# GtiCommX.dll
# v1.8
# 24-10-2006

# DLL de Communication


# Orphy GTI – 2 usb

**Modification en Vert depuis la version précédente**

# SOMMAIRE

# ARCHITECTURE

Le GTI-2 est composé de :

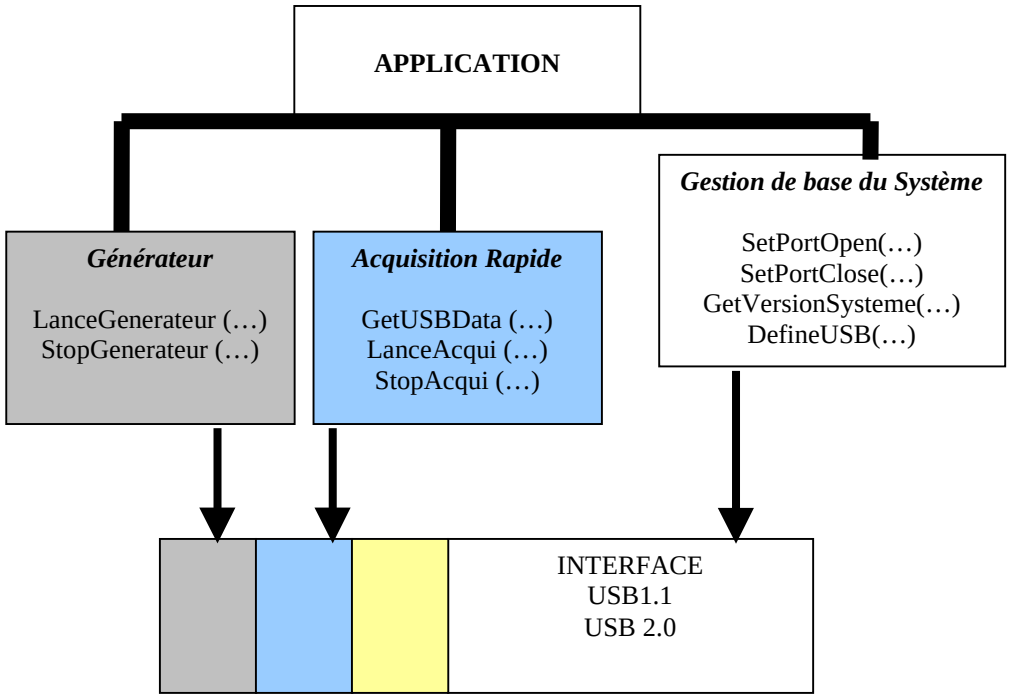
- Un système d'acquisition Lent.
- Un système d'acquisition Rapide.
- Un Générateur de Fonction.

Le jeu de commande pour la voie lente est basé sur le protocole GTS2 (cf : xxxxxxxxx.doc). Une liaison série émulée permet de faire la liaison.

