Projet de 3e année ESE

Réalisation d'un driver pour piloter un écran tactile

Réalisé par :

Chaimaa BAYOUMI et Quentin COMTE-GAZ

Encadré par :

M. Lounis KESSAL

Année : 2013 - 2014

SOMMAIRE

INTRODUCTION

Lors de la présentation des mini-projets de troisième année, parmi la liste de tous les sujets proposés, notre choix s'est porté sur celui de la réalisation d'un driver pour écran tactile.

En effet, il nous a semblé important et surtout intéressant d'en savoir un peu plus sur le sujet. Ce rapport sera donc l'occasion de revenir sur ce que l'on a pu effectuer au cours des 10 séances consacrées à la réalisation de ce projet.

Nous allons alors, dans un premier temps, commencer par décrire le contexte du projet. Il s'agira ici d'étudier le cahier des charges en définissant les différents objectifs à atteindre mais également de présenter l'environnement de développement dans lequel le travail a été effectué.

Nous poursuivrons par la suite, en identifiant et en analysant les différents blocs constituant l'architecture du système.

Puis, nous nous intéresserons plus en détails aux caractéristiques techniques de l'écran tactile ainsi que de l'implémentation du projet.

Enfin, nous terminerons en exposant les principales difficultés que nous avons pu rencontrer durant les séances.

I. CONTEXTE DU PROJET

1) Le cahier des charges

L'objectif du projet est de concevoir un driver pour un écran tactile de type LTM pouvant être utilisé par de nombreuses cartes FPGA. Dans notre cas et dans tout ce qui suivra, ce projet sera testé sur une carte FPGA Altera DE2-70.

Plus concrètement, le but est de pouvoir récupérer, via interruption, les coordonnées (X et Y) d'un appui sur l'écran.

Dès lors, ce projet peut alors se décomposer en trois phases majeures :

- Tout d'abord, la prise en main et la compréhension de l'outil Quartus/QSys en manipulant des circuits logiques simples. Cette étape consiste également à se familiariser avec l'écran tactile.
- Il s'agira ensuite d'implémenter une version permettant d'afficher les coordonnées de l'écran sur les afficheurs 7 segments de la carte FPGA.
- Puis finalement, viendra le développement d'une interface et d'un driver permettant d'intégrer l'écran tactile.

2) L'environnement de développement

Le projet consiste à mener conjointement le développement du système en prenant en compte deux aspects : matériel (hardware) et logiciel (software).

Les principaux logiciels et environnements de développement utilisés ont donc été Quartus, QSys, Eclipse Indigo ainsi que la plateforme FPGA (Altera 2).

• Quartus

Ce logiciel permet de faire une saisie graphique ou une description HDL (VHDL ou verilog) d'architecture numérique, d'en réaliser une simulation, une synthèse et une implémentation sur cible reprogrammable.

L'utilisation de ce logiciel a fait partie intégrante de notre projet. En effet, notons qu'il est possible de concevoir le projet complet presque uniquement en VHDL. Il pouvait en être autrement mais c'est un choix que l'on a fait.

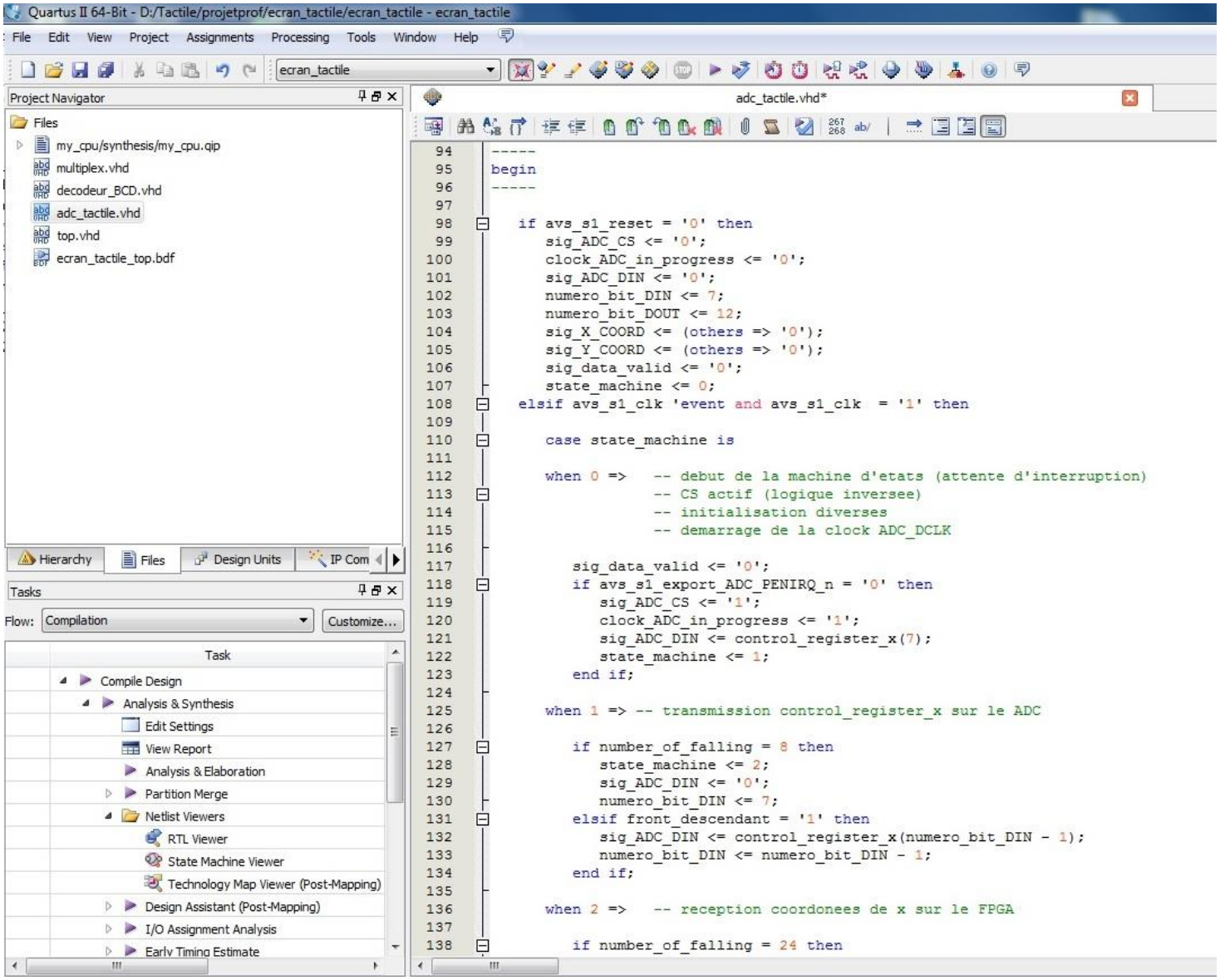


Figure 1 : Vue d'ensemble de Quartus

- **Qsys**

Cet utilitaire permet d'ajouter et de relier des composants d'une liste entre eux (Processeurs, mémoires, LEDs, Interface UART, …). Il est également possible d'ajouter nous-même des composants (et les drivers associés) à cette liste.

