - **Linux**
    - **Mac-OS**

Remarque : La prise en charge de Linux et Mac-OS nécessite l'utilisation de LabVIEW pour les plates-formes correspondantes.

- **Noyau de communication :** Ce noyau réside dans la mémoire du DSP, avec le code spécifique de l'application de l'utilisateur. Il améliore la communication entre le PC et le DSP.
- **Bibliothèques LabVIEW :** Ces bibliothèques prennent en charge un large éventail de fonctions de communication, de programmation et de contrôle avec le noyau résident. Ils peuvent être utilisés pour créer un exécutable LabVIEW spécifique à une application afin de contrôler le dispositif *SignalRanger* intégré.
- **les bibliothèques C/C++ :** Pour les développeurs qui préfèrent travailler dans un environnement C/C++, nous fournissons des bibliothèques sous forme de DLL. Ces bibliothèques ont des fonctionnalités similaires à celles des bibliothèques LabVIEW.
- **Mini-Debugger :** Le Mini-Debugger Debugger est une application d'interface polyvalente qui prend en charge la programmation et le débogage symbolique en temps réel du code générique intégré au DSP. Il comprend des fonctions telles que le tracé graphique de données en temps réel, l'accès symbolique en lecture/écriture aux variables, l'exécution dynamique, les fonctions Flash

programmation... etc. À la base, le mini-débogueur utilise les mêmes bibliothèques d'interface qu'un développeur utilise pour concevoir une application DSP autonome. Cela assure une transition transparente entre l'environnement de développement/débogage et l'application déployée.

4.2 Autres outils logiciels

Le logiciel comprend également les éléments suivants

- **Application d'autotest :** Cette application teste tout le matériel de la carte DSP.
- **Exemples de code :** Plusieurs applications de démonstration LabVIEW illustrent le développement du code DSP. Elles montrent également comment interfacer ce code avec une application PC écrite en LabVIEW.
- **Pilote Flash et code d'exemple :** Ce pilote comprend tout le code permettant de configurer et d'utiliser la ROM Flash embarquée de 32 Mo à partir du code DSP de l'utilisateur.
- **Pilote CODEC et code d'exemple :** Ce pilote comprend tout le code permettant de configurer et d'utiliser le CODEC embarqué à partir du code DSP de l'utilisateur.
- **Pilote DAQ/Datalogger :** Une puissante application DAQ Driver/Datalogger est fournie. Cette application ne fait pas partie du kit d'installation. Elle peut être téléchargée à partir de http://www.softdb.com

5 Installation et tests

Note : Ne pas connecter la carte SignalRanger_mk3 au port USB du PC tant que le logiciel n'a pas été installé sur le PC. Le processus d'installation du pilote, qui se produit dès que la carte est connectée au PC, exige que les fichiers du pilote soient présents et accessibles sur le disque.

5.1 Installation du logiciel

Il existe deux ensembles de développeurs différents qui fournissent des fonctions similaires dans des formats différents :

- ***SR3_DDCI_Library_Distribution.zip :*** Le pack développeur LabVIEW comprend toutes les bibliothèques LabVIEW et toutes les applications de test et utilitaires sous forme de VI. C'est le package préféré des développeurs LabVIEW car il permet d'accéder au code LabVIEW des applications de démonstration et des applications utilitaires.
- ***SR3_Applications_Installer.exe :*** Le paquet de développement C/C++ comprend les bibliothèques au format DLL et les applications de test et utilitaires sous forme d'exécutable Windows (.exe).

5.1.1 LabVIEW Developer's Package (SR3_DDCI_Library_Distribution.zip)

Le paquetage LabVIEW est contenu dans un seul fichier zip nommé *SR3_DDCI_Library_Distribution*.

- Décompressez le fichier dans le répertoire de votre choix.
- Allez dans le dossier *Drivers* et exécutez *SRm3_Driver_Install.exe*.
- Suivre les instructions à l'écran

Le contenu du paquet est le suivant :

- **COFF_Management_Libraries :** Répertoire contenant tous les VIs de *COFF_Management* bibliothèque.
- **SRanger_mk3_Base_Library :** Répertoire contenant tous les VIs du *SignalRanger_mk3* bibliothèque
- **Firmware_Containers :** Répertoire contenant les VI permettant de créer des conteneurs de microprogrammes

- **Divers :** Répertoire contenant quelques VI de support
- **Documentation :** Répertoire contenant tous les manuels d'utilisation
- **DSP_Code :** Répertoire contenant les répertoires des différents projets DSP.
- **Pilotes :** Répertoire contenant les pilotes USB pour toutes les plates-formes prises en charge.
- **COFF_Management.lvlib :** Bibliothèque LabVIEW des VIs de gestion COFF
- **Firmware_Containers.lvlib :** Bibliothèque LabVIEW des VIs utilisés pour créer des VIs de conteneurs de firmware
- **SignalRanger_mk3.lvlib** Bibliothèque LabVIEW des VIs de l'interface
- **Divers fichiers DSP ".out".**

*Note : Le répertoire Drivers doit être stocké à un endroit fixe : Le répertoire **Drivers** doit être stocké à un endroit fixe. Dans certaines versions de Windows, lorsque l'ordinateur doit recharger les pilotes, il les recherche à l'endroit où ils ont été trouvés pour la première fois.*

5.1.2 Paquet du développeur C/C++ (SR3_Applications_Installer.zip)

Pour installer le paquet C/C++, exécutez *SR3_Applications_Installer_Vxyy.exe*. Ce programme installe les éléments suivants dans le répertoire *C:\NProgram Files\NSR3_Applications*

- **Documentation :** Répertoire contenant tous les manuels d'utilisation
- **Pilotes :** Répertoire contenant les pilotes USB pour toutes les plates-formes prises en charge.
- **Applications :** Répertoire contenant tous les exécutables
- **DSP_Code :** Répertoire contenant les répertoires des différents projets DSP.
- **SRm3_HL_DLL :** Répertoire contenant la DLL SRm3_HL utilisée pour le développement C/C++.
- **Visual_Studio_Code_Example :** Répertoire contenant un exemple d'application pour *SignalRanger_mk3* développé en Visual Studio. Cette application utilise la DLL *SRm3_HL*.

En outre, toutes les applications et la documentation sont accessibles à partir du menu *Démarrer*, sous l'onglet *SignalRanger_mk3* index.

5.2 Installation du matériel

Note : N'alimentez la carte qu'avec le bloc d'alimentation fourni ou avec un bloc d'alimentation de 5V/1A (+-5%). Si vous utilisez une alimentation personnalisée, assurez-vous que le côté positif de l'alimentation se trouve au centre de la fiche. Si vous n'utilisez pas l'alimentation appropriée, vous risquez d'endommager la carte.

- Mise sous tension de la carte à partir de l'adaptateur 5V
- La Led doit s'allumer en rouge pendant 1/2s, pour indiquer que la carte est correctement alimentée, puis en orange pour indiquer que la section DSP est fonctionnelle.
- Connectez la carte *SignalRanger_mk3* au port USB du PC.
- Si le pilote est correctement installé, la LED devient verte pour indiquer que le PC a pris le contrôle de la carte.
- À tout moment après que la carte a été connectée au PC et que ce dernier en a pris le contrôle, la LED doit être verte. La LED doit être verte avant d'essayer d'exécuter une application PC qui communique avec la carte.

5.3 Que faire en cas d'échec de l'installation du pilote ?

Pour procéder à l'installation manuelle d'un pilote, suivez les étapes suivantes :

- Mettez la carte sous tension et connectez-la au PC.
- Ouvrez le gestionnaire de périphériques (par exemple à partir du menu du panneau de configuration).
- L'arbre doit contenir un dispositif marqué *Unknown Device (dispositif inconnu*) ou *SignalRanger_mk3* avec un point d'exclamation à côté.
- Cliquez avec le bouton droit de la souris sur l'icône du périphérique et sélectionnez *Propriétés*.
- Dans la page des propriétés, cliquez sur *Mettre à jour le pilote*.
- Indiquez que vous souhaitez sélectionner le pilote manuellement et accédez au répertoire *C:\NProgram Files\SR3_Applications\NDrivers\SRm3cd.*

5.4 Indicateur LED

La LED de la carte *SignalRanger_mk3* a le comportement suivant :

- Il s'allume en rouge lorsque l'alimentation de 5V est appliquée pour la première fois à la carte. Cela indique que la carte est correctement alimentée.
- Elle devient orange d'elle-même 1/2s après la mise sous tension. Cela indique que la carte a suivi la séquence de réinitialisation et d'initialisation appropriée. Si du code DSP était présent dans la Flash, ce code a démarré lorsque la LED est orange.
- Elle devient verte lorsque la carte est connectée au PC via son port USB et que le PC en a pris le contrôle. La LED doit être verte avant qu'une application PC puisse communiquer avec la carte.
- Le voyant repasse à l'orange dès que la connexion USB est interrompue. C'est le cas lorsque le PC est éteint ou mis en veille, ou si le câble USB est déconnecté. Le fait que la LED devienne orange ne signifie pas que le code DSP a cessé de fonctionner, mais simplement que le PC ne peut pas communiquer avec la carte.
- La LED peut également devenir orange et repasser temporairement au vert lorsque la carte est réinitialisée par une application PC.
- Enfin, la DEL peut être modifiée à tout moment par une application utilisateur. Cependant, aucune des applications fournies ne présente ce comportement.

5.5 Test du conseil d'administration

L'application *SR3_SelfTest* peut être lancée à tout moment après que la carte a été mise sous tension et qu'elle est connectée à un PC (après que la LED est devenue verte).

Pour les utilisateurs qui ont installé le pack de développement C/C++, l'application *SR3_SelfTest* se trouve dans le menu de démarrage sous *Start\SignalRanger_mk3\SR3_Applications\SR3_SelfTest*. Pour les utilisateurs qui ont installé le package de développement LabVIEW, le VI équivalent se trouve dans la bibliothèque *Signal_Ranger_mk3.lvlib* dans le dossier *SelfTest*.

L'interface utilisateur de l'application *SR3_SelfTest* est présentée ci-dessous :

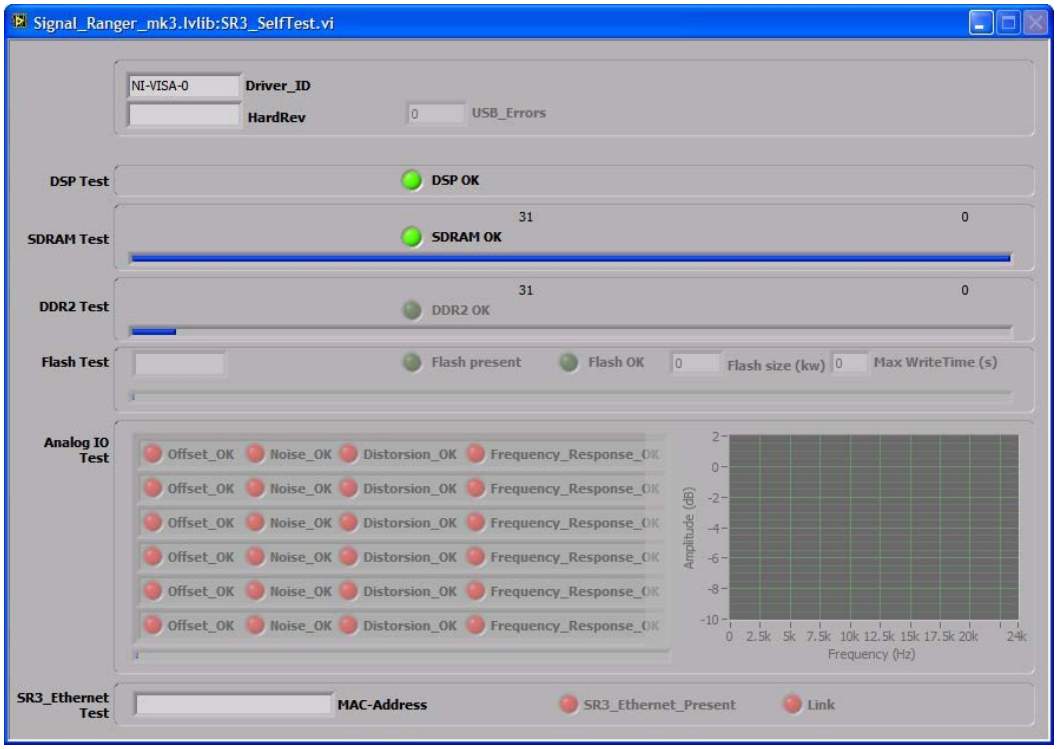


Figure 1 **Face avant de l'application *SR3_SelfTest.***

- Avant d'exécuter l'application, connectez le test d'entrée-sortie qui relie chaque entrée à la sortie correspondante.
- Pour lancer l'application, il suffit de cliquer sur la flèche blanche qui apparaît en haut à gauche de la fenêtre.
- L'application initialise la carte, charge le noyau sur le DSP, puis procède au test :
    - DSP
    - RAM sur puce
    - RAM DDR2
    - Flash
    - IOs analogiques
- Après chaque test, l'indicateur correspondant s'allume. Un indicateur vert indique que le test est réussi. Un indicateur rouge indique un échec.

Remarque : En raison de leur taille très importante, les tests de la mémoire vive DDR2 et de la mémoire flash prennent beaucoup de temps. Le test de la mémoire flash, en particulier, peut durer jusqu'à 30 minutes.

Note : Le test Flash efface le contenu de la mémoire flash : Le test Flash efface le contenu de la mémoire flash. Pour éviter de perdre le contenu de la mémoire flash, ce test peut être ignoré.

5.6 Évaluation de la performance analogique

L'application de démonstration *SR3_SignalTracker* a été conçue pour permettre le test et l'évaluation des canaux d'entrée/sortie analogiques. L'application permet à l'utilisateur d'envoyer des signaux de test à une sortie de canal sélectionnée et de surveiller le signal échantillonné sur une entrée sélectionnée. Les entrées sont affichées à la fois en termes de temps

ainsi que les spectres d'énergie instantanés ou moyennés. Les spectres d'énergie moyennés sont utiles pour évaluer le bruit d'entrée en fonction de la fréquence.

Pour les utilisateurs qui ont installé le kit de développement C/C++, l'application *SR3_SignalTracker* se trouve dans le menu de démarrage sous *Start\SignalRanger_mk3\SR3_Applications\SR3_SignalTracker*. Pour les utilisateurs qui ont installé le pack de développement LabVIEW, le VI équivalent se trouve dans la bibliothèque *Signal_Ranger_mk3.lvlib* dans le dossier *SignalTracker*.

Le panneau avant de l'application est divisé en plusieurs onglets, un pour chaque groupe de fonctions.

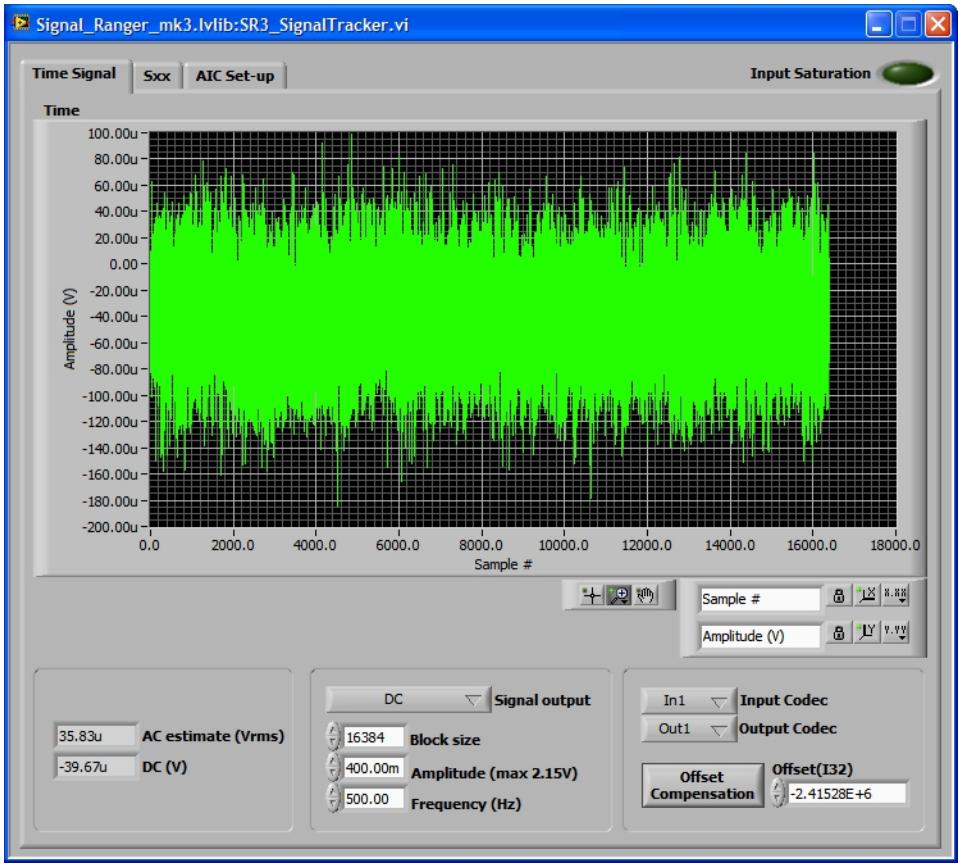


Figure 2 - Application SR2_SignalTracker Application SR2_SignalTracker - Onglet Signal temporel

Pour lancer l'application, il suffit de cliquer sur la flèche blanche en haut à gauche de la fenêtre.

L'application envoie des blocs d'échantillons de la longueur et de la forme d'onde sélectionnées à la sortie sélectionnée, et enregistre des blocs d'échantillons de la même longueur sur l'entrée sélectionnée. Les échantillons d'entrée enregistrés sont synchronisés avec les échantillons de sortie.

5.6.1 Onglet Signal temporel

5.6.1.1 Indicateur de temps

L'onglet signal temporel présente un tracé temporel du signal échantillonné sur l'entrée sélectionnée. L'échelle d'amplitude prend en compte le gain de la chaîne analogique et le gain du PGA, de sorte que l'amplitude du signal est représentée en Volts au niveau du connecteur.

5.6.1.2 Estimation CA (Vrms) Indicateur

Cet indicateur présente la valeur efficace du signal d'entrée (tout décalage en courant continu est supprimé avant le calcul de la valeur efficace).

5.6.1.3 Indicateur DC(V)

Cet indicateur présente la valeur moyenne en courant continu du signal temporel.

5.6.1.4 Contrôle de la sortie du signal

La commande *Signal output* sélectionne un type de forme d'onde à partir d'une liste de formes d'onde prédéfinies. La sélection *No Output* envoie zéro échantillon à la sortie.

5.6.1.5 Contrôle de la taille des blocs

Le contrôle de la *taille des blocs* définit le nombre d'échantillons envoyés à la sortie et enregistrés de manière synchrone à partir de l'entrée.

5.6.1.6 Contrôle de l'amplitude

La commande d'*amplitude* permet de régler l'amplitude de la forme d'onde de sortie. Les échantillons de sortie sont calculés en fonction de l'amplitude sélectionnée, ainsi que du gain de sortie sélectionné (PGA).

5.6.1.7 Contrôle de la fréquence

Le contrôle de la *fréquence* n'est utilisé que pour les formes d'onde périodiques. Elle permet de régler la fréquence fondamentale de la forme d'onde.

5.6.1.8 Contrôle du codec d'entrée

La commande *Input Codec* permet de sélectionner le canal d'entrée entre 0 et 6.

5.6.1.9 Contrôle du codec de sortie

Le *codec de sortie* sélectionne le canal de sortie entre 0 et 6.

5.6.1.10 Contrôle de la compensation du décalage

Le bouton *Compensation de décalage* permet d'effectuer une compensation de décalage. Cette procédure lit un bloc d'échantillons d'entrée à partir de l'entrée sélectionnée tout en envoyant des échantillons nuls. La moyenne des échantillons enregistrés est ensuite soustraite des échantillons d'entrée. Par conséquent, si un décalage est présent sur l'entrée sélectionnée, il est ramené à zéro. La moyenne est affichée dans l'indicateur *Offset(I32)*. Cet indicateur est mis à l'échelle en bits. La compensation du décalage est logicielle.

5.6.1.11 Offset(I32) Contrôle

L'indicateur *Offset(I32)* peut également servir de contrôle. Le simple fait de modifier le contenu de ce champ impose un nouveau décalage logiciel aux échantillons d'entrée enregistrés.

5.6.2 Onglet Sxx

5.6.2.1 Indicateur de spectre

L'indicateur de *spectre* présente le spectre de puissance instantané ou moyenné du bloc échantillonné d'entrée.

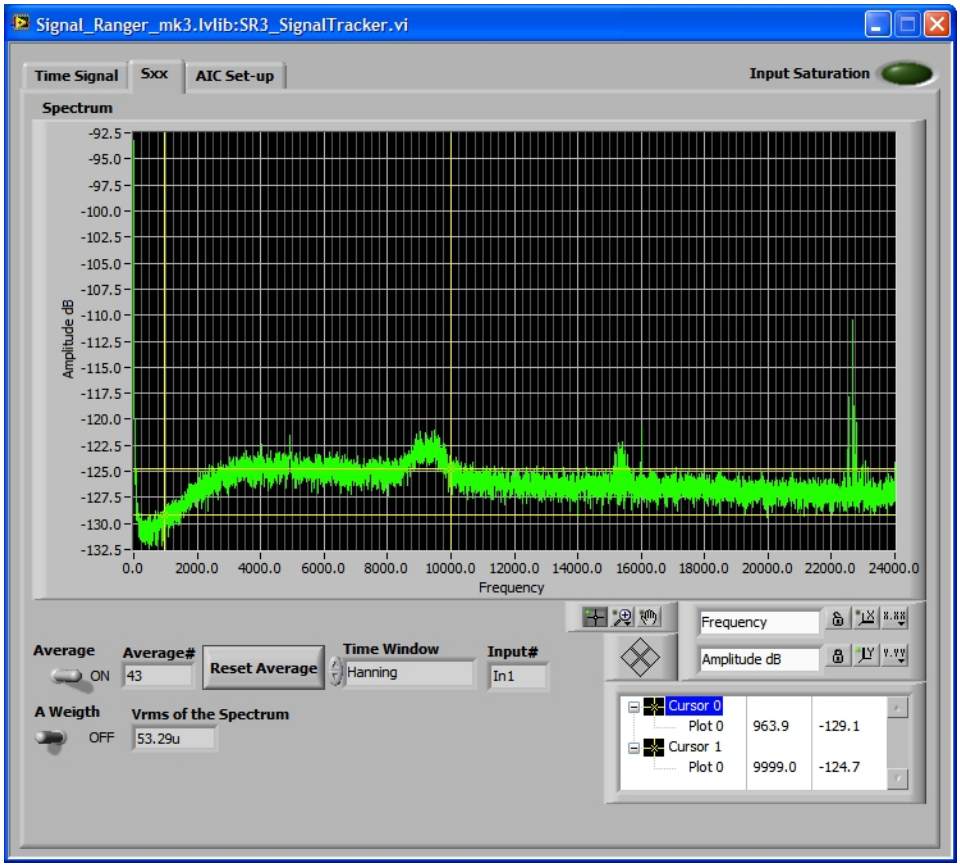


Figure 3 : Application SR2_SignalTracker - onglet Sxx **Application SR2_SignalTracker - Onglet Sxx**

L'échelle verticale est en dB. Une valeur de 0 dB représente une amplitude de 1Vrms.

5.6.2.2 Contrôle moyen

Pour calculer la moyenne du spectre de puissance, il suffit de placer la commande *Average* en position ON. Sinon, l'écran affiche les spectres instantanés.

5.6.2.3 Bouton de réinitialisation de la moyenne

Le bouton *Réinitialiser la moyenne* permet de réinitialiser la moyenne.

5.6.2.4 Sélecteur de fenêtre temporelle

Une fenêtre d'analyse optionnelle peut être choisie dans la liste *Time-Window.*

5.6.2.5 Un contrôle du poids

Le spectre de bruit peut être affiché avec une pondération A optionnelle. Utilisez la commande pour activer ou désactiver la pondération A.

5.6.2.6 Indicateur Vrms

Cet indicateur présente la valeur efficace du signal d'entrée. Le calcul est effectué dans le domaine des fréquences. Deux facteurs peuvent expliquer une différence entre cette valeur et la valeur affichée dans le domaine temporel :

- La composante continue n'est pas soustraite du signal avant le calcul.
- L'effet du filtre de pondération A est inclus dans le calcul.

5.6.2.7 Contrôles du graphique et du zoom

Les commandes du graphique peuvent être utilisées pour modifier le facteur de zoom. Par défaut, le graphique est mis à l'échelle automatiquement en X et en Y, ce qui est indiqué par les verrous fermés à côté de chaque nom d'échelle. Pour désactiver l'échelle automatique, il suffit d'appuyer sur le bouton de verrouillage.

5.6.2.8 Curseurs

Deux curseurs peuvent être déplacés sur le graphique. La fréquence et la valeur de l'amplitude au niveau du curseur sont affichées dans la fenêtre du curseur.

5.6.3 Onglet de configuration de l'AIC

L'onglet Configuration AIC présente les différents contrôles pour la configuration de l'acquisition.

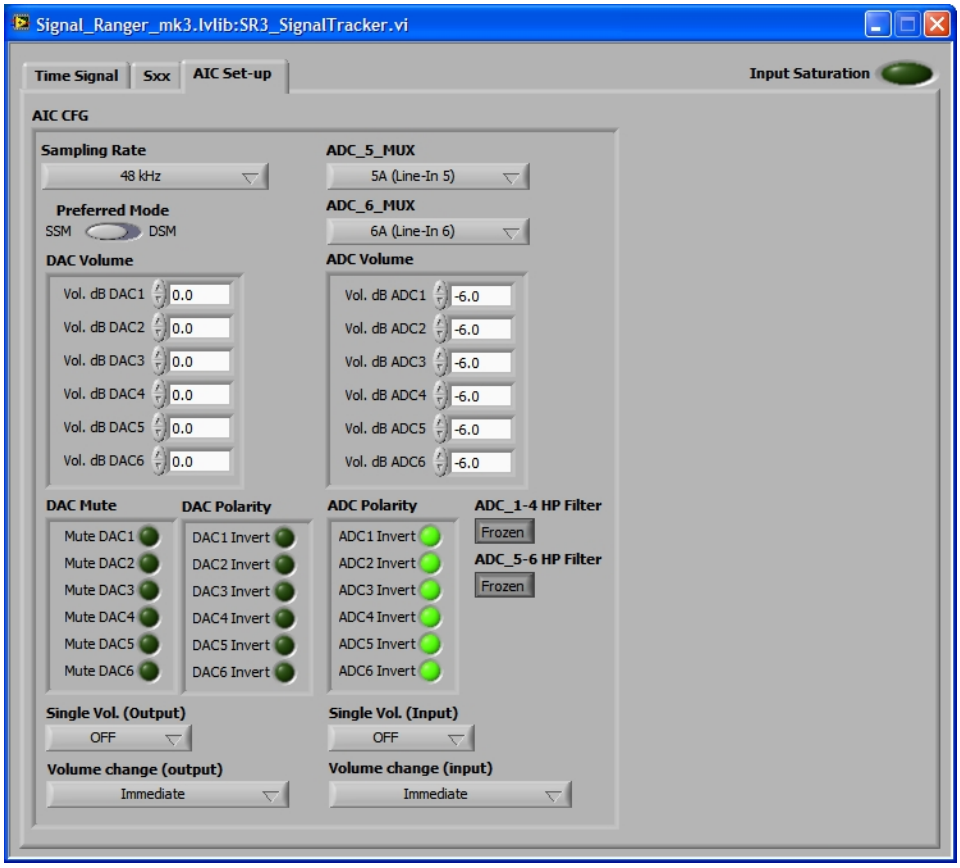


Figure 4 Application SR2_SignalTracker - Onglet Configuration AIC

5.6.3.1 Contrôle AIC_Setup_Array

Cet onglet permet de contrôler individuellement tous les paramètres du CODEC.

5.6.3.1.1 Taux d'échantillonnage

Cette commande permet de sélectionner le taux d'échantillonnage. La fréquence d'échantillonnage peut être choisie parmi un ensemble de valeurs comprises entre 4 kHz et 96 kHz. Notez que la commande n'expose que les fréquences d'échantillonnage les plus fréquentes, les autres étant possibles. Le *mode préféré* est utilisé chaque fois que la fréquence d'échantillonnage choisie le permet. Certains choix de fréquence d'échantillonnage ne sont compatibles qu'avec le mode DSM.

5.6.3.1.2 Mode préféré

Cette commande indique si le mode d'échantillonnage doit être *DSM* (Double-Rate Sampling) ou *SSM* (Single-Rate Sampling). Certaines fréquences d'échantillonnage ne permettent que le mode *DSM*. Dans ce cas, le mode préféré est ignoré. Au niveau inférieur, cette commande agit sur les variables de pilotage *Div_osc* et *DSM_SSM.*

5.6.3.1.3 Volume du DAC

Ce groupe contient le volume de chaque sortie. Le volume peut être réglé de -127,5 dB à 0 dB en
Pas de 0,5 dB.

5.6.3.1.4 Volume de l'ADC

Ce groupe contient le volume de chaque entrée. Le volume peut être réglé de -64,0 dB à +24,0 dB par pas de 0,5 dB.

5.6.3.1.5 DAC Mute

Cette commande permet de couper le son de chaque sortie individuellement.

5.6.3.1.6 Polarité du DAC

Cette commande permet de sélectionner la polarité de la sortie (positive ou négative).

5.6.3.1.7 Polarité de l'ADC

Cette commande permet de sélectionner la polarité de l'entrée (positive ou négative).

5.6.3.1.8 ADC_x-y Filtre HP

Cette commande active les filtres passe-haut du CAN ou les fige sur la dernière valeur. Le fait d'engager brièvement le filtre et de le geler annule effectivement le décalage en courant continu présent sur l'entrée du CAN. Cette opération s'effectue au niveau du CAN, tandis que la compensation du décalage s'effectue au niveau du logiciel.

5.6.3.1.9 AIN5_MUX et AIN6_MUX

Ces commandes permettent de sélectionner le chemin d'entrée pour les entrées analogiques 5 et 6, soit Line-In, soit Electret-Microphone Input.

6 Description du matériel

6.1 Carte du connecteur

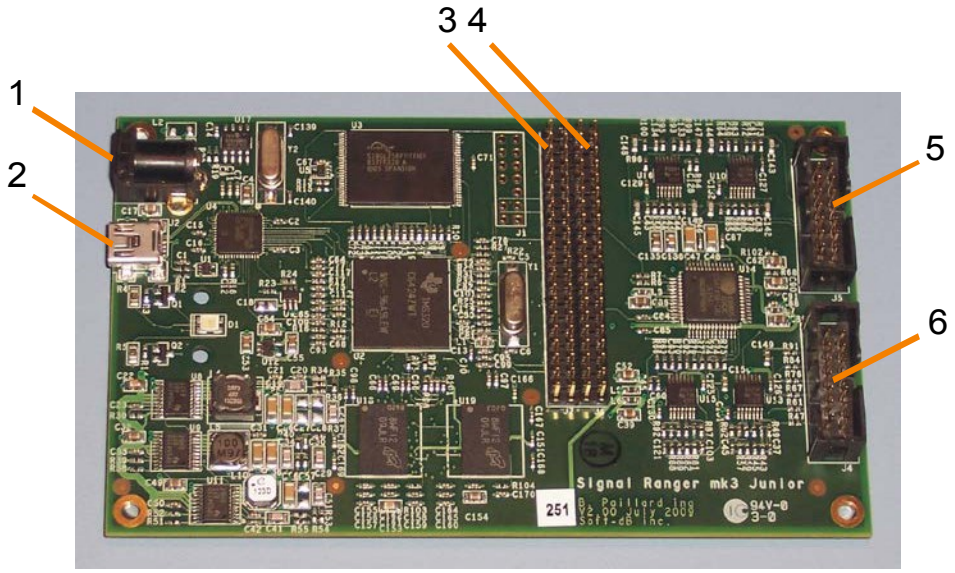


Figure 5 **Carte DSP SignalRanger_mk3**

Légende :

Alimentation 5V J3
Connecteur USB (Mini-B)
Connecteur d'extension J7
Connecteur d'extension J6
Entrées analogiques J5
Sorties analogiques J4

6.2 Connecteurs d'extension J6 et J7

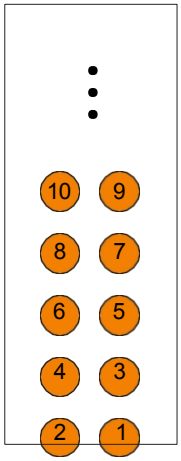


Figure 6 **Brochage des connecteurs J6 et J7**

6.2.1 Brochage J6

Non	Fonction	Non	Fonction
40	+5V	39	Gnd
38	NC	37	Gnd
36	+3.3V	35	Gnd
34	NC	33	Gnd
32	-3.3V	31	Gnd
30	SCL	29	Gnd
28	SDA	27	Gnd
26	UTXD0/GP[86]	25	Gnd
24	URXD0/GP[85]	23	Gnd
22	URTS0/PWM0/GP[88]	21	Gnd
20	UCTS0/GP[87]	19	Gnd
18	ACLKX0/CLKX1/GP[106]	17	Gnd
16	TOUT1L/UTXD1/GP[55]	15	Gnd
14	TINP1L/URXD1/GP[56]	13	Gnd
12	AMUTE0/DR1/GP[110]	11	Gnd
10	AMUTEIN0/FSX1/GP[109]	9	Gnd
8	AHCLKX0/CLKR1/GP[108]	7	Gnd
6	CLKS1/TINPOL/GP[98]	5	Gnd
4	AXR0[0]/FSR1/GP[105]	3	Gnd
2	AFSX0/DX1/GP[107]	1	Gnd

Tableau 1 **Connecteur J6**

6.2.1.1 Broches d'alimentation

6.2.1.1.1 +5V

Il s'agit de la même alimentation que celle fournie au connecteur d'alimentation 5V J3. Le courant maximum qui peut être tiré de cette broche est de 500 mA. Il peut être limité par la capacité de l'alimentation utilisée.

6.2.1.1.2 +3,3V

Il s'agit de l'alimentation logique principale. Le courant maximal pouvant être tiré de cette broche est de 400 mA.

6.2.1.1.3 -3.3V

Il s'agit d'une petite alimentation de polarisation. Le courant maximum qui peut être tiré de cette broche est de 40 mA.

