

# Développement d'un module NIM « trigger » de logique programmable.

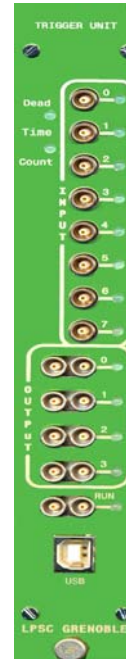
Olivier BOURRION, Bernard Boyer

<b>1</b>	<b>OBJECTIFS DU PROJET</b>	<b>2</b>
<b>2</b>	<b>CARACTERISTIQUES TECHNIQUES DETAILLEES</b>	<b>3</b>
<b>3</b>	<b>REALISATION MATERIELLE</b>	<b>3</b>
<b>3.1</b>	<b>CARTE ELECTRONIQUE</b>	<b>3</b>
3.1.1	SYNOPTIQUE	4
3.1.2	ALIMENTATIONS	5
3.1.3	ENTREES	5
3.1.4	MICROCONTROLEUR	5
3.1.5	SORTIES	6
3.1.6	FPGA	6
3.1.7	LED DE SIGNALISATION	6
<b>3.2</b>	<b>ARCHITECTURE DU FPGA</b>	<b>7</b>
3.2.1	SYNOPTIQUE	7
3.2.2	DESCRIPTION	8
3.2.3	TABLEAUX D'ADRESSAGES	10
<b>4</b>	<b>LOGICIEL MICROCONTROLEUR</b>	<b>12</b>
<b>4.1</b>	<b>ARCHITECTURE DU MICRO LOGICIEL</b>	<b>13</b>
<b>4.2</b>	<b>INSTRUCTION INST_SET_ADD</b>	<b>13</b>
<b>4.3</b>	<b>INSTRUCTION INST_READ_COUNTERS</b>	<b>14</b>
4.3.1	INSTRUCTION INST_READ_STATUS	14
<b>4.4</b>	<b>INSTRUCTION INST_WRITE_DAC</b>	<b>14</b>
<b>4.5</b>	<b>INSTRUCTIONS PERMETTANT LA CONFIGURATION DU FPGA</b>	<b>15</b>
4.5.1	INSTRUCTION INST_CLEAR_FPGA	15
4.5.2	INSTRUCTION CHECK_PROG_FPGA	15
<b>4.6</b>	<b>INSTRUCTION CLEAR_MEM_POINTER</b>	<b>15</b>
<b>4.7</b>	<b>ECRITURE DE DONNEES DANS LES BLOCS MEMOIRES</b>	<b>16</b>
<b>5</b>	<b>API DE CONTROLE DE LA CARTE</b>	<b>16</b>
<b>5.1</b>	<b>TCPT_TRIGGER_API CLASS REFERENCE</b>	<b>17</b>
<b>5.2</b>	<b>TLOGICINTERPRETER CLASS REFERENCE</b>	<b>22</b>
<b>6</b>	<b>INTERFACE GRAPHIQUE DE CONTROLE / MANUEL UTILISATEUR</b>	<b>25</b>
<b>7</b>	<b>ANNEXES</b>	<b>26</b>
<b>7.1</b>	<b>INSTALLATION SUR UNE PLATEFORME WINDOWS</b>	<b>26</b>
<b>7.2</b>	<b>INSTALLATION SUR UNE PLATEFORME LINUX</b>	<b>27</b>
<b>7.3</b>	<b>INSTALLATION SUR UNE PLATEFORME MAC</b>	<b>27</b>

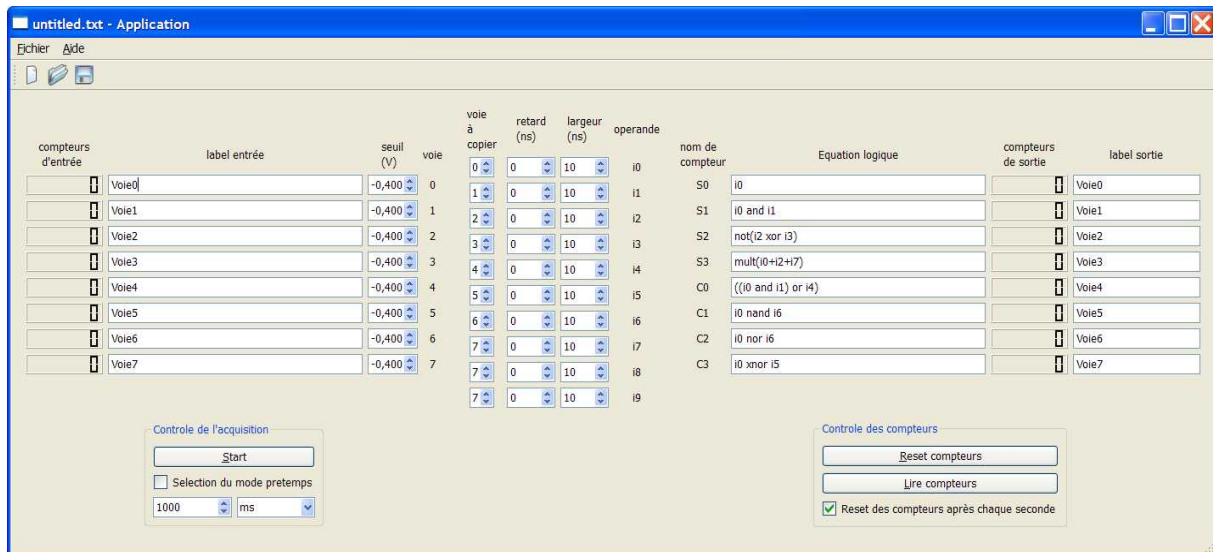
# 1 Objectifs du projet

Afin de remplacer les vieux modules NIM (discriminateur, coïncidence, mise en forme, échelle de comptage,...) sur certains travaux pratiques de la plateforme de travaux pratique, Nucléaire/Particules de UJF/ENSPG/INPG, un nouveau module NIM programmable a été développé par le service électronique du laboratoire. Il est construit autour d'un FPGA 100 MHz et comprend les fonctionnalités suivantes :

- ✚ 8 entrées analogiques avec discriminateurs indépendants programmables.
- ✚ Possibilité de dupliquer les sorties des discriminateurs.
- ✚ Mise en forme des portes logiques (retard et largeur réglables par pas de 10 ns).
- ✚ Possibilité d'opérations logiques (AND, OR,...) entre les signaux.
- ✚ Quatre sorties logiques.
- ✚ Seize compteurs.
- ✚ Possibilité de fixer la durée de l'acquisition (mode pré temps)



Le module est configuré via une interface graphique sous PC. Le fichier résultant est ensuite chargé sur le module via une connexion USB. L'interface graphique permet aussi de relire les différents compteurs du module et de sauvegarder la configuration.



## 2 Caractéristiques techniques détaillées

Consommation sur le -6V : jusqu'à 250mA toutes sorties actives

Consommation sur le +6V : 350 mA

Communication par USB 1.1.

8 entrées analogiques avec seuil réglables entre -1V et 0V et avec pas de réglage de 60  $\mu$ V

4x2 sorties trigger au standard NIM.

1x2 sorties run au standard NIM.

Latence minimum de la carte avec les retards réglés à zéro : 60 à 70 ns

La gamme de réglage des retards est de 0 à  $2^{13}-1$  périodes d'horloge soit de 0 ns à 81910 ns

La gamme de réglage des largeurs est de 1 à  $2^{13}-1$  périodes d'horloge soit de 10 ns à 81910 ns

La gamme de réglage du pretemps est de 0 à  $2^{32}-1$  pas de 1 ms soit de 1 ms à 49 jours

16 compteurs de 24 bit soit une dynamique > 16 millions

Fréquence maximum des compteurs d'entrée : 140 MHz

Equations logiques ayant jusqu'à 10 opérandes.

Latence électronique indépendante de l'équation et du nombre d'opérande.