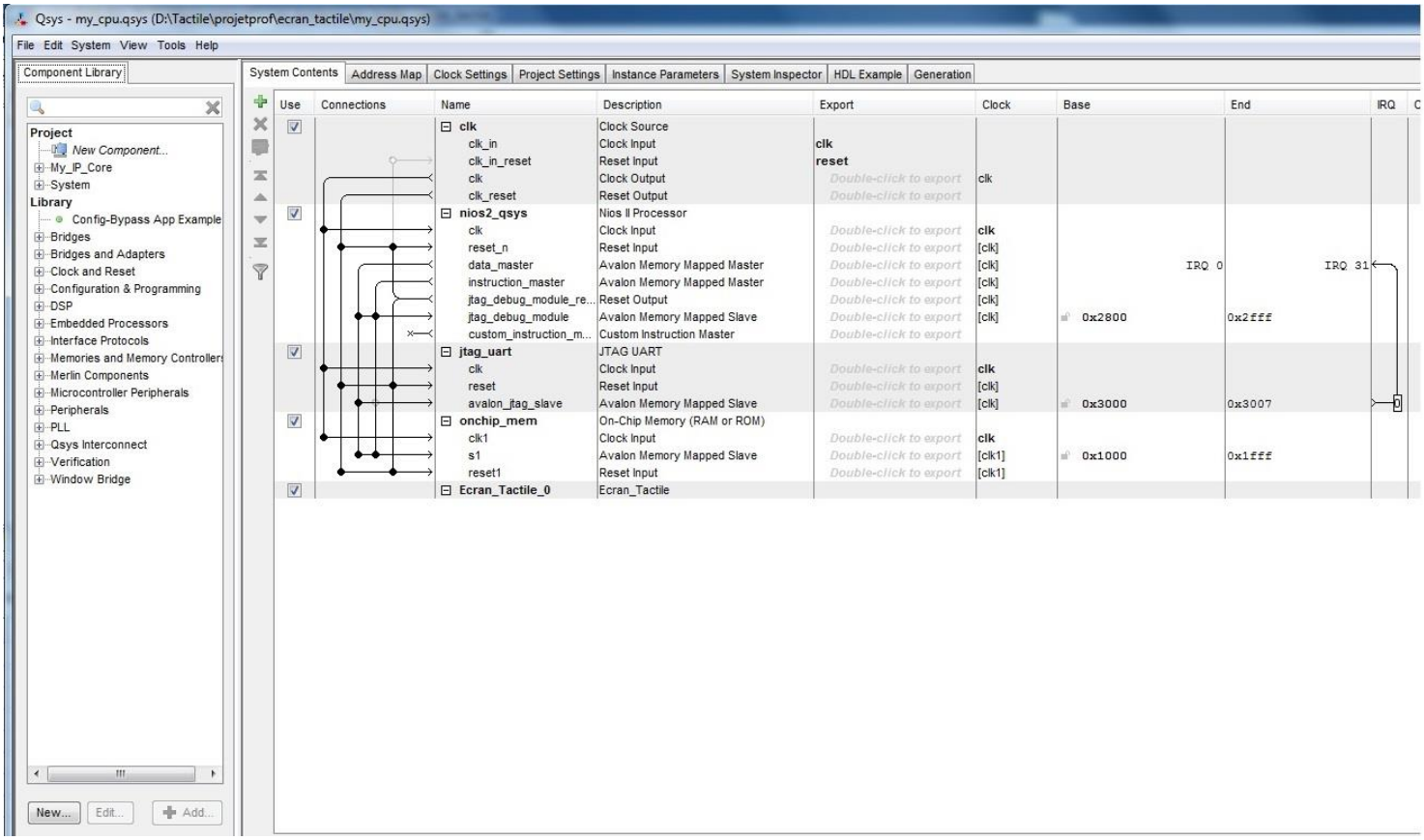


Figure 2 : Vue d'ensemble de *QSys*

- **L'environnement Nios II EDS d'Altera (version Eclipse)**

Il permet de concevoir la partie applicatif du projet. Notons que la partie « VHDL » faite sur QSys et Quartus permet de générer un code automatique contenant les différents drivers utiles directement dans Eclipse!

