HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Prof. Yann Thoma

Laboratoire VSN

semestre de printemps 2017 - 2018

Banc de test pour système synchrone : Détection de spike

Introduction

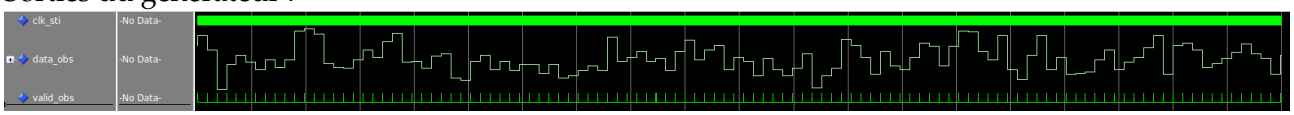
Dans le cadre d'un projet de recherche, nous disposons d'un système capable de détecter des signaux électriques dans un puit contenant des neurones réels. Ces neurones émettent de temps en temps un signal qui s'appelle un *spike* et correspond à l'activation du neurone. Pour les biologistes, l'intérêt principal des expériences qu'ils effectuent est la détection de ces spikes.
Dans ce contexte, ce laboratoire vise vérification du bloc réalisant la détection ainsi que l'extraction de zones d'interêts d'un signal autour d'un spike. Le cahier des charges ci-dessous était destiné à des étudiants devant réaliser le système. Dans ce cahier des charges, le terme *vous* se réfère donc aux designers du système. Le travail à faire pour vous est décrit plus loin, dans les étapes.

Cahier des charges

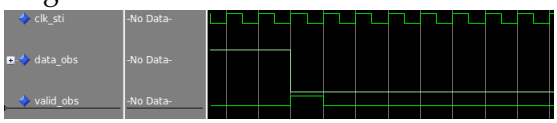
Vous allez devoir réaliser une architecture capable d'effectuer la détection de Spikes et permettre l'enregistrement des données dans un block RAM. L'entité que vous devez réaliser devra être la suivante :

En entrée :
Vous allez recevoir un signal qui correspond à des données mesurées dans un puit de neurones. A chaque fois qu'un échantillon est généré, le générateur vous informe qu'il doit être pris en compte grâce à un flag de validité (qui dure une période d'horloge). Le générateur fourni les datas qui viendront connectées sur l'entrée `sample_i` de votre bloc et vous donnera le flag valid sur votre entrée `sample_valid_i`.
Le signal `ready_o` vous permet de bloquer l'envoi, et donc le sample n'est consommé que si les deux signaux `sample_valid_i` et `ready_o` valent `'1'`. Pour illustrer le fonctionnement, voici les signaux générés par le générateur fourni :

Sorties du générateur :

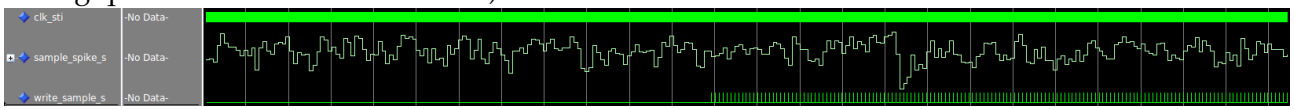


Flag valid zoomé :



En sortie :

Vous allez devoir mettre à disposition les données à enregistrer et nous indiquer quand nous devons les écrire en RAM. Il faudra simplement mettre votre échantillon à enregistrer en sortie et activer le `samples_spikes_valid_o` pendant 1 cycle d'horloge. De plus il faudra signaler qu'il s'agit du dernier échantillon lorsque c'est le cas. Ceci doit être fait en plaçant `spike_detected_o` à '1' pendant le dernier cycle. Le signal que vous aller ainsi généré ressemblera à cela (avec une représentation analogique des valeurs d'échantillons) :



