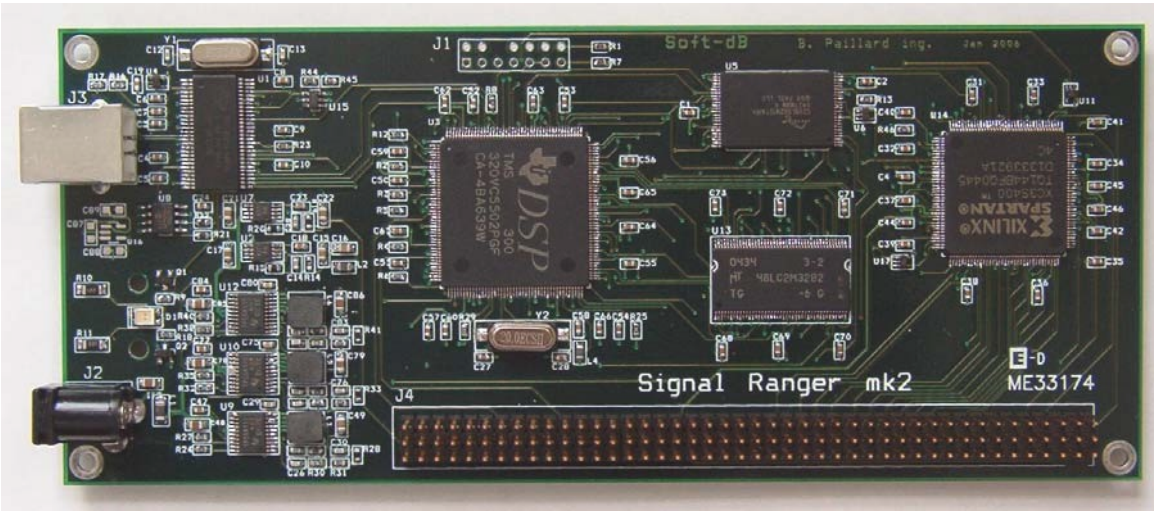
Carte DSP Signal Ranger_mk2

Manuel de l'utilisateur



Bruno Paillard

Rev 03 28 février 2006

Caractéristiques principales

Signal_Ranger_mk2 est une carte DSP à point fixe comprenant un DSP TMS320C5502 de 300 MHz, un FPGA SPARTAN 3 de 400 kgates et une interface USB 2 à grande vitesse, permettant des communications rapides avec la carte. Le pilote 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.
Grâce à ses ressources très flexibles (DSP+FPGA) et au fait qu'elle peut fonctionner comme une carte autonome, la carte *Signal_Ranger_mk2* peut être utilisée dans de nombreuses applications. Avec l'ajout de cartes filles analogiques, *Signal_Ranger_mk2* couvre aisément les applications suivantes :

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

Architecture et modes de démarrage

La carte *Signal Ranger_mk2* comprend une ROM Flash de 1M de mots, à partir de laquelle le DSP peut démarrer. En outre, la Flash peut également contenir le code de configuration du FPGA, permettant au DSP d'initialiser le FPGA avec sa logique dans le cadre du processus de démarrage initial.
Le DSP peut démarrer de deux manières :

- **Démarrage autonome :** Lors de la mise sous tension, le contrôleur USB en configuration autonome prend le contrôle du DSP et du FPGA. Il charge un noyau de communication dans la RAM du DSP et l'exécute. Ce noyau recherche ensuite dans la mémoire Flash un fichier de description logique du FPGA. S'il le trouve, il charge le FPGA. Il recherche ensuite dans la mémoire flash le code DSP. S'il le trouve, il le charge dans la RAM et l'exécute. En préprogrammant la mémoire Flash avec la logique FPGA et/ou 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 :** Une fois la carte connectée à un PC, ce dernier peut forcer le DSP à redémarrer. Dans ce mode, le PC peut forcer le rechargement de la nouvelle logique FPGA et du code DSP. Ce mode peut être utilisé pour "prendre le contrôle" du DSP à tout . En particulier, il peut utilisé pour reprogrammer la mémoire Flash de manière totalement transparente, sans utiliser de cavaliers.

Même lorsque la carte DSP a démarré en mode autonome, un PC peut être connecté à tout pour lire/écrire la mémoire du DSP sans interrompre le code DSP déjà en cours d'exécution. Ces fonctions peuvent être utilisées pour fournir une visibilité en temps réel ou pour envoyer des commandes au code DSP intégré en cours d'exécution.

Données techniques :

- **USB**

Connexion PC USB 2.0. Débit moyen : 18 Mb/s (lecture), 22 Mb/s (écriture). Le contrôleur USB autonome ne nécessite aucune gestion de la part du logiciel DSP.

- **DSP**

TMS320C5502 DSP 16 bits à virgule fixe, fonctionnant à 300 MHz, avec 32 mots de RAM sur la puce.

- **FPGA**

FPGA XC3S400. 400 000 portes. 56 kbits de RAM distribuée, 288 kbits de RAM en bloc, 16 multiplicateurs 18x18 dédiés, 4 DCM. Fournit 63 E/S configurables par l'utilisateur.

- **Alimentation électrique**

Signal Ranger_mk2 est auto-alimenté par un bloc d'alimentation externe de 5V (+-5%). Il peut fonctionner sans connexion à un PC.

- **Mémoire**
  - 64 koctets de RAM à double accès sur la puce (DSP), répartis entre les espaces de données et de programme.
  - 4 Mbytes de RAM dynamique synchrone externe à 75 MHz, mappés dans l'espace de données et l'espace de programme.
  - Flash Rom externe de 2 Mbytes, mappé dans l'espace de données et de programme.

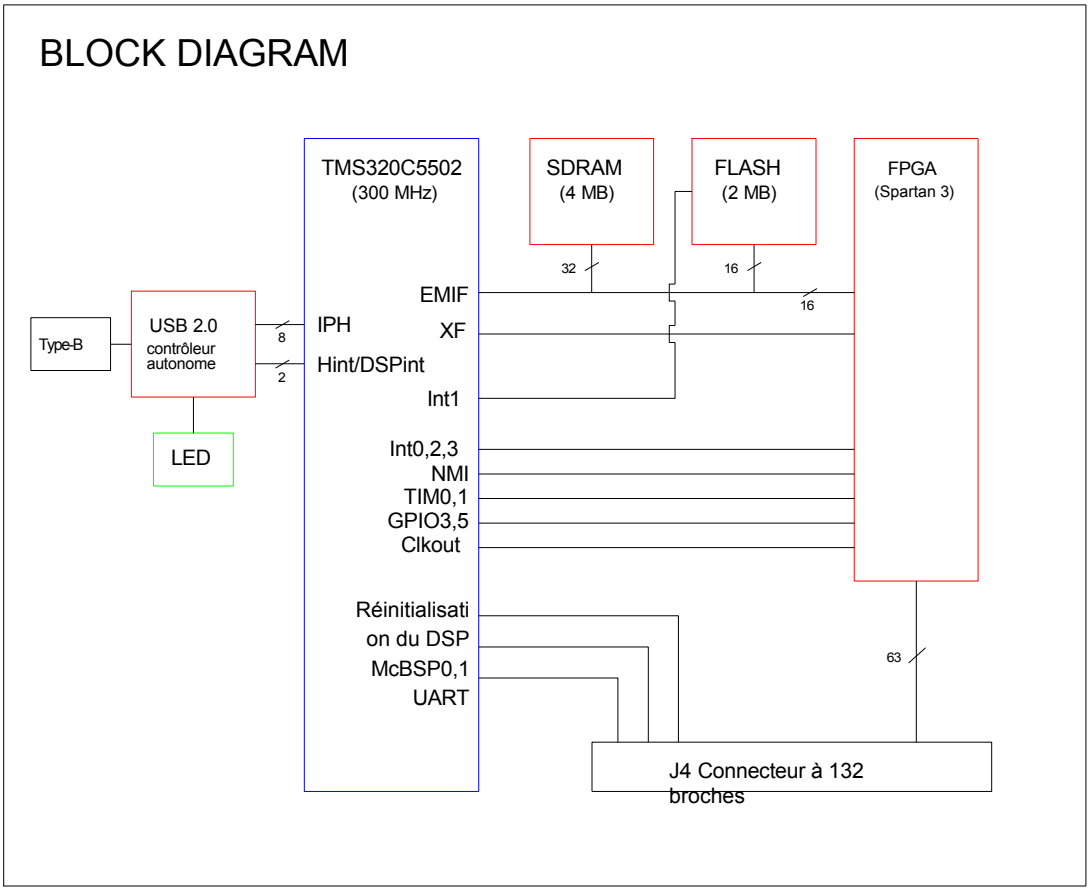


Figure 1 : Schéma fonctionnel de la carte DSP Signal Ranger mk2

Logiciel :

- **Pilote pour Win2k et WinXP :**
  Ce pilote permet de connecter un nombre illimité de cartes au PC.
- **Débogueur symbolique complet :**
  Le débogueur comprend des fonctions telles que le traçage graphique de données en temps réel, l'accès symbolique en lecture/écriture aux variables, l'exécution dynamique, la programmation Flash... etc. A la base, le mini

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.

- **Interface LabView :**
  Cette bibliothèque de VI LabView permet de développer un code LabView pour interfacer avec la carte DSP. Elle comprend des VI pour télécharger le code DSP (chargeur COFF), lancer les fonctions DSP et lire/écrire la mémoire DSP pendant l'exécution du code DSP.
- **Interface C/C++ :**
  Cette DLL permet de développer un code PC écrit en C/C++ pour interfacer avec la carte DSP. Elles comprennent des fonctions de téléchargement du code DSP (chargeur COFF), de lancement des fonctions DSP et de lecture/écriture de la mémoire DSP pendant l'exécution du code DSP.
- **Application d'autodiagnostic :**
  Cette application teste tout le matériel de la carte DSP.
- **Exemples de codes :**
  Plusieurs applications LabView de démonstration montrent le développement d'un code DSP en C et en assembleur. Elles montrent également comment interfacer ce code avec une application PC écrite en LabView. Une application de démonstration Visual Studio montre le développement d'une application PC écrite en C/C++.
- **Pilote Flash et code d'exemple :**
  Ce pilote comprend tout le code permettant de configurer et d'utiliser la ROM Flash embarquée de 2 Mbytes à partir du code DSP de l'utilisateur.
- **Configuration FPGA par défaut :**
  La carte est fournie avec une configuration FPGA par défaut, qui fournit 63 E/S numériques configurables.

Installation

Signal Ranger_mk2 fonctionne avec Windows 2000 et Windows XP. Aucun des logiciels de *Signal Ranger_mk2*, y compris le pilote, n'est compatible avec le logiciel précédent de *Signal Ranger.* Le nouveau logiciel doit être installé.

Note : Ne pas connecter la carte SignalRanger_mk2 DSP au port USB du PC tant que le logiciel n'a pas été installé. Le processus d'installation du pilote, qui se produit dès que la carte est connectée au PC, exige que le fichier du pilote soit présent sur le PC.

Installation du logiciel

Il suffit d'exécuter le fichier Setup.exe, qui installera deux ensembles de fichiers :

- Le logiciel *SignalRanger_mk2*, y compris les bibliothèques, la documentation et les applications de démonstration requises, sera installé en premier. Le programme d'installation crée un répertoire nommé *SignalRanger_mk2* dans *C:\NProgram Files*, et y stocke tous les fichiers requis. Il crée également des raccourcis dans le menu START de Windows.
- Le moteur d'exécution LabVIEW 7.1 est ensuite installé. Ce moteur d'exécution est nécessaire pour exécuter les versions compilées des applications de démonstration. Il est également nécessaire pour utiliser l'interface logicielle C/C++. Il n'est pas nécessaire pour utiliser l'interface logicielle LabVIEW. L'interface LabVIEW
  7.1 est installé dans le répertoire *C:\NProgram Files\NNational Instruments\NShared\NLabVIEW Run-Time\N7.1.*

Après cette installation, il est nécessaire d'installer le pilote USB pour la carte. Pour ce faire, reportez-vous à la section relative à l'installation du matériel ci-dessous.

Qu'est-ce qui est installé et où ?

Le programme d'installation crée le répertoire *C:\NProgram Files\NSignalRanger_mk2*. Ce répertoire contient tous les outils logiciels :

- Un répertoire nommé *Documentation* contenant toute la documentation requise, y compris le présent document.
- Un répertoire nommé *Driver* contenant le pilote de la carte.
- Un répertoire nommé *Exemples* contenant :
  - Un répertoire nommé *C_Examples* contenant un fichier zip. Une fois dégonflé, ce fichier zip contient l'application de démonstration de l'interface C/C++, y compris le projet PC Visual .net et le code DSP.
  - Un répertoire nommé *LabVIEW_Examples_DSPCode* contenant un fichier zip. Une fois dégonflé, ce fichier zip contient le code DSP des exemples LabVIEW ainsi que la documentation. Le code LabVIEW de ces exemples se trouve dans le répertoire d'installation principal, dans la bibliothèque DemoLabview.llb.
- Toutes les bibliothèques LabVIEW décrites dans ce document.
- La DLL *SRanger2.dll*, qui constitue le niveau d'interface le plus bas, est requise par tous les autres niveaux d'interface.
- Tous les fichiers exécutables DSP (".out") requis par les interfaces. Cela comprend les deux noyaux, ainsi que le code de support Flash et FPGA.
- Le fichier *SR2_SelfTest.rbt*, qui contient la logique FPGA par défaut.
- Le fichier *Revision_History.txt*, qui détaille l'historique des révisions.

Installation du matériel

Remarque : N'alimentez la carte qu'à l'aide de l'alimentation fournie ou d'une alimentation de 5V (+- 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 une alimentation appropriée, vous risquez d'endommager la carte.

- Mettez d'abord la carte sous tension 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_mk2* au port USB du PC.
- Après quelques secondes, Windows devrait détecter la nouvelle carte et présenter un assistant d'installation de pilote standard.
- Effectuez les sélections appropriées pour spécifier l'emplacement du pilote.
- Lorsque vous est demandé, accédez au répertoire *C:\NProgram* Files\NSignalRanger_mk2\NDriver, et sélectionnez le fichier *SRm2.inf.*
- Windows devrait installer le pilote.
- À ce moment-là, la DEL devient verte pour indiquer que le PC communique avec la carte.
- Après cette première installation, le PC devrait toujours reconnaître automatiquement la carte quelques secondes après sa connexion. Cela peut prendre plus de temps si la carte est connectée à une racine ou à un hub USB différent sur le PC. Dans ce cas, il est possible que le PC indique qu'un nouveau périphérique a été trouvé. Cependant, il devrait être en mesure de trouver le pilote automatiquement.
- À tout moment après que la carte a été connectée au PC et que ce dernier l'a reconnue, DEL doit être verte. La LED doit être verte avant d'essayer d'exécuter une application PC qui communique avec la carte.

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.
- Allez dans le menu DÉMARRAGE, sous *Paramètres, Panneau de configuration, Système*.
- Sélectionnez l'onglet *Matériel.*
- Appuyez sur le bouton *Gestionnaire de périphériques*.
- Accédez à l'élément *Contrôleurs de bus universel de série* et développez-le.

- L'arborescence doit comporter un élément "*Unknown Device" (dispositif inconnu).*
- Cliquez avec le bouton droit de la souris sur l'icône *Périphérique inconnu.*
- Cliquez sur *Mettre à jour le pilote.*
- Suivez ensuite la procédure d'installation du pilote décrite ci-dessus.

Indicateur LED

La LED de la carte Signal Ranger_mk2 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.
- Il devient orange de lui-même 1/2s après la mise sous tension de la carte. Cela indique que la carte est passée par la séquence de réinitialisation et d'initialisation appropriée. Si un code DSP de démarrage a été précédemment programmé dans la Flash ROM, ce code est en cours d'exécution lorsque la LED est orange.
- Elle devient verte lorsque la carte a été connectée au PC et que ce dernier a chargé son pilote. Le voyant vert indique que le PC est capable de communiquer avec la carte. La LED doit être verte avant d'essayer d'exécuter une application PC qui communique avec la carte.
- La LED devient orange lorsque la connexion USB est interrompue. C'est le cas lorsque le PC est éteint ou en veille, ou si le câble USB est débranché de la carte ou PC. Cependant, le fait que la LED devienne orange ne signifie pas que le code DSP a cessé de fonctionner.
- La LED peut également devenir orange temporairement lorsque la carte est en cours d'initialisation ou lorsque le FPGA est rechargé par une application PC.
- La couleur de la LED peut être modifiée à tout moment par une application utilisateur.

Note : Ce comportement des LED est différent de celui de la carte Signal Ranger SP2 : Le comportement de cette LED est différent de celui de la carte Signal Ranger SP2. Dans ce dernier cas, la LED devenant orange indiquait l'énumération du PC, tandis que la LED devenant verte indiquait l'initialisation du DSP. La signification des deux couleurs a été inversée dans la carte Signal Ranger_mk2.

Test de la carte

L'application *SR2_Self_Test* peut être lancée à tout moment après que la carte a été mise sous tension et connectée à un PC (une fois que la LED est devenue verte).

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

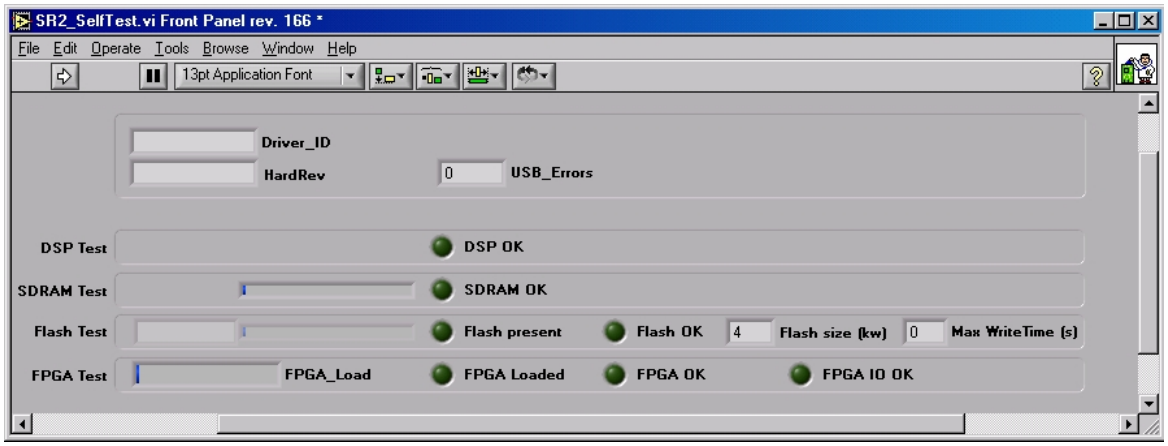


Figure 2 Application SR2_Self_Test

Pour relancer l'application après son arrêt, il suffit de cliquer sur la flèche en haut à gauche de la fenêtre.

L'application initialise la carte, charge le noyau sur le DSP et teste DSP, la RAM externe, la Flash et le FPGA. Chaque test dure quelques secondes.

Note : Le test Flash efface le contenu de la mémoire Flash : Le test Flash efface le contenu de la mémoire Flash. Une boîte de dialogue est présentée avant opération afin que l'utilisateur puisse l'annuler pour préserver le contenu de la mémoire flash.

Note : Le test des E/S du FPGA configure toutes les E/S comme des sorties pour les tester : Le test des E/S du FPGA configure toutes les E/S en tant que sorties pour les tester. Cela peut endommager une E/S ou une logique externe si une logique est connectée au FPGA (via le connecteur d'extension J4) pendant le test. Une boîte de dialogue est présentée avant cette opération afin que l'utilisateur puisse l'annuler pour éviter d'endommager les E/S connectées.

Indicateurs :

- **Driver_ID** : une chaîne de caractères de la forme *SRm2_x*, où 0<x<n-1 fait référence à la carte à laquelle on accède.

Les indices x sont attribués par le PC aux cartes *Signal_Ranger_mk2*, au moment de la connexion, dans l'ordre où elles sont connectées à la chaîne USB. Par exemple :

- SRm2_0 sera attribué à la première carte connectée
- SRm2_1 sera donné au second...etc.

- **HardRev** : Numéro de révision du micrologiciel du contrôleur USB embarqué.
- **USB_Errors** : Indique le nombre d'erreurs USB détectées pendant le test. Notez que la présence de certaines erreurs USB n'empêche pas le fonctionnement de la carte *Signal_Ranger_mk2*. Le protocole USB est très résistant aux erreurs. Cependant, il s'agit généralement d'une indication d'un mauvais câble USB ou d'une mauvaise connexion USB. Lorsque le taux d'erreur est trop élevé, il peut entraîner une rupture de la communication USB.
- **DSP OK** : S'allume en vert si le test DSP est réussi. Il s'allume en rouge dans le cas contraire.
- **SDRAM OK** : S'allume en vert si le test SDRAM est réussi. Il s'allume en rouge dans le cas contraire.
- **Flash présent** : S'allume en vert si le flash est détecté. S'allume en rouge s'il ne l'est pas.
- **Flash OK** : S'allume en vert si le test Flash est réussi. Il s'allume en rouge dans le cas contraire. Ne s'allume pas si le test est ignoré.
- **Taille du flash** : Devrait indiquer 1024 mots clés si la mémoire flash est correctement détectée.
- **Temps d'écriture maximum** : Indique le temps nécessaire pour programmer un mot de 16 bits dans la Flash. Il devrait être d'environ 60µs si la Flash est en bon état.
- **FPGA Loaded** : S'allume en vert si le FPGA est chargé avec succès. Il s'allume en rouge dans le cas contraire.
- **FPGA OK** : S'allume en vert si le FPGA est capable de communiquer avec le DSP. Il s'allume en rouge dans le cas contraire. Ne s'allume pas si le test est ignoré.
- **FPGA IO OK** : S'allume en vert si le test FPGA IO est réussi. Il s'allume en rouge dans le cas contraire. Ne s'allume pas si le test est ignoré.

Description du matériel

Carte du connecteur

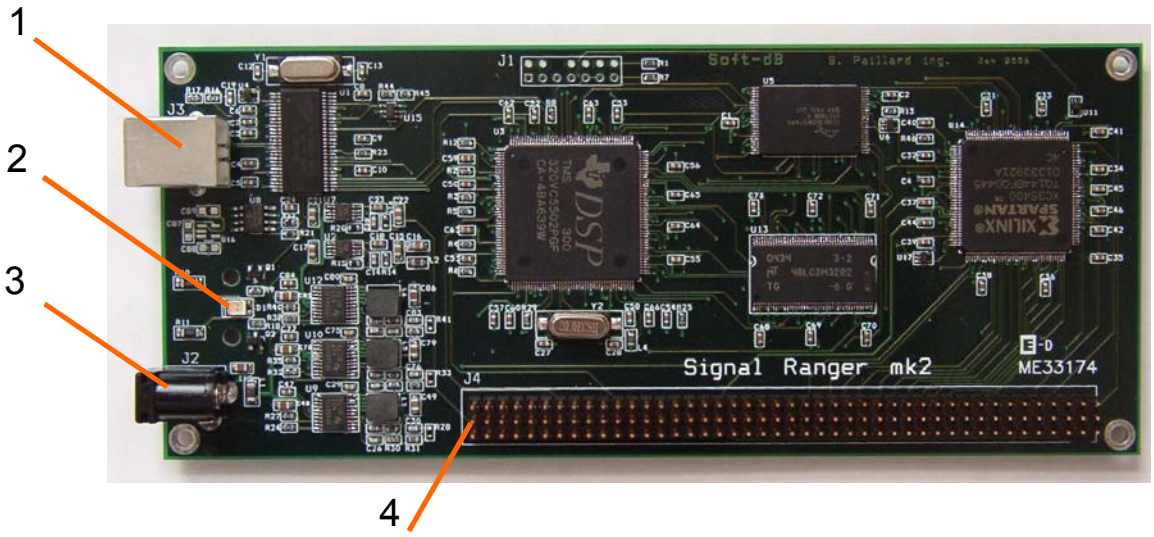
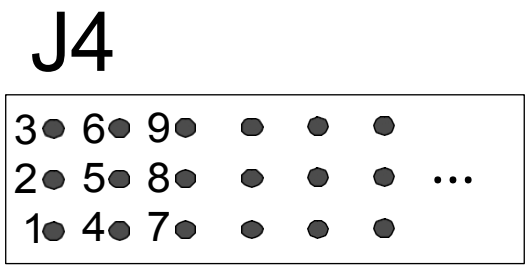


Figure 3

Légende :

1	Port USB
2	LED bicolore
3	Alimentation 5V
4	Connecteur d'extension J4

Connecteur d'extension J4



Non	Fonction	Non	Fonction	Non	Fonction
1	+5V	2	Gnd	3	+2.5V
4	+1.8V	5	Gnd	6	+1.2V
7	+1.26V	8	Gnd	9	+3.3V
10	-3.3V	11	Gnd	12	DSP_Reset
13	CLKR0	14	Gnd	15	DR0
16	FSR0	17	Gnd	18	CLKX0
19	DX0	20	Gnd	21	FSX0

22	CLKR1	23	Gnd	24	DR1
25	FSR1	26	Gnd	27	CLKX1
28	DX1	29	Gnd	30	FSX1
31	UART_Rx	32	Gnd	33	UART_Tx
34	NC	35	Gnd	36	NC
37	FPGA_17	38	Gnd	39	FPGA_15
40	FPGA_14	41	Gnd	42	FPGA_13
43	FPGA_12	44	Gnd	45	FPGA_11
46	FPGA_10	47	Gnd	48	FPGA_8
49	FPGA_7	50	Gnd	51	FPGA_6
52	FPGA_5	53	Gnd	54	FPGA_4
55	FPGA_2	56	Gnd	57	FPGA_1
58	FPGA_141	59	Gnd	60	FPGA_140
61	FPGA_137	62	Gnd	63	FPGA_135
64	FPGA_68	65	Gnd	66	FPGA_69
67	FPGA_70	68	Gnd	69	FPGA_73
70	FPGA_74	71	Gnd	72	FPGA_76
73	FPGA_77	74	Gnd	75	FPGA_78
76	FPGA_79	77	Gnd	78	FPGA_80
79	FPGA_82	80	Gnd	81	FPGA_83
82	FPGA_84	83	Gnd	84	FPGA_85
85	FPGA_86	86	Gnd	87	FPGA_87
88	FPGA_89	89	Gnd	90	FPGA_90
91	FPGA_92	92	Gnd	93	FPGA_93
94	FPGA_95	95	Gnd	96	FPGA_96
97	FPGA_97	98	Gnd	99	FPGA_98
100	FPGA_99	101	Gnd	102	FPGA_100
103	FPGA_102	104	Gnd	105	FPGA_103
106	FPGA_104	107	Gnd	108	FPGA_105
109	FPGA_107	110	Gnd	111	FPGA_108
112	FPGA_112	113	Gnd	114	FPGA_113
115	FPGA_116	116	Gnd	117	FPGA_118
118	FPGA_119	119	Gnd	120	FPGA_122
121	FPGA_123	122	Gnd	123	FPGA_124
124	FPGA_125	125	Gnd	126	FPGA_129
127	FPGA_130	128	Gnd	129	FPGA_131
130	FPGA_132	131	Gnd	132	FPGA_HS_EN

Broches d'alimentation

+5V

Il s'agit de la même ligne +5V qui est amenée au connecteur d'alimentation J2. Le courant maximum qui peut être tiré de cette broche est de 500mA. Il peut être limité par la capacité de l'alimentation utilisée.

+2.5V

Il s'agit de l'alimentation Vccaux du FPGA. Un maximum de 200mA peut être tiré de cette broche.

Note : Cette valeur prend en compte le courant de repos Vccaux du FPGA : Cette valeur tient compte du courant de repos Vccaux du FPGA. Elle est valable lorsque le FPGA n'est pas chargé ou qu'il est chargé avec la logique par défaut.

+1.8V

Cette alimentation n'est pas utilisée à bord. Elle est destinée aux dispositifs externes, tels que les ADC/DAC. Un maximum de 250mA peut être tiré de cette broche.

+1.2V

Il s'agit de l'alimentation Vccint du FPGA. Un maximum de 400mA peut être tiré de cette alimentation.

Note : Cette valeur prend en compte le courant de repos Vccint du FPGA : Cette valeur tient compte du courant de repos Vccint du FPGA. Elle est valable lorsque le FPGA n'est pas chargé ou qu'il est chargé avec la logique par défaut.

+1.26V

Il s'agit de l'alimentation CVdd du DSP. Un maximum de 250 mA peut être tiré de cette alimentation.

+3.3V

Cette alimentation est utilisée par le DSP, le FPGA, la Flash, la SDRAM et le contrôleur USB. Un maximum 100mA peut être tiré de cette alimentation.

Note : Cette valeur prend en compte le courant de repos Vcco du FPGA : Cette valeur tient compte du courant de repos Vcco du FPGA. Elle est valable lorsque le FPGA n'est pas chargé ou est chargé avec la logique par défaut. Elle ne tient pas compte du courant tiré des broches d'E/S du FPGA.

-3.3V

Cette alimentation n'est pas utilisée à bord. Elle est destinée aux dispositifs analogiques, tels que les ADC/DAC. Un maximum de 60mA peut être tiré de cette broche.

Broches DSP

DSP_Reset

Il s'agit de la broche de réinitialisation du DSP. Elle est activée (bas) à la mise sous tension et sous le contrôle du contrôleur USB. Elle peut être utilisée pour réinitialiser la logique externe chaque fois que le DSP est réinitialisé.

McBSP_0, McBSP_1

Tous les signaux McBSP_0 et McBSP_1 du DSP sont fournis sur J4. Ils peuvent être utilisés pour interfacer des dispositifs externes, tels que des AIC, avec le DSP.

UART

Les broches UART Tx et Rx du DSP sont fournies sur J4.

Broches FPGA

FPGA_i

Toutes les E/S libres et les broches d'horloge du FPGA sont fournies sur J4.

FPGA_HS_EN

Il s'agit de la broche HSWAP_EN du FPGA. Si elle est tirée vers le haut ou laissée flottante, toutes les E/S du FPGA sont flottantes pendant la configuration. Si elle est tirée vers le bas, les E/S du FPGA sont tirées vers le haut pendant la configuration. L'état des E/S du FPGA après la configuration est défini par la logique chargée dans le FPGA, et l'état de HSWAP_EN n'a aucune incidence sur celui-ci.

Fréquences du système

Le cristal du DSP a une fréquence de 20 MHz. Immédiatement après la réinitialisation, les différentes fréquences du système sont les suivantes :

- Horloge centrale du CPU : 20MHz
- SYSCLK1 (Périphériques rapides) : 5MHz
- SYSCLK2 (Périphériques lents) : 5MHz
- SYSCLK3 (EMIF) : 5MHz

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

- Horloge centrale du CPU : 300 MHz
- SYSCLK1 (Périphériques rapides) : 150 MHz
- SYSCLK2 (Périphériques lents) : 75 MHz
- SYSCLK3 (EMIF) : 75 MHz

Interfaces périphériques

Carte mémoire

La figure suivante décrit la carte mémoire du DSP. Les adresses sont exprimées en octets.