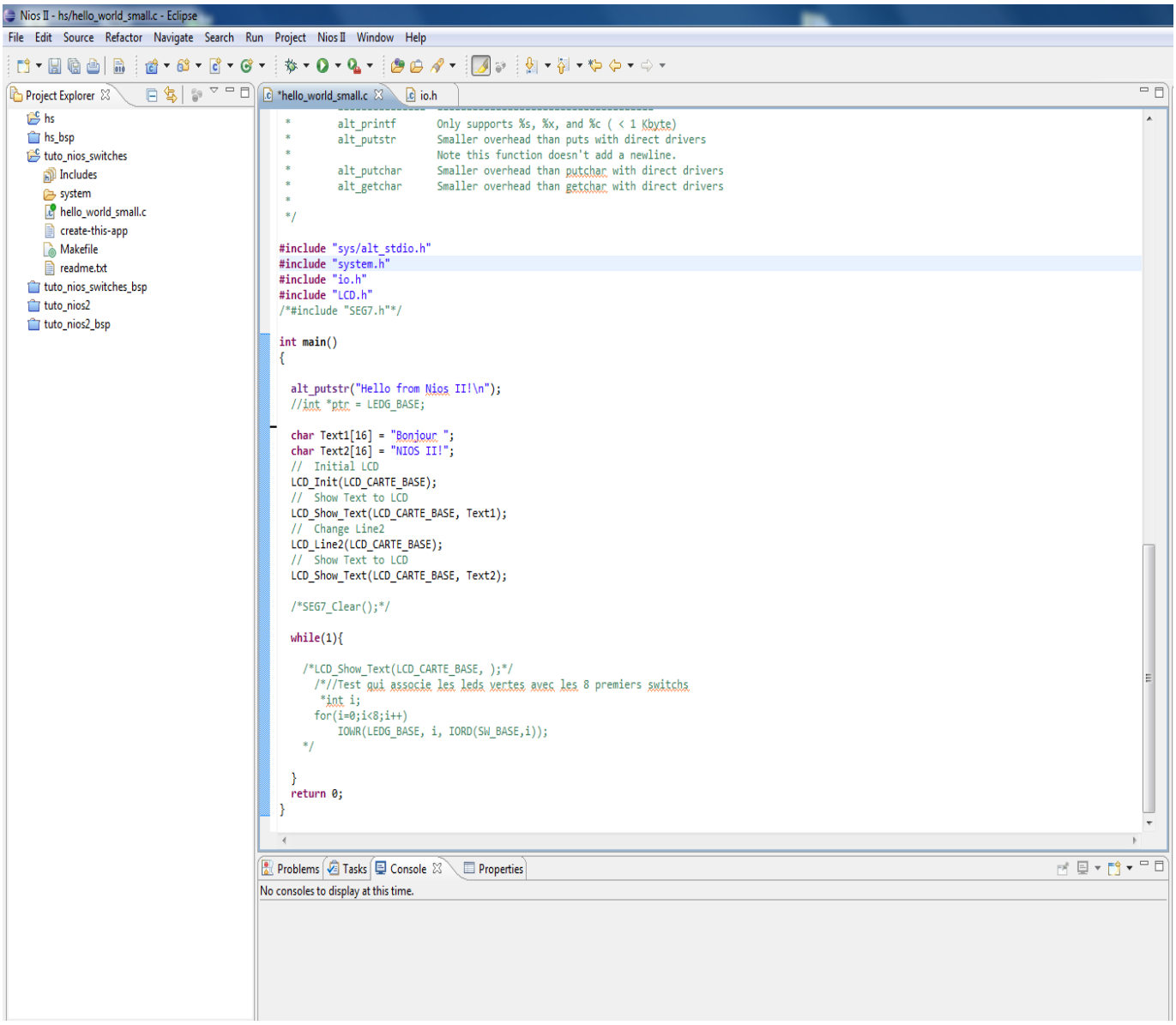


Figure 3 : Eclipse

- **FPGA (Altera 2)**

C'est la cible reprogrammable. C'est sur cette dernière que sera implémenté le code écrit.

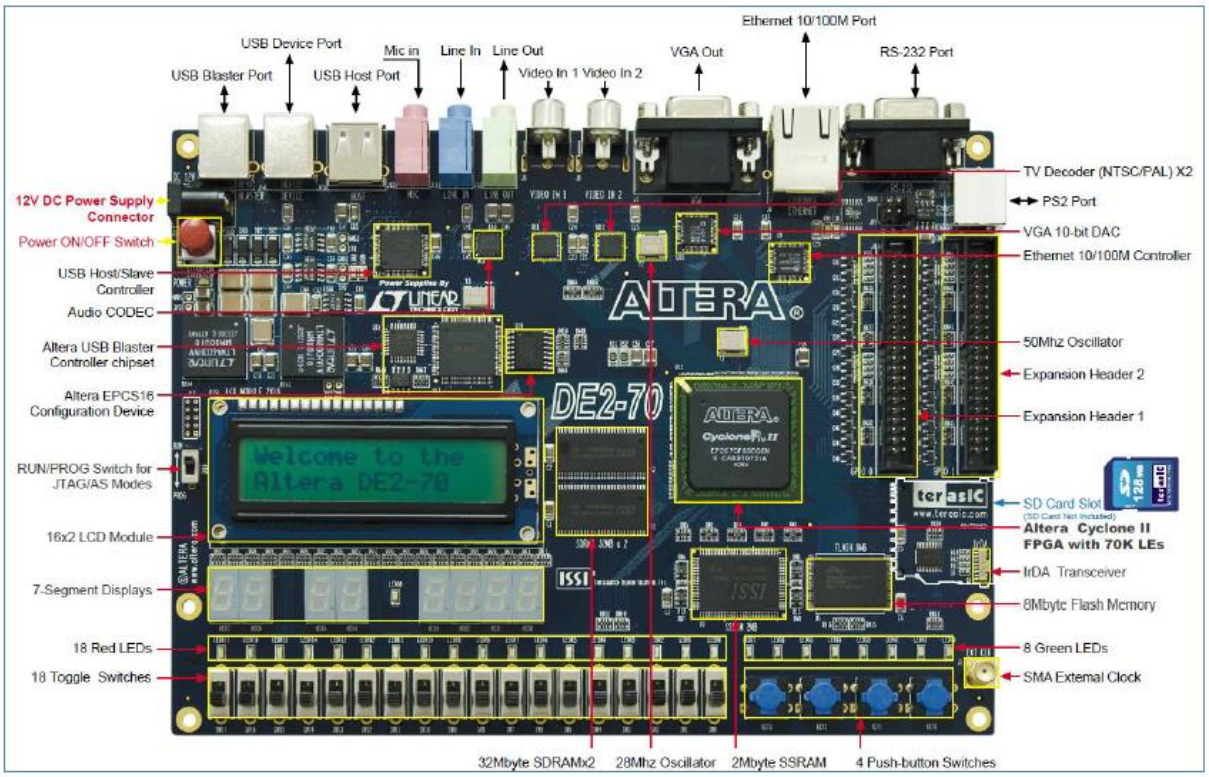


Figure 4 : FPGA Altera DE2-70

Tout au long de ce projet, nous coderons principalement en langage VHDL et en C.
Notons ici que nos codes VHDL-C devraient être utilisables sur d'autres cartes à partir du moment où l'on redéfinit correctement les PINs associés.