Processing de Spikes

Détection de Spikes

Afin d'avoir un système robuste et qui s'adapte largement plus au bruit du signal qu'un simple threshold, vous allez devoir utiliser le concept suivant : nous allons regarder si notre échantillon est plus grand qu'un certains nombre de fois (facteur variable) la déviation standard du signal.
Pour ce faire, nous allons utiliser le développement mathématique suivant (qui offre une simplification intéressante pour l'implémentation sur FPGA).
La formule de base que nous allons poser consiste en la comparaison entre l'échantillon actuel, la moyenne glissante et un certain facteur multiplié par la déviation standard. Nous allons ainsi poser :

$$Deviation = |Echantillon - Moyenne| > Deviationstandard * Facteur$$

$$Deviation_t = |x_t - \bar{x}_t| > s * C$$

Premièrement, nous allons devoir calculer la moyenne glissante de notre signal. Cette moyenne glissante se fait sur un certains nombre de point. Dans notre cas, nous allons définir que la moyenne glissante se fait sur 128 points (comme nous aurons une division à faire, l'aligner sur une puissance de deux permet de faire un shift (cablâge) au lieu d'une division). Afin de calculer la moyenne glissante sur N points, nous allons partir de la formule de base suivante :

$$\bar{x}_t = (\sum_{i=t-N+1}^{t} x_i)/N$$

Afin de ne pas stocker la valeur des échantillons pour ensuite les soustraires à la moyenne glissante, nous allons approximer l'échantillon au temps $t - N$ par l'échantillon actuel (cela provoque une erreur qui serait égale à la différence entre l'échantillon actuel et l'échantillon à $t - N$ divisée par la taille de la fenêtre, mais comme N est relativement grand, nous acceptons cette erreur car elle nous permet de ne pas stocker les valeurs des échantillons dans le temps. On obtient ainsi une formule approximative :

$$\bar{x}_{t+1} \approx \bar{x}_t + (x_{t+1})/N - (\bar{x}_t)/N$$

Où on peut encore simplifier par :

$$\bar{x}_{t+1} \approx \bar{x}_t + ((x_{t+1}) - (\bar{x}_t))/N$$

Nous avons à présent la formule pour la moyenne glissante approximée que vous allez devoir convertir en hardware.

En exploitant cette moyenne est il est alors possible de calculer la déviation standard des échantillons, qui correspond à :

$$s_t = \sqrt{\frac{\sum_{i=t-N+1}^{t}(x_i - \bar{x}_t)^2}{N}}$$

NOTE : il est important de noter que la formule divise par N alors que la formule de la déviation standard divise par $N-1$. Ceci est un choix afin de pouvoir faire une division câblée comme au point précédent. Afin de s'autoriser à utiliser la formule sous cette forme, il faut calculer l'erreur possible. Celle-ci est de $1/128$ ce qui est moins de 1%, ce que nous avons admis comme acceptable.

Afin d'éviter de devoir faire une racine carrée dans la FPGA, nous allons élever cette formule au carré. On obtient (après simplification) :

$$s_t^2 = (\frac{1}{N}\sum_{i=t-N+1}^{t} x_i^2) - \bar{x}_t^2$$

Avec $\sum x_i^2$ qui correspond à la somme des carrés défini par :

$$Sum_t^2 = \sum_{i=t-N+1}^{t} x_i^2 = x_t^2 - \frac{Sum_{t-1}^2}{N} + Sum_{t-1}^2$$

Afin de s'adapter à la modification sur le calcul de la déviation standard, la formule de base pour la détection de Spike deviendra alors :

$$Spike =' 1' \Leftrightarrow D_t^2 > s^2 * C^2$$

Où :

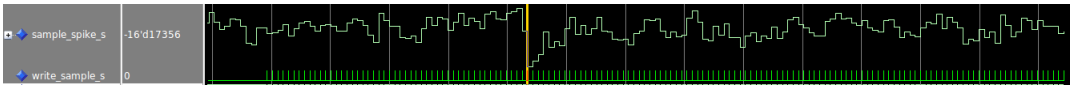
- `D` : différence entre échantillon et moyenne flottante
- `s` : déviation standard
- `C` : facteur de détection