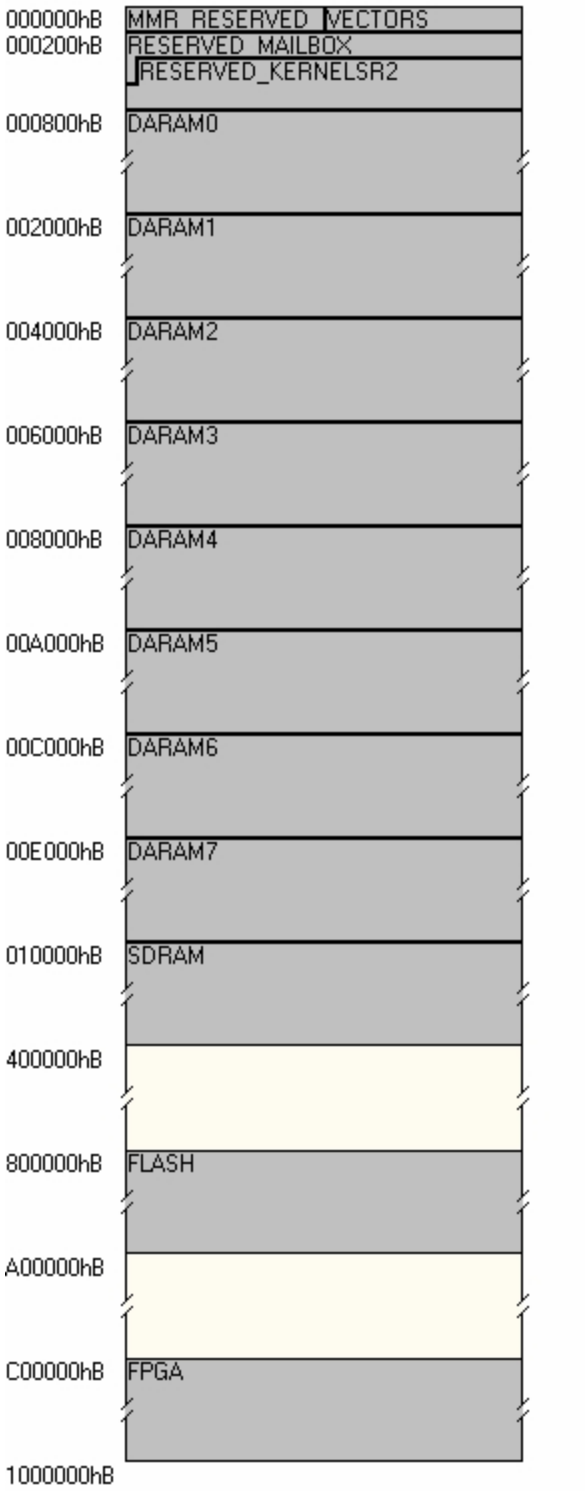


Figure 4 Carte de la mémoire

Configuration EMIF

L'EMIF est correctement configuré par le noyau pour accéder à la Flash, à la SDRAM et au FPGA.

Lorsque le DSP fonctionne à 300 MHz, l'EMIF est configuré par le noyau du DSP pour fonctionner à 75 MHz (1/4 de l'horloge du CPU).
Les registres EMIF sont configurés comme suit :

- Registre de contrôle global EMIF 1 (0x0800) Valeur : 0x07FC
  - NOHOLD = 1
  - EK1HZ = 1
  - EK1EN = 1

- Registre de contrôle global EMIF 2 (0x0801) Valeur : 0x000A
  - EK2RATE = 2
  - EK2HZ = 1
  - EK2EN = 0 (désactivé)

- EMIF CE0 Registre de contrôle de l'espace 1 (0x0804) Valeur : 0xFF33
(tous les champs sont des valeurs par défaut, sauf MTYPE)
  - TA = 3
  - READ STROBE = 63
  - MTYPE = 3
  - WRITE HOLD MSB = 0
  - READ HOLD = 3

- EMIF CE0 Registre de contrôle de l'espace 2 (0x0805) Valeur : 0xFFFF
(tous les champs sont par défaut)
  - WRITE SETUP (CONFIGURATION DE L'ÉCRITURE) = 15
  - ÉCRIRE STROBE = 63
  - WRITE HOLD = 3
  - READ SETUP = 15

- EMIF CE2 SPACE Control Register 1 (Registre de contrôle de l'espace) (0x0808) Valeur : 0xC811
  - TA = 3
  - READ STROBE = 8
  - MTYPE = 1
  - WRITE HOLD MSB = 0
  - READ HOLD = 1

- EMIF CE2 SPACE Control Register 2 (Registre de contrôle de l'espace) (0x0809) Valeur : 0x11E2
  - WRITE SETUP (CONFIGURATION DE L'ÉCRITURE) = 1
  - ÉCRITURE STROBE = 7
  - WRITE HOLD = 2
  - READ SETUP = 2

- EMIF CE3 SPACE Control Register 1 (Registre de contrôle de l'espace) (0x080A) Valeur : 0xC422
  - TA = 3
  - LIRE STROBE = 4
  - MTYPE = 1
  - WRITE HOLD MSB = 0
  - READ HOLD = 2

- EMIF CE3 SPACE Control Register 2 (Registre de contrôle de l'espace) (0x080B) Valeur : 0x2122
  - WRITE SETUP (CONFIGURATION DE L'ÉCRITURE) = 2
  - ÉCRITURE STROBE = 4
  - WRITE HOLD = 2

- READ SETUP = 2

- Registre de contrôle SDRAM EMIF 1 (0x080C) Valeur : 0x5000
  - CRT = 5 (6 cycles @75MHz pendant 70ns)
  - SLFRFR = 0 (SDRAM en fonctionnement normal)

- Registre de contrôle SDRAM EMIF 2 (0x080D) Valeur : 0x4611 (0x4711 pour init)
  - SDWDTH[4:0] = 3 (4 banques, 11 bits de ligne, 8 bits
  - RFEN = 1 de couleur) (Activation du
  - INIT = 0 rafraîchissement)
    (Pas d'Init)
  - TRCD = 1 (2 cycles @75MHz pendant 20ns)
  - TRP = 1 (2 cycles @75MHz pendant 20ns)

- Registre de contrôle du rafraîchissement de la SDRAM EMIF (0x080E) Valeur : 0x0494
  - Période = 1172 (15,625µs/rangée @ 75MHz)

- Registre 2 de contrôle du rafraîchissement de la SDRAM EMIF (0x080F)
  Valeur : 0x0000 (ne pas initialiser, laisser la valeur par défaut)
  - Rafraîchissements supplémentaires = 0 (1 rafraîchissement)

- Registre d'extension EMIF SDRAM (0x0810) Valeur : 0x4527
  - R2WDQM(L) = 0 (3 cycles valeur recommandée (CL= 3))
  - RD2WR = 4 (5 cycles valeur recommandée (CL= 3))
  - RD2DEAC = 1 (2 cycles valeur recommandée (CL= 3))
  - RD2RD = 0 (valeur recommandée pour 1 cycle (CL= 3))
  - THZP = 2 (3 cycles à 75 MHz)
  - TWR = 1 (2 cycles @75MHz pour 20,33ns)
  - TRRD = 0 (2 cycles @75MHz pendant 14ns)
  - TRAS = 3 (4 cycles @75MHz pendant 42ns)
  - TCL = 1 (3 cycles)

- Registre d'extension SDRAM EMIF 2 (0x0811) Valeur : 0x0005
  - WR2RD = 0 (valeur recommandée pour 1 cycle (CL= 3))
  - WR2DEAC = 1 (2 cycles valeur recommandée (CL= 3))
  - WR2WR = 0 (valeur recommandée pour 1 cycle (CL= 3))
  - R2WDQM(H) = 1 (3 cycles valeur recommandée (CL= 3))

SDRAM

Il s'agit d'un dispositif 2Mx32 (8 Mbytes). Cependant, l'interface EMIF ne permet d'accéder qu'à la première moitié (1Mx32) du dispositif. Avec la RAM intégrée, le dispositif SDRAM couvre la totalité de l'espace CE0.

Carte mémoire

La SDRAM est interfacée sur CE0, aux adresses d'octets 10000_H à $3FFFFF_{(H)}$ (adresses de mots 8000_H à $1FFFFF_{H)}$. Elle suit la DARAM du DSP sur la puce.

Vitesse

Le dispositif SDRAM est cadencé par ECLKOUT1. Il peut être cadencé à une fréquence allant jusqu'à 100 MHz. Cependant, lorsque le DSP fonctionne à 300MHz, ECLKOUT1 doit être réglé à 1/4 ^{de}la fréquence d'horloge du CPU (75MHz). C'est la configuration qui est fournie par défaut.

Si le CPU est réglé pour fonctionner à 200 MHz, ECLKOUT1 peut être réglé à ½ de la fréquence d'horloge du CPU. Dans ce cas, la SDRAM serait cadencée à 100 MHz. Cette configuration alternative donne un temps d'accès à la SDRAM légèrement plus rapide, au prix d'un DSP plus lent. Cette configuration alternative est possible. Cependant, ce n'est pas la configuration fournie par défaut.

Flash

Il s'agit d'un dispositif de 4 Mbytes (2Mx16). Cependant, l'interface EMIF ne permet d'accéder qu'à la moitié (1Mx16) du dispositif.

Carte mémoire

L'appareil est interfacé sur CE2, aux adresses d'octets 800000_Hà $9FFFFF_{(H)}$ (adresses de mots 400000_Hà $4FFFFF_{H)}$.

Secteurs

La FLASH est segmentée en 32 secteurs de 32 mots chacun.

Interruption

La sortie RY/BY (inversée) de l'appareil est connectée à l'entrée INT1 du DSP. Ainsi, les opérations de programmation et d'effacement peuvent être gérées à l'aide d'interruptions INT1. Le pilote Flash fourni utilise INT1 pour effectuer les écritures.

Efficacité de l'accès

Comme ce dispositif est configuré dans l'EMIF comme une mémoire de 16 bits de large, l'EMIF lit toujours deux mots à la fois, qu'un ou deux mots soient spécifiés par l'instruction DSP. Lorsqu'une instruction du processeur spécifie la lecture d'un seul mot de 16 bits, l'EMIF lit deux mots consécutifs et rejette celui qui n'est pas spécifié dans l'instruction. Ce processus coûte un cycle de lecture chaque fois qu'un mot de 16 bits est lu.
Cela ne se produit pas lorsque l'EMIF écrit dans la mémoire Flash.

Programmation incrémentale

Contrairement aux générations précédentes de dispositifs Flash utilisés dans les cartes *Signal Ranger*, le dispositif Flash utilisé dans *Signal_Ranger_mk2* ne peut pas être programmé de manière incrémentielle. Cela signifie qu'un emplacement de mot qui a été précédemment programmé DOIT être effacé avant d'être reprogrammé. Cela est vrai même si l'opération de reprogrammation ne vise qu'à transformer certains des "1" restants en "0".

FPGA

Carte mémoire

Le dispositif est interfacé sur CE3, aux adresses d'octets $C00000_H$à $FFFFFF_{(H)}$ (adresses de mots 600000_Hà $7FFFFF_{H)}$.

Interface physique

Les détails de l'interface entre le FPGA et le DSP sont les suivants :

- **Lignes de données :** Les 16 bits inférieurs du bus de données de l'EMIF sont connectés au FPGA. Seuls les 8 bits inférieurs sont utilisés lors de la configuration.
  - **Lignes de données (basses) :** Les lignes de bus de données EMIF D7 à D0 sont connectées aux broches 46, 47, 50, 51, 59, 60, 63 et 65 du FPGA respectivement. Elles sont utilisées pour la configuration du FPGA et peuvent être utilisées après la configuration pour la lecture/écriture du FPGA.

- **Lignes de données (haut) :** Les lignes de bus de données EMIF D15 à D8 sont connectées aux broches FPGA 28, 30, 31, 32, 33, 35, 36, 44 respectivement. Elles doivent être utilisées pour la lecture/écriture du FPGA après la configuration.

- **Lignes d'adresse :** Les lignes A6 à A2 de l'EMIF sont connectées aux broches 23, 24, 25, 26 et 27 du FPGA respectivement. Elles peuvent être utilisées après la configuration pour distinguer un maximum de 32 adresses logiques pendant la lecture/écriture du FPGA.
- **RDWR_B** L'entrée RDWR_B du FPGA est liée à la masse. Il n'est donc pas possible de relire les données de configuration. Cette broche ne doit pas être utilisée après la configuration car elle est liée à la terre.
- **PROG_B** L'entrée PROG_B du FPGA est connectée à la sortie XF du DSP et est utilisée pour initier la configuration du FPGA.
- **DONE** La broche DONE du FPGA est tirée vers le haut à 3,3V à l'aide d'une résistance de 10k, et connectée à la broche GPIO3 du DSP. GPIO3 doit être configuré comme une entrée et est nécessaire pour contrôler la configuration du FPGA.
- **INIT_B** La broche INIT_B du FPGA est tirée vers le haut à 3,3V à l'aide d'une résistance de 10k, et est connectée à l'INT3 du DSP. Elle est nécessaire pour surveiller le processus de configuration du FPGA et peut être utilisée pour déclencher l'interruption INT3 après la configuration. La broche INIT_B est également connectée à la broche GPIO5 du DSP. GPIO5 doit être configuré comme une entrée et peut être utilisé comme alternative à INT3 pour surveiller le processus de configuration du FPGA.
- **AWE** La ligne EMIF AWE est connectée au CCLK du FPGA par l'intermédiaire d'un tampon tolérant à 3,3V. Ce signal est utilisé pour configurer le FPGA. Il y a un délai maximum de 4,4ns entre la ligne EMIF AWE et l'entrée CCLK du FPGA. AWE est également connecté à la broche 56 du FPGA. Il s'agit d'une entrée GCLK. Elle peut être utilisée après la configuration pour un cycle d'écriture du FPGA.
- **ARE** La ligne ARE de l'EMIF est connectée à la broche 55 du FPGA. Il s'agit d'une entrée GCLK. Elle peut être utilisée après la configuration pour un cycle de lecture du FPGA.
- **AOE** La ligne AOE de l'EMIF est connectée à la broche 53 du FPGA. Il s'agit d'une entrée GCLK. Elle peut être utilisée après la configuration pour un cycle de lecture du FPGA.
- **CE3** La ligne EMIF CE3 est connectée au FPGA CS_B. Cette broche est utilisée pour sélectionner le FPGA pendant la configuration. Elle peut être utilisée après la configuration pour un cycle de lecture ou d'écriture du FPGA.
- **INT2** L'entrée INT2 du DSP est connectée à la broche 20 du FPGA. Elle peut être utilisée après la configuration pour déclencher l'interruption INT2. Pour l'utiliser comme entrée d'interruption externe, un simple suiveur peut être implémenté à l'intérieur du FPGA pour connecter une ligne de n'importe quelle E/S à usage général sur le connecteur d'extension J4 à la broche 20 du FPGA.
- **INT0** L'entrée INT0 du DSP est connectée à la broche 18 du FPGA. Elle peut être utilisée après la configuration pour déclencher l'interruption INT0. Pour l'utiliser comme entrée d'interruption externe, un simple suiveur peut être implémenté à l'intérieur du FPGA pour connecter une ligne de n'importe quelle E/S à usage général sur le connecteur d'extension J4 à la broche 18 du FPGA.
- **NMI** L'entrée NMI du DSP est connectée à la broche 21 du FPGA. Elle peut être utilisée après la configuration pour déclencher l'interruption NMI. Pour l'utiliser comme entrée d'interruption externe, un simple suiveur peut être implémenté à l'intérieur du FPGA pour connecter une ligne de n'importe quelle E/S à usage général sur le connecteur d'extension J4 à la broche 21 du FPGA.
- **DSP CLKOUT** La sortie d'horloge DSP CLKOUT est connectée à la broche 52 du FPGA. Il s'agit d'une entrée GCLK qui peut être utilisée pour cadencer la conception du FPGA. Par défaut, il fonctionne à 150 MHz après le chargement du noyau.
- **Horloge TIM0** L'horloge IO du DSP TIM0 est connectée à la broche 128 du FPGA. Il s'agit d'entrée GCLK qui peut être utilisée pour cadencer la conception du FPGA.
- **Horloge TIM1** L'entrée/sortie de l'horloge du DSP TIM1 est connectée à la broche 127 du FPGA. Il s'agit d'entrée GCLK qui peut être utilisée pour cadencer la conception du FPGA.

Configuration EMIF

L'EMIF est configuré sur CE3 (la plage d'adresses du FPGA) comme un dispositif 32 bits. La raison en est que si une autre largeur d'interface est utilisée (16 ou 8 bits), l'EMIF fait automatiquement plusieurs

au cours d'un cycle de lecture vers dispositif (2 accès pour une interface 16 bits et 4 accès pour une interface 8 bits - les accès en lecture sont toujours effectués pour un total de 32 bits). Bien que le comportement de lecture de l'EMIF ne soit généralement qu'une gêne lors de la lecture de mémoires d'une largeur autre que 32 bits (les cycles de lecture sont gaspillés et les données inutiles sont rejetées), les cycles de lecture supplémentaires peuvent poser des problèmes lors de la lecture de mémoires d'une largeur autre que 32 bits. Les cycles de lecture supplémentaires peuvent poser des problèmes lors de la lecture d'une logique spécialisée. Par exemple, si la logique implémentée sur le FPGA est une FIFO, les cycles de lecture supplémentaires feraient avancer la FIFO. Pour plus d'informations sur l'EMIF, voir le document SPRU621 de Texas Instrument.
Même si l'interface avec l'EMIF est une interface 32 bits, seuls les 16 bits inférieurs du bus de données de l'EMIF sont utilisés pour transporter les données vers et depuis le FPGA. Cela permet d'utiliser plus d'E/S dans la logique utilisateur du FPGA.
En raison de cette disposition particulière, il convient d'être prudent lors du transfert de données vers et depuis le FPGA, et en particulier lors de la configuration du FPGA. De plus amples détails sont fournis ci-dessous.

Configuration du FPGA

Pendant la configuration, les bits d'adresse EMIF sont ignorés par le FPGA. Une écriture à n'importe quelle adresse dans l'espace CE3 devrait être valide pour envoyer des données de configuration dans le FPGA. Cependant, comme l'EMIF est une interface 32 bits, l'adresse utilisée pour l'accès a un impact sur l'utilisation des 16 bits supérieurs ou inférieurs du bus de données pour le transfert. Pour s'assurer que les données sont transportées sur la partie basse du bus de données, qui est celle interfacée avec le FPGA, il faut s'assurer qu'une adresse mot impaire (une adresse octet dont le bit n° 1 est mis à 1) est utilisée pour accéder au FPGA, et qu'un accès 16 bits est utilisé (et non un accès 32 bits). Par exemple, une série d'écritures de mots à l'adresse-octet $C00002_H$peut être utilisée pour configurer le FPGA.

Pour lancer la configuration, le DSP doit envoyer une impulsion basse d'au moins 300ns sur XF. Il doit ensuite attendre que INIT_B (GPIO5) passe à l'état haut ou attendre au moins 5 ms après le front montant sur XF.

Lors de la configuration du FPGA, seuls les 8 bits inférieurs des données sont utilisés.

Installation suggérée après la configuration

Après la configuration, il est suggéré que l'interface EMIF vers le FPGA reste configurée comme une interface asynchrone 32 bits, utilisant uniquement les bits de données 0 à 15 (les bits de données 16 à 31 de l'EMIF ne sont pas physiquement connectés au FPGA).
Seuls les 5 bits d'adresse inférieurs de l'EMIF sont connectés aux E/S d'usage général du FPGA. Ces adresses peuvent être utilisées pour sélectionner un registre parmi un maximum de 32 registres qui peuvent être définis dans la logique du FPGA. Le tableau suivant indique l'adresse qui doit être utilisée point de du DSP, en fonction des 5 lignes d'adresse. Comme pour la configuration, le tableau suppose qu'un accès par mot est effectué par le DSP.

Adresse du registre FPGA	Adresse de l'octet du DSP	Adresse du mot DSP
00_H	$C00002_H$	600001_H
01_H	$C00006_H$	600003_H
02_H	$C0000A_H$	600005_H
03_H	$C0000E_H$	600007_H
04_H	$C00012_H$	600009_H
05_H	$C00016_H$	$60000B_H$
06_H	$C0001A_H$	$60000D_H$
07_H	$C0001E_H$	$60000F_H$
08_H	$C00022_H$	600011_H
09_H	$C00026_H$	600013_H
$0A_H$	$C0002A_H$	600015_H
$0B_H$	$C0002E_H$	600017_H
$0C_H$	$C00032_H$	600019_H
$0D_H$	$C00036_H$	$60001B_H$

$0E_H$	$C0003A_H$	$60001D_H$
$0F_H$	$C0003E_H$	$60001F_H$
10_H	$C00042_H$	600021_H
11_H	$C00046_H$	600023_H
12_H	$C0004A_H$	600025_H
13_H	$C0004E_H$	600027_H
14_H	$C00052_H$	600029_H
15_H	$C00056_H$	$60002B_H$
16_H	$C0005A_H$	$60002D_H$
17_H	$C0005E_H$	$60002F_H$
18_H	$C00062_H$	600031_H
19_H	$C00066_H$	600033_H
$1A_H$	$C0006A_H$	600035_H
$1B_H$	$C0006E_H$	600037_H
$1C_H$	$C00072_H$	600039_H
$1D_H$	$C00076_H$	$60003B_H$
$1E_H$	$C0007A_H$	$60003D_H$
$1F_H$	$C0007E_H$	$60003F_H$

Contraintes liées au brochage du FPGA

Étant donné que certaines des broches du FPGA qui peuvent devenir des E/S utilisateur après la configuration sont physiquement liées à des sorties DSP, Vcc ou Gnd, il est essentiel que le concepteur du FPGA vérifie correctement le brochage du FPGA après la configuration. Un mauvais brochage du FPGA, conduisant par exemple à ce qu'une sortie du FPGA soit reliée à une masse ou à une sortie EMIF, peut endommager le matériel. En particulier, les broches suivantes du FPGA doivent être vérifiées :

- Les broches 28, 30, 31, 32, 33, 35, 36, 44, 46, 47, 50, 51, 59, 60, 63 et 65 du FPGA sont connectées au bus de données de l'EMIF (D0 à D15). Après configuration, ces broches FPGA ne doivent être utilisées que pour échanger des données avec le DSP (conformément au cycle de lecture/écriture propre à l'EMIF), configurées en tant qu'entrées ou laissées flottantes.
- Les broches 23, 24, 25, 26 et 27 du FPGA sont connectées au bus d'adresse de l'EMIF (a0 à A4). Après la configuration, ces broches ne doivent être configurées que comme des entrées ou laissées flottantes.
- La broche 41 du FPGA (RDWR_B) est liée à la masse. Après la configuration, cette broche ne doit configurée que comme une entrée ou laissée flottante.
- Les broches 40, 53, 55 et 56 du FPGA sont connectées aux sorties de contrôle EMIF. Après la configuration, ces broches ne doivent être configurées qu'en tant qu'entrées ou laissées flottantes.
- Les broches 52, 127 et 128 du FPGA sont connectées aux sorties d'horloge du DSP. Après la configuration, ces broches ne doivent être configurées que comme des entrées ou laissées flottantes.

Critères d'évaluation de l'accès périphérique

Remarque : les repères ci-dessous supposent que l'EMIF est prêt à exécuter le cycle de lecture ou d'écriture. En particulier, aucun cycle de rafraîchissement de la SDRAM ne se produit pendant l'accès à la lecture ou à l'écriture.

SDRAM

Écriture 16 bits

147 ns / 11 cycles EMIF

Lecture 16 bits

240 ns / 18 cycles EMIF

Écriture 32 bits

147 ns / 11 cycles EMIF

Lecture 32 bits

240 ns / 18 cycles EMIF

Flash

Écriture 16 bits

147 ns / 11 cycles EMIF

- 1 cycle Configuration de l'écriture
- 7 cycles Strobe d'écriture
- 2 cycles Maintien de l'écriture
- 1 cycle Période de maintien de l'écriture (ajoutée pour que le nombre total de cycles soit de>= 11)

Note : Il ne s'agit pas du temps de programmation : Il ne s'agit pas du temps de programmation, mais du temps du cycle d'écriture, qui fait partie de lopération de programmation.

Lecture 16 bits

440 ns / 33 cycles EMIF

- 2 cycles Configuration de la lecture
- 8 cycles Lecture stroboscopique
- 1 cycle Maintien de la lecture
- 2 cycles Configuration de la lecture
- 8 cycles Lecture stroboscopique
- 1 cycle Maintien de la lecture
- 11 cycles Période de maintien de la lecture (ajoutée à la dernière période de maintien de la lecture de sorte que la somme soit égale à 12 cycles)

Remarque : la Flash étant configurée comme un périphérique de 16 bits, une opération de lecture dans la Flash déclenche en fait deux cycles de lecture consécutifs de la part de l'EMIF. Le premier est à l'adresse requise. Le second est à l'adresse suivante. Les données de la deuxième lecture sont rejetées par l'EMIF. Cela ne se produit pas pour les écritures.

FPGA

Écriture 16 bits

147 ns / 11 cycles EMIF

- 2 cycles Configuration de l'écriture
- 4 cycles Strobe d'écriture
- 2 cycles Maintien de l'écriture
- 3 cycles Période de maintien de l'écriture (ajoutée pour que le nombre total de cycles soit de>= 11)

Lecture 16 bits

240 ns / 18 cycles EMIF

- 2 cycles Configuration de la lecture
- 4 cycles Lecture stroboscopique
- 2 cycles Maintien de la lecture
- 10 cycles Période de maintien de la lecture (ajoutée à la dernière période de maintien de la lecture de sorte que la somme soit égale à 12 cycles)

Remarque : le FPGA étant configuré comme un périphérique 32 bits, l'EMIF n'effectue qu'un cycle de lecture (au lieu de deux cycles de lecture consécutifs) pour chaque opération de lecture à une adresse donnée.

Logique FPGA par défaut

La Flash est chargée en usine avec un fichier logique FPGA par défaut. Ce fichier est chargé dans le FPGA à la mise sous tension par le noyau Power-Up. Le fichier s'appelle *SR2_SelfTest.rbt* et se trouve dans le répertoire *C:\NProgram Files\NSignalRanger_mk2*. Au cas où la Flash serait réécrite avec une configuration FPGA différente, la logique FPGA par défaut peut être programmée à nouveau dans la Flash à l'aide du mini-débogueur.
Cette logique par défaut met en œuvre trois registres d'E/S générales de 16 bits et un registre d'E/S générales de 15 bits.
La logique n'utilise que 3% des ressources logiques du FPGA et ne consomme pas d'énergie significative au-delà des valeurs de courant de repos du FPGA (voir la documentation Xilinx).
Les E/S de ces registres sont mappées sur les broches suivantes du connecteur d'extension J4 :

Non	Fonction	Non	Fonction	Non	Fonction
37	PortC_0	38	Gnd	39	PortD_1
40	PortC_1	41	Gnd	42	PortD_2
43	PortC_2	44	Gnd	45	PortD_3
46	PortC_3	47	Gnd	48	PortD_4
49	PortC_4	50	Gnd	51	PortD_5
52	PortC_5	53	Gnd	54	PortD_6
55	PortC_6	56	Gnd	57	PortD_7
58	PortC_7	59	Gnd	60	PortD_8
61	PortC_8	62	Gnd	63	PortD_9
64	PortC_9	65	Gnd	66	PortD_10
67	PortC_10	68	Gnd	69	PortD_11
70	PortC_11	71	Gnd	72	PortD_12
73	PortC_12	74	Gnd	75	PortD_13
76	PortC_13	77	Gnd	78	PortD_14
79	PortC_14	80	Gnd	81	PortD_15
82	PortC_15	83	Gnd	84	PortB_0
85	PortA_0	86	Gnd	87	PortB_1
88	PortA_1	89	Gnd	90	PortB_2
91	PortA_2	92	Gnd	93	PortB_3
94	PortA_3	95	Gnd	96	PortB_4
97	PortA_4	98	Gnd	99	PortB_5
100	PortA_5	101	Gnd	102	PortB_6
103	PortA_6	104	Gnd	105	PortB_7
106	PortA_7	107	Gnd	108	PortB_8
109	PortA_8	110	Gnd	111	PortB_9
112	PortA_9	113	Gnd	114	PortB_10
115	PortA_10	116	Gnd	117	PortB_11
118	PortA_11	119	Gnd	120	PortB_12
121	PortA_12	122	Gnd	123	PortB_13
124	PortA_13	125	Gnd	126	PortB_14
127	PortA_14	128	Gnd	129	PortB_15
130	PortA_15	131	Gnd	132	FPGA_HS_EN

Chacun des quatre ports (PortA, PortB, PortC, PortD) possède deux registres :

- **PortX_Dir** est un registre de direction. Il définit la broche correspondante comme une entrée ou sortie. En écrivant un modèle de bit dans le registre *PortX_Dir*, chaque broche du port peut être configurée comme une entrée (0) ou une sortie (1).
- **PortX_Dat** est un registre de données. Il peut être lu pour déterminer l'état d'une broche d'entrée ou écrit pour définir l'état d'une broche de sortie.

Détails du matériel

Toutes les broches sont configurées comme des entrées à la mise sous tension.

Toutes les entrées et sorties sont programmées pour être conformes à la norme LVCMOS 3,3V.

Lorsqu'une broche de port est configurée comme entrée, un gardien faible est instancié pour l'entrée. De cette façon, la broche conservera son dernier état si elle est laissée flottante.

Les broches configurées comme sorties ont une capacité de courant de +- 12mA.

Si FPGA_HS_EN est tiré vers le bas lors de la mise sous tension de la carte, alors des pull-ups apparaîtront sur toutes les broches du FPGA pendant la configuration. Si FPGA_HS_EN est tiré vers le haut ou laissé flottant lors de la mise sous tension de la carte, les pull-ups ne seront pas présents lors de la configuration. Dans tous les cas, les pull-ups disparaissent une fois que le FPGA est configuré, et seuls les weak keepers sont laissés sur les broches.

Remarque : Les broches du FPGA configurées comme entrées ne doivent pas être pilotées par une tension supérieure à 3.8V lorsque le FPGA est alimenté, ou supérieure à 0.5V lorsque le FPGA n'est pas alimenté. Alternativement, le courant à travers n'importe quelle entrée ne doit pas dépasser 10mA. Se référer à la documentation Xilinx pour plus de détails.
Pour les applications où les E/S du FPGA peuvent être pilotées par des tensions supérieures à 3.3V, ou peuvent être pilotées lorsque la carte Signal Ranger mk2 n'est pas alimentée, nous recommandons l'utilisation d'un commutateur de bus tel que le SN74CB3T16211.

Carte du registre

Registre	Octet-Adresse	Adresse du mot
Port_A_Dat	$C00002_H$	600001_H
Port_A_Dir	$C00006_H$	600003_H
Port_B_Dat	$C0000A_H$	600005_H
Port_B_Dir	$C0000E_H$	600007_H
Port_C_Dat	$C00012_H$	600009_H
Port_C_Dir	$C00016_H$	$60000B_H$
Port_D_Dat	$C0001A_H$	$60000D_H$
Port_D_Dir	$C0001E_H$	$60000F_H$

Interfaces logicielles

Dans les sections suivantes, l'interface LabVIEW est utilisée comme exemple. concepts abordés sont également applicables à l'interface C/C++. Il suffit de remplacer le terme "VI" par "fonction" dans le texte. Chaque VI spécifique discuté ci-dessous a une fonction correspondante dans l'interface C/C++.

Comment les cartes DSP sont-elles gérées ?

Chaque fois qu'une carte DSP est connectée au PC, celui-ci charge son pilote et crée une structure de données représentant la carte dans le système. Cette structure de données est accessible par un nom symbolique de la forme "nom de *base_i*", où *nom de base* est un nom symbolique qui représente le type carte (par exemple, les cartes SignalRanger_mk2 portent le nom *SRm2_*), et *i* est un nombre à base zéro qui est attribué à la carte au moment de la connexion. Par exemple, si 3 cartes SignalRanger_mk2 sont connectées en séquence sur le PC, leurs noms symboliques complets seront *SRm2_0*, SRm2_1et *SRm2_2*.

Pour chaque carte ouverte (à l'aide du VI *SR2_Base_Open_Next_Avail_Board* dans l'interface LabVIEW), le logiciel d'interface crée et conserve une entrée dans une *structure globale d'information sur* les cartes qui contient des informations sur les cartes ouvertes. Cette structure contient les informations suivantes :

- Une poignée sur le pilote de la carte qui est nécessaire pour accéder à la carte
- Une structure d'identification de la carte qui contient les éléments suivants :
  - Numéro d'identification du fournisseur USB
  - Numéro d'identification du produit USB
  - Numéro de révision du matériel
- Une table de symboles pour les symboles du noyau actuellement chargé sur le DSP. Cette table est créée/mise à jour à chaque fois qu'un noyau est chargé ou rechargé.
- Une table de symboles pour les symboles du code utilisateur actuellement chargé sur le DSP. Cette table est créée/mise à jour chaque fois qu'un nouveau code DSP est chargé sur le DSP.
- Une structure USB retries de haut niveau. Lorsqu'une transaction USB échoue (en raison d'un bruit sur la ligne USB ou d'autres conditions anormales), l'interface retente la transaction 5 fois, avant de renvoyer une erreur. La structure est mise à jour chaque fois qu'une transaction est retentée. Elle contient la chaîne d'appel complète de Vis qui a généré la transaction défectueuse. Il convient de noter que cette structure de contrôle d'erreur est mise en œuvre à un niveau élevé, "au-dessus" du contrôle d'erreur intrinsèque de l'USB. Le protocole USB lui-même réessaie jusqu'à trois fois en cas d'erreurs USB, avant de faire échouer la transaction. Les tentatives de haut niveau n'ont lieu qu'après l'échec d'une transaction USB de bas niveau.

