Programmation des contrôleurs d'axes SOPROLEC

Summary

1 Préambule

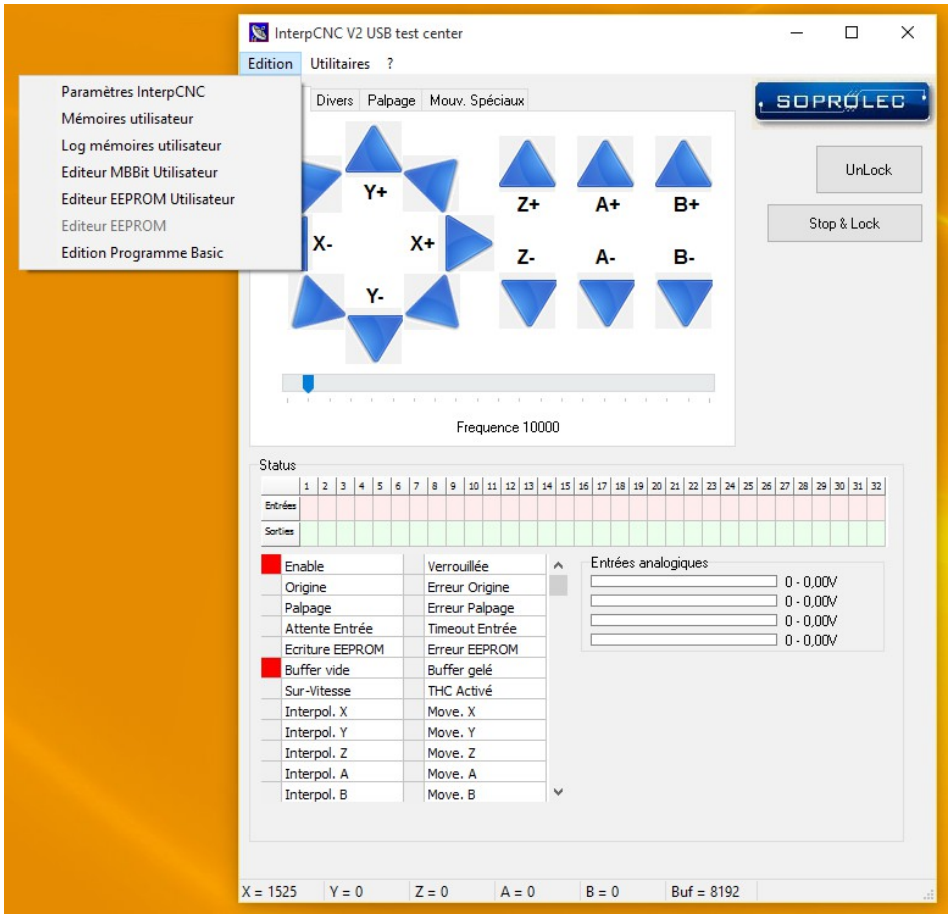
Ce document est destiné à vous expliquer comment programmer nos controleurs d'axes programmables industriels 3 axes et 5 axes.

Au préalable il est fortement recommandé de prendre connaissance des notices relatives à ces cartes et d'installer le logiciel Test Center (disponibles sur notre site www.soprolec.com dans l'onglet Télécharger de nos cartes).

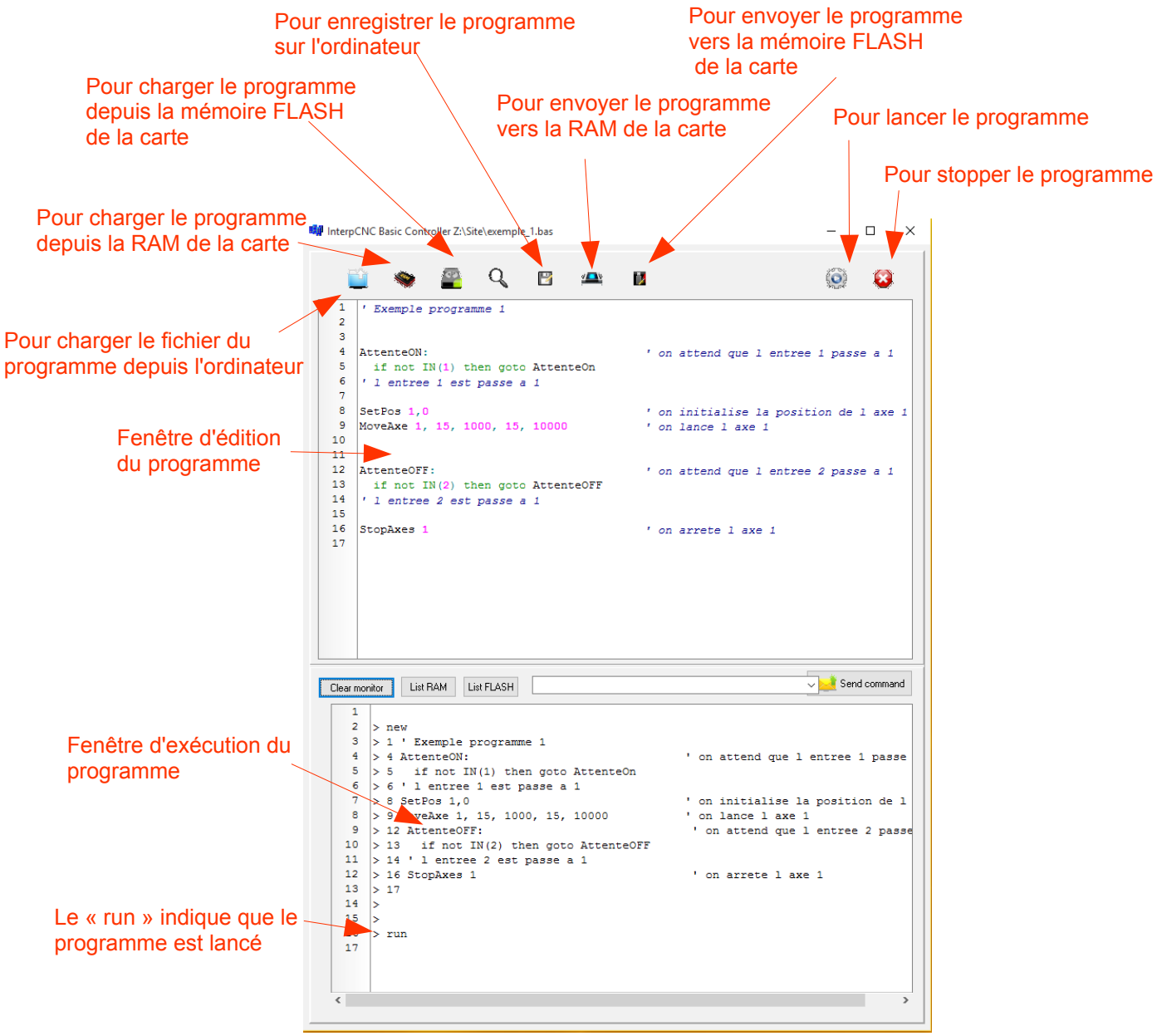
Le logiciel Test Center se trouve dans le dossier "Diagnostic" de l'archive, il n'y a pas d'installation vous le déposez où vous le souhaitez.

2 Utilisation de Test Center en mode programmmation

Menu Edition sur l'écran d'accueil de Test Center :



L'éditeur Basic est accessible à partir de l'élément de menu “Edition Programme Basic” :



Le contrôleur d'axes utilise 2 types de mémoire :

- De la RAM pour stocker le programme de façon volatile : en cas d'extinction de la carte le programme est perdu,
- De la mémoire FLASH pour stocker le programme de façon non-volatile, c'est ce qui permet de lancer l'exécution automatique d'un programme à l'allumage de la carte.

De façon générale on utilise la RAM en phase de développement et la FLASH en phase d'exploitation.

Le programme doit avoir été chargé en RAM pour pouvoir ensuite l'écrire en FLASH.

On peut charger un programme dans la fenêtre d'édition depuis 3 sources différentes : un fichier stocké sur l'ordinateur, la mémoire RAM et la mémoire FLASH.

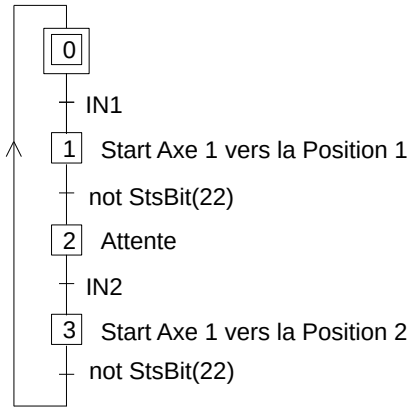
3 Exemples de programmes Basic

3.1 Exemple 1 : lancement et arrêt d'un axe avec 2 boutons

Ce programme attend l'appui sur le bouton START raccordé sur l'entrée 1 pour lancer un axe (l'axe 1) pour 10 000 pulses avec une accélération et une décélération de 15 kHz/s et une vitesse de 1000 Hz.

Le moteur s'arrête tout seul lorsqu'il a atteint sa position cible ou lorsqu'on appuie sur le bouton STOP raccordé sur l'entrée 2.

3.1.1 Cycle



3.1.2 Programme Basic

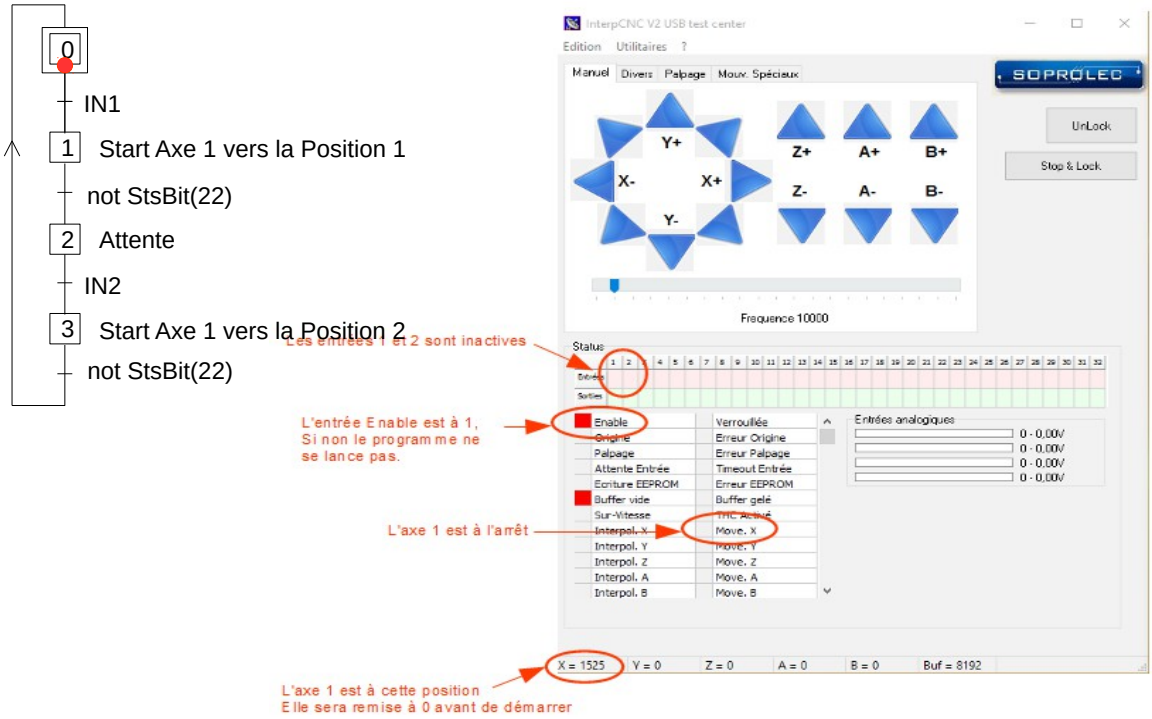
```
' Exemple programme 1
AttenteIN1:                              ' on attend que l entree 1 passe a 1
  If Not In(1) Then GoTo AttenteIN1
  ' l entree 1 est passe a 1
  SetPos 1,0                             ' on initialise la position de l axe 1
  MoveAxe 1, 15, 1000, 15, 10000         ' on lance l axe 1
AttentePos1:
  if not StsBit(22) then GoTo AttentePos1  ' attente fin de mouvement
AttenteIN2:                              ' on attend que l entree 2 passe a 1
  If Not In(2) Then GoTo AttenteIN2
  ' l entree 2 est passe a 1
  MoveAxe 1, 15, 1000, 15, 0             ' on lance l axe 1 vers position 0
AttentePos2:
  if not StsBit(22) then GoTo AttentePos2  ' attente fin de mouvement
Goto AttenteIN1                          ' retour début de programme
```

En utilisant Test Center saisissez le programme dans l'éditeur Basic.

3.1.3 Exécution du programme

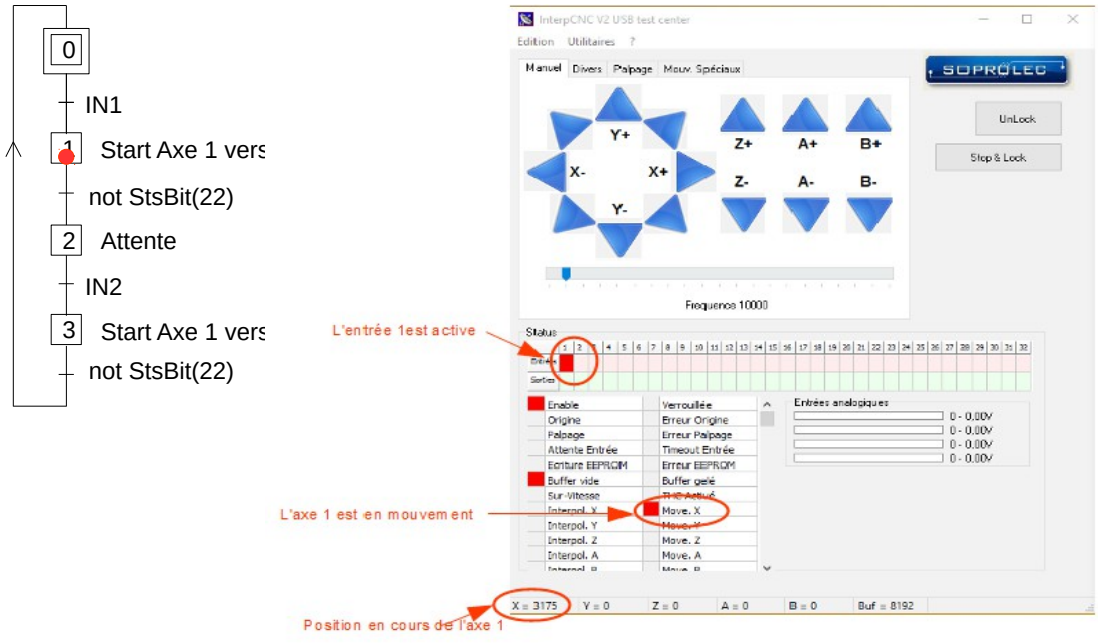
L'étape active est l'étape 0 la transition attend le passage à 1 de l'entrée 1, le programme tourne en boucle sur AttenteIN1 :

```
AttenteIN1:                              ' on attend que l entree 1 passe a 1
  If Not In(1) Then GoTo AttenteIN1
```