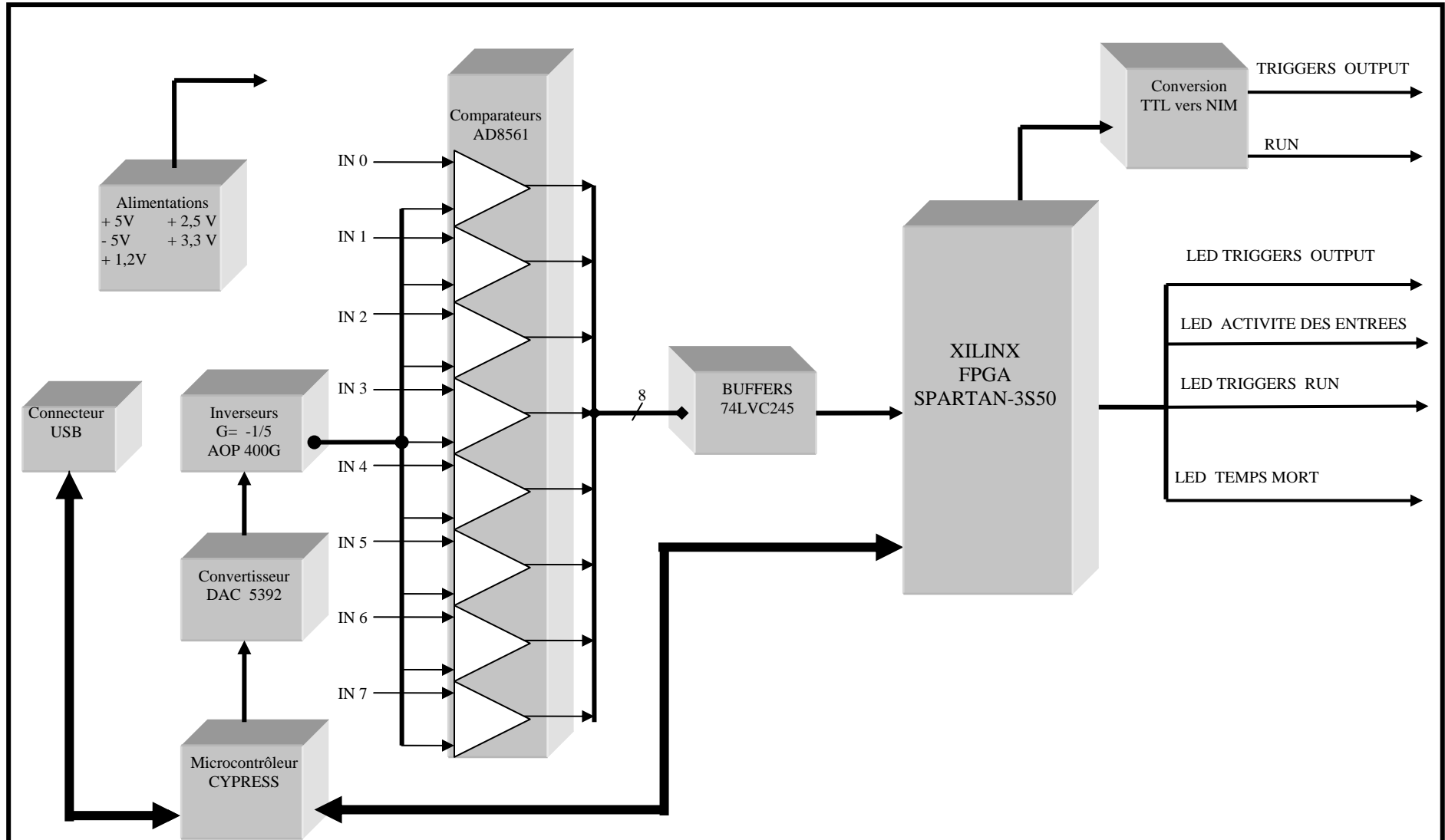
Jitter ajouté aux signaux par la carte : 10 ns

## 3 Réalisation matérielle

### 3.1 Carte électronique

Pour réaliser cet objectif, une carte électronique a du être réalisée. Elle a été conçue avec l'objectif d'être la moins chère possible et de pouvoir fonctionner indifféremment sur table ou dans un châssis NIM. Son synoptique est fourni ci-dessous.

### 3.1.1 Synoptique



### 3.1.2 Alimentations

Les alimentations locales sont fabriquées à partir du +6V et -6V à l'aide de régulateurs linéaires. Le -5V et le +5V est utilisé par les composants analogiques, et les autres tensions sont utilisées par les composants numériques.

### 3.1.3 Entrées

La carte dispose de 8 entrées analogiques qui alimentent des discriminateurs. Les seuils de ces discriminateurs sont réglables individuellement entre -1V et 0V à l'aide de DACs 14 bit.

### 3.1.4 Microcontrôleur

La communication avec la carte est faite grâce à un microcontrôleur offrant une connectivité USB aisée à mettre en œuvre. Son autre avantage est la possibilité de n'avoir aucune mémoire non-volatile implantée sur la carte que ce soit pour lui-même ou pour le FPGA. En effet, après sa mise sous tension, il offre un fonctionnement minimum qui permet le téléchargement du programme par le lien USB.

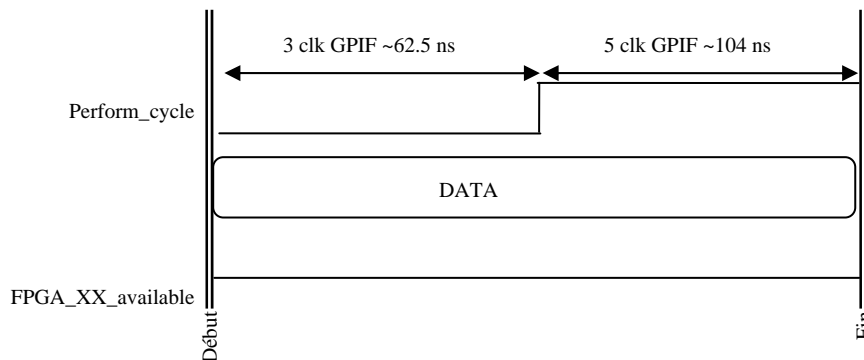
Ses interfaces avec le reste de la carte sont :

- Un lien série synchrone pour configurer le DAC 8 voies qui permet de régler les seuils analogique.
- Un lien série synchrone pour configurer le FPGA après la mise sous tension.
- Un protocole propriétaire pour communiquer avec le FPGA et lui transmettre les différents paramètres de trigger.

#### Détails sur le protocole propriétaire

Deux modes de transfert sont définis, le mode adresse et le mode donnée. Ils sont en tous points similaires, à ceci près que l'un des fils de l'interface sert à définir le type de cycle en cours. Cela est fait pour permettre un bus de taille réduite, mais qui permet tout de même de faire un adressage en interne du FPGA.

Cette fonctionnalité est réalisée par le périphérique GPIF (*General Purpose Interface*) offert par le microcontrôleur Cypress CY7C68013. Le transfert se déroule de manière asynchrone. Le front montant du signal `perform_cycle` sert à confirmer le transfert.



### 3.1.5 Sorties

A la sortie du FPGA, 4 signaux de trigger sont dupliqués et convertis au format NIM afin d'être rendus disponible au monde extérieur. Par ailleurs, un signal « run » est fourni, il donne l'image de l'état d'activation ou d'arrêt de l'acquisition, cette information est particulièrement utile en mode pré temps. Il est aussi dupliqué et fourni au standard NIM.

### 3.1.6 FPGA

Ce composant est en charge de toute la partie mise en forme des impulsions, génération des duplications et des équations logiques (par l'intermédiaire de tables de vérités). Il contient aussi les différents compteurs d'activité ainsi que la gestion des affichages LED.