Tous les Vis de l'interface doivent recevoir un numéro **BoardRef.** Ce numéro renvoie à l'entrée correspondant à la carte sélectionnée dans la *structure globale d'information sur les cartes*. Il est créé par le VI *SR2_Base_Open_Next_Avail_Board*.

Lorsqu'une commission est fermée, son entrée dans la *structure d'information globale de la commission* est supprimée.

Remarque : La poignée que le pilote fournit pour accéder à la carte est exclusive. Cela signifie qu'une seule application ou un seul processus à la fois peut ouvrir et gérer une carte. 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 SR2_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 SR2_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 ne parviendra pas à trouver et à ouvrir la carte, simplement parce que l'instance de pilote correspondante est toujours ouverte.
Une procédure pour sortir de ce blocage consiste à fermer complètement l'application, à déconnecter la carte, puis à reconnecter la carte et à rouvrir et exécuter l'application. La fermeture de l'application est nécessaire en plus de la déconnexion de la carte pour que Windows puisse décharger le pilote.

Interface noyau et interface non-noyau Vis

Certaines des fonctions de transfert de mémoire sont prises en charge par deux groupes apparemment équivalents de Vis : ceux qui utilisent un noyau DSP et ceux qui ne l'utilisent pas. Il existe des différences fonctionnelles majeures entre les deux :

- La différence la plus fondamentale est que les Vis qui exécutent des transferts à l'aide du noyau exigent que le noyau soit chargé dans la mémoire du DSP et qu'il fonctionne correctement pour effectuer le transfert. Ils risquent donc d'échouer si le DSP tombe en panne. En revanche, les Vis qui n'utilisent pas le noyau s'appuient uniquement sur le matériel de l'IPH pour exécuter le transfert. Ils n'échoueront pas même si le DSP s'est planté. Il s'agit d'un élément important à prendre en compte lors du débogage du code.
- Les visiteurs qui utilisent le noyau ont accès à n'importe quelle plage d'adresses dans n'importe quel espace mémoire. En revanche, les Vis qui n'utilisent pas le noyau ont un champ d'action limité. Ils ne peuvent transférer que vers et depuis la mémoire directement accessible par le matériel de l'IPH (essentiellement la DARAM sur puce du DSP).
- Les vis qui utilisent le noyau utilisent des conduites en vrac pour effectuer le transfert. Les vis qui n'utilisent pas le control pipe 0.
    - L'une des conséquences de l'utilisation de bulk pipes pour les transferts utilisant le noyau est que la bande passante du transfert est beaucoup plus rapide. Les transferts utilisant le control pipe zero n'ont qu'une bande passante limitée (environ 500kb/s pour USB 1.1).
    - Une autre conséquence de l'utilisation de bulk pipes pour les transferts utilisant le noyau est que la bande passante est partagée avec d'autres dispositifs USB dans la même chaîne et n'est pas garantie. En principe, il est possible qu'un autre périphérique de la chaîne USB prenne tellement de bande passante qu'il ralentirait considérablement les transferts de la carte DSP. D'autre , les transferts qui n'utilisent pas le noyau bénéficient de la bande passante garantie du control pipe zéro (aussi faible).

En bref, les transferts qui utilisent le noyau sont plus performants et ont une portée beaucoup plus large. D'autre part, les transferts qui n'utilisent pas le noyau sont beaucoup plus fiables, en particulier pendant le débogage ou dans des circonstances où le code DSP peut se bloquer.

Contrôle des erreurs

Il existe quatre niveaux de contrôle des erreurs et quatre types d'erreurs de base :

Erreurs USB de bas niveau

Le protocole USB lui-même fournit un grand nombre de contrôles d'erreurs. Le pilote du contrôleur hôte sur le PC réessaie les paquets qui contiennent des erreurs, en raison du bruit, d'un câble défectueux ou d'autres conditions anormales, jusqu'à trois fois par transaction avant de les rejeter. Ce niveau de contrôle des erreurs est généralement caché à l'utilisateur et au développeur. Le contrôleur USB de la carte DSP dispose toutefois d'un compteur d'erreurs auquel le code PC peut accéder pour contrôler la fiabilité de connexion USB. Ce compteur d'erreurs circulaire de 4 bits est incrémenté chaque fois qu'une erreur USB est détectée. L'interface Vis est fournie pour lire et réinitialiser ce compteur. Ce compteur indique les erreurs survenant au niveau du protocole USB (bas niveau).

Tentatives USB

L'interface logicielle décrite ci-dessous possède sa propre couche de contrôle d'erreur USB qui opère à un niveau supérieur. Chaque fois que le pilote du contrôleur hôte sur le PC décide de faire échouer une transaction (après 3 tentatives), le logiciel de l'interface la retente jusqu'à 5 fois. À chaque fois, il enregistre une entrée dans le membre *Retry* de la *structure d'information globale de la carte* (voir ci-dessus), indiquant le nombre de tentatives et la chaîne d'appels qui a conduit à l'erreur. Si, après cinq tentatives, la transaction n'a toujours pas abouti, le VI à l'origine de l'erreur abandonne et renvoie une erreur. Le membre *Retry* de la *structure d'information globale du conseil d'administration* peut être lu à tout moment afin de déterminer si des erreurs se produisent régulièrement.

Erreurs de haut niveau

Lorsqu'une transaction USB échoue au niveau de l'interface (après 5 tentatives de haut niveau, soit 15 tentatives de bas niveau), la transaction échoue au niveau du haut niveau et le VI qui a créé les conditions renvoie un code d'erreur approprié dans sa structure *Error-Out.* Ce code d'erreur peut être traité par le VI *SR2_Base_Error_Message* pour obtenir une description textuelle de l'erreur, ainsi que la chaîne d'appels qui a conduit à l'erreur. Notez que les erreurs autres que les transactions USB échouées sont également signalées dans la structure Error-Out de SR2_Base_Error_Message.

la structure *Error-Out*. Par exemple, si des fichiers critiques sont manquants (comme un noyau), cela également signalé à ce niveau.

Erreurs et codes d'achèvement DSP spécifiques à l'application

Le noyau DSP gère un champ dans sa boîte aux lettres qui peut être utilisé pour renvoyer un code d'erreur ou d'achèvement spécifique à l'utilisateur chaque fois qu'une fonction DSP utilisateur est exécutée. Ce code d'erreur ou d'achèvement est entièrement sous le contrôle du développeur. Il peut être utilisé ou non. Ce code est renvoyé en tant qu'indicateur *ErrorCode* par tous les Vis qui accèdent à la DSP via le noyau.

Remarque : Pour que ce code d'erreur soit renvoyé par le DSP, il ne doit pas y avoir d'erreur de communication USB.

Verrouillage de l'USB

Contrairement à ce qui s'est passé pour les cartes précédentes de Signal Ranger, il y a des situations dans lesquelles
Signal_Ranger_mk2 qui peut entraîner un blocage du canal USB :

- Le TMS320VC5502 dispose d'une horloge étendue et d'une fonctionnalité périphérique sous contrôle logiciel. A tel point qu'il est possible d'arrêter l'horloge du DSP par logiciel. L'arrêt de l'horloge du processeur gèle le canal de communication USB.
- Il est également possible de désactiver complètement le HPI par logiciel. Cette situation gèlera également le canal de communication.
- La DARAM ne peut supporter que 2 accès par cycle, pour lesquels le CPU a toujours la priorité sur les accès DMA et HPI. Certaines séquences rares d'instructions, lorsqu'elles sont exécutées dans une boucle serrée, nécessitent ces deux accès à la DARAM par cycle, refusant ainsi complètement l'accès du bloc DARAM à partir duquel le code s'exécute à la DMA et à la HPI. Cette situation ne se produit que lorsque le code est exécuté à partir du même bloc DARAM que celui où se trouve la boîte aux lettres (le premier bloc).

Dans l'une ou l'autre de ces conditions, le contrôleur USB est incapable de transférer des données vers et depuis le HPI. Plus précisément, le contrôleur USB attend indéfiniment que le HPI signale qu'il est prêt à transférer des données, ce qui ne se produit jamais. La communication USB et l'application PC contrôlant la carte DSP semblent se figer. Aucun message d'erreur n'est transmis à l'utilisateur.

Les actions les plus simples pour sortir la carte de cette situation de blocage sont les suivantes :

- Débrancher et rebrancher le connecteur USB ou
- Faites un cycle de l'alimentation de la carte.

Accès symbolique

Le Vis d'accès au DSP, y compris le Vis de lecture et d'écriture de la mémoire, et le Vis d'exécution des fonctions du DSP comprennent un accès symbolique complet. Cela signifie que les lectures et les écritures en mémoire, ainsi que l'exécution des fonctions, peuvent être dirigées vers un symbole défini lors de la création du code DSP, plutôt que vers une adresse absolue. Il s'agit d'une caractéristique puissante, car elle permet au code PC d'être insensible à la reconstruction du code DSP. En effet, après la reconstruction d'un code DSP, les adresses des symboles accédés peuvent avoir changé, mais les symboles restent les mêmes et, par conséquent, le côté PC du code n'a pas besoin d'être reconstruit.
L'accès symbolique fonctionne en comparant la chaîne ASCII du symbole spécifié à une table de symboles qui est chargée dans la *structure Global Board Information*, lorsque le code DSP est chargé en mémoire. Lorsqu'une correspondance est trouvée, le symbole est remplacé par sa valeur et son espace mémoire, tels qu'ils figurent dans la table des symboles. Si aucune correspondance n'est trouvée, une erreur est générée. Pour désactiver l'accès symbolique, il suffit de laisser la chaîne ASCII du symbole vide. Dans ce cas, l'accès est effectué en utilisant l'adresse absolue et l'espace mémoire spécifiés explicitement.
La table de symboles correspondant au code DSP est analysée et rechargée dans la *structure Global Board Information* à partir du fichier exécutable ".out" du DSP chaque fois qu'un nouveau code DSP est rechargé dans la mémoire du DSP. Une nouvelle table de symboles peut également être rechargée à partir du fichier ".out" du DSP,

sans modifier le code DSP. Ceci est utile pour obtenir un accès symbolique à un code DSP qui est chargé et exécuté à partir de la mémoire Flash lors de la mise sous tension.

Les lignes directrices suivantes doivent être respectées pour que l'accès symbolique fonctionne correctement :

- Seuls les symboles globaux trouvés dans le fichier ".out" du DSP sont conservés dans la table des symboles. Cela signifie que pour permettre l'accès symbolique, les variables déclarées en C doivent être déclarées comme globales (en dehors de tous les blocs et fonctions, et sans le mot-clé *static*). Les variables et les étiquettes déclarées en assembleur (points d'entrée des fonctions par exemple) doivent être déclarées avec la directive *.global*.
- Lorsqu'une variable ou un point d'entrée de fonction est déclaré en C, le compilateur C ajoute un trait de soulignement au début du nom. Ce trait de soulignement ne doit pas être omis lors de la spécification d'un nom symbolique pour un accès.

Dans cette nouvelle version des interfaces logicielles, la fonction d'accès symbolique a été étendue pour permettre l'accès aux membres de la structure, avec une capacité d'imbrication illimitée. Toutefois, pour bénéficier de cette nouvelle fonctionnalité, il convient de respecter les lignes directrices suivantes :

- Le nom symbolique auquel on accède doit être construit à l'aide de noms de membres symboliques, séparés par des ".", comme on le fait en C : "*Nom_de_structure_globale.Nom_de_membre_1.Nom_de_membre_2...*" où :
  - *Nom_de_la_structure_globale* est le nom global donné à l'instance de la structure primaire à laquelle on accède
  - *Nom_du_membre_1* est le nom du membre de la structure auquel il faut accéder
  - *Nom_du_membre_2* est le nom du membre d'une structure imbriquée dans structure primaire, à laquelle il faut accéder.
- Lorsque les membres d'une structure sont déclarés en C, le compilateur C ajoute un trait de soulignement début du nom. Ce trait de soulignement ne doit pas être omis lors de la spécification d'un nom de membre symbolique pour un accès.
- Le code DSP doit être compilé avec l'option ***Full Symbolic Debugging***. Sinon, les informations sur la structure ne sont pas présentes dans le fichier exécutable ".out".

Par exemple, si les structures sont déclarées de la manière suivante :

```
struct      SignatureIndex      {       unsigned int        i_sample [N_SigNibbles] ;
                                        unsigned int        i_channel [N_SigNibbles] ;
                                        unsigned int        i_mask    [N_SigNibbles] ;
                                        } ;

#define     SigBufSize                  1000*6

struct      SigBuffer           {       unsigned int                Erreur ;
                                        unsigned int                Remplir ;
                                        unsigned int                Taille ;
                                        unsigned int                InPoint ;
                                        unsigned int                OutPoint ;
                                        unsigned int                SigWord[SigBufSize] ;
                                        struct SignatureIndex       Index ;
                                } ;

struct SigBuffer                        SigBuf ;
```

Ensuite, pour accéder au membre *i_Channel* de la structure *SignatureIndex* qui est imbriquée dans la structure SigBuffer nommée *SigBuf*, la chaîne de symboles suivante doit être utilisée :

SigBuf._SignatureIndex._i_Channel

Notez le " _ " qui précède le nom de la structure, ainsi que le nom de chaque membre.
Note : L'accès symbolique à un élément particulier du tableau i_Channel n'est pas autorisé. Cependant, toutes les Vis d'accès à la mémoire permettent de spécifier un décalage. Ce décalage d'adresse d'octet est simplement ajouté à l'adresse de base calculée à partir du nom symbolique.

Toutes les Vis d'accès à la mémoire permettent de spécifier un décalage pour l'accès. Ce décalage d'adresse d'octet est simplement ajouté à l'adresse de base spécifiée explicitement (si l'accès symbolique est désactivé) ou résolu à partir du nom symbolique spécifié. Ce décalage peut être utilisé pour accéder à un élément particulier d'un tableau. Toutefois, il faut tenir compte du fait que ce décalage est toujours exprimé en nombre d'adresses-octets, plutôt qu'en nombre d'éléments du type spécifié pour l'accès. Cela signifie que pour les accès de type 16 bits, le décalage doit être spécifié comme un multiple de 2. Pour les types 32 bits, le décalage doit être spécifié comme un multiple de 4. Ce comportement indépendant du type est nécessaire pour permettre la plus grande flexibilité lors de l'accès à des tableaux de type indéfini (par exemple, des tableaux de structures).

Spécification d'adresse

Contrairement à la plate-forme Signal *Ranger* précédente, la nouvelle plate-forme *Signal_Ranger_mk2* (TMS320VC5502) et son EMIF prennent en charge différentes normes d'adressage, notamment l'octet, le mot de 16 bits et le mot de 32 bits.
Pour des raisons de simplicité et de cohérence, et parce que toutes les adresses décrites dans le fichier ".out" d'un DSP sont des adresses en octets, tous les logiciels d'interface fonctionnent avec le **standard d'adressage en octets.** Cela signifie que :

- Toutes les adresses fournies par le fichier ".out" sont des adresses d'octets. En particulier, les adresses fournies par le VI *SR2_Base_Resolve_UserSymbol* sont des adresses-octets.
- Toutes les adresses symboliques fournies à tous les Vis d'accès (*SR2_Base_K_Exec, SR2_Base_Bulk_Move_Offset, SR2_Base_User_Move_Offset, SR2_Base_HPI_Move_Offset, SR2_Flash_FlashMove*) représentent et pointent vers des adresses d'octets.
- Les adresses explicites fournies en externe à tous les Vis d'accès (*SR2_Base_K_Exec, SR2_Base_Bulk_Move_Offset, SR2_Base_User_Move_Offset, SR2_Base_HPI_Move_Offset, SR2_Flash_FlashMove*) doivent être des adresses d'octets.

Cependant, il est important de noter que

- En raison des contraintes matérielles de l'IPH, tous les accès au PC (lecture et écriture) sont effectués par mots de 16 bits (2 octets à la fois). Cela signifie que les fonctions d'accès qui transfèrent (lisent ou écrivent) des octets individuels lisent ou écrivent toujours un nombre pair d'octets. Les tableaux d'octets sont complétés si nécessaire.
- Le téléchargement du code utilisateur est également soumis à cette contrainte puisqu'il est effectué à l'aide des mêmes fonctions que celles qui lisent et écrivent les données. Cela ne devrait pas poser de problème lors du développement en C car le compilateur crée toujours des sections alignées sur une adresse paire d'octets. Lors du développement en assembleur, la directive ".even" doit être utilisée systématiquement au début de toute section de code, afin de s'assurer que les sections de code sont alignées sur des adresses d'octets paires.
- En raison des mêmes contraintes matérielles, les accès en lecture et en écriture doivent toujours se produire sur des adresses d'octets paires. Si une adresse d'octet impaire est transmise à un VI de transfert, elle est tronquée de sorte que l'adresse de transfert effective est l'adresse d'octet paire précédente. Cependant, cela ne devrait jamais poser de problème car les données sont toujours alignées sur une adresse paire dans la mémoire du DSP par l'éditeur de liens.
- Les points d'entrée ne sont pas soumis à cette contrainte, de sorte que l'adresse d'exécution transmise au VI *SR2_Base_K_Exec* peut être une adresse d'octet impaire. De même, l'*adresse de branchement* transmise au *SR2_Base_User_Move_Offset* peut être une adresse impaire.
- La plupart des Vis d'interface qui effectuent des transferts (*SR2_Base_Bulk_Move_Offset, SR2_Base_User_Move_Offset, SR2_Base_HPI_Move_Offset*) permettent l'ajout d'un décalage à l'adresse de transfert. Tout comme l'adresse, ce décalage est également un décalage d'adresse d'octet. Tout comme l'adresse, le décalage doit être pair.
- Bien que les adresses de transfert soient toujours des adresses d'octets, le nombre d'éléments à transférer vers et depuis le PC est toujours spécifié en nombre d'éléments **du type à transférer**.

Interface LabView

L'interface LabView fournie est organisée en plusieurs bibliothèques VI. 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_mk2*, avec la facilité d'utilisation, la puissance de traitement de haut niveau et l'interface utilisateur graphique de LabView.

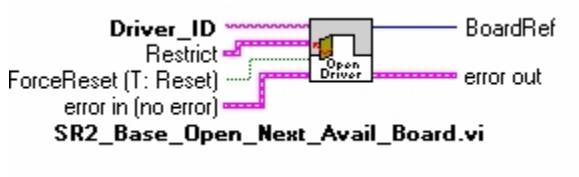
Interface principale Vis

Ces Vis se trouvent dans la bibliothèque *Sranger2.llb.*

SR2_Base_Open_Next_Avail_Board

Cette Vi effectue les opérations suivantes :

- Essaie de trouver une carte DSP avec l'ID de pilote sélectionné qui est connectée, mais actuellement libre (non ouverte), sur le PC.
- 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 "ForceReset" est vrai, la réinitialisation du DSP est forcée, puis le noyau de téléchargement de l'hôte est rechargé.
- Place la table des symboles du noyau actuel dans la structure "Global Board Info".



Contrôles :

- **Driver ID** : Il s'agit d'une chaîne de caractères représentant le nom de base de la carte sélectionnée. Par exemple, pour les cartes *Signal Ranger_mk2*, cette chaîne doit être définie comme *SRm2_*.
- **Restreindre** : Il s'agit d'une structure utilisée pour restreindre l'accès par ID de fournisseur, ID de produit ou numéro de révision du matériel. Si ce contrôle est câblé et si l'un des membres de la structure Restrict est modifié par rapport à sa valeur par défaut, l'accès est limité aux cartes ayant les paramètres sélectionnés. Chaque membre fonctionne indépendamment des autres. Par exemple, si HardRev est défini, mais que idVendor et idProduct sont laissés à leur valeur par défaut de 0, seules les cartes DSP avec la révision matérielle sélectionnée seront ouvertes, indépendamment de leur numéro de fournisseur ou de produit.
- **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.
- **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*. Toutes les autres interfaces Vis utilisent ce numéro pour accéder à la carte appropriée.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabView. Contient le numéro et la description de l'erreur.

Remarque : La poignée que le pilote fournit pour accéder à la carte est exclusive. Cela signifie qu'une seule application ou un seul processus à la fois peut ouvrir et gérer une carte. 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 SR2_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 SR2_Base_Close_BoardNb. Ceci est particulièrement préoccupant lorsque l'application gérant

le tableau est fermé dans des conditions anormales. Si l'application est fermée sans fermer correctement la carte. L'exécution suivante de l'application ne parviendra pas à trouver et à ouvrir la carte, simplement parce que l'instance de pilote correspondante est encore ouverte.

SR2_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 dernier accès à la carte, pour libérer les ressources Windows qui ne sont plus utilisées.

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 SR2_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 SR2_Base_Open_Next_Avail_Board.vi Utilisez 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.

SR2_Base_Get_BoardInfo

Ce Vi renvoie l'entrée de la carte de la *structure d'information globale de la carte*.

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 du conseil global*. Elle est créée par SR2_Base_Open_Next_Avail_Board.vi

Indicators :

- **BoardInfo** : Contient des informations sur la carte DSP. Ces informations sont décrites dans la section "Comment les cartes DSP sont gérées"

SR2_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 SR2_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.

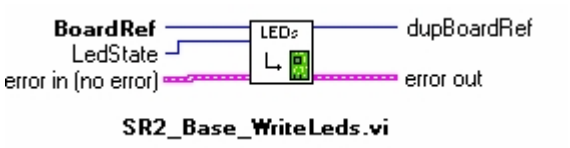
Indicateurs :

- **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 SR2_Base_Open_Next_Avail_Board.vi Utilisez 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.

SR2_Base_WriteLeds

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

- Arrêt
- Rouge
- Vert
- Orange



Contrôles :

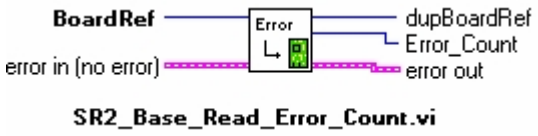
- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par SR2_Base_Open_Next_Avail_Board.vi
- **LedState** : Cette énumération spécifie l'état des DEL (rouge, vert, orange ou éteint).
- **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 SR2_Base_Open_Next_Avail_Board.vi Utilisez 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.

SR2_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 d'un 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.



Contrôles :

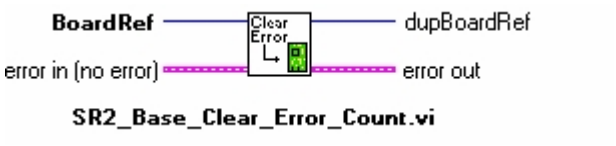
- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par SR2_Base_Open_Next_Avail_Board.vi
- **Erreur dans** : Grappe d'erreurs de type instrument LabView. Contient le numéro d'erreur 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 SR2_Base_Open_Next_Avail_Board.vi Utilisez 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.

SR2_Base_Clear_Error_Count

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



Contrôles :

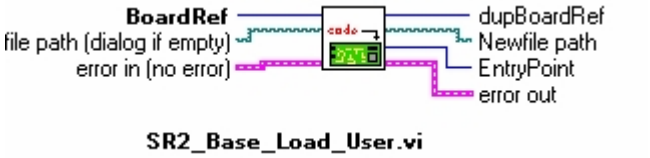
- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par SR2_Base_Open_Next_Avail_Board.vi
- **Erreur dans** : Grappe d'erreurs de type instrument LabView. Contient le numéro d'erreur 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 SR2_Base_Open_Next_Avail_Board.vi Utilisez 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.

SR2_Base_Load_User

Ce VI charge un code DSP utilisateur dans la mémoire du DSP. Si *le chemin du fichier* est vide, une boîte de dialogue est utilisée. Le noyau doit être chargé avant l'exécution de ce VI. Le DSP est réinitialisé avant le chargement. Après le chargement du code, la table des symboles est mise à jour dans la *structure d'information globale de la carte*.
Le VI vérifie si le type de fichier COFF est adapté au DSP cible. Si ce n'est pas le cas, une erreur est générée.



Contrôles :

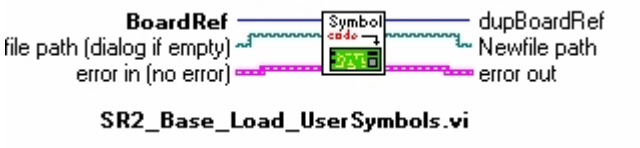
- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du *Structure d'information de la carte globale*. Elle est créée par SR2_Base_Open_Next_Avail_Board.vi
- **Chemin d'accès au fichier** : Il s'agit du chemin d'accès au fichier COFF (.out) du code utilisateur DSP. Une boîte de dialogue est présentée si le chemin est vide.
- **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 SR2_Base_Open_Next_Avail_Board.vi Utilisez cette sortie pour propager le numéro de référence à d'autres Vis.
- **NewFile path** : Il s'agit du chemin d'accès fichier COFF.
- **Point d'entrée** : Il s'agit de l'adresse dans la mémoire du DSP où l'exécution doit commencer.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabView. Contient le numéro et la description de l'erreur.

SR2_Base_Load_UserSymbols

Ce VI charge la table des symboles dans la *structure d'information globale de la carte*. Si *le chemin du fichier* est vide, une boîte de dialogue est utilisée. En général, ce VI est utilisé pour obtenir un accès symbolique lorsque le code est déjà chargé et s'exécute sur le DSP (par exemple le code qui a été chargé à la mise sous tension). Il n'est pas nécessaire de charger les symboles après avoir exécuté SR2_Base_Load_User, ou SR2_Base_LoadExec_User, puisque les deux Vis mettent à jour la table des symboles automatiquement.
Le VI vérifie si le type de fichier COFF est adapté au DSP cible. Si ce n'est pas le cas, une erreur est générée.



Contrôles :

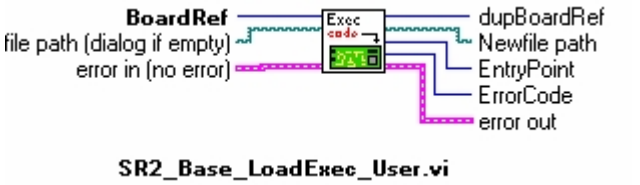
- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du
  Structure d'information de la carte globale. Elle est créée par SR2_Base_Open_Next_Avail_Board.vi
- **Chemin d'accès au fichier** : Il s'agit du chemin d'accès au fichier COFF (.out) du code utilisateur DSP. Une boîte de dialogue est présentée si le chemin est vide.
- **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 SR2_Base_Open_Next_Avail_Board.vi Utilisez cette sortie pour propager le numéro de référence à d'autres Vis.
- **NewFile path** : Il s'agit du chemin d'accès fichier COFF.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabView. Contient le numéro et la description de l'erreur.

SR2_Base_LoadExec_User

Ce VI 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 *le chemin du fichier* est vide, une boîte de dialogue est utilisée. Le noyau doit être chargé avant l'exécution de ce VI. 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*.
Le VI 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 complété la branche, le contrôleur USB et le VI attendent un accusé de réception de la part du DSP, pour terminer son exécution. Si ce signal ne se produit pas dans les 5 secondes, la Vi abandonne et renvoie une erreur. Normalement, le code DSP lancé par cette Vi doit accuser réception de la branche 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 SR2_Base_Open_Next_Avail_Board.vi
- **Chemin d'accès au fichier** : Il s'agit du chemin d'accès au fichier COFF (.out) du code utilisateur DSP. Une boîte de dialogue est présentée si le chemin est vide.
- **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 SR2_Base_Open_Next_Avail_Board.vi Utilisez cette sortie pour propager le numéro de référence à d'autres Vis.
- **NewFile path** : Il s'agit du chemin d'accès fichier COFF.
- **Point d'entrée** : Il s'agit de l'adresse dans la mémoire du DSP où l'exécution doit commencer.
- **ErrorCode** : Il s'agit du code d'erreur, ou code d'achèvement, renvoyé par la fonction DSP utilisateur exécutée (fonction résidant au point d'entrée). La documentation du noyau mentionne que la fonction DSP appelée (la fonction d'entrée dans ce cas) doit contenir un accusé de réception pour signaler que la branche a été prise. Juste avant d'envoyer l'accusé de réception, le code DSP de l'utilisateur a la possibilité de modifier le champ ErrorCode dans la boîte aux lettres. Ce code d'erreur est renvoyé au PC par le contrôleur USB après avoir reçu l'accusé de réception du DSP. Ce code d'erreur est totalement spécifique à l'application de l'utilisateur. Il ne doit pas être géré par l'interface. Si le code DSP ne modifie pas le code d'erreur, zéro est renvoyé.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabView. Contient le numéro et la description de l'erreur.

SR2_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.
Le noyau doit être chargé et exécuté pour que cette Vi soit fonctionnelle.
Après avoir effectué la dérivation, le contrôleur USB et la Vi attendent un accusé de réception du DSP pour terminer leur exécution. Si ce signal ne se produit pas dans les 5 secondes, l'interface Vi s'interrompt et renvoie une erreur. Normalement, le code DSP lancé par cette Vi doit accuser réception de la branche 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 base de données du *Structure d'information de la carte globale*. Elle est créée par SR2_Base_Open_Next_Avail_Board.vi
- **DSPAddress** : Adresse physique de la branche. Elle est utilisée pour la branche si *Symbol* ou *SymbolTable* sont vides ou ne sont pas câblés.
- **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 SR2_Base_Open_Next_Avail_Board.vi Utilisez cette sortie pour propager le numéro de référence à d'autres Vis.
- **ErrorCode** : Il s'agit du code d'erreur, ou code d'achèvement, renvoyé par la fonction DSP utilisateur qui est exécutée. La documentation du noyau mentionne que la fonction DSP appelée doit contenir un acquittement pour signaler que la branche a été prise. Juste avant d'envoyer l'accusé de réception, le code DSP utilisateur a la possibilité de modifier le champ ErrorCode dans la boîte aux lettres. Ce code d'erreur est renvoyé au PC par le contrôleur USB après avoir reçu l'accusé de réception du DSP. Ce code d'erreur est totalement spécifique à l'application de l'utilisateur. Il ne doit pas être géré par l'interface. Si le code DSP ne modifie pas le code d'erreur, zéro est renvoyé.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabView. Contient le numéro et la description de l'erreur.

SR2_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 conduites en vrac (bulk pipes). La bande passante est généralement élevée (jusqu'à 22 Mb/s pour USB 2).
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 U16 du côté du DSP (cast du type requis vers un tableau de U16 pour l'écrire au DSP, lecture d'un tableau de U16 et cast vers le 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 de programmation= page numéro 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* n'est pas câblé, *DSPAddress* est utilisé comme adresse d'octet pour le transfert, et *MemSpace* est utilisé comme espace mémoire. Notez que *DSPAddress* doit être paire, afin de pointer vers un mot de 16 bits valide (voir la section sur la spécification de l'adresse).
- 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 (tout comme *DSPAddress*, il s'agit d'un décalage d'adresse en octets, qui doit être pair). Ceci est nécessaire pour accéder à un membre individuel d'une structure hétérogène.
- Dans le cas d'une écriture, si le type de données accédées est un type 8 bits (I8, U8 ou chaîne), un octet supplémentaire est ajouté si le nombre d'octets à transférer est impair. Cela est nécessaire car les transferts natifs ont une largeur de 16 bits. L'octet supplémentaire est mis à FF_H.

- Même si le type de données accédées est un type de 8 bits (I8, U8 ou chaîne de caractères), l'accès se faire à une adresse d'octet pair. En effet, les transferts natifs utilisant l'interface HPI ont une largeur de 16 bits.

Note : Le noyau doit être chargé et s'exécuter pour que cette Vi soit fonctionnelle :Le noyau doit être chargé et exécuté pour que cette Vi soit fonctionnelle.

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.

