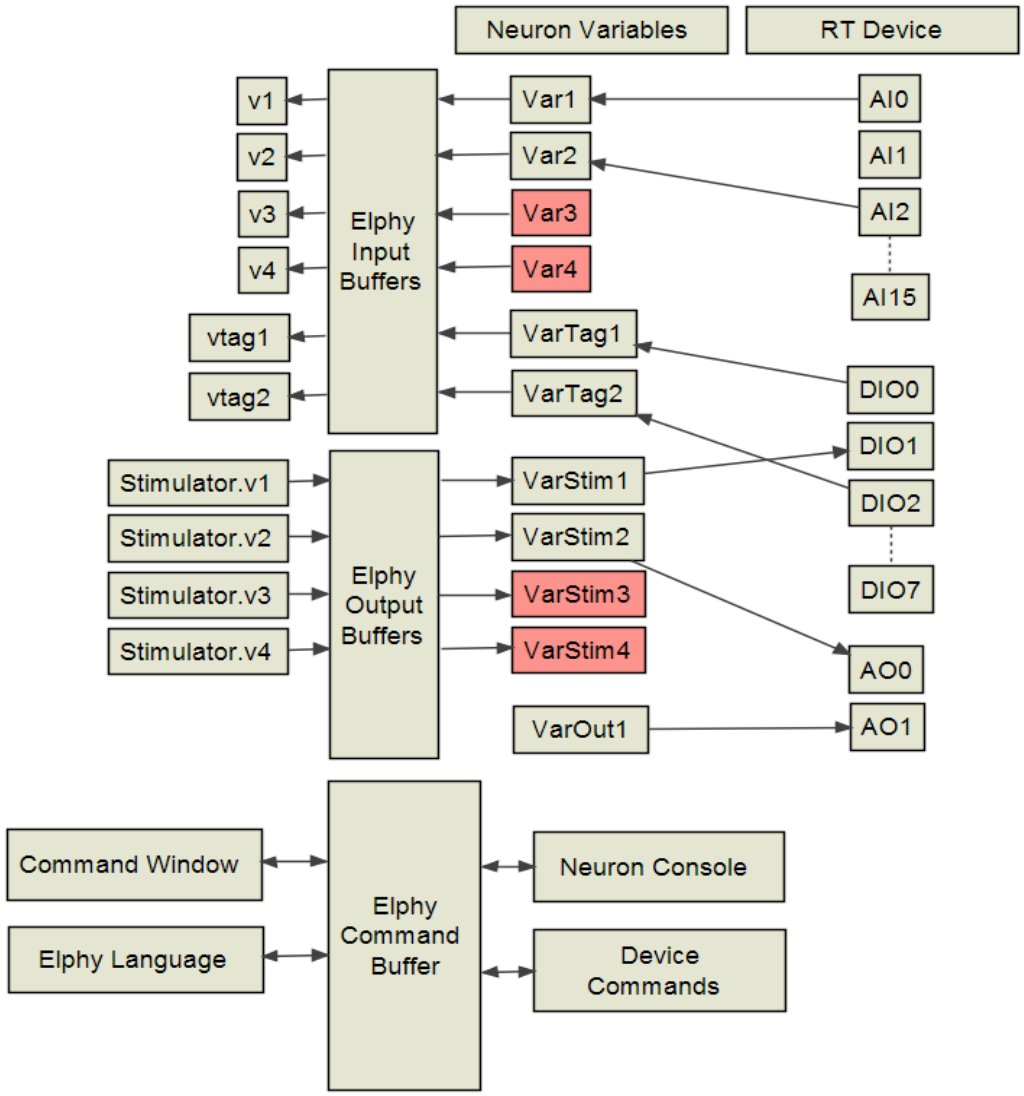
Le système temps-réel Elphy-Intime

Principe

Le diagramme suivant montre le principe de fonctionnement:



Le système est formé de deux programmes,Elphy et Rt-Neuron, qui tournent chacun dans un environnement différent, tout en communiquant via une mémoire partagée et un ensemble d'objets de synchronisation.
Tout le paramétrage se fait à partir de Elphy, mais une fois l'acquisition lancée, les deux programmes travaillent séparément.

