

LABVIEW

Edition 2011

R. Decourt

1. Introduction à LabVIEW	15
LabVIEW	16
Les Instruments Virtuels	17
Environnement LabVIEW	18
Les fenêtres de la face avant et du diagramme	18
Barre d'outils de la face avant	19
Barre d'outils du diagramme	20
Menus contextuels	20
Menus	20
Palettes	20
Palette d'outils	20
Palette de commandes	21
Palette de Fonctions	22
Chargement de VIs	23
Enregistrement des VIs	23
Aide et Manuels	26
Aide contextuelle	26
Aide LabVIEW	26
2. Création, Modification, mise au point d'un VI	27
Création d'un VI	28
Face Avant	28
Diagramme	29
Nœuds	29
Terminaux	29
Fils	29
Câblage automatique des données	30
Programmation par flots de données	30
Recherche de contrôles, de VIs et de fonctions	31
Techniques de modification	32
Création d'objets	33
Sélection d'objets	33
Déplacement d'objets	33
Effacement des objets	33
Duplication des objets	33
Nommer les objets	33
Sélection ou suppression de fils	34
Etirement des fils	34
Fils cassés	34
Outils d'édition de la face avant	35
Changer la couleur des objets	35
Modifications	38

3. Créer un sous VI	41
Sous VI	42
Icône et connecteur	43
Création de l'icône	43
Définir le connecteur	43
Sélectionner ou modifier le motif du connecteur	44
Affectation de terminaux aux commandes et aux indicateurs	44
Utilisation des sous VIs	47
Ouverture / Edition de sous VIs	47
Création d'un sous VI à partir d'une sélection.	49
Résumé trucs et astuces	50
4. Boucles et graphes déroulants	51
Boucles de répétition conditionnelle (While)	52
Les graphes déroulants	52
Câblage des graphes déroulants.	53
Registres à décalages	58
Initialisation des registres	59
Nœuds de rétroaction (feedback node)	59
Boucles For	61
5. Tableau, Graphes et clusters	63
Les Tableaux	64
Créer des tableaux sur la face avant.	64
Tableaux à deux dimensions	64
Tableaux de constantes	64
Auto Indexation	65
Boucles et tableaux 2D	65
Fonctions sur tableaux	66
Polymorphisme et tableaux	67
Graphes fonction du temps et graphes XY	69
Les Graphes	69
Les graphes XY	70
Graphe multi courbes	71
Clusters	73
Créer des clusters sur la face avant	73

Constantes de type Cluster	73
Ordonner les Clusters	73
Fonctions sur Cluster	75
Assembler & Assembler par nom	75
Désassemblage des clusters	75
Polymorphisme des clusters	77
6. Structures de choix, séquences et nœuds de calcul	79
Structure de choix	80
Tunnels d'entrée/sortie	80
Exemples	80
Sélection des choix	82
Les Séquences	84
Variable locale de séquence	84
Comment éviter l'utilisation des séquences	85
Boîtes de calcul et nœuds d'expression	86
Nœuds d'expression	86
Boîtes de calcul	86
7. Chaînes et Entrées/Sortie fichier	88
Chaînes	89
Créer des commandes et des indicateurs de type chaîne	89
Tables	89
Fonctions sur chaîne	91
Manipulation de chaînes	91
Chaînes et Nombres	91
Entrées/Sorties sur fichier	93
Fonctions de haut niveau	93
Fonctions de bas niveau	93
Fonctions de base	93
Gestion des erreurs	94
Utilisation de VIs de haut niveau	95
Les VI express	98
8. Programmation multithread	101
Introduction au multithreading sous LabVIEW.	102
Généralités.	102
Avantages	102

Meilleure utilisation du CPU	102
Facilité de programmation	103
Meilleure confiance dans le code	103
Amélioration des performances en multiprocesseurs	103
Déport de threads sur d'autres machines	103
Contraintes	103
Communications inter threads	103
Synchronisation de threads	103
Gestion de ressources communes	103
Corruption mémoire aléatoire	104
Partage de ressources physiques	104
Deadlock	104
Priorités d'exécution	104
Changement de priorité	104
Attente de libération	104
Noeuds synchrone	105
Modèle de programmation	106
Modèle maître/esclaves (Master/Slaves)	106
Modèle parallèle	106
Exemple : Traitement parallèle de données concurrentes	106
Traitement série par lot d'un flot de données (pipeline)	107
Exemple : ACQUISITION-TRAITEMENT-ENREGISTREMENT	107
Gestion des applications multithreads	109
Gestion des communications inter thread	109
Mécanismes de partage de variables	109
Messages	112
Synchronisation de threads	114
Occurrences & notification	114
Rendez vous	115
Gestion de ressources communes	115
Variables fonctionnelles	115
Sémaphores	116
Priorité « subroutine »	117
Priorité et cadencement de threads	117
Introduction	117
Attentes et boucles temporisées	117
9. Chargement dynamique, « VI Server »	119
 Introduction	120
 Chargement dynamique de VI	121
Préambule	121
Référencement d'un VI	121
Chargement d'un VI	121
Lancement et transmission des paramètres	122

10.	Communication entre applications distantes	123
Visualisation et contrôle		124
Via le serveur WEB		124
Activation du serveur WEB		124
Création de la page HTML		124
Prise de contrôle		124
Suivi des connexions		125
Via LabView		125
Mode manuel.		126
Mode programmé		126
Applications partagées		128
Transmission de données		128
Variables partagées		128
Data Sockets		129
Protocoles TCP/UDP		129
Connexion au « VI server »		132
Principe		132
11.	Amélioration de l'interface	135
Importation des titres d'objet.		136
Gestion des menus		137
Edition.		137
Gestion.		137
Menus des objets de face avant		138
Structure d'événement		139
Définition		139
Gestion		139
Evénement provenant d'autres VI		140
Enregistrement d'événements utilisateur		142
Changement du nom d'événement		143
Objets ActiveX et .NET		144
Qu'est ce ?		144
ActiveX		144
.Net		144
Utilisation d'un composant ActiveX et .NET		144
ActiveX		144
Enregistrement d'activité d'un ActiveX ou d'un .NET		145
Généralité sur les « Callback »		145
Enregistrement d'une « Callback »		145
Les CommandeX		148
Généralité		148

Création	148
12. <i>Programmation orientée objet</i>	151
Aperçu	152
La programmation orientée objet	152
Modèles à objets	152
La classe	152
L'encapsulation	152
L'héritage	152
Polymorphisme	153
DEUX MOTS d'UML et de POO	154
Représentation des classes	154
Visibilité des attributs et des méthodes	154
Représentation des dépendances	155
LabVIEW Object-Oriented Programming: the Decisions Behind the Design.	156
Les choix	156
l'encapsulation	156
L'héritage	156
Création d'objets	157
Les bibliothèques	157
Les Propriétés (ATTRIBUTS)	157
Les méthodes	157
Dynamique versus Statique	158
Accesseurs et mutateurs	159
Accès aux ATTRIBUTS	159
Droits d'accès aux méthodes	160
Hiérachisation des classes	161
Changement de l'héritage	161
Explorateur de hierarchie	162
Options de surcharge des méthodes	162
Surcharge obligatoire des méthodes filles	162
Utilisation des méthodes surchargées.	163
Appel des méthodes parentes par les méthodes surchargées	164
Implémentation d'une relation d'agrégation/composition	165
Création Dynamique d'Objets	166
Transfert d'obligation de Surcharge des méthodes à des petits enfants	167
Un peu plus loin	168
Optimisation du code	168
Variant méthodes générique	168
Position du problème	168
Les Variants	169
Référencement des objets	170
Références	170

In Place Element Structure	171
Dans le cas des Objets	171
Aspects cosmétiques	174
Les fils	174
Les icônes	174
13. Acquisition de données	175
Aperçu	176
Configuration matérielle	176
VIs d'acquisition	178
Acquisition de données DAQmx	178
Tâches et Voies virtuelles dans MAX	178
Entrées analogiques type Waveform	182
Donnée type Waveform	182
DAQ Assistant	184
Sorties analogiques	186
Génération de Waveform	186
Les compteurs et entrées/sorties numériques	187
14. Contrôle d'instruments	189
Généralités	190
Configuration et Communication GPIB	191
Architecture du logiciel	191
Configuration du logiciel	191
Communication avec les instruments	194
Caractéristiques propres d'un appareil	194
Etapes essentielles d'une communication PC <=> périphérique	194
Sources d'erreurs classiques	195
Bibliothèques de contrôle d'appareils dans LabVIEW	197
Vi's spécifiques à l'interface GPIB ou RS 232C	197
Visa	197
Utilisation des VISA	198
Assistant d'E/S instruments	200
Driver d'instrument	202
Exemple d'application simple	202

Pilotage d'appareils par port USB	205
Installation et configuration	205
Visa et USB	205
Les drivers I ^V I	205
Que vérifier si le programme minimal ne fonctionne pas ?	208
Que faire si une tâche spécifique ne fonctionne pas ?	208
Communications et configuration série	209
Paramétrages possibles	209
Connexion matérielle	209
VISA et liaison série	210
Nœud de propriété	210
15. <i>Le module Datalogging & supervisory control</i>	213
Introduction	214
Le module DSC pour les nuls	215
Connection à une variable “terminale”	215
Gestion d’alarmes	218
Enregistrement des données	218
Gestionnaire de systèmes distribués	220
Interface Homme Machine	221
Pour aller plus loin	222
Architecture d'un SCADA (Supervisory Control And Data Acquisition)	223
Client / Serveur	223
Librairies et processus	223
Le moteur de variable partagées	224
Types de variables partagées	224
Utilisation des variables partagées	224
Le moteur de publication NI PSP	224
Accès aux variables partagées	224
Accès statique	224
Accès Dynamique	225
Nœuds de Propriétés des variables partagées	226
Alarmes et événements	228
Types d’alarmes	228
Changement des propriétés d’alarmes	228
Récupération et visualisation des alarmes	228
Alarmes Utilisateurs	229
Base de données	231
Enregistrement des données	231
Changement du nom de base	231
Activation de l’enregistrement d’une variable	231

Enregistrement manuel des données	231
Lecture des données enregistrées	232
Serveurs d'E/S	233
Serveurs Standards	233
Serveurs de données	233
Serveur d'impression	233
Serveur d'enregistrement par lots	233
Serveurs personnels	234
Gestion des librairies et des variables	236
Déploiement des bibliothèques	236
Création de processus	237
Création de variables partagées	237
Création de serveurs d'E/S	237
Un petit exemple	238

16. *Le module StateChart* **239**

Introduction	240
Les diagrammes etat transition UML	241
Symbolique UML et Statechart	242
Etat	242
État initial, État final	242
Événement	242
Transitions	243
Entre états	243
Internes	243
JonctionS	243
Etats composé	244
Transitions dans les etats composés	244
Historiques	245
Synchronisation	245
Les StateCharts LabView	246
Création de la bibliothèque	246
Création d'un stateChart	247
Code associé aux états	247
Les Transitions	248
Autour d'un exemple	250
Le Diagramme statechart UML	250
Commande de l'éclairage.	250
Commande de ventilation	250
Deux processus concurrents	251
Mise en marche et arrêt	251
Le diagramme Statechart Labview	251

Création	251
Création des transitions	252
Synchrone ou asynchrone	252
Exécution du statechart	253
Variables de sortie	254
Déboggage	255
Un peu plus loin	257
Transitions simultannées	257
File d'attente de triggers	257
Répétition d'action d'un état	257
17. Labview FPGA	258
Introduction	259
Qu'est-ce un FPGA ?	259
Labview et les FPGA	259
Les cibles supportées	259
Labview RT et FPGA	259
Carte support des exemples	259
Préparation de l'environnement	260
Utilisation des voies numériques	260
Organisation du projet	261
Communication entre les différentes parties.	261
Communication et FPGA et RT	261
Communication entre cible RT et Supervision	261
Premiers programmes	262
Un programme 100% FPGA	262
Création d'un VI FPGA	262
Palette d'outils	262
Programme	263
Compilation	263
Communications entre VI FPGA et VI RT	264
Exécution en parallèle	264
Exemple : Un période- mètre	265
Implantation d'un protocole SPI	266
Outils de cadencement	266
Structure de séquencement	266
Le code	267
Organisation	267
Lancement de la conversion	267
Attente de la fin de conversion	267
Lecture du résultat	267
Code final	268
Accès aux lignes par des nœuds de propriété	268
Gestion des temps de boucles	268

Temps d'exécution d'une boucle	268
Décodeur BCD 7 segments	269
LUT	269
Entrée des données	269
Utilisation	270
Exécution sur une autre cible	270
Multiplexage	270
Vecteurs de taille quelconque	270
Implémentation	271
Le décodeur Binaire/BCD utilisation d'une IP VHDL	272
Nœud d'intégration IP versus CLIP	272
Le code VHDL	272
Nœud d'intégration IP	274
ip	275
Affichage du résultat de la conversion	276
Passage de données entre deux boucles	276
Les variables globales et locales :	276
Les blocs mémoire	277
Création d'un bloc mémoire	277
Accès aux données	277
Exemple d'un buffer circulaire	278
FIFO	279
Création d'une FIFO	279
Accès aux données	279
Exemple avec une FIFO	280
Protocole I2C	281
Lignes bidirectionnelles	281
Envoie réception de données	282
Ecriture d'une trame	282
Optimisation du code FPGA	284
Optimisation générale	284
Limiter la taille et le nombre des objets de la face avant	284
Utiliser les types les plus petits	284
Evitez les fonctions gourmandes	284
Evitez de câbler les erreurs	284
Boucles en un cycle	285
Pipeline	286
Domaines temporels	287
horloges dérivées	287
Création	287
Utilisation	287
Violation temporelles	287
Communication entre domaines temporels	288

Annexes	289
Connections de la carte prototype	289
Bouton poussoir	289
LED	289
BNC	289
Afficheurs	289
ADC	289
DAC	290
Rappel sur l'I2C.	291

1. INTRODUCTION A LABVIEW

LABVIEW

LabView (Laboratory Virtual Instrument Engineering Workbench) est un langage de programmation dédié au contrôle d'instruments et l'analyse de données. Contrairement à la nature séquentielle des langages textuels, LabView est basé sur un environnement de programmation graphique utilisant la notion de flot de données pour ordonner les opérations.

LabView intègre l'acquisition, l'analyse, le traitement et la présentation de données.

Pour l'acquisition de données et le contrôle d'instruments, LabView supporte les standards RS-232/422, USB, IEEE 488 (GPIB) et VXI/PXI, ainsi que les cartes d'acquisition de données.

Pour l'analyse et le traitement des données, la bibliothèque d'analyse étendue contient les fonctions pour la génération et le traitement de signaux, les filtres, les fenêtres, les statistiques, la régression, l'algèbre linéaire et l'arithmétique matricielle.

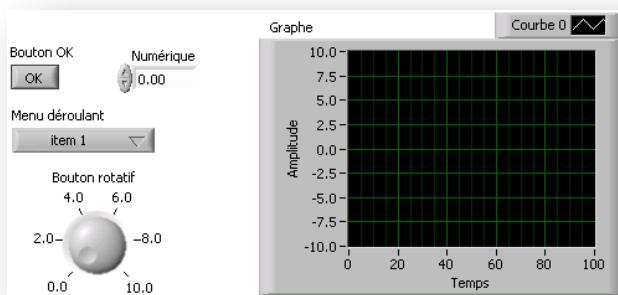
LabView intègre un grand nombre d'éléments de présentation tels les graphes déroulants, des graphes XY, des abaques de Smith, jauge, cadran à aiguille...

LES INSTRUMENTS VIRTUELS

Les programmes LabView s'appellent des *Instruments Virtuels* (VIs). Ces VIs ont trois parties principales : la *Face Avant*, le *Diagramme* et l'*Icône/Connecteur*.

La face avant d'un VI est avant tout une combinaison de commandes et d'indicateurs. Les commandes sont les entrées des VIs, elles fournissent les données au diagramme. Les indicateurs sont les sorties des VIs et affichent les données générées par le diagramme. Vous pouvez utiliser plusieurs types de commandes et d'indicateurs tels que

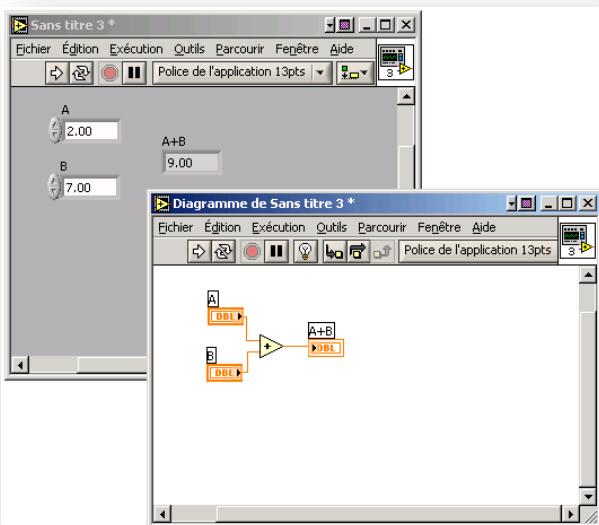
les commandes et les indicateurs numériques, à curseur, booléens, chaîne de caractères, les tables et les graphes (Cf. Figure 1-1).



Vous construisez la face avant en plaçant des éléments graphiques accessibles dans une palette. Vous disposez de boutons, d'indicateurs numériques et de chaînes, de graphes, de LEDs, de listes déroulantes, de menus...

Figure 1-1 la face avant d'un VI

Chaque objet déposé sur la face avant génère dans le diagramme un symbole appelé *Terminal*. Ce terminal contient la *valeur* de l'objet graphique correspondant. Le symbole représente le type de la donnée (par ex. DBL pour double), le sens du flot, (maigre avec une flèche à droite s'il s'agit d'un indicateur, gras avec une flèche à gauche s'il s'agit d'une commande)



Le diagramme contient les terminaux, les sous VIs, les fonctions, les constantes, les structures ainsi que les fils qui relient les différents objets pour leur transmettre les données.

Figure 1-2 La face avant et son diagramme

Après avoir construit la face avant et le diagramme, vous pouvez créer son icône et son connecteur. Si ce VI est utilisé dans un autre VI, il devient un *Sous VI*, il correspond à une routine dans un langage classique, le connecteur représente alors les paramètres entrant et sortant de la routine. L'icône identifiera le VI comme le ferait le nom de la routine. La hiérarchisation des



Figure 1-3 L'icône et son connecteur

applications facilite grandement la réutilisation du code et le débogage.

ENVIRONNEMENT LABVIEW

Au lancement de l'application, la boîte de dialogue suivante apparaît.



Figure 1-4 Fenêtre de démarrage

LES FENETRES DE LA FACE AVANT ET DU DIAGRAMME

Lorsque l'on ouvre un nouveau VI, une fenêtre de face avant et une fenêtre de diagramme apparaissent. L'illustration suivante présente les principaux éléments de ces deux fenêtres.

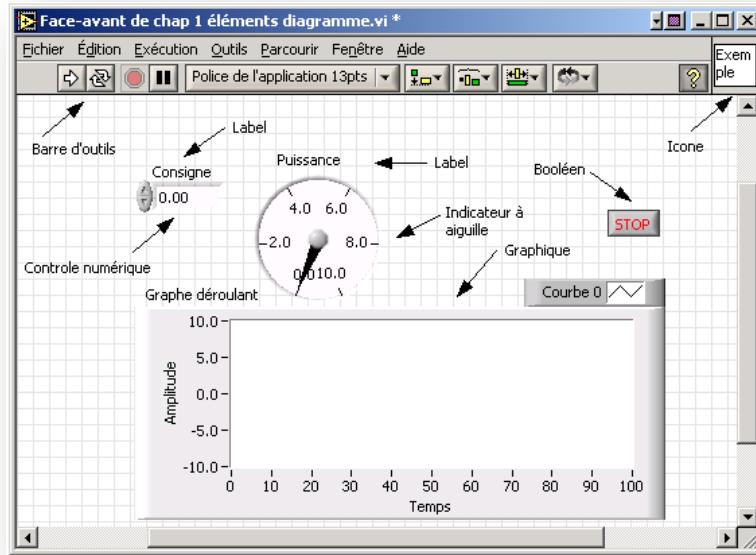


Figure 1-5 La face avant

Elle permet les actions suivantes:

Dans la zone New il est possible d'ouvrir un nouveau VI ou un nouveau projet. L'option More... permet de créer d'autres types d'objets LabVIEW (des contrôles personnalisés, des variables globales, des VIs polymorphes, des VI pré-remplis à partir de modèles...).

La zone Open permet d'ouvrir un VI existant.

La zone de droite donne accès à l'aide et aux exemples ainsi qu'aux différentes ressources d'aide disponibles sur la toile.

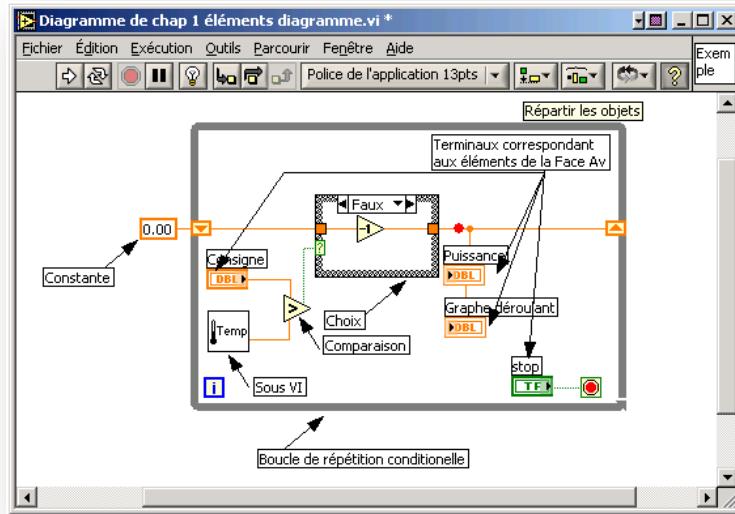


Figure 1-6 Son diagramme

BARRE D'OUTILS DE LA FACE AVANT

La barre d'outils suivante est présente sur la face avant, elle donne accès aux outils d'exécution et de présentation.



Lance l'exécution du VI.



Indique que le VI est en cours d'exécution et qu'il s'agit d'un VI de niveau supérieur (il n'a pas été appelé par un autre VI).



Indique que le VI est en cours d'exécution et qu'il s'agit d'un sous VI (il a été appelé par un autre VI)



Le bouton d'exécution apparaît brisé : le VI n'est pas exécutable (il contient des erreurs).



Relance continuellement le VI après chaque fin d'exécution (équivalent à déposer le VI dans une boucle infinie). Les boutons stop ou pause arrêtent l'exécution.



Arrête l'exécution du VI.



Note Le bouton stop ne doit servir d'arrêt que lors des phases de mise au point, il faut prévoir une structure de programmation capable de finir correctement l'exécution du VI !



Suspend l'exécution du VI, l'icône devient rouge pour indiquer que le VI est en pause, Appuyer de nouveau sur le bouton pour continuer l'exécution.



Permet de choisir la fonte, la graisse, la couleur...d'un champ contenant du texte.



Permet d'aligner des objets, d'en égaliser l'espacement et la taille.



Change le plan d'un objet, pour permettre des superpositions.



Cache ou affiche l'aide contextuelle.

BARRE D'OUTILS DU DIAGRAMME

La barre d'outils suivante est présente sur le diagramme, elle donne accès aux outils de mise au point et de présentation.



Fait apparaître le flot de données sur les fils et sur les connexions des VI.



Entre dans une structure ou dans un sous VI, lors de l'exécution pas à pas. Chaque structure ou sous VI se met à clignoter lorsqu'il est prêt à être exécuté.



Saute l'exécution détaillée d'une structure ou un VI lors de l'exécution pas à pas.



Termine l'exécution détaillée d'un boucle ou d'un sous VI.



Dénote un problème d'exécution potentiel, mais n'empêchant pas le programme d'être exécuté.
Cette option n'est pas naturellement active, on y accède par **Outils»Options»Déboguage**.

MENUS CONTEXTUELS

La plupart des objets présents dans les fenêtres de face avant et de diagramme possèdent des menus contextuels accessibles par un clic droit de souris.

MENUS

Une partie des menus contient des fonctions classiques d'enregistrement, d'édition, de changement de fenêtres... . D'autres sont spécifiques à LabView.



Note : les menus sont parfois inhibés lorsque le VI est en cours d'exécution.

Fichier donne accès à l'ouverture, l'enregistrement ou l'impression de VIs.

Édition permet les copier/coller, la recherche, la création de menus...

Exécution exécute, arrête, accède aux options d'exécution d'un VI.

Outils facilite par le biais d'outils spécifiques, la communication avec des instruments, l'édition de bibliothèques, la comparaison de VIs, la configuration du serveur WEB...

Parcourir facilite la navigation dans un VI et dans sa hiérarchie.

Fenêtre accède aux fenêtres de l'application et aux palettes d'outils.

Aide affiche l'aide, donne accès aux exemples...

PALETTES

LabVIEW possède trois palettes flottantes respectivement nommées: **Outils**, **Commandes**, et **Fonctions**.

PALETTE D'OUTILS

La palette d'outils existe sur le diagramme et sur la face avant. Elle permet de modifier des valeurs, des couleurs, mais aussi de câbler les entrées et les sorties des icônes entre elles, de poser des points d'arrêt, des sondes...

On y accède par Fenêtre»Afficher la palette d'outils.



Note : Il est possible de faire apparaître temporairement la palette d'outils en appuyant sur la touche maj. et en faisant simultanément un clic droit.



Figure 1-7 La palette d'outils



Active la sélection automatique des outils. Dans ce mode, LabVIEW choisit l'outil adapté en fonction de l'emplacement du curseur (pas toujours très pratique). On peut lui préférer le raccourci clavier « touche **espace** », qui commute les deux outils des plus utilisés (élection et doigt dans la face avant, sélection et bobine dans le diagramme) ou la « touche **Tab** » qui propose successivement les quatre outils les plus courants.



L'outil « doigt » change la valeur d'une commande, sélectionne un texte, Le curseur adopte la forme



lorsqu'il est placé dans une zone contenant des caractères.



L'outil « flèche » sélectionne, déplace, redimensionne les objets. Il adopte l'une des formes suivantes



lorsqu'il est sur l'angle d'un objet redimensionnable.



L'outil « édition de texte » permet de changer les étiquettes, d'édition des objets de type caractère et de placer du texte libre dans une fenêtre.



L'outil « bobine » sert au câblage des VI's.



Accède au menu contextuel par un clic à gauche (utilité douteuse !).



Pour se déplacer dans une fenêtre sans les ascenseurs.



L'outil « point d'arrêts » définit l'emplacement sur le diagramme où le programme passera en pause pour permettre le débogage.



L'outil « sonde » visualise la valeur de connections particulières.



L'outil « pipette » mesure la couleur d'un point.



Le pinceau colorie un objet.

PALETTE DE COMMANDES

La palette de commandes est disponible dans la fenêtre de face avant. Elle apparaît fugitivement lors d'un clic droit dans un endroit vierge de la fenêtre, ou par Fenêtre »Afficher la palette de commandes. Elle contient tous

les éléments graphiques disponibles pour créer l'interface utilisateur. Ceux-ci sont hiérarchisés par type de données ou par grandes familles d'objets. Vous apercevez ci-dessous la palette de commandes.



Figure 1-8 La palette de commandes

PALETTE DE FONCTIONS

Elle est accessible dans le diagramme par les mêmes méthodes que celle de commandes. Elle contient l'ensemble des fonctions de LabVIEW regroupées par type de fonctionnalités (Programmation, acquisition, traitement mathématiques, connectivité....).

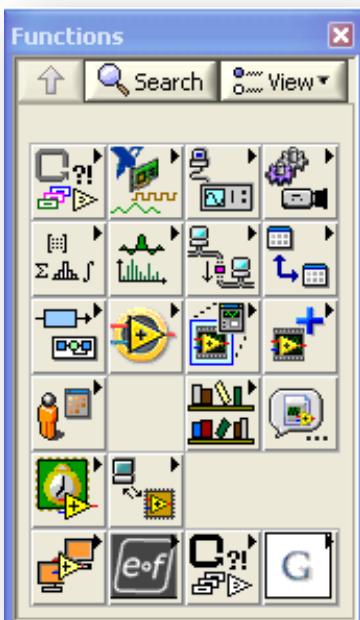


Figure 1-9 La palette de fonctions

CHARGEMENT DE VIS

Lors du chargement la boîte de dialogue suivante apparaît (parfois très furtivement).

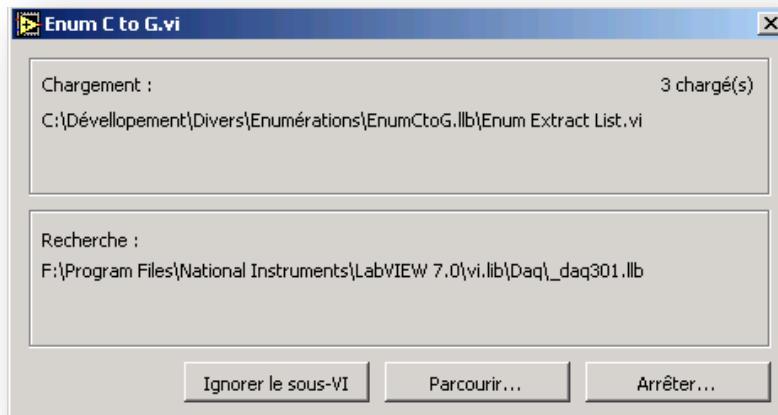


Figure 1-10 Fenêtre de chargement des Vis en mémoire

Le champ **Chargement** du panneau ci-dessus affiche le VI en cours de chargement et comptabilise les VIs en mémoire.

Si LabVIEW ne trouve pas un sous VI, il le cherche dans les chemins spécifiés dans le menu **Outils»Options»Item Chemin** du menu déroulant.

ENREGISTREMENT DES VIS

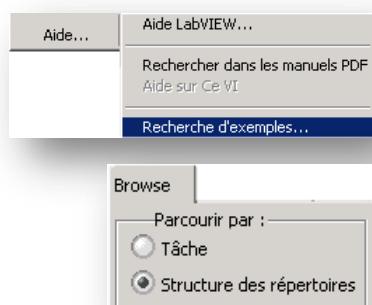
Utilisez les commandes **Enregistrer**, **Enregistrer sous**, **Enregistrer tout** ou **Enregistrer avec options** du menu **Fichier** pour enregistrer vos VIs. Il est possible de créer des librairies d'extension .llb regroupant plusieurs VI. Cette fonctionnalité date de l'époque des noms limités à 8 caractères par MSDOS, National Instruments recommande actuellement d'enregistrer les VIs individuellement dans une architecture répertoire/sous répertoire.

LabVIEW utilise des boîtes de dialogues propres à NI pour les accès fichiers (pour une compatibilité avec d'autres plateformes), cette option peut être désactivée dans le menu **Outils»Options»Item Divers** du menu déroulant.

Exercice 1-1 : Simulation de réponse en fréquence

OBJECTIF: OUVRIR ET EXECUTER UN VI.

Démarrez l'application LabVIEW.



Cliquez sur la flèche de déroulement du bouton **Aide**, situé sur le panneau d'accueil, sélectionnez **Recherche d'exemples**.

Les exemples sont également accessibles depuis le menu aide situé dans la barre de menu de l'application. Une boîte de dialogue affiche l'ensemble des exemples disponibles.

Dans l'onglet **Browse**, sélectionnez **Structure des répertoires**

Figure 1-11 Recherche d'exemples

et chargez **Frequency Response.VI** dans l'arborescence **Apps\Freqresp.llb**.

Note Vous accédez aussi aux exemples par le menu **Fichier»Ouvrir** puis en se déplaçant vers C:\ProgramFiles\NationalInstruments\LabVIEW7.0\examples\apps\freqresp.llb\Frequency Response.vi.

FACE AVANT

Exécutez le VI en cliquant sur le bouton **Run**. Ce VI simule un test de réponse en fréquence d'un filtre, et affiche la courbe de réponse sur l'écran.

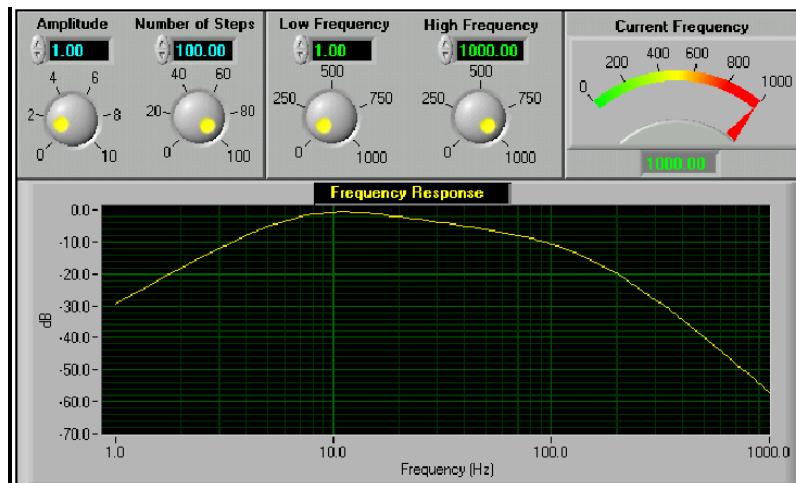


Figure 1-12 Face avant de frequency response

- 👉 Utilisez l'outil Doigt pour changer la valeur des commandes, soit en tournant les boutons, (en cliquant sur la marque d'index et en faisant tourner), soit en cliquant sur les flèches à gauche des commandes, ou en entrant une valeur numérique dans l'afficheur numérique.
- 👉 Lorsque vous entrez une valeur numérique, celle-ci n'est prise en compte qu'après validation par le bouton **Entrer**, ou par la touche **Entr** du pavé *numérique* (pas la touche **Entrée** de la zone alpha qui introduit un retour chariot s'il s'agit d'une zone texte).

Relancez le VI de nouveau pour constater les changements effectués.

DIAGRAMME

Visualisez le diagramme par le biais du menu **Fenêtre» Diagramme** ou par le raccourci <Ctrl-E>

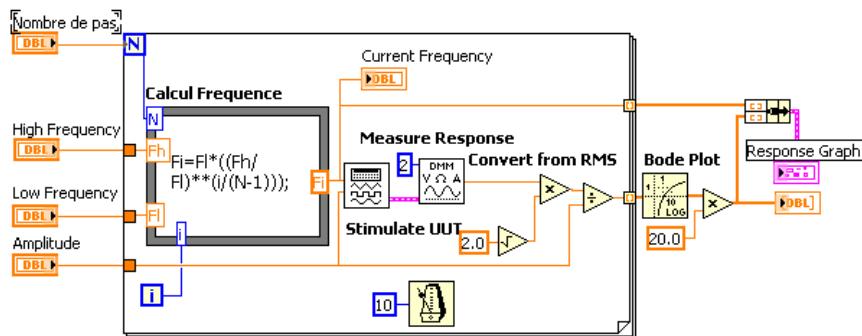


Figure 1-13 Diagramme de frequency response

Vous y voyez la plupart des éléments disponibles sur un diagramme, des fonctions, des sous VIs, des structures, des constantes... Ces notions seront détaillées dans ce cours.



En double-cliquant avec l'outil de sélection sur le sous VI suivant. Vous ouvrez la face avant du sous VI nommé Demo Fluke 8840A

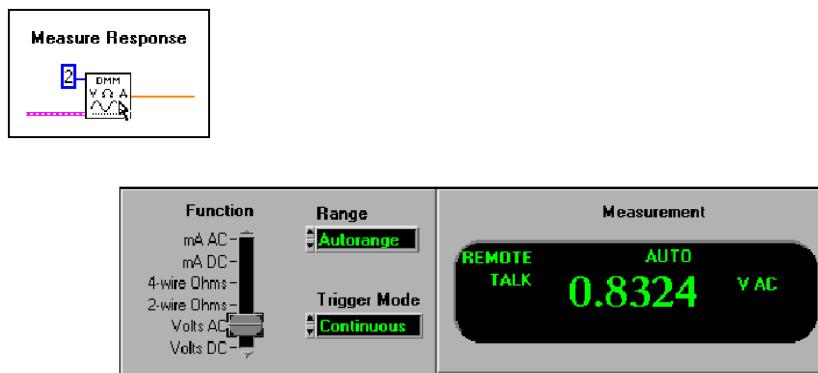


Figure 1-14 Demo Fluke 8840A

La face avant est conçue pour ressembler à l'appareil réel c'est pourquoi les applications LabView sont nommées instruments virtuels. La hiérarchisation des applications en sous VIs permet d'améliorer la réutilisation du code.

Fermez le sous-VI Demo Fluke 8840A, et gardez le VI Frequency Response en mémoire.

Fin de l'exercice 1-1

AIDE ET MANUELS

LabVIEW possède une aide classique, et une aide contextuelle.

AIDE CONTEXTUELLE

-  Elle s'affiche soit par le menu **Aide»Aide contextuelle** soit par le raccourci <Ctrl-H> ou enfin en cliquant sur l'icône. La fenêtre d'aide affiche continûment une information sur l'objet situé sous le curseur. Ce peut être une information sur la nature d'une liaison, le type de données à fournir à un indicateur, le fonctionnement d'un sous VI ou d'une fonction.
Ci-dessous un exemple de contenu d'une aide contextuelle.

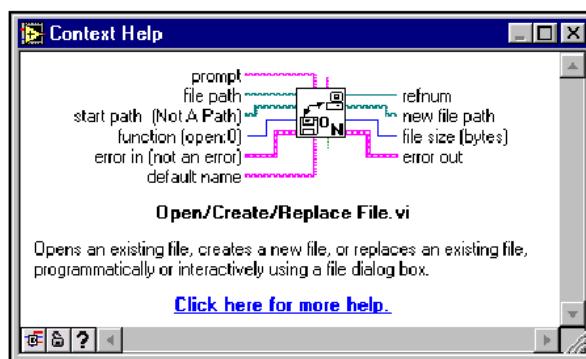


Figure 1-15 Fenêtre d'aide contextuelle



- Ce bouton affiche ou non les connections optionnelles d'un VI.
- Ce bouton fige le contenu de la fenêtre (le déplacement de la souris ne met plus à jour la fenêtre).



- Affiche l'aide classique (plus détaillée), associée à l'item correspondant au moment de l'appui.

AIDE LABVIEW

L'aide de LabView détaille l'ensemble des fonctionnalités de l'application. National Instruments fournit également des tutoriaux et des manuels au format PDF. Le site www.ni.com contient de nombreux exemples, des notes d'applications, des liens...

2. CREATION, MODIFICATION, MISE AU POINT D'UN VI

CREATION D'UN VI

Les VI comportent trois parties – la face avant, le diagramme ainsi que l'icône et son connecteur. Se référer au chap. 3, Crédation d'un sous VI, pour plus d'information concernant l'icône et le connecteur associé.

FACE AVANT

La face avant comporte des indicateurs et des contrôles qui sont les entrées et les sorties du VI.

Les boutons poussoirs ou rotatifs sont par défaut des contrôles.

Les graphiques, voyants, vumètres sont par défaut des indicateurs.

Les contrôles simulent les entrées des instruments virtuels et fournissent les données au diagramme. Les indicateurs simulent la réponse des instruments et affichent les données acquises ou engendrées par les VIs.

Utilisez la palette de commandes pour placer contrôles et indicateurs sur la face avant.

CONTROLES ET INDICATEURS NUMERIQUES

Les deux objets les plus couramment utilisés sont les contrôles et les indicateurs numériques, représentés ci-dessous

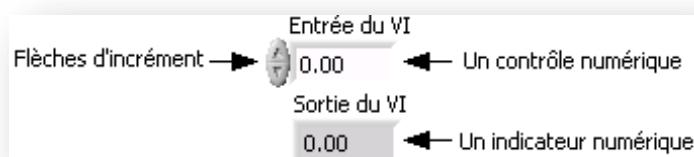


Figure 2-1 Contrôles et indicateurs

Pour saisir ou modifier la valeur d'un contrôle numérique, on peut utiliser les flèches d'incrément ou entrer une valeur avec l'outil texte.

CONTROLES ET INDICATEURS BOOLEENS

A utiliser pour saisir et afficher des valeurs binaires. Les objets booléens les plus communs sont les interrupteurs à bascules et les diodes LED.

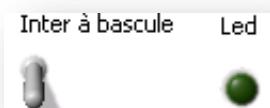


Figure 2-2 Les booléens

CONFIGURATION DES CONTROLES ET DES INDICATEURS

Pour configurer les contrôles et indicateurs, utilisez leur menu contextuel, accessible par un clic droit sur l'objet concerné.

DIAGRAMME

Le diagramme est composé de nœuds, terminaux, connexions comme le montre la figure suivante :

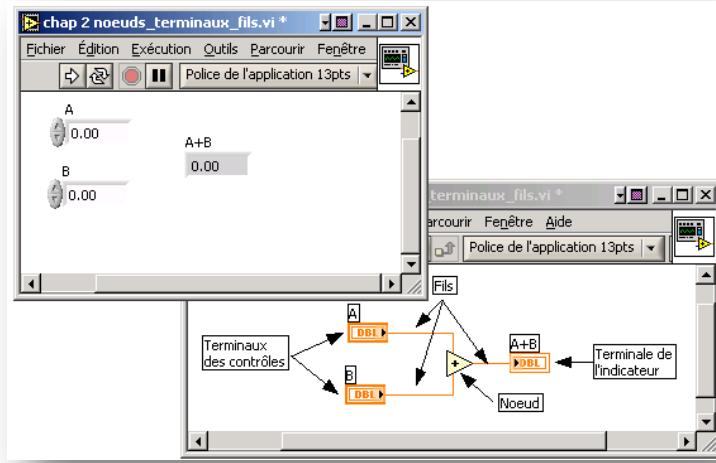


Figure 2-3 Les controles et leurs terminaux

NŒUDS

Les nœuds sont des objets sur le diagramme. Ils possèdent des entrées et/ou des sorties et effectuent des tâches spécifiques lorsqu'un VI fonctionne. Ils sont équivalents à des fonctions dans les langages textuels.

TERMINAUX



Les objets situés dans la face avant apparaissent comme des terminaux dans le diagramme. Les terminaux reflètent le type de données du contrôle ou de l'indicateur. Ainsi le terminal situé à gauche représente un contrôle numérique, défini comme un réel en double précision.
Les terminaux sont des ports de communication entre la face avant et le diagramme. Ils sont équivalents aux paramètres et aux constantes dans les langages textuels. Les données rentrées dans les contrôles de la face avant (A et B Fig. 2-1) entrent dans le diagramme par les terminaux. La fonction addition produit une nouvelle donnée qui arrive sur le terminal de l'indicateur et s'affiche dans la face avant.
Les connecteurs des opérateurs peuvent être visualisés en sélectionnant l'option **Terminaux visibles** du menu contextuel.



FILS

Les fils transfèrent les données dans le diagramme, ils sont ainsi analogues aux variables dans les langages textuels. Chaque fil provient d'une seule source mais peut être réuni à beaucoup de VIs ou de fonctions destinées à les traiter.

Type de la donnée	Scalaire	Tableau 1D	Tableau 2D	Couleur
Numérique	—	—	==	Orange (réel) Bleu (entier)
Booléen	Vert
Chaîne	Rose

Tableau 2-1 types de fils

CABLAGE AUTOMATIQUE DES DONNEES

LabVIEW peut connecter automatiquement les objets lors de leur placement dans le diagramme. Lorsqu'un objet est approché près des autres, LabVIEW visualise provisoirement les connexions possibles. Lors du lâcher, LabVIEW établit les connexions proposées.

Le mode de câblage automatique s'enclenche lorsqu'un objet est déplacé par l'outil de déplacement (flèche), la barre d'espacement étant maintenue enfoncée. Les paramètres de câblage automatique sont définis dans le menu **Outils>>Options**.

PROGRAMMATION PAR FLOTS DE DONNEES

Utilisant le principe du contrôle de flot de données, LabVIEW n'exécute un nœud que lorsque l'ensemble des données arrivant sur ses entrées est présent. Après exécution, le nœud transmet les données sur ses sorties, les passant ainsi au nœud suivant.

Considérons le diagramme suivant :

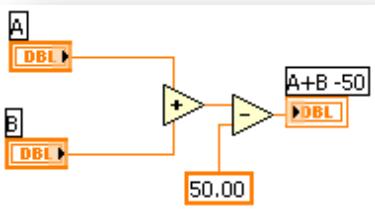


Figure 2-4 le flux de donnée détermine l'ordre d'exécution

Il additionne deux nombres et soustrait 50 au résultat de l'addition. Dans ce cas, le diagramme s'exécute de la gauche vers la droite, non pas parce que les objets sont placés dans cet ordre, mais parce que l'une des entrées de la fonction « Soustraire » reste invalide tant que la fonction « Additionner » n'a pas fini son exécution. Retenir qu'un nœud (fonction) s'exécute seulement quand les données sont disponibles à tous ses terminaux d'entrée et il ne fournit les données à ses terminaux de sortie qu'une fois l'exécution achevée.

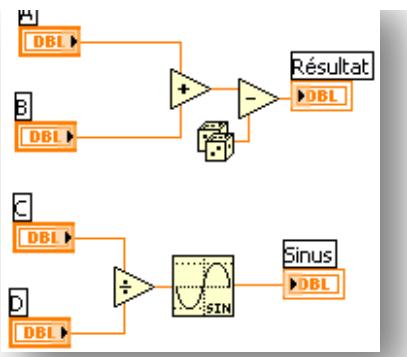


Figure 2-5 exécution parallèle

Dans cet exemple quel segment de code devrait s'exécuter en premier : la fonction « Ajouter », la fonction « Nombre Aléatoire » ou la fonction « Diviser » ?

Impossible à savoir : les entrées des fonctions « Additionner » et « Diviser » sont validées au même moment et la fonction Nombre Aléatoire n'a pas d'entrée. Dans cette situation si un segment de code doit s'exécuter avant un autre et qu'il n'y a pas de dépendance entre les fonctions, il est nécessaire d'utiliser la structure Séquence pour donner un ordre d'exécution. (Cf. chapitre 6).

RECHERCHE DE CONTROLES, DE VIS ET DE FONCTIONS

Les boutons de la palette de fonctions permettent de naviguer et de rechercher les contrôles, les VIs et les fonctions.



Remonter— Remonte d'un cran dans la hiérarchie de la palette.



Options—Ouvre l'option **Parcourir les fonctions** (Browser), à partir de laquelle vous pouvez reconfigurer les palettes.



Rechercher —Change la palette en mode recherche. Celle-ci permet d'effectuer des recherches textuelles pour localiser les contrôles, les VIs, les fonctions dans les palettes.

Exercice 2-1 : Conversion C to F VI

OBJECTIF: CREER UN VI.

Créer un VI qui convertit un nombre, exprimé en degrés Celsius, en degrés Fahrenheit.

Dans les dessins illustrant le câblage, la flèche à la pointe de la souris indique où cliquer. Le nombre à l'extrémité de la flèche précise le nombre de clics à effectuer.

FACE AVANT



Choisir Fichier»Nouveau pour ouvrir une nouvelle face avant. Puis créer une face avant rassemblant à :



2-6 Exercice : la face avant

Sélectionner Fenêtre»Mosaïque

horizontale pour afficher la face avant et le diagramme côté à côté

Créer un contrôle numérique pour saisir la température exprimée en degrés Celsius.

Sélectionner **Commande numérique** dans

la palette **Numérique**. Si la palette « Commandes » n'est pas visible, effectuer un clic à droite dans une zone libre de la face avant pour l'afficher.



Ecrire **deg C** dans l'étiquette puis cliquer en dehors de celle-ci ou valider le raccourci Entrée, situé à gauche dans la barre d'outils ou bien frapper la touche **Entrée** du pavé numérique. sinon LabVIEW



utilise une valeur par défaut qu'il est possible de modifier en utilisant l'outil d'écriture.



Créer un indicateur numérique et l'utiliser pour spécifier la température exprimée en degré Fahrenheit

Sélectionner et placer un indicateur numérique.

Ecrire **deg F** dans la zone d'étiquette.

LabVIEW crée le contrôle et l'indicateur ainsi que les terminaux sur le diagramme. Le terminal est caractéristique du contrôle ou de l'indicateur associé. Ainsi, le terminal ci-contre représente un contrôle de type réel double précision.



Note : Les terminaux des contrôles ont des listels plus épais que ceux des indicateurs.

DIAGRAMME

Afficher le diagramme (Fenêtre»Afficher le diagramme)

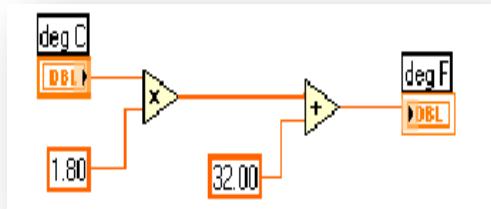


Figure 2-7 diagramme l'exercice

Choisir les fonctions Multiplication et Addition dans la palette fonctions numériques. Si celle-ci n'est pas visible, cliquer dans le diagramme dans une zone libre

Placer deux constantes numériques dans le diagramme. Lors de son placement initial, celle-ci est en surbrillance indiquant qu'une valeur peut y être directement saisie.



Saisir 1.8 dans l'une des constantes et 32.0 dans l'autre.



Utiliser l'outil de câblage pour réaliser le câblage du diagramme.

Pour relier un terminal à l'autre, cliquer à l'aide de la bobine sur le premier terminal, déplacer la jusqu'au second, cliquer dessus comme montré ci-dessous.

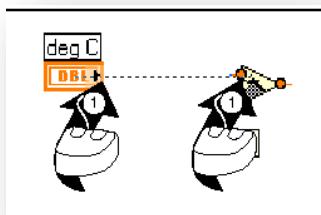


Figure 2-8 Connection des terminaux

Ancrer le fil à l'aide d'un clic de souris puis le courber à angles droits. Un appui sur la barre d'espace commute la direction proposée pour le câblage.

Pour identifier proprement les terminaux des noeuds, effectuer un clic droit sur les fonctions Multiplier puis Additionner. Sélectionner **éléments visibles»Terminaux** pour afficher le connecteur. La même opération affiche à nouveau les icônes.

Quand la bobine passe sur un terminal celui-ci clignote, indiquant qu'un clic effectuerait la connexion. De plus un message précisant le nom du terminal apparaît.

Pour supprimer un fil en cours de création, appuyer sur <Echap>, cliquer à droite ou à gauche sur la source du fil.

Afficher la face avant

Enregistrer le VI car il servira plus tard.

Choisir **Fichier»Enregistrer**, Aller en c:\exercises\LV Basics I.



Note Enregistrer tous les VIs de ce cours dans le répertoire c:\exercises\LV Basics I.

Enregistrer sous le nom Convert C to F.vi.



Entrer un nombre dans le contrôle numérique et lancer le VI.



A l'aide de l'outil contrôle ou de l'outil d'écriture saisir un nouveau nombre. Essayer plusieurs nombres et relancer le VI.

Fin de l'exercice 2-1

TECHNIQUES DE MODIFICATION

Diverses méthodes permettent d'insérer des objets et de modifier les faces avant et les diagrammes

CREATION D'OBJETS

La création d'objets est possible à partir de la palette de contrôle mais également en effectuant un clic à droite sur le terminal d'un nœud puis en sélectionnant l'option **Créer**.

La suppression des contrôles et des indicateurs n'est possible que depuis la face avant. Les terminaux correspondants disparaissent alors automatiquement du diagramme.

SELECTION D'OBJETS

Utiliser la flèche de position pour sélectionner un objet. Une fois sélectionné l'objet est entouré d'un pointillé. Pour sélectionner plusieurs objets, enfoncez la touche <majuscule> puis cliquer sur les objets concernés. On peut également cliquer dans une zone dépourvue d'objets, puis déplacer la souris jusqu'à entourer les objets à sélectionner.

DEPLACEMENT D'OBJETS

Pour déplacer un objet, le sélectionner puis le tirer jusqu'à la position choisie ou le déplacer à l'aide des flèches.

Le déplacement peut être limité à une seule direction (horizontale ou verticale) en appuyant sur la touche < majuscule> pendant la translation.

EFFACEMENT DES OBJETS

Une fois sélectionné l'objet à supprimer, appuyer sur la touche <Suppr>.

Annuler /Rétablir

En cas d'erreur d'édition d'un VI, on peut **Annuler** ou **Rétablir** les étapes précédentes en choisissant l'option correspondante dans le menu **Édition**. Le nombre d'actions pouvant ainsi être annulées ou revalidées est défini dans le menu **Outils»Options»Diagramme**.

DUPLICATION DES OBJETS

Sélectionner l'objet à dupliquer, enfoncez la touche <Ctrl>, puis tirer l'objet ainsi dupliqué vers sa position finale. Copier/Coller permet d'obtenir le même résultat.

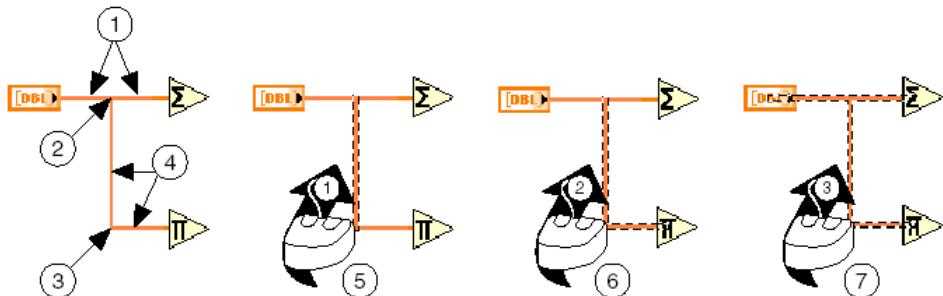
NOMMER LES OBJETS

Utiliser les étiquettes pour identifier les objets sur la face avant et le diagramme. Associées aux objets, celles ci se déplacent avec eux, leur position vis à vis de l'objet est cependant modifiable.

Il est possible d'écrire dans des zones de texte, rattachées à aucun objet, pour insérer des commentaires. Pour créer une zone de texte, utiliser l'outil texte. En fin de message, cliquer dans le bouton <Entrée> de la boîte d'outil ou <Entrée> du pavé numérique ou <Majuscule>+ **Entrée**. La touche <Entrée> ajoute un saut de ligne dans la boîte de texte.

SELECTION OU SUPPRESSION DE FILS

Un segment de fil est un tronçon horizontal ou vertical. Un coude est la jonction de 2 segments. Une branche est la zone de fil entre 2 jonctions ou 2 terminaux. Pour sélectionner un segment cliquer une fois dessus, pour une branche cliquer 2 fois, pour un fil entier 3fois.

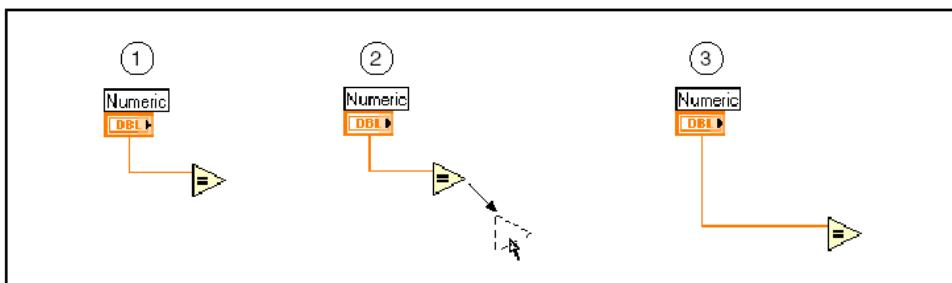


2-9 Sélection des fils

① Segment	④ Branche	⑥ 2 clics sélection d'une branche
② Jonction	⑤ 1 clic sélection d'un segment	⑦ 3 clics sélection de tous les fils
③ Changement de direction		

ETIREMENT DES FILS

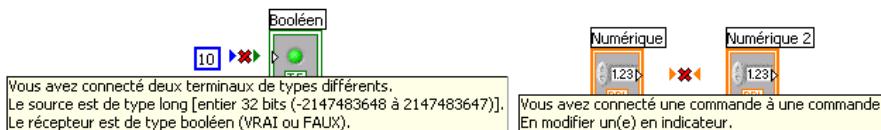
Des objets câblés peuvent se déplacer avec leurs fils. Pour cela utiliser la flèche de positionnement comme indiqué ci-dessous.



2-10 Etirement des fils

FILS CASSES

Un fil cassé apparaît en pointillé. Cela arrive si l'on tente de relier 2 objets véhiculant des données incompatibles.



En plaçant la bobine de fil sur un fil cassé, un message apparaît décrivant l'origine du conflit. Pour supprimer un fil, effectuer un triple clic dessus et frapper la touche <Suppr>. Pour supprimer tous les fils cassés utiliser le raccourci <CTRL>+B

OUTILS D'EDITION DE LA FACE AVANT



Attention : un fil apparaît parfois en pointillé tant que le diagramme n'est pas complètement câblé.

Modification de la police, du style ou de la taille du texte

Pour changer l'aspect d'un texte, utiliser le menu <Police de l'application> situé dans la barre d'outils.

Modification de la taille des objets



La taille de la majorité des objets peut être changée. Après avoir pointé sur un objet avec la flèche de sélection, des poignées apparaissent dans les coins des objets de forme rectangulaire ou autour des objets circulaires. Lors de la modification de taille, la police reste la même. Etirer les objets jusqu'à obtenir la taille recherchée. En appuyant sur la touche <Majuscule> pendant l'étirement, les proportions de l'objet sont conservées.



Alignement, distribution et dimensionnement des objets
Pour aligner des objets sur un axe : sélectionner les puis utiliser le menu <Aligner> les objets dans la barre d'outils. Pour les distribuer de façon homogène, utiliser le menu <Répartir>. Enfin, pour homogénéiser la taille d'objets de la face avant, utiliser le menu <Redimensionner>.

Ordonner, Regrouper, Bloquer des objets



Si des objets se chevauchent, il est possible d'imposer qui est devant qui est derrière. Sélectionner un objet et imposer sa position en utilisant <Réorganiser> de la barre d'outils.

Il est possible de grouper et/ou verrouiller des objets depuis le menu <Réorganiser>. Les objets groupés gardent leurs dispositions relatives. Les objets verrouillés conservent leur emplacement sur la face avant et ne peuvent pas être supprimés.

CHANGER LA COULEUR DES OBJETS

On peut modifier la couleur de beaucoup d'objets, sauf ceux dont la couleur est spécifique : fils, représentation des données.

Pour les autres, choisir l'outil couleur et effectuer un clic droit sur l'objet (face avant ou diagramme).

La couleur par défaut peut être modifiée en choisissant les menus **Outils>>Option>> Couleurs sur la barre de menus.**

De la même façon, il est possible de rendre les objets transparents.

Exercice 2-2 : Edition d'un VI

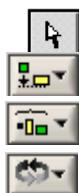
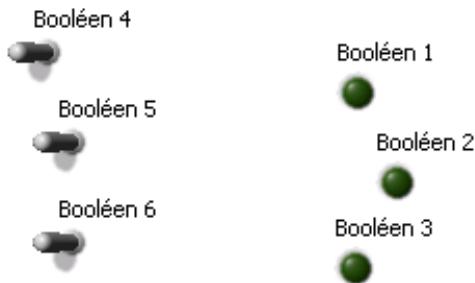
OBJECTIF: UTILISER LES OUTILS D'EDITION.



On peut toujours revenir en arrière par le menu **Annuler/Rétablir**.

FACE AVANT

Déposez sur la face avant trois interrupteurs à bascules et trois LED sans vous soucier de leurs labels ni de leurs positions. Vous pouvez utiliser le clonage avec la touche <CTRL>.



- Utilisez l'outil de sélection pour sélectionner les trois interrupteurs.
- Alignez les interrupteurs avec l'option **Bords gauches** de l'outil d'alignement.
- Réorganisez la disposition avec l'option **Compression verticale** de l'outil de répartition.
- Groupez les trois objets ensemble, notez que vous ne pouvez plus déplacer individuellement les interrupteurs.
- Effectuez les mêmes opérations sur le groupe de LEDs.
- Changez les couleurs ON et OFF des LEDs en utilisant l'outil doigt pour changer son état, et l'outil pinceau pour modifier sa couleur.
- Chargez les noms de labels, les fontes, la couleur du texte.....

DIAGRAMME

Câblez chaque indicateur à chaque contrôle, écrire des commentaires en texte libre, essayez les fonctionnalités de câblage automatique présentées plus haut.

Fin de l'exercice 2-2

Techniques de mise au point

- Si le bouton <**Lancer un VI**> semble cassé, comme montré ci contre, le VI n'est pas exécutable. Si en fin de câblage la flèche est toujours cassée le VI ne fonctionnera pas
- 
- Recherche des erreurs
- Cliquer sur le bouton **Exécuter** ou choisir le menu **Fenêtre»liste des erreurs**. Un double clic sur la description de l'erreur met en surbrillance l'objet qui contient l'erreur.
- Visualisation du flux de données
- Une visualisation du flux de données est obtenue en cliquant sur le menu **Animer l'exécution**. Cette option utilisée conjointement avec le mode pas à pas permet de visualiser les données transmises d'un nœud à l'autre.
- 



Note : Cette visualisation réduit fortement la vitesse d'exécution VI.

Pas à pas



Le mode pas à pas visualise intégralement le déroulement d'un VI ou sous VI. Ce mode affecte exclusivement un seul VI à la fois. Pour rentrer dans ce mode cliquer sur le bouton **Commencer l'exécution en mode pas à pas (icône ci-contre)**. Déplacer la souris sur **Commencer l'exécution**. Un message annonçant la tâche à venir apparaît. On peut exécuter les sous VIs en pas à pas ou normalement.



Si le mode pas à pas est activé ainsi que la visualisation du flot, une flèche verte, en surimpression, apparaît sur les icônes des VIs en train de s'exécuter. (cf. Chap III)

Sondes



Utiliser l'outil **Sonde** pour visualiser immédiatement la valeur qui chemine dans une connexion quand le VI fonctionne. En mode pas à pas ou après un point d'arrêt, il est possible de visualiser la valeur qui vient de transiter dans un fil. Les sondes disparaissent lorsque le VI est fermé.

Points d'arrêt



Utiliser l'outil point d'arrêt, présenté ci contre, pour placer un point d'arrêt sur un VI, un nœud ou un fil du diagramme et arrêter l'exécution du diagramme en ce point.

Placé sur un fil, l'exécution stoppe après que la donnée y ait transité. Placé sur un diagramme, le VI stoppe son exécution une fois tous les nœuds exécutés. Quand un VI est mis en pause, LabVIEW l'entoure d'un listel rouge.

Positionner l'outil **Point d'arrêt** sur le point d'arrêt ou sur la structure concernée, pour l'enlever.

RESUME ET ASTUCES

RESUME

Les contrôles et les indicateurs sont les entrées et sorties interactives du VI.

La bordure d'un contrôle est plus épaisse que celle d'un indicateur.

Pour transformer un contrôle en indicateur, choisir dans le menu contextuel **Changer en indicateur**.

Utiliser l'outil doigt pour configurer les contrôles et les indicateurs

Utiliser l'outil position (flèche) pour choisir, déplacer, modifier la taille des objets.

Utiliser l'outil loupe pour effectuer des recherches dans les palettes de contrôles et de fonctions.

Lorsque la flèche <Lancer> est cassée le VI ne s'exécute pas. Cliquer sur la flèche cassée, affiche les erreurs

Utiliser la lampe d'animation, le pas à pas, les sondes et les points d'arrêt pour la mise au point.

CONSEILS ET ASTUCES

La plus part des raccourcis combinent la touche CTRL avec une autre touche :

RACCOURCIS

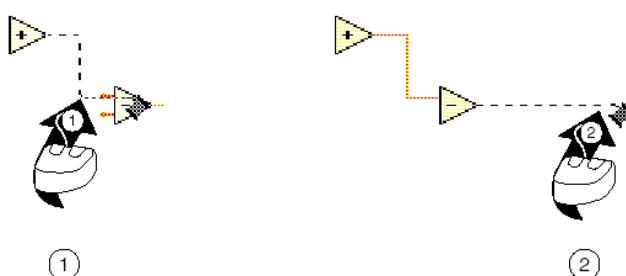
<Ctrl-S> Sauve un VI.
 <Ctrl-R> Lance un VI.
 <Ctrl-E> Commute de la face avant au diagramme.
 <Ctrl-H> Affiche l'aide contextuelle.
 <Ctrl-B> Supprime les mauvais fils (Bad wires).
 <Ctrl-F> Trouve les VIs, les textes, les objets chargés en mémoire ou dans une liste de VIs.
 Pour balayer les objets de la palette d'outils, presser la touche TAB.
 Pour passer de la flèche à la bobine, appuyer sur la barre d'espace.
 Pour incrémenter ou décrémenter rapidement des contrôles, appuyer simultanément sur la touche <majuscules> et sur les flèches du clavier numérique.

MODIFICATIONS

Tirer et lâcher les contrôles et les indicateurs de la face avant vers le diagramme pour créer des constantes et vice-versa.
 Cloner un objet en appuyant sur la touche <Ctrl> et en le tirant jusqu'à sa position finale.
 Limiter le déplacement d'un objet en appuyant sur la touche <Maj.> lors de son déplacement.
 Maintenir la proportionnalité d'un objet lors de sa réduction en appuyant sur la touche <Maj.>.
 Pour retoucher symétriquement la taille d'un objet, appuyer sur la touche <Ctrl> en l'étirant.
 Pour remplacer plutôt qu'effacer, utiliser le menu <Remplacer> depuis le menu contextuel.
 Pour terminer la frappe d'une étiquette, taper < Maj.-Entrée> ou <Entrée> du pavé numérique.
 Pour cloner la couleur d'un objet utiliser l'outil **copier la couleur** (pipette de la palette outils) et repeindre la cible avec le pinceau.
 Pour revenir en arrière, choisir les menus Annuler/Rétablissement.
 Pour créer de l'espace libre dans une zone du diagramme : enfoncez <CTRL> en même temps que l'outil flèche dessine un rectangle sur le diagramme.

CABLAGE

Pour savoir quels terminaux câbler, afficher l'aide contextuelle , en frappant <CTRL>+H. Les connexions obligatoires sont en gras, celles qui sont recommandées en normal, les autres en gris.
 Pou sélectionner tout ou partie d'un fil cliquer 1, 2ou 3 fois.
 Pour couder un fil, l'ancre par un clic de souris avant de changer de direction.



①Ancrer un fil en cliquant	②Ancrer et couper le fil par un double clic
----------------------------	---

Pour matérialiser la jonction des fils par des points, aller dans le menu **Outils>>Option>>Diagramme >>Points aux jonctions.**

Pour bouger un objet d'un pixel utiliser les flèches du pavé numérique. Pour le déplacer de plusieurs appuyer simultanément sur <Majuscules>.

Pour supprimer un fil en cours de création appuyer sur <Echap>.

3. CREER UN SOUS VI

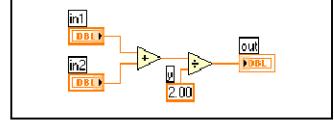
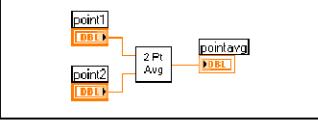
SOUS VI

Après avoir crée un VI, son icône et son connecteur, il est possible de l'utiliser dans un autre VI.

Un VI inclus dans un autre VI est appelé sous VI. Il correspond à une fonction dans un langage textuel.

Le nœud d'appel du sous VI correspond à l'appel de la fonction. L'usage de sous VI augmente la lisibilité, la réutilisation de code et le débogage.

La comparaison entre le code ci dessous et le diagramme montre l'analogie entre les approches.

Function Code	Calling Program Code
<pre>function average (in1, in2, out) { out = (in1 + in2) / 2.0; }</pre>	<pre>main { average (point1, point2, pointavg) }</pre>
SubVI Block Diagram	Calling VI Block Diagram
	

3-1 Sous VI = fonction

ICONE ET CONNECTEUR

Après avoir créé une face avant et son diagramme, il faut personnaliser l'icône associée et son connecteur.

CREATION DE L'ICONE

Chaque VI possède une icône dans le coin supérieur droit de la face avant et du diagramme.

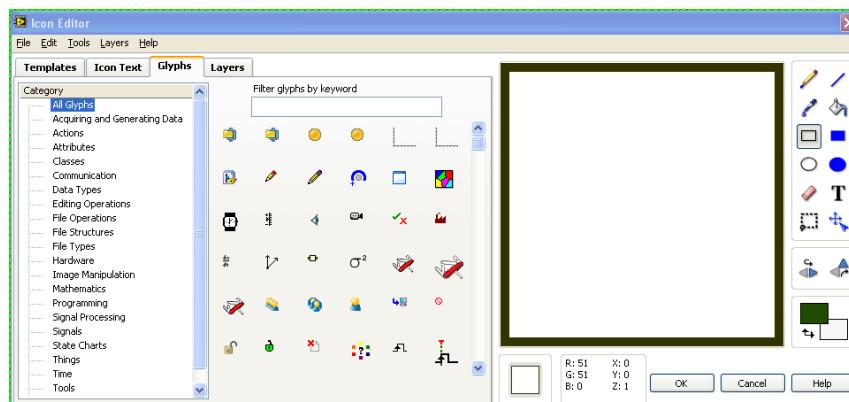
L'icône est la représentation graphique du VI. Si un VI est utilisé en tant que sous VI, il est identifié par son icône.



Éditer l'icône...

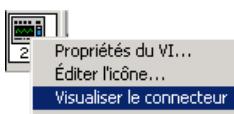
L'icône par défaut est représentée à gauche. Utilisez l'éditeur d'icône pour personnaliser l'icône en sélectionnant **Editer l'icône** dans le menu contextuel de l'icône par défaut.

Utilisez les outils situés sur la droite pour concevoir l'icône dans la zone d'édition. L'éditeur d'icônes permet de concevoir des icônes à partir de motif prédefinis, d'ajouter du texte, des images d'une bibliothèque, de concevoir l'icône à partir de plusieurs couches avec des transparencies... bref normalement de faire quelque chose de beau !



3-2 l'éditeur d'icônes

DEFINIR LE CONNECTEUR



Pour utiliser un sous VI il faut définir un connecteur. Le connecteur est un ensemble de terminaux qui correspondent à tout ou partie des contrôles et indicateurs du VI à la façon des paramètres passés dans l'appel d'une fonction en langage textuel. Le connecteur définit ainsi les entrées/sorties du VI.

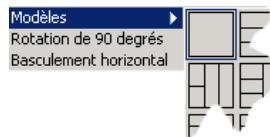


Pour visualiser les bornes du connecteur, effectuer un clic droit sur l'icône dans la face avant, choisir l'option **Visualiser le connecteur**. Chaque rectangle sur le connecteur représente une connexion. Le nombre de connexions par défaut est égal au nombre de contrôles et d'indicateurs de la face avant. Dans l'exemple ci-dessous, on distingue trois contrôles et un indicateur



3-3 Connecteur = liste des paramètres

SELECTIONNER OU MODIFIER LE MOTIF DU CONNECTEUR



Pour sélectionner un autre connecteur, appeler le menu contextuel du connecteur et choisir

Modèles. Il est souvent préférable de choisir une configuration avec un nombre de connexions plus important que nécessaire. Les connexions supplémentaires peuvent rester non câblées, cela permet d'effectuer des modifications sans bouleverser la hiérarchie du VI, notamment l'édition de lien.

Le nombre maximum de terminaux pour un sous VI est fixé à 28.



Note Eviter de définir plus de 16 connexions sur le même VI. Trop de terminaux réduisent sa lisibilité du VI et sa facilité de mise en œuvre. Il est toujours possible de faire un cluster de plusieurs fils (mis dans une gaine). Dans ce cas on passe alors la gaine et non les fils individuellement. (Cf. Chapitre 5)

Pour modifier l'agencement des terminaux du connecteur, choisir l'une des commandes :

Basculement horizontal, Basculement vertical ou Rotation de 90 degrés.

AFFECTATION DE TERMINAUX AUX COMMANDES ET AUX INDICATEURS

Après avoir choisi un modèle de connecteur, il faut le câbler. Placez de préférence les entrées à gauche et sorties à droite.

Pour faire correspondre un terminal à un contrôle (ou un indicateur) cliquer sur le terminal avec la bobine puis sur l'objet graphique correspondant. Le terminal prend la couleur du type d'objet auquel il correspond.

L'ordre dans lequel on clique sur les objets n'a pas d'importance.



Note Bien que les liens soient établis en utilisant l'outil bobine, aucun trait ne relie le connecteur et les objets avec lesquels il communique.

Exercice 3-1 : Conversion °C en degré F

OBJECTIF: CREER UNE ICONE ET UN CONNECTEUR POUR UTILISER CE VI EN TANT QUE SOUS VI

Suivre les étapes suivantes pour créer une icône et un connecteur pour le VI déjà existant qui transforme des °C en Fahrenheit

Afficher la Face avant

Ouvrir Convert C to F.vi

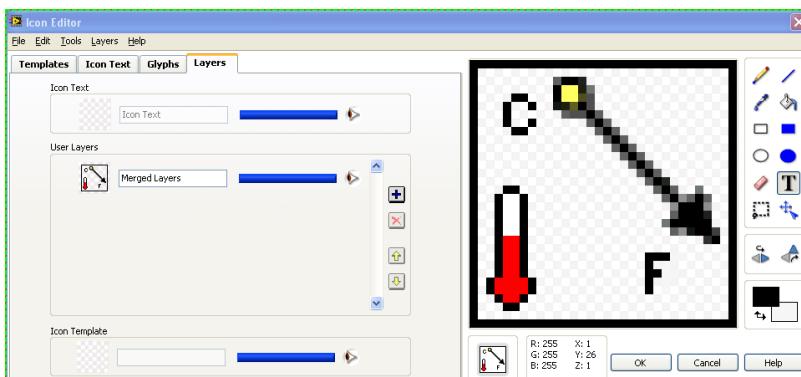


Truc L'appel du menu **Fichier>Fichiers récemment Ouverts** permet d'accéder aux derniers fichiers utilisés.

La face avant ci dessous apparaît.



Dessiner l'icône



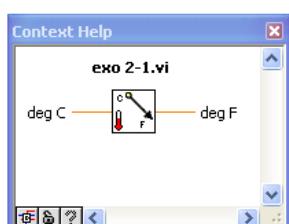
Créer les connexions



Cliquez à droite sur l'icône et choisir Visualiser le connecteur pour définir les terminaux. LabVIEW choisit une configuration basée sur le nombre de contrôles et d'indicateurs de la face avant.

Celle-ci comporte 2 terminaux degC et degF, LabVIEW propose une configuration avec 2 connexions.

Assignez deux terminaux, l'un au contrôle, l'autre à l'indicateur.



Cliquez sur le terminal de gauche, l'outil bobine apparaît automatiquement. La connexion se colore en noir.

Cliquez sur **deg C**, le terminal devient orange et un pointillé entoure la connexion affectée.

Cliquez dans une zone libre de la face avant. Le pointillé disparaît et la connexion se pare de la couleur de l'objet.

Opérez de même avec l'indicateur **deg F**.

Déplacez le curseur sur le connecteur. La fenêtre d'aide montre 2 terminaux orange (réels) et leurs labels.

Sauvez le VI.

Fin de l'exercice 3-1

UTILISATION DES SOUS VIS



Une fois un VI créé, celui-ci est utilisable en tant que sous VI. Pour le placer dans le diagramme d'un autre VI, utilisez **Sélection d'un VI**, accessible depuis la palette de fonctions, l'icône. Si le VI est déjà ouvert vous pouvez, à l'aide de l'outil de sélection, cliquer dans son icône et la déposer dans le diagramme du VI appelant.

OUVERTURE / EDITION DE SOUS VIS

Pour ouvrir la face avant d'un sous VI depuis un VI appelant, double cliquez avec l'outil sélection sur l'icône. Les changements que vous opérez n'affecteront que cette instance tant que le sous VI n'est pas enregistré. Une fois la modification enregistrée, elle affectera **toutes les instances dans tous les programmes** où ce VI est déjà utilisé !

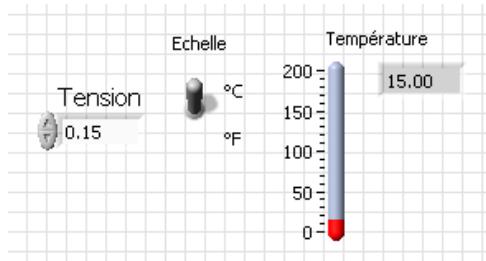
Exercice 3-2 : Thermomètre VI

OBJECTIF: CREER UN VI UTILISANT UN SOUS VI.

Vous disposez d'un capteur fournissant une tension proportionnelle à la température tel que $T(^{\circ}\text{C}) = 100^{\circ}\text{V}$. On se propose de réaliser un thermomètre utilisant ce capteur et capable d'afficher la température en $^{\circ}\text{C}$ ou $^{\circ}\text{F}$.

FACE AVANT

Dans un nouveau VI créez la face avant :



Le thermomètre est dans la palette **Numérique**.

Définir les propriétés de l'objet dans le menu contextuel **Eléments visibles»Afficheur numérique**.

L'interrupteur à bascule est vrai en haut, on peut s'en assurer par le menu contextuel **Opération sur les données»Changer la valeur en...**

Il est possible de documenter le VI (cette description apparaîtra dans l'aide contextuelle) en sélectionnant dans le menu contextuel de l'icône Propriété du VI puis Documentation dans la liste déroulante. Entrez une brève description du VI dans le cadre.

DIAGRAMME

Vous utilisez les éléments suivants :



Numérique»Nombre aléatoire pour simuler une tension mesurée

Numérique»Multiplier pour le coefficient de proportionnalité de 100 . 0 . La constante peut

être placée directement en déroulant le menu contextuel de l'entrée y en en sélectionnant

 **Créer»Constante.**



Sélection d'un VI, pour placer votre VI Convert C to F.vi.



Comparaison»Sélectionner permet d'aiguiller l'une ou l'autre de ses deux entrées vers la sortie en fonction d'un booléen.



Reliez un connecteur de sortie à l'indicateur Température et un connecteur d'entrée au contrôle

Echelle. Enregistrez le VI sous le nom Thermometer.vi .Affichez la face avant et exécutez le code.

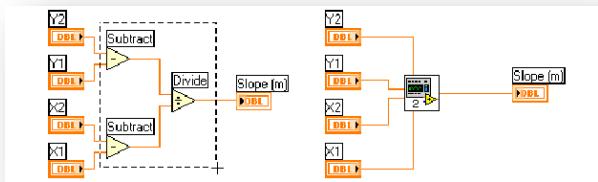
Vous avez la possibilité d'exécuter en boucle le code sans structure particulière dans la phase de test en utilisant le bouton Exécuter en continu au lieu du bouton Exécuter.

Fin de l'exercice 3-2

CREATION D'UN SOUS VI A PARTIR D'UNE SELECTION.

Il est possible de créer un sous VI en sélectionnant une partie de diagramme puis en utilisant

Edition»Créer un sous VI. LabVIEW crée automatiquement un sous VI contenant les contrôles et indicateurs nécessaires. D'un point de vue conceptuel il ne semble pas que cette fonctionnalité fournisse un code bien réfléchi !



3-4 Création d'un sous VI à partir d'une sélection

RESUME TRUCS ET ASTUCES

Un VI appelé d'un autre VI est un sous VI, l'utilisation de sous VI rend le code plus fonctionnel et facile à déboguer.

Un sous VI est identifié par son icône, les paramètres sont passés par son connecteur.

Il est possible d'ajouter une description succincte des fonctionnalités de ses VI en sélectionnant propriétés dans le menu contextuel de l'icône.

4. BOUCLES ET GRAPHES DEROULANTS

BOUCLES DE REPETITION CONDITIONNELLE (WHILE)



Identique à une structure de type **Faire...Répéter tant que**, la boucle While exécute le code contenu dans son cadre tant qu'une condition est Vraie ou Fausse. Cette boucle est située dans la palette **Fonctions»Structures**.

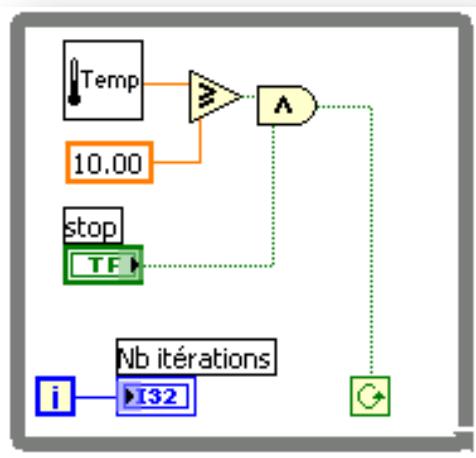
La souris est utilisée pour définir la zone de code à répéter. Lors du relâchement du bouton de la souris, la zone sélectionnée, enfermée à l'intérieur de la structure sera répétée. Il est possible d'enrichir le code contenu dans la boucle en y déposant d'autre VIs, fonctions ou structures.



La condition d'arrêt, présentée ci-contre, permet de contrôler l'exécution de la boucle. La condition par défaut est **Continuer si vrai**. D'un clic droit il est possible d'inverser cette condition. L'icône prend alors l'allure présentée à gauche



Le nombre d'itérations effectuées est contenu dans l'icône présentée à gauche. Cette variable est initialisée à 0 lors de la première itération.



4-1 la boucle While

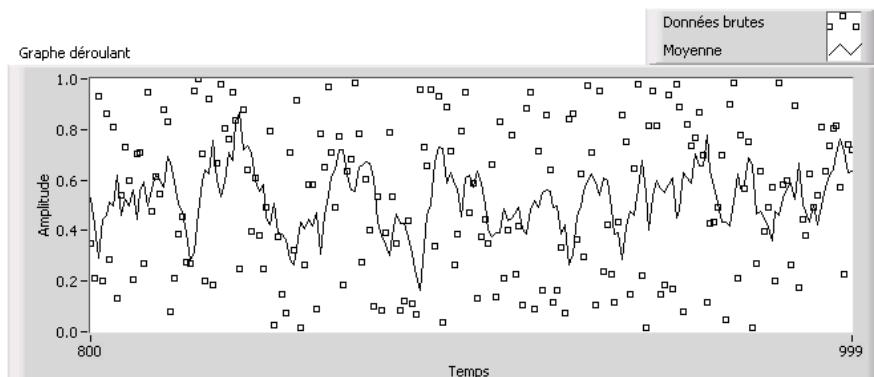
L'exemple suivant présente une boucle dont l'exécution se poursuit tant que la sortie du sous-VI est inférieure à 10, ou que le bouton **stop** est faux (continuer tant que vrai \Leftrightarrow arrêter dès que faux).

LES

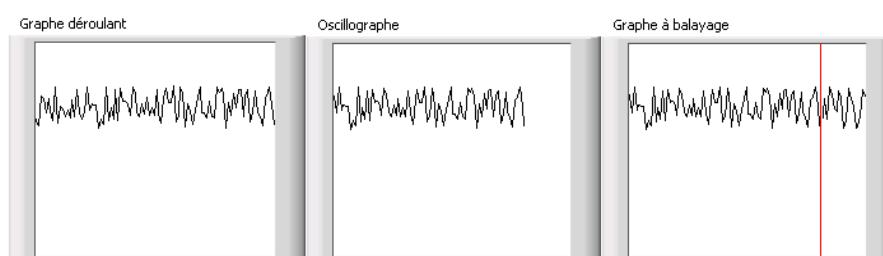
GRAPHES DEROULANTS

Les graphes déroulants sont des indicateurs numériques destinés à afficher l'évolution d'une (ou plusieurs) variable(s) sur une (ou plusieurs) courbe(s). L'axe des X a donc pour échelle un numéro d'ordre d'arrivée du point. Les graphes déroulants sont situés dans la palette **Graphes**. La figure

ci-dessous présente un **graphe multi courbes**.



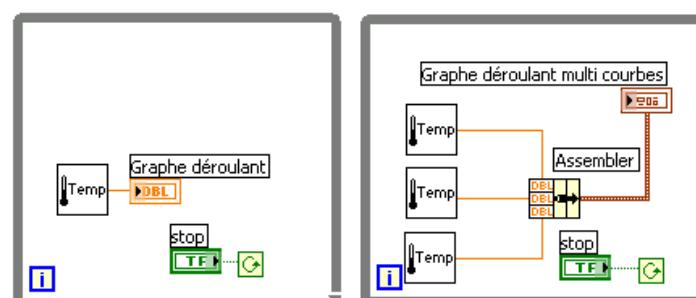
Ce type de graphe dispose de trois modes de rafraîchissement accessibles depuis le menu contextuel **Avancé»Mode de rafraîchissement**.



Le graphe déroulant est une fenêtre dans laquelle l'ensemble des données est décalé d'un point vers la gauche à chaque nouvelle arrivée. **L'oscillographe** efface complètement la fenêtre à chaque nouveau remplissage du buffer. Enfin le **graphe à balayage** efface les anciennes données à chaque nouvelle arrivée, une barre sépare les anciennes des nouvelles, façon écran radar.

CABLAGE DES GRAPHES DÉROULANTS.

L'entrée d'un graphe déroulant est un scalaire. Le terminal d'entrée du graphe s'adapte automatiquement au type de données.



L'affichage de courbes multiples s'obtient en assemblant plusieurs courbes uniques à l'aide de la fonction **<Assembler>** située dans la palette **Fonctions»Cluster**.

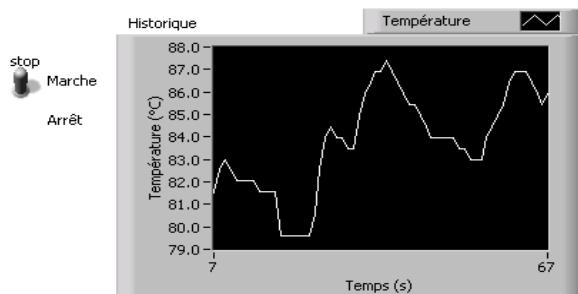
Exercice 4-1 : Surveillance de température

OBJECTIF: UTILISER UNE BOUCLE WHILE ET UN GRAPHE DÉROULANT POUR FAIRE UNE ACQUISITION ET AFFICHER DES DONNÉES.

Réalisez les étapes suivantes pour simuler un suivi et une acquisition de température.

FACE AVANT

Créez une nouvelle face avant ci-dessous.



Prenez un interrupteur à bascule verticale, il servira à arrêter le programme. Il se situe dans la palette **Commandes »Booléen**. Positionnez le vers haut à l'aide de l'outil doigt, d'un clic droit. Impossez cette position comme position par défaut, depuis le menu contextuel : **Opération sur les données»Désigner les valeurs actuelles comme défaut**

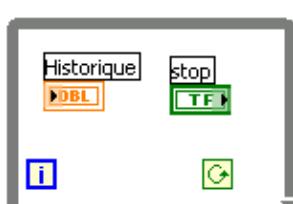
- A l'aide de l'outil Texte, entrez Stop comme label du bouton puis les commentaires Marche et Arrêt. La validation se fait en cliquant sur l'icône Entrée ou par la touche *Entr.* du pavé numérique
- Placez un graphe déroulant, modifiez les labels d'axe, de légende et de graphe en conséquence.

Le VI est sensé mesurer la température de la pièce, changez (directement sur l'axe) du graphe les valeurs minimum et maximum de l'axe Y en 0 et 50

DIAGRAMME

Passez à la fenêtre diagramme (CTRL-E).

Rajoutez une boucle While autour des deux terminaux présents (Eventuellement les rapprocher avant). Pour cela :



- Sélectionnez la boucle While dans **Fonctions»Structures**.
- Cliquez et agrandissez le rectangle de sélection autour des deux terminaux.
- Utilisez l'outil de positionnement pour retoucher la boucle si nécessaire.
- Positionnez le VI Thermometer.vi que vous avez créé.

Connectez la sortie du VI au graphe déroulant. Pour obtenir une température en °C, il faut connecter une constante booléenne vraie à l'entrée du VI, accessible par un clic droit sur son entrée. Choisissez l'option **Créer» Constante**.

Enregistrez ce VI sous le nom Surveillance de température.vi, vous le réutiliserez plus tard.

Lancez l'exécution en cliquant sur la flèche de la barre d'outils ou en tapant <ctrl> r.

Le programme exécute la section de code à l'intérieur de la boucle tant que la condition d'arrêt est VRAIE.

Basculez l'interrupteur en position Arrêt pour mettre fin au VI.

Vous pouvez modifier l'aspect des graduations, la mise à l'échelle, le format des points, la grille... par un clic droit sur le graphe, à vous de jouer...

Action mécanique des interrupteurs

Lorsque vous appuyez sur l'interrupteur pour mettre fin au VI, l'interrupteur demeure en position Arrêt. Il est possible de changer le comportement mécanique en sélectionnant **Action mécanique** dans les propriétés de l'interrupteur.

Actions non bufférisées (si le contrôle n'est pas lu par le programme au moment où l'action de l'utilisateur se produit (mode pas à pas ou tâche longue en cours), cette dernière est ignorée)



Commutation à l'appui—L'interrupteur bascule entre deux positions stables à chaque appui du bouton de souris (interrupteur bistable à bascule). Il peut être sensible au relâchement du bouton de la souris en choisissant Commutation au relâchement.



Commutation jusqu'au relâchement—La valeur du contrôle reproduit fidèlement l'état d'un bouton monostable.



Actions bufférisées (l'action de l'utilisateur est enregistrée jusqu'au moment où le programme lit la valeur du contrôle correspondant, utile pour les boutons poussoirs.)

Armement à l'appui—La commande change de valeur à l'appui sur le bouton de souris et retombe lors de la première lecture de cet état par le programme (mono coup). Cette commande est lue une et une seule fois à chaque action de l'utilisateur.



Armement au relâchement—Idem mais au relâchement du bouton de souris.

Armement jusqu'au relâchement—La commande change de valeur à l'appui sur le bouton de souris et retombe lors de la première lecture **après le relâchement**. La valeur est donc vraie tant que le bouton reste enfoncé. Elle peut ainsi être lue plusieurs fois

Modifiez l'action mécanique de l'interrupteur pour qu'il revienne seul en position par défaut lorsqu'il est appuyé.

Enregistrez et lancez l'exécution.

AJOUTER UNE TEMPORISATION

Lors que le VI s'exécute dans la boucle, il tourne aussi vite que possible, il est possible de ralentir l'exécution de la boucle à l'aide d'une temporisation. Nous utiliserons une temporisation de 0.5s.



Placez la fonction **Attendre un Multiple de ms** située dans la palette Fonctions» Temps & Dialogues. Cette fonction bloque le programme jusqu'à ce que l'horloge interne atteigne un multiple entier de son entrée, en l'occurrence 500 ms.



Note: si le temps d'exécution des autres VIs de la boucle est < à 500 ms, cette fonction assure un timing parfait. De plus, la fonction met cette partie de code dans un état "idle" (inhibé) ce qui rend du temps processeur pour les autres parties du programme.