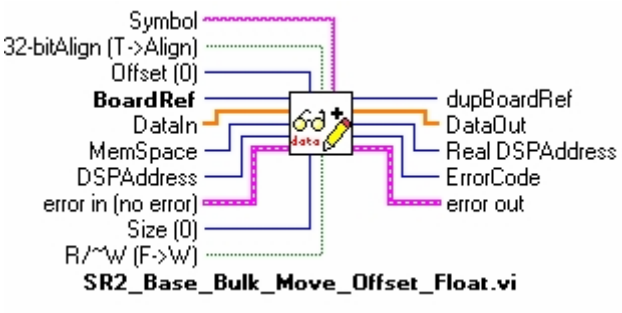
Remarque : Lors de la lecture ou de l'écriture de scalaires, la taille ne doit pas être câblée.

La représentation interne du DSP utilise des 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.
Lors de la lecture ou de l'écriture de mots de données de 32 bits (I32, U32 ou Float), le PC effectue 2 accès séparés (à 2 adresses mémoire successives) pour chaque mot de 32 bits 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, le PC 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 le PC serait complètement erronée.
Symétriquement, lors d'une écriture, le PC 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 DSP travaille avec une "ancienne" version de la moitié du flottant et une nouvelle version de l'autre moitié.
Ces problèmes peuvent être évités si les précautions suivantes sont respectées :

- Lorsque le PC accède à un groupe de valeurs, il le fait par blocs de 32 mots de 16 bits maximum à la fois (jusqu'à 256 mots si la carte a été énumérée sur un concentrateur ou une racine compatible USB_2). Chacun 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 DSP concurrente 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 par exemple, les accès au PC sont alors atomiques des deux côtés, et les données peuvent être transférées en toute sécurité 32 bits à la fois.

Remarque : Les données sont transférées entre le PC et le DSP par blocs atomiques de 32 (connexion USB à grande vitesse) ou 256 (connexion USB à grande vitesse) mots. Pour ce , la fonction du noyau qui effectue le transfert par blocs désactive les interruptions pendant le transfert. Le temps nécessaire pour transférer 32 ou 256 mots de données dans la mémoire vive du DSP est généralement très court (3,3ns/mot). Cependant, il peut être beaucoup plus long (jusqu'à 240ns/mot) si le transfert est effectué vers ou depuis la SDRAM. Dans ce cas, le temps de transfert peut être si long qu'il interfère avec le service d'autres interruptions dans le système. Ce facteur peut être particulièrement important lorsque la connexion USB est à grande vitesse, car dans ce cas, la taille du bloc est importante (256 mots). Si c'est le cas, et que l'atomicité du transfert n'est pas requise, une fonction DSP personnalisée doit être utilisée pour gérer le transfert, au lieu de la fonction standard du noyau qui est instanciée par SR2_Base_Bulk_Move_Offset. Une telle fonction de transfert personnalisée peut être appelée par SR2_Base_User_Move_Offset. Un exemple d'utilisation d'une telle fonction de transfert personnalisée est fourni dans le répertoire des exemples. Une autre façon de traiter ce cas serait de diviser le transfert à haut niveau de sorte que seuls de petits blocs soient transférés à la fois.

Remarque : Les données sont transférées entre le PC et le DSP par blocs atomiques de 32 (connexion USB à grande vitesse) ou 256 (connexion USB à grande vitesse) mots, uniquement si le transfert ne traverse pas une limite de 64k mots. Si le transfert traverse une limite, il est divisé à haut niveau de sorte que les deux moitiés du transfert se produisent dans la même page de 64k. Dans ce cas, le transfert perd son "atomicité".



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 SR2_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.
- **32-bitAlign** : Ce contrôle est fourni pour des raisons de compatibilité avec d'autres cartes. Sur Signal Ranger_mk2, ce contrôle n'a aucun effet tant que l'adresse de l'octet fournie est paire (comme doit l'être).
- **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. 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 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 SR2_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 . 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. Les lectures du noyau renvoient toujours un code d'achèvement de 1, les écritures du noyau renvoient toujours un code d'achèvement de 2.

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

SR2_Base_User_Move_Offset

Ce VI est similaire à SR2_Base_Bulk_Move_Offset, sauf qu'il permet à une fonction DSP définie par l'utilisateur de remplacer la fonction noyau intrinsèque utilisée par SR2_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 ce faire, 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, SR2_Base_User_Move_Offset comprend deux contrôles pour définir le point d'entrée de fonction qui doit être utilisée pour remplacer la fonction noyau intrinsèque.
Lorsque la fonction définie par l'utilisateur est appelée, la boîte aux lettres contient :

- Le champ BranchAddress correspond au point d'entrée de la fonction. Ce champ n'est normalement pas utilisé directement par la fonction DSP. Il est fixé à son point d'entrée, ce qui conduit à son exécution.
- Le champ TransferAddress. Il contient l'adresse correspondant à Symbol, si Symbol est utilisé, ou le nombre de 32 bits défini par l'utilisateur DSPAddress si Symbol n'est pas câblé ou est vide. Notez que ce champ n'est pas obligé de contenir une adresse valide. Par exemple, dans le cas d'une fonction de gestion FIFO, il peut s'agir d'un numéro FIFO.
- Le champ NbWords. Ce nombre de 16 bits représente le nombre de mots à transférer. Il est toujours compris entre 1 et 32 (connexion USB à pleine vitesse), ou entre 1 et 256 (connexion USB à haute vitesse). Il représente la taille du champ de données de la boîte aux lettres. Les transferts de blocs de données plus importants sont segmentés en transferts de 32 ou 256 mots maximum.
- Le champ ErrorCode. Il vaut 1 pour les lectures et 2 pour les écritures.
- Dans le cas d'un transfert en écriture, de 1 à 32 mots, comme indiqué par NbWords, (1 à 256 mots pour une connexion USB à grande vitesse) ont été écrits dans le champ de données de la boîte aux lettres par le contrôleur USB avant l'appel de la fonction DSP.

La fonction définie par l'utilisateur doit exécuter sa fonction. S'il s'agit d'une lecture, elle doit lire le nombre de mots requis dans le champ de données de la boîte aux lettres. Elle doit ensuite mettre à jour la boîte aux lettres avec les données suivantes :

- S'il s'agit d'une écriture, elle doit fournir le nombre de mots requis dans le champ de données de la boîte aux lettres.
- Il peut ensuite mettre à jour le champ ErrorCode de la boîte aux lettres avec un code d'achèvement approprié à la situation (c'est le code d'erreur qui est renvoyé en tant qu'indicateur ErrorCode du VI).

Ensuite, la fonction définie par l'utilisateur doit simplement envoyer un accusé de réception.

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.

Le champ *TransferAddress* de la boîte aux lettres n'est initialisé qu'au premier segment. La fonction définie par l'utilisateur peut choisir de l'incrémenter pour pointer vers des adresses successives (c'est ce que font les fonctions intrinsèques du noyau), ou de le laisser intact (ce qui serait approprié si le champ contenait un numéro FIFO par exemple). La manière dont ce champ est géré est totalement spécifique à l'application.

Remarque : 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 n'est conservé 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.

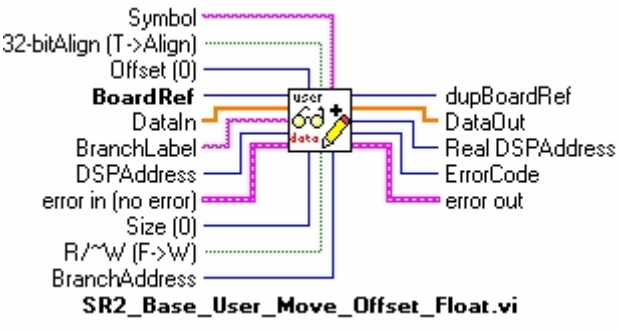
- *Le transfert ne doit pas franchir une limite de 64 kWord. Les transferts qui franchissent une limite de 64 kWord sont divisés en deux transferts consécutifs. Les informations contenues dans TransferAddress ne sont conservées que pendant le premier de ces segments de niveau supérieur. Au suivant, TransferAddress 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.*

Le champ NbWords est initialisé avec la taille du segment transféré, à chaque segment. Le champ ErrorCode n'est initialisé qu'au premier segment avec la valeur 1 (lecture) ou 2 (écriture).
La valeur renvoyée à l'indicateur ErrorCode du VI est la valeur qui peut être mise à jour par la fonction définie par l'utilisateur avant d'acquitter le segment LAST. Si la fonction définie par l'utilisateur ne met pas à jour le code d'erreur, la même valeur (1 pour les lectures et 2 pour les écritures) est renvoyée au PC.

Remarque : Le noyau ET le code DSP de la fonction définie par l'utilisateur doivent être chargés et exécutés pour que cette Vi soit fonctionnelle.

Note : Le contrôle MemSpace du VI n'est pas utilisé : Le contrôle MemSpace du VI n'est pas utilisé. La fonction définie par l'utilisateur effectue le type d'accès codé dans la fonction, quelle que soit la valeur de MemSpace.

Note : La valeur de l'indicateur R/~W est reflétée par le contenu du champ ErrorCode de boîte aux lettres à l'entrée de la fonction définie par l'utilisateur. Pour les lectures, la valeur est 1, pour les écritures, la valeur est 2.



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 SR2_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.
- **Espace mémoire** : Non utilisé
- **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).
- **32-bitAlign** : Ce contrôle est fourni pour des raisons de compatibilité avec d'autres cartes. Sur Signal Ranger_mk2, ce contrôle n'a aucun effet tant que l'adresse de l'octet fournie est paire (comme doit l'être).

- **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 DSP. 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.

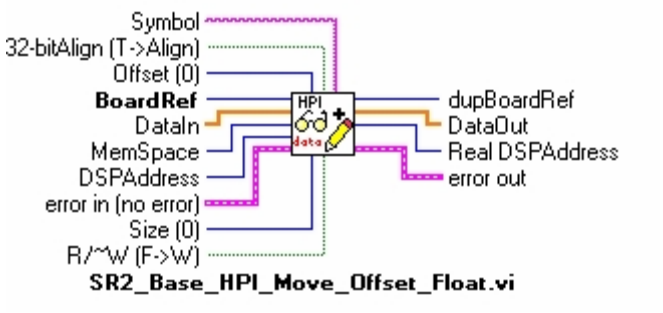
Indicateurs :

- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par SR2_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 . Cette adresse tient compte de la résolution du *symbole* (s'il est utilisé) et de l'effet du *décalage*.
- **ErrorCode** : La valeur renvoyée par l'indicateur ErrorCode est la valeur mise à jour par la fonction DSP définie par l'utilisateur avant d'acquitter le DERNIER segment transféré. Si la fonction utilisateur ne met pas à jour le code d'erreur, la fonction renvoie 1 pour les lectures et 2 pour les écritures.
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabView. Contient le numéro et la description de l'erreur.

SR2_Base_HPI_Move_Offset

Ce VI est similaire à SR2_Base_Bulk_Move_Offset, sauf qu'il s'appuie sur le matériel de 'IPH, plutôt que sur le noyau pour effectuer le transfert.
Les transferts sont limités à la RAM sur puce directement accessible via l'interface HPI (adresses d'octets $00C0_H$à $FFFF_{H)}$. Le contrôle MemSpace n'est pas utilisé. Ce VI effectuera 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 l'interface HPI, quel que soit l'espace mémoire. L'espace E/S sur la puce n'est pas accessible. Si l'on tente d'écrire des données en dehors de la plage autorisée, les données ne sont pas écrites. Si l'on tente de lire des données en dehors de la plage autorisée, des données erronées sont renvoyées.

Note : Le noyau n'a pas besoin d'être chargé ou fonctionnel pour que ce VI soit exécuté correctement : 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 SR2_Base_Bulk_Move_Offset. La bande passante pour ces transferts est généralement faible (500kb/s pour USB 1.1). 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 SR2_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.

Note : la sélection de l'espace mémoire n'est pas utilisée. Le transfert réel utilise le matériel de l'IPH, indépendamment de la sélection de l'espace mémoire.

- **DSPAddress** : Adresse physique de base du DSP pour l'échange. DSPAddress n'est utilisé que si le symbole est vide ou non câblé.
- **32-bitAlign** : Ce contrôle est fourni pour des raisons de compatibilité avec d'autres cartes. Sur Signal Ranger_mk2, ce contrôle n'a aucun effet tant que l'adresse de l'octet fournie est paire (comme doit l'être).
- **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 de l'erreur précédente.

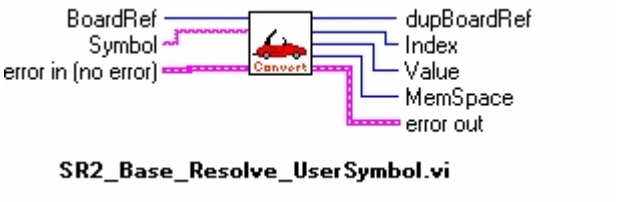
Indicateurs :

- **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 SR2_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 . 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.

SR2_Base_Resolve_UserSymbol

Ce Vi peut être utilisé pour fournir l'adresse correspondant à un symbole particulier dans la table des symboles du code DSP actuellement chargé. Utilisée en conjonction avec les Vis de transfert de données (SR2_Base_Bulk_Move_Offset, SR2_Base_User_Move_Offset et SR2_Base_HPI_Move_Offset), elle permet une plus grande flexibilité dans le choix de l'adresse de transfert.

Par exemple, l'adresse peut être transformée d'une manière spécifique à l'application avant le transfert.



Contrôles :

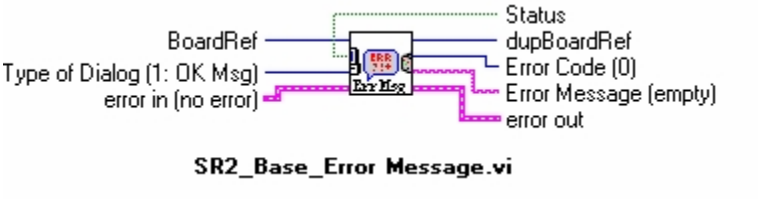
- **BoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la base de données du
  Structure d'information de la carte globale. Elle est créée par SR2_Base_Open_Next_Avail_Board.vi
- **Symbole** : Chaîne de caractères du symbole auquel il faut accéder.
- **Erreur dans** : 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 SR2_Base_Open_Next_Avail_Board.vi Utilisez cette sortie pour propager le numéro de référence à d'autres Vis.
- **Index** : Il s'agit de l'index du symbole dans la table des symboles.
- **Valeur** : Il s'agit de la valeur de résolution du symbole.
- **MemSpace** : Il s'agit de l'espace mémoire du symbole. Notez que cette valeur reflète en fait le "numéro de page" qui a été utilisé au moment de la liaison pour ce symbole. Pour que ce numéro indique un espace mémoire valide, le symbole doit être lié selon la convention habituelle :
  - 0 -> Espace de programmation
  - 1 -> Espace de données
  - 2 -> Espace IO
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabView. Contient le numéro et la description de l'erreur.

SR2_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 SR2_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 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 SR2_Base_Open_Next_Avail_Board.vi Utilisez cette sortie pour propager le numéro de référence à d'autres Vis.
- **Code d'erreur** : Indique le code d'erreur.

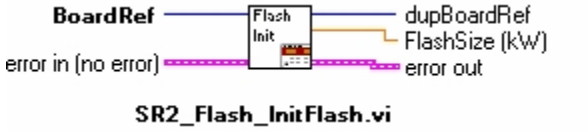
- **Message d'erreur** : Fournit une explication textuelle de l'erreur, le cas échéant.
- **Status** : Booléen, indique s'il y a une erreur (True) ou non (False).
- **Sortie d'erreur** : Grappe d'erreurs de type instrument LabView. Contient le numéro et la description de l'erreur.

Support Flash Vis

Ces Vis se trouvent dans la bibliothèque *SR2_Flash.llb.*
Ces Vis sont fournies pour soutenir les opérations de programmation Flash. Les Vis couvrent également les opérations de lecture de la Flash pour des raisons de symétrie. Cependant, le VI SR2_Base_Bulk_Move_Offset peut parfaitement utilisé pour lire la Flash, et ne nécessite pas la présence d'un code de support Flash.

SR2_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 code de support Flash doit être exécuté en plus du noyau pour supporter la programmation Flash Vis. 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.



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 SR2_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 :

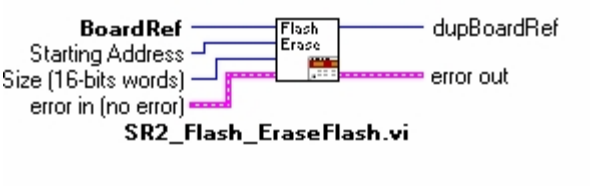
- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par SR2_Base_Open_Next_Avail_Board.vi Utilisez cette sortie pour propager le numéro de référence à d'autres Vis.
- **FlashSize** : Cet indicateur indique 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.

SR2_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 précédant l'adresse de départ seront effacés jusqu'au début de la section. De même, si le dernier mot sélectionné pour l'effacement n'est pas le dernier mot d'une section, 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.

Note : La taille du secteur est de 32 mots : La taille du secteur est de 32 mots-clés.

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 SR2_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 SR2_Base_Open_Next_Avail_Board.vi Utilisez 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.

SR2_Flash_FlashMove

Ce VI lit ou écrit un nombre illimité de mots de données vers/depuis mémoire Flash. Notez que si seule la lecture de la mémoire Flash est nécessaire, le VI SR2_Base_Bulk_Move_Offset doit être utilisé à la place (en spécifiant la mémoire programme), puisqu'il ne nécessite pas la présence du code de support Flash.

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 de 32 bits non signés (U32) ou tableaux de ce type.
- des nombres à virgule flottante de 32 bits (float) ou des 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 U16 du côté du DSP (cast du type requis vers un tableau de U16 pour l'écrire au DSP, lecture d'un tableau de U16 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 , 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.

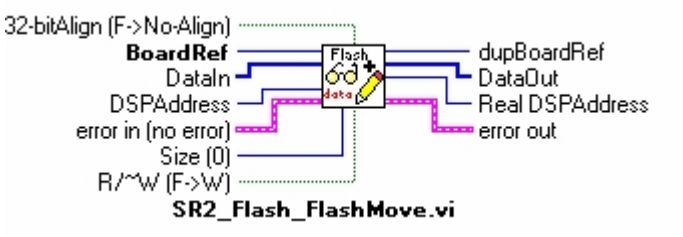
Remarque : Contrairement aux générations précédentes de dispositifs Flash utilisés dans les cartes Signal Ranger, le dispositif Flash utilisé dans Signal_Ranger_mk2 ne peut pas être programmé de manière incrémentielle. Cela signifie qu'un emplacement de mot qui a été précédemment programmé DOIT être effacé avant d'être reprogrammé. Cela est vrai même si l'opération de reprogrammation ne vise qu'à transformer certains des "1" restants en "0".

Dans le cas d'une écriture, si le type de données accédées est un type 8 bits (I8, U8 ou chaîne), un octet supplémentaire est ajouté si le nombre d'octets à transférer est impair. Cela est nécessaire car les transferts natifs ont une largeur de 16 bits. L'octet supplémentaire est mis à 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 du DSP utilise des 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 SR2_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.
- **32-bitAlign** : Ce contrôle est fourni pour des raisons de compatibilité avec d'autres cartes. Sur Signal Ranger_mk2, ce contrôle n'a aucun effet tant que l'adresse de l'octet fournie est paire (comme doit l'être).
- **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 :

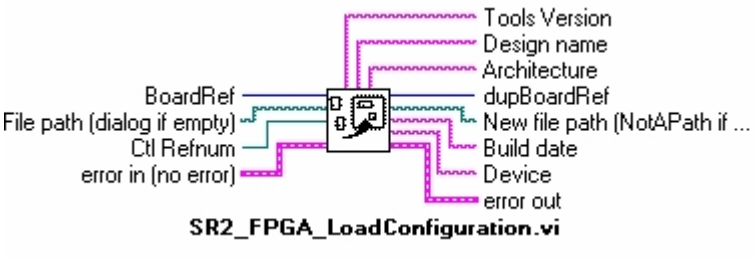
- **DupBoardRef** : Il s'agit d'un numéro qui pointe vers l'entrée correspondant à la carte dans la *structure d'information globale de la carte*. Il est créé par SR2_Base_Open_Next_Avail_Board.vi Utilisez cette sortie pour propager le numéro de référence à d'autres Vis.
- **DataOut** : Données lues à partir de la mémoire Flash.
- **Real DSPAddress** : Adresse réelle où le transfert a eu . 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.

Support FPGA Vis

Ces Vis se trouvent dans la bibliothèque *SR2_FPGA.llb*.

SR2_FPGA_LoadConfiguration

Ce Vi télécharge un fichier de configuration logique "*.rbt" dans le FPGA. Le DSP est réinitialisé avant 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.



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 SR2_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.
- **Ctl Refnum** : Il s'agit d'un numéro de référence sur la barre de progression qui est mis 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 SR2_Base_Open_Next_Avail_Board.vi Utilisez 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" décrivant la logique FPGA.
- **Version des outils** : Chaîne ASCII indiquant la version des outils utilisés pour générer le 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** : 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.

Codes d'erreur

Le tableau suivant liste les codes d'erreur qui peuvent être renvoyés par les différentes Vis de l'interface LabVIEW.

Note : La liste n'est pas exhaustive : La liste n'est pas exhaustive. Seuls les codes les plus courants y figurent.

Erreur Nb	Cause
00000002h	La mémoire est pleine
00000003h	Accès à une mauvaise zone de mémoire
00000004h	Fin de fichier rencontrée
00000005h	Fichier déjà ouvert
00000006h	Erreur d'E/S de fichier
00000007h	Fichier non trouvé
00000008h	Erreur de permission de fichier
00000009h	Disque plein
0000000Ah	Chemin d'accès en double
0000000Bh	Trop de fichiers ouverts
BFFC0804h	Aucune carte détectée
BFFC0805h	Erreur de génération de carte ouverte

BFFC0806h	Les informations de la carte génèrent une erreur
BFFC0807h	La carte de fermeture génère une erreur
BFFC0808h	Fichier DSP non accessible
BFFC0809h	Réinitialiser l'erreur générée par le DSP
BFFC0812h	Ce noyau DSP n'est pas pour le c5000
BFFC080Ah	Unreset_DSP genrate error
BFFC080Bh	Fichier DSP introuvable
BFFC080Ch	Fichier DSP non valide
BFFC080Dh	Le type de fichier DSP n'est pas pris en charge
BFFC080Eh	Le noyau DSP ne compare pas
BFFC080Fh	Write_hpi génère une erreur
BFFC0810h	Read_hpi génère une erreur
BFFC0811h	Ce fichier DSP n'est pas pour le c5000
BFFC0813h	Erreur de test d'écriture et de lecture
BFFC0814h	Le DSP n'est pas revenu... il est peut-être tombé en panne.
BFFC0815h	Symbole non trouvé dans le tableau
BFFC0816h	DSPInt génère une erreur
BFFC0817h	Délai d'attente DSP
BFFC0818h	Le noyau n'est pas chargé
BFFC0820h	Erreur de fermeture du pilote
3FFC0104h	Erreur Requête non prise en charge
BFFC0821h	Erreur d'ouverture du pilote
BFFC0822h	Le déplacement de données avec le noyau génère une erreur
BFFC0823h	Le niveau bas de l'exécution génère une erreur
BFFC0824h	L'obtention d'informations sur les tuyaux génère des erreurs
BFFC0825h	Réinitialisation Le DSP génère une erreur
BFFC0826h	Le déménagement de l'IPH génère une erreur
BFFC0827h	DSP int génère une erreur
BFFC0828h	Erreur dans le descripteur de configuration
BFFC0829h	L'écriture de Led génère une erreur
BFFC082Ah	Erreur de descripteur de périphérique
BFFC082Bh	Erreur de contrôle HPI en lecture et en écriture
BFFC082Ch	Modifier l'erreur du mode Timeout de l'USB
BFFC082Dh	La section du code DSP ne vérifie pas
BFFC082Eh	Fichier FPGA non valide
BFFC082Fh	Configuration du FPGA interrompue avant la fin
BFFC0830h	Erreur d'accès au pilote
BFFC0831h	Famille DSP non reconnue ou carte non ouverte
BFFC0832h	Le fichier FPGA ne correspond pas à la famille de dispositifs cible
BFFC0833h	Section trop grande pour être affichée dans Flash
BFFC0834h	Le fichier DSP ne peut pas être lié
BFFC0835h	Section non alignée sur une adresse d'octets pairs
BFFC0836h	Erreur d'effacement de la mémoire flash
BFFC0837h	Erreur d'écriture Flash

Exemple de code

Des exemples sont fournis pour accélérer le développement de l'utilisateur. Ces exemples comprennent le côté PC et le côté DSP du code. Du côté du DSP, ils comprennent du code écrit en assemblage, ainsi que du code écrit en C. Le côté LabVIEW de ces exemples est contenu dans la bibliothèque LabVIEWDemo.llb dans le répertoire d'installation principal. Les exemples côté DSP sont zippés et stockés dans le répertoire suivant : *C:\NProgram Files\NSignalRanger_mk2\NExemples\NLabVIEW_Examples_DSPCode\NLabviewInterfaceDemo.zip.* Il suffit de décompresser *chaque fichier* pour déflater l'exemple correspondant ainsi que la documentation.

Interface C/C++.

Cette interface a été conçue le développement en C/C++ et n'a été testée que sur la version ".net" de Visual Studio de Microsoft. Cependant, il est possible de l'utiliser avec d'autres environnements de développement permettant l'utilisation de DLL.
L'interface C/C++ est fournie sous la forme d'une DLL appelée *SRm2_HL.dll*. Comparée à l' LabVIEW, cette interface offre des fonctionnalités légèrement plus limitées. Cependant, elle couvre l'essentiel.

Les aspects de l'interface C/C++ qui ont été limités par rapport à l'interface LabVIEW sont les suivants :

- L'interface LabVIEW fournit des Vis polymorphes pour échanger des données entre le DSP et PC hôte. La DLL C/C++ ne fournit qu'une seule fonction pour échanger des tableaux de type I16 (court). Cela n'est pas limitatif car, à bas niveau, les transferts sont effectués à l'aide de tableaux de données I16. Pour échanger d'autres types, le développeur doit simplement convertir ces tableaux de données I16 en d'autres types.
- Le VI SR2_Base_Get_BoardInfo dans l'interface LabVIEW n'a pas d'équivalent dans l'interface C/C++.
- Le VI SR2_Base_ErrorMessage dans l'interface LabVIEW n'a pas d'équivalent dans l'interface C/C++.

Pour fonctionner au moment de l'exécution, cette DLL nécessite que les fichiers suivants se trouvent dans le même répertoire que 'application en mode utilisateur qui l'utilise :

- SRm2_HL.dll - La DLL principale supportant l'API
- SRanger2.dll - Une DLL de support de bas niveau
- SR2Kernel_HostDownload.out - Le code du noyau DSP "Host-Download".
- SR2Kernel_PowerUp.out - Le code du noyau DSP "Power-Up".
- SR2_Flash_Support.out - Le code DSP du support Flash
- SR2_FPGA_Support.out - Le code DSP du support FPGA

En outre, le moteur d'exécution LabVIEW 7.1 doit être installé sur l'ordinateur qui doit utiliser la DLL. Le moteur d'exécution LabVIEW 7.1 est installé automatiquement lors de l'installation du logiciel décrite au début de ce document. Toutefois, si l'utilisateur souhaite déployer une application utilisant l'interface C/C++, qui doit fonctionner sur des ordinateurs autres que ceux sur lesquels elle a été développée, le moteur d'exécution LabVIEW 7.1 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.

Un exemple est fourni, qui couvre le développement du code dans Visual C/C++. Cet exemple est discuté à la fin de ce chapitre.

Temps d'exécution et gestion des threads

Deux fonctions de la DLL *SRm2_HL* accédant à **la même** carte *Signal_Ranger_mk2* ne peuvent pas s'exécuter simultanément. La fonction précédente doit être terminée avant qu'une nouvelle fonction puisse être appelée.
Dans les environnements multithread, il faut veiller à ce que toutes les fonctions de l'interface accédant à la même carte ne s'exécutent pas en même temps (dans des threads différents). La méthode la plus simple consiste à s'assurer que les fonctions accédant à la même carte sont exécutées dans le même thread. Toutefois, les fonctions de l'interface accédant à différentes cartes peuvent être appelées simultanément. Toutes les fonctions de l'interface accèdent à une carte *Signal_Ranger_mk2* via son port USB
2.0. Les paramètres temporels (temps d'accès et débit) dépendent beaucoup du type de connexion (pleine vitesse ou haute vitesse) et du type de logiciel du contrôleur hôte USB sur le PC. Ils dépendent également dans une certaine mesure du trafic USB avec d'autres périphériques qui peuvent être connectés à la même racine USB (imprimante, modem, disque dur, webcam...etc.), ainsi que de la vitesse du PC. Notez que certains de ces autres périphériques peuvent utiliser une grande partie de la bande passante de l'ordinateur.

La bande passante USB est partagée entre tous les périphériques USB connectés au même contrôleur *USB*, et peut réduire les performances de transfert de données entre le PC et la carte DSP *Signal_Ranger_mk2*. La bande passante USB est partagée entre tous les périphériques USB connectés au même contrôleur USB.
La carte *Signal_Ranger_mk2* possède un port USB 2.0. Il se connecte à haut débit (480mb/s) sur toute racine ou hub compatible USB 2.0. Il se connecte à pleine vitesse (12Mb/s) sur toute racine ou hub compatible USB 1.1.
Toutes les fonctions de la DLL *SRm2_HL* sont bloquantes. Cela signifie qu'elles ne reviennent pas tant que l'action demandée n'a pas été effectuée sur la carte.
Aucune des fonctions de la DLL ne peut être bloquée indéfiniment. Si la fonction ne peut pas effectuer l'opération demandée en quelques secondes, un dépassement de temps se produit et la fonction retourne avec un code d'erreur non nul.

Pour plus de détails sur les tests de vitesse de transfert, voir la section *Sous le capot - Tests de performance USB.*

Conventions d'appel

Les fonctions sont appelées en utilisant les conventions d'appel du langage C, plutôt que les conventions standard de l'API Windows (Pascal).
Toutes les fonctions renvoient un *code d'erreur USB* sous la forme d'un entier signé de 32 bits (long). Ce code d'erreur est égal à zéro si aucune erreur ne s'est produite. S'il est différent de zéro, sa valeur indique le type d'erreur. La section *Codes d'erreur* énumère les codes d'erreur. Ils sont les mêmes pour l'interface C/C++ que pour l'interface LabVIEW.
Lorsqu'une fonction doit renvoyer un nombre, un tableau ou une chaîne de caractères, l'espace correspondant (de taille suffisante) doit être alloué par l'appelant et une référence à cet espace doit être transmise à la fonction.
Lorsqu'une fonction doit renvoyer un élément de taille variable (un tableau ou chaîne), la taille de 'élément alloué par l'appelant est également transmise à la fonction. Cet argument de taille est l'argument qui suit immédiatement le pointeur sur le tableau ou la chaîne dans la ligne d'appel.

Construction d'un projet à l'aide de Visual C++ ".Net"

Pour élaborer un projet à l'aide de Visual C++ ".net", il convient de suivre les lignes directrices suivantes. Un exemple est fourni pour accélérer la courbe d'apprentissage.

- Si le projet est lié statiquement à la bibliothèque *SRm2_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 C++ ".net", il se trouve dans *Program Files\Microsoft Visual Studio .NET 2003\Vc7\lib*\) ; dans les *propriétés du projet*, sous *Linker\Command Line\Options supplémentaires*, ajoutez la commande */DELAYLOAD:SRm2_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 *SRm2_HL.lib* sans utiliser la fonction DELAYLOAD.
- Ajouter *#include "SRm2_HL.h"* dans le fichier principal.
- Si vous utilisez la fonction DELAYLOAD pour établir un lien statique avec la bibliothèque *SRm2_HL.lib*, ajoutez *SRm2_HL.lib* au projet.
- Les fichiers suivants doivent être placés dans le dossier contenant les sources du projet :
  - extcode.h
  - fundtypes.h
  - platdefines.h
  - SRm2_HL.lib

Notes : Tous ces fichiers se trouvent dans l'exemple fourni. Voir la section de l'exemple ci-dessous.