On active l'entrée 1, la transition est franchie, l'étape active devient l'étape 1 :

```
MoveAxe 1, 15, 1000, 15, 10000           ' on lance l axe 1
```



L'entrée 1 est redevenue inactive, l'action MoveAxe a été lancée, l'axe se déplace, on attend le franchissement de la transition de l'étape 1, le programme tourne en boucle sur AttentePos1:

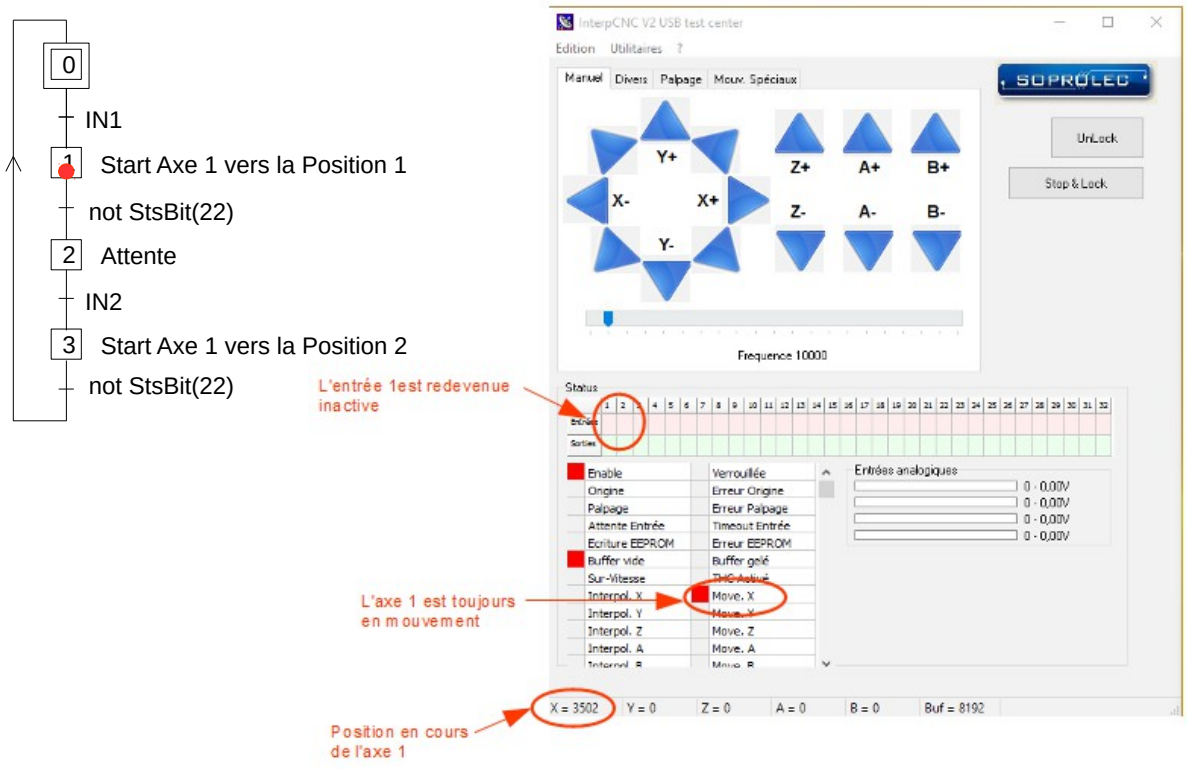
```
AttentePos1:
  if not StsBit(22) then GoTo AttentePos1  ' attente fin de mouvement
```

StsBit(22) est le bit d'état de mouvement de l'axe 1, il est à 1 pendant tout le temps que dure le mouvement.

Il est important de comprendre que la commande MoveAxe gère le déplacement de l'axe du début jusqu'à la fin du déplacement, c'est à dire depuis la phase d'accélération jusqu'à la phase de décélération. Si on indique un déplacement de 10 000 pas l'axe effectuera exactement 10 000 pas.

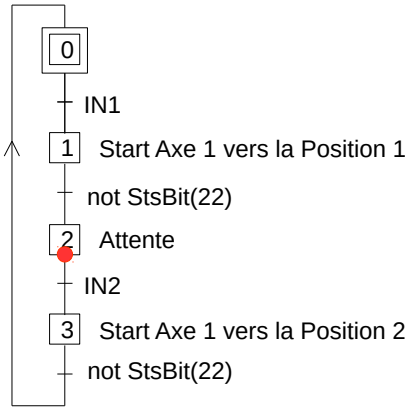
Il ne faudrait pas utiliser la commande GetPos pour savoir si l'on est arrivé à 10 000 pas puis faire un StopAxe. Dans ce cas le déplacement serait de 10 000 plus le nombre de pas nécessaires à l'arrêt de l'axe.

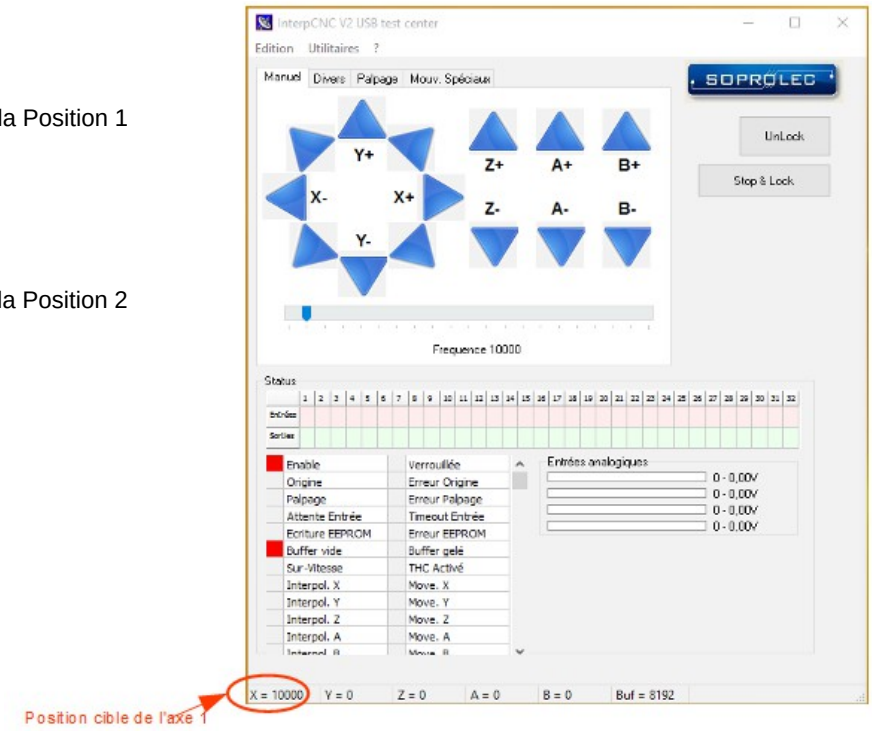
GetPos doit être utilisé pour connaître la position en cours de l'axe et lancer d'autres actions par exemple.



L'axe a terminé son déplacement, la transition est franchie. C'est maintenant l'étape 2 qui est active, on attend le franchissement de la transition, le programme tourne en boucle sur AttenteIn2 :

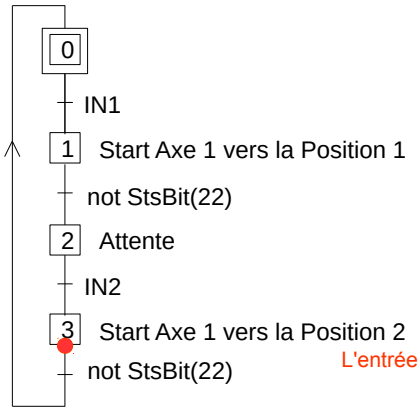
```
AttenteIN2:                          ' on attend que l entree 2 passe a 1
  If Not In(2) Then GoTo AttenteIN2
```

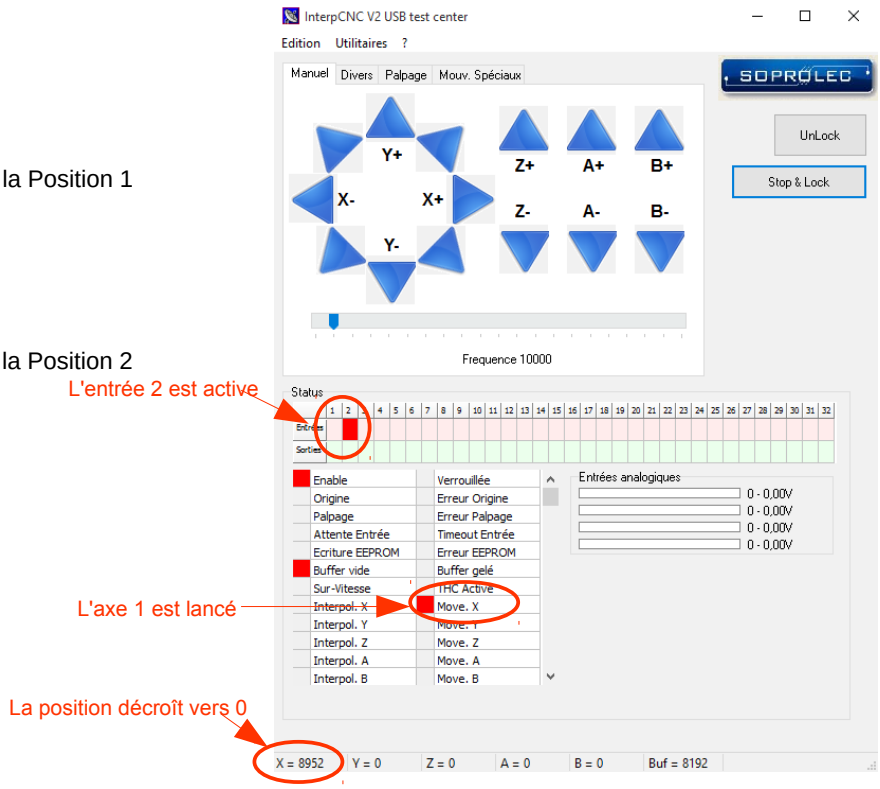




On active l'entrée 2, la transition est franchie, l'axe repart vers la position 0, C'est maintenant l'étape 3 qui est active :

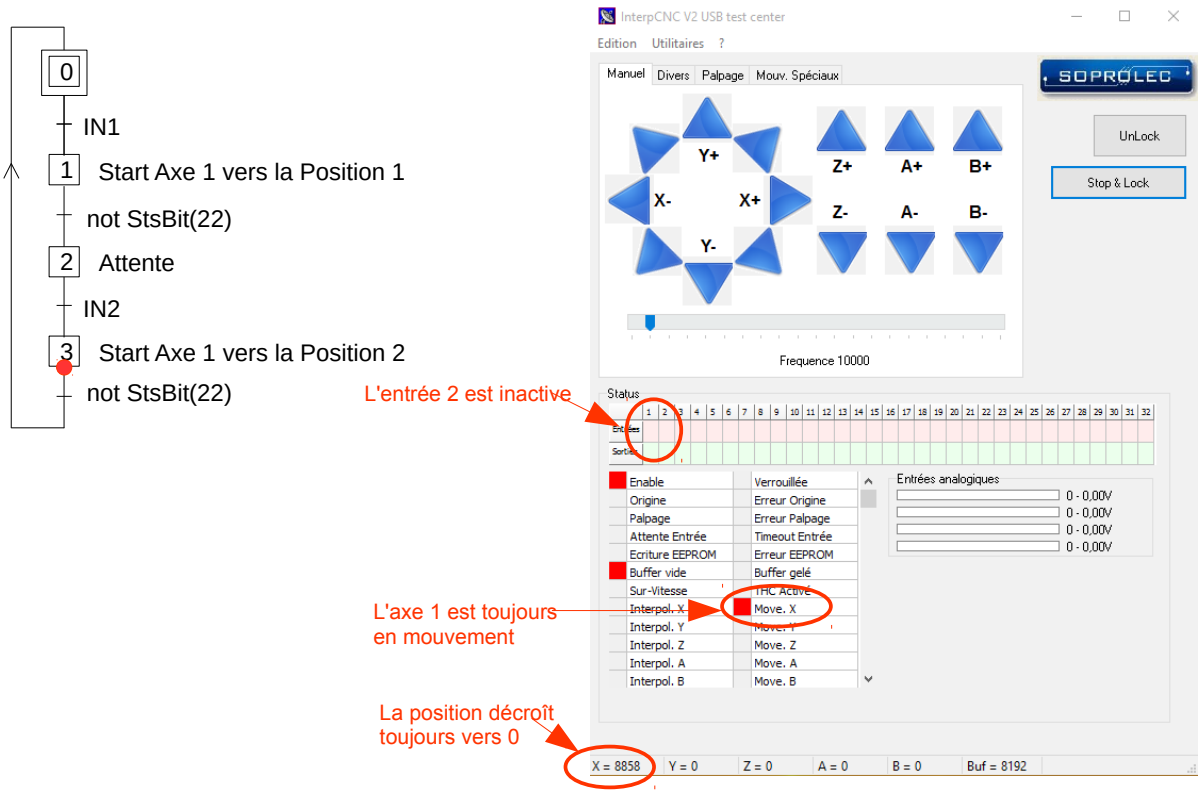
```
MoveAxe 1, 15, 1000, 15, 0           ' on lance l axe 1 vers position 0
```



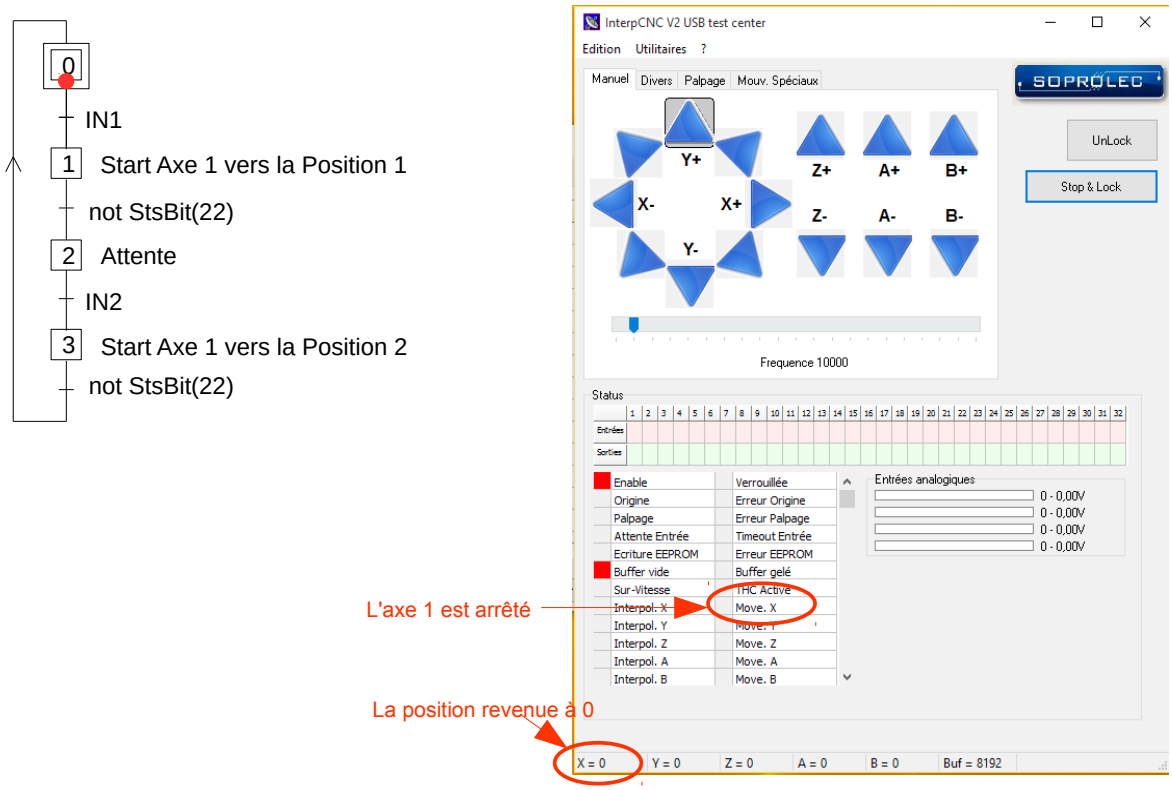


L'entrée 2 est redevenue inactive, l'action MoveAxe a été lancée, l'axe se déplace, on attend le franchissement de la transition de l'étape 3, le programme tourne en boucle sur AttentePos2:

```
AttentePos2:
  if not StsBit(22) then GoTo AttentePos2  ' attente fin de mouvement
```