6.2.1.1.4 Autres broches

Toutes les autres broches de J6 sont des broches DSP. Voir la documentation du DSP pour la fonction.

6.2.2 Brochage J7

Non	Fonction	Non	Fonction
40	EM_CS3/GP[13]	39	Gnd
38	GP[4]/PWM1	37	Gnd
36	GP[22]/(BOOTMODE0)	35	Gnd
34	GP[23]/(BOOTMODE1)	33	Gnd
32	GP[24]/(BOOTMODE2)	31	Gnd
30	GP[25]/BOOTMODE3)	29	Gnd
28	GP[26]/(FASTBOOT)	27	Gnd
26	GP[53]	25	Gnd
24	GP[54]	23	Gnd
22	RMREFCLK/GP31]	21	Gnd
20	RMCRSDV/GP[30]	19	Gnd
18	RMTXEN/GP[29]	17	Gnd
16	RMRXER/GP[52]	15	Gnd
14	RMTXD1/GP[27] /(LENDIAN)	13	Gnd
12	RMTXD0/GP[28]/8_16	11	Gnd
10	RMRXD1/EMCS5/GP[33]	9	Gnd
8	RMRXD0/EMCS4/GP[32]	7	Gnd
6	AD4/GP[3]	5	Gnd
4	AD2/GP[2]	3	Gnd
2	VLYNQ_Clock/PCICLK/GP[57]	1	Gnd

Tableau 2 Connecteur J7

6.2.2.1.1 Pins DSP

Toutes les broches de J7 sont des broches DSP. Voir la documentation du DSP pour la fonction.

6.3 Connecteurs analogiques J4 et J5

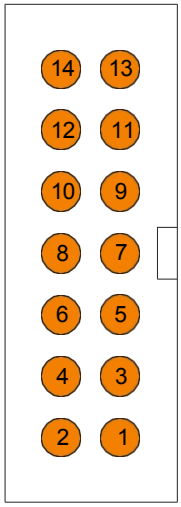


Figure 7 : Connecteurs analogiques J4 et J5 **Connecteurs analogiques J4 et J5**

6.3.1 Brochage J4

Non	Fonction	Non	Fonction
14	OUT_1	13	Gnd
12	OUT_2	11	Gnd
10	OUT_3	9	Gnd
8	OUT_4	7	Gnd
6	OUT_5	5	Gnd
4	OUT_6	3	Gnd
2	+3.3V	1	-3.3V

Tableau 3 **Connecteur J4**

6.3.2 Brochage J5

Non	Fonction	Non	Fonction
14	IN_1	13	Gnd
12	IN_2	11	MIC_IN_5
10	IN_3	9	Gnd
8	IN_4	7	MIC_IN_6
8	IN_5	5	Gnd
4	IN_6	3	Gnd
2	+3.3V	1	-3.3V

Tableau 4 **Connecteur J5**

Remarque : les ADC 5 et 6 peuvent être configurés pour prendre en charge un microphone à électret. Le microphone doit être connecté entre les broches 11 et la masse (ADC 5), et entre les broches 7 et la masse (ADC 6). Pour prendre en charge cette entrée microphone, l'ADC doit être configuré avec le bit AINx_MUX correspondant du registre 5 (ADC_Control) réglé sur 1.

6.4 Fréquences du système

Le cristal du DSP a une fréquence de 24,576 MHz. Immédiatement après la réinitialisation, les différentes fréquences du système sont les suivantes :

- Horloge centrale du CPU - SYSCLK1 (/1) : 24,576 MHz
- SYSCLK2 (/3) : 8,192 MHz

- SYSCLK3 (/6) : 4,096 MHz

Cependant, la configuration ci-dessus est de courte durée. Juste après la réinitialisation, le noyau est chargé automatiquement dans la mémoire du DSP et exécuté. Le noyau configure le générateur d'horloge comme suit :

- Horloge centrale du processeur - SYSCLK1 (/1) : 589,824 MHz
- SYSCLK2 (/3) : 196,608 MHz
- SYSCLK3 (/6) : 98,304 MHz

6.5 Interfaces périphériques

6.5.1 Flash ROM

6.5.1.1 Carte mémoire

La ROM Flash de 32 MByte est mappée sur le chip-select 2 (CS2) aux adresses $_{42000000(H)}$à $43FFFFFF_H$.

6.5.1.2 Secteurs

Le dispositif Flash ROM est divisé en 256 secteurs de longueur égale de 128 Ko chacun.

6.5.1.3 Programmation incrémentale

La programmation incrémentale (programmation de la même adresse plusieurs fois) est autorisée. Chaque opération de programmation ne peut que remettre à zéro des bits individuels de 1 à 0. La mise à zéro des bits (de 0 à 1) ne peut être effectuée que par un cycle d'effacement sur un secteur entier. Cependant, les opérations de programmation peuvent réinitialiser des bits individuels à tout moment sans effacement intermédiaire.

6.5.1.4 Interface de bus

L'interface entre la mémoire flash et le DSP a une largeur de 16 bits.

6.5.1.5 Vitesse d'accès

- Lecture de 8 ou 16 bits : 132,2 ns
- Lecture 32 bits : 264,5 ns (2 cycles)

6.5.1.6 Configuration EMIF

Le tableau suivant décrit le contenu du registre A1CR, qui définit les paramètres des accès à la Flash.

Champ de bits	Fonction	Paramètres
SS	Select-Strobe	0 (mode normal)
EW	Attente prolongée	0 (pas d'attente prolongée)
W_Setup	Temps d'écriture - 1	4 (5 cycles)
W_Strobe	Temps d'écriture -1	5 (6 cycles)
W_Hold	Temps de maintien de l'écriture - 1	0 (1 cycle - réglage minimum)
R_Setup	Temps de préparation de la lecture - 1	5 (6 cycles)
R_Strobe	Temps de lecture - 1	5 (6 cycles)
R_Hold	Temps de maintien de la lecture - 1	0 (1 cycle - réglage minimum)
TA	Délai d'exécution	1 (2 cycles)
ASize	Largeur du bus	1 (16 bits)

Tableau 5 Configuration EMIF pour CS2 (contenu du registre A1CR)

Avec ces paramètres, les cycles d'accès à la mémoire flash sont les suivants :

- Temps de lecture : 132ns (13 cycles)
- Temps d'écriture : 122ns (12 cycles)

- Temps de réponse : 20,3ns (2 cycles entre la lecture et l'écriture ou entre l'écriture et la lecture)

6.5.2 RAM DDR2

6.5.2.1 Carte mémoire

La RAM DDR2 de 128 Mo est mappée aux adresses $_{80000000(H)}$à $87FFFFFF_H$.

6.5.2.2 Vitesse d'horloge

La mémoire vive DDR2 est cadencée à 165,85 MHz (331,7 MHz en fréquence double).

6.5.2.3 Interface de bus

L'interface DDR2-RAM vers DSP a une largeur de 32 bits.

6.5.2.4 Vitesse d'accès

6.5.2.4.1 Lecture et écriture à l'aide de DMA

- Lecture sur 8, 16 ou 32 bits : 9,3 ns
- Écriture sur 8, 16 ou 32 bits : 6,4 ns

6.5.2.4.2 Lecture et écriture à partir de l'unité centrale

- Lecture sur 8, 16 ou 32 bits : 125 ns
- Écriture sur 8, 16 ou 32 bits : 30,5 ns

6.5.2.4.3 Exécution de code à partir de la mémoire vive DDR2

- Le temps dépend de nombreux facteurs. Grâce à l'IDMA et au pipeline logiciel, l'exécution à partir de DDR2 peut parfois être aussi rapide qu'à partir de L1P. Dans la plupart des cas, l'exécution à partir de L1P est plusieurs fois plus rapide.

6.5.3 Codec

Le codec dispose de 6 entrées et 6 sorties analogiques de haute performance.

Chaque entrée a une plage dynamique de +-3V. Il comprend un gain réglable de -64 dB à +24 dB en Pas de 0,5 dB.

Note : En pratique, les gains supérieurs à -6dB ne présentent aucun avantage : Dans la pratique, les gains supérieurs à -6dB ne présentent aucun avantage. Ils n'améliorent pas le facteur de bruit et limitent la plage dynamique. Les gains inférieurs à -6 dB n'améliorent pas la gamme dynamique au-delà de +-3V.

Chaque sortie a une plage dynamique de +-2,1V. Il comprend un atténuateur réglable de -90 dB à 0 dB par pas de 0,5 dB.

Les ADC 5 et 6 ont deux entrées microphones. Ces entrées sont activées en mettant les bits *AINx_MUX* du registre 5 à 1. La remise à zéro du bit sélectionne l'entrée ligne. Voir la section sur les connecteurs pour les détails de connexion des microphones.

Le codec est connecté au DSP via McBSP-0.

Une bibliothèque de code DSP est fournie pour prendre en charge le codec, et l'application *SR3_SignalTracker* est fournie pour démontrer ses caractéristiques.

7 Stratégie de développement du code

Depuis la série originale *Signal Ranger*, la stratégie de développement que nous proposons est basée sur l'approche *DDCI* (*Development to Deployment Code Instrumentation*). Cette stratégie est légèrement différente de celle à laquelle de nombreux développeurs sont habitués. Cependant, elle accélère considérablement le temps de développement et, en même temps, la visibilité en temps réel qu'elle offre sur l'exécution du code DSP se traduit par un code plus fiable.

Contrairement aux techniques traditionnelles de débogage sur émulateur, l'interface *DDCI* s'appuie sur la connexion USB existante (ou Ethernet le cas échéant) et sur des bibliothèques logicielles déployables pour communiquer avec la carte à des fins de débogage. De ce fait, le développeur travaille dans les mêmes conditions pendant le développement qu'après le déploiement du produit final.

Note : La série SignalRanger fournit une connexion émulateur JTAG. Les développeurs qui ont absolument besoin d'utiliser cette approche peuvent donc le faire. Cependant, l'approche DDCI offre généralement une meilleure visibilité et un meilleur contrôle sur le code DSP et moins de restrictions. C'est donc généralement le choix préféré.

L'interface *DDCI* s'appuie sur un ensemble de bibliothèques (bibliothèques LabVIEW ou DLL) pour communiquer avec la carte en temps réel, pendant que le code DSP est en cours d'exécution. Comme les mêmes voies de communication et bibliothèques sont utilisées pendant le développement et après le déploiement, le code n'a pas besoin d'être ajusté ou redéveloppé pour tenir compte des différents canaux de communication entre les deux phases de développement.

En outre, par rapport aux techniques de débogage basées sur un émulateur, l'approche *DDCI* permet l'instrumentation du code en temps réel, sans arrêter l'unité centrale.

Cette capacité d'instrumentation du code en temps réel permet à son tour la mise en œuvre de tests avec du "matériel dans la boucle". Par rapport à l'approche de simulation plus traditionnelle, l'approche "hardware-in-the-loop" fournit de meilleurs résultats qui prennent tout en compte et elle est généralement plus facile à configurer et à mettre en œuvre.

La stratégie de développement de *la DDCI* suit généralement le schéma ci-dessous :

- Le code DSP est développé et intégré dans un exécutable à l'aide du *studio Code Composer*. Au début du développement, cette tâche est facilitée par les exemples, les bibliothèques et les shells de code que nous fournissons.
- Au début du développement, le développeur teste ce code DSP à l'aide du *Mini-Debugger*. Le *mini-débogueur* permet au développeur de télécharger le code, de le démarrer, de lire/écrire les emplacements de mémoire et les périphériques pendant l'exécution du code, de lancer l'exécution de fonctions spécifiques... etc. L'architecture *DDCI* ne nécessite pas de code DSP spécial ou d'ajustements pour mettre en œuvre les communications requises pour le débogage. Par conséquent, le code DSP est le même pendant la phase de débogage qu'après le déploiement. Le code déployé peut être instrumenté directement et facilement sans aucune modification.
- Au fur et à mesure que le développement progresse, le développeur trouvera plus facile de développer de petites applications pour interagir avec le code DSP d'une manière spécifique et tester ses différentes fonctions. Ces applications sont développées avec LabVIEW ou Visual C++, ou tout autre environnement de développement qui supporte les appels DLL. Ces applications s'appuient sur les bibliothèques de communication *DDCI* que nous fournissons. Ce sont ces mêmes bibliothèques de communication qui sont à la base du *Mini-Debugger*. Comme les bibliothèques de communication utilisent des informations symboliques, les applications côté PC n'ont pas besoin d'être ajustées lorsque le code du DSP est modifié ou reconstruit.
- À ce niveau de développement, de nombreuses applications DSP nécessitent une analyse fine du comportement de haut niveau de l'algorithme DSP. L'interface *DDCI* permet au développeur d'observer les données traitées par l'algorithme en temps réel, ainsi que d'agir sur les paramètres de l'algorithme ou de les ajuster pendant le traitement. Dans de nombreux cas, cette approche peut avantageusement remplacer les passes de simulation, en fournissant de meilleures données provenant d'un test réel. Grâce à ses nombreuses bibliothèques de traitement, d'analyse et d'affichage des signaux, LabVIEW facilite considérablement ce type de tâches.
- Pas à pas, ces petites applications d'essai se transforment généralement en applications complètes pour l'utilisateur final, qui sont finalement déployées avec le produit final. L'interaction du code PC-DSP étant basée sur les mêmes canaux de communication et bibliothèques logicielles, il n'y a pas de surprise lors de la migration des applications de test vers les applications déployées.
- En fin de compte, le code du DSP (et la logique FPGA pour les cartes qui incluent un FPGA) peut être flashé dans la ROM embarquée. Du point de vue du DSP, le processus de démarrage est exactement le même lorsque le code est téléchargé à partir du PC que lorsqu'il est copié à partir de la ROM FLASH embarquée. Ce processus

assure que le développeur n'a pas de surprises au moment du déploiement. La gravure du code DSP dans la mémoire Flash embarquée s'effectue à l'aide du *mini-débogueur*.

8 Mini-Debugger

Le mini-débogueur permet au développeur d'interagir :

- Réinitialiser la carte DSP.
- Télécharger un fichier exécutable DSP dans la mémoire du DSP ou utiliser les symboles d'un code DSP déjà en mémoire.
- Lancer l'exécution d'un code (fonction simple ou code entier) à partir d'une adresse spécifiée ou d'une étiquette symbolique.
- Lecture et écriture des registres de l'unité centrale.
- Lecture et écriture de la mémoire du DSP avec ou sans accès symbolique.
- Effacer, programmer et vérifier la mémoire Flash.
- Téléchargement interactif d'un fichier logique FPGA dans le FPGA

Le mini-débogueur peut être utilisé pour explorer les fonctions du DSP ou pour tester le code du DSP pendant le développement.

Le mini-débogueur est simplement une application d'interface utilisateur qui exploite les capacités des bibliothèques d'interface PC pour permettre au développeur d'observer et de modifier le code et les variables du DSP en temps réel, pendant que le code du DSP est en cours d'exécution. Comme il s'agit des mêmes bibliothèques que celles fournies pour développer les applications PC qui utilisent la carte, la transition entre le débogage et le déploiement sur le terrain est totalement transparente.

Au démarrage, le mini-débogueur demande à l'utilisateur si le DSP doit être réinitialisé. La réinitialisation du DSP peut résoudre des problèmes de connexion, tels qu'un plantage du DSP, mais elle interrompt le code déjà en cours d'exécution.

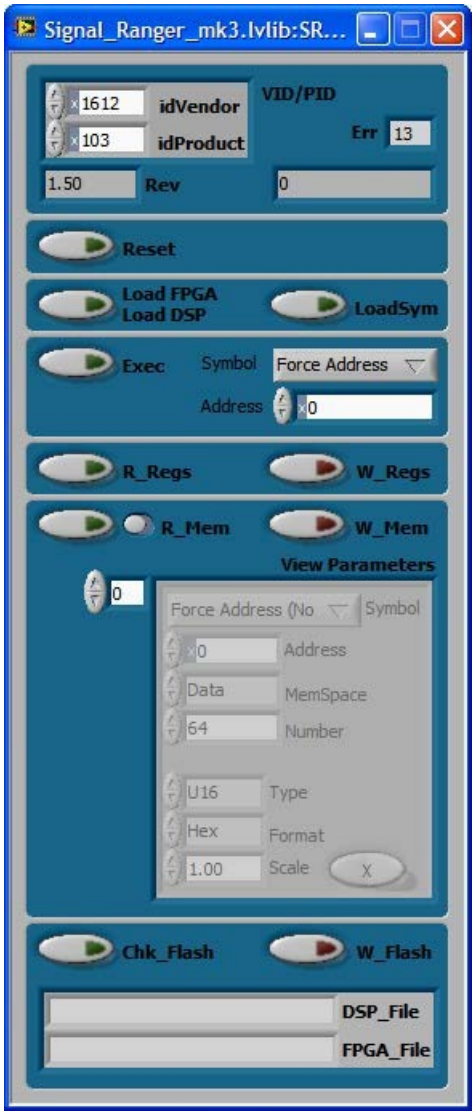


Figure 8 **Interface utilisateur du mini-débogueur**

8.1 Description de l'interface utilisateur

- **VID/PID** Le développeur doit saisir l'ID du fournisseur et l'ID du produit de la carte cible. Le mini-débogueur prend en charge plusieurs cartes DSP apparentées. Ce champ est utilisé pour indiquer le type de carte dont le mini-débogueur doit prendre le contrôle. La figure montre le VID et le PID de la carte *SignalRanger_mk3*.
- **Rev** Ce champ indique le numéro de révision du micrologiciel de la carte.
- **Erreur** Cet indicateur montre le nombre d'erreurs USB en temps réel. Il peut être utilisé pour surveiller la "santé" de la connexion USB. Cet indicateur est un compteur enveloppant de 4 bits qui se trouve dans le matériel du contrôleur USB embarqué. Notez que l'USB est un bus et que, par conséquent, bon nombre des erreurs détectées peuvent en fait provenir d'autres dispositifs sur le bus USB. En outre, les erreurs USB sont corrigées dans le cadre du protocole USB. Par conséquent, un grand nombre d'erreurs ne signifie pas nécessairement que la connexion n'est pas fiable.
- **Réinitialisation** Force une réinitialisation de la carte et recharge le noyau Host-Download. Tous les codes DSP précédemment exécutés sont interrompus. Cela peut être nécessaire pour prendre le contrôle d'un DSP qui s'est planté. Notez que le DSP n'est pas réinitialisé par défaut lorsque le mini-débogueur se connecte à la carte. Cela permet au code qui a pu être chargé et exécuté par le noyau de mise sous tension de continuer sans interruption.

- **ChargerFPGA/DSP** Charge un fichier COFF DSP et/ou un fichier FPGA. Le DSP est automatiquement réinitialisé avant le chargement. L'application présente un navigateur de fichiers pour permettre à l'utilisateur de sélectionner les fichiers DSP et FPGA. L'utilisateur peut choisir de ne pas charger le fichier DSP ou le fichier FPGA ou les deux. Les fichiers doivent être des fichiers COFF et .rbt légitimes pour le DSP et le FPGA cibles respectivement. Une fois que le fichier COFF a été chargé avec succès dans la mémoire du DSP, la table de symboles correspondante est chargée dans la mémoire du PC pour permettre l'accès symbolique aux variables et aux étiquettes. Le code n'est pas exécuté. Une fois que le fichier FPGA a été chargé avec succès dans le FPGA, la logique est fonctionnelle.
- **LoadSym** Charge la table des symboles correspondant à un code DSP spécifié dans la mémoire du PC pour permettre l'accès symbolique aux variables et aux étiquettes. Rien n'est chargé dans la mémoire du DSP. Cette fonction est utile pour obtenir un accès symbolique à un code DSP qui peut déjà être en cours d'exécution, tel que le code chargé à partir de Flash par le processus de démarrage. L'application présente un navigateur de fichiers pour permettre à l'utilisateur de sélectionner le fichier. Le fichier doit être un fichier COFF légitime pour le DSP cible.
- **Exécution** Force l'exécution à passer à l'étiquette ou à l'adresse spécifiée. Le code DSP ainsi activé doit contenir un *accusé de réception* sous la forme d'une *demande d'interruption de l'hôte (HINT)*. Sinon, le contrôleur USB s'arrêtera et une erreur sera détectée par le PC au bout de 5 secondes. La façon la plus simple de procéder est d'inclure la macro d'*acquittement* au début du code sélectionné. Cette macro est décrite dans les deux applications de démonstration.
    - ***Symbole*** ou ***Adresse*** est utilisé pour spécifier le point d'entrée du code DSP à exécuter. ***Symbole*** peut être utilisé si un fichier COFF contenant le symbole a été chargé précédemment. Sinon, l'***adresse*** permet de spécifier une adresse de branchement absolue. ***L'adresse*** n'est utilisée que si ***le symbole*** est réglé sur la position "Force Address (No Symbol)".
      Lorsqu'un nouveau fichier COFF est chargé, le mini-débogueur essaie de trouver le symbole ***_c_int00*** dans la table des symboles. S'il est trouvé et que sa valeur est valide (différente de 0), ***Symbol*** pointe vers son adresse. S'il n'est pas trouvé, ***Symbol*** est mis à la position "Force Address (No Symbol)".
- **R_Regs** Lit les registres de l'unité centrale et présente les données dans un format facile à lire.

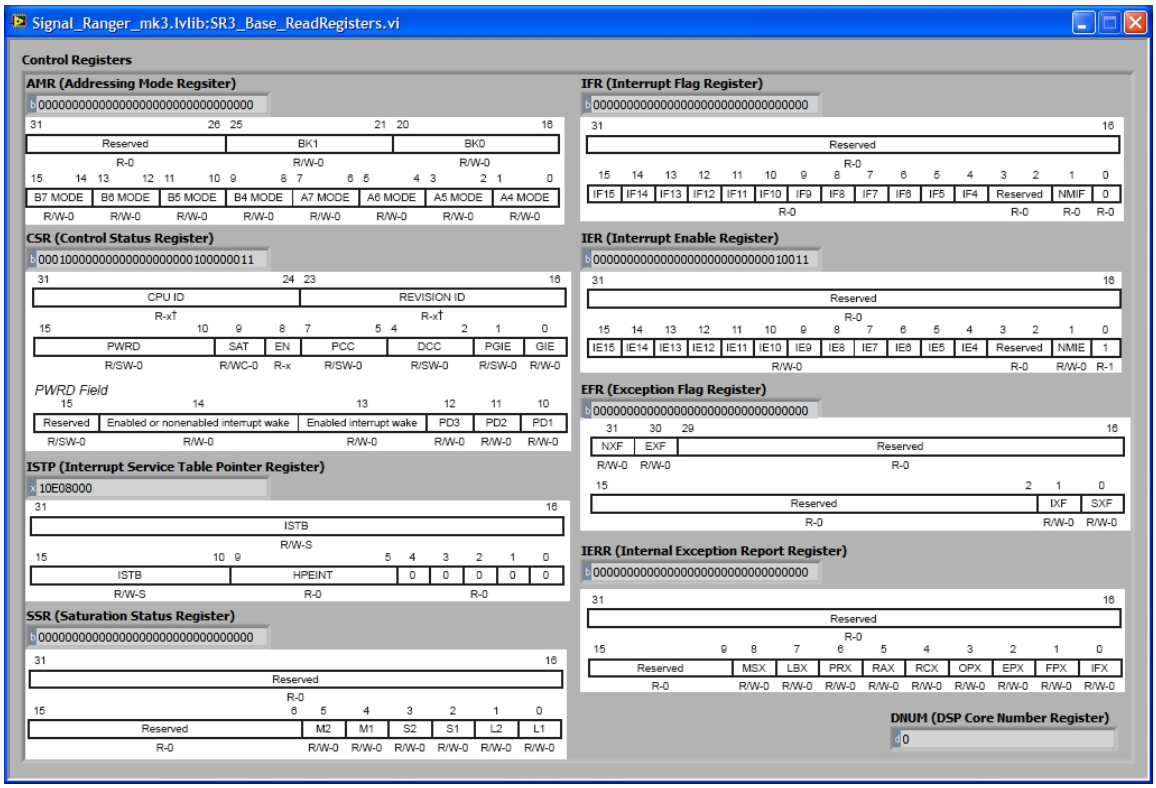


Figure 9 **Panneau de présentation du registre**

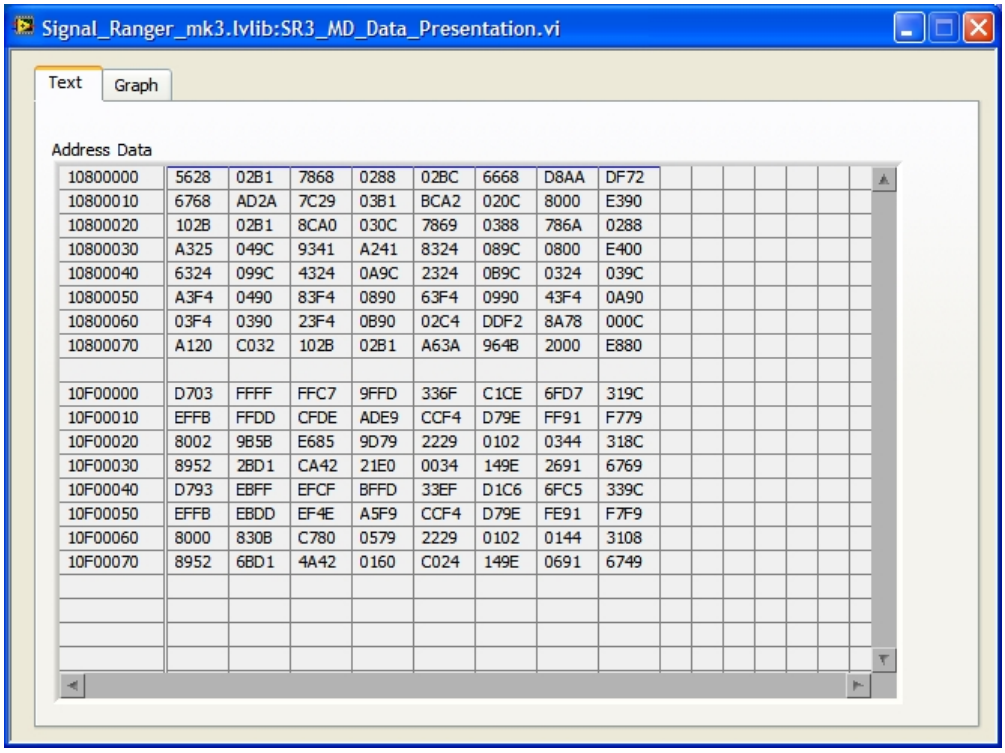
- **W_regs** Écriture/modification des registres de l'unité centrale. Cette fonction n'est pas supportée par tous les Les familles de *SignalRanger*.

- **R_Mem** Lit la mémoire du DSP et présente les données à l'utilisateur.
  Le petit bouton coulissant situé à côté du bouton permet une lecture continue. Pour arrêter la lecture continue, il suffit de replacer le curseur dans sa position par défaut.
  Le tableau des *paramètres d'affichage* est utilisé pour sélectionner un ou plusieurs blocs de mémoire à afficher. Chaque index du tableau sélectionne un bloc de mémoire différent.
  Pour ajouter un nouveau bloc de mémoire, il suffit d'avancer l'index à la valeur suivante et d'ajuster les paramètres d'affichage du nouveau bloc. Pour vider complètement le tableau, cliquez avec le bouton droit de la souris sur l'index et choisissez le menu "Vider le tableau". Pour insérer ou supprimer un bloc dans le tableau, il suffit d'avancer l'index jusqu'à la position correcte, de cliquer avec le bouton droit de la souris sur le champ *Symbole* et de choisir le menu "Insérer l'élément avant" ou "Supprimer l'élément".

  Pour chaque bloc :
  - **Symbole** ou **adresse** est utilisé pour spécifier le début du bloc de mémoire à afficher. Le **symbole** peut être utilisé si un fichier COFF contenant le symbole a été chargé précédemment. Si **Symbole** est réglé sur une position autre que "Forcer l'adresse (pas de symbole)", **Adresse** et **Espace mémoire** sont forcés à la valeur spécifiée dans le fichier COFF pour ce symbole.

Note : Les adresses spécifiées sont des adresses d'octets : Les adresses spécifiées sont des adresses d'octets.

  - ***Espace mémoire*** indique l'espace mémoire utilisé pour l'accès. La position " ???" (Inconnu) correspond par défaut à un accès dans l'espace de données. Si ***Symbol*** est défini sur un symbole spécifique, ***MemSpace*** est forcé à la valeur spécifiée dans le fichier COFF pour ce symbole. L'***espace mémoire*** peut ne pas être significatif pour certaines architectures.
  - ***Number*** spécifie le nombre d'éléments à afficher.
  - ***Type*** spécifie le type de données à afficher. Trois largeurs de base peuvent être utilisées : 8 bits, 16 bits et 32 bits. Toutes les largeurs peuvent être interprétées comme des données signées (*I8, I16, I32*), non signées (*U8, U16, U32*) ou à virgule flottante.
  - ***Format*** spécifie le format de présentation des données (hexadécimal, décimal ou binaire).
  - ***Échelle*** spécifie un facteur d'échelle pour la représentation graphique.
  - ***X ou 1/X*** indique si les données doivent être multipliées ou divisées par le facteur d'échelle.



Address								
10800000	5628	02B1	7868	0288	02BC	6668	D8AA	DF72
10800010	6768	AD2A	7C29	03B1	BCA2	020C	8000	E390
10800020	102B	02B1	8CA0	030C	7869	0388	786A	0288
10800030	A325	049C	9341	A241	8324	089C	0800	E400
10800040	6324	099C	4324	0A9C	2324	0B9C	0324	039C
10800050	A3F4	0490	83F4	0890	63F4	0990	43F4	0A90
10800060	03F4	0390	23F4	0B90	02C4	DDF2	8A78	000C
10800070	A120	C032	102B	02B1	A63A	964B	2000	E880
10F00000	D703	FFFF	FFC7	9FFD	336F	C1CE	6FD7	319C
10F00010	EFFB	FFDD	CFDE	ADE9	CCF4	D79E	FF91	F779
10F00020	8002	9B5B	E685	9D79	2229	0102	0344	318C
10F00030	8952	2BD1	CA42	21E0	0034	149E	2691	6769
10F00040	D793	EBFF	EFCF	BFFD	33EF	D1C6	6FC5	339C
10F00050	EFFB	EBDD	EF4E	A5F9	CCF4	D79E	FE91	F7F9
10F00060	8000	830B	C780	0579	2229	0102	0144	3108
10F00070	8952	6BD1	4A42	0160	C024	149E	0691	6749

Figure 10 **Présentation des données (mode texte)**

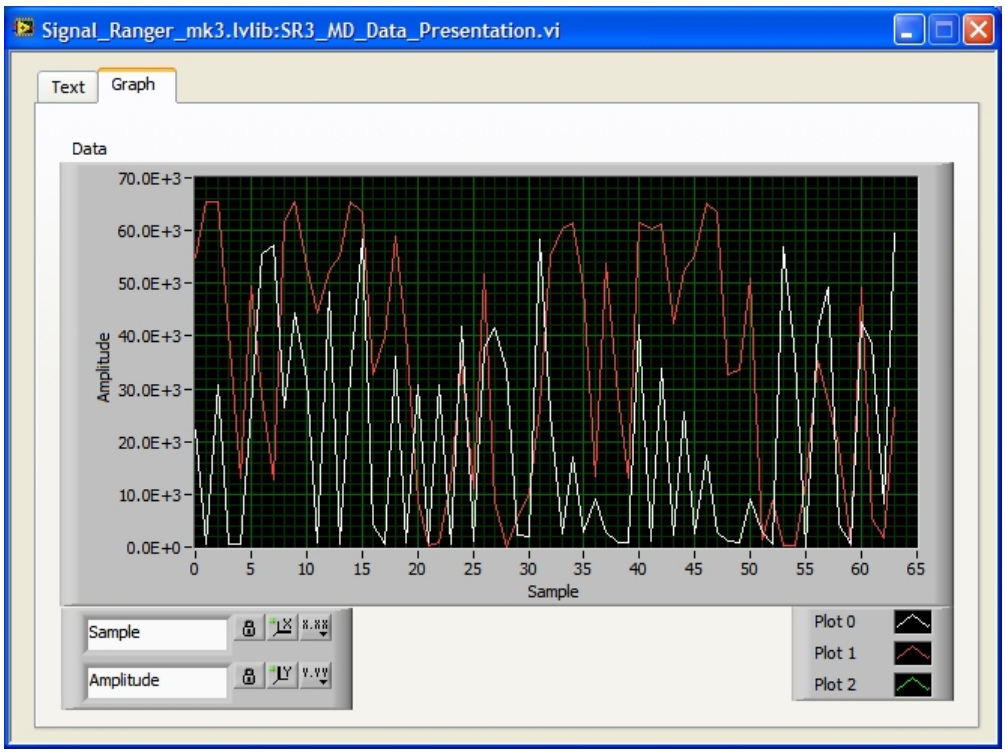


Figure 11 **Présentation des données (mode graphique)**

L'utilisateur peut choisir entre le mode *texte* (figure 10) et le mode *graphique* (figure 11) pour la présentation des données de la mémoire. En *mode texte*, chaque bloc de mémoire demandé est présenté sous forme de

séquence. Les adresses sont indiquées dans la première colonne. En mode *graphique*, chaque bloc de mémoire est mis à l'échelle et représenté par une ligne distincte du graphique.

- **W_Mem** Permet de lire et de modifier le contenu de la mémoire. La fonction lit d'abord la mémoire, en utilisant les *paramètres View_*, et présente un panneau de texte similaire à celui présenté pour la fonction *R_Mem.* L'utilisateur peut alors modifier n'importe quelle valeur dans le panneau et appuyer sur le bouton *Write* pour écrire les données dans la mémoire.
  Plusieurs points doivent être observés :
  - Même si la saisie de données est autorisée dans toutes les cellules du panneau, seules les cellules affichées pendant la phase de lecture (celles qui ne sont pas vides) sont prises en compte lors de l'écriture.
  - Les données doivent être saisies en utilisant le même type et le même format que ceux utilisés lors de la phase de lecture.
  - Pendant la phase d'écriture, TOUTES les données présentées dans le panneau sont réécrites dans la mémoire du DSP, et pas seulement les données qui ont été modifiées par l'utilisateur. Normalement, il s'agit des mêmes données que celles qui ont été lues, mais cela peut être important si les données changent en temps réel sur le DSP, car elles peuvent avoir changé entre la lecture et l'écriture.

