SOPROLEC
www.soprolec.com
ZAC DE L'EPINE
72460 SAVIGNE L'EVEQUE
Tél. : +33 (0)2 43 76 44 76

Modbus ICNC Labview

Sommaire

Introduction

L'utilisation des cartes InterpCNC par USB et par liaison série est très différente.

Pour la programmation par USB une librairie a été créée, pour la programmation par liaison série, les cartes disposent d'un interpréteur de commande via écriture de registres dans la mémoire.
Ceci signifie que plusieurs registres des cartes sont réservés à l'écriture et l'envoi de commandes et sont directement interprétés par la carte.

Le premier de ces registres sert à définir le numéro de commande à utiliser. Ce numéro permet à la carte de reconnaître la commande et de traiter le contenu des registres suivant comme les paramètres de cette commande.

Il sera donc important de suivre comment sont formées ces commandes et comment s'en servir.

Pour communiquer via liaison série Modbus nous utilisons la librairie Modbus de National Instruments. Il vous faudra aussi installer VISA pour Labview sur votre pc pour avoir accès dans Labview à vos ports séries.

Installation librairie Modbus National Instruments

Vous pourrez trouver avec cette documentation le dossier nimodbus121 qui contient la librairie Modbus de National Instrument.

Les différents sous-dossiers de ce dossier contiennent la librairies Modbus. Ils sont nommés suivant les mise à jour de Labview. Si aucun ne correspond à votre version de Labview , prenez le dernier par défaut : le 86.

Ce dossier doit être copié dans :

C:/ Program Files(x86)/National Instruments/Labview 2012/user.lib/
(Vous devriez avoir un National Instruments dans vos programmes et dans celui-ci votre répertoire Labview , le dossier user.lib contient les différentes librairies utilisateur que vous avez)

Une fois copié, (re)lancez Labview et vous devriez avoir dans votre palette librairies utilisateur le dossier 86 contenant les différentes fonctions(VIs) de la librairie Modbus.

Installation VISA

VISA de national instrument est un logiciel qui permet de donner accès à Labview au différents composant de votre machine. Il permet notamment de donner accès à Labview à vos ports séries, ce qui vous sera nécessaire pour la création d'application série.

Vous devriez trouver à cette adresse : https://www.ni.com/visa/ un lien vers download qui vous permettra de télécharger VISA. Sélectionnez celui qui correspond à votre système d'exploitation.

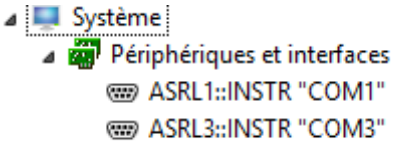
Suivez ensuite les instructions de l'installeur VISA.

Dans Labview , dans l'onglet outils, vous trouvez Measurement & Automation Explorer.

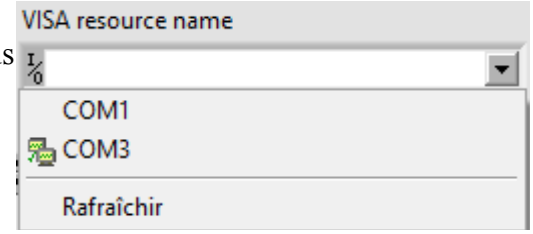
Cette fenêtre vous indique les logiciels et les périphériques et interfaces dont Labview a accès.

Avant l'installation de VISA, le champ périphérique et interfaces ne devait pas êtres visible.

Après l'installation de VISA, ce champ devrait apparaître, et dans celui-ci, vous devriez voir les différents ports COM de votre machine.



Si vous n'avez pas accès à vos ports COM, vous devrez réinstaller VISA.



Utilisation Modbus ICNC

Plusieurs dossiers sont à votre disposition avec ce document. Vous trouverez un projet Commande qui vous présentera les différentes possibilités disponibles pour former des commandes.

Il y a aussi un projet Application Modbus qui est un exemple concret d'application utilisant la communication Modbus.

Vous y trouverez aussi une librairie Modbus qui peut répondre à vos attentes, vous évitant de faire vos propres commandes.

Il y a la librairie Modbus de National Instrument que nous avons modifiée pour l'adapter à notre matériel. Vous trouverez l'historique de ces modifications dans ce document.

Les différents dossiers énoncés ci-dessus sont tous expliqués et détaillés dans ce document.

La librairie ICNC Modbus peut répondre à vos attentes. Dans ce cas, en utilisant quelques fonctions simples de la librairie Modbus de National Instrument, vous constituerez rapidement votre application Modbus. Si la librairie ICNC Modbus ne répond pas à vos besoins, vous pouvez vous inspirer du projet Commandes pour comprendre la création et l'utilisation des commandes Modbus et vous inspirer du projet Application Modbus pour réaliser vos propres VIs et votre application.

Nous vous conseillons, dans les 2 cas, d'utiliser la librairie Modbus de National Instrument que nous avons modifiée pour réaliser vos applications. Vous trouverez le détail de ses modifications page 20.

Pilotage par Commandes

Vous trouverez un pdf Protocole MODBUS InterpCNC V2.1 avec cette documentation.

Dans ce pdf, vous trouverez la liste des différentes registres accessibles de la carte et aussi les différentes commandes existantes.

Tout ces registres sont au format 16 Bits.

Modbus créé un décalage sur l'accès aux registres, donc sur les adresses, vous devez reculer d'une adresse par rapport aux adresses communiquées dans le pdf Modbus InterpCNC.

Informations générales sur le lancement de commandes

Le contrôle des cartes se fait par l'envoi de commandes à l'adresse 4095 (non pas 4096, voir paragraphe ci-dessus en rouge).

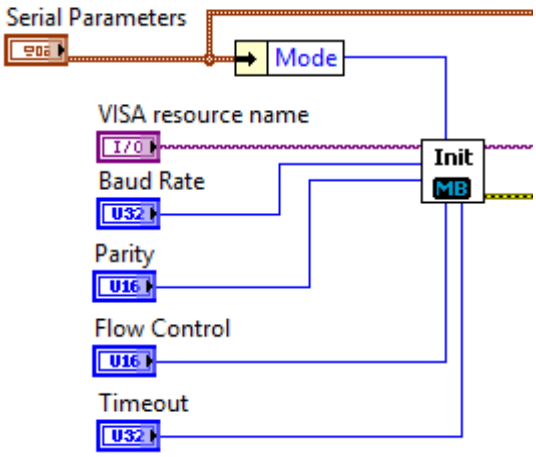
Les arguments suivant l'adresse 4095 sont les paramètres de la commande.

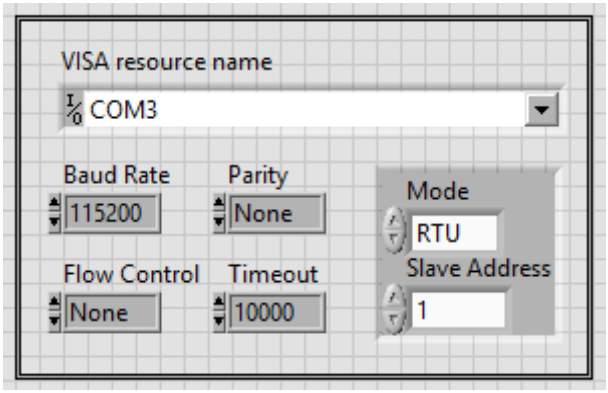
Chaque écriture du registre 4095 est considéré comme l'envoi d'une nouvelle commande.

Si vous ne pouvez pas écrire plusieurs registres à la fois, écrivez dans un premier temps les arguments de la commande, puis écrivez le registre avec le numéro de commande en dernier pour envoyer la commande avec ses arguments.

Exemples Commandes Modbus

Avant tout, la communication Modbus doit être initialisée. Pour ce faire, la librairie Modbus contient un VI Init qui permet de définir les paramètres de la liaison.