Fonctions d'interface exportées

SR2_DLL_Open_Next_Avail_Board

```
long cdecl SR2_DLL_Open_Next_Avail_Board(char Driver_ID[],
       unsigned short ForceReset, unsigned short idVendor_Restrict, unsigned
       short idProduct_Restrict, char HardRev_Restrict[], long *BoardRef) ;
```

Description :

Cette fonction permet d'effectuer les opérations suivantes :

- Essaie de trouver une carte DSP avec l'ID de pilote sélectionné qui est connectée, mais actuellement libre, sur le PC.
- 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 "ForceReset" est vrai, la réinitialisation du DSP est forcée, puis le noyau de téléchargement de l'hôte est rechargé.
- Place la table des symboles du noyau actuel dans la structure "Global Board Info".

Entrées :

- **Driver ID** : Il s'agit d'une chaîne de caractères représentant le nom de base de la carte sélectionnée. Par exemple, pour les cartes *Signal Ranger_mk2*, cette chaîne doit être définie comme *SRm2_*.
- **ForceReset** : Mettre à 1 pour réinitialiser le DSP et charger le noyau Host-Download. Dans ce cas, tout le code précédemment exécuté est interrompu. La valeur 0 permet de ne pas perturber le DSP. Dans ce , le code déjà en cours d'exécution (le code lancé par le noyau de mise sous tension, par exemple) continue de s'exécuter.
- **idVendor_Restrict** : Cet argument est utilisé pour restreindre l'accès par ID de fournisseur. Si ce contrôle est différent de zéro, l'accès est limité aux cartes ayant l'ID du fournisseur sélectionné. La valeur zéro permet d'éviter cette restriction.
- **idProduct_Restrict** : Cet argument est utilisé pour restreindre l'accès par ID de produit. Si ce contrôle est différent de zéro, l'accès est limité aux cartes ayant l'ID de produit sélectionné. La valeur zéro permet d'éviter cette restriction.
- **HardRev_Restrict** : Cet argument est utilisé pour restreindre l'accès en fonction de la révision du micrologiciel du contrôleur USB. Si cette chaîne n'est pas vide, l'accès est limité aux cartes ayant la chaîne sélectionnée pour une révision matérielle. Laissez la chaîne vide pour éviter toute restriction.

Sorties :

- **BoardRef** : Il s'agit d'un numéro indiquant l'entrée correspondant à la carte dans la *structure globale d'information sur les cartes*. Toutes les autres fonctions d'interface utilisent ce numéro pour accéder à la carte appropriée.

Remarque : La poignée que le pilote fournit pour accéder à la carte est exclusive. Cela signifie qu'une seule application ou un seul processus à la fois peut ouvrir et gérer une carte. 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 SR2_DLL_Open_Next_Avail_Board ne peut pas être ouverte à nouveau tant qu'elle n'est pas correctement fermée à l'aide de la fonction SR2_DLL_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 ne parviendra pas à trouver et à ouvrir la carte, simplement parce que l'instance de pilote correspondante est toujours ouverte.

SR2_DLL_Close_BoardNb

```
long cdecl SR2_DLL_Close_BoardNb(long BoardRef) ;
```

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 carte, pour libérer les ressources Windows qui ne sont plus utilisées.

Entrées :

- **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 globale du tableau*. Elle est créée par SR2_DLL_Open_Next_Avail_Board

Sorties :
N/A

SR2_DLL_Complete_DSP_Reset

```
long cdecl SR2_DLL_Complete_DSP_Reset(long BoardRef) ;
```

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.

Entrées :

- **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 globale du tableau*. Elle est créée par SR2_DLL_Open_Next_Avail_Board

Sorties :
N/A

SR2_DLL_WriteLeds

```
long cdecl SR2_DLL_WriteLeds(long BoardRef, unsigned short LedState) ;
```

Description :
Cette fonction permet l'activation sélective de chaque élément de la Led bicolore.

- Arrêt
- Rouge
- Vert
- Orange

Entrées :

- **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 globale de la carte*. Elle est créée par SR2_DLL_Open_Next_Avail_Board.
- **LedState** : Cette énumération spécifie l'état des diodes électroluminescentes (0->Off, 1->Rouge, 2->Vert, 3>Orange).

Sorties : N/A

SR2_DLL_Read_Error_Count

```
long cdecl SR2_DLL_Read_Error_Count(long unsigned BoardRef,
      char *Error_Count) ;
```

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.

Entrées :

- **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 globale de la carte*. Elle est créée par SR2_DLL_Open_Next_Avail_Board.

Sorties :

- **Error_Count** : Il s'agit de la valeur contenue dans le compteur (entre 0 et 15).

SR2_DLL_Clear_Error_Count

```
long cdecl SR2_DLL_Clear_Error_Count(long BoardRef) ;
```

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

Entrées :

- **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 globale de la carte*. Elle est créée par SR2_DLL_Open_Next_Avail_Board.

Sorties :
N/A

SR2_DLL_Load_User

```
long cdecl SR2_DLL_Load_User(long BoardRef, char file_path[],
      unsigned long *EntryPoint) ;
```

Description :
Cette fonction charge un code DSP utilisateur dans la mémoire du DSP. Si *le chemin d'accès au fichier* est vide, une boîte de dialogue est utilisée. Le noyau doit être chargé avant l'exécution de cette fonction. Le DSP est réinitialisé avant 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.

Entrées :

- **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 globale de la carte*. Elle est créée par SR2_DLL_Open_Next_Avail_Board.
- **Chemin d'accès au fichier** : Il s'agit du chemin d'accès au fichier COFF (.out) du code utilisateur DSP. Une boîte de dialogue est présentée si le chemin est vide.

Sorties :

- **Point d'entrée** : Il s'agit de l'adresse dans la mémoire du DSP où l'exécution doit commencer.

SR2_DLL_Load_UserSymbols

```
long cdecl SR2_DLL_Load_UserSymbols(long BoardRef, char file_path[]) ;
```

Description :
Cette fonction charge la table des symboles dans la *structure d'information globale de la carte*. Si *file_path* est vide, une boîte de dialogue est utilisée. En général, cette fonction est utilisée pour obtenir un accès symbolique lorsque le code est déjà chargé et s'exécute sur le DSP (par exemple, le code qui a été chargé à la mise sous tension). Il n'est pas nécessaire de charger les symboles après l'exécution de SR2_DLL_Load_User, ou SR2_DLL_LoadExec_User, puisque les deux fonctions mettent à jour la table des symboles automatiquement.

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.

Entrées :

- **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 globale de la carte. Elle est créée par SR2_DLL_Open_Next_Avail_Board.
- **Chemin d'accès au fichier** : Il s'agit du chemin d'accès au fichier COFF (.out) du code utilisateur DSP. Une boîte de dialogue est présentée si le chemin est vide.

Sorties :
N/A

SR2_DLL_LoadExec_User

```
long cdecl SR2_DLL_LoadExec_User(long                                BoardRef, char file_path[],
       unsigned long *EntryPoint, unsigned short *ErrorCode) ;
```

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 *file_path* est vide, une boîte de dialogue est utilisée. Le noyau doit être chargé avant l'exécution de cette fonction. 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, le contrôleur USB et la fonction attendent un accusé de réception du DSP pour terminer son exécution. Si ce signal ne se produit pas dans les 5 secondes, la fonction s'interrompt et renvoie une erreur. 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).

Entrées :

- **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 globale de la carte. Elle est créée par SR2_DLL_Open_Next_Avail_Board.
- **Chemin d'accès au fichier** : Il s'agit du chemin d'accès au fichier COFF (.out) du code utilisateur DSP. Une boîte de dialogue est présentée si le chemin est vide.

Sorties :

- **Point d'entrée** : Il s'agit de l'adresse dans la mémoire du DSP où l'exécution doit commencer.
- **ErrorCode** : Il s'agit du code d'erreur, ou code d'achèvement, renvoyé par la fonction DSP utilisateur exécutée (fonction résidant au point d'entrée). La documentation du noyau mentionne que la fonction DSP appelée (la fonction d'entrée dans ce cas) doit contenir un accusé de réception pour signaler que la branche a été prise. Juste avant d'envoyer l'accusé de réception, le code DSP de l'utilisateur a la possibilité de modifier le champ ErrorCode dans la boîte aux lettres. Ce code d'erreur est renvoyé au PC par le contrôleur USB après avoir reçu l'accusé de réception du DSP. Ce code d'erreur est totalement spécifique à l'application de l'utilisateur. Il ne doit pas être géré par l'interface. Si le code DSP ne modifie pas le code d'erreur, zéro est renvoyé.

SR2_DLL_K_Exec

```
long cdecl SR2_DLL_K_Exec(long BoardRef, char Symbol[],
       unsigned long DSPAddress, unsigned short *ErrorCode) ;
```

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.

Le noyau doit être chargé et exécuté pour que cette fonction fonctionne.
Après avoir effectué le branchement, le contrôleur USB et la fonction attendent un accusé de réception du DSP pour terminer son exécution. Si ce signal ne se produit pas dans les 5 secondes, la fonction s'interrompt et renvoie une erreur. 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).

Entrées :

- **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 globale de la carte. Elle est créée par SR2_DLL_Open_Next_Avail_Board.
- **Symbole** : Chaîne de caractères du symbole auquel il faut accéder. Si *Symbol* est vide, *DSPAddress* est utilisé à la place.
- **DSPAddress** : Adresse physique de la branche. Elle est utilisée pour la branche si *Symbol* est vide.

Sorties :

- **ErrorCode** : Il s'agit du code d'erreur, ou code d'achèvement, renvoyé par la fonction DSP utilisateur exécutée (fonction résidant au point d'entrée). La documentation du noyau mentionne que la fonction DSP appelée (la fonction d'entrée dans ce cas) doit contenir un accusé de réception pour signaler que la branche a été prise. Juste avant d'envoyer l'accusé de réception, le code DSP de l'utilisateur a la possibilité de modifier le champ ErrorCode dans la boîte aux lettres. Ce code d'erreur est renvoyé au PC par le contrôleur USB après avoir reçu l'accusé de réception du DSP. Ce code d'erreur est totalement spécifique à l'application de l'utilisateur. Il ne doit pas être géré par l'interface. Si le code DSP ne modifie pas le code d'erreur, zéro est renvoyé.

SR2_DLL_Bulk_Move_Offset_I16

```
long cdecl SR2_DLL_Bulk_Move_Offset_I16(long BoardRef,
       unsigned short ReadWrite, char Symbol[], unsigned long DSPAddress, unsigned short
       MemSpace, unsigned long Offset, short Data[],
       long Size, unsigned short *ErrorCode) ;
```

Description :
Cette fonction 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. La bande passante est généralement élevée (jusqu'à 22 Mb/s pour USB 2).

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 de programmation= page numéro 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* n'est pas câblé, *DSPAddress* est utilisé comme adresse d'octet pour le transfert, et *MemSpace* est utilisé comme espace mémoire. Notez que *DSPAddress* doit être paire, afin de pointer vers un mot de 16 bits valide (voir la section sur la spécification de l'adresse).
- 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 (tout comme *DSPAddress*, il s'agit d'un décalage d'adresse en octets, qui doit être pair). Ceci est nécessaire pour accéder à un membre individuel d'une structure hétérogène.

Note : Le noyau doit être chargé et s'exécuter pour que cette fonction fonctionne : Le noyau doit être chargé et en cours d'exécution pour que cette fonction fonctionne.

Remarque : Les données sont transférées entre le PC et le DSP par blocs atomiques de 32 (connexion USB à grande vitesse) ou 256 (connexion USB à grande vitesse) mots. Pour ce , la fonction du noyau qui effectue le transfert par blocs désactive les interruptions pendant le transfert. Le temps nécessaire pour transférer 32 ou 256 mots de données dans la mémoire vive du DSP est généralement très court. Cependant, il peut être beaucoup plus long (jusqu'à 240ns/mot) si le transfert est effectué vers ou depuis la SDRAM. Dans ce , le temps de transfert peut être si long qu'il interfère avec le service d'autres interruptions dans le système. Ce facteur peut être particulièrement important lorsque la connexion USB est à grande vitesse, car dans ce cas, la taille du bloc est importante (256 mots). Si c'est le cas, et que l'atomicité du transfert n'est pas requise, une fonction DSP personnalisée doit être utilisée pour gérer le transfert, au lieu de la fonction noyau standard qui est instanciée par SR2_DLL_Bulk_Move_Offset_I16. Une telle fonction de transfert personnalisée peut être appelée par SR2_DLL_User_Move_Offset_I16. Une autre façon de traiter ce cas serait de diviser le transfert à haut niveau de sorte que seuls de petits blocs soient transférés à la fois.

Remarque : Les données sont transférées entre le PC et le DSP par blocs atomiques de 32 (connexion USB à grande vitesse) ou 256 (connexion USB à grande vitesse) mots. Toutefois, ce n'est le cas que si le transfert ne traverse pas une limite de 64k mots. Si le transfert traverse une limite, il est divisé à haut niveau de sorte que les deux moitiés du transfert se produisent dans la même page de 64k. Dans ce cas, le transfert perd son "atomicité".

Entrées :

- **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 globale du tableau. Elle est créée par SR2_DLL_Open_Next_Avail_Board
- **ReadWrite (lecture-écriture)** : Indique le sens du transfert (1->lecture, 0->écriture).
- **Symbole** : Chaîne de caractères du symbole auquel il faut accéder. Si *Symbol* est vide, *DSPAddress* et *MemSpace* sont utilisés à la place.
- **DSPAddress** : Adresse physique de base du DSP pour l'échange. *DSPAddress* n'est utilisé que si *Symbol* est vide.
- **MemSpace** : Espace mémoire pour l'échange (0-> programme, 1->données ou 2->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.
- **Taille** : Représente la taille du tableau de *données* alloué, ainsi que le nombre déléments à lire ou à écrire.

Sorties :

- **ErrorCode** : Il s'agit du code d'erreur renvoyé par la fonction du noyau qui est exécutée. Les lectures du noyau renvoient toujours un code d'achèvement de 1 ; les écritures du noyau renvoient toujours un code d'achèvement de 2. Ce code d'erreur est différent du *code USB_Error_Code* renvoyé par la fonction.

Entrées/sorties :

- **Données** : Mots de données à lire ou à écrire dans la mémoire du DSP. Pour une lecture, l'appelant doit allouer le tableau de *données* avant l'appel.

SR2_DLL_User_Move_Offset_I16

```
long cdecl SR2_DLL_User_Move_Offset_I16(long BoardRef,
              unsigned short ReadWrite, char Symbol[], unsigned long
              DSPAddress, unsigned long Offset, char BranchLabel[],
              unsigned long BranchAddress,
              short Data[], long Size, unsigned short *ErrorCode) ;
```

Description :

Cette fonction est similaire à SR2_DLL_Bulk_Move_Offset, sauf qu'elle permet à une fonction DSP définie par l'utilisateur de remplacer la fonction noyau intrinsèque utilisée par SR2_DLL_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. En plus de 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, SR2_DLL_User_Move_Offset comprend deux contrôles pour définir le point d'entrée de fonction qui doit être utilisée à la place de la fonction noyau intrinsèque.
Lorsque la fonction définie par l'utilisateur est appelée, la boîte aux lettres contient :

- Le champ BranchAddress correspond au point d'entrée de la fonction. Ce champ n'est normalement pas utilisé directement par la fonction DSP. Il est fixé à son point d'entrée, ce qui conduit à son exécution.
- Le champ TransferAddress. Il contient l'adresse correspondant à *Symbol*, si *Symbol* est utilisé, ou le nombre de 32 bits défini par l'utilisateur *DSPAddress* si *Symbol* est vide. Il est à noter que ce champ n'est pas tenu de contenir une adresse valide. Par exemple, dans le cas d'une fonction de gestion FIFO, il peut s'agir d'un numéro FIFO.
- Le champ NbWords. Ce nombre de 16 bits représente le nombre de mots à transférer. Il est toujours compris entre 1 et 32 (connexion USB à pleine vitesse), ou entre 1 et 256 (connexion USB à haute vitesse). Il représente la taille du champ de données de la boîte aux lettres.
- Le champ ErrorCode. Il vaut 1 pour les lectures et 2 pour les écritures.
- Dans le cas d'un transfert en écriture, de 1 à 32 mots, comme indiqué par NbWords, (1 à 256 mots pour une connexion USB à grande vitesse) ont été écrits dans le champ de données de la boîte aux lettres par le contrôleur USB avant l'appel de la fonction DSP.

La fonction définie par l'utilisateur doit exécuter sa fonction. S'il s'agit d'une lecture, elle doit lire le nombre de mots requis dans le champ de données de la boîte aux lettres. Elle doit ensuite mettre à jour la boîte aux lettres avec les données suivantes :

- S'il s'agit d'une écriture, elle doit fournir le nombre de mots requis dans le champ de données de la boîte aux lettres.
- Il peut ensuite mettre à jour le champ ErrorCode de la boîte aux lettres avec un code d'achèvement approprié à la situation (c'est le code d'erreur qui est renvoyé en tant qu'indicateur ErrorCode du VI).

Ensuite, la fonction définie par l'utilisateur doit simplement envoyer un accusé de réception.

Un transfert d'un nombre de mots supérieur à 32 (supérieur à 256 pour une connexion USB à haut débit) est segmenté par l'interface 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.

Le champ TransferAddress de la boîte aux lettres n'est initialisé qu'au premier segment. La fonction définie par l'utilisateur peut choisir de l'incrémenter pour pointer vers des adresses successives (c'est ce que font les fonctions intrinsèques du noyau), ou de le laisser intact (ce qui serait approprié si le champ contenait un numéro FIFO par exemple). La manière dont ce champ est géré est totalement spécifique à l'application.

Remarque : 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 scindés en deux transferts consécutifs. Les informations contenues dans TransferAddress sont uniquement conservées*

au cours du 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 d'octet, c'est pourquoi son bit 0 est masqué à haut niveau.*

Le champ NbWords est initialisé avec la taille du segment transféré, à chaque segment. Le champ ErrorCode n'est initialisé qu'au premier segment avec la valeur 1 (lecture) ou 2 (écriture).
La valeur renvoyée à l'indicateur ErrorCode du VI est la valeur qui peut être mise à jour par la fonction définie par l'utilisateur avant d'acquitter le segment LAST. Si la fonction définie par l'utilisateur ne met pas à jour le code d'erreur, la même valeur (1 pour les lectures et 2 pour les écritures) est renvoyée au PC.

Remarque : Le noyau ET le code DSP de la fonction définie par l'utilisateur doivent être chargés et exécutés pour que cette Vi soit fonctionnelle.

Note : La valeur de l'indicateur R/~W est reflétée par le contenu du champ ErrorCode de boîte aux lettres à l'entrée de la fonction définie par l'utilisateur. Pour les lectures, la valeur est 1, pour les écritures, la valeur est 2.

Entrées :

- **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 globale du tableau. Elle est créée par SR2_DLL_Open_Next_Avail_Board
- **ReadWrite (lecture-écriture)** : Indique le sens du transfert (1->lecture, 0->écriture).
- **Symbole** : Chaîne de caractères du symbole auquel il faut accéder. Si *Symbol* est vide, *DSPAddress* est utilisé à la place.
- **DSPAddress** : Adresse physique de base du DSP pour l'échange. *DSPAddress* 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.
- **BranchLabel** : Chaîne de caractères correspondant à l'étiquette de la fonction définie par l'utilisateur. Si la fonction est définie par l'utilisateur, elle doit être définie par l'utilisateur.
  BranchLabel est vide, *BranchAddress* est utilisé à la place.
- **BranchAddress** : Adresse physique de base du DSP pour la fonction définie par l'utilisateur.
- **Taille** : Représente la taille du tableau de *données* alloué, ainsi que le nombre déléments à lire ou à écrire.

Sorties :

- **ErrorCode** : Il s'agit du code d'erreur renvoyé par la fonction utilisateur exécutée. Ce code d'erreur est différent du *code USB_Error_Code* renvoyé par la fonction.

Entrées/sorties :

- **Données** : Mots de données à lire ou à écrire dans la mémoire du DSP. Pour une lecture, l'appelant doit allouer le tableau de *données* avant l'appel.

SR2_DLL_HPI_Move_Offset_I16

```
long cdecl SR2_DLL_HPI_Move_Offset_I16(long           BoardRef,
             unsigned short ReadWrite, char Symbol[],
             unsigned long DSPAddress, unsigned short MemSpace, long Offset, short
             Data[], long Size) ;
```

Description :

Cette fonction est similaire à SR2_DLL_Bulk_Move_Offset, sauf qu'elle s'appuie sur le matériel de l'IPH, plutôt que sur le noyau, pour effectuer le transfert.
Les transferts sont limités à la RAM sur puce directement accessible via l'interface HPI (adresses d'octets $00C0_H$à $FFFF_{H)}$. Le contrôle MemSpace n'est pas utilisé. Ce VI effectuera 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, quel que soit l'espace mémoire. L'espace E/S sur la puce n'est pas accessible. Si l'on tente d'écrire des données en dehors de la plage autorisée, les données ne sont pas écrites. Si l'on tente de lire des données en dehors de la plage autorisée, des données erronées sont renvoyées.

Remarque : il n'est pas nécessaire que le noyau soit chargé ou fonctionnel pour cette fonction s'exécute correctement. Cette fonction terminera le transfert même si le DSP s'est planté, ce qui en fait un bon outil de débogage.
Les transferts avec l'interface HPI utilisent le control pipe 0 au lieu des bulk pipes rapides utilisés par SR2_DLL_Bulk_Move_Offset. La bande passante pour ces transferts est généralement faible (500kb/s pour USB 1.1). Elle est cependant garantie.

Entrées :

- **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 globale du tableau*. Elle est créée par SR2_DLL_Open_Next_Avail_Board
- **ReadWrite (lecture-écriture)** : Indique le sens du transfert (1->lecture, 0->écriture).
- **Symbole** : Chaîne de caractères du symbole auquel il faut accéder. Si *Symbol* est vide, *DSPAddress* et *MemSpace* sont utilisés à la place.
- **DSPAddress** : Adresse physique de base du DSP pour l'échange. *DSPAddress* n'est utilisé que si *Symbol* est vide.
- **MemSpace** : Espace mémoire pour l'échange (0-> programme, 1->données ou 2->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.
- **Taille** : Représente la taille du tableau de *données* alloué, ainsi que le nombre déléments à lire ou à écrire.

Sorties :
N/A

Entrées/sorties :

- **Données** : Mots de données à lire ou à écrire dans la mémoire du DSP. Pour une lecture, l'appelant doit allouer le tableau de *données* avant l'appel.

SR2_DLL_Resolve_UserSymbol

```
long cdecl SR2_DLL_Resolve_UserSymbol(long BoardRef, char Symbol[],
                    unsigned long *Value, unsigned short *MemSpace) ;
```

Description :
Cette fonction peut être utilisée pour fournir l'adresse correspondant à un symbole particulier dans la table des symboles du code DSP actuellement chargé. Utilisée conjointement avec les fonctions de transfert de données (SR2_DLL_Bulk_Move_Offset, SR2_DLL_User_Move_Offset et SR2_DLL_HPI_Move_Offset), elle permet une plus grande flexibilité dans le choix de l'adresse de transfert réelle. Par exemple, l'adresse peut être transformée d'une manière spécifique à l'application avant le transfert.

Entrées :

- **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 globale du tableau. Elle est créée par SR2_DLL_Open_Next_Avail_Board
- **Symbole** : Chaîne de caractères du symbole auquel il faut accéder. Si *Symbol* est vide, *DSPAddress* et *MemSpace* sont utilisés à la place.
- **MemSpace** : Espace mémoire pour l'échange (0-> programme, 1->données ou 2->IO). *MemSpace* n'est utilisé que si *Symbol* est vide.

Sorties :

- **Valeur** : Il s'agit de la valeur de résolution du symbole.
- **MemSpace** : Il s'agit de l'espace mémoire du symbole. Notez que cette valeur reflète en fait le "numéro de page" qui a été utilisé au moment de la liaison pour ce symbole. Pour que ce numéro indique un espace mémoire valide, le symbole doit être lié selon la convention habituelle :
  - 0 -> Espace de programmation
  - 1 -> Espace de données
  - 2 -> Espace IO

SR2_DLL_InitFlash

```
long cdecl SR2_DLL_InitFlash(long BoardRef, double *FlashSize) ;
```

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 exécuté en plus du noyau 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.

Entrées :

- **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 globale du tableau. Elle est créée par SR2_DLL_Open_Next_Avail_Board

Sorties :

- **FlashSize** : Cet indicateur indique la taille du flash détecté en kWords. Si aucun flash n'est détecté, il renvoie zéro.

SR2_DLL_EraseFlash

```
long cdecl SR2_DLL_EraseFlash(long BoardRef,
       unsigned long StartingAddress, unsigned long Size) ;
```

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 précédant l'adresse de départ seront effacés jusqu'au début du secteur. 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.

Note : La taille du secteur est de 32 mots : La taille du secteur est de 32 mots-clés.

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.

Entrées :

- **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 globale de la carte. Elle est créée par SR2_DLL_Open_Next_Avail_Board.
- **Adresse de départ** : Adresse du premier mot à effacer.
- **Taille** : Nombre de mots à effacer.

Sorties :
- **N/A**

SR2_DLL_FlashMove_I16

```
long cdecl SR2_DLL_FlashMove_I16(long BoardRef,
       unsigned short ReadWrite, unsigned long DSPAddress, short
       DataIn[], long Size) ;
```

Description :
Cette fonction 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, la fonction SR2_DLL_Bulk_Move_Offset doit utilisée à la place, puisqu'elle ne nécessite pas la présence du code de support Flash.

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 , 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.

Remarque : Contrairement aux générations précédentes de dispositifs Flash utilisés dans les cartes Signal Ranger, le dispositif Flash utilisé dans Signal_Ranger_mk2 ne peut pas être programmé de manière incrémentielle. Cela signifie qu'un emplacement de mot qui a été précédemment programmé DOIT être effacé avant d'être reprogrammé. Cela est vrai même si l'opération de reprogrammation ne vise qu'à transformer certains des "1" restants en "0".

Entrées :
- **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 globale du tableau. Elle est créée par SR2_DLL_Open_Next_Avail_Board
- **ReadWrite (lecture-écriture)** : Indique le sens du transfert (1->lecture, 0->écriture).
- **DSPAddress** : Adresse physique de base de la DSP pour l'échange.
- **Taille** : Représente la taille du tableau de *données* alloué, ainsi que le nombre déléments à lire ou à écrire.

Sorties :
- **N/A**

Entrées/sorties :
- **Données** : Mots de données à lire ou à écrire dans la mémoire Flash. Pour une lecture, l'appelant doit allouer le tableau de *données* avant l'appel.

SR2_DLL_FPGA_LoadConfiguration

```
long cdecl SR2_DLL_FPGA_LoadConfiguration(long BoardRef,
       char file_path[], char Tools_Version[], long lenTools, char
       Design_Name[], long lenDesign, char Architecture[],
       long lenArch, char Build_Date[], long lenBuild, char Device[], long lenDev)
       ;
```

Description :
Cette fonction télécharge un fichier de configuration logique "*.rbt" dans le FPGA. Le DSP est réinitialisé avant le téléchargement. Tout le code DSP est interrompu. Le fichier .rbt doit être valide et correct pour le FPGA spécifié. Le chargement d'un fichier rbt non valide dans un FPGA peut endommager la pièce.

Entrées :

- **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 globale de la carte. Elle est créée par SR2_DLL_Open_Next_Avail_Board.
- **file_path** : Chemin d'accès au fichier ".rbt" décrivant la logique FPGA. Une boîte de dialogue est présentée si le chemin est vide.
- **lenTools** : Il s'agit de la taille de la chaîne de caractères qui a été allouée pour contenir les *Tools_Version*. Un minimum de 60 caractères doit être alloué.
- **lenDesign** : Il s'agit de la taille de la chaîne de caractères qui a été allouée pour contenir l'élément *Nom du dessin ou modèle*. Un minimum de 60 caractères doit être attribué.
- **lenArch** : Il s'agit de la taille de la chaîne de caractères qui a été allouée pour contenir l'élément Sortie de l'*architecture*. Un minimum de 60 caractères doit être alloué.
- **lenBuild** : Il s'agit de la taille de la chaîne de caractères qui a été allouée pour contenir la *Build_Date.* de sortie. Un minimum de 60 caractères doit être attribué.
- **lenDev** : Il s'agit de la taille de la chaîne qui a été allouée pour contenir le *dispositif.* de sortie. Un minimum de 60 caractères doit être attribué.

Sorties :

- **Tools_Version** : Cette chaîne contient les informations relatives à la version des outils figurant dans le fichier .rbt. Une chaîne d'au moins 60 caractères doit être allouée par l'appelant avant l'appel.
- **lenDesign** : Cette chaîne contient les informations relatives au nom du dessin ou modèle figurant dans le fichier .rbt. Une chaîne d'au moins 60 caractères doit être allouée par l'appelant avant l'appel.
- **lenArch** : Cette chaîne contient les informations relatives à l'architecture contenues dans le fichier .rbt. Une chaîne d'au moins 60 caractères doit être allouée par l'appelant avant l'appel.
- **lenBuild** : Cette chaîne contient les informations relatives à la date de construction figurant dans le fichier .rbt. Une chaîne d'au moins 60 caractères doit être allouée par l'appelant avant l'appel.
- **lenDev** : Cette chaîne contient les informations sur le dispositif trouvées dans le fichier .rbt. Une chaîne d'au moins 60 caractères doit être allouée par l'appelant avant l'appel.

Codes d'erreur

Le tableau suivant énumère les codes d'erreur qui peuvent être renvoyés par les différentes fonctions de SRm2_HL.dll.

Note : La liste n'est pas exhaustive : La liste n'est pas exhaustive. Seuls les codes les plus courants y figurent.

Erreur Nb	Cause
00000002h	La mémoire est pleine
00000003h	Accès à une mauvaise zone de mémoire
00000004h	Fin de fichier rencontrée
00000005h	Fichier déjà ouvert
00000006h	Erreur d'E/S de fichier
00000007h	Fichier non trouvé
00000008h	Erreur de permission de fichier
00000009h	Disque plein
0000000Ah	Chemin d'accès en double
0000000Bh	Trop de fichiers ouverts
BFFC0804h	Aucune carte détectée
BFFC0805h	Erreur de génération de carte ouverte
BFFC0806h	Les informations de la carte génèrent une erreur
BFFC0807h	La carte de fermeture génère une erreur
BFFC0808h	Fichier DSP non accessible
BFFC0809h	Réinitialiser l'erreur générée par le DSP
BFFC0812h	Ce noyau DSP n'est pas pour le c5000
BFFC080Ah	Unreset_DSP genrate error

BFFC080Bh	Fichier DSP introuvable
BFFC080Ch	Fichier DSP non valide
BFFC080Dh	Le type de fichier DSP n'est pas pris en charge
BFFC080Eh	Le noyau DSP ne compare pas
BFFC080Fh	Write_hpi génère une erreur
BFFC0810h	Read_hpi génère une erreur
BFFC0811h	Ce fichier DSP n'est pas pour le c5000
BFFC0813h	Erreur d'écriture et de lecture
BFFC0814h	Le DSP n'est pas revenu... il est peut-être tombé en panne.
BFFC0815h	Symbole non trouvé dans le tableau
BFFC0816h	DSPInt génère une erreur
BFFC0817h	Délai d'attente DSP
BFFC0818h	Le noyau n'est pas chargé
BFFC0820h	Erreur de fermeture du pilote
3FFC0104h	Erreur Requête non prise en charge
BFFC0821h	Erreur d'ouverture du pilote
BFFC0822h	Le déplacement de données avec le noyau génère une erreur
BFFC0823h	Le niveau bas de l'exécution génère une erreur
BFFC0824h	L'obtention d'informations sur les tuyaux génère des erreurs
BFFC0825h	Réinitialisation Le DSP génère une erreur
BFFC0826h	Le déménagement de l'IPH génère une erreur
BFFC0827h	DSP int génère une erreur
BFFC0828h	Erreur dans le descripteur de configuration
BFFC0829h	L'écriture de Led génère une erreur
BFFC082Ah	Erreur de descripteur de périphérique
BFFC082Bh	Erreur de contrôle HPI en lecture et en écriture
BFFC082Ch	Modifier l'erreur du mode Timeout de l'USB
BFFC082Dh	La section du code DSP ne vérifie pas
BFFC082Eh	Fichier FPGA non valide
BFFC082Fh	Configuration du FPGA interrompue avant la fin
BFFC0830h	Erreur d'accès au pilote
BFFC0831h	Famille DSP non reconnue ou carte non ouverte
BFFC0832h	Le fichier FPGA ne correspond pas à la famille de dispositifs cible
BFFC0833h	Section trop grande pour être affichée dans Flash
BFFC0834h	Le fichier DSP ne peut pas être lié
BFFC0835h	Section non alignée sur une adresse d'octets pairs
BFFC0836h	Erreur d'effacement de la mémoire flash
BFFC0837h	Erreur d'écriture Flash

Exemple de code

Un exemple est fourni, incluant le côté PC et le côté DSP. Cet exemple se trouve dans le répertoire suivant : *C:\NProgram Files\NSignalRanger_mk2\NExemples\NC_Exemples\NHLDLLCallExample.zip.*
Il suffit de décompresser le *fichier HLDLLCallExample.zip* pour déflater
l'exemple. L'exemple contient les éléments suivants :

- Un répertoire nommé *UserFunctionDSPCode*, qui contient le projet DSP et le code source.
- Un répertoire nommé *TestHLDLL*, qui contient le projet Visual C++ .net et le code source.