  Note : Cette fonction peut être utilisée en option pour écrire des valeurs spécifiques à des adresses sélectionnées dans la mémoire Flash : Cette fonction peut être utilisée en option pour écrire des valeurs spécifiées à des adresses sélectionnées dans la mémoire Flash. Tous les secteurs Flash requis sont effacés avant l'écriture.

- **W_Flash** Permet au développeur de charger un code DSP et/ou une logique FPGA dans la mémoire Flash, ou d'effacer la mémoire. Le code DSP et/ou la logique FPGA spécifiés s'exécutent alors au démarrage.
  Le bouton *W_Flash* ouvre un navigateur qui permet au développeur de choisir les fichiers de code DSP (.out) et de logique FPGA (.rbt).
  L'opération réinitialise systématiquement le DSP et charge le code de support Flash nécessaire aux opérations de programmation Flash.
    - ***Fichier DSP*** Indique le chemin choisi pour le code DSP.
    - ***Nb Sections*** Indique le nombre de sections du code DSP à charger dans la ROM Flash. Les sections vides dans le fichier exécutable .out sont éliminées.
    - ***Point d'entrée*** Spécifie le point d'entrée du code, tel qu'il est défini dans le fichier COFF.
    - ***DSP_Load_Address*** Indique l'adresse de chargement de la table de démarrage dans la mémoire Flash.
    - DSP_Last_AddressIndique la dernière adresse de la table de démarrage dans la mémoire Flash.
    - ***Fichier FPGA*** Indique le chemin choisi pour le fichier logique FPGA.
    - ***Version des outils*** La version des outils ISE qui ont été utilisés pour générer le fichier .rbt.
    - ***Nom du dessin ou modèle*** Le nom du dessin ou modèle tel qu'il apparaît dans le fichier .rbt
    - ***Architecture*** Le type de FPGA pour lequel le fichier rbt est construit.
    - ***Dispositif*** Le numéro de modèle du FPGA pour lequel le fichier .rbt est construit.
    - ***Date*** Date de création du fichier .rbt.
    - ***FPGA_Load_Address*** C'est l'adresse du début de la table de démarrage du FPGA dans la mémoire Flash.
    - ***FPGA_Last_Address*** Il s'agit de la dernière adresse de la table de démarrage du FPGA dans la mémoire Flash. Elle est normalement de $1FFFF_H$.
    - ***Ecrire DSP*** Appuyez sur ce bouton pour sélectionner le code DSP et le graver dans la Flash. Les secteurs requis de la Flash sont effacés avant la programmation. Comme la Flash est effacée secteur par secteur, plutôt que mot par mot, l'effacement efface généralement plus de mots que ce qui est strictement nécessaire pour contenir le code DSP.
    - ***Ecrire FPGA*** Appuyez sur ce bouton pour sélectionner la logique FPGA et la graver dans la Flash. Les secteurs requis de la Flash sont effacés avant la programmation. Comme la Flash est effacée secteur par secteur, plutôt que mot par mot, la

efface généralement plus de mots que ce qui est strictement nécessaire pour contenir la logique du FPGA.

- ***Effacer DSP*** Appuyez sur ce bouton pour effacer le code DSP de la mémoire flash.
- ***Effacer FPGA*** Appuyez sur ce bouton pour effacer la logique FPGA de la Flash.
- ***Taille du flash*** Si la Flash est détectée, ce champ indique sa taille en mots clés.

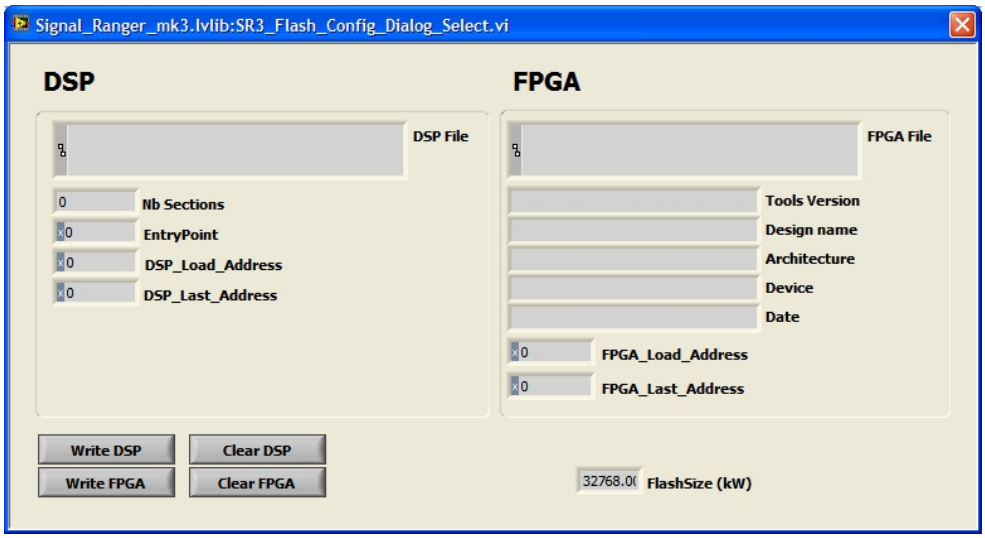


Figure 12

- **Chk_Flash** Ce bouton fait apparaître un panneau très similaire au bouton *W_Flash*. La seule différence est que cet utilitaire vérifie le contenu de la Flash par rapport aux fichiers spécifiés par l'utilisateur, au lieu de la programmer. Si aucun fichier n'est sélectionné pour le DSP et/ou le FPGA, la vérification correspondante est annulée et indique toujours un résultat positif.

Figure 13

- **Fichier DSP** Ce champ indique si un code DSP est présent dans la Flash embarquée. Lorsque ce champ indique un code DSP, les symboles associés sont généralement chargés à partir de la Flash et peuvent être utilisés pour interagir avec le code en cours d'exécution.
- **FPGA_File** Ce champ indique si la logique FPGA est présente dans la Flash embarquée et si elle est active. Ce champ est vide lorsque la carte est réinitialisée.

9 Interface USB LabVIEW

9.1 Remarques préliminaires

Cette nouvelle interface USB LabVIEW a été entièrement retravaillée pour prendre en charge deux architectures :

- Les anciennes cartes *SignalRanger_mk2*. Pour être contrôlées par la nouvelle interface, les anciennes cartes doivent avoir un nouveau *PID* qui leur permet d'être liées au nouveau pilote x86/x64. Ces cartes sont appelées *SignalRanger_mk2_Next_Generation* ou *SR2_NG*.
- Les nouvelles cartes SignalRanger_mk3, désignées *SR3*.

Comme il gère les deux architectures, certaines de ses caractéristiques peuvent ne pas s'appliquer à la carte cible. Par exemple, le concept d'*espaces de programme, de données et d'E/S* ne s'applique pas à l'architecture *SignalRanger_mk3*. Les paramètres correspondants sont simplement ignorés. De même, seule la carte *SignalRanger_mk3-Pro* dispose d'un FPGA.

9.2 Soutien au développement de produits

L'interface LabVIEW est conçue pour aider le concepteur à développer et à déployer des produits. Un produit comprend généralement un dispositif matériel basé sur l'une des cartes de la série *SignalRanger*, ainsi qu'une application hôte spécifique au produit, conçue pour gérer et prendre en charge le matériel.

Toute application spécifique à un produit doit connaître le produit matériel particulier auquel elle accède, y compris ses caractéristiques matérielles exactes, son micrologiciel et d'autres détails. Le fait de ne pas reconnaître la révision spécifique du micrologiciel, par exemple, peut entraîner une incohérence entre les fonctions au niveau de l'application hôte et les fonctions au niveau du DSP.

En outre, comme différents OEM utilisent la série *SignalRanger* comme base matérielle de différents produits, il arrive qu'une même carte cible représente un certain nombre de produits différents sur le terrain. Ces différents produits peuvent éventuellement être connectés simultanément au même PC et gérés par différentes applications spécifiques au produit, conçues par différents OEM.

Pour l'application hôte, la connaissance de la cible et de ses paramètres est basée sur les informations suivantes, intégrées dans la carte de la cible :

- **Une paire *VID/PID* USB :** Cette paire *VID/PID* est présente au niveau du protocole USB. Elle est utilisée pour ouvrir la carte cible. Les informations qu'elle contient sont donc implicites. Seules les cartes ayant la bonne paire *VID/PID* peuvent être ouvertes par l'application hôte. Ces informations définissent le type particulier de carte ouverte et, implicitement, ses caractéristiques matérielles. Cette paire *VID/PID* ne peut pas être modifiée par le développeur. Chaque carte d'un modèle donné de la série *SignalRanger* possède la même paire *VID/PID*, déterminée lors de la conception de la carte. Une base de données incluse dans l'interface LabVIEW contient des informations sur toutes les cartes standard et personnalisées qui ont été conçues. De nouvelles informations sont ajoutées à cette base de données au fur et à mesure que de nouvelles cartes sont conçues. Si une carte particulière n'est pas reconnue par l'interface LabVIEW, c'est généralement parce que le numéro de révision de l'interface LabVIEW est trop bas pour prendre en charge ce modèle de carte particulier.

- **Noms des fichiers du micrologiciel du DSP et du FPGA :** Lorsque le micrologiciel (code DSP et logique FPGA) est stocké en Flash, les noms exacts des fichiers correspondants sont également écrits. Cette information permet de restreindre la cible à un produit spécifique, et éventuellement à un numéro de révision si ce numéro est inclus dans les noms de fichiers. L'interface LabVIEW peut éventuellement limiter l'ouverture de la cible à une liste spécifique de noms. Cela permet à une application spécifique à un produit de n'ouvrir que les cibles qui correspondent au type de produit et éventuellement au numéro de révision du micrologiciel, et d'éviter d'ouvrir une cible qui pourrait correspondre à un produit d'un autre OEM.
- **UTC du micrologiciel du DSP :** Lorsque le micrologiciel du DSP est écrit dans la Flash, une somme de contrôle du contenu du micrologiciel est également écrite dans la Flash. Cette somme de contrôle est appliquée au contenu de la Flash, et non au contenu du fichier original.
- **FPGA UTC :** Lorsque la logique FPGA est écrite dans la Flash, une somme de contrôle du contenu FPGA est également écrite dans la Flash. Cette somme de contrôle est appliquée au contenu de la Flash, et non au contenu du fichier original.

9.3 Informations implicites sur la révision

Pour que l'application hôte puisse prendre le contrôle d'un produit sans ambiguïté, il doit y avoir des noms de fichiers firmware et FPGA uniques pour chaque révision du code DSP ou de la logique FPGA qui doit être distinguée. Si différentes révisions d'un fichier de micrologiciel utilisent le même nom, l'application hôte ne sera pas en mesure de les distinguer. Des incohérences apparaîtront entre les versions du microprogramme que l'hôte pense être intégrées dans la cible et le microprogramme réel intégré dans la cible. Ces incohérences peuvent conduire à des symboles manquants, des fonctions manquantes, des comportements différents, etc.

Un produit particulier est défini par sa paire de noms de code DSP et de fichier FPGA. Cette paire de noms définit complètement le produit et éventuellement sa révision. Si des informations sur la révision sont requises dans les noms de fichiers, un fichier
Le suffixe *_Vxyy* dans le nom du fichier est une bonne approche.

Remarque : si le code DSP ou la logique FPGA n'est pas programmé dans la mémoire Flash pour un produit donné, le nom du fichier correspondant enregistré dans la mémoire Flash est une chaîne vide. Si les deux fichiers sont absents, le produit risque de ne pas être correctement défini. Pour éviter cette situation, nous suggérons de charger dans Flash au moins un code DSP minimal qui ne fait rien d'autre que de retourner au noyau. Le nom de fichier de ce code sera choisi pour fournir une définition correcte du produit.

9.4 Stratégie suggérée de mise à jour du micrologiciel

Lors du déploiement d'une nouvelle révision du micrologiciel d'un même produit, deux opérations doivent être effectuées :

- Reprogrammer la cible embarquée avec les nouveaux fichiers du micrologiciel. Cette opération est réalisée soit par un outil manuel, soit par un outil automatisé qui reconnaît la cible et reprogramme sa mémoire flash. Ce nouveau micrologiciel est contenu dans de nouveaux fichiers DSP et/ou FPGA, éventuellement avec des noms uniques (s'ils doivent être distingués par l'application hôte).
- Installez une nouvelle application hôte pour gérer la nouvelle cible. Cette nouvelle application est au courant de la nouvelle version du micrologiciel. Elle exécute généralement de nouvelles fonctions qui sont activées par le nouveau micrologiciel.

Une méthode pratique permettant d'effectuer ces deux opérations de manière efficace consiste à concevoir une application hôte capable de.. :

- Reconnaître que le numéro de révision du micrologiciel de la cible connectée est inférieur au numéro de révision le plus élevé qu'elle est capable de gérer.
- Re-flasher la cible avec le numéro de révision le plus élevé qu'il a dans ses fichiers pour cette cible.

Selon cette approche, la procédure de mise à niveau serait la suivante :

- Installez d'abord la dernière version de la nouvelle application de gestion sur le PC hôte.

- Vous pouvez ensuite compter sur cette application pour reconnaître que la cible n'est pas à la dernière révision et suggérer un re-flash du firmware.

Pour faciliter cette opération, le fichier *SR3_Base_Open_Next_Avail_Board.vi* fournit les noms des fichiers du micrologiciel et les sommes de contrôle correspondantes.

9.5 Développement d'une application spécifique au produit

Sur la base des bibliothèques d'interface LabVIEW fournies, le développeur du produit doit concevoir et organiser une application hôte spécifique au produit. Pour ce faire, il est utile de comprendre les caractéristiques suivantes de la nouvelle interface LabVIEW :

9.5.1 Ouverture de la cible

L'interface LabVIEW peut gérer plusieurs connexions cibles simultanées. Chaque fois qu'une connexion cible est ouverte, le fichier *SR3_Base_Open_Next_Avail_Board*.vi crée une structure de données qui représente la carte cible et renvoie un indicateur *BoardRef_Out.* Cet indicateur *BoardRef_Out* est utilisé comme un handle sur la cible spécifique. Il représente la cible spécifique et la structure de données qui lui est associée. Tous les VIs de l'interface utilisent ce handle comme entrée.

Le handle fourni par le *SR3_Base_Open_Next_Avail_Board.vi* est exclusif. Une fois qu'une cible est ouverte, aucune autre application ne peut en prendre le contrôle jusqu'à ce qu'elle soit fermée par l'application en cours. Une cible particulière ne peut avoir qu'une seule connexion avec une application hôte.

Une application hôte, en revanche, peut avoir des connexions avec plusieurs cartes cibles et les gérer.

Le *SR3_Base_Open_Next_Avail_Board.vi* utilise une paire *VID/PID* pour ouvrir la cible. Cette paire *VID/PID* pointe vers une base de données dans l'interface LabVIEW qui contient les caractéristiques matérielles de la carte, telles que son modèle, le type de DSP, les adresses Flash... etc. Lorsque la cible est ouverte, ces informations sont stockées dans un tableau de variables globales. Il y a un élément de tableau, représentant une cible, pour chaque cible ouverte par l'hôte.

Après avoir ouvert la carte, le *SR3_Base_Open_Next_Avail_Board.vi* lit le code DSP et les noms des fichiers FPGA dans la Flash, s'il y en a, ainsi que leurs sommes de contrôle. Si un firmware DSP était présent dans la Flash, le *SR3_Base_Open_Next_Avail_Board.vi* charge alors la table de symboles qui suit le code dans la Flash. Cette table de symboles est utilisée pour fournir un accès symbolique au code DSP. Cette information est également stockée dans le tableau de variables globales afin que chaque VI de l'interface puisse l'utiliser.

Optionnellement, le *SR3_Base_Open_Next_Avail_Board.vi* peut ouvrir sélectivement seulement les cibles qui ont une paire de noms de fichiers DSP/FPGA qui fait partie d'une liste fournie. Cela permet d'ouvrir sélectivement les produits et les numéros de révision fournis dans la liste. En pratique, le *SR3_Base_Open_Next_Avail_Board.vi* ouvre brièvement la cible, lit les noms de fichiers depuis la Flash et referme la cible si les noms de fichiers ne font pas partie de la liste fournie.

9.5.2 Séquencement de l'exécution

Deux VIs de l'interface LabVIEW qui accèdent à la même carte *SignalRanger_mk3* DSP ne peuvent pas s'exécuter simultanément. Le premier VI doit être terminé avant que le second puisse être appelé. Comme LabVIEW exécute en parallèle des sections de code qui ne dépendent pas les unes des autres, il faut veiller à ce que les VIs qui accèdent à la carte ne puissent pas s'exécuter en même temps. Les VIs qui accèdent à la carte sont ceux qui ont un contrôle *BoardRef*. La technique la plus simple est de s'assurer que tous ces VIs sont enchaînés dans le diagramme en utilisant les connecteurs *BoardRef* et *dupBoardRef.* Cependant, les fonctions de l'interface accédant à différentes cartes (avec différentes valeurs *BoardRef*) peuvent être appelées simultanément si nécessaire.

Tous les VIs qui accèdent à la carte sont bloquants. Ils ne reviennent pas tant que l'action demandée n'a pas été effectuée sur la carte.

9.5.3 Chargement et exécution du code de manière dynamique

Les bibliothèques d'interface fournissent des fonctions permettant de charger dynamiquement le code DSP et la logique FPGA. Ceci est utile pour les applications dans lesquelles le matériel est multifonctionnel. Dans ce cas, l'application hôte (re)configure la plate-forme matérielle d'une manière spécifique à l'application après en avoir pris le contrôle. Elle est également utile dans les situations où des fonctions spéciales, qui ne sont normalement pas disponibles, doivent être mises en œuvre sur une base ad hoc. Il peut s'agir, par exemple, de fonctions de diagnostic nécessitant un code DSP spécial et/ou une logique FPGA.

Lorsque le code DSP est chargé dynamiquement, la table de symboles correspondante est chargée directement à partir du fichier COFF où le code est stocké.

Pour pouvoir charger le code DSP et/ou la logique FPGA de manière dynamique à partir du PC, les fichiers correspondants doivent respecter les règles de stockage et d'emplacement suivantes.

9.5.4 Règles de stockage et de localisation des microprogrammes

L'accès aux fichiers du micrologiciel est nécessaire pour permettre le chargement dynamique. Il existe deux méthodes pour stocker les fichiers du micrologiciel :

- **Dans un conteneur de micrologiciel VI** Les avantages de cette méthode sont les suivants :
    - Une fois que l'application hôte est intégrée dans un fichier ".exe", le fichier du micrologiciel cible n'est pas divulgué à l'utilisateur. Il est "caché" dans le binaire du code de l'application.
    - L'exécutable de l'application hôte est autonome. Il ne nécessite aucun fichier supplémentaire stocké sur le système hôte pour fonctionner.
- **En tant que fichier original ".out" pour le code DSP, et ".rbt" pour le FPGA.** Les avantages de cette méthode sont les suivants :
    - Il n'est pas nécessaire de construire le conteneur VI du micrologiciel. Pour être efficace, le fichier peut simplement être placé dans la hiérarchie des répertoires de l'application, ce qui permet une meilleure modularisation du code.
    - Lors de la reconstruction de l'application, il n'est pas nécessaire de prendre des mesures supplémentaires pour charger le nouveau code cible dans les VI conteneurs. Aucun VI dynamique ne doit être inclus dans le constructeur de l'application. Le constructeur de l'installateur rechargera simplement les fichiers du micrologiciel, qui reflètent les dernières modifications.

Une approche courante consiste à utiliser les fichiers .out et .rbt originaux pendant le développement, car cela facilite les tests des nouveaux microprogrammes et de la logique FPGA, et à encapsuler les fichiers dans des conteneurs de microprogrammes lorsque l'application est déployée.

Le processus de chargement du micrologiciel suit la logique ci-dessous :

- La fonction d'interface qui a besoin du microprogramme tentera d'abord de trouver un fichier ".out" et/ou ".rbt" portant le nom spécifié. La fonction d'interface recherche le fichier dans le répertoire situé un niveau au-dessus du niveau VI supérieur de l'application.
- En cas d'échec, l'interface tente alors de trouver un VI du conteneur de micrologiciel portant le même nom (après avoir supprimé l'extension .vi) dans le même répertoire que le VI de premier niveau de l'application.
- Si ce n'est pas le cas, il renvoie une erreur.

Dans la pratique, ces règles fournissent des emplacements naturels dans les deux conditions ci-dessous :

- Lorsque l'on utilise des VIs conteneurs de firmware, les fichiers firmware sont situés dans le même répertoire que le VI de premier niveau pendant le développement. Pour un exécutable construit, les VIs conteneurs de firmware sont naturellement situés au même niveau que le VI de premier niveau, à l'intérieur du fichier exécutable.
- Lors de l'utilisation de fichiers .out et .rbt standard, les fichiers du microprogramme doivent être situés un niveau au-dessus du VI de premier niveau pendant le développement. Dans le cas d'un exécutable construit, les fichiers du microprogramme doivent être situés dans le répertoire

au même niveau que le fichier exécutable, c'est-à-dire un niveau de répertoire au-dessus du VI de premier niveau intégré dans l'exécutable.

Note : Les fichiers originaux ".out" et ".rbt" ont la priorité sur les VIs conteneurs. Il est donc facile de remplacer un VI conteneur existant en ajoutant simplement un nouveau fichier .out ou .rbt portant le même nom dans le répertoire situé au-dessus du VI de premier niveau. Ceci est vrai même après qu'une application ait été construite en tant qu'exécutable. Tout fichier .out ou .rbt ajouté dans le même répertoire que l'exécutable remplacera effectivement tout VI intégré dans le conteneur. Cela permet de mettre à jour le micrologiciel sur le terrain ou en cours de développement sans avoir à reconstruire l'exécutable.

9.5.4.1 Firmware stocké dans un conteneur de firmware Vis

Le contenu de chaque fichier de micrologiciel est stocké dans un VI sous la forme d'un indicateur de chaîne de caractères. Ces VIs conteneurs de firmware sont stockés avec les autres VIs spécifiques au produit. Un VI utilitaire nommé *Create_Firmware_Container.vi* est fourni dans la bibliothèque d'interface LabVIEW pour créer ces VI conteneurs et initialiser l'indicateur de chaîne. Les VI sont chargés dynamiquement par la bibliothèque d'interface LabVIEW lorsque cela est nécessaire.

9.5.4.1.1 Création des VIs du conteneur de micrologiciel

Pour utiliser cette méthode, les VIs contenant le firmware doivent être créés avec les noms de fichiers exacts indiqués dans la cible Flash, sauf qu'ils ont une extension .vi après l'extension ".out" ou ".rbt". Les conteneurs de micrologiciel doivent être stockés dans la même hiérarchie que le VI d'application supérieur.

La figure suivante montre le panneau avant du *fichier Create_Firmware_Container.vi*. Ce VI est utilisé pour créer un conteneur pour un fichier ".out" pour le code DSP, ainsi qu'un fichier ".rbt" pour la logique FPGA.

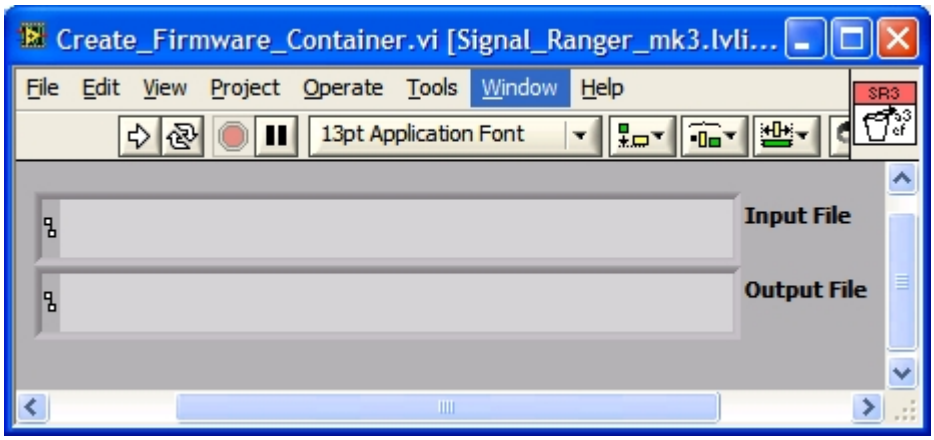


Figure 14 ***Create_Firmware_Container.vi***

Pour utiliser ce VI, suivez les étapes ci-dessous :

1. Exécuter le VI
2. A l'invite, sélectionnez le fichier ".out" ou ".rbt" contenant les données.
3. Le VI crée un VI conteneur portant le même nom que le fichier d'entrée et ajoute l'extension ".vi" à la fin.
4. A l'invite, enregistrez le VI du conteneur dans le répertoire approprié (voir *Emplacement des VI du conteneur de micrologiciel).*
   ci-dessous).

9.5.4.1.2 Emplacement des VI du conteneur de micrologiciel

Les VIs du conteneur de firmware doivent être situés dans le même répertoire que le VI de premier niveau de l'application qui les utilise.

9.5.4.2 Firmware stocké sous forme de fichiers binaires

Les fichiers du microprogramme sont stockés sur l'hôte sous forme de fichiers originaux ".out" et ".rbt". Les fichiers du micrologiciel sont chargés par la bibliothèque d'interface LabVIEW lorsque cela est nécessaire.

9.5.4.2.1 Emplacement des fichiers du micrologiciel

Les fichiers ".out" et ".rbt" doivent se trouver dans le répertoire situé un niveau au-dessus de l'application VI de premier niveau. Notez qu'il s'agit d'un niveau supérieur à celui où se trouvent les VIs conteneurs. Les fichiers doivent porter le nom exact qui apparaît dans le Flash de la cible.

9.5.5 Création d'un exécutable LabVIEW

Les éléments suivants doivent être inclus dans la construction d'un exécutable spécifique à une application :

- Si des VIs contenant le firmware sont utilisés pour contenir les différents codes DSP et les fichiers FPGA, ils doivent être explicitement inclus dans le build. Ils doivent être inclus dans le champ *Always-Included* de la section *Source Files*. Les VIs ne sont pas automatiquement inclus dans le build car ils sont chargés dynamiquement par les VIs de l'interface LabVIEW. Les VIs du conteneur de firmware doivent être reconstruits, ou de nouveaux VIs doivent être créés avec le firmware le plus récent chaque fois qu'un nouveau firmware est en vigueur dans Flash. Cette étape doit être réalisée avant que l'application exécutable ne soit reconstruite.
- Si des fichiers .out ou .rbt standard sont utilisés, aucune démarche particulière n'est nécessaire.

9.5.6 Création d'un programme d'installation

Les éléments suivants doivent être inclus dans la construction d'un exécutable spécifique à une application :

- Lorsque le code DSP et les fichiers FPGA sont stockés sous forme de fichiers standard ".out" et ".rbt", ces fichiers doivent être inclus dans la spécification du programme d'installation de l'application et être installés dans le même répertoire que l'application exécutable. Cet emplacement correspond en fait à un niveau au-dessus du VI de premier niveau qui constitue l'application. Chaque fois que le programme d'installation est reconstruit, il recharge automatiquement les derniers fichiers du micrologiciel si ceux-ci ont été modifiés depuis la dernière reconstruction.
- Lorsque l'on utilise des VIs conteneurs pour contenir le code DSP et la logique FPGA, les VIs font déjà partie de l'exécutable construit. L'installateur n'a rien d'autre à faire.
- La dernière version du moteur d'exécution NI-VISA doit être incluse dans le programme d'installation.

9.5.7 Firmware de support requis

Quelques fichiers de microprogrammes sont nécessaires pour prendre en charge un VI ou un exécutable intégré. Ces fichiers doivent être stockés conformément aux *règles de stockage et d'emplacement des microprogrammes* décrites ci-dessus. Ils doivent être inclus dans la construction d'un exécutable ou d'un installateur, selon le type de stockage (micrologiciel d'origine ou VI conteneur).

9.5.7.1 SR2_NG Plate-forme

- **SR2Kernel_HostDownload.out** — Toujours requis
- **SR2Kernel_PowerUp.out** — Toujours nécessaire
- **SR2_Flash_Support.out** — Nécessaire pour utiliser les VIs Flash-Support
- **SR2_FPGA_Support.out** — Nécessaire pour utiliser les VIs FPGA-Support

Des fichiers .out ou .rbt spécifiques à l'application peuvent également être nécessaires.

9.5.7.2 Plate-forme SR3

- **SR3Kernel_HostDownload.out** — Toujours requis

- **SR3Kernel_PowerUp.out** Toujours nécessaire
- **SR3_Flash_Support.out** Nécessaire pour utiliser les VIs Flash-Support

Des fichiers .out ou .rbt spécifiques à l'application peuvent également être nécessaires.

9.6 Interface LabVIEW Vis

L'interface LabVIEW est organisée en plusieurs dossiers dans la bibliothèque *Signal_Ranger_mk3.lvlib*. Toutes les bibliothèques dont le nom se termine par "_U" contiennent des Vis de support et il n'est pas prévu que le développeur doive utiliser des Vis individuelles dans ces bibliothèques.

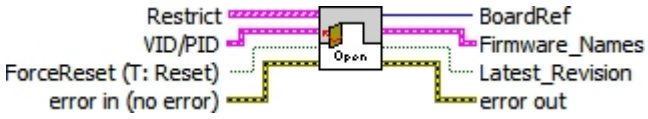
Dans l'ensemble, l'interface LabVIEW permet au développeur d'exploiter les capacités de traitement en temps réel et d'E/S numériques *du Signal_Ranger_mk3*, avec la facilité d'utilisation, la puissance de traitement de haut niveau et l'interface utilisateur graphique de LabVIEW.

9.6.1 Interface principale VIs

9.6.1.1 SR3_Base_Open_Next_Avail_Board

Cette Vi effectue les opérations suivantes :

- Essaie de trouver une carte DSP libre avec le VID/PID sélectionné, et éventuellement avec le firmware indiqué dans le *Restrict* control.
- S'il en trouve un, il crée une entrée dans la *structure d'information globale du conseil d'administration*.
- Attend que le noyau de mise sous tension soit chargé sur la carte.
- Si un microprogramme DSP est détecté dans la Flash (le code a été chargé et démarré dans le cadre de la séquence de mise sous tension), charge le nom de fichier correspondant et la table de symboles à partir de la Flash.
- Si *ForceReset* est vrai, il force la réinitialisation du DSP, puis recharge le noyau Host-Download. Dans ce cas, tout le code présent dans la Flash et exécuté à la mise sous tension est annulé et la table de symboles correspondante trouvée dans la Flash n'est pas chargée.
- Place la table des symboles du noyau actuel dans la *structure d'information globale de la carte*.



Contrôles :

- **Restreindre** : Il s'agit d'une structure utilisée pour restreindre l'accès par noms de microprogrammes. Si ce contrôle n'est pas vide, l'accès est limité aux cartes ayant une paire de noms de fichiers DSP et FPGA dans la liste fournie. Chaque élément du tableau est une paire de noms de fichiers. Le micrologiciel en Flash doit correspondre aux deux noms d'un élément du tableau pour que la carte soit acceptée. Cette commande doit être câblée de manière à limiter l'ouverture aux cartes qui ont été configurées comme des produits spécifiques et à éviter d'ouvrir des cartes utilisées par d'autres OEM ou d'autres produits du même OEM.
- **VID/PID** : Il s'agit d'une structure utilisée pour sélectionner le type de carte à utiliser. Il existe plusieurs variantes matérielles de l'architecture *SignalRanger_mk3*, y compris des implémentations personnalisées. Chaque type de carte de la série possède sa propre paire *VID/PID*. Utilisez la paire *VID/PID* appropriée pour ouvrir la carte adéquate. Consultez les exemples pour trouver la paire *VID/PID* correspondant à votre carte. En cas de doute, contactez *Soft-dB*.
- **ForceReset** : Si la valeur est vraie, le DSP est réinitialisé et le *noyau de téléchargement de l'hôte* est chargé. Tout le code DSP en cours d'exécution est interrompu. Utilisez ce paramètre pour recharger dynamiquement le nouveau code DSP dans la carte. Utilisez le paramètre *false* pour prendre le contrôle du code DSP déjà en cours d'exécution à partir de la Flash sans l'interrompre.

- **Erreur dans** Groupe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur. Ne pas le câbler s'il s'agit du premier VI d'interface de la séquence.

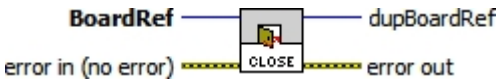
Indicateurs :

- **BoardRef** : Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la *structure globale d'information sur les cartes*. L'interface peut gérer une multitude de cartes connectées au même PC. Un numéro *BoardRef* correspondant est attribué à chacune d'entre elles lorsqu'elles sont ouvertes. Tous les autres Vis de l'interface utilisent ce numéro pour accéder à la carte appropriée.
- **Firmware_Names** : Grappe contenant les noms des fichiers de microprogrammes du DSP et du FPGA qui se trouvent dans la Flash. Les champs sont vides si la Flash ne contient aucun micrologiciel. Les champs sont également vides si la commande *ForceReset* est vraie.
- **Latest_Revision** : Cet indicateur est *vrai* si la paire de noms de fichiers de microprogrammes trouvés dans Flash correspond au dernier élément fourni dans le tableau *Restrict*. Dans certaines implémentations, le tableau *Restrict* contient les paires de noms des différentes révisions du microprogramme d'un produit donné, par ordre croissant. Dans ce cas, l'indicateur *Latest_Revision* est vrai lorsque le micrologiciel détecté dans la mémoire flash de la carte est effectivement le dernier micrologiciel connu de l'application de contrôle.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

Note : La poignée que l'interface fournit pour accéder à la carte est exclusive. Cela signifie qu'une seule application à la fois peut ouvrir et gérer un tableau. Il en résulte qu'une carte ne peut pas être ouverte deux fois. Une carte qui a déjà été ouverte à l'aide du VI SR3_Base_Open_Next_Avail_Board ne peut pas être ouverte à nouveau tant qu'elle n'a pas été correctement fermée à l'aide du VI SR3_Base_Close_BoardNb. Ceci est particulièrement préoccupant lorsque l'application gérant la carte est fermée dans des conditions anormales. Si l'application est fermée sans fermer correctement la carte, l'exécution suivante de l'application peut échouer à trouver et ouvrir la carte, simplement parce que l'instance de pilote correspondante est toujours ouverte. Dans ce cas, il suffit de déconnecter et de reconnecter la carte pour forcer le PC à ré-énumérer la carte.