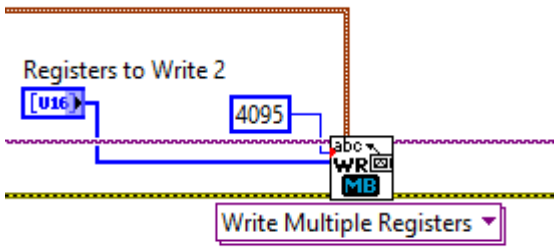
Dans notre exemple, il est nécessaire de pré-remplir ces informations avant le lancement de l'application, car notre initialisation de communication est réalisée au lancement de l'application.

Pour envoyer une commande, la fonction Write Multiple Registers de la librairie Modbus est la plus adaptée.

Il faut renseigner le port et les paramètres de la liaison pour chaque fonction Modbus.

Il faut aussi renseigner l'adresse à laquelle écrire la commande, c'est-à-dire 4095 comme signifié précédemment.

Il faut aussi renseigner le tableaux des valeurs à écrire dans les registres.



Utilisation Documentation « Protocole Modbus InterpCNC V2.1 »

La liste et description des commandes Modbus sont répertoriées dans le pdf « Protocole Modbus InterpCNC V2.1 ».

Dans ce pdf vous trouverez comment utiliser et appeler chaque fonction.

Par exemple :

Commande 42 : Arrêt avec rampe des axes (ICNC_CMD_BREAKE_AXES_AND_CLEAR)

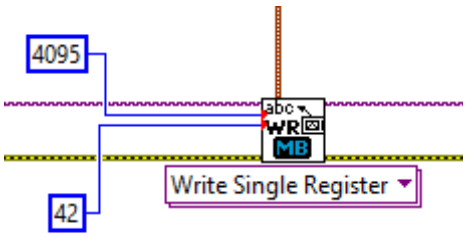
La commande N°42 provoque un arrêt de tous les axes en cours de mouvement.
L'arrêt se fait avec une rampe de décélération.

Adresse	Valeur
4096	Commande = 42

La commande 42 pour l'arrêt des axes avec rampe.

Cette fonction arrête le déplacement sur tout les axes (avec rampe , donc décélération).

Cette commande est composée d' un seul registre à écrire, la fonction Write Single Register de la librairie Modbus suffit.



Donc un appui sur un bouton déclenchant l'appel ci-dessus suffira à arrêter les déplacements sur tout les axes, 42 étant le numéro de commande (le contenu du registre à écrire) et 4095 l'adresse du registre a écrire.

Pour les commandes nécessitant l'écriture de plusieurs registres, l'utilisation de la fonction Write Multiple Registers est plus adaptée. En effet, celle-ci permet l'écriture d'un tableau fourni à la fonction et d'une adresse de départ correspondant au premier registre à écrire pour la 1ère case du tableau et ensuite les autres cases sont écrites dans les registres suivants.

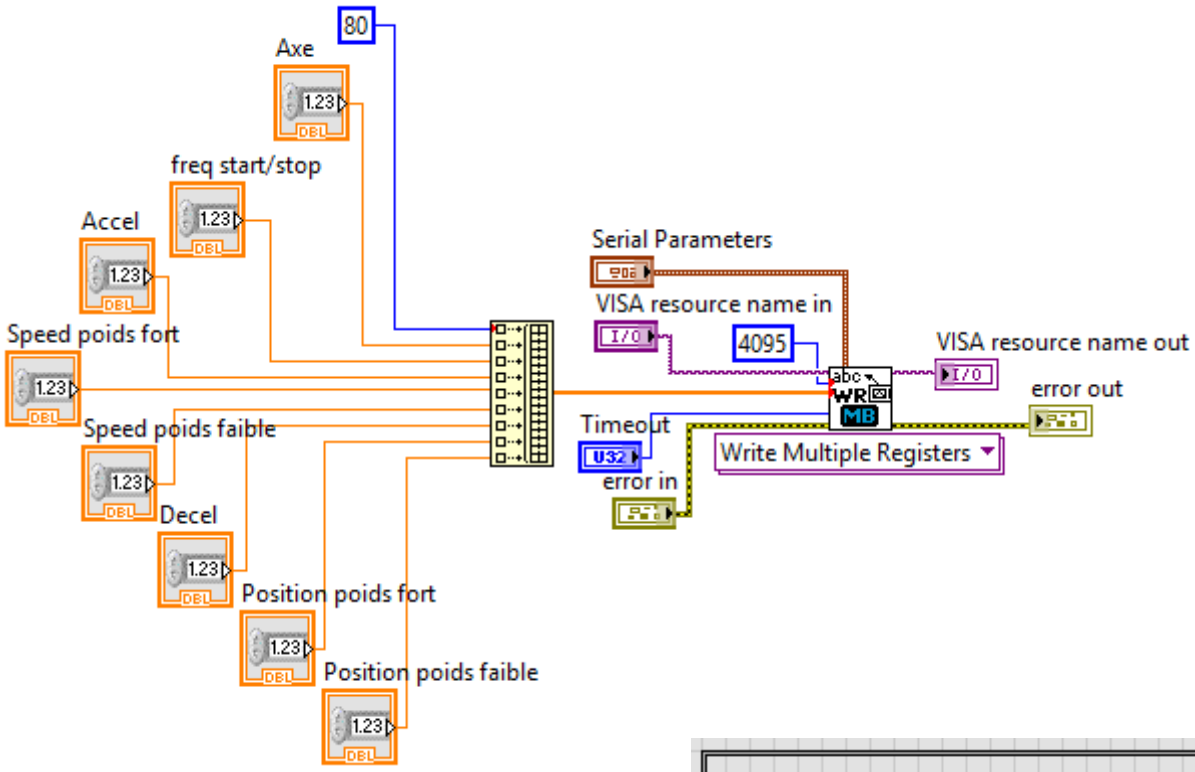
Ce procédé permet d'écrire un tableau contenant une commande complète par exemple et de lancer celle-ci, ce qui s'avère bien plus efficace que de remplir chaque registre de votre commande un par un.

VI Déplacement détaillé

Une des parties les plus importantes sera la gestion et la manipulation des tableaux qui vous permettront de former des commandes Modbus et de piloter la carte.

Un VI de déplacement détaillé a été créé pour illustré ceci.

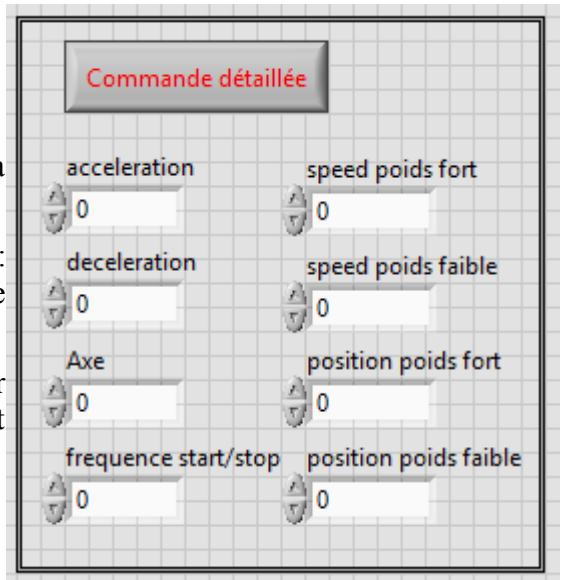
Ce VI passe en paramètres les différents éléments d'une commande 80 et les assemble en un seul tableau qui est passé en paramètre à la fonction Write Multiple Registers.



Nous avons, dans notre interface, les différents champs à remplir pour le lancement de cette commande.

Le bouton est associé à un événement qui lance le VI : commande détaillée où les différents champs ci-contre sont câblés.

