Rapport de TP - ESN12

Davy MILLION
M2 SETSIS

20 janvier 2023

1 Introduction

Ce *lab* final nous propose de réaliser un système complet basé sur une carte DE10-Lite, qui possède un accéléromètre *ADXL345* (sur bus I2C) et 6 afficheurs 7-segments, et ainsi qu'une *IP Open Source* `OpenCore I2C`. Ce système devra périodiquement (toutes les 1 secondes) afficher la valeur de l'accélération (sur les afficheurs) mesuré le long d'un des axes (X, Y ou Z) ; l'appui sur un des boutons poussoirs (`KEY1` dans notre cas) nous permettant de changer l'axe sélectionné.

2 Travail effectué

2.1 Synthèse de la solution

Le système *SoPC* réalisé s'appuie sur 5 composants principaux :

- un *timer*, cadencé à une certaine période (1 seconde) et levant périodiquement une interruption ;
- un *Parallel I/O* (nommé `pio_button`) branché sur le bouton poussoir (donc configuré en entrée, registre sur 1-bit) afin de permettre au Nios II de récupérer sa valeur via l'interface `Avalon-MM` ;
- un *Parallel I/O* (nommé `pio_displayers`) en sortie afin d'envoyer les signaux "qui vont bien" aux afficheurs. Il est intéressant de fournir une interface de programmation simpliste à l'utilisateur (le programmeur) afin que celui-ci puisse programmer les afficheurs un par un. Pour ce faire, nous avons fixé la taille de ce `PIO` à 30-bit :
  - 24 bits ($6 * 4$-bit) pour l'écriture des caractères `BCD` (4-bit donc) sur les 6 afficheurs, il s'agit donc du "flux caractère" ;
  - 6 bits pour le contrôle individuel du segment "*point*"/"*virgule*" des 6 afficheurs.
- l'*IP* `OpenCore I2C`, qui fait office de contrôleur I2C (*Master*, dans le circuit fermé composé de lui-même et de l'accéléromètre) et de *Slave* `Avalon-MM` (côté *SoPC*, Nios-II) ;
- un softcore Nios-II réagissant aux interruptions (provoquées par `pio_button`), traitant les données de l'accéléromètre (incluant notamment la gestion des trames I2C et le calcul des accélérations en prenant en compte le facteur d'échelle), ainsi que le formatage de la donnée en vue de la restitution via les afficheurs (`pio_displayers`).

Précisons également que nous avons été généreux quant au dimensionnement de la *on-chip memory* (160KB, saturant la partie *memory bits* du Max10 à hauteur de 77%) afin d'éviter tout problème lors de la compilation du *soft* en version *debug*, le code étant pourvue dans ce cas d'instruction du type `printf("%f",my_float)`.

Ci-dessous figure le système Qsys, tels que présenté dans cette note de synthèse :