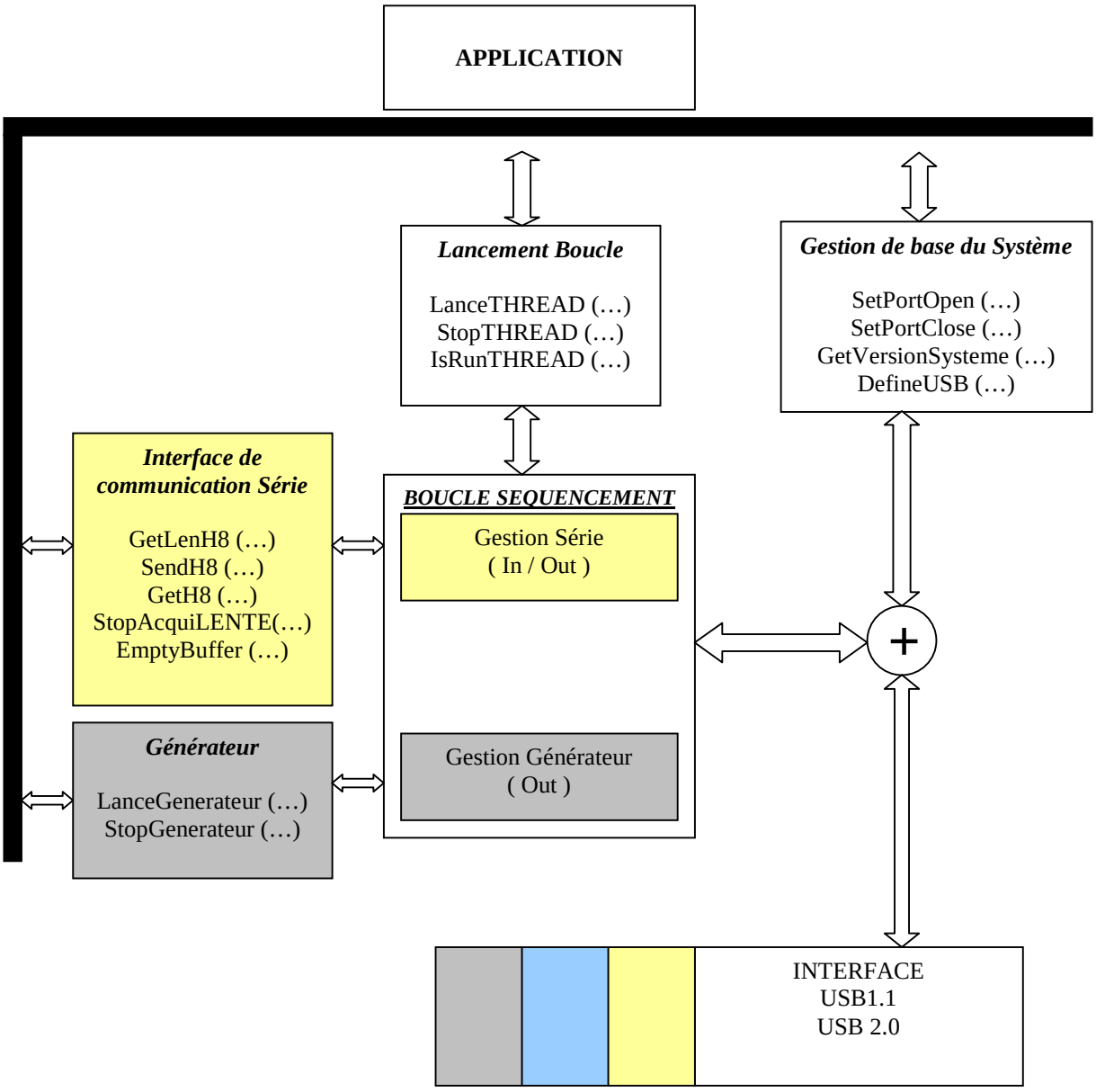
On a deux modes de fonctionnement bien distincts :

MODE 1 : Système d'acquisition Rapide et / ou Générateur de Fonction.
MODE 2 : Système d'acquisition Lent et / ou Générateur de Fonction.

### MODE 1 : Système d'acquisition Rapide et / ou Générateur de Fonction



**MODE 2 : Système d'acquisition Lent et / ou Générateur de Fonction**



# FONCTIONS

# FONCTIONS D'OUVERTURE

**SetPortOpen**

- *Appel en _stdcall*

**handle SetPortOpen**
(
BSTR strDeviceName // Nom du système à ouvrir.
)

**_ SetPortOpen**

- *Appel en __declspec(dllimport)*

**handle _SetPortOpen**
(
BSTR strDeviceName // Nom du système à ouvrir.
)

Fonction :
Cette fonction permet de créer une instance de communication USB vers le périphérique.

Paramètres :
- StrDeviceName : Nom du système à ouvrir. C'est une chaîne de caractères (UNICODE).

Valeur retournée :
Si elle est non nulle, c'est que l'ouverture du périphérique c'est bien passée.

**REMARQUE**

Le Nom du Système est nommé «OrphyGTI2- » suivi du numéro de connexion (0 à 8).
Ce numéro représente l'indice du système connecté :

Exemple :
Si l'on connecte un GTI2, le numéro sera 0
donc : strDeviceName = ''OrphyGTI2-0''

cependant il peut arriver que sous certains systèmes d'exploitation, Windows détecte mal la déconnections / reconnections de l'interface. Il peut parfois être obligatoire de passer à une valeur supérieures ''OrphyGTI2-1'' par exemple pour pouvoir communiquer avec elle.

**A noter : Cela arrive parfois lorsque l'on connecte l'interface sur un port USB puis on le reconnecte sur un autre port**.

## RECUPERATION DU TYPE DE TRANSFERT

**DefineUSB** • *Appel en _stdcall*

**long DefineUSB**
**(**
HANDLE hHandle // handle du périphérique à fermer.
)

**_DefineUSB** • *Appel en _stdcall*

**long _DefineUSB**
**(**
HANDLE hHandle // handle du périphérique à fermer.
)

Fonction :
Cette fonction permet de récupérer de connaître le type de connexion USB.

Paramètres :
- HANDLE: Handle du système. Elle est donnée par la fonction _SetPortOpen.
-
- Valeur retournée :
  1 USB1.1 - valeurs d'acquisition retournée sur 64 octets ….
  2 USB2.0 - valeurs d'acquisition retournée sur 512 octets ….