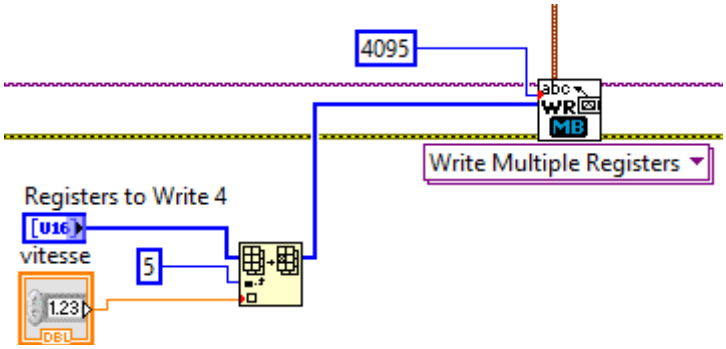
Ce type de présentation est plus explicite pour l'utilisateur que le remplissage manuelle d'un tableau affiché où il est difficile de préciser à quoi correspond chaque case.



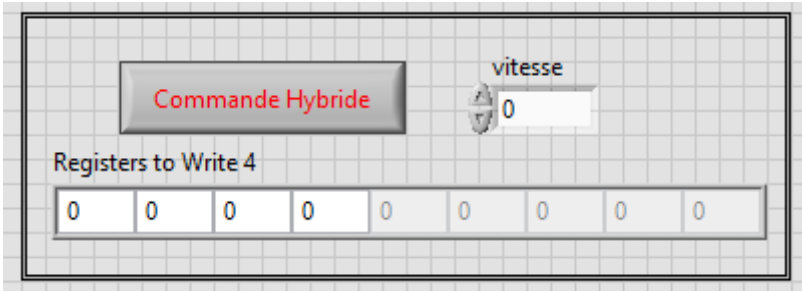
Manipulation de Tableaux

La Manipulation de tableaux est importante pour la formation de commandes ou de commandes partielles.

Par exemple :



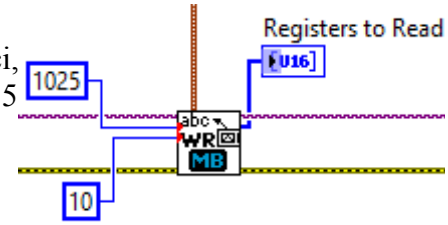
Ici nous lançons une commande par un tableau. Cependant ce tableau est modifié, sa 5ème case est remplacé par la valeur contenue dans le champ vitesse.



Il est possible de prédéfinir des commandes avec des tableaux pré-remplis et de remplacer certaines valeurs par celle présentes dans des champs avant l'envoi de la commande.

Attention cependant à ce type de tableaux, car, par défaut, les tableaux sont vides au (re)lancement de l'application ; il faudra donc les initialiser.

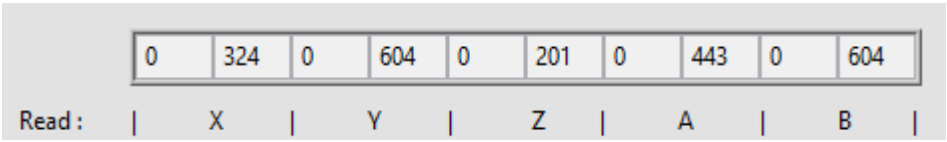
Un affichage de la position des axes en temps réel est possible avec la commande MB Serial Master Query Read.



Cette fonction permet de lire un nombre de registres défini . Ici, 10 sont lus, correspondant aux positions poids fort et faible des 5 axes.

Ceux-ci sont retournés dans un tableau.

On a un affichage de type poids fort/poids faible pour chaque axe.



Manipulation de Registres avec des Masques

Le pilotage de cartes via Modbus est intéressant car il permet aussi l'utilisation de TestCenter en parallèle via une connexion USB simultanée. Il est donc facile de tester les différentes commandes et manipulation effectuées par le programme sur Labview.

Déverrouillage de la carte

Cependant, la liaison USB étant limitée et pas toujours nécessaire, il intéressant de pouvoir s'en passer.

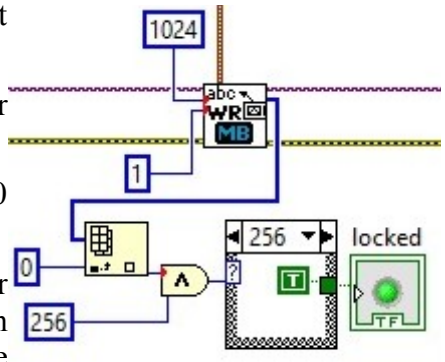
Pour ce faire, un des principal problème à pallier est le verrouillage de la carte. En effet, les cartes sont verrouillées à leur démarrage. Pour pouvoir se défaire de la liaison USB il faut être en mesure de savoir quand la carte est verrouillée et de pouvoir la déverrouillé via Modbus.

Il existe un bit dans le registre de statuts qui sauvegarde l'état de la carte : verrouillé ou déverrouillé.

Il faut faire un masque sur la lecture de ce registre pour savoir si la carte est verrouillé ou déverrouillé.

Pour faire ce masque on extrait d'abord le registre à l'index 0 du tableau qui est renvoyé par la fonction.

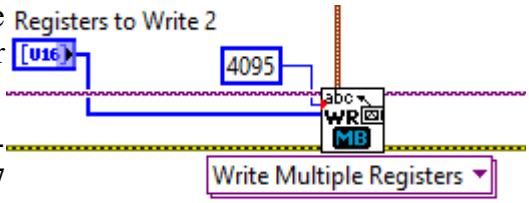
Ensuite, on réalise un ET logique avec une valeur correspondant au bit qui nous intéresse. Nous avons donc en sortie du ET logique : 256 ou 0 , 256 correspondant à la carte verrouillée et 0 correspondant à la carte déverrouillée.



Grâce à des masques comme celui-ci, il est possible de récupérer l'état de chaque bit dans ces registres de statuts. Vous trouverez le détail de ceux-ci dans le pdf « Protocole Modbus InterpCNC V2.1 ».

Maintenant que nous avons une LED qui nous affiche si la carte est verrouillée ou déverrouillée, il ne nous reste plus qu'à créer un bouton qui déverrouille la carte.

Pour ce faire, nous utiliserons la commande 66 de Reset et Ré-Armement, avec un tableau contenant la valeur 66 et 7 correspondant au 3 bits de ré-armement.

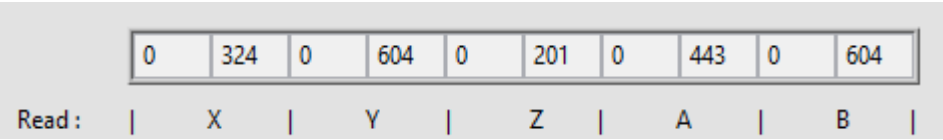


Nous avons donc un résultat de ce type dans l'interface :

N'ayant plus besoin de TestCenter pour déverrouiller la carte, la liaison Modbus est maintenant indépendante.