Test du code d'exemple

Pour tester le code de l'exemple, il suffit de mettre la carte sous tension et d'attendre que sa DEL devienne verte. Exécutez ensuite l'application *TestHLDLL.exe* qui se trouve dans le répertoire *HLDLLCallExample\TestHLDLL\Debug*. La figure suivante montre l'interface utilisateur de l'application d'exemple :

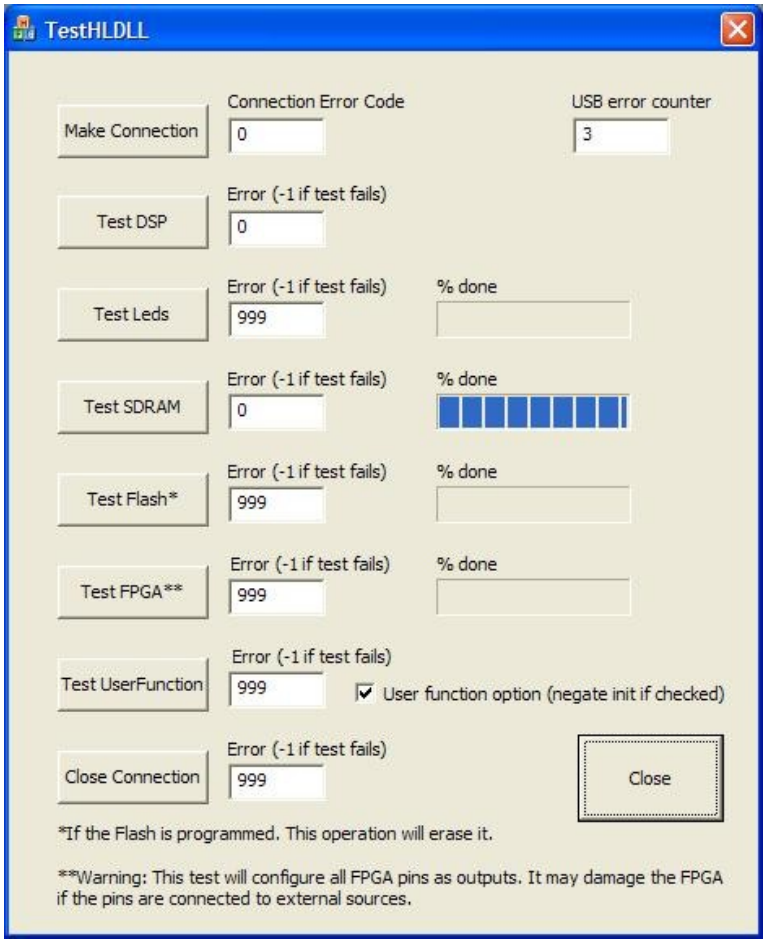


Figure 5 Exemple d'application C

Avant de tenter d'exécuter l'un des tests, appuyez sur le bouton *Établir la connexion* et attendez que l'application renvoie un code d'erreur nul. Cette première opération peut prendre quelques secondes, car c'est à ce moment que la DLL *SRm2_HL.dll* est chargée.
Une fois que la connexion à la carte a été établie avec succès, n'importe quel test peut être effectué dans n'importe quel ordre.
Lorsque tous les tests ont été effectués, appuyez sur le bouton *Fermer la connexion* pour fermer le pilote et effectuer les opérations de nettoyage nécessaires.
Ensuite, le fait d'appuyer sur le bouton *Fermer* met fin à l'application.

Fichiers nécessaires à la compilation

La ligne suivante doit être présente dans le fichier source principal :

```
# include SRm2_HL.h
```

Dans l'exemple, il est placé au début du fichier source *TestHLDLLDlg.cpp.*
Ce fichier include comprend lui-même les fichiers suivants : *extcode.h, fundtypes.h et platdefines.h,*
Les quatre fichiers doivent être présents dans le même répertoire que *TestHLDLLDlg.cpp* pour que le compilateur puisse construire le code objet.
Les fichiers *SRm2_HL.lib* et *SRm2_HL.dll* doivent être présents dans le répertoire du projet.

Configuration du projet

Pour établir un lien statique avec SRm2_HL.dll, procédez comme suit :

- Allez dans le menu *Ajouter un nouvel élément au projet* et choisissez le fichier *SRm2_HL.lib.*
- Allez dans le menu *Add New Item To Project* (*Ajouter un nouvel élément au* projet) et choisissez le *fichier delayimp.lib* dans le projet (dans Visual C++ ".net", il se trouve dans *Program Files\Microsoft Visual Studio .NET 2003\Vc7\lib*\).
- Allez dans le menu *ProjectProperties*, choisissez *Linker Options* et cliquez sur *Command Line*. Ajoutez la ligne de commande suivante :

```
/DELAYLOAD:SRm2_HL.dll
```

Cette ligne de commande retarde le chargement de la DLL au moment de l'exécution jusqu'à ce que le premier appel ait lieu. Elle est nécessaire au bon fonctionnement de l'interface. La DLL peut également être liée dynamiquement (voir *Construire un projet en utilisant Visual C/C++ ".net").*

Fichiers nécessaires à l'exécution

Pour garantir le bon fonctionnement de tout exécutable créé, les fichiers de support suivants doivent être présents dans le même répertoire que l'exécutable :

- *Sranger2.dll*
- *SRm2_HL.dll*
- *SR2_Flash_Support.out*
- *SR2_FPGA_Support.out*
- *SR2_Kernel_HostDownload.out*
- *SR2_Kernel_PowerUp.out*

En outre, les fichiers suivants, spécifiques à l'utilisateur, doivent également être présents :

- *UserFunctionDSPCode.out*. Il s'agit du code DSP requis par l'application utilisateur spécifique (dans ce cas, l'application *TestHLDLL.exe)*.
- *SR2_SelfTest.rbt.* Il s'agit de la logique FPGA requise par l'application utilisateur spécifique.

Note : Les fichiers spécifiques à l'utilisateur peuvent avoir un nom différent : Certains fichiers spécifiques à l'utilisateur peuvent avoir un nom différent. Ce sont les fichiers qui sont chargés par le code de l'exemple.

Remarque : Une logique FPGA spécifique peut ne pas être nécessaire. Cet exemple d'application charge cette logique FPGA pour pouvoir tester le FPGA.

Autres détails

D'autres détails de la mise en œuvre peuvent être trouvés dans le code source (côté DSP et côté PC) sous forme de commentaires.

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 lors du développement d'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 avant ce code.
- Le code DSP est une fonction simple, destinée à s'exécuter, 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 courts segments de code de manière asynchrone par rapport à d'autres codes s'exécutant en arrière-plan sur le DSP. *L'autre code* s'exécutant en arrière-plan peut être le noyau ou un autre 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 le code DSP développé pour la carte Signal_Ranger_mk2 doit être conforme aux exigences suivantes.

Configuration de Code Composer Studio

La configuration de Code Composer Studio doit utiliser un simulateur ou un émulateur C55x. Dans le cas contraire, l'éditeur de liens visuels pourrait ne pas fonctionner comme prévu. Si un simulateur est utilisé, nous recommandons d'utiliser le simulateur C5502 "cycle-accurate". Il est le plus proche du DSP physique utilisé sur la carte.

Exigences du projet

Dans Code Composer Studio, le projet doit être créé pour une plate-forme C55x (PAS la plate-forme C54x).

Exigences du code C

- Lors du développement d'une fonction en C qui sera lancée par le processus du noyau K_Exec et qui doit revenir au code précédemment exécuté après son exécution, la fonction doit être déclarée en utilisant le qualificateur d'*interruption*. Cela indique au compilateur de protéger les registres DSP requis sur la pile et de terminer la fonction par une instruction RETI, qui restaure correctement le contexte de la pile à la fin de l'exécution. En outre, la fonction doit inclure la macro d'*acquittement* (voir les exemples de code), qui doit être exécutée dans les 5 secondes suivant le lancement de la fonction.
- Lorsque la fonction n'est pas destinée à revenir au code précédemment exécuté (la fonction *principale* d'un projet C par exemple), il n'est pas nécessaire de la déclarer avec le qualificateur d'*interruption.* Cependant, elle doit toujours contenir l'accusé de réception dans les 5 secondes suivant l'entrée. L'accusé de réception signale au PC que la fonction a été lancée avec succès (voir les sections sur le noyau pour plus de détails).

Exigences en matière d'assemblage

- Lors du développement d'une fonction en assembleur qui sera lancée par le processus du noyau K_Exec et qui doit revenir au code précédemment exécuté après son exécution, la fonction doit protéger tous les registres DSP qu'elle utilise sur la pile. La fonction doit se terminer par une instruction RETI, qui restaure correctement le contexte de la pile à la fin de l'exécution. En outre, la fonction doit inclure la macro d'*accusé de réception* (voir les exemples de code), qui doit être exécutée dans les 5 secondes suivant le lancement de la fonction.
- Lorsque la fonction n'est pas destinée à revenir au code précédemment exécuté, elle n'est pas tenue de protéger les registres qu'elle utilise. Cependant, elle doit toujours contenir l'accusé de réception dans les 5 secondes qui suivent l'entrée. L'accusé de réception signale au PC que la fonction a été lancée avec succès (voir les sections sur le noyau pour plus de détails).
- Lors du développement d'une section de code en assembleur, la directive *.even* doit être utilisée au début de la section. Cela permet de s'assurer que les sections de code commencent toujours à une adresse d'octet paire. Cela n'est pas nécessaire pour développer du code en C, car le compilateur C utilise lui-même la directive *.even* dans le code assembleur qu'il génère. Cette exigence ne s'applique qu'aux sections de code.

Options de construction

Compilateur

- Basique : L'option *Full Symbolic Debug* doit utilisée. Cette option ne produit pas un code plus grand ou moins efficace. Cependant, elle fournit les informations symboliques dans le fichier ".out" qui sont nécessaires pour accéder aux étiquettes et aux variables à partir de leurs noms symboliques. En particulier, les informations symboliques concernant les structures ne sont fournies dans le fichier ".out" que si cette option est utilisée.
- Avancé : Le champ *version du processeur (-v)* doit être initialisé avec *5502*. De cette manière, le code est optimisé pour le DSP TMS320VC5502. Certaines plates-formes de la famille C55x ne disposent pas du jeu d'instructions complet du C5502. En outre, lorsqu'une plate-forme spécifique n'est pas indiquée, le compilateur peut devoir contourner des bogues de silicium qui ne sont pas présents dans le C5502, mais qui peuvent être présents dans d'autres DSP de la même famille, créant ainsi un code moins efficace.
- Pour les versions de Code Composer Studio 3 et suivantes, le nouveau format DWARF est utilisé pour stocker les informations de débogage dans le fichier COFF. Pour avoir accès aux informations sur les membres de la structure, l'option de compilation "`--symdebug:coff`" doit être utilisée. Il suffit d'insérer l'option "`-- symdebug:coff`" à la fin de la ligne de commande du compilateur.

Modules requis

Support en temps réel

Pour un code DSP écrit en C, le fichier rts55.lib ou rts55x.lib (selon le modèle de mémoire) doit être ajouté aux fichiers du projet.

Vecteurs d'interruption

- Si le projet utilise des interruptions, le module *vectors.asm* doit être ajouté au projet. Le fichier modèle fourni avec les exemples doit être utilisé comme exemple pour générer une table de vecteurs d'interruption qui fournit les vecteurs pour le code utilisateur. Il suffit de modifier le tableau en fonction des vecteurs à implémenter.
- Ne pas modifier les valeurs des symboles suivants :
  - Noyau BVR
  - _RESET_K

- Ne modifiez pas le vecteur d'interruption DSPINT. Cela ferait planter le noyau DSP qui utilise l'interruption.
- N'ajoutez pas de vecteur d'interruption pour la dernière interruption (TRAP #31). Ce vecteur a été intentionnellement omis de la table parce que le noyau le modifie dynamiquement. L'initialisation de ce vecteur ferait planter le noyau dès que la table des vecteurs serait chargée.

- Si aucune interruption autre que DSPInt et Trap #31 (les interruptions utilisées par le noyau) n'est utilisée dans le projet, il n'est normalement pas nécessaire de fournir une section de *vecteurs*. En effet, les vecteurs des interruptions utilisées par le noyau sont déjà initialisés lorsque l'utilisateur prend le contrôle du DSP. Cependant, il peut être préférable de toujours inclure une section vecteurs pour la raison expliquée ci-dessous.

Notez que pour les projets écrits en C, une section de *vecteurs* par défaut est automatiquement incluse dans *rts55.lib* ou *rts55x.lib* si le projet n'en définit pas déjà une. Cette section de vecteurs par défaut n'est pas compatible avec le noyau et ne doit jamais être utilisée. La façon la plus simple d'éviter cette section de vecteurs par défaut est d'inclure une section de *vecteurs* définie par l'utilisateur qui contient au moins les vecteurs appropriés pour le fonctionnement du noyau. Il suffit d'inclure le fichier modèle fourni avec l'option

dans le projet. Ce problème ne se produit que dans les projets qui incluent le *fichier rts55.lib* ou *rts55x.lib*. Ce n'est généralement le cas que pour les projets écrits en C/C++.

Exigences en matière de liens

Mémoire Description Fichier

Pour lier le code à l'aide de l'éditeur de liens visuels, utilisez le fichier de description *Signal_Ranger_mk2.mem.* Ce fichier décrit la carte mémoire de la carte Signal Ranger_mk2. Il inclut les plages d'adresses qui sont réservées et ne doivent pas être modifiées.

Section des vecteurs

La section des *vecteurs* doit être placée dans la plage de mémoire des *vecteurs*, entre les adresses d'octets 100_Het $1FF_H$. Notez que la section *vecteurs* est un peu plus petite que la plage de mémoire *Vecteurs.* C'est parce que le vecteur pour le TRAP #31 qui est utilisé par le noyau a été intentionnellement omis de la section (voir ci-dessus). Veillez à ne rien charger dans ce petit espace à la fin de la section, car cela interférerait avec le fonctionnement du noyau et ferait très probablement planter pendant le chargement du code DSP.

Section ISR inutilisée

La petite section *Unused_ISR* fournit un simple RETI pour toutes les interruptions qui ne sont pas utilisées. De cette façon, si un tel ISR est déclenché, il retourne de manière inoffensive au code de l'utilisateur immédiatement après avoir été déclenché. Cette petite section peut être chargée n'importe où avec le reste du code. Elle ne doit jamais être chargée à la fin de la section des *vecteurs*.

Symboles mondiaux

Seuls les symboles globaux trouvés dans le fichier ".out" du 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 assembleur (points d'entrée des fonctions par exemple) doivent être déclarées avec la directive d'assembleur *.global*.

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, dans les 5s d'exécution, même si 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 SR2_K_Exec.vi, ou en utilisant le bouton ***Exec*** *du mini-débogueur. Par conséquent, le code DSP qui a été développé et testé en utilisant le mini-débogueur, ou le bouton Exec du mini-débogueur, doit être validé.*

qui est habituellement téléchargé et exécuté à partir du PC devrait normalement être prêt à être programmé dans la table de démarrage "tel quel".

De plus amples informations sur la commande acknowledge sont disponibles dans les sections suivantes concernant les noyaux.

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.

Mini-Debugger

Le mini-débogueur permet au développeur d'interagir :

- Réinitialiser la carte Signal Ranger mk2.
- Télécharger un fichier exécutable DSP dans la mémoire du DSP ou utiliser les symboles d'un code DSP déjà en mémoire.
- Lance l'exécution du code 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 un shell d'interface utilisateur qui exploite les capacités des bibliothèques d'interface PC pour permettre au développeur d'observer et de modifier la mémoire 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 qui sont fournies pour développer les applications PC qui utilisent la carte, la transition entre le débogage et le déploiement sur le terrain est totalement transparente.

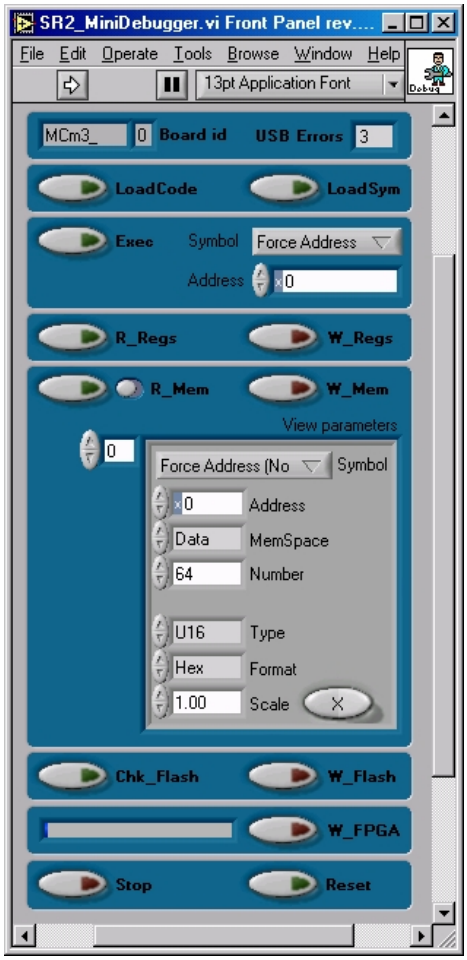


Figure 6 **Interface utilisateur du mini-débogueur**

Description de l'interface utilisateur

- **ID de la carte** Il s'agit d'un champ de texte dans lequel le préfixe de l'ID du pilote de la carte doit être saisi. 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. Le petit champ à droite de Board ID indique le numéro d'instance du pilote pour la carte qui a été ouverte par le mini-débogueur. Le mini-débogueur ouvre toujours la première carte du type spécifié qui n'est pas déjà ouverte. Pour les cartes Signal Ranger mk2, le champ doit être initialisé avec *SRm2_*.
- **Erreurs USB** 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 de nombreuses erreurs USB peuvent être détectées, qui n'affecteront pas la transmission à haut niveau. Cela est dû à la robustesse intrinsèque du protocole USB aux erreurs.
- **LoadCode** Charge un fichier COFF du DSP. Le DSP est réinitialisé avant le téléchargement du code. L'application présente un navigateur de fichiers pour permettre à l'utilisateur de sélectionner le fichier. Le fichier doit être un fichier COFF (".out") légitime pour le DSP cible. 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.
- **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,

comme 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 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'*accusé de réception* 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, ***Address*** 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 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 du CPU mappés dans la RAM à l'adresse 0000_H, et présente données dans un format facile à lire.

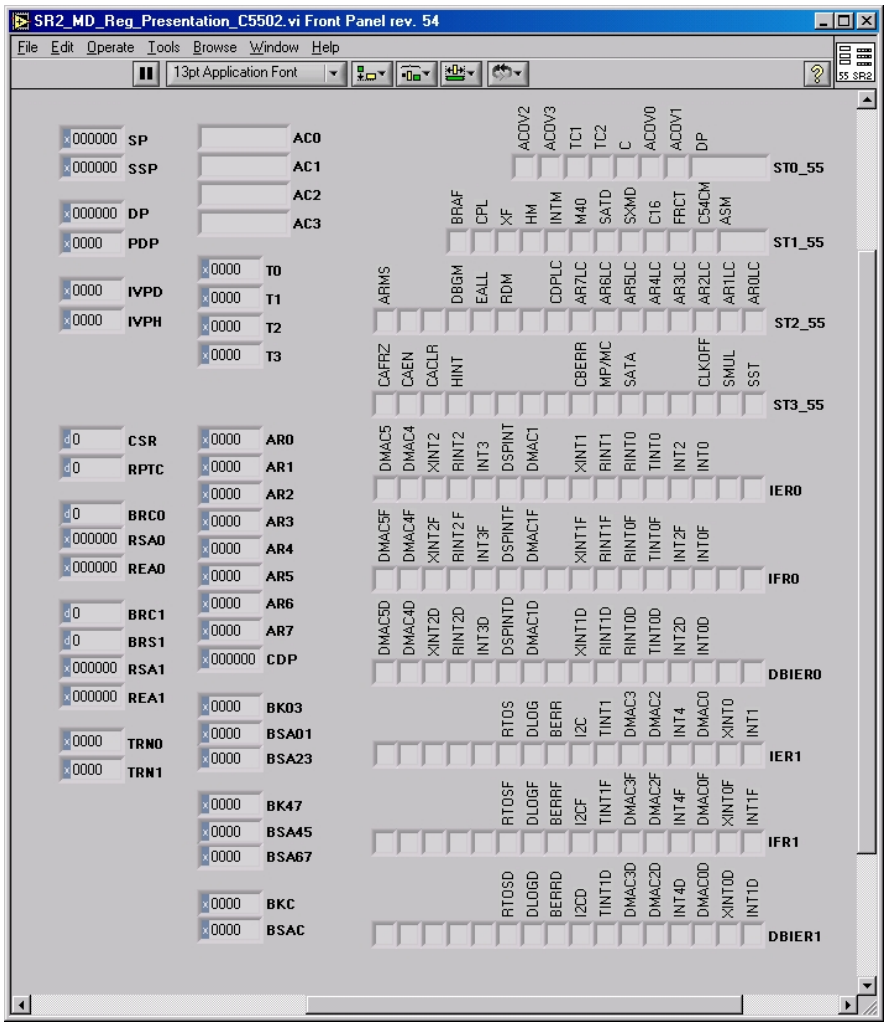


Figure 7 Panneau de présentation du registre

- **W_regs** Écrit la plupart des registres du CPU mappés en RAM à l'adresse 0000_H. Certains champs sont grisés et ne peuvent pas être écrits, soit parce que le noyau les utilise et les restaurerait après modification, soit parce que leur modification compromettrait son fonctionnement.

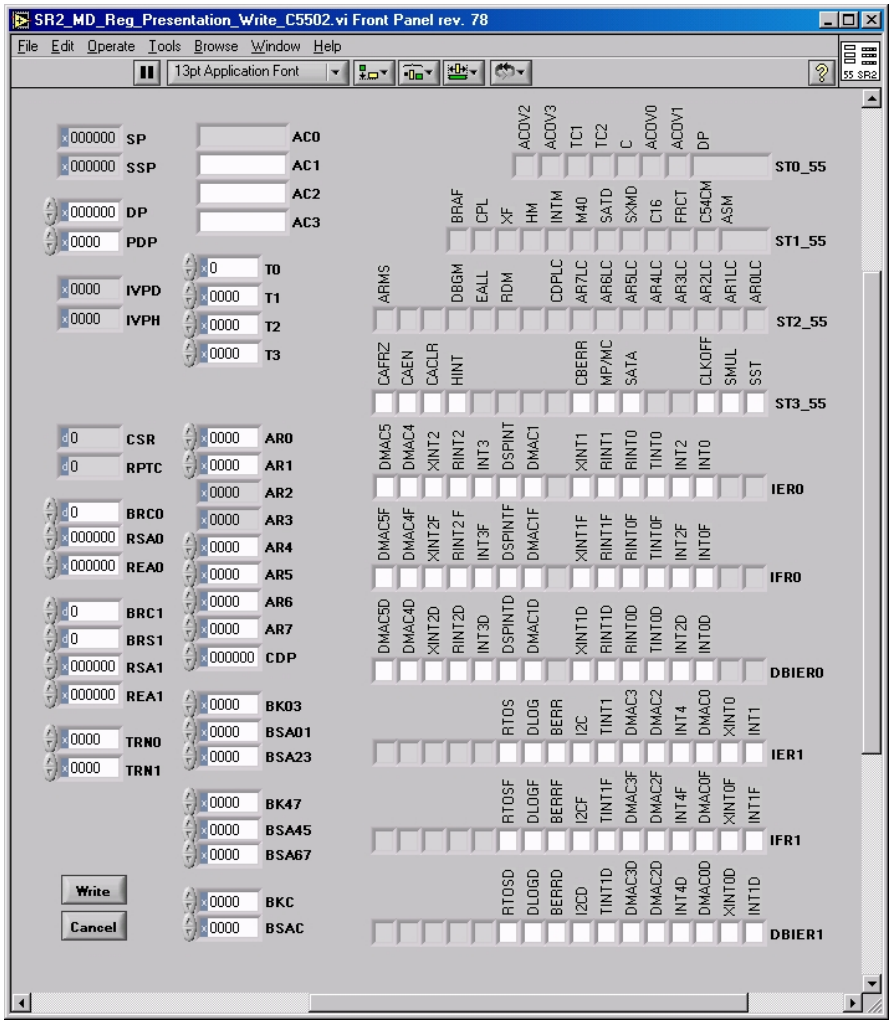


Figure 8 **Panneau de présentation de l'écriture du registre**

- **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 de *paramètres View* 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 du bloc à afficher. 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 :

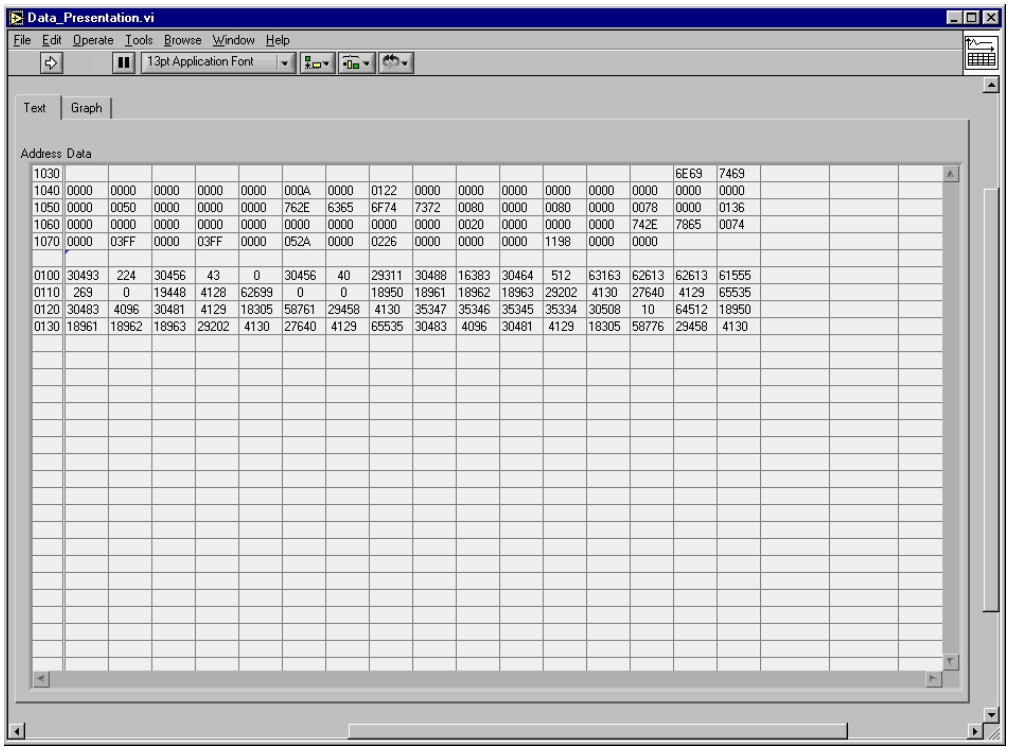
  - **Symbole** ou **adresse** est utilisé pour spécifier le début du bloc de mémoire à afficher. Le **symbole** peut être utilisé si un fichier COFF contenant le symbole a été chargé précédemment. Si **le symbole** est réglé sur une position autre que "Forcer l'adresse (pas de symbole)", l'**adresse** et l'**espace mémoire (MemSpace)** sont affichés.

sont forcées à la valeur spécifiée dans le fichier COFF pour ce symbole. La liste des symboles est effacée lorsqu'un nouveau fichier COFF est chargé, ou lorsque le Mini-Debugger est arrêté et . Elle n'est pas effacée lorsque la carte DSP est réinitialisée.
En cliquant avec le bouton droit de la souris sur le champ Symbole, il est possible de supprimer ou d'insérer un élément individuel.

*Remarque pour que **MemSpace** spécifie l'espace correct, le code DSP doit être lié en utilisant la convention habituelle de numéro de page :*
0 -> Espace de programmation
1 -> Espace de données
2 -> Espace IO

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.
- ***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. La représentation native du DSP a une largeur de 16 bits. Lors de la présentation de données sur 8 bits, les octets représentent les parties haute et basse des registres de mémoire sur 16 bits. Ils sont présentés MSB en premier et LSB ensuite. Lors de la présentation de données 32 bits (I32, U32 ou Float), l'adresse de départ n'est PAS automatiquement alignée sur l'adresse paire suivante. La première adresse est considérée comme les 16 bits supérieurs et l'adresse suivante comme les 16 bits inférieurs. Cela correspond à l'ordre standard des bits pour les données de 32 bits. Il incombe au développeur de s'assurer que l'alignement est correct, faute de quoi les données n'ont aucun sens.
- ***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	Data																
1030															6E69	7469	
1040	0000	0000	0000	0000	0000	000A	0000	0122	0000	0000	0000	0000	0000	0000	0000	0000	
1050	0000	0050	0000	0000	0000	762E	6365	6F74	7372	0080	0000	0080	0000	0078	0000	0136	
1060	0000	0000	0000	0000	0000	0000	0000	0000	0000	0020	0000	0000	0000	742E	7865	0074	
1070	0000	03FF	0000	03FF	0000	052A	0000	0226	0000	0000	0000	1198	0000	0000			
0100	30493	224	30456	43	0	30456	40	29311	30488	16383	30464	512	63163	62613	62613	61555	
0110	269	0	19448	4128	62699	0	0	18950	18961	18962	18963	29202	4130	27640	4129	65535	
0120	30483	4096	30481	4129	18305	58761	29458	4130	35347	35346	35345	35334	30508	10	64512	18950	
0130	18961	18962	18963	29202	4130	27640	4129	65535	30483	4096	30481	4129	18305	58776	29458	4130	

Figure 9 : Présentation des données (mode texte) **Présentation des données (mode texte)**

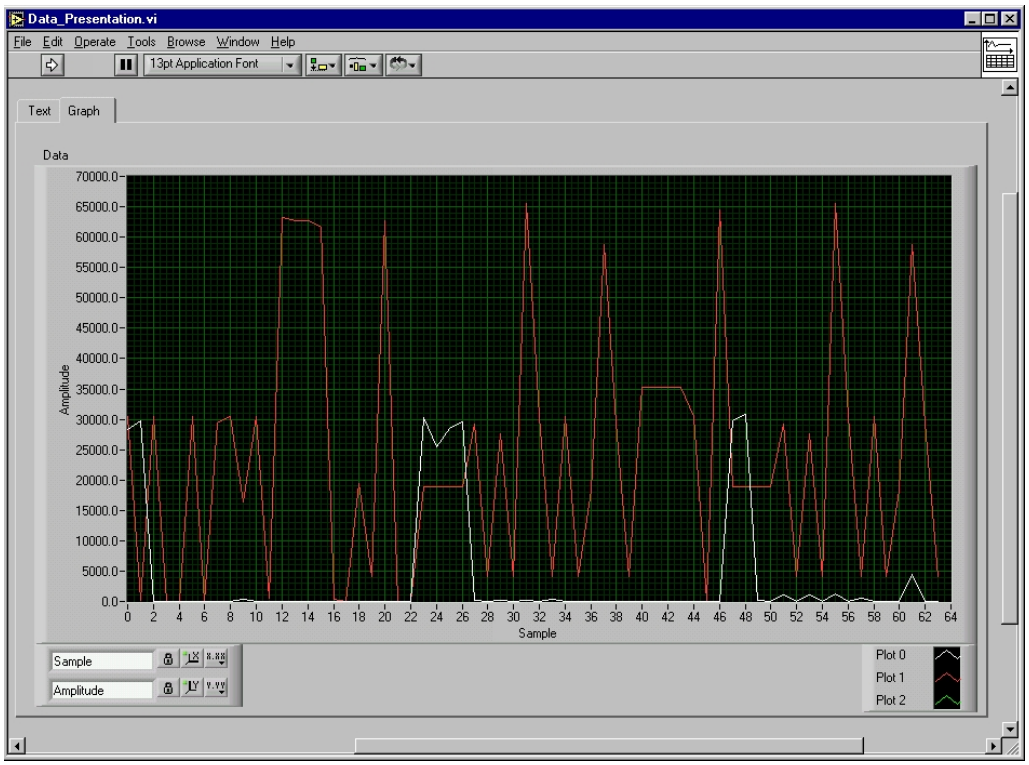


Figure 10 **Présentation des données (mode graphique)**

L'utilisateur peut choisir entre le mode *texte* (figure 9) et le mode *graphique* (figure 10) pour présentation des données de la mémoire. En *texte*, chaque bloc de mémoire demandé est présenté dans l'ordre. 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 d'un 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 n'importe quelle cellule 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.
  - Lorsque l'on tente d'écrire un nombre impair d'octets, un octet supplémentaire est ajouté au tableau d'octets. Cet octet est mis à FF_H. Cette opération est nécessaire car tous les transferts natifs sont effectués 16 bits à la fois.
  - Les données doivent être introduites 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, mais cela peut avoir une importance si les données changent en temps réel sur le DSP, car elles peuvent avoir changé entre la lecture et l'écriture.

