Cahier des charges

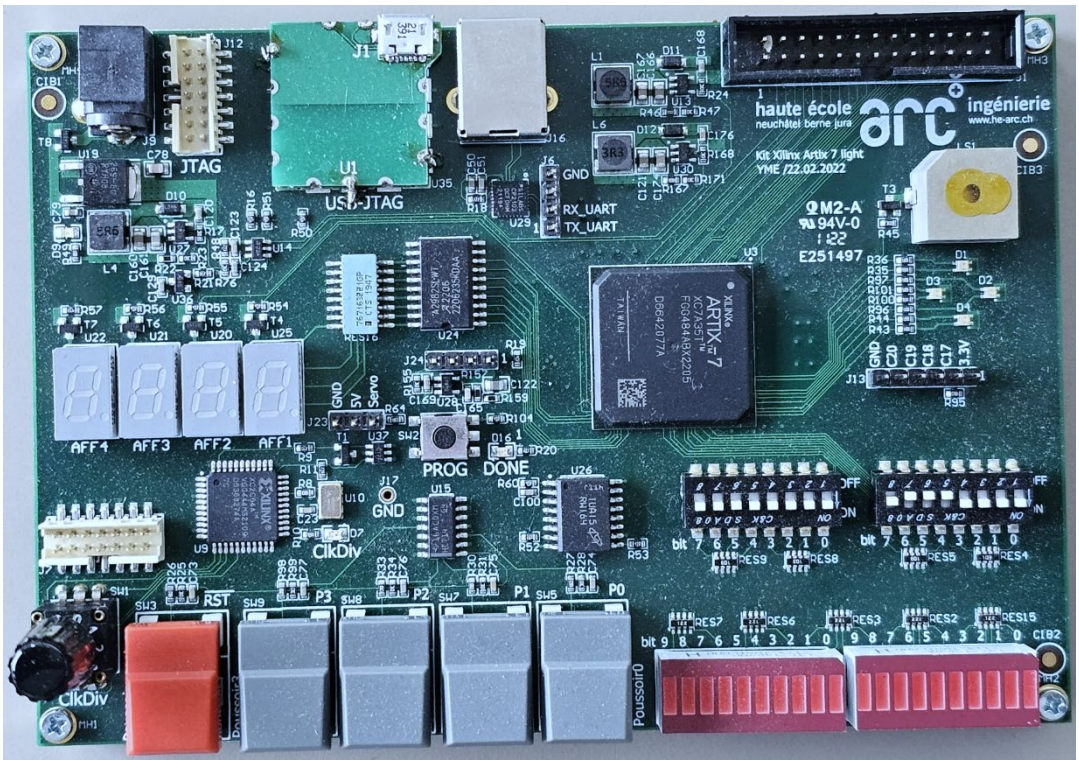

Figure 1:Kit Xilinx 7

1. Déroulement

- Le projet est réalisé par groupe de 2 au maximum.
- Le travail en dehors des heures de cours est **indispensable** à l'aboutissement du projet.
- Chaque personne doit tenir à jour un « cahier de laboratoire » où il note proprement ce qu'il a fait personnellement et quand il l'a fait (date et durée), les problèmes rencontrés, la raison de certains choix, etc. Le but est d'avoir une traçabilité individuelle du déroulement du projet. Ce cahier est à **envoyer par email à la fin de chaque leçon** au professeur.
- Il est autorisé de collaborer entre les groupes, par contre, la copie partielle ou totale et/ou la transmission de documents sont interdits et seront sanctionnés.
- Tous les documents non restitués dans les délais ou copiés (même partiellement) seront taxés de la note 1.

2. Rendus

- Chaque groupe doit rendre **par email** dans un document Nom1_Nom2.zip:
    - Un document complet (PDF) présentant la conception, **à la fin de la semaine 11 (vendredi 16h au plus tard)**
        - Schéma bloc du processeur (nanoProcesseur)
        - Schéma bloc du contrôleur (nanControleur)
        - Graph des états du séquenceur
        - Table de vérité des sorties du séquenceur
        - Plan mémoire
        - Cahier de laboratoire complet

      Remarques : Les documents faits proprement à la main (avec une règle) et scannés suffisent.