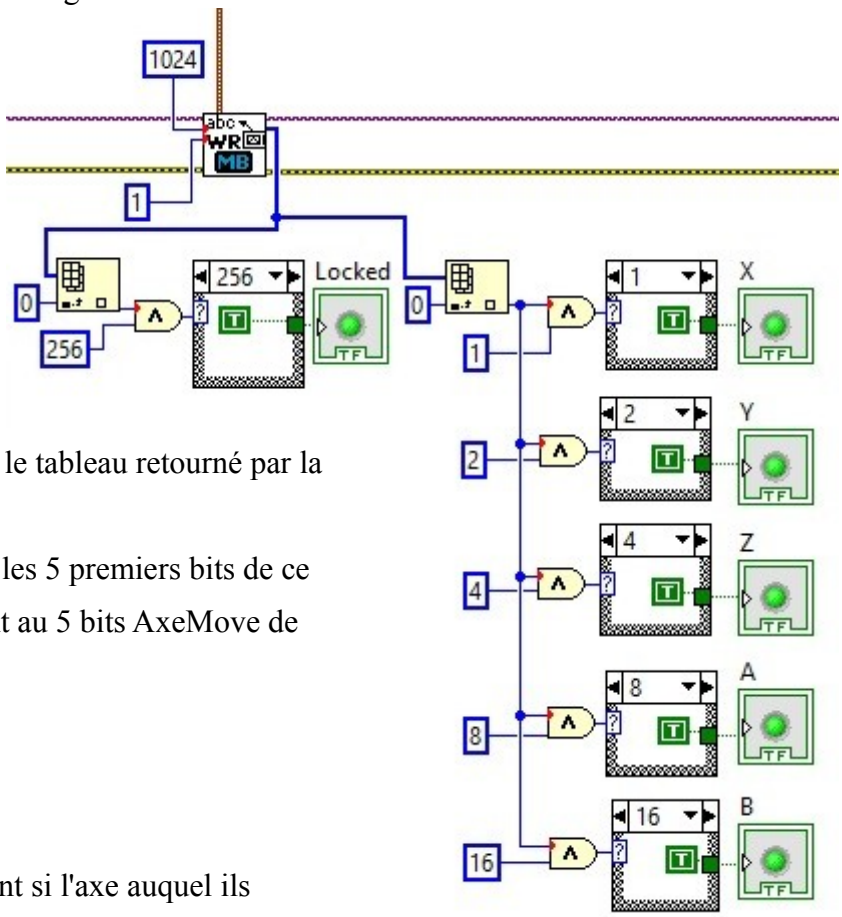
L'utilisation des masques vous sera très utile avec les registres de statuts.

Un autre exemple :



Nous avons la position de chaque axe. Maintenant, avec les masques et le registre de statuts, nous allons afficher une LED par axe qui signalera si l'axe est en mouvement ou non.

Pour ce faire, il faut refaire un traitement proche de celui effectué pour le bit de verrouillage pour chaque bit AxeMove du registre de statuts.

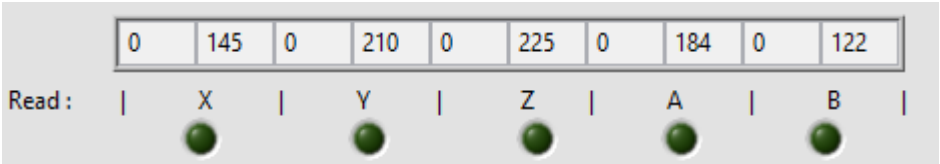


Nous reprenons donc le tableau retourné par la fonction de lecture.

Ensuite, nous testons les 5 premiers bits de ce registre correspondant au 5 bits AxeMove de la carte.

Ceux-ci nous indiquent si l'axe auquel ils correspondent est en mouvement ou non.

Nous avons un résultat toujours plus explicite à propos des axes, de leur position et de quand ils sont en mouvements.



Il reste néanmoins un inconvénient à ce format d'affichage de la position : la répartition en 2 registres. Le fait que le valeur de la position soit supérieur à 65536 ne nous permet pas de le stocker dans un seul registre 16 bits. La position est donc définie sur 2 registres , avec un registre de poids fort et un registre de poids faible.

Pour palier à ce problème, nous allons assembler la valeur de ces 2 registres en une seule valeur pour en faciliter la lecture et la compréhension.

Nous allons réaliser un masque pour le registre de poids faible et un décalage logique pour le registre de poids fort.

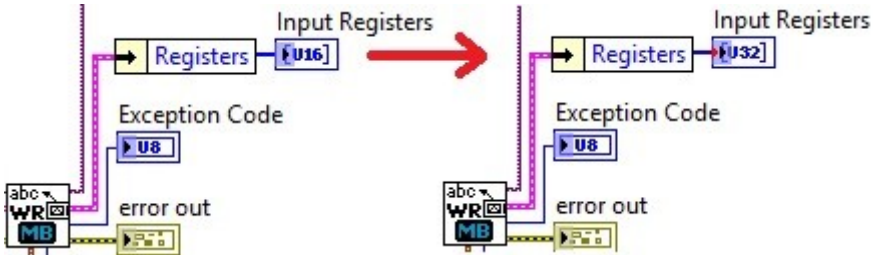
Le décalage nous permet de définir la valeur stockée dans le registre comme valeur commençant au 17eme bits d'une valeur 32 Bits, on a donc 2 registres 16 bits assemblés dans un champ 32 Bits.

Par exemple : pour une valeur de 1 dans le registre de poids fort c'est la valeur 65536 qui est ajoutée à la valeur du poids faible pour obtenir le résultat.

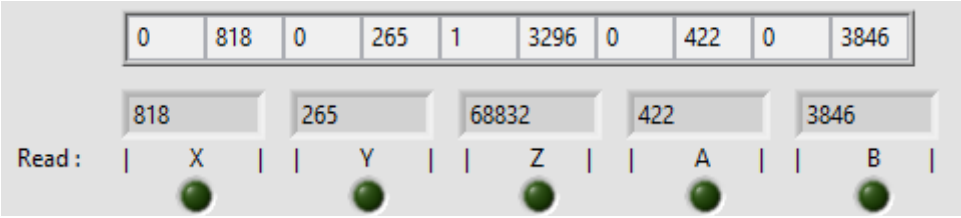
Cependant un problème se pose, lorsqu'on applique le décalage de 16 bits à la valeur rentrée, nous avons en sortie un 0, peu importe la valeur rentrée.

Ce problème est causé par la fonction de la librairie Modbus que nous utilisons (MB Serial Master Query Read). En effet, celle-ci retourne les contenus des registres dans des valeurs 16 bits, avec notre décalage nous sortons des valeurs qu'elle enregistre.

Pour résoudre ce problème, il va nous falloir modifier le format de valeur que la fonction retourne. Nous avons affiché le contenu du VI et modifié la représentation du tableau retourné de 16 bits en 32 bits.



Grâce à cette modification, notre décalage est bien pris en compte et nous affichons maintenant la positon des axes dans une seule valeur 32 bits dans notre interface. Nous avons laissé le tableau représentant les valeurs brutes des registres et en dessous les valeurs « assemblées » de ces registres formant les valeurs exactes de la position des Axes. Cette affichage facilite grandement leur lecture et compréhension.



VI Déplacement Simplifié

Toujours dans une optique de simplification, un VI Déplacement Simplifié a été créé.

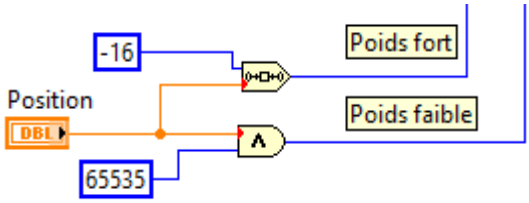
Ce VI permet de récupérer des paramètres définis dans une variable globale, donc précédemment remplis (dans une interface paramètre par exemple). Il utilise ces paramètres et demande à l'utilisateur d'indiquer l'axe à déplacer, la vitesse à laquelle le déplacer et la position à laquelle il doit être déplacé.

La partie particulièrement intéressante est la répartition de la vitesse et de la position en chacun des 2 registres différents. En effet, ces 2 paramètres sont à fournir à la fonction en 2 valeurs différentes, poids fort et poids faible, à la fonction d'écriture de registres.

Ce format n'est pas toujours simple d'utilisation notamment lorsque de grandes valeurs sont utilisées en vitesse et position.

Pour contourner ce problème, le VI réparti la valeur de vitesse et de position que l'utilisateur fournit en 2 valeurs de poids fort et poids faible.

Pour ce faire, il a fallu appliquer un décalage de 16 bits pour avoir la valeur de poids fort et réaliser un masque pour avoir la valeur de poids faible.