L'axe a terminé son déplacement, la transition est franchie. On remonte à l'étape 0 :



3.2 Exemple 2 : déplacemennt d'un axe sur une course donnée et homing

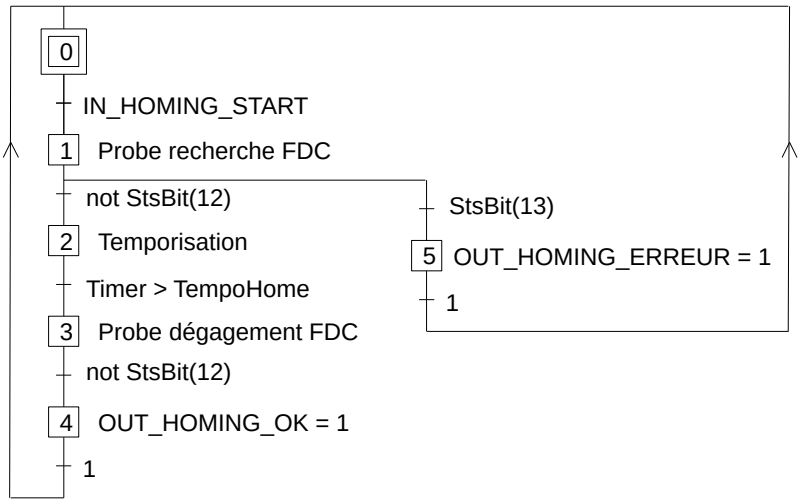
3.2.1 Cycles

Dans cet exemple l'architecture du programme diffère un peu de celle de l'exemple 1 :

- au lancement du programme :
  - déclaration des constantes et des entrées/sorties
  - initialisation de la carte
- boucle DO – LOOP qui gère l'exécution des cycles
  - calcul de la position et affectation à UserMem 0
  - cycle AR
  - cycle Homing

Dans la boucle on recherche l'étape active pour chaque cycle et on teste la "franchissabilité" de la transition.

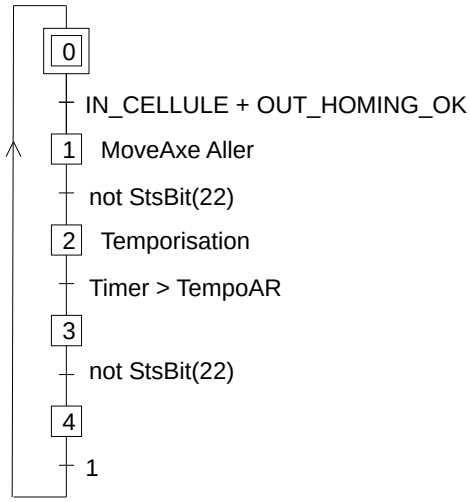
Cycle de Homing



Le cycle est déclenché par la mise à l'état 1 de l'entrée 2 (IN_HOMING_START).

Le FDC est raccordé sur l'entrée 1 (IN_HOME).

Cycle Aller Retour



Le cycle est déclenché sur un front montant de l'entrée IN_CELLULE et si la sortie OUT_HOMING_OK est à 1, c'est à dire si le Homing a été fait et s'est bien terminé.

Le programme Basic "élagué" de ce cycle est le suivant :

```
do   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -' boucle
        SetUserMem 0, GetPos(1)/GetEEData16(EE_RESOLUTION)   -   -   -' commandes exécutées a chaque
passage dans la boucle

        if EtapeAR = 0 then   -   -   -   -   -   -   -   -   -   -   -' étape active
                if DFM(IN_CELLULE) and GetOut(OUT_HOMING_OK) then   -' franchissement de la transition
                        'Action de l'étape 1
                        EtapeAR = 1 -   -   -   -   -   -   -   -   -' nouvelle étape active
        elseif EtapeAR = 1 then
                if not StsBit(22) then
                        'Action
                        EtapeAR = 2
                endif
        elseif EtapeAR = 2 then
                if Timer > TempoAR then
                        'Action
                        EtapeAR = 3
                endif
        elseif EtapeAR = 3 then
                if not StsBit(22) then
                        'Action
                        EtapeAR = 0
                endif
        endif
loop
```

Comme on peut le voir l'instruction SetUserMem est exécutée à chaque itération, ensuite on recherche l'étape active ainsi seule la réceptivité de la transition de l'étape active est testée et éventuellement franchie. Il est possible de tester ainsi plusieurs cycles qui s'exécutent simultanément.

Encore une fois il est impératif que les actions et tests de réceptivité ne soient pas bloquant pendant l'exécution de chacun des cycles, exemple de points bloquants :

- ➔ Pause x secondes
- ➔ boucle du style :

```
AttenteIN1:                              ' on attend que l entree 1 passe a 1
  If Not In(1) Then GoTo AttenteIN1
```

- ➔ ...

3.2.2 Programme Basic

Dans cet exemple on introduit l'utilisation de plusieurs fonctions et commandes :

- CONST : pour stocker un numéro de sortie, un numéro de sortie, un numéro de registre…
- Une étiquette ou Label (Initialisation:) pour réinitialiser la carte dans un état précis
- IN /OUT
- GetEEData16(Numéro de registre) pour accéder aux données stockées en mémoire FLASH.
- …

```
' Programme demo pour aller-retour + homing
' V1 24/11/2015 : Version initiale


' Affectation des entrees
const IN_HOME              = 1 ' le capteur de fin de course commande Probe sur entree 1
const IN_HOMING_START      = 2 ' bouton de lancement homing sur entree 2
const IN_CELLULE           = 3 ' cellule de debut cycle sur entree 3

' Affectation des sorties
const OUT_HOMING_ERREUR    = 1 ' pour voyant erreur de homing sur sortie 1
const OUT_HOMING_OK        = 2 ' pour voyant homing ok sur sortie 2
const OUT_MOVING           = 3 ' pour voyant deplacement en cours sur sortie 3

' Paramètres
' stockes dans l'EEPROM de la carte:  0 = registre 5120
const EE_COURSE            = 0     ' (mm)
const EE_VITESSE           = 1     ' (mm/s)
const EE_TEMPO             = 2     ' (ms)
const EE_ACCEL             = 3     ' (kHz/s)
const EE_V_HOMING          = 4     ' (mm/s)
const EE_ACCEL_HOMING      = 5     ' (kHz/s)
const EE_RESOLUTION        = 6     ' (pulses/mm)

Initialisation:
    ? "Initialisation" : ? " "
    EtapeHoming = 0
    EtapeAR = 0
    StopAxes 1
    For i=320 To 399 : SetMBBit i, 0 : Next i ' remise a zero bits d etat
    OUTALL 0                                  ' remise a zero de toutes les sorties
    Pause 200
    Unlock

do
  ' L'entree ENABLE a ete coupee = Arret d Urgence
  if stsbit(8)=1 then goto Initialisation

    SetUserMem 0, GetPos(1)/GetEEData16(EE_RESOLUTION)    'Calcul de la position en mm et on place la
valeur dans la UserMem 0
    SetUserMem 1, EtapeHoming                             ' pour suivre les etapes du homing
    SetUserMem 2, EtapeAR                                 ' pour suivre les etapes AR

  ' cycle AR
  if EtapeAR= 0 then
    if DFM(IN_CELLULE) and GetOut(OUT_HOMING_OK) then  ' detection de la cellule sur front montant et le
voyant homing ok doit etre allume
      Cible = GetEEData16(EE_COURSE)*GetEEData16(EE_RESOLUTION)
      ? "Start axe 1 Aller vers Cible : ", Cible, " pulses, ", GetEEData16(EE_COURSE), " mm"
      MoveAxe 1, GetEEData16(EE_ACCEL), GetEEData16(EE_VITESSE)*GetEEData16(EE_RESOLUTION),
GetEEData16(EE_ACCEL),  Cible ' lancement axe 1
      OUT OUT_MOVING, 1                       ' on allume le voyant axe en mouvement
      EtapeAR = 1                             ' on rend active l etape 1, deplacement aller de l axe
    endif
  elseif EtapeAR = 1 then                  ' cas ou l etape 1 du cycle AR est active  (aller en cours)
    if not StsBit(22) then                 ' bit d indication de mouvement de l axe 1, lorsqu il
revient a 0 le mouvement est termine on franchit la transition de l etape active 1
      OUT OUT_MOVING, 0                    ' on eteint le voyant axe en mouvement
      ? "Axe 1 cible Aller atteinte"
```

```
      ? "Debut tempo AR"
      TempoAR = Timer + GetEEData16(EE_TEMPO)
      EtapeAR = 2                          ' on rend active l etape 2, temporisation entre 2
deplacements
    endif
  elseif EtapeAR = 2 then                  ' cas ou l etape 2 du cycle AR est active
    if Timer > TempoAR then                ' tempo AR terminee on franchit la transition de l etape
active
      ? "Tempo ecoulee"
      Cible = 0
      ? "Start axe 1 Retour vers Cible : ", Cible, " pulses, ", 0, " mm"
      MoveAxe 1, GetEEData16(EE_ACCEL), GetEEData16(EE_VITESSE)*GetEEData16(EE_RESOLUTION),
GetEEData16(EE_ACCEL),  Cible ' lancement axe 1
      OUT OUT_MOVING, 1
      EtapeAR = 3                          ' on rend active l etape 3, deplacement retour de l axe
    endif
  elseif EtapeAR = 3 then                  ' cas ou l etape 3 du cycle AR est active  (retour en cours)
    if not StsBit(22) then                 ' bit d indication de mouvement de l axe 1, lorsqu il
revient a 0 le mouvement est termine on franchit la transition de l etape active 3
      ? "Axe 1 cible Retour atteinte" : ? " "
      OUT OUT_MOVING, 0
      EtapeAR = 0                          ' on rend active l etape 0, attente cellule
    endif
  endif

  ' Homing

  ' il y a 4 etapes :
  '  - etape 0 : etape d attente
  '  - etape 1 : etape de recherche de FDC
  '  - etape 2 : etape de tempo entre les 2 deplacements
  '  - etape 3 : etape de degagement du FDC

  if EtapeHoming= 0 then
    if DFM(IN_HOMING_START) then ' detection du bouton homing
      ? "Start probe axe 1 recherche capteur"
      Probe 1, -1, IN_HOME, 1, GetEEData16(EE_COURSE)*GetEEData16(EE_RESOLUTION),
GetEEData16(EE_ACCEL_HOMING), GetEEData16(EE_V_HOMING)*GetEEData16(EE_RESOLUTION),
GetEEData16(EE_ACCEL_HOMING)
      OUT OUT_HOMING_ERREUR, 0             ' on initialise les sorties a 0
      OUT OUT_HOMING_OK, 0                 ' on initialise les sorties a 0
      EtapeHoming = 1                      ' on rend active l etape 1, etape recherche capteur
    endif
  elseif EtapeHoming = 1 then              ' cas ou l etape 1 est active
    if StsBit(13) then                     ' si fonction probe en erreur
      ? "Capteur FDC axe 1 non trouve"
      OUT OUT_HOMING_ERREUR, 1             ' on allume le voyant erreur homing
      OUT OUT_HOMING_OK, 0                 ' le homing n est pas ok
      EtapeHoming = 0                      ' on rend active l etape 0, etape attente
    elseif  not StsBit(12) then            ' sinon si probe est termine : la transition qui suit l etape
active 1 est franchie
      ? "Capteur FDC axe 1 trouve, degagement capteur"
      ' on lance une petite tempo (500ms)
      TempoHome = Timer + 500
      EtapeHoming = 2                      ' on rend active l etape 2, temporisation
    endif
  elseif EtapeHoming = 2 then              ' cas ou l etape 2 est active
    if Timer > TempoHome then              ' si tempo est terminee  : la transition qui suit l etape
active 2 est franchie
       ' on lance un probe en sens inverse sur une petite course (1/10) a vitesse lente (1/10) pour
detecter la sortie du capteur
        Probe 1, 1, IN_HOME, 0, GetEEData16(EE_COURSE)*GetEEData16(EE_RESOLUTION)/10,
GetEEData16(EE_ACCEL_HOMING), GetEEData16(EE_V_HOMING)*GetEEData16(EE_RESOLUTION)/10,
GetEEData16(EE_ACCEL_HOMING)
        EtapeHoming = 3                    ' on rend active l etape 3, etape degagement capteur
    endif
  elseif EtapeHoming = 3 then              ' cas ou l etape 3 est active
    if not StsBit(12) then                 ' si probe est termine  : la transition qui suit l etape
active 3 est franchie
      ? "Capteur FDC axe 1 degage" : ? " "
      EtapeHoming = 0
      OUT OUT_HOMING_OK, 1                 ' on allume le voyant homing OK, on l utilise aussi pour
autoriser le lancement du cycle AR
      SetPos 1, 0                          ' on initialise la position de l axe a 0
    endif
  endif
loop
```

3.2.3 Exécution du programme

Dans cet exemple nous utilisons la commande Probe pour réaliser le Homing. Vous verrez dans des exemples plus loin que nous pouvons aussi utiliser une autre méthode. La prinicipale différence réside dans le fait que la commande Probe ne gère qu'un seul homing à la fois alors que l'autre méthode (un peu plus complexe) peut gérer plusieurs homing simultanés.

La commande Probe prend plusieurs paramètres :

1. le numéro d'axe
2. la direction du déplacement : positif ou négatif
3. le numéro de l'entrée associée, celle où est raccordé le capteur de fin de course (FDC)
4. le niveau attendu de l'entrée (0 ou 1) en fonction du type de capteur (NC ou NO) mais aussi de la signification du changement d'état
5. la limite de course pour la recherche du FDC
6. l'accélération
7. la vitesse
8. la décélération

Lorsque la commande Probe est lancée, la recherche de FDC commence mais elle rend tout de suite la main, c'est à dire que l'on peut exécuter d'autre commandes, en contrepartie il faut surveiller les bits d'état liés à la commande Probe :

- ➔ StsBit(12) : à 1 tant que la commande est en cours
- ➔ StsBit(13) : à 1 en cas d'erreur si le FDC n'est pas trouvé par exemple

Dans le cycle de Homing de notre exemple, nous lançons la recherche dans une première direction (négative), l'objectif est de trouver le FDC qui est connecté sur l'entrée 1 et on attend qu'elle passe à 1, puis on surveille les bits d'erreur et de Probe en cours.