Le programme Elphy fonctionne sous Windows. En acquisition, son rôle est de gérer les flux de données qui entrent ou sortent des buffers de la mémoire partagée. Les actions Elphy peuvent se faire avec une latence de quelques dizaines de millisecondes.

Le programme Rtneuron (une version modifiée de Neuron 6.0) fonctionne sous Intime. Son rôle est de mettre en place une routine qui sera appelée à une cadence régulière (le pas de temps) , typiquement toutes les 100 micro-secondes. Cette routine devra faire son travail en quelques dizaines de micro-secondes, sans jamais prendre de retard. En simplifiant beaucoup, les tâches de cette routine sont:

- lire les entrées de la carte d'acquisition
- appeler la routine de mise à jour du modèle Neuron.
- mettre à jour les sorties

Tout le système est construit dans le but faire coexister deux logiciels qui travaillent avec des latences complètement différentes. Le point clé est l'utilisation des buffers d'entrée et de sortie.

Quand Rtneuron veut transmettre une donnée à Elphy, il la range dans le buffer d'entrée et incrémente un compteur qui se trouve dans la mémoire partagée. En gros, Elphy vérifie les compteurs toutes les 20 mset traite alors les échantillons stockés.
Quand Elphy envoie des données à Rtneuron (pour les stimulations), il les range dans le buffer de sortie avec suffisamment d'avance (au moins 100 ms). RTneuron incrémente un compteur à chaque fois qu'il utilise une donnée de ce buffer, ce qui permet à Elphy d'ajuster le nombre d'échantillons devant être stockés.

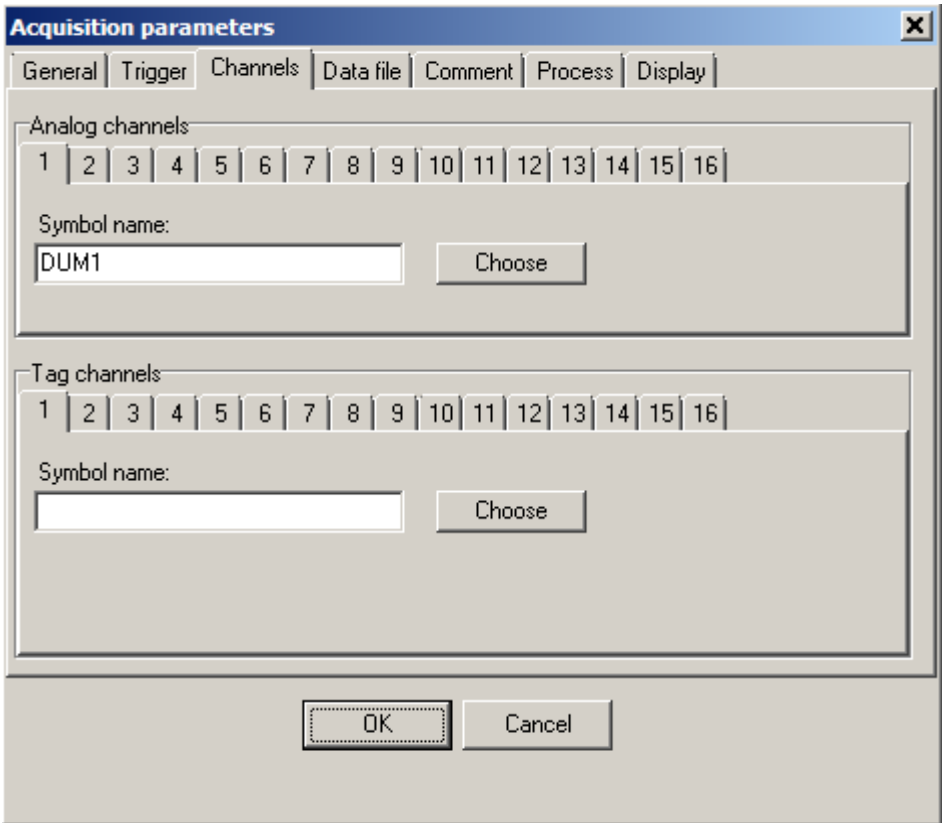
La partie centrale du diagramme montre les buffers se trouvant dans la mémoire partagée. La partie gauche contient la partie Elphy. La partie droite contient la partie Rtneuron.