Grâce à ces opérations, la commande devient bien plus simple d'utilisation. Plusieurs commandes requièrent le passage de valeurs réparties en plusieurs registres, vous pouvez être amené à répéter cette opération pour d'autres commandes.

VI Position Simplifiée

La commande 67 permet de définir la position des axes.

Cette commande demande le(s) axe(s) dont la position doit être modifiée et la position à leur attribuer. Cependant, comme vu précédemment, les positions à fournir ne pouvant être stockées dans un registre 16bits. Ces positions doivent être passées via 2 valeurs 16bits, le poids fort et le poids faible de la position.

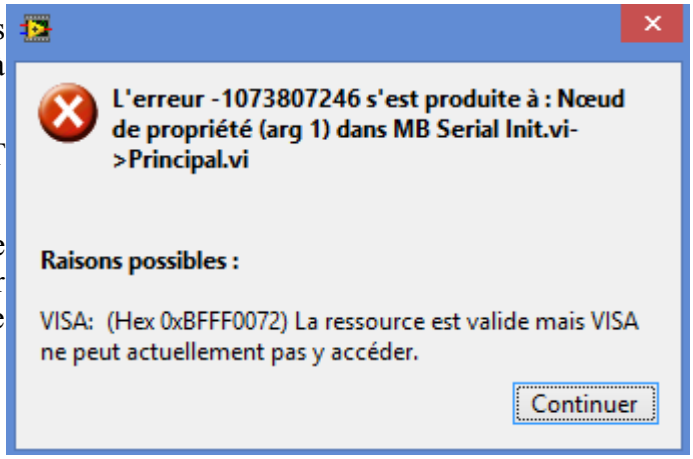
Pour simplifier l'utilisation de cette commande, le même mécanisme que celui appliqué dans le VI Déplacement simplifié a été employé. On a la valeur de position de chaque Axe qui est répartie en 2 valeurs 16bits et assemblées dans un tableau qui sert de registres à écrire pour une fonction Write Multiple Registers.

Ce VI permet donc de ne passer que l'axe et les valeurs (32bits) des positions sans avoir à les décomposer en 2 valeur 16bits.

Gestion des Erreurs

Il est nécessaire pour chaque fonction Modbus de fournir le Visa ressource name (port) et aussi les paramètres série. Il est aussi recommandé de fournir une entrée d'erreur afin que de l'initialisation de la communication jusqu'à sa fermeture toutes les fonctions transmettent ce flux d'erreur qui vous permettra d'avoir un résumé détaillé du problème.

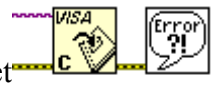
Par exemple ici, en choisissant le mauvais port COM, on reçoit ce message à l'exécution de l'application.



Ce message fait référence à l'arg1 de INIT correspondant au port choisi.

L'utilisation de ce système d'erreur s'avère très efficace et explicite et peut vous aider grandement surtout en cas de problème de communication série.

Voici comment devrait se terminer chaque application Modbus :



Avec la fermeture (Close) de VISA permettant de libérer le port et l'affichage des erreurs comme décrit ci-dessus.

Projet Commandes

Un projet complet regroupant les différents exemples vous ayant été exposés jusque là est disponible.

Tous les boutons de l'interface sont liés à un événement dans la structure événement.

Les fonctions de lecture devant être appelées régulièrement sont dans l'événement Timeout.

Les autres parties du programme ont été décrites dans les exemples de ce document.

Grâce à ces différents exemples vous devriez maintenant être en mesure de faire des commandes et de les exploiter.

Exemple Application Modbus

Comme pour la librairie ICNC USB, une application servant d'exemple a été réalisée.

Cette application est la même que celle réalisée pour la librairie ICNC USB.

C'est une interface de pilotage pour une tournette universelle. Cette tournette doit être pilotée en angle et vitesse, disposer de commandes manuelles et être adaptable à toutes tailles de tournette.

Nous avons réalisé cette application en utilisant les différents exemples exposés dans le projet Exemple Commandes Modbus.

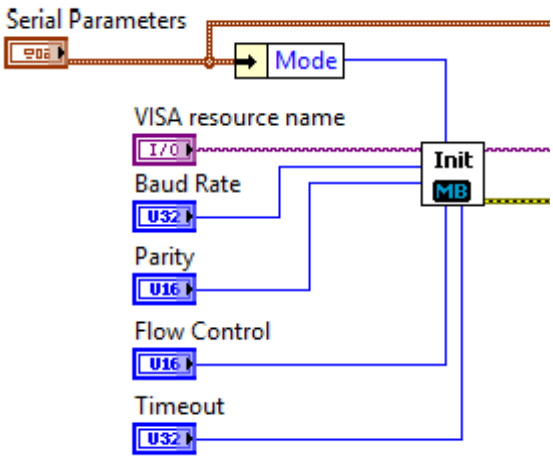
Nous avons dans notre interface principale :

- Des commandes manuelles (+ et -).
- Une partie exploitation qui permet de lancer plusieurs déplacements à la suite.
- Une partie test déplacement pour effectuer des essais.
- Une partie Modbus qui permet de configurer la communication.

Il y a un bouton PARAM qui permet d'accéder aux différents paramètres système et utilisateur relatifs à l'application. Un bouton UNLOCK existe aussi pour déverrouiller la carte. Un bouton RESET POS 0 pour remettre à 0 la position actuelle de l'axe de la carte. Un bouton STOP MOTEURS est aussi disponible, principalement dans le but de servir d'arrêt d'urgence en cas de problème.

Diagramme de l'interface principale

Nous devons, dans un premier temps, initialiser la communication Modbus avec ses paramètres.



Cette fonction Init retourne un nom de ressource VISA, autrement dit , le port COM que nous utilisons et aussi le flux d'erreurs qui pourrons se produire sur la communication.

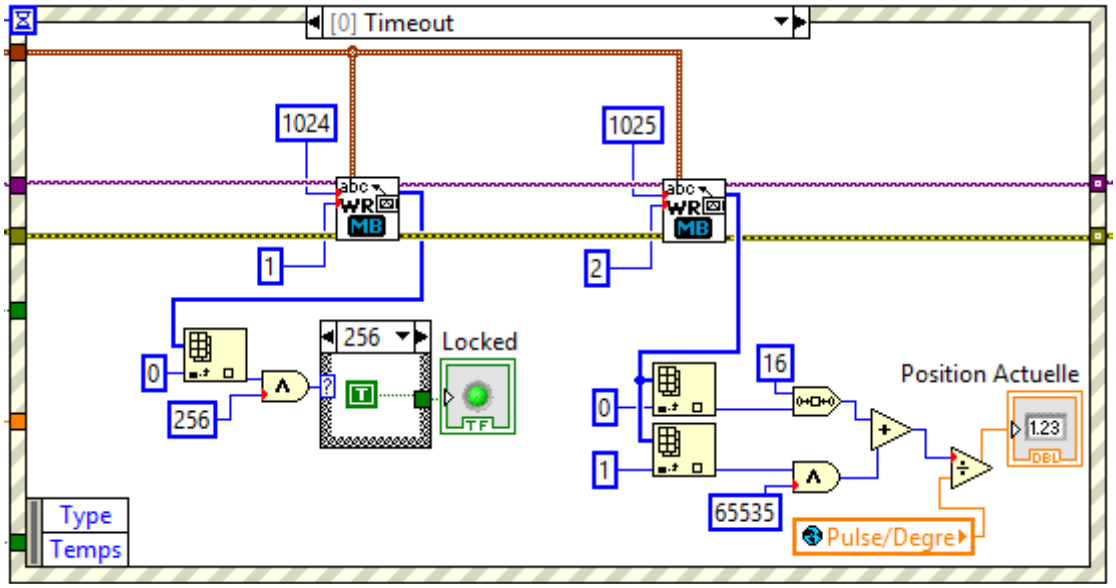
Plusieurs informations sont à fournir à la fonction Init. Il faut surtout préciser le port COM qui sera utilisé, les paramètres série avec l'adresse esclave de la carte, et surtout la vitesse de transmission qui doit correspondre à celle définie dans les paramètres de la carte, sinon il y aura des problèmes avec la communication.