## RECUPERATION DE LA VERSION DU SYSTEME

**GetVersionSysteme** • *Appel en _stdcall*

**long GetVersionSysteme**
**(**
HANDLE hHandle // handle du périphérique
,char *strVersion // Adresse de la chaîne de retour ;
)

**_ GetVersion** • *Appel en __declspec(dllimport)*

**long _ GetVersion**
(
HANDLE hHandle // handle du périphérique à fermer.
,char *strVersion // Adresse de la chaîne de retour ;
)

Fonction :
Cette fonction permet de récupérer la version du Firmware implanté dans le système.

Paramètres :
- HANDLE: Handle du système. Elle est donnée par la fonction _SetPortOpen.
- StrVersion : Pointeur vers une chaîne de caractère ( la version y sera stockée)

Valeur retournée :
Si 1 alors la fonction c'est bien passée sinon, erreur de communication.

## FONCTIONS DE FERMETURE

**SetPortClose**

- *Appel en _stdcall*

**Long SetPortClose**
(
HANDLE hHandle // handle du périphérique à fermer.
)

**_SetPortClose**

- *Appel en __declspec(dllimport)*

**Long _SetPortClose**
(
HANDLE hHandle // handle du périphérique à fermer.
)

Fonction :
Cette fonction permet de fermer le périphérique.

Paramètres :
- HANDLE: Handle du système. Elle est donnée par la fonction _SetPortOpen.

Valeur retournée :
La valeur retournée est toujours 1 .

## LANCEMENT DU PROCESS

**LanceTHREAD**

- *Appel en _stdcall*

**Long LanceTHREAD**
(
HANDLE hHandle // handle du périphérique.
)

**_ LanceTHREAD**

- *Appel en __declspec(dllimport)*

**Long _ LanceTHREAD**
(
HANDLE hHandle // handle du périphérique.
)

Paramètres :
- Hhandle : Handle du système. Elle est donnée par la fonction SetPortOpen.

Valeur retournée :
Si 0 : soit la fonction a détecté une erreur de Communication, soit aucune trame n'a été reçue.
Si >0 : Renvoi l'adresse du process ainsi lancé. (ne sert pas ! )

Fonction :
Cette fonction permet de lancer la boucle de sequencement des différentes commandes USB.

## ARRET DU PROCESS

**StopTHREAD**

- *Appel en _stdcall*

**Long StopTHREAD**
(
)

**_ StopTHREAD**

- *Appel en __declspec(dllimport)*

**Long _ StopTHREAD**
(
)

Fonction :
Cette fonction permet d'arrêter la boucle de sequencement des différentes commandes USB.

## ETAT DU PROCESS

**IsRunTHREAD**

- *Appel en _stdcall*

**Long IsRunTHREAD**
(
)

**_ IsRunTHREAD**

- *Appel en __declspec(dllimport)*

**Long _ IsRunTHREAD**
(
)

Valeur retournée :
1 si une boucle de séquencement est en cours ; 0 le cas échéant.

Fonction :
Cette fonction permet de verifier la boucle de sequencement des différentes commandes USB.

## LANCEMENT DU GENERATEUR

**LanceGenerateur** • *Appel en _stdcall*

**Long LanceGenerateur**
(
HANDLE hHandle // handle du périphérique.
, UCHAR Loop //
, DATA_GENERATEUR *PTransfer //
)

**_LanceGenerateur** • *Appel en __declspec(dllimport)*

**Long _LanceGenerateur**
(
HANDLE hHandle // handle du périphérique.
, UCHAR Loop //
, DATA_GENERATEUR *PTransfer //
)

Paramètres :
- hHandle: Handle du système. Elle est donnée par la fonction SetPortOpen.
- Loop : si Loop = 0 alors lancement en mode « mono coup » sinon lancement en boucle.
- PTransfer : Pointeur vers la structure de Communication de type DATA_ACQUISITION.

Valeur retournée :
Si 0 alors la fonction c'est bien passée sinon, erreur de communication.

Fonction :
Cette fonction permet de déclencher le générateur.

## ARRET DU GENERATEUR

**StopGenerateur** • *Appel en _stdcall*

**Long StopGenerateur**
(
HANDLE hHandle // handle du périphérique.
)

**_ StopGenerateur** • *Appel en __declspec(dllimport)*

**Long _ StopGenerateur**
(
HANDLE hHandle // handle du périphérique.
)

Paramètres :
- hHandle: Handle du système. Elle est donnée par la fonction SetPortOpen.

Fonction :
Cette fonction permet d'arrêter le générateur si celui-ci etait configuré en mode « boucle ».

## LANCEMENT DE L'ACQUISITION

**LanceAcqui** • *Appel en _stdcall*

```
Long LanceAcqui
        (
        HANDLE hHandle                  // handle du périphérique.
        , DATA_ACQUISITION  *PTransfer  //
        )
```

**_LanceAcqui** • *Appel en __declspec(dllimport)*

```
Long _LanceAcqui
        (
        HANDLE hHandle                  // handle du périphérique.
        , DATA_ACQUISITION  *PTransfer  //
        )
```

Paramètres :

- hHandle: Handle du système. Elle est donnée par la fonction SetPortOpen.
- PTransfer : Pointeur vers la structure de Communication de type DATA_ACQUISITION.