### 3.1.7 LED de signalisation

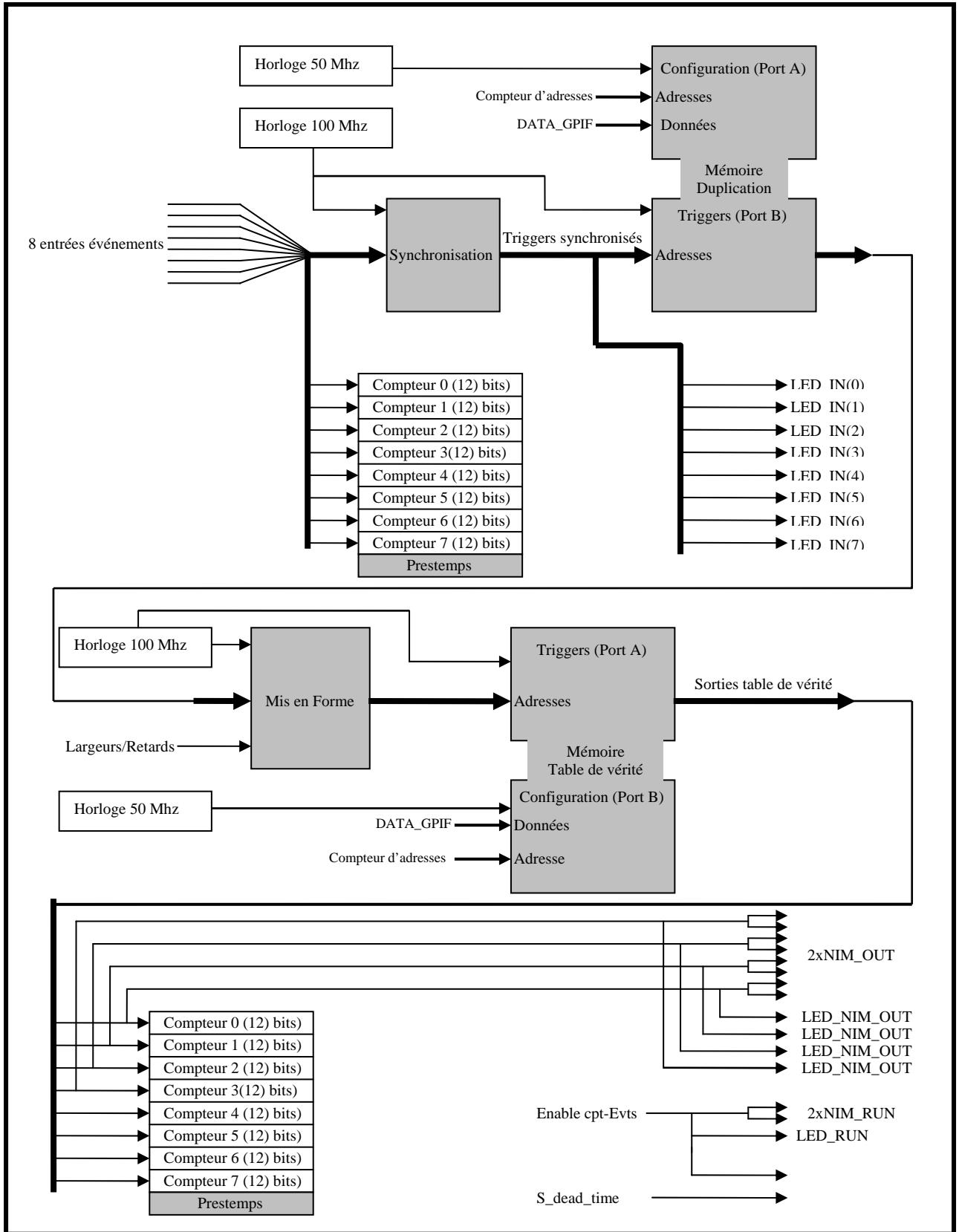
Chaque entrée et chaque sortie trigger a une LED d'activité associée. Celle-ci clignote à environ 4Hz tant qu'il y a de l'activité sur l'entrée (ou sortie) considérée. La fréquence de clignotement maximum est de 4 Hz.

Par ailleurs, tant que l'acquisition est active, la LED « run actif » est allumée, elle s'éteint soit de façon manuelle en passant de run à stop, soit automatiquement à la fin du délai imparti dans le mode pré temps.

Enfin, une LED de temps mort affiche l'état du OU logique de tous les états des modules de mise en forme. En effet ceux-ci ne pouvant gérer d'une impulsion à la fois, si une deuxième se présente trop rapidement, elle est ignorée.

### 3.2 Architecture du FPGA

#### 3.2.1 Synoptique



## 3.2.2 Description

### 3.2.2.1 Partie acquisition

Après leur mise en forme par les discriminateurs, les signaux logiques d'entrées sont injectés dans le FPGA. L'activité de ces signaux est mesurée à l'aide des compteurs d'entrée, ceux-ci comptent directement les fronts montants des impulsions d'entrée lorsqu'ils sont validés (cette remarque est importante, voir les compteurs de sortie). D'autre part, dans le bloc *synchronisation*, ces signaux sont resynchronisés par rapport à l'horloge de fonctionnement du FPGA qui est de 100MHz. Cette opération ajoute un *jitter* (bruit de phase) au signal.

La deuxième étape, est de faire passer les signaux resynchronisés à travers une mémoire dite « *de duplication* ». Celle-ci a la particularité d'avoir son contenu initialisé de telle manière, qu'en fonction de la combinaison des signaux d'entrée (câblés sur le bus d'adresse de cette mémoire), la sortie donnée présente une duplication du ou des signaux désirés.

Après la duplication, les signaux sont remis en forme par les modules de *mise en forme*, ceux-ci détectent les passages de 0 à 1 des signaux et génèrent de nouvelles impulsions retardées d'une valeur programmable et dont la largeur est elle aussi réglée.

Enfin, les signaux passent à travers une nouvelle mémoire dite « *table de vérité* » qui a été initialisée avec tous les résultats possibles des différentes équations logiques saisies dans l'interface de contrôle. Comme précédemment, les vecteurs d'entrées sont câblés sur le bus d'adresse de la mémoire et le bus de donnée est utilisé comme résultat des équations. Encore une fois prenons l'exemple d'une équation logique à 2 entrées : le **ET** logique.

Sa table de vérité :

Vecteur	sortie
00	0
01	0
10	0
11	1

Il suffit alors d'initialiser une mémoire ayant un bus d'adresse de 2 bit et un bus de donnée de 1 bit avec le contenu de la table de vérité pour voir cette fonction réalisée (c'est ce qui est fait dans les FPGA par les synthétiseurs). Si on généralise le raisonnement, il suffit d'avoir une mémoire assez profonde (taille du bus d'adresse) pour gérer plus d'opérandes et assez large (taille du bus de données) pour juxtaposer plusieurs équations.

A la sortie de la table de vérité, les taux de comptage des différents triggers sont mesurés par des compteurs lorsqu'ils sont activés. Ceux-ci comptent l'activité de signaux resynchronisés, cela veut dire qu'il peut y avoir un léger écart par rapport à un même signal (avec une équation sortie=entrée) compté directement à l'entrée.

### 3.2.2.2 Partie contrôle

L'ensemble des opérations précédentes n'est possible que si :

- Le contenu des mémoires peut être chargé à la demande par USB.
- Les 16 compteurs peuvent être lus et effacés.
- Les retards et largeurs peuvent être réglés individuellement.
- Le pré temps peut être réglé.
- L'acquisition peut être démarrée ou arrêtée à souhait.

L'ensemble de ces accès repose sur le protocole propriétaire définie en 3.1.4. Pour faire un accès à une adresse précise, le transfert se déroule en 2 temps :

1. Tout d'abord un cycle adresse est exécuté de manière à ce que l'un des registres internes du FPGA mémorise ce pointeur. L'adresse est spécifiée sur 8 bit.
2. Puis le cycle donnée est exécuté (en mode lecture ou écriture).

Presque tous les accès se déroulent de cette façon. Il y a deux légères variations

1. pour les blocs mémoires, car sinon ils nécessiteraient une plage d'adressage bien supérieure (chaque bloc fait 2ko),
2. les retards / largeurs et le registre de prétemps, car ils nécessiteraient une logique de décodage bien fournie.

La parade à ces problèmes est de faire un accès BURST pour ces zones mémoires, c'est-à-dire que seule une adresse de base est attribuée à une des ces zones. Pour réaliser ces accès, il y a deux implémentations différentes :

- dans le cas des zones mémoire, un pointeur commun d'offset est incrémenté. Après un accès ce pointeur doit être réinitialisé pour ne pas pénaliser les accès aux autres zones.
- dans le cas des banques de registres, les données sont décalées d'un registre à l'autre pour chaque accès. Il suffit d'envoyer le nombre exact de données dans le bon ordre, pour qu'à la fin des décalages chacune soit à sa place.

Ces 2 solutions imposent un accès séquentiel, ce qui n'est pas préjudiciable dans notre cas, mais solutionne nos problèmes.

Il faut aussi noter que l'architecture est relativement simple, car toute l'interface de contrôle fonctionne à 50 MHz, et non pas à la fréquence de la partie acquisition. En effet, les accès aux zones mémoires sont effectués, lorsque l'acquisition est stoppée, et comme ces blocs présentent une fonctionnalité double port / double domaine d'horloge, il n'y a de nécessité de mettre en place des systèmes de multiplexages complexes.

### 3.2.3 Tableaux d'adressages

Lors de la lecture de ces tableaux, il faut garder en mémoire que les accès sont faits par octets.