ARCHITECTURE GLOBALE DU SYSTEME

Les différents périphériques

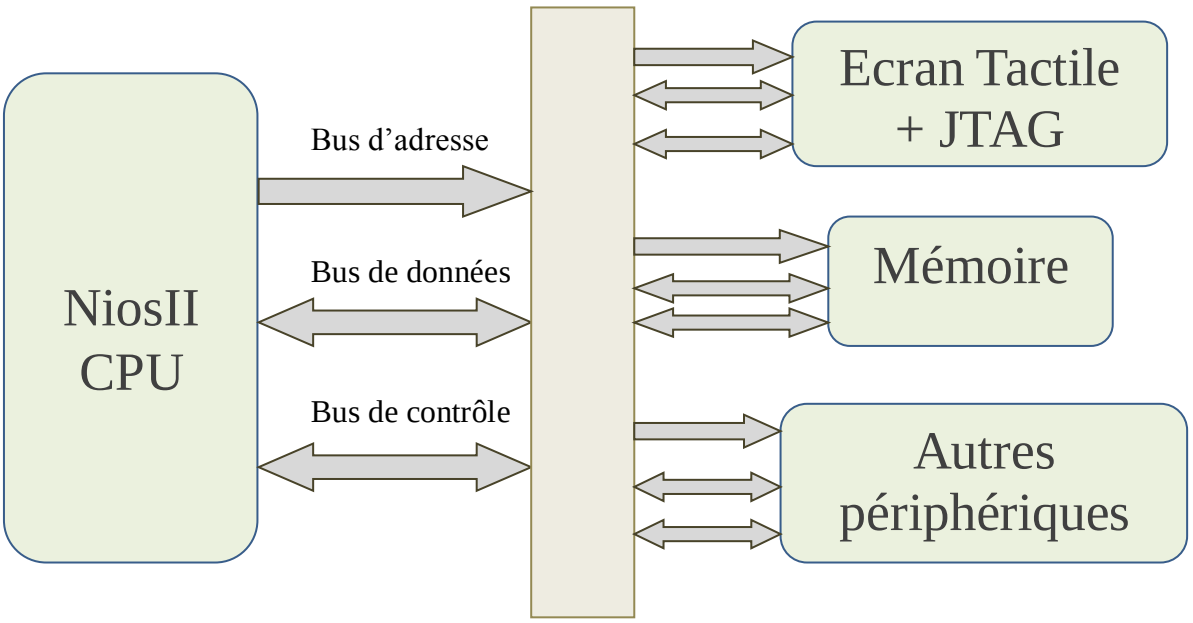


Figure 5 : Schéma-bloc du système

Le processeur NiosII représenté ici est le composant qui va permettre l'exécution des programmes.
On remarque, à travers ce schéma-bloc, que les différents éléments du système sont reliés par trois bus :

- Le bus de données, qui permet comme son nom l'indique, la circulation des données, y compris les instructions de programme, entre le processeur et les autres blocs.
- Le bus d'adresse permet au processeur de désigner à chaque instant la case mémoire ou le périphérique auquel il veut faire appel.
- Le processeur utilise le bus de contrôle pour indiquer la direction de la transaction sur le bus de données. Par exemple, s'il veut faire une écriture ou une lecture dans une case mémoire, ou une entrée/sortie de ou vers un périphérique. On trouve également dans le bus de contrôle une ou plusieurs lignes qui permettent aux circuits périphériques d'effectuer des demandes au processeur. Ces lignes sont appelées lignes d'interruptions matérielles (IRQ).

L'interface présente ici permet des échanges entre le processeur et les périphériques. Elle donnera accès à des registres par le biais de l'écran tactile.

L'ETUDE ET L'IMPLEMENTATION DU SYSTEME

1) La prise en main de l'écran tactile

Cette partie aura pour principal objectif de décrire les différentes étapes préalables nécessaires à la bonne réception des coordonnées X et Y lors d'un appui sur l'écran.

Figure 6 : Ecran tactile

Tout d'abord rappelons que l'écran tactile qui nous est fourni ici est composé de trois composants majeurs : l'écran LCD, le convertisseur analogique numérique (l'ADC), ainsi que les 40 broches de connexions.

L'ADC a pour rôle de convertir les coordonnées du point d'appui (une tension analogique) en ses données numériques. Il l'enverra par la suite sur le FPGA à l'aide des 40 broches.

Afin d'obtenir les coordonnées provenant de l'ADC, une des premières choses à faire est d'étudier le signal ADC_PENIRQ_n (sortie de l'ADC).

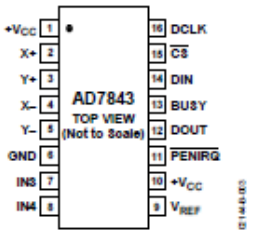


Figure 3. Pin Configuration QSOP/TSSOP

Table 4. Pin Function Descriptions

Pin No.	Mnemonic	Function
1, 10	$+V_{CC}$	Power Supply Input. The $+V_{CC}$ range for the AD7843 is from 2.2 V to 5.25 V. Both $+V_{CC}$ pins should be connected directly together.
2	X+	X+ Position Input. ADC Input Channel 1.
3	Y+	Y+ Position Input. ADC Input Channel 2.
4	X–	X– Position Input.
5	Y–	Y– Position Input.
6	GND	Analog Ground. Ground reference point for all circuitry on the AD7843. All analog input signals and any external reference signal should be referred to this GND voltage.
7	IN3	Auxiliary Input 1. ADC Input Channel 3.
8	IN4	Auxiliary Input 2. ADC Input Channel 4.
9	V_{REF}	Reference Input for the AD7843. An external reference must be applied to this input. The voltage range for the external reference is 1.0 V to $+V_{CC}$. For specified performance, it is 2.5 V.
11	$\overline{PENIRQ}$	Pen Interrupt. CMOS logic open-drain output (requires 10 kΩ to 100 kΩ pull-up register externally).
12	DOUT	Data Out. Logic Output. The conversion result from the AD7843 is provided on this output as a serial data stream. The bits are clocked out on the falling edge of the DCLK input. This output is high impedance when $\overline{CS}$ is high.
13	BUSY	BUSY Output. Logic Output. This output is high impedance when $\overline{CS}$ is high.
14	DIN	Data In. Logic input. Data to be written to the AD7843 control register is provided on this input and is clocked into the register on the rising edge of DCLK (see the Control Register section).
15	$\overline{CS}$	Chip Select Input. Active Low Logic Input. This input provides the dual function of initiating conversions on the AD7843 and also enables the serial input/output register.
16	DCLK	External Clock Input. Logic Input. DCLK provides the serial clock for accessing data from the part. This clock input is also used as the clock source for the AD7843 conversion process.

Figure 7 : Le brochage de l'ADC de l'écran tactile

Lorsqu'on appuie sur l'écran tactile, l'ADC_PENIRQ_n passe à l'état bas. Ce qui provoque l'interruption du FPGA. Par la suite, un mot de contrôle provenant de l'ADC_DIN peut alors être écrit à l'ADC à l'aide de l'interface port série. Ce qui provoque le début de la conversion.