Si la première commande Probe s'est bien terminée, on fait une pause, puis on relance une autre commande Probe pour se dégager du capteur FDC, c'est à dire trouver l'endroit exact où il change d'état.

Pour cette 2éme commande certains paramètres diffèrent :

2. la direction est inverse

4. le niveau attendu du FDC est inverse également : on attend qu'il passe à 0 (puisqu'il était passé à 1)

5. la limite de course est plus petite (généralement 1/10 de la précédente)

7. la vitesse est plus faible (généralement 1/10 de la précédente)

On utilise aussi la fonction Timer pour gérer une temporisation, le principe est simple: on affecte la valeur de Timer + la valeur de la tempo à une variable et on teste si Timer est arrivé à la valeur de la variable.

La différence avec la commande Pause c'est qu'elle est bloquante, le programme attend qu'elle soit terminée : si elle est de 10 secondes le programme attend pendant 10 secondes. Dans notre exemple le programme fait d'autres choses et vérifie régulièrement si les 10 secondes sont écoulées.

Nous verrons dans d'autres exemples que cette méthode est aussi utilisée pour gérer les TimeOut.

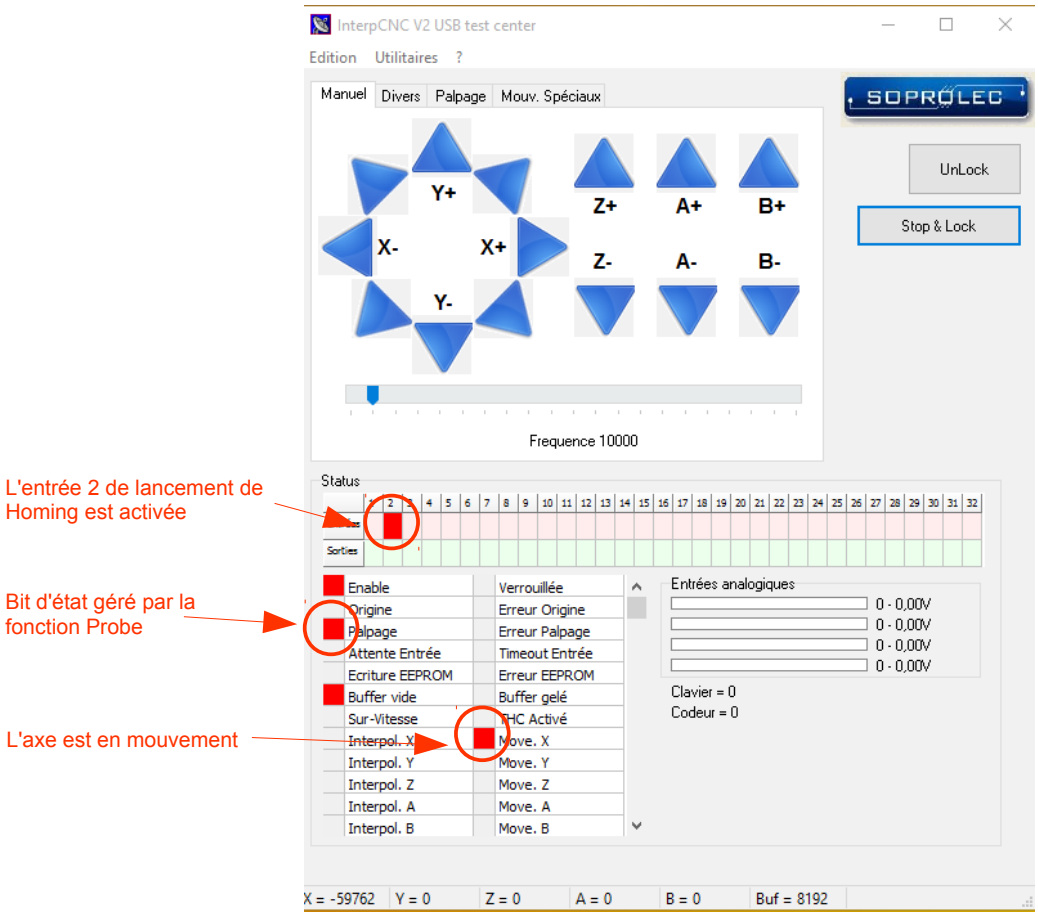
A chaque passage dans la boucle nous calculons et affichons la position de l'axe :

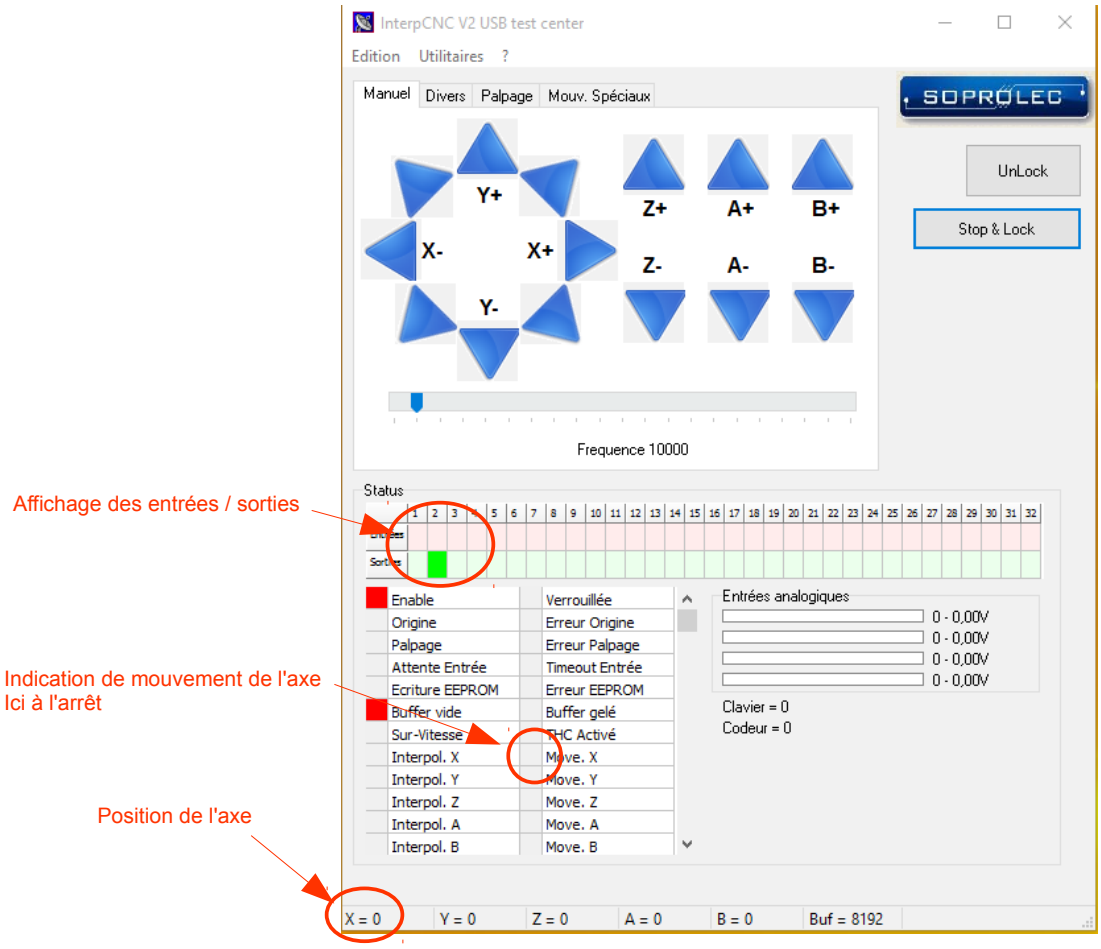
```
    SetUserMem 0, GetPos(1)/GetEEData16(EE_RESOLUTION)     'Calcul de la position en mm et on place la valeur dans la UserMem 0
```

La fonction GetPos(axe) permet de récupérer la position en pas, la division par la résolution (GetEEData16(EE_RESOLUTION)) la convertit en mm (pour mémoire la résolution telle qu'indiquée dans le programme attend des pulses par mm).

Nous ne reprendrons pas de façon aussi détaillée que dans le premier exemple les différentes étapes et écrans, néanmoins vous pouvez suivre l'exécution et l'évolution des paramètres dans ces différentes fenêtres, par exemple :

Dans la fenêtre principale de Test Center





Grâce à cette fenêtre nous pouvons suivre l'état des entrées et des sorties, le déplacement ou non de l'axe, et la position de l'axe :

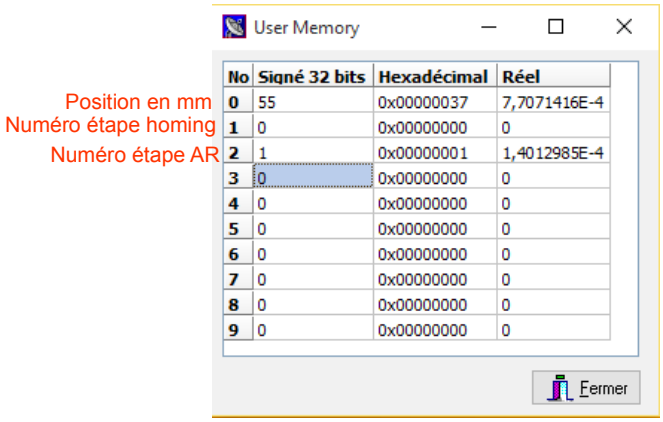
- ➔ position au moment de la demande de Homing
- ➔ évolution pendant le déplacement
- ➔ mise à 0 une fois que le Homing est terminé correctement
- ➔ déplacement vers la cible puis vers la position 0 pendant le cycle d'AllerRetour

Dans l'exemple nous avons utilisé des sorties pour indiquer par des voyants :

- ➔ erreur de homing
- ➔ homing OK
- ➔ axe en cours de déplacement

mais nous pouvons aussi le visualiser dans cette fenêtre.

Fenêtre User Memory



No	Signé 32 bits	Hexadécimal	Réel
0	55	0x00000037	7,7071416E-4
1	0	0x00000000	0
2	1	0x00000001	1,4012985E-4
3	0	0x00000000	0
4	0	0x00000000	0
5	0	0x00000000	0
6	0	0x00000000	0
7	0	0x00000000	0
8	0	0x00000000	0
9	0	0x00000000	0

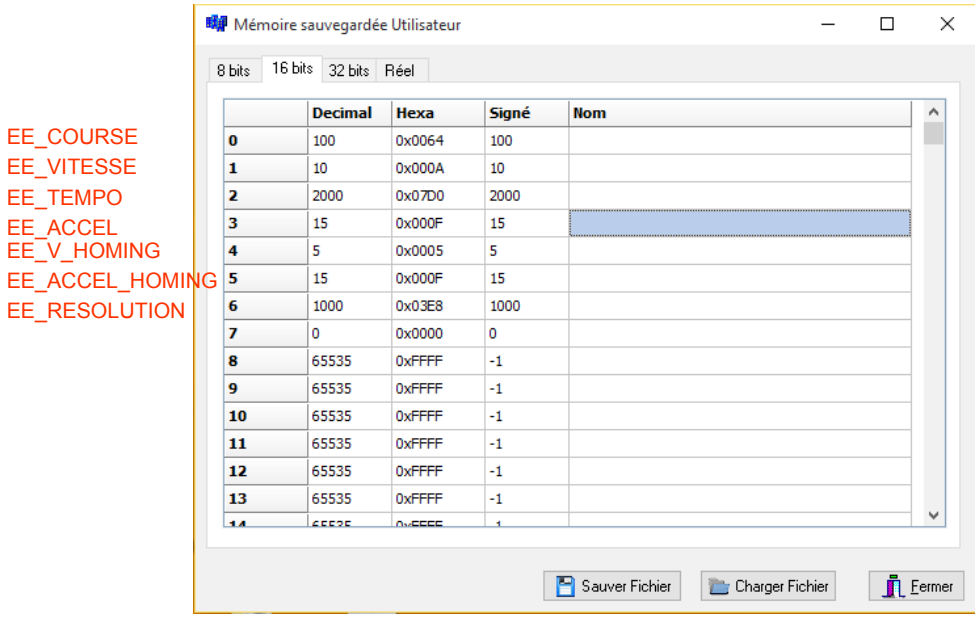
Dans le programme on trouve :

```
    SetUserMem 0, GetPos(1)/GetEEData16(EE_RESOLUTION)     'Calcul de la position en mm et on place la
valeur dans la UserMem 0
    SetUserMem 1, EtapeHoming                              ' pour suivre les etapes du homing
    SetUserMem 2, EtapeAR                                  ' pour suivre les etapes AR
```

Dans cette capture de fenêtre :

- ➔ la mémoire Numéro 0 contient 55 soit 55mm = valeur calculée
- ➔ la mémoire Numéro 1 contient 0 = on est à l'étape 0 du cycle de Homing
- ➔ la mémoire Numéro 2 contient 1 = on est à l'étape 1 du cycle AllerRetour

Fenêtre Mémoire Sauvegardée Utilisateur (EEPROM)



	Decimal	Hexa	Signé	Nom
0	100	0x0064	100	
1	10	0x000A	10	
2	2000	0x07D0	2000	
3	15	0x000F	15	
4	5	0x0005	5	
5	15	0x000F	15	
6	1000	0x03E8	1000	
7	0	0x0000	0	
8	65535	0xFFFF	-1	
9	65535	0xFFFF	-1	
10	65535	0xFFFF	-1	
11	65535	0xFFFF	-1	
12	65535	0xFFFF	-1	
13	65535	0xFFFF	-1	

Dans le programme on trouve :

```
' Paramètres
' stockes dans l'EEPROM de la carte:  0 = registre 5120
const EE_COURSE             = 0      ' (mm)
const EE_VITESSE            = 1      ' (mm/s)
const EE_TEMPO              = 2      ' (ms)
const EE_ACCEL              = 3      ' (kHz)
```

```
const EE_V_HOMING          = 4     ' (mm/s)
const EE_ACCEL_HOMING      = 5     ' (kHz)
const EE_RESOLUTION        = 6     ' (pulses/mm)
```

Puis plus loin :

```
Cible = GetEEData16(EE_COURSE)*GetEEData16(EE_RESOLUTION)
? "Start axe 1 Aller vers Cible : ", Cible, " pulses, ", GetEEData16(EE_COURSE), " mm"
MoveAxe 1, GetEEData16(EE_ACCEL), GetEEData16(EE_VITESSE)*GetEEData16(EE_RESOLUTION),
        GetEEData16(EE_ACCEL),  Cible ' lancement axe 1
```

Dans un premier temps on dit que EE_RESOLUTION est dans le registre numéro 6, ensuite on récupère la valeur du registre dans l'EEPROM grâce à GetEEData16(EE_RESOLUTION)

Dans cette fenêtre on peut lire et modifier les paramètres utilisés pour gérer les déplacements.