#### 3.2.3.1 Compteurs

Règle de composition des adresses compteurs

7	6	5	4	3	2	1	0
0	0	Numéro du compteur à lire				Choix de l'octet à lire	

Ce qui donne le tableau suivant

Nom des Compteurs	Adresse HEXA	No des Bits du Compteur
Compteur_Input_0	0	Bit 0 à bit 7
	1	Bit 8 à Bit 15
	2	Bit 16 à Bit 23
Compteur_Input_1	4	Bit 0 à bit 7
	5	Bit 8 à Bit 15
	6	Bit 16 à Bit 23
Compteur_Input_2	8	Bit 0 à bit 7
	9	Bit 8 à Bit 15
	A	Bit 16 à Bit 23
Compteur_Input_3	C	Bit 0 à bit 7
	D	Bit 8 à Bit 15
	E	Bit 16 à Bit 23
Compteur_Input_4	10	Bit 0 à bit 7
	11	Bit 8 à Bit 15
	12	Bit 16 à Bit 23
Compteur_Input_5	14	Bit 0 à bit 7
	15	Bit 8 à Bit 15
	16	Bit 16 à Bit 23
Compteur_Input_6	18	Bit 0 à bit 7
	19	Bit 8 à Bit 15
	1A	Bit 16 à Bit 23
Compteur_Input_7	1C	Bit 0 à bit 7
	1D	Bit 8 à Bit 15
	1E	Bit 16 à Bit 23
Compteur_Output_8	20	Bit 0 à bit 7
	21	Bit 8 à Bit 15
	22	Bit 16 à Bit 23
Compteur_Output_9	24	Bit 0 à bit 7
	25	Bit 8 à Bit 15
	26	Bit 16 à Bit 23
Compteur_Output_10	28	Bit 0 à bit 7
	29	Bit 8 à Bit 15
	2A	Bit 16 à Bit 23
Compteur_Output_11	2C	Bit 0 à bit 7
	2D	Bit 8 à Bit 15
	2E	Bit 16 à Bit 23
Compteur_Output_12	30	Bit 0 à bit 7
	31	Bit 8 à Bit 15
	32	Bit 16 à Bit 23
Compteur_Output_13	34	Bit 0 à bit 7
	35	Bit 8 à Bit 15
	36	Bit 16 à Bit 23
Compteur_Output_14	38	Bit 0 à bit 7
	39	Bit 8 à Bit 15
	3A	Bit 16 à Bit 23

Compteur_Output_15	3C	Bit 0 à bit 7
	3D	Bit 8 à Bit 15
	3E	Bit 16 à Bit 23

### 3.2.3.2 Zones mémoires

Mémoire adressée	Adresse HEXA
duplication	40
Table de vérité des sorties	41
Table de vérité compteurs de sortie	42

L'effacement du pointeur d'offset se fait par une écriture à l'adresse 0.

### 3.2.3.3 Retards et largeurs

Tableau	Ordre d'envoi des données	Tableau	Ordre d'envoi des données
Largeur (9) <7..0>	1	Retard (9) <7..0>	21
Largeur (9) <13..8>	2	Retard (9) <13..8>	22
Largeur (8) <7..0>	3	Retard (8) <7..0>	23
Largeur (8) <13..8>	4	Retard (8) <13..8>	24
Largeur (7) <7..0>	5	Retard (7) <7..0>	25
Largeur (7) <13..8>	6	Retard (7) <13..8>	26
Largeur (6) <7..0>	7	Retard (6) <7..0>	27
Largeur (6) <13..8>	8	Retard (6) <13..8>	28
Largeur (5) <7..0>	9	Retard (5) <7..0>	29
Largeur (5) <13..8>	10	Retard (5) <13..8>	30
Largeur (4) <7..0>	11	Retard (4) <7..0>	31
Largeur (4) <13..8>	12	Retard (4) <13..8>	32
Largeur (3) <7..0>	13	Retard (3) <7..0>	33
Largeur (3) <13..8>	14	Retard (3) <13..8>	34
Largeur (2) <7..0>	15	Retard (2) <7..0>	35
Largeur (2) <13..8>	16	Retard (2) <13..8>	36
Largeur (1) <7..0>	17	Retard (1) <7..0>	37
Largeur (1) <13..8>	18	Retard (1) <13..8>	38
Largeur (0) <7..0>	19	Retard (0) <7..0>	39
Largeur (0) <13..8>	20	Retard (0) <13..8>	40

### 3.2.3.4 Registre de commande

Ce registre est disponible à l'adresse 0xC4.

bit	7	...	3	2	1	0
	Non affectés		Run_nstop	Reset_compteurs	Pretemps_select	

Run\_nstop : lorsque ce bit est à 1 l'acquisition est activée.

Reset\_compteurs : lorsque ce bit est à 1, les compteurs sont en mode reset.

Pretemps\_select : lorsque ce bit est à 1, le mode pre temps est activé ;

### 3.2.3.5 Registre de pretemps

Ce registre est disponible à l'adresse 0xC0

Tableau	Ordre d'envoi des données
Pretemps <7..0>	1
Pretemps <15..8>	2
Pretemps <23..16>	3
Pretemps <31..24>	4

## 4 Logiciel microcontrôleur

La carte est contrôlée par le microcontrôleur USB 1.1. A la mise sous tension, il doit être configuré par le PC hôte, c'est-à-dire que son exécutable est téléchargé par l'USB. Ensuite, grâce à celui-ci, il devient possible de configurer le FPGA (car il est à base de SRAM, donc volatile), et finalement la programmation des différentes mises en forme, seuil, équations logiques devient possible par 2 mécanismes différents : Le protocole SPI (lien série) et par l'utilisation des machine d'état programmable (GPIF).

Dans ce paragraphe nous allons lister les différentes commandes et leur utilisation. L'algorithme et la séquence adéquate d'utilisation sera expliquée dans le paragraphe suivant (API).

L'ensemble des commandes est définie dans le fichier *cpt\_trigger\_micro.h*

```

////////////////////////////////////
// instructions
////////////////////////////////////
#define INST_SET_ADD          0xA0
#define INST_READ_COUNTERS   0xA2
#define INST_READ_STATUS     0xA3
#define INST_WRITE_DAC       0xA4

#define INST_CLEAR_FPGA      0xA5
#define CHECK_PROG_FPGA      0xA6

#define CLEAR_MEM_POINTER    0xB3

////////////////////////////////////
// status
////////////////////////////////////
#define FPGA_CLEAR_SUCCESS   0x01
#define FPGA_CLEAR_FAILURE   0x00

#define FPGA_INIT_SUCCESS    0x01
#define FPGA_INIT_FAILURE    0x00

#define FPGA_DONE_SUCCESS    0x01
#define FPGA_DONE_FAILURE    0x00

////////////////////////////////////
// controle bit definitions
////////////////////////////////////
#define mode_select          0x01
#define reset_cpt            0x02
#define run_stop              0x04

#define set_raz_mem_bit      0x01

////////////////////////////////////
// adresses FPGA
////////////////////////////////////
#define add_pretemps         0xC0
#define add_CTRL              0xC4

```

```

#define add_mem_duplic           0x40
#define add_mem_tb_verite_out 0x41
#define add_mem_tb_verite_cpt 0x42
#define add_raz_mem_pointer     0x00

#define add_retard_largeur      0x80
#define add_base_compteur      0x00

#define add_status              0x03

```

## 4.1 Architecture du micro logiciel

Avant de détailler l'ensemble des commandes et leur utilisation, il convient de préciser quels sont les *endpoint* utilisés :

- **EP2OUT**, mode bulk : reçoit l'intégralité des instructions.
- **EP2IN**, mode bulk : permet de retourner les différents *status* et aussi les valeurs des différents compteurs.
- **EP4OUT**, mode bulk : permet de faire transfert rapide ayant pour effet de déclencher des transactions en mode burst (par l'intermédiaire du GPIF). Par défaut le bus est positionné en mode donnée.
- **EP6OUT**, mode bulk : C'est dans cet endpoint que sont écrites les données de configuration du FPGA. Chaque octet écrit est sérialisé par et envoyé au FPGA.