Programmation du modèle Neuron

Le modèle Neuron est mis en place de la façon habituelle. En général, on charge une dll appelée nrnmech.dll. Ensuite un fichier hoc est exécuté pour initialiser le modèle.

Programmation de l'acquisition - stimulation Elphy

Elphy ne pilote pas directement la carte d'acquisition.
C'est pourquoi dans le dialogue Acquisition/Parameters/Channel, il n'est pas question de numéro physique sur la carte ou encore de paramètres d'échelle.
La seule information qui est demandée est le nom de la variable Neuron associée au canal logique.



Ce nom de variable peut être un nom de variable existant dans le modèle mis en place, mais peut aussi être un nouveau nom. Dans le premier cas, on récupèrera sur ce canal, la valeur de la variable calculée à chaque pas de temps. Dans le second cas, on crée une nouvelle variable qui pourra être utilisée de différentes façons. Cette variable n'est pas accessible par le modèle Neuron mais elle pourra servir de stockage intermédiaire pour la carte d'acquisition (voir plus loin).

On trouve la même idée pour la programmation des entrées digitalesVtag. On associe les entrées utilisées au nom d'une variable.

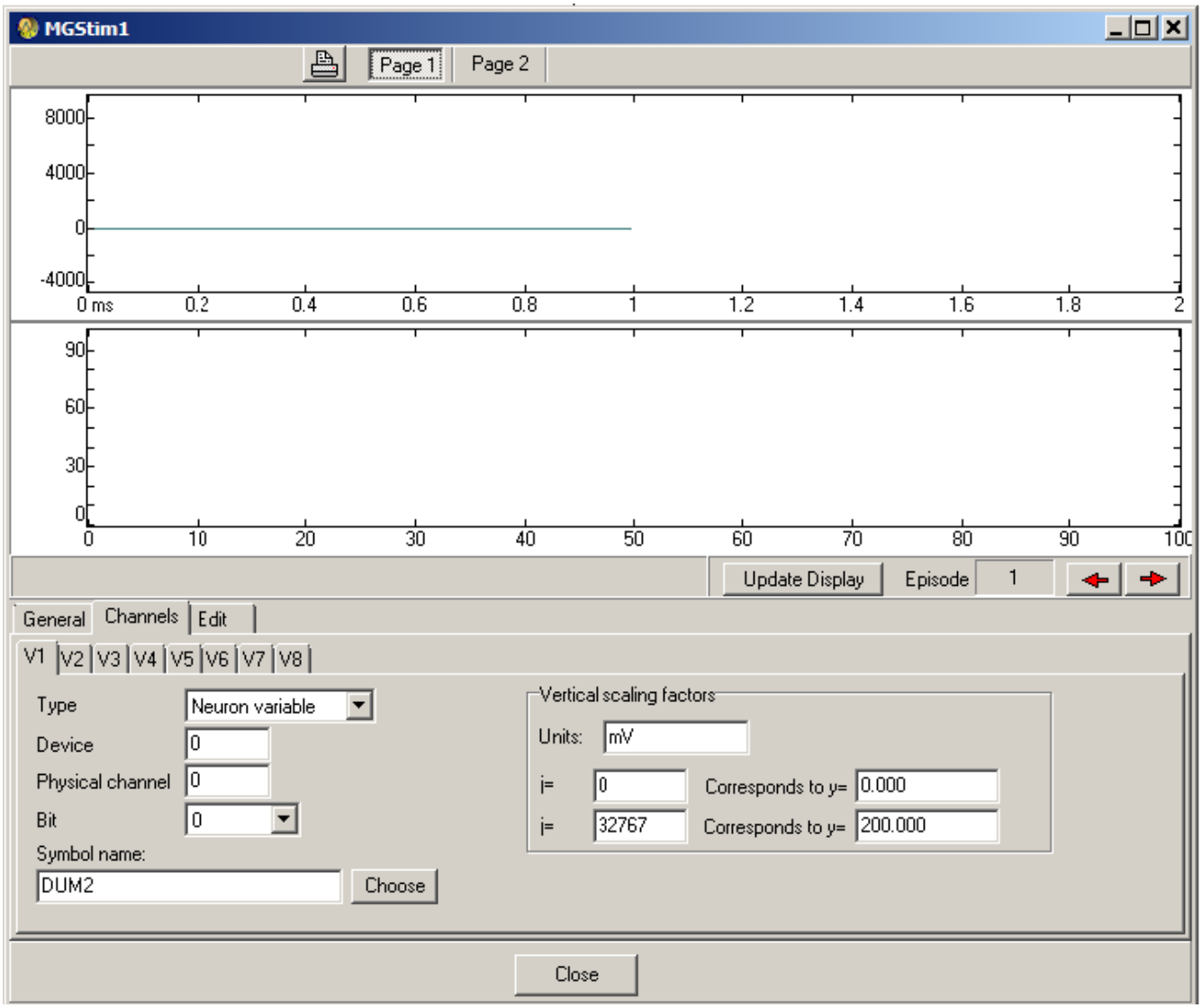
Les sorties analogiques ou digitales sont programmées dans Acquisition/Stimulation/Channels.