9.6.1.2 SR3_Base_Close_BoardNb

Cet Vi ferme l'instance du pilote utilisé pour accéder à la carte et supprime l'entrée correspondante dans la *structure d'information globale de la carte*. Utilisez-la après le dernier accès à la carte, pour libérer les ressources qui ne sont plus utilisées.



Contrôles :

- **BoardRef** : Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description de l'erreur précédente.

Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.1.3 SR3_Base_Complete_DSP_Reset

Ce VI effectue les opérations suivantes :

- La LED clignote temporairement en orange
- Réinitialise le DSP
- Réinitialise le PCSI
- Charge le noyau Host-Download

Ces opérations sont nécessaires pour prendre complètement le contrôle d'un DSP qui exécute un autre code ou qui s'est planté. L'opération complète prend 500 ms.

Contrôles :

- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description de l'erreur précédente.

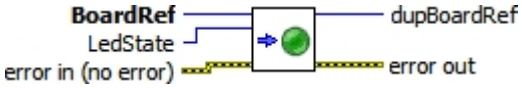
Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.1.4 SR3_Base_WriteLeds

Ce Vi permet l'activation sélective de chaque élément de la Led bicolore.

- Arrêt
- Rouge
- Vert
- Orange



Contrôles :

- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*

- **LedState** : Cette énumération spécifie l'état des DEL (rouge, vert, orange ou éteint).
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description de l'erreur précédente.

Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.1.5 SR3_Base_Bulk_Move_Offset

Ce VI lit ou écrit un nombre illimité de mots de données vers/depuis l'espace de programme, de données ou d'E/S du DSP, en utilisant le noyau. Ce transfert utilise des tuyaux en vrac. Cela se traduit par une bande passante élevée.

Le VI est polymorphe et permet les transferts des types suivants :

- Octets signés de 8 bits (I8), ou tableaux de ce type.
- Octets non signés de 8 bits (U8), ou tableaux de ce type.
- Mots de 16 bits signés (I16) ou tableaux de ce type.
- Mots de 16 bits non signés (U16) ou tableaux de ce type.
- Mots de 32 bits signés (I32) ou tableaux de ce type.
- Mots non signés de 32 bits (U32) ou tableaux de ce type.
- les nombres à virgule flottante de 32 bits (float) ou les tableaux de ce type.
- Cordes

Ils représentent tous les types de données de base utilisés par le compilateur C pour le DSP.

Pour transférer tout autre type (structures par exemple), la méthode la plus simple consiste à utiliser un "cast" pour permettre à ce type d'être représenté comme un tableau de U8 du côté du DSP (cast du type requis vers un tableau de U8 pour l'écrire dans le DSP, lecture d'un tableau de U8 et cast du type requis pour une lecture).

L'adresse du DSP et l'espace mémoire du transfert sont spécifiés comme suit :

- Si *Symbol* est câblé et que le symbole est représenté dans la table des symboles, le transfert a lieu à l'adresse et dans l'espace mémoire correspondant à *Symbol*. Notez que *Symbole* doit représenter une adresse valide. En outre, le fichier DSP COFF doit être lié avec la convention habituelle de numéro de page :
  - Espace programme = numéro de page 0
  - Espace de données = page numéro 1
  - Espace OI = page numéro 2
  - Tous les autres numéros de page sont accessibles en tant qu'espace de données.
- Si *le symbole* n'est pas câblé, *DSPAddress* est utilisé comme adresse d'octet pour le transfert, et *MemSpace* est utilisé comme espace mémoire.
- Il convient de noter que *DSPAddress* peut être aligné sur la largeur appropriée, en fonction de l'architecture.
- La valeur du *décalage* est ajoutée à *DSPAddress*. Cette fonctionnalité est utile pour accéder à des membres individuels de structures ou de tableaux sur le DSP. Notez que la valeur de *Offset* est toujours comptée en octets, et non en éléments du type spécifié. Ceci est nécessaire pour accéder à un membre individuel d'une structure hétérogène.
- En cas d'écriture d'un type de données plus étroit que le type natif de la plate-forme, des éléments supplémentaires sont ajoutés pour compléter l'écriture jusqu'à la frontière suivante du type natif. Les valeurs ajoutées prennent la valeur FF_H. Cela ne se produit pas sur *SignalRanger_mk3* puisque le type natif est l'octet.

Remarque : Le VI étant polymorphe, la lecture d'un type spécifique nécessite que ce type soit câblé à l'entrée DataIn. Ceci force simplement le type pour l'opération de lecture.

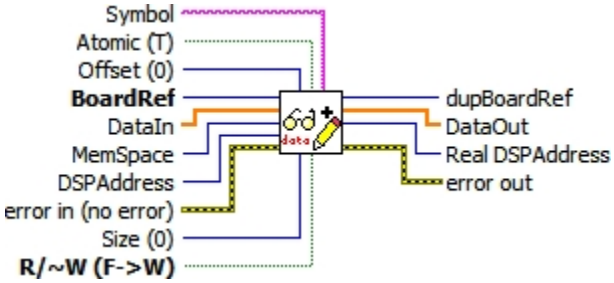
Remarque : Lors de la lecture ou de l'écriture de scalaires, la taille ne doit pas être câblée.

9.6.1.5.1 Notes sur l'atomicité de transfert

Lors de la lecture ou de l'écriture de types plus grands que le type natif de la plate-forme, le PC effectue plusieurs accès séparés pour chaque type long transféré. En principe, il est possible que le DSP ou l'hôte accède à un mot au milieu de l'échange, corrompant ainsi les données. Par exemple, lors d'une lecture sur une plate-forme où le type natif est U16, l'hôte pourrait télécharger une valeur à virgule flottante juste après que le processeur de signal numérique a mis à jour un mot de 16 bits constituant la virgule flottante, mais avant qu'il n'ait mis à jour l'autre. Il est évident que la valeur lue par l'hôte serait complètement erronée. Symétriquement, lors d'une écriture, l'hôte pourrait modifier les deux mots de 16 bits constituant un flottant dans la mémoire du DSP, juste après que le DSP ait lu le premier, mais avant qu'il n'ait lu le second. Dans cette situation, le processeur travaille avec une "ancienne" version d'une moitié du flottant et une nouvelle version de l'autre moitié.
Ces problèmes peuvent être évités si l'on comprend les faits suivants :

Sur la plate-forme *SignalRanger_mk2_Next_Generation*, lorsque le PC accède à un groupe de valeurs, il le fait toujours par blocs de 32 mots de 16 bits à la fois (jusqu'à 256 mots si la carte est connectée à un concentrateur USB à haute vitesse ou à une racine). Chacun de ces accès aux blocs est atomique. Le DSP est ininterrompu et ne peut effectuer aucune opération au milieu d'un bloc du transfert PC. Par conséquent, le DSP ne peut pas "interférer" au milieu d'un accès unique de 32 ou 256 blocs par le PC. Cela ne suffit pas à garantir l'intégrité des valeurs transférées, car le PC peut toujours transférer un bloc complet de données au milieu d'une autre opération concurrente du DSP sur ces mêmes données. Pour éviter cette situation, il suffit de rendre atomique toute opération DSP sur des données de 32 bits qui pourraient être modifiées par le PC. Ceci peut être facilement réalisé en désactivant les interruptions DSPInt pour la durée de l'opération. Les accès au PC sont alors atomiques des deux côtés et les données peuvent être transférées en toute sécurité 32 bits à la fois. Sur cette plate-forme, les transferts sont toujours atomiques du côté du PC. Le contrôle *atomique* est présent pour des raisons de compatibilité mais n'a aucun effet.

Sur la plate-forme *SignalRanger_mk3*, l'utilisateur a le choix entre les transferts atomiques et les transferts non atomiques. L'utilisation d'un transfert atomique présente l'avantage que, du point de vue du DSP, toutes les parties du bloc sont transférées simultanément. Ce comportement est compatible avec celui de la plate-forme *SignalRanger_mk2_Next_Generation*. Les transferts non atomiques présentent l'avantage que les tâches critiques du DSP peuvent interrompre le transfert et donc avoir la priorité sur le transfert USB. La seule façon d'utiliser un transfert non atomique sur *SignalRanger_mk2_Next_Generation* est d'utiliser une fonction utilisateur personnalisée et le VI *SR3_Base_User_Move_Offset*.



Contrôles :

- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **DataIn** : Mots de données à écrire dans la mémoire du DSP. *DataIn* doit être câblé, même pour une lecture, afin de spécifier le type de données à transférer.
- **MemSpace** : Espace mémoire pour l'échange (données, programme ou IO). *MemSpace* n'est utilisé que si *Symbol* est vide ou non câblé.
- **DSPAddress** : Adresse physique de base du DSP pour l'échange. *DSPAddress* n'est utilisé que si *Le symbole* est vide ou non câblé.
- **Size (taille)** : Utilisé uniquement pour les lectures de types de tableaux, représente la taille (en nombre d'éléments du type de données demandé) du tableau à lire dans la mémoire du processeur. Pour les écritures, tout le contenu de *DataIn* est écrit dans la mémoire du processeur, quelle que soit la *taille*. Lorsque la *taille* est câblée, les données ne peuvent être transférées que sous forme de tableaux, et non de scalaires.
- **Symbole** : Chaîne de caractères du symbole auquel il faut accéder. Si *le symbole* est vide ou non connecté, *DSPAddress* et *MemSpace* sont utilisés.
- **Offset** : Représente le décalage des données à accéder par rapport à l'adresse de base indiquée par *Symbol* ou *DSPAddress*. L'adresse d'accès réelle est calculée comme la somme de l'adresse de base et du décalage. L'*offset* est utile pour accéder à des membres individuels d'une structure ou d'un tableau.
- **R/~W** : Booléen indiquant le sens du transfert (true->read, false->write).
- **Atomique** : Booléen indiquant si le transfert est rendu atomique ou non. Par défaut, le transfert est toujours atomique.
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description de l'erreur précédente.

Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **DataOut** : Données lues dans la mémoire du DSP.
- **Real DSPAddress** : Adresse réelle où le transfert a eu lieu. Cette adresse tient compte de la résolution du *symbole* (s'il est utilisé) et de l'effet du *décalage*.
- **ErrorCode** : Il s'agit du code d'erreur renvoyé par la fonction du noyau qui est exécutée. La valeur de cet indicateur n'est pas pertinente dans cette interface.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.1.6 SR3_Base_User_Move_Offset

Ce VI est similaire à *SR3_Base_Bulk_Move_Offset*, sauf qu'il permet à une fonction DSP définie par l'utilisateur de remplacer la fonction noyau intrinsèque utilisée par *SR3_Base_Bulk_Move_Offset.*

Le fonctionnement du contrôleur USB et du noyau permet à une fonction DSP définie par l'utilisateur de remplacer les fonctions intrinsèques du noyau (voir la documentation du noyau ci-dessous). Pour cela, la fonction DSP définie par l'utilisateur doit effectuer les mêmes actions avec la boîte aux lettres que la fonction intrinsèque du noyau (lecture ou écriture du noyau). Cela peut être utile pour définir de nouvelles fonctions de transfert avec une fonctionnalité spécifique à l'application. Par exemple, une fonction de lecture ou d'écriture d'une FIFO pourrait être définie de cette manière. Outre la fonctionnalité de transfert de données, une fonction de lecture ou d'écriture FIFO inclurait également la gestion nécessaire des pointeurs, qui n'est pas présente dans les fonctions intrinsèques du noyau.

En conséquence, *SR3_Base_User_Move_Offset* comprend deux contrôles pour définir le point d'entrée de la fonction qui doit être utilisée pour remplacer la fonction intrinsèque du noyau.

Un transfert d'un nombre de mots supérieur à 32 (supérieur à 256 pour une connexion USB à haut débit) est segmenté en autant de transferts de 32 mots (256 mots) que nécessaire. La fonction définie par l'utilisateur est appelée à chaque nouveau segment. Si le nombre total de mots à transférer n'est pas un multiple de 32 (256), le dernier segment contient le reste.

La fonction définie par l'utilisateur doit être créée pour fonctionner de la même manière que les fonctions de lecture et d'écriture du noyau. En d'autres termes, elle doit effectuer les mêmes transferts, l'entretien de la boîte aux lettres et les échanges que les fonctions du noyau :

9.6.1.6.1 SR2_NG Plate-forme

- S'il s'agit d'un transfert, la fonction transfère les mots nécessaires vers ou depuis la zone de *données* de la boîte aux lettres. Le nombre de mots à transférer est la valeur du champ *NbWords* de la boîte aux lettres. Dans SR2_NG, ce champ représente le nombre de mots à transférer dans ce seul segment, et non le nombre total de mots à transférer pour l'ensemble de la requête USB. Ceci est différent du cas SR3.
- Le champ *TransferAddress* peut avoir besoin d'être incrémenté, selon la manière dont la fonction utilise cette information. En règle générale, les fonctions du noyau *K_Read* et *K_Write* incrémentent *TransferAddress* de manière à ce que le segment suivant soit transféré vers/depuis l'adresse mémoire suivante. Cependant, certaines fonctions utilisateur peuvent utiliser ce champ d'une manière différente. Par exemple, il peut être utilisé comme un numéro de tampon FIFO arbitraire dans certaines implémentations. Pour *K_Read* et *K_Write, TransferAddress* représente un nombre d'octets.
- Après l'exécution de la fonction demandée, le DSP active le signal HINT pour signaler au contrôleur USB que l'opération est terminée. Cette opération a été définie de manière pratique dans une macro *Acquittement* dans les codes d'exemple, et peut être insérée à la fin de n'importe quelle fonction utilisateur. Notez que du point de vue du PC, la commande semble "suspendue" jusqu'à ce que l'accusé de réception soit émis par le DSP. Le code utilisateur ne doit pas prendre trop de temps avant d'émettre l'accusé de réception.

9.6.1.6.2 Plate-forme SR3

- S'il s'agit d'un transfert, la fonction doit lire le *code de contrôle* (adresse de boîte aux lettres 0x10F0400A), masquer les deux bits inférieurs qui représentent le type d'opération. Le résultat, appelé *USBTransferSize*, représente le nombre maximum d'octets qui doivent être transférés dans ce segment. Notez que le contenu du champ *ControlCode* de la boîte aux lettres ne doit pas être modifié.
- Si nécessaire, la fonction transfère les octets requis vers ou depuis la zone de *données* de la boîte aux lettres. Le nombre d'octets à transférer est le plus petit entre le champ *NbBytes* de la boîte aux lettres et le résultat *USBTransferSize* qui vient d'être calculé.
- Enfin, le nombre d'octets transférés doit être soustrait de la valeur *NbBytes*, et la valeur
  Le champ *NbBytes* doit être mis à jour avec la nouvelle valeur dans la boîte aux lettres.
- Le champ *TransferAddress* peut avoir besoin d'être incrémenté, selon la manière dont la fonction utilise cette information. En règle générale, les fonctions du noyau *K_Read* et *K_Write* incrémentent *TransferAddress* de manière à ce que le segment suivant soit transféré vers/depuis l'adresse mémoire suivante. Cependant, certaines fonctions utilisateur peuvent utiliser ce champ d'une manière différente. Par exemple, il peut être utilisé comme un numéro de tampon FIFO arbitraire dans certaines implémentations. Pour *K_Read* et *K_Write, TransferAddress* représente un nombre d'octets.
- Après l'exécution de la fonction demandée, le DSP active le signal HINT pour signaler au contrôleur USB que l'opération est terminée. Cette opération a été définie de manière pratique dans une macro *Acquittement* dans les codes d'exemple, et peut être insérée à la fin de n'importe quelle fonction utilisateur. Notez que du point de vue du PC, la commande semble "suspendue" jusqu'à ce que l'accusé de réception soit émis par le DSP. Le code utilisateur ne doit pas prendre trop de temps avant d'émettre l'accusé de réception.

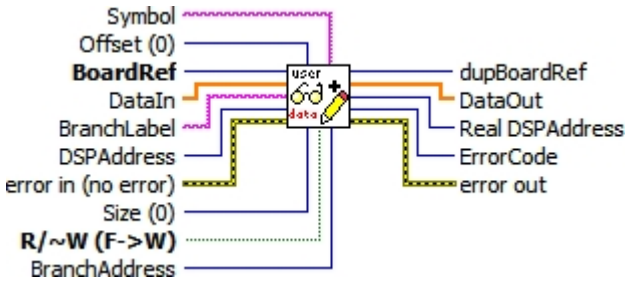
Pour plus d'informations, voir la section sur le fonctionnement du noyau.

Remarque : Sur SignalRanger_mk2_Next_Generation, si TransferAddress est utilisé pour transporter des informations autres qu'une adresse de transfert réelle, les restrictions suivantes s'appliquent :

- La taille totale du transfert doit être inférieure ou égale à 32768 mots. En effet, les transferts sont segmentés en transferts de 32768 mots à un niveau supérieur. Les informations contenues dans TransferAddress sont

n'est préservé que pendant le premier de ces segments de niveau supérieur. Au segment suivant, TransferAddress est mis à jour comme s'il s'agissait d'une adresse pointant vers le bloc suivant.

- Le transfert ne doit pas franchir une limite de 64 kWord. Les transferts qui franchissent une limite de 64 kWord sont divisés en deux transferts consécutifs. Les informations contenues dans TransferAddress ne sont conservées que pendant le premier de ces segments de niveau supérieur. Au suivant, TransferAddress est mis à jour comme s'il s'agissait d'une adresse pointant vers le bloc suivant
- TransferAddress doit être paire. Elle est considérée comme une adresse de transfert d'octets, c'est pourquoi son bit 0 est masqué à haut niveau.



Contrôles :

- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **DataIn** : Mots de données à écrire dans la mémoire du DSP. *DataIn* doit être câblé, même pour une lecture, afin de spécifier le type de données à transférer.
- **DSPAddress** : Adresse physique de base du DSP pour l'échange. *DSPAddress* n'est utilisé que si *Symbol* est vide ou non câblé. *DSPAddress* est écrit dans le champ *TransferAddress* de la boîte aux lettres avant le premier appel de la fonction DSP définie par l'utilisateur (l'appel correspondant au premier segment). *DSPAddress* n'est pas tenu de représenter une adresse valide. Elle peut être utilisée pour transmettre un code spécifique à l'application (un numéro FIFO par exemple).
- **Taille** : Utilisé uniquement pour les lectures de types de tableaux. Représente la taille (en nombre d'éléments du type de données demandé) du tableau à lire dans la mémoire du processeur. Pour les écritures, tout le contenu de *DataIn* est envoyé à la mémoire du processeur, quelle que soit la *taille*. Lorsque la *taille* est câblée, les données ne peuvent être transférées que sous forme de tableaux, et non de scalaires.
- **Symbole** : Chaîne de caractères du symbole auquel il faut accéder. Si *le symbole* est vide ou non connecté, *DSPAddress* et *MemSpace* sont utilisés.
- **Offset** : Représente le décalage des données à accéder par rapport à l'adresse de base indiquée par *Symbol* ou *DSPAddress*. L'adresse d'accès réelle est calculée comme la somme de l'adresse de base et du décalage. Le *décalage* est utile pour accéder à des membres individuels d'une structure ou d'un tableau. Si *DSPAddress* est utilisé pour transporter des données spécifiques à l'application, *Offset* ne doit pas être connecté.
- **R/~W** : Booléen indiquant le sens du transfert (true->read, false->write).
- **BranchLabel** : Chaîne de caractères correspondant à l'étiquette de la fonction définie par l'utilisateur. Dans le cas où *BranchLabel* est vide ou non connecté, *BranchAddress* est utilisé à la place.
- **BranchAddress** : Adresse physique de base du DSP pour la fonction définie par l'utilisateur.
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description de l'erreur précédente.

Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi*. Utilisez cette sortie pour propager le numéro de référence à d'autres Vis.
- **DataOut** : Données lues dans la mémoire du DSP.

- **Real DSPAddress** : Adresse réelle où le transfert a eu lieu. Cette adresse tient compte de la résolution du *symbole* (s'il est utilisé) et de l'effet du *décalage*.
- **ErrorCode** : Il s'agit du code d'erreur renvoyé par la fonction du noyau qui est exécutée. La valeur de cet indicateur n'est pas pertinente dans cette interface.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

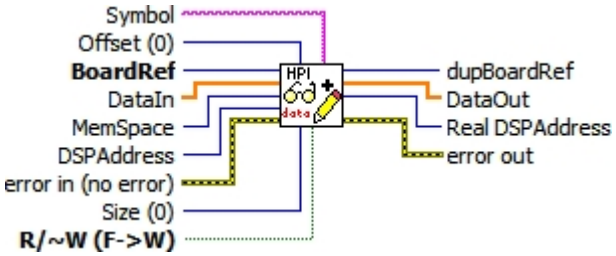
9.6.1.7 SR3_Base_HPI_Move_Offset

Ce VI est similaire à *SR3_Base_Bulk_Move_Offset*, sauf qu'il s'appuie sur le matériel de l'IPH, plutôt que sur le noyau, pour effectuer le transfert.

Les transferts sont limités aux emplacements de la RAM qui sont directement accessibles via le HPI. Le contrôle *MemSpace* n'est pas utilisé. Ce VI effectue des transferts dans et hors de l'espace de données et de l'espace programme. Ce VI effectuera des transferts vers n'importe quelle adresse accessible via le HPI, indépendamment de l'espace mémoire.

Note : Le noyau n'a pas besoin d'être chargé ou fonctionnel pour que ce VI s'exécute correctement. Ce VI terminera le transfert même si le DSP s'est planté, ce qui en fait un bon outil de débogage.

Les transferts avec l'IPH utilisent le control pipe 0 au lieu des bulk pipes rapides utilisés par SR3_Base_Bulk_Move_Offset. La bande passante pour ces transferts est généralement faible (500kb/s). Elle est cependant garantie.



Contrôles :

- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du
  Structure d'information de la carte globale. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **DataIn** : Mots de données à écrire dans la mémoire du DSP. *DataIn* doit être câblé, même pour une lecture, afin de spécifier le type de données à transférer.
- **MemSpace** : Espace mémoire pour l'échange (données, programme ou IO). *MemSpace* n'est pas utilisé dans ce VI. Il est fourni pour des raisons de compatibilité avec *le SR3_Bulk_Move_Offset.vi*. De cette façon, ces deux Vis sont interchangeables.
- **DSPAddress** : Adresse physique de base de la DSP pour l'échange. *DSPAddress* n'est utilisé que si *Le symbole* est vide ou n'a pas été câblé.
- **Size (taille)** : Utilisé uniquement pour les lectures de types de tableaux, représente la taille (en nombre d'éléments du type de données demandé) du tableau à lire dans la mémoire du processeur. Pour les écritures, tout le contenu de *DataIn* est écrit dans la mémoire du processeur, quelle que soit la *taille*. Lorsque la *taille* est câblée, les données ne peuvent être transférées que sous forme de tableaux, et non de scalaires.
- **Symbole** : Chaîne de caractères du symbole auquel il faut accéder. Si *le symbole* est vide ou non connecté,
  DSPAddress est utilisé.
- **Offset** : Représente le décalage des données à accéder par rapport à l'adresse de base indiquée par *Symbol* ou *DSPAddress*. L'adresse d'accès réelle est calculée comme la somme de l'adresse de base et du décalage. Le *décalage* est utile pour accéder à des membres individuels d'une structure ou d'un tableau.
- **R/~W** : Booléen indiquant le sens du transfert (true->read, false->write).

- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description du VI précédemment exécuté.

Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **DataOut** : Données lues dans la mémoire du DSP.
- **Real DSPAddress** : Adresse réelle où le transfert a eu lieu. Cette adresse tient compte de la résolution du *symbole* (s'il est utilisé) et de l'effet du *décalage*.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

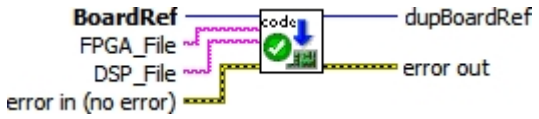
9.6.1.8 SR3_Base_LoadExec_User

Ce VI charge la logique FPGA d'un nouvel utilisateur et/ou le code DSP et exécute le code DSP à partir de l'adresse du point d'entrée trouvé dans son fichier COFF. Seuls les noms des fichiers doivent être indiqués. Les fichiers eux-mêmes doivent être stockés conformément aux règles décrites au point 8.5.4. Si *FPGA_File* ou *DSP_File* est vide, le VI ne charge pas le FPGA ou le DSP. Le noyau doit être chargé avant l'exécution de ce VI. Le DSP est réinitialisé avant de commencer le chargement. Après le chargement d'un nouveau code DSP, la table de symboles correspondante est mise à jour dans la *structure d'information globale de la carte*.

Le VI vérifie si le type de fichier COFF et/ou de fichier logique FPGA est adapté à la cible. Si ce n'est pas le cas, une erreur est générée.

Le VI réinitialise toujours le DSP, ce qui réinitialise également le FPGA. Si aucun nouveau fichier FPGA n'est spécifié, le FPGA est effacé après l'exécution du VI.

Après avoir effectué le branchement au point d'entrée, le contrôleur USB et le VI attendent un accusé de réception du DSP pour reprendre leur exécution. Si ce signal ne se produit pas, le VI se bloque. Normalement, le code DSP qui est lancé par ce VI doit acquitter le branchement en affirmant le signal *HINT* (voir la section sur le noyau de communication DSP).



Contrôles :

- **BoardRef** : Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi*
- **FPGA_File** : Il s'agit du nom du fichier FPGA. Si *FPGA_File* est connecté et n'est pas vide, le VI résout le chemin en utilisant les règles standard pour les emplacements du firmware et du VI conteneur. Si *FPGA_Files* est vide, le VI ne charge aucune logique FPGA. Le FPGA est réinitialisé par le VI de sorte que toute logique précédente est effacée. Depuis le FPGA.
- **DSP_File** : Nom du fichier DSP. Si *DSP_File* est connecté et n'est pas vide, le VI résout le chemin en utilisant les règles standard pour les emplacements de firmware et de conteneurs VI. Si *DSP_File* est vide, le VI ne charge aucun code DSP. La DSP est réinitialisée par le VI.
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description de l'erreur précédente.

Indicateurs :

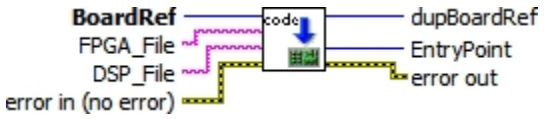
- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.1.9 SR3_Base_Load_User

Ce VI charge la logique FPGA et/ou le code DSP d'un nouvel utilisateur. Seuls les noms des fichiers doivent être indiqués. Les fichiers eux-mêmes doivent être stockés selon les règles décrites dans la section 8.5.4. Si *FPGA_File* ou *DSP_File* est vide, le VI ne charge pas le FPGA ou le DSP. Le noyau doit être chargé avant l'exécution de ce VI. Le DSP est réinitialisé avant de commencer le chargement. Après le chargement d'un nouveau code DSP, la table de symboles correspondante est mise à jour dans la *structure d'information globale de la carte*.

Le VI vérifie si le type de fichier COFF et/ou de fichier logique FPGA est adapté à la cible. Si ce n'est pas le cas, une erreur est générée.

Le VI réinitialise toujours le DSP, ce qui réinitialise également le FPGA. Si aucun nouveau fichier FPGA n'est spécifié, le FPGA est effacé après l'exécution du VI.



Contrôles :

- **BoardRef** : Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi*
- **FPGA_File** : Il s'agit du nom du fichier FPGA. Si *FPGA_File* est connecté et n'est pas vide, le VI résout le chemin en utilisant les règles standard pour les emplacements du firmware et du VI conteneur. Si *FPGA_Files* est vide, le VI ne charge aucune logique FPGA. Le FPGA est réinitialisé par le VI de sorte que toute logique précédente est effacée. Depuis le FPGA.
- **DSP_File** : Nom du fichier DSP. Si *DSP_File* est connecté et n'est pas vide, le VI résout le chemin en utilisant les règles standard pour les emplacements de firmware et de conteneurs VI. Si *DSP_File* est vide, le VI ne charge aucun code DSP. La DSP est réinitialisée par le VI.
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description de l'erreur précédente.

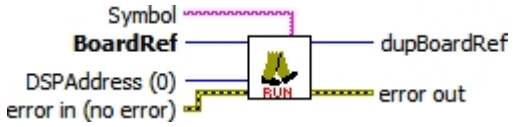
Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.1.10 SR3_Base_K_Exec

Ce VI force l'exécution du code DSP à se brancher sur une adresse spécifiée, passée en argument. Si *Symbol* est câblé et non vide, le VI recherche dans la table des symboles l'adresse correspondant à l'étiquette symbolique. Si le symbole n'est pas trouvé, une erreur est générée. Si *Symbol* n'est pas câblé ou est une chaîne vide, la valeur passée dans *DSPAddress* est utilisée comme point d'entrée.

Après avoir effectué le branchement au point d'entrée, le contrôleur USB et le VI attendent un accusé de réception du DSP pour reprendre leur exécution. Si ce signal ne se produit pas dans le délai spécifié, le VI se bloque indéfiniment. Normalement, le code DSP qui est lancé par ce VI doit acquitter le branchement en affirmant le signal *HINT* (voir la section sur le noyau de communication DSP).



Contrôles :

- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **DSPAddress** : Adresse physique de la branche. Elle est utilisée pour la branche si *le symbole* est vide ou non câblé.
- **Symbole** : Chaîne de caractères du symbole auquel il faut accéder. Si *Symbol* est vide, *DSPAddress* est utilisé à la place.
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description de l'erreur précédente.

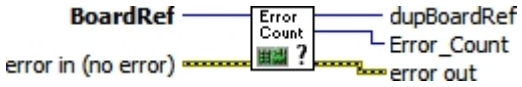
Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.1.11 SR3_Base_Read_Error_Count

Le matériel du contrôleur USB contient un compteur d'erreurs. Ce compteur circulaire de 4 bits est incrémenté chaque fois que le contrôleur détecte une erreur USB (à cause du bruit ou d'une autre raison).

Le contenu de ce compteur peut être lu périodiquement pour contrôler l'état de la connexion USB. Notez qu'une erreur USB n'entraîne généralement pas l'échec de la transaction. Le protocole USB réessaie les paquets contenant des erreurs jusqu'à trois fois au cours d'une même transaction avant de la faire échouer.



Contrôles :

- **BoardRef** : Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description de l'erreur précédente.

Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **Error_Count** : Il s'agit de la valeur contenue dans le compteur (entre 0 et 15).

- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.1.12 SR3_Base_Clear_Error_Count

Ce VI permet d'effacer le compteur d'erreurs USB de 4 bits.

Contrôles :

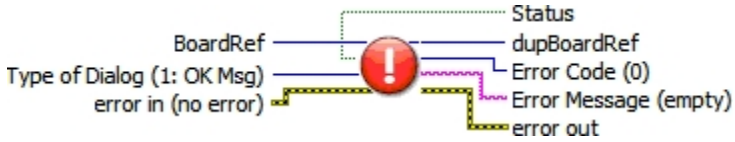
- **BoardRef** : Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description de l'erreur précédente.

Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.1.13 SR3_Base_Error Message

Ce VI peut être utilisé pour fournir une description textuelle d'une erreur générée par un VI de l'interface. Il est similaire à un VI traditionnel de message d'erreur LabVIEW, sauf qu'il contient tous les codes d'erreur qui peuvent être générés par l'interface, en plus des codes d'erreur LabVIEW normaux.



Contrôles :

- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **Type de dialogue** : Sélectionne l'un des trois comportements standard pour la gestion des erreurs.
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description du VI précédemment exécuté.

Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **Code d'erreur** : Indique le code d'erreur.
- **Message d'erreur** : Fournit une explication textuelle de l'erreur, le cas échéant.
- **Status** : Booléen, indique s'il y a une erreur (True) ou non (False).

- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.2 VI de support Flash

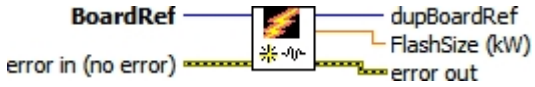
Ces VIs sont fournis pour supporter les opérations de programmation Flash. Les VIs supportent également les opérations de lecture et de programmation de la Flash pour des raisons de symétrie. Cependant, la lecture de la Flash ne nécessite pas la présence d'un code de support DSP et peut être effectuée par *SR3_Base_Bulk_Move_Offset*.

Note : Les VIs de cette bibliothèque requièrent un firmware spécial de support DSP pour être chargé en RAM et fonctionner. Ceci est normalement incompatible avec tout autre code DSP de l'utilisateur. Cette bibliothèque ne peut pas être utilisée pour lire-écrire la Flash dans le cadre d'un code DSP spécifique à une application. Dans ce cas, une solution personnalisée doit être conçue.

9.6.2.1 SR3_Flash_InitFlash

Ce VI télécharge et exécute le code DSP du support Flash. Le DSP est réinitialisé dans le cadre du processus de téléchargement. Tout le code DSP est abandonné, le FPGA est effacé. Le code de support Flash doit être exécuté en plus du noyau pour supporter les VIs de programmation Flash. Le VI détecte également la Flash, et s'il en trouve une, il renvoie sa taille en kWords. Si aucune mémoire flash n'est détectée, l'indicateur de taille est mis à 0.

Le code DSP de support Flash peut être soit un fichier COFF (.out), soit un firmware container VI. Le fichier doit être situé conformément aux règles de localisation des fichiers de microprogrammes décrites dans les sections précédentes.



Contrôles :

- **BoardRef** : Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description du VI précédemment exécuté.

Indicateurs :

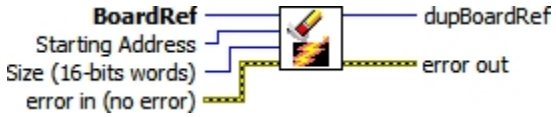
- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **FlashSize** : Cet indicateur renvoie la taille du flash détecté. Si aucun flash n'est détecté, il renvoie zéro.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.2.2 SR3_Flash_EraseFlash

Ce VI efface le nombre requis de mots de 16 bits de la Flash, à partir de l'adresse sélectionnée. L'effacement s'effectue par secteurs, ce qui signifie que le nombre de mots effacés peut être supérieur au nombre de mots sélectionnés. Par exemple, si l'adresse de départ n'est pas le premier mot d'un secteur, les mots du même secteur précédant l'adresse de départ seront effacés. De même, si le dernier mot sélectionné pour l'effacement n'est pas le dernier mot d'un secteur, des mots supplémentaires seront effacés jusqu'à la fin du dernier secteur sélectionné. L'effacement est tel que les mots sélectionnés, y compris l'adresse de départ, sont toujours effacés.