Les différents chronogrammes des accès GPIF sont définis dans les paragraphes précédents. A ce stade, il convient de se rappeler qu'il n'y a que 2 modes possibles :

- mode adresse : qui va positionner un pointeur d'adresse dans le FPGA
- mode donnée : qui permet d'aller lire ou écrire directement dans le FPGA

## 4.2 Instruction *INST\_SET\_ADD*

Trame :

octet	Contenu
0	INST_SET_ADD
1	Adresse

C'est une commande à 1 paramètre, l'adresse passée doit être l'une de celles spécifiée dans le fichier *cpt\_trigger\_micro.h*. A la réception de cette commande, une transaction en mode adresse est déclenchée sur le bus GPIF.

Détail des adresses disponible :

- **Add\_pretemps** : Registre de pretemps qui est un registre de 32 bit qui sert à programmer la durée du temps de comptage. Le LSB a une durée de 1 ms.
- **Add\_CTRL** : Registre de contrôle
- **Add\_mem\_duplic** : Adresse de base de la mémoire de duplication (2048 octets)
- **Add\_mem\_tb\_verite\_out** : Adresse de base de la table de vérité dont les sorties sont rendues disponibles en NIM et aux compteurs d'activités
- **Add\_mem\_tb\_verite\_out** : Adresse de base de la table de vérité dont les sorties ne sont envoyées qu'aux compteurs d'activité
- **Add\_raz\_mem\_pointer** : quand cette adresse est positionnée et qu'un cycle d'écriture a lieu, le pointeur d'offset mémoire, qui est commun aux 3 blocs, est remis à zéro.
- **Add\_retard\_largeur** : adresse de base du tableau contenant les différentes valeurs de retard et de largeur.

- **Add\_base\_compteur** : Cette adresse ne doit normalement pas être positionnée manuellement, lors de la commande de lecture des compteurs, elle est positionnée automatiquement.
- **Add\_status** : Cette adresse est positionnée automatiquement lors de la lecture de status, elle pointe sur le registre d'état du FPGA)

### 4.3 Instruction **INST\_READ\_COUNTERS**

Trame :

octet	Contenu
0	INST_READ_COUNTERS

Une requête de ce type va déclencher 16 lectures de compteurs 24 bits, c'est-à-dire 48 lectures GPIF en mode données, mais aussi 48 positionnements d'adresse afin de sélectionner les compteurs et les octets les constituant leur contenu. Lors du premier accès à un compteur donné (LSB en premier), le contenu est en fait mémorisé dans un buffer intermédiaire afin de ne pas lire des valeurs erronées. Par contre, du fait de la durée de lecture qui peut être assez longue au regard des taux de comptage en présence, les différents compteurs peuvent présenter des disparités alors même qu'ils comptent un signal identique.

Format de la trame de retour (disponible dans l'EP2IN)

octet	Contenu
0	CPT0<7..0>
1	CPT0<15..8>
2	CPT0<23..16>
...	...
45	CPT15<7..0>
46	CPT15<15..8>
47	CPT15<23..16>

#### 4.3.1 Instruction **INST\_READ\_STATUS**

Trame :

octet	Contenu
0	INST_READ_COUNTERS

Cette instruction permet d'avoir accès au registre d'état du FPGA. L'octet est retourné dans l'EP2IN.

Le bit 0 de cet octet représente l'état du compteur de prétemps. C'est-à-dire que lorsqu'il vaut 1, le prétemps est toujours actif et en décomptage.

### 4.4 Instruction **INST\_WRITE\_DAC**

Trame :

octet	Contenu
0	INST_WRITE_DAC
1	DAC_MSB
2	DAC_middle
3	DAC_LSB

Cette instruction permet de faire transférer par le microcontrôleur la donnée de contrôle du DAC 8 voies qui est embarqué sur la carte. Les différents mots à transférer sont assez complexes, mais ce protocole est pris en charge par l'API. Le transfert est effectué par lien série « SPI », cela implique de retourner l'ensemble des mots (en effet le périphérique série du microcontrôleur transfère les données en LSB first), une fonction est implémentée sur le microcontrôleur afin de prendre cela en charge.

#### 4.5 Instructions permettant la configuration du FPGA

Seules 2 commandes sont fournies pour permettre la programmation du FPGA :

- La commande d'effacement qui permet de réinitialiser un cycle de programmation
- La commande de contrôle qui permet de vérifier si la configuration s'est bien passée

Par contre, il n'y a pas de commande spécifique pour envoyer les données (sérialisées par le microcontrôleur) vers le FPGA. Il suffit tout simplement de transférer les données dans l'EP6OUT.

##### 4.5.1 Instruction INST\_CLEAR\_FPGA

Trame :

octet	Contenu
0	INST_CLEAR_FPGA

Cette instruction fait passer le FPGA en mode d'effacement, et tant que celui-ci n'a pas terminé cette séquence le mot d'état indiquant la fin de l'opération n'est pas retourné. Aussi, elle passe le FPGA et le DAC en mode reset (le fil est positionné à l'état bas).

Retour disponible dans l'EP2IN :

octet	Contenu
0	FPGA_CLEAR_SUCCESS ou FPGA_CLEAR_FAILURE

##### 4.5.2 Instruction CHECK\_PROG\_FPGA

Trame :

octet	Contenu
0	CHECK_PROG_FPGA

Cette instruction permet de déterminer si la configuration du FPGA a réussi ou non.

Retour dans EP2IN :

octet	Contenu
0	FPGA_INIT_SUCCESS ou FPGA_INIT_FAILURE
1	FPGA_DONE_SUCCESS ou FPGA_DONE_FAILURE

Si l'erreur se trouve sur l'INIT, cela permet de déduire que le problème est dû à un problème de checksum (données corrompues). Par contre si l'erreur se trouve sur DONE, il s'agit plus probablement de données manquantes ou d'un grave défaut matériel.

#### 4.6 Instruction CLEAR\_MEM\_POINTER

Trame :

octet	Contenu
0	INST_CLEAR_FPGA

Cette instruction sert à remettre à zéro les pointeurs d'offset des blocs mémoire. Les mémoires étant accédées en mode FIFO : seule l'adresse de base du bloc est configurée. Puis chaque accès incrémente le pointer d'offset.

#### **4.7 Ecriture de données dans les blocs mémoires**

Pour écrire des données, il faut simplement les données dans l'EP4OUT.

Les zones mémoires concernées par ce type d'accès sont

- Mémoire duplication : 2048 octets, LSB first
- Mémoire table de vérité de sortie : 2048 octets, LSB first
- Mémoire table de vérité interne pour compteurs : 2048 octets, LSB first
- Mémoire retard largeur 40 octets, MSB first

## **5 API de contrôle de la carte**

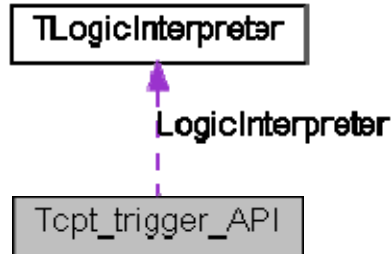
Cette interface se présente sous la forme d'une classe C++, et offre un ensemble de fonctions de haut niveau se chargeant de tous les accès matériels de manière transparente pour l'utilisateur avancé. L'interface graphique décrite au paragraphe suivant repose intégralement sur cette API. La documentation ci-dessous a été générée par Doxygen.

## 5.1 Tcpt\_trigger\_API Class Reference

API de la carte compteur trigger.

```
#include <cpt_trigger_API.h>
```

Collaboration diagram for Tcpt\_trigger\_API:



### Public Member Functions

- bool [Connect\\_USB](#) (char \*FirmwarePath)  
*Fait la connection USB avec la carte et transfere l'executable.*
- int [ConfigureFPGA](#) (char \*FPGAConfigFile)  
*Methode de configuration du FPGA.*
- bool [InitDac](#) (float Vref)  
*Methode d'initialisation des DAC.*
- bool [SetVoltDacs](#) (double \*Thres\_value)  
*Positionne les 8 seuils des discriminateurs d'entrées aux valeurs désirées.*
- bool [load\\_duplication\\_table](#) (int \*vector)  
*Charge la mémoire de duplication avec les paramètres adéquats.*
- bool [load\\_tb\\_verite\\_out\\_table](#) (unsigned char \*vector)  
*Charge la mémoire de la table de vérité avec les paramètres adéquats.*
- bool [load\\_delay\\_width\\_table](#) (int \*delay\_vector, int \*width\_vector)  
*Charge la table de retards / largeur avec les paramètres transmis.*
- bool [SetPretemps](#) (unsigned int pretemps)  
*Réglage du pretemps.*
- bool [ReadCounters](#) (int \*Vector\_counter)  
*lit les compteurs d'activités.*
- bool [mode\\_pretemps\\_select](#) (bool active)  
*Méthode d'activation du mode pretemps.*
- bool [reset\\_counters](#) ()  
*Fonction de remise à zéro de l'ensemble des compteurs.*
- bool [start\\_triggers](#) ()  
*Activation des triggers et du comptage.*
- bool [stop\\_triggers](#) ()  
*Arret des triggers et du comptage.*
- bool [Read\\_status](#) (int &status)  
*Lecture de l'état de l'électronique.*
- int [FabricationMasqueBitn](#) (int bit\_a\_tester, int bit\_a\_affecter, int address\_courante)  
*Fonction servant à remplir la mémoire duplication.*
- [Tcpt\\_trigger\\_API](#) ()  
*Constructeur de l'API.*

### Public Attributes

[TLogicInterpreter](#) [LogicInterpreter](#) [8]  
calculatrice logique.