Pour gérer les différents boutons présents sur l'interface, la solution la plus adaptée est une structure événement avec un événement associé à chaque bouton de l'interface.

Cette structure est contenue dans une boucle while où le bouton stop sert à arrêter l'application.

Cette structure est précédée de la partie Initialisation vue précédemment et elle est suivie de la partie fermeture visa expliquée plus loin dans ce document.

Nous utilisons le Timeout de cette structure pour permettre une lecture continue des différents paramètres qui nous intéressent, ici, ce sont le verrouillage de la carte et la position de l'axe.



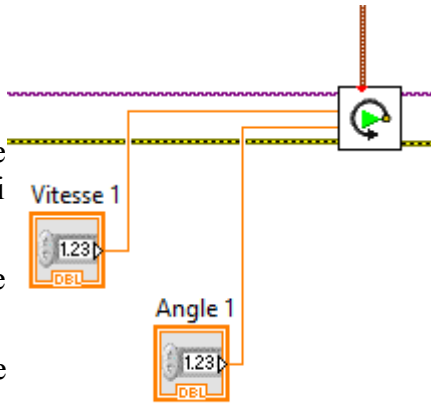
Un bit d'un registre de statuts de la carte nous indique si la carte est verrouillée ou non. Pour vérifier ce bit nous récupérons le contenu du registre qui nous intéresse avec la fonction MB Serial Master Read Query. Ensuite nous réalisons un masque pour récupérer la valeur du bit et affichons l'état de celui-ci via une LED.

Une seconde lecture est réalisée, cette fois dans le but de connaître la position actuelle de l'axe sur lequel le moteur est connecté. Cette valeur est répartie en 2 registres (poids fort et poids faible) nous réalisons un masque et un décalage pour remettre en forme ceux-ci en une seule valeur et nous l'affichons. Cette valeur est celle de la position sur l'axe où est le moteur permettant la rotation de la tournette. Cette valeur n'est pas intéressante car la position sur cet axe ne nous indique rien de compréhensible par rapport à la tournette. Pour rendre cette valeur digne d'intérêt nous la divisons par le nombre de pulse/degrés du système. Grâce à cette opération, la valeur affichée devient le nombre de degrés dont s'est déplacée la tournette. une fois le champ Position Actuelle configuré, nous avons l'angle précis de la tournette.

Il y a un bouton RESET POS 0 présent dans notre interface principale. C'est maintenant qu'il prend tout son sens. Il permet de remettre à 0 la position de l'axe actuel, ce qui va indirectement remettre à 0 l'angle dont la tournette s'est déjà déplacé dans l'affichage et permettre de repartir de la position actuelle comme l'angle 0 ou la position 0.

Ces lectures étant réalisées dans le Timeout de la structure événement, à chaque fois que le délais de Timeout (ici 100ms) est dépassé, sans qu'un événement ne se soit déclenché, les lectures sont effectuées.

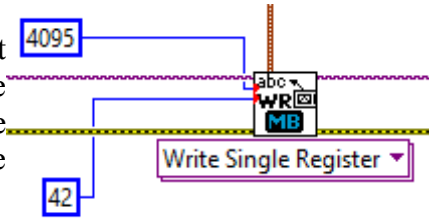
Diagramme partie Exploitation



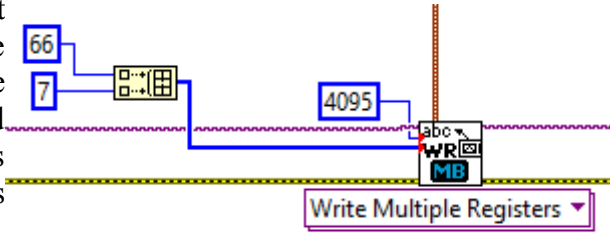
Un événement est associé à chaque bouton Go permettant de lancer un déplacement avec les valeurs de vitesse et d'angle qui lui sont associées.

Pour ce faire, nous utilisons le VI Déplacement du projet qui ne nécessite que la vitesse et l'angle pour lancer le déplacement.

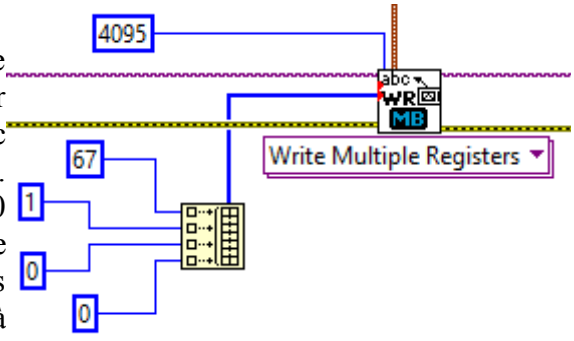
Cette opération est répétée pour chaque bouton Go de la partie Exploitation.



Le bouton STOP MOTEURS a son propre événement. Cet événement envoie simplement une commande ICNC_CMD_BREAKE_AXES_AND_CLEAR (Commande 42) à la carte, qui va arrêter tout déplacement sur les axes de celle-ci.



Le bouton UNLOCK a son propre événement et suit le même principe. Cependant, pour lui, la commande (Commande 66) qu'il envoie est sur 2 registres, le premier étant le numéro de la commande et le second la valeur débloquant les bits de verrouillage. Nous assemblons donc 2 valeurs en un tableau que nous passons à la fonction Write Multiple Registers.



Le bouton RESET POS 0 a lui aussi son propre événement. Nous utilisons le même principe que pour le bouton UNLOCK, nous construisons un tableau avec plusieurs constantes. 67 est le numéro de la commande. 1 représente l'axe à réinitialiser (l'axe X), et 0 et 0 correspondent aux valeurs que nous attribuons à cet axe en poids fort et en poids faible. Ce tableau est transmis à la fonction Write Multiple Registers toujours à l'adresse 4095 qui est réservée à l'envoi de commandes.

Le bouton PARAM quant à lui appelle simplement le VI Param du projet et l'ouvre.

Le bouton Test déplacement a un événement dont le contenu est semblable à celui des boutons Go de la partie Exploitation.

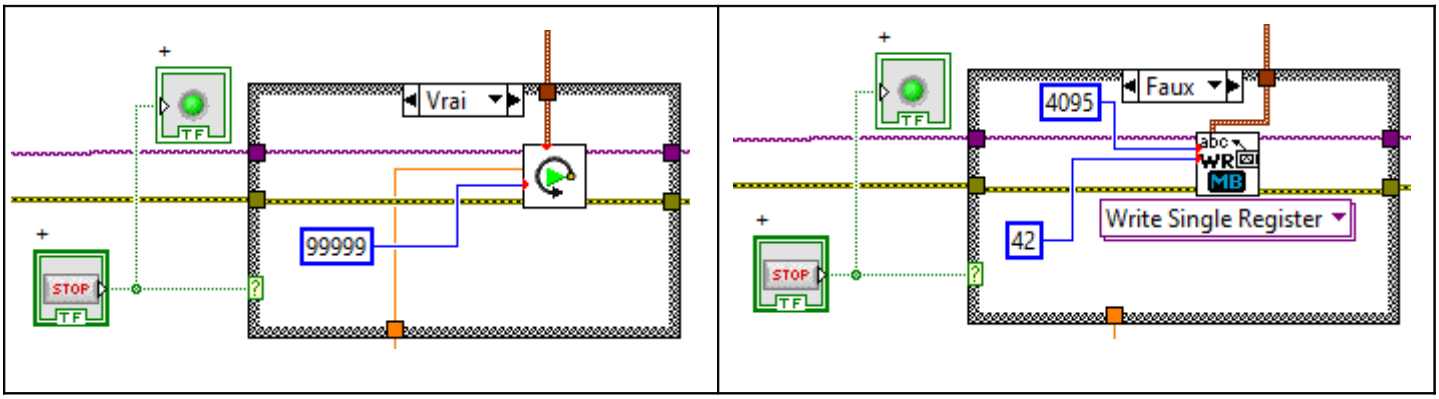
Diagramme partie Commandes Manuelles

Un événement est associé à chacun des boutons des commandes manuelles (+ et -).