Valeur retournée :
Si 1 alors la fonction c'est bien passée sinon, erreur de communication.

Fonction :
Cette fonction permet de déclencher le début de l'acquisition.

## LANCEMENT DE L'ACQUISITION AVEC PRETRIG

**LanceAcquiManuel** • *Appel en _stdcall*

```
Long LanceAcquiManuel
        (
        HANDLE hHandle                  // handle du périphérique.
        , DATA_ACQUISITION  *PTransfer  //
        , long CLK
        , long BT_Pretrig
        )
```

**_LanceAcquiManuel** • *Appel en __declspec(dllimport)*

```
Long _LanceAcquiManuel
        (
        HANDLE hHandle                  // handle du périphérique.
        , DATA_ACQUISITION  *PTransfer  //
        , long CLK
        , long BT_Pretrig
        )
```

Paramètres :

- hHandle: Handle du système. Elle est donnée par la fonction SetPortOpen.
- PTransfer : Pointeur vers la structure de Communication de type DATA_ACQUISITION.
- CLK : Reload du PRETRIG (0 à 255)
- BT_Pretrig : Base de Temps du PRETRIG ( 0 => 12MHz - 1 => 4MHz )

Valeur retournée :
Si 1 alors la fonction c'est bien passée sinon, erreur de communication.

Fonction :
Cette fonction permet de déclencher le début de l'acquisition en configurant manuellement le calcul du pretrig

## ARRET DE L'ACQUISITION

**StopAcqui** • *Appel en _stdcall*

**Long StopAcqui**
(
HANDLE hHandle // handle du périphérique.
)

**_StopAcqui** • *Appel en __declspec(dllimport)*

**Long _StopAcqui**
(
HANDLE hHandle // handle du périphérique.
)

Paramètres :
- HANDLE: Handle du système. Elle est donnée par la fonction SetPortOpen.

Valeur retournée :
Si 1 alors la fonction c'est bien passée sinon, erreur de communication.

Fonction :
Cette fonction permet d'arrêter l'acquisition en cours.

## RECEPTION DE DONNEES

**GetUSBData** • *Appel en _stdcall*

**Long GetUSBData**
(
HANDLE hHandle // handle du périphérique.
, DATA_GTI *pTransfer // Paramètre de communication
, DATA_ACQUISITION *pTransfer2 // Type de réception.
)

**_GetUSBData** • *Appel en __declspec(dllimport)*

**Long _GetUSBData**
(
HANDLE hHandle // handle du périphérique.
, DATA_GTI *pTransfer // Paramètre de communication
, DATA_ACQUISITION *pTransfer2 // Type de réception.
)

Paramètres :
- Hhandle : Handle du système. Elle est donnée par la fonction SetPortOpen.
- PTransfer : Pointeur vers la structure de Communication de type DATA_GTI.
- Ptransfer2 : Pointeur vers la structure de Communication de type DATA_ACQUISITION.

Valeur retournée :
Si 0 : soit la fonction a détecté une erreur de Communication, soit aucune trame n'a été reçue.
Si >0 : Nombre de points total récupéré (lDataLen1 + lDataLen2 + lDataLen3 + lDataLen4) de DATA_GTI

Fonction :
Cette fonction permet de récupérer les trames acquises et contenues dans le système.

## NOMBRE DE DONNEES - H8 - A PRENDRE

**GetLenH8**

- *Appel en _stdcall*

**Long GetLenH8**
(
)

**_GetLenH8**

- *Appel en __declspec(dllimport)*

**Long _GetLenH8**
(
)

Fonction :
Cette fonction permet de récupérer le nombre de données disponibles dans le buffer de la voie de communication H8.

## RECUPERE LES DONNEES - H8 -

**GetH8**

- *Appel en _stdcall*

**Long GetH8**
(
DATA_CANAL *pBuffer // (donnée de type DATA_CANAL)
,long LenBuffer //
,BYTE UNICODE // toujours à 0
)

**_GetH8**

- *Appel en __declspec(dllimport)*

**Long _GetH8**
(
UCHAR *BufferSend // (donnée de type BYTE)
,long LenBuffer //
,BYTE UNICODE // toujours à 0
)

Paramètres :
- pBuffer: Pointeur vers le vers la structure de transfert.
  LenBuffer : Nombre de données a récupérer. (ex : 512 ou 1024)

ou

- BufferSend: Pointeur vers le buffer de transfert .
- LenBuffer : Nombre de données a récupérer. (ex : 512 ou 1024)

Fonction :
Cette fonction permet de récupérer les données dans le buffer de la voie de communication H8.