500

Pour placer très simplement une constante à l'entrée d'un VI donné, cliquez à droite au niveau de la connexion d'entrée, et choisissez **Créer»Constante**

Enregistrez et lancez l'exécution.

Fin de l'exercice 4-1

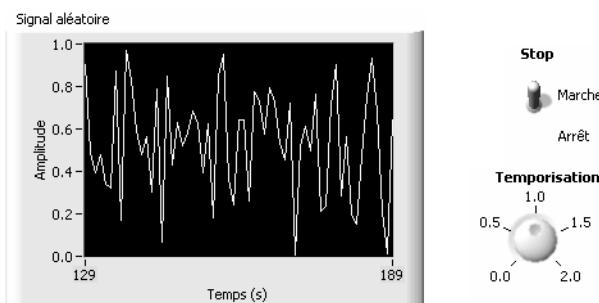
Exercice 4-2 : Signal aléatoire

OBJECTIF: CREER UN VI QUI GENERE UN NOMBRE ALEATOIRE ET L'AFFICHE A L'ECRAN

AJOUTER UNE TEMPORISATION VARIABLE A UN GRAPHE DEROULANT.

Réalisez les étapes suivantes :

Placez un bouton rotatif, un interrupteur et un graphe déroulant sur la face avant.



Configurez l'interrupteur pour qu'il revienne en position marche à chaque départ d'exécution, définissez les graduations du bouton (connexion jusqu'au relâchement) , les labels...

Construisez le diagramme en suivant les recommandations suivantes :

Utiliser Générer un nombre aléatoire situé dans Fonctions» Numérique pour fabriquer le données.

Multiplier la valeur du bouton rotatif par 1000 pour obtenir des ms, unité utilisée par le timer.

Enregistrez votre VI et l'exécuter changez la temporisation durant l'exécution et vérifiez son effet.

Fin de l'exercice 4-2

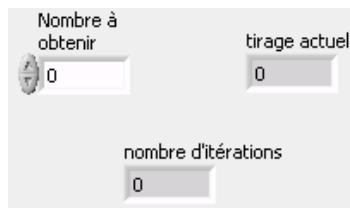
Exercice 4-3 : Avec de la chance

OBJECTIF: RENVOYER DES DONNÉES AU TRAVERS D'UNE BOUCLE, VIA UN TUNNEL.

Réalisez les étapes suivantes pour créer un VI qui génère des nombres aléatoires entiers compris entre 0 et 100 000 tant qu'aucun ne correspond à un nombre spécifié par l'utilisateur. La boucle, une fois terminée, fournit le nombre d'itérations nécessaires.

FACE AVANT

Créer la face avant suivante.



Nombre à obtenir est le nombre que vous voulez voir sortir au tirage.

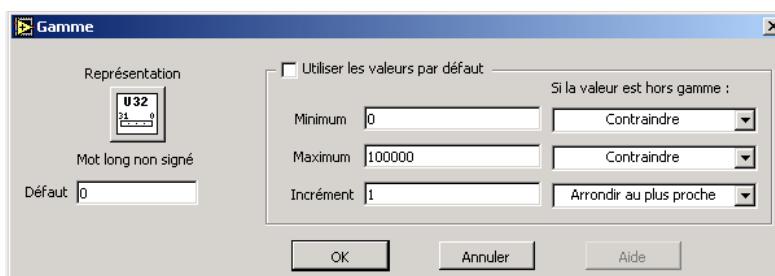
Tirage actuel est le nombre compris entre 0 et 100 000 que l'ordinateur vient de choisir.

Nombre d'itérations est le nombre de coups nécessaires pour obtenir le tirage.

Gamme des données.

Utilisez la propriété **Gamme des données** des objets numériques pour limiter l'étendue des valeurs possibles.

Par un clic droit, ouvrez la propriété Gamme des données de la commande Nombre à obtenir. Sélectionnez les options suivantes. (U32 pour Unsigned 32 bits)



DIAGRAMME

Construisez le diagramme, en vous aidant des indications suivantes :

- **Générer un nombre aléatoire** fournit un nombre compris entre 0 et 1, il faut donc le multiplier pour obtenir un nombre entre 0 et 100 000.
- **Fonctions»Numérique»Arrondir à l'entier le plus proche** permet d'arrondir la valeur obtenue après multiplication.
- **Fonctions»Comparaison»Différent** retourne une valeur booléenne capable d'arrêter l'exécution de la boucle While.
- **i** Ce terminal contient le numéro de l'itération courante (attention la première itération a le numéro 0 !). Il faut sortir cette valeur de la boucle en la connectant à un tunnel créé

automatiquement en câblant cet élément à une entrée de fonction ou à un indicateur extérieur à la boucle.



Fonctions»Numérique»Incrémenter incrémente de 1 l'entrée. Pour palier la valeur 0 associée à l'itération 1...

Sauvegardez votre VI sous le nom Avec de la chance.vi

Entrez un nombre dans la commande « Nombre à obtenir » et lancer le VI.



Vous pouvez observer le fonctionnement de la boucle grâce à l'outil Animer l'exécution qui fait apparaître le flot de données le long des fils, et les valeurs qui y circulent.

Entrez un nombre supérieur à 100000 ou inférieur à 0 et vérifiez que lors du lancement, LabVIEW constraint l'entrée au mini ou maxi le plus proche.

Fin de l'exercice 4-3

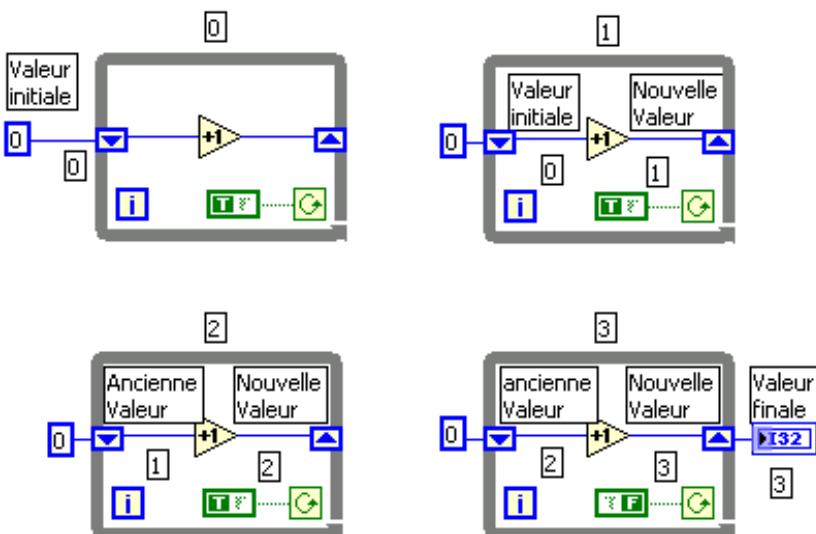
REGISTRES A DECALAGES

Les registres à décalages sont présents uniquement dans les boucles, et permettent à une itération i , de connaître la valeur d'une variable évaluée à l'itération $i-1$ (mais aussi $i-2, i-3\dots$). On insère un registre à décalage en cliquant à droite, à l'aide la bobine, sur le côté d'une boucle et en choisissant l'option « **Ajouter un registre à décalage.**»



Les registres à décalage apparaissent sous la forme d'une paire de terminaux. Le registre de droite reçoit des données de l'intérieur de la boucle, elles sont transférées au registre de gauche au début de l'itération suivante. Le registre de gauche est alors une source de données pour les éléments de l'intérieur de la boucle.

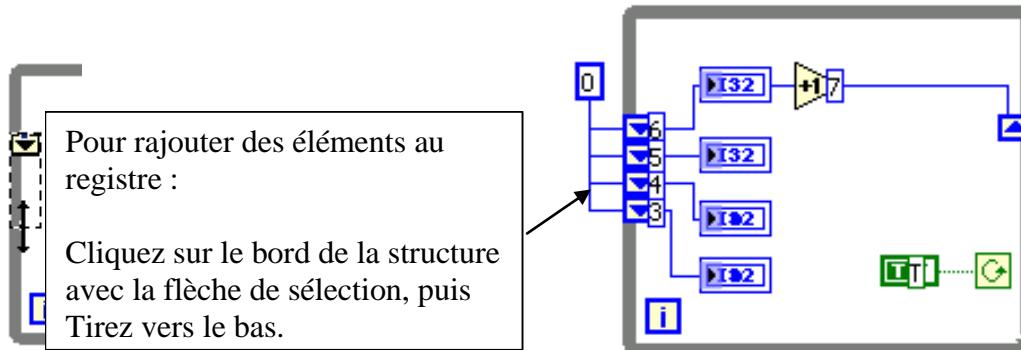
L'illustration suivante montre le fonctionnement du registre à décalage



Au temps 0, la valeur initiale est transférée dans le registre à décalage. Au temps 1, premier tour de boucle, la valeur initiale 0 est transférée pour être incrémentée, la valeur 1 est stockée dans la partie droite du registre. Lorsque tout le code contenu dans la boucle a été exécuté, la valeur 1 est transférée de la partie droite à la partie gauche du registre à décalage. Au temps 2, deuxième

tour de boucle, la valeur 1 est extraite du registre gauche puis incrémentée, le résultat 2 est stocké dans la partie droite du registre ... lorsque la boucle se termine, la dernière valeur peut sortir de la boucle par une connexion du registre extérieure à la boucle.

Il est possible de rajouter des éléments aux registres de gauche pour obtenir les valeurs à t-1, t-2, t-3.... comme l'illustre la figure ci après.



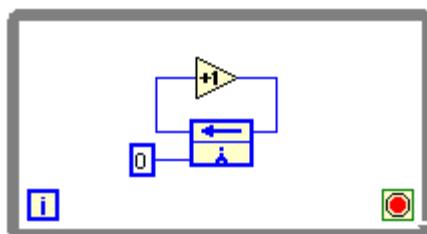
INITIALISATION DES REGISTRES

Pour initialiser un registre, nous avons vu qu'il suffisait de câbler une valeur, externe à la boucle, au registre de gauche.

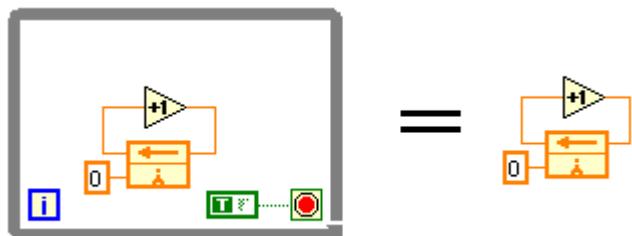
Si cette valeur n'est pas câblée, au premier appel du VI, le registre est initialisé à **0**, s'il s'agit d'un numérique et à **FAUX**, s'il s'agit d'un booléen. Aux appels suivants, les registres gardent la valeur qu'ils avaient au dernier appel.

NŒUDS DE RETROACTION (FEEDBACK NODE)

Si un grand nombre de registres à décalage sont utilisés, le diagramme s'encombre rapidement. Il est possible de transformer les registres à décalages en nœud de rétroaction par un clic droit sur le registre en sélectionnant **Remplacer par un nœud de rétroaction**



Les nœuds de rétroactions ont des options de configuration que dépassent le cadre d'une initiation. Sachez que l'on peut les initialiser, et qu'ils ne peuvent avoir qu'une sortie (par défaut à t-1). Les noeuds de rétroactions peuvent être cablés hors d'un boucle, dans ce cas, tout se passe comme s'ils étaient dans une boucle avec un condition de fin toujours vraie.



Exercice 4-4 : Moyennage de Température

OBJECTIF: UTILISER UN REGISTRE A DECALAGE POUR REALISER UNE MOYENNE GLISSANTE.

Reprenez le code de l'exercice 4-1 Surveillance de température.vi Modifiez le VI pour afficher, non pas la température instantanée, mais la moyenne des trois dernières prises de points.

FACE AVANT

Pas de modifications, sélectionnez Fichier»Enregistrer sous et renommez le fichier Surveillance de température moyenne.vi.

DIAGRAMME

Ajoutez un registre à décalage à la boucle.

Ajoutez un élément au registre de gauche (lors de la prise de température à l'instant t, vous aurez les valeurs aux instants t-1 et t-2 pour moyenner sur trois éléments).

Modifiez le diagramme pour visualiser la moyenne des trois derniers points sur le graphe. Tenez compte des remarques suivantes :



L'opérateur Fonctions»Numérique»Opérateur arithmétique permet d'appliquer le même opérateur à plusieurs entrées (simplification du diagramme).

Placez une temporisation de 500 ms pour modérer les ardeurs de l'exécution.

Enregistrez les modifications et exécutez le VI.

A chaque itération, le VI thermomètre fournit une température, le VI additionne cette température aux deux dernières stockées dans le registre et divise le résultat par trois.

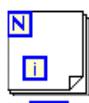


Placez la fonction Fonctions»Cluster»Assembler pour permettre l'affichage simultané de la température instantanée et de la température moyenne. L'aide contextuelle sur les graphes fournit les renseignements essentiels pour cette opération.

Enregistrez les modifications et exécutez le VI, changez le type d'affichage des points, des traits... pour chacune des courbes affichées.

Fin de l'exercice 4-4

BOUCLES FOR



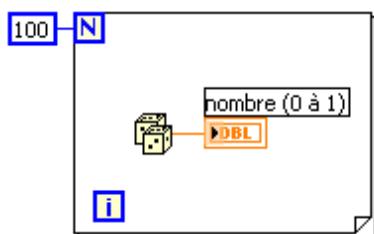
La boucle d'itération (boucle For) représentée à gauche exécute un nombre de fois déterminé la partie de diagramme qu'elle englobe. Ce type de boucle est accessible dans :



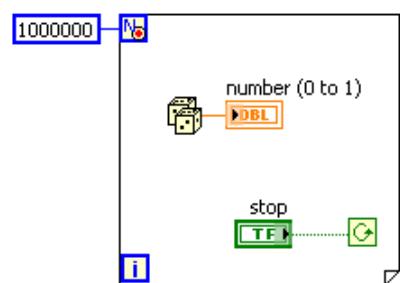
Fonctions»Structures»Boucles For. Une fois lancée, il est possible d'arrêter l'itération prématurément (équivalent du *break*), il faut au préalable afficher le terminal de condition d'arrêt avec un click droit en bordure du cadre de la boucle.

L'entrée **N** correspond au nombre d'itérations à faire. Le terminal **i**, au numéro de l'itération en cours.

La boucle suivante tire 100 nombres aléatoires.



Celle-ci en tire un million et possède un bouton d'arrêt



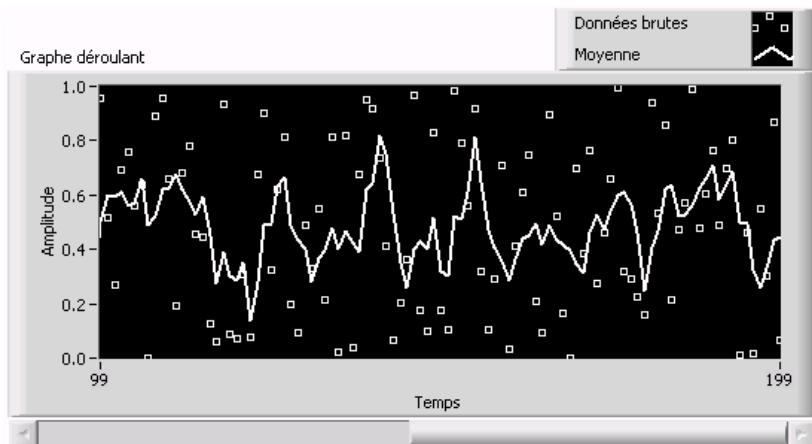
Exercice 4-5: Moyenne de nombres aléatoires

OBJECTIF: CRÉER UN VI QUI AFFICHE DEUX COURBES, UN NOMBRE ALÉATOIRE, ET UNE MOYENNE GLISSANTE DES 4 DERNIERS POINTS.

Construisez un VI en suivant les recommandations suivantes:

Utilisez une boucle For de 200 itérations.

Créez une face avant ressemblant à celle-ci :



Utilisez un registre à décalage pour conserver les trois dernières valeurs.

Vous pouvez utiliser une boîte de calcul (Fonctions » Structures » Boîte de calcul) pour réaliser le tirage, la somme et la division.

Enregistrez les modifications et exécutez le VI.

Fin de l'exercice 4-5

EXERCICE SUPPLEMENTAIRE

Réaliser un VI qui lit l'état d'un bouton rotatif une fois par seconde et l'affiche sur un graphe déroulant. Deux commandes situées sur la face avant permettent de fixer des seuils d'alarme haute et basse. Deux voyants s'allument lorsque ces seuils sont dépassés dans un sens ou dans l'autre. En plus de la température instantanée, le graphe (de type multi-courbe) affiche les valeurs minimale et maximale du contrôle.

5. TABLEAU, GRAPHES ET CLUSTERS

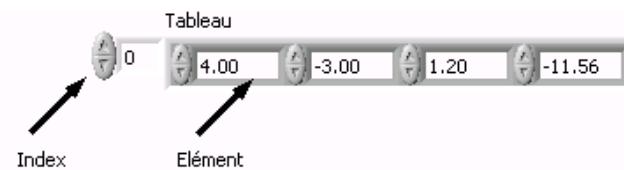
LES TABLEAUX

Les tableaux sont des groupements ordonnés de données de même type. La dimension d'un tableau est le nombre d'indices définissant un élément. (1 dimension, le tableau contient 1 seule colonne, 2 dimensions : ligne/ colonne, 3 dimensions : ligne/ colonne/ profondeur...). Chaque élément est ainsi repéré de façon unique par ses coordonnées.

Les tableaux peuvent contenir n'importe quel type d'éléments (diodes, interrupteurs...), excepté d'autres tableaux.

CRÉER DES TABLEAUX SUR LA FACE AVANT.

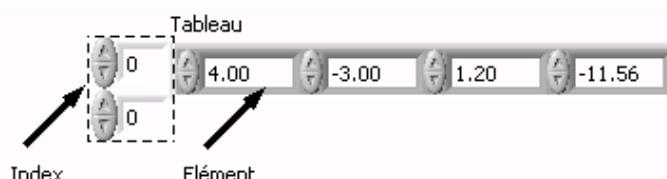
Pour créer un tableau sur la face avant, sélectionnez **Commande»Tableaux et Cluster»Tableaux**. Cette commande place un tableau vide sur la face avant. A présent déposez un indicateur ou une commande du type voulu, à l'intérieur du tableau.



Vous pouvez changer la taille de l'affichage des éléments, ainsi que le nombre d'éléments visibles. L'index indique le numéro du premier élément affiché.

TABLEAUX À DEUX DIMENSIONS

Les données des tableaux à deux dimensions sont affichées dans une grille lignes/colonnes. On augmente la dimension en redimensionnant l'affichage de l'index vers le bas.



Il devient alors possible d'étirer aussi l'affichage des éléments.



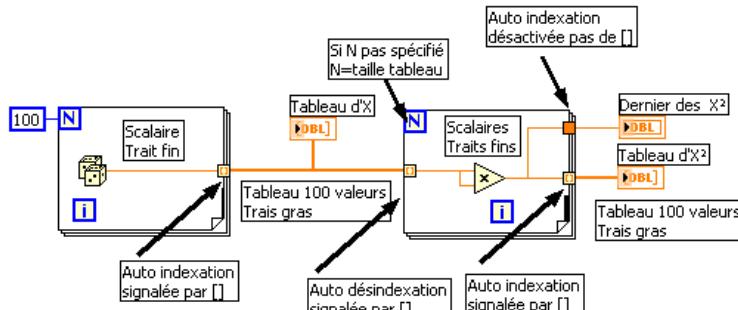
La méthode s'applique aux tableaux de toutes dimensions. Cependant, l'affichage reste 2D...

TABLEAUX DE CONSTANTES

Même méthode, mais dans le diagramme : **Fonctions » Tableau » Constante tableau** puis placer une constante du type voulu dans la structure vide.

AUTO INDEXATION

Lorsque l'on câble une connexion vers l'extérieur d'une boucle FOR, LabVIEW stocke **automatiquement** les valeurs successives dans un tableau, c'est **Auto Indexation**. De façon réciproque, un tableau connecté à l'entrée d'une boucle sera automatiquement désindexé (les éléments sont passés un à un), et la boucle sera exécutée autant de fois qu'il y a d'éléments dans le tableau. L'illustration suivante illustre ce mécanisme.



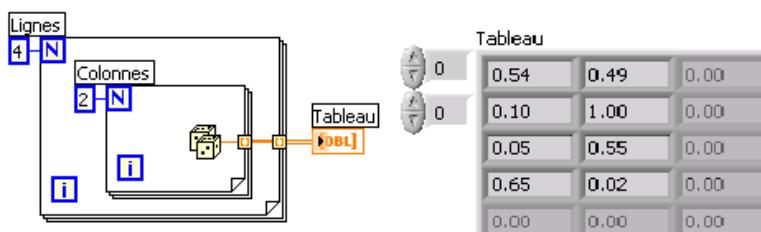
L'auto indexation est désactivée en cliquant à droite sur le tunnel et en sélectionnant **Désactiver l'indexation**.



Note Parce que les boucles FOR sont plus spécifiquement dédiées à l'exploration des tableaux, l'auto indexation y est automatiquement activée, en revanche dans les boucles While, elle ne l'est pas, il faut agir comme ci-dessus en sélectionnant **Activer l'indexation**.

BOUCLES ET TABLEAUX 2D

Pour construire un tableau 2D, il suffit d'imbriquer deux boucles FOR. La boucle intérieure crée les colonnes, la boucle extérieure, les lignes. Notez, l'augmentation de l'épaisseur des fils de liaisons avec le nombre de dimensions du tableau. Le tableau 2D peut également être créé en ajoutant les lignes les unes aux autres (cf exo 5.1)



FONCTIONS SUR TABLEAUX

Les fonctions sur les tableaux sont accessibles par **Fonctions»Tableaux**. Parmi les fonctions les plus fréquemment utilisées, citons :

Initialiser un tableau

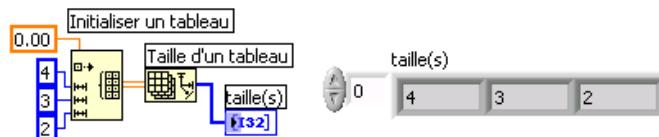


Crée un tableau à N dimensions dont tous les éléments ont la même valeur. Cette fonction (comme la plupart des fonctions opérant sur les tableaux), est redimensionnable pour s'adapter aux nombres de dimensions des tableaux. Cette fonction initialise donc le tableau.

Taille d'un tableau



Renvoie la taille du tableau. Si le tableau est à N dimensions, la fonction renvoie un tableau contenant la taille de chaque dimension.



Construire un tableau



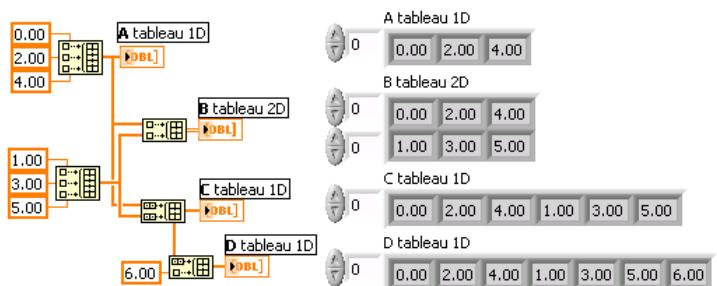
Permet de :

Placer des éléments dans un tableau. S'il s'agit de scalaires séparés, le résultat est un tableau 1D (tableau A ci-dessous) ; s'il s'agit de plusieurs tableaux 1D, le résultat obtenu est un tableau 2D (Tableau B). Mettre alors le Vi dans une boucle.

Ajouter en début ou fin d'un tableau, un élément (Tableau D).

concaténer des tableaux de mêmes dimensions, en un tableau unique de mêmes dimensions (Tableau C)

La concaténation est activée en cochant l'option **Concaténer les entrées** du menu contextuel.



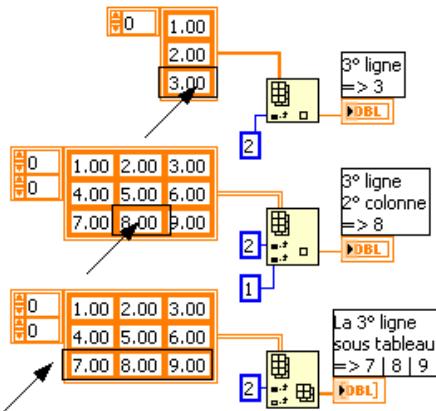
Indexer un tableau



Permet de:

Extraire un élément du tableau d'entrée.

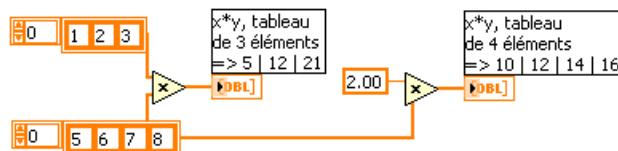
Extraire une ligne (ou une colonne) d'un tableau 2D



POLYMORPHISME ET TABLEAUX

Les fonctions arithmétiques sont valides pour les calculs sur les tableaux. Toutes ces fonctions sont polymorphiques, c'est à dire qu'elles opèrent sur des variables de types différents.

Par exemple, la fonction multiplier, peut multiplier les éléments du tableau 1 avec les éléments du tableau 2 câblés à ses entrées. Si un tableau et un scalaire sont reliés aux entrées, l'ensemble des éléments du tableau est multiplié par le scalaire.



EXERCICE 5-1: GENERATION D'UNE TABLE TRIGONOMETRIQUE

OBJECTIF: CREER UN TABLEAU 2D CONTENANT LES ANGLES ET LEURS FONCTIONS TRIGONOMETRIQUES SIN, COS, ET TG.

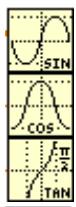
L'exercice suivant vous propose de réaliser une table trigonométrique donnant, pour des angles allant de 0 à 90° par pas de 10°, les sinus, cosinus et tangentes.

FACE AVANT

Créez la face avant suivante :

	Angle	Sinus	Cosinus	Tang
0	0.00E+0	0.00E+0	1.00E+0	0.00E+0
10	10.00E+0	173.65E-3	984.81E-3	176.33E-3
20	20.00E+0	342.02E-3	939.69E-3	363.97E-3
30	30.00E+0	500.00E-3	866.03E-3	577.35E-3
40	40.00E+0	642.79E-3	766.04E-3	839.10E-3
50	50.00E+0	766.04E-3	642.79E-3	1.19E+0
60	60.00E+0	866.03E-3	500.00E-3	1.73E+0
70	70.00E+0	939.69E-3	342.02E-3	2.75E+0
80	80.00E+0	984.81E-3	173.65E-3	5.67E+0
90	90.00E+0	1.00E+0	61.23E-18	16.33E+15

DIAGRAMME



Construisez le diagramme en tenant compte de ces éléments :

Les fonctions trigonométriques sont dans la palette Numérique»Trigonométrique. Les angles sont en radians.

Utilisez la fonction Construire un tableau pour assembler dans une ligne de tableau : angle, sin, cos et tg.

Une boucle For permettra de calculer les 10 ensembles de valeurs.

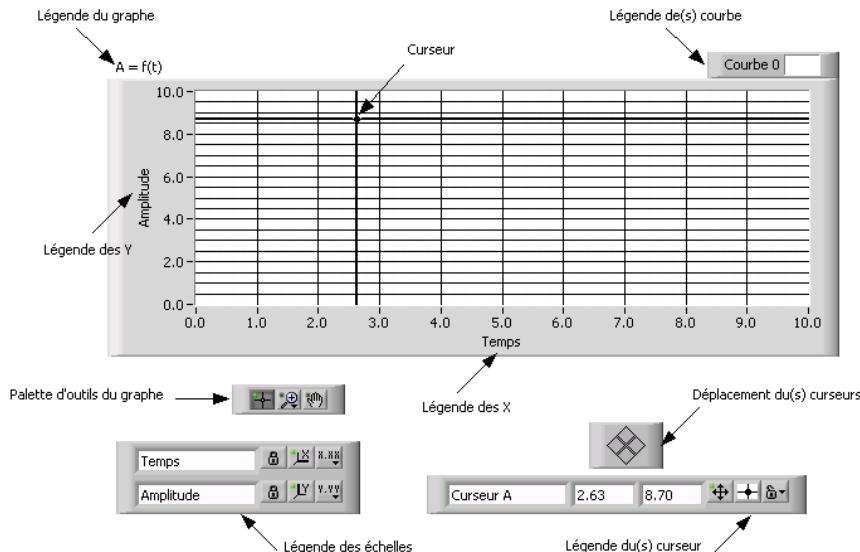
Enregistrez le VI sous le nom trigo.vi, exécutez le.

Il est possible de remplacer les icônes de calculs par une boîte de calcul. Tentez de remplacer l'ensemble des fonctions numériques par une seule boîte de calcul.(Cf Chap6 §C)

Fin de l'exercice 5-1

GRAPHES FONCTION DU TEMPS ET GRAPHES XY

Les VIs utilisant des graphiques collectent généralement les données dans des tableaux, les traitent puis les affichent. L'illustration suivante montre les éléments d'un graphe



Les graphes sont accessibles par **Commande»Graphes**. Les graphes inscrivent les points en donnant une abscisse initiale et un incrément à chaque nouveau point, cela suppose donc des points régulièrement distribués sur l'axe des X (par ex. toutes les secondes, tous les °C...). Les graphes de type XY nécessitent un tableau de valeurs d'X, et peuvent donc afficher des points pris non régulièrement. Ces deux graphes sont capables d'afficher des courbes multiples.

LES GRAPHES

Les graphes acceptent :

Graphes :	
Câblez les données directement au graphe :	
Tableau Y	Graphe résultant
1D	Courbe unique
WDT	Courbe unique
2D	Multicourbes

Le type WDT contient les infos temporelles.
Les autres types ont par défaut la valeur 0 pour x_0 et 1 pour Δx . Combinez les informations temporelles via la fonction assembler : x_0 Δx tableau Y graphique

Consultez l'exemple : [Waveform graph.vi](#)

En mono courbe

Soit un tableau 1D, dans ce cas, l'axe des X est gradué, par défaut de 1 en 1 à partir de 0. On peut modifier ces valeurs à partir du menu contextuel **Echelle des X»Formatage...**

Soit un cluster contenant la valeur de X_0 , la valeur de Δx , et le tableau d'Y nécessaires à la construction de la forme d'ondes (waveform)

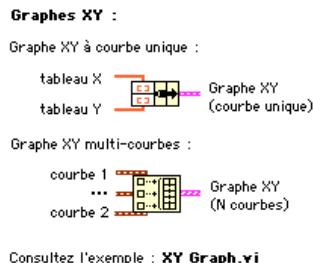
Multi courbes

Soit un tableau 2D, chaque ligne correspond à une courbe. Dans ce cas, l'axe des X est gradué, par défaut de 1 en 1 à partir de 0. On peut modifier ces valeurs en prenant au clic droit **Echelle des X»Formatage...**

Soit un cluster contenant la valeur de X_0 , la valeur de Δx , et le tableau 2D d'Y comme précédemment. Il n'y a qu'un seul axe des X, et donc qu'une seule graduation pour les deux courbes (même X_0 et Δx).

LES GRAPHES XY

Les graphes XY acceptent :



En mono courbe

Un couple de tableau d'X et d'Y assemblés dans un cluster par la fonction **Cluster»Assembler**.

Multi courbes

Un tableau 1D, dont chaque colonne contient les données (portées en abscisse) d'une courbe.

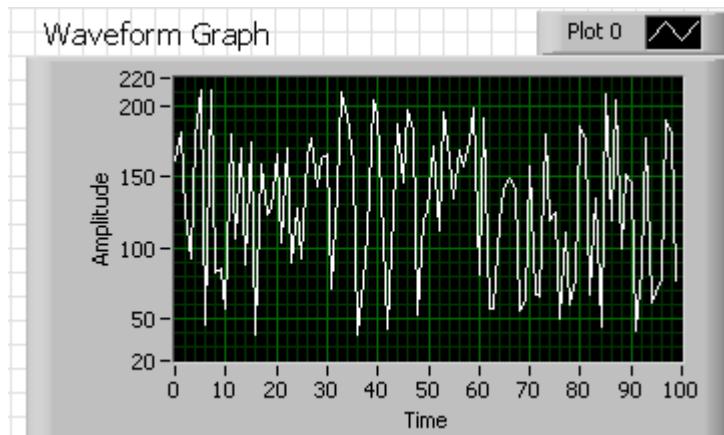
Exercice 5-2 : Graphe

OBJECTIF: FAIRE UNE BOUCLE POUR CRÉER UN TABLEAU 1D ET AFFICHER LE RÉSULTAT SUR UN GRAPHE (DE TYPE GRAPHE ET PAS GRAPHE XY).

On se propose de tracer l'évolution d'une température (simulée) durant 100 points d'acquisition comme dans l'exercice 4-1. Vous utiliserez une boucle For.

FACE AVANT

Vous devriez obtenir quelque chose comme (il s'agit d'un objet de type GRAPHE) :

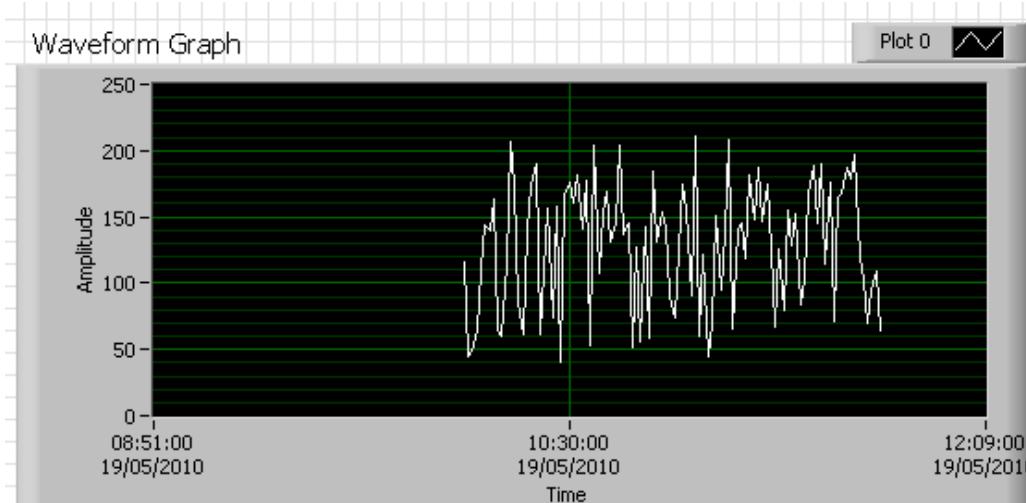


DIAGRAMME

-  Modifiez le code pour avoir un $X_0 = \text{« temps actuel en secondes depuis } 1/1/1904 \text{ »}$ et $\Delta x = 60\text{s}$, en utilisant la fonction Cluster»Assembler. Relancez l'exécution pour constater le changement sur l'échelle des X. (Attention de respecter l'ordre X_0 , Δx , température dans le câblage du VI « assembler »).
-  Get Date/Time permet d'obtenir la date actuelle dans un format propre à Labview.
-  La fonction To Unsigned Long Integer permet de l'obtenir en secondes

FACE AVANT

Changez les propriétés de formatage de l'échelle des X, choisissez le format « Heure & Date, 24 heures, HH :MM :SS, J/M/A, Année sur 4 chiffres ». Vous devriez obtenir :



GRAPHE MULTI COURBES

DIAGRAMME

-  Ajoutez un VI : DigitalThermometer.vi dans la boucle For pour obtenir deux flots de données, réunissez les deux tableaux 1D en un tableau 2D grâce à Tableau»Construire un tableau qui entrera dans la fonction Assembler en lieu et place du tableau 1D précédent. (attention Assembler ≠ Concaténer, l'item Concaténer les entrées ne doit pas être coché.)

FACE AVANT

Faire apparaître la « Palette du graphe » depuis le menu contextuel éléments visibles

Pour zoomer une partie du graphique :



Sélectionnez l'outil Loupe à l'aide de l'outil doigt. Un menu se déroule alors.

Les options sont :

<Zoom dans un rectangle>. <Zoom entre deux X>. <Zoom entre deux Y>. <Annuler le zoom>. <Agrandir à chaque clic>. <Rétrécir à chaque clic>

Essayez ces différentes fonctionnalités.

L'outil déplacement permet de mouvoir la courbe dans sa fenêtre lorsqu'on l'a agrandie.



L'outil curseur est l'outil par défaut pour déplacer les curseurs, mais c'est pour plus tard...

Enregistrez le VI sous le nom multigrapher.vi et fermez le.

Fin de l'exercice 5-2

Exercice 5-3 : Analyse de l'évolution de Température

OBJECTIF: AFFICHER ET ANALYSER L'EVOLUTION D'UNE TEMPERATURE.

Créez de nouveau une boucle FOR de 40 itérations contenant un Thermometer.vi qui fournira un tableau de température:

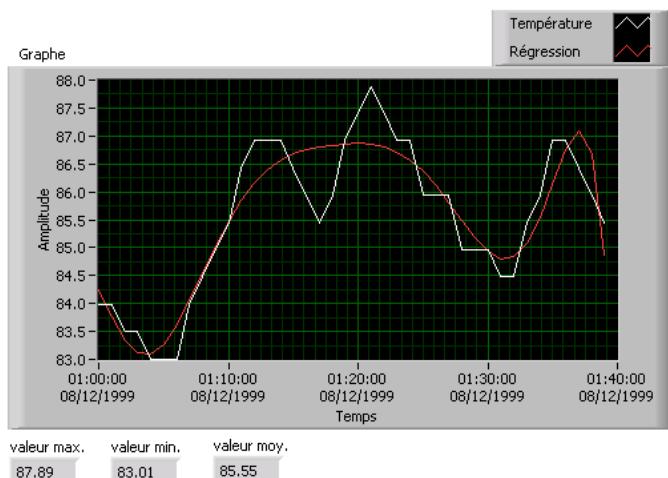
Vous en afficherez le minimum et le maximum

La moyenne

Vous calculerez une régression polynomiale d'ordre 8 et ajouterez la courbe régressée au graphe.

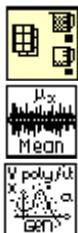
FACE AVANT

Vous devriez obtenir quelque chose comme :



DIAGRAMME

Utilisez les éléments nouveaux suivants : (Pour les détails, utilisez l'aide en ligne)



Tableau» Max & Min d'un tableau pour rechercher les extrema du tableau de température.

Fonctions » Analyse » Mathématiques» Probabilité et Statistiques »Moyenne pour calculer la moyenne des valeurs du tableau de température.

Mathématiques»Ajustement de courbes»Ajustement polynomial général pour calculer le tableau de valeurs ajustées. Le tableau d'X que réclame la fonction sera créé le plus simplement possible... L'ordre de régression est de 8. Vous procéderez comme précédemment pour ajouter la courbe régressée au graphe.

Enregistrez le VI sous le nom analysetempérature.vi et lancer l'exécution du code.

Fin de l'exercice 5-3

CLUSTERS

Les clusters sont des assemblages d'éléments disparates qui sont l'équivalent des structures en langage C. Ils permettent de rassembler dans une même connexion des données de types différents ayant une relation plus ou moins proche entre elles (préférable pour l'analyse !), par exemple Nom, Prénom, Age.

Il est important d'utiliser les clusters pour deux raisons :

Regrouper les données dans des ensembles structurellement cohérents

Diminuer le nombre des connexions des VIs, limité à 28.



Notes : Il est impossible de regrouper des contrôles et des indicateurs dans le même cluster. Il est possible d'assembler et de désassembler les éléments d'un cluster, soit comme en C, par leur nom, soit par leur ordre.

CREER DES CLUSTERS SUR LA FACE AVANT



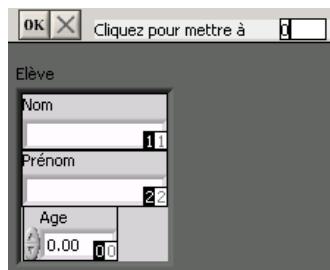
Pour créer un cluster sur la face avant, sélectionnez **Tableau & Cluster»Cluster** depuis le menu contextuel/**Commandes**. Une fois le cadre posé, déposez à l'intérieur les différents éléments constitutifs. L'ordre de placement sera l'ordre des éléments dans le cluster leur position physique ne change rien à l'ordre, les étiquettes seront le nom des éléments du cluster. Tous les éléments sont déplaçables, et modifiables, la taille du cluster peut être ajustée à la main ou automatiquement. Cependant, un cluster étant un ensemble, cet ensemble est, soit une commande, soit un indicateur. Les éléments le constituant sont ainsi **tous** des commandes, ou **tous** des indicateurs.

CONSTANTES DE TYPE CLUSTER

Il est possible de créer sur le diagramme une constante de type cluster, en utilisant **Cluster»Cluster constante**, puis en y plaçant comme précédemment des constantes du type désiré.

ORDONNER LES CLUSTERS

Les éléments de cluster sont ordonnés, nous l'avons dit, dans leur ordre d'insertion. Si vous désirez changer l'ordre des éléments d'un cluster (par ex., vous ajoutez un élément omis qui ne doit pas être le dernier) il faut choisir, dans le menu contextuel, **Ordonner les commandes dans le cluster**. La barre d'outils prend alors l'aspect suivant :



Chaque élément voit son numéro d'ordre apparaître à droite de la commande, en gris sur fond blanc, l'ordre avant modification, sur fond noir l'ordre actuel. Il suffit de cliquer sur chaque commande dans l'ordre souhaité pour réordonner le cluster.

Attention, changer l'ordre d'un cluster peut rendre le programme non exécutable : si l'élément n'était une chaîne, et qu'une fois réordonné il s'agisse d'un nombre, toutes les liaisons entre cet élément et les fonctions qui l'utilisent, seront brisées. De fait, nous recommandons vivement d'utiliser l'assemblage et le désassemblage par nom qui évite cet inconvénient, mais qui est cependant plus lourd à utiliser.

FONCTIONS SUR CLUSTER

Quatre fonctions principales permettent d'assembler ou de désassembler par ordre ou par nom les clusters, elles sont situées dans la palette d'outils **Cluster**

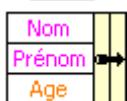


ASSEMBLER & ASSEMBLER PAR NOM

Les fonctions d'assemblages permettent de regrouper divers éléments dans un cluster ou de



changer la valeur de certains éléments d'un cluster existant. Ces fonctions sont redimensionnables.

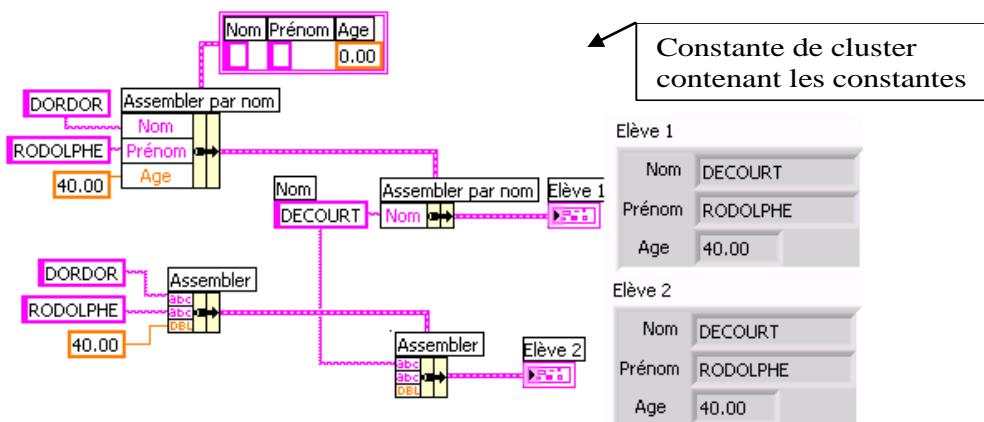


L'exemple ci-dessous montre l'utilisation de ces fonctions. Les fonctions de gauche créent un cluster à partir de constantes, celles de droite modifient un élément du cluster créé. Le problème est résolu simultanément avec un assemblage par nom et avec un assemblage par ordre.

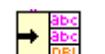
Plusieurs remarques s'imposent :

L'assemblage par nom est éminemment plus clair! Si le cluster n'existe pas déjà, la connexion cluster d'entrée de la fonction cluster/assembler doit être reliée à une constante, de type cluster. Il faut ainsi mettre dans celle-ci les prototypes ou les valeurs initiales des éléments du cluster. Il est possible de sélectionner l'élément dont on veut changer la valeur : clic avec l'outil doigt sur l'entrée ou menu contextuel/sélectionner un élément.

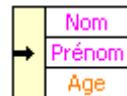
L'assemblage par ordre est plus compact, on ne câble que l'entrée qui doit changer de valeur, mais on ne sait pas à quoi elle correspond (il faut alors commenter le code).



DESASSEMBLAGE DES CLUSTERS

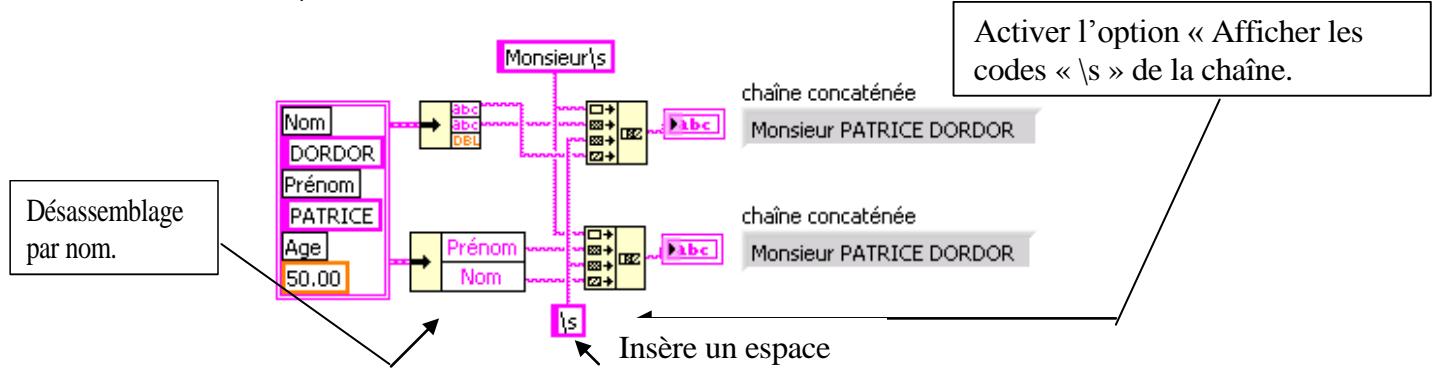


Les fonctions de désassemblage permettent d'extraire des éléments d'un cluster pour les utiliser individuellement. La fonction de désassemblage par ordre s'ajuste automatiquement au nombre total d'éléments du cluster. Le désassemblage par nom doit être dimensionné à la main en



fonction du nombre d'éléments à extraire, l'outil doigt permettant de sélectionner le nom des éléments.

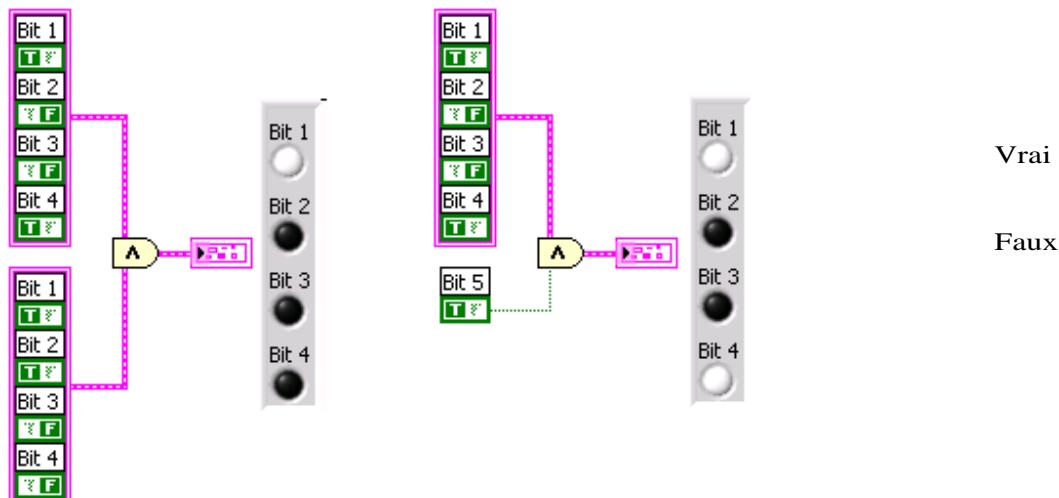
L'exemple suivant montre l'utilisation des fonctions de désassemblage, notez que le désassemblage par nom permet d'extraire le nombre souhaité d'éléments et dans l'ordre le plus pratique pour le câblage. Le caractère « \s » apparaissant dans les constantes chaînes est le caractère d'échappement correspondant à « espace ». Les fonctions les plus à droite sont des fonctions de concaténation de chaîne.



POLYMORPHISME DES CLUSTERS

Si les données d'un cluster sont de même type (chaîne, nombre, booléen), il est possible d'effectuer directement des opérations logiques sur des clusters.

L'exemple suivant montre le résultat obtenu lors d'un « ET » logique entre deux clusters de booléens puis entre un cluster de booléens, et une constante booléenne. Dans le premier cas le « ET » est réalisé élément de cluster à élément de cluster (par ordre de création et pas par ordre visualisé à l'écran), dans le second cas entre chaque élément du cluster et la constante (la LED noire indique une valeur fausse).



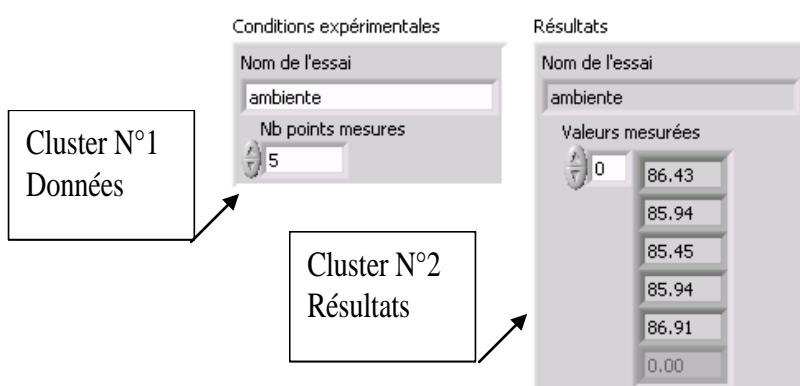
Exercice 5-4 : Paramètre de mesure et résultats

OBJECTIF: CREER DES CLUSTERS SUR LA FACE AVANT ET UTILISER LES FONCTIONS D'ASSEMBLAGE ET DE DESASSEMBLAGE.

Les conditions expérimentales d'un essai sont regroupées dans un cluster qui contient les éléments suivants : le nom de l'essai et le nombre de points de mesure. Les résultats seront regroupés dans un autre cluster contenant le nom de l'essai, et un tableau de valeurs d'essai. Ces valeurs seront produites comme précédemment par une boucle contenant le VI Digital Thermometer.vi

FACE AVANT

La face avant peut ressembler à :



DIAGRAMME

Le diagramme n'utilise pas de fonctions nouvelles particulières, il est conseillé d'utiliser les fonctions d'assemblage et de désassemblage par nom.

Enregistrer le VI sous le nom Cluster manip.vi puis l'exécuter.

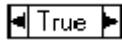
Fin de l'exercice 5-4

6. STRUCTURES DE CHOIX, SEQUENCES ET NŒUDS DE CALCUL

STRUCTURE DE CHOIX



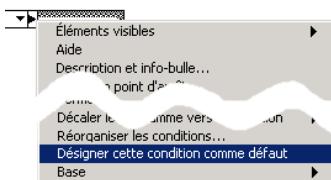
La structure de choix correspond à la structure « if...then...else » utilisée dans les langages textuels.



Elle comporte un sélecteur de choix dans sa partie centrale supérieur (le « case » du C) et une entrée de sélection (le « Switch » du C) en forme de point d'interrogation. Les flèches de part et d'autre du sélecteur de choix, permettent de faire défiler les divers cas possibles.

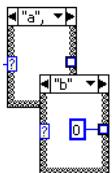
Le code de chaque cas est écrit dans la case correspondante, une seule case est visible à la fois à l'écran, elles sont détaillées à l'impression.

Par défaut, l'entrée de sélection est booléenne, il est cependant possible d'y connecter des entiers, des énumérations (collection d'entiers et de labels disponibles dans **Commande»Menu déroulant & enum**) et des chaînes. Le sélecteur de choix s'adapte automatiquement au type d'entrée connectée.



Un <cas par défaut> doit être spécifié si l'ensemble des cas possibles correspondant à l'entrée de sélection n'est pas décrit (ex : l'entrée de sélection est reliée à un entier ou un chaîne). On spécifie le <cas par défaut> en sélectionnant **Designer cette condition par défaut**, dans le menu contextuel du sélecteur de cas. Ce menu permet aussi d'ajouter, d'enlever ou de dupliquer des cas.

TUNNELS D'ENTREE/SORTIE



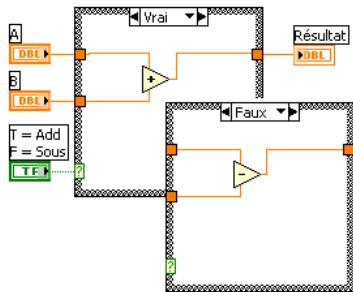
Il est possible de créer des tunnels d'entrée et de sortie sur une structure de choix. Cependant, les connexions de sorties doivent être **évaluées** dans **tous les cas**. Ainsi tant qu'un tunnel de sortie n'est pas câblé pour tous les cas du sélecteur, l'intérieur du cadre du tunnel reste blanc, et la flèche d'exécution est brisée.

EXEMPLES

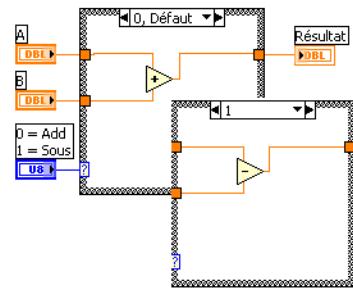
Les exemples suivants exposent les différents types de variables connectables à l'entrée de sélection. Chaque exemple est identique et opère soit une addition, soit une soustraction en fonction d'une variable de choix.

Structure de choix sur booléen

Attention, à l'écran, tous les cas sont superposés dans une seule et unique boite. Cette représentation didactique est le fruit d'un montage.

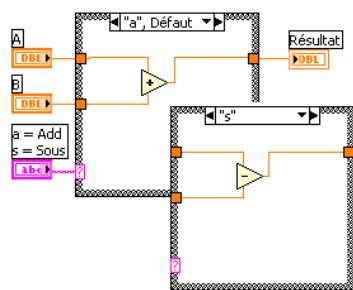


Structure de choix sur entier

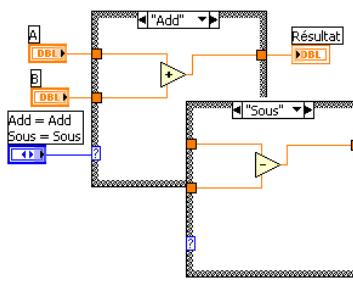


: apparition d'un cas par Défaut, pour définir les opérations à faire dans les cas non prévus.

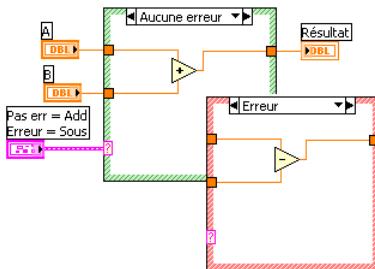
Structure de choix sur chaîne



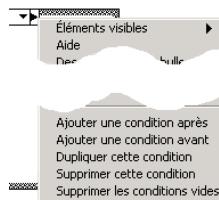
Structure de choix sur énumération



Structure de choix sur erreur



SELECTION DES CHOIX



Pour entrer la valeur de sélection, frappez à l'aide de l'outil texte la valeur désirée dans la case de sélection. Le menu contextuel permet d'ajouter des cas avant ou après le cas actuel, de dupliquer le code d'un cas, d'enlever un cas. Pour chaque cas, il est possible d'entrer une valeur unique, une liste de valeurs distinctes ou une étendue de valeurs continues.

Pour entrer une liste, il faut séparer les valeurs par des virgules. Pour entrer une étendue, il faut entrer la borne inférieure, deux points et la borne supérieure (les bornes sont incluses). Si une borne est omise, cela signifie que toutes les valeurs inférieures (ou supérieures) sont admises.

Par exemple : ..5, 7, 11..20, 30 signifie que ce cas sera exécuté pour toutes valeurs inférieures à 5, pour la valeur 7, les valeurs comprises entre 11 et 20 et la valeur 30.

EXERCICE 6-1 : RACINE CARREE

OBJECTIF: UTILISER UNE STRUCTURE DE CAS SIMPLE.

Il s'agit de réaliser un VI capable de calculer la racine carrée d'un nombre positif ou nul, et d'afficher un message d'erreur si l'utilisateur entre une valeur négative. (l'indicateur Racine est **dans** la structure)

FACE AVANT



DIAGRAMME

Les éléments nouveaux hormis la structure de choix, sont :

Les outils de comparaison disponibles dans la palette **Fonctions»Comparaison**.

Une boîte de dialogue à un bouton, accessible dans la palette de **Fonctions»Temps & Dialogue**.

La fonction racine carrée accessible dans la palette de **Fonctions»Numérique**

Enregistrez le VI sous le nom Racine.vi et exécutez le.

FIN DE L'exercice 6-1

EXERCICE 6-2 : PROGRAMMATION D'UN APPAREIL IEEE488

OBJECTIF: UTILISER UNE STRUCTURE DE CHOIX ET UNE ENUMERATION.

Le choix de la fonction d'un multimètre interfacé se fait par l'envoi de codes ASCII. Le VI est chargé de transformer une énumération de fonctions en chaînes de programmation selon la relation suivante :

Fonction	chaîne de programmation
Voltmètre	FOX
Ampèremètre	F1X
Ohmmètre	F2X

FACE AVANT



Elle contient une commande de type énumération et un indicateur de type chaîne de caractère. Les énumérations sont des ensembles finis de valeurs, associant un entier à un label texte. Vous les trouverez dans la palette Commandes»Menu déroulant & Enum. Ceci permet d'offrir un choix de valeurs limitées à l'utilisateur.

La face avant peut ressembler à :



Les labels de l'énumération sont entrés à l'aide de l'outil texte, il est possible d'ajouter avant ou après la position actuelle des éléments, et d'en retirer dans le menu contextuel de l'énumération.

DIAGRAMME

Le diagramme ne contient qu'une structure de choix, vous constaterez que le sélecteur de la structure connaît automatiquement les choix de l'énumération une fois l'entrée de sélection reliée.

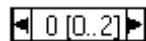
Enregistrez le VI sous le nom MotPrg.vi et exécutez le.

FIN DE L'exercice 6-2

LES SEQUENCES



Une séquence est un artifice pour imposer l'ordre d'exécution de VI n'ayant pas de liaison (si il y avait des liaisons, l'ordre d'exécution serait imposé par le flux de données). Sur le plan graphique, son cadre ressemble à une diapositive, les diapositives sont placées les unes derrières les autres ; l'exécution commence par le code contenu dans la première (n°0) puis continue dans l'ordre 1, 2, 3...



Un sélecteur de séquences permet d'écrire le code dans la séquence choisie, il est possible d'enlever, d'ajouter, de déplacer des séquences par le biais du menu contextuel du sélecteur.

VARIABLE LOCALE DE SEQUENCE



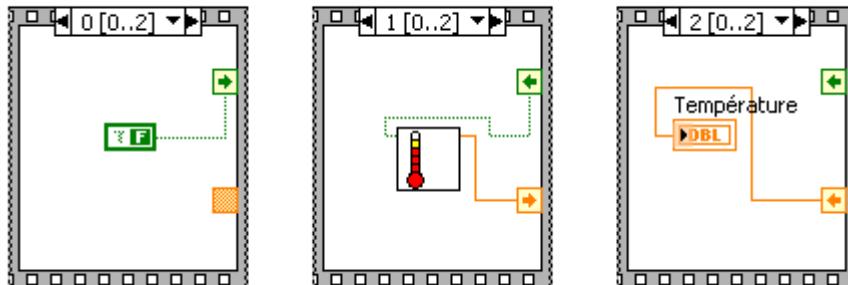
Le passage de variable(s) d'une séquence à l'autre se fait par des variables locales de séquences, ajoutées par le menu contextuel.

L'exemple suivant en montre l'utilisation :

Séquence 0, la variable locale *température* n'a pas été affectée, elle est grise, et inutilisable. Le booléen est initialisé à Faux

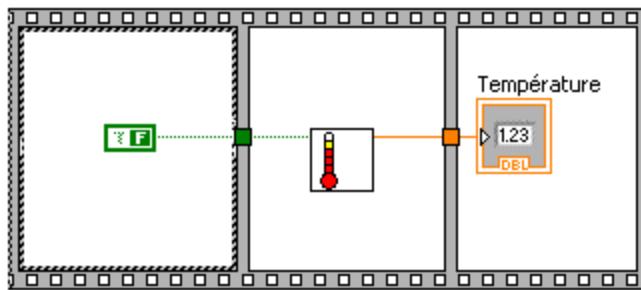
Séquence 1, la variable locale est *température* affectée, la flèche indique que cette valeur sort de la séquence, le booléen est lu, la flèche indique que la valeur entre dans la séquence.

Séquence 2 et suivantes, la variable locale peut être connectée à l'entrée d'un VI, la flèche indique que cette valeur entre dans la séquence.



II

Il existe aussi des séquences « à plat » qui offrent une vision plus claire des opérations séquentielles, mais plus encombrante ! Il est possible via le menu local de passer d'une représentation à l'autre.

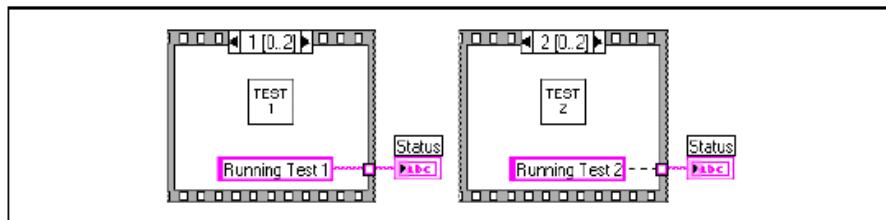


COMMENT EVITER L'UTILISATION DES SEQUENCES

L'utilisation des séquences va à l'encontre du concept de flux de données de LabVIEW, et diminue grandement la lisibilité du code en superposant des pages de code dans le même cadre. Il est grandement préférable d'utiliser l'ordre d'exécution naturel en liant vos VI, ne serait-ce que par une connexion d'erreur.

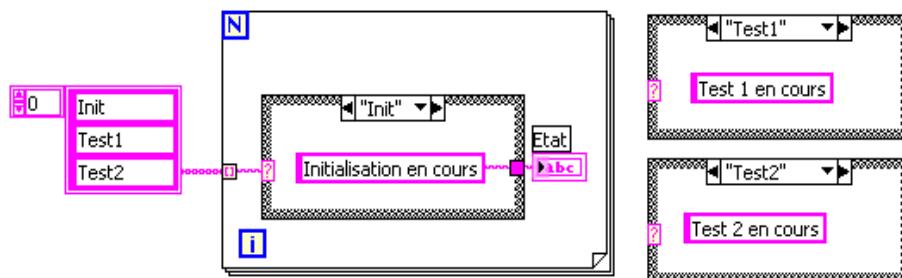
Il est souvent intéressant d'utiliser le parallélisme inhérent à LabVIEW, mais des tâches asynchrones utilisant des ports d'entrée/sortie peuvent parfois entrer en conflit, il faut dans ce cas utiliser soit des séquences, soit, plus rationnellement des mécanismes d'autorisation (jetons, sémaphores).

Autre inconvénient, une donnée doit sortir que d'une seule séquence, puisque toutes les séquences doivent être exécutées avant de sortir de la structure. L'exemple suivant montre l'impossibilité de mettre à jour un simple indicateur.



Nous recommandons l'utilisation d'une structure de type case dans une boucle While ou For, comme dans l'exemple ci-dessous. Notez la possibilité de ne pas exécuter certains cas en modifiant la constante tableau, sans pour autant supprimer le code qui leurs est associé.

C'est ce que l'on appelle une machine d'état

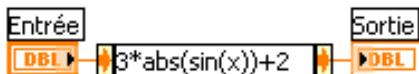


BOITES DE CALCUL ET NŒUDS D'EXPRESSION

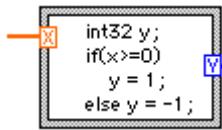
Les boîtes de calcul et les nœuds d'expression permettent d'effectuer des calculs complexes dans un mode texte. Ils rendent le diagramme plus lisible, lorsque les expressions sont complexes.

NŒUDS D'EXPRESSION

Cet outil est très utile pour calculer une valeur fonction d'une seule variable, il se situe dans les **Fonctions/ Numériques**. On y entre une expression mathématique, les opérateurs et leurs priorités sont explicités dans l'aide. X représente dans la formule la grandeur d'entrée. (Attention : pas de ; en fin d'expression).

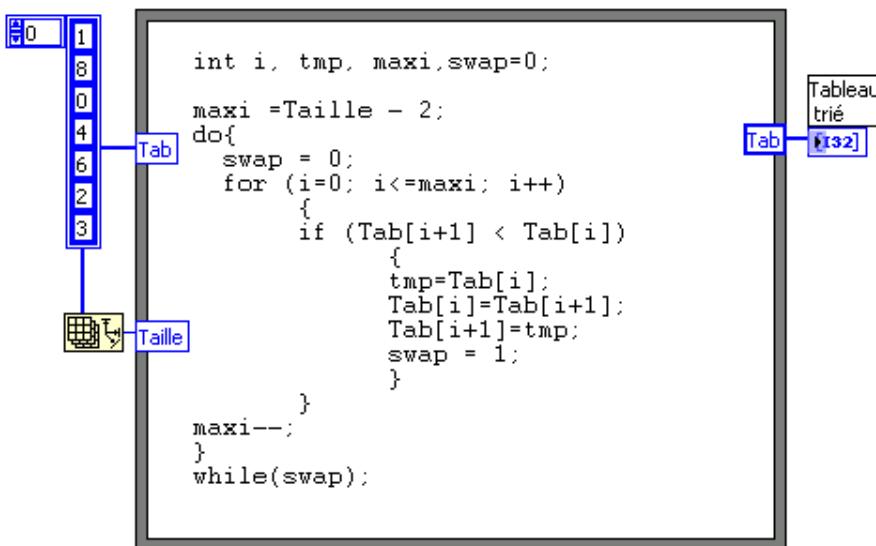


BOITES DE CALCUL

 Elles sont utilisées dans les opérations multi-variables, dès que les expressions sont un peu complexes ou lorsque l'on reprend un code de calcul écrit en C. Elles sont accessibles par la palette **Fonctions/ Structure**.

Le menu contextuel permet de créer des entrées et des sorties qui seront nommées à l'aide de l'outil texte.

Les boîtes de calcul ont une syntaxe comparable à celle du C. On peut y effectuer de multiples opérations, y compris des structures de boucles ou de choix, consultez l'aide pour de plus amples informations. L'exemple suivant vous montre un tri à bulle, comme vous pouvez le constater, le code est exécutable en C.



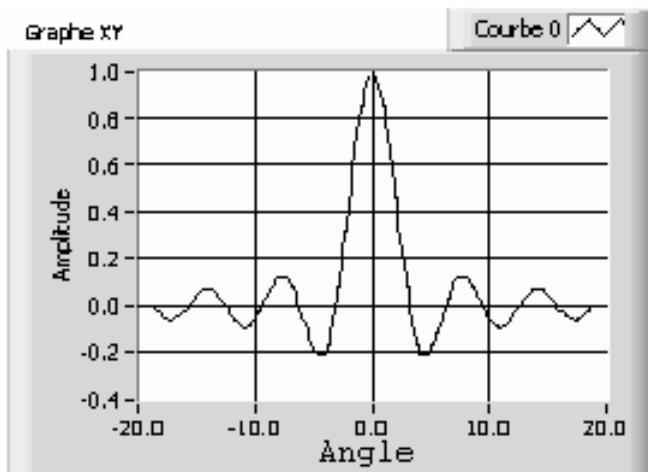
EXERCICE 6-3 : NŒUDS DE CALCUL

OBJECTIF: UTILISER DES FONCTIONS SIMPLES D'UN NŒUD DE CALCUL.

On se propose d'utiliser un nœud de calcul dans une boucle for de 100 pas, pour tracer la courbe $f(x) = \sinus(x)/x$ pour des valeurs d'angle comprises entre -6π et $+6\pi$.

FACE AVANT

La face avant est présentée ci-dessous.



Attention le séparateur décimal doit être un « point »
Démarrer/Paramètre/panneau de config/option régionale/ séparateur

DIAGRAMME

Utilisez une boucle FOR à l'intérieur de laquelle est placée la boîte de calcul. La boîte possède trois terminaux, une entrée, l'indice de boucle, deux sorties, $\sinus(x)/x$, et x .

Attention aux points-virgules en fin de lignes

Vous pouvez déclarer des variables intermédiaires pour clarifier le calcul

Les boîtes de calcul distinguent minuscules et majuscules (syntaxe du C).

FIN DE L'exercice 6-3

7. CHAINES ET ENTREES/SORTIE FICHIER

CHAINES

Les chaînes sont des suites de caractères ASCII, affichables ou non. Elles permettent de transférer facilement des données entre des matériels quelconques. Les fonctions principales dans cette optique sont :

- Créer et manipuler des chaînes.
- Convertir des nombres en chaînes et réciproquement.
- Enregistrer et lire des fichiers ASCII.

CREER DES COMMANDES ET DES INDICATEURS DE TYPE CHAINE



Utiliser la palette **Chaîne&chemins pour** créer des commandes et des indicateurs de type chaîne. La taille du cadre d'affichage est modifiable à l'aide de l'outil flèche. Il est possible d'afficher une barre de défilement pour les textes longs en sélectionnant **Eléments visibles»Barre de défilement.**

Par défaut, le mode d'affichage interprète les codes ASCII et les transforment en caractères. Pour visualiser ou entrer des caractères non imprimables, vous disposez du mode **Affichage des codes ** qui affiche les caractères d'échappement comme en C :

\00 - \FF	Valeur hexadécimal d'un caractère 8 bits; doit être en majuscules
\b	Retour arrière (ASCII BS, équivalent à \08)
\f	Saut de page (ASCII FF, équivalent à \0C)
\n	Retour à la ligne (ASCII LF, équivalent à \0A)
\r	Retour chariot (ASCII CR, équivalent à \0D)
\t	Tabulation (ASCII HT, équivalent à \09)
\s	Espace (équivalent à \20)
\\\	Barre oblique inverse (ASCII \, équivalent à \5C)

valeurs hexadécimales des caractères.

L'Affichage style mot de passe affiche des étoiles à la place des caractères.

TABLES



Les tableaux de chaînes peuvent être créés comme les autres tableaux, cependant, un type spécial de commande existe dans la palette **Liste et Tables**, ces sont les tables. Une table est un tableau 2D de chaînes, auquel il est possible d'ajouter des entêtes de lignes et de colonnes, des barres de défilement, de donner un style de caractères pour chaque case, de changer la largeur d'une colonne ou la hauteur d'une ligne, bref de faire comme dans un tableur. Les doubles lignes de séparation sont obtenues en sélectionnant « En-tête de ligne » et « En-tête de colonne » dans le menu contextuel.

The screenshot shows a software interface with a context menu open on the left and a table on the right.

Context Menu (Eléments visibles):

- Rechercher le terminal
- Changer en commande
- Description et info-bulle..
- Créer

Table:

	T (K)	P (MPa)	Commentaire
Essai 1	10.65	12.4	SUP
Essai 2	14.87	12.5	SUP
Essai 3	54.38	12.5	RAS
Essai 4	77.21	12.4	RAS
Essai 5	96.35	12.5	RAS
Essai 6	45.87	12.4	RAZ

FONCTIONS SUR CHAINE



La palette **Fonctions»Chaîne** contient une grande diversité de fonctions sur chaînes, voici les principales :

MANIPULATION DE CHAINES



Longueur d'une chaîne—renvoie le nombre total de caractères d'une chaîne (y compris les caractères non imprimables).



Concaténer des chaînes—permet de « coller » bout à bout des chaînes. Cette fonction est redimensionnable.



Sous-ensemble d'une chaîne—permet d'extraire une sous chaîne d'une chaîne de départ, en spécifiant le point de départ et le nombre de caractères de la sous chaîne.



Rechercher une expression—recherche un motif dans une chaîne et extrait la chaîne précédant le motif, le motif, la chaîne suivant le motif.



En majuscule/En minuscule—change la casse d'une chaîne.



Constantes de type chaîne— pour créer des constantes, ou obtenir des caractères non imprimables « chaîne vide », « retour chariot CR », « saut de ligne LF », « CR/LF », et « tabulation ».



Note : il existe d'autres fonctions de manipulation de chaînes, accessibles par **fonctions de chaînes supplémentaires** depuis la palette des chaînes.

CHAINES ET NOMBRES

Les fonctions **Balayer une chaîne** et **Formater une chaîne** (équivalentes aux fonctions Fmt et Scan du C) permettent d'éclater une chaîne en éléments séparés ou de composer une chaîne à partir d'éléments séparés.

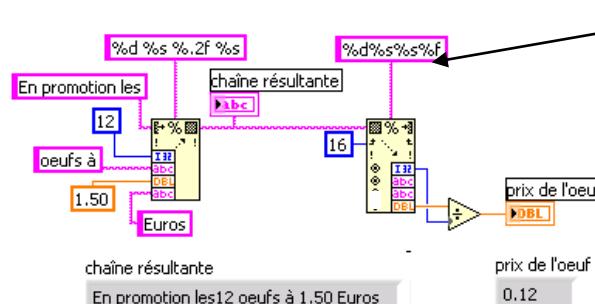


Formater une chaîne : convertit des arguments de tous types en une chaîne.



Balayer une chaîne : extrait d'une chaîne des variables de tous types.

Ces deux fonctions utilisent une chaîne de format comme les fonctions Scan et Fmt du C pour spécifier les types de données à convertir, leur longueur ... Consultez l'aide en ligne LabVIEW pour de plus amples renseignements. L'exemple ci-dessous illustre une utilisation de ces fonctions. Le nombre d'arguments de ces fonctions est modifiable en sélectionnant **Ajouter/Supprimer un paramètre** dans le menu contextuel.



Chaîne de format correspondant aux éléments mis bout à bout

Espace dans la chaîne de format délimite la chaîne lors de la relecture

Pour des conversions moins sophistiquées, vous disposez d'une palette d'outils dans **Chaîne»Conversion chaînes/nombres** qui ne nécessitent pas de chaînes de format.

EXERCICE 7-1 : EXTRACTION DE DONNEES

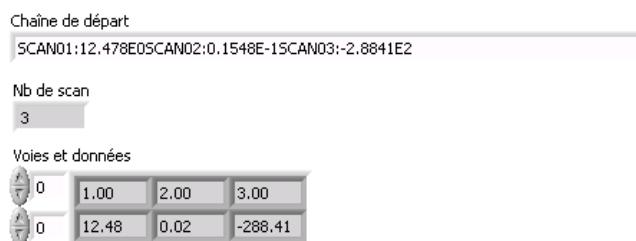
OBJECTIF: UTILISER QUELQUES FONCTIONS DE MANIPULATION DE CHAINES.

Une centrale d'acquisition automatique renvoie des données au format suivant :

SCANxx:yy.yyyEy SCANxx:yy.yyyEy SCANxx:yy.yyyEy etc. où xx est un entier correspondant au numéro de la voie, et yy.yyyEy, un nombre en virgule flottante représentant la valeur lue sur la voie correspondante. On se propose de réaliser un VI capable d'extraire les couples Voie/Valeur dans un tableau 2D.

FACE AVANT

Elle se présente comme suit :



DIAGRAMME

Le VI utilise une boucle WHILE et un registre à décalage. Il recherche la séquence « SCAN » pour extraire ensuite le numéro de la voie et la valeur. Les fonctions de manipulation de chaîne utilisées sont :

 **Rechercher une expression**, cette fonction recherche une expression dans une chaîne de caractère, elle renvoie la sous chaîne avant l'expression, l'expression, la sous chaîne après l'expression et l'offset pour lequel la correspondance est trouvée. Si aucune correspondance n'est trouvée, elle renvoie -1 dans l'offset.

 **Balayer une chaîne** vous permettra de convertir la séquence xx:yy.yyyEy en deux nombres. (Cf. la fin de l'aide en ligne de la fonction : exemples de balayage de chaînes)

Enregistrez votre VI sous le nom `Extraitvaleurs.vi`, entrez une chaîne au format souhaité et exécutez votre VI.

FIN DE L'exercice 7-1

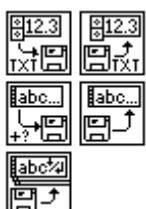
ENTREES/SORTIES SUR FICHIER

Les fonctions **E/S sur fichier** permettent de gérer toutes les opérations sur fichiers, notamment :



- Ouverture/Fermeture de fichiers
- Lecture/Ecriture
- Lecture/Ecriture au format tableur
- Déplacement, effacement, création de répertoires..
- Changement d'attributs
- Gestion de fichiers de configuration (.ini)

FONCTIONS DE HAUT NIVEAU



Les fonctions de haut niveau (icônes blanches en haut à gauche de la palette) permettent la lecture et écriture directe de fichiers textes, ou de fichiers tableurs (tableaux 2D avec séparateur). Elles intègrent l'ouverture/fermeture des fichiers, les boîtes de dialogues, la conversion des données... Ces fonctions sont à utiliser en priorité dans tous les cas simples.

FONCTIONS DE BAS NIVEAU



Ces fonction sont situées dans la sous palette **Fonction de fichiers avancée**, elle permettent l'ouverture/fermeture, l'accès séquentiel, la gestion des droits, la manipulation de répertoires, de noms... Ces fonctions ne seront pas étudiées dans ce cours.

FONCTIONS DE BASE

Une opération standard sur fichier opère ainsi :



- Création ou ouverture d'un fichier.
- Lecture ou écriture des données (éventuellement formatage).
- Fermeture du fichier.



Suivants permettent ces opérations :



Ouvrir/créer/Remplacer un fichier—renvoie un « Refnum » (un handler en C) qui identifie le canal de communication vers le fichier. Si le nom de fichier n'est pas spécifié dans les entrées du VI, une boîte de dialogue Windows standard s'ouvre.



Lire un fichier—extrait un nombre de caractères d'un fichier, identifié par son Refnum à partir d'un emplacement spécifié.



Ecrire dans un fichier—envoie une suite de caractères dans un fichier à l'emplacement spécifié. Ces deux fonctions travaillent sur des chaînes (ou des binaires après un changement de type)



Balayer à partir d'un fichier et formater dans un fichier, permettent d'effectuer les opérations de lecture/écriture en même temps que la transformation des données.



Fermer un fichier—ferme le fichier spécifié par Refnum.

GESTION DES ERREURS

La gestion des fichiers est l'occasion de parler de la gestion des erreurs, car ce type d'opération en provoque souvent (média absent, disque plein, fichier en lecture seule...). La majeure partie des fonctions avancées de LabVIEW fait circuler une connexion d'erreur. Cette connexion est un cluster qui comporte :

sortie d'erreur



Un **état** (booléen) indiquant si une erreur a eu lieu.

Un **code** (int32) qui identifie le type d'erreur (codes d'erreurs complets dans l'aide), si code est différent de zéro et que l'état est faux, il s'agit d'un avertissement.

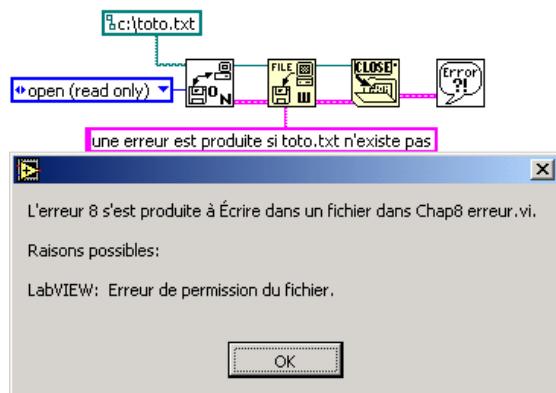
Une **source** (string) qui indique le VI responsable de l'erreur.

Les VIs recevant une erreur n'effectuent aucune opération, il est donc important de prévoir au minimum l'affichage du cluster d'erreur, car on peut croire que l'ensemble des opérations s'est normalement déroulé alors qu'aucune n'a été effectuée.



Utilisez le **gestionnaire d'erreurs simples** de la palette **Temps&Dialogue** pour afficher un panneau signalant l'erreur.

L'exemple suivant illustre ce propos, `toto.txt` est ouvert en mode lecture seule. La fonction **Ecrire dans un fichier** génère donc une erreur. Notez que si l'on ne pose pas le gestionnaire d'erreur sur le diagramme rien ne permet de penser que `toto.txt` n'a pas été écrit.



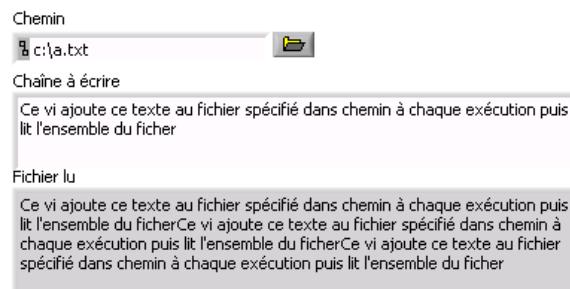
EXERCICE 7-2 : LECTURE/ECRITURE D'UN FICHIER

OBJECTIF: UTILISER LES FONCTIONS DE BASE POUR FAIRE DES E/S SUR FICHIER.

Créez un VI capable d'écrire une chaîne de caractères à la fin d'un fichier à chaque exécution et de lire l'intégralité du fichier avant de le fermer.

FACE AVANT

Voici un exemple de la face avant, après la troisième exécution :



Les commandes de type Chemin d'accès sont dans la palette **Chaînes&chemins**. L'icône à droite du champ ouvre une boîte de dialogue fichier de Windows.

DIAGRAMME

Le diagramme utilise les éléments suivants.



Ouvrir/créer/Remplacer un fichier permet de sélectionner le mode d'ouverture, choisissez **open or create**. Elle renvoie aussi le nombre de caractères contenus dans le fichier ; il vous sera utile lors de la lecture.



Ecrire dans un fichier permet d'écrire une chaîne de caractère à partir de la position spécifiée. Choisissez le « mode pos » adéquat.



Lire un fichier récupère un certain nombre d'octets à partir d'une position donnée.



Longueur d'une chaîne vous permettra de calculer le nombre de caractère à lire dans le fichier, connaissant sa taille à l'ouverture et le nombre de caractères ajoutés.



Fermer un fichier permet de clore la communication avec le périphérique de stockage correspondant.



N'oubliez le **Gestionnaire d'erreurs** pour afficher d'éventuels problèmes d'accès.

Enregistrez votre VI sous le nom demofichier.vi et vérifiez son fonctionnement.

FIN DE L'exercice 7-2

UTILISATION DE VIS DE HAUT NIVEAU

Les VIs de haut niveau effectuent l'ensemble des opérations

Ouverture/Conversion/Lecture&Ecriture/Fermeture avec un seul VI. Il faut les utiliser dès qu'un fichier texte est accédé de façon occasionnelle car la procédure ouverture/fermeture systématiquement effectuée consomme inutilement des ressources système lors d'accès fréquents.

Les VIs de haut niveau sont :



Ecrire dans un fichier tableau—Convertit un tableau 2D ou 1D de réels en une chaîne de caractères. Les données sont séparées par défaut en colonnes par des tabulations et en lignes par des retours chariot. Une chaîne de format peut changer le nombre de chiffres significatifs, la fonction peut transposer le tableau avant de l'enregistrer. Ce type de fichier est un format d'importation standard à tous les tableurs.



Lire dans un fichier tableur —Pour relire le fichier tableur, retourne un tableau 2D. Il est possible de ne lire qu'un certain nombre de lignes à partir d'une ligne donnée.



Ecrire des caractères dans un fichier —Écrit une chaîne dans un nouveau fichier ou l'ajoute à un fichier existant.



Lire des caractères dans un fichier —Lit un certain nombre de caractères dans un fichier à partir d'une position donnée.



Lire des lignes dans un fichier —Identique, mais par ligne entière.



VIls de fichiers binaires —Lecture/Ecriture binaire d'entiers ou de réels simple précision (gain de place important).

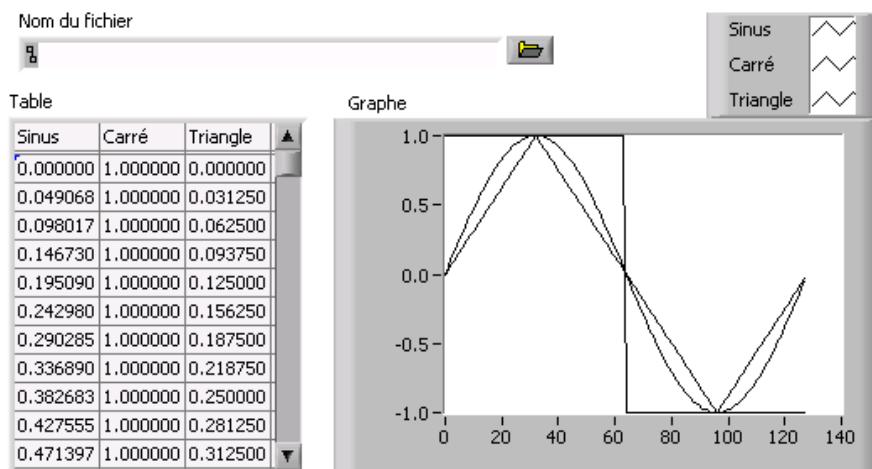
EXERCICE 7-3 : EXEMPLE D'ENREGISTREMENT TABLEUR

OBJECTIF: ENREGISTRER UN TABLEAU 2D DANS UN FICHIER TEXTE TABULE.

Dans cet exercice, vous créez trois formes d'ondes, Sinus, Carré et Triangle (128pts, Amplitude 1, fréquence 7.81^E-3 (1/128)). Ces signaux seront présentés sur un graphe, dans une table puis enregistrés au format tableur dans un fichier.

FACE AVANT

En voici une représentation, l'indicateur à gauche est une table (rappel : disponible dans la palette Liste&tables indicateur de type texte, choisir dans le menu contextuel éléments visibles/entête).



DIAGRAMME

Le diagramme ne contient pas de structure, les signaux seront générés par les fonctions



d'analyse. Vous aurez besoin principalement des éléments suivants :

Signal sinus / Signal carré / Signal triangulaire. Ces VIls génèrent des formes d'ondes dans un tableau, elles sont disponibles dans la palette Analyse»Traitement du signal»Génération de signaux.



Transposer un tableau 2D permet de renverser lignes/colonnes en effet une fois les trois tableaux 1D (sinus, carré, triangle) convertis en un tableau 2D, vous obtenez un tableau 3 lignes, 128 colonnes, et il est plus classique de présenter l'affichage en colonnes.

 Ecrire dans un fichier tableur vous permettra l'enregistrement du tableau 2D. Limiter, par une chaîne de format, l'enregistrement à 2 chiffres après la virgule.

Les outils de conversion nombre -> chaîne sont dans la palette **Chaîne»Conversions chaînes/nombres**.

Enregistrer le VI sous le nom onde.VI et exécuter le. Donner le nom fichier.txt au fichier tableur, puis importer le dans Excel.

FIN DE L'exercice 7-3

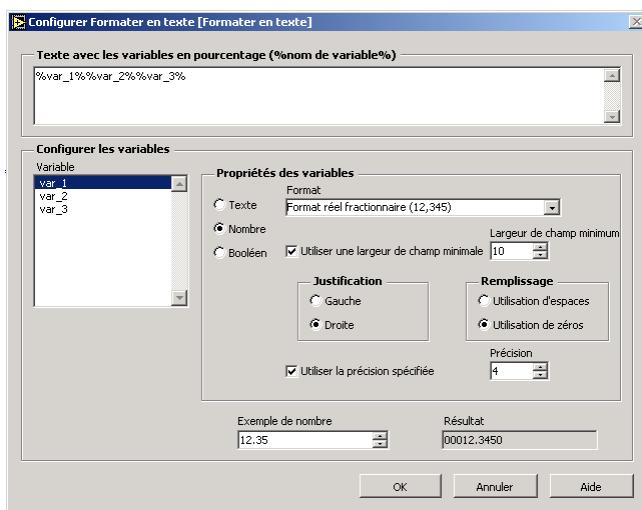


Truc Pour rajouter une entête à un tableau 2D, obtenu par un VI write to spread sheet, on peut créer une chaîne avec des tabulations (\t) via une écriture dans un fichier standard (VI : Write File) .

LES VI EXPRESS

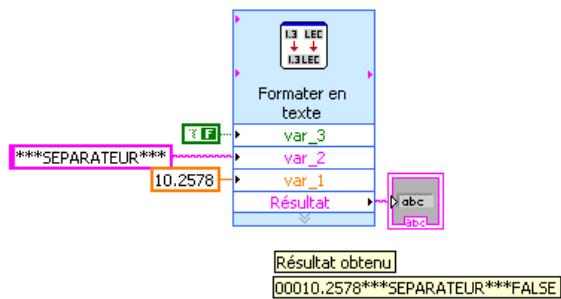


Depuis la version 7, LabVIEW s'est enrichie du concept de VI express. Les VIs express sont des VIs de très haut niveau paramétrables par l'utilisateur. Ils sont regroupés dans la palette **Express**, et se retrouvent aussi dans diverses palettes. Lorsque l'on pose un VI express sur le diagramme, on peut le configurer par un double clic avec l'outil flèche. La fenêtre suivante est le panneau de configuration du VI « Formater en texte »



Vous pouvez constater la polyvalence de l'outil, puisqu'il est possible de changer le nombre de variables d'entrées, leur type, de créer des justifications très élaborées...