Extraction de Spikes

Pour réaliser ce laboratoire, vous allez devoir trouver un spike dans le signal généré. Les biologistes sont intéressés à enregistrer un certain nombre d'échantillons autour du spike. Nous avons défini qu'une fenêtre de 5 ms était nécessaire pour avoir une bonne visibilité du spike. Comme le signal que nous recevons génère des signaux à 30kHz, on peut calculer le nombre de point dans notre fenêtre, soit :

$$N = WindowDuration * SamplingFreq = 0.005 * 30000 = 150$$

Nous avons ainsi défini le nombre de point que nous devons enregistrer dans le block RAM.
Afin de pouvoir avoir une bonne visibilité du Spike, sa position dans l'écran doit être définie. Cette valeur est paramètrable dans la version de SpikeOnChip mais dans ce laboratoire nous allons la fixer à 50, ce qui nous amène à une visibilité tout à fait correcte (1/3 de la fenêtre).
Nous aurons ainsi un signal qui sera enregistré comme l'image présentée ci-dessous :



La barre verticale représente la position du Spike dans la fenêtre. Il y a donc 49 échantillons avant l'échantillon du spike et 100 après (ce qui fait les 150).

Visualisation du comportement du système

Dans les fichiers qui vous sont fournis, vous trouverez un dossier nommé *visualisation*. Ce dossier vous permet de voir quel est le comportement du système au travers d'un stimulus basique. Pour lancer le système avec un fonctionnement normal, placez-vous dans le dossier *comp* et tapez la commande :

```
> vsim -do ../scripts/sim.do
```

Ceci va vous permettre de lancer la simulation du système. Ce dernier va simplement vérifier qu'il reçoit **4 spikes détectés** en moins de **500 ms** sinon il échoue la simulation. Le test bench reçoit un paramètre générique *ERRNO* afin de réagir différement durant la simulation. Vous pouvez tester les différents code d'erreur (de 0 à 15) grâce à la commande :

```
> do ../scripts/sim.do sim X
```

Avec X comme valeur de ERRNO.
Afin de voir quels sont les tests qui pass ou fail ce test bench très basique, vous pouvez utiliser la commande **vrun** vue précédemment au cours. Un fichier de configuration est présent dans le dossier *visualisation*. Pour lancer le manager, placez-vous dans le dossier *visualisation* et tapez la commande :

```
> vrun -gui directed
```

Ceci va lancer 17 tests avec des paramètres génériques différents pour chacun. Vous aller ensuite voir lesquels fail et lesquels pass, sachant que le test bench ne test rien d'autre que le nombre de spike détectés en un certain laps de temps.

Vérification du système

Pour la vérification du système nous allons mettre en place un banc de test exploitant des transactions (TLM). Son architecture vous est fournie, dans le dossier `code`. Elle comprend un banc de test instanciant le DUV et lançant les éléments de vérification sous forme de lancement de procédures.
Deux agents sont présents, un pour l'entrée et un pour la sortie. Celui de l'entrée possède un séquenceur, un driver et un moniteur, alors que celui de sortie seulement un moniteur. Un scoreboard général est également présent, et est responsable de récupérer les transactions générées par les deux moniteurs.
Les différents éléments du banc de test ne sont évidemment pas complets, et ce sera à vous de les faire fonctionner correctement.
Les différents points que vous devrez résoudre sont les suivants :

1. Définir quelles sont les transactions d'entrée/sortie (fichier `transaction_pkg.vhd`)
2. Réaliser un séquenceur générant des transactions pertinentes (fichier `agent0_pkg.vhd`)
3. Réaliser un driver capable de jouer ces transactions (fichier `agent0_pkg.vhd`)
4. Réaliser un moniteur d'entrée capable de récupérer les transactions jouées pour les envoyer au scoreboard (fichier `agent0_pkg.vhd`)
5. Réaliser un moniteur de sortie capable de récupérer les transactions de sortie pour les envoyer au scoreboard (fichier `agent1_pkg.vhd`)
6. Réaliser un scoreboard capable de vérifier que les transactions de sortie correspondent à ce que nous attendons

Un paramètre générique, `ERRNO` permet de vérifier différents designs. Si sa valeur est comprise entre 0 et 4, le DUV est correct, et si elle est comprise entre 5 et 15, alors le comportement n'est pas correct (du moins, pas totalement).
Un fichier `default.rmdb` vous permet de lancer le Verification Run Manager automatiquement.

A rendre

L'ensemble de votre projet est à rendre sur Cyberlearn. Un mini rapport est également demandé. Il devra notamment comprendre une description de vos choix architecturaux.