Remarque : Sur SignalRanger_mk2_Next_Generation, la taille du secteur est de 32 mots. Sur le SignalRanger_mk3, la taille du secteur est de 64 mots (128 kBytes).

Note : L'effacement ne doit être tenté que dans les sections de la carte mémoire qui contiennent de la Flash : L'effacement ne doit être tenté que dans les sections de la carte mémoire qui contiennent de la mémoire Flash. Les tentatives d'effacement en dehors de la Flash échoueront.



Contrôles :

- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **Adresse de départ** : Adresse du premier mot à effacer.
- **Taille** : Nombre de mots à effacer.
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description de l'erreur précédente.

Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.2.3 SR3_Flash_FlashMove

Ce VI lit ou écrit un nombre illimité de mots de données vers/depuis la mémoire Flash. Notez que si seules des lectures de mémoire Flash sont nécessaires, le VI *SR3_Base_Bulk_Move_Offset* doit être préféré, car il ne nécessite pas la présence du code de support Flash du DSP.

Le VI est polymorphe et permet les transferts des types suivants :

- Octets signés de 8 bits (I8), ou tableaux de ce type.
- Octets non signés de 8 bits (U8), ou tableaux de ce type.
- Mots de 16 bits signés (I16) ou tableaux de ce type.
- Mots de 16 bits non signés (U16) ou tableaux de ce type.
- Mots de 32 bits signés (I32) ou tableaux de ce type.
- Mots non signés de 32 bits (U32) ou tableaux de ce type.
- les nombres à virgule flottante de 32 bits (float) ou les tableaux de ce type.
- Cordes

Ils représentent tous les types de données de base utilisés par le compilateur C pour le DSP.

Pour transférer tout autre type (structures par exemple), la méthode la plus simple consiste à utiliser un "cast" pour permettre à ce type d'être représenté comme un tableau de U8 du côté du DSP (cast du type requis vers un tableau de U8 pour l'écrire dans le DSP, lecture d'un tableau de U8 et cast du type requis pour une lecture).

Toute tentative d'écriture en dehors de la mémoire Flash se soldera par un échec.

Le processus d'écriture peut transformer les uns en zéros, mais pas les zéros en uns. Si une opération d'écriture est tentée alors qu'elle devrait aboutir à la transformation d'un zéro en un, elle se solde par un

échec.

Normalement, un effacement doit être effectué avant l'écriture, afin que tous les bits de la zone d'écriture sélectionnée soient transformés en uns.

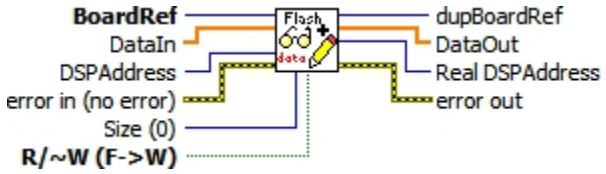
Note : Contrairement à la génération SignalRanger_mk2, la programmation incrémentale (programmation de la même adresse plusieurs fois) est autorisée. Chaque opération de programmation ne peut pas faire passer des bits individuels de 0 à 1, ce qui ne peut être fait que par un cycle d'effacement sur un secteur entier. Cependant, les opérations de programmation peuvent remettre des bits individuels de 1 à 0 à tout moment sans effacement intermédiaire.

En cas d'écriture d'un type de données plus étroit que le type natif de la plate-forme, des éléments supplémentaires sont ajoutés pour compléter l'écriture jusqu'à la frontière suivante du type natif. Les valeurs ajoutées prennent la valeur FF_H.

Le VI étant polymorphe, la lecture d'un type spécifique nécessite que ce type soit connecté à l'entrée *DataIn.* Ceci force simplement le type pour l'opération de lecture.

Remarque : Lors de la lecture ou de l'écriture de scalaires, la taille ne doit pas être câblée.

La représentation interne de la Flash est constituée de mots de 16 bits. Lors de la lecture ou de l'écriture de données de 8 bits, les octets représentent les parties haute et basse des registres de mémoire de 16 bits. Ils sont présentés MSB d'abord et LSB ensuite.



Contrôles :

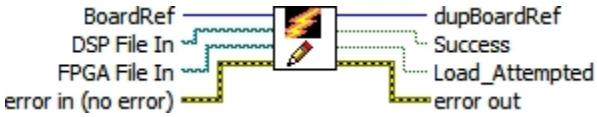
- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **DataIn** : Mots de données à écrire dans la mémoire Flash. *DataIn* doit être câblé, même pour une lecture, afin de spécifier le type de données à transférer.
- **DSPAddress** : Adresse physique de base de la DSP pour le transfert.
- **Size (taille)** : Utilisé uniquement pour les lectures de types de tableaux, représente la taille (en nombre d'éléments du type de données demandé) du tableau à lire dans la mémoire du processeur. Pour les écritures, tout le contenu de *DataIn* est écrit dans la mémoire du processeur, quelle que soit la *taille*. Lorsque la *taille* est câblée, les données ne peuvent être transférées que sous forme de tableaux, et non de scalaires.
- **R/~W** : Booléen indiquant le sens du transfert (true->read, false->write).
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description de l'erreur précédente.

Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **DataOut** : Données lues à partir de la mémoire Flash.
- **Real DSPAddress** : Adresse réelle où le transfert a eu lieu. Cette adresse tient compte de l'effet du *décalage*.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.2.4 SR3_Flash_Config_NoDialog

Ce VI programme automatiquement un fichier DSP et/ou un fichier FPGA dans Flash. Il ne nécessite pas d'interaction de la part de l'utilisateur, mais présente son panneau avant à l'utilisateur pendant la programmation afin que l'opération puisse être surveillée. Le VI présente des dialogues d'erreur et/ou d'achèvement.



Contrôles :

- **BoardRef** : Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **Fichier DSP In** : Chemin d'accès du fichier DSP à programmer en Flash. Aucun fichier n'est programmé si le chemin est vide. Nouveauté dans *SignalRanger_mk3*, le *chemin d'accès au fichier* peut pointer vers un VI contenant le micrologiciel, ainsi que vers un fichier COFF (.out).
- **FPGA File In** : Chemin d'accès du fichier FPGA à programmer dans la Flash. Aucun fichier n'est programmé si le chemin est vide. Nouveauté dans *SignalRanger_mk3*, le *chemin d'accès* peut pointer vers un VI contenant le micrologiciel, ainsi que vers un fichier FPGA (.rbt).
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description du VI précédemment exécuté.

Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **Succès** : Renvoie *la valeur true* si le chargement a réussi. Dans le cas contraire, renvoie la valeur *false*.
- **Load_Attempted** : Retourne toujours *à true*. Ce booléen est fourni pour des raisons de compatibilité.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.2.5 SR3_Flash_Config_Dialog

Ce VI programme un fichier DSP et/ou un fichier FPGA dans la Flash. Contrairement à *SR3_Flash_Config_NoDialog*, le VI est interactif. Il demande à l'utilisateur les fichiers d'entrée et l'action (*écrire* ou *annuler*). Le VI présente des dialogues d'erreur et/ou d'achèvement. Nouveau dans *SignalRanger_mk3*, les fichiers DSP et FPGA peuvent être contenus dans des VIs de conteneurs de firmware, ainsi que dans des fichiers .out ou .rbt.

Contrôles :

- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du
  Structure d'information de la carte globale. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description du VI précédemment exécuté.

Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.

- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.2.6 SR3_Flash_Check_Dialog

Ce VI vérifie le Flash par rapport à un fichier DSP et/ou un fichier FPGA. Le VI est interactif. Il demande à l'utilisateur les fichiers d'entrée et l'action (*Vérifier* ou *Annuler*). Le VI présente des dialogues d'erreur et/ou d'achèvement. Nouveau dans *SignalRanger_mk3*, les fichiers DSP et FPGA peuvent être contenus dans des VIs conteneurs de firmware, ainsi que dans des fichiers .out ou .rbt.

Contrôles :

- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description du VI précédemment exécuté.

Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

9.6.3 Support FPGA VIs

Ces VIs sont fournis pour soutenir les opérations de configuration du FPGA.

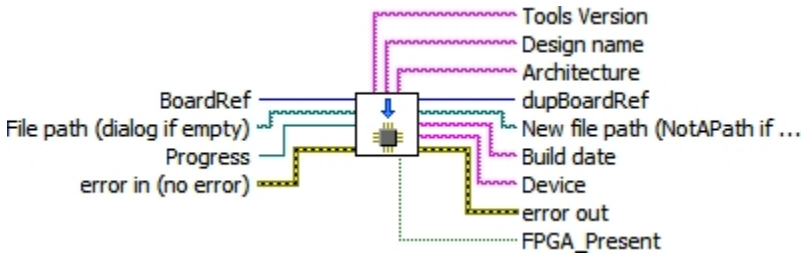
Sur *SR2_NG*, les VI de cette bibliothèque nécessitent le chargement en RAM d'un micrologiciel spécial de support du DSP et la réinitialisation du DSP. Sur *SR3*, ce n'est pas le cas.

Sur *SR2_NG*, la réinitialisation du DSP et le chargement d'un micrologiciel spécial de support du FPGA sont normalement incompatibles avec l'exécution de tout autre code DSP utilisateur. Ces VIs ne peuvent pas être utilisés pour reconfigurer le FPGA dans le cadre d'un code DSP spécifique à une application sans interférer avec un code DSP déjà en cours d'exécution.

Pour recharger les deux DSP et et le code FPGAlogique utiliser l'utilisateur *SR3_Base_LoadExec_User* ou *SR3_Base_Load_User* VIs.

9.6.3.1 SR3_FPGA_LoadConfiguration_All_Platforms

Ce Vi télécharge un fichier de configuration logique .rbt dans le FPGA. Sur *SR2_NG*, le DSP est réinitialisé avant le téléchargement. Tout le code DSP est interrompu. Le fichier .rbt doit être valide, et doit être correct pour le FPGA spécifié. Le chargement d'un fichier rbt non valide dans un FPGA peut endommager la pièce. Le fichier FPGA peut être contenu dans un VI contenant le micrologiciel, ainsi que dans un fichier .rbt réel.



Contrôles :

- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par *SR3_Base_Open_Next_Avail_Board.vi*
- **Chemin d'accès au fichier** : Il s'agit du chemin d'accès au fichier ".rbt" décrivant la logique FPGA. Une boîte de dialogue est présentée si le chemin est vide. Nouveau dans *SignalRanger_mk3*, le fichier FPGA peut être contenu dans un VI conteneur de firmware, ainsi que dans un fichier .rbt réel.
- **Progression** : Il s'agit d'un numéro de référence sur la barre de progression qui est mise à jour par le VI. De cette façon, une barre de progression peut être affichée sur le panneau avant du VI appelant.
- **Erreur dans** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro d'erreur et la description de l'erreur précédente.

Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board.vi* Utiliser cette sortie pour propager le numéro de référence à d'autres Vis.
- **Nouveau chemin d'accès au fichier** : Il s'agit du chemin d'accès au fichier .rbt ou au conteneur VI décrivant la logique FPGA.
- **Version des outils** : Chaîne ASCII indiquant la version des outils qui ont été utilisés pour générer l'information. fichier .rbt.
- **Nom de la conception** : Chaîne ASCII indiquant le nom de la conception logique.
- **Architecture** : Chaîne ASCII indiquant le type d'appareil visé par le fichier .rbt
- **Appareil** : Chaîne ASCII indiquant le numéro de l'appareil visé par le fichier .rbt
- **Build Date (Date de construction**) :Chaîne ASCII indiquant la date de création du fichier .rbt
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabVIEW. Contient le numéro et la description de l'erreur.

10 Interface USB C/C++

L'interface C/C++ est fournie sous la forme d'une DLL nommée *SRm3_HL.dll*. Cette interface a été conçue comme une image miroir de l'interface LabVIEW. La documentation de l'interface LabVIEW s'applique dans une large mesure à l'interface C/C++.

Cette interface a été conçue pour le développement en C/C++ et n'a été testée que sur la version 2005 de Visual Studio de Microsoft. Cependant, il est possible de l'utiliser avec d'autres environnements de développement permettant l'utilisation de DLL.

Pour fonctionner à l'exécution, cette DLL exige que le fichier suivant se trouve dans le même répertoire que l'application utilisateur qui l'utilise :

- *SRm3_HL.dll* La DLL de l'interface principale

En outre, le moteur d'exécution LabView 2009 doit être installé sur l'ordinateur qui doit utiliser la DLL. Si l'utilisateur souhaite déployer une application à l'aide de l'interface C/C++, qui est

Si le moteur d'exécution LabView 2009 doit fonctionner sur des ordinateurs autres que ceux sur lesquels il a été développé, il doit être installé séparément sur ces ordinateurs. Un programme d'installation du moteur d'exécution est disponible gratuitement sur le site web de National Instruments www.ni.com.

Deux exemples sont fournis, qui couvrent le développement de code dans Visual Studio :

- Le premier exemple se trouve dans : *C:\ProgramFiles\SR3_Applications\Visual_Studio_Code_Example\SRm3_HL_DLL_VS_Example. zip*. Il couvre l'utilisation de l'interface USB sur la plate-forme *SR3*.
- Le deuxième exemple se trouve dans : *C:\ProgramFiles\SR3_Applications\Visual_Studio_Code_Example\SRm3PRO_HL_DLL_VS_Exa mple.zip*. Il couvre l'utilisation de l'interface Ethernet sur la plate-forme *SR3_Pro.*

10.1 Temps d'exécution et gestion des threads

Deux fonctions de la DLL *SRm3_HL* accédant à la même carte DSP *SignalRanger_mk3* ne peuvent pas s'exécuter simultanément. La première fonction doit se terminer avant que la seconde puisse être appelée. Dans les environnements multithread, il faut veiller à ce que des fonctions distinctes de la DLL ne s'exécutent pas en même temps (dans des threads distincts). La méthode la plus simple consiste à s'assurer que tous les appels aux fonctions de la DLL sont effectués dans le même thread. Toutefois, les fonctions de l'interface accédant à différentes cartes peuvent être appelées simultanément.

Toutes les fonctions de la DLL *SRm3_HL* sont bloquantes. Elles ne reviennent pas tant que l'action demandée n'a pas été effectuée sur la carte.

10.2 Conventions d'appel

Les fonctions sont appelées en utilisant les conventions d'appel du langage C.

Toutes les fonctions renvoient un *USB_Error_Code* sous la forme d'un entier signé de 32 bits. Ce code d'erreur est égal à zéro si aucune erreur ne s'est produite.

Lorsqu'une fonction doit renvoyer un tableau ou une chaîne de caractères, l'espace correspondant (de taille suffisante) doit être alloué par l'appelant et un pointeur sur cet espace est transmis à la fonction. En outre, la taille de l'élément alloué par l'appelant est transmise à la fonction. L'argument de *taille* associé au tableau ou à la chaîne de caractères suit normalement le tableau ou la chaîne de caractères dans la ligne d'argument.

10.3 Construire un projet à l'aide de Visual Studio

Pour créer un projet à l'aide de Visual Studio, il convient de suivre les lignes directrices suivantes. Un exemple est fourni pour accélérer la courbe d'apprentissage (voir la dernière section du chapitre en cours).

- Si le projet est lié statiquement à la bibliothèque *SRm3_HL.lib*, il doit être chargé à l'aide de la fonction DELAYLOAD de Visual C++. Pour utiliser DELAYLOAD, ajoutez *delayimp.lib* au projet (dans Visual Studio 2005, il se trouve dans *Program Files\Microsoft Visual Studio 8\VC\lib*) ; dans *Project Properties*, sous *Linker\Command Line\Additional Options*, ajoutez la commande */DELAYLOAD:SRm3_HL.dll*.
- La DLL peut également être chargée dynamiquement à l'aide de *LoadLibrary* et les fonctions de la DLL doivent être appelées à l'aide de *GetProcAddress*. Ne pas établir de lien statique avec la bibliothèque *SRm3_HL.lib* sans utiliser la fonction DELAYLOAD.
- Ajouter *#include "SRm3_HL.h"* dans le fichier principal.
- Si vous utilisez la fonction DELAYLOAD pour établir un lien statique avec la bibliothèque *SRm3_HL.lib*, ajoutez *SRm3_HL.lib* au projet.
- Les fichiers suivants doivent être placés dans le dossier contenant les sources du projet :
    - cvilvsb.h
    - extcode.h
    - fundtypes.h

- hosttype.h
- ILVDataInterface.h
- ILVTypeInterface.h
- platdefines.h
- SRm3_HL.h

Tous ces fichiers font partie de l'exemple fourni.

10.4 Fonctions d'interface exportées

10.4.1 SR3_DLL_Open_Next_Avail_Board

10.4.1.1 Prototype

int32_t **SR3_DLL_Open_Next_Avail_Board**(uint16_t idVendor_Restrict, uint16_t idProduct_Restrict, uint16_t ForceReset, int32_t *BoardRef, char DSP_Firmware_Name[], int32_t DSP_Firmware_Name_Size, char FPGA_Logic_Name[], int32_t FPGA_Logic_Name_Size)

10.4.1.2 Description

Cette fonction permet d'effectuer les opérations suivantes :

- Essaie de trouver une carte DSP libre avec le VID/PID sélectionné, et éventuellement avec le firmware indiqué dans le *Restrict* control.
- S'il en trouve un, il crée une entrée dans la *structure d'information globale du conseil d'administration*.
- Attend que le noyau de mise sous tension soit chargé sur la carte.
- Si un microprogramme DSP est détecté dans la Flash (le code a été chargé et démarré dans le cadre de la séquence de mise sous tension), charge le nom de fichier correspondant et la table de symboles à partir de la Flash.
- Si *ForceReset* est vrai, il force la réinitialisation du DSP, puis recharge le noyau Host-Download. Dans ce cas, tout le code présent dans la Flash et exécuté à la mise sous tension est annulé et la table de symboles correspondante trouvée dans la Flash n'est pas chargée.
- Place la table des symboles du noyau actuel dans la *structure d'information globale de la carte*.

10.4.1.3 Entrées

- **idVendor_Restrict** Doit correspondre à l'identifiant du fournisseur pour la carte spécifiée. Par exemple, l'ID
  Le VID pour *Soft-dB* est 0x1612.
- **idProduct_Restrict** Doit correspondre à l'identifiant du produit pour la carte spécifiée. Par exemple, *le PID* de *SignalRanger_mk3* est 0x102.
- **ForceReset** Si cette valeur est fixée à 1, le DSP est réinitialisé et le *noyau de téléchargement de l'hôte* est chargé. Tout le code DSP en cours d'exécution est interrompu. Utilisez ce paramètre pour recharger dynamiquement le nouveau code DSP sur la carte. Remettre à zéro pour prendre le contrôle du code DSP déjà en cours d'exécution à partir de la Flash sans l'interrompre.
- **DSP_Firmware_Name_Size** Une chaîne de caractères suffisamment longue doit être allouée pour que la fonction renvoie le nom du fichier DSP présent dans Flash. *DSP_Firmware_Name_Size* doit être réglé sur la taille réelle allouée à la chaîne.
- **FPGA_Logic_Name_Size** Une chaîne de caractères suffisamment longue doit être allouée pour que la fonction renvoie le nom du fichier FPGA présent dans la mémoire Flash. *FPGA_Logic_Name_Size* doit être réglé sur la taille réelle allouée à la chaîne.

10.4.1.4 Sorties

- **BoardRef** Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la *structure globale d'information sur les cartes*. L'interface peut gérer une multitude de cartes connectées au même PC. Un numéro *BoardRef* correspondant est attribué à chacune d'entre elles lorsqu'elles sont ouvertes. Toutes les autres fonctions de l'interface utilisent ce numéro pour accéder à la carte appropriée.

- **DSP_Firmware_Name[]** Cette chaîne contient le nom du fichier de micrologiciel DSP présent dans la Flash. La chaîne est vide si la Flash ne contient aucun micrologiciel. La chaîne est également vide si le contrôle *ForceReset* est vrai.
- **FPGA_Logic_Name[]** Cette chaîne contient le nom du fichier logique FPGA présent dans la Flash. La chaîne est vide si la Flash ne contient aucune logique FPGA. La chaîne est également vide si la commande *ForceReset* est vraie.

Note : La poignée que l'interface fournit pour accéder à la carte est exclusive. Cela signifie qu'une seule application à la fois peut ouvrir et gérer un tableau. Il en résulte qu'une carte ne peut pas être ouverte deux fois. Une carte qui a déjà été ouverte à l'aide de la fonction SR3_Base_Open_Next_Avail_Board ne peut pas être ouverte à nouveau tant qu'elle n'a pas été correctement fermée à l'aide de la fonction SR3_Base_Close_BoardNb. Ceci est particulièrement préoccupant lorsque l'application gérant la carte est fermée dans des conditions anormales. Si l'application est fermée sans fermer correctement la carte. L'exécution suivante de l'application peut échouer à trouver et à ouvrir la carte, simplement parce que l'instance de pilote correspondante est encore ouverte. Dans ce cas, il suffit de déconnecter et de reconnecter la carte pour forcer le PC à ré-énumérer la carte.

10.4.2 SR3_DLL_Close_BoardNb

10.4.2.1 Prototype

int32_t **SR3_DLL_Close_BoardNb**(int32_t BoardRef)

10.4.2.2 Description

Cette fonction ferme l'instance du pilote utilisé pour accéder à la carte et supprime l'entrée correspondante dans la *structure d'information globale de la carte*. Utilisez-la après le dernier accès à la carte, pour libérer les ressources qui ne sont plus utilisées.

10.4.2.3 Entrées

- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure globale d'information sur les cartes*. Elle est créée par *SR3_Base_Open_Next_Avail_Board()*

10.4.2.4 Sorties

Aucun

10.4.3 SR3_DLL_Complete_DSP_Reset

10.4.3.1 Prototype

int32_t **SR3_DLL_Complete_DSP_Reset**(int32_t BoardRef)

10.4.3.2 Description

Cette fonction permet d'effectuer les opérations suivantes :

- La LED clignote temporairement en orange
- Réinitialise le DSP
- Réinitialise le PCSI
- Charge le noyau Host-Download

Ces opérations sont nécessaires pour prendre complètement le contrôle d'un DSP qui exécute un autre code ou qui s'est planté. L'opération complète prend 500 ms.

10.4.3.3 Entrées

- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure globale d'information sur la carte*. Elle est créée par *SR3_Base_Open_Next_Avail_Board()*

10.4.4 SR3_DLL_WriteLeds

10.4.4.1 Prototype

int32_t **SR3_DLL_WriteLeds**(int32_t BoardRef, uint16_t LedState)

10.4.4.2 Description

Cette Vi permet l'activation sélective de chaque élément de la Led bicolore.

- Off 0
- Rouge 1
- Vert 2
- Orange 3

10.4.4.3 Entrées

- **BoardRef** Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans le fichier *Structure globale d'information sur la carte*. Elle est créée par *SR3_Base_Open_Next_Avail_Board()*
- **LedState** La valeur définit la couleur : (0-Off, 1-Rouge, 2-Vert, 3-Orange)

10.4.4.4 Sorties

Aucun

10.4.5 SR3_DLL_Bulk_Move_Offset_U8

10.4.5.1 Prototype

int32_t **SR3_DLL_Bulk_Move_Offset_U8**(int32_t BoardRef, uint16_t ReadWrite, char Symbol[], uint32_t DSPAddress, uint16_t MemSpace, uint32_t Offset, uint8_t Data[], int32_t Size, uint16_t Atomic)

10.4.5.2 Description

Cette fonction lit ou écrit un nombre illimité d'octets vers/depuis l'espace de programme, de données ou d'E/S du DSP, en utilisant le noyau. Ce transfert utilise des tuyaux en vrac. Cela se traduit par une bande passante élevée.

Cette fonction ne transfère que des tableaux d'octets. Pour transférer tout autre type, la méthode la plus simple consiste à utiliser un "cast" pour permettre à ce type d'être représenté comme un tableau de U8 du côté du DSP (cast du type requis vers un tableau de U8 pour l'écrire au DSP, lecture d'un tableau de U8 et cast vers le type requis pour une lecture).

L'adresse du DSP et l'espace mémoire du transfert sont spécifiés comme suit :

- Si *Symbole* n'est pas vide et que le symbole est représenté dans la table des symboles, le transfert a lieu à l'adresse et dans l'espace mémoire correspondant à *Symbole*. Notez que *Symbole* doit représenter une adresse valide. En outre, le fichier DSP COFF doit être lié avec la convention habituelle de numéro de page :
  - Espace programme = numéro de page 0
  - Espace de données = page numéro 1
  - Espace OI = page numéro 2
  - Tous les autres numéros de page sont accessibles en tant qu'espace de données.
- Si *Symbol* est vide, *DSPAddress* est utilisé comme adresse d'octet pour le transfert et *MemSpace* est utilisé comme espace mémoire.
- Notez que *DSPAddress* peut devoir être aligné sur la largeur appropriée, en fonction de la plate-forme spécifique.
- La valeur du *décalage* est ajoutée à *DSPAddress*. Cette fonctionnalité est utile pour accéder à des membres individuels de structures ou de tableaux sur le DSP. Notez que la valeur de *Offset* est toujours comptée en octets.

10.4.5.2.1 Notes sur l'atomicité de transfert

Lors de la lecture ou de l'écriture de types plus grands que le type natif de la plate-forme, le PC effectue plusieurs accès séparés pour chaque type long transféré. En principe, il est possible que le DSP ou le PC accède à un mot au milieu de l'échange, corrompant ainsi les données. Par exemple, lors d'une lecture sur une plate-forme DSP où le type natif est U16, l'hôte pourrait télécharger une valeur en virgule flottante juste après que le DSP ait mis à jour un mot de 16 bits constituant la virgule flottante, mais avant qu'il n'ait mis à jour l'autre. Il est évident que la valeur lue par l'hôte serait complètement erronée. Symétriquement, lors d'une écriture, l'hôte pourrait modifier les deux mots de 16 bits constituant un flottant dans la mémoire du DSP, juste après que le DSP ait lu le premier, mais avant qu'il n'ait lu le second. Dans cette situation, le processeur travaille avec une "ancienne" version d'une moitié du flottant et une nouvelle version de l'autre moitié.
Ces problèmes peuvent être évités si l'on comprend les faits suivants :

Sur la plate-forme *SignalRanger_mk2_Next_Generation*, lorsque le PC accède à un groupe de valeurs, il le fait toujours par blocs de 32 mots de 16 bits à la fois (jusqu'à 256 mots si la carte est connectée à un concentrateur USB à haute vitesse ou à une racine). Chacun de ces accès aux blocs est atomique. Le DSP est ininterrompu et ne peut effectuer aucune opération au milieu d'un bloc du transfert PC. Par conséquent, le DSP ne peut pas "interférer" au milieu d'un accès unique de 32 ou 256 blocs par le PC. Cela ne suffit pas à garantir l'intégrité des valeurs transférées, car le PC peut toujours transférer un bloc complet de données au milieu d'une autre opération concurrente du DSP sur ces mêmes données. Pour éviter cette situation, il suffit de rendre atomique toute opération DSP sur des données de 32 bits qui pourraient être modifiées par le PC. Les accès au PC sont alors atomiques des deux côtés, et les données peuvent être transférées en toute sécurité 32 bits à la fois. Sur cette plate-forme, les transferts sont toujours atomiques du côté du PC. Le contrôle *atomique* est présent pour des raisons de compatibilité mais n'a aucun effet.

Sur la plate-forme *SignalRanger_mk3*, l'utilisateur a le choix entre les transferts atomiques et les transferts non atomiques. L'utilisation d'un transfert atomique présente l'avantage que, du point de vue du DSP, toutes les parties du bloc sont transférées simultanément. Ce comportement est compatible avec celui de la plate-forme *SignalRanger_mk2_Next_Generation*. Les transferts non atomiques présentent l'avantage que les tâches critiques du DSP peuvent interrompre le transfert et donc avoir la priorité sur le transfert USB. La seule façon d'utiliser un transfert non atomique sur *SignalRanger_mk2_Next_Generation* est d'utiliser une fonction utilisateur personnalisée et la fonction *SR3_Base_User_Move_Offset()*.

10.4.5.3 Entrées

- **BoardRef** Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans le fichier *Structure globale d'information sur la carte*. Elle est créée par *SR3_Base_Open_Next_Avail_Board()*
- **Lecture-écriture** : 1->Lecture, 0->Écriture.
- **Symbole** : Chaîne de caractères du symbole auquel il faut accéder. Si *Symbol* est vide, *DSPAddress* et *MemSpace* sont utilisés.
- **DSPAddress** : Adresse physique de base de la DSP pour l'échange. *DSPAddress* n'est utilisé que si Le *symbole* est vide.
- **MemSpace** : Espace mémoire pour l'échange (données, programme ou IO). *MemSpace* n'est utilisé que si *Symbol* est vide.
- **Offset** : Représente le décalage des données à accéder par rapport à l'adresse de base indiquée par *Symbol* ou *DSPAddress*. L'adresse d'accès réelle est calculée comme la somme de l'adresse de base et du décalage. L'*offset* est utile pour accéder à des membres individuels d'une structure ou d'un tableau.
- **Données** : Tableau d'octets à écrire ou à lire dans la mémoire du DSP.
- **Taille** : Représente le nombre d'octets à transférer. Pour une lecture ou une écriture, le paramètre *Data*
  alloué doit être supérieur ou égal à la *taille*.
- **Atomique** : 1-> transfert atomique, 0-> transfert non atomique.

10.4.5.4 Sorties

- **Données** : Tableau d'octets écrits ou lus dans la mémoire du DSP. Le tableau *Data* transmis en argument à la fonction doit être supérieur ou égal à *Size*.

10.4.6 SR3_DLL_User_Move_Offset_U8

10.4.6.1 Prototype

int32_t **SR3_DLL_User_Move_Offset_U8**(int32_t BoardRef, uint16_t ReadWrite, char Symbol[], uint32_t DSPAddress, uint32_t Offset, char BranchLabel[], uint32_t BranchAddress, uint8_t Data[], int32_t Size)

10.4.6.2 Description

Cette fonction est similaire à *SR3_Base_Bulk_Move_Offset()*, sauf qu'elle permet à une fonction DSP définie par l'utilisateur de remplacer la fonction noyau intrinsèque utilisée par *SR3_Base_Bulk_Move_Offset().*

Cette fonction ne transfère que des tableaux d'octets. Pour transférer tout autre type, la méthode la plus simple consiste à utiliser un "cast" pour permettre à ce type d'être représenté comme un tableau de U8 du côté du DSP (cast du type requis vers un tableau de U8 pour l'écrire au DSP, lecture d'un tableau de U8 et cast vers le type requis pour une lecture).

Le fonctionnement du contrôleur USB et du noyau permet à une fonction DSP définie par l'utilisateur de remplacer les fonctions intrinsèques du noyau (voir la documentation du noyau ci-dessous). Pour cela, la fonction DSP définie par l'utilisateur doit effectuer les mêmes actions avec la boîte aux lettres que la fonction intrinsèque du noyau (lecture ou écriture du noyau). Cela peut être utile pour définir de nouvelles fonctions de transfert avec une fonctionnalité spécifique à l'application. Par exemple, une fonction de lecture ou d'écriture d'une FIFO pourrait être définie de cette manière. Outre la fonctionnalité de transfert de données, une fonction de lecture ou d'écriture FIFO inclurait également la gestion nécessaire des pointeurs, qui n'est pas présente dans les fonctions intrinsèques du noyau.

Par conséquent, *SR3_Base_User_Move_Offset()* comprend deux arguments d'entrée pour définir le point d'entrée de la fonction qui doit être utilisée pour remplacer la fonction intrinsèque du noyau.

Un transfert d'un nombre de mots supérieur à 32 (supérieur à 256 pour une connexion USB à haut débit) est segmenté en autant de transferts de 32 mots (256 mots) que nécessaire. La fonction définie par l'utilisateur est appelée à chaque nouveau segment. Si le nombre total de mots à transférer n'est pas un multiple de 32 (256), le dernier segment contient le reste.

La fonction définie par l'utilisateur doit être créée pour fonctionner de la même manière que les fonctions de lecture et d'écriture du noyau. En d'autres termes, elle doit effectuer les mêmes transferts, l'entretien de la boîte aux lettres et les échanges que les fonctions du noyau :

10.4.6.2.1 SR2_NG Plate-forme

- S'il s'agit d'un transfert, la fonction transfère les mots nécessaires vers ou depuis la zone de *données* de la boîte aux lettres. Le nombre de mots à transférer est la valeur du champ *NbWords* de la boîte aux lettres. Dans SR2_NG, ce champ représente le nombre de mots à transférer dans ce seul segment, et non le nombre total de mots à transférer pour l'ensemble de la requête USB. Ceci est différent du cas SR3.
- Le champ *TransferAddress* peut avoir besoin d'être incrémenté, selon la façon dont la fonction DSP utilise cette information. En général, les fonctions du noyau *K_Read* et *K_Write* incrémentent *TransferAddress* de manière à ce que le segment suivant soit transféré vers/depuis les adresses mémoire suivantes. Cependant, certaines fonctions utilisateur peuvent utiliser ce champ d'une manière différente. Par exemple, il peut être utilisé comme un numéro de tampon FIFO arbitraire dans certaines implémentations. Pour *K_Read* et *K_Write, TransferAddress* représente un nombre d'octets.
- Après l'exécution de la fonction demandée, le DSP active le signal HINT pour signaler au contrôleur USB que l'opération est terminée. Cette opération a été définie de manière pratique dans une macro *Acquittement* dans les codes d'exemple, et peut être insérée à la fin de n'importe quelle fonction utilisateur.

Notez que du point de vue du PC, la commande semble "suspendue" jusqu'à ce que l'accusé de réception soit émis par le DSP. Le code utilisateur ne doit pas prendre trop de temps avant d'émettre l'accusé de réception.