L'utilisation de variables stockées dans l'EEPROM présente plusieurs avantages :

- ➔ c'est plus simple que de modifier le programme à chaque fois que l'on veut modifier un paramètre
- ➔ c'est facile d'accès
- ➔ elles sont non volatiles
- ➔ en ajoutant un IHM (Interface Homme Machine) comme l'écran tactile 4,3” MT4230T, par exemple, il devient très facile de visualiser et saisir des paramètres, voire de gérer des “Recettes” de paramètres.

4 Exemples de programmes Basic associés à un écran

4.1 Homing et déplacement d'un axe

4.1.1 Cycles

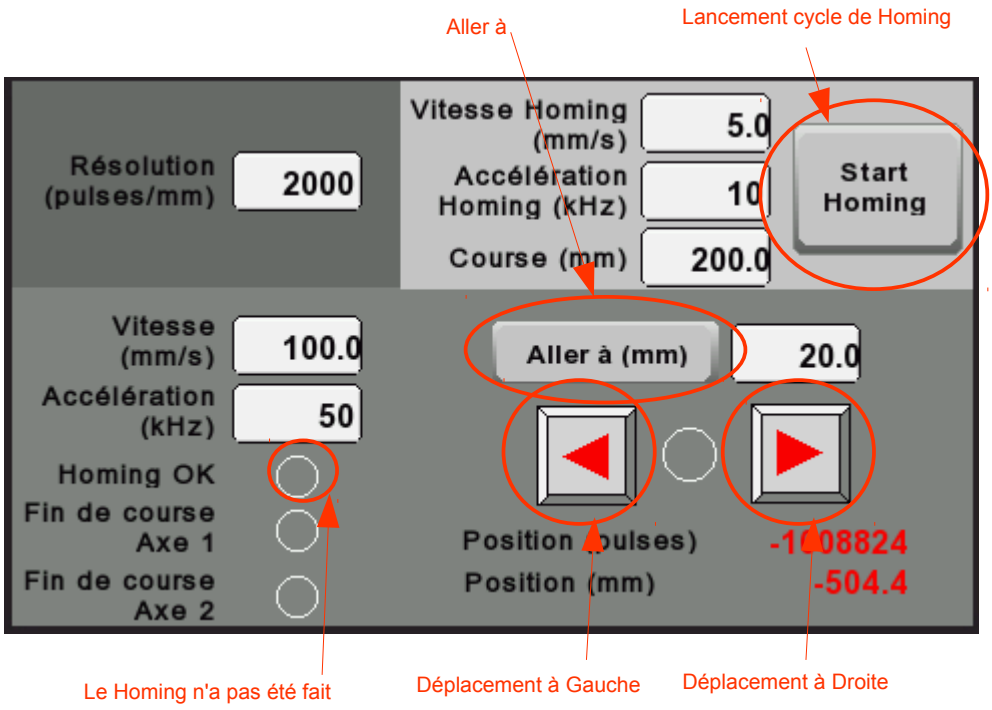
Homing : La gestion de cycle proposée permet de gérer plusieurs axes. Il suffit de renseigner les paramètres en début de cycle de Homing.

L'axe se déplace vers le capteur de fin de course, le mouvement s'arrête et s'inverse à vitesse lente (1/10 de la vitesse de Homing) jusqu'à détection de perte du capteur. L'axe s'arrête et la position de l'axe est initialisée.

Il n'est pas possible d'utiliser les boutons de déplacement tant que le Homing n'a pas été fait.

Déplacement Droite et Gauche : l'appui et le maintien du bouton droite ou gauche lance l'axe, le relachement du bouton l'arrête.

Aller à : l'appui sur le bouton envoie à la position demandée.



4.1.2 Programme Basic

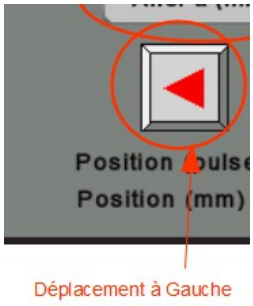
```
' Programme demo pour aller a une position cible
' V1 13/11/2015 : Version initiale
' Constantes
const INHOME_ACTIF = 0    ' Contact prise d'origine type (1=NC, 0=NO)

' Affectation des entrees
const INHOME1         = 1 ' le capteur de fin de course du homing est raccorde sur l entree 1
const INHOME2         = 2 ' capteur axe 2

' Paramètres
' stockes dans l'EEPROM de la carte:  0 = registre 5120
const EE_RESOLUTION        = 0     ' (pulses/mm)
const EE_VITESSE           = 1     ' (mm/s) appliquer facteur 10 en raison resolution champ
const EE_ACCEL             = 2     ' (kHz)
const EE_CIBLE             = 3     ' (mm)   appliquer facteur 10 en raison resolution champ
const EE_V_HOMING          = 4     ' (mm/s) appliquer facteur 10 en raison resolution champ
const EE_ACCEL_HOMING      = 5     ' (kHz)
const EE_COURSE            = 6     ' (mm)   appliquer facteur 10 en raison resolution champ

' bits cycles
const MBB_HOMING_START     = 320    ' bit cycle homing
const MBB_JOG_PLUS         = 321    ' bit cycle jog +
const MBB_JOG_MOINS        = 322    ' bit cycle jog -
const MBB_GOTO             = 323    ' bit cycle aller a position
const MBB_HOME_OK          = 330   ' bit homing OK

' Copie des définition dans des tableaux indexes (utilises dans la sequence Homing)
DIM V_HIGH(2)
DIM V_LOW(2)
DIM ACCEL(2)
DIM COURSE(2)
DIM INHOME(2)  :  INHOME(1) = INHOME1  :  INHOME(2) = INHOME2
DIM HOMING_OK(2) : HOMING_OK(1)=0 : HOMING_OK(2) = 0

DIM H(2)  :  H(1) = 0  : H(2) = 0 ' Sequence prise d'origine axe 1 et 2

Initialisation:
    ? "Initialisation"
    StopAxes 1
    Home = 0
    For i=320 To 399 : SetMBBit i, 0 : Next i
    Pause 200
    Unlock


do
  ' L'entree ENABLE a ete coupee
  if stsbit(8)=1 then goto Initialisation
      'Calcul de la position en mm et on place la valeur dans la UserMem 0
    SetUserMem 0, 10*GetPos(1)/GetEEData16(EE_RESOLUTION)
  ' Jog +
  if DFMBit(1, GetMBBit(MBB_JOG_PLUS)) and GetMBBit(MBB_HOME_OK) then       ' detection front montant du
bouton Jog + et verification que le homing est ok
    ? "Start axe 1 Jog +"
    MoveAxe 1, GetEEData16(EE_ACCEL), GetEEData16(EE_VITESSE)*GetEEData16(EE_RESOLUTION)/10,
GetEEData16(EE_ACCEL),  999999999 ' lancement axe 1
   endif
  ' Jog -
  if DFMBit(2, GetMBBit(MBB_JOG_MOINS)) and GetMBBit(MBB_HOME_OK) then       ' detection front montant
du bouton Jog - et verification que le homing est ok
    ? "Start axe 1 Jog -"
    MoveAxe 1, GetEEData16(EE_ACCEL), GetEEData16(EE_VITESSE)*GetEEData16(EE_RESOLUTION)/10,
GetEEData16(EE_ACCEL),  -999999999 ' lancement axe 1
   endif
  ' aller a
  if DFMBit(5, GetMBBit(MBB_GOTO)) and GetMBBit(MBB_HOME_OK) then       ' detection front montant du
bouton "Aller a" et verification que le homing est ok
    Cible = GetEEData16(EE_CIBLE)*GetEEData16(EE_RESOLUTION)/10
    ? "Start axe 1 vers Cible : ", Cible, " pulses"
    ? "Start axe 1 vers Cible : ", GetEEData16(EE_CIBLE)/10, " mm"
    MoveAxe 1, GetEEData16(EE_ACCEL), GetEEData16(EE_VITESSE)*GetEEData16(EE_RESOLUTION)/10,
GetEEData16(EE_ACCEL),  Cible ' lancement axe 1
```

```
  endif

  ' arret axe, commun aux 3 fonctions precedentes
  if DFDBit(3, GetMBBit(MBB_JOG_PLUS)) or DFDBit(4, GetMBBit(MBB_JOG_MOINS)) or DFDBit(6,
GetMBBit(MBB_GOTO)) and GetMBBit(MBB_HOME_OK) then
    ? "Stop axe 1"
    StopAxes 1
  endif

  if stsbit(22) then
  elseif GetMBBit(MBB_GOTO) then
    SetMBBit MBB_GOTO, 0
  endif

    '###################################
    '###  GHome : prises d'origine ######
    '###################################
  if GHome = 0 then
    if GetMBBit(MBB_HOMING_START) then
      ? "Lancement Homing"
      GHome = 10
      H(1) = 100
      H(2) = 100
      ' Vitesses, accel  et course axe 1
      V_HIGH(1) = GetEEData16(EE_V_HOMING)*GetEEData16(EE_RESOLUTION)/10
      V_LOW(1)  = V_HIGH(1)/10
      ACCEL(1)  = GetEEData16(EE_ACCEL_HOMING)
      COURSE(1) = GetEEData16(EE_COURSE)*GetEEData16(EE_RESOLUTION)/10
      ' Vitesses, accel  et course axe 2
      ' pour utiliser le 2eme axe il faudra affecter les vraies valeurs, ici on recopie celles de l axe
1
      V_HIGH(2) = V_HIGH(1)
      V_LOW(2)  = V_HIGH(1)/10
      ACCEL(2)  = ACCEL(1)
      COURSE(2) = COURSE(1)

      SetMBBit MBB_HOME_OK, 0
      SetMBBit MBB_GOTO, 0
      SetMBBit MBB_JOG_PLUS, 0
      SetMBBit MBB_JOG_MOINS, 0

    endif
  elseif GHome = 10 then
    if H(1)=0 and H(2) = 0 then  ' Séquences homing terminées
       if HOMING_OK(1) and HOMING_OK(2) then
         SetMBBit MBB_HOME_OK, 1
         SetPos 1, 0
         SetPos 2, 0
         ? "Origine terminée Succés"
       else
         ? "Origine terminée ERREUR"
       endif
       GHome = 0
       SetMBBit MBB_HOMING_START, 0
    endif
  endif  'endif GHome

  if GetOUT(31) then 'simulation Homing OK'
      Out 31,0
         SetMBBit MBB_HOMING_START, 1
         SetPos 1, 0
         SetPos 2, 0
         ? "Origine terminée Succés"
  endif

  ' ################################################
  ' #####   Sequences H : Origine Axes (Ghome)  ######
  ' ################################################

  for ii = 1 to 2 ' Pour les axes 1 et 2
        FDC = IN(INHOME(ii))
        NumAxe = ii
        if H(ii) = 0 then
            ' Etape d'attente
        elseif H(ii) = 100 then
            print "Lancement origine axe ", ii
```

```
            HOMING_OK(ii) = 0
            if FDC <> INHOME_ACTIF then   ' Capteur origine déja enclenché => dégagement
                H(ii) = 500
            else
                H(ii) = 200
            endif
        elseif H(ii) = 200 then

            MoveAxe NumAxe, ACCEL(ii), V_HIGH(ii), ACCEL(ii), GetPos(ii) - (COURSE(ii)*1.5)
            H(ii) = 210
        elseif H(ii) = 210 then
            if FDC <> INHOME_ACTIF then ' Detection capteur. Arret axe
                StopAxes (2 ^ (NumAxe-1))
                H(ii) = 220
            elseif (stsBit(22+(NumAxe-1))=0) then
                H(ii) = 0
                Print "Erreur 1 origine Axe", ii
            endif
        elseif H(ii) = 220 then            ' Attendre arret mouvement
            if (stsBit(22+(NumAxe-1))=0) then
                H(ii) = 500  ' Lancer dégagement
            endif

        ' ### Dégagement du capteur
        elseif H(ii) = 500 then
            MoveAxe NumAxe, ACCEL(ii), V_LOW(ii), ACCEL(ii), GetPos(ii) + COURSE(ii)/10
            H(ii) = 510
        elseif H(ii) = 510 then
            if FDC = INHOME_ACTIF then ' Perte capteur. Arret axe
                StopAxes (2 ^ (NumAxe-1))
                H(ii) = 520
            elseif (stsBit(22+(NumAxe-1))=0) then
                H(ii) = 0
                Print "Erreur 2 origine Axe ", ii
            endif
        elseif H(ii) = 520 then            ' Attendre arret mouvement
            if (stsBit(22+(NumAxe-1))=0) then
                HOMING_OK(ii) = 1
                print "Origine axe ", ii, " terminee"
                H(ii) = 0  ' Origine terminee
            endif

        ' ### Interruption séquence
        elseif H(ii) = 1000 then ' Etape d'arret cycle
            StopAxes (2 ^ (NumAxe-1))
            print "Arret origine axe ", ii
            H(ii) = 1100
        elseif H(ii) = 1100 then
            if stsBit(22+(NumAxe-1)) = 0 then     ' Si axe a l'arret
                H(ii) = 0
            endif
        endif
    next ii

loop
```

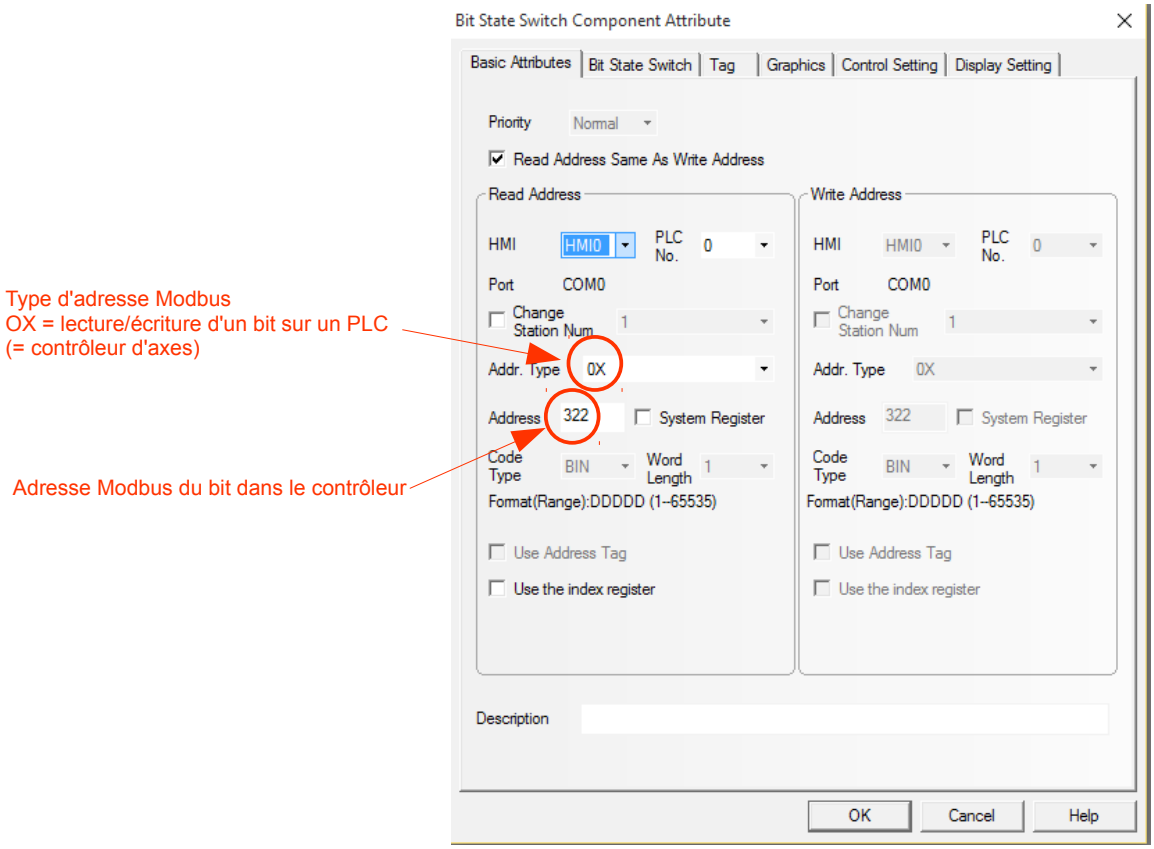
4.1.3 Paramétrage de l'écran et liens avec le programme Basic

Boutons de lancement de cycle



Les boutons (*Bit State Switch*) sont utilisés pour changer l'état des bits Modbus associés.