## ENVOI DES DONNEES VERS LE - H8 -

**SendH8** • *Appel en _stdcall*

**SendH8**
(
DATA_CANAL *pBuffer // (donnée de type DATA_CANAL)
)

**_ SendH8** • *Appel en __declspec(dllimport)*

**_ SendH8**
(
UCHAR *BufferSend // (donnée de type BYTE)
,long LenBuffer //
)

Paramètres :
- pBuffer: Pointeur vers le vers la structure de transfert.

Ou

- BufferSend: Pointeur vers le buffer de transfert du message a envoyer vers le H8 .
- LenBuffer : Longueur du buffer a envoyer

Fonction :
Cette fonction permet d'envoyer les données dans le buffer de la voie de communication H8.

## ETAT DU BUFFER DE TRANSMISSION - H8 -

**EmptyBuffer** • *Appel en _stdcall*

**Long EmptyBuffer**
(
)

**_ EmptyBuffer** • *Appel en __declspec(dllimport)*

**Long _ EmptyBuffer**
(
)

Fonction :
Cette fonction permet de connaître le nombre de données en cours d'envoi dans le buffer de transmission.

## ARRET DE L'ACQUISITION LENTE

**StopAcquiLENTE** • *Appel en _stdcall*

**StopAcquiLENTE**
(
)

**_ StopAcquiLENTE** • *Appel en __declspec(dllimport)*

**_ StopAcquiLENTE**
(
)

Fonction :
Arrêt d'acquisition Lente. Cela va mettra en pause l'envoi de donnée.

## CHANGEMENT DE VITESSE

**ChangeVitesse** • *Appel en _stdcall*

**ChangeVitesse**
(
HANDLE hHandle // handle du périphérique.
,long Vitesse // Vitesse de transmission Choisie
)

**_ ChangeVitesse** • *Appel en __declspec(dllimport)*

**_ChangeVitesse**
(
HANDLE hHandle // handle du périphérique.
,long Vitesse // Vitesse de transmission Choisie
)

Paramètres :
- Hhandle : Handle du système. Elle est donnée par la fonction SetPortOpen.
- Vitesse : Vitesse = 0 : Changement a 9600Bauds // Vitesse = 1 : Changement a 125000 Bauds .

Fonction :
Permet de Changer la vitesse de transmissions Interne au GTI2 entre les Deux Processeurs. Par défaut, la vitesse est paramétrée a 125000Bauds

# PARAMETRES ET STRUCTURES

## GENERATEUR

On peut choisir soit une fréquence de génération 48Mhz, soit un timer 16bits configurable avec une base de temps de 12MHz ou 4MHz . Dans le cas de l'utilisation du timer 16bits, il faut toujours choisir la base de temps (clock) la plus faible.

**fréquence de génération = (base de temps)/(65536 - Reload)**

*Début Structure DATA_GENERATEUR*
**NbreData** // nombre de données envoyées - (donnée type LONG )
**BaseDeTemps** // 0 (48MHz) ou 1 (12MHz) ou 2 (4MHz) - (donnée type BYTE )
**Reload** // Subdivision de la base de temps - (donnée type LONG ) 0 si 48MHz
**BufferOut[16000]** // Signal - (donnée type LONG )
*Fin Structure*

Exemple :

```
for(i=0;i<48;i++) USB.DataGenerateur.BufferOut[i]=Sinus2[i]
USB.DataGenerateur.NbreData=48        // Nombre de données

USB.DataGenerateur.Reload=0           // pas de subdivision
USB.DataGenerateur.BaseDeTemps=0      // 48MHz

USB.DataGenerateur.Reload=65535       //
USB.DataGenerateur.BaseDeTemps=1      // F = 12MHz (12MHz / (1)

USB.DataGenerateur.Reload=65523       //
USB.DataGenerateur.BaseDeTemps=1      // F = 1MHz (12MHz / (65536-65523)
```

## ACQUISITION RAPIDE - CONFIGURATION

Ces convertisseurs AD ont la particularité de pouvoir s'auto-calibrer. La documentation technique recommande d'effectuer un calibrage à chaque fois que l'on change la base de temps. Je n'ai pas pu tester son efficacité donc à voir.
Par contre il est primordial d'effectuer cette auto-calibrage à chaque fois que l'on change le calibre (en fait on change la référence interne du convertisseur).
Cette auto-calibrage se lance par :

**Calibrage = 1**

Une fois ce calibrage effectué, l'acquisition est lancée immédiatement.

*Début Structure DATA_ACQUISITION*
**Calibrage** // (0) pas de calibrage (1) calibrage - (donnée type BYTE )
**Voie[4]** // Tableau de configuration des voies - (de type ACQUISITION_ *VOIE*)
**Config** // Configuration du système - (de type ACQUISITION_CONFIG)
**Trig** // Configuration du TRIG - (de type ACQUISITION_ TRIG)
*Fin Structure*

## CONFIGURATION DES VOIES

L'ORPHY GTI2 possède 4 canaux d'acquisitions rapides simultanées. Chacun de ces canaux est configuré soit en mode unipolaire, soit en mode différentielle. Chacun de ces canaux possèdent un calibre +/-15V et un calibre +/-7,5V.