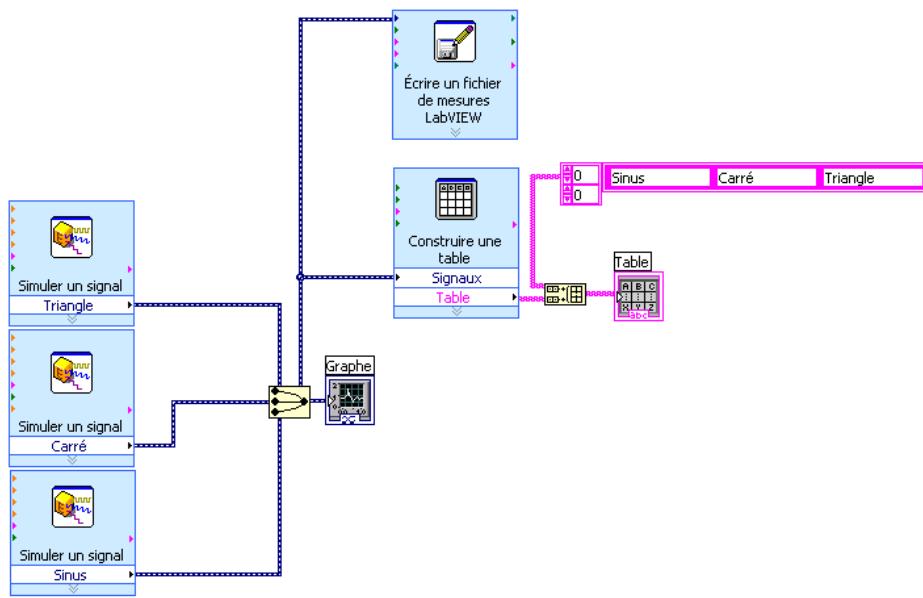
Voici le diagramme utilisant la configuration ci-dessus :



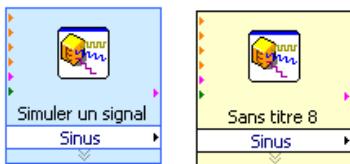
Certains VI express sont posés sur le diagramme en même temps qu'un objet de la face avant ; tel est le cas des **tables express** et des **graphes express**. La figure ci-dessous représente le diagramme d'un VI contenant une table express lisible sur sa face avant.



Voici une version « Express » de l'exercice précédent :



Le code gagne en lisibilité, mais ces « macro-outils » ne sont pas toujours très bien adaptés au cas précis qu'il faut traiter. Ceci oblige soit à se contenter de ce que fait le VI express, ou d'en reprendre le code en sélectionnant « Ouvrir la face avant » dans le menu contextuel du VI. Cette opération convertit un VI express en VI standard, ce qui permet d'en modifier le code pour l'adapter au mieux. L'icône passe alors du bleu au jaune, indiquant la conversion



Ces fonctionnalités engendrent malheureusement un code très lourd. Les VI express d'acquisition, de gestion de fichiers, et de traitement du signal sont parmi les plus utiles

EXERCICE 7-4 : EXEMPLE D'ENREGISTREMENT TABLEUR EXPRESS

OBJECTIF: ENREGISTRER UN TABLEAU 2D DANS UN FICHIER TEXTE TABULE.

Reprenez l'exercice précédent pour le transformer en une version express. Explorez, notamment, les fonctionnalités du VI d'enregistrement fichier qui est des plus performant.

FIN DE L'exercice 7-4

8. PROGRAMMATION MULTITHREAD

INTRODUCTION AU MULTITHREADING SOUS LABVIEW.

GENERALITES.

Les systèmes d'exploitation récents sont tous multitâche et multithread. Le **multitâche** est la capacité d'un système à partager le temps CPU entre plusieurs **processus (lourds)** (*process*). Dans ce cas, le système offre :

- Des mécanismes de protection inter processus (protection de la mémoire allouée, des contextes d'exécution...).
- Des mécanismes permettant à des processus concurrents de communiquer (d'abord par le biais de tubes communicants (*pipes*), de signaux... puis des bibliothèques de communication (*Inter-Process Communication ou IPC*)).

La gestion de communication inter process n'est donc pas chose facile et nécessite l'appel aux fonctions du système d'exploitation, ou des appels à une interface de programmation (**API** ou **Application Programming Interface**), fournie par le concepteur de l'application. Ces appels sont coûteux en temps car le changement de contexte est une opération lente. Une application réactive ne devrait pas faire appel à plusieurs processus mono programmé, mais un seul processus multiprogrammé contenant plusieurs chemins d'exécution. Ces chemins d'exécution sont nommés processus légers (*thread*).

Les **processus légers** (*thread*) sont similaires aux processus (*process*), ils représentent tous deux l'exécution d'un ensemble d'instructions. Ces exécutions semblent se dérouler simultanément pour l'utilisateur. Toutefois, là où chaque processus possède son propre espace mémoire alloué par le système d'exploitation et des mécanismes de communication interprocessus fiables, les processus légers appartenant au **même processus père** partagent une même partie de sa mémoire. Le qualificatif "léger" est donné car, comparativement aux processus lourds, ils ne consomment que très peu de ressources (quelques Kilo-octets pour la pile et le jeu de registres). Les autres ressources sont partagées entre tous les processus légers s'exécutant dans le même processus lourd.

Le multithread est la capacité de l'OS de gérer plusieurs chemins d'exécution au sein d'un même processus (de fait l'OS lance plusieurs routines du même processus simultanément).

LabVIEW possède depuis sa création un séquenceur de processus légers coopératif où chaque processus légers doit explicitement permettre à un autre processus légers de s'exécuter (par opposition aux séquenceurs de processus systèmes communément préemptifs). De fait il était, vu du système, comme mono thread. Depuis la version 7.0 il demande des threads au système. Le séquenceur de LabView reste actif et les threads du séquenceur interne sont regroupés par priorité d'exécution dans des threads systèmes. Sur les system multi cœurs, il est possible d'affecter les cœurs à certain threads

AVANTAGES

MEILLEURE UTILISATION DU CPU

Les processus légers ont été surtout conçus pour faciliter la programmation parallèle. Dans la grande majorité des applications LabView il existe des tâches d'acquisition peu coûteuses en ressource CPU mais lentes. Dans ce cas le programme reste bloqué par cette tâche et ne peut effectuer autre chose. Une approche multithread permet de garder actives d'autres parties du code (calcul, affichage, communications ...)

FACILITE DE PROGRAMMATION

Les programmes dont la structure implique une certaine concurrence entre les parties du code sont adaptés pour les processus légers. L'utilisation des processus légers facilite l'écriture d'un programme complexe où chaque type de traitement est réalisé par un fil d'exécution spécifique. Ainsi, chaque fil d'exécution a une tâche simple à réaliser.

MEILLEURE CONFIANCE DANS LE CODE

Le fait d'avoir plusieurs VI en exécution permet de gérer plus facilement des événements critiques. Un VI peut par exemple scruter en permanence des éléments de sécurité et prendre la main dans une situation d'urgence en asynchronisme complet avec les autres tâches.

AMELIORATION DES PERFORMANCES EN MULTIPROCESSSEURS

Les applications multithread qui s'exécutent sur une machine multi processeurs ou sur des processeurs multi coeurs (Intel & AMD) bénéficient du gain de performance de l'ordre de 30 à 50 % par processeur, chaque thread pouvant être traité par un processeur.

DEPORT DE THREADS SUR D'AUTRES MACHINES

Dans le même esprit, il est possible si une application est conçue multithread, de déporter l'exécution de threads sur d'autres machines, pour bénéficier d'une puissance de calcul supplémentaire, d'un OS temps réel, pour déporter l'acquisition du contrôle...

CONTRAINTE

COMMUNICATIONS INTER THREADS

Du fait que les threads s'exécutent de manière asynchrones, voir sur des processeurs distincts, ou sur des machines distantes, il est impossible de câbler des communications 'au sens LabVIEW de flux de données' entre eux. Il faut donc recourir à d'autres types de communications. Les processus légers d'un même contexte communiquent au travers d'espaces mémoire partagés (variables globales/ partagées). Pour les processus d'un contexte différent (répartis entre plusieurs processeurs, ou sur des machines distantes) il est possible d'utiliser, des messages, des files d'attentes, des variables partagées, les datasockets, et les communications TCP/UDP

SYNCHRONISATION DE THREADS

Comme rien n'est jamais parfait, il est souvent nécessaire de synchroniser des tâches asynchrones ! En effet, une séquence d'initialisation, de mesure, une fin de programme doivent souvent soit commencer dans un certain ordre, ou au même moment ; soit se terminer lorsque tous les acteurs ont finis. C'est la synchronisation des threads. Dans la mesure du possible il faut éviter, par le biais du câblage des VI de lancer des exécutions parallèles non concurrentes.

GESTION DE RESSOURCES COMMUNES

Tous les threads d'un processus partagent ses ressources : fichiers, variables statiques, mémoire tas (*heap*). La cohérence des données ou des ressources partagées entre les processus légers doit être maintenue. Il faut gérer l'accès exclusif d'un thread à ces ressources partagées et il est préférable d'en avoir le moins possible.

L'utilisation de *sémaforos* (compteur d'accès) ou de *mutex* (verrou d'exclusion mutuelle) permet de gérer ces partages.

CORRUPTION MEMOIRE ALEATOIRE

En anglais '*race condition*' se produit lorsque qu'un thread lit en même temps qu'un autre écrit une variable, il faut gérer l'accès exclusif. Il semblerait que les variables partagées intègrent un mécanisme de protection de fait de leur diffusion par le canal TCP. Le caractère aléatoire des erreurs entraîne des difficultés de débogage énormes, le programme peut parfois tourner des heures sans soucis.

PARTAGE DE RESSOURCES PHYSIQUES

Plus intuitif, par exemple l'accès à un fichier, un appareil de mesure, une imprimante. Les mécanismes de gestion sont identiques aux gestions de corruption mémoire. Il existe cependant dans la classe VISA un certain nombre d'outils spécifiques à la réservation de ressources matérielles.

DEADLOCK

Un *deadlock* est un blocage du système dû à deux réservations exclusives de ressources. Un thread attend que l'autre libère une ressource pour continuer et vice versa. Cette situation peut arriver fréquemment (notamment dans les transactions réseau). Il existe des algorithmes de réservation qui peuvent prévenir ou détecter des *deadlock*.

PRIORITES D'EXECUTION

Par défaut tous les threads ont la même priorité d'exécution ; il est possible d'en changer par deux moyens :

CHANGEMENT DE PRIORITE

Dans les propriétés du VI on trouvera 6 niveaux de priorité d'exécution, de **background** (la plus faible) à **subroutine** (la plus forte), les VIs de plus forte priorité sont placés en tête de la file d'attente d'exécution, le séquenceur d'exécution bascule le CPU entre tous les VIs ayant la plus haute priorité, tous les autres sont en attente !

Deux problèmes peuvent se poser :

Si un thread de priorité haute est bloqué (boucle infinie, attente d'un événement extérieur) tout le système est bloqué !

Famine (*starvation*) : un thread, prêt à être exécuté, est toujours devancé par un autre plus prioritaire.

ATTENTE DE LIBERATION

Le séquenceur de LabVIEW étant coopératif, si deux tâches ont la même priorité elles ont le même temps processeur. Une attente (icône Wait) dans le VI le moins 'utile' enlève momentanément le VI de la liste d'attente et libère du temps processeur pour les autres threads. C'est un moyen 'moins beau' mais plus sûr que

de changer les priorités d'exécutions. Une attente de quelques dizaines de ms dans une interface utilisateur passe inaperçue et libère du temps machine pour des tâches plus importantes.

NOEUDS SYNCHRONE

Un certain nombre de nœuds sont synchrones, c'est-à-dire qu'ils ne font pas de multitâches avec les autres nœuds.

Il s'agit principalement des nœuds de propriété et de méthode qui sont synchrones avec le thread d'affichage ce qui peut entraîner des ralentissements très importants si par exemple on utilise des propriétés (value) pour changer la valeur d'indicateurs alors que l'affichage traite de grandes quantités de données

Les nœuds de code (appels de DLL externes) s'exécutent également de façon synchrone. Il faut les configurer comme non réentrants pour être certain qu'un seul thread exécute le code de la DLL à la fois.

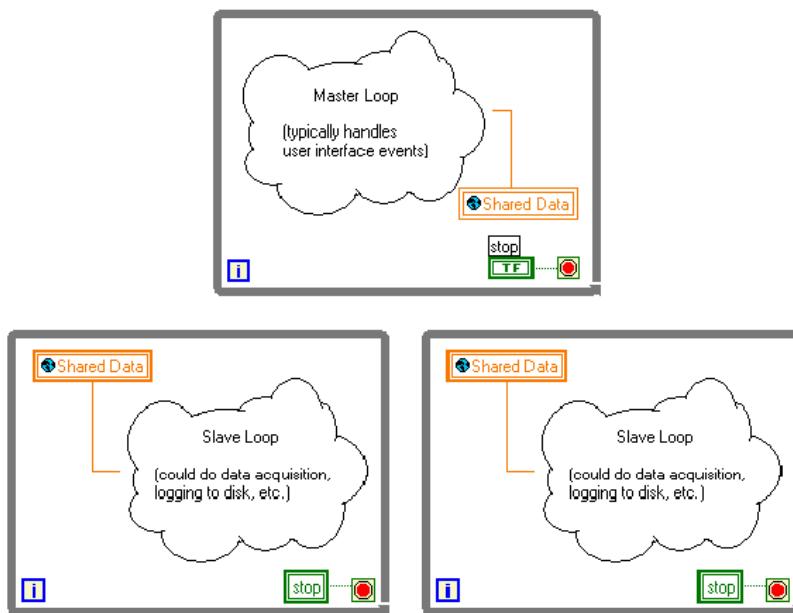
MODELE DE PROGRAMMATION

Il existe quelques modèles de programmation particulièrement adaptés aux processus léger.

MODELE MAITRE/ESCLAVES (MASTER/SLAVES)

Dans le modèle maître/esclave, un processus léger répartiteur (le maître) reçoit des ordres ou des requêtes (de l'interface utilisateur, d'éléments extérieurs, du système...) et les distribuent en fonction de leur type à des processus esclaves qui les exécutent. Les requêtes peuvent être distribuées par des files d'attente, ou bien le maître peut attendre la fin de traitement d'un esclave avant de lui réaffecter une mission. Les esclaves peuvent être créés dynamiquement lors de la réception de la requête, ou au départ de l'application (un par type de requêtes)

Le modèle maître/esclave est particulièrement adapté aux serveurs de requêtes (SQL, HTTP, FTP), et aux gestionnaires de fenêtres (interface utilisateur).



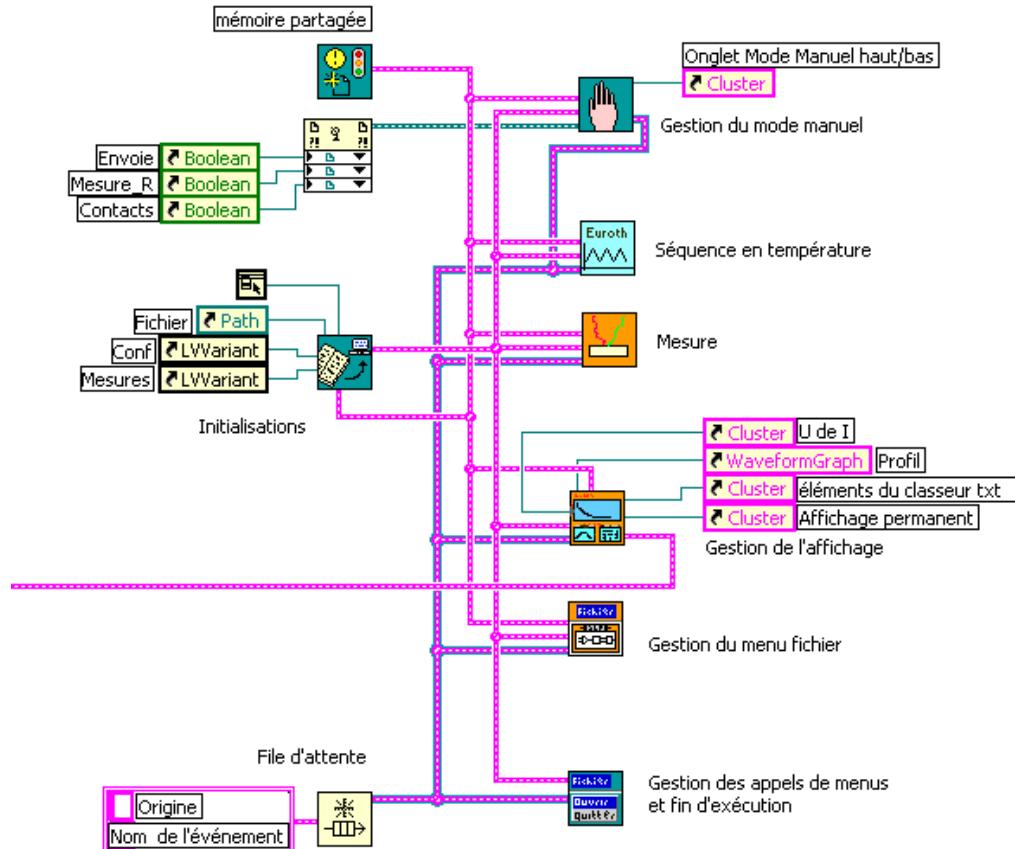
MODELE PARALLELE

Dans le modèle parallèle, chaque processus léger réalise les traitements de façon concurrents sans être commandé par un maître. Chaque processus léger reconnaît et traite ses propres requêtes.

Ce modèle convient aux applications ayant des traitements prédéfinis à réaliser (génération d'une rampe de température, un balayage en fréquence...).

EXEMPLE : TRAITEMENT PARALLELE DE DONNEES CONCURRENTES

Une application peut souvent ressembler à ceci, un VI d'initialisation, un ou plusieurs VI d'acquisition, un ou plusieurs VI d'excitation, des VI de gestion des menus et de l'affichage, d'enregistrement, il pourrait y avoir une communication vers un superviseur distant... Tous ces VI font des opérations concurrentes et communiquent entre eux.



On remarque une file d'attente reliée à tous les VIs et un sémaphore de protection mémoire.

TRAITEMENT SERIE PAR LOT D'UN FLOT DE DONNEES (PIPELINE)

Dans ce modèle, l'exécution d'une requête est réalisée par plusieurs processus légers exécutant une partie de la requête en série. Les traitements sont effectués par étape sur une partie des données.

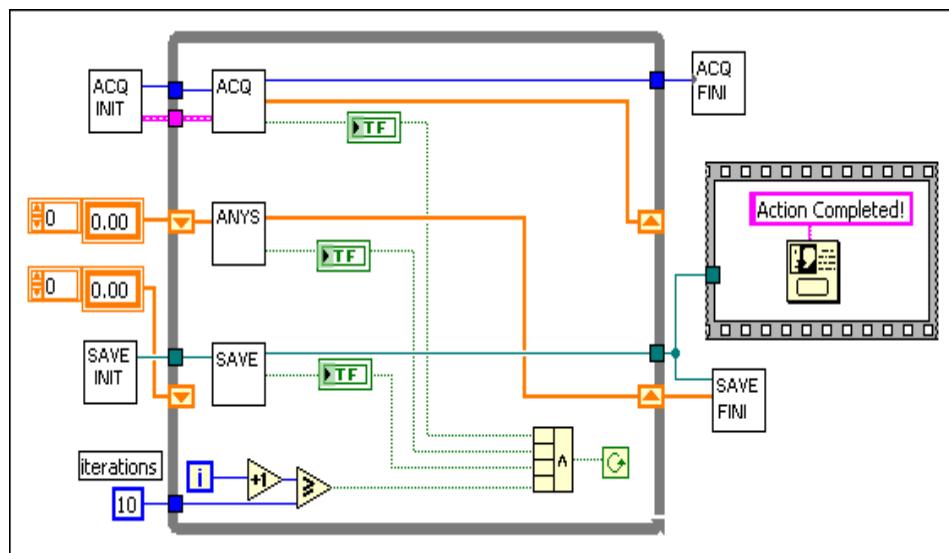
EXEMPLE : ACQUISITION-TRAITEMENT-ENREGISTREMENT

Etudions une tâche d'acquisition classique, ACQUISITION-TRAITEMENT-ENREGISTREMENT. Dans une approche classique, les icônes seraient exécutées séquentiellement.

Pour obtenir des chemins d'exécution parallèles il faut évidemment que les VI ne se passent pas directement des données, un nœud ne pouvant être exécuté que lorsque toutes ses entrées ont été évaluées. Dans l'exemple suivant le VI 'ACQ' **ne peut** commencer son exécution **avant** que le VI 'ACQ INIT' ne soit exécuté.

Cependant, si l'on suppose que 'ACQ' prend à chaque tour de boucle quelques données brutes, que 'ANYS' soit capable de traiter le flot, paquets par paquets, et que, 'SAVE' puisse enregistrer les données traitées par lot, les

trois VI tourneront en parallèle dans le thread d'exécution. Une telle approche peut augmenter la performance de l'application, notamment si la tâche d'acquisition ne consomme pas de ressource système.



GESTION DES APPLICATIONS MULTITHREADS

GESTION DES COMMUNICATIONS INTER THREAD

Il s'agit d'énumérer les méthodes permettant d'envoyer des données à des VI interconnectés uniquement par des fils d'entrée, comme ceux de l'exemple précédent.

Deux méthodes cohabitent, il est possible de partager des zones mémoires ou d'envoyer des messages

MECANISMES DE PARTAGE DE VARIABLES

Deux mécanismes permettent d'accéder à une ressource mémoire commune

La variable globale/partagée

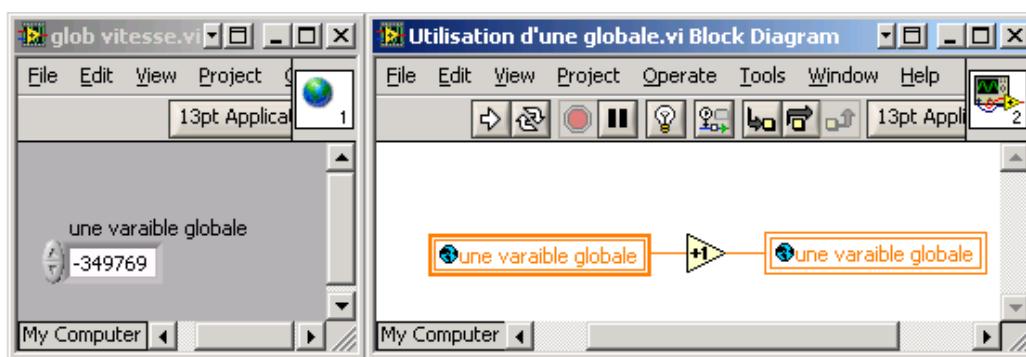
La transmission d'une référence d'objet de la face avant

VARIABLES GLOBALES/PARTAGEES : GENERALITES

Les variables partagées sont des zones mémoires connues de tous les threads d'une application. Comme toutes les variables globales (en C) elles doivent être en nombre strictement suffisant, pour limiter l'espace mémoire occupé et pour la lisibilité du code. Bien qu'ayant fait l'objet de grandes améliorations au fil des versions de LabVIEW, leur niveau de performance reste environ dix fois plus faible qu'un registre à décalage en termes de vitesse de lecture/écriture, si l'on ne gère pas la protection contre les corruptions. Si le code est écrit proprement cette vitesse baisse encore d'un facteur 10 au minimum.

VARIABLES GLOBALES GLOB

Les variables globales sont des variables contenues dans une face avant d'un VI spécial ne contenant pas de diagramme. Le VI est chargé en mémoire et l'ensemble des VI y faisant référence peut y écrire ou y lire. Il est préférable de regrouper les variables relevant un intérêt commun dans un seul VI pour améliorer la vitesse d'exécution. On les trouve dans la palette de structure. Ces variables ne font pas l'objet de protection de corruption mémoire !



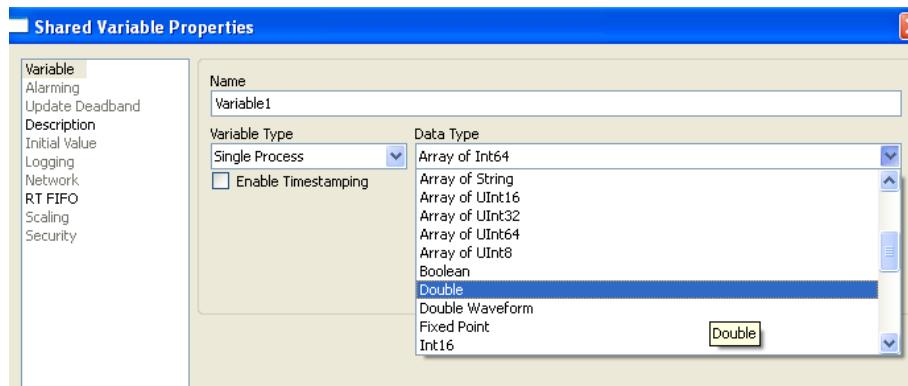
L'opération d'écriture ou de lecture est sélectionnée par le menu contextuel de la variable. Si plusieurs variables globales sont regroupées sur la même face avant, il est possible de les sélectionner à l'aide de l'outil doigt, ou du menu contextuel de la variable.

VARIABLES PARTAGEES VAR

Les variables partagées étendent la notion de global au réseau grâce à un moteur de publication appelé NI Publish-Subscribe Protocol (NI-PSP). Il semble, du fait du mécanisme de publication, qu'elles soient protégées des corruptions mémoires. Elles sont déclarées dans le **gestionnaire de projets** par « New>Variable ». Une variable partagée appartient nécessairement à une **bibliothèque LabVIEW**.

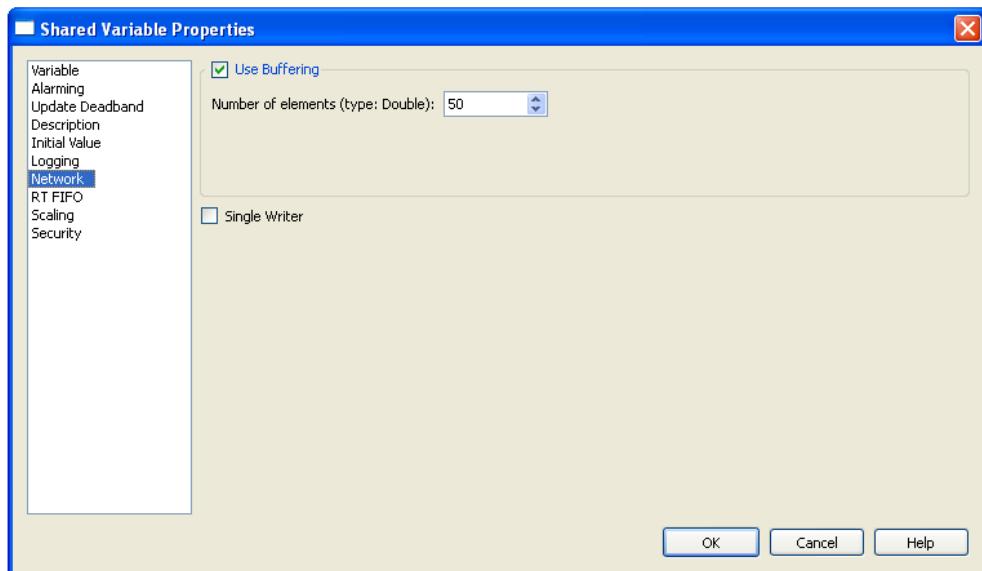
Publiée par

La fenêtre de propriété de la variable permet de sélectionner le type de donnée (entier, réel...). Le type de variable, publiée sur le réseau, ou local à la machine. En local il est possible d'activer une fonction d'horodatage qui enregistre la dernière date de publication de la variable, cette fonction est nécessairement activée en cas de publication.

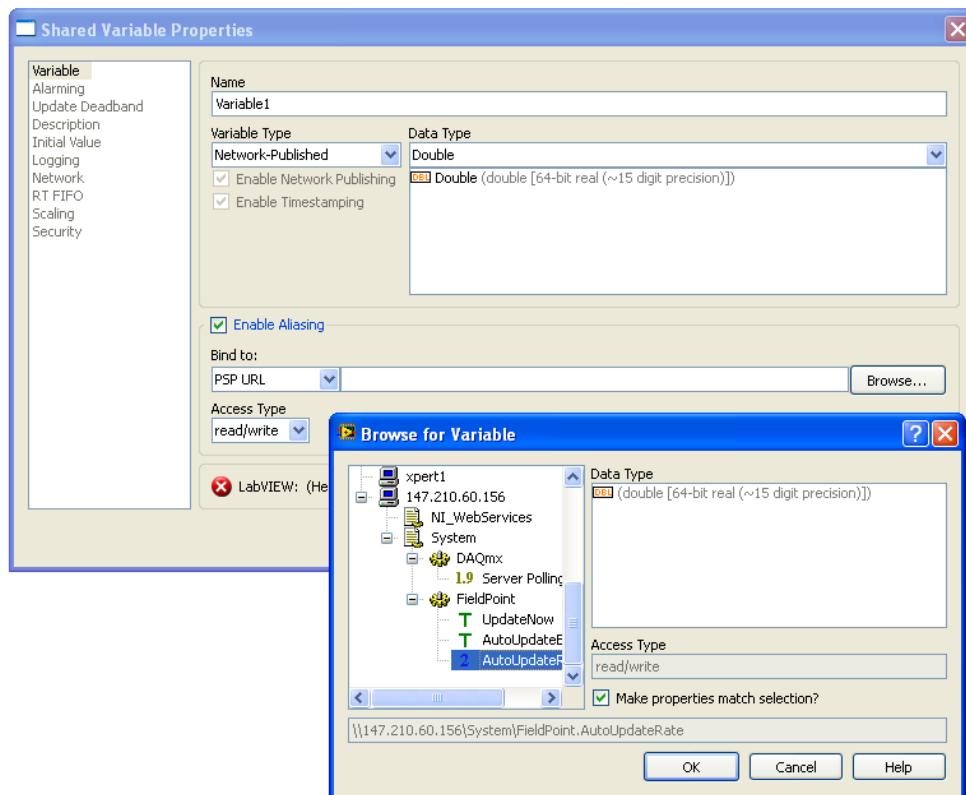


La publication réseau permet, au détriment de la vitesse l'accès à d'autres fonctionnalités :

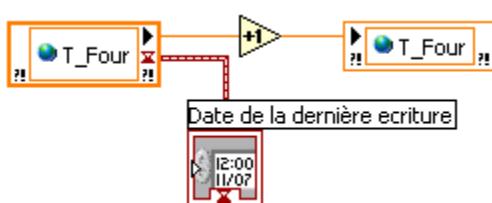
La bufférisation : Vous permet de stocker les données de la variable dans le tampon FIFO (First-In-First-Out). Dans une application temps réel cela permet, par exemple, de maintenir le déterminisme des sections critiques lors de transfert de données. Plus classiquement cela permet d'accepter des variations momentanées de vitesses de lecture/écriture (à condition de ne pas saturer la FIFO).



Lier à la source : Lie (*Bind*) la variable partagée à une variable partagée existante localement, ou à une variable déjà publiée par une autre machine. En clair cela revient à écrire `a := b`, mais avec une variable `b` présente sur machine distante.



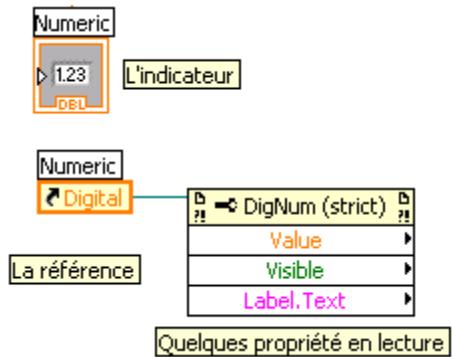
La variable est placée sur le diagramme soit à partir de la palette structure, soit par un déplacé lâché depuis le gestionnaire de projet. Si cette variable est publiée sur le réseau elle peut être directement attachée à un contrôle sur une face avant distante (attention aux configurations des passerelles). Dans le cas de flots de données très importants entre des threads distants (vidéo) il est préférable, moyennant une programmation plus complexe d'utiliser des canaux TCP ou UDP propres à ces échanges. Quoiqu'il en soit n'espérez aucun déterminisme temporel sur une liaison LAN surtout si elle est partagée !



L'utilisation ne pose pas de problèmes, la vitesse est un peu inférieure, en local, à la vitesse des globales. En distant, cela dépend de beaucoup de la machine et du réseau.

REFERENCES ET PROPRIETES

Tous les objets de la face avant possèdent des propriétés, on y accède via un nœud de propriété situé dans la palette « application control » ; ce nœud reçoit une référence obtenue en déroulant le menu contextuel de l'objet et en sélectionnant « **Create reference** ». Il est possible d'accéder ainsi à toutes les propriétés de l'objet, et notamment à sa valeur.



La référence de l'objet étant statique, il est possible de la transmettre à tous les VI ayant besoin d'accéder à une propriété de l'objet.

Comme indiqué plus haut ce nœud est synchrone au thread d'affichage. Il n'est pas recommandé d'utiliser souvent les nœuds de propriété sauf évidemment pour l'affichage sur des objets de la face avant. La vitesse de lecture/écriture est plusieurs milliers de fois plus faible que pour des variables globales.

EXERCICE:8 1

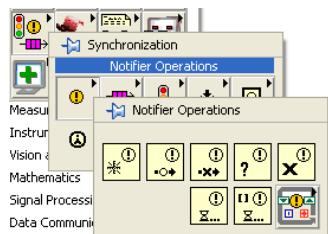
OBJECTIF : COMPARER LES VITESSES D'ACCÈS AUX DIFFÉRENTS TYPES DE VARIABLES PARTAGEES

Créer un Vi capable de compter le nombre d'opérations de Lecture/Ecriture possible durant 10 secondes pour des variables globales, partagées, locales, nœuds de propriété et registres à décalage. Dans un premier temps sur un réel en double précision, puis sur un tableau de 10000 réels. Ceux qui aiment l'aventure peuvent essayer de déployer une variable partagée réseau entre deux machines. Lorsque la première variable est créée, il faut la déployer en sélectionnant l'item « Tout déployer » du menu contextuel de la bibliothèque contenant la variable. Elle est alors disponible pour la liaison au niveau de toutes les autres machines.

MESSAGES

Les messages peuvent servir de communication inter threads, mais il faut les voir plus comme des passages d'ordres (dans une vision maître/esclave) que comme des partages de données. Ils vont servir à assurer le lancement de tâches d'une machine d'état, synchroniser ces tâches, transmettre des requêtes de l'utilisateur (menus).

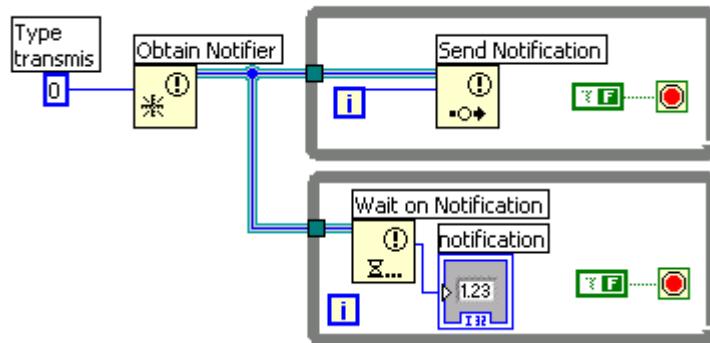
NOTIFICATIONS



Une notification est un 'post it' envoyé par un émetteur pour tous les receveurs. Il reste disponible tant que l'on n'a pas collé un nouveau 'post it' dessus, il n'y a pas de tampon mémoire, il est seulement possible de lire

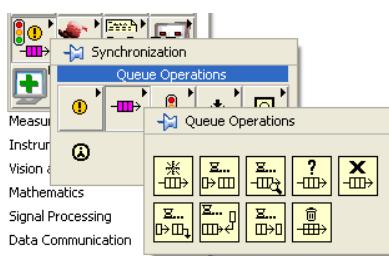
la dernière notification. Les receveurs peuvent rester bloqués en attente d'une notification, ou de plusieurs notifications d'émetteurs différents.

La notification transmet un type de données quelconque, c'est lors de la création de la notification que l'on va typer la donnée transmise. Tous les émetteurs et receveurs ont un fil commun qui référence cette notification. Le diagramme suivant illustre l'utilisation des notifications.

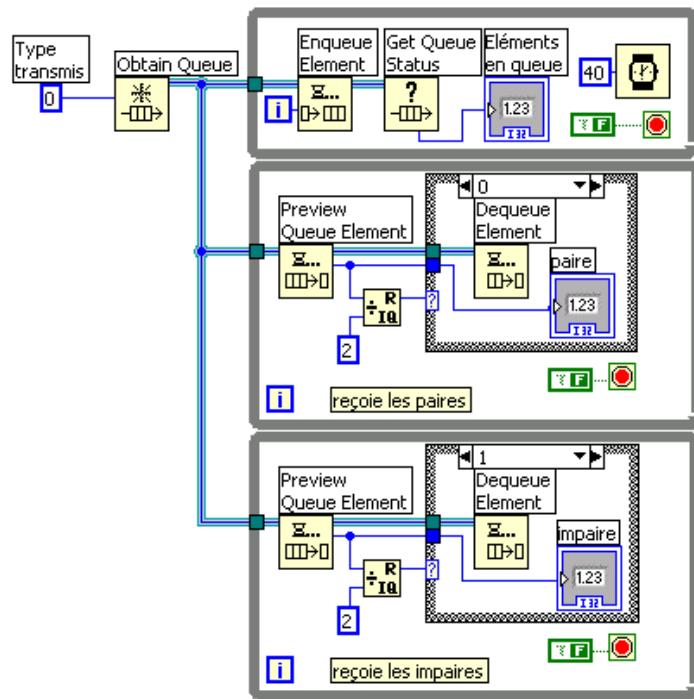


Le débit de données est de quelques Mo/s

FILES D'ATTENTE



Les files d'attente sont identiques aux notifications, elles intègrent un mécanisme de gestion de pile qui permet d'accumuler les messages. Si plusieurs receveurs sont présents il faut prendre garde à ne pas dépiler les messages destinés aux autres receveurs. Il est possible de limiter la taille de la pile, lorsque cette limite est atteinte, le VI « Enqueue Element » attend la libération d'une place pour s'exécuter. La gestion de la pile est très gourmande en ressource. Le diagramme suivant illustre l'utilisation des files d'attentes.



PROJET: PARTIE 1

Utilisez une file d'attente et une notification pour réaliser un VI qui change l'état de deux autres VI nommés A et B, d'un mode ATTENTE à un mode ACQUISITION. Il faudra quatre secondes pour passer d'ATTENTE à ACQUISITION et deux pour revenir en ATTENTE. Ces ordres sont transmis par une file d'attente. Lorsque que les deux VI ont fini de changer d'état, ils envoient une notification au programme principal. S'ils reçoivent l'ordre de passer dans un état dans lequel ils sont déjà, rien ne se passe, aucune notification n'est envoyée.

La face avant pourrait ressembler à :

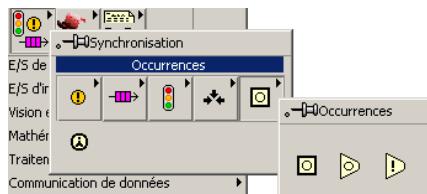


En fonction de la notification reçue, les deux LEDs changent d'état.

SYNCHRONISATION DE THREADS

La synchronisation de threads permet, soit de définir l'ordre d'exécution, soit de démarrer des tâches simultanément, soit d'attendre qu'un certain nombre de tâches soit arrivé dans un état donné.

OCCURRENCES & NOTIFICATION



Les occurrences sont des notifications dont le type est un booléen, je n'encourage pas leur emploi. Les notifications, comme nous venons de le voir permettent d'attendre la réception d'un ordre ce qui va synchroniser le lancement de certaines tâches, cadencer une exécution, définir un ordre...

RENDEZ VOUS



Les VI de rendez vous permettent d'attendre qu'un certain nombre de VI soit dans un « Wait at rendez vous ». Lorsque le nombre de participants requis est atteint, l'exécution continue. Cela permet de se fixer à un état donné de tous les intervenants.

PROJET: PARTIE 2

Rajoutez au menu de l'exercice précédent un item ARRET. Lorsque l'opérateur sélectionne cet item, les VIs A et B finissent leurs tâches (contenues dans la file d'attente) et se donnent rendez vous pour que le plus rapide attende le plus lent pour terminer leurs exécutions simultanément.

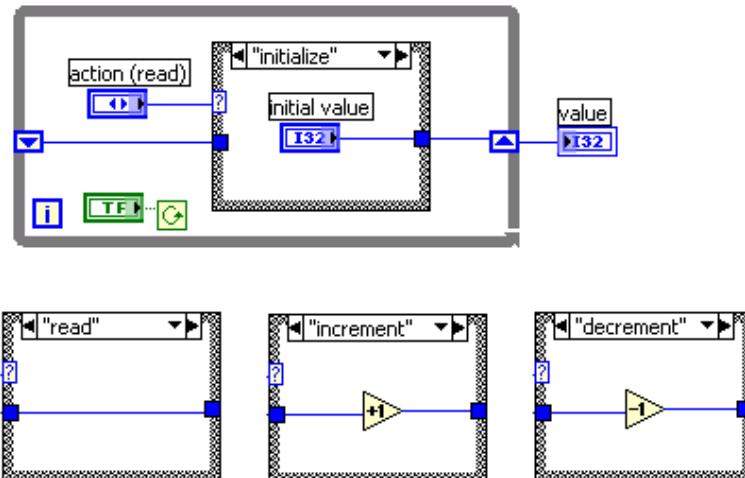
GESTION DE RESSOURCES COMMUNES

La gestion des ressources communes est un point important de la qualité du code. Une corruption mémoire est indétectable et peut entraîner des résultats inattendus

VARIABLES FONCTIONNELLES

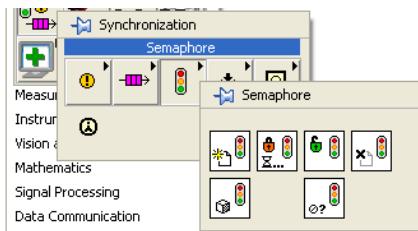
Une variable fonctionnelle est un VI contenant un registre à décalage destiné à stocker la valeur de la variable protégée. Un VI étant par défaut non réentrant, son code ne peut être exécuté par deux threads simultanément, assurant ainsi la protection recherchée.

L'exemple suivant montre une variable fonctionnelle ayant quatre actions possibles, il pourrait bien sûr n'y avoir que des actions lecture/écriture, ou des accès à des ressources physiques...

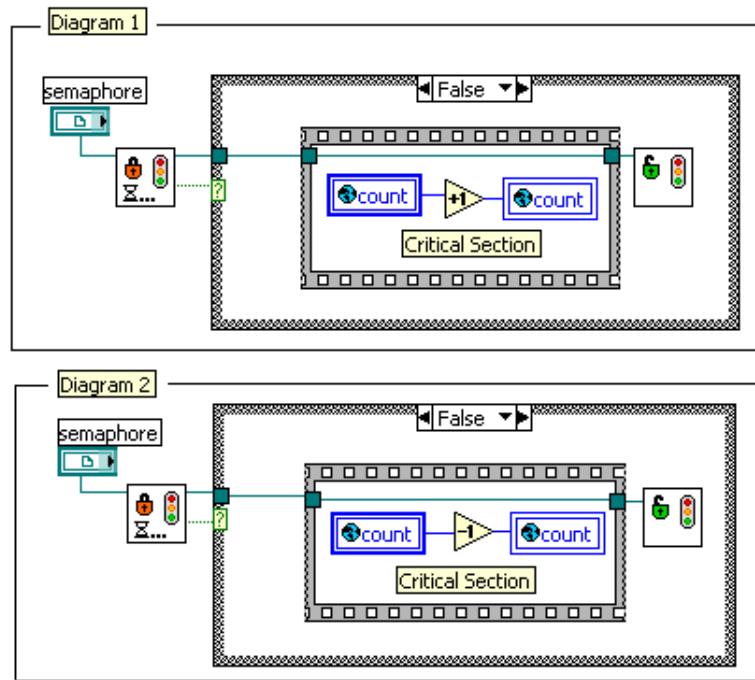


Une variable fonctionnelle est suffisante pour assurer la protection des ressources communes. Il faut toutefois en créer autant qu'il y a de type de données à protéger, et en modifier le code si ce type de données évolue en cours de développement (ça n'arrive jamais, quoique...).

SEMAPHORES



Les sémaphores permettent d'obtenir un accès exclusif à une ressource partagée. Le VI qui bloque le sémaaphore attend qu'il soit libre avant de le bloquer et de passer dans la section critique. La ressource est libérée dès que la section critique est passée. Une icône « **create semaphore** » fournit une référence qui est passée à tous les VI utilisant ce sémaaphore.



Bien entendu ces opérations consomment du temps processeur et le niveau de performance lecture/écriture est largement diminué. Il est important de n'avoir qu'un minimum de variables partagées.

PRIORITE « SUBROUTINE »

Si vous configurez LabVIEW pour ne demander qu'un seul thread système (Tools/Options/Performance & Disk), il est possible d'assurer à un VI un accès exclusif en lui donnant la priorité « Subroutine ».

PROJET: PARTIE 3

Reprendre l'exercice précédent. Les VIs A et B lorsqu'ils sont dans l'état ACQUISITION fournissent des données (aléatoires pour l'instant) à un futur VI d'affichage. Utilisez le VI express « Simulate Arbitrary Signal » pour générer ces signaux points par points. Le cadencement sera dû au time out de 10 ms imposé à l'attente de lecture de la file d'attente. Créez une variable globale contenant deux tableaux A et B pour y stocker ces signaux. L'accès à ces tableaux est protégé par un sémaphore.

PRIORITE ET CADENCEMENT DE THREADS

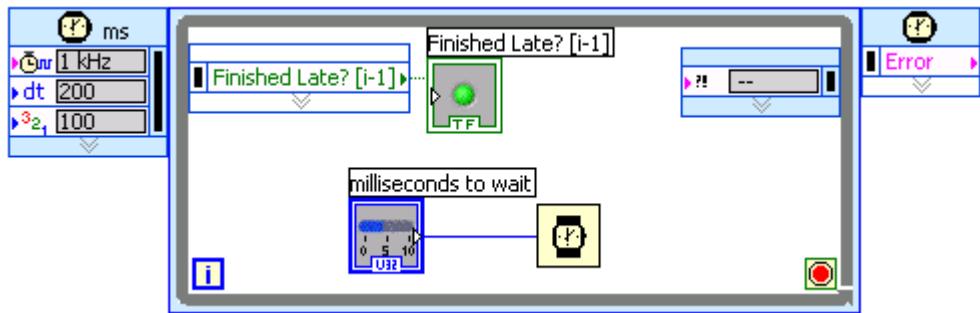
INTRODUCTION

Le séquenceur de LabVIEW étant coopératif, les threads les plus prioritaires ne sont enlevés de la pile d'exécution que s'ils sont mis en attente. Il est donc important de laisser du temps à ceux qui en ont besoin ! Tout vient à point à **qui sait attendre**. Il est préférable d'utiliser des attentes plutôt que de changer les priorités à moins d'être certain de ce qui est fait.

ATTENTES ET BOUCLES TEMPORISEES

Le moyen le plus simple d'attendre est l'utilisation de la fonction « Wait » ou « Wait Until Next ms Multiple ». La première est une attente simple, la seconde attend que le compteur d'horloge passe par un modulo du temps imposé, elle permet un cadencement précis si le temps d'exécution de la tâche est constant.

La meilleure méthode est la boucle cadencée, elle permet un cadencement précis, la synchronisation de boucles (avec une différence de marche si l'on veut). Il est possible de savoir si la boucle a duré plus longtemps que prévu. Dans l'exemple suivant, la LED s'illumine si l'attente est supérieure à 200 ms.



PROJET: PARTIE 4

Reprendre l'exercice précédent. Ajouter un graphe en face avant qui affiche les deux courbes de données des VI a et B. Le VI qui s'acquitte de cette tâche est peu prioritaire. Il est cadencé par une boucle temporisée de 250 ms. Les données sont inscrites dans le graphe par un nœud de propriété « Value », la lecture des tableaux utilise évidemment des sémaphores.

9. CHARGEMENT DYNAMIQUE, « VI SERVER »

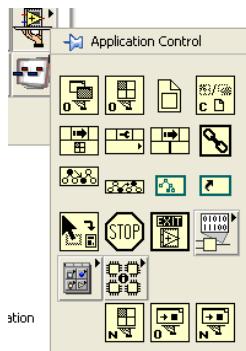
INTRODUCTION

Une application moderne doit être facilement évolutive, il est parfois important de pourvoir rajouter des fonctionnalités lors d'évolution imprévues, sans nécessairement changer le cœur de l'application (les fameux AddOns et autres Plugins). Il est possible aussi de concevoir des applications formées par un module principal (le maître) et des modules secondaires chargés en fonction du type et du nombre de requêtes reçues. Le traitement de grandes quantités de données peut aussi être déporté sur plusieurs machines en fonction de leur charge. Enfin dans le cas de machines très chargées, il est parfois intéressant de ne charger en mémoire de grosses parties de code qu'au moment opportun. Ceci nous conduit à l'étude du chargement dynamique de VIs en mémoire et à leur lancement.

CHARGEMENT DYNAMIQUE DE VI

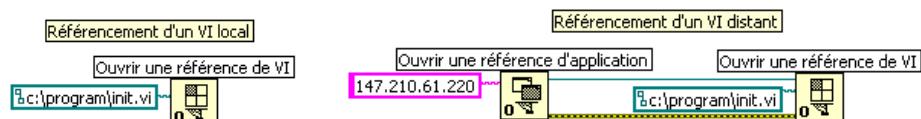
PREAMBULE

Presque tous les VIs utilisés ici sont dans la palette contrôle d'applications



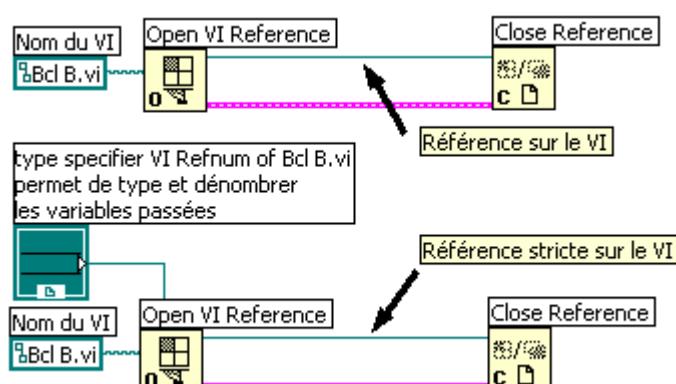
REFERENCEMENT D'UN VI

Le référencement d'un VI sert à créer un pointeur sur la fonction (au sens du C). Ce pointeur peut référencer un VI local (un pointeur sur la fonction de l'application tournant sur la machine locale), soit distant (on contrôle alors l'application sur des machines distantes, via le VI server de LabVIEW). Dans le cas du référencement d'un VI distant, il faut fournir une référence de l'application LabVIEW avant d'ouvrir la référence du VI



CHARGEMENT D'UN VI

Le chargement en mémoire se fait en ouvrant une référence sur un VI. Cette référence peut être strictement typée (on connaît le nombre et le type de chaque variable passée) ou complètement non typée (permet de réaliser un lanceur de VI plus générique). Le déchargement se fait en fermant cette référence. La mémoire est libérée s'il n'y a pas d'autres références à ce VI et si sa face avant est fermée.

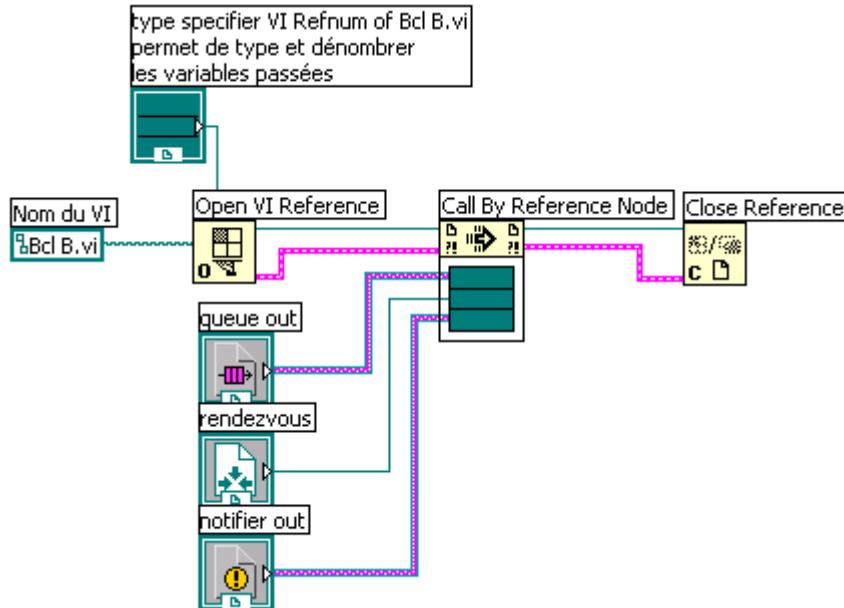


Le référencement strict s'obtient en créant une constante sur le fil « **type specifier** » puis en sélectionnant « **Vi class Server->Browse** ».

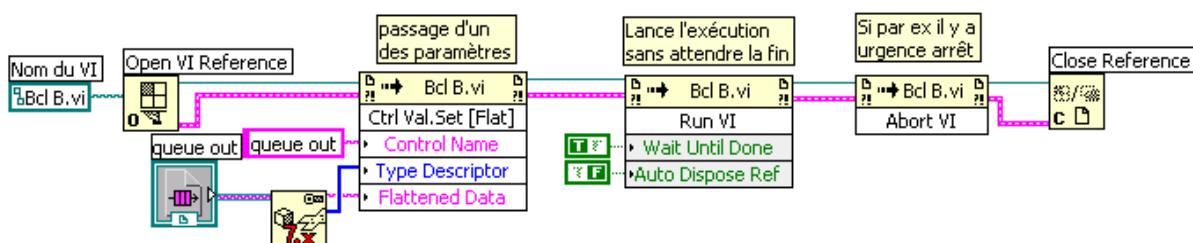
LANCLEMENT ET TRANSMISSION DES PARAMETRES

Une fois chargé en mémoire, le VI doit être exécuté ; deux possibilités existent :

L'exécution du code de la fonction chargé par « Open VI reference » fait appel au VI « Call By Reference Node ». Ce dernier offre un connecteur adapté au type et nombre de données de la fonction. La référence doit donc être stricte. Ce type d'appel est identique à l'appel standard d'un VI, le nœud « Call By Reference Node » se termine lors que le VI appelé a terminé sont exécution.



Il est possible de lancer le VI par un nœud de méthode « Run VI » après avoir éventuellement transmit des paramètres par un nœud de méthode « Ctrl Val.Set ». A la différence du « Call By Reference Node » il est possible de ne pas attendre la fin de l'exécution pour quitter le nœud de méthode « Run VI » ceci permet de lancer plusieurs exécutions, de laisser un VI tourner en tâche de fond (par exemple, réception d'ordres) ou aussi d'arrêter brutalement un VI par un nœud de méthode « Abort » équivalent au bouton STOP (pas très beau, mais parfois bien utile).



Projet : partie 5

Créez un VI capable de charger dynamiquement les VI A et B à l'aide d'un « Call By Reference Node » ou par un nœud de méthode « RUN » lorsque l'utilisateur sélectionne sur le Ring un nouvel item « MARCHE ».

10. COMMUNICATION ENTRE APPLICATIONS DISTANTES

VISUALISATION ET CONTROLE

VIA LE SERVEUR WEB

LabVIEW intègre un serveur qui permet d'exporter les faces avant de VI très simplement. Ce serveur utilise un protocole http sur port 80, donc, à priori, tout ce qu'il y a de plus standard, normalement les passerelles laissent passer le flux de données. La visualisation distante nécessite cependant d'installer LabVIEW ou au minimum LabVIEW Run-Time Engine car des Plug-in sont installés dans Internet Explorer en même temps que ces produits. Consultez les termes de la licence pour l'installation du Run-Time

ACTIVATION DU SERVEUR WEB

L'activation manuelle est présentée ici ; toutes ces opérations sont réalisables par programme en utilisant des noeuds de propriété relatifs à l'application.

Le serveur est activé dans le menu « Tools>Options>Web Server : Configuration ». Il y est demandé : le chemin où sont placées les pages WEB, éventuellement le port et le timeout et un fichier de log. L'item « Web Server : Visible VIs » permet de spécifier quels sont les VIs qui seront exportés vers le serveur WEB la valeur par défaut et *, c'est-à-dire tous ceux dont la face avant est visible. Enfin, l'item « Web Server : Browser Access » spécifie les machines qui peuvent se connecter, et leurs droits d'accès. National recommande de ne pas activer le fichier log car il existe une vulnérabilité (l' envoi de deux LF LF à la place de la séquence CR LF CR LF entraîne un DoS (Denial of Service)) qui crash les serveur WEB et toutes les applications LabVIEW, National recommande d'ailleurs de ne pas publier d'application à risque sur le WEB....

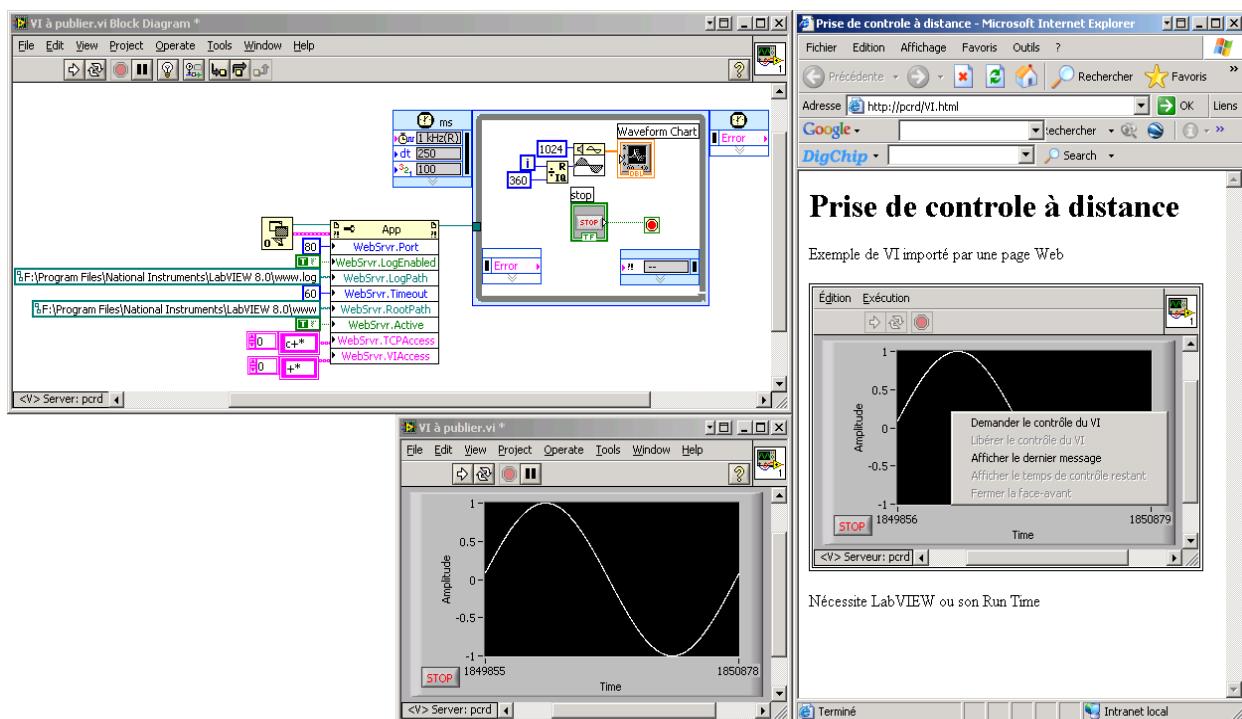
CREATION DE LA PAGE HTML

Une fois le serveur WEB activé, il faut créer une page html sur le disque dans le répertoire du serveur WEB. L'item « Tools>Web Publishing Tools » lance un utilitaire permettant de créer une page WEB simple contenant seulement un titre, une phrase avant et après l'image de la face avant. Il est possible de reprendre le fichier html pour l'enrichir ou d'inclure les parties propres à LabVIEW dans un fichier existant. C'est la page WEB qui accède au VI, il n'y a donc pas de méthode pour lier un VI à une page WEB.

PRISE DE CONTROLE

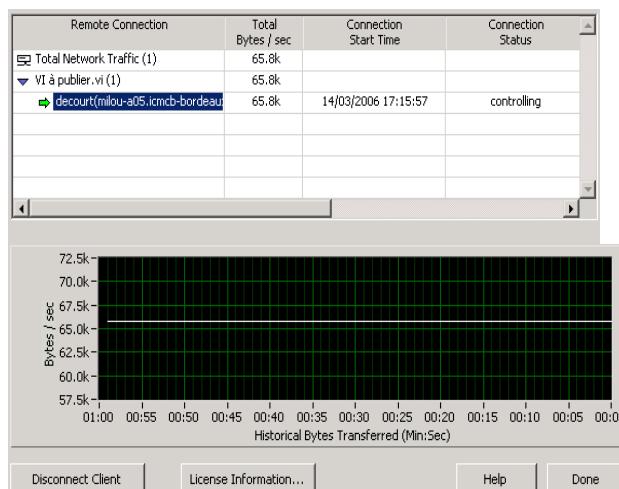
Dans l'étape précédente un nom est donné à la page html. Il suffit dans le navigateur Internet d'entrer le nom de la page, précédée de l'adresse ou du nom DNS de la machine serveur. Une fois la prise de contrôle effectuée, il n'est plus possible de changer le code ou la face avant (ce serait d'ailleurs idiot puisque le VI est censé fonctionner de façon continue et satisfaisante). Le serveur peut à tous moments reprendre le contrôle et s'il le faut le bloquer en faisant un clic droit dans la face avant.

L'exemple suivant vous donne une idée du résultat ; vous avez le code, la face avant et l'explorateur prêt à prendre le contrôle du VI.



SUIVI DES CONNEXIONS

Le « Remote Panel Connection Manager » situé dans le menu « Tools » permet de suivre l'état des connexions aux VIs (Les clients qui attendent le contrôle, celui qui l'a), de voir le débit de données et de déconnecter un client.



Projet : partie 6

Tentez de prendre le contrôle de votre projet via un navigateur internet.

VIA LABVIEW

MODE MANUEL.

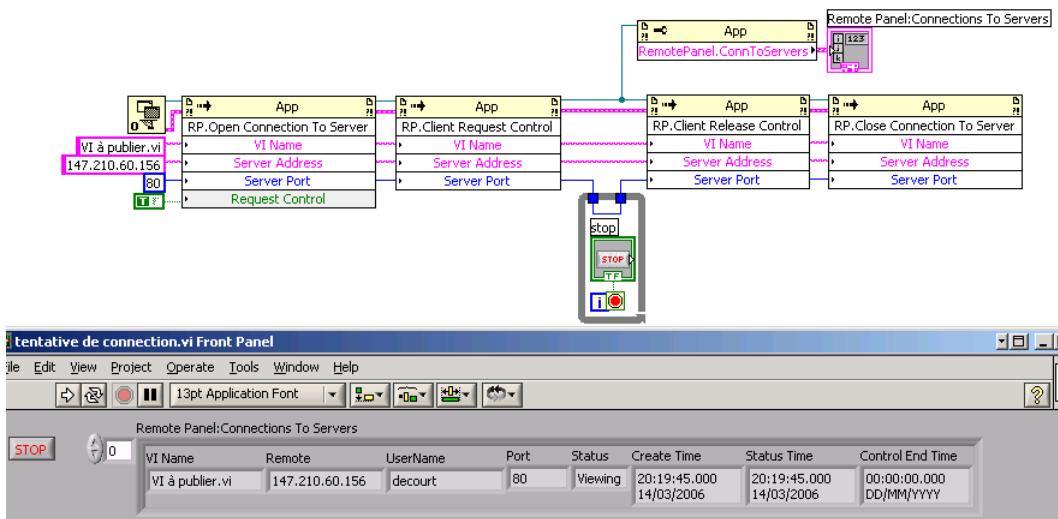
Cette procédure s'apparente à la précédente, mais la prise de contrôle se fait dans l'environnement LabVIEW. Il faut ouvrir « Connect to Remote Panel » dans le menu « Operate », indiquer l'adresse du poste à connecter, et nom du VI (voir l'aide pour les conventions de nom si le VI est dans une librairie ou un projet).

MODE PROGRAMME

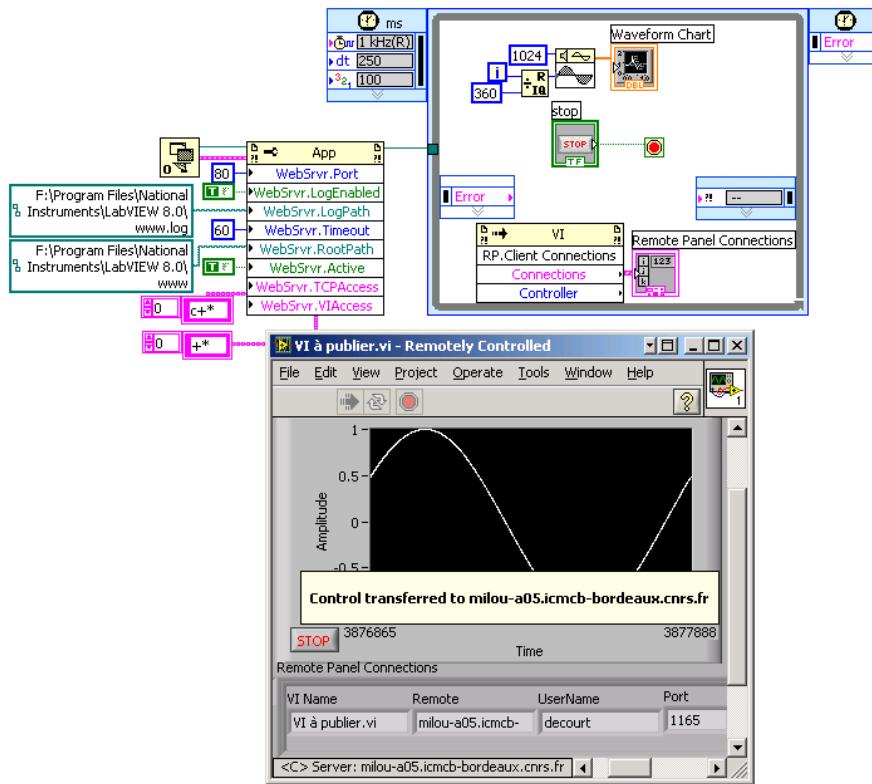
Cette méthode manuelle peut être automatisée et un client LabVIEW peut se connecter via des méthodes de l'application aux VI distants. Attention, le VI n'est pas en mémoire dans la machine cliente, il n'est pas possible d'en récupérer les données ; il est permis simplement de prendre le contrôle et de commander à distance les éléments de sa face avant.

Le client peut connaître les VIs qui ont acceptés d'être connectés, les VIs connectés peuvent connaître qui a pris leur contrôle, et le reprendre.

L'exemple suivant prend le contrôle de l'application « VI à publier »



Voici le code de « VI à publier » et sa face avant après transfert du contrôle.



Vous constatez que le port de connexion entrée (qui reçoit la face avant) est le port 80 et celui de sortie (qui commande le VI distant) est le port 1165.

Le c+* dans la propriété liste d'accès TCP/IP autorise (+) l'accès et le contrôle (c) à toutes les adresses (*).

Le +* dans la propriété liste d'accès aux VIs autorise l'accès à tous les VI.

Projet : partie 7

Expérimenez ces fonctionnalités par programmation.

APPLICATIONS PARTAGEES

Nous traitons ici, non plus de publication WEB et de contrôle à distance, mais d'applications distribuées communiquant entre elles. Ceci signifie que chaque machine contient une partie du code de l'application. Ces codes peuvent être « indépendants » et se communiquer des données ou être dans une architecture « maître esclave » où une partie du code lance et termine des exécutions d'applications distantes.

TRANSMISSION DE DONNEES

VARIABLES PARTAGEES

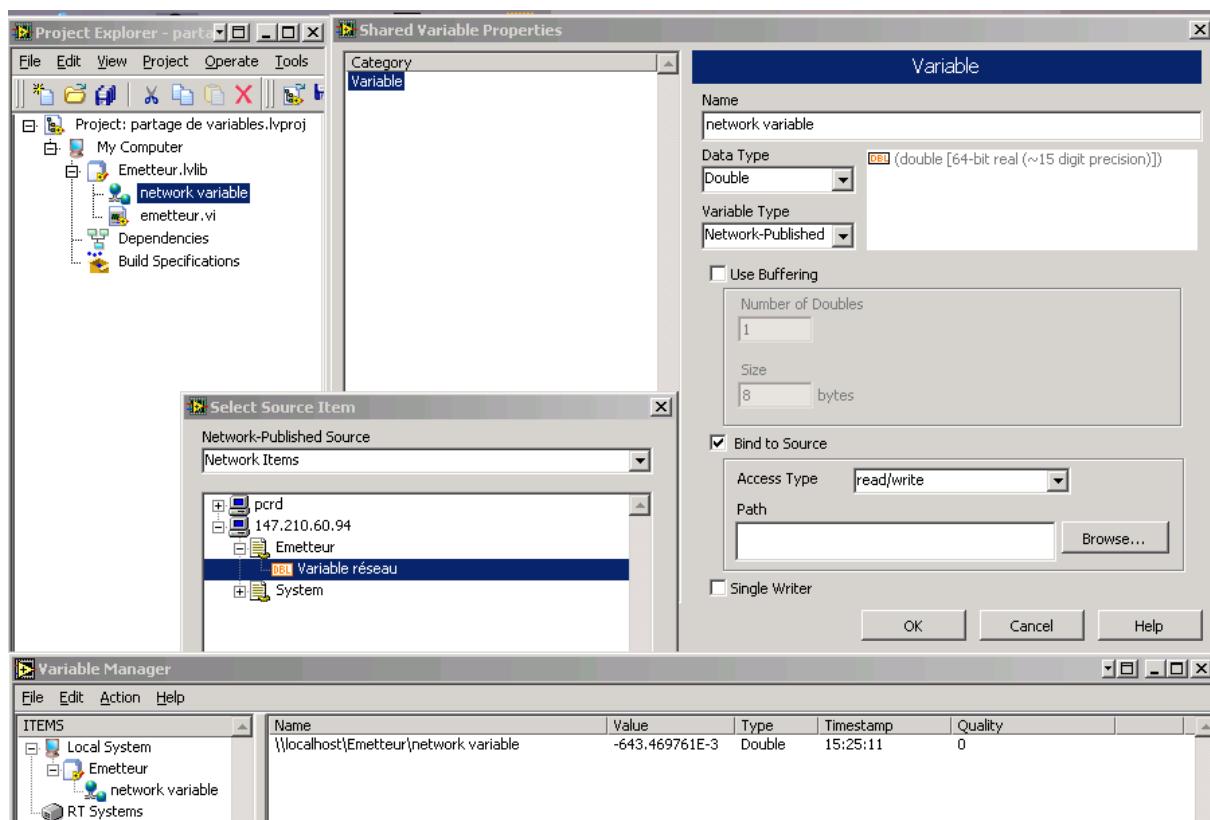
Les variables partagées sont destinées à assurer la communication entre des machines distantes, par exemple une machine RT et un superviseur. Il n'y a pas de systèmes de protection particuliers (cryptages) et cette méthode est réservée à des réseaux locaux ; de plus les ports UDP utilisés par le protocole sont souvent filtrés par les passerelles (port UDP 2343, 5000..., 6000...). Si le pare-feu WINDOWS est activé, il faut autoriser les exécutables suivants : \National Instruments\Shared\Tagger\tagsrv.exe et \windows\system32\lkads.exe.

L'option « network published » des propriétés d'une variable permet de la publier sur le réseau. Il n'est pas possible de lire ou d'écrire directement une variable publiée, il faut que les clients lient leurs variables partagées locales aux variables publiées en cochant « bind to source ».

Pour pouvoir lier une variable réseau, le plus simple est de la déployer sur le serveur de variable. Il faut sélectionner « Deploy » dans le menu contextuel de la librairie pour transmettre sur le serveur le nom, le type et l'adresse réseau de la variable.

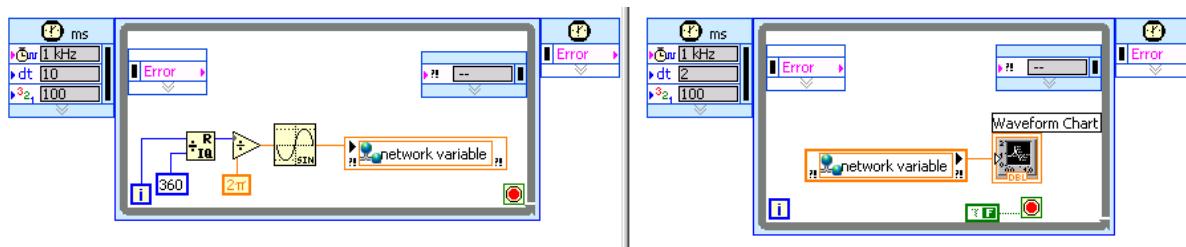
La variable doit alors être vue de tous les ordinateurs du même sous réseau. Si ce n'est pas le cas, il faut enregistrer la machine distante par « Tools>Shared Variable>Register Remote Computer ». Elle doit alors être visible.

En cochant la case « Bind to Source » dans la fenêtre de propriété de la variable, puis en cliquant sur « Browse » une fenêtre de sélection apparaît, il n'y a plus qu'à choisir la machine et la variable distante pour lier la variable partagée locale à la variable partagée distante.



Un utilitaire « Tools>Shared Variable>Variable manager» permet de visualiser et de tracer les variables partagées.

Une application de supervision peut être aussi simple que celle-ci :



Projet : partie 8

Avant de déployer le VI de lancement de A et B sur un poste distant, nous allons remplacer les variables globales par des variables partagées publiées sur le réseau, en cochant « Single writer » il est possible de ne pas utiliser de sémaphores (ces derniers restent d'usage en local !). Vous pouvez constater que le code fonctionne comme avant, pour pouvoir déplacer une partie de l'application il va falloir transmettre les ordres de la file d'attente via le réseau et être capable de référencer un VI distant...

DATA SOCKETS

Les « Data Sockets » ne sont plus recommandés par NI, l'usage de variables partagées doit leur être préféré.....

PROTOCOLES TCP/UDP

INTRODUCTION

Les variables partagées sont un moyen simple de partager des données, mais leur usage est limité à la publication de données entre environnements LabVIEW, et n'est pas vraiment destiné à la transmission de messages. Il est parfois utile de faire usage de protocoles plus standard.

LabVIEW intègre les outils nécessaires à la transmission de données via Internet en utilisant les protocoles UDP et TCP. Ces deux protocoles de la couche de transport se distinguent par leur niveau de fiabilité et leur vitesse.

TCP (Transmission Control Protocol) est un protocole orienté connexion opérant un contrôle de transmission des données pendant une communication établie entre deux machines. La machine réceptrice envoie des accusés de réception lors de la communication, la machine émettrice est garante de la validité des données. Les données sont envoyées par flots (segments) contrôlés uns à uns.

Lors d'une communication, les deux machines doivent au préalable établir une connexion. La machine émettrice (celle qui demande la connexion) est appelée client, tandis que la machine réceptrice est appelée serveur.

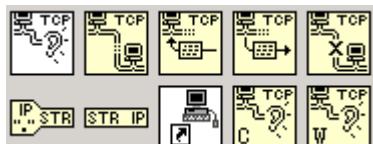
C'est le protocole utilisé par les applications HTTP, FTP, SMTP...

UDP (User Datagram Protocol) n'est pas un protocole orienté connexion. La machine émettrice envoie des données sans prévenir la machine réceptrice, et la machine réceptrice reçoit les données sans envoyer d'avis de réception. Les données sont envoyées par blocs (datagrammes).

C'est le protocole utilisé par les applications DNS, DHCP, jeux réseaux, le streaming.

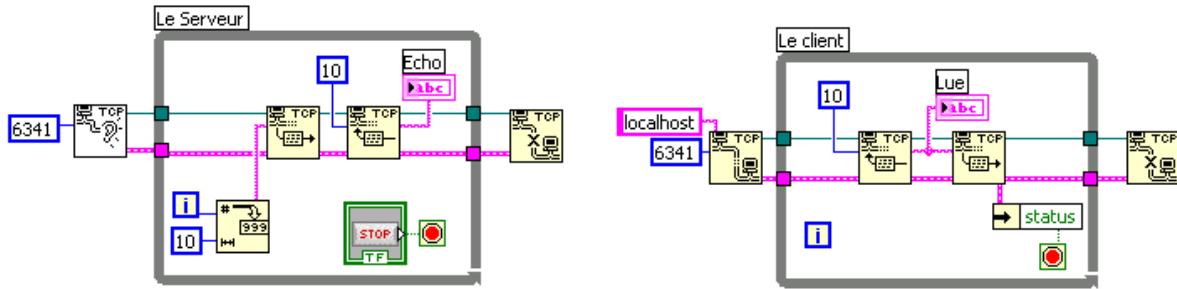
PALETTE TCP

La palette TCP contient les éléments suivants :

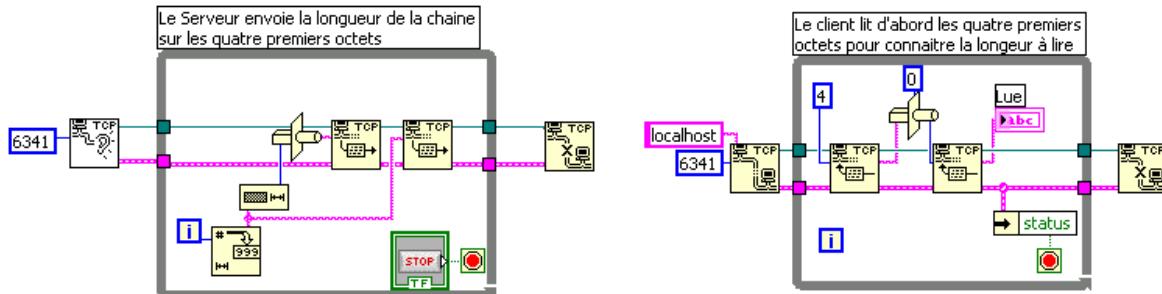


Pour établir une connexion, le client doit ouvrir une communication vers un serveur « listener » à l'aide du VI « TCP Open Connection » en spécifiant l'adresse du serveur. Le serveur, de son coté doit au préalable ouvrir une connexion « TCP Create Listener » puis attendre la connexion avec « TCP Wait On Listener ». Le VI « TCP Listen » réalise ces deux opérations plus certains contrôles de disponibilité du port. Evidemment les ports de communication doivent être identiques.

Une fois la communication établie, l'échange de données se fait par « TCP Read & Write ». Dans l'exemple ci-dessous le serveur transmet sur 10 caractères l'indice de boucle, le client lit cette valeur et la renvoie en écho.



Il est possible de terminer la lecture sur nombre de caractères, ou sur terminateur, il est courant de transmettre en premier, sur 32 bits, le nombre de caractères à lire :

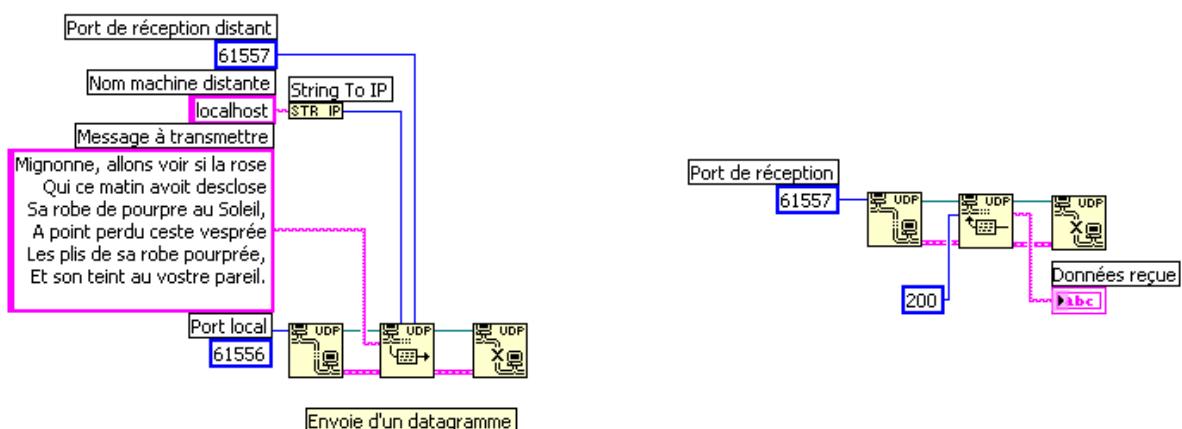


PALETTE UDP

Si l'on doit transmettre des flots très importants de données et qu'une perte de données est sans conséquence (flux audio, vidéo...). La palette UDP contient les éléments suivants :



Une communication UDP comporte une ouverture, un envoi ou réception, une fermeture. Le VI d'envoi doit connaître l'adresse de la machine distante et le port UDP associé, le VI de réception n'a besoin que du nombre de caractères à recevoir et du numéro de port.



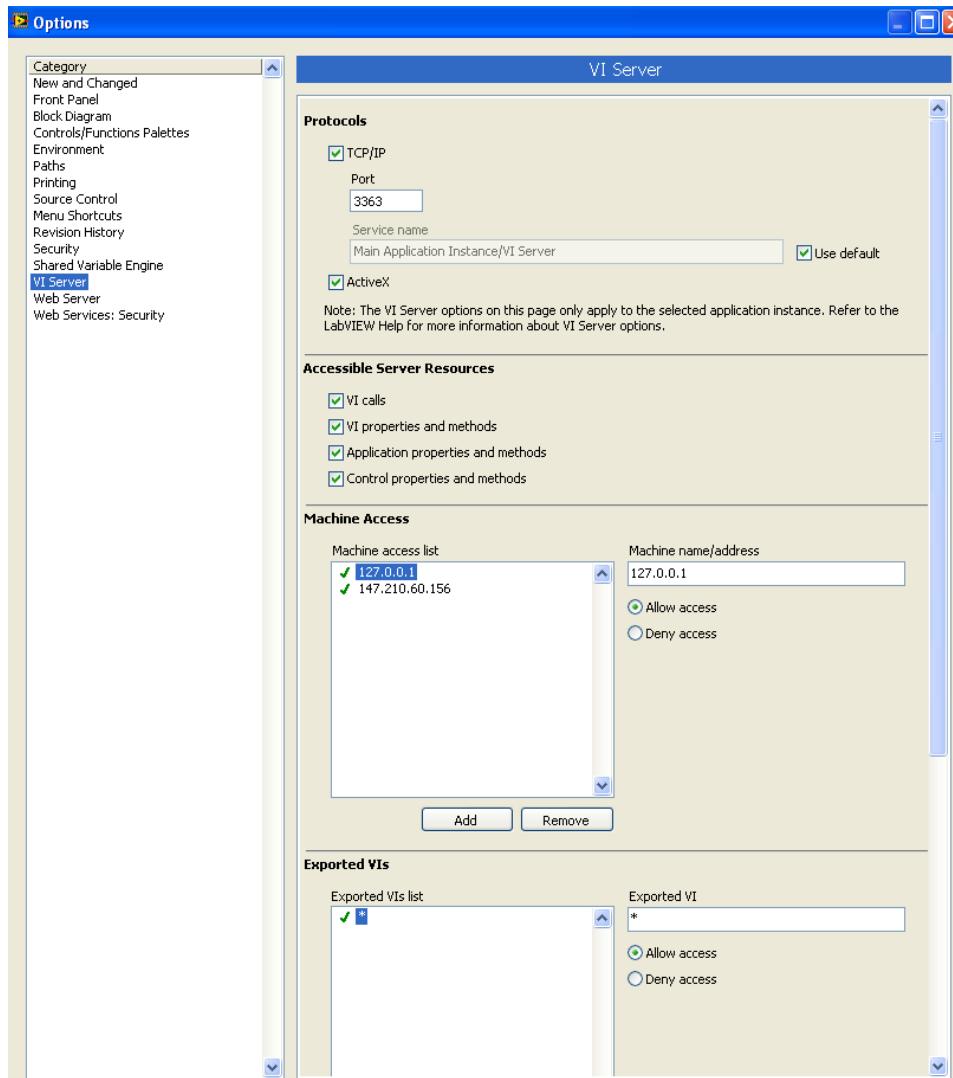
Projet : partie 9

Implémentez la file d'attente sur un protocole TCP et la notification sur un protocole UDP

CONNEXION AU « VI SERVER »

L'application Labview dispose d'un chargeur de VI nommé « VI serveur », il est utilisé par Labview pour charger et exécuter les VI mais peut aussi être utilisé par un VI pour charger un autre VI (Cf : Chargement dynamique, « VI Server »)

Le VI serveur est capable d'exécuter des ordres provenant d'une autre machine, le support de cette fonctionnalité est défini dans le menu « Tools/Options ». Il est possible de limiter les accès à certaines machines et de filtrer les actions possibles, de limiter ces actions à certains VI.



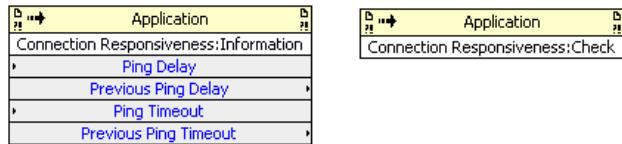
PRINCIPE

Pour exécuter un VI sur une autre machine, il faut dans un premier temps créer une référence de l'application Labview sur la machine distante (Labview doit donc être chargé en mémoire sur la machine distante par exemple via DCOM)

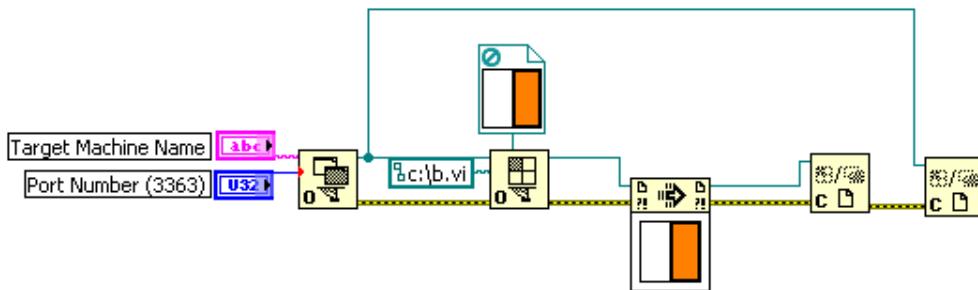


Machine name est soit une adresse IP, soit un nom de machine connu par un serveur DNS

Il est possible de vérifier la disponibilité de la machine distante en utilisant un nœud de méthode **Connection Responsiveness:Information** ou **Connection Responsiveness:Check**.



Une fois connecté, il faut utiliser les mêmes procédés qu'en local (ouverture d'une référence sur le VI, nœud d'exécution)



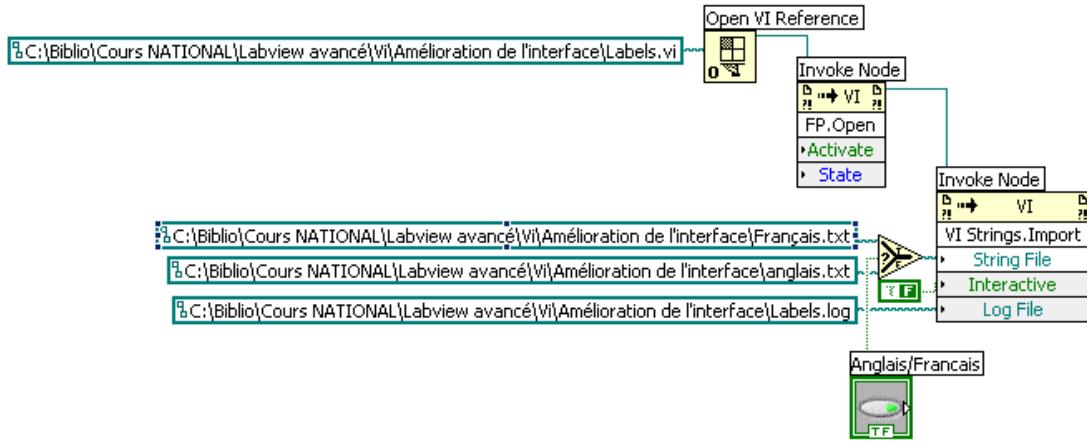
Projet : partie 10

Tentez d'exécuter le VI A sur le poste voisin, en transmettant la file d'attente et la notification non plus sur LocalHost, mais sur l'IP du poste voisin.

11. AMELIORATION DE L'INTERFACE

IMPORTATION DES TITRES D'OBJET.

Il est possible d'importer et d'exporter l'ensemble des textes contenus dans une face avant et/ou dans un diagramme vers un fichier texte. Ceci permet d'adapter les versions du logiciel à des langues différentes. Cette fonctionnalité peut être manuelle (en passant par Tools->Advanced->Import/Export String) ou réalisée par la méthode « VI String.Import » au sein d'un programme. L'exemple suivant permet de changer la langue de la face avant du VI Label.vi en sélectionnant soit le fichier anglais.txt ou français.txt.



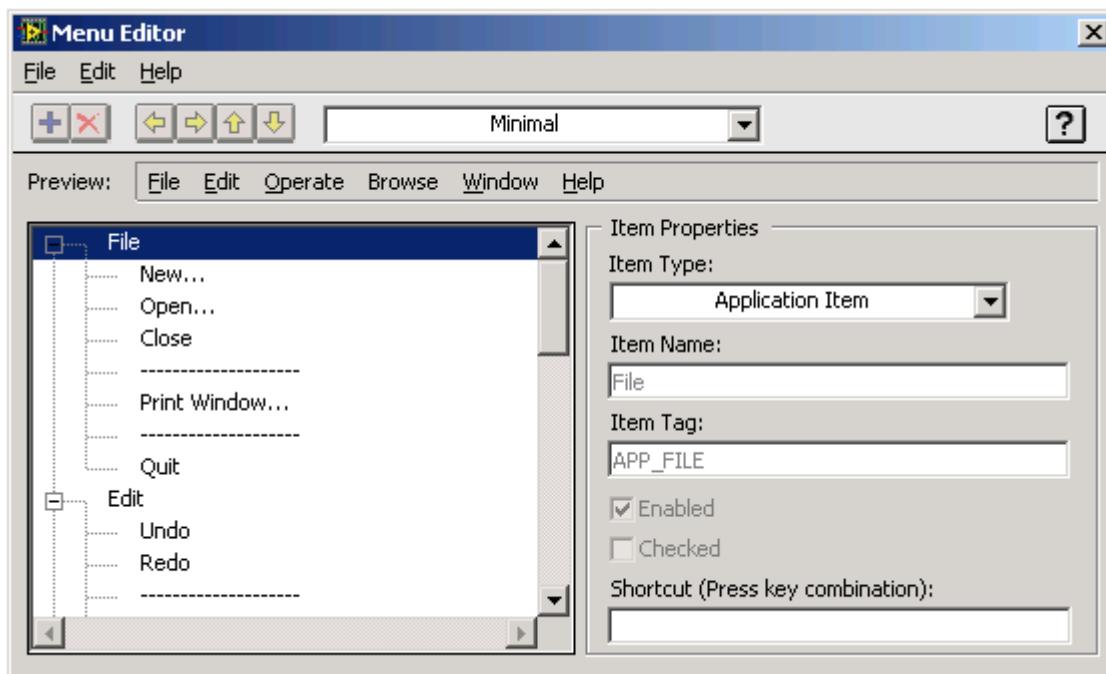
Projet : partie 10

Ajoutez à votre projet la possibilité de sélection de langue Anglais/français. Attention cette propriété n'est pas modifiable en cours d'exécution, il faut donc prévoir un VI lanceur.