Remarque : Lors de la présentation ou de l'écriture de mots de données de 32 bits (I32, U32 ou Float), le PC effectue 2 accès séparés (à 2 adresses mémoire successives) pour chaque mot de 32 bits 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, le PC 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 le PC serait complètement erronée.
Symétriquement, lors d'une écriture, le PC 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 DSP travaille avec une "ancienne" version de la moitié du flottant et une nouvelle version de l'autre moitié.
Ces problèmes peuvent être évités si les précautions suivantes sont respectées :
Lorsque le PC accède à un groupe de valeurs, il le fait par blocs de 256 mots de 16 bits à la fois (32 mots si la carte est connectée à pleine vitesse). Chacun de ces accès par blocs de 256 mots est atomique (le DSP ne peut effectuer aucune opération au milieu de l'accès du PC). Par conséquent, le DSP ne peut pas "interférer" au milieu d'un accès à un seul mot de 32 bits.
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 opération DSP sur ces données. Pour éviter cette situation, il suffit de rendre atomique l'opération DSP sur toute donnée de 32 bits (en désactivant les interruptions pendant la durée de l'opération), les accès sont alors atomiques des deux côtés et les données peuvent être transférées en toute sécurité 32 bits à la fois.

- **W_Flash** Permet au développeur de charger un code DSP et/ou une table de démarrage logique FPGA dans la mémoire Flash, ou d'effacer la mémoire. Une table d'amorçage indique au noyau de mise sous tension de charger le DSP et/ou le FPGA avec le code et/ou la logique spécifiés au démarrage. Les tables d'amorçage sont décrites dans d'autres sections de ce document.
  Le bouton *W_Flash* ouvre un navigateur qui permet au développeur de choisir les fichiers de code DSP (.out) et de logique FPGA (.rbt). Le fait de ne pas choisir de fichier ne préserve pas le contenu de la Flash. Si aucun fichier n'est choisi pour le code DSP ou la logique FPGA, la table de démarrage correspondante est effacée de la Flash.
  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_Address*** Indique la dernière adresse de la table d'amorçage 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 $1FFFF_H$.
  Écriture Appuyez sur ce bouton pour exécuter la programmation ou l'effacement de la mémoire 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 ou la logique FPGA.
  Annuler Ce bouton permet d'annuler toutes les opérations et de revenir au mini-débogueur.
  Taille du flash Si la Flash est détectée, ce champ indique sa taille en mots clés.

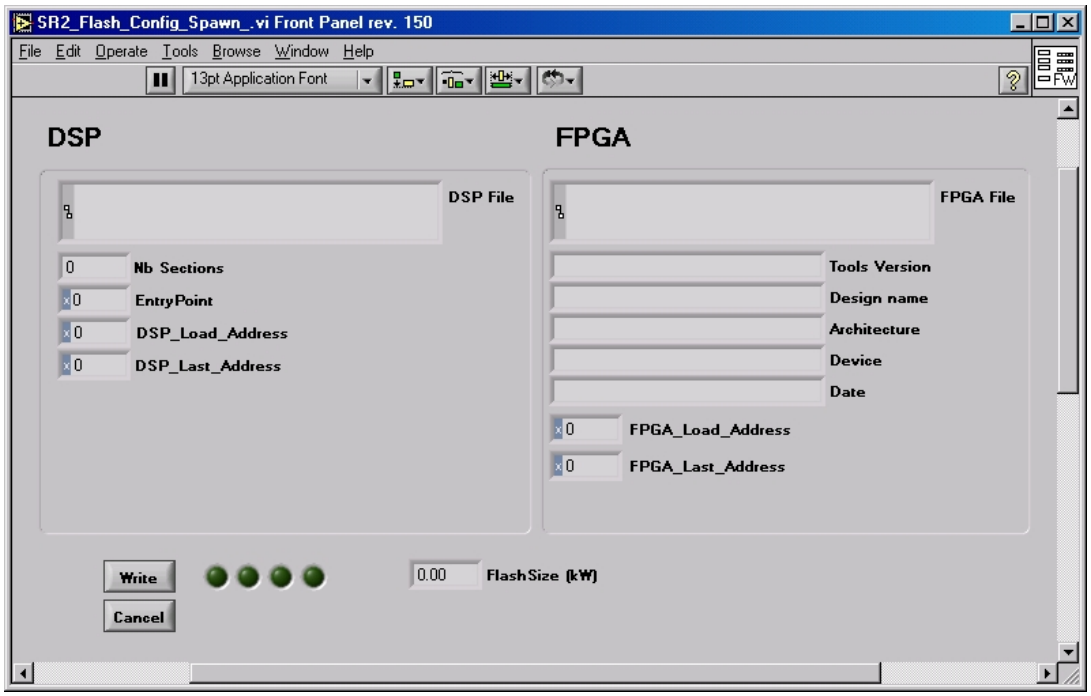


Figure 11

- **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, plutôt que 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.

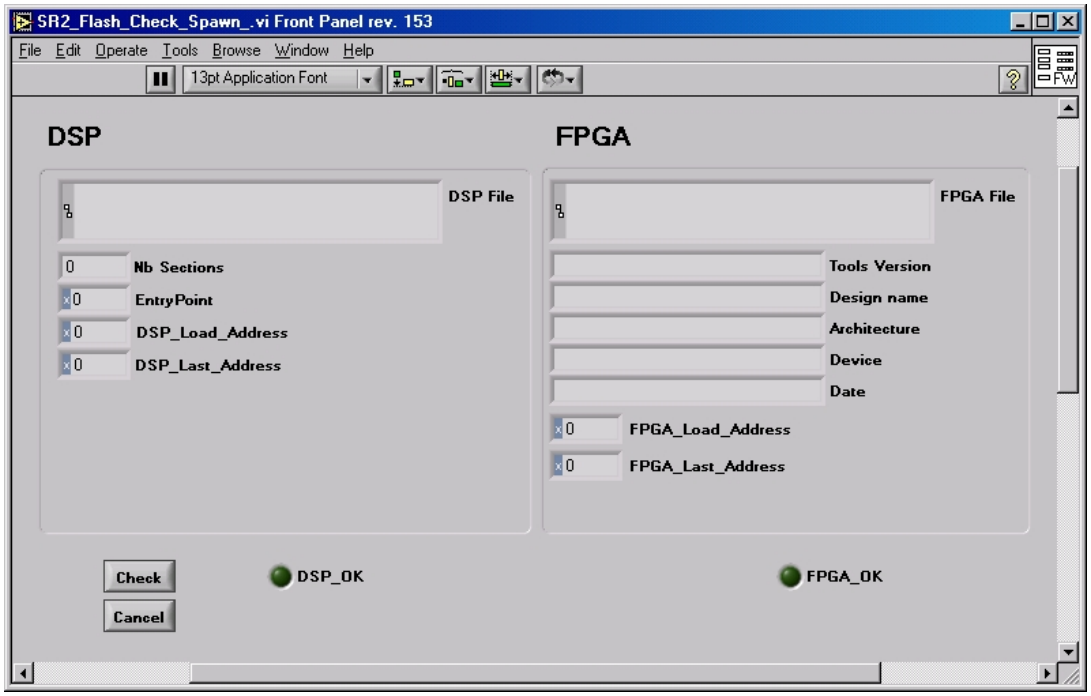


Figure 12

- **W_FPGA** Permet le téléchargement interactif d'un fichier logique FPGA (.rbt) vers le FPGA.
  Un navigateur s'affiche lorsque l'on appuie sur le bouton pour permettre au développeur de choisir le fichier .rbt.
  L'opération réinitialise systématiquement le DSP et charge le code de support FPGA requis pour les opérations de chargement FPGA.
- **Arrêter** Arrête l'exécution du mini-débogueur.
- **Réinitialisation** Force une réinitialisation de la carte et recharge le noyau Host-Download. Tout le code DSP précédemment exécuté est interrompu. Cela peut être nécessaire pour prendre le contrôle d'un DSP qui s'est planté. Le DSP n'est pas réinitialisé par défaut lorsque le mini-débogueur prend le contrôle de la carte. Cela permet au code qui a pu être chargé et exécuté par le noyau de mise sous tension de continuer sans interruption.

"Sous le capot

Communications USB

Grâce à la connexion USB, le PC peut lire et écrire la mémoire du DSP et lancer l'exécution du code DSP. La communication USB PC-DSP utilise deux mécanismes :

- Le PC utilise la ligne de commande 0, via "Vendor Requests", 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 USB, tels que *DSPState* et *USB Error Count*.
- Le PC utilise les conduites de masse à grande vitesse 2 (sortie) et 6 (entrée) pour transférer des données vers et depuis n'importe quel emplacement dans n'importe quel espace DSP, ainsi que pour lancer l'exécution du code DSP de l'utilisateur.

Les opérations effectuées à l'aide du control pipe zero sont lentes et limitées dans leur portée, 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 lorsque le code du DSP est bloqué. En particulier, ces opérations sont utilisées pour initialiser le DSP.
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 que ceux utilisant 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 est bloqué. Les opérations effectuées sur le DSP par l'intermédiaire du noyau doivent suivre un protocole décrit dans les sections suivantes.

Communication 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 **wLongueur** = N/A
HPIMove	Entrée/sortie	13_H	Lecture ou écriture de 1 à 2048 mots de 16 bits (2 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 (XHPIA). **wLength** = Nb d'octets à transférer (doit être pair) **Bloc de données** 1 à 2048 mots de 16 bits peuvent être transportés dans la phase de données de la demande. *REMARQUE : pour les transferts OUT, l'adresse stockée dans wIndex-wValue doit être prédécrémentée (c'est-à-dire qu'elle doit* *pointe sur l'adresse juste avant que le premier mot ne soit 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 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 bloc de données. Ce mot représente l'état actuel de l'USB le nombre d'erreurs (entre 0 et 15).
Reset_ErrCount	Sortie	23_H	Remet à zéro le registre de comptage des erreurs USB.

Communication 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. 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 dans n'importe quel espace (données, programme ou E/S), et permet la redirection de l'exécution à tout moment, à partir de n'importe quel emplacement, même d'une manière réentrante.

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 'exécute. Le contrôleur USB exécute cette fonction , qu'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 à tout moment un *noyau de téléchargement hôte* plus simple dans la mémoire. 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 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. Le code DSP qui répond à la commande

doit inclure cet accusé de réception dans les 5 secondes suivant l'exécution, sinon un dépassement de délai se produit. 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 que (par conception ou non) l'accusé de réception prenne beaucoup de temps à être renvoyé. La spécification USB 1.0 précise qu'une requête ne doit pas être NAKée pendant plus de 5 secondes. C'est pourquoi l'accusé de réception doit être renvoyé dans les 5 secondes. Normalement, l'accusé de réception est utilisé pour signaler l'achèvement de la fonction, mais pour les fonctions qui prennent beaucoup de temps à 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).

Table d'amorçage du FPGA

Une table de configuration du FPGA peut être programmée dans la mémoire Flash pour que le noyau de mise sous tension charge le FPGA. Cette table commence à la fin de la mémoire flash, à l'adresse $9FFFFF_{(H)}$ (adresse mot $4FFFFF_{H)}$, et remonte vers l'adresse $9C0000_{(H)}$ (adresse mot $4E0000_{H)}$. Les adresses d'octets $9C0000_H$ à $9FFFFF_{(H)}$ (4 secteurs) de la Flash sont réservées à la table de configuration du FPGA.

Octet-Adresse	Données	Description
...	Dernier mot	Dernier mot de configuration
...		
...		Mot de données
...		Mot de données
$9FFFF6_H$	3rd et 4th octets	Mot de données
$9FFFF8_H$	1st et 2nd octets	Mot de données
$9FFFFA_H$	Longueur du fichier (LSB)	Nombre de mots suivants (LSB)
$9FFFFC_H$	Longueur du fichier (MSB)	Nombre de mots suivants (MSB)
$9FFFFE_H$	3008_H	Nombre magique.

Les octets de configuration sont disposés LSB en premier. Cela signifie que l'octet inférieur du premier mot à l'adresse $9FFFF8_H$ est en fait le premier octet à être envoyé au FPGA. L'octet supérieur est le second, et ainsi de suite...

Le noyau de mise sous tension vérifie la présence du nombre magique 3008_H aux adresses d'octets $9FFFFE_H$-$9FFFFF_H$. S'il le trouve, il charge les données dans le FPGA. La présence numéro magique suffit à elle seule à ordonner au noyau de charger les données. Aucune vérification n'est effectuée pour s'assurer que les données sont valides.

Note : Le nombre magique ne doit pas être écrit si les données de configuration ne sont pas présentes dans la Flash :
Le nombre magique ne doit pas être écrit si les données de configuration ne sont pas présentes dans la Flash, car le FPGA sera alors configuré avec des données erronées. CECI PEUT ENDOMMAGER LE FPGA.

Table d'amorçage du DSP

Une table de démarrage du DSP peut être programmée dans la mémoire Flash pour que le noyau de mise sous tension charge le DSP. Cette table commence au début de la mémoire Flash, à l'adresse $800000_{(H)}$ (adresse $400000_{H)}$, et va jusqu'à l'adresse $9BFFFF_{(H)}$ (adresse $4DFFFF_{H)}$.
Les tables d'amorçage du DSP et du FPGA se développent en sens inverse pour permettre à la table d'amorçage du DSP d'avoir taille maximale de 917504 mots (28 secteurs) sans interférer avec la table d'amorçage du FPGA. Si le code DSP est plus petit, les secteurs entre les deux tables d'amorçage peuvent être utilisés pour le stockage général.

Le "Power-Up Kernel" vérifie la présence du numéro magique à l'adresse 800000_H. S'il est présent, il charge les sections suivantes dans la RAM et passe au point d'entrée code.
Aucune vérification n'est effectuée pour s'assurer que les données du tableau ont un sens. Selon toute vraisemblance, si le nombre magique est correct mais que le reste des données n'a pas été programmé ou est incohérent, le DSP se bloquera pendant la procédure de démarrage.

Remarque : il s'agit d'une table d'amorçage. Le DSP n'exécute pas de code à partir de ces emplacements. Il lit les différentes sections et les charge dans la mémoire vive de la puce, à partir de laquelle le code est ensuite exécuté.

Octet-Adresse	Données	Description
800000_H	3009_H	Nombre magique
800002_H	Nb_Sections	Nombre total de sections à charger dans la RAM
800004	Section Espace 1stSection	Espace mémoire où charger la première section
800006_H	Longueur de la section (MSB) 1stSection	MSB du nombre de mots suivants
800008_H	Longueur de la section (LSB) 1st Section	LSB du nombre de mots qui suivent
$80000A_H$	Adresse (MSB) 1stSection	MSB de l'adresse de chargement pour la section suivante
$80000C_H$	Adresse (LSB) 1stSection	LSB de l'adresse de chargement pour la section suivante
$80000E_H$	1stMot 1stSection	Mots de données
800010_H	2(nd) Mot 1stSection	...
800012_H	...	...
...	...	...
...	Dernier mot 1stSection	...
...	Section Espace 2ndSection	Espace mémoire où charger la deuxième section
...	Longueur de la section (MSB) 2ndSection	MSB du nombre de mots suivants
...	Longueur de la section (LSB) 2nd Section	LSB du nombre de mots qui suivent
	Adresse (MSB) 2ndSection	MSB de l'adresse de chargement pour la section suivante
	Adresse (LSB) 2ndSection	LSB de l'adresse de chargement pour la section suivante
...	1stMot 2ndSection	Mot de données
...	2(nd) Mot 2ndSection	Mot de données
...	...	...
...	Point d'entrée (MSB)	Après avoir chargé les différentes sections, le DSP transfère l'exécution à cette adresse.
...	Point d'entrée (LSB)	Après avoir chargé les différentes sections, le DSP transfère l'exécution à cette adresse.

Remarque : Les longueurs de section, les adresses de chargement et le point d'entrée sont des mots de 32 bits. Ils sont stockés dans le tableau avec le MSB en premier. Toutefois, il n'est pas nécessaire de les aligner sur les adresses paires.

Contraintes sur le code DSP chargé dans la mémoire flash de démarrage

Le code DSP chargé dans la Flash doit contenir un acquittement à son début (dans les 5 secondes de l'exécution). 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 entraîne à son tour l'interruption du code DSP chargé au démarrage.

Remarque : Cette exigence s'applique également au code écrit pour être chargé directement à partir du PC et exécuté à l'aide de l'interface VI SR2_K_Exec.vi. 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.

Vitesse de signalisation de l'IPH

Sur le Signal Ranger_mk2, 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 du 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++.

Benchmarks USB

La carte *Signal Ranger_mk2* possède un port USB 2.0. Ce port se connecte à haut débit (480mb/s) sur toute racine ou hub compatible USB 2.0. Il se connecte à pleine vitesse (12Mb/s) sur toute racine ou hub compatible USB 1.1.

Chronométrage à pleine vitesse

À pleine vitesse (compatible USB 1.1), les trames USB se produisent à des intervalles de 1 ms. Un transfert de données entre le PC et la carte *Signal Ranger_mk2* peut prendre jusqu'à 6 trames (jusqu'à 6 ms). Le nombre exact de trames dépend du type de logiciel du contrôleur hôte sur le PC et de sa capacité à programmer les transactions USB en attente dans le temps restant de la trame USB en cours. Cela signifie que chaque fonction des interfaces LabVIEW et C/C++ qui doit accéder au bus USB peut prendre jusqu'à 6 ms, même si la quantité de données transférées est minime. Il s'agit d'un *temps d'accès minimum*. Il s'agit généralement d'un facteur pour les fonctions qui transportent une petite quantité de données.
À pleine vitesse, le taux de transfert maximal est de 4,9 Mb/s pour la lecture et de 5,8 Mb/s pour l'écriture. Ce taux de transfert limité augmente le temps d'exécution des fonctions de l'interface. Le taux de transfert limité est généralement un facteur pour les fonctions qui transportent de grandes quantités de données.

Timing à grande vitesse

À grande vitesse (compatible USB 2.0), les microtrames USB sont 8 fois plus fréquentes qu'à vitesse maximale (toutes les 125µs). Cela signifie que le temps d'accès minimum est normalement 8 fois plus court qu'à vitesse maximale (jusqu'à 750µs). Le débit de données est limité à 18 Mb/s pour les lectures et à 22 Mb/s pour les écritures.

Critères de référence typiques

En haute vitesse sur un contrôleur hôte amélioré, le temps d'accès typique est de 475µs, et bande passante est de 18Mb/ pour les lectures, et de 22Mb/s pour les écritures.
En vitesse maximale, sur un contrôleur hôte universel, le temps d'accès typique est de 6 ms et la bande passante est de 4,9 Mb/s pour les lectures et de 5,8 Mb/s pour les écritures.

Noyau de communication DSP

Différences avec les versions précédentes

Il existe plusieurs différences entre les noyaux utilisés dans cette version de Signal Ranger et noyaux utilisés dans les versions précédentes :

Emplacement de la boîte aux lettres

La MailBox n'est située à l'adresse 1000_H. Elle se trouve maintenant dans le code noyau à l'adresse $0100_{(H)}$ (adresse d'octet $0200_{H)}$. Cela facilite la liaison du code utilisateur, qui ne doit plus éviter la MailBox à l'adresse 1000_H. Cela se reflète dans le fichier de description de la mémoire *SignalRanger_mk2.mem.*

Contenu de la boîte aux lettres

L'adresse de branchement et l'adresse de transfert dans la boîte aux lettres sont désormais des mots de 32 bits, afin de prendre en charge les transferts et les branchements au-delà des 64 premiers mots de l'espace mémoire.
Un code d'erreur a été ajouté dans la boîte aux lettres afin de renvoyer un éventuel code d'erreur (ou d'achèvement) lors de l'exécution de fonctions utilisateur. Ce code d'erreur peut être utilisé par le code utilisateur. Les fonctions du noyau renvoient toujours le même code d'achèvement (1 pour les lectures et 2 pour les écritures) car elles ne peuvent pas échouer (voir plus loin dans le texte).

Taille de la boîte aux lettres

Le champ de données de la boîte aux lettres est maintenant de 256 mots lorsque la carte DSP est connectée au PC à grande vitesse (480Mb/s). Il est encore de 32 mots lorsque la carte DSP est connectée en pleine vitesse (12Mb/s).

Adresses des différentes fonctions du noyau

Les différentes fonctions du noyau ne sont plus situées à des adresses fixes, afin de permettre un noyau plus compact. Les points d'entrée des fonctions sont différents entre le noyau de mise sous tension et le noyau de téléchargement vers l'hôte. Pour cette raison, l'accès à ces fonctions est désormais symbolique, les symboles étant automatiquement chargés dans une entrée de la *structure d'information globale de la carte à* chaque fois que le noyau est chargé ou rechargé.

Taille du noyau

La taille globale du noyau a changé. Cependant, les deux noyaux (mise sous tension et téléchargement par l'hôte) s'insèrent complètement dans l'adresse-octet $0800_{(H)}$ (adresse-mot $0400_{H)}$. Cela se reflète dans le fichier de description de la mémoire *SignalRanger_mk2.mem.*

Vue d'ensemble

Du du DSP, 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 à partir du PC attendent l'achèvement des processus critiques en temps qui s'exécutent sur le DSP, ce qui minimise les interférences entre les accès au PC et les processus en temps réel du DSP.

Trois commandes USB (K_Read, K_Write et K_Exec) sont utilisées pour déclencher 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 262 mots dans la page 0 de la mémoire vive du DSP.
L'interface DSP fonctionne sur 3 niveaux distincts :

- **Niveau 1** : Au niveau 1, noyau n'a pas encore été chargé sur le DSP. Le PC s'appuie sur 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 RAM du DSP sur puce. Par exemple, sur le C5502, le PC peut accéder à des adresses d'octets comprises entre $00FE_H$et $FFFF_H$. Ce niveau est utilisé, entre autres, pour télécharger le noyau dans la mémoire vive du DSP et lancer son exécution.
- **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.

Modes de démarrage

Comme décrit dans les sections précédentes, deux noyaux sont utilisés :
Un *noyau de mise sous tension* est téléchargé directement par le contrôleur USB embarqué peu après la mise sous tension. Ce noyau recherche dans la Flash ROM si un code utilisateur DSP et/ou un fichier de configuration FPGA sont programmés. Si c'est le cas, il charge d'abord le FPGA, puis le code utilisateur du DSP et passe à son point d'entrée. Par conséquent, lorsque le PC hôte prend le contrôle du DSP, il est déjà au niveau 2.
Plus tard, le PC hôte peut réinitialiser le DSP et recharger un *noyau de téléchargement hôte* plus limité. Ce noyau est similaire au noyau de mise sous tension, sauf qu'il ignore le contenu de la mémoire Flash ROM. De cette manière, il est possible pour le PC hôte de prendre complètement le contrôle du DSP, sans aucune interférence du code utilisateur résidant dans la Flash ROM.

État du processeur après réinitialisation

L'état du processeur après la réinitialisation dépend du mode de démarrage :

Amorçage par téléchargement de l'hôte :

Dans ce cas, le noyau effectue les initialisations suivantes :

- Place le tableau des vecteurs d'interruption à l'adresse d'octet $0x0100_H$dans la mémoire vive de la puce.
- Initialise les vecteurs suivants qui sont utilisés par le noyau :
  - **Vecteur de réinitialisation :** Pointe vers la routine de réinitialisation du noyau
  - **DSPInt vector :** Pointe vers le vecteur DSPInt du noyau. Utilisé pour gérer les communications DSP-PC.
  - **Vecteur TRAP #31 :** Pointe vers une adresse qui est ajustée dynamiquement par le contrôleur USB. Utilisé pour gérer les communications DSP-PC.
- Règle les horloges comme suit :
  - Horloge centrale du CPU : 300 MHz
  - SYSCLK1 (Périphériques rapides) : 150 MHz
  - SYSCLK2 (Périphériques lents) : 75 MHz
  - SYSCLK3 (EMIF) : 75 MHz

    - Règle la pile comme suit :
    - Double pile avec retour rapide
    - Pile système à l'adresse mot $0x8000_{(H)}$ (haut de la mémoire sur puce)
    - Pile utilisateur à l'adresse $0x7E00_{(H)}$ (512 mots pour la pile système).
- Définit le mode CPU C55x, plutôt que le mode compatible C54x (bit C54CM= 0).
- Retire la ROM d'amorçage sur puce de la carte mémoire (bit MPNMC= 1)
- Initialise l'EMIF afin que les périphériques suivants puissent être utilisés :
    - FLASH ROM
    - SDRAM
    - FPGA
- Démasquer l'interruption DSPInt pour permettre les communications DSP-PC :

Démarrage à la mise sous tension et absence de code utilisateur ou de logique FPGA dans la mémoire flash :

Les initialisations du noyau sont les mêmes que pour le démarrage par téléchargement de l'hôte.

Démarrage à la mise sous tension et logique FPGA détectée dans la mémoire flash :

Dans ce , en plus des initialisations effectuées normalement, la logique FPGA spécifiée dans la Flash est également téléchargée dans le FPGA. La logique FPGA est fonctionnelle lorsque le code utilisateur prend le contrôle du CPU. L'environnement C a également été établi. L'environnement C est en lorsque l'utilisateur prend le contrôle du DSP.
Le contenu des registres de l'unité centrale est conforme à ce qui suit :

- ST1_55 : CPL=1 M40=0 SATD=0 SXMD=1 C16=0 FRCT=0 C54CM=0
- ST2_55 : ARMS=1 RDM=0 CDPLC=0 AR[0-7]LC=0
- ST3_55 : SATA=0 SMUL=0

Démarrage à la mise sous tension et exécution du code utilisateur à partir de la mémoire flash :

Dans ce , en plus des initialisations effectuées normalement, le code utilisateur est téléchargé et exécuté. En outre, l'environnement C a été établi. L'environnement C est en vigueur au début du code utilisateur.
Le contenu des registres de l'unité centrale est conforme à ce qui suit :

- ST1_55 : CPL=1 M40=0 SATD=0 SXMD=1 C16=0 FRCT=0 C54CM=0
- ST2_55 : ARMS=1 RDM=0 CDPLC=0 AR[0-7]LC=0
- ST3_55 : SATA=0 SMUL=0

Ressources utilisées par le noyau côté DSP

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 0000_H et $07FF_H$ dans la mémoire vive du DSP. L'utilisateur doit éviter de charger du code ou de modifier l'espace mémoire en dessous de l'adresse 0800_H.
- Le PC (via le contrôleur USB) utilise l'interruption DSPInt du HPI pour demander un accès au DSP. Cette interruption déclenche à son tour TRAP#31, qui passe au code utilisateur ou à une fonction intrinsèque du noyau. Si nécessaire, le code utilisateur peut désactiver temporairement l'interruption DSPInt par le biais de son masque ou du masque d'interruption global INTM. Pendant la période où 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 localise le tableau des vecteurs d'interruption à l'adresse 0100_H. Le code utilisateur ne doit pas relocaliser les vecteurs d'interruption ailleurs.
- Le noyau de communication initialise la pile comme une double pile avec retour rapide. Les pointeurs de pile sont initialisés aux adresses $7FFF_{(H)}$ (pile système) et $7DFF_{(H)}$ (pile de données). Le code utilisateur peut déplacer temporairement les pointeurs de pile, mais doit les 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 sans problème.

Remarque : 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'agit pas d'une boucle sans fin), il retourne à une boucle sans fin dans la fonction c_int00. Il ne retourne pas au noyau.

- Le noyau utilise le DSP en mode C55x (et non en mode de compatibilité C54x).

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, 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.

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 , 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 DSP avant le branchement. La boîte aux lettres est une zone de la page 0 de la RAM du DSP accessible par HPI.
La fonction de chaque champ de la boîte aux lettres est décrite ci-dessous.

Adresse	Nom	Fonction
0100_h-0101_h	Adresse de la succursale	Adresse de branchement 32 bits (fonctions intrinsèques de lecture et d'écriture, ou fonction utilisateur)
0102_h-0103_h	Adresse de transfert	Adresse de transfert 32 bits (pour les commandes K_Read et K_Write)
0104_h	NbMots	Nombre de mots à transférer
0105_h	ErrorCode	Code d'erreur spécifique à l'application de l'utilisateur ou code d'achèvement.
0106_h-0205_h	Données	256 mots de données utilisés pour les transferts entre le PC et DSP. Seuls les 32 premiers mots sont utilisés lorsque la carte est en cours d'utilisation. connecté en tant que périphérique USB à grande vitesse.

Pour lancer une fonction DSP (code intrinsèque ou code utilisateur), le PC, via le contrôleur USB, effectue opérations suivantes :
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 mots de données à écrire au DSP.

- Le contrôleur USB place l'adresse de la branche DSP dans le champ *BranchAddress* de boîte aux lettres. Cette adresse de branche peut être une adresse de branche utilisateur (dans le cas d'une commande K_Exec) ou une adresse de fonction du noyau (dans le cas d'une commande K_Read ou K_Write).
- Si des mots de données doivent être écrits au DSP (K_Write), le contrôleur USB place ces mots dans le champ *Data* de la boîte aux lettres.
- Si des mots doivent être transférés vers ou depuis le DSP (K_Read ou K_Write), le contrôleur USB place le nombre de mots à transférer, entre 1 et 32 (connexion USB à grande vitesse), ou entre 1 et 256 (connexion USB à grande vitesse) dans le champ *NbWords* de la boîte aux lettres.
- Si des mots doivent être transférés vers ou depuis le DSP (K_Read ou K_Write), le contrôleur USB place l'adresse de transfert du DSP dans le champ *TransferAddress* de la boîte aux lettres.
- Le contrôleur USB efface le signal HINT (host interrupt), qui sert d'accusé de réception de la fonction DSP.
- Le contrôleur USB envoie une interruption DSPInt au DSP, ce qui force le DSP à passer à la fonction intrinsèque ou utilisateur.

