UNIVERSITE PAUL SABATIER

TOULOUSE III

M2 EEA

SME

Synthese et mise en oeuvre des systèmes

Cahier de TP de VHDL

TP 1

Initiation au VHDL

1 Introduction

Le but de ce TP est de prendre en main le langage VHDL afin de maîtriser ses fondamentaux. Ce TP a pour but aussi de prendre en main le logiciel Quartus qui permet de mettre en œuvre des solutions sur les FPGA Altera.

2 Logique combinatoire

2.1 Porte ET

Voici la table de vérité d'une porte ET à deux entrées.

A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

TABLE 1.1 – Table de vérité d'une porte ET

Créer d'abord un projet Quartus.

Décrire l'entité de la porte ET à l'aide de la table de vérité.

Implémenter l'architecture de la porte ET à l'aide de la table de vérité.

Simuler et vérifier le bon fonctionnement de la porte ET à l'aide de la table de vérité.

Connecter les entrées A et B à des boutons et la sa sortie S à une LED. Vérifier son fonctionnement.

2.2 Décodeur 7seg

Un afficheur sept segments est utilisé pour afficher des informations vers un utilisateur en allumant une partie des segments. Par exemple si on veut afficher le chiffre *2*, on allume les segments *A, B, D, E, G*.

Faire la table de vérité de l'afficheur sept segments pour les chiffres allant de **0 à 15 en hexadécimal**

Décrire l'entité et l'architecture votre table de vérité en VHDL.

Simuler et vérifier le bon comportement de votre table de vérité

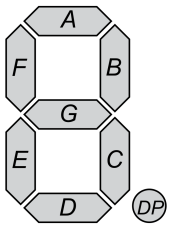

FIGURE 1.1 – Segments d'un afficheur

Connecter les entrées de votre décodeur sept segments aux switchs (SWx) et les sorties aux afficheurs sept segments (HEXx) de la carte DE2. Tester et vérifier le bon fonctionnement.

3 Logique séquentielle

3.1 Compteur

Voici la table de vérité d'un simple compteur :

$ARst\_N$	Clk	$SRst$	En	Q_t
0	*	*	*	0
1	⎍	1	*	0
1	⎍	0	0	Q_{t-1}
1	⎍	0	1	$Q_{t-1}+1$

TABLE 1.2 – Table de vérité d'un compteur

Implémenter cette table de vérité en VHDL en utilisant un *process* et **respectant bien la priorité**. Contraigner le vecteur Q_n sur **8 bits**.

Simuler votre compteur et vérifier que votre implémentation respecte bien la table de vérité.

3.2 Cascade de compteurs

Faire un compteur qui comptera toutes les secondes. Afficher la valeur de ce compteur sur un afficheur sept segments. Ce compteur devra compter 0 à 15.

Faire un schéma fonctionnel en faisant apparaître deux compteurs en cascades.
NB : Il est interdit de connecter aux entrées horloges autres que des signaux horloges.

Implémenter votre schéma fonctionnel en VHDL.

A l'aide de l'analyseur logique (Signal Tap Logic Analyzer) visualiser le signal Q des compteurs ainsi que le signal *En* du second compteur.

Compiler, programmer et en utilisant l'analyseur logique, déclencher l'acquisition lorsque *En* vaut 1. Vérifier que *En* est à 1 que pendant un seul coup d'horloge.

3.3 Machine à états

Modéliser une machine a états qui permet d'incrémenter un compteur **d'une unité** lorsque l'on appuie sur un bouton.

Créer le composant *FsmBtn* et implémenter **uniquement** la machine à états.

Valider le bon fonctionnement de votre machine à états en utilisant un compteur et en visualisant son contenu sur un afficheur sept segments. Utiliser le bouton *KEY0* incrémenter le compteur.

Sur le compteur de 3.1, ajouter le signal *ud* qui permettra de changer la direction de comptage du compteur. Ce signal est le moins prioritaire de tous.

Ajouter le nouveau signal à la table de vérité 1.2.

Valider le bon fonctionnement en simulation.