GESTION DES MENUS

EDITION.

Une barre de menu peut être adjointe à un VI, un éditeur de menu est accessible par la commande Edit->Run time menu. La hiérarchie des menus est connue par l'indentation dans la fenêtre de droite, pour chaque item est défini un ITEM TAG qui sera retourné au programme lors de l'utilisation de la barre de menu. Il est possible de retrouver, soit « item tag » de la sélection, soit le chemin complet (tous les items de la hiérarchie séparés par des :) :

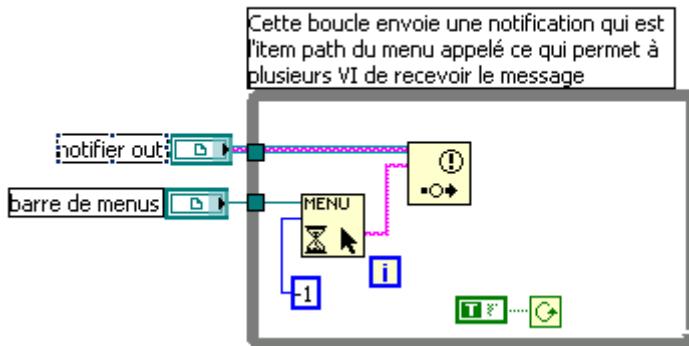


Le menu édité avec le VI lui est automatiquement lié. Il est possible de changer le nom du fichier grâce à la propriété « RT menu Path » du VI. Les menus étant séparés de la face avant du VI ne sont pas exportés lors de la procédure décrite plus haut.

GESTION.

La palette de gestion des menus se trouve dans Dialog & Interface

La gestion de menu réside essentiellement dans une boucle qui attend l'action de l'opérateur. Elle récupère le Item Tag et l'envoie via une file d'attente aux VIs qui doivent effectuer une action.



L'icône « Current VI menu bar » renvoie une référence pour tous les VIs de gestion de menu. L'icône « Get menu selection » renvoie le choix de l'utilisateur. D'autres VIs permettent de modifier les éléments du menu.

Si la face avant doit réagir à d'autres sollicitations de l'utilisateur, il est préférable d'utiliser une structure de type événement pour gérer à la fois les menus et les autres objets de la face avant. Mais c'est une autre histoire que l'on vous racontera bientôt.

Projet : partie 11

Continuons notre projet en remplaçant maintenant la sélection faite par le Ring par un menu. Il contiendra deux Items principaux : Fichier et Mesure. Le premier conduira à Quitter, le second aux Acq A Attente A.....

MENUS DES OBJETS DE FACE AVANT

Tous les objets de la face avant peuvent avoir des menus contextuels personnalisés. Ils sont accessibles dans le menu contextuel « advanced » des objets. La méthode est la même que pour les menus de panneaux. Il est alors intéressant d'utiliser une structure événement pour gérer l'ensemble des interactions entre utilisateur et menus.

STRUCTURE D'EVENEMENT

DEFINITION

Windows est un « message-based system », Toute action demandée par un utilisateur crée un ou plusieurs messages pour être réalisée. Ces messages sont transmis entre les objets logiciels et portent en eux des informations concernant les destinataires, la nature de l'action...

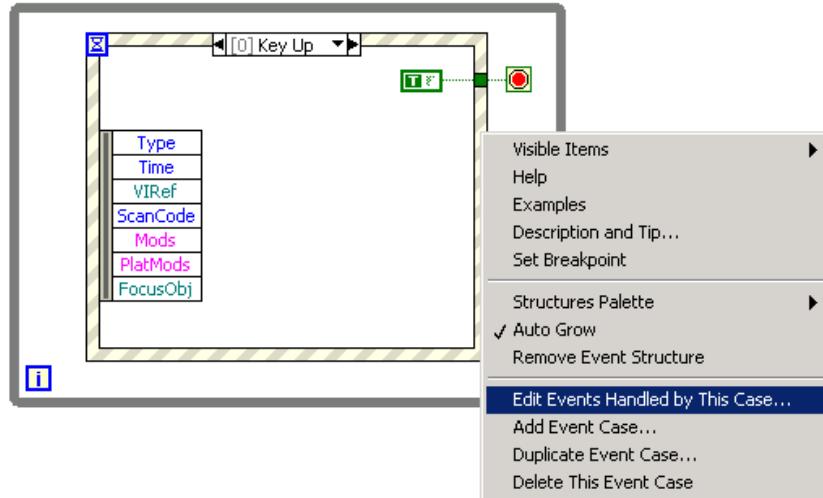
La majorité des messages windows sont logés dans une file d'attente, les messages prioritaires sont envoyés directement à l'application.

Les structures événementielles permettent de récupérer ces messages « au vol » pour les traiter. Le concepteur peut aussi générer ses propres événements destinés à d'autres parties du programme.

Une structure événementielle est une structure de type CASE, à ceci près que c'est un EVENT qui détermine le cas traité.

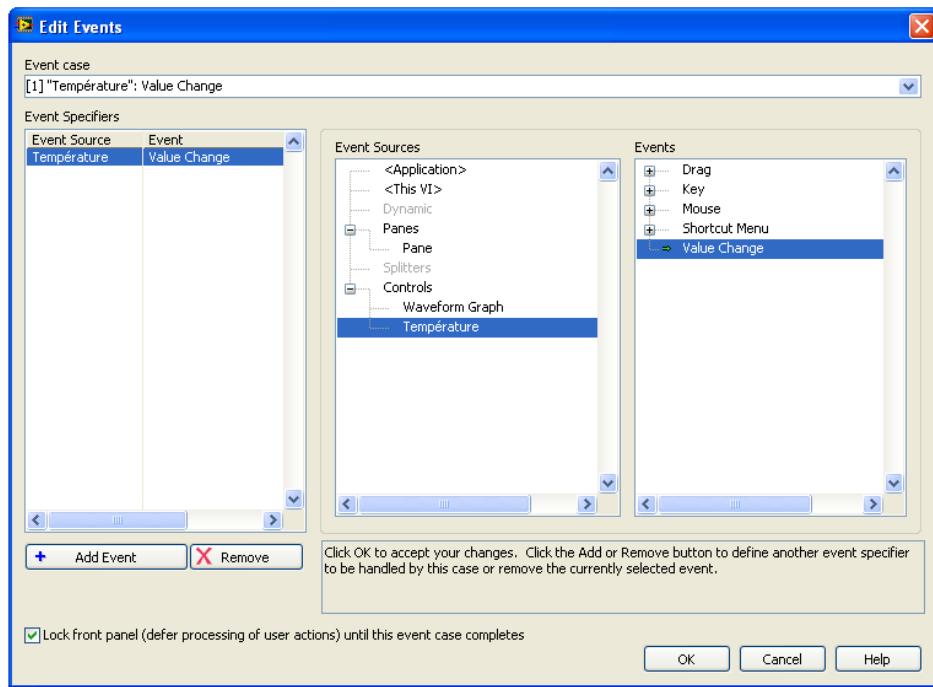
GESTION

Une structure événement se place dans une boucle WHILE. Elle peut soit attendre indéfiniment une action ou exécuter un code après un TIMEOUT. La boucle ci-dessous se termine lorsque l'on relâche le bouton de la souris dans la fenêtre de l'application.



Chaque cas de la structure peut traiter un ou plusieurs événements, par défaut le seul événement traité est « TIMEOUT ». Lorsque l'on édite les événements traités par la structure, l'ensemble des éléments de la face avant apparaît à droite, les événements répertoriés pour ces objets apparaissent à gauche. S'il s'agit de commandes de la face avant, la propriété 'Value Change' est la plus courante.

Il est recommandé de ne mettre qu'une seule structure événement dans une boucle, et de ne pas gérer un même événement par plusieurs structures événement.



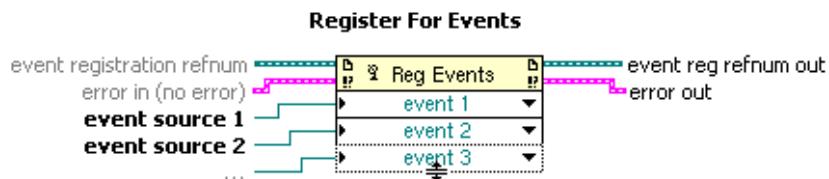
Par défaut la case « Lock front panel..... » est cochée pour éviter d'empiler des événements utilisateurs (pour les clicomaniacodéprésifs)

Projet : partie 12

Continuons notre projet en remplaçant maintenant la gestion classique des menus par une structure événement. Un menu local sera ajouté aux deux LED pour en définir la couleur dans l'état On et Off.

EVENEMENT PROVENANT D'AUTRES VI

Si le VI contenant la structure événement n'est pas celui qui contient les éléments de la face avant, ces derniers n'apparaîtront pas dans la liste des objets. Il faut alors référencer l'objet dynamiquement lors de l'exécution en transmettant une référence de l'objet au VI 'Register For Events'



Ce VI reçoit des références d'objets sources d'événements et enregistre les types d'événements autorisés

L'exemple suivant illustre le principe. Le VI 'Bt Stop' possède un bouton stop sur la face avant qui doit arrêter l'exécution du VI 'Wait for stop'.

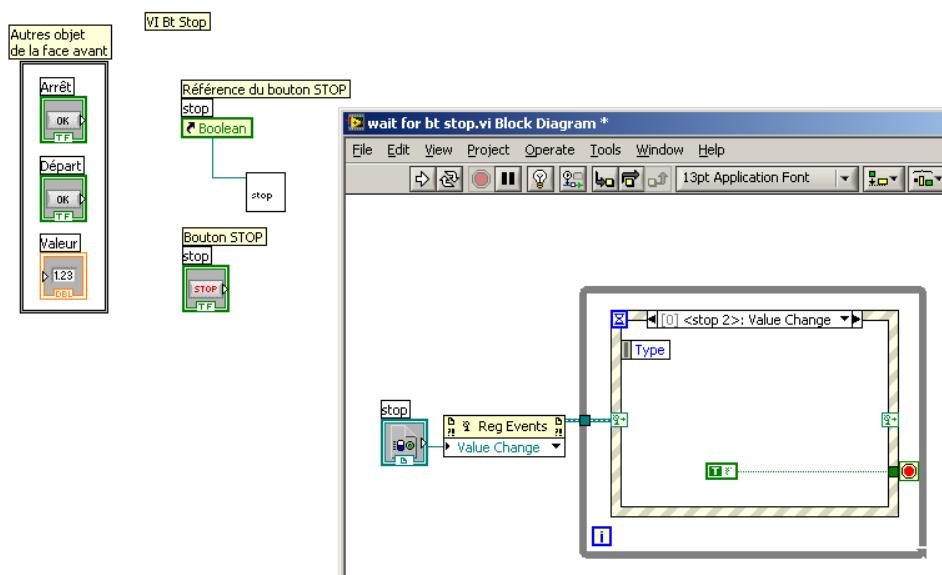
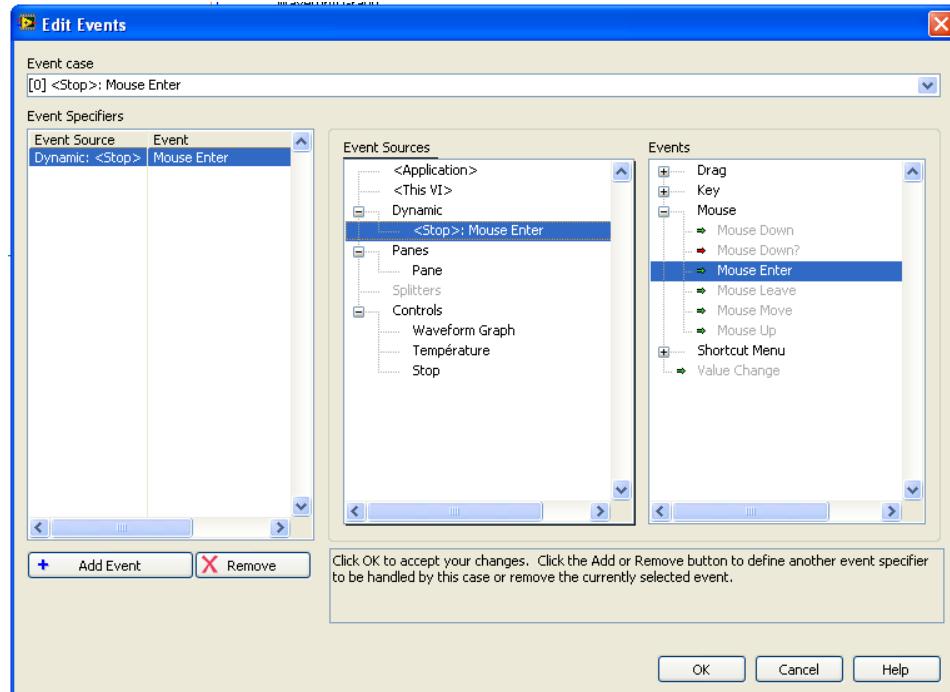
Les étapes sont les suivantes :

Créer sur la face avant une référence de control type booléen

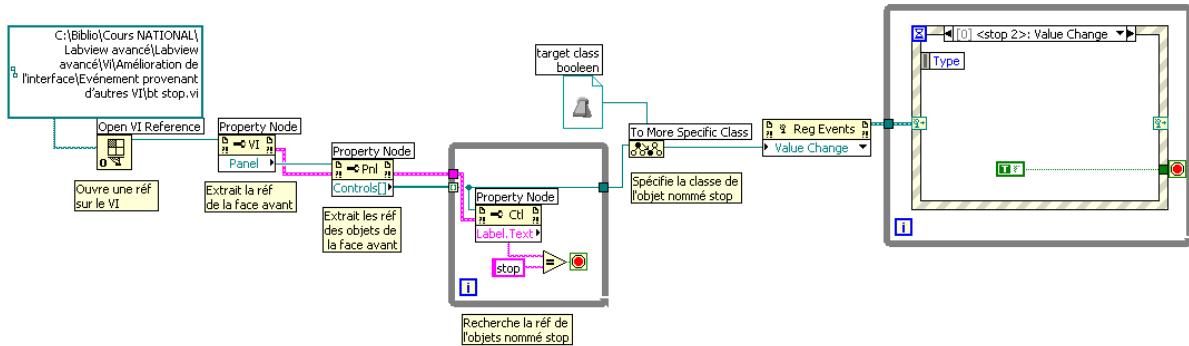
Câbler la référence dans ‘Register For Events’, sélectionner l(es) événement(s) souhaité(s) . ‘Register For Events’ est dans la palette ‘Dialog & User Interface > Events’.

Pour câbler ‘l’Event Reg Refnum Out’ vers la structure événement, il faut faire apparaître le terminal ‘Dynamic event terminal’ dans le menu contextuel de la structure.

Une fois le terminal ‘Dynamic event terminal’ câblé, l’item ‘Dynamic’ précédemment grissé de la fenêtre ‘Edit Item’ apparaît.



S'il n'est pas possible de transmettre la référence de l'objet (par exemple, le VI Bt Stop est chargé dynamiquement par un bootloader) il faut chercher dans l'ensemble des objets celui qui s'appelle ‘stop’ et transmettre sa référence :

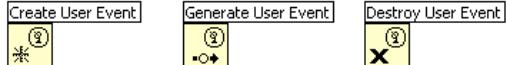


Projet : partie 13

Que diriez-vous de gérer localement au niveau des VIs A et B la réception des commandes des menus sans passer par les files d'attente? Attention l'événement « sélection dans un menu utilisateur » est un événement relatif au VI.

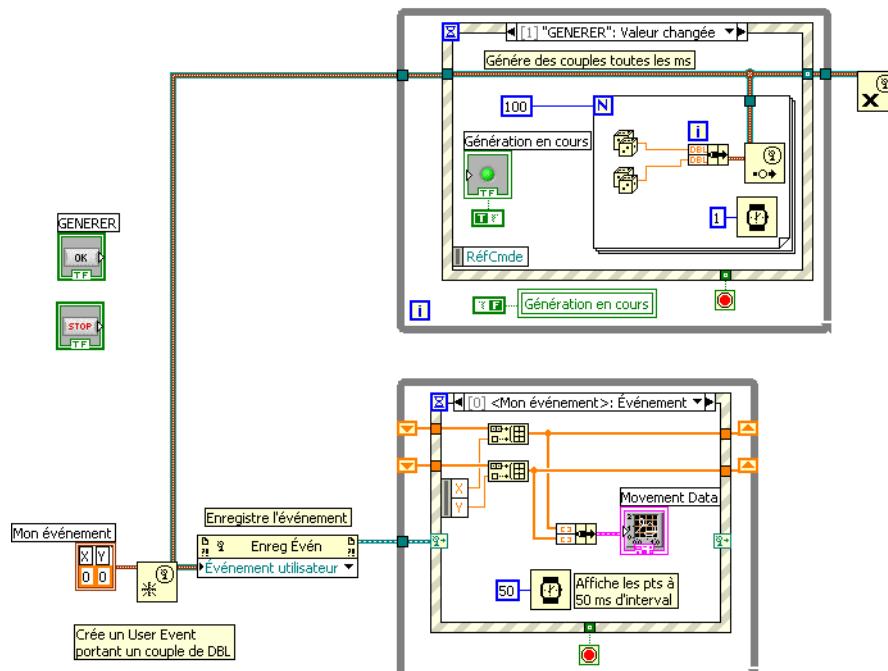
ENREGISTREMENT D'EVENEMENTS UTILISATEUR

Il est possible de générer un événement utilisateur et de transmettre à une structure événementielle des données. Les événements et les données associées sont posés dans une file d'attente automatiquement. Les trois VI destinées à créer, générer et détruire un événement utilisateur sont :



Les événements utilisateur simplifient le traitement de messages entre VI, ils sont empilés à chaque appel. Ils sont utiles pour l'exécution de scripts, le traitement d'exceptions...

L'exemple suivant utilise deux boucles, l'une produit 100 couples (valeur, temps), et 100 événements utilisateurs, l'autre attend ces couples sur un 'User Event'. Les couples sont affichés 50 fois plus lentement qu'ils ne sont produits.



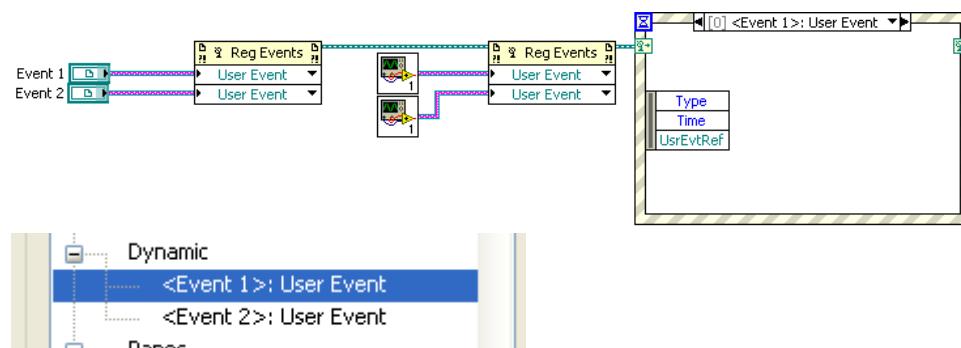
Le nom du type de données transmises donne le nom de l'événement, indispensable si plusieurs événements utilisateurs sont enregistrés...

Projet : partie 14

Et maintenant si les VI A et B généraient des événements utilisateur au lieu de notifications ?

CHANGEMENT DU NOM D'EVENEMENT

Si un VI qui fournit un événement est utilisé plusieurs fois, tous les événements auront le même nom. L'exemple suivant permet de prénommer les événements.



OBJETS ACTIVEX ET .NET

QU'EST CE ?

ACTIVEX

ActiveX regroupe plusieurs composants logiciels basés sur « Component Object Model (COM) ». Cette architecture permet de concevoir ces composants à partir de n'importe quelle plate-forme de développement qui supporte le modèle COM. Ils peuvent ensuite être utilisés en tant qu'objet dans n'importe quel langage de programmation. La spécification COM permet l'appel de méthodes et l'accès aux propriétés d'un objet.

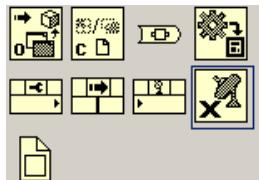
.NET

".NET" ("dot Net" en anglais) dessine la stratégie Internet de Microsoft. Objectif: faire évoluer les solutions Windows vers un modèle d'applications hébergées et proposer une plate-forme logicielle sur laquelle les entreprises pourront s'appuyer pour échanger et mettre à disposition des données et des services applicatifs. Les composants .NET sont basés sur .NET Framework. Ils sont utilisables comme les activeX.

UTILISATION D'UN COMPOSANT ACTIVEX ET .NET

LabVIEW peut être client ActiveX et .NET, c'est-à-dire accéder à des objets, des propriétés, des méthodes et événements d'applications ActiveX. Il peut aussi être serveur (pas directement pour .NET), d'autres applications peuvent accéder à des objets, des propriétés et des méthodes de LabVIEW. Les composants ActiveX et .NET peuvent posséder une interface utilisateur ou non. La principale difficulté est de trouver la documentation complète du composant que vous voulez utiliser.

ACTIVEX



Pour accéder à une application ActiveX, utilisez une commande ou une constante « refnum automation » créer une référence. En choisissant « Sélectionnez une classe ActiveX>parcourir » dans le menu contextuel du « refnum », vous voyez l'ensemble des composants ActiveX enregistrés, sélectionnez celui désiré (hé hé hé...). Le VI « Automation Ouvrir » ouvre l'application. Il faut ensuite jongler avec noeuds de propriété et de méthode pour obtenir le résultat escompté. Le code suivant ouvre Excel, rend l'application visible, ajoute un « livre de travail (les trois feuilles par défaut) » et ferme les références.

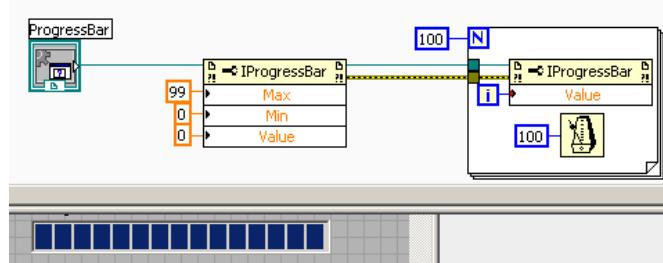


Il est utile de consulter la documentation de l'application. Voici les premières lignes d'introduction de MS pour excel._application :

L'objet **Application** d'Excel représente l'application Microsoft Office Excel 2003. L'objet **Application** contient une grande quantité d'informations sur l'exécution de l'application, les options appliquées à cette instance et les objets de l'utilisateur actuel ouverts dans l'instance. L'objet **Application** fournit de nombreux membres que, pour la plupart, vous n'aurez jamais besoin d'étudier. En revanche, certains sont cruciaux pour garantir le bon comportement de votre application.

Une bonne connection Internet et trois écrans géants peuvent servir.

Pour accéder à un control ActiveX (objet ayant une interface utilisateur), il faut déposer un conteneur sur la face avant du VI, puis dans le menu local « Sélectionner un objet ActiveX ». Comme précédemment : jongler avec nœuds de propriété et de méthode. Le code suivant fait avancer une barre de progression microsoft.



ENREGISTREMENT D'ACTIVITE D'UN ACTIVEX OU D'UN .NET

GENERALITE SUR LES « CALLBACK »

Les composants ActiveX et .NET génèrent des événements et renvoient au conteneur .NET des données propres à cet événement. La procédure de gestion d'événement fait appel à des « Callback ».

Une « Callback » est une fonction appelée via un pointeur sur fonction (on stocke l'adresse de retour et on fait un « GOTO 0xHHHH »). De fait il n'y a pas vraiment de couple appelant/appelé, l'appelant ne sait pas qui est l'appelé (il n'a que son adresse de début), il ne connaît qu'un prototype de la fonction. Ceci permet de changer dynamiquement la fonction appelée simplement en changeant l'adresse du pointeur d'appel, (s'entend bien qu'il faut que les prototypes soient identiques). Cette technique est très utilisée dans l'API Windows, car le système gère les événements sur des objets et demande à l'applicatif 'si on clic sur l'objet X qui dois-je appeler ?' comme l'API est déjà compilée, il n'est pas possible de lui passer un nom de fonction (il faudrait tout recompiler), alors l'adresse de la fonction à appeler est passée en paramètre, la fameuse « Callback ».

ENREGISTREMENT D'UNE « CALLBACK »

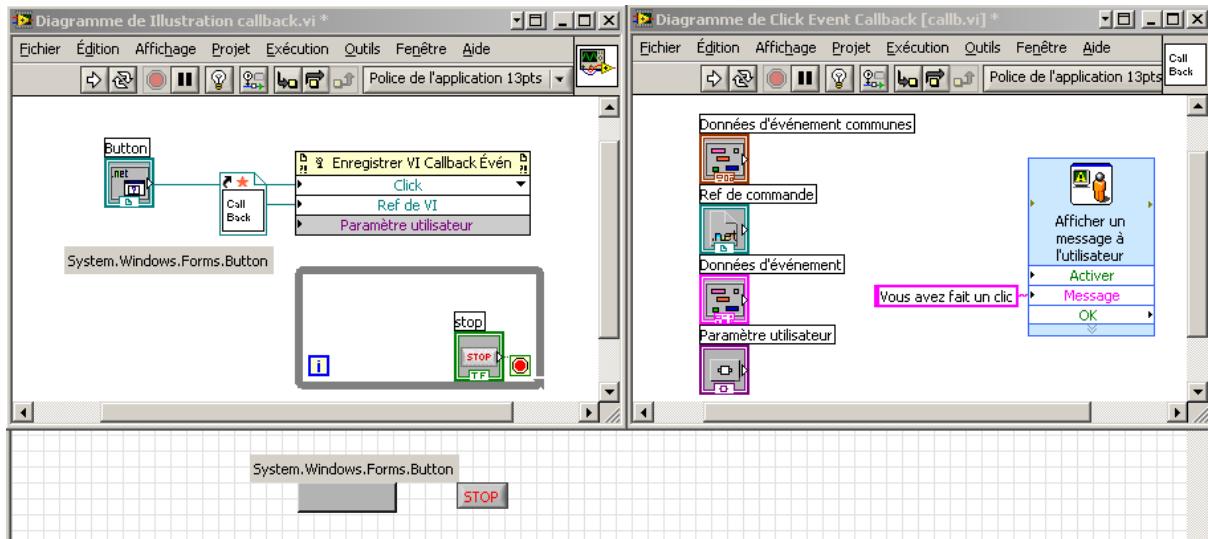
A la vue de ce qui précède, il apparaît que pour un événement donné il faut enregistrer l'adresse de la fonction à appeler. Ceci se fait avec le nœud « Enregister un VI Callback d'événement »



réf. de source d'événement une référence ActiveX ou .NET. Une fois câblé, le conteneur export les événements disponibles pour cet objet, sélectionnez l'événement que vous voulez gérer.

référence au VI est une référence de VI stricte au VI callback qui doit être réentrant. Le VI (prototype vide) est créé par le menu local «Créer un VI Callback ». Ce VI doit être créé lorsque tous les fils cités ici sont câblés et l'événement à traiter choisi car les paramètres transmis en dépendent.

paramètre utilisateur contient des données que LabVIEW transfère au VI callback lorsque l'objet .NET ou ActiveX génère l'événement. Tout type de données LabVIEW, est accepté (ne sert qu'à créer la commande en face avant du prototype).

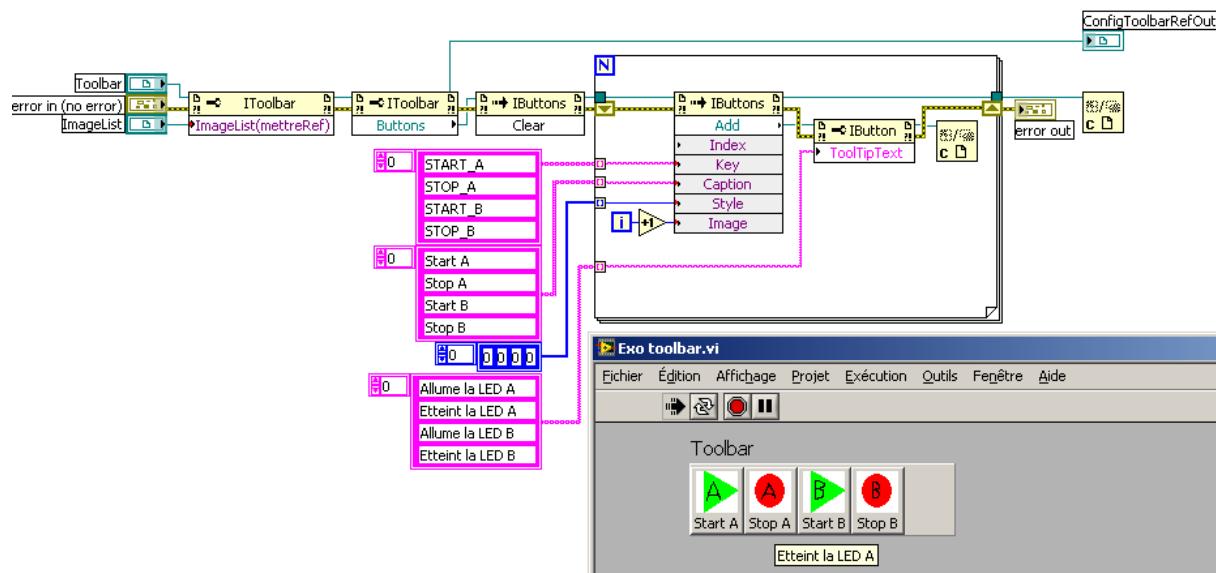


L'exemple ci-dessus utilise un composant .Net de la classe « System.Windows.Form.Button ». Le programme de gauche enregistre la callback d'événement « Click » de l'objet .NET et attend. La Callback (à droite) affiche un message lorsque l'utilisateur clique sur le bouton.

Projet : partie 15

Vous allez créer une barre d'outils contenant 4 boutons StartA...StopB, l'appui sur l'un de ces boutons génère un événement utilisateur comme les VI A et B (il serait ardu d'essayer de générer des événements comme si l'on cliquait sur les menus car ce sont des menus Windows) ? L'objet ActiveX est un « Microsoft Toolbar Control », il faut dans un premier temps extraire la une référence de bouton par la propriété « Buttons », puis utiliser la méthode « Clear » et des méthodes « Add » pour ajouter les différents boutons à la barre d'outils.

Pour vous inspirer un peu ...



LES COMMANDEX

GENERALITE

Les CommandeX sont des commandes LabVIEW à qui l'on a rajouté des méthodes et des propriétés personnalisées, cela permet de réutiliser très facilement ces contrôles au lieu de mettre le code de gestion dans l'application même.

Une CommandeX se crée dans un projet, quatre fichiers au minimum sont créés.

Donnée représente le type de données que reçoit ou fournit le contrôle (données simples, tableau, cluster...)

Etat est un cluster qui contient l'ensemble des variables capables d'agir sur l'état de la commande.



Façade définit l'apparence du contrôle lorsqu'il sera posé sur la face avant, ce VI contient le code définissant le fonctionnement du contrôle dans une grande structure événement.

Init est utilisé par LabVIEW lors du placement du VI sur la face avant, ou lors du chargement en mémoire.

CREATION

Pour illustrer la création de CommandeX, nous allons créer une LED d'un type particulier, cette LED aura une propriété Triste/Joyeuse qui change les couleurs de la LED (triste -> couleurs sombres, joyeuse -> couleurs vives).

Créez un nouveau projet, puis une nouvelle CommandeX.

Dans le **Données.ctl** posez une LED (définit le type de données en E/S pour ce contrôle).

Dans **Etat.ctl**, posez dans le cluster un booléen nommé Joyeuse_Triste (donne l'ensemble des états possibles de ce contrôle).

Dans Façade.vi posez une LED (se sera la vue de l'objet sur la face avant).

Editez le code de Façade.vi. Il s'agit d'une boucle contenant une grande structure case.

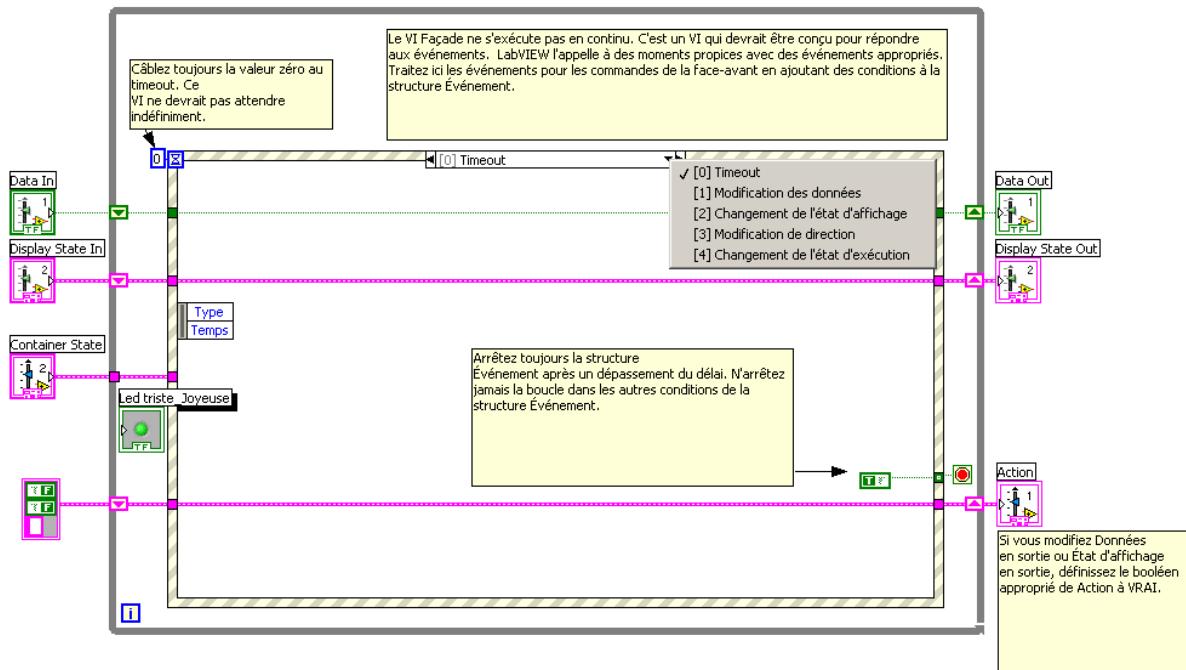
Les données qui y circulent sont :

Data : c'est la valeur que reçoit ou que fournit l'indicateur, ou la commande, c'est donc **Données.ctl**.

Display state : c'est l'état de la commande (triste ou joyeuse) , c'est donc **Etat.ctl**

Action : arme manuellement l'opération à faire au prochaine tour de boucle (data changed/ State changed)

Container State : c'est l'état du conteneur de la commande (indicateur/commande, Edition/Exécution)



Par défaut elle traite les événements :

Timeout : ne rien changer dans ce cas

Modification des données, c'est-à-dire que data_in est différent de data_out, en général on transfère data vers une locale de la variable affichée

Changement d'état de l'affichage cet événement est traité lorsque l'on appelle une propriété ou une méthode, c'est ici que le changement de couleur de la LED doit être fait un fonction de la valeur de « display state ».

Modification de direction traite le passage Indicateur/Contrôle, et **Changement de l'état d'exécution** traite le passage Edition/Exécution, nous n'avons pas de code à y écrire.

Si par exemple vous ajoutez un menu local à votre contrôle, il faudra ajouter une condition d'événement pour gérer ce menu...

Projet : partie 16

Vous allez utiliser les LEDs Triste/Joyeuse que vous venez de créer.

12. PROGRAMMATION ORIENTEE OBJET

APERÇU

La programmation objet sous LabView date d'une dizaine d'années. Elle était implémentée en langage G directement par les utilisateurs ou par des tiers (Endevo en particulier), elle s'appuyait sur l'utilisation de clusters et de files d'attente. NI a créé sa propre vision de la programmation objet à partir de la version 8.5 en imaginant des librairies de classes. C'est cette implémentation (largement inspirée des versions communautaires) que nous utiliserons

LA PROGRAMMATION ORIENTEE OBJET

Nous ne ferons pas ici un traité sur la programmation orientée objet, mais de brefs rappels de concepts et de vocabulaires. La programmation classique est dite structurée, le principe est de diviser un programme en sous-programmes, afin de pouvoir en gérer la complexité, faciliter la réutilisation du code et le débogage. Le traitement est le maître mot : "Que fait le programme ?".

Les langages orientés objets s'intéressent plus aux données : "Sur quoi travaille le programme ?"

MODELES A OBJETS

Les modèles à objets ont été créés pour modéliser le monde (c'est très prétentieux !), toute entité du monde réel est un objet, et vice versa.

LA CLASSE

Un objet, pour représenter une entité réelle doit pouvoir la matérialiser. Ceci est fait à l'aide **d'attributs** qui caractérisent ce **qu'est** l'objet et de **méthodes** qui caractérisent ce **que fait** l'objet. L'ensemble des propriétés d'un objet (attributs & méthodes) constitue un ensemble appelé une **classe**. Un objet sera l'**instanciation** d'une **classe**, son nom identifiera de façon unique cette **instance**.

L'ENCAPSULATION

A cet objet correspond un bloc de code (les méthodes partagées par tous les objets de la même classe) et un bloc de données (les attributs, uniques pour cet objet), le tout enfermé dans la même boîte, c'est la notion **d'encapsulation**. Cette boîte doit communiquer avec le monde extérieur, et cependant doit protéger son contenu, c'est pourquoi une partie est visible de l'extérieur, c'est la partie **publique**, l'autre n'est accessible qu'aux méthodes de la classe, c'est la partie **privée**. Les attributs sont souvent privés et ne sont accessibles que via des méthodes (les fameux **Get & Set**) appelées **accesseurs** et **mutateurs**, ce qui permet de protéger l'accès aux données internes (par exemple l'étendue de validité d'un attribut).

L'HERITAGE

L'**héritage**, permet de créer une nouvelle classe (nommée : « classe fille », « classe dérivée » ou « sous classe ») à partir d'une classe existante (nommée : « super classe », « classe de base » ou « classe parente »).

La classe dérivée contient les attributs et les méthodes de sa superclasse. L'intérêt est de pouvoir définir de nouveaux attributs et de nouvelles méthodes pour la classe dérivée, qui viennent s'ajouter à ceux et celles héritées. La classe fille est plus « spécialisée » que la super classe (Par exemple, « voltmètre » pourrait être une classe dérivée de « Appareil de mesure »)

POLYMORPHISME

Le polymorphisme (*qui peut prendre plusieurs formes*) est relatif aux types de données traitées par les méthodes des objets. Pour parler simplement, la même méthode (identifiée par le même nom) peut traiter plusieurs types de données, il est bien évident que cette méthode est écrite autant de fois qu'elle à de types de données à traiter (par exemple il est possible de surcharger l'opérateur + pour qu'il opère sur des entiers ou des tableaux d'entiers)

Certaines méthodes d'une super classe peuvent être ainsi redéfinies dans classes dérivées, c'est elles qui seront appelées par les instances de la classe dérivée. Si toutes les classes dérivées surchargent cette méthode, une méthode **abstraite** (méthode qui ne possède pas d'implémentation) sera définie dans la super classe, et elle sera surchargée dans chaque classe dérivée (par exemple une méthode « mesure » sera définie mais non implémentée dans la classe « appareil de mesure », et elle sera redéfinie dans la classe « voltmètre » et dans la classe « fréquencemètre »).

DEUX MOTS D'UML ET DE POO

Pour illustrer la programmation orientée objet, nous présenterons un certain nombre de diagrammes de classes au format du langage de modélisation nommé UML (Unified modeling language).

Les exemples présentés sont réalisés avec StarUML, qui est un logiciel libre. La société AddQ éditrice du composant logiciel G# (ensemble dédié à la programmation orienté objet en licence libre) fournit un module pour transformer les diagrammes de classe StarUML en bibliothèques objet LabView.

REPRESENTATION DES CLASSES

La classe est représentée par un **rectangle** (un classeur en vocabulaire UML) dans le diagramme de classe. Tout classeur non explicitement stéréotypé autrement **est une classe**. Ce rectangle contient trois champs (il peut y en avoir jusqu'à cinq), de haut en bas : le nom de la classe, ses attributs, ses méthodes.

VISIBILITE DES ATTRIBUTS ET DES METHODES

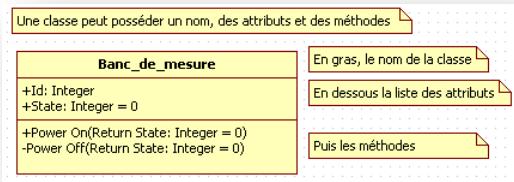


Figure 12-1 Visibilité des attributs et des méthodes

Devant les méthodes et les attributs, on trouve les signes suivants :

+ : **Publique** : Toutes les autres classes ont accès à cet attribut.

: **Protégé** : La classe et ses classes filles ont accès à cet attribut.

- : **Privé** : Seule la classe elle-même a accès à cet attribut

~ : **Paquetage** : Les classes appartenant au même paquetage que la classe elle-même ont accès à cet attribut.

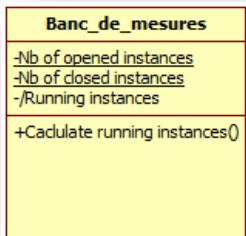
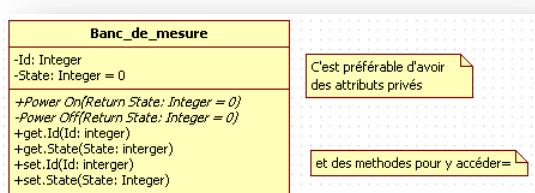


Figure 12-2 Attributs de classe et attributs dérivés

Les attributs sont propres à chaque instance de la classe (chaque objet). Il est parfois utile de définir des attributs propres à la classe et qui seront partagés par toutes ses instances (elles peuvent y accéder mais n'en possèdent pas de copies). Par exemple un compteur qui donne le nombre d'objets qui ont été instanciés. C'est ce qu'UML appelle un **attribut de classe**. Il est symbolisé par le **soulignement du nom** de l'attribut.

Certains attributs sont calculés à partir d'autres attributs, ils sont dits **dérivé** et sont symbolisé par un "/" devant leur nom.

Il est préférable, pour respecter le principe d'encapsulation, que **tous les attributs soient privés**. On crée alors des méthodes de lecture et d'écriture de ces attributs (accesseurs et mutateurs vont permettre le contrôle de validité des valeurs de ces attributs)



Les méthodes get .xxx et set.xxx sont en **caractère droit** car elles sont **implémentées** au niveau de la classe (leur code est écrit dans le corps des fonctions et a priori ne dépend pas du type de banc de mesure utilisé).

Les méthodes Power.xxx sont écrites en **italique**, cela signifie que ce sont des **méthodes abstraites** (elles ont un nom, une liste de paramètres mais pas de code). Elles devront être surchargées par des méthodes implémentées dans des classes dérivées (même noms et paramètres).

Figure 12-3 accesseurs et mutateurs permettent d'accéder aux valeurs des attributs privés

Les classes peuvent aussi être abstraites elles ne comportent alors que des méthodes abstraites, elles permettent d "unifier" l'appel aux classes filles. Elles servent d'interface entre l'appel et l'implémentation d'où leur nom d'**interface**

REPRESENTATION DES DEPENDENCES

Si nous faisons de la programmation objet, nous avons un diagramme de classe qui implémente plusieurs classes avec des relations entre elles. Ces relations sont nombreuses en UML, mais il faut en retenir trois types : la filiation la relation et l'association et l'agrégation/composition.

La filiation repose sur le concept : est une sorte de
 <un Homme est une sorte d'animal>

L'association exprime une connexion sémantique
 <un Homme travail pour une entreprise>

L'agrégation/composition exprime une relation contenant/contenu
 <un Homme a deux jambes>

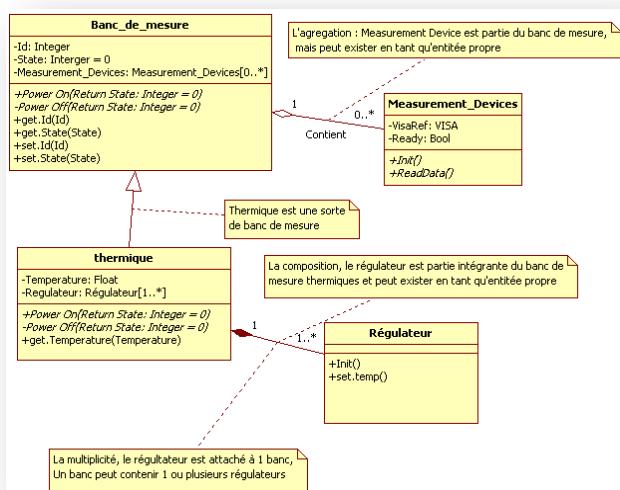


Figure 12-4 Notre modèle UML de bancs de mesures

La distinction sémantique entre ces relations (être ou avoir) est parfois difficile à cerner et donc à choisir : pour faire simple, héritage quand on peut appliquer la relation « est-un » et qu'on veut faire du polymorphisme, et délégation quand un composant peut changer durant la vie de l'objet ou du programme.

Si nous implementons une classe fille (donc plus spécialisée) de notre classe **Banc_de_mesure**, par exemple : **Banc_de_mesures_thermiques** la relation de filiation sera représentée par une flèche dirigée de la classe dérivée vers la classe mère.

Cette relation de filiation traduit l'idée que

Thermique EST UNE SORTE de Banc_de_mesure. Elle en possède tous les attributs et les méthodes. Elle surcharge les méthodes Power_On et Power_Off pour les adapter à ce banc en particulier. La température étant un élément indissociable d'un banc de mesures thermiques, la classe s'enrichie d'un attribut Température et de son assesseur.

Mais notre banc de mesure est constitué d'autres choses, par exemple de multimètres qui seront des enfants de la classe « **Appareils de mesure** ». Ces derniers sont partie de **Banc_de_mesure**, et y sont reliés, dans cet exemple, par une relation d'agrégation symbolisée par le losange vide. Une classe **Régulateur** apparaît comme un composant indissociable de **Thermique**, cette relation de composition est symbolisée par le losange plein. Aux intersections on trouve les cardinalités de la relation, **thermique** possède au moins 1 régulateur (symbole **1..***), et un régulateur est associé à un et un seul banc thermique.

UML relève de la conception et non de l'implémentation, en revanche l'implémentation va directement dépendre de cette conception et des erreurs qui y seront faites...

LABVIEW OBJECT-ORIENTED PROGRAMMING: THE DECISIONS BEHIND THE DESIGN.

Les ingénieurs de chez NI ont choisi certaines options d'implémentation décrites dans un document qui porte le nom de ce chapitre et est disponible sur le site de NI. Il est important dans rappeler les raisons et d'expliquer qu'elles en sont les conséquences.

LES CHOIX

NI a retenu deux maîtres mots dans sa conception de LVOOP: l'**encapsulation** et l'**héritage**.

L'ENCAPSULATION

LabView est un langage de **flux de données** implicitement relié à un passage de **paramètres par valeurs** qui permet un parallélisme naturel du code. Cette vision est celle retenue pour les objets. Elle présente l'avantage de l'efficacité, d'éviter les violations mémoire. Le partage d'objets entre différentes parties de code est cependant difficile. Un « **Data Value References** » introduit avec la version 2009 permet un passage par référence.

Les mécanismes **d'encapsulation** retenus sont les suivants : les **attributs** seront contenus dans un agrégat (Cluster) qui ne pourra être éclaté que par les Vis de la même classe ils sont donc tous **privés**. Ceci implique l'utilisation d'**accesseurs** et **mutateurs** pour tous les attributs (heureusement une procédure de création automatique existe). Les **méthodes** d'une classe ont quatre droits d'accès : "**privé**," "**protégé**," "**publique**" et "**communauté**". Une **classe** est une **librairie** particulière qui contient l'agrégat des attributs et les méthodes de la classe.

L'HERITAGE

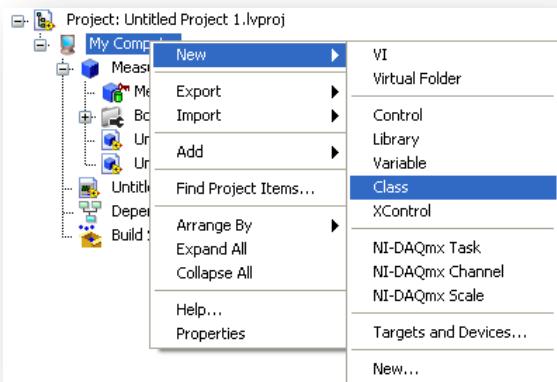
Le mécanisme **d'héritage** est basé sur la notion de classe primitive « **LabVIEW Object class** » qui est l'ancêtre de toutes autres classes. Ceci permet d'avoir un type commun à toutes les classes et de les gérer d'une façon homogène (par exemple de mettre dans un tableau des classes différentes). Le fait que toute classe dérive de cette classe mère implique qu'il n'existe pas de **constructeur** (New.xxx) pour instancier les objets à partir d'une classe. De même, le fait que LabView décide de libérer la mémoire occupée par quoique ce soit (le fameux ramasse miettes ou garbage collector), lorsqu'il n'est plus utilisé libère le programmeur de l'emploi de **destructeurs**. Cependant l'utilisation de références via « **Data Value References** » annule ce principe (puisque l'on vient de créer de façon explicite une réservation mémoire). LabView ne supporte pas d'héritage multiple (C hérite de A & B) il faut dire que l'héritage multiple pose autant de problèmes qu'il n'en résout (conflits de noms, héritages en diamants D hérite de C&B qui héritaient de A, référence circulaires...) et il est toujours possible de s'en passer (principe de délégation d'opération par une classe à une autre classe).

CREATION D'OBJETS

Pour l'ensemble des exemples et exercices nous reprenons le diagramme de classe précédent.

La création d'objets LabView passe par la création de bibliothèques spécifiques dont l'extension est **.lvclass**

LES BIBLIOTHEQUES



La création de la bibliothèque se fait par « **New->Class** » dans l'explorateur de projet.

Ceci à pour effet de créer une arborescence contenant un **.ctl** (ce n'est pas un fichier sur disque mais une partie de la bibliothèque de classe créée).



La clef rouge de l'icône indique que le contenu est privé, le cylindre vert qu'il s'agit de données.

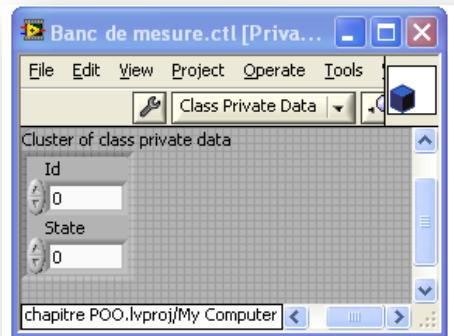
Figure 12-5 Création d'une classe



Exercice 12-1 Créez la classe Banc_de_mesures

LES PROPRIETES (ATTRIBUTS)

Elles sont donc contenues dans le **.ctl** (pour contrôle) qui est un cluster. Il faut en ouvrir la face avant et y placer les éléments qui sont les données de la classe. Par exemple la classe banc de mesure :



Id → identifie le banc

State → identifie son état

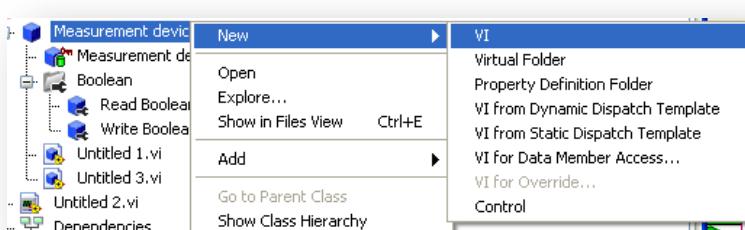
Rappelons que ces données sont privées et donc ne sont accessibles (via une fonction désassembler par nom) qu'aux VI faisant partie de la bibliothèque de cette classe.

Des accesseurs et des mutateurs devront être définis pour accéder à ces attributs depuis une autre classe.



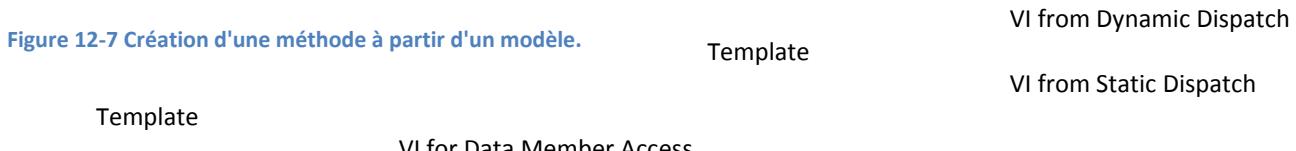
Exercice 12-2 Créez attributs de la classe

LES METHODES



Les méthodes sont des VI de la classe, elles sont créées dans le menu local de la classe :

Il est possible de créer des méthodes entièrement à la main, mais trois modèles sont là pour vous faciliter la tâche, il s'agit de :



Ces trois modèles vont créer des VIs contenant une entrée et une sortie d'erreur, une entrée et une sortie de donnée. Le code contient une structure de choix connectée à l'entrée d'erreur.

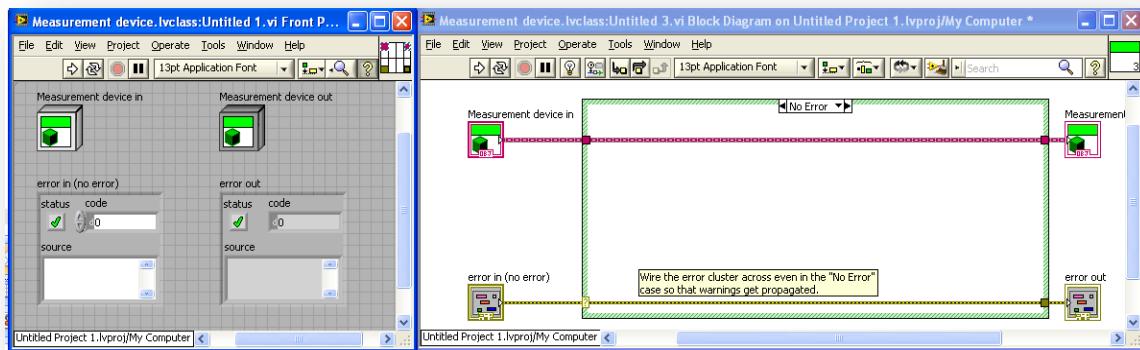


Figure 12-8 Exemple de modèle créé

Le concepteur « n'a plus qu'à » inclure son code dans la structure de choix.

DYNAMIQUE VERSUS STATIQUE

Les deux premiers modèles sont identiques à l'option prêt que l'entrée et la sortie de la connexion de classe est ou n'est pas « **Dynamic Dispatch** ».

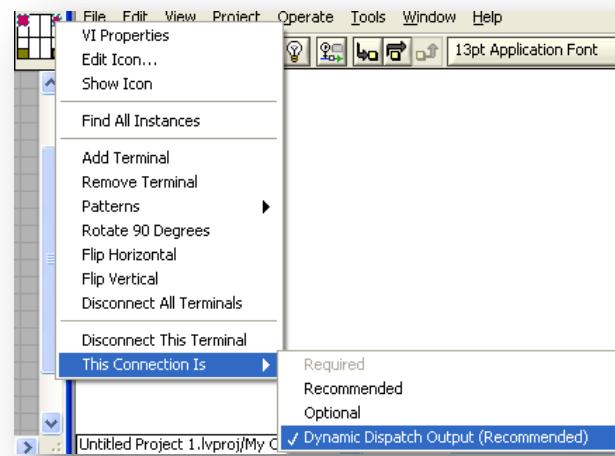


Figure 12-9 Dynamic Dispatch du fil de connection de la classe

Nous avons vu que le polymorphisme était un élément clef de la programmation objet.

Cela signifie qu'une méthode de nom XXX sera choisie au moment de l'exécution parmi toutes les méthodes XXX de la hiérarchie de la classe, en fonction du type de la classe connectée à son entrée.

Par exemple la classe « **Banc de mesure** » possède une méthode « **Power_On** » (qui ne fait rien, elle est abstraite), ses filles de classes « **Thermique** » et « **Optique** » possèdent-elles aussi une méthode « **Power_On** » (implémentées réellement et différentes). Pour que, à l'exécution, la bonne

méthode « **Init** » soit choisie, il faut que l'entrée de connexion de classe au VI ait ce côté « dynamique ». A contrario, si la méthode est implémentée sans activer l'appel dynamique, c'est nécessairement la méthode de la classe qui sera appelée (le câblage sera refusé !). Attention, tout ce qui facilite la vie peut avoir un revers, en

l'occurrence, le fait de devoir choisir à l'exécution la fonction appelée, ralenti sérieusement l'exécution du code.

ACCESSEURS ET MUTATEURS

Nous l'avons dit, les données sont nécessairement privées, il faudra une méthode publique pour accéder aux données de la classe en dehors de la librairie de cette classe. Le modèle « **VI for Data Member Access** » permet de créer ces méthodes.

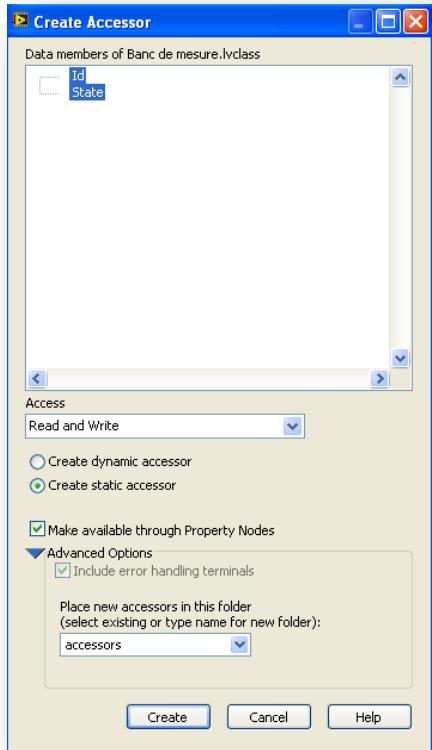


Figure 12-10 Création automatique d'accesseurs et de mutateurs

La fenêtre de création des accesseurs et mutateurs apparaît. Elle contient une liste de différentes propriétés de l'objet. Le type de VI à créer (Accesseurs et/ou Mutateurs), le mode d'accès à la méthode (Dynamique ou Statique). La coche autorise l'accès aux données via un nœud de propriété (qui appellera le VI créé). Il est possible de spécifier un répertoire virtuel du projet ou mettre les Vis ainsi créés. Les Vis ressemblent à :

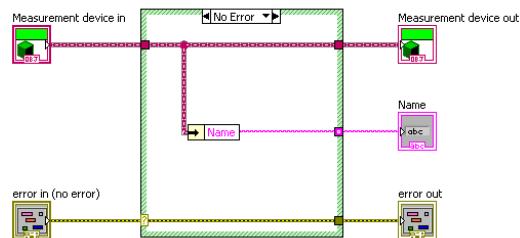


Figure 12-11 VI d'accès à une propriété

L'utilisation de l'option "Make available through Property Nodes" peut poser quelques problèmes pour accéder à des éléments particuliers d'une collection (Tableaux ou Agrégats)

car il n'est pas possible de lire ou d'écrire plus d'un paramètre. Il est donc impossible, par exemple de lire l'élément x du tableau T[] puisque cela fait 2 paramètres, vous ne pourrez que lire le tableau entier.

Exercice 12-3 Créez les accesseurs et mutateurs aux attributs de la classe **Banc_de_mesures**



Figure 12-12 Les accesseurs / mutateurs

Les accesseurs et mutateurs se retrouvent dans l'arborescence de la classe, la clef plate noire signale que ces valeurs sont accessibles via un nœud de propriété. Le nom de l'item de la propriété sera le nom de dossier virtuel dans lequel les accesseurs/mutateurs seront rangés.

ACCES AUX ATTRIBUTS

La classe « **Banc_de_mesures** » possède maintenant des méthodes d'accès à ses attributs. Le diagramme suivant les initialise.

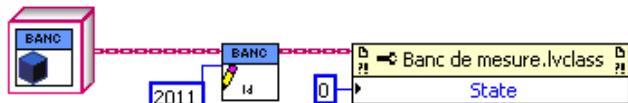


Figure 12-13 Accès aux valeurs des attributs

L'instance de la classe est créée par un glisser/déposer de la classe, du projet au diagramme.
L'**Id** est initialisé via l'appel direct à l'accesseur, **State** en passant par un nœud de propriétés.

DROITS D'ACCES AUX METHODES

Les méthodes peuvent avoir quatre droits d'accès :

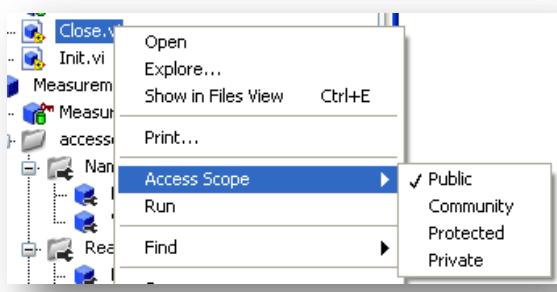


Figure 12-14 niveaux d'accès aux méthodes

- Publique** : tout VI peut appeler la méthode
- Communauté** : seuls les VI de la classe, ou d'une classe amie ont accès à la méthode, cela ressemble à la visibilité "UML Package" (clé bleue)
- Protégé** : seuls les Vis de la classe et ses descendants ont accès aux méthodes (clé jaune).
- Privé** : seuls les Vis de la classe ont accès (clé rouge).

Les amis

La notion de fonctions amies ou de classes amies mérite d'être précisée. Normalement, une méthode est accessible, soit à tous (publique), soit à sa hiérarchie (protégée, privée). La notion de classe amie (issue du C++), pouvant donc accéder aux attributs d'une autre classe sans passer par les accesseurs et mutateurs est

une facilité (au détriment de la sécurité) utilisée pour une meilleure efficacité. L'héritage multiple étant impossible, on trouvera souvent une délégation de classe (la classe A fait des choses pour la classe B, mais ne sont pas parentes), il est pratique dans ce cas de déclarer la classe A amie de la classe B. La notion d'amie et à rapprocher de la visibilité « ~package » d'UML.

Les amitiés se nouent dans les propriétés de la classe : elles portent sur un VI, une classe ou une librairie.

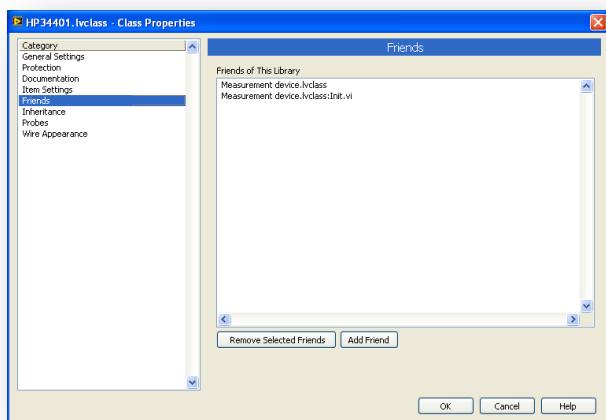


Figure 12-15 Fenêtre de configuration des classes amies (Community Scope)

Exercice 12-4 Créez les méthodes **Power_On** et **Power_Off** de la classe **Banc_de_mesure**. Elles ont respectivement les portées **Publique** (le banc peut être démarré par un VI quelconque) et **Protégée** (Le banc doit suivre une procédure d'arrêt), **les méthodes devront être surchargées** par les classes filles.

HIERARCHISATION DES CLASSES

Actuellement nous n'avons qu'une classe « **Banc_de_mesures** » ajoutons au projet une autre classe, « **Thermique** ».

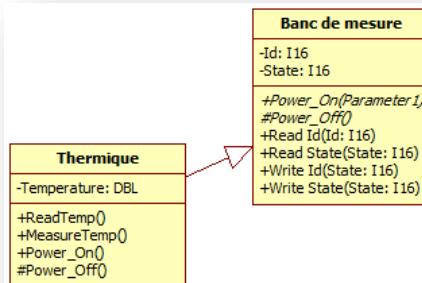


Figure 12-16 Diagramme de classe avec la classe dérivée Thermique

Exercice 12-5 Créez la nouvelle classe : Thermique. Elle possède un attribut privé : Température .Puis créez l'accesseurs une méthode de mesure (qui renvoie un nombre aléatoire par exemple)

CHANGEMENT DE L'HERITAGE

Cette classe doit être fille de la classe mère « **Banc_de_mesures** », pour définir cet héritage, il faut ouvrir le panneau de propriété de chacune des futures classes filles et aller dans héritage.

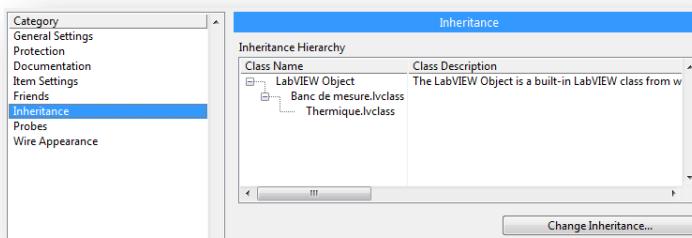
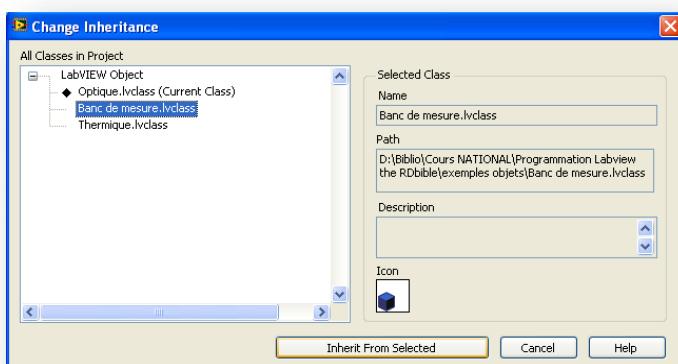


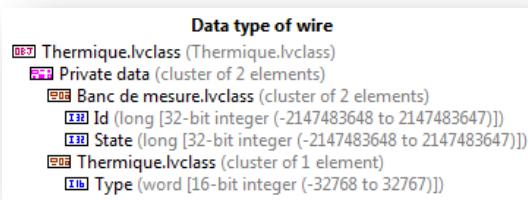
Figure 12-17 Propriétés de l'héritage

« **Optique** » hérite par défaut de la classe « « **LabView Object** » (la classe supérieur appelée souvent « **Méta classe** »), le bouton « **Change Inheritance** » permet de changer l'héritage.



En sélectionnant « **Banc_de_mesures** », puis en cliquant sur « **Inherit From Selected** », « **Thermique** » hérite alors de « **Banc_de_mesures** », cela signifie que les attributs et les méthodes de la classe mère sont transférés à la classe fille.

Figure 12-18 Changement de l'héritage



Les données véhiculées par le fil sont dans un agrégat nommé « **Thermique.lvclass** ».

Il contient deux agrégats correspondant aux attributs de la classe mère et de la classe fille.

Figure 12-19 Données véhiculées par la classe Optique

EXPLORATEUR DE HIERARCHIE

L'explorateur de hiérarchies (**View->LabView Class Hierarchy**) permet de visualiser l'arborescence des classes ainsi réalisées, car elle n'apparaît pas dans l'explorateur de projets.

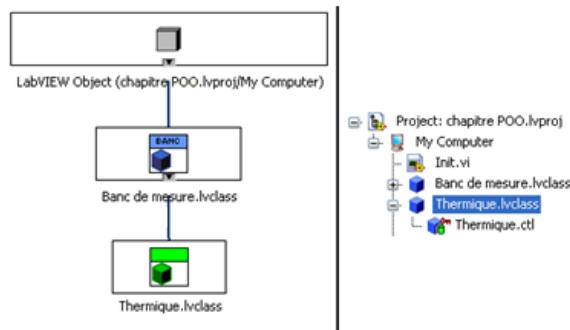


Figure 12-20 Arborescence des classes

Exercice 12-6 Changez l'héritage de la classe thermique pour qu'elle dérive de la classe banc de mesures. Vérifiez avec l'explorateur de hiérarchies les liens de fratrie.

OPTIONS DE SURCHARGE DES METHODES

SURCHARGE OBLIGATOIRE DES METHODES FILLES

Nous avons créé deux méthodes « **Power_On** » et « **Power_Off** » pour la classe « **Banc_de_mesures** ». Nous savons que ces méthodes doivent être surchargées par des méthodes spécifiques à chaque banc de mesure.

Elles définissent ce que l'on appelle des « **méthodes abstraites** » .

Elles spécifient les paramètres passés (nombre et type), dans notre cas aucun hormis la classe et l'erreur en entrées et en sorties. Elle n'implémente aucun code spécifique.

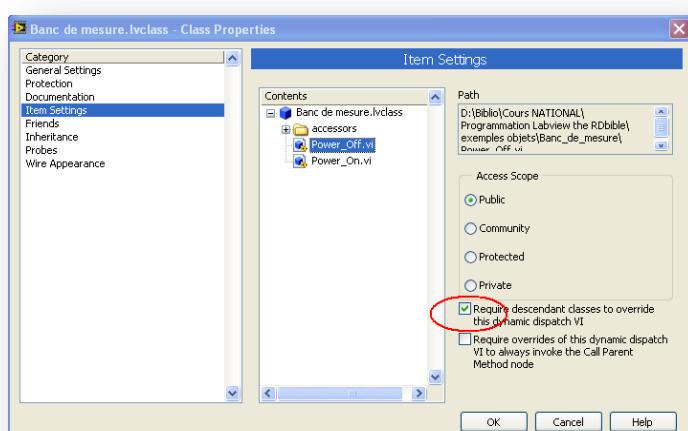


Figure 12-21 Comment obliger la surcharge d'une méthode

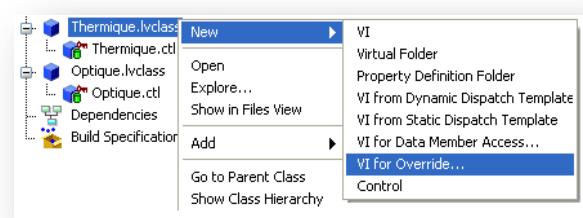
Il faut obliger les classes filles à surcharger cette méthode.

En effet, si ce n'était pas le cas, on utiliserait des méthodes qui ne font rien. Pour cela, dans les propriétés de la classe, catégorie « **Item setting** » cochez la case « **Require descendant classes to override this dynamic dispatch VI** ».

Si l'on câble une référence de classe ou dépose un VI de la fratrie « **Banc_de_mesures** », une erreur de compilation apparaît :

This class does not implement at least one dynamic dispatch member VI marked in an ancestor class as a required override.

Figure 12-22 message d'erreur si la surcharge est obligatoire et non respectée dans les classes filles



Un modèle existe pour créer des Vis de surcharge. Dans la classe ou vous voulez créer le VI, sélectionnez « **New->VI for Override** ». L'ensemble des méthodes parentes apparaissent, l'astérisque marque celles qui doivent être surchargées.

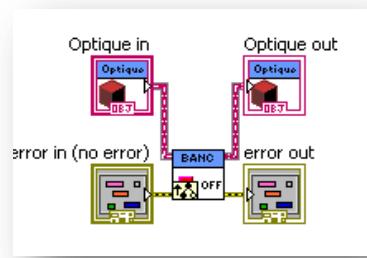


Figure 12-23 Crédit de surcharges

Une icône à l'entête de la classe et avec le corps de la méthode parente est créée. Il est bien entendu nécessaire de créer toutes les méthodes de toutes les filles de la classe pour que la flèche n'indique plus d'erreur de compilation.

Le code d'initialisation correspondant à l'appareil sera écrit dans la structure de sélection.

Exercice 12-7 Créez les méthodes **Power_On** et **Power_Off** de la classe **Thermique**. Elle peut, par exemple, simplement affecter la valeur 0 à **State** pour indiquer que le banc est Off et 1 pour On.



Dans ce cas, LabView recopie les noms, icônes et paramètres des méthodes de la classe parente et y pose un appel à la méthode de la classe parente (Cf § APPEL DES METHODES PARENTES PAR LES METHODES SURCHARGEES). Ceci permet d'obtenir des VIs fonctionnels, mais dont il faut tout de même reprendre le code puisque l'on avait décidé de créer des surcharges, une bizarrie qui perdure depuis la version 2009 !

Figure 12-24 Le VI de surcharge créé

UTILISATION DES METHODES SURCHARGEES.

Supposons que nous voulions démarrer deux bancs de mesures, l'un de mesure Optique, l'autre de mesures

Thermiques. Très classiquement, nous utiliserons le VI Power_On d'Optique puis le VI Power_On de Thermique. La surcharge de méthode nous permet maintenant d'utiliser la même méthode pour nos deux bancs, c'est le programme qui ira chercher dans l'arborescence des

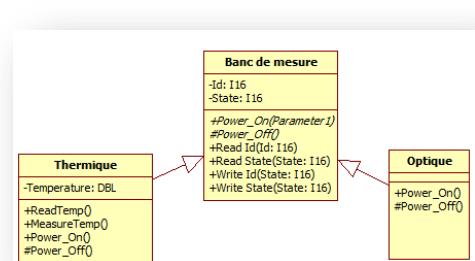
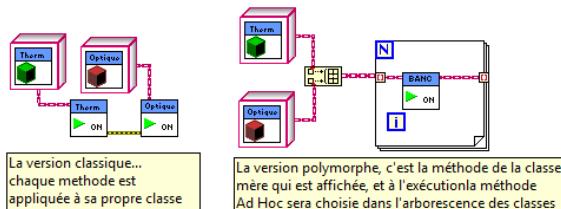


Figure 12-25 Deux classes filles surchargent les méthodes de démarrage et d'arrêt

objets la méthode Ad Hoc. Soit à la compilation (s'il ne peut y avoir de doute, c'est le cas, par exemple si on passe un seul objet), soit à l'exécution (si par exemple on passe un tableau d'objets)

Exercice 12-8 Créez une classe **Optique** qui hérite de **Banc_De_Mesures**. Cette classe surcharge les méthodes **Power_On** et **power_Off**. Dans un VI Init.vi créez un tableau contenant un objet de type **Thermique** et un objet de type **Optique**. Utilisez la méthode **Power_On** dans une boucle pour activer ces bancs. Vérifiez que l'attribut **State** est bien passé à 1



APPEL DES METHODES PARENTES PAR LES METHODES SURCHARGEES

A l'inverse du cas précédent, parfois le concepteur veut que la méthode parente soit nécessairement appelée, parce qu'elle assure des fonctions précises qui ne doivent pas être modifiées par les enfants de la classe. Dans ce cas, soit les filles ne possèdent pas de surcharge de cette méthode, (c'est le plus simple, mais difficile à assurer et à maintenir dans un développement très important), soit on les oblige à appeler la méthode de la classe mère.

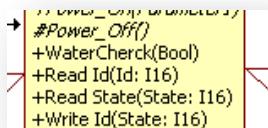


Figure 12-26 La méthode WaterCheck

d'une méthode)

Imaginons que **tous** nos bancs de mesures soient refroidis par eau, une méthode « **WaterCheck** » soit présente dans la classe parente. Elle doit nécessairement être appelée, même si des filles l'ont surchargée. Pour cela, il faut cocher la case « **Require overrides of this dynamic dispatch VI to always invoke the Call Parent Node** » dans le panneau de propriétés de la classe « **Banc de mesure** » (cf. Figure 12-21 Comment obliger la surcharge

Si maintenant une fille de « **Banc de mesure** », par exemple « **Thermique** » implémente elle aussi une méthode « **WaterCheck** » un conflit doit interdire l'exécution du code et l'on doit voir un message d'erreur :

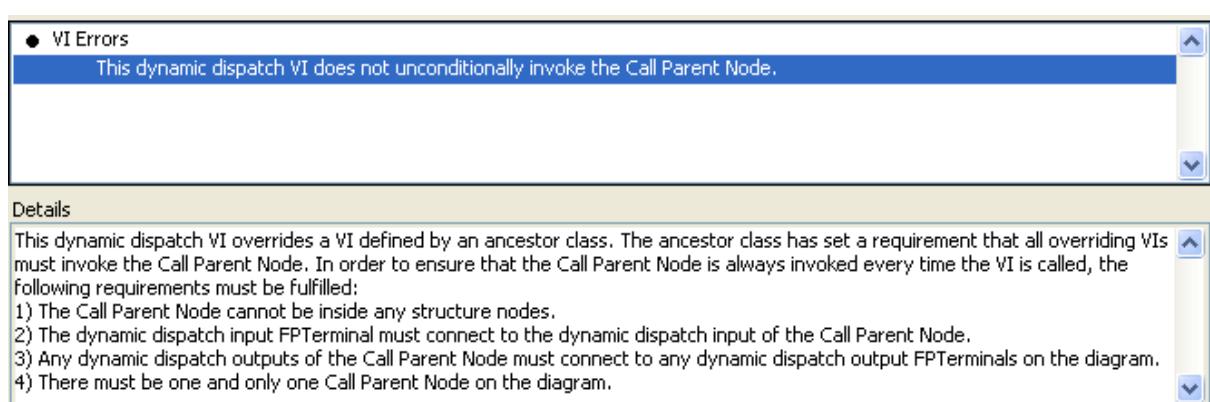
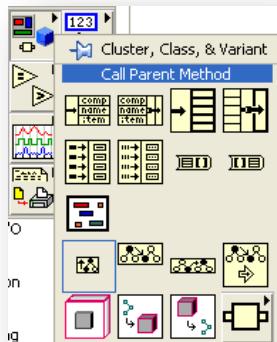


Figure 12-27 Message d'erreur généré si une méthode de surcharge n'appelle pas la méthode parente.

 Exercice 12-9 Créez une méthode « **WaterCheck** » de la classe **Banc_De_Mesures** qui revoie un booléen toujours Vrai. Posez cette méthode dans le code **Init.vi** précédent et vérifiez que la flèche d'exécution n'est pas cassée. Puis créez une autre méthode « **WaterCheck** » de la classe « **Thermique** » (même nom et connecteur). Vérifiez que maintenant la flèche d'exécution est cassée.

Ceci oblige le concepteur d'une surcharge à utiliser le VI « **Call Parent Method** » de la palette « **Cluster, Class & Variant** ».



Lorsque l'on dépose cette fonction dans une méthode d'une classe fille, LabView va rechercher dans les méthodes de sa mère une méthode de même nom et va l'appeler.

Son icône ressemble alors à : . C'est l'icône de la méthode mère avec le  indiquant une remontée dans les hiérarchies.

Si le compilateur ne trouve pas la méthode parente, et s'il ne peut s'assurer qu'elle sera exécutée quoi qu'il arrive (par exemple si elle est dans une structure de choix), une erreur apparaittra.

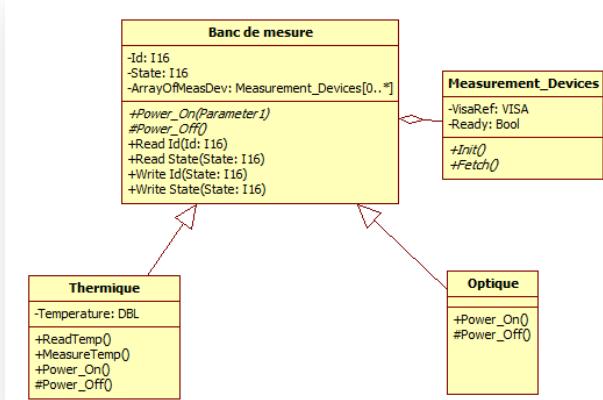
12-28 La palette Class

 Exercice 12-10 Modifiez la méthode « **WaterCheck** » de la classe « **Thermique** » en utilisant « **Call Parent Method** ». Vérifiez que maintenant la flèche d'exécution n'est plus cassée.

IMPLEMENTATION D'UNE RELATION D'AGREGATION/COMPOSITION

Les relations de filiations sont supportée directement par LabView, les relations d'agrégation ou de

composition ne le sont pas (comme dans beaucoup de langages). Il faut les implémenter en créant un conteneur, de références aux classes contenues, dans les propriétés de la classe contenante. Ce peut être : des références simples, des tableaux, des listes, des tables de hachage, des arbres... ou tout autre conteneur cela dépendra des cardinalités, de la notion de tri des références...



12-29 L'association des classes se fait par l'ajout d'un tableau de références aux classes pointées

Dans notre cas, notre banc pouvant contenir plusieurs appareils de mesures, le plus simple est d'utiliser un tableau de références de classes « **Measurement_Devices** » dans la liste des propriétés de « **banc de mesures** ». LabView nous offrant des tableaux dynamiques, il n'est pas nécessaire de prévoir quelque chose de plus sophistiqué. La subtile nuance UML pose entre agrégation et composition n'a pas de sens en LabView car comme il n'existe pas de constructeurs n'y de destructeurs, la durée de vie du contenant par rapport au contenu n'a pas de sens elle non plus.

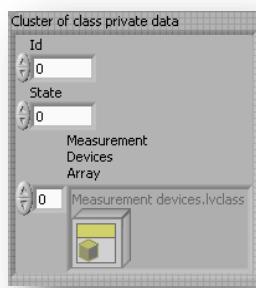


Figure 12-30 Un tableau de références est ajouté pour assurer la relation d'agrégation

Exercice 12-11 Après avoir créé une classe « **Measurement_Devices** » possédant les deux propriétés **VisaRef** et **Ready**, ainsi que les deux méthodes abstraites avec surcharge obligatoire **Init()** et **Fetch()**, modifiez la classe « **Banc de mesure** » pour qu'elle puisse supporter la relation d'agrégation avec la classe « **Measurement_Devices** »

CREATION DYNAMIQUE D'OBJETS

Pour le moment, notre banc de mesure ne possède que deux types d'appareils, des multimètres soit HP3458 (une référence!), soit HP34401 (un modèle économique). Le nombre et le type de ces appareils dépend du type de mesures effectuées et n'est pas, à priori connu lors de la compilation du code (heureusement il est connu lors de l'exécution...). Nous allons voir comment instancier dynamiquement des appareils pour le banc de mesure thermiques.

Exercice 12-12 Tout d'abord, créez les classes **HP3458** et **HP34401**. Désignez-les comme filles de « **Measurement_Devices** ». Créez les méthodes de surcharge **Init()** et **Fetch()**, **Init** met le flag **Ready** à Vrai et **Fetch** renvoie un nombre aléatoire. Enfin, créez une méthode **AddMeasurementDevice** de la classe **Banc_De_Mesure**. Cette méthode reçoit une variable **NewDevice** de type **Measurement_Devices** et l'ajoute au tableau **MeasurementDevicesArray**.

Si le banc comporte un multimètre de chaque type, le code suivant permet de configurer notre banc:

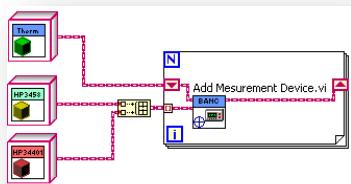


Figure 12-31 Version statique de la configuration

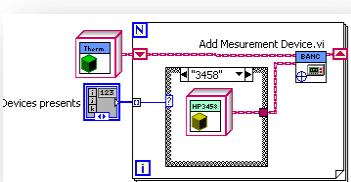


Figure 12-32 Un peu plus générique

Mais cette version est statique, supposons qu'en face avant, un tableau d'énumérations contienne les différents équipements présents (par exemple, ils viendraient d'un fichier de configuration) Notre code ne peut plus convenir !

La version suivante est moins « statique », mais elle impose tout de même de connaître au préalable les types de multimètres disponibles. Si nous désirons obtenir une programmation vraiment générique, il faut pouvoir remplacer à tout moment un multimètre par un autre, même s'il n'a pas été prévu lors de la conception du code.

Pour cela il est nécessaire de charger l'objet en mémoire à partir du fichier .lvclass. C'est ce que fait la fonction "get LV class default value". Cette fonction renvoie un objet de la classe « **LabView**

Object » c'est-à-dire du plus haut niveau dans la hiérarchie des objets. Pour utiliser les méthodes propres aux appareils de mesure, il faut spécialiser cet objet vers la classe « **Measurement_Device** ». La fonction « **To More Specific Class** » permet le « typecast » de type « **LabView Object** » vers « **Measurement_Device** ». Cette fonction renvoie une erreur si le type d'objet entrant n'est pas parent du type cible. Il existe bien entendu une fonction « **To More Generic Class** » pour remonter dans la hiérarchie de classe.

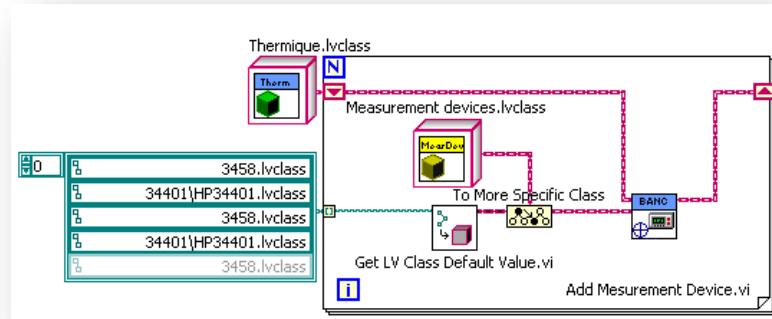


Figure 12-33 I have a dream that one day modularity will be easy

Exercice 12-13 Instanciez statiquement puis dynamiquement quelques multimètres puis après les avoir initialisés lisez leurs données.

TRANSFERT D'OBLIGATION DE SURCHARGE DES METHODES A DES PETITS ENFANTS

Si dans notre arborescence, nous avions interposé entre « **Measurement Devices** » et « **HP3458** » une classe « **DMM** », cette dernière aurait été obligée de surcharger les méthodes **Init()** et **Fetch()**, puisque la surcharge a été rendue obligatoire dans la classe « **Measurement Devices** ».

Hors la classe « **DMM** » ne peut initialiser les multimètres puisqu'elle n'en connaît pas le type et donc les mots de programmation. Dans ce cas la méthode « **Init** » serait elle aussi une méthode abstraite, vide.

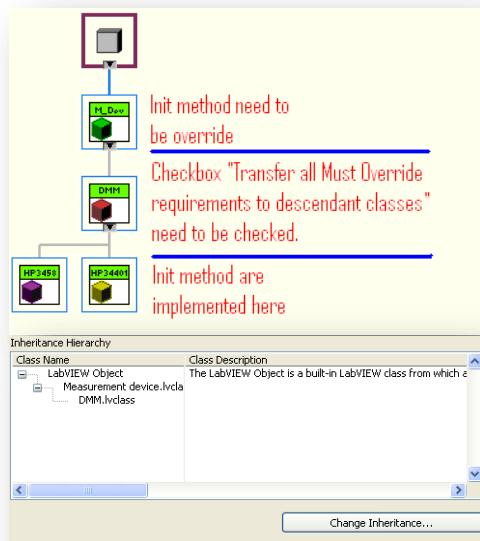


Figure 12-34 Transfert de l'obligation de surcharge

Il en serait de même pour des classes « **Generator** », « **Power supply** »....

Pour éviter de créer ces méthodes abstraites dans toute la hiérarchie, il faut cocher la case « **Transfer all Must Override requirements to descendant classes** » dans les propriétés d'héritage de la classe « **DMM** ».

Le compilateur vérifie alors que les méthodes de la classe mère qui doivent être surchargées le sont effectivement, mais plus loin dans la hiérarchie.

Exercice 12-14 Créez une classe **DMM** dans la hiérarchie et transférez l'obligation de surcharge des méthodes **Init** et **Fetch** aux descendants.

UN PEU PLUS LOIN

OPTIMISATION DU CODE

Lorsque vous créez des méthodes « **dynamic dispatch methods** », chaque classe enfant hérite de toutes les méthodes publiques et protégés définies sur la classe parent. La classe enfant peut surcharger, ou étendre, ces VIs de membres avec leurs propres versions. Lors de l'appel de la méthode, LabView ne sait pas quelle version de la méthode il invoquera jusqu'au moment de l'exécution. LabView optimise l'allocation mémoire en supposant que toutes les méthodes de surcharges se comportent, en terme d'entrée/sorties, comme la méthode de la classe mère. Si des entrées sont constantes dans le parent, LabView suppose qu'elles sont constantes dans tous les enfants. De même si des entrées sont connectées à des sorties, si des entrée ne sont pas modifiées... Dans tous les cas où les méthodes de surcharge ne gèrent pas le flux d'entrée/sorties comme la méthode mère, LabView génère du code pour corriger ce comportement, et compromet l'optimisation.

Pour cela les méthodes de la classe parent doivent être aussi proche que possible, en terme de flux de données, des méthodes de surcharge des enfants. Le moyen le plus sûr d'indiquer au compilateur ce que vont devenir les données est d'utiliser une « **In Place Element Structure** ». Ainsi LabView saura qu'elle entrées sont connectées à qu'elles sorties, qu'elles sont celles qui sont constantes....

Ce qui donne pour la méthode mère :

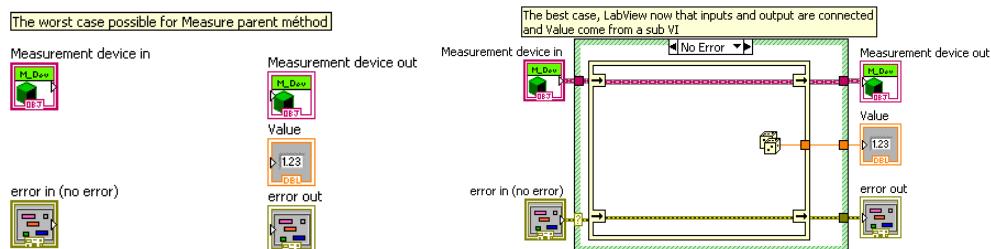


Figure 12-35 Deux implémentations possibles de la méthode abstraite

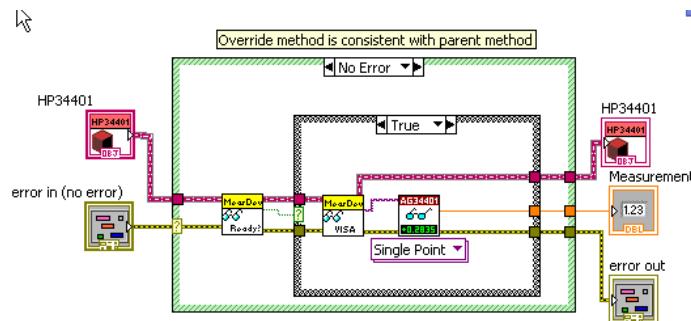


Figure 12-36 L'implémentation de la surcharge avec une structure de flux comparable à la méthode abstraite de droite dans l'illustration précédente.