Dans le dialogue ci-dessous, les seuls paramètres utiles sont:

- type: obligatoirement Neuron Variable
- Symbol Name : le nom de la variable

Pour les entrées ou sorties digitales, toute valeur différente de zéro sera considérée comme le niveau logique 1.

Du point de vue Elphy, toutes les entrées et sorties analogiques sont de type réel simple précision (single). C'est à dire que les buffers de communication contiennent des réels simple précision, tout comme les fichiers de données.



Programmation du matériel sous RT-neuron

Dans le dialogue Acquisition/RT-Neuron Parameters, on programme effectivement les entrées et sorties de la carte d'acquisition. La carte utilisée actuellement est la NI-PCI-6251

Entrées analogiques

Chaque panel correspond à un numéro d'entrée physique.

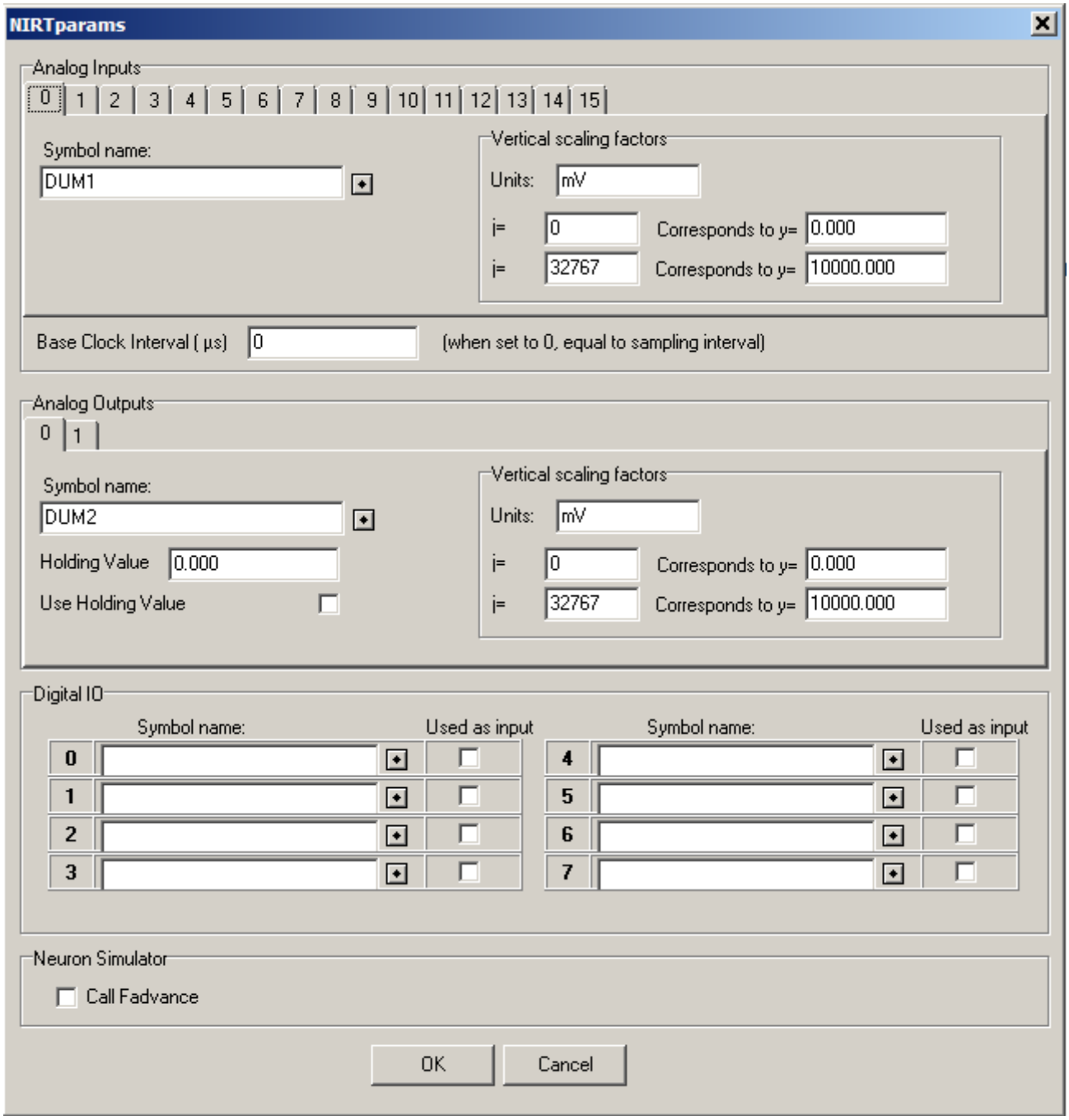
Si le numéro est associé à une variable Neuron (Symbol Name non vide), l'entrée est considérée comme active. Dans ce cas, il faut fixer les paramètres d'échelle. Ces paramètres indiquent comment les valeurs entières lues sur l'entrée seront converties pour être rangées dans la variable Neuron.

Sorties analogiques

Le principe est le même que ci-dessus: si le nom de variable est attribué, la sortie est active et il faut fixer les paramètres d'échelle.

Le paramètre Holding Value est une valeur qui sera envoyée sur la sortie à la fin d'une acquisition.