Le fonctionnement est le suivant :



Lorsque l'on place un *Bit State Switch* on indique le type de bit et l'adresse du bit, dans cette copie d'écran il s'agit du bit 322 de la carte.

Il a été déclaré dans le programme :

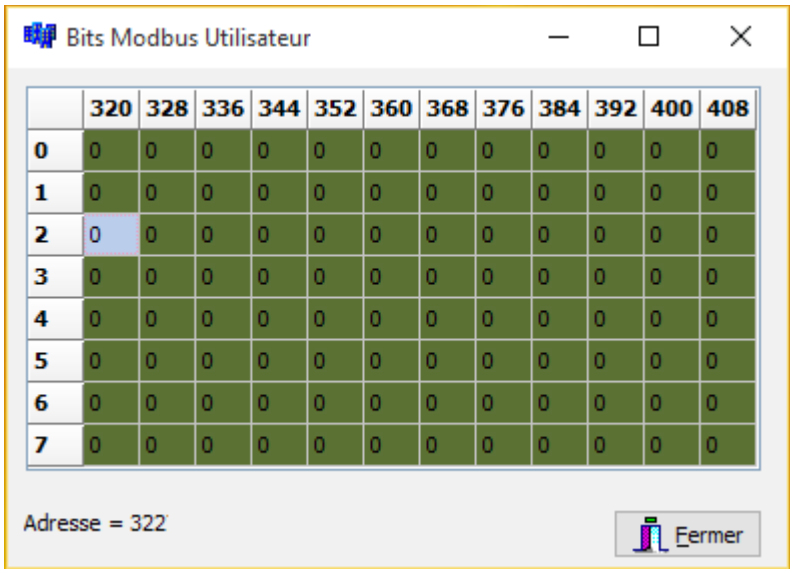
```
const MBB_JOG_MOINS        = 322    ' bit cycle jog -
```

On le retrouve dans la séquence :

```
  if DFMBit(2, GetMBBit(MBB_JOG_MOINS)) and GetMBBit(MBB_HOME_OK) then       ' detection front montant
du bouton Jog - et verification que le homing est ok
```

Pour rappel la commande DFMBit (x, bit) détecte un front montant lors du changement d'état du bit.

On retrouve aussi ce bit dans la fenêtre “Bit Modbus Utilisateur” de Test Center



Pour faciliter la compréhension des exemples associés aux différents champs vous trouverez ci-dessous une synthèse des correspondances de types de champs en fonction des adresses Modbus, pour plus de détails sur la fonction des adresses reportez vous à la documentation Mobus des cartes :

Types	**Données**	**Accès**	**Adresses**	**Commandes Basic ***
0X	Bit	Lecture Ecriture	256 à 512	`GetMBBit(adresse)` `SetMBBit adresse, valeur`
1X	Bit	Lecture seule	2048 à 2304	`GetMBBit(adresse)`
3X	WORD	Lecture seule	1040 à 1043	
4X	WORD	Lecture Ecriture	5120 à 5631	`GetEEData16 (adresse)` `SetEEData16 adresse, valeur`
4X Poids faible Poids Fort	DWORD	Lecture Ecriture	4198 à 4217	`GetUserMem (adresse)` `SetUserMem adresse, valeur`
3X-DINV Poids Fort Poids faible	DWORD	Lecture seule	1024 à 1039 1044 à 1055	
4X-DINV Poids Fort Poids faible	DWORD	Lecture/Ecriture	5120 à 5631	`GetEEData32 (adresse)` `SetEEData32 adresse, valeur`

* Les origines des adresses dans les commandes Basic peuvent être différentes des adresses Modbus :

Cas des Bits Modbus : GetMBBit(320) correspond au Bit Modbus 0X - 320

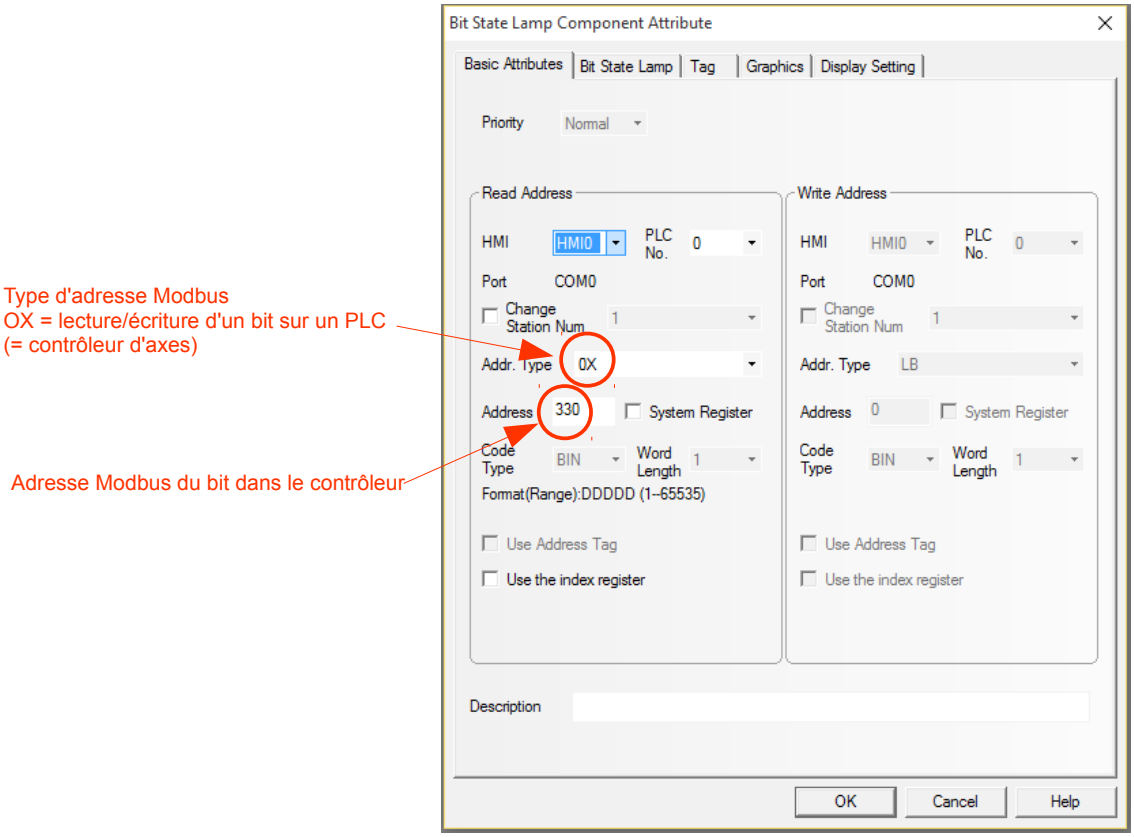
Cas des registres UserMem : GetUserMem (0) correspond aux registres 4198 et 4199 (32bits)

Cas des registres EEProm : GetEEData16 (0) correspond au registre 5120 (16bits),
GetEEData32 (0) correspond aux registres 5120 et 5121 (32bits)

Voyant d'état dun bit

Les voyants (*Bit State Lamp*) sont utilisés pour lire l'état des bits Modbus associés.

Le fonctionnement est le suivant :



Lorsque l'on place un *Bit State Lamp* on indique le type de bit et l'adresse du bit, dans cette copie d'écran il s'agit du bit 330 de la carte. Comme le bit est en lecture/écriture on peut utiliser le type d'adresse 0X (lecture/écriture) ou 1X (lecture seule).

Il a été déclaré dans le programme :

```
const MBB_HOME_OK          = 330    ' bit homing OK
```

On le retrouve dans la séquence :

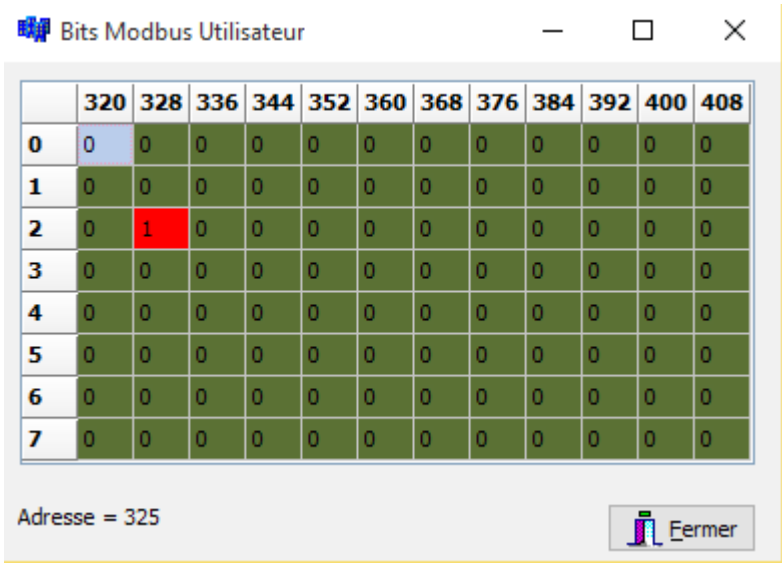
```
  if DFMBit(2, GetMBBit(MBB_JOG_MOINS)) and GetMBBit(MBB_HOME_OK) then       ' detection front montant
du bouton Jog - et verification que le homing est ok
```

ou

```
   if H(1)=0 and H(2) = 0 then  ' Séquences homing terminées
      if HOMING_OK(1) and HOMING_OK(2) then
        SetMBBit MBB_HOME_OK, 1
        SetPos 1, 0
        SetPos 2, 0
        ? "Origine terminée Succés"
```

par exemple.

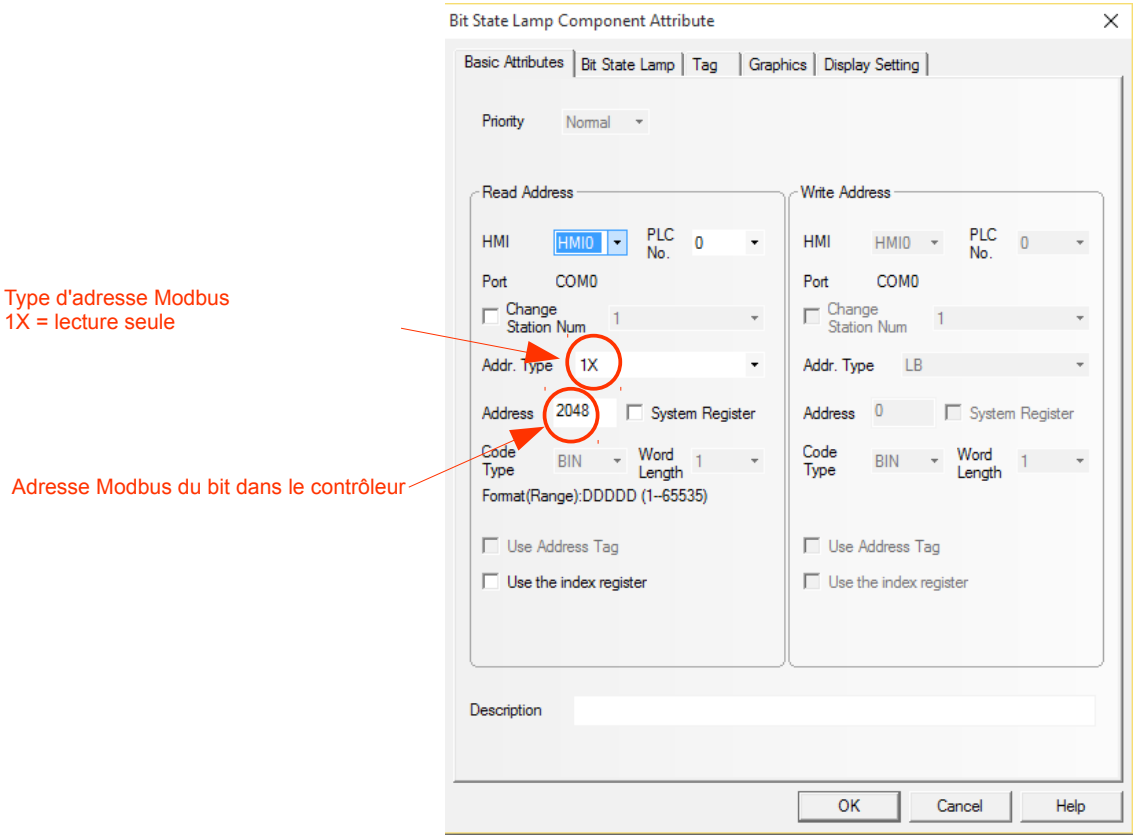
On retrouve aussi ce bit dans la fenêtre “Bit Modbus Utilisateur” de Test Center



Voyant d'état d'une entrée

Les voyants (*Bit State Lamp*) sont utilisés pour lire l'état des bits Modbus associés aux entrées.

Le fonctionnement est le suivant :



Lorsque l'on place un *Bit State Lamp* on indique le type de bit et l'adresse du bit, dans cette copie d'écran il s'agit du bit 2048 de la carte qui représente l'état de l'entrée 1.

Une entrée étant par définition en lecture seule il est impératif d'utiliser le type 1X (lecture seule), sinon le voyant ne sera pas affiché (en règle générale les objets HMIware ne fonctionne pas si on utilise le type lecture/écriture sur un bit ou un registre en lecture seule).