VARIANT METHODES GENERIQUE

POSITION DU PROBLEME

LabView ne permet pas de créer de véritables méthodes génériques. L'utilisation de Vis polymorphes n'est possible qu'avec des méthodes dont l'entrée de classe est : « **Static Dispatch** ». En effet dans un appel dynamique des méthodes surchargées, le compilateur ne peut obtenir le type de données transmises avant

l'exécution, hors, au moment du câblage il lui faut bien connaître le type de données transmises par un fil (ne serait ce que pour lui attribuer une couleur). Toutes les méthodes ont donc les mêmes terminaux (nombre et type) en entrée et sortie!

Par exemple notre multimètre 34401 possède un buffer de données de 512 mesures, et de 65535 mesures pour le 3458. La méthode « **Fetch** » pourrait avoir une entrée « **Nb Of Readings** » et renvoyer en sortie, soit un scalaire (si l'on demande 1 mesure), soit un tableau (si l'on demande plus d'une mesure, je sais, ce serait plus simple de toujours sortir un tableau !). Ceci n'est pas possible en LabView car, le compilateur doit nécessairement connaître le type de données en sortie avant la compilation. La seule solution pour obtenir une programmation complètement générique est d'utiliser des « **Variants** » comme paramètres transmis.

LES VARIANTS

Le type « Variant » fut introduit par Microsoft (pour les systèmes Windows) dans les années 90 en même

temps que « **Component Object Model** ». Il s'agit d'une union entre un descripteur de type et les données, destinée à simplifier le passage de paramètre entre applications.

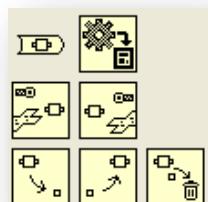


Figure 12-37 La palette variant

LabView possède des « **Variants** » (évidemment pas compatible avec ceux de Microsoft car LabView intègre beaucoup de types de données différents).

L'ensemble des outils est dans la palette « **Cluster, Class & Variant** », « **Variants** »

Les icônes de la première ligne permettent de transformer un type quelconque en variant et de faire l'opération inverse. Celles de la seconde ligne, d'aplatis un variant en un descripteur de type et une chaîne contenant les données et réciproquement (Utile, par exemple pour transmettre des données via TCP.car les Vis n'acceptent que des chaînes). Enfin la dernière ligne permet de nommer des attributs et de leur donner le type désiré. Ceci permet de stocker dans le variant plusieurs variables nommées de type différent, d'en ajouter, d'en enlever... à la façon d'un Cluster dynamique.

Le code d'essai de « **Fetch** » pourrait avoir l'allure suivante : notez bien que dans un cas le variant contient un scalaire, dans l'autre un tableau! Puis à coté la valeur affichée dans l'indicateur de type Variant pour une mesure et dix mesures:

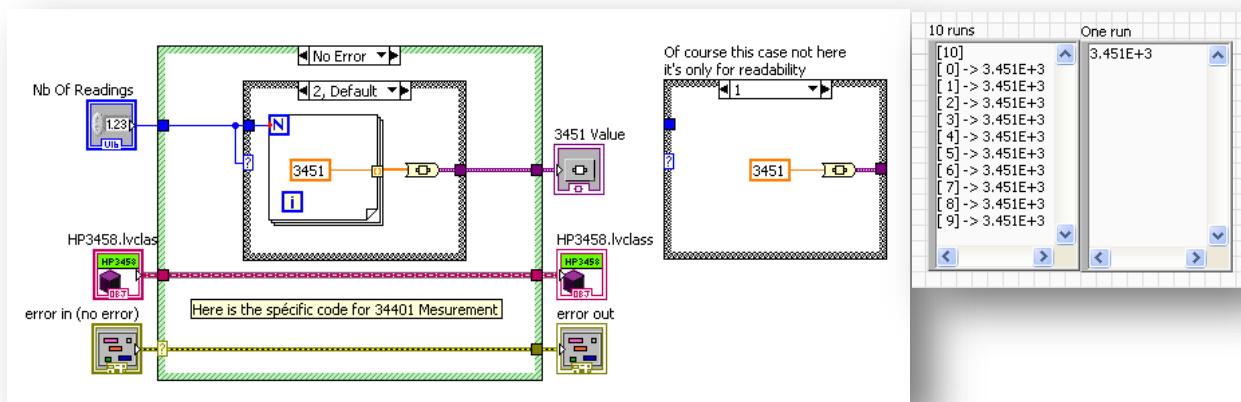


Figure 12-38 Version générique

Attention,
ces
solutions

sont séduisantes, pratiques, mais gourmandes en mémoire et en ressources....A vous de voir.

 Exercice 12-15 modifiez les méthodes **Fetch** pour améliorer la générnicité de vote code comme dans l'exemple précédent.

REFERENCEMENT DES OBJETS

REFERENCES

La version 2009 a introduit une fonctionnalité pour créer des références sur des objets quelconques, ce qui était impossible jusqu'alors. Elles se trouvent dans la palette « **Application Control->Memory Control** ». Il s'agit de « **New Data Value Reference** » et de « **Delete Data Value Reference** ».

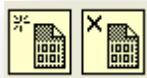


Figure 12-39 VIs de
Création et Destruction
de références

Lors de l'appel de la fonction « **New Data Value Reference** », une référence statique sur la donnée est créée, la ressource mémoire ne sera libérée qu'à l'appel de « **Delete Data Value Reference** ». Ceci est important car en général l'utilisateur de LabView ne se souci pas de ce genre de détails, c'est le ramasse miette qui s'en charge. L'utilisation de référence implique une gestion manuelle de la libération de l'espace mémoire.

L'utilisation de ces références va de pair avec l'utilisation de la structure « **In Place Element Structure** », En effet, le flux de données conduit à un code hautement parallélisable.

Lorsqu'un fil bifurque, c'est une copie de la donnée présente sur ce fil qui copiée, les fonctions utilisant ces données ne peuvent pas les corrompre car ce sont des copies.

Il en est de même si le fil qui bifurque est une référence, ce sont deux copies de la référence qui vont être propagées, excepté qu'elles pointent sur LES MEME DONNEES. Il faut donc un mécanisme de protection qui assure l'exclusivité de l'accès aux données pointées par la référence. C'est précisément ce que fait « **In Place Element Structure** ».

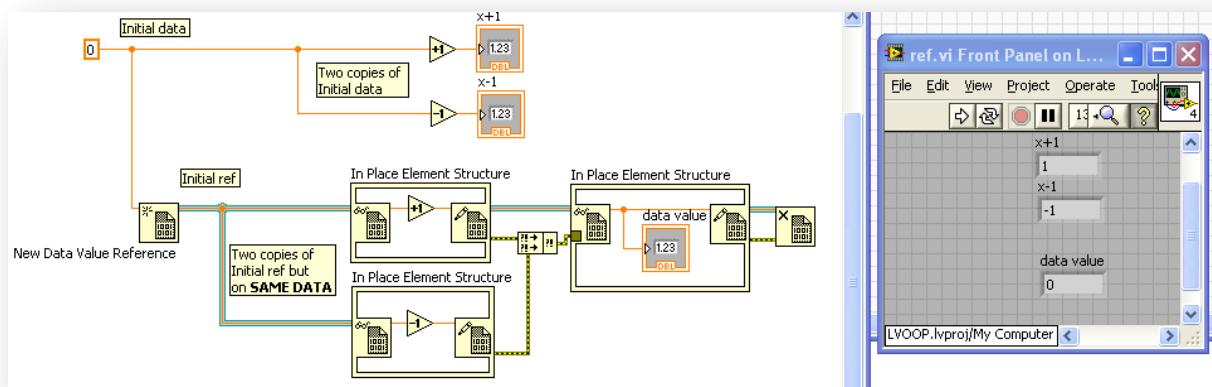


Figure 12-40

Le code de la Figure 12-40 illustre le propos, dans le code du haut, le double est répliqué, et chaque fil vit sa vie. Deux variables distinctes sont créées, l'une incrémentée, l'autre décrémentée. Le code du bas crée une

référence qui sera dupliquée et envoyée aux deux structures « **In Place Element Structure** ». Dans chacune des structures, mais de manière exclusive (sérialisation des accès au sein du VI), les données pointées sont extraites, traitées, mises à jour et libérées. Puisque la référence porte sur le même double, celui-ci sera incrémenté puis décrémenté, ou l'inverse (l'histoire ne le dit pas, il peut même y avoir une corruption mémoire si la référence est utilisée dans un autre VI) et le résultat sera 0.

IN PLACE ELEMENT STRUCTURE

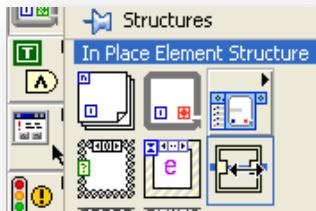
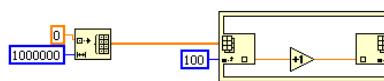


Figure 12-42 In Place Element Structure dans la palette Structures



Figure 12-41 Menu local de la structure "In Place"

Située dans la palette des structures, cet élément est destiné initialement à l'optimisation de la consommation mémoire. Il évite la duplication des données lors du traitement de tableau, d'agrégats, de chaînes et de variants. L'optimisation de code obtenue est toute relative et il ne faut pas penser qu'il s'agit de la panacée (en deux mots, il vaut mieux réfléchir!). Pour faire ces opérations, il convient d'ajouter les éléments spécifiques au type de données à traiter en déroulant le menu contextuel et en sélectionnant « **Add...** ». Par exemple pour changer la valeur d'un élément d'un tableau, sélectionnez, « **Add Array Index / Replace Elements** », le VI suivant ajoute 1 à l'élément 100 du tableau.



L'élément spécifique aux références est : « **Add Data Value Reference Read / Write Element** »

DANS LE CAS DES OBJETS

Pour utiliser les mêmes objets dans plusieurs branches d'exécution parallèles, il faudra donc créer des références sur ces objets et utiliser des structures « **In Place Element Structure** ». Par exemple notre application pourrait lire les multimètres toutes les 100ms et les stocker dans la propriété « **Value** » de la classe « **Measurement Device** ». D'autre part, un thread d'affichage pourrait, toutes les 250 ms vouloir afficher ces données. Le code version presque opérationnelle peut ressembler à :

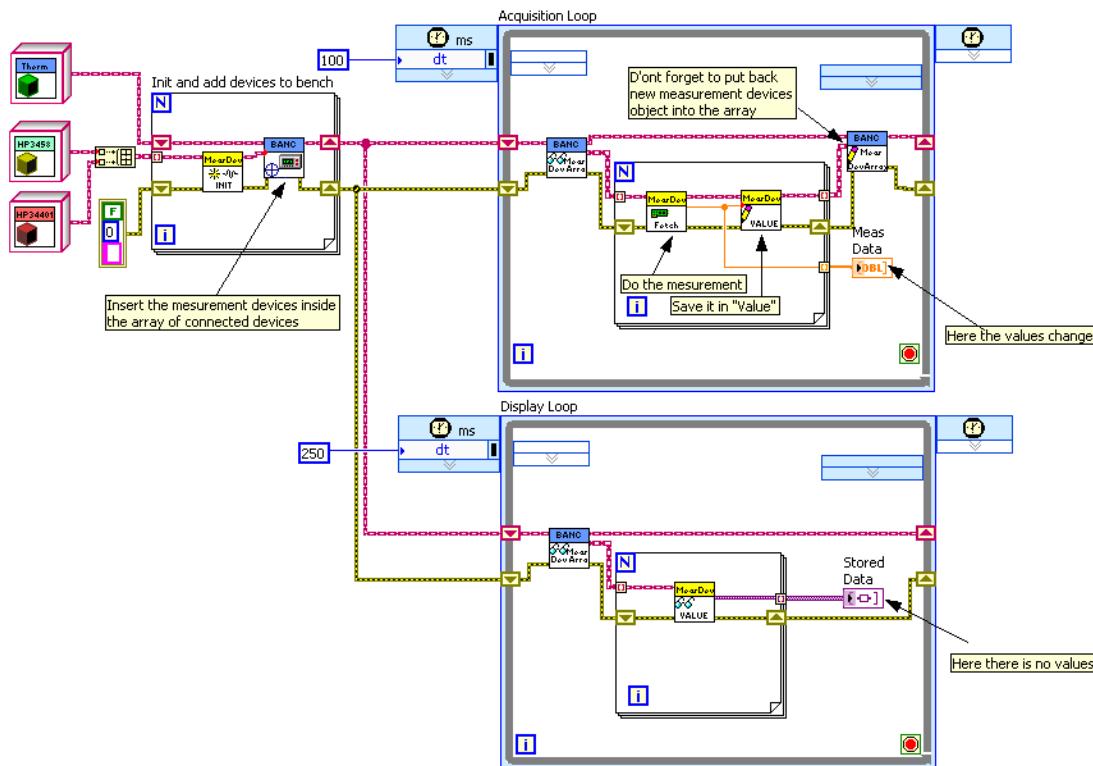


Figure 12-43 Une boucle d'initialisation, une d'acquisition et une d'affichage.
Malheureusement cela ne fonctionne pas !

Excepté que cela ne fonctionne pas, les deux tableaux ne sont pas égaux, car on a recopié deux objets de types « Banc De Mesures ». Les mesures sont faites dans le premier et l'affichage dans le second !

Il faut utiliser des références pour éviter de dupliquer les objets.

Attention, par défaut, LabView restreint la possibilité de créer des références qu'aux VIs de la classe. Cette option est dans la page héritage des propriétés de la classe (Cf Figure 12-44).

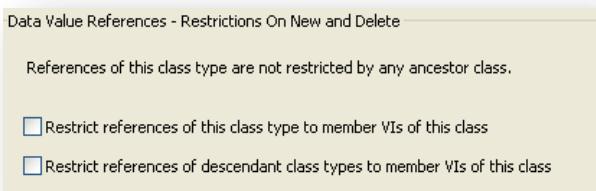


Figure 12-44 Décocher les restrictions par défaut pour pouvoir faire des bêtises (à éviter dans un vrai code)

Il faut décocher la case « **Restrict references of this class type to member VIs of this class** ». Cette option est active car normalement ce n'est pas l'utilisateur des classes qui devrait créer des références, mais des méthodes des classes.

Voici ce que devient le code :

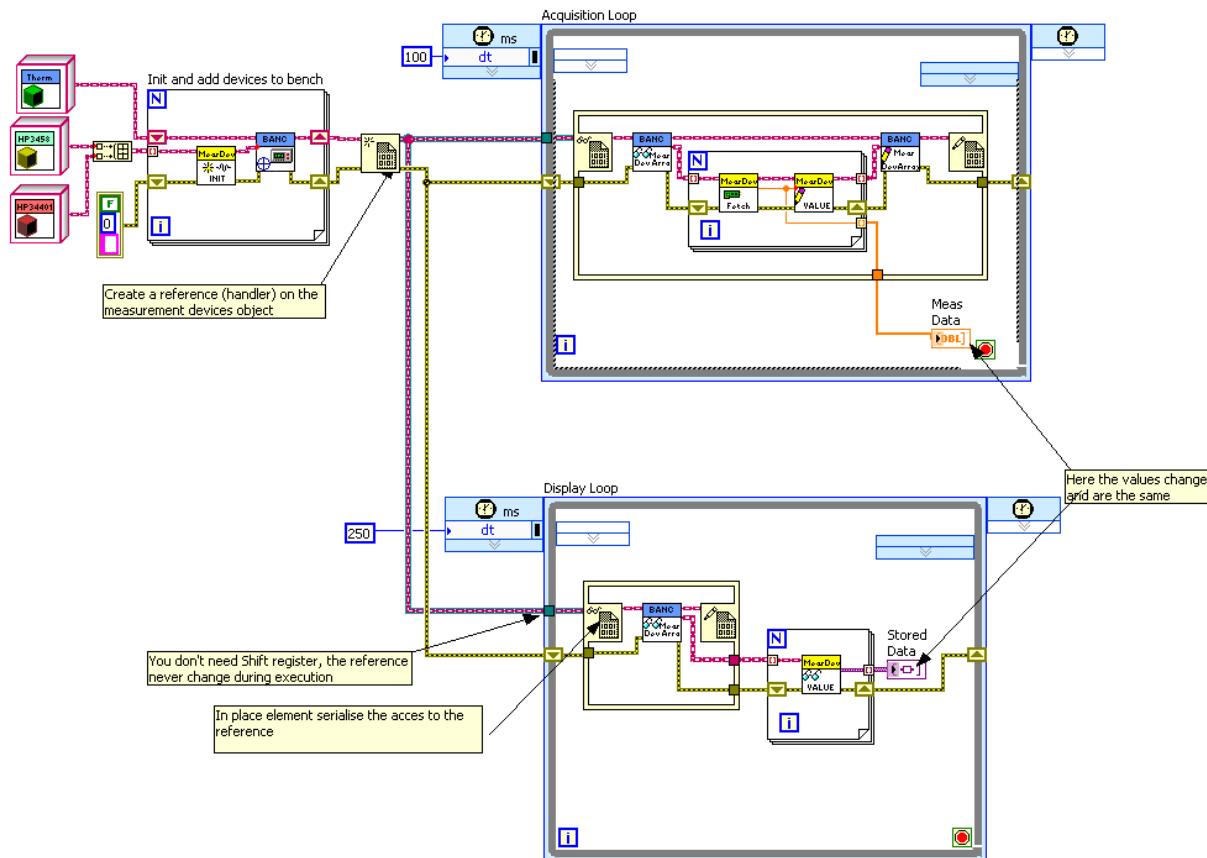


Figure 12-45 Une solution qui n'a que le mérite d'explorer clairement les concepts de passage d'objets par adresse mais qui est loin d'être optimisée

Les deux boucles de la Figure 12-45 partagent maintenant la même référence et donc le même objet. Dans ce cas (utilisation de références) et uniquement dans ce cas, la structure « **In Place Element Structure** » assure la sérialisation de l'accès aux données pointées par les références. Le contenu de l'objet est extrait dans la structure « **In Place Element Structure** » par le « **Value Reference Read Element** ». Les propriétés « **Value** » de « **mearement devices** » sont modifiée par la méthode « **Fetch** » puis mis à jours par « **Value Reference Write Element** ». Il faut donc impérativement connecter la sortie « **Object** » de « **Measure** » à l'entrée du « **Value Reference Write Element** » pour que l'objet soit à jour (rappelons que si un fil passe une donnée directement entre l'entrée et la sortie d'un nœud c'est une copie de l'entrée qui est créée car le compilateur ne sait pas dans le cas d'un VI dynamique si toutes les instances de la méthode se comporteront de la même manière).

Exercice 12-16 Tentez l'expérience du référencement d'objets.....Mais en utilisant des références pour tous les objets du projet. Il est préférables de sauvegarder votre ancien projet avant de commencer à tout détruire !

Dans ce type d'approche il est préférable de choisir, soit le passage d'objets par valeurs, soit par adresses, un mixte comme présenté plus haut n'est vraiment ni simple, ni optimisé. Cependant il est préférable d'utiliser un passage d'objets par valeurs car il est conforme au paradigme de flot de données de LabView. Il convient dans ce cas de mettre en place des moyens de communications entre objets si des données doivent être partagées entre différentes boucles indépendantes.

ASPECTS COSMETIQUES

LES FILS

Afin de repérer les connections de nos objets, LabView offre la possibilité de modifier le style et les couleurs des fils. C'est une propriété de la classe dans la page « **Wire Appearance** ». Il n'est pas idiot de garder des traits communs aux classes parentes et de distinguer les enfants par des détails de couleur.

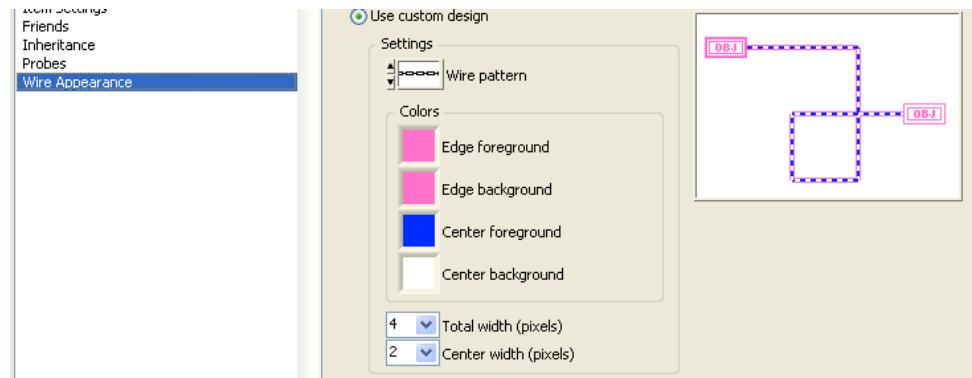


Figure 12-46 Personnalisation de l'apparence des fils

LES ICIONES

De la même manière les icônes de toutes les méthodes d'un classe peuvent avoir un trait commun, par exemple le nom de la classe, et toutes les icônes de la hiérarchie un autre trait commun, par exemple la couleur de fond du bandeau. Ces éléments sont définis dans la page « **General Settings** » des propriétés de la classe.

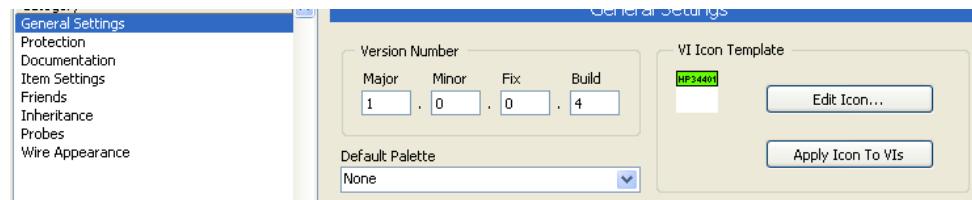


Figure 12-47 Personalization des entêtes d'icônes

13. ACQUISITION DE DONNEES

APERÇU

La palette **Acquisition de données** contient les VIs pour contrôler les cartes DAQ de National Instruments. Ces cartes sont souvent multifonctions : conversion analogique-numérique & numérique-analogique, entrée/sortie numérique et compteur/timer. Il est indispensable de bien connaître les fonctionnalités de la carte utilisée pour pouvoir la programmer correctement.

CONFIGURATION MATERIELLE

Sur les plateformes PC, les cartes actuelles sont toutes « Plug&Play », le port PCI attribue automatiquement les adresses et les interruptions requises par le matériel.



National propose un utilitaire d'exploration et de configuration de ses matériels, nommé MAX pour Measurement & Automation eXplorer.

Lorsque MAX est lancé, il détecte les périphériques qu'il connaît et les place dans la section **Périphériques et Interfaces**. Vous pouvez alors tester, de façon manuelle, la ressource matérielle et sa connexion au monde extérieur, (toujours utile avant d'incriminer autre chose...).

MAX permet également d'indiquer aux VI d'acquisition comment est configurée la carte insérée dans le PC, c'est-à-dire par exemple, sa tension de référence, le mode bipolaire, entrées référencées en différentielles... Cette configuration permet aux VIs d'acquisition d'assurer la correspondance entre mots binaires et valeurs réelles.

Enfin et surtout MAX permet d'assigner un **numéro** à une **carte**, c'est ce numéro qui identifiera la carte dans LabVIEW. Cette indirection permet de changer de carte (si elles ont des possibilités semblables) sans modifier le code.

EXERCICE 13-1 : UTILITAIRE MAX : CONFIGURATION ET TEST

OBJECTIF: UTILISER MEASUREMENT & AUTOMATION EXPLORER POUR RECONNAITRE LA CONFIGURATION ACTUELLE.

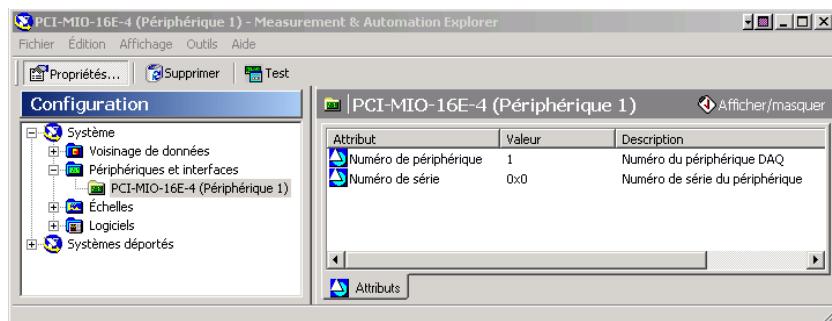
MAX va vous permettre de connaître le type de carte installée dans la machine mise à votre disposition et de vérifier le fonctionnement de l'ensemble.

Examen de la configuration

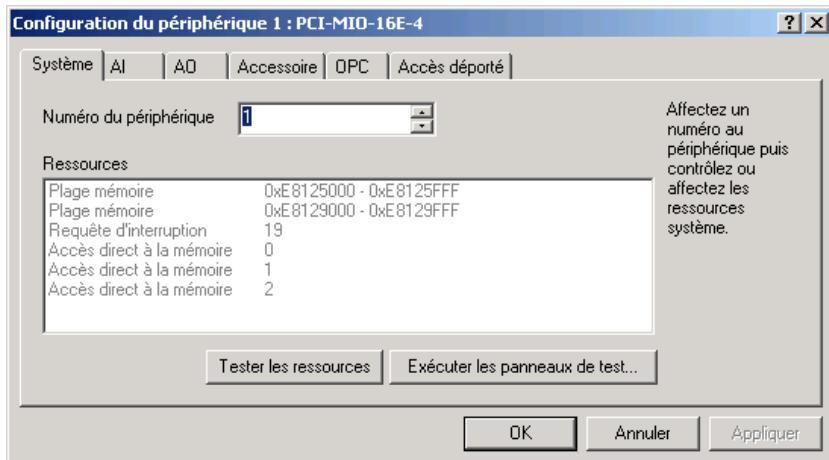
Lancer MAX depuis le raccourci bureau.



Développez les périphériques et interfaces pour voir les cartes installées, vous devriez voir des PCI MOI E4 ou des PCI 1200 qui sont des cartes E/S multifonction “low cost”.



Vous pouvez obtenir des informations sur une carte spécifique en sélectionnant Propriétés dans le menu contextuel associé à la carte. la fenêtre ci-dessous apparaît :



Cette fenêtre possède plusieurs onglets. Le premier onglet, **Système**, indique les ressources système assignées à la carte et le **numéro du périphérique** qui vous servira dans LabVIEW. Les autres onglets permettent de configurer les paramètres d'entrées/sorties analogiques. Appuyez sur le bouton **Tester les ressources** pour vérifier la configuration de la carte.

Tester les E/S de la carte.

Appuyez sur le bouton Exécuter les panneaux de test. Ces derniers vous permettent de tester les fonctionnalités du périphérique DAQ de votre choix, telles que les entrées et les sorties analogiques et numériques. Examinez les différents panneaux et leurs possibilités.

FIN DE L'exercice 13-1

VIS D'ACQUISITION

Les VIs d'acquisition sont situés dans la palette **Mesures NI**,

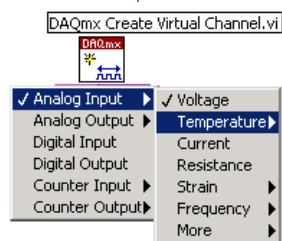


Acquisition de données (anciens VIs destinés à assurer la compatibilité ascendente)

Acquisition de données DAQmx (nouvelle version des drivers de carte d'acquisition)

« NI Switch », « vision » et « motion » sont des VI spécialisés pour la commutation, l'acquisition, le traitement d'images et les systèmes de déplacements.

ACQUISITION DE DONNEES DAQMX



L'acquisition de données par DAQmx repose sur la notion de tâche. Une tâche est ici un paramètre de contrôle : un gain, une mise à l'échelle, un timing, un déclenchement... bref, tout ce qui caractérise une acquisition. Tous les VIs traitant d'acquisition DAQmx sont polymorphes : le même VI crée une tâche de lecture de tension, de sortie d'impulsion, écriture sur une sortie analogique.

La configuration des différents VIs est assez complexe, il est préférable d'utiliser :



Soit MAX pour créer une tâche DAQmx.



Soit un VI Express DAQ assistant pour engendrer automatiquement le code contrôlant l'acquisition.

Ces deux méthodes sont comparables et font intervenir les mêmes assistants, le fait d'utiliser MAX permet une plus grande souplesse (changement de voies, de cartes sans changer le code), utiliser « **DAQ assistant** » fige un peu plus les choses, mais donne plus de robustesse.

TACHES ET VOIES VIRTUELLES DANS MAX



Les voies et les tâches sont définies dans MAX au niveau de la branche « **Voisinage de données** » par un clic droit « **Créer un nouvel objet** ».

Il est possible de définir :

Une tâche (une voie virtuelle ou physique, ainsi que les conditions d'échantillonnage et de déclenchement)

Une voie virtuelle (relie une voie physique à un nom)

EXERCICE 13-2 : UTILITAIRE MAX CREATION D'UNE TACHE

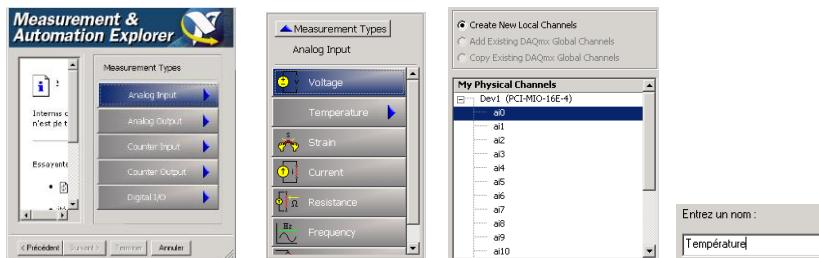
OBJECTIF: UTILISER MEASUREMENT & AUTOMATION EXPLORER CREER UNE TACHE D'ACQUISITION.

MAX va vous permettre de configurer l'ensemble des paramètres d'une acquisition par carte. Dans cet exemple, nous configurerons une tâche pour 100 mesures sur la voie 0 de la carte à la fréquence de 1000Hz avec un déclenchement en mode immédiat.

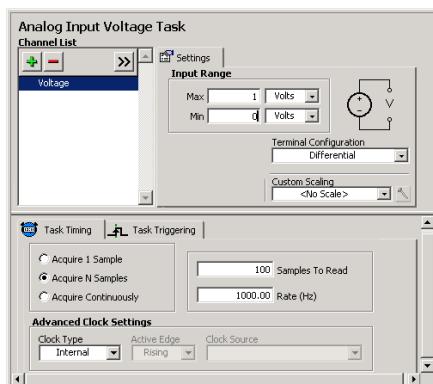
Lancement de l'utilitaire de configuration

Lancer MAX depuis le raccourci bureau. Dans « Voisinage de données », par un clic droit, sélectionnez « Créer un nouvel objet » puis, dans la fenêtre qui apparaît, sélectionnez NI-Daqmx Task.

Une fenêtre permettant de définir le type de mesures à effectuer apparaît. Sélectionnez Analog Input, puis Voltage, puis Voie ai0. Enfin donnez-lui le nom de Température.



Il faut maintenant définir la sensibilité d'entrée, le nombre et la fréquence d'échantillonnage, puis le mode de déclenchement. Choisissez les valeurs suivantes :



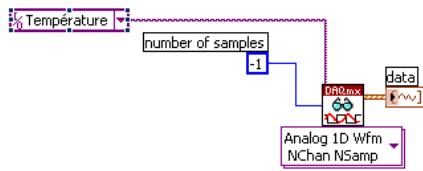
Le bouton de test permet de vérifier la configuration. Exécutez le test et vérifiez son bon fonctionnement. Il ne reste plus qu'à enregistrer la tâche par le bouton et à quitter MAX.

Acquisition dans LabVIEW

Pour exécuter cette tâche dans LabVIEW, il faut créer une référence sur cette tâche puis lancer son exécution et enfin transférer les données. Ces opérations sont faites à l'aide de deux VI :

- Constante de nom de tâche DAQmx renvoie une référence (un pointeur) sur les tâches définies dans MAX (votre tâche Température doit apparaître dans la liste déroulante). Cette constante de nom est en fait un Vi complexe.
- DAQmx Read initialise les paramètres de conversion, lance l'acquisition, convertit les données binaires en tensions. Ce VI est polymorphe, il est donc nécessaire de sélectionner dans la liste déroulante l'option correspondant au type d'acquisition : « Analog→Single channel→Multiple sample→1D DBL ».

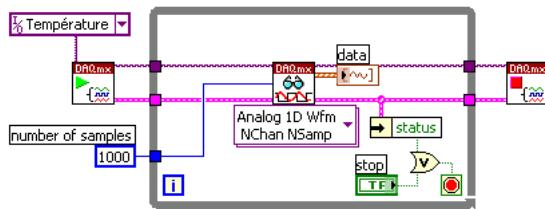
Le code se résume ainsi à :



Ce code est généré automatiquement en sélectionnant dans le menu contextuel de la Constante de nom de tâche DAQmx→Générer le code→Exemple.

Si par exemple au lieu de prendre 1000 points, nous avions voulu une acquisition continue, il suffit de rééditer la tâche en sélectionnant dans le menu contextuel de la Constante de nom de tâche DAQmx→Editer la tâche pour relancer DAQ assistant et sélectionner Acquire Continuously.

En reprenant la procédure précédente, le code généré devient :



Apparaissent deux nouveaux VI, l'un démarre la tâche continue, l'autre la stoppe. Une boucle lit en permanence les données par flots de 1000 points.

FIN DE L'exercice 13-2

EXERCICE 13-3 : VOLTMETRE

OBJECTIF: ACQUERIR UN SIGNAL ANALOGIQUE EN UTILISANT UNE CARTE D'ACQUISITION DE DONNEES.

Vous allez construire un VI qui mesure la tension générée par le capteur de température intégré dans la boîte de démonstration. Ce capteur génère une tension proportionnelle à la température ($10\text{mV}/^\circ\text{C}$). Il est câblé physiquement sur la voie 0.

FACE AVANT

Par exemple :



Le capteur fournit une tension comprise entre 0 et 0.4V. Calibrez l'échelle du vu-mètre en conséquence. (le gain donné à la voie de mesure n'affecte pas la valeur lire en sortie)

Par un clic droit sur le contrôle Nom d'une tâche DAQmx (palette de commande E/S→Commandes nom DAQmx), sélectionnez Nouvelle tâche (assistant DAQ). Créez une nouvelle tâche nommée « T », associée à une entrée en voie 0

Entrée analogique.

Voltage

Voie ai0

Nom « T »

Gamme Min : 0 , Max :40 degrés C.

Mode différentiel

Echelle personnalisée (sélectionner « Create New) dans le menu déroulant de « Custom Scaling »

Echelle linéaire.

Nom : Celsius.

$Y = 100 X + 0$. (« Slope »= 100, « Intercept »= 0), « Scaled unit » :°C

« Acquire 1 Sample », « Clock Type » : Internal.

Une fois ces paramètres saisis, vous devriez, sous MAX, voir apparaître dans « Voisinage de données→NI-DAQmx Tasks », une voie T, et dans « Echelles→ NI-DAQmx Scales », une échelle Celsius.

DIAGRAMME



Dans une boucle while dont l'arrêt est assujetti au bouton STOP, faites l'acquisition d'un point

toutes les 100 ms

Analog DBL
1Chan 1Samp ▾

La fonction « DAQmx Read» option **Analog DBL, 1Channel, 1 Sample** lit la tension sur la voie concernée.

Enregistrez votre VI sous le nom Voltmetre.vi. Exécutez le, en posant le doigt sur le capteur de température vous devriez voir évoluer la tension mesurée.

Faire une moyenne glissante sur 4 points des données acquises.

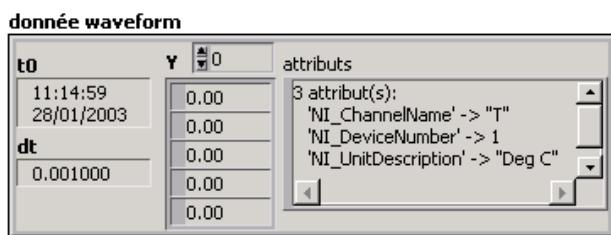
FIN DE L'exercice 13-3

ENTREES ANALOGIQUES TYPE WAVEFORM

Dans de nombreux cas, l'acquisition point par point n'est ni assez rapide, ni suffisamment précise en temps. Dans ce cas, il convient d'utiliser l'option « Multiple Samples » du VI **DAQmx Read**. Ces VIs font l'acquisition d'une rafale de points, à une fréquence d'échantillonnage déterminée. L'opération étant gérée matériellement par la carte, on peut compter sur des débits très importants sans perte de points. Le format natif renvoyé est de type Waveform.

DONNEE TYPE WAVEFORM

Un type waveform est un cluster qui contient les données d'acquisition, les éléments temporels et des attributs. Il peut s'agir aussi d'un tableau de clusters en cas d'acquisitions sur plusieurs voies.



Il est possible de les câbler directement aux Graphes, les échelles de temps sont alors fournies par le waveform.

Il existe une multitude de fonctions pour traiter ces données, elles sont situées dans les palettes **Analyse** et **Waveform**, consultez ces bibliothèques pour vous faire une idée de la richesse des traitements possibles.

EXERCICE 13-4 : ACQUISITION TRAITEMENT ENREGISTREMENT D'UNE WAVEFORM



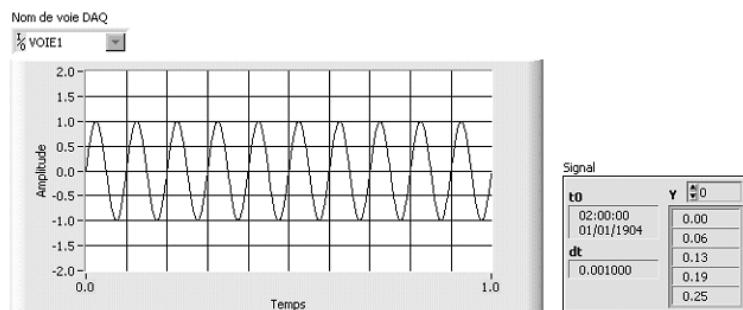
OBJECTIF: ECRIRE UNE ACQUISITION DANS UN FICHIER TEXTE.

On se propose de réaliser une acquisition de 1000 points à la vitesse de 1000 échantillons/seconde et d'enregistrer le résultat dans un fichier texte.

Sur le boîtier de démonstration, connecter l'entrée analogique 1 à la sortie sinus du générateur de fonctions.

FACE AVANT

Créez une face avant ressemblant à:



Créez une voie virtuelle de gain 1 appelée VOIE1, faisant l'acquisition sur l'entrée analogique 1.



DIAGRAMME

Construisez le diagramme en tenant compte des éléments suivants :

Sélectionnez l'option ci-contre.

Exporter des waveforms dans un fichier tableur, écrit les données au format txt en créant une entête. Vous pouvez également utiliser le VI express « Ecrire un fichier de mesures LabVIEW »

Pensez à la gestion d'erreurs.



Enregistrez le VI sous le nom Fichier_Waveform.vi.

Lancez le code, enregistrez les données sous le nom acq.txt..

Ouvrez acq.txt à l'aide du bloc note ou d'Excel, examinez les données et l'entête.

FIN DE L'exercice 13-4.

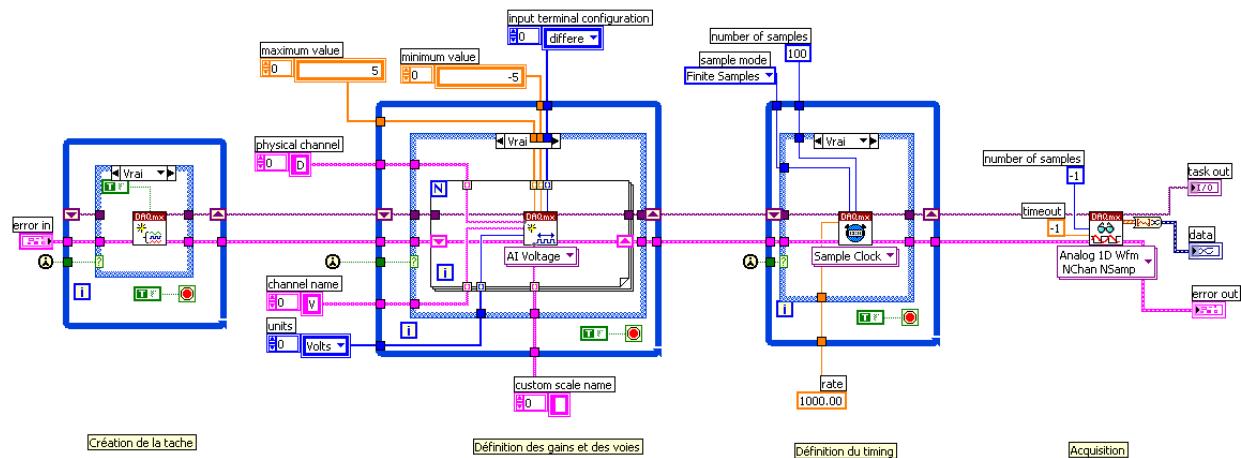
DAQ ASSISTANT



DAQ Assistant

L'utilisation de tâches créées et modifiables dans **MAX** peut s'avérer flexible mais pas sûr, il est possible de créer une tâche dans l'environnement **LabVIEW** qui ne sera visible que pour le programmeur. Le moyen le plus efficace pour créer une tâche d'acquisition est d'utiliser « **DAQ Assistant** ».

Un double clic sur l'icône lancera l'assistant que vous avez déjà vu, le code généré ressemble à :



Les trois boucles While définissent l'ensemble de la configuration d'acquisition. Le code de configuration n'est lancé qu'à la première exécution, les registres à décalage permettent de conserver la référence de la tâche si le code est exécuté plusieurs fois. L'ensemble des paramètres de configuration est défini par des constantes, mais il est très facile de les remplacer par des contrôles.

EXERCICE 13-5 : ACQUISITION TRAITEMENT ENREGISTREMENT EN VI EXPRESS

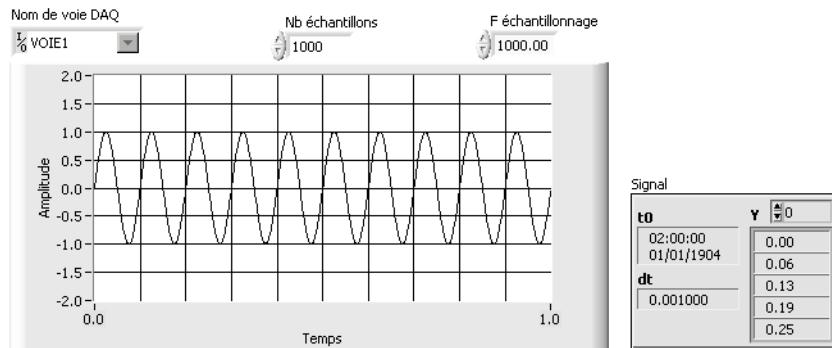
**OBJECTIF: ECRIRE UNE ACQUISITION DANS UN FICHIER TEXTE EN UTILISANT UN VI EXPRESS D'ACQUISITION.**

On se propose de réaliser une acquisition de X points à la vitesse de Y échantillons/seconde et d'enregistrer le résultat dans un fichier texte.

Sur le boîtier de démonstration, connecter l'entrée analogique 1 à la sortie sinus du générateur de fonctions.

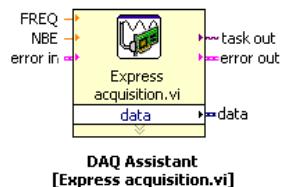
FACE AVANT

Créez une face avant ressemblant à:



« Nombre d'échantillons » spécifie le nombre de points à acquérir et « fréquence d'échantillonnage » la vitesse d'acquisition.

DIAGRAMME



Construisez le diagramme en utilisant un VI express d'acquisition. Vous éditez ce VI pour rajouter deux contrôles FREQ et NBE sur la face avant. N'oubliez pas de les relier à deux entrées du connecteur.

SORTIES ANALOGIQUES

Les fonctions de sorties analogiques ont un fonctionnement identique aux entrées, elles travaillent soit sur une donnée unique, soit sur une waveform, et cela sur une ou plusieurs voies.

Si une erreur survient, un message d'alerte apparaît et demande de stopper l'exécution ou de l'arrêter.

GENERATION DE WAVEFORM

Il est possible de générer continuellement une forme d'onde correspondant à une waveform, en cadençant la mise à jour des valeurs par une horloge propre à la carte, donc sans utiliser de ressources machines.



EXERCICE 13-6 SORTIE EN TENSION



Reprenez l'[exercice 13-4](#), modifiez le câblage de la boîte de démonstration en connectant l'entrée analogique 1 à la sortie analogique 0.

Changez le code pour générer une forme d'onde que vous choisirez dans la palette **Génération de waveform**. Cette onde sera ensuite générée de façon continue par la carte, puis acquise et enregistrée par les deux VI express déjà présents dans le code.

FACE AVANT

Vous pouvez ajouter un graphe pour visualiser la forme d'onde avant la sortie sur la carte.

DIAGRAMME

Tenez compte des éléments suivants :

Utilisez le VI express DAQ.

Vérifiez que la case « Use timing from waveform » est cochée dans l'assistant DAQ de sortie analogique.

Vous pouvez jeter un petit coup d'œil au code généré.

Vérifiez le fonctionnement de la chaîne complète.

FIN DE L'exercice 13-6

LES COMpteURS ET ENTREES/SORTIES NUMERIQUES

Les compteurs sont des ensembles de registres capables de s'incrémenter ou de se décrémenter sur des fronts venant de l'extérieur ou d'une horloge interne. Ils sont utilisés pour le comptage, la mesure de fréquence, de période, pour générer des impulsions...

Les entrées/sorties numériques sont des lignes logiques, physiquement disponibles. Elles permettent de commander des actionneurs tout ou rien, et de lire l'état de capteurs binaires.

La configuration de ces lignes est aussi l'affaire de **DAQ Assistant** ou de **MAX**.

EXERCICE 13-7 COMPTAGE ET AFFICHAGE

OBJECTIF: CREER UN VI DE COMPTAGE ET D'AFFICHAGE.

Cet exercice met en œuvre le codeur incrémental du pupitre de démonstration, le compteur 0 de la carte ainsi que les quatre LED du pupitre, associées au port 0. Le codeur fournit 24 impulsions par révolution, nous nous proposons de les compter, puis de les afficher (1) sur la face avant du VI dans un entier et sur quatre indicateurs binaires, (2) sur les quatre LED du pupitre de démonstration.

FRONT PANEL

La face avant peut ressembler à :



Les quatre LED du pupitre doivent refléter l'état des indicateurs binaires de la face avant (tableau de LEDs), qui codent eux même la valeur décomptée modulo 16. (Rappel : $18=2$ modulo 16)

DIAGRAMME

Le diagramme comporte une boucle While dont la condition d'arrêt est liée à l'état du bouton STOP. Le VI compteur doit être initialisé avant l'entrée dans la boucle de comptage et arrêté en fin de boucle.

Vous utiliserez les éléments suivants:

Pour la configuration des compteurs, dans l'utilitaire de configuration, sélectionnez « **Counter Input** » puis « **Edge Count** ». La carte possède deux compteurs pour compter un nombre d'événements ou un temps écoulé, codés par un entier long sur 32 bits. Un événement externe est une transition du signal sur la broche SOURCE spécifiée du compteur. Cette broche sera connectée à la sortie A ou B du codeur. Dans la partie « **settings** » et « **Task Timing** » laissez les valeurs par défaut. Le VI configure le compteur et lance l'acquisition tant que l'entrée STOP est fausse. S'il est exécuté dans une boucle ce VI renvoie à chaque tour le nombre d'événements comptés.

La configuration des sorties numériques est effectuée par « **Digital I/O** » puis « **Port Output** ». La carte utilisée ne possède qu'un seul port, les LED du pupitre sont connectées aux lignes 0 à 3. L'entrée « **data** » du VI est un

tableau d'entiers 32bits (si la carte contenait plusieurs ports, chaque ligne du tableau contiendrait la valeur d'un port). Nous utiliserons ici un tableau d'une ligne (1D).

 **Numérique » Quotient & Reste** donne accès à la division entière. Cette fonction permet de connaître la valeur à afficher modulo 16

 **Tableau » Sous-ensemble d'un tableau** permet d'extraire les quatre bits de poids faible.

Renverser un tableau 1D permettra d'afficher l'élément de poids fort à gauche.

Enregistrez le VI et exécutez-le. Vérifiez l'incrémentation de l'indicateur décompte, l'affichage des LED sur le pupitre et sur la face avant.

FIN DE L'exercice 13-7

14. CONTROLE D'INSTRUMENTS

GENERALITES

Les instruments de mesure professionnels sont en général pourvus d'une interface permettant de les relier à un calculateur. Cette connexion permet de configurer l'instrument et de récupérer les données acquises. Deux grands standards se partagent le marché de l'instrumentation, le bus GPIB et la liaison série. La liaison GPIB est mieux définie par la norme. Plus robuste, elle offre des avantages précieux en terme de synchronisation. Son coût est cependant nettement plus élevé.

La liaison série équipe en standard les ordinateurs, mais les problèmes de connexion et de configuration sont parfois déconcertants !

CONFIGURATION ET COMMUNICATION GPIB

La norme ANSI/IEEE 488.1-1987, plus connue sous le nom General Purpose Interface Bus (GPIB), décrit les spécifications électriques, mécaniques et fonctionnelles de l'interface. La norme 488.2 étend ces spécifications au langage utilisé.

La liaison GPIB est une liaison parallèle 8 bits, capable de débits de $\geq 1\text{Mo/s}$. Chaque interlocuteur du bus est repéré par une adresse unique (comprise en 0 et 30). Les spécifications générales sont les suivantes :

- Un maximum de 15 appareils connectés au bus,
- Une longueur de câble de 4m au maximum entre 2 appareils,
- Une longueur totale de câble maximale de 20m,
- Au minimum, les 2/3 des appareils connectés doivent être sous tension.

ARCHITECTURE DU LOGICIEL

L'implémentation des ressources GPIB est comparable à celle utilisée pour les cartes d'acquisition. Le pilote (DLL) contenant l'ensemble des commandes GPIB est installé automatiquement par le module « plug & play » lors du démarrage de la machine. Le logiciel « Measurement & Automation eXplorer – aussi appelé MAX » permet de configurer et de tester la liaison.

CONFIGURATION DU LOGICIEL

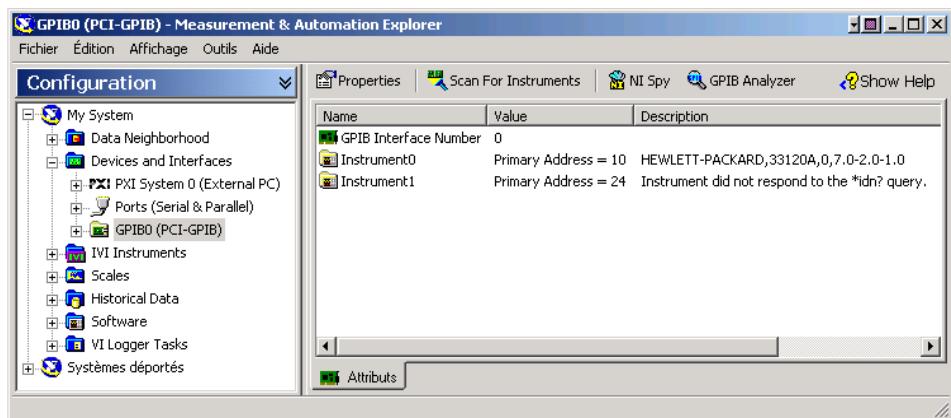


Pour configurer la carte une fois le pilote installé, lancer MAX. Développer la section **Périphériques et interfaces**, une section **GPIB0** doit apparaître. Cette dernière peut également être développée indiquant ainsi tous les appareils sous tension, physiquement connectés au bus. Un clic droit sur un instrument spécifique permet de vérifier son adresse et de communiquer avec lui de façon sommaire pour vérifier que la communication fonctionne correctement.

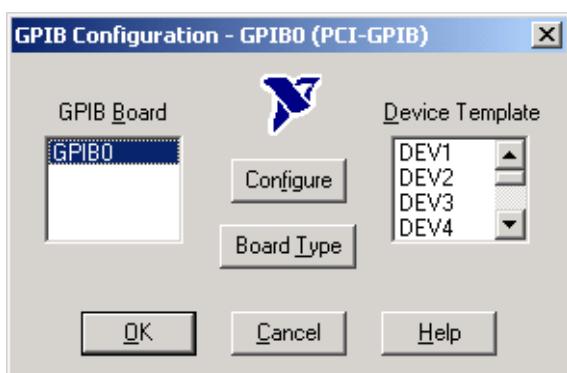
La façon la plus simple de vérifier que le périphérique communique correctement avec le PC, consiste à demander au périphérique son nom.

Pour cela il faut envoyer (écrire) la commande « *idn ? », puis dans un deuxième temps, lire la réponse de l'appareil. Certains appareils anciens, ne possédant pas de nom propre, renvoient une donnée en réponse à la commande. L'appareil ayant reconnu que le PC lui parle doit, sur sa propre face avant, allumer ses voyants « Remote » et « Listen ».

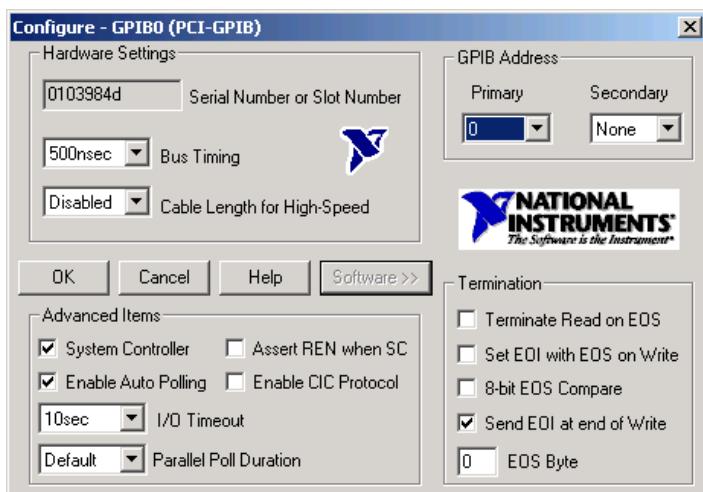
La copie d'écran présentée ci-dessous vous donne le résultat de cette opération réalisée par MAX en cliquant sur le bouton **Scan For Instruments**. Vous pouvez constater que le générateur HP33120 répond à la commande <*idn?>, mais l'appareil d'adresse 24 ne comprend pas cette commande car il n'est pas conforme à la norme IEEE 488.2 (appareils fabriqués avant 1992). Notez aussi que la carte porte le numéro 0, il est possible d'utiliser simultanément plusieurs cartes GPIB dans le PC.



le bouton **Properties** de la carte GPIB ouvre le panneau de configuration, il est possible de configurer indépendamment les cartes interfaces et les périphériques.



Dans un premier temps, étudions la configuration de la carte GPIB0. Les valeurs présentées ci-dessous sont les valeurs par défaut.



Le cadre **Hardware Setting** définit le timing du bus, avec des périphériques standard et une carte GPIB non « High Speed », il est préférable de garder la configuration par défaut.

Le cadre **GPIB Address** définit l'adresse de la carte sur le bus IEEE (comme tout interlocuteur connecté au bus), la valeur par défaut est 0, l'étendue valide est de 0 à 30. Aucun autre périphérique ne doit avoir cette adresse. Les adresses secondaires sont rarement utilisées.

Le cadre **Advanced Items** définit :

System Controller le PC est le maître des échanges, il définit qui parle et qui écoute. Il n'y a qu'un seul contrôleur sur le bus!

Enable Auto Polling autorise la scrutation automatique des appareils lorsque l'un d'entre eux fait une demande de service

Assert REN when SC bascule automatiquement la ligne Remote enable lorsque le système est contrôleur. Il est parfois utile de cocher cette case avec certains appareils anciens qui ne passent pas spontanément en mode remote, à la reconnaissance de leur adresse.

Enable CIC Protocol active une procédure de reprise de contrôle automatique du bus.

Time Out est la durée maximale d'attente de réponse d'un périphérique ; passé ce délai, le contrôleur reprend la main de façon autoritaire (évite tout blocage du bus, attention cependant à ne pas mettre des valeurs trop faibles pour la RAZ de certains appareils complexes).

Parallel Poll Duration est le temps maximum d'attente à une demande de scrutation parallèle, en standard 2 µs, plus si des extensions de bus sont utilisées.

Le cadre **Termination** configure la façon dont l'échange entre deux appareils connectés au bus se termine. La norme GPIB prévoit qu'une ligne (un fil physique du bus) particulière, nommée EOI (End of idendify), peut basculer pour indiquer la fin d'une transmission. Les périphériques utilisent souvent un (des) caractère(s) terminateur(s) (EOS End of String) en plus du basculement d'EOI. Les plus fréquents : 'CR', 'LF' et bien sûr la combinaison 'CR LF'. Une dernière méthode pour connaître la fin d'un message, connaître le nombre de caractères attendus (de loin la plus mauvaise). La configuration matérielle de la carte propose les options suivantes :

Terminate Read On EOS cette option termine la lecture sur réception du caractère terminateur

Set EOI With EOS on Write Bascule la ligne EOI au moment de l'envoie du terminateur. Les terminateurs hard et soft sont ainsi envoyés.

8 Bits EOS Compare, par défaut les codes ascii sont sur 7 bits, et le terminateur est cherché sur les 7 bits de poids faible, cette option force la comparaison sur 8 bits.

Send EOI At End Of Write force la ligne EOI en fin d'écriture

EOS byte valeur décimale de l'EOS

COMMUNICATION AVEC LES INSTRUMENTS

CARACTERISTIQUES PROPRES D'UN APPAREIL

Chaque appareil possède des caractéristiques propres dont il faut prendre connaissance avant de commencer à programmer l'appareil.

ADRESSE

L'adresse d'un appareil permet de l'identifier sur le bus. En conséquence deux appareils ne peuvent pas avoir la même adresse. Chaque appareil possède une adresse par défaut qui peut être changée sur le périphérique lui-même, en utilisant (généralement) les menus accessibles manuellement par sa face avant.

Un périphérique, reconnaissant son adresse cheminant sur le bus, passe en mode commande à distance (remote) ; en conséquence sa face avant est inhibée. Un appui sur le bouton « local » de cette même face avant peut la rendre à nouveau active

MOT DE PROGRAMMATION

Pour commander à un périphérique d'exécuter des tâches, il est nécessaire de lui envoyer des ordres sous la forme de chaînes de caractères ASCII. Ces derniers sont généralement écrits dans un protocole intitulé SCPI (Standard Code for Programming Instruments). Un effort de normalisation du langage de commande a permis l'apparition du langage SCPI. Celui-ci est accepté par tous les appareils postérieurs à 1990. Chaque phrase du message correspond à une chaîne de caractères (ex : « *idn? » ou « :FUNC:MEAS »). La compréhension par le programmeur des nombreuses commandes SCPI et leur assemblage dans une syntaxe très stricte requiert une lecture très approfondie des notices des appareils. Elle permet théoriquement d'écrire un code correspondant exactement à l'application recherchée. La phase de développement peut être assez longue et fastidieuse.

Les codes correspondent à des tâches génériques. Ainsi le code de deux appareils, de fonction identique, sont-ils très proches et souvent compatibles. Les mots de programmation sont précisés dans la notice de chaque appareil.

FORMAT DES DONNEES

Les données sont exprimées en ASCII ou parfois en binaire, dans un format propre à chaque appareil. Une fois choisi le format, celui-ci reste invariant – en particulier le nombre de caractères- quelle que soit l'amplitude de la donnée.

Le format peut être totalement numérique : 1.23456 ou 1.23E-4. Il peut également comporter une partie alpha-décimale : NDCV 1.234E-3, spécifiant la nature de la donnée (normal DC voltage). Dans les 2 cas, la donnée reste sous forme de chaîne et doit être transformée en nombre pour autoriser des traitements numériques.

ETAPES ESSENTIELLES D'UNE COMMUNICATION PC <=> PERIPHERIQUE

Un programme informatique de commande à distance d'un instrument de mesure contient un certain nombre de tâches standards qui sont abordées ci dessous.

REMISE A ZERO

Remettre un périphérique à zéro consiste à le configurer dans un état initial unique, prévu en entreprise, et normalement identique à celui observé lors la mise sous tension (conditions par défaut ou « default factory conditions »)..

La remise à zéro (RAZ) prenant souvent plusieurs secondes, il est nécessaire d'introduire dans le programme une temporisation permettant d'attendre la fin de cette RAZ.

CONFIGURATION

Les appareils de mesures actuels possèdent un très grand nombre de modes de fonctionnement. En mode commande à distance (remote) ceux-ci sont imposés exclusivement par voie informatique. L'usage de la face avant des appareils reste alors limitée à la visualisation de résultats ou des status.

A chaque mode de fonctionnement correspond une **chaîne de caractères** (ou mot de programmation) qui devra être écrite par le PC sur le périphérique, pour lui imposer la ou les fonctions correspondantes. Les commandes des périphériques étant très nombreuses, il est nécessaire d'envoyer exclusivement celles qui diffèrent des conditions par défaut.

DECLENCHEMENT

La configuration d'un appareil prépare la mesure, mais ne la déclenche généralement pas. L'exécution de la mesure ne s'effectue qu'après apparition des conditions de déclenchement.

A un ordre de déclenchement unique peuvent être associées une ou plusieurs mesures. Le déclenchement peut être :

Interne : automatique donc non maîtrisé.

externe : associé au basculement d'une ligne ou à la fermeture d'un interrupteur.

logiciel : associé à l'expédition d'une commande informatique spécifique.

La maîtrise des conditions de déclenchement est souvent une étape critique de la commande à distance.

LECTURE

Une fois la mesure effectuée, la donnée est disponible dans la mémoire du périphérique. Le transfert de la donnée vers le calculateur est une opération de lecture.

La fin du transfert est associée à la détection du caractère de fin de chaîne, au basculement d'une ligne spécifique (EOI), à l'obtention d'un nombre de caractères ou enfin au dépassement du temps maximum autorisé (time out).

TRAITEMENT DES DONNEES

Les données peuvent être l'objet de traitements ultérieurs, une fois mises sous forme numérique. La bibliothèque de traitement de LV6.1 est riche de nombreux outils : visualisation, filtrage, régression, FFT,

Les traitements effectués par les appareils eux-mêmes : mise à l'échelle, FFT... relèvent de la configuration de l'appareil.

SOURCES D'ERREURS CLASSIQUES

Les 3 points ci dessous étant responsables d'un grand nombre d'erreurs, il est souhaitable de préciser leur concept.

TIME OUT

La fiabilité des échanges impose de ne transmettre une nouvelle commande qu'après acceptation de la précédente. Pour ne pas bloquer le bus et le PC indéfiniment, on convient a priori d'arrêter l'échange en cours si

celui-ci dure plus de X secondes (classiquement 10s). Le calculateur reprend alors la main, mais la commande ou la donnée en cause est perdue.

TERMINATEUR

Le terminateur indique la fin d'une commande ou d'une donnée. Sous peine d'erreur, le terminateur envoyé par le parleur doit être le même que celui attendu par l'écouteur (qu'il soit matériel ou (et) logiciel).

SEPARATEUR DECIMAL

Le séparateur décimal reconnu par les appareils de mesures est le point alors que classiquement en français, la virgule est utilisée. Pour changer ce dernier, aller sur le bureau, dans le menu Paramètre/Panneau de configuration/ Options régionales /Nombre.

BIBLIOTHEQUES DE CONTROLE D'APPAREILS DANS LABVIEW

VI'S SPECIFIQUES A L'INTERFACE GPIB OU RS 232C



Etant avant tout un langage destiné à l'instrumentation, LabVIEW est doté de toutes les fonctions permettant de piloter les interfaces. Il dispose en particulier de jeux de Vi's spécifiques au GPIB (IEEE488) et à la liaison série RS232C. Les fonctions de commandes spécifiques sont situées dans la palette **E/S Instruments**.

La norme IEEE488 ayant évolué de la version 1 vers 2, deux bibliothèques spécifiques correspondent à chacune des étapes. L'extension des liaisons possibles (USB, TCP, WIFI...) a induit une nouvelle couche logiciel qui permet de s'affranchir un peu plus de l'interface, il s'agit de VISA qui est portée par VXIplug&play Systems Alliance.

VISA



Virtual Instrument Software Architecture (VISA) est une couche logicielle de niveau application qui permet de communiquer avec les pilotes d'entrées/sorties installés sur le système. Le code (notamment les drivers d'instruments) est plus facilement réutilisable pour d'autres types d'interfaces, et d'autres langages et d'autres plateformes. VISA n'est pas un élément de LabVIEW, mais une interface d'application (API) standardisant les appels aux drivers.

TERMINOLOGIE

Les fonctions VISA font appel à une **Ressource** du système terme générique désignant un appareil connecté sur un port (série, parallèle, GPIB, USB, VXI...).

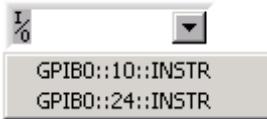
Cette ressource possède un **Nom**, qui identifie son interface, son adresse et le type de **session**. Une **session** VISA est un identificateur logique unique utilisé par VISA pour communiquer avec une ressource, elle peut être de type E/S ou événement.

Les noms de ressources valides de la version 3 sont les suivants :

Interface	Syntaxe
VXI INSTR	VXI[carte] ::adresse logique VXI[::INSTR]
VXI MEMACC	VXI[carte] ::MEMACC
VXI BACKPLANE	VXI[carte][::adresse logique VXI] ::BACKPLANE
VXI SERVANT	VXI[carte] ::SERVANT
GPIB-VXI INSTR	GPIB-VXI[carte] ::adresse logique VXI[::INSTR]

GPIB-VXI MEMACC	GPIB-VXI[carte] ::MEMACC
GPIB-VXI BACKPLANE	GPIB-VXI[carte][::adresse logique VXI] ::BACKPLANE
GPIB INSTR	GPIB[carte] ::adresse primaire[::adresse secondaire][::INSTR]
GPIB INTFC	GPIB[carte] ::INTFC
GPIB SERVANT	GPIB[carte] ::SERVANT
PXI INSTR	PXI[carte] ::périphérique[::fonction][::INSTR]
Serial INSTR	ASRL[carte][::INSTR]
TCP/IP INSTR	TCP/IP[carte] ::adresse hôte[::nom du périphérique LAN][::INSTR]
TCP/IP SOCKET	TCP/IP[carte] :: adresse hôte ::port ::SOCKET
USB Instr	
USB Raw	

Nom de ressource VISA



Si vous utilisez un appareil GPIB d'adresse 21 connecté à la carte n°0 la ressource a pour nom

GPIB0::21::INSTR. Si vous connectez un périphérique à la liaison COM1, la ressource prend pour nom : **ASRL0::INSTR.**

Si les ressources sont présentes sur la machine lors du développement, elles apparaissent automatiquement dans la liste déroulante associée à un contrôle de type VISA. Si ce n'est pas le cas, il faut entrer le nom de la ressource au clavier.

UTILISATION DES VISA

Les fonctions VISA les plus couramment utilisées pour communiquer avec les instruments de mesure sont les fonctions VISA : VISA Write et VISA Read.

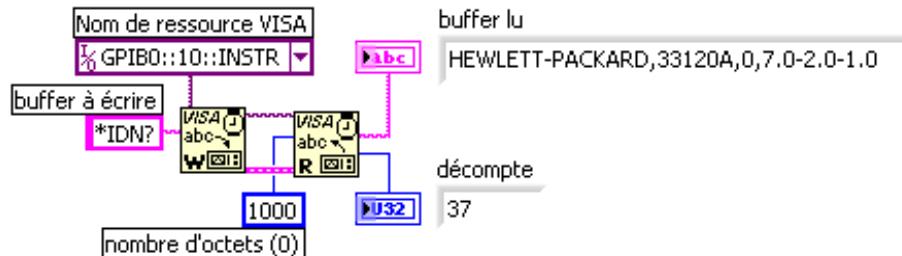


VISA Write écrit le contenu d'un tampon vers le périphérique désigné par le nom de la ressource VISA.



VISA Read lit les données en provenance du périphérique. Le VI demande le nombre de caractères à lire. Cette valeur doit être supérieure ou égale au nombre d'octets à transmettre. Si la fin de message est implémentée par le matériel (GPIB) la lecture s'arrête sur le terminateur, sinon elle s'arrête au nombre de caractères ou au time out.

L'exemple suivant fait une identification d'un appareil connecté sur une carte GPIB.



EXERCICE 14-1 : PROGRAMMATION D'UN VOLTMETRE ET LECTURE D'UNE RESISTANCE

OBJECTIF: CONFIGURER UN APPAREIL DE MESURE, DECLENCHEZ LA CONVERSION ET RAPATRIER LA DONNEE.

Dans cet exercice, vous configurez un multimètre en ohmmètre 4 fils, avec un déclenchement sur GET (Group execute trigger) de 4 mesures successives puis vous transférerez les données vers le calculateur.

FACE AVANT

En voici une représentation, l'indicateur à gauche est un tableau, le bouton déclenchement envoie un ordre GET.



DIAGRAMME

Le diagramme contiendra une boucle infinie dans laquelle on entre après une phase d'initialisation du multimètre. Lisez attentivement la documentation de l'appareil pour trouver les mots de programmation. Vous aurez besoin principalement des éléments suivant :

- Les fonctions **VISARead/Write** pour configurer l'appareil et lire les données. Parmi les éléments à écrire dans la phase d'initialisation, il faut :
 - définir la configuration de l'appareil (ohmmètre 4 fils)
 - spécifier la source de déclenchement (ici le bus)
- La fonction **VISA Assert Trigger** assure le déclenchement logiciel de la mesure par le périphérique (le mode de déclenchement dépend du type de la ressource VISA ; en GPIB envoie la commande *trg).

Vous allez certainement avoir besoin des fonctions de conversion chaîne↔chiffres (Fonction/conversion chaîne→Nombre/chaîne décimale en nombre).

Enregistrez le VI sous le nom acquieee.VI et exécutez le.

FIN DE L'exercice 14-1

ASSISTANT D'E/S INSTRUMENTS

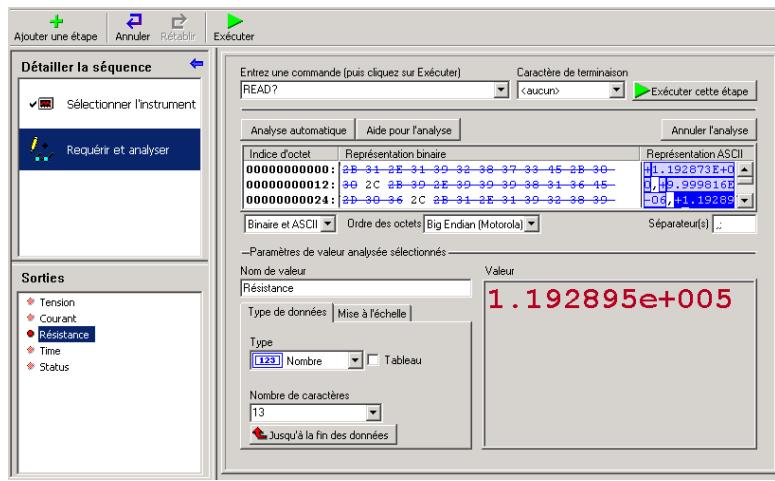


La version 2009 de LabVIEW dispose d'un VI express **Instrument I/O Assistant** permettant de chaîner des opérations telles que : initialisation, configuration, mesure et traitement. Une fois le VI posé sur le diagramme, un double clic permet de le configurer. Le VI express permet de séquencer un certain nombre d'étapes. Il y a quatre types d'étapes différentes :

- Selectionner un instrument
- Envoyer une requête et analyser la réponse
- Envoyer une requête
- Lire une réponse et l'analyser



L'interface permet d'ajouter les étapes, de les tester, d'extraire des réponses les éléments essentiels. La saisie d'écran suivante récupère, par une commande « READ? », les données d'un multimètre (2400 Keithley). Ce dernier envoie 5 variables séparées par des « , » et nommées dans la partie **Sorties**. Il est possible par un clic droit de choisir celles qui sortiront du VI. Chaque ligne barrée dans le cadre **Représentation binaire** indique que cette partie des données est traitée.



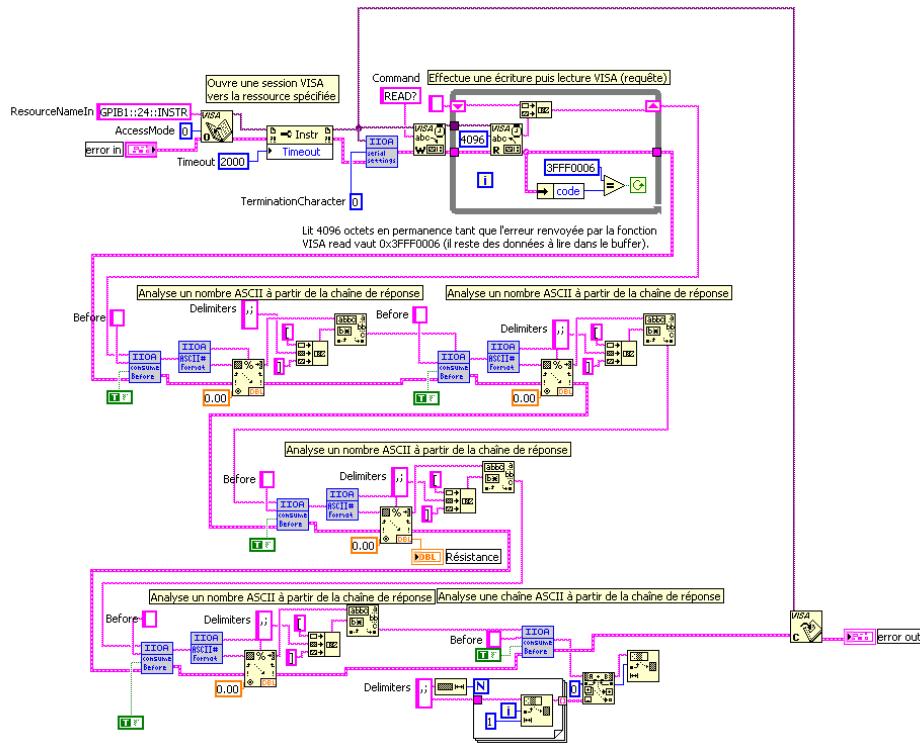
Lorsque la séquence est testée, un appui sur OK enregistre les paramètres de cette occurrence de

Instrument I/O Assistant.

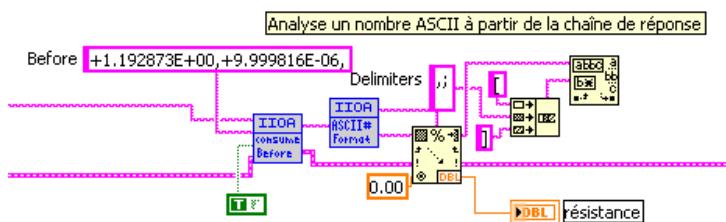
Il est possible de voir le code généré en sélectionnant **Ouvrir la face avant** dans le menu contextuel du VI, cette conversion est définitive, il ne sera plus possible d'ouvrir la fenêtre de configuration du VI express.

Voici le code généré, il appelle plusieurs commentaires :





Une ouverture et une fermeture de session VISA sont présentes, il faudra les retirer si le VI est utilisé dans une boucle rapide. L'extraction des données n'est pas « très optimisée », les données inutiles étant tout de même extraites mais non utilisées. Attention, si vous n'aviez sélectionné que la zone de données correspondant à la résistance, le code généré serait pour la partie extraction :



Vous constatez que le code recherche l'expression exacte +1.192873E+00,+9.999816E-06, qui correspond à la valeur de courant et tension durant la mesure, mais qui est tout sauf constante, ce code a toutes les chances de ne fonctionner qu'une seule fois, le jour de sa création ! Si les VI express sont sans conteste de puissants outils, il faut penser qu'ils générèrent un code standard, suffisant dans 90% des cas, mais cela n'empêche pas de vérifier le code et de le modifier au besoin.

EXERCICE 14-2 : PROGRAMMATION D'UN VOLTMETRE ET LECTURE D'UNE RESISTANCE EXPRESS

OBJECTIF: CONFIGURER UN APPAREIL DE MESURE, DECLENCHEZ LA CONVERSION ET RAPATRIER LA DONNEE.

Tentez de créer avec les VIs express **Instrument I/O Assistant** l'ensemble des opérations de l'exercice [exercice 14-1](#).

FIN DE L'exercice 14-2

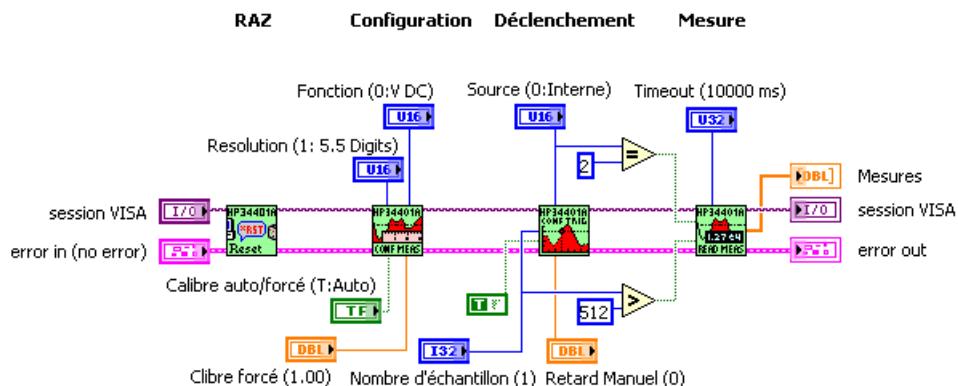
DRIVER D'INSTRUMENT

Pour faire gagner du temps aux utilisateurs, les grands constructeurs écrivent eux même et pour chaque appareil, un driver permettant d'effectuer la majorité des tâches susceptibles d'être demandées à celui-ci. La taille du VI correspondant croît, mais la mise en application devient alors beaucoup plus simple et rapide. La tâche du programmeur consiste alors à chaîner entre eux les Vi's issus du driver pour construire sa propre application.

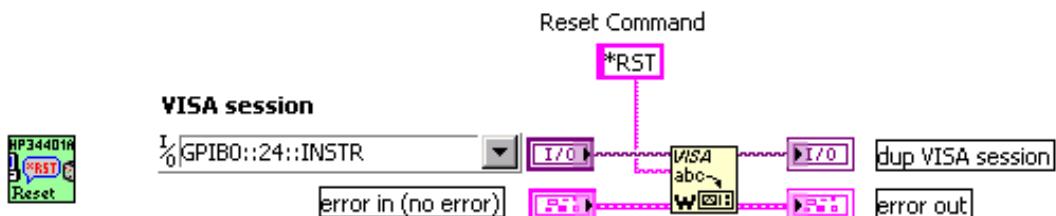
Un driver est une librairie LabVIEW (*.llb) faisant appel, soit aux fonctions VISA de LabVIEW (drivers Plug&Play), soit à une DLL écrite dans un autre langage (driver IVI). La librairie est généralement téléchargeable depuis le réseau. Pour être visibles depuis LV7, les librairies standard **doivent** être déposées dans le répertoire \Program Files\National Instruments\LabVIEW 7.0, les librairies IVI ont un installateur qui se charge de tout.

EXEMPLE D'APPLICATION SIMPLE

Voici un exemple des VIs du driver Plug&Play disponibles pour le multimètre HP34401. Le programme ci dessous comporte quatre parties bien séparées

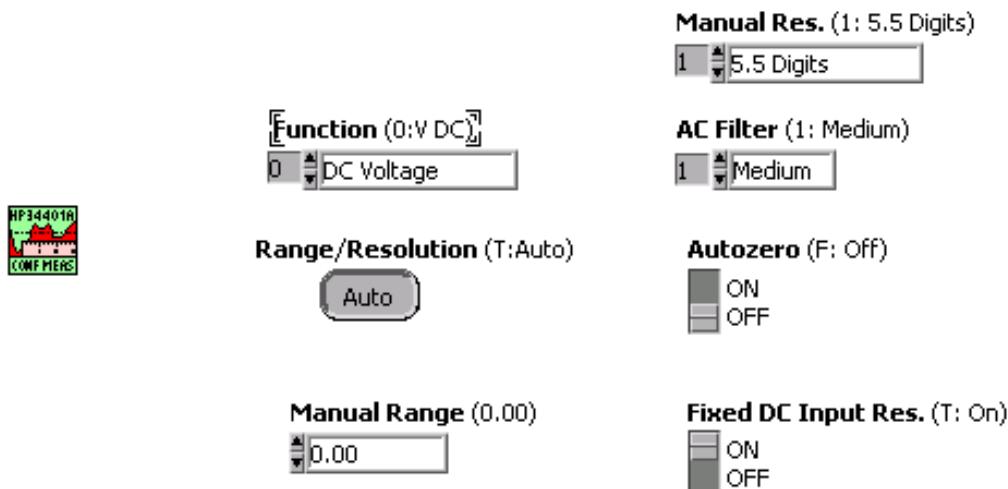


INITIALISATION



Ce VI définit le nom de la session visa (ici GPIBO ::10 ::INSTR) puis écrit la commande * RST sur ce périphérique pour le remettre dans les conditions par défaut. Ainsi l'usage du Vi <Reset>, extrait du driver, rend l'action de remise à zéro totalement transparente.

CONFIGURATION



Ce Vi reçoit du Vi précédent le nom de la session VISA et la gestion des erreurs

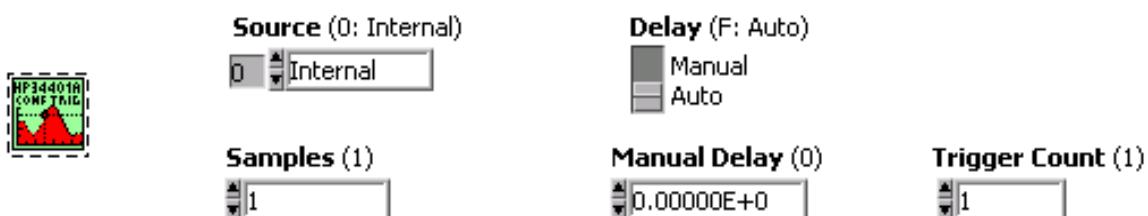
Il permet définir toutes les configurations possibles du multimètre Hp34401 voltmètre continu ou alternatif, ampèremètre, ohmmètre, thermomètre...

Le choix du calibre peut s'effectuer manuellement ou automatiquement (booléen vrai → choix automatique)

En mode manuel, la résolution varie entre 1 et 5.5 digits, imposée par une liste déroulante.

Dans le cas des mesures sur des grandeurs alternatives, l'efficacité du filtrage est accessible (forte, moyenne, faible) est accessible.

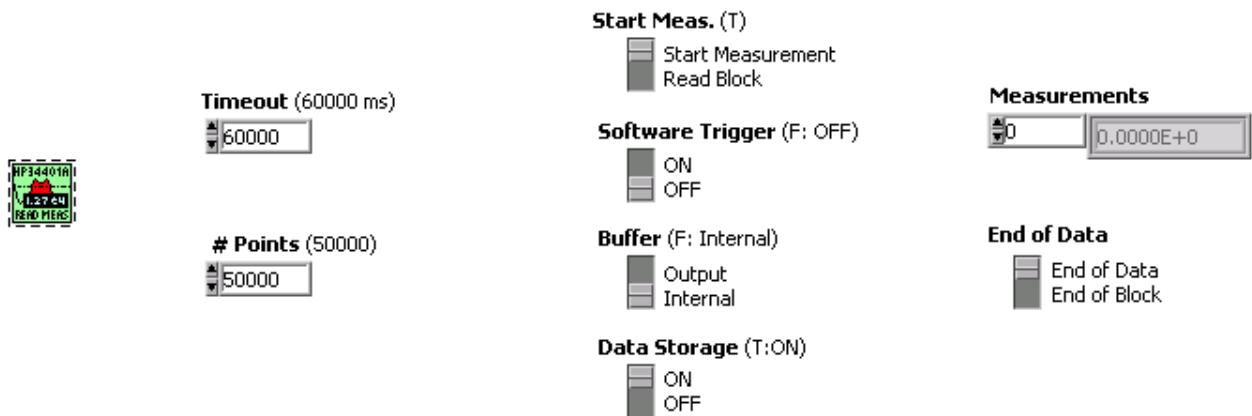
DECLENCHEMENT



Le contrôle « trigger count » fixe le nombre de déclenchement associé à une exécution du Vi « Conf Trig »

Le nombre d'échantillons et le retard introduit entre chaque prise de points sont également accessibles de façon évidente. Par exemple si « trigger count = 2 » et si « sample count =100 » alors le nombre de points de mesures est de 200.

LECTURE



La face avant permet entre autre de :

Autoriser la prise des points de mesures tout en combinant les 8 modes de stockage et de transfert correspondant aux états des boutons à glissière

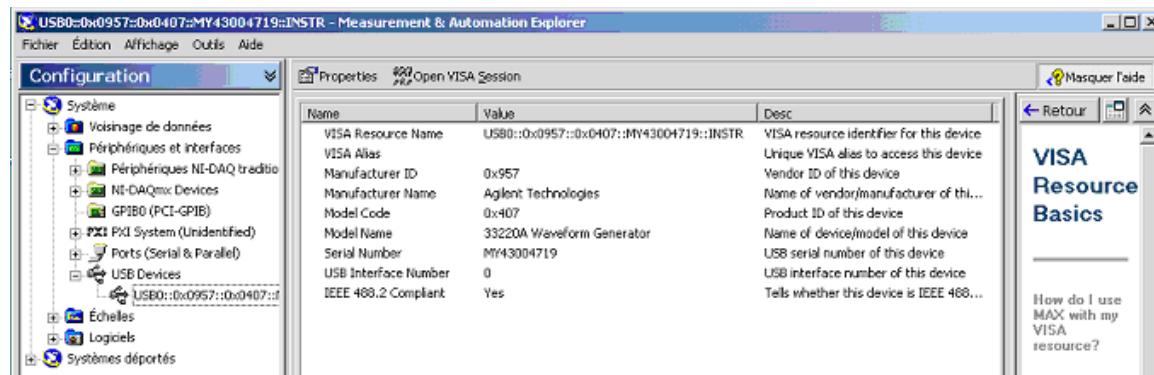
S'adapter aux différences de fin de message suivant que la communication se fait par GPIB ou par RS 232C

Préciser la durée du time out pour prendre en compte fait que l'acquisition de 1000 ou 2000 points de mesures peut prendre un temps non négligeable et que le calculateur doit patienter pendant tout ce temps là.

NB : la complexité du VI est évidente, la simplicité de sa mise en œuvre aussi ! C'est bien là l'intérêt de faire appel à un driver d'instrument.

PILOTAGE D'APPAREILS PAR PORT USB

INSTALLATION ET CONFIGURATION



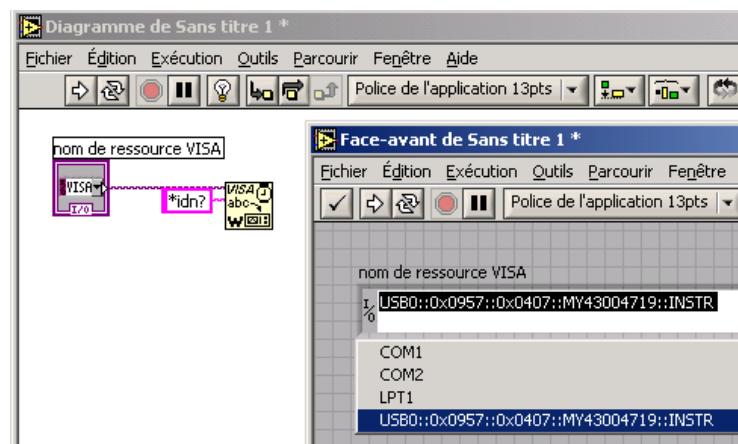
Il n'y a pas de configuration spécifique du port USB mais une installation particulière du périphérique. Pour cela, il est nécessaire d'exécuter le fichier d'installation fourni avec le périphérique. Cette opération DOIT être effectuée AVANT de relier l'appareil au PC par son cordon USB. Cette installation requiert des droits « administrateur ». L'usage ultérieur du périphérique est ensuite donné à l'utilisateur standard qui doit cependant installer le périphérique sur le port sur lequel il a été initialement configuré.

Une fois l'installation terminée le périphérique est visible sous MAX.

Il est conseillé de débrancher logiciellement le périphérique avant de le déconnecter du port USB

VISA ET USB

Il n'existe pas de fonctions spécifiques dédiées au contrôle du port USB. Lors de l'ouverture de la session Visa, il suffit de déclarer le numéro correspondant à l'appareil associé dans le contrôle « nom de ressource Visa »



Celui-ci est disponible directement dans la commande pour autant que le logiciel MAX ait été lancé et rafraîchi.

La session Visa se déroule ensuite sans autres spécificités. La communication est alors identique à celle que l'on développerait avec un bus IEEE 488.

LES DRIVERS IVI

La technologie IVI se veut plus sûre et plus portable que les drivers écrits en LabVIEW. Ils sont écrits en C et répondent à un cahier des charges très strict. Ils offrent notamment les avantages suivants :

Drivers génériques (Multimètre, Générateur, Oscilloscope...), SCPI

Drivers spécifiques

Possibilité de changer d'appareil et de driver depuis MAX, sans changer le code.

Mise en mémoire cache de l'état de l'instrument pour améliorer les performances : une machine d'états, implantée dans le driver, connaît en permanence l'état du périphérique et n'envoie que les commandes nécessaires.