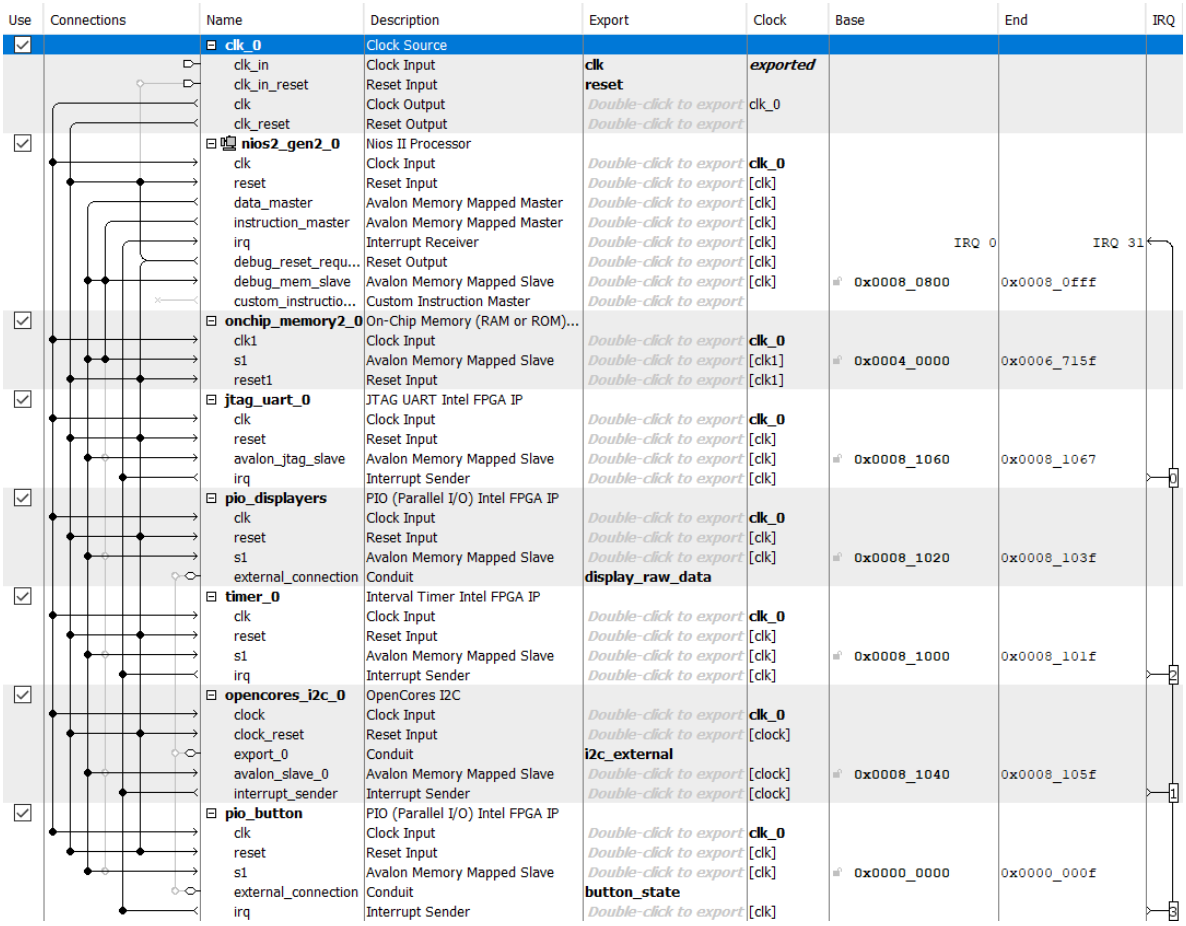

FIGURE 1 – Disposition interne des différents composants du *SoPC*

2.2 Auto-calibration du capteur

Le capteur de l'accéléromètre accumule de l'erreur lors de sa vie. Il est donc nécessaire de la prendre en compte lorsque les mesures d'accélérations sont réalisées. Afin de rendre la partie *software* plus "portable" [1], nous avons inclus une fonction d'auto-calibration, exécutée à chaque *reset* (`KEY0`). Cette fonction réalise un ensemble de mesure, en calcul les moyennes (sur les 3 axes) puis les *offsets* à appliquer. Lors de l'exécution de la calibration, il est important que l'ADXL345 soit orienté dans sa position de mesure, i.e. la position telle que :

- $Acc_x = 0$
- $Acc_y = 0$
- $Acc_z = g$

Avec la carte DE10-lite, cette position s'obtient en posant tout simplement la carte à plat sur une surface non inclinée.

En pratique, nous avons constaté que cette fonction ne calibrait pas assez précisément : il faudrait pour se faire réaliser plus de mesures en insérant une temporisation plus importante entre chaque mesure.

1. sans rentrer les coefficients en "dur"