---

## Detailed Description

API de la carte compteur trigger.

Grace à cette API il est possible de contrôler directement la carte trigger à travers l'USB. Cette API est compatible avec Windows 2000/XP (sous réserve d'installer le filter driver libusb-win32 <http://libusb-win32.sourceforge.net/>) et avec LINUX, dans ce cas il faut veiller à ce que les paquets (libusb et libusb-dev) soient bien installés (apt-get ou yum suivant la distribution).

Séquence typique d'initialisation après un redémarrage :

[Connect\\_USB\(char \\*FirmwarePath\)](#)

[ConfigureFPGA\(char \\*FPGAConfigFile\)](#)

[InitDac\(float Vref\)](#)

Séquence de configuration :

[SetVoltDacs\(double \\*Thres\\_value\)](#)

[load\\_duplication\\_table\(int \\*vector\)](#) (utiliser [FabricationMasqueBitn\(int bit a tester,int bit a affecter,int address\\_courante\)](#) pour créer le tableau)

[load\\_tb\\_verite\\_out\\_table\(unsigned char \\*vector\)](#) (utiliser la calculatrice logique [TLogicInterpreter](#) pour créer les tables de vérités)

[load\\_delay\\_width\\_table\(int \\*delay\\_vector,int \\*width\\_vector\)](#)

[SetPretemps\(unsigned int pretemps\)](#) et [mode\\_pretemps\\_select\(bool active\)](#) (si l'on souhaite fonctionner en mode prétemps)

[reset\\_counters\(\)](#)

Ensuite il suffit de démarrer et arrêter l'acquisition à volonté avec :

[start\\_triggers\(\)](#)

[stop\\_triggers\(\)](#) (si l'on souhaite fonctionner en mode prétemps il peut être intéressant de faire un polling sur

[Read\\_status\(int &status\)](#) avant de stopper)

Aussi, il est à noter que certains accès au hardware peuvent se dérouler pendant que les triggers et les compteurs sont actifs, la liste suivante est exhaustive:

[SetVoltDacs\(double \\*Thres\\_value\)](#)

[load\\_delay\\_width\\_table\(int \\*delay\\_vector,int \\*width\\_vector\)](#)

[ReadCounters\(int \\*Vector\\_counter\)](#)

[Read\\_status\(int &status\)](#)

[reset\\_counters\(\)](#)

---

## Constructor & Destructor Documentation

**Tcpt\_trigger\_API::Tcpt\_trigger\_API ()**

Constructeur de l'API.

Remet à 0 le mot de contrôle.

---

## Member Function Documentation

**bool Tcpt\_trigger\_API::Connect\_USB (char \* FirmwarePath)**

Fait la connexion USB avec la carte et transfère l'exécutable.

Cette méthode est la toute première à exécuter avant toute autre opération.

**Parameters:**

*FirmwarePath* : Chaîne de caractère contenant le nom et le chemin du fichier ASCII de programmation.

**Returns:**

true en cas de succès.

**int Tcpt\_trigger\_API::ConfigureFPGA (char \* *FPGAConfigFile*)**

Methode de configuration du FPGA.

L'appel a cette fonction met en mémoire les données de configuration fournie par *FPGAConfigFile* et génère toute la séquence de communication requise avec le hardware.

**Parameters:**

*FPGAConfigFile* : Nom et chemin du fichier contenant les données de configuration du FPGA.

**Returns:**

- 0 : Succès.
- 1 : Problème de lecture du fichier de configuration ou d'allocation mémoire.
- 2 : Echec à la requete d'effacement du FPGA.
- 3 : Echec de la requete de status de l'effacement.
- 4 : Echec de l'effacement.
- 5 : Erreur de transfert des données de configuration.
- 6 : Echec de la requete de status de configuration.
- 7 : Echec de récupération du status de la configuration.
- 8 : Erreur d'initialisation.
- 9 : Erreur de configuration (après vérification du signal DONE)

**See also:**

PutConfigInMem(char \**FPGAConfigFile*)

[INST\\_CLEAR\\_FPGA](#)

[CHECK\\_PROG\\_FPGA](#)

[FPGA\\_INIT\\_FAILURE](#)

[FPGA\\_DONE\\_FAILURE](#)

**bool Tcpt\_trigger\_API::InitDac (float *Vref*)**

Methode d'initialisation des DAC.

Permet de choisir la référence de tension qui va être utilisée par le DAC AD5392 (*Vref*). il y aussi toute une séquence d'initialisation qui est executé (soft powerup, etc). Cette méthode doit être appelée une fois avant d'utiliser les DAC et après avoir configuré le FPGA.

**Parameters:**

*Vref* : mettre 2.5 si l'on souhaite cette référence, sinon c'est 1.25

**See also:**

WriteDac()

**bool Tcpt\_trigger\_API::SetVoltDacs (double \* *Thres\_value*)**

Positionne les 8 seuils des discriminateurs d'entrées aux valeurs désirée.

Cette méthode recalcule les tensions à positionner en sortie de DAC afin d'obtenir les seuils désirés. Ensuite les mots de contrôle sont fabriqués et envoyés.

**Parameters:**

*Thres\_value* : Tableau de 8 valeurs DAC exprimées en volt. La valeur maximum est de 1V.

**See also:**

WriteDac()

**bool Tcpt\_trigger\_API::load\_duplication\_table (int \* *vector*)**

Charge la mémoire de duplication avec les paramètres adéquats.

**See also:**

Clear\_Pointers()

load\_memory\_table(unsigned char \**vector*, unsigned char address)

**bool Tcpt\_trigger\_API::load\_tb\_verite\_out\_table (unsigned char \* *vector*)**

Charge la mémoire de la table de vérité avec les paramètres adéquats.

**See also:**

Clear\_Pointers()  
load\_memory\_table(unsigned char \*vector,unsigned char address)

**bool Tcpt\_trigger\_API::load\_delay\_width\_table (int \* delay\_vector, int \* width\_vector)**

Charge la table de retards / largeur avec les paramètres transmis.

Cette méthode met en forme et transmet les données permettant de régler les 10 retards et 10 largeurs des signaux après passage par la mémoire de duplication. Pour les 2 valeurs le maximum est de  $2^{13}-1$ .

**Parameters:**

*delay\_vector* : Vecteur de taille 10 contenant les valeurs de retard exprimées en multiple de 10 ns.  
*width\_vector* : Vecteur de taille 10 contenant les valeurs de largeur exprimés en multiple de 10 ns.

**See also:**

[INST SET ADD](#)

**bool Tcpt\_trigger\_API::SetPretemps (unsigned int pretemps)**

Réglage du pretemps.

Lorsque le mode pretemps est choisi, cette valeur est utilisée pour déterminer la durée de comptage.

**Parameters:**

*pretemps* : valeur sur 32 bit dont le LSB est équivalent à une durée de 1 ms.

**See also:**

[INST SET ADD](#)  
[mode\\_pretemps\\_select\(bool active\)](#)

**bool Tcpt\_trigger\_API::ReadCounters (int \* Vector\_counter)**

lit les compteurs d'activités.

Cette méthode se charge de faire les accès nécessaires au hardware pour relever les 16 valeurs de compteurs.

**Parameters:**

*Vector\_counter* : buffer de destination pour les 16 valeurs de compteurs (24 bit significatifs).

**See also:**

[INST READ COUNTERS](#)

**bool Tcpt\_trigger\_API::mode\_pretemps\_select (bool active)**

Méthode d'activation du mode pretemps.