Une conversion complète se fait en 24 cycles d'horloge (ADC_DCLK). Ceci sera expliqué en détails un peu loin.

2) La réception des coordonnées X et Y en continu

Etudions maintenant plus en détails l'ADC, plus particulièrement son chronogramme de conversion de coordonnée (Figure 8).

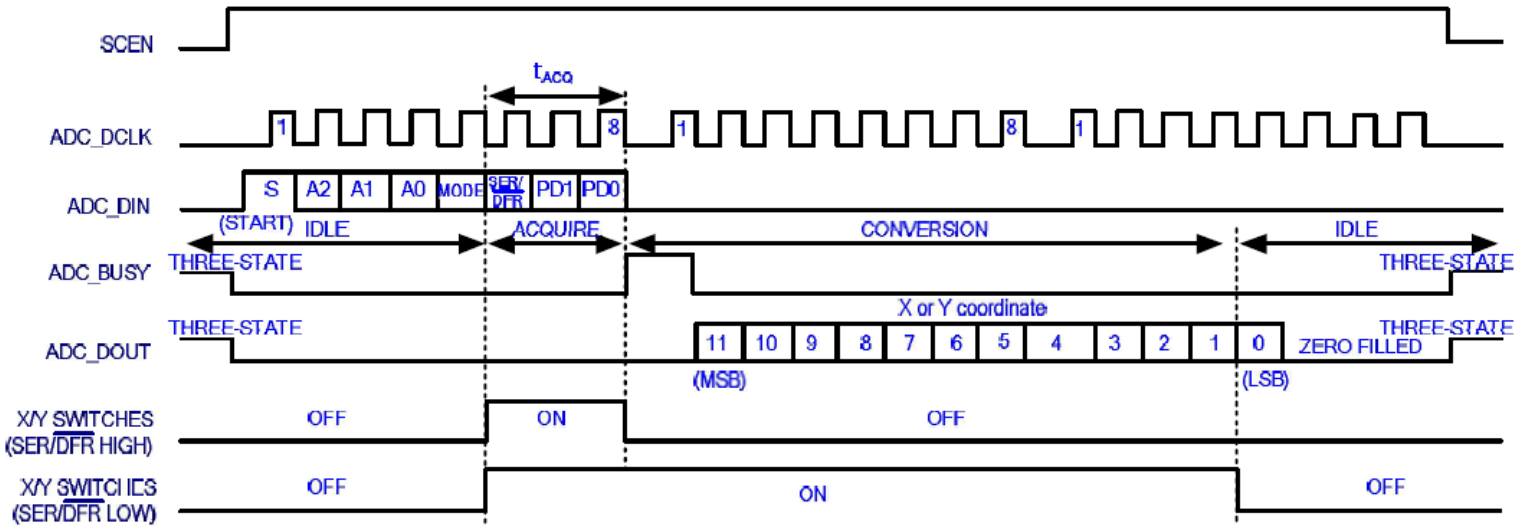


Figure 8 : Chronogramme de conversion de coordonnée

On peut remarquer ici que le mot de contrôle de l'ADC_DIN se fait sur 1 octet pendant les 8 premiers fronts descendants de l'ADC_DCLK.

Les 8 bits de ce mot de contrôle se présentent comme suit :

MSB							LSB
S	A2	A1	A0	MODE	SER / $\overline{\text{DEF}}$	PD1	PD0

Figure 9 : Les 8 bits du mot de contrôle

- **S :** C'est le bit de Start. Un nouveau mot de contrôle peut débuter tous les 15 cycles d'horloge de DCLK si le mode de conversion est de 12 bits, tous les 11 cycles d'horloge de DCLK si le mode de conversion est de 8 bits.
- **A2-A0 et SER/DEF :** Adresse sur 3 bits permettant d'activer la conversion sur X ou Y.
- **Mode :** Ce bit contrôle la résolution de la conversion. S'il est à '0', la conversion se fait en 12 bits. S'il est à '1', elle se fait en 8 bits. Nous avons choisis une résolution sur 12 bits.
- **PD1, PD0 :** Ces deux bits permettent ici de laisser activer le convertisseur en cas d'interruption.

A2[1]	A1[1]	A0[1]	SER/DFR	Analog Input	X Switches	Y Switches	+REF[2]	–REF[2]
0	0	1	1	X+	OFF	ON	V_{REF}	GND
0	1	0	1	IN3	OFF	OFF	V_{REF}	GND
1	0	1	1	Y+	ON	OFF	V_{REF}	GND
1	1	0	1	IN4	OFF	OFF	V_{REF}	GND
0	0	1	0	X+	OFF	ON	Y+	Y–
1	0	1	0	Y+	ON	OFF	X+	X–
1	1	0	0	Outputs Identity Code, 1000 0000 0000				

Ce mot de contrôle transmis provoque l'activation (mise à 1) du signal ADC_BUSY qui à son tour provoque le début de conversion de la coordonnée en question (X ou Y).

Cette conversion durera 12 cycles d'horloge. Une fois terminé, on entre dans un état d'attente (attente d'un appui sur l'écran) pendant 4 cycles d'horloge.

Une conversion complète se fait donc en 24 cycles d'horloge.

Le signal SCEN désigne le chip enable de l'interface port série. Le fait qu'il soit à 1 exprime le fait que le système réponde aux changements sur ses pins. S'il était à 0, il ignorerait tout. On remarque par ailleurs que lorsque ce signal est inactif ('0'), les sorties ADC_BUSY et ADC_DOUT sont dans l'état three-state (l'état haute impédance) qui indiquent la non validité dans la sortie.

Une fois avoir compris ce chronogramme, on souhaite maintenant pouvoir afficher sur les afficheurs sept segments de la carte FPGA les coordonnées X et Y. Pour cela, on implémente un code VHDL dont voici l'architecture :

Figure 10 :

La sortie data_valid permet d'informer sur la validité des coordonnées.

La sortie ADC_CS (actif à l'état bas) correspond au Chip Select. Quand elle est actif, le composant peut être adressé, quand au contraire elle ne l'est pas, le composant est dans un mode dit standby (repos).

Quand l'utilisateur appuie sur l'écran, les coordonnées X et Y sont affichées en hexadécimal respectivement sur les afficheurs 7 segments HEX0-HEX3 et HEX4-HEX6 comme on peut le voir sur la figure 11.