*Début Structure ACQUISITION_VOIE*
**On** // (0) voie inhibée (1) voie activée - (donnée type BYTE )
**Calibre** // (0) calibre +/- 15V (1) calibre +/- 7.5V - (donnée type BYTE )
**Differentiel** // (0) Mode unipolaire (1) Mode différentiel - (donnée type BYTE )
*Fin Structure*

## CONFIGURATION DE L'ACQUISITION

Lec ADC sont des ADC rapides et par conséquent ne peuvent pas "tourner" à des fréquences inférieures à 10kHz. Afin de pouvoir faire des acquisitions avec des bases de temps inférieures, il faut configurer le timer avec la valeur la supérieure ou égale à 10kHz ( **BaseDeTemps = 2MHz / Reload = 56 )** et modifier la variable UnSurN

La fréquence d'échantillonnage est déterminée par :
**Fech = base de temps/(256- Reload)**

*Début Structure ACQUISITION_CONFIG*
**BaseDeTemps** // (0) 6MHz (1) 2MHz (2) 10MHz - (donnée type BYTE )
**Reload** // Subdivision de la bas de temps - (donnée type BYTE ) uniquement pour 6 MHz et 2MHz
**UnSurN** // Rapatrie 1 point sur N - (donnée type LONG )
*Fin Structure*

Exemples :

| Fech = 1MHz | Fech = 100KHz | Fech = 1KHz |
|---|---|---|
| **BaseDeTemps= 1 (2MHz)**<br>**Reload = 254**<br>**UnSurN = 1** | **BaseDeTemps= 1 (2MHz)**<br>**Reload = 236**<br>**UnSurN = 1** | **BaseDeTemps= 1 (2MHz)**<br>**Reload = 56**<br>**UnSurN = 10** |

Fréquences possibles :

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 MHz<br>6MHz | 3MHz<br>2MHz | 1.5MHz<br>1.2MHz | 1MHz<br>857KHz | 750KHz<br>666KHz | 600KHz<br>545KHz | 500KHz<br>etc … |

## CONFIGURATION DU TRIG / PRETRIG

*Début Structure ACQUISITION_TRIG*

| | |
|---|---|
| **TrigOn** | // (0) pas de TRIG (1-2-3 ou 4) TRIG sur la voie sélectionnée - (donnée type BYTE ) |
| **PreTrigOn** | // PRETRIG activé (Activation du TrigOn nécessaire !) |
| **FrontMontant** | // (0) Front descendant (1) Front Montant - (donnée type BYTE ) |
| **Valeur** | // Valeur du seuil de trig (12bits) - (donnée type LONG ) |

*Fin Structure*

### PRE-TRIG ➔ QUELQUES EXPLICATIONS

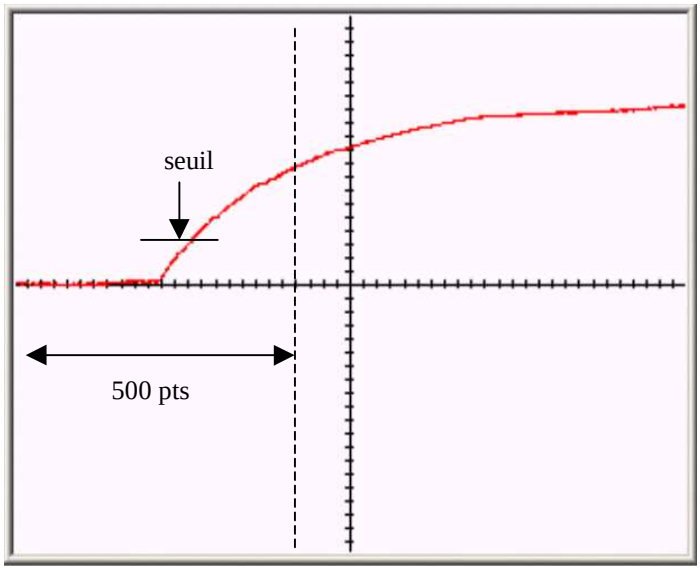
Le pretrig fonctionne de la façon suivante :
Lorsqu'on lance l'acquisition, on lance un Timer (16bits). Dans le programme principal, on teste si le trig est arrivé pendant que le timer n'a pas débordé. Si c'est le cas l'acquisition se termine normalement.
Si le timer déborde alors que le trig n'est pas arrivé, on stoppe l'acquisition, on reset les fifos, on relance l'acquisition et le timer 1 et ainsi de suite… ce temps d'arrêt n'est pas négligeable (environ 10µs), donc il se peut que le système ne voit pas le trig, surtout dans les faibles bases de temps.