Un appel a cette méthode avec le paramètre approprié active ou desactive le mode pretemps.

**Parameters:**

*active* : true active le mode pretemps.

**See also:**

[SetPretemps\(unsigned int pretemps\)](#)

**bool Tcpt\_trigger\_API::reset\_counters ()**

Fonction de remise à zéro de l'ensemble des compteurs.

**See also:**

SetCTRL\_reg()

**bool Tcpt\_trigger\_API::start\_triggers ()**

Activation des triggers et du comptage.

Lorsque cette méthode est appelée :

les triggers sont autorisés

si requis, le compteur de pretemps est démarré

Les compteurs d'activités sont autorisés à compter

**See also:**

[stop\\_triggers\(\)](#)

SetCTRL\_reg()

**bool Tcpt\_trigger\_API::stop\_triggers ()**

Arrêt des triggers et du comptage.

Si le mode pretemps est utilisé, un appel à cette méthode forcera l'arrêt des compteurs d'activités et des triggers.

**See also:**

[start\\_triggers\(\)](#)

SetCTRL\_reg()

**bool Tcpt\_trigger\_API::Read\_status (int & status)**

Lecture de l'état de l'électronique.

Permet de savoir si le pretemps c'est écoulé.

**Parameters:**

*status* : 1 quand terminé, 0 encore en cours.

**See also:**

[SetPretemps\(unsigned int pretemps\)](#)

[mode\\_pretemps\\_select\(bool active\)](#)

[INST\\_READ\\_STATUS](#)

**int Tcpt\_trigger\_API::FabricationMasqueBitn (int bit\_a\_tester, int bit\_a\_affecter, int address\_courante)**

Fonction servant à remplir la mémoire duplication.

La mémoire duplication permet soit :

à dupliquer des signaux afin de pouvoir les utiliser dans des équations différentes.

à ré aiguiser un signal vers une autre voie (interne).

Pour utiliser cette fonction, il suffit de lui passer les paramètres adéquats pour chaque entrée (bit) à tester et en retour on obtient le masque à appliquer (en OR).

**Parameters:**

*bit\_a\_tester* : voie d'entrée à dupliquer ou ré aiguiser (valeur de 0 à 7).

*bit\_a\_affecter* : Numéro de la sortie mémoire à affecter (voie interne) (valeur de 0 à 9).

*address\_courante* : Vecteur représentant la combinaison courante des 8 entrées (valeur de 0 à 255).

**Returns:**

masque servant à modifier la variable courante.

---

## Member Data Documentation

**[TLogicInterpreter Tcpt\\_trigger\\_API::LogicInterpreter](#)[8]**

calculatrice logique.

Interprete les équations logiques et fournit les contenus des tables de vérités.

---

## 5.2 TLogicInterpreter Class Reference

Classe offrant toutes les méthodes permettant de faire un calcul logique à partir d'une chaîne de caractères.

```
#include <LogicInterpreter.h>
```

### Public Member Functions

string [GetStatus](#) ()

*Fonction permettant de récupérer le dernier message d'erreur en cours.*

bool [CleanString](#) (string Op)

*Fonction de nettoyage et de recherche d'erreur de syntaxe.*

bool [ComputeAgainstFormula](#) (int in\_vector)

*Formule de calcul d'une seule table de vérité.*

unsigned char [ComputeAgainstFormula](#) (int in\_vector, int bit\_a\_affecter, unsigned char masque)

*Fonction de calcul.*

int [ComputeAgainstFormula](#) (int in\_vector, int bit\_a\_affecter, int masque)

*Fonction de calcul.*

[TLogicInterpreter](#) ()

*Constructeur de la classe.*

---

### Detailed Description

Classe offrant toutes les méthodes permettant de faire un calcul logique à partir d'une chaîne de caractères.

Elle fonctionne indifféremment sous Linux et sous windows. Pour l'utiliser il suffit :

De déclarer la classe

```
TLogicInterpreter LogicInterpreter
```

D'interpréter la formule

```
LogicInterpreter.CleanString(formule)
```

Ensuite et autant de fois que nécessaire évaluer la formule au regard d'une combinaison d'entrée

```
LogicInterpreter.ComputeAgainstFormula(i)
```

Voir l'exemple de code complet fournit ci-dessous.

```
#include <string>
#include <iostream>

using namespace std;
#include "LogicInterpreter.h"

int main()
{
    string saisie;
    bool done=false;
    TLogicInterpreter LogicInterpreter;
    cout << "calculatrice logique" <<endl;

    do
        {
            cout << "entrez la formule a evaluer:" <<endl;
            getline(cin,saisie);

            if (saisie.find("q")==0)
                break;

            if (!LogicInterpreter.CleanString(saisie))
                {
                    cout << LogicInterpreter.GetStatus() <<endl;
                }
            else
```

```

    {
    cout << "chaîne a traiter : " << LogicInterpreter.GetStatus() << endl;
    for (int i =0;i<16;i++)
        {
            for (int j=9;j>=0;j--)
                {
                    if ((1<<j) & i)
                        cout << 1;
                    else
                        cout << 0;
                }
            cout << " sortie binaire: ";
            cout << LogicInterpreter.ComputeAgainstFormula(i);
        }
    } while(!done);

cout << "au revoir" << endl;

return 0;
}

```

Les opérateurs reconnus sont :

and  
or  
not()  
xor  
xnor  
nand  
nor  
sup()

Les opérandes reconnus sont :

i0  
i1  
i2  
i3  
i4  
i5  
i6  
i7  
i8  
i9

Quelques exemples de formules interprétables par cette classe:

i0 and i1 or i2  
(i0 and i1) or i2  
((i0 and i1)) or i2  
not(i0) and i2  
not (i1 and i0) xor i2  
sup (i0 + i1 +i2;2) (cherche une multiplicité de 2 parmi i0,i1,i2)  
sup ((i0 and i2) + i1 +i2;2)

Erreurs d'interprétation de formule possible (cas d'erreurs testés) :

pas de suffisamment parenthèses apparentées  
opérateur/opérande inconnu  
2 ou plusieurs opérateurs (ou opérandes) successifs  
opérateur binaire en début ou en fin de formule (ou accolé à une parenthèse)  
opérateur unitaire utilisé de façon inappropriée (sans parenthèse, mauvaise syntaxe interne)

Si dans une phase de debug, des messages supplémentaires sont souhaités, il suffit de positionner DEBUG lors de la compilation.

---

## Constructor & Destructor Documentation

### TLogicInterpreter::TLogicInterpreter ()

Constructeur de la classe.

C'est ici que sont définies toutes les équivalences entre opérateurs/opérandes externes et les opérateurs/opérandes internes. En effet pour simplifier l'interprétation d'une formule, il faut que tous les opérandes et tous les opérateurs aient la même taille, à savoir 1 caractère.

---

## Member Function Documentation

### string TLogicInterpreter::GetStatus ()

Fonction permettant de récupérer le dernier message d'erreur en cours.

### bool TLogicInterpreter::CleanString (string Op)

Fonction de nettoyage et de recherche d'erreur de syntaxe.

Cette fonction est la première à exécuter sur une nouvelle chaîne de caractère. Elle va tout d'abord rechercher les erreurs dans la formule passée en argument, puis elle va procéder à toute une série de remplacement et d'effacement de caractères afin d'avoir une formule plus facilement utilisable par la calculatrice.

#### Parameters:

*Op* : Chaîne de caractère contenant la formule logique à évaluer.

#### See also:

[GetStatus\(\)](#)

### bool TLogicInterpreter::ComputeAgainstFormula (int in\_vector)

Formule de calcul d'une seule table de vérité.

Cette fonction est dédiée à fournir le résultat de l'évaluation de la fonction avec la combinaison d'entrée. Elle fournit un résultat binaire pour une seule sortie.

#### Parameters:

*in\_vector* : Combinaison logique d'entrée à tester (point d'entrée de la table de vérité)

### unsigned char TLogicInterpreter::ComputeAgainstFormula (int in\_vector, int bit\_a\_affecter, unsigned char masque)

Fonction de calcul.

Cette fonction est un wrap up de la méthode [ComputeAgainstFormula\(int in\\_vector\)](#), car elle permet d'évaluer le résultat de la formule au regard du vecteur d'entrée, mais aussi de modifier le bit approprié dans une table de vérité gérant déjà 8 sorties (d'où le type du paramètre de sortie).