Dans cette événement, une structure condition teste la valeur envoyée par le bouton et en fonction de celle-ci, si la valeur est vraie ; un déplacement est lancé avec la vitesse définie dans l'interface et une constante d'angle, si la valeur est fausse, alors c'est un stop moteurs qui est envoyé (commande 42).

Pour que les valeurs d'angle ne soient pas atteintes, de très grandes valeurs d'angle sont passées en constantes, 99999° et -99999° qui ne devraient normalement jamais être atteintes. Ces valeurs sont définies pour que le mouvement paraisse infini tant qu'on maintient le bouton enfoncé et que le déplacement s'arrête lorsque le bouton est relâché.

Vous trouverez ci-dessous le contenu de l'événement associé au bouton +, la même chose est réalisée pour le bouton – avec -99999 comme valeur d'angle envoyé.



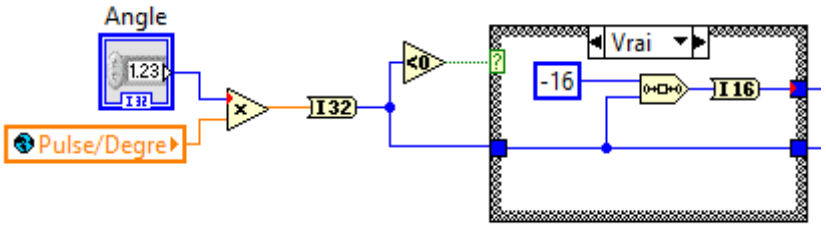
Un problème est apparu avec l'utilisation de valeur négatives qui sont envoyées à notre VI déplacement. Les valeurs étaient jusque-là traitées en valeur non signées. Pour remédier à cela, un traitement supplémentaire est effectué si notre valeur est négative. Nous avons du adapter le type de registres envoyés via la librairie Modbus en valeur signées.

Cependant nous ne pouvons pas modifier simplement la représentation des registres envoyés. Dans un souci de simplification, nous « divisons » les valeurs rentrées par l'utilisateur qui nécessitent d'être passées à la fonction en 2 registres, de poids fort et de poids faible. Le problème apparaît ici, lorsque nous divisons cette valeur elle est répartie en 2 registres. Nous pouvons décider que la valeur de ces registres soit signée ou non, nous choisissons donc des valeurs signées. Cela provient du fait qu'on ne peut pas stocker autant de valeurs dans un registre 32 bits signés que dans 2 registres 16 bits signés. Ce problème est causé par le bit de signe des registres. Il y a 1 bit de signe dans un registre 32 bits et 2 bits de signe , 1 par registre dans les 2 16 bits que nous utilisons.

Nous devions trouver une solution pour permettre l'utilisation de valeurs négatives sans avoir une importante perte de précision.

Nous avons trouvé une solution en modifiant une nouvelle fois la librairie Modbus National Instruments. Nous avons modifié la représentation des valeurs du tableau envoyés au Write Multiple Registers de I16 vers I32. Avec cette modification, nous pouvons utiliser les 16 bits pour le poids faible sans bit de signe.

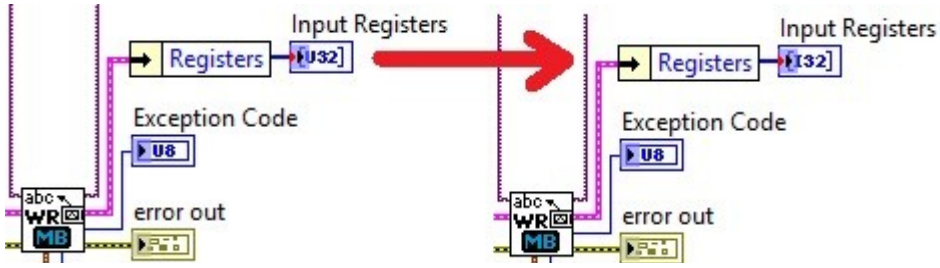
Nous testons dans l'application si la valeur rentrée est négative ou positive. Si elle est positive, nous utilisons masque et décalage, si elle est négative, alors nous passons le poids faible sans traitement et nous convertissons le 32 bits du poids fort en 16 bits dans notre application afin que le bit de signe soit bien pris en compte.



De cette façon, nous avons notre valeur 16 bits en poids faible sans bit de signe et notre valeur 16 bits de poids fort avec son bit de signe.

Dans le Timeout de notre structure événement, la lecture de l'angle actuel était faite en valeur non signée. Lorsque nous avions un angle négatif, des valeurs aberrantes étaient affichées. Nous avons modifié la représentation de la valeur retournée.

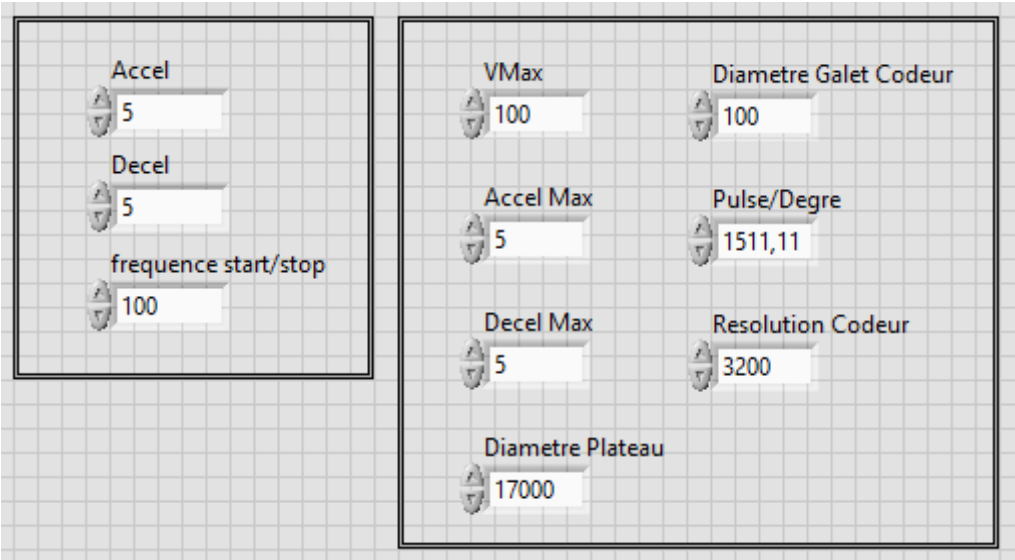
Nous avons modifié la représentation des Registres reçus dans MB Serial Query Read, nous les avons fait passer de U32 (mot long non signé) vers I32 (mot long (sous entendu signé)).



Comme dans toutes application Modbus, l'initialisation de la communication par la fonction INIT induit la fermeture de la liaison série et l'affichage des éventuelles erreurs de communication effectuées à la fermeture de l'application. C'est pourquoi nos Visa Ressource Name et Error Out finissent leur course dans un Visa Close.

Les VIs

- Une Variable globale regroupe les données écrites dans le VI Paramètre et ces valeurs accessibles dans le VI Déplacement pour les conversions nécessaires.



- Un VI Paramètre permet de configurer des valeurs systèmes et utilisateur telles que l'accélération et la décélération des déplacements mais aussi la taille de la tournette, de son galet codeur, la résolution du codeur… Les valeurs rentrées par l'utilisateur sont directement envoyées dans la variable globale de l'application.