2.3 Côté *Software*

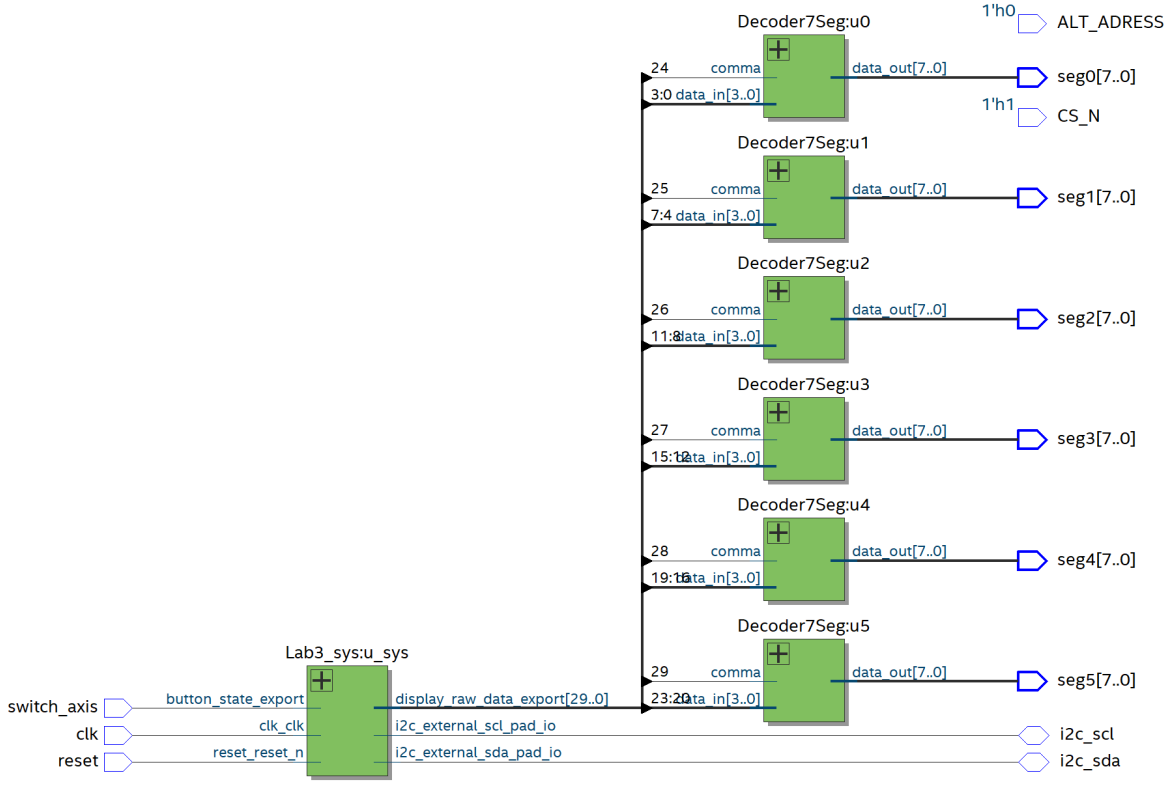

FIGURE 2 – Schéma bloc du système QSYS intégré avec entrées/sorties

Le *soft* du système est répartis sur 3 modules `C` :

- `System.c` : ce module comporte la fonction `main` qui :
  - initialise les différents composants (contrôleur I2C, *timer*, *PIO*...) du *SoPC* ;
  - interroge le registre `DATA_FORMAT` de l'ADXL et calcule le facteur d'échelle ;
  - exécute l'auto-calibration de l'accéléromètre (mesure, calcul et écriture des offsets) ;
  - lance le *timer* (provoquant une `IRQ` à chaque *timeout*).

  Il comporte également la fonction `main_irq` qui traite nos 2 interruptions :
  - celle provenant du `timer`, cadençant la lecture/écriture des accélérations de l'ADXL sur les afficheurs ;
  - celle provenant du *PIO* en entrée (`KEY1`), signalant un changement d'axe.
- `ADXL345.c` : il comporte les fonctions de base afin de récupérer les valeurs d'accélérations de l'ADXL345 ;
- `Display.c` : il comporte des fonctions, des macros de façon à fournir un certains niveau d'abstraction pour l'écriture des informations sur les afficheurs.

Par défaut, aucun *log* n'est reporté sur le `JTAG`. Toutefois, deux macros ont été prévues (fournissant 2 niveaux différents de verbosité) afin d'assurer le débuggage pour le programmeur. Ces macros sont `ADXL345_DEBUG` et `ADXL345_FULL_DEBUG`. Elles doivent être définies avant l'inclusion du fichier *header* `ADXL345.h` dans `ADXL345.c` (car le *makefile* généré ne tient pas compte d'une modification de `ADXL345.h` lors de la génération de la cible `ADXL345.o`).

3 Conclusion

Ce *lab* nous proposait d'implémenter un système complet afin de piloter/afficher les valeurs d'accélérations provenant d'un accéléromètre. Il nous a permit d'appréhender la conception d'un *SoPC* en incluant une *IP* externe.