Elle a été déclarée dans le programme :

```
const INHOME1          = 1 ' le capteur de fin de course du homing est raccorde sur l entree 1
```

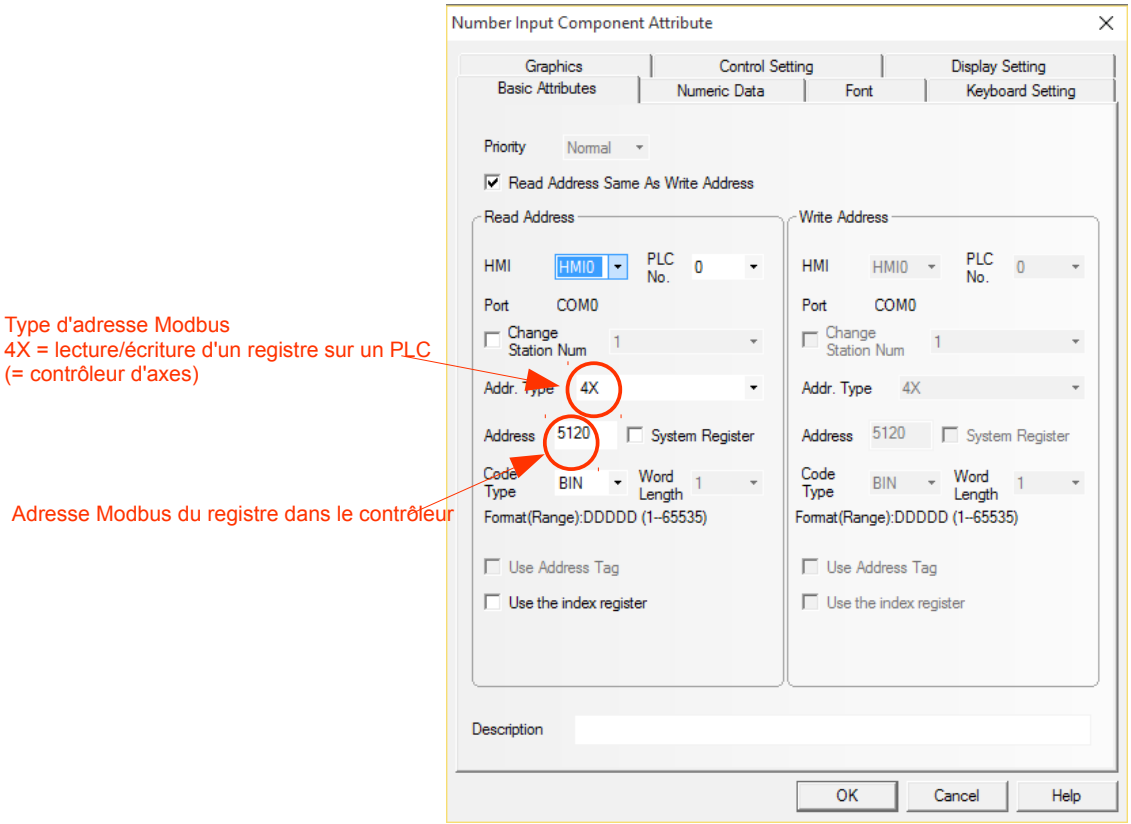
On la retrouve dans la séquence :

```
DIM INHOME(2)  :  INHOME(1) = INHOME1  :  INHOME(2) = INHOME2
```

Champ de saisie d'une valeur numérique sans décimale

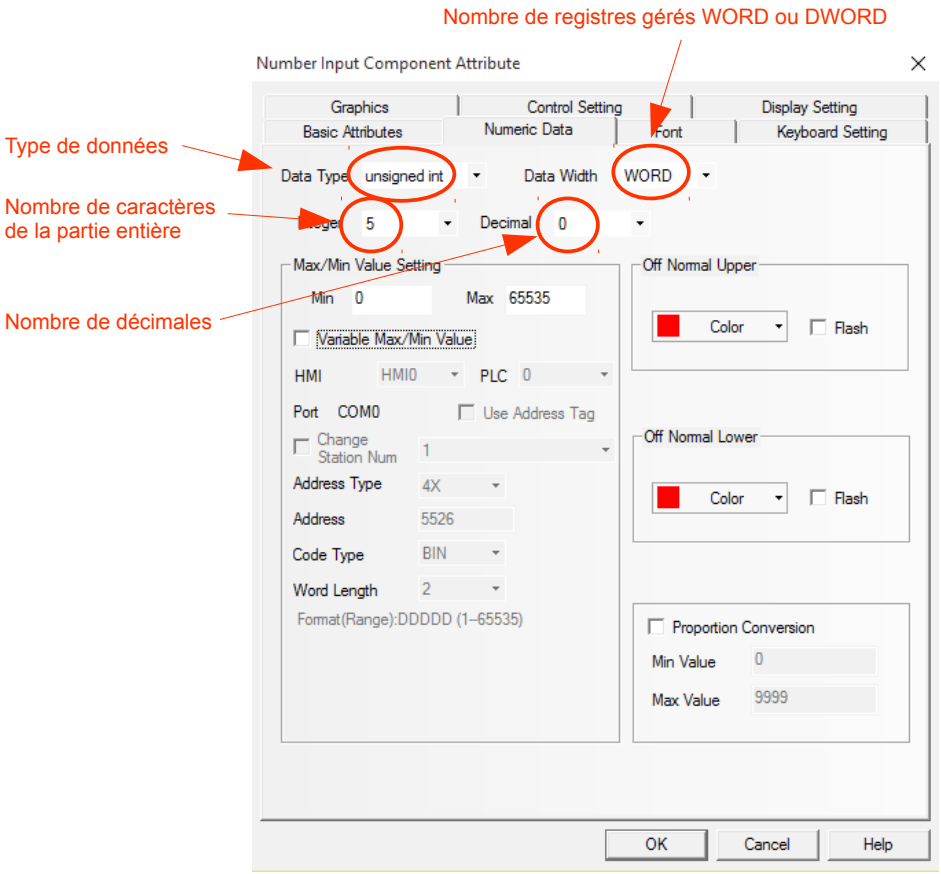
Les champs de saisie d'une valeur numérique (*Input Number*) sont utilisés pour lire et surtout écrire une valeur d'un registre Modbus.

Le fonctionnement est le suivant :



De la même façon que précédement lorsque l'on place un *"Number Input"* on indique le type de registre et l'adresse du registre, dans cette copie d'écran il s'agit du registre 5120 de la carte qui est accessible en lecture/écriture.

Il faut aussi spécifier d'autres informations :



Le type de données : signed int, unsigned int, …

Le nombre de registre WORD ou DWORD (un mot ou 2 mots), nous verrons plus loin dans la partie "*Number Display"* comment ils doivent être gérés.

Le nombre de caractères affichés pour la partie entière.

Le nombre de décimales, ici 0.

Il s'agit du champ Résolution, c'est un entier non signé sur un seul mot. Dans cet écran on peut aussi fixer les limites min et max de la saisie.

Le registre a été déclaré dans le programme :

```
const EE_RESOLUTION        = 0      ' (pulses/mm)
```

Attention le registre 0 dans le programme Basic correspond au registre 5120 en accès Modbus, et ainsi de suite : registre 1 pour 5121 en Modbus…

Il est très important de bien définir les unités avec lesquelles on travaille afin de faciliter la compréhension des formules de calculs des paramètres passés ensuite dans la commande MoveAxe.

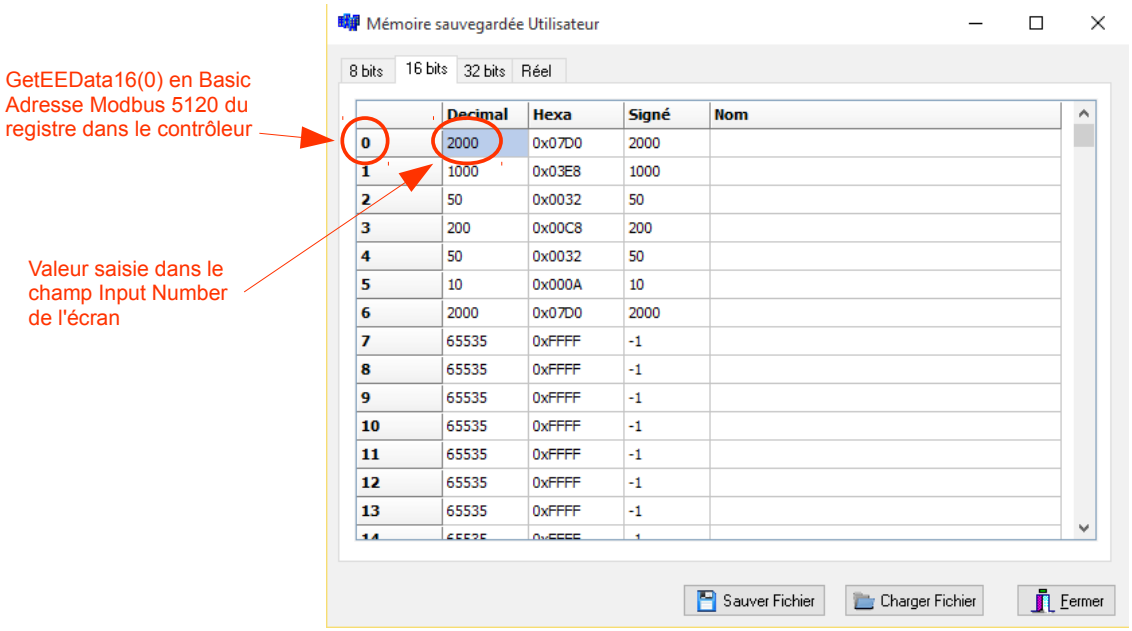
On le retrouve dans la séquence :

```
Cible = GetEEData16(EE_CIBLE)*GetEEData16(EE_RESOLUTION)/10
```

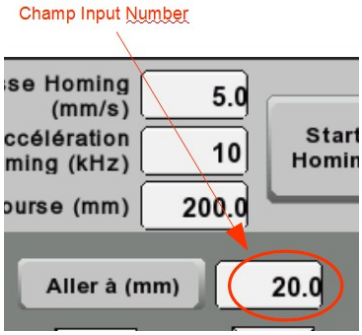
Nous expliquerons cette formule de calcul dans la description du champ Cible plus loin.

Pour rappel GetEEData16(x) signifie qu'on lit le mot de 16 bits numéro x dans l'EEPROM de la carte.

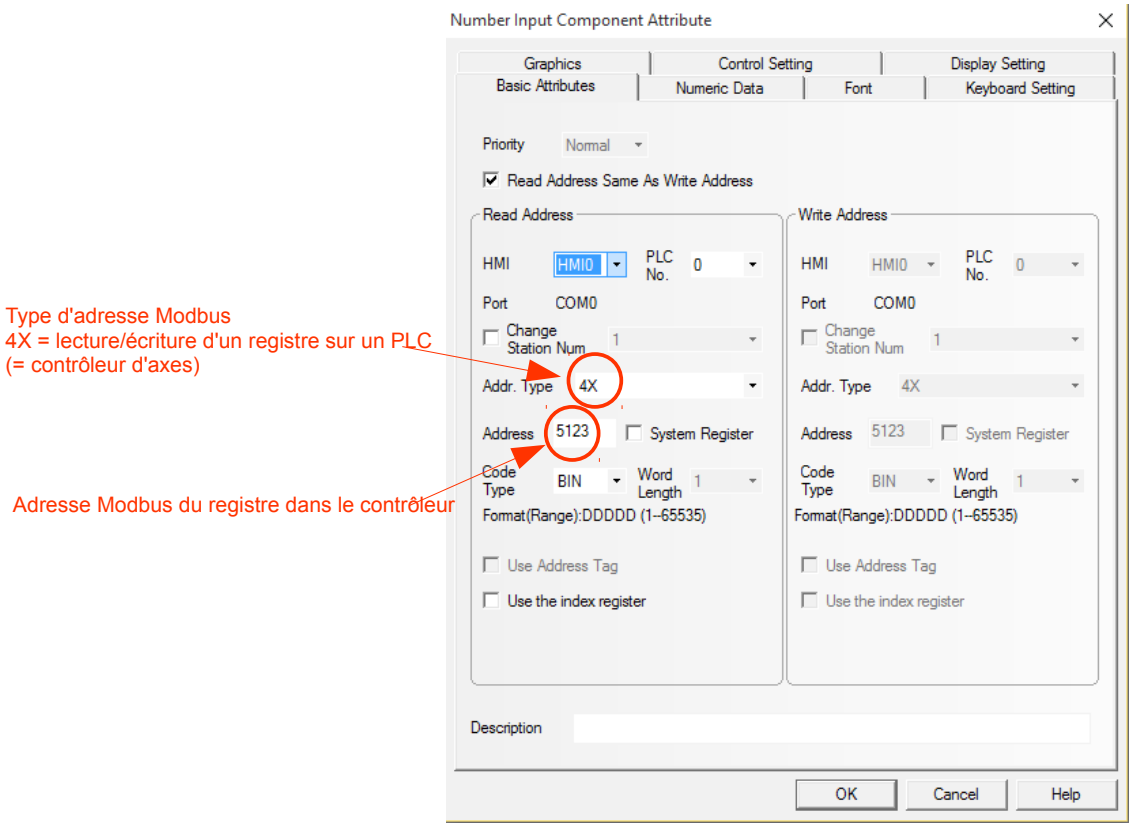
On retrouve aussi ce registre dans la fenêtre “Editeur EEPROM Utilisateur” de Test Center



Champ de saisie d'une valeur numérique avec décimale

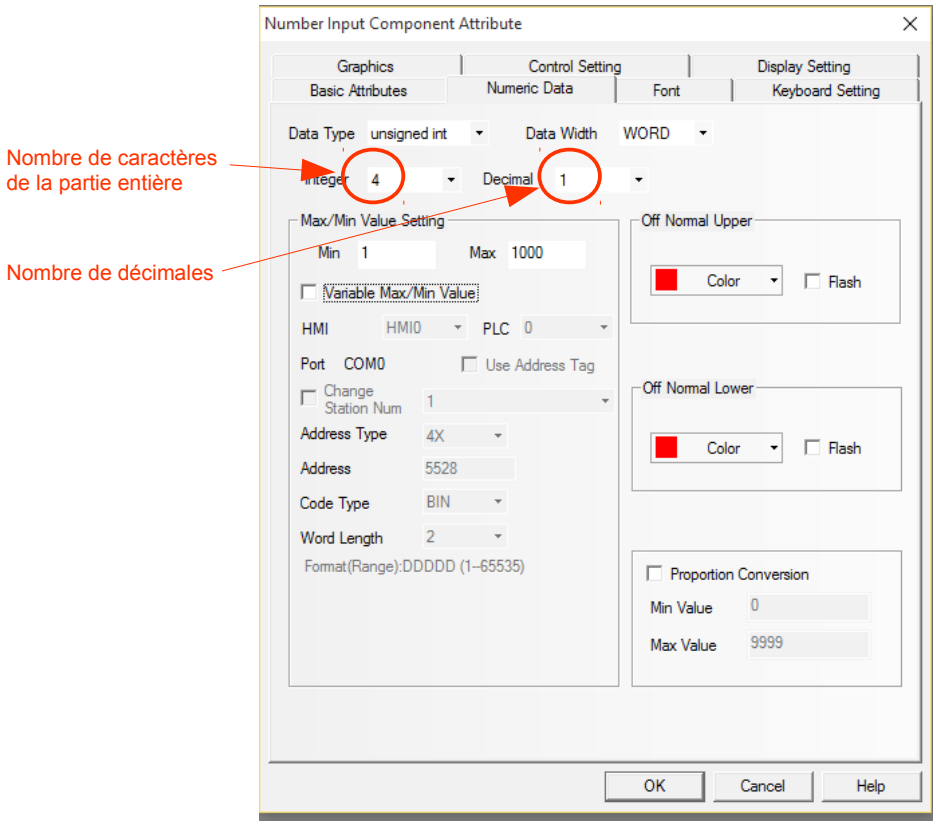


Le principe est le même que dans l'exemple précédent seule l'adresse change bien sûr:



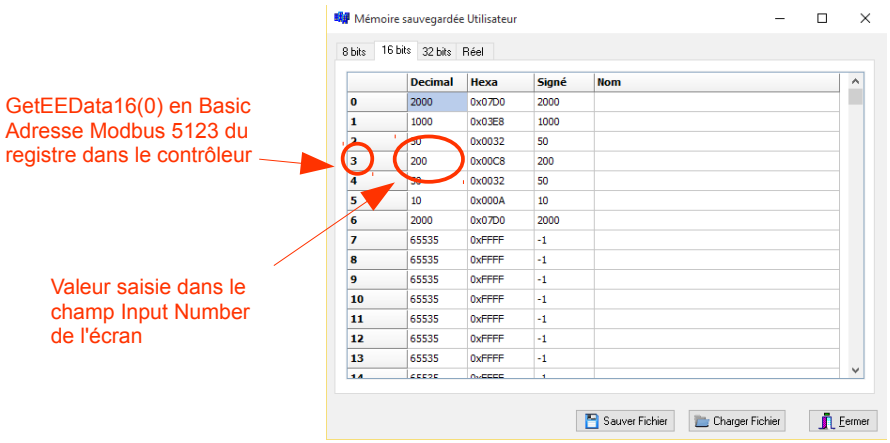
Ici il s'agit du registre 5123 de la carte qui est accessible en lecture/écriture.