Par défaut, la valeur de la sortie reste sur la dernière valeur envoyée pendant l'acquisition, mais si Use Holding Value est coché, c'est Holding Value qui sera maintenue.



Entrées-sorties digitales

Sur la carte PCI 6251, les 8 bits du port digital 0 peuvent être considérés comme des entrées ou des sorties (cocher Use as input si nécessaire).

Si le nom de variable est attribué, le tranfert variable vers sortie digitale ou entrée digitale vers variable sera fait à chaque pas de temps. En sortie, toute valeur de variable différente de zéro donnera 1.

Activation du modèle

La routine fadvance (appelée aussi Nrn_fixed_Step dans le code Neuron) sera appelée à chaque pas de temps si la case Call Fadvance est cochée.
Il est utile de pouvoir travailler sans appeler Fadvance si on veut faire une acquisition ordinaire dans la configuration temps-réel.

Déroulement des opérations pendant un pas de temps

L'acquisition analogique est programmée en mode déclenché par le matériel. C'est à dire que, à chaque tick de l'horloge de la carte, un échantillon est prélevé sur une voie analogique et cet échantillon est rangé dans le buffer interne de la carte (4096 échantillons). Tant que le buffer ne déborde pas, il ne peut donc pas y avoir de perturbation sur l'acquisition analogique.