Simulation simple, sans instrument, et sans carte d'interface ou plus sophistiquée (nécessite l'achat d'un toolkit)

Sécurité multithreads

Accès aux attributs de l'instrument

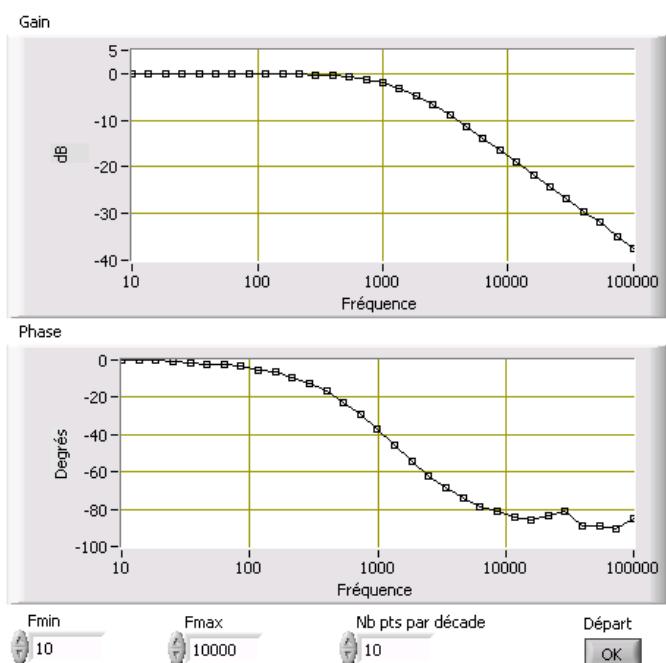
EXERCICE 14-3 : REPONSE EN FREQUENCE D'UN FILTRE

OBJECTIF: CONFIGURER UN GENERATEUR POUR FAIRE UN BALAYAGE POINT PAR POINT, RECUPERER SUR LES DEUX VOIES D'UN OSCILLOSCOPE LA VALEUR EFFICACE DES TENSIONS D'ENTREE ET DE SORTIE DU FILTRE.

On se propose de réaliser un système capable de relever la courbe de réponse en fréquence d'un système, par balayage de la fréquence du générateur d'attaque. Pour cela, les fréquences Fmin et Fmax sont entrées par l'utilisateur ainsi que le nombre de points par décades. A chaque point l'oscilloscope mesure le gain et la phase de la caractéristique de transfert.

FACE AVANT

En voici une représentation, l'indicateur à gauche est un tableau, le bouton déclenchement envoie un ordre GET.



DIAGRAMME

Vous utiliserez les drivers du générateur AG33220A et du scope AG54621A qui sont installés dans la machine et que vous trouverez dans la palette E/S Instruments»Drivers d'instruments. Vous aurez besoin principalement des éléments suivant (inspirez vous grandement des VI « Application example » Getting started ») :

Pour le générateur:



« **hp33120a Initialize.vi** » qui vérifie s'il s'agit bien de l'appareil, et qui fait une remise à zéro rétablissant ainsi l'ensemble des paramètres par défaut



« **hp33120a Configure Standard Waveform.vi** » pour définir l'amplitude (1V), la forme d'onde, la fréquence et l'offset.



« **hp33120a Configure Output Enabled.vi** » qui permet d'activer la sortie du générateur

Commencez par un code simple capable de programmer un sinus 1V 1000Hz.

Pour le scope:



« **ag546xx Initialize.vi** » identique au VI d'initialisation du générateur.



« **ag546xx Configure Channel.vi** » configure la sensibilité des voies, l'offset et les caractéristiques de la sonde (on ne changera pas la sensibilité en cours de balayage, elle sera égale à 2V)). Active la voie et configure le couplage d'entrée.



« **ag546xx Configure Acquisition Record.vi** » configure la base de temps, on prendra deux périodes par enregistrement.

Le déclenchement n'est pas modifié, les par défaut sont, déclenchement sur front positif au passage par zéro couplage DC voie 1.



« **ag546xx Initiate Acquisition.vi** » lance une acquisition (comme un appui sur **Single**).



« **ag546xx Fetch Waveform.vi** » rapatrie les données d'une voie dans un tableau ainsi que les éléments de la base de temps.

Tentez dans un premier temps d'acquérir le signal du générateur.

Pour le calcul du tableau de fréquences :

Le nombre de points de balayage est donné par

$$Nb_pt = (Nb_pt_p_dec \times (\log_{10}(F \max) - \log_{10}(F \min))) + 1$$

(Le +1 inclut la borne supérieure).

Le rapport entre deux fréquences successives est :

$$F_2 = \sqrt[Nb_pt_p_dec]{10} = F_1 \times 10^{\frac{1}{Nb_pt_p_dec}}$$

Pour le calcul de la réponse en fréquence :



« **Extraire une information mono fréquentielle** » retourne la phase et l'amplitude du signal (utilisez le bouton recherche de la palette de fonction pour localiser le VI). Il ne reste plus qu'à calculer $20 \log(Vs/Ve)$ et faire la différence des phases.

Enregistrez le VI sous le nom réponse en fréquence.VI et exécutez le.

FIN DE L'exercice 14-3

QUE VERIFIER SI LE PROGRAMME MINIMAL NE FONCTIONNE PAS ?

Si un périphérique ne fonctionne pas dans son environnement, il est conseillé d'écrire un 'petit bout de code' dit programme minimum, n'ayant rien avoir avec le programme (ou la chaîne de mesures) à déterminer mais effectuant une entrée sortie via l'interface incriminée. Cette approche permet de prendre du recul par rapport au problème avant de consulter une <hot line>. Il est en outre conseillé de :

Vérifier si, depuis le logiciel, la carte est reconnue : pour cela observer visuellement l'état des voyants REM, Lstn (listen) et Tlk (talker). Après un ordre d'écriture REM et Lstn doivent être allumés.

Vérifier compatibilité des adresses sur le périphérique et dans le code. Si l'adresse est bonne le voyant REM doit être allumé. Dans le cas contraire, Forcer éventuellement la ligne REN en envoyant la commande <Visa GPIB Contrôle REM>

Vérifier la compatibilité des terminateurs à l'émission et à la réception,

Changer le câble ou le brancher si ce n'était pas le cas !

Essayer des Mots de programmation simples qui doivent entraîner des changements de fonction et d'affichage. (préférer les déclenchements automatiques les plus simples).

Vérifier si le programme minimum fonctionne en mode pas à pas .Dans l'affirmative, introduire 'judicieusement' une temporisation ≈500ms derrière des tâches un peu longues à exécuter

Contrôler l'état du témoin de synchronisation ou de prise de mesure : si celui ci est toujours éteint, il y a vraisemblablement un problème de mode déclenchement,

Compatibilité Time OUT avec tâche longue asynchrone

Tester l'état d'adressage du périphérique (Lstn ou Tlk) avant toute nouvelle tâche,

Rechercher des exemples types dans la documentation papier ou sur Internet

QUE FAIRE SI UNE TACHE SPECIFIQUE NE FONCTIONNE PAS ?

Débrancher tous les autres périphériques,

Relire la documentation et en particulier les modes déclenchement,

Ecrire un programme à part qui teste cette fonction spécifique,

Rechercher une tâche précédente mal terminée,

Eteindre et rallumer le périphérique qui peut être bloqué. Cette méthode brutale vise efficacement son tampon de sortie.

Quitter LV7 ou dans un deuxième temps éteindre le PC,

Changer ordre d'allumage,

Lancer un logiciel espion sur le bus,

Remplacer le périphérique par un périphérique équivalent.

COMMUNICATIONS ET CONFIGURATION SERIE

PARAMETRAGES POSSIBLES

La liaison série est mode de communication répandu dans le domaine informatique. Initialement prévue pour connecter des terminaux distants, elle s'est ouverte, en raison de son caractère économique, au contrôle d'instruments.

Il n'existe pas de protocole permettant un paramétrage automatique de la transmission. Il est donc nécessaire de les régler soi même. Parmi ces paramètres citons :

La vitesse de transmission donne le nombre de bits par seconde, transmis par la ligne de communication (entre 110 et 56000 bauds).

Le nombre de bits de données (7 ou 8).

Eventuellement un bit de parité pour fiabiliser la transmission. Il s'agit d'un bit rajouté artificiellement aux bits constituant le message. Ce bit complémentaire, dit de parité, est forcé à 0 ou 1 de sorte que le nombre binaire transmis soit systématiquement pair (ou systématiquement impair), en anglais : even / odd respectivement.

Le nombre de bits de stop (1, 1.5 ou 2) définit une durée pendant laquelle le signal reste bas après une transmission.

Le contrôle de flux (handshake) qui peut exister, soit par le biais de lignes physiques (RTS/CTS), soit par un protocole logiciel (XON/XOFF), ou ne pas être géré.

Tous ces paramètres sont indépendants ; chacun d'entre eux peut rendre la liaison inopérante ! Ils doivent impérativement être configurés de la même façon pour les 2 partenaires de la communication série

CONNEXION MATERIELLE

Il existe deux normes principales de liaison série :

RS-232

La RS-232 est la liaison la plus commune sur les PC. Elle utilise des tensions référencées par rapport à une masse commune, ce qui la rend sensible aux perturbations et limite les longueurs de câbles ($| \approx 15\text{mètres}$). Beaucoup d'appareils de mesures en sont équipés.

RS-449, RS-422, RS-423

La RS-449, et ses variantes RS-422, et RS-423 utilisent des paires différentielles qui permettent des communications sur de longues distances. Elles ont la faveur des fabricants de contrôleur de process (capteurs intelligents, régulateurs, automates) répartis dans des usines en environnement très bruité. ($| \approx 1\text{km}$ si combiné avec une boucle de courant 4-20mA)

CABLAGE RS-232

Deux prises coexistent pour la connexion RS232, la DB25 et la DB9. Les appareils connectés sont soit des DTE (Data Terminal Equipment), soit des DCE (Data Communications Equipment). Les DTE sont des calculateurs, des terminaux ou des appareils de mesure, ils ont des prises mâles, les DCE sont des modems, des tables traçantes...,

ils ont des prises femelles. Il faut croiser les fils de transmission et de handshake lorsque l'on connecte deux DTE ou deux DCE entre eux. Généralement les câbles Mâle/Femelle sont câblés fils à fils, les Femelle/Femelle et Mâle/Mâle sont croisés.

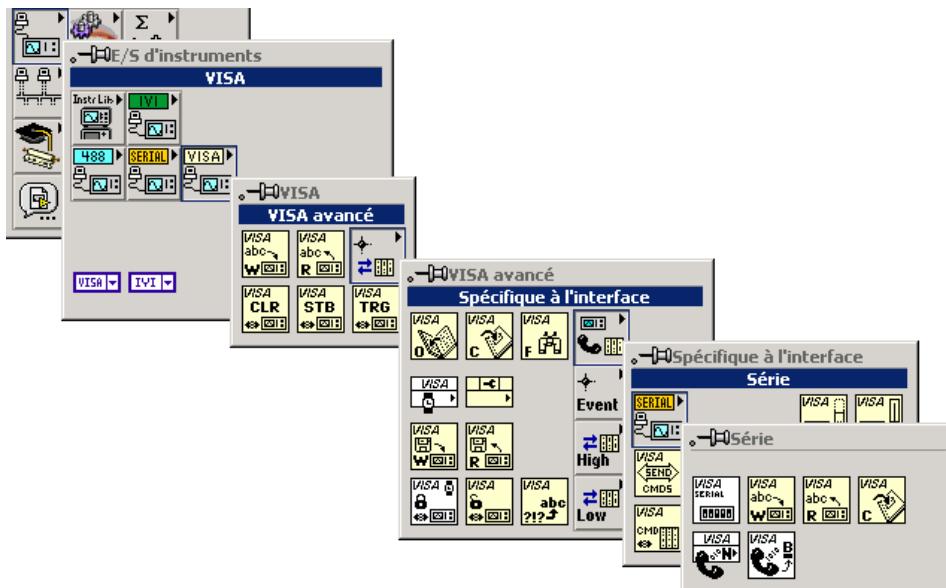


Note Pour simplifier les choses, les fiches DB-9 et DB25 ont les broches 2 et 3 inversées (2=Tx et 3=Rx sur DB25 alors que 2=Rx et 3=Tx sur une DB9 !). Bref c'est pas toujours facile pour câbler tout ça, mais de nombreux sites existent sur le net pour faciliter la tâche. En cas d'erreur, cela ne fonctionne pas mais cette situation n'est dangereuse ni l'interface ni pour le périphérique.

VISA ET LIAISON SERIE

VISA SPECIALISES

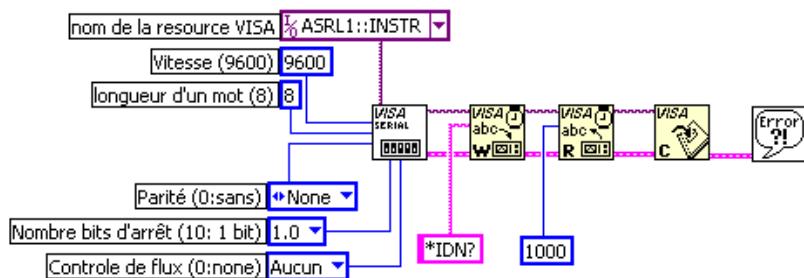
Il existe dans la palette des fonctions VISA avancées une sous palette dédiée à la liaison série.



Cette palette regroupe

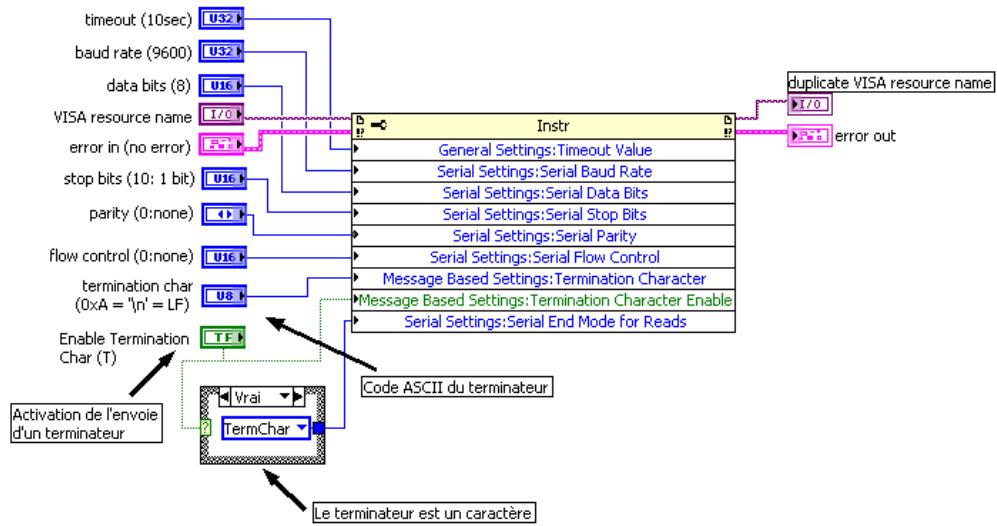
des VIs non spécifiques (en jaune) et trois fonctions dédiées : à la configuration de la liaison, à la lecture du nombre d'octets et à l'envoie d'une pose.

L'exemple suivant reprend le premier exemple pour l'interface GPIB revue et corrigée pour la RS. Comme vous pouvez le constater, la différence n'est pas marquante hormis la configuration de la liaison. Par défaut l'envoie d'un caractère terminator est activée.



NŒUD DE PROPRIÉTÉ

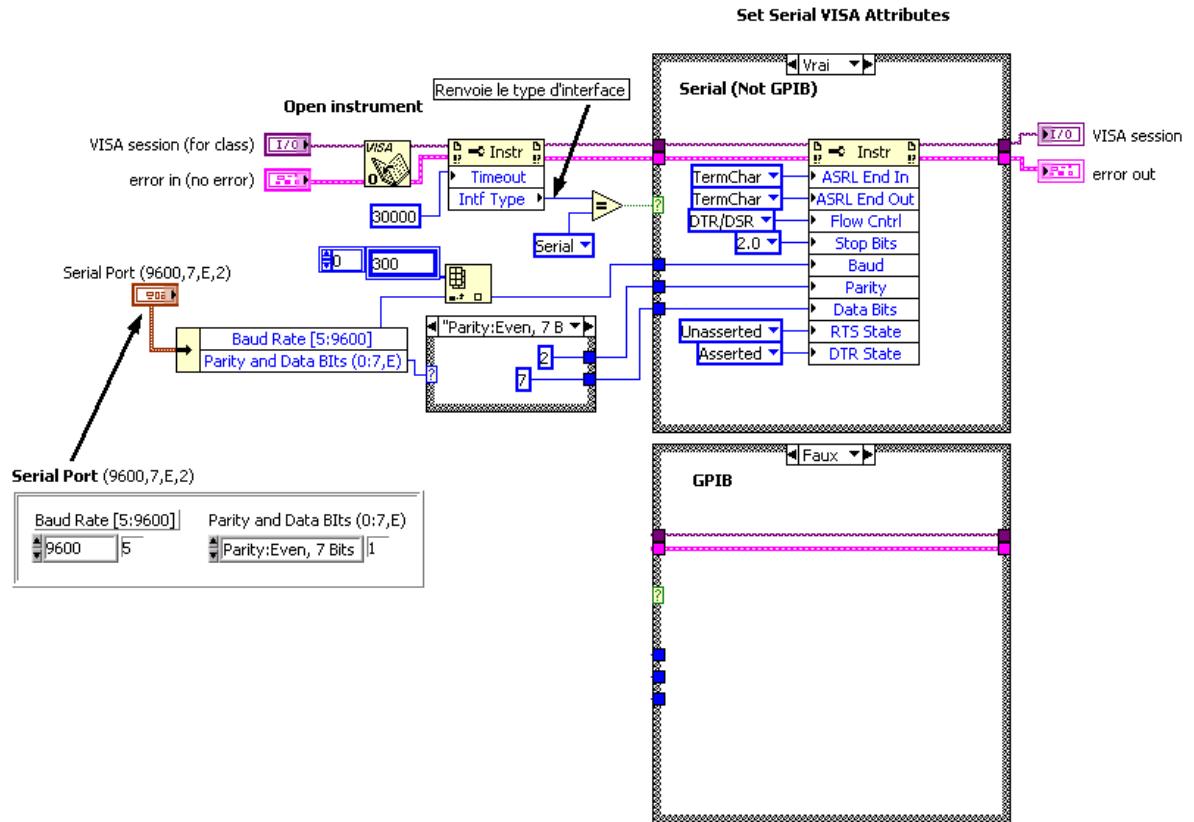
Si vous regardez comment est écrit le VI de configuration vous verrez ceci dans le diagramme :



L'élément central de ce VI est un nœud de propriété. De façon générale, un nœud définit l'ensemble des propriétés d'un objet quelconque ; son contenu dépend donc du type d'objet auquel il est connecté. Par exemple s'il s'agit d'un élément de la face avant, la propriété **Visible** doit être dans la liste. Il est possible de redimensionner la liste, de choisir les items uns à uns de les paramétriser en lecture ou en écriture.

Les nœuds de propriétés sont disponibles dans la palette **E/S d'instruments»VISA »VISA avancées** ainsi que dans la palette **Contrôle d'applications**.

L'exemple ci-dessous est extrait du pilote du HP34401. Il montre comment la spécificité du support (RS ou du GPIB) est assurée lors de l'initialisation de l'objet Visa.



En fonction du type d'interface, un paramétrage spécifique peut être introduit. Une fois ce VI exécuté, plus aucune autre modification n'est nécessaire à la prise en compte du mode de transmission (GPIB ou série) magique...

15. LE MODULE DATALOGGING & SUPERVISORY CONTROL

INTRODUCTION

Le module DSC est conçu pour faciliter la communication, la surveillance et l'enregistrement de variables distribuées sur un réseau. Les domaines privilégiés sont :

L'automatique via des API

La télésurveillance et acquisition de données (SCADA pour Supervisory Control And Data Acquisition)

La surveillance d'équipements distants.

LE MODULE DSC POUR LES NULS

Le grand intérêt du module DSC est sa capacité de connections à des variables distantes, leur surveillance et leur enregistrement. Le module DSC offre une connectivité vers :

Des clients et serveurs OPC (Object Linking and Embedding (OLE) for Process Control) technique de communication utilisés en industrie pour la communication avec les API et d'autres systèmes déportés

Des clients et serveurs EPICS (Experimental Physics and Industry Control System) technique surtout utilisés en instrumentation scientifique dans les grands instruments (physique des particules, astronomie...)

Un serveur d'entrée/sortie au protocole ModBus, protocole entre un maître et plusieurs esclaves, utilisé très largement en industrie pour la communication vers les API et vers les capteurs, actionneurs intelligents.

Le concepteur peut aussi programmer des drivers et des serveurs d'E/S

Le cœur du module réside dans la communication entre les variables publiées par les serveurs qui viennent d'être énumérés et les variables partagées de LabView.

Une fois installé, le module DSC fournit de nouvelles fonctionnalités aux variables partagées :

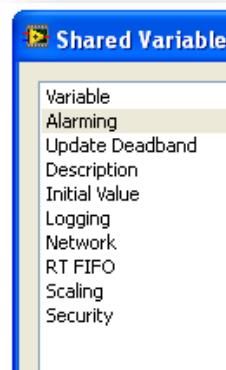


Figure 15-1 Nouveaux items des variables partagées

CONNECTION A UNE VARIABLE "TERMINALE"

Pour connecter une variable partagée à une variable « terminale » (une variable d'un automate, d'un régulateur,...) il faut tout d'abord créer un serveur E/S dans une librairie du projet.

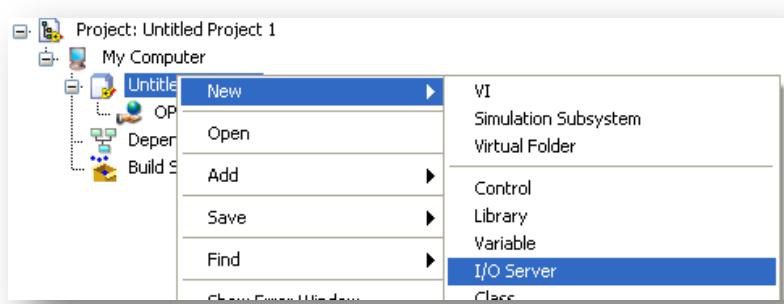


Figure 15-2 Connection d'une variable partagée à un serveur

Une fenêtre apparaît pour créer un nouveau serveur d'E/S.

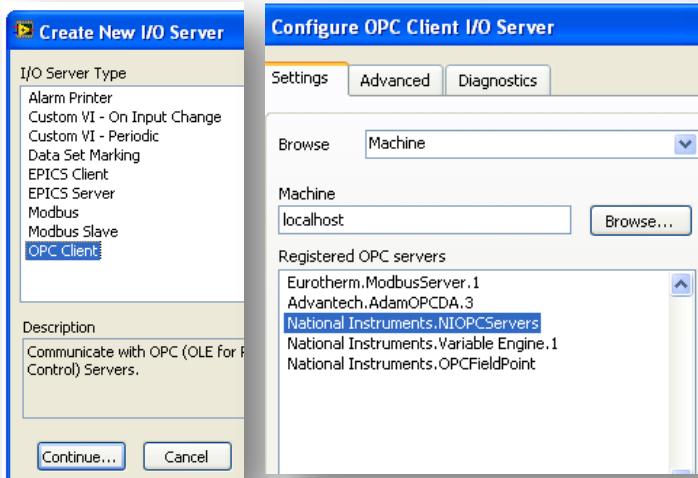
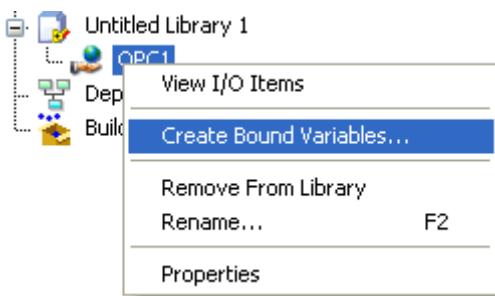


Figure 15-3 Sélection du type de serveur

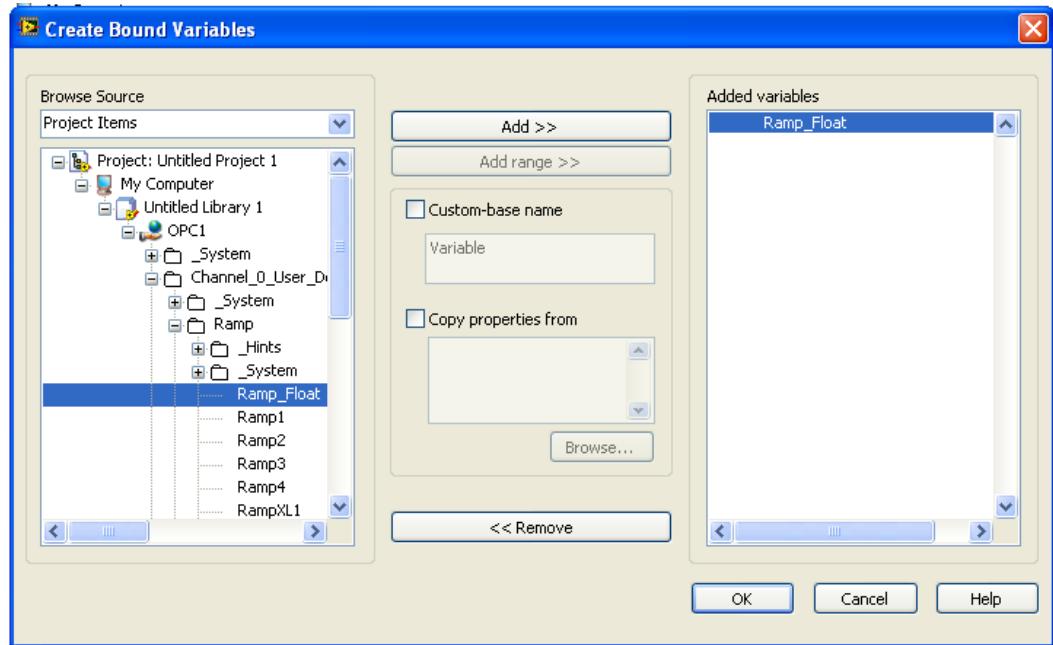
National Instrument fournit un serveur OPC de démonstration (simulant un API) qui doit, si l'installation est correcte, être démarré. Nous l'utiliserons pour cette présentation.
Ce serveur doit se retrouver dans la liste des serveurs OPC enregistrés sous le nom **NationalInstruments.NIOPCServers**
Update rate définit la fréquence

d'interrogation du serveur OPC, entrez 100 ms, Deadband le pourcentage de variation de la variable provoquant sa mise à jour.

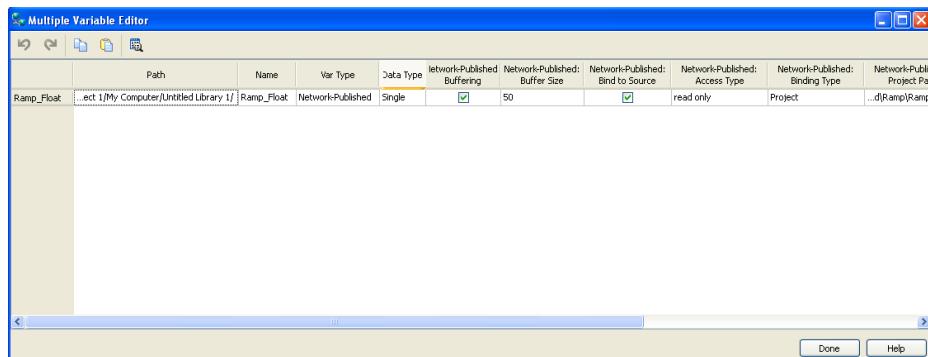
Le lien avec les variables publiées par le serveur OPC est fait, il faut maintenant lier des variables partagées à certaines de ces variables « terminale ». Par un clic droit sur l'icône du client OPC, sélectionnez **Create Bound Variables**.



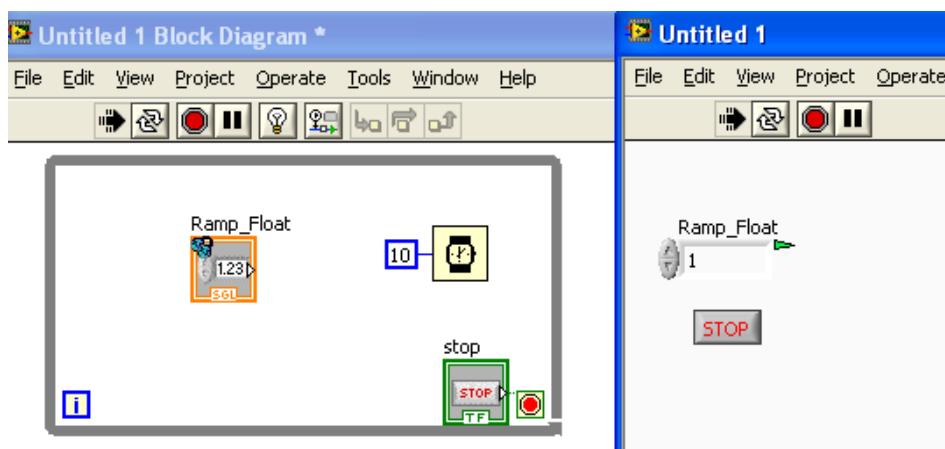
Ceci va créer et lier des variables partagées à certaines variables "terminales"



Sélectionnez une des nombreuses variables publiées par le simulateur OPC (par exemple la rampe), après avoir fermé cette fenêtre, vous voyez la fenêtre de l'éditeur de variable multiples, qui permet de changer les propriétés d'un ensemble de variables en une seule fois.



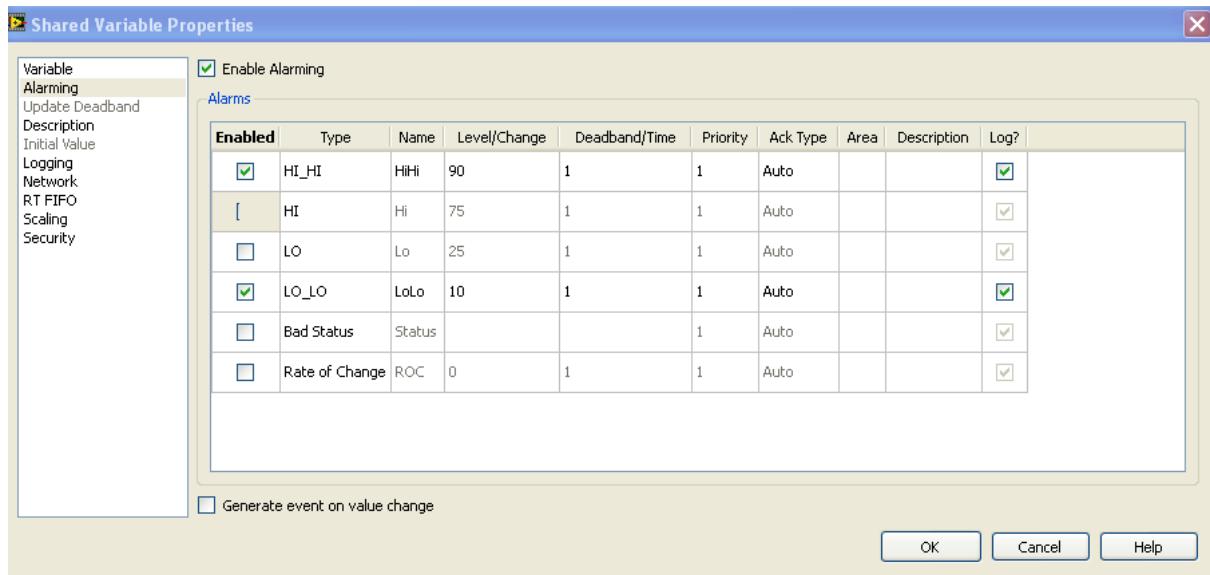
Cliquez sur OK, la variable est créée. Un glisser/déposer dans la face avant d'un VI et le tour est joué, une boucle de répétition sur le diagramme pour obtenir une exécution continue, et la variable en face avant suit scrupuleusement la variable publiée par le serveur OPC (ce pourrait être la consigne de température d'un régulateur).



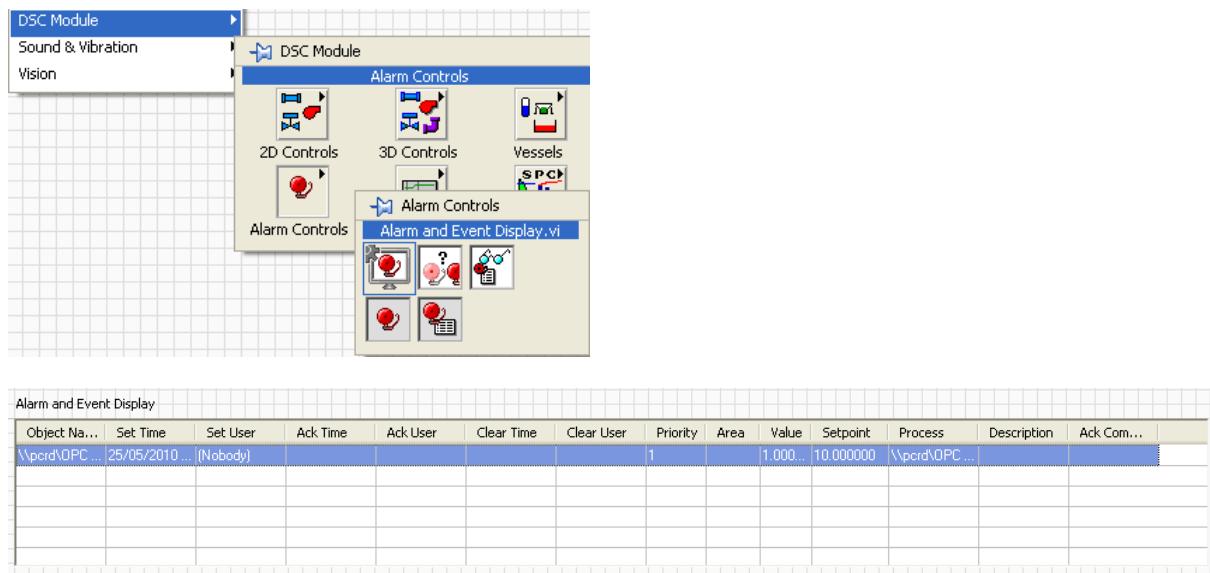
Le triangle en haut à droite de l'indicateur numérique indique que la variable est liée à une source, il est vert si la liaison est valide, rouge si elle est coupé.

GESTION D'ALARMS

Les alarmes sont directement gérées par DSC, chaque variable peut se voir affecter des alarmes sur des niveaux « **trop bas, bas, haut, trop haut** », mais aussi sur une vitesse de variation. Ces niveaux sont réglés dans la page **Alarming** des propriétés de la variable.



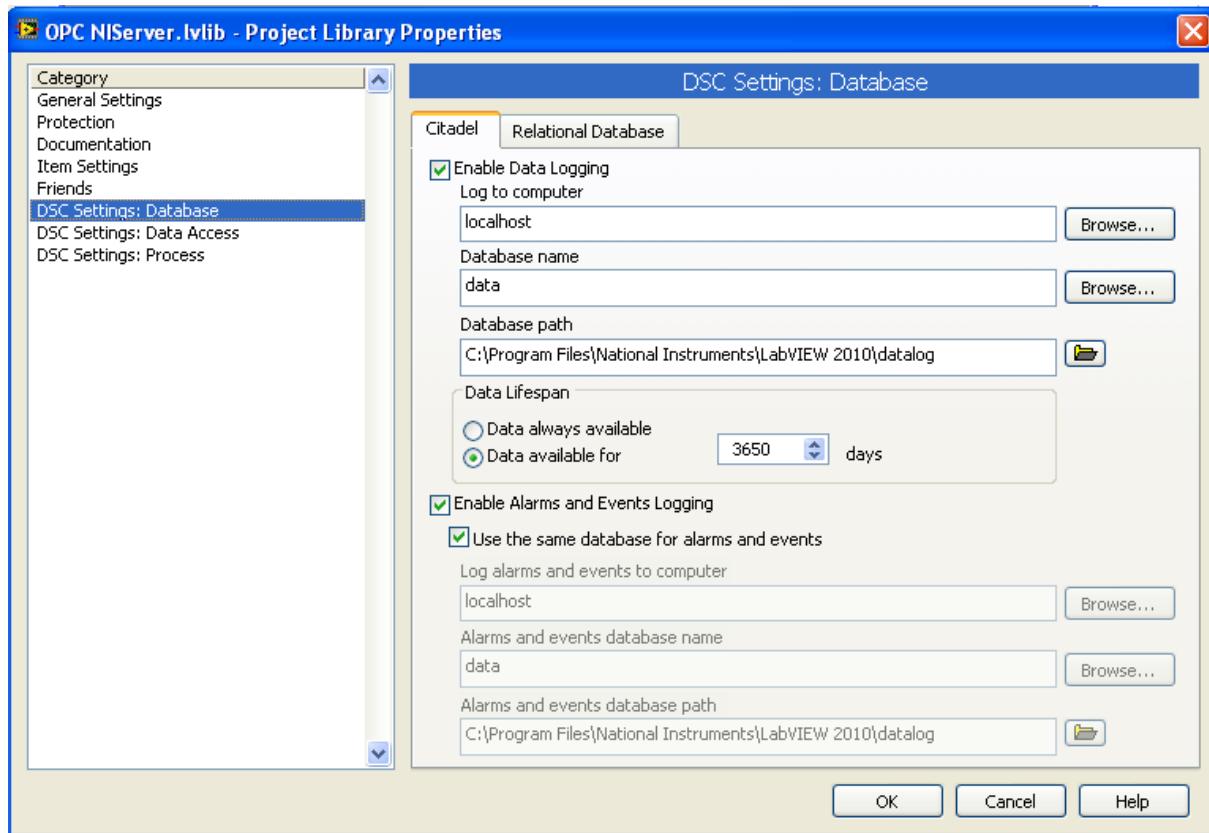
Une fois ces valeurs d'alarmes fixées, DSC compare à ces limites la valeur de la variable à chaque mise à jour. Un contrôle spécifique permet de visualiser les alarmes sur la face avant du superviseur « **Alarm and Event Display** »



D'autres fonctionnalités sont disponibles pour gérer les alarmes, mais nous en sommes au début...

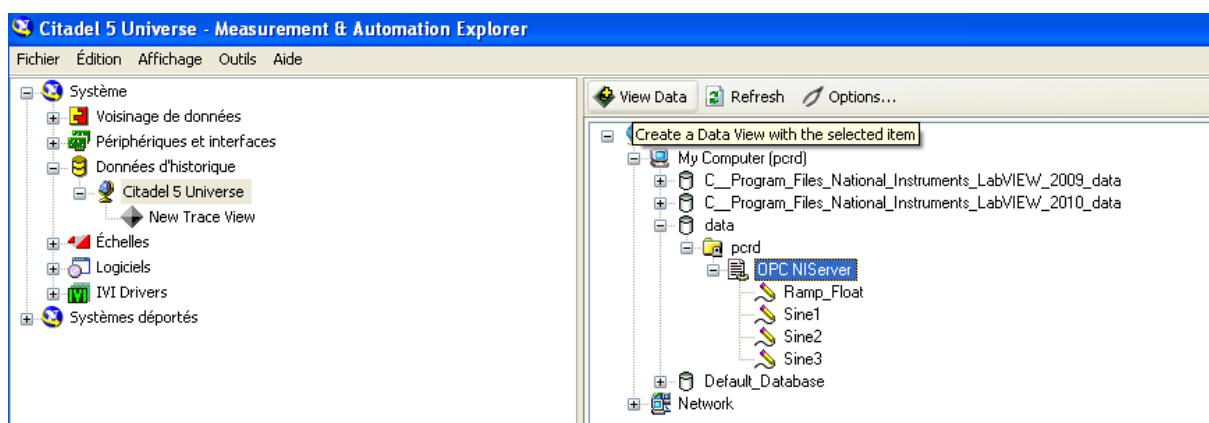
ENREGISTREMENT DES DONNEES

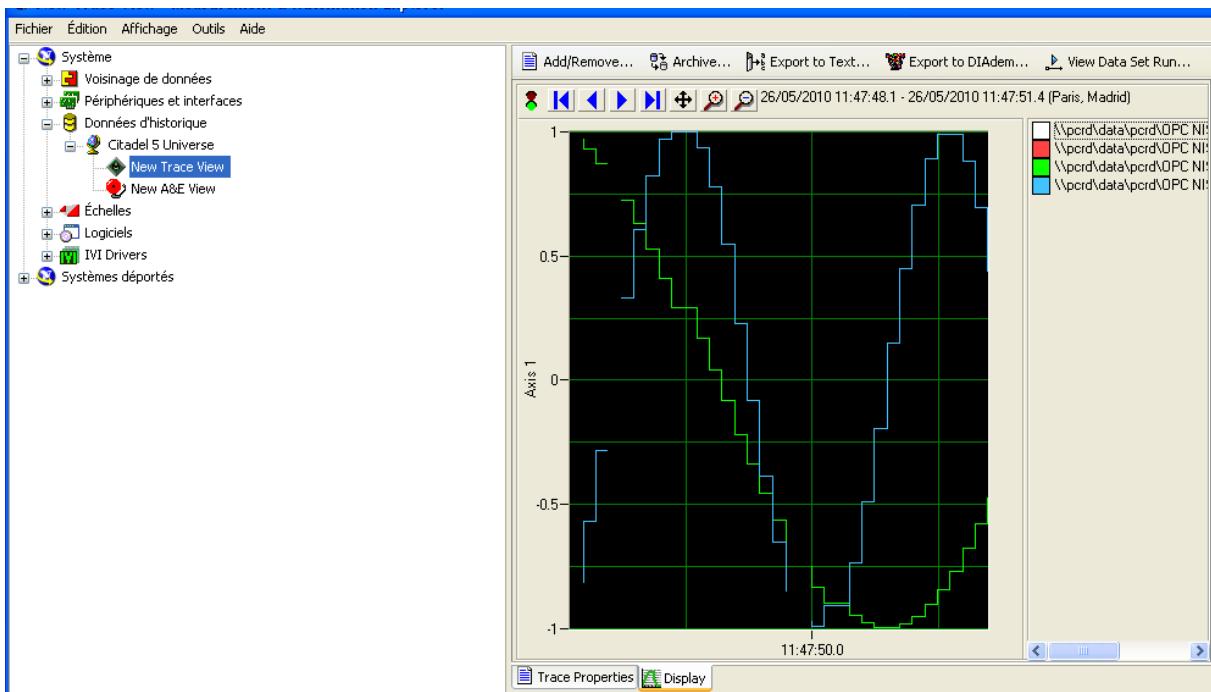
DSC gère une base de données propriétaire appelée **Citadel** ou accède à des bases Microsoft Access, MySQL, Oracle, PostgreSQL, et SQL Server. L'emplacement de la base est une propriété de la librairie contenant les variables.



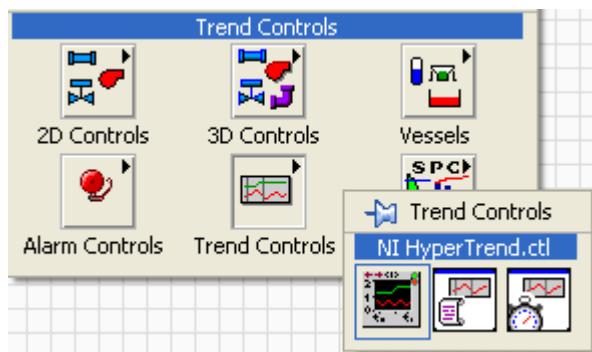
L'enregistrement des données à lieu à chaque mise à jour de la variable par le serveur d'E/S, il est possible d'améliorer les performances en jouant sur la résolution, sur la valeur de la zone morte... NI recommande de ne pas dépasser 5GB pour une base de taille moyenne. Les événements et alarmes sont quant à eux enregistrés dans une base Microsoft SQL Server 2005 Express Edition qui est limité par sa licence à 4GB (la limitation peut être levée par l'achat de la licence complète).

Votre système enregistre vos données dès maintenant, que LabView soit en mémoire ou non ! Dès qu'une donnée est actualisée, la base de données sera mise à jour (attention à ne pas lancer une trop grande quantité de processus en tâche de fond par mégarde). MAX permet la visualisation de ces données, dans « **Historical data** » vous pouvez visualiser les bases de données, les archiver, supprimer...





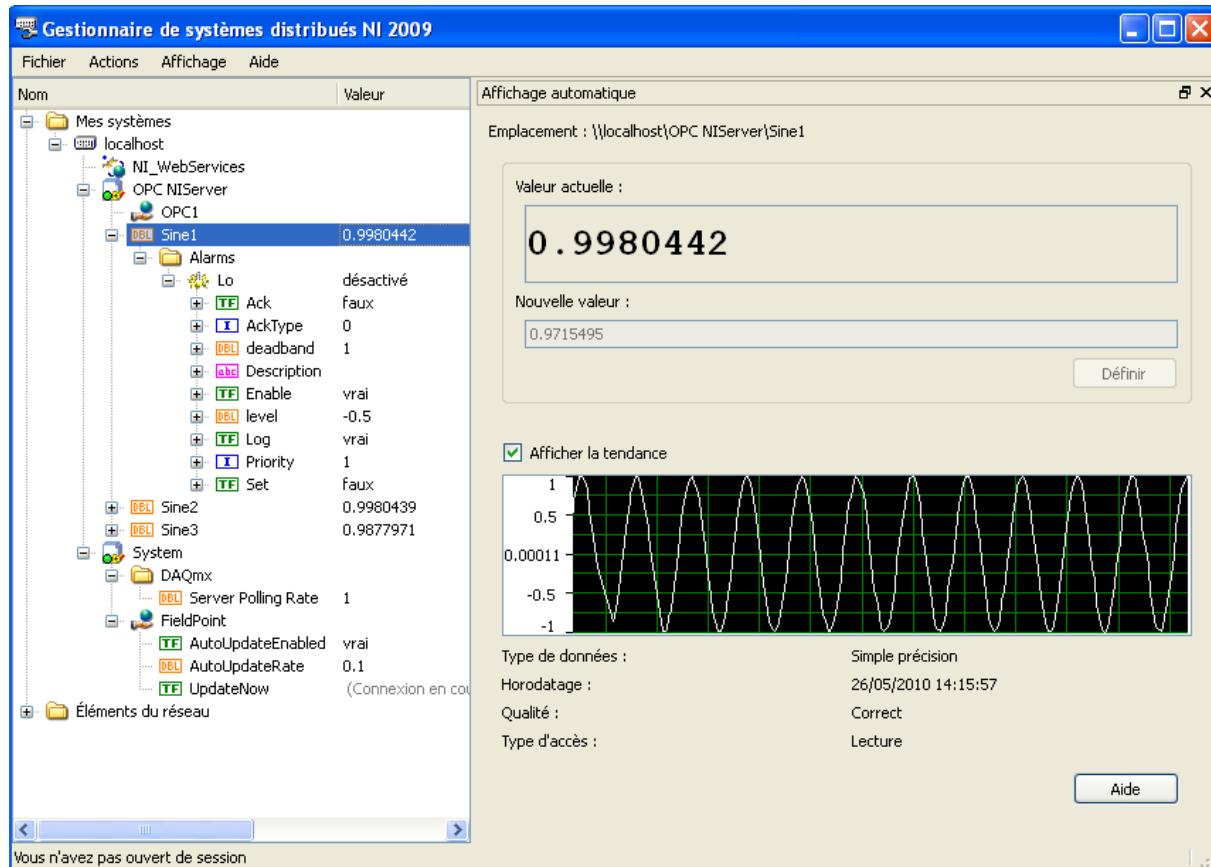
Pour visualiser le contenu d'une base dans votre programme LabView, DSC propose un contrôle nommé « **HyperTrend** » cet ActiveX (identique à celui de l'onglet **Display** de **MAX**) permet de naviguer dans la base de données **Citadel** sélectionner les traces souhaitées, les regrouper....Le contrôle fonctionne de lui-même sans besoin d'autre code !



GESTIONNAIRE DE SYSTEMES DISTRIBUES

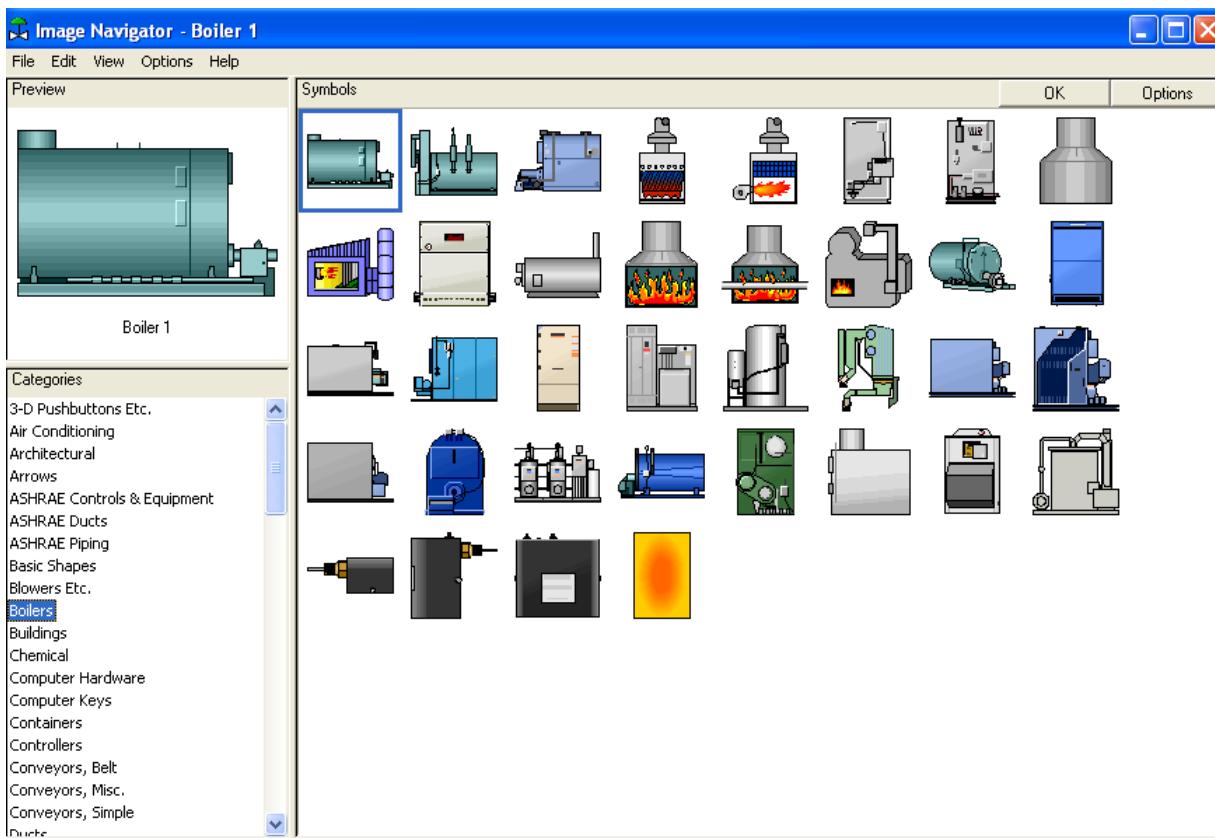
Dans le menu « **Démarrer/programme/nationalInstruments** » se trouve le gestionnaire de systèmes distribués. Cet outil permet entre autre de :

- Créer ,surveiller des variables partagées ou réseau et configurer leurs alarmes.
- Créer ou surveiller des processus.
- Créer ou surveiller des serveurs d'E/S.
- Enregistrer et surveiller les historiques, alarmes et événements.



INTERFACE HOMME MACHINE

DSC intègre des contrôles 2D et 3D pour les applications industrielles (tuyau, vannes, ventilateurs, réservoirs) et différents graphes (visualisation des limites, statistique). DSC fournit également une bibliothèque d'objet dans « **Tools/DSC module/Image Navigator** » qui peuvent être utilisés pour créer des contrôles personnalisés ou de simples décos.



POUR ALLER PLUS LOIN

Cette introduction illustre les possibilités du module DSC. Dans un système industriel cette façon de procéder n'est pas possible (définir de façon statique l'ensemble des paramètres).

Un système industriel doit pouvoir :

- Démarrer et stopper un processus d'E/S
- Faire des associations dynamiques de variables
- Choisir d'enregistrer ou non certaine variables
- Définir par programme les seuils d'alarmes
- Gérer les stratégies de sécurité réseau....

Ce sera la suite du programme...

ARCHITECTURE D'UN SCADA (SUPERVISORY CONTROL AND DATA ACQUISITION)

CLIENT / SERVEUR

La supervision (interface utilisateur) d'un côté, l'acquisition et le contrôle (Acq, traitement, alarme...) de l'autre sont souvent associés aux termes client et serveur dans les applications distribuées.

Dans un sens plus général :

Le serveur est la partie du système qui collecte les données et coordonne les transactions entre clients.

Le client est la partie du système qui utilise ces données.

C'est l'approche à retenir pour le développement d'applications DSC.

Le **serveur** se compose de:

Le SVE (**Shared Variable Engine**) qui contient des variables partagées (acquisent via OPC, Modbus, Serveur E/S..).

Les VIs d'analyse ou de contrôle (optionnel si le SVE suffit ->alarme, looging)

Il tourne de manière indépendante et n'attend pas de connexion avec les clients, son indépendance de IHM assure en partie sa robustesse.

Les **clients** reçoivent les données et les mettent à disposition des utilisateurs (Affichage, statistiques, analyses..). Ils communiquent des valeurs de consignes au serveur.

Les variables partagées sont le cœur d'une application DSC

LIBRAIRIES ET PROCESSUS

Dans DSC les librairies définissent des processus (au sens module indépendant et autonome). Elles contiennent la configuration de la base de données, des variables partagées et la configuration des serveurs d'E/S. Elle est capable de gérer la connexion entre le serveur d'E/S et le moteur de partage de variable, qui lui-même, assure le contrôle de la qualité des variables, la surveillance des alarmes et l'enregistrement.

Si dans une application industrielle vous avez plusieurs lignes de production (connectée ou non) chacune fera partie d'une bibliothèque unique.

Une fois déployée dans le SVE, la librairie peut être soit en exécution, soit en attente d'exécution, cet état est conservé même après un redémarrage du système. Le SVE est un processus système (**tagsrf.exe**) qui démarre toutes les librairies activées dès le démarrage système (prendre gare à ce que la partie opérative soit en sécurité au démarrage de l'ordinateur).

LE MOTEUR DE VARIABLE PARTAGEES

TYPES DE VARIABLES PARTAGEES

Lors de la création d'une variable vous pouvez utiliser :

-  Network published Une « **network-published shared variables** » dans une application réseau ou pour lier les variables partagées à une source. Elle est le seul type de variable partagée qui prend en charge les options du module DSC telles que la liaison de données, journalisation, alarmes, paramètre de valeur initiale et sécurité.
-  Single process Une **single-process shared variables** pour les applications qui s'exécutent entièrement sur une seule cible.
-  FIFO Une **Real-time FIFO-enabled shared variables** pour mettre en œuvre une architecture déterministe de communications entre le programme hôte et de la cible RT.

UTILISATION DES VARIABLES PARTAGEES

Dans la suite nous ne parlerons que des variables publiées (avec FIFO ou non). Pour fournir cette fonctionnalité supplémentaire, l'implémentation interne de la variable est beaucoup plus complexe celle d'une variable globale. Leur mise en œuvre nécessite quelques recommandations pour obtenir les meilleures performances.

LE MOTEUR DE PUBLICATION NI PSP

Les variables sont publiées par NI-PSP (**NI Publish-Subscribe Protocol**). NI-PSP est basé sur un protocole UDP (**User Datagram Protocol**), il tire avantage du mode non-connecté d'UDP, cependant, NI-PSP garantit la livraison des paquets.

Les variables doivent être déployées sur le réseau pour tous les systèmes distants qui hébergent les valeurs de variables. Lorsque vous écrivez sur un nœud de variable, LabVIEW envoie la nouvelle valeur au SVE (**Shared Variable Engine**) qui a déployé et héberge la variable partagée (Il y a un SVE sur chaque cible). Le SVE publie ensuite cette valeur pour obtenir la valeur de mise à jour. Ce déploiement est automatique pour les variables statiques (celle définies dans le projet).

ACCES AUX VARIABLES PARTAGEES

Il existe deux possibilités d'accès aux variables partagées :

Accès statique via :

Un nœud de variable

Accès dynamique via :

Une liaison avec un objet de la face avant

Une liaison par l'URL de la variable

Un nœud d'événement

ACCES STATIQUE

Un Glisser/Déposer d'une variable partagé de la librairie au diagramme crée un nœud statique rattaché à cette variable.



Cette simplicité a cependant quelques inconvénients.

NI ne recommande pas l'utilisation plus de 100 nœuds statiques dans une application.

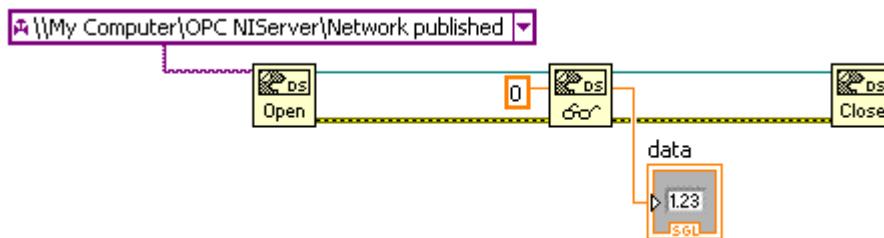
Comme leur nom l'indique, il n'est pas possible à l'exécution de changer de source (par ex. changer un API en panne)

Leur utilisation est difficile dans les sous Vis dont il est impossible de réutiliser le code dans d'autres applications, les mises à jours de noms entre projets peuvent entraîner des sacs de nœuds (au propre comme au figuré) difficiles à comprendre !

ACCES DYNAMIQUE

ACCES PAR « DATA SOCKET » :

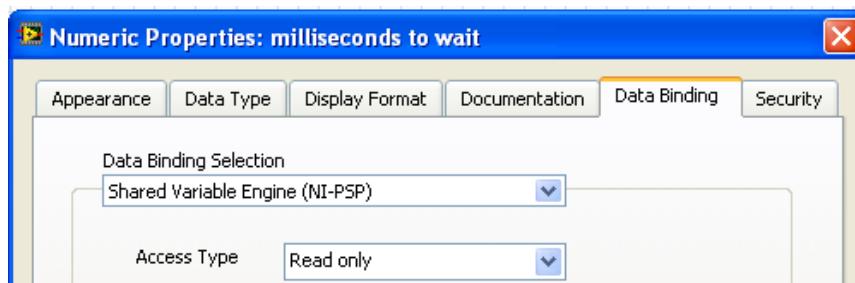
Utilise les classiques OPEN/READ/WRITE/CLOSE, l'accès est semblable au nœud de variables, mais l'URL de la variable partagé est en entrée ce qui permet de reconfigurer la source à l'exécution.



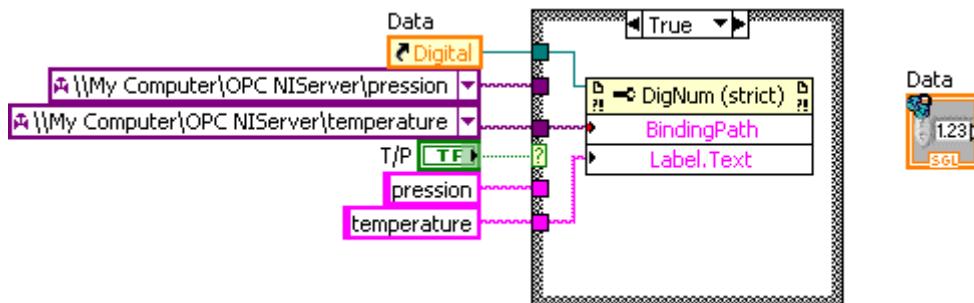
Notez qu'il faut plusieurs ms pour assurer la connexion à l'URL, il est donc préférable de refermer cette connexion que lorsque la variable n'est plus utilisée. Les Vis « **datasockets** » et « **Shared variables constants** » sont dans la palette « **Shared variables** » du module DSC

ACCES PAR « FRONT PANEL BINDING »

Tous les objets de la face avant ont une propriété « BindingPath » qui permet de les lier à une URL. Pour cela ils doivent au préalable (lors de leur création) être configurés comme liés à une variable partagée.



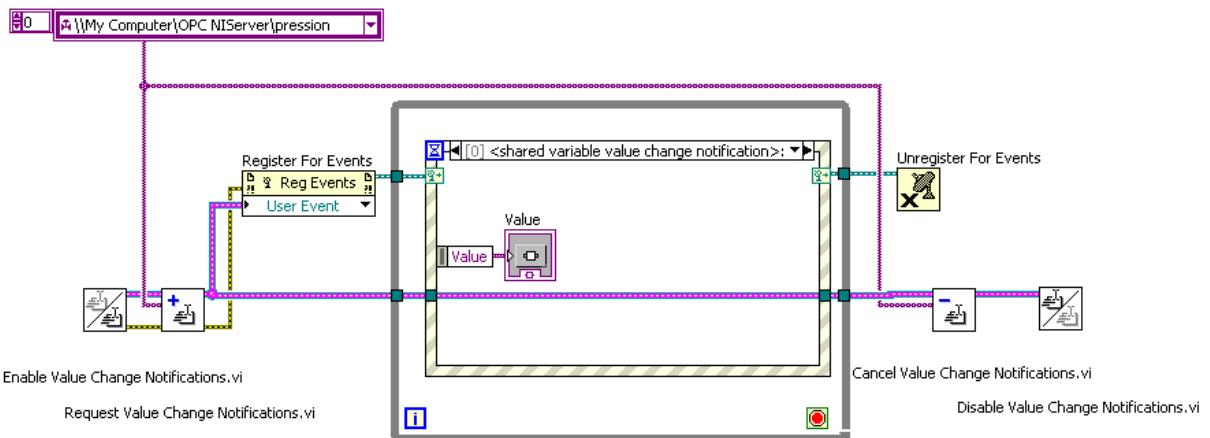
C'est la méthode idéale pour afficher la valeur de variables sur une IHM, ceci permet entre autre d'afficher sur un même indicateur les données provenant de sources différentes en fonction de l'état du programme, évitant ainsi une multitude d'indicateurs cachés ou affichés pour chaque variable à l'écran.



Les nœuds de propriété sont dans la palette « **programming/application Control** » ou bien clic droit sur le control et « **Create/property node** ».

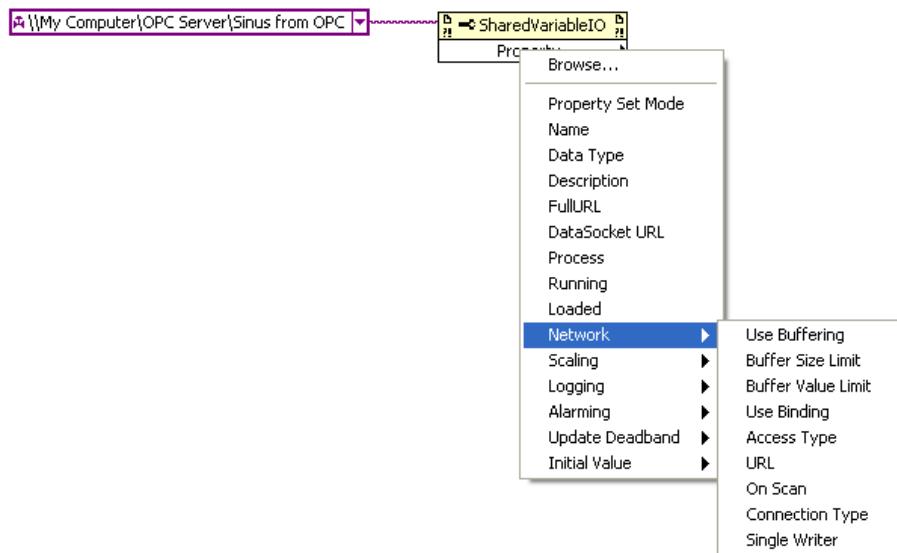
ACCES PAR « SHARED VARIABLE EVENT STRUCTURE SUPPORT »

DSC vous permet d'enregistrer les événements et de les traiter dans une structure d'événement. Cette méthode est particulièrement adaptée aux systèmes possédant déjà une structure événementielle sur l'IHM

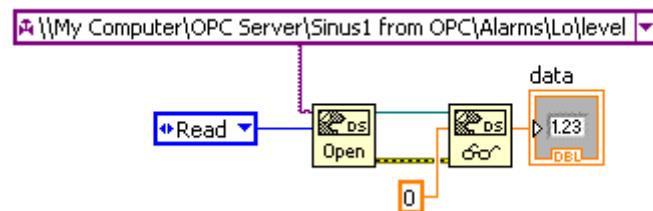


NŒUDS DE PROPRIÉTÉS DES VARIABLES PARTAGEES

L'ensemble des propriétés définies lors de la création d'une variable partagée peuvent être redéfinies par programmation dans un nœud de propriétés



Ou en utilisant directement l'URL de la propriété, par exemple pour lire le niveau de l'alarme basse :



ALARMES ET EVENEMENTS

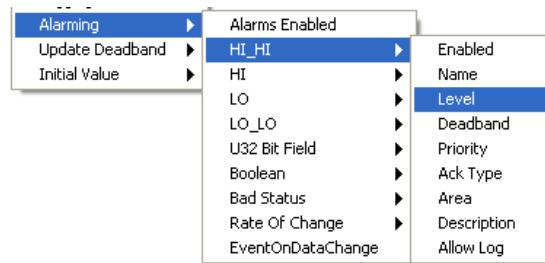
TYPES D'ALARMS

Les alarmes permettent générer des événements basés sur des valeurs particulières des données. DSC évalue la valeur d'une variable partagée pour déclencher une alarme et envoie des notifications lorsque la valeur est au-dessus, en dessous de limites prédefinies, ou encore si la vitesse de variation n'est pas atteinte. Un niveau de priorité permettra, par exemple de traiter les alarmes les plus élevées en priorité et les plus faibles en « Warning ».

Les alarmes sont remises à zéro, soit automatiquement (elles ne durent que le temps de dépassement du seuil), soit par programme (elles perdurent jusqu'à son acquittement)

CHANGEMENT DES PROPRIETES D'ALARMS

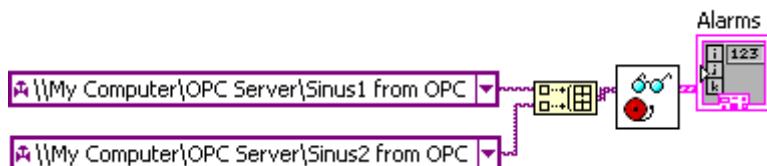
Pour changer, par exemple le niveau d'une alarme, activer tel type d'alarme, déposez un nœud de propriété sur la variable, puis :

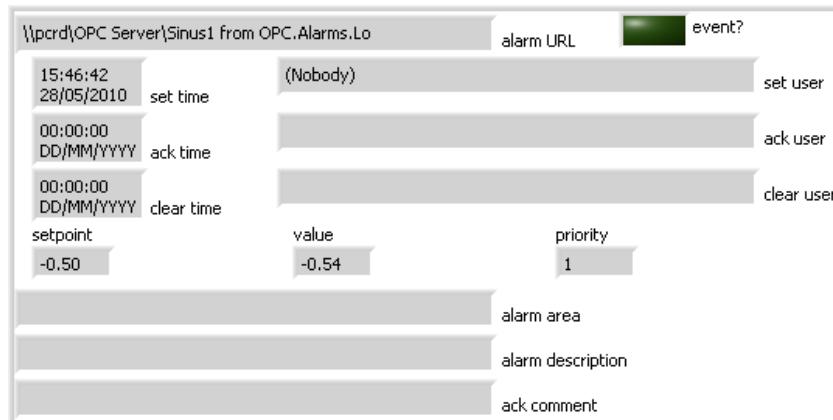


RECUPERATION ET VISUALISATION DES ALARMES

Il existe plusieurs moyens pour récupérer les événements d'alarme par programme (Palette : **Alarm and Event**).

Le plus simple est d'utiliser « **Read Alarms** » qui prend en entrée un tableau d'URL de variables partagées et rend en sortie un Tableau d'Alarms



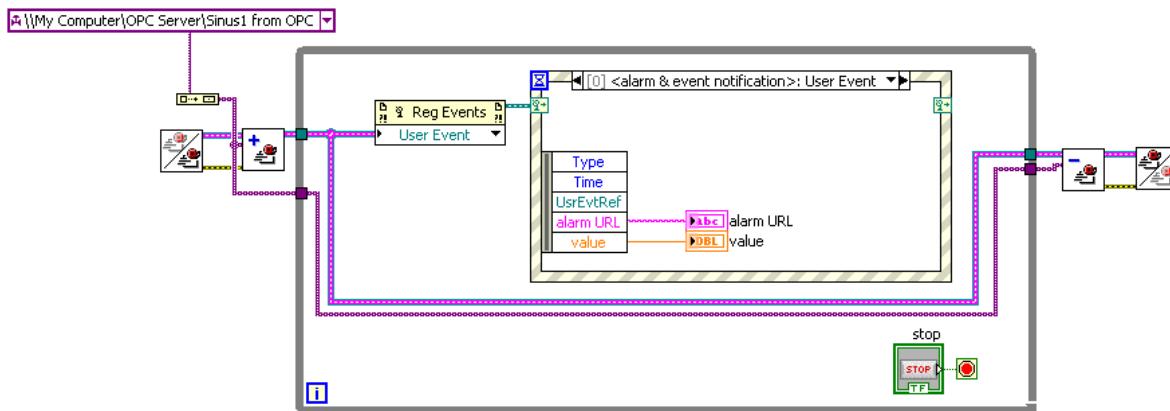


Si votre IHM utilise déjà une structure d'événement, il est préférable d'y enregistrer l'activité des alarmes.

Les Vis suivant permettent d'activer et désactiver la gestion des alarmes par des structures événement :



Et l'exemple suivant vous montre comment les employer :



ALARMS UTILISATEURS

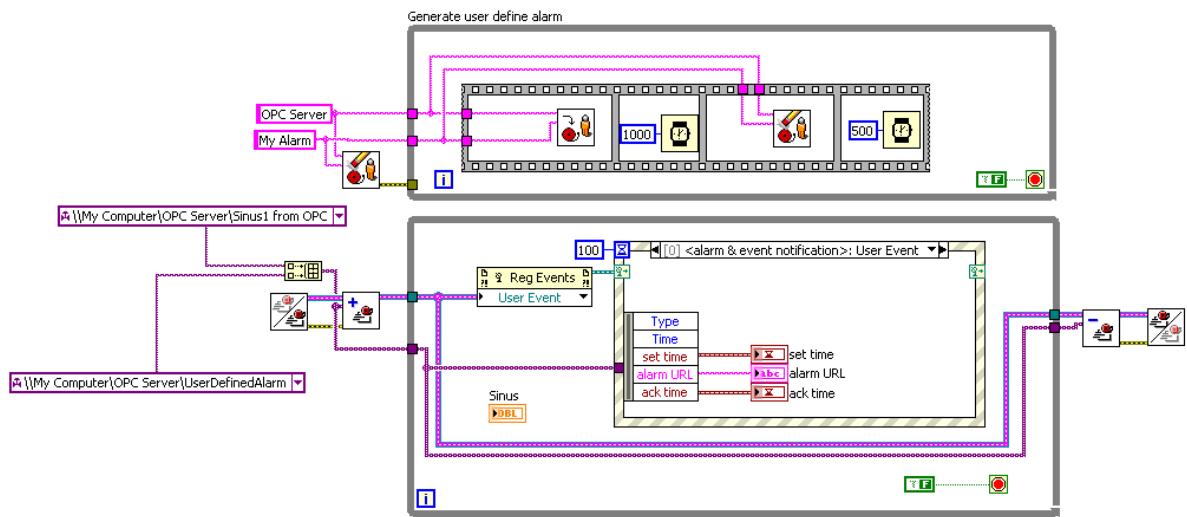
Les alarmes que nous avons utilisées étaient générées directement par le SVE, l'utilisateur peut aussi déclencher des alarmes. Pour cela, il faut simplement définir le nom de l'alarme et le processus (librairie) à laquelle elle est rattachée. Un **Set** et un **Clear** active et désactive les alarmes utilisateurs.



L'entrée “**process name**” détermine le second élément de l'URL de l'alarme qui est au format:

`\computer_name\process_name\UserDefinedAlarm.alarm_name`

L'entrée “**alarm name**” détermine le dernier élément de l'URL de l'alarme. Notez bien que toutes les alarmes utilisateurs s'appellent « **UserDefinedAlarm.xxxxxx** ». L'exemple ci-dessous reprend le précédent en ajoutant la génération et l'effacement d'alarmes utilisateurs et leur affichage.



BASE DE DONNEES

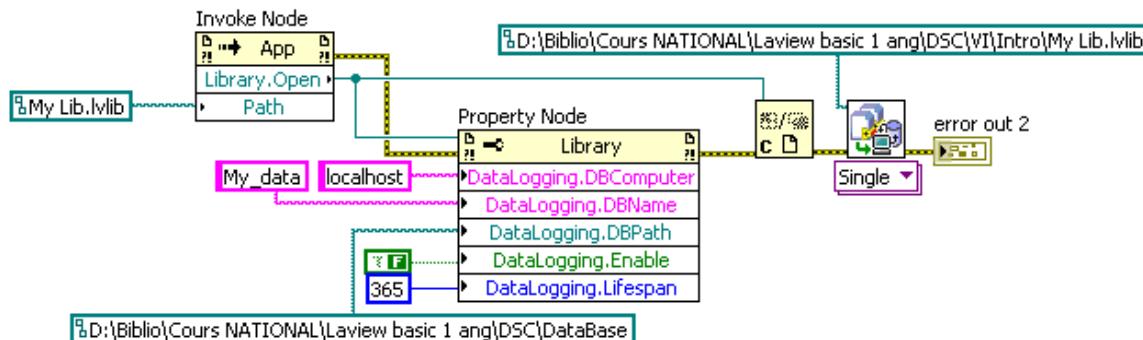
La base de données **citadel** est toujours en action quand DSC est installé, l'enregistrement d'une variable est une propriété de cette variable, pas de la base de données. La palette « **Historical** » permet de lire des données enregistrées, les alarmes et de faire des opérations de maintenance sur la base.

ENREGISTREMENT DES DONNEES

Elle est configurée soit lors de la création de la librairie et des variables, soit par code en utilisant les nœuds de propriété et de méthode des librairies et des variables partagées.

CHANGEMENT DU NOM DE BASE

Il s'agit d'une propriété de la librairie, il faut donc appeler la méthode « **Library.Open** » de l'objet « **Application** » pour obtenir une référence sur la librairie, puis utiliser les propriétés située dans les sous menuie « **Datalogging** » du nœud de propriétés.



Attention, les valeurs entrées dans le nœud de propriété ne sont prisent en compte qu'au redéploiement de la librairie, c'est pourquoi le nœud de propriété est suivi de la fonction « **Deploy Librarie** ».

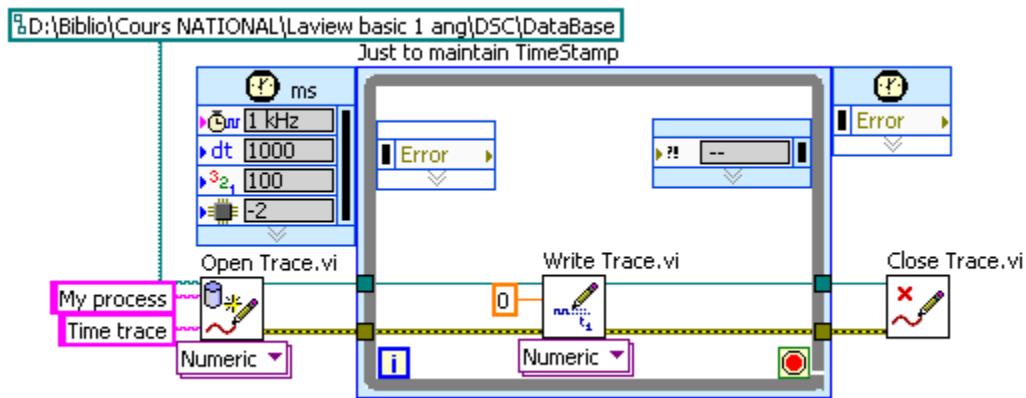
ACTIVATION DE L'ENREGISTREMENT D'UNE VARIABLE

C'est une propriété de la variable, elle est donc activée par un nœud de propriété :



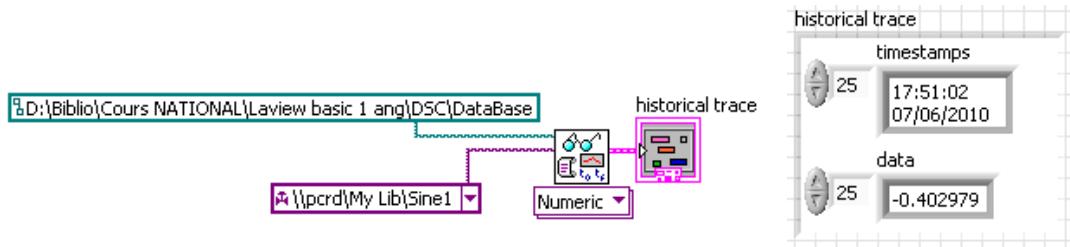
ENREGISTREMENT MANUEL DES DONNEES

Ce n'est pas le mode standard d'enregistrement de données, mais il se peut, qu'à un moment donné, il faille enregistrer une trace d'une variable non partagée, ou alors pour maintenir le marquage temporel d'une variable dont la valeur change peu (l'enregistrement dans **Citadel** ne se fait que sur variation de la valeur de la variable, pas à intervalle régulier)....

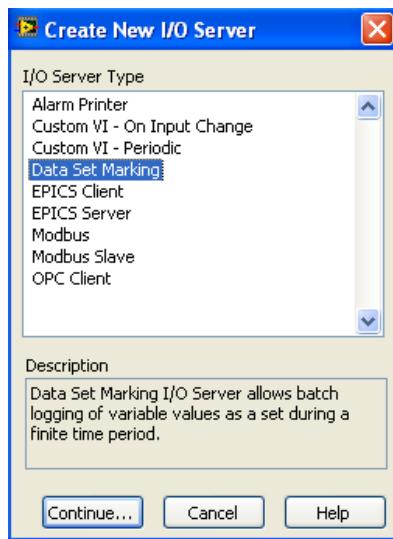


LECTURE DES DONNEES ENREGISTREES

La palette **historical** offre toutes les fonctions nécessaires pour l'extraction des données de la base **Citadel**, par exemple la lecture d'une trace :



SERVEURS D'E/S



SERVEURS STANDARDS

SERVEURS DE DONNEES

Nous avons vu que le module DSC assure la connectivité vers des serveurs d'E/S de type :

ModBus
EPICS
OPC

Ces serveurs font offices de passerelles entre des « variables terminales » (celles qui sont au bout du process) et des variables partagées. **ModBus** et **EPICS** sont des protocoles de communication complets pour du matériel industriel et de recherche. OPC n'est qu'une couche logicielle entre un logiciel de contrôle et l'environnement Windows.

SERVEUR D'IMPRESSION

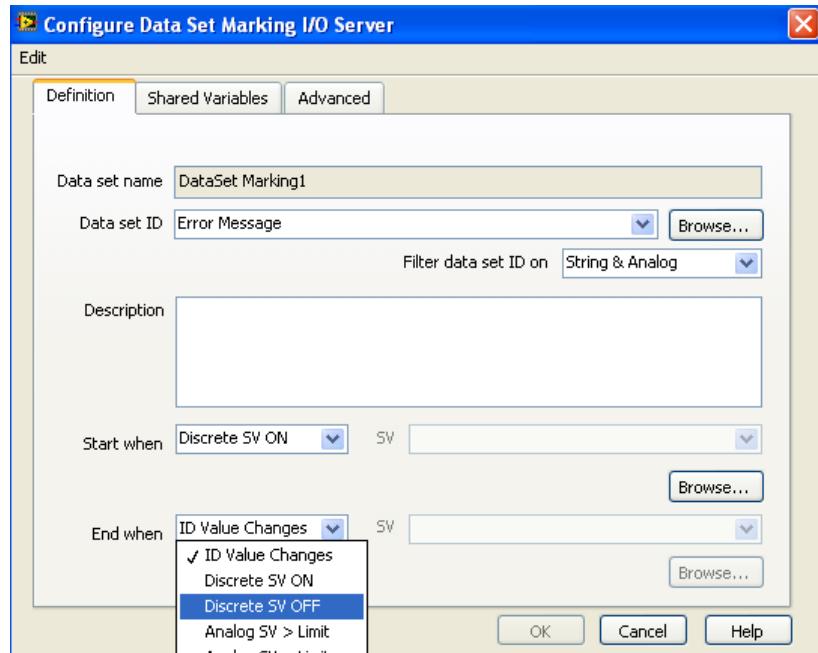
Alarm Printer

Ce serveur est dédié à l'envoie de l'impression de toutes les alarmes directement vers une imprimante LPT ou COM pour garder une trace papier

SERVEUR D'ENREGISTREMENT PAR LOTS

Data set marking I/O servers

Ce serveur est dédié à l'envoi de lots de variables partagées vers la base **Citadel**. Un lot est identifié par un « **data Set ID** », une cause de départ, une cause d'arrêt, et le nom de toutes les variables partagées à enregistrer.



Il est donc possible de déclencher par un événement (le résultat d'un calcul ou l'entrée d'un API) l'enregistrement de l'activité de certaines variables. Ceci est particulièrement utile pour les systèmes de test. Des fonctions permettent de retrouver l'ensemble des « **Data Set** » d'une base de données, de connaître le nombre de « **Runs** » dans ce « **Data Set** » et de les lire.

SERVEURS PERSONNELS

Si votre application utilise un voltmètre USB ou GPIB et que le fabricant du dit appareil ne vend pas de serveur OPC, que faire ?... Utiliser les « **Custom VI** », ils sont de deux sortes, « **On Input Change** » et « **Periodic** ». Ces deux options vont transformer un VI d'acquisition déjà écrit et testé en un processus qui sera exécuté par SVE dès le démarrage de la machine.

Le VI doit avoir les caractéristiques suivantes :

Etre réentrant.

Etre conçu pour s'exécuter le plus rapidement possible

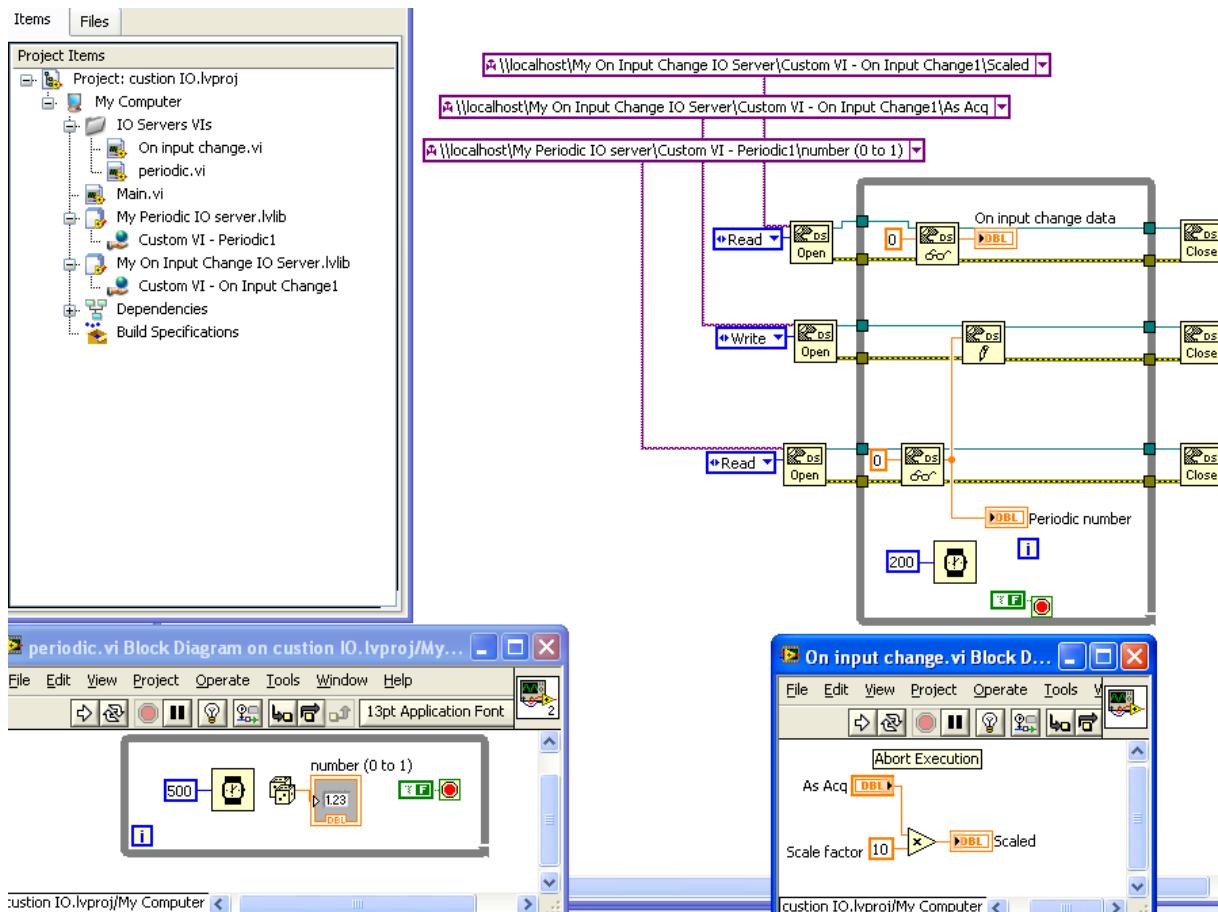
Ne pas contenir de boucles nécessitant l'intervention de l'utilisateur

Si possible n'utiliser que des doubles, des booléens et des chaînes

Les Vis , « **Periodic** » sont destinés à l'acquisition à cadence donnée, les , « **On Input Change** » plutôt à implémenter des calculs entre variables partagées (par exemple pour un mise à l'échelle d'une valeur brute sans autre code).

Une aide permet de créer ces serveurs, il faut choisir « **New-> VI Server -> Periodic ou On Input change** » et suivre les instructions.

L'exemple suivant montre un VI serveur « **Periodic** » et un autre sur « **On Input Change** », et le code permettant de récupérer leurs valeurs via des « **data sockets** ». Une fois inclus dans la bibliothèque, les deux Vis n'apparaissent plus à l'utilisateur, ce sont des boîtes noires qui fournissent et reçoivent des données. Notez les conventions de nommage des variables publiées.



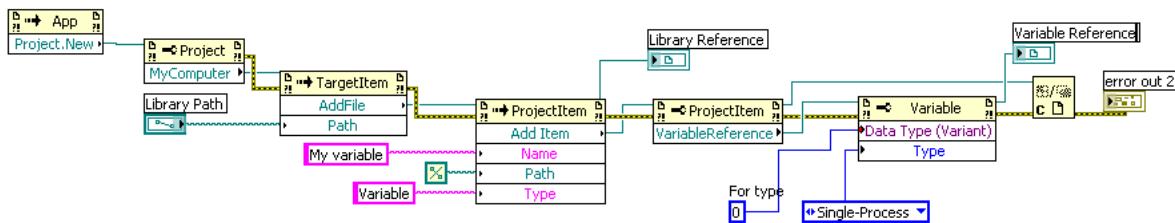
GESTION DES LIBRAIRIES ET DES VARIABLES

Le module de DSC comprend des fonctions avancées de la gestion des bibliothèques, des processus et des variables partagées pour créer par code tout ce qui est faisable via le projet. Deux façons de faire existent :

L'une va créer dans une librairie, les processus et les variables partagées, ces actions ne seront effectives qu'au redéploiement de la librairie. Elle s'apparente à un script de génération d'éléments d'un projet, élément pouvant être sauvegardés par l'enregistrement du projet.

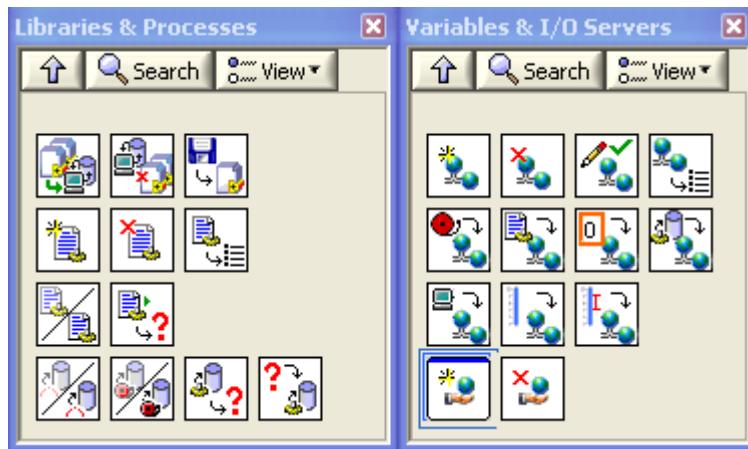
L'autre va les créer dynamiquement des processus et des variables partagées en dehors de toute librairie, l'action de ces Vis est immédiate mais ne peut être sauvée.

Dans le premier cas on utilise des nœuds de propriété et des méthodes pour créer les éléments d'un projet, c'est fastidieux et nécessite une bonne connaissance des propriétés & méthodes applicables aux éléments d'un projet, mais ça peut être utile si par exemple on veut créer un nombre important de variable numérotées. Le code peut ressembler à :



Nous n'aborderons pas plus cette possibilité qui n'est pas spécifique au module DSC.

Dans le deuxième cas il faut utiliser les fonctions de la palette « **EngineControl** ». Les fonction sont séparées en deux groupes, « **Library & Processes** » et « **Variables & I/O Servers** »



La première palette va permettre de déployer, d'annuler un déploiement, créer un processus, l'arrêter, connaître son état... La seconde de créer des variables partagées, les paramétrier, créer et tuer des serveurs d'E/S.