#### Parameters:

*in\_vector* : Combinaison logique d'entrée à tester (point d'entrée de la table de vérité)

*bit\_a\_affecter* : Sortie à positionner en fonction du résultat de la formule et de la combinaison d'entrée

*masque* : octet dont un des bit est à modifier

#### See also:

[CleanString\(string Op\)](#)

### int TLogicInterpreter::ComputeAgainstFormula (int in\_vector, int bit\_a\_affecter, int masque)

Fonction de calcul.

Cette fonction est un wrap up de la méthode [ComputeAgainstFormula\(int in\\_vector\)](#), car elle permet d'évaluer le résultat de la formule au regard du vecteur d'entrée, mais aussi de modifier le bit approprié dans une table de vérité gérant déjà 32 sorties (d'où le type du paramètre de sortie).

**Parameters:**

*in\_vector* : Combinaison logique d'entrée à tester (point d'entrée de la table de vérité)

*bit\_a\_affecter* : Sortie à positionner en fonction du résultat de la formule et de la combinaison d'entrée

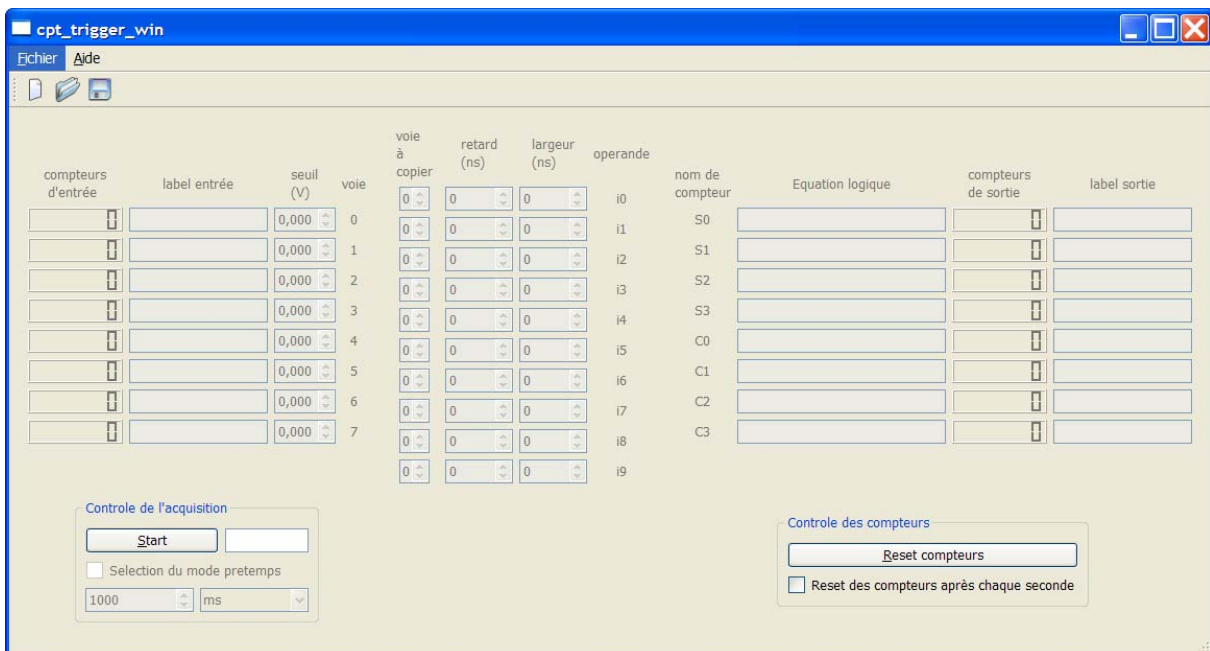
*masque* : entier dont un des bit est à modifier

**See also:**

[CleanString\(string Op\)](#)

## 6 Interface graphique de contrôle / manuel utilisateur

L'interface graphique a été conçue à l'aide de QT4.3.2 afin de rester portable entre les plateformes Windows, Linux et MacOSX. Dans cette partie, seul l'aspect interface utilisateur va être décrit.



Description en fonction des colonnes

- **Compteurs d'entrée** : Valeur des 8 compteurs d'entrée. Mis à jour chaque seconde, contient soit la valeur intégrée depuis le début du run, soit la valeur vue en 1 seconde.
- **Label entrée** : Nom unique pouvant être attribué à chaque voie d'entrée.
- **Seuil** : valeur de chaque seuil d'entrée, réglable entre -1V et 0V avec une résolution de 14 bit, soit un LSB à 60  $\mu$ V. Ces champs peuvent être mis à jour pendant l'acquisition.
- **Voie** : C'est un rappel du numéro de voie « physique » associée à la ligne.
- **Voie à copier** : c'est le numéro de voie physique à utiliser dans la ligne courante pour faire la mise en forme. Cela permet aussi l'association de l'opérande « interne » à l'entrée physique choisie.
- **Retard** : retard exprimé en nanoseconde, la valeur maximum est de 81920 ns. La valeur réelle de retard appliquée est la plus proche en multiples de 10 ns.
- **Largeur** : largeur exprimée en nanoseconde, la valeur minimum est de 10 ns et la valeur maximum est de 81920 ns. Concrètement, la valeur réelle de largeur appliquée est la plus proche en multiples de 10 ns.

- **Operande** : Indique le nom de l'opérande associé à l'entrée physique sélectionnée et remise en forme.
- **Nom de compteur** : Rappelle qu'il y a 4 compteurs qui comptent les activités des sorties (Sx) et quatre compteurs qui comptent les activités de signaux internes (Cx).
- **Equation logique** : C'est dans ces champs que sont saisies les équations associées à chaque compteur
- **Compteurs de sortie** : Valeur des 8 compteurs de sortie. Mis à jour chaque seconde, contient soit la valeur intégrée depuis le début du run, soit la valeur vue en 1 seconde.
- **Label sortie** : Nom unique pouvant être attribué à chaque voie de sortie.

Le groupe de contrôle acquisition permet de démarrer ou arrêter l'acquisition. Le champ présent à coté du bouton start/stop sert à présenter le temps écoulé depuis le début du run. Si le mode pré temps est désiré, il suffit de cocher la case l'activant et saisir la durée désirée ainsi que son unité. Si l'acquisition est lancée dans ce mode, le champ de saisie du pretemps va se décrémenter pour chaque échéance d'une unité.

Le groupe contrôle des compteurs permet soit :

- de faire une remise à zéro manuelle des compteurs
- de faire une remise à zéro automatique des compteurs chaque seconde.

Remarques :

- Toutes les valeurs de champs (à l'exception des valeurs de compteurs) et équations peuvent être sauvegardées dans un fichier de paramètres afin de ne pas avoir à les ressaisir à chaque nouveau démarrage
- Tous les champs pouvant être mis à jour sans stopper l'acquisition sont automatiquement transmis à l'électronique.
- Toute modification de paramètre est détectée et en cas de fermeture de l'application, une sauvegarde est proposée.

## ICI EXEMPLE SIMPLE

## 7 Annexes

### 7.1 Installation sur une plateforme Windows

Fichiers fournis :

Fichier « setup.exe » → contient le logiciel de contrôle et quelques exemples

Fichier « libusb-win32-filter-bin-20041118.exe » → installe le « Filter Driver ».

Comment installer les logiciels sur le PC :

- 1) Ne pas connecter la carte avant la procédure suivante
- 2) Lancer le fichier « Libusb-win32 » en vous assurant qu'un composant USB quelconque (souris,...) soit connecté à votre PC. Une fois l'installation terminée, pour vérifier le bon fonctionnement du « Filter Driver », lancer « Test Program » (présent dans vos programmes). Si le programme détecte votre composant USB, c'est que l'installation du « Filter Driver » s'est bien déroulée.

- 3) Lancer setup.exe
- 4) Une fois l'installation du logiciel terminée, connecter l'interface à votre PC et patienter le temps qu'il identifie le dispositif.
- 5) Un assistant d'installation s'ouvre. Refuser « la recherche des mises à jour » (cocher non). Ensuite cocher « Installer à partir d'une liste ». La page qui suit, décocher « Rechercher dans les médias » et cocher « Inclure cet emplacement ». Cliquer sur « parcourir » et sélectionner l'emplacement d'installation de votre application. Cliquer ensuite sur « Continuer » (driver non signé par Windows). Et pour finir cliquer sur « Terminer ».

## ***7.2 Installation sur une plateforme LINUX***

A venir

## ***7.3 Installation sur une plateforme MAC***

A venir