10.4.6.2.2 Plate-forme SR3

- S'il s'agit d'un transfert, la fonction doit lire le *code de contrôle* (adresse de boîte aux lettres 0x10F0400A), masquer les deux bits inférieurs qui représentent le type d'opération. Le résultat, appelé *USBTransferSize*, représente le nombre maximum d'octets qui doivent être transférés dans ce segment. Notez que le contenu du champ *ControlCode* de la boîte aux lettres ne doit pas être modifié.
- Si nécessaire, la fonction transfère les octets requis vers ou depuis la zone de *données* de la boîte aux lettres. Le nombre d'octets à transférer est le plus petit entre le champ *NbBytes* de la boîte aux lettres et le résultat *USBTransferSize* qui vient d'être calculé.
- Enfin, le nombre d'octets transférés doit être soustrait de la valeur *NbBytes*, et la valeur
  Le champ *NbBytes* doit être mis à jour avec la nouvelle valeur dans la boîte aux lettres.
- Le champ *TransferAddress* peut avoir besoin d'être incrémenté, selon la manière dont la fonction utilise cette information. En règle générale, les fonctions du noyau *K_Read* et *K_Write* incrémentent *TransferAddress* de manière à ce que le segment suivant soit transféré vers/depuis l'adresse mémoire suivante. Cependant, certaines fonctions utilisateur peuvent utiliser ce champ d'une manière différente. Par exemple, il peut être utilisé comme un numéro de tampon FIFO arbitraire dans certaines implémentations. Pour *K_Read* et *K_Write, TransferAddress* représente un nombre d'octets.
- Après l'exécution de la fonction demandée, le DSP active le signal HINT pour signaler au contrôleur USB que l'opération est terminée. Cette opération a été définie de manière pratique dans une macro *Acquittement* dans les codes d'exemple, et peut être insérée à la fin de n'importe quelle fonction utilisateur. Notez que du point de vue du PC, la commande semble "suspendue" jusqu'à ce que l'accusé de réception soit émis par le DSP. Le code utilisateur ne doit pas prendre trop de temps avant d'émettre l'accusé de réception.

Pour plus d'informations, voir la section sur le fonctionnement du noyau.

Remarque : Sur SignalRanger_mk2_Next_Generation, si TransferAddress est utilisé pour transporter des informations autres qu'une adresse de transfert réelle, les restrictions suivantes s'appliquent :

- La taille totale du transfert doit être inférieure ou égale à 32768 mots. En effet, les transferts sont segmentés en transferts de 32768 mots à un niveau supérieur. Les informations contenues dans TransferAddress ne sont conservées que pendant le premier de ces segments de niveau supérieur. Au suivant, TransferAddress est mis à jour comme s'il s'agissait d'une adresse pointant vers le bloc suivant.
- Le transfert ne doit pas franchir une limite de 64 kWord. Les transferts qui franchissent une limite de 64 kWord sont divisés en deux transferts consécutifs. Les informations contenues dans TransferAddress ne sont conservées que pendant le premier de ces segments de niveau supérieur. Au suivant, TransferAddress est mis à jour comme s'il s'agissait d'une adresse pointant vers le bloc suivant
- TransferAddress doit être paire. Elle est considérée comme une adresse de transfert d'octets, c'est pourquoi son bit 0 est masqué à haut niveau.

10.4.6.3 Entrées

- **BoardRef** Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans le fichier *Structure globale d'information sur la carte*. Elle est créée par *SR3_Base_Open_Next_Avail_Board()*
- **Lecture-écriture** : 1->Lecture, 0->Écriture.
- **Symbole** : Chaîne de caractères du symbole auquel il faut accéder. Si *Symbol* est vide, *DSPAddress* et *MemSpace* sont utilisés.
- **DSPAddress** : Adresse physique de base du DSP pour l'échange. *DSPAddress* n'est utilisé que si Le *symbole* est vide.
- **Offset** : Représente le décalage des données à accéder par rapport à l'adresse de base indiquée par *Symbol* ou *DSPAddress*. L'adresse d'accès réelle est calculée comme la somme de l'adresse de base et du décalage. L'*offset* est utile pour accéder à des membres individuels d'une structure ou d'un tableau.

- **BranchLabel** : Chaîne de caractères représentant le nom de la fonction DSP qui doit être appelée dans le cadre de l'opération. Si *BranchLabel* est vide, *BranchAddress* est utilisé.
- **BranchAddress** : Point d'entrée de la fonction DSP qui doit être appelée dans le cadre de l'opération. *BranchAddress* n'est utilisé que si *BranchLabel* est vide.
- **Données** : Tableau d'octets à écrire dans la mémoire du DSP.
- **Taille** : Représente le nombre d'octets à transférer. Pour une lecture ou une écriture, le paramètre *Data* alloué doit être supérieur ou égal à la *taille*.

10.4.6.4 Sorties

- **Données** : Tableau d'octets lus à partir de la mémoire du DSP. Le tableau *Data* transmis en argument à la fonction doit être supérieur ou égal à *Size*.

10.4.7 SR3_DLL_HPI_Move_Offset_U8

10.4.7.1 Prototype

int32_t **SR3_DLL_HPI_Move_Offset_U8**(int32_t BoardRef, uint16_t ReadWrite, char Symbol[], uint32_t DSPAddress, uint16_t MemSpace, uint32_t Offset, uint8_t Data[], int32_t Size)

10.4.7.2 Description

Cette fonction est similaire à *SR3_Base_Bulk_Move_Offset()*, sauf qu'elle s'appuie sur le matériel de l'IPH, plutôt que sur le noyau, pour effectuer le transfert.

Cette fonction ne transfère que des tableaux d'octets. Pour transférer tout autre type, la méthode la plus simple consiste à utiliser un "cast" pour permettre à ce type d'être représenté comme un tableau de U8 du côté du DSP (cast du type requis vers un tableau de U8 pour l'écrire au DSP, lecture d'un tableau de U8 et cast vers le type requis pour une lecture).

Les transferts sont limités aux emplacements de la RAM qui sont directement accessibles via le HPI. Le contrôle *MemSpace* n'est pas utilisé. Ce VI effectue des transferts dans et hors de l'espace de données et de l'espace programme. Ce VI effectuera des transferts vers n'importe quelle adresse accessible via le HPI, indépendamment de l'espace mémoire.

Note : Le noyau n'a pas besoin d'être chargé ou fonctionnel pour que ce VI s'exécute correctement. Ce VI terminera le transfert même si le DSP s'est planté, ce qui en fait un bon outil de débogage.

Les transferts avec l'IPH utilisent le control pipe 0 au lieu des bulk pipes rapides utilisés par SR3_Base_Bulk_Move_Offset. La bande passante pour ces transferts est généralement faible (500kb/s). Elle est cependant garantie.

10.4.7.3 Entrées

- **BoardRef** Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans le fichier *Structure globale d'information sur la carte*. Elle est créée par *SR3_Base_Open_Next_Avail_Board()*
- **Lecture-écriture** : 1->Lecture, 0->Écriture.
- **Symbole** : Chaîne de caractères du symbole auquel il faut accéder. Si *Symbol* est vide, *DSPAddress* et *MemSpace* sont utilisés.
- **DSPAddress** : Adresse physique de base de la DSP pour l'échange. *DSPAddress* n'est utilisé que si Le *symbole* est vide.
- **MemSpace** : Espace mémoire pour l'échange (données, programme ou IO). *MemSpace* n'est utilisé que si *Symbol* est vide ou non câblé.
- **Offset** : Représente le décalage des données à accéder par rapport à l'adresse de base indiquée par *Symbol* ou *DSPAddress*. L'adresse d'accès réelle est calculée comme la somme de l'adresse de base et du décalage. L'*offset* est utile pour accéder à des membres individuels d'une structure ou d'un tableau.
- **Données** : Tableau d'octets à écrire dans la mémoire du DSP.

- **Taille** : Représente le nombre d'octets à transférer. Pour une lecture ou une écriture, le paramètre *Data* alloué doit être supérieur ou égal à la *taille*.

10.4.7.4 Sorties

- **Données** : Tableau d'octets lus à partir de la mémoire du DSP. Le tableau *Data* transmis en argument à la fonction doit être supérieur ou égal à *Size*.

10.4.8 SR3_DLL_LoadExec_User

10.4.8.1 Prototype

int32_t **SR3_DLL_LoadExec_User**(int32_t BoardRef, char Complete_File_Path[], char File_Name[], uint32_t *EntryPoint)

10.4.8.2 Description

Cette fonction charge un code DSP utilisateur dans la mémoire du DSP et l'exécute à partir de l'adresse du point d'entrée trouvé dans le fichier COFF. Si *Complete_File_Path* et *File_Name* sont vides, une boîte de dialogue est utilisée. Le DSP est réinitialisé avant de commencer le chargement. Après le chargement du code, la table des symboles est mise à jour dans la *structure d'information globale de la carte*.

La fonction vérifie si le type de fichier COFF est adapté au DSP cible. Si ce n'est pas le cas, une erreur est générée.

Après avoir effectué le branchement vers le point d'entrée, la fonction attend un accusé de réception du DSP pour reprendre son exécution. Si ce signal ne se produit pas, la fonction se bloque. Normalement, le code DSP lancé par cette fonction doit acquitter le branchement en affirmant le signal *HINT* (voir la section sur le noyau de communication DSP).

10.4.8.3 Entrées

- **BoardRef** Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board()*
- **Complete_File_Path** : Cette chaîne représente le chemin d'accès au fichier COFF (.out) du code utilisateur DSP. Une boîte de dialogue est présentée si *Complete_File_Path* et *Firmware_File* sont tous deux vides.
- **Nom_de_fichier** : Il s'agit du nom du fichier du micrologiciel. Si *Firmware_File* n'est pas vide, la fonction résout le chemin en utilisant les règles standard pour les emplacements de microprogrammes. Si *Firmware_Files* est vide, le VI examine l'argument *Complete_File_Path*. Si *Complete_File_Path* est vide, une boîte de dialogue est affichée.

10.4.8.4 Sorties

- **Point d'entrée** : Point d'entrée où le code a été lancé. Le *point d'entrée* se trouve dans le fichier COFF.

10.4.9 SR3_DLL_Load_User

10.4.9.1 Prototype

int32_t **SR3_DLL_Load_User**(int32_t BoardRef, char Complete_File_Path[], char File_Name[], uint32_t *EntryPoint)

10.4.9.2 Description

Cette fonction charge un code DSP utilisateur dans la mémoire du DSP mais ne l'exécute pas. Elle peut être utilisée lorsque des initialisations doivent être effectuées avant l'exécution. Si *Complete_File_Path* et *File_Name* sont vides, une boîte de dialogue est utilisée. Le DSP est réinitialisé avant de commencer le chargement. Après le chargement du code, la table des symboles est mise à jour dans la *structure d'information globale de la carte*.

La fonction vérifie si le type de fichier COFF est adapté au DSP cible. Si ce n'est pas le cas, une erreur est générée.

10.4.9.3 Entrées

- **BoardRef** Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board()*
- **Complete_File_Path** : Cette chaîne représente le chemin d'accès au fichier COFF (.out) du code utilisateur DSP. Une boîte de dialogue est présentée si *Complete_File_Path* et *Firmware_File* sont tous deux vides.
- **Nom_de_fichier** : Il s'agit du nom du fichier du micrologiciel. Si *Firmware_File* n'est pas vide, la fonction résout le chemin en utilisant les règles standard pour les emplacements de microprogrammes. Si *Firmware_Files* est vide, le VI examine l'argument *Complete_File_Path*. Si *Complete_File_Path* est vide, une boîte de dialogue est affichée.

10.4.9.4 Sorties

- **Point d'entrée** : Point d'entrée du code. Le *point d'entrée* se trouve dans le fichier COFF.

10.4.10 SR3_DLL_K_Exec

10.4.10.1 Prototype

int32_t **SR3_DLL_K_Exec**(int32_t BoardRef, char Symbol[], uint32_t DSPAddress)

10.4.10.2 Description

Cette fonction force l'exécution du code DSP à passer à une adresse spécifiée, passée en argument. Si *Symbole* n'est pas vide, la fonction recherche dans la table des symboles l'adresse correspondant à l'étiquette symbolique. Si le symbole n'est pas trouvé, une erreur est générée. Si *Symbol* est une chaîne vide, la valeur passée dans *DSPAddress* est utilisée comme point d'entrée.

Après avoir effectué le branchement vers le point d'entrée, la fonction attend un accusé de réception du DSP pour reprendre son exécution. Si ce signal n'est pas émis, la fonction reste suspendue indéfiniment. Normalement, le code DSP lancé par cette fonction doit acquitter le branchement en affirmant le signal *HINT* (voir la section sur le noyau de communication DSP).

10.4.10.3 Entrées

- **BoardRef** Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans le fichier *Structure globale d'information sur la carte*. Elle est créée par *SR3_Base_Open_Next_Avail_Board()*
- **Symbole** : Chaîne de caractères du symbole auquel il faut accéder. Si *Symbol* est vide, *DSPAddress* est utilisé.
- **DSPAddress** : Point d'entrée de la fonction à exécuter. *DSPAddress* n'est utilisé que si *Symbol* est vide.

10.4.10.4 Sorties

Aucun

10.4.11 SR3_DLL_Load_UserSymbols

10.4.11.1 Prototype

int32_t **SR3_DLL_Load_UserSymbols**(int32_t BoardRef, char Complete_File_Path[], char File_Name[])

10.4.11.2 Description

Cette fonction charge la table des symboles d'un code DSP utilisateur dans la *structure d'information globale de la carte*. Si *Complete_File_Path* et *File_Name* sont vides, une boîte de dialogue est utilisée. Dans les versions précédentes de la plate-forme *SignalRanger*, cette fonction était utilisée pour obtenir le contrôle symbolique d'un code DSP qui avait été

en cours d'exécution dès la mise sous tension. Dans la nouvelle architecture, cela est moins utile car, dans ce cas, la table des symboles se trouve dans la Flash.

La fonction vérifie si le type de fichier COFF est adapté au DSP cible. Si ce n'est pas le cas, une erreur est générée.

10.4.11.3 Entrées

- **BoardRef** Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board()*
- **Complete_File_Path** : Cette chaîne représente le chemin d'accès au fichier COFF (.out) du code utilisateur DSP. Une boîte de dialogue est présentée si *Complete_File_Path* et *Firmware_File* sont tous deux vides.
- **Nom_de_fichier** : Il s'agit du nom du fichier du micrologiciel. Si *Firmware_File* n'est pas vide, la fonction résout le chemin en utilisant les règles standard pour les emplacements de microprogrammes. Si *Firmware_Files* est vide, le VI examine l'argument *Complete_File_Path*. Si *Complete_File_Path* est vide, une boîte de dialogue est affichée.

10.4.11.4 Sorties

Aucun

10.4.12 SR3_DLL_Read_Error_Count

10.4.12.1 Prototype

int32_t **SR3_DLL_Read_Error_Count**(int32_t BoardRef, uint8_t *Error_Count)

10.4.12.2 Description

Le matériel du contrôleur USB contient un compteur d'erreurs. Ce compteur circulaire de 4 bits est incrémenté chaque fois que le contrôleur détecte une erreur USB (à cause du bruit ou d'une autre raison).

Le contenu de ce compteur peut être lu périodiquement pour contrôler l'état de la connexion USB. Notez qu'une erreur USB n'entraîne généralement pas l'échec de la transaction. Le protocole USB réessaie les paquets contenant des erreurs jusqu'à trois fois au cours d'une même transaction avant de la faire échouer.

10.4.12.3 Entrées

- **BoardRef** Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board()*

10.4.12.4 Sorties

- **Compte_erreur** C'est la valeur contenue dans le compteur (entre 0 et 15).

10.4.13 SR3_DLL_Clear_Error_Count

10.4.13.1 Prototype

int32_t **SR3_DLL_Clear_Error_Count**(int32_t BoardRef)

10.4.13.2 Description

Cette fonction permet d'effacer le compteur d'erreurs USB de 4 bits.

10.4.13.3 Entrées

- **BoardRef** Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board()*

10.4.13.4 Sorties

Aucun

10.4.14 SR3_DLL_Flash_InitFlash

10.4.14.1 Prototype

int32_t **SR3_DLL_Flash_InitFlash**(int32_t BoardRef, double *FlashSize)

10.4.14.2 Description

Cette fonction télécharge et exécute le code DSP du support Flash. Le DSP est réinitialisé dans le cadre du processus de téléchargement. Tout le code DSP est abandonné. Le code de support Flash doit être en cours d'exécution pour prendre en charge les fonctions de programmation Flash. La fonction détecte également la Flash et, si elle en trouve une, elle renvoie sa taille en kWords. Si aucune mémoire flash n'est détectée, l'indicateur de taille est mis à 0.

10.4.14.3 Entrées

- **BoardRef** Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board()*

10.4.14.4 Sorties

- **FlashSize** : Cet argument renvoie la taille du Flash détecté. Si aucun flash n'est détecté, il renvoie zéro.

10.4.15 SR3_DLL_Flash_EraseFlash

10.4.15.1 Prototype

int32_t **SR3_DLL_Flash_EraseFlash**(int32_t BoardRef, uint32_t StartingAddress, uint32_t Size)

10.4.15.2 Description

Cette fonction efface le nombre requis de mots de 16 bits de la Flash, à partir de l'adresse sélectionnée. L'effacement s'effectue par secteurs, ce qui signifie que le nombre de mots effacés peut être supérieur au nombre de mots sélectionnés. Par exemple, si l'adresse de départ n'est pas le premier mot d'un secteur, les mots du même secteur précédant l'adresse de départ seront effacés. De même, si le dernier mot sélectionné pour l'effacement n'est pas le dernier mot d'un secteur, des mots supplémentaires seront effacés jusqu'à la fin du dernier secteur sélectionné. L'effacement est tel que les mots sélectionnés, y compris l'adresse de départ, sont toujours effacés.

Remarque : Sur SignalRanger_mk2_Next_Generation, la taille du secteur est de 32 mots. Sur le SignalRanger_mk3, la taille du secteur est de 64 mots (128 kBytes).

Note : L'effacement ne doit être tenté que dans les sections de la carte mémoire qui contiennent de la Flash : L'effacement ne doit être tenté que dans les sections de la carte mémoire qui contiennent de la mémoire Flash. Les tentatives d'effacement en dehors de la Flash échoueront.

10.4.15.3 Entrées

- **BoardRef** Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board()*
- **Adresse de départ** : Adresse du premier mot à effacer.
- **Taille** : Nombre de mots à effacer.

10.4.15.4 Sorties

Aucun

10.4.16 SR3_DLL_Flash_FlashMove_U8

10.4.16.1 Prototype

int32_t **SR3_DLL_Flash_FlashMove_U8**(int32_t BoardRef, uint16_t ReadWrite, char Symbol[], uint32_t DSPAddress, uint8_t Data[], int32_t Size)

10.4.16.2 Description

Cette fonction lit ou écrit un nombre illimité d'octets vers/depuis la mémoire Flash. Notez que si seule la lecture de la mémoire Flash est nécessaire, la fonction *SR3_Base_Bulk_Move_Offset()* doit être utilisée à la place, puisqu'elle ne nécessite pas la présence du code de support Flash du DSP.

Cette fonction ne transfère que des tableaux d'octets. Pour transférer tout autre type, la méthode la plus simple consiste à utiliser un "cast" pour permettre à ce type d'être représenté comme un tableau de U8 du côté du DSP (cast du type requis vers un tableau de U8 pour l'écrire au DSP, lecture d'un tableau de U8 et cast vers le type requis pour une lecture).

Toute tentative d'écriture en dehors de la mémoire Flash se soldera par un échec.

Le processus d'écriture peut transformer les uns en zéros, mais pas les zéros en uns. Si une opération d'écriture est tentée alors qu'elle devrait aboutir à la transformation d'un zéro en un, elle se solde par un échec. Normalement, un effacement doit être effectué avant l'écriture, afin que tous les bits de la zone d'écriture sélectionnée soient transformés en uns.

Note : Contrairement à la génération SignalRanger_mk2_NG, la programmation incrémentale (programmation de la même adresse plusieurs fois) est autorisée sur SignalRanger_mk3. Chaque opération de programmation ne peut pas faire passer des bits individuels de 0 à 1, ce qui ne peut être fait que par un cycle d'effacement sur un secteur entier. Cependant, les opérations de programmation peuvent remettre les bits individuels de 1 à 0 à tout moment sans effacement intermédiaire.

La représentation interne de la Flash est constituée de mots de 16 bits. En cas d'écriture d'un nombre impair d'octets, un octet supplémentaire est ajouté pour compléter l'écriture sur la frontière suivante de 16 bits. L'octet ajouté est mis à FF_H.

10.4.16.3 Entrées

- **BoardRef** Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par *SR3_Base_Open_Next_Avail_Board()*
- **Lecture-écriture** : 1->Lecture, 0->Écriture.
- **Symbole** : Chaîne de caractères du symbole auquel il faut accéder. Si *Symbol* est vide, *DSPAddress* et *MemSpace* sont utilisés.
- **DSPAddress** : Adresse physique de base de la DSP pour l'échange. *DSPAddress* n'est utilisé que si Le *symbole* est vide.
- **Données** : Tableau d'octets à lire ou à écrire dans la mémoire flash.
- **Taille** : Représente le nombre d'octets à transférer. Pour une lecture ou une écriture, le paramètre *Data* alloué doit être supérieur ou égal à la *taille*.

10.4.16.4 Sorties

- **Données** : Tableau d'octets lus ou écrits dans la mémoire flash. Le tableau *Data* transmis en argument à la fonction doit être supérieur ou égal à *Size*.

11 Développement du code DSP

Lors du développement du code DSP, deux situations peuvent se présenter :

- Le code DSP est une application complète qui n'est pas destinée à revenir au code DSP précédemment exécuté. C'est généralement le cas lorsqu'on développe une application DSP complète en C. Dans ce cas, la *fonction principale* de l'application DSP est lancée par une fonction appelée *c_int00* qui est créée par le compilateur. Lorsque (si) la *fonction principale* revient, elle retourne à une boucle sans fin à l'intérieur de *c_int00*. Il ne revient jamais au code qui s'exécutait précédemment.
- Le code DSP est une fonction simple, destinée à s'exécuter une fois, puis à revenir au code DSP (noyau ou code utilisateur) qui s'exécutait précédemment. Ce processus peut être utilisé pour forcer le DSP à exécuter de courtes fonctions de manière asynchrone par rapport à d'autres codes s'exécutant sur le DSP en arrière-plan. *L'autre code* s'exécutant en arrière-plan peut être soit le noyau, soit le code utilisateur qui a été lancé précédemment.

Plusieurs exemples sont fournis pour acquérir de l'expérience dans la programmation de la carte DSP et son interface avec une application PC. Le répertoire d'exemples contient des exemples de code DSP écrit en C, ainsi qu'en assembleur. Tout code DSP développé pour la carte SignalRanger_mk3 doit être conforme aux exigences suivantes.

11.1 Configuration de Code Composer Studio

Tous les exemples et codes pour Signal Ranger Mk3 ont été développés avec la version 4.0.1.01000 de *Code Composer Studio™*.

11.2 Exigences du projet

Dans Code Composer Studio, le projet doit être créé pour une plate-forme C64x+ little-endian.

11.3 Exigences du code C

- Lorsque vous développez une fonction en C qui sera lancée par le processus du noyau *K_Exec*, la fonction doit inclure un *accusé de réception*. Pour ce faire, ajoutez la ligne suivante dans la fonction :

```
HPI_HPIC= Accusé de réception
```

Le fichier *SR3_Reg.h* doit être inclus dans le fichier C. L'acquittement doit être exécuté dès que possible après l'entrée de la fonction. Le VI LabVIEW qui a lancé la fonction se termine dès que l'acquittement est transmis. Ces deux fichiers font partie du projet d'exemple *SR3_SignalTracker*.

Remarque : Contrairement aux noyaux SR1 et SR2, le noyau SR3 ne supporte pas de fonction déclarée comme interruption. Une fonction définie en C comme une interruption ne peut pas être lancée via le processus K_Exec.

Note : L'accusé de réception est normalement envoyé à la fin de la fonction pour signaler son achèvement. Cependant, lorsque la fonction est longue à s'exécuter, ou lorsqu'elle ne retourne pas du tout (comme dans le cas du main par exemple), l'Acknowledge doit être envoyé au début de la fonction afin que le VI LabVIEW qui a lancé la fonction puisse sortir le plus tôt possible.

11.4 Exigences en matière d'assemblage

- Toutes les fonctions développées en assembleur doivent inclure un *acquittement*. L'*acquittement* est envoyé en écrivant la valeur constante *acknow_asm* dans le registre *HPI_HPIC*. Il y a plusieurs façons de procéder. L'une d'entre elles est décrite ci-dessous.

```
MVKL .S2     HPI_HPIC,B0
MVKH .S2     HPI_HPIC,B0
MVK    .S2   acknow_asm,B1
STW    .D2   B1,*B0
```

Le fichier *ASM_Definition_SR3.inc* résout les symboles *HPI_HPIC* et *acknow_asm*. Il doit être inclus dans le fichier d'assemblage.

- Pour être *compatible avec le lancement du noyau*, une fonction n'a pas besoin de caractéristiques particulières. Toute la protection contextuelle requise est assurée par le noyau avant le lancement. Par défaut, la fonction ne peut pas être interrompue. Mais le bit d'activation d'interruption globale (GEI bit du registre CSR) peut être mis à 1 pour permettre à la fonction d'être interrompue.
- Lors du développement d'une section de code en assembleur, la directive *.align 32* doit être utilisée au début de la section. Cela permet de s'assurer que les sections de code commencent toujours à une adresse de 32 octets. Cela n'est pas nécessaire lors du développement de code en C, car le compilateur C gère l'alignement. Cette exigence ne s'applique qu'aux sections de code.
- Le noyau utilise B15 comme pile logicielle. Le code assembleur ne doit pas utiliser B15 à d'autres fins.

Remarque : Contrairement aux noyaux SR1 et SR2, une fonction à lancer par le processus K_Exec ne doit pas être construite comme une fonction d'interruption.

11.5 Options de construction

Remarque : Les exemples fournis servent de guide.

11.5.1 Compilateur

- **Base :**
  - Utilisez la version cible : C64x+
- **Options du modèle d'exécution :**
  - Ne pas activer l'option *Générer un code big endian*

11.5.2 Lien

- **Options de base :**
  - Fixer la taille de la pile à 0x1000 au moins
- **Environnement d'exécution :**
  - Utiliser le *lien en utilisant le modèle d'auto-initialisation de la RAM*. Ce modèle est plus efficace en termes d'espace.

11.6 Modules requis

11.6.1 Vecteurs d'interruption

- La section du vecteur d'interruption est toujours située à l'adresse : 0x10E08000 (le début de la L1P-RAM).
- Le vecteur INT4 (à l'adresse 0x10E08080) est réservé à l'interruption *HPIInt* et au noyau.
- Tous les vecteurs doivent avoir une longueur de 32 octets. L'extrait de code ci-dessous montre une façon typique de définir un vecteur :

```
    .global _SR3AIC
    .sect .vectors
    .nocmp

_ISRINT5 :

      STW      .D2T2     B10,*B15--[2]
   || MVKL .S2 _SR3AIC,B10 MVKH
      .S2 _SR3AIC,B10 B       .S2 B10
      LDW      .D2T2 *++B15[2],B10
      NOP      4
      NOP
      NOP
```

.fin

La directive .nocmp est utilisée pour éviter l'instruction compacte du 64x+. Cette section vectorielle doit être liée à l'adresse 0x10E080A0 et sa taille est exactement de 32 octets. Pour l'ISR, une fonction C doit être déclarée avec le qualificateur d'*interruption* et le nom SR3AIC.

- Si le registre *INTC_INTMUX1* doit être modifié, les numéros d'événements pour INT4 doivent être conservés. Le contenu du *registre INTC_INTMUX1* doit être de la forme xxxxxx2Fh.
- Si aucune interruption (autre que l'interruption utilisée par le noyau) n'est utilisée dans le projet, il n'est normalement pas nécessaire de fournir une section de *vecteurs*. Cependant, nous suggérons d'inclure un simple vecteur vide (voir ci-dessous) pour s'assurer que le Run-Time-Support (RTS) n'inclura pas sa propre section de vecteurs qui pourrait ne pas être compatible avec le noyau. Cette section de vecteurs simple doit être liée à l'adresse 0x10E080C0 :

```
        .sect         .vecteurs

.nocmp

SimpleISR :

        B       IRP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

        .fin
```

11.7 Exigences en matière de liens

11.7.1 Mémoire Description Fichier

Le fichier de commande de description de l'éditeur de liens *CMD_SR3.cmd* peut être utilisé comme point de départ. Ce fichier décrit la carte mémoire de la carte Signal Ranger_mk3. Il inclut les plages d'adresses qui sont réservées et ne doivent pas être modifiées. Ce fichier fait partie de l'exemple de code DSP *SR3_SignalTracker*.

11.7.2 Éviter les piles

Le noyau initialise la pile à l'adresse 0x10F17FF8 dans la L1DRAM et utilise 128 octets. Si des données initialisées sont définies dans cette zone, la pile sera corrompue pendant le processus de chargement. Le chargement ne pourra pas s'effectuer correctement.

Pour éviter cette situation, assurez-vous que le fichier de commandes de l'éditeur de liens ne définit aucune donnée initialisée entre les adresses 0x10F17F78 et 0x10F17FF8.

11.8 Symboles mondiaux

Seuls les symboles globaux trouvés dans le fichier exécutable DSP sont conservés dans la table des symboles. Cela signifie que pour permettre l'accès symbolique à l'interface logicielle, les variables déclarées en C doivent être déclarées globales (en dehors de tous les blocs et fonctions, et sans le mot-clé *static*). Les variables et les étiquettes déclarées en assemblage (points d'entrée de fonction par exemple) doivent être déclarées avec la directive d'assemblage *.global*.

11.9 Préparation du code pour le "Self-Boot" (auto-démarrage)

Le circuit Flash embarqué peut être programmé pour charger le code DSP et/ou la logique FPGA à la mise sous tension, et pour lancer l'exécution du code DSP spécifié. De plus amples informations sur les tables de démarrage du DSP et du FPGA Flash sont disponibles dans les sections suivantes.

Une fois que le code DSP et/ou la logique FPGA ont été développés et testés sous le contrôle du PC (éventuellement à l'aide du mini-débogueur), ils peuvent être programmés en Flash à l'aide des fonctions appropriées du mini-débogueur.

Le code DSP chargé dans la Flash doit contenir un acquittement (voir exemples) à son début, même lorsque le code est lancé à partir de la Flash. Habituellement, lors du développement en C, cet acquittement est placé dans les premières lignes de la fonction principale.

Le fait de ne pas inclure l'accusé de réception n'entraîne pas le plantage du code DSP. Cependant, le contrôleur USB ne reconnaît pas que le noyau de mise sous tension a été chargé. Dans ce cas, il est nécessaire de réinitialiser la carte afin d'obtenir l'accès du PC. Cette réinitialisation entraînerait à son tour l'interruption du code DSP chargé au démarrage.

Note : L'acquittement est également nécessaire pour le code qui est écrit pour être chargé directement à partir du PC et exécuté en utilisant le VI d'interface SR3_K_Exec.vi, ou en utilisant le bouton Exec du mini-débogueur. Par conséquent, le code DSP qui a été développé et testé à l'aide du mini-débogueur, ou le code qui est habituellement téléchargé et exécuté à partir du PC devrait normalement être prêt à être programmé dans la table d'amorçage "tel quel".

Pour que le code DSP programmé dans la Flash puisse démarrer correctement, son point d'entrée doit être correctement défini dans la liaison. Il ne s'agit pas d'une exigence absolue pour le code chargé à partir du PC. En effet, le code chargé à partir du PC peut être lancé à partir de n'importe quelle étiquette existante, ou même d'une adresse arbitraire, en utilisant soit le mini-débogueur, soit les interfaces LabVIEW ou C/C++. Cependant, le chargeur de démarrage qui s'exécute à partir du noyau ne lance le code qu'à partir du point d'entrée défini. Si aucun point d'entrée n'est défini, le code du DSP se bloquera très probablement à la mise sous tension.

11.10 Sous le capot

Cette section fournit des informations détaillées sur le fonctionnement du contrôleur USB et du noyau de communication. Elle est destinée aux rares développeurs qui ont besoin de descendre à ce niveau de détail, ou à ceux qui souhaitent mieux comprendre le fonctionnement du système. Une bonne compréhension de cette section est nécessaire pour les développeurs qui veulent implémenter les fonctions *SR3_Base_User_Move_Offset*.

11.10.1 Processus de démarrage

Lors de la mise sous tension, le processus de démarrage effectue les opérations suivantes :

- Le contrôleur USB se met sous tension. Il charge le *Power-Up_Kernel* dans la mémoire du DSP et commence son exécution.
- Le *Power-Up_Kernel* effectue ensuite les opérations suivantes :
    - Les registres CFG_PINMUX0 et CFG_PINMUX01 sont tous deux réglés pour obtenir la sélection des broches matérielles suivantes : HPI, EMAC (mode RMII), EMIFA 16 bits en CS2/CS3, UART0 et UART1 sans flux de contrôle matériel, McBSP0 et McBSP1 avec clks complets, PWM0, PWM1 et ClockOut0.
    - Initialiser *le contrôleur de mise en veille* (PSC) pour activer tous les modules.
    - Initialiser le registre *INTC_INTMUX1* pour lier l'événement 47 (interruption HPI) à l'interruption INT4.
    - Initialiser le vecteur d'interruption et le registre *IER* pour l'interruption INT4.
    - Initialiser PLL1 pour obtenir une horloge de 589,824 MHz et initialiser les diviseurs à /1, /3 et /6.
    - Configurer le canal EDMA-63 utilisé pour les lectures et écritures du noyau. Le noyau utilise les structures *PaRAM63 et 127*. L'événement et l'interruption sont utilisés pour le canal 63.
    - Configurer l'EMIFA pour gérer la flash dans la section mémoire CS2. L'horloge principale de la flash est de 98,304 MHz.
    - Initialiser le pointeur de pile (registre *B15*) à l'adresse 0x10F17FF8 (première adresse alignée sur 8 octets à la fin de la L1DRAM).

- Initialiser le pointeur de données (registre *B14*) à l'adresse 0x10F04000 (début de la L1DRAM).
- Initialiser l'*ISTP* (pointeur de table vectorielle) à 0x10E08000 (début de la L1PRAM).
- Lancez le compteur *TSC*. Ce compteur de 64 bits fonctionne à la vitesse du CPU - 589,824 MHz, et peut être utilisé pour chronométrer le code DSP. Voir le pilote flash pour des exemples et des fonctions permettant de chronométrer le code DSP à l'aide de ce compteur.
- Vérifier la présence d'un code DSP utilisateur dans la mémoire flash.
  - Si un code DSP est présent, le code utilisateur est chargé et lancé. Le noyau est toujours résident et prêt à répondre aux demandes du PC.
  - Si aucun code utilisateur n'est détecté, le noyau entre dans une simple boucle d'attente, prêt à répondre aux demandes du PC.