Lorsque le DSP reçoit l'interruption DSPInt du HPI, le noyau effectue les opérations suivantes :
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 quand - la DSP est interruptible, elle :

- Récupère l'adresse de la branche dans le champ *BranchAddress* de la boîte aux lettres et la réinscrit dans le vecteur de l'interruption logicielle TRAP#31.
- Efface INTM. À partir de cet instant, le processeur de signal numérique redevient interruptible et peut exécuter une routine de service d'interruption (ISR) critique de l'utilisateur. Si aucun ISR d'utilisateur n'est en attente, le DSP commence à exécuter la fonction demandée par le PC (fonction intrinsèque ou d'utilisateur).
- Déclenche l'interruption TRAP #31, qui amène le DSP à passer à la fonction ou au code spécifié par *BranchAddress*.
- Si des mots de données doivent être transférés depuis le PC (K_Write), la fonction lit ces mots dans le champ *Data* de la boîte aux lettres et les place aux adresses DSP requises.
- Si des mots de données doivent être transférés au PC (K_Read), ces mots sont écrits par le DSP dans le champ *Data* de la boîte aux lettres.
- Si un code d'erreur ou d'achèvement doit être renvoyé au PC, la fonction DSP doit mettre à jour le champ *ErrorCode* de la boîte aux lettres.
- Après l'exécution de la fonction demandée et la mise à jour du champ *ErrorCode*, 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. Si l'accusé de réception n'est pas renvoyé dans les 5 secondes suivant la demande de transfert, la demande est interrompue sur le PC et une erreur est émise.
- Si des mots de données doivent être transférés au PC, dès réception du signal HINT, le contrôleur USB récupère ces mots dans le champ *Data* de la zone de la boîte aux lettres et les envoie au PC dans la phase de données de la demande (K_Read).
- Une fois que la fonction DSP a terminé son exécution, une instruction RETI doit être utilisée pour renvoyer le contrôle au noyau ou au code utilisateur précédemment exécuté, après l'accusé de réception. Cela n'est toutefois pas nécessaire et le code utilisateur peut continuer à s'exécuter sans retour au noyau si c'est ce que l'utilisateur souhaite. Dans ce , les accès ultérieurs au PC sont toujours autorisés, ce qui signifie que le noyau est réentrant.

Note : La taille du champ de données de la boîte aux lettres est de 256 mots : La taille du champ de données de la boîte aux lettres est de 256 mots. Toutefois, la taille maximale d'un bloc de données transféré dépend du fait que la carte Signal Ranger_mk2 est ou non connectée au PC à l'aide d'une connexion USB à haut débit (12Mb/s) ou d'une connexion USB à haut débit.

(480Mb/s). La carte est compatible avec l'USB 2.0 et s'énumère à grande vitesse sur une racine ou un concentrateur USB 2.0. Lorsque la carte est connectée à haut débit, les transferts sont effectués 256 mots à la fois. Lorsque la carte est connectée à vitesse maximale, les transferts sont effectués à raison de 32 mots à la fois. Dans ce cas, seuls les 32 premiers mots du champ de données de la boîte aux lettres sont utilisés.

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.
En général, le noyau est utilisé au niveau 2 pour télécharger et lancer un programme utilisateur principal, qui peut ou non revenir 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 du DSP, ou pour forcer l'exécution d'autres fonctions utilisateur du DSP, 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 (RETI) est utilisée à la fin de la fonction utilisateur, 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 a pour seul but de signaler à la commande hôte initiatrice que la fonction DSP demandée a été exécutée 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 la demande d'accès au PC, le DSP n'est ininterruptible que pendant une très courte période (entre la prise de l'interruption DSPInt et le passage au début de la fonction utilisateur ou intrinsèque - l'équivalent de 10 cycles). Par conséquent, le processus d'accès au PC ne bloque pas les tâches critiques qui pourraient être exécutées pendant l'interruption sur le DSP (gestion des E/S analogiques, par exemple).

Lors d'un transfert vers et depuis le DSP, le DSP est également ininterrompu pendant le transfert effectif de chaque bloc de 32 mots (ou de 256 mots pour une connexion USB à grande vitesse). Le temps d'ininterruption réel dépend du périphérique cible. Pour une lecture ou une écriture de/vers la RAM DSP sur puce, le temps ininterrompu en cycles CPU est égal au nombre de mots transférés. Le fait de rendre le transfert ininterrompu présente l'avantage que tout transfert jusqu'à 32 mots (pleine vitesse) ou 256 mots (haute vitesse) est atomique du point de vue du DSP. En particulier, cela garantit que lors du transfert de mots doubles, la partie haute et la partie basse du mot sont transférées simultanément. Cependant, la taille du bloc est suffisamment petite pour que le transfert soit rapide et ne pose pas de problème dans la plupart des situations.

Lors du transfert de données vers et depuis un périphérique lent tel que la SDRAM, le temps ininterrompu peut atteindre 240ns/mot. Si la connexion USB est à grande vitesse, la taille du bloc est de 256 mots. Dans ce cas, le temps d'interruption peut être si long qu'il interfère avec le service d'autres interruptions dans le système. Dans une telle situation, deux actions peuvent être entreprises pour résoudre le problème :

- Soit fractionner le transfert à haut niveau de manière à ce que seuls de petits blocs soient transférés à la fois.
- Ou utiliser une fonction DSP personnalisée pour gérer le transfert à la place de la fonction intrinsèque du noyau. Concevez cette fonction personnalisée de manière à ce qu'elle soit non atomique et interruptible. Le dossier des exemples contient un exemple d'une telle fonction personnalisée.

Remarque : Au début d'une fonction utilisateur DSP, il est nécessaire de protéger (stocker) tous les registres DSP qui sont utilisés dans la fonction, et de les restaurer juste avant le retour. Comme la fonction est lancée par l'intermédiaire de l'interruption DSPInt, elle est exécutée de manière asynchrone par rapport au code principal (noyau ou code utilisateur). La responsabilité de la protection des registres doit alors être assumée par la fonction utilisateur appelée. La situation est la même que pour toute fonction d'interruption (il s'agit en fait d'une fonction d'interruption). Toutes les fonctions intrinsèques du noyau assurent cette protection et n'interfèrent pas avec le code utilisateur.
Lorsque la fonction est écrite en C, elle doit être déclarée avec le qualificatif "interrupt". Sinon, tous registres utilisés dans la fonction risquent de ne pas être protégés.
Voir les sections sur le développement du code DSP pour plus de détails.

Lecture et écriture de la mémoire

Le noyau comprend 6 fonctions intrinsèques (*ReadMem*, *WriteMem*, *ReadProg*, *WriteProg*, *ReadIO* et *WriteIO*), qui font partie et résident dans la mémoire au moment où le noyau s'exécute. Ces 6 fonctions permettent au PC de lire ou d'écrire dans la mémoire du DSP.

ReadMem, ReadProg, ReadIO :

Ces fonctions lisent *nn* mots successifs de la mémoire du DSP à l'adresse *aaaaaaaa*
(1<=nn<=32 ou *1<=nn<=256*, selon la vitesse de la connexion USB).
Pour lancer l'exécution de l'une de ces fonctions, le PC (via le contrôleur USB et linvocation de K_Read) procède comme suit :

- Écrit le point d'entrée 32 bits de la fonction (ReadMem, ReadProg ou ReadIO respectivement) dans le fichier *BranchAddress* dans la boîte aux lettres.
- Écrit l'adresse de transfert DSP 32 bits *aaaaaaaa*, l'*adresse de transfert* dans la boîte aux lettres.
- Écrit le *nombre nn* de mots à transférer dans *NbWords* dans la boîte aux lettres.
- Attribue la valeur 1 au champ *ErrorCode* de la boîte aux lettres
- Efface le signal *HINT.*
- Envoie une interruption *DSPInt* au DSP.

En réponse à ces actions, le noyau effectue les opérations suivantes :

- Reprend le début de la fonction et redevient interruptible.
- Repousse tous les registres utilisés dans la fonction au sommet de la pile (protection du contexte).
- Lit l'adresse de début de transfert *aaaaaaaa* dans le champ *TransferAddress* de la boîte aux lettres.
- Lit *nn* mots dans la mémoire du DSP et les écrit dans le champ *Data* de la boîte aux lettres.
- Place l'adresse qui suit directement le bloc qui vient d'être transféré dans la variable *TransferAddress* de la boîte aux lettres. De cette manière, les accès ultérieurs ne pas réinitialiser l'adresse pour commencer à transférer les mots suivants.
- Rétablit le contexte.
- Affecte le signal *HINT*, qui signale la fin de l'opération.
- Retourne à l'appelant.

À ce stade, le contrôleur USB effectue les opérations suivantes :

- Lit les *nn* mots du champ de données de la boîte aux lettres et les envoie au PC.
- Lit le champ *ErrorCode* de la boîte aux lettres et le renvoie au PC. Ce code d'erreur est 1 pour les lectures.

Le tuyau USB (tuyau 6) qui supporte l'échange a une taille de 64 octets (32 mots) pour une connexion USB à pleine vitesse, et une taille de 512 octets (256 mots) pour une connexion à grande vitesse. Pour les transferts supérieurs à la taille du tuyau, le transfert est divisé en blocs de la taille du tuyau. Le contrôleur USB invoque la fonction et exécute les opérations ci-dessus pour chacun des segments de la taille du tuyau. L'affirmation du signal *HINT* à la fin de chaque segment déclenche le transfert du bloc vers le PC.

Dans ce , le champ *ErrorCode* de la boîte aux lettres n'est mis à un par le contrôleur USB embarqué qu'avant le premier segment. La valeur du champ *ErrorCode* qui est renvoyée au PC est la dernière valeur mise à jour par le DSP avant le dernier acquittement.

Note : Le nombre de mots à lire ne doit pas être égal à zéro : Le nombre de mots à lire ne doit pas être nul.

WriteMem, WriteProg, WriteIO :

Ces fonctions écrivent *nn* mots successifs dans la mémoire du DSP, à partir de l'adresse *aaaaaaaa* (1<=nn<=32 ou *1<=nn<=256*, selon la vitesse de la connexion USB).
Pour lancer l'exécution de l'une de ces fonctions, le PC (via le contrôleur USB et linvocation de K_Write) procède comme suit :

- Place les *nn* mots à écrire dans la mémoire DSP dans le champ *Données* de la boîte aux lettres.
- Écrit le point d'entrée de la fonction (WriteMem, WriteProg ou WriteIO respectivement) dans le fichier Variable *BranchAddress* dans la boîte aux lettres.
- Écrit l'adresse de transfert du DSP (*aaaaaaaa*) dans la variable *TransferAddress* de la boîte aux lettres.
- Écrit le *nombre nn* de mots à transférer dans le champ *NbWords* de la boîte aux lettres.
- Attribue la valeur 2 à la variable *ErrorCode* de la boîte aux lettres.
- Efface le signal *HINT.*
- Envoie une interruption *DSPInt* au DSP.

En réponse à ces actions, le noyau effectue les opérations suivantes :

- Reprend le début de la fonction et redevient interruptible.
- Repousse tous les registres utilisés dans la fonction au sommet de la pile (protection du contexte).
- Lit l'adresse de transfert dans la variable *TransferAddress* de la boîte aux lettres.
- Lit les *nn* mots du champ de *données* de la boîte aux lettres et les réinscrit dans la mémoire du DSP à l'adresse de transfert *aaaaaaaa.*
- Place l'adresse qui suit directement le bloc qui vient d'être transféré dans la variable *TransferAddress* de la boîte aux lettres. De cette manière, les accès ultérieurs ne pas réinitialiser l'adresse pour commencer à transférer les mots suivants.
- Rétablit le contexte.
- Affecte le signal *HINT*, qui signale la fin de l'opération.
- Retourne à l'appelant.

À ce stade, le contrôleur USB effectue les opérations suivantes :

- Lit le champ *ErrorCode* de la boîte aux lettres et le renvoie au PC. Ce code d'erreur est 2 pour les écritures, car les fonctions du noyau n'utilisent pas ce code.

Le tuyau USB (tuyau 2) qui supporte l'échange a une taille de 64 octets (32 mots) pour une connexion USB à pleine vitesse, et une taille de 512 octets (256 mots) pour une connexion à grande vitesse. Pour les transferts plus importants que la taille du tuyau, le transfert est divisé en blocs de la taille du tuyau. Le contrôleur USB invoque la fonction et exécute les opérations ci-dessus pour chacun des segments de la taille du tuyau. L'affirmation du signal *HINT* à la fin de chaque segment déclenche le transfert du bloc suivant à partir du PC.
Dans ce , le champ *ErrorCode* de la boîte aux lettres n'est mis à 2 par le contrôleur USB embarqué qu'avant le premier segment. La valeur du champ *ErrorCode* qui est renvoyée au PC est la dernière valeur mise à jour par le DSP avant le dernier acquittement.

Note : Le nombre de mots à lire ne doit pas être égal à zéro : Le nombre de mots à lire ne doit pas être nul.

Utilisation des requêtes K_Read et K_Write pour les fonctions utilisateur

En principe, les requêtes K_Read et K_Write ne sont utilisées que pour invoquer les 6 fonctions intrinsèques du noyau. Cependant, rien n'empêche le développeur d'utiliser ces requêtes pour invoquer une fonction utilisateur du noyau.

fonction. 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.
Vous trouverez plus de détails à ce sujet dans description du VI *SR2_Base_User_Move_Offset* dans la section *Interface LabView*, et dans la fonction *SR2_DLL_User_Move_Offset_I16* dans la section *Interface C/C++.*

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 requêtes 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.

Protocole de communication à grande vitesse

La communication à grande vitesse avec la carte DSP est assurée par l'utilisation des tuyaux 2 (sortie) et 6 (entrée). Le PC utilise le bulk pipe 2 pour envoyer des paquets au DSP, et le pipe 6 pour recevoir des paquets du DSP. La communication par ces tuyaux doit suivre le protocole décrit ci-dessous.

Le PC hôte peut déclencher trois types d'opérations sur le DSP :

- K_Read : L'exécution d'une fonction du DSP qui renvoie des données du DSP via la boîte aux lettres.
- K_Write : L'exécution d'une fonction DSP qui envoie des données au DSP via la boîte aux lettres.
- K_Exec : L'exécution d'une fonction DSP sans transport de données.

Le noyau fournit des fonctions intrinsèques permettant d'effectuer des opérations de lecture et d'écriture de base à partir de/vers n'importe quel emplacement dans n'importe quel espace mémoire du DSP. Cependant, l'utilisateur peut également fournir d'autres fonctions de lecture et d'écriture qui utilisent le même mécanisme K_Read et K_Write. Cela permet au développeur de remplacer les fonctions du noyau par du code DSP fournissant des fonctions plus spécialisées. Par exemple, les fonctions utilisateur de lecture et d'écriture peuvent également gérer les opérations de synchronisation et de pointeur nécessaires à la gestion d'une FIFO sur le DSP.

Les trois types d'opérations décrits ci-dessus se déroulent de la même manière :

- Le PC envoie d'abord un paquet de configuration à la carte DSP, indiquant divers éléments d'information tels que le sens du transfert, l'adresse de transfert, l'adresse de la branche DSP... etc. La structure du paquet de configuration est la même pour les trois types de commande.
- Pour un K_Write, le PC envoie les données à la carte DSP, pour un K_Read, le PC collecte données renvoyées par la carte DSP. Pour un K_Exec, cette étape est sautée.
- Le PC attend ensuite un paquet d'achèvement de la carte DSP, indiquant que l'opération demandée est terminée.

Paquet de configuration

Le paquet d'installation contient les informations suivantes :

- Adresse de branchement 32 bits du DSP ***Adresse de branchement***. Il s'agit de l'adresse de la mémoire du DSP où se trouve la fonction à exécuter (lecture, écriture ou fonction utilisateur).
- Adresse de transfert DSP 32 bits ***Adresse de transfert***. C'est l'adresse dans la mémoire du DSP où les données doivent être lues ou écrites. Pour une exécution sans transfert de données (K_Exec), l'adresse est ignorée.
- Nombre ***de mots du*** compte de transfert sur 16 bits. Pour une exécution sans transfert de données (K_Exec), le nombre de transferts est ignoré. Le nombre de transferts est exprimé en mots. Il doit être compris entre 1 et 32768.

- Un code ***OpType de*** 16 bits pour ***le*** type d'opération. Ce code est 0 pour un K_Exec, 1 pour un K_Read, 2 pour un K_Write.

Octet Nb	Données
0	BranchAddress (LSB)
1	BranchAddress (octet 1)
2	BranchAddress (octet 2)
3	BranchAddress (MSB)
4	Adresse de transfert (LSB)
5	Adresse de transfert (octet 1)
6	Adresse de transfert (octet 2)
7	Adresse de transfert (MSB)
8	NbMots (LSB)
9	NbMots (MSB)
10	OpType (LSB)
11	OpType (MSB)

Remarque : Les données du paquet de configuration sont copiées dans les champs correspondants de la boîte aux lettres par contrôleur USB avant que la fonction DSP résidant à BranchAddress ne soit appelée. Ce sont les mêmes champs qui sont présents au début de la boîte aux lettres. Le champ OpType du paquet Setup correspond au champ ErrorCode de la boîte aux lettres. Par conséquent, dans le cas d'une fonction DSP utilisateur, ce sont les données qui se trouvent dans la boîte aux lettres lorsque la fonction utilisateur s'exécute.

Remarque : Dans le cas d'un transfert de données multi-paquets (paquets de 32 octets en Full-Speed et de 256 octets en High-Speed), les champs ci-dessus de la boîte aux lettres ne sont mis à jour qu'avant le PREMIER appel de la fonction DSP, correspondant au transfert du premier paquet. Les appels ultérieurs correspondant aux paquets suivants ne modifient pas ces champs. Toutefois, cette affirmation n'est valable que si le transfert de plusieurs paquets est inférieur à 32768 mots et si le transfert ne franchit pas une limite de 64 kWords. En effet, les transferts de plus de 32768 mots sont segmentés à haut niveau en transferts de 32768 mots. Au début de chacun de ces transferts, l'ensemble du contenu de la boîte aux lettres est mis à jour. De même, les transferts qui traversent une limite de 64 mots sont divisés en deux transferts qui ne chevauchent pas la limite. Pour chacun de ces transferts, tout le contenu de boîte aux lettres est mis à jour.

Dossier d'achèvement

Après l'opération, la carte DSP renvoie un paquet d'achèvement au PC, indiquant que l'opération est terminée. Ce paquet d'achèvement est constitué comme suit :

- Un **code d'erreur de** 16 bits. Ce code est le contenu du champ *ErrorCode* de la boîte aux lettres qui a été mis à jour par la fonction DSP avant l'envoi du dernier accusé de réception au contrôleur USB. Ce code d'erreur doit être utilisé d'une manière spécifique à l'application. Par conséquent, l'utilisateur doit définir les codes d'erreur pertinents, le cas échéant.

Octet Nb	Données
0	ErrorCode (LSB)
1	ErrorCode (MSB)

Remarque : Dans le cas d'un transfert de données en plusieurs paquets (K_Read ou K_Write), le code d'erreur renvoyé au PC dans le cadre du paquet d'achèvement représente le contenu du champ ErrorCode de la boîte aux lettres avant l'accusé de réception correspondant au DERNIER appel de la fonction DSP.

Remarque : Ce code d'erreur ne représente pas les conditions qui peuvent se produire si le transfert ne se termine pas (par exemple si le noyau n'est pas en ligne ou si le code DSP est bloqué au moment du transfert), dans de telles conditions, l'opération PC se termine de manière anormale et indique une erreur USB. La transmission du code d'erreur ne peut avoir lieu que si le transfert se déroule normalement. Un code d'erreur peut être utilisé pour signaler des conditions spécifiques à l'application, telles que l'impossibilité pour le DSP d'accepter des données à ce moment-là parce que la FIFO est pleine, ou l'impossibilité d'exécuter une fonction en raison de l'état dans lequel il se trouve au moment où il reçoit la commande.

Note : Dans le cas d'une transaction K_Read ou K_Write, que le code d'achèvement soit ou non réglé sur une valeur significative, et que le DSP soit ou non capable d'absorber ou de fournir les données, la fonction DSP qui gère le transfert doit envoyer un accusé de réception pour chaque transfert élémentaire de la taille d'un tube de la transaction complète. Par exemple, si la carte Signal_Ranger_mk2 est connectée au PC en tant que dispositif à grande vitesse, la taille du tube est de 64 octets (32 mots). Pour un K_Read de 68 mots, la transaction complète est segmentée en deux transactions de 32 mots, suivies d'une transaction de 4 mots. Dans ce cas, le DSP doit acquitter chacun des trois transferts de segments élémentaires, même s'il n'est pas en mesure de fournir les données. Dans ce cas, un code d'erreur approprié peut être utilisé pour indiquer au PC, d'une manière spécifique à l'application, que les données collectées à partir de la boîte aux lettres ne sont pas valides. Le code d'erreur est lu par le contrôleur USB après le dernier accusé de réception du DSP. Dans l'exemple ci-dessus, le DSP peut définir le code d'erreur à n'importe quel stade du transfert. Cependant, la valeur renvoyée au PC est la dernière valeur mise à jour avant le dernier accusé de réception.

Code de support DSP

Code d'aide au pilotage et à la programmation de la mémoire flash des DSP

Plusieurs niveaux de code d'assistance sont fournis aux développeurs :

- Une bibliothèque de pilotes DSP est fournie aux développeurs qui souhaitent inclure des fonctions de programmation Flash dans leur code DSP. Ce code pilote est décrit ci-dessous.
- Une application DSP de programmation Flash est fournie 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 *SR2_Flash_Support.out*. Ce code DSP est chargé et exécuté par les fonctions d'interface qui requièrent sa présence.

Vue d'ensemble du pilote Flash

Un pilote DSP est fourni pour faciliter le développement du code DSP de l'utilisateur. Ce pilote se présente sous la forme d'une bibliothèque nommée **SR2FlashBootDriver.lib**.
Le pilote se trouve dans le fichier *C:\NProgram Files\NSignalRanger_mk2\NDSPSupport_Code*. Il suffit de décompresser le fichier pour accéder à son contenu.

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

Ces fonctions permettent la lecture, l'effacement et l'écriture séquentielle sur la mémoire FLASH. Le pilote utilise une mémoire tampon FIFO d'écriture logicielle, de sorte que les fonctions d'écriture n'ont pas à attendre que chaque opération d'écriture soit terminée.

Les fonctions de lecture sont exécutées de manière asynchrone et sont très rapides. Les écritures sont séquentielles et sont effectuées sous l'interruption INT1. Le temps d'écriture typique est de 60 µs par mot. L'effacement est asynchrone et peut être assez long (typ 0,5s/secteur, max 3,5s 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 n'est pas adressable par octet. 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. Ceci est fait pour assurer la cohérence avec le reste du logiciel d'interface Signal_Ranger_mk2 qui n'utilise que des adresses d'octets. Comme la Flash n'est pas adressable par octet, les adresses par octet transmises au pilote sont divisées par deux en interne pour pointer vers les mots de 16 bits corrects. Les adresses renvoyées par le pilote sont multipliées par deux en interne avant d'être renvoyées.*

Les lectures sont très simples. Elles sont effectuées de manière asynchrone à l'aide de la fonction **SR2FB_Read**. Cette fonction renvoie le contenu d'une adresse 32 bits.

Les écritures sont effectuées séquentiellement à l'aide de la fonction **SR2FB_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 **SR2FB_SetAddress** ou de la fonction **SR2FB_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.

Remarque : Contrairement aux générations précédentes de dispositifs Flash utilisés dans les cartes Signal Ranger, le dispositif Flash utilisé dans Signal_Ranger_mk2 ne peut pas être programmé de manière incrémentielle. Cela signifie qu'un emplacement de mot qui a été précédemment programmé DOIT être effacé avant d'être reprogrammé. Cela est vrai même si l'opération de reprogrammation ne vise qu'à transformer certains des "1" restants en "0".

La fonction **SR2FB_WritePrepare** efface au préalable tous les secteurs commençant à l'adresse spécifiée et contenant au moins le nombre séquentiel de mots spécifié. Comme l'effacement est effectué secteur par secteur, cette fonction peut effacer plus de mots que ceux spécifiés à la fonction. La fonction initialise ensuite le registre **FB_WriteAddress** à l'adresse de début spécifiée, de sorte que l'écriture suivante soit effectuée au début du segment de mémoire spécifié.

La fonction **SR2FB_Write** n'attend pas que l'écriture soit terminée. Elle place simplement le mot 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 **SR2FB_FIFOState.**

Configuration du pilote

Remarque : Contrairement aux pilotes Flash fournis dans le passé avec les versions antérieures des cartes Signal Ranger, ce pilote nécessite '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 qui a configuré l'environnement C avant d'appeler l'une des fonctions (voir les détails ci-dessous).

- Toutes les fonctions du pilote définies ci-dessous sont contenues dans la bibliothèque **SR2FlashBootDriver.lib.** Le code utilisateur doit être lié à cette bibliothèque pour fonctionner correctement (la bibliothèque doit être ajoutée aux fichiers source du projet).
- Lorsque la bibliothèque est liée à un projet C, le projet doit utiliser le modèle de mémoire LARGE. Cela est nécessaire pour pouvoir accéder à tous les secteurs de la FLASH, qui s'étendent sur plusieurs pages de 64k.
- Étant donné que les écritures sont effectuées sous des interruptions INT1, le vecteur d'interruption INT1 doit être initialisé à l'étiquette "**SR2FBINT**". Ce label est le point d'entrée de la routine d'interruption INT1. La routine d'interruption INT1 est définie dans la bibliothèque **SR2FlashBootDriver.lib.** Voir l'exemple de code C fourni avec la carte pour un exemple de configuration du vecteur d'interruption.
- Un fichier d'en-tête nommé **SR2_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 Fonction **SR2FB_Init.**

C-Environnement

Comme les fonctions sont appelables en C, le pilote suppose la présence de l'environnement C. Cet environnement est en vigueur par défaut lors de l'appel d'une fonction du pilote à partir d'un code écrit en C. Cet environnement est en vigueur par défaut lors de l'appel d'une fonction du pilote à partir d'un code écrit en C. Toutefois, lors de l'appel de ces fonctions à partir d'un code écrit en assembleur, les exigences suivantes doivent être respectées :
Le contenu des registres de l'unité centrale doit être conforme à ce qui suit :

- ST1_55 : CPL=1 M40=0 SATD=0 SXMD=1 C16=0 FRCT=0 54CM=0
- ST2_55 : ARMS=1 RDM=0 CDPLC=0 AR[0-7]LC=0
- ST3_55 : SATA=0 SMUL=0

Conformément aux règles de l'environnement C, les registres suivants ne sont pas protégés par les fonctions du pilote et peuvent être modifiés par n'importe laquelle de ces fonctions :
AC[0-3], XAR[0-4], T[0,1], ST0_55, ST1_55, ST2_55, ST3_55, RPTC, CSR, BRC0, BRC1, BRS1, RSA0, RSA1, REA0 et REA1.

Pour plus d'informations sur les règles d'appel C, voir la documentation correspondante de Texas Instrument.

Structures de données

FB_WriteFIFO

FB_WriteFIFO est un tampon de 32 mots 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 que par la fonction **SR2FB_Write**. Elle n'est vidée que par la routine de service d'interruption INT1.

FB_WriteAddress

FB_WriteAddress est un mot non signé de 32 bits qui contient toujours l'adresse du prochain mot à écrire. **FB_WriteAddress** peut être initialisé par la fonction **SR2FB_SetAddress** ou la fonction **SR2FB_WritePrepare**. Après chaque écriture, le registre **FB_WriteAddress** est automatiquement incrémenté. Cette incrémentation se produit dans la routine d'interruption INT1. Par conséquent, pour 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 à l'aide de la fonction **SR2FB_FIFOState** fonction.

FB_WriteEraseError (erreur d'écriture)

FB_WriteEraseError est un entier qui contient divers bits d'état d'erreur. Il est renvoyé par plusieurs fonctions, notamment **SR2FB_SetAddress**, **SR2FB_FIFOState**, **SR2FB_WritePrepare** et **SR2FB_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
SR2FB_ErrorClear. L'exécution de **SR2FB_Init** efface également le mot d'erreur.

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
													SE	WP	WE

WE ***Erreur d'écriture*** 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 ROM ne fonctionne pas correctement.

WP ***Écriture en cours*** Lorsqu'il vaut un, ce bit indique que des écritures sont en . 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 ***Erreur d'effacement de secteur*** Ce bit indique qu'une erreur s'est produite au cours de l'opération d'effacement sectoriel demandée. 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, le comportement habituel est que le processus se bloque indéfiniment. Le bit WP reste à un indéfiniment. Il n'y a pas de délai pour débloquer 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 ***SR2FB_Write*** *se bloque.*

Fonctions de l'utilisateur

unsigned long SR2FB_Init ()

Initialise le pilote et réinitialise le circuit Flash. Elle détecte la ROM Flash et renvoie la taille de la mémoire en mots, ou zéro si le circuit n'est pas détecté. Cette fonction doit être au moins une fois avant toute autre fonction du pilote. Cette fonction peut être appelée pour réinitialiser le pilote.

Entrée : pas
d'entrée Sortie :
pas de sortie
Retour : La taille de la ROM flash en mots

unsigned int SR2FB_SetAddress(unsigned long FB_WAddress)

Cette fonction attend que toutes les écritures en attente soient terminées. Elle place ensuite le pointeur **FB_WriteAddress** sur l'adresse 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 long **FB_WAddress** (byte) : il s'agit de l'adresse 32 bits pour la prochaine écriture Output : no output
Retourner : L'erreur d'écriture actuelle

unsigned short SR2FB_FIFOState(unsigned int *FB_FIFOCount, unsigned long *FB_WAddress)

Cette fonction renvoie le nombre de mots 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.*

(L'*adresse FB_Waddress* est convertie en adresse d'octets en interne avant d'être renvoyée). 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 . Pour vérifier que toutes les écritures ont bien été , il convient de vérifier le bit WP dans le registre d'état **FB_WriteEraseError.***

Entrée : ushort * FB_FIFOCount : Il s'agit du pointeur vers une variable pour FIFOCount. sortie
unsigned long* WAddress : Il s'agit du pointeur vers une variable de 32 bits Sortie FB_WAddress (sp(2))
Sortie : ushort FB_FIFOCount et unsigned long FB_WAddress Retour :
L'erreur FB_WriteEraseError actuelle

void SR2FB_ErrorClear()

Cette fonction efface le registre d'état **FB_WriteEraseError** actuel.
Entrée : pas d'entrée Sortie : pas de sortie
Retour : pas de retour

int SR2FB_Read(unsigned long FB_RAddress)

La fonction renvoie le mot lu à partir de l'adresse-octet *FB_RAddress*. Notez qu'aucune vérification n'est effectuée pour s'assurer que la lecture a lieu dans l'espace mémoire occupé par la ROM Flash. Cette fonction peut être utilisée pour retourner le contenu de n'importe quelle mémoire, quel que soit son type.
Entrée : unsigned long FB_RAdress (byte), il s'agit de l'adresse de l'octet lu sur 32 bits. Sortie : pas de sortie
Retourne : Le mot lu à l'adresse de l'octet spécifié

unsigned int SR2DFB_WritePrepare(unsigned long FB_WAddress, unsigned long FB_WSize)

La fonction préérase tous les secteurs du circuit Flash, nécessaires pour écrire un segment *FB_WSize* long, à partir de l'adresse-octet *FB_WAddress*. Elle initialise ensuite le registre **FB_WriteAddress** à la valeur de *FB_WAddress*, de sorte que le prochain appel à **SR2FB_Write** écrira effectivement au début du segment préparé.
Comme l'effacement est effectué secteur par secteur uniquement, cette fonction peut effacer plus de mots 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.
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 de la Flash.
Si, au cours de la préparation, un effacement de secteur est tenté en dehors de la plage d'adresses utilisables, la fonction échoue simplement.
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 dure généralement 0,7 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 long FB_Waddress Il s'agit de l'adresse de départ du segment à préparer.
unsigned long FB_WSize C'est la taille du segment à préparer

Sortie : pas de sortie
Retourne : L'erreur FB_WriteEraseError actuelle

unsigned int SR2FB_Write(int Data)

La fonction place la valeur de *Data* dans la FIFO d'écriture. Elle revient normalement sans attendre la fin de l'écriture. Les écritures de bas niveau sont effectuées sous les interruptions INT1. 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.
Il faut généralement 60µs par mot pour programmer, donc si la fonction est appelée alors que la FIFO est pleine, il se peut qu'elle ne revienne pas avant que 60µs 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 échouera tout simplement. L'échec ne sera pas détecté avant que les données ne effectivement écrites de la FIFO vers la mémoire Flash. La fonction renvoie l'état actuel **FB_WriteEraseError.** Cependant, 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 à placer dans la
FIFO Output : pas de sortie
Retourne : L'erreur FB_WriteEraseError actuelle