Dans cet onglet, à la différence de l'exemple précédent, on spécifie une décimale :



Il s'agit du champ "Cible" qui peut être saisi en 1/10 de mm : 20.0 dans notre exemple.

C'est une facilité d'affichage dans le champ mais attention la valeur stockée dans le registre (c'est un Data Width "WORD") sera un entier égal à 200.



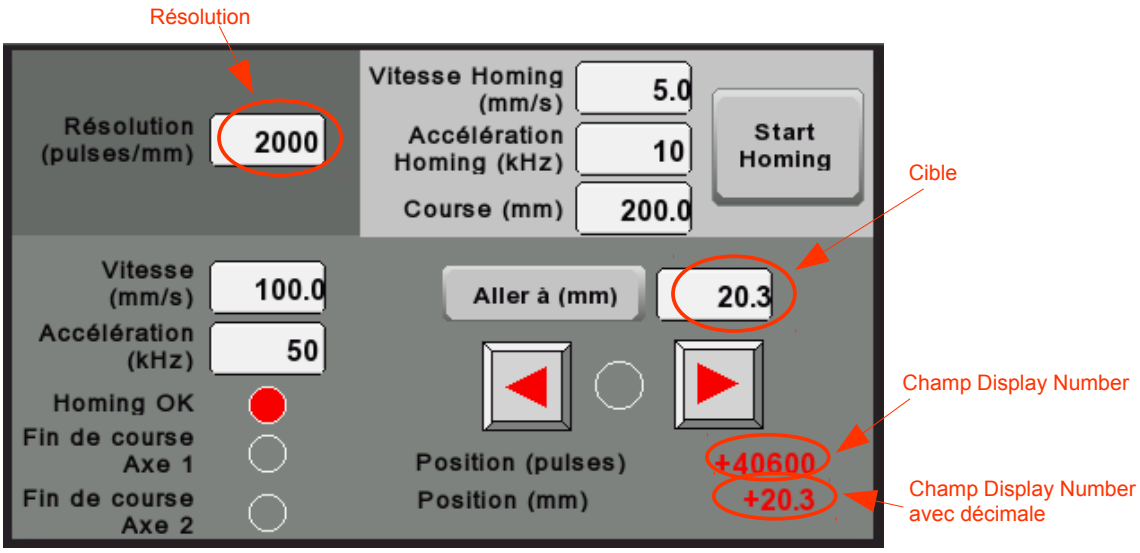
Le champ gère une virgule et sa position mais, au moment de l'utilisation de la valeur du registre dans le programme, il faut gérer un coéfficient diviseur. Dans cet exemple il faut diviser la valeur stockée par 10.

Si on reprend le code de la séquene précédente :

```
Cible = GetEEData16(EE_CIBLE)*GetEEData16(EE_RESOLUTION)/10
```

On multiplie la cible saisie en mm*10 par la résolution en pulses/mm puis on divise par 10 pour tenir compte de la décimale du champ. On trouve le nombre de pulses cibles du mouvement.

Champ d'affichage d'une valeur numérique

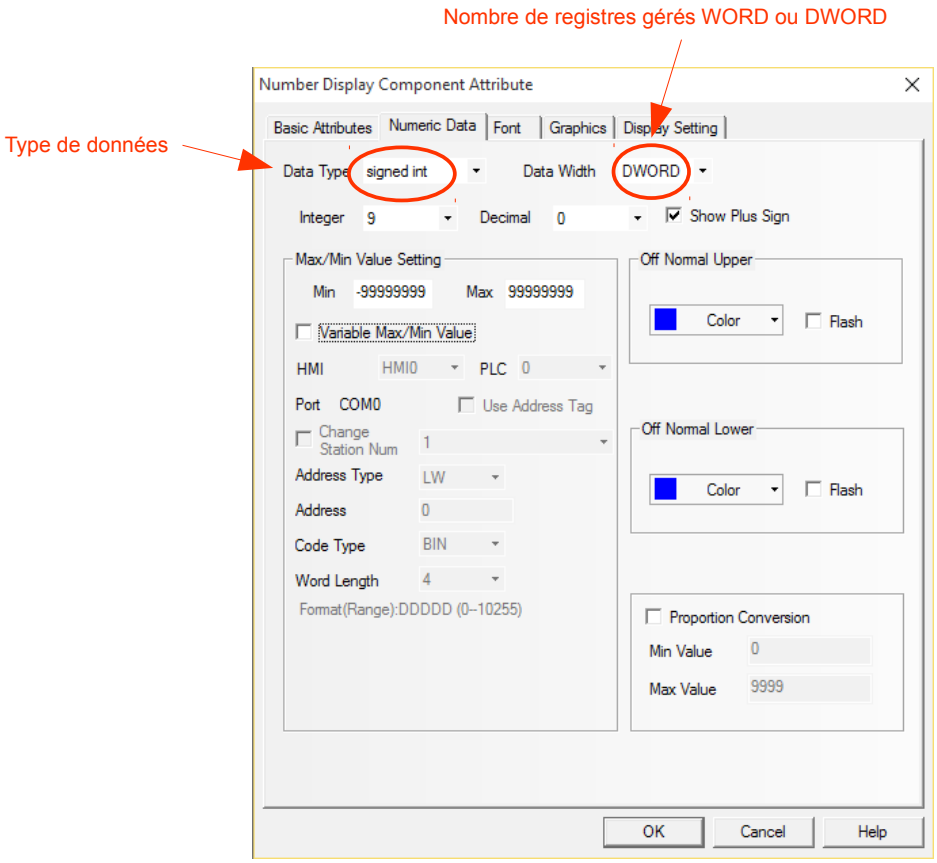


Les champs “Display Number” permettent l'affichage de valeurs numériques stockées dans des registres.

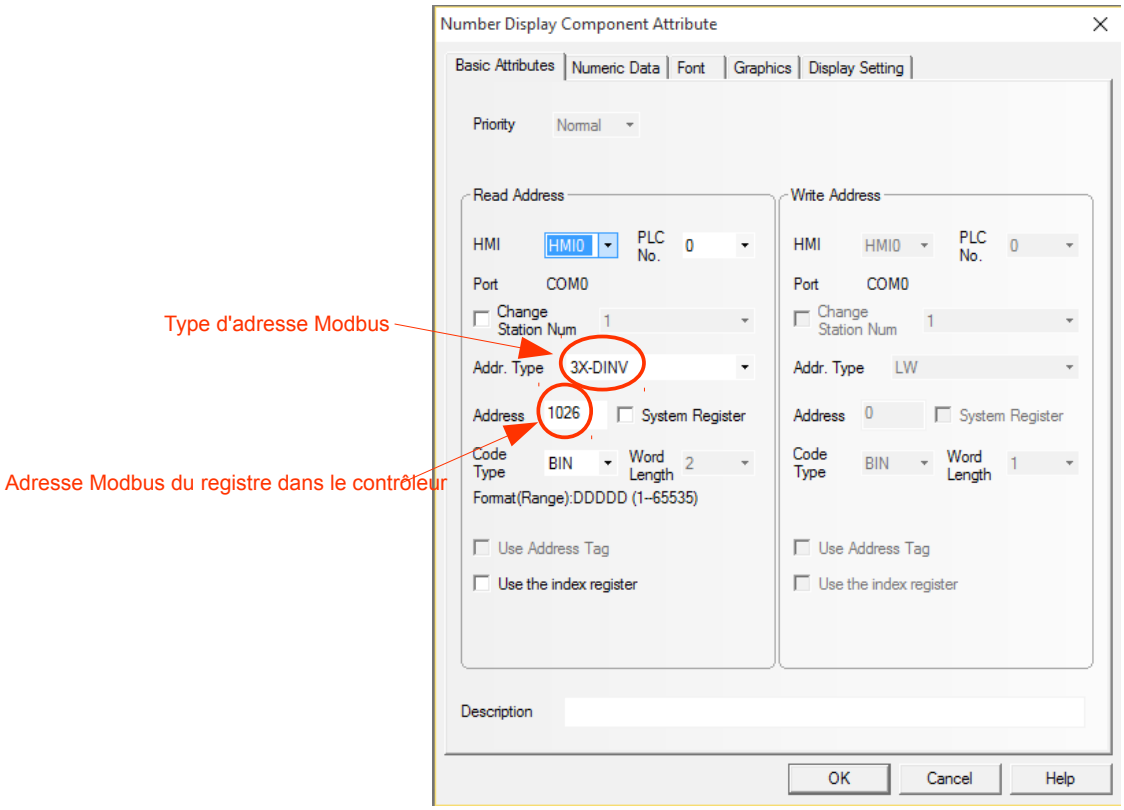
Le principe de gestion des décimales est le même que dans l'exemple précédent.

Ici chaque valeur est stockée sur 2 registres, on utilise le Data Width “DWORD”.

Cas de la position en pulses :



De plus, comme on affiche une position c'est un type “signed int” pour pouvoir afficher des positions négatives.



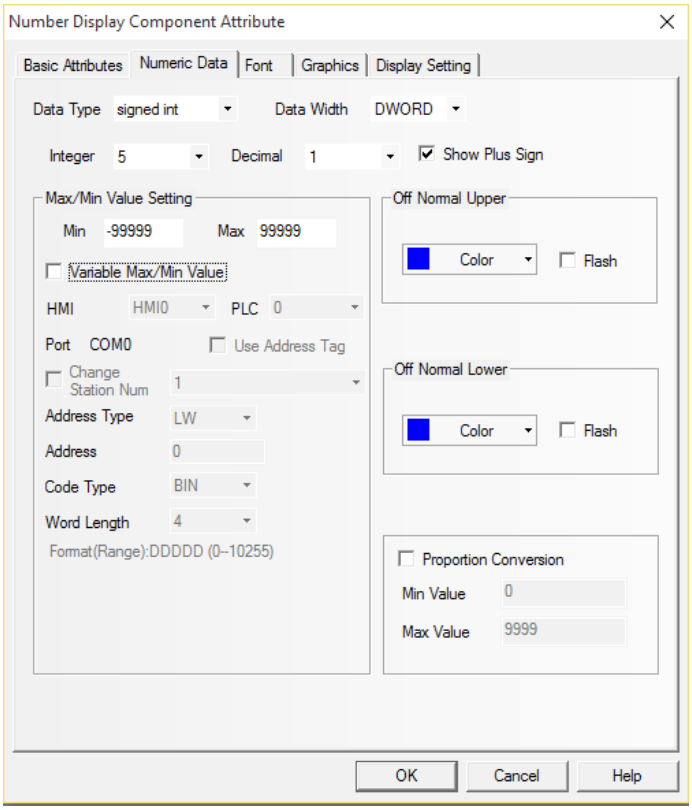
Le type d'adresse est 3X-DINV : les registres ne sont accessibles qu'en lecture et, comme il s'agit de la lecture d'un DWORD on spécifie dans quel sens on lit les registres.

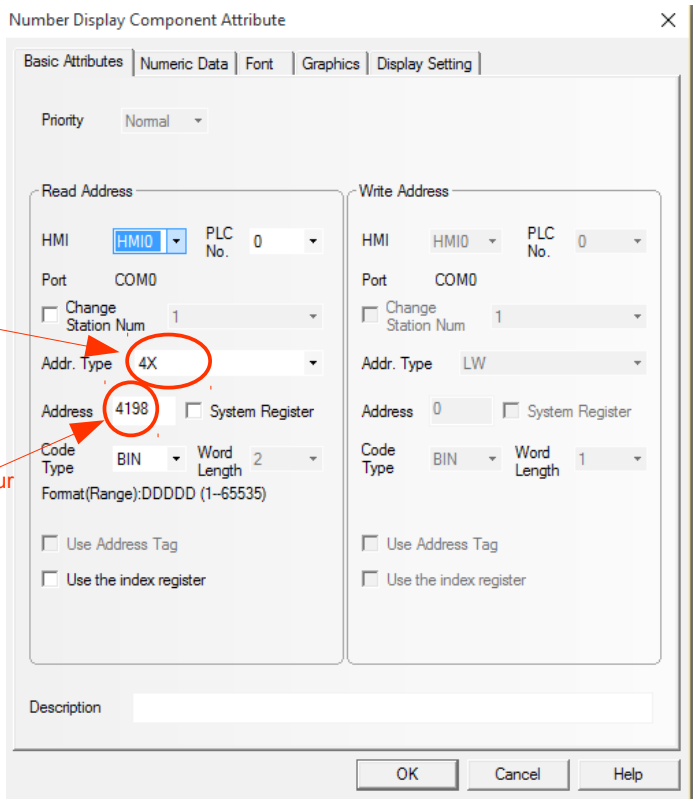
Le registre 1026 de la carte correspond au mot de poids fort de la position de l'axe X, le deuxième mot (1027) au poids faible. Le DINV de 3X-DINV permet de spécifier le sens de lecture des registres.

Pour pouvoir utiliser ce type d'adresses Modbus il faut choisir le PLC “Modbus RTU Extended” lors de l'association de l'écran avec le/les PLC.

Cette donnée est lue par la commande GetPos (1).

Cas de la position en mm :





Type d'adresse Modbus

Adresse Modbus du registre dans le contrôleur

C'est un DWORD avec une décimale.

C'est un type d'adresse Modbus 4X en mode lecture/écriture. Il n'y a pas le DINV, le sens d'accès est donc poids faible – poids fort.

Le registre 4198 correspond au registre de poids faible de la variable en RAM UserMem 0 du Basic de la carte.

Elle est calculée à chaque boucle :

```
'Calcul de la position en mm et on place la valeur dans la UserMem 0
SetUserMem 0, 10*GetPos(1)/GetEEData16(EE_RESOLUTION)
```

On récupère la position de l'axe 1 avec un GetPos (1), on la divise par la résolution et on multiplie par 10 à cause de la décimale du champ.