A chaque tick d'horloge , une interruption est également activée. Les tâches réalisées par la routine d'interruption proprement dite sont les suivantes:

- les échantillons acquis sont enlevés du buffer de la carte et sont rangés dans un buffer local beaucoup plus grand
- les entrées digitales sont également lues et rangées dans un buffer local

La routine d'interruption déclenche elle même un thread d'interruption qui réalise les opérations suivantes:

- les valeurs des entrées de la carte sont rangées à la fois dans les variables Neuron associées et dans le buffer d'entrée Elphy (ex: Var1 et Var2 , VarTag1 et VarTag2)
- les valeurs du buffer de stimulation Elphy sont rangées dans les variables Neuron (ex: VarStim1 et VarStim2)
- la routine Nrn_fixed_step de Neuron est appelée (si nécessaire)
- les valeurs calculées par Neuron sont rangées dans le buffer d'entrée Elphy(ex: Var3 et Var4)
- les valeurs calculées sont également envoyées sur les sorties de la carte (ex: VarOut1)

Le thread d'interruption n'est activé que lorsqu'un agrégat est complet, c'est à dire quand on dispose de toute les entrées correspondant à un pas de temps.

La décomposition de l'interruption en deux parties routine+thread est destinée à donner plus de sécurité au système. Si toutes les tâches étaient implémentées dans la routine d'interruption, la moindre anomalie dans les calculs (erreur ou simplement durée trop longue) pourrait être fatale au système.
La routine d'interruption ne contient que les tâches absolument indispensables, sa durée ne dépasse pas 1 ou 2 micro-secondes. Elle ne doit jamais être mise en défaut.
Le thread d'interruption contient tout le reste.

Retour sur le diagramme de principe

La partie complètement à droite représente les entrées et sorties de la carte d'acquisition. Certaines entrées et sorties sont associées à des variables Neuron (des vraies variables du modèle installé ou bien des variables supplémentaires installées par Elphy). Quand l'acquisition est lancée, toutes les données transitent par ces variables et on peut imaginer toutes sortes de combinaisons.

Sur l'exemple de diagramme, on voit qu'Elphy peut enregistrer:

- des entrées analogiques: v1 et v2 récupèrent les entrées AI0 et AI2
- des variables calculées par le modèle Neuron: v3 et v4 récupèrent les variables Var3 et Var4
- des entrées digitales : Vtag1 et Vtag2 sont associées à DIO0 et DIO2
- des variables booléennes (non représentées ici) calculées par le modèle Neuron : on pourrait stocker un résultat dans une entrée Vtag plutôt que dans une voie v[i]

Les signaux de stimulation Elphy peuvent être envoyés:

- sur des sorties analogiques : stimulator.v2 est envoyé sur AO0
- sur des sorties digitales : stimulator.v1 est envoyé sur DIO1
- sur des variables Neuron : c'est le cas de stimulator.v3 et stimulator.v4

Le résultat d'un calcul Neuron peut être envoyé directement sur une sortie de la carte sans passer par Elphy, c'est le cas de VarOut1.
Cette variable pourrait aussi être associée à une entrée logique Elphy (par exemple v5)

Dans le programme de test qui suit, on n'utilise pas Neuron, on crée simplement deux variables DUM1 et DUM2 qui permettent d'acheminer les données de Elphy vers la carte et de la carte vers Elphy.
L'idée est de générer un signal , d'en faire l'acquisition en reliant une entrée sur une sortie, et de vérifier que le signal enregistré est bien conforme à ce qui est souhaité.

Test du système

Pour détecter des anomalies dans le processus d'acquisition, nous avons implémenté le mécanisme suivant: à chaque interruption, on compte le nombre d'échantillons dans le buffer d'acquisition de la carte. Quand tout se passe bien, ce nombre est égal à 1. S'il est supérieur à 1, c'est que l'interruption a été occultée 1 ou plusieurs fois. Pour contrôler, on range dans un vecteur les temps des anomalies ,et dans un autre vecteur, on range le nombre d'interruptions masquées.
La méthode Rtneuron.getTestData permet de récupérer ces deux vecteurs.