Figure 11 : Affichage des coordonnées X et Y sur les afficheurs 7 segments

Le reset est obtenu en appuyant sur un bouton poussoir de la carte Altera.
Ci-dessous la machine à états du code que l'on a implémenté pour pouvoir obtenir ce résultat :

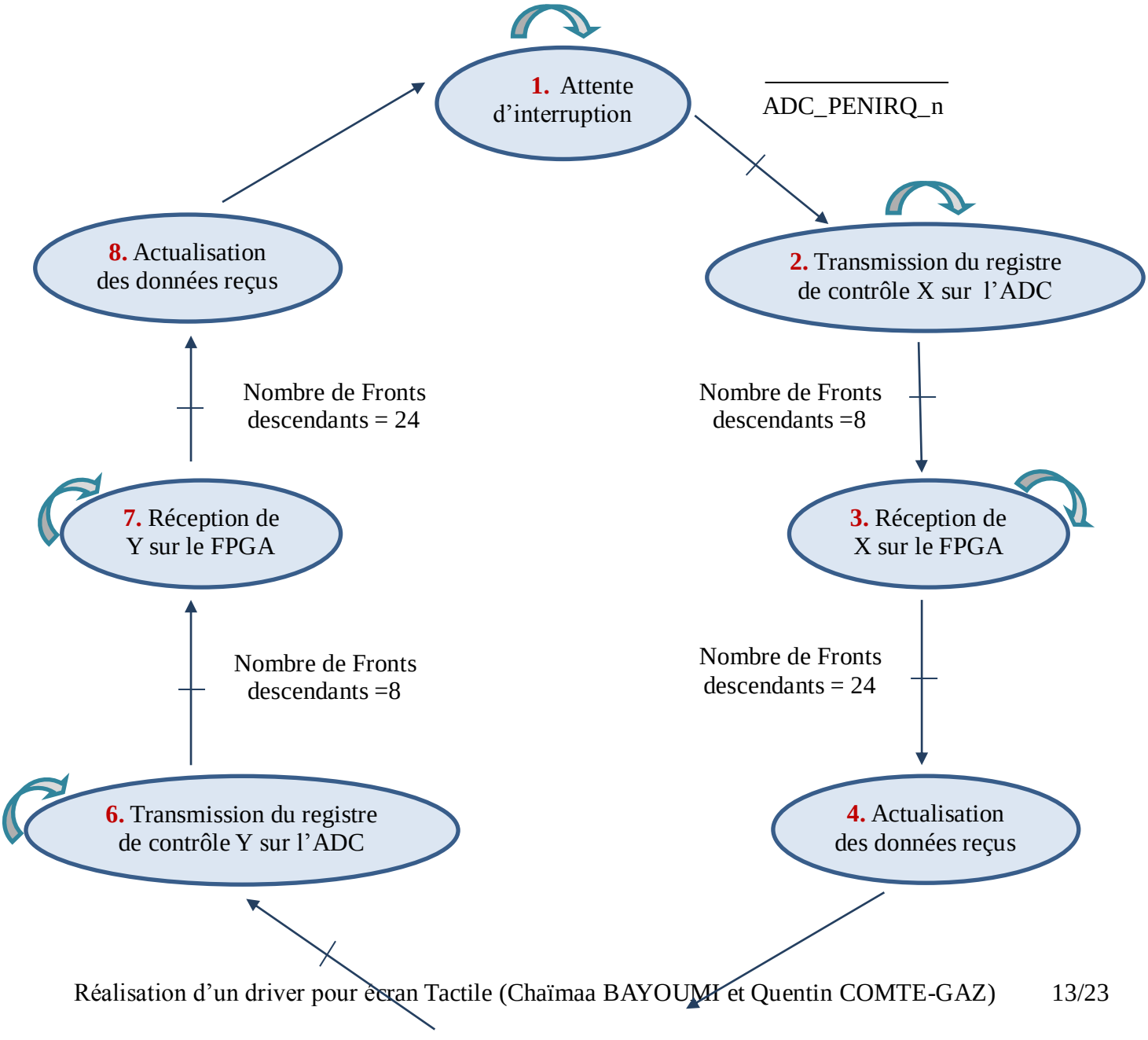


Figure 12 : Machine à états des étapes permettant l'affichage des coordonnées

Explication sur les différents états :

Etat 1 : On attend que l'utilisateur appuie sur l'écran tactile. Ce qui, on le rappelle, provoquera la mise à l'état bas du signal ADC_PENIRQ_n puis l'interruption du FPGA. On se trouve donc en attente d'interruption.
Dans cet état, la sortie data_valid, qui informe sur la validité des coordonnées est bien évidemment à '0' et l'ADC_CS à '1'.

Etat 2 : Transmission du registre de contrôle X sur l'ADC. (Mot de contrôle sur l'ADC_DIN).

Etat 3 : Suite aux 8 fronts descendants et la transmission du mot de contrôle, la conversion de X est effectuée et le FPGA le reçoit.

Etat 4 : Les nouvelles données de X reçus sont actualisées. Les anciennes valeurs sont écrasées et les nouvelles sauvegardées.
La conversion complète de 24 cycles étant faite, l'ADC_CS repasse à '0' et entre dans un mode de repos.

Etat 5 : Attente d'interruption. Dans notre code, l'ADC_PENIRQ_n n'a pas changé d'état depuis l'état 2 (il est toujours à l'état bas). Cet état 5 est donc peut être inutile mais ce n'est qu'une question de sûreté.
L'ADC_CS est mis à '1' pour que le système puisse répondre aux changements.

Etat 6 : Transmission du registre de contrôle Y sur l'ADC (Mot de contrôle sur l'ADC_DIN).

Etat 7 : Suite aux 8 fronts descendants et la transmission du mot de contrôle, la conversion de Y est effectuée et le FPGA le reçoit.

Etat 8 : Les nouvelles données de Y reçus sont actualisées. Les anciennes valeurs sont écrasées et les nouvelles sauvegardées.
Et c'est bien évidemment dans ce dernier état que la sortie data_valid passe à '1' (Lorsque X et Y ont été actualisés).

La conversion complète de 24 cycles étant faite, l'ADC_CS repasse à '0' et entre dans un mode de repos.