11.10.2 Connexion PC

Chaque fois que la carte est connectée au PC, le contrôleur USB énumère la carte au PC, qui lance le pilote de la carte et permet au PC de prendre le contrôle de la carte. Cette étape ne perturbe pas, ni même n'interagit d'aucune manière avec le code DSP qui peut déjà être en cours d'exécution.

Cependant, à partir de ce moment, le PC peut faire des demandes au noyau, qui peut lire et écrire dans la mémoire du DSP, ou exécuter des fonctions dans la mémoire du DSP.

Lorsqu'une application PC ouvre la carte, elle lit la Flash pour détecter si un code DSP est résident. Si c'est le cas, il charge sa table de symboles à partir de la Flash. L'application a alors un accès symbolique au code DSP.

11.10.3 PC-Reset

Une fois la carte connectée au PC, ce dernier peut réinitialiser le DSP à tout moment. A partir de ce moment, il charge le *Host-Download_Kernel*. Le *Host-Download_Kernel* effectue exactement les mêmes opérations que le *Power-Up_Kernel (voir ci-dessus)*, sauf qu'il ne recherche pas le code DSP dans la Flash. Il attend simplement les demandes de noyau du PC. Ce schéma simple permet au PC de prendre positivement le contrôle du DSP à tout moment, même dans les situations où le code du DSP s'est écrasé. Chaque fois que le DSP est réinitialisé, la table de symboles qui aurait pu être en vigueur précédemment est vidée. Une nouvelle table de symboles peut être mise en œuvre à l'avenir si un nouveau code est chargé.

11.10.4 Ressources utilisées par le noyau

Pour fonctionner correctement, le noyau utilise les ressources suivantes sur le DSP. Après le lancement du code utilisateur, ces ressources ne doivent pas être utilisées ou modifiées, afin d'éviter d'interférer avec le fonctionnement du noyau et de conserver toutes ses fonctionnalités.

- Le noyau réside entre les adresses 0x10E08000 et 0x10E08FFF dans la L1PRAM du DSP. L'utilisateur doit éviter de charger du code ou de modifier la mémoire en dessous de l'adresse 0x10E09000 dans la L1PRAM.
- La boîte aux lettres du noyau se trouve entre les adresses 0x10F04000 et 0x10F0420B dans la L1DRAM. Cette section est réservée et de nouvelles données ne peuvent y être chargées. Toutefois, une fonction utilisateur peut utiliser la boîte aux lettres pour développer une fonction de niveau 3 du noyau.
- Le noyau localise la table des vecteurs d'interruption à l'adresse 0x10E08000 dans la L1PRAM. Le code utilisateur ne doit pas déplacer les vecteurs d'interruption ailleurs.
- Les vecteurs d'interruption de l'adresse 0x10E08000 à 0x10E0809F ne peuvent pas être modifiés. Tous les nouveaux vecteurs doivent être définis à partir de l'adresse 0x10E080A0.
- Le PC (via le contrôleur USB) utilise l'interruption HPI_Int du HPI pour demander un accès au DSP. L'événement du HPI est lié à l'interruption masquable INT4. Si nécessaire, le code utilisateur peut désactiver temporairement l'interruption HPI_Int (ou INT4) par le biais de son masque ou du masque d'interruption global (GEI par du registre CSR). Pendant que cette interruption est désactivée, toutes les demandes d'accès au PC sont mises en latence, mais sont maintenues jusqu'à ce que l'interruption soit réactivée. Une fois l'interruption réactivée, la demande d'accès reprend normalement.

- Le noyau initialise la pile à l'adresse 0x10F17FF8. Le noyau utilise moins de 128 octets de la pile (entre les adresses 0x10F17F78 et 0x10F17FF8). Le code utilisateur peut déplacer temporairement le pointeur de pile, mais il doit le remplacer avant le dernier retour au noyau (si ce dernier retour est prévu). Parmi les exemples où le dernier retour au noyau n'est pas prévu, on peut citer les situations où le code utilisateur est une boucle sans fin qui sera interrompue par une réinitialisation de la carte ou un arrêt de l'alimentation. Dans ces cas, la pile peut être déplacée librement sans inquiétude.

Note : Lorsque l'on passe au point d'entrée d'un programme développé en C, le DSP exécute d'abord une fonction appelée c_int00, qui établit de nouvelles piles, ainsi que l'environnement C. Cette fonction appelle ensuite la fonction définie par l'utilisateur "main". Cette fonction appelle ensuite la fonction "main" définie par l'utilisateur. Lorsque main cesse de s'exécuter (en supposant qu'il ne s'agisse pas d'une boucle sans fin), il retourne à une boucle sans fin au sein de la fonction c_int00. Il ne retourne pas au noyau.

- Pour gérer les transferts de données, le noyau utilise le canal EDMA-63 (file d'attente n°2). L'événement et l'interruption sont activés pour le canal 63 dans la région 1. Le canal EDMA-63 ne peut pas être utilisé par le code utilisateur. De même, si l'interruption globale EDMA est utilisée par le code utilisateur, le registre d'interruption en attente (*EDMA_IPR*) doit être vérifié pour éviter de gérer une interruption déclenchée par le canal EDMA-63. Notez que le noyau n'utilise pas d'interruption DSP pour la région 1 de l'EDMA ; une technique d'interrogation est utilisée. Ainsi, l'interruption de la région 1 de l'EDMA est toujours disponible pour l'utilisateur mais, comme pour l'interruption globale de l'EDMA, le code utilisateur doit vérifier le registre d'interruption en attente (*EDMA_IPRH* - région 1) pour éviter de servir une interruption déclenchée par le canal 63 de l'EDMA.
- Les registres *CFG_PINMUX0* et *CFG_PINMUX01* sont définis par le noyau et ne doivent pas être modifiés par le code utilisateur.

11.10.5 Communications USB

Grâce à sa connexion USB à la carte, le PC peut lire et écrire la mémoire du DSP et lancer l'exécution du code du DSP. Les communications USB PC-DSP utilisent deux mécanismes :

- Le PC utilise le tuyau de contrôle 0, via les *demandes du fournisseur* USB, pour effectuer les opérations suivantes :
    - Réinitialiser le DSP
    - Modifier la couleur de la LED
    - Lecture ou écriture de la mémoire DSP sur puce à l'aide du matériel HPI.
    - Lire ou écrire divers registres, tels que *DSPState* et *USB Error Count*.
    - Chargez le noyau *Host_Download* pour permettre à l'hôte de prendre le contrôle de la carte DSP.

  Ce mécanisme n'utilise que le matériel de l'IPH.
- Le PC utilise les conduites de masse à grande vitesse 2 (sortie) et 6 (entrée) pour transférer des données vers et à partir de n'importe quel emplacement dans n'importe quel espace DSP, ainsi que pour lancer l'exécution du code DSP de l'utilisateur. Ce mécanisme utilise les fonctions du noyau DSP résident.

Les opérations effectuées à l'aide du tuyau de contrôle zéro sont lentes et limitées, mais très fiables. Elles permettent au PC de prendre le contrôle du DSP à un niveau très bas. En particulier, les transferts de mémoire ne dépendent pas de l'exécution d'un code noyau sur le DSP. Toutes ces opérations peuvent être effectuées même après que le code du DSP se soit écrasé.

Les opérations effectuées à l'aide des conduites de masse à grande vitesse 2 et 6 sont prises en charge par le noyau résident du DSP. Elles permettent d'accéder à n'importe quel emplacement dans n'importe quel espace mémoire du DSP. Les transferts sont beaucoup plus rapides qu'avec le control pipe 0. Cependant, ils dépendent de l'exécution du noyau. Ce dernier doit être chargé et fonctionner avant que l'une de ces opérations puisse être tentée. Ces opérations peuvent ne pas fonctionner correctement lorsque le code du DSP s'est écrasé.

Les opérations effectuées sur le DSP par l'intermédiaire du noyau doivent suivre un protocole décrit dans les sections suivantes.

11.10.5.1 Communications via le tuyau de contrôle 0

Les demandes suivantes du fournisseur concernent les opérations que le PC peut effectuer sur la carte DSP par l'intermédiaire de la conduite de commande 0. Toutes ces opérations sont encapsulées dans des fonctions de bibliothèque dans les interfaces logicielles du PC.

Demande	Direction	Code	Action
DSPReset	Sortie	10_H	Affecte ou libère la réinitialisation du DSP. *NOTE : L'affirmation de la réinitialisation du DSP remet également à zéro les variables KPresent et K_State, indiquant que le noyau n'est pas présent (voir ci-dessous).* wValue = 1 : assert / 0 : release. wIndex = N/A wLongueur = N/A
DSPInt	Sortie	11_H	Envoyer une interruption DSPInt via le processus d'interruption HPI (vecteur d'interruption $xx64_{H)}$.
W_Leds	Sortie	12_H	Modifier la couleur de la LED bicolore (verte, rouge, orange ou éteinte). wValeur = 0 : désactivé wValeur = 1 : rouge wValue = 2 : vert wValue = 3 : orange wIndex = N/A wLength = N/A
HPIMove	Entrée/sortie	13_H	Lecture ou écriture de 4 à 4096 octets dans la mémoire du DSP accessible par le HPI. wValue = Adresse de transfert inférieure dans la carte mémoire du DSP. wIndex = Adresse de transfert supérieure dans la carte mémoire du DSP. wLength = Nb d'octets à transférer (doit être pair) DataBlock De 4 à 4096 octets peuvent être transportés dans la phase de données de la demande. Le nombre doit être un multiple de 4 octets. *NOTE : Pour les transferts OUT, l'adresse stockée dans wIndex-wValue doit être prédécrémentée (c'est-à-dire qu'elle doit pointer vers l'adresse juste avant que le premier mot ne soit décrémenté).* *écrit).*
W_HPI_Control	Sortie	14_H	Écrire le registre de contrôle HPI. Ceci peut être utilisé pour interrompre le DSP (DSPInt), ou pour effacer l'interruption HINT (DSP to Host interrupt). *Note : Les signaux DSPInt et HINT sont utilisés par le noyau pour communiquer avec le PC. Les développeurs ne devraient essayer de les utiliser que s'ils en comprennent les conséquences (voir la description du noyau).* *Le bit BOB (bits 0 et 8 du registre de contrôle) doit toujours être activé (jamais désactivé). Sinon, l'interface DSP ne fonctionnera pas correctement.* wValue = Mot de 16 bits à écrire dans le PCSI NOTE : Les deux octets LSB et MSB doivent être identiques. wIndex= N/A

			wLongueur = N/A
Set_HPI_Speed	Sortie	15_H	Définit la vitesse de l'IPH comme lente ou rapide. *Note : La vitesse du HPI est automatiquement réglée sur lente à la mise sous tension et à chaque fois que le DSP est réinitialisé par la commande DSPReset. Utilisez cette commande pour mettre le HPI en vitesse rapide après l'un ou l'autre de ces événements.* wValue = 1 : Rapide / 0 : Lent. wIndex = N/A wLongueur = N/A
Déplacer_DSPState	Entrée/sortie	20_H	Lit ou écrit l'état du DSP. wValue = N/A wIndex = N/A wLongueur = N/A DataBlock : 2 octets représentant l'état du DSP : bKpresent (octet) : - Le noyau n'est pas chargé -> 0 - Power-Up Kernel Loaded -> 1 - Host-Download Kernel Loaded -> 2 La variable Kpresent est à 0 lors de la mise sous tension. Il faut quelques secondes après la mise sous tension pour que le contrôleur USB charge et lance le noyau. L'hôte doit interroger cette variable après l'ouverture du pilote et différer les accès au noyau jusqu'à ce que celui-ci soit chargé. bKstate (octet) : - Kernel_Idle -> 0
R_ErrCount	En	22_H	Renvoie le nombre d'erreurs USB wValue = N/A wIndex = N/A wLongueur = N/A Bloc de données 1 mot est transporté dans le bloc de données. Ce mot représente le nombre actuel d'erreurs USB (entre 0 et 15).
Reset_ErrCount	Sortie	23_H	Remet à zéro le registre de comptage des erreurs USB.

Tableau 6 Demandes des fournisseurs

11.10.5.2 Communications via le noyau DSP

Le noyau de communication améliore les communications avec le PC. Les échanges de mémoire sans le noyau sont limités à l'espace mémoire directement accessible à partir du HPI et les accès sont limités à des mots de 32 bits. La redirection de l'exécution du DSP est limitée au chargement du code à une adresse fixe immédiatement après la réinitialisation. Le noyau permet des lectures et des écritures vers/depuis n'importe quel emplacement de la carte mémoire du système, et permet la redirection de l'exécution à tout moment, à partir de n'importe quel emplacement, même de manière ré-entrante.

En fait, deux noyaux peuvent être utilisés à différents moments de la vie d'une application DSP :

- Immédiatement après la mise sous tension, le contrôleur USB charge un *noyau de mise sous tension* dans la mémoire du DSP et l'exécute. Le contrôleur USB exécute cette fonction seul, qu'un PC hôte soit connecté à la carte ou non. Le fonctionnement du noyau est indiqué par la diode électroluminescente qui devient orange. Après cela, la commande *READ_DSPState* Vendor renverra une valeur *KPresent* de 1 (voir Commandes Vendor ci-dessus).

Remarque : l'hôte ne doit invoquer les commandes du noyau qu'après que la variable KPresent a atteint une valeur non nulle.

Ce noyau de mise sous tension remplit les fonctions suivantes :

- Il vérifie dans la mémoire Flash si un fichier descripteur FPGA est présent, et si c'est le cas, il charge le FPGA avec la logique correspondante.
- Il vérifie ensuite dans la mémoire Flash si un fichier DSP exécutable est présent, et si c'est le cas, il le charge et l'exécute.
- Il reste résident pour répondre aux commandes du noyau de l'hôte (K_Read, K_Write et K_Exec - voir ci-dessous) une fois que la carte a été connectée à un PC.

- Chaque fois que la carte est connectée à un PC, ce dernier peut réinitialiser la carte et charger un *noyau de téléchargement hôte* plus simple dans la mémoire à tout moment. Ce noyau de téléchargement hôte ne vérifie pas la logique FPGA ou le code DSP dans la mémoire Flash. Il attend seulement les commandes K_Read, K_Write et K_Exec de l'hôte et y répond. Cela permet au PC hôte de prendre le contrôle du DSP sans interférence avec le code embarqué, et de recharger la logique FPGA et le code DSP différents de ceux décrits dans la mémoire Flash. Cela est notamment nécessaire pour reprogrammer la mémoire Flash avec une nouvelle logique FPGA et/ou un nouveau code DSP.

Une fois que l'un ou l'autre de ces noyaux est en ligne, le PC hôte peut envoyer des commandes K_Read, K_Write et K_Exec. Chaque commande lance une opération du DSP (déplacement de données ou branchement de code) et attend que le DSP accuse réception de l'opération. Les fonctions intrinsèques du noyau qui prennent en charge les commandes K_Read et K_Write incluent cet accusé de réception. Le code DSP utilisateur lancé par la commande K_Exec doit absolument inclure l'accusé de réception. Pour les fonctions DSP utilisateur invoquées par la commande K_Exec, il est possible (volontairement ou non) que l'accusé de réception prenne beaucoup de temps à être renvoyé. La fonction du PC qui lance les demandes attend l'accusé de réception pour sortir. C'est pourquoi l'accusé de réception doit être renvoyé dans un délai raisonnable (5 secondes sont conseillées). Normalement, l'accusé de réception est utilisé pour signaler l'achèvement de la fonction, mais pour les fonctions qui prennent beaucoup de temps pour s'achever, l'accusé de réception devrait être renvoyé au début de la fonction (pour signaler le branchement), et un autre moyen de signaler l'achèvement devrait être envisagé (interrogation d'un drapeau d'achèvement dans la mémoire du DSP, par exemple).

11.10.5.3 Tables d'amorçage des DSP et FPGA

Pour a détaillée description de la DSP et du FPGA d'amorçage FPGA voir le document *Signal_Ranger_mk3_Boot_Tables.pdf*.

11.10.5.4 Vitesse de signalisation de l'IPH

Sur le *Signal Ranger_mk3*, la vitesse de signalisation du HPI doit être lente immédiatement après que le DSP est sorti de la réinitialisation. Cela comprend la réinitialisation après la mise sous tension et la réception de la commande vendeur *DSPReset*. En effet, dans ces circonstances, le DSP (et le HPI) fonctionne à faible vitesse. Ce n'est qu'une fois que le noyau de mise sous tension ou de téléchargement de l'hôte a été chargé et qu'il a eu le temps d'ajuster la vitesse du CPU et du HPI au maximum pour le DSP que le contrôleur USB peut utiliser la signalisation HPI rapide. Cette commande est normalement utilisée pour régler la vitesse HPI au maximum après que le noyau de mise sous tension a été détecté ou après que le noyau de téléchargement de l'hôte a été téléchargé. Notez qu'il peut s'écouler jusqu'à 500us après que le noyau ait ajusté la PLL, jusqu'à ce que le DSP et le HPI soient cadencés à l'aide de la vitesse rapide. Le logiciel PC doit donc attendre au moins cette durée avant d'envoyer la commande. Le passage à la vitesse de signalisation élevée du HPI est automatiquement effectué par les fonctions d'initialisation de la carte des interfaces LabVIEW et C/C++.

11.10.6 Noyau de communication SR3 DSP

11.10.6.1 Différences entre SR2_NG et SR3

La plate-forme SR2_NG est identique à la plate-forme SR2 en ce qui concerne le matériel et le micrologiciel DSP. Voir la documentation du SR2 pour plus de détails sur le fonctionnement de son noyau. Il existe plusieurs différences entre les noyaux utilisés dans les plateformes SR3 et SR2_NG :

11.10.6.1.1 Emplacement de la boîte aux lettres

Dans SR3, la MailBox est située à l'adresse $10F04000_H$.

11.10.6.1.2 Contenu de la boîte aux lettres

Dans SR3, le paramètre *NbBytes* remplace le paramètre *NbWords.* Il représente désormais le nombre d'octets à transférer.

Dans le SR3, le *ControlCode* remplace le paramètre *ErrorCode* et a une importance accrue. Il indique désormais la taille totale du bloc du transfert, en plus du type d'opération (*K_Read*, *K_Write* ou *K_Exec*). Voir ci-dessous pour l'encodage précis.

11.10.6.2 Aperçu du noyau SR3

Le noyau utilisé pour prendre en charge les communications PC est extrêmement polyvalent, tout en utilisant des ressources DSP minimales en termes de mémoire et de temps de traitement. Les accès provenant du PC attendent l'achèvement des processus à temps critique fonctionnant sur le DSP, ce qui minimise les interférences entre les accès du PC et les processus en temps réel du DSP.

Trois commandes (*K_Read*, *K_Write* et *K_Exec*) sont utilisées pour déclencher toutes les opérations du noyau.

L'échange de données et de commandes entre le PC et le DSP se fait par l'intermédiaire d'une zone de boîte aux lettres de 524 octets dans la mémoire vive du DSP.

L'interface DSP fonctionne sur 3 niveaux distincts :

- **Niveau 1** : Au niveau 1, le noyau n'a pas encore été chargé sur le DSP. Le PC s'appuie sur le matériel du DSP (HPI et DMA), ainsi que sur le contrôleur USB, pour échanger du code et/ou des données avec la RAM du DSP. À ce niveau, le PC n'a qu'un accès limité à la mémoire vive du DSP. Ce niveau est utilisé, entre autres, pour télécharger le noyau dans la RAM du DSP et lancer son exécution. Les fonctions de ce niveau sont également utiles dans les situations de débogage difficiles, car elles ne dépendent pas de l'exécution du code sur le DSP.
- **Niveau 2** : Au niveau 2, le noyau est chargé et fonctionne sur le DSP. Grâce aux fonctions intrinsèques du noyau, le PC peut accéder à n'importe quel emplacement dans n'importe quel espace mémoire du DSP et peut lancer le code du DSP à partir d'un point d'entrée situé n'importe où dans la mémoire. Ce niveau est utilisé pour charger le code utilisateur dans la mémoire du DSP et le lancer. Les fonctions de niveau 1 sont toujours fonctionnelles au niveau 2, mais elles sont rarement utilisées car les fonctions de niveau 2 offrent un meilleur accès. Cependant, un avantage possible des fonctions de niveau 1 est qu'elles ne dépendent pas du logiciel du DSP. Par conséquent, elles réussissent toujours, même lorsque le code du DSP est bloqué.
- **Niveau 3 :** Le niveau 3 est défini lorsque le code utilisateur est chargé et exécuté sur le DSP. Il n'y a pas de différence fonctionnelle entre les niveaux 2 et 3. Les fonctions des niveaux 1 et 2 sont toujours disponibles pour assurer l'échange de données entre le PC et le DSP, et pour rediriger l'exécution du code utilisateur du DSP. La principale différence est qu'au niveau 3, les fonctions utilisateur sont également disponibles, en plus des fonctions intrinsèques du noyau. Au niveau 3, lorsque le code utilisateur est en cours d'exécution, la commande K_Exec de niveau 2 peut toujours être invoquée pour forcer l'exécution du DSP à passer à une nouvelle adresse.

11.10.6.3 Description fonctionnelle du noyau

Une fois que le noyau de mise sous tension a fini d'initialiser le DSP, s'il trouve du code DSP dans la mémoire Flash, il le charge et l'exécute. Ce code DSP est normalement en cours d'exécution lorsque l'utilisateur prend le contrôle de la carte DSP.

Après que le noyau Host-Download a fini d'initialiser le DSP, ou après que le noyau Power-Up a fini d'initialiser le DSP et n'a pas trouvé de code DSP dans la mémoire Flash, le noyau est normalement en cours d'exécution lorsque l'utilisateur prend le contrôle du DSP. Le noyau est simplement une boucle sans fin qui attend la prochaine demande d'accès du PC. Les demandes d'accès du PC sont déclenchées par l'interruption DSPInt du HPI.

11.10.6.3.1 Lancement d'une fonction DSP

Aux niveaux 2 et 3, le protocole du noyau ne définit qu'un seul type d'action, qui est utilisé pour lire et écrire la mémoire du DSP, ainsi que pour lancer une fonction simple (une fonction qui inclut un retour au noyau ou au code précédemment exécuté) ou un programme complet (une fonction sans fin qui n'est pas destinée à revenir au noyau ou au code précédemment exécuté). En fait, une lecture ou une écriture en mémoire s'effectue en lançant une fonction *ReadMem* ou *WriteMem*, qui appartient au noyau (fonction intrinsèque) et réside dans la mémoire du DSP. Le lancement d'une fonction utilisateur utilise le même processus de base, mais nécessite que la fonction utilisateur soit chargée dans la mémoire du processeur avant le branchement.

La boîte aux lettres est une zone de la mémoire vive du DSP

accessible par le HPI. La fonction de chaque champ de la boîte

aux lettres est décrite ci-dessous.

Adresse	Nom	Fonction
$10F04000_H$	Adresse de la succursale	32 bits - adresse de branchement (fonctions intrinsèques de lecture et d'écriture, ou fonction utilisateur)
$10F04004_H$	Adresse de transfert	32 bits - adresse de transfert (pour les commandes K_Read et K_Write)
$10F04008_H$	NbOctets	16 bits - nombre de mots à transférer
$10F0400A_H$	Code de contrôle	Ce code contient la commande (*K_Read*, *K_Write* ou *K_Exec*) dans les bits 0 et 1. 00 >* 01 >* 10 >* Il contient également la taille du bloc en octets pour le transfert (*K_Read*, *K_Write*). La taille du bloc est stockée en notation binaire non signée dans les bits 2 à 15, alignés sur le bit 0 à droite. Cela signifie qu'il suffit de masquer les bits 0 et 1 dans le *code de contrôle* pour obtenir la taille du bloc. La taille du bloc est normalement de 512 octets pour une connexion à haut débit et de 64 octets pour une connexion à haut débit.
$10F0400C_H$	Données	512 octets de données utilisés pour les transferts entre le PC et le DSP. Seuls les 64 premiers octets sont utilisés lorsque la carte est connectée en tant que périphérique USB à grande vitesse.

Tableau 7 Boîte aux lettres

Pour lancer une fonction DSP (code intrinsèque ou utilisateur), le PC, via le contrôleur USB, lance une commande *K_Read*, *K_Write* ou *K_Exec.* Cette commande contient des informations sur l'adresse DSP de la fonction à exécuter (utilisateur ou intrinsèque). Pour *K_Read* et *K_Write*, elle contient également des informations sur l'adresse de transfert et, dans le cas d'un transfert en écriture, elle contient les octets de données à écrire au DSP.

L'exécution d'une telle commande se fait en 3 étapes :

11.10.6.3.1.1 Étape 1

- Le PC envoie un *paquet de configuration* à la carte DSP. Ce paquet de configuration contient des informations qui définissent la commande :

Octet Nb	Données
0	BranchAddress octet 2
1	BranchAddress octet 3 (MSB)
2	BranchAddress octet 0 (LSB)
3	BranchAddress octet 1
4	TransferAddress octet 2
5	TransferAddress octet 3 (MSB)
6	TransferAddress byte 0 (LSB)
7	TransferAddress octet 1
8	NbOctets (LSB)
9	NbOctets (MSB)
10	Code de contrôle (LSB)
11	Code de contrôle (MSB)

Note : Le code de contrôle ne doit inclure la commande que dans les bits 0 et 1 à ce stade : Le code de contrôle ne doit inclure la commande que dans les bits 0 et 1 à ce stade. Le contrôleur USB embarqué ajoute la taille du bloc qui dépend de la connexion.

- Le contrôleur USB reçoit le *paquet d'installation* et écrit les informations pertinentes dans la zone d'en-tête de la boîte aux lettres. Cet en-tête se compose des éléments suivants
    - *Adresse de la succursale*
    - *Adresse de transfert*
    - *NbOctets*
    - *Code de contrôle*

Note : Le contrôleur USB ajoute la taille du bloc au ControlCode envoyé par le PC : Le contrôleur USB ajoute la taille du bloc au ControlCode envoyé par le PC.

- Dans le cas d'un *K_Write*, le contrôleur USB écrit les octets reçus dans le paquet USB actuel dans le champ *Data* de la boîte aux lettres.
- Ensuite, le contrôleur USB efface le signal *HINT* (interruption de l'hôte), qui sert d'accusé de réception de la fonction DSP.
- Le contrôleur USB envoie alors une interruption *DSPInt* au DSP, ce qui oblige le DSP à passer à la fonction intrinsèque ou utilisateur.

11.10.6.3.1.2 Étape 2

- Si le DSP n'est pas interruptible (parce que l'interruption *DSPInt* est temporairement masquée ou parce que le DSP sert déjà une autre interruption), l'interruption *DSPInt* est verrouillée jusqu'à ce que le DSP redevienne interruptible, moment où il servira la demande d'accès au PC.
- Si - ou lorsque - le DSP est interruptible, il passe à l'*adresse de branchement*. À ce stade, le code du DSP exécute la fonction requise.
    - S'il s'agit d'un transfert, la fonction doit lire le *code de contrôle* (adresse de boîte aux lettres 0x10F0400A), masquer les deux bits inférieurs qui représentent le type d'opération. Le résultat, appelé *USBTransferSize*, représente le nombre maximum d'octets qui doivent être transférés dans ce segment. Notez que le contenu du champ *ControlCode* de la boîte aux lettres ne doit pas être modifié.
    - Si nécessaire, la fonction transfère les octets requis vers ou depuis la zone de *données* de la boîte aux lettres. Le nombre d'octets à transférer est le plus petit entre le champ *NbBytes* de la boîte aux lettres et le résultat *USBTransferSize* qui vient d'être calculé.
    - Enfin, le nombre d'octets transférés doit être soustrait de la valeur de *NbBytes* et le champ *NbBytes* doit être mis à jour avec la nouvelle valeur dans la boîte aux lettres.
- Pour une *K_Read* ou une *K_Write*, le champ *TransferAddress* est incrémenté de manière à ce que le segment suivant soit transféré vers/depuis les adresses de mémoire postérieures au segment actuel. *TransferAddress* représente un nombre d'octets.
- Après l'exécution de la fonction demandée, le DSP active le signal HINT pour signaler au contrôleur USB que l'opération est terminée. Cette opération a été définie de manière pratique dans un

La macro *Accusé de réception* dans les codes d'exemple peut être insérée à la fin de n'importe quelle fonction utilisateur. Notez que, du point de vue du PC, la commande semble "suspendue" jusqu'à ce que l'accusé de réception soit émis par le DSP. Le code utilisateur ne doit pas prendre trop de temps avant d'émettre l'accusé de réception. Dans le cas d'une fonction qui ne retourne pas (*K_Exec* au point d'entrée d'une boucle sans fin par exemple), l'*accusé de réception* peut être émis au début de la fonction pour indiquer que la branche a été prise. L'*acquittement* est simplement le signal au contrôleur USB d'avancer à l'étape 3.

11.10.6.3.1.3 Étape 3

- Dans le cas d'une commande *K_Read*, le contrôleur USB lit tous les octets requis dans la boîte aux lettres et les renvoie au PC via le paquet USB actuel.
- Dans le cas d'un K_Exec ou d'un K_Write, le contrôleur USB envoie alors un petit *paquet* d'*achèvement de* 4 octets au PC pour signaler l'achèvement de la commande.

Comme l'accès au PC est demandé par l'intermédiaire de l'interruption DSPInt du HPI, il peut être obtenu même lorsque le code utilisateur est déjà en cours d'exécution. Il n'est donc pas nécessaire de revenir au noyau pour pouvoir lancer une nouvelle fonction DSP. De cette manière, les fonctions utilisateur peuvent être réintroduites. Habituellement, le noyau est utilisé au niveau 2 pour télécharger et lancer un programme principal d'utilisateur, qui peut ou non retourner au noyau à la fin. Pendant que ce programme tourne, le même processus décrit ci-dessus peut être utilisé au niveau 3 pour lire ou écrire des emplacements de mémoire DSP, ou pour forcer l'exécution d'autres fonctions DSP utilisateur, qui elles-mêmes peuvent ou non revenir au code utilisateur principal, et ainsi de suite... C'est entièrement la décision du développeur. Si une instruction de retour est utilisée à la fin de la fonction utilisateur (ou si la fonction est écrite en C, elle inclut implicitement un retour), l'exécution est renvoyée au code qui s'exécutait avant la demande. Ce code peut être le noyau au niveau 2, ou le code utilisateur au niveau 3.

L'acquittement de l'achèvement de la fonction DSP (code intrinsèque ou utilisateur) se fait par l'affirmation du signal HINT. Cette opération est encapsulée dans la macro d'*acquittement* du code d'exemple pour le bénéfice du développeur. Cette opération d'acquittement est effectuée dans le seul but de signaler à la commande hôte initiatrice que la fonction DSP demandée est terminée et que l'exécution peut reprendre du côté du PC. Normalement, pour une fonction utilisateur simple, qui comprend un retour (vers le code utilisateur principal ou vers le noyau), cet accusé de réception est placé à la fin de la fonction utilisateur (juste avant le retour) pour indiquer que la fonction est terminée. Pour des raisons de bonne programmation, le développeur ne doit pas mettre en œuvre des fonctions DSP qui prennent beaucoup de temps avant de renvoyer un accusé de réception. Dans le cas d'une fonction qui ne peut pas retourner au noyau ou au code utilisateur précédemment exécuté, ou dans tous les cas où l'utilisateur ne veut pas que la commande hôte qui a initié l'accès soit suspendue jusqu'à la fin de la fonction DSP, l'accusé de réception peut être placé au début de la fonction utilisateur. Dans ce cas, il signale uniquement que la branche a été prise. Un autre moyen de signaler l'achèvement de la fonction DSP doit alors être utilisé. Par exemple, le PC peut interroger un indicateur d'achèvement dans la mémoire du processeur.

Pendant l'exécution de la commande, le processeur n'est ininterruptible que pendant une très courte période (entre le déclenchement de l'interruption DSPInt et le passage au début de la fonction utilisateur ou intrinsèque). Par conséquent, l'exécution du code de commande sur le DSP ne bloque pas les tâches critiques qui pourraient être exécutées sous interruptions (gestion des E/S analogiques par exemple).

Le noyau comprend des fonctions permettant d'effectuer des transferts atomiques et non atomiques. Pour les transferts atomiques, le DSP est également ininterrompu pendant le transfert effectif de chaque bloc de 64 octets (ou de 512 octets pour une connexion USB à grande vitesse). Le temps de transfert réel dépend du périphérique cible, la RAM sur puce étant la plus rapide. Le fait de rendre le transfert ininterrompu présente l'avantage que, du point de vue du DSP, toutes les parties du bloc sont transférées simultanément. Pour les transferts non atomiques, différentes parties du bloc peuvent être transférées à des moments différents du point de vue du DSP. Cela signifie par exemple que la partie basse d'un mot peut être transférée lors d'un cycle de l'unité centrale, tandis que la partie haute est transférée lors d'un cycle différent. Les transferts non atomiques présentent l'avantage que les tâches critiques du DSP peuvent interrompre le transfert lui-même et donc avoir la priorité sur le transfert USB. Les transferts non atomiques

sont utiles dans les situations où des délais extrêmement rapides doivent être assurés sur le DSP et où même le temps de transfert très court est suffisant pour perturber ces tâches critiques.

11.10.6.3.2 Utilisation des requêtes K_Read et K_Write pour les fonctions utilisateur

En principe, les commandes K_Read et K_Write ne sont utilisées que pour invoquer les fonctions intrinsèques du noyau. Cependant, rien n'empêche le développeur d'utiliser ces commandes pour invoquer une fonction utilisateur. Cela peut être utile pour mettre en œuvre des fonctions utilisateur qui doivent recevoir ou envoyer des données depuis/vers le PC, car cela leur donne un moyen d'utiliser efficacement la boîte aux lettres et le processus de transfert du contrôleur USB embarqué. Pour ce faire, la fonction utilisateur doit se comporter exactement de la même manière que les fonctions *intrinsèques* pour les transferts de lecture et d'écriture. Le champ *BranchAddress* de la boîte aux lettres doit contenir le point d'entrée d'une fonction utilisateur, plutôt que l'adresse d'une fonction intrinsèque du noyau.