DEPLOIEMENT DES BIBLIOTHEQUES



Normalement lors du lancement de l'application, les bibliothèques sont automatiquement déployées, cependant, il est possible de déployer par programme des librairies (par ex. si elles ont été modifiées par programme) et d'annuler leur déploiement (pour quitter joliment une application ou en cas d'erreur)

CREATION DE PROCESSUS



Les fonctions présentées permettent de connaître les processus en cours, d'en charger ou d'en détruire, il s'agit de :

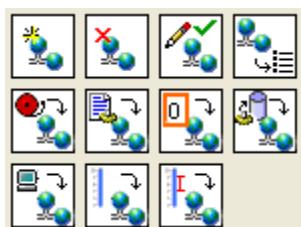
- Create Process**
- Delete Process**
- Get Process List**
- Enable Process**
- Get Process Run State**



Ces fonctions quant à elles permettent d'activer l'enregistrement des données et des alarmes, il s'agit des fonctions :

- Enable Data Logging On Process**
- Enable Alarm Logging On Process**
- Get Process Log State**
- Configure Process Log State**

CREATION DE VARIABLES PARTAGEES



Les fonctions de création/destruction de variables partagées sont dans la palette « **Variables & I/O Servers** » il s'agit principalement des fonctions

- Create Shared Variable**
 - Delete Shared Variable**
 - Commit Shared Variables**
 - Get Shared Variable List**
- Et tous les VI de configuration de ces variables (accessible aussi via les nœuds de propriétés)

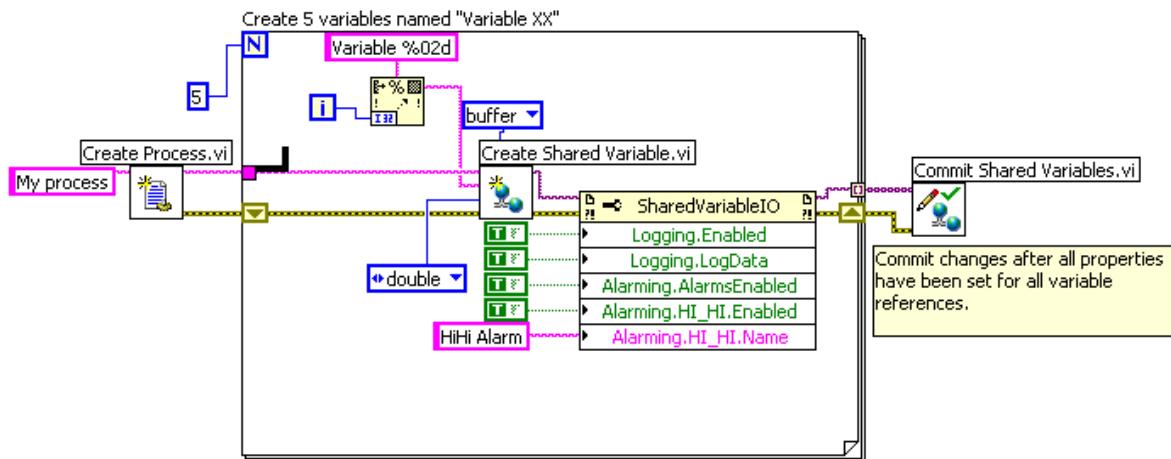
CREATION DE SERVEURS D'E/S



On y trouve simplement un VI express de chargement d'une serveur E/S et un tueur de serveur...

UN PETIT EXEMPLE

Le code suivant crée un processus nommé « **My Process** » et 5 variables et les inscrits dans le SVE.



16. LE MODULE STATECHART

INTRODUCTION

Beaucoup d'applications, notamment dans le domaine de l'automatisation de systèmes réels peuvent être décrits par une machine à états finie. Cette machine virtuelle possède un certains nombre d'états (d'où le nom de fini). La transition d'un état à l'autre est assujettit à une condition. Elle se représente sous la forme d'un graphe orienté étiqueté, dont les états sont les sommets et les transitions les arêtes étiquetées.

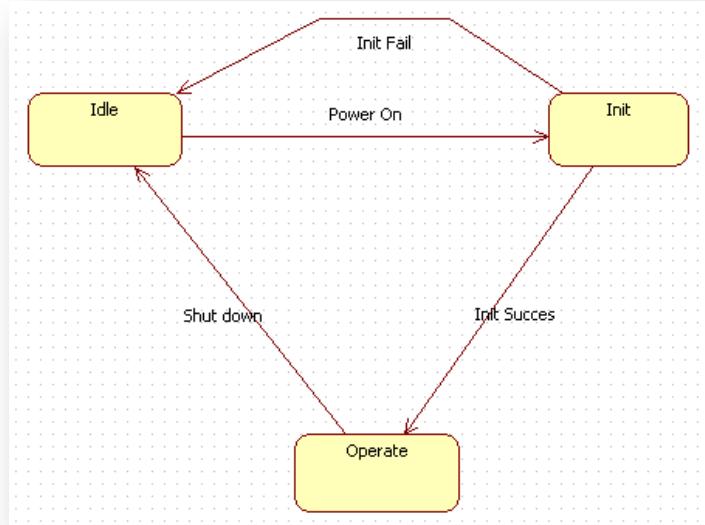


Figure 16-1 Diagramme état transition selon UML

La Figure 16-1 illustre un graphe état transition conçu avec le freeware « StartUML » toutes les illustrations sont faites avec ce logiciel. Un état représente une période de temps pendant laquelle il attend un événement et accomplit une activité. Si le graphe possède plusieurs branches concurrentes, plusieurs états peuvent être actifs en même temps

LES DIAGRAMMES ETAT TRANSITION UML

LabView Statchart implémente les diagrammes état transition selon une notation conforme au langage UML. UML, (**Unified Modeling Language**) est un langage de modélisation graphique utilisé dans le génie logiciel en particulier lors d'approches objets. Il utilise divers diagrammes pour décrire un système.

Parmi eux, le diagramme état transition sera notre centre d'intérêt. Nous détaillerons les éléments des états transitions d'UML en rapport avec ceux utilisés dans LabView StateChart

Pour plus de détails sur UML, reportez-vous au cours de Laurent Audibert sur developpez.com.

SYMBOLIQUE UML ET STATECHART

ETAT

Un état est représenté par un rectangle à coins arrondis, il a un nom ou bien il est anonyme. Le rectangle peut être séparé par un trait horizontal, la partie haute est réservée au nom, la partie basse à des transitions internes.



Les transitions internes seront expliquées plus loin, mais comme l'illustre la Figure 16-2, la syntaxe est :

Événement déclencheur / Action à effectuer

Figure 16-2 un état et ses transitions internes

ÉTAT INITIAL, ÉTAT FINAL

L'état initial est un pseudo état qui indique le début du graphe orienté, l'état final sa fin. La symbolique est représentée Figure 16-3

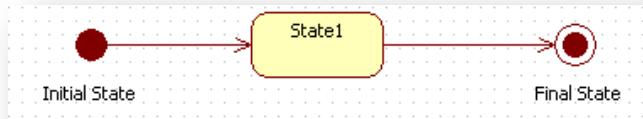


Figure 16-3 Etat Initial et Final

LabView ne permet pas de créer de transition internes pour ces états, UML le prévoit pour l'état final.

EVENEMENT

Un événement est quelque chose de remarquable qui se produit pendant l'exécution. Les diagrammes d'états-transitions spécifient les réactions du système à ces événements. Un événement se produit à un instant précis et n'a pas de durée. Les transitions sont déclenchées par les événements. Classiquement on reconnaît plusieurs types d'événements :

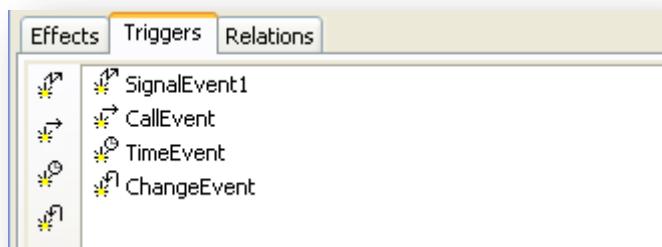


Figure 16-4 les différents types d'événements déclencheurs reconnus par UML

Mais LabView n'en fait pas la distinction, ils seront tous des « **Trigger** ».

TRANSITIONS

ENTRE ETATS

Une transition définit la réponse de la machine à l'occurrence d'un événement. Elle lie deux états et indique le sens dans lequel la machine peut passer d'un état à l'autre.

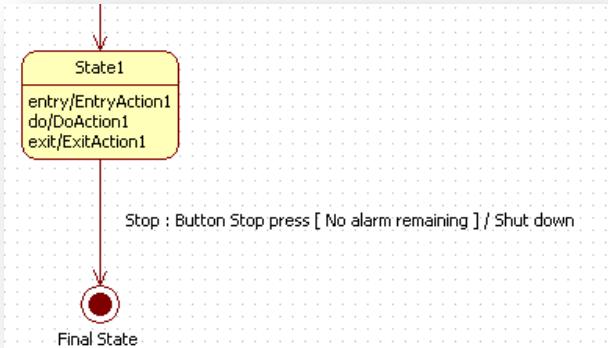


Figure 16-5 Transition entre deux états

Elle possède un nom, des événements déclenchant, une garde, des actions.

La syntaxe est la suivante :

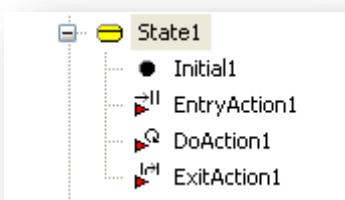
nom : événement [garde] / action

Tous les champs sont optionnels, la garde est une condition qui empêche, malgré l'apparition du déclenchement d'exécuter l'action. La transition est devenue active mais elle est bloquée par la garde.

Une transition peut ne pas avoir d'événement déclencheur, elle contient alors une condition de garde, c'est notamment le cas pour les transitions venant d'états initiaux ou d'états historiques, ces pseudo états ne pouvant rester actif.

INTERNES

Les transitions internes se produisent à l'intérieur d'un même état, elles ne sont donc pas orientées et n'ont pas de cible. LabView reconnaît les trois transitions internes d'UML :



« **Entry** » spécifie une activité effectuée lors de l'entrée dans l'état (généralement des initialisations)

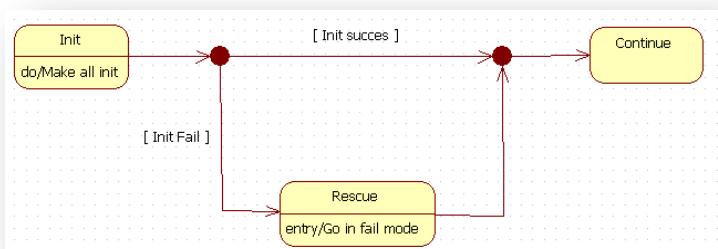
« **Do** » spécifie ce qui doit être fait après « **Entry** » et avant « **Exit** »

« **Exit** » Spécifie ce qui est fait lorsque le trigger vers l'état suivant arrive (généralement du nettoyage)

Figure 16-6 Les transitions internes

« **Entry & Exit** » n'ont pas d'événement de déclenchement, « **Do** » en possède, ainsi qu'une garde.

JONCTIONS



Un point de jonction est un pseudo état qui permet de couper des segments de transition dans le but de rendre plus lisible le diagramme. La jonction peut avoir plusieurs segments de transition entrant et sortant, mais n'a pas d'activité. Les

Figure 16-7 Utilisation des jonction avec des gardes différentes

transitions sortantes n'ont pas d'événement déclencheur, uniquement une action et une garde.

ETATS COMPOSE

Un état composé et un état constitué de régions dans lesquels il y a des diagrammes d'états, ils permettent de concevoir le diagramme d'état par une approche Top/Down, en faisant des raffinements successifs, du macroscopique au détail.

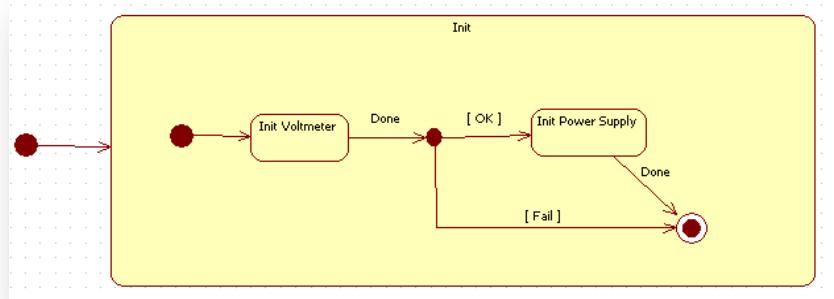


Figure 16-8 Description détaillée de l'état composite Init

Dans un diagramme de plus haut niveau, la représentation « **SubMachineState** » sera préférée car elle masque les détails de l'implantation. Le symbole avec deux cercles reliés par une droite indique qu'un diagramme état transition sous-jacent existe.

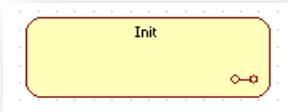


Figure 16-9 Sa représentation abrégée

Si un état composé contient plusieurs diagrammes (appelés régions), ceux-ci sont parcourus de façon concurrente. Il y a toujours un état d'une des régions actif à un moment donné. On qualifie un état composé d'orthogonal lorsqu'il contient plus d'une région.

Des outils graphiques supplémentaires permettent de synchroniser l'exécution d'états orthogonaux

TRANSITIONS DANS LES ETATS COMPOSÉS

Une transition vers la bordure d'un état composé à pour cible l'état initial de celui-ci. Une transition qui

traverse plusieurs frontières active les transitions internes « **Entry** » de chaque état composé imbriqué.

Une transition provenant de la frontière d'un état composé est équivalent à une multitude de transitions venant de tous les états de tous les états composés, quelle que soit leur profondeur d'imbrication. Dans ce cas, toutes les transitions internes « **Exit** » de tous les états et états composés actifs seront déclenchées.

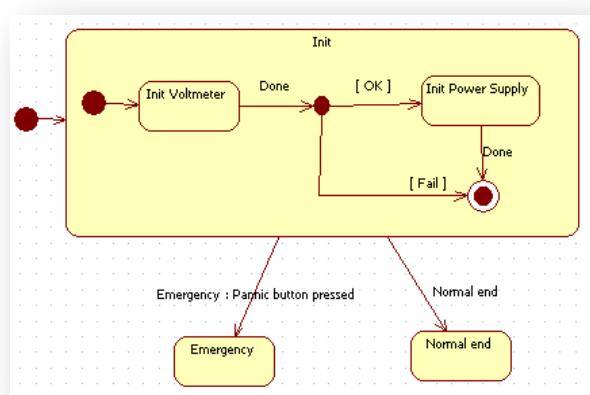


Figure 16-10 La transition « Normal End » qui n'a pas de trigger attendra que l'état final de l'état composé soit atteint avant de s'activer. La transition « Emergency » sera active dès la réception du signal « Panic button pressed », où que l'on soit dans le diagramme à ce moment-là.

En revanche, si cette transition ne porte pas d'événement déclencheur, elle ne sera franchissable que lorsque l'état final de l'état composé sera atteint.

HISTORIQUES

Le pseudo état historique permet lorsque l'on quitte un état composé pour y revenir de garder en mémoire l'état actif au moment où on avait quitté l'état composé. On distingue deux états historiques :

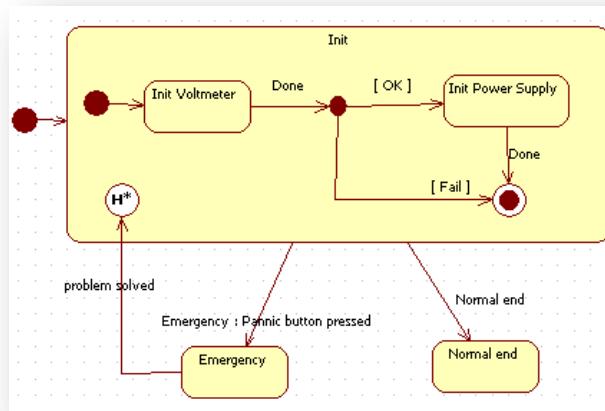


Figure 16-11 Si la transition Emergency est activée, lorsque l'on revient dans le diagramme ce n'est plus l'état initial qui est activé, mais le dernier état qui était actif lors de l'arrêt d'urgence.

pseudo état, cela signifie que lorsque cette transition devient active, c'est le dernier état actif qui le redévie.

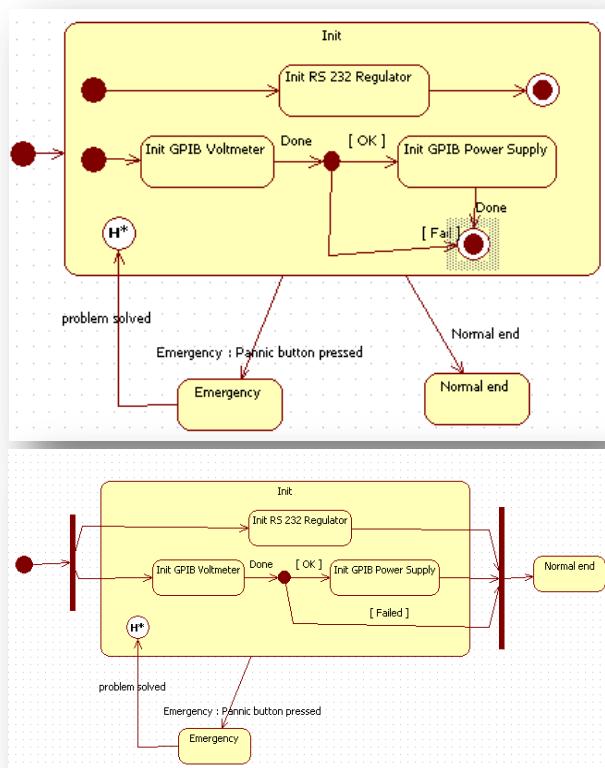
L'état historique peu profond (Shallow) qui mémorise le dernier sous-état actif d'un état composite (pas dans un état imbriqué). Il est représenté par un H dans un cercle.

L'état historique profond (Deep) qui mémorise le dernier sous-état actif quelque soit son niveau dans des diagrammes composés imbriqués les uns dans les autres. Il est représenté par un H* dans un cercle.

Lorsqu'une transition externe pointe vers ce

SYNCHRONISATION

Les diagrammes d'états-transitions permettent de décrire des processus concurrents grâce à l'utilisation d'états orthogonaux. Il s'agit un état composite comportant plus d'une région, chaque région représentant un flot d'exécution.



Lors de l'entrée dans le diagramme les deux états initiaux sont activés, et la sortie du diagramme vers l'état « **Normal End** » se fera lorsque les deux états finaux auront été atteints.

Des outils graphiques de synchronisation permettent de représenter ce mécanisme de façon plus compréhensible. Ils s'agir de barres épaisses constituant des transitions concurrentes, soit divergentes, (Fork) vers des processus simultanés, soit convergentes (Join) impliquant un point de synchronisation.

Figure 16-12 les barres de synchronisation permettent une représentation plus compréhensibles des processus concurrents et de leur flot d'exécution.

LES STATECHARTS LABVIEW

CREATION DE LA BIBLIOTHEQUE

Pour créer un StateChart, il faut, dans la fenêtre projet, sélectionner « **New->StateChart** », si l'item n'apparaît pas, c'est que le module n'est pas installé. Cette opération va créer une bibliothèque spéciale contenant les éléments suivants :

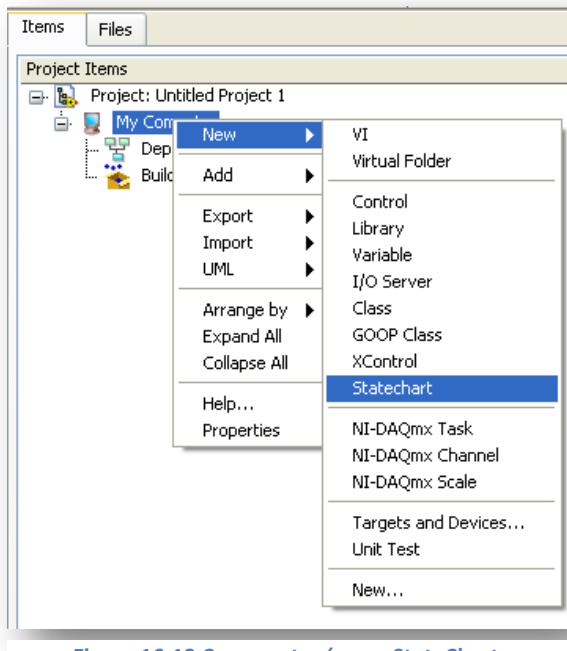
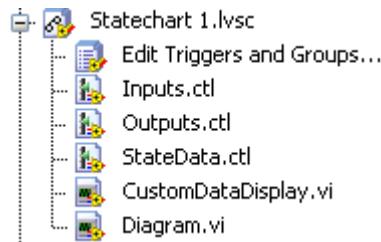


Figure 16-13 Comment créer un StateChart.



Le diagramme est le StateChart à proprement parler. « **CustomDataDisplay.vi** » permet de personnaliser l'affichage des données internes du StateChart. Les trois « .ctl » contiennent les données d'entrée du StateChart, de sortie et les données internes. Enfin, « **Edit Triggers and Groups** » va définir les conditions possibles sur les transitions.

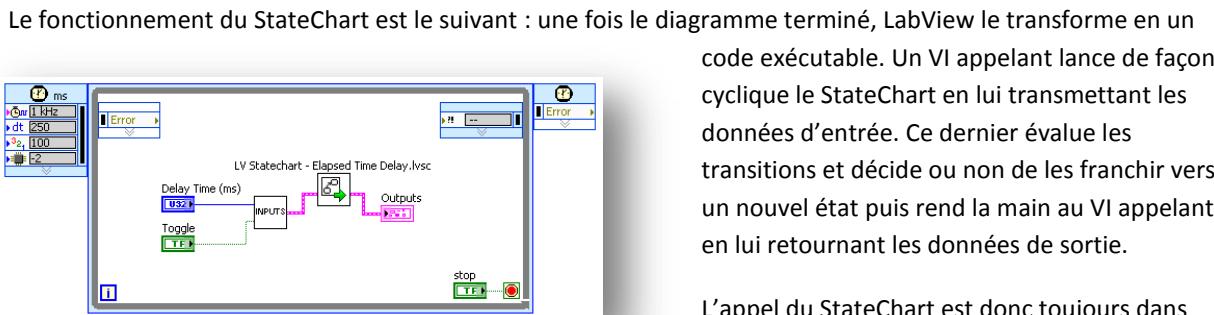


Figure 16-15 appel d'un StateChart synchrone

Le fonctionnement du StateChart est le suivant : une fois le diagramme terminé, LabView le transforme en un code exécutable. Un VI appelant lance de façon cyclique le StateChart en lui transmettant les données d'entrée. Ce dernier évalue les transitions et décide ou non de les franchir vers un nouvel état puis rend la main au VI appelant en lui retournant les données de sortie.

L'appel du StateChart est donc toujours dans une boucle. Les StateCharts peuvent être construits pour être synchrones ou asynchrones. Un StateChart synchrone fait une itération à chaque fois que le noeud « **RunStateChart** » est exécuté, ceci permet de connaître périodiquement l'état de la machine, l'appel est typiquement dans une boucle cadencée. L'exécution d'un StateChart asynchrone repose sur la présence d'une file d'attente de trigger externes, lorsque le noeud « **RunStateChart** » est exécuté, le StateChart va voir si il y a un trigger dans la file d'attente, s'il n'y en a pas, alors il s'endort jusqu'à l'arrivée d'un trigger qui est envoyé par une

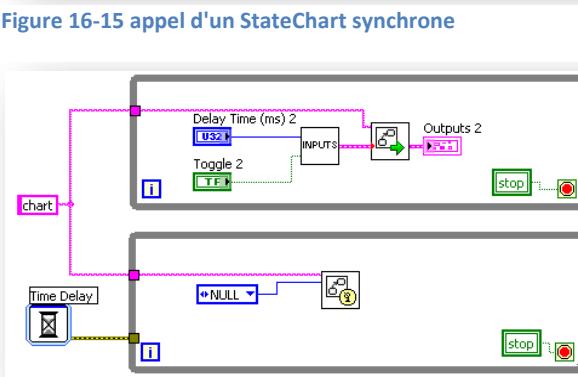


Figure 16-14 Appel d'un StateChart Asynchrone

« **SendExternalTrigger** ». L'appel est typiquement fait dans une boucle While, une deuxième boucle envoie les triggers quand ils sont disponibles.

CREATION D'UN STATECHART

Le StateChart est dessiné dans « **Diagram.vi** » à l'aide des outils de la palette « **StateChart Development** ». Ce diagramme spécial ne possède pas de face avant.

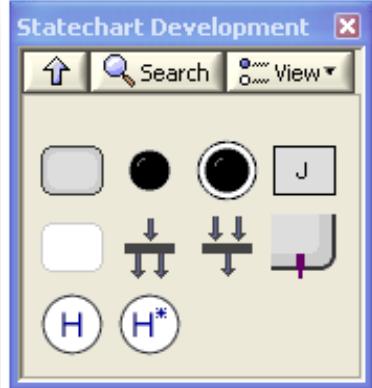


Figure 16-16 La palette de création de StateChart.

Sur cette palette nous retrouverons, les états, les speudo états, état initial, état final, historique profond et peu profond, jonction. Les outils de synchronisation avec les transitions divergente et convergentes, et enfin les régions.

Pour placer un état sur le diagramme, il faut en sélectionner l'outil et le poser sur le diagramme à la façon des boucles de répétition. Pour placer une transition, il faut s'approcher du bord intérieur d'un état de départ et, lorsque l'outil bobine apparaît tirer le fil vers le bord intérieur d'un état final.

Le code des actions des états, tout comme le code de garde ou les événements déclenchant des transitions sont accessibles par un double clic sur l'état ou la transition.

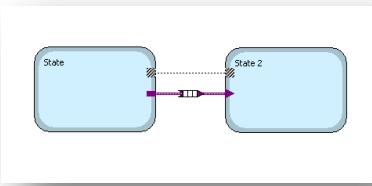


Figure 16-17 Câblage d'une transition

CODE ASSOCIE AUX ETATS

Lorsque l'on double clic sur un état, il passe en mode édition, et la fenêtre suivante apparaît :

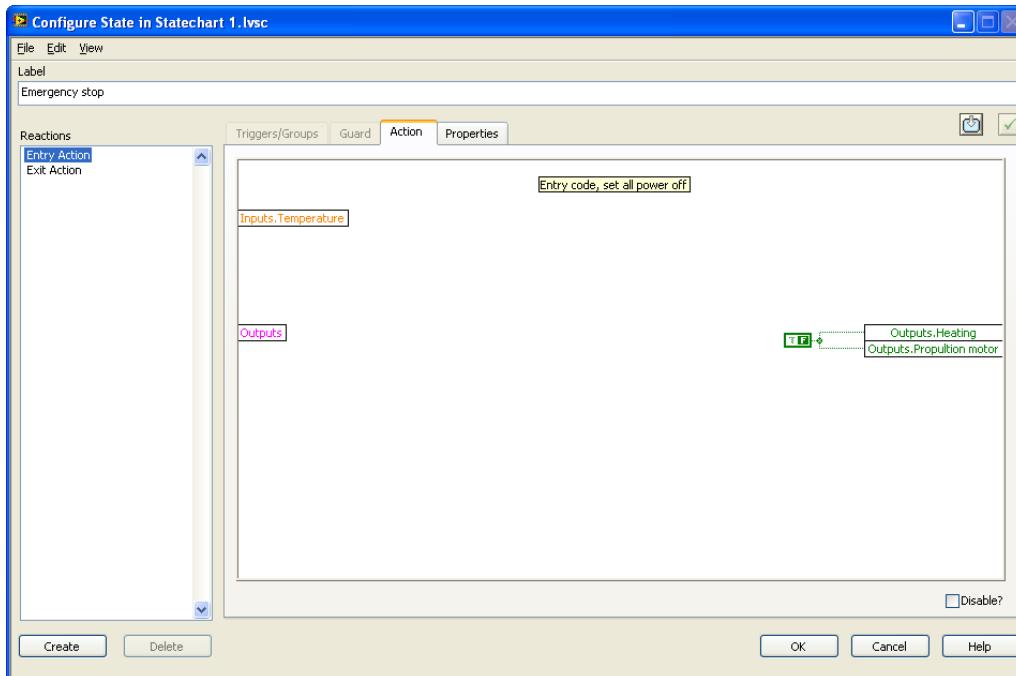


Figure 16-18 Fenêtre d'édition d'un état

A gauche se trouvent les transitions internes « **Entry & Exit** », à droite, dans l'onglet « **Action** » le code LabView correspondant à ces transitions. L'onglet « **properties** » permet de documenter le code. Par défaut il n'y a que ces deux transition internes qui n'ont n'y déclencheur n'y garde. Pour créer une transition interne supplémentaire, il faut cliquer sur le bouton « **Create** ». Une transition interne est alors créée, elle possède des déclencheurs et une garde. Les noeuds situés à droite et à gauche permettent de lire ou d'écrire certains éléments du StateChart.

Le concepteur peut décider d'envoyer au StateChart des données que ce dernier utilisera (un mot à écrire sur

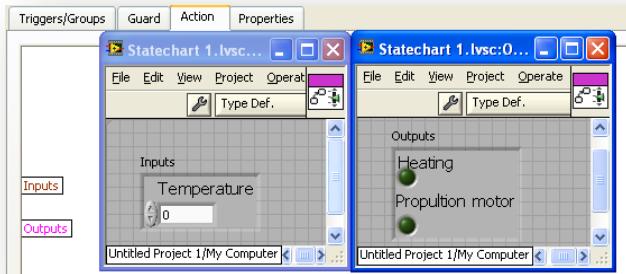


Figure 16-19 Les variables d'entrée et de sortie

une liaison série, la température d'un élément du processus...). Ces données seront contenues dans l'agrégat nommé **Inputs.ctl**. De même le diagramme d'état peut fournir un certain nombre de variables à l'extérieur, elles seront contenues dans l'agrégat **Outputs.ctl**. L'outil doigt permet de sélectionner la variable à lire ou à écrire. Ces données sont transmises et reçues par le VI appelant. Enfin un dernier agrégat nommé **StateData.ctl** contient des données que le concepteur veut transmettre d'état à état, mais pas au VI appelant.

LES TRANSITIONS

L'onglet trigger associe à une transition interne ou externe un déclencheur. Les triggers sont une énumération qui est créée dans la page « **Edit Triggers and Groups** ». Les triggers sont envoyés, soit par le VI appelant qui les reçoit du monde extérieur, soit pas du code placé dans les « **actions** ». Un groupe de trigger rassemble plusieurs déclencheurs, si l'un d'entre eux est actif, le groupe l'est. Par défaut un trigger particulier est systématiquement envoyé, c'est le trigger « **NULL** ». C'est le trigger actif par défaut pour toutes les transitions.

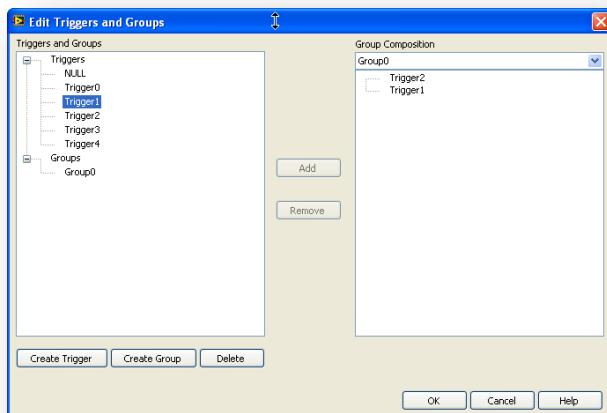


Figure 16-20 Page de configuration des triggers

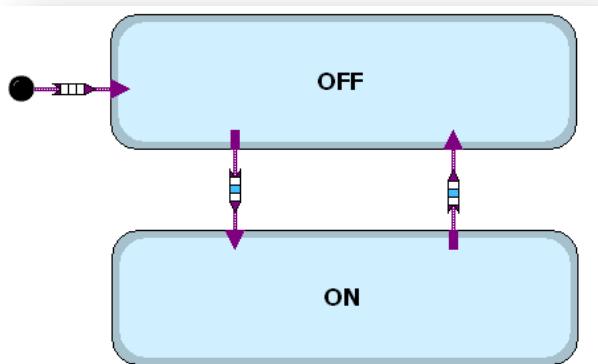


Figure 16-21 Les transitions configurées

Une fois ces triggers créés, ils sont utilisé dans la page de configuration d'une transition. La transition est symbolisée par une flèche contenant trois cases, blanches ou bleu selon qu'un élément de la transition est configuré ou non.

La première case (côté opposé à la pointe) indique si un trigger autre que « **NULL** » est configuré.

La seconde, si un code de garde est défini.

La troisième si une action à été codée.

On accède à la page de configuration des transitions par un double clic. Elle est comparable à celle des états, hormis le fait qu'il n'y a qu'une action possible (UML ne prévoit pas de transitions internes dans les transitions)

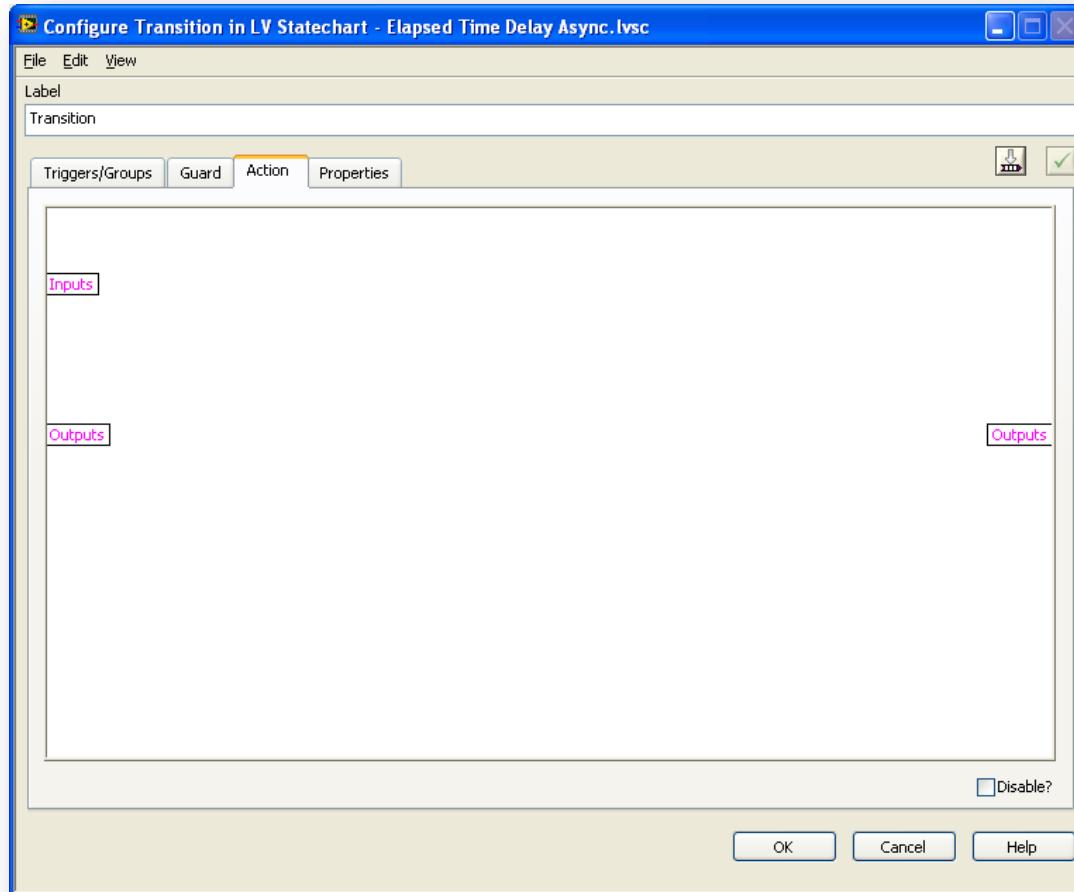


Figure 16-22 La page de configuration des transitions

Après cette présentation rapide, l'utilisation autour d'un exemple simple devrait clarifier ces concepts

AUTOUR D'UN EXEMPLE

Nous allons reprendre un exemple largement illustré dans le site NI, celui du ventilateur de plafond.

Cet équipement de haute technologie possède trois commandes, une commande générale murale qui met en marche l'appareil, une tirette pour la lumière, une tirette pour la vitesse de rotation.



Figure 16-23 L'objet !

La commande murale arrête, ou met en marche dans l'état où on l'a arrêté, l'appareil.

La tirette de lumière, à chaque action, éteint ou met en marche l'ampoule électrique.

Enfin la tirette vitesse fait passer, de façon circulaire la vitesse de l'appareil de Arrêté->Lent->Moyen->Rapide->Arrêté->...

En tant que bon élève nous ajouterons un trigger stop qui permet de sortir proprement de la machine d'état, mais normalement sur un système embarqué, ce sera l'arrêt de l'alimentation !

LE DIAGRAMME STATECHART UML

COMMANDE DE L'ECLAIRAGE.

Le diagramme UML ressemble à :

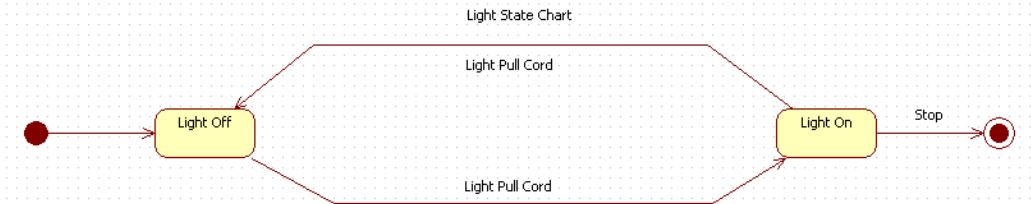


Figure 16-24 Diagramme d'état de l'éclairage

Il n'y a que deux états, la transition entre les deux est la même, l'action de la tirette de lumière.

COMMANDE DE VENTILLATION

Le diagramme UML ressemble à :

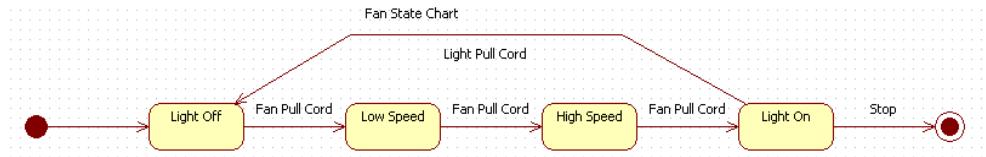


Figure 16-25 Diagramme d'état de l'éclairage

Nous y retrouvons les quatre vitesses de rotation activées par la tirette, le stop de l'interrupteur mural active la pseudo étape finale.

DEUX PROCESSUS CONCURRENTS

Ces deux commandes sont indépendantes l'une de l'autre et constituent donc deux processus concurrents. Pour représenter ces deux processus, il est aisément de les mettre dans un état composé :

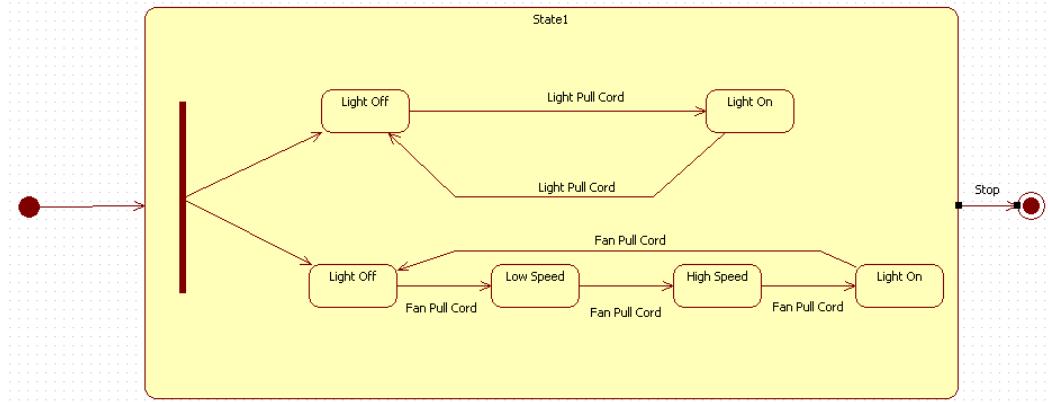
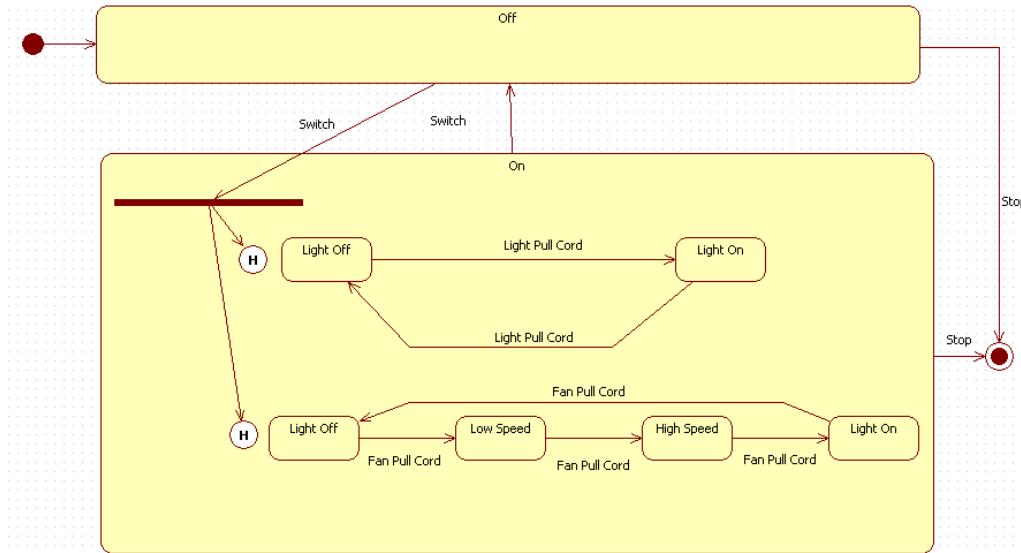


Figure 16-26 Le diagramme d'état composé des deux processus

MISE EN MARCHE ET ARRET

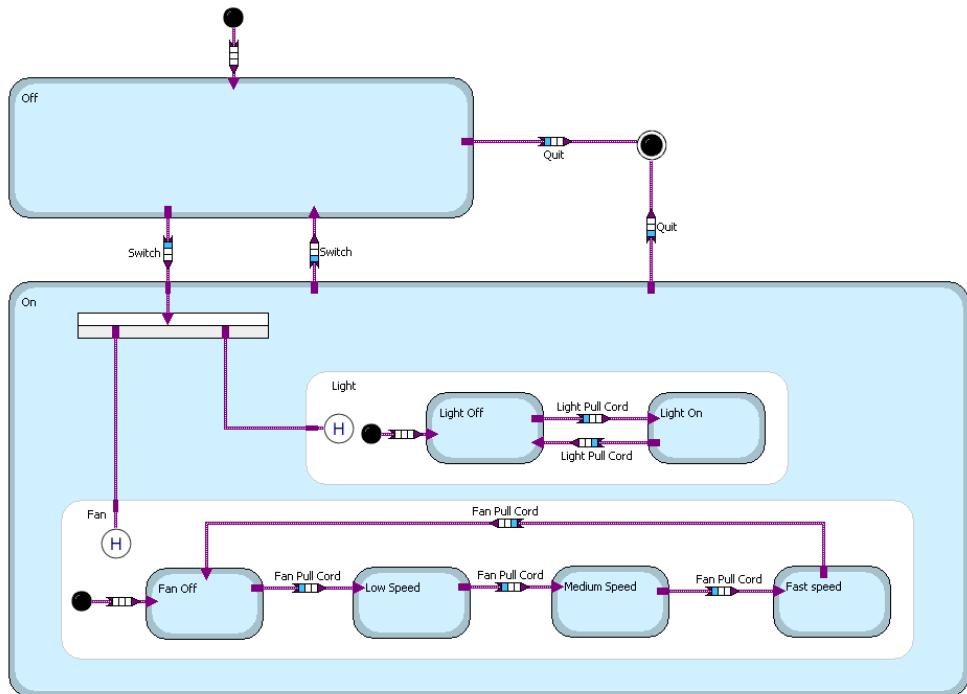
Ces deux processus peuvent être interrompus par un appui sur le bouton mural, une transition part donc de l'état composé vers l'état off. La mise en marche se fait dans la configuration précédente l'arrêt, un historique pour chaque région est nécessaire.



16-27 Le diagramme complet

LE DIAGRAMME STATECHART LABVIEW

CREATION

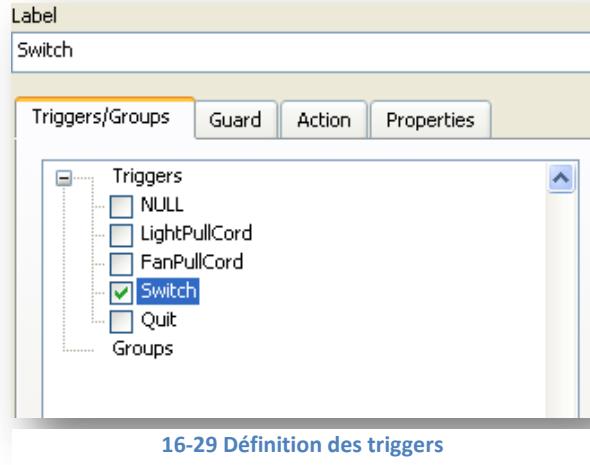


16-28 La version LabView du StateChart

Une fois le StateChart conçu, il est aisément de le transformer en un modèle LabView.

CREATION DES TRANSITIONS

Nous avons quatre triggers possibles : LightPullCord, FanPullCord, Switch, Quit. Il faut les créer dans « **Edit Triggers and Groups** ».

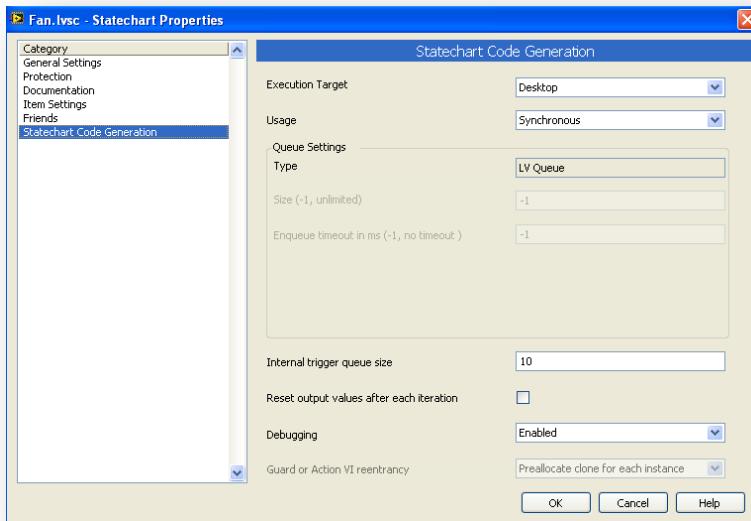


16-29 Définition des triggers

Puis les configurer les transitions pour qu'elles soient déclenchées par leur trigger respectifs. De nombreux exemples de chez NI omettent les trigger et les remplacent par une condition de garde, notamment dans les configurations synchrones, c'est une façon de voir les choses, mais...il me semble que celle-ci est plus compréhensible.

SYNCHRONE OU ASYNCHRONE

Cette option (dans la fenêtre de propriétés du StateChart) devrait être décidée dès la conception (car elle impacte fortement sur la façon de concevoir le code), mais elle n'est pas supportée par UML et ne « colle » donc pas très bien au modèle.



16-30 Options de génération du code

d'état est implantée dans un FPGA, elle sera évaluée à chaque coup d'horloge. Implémenter une machine synchrone dans un système Windows peut entraîner une consommation de ressources système importante, même si on reste dans un état stationnaire. L'option asynchrone va mettre la machine en « **Idle** » tant qu'un trigger externe n'est pas reçu ce qui sauve de la ressource. En revanche, les variables d'entrée et de sortie ne sont évaluées qu'à la réception d'un trigger, il ne faut pas compter sur elles pour faire évoluer l'état de la machine, c'est, de toute façons une très mauvaise façon de programmer. Notre système devant être implanté dans une carte FPGA, nous resterons sur l'option synchrone.

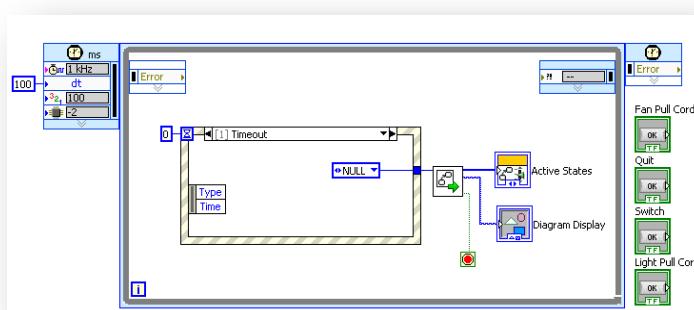
Globalement, si la machine d'état est implantée dans un FPGA ou un système temps réel, il est préférable de choisir un modèle synchrone et de fait un modèle asynchrone pour les systèmes Windows.

En effet, le modèle synchrone impose une boucle cadencée autour de l'appel du StateChart, cette boucle va déterminer la vitesse de réaction de la machine à une sollicitation extérieure, les systèmes RT sont prévus pour assurer ce cadencement, et si la machine

EXECUTION DU STATECHART

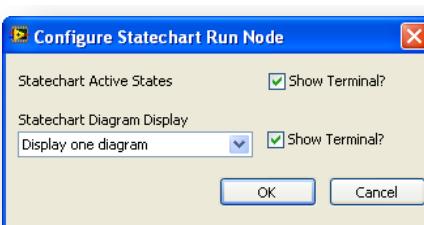
Le StateChart peut être exécuté tel quel, il n'est pas capable d'exécuter la moindre interaction avec l'extérieur,

mais il est aisément vérifier s'il répond bien aux triggers et si les étapes se déroulent correctement. Le VI appelant doit lancer la machine d'état périodiquement, il lui transmet les données d'entrée et récupère les données de sortie (aucune pour l'instant). Le VI « **Run StateChart** » est dans la palette StateChart, mais il est plus simple de faire un glisser/déposer de Fan.lvsc de la fenêtre de projet au diagramme.



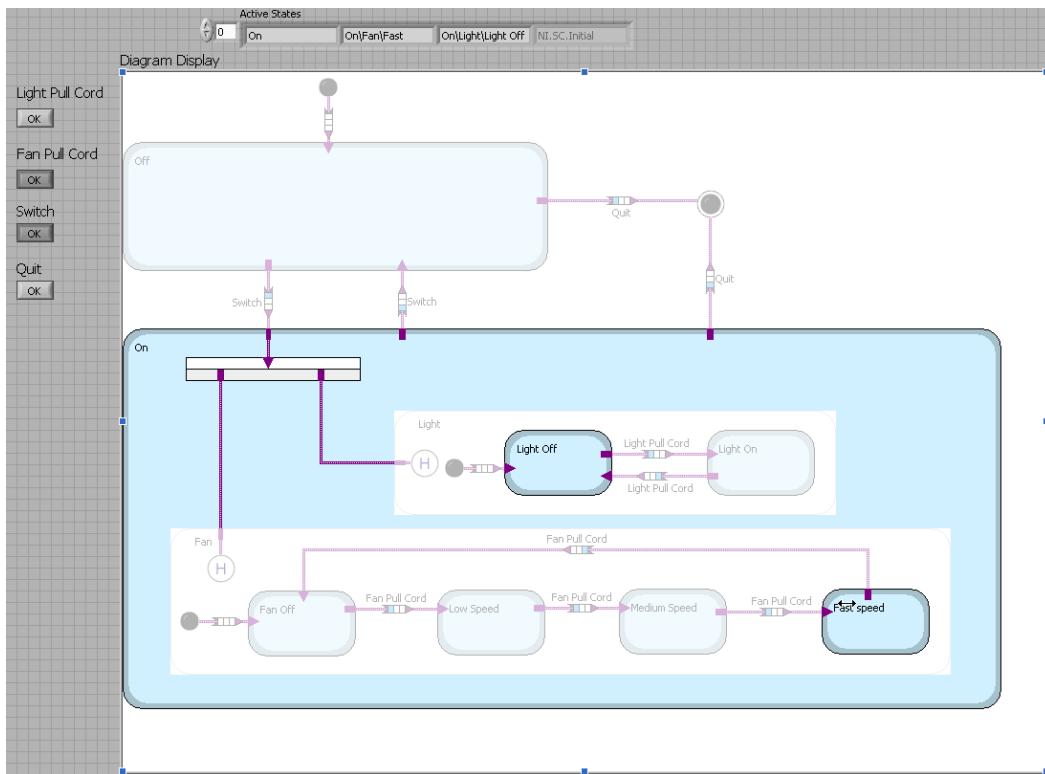
16-31 Diagramme du VI appelant

Le VI « **Run StateChart** » possède un menu de configuration qui permet de faire apparaître deux terminaux spéciaux. L'un permettra de connaître les états actifs dans un tableau, l'autre les illustrera dans une image. Ces options sont pour faire du débogage, car elles sont gourmandes. Notre VI « **Run**



16-32 Options de configuration

StateChart » recevra un trigger « **NULL** » si aucun bouton ne change de valeur (dans le « **timeout** » de la structure événement) et une trigger idoine selon le bouton utilisé (dans un « **Change Value** » de la structure événement). Si la flèche d'exécution est cassée, il est probable qu'il faille générer le code du StateChart (clic droit du le StateChart dans la fenêtre projet -> « **Generate Code** »)



16-33 Vue de la face avant

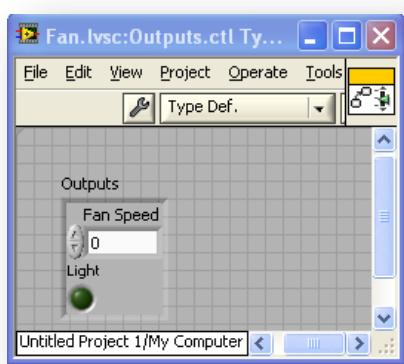
L'image du StateChart est créée en utilisant « Crée un indicateur » sur la sortie « **StateChart diagramme display** » du VI « **Run StateChart** ».

VARIABLES DE SORTIE

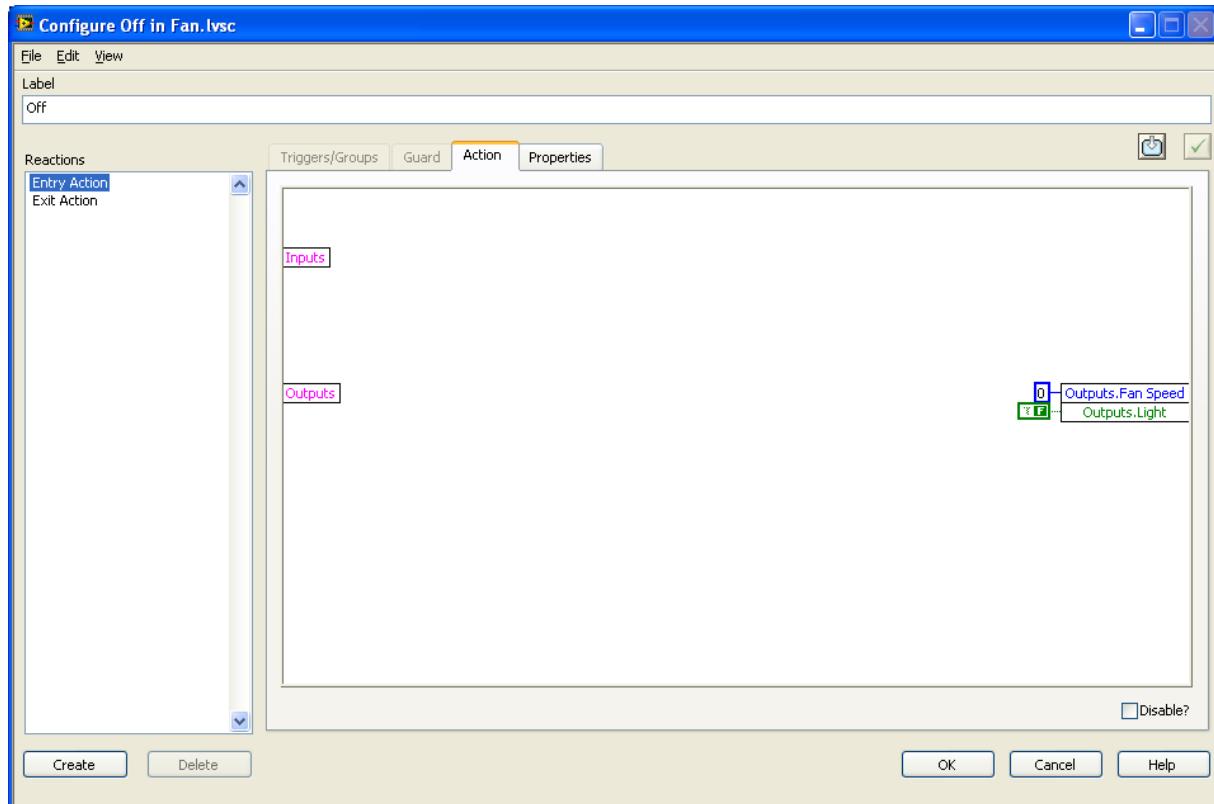
Notre StateChart n'a pas besoin de variables de sortie (le FPGA commanderait directement la partie opérationnelle), mais pour illustrer, nous allons créer deux variables, un booléen pour la lumière, un entier borné entre 0 et 3 pour la vitesse du ventilateur. Ces variables sont contenues dans le cluster « **Outputs.ctl** »

Les valeurs à attribuer à ces variables sont transmises par le code associé aux transitions « **Enter** » des états.

Par exemple pour l'état « **Off** » le code suivant initialise les variables à l'état de repos, ce code pourrait être identique dans l'action associée aux transitions « **Quit** »



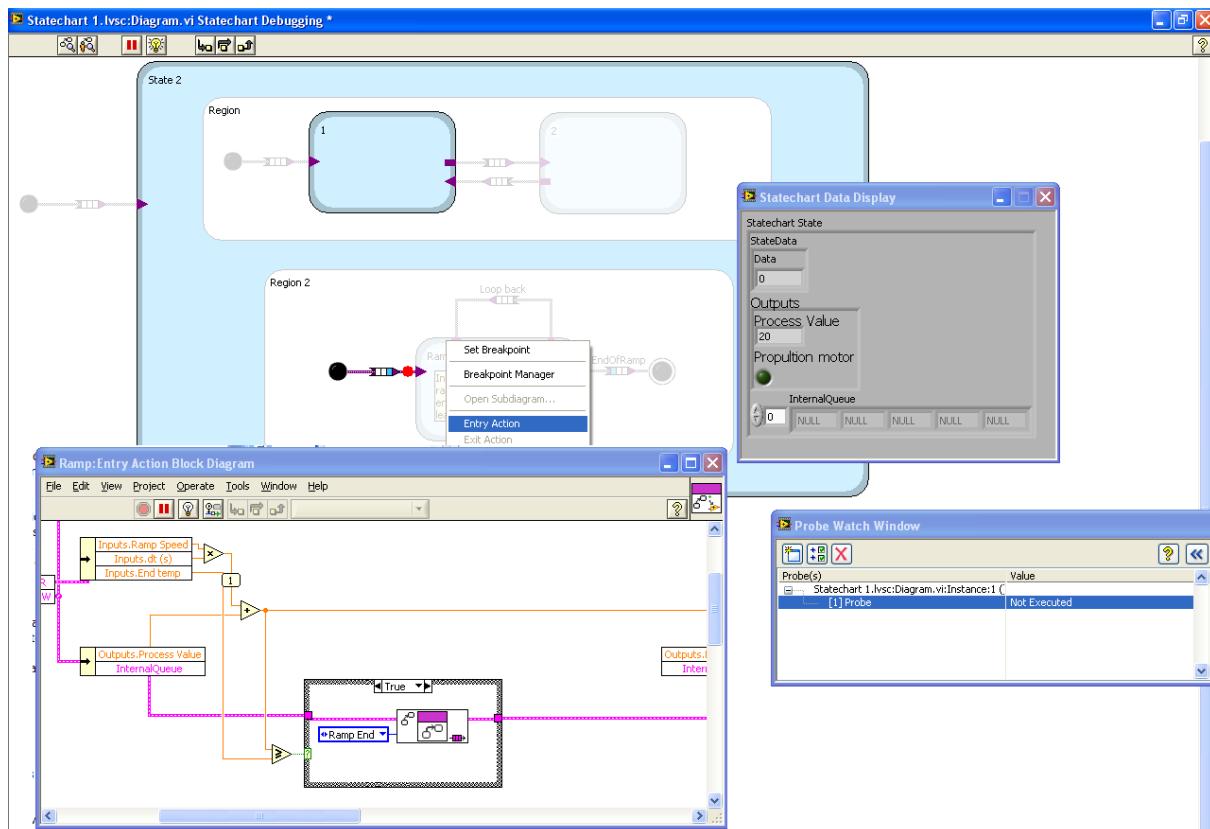
16-34 Le cluster de sortie



16-35 Code associé à l'état Off

DEBOGGAGE

Le menu contextuel de la fonction « **Run StateChart** » possède un item « **Debug StateChart** » elle ouvre une fenêtre représentant le StateChart. Il est alors possible de l'exécuter en pas à pas, de visualiser « **StateData** », « **Outputs** » et « **InternalQueue** », d'entrer, par le menu contextuel des états et des transitions dans le code d'action ou de garde, d'y poser des sondes...



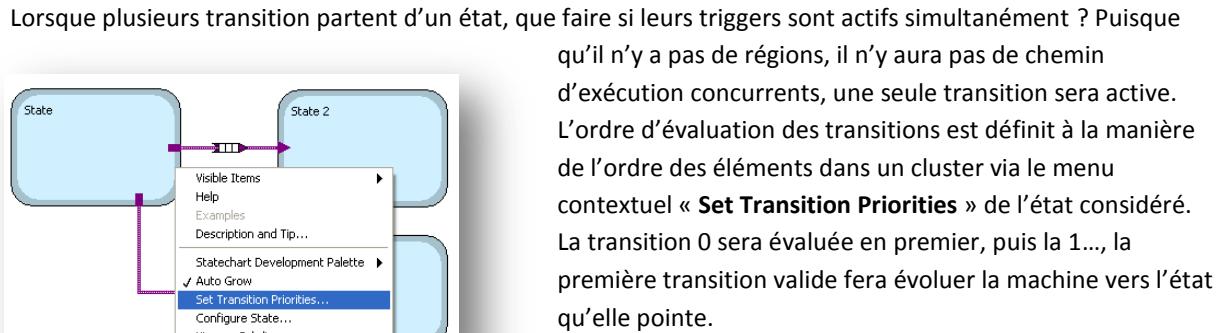
16-36 Exemple d'une fenêtre de débogage

Les icônes ouvrent la fenêtre « **StateChart Display Data** »

La fenêtre « **Ramp :Entry Action** » est appelée par un clic droit sur l'état « **Ramp ->Entry Action** ». Une fois ouvert, le débogage se passe classiquement, points d'arrêt, sondes...

UN PEU PLUS LOIN

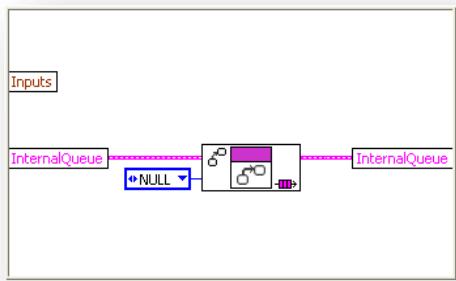
TRANSITIONS SIMULTANNEES



16-37 Priorité des transitions

FILE D'ATTENTE DE TRIGGERS

Si un état doit provoquer un trigger, par exemple pour engendrer une transition dans une région concurrente, il doit ajouter cette transition à la file d'attente des transitions internes. Cette dernière sera évaluée à la prochaine exécution du VI « **Run StateChart** ».



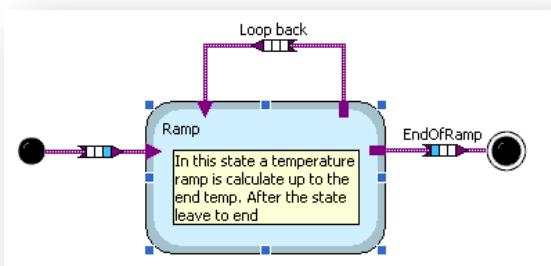
16-38 Envoie d'un trigger interne

Le VI « **Send Internal Trigger** » rempli cette fonction, il ne peut être utilisé que dans le code « **Action** » d'un état ou d'une transition. L'entrée et la sortie de la fonction doivent être reliées à la file d'attente interne en sélectionnant « **Outputs->InternalQueue->All Elements** ». Dans un diagramme d'état asynchrone, il y a deux files d'attente, une externe, une interne. La file d'attente externe ne sera évalué que lorsque la file d'attente interne sera vide

donnant ainsi priorité aux événements internes de la machine d'état.

REPETITION D'ACTION D'UN ETAT

Lorsque l'on veut répéter une action dans un état, par exemple incrémenter un compteur pour commander la température d'un four jusqu'à une valeur donnée, il est dangereux de mettre une boucle conditionnelle dans l'action d'un état. En effet, les transitions ne sont évaluées que lorsque les actions relatives aux états actifs sont terminées, si une boucle infinie tourne dans une des actions, elle bloque l'exécution de tous les processus concurrents.



La méthode la plus claire pour obtenir l'effet souhaité est de créer une transition qui se reboucle sur l'état de départ, lui associer un trigger « **NULL** », et de donner aux transitions sortantes une priorité supérieure à la transition de rebouclage.

16-39 Répétition de l'action d'un état

17. LABVIEW FPGA

INTRODUCTION

LabView FPGA est le module de programmation des "réseaux logiques programmable". Ces circuits sont implantés sur divers types de matériels de chez NI, ils proviennent du fondeur Xilinx. Ils font partie de la gamme Startan économique ou Virtex haute performance.

QU'EST-CE UN FPGA ?

Les FPGA sont des circuits logiques reprogrammables. À l'aide de blocs logiques prédéfinis et de ressources programmables de routage, vous interconnectez ces portes pour mettre en œuvre des fonctionnalités matériel personnalisées comme vous l'auriez fait il y a peu de temps avec votre stock de CD4XXX et votre fer à souder. Des outils permettent de créer un fichier de configuration ou « bitstream » qui contient des informations d'interconnections. Ces "bitstreams" sont stockés dans des EPROM séparées et sont transférés dans la RAM du composant Xilinx lors de la mise sous tension. Les FPGAs fournissent robustesse, vitesse et fiabilité, ils sont véritablement parallèles et les traitements ne dépendent pas d'un OS.

LABVIEW ET LES FPGA

Labview n'est pas une suite logicielle de développement de matériel basé sur des FPGAs, vous ne pourrez pas développer (pour l'instant) en LabView une application pour une cible autre qu'un matériel NI. En revanche l'environnement offre une interface directe vers ces matériels et la programmation se fait sans autre outil que labview (pas d'autre langage, ni environnement de développement).

LES CIBLES SUPPORTÉES

Les cibles supportées font partie de la gamme RIO (E/S reconfigurables), il s'agit des matériels DAQ NI PCI et PXI de la Série R (acquisition), les matériels NI CVS-145x (Vision), CompactRIO(système modulaire durcis)et SingleBoardRIO (PCB nu). Les exemples sont pour cette dernière plateforme, elle possède un processeur powerPC embarquant un OS temps réel VXWorks et un Spartan3 avec 2 millions de portes. Les FPGA embarqués dans les cartes NI sont toujours de chez Xilinx dans la famille Virtex 2, 5 et 6 ainsi que des Spartan 3 et 6.

LABVIEW RT ET FPGA

Les modules RT et FPGA sont indépendant, mais souvent intimement liés, car la communication entre l'OS et le monde extérieur, dans les matériels qui en sont équipés passe par le FPGA. Dans ce chapitre nous ne parlerons en ce qui concerne le temps réel que de l'interfaçage avec un VI implanté dans le FPGA

CARTE SUPPORT DES EXEMPLES

Les exemples sont donnés pour une carte fille s'enfichant dans le connecteur P2 d'une carte SbRIO 9602 (E/S numériques uniquement et FPGA 2 millions de portes). Cette carte comporte quatre afficheurs sept segments multiplexés, deux LED, un bouton poussoir, un CNA et un CAN reliés, ainsi que trois BNC. Les schémas et câblages sont donnés en annexe.

PRÉPARATION DE L'ENVIRONNEMENT

Labview doit être installé ainsi que les modules FPGA et RT. Labview fournit un fichier VHDL au compilateur

Xilinx qui doit donc être installé.

Un « wizard » permet de créer une ébauche de projet FPGA.

L'utilitaire permet la détection automatique de la cible et de ses accessoires (si la cible n'est pas présente, il est possible de configurer manuellement l'ensemble). Une fois fini, le projet contient la description complète du matériel de la cible (E/S, horloges...)

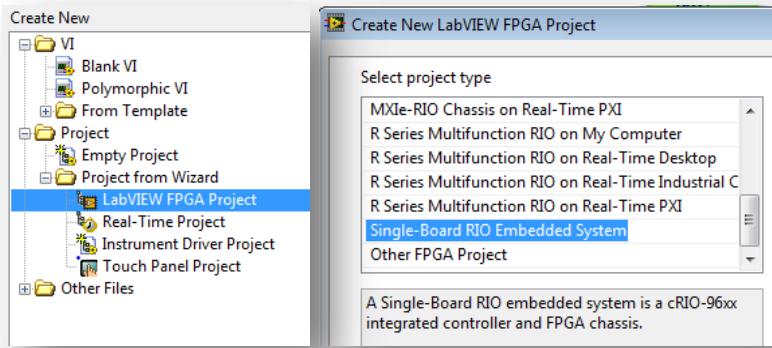


Figure 17-1 Crédation d'un projet FPGA

UTILISATION DES VOIES NUMERIQUES

La carte est équipée de quatre connecteurs permettant d'accéder aux 10 ports de 11 lignes.

Chaque port est composé de 10 lignes nommées Portx/DIOy et d'une ligne Portx/DIOCTL. Les premières sont des lignes rapides routées ensemble et appariées en longueur, la seconde est destinée à des E/S lentes (possibilité de d'interférences si utilisées à haute fréquence). Sous le nom Portx/Dio9 :0 sont regroupées les dix lignes d'un port. Chaque ligne peut être renommée (F2), et les lignes d'un même type regroupées dans un répertoire virtuel

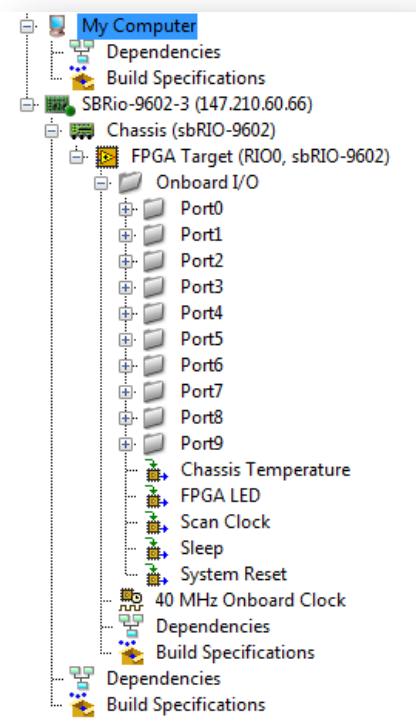


Figure 17-2 Regroupement des lignes dans un répertoire virtuel

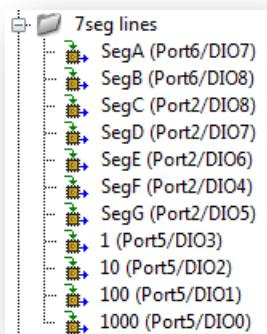


Figure 17-3 l'ensemble des E/S de la cible

ORGANISATION DU PROJET

Une application LabView FPGA/RT contient en général trois parties :

- a) Les VIs implémentés sur le FPGA.
- b) Les VIs implémentés sur le processeur de l'OS RT.
- c) Les VIs implémentés sur le PC de supervision.

Ceci n'est pas une règle absolue, mais un cas très commun. Ces différents programmes doivent communiquer entre eux via différents canaux (ils sont sur des chips physiquement séparés, au moins pour le FPGA) la communication s'établie dans l'ordre : **FPGA ↔ RT ↔ Supervision**. La topologie de l'ensemble peut prendre différentes formes (par exemple une carte PCI série R dans un PC équipé d'un processeur multi-cœurs exécutant un OS RT et XP), mais l'organisation générale reste la même.

COMMUNICATION ENTRE LES DIFFERENTES PARTIES.

COMMUNICATION ET FPGA ET RT

La communication entre les variables du code FPGA et des variables de la cible RT peuvent se faire :

- a) par des noeuds de propriété « Read/Write » des contrôles de la face avant du VI principal FPGA, c'est la méthode la plus simple, aucun code n'est nécessaire.
- b) Par des FIFOs, pour le transfert de volume données plus important avec la possibilité d'implémenter des mécanismes évitant la perte de données.

COMMUNICATION ENTRE CIBLE RT ET SUPERVISION

Cette communication utilise une liaison réseau physique ou virtuelle (dans le cas où deux OS tournent sur le même processeur avec NI Real-Time Hypervisor)

- a) Par des variables réseau, liaison directe sans code, la mise à jour de données est faite par le protocole NI PSP (attention à bien configurer le pare-feu pour autoriser les ports UDP et TCP 2343 ainsi que les ports TCP 59110 et supérieurs à raison d'un par application).
- b) Par des liens TCP/UDP ou des sockets pour améliorer la performance, notamment en encapsulant plusieurs variables dans le même paquet TCP ou UDP.

PREMIERS PROGRAMMES

UN PROGRAMME 100% FPGA

Pour débuter nous allons créer un programme entièrement implémenté dans le FPGA et ne communiquant avec l'extérieur que par les pattes d'entrée sortie du connecteur P2.

CREATION D'UN VI FPGA

Lors de la création d'un VI dans un projet, ce dernier est spécifique à la cible pointée. Pour créer un VI destiné à

être exécuté sur le FPGA il faut donc pointer l'icône de la cible FPGA avant de choisir « New → VI » dans le menu contextuel.

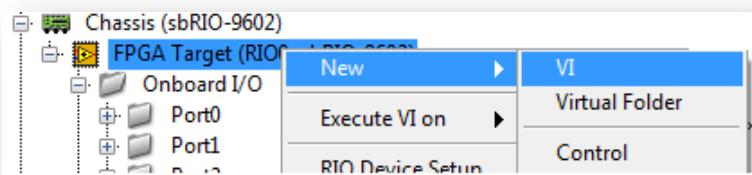


Figure 17-4 Création d'un nouveau Vi FPGA

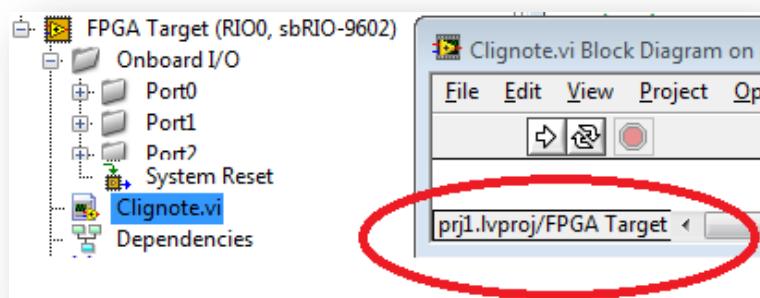
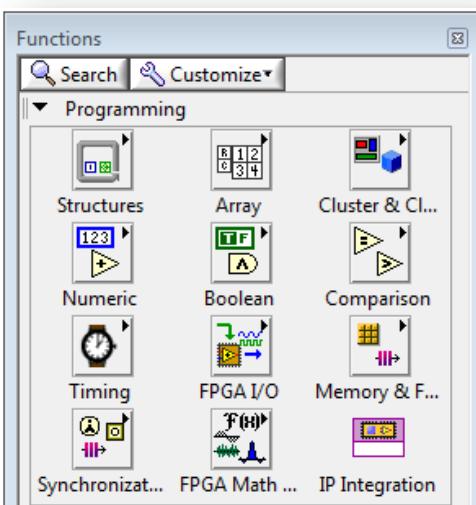


Figure 17-5 Affichage de la cible d'exécution.

Le nouveau VI doit se retrouver dans l'arborescence du « FPGA Target » et doit porter le nom de la cible dans le coin inférieur gauche de la face avant et du diagramme. Lors d'un développement multiplateforme, toutes les fenêtres indiquent sur quelle cible elles s'exécutent.

PALETTE D'OUTILS

La palette d'outil pour les FPGA est plus réduite que la palette normale.



Les principales différences sont dans les types des données numériques, il n'y pas de nombres en virgule flottante.

Les tableaux sont de taille fixe.

Ces limitations étant induites par la technologie FPGA.

Les opérations sur chaîne, les dialogues et autre E/S fichiers sont sans objets sur un FPGA.

A noter le support de la programmation objet et des machines d'état (StateChar).

Figure 17-6 Palette de fonctions FPGA

PROGRAMME

Un programme FPGA est en général une collection de boucles « While » tournant en parallèle pour gérer des flux d'entrées/sorties de périphériques en effectuant au passage des opérations logiques ou de calculs.

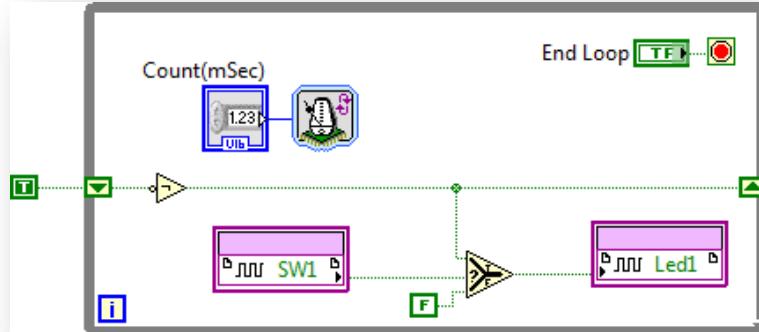


Figure 17-7 Un programme FPGA simple

Le programme suivant fait clignoter la LED de la carte d'essai, lorsque l'on appuie sur le bouton poussoir. Le dépôt du nœud de entrée/sortie sur le diagramme se fait par un glisser/déposer depuis la fenêtre de projet, on trouve dans le menu contextuel « Change To Write/Read » pour donner la direction de l'échange. La fonction « Wait » est configurable (durée en coups d'horloge, µs ou ms et longueur du mot d'entrée, 8, 16 ou 32 bits)

COMPILEATION

Un programme FPGA n'est qu'une table de routage entre des blocs logiques. Cependant le processus d'élaboration de ce routage peut être très long et dépend du nombre de cellules à interconnecter et des optimisations éventuelles. Cela peut représenter plusieurs heures sur des cibles type Virtex6 et plusieurs dizaines de minutes sur un Spartan. Le processus de compilation est lancé par la flèche « Run », Labview élabore dans un premier temps un fichier VHDL (le langage de programmation classique des FPGA). Ce fichier

est transmis au compilateur Xilinx qui synthétise le circuit, fait le mappage, puis le place et le route. La moindre modification du code entraîne une nouvelle compilation. Une fois le fichier « bitstream » créé, il est copié dans la mémoire flash et transféré sur le FPGA. Une fenêtre fournit des informations d'avancement et les statistiques d'utilisation.

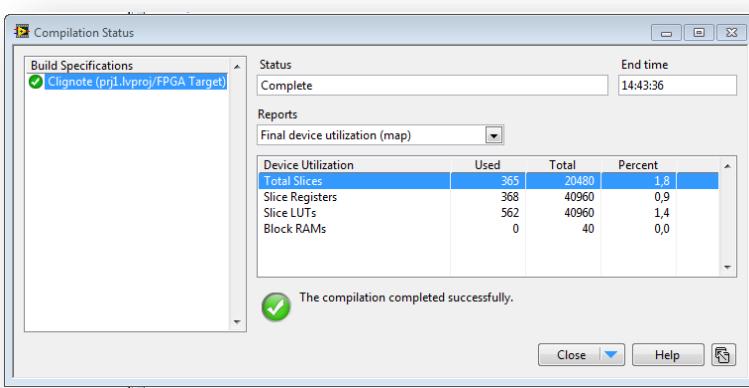


Figure 17-8 Fenêtre d'état de la compilation

La LED de la carte doit clignoter à 2 Hz. Attention **les outils de débogage ne peuvent être utilisés** sur la cible FPGA, il est possible d'ajouter des indicateurs pour vérifier la valeur d'une data, ou encore de simuler l'exécution sur le PC hôte (en simulant les E/S et sans respect des timings).

COMMUNICATIONS ENTRE VI FPGA ET VI RT

Ce programme est parfaitement fonctionnel et peut dans de nombreux cas être suffisant (implantation de relation combinatoire fixes entre entrées et sorties). Si le système doit faire varier la vitesse de clignotement de la LED, il faut transformer la constante en un contrôle puis relancer la compilation.

La face avant du VI FPGA permet de transmettre de façon transparente des données en entrée ou en sortie du code FPGA (en fait une liaison est établie entre le PC, la cible RT et le FPGA), cette communication a un coût en terme de portes logiques utilisée pour la partie FPGA et en ressource processeur pour la partie RT.

Si la fréquence de clignotement est le résultat, par exemple, d'un calcul effectué sur la cible RT, il faut transmettre cette valeur au contrôle du VI FPGA.

Dans un premier temps il faut créer un VI dans le module RT :

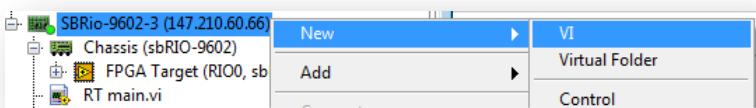


Figure 17-9 Crédit d'un VI sur la cible RT

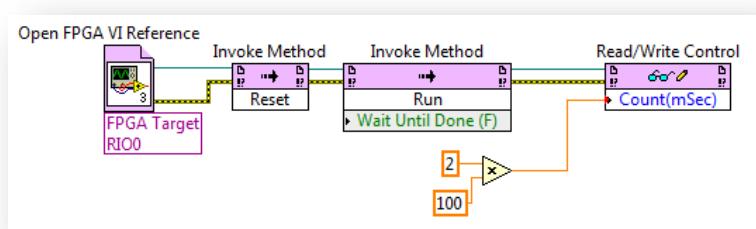


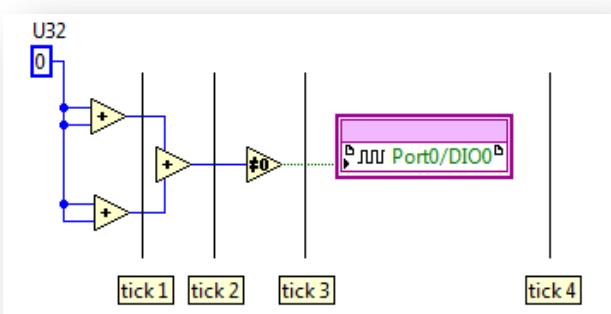
Figure 17-10 communication entre cible RT et FPGA

Puis utiliser les fonctions de la palette « FPGA Interface » pour créer une référence sur le VI implanté dans le FPGA, en assurer un démarrage correct et transmettre l'information. Le processus est tout à fait comparable à celui utilisé pour exécuter un VI à distance avec le VI-Serveur.

Ce code transmet la valeur calculée 200 au contrôle Count du VI FPGA. La référence doit être précisée (clic droit, « Configure Open FPGA VI Reference ») ainsi que la/les variables à lire ou écrire.

EXECUTION EN PARALLÈLE

Tout le code non séquentiel est exécuté strictement en parallèle (attention cela ne signifie pas qu'il est exécuté en une période d'horloge). Lorsque le code est séquentiel, il nécessite généralement une période d'horloge par opération (excepté dans le cas de fonctions complexes : division, FFT, filtrage, E/S analogiques...)



Le code suivant nécessite 4 coups d'horloge, les deux premières additions sont exécutées **strictement** en parallèle

Nous verrons comment forcer l'exécution en un coup d'horloge si le temps de propagation le permet.

Figure 17-11 Parallélisations et cadencement

EXEMPLE : UN PERIODE- METRE

Si l'on veut mesurer la période de clignotement de notre LED précédente, en passant par des pattes physiques, il suffit de rajouter un nœud de lecture et d'écriture d'une patte du FPGA (par exemple LED1 reliée à Port6/DIO5). Attention, il faut être certain que la durée soit compatible avec la taille du « Tick count » et donc utiliser 16 bits.

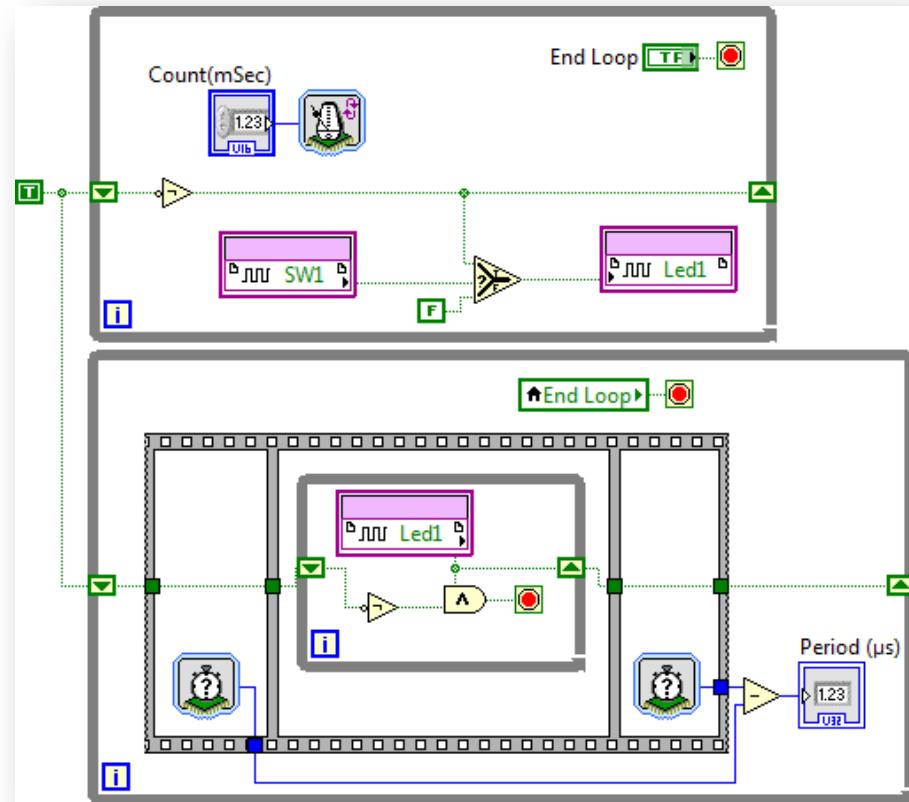


Figure 17-12 Deux boucles parallèles pour la mesure de période

IMPLANTATION D'UN PROTOCOLE SPI

La carte d'essai est équipée d'un convertisseur analogique numérique qui utilise une liaison SPI (§ page 289). Ce protocole très simple utilise une horloge et une ligne de données et dans ce cas précis deux lignes de contrôle, l'une démarre la conversion, l'autre indique qu'elle est terminée. Les temps minimums et/ou maximum entre les fronts des signaux logiques sont donnés dans la datasheet du circuit et doivent être respectés pour obtenir un fonctionnement fiable.

L'implémentation de ce type de protocole fait appel à une structure de type séquence. Les temps seront gérés par des fonctions « wait »

OUTILS DE CADENCEMENT

La palette « Timing » regroupe trois Vis :

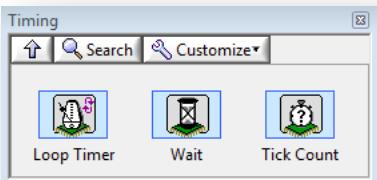


Figure 17-13 Palette timer

Loop Timer attend un temps spécifié entre deux itérations de boucle, elle est destinée à faire des opérations E/S à une vitesse donnée. A la première exécution, il n'y a pas d'attente, mais initialisation du compteur, ensuite il y a attente jusqu'à ce que le compteur atteigne la prochaine valeur (l'ancienne plus le temps d'attente). Si une itération arrive trop tard (le compteur a dépassé la prochaine valeur) il n'y a pas d'attente mais réinitialisation du compteur

Wait Tick Count permet de créer des attentes entre des actions, typiquement dans les communications numériques.

Tick Count renvoie la valeur d'un compteur interne du FPGA, par exemple pour mesurer un temps entre deux événements, ou bien un temps d'exécution. Il faut tenir compte de la profondeur de comptage pour choisir la bonne taille du compteur.

Dépassement de comptage : le tableau suivant donne les durées avant un dépassement de la capacité de comptage pour une fréquence de 40 MHz

Taille du compteur	Unité de temps en coup d'horloge	Unité de temps en μ s	Unité de temps en ms
32 bits	4294967296	71 minutes	49 jours
16 bits	65536	65 ms	65 s
8 bits	256	256 μ s	256 ms

STRUCTURE DE SÉQUENCEMENT

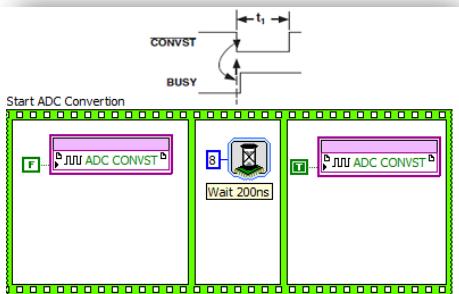
Les structures de séquencement vont permettre de fixer l'ordre chronologique des opérations. Dans un programme non FPGA elles sont à bannir autant que possible, mais dans ce cas, l'utilisation du flot de données (par exemple en reliant les terminaux d'erreur) pour séquencer les opérations impacte très négativement sur l'utilisation des portes.

LE CODE

ORGANISATION

Le chronogramme montre clairement trois phases, le programme comporte donc trois VIs:

LANCÉMENT DE LA CONVERSION



Impulsion négative d'au moins 40ns :



Figure 17-14 Lancement de la conversion

ATTENTE DE LA FIN DE CONVERSION

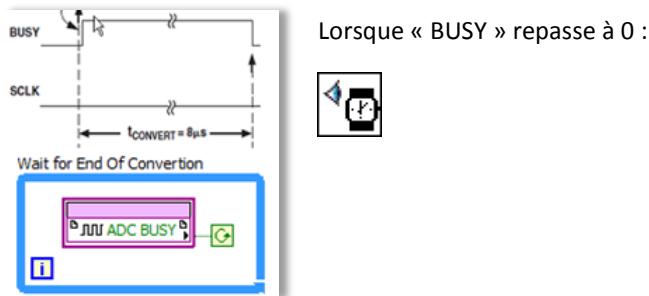


Figure 17-15 Boucle d'attente de fin de conversion

LECTURE DU RÉSULTAT

Envoie de 16 coups d'horloge, lecture de la donnée 60ns minimum après le front descendant. Attente de 400 ns minimum en fin de lecture.