Pour tester le bon fonctionnement de ce module. Il faut utiliser les ressources du kit de développement. Pour cela, il faut commencer par définir les broches des entrées/sorties : c'est le rôle du fichier des contraintes, "*assignments.csv*", comme défini sur la figure suivante :

```
ADC_BUSY,Input,PIN_T24,6,B6_N0,,
ADC_CS,Output,PIN_N21,5,B5_N2,,
ADC_DCLK,Output,PIN_E28,5,B5_N0,,
ADC_DIN,Output,PIN_C29,5,B5_N0,,
ADC_DOUT,Input,PIN_C30,5,B5_N0,,
ADC_PENIRQ_n,Output,PIN_H23,6,B6_N0,,
CLK,Input,PIN_AD15,7,B7_N3,,
RST_n,Input,PIN_T29,6,B6_N0,,
DATA_VALID, Output,PIN_H24,B6_N0,,
```

Figure 12 : Fichier de contraintes

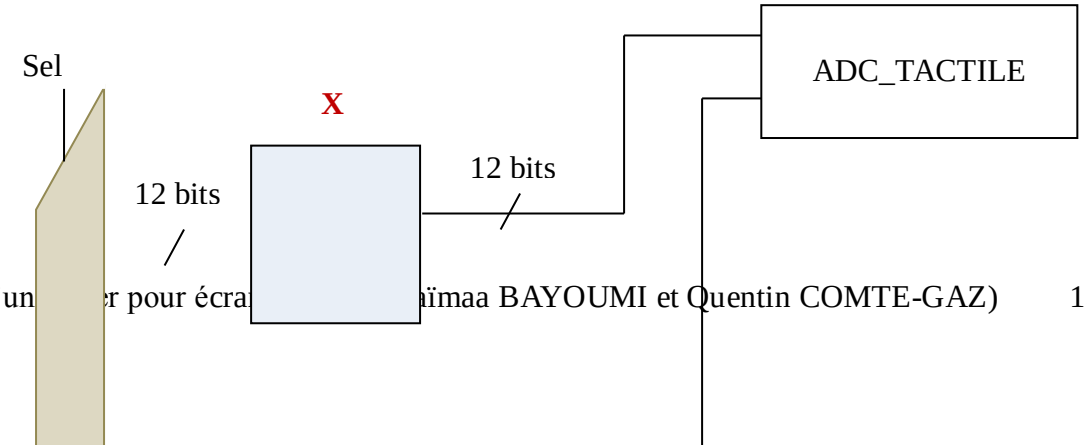
Ceci est une méthode donnant la possibilité d'importer les contraintes à partir d'un fichier de contraintes. Mais on peut également affecter toutes les broches manuellement une à une avec la méthode du Pin Planner.

Notre code permet donc de récupérer les coordonnées de l'appui en continu. Nous avons fait cela afin que le processeur puisse recevoir très rapidement les coordonnées.

En effet, il nous paraissait plus intéressant de concevoir une partie VHDL efficace afin que par la suite, le code applicatif soit très simple à concevoir.

3) Le choix d'affichage de X ou de Y sur les afficheurs 7 segments

L'objectif de cette partie est de pouvoir afficher selon une demande soit X, soit Y sur les afficheurs 7 segments. Pour ce faire, on utilise alors un multiplexeur comme dessiné ci-dessous :



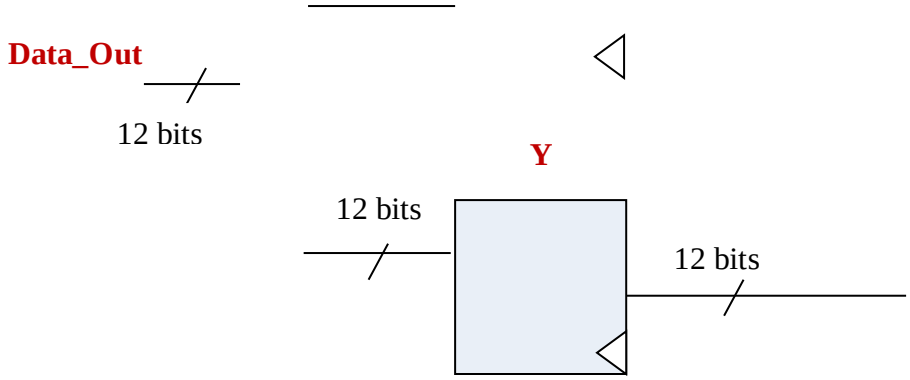


Figure 13 : Choix d'affichage de X ou de Y

La sortie Data_out correspond aux coordonnées X ou Y suivant la valeur de Sel. La sélection étant pour commencer un simple switch de la carte DE2-70.

L'architecture du code VHDL devient alors :

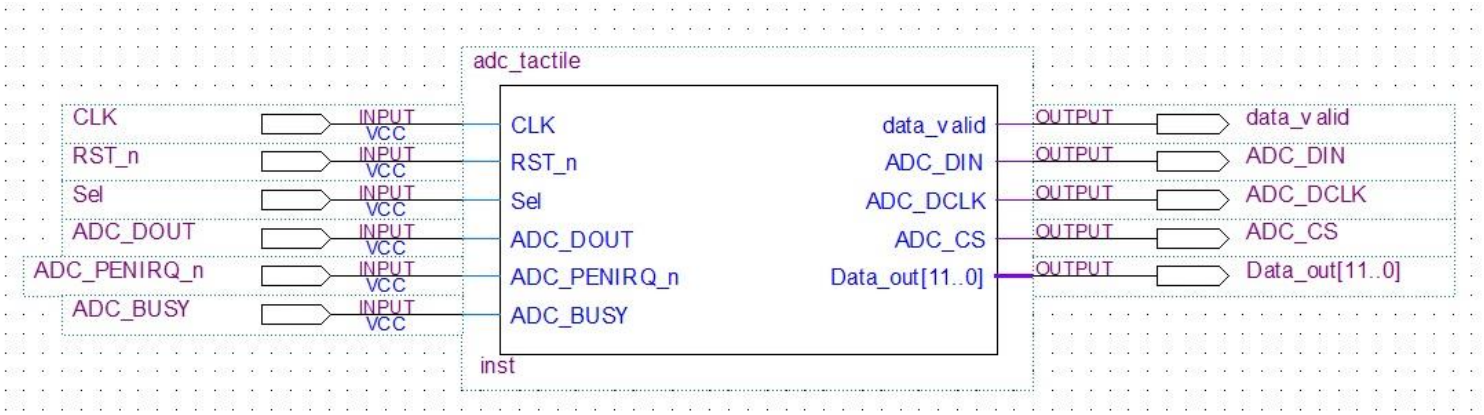


Figure 14 : Architecture du code VHDL modifié

4) La mise en œuvre de l'interface et du driver

Notre code ne doit pas seulement fonctionner, il doit être facilement intégrable par un utilisateur voulant l'intégrer sans se plonger dans le code associé.