En ajoutant une nouvelle instance du composant *FsmBtn*, ajouter la fonctionnalité de décrémenter le compteur **d'une unité** lorsque l'on appuie sur un autre bouton. Utiliser le bouton *KEY1* décrémenter le compteur.

4 Mini projet - Horloge

L'horloge n'affichera que les **heures** et les **minutes**. L'affichage des dizaine et unités des heures et des minutes se fera sur afficheur sept segments chacun.

L'horloge aura deux modes de fonctionnement. Un mode normal où l'heure compte normalement et un mode configuration où on pourra venir modifier les heures et les minutes.

Si on appuie sur *KEY1*, on rentre en mode configuration.

Lorsque l'on appuie sur *KEY2* ou *KEY0*, on incrémentera ou décrémentera respectivement soit les heures soit les minutes.

Lorsque l'on rentre en mode configuration, ce sont les minutes qui peuvent être modifies. Un nouvel appuie sur *KEY1*, ce seront les heures qui pourront être modifies. Un nouvel appui sur *KEY1*, on revient en mode normal.

En utilisant les composants et en modélisant des nouveaux, faites un schema fonctionnel et les machines à états nécessaires pour modéliser l'horloge.

Implémenter votre modélisation en VHDL.

Valider votre implémentation en simulation.

Valider votre implémentation sur votre carte.

TP 2

Utilisation du processeur Nios

1 Introduction

Le but de ce TP est de prendre en main l'outil *Platform Designer* afin de pouvoir mettre en œuvre des solutions complexes et ciblés basé sur le processeur *Nios II* et le bus *Avalon*.

2 Construction d'un microcontrôleur personnalisé

A l'aide de l'outil *Platform Designer*, construire un microcontrôleur basé sur le processeur *Nios II* en ajoutant les périphériques suivants :

- Nios II
- On Chip RAM (où le code sera stocké)
- JTAG UART (pour avoir une console série + pouvoir debugger)
- PIO (pour pouvoir contrôler des Leds)

Générer le VHDL.

Connecter la sortie du périphérique *PIO* aux Leds de la carte DE0 Nano.

Compiler et téléverser la configuration du FPGA vers votre carte.

En utilisant le logiciel *Nios II Software Build Tools for Eclipse* créer un projet en prenant comme modèle de projet **Hello World Small**.

Sans toucher au code qui a été généré, compiler et téléverser vers le Nios. Vérifier que *"Hello from Nios II!"* s'affiche bien sur la console *Nios II Console*.

Écrire un code qui permet de faire un chenillard va et vient sur les Leds en utilisant le périphérique PIO.

3 Ajout d'un périphérique personnalisé au microcontrôleur

3.1 PWM

Créer un composant VHDL qui permet de générer un signal PWM sur une broche. Le PWM doit avoir une résolution de **32 bits**.

Faire un diagramme fonctionnel de ce composant.

Décrire l'entité et implémenter l'architecture en utilisant le diagramme fonctionnel.

Simuler et vérifier que votre composant crée bien un signal PWM avec la fréquence et le rapport cyclique variable.

3.2 Interface avec le bus Avalon

Créer un composant VHDL qui permet d'interfacer le composant PWM que vous venez créer, avec le processeur Nios II à l'aide du bus Avalon. La table d'adressage doit être la suivante :

31 … 24	23 … 16	15 … 8	7 … 0	
Duty				00h
Freq				01h

TABLE 2.1 – Adressage du composant PWM sur le bus Avalon

En utilisant l'outil *Platform Designer*, ajouter votre composant d'interface que vous venez de créer pour qu'il soit reconnu comme un périphérique Avalon.

Intégrer ce composant au microcontrôleur que vous avez créer en 2 et exporter la sortie du composant PWM. Générer le VHDL du système.

Affectez la sortie de votre PWM à une broche quelconque.

Compiler et téléverser la nouvelle configuration du FPGA vers votre carte.

Avec *Nios II Software Build Tools for Eclipse*, tester votre périphérique en changeant le rapport cyclique et la fréquence. Valider le fonctionnent à l'aide d'un oscilloscope.