- Un VI Déplacement effectue les conversions pour le système et constitue la commande de déplacement. Ce VI reprend les conversions déjà utilisées dans le VI Déplacement de la librairie ICNC USB, puis ces valeurs sont rassemblées pour former un tableau qui est passé en paramètre d'une fonction Write Multiple Registers.

Utilisation de l'Application

Cette application sert d'exemple de l'utilisation du pilotage par communication Modbus utilisant les commandes de la carte. Cette application est fonctionnelle comme décrite dans le cahier des charges page 14.

Pour utiliser cette application, vous devez dans une premier temps connecter la carte à un port COM de votre ordinateur. Ensuite, vous devez configurer les paramètres de la communication Modbus dans l'interface principale avant même de l'exécuter. Vous pouvez trouver dans TestCenter « Edition → Paramètres InterpCNC → Divers » la configuration Modbus exacte de votre carte, telle que son adresse esclave, sa vitesse de communication et la parité.

Une fois la communication configurée, vous pouvez lancer l'application. Il vous faudra configurer les paramètres machine de votre application via l'interface paramètre et ensuite votre application sera prête à être utilisée.

Historique Modification Librairie Modbus

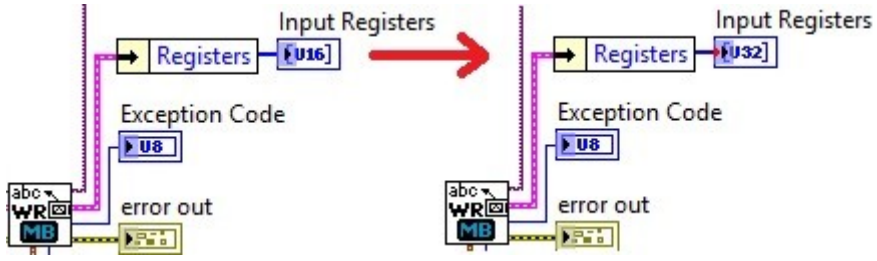
Nous utilisons la librairie Modbus de National Instrument pour toutes les communications effectuées avec les cartes. Cependant, pour nous adapter à notre matériel et à certaines contraintes nous avons été amenés à modifier directement la librairie Modbus de National Instruments principalement pour adapter certaines représentations des données transmises.

Nous avons regroupé dans cette partie la liste des modifications effectuées sur la librairie Modbus pour qu'elle s'adapte à notre matériel.

MB Serial Master Query Read

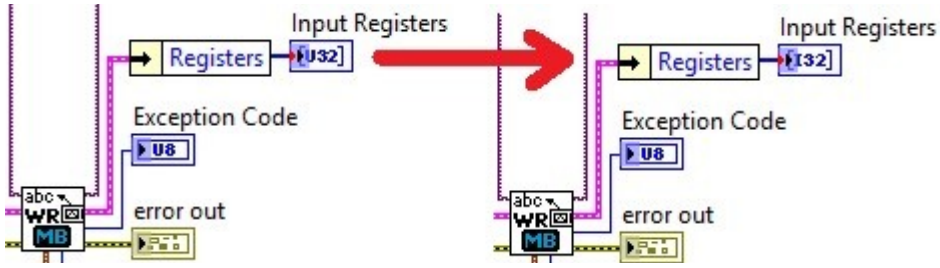
Cette fonction retourne les contenus des registres dans des valeurs 16bits, donc avec nos décalages nous sortons des valeurs qu'elle enregistre.

Pour résoudre ce problème, il va nous falloir modifier le format de valeur que la fonction retourne. Nous avons affiché le contenu du VI et modifié la représentation du tableau retourné de 16bits en 32bits.



Lorsque nous avons des valeurs négatives, des valeurs aberrantes étaient affichées. Nous avons modifié une seconde fois la représentation de la valeur retournée.

Nous avons fait passer la représentation des registres de U32 (mot long non signé) vers I32 (mot long (sous entendu signé)). Les valeurs négatives sont maintenant correctement affichées.



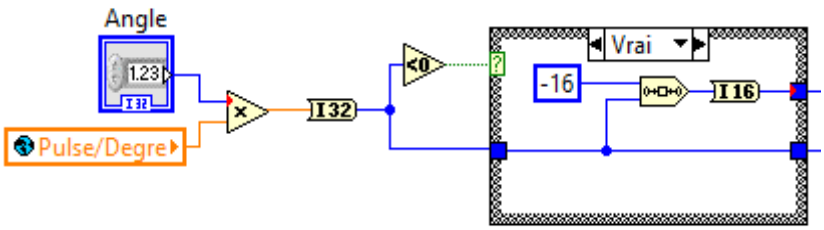
Write Multiple Registers

Le tableau de registre que nous envoyons était de type U16 (mot non signé). Cette représentation ne nous permettait pas d'envoyer des valeurs négatives. Nous avons changer cette représentation pour le type I16 (mot (sous entendu signé)) qui nous permet l'utilisation de valeurs négatives grâce à un bit de signe.

Cependant, un problème est apparu à la suite de cette modification. Le changement de la représentation en 16 bits signés pour le tableau implique que toutes les valeurs soient passées en signés. Le problème est qu'on ne peut pas stocker autant de valeurs avec un 32 bits signés qu'avec 2 16 bits tous les deux signés. Nous avons 2 bits de signe au lieu d'un, et il nous est impossible de stocker autant de valeurs. De plus la position du bit de signe du registre de poids faible réduit fortement la précision.

Nous avons trouvés une solution en modifiant une nouvelle fois la librairie Modbus National Instruments. Nous avons modifié la représentation des valeurs du tableau envoyé au Write Multiple Registers de I16 vers I32. Avec cette modification, nous pouvons utiliser les 16 bits pour le poids faible sans bits de signe.

Nous testons dans l'application si la valeur rentrée est négative ou positive. Si elle est positive, nous utilisons masque et décalage, si elle est négative alors, nous passons le poids faible sans traitement et nous convertissons le 32 bits du poids fort en 16 bits dans notre application afin que le bit de signe soit bien pris en compte.



De cette façon nous avons notre valeur 16 bits en poids faible sans bit de signe et notre valeur 16 bits de poids fort avec son bit de signe.

Librairie ICNC Modbus

Vous pouvez installer cette librairie comme vous avez installé la librairie Modbus de National Instrument au début de ce document.

Une librairie ICNC Modbus à été créée. Celle-ci a pour but de fournir une réponse rapide à un besoin d'utiliser Labview pour piloter les cartes ICNC via Modbus.

Les VIs qui composent cette librairie sont des modèles permettant d'utiliser les différentes commandes disponibles via Modbus. Ils vont vous permettre d'utiliser ces commandes de manière simple.

Il est possible que vous ayez besoin d'effectuer des commandes de manière plus complexe dans quel cas nous devrez les former vous-même, si les VIs disponibles ici ne suffisent pas.

Vous pourrez trouver dans les diagrammes de ces VIs des informations supplémentaires sur le type de valeurs attendues.

Ces VIs utilisent une Write Multiple Registers pour envoyer leur commande à l'exception des commandes d'arrêt 42 et 43 qui n'ont nécessité qu'un Write Single Register.

Tous ces VIs envoient une commande à l'adresse 4095. Ils envoient à cette adresse un tableau contenant les différents paramètres de la fonction telles que décrite dans le pdf Protocole MODBUS InterpCNC V2.1 . Ce tableau est assemblé avec les différentes valeurs attendues pour l'utilisation de la commande concernée et il est envoyé directement à la fonction Modbus. La première de ces valeurs est toujours le numéro de la commande. Il est suivi par les différentes valeurs propres à chaque commande dont vous trouverez le détail dans le pdf Protocole MODBUS InterpCNC V2.1.