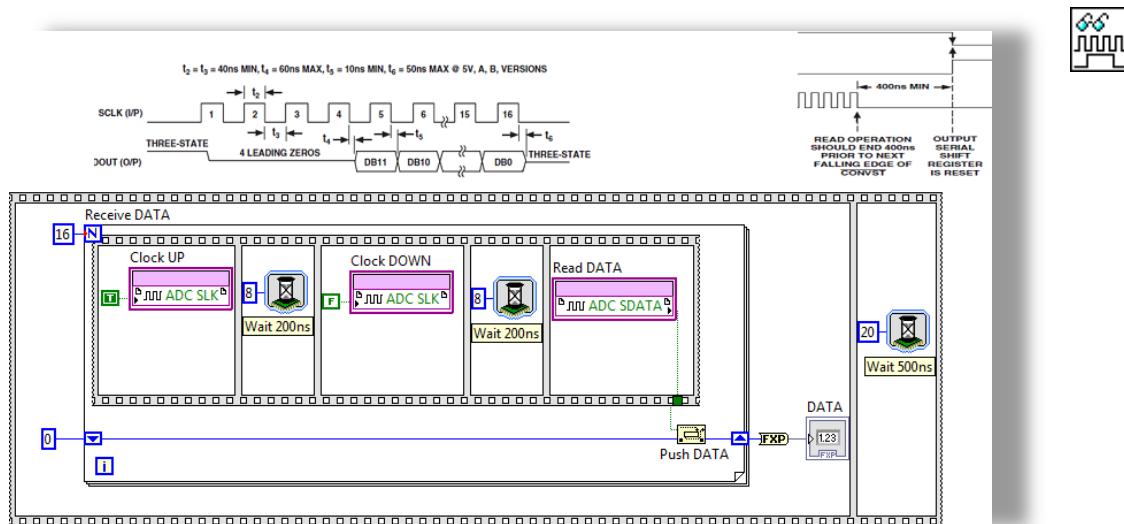


Figure 17-16 Lecture des 16 bits de données

CODE FINAL

Le code final reprend ces trois éléments dans une séquence. Cette dernière est placée dans une boucle infinie, une initialisation du niveau des lignes de commande est ajoutée en tête.

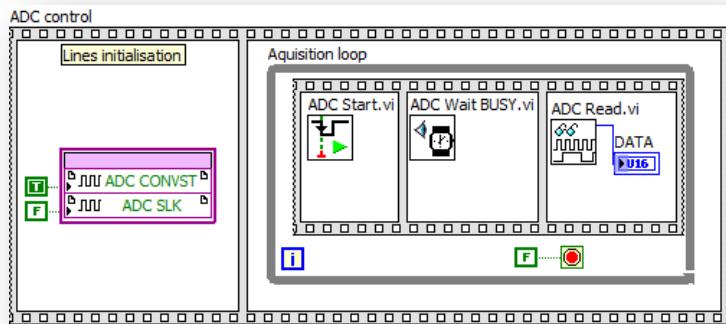
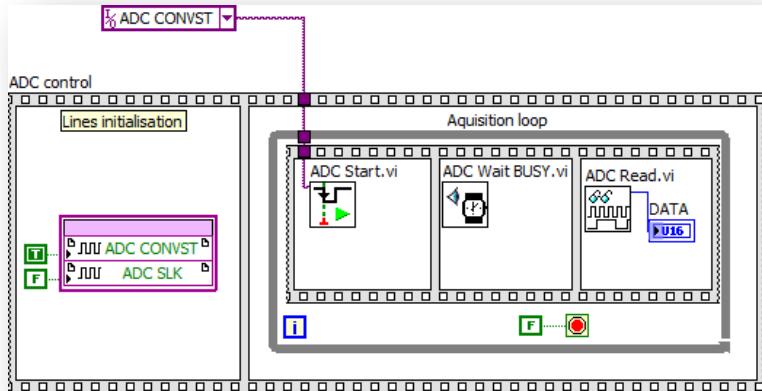


Figure 17-17 Le programme final

ACCES AUX LIGNES PAR DES NŒUDS DE PROPRIÉTÉ

Le code précédent est figé en ce qui concerne les lignes d'entrée/sorties utilisées par les sous VI. Pour créer un code plus générique, il faut placer sur la face avant un contrôle de type « FPGA I/O » puis le relier aux entrées des nœuds d'entrée/sortie.



Il est alors possible de spécifier dans le « main » la ligne à utiliser.

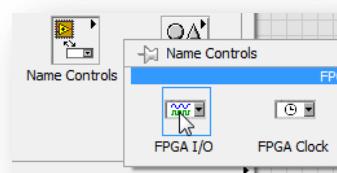


Figure 17-18 Utilisation d'une référence de ligne d'entrée/sortie

GESTION DES TEMPS DE BOUCLES

La gestion du temps d'exécution des boucles est donc une problématique importante. La vitesse d'horloge primaire va définir le cadencement de base (en général 40 MHz avec la possibilité d'utiliser des horloges dérivées grâce aux PLL intégrées dans les FPGA). Puis, c'est le code à l'intérieur de la boucle qui définira le cadencement secondaire.

TEMPS D'EXÉCUTION D'UNE BOUCLE

Le temps d'exécution intrinsèque d'une boucle est de 2 coups d'horloge. Il faut y ajouter le temps d'exécution du code interne. Si deux actions parallèles sont implémentées dans la boucle c'est la plus longue qui limitera le temps de boucle. Les tâches d'entrée/sortie numériques prennent 1 coup d'horloge, les tâches d'entrée/sortie analogiques sur modules CompactRIO environ 35 coups d'horloge (très dépendant du hardware).

DECODEUR BCD 7 SEGMENTS

La carte est équipée de quatre afficheurs 7 segments multiplexés, pour réaliser un décodeur BCD vers les 7 segments, il faut, soit utiliser les équations logiques, soit implanter un tableau de correspondance en mémoire, c'est cette solution qui sera retenue.

LUT

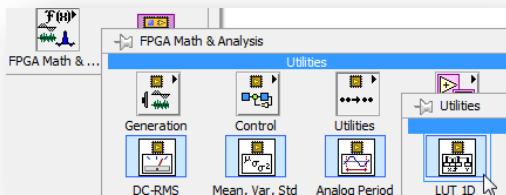


Figure 17-19 Les look Up Tables

LabView FPGA offre une fonction « LUT 1D » qui permet de créer des tables de correspondance, et même d'interpoler des index fractionnaires si nécessaire (pour une table de sinus par exemple)

ENTREE DES DONNEES

Le bouton « Define Table » permet d'entrer les données, soit une à une, soit en créant une droite ou un

sinus/cosinus, ou enfin en générant les valeurs grâce à un VI.

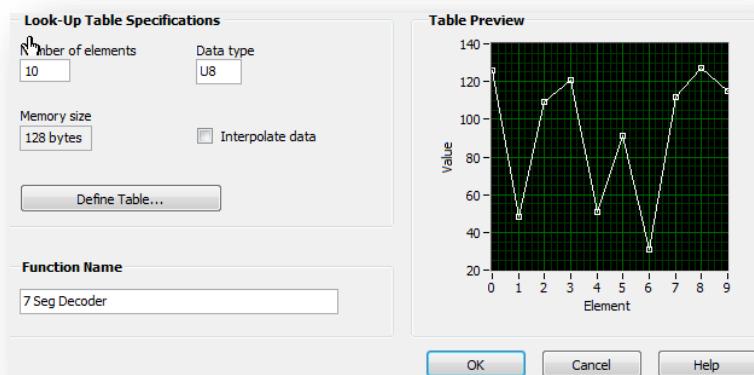


Figure 17-20 Affectations de valeurs à la LUT

La datasheet du CD4511 nous donne,

ce qui nous fait en mémoire :

D ₀	D _C	D _B	D _A	O _a	O _b	O _c	O _d	O _e	O _f	O _g	DISPLAY
L	L	L	L	H	H	H	H	H	H	L	0
L	L	L	H	L	H	H	L	L	L	L	1
L	L	H	L	H	H	L	H	H	L	H	2
L	L	H	H	H	H	H	H	L	L	H	3
L	H	L	L	L	H	H	L	L	H	H	4
L	H	L	H	H	L	H	H	L	H	H	5
L	H	H	L	L	L	H	H	H	H	H	6
L	H	H	H	H	H	H	L	L	L	L	7
H	L	L	L	H	H	H	H	H	H	H	8
H	L	L	H	H	H	H	L	L	H	H	9

Data Points	
Element	Value
0	126
1	48
2	109
3	121
4	51
5	91
6	31
7	112
8	127
9	115

Figure 17-21 Valeurs définies pour le décodeur BCD->7 segments

UTILISATION

Le décodeur est prêt à fonctionner, il faut extraire de l'entier 8 bits les valeurs de chaque segment, a étant le bit de poids 6 et g le bit de poids 0.

EXECUTION SUR UNE AUTRE CIBLE

L'implantation d'équations logiques complexes entre de nombreuses variables est souvent source d'erreur. Il serait souhaitable de pouvoir tester notre décodeur sur le PC avant de lancer une longue compilation sur le FPGA. Pour cela il faut choisir dans le menu local de la cible FPGA la cible souhaitée. Lorsque le PC est choisi

comme cible d'exécution, les nœuds d'entrée/sortie sont évidemment inutilisés (certaines cartes de série R acceptent cependant des E/S physiques réel sur le FPGA). Les nœuds d'entrée fournissent des données aléatoires ou provenant d'un VI de définition, les nœuds de sortie sont ignorés. Les timings ne peuvent être respectés, il s'agit bien de tester la logique et non la vitesse. Un afficheur 7 segments est créé sur la face avant du VI pour vérifier le résultat :

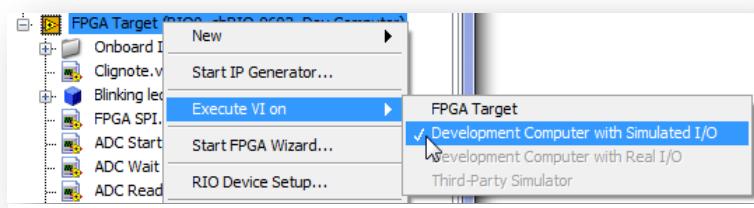


Figure 17-23 Exécution du code sur la machine de développement

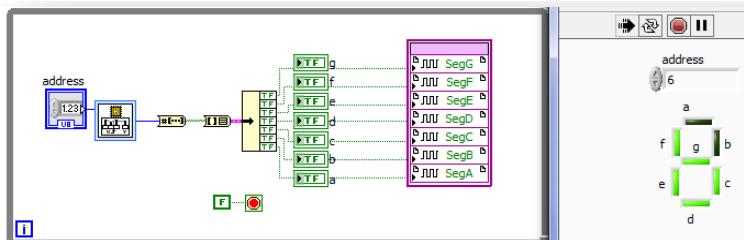


Figure 17-22 Résultat obtenu

MULTIPLEXAGE

L'ADC fournit un résultat sur 12 bits, donc un chiffre compris entre 0 et 4095. Les données sont affichées sur des afficheurs multiplexés à anode commune. Un bit par afficheur commande un transistor PNP qui relie l'anode au +5V, ce bit doit être à 0 pour activer l'afficheur correspondant.

Une solution simple et efficace consiste à prendre un vecteur de 4 bits « 0001 » et à décaler le dernier 1 en utilisant la fonction « Scale By Power Of 2 »

VECTEURS DE TAILLE QUELCONQUE

La palette des entiers permet de créer des nombres de taille 8, 16, 32 et 64 bits. Pour créer des vecteurs de

taille quelconque (inférieure à 64 bits), il faut utiliser une autre représentation, les nombres en virgule fixe. Ils permettent de coder des nombres de taille quelconque avec une partie entière et une partie réelle fixe. Ces



Figure 17-24 La palette "Fixed Points"

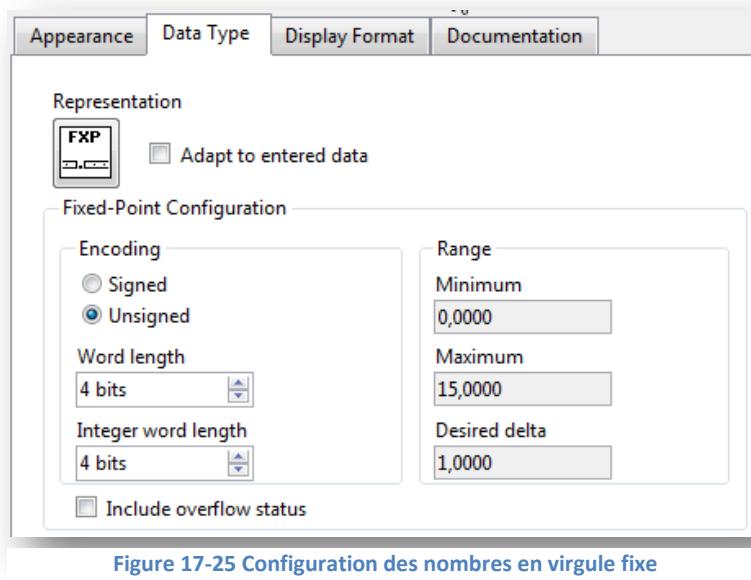


Figure 17-25 Configuration des nombres en virgule fixe

paramètres sont fixés dans l'onglet « Data Type » des propriétés. Ce sont les seuls nombres à **virgule disponibles dans le module FPGA** (les opérations sur nombres en virgule flottante sont consomment trop de portes). Pour les opérations mathématiques sur ces nombres, une palette optimisée nommée « High Throughput Math Functions » utilise au mieux les ressources du FPGA (accumulateurs, DSP...) pour optimiser la vitesse d'exécution.

IMPLEMENTATION

Les quatre chiffres à afficher sont contenus dans un tableau (attention les tableaux sont de taille fixe dans le module FPGA -> onglet « size » des propriétés). Une boucle les désindexe tout en activant le digit correspondant.

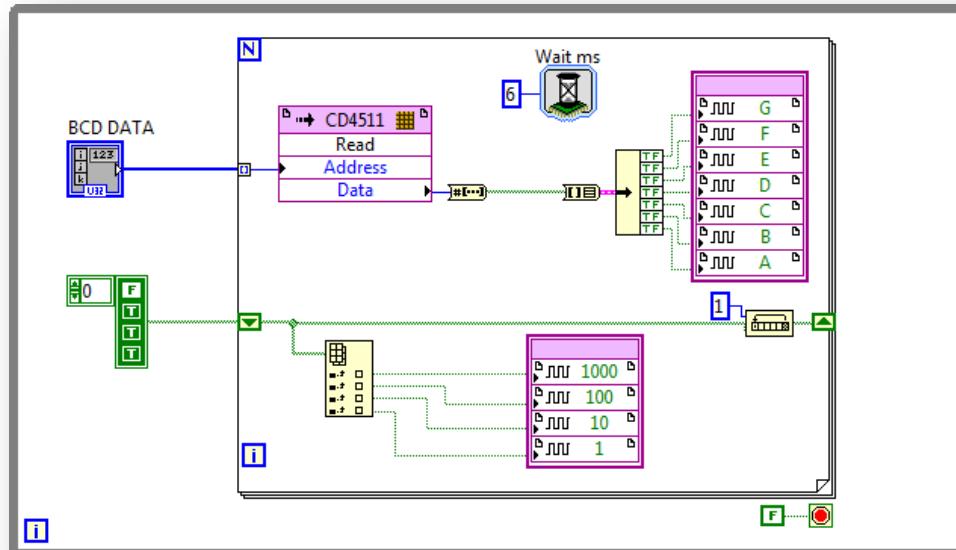


Figure 17-26 Le code final du décodeur avec multiplexage.

LE DECODEUR BINAIRE/BCD UTILISATION D'UNE IP VHDL

Tout le code implémenté actuellement est écrit en LabView et sera traduit en VHDL lors de l'implémentation. Si vous disposez d'IP écrits en VHDL, et dument testées avec des outils tiers ('ISim fait partie de la licence FPGA LabView avec ISE), vous pouvez intégrer directement des IP dans votre code. Deux moyens sont disponibles, les nœuds d'intégration IP, et les CLIP « Component-Level Intellectual Property ».

NŒUD D'INTEGRATION IP VERSUS CLIP

Il existe des différences assez fondamentales en ces deux méthodes. Les nœuds d'intégration IP sont sur le diagramme et leur exécution dépend donc du flot de donnée, leur code peut être simulé sur l'ordinateur de développement. A contrario, un CLIP est un module qui « vit sa vie » indépendamment du code LabView, il sera compilé par l'utilitaire Xilinx et ne peut être simulé sur le PC de développement. Le tableau suivant résume ces différences :

	CLIP	IP Integration Node
Supported synthesis file formats	.vhd, .ngc, .ucf  Note You cannot use .ngc or .xco files as top-level synthesis files.	.vhd, .ngc, .xco
Support for <u>simulation</u>		
Supported LabVIEW data type	<ul style="list-style-type: none"> • Boolean • Integer • Fixed-point  Note CLIP sockets for certain FPGA hardware targets support the Boolean array data type. Refer to the target hardware documentation for information about CLIP socket support.	<ul style="list-style-type: none"> • Boolean • Boolean array • Integer • Fixed-point
Execution model	Executes parallel to, and independent of, the <u>data flow</u> of an FPGA VI	Executes as defined by the data flow of an FPGA VI
Place of declaration and behavior	Declared in a LabVIEW project; acts as a global	Declared locally in an FPGA VI
Supported execution modes	<ul style="list-style-type: none"> • Outside single-cycle Timed Loop • Inside single-cycle Timed Loop 	Inside single-cycle Timed Loop
Support for <u>multiple clock domains</u>	Maximum number of clocks defined by FPGA	Maximum of two clocks: an SCTL clock and an FPGA-derived clock, where the derived clock executes at a rate that is an integer multiple of the SCTL clock
Support for accessing clocks from the IP as clocks for a single-cycle Timed Loop		
Support for VHDL generics		
Support for constraints on IP		
Support for accessing FPGA I/O		

Figure 17-27 Principales différences entre CLIP et IP Integration Node

Pour faire très simple, disons que le CLIP est plus proche du composant FPGA et de sa suite de compilation alors que le nœud d'intégration IP est plus proche de LabView.

LE CODE VHDL

Le code suivant est celui d'un décodeur 12 bits binaires vers 4 digits décimaux. Il utilise l'algorithme « Double-Dabble ». Il servira à afficher la valeur de la tension lue.

```

-----
-- Company: ICMCB CNRS
-- Engineer: R. DECOURT
--
-- Create Date: 9-2011
-- Design Name: Binary-to-BCD Converter
-- Target Device: SBrio
-- Tool versions: 12.4
-- Description:
-- A 12 bits binary to BCD converter for Labview CLIP IP Integration
-----

-- Title: Binary-to-BCD Converter
library IEEE;
use      IEEE.std_logic_1164.all;
use      IEEE.std_logic_unsigned.all;

entity Binary_to_BCD is
    port (
        Bin   : in STD_LOGIC_VECTOR (11 downto 0); --Binary input
        Unit  : out STD_LOGIC_VECTOR (3  downto 0); --LSB Decimal digit
        Dix   : out STD_LOGIC_VECTOR (3  downto 0);
        Cent  : out STD_LOGIC_VECTOR (3  downto 0);
        Mille : out STD_LOGIC_VECTOR (3  downto 0)           --MSB Decimal digit
    );
end Binary_to_BCD;

architecture behavioral of Binary_to_BCD is
begin

    Shift_Add3 : process(Bin)
        variable temp: STD_LOGIC_VECTOR (27 downto 0);           --Use to store the 12 bits
    in and the

        begin
            temp  := "00000000000000000000000000000000";          --Init for shifting
            temp(14 downto 3) := Bin;                                --reduce the number of shift
            --Main Loop

            for i in 0 to 8 loop
                if temp(15 downto 12) > 4 then
                    temp(15 downto 12) := temp(15 downto 12) + 3;
                end if;
                if temp(19 downto 16) > 4 then
                    temp(19 downto 16) := temp(19 downto 16) + 3;
                end if;
                if temp(23 downto 20) > 4 then
                    temp(23 downto 20) := temp(23 downto 20) + 3;
                end if;
                if temp(27 downto 24) > 4 then
                    temp(27 downto 24) := temp(27 downto 24) + 3;
                end if;
            end loop;

            Mille <= temp(27 downto 24);
            Cent  <= temp(23 downto 20);
            Dix   <= temp(19 downto 16);
            Unit  <= temp(15 downto 12);

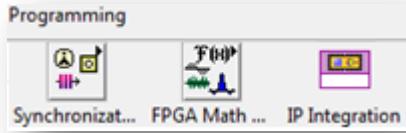
        end process Shift_Add3;
    end behavioral;

```

Ce fichier doit être incorporé dans le programme labview pour pouvoir être compilé et exécuté.

Nœud d'intégration IP

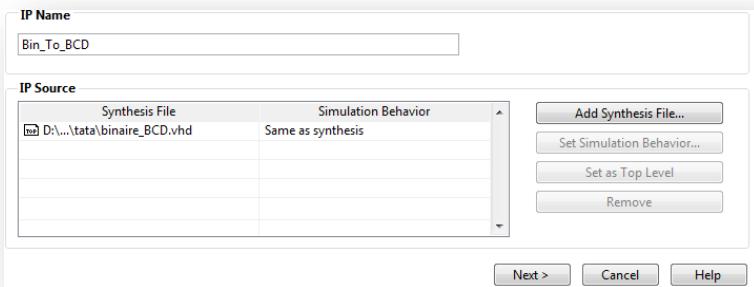
Le nœud d'intégration IP est disponible sur la palette de programme. Une fois déposé sur le diagramme il faut spécifier divers éléments en ouvrant une fenêtre de dialogue par un double clic.



Il faut spécifier le nom de l'IP (tous les IP doivent avoir un nom différent), est donner le chemin d'accès au fichier .vhd

Figure 17-28 noeud d'intégration IP sur la palette

Puis définir l'entité et l'architecture à utiliser (il peut y avoir plusieurs couples entité/architecture dans le .vhd).



Le volet suivant permet de définir la valeur des variables instanciées en tant que génériques dans l'entité. De vérifier la syntaxe et de générer les fichiers nécessaires à la compilation. Volet suivant est pour la définition des horloges (pour simplifier aucune horloge n'est définie, il serait bon de synchroniser le code sur un CLK'event) et éventuellement pour donner le nom d'un variable

Figure 17-29 Sélection de l'entité souhaitée

d'activation de l'IP (utile uniquement si le nœud d'appel est dans une structure de choix). Volet suivant, définition des conditions de reset : il n'y en a pas. Puis la définition des noms et des types des terminaux (prédéfinis d'après la déclaration de port de l'entité VHDL). Une fois cette opération terminée, le nœud d'intégration IP présente des terminaux d'entrée et de sortie. Il est exécutable sur la cible ainsi que sur l'ordinateur de développement. Il doit être nécessairement placé dans une boucle « Single Cycle Loop ». Ces boucles suppriment la gestion du flot de données et assure que l'exécution se fait en un cycle d'horloge

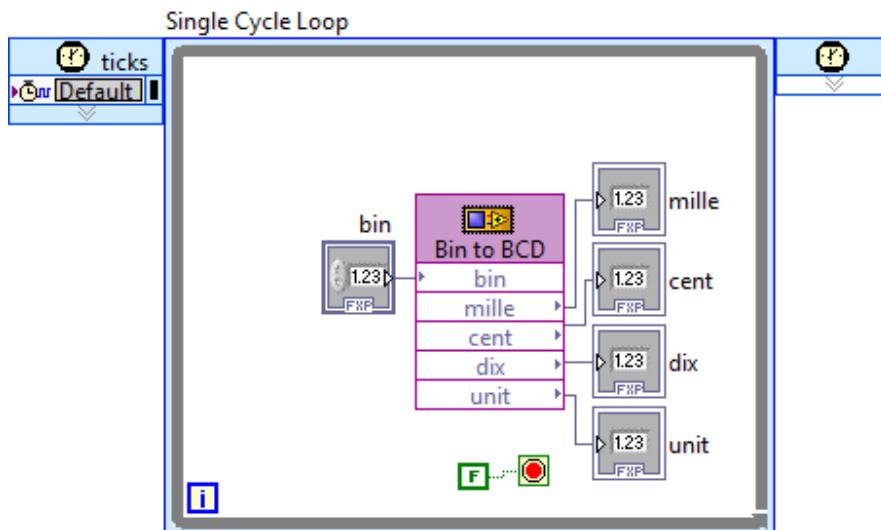


Figure 17-30 Utilisation du Nœud d'intégration IP dans une SCL

CLIP

Un CLIP est défini au niveau du projet, dans les propriétés de la partie FPGA, item Component-Level-IP. Les informations sont sauvegardées dans un fichier XML. Qu'il faut tout d'abord créer bouton « Create File »

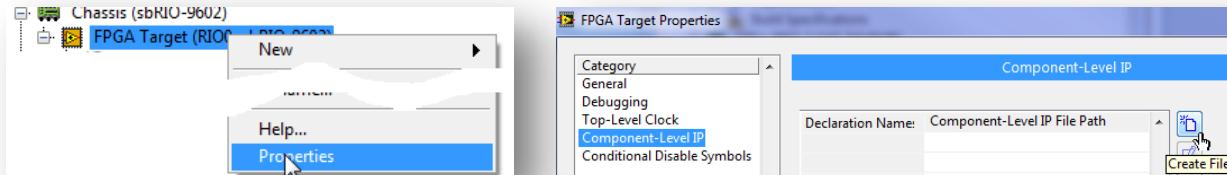


Figure 17-31 Crédit d'un CLIP

Un ensemble de pages assez semblables aux précédentes permettent de définir l'IP, entité et l'architecture voulue. Le bouton « Check Syntax » doit être validé pour pouvoir continuer la procédure.

Pour une approche simple (il n'y a pas d'horloges dérivées, le clip n'utilise pas d'E/S physiques....), la procédure se limite à cliquer sur le bouton « Next », jusqu'à « Finish ».

Notre composant est maintenant disponible, pour l'utiliser il faut, dans le menu local de la partie FPGA du projet sélectionner « New->Component Level IP ».

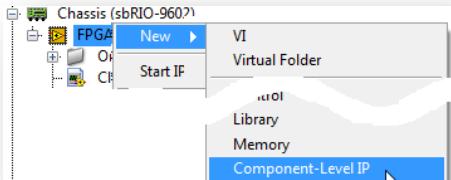


Figure 17-33 Utilisation d'un CLIP

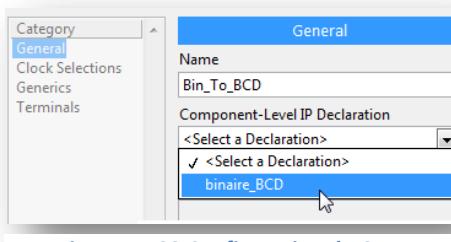


Figure 17-32 Configuration du CLIP

Seule la page « General » est à renseigner, Il faut nommer l'instance du CLIP et sélectionner quelle IP utiliser.

Cette procédure fait apparaître le composant « Bin_To_BCD » dans le projet. Ce composant sera implémenté lors du déploiement vers la cible et fonctionnera **indépendamment** du programme LabView. Par exemple, si l'IP ne communique avec le monde extérieur que par des pattes d'E/S du FPGA (une relation combinatoire entre des entrées et une sortie) vous ne verrez pas d'autre chose sur le diagramme. Dans notre cas, l'IP communique avec le code LabView puisqu'elle reçoit un nombre binaire et renvoie quatre vecteurs de 4 bits.

L'accès à ces données utilise des noeuds d'E/S (comme pour les pattes physiques), il suffit donc de mettre dans une boucle infinie une structure de séquencement qui va donner une valeur à « bin » puis lire les quatre chiffres.

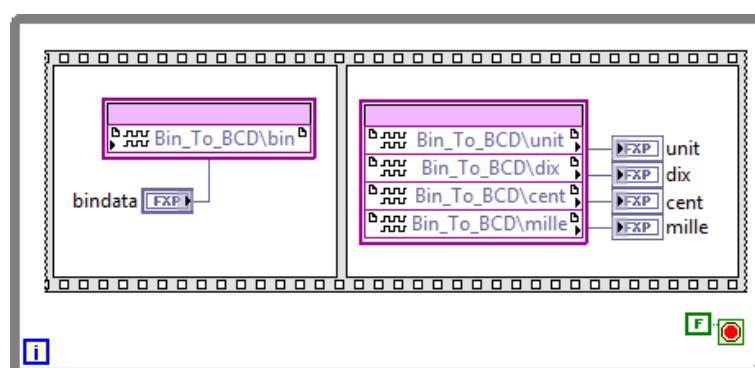


Figure 17-34 Utilisation d'un CLIP

AFFICHAGE DU RESULTAT DE LA CONVERSION

Actuellement, nous avons trois boucles séparées, l'une pour la conversion, la conversion BCD, l'autre pour l'affichage. Considérons que cette situation est nécessaire car les deux opérations sont cadencées séparément (par exemple la boucle d'affichage est plus lente que celle d'acquisition). Ces boucles étant infinie, il faut trouver un moyen de passer le résultat de la conversion au VI d'affichage.

PASSAGE DE DONNEES ENTRE BOUCLES

Les ressources pouvant être partagées sont :

Les E/S numériques et analogiques (dépendant du matériel)

Les blocs mémoire

Les FIFO

Les VI non réentrant (variables globales fonctionnelles)

Les variables globales et locales.

LES VARIABLES GLOBALES ET LOCALES :

Elles sont stockées dans les bascules du FPGA, c'est le bon choix si vous n'avez pas besoins d'un historique des valeurs. La variable locale n'est

accessible qu'au sein de son VI, la globale est accessible par tous les VI implantés sur le FPGA. Evitez les variables locales dans le VI principal FPGA car elle apparaît sur la face avant et les données sont donc transmises à la cible RT (sauf si vous en avez besoin !).

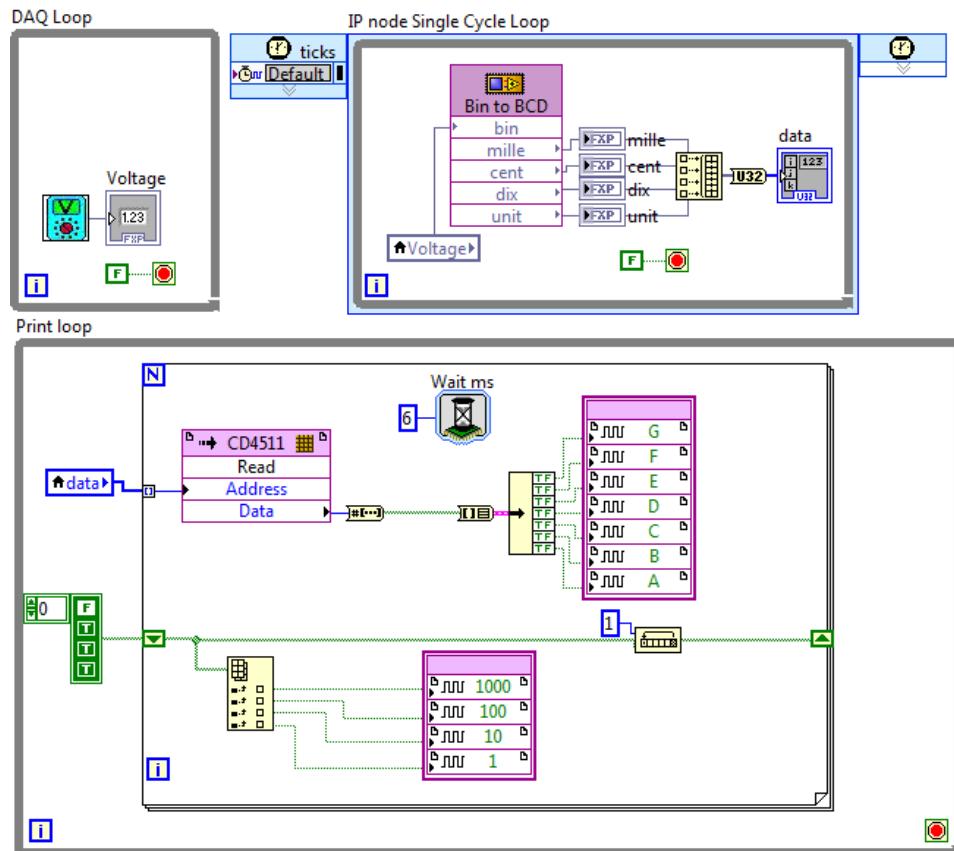


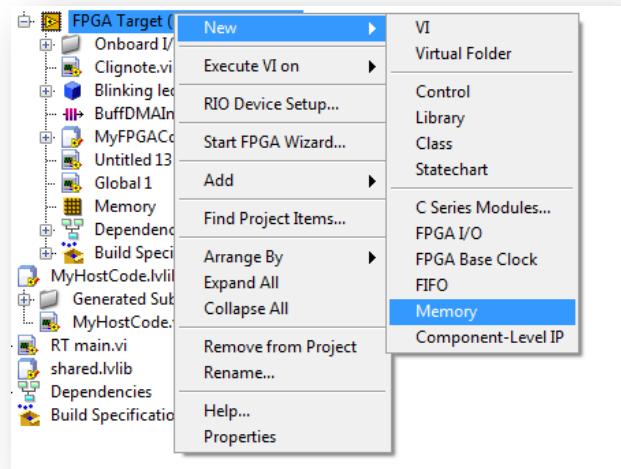
Figure 17-35 Utilisation d'une variable locale

LES BLOCS MEMOIRE

Les FPGA possèdent des ressources mémoire. Il y a toujours la possibilité d'implémenter les ensembles de données sous forme de tableau utilisant les LUT, mais les tableaux volumineux sont consommateurs de ressources. Un tableau comportant 100 éléments de nombres de 32 bits consomme 30 % des bascules d'un Virtex-II 1000 et moins de 1 % du bloc de RAM embarquée.

CREATION D'UN BLOC MEMOIRE

Pour créer un bloc : il faut sélectionner dans le menu contextuel de la cible FPGA « New->Memory »



Une fenêtre permet de définir la taille en nombre d'éléments, l'implémentation (en mémoire dans les bascules d'une LUT), le type de données et l'accès à la mémoire. Les blocs mémoire ont deux interfaces, normalement une pour la lecture, l'autre pour l'écriture, mais si la mémoire est initialisée (une forme d'onde, des coefficients...) il est alors possible de lire deux emplacements simultanément. Les menus déroulants arbitration définissent le comportement lors d'un accès commun aux interfaces.

Figure17-36 Crédit d'un bloc mémoire

Always Arbitrate : implémente un algorithme type « round robin » qui assure un accès séquentiel à la mémoire en lecture et en écriture.

Arbitrate if Multiple Requestors Only : implémente un arbitrage d'accès uniquement si plusieurs accesseurs existent dans le code, même s'ils n'ont pas la possibilité d'un accès simultané.

Never Arbitrate : n'implémente pas de code d'arbitrage (il faut alors être certain de ne pas avoir d'accès simultané).

Enfin l'onglet « initial values » permet de définir de façon statique les données initiales contenue dans la mémoire.

ACCES AUX DONNEES

Dans la palette « Memory & FIFO » vous trouvez les nœuds de méthode pour accéder aux données. Il n'en existe que deux : « Read et Write ! »



Figure 17-37 Nœud d'accès Read/Write à la mémoire

Attention, il n'y a pas de code prévenant d'une corruption des données entre deux domaines temporels différents (sur des FPGA qui peuvent générer

plusieurs horloges via des PLL)

Les blocs mémoire ainsi définis sont dits « target scoped » et sont accessibles de tous les Vis implantés sur le

FPGA. Il existe des blocs mémoire définis dans le code d'un VI qui ne sont accessibles que du VI de définition et appellés « VI scoped ». Ils sont identiques aux précédents mais permettent d'éviter des conflits de noms entre divers Vis.

Les mémoires seront utilisées comme les variables locales ou globales, lorsque la perte d'une donnée n'est pas critique et que le volume de

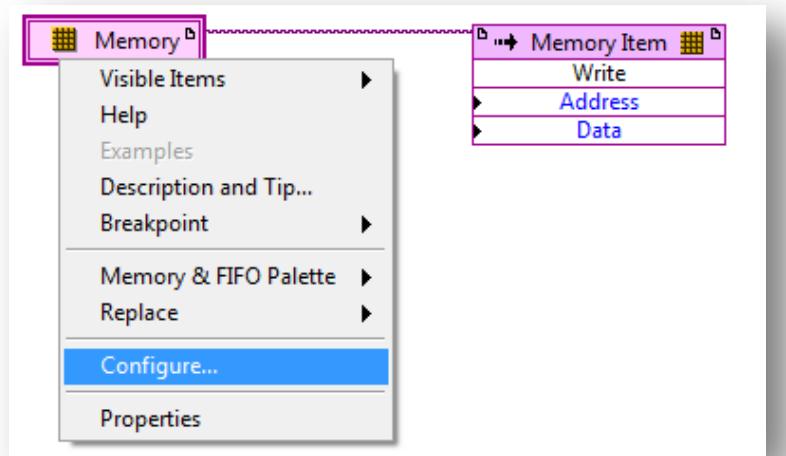


Figure 17-38 Bloc mémoire défini au niveau du VI

données est important.

EXEMPLE D'UN BUFFER CIRCULAIRE

Considérons que la boucle d'acquisition stocke les valeurs dans un buffer circulaire de 256 mots de 12 bits et que celle d'affichage fasse la moyenne de ces huit données pour les afficher (aucun système de control de la validité des données ne sera implémenté !).

La partie acquisition se retrouve avec une méthode « Write » de la mémoire que vous avez créé (un glisser déposer depuis le bloc mémoire du projet vers le diagramme suffit à créer le nœud de méthode), ainsi qu'un compteur trois bits.

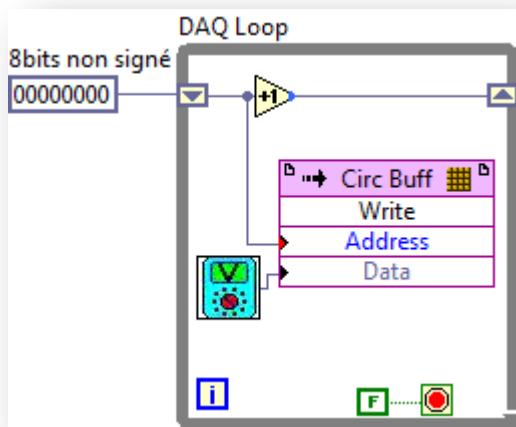


Figure 17-39 Implantation d'un buffer circulaire en mémoire sur le VI d'acquisition

éviter les erreurs de dépassement. La structure for étant par définition pas exécutable en une passe, elle ne peut être dans la boucle SCL. Une boucle « while » infinie englobe donc la première boucle. Attention à modifier la condition d'arrêt de la boucle SCL.

La fonction d'incrémentation nécessite d'être configurée pour que sa sortie reste sur trois bits et que son mode de dépassement soit « Wrap » et non « Saturate ». Ces paramètres sont dans l'onglet « Output configuration » des propriétés de la fonction.

La boucle de conversion BCD s'enrichie d'une structure de répétition inconditionnelle pour sommer les 256 valeurs et d'une division par huit réalisée par un décalage à droite de trois bits. Prendre garde à faire une addition sur 21 bits pour

Pour une simulation sur PC il convient d'insérer des fonctions d'attente dans les deux diagrammes.

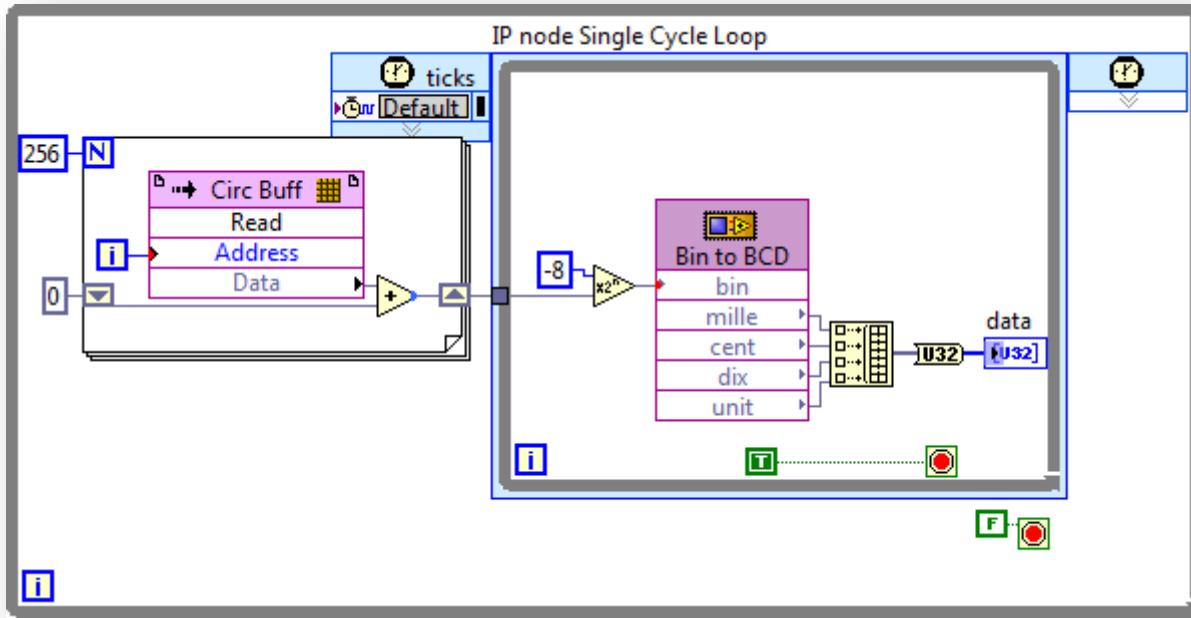


Figure 17-40 Calcul de moyenne des données en mémoires

FIFO

CREATION D'UNE FIFO

Le processus de création est identique (il faut sélectionner FIFO au lieu de memory). La FIFO possède le code nécessaire pour vérifier l'intégrité des données, tant dans le contenu que dans le nombre. Elles peuvent de même être « Target Scope » ou « VI scoped ». Les FIFO peuvent être implantées dans les bascules pour obtenir les meilleures performances mais avec une taille limité (~100 Octets) dans les LUT (~300 Octets) ou dans les blocs mémoire pour libérer de la logique combinatoire pour les Vis (moindre performance en vitesse)

ACCES AUX DONNEES

Les méthodes d'accès possèdent une entrée et une sortie « Time Out », L'entrée fixe le nombre de coup d'horloge que doit attendre la FIFO si elle est pleine en écriture ou vide en lecture. En sortie le booléen est vrai

si aucun élément n'a pu entrer ou sortir dans le laps de temps imparti. Aucun élément n'est écrasé en écriture, il y a possibilité de pertes de données si la FIFO sort en « TimeOut »



Figure 17-41 Nœuds d'accès "Read/Write" des FIFO

Pour éviter les débordements, il convient de ne pas écrire plus d'éléments que de place disponible, ni de lire plus d'éléments que de données engrangées. Pour cela deux méthodes sont disponibles :

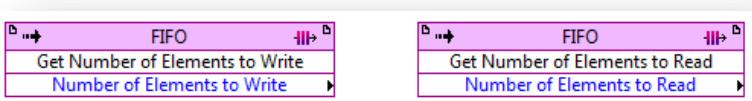


Figure 17-42 Accès au nombre d'éléments dans la FIFO

Enfin une méthode « Clear » permet de vider complètement une FIFO.

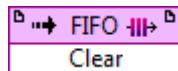
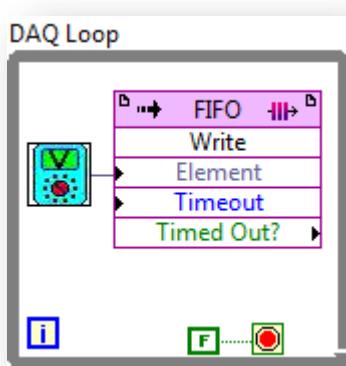


Figure 17-43 Effacement de la FIFO

EXEMPLE AVEC UNE FIFO

Reprenons l'exemple précédent. Cependant, au lieu de cadencer l'exécution de la boucle d'affichage par une attente huit fois plus longue que la boucle d'acquisition, elle sera cadencée par le nombre d'éléments présents dans la FIFO.



La partie acquisition se retrouve avec une méthode « Write » de la FIFO (la FIFO a une taille de 256 éléments).

On suppose qu'aucun débordement ne peut avoir lieu, donc pas de contrôle du « Time Out ».

Figure 17-44 Stockage des données dans un FIFO

La boucle d'affichage s'enrichie d'une structure de répétition conditionnelle qui attend que le buffer contienne au moins 256 valeurs avant d'entrer dans la boucle de vidange de la FIFO. L'affichage est cadencé à 100ms pour voir le résultat à l'écran de l'ordinateur. Quatre digits à 100ms c'est plus court que 8 mots de 12 bits à 100ms donc pas de débordement de la FIFO.

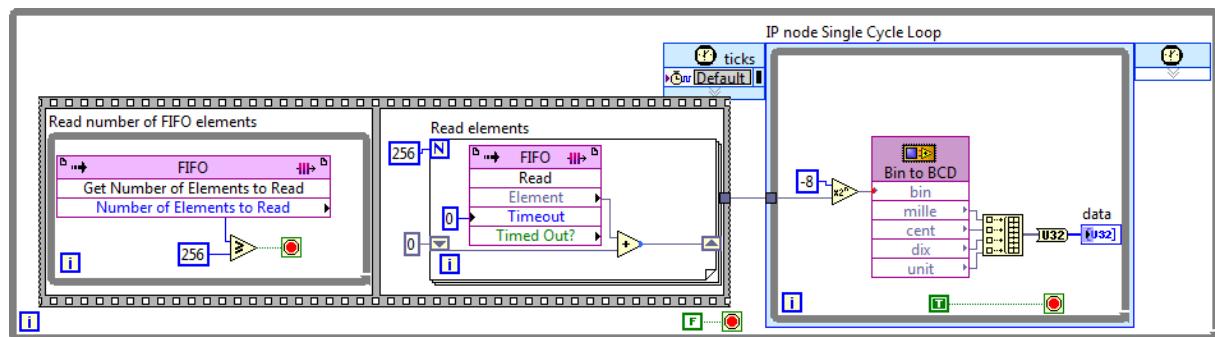


Figure 17-45 Une boucle attend que 256 éléments soient dans la FIFO

Il est temps maintenant de compiler le projet pour l'exécuter sur la cible FPGA.

PROTOCOLE I2C

Le protocole I2C est utilisé par le convertisseur Numérique/analogique. Il s'agit d'un protocole qui utilise que deux fils actifs, une horloge SCL et une ligne de donnée SDA, il est multi-maitres et multi-esclaves. La vitesse standard est de 100 kbits/s les adresses sont codées sur 7bits.

Initialement développé pour la télévision par Philips, il est maintenant très largement diffusé. Puisque qu'une ligne unique est destinée aux données celle-ci est nécessairement bidirectionnelle, ceci impose certaines contraintes au niveau du FPGA.

LIGNES BIDIRECTIONNELLES

Les nœuds de lecture écriture permettent de configurer ces lignes en entrée ou en sortie. Les blocs d'entrée sortie du FPGA sont constitués d'une bascule et d'un buffer trois états.

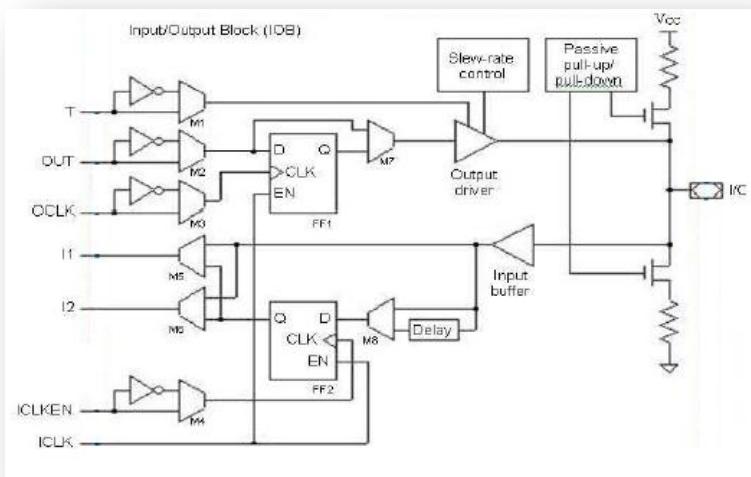


Figure 17-46 Structure des connexion d'E/S

Il faut utiliser le nœud de méthode « **Set Output Enable** » pour forcer le passage haute impédance dans le cas d'une ligne bidirectionnelle (Palette FPGA I/O).

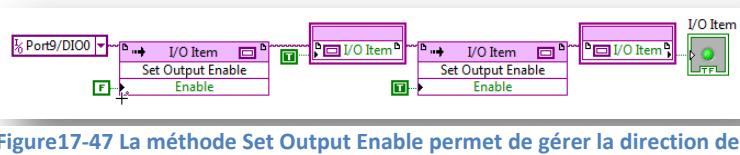


Figure 17-47 La méthode Set Output Enable permet de gérer la direction des échanges

Il est aussi possible de poser une valeur dans la bascule de sortie sans activer le buffer de sortie grâce au nœud de propriété « **Set Output Data** » (par exemple une partie du code génère des données et une autre active ou non la sortie).

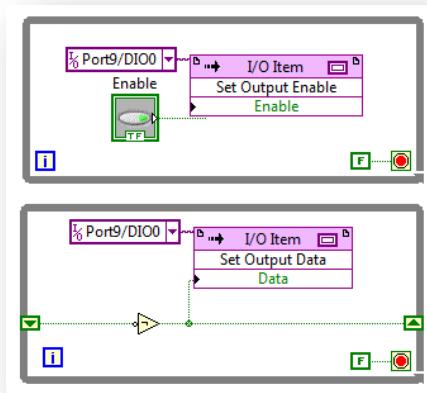


Figure 17-48 la méthode "Set Output Data" permet d'attribuer un niveau à la bascule de sortie sans activer la broche physique

ENVOIE RECEPTION DE DONNEES

Le but n'étant pas ici d'apprendre le fonctionnement du bus I2C, mais l'utilisation de lignes bidirectionnelles, l'exemple suivant se contentera de transmettre une trame d'écriture. L'adresse I2C du DAC est 14, SDA est reliée à Port5/DIO4 et SCL à Port5/DIO5.

ECRITURE D'UNE TRAME

La trame transmise doit être :

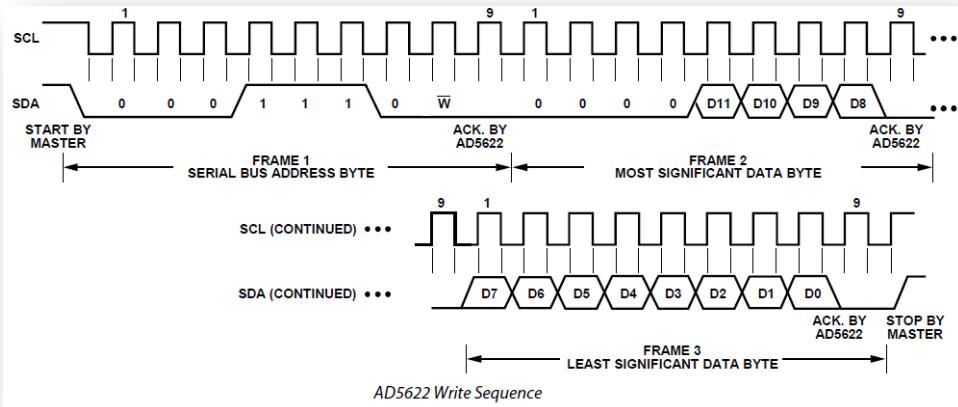


Figure17-49 Trame d'écriture des douze bits de données

La première phase une est condition de « Start » ->SDA passe de 1 à 0 alors que SCL reste à 1.

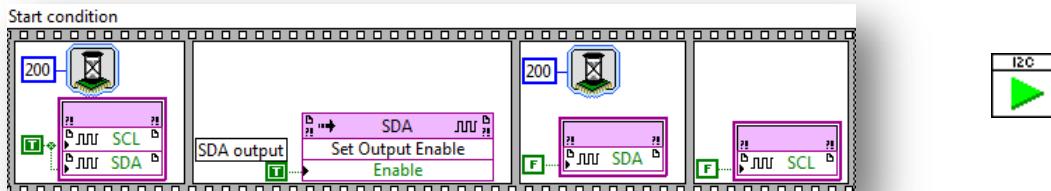


Figure 17-50 Envoi d'une condition de "Start"

Puis l'adresse est transmise sur 7 bits suivie d'un 0 pour indiquer une opération d'écriture.

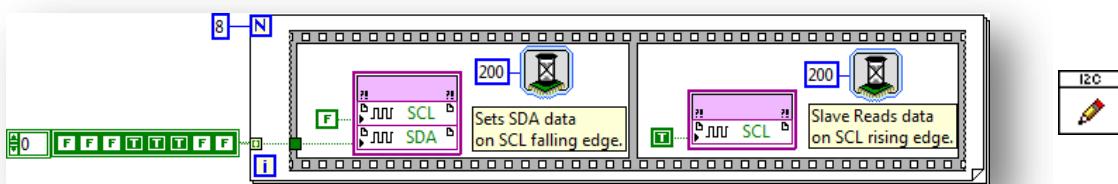


Figure 17-51 Transmission de l'adresse 14 et d'un « Write »

Puis le maître lit l'acquittement de la transmission par l'esclave (en cas de non acquittement nous ne ferons rien pour simplifier le diagramme, normalement une condition de « Stop » est envoyée et une transaction recommence). SDA étant en lecture il faut envoyer un faux au noeud de propriété « Set Output Enable ».

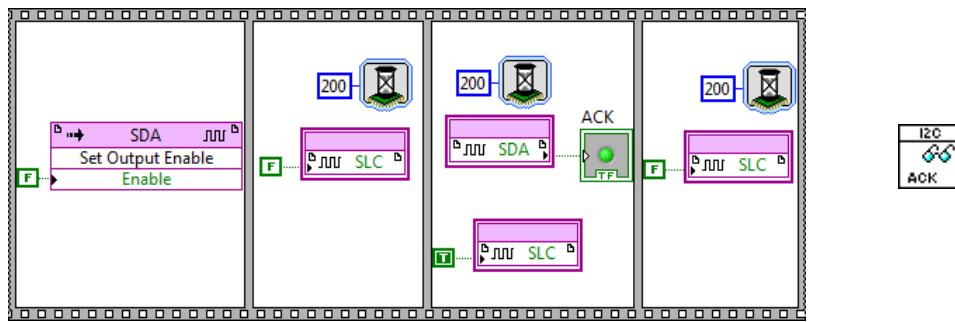


Figure 17-52 Lecture de l'Acknoledge"

Le cycle envoie d'un octet, réception de l'ACK recommence ensuite deux fois, mais avec les données et non l'adresse. Puis une condition de « Stop » termine la transaction, SDA passe à 1 après que SCL soit passé à 1.

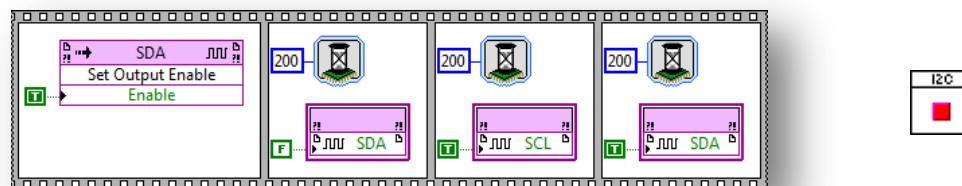


Figure 17-53 Envoie d'une condition de "Stop"

La séquence finale ressemble donc à ceci :

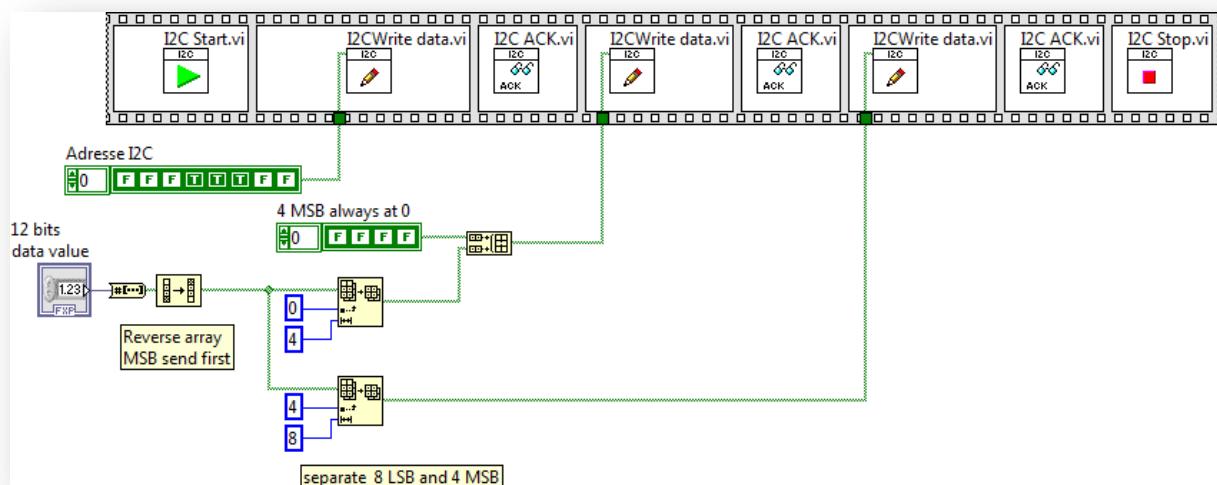


Figure 17-54 Séquence complète d'une transaction d'écriture sur bus I2C

OPTIMISATION DU CODE FPGA

OPTIMISATION GENERALE

Les FPGA ont des ressources limitées, et une opération banale comme une division peut s'avérer très couteuse en portes. Il faut avant tout :

LIMITER LA TAILLE ET LE NOMBRE DES OBJETS DE LA FACE AVANT

Tous les objets de la face avant doivent être utilisés : enlevez les objets de débogage, les objets ne servant qu'à créer des variables locales.

Utilisez le moins possible d'objets, par exemple prenez un U8 pour transférer 8 booléens ou un tableau plutôt que huit booléens individuels (le code associé aux contrôles utilisent souvent plus de mémoire que la taille de la variable qu'ils transportent)

Chaque opération de lecture/écriture implique un hachage des données transférées depuis la face avant du Vi FPGA en paquets de 32 bits (si vous transférez 33 bits, vous doublez le temps de transfert).

Bannissez les tableaux et cluster de plus de 32 bits qui nécessite des copies pour leur transfert (utilisez des FIFO DMA)

UTILISER LES TYPES LES PLUS PETITS

Oubliez « plus c'est gros mieux c'est », réduisez les données au strict nécessaire

Utilisez les VI de changement de type plutôt que la coercition automatique

EVITEZ LES FONCTIONS GOURMANDES

Faire des décalages à droite ou gauche plutôt que des division/multiplication par 2^n

Faire des compteurs plutôt qu'une division avec reste pour obtenir un modulo

EVITEZ DE CABLER LES ERREURS

Dans la mesure du possible ne faites apparaître les terminaux d'erreur (menu contextuel du nœud de méthode « Hide/Show Error Terminal ») que sur les fonctions critiques pour le système, chaque liaison d'erreur entraînera l'affectation de bascules pour les enregistrer !

Utilisez des séquences pour organiser l'enchaînement des fonctions plutôt qu'un lien d'erreur.

Traitez les erreurs immédiatement, évitez de les passer au système RT

Vous traiterez les erreurs par exemple :

Pour reconnaître un module absent

Gérer un timeout de communication d'un module CAN ou encore l'accès à un fichier d'un module de stockage.

BOUCLES EN UN CYCLE

La structure « Single Cycle Timed Loop » (palette de structures SCTL) permet d'optimiser l'exécution en forçant le synthétiseur, pour cette partie du code, à optimiser la vitesse et la taille du code.

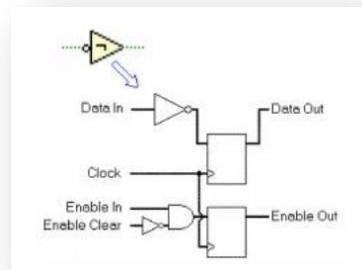


Figure 17-55 Structure de la chaîne d'activation

Normalement, le flot de données créé en LabView pour le FPGA est maintenu par trois éléments :

- La fonction logique codée dans la LUT
- La bascule de synchronisation
- La chaîne d'activation

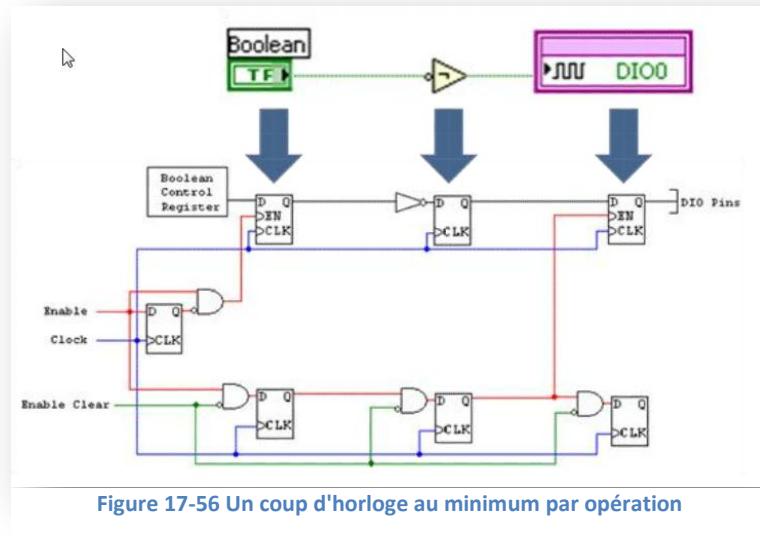
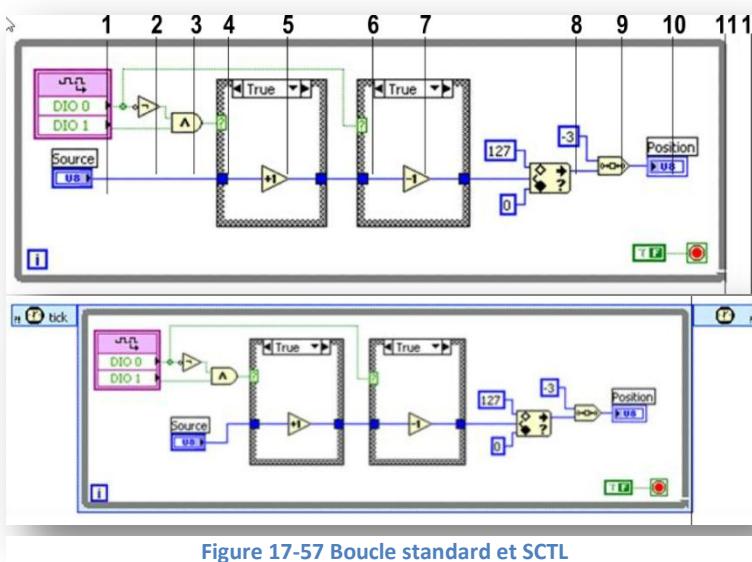


Figure 17-56 Un coup d'horloge au minimum par opération

La chaîne d'activation va activer élément par éléments les actions logiques effectuées par le FPGA, chaque « Enable In » est relié au « Enable Out » précédent. La chaîne d'activation garantie que les opérations sont réalisées sur le FPGA dans le même ordre que sur de diagramme LabView. Cet extra code implique que chaque VI nécessite au minimum un coup d'horloge pour s'exécuter.

Dans la structure SCTL, cette chaîne est désactivée, seuls les éléments « de la logique voulue » demeurent. Il faut bien entendu que le temps de propagation du signal à travers l'ensemble des portes soit inférieur à un coup d'horloge. Lors de la synthèse et du routage, le compilateur vérifie que c'est bien le cas. Une fenêtre d'avertissement apparaît dans le cas contraire.



Cette boucle demande 12 coups d'horloge pour être exécutée

Celle-ci, 1 coup d'horloge...

Figure 17-57 Boucle standard et SCTL

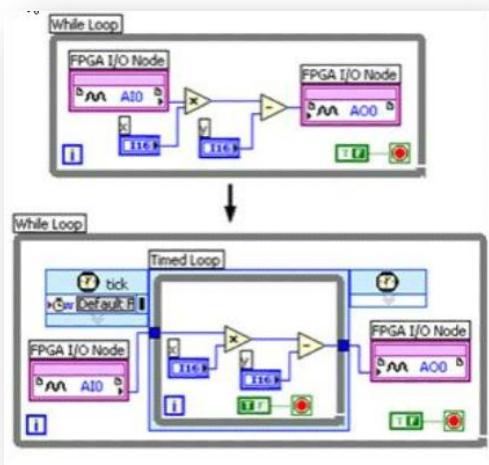


Figure 17-58 Optimisation d'une partie de code

Certaines limitations existent, notamment il n'est pas possible d'imbriquer des structures, d'exécuter des opérations d'E/S analogiques (elles consomment environ 35 coups d'horloge), d'utiliser certaines fonctions prenant plus d'un cycle (presque toutes les fonctions mathématiques à part l'addition, soustraction et multiplication) et bien évidemment les fonctions d'attente.

Il est possible d'utiliser les boucles SCTL pour optimiser certaine parties du code en laissant à l'extérieur les fonctions occupant plus d'un cycle.

Seules les SCTL permettent l'utilisation d'horloges dérivées

Pipeline

La technique du pipeline consiste en un découpage du code en morceaux élémentaires et à échanger les informations entre ces morceaux par des tunnels de registre à décalage. Il faut cependant faire attention à l'augmentation du temps de latence (temps avant que la première donnée valide soit disponible), car le nombre de tours de boucle par opération augmente avec la profondeur du pipeline (deux cycles d'horloge par tour de boucle).

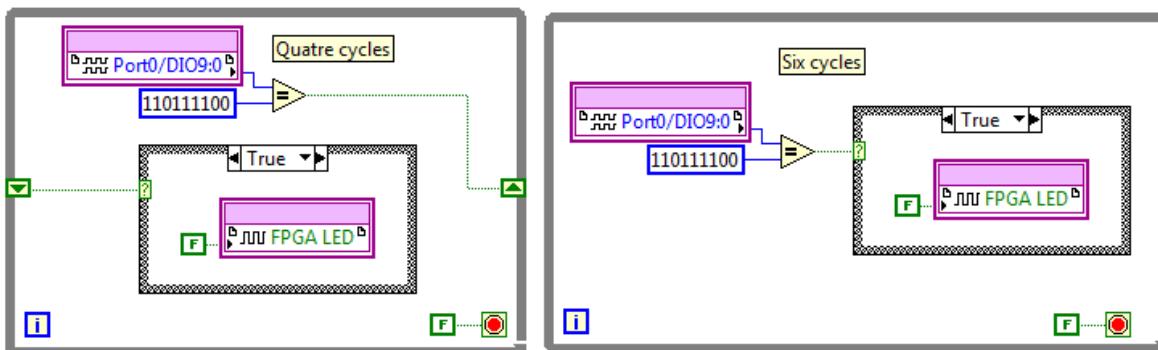


Figure 17-59 Séparation des opération par un Pipeline

La technique peut bien évidemment être utilisée avec une SCTL si le temps de propagation n'est trop long

DOMAINES TEMPORELS

Les FPGA peuvent générer des horloges dérivées de l'horloge principale à 40 MHz. La famille Spartan contient quatre « Digital Clock Manager (DCM) ». La synthèse de fréquence est obtenue par multiplication/division de la fréquence principale.

HORLOGES DERIVEES

CREATION

Les horloges dérivées sont créées dans le projet en déroulant le menu contextuel de l'horloge principale. Une fenêtre de configuration apparaît.

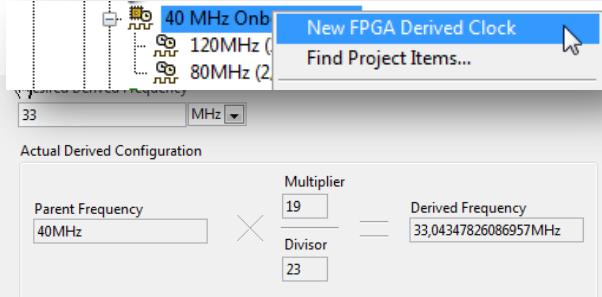
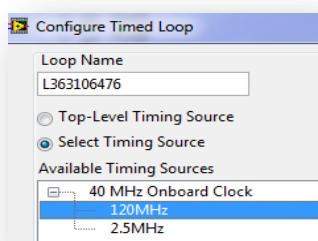


Figure 17-60 Création et configuration des horloges dérivées

L'utilisateur entre la fréquence désirée, les diviseurs et multiplicateurs optimaux sont calculés, et la fréquence réellement synthétisable est affichée. Sur la carte SB-RIO les fréquences disponibles s'échelonnent de 2.5 à 280 MHz

Bien que le FPGA le permette, il n'est pas possible de sélectionner la phase cadran par cadran.

UTILISATION



Les horloges dérivées sont accessibles dans les SCTL par un double clic sur l'horloge de cadencement. La fenêtre de dialogue propose toutes les horloges créées précédemment.

Figure 17-61 Sélection de l'horloge

VIOLATION TEMPORELLES

Chaque élément du diagramme nécessite un certain nombre de portes et donc un certain temps de propagation. Ce n'est que lors de la phase de synthèse qu'il sera possible de connaître si le code écrit est exécutable à la vitesse souhaitée.

Si nous reprenons notre tout premier code et que l'on fixe la fréquence d'exécution de la boucle à 250 MHz, un message d'erreur apparaît, indiquant qu'il y a une violation temporelle.

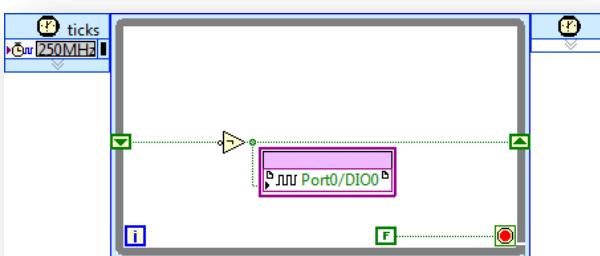


Figure 17-62 Le programme de test

Le bouton « Investigate Timing Violation » ouvre un rapport des différents temps, répartis entre temps de propagation dans les portes et temps de propagation dans les interconnections.

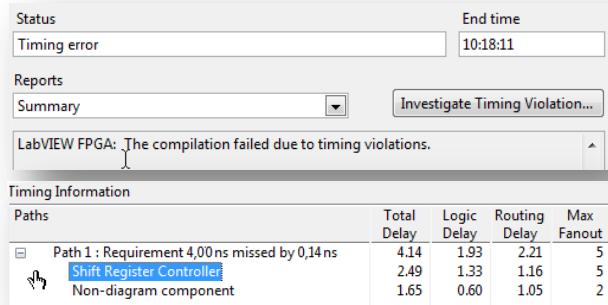


Figure 17-63 Violation temporelle.

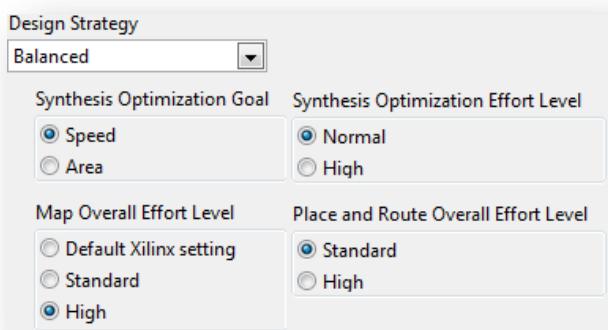


Figure 17-64 Optimisation du synthétiseur

Le temps total de propagation devrait être inférieur à 1/250MHz, soit 4 ns, hors, il est de 2.49 ns pour la boucle et de 1.65 ns pour l'extra-code. Vous constatez que le temps de propagation dans les lignes n'est pas négligeable, ce résultat peut dépendre de la configuration du synthétiseur et du positionnement des portes au départ (le synthétiseur ne donne pas toujours le même résultat).

Il est possible d'exiger du synthétiseur d'optimiser le code plutôt pour la vitesse ou pour l'encombrement, mais aussi d'ajuster le nombre de passes lors des phases de synthèse, placement routage. Ces options sont dans la page de propriété du « Build Specification » item « Xilinx Option ».

COMMUNICATION ENTRE DOMAINES TEMPORELS

La FIFO est l'unique moyen de transférer des données entre deux boucles de domaines temporels différents. Dans ce cas, l'opération d'écriture doit appartenir à un seul domaine temporel et l'opération de lecture à un seul autre. L'utilisation de variables locales ou globale reste possible, mais il faut implémenter un système de handshaking si l'on veut éviter la perte de données.

ANNEXES

CONNEXIONS DE LA CARTE PROTOTYPE

BOUTON POUSSOIR

Un seul est disponible

SW1 : Port6/DIO9, passe à un lors de l'appui

LED

Deux les sont disponibles

LED1 : Port6/DIO4

LED2 : Port6/DIO5

BNC

Trois prises BNC sont reliées directement aux entrées du FPGA

BNC1 : Port5/DIO6

BNC2 : Port5/DIO7

BNC3 : Port5/DIO8

AFFICHEURS

Quatre afficheurs 7 segments à anodes communes sont disponibles, ils sont multiplexés. Les segments sont contrôlés par :

A : Port6/DIO7

B : Port6/DIO8

C : Port2/DIO8

D : Port2/DIO7

E : Port2/DIO6

F : Port2/DIO4

G : Port2/DIO5

Les anodes

1000 : Port5/DIO0

100 : Port5/DIO1

10 : Port5/DIO2

1 : Port5/DIO3

ADC

Un ADC 12 bits **AD7896** avec une liaison série trois fils « genre SPI ». La référence est VDD, les 12 bits sont donc répartis entre GND et VDD

Les connections avec le FPGA sont les suivantes :

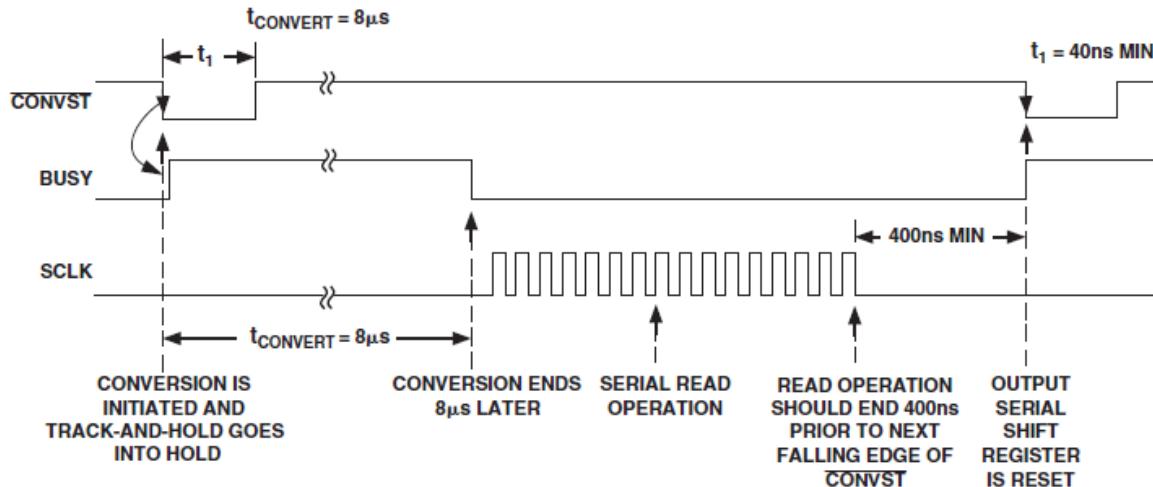
BUSY : Port6/DIO0

CONVST : Port6/DIO1

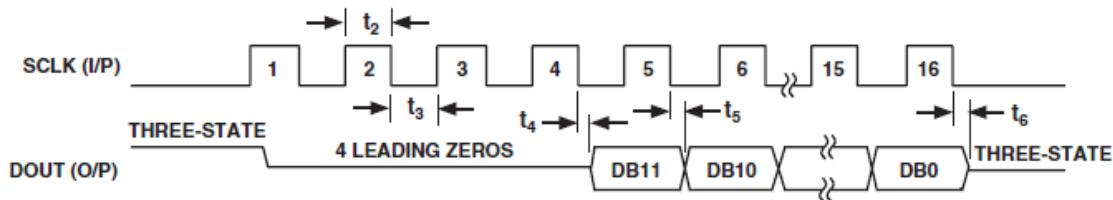
SDATA : Port6/DIO2

SLK : Port6/DIO3

Le fonctionnement de l'ADC est le suivant : La conversion est lancée par un front descendant sur l'entre CONVST. L'horloge de conversion est générée en interne le temps de conversion est 8 μ s. Après la conversion, La ligne BUSY passe à l'état bas, le cycle de lecture peut commencer, la donnée est sur 16 bits avec quatre bits à 0 en tête.



$$t_2 = t_3 = 40\text{ns MIN}, t_4 = 60\text{ns MAX}, t_5 = 10\text{ns MIN}, t_6 = 50\text{ns MAX @ 5V, A, B, VERSIONS}$$



TIMING CHARACTERISTICS¹ ($V_{DD} = 2.7 \text{ V to } 5.5 \text{ V}$, AGND = DGND = 0 V)

Parameter	A, B Versions	J Version	S Version	Unit	Test Conditions/Comments
t_1	40	40	40	ns min	$\overline{\text{CONVST}}$ Pulsewidth
t_2	40^2	40^2	45^2	ns min	SCLK High Pulsewidth
t_3	40^2	40^2	45^2	ns min	SCLK Low Pulsewidth
t_4	60^3	60^3	70^3	ns max	Data Access Time after Falling Edge of SCLK
	100^3	100^3	110^3	ns max	$V_{DD} = 5 \text{ V} \pm 10\%$
t_5	10	10	10	ns min	$V_{DD} = 2.7 \text{ V to } 3.6 \text{ V}$
t_6	50^4	50^4	50^4	ns max	Data Hold Time after Falling Edge of SCLK
					Bus Relinquish Time after Falling Edge of SCLK

DAC

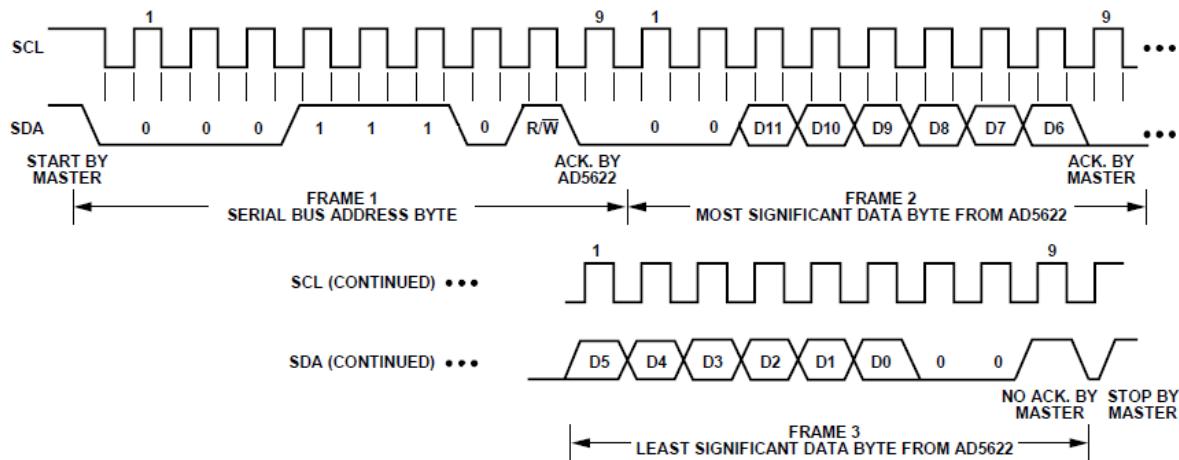
Un DAC 12 bits **AD5622** est relié à l'entrée de l'ADC par un cavalier à deux position (l'ADC reçoit soit $V_{CC}/2$, soit la sortie du DAC). Il a l'adresse : 2(partie variable) + 12(partie fixe)=14

Les connections avec le FFPGA sont les suivantes :

SDA : Port5/DIO4

SLC: Port5/DIO5

Le DAC n'a qu'un registre 16 bits qui contient la valeur binaire correspondant à la tension souhaitée en sortie.
La trame à envoyer est :



RAPPEL SUR L'I2C.

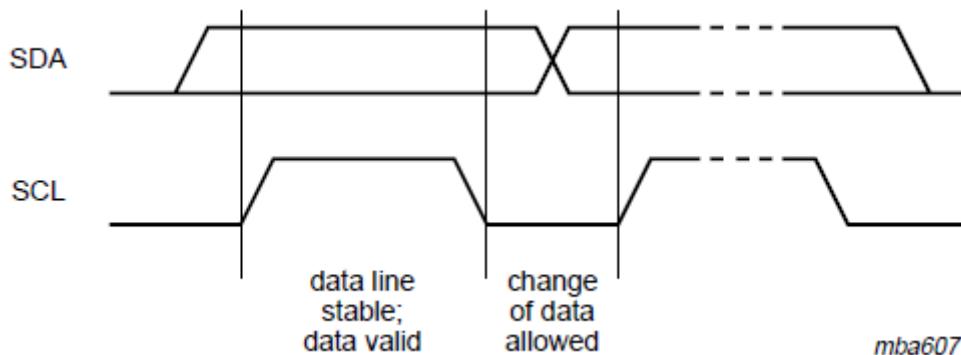
L'I2C est une liaison série deux fils (une horloge SCL émise par le maître et une ligne de donnée bidirectionnelle SDA) la fréquence standard de l'horloge est de 100 kHz. Le bus est multi-maitres et multi-esclaves.

LA TRAME

Une trame I2C contient : Une condition de START, l'adresse de l'esclave, un bit de direction, un bit d'acquittement, puis huit bits de data, un bit d'acquittement (cette séquence DATA/ACK peut être répétée à l'infini) et enfin une condition de stop.

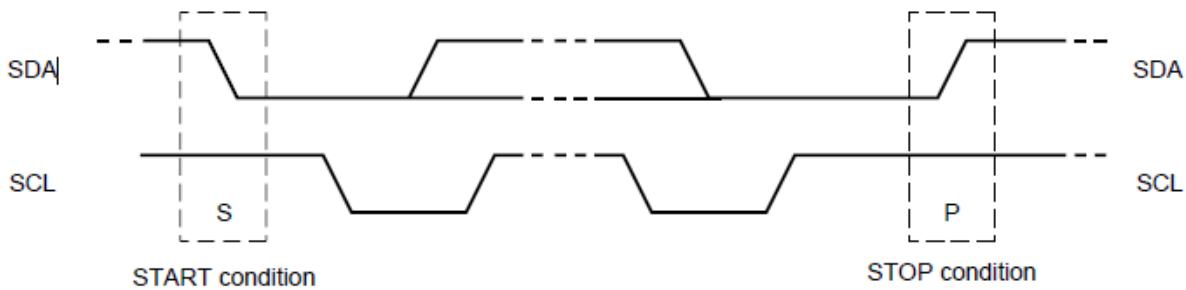
VALIDITE DES DONNEES

Les données sur la ligne SDA doivent être stables pendant la période haute de l'horloge. L'état de la ligne de données peut changer seulement lorsque le signal d'horloge sur la ligne SCL est bas. Une impulsion de l'horloge est générée pour chaque bit de données transférée.



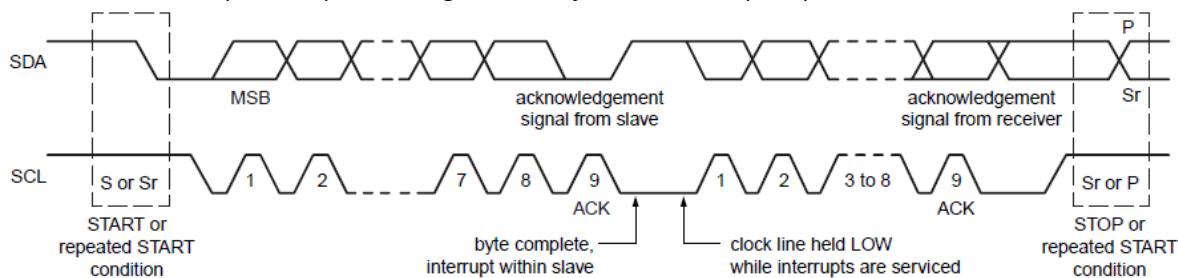
PRISE DU BUS ET RELACHEMENT

Toutes les transactions commencent avec un START (S) et peuvent être résiliées par un STOP (P). Une transition de HAUT vers BAS sur la ligne SDA alors que SCL est HAUTE définit une condition de START. Une transition de BAS vers HAUT sur la ligne SDA alors que SCL est HAUTE définit une condition STOP.



FORMAT DE LA TRANSMISSION

Chaque octet mis sur la ligne SDA doit être de 8 bits. Le nombre d'octets qui peuvent être transmis est illimité. Chaque octet doit être suivi d'un bit d'acquittement. Données sont transférées avec le MSB en premier. Si un esclave ne peut recevoir ou transmettre un autre octet complète de données jusqu'à ce qu'il ait effectué une autre fonction, par exemple une interruption interne, il peut maintenir la ligne SCL BAS pour forcer le maître dans un état d'attente. Transfert de données continue ensuite lorsque l'esclave est prêt pour un autre octet de données. Si l'esclave peut bloquer l'horloge, c'est toujours le maître qui la pilote.



ACQUITTEMENT

L'acquittement a lieu après chaque octet. Le bit d'acquittement permet au récepteur signaler à l'émetteur que l'octet a été reçu avec succès et un autre octet peut être envoyé. Toutes les impulsions d'horloge y compris la 9e sont générées par le maître.

Le signal d'acquittement est défini comme suit :

- 1) l'émetteur libère la ligne SDA (il met sa ligne en entrée) durant l'impulsion d'horloge d'acquittement donc le récepteur peut commander la ligne SDA
- 2) SI SDA est BAS durant la période haute de l'horloge, il y a acquittement de la part du récepteur. Si SDA est HAUTE alors il y a eu un problème de reception.

Le maître peut alors générer une condition STOP pour abandonner le transfert ou une condition START pour démarrer un nouveau transfert.

ADRESSE D'ESCLAVE ET BIT R/W

Les transferts de données suivent le format illustré. Après le START (S), une adresse esclave est envoyée. Cette adresse est de 7 bits suivies d'un huitième bit qui est la direction des données (R/W) — un « zéro » indique une transmission (écriture), une « un » indique une requête (lecture). Un transfert de données est toujours terminé par une condition STOP (P) générée par le maître. Toutefois, si un maître souhaite toujours communiquer sur le bus, il peut re-générer une condition de START (Sr) et l'adresse d'un autre esclave sans générer la condition STOP.

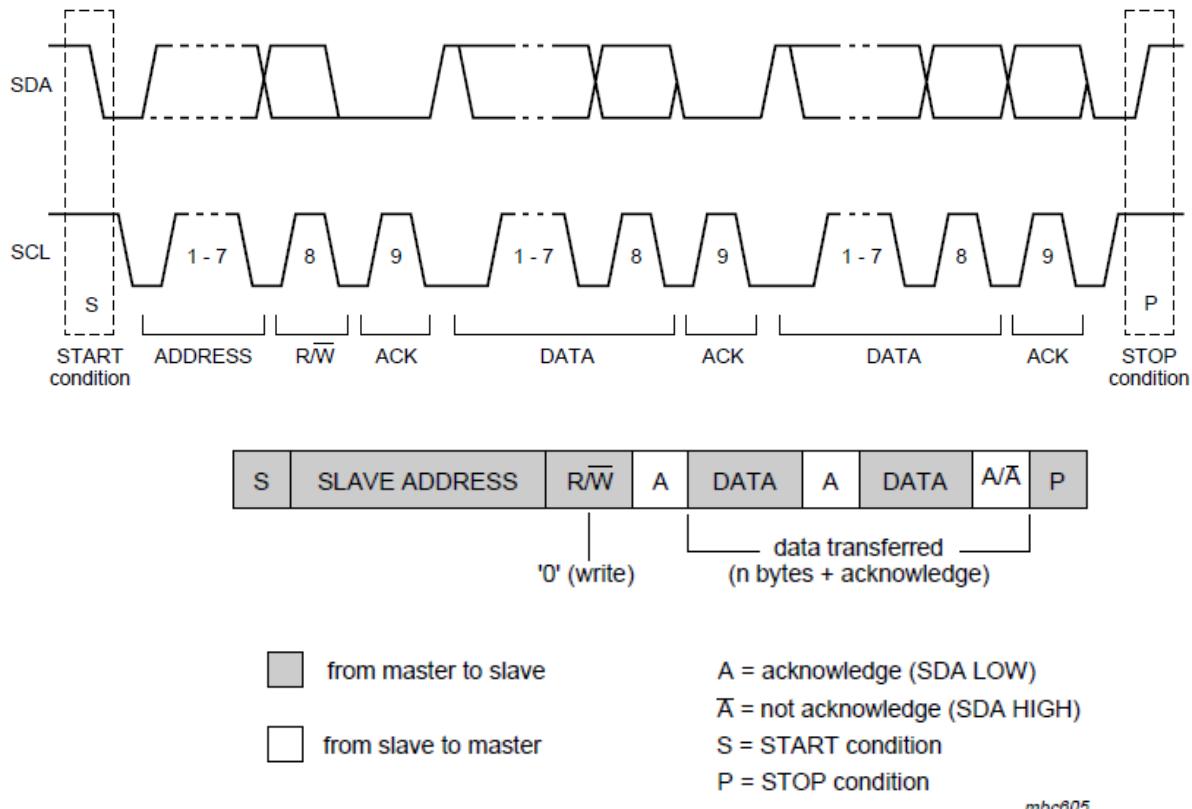


Fig 11. A master-transmitter addressing a slave receiver with a 7-bit address (the transfer direction is not changed)

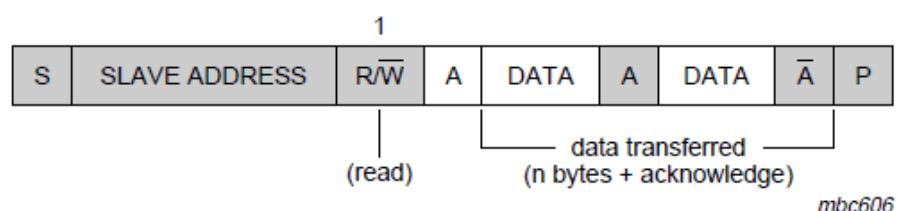


Fig 12. A master reads a slave immediately after the first byte

Biblio : FPGA module training : <http://zone.ni.com/devzone/cda/tut/p/id/3555>

I2C NXP User Manual UM10204 : http://www.nxp.com/documents/user_manual/UM10204.pdf