En pratique, on va scanner pendant un temps donné (interne au système) une fenêtre de 500pts, de la fréquence d'échantillonnage, dans laquelle on va détecter le déclenchement du signal.

<u>2 solutions :</u>

Cas 1 : aucune détection n'est faite. Dans ce cas, le système relance la détection jusqu'à ce qu'une demande d'arrêt d'acquisition soit reçue.

Cas 2 : Le signal a été détecté. Le seuil de déclenchement se trouvera dans cette fenêtre de 500 pts. Cependant le nombre de points avant ce seuil sera aléatoire.




Avec la fonction LanceAcquiManuel, il est possible de définir manuellement la fenêtre de pretrig en paramétrant directement le timer associé .

Le calcul actuel est :

$$\text{Reload Timer Pretrig} = 65536 - \text{Fenetre} \times \frac{\text{FréquencePretrig} \times (256 - \text{reload})}{\text{BaseDeTemps}}$$

FréquencePretrig : 4MHz ou 12MHz
Reload : reload de la fréquence d'échantillonnage calculée [ Fech = base de temps/(256- Reload) ]
BaseDeTemps : base de temps choisie (6MHz, 2MHz ou10MHz) [ Fech = base de temps/(256- Reload) ]
Fenetre : Nombre de points constituant la fenêtre de recherche.

## STRUCTURE DE RECUPERATION DE DONNEES

*Début Structure DATA_GTI*

**lDataLen1** // longueur du tableau de données reçues sur la voie 1 - (donnée type LONG )
**lDataLen2** // longueur du tableau de données reçues sur la voie 2 - (donnée type LONG )
**lDataLen3** // longueur du tableau de données reçues sur la voie 3 - (donnée type LONG )
**lDataLen4** // longueur du tableau de données reçues sur la voie 4 - (donnée type LONG )
**bBufferIn1[256]** // Données de la voie 1 - (donnée type LONG )
**bBufferIn2[256]** // Données de la voie 2 - (donnée type LONG )
**bBufferIn3[256]** // Données de la voie 3 - (donnée type LONG )
**bBufferIn4[256]** // Données de la voie 4 - (donnée type LONG )

*Fin Structure*

## STRUCTURE DE COMMUNICATION PIPE

*Début Structure DATA_CANAL*

**lDataLen** // longueur du tableau de données reçues - (donnée type LONG )
**bBuffer [512]** // Données de la voie 1 - (donnée type BYTE )

*Fin Structure*

# DIVERS

***Exemple de connexion :***

```
hHandle= SetPortOpen("OrphyGTI-0")
Si hHandle ≠ 0
  GetVersion(hHandle,Buffer)
  LanceTHREAD(hHandle)
                -
                -
        protocole serie
                -
                -
   StopTHREAD(hHandle)

  DataAcqui.Calibrage = yes
  LanceAcqui(hHandle,adresseof(DataAcqui))
  Faire
        LenUSBData = GetUSBData(hHandle,&DataGTI,&DataAcqui)
        Si LenUSBData different de 0 alors
            Pour i = 1 à LenUSBData
                Yea[0]=TensionCalcul(USB.DataGTI.bBufferIn1[i])
                Yea[1]=TensionCalcul(USB.DataGTI.bBufferIn2[i])
                Yea[2]=TensionCalcul(USB.DataGTI.bBufferIn3[i])
                Yea[3]=TensionCalcul(USB.DataGTI.bBufferIn4[i])
                Affiche(Xn,YEA[0], YEA[1], YEA[2], YEA[3])
            i suivant
        Fin de Si
   Jusqu'à la fin de l'acquisition
   StopAcqui(hHandle);
SetPortClose(hHandle)
```