Il convient de noter que les arguments, les données et les paramètres peuvent alternativement être transmis du PC aux structures statiques du DSP par des commandes *K_Read* ou *K_Write* régulières (du noyau) après et/ou avant l'invocation d'une fonction utilisateur. Il s'agit là d'un autre moyen, moins efficace mais plus conventionnel, de transférer des arguments vers/depuis des fonctions DSP.

12 Code de support DSP

12.1 Code de support pour le pilote et la programmation Flash

Deux niveaux de code de support Flash sont fournis aux développeurs :

- Une bibliothèque de pilotes DSP pour les développeurs qui souhaitent inclure des fonctions de programmation Flash dans leur code DSP. Ce pilote est décrit ci-dessous.
- Programmation Flash Le code DSP est fourni pour prendre en charge la fonctionnalité de programmation Flash qui fait partie des bibliothèques d'interface (LabVIEW et C/C++), ainsi que l'interface du mini-débogueur. Ce code n'est pas décrit ci-dessous. Il est fourni sous la forme d'un fichier exécutable nommé *SR3_Flash_Support.out*. Ce code DSP est chargé et exécuté par les fonctions d'interface qui requièrent sa présence. Ce code est basé sur le pilote Flash décrit ci-dessous.

12.1.1 Aperçu du pilote flash

Un pilote DSP est fourni pour aider l'utilisateur à développer un code DSP qui inclut des fonctions de programmation Flash. Ce pilote se présente sous la forme d'une bibliothèque nommée *SR3_FB_Driver.lib*.

Le pilote se trouve dans le répertoire *C:\NProgram Files\SR3_Applications\NDSP_Code*.

Le pilote est composé de fonctions pouvant être appelées en C, ainsi que de structures de

données appropriées. Les fonctions permettent la lecture, l'effacement et l'écriture séquentielle de la

mémoire Flash.

Le pilote utilise une mémoire tampon FIFO d'écriture logicielle, de sorte que les fonctions d'écriture ne doivent pas attendre la fin de chaque opération d'écriture.

Les fonctions de lecture sont exécutées de manière asynchrone et sont très rapides. Les écritures sont séquentielles et sont effectuées sous l'interruption GPIO_0 (liée à l'interruption INT6 du DSP). Le temps d'écriture typique est de 60 µs par mot. L'effacement est asynchrone et peut être assez long (typ 0,5 s/secteur, max 3,5 s par secteur). Les fonctions d'effacement attendent que toutes les écritures soient terminées avant de commencer. Elles sont bloquantes, ce qui signifie que l'exécution est bloquée dans la fonction d'effacement tant que l'effacement n'est pas terminé.

Les adresses de lecture, d'écriture et d'effacement sont de 32 bits.

Remarque : toutes les adresses transmises au pilote et en provenance de celui-ci sont des adresses d'octets. Cela est vrai, même si la mémoire Flash est composée de mots de 16 bits seulement. Au niveau le plus bas, toutes les opérations sont effectuées et comptées en mots de 16 bits. Le nombre d'opérations à effectuer (lecture, écriture et effacement) est spécifié au pilote en nombre de mots de 16 bits. Néanmoins, toutes les adresses sont des adresses d'octets.

Les lectures sont très simples. Elles sont effectuées de manière asynchrone à l'aide de la fonction *SR3FB_Read*. Cette fonction renvoie le contenu d'une adresse 32 bits.

Les écritures sont effectuées séquentiellement à l'aide de la fonction *SR3FB_Write*. Les écritures sont effectuées aux adresses définies dans le registre *FB_WriteAddress.* Ce registre n'est pas accessible à l'utilisateur. Il doit être initialisé avant la première écriture d'une séquence et est automatiquement incrémenté après chaque écriture. Le registre *FB_WriteAddress* peut être initialisé à l'aide de la fonction *SR3FB_SetAddress* ou de la fonction *SR3FB_WritePrepare*.

Une opération d'écriture peut transformer des uns en zéros, mais ne peut pas transformer des zéros en uns. Normalement, un secteur de la mémoire flash doit être effacé avant toute tentative d'écriture dans ce secteur. Cependant, la flash peut être programmée plusieurs fois, en transformant à chaque fois certains des "1" restants en "0".

La fonction *SR3FB_WritePrepare* efface au préalable tous les secteurs commençant à l'adresse-octet spécifiée et contenant au moins le nombre séquentiel d'octets spécifié. Comme l'effacement est effectué secteur par secteur, cette fonction peut effacer plus d'octets que ce qui est réellement spécifié à la fonction. La fonction initialise ensuite le registre *FB_WriteAddress* à l'adresse de début spécifiée, de sorte que la prochaine écriture soit effectuée au début du segment de mémoire spécifié.

La fonction *SR3FB_Write* n'attend pas que l'écriture soit terminée. Elle place simplement le mot de 16 bits à écrire dans le tampon *FB_WriteFIFO* et retourne. Les écritures sont en fait effectuées sous contrôle d'interruption, sans intervention du code utilisateur.

L'état de remplissage de la FIFO d'écriture, ainsi que l'état des erreurs d'écriture et d'effacement peuvent être surveillés à l'aide de la fonction
Fonction *SR3FB_FIFOState*.

12.1.2 Ressources utilisées

Les opérations d'écriture utilisent INT6 déclenché par l'interruption GPIO_0. Aucune de ces ressources ne doit être utilisée à d'autres fins dans le code utilisateur.

Le code utilisateur doit initialiser le vecteur d'interruption INT6 à l'étiquette *_SR3FBINT*. Ce vecteur d'interruption réside à l'adresse 0x10E080C0. Voir ci-dessous un exemple de code.

```
        .global _SR3FBINT
        .sect .vectors
        .nocmp

_ISRINT6 :

        STW     .D2T2   B10,*B15--[2]
    || MVKL  .S2  _SR3FBINT,B10 MVKH
        .S2 _SR3FBINT,B10 B  .S2 B10
        LDW     .D2T2 *++B15[2],B10
        NOP     4
        NOP
        NOP

        .fin
```

12.1.3 Installation du pilote

Note : Ce pilote nécessite l'environnement C pour fonctionner correctement. Cela signifie que les fonctions du pilote ne doivent être appelées qu'à partir d'un code écrit en C, ou à partir d'un code assembleur qui a configuré l'environnement C avant d'appeler les fonctions (voir la documentation de Texas Instruments pour plus de détails).

- Toutes les fonctions du pilote définies ci-dessous sont contenues dans la bibliothèque *SR3_FB_Driver.lib.* Le code utilisateur doit être lié à cette bibliothèque pour fonctionner correctement (la bibliothèque doit être ajoutée aux fichiers source du projet).
- Lorsque vous liez la bibliothèque à un projet C, ce dernier doit utiliser le modèle de mémoire *Far.* Cela est nécessaire pour pouvoir accéder à tous les secteurs de la flash, qui s'étendent sur plusieurs pages de 32k (le mode d'adressage direct utilisant B14 ou B15 n'accepte que des décalages de 32k octets).
- Étant donné que les écritures sont effectuées sous l'interruption INT6 (GPIO_0), le vecteur d'interruption INT6 doit être initialisé à l'étiquette *_SR3FBINT* et doit être lié à l'adresse 0x10E080C0. Ce label est le point d'entrée de la routine d'interruption INT6. La routine d'interruption INT6 est définie dans la bibliothèque *SR3_FB_Driver.lib.* Nous suggérons d'utiliser le fichier *vectors.asm* fourni dans le dossier *C:\Program Files\SR3_Applications\DSP_Code* lors de l'utilisation de la bibliothèque *SR3_FB_Driver.lib.* Le fichier de commande de l'éditeur de liens fourni dans le dossier du pilote flash peut également être utilisé pour assurer le lien correct pour le vecteur INT6.
- Un fichier d'en-tête nommé *SR3_FB_Driver.h* est fourni et déclare toutes les fonctions du pilote.
- Avant d'appeler toute autre fonction du pilote, celui-ci doit être initialisé à l'aide de la fonction *SR3FB_Init* fonction.

Tous les fichiers décrits ci-dessus se trouvent dans l'exemple de code DSP *SR3_Flash_Support.*

12.1.4 Structures de données

12.1.4.1 FB_WriteFIFO

FB_WriteFIFO est un tampon de 32 mots de 16 bits auquel on accède par la logique d'accès FIFO. La FIFO elle-même n'est pas accessible à l'utilisateur. Elle ne peut être écrite qu'en utilisant la fonction *SR2FB_Write*. Elle n'est vidée que par la routine de service d'interruption INT6.

12.1.4.2 FB_WriteAddress

FB_WriteAddress est une variable non signée de 32 bits qui contient toujours l'adresse du prochain mot de 16 bits à écrire. *FB_WriteAddress* peut être initialisé par la fonction *SR3FB_SetAddress()* ou la fonction *SR3FB_WritePrepare()*. Après chaque écriture, le registre *FB_WriteAddress* est automatiquement incrémenté. Cette incrémentation se produit dans la routine d'interruption INT6. Par conséquent, pour le code utilisateur, la valeur indique toujours l'adresse de la prochaine écriture, jamais la valeur de l'écriture en cours.

La valeur actuelle du registre *FB_WriteAddress* peut être lue avec la fonction *SR3FB_FIFOState*.

12.1.4.3 FB_WriteEraseError (erreur d'écriture)

FB_WriteEraseError est une variable de 16 bits qui contient divers bits d'état d'erreur. Elle est retournée par plusieurs fonctions, dont *SR3FB_SetAddress*(), *SR3FB_FIFOState*(), *SR3FB_WritePrepare*() et *SR3FB_Write*().

Une fois qu'un bit d'erreur est mis à un, indiquant une erreur, il reste à un jusqu'à ce que le mot d'erreur soit effacé à l'aide de
SR3FB_ErrorClear(). L'exécution de la fonction *SR3FB_Init*() efface également le registre d'état des erreurs.

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
											SE			WP	WE

WE (*Write Error)* **:** Une erreur s'est produite lors d'une écriture. Soit une écriture a été tentée à une adresse qui n'avait pas été effacée auparavant, soit à une adresse en dehors de la plage d'adresses utilisables, soit la mémoire flash ROM ne fonctionne pas correctement.

WP (*Write in Progress*) **:** Lorsqu'il vaut un, ce bit indique que des écritures sont en cours. Ce bit n'est mis à 0 que lorsque la FIFO d'écriture est vide et que la dernière opération d'écriture est terminée. Il est mis à un dès qu'un nouveau mot est écrit dans la FIFO d'écriture.

SE Secteur (Erreur d'effacement) **:** Ce bit indique qu'une erreur s'est produite pendant l'opération d'effacement du secteur demandé. Soit un effacement a été tenté à une adresse en dehors de la plage d'adresses utilisables, soit la Flash ne fonctionne pas correctement.

Remarque : lorsqu'une écriture est tentée à une adresse qui n'a pas été effacée précédemment et que les données écrites impliquent le retour d'un ou de plusieurs bits à un, le comportement habituel est que le processus se bloque indéfiniment. Le bit WP reste indéfiniment à un. Il n'y a pas de délai pour déverrouiller le processus d'écriture. Les prochaines écritures dans la FIFO d'écriture peuvent être acceptées, mais la FIFO n'est pas vidée, donc à un moment donné la FIFO est pleine et la fonction SR3FB_Write se bloque.

12.1.5 Fonctions de l'utilisateur

12.1.5.1 unsigned int SR3FB_Init()

Initialise le pilote et réinitialise la mémoire Flash. Elle détecte la Flash ROM et renvoie la taille de la mémoire en octets, ou zéro si le circuit n'est pas présent. Cette fonction doit être appelée au moins une fois avant l'appel de toute autre fonction du pilote. Cette fonction peut être appelée pour réinitialiser le pilote. Notez que l'interruption GPIO_0 est activée et que INT6 du DSP est utilisé.

Entrée :

pas d'entrée

Sortie :

pas de sortie

Retourner :

Taille de la ROM flash en octets

12.1.5.2 unsigned short SR3FB_SetAddress(unsigned int FB_WAddress)

Cette fonction attend que toutes les écritures en attente soient terminées. Ensuite, elle place le pointeur *FB_WriteAddress* à la valeur de 32 bits passée en argument. La fonction ne vérifie pas que l'adresse passée en argument se trouve dans la plage d'adresses autorisée. Si ce n'est pas le cas, les écritures suivantes échoueront. La fonction renvoie le statut *FB_WriteEraseError* actuel.

Entrée :

unsigned int FB_WAddress C'est l'adresse de l'octet de 32 bits pour la prochaine écriture.

Sortie :

pas de sortie

Retourner :

L'erreur FB_WriteEraseError actuelle

12.1.5.3 unsigned short SR3FB_FIFOState(unsigned short *FB_FIFOCount, unsigned int *FB_WAddress)

Cette fonction renvoie le nombre de mots de 16 bits encore présents dans la FIFO d'écriture dans l'argument *FB_FIFOCount*, et la valeur actuelle du registre *FB_WriteAddress* dans l'argument *FB_Waddress.* La fonction renvoie l'état actuel de *FB_WriteEraseError.*

Remarque : une valeur de retour de zéro pour FB_FIFOCount ne signifie pas que toutes les écritures sont terminées. La dernière écriture peut encore être en cours. Pour vérifier que toutes les écritures ont bien été effectuées, il convient de vérifier le bit WP dans le registre d'état FB_WriteEraseError.

Entrée :

unsigned short *FB_FIFOCount : pointeur sur une variable pour la sortie FIFOCount unsigned int*

WAddress : pointeur sur une variable 32 bits FB_WAddress sortie **:**

unsigned short FB_FIFOCount

unsigned int FB_WAddress

Retour :

L'erreur FB_WriteEraseError actuelle

12.1.5.4 void SR3FB_ErrorClear()

Cette fonction efface le registre d'état *FB_WriteEraseError* actuel.

Entrée :

pas d'entrée

Sortie :

pas de sortie

Retourner :

pas de retour

12.1.5.5 short SR3FB_Read(unsigned int FB_RAddress)

La fonction renvoie le mot de 16 bits lu à l'adresse *FB_RAddress*. Notez qu'aucune vérification n'est effectuée pour s'assurer que la lecture a lieu dans la section occupée par la Flash ROM. Si une lecture est tentée dans la plage d'adresses de la RAM sur puce, la fonction renvoie simplement le contenu de la RAM sur puce plutôt que le contenu de la Flash.

Entrée :

unsigned int FB_Radress : Il s'agit de l'adresse de l'octet lu sur 32 bits.

Sortie :

pas de sortie

Retourner :

Le mot de 16 bits lu à l'adresse de l'octet spécifié

12.1.5.6 unsigned short SR3FB_WritePrepare(unsigned int FB_WAddress, unsigned int FB_WSize)

La fonction préérise tous les secteurs du circuit Flash, nécessaires pour écrire un segment *FB_WSize* long, à partir de l'adresse *FB_WAddress*. Elle initialise ensuite le registre *FB_WriteAddress* à la valeur de *FB_WAddress*, de sorte que le prochain appel à *FB_Write* écrira effectivement au début du segment préparé.

Comme l'effacement est effectué secteur par secteur uniquement, cette fonction peut effacer plus de mots de 16 bits que ce qui est réellement spécifié. C'est le cas si *FB_Waddress* n'est pas une adresse correspondant au début d'un secteur, ou si *FB_Waddress*+ *FB_WSize* -1 n'est pas une adresse correspondant à la fin d'un secteur. Les secteurs ont une longueur de 131 072 (0x20000) octets et la première adresse flash est 0x42000000.

La fonction attend que toutes les écritures en attente soient terminées avant de commencer l'effacement.

La fonction ne vérifie pas que l'effacement n'inclut pas d'adresses en dehors de la plage d'adresses utilisables.

Si, au cours de la préparation, un effacement de secteur est tenté en dehors de la plage d'adresses utilisables, la fonction échoue simplement. Elle active le bit SE de l'état *FB_WriteEraseError* et retourne.

La fonction renvoie l'état actuel de *FB_WriteEraseError.* La fonction ne revient pas tant que l'effacement n'est pas terminé. Le temps dépend de la longueur du segment à préparer. L'effacement prend généralement 0,5 seconde par secteur.

Note : Le comportement de la fonction est indéfini si la taille FB_WSize demandée est nulle : *Le comportement de la fonction est indéfini si la taille FB_WSize demandée est nulle.*

Entrée :

unsigned int FB_Waddress : Adresse de départ en octets du segment à préparer unsigned

int FB_Wsize : Taille en octets du segment à préparer

Sortie :

pas de sortie

Retourner :

L'*erreur FB_WriteEraseError* actuelle

12.1.5.7 unsigned short SR3FB_Write(short Data)

La fonction place la valeur de *Data* dans la FIFO d'écriture. Elle retourne normalement sans attendre la fin de l'écriture. Les écritures sont effectuées sous des interruptions INT6. Toutefois, si la FIFO est pleine lorsque la fonction est appelée, la fonction attend qu'un emplacement soit disponible dans la FIFO avant de placer la valeur suivante dans la FIFO et de revenir.

La programmation prend généralement 60 secondes par mot de 16 bits. Par conséquent, si la fonction est appelée alors que la FIFO est pleine, il se peut qu'elle ne soit pas renvoyée avant que 60 secondes se soient écoulées.

L'écriture demandée commence dès que les écritures précédentes dans la FIFO sont terminées. Les données sont écrites à la valeur actuelle de *FB_WriteAddress*. La fonction ne vérifie pas que l'écriture est tentée à l'intérieur de la plage d'adresses utilisables. Si ce n'est pas le cas, l'écriture échoue tout simplement. L'échec ne sera pas détecté tant que les données n'auront pas été écrites de la FIFO vers la mémoire Flash.

La fonction renvoie l'état actuel *FB_WriteEraseError.* Toutefois, ce mot d'erreur ne reflète pas l'état de l'écriture demandée, car la fonction n'attend pas que cette écriture commence réellement.

Entrée :

short Data : il s'agit des données de 16 bits à placer dans la FIFO.

Sortie :

pas de sortie

Retourner :

L'*erreur FB_WriteEraseError* actuelle

12.2 Pilote CODEC et exemple de code

12.2.1 Vue d'ensemble

Un pilote pour les E/S analogiques (CODEC) est fourni, ainsi que le code DSP d'une application de démonstration qui utilise ce pilote, ainsi qu'un projet "shell" vide. Le code source du pilote CODEC se trouve dans le dossier *SR3_AICDriver*. Le code source de l'application de démonstration se trouve dans le dossier *SR3_SignalTracker*. Ce dossier contient le code DSP de l'application de démonstration *SignalTracker* discutée dans la section. Le code source du projet shell se trouve dans le dossier *SR3_IO_Shell*. Le projet shell constitue un excellent point de départ pour le développement du code DSP qui utilise le CODEC.

Le pilote a été optimisé en langage assembleur et C, mais peut être utilisé soit en C, soit en langage assembleur. Il se présente sous la forme d'une bibliothèque d'objets DSP *AICDriverSr3Jr.lib.* Le pilote contient des fonctions appelables en C pour configurer et utiliser le CODEC. Une fonction appelée *dataprocess()* est fournie en C, où les développeurs peuvent commodément placer leur propre code de traitement des E/S analogiques.

12.2.2 Ressources utilisées

Le pilote CODEC utilise les ressources matérielles suivantes. Aucune de ces ressources ne doit être utilisée à d'autres fins dans le code utilisateur.

- Le McBSP0 est utilisé par le pilote.
- Les canaux EDMA 2 et 3 sont utilisés par le pilote. Les paramètres de configuration (*PaRAM*) des canaux EDMA 2,3,66,67,68 et 69 sont utilisés par le pilote.
- L'interruption de la région 0 de l'EDMA est utilisée par le pilote et liée au vecteur d'interruption INT5 du DSP. Remarque : si l'interruption globale de l'EDMA est utilisée par le code utilisateur, le registre d'interruption en attente (EDMA_IPR) doit être vérifié pour éviter de gérer une interruption déclenchée par le canal 2 ou 3 de l'EDMA.
- Il convient d'être prudent lors de l'utilisation d'un autre DMA ayant une priorité supérieure ou égale à celle des canaux utilisés par le pilote. Les autres DMA doivent utiliser une priorité strictement inférieure à 0, sinon les DMA concurrents peuvent ralentir les DMA utilisés par le pilote au point de lui faire perdre des échantillons.
- Le GPIO_1 est utilisé pour gérer la réinitialisation du CODEC.
- Le port I2C du DSP est utilisé pour configurer le CODEC.
- Le pilote utilise 3040 octets de code et 225 octets de données.
- La sortie DSP *CLKOUT0* et le diviseur d'oscillateur (*PLL1_OSCDIV1*) sont utilisés pour générer l'horloge principale du McBSP0 et du CODEC.

Il convient d'être prudent lors de l'utilisation d'un autre DMA ayant une priorité supérieure ou égale à celle des canaux utilisés par le pilote. Les autres DMA doivent utiliser une priorité strictement inférieure à 0. Sinon, les DMA concurrents peuvent ralentir les DMA utilisés par le pilote au point de lui faire perdre des échantillons.

12.2.3 Restrictions

Les restrictions suivantes s'appliquent au développement de code C ou de code assembleur à l'aide du pilote CODEC :

- La fonction de traitement des E/S définie par l'utilisateur, *dataprocess(),* doit être présente dans le code de l'utilisateur. *Dataprocess()* est une fonction normale et aucune protection de registre n'est nécessaire puisque le pilote CODEC enregistre déjà l'ensemble du contexte. Par défaut, *dataprocess()* n'est pas interruptible, mais le bit d'activation d'interruption globale (bit GEI du registre CSR) peut être mis à 1 pour rendre la fonction interruptible. Lors du développement en assembleur, le symbole *_dataprocess* doit être défini à l'aide de la directive *.global.*
- Tous les symboles et étiquettes accessibles en C définis dans la bibliothèque *SR3_AICDriver.lib* doivent avoir un préfixe "_" lorsqu'ils sont utilisés en langage assembleur.
- L'interruption INT5 du DSP est utilisée par le pilote pour appeler la routine de service d'interruption du pilote et enfin la *fonction dataprocess()*. Le vecteur de cette interruption doit être correctement déclaré et lié à l'adresse 0x10E080A0. L'étiquette *_SR3AIC* pour l'ISR du pilote doit apparaître explicitement dans le vecteur pour l'interruption DSP INT5. Voir le code DSP de la démo *SignalTracker* pour un exemple de *vecteurs.asm* corrects.

12.2.4 Variables et fonctions accessibles à l'utilisateur

La bibliothèque *SR3_AICDriver.lib* définit et alloue les variables suivantes accessibles à l'utilisateur (utilisez le fichier *SR3_AICDriver.h* pour accéder à ces variables dans votre code DSP) :

int Div_osc ;

Cet entier de 32 bits sélectionne la fréquence *CLKS* pour le McBSP0. Cette valeur, ainsi que *DSM_SSM,* ajuster d'échantillonnage d'échantillonnage d'échantillonnage. Vous pouvez pouvez utiliser l'outil LabVIEW VI *SR3_Junior_DetermineRegister_CS42436.vi* pour générer toutes les variables de configuration du CODEC, notamment *Div_osc*.

int DSM_SSM ;

Cet entier de 32 bits sélectionne le mode DSM ou SSM. Zéro sélectionne le mode DSM et 1 sélectionne le mode SSM.

unsigned char AICReg[17] ;

Ce vecteur de 8 bits contient les registres CODEC 5 à 13 et 16 à 23. Vous pouvez utiliser le LabVIEW VI *SR3_Junior_DetermineRegister_CS42436.vi* pour générer toutes les variables de configuration du CODEC, y compris ce vecteur. Consultez la fiche technique du CS42436 de Cirrus-Logic pour obtenir plus de détails sur les fonctions de registre. En outre, l'application *SR3_SignalTracker* est un bon point de départ pour se familiariser avec le CODEC.

int IOBuf[12] ;

Ce vecteur entier de 32 bits est conçu pour contenir les échantillons d'entrée et de sortie vers/depuis le CODEC. Les 6 premiers éléments du vecteur sont les échantillons d'entrée et les 6 éléments suivants sont les échantillons de sortie. Le code utilisateur lit et écrit les échantillons dans le vecteur *IOBuf* à chaque appel de la fonction *dataprocess()*. Le code DSP de l'utilisateur dispose d'une période d'échantillonnage complète pour exécuter la fonction *dataprocess().* Si la fonction n'est pas terminée au cours d'une période d'échantillonnage, les échantillons d'entrée sont remplacés par les nouveaux échantillons et les mêmes échantillons de sortie sont envoyés au CODEC. Les échantillons sont justifiés à gauche.

unsigned char i2c_data[2] ;

Ce vecteur de 8 bits à 2 éléments est utilisé pour transmettre des paramètres aux fonctions *User_I2C_ReadReg()* et *User_I2C_WriteReg*(). Ces deux fonctions peuvent être appelées par le code utilisateur du PC ou du DSP pour lire ou écrire les registres CODEC. Voir ci-dessous pour plus de détails sur ces fonctions et la façon dont le vecteur i2c_data est utilisé lors de l'appel de ces fonctions.

void startAIC()

Cette fonction configure le CODEC et lance le processus de conversion du CODEC. Elle n'a pas d'arguments. Elle utilise les valeurs de configuration des registres trouvées dans les variables de configuration et configure le CODEC en conséquence. Ces valeurs doivent être initialisées avant d'appeler *startAIC*. Elles définissent les paramètres du CODEC tels que la fréquence d'échantillonnage, le gain d'entrée et de sortie, etc. Après l'exécution de cette fonction, la fonction de traitement définie par l'utilisateur*, dataprocess(),* commence à être déclenchée à chaque période d'échantillonnage. La bibliothèque d'interface CODEC LabVIEW comprend un VI qui génère automatiquement le contenu des registres.

void stopAIC()

Cette fonction arrête le processus de conversion du CODEC. Après l'exécution de cette fonction, la fonction *dataprocess()* définie par l'utilisateur cesse d'être déclenchée et le CODEC émet en continu les dernières valeurs d'échantillonnage.

void User_I2C_ReadReg ()

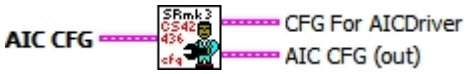
Cette fonction lance une séquence de lecture d'un octet sur le port de contrôle du CODEC. Avant d'appeler la fonction *User_I2C_ReadReg()*, le code du PC ou du DSP doit initialiser *i2c_data[0]* avec le numéro de registre à lire. La fonction *User_I2C_ReadReg*() place le contenu du registre dans *i2c_data[1]*.

void User_I2C_WriteReg()

Cette fonction lance une séquence d'écriture d'un registre sur le port de contrôle du CODEC. Avant d'appeler *User_I2C_WriteReg*(), le code du PC ou du DSP doit initialiser *i2c_data[0]* avec le numéro du registre à écrire et *i2c_data[1]* avec le contenu du registre.

12.2.5 Support LabVIEW VI

L'interface VI LabVIEW SR3_DetermineRegister_CS42436 est fournie pour faciliter le réglage des registres CODEC et des variables de configuration. Le cluster de sortie de ce VI contient les valeurs des variables de pilotage du CODEC *Div_osc*, *DSM_SSM* et *AICReg[17].*



12.2.5.1 Contrôles et indicateurs

12.2.5.1.1 AIC CFG

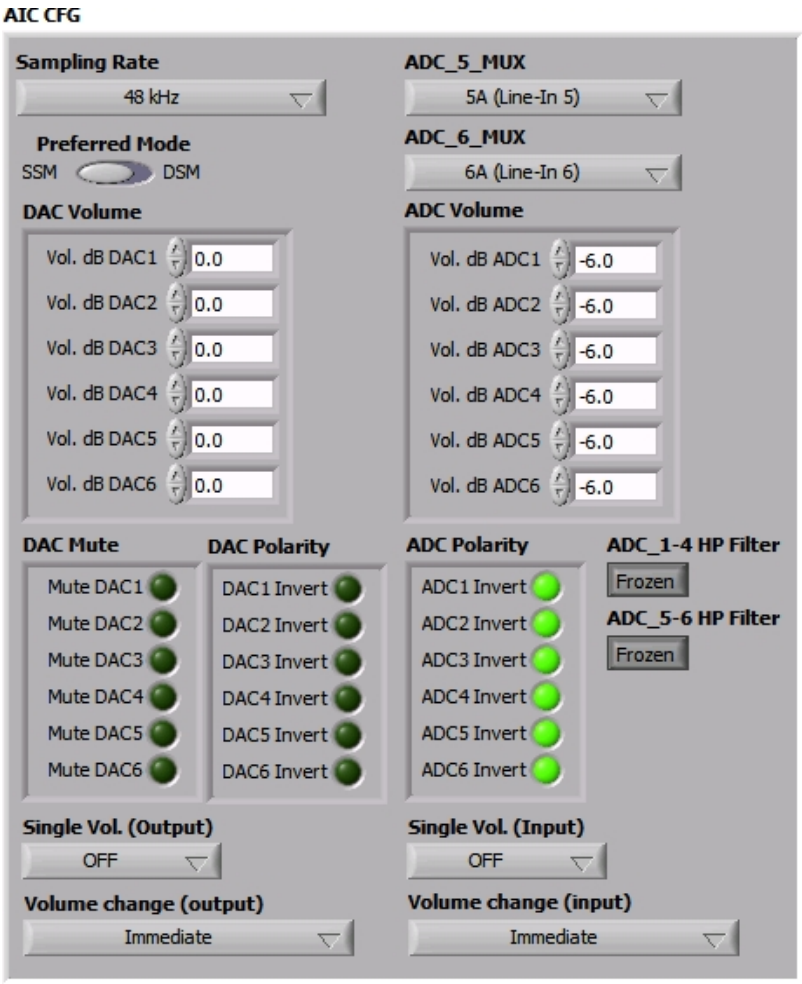


Figure 15 **Commandes de *configuration de l'AIC***

12.2.5.1.2 Taux d'échantillonnage

Cette commande permet de sélectionner le taux d'échantillonnage. La fréquence d'échantillonnage peut être choisie parmi un ensemble de valeurs comprises entre 4 kHz et 96 kHz. Notez que la commande n'expose que les fréquences d'échantillonnage les plus fréquentes, les autres étant possibles. Le *mode préféré* est utilisé chaque fois que la fréquence d'échantillonnage choisie le permet. Certains choix de fréquence d'échantillonnage ne sont compatibles qu'avec le mode DSM.

12.2.5.1.3 Mode préféré

Cette commande indique si le mode d'échantillonnage doit être *DSM* (Double-Rate Sampling) ou *SSM* (Single-Rate Sampling). Certaines fréquences d'échantillonnage ne permettent que le mode *DSM*. Dans ce cas, le mode préféré est ignoré. Au niveau inférieur, cette commande agit sur les variables de pilotage *Div_osc* et *DSM_SSM.*

12.2.5.1.4 Volume du DAC

Ce groupe contient le volume de chaque sortie. Le volume peut être réglé de -127,5 dB à 0 dB en
Pas de 0,5 dB.

12.2.5.1.5 Volume de l'ADC

Ce groupe contient le volume de chaque entrée. Le volume peut être réglé de -64,0 dB à +24,0 dB par pas de 0,5 dB.

12.2.5.1.6 DAC Mute

Cette commande permet de couper le son de chaque sortie individuellement.

12.2.5.1.7 Polarité du DAC

Cette commande permet de sélectionner la polarité de la sortie.

12.2.5.1.8 Polarité de l'ADC

Cette commande permet de sélectionner la polarité de l'entrée.

12.2.5.1.9 ADC_x-y Filtre HP

Cette commande active les filtres passe-haut du CAN ou les fige sur la dernière valeur. Le fait d'engager brièvement le filtre et de le geler annule effectivement le décalage en courant continu présent sur l'entrée du CAN. Cette opération s'effectue au niveau du CAN, tandis que la compensation du décalage s'effectue au niveau du logiciel.

12.2.5.1.10 AIN5_MUX et AIN6_MUX

Ces commandes permettent de sélectionner le chemin d'entrée pour les entrées analogiques 5 et 6, soit Line-In, soit Electret-Microphone Input.

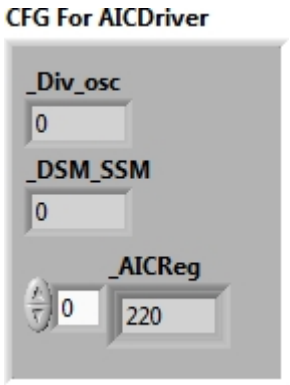


Figure 16 : Registres de configuration du CODEC Registres de configuration du CODEC.

Cet indicateur contient les variables de configuration du pilote. Avant de démarrer le CODEC avec la fonction *startAIC()* du pilote, le PC ou le code DSP doit charger les variables du pilote *_Div_osc*, *_DSM_SSM* et *AICReg*.

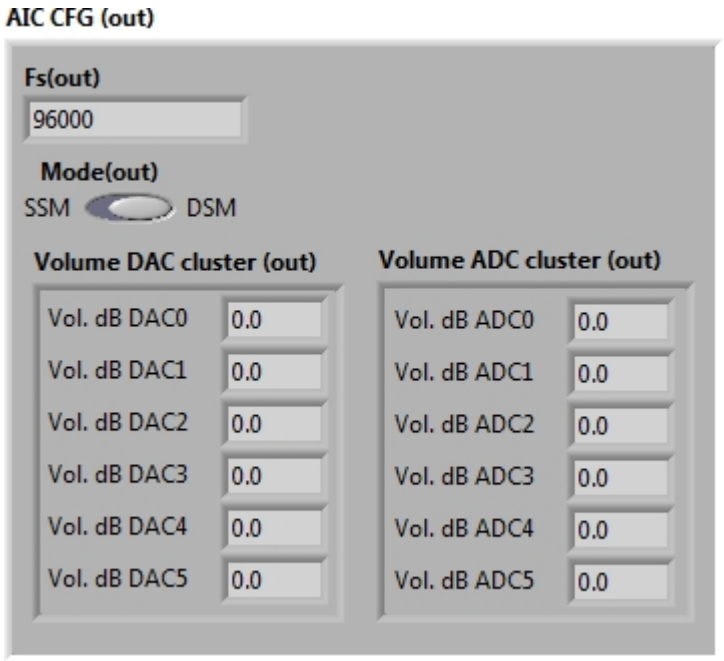


Figure 17 : Configuration réelle.Configuration réelle.

Cet indicateur contient la configuration réelle. L'utilisateur peut consulter cet indicateur pour voir si le mode d'échantillonnage réel est le mode *DSM* ou *SSM* préféré a été accepté. Les volumes réels de sortie et d'entrée sont également présentés dans cet indicateur.