Pour cela, nous avons, en partie, éditer notre code VHDL afin qu'il soit compatible avec l'architecture que nous vous avons présenté précédemment (bus de données / adresse / contrôle) et nous avons fait en sorte que les PINs de connexion avec l'écran tactile puissent être définit par l'utilisateur à l'aide du fichier de contraintes.

Ainsi, nous avons instancié les entrées/sorties connectées à l'écran tactile en export.

Pour ce qui est des bus, nous avons rajouté un bus d'adresse (adress), les signaux "read" et "write" faisant office de bus de contrôle et bien entendu d'un bus de données ("writedata" et "readdata").

Notons que le bus d'adresse n'est que sur "1 bit" du fait que l'on n'a besoin de communiquer uniquement les coordonnées suivant X ou Y.

Le bus de données est quant à lui sur 16 bits du fait que les coordonnées sont sur 12 bits et qu'il faut que le bus soit de taille 8/16/24/32 bits.

Cependant, on pourrait augmenter la taille de ce bus afin de communiquer d'autres informations utiles pour le processeur tels que la validité des coordonnées (Y a t-il eu un appui?) et le temps depuis le dernier appui.

Il est ensuite possible de passer aux étapes suivantes:

En générant le processeur grâce à l'onglet generate dans qsys, le système my_cpu devient disponible dans la bibliothèque de travail et peut alors être instancié dans la cellule schématique.

Il ne faut pas oublier d'ajouter au projet le fichier "mySOPC.qip" qui vient d'être généré.

La phase suivante consiste à écrire un programme (logiciel) pour le processeur Nios2.

Le langage que nous utilisons ici est le langage C. Pour ce faire, nous utilisons l'environnement Eclipse.

Notons que notre code C est très simple (lecture dans un registre) car nous avons conçu une partie VHDL conséquente pour cette raison.

LES DIFFERENTS TESTS POUR LE DEBUG

Nous présentons ici les différents composants qui nous ont permis de tester notre système.

1) Avec l'écran LCD de la carte FPGA

Figure 14 : Ecran LCD du FPGA

```
char Text1[16] = "Bonjour ";
char Text2[16] = "NIOS II!";
//  Initial LCD
LCD_Init(LCD_CARTE_BASE);
//  Show Text to LCD
LCD_Show_Text(LCD_CARTE_BASE, Text1);
//  Change Line2
LCD_Line2(LCD_CARTE_BASE);
//  Show Text to LCD
LCD_Show_Text(LCD_CARTE_BASE, Text2);

/*SEG7_Clear();*/
```

Figure 15 :

2) Avec les switchs et les LEDs de la carte FPGA

Dans un premier temps, cette étape nous a seulement permis de prendre en main le logiciel Quartus et Qsys.

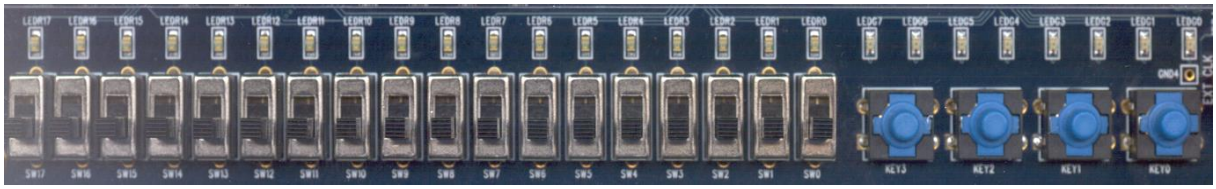


Figure 16 : Les switchs, les boutons poussoirs et les LEDs de la carte FPGA

```
//Test qui associe les leds vertes avec les 8 premiers switchs
int i;
for(i=0;i<8;i++)
    IOWR(LEDG_BASE, i, IORD(SW_BASE,i));
```

Figure 17 :

Ce code permet d'afficher sur les LEDs l'état des switchs.

3) Avec les afficheurs 7 segments de la carte FPGA.

L'intégration des 7 segments a été faite directement dans Qsys sans passer par le code C d'Eclipse.

Figure 18 : Les afficheurs 7 segments de la carte FPGA

LES DIFFICULTEES RENCONTREES

Durant ce projet, les difficultés rencontrées ont principalement été d'ordre technique. En effet, ayant l'habitude de travailler avec le logiciel Xilinx, nous avons eu du mal au départ à prendre en main le logiciel Quartus.

Nous avons passé énormément de temps à comprendre comment fonctionnait ce logiciel, comment ses fonctions devaient être utilisés (comme la création d'un nouveau composant dans qsys par exemple ou encore…).

Cependant, avec le temps et la persévérance ces difficultés ont été surmontées. Nous sommes devenus de plus en plus efficaces au cours des séances et le travail demandé a été finalement mené à bien.

CONCLUSION

Comme nous l'avions dit un peu plus haut, les objectifs de ce projet ont en partie été atteints et sa réalisation nous a été particulièrement bénéfique.

En effet, d'une part ce projet nous a permis d'acquérir des connaissances sur un nouvel environnement de développement, celui de Quartus/QSys. D'autre part, cela nous a permis d'améliorer nos connaissances en VHDL-C et de réaliser un projet concret avec ces langages.

Cependant nous pouvons aller encore plus loin dans ce projet en ajoutant dans le driver une partie affichage, sur l'écran, d'un pixel puis d'une ligne/colonne (suite de pixels) et enfin des caractères/images (gérer correctement les lignes/colonnes).

De plus, nous pourrions rajouter une interruption sur le processeur lors d'un appui sur l'écran. Cependant, par manque de temps et de connaissance du logiciel, nous n'avons pas réussi à ajouter la fonctionnalité (notons tout de même que nous avons ajouté un signal de sortie dans le code VHDL correspondant à l'interruption. Cependant, nous ne l'avons pas intégrer sur QSYS).

Nous pourrions également concevoir un guide utilisateur afin de simplifier l'utilisation de notre driver (exemple de connexions des PINs pour différentes cartes, exemple type d'utilisation) .

TABLE DES ILLUSTRATIONS