De la même façon, on peut, à chaque activation du thread d'interruption, compter le nombre de nouveaux échantillons dans le buffer local. Un nombre d'échantillons en excès indique qu'un retard a été pris d'une façon ou d'une autre.
La méthode Rtneuron.getTestData2 permet de récupérer les résultats.

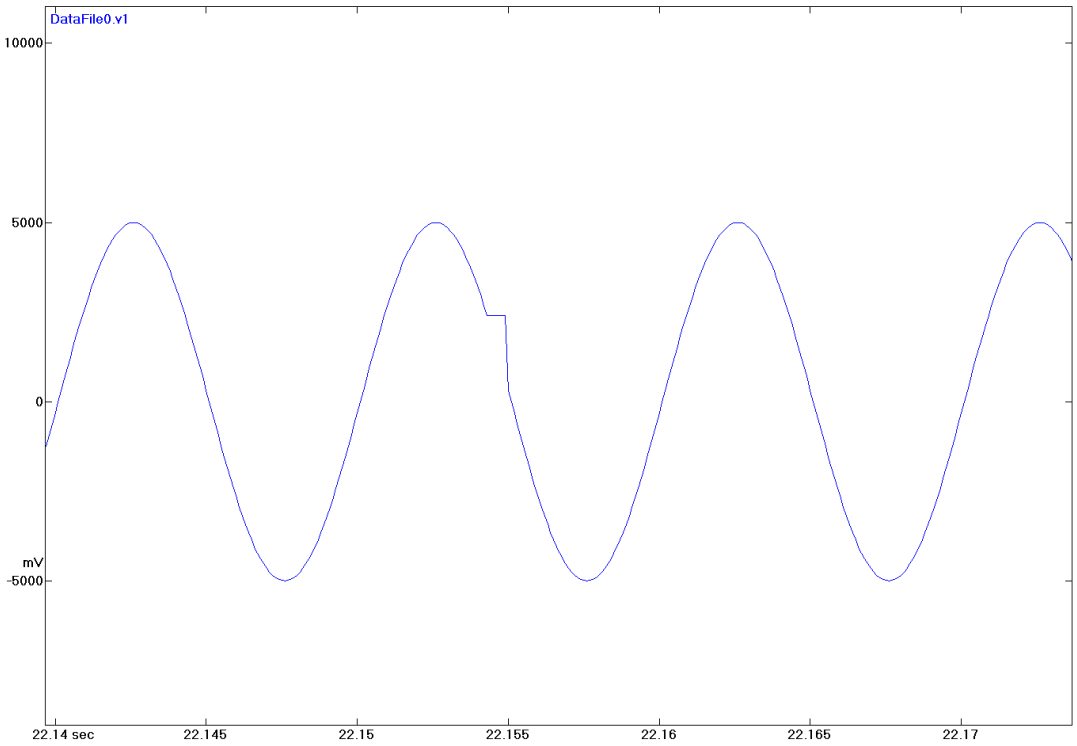
Voici, ci-dessous, les résultats d'un test sur ma machine habituelle.

- la fréquence d'échantillonnage est 10 kHz
- la stimulation est envoyée par Elphy sur une variable Neuron DUM1. Cette variable Neuron est elle-même envoyée sur la sortie analogique 0 de la carte.
- L'acquisition de v1 sur Elphy est associée à la variable Neuron DUM2. Cette variable Neuron reçoit elle même la voie analogique 0 de la carte.