### *Déclaration VB*

```
Declare Function DefineUSB Lib "GtiCommX" (ByVal hHandle As Long) As Long
Declare Function SetPortOpen Lib "GtiCommX" (ByVal strVersion As String) As Long
Declare Function SetPortClose Lib "GtiCommX" (ByVal hHandle As Long) As Long
Declare Function GetVersionSysteme Lib "GtiCommX" (ByVal hHandle As Long, ByRef strVersion As String) As Long
Declare Sub StopTHREAD Lib "GtiCommX" ()
Declare Function IsRunTHREAD Lib "GtiCommX" () As Long
Declare Function LanceTHREAD Lib "GtiCommX" (ByVal hHandle As Long) As Long
Declare Function GetUSBData Lib "GtiCommX" (ByVal hHandle As Long, ByRef pTransfer As DATA_GTI, ByRef pTransfer2 As
DATA_ACQUISITION) As Long
Declare Function LanceAcqui Lib "GtiCommX" (ByVal hHandle As Long, ByRef pTransfer As DATA_ACQUISITION) As Long
Declare Function StopAcqui Lib "GtiCommX" (ByVal hHandle As Long) As Long
Declare Function GetLenH8 Lib "GtiCommX" () As Long
Declare Function EmptyBuffer Lib "GtiCommX"  () As Long
Declare Sub StopAcquiLENTE Lib "GtiCommX"  ()
Declare Sub SendH8 Lib "GtiCommX" (ByRef Data As DATA_CANAL)
Declare Function GetH8 Lib "GtiCommX" (ByRef Data As DATA_CANAL, ByVal LenBuffer As Long, ByVal Unicode As Byte) As Long
Declare Function EmptyBuffer Lib "GtiCommX" () As Long
Declare Sub StopGenerateur Lib "GtiCommX" (ByVal hHandle As Long)
Declare Function LanceGenerateur Lib "GtiCommX" (ByVal hHandle As Long, ByVal Boucle As Long, ByRef SignalOut As
DATA_GENERATEUR) As Long
Declare Function ChangeVitesse Lib "GtiCommX" (ByVal hHandle As Long, ByVal Vitesse As Long) As Long
Declare Function LanceAcqui Lib "GtiCommX" (ByVal hHandle As Long, ByRef pTransfer As DATA_ACQUISITION, CLK as Long, BT
as long) As Long
```

### *Déclaration VC++*

```
extern "C" __declspec(dllimport) long _SetPortClose (HANDLE  hHandle);
extern "C" __declspec(dllimport) HANDLE _SetPortOpen (BSTR strDeviceName);
extern "C" __declspec(dllimport) long _GetVersion(HANDLE hHandle,char *strVersion);
extern "C" __declspec(dllimport) long _GetUSBData (HANDLE hHandle,DATA_GTI *pTransfer,DATA_ACQUISITION *pTransfer2);
extern "C" __declspec(dllimport) long _LanceAcqui(HANDLE hHandle,DATA_ACQUISITION *pTransfer );
extern "C" __declspec(dllimport) long _StopAcqui(HANDLE hHandle );
extern "C" __declspec(dllimport) long _GetPipe (HANDLE hHandle,char *Buffer1,long LenSend,long Canal);
extern "C" __declspec(dllimport) long _SendPipe (HANDLE hHandle,char *Buffer1,long LenSend,long Canal);
extern "C" __declspec(dllimport) long _LanceTHREAD (HANDLE hHandle);
extern "C" __declspec(dllimport) void _StopTHREAD ();
extern "C" __declspec(dllimport) long _IsRunTHREAD ();
extern "C" __declspec(dllimport) void _SendH8 (unsigned char *pBuffer,long LenBuffer);
extern "C" __declspec(dllimport) long _GetH8 (unsigned char *BufferSend, long LenBuffer,BYTE UNICODE);
extern "C" __declspec(dllimport) long _GetLenH8 ();
extern "C" __declspec(dllimport) long _LanceGenerateur (HANDLE hHandle,UCHAR Loop,DATA_GENERATEUR* Generateur);
extern "C" __declspec(dllimport) long _EmptyBuffer ();
extern "C" __declspec(dllimport) void _StopAcquiLENTE ();
extern "C" __declspec(dllimport) long _ChangeVitesse (HANDLE hHandle, long Vitesse);
extern "C" __declspec(dllimport) long _LanceAcquiManuel (HANDLE hHandle,DATA_ACQUISITION *pTransfer , long CLK, long BT);
```

### *Déclaration DELPHI*

```
implementation
function DoEvents; external 'GTICOMMX.DLL';
function _SetPortClose; external 'GTICOMMX.DLL';
function _SetPortOpen; external 'GTICOMMX.DLL';
function _GetVersion; external 'GTICOMMX.DLL';
function _GetUSBData; external 'GTICOMMX.DLL';
function _LanceAcqui; external 'GTICOMMX.DLL';
function _StopAcqui; external 'GTICOMMX.DLL';
function _LanceTHREAD; external 'GTICOMMX.DLL';
function _StopTHREAD; external 'GTICOMMX.DLL';
function _IsRunTHREAD; external 'GTICOMMX.DLL';
function _EmptyBuffer; external 'GTICOMMX.DLL';
procedure _SendH8; external 'GTICOMMX.DLL';
function _GetH8; external 'GTICOMMX.DLL';
function _GetLenH8; external 'GTICOMMX.DLL';
function _StopAcquiLENTE; external 'GTICOMMX.DLL';
function _GetPipe; external 'GTICOMMX.DLL';
function _SendPipe; external 'GTICOMMX.DLL';
function _LanceGenerateur; external 'GTICOMMX.DLL';
function _StopGenerateur; external 'GTICOMMX.DLL';
function _ChangeVitesse; external 'GTICOMMX.DLL';
function _ LanceAcquiManuel; external 'GTICOMMX.DLL';
END.
```