- On relie la sortie 0 à l'entrée 0 au moyen d'un câble BNC
- Il n'y a aucun traitement Neuron (pas de modèle installé)

Après une acquisition d'un vingtaine de secondes, on obtient les résultats suivants:

	Time	Duration
Row 1	0.92	0.20
Row 2	1.06	0.70
Row 3	2.36	0.20
Row 4	2.56	0.20
Row 5	2.72	0.20
Row 6	2.72	0.20
Row 7	3.41	0.70
Row 8	4.56	0.20
Row 9	5.75	0.70
Row 10	6.44	0.20
Row 11	6.78	0.20
Row 12	7.24	0.40
Row 13	8.09	1.00
Row 14	10.39	0.20
Row 15	10.44	0.90
Row 16	12.78	0.70
Row 17	15.13	1.10
Row 18	15.44	0.30
Row 19	15.96	0.20
Row 20	17.47	0.70
Row 21	18.77	0.20
Row 22	19.81	0.70
Row 23	20.44	0.30
Row 24	22.16	0.70



A gauche, on voit la liste des anomalies (temps en secondes, et durée en millisecondes)
A droite, on voit la voie enregistrée. Nous avons pu vérifier que chaque anomalie de la liste correspond bien a une perturbation sur la sortie.

Nous avons agrandi la voie acquise autour de l'anomalie n°24. Comme la sortie n'a pas été rafraichie pendant environ 0.7 ms, on observe un palier correspondant à cette durée.
En moyenne, on observe une telle anomalie toutes les secondes environ.
J'ai demandé à Tenasys s'ils avaient une idée de la source de ce problème, ils m'ont expliqué que la carte mère générait une SMI (System Management Interrupt). Il s'agit d'une interruption non masquable imposée par le constructeur, il est impossible de s'en débarrasser.

Cette SMI a un comportement assez complexe. Par exemple, quand on augmente la fréquence d'échantillonnage, en passant à SampleInt= 50 µs, 20 µs ou même 10 µs, les anomalies sont de plus en plus fréquentes mais leur durée diminue!

J'ai effectué le même test sur la dernière machine proposée par Paul (carte mère ASUS Z97K), je n'observe tout simplement aucune anomalie, même avec sampleInt= 10µs

De plus, les temps de calcul sur la nouvelle machine sont environ 25% plus courts.

Programmation Elphy

Il est possible de rentrer tous les paramètres à la main dans les dialogues présentés ci-dessus mais c'est souvent fastidieux. Il est plus pratique d'écrire un programme PG2 comme dans l'exemple qui suit.
Cet exemple programme le test précédent (on ne s'intéresse qu'à l'acquisition proprement dite).

La procedure InitAcq programme l'acquisition/stimulation côté Elphy .
La procedure InitRTparams programme l'acquisition côté Rtneuron.
Ces deux procédures sont appelées dans le bloc InitProcess0.

```
procedure InitAcq;
var
  i:integer;
begin
  with Acquisition do
  begin
    Fcontinuous:=true;
    PeriodPerChannel:=  0.100;
    Fstimulate:=true;

    UseTagStart:=false;

    ChannelCount:=1;

    with Channels[1] do
    begin
      ChannelType := TI_Neuron;
      DownSamplingFactor:=1;
      unitY:='mV';
      NrnSymbolName:='DUM1';
    end;
  end;

  with Stimulator do
  begin
  //On programme une seconde de stimulation dans un buffer unique
    SetByProg:=true;
    BufferCount:=1;
    BufferSize:=round(1000/acquisition.PeriodPerChannel);
  //Nb points correspondant à 1 seconde

    ChannelCount:=1;

    with Channels[1] do
    begin
      ChannelType:= to_Neuron;
      unitY:='mV';
      NrnSymbolName:='DUM2';
```

```
    end;

    initVectors;
   // on remplit le buffer avec une sinusoïde de période 10 ms
    vector[1].sinewave(5000,0.010,0);
  end;
end;

procedure InitRTparams;
begin
  with RTneuron do
  begin
    resetParams;
    AdcChan[0].setScale(0,32767,0,10000);
    AdcChan[0].NrnSymbolName:='DUM1';

    DacChan[0].setScale(0,32767,0,10000);
    DacChan[0].NrnSymbolName:='DUM2';

    FadvanceON:=false;
  end;
end;

InitProcess0
InitAcq;
InitRTparams;
```