    - Un projet vivado avec testbench de votre nanoControleur qui effectue une application en « assembleur » (voir description en fin de document) en utilisant le jeu d'instruction fourni **au début du cours de la semaine 15**. Des questions seront posées afin de vérifier la compréhension du fonctionnement du nanoControleur et de votre application « assembleur » en ROM.
        - Cahier de laboratoire complet est à rendre également

    - Un document complet (PDF) présentant les modifications de la conception pour effectuer des appels de routines ainsi que des interruptions, **à la fin du cours de la semaine 18**
        - Schéma bloc du processeur (nanoProcesseur)
        - Schéma bloc du contrôleur (nanControleur)
        - Graph des états du séquenceur
        - Table de vérité des sorties du séquenceur
        - Plan mémoire
        - Jeu d'instructions
        - Cahier de laboratoire complet

    - Un projet complet de votre microcontrôleur **à la fin du cours de la semaine 21**
        - Projet Vivado avec les nouveaux fichiers sources y compris la ROM mais **sans** le banc de test (testbench)

    - Un projet complet de votre microcontrôleur **durant le cours de la semaine 24**
        - Projet Vivado avec les fichiers sources finaux et le banc de test (testbench)
        - Cahier de laboratoire complet
        - Faire une présentation (oral sans support Powerpoint) ainsi qu'une démonstration sur le kit Xilinx.
        - Des questions seront posées par les professeurs afin d'évaluer votre degré de compréhension du projet.

3. But

Intégrer dans le circuit logique programmable du Kit Xilinx 7 le nanoControleur permettant d'exécuter une petite application logicielle écrite en assembleur propre à votre nanoProcesseur. Ensuite, il faudra modifier l'architecture du nanoControleur pour lui permettre d'effectuer des appels de routine ainsi que de traiter une interruption.

Toutes les bascules doivent être synchronisées (clockées) par le signal d'horloge de 100MHz et toutes doivent avoir un **reset asynchrone actif bas**.

4. Description du nanoControleur

Le nanoControleur est composé de :

- Un nanoProcesseur
- Une RAM de 32 bytes
- Une ROM contenant le programme exécuté par le nanoProcesseur
- Un décodeur d'adresses
- Un multiplexeur permettant de choisir les données lues par le nanoProcesseur
- Deux registres de sortie de 8 bits, deux ports d'entrée de 8 bits

Le nanoProcesseur est composé de :

- Un Program Counter de 8 bits
- Un registre IR (Instruction Register) qui mémorise l'instruction à exécuter
- Une ALU avec calcul des indicateurs (flag : Z, C, N et V)
- Un accumulateur 8 bits pour recevoir le résultat de l'ALU
- Un multiplexeur permettant de choisir les opérandes de l'ALU
- Un registre permettant de mémoriser les 2 opérandes de l'ALU
- Un registre de contrôle de 4 bits contenant Z C N V
- Un séquenceur sous la forme d'une machine d'états synchrones qui « cadence » le nanoProcesseur

Le nanoProcesseur est réalisé selon l'architecture de type Harvard qui sépare physiquement la mémoire de données et la mémoire programme. L'accès à chacune des deux mémoires s'effectue via deux bus distincts.

Rappel des indicateurs :

Flag	Nom	Description
Z	Zero flag	Indique que le résultat d'une opération arithmétique ou logique ou d'un chargement était zéro.
C	Carry flag	Retenue qui permet d'ajouter/soustraire des nombres supérieurs à un octet. Il est également utilisé pour étendre les décalages binaires.
N	Negative flag	Indique que le résultat d'une opération mathématique est négatif.
V	Overflow flag	Indique que le résultat signé d'une opération est trop grand (dépassement) pour tenir dans la largeur du registre en utilisant la représentation en complément à deux.

5. Blocs du nanoControleur

Le nanoControleur est décomposé hiérarchiquement afin d'obtenir des blocs simples, étudiés durant le cours de systèmes numériques. Vous trouverez ci-dessous une description des blocs qui figurent dans le nanoControleur :

ROM

Le bloc ROM contient le programme qui sera exécuté par le nanoControleur. Il est réalisé sous forme d'un multiplexeur (combinatoire avec assignation sélective concurente) qui fournit l'instruction (14 bits) à exécuter en fonction de la valeur du registre Program Counter PC. Pour les instructions sans opérande il faut ajouter 8 bits à '0', '1' ou '-' pour compléter le bus

RAM

Le bloc RAM permet de mémoriser les 32 bytes de données. Il est réalisé avec un bloc RAM (voir doc Xilinx). Un bus d'adresses de 5 bits permet d'accéder aux données 8 bits.

Le décodeur d'adresse

Ce bloc combinatoire génère les 3 signaux de sélection (chip select) pour les accès dans la RAM ou dans les 2 ports d'entrées ou sorties. Les signaux chip select sont générés en fonction du bus d'adresse. Le plan mémoire (memory map) définit les zones actives des chip select.

Le multiplexeur

Ce bloc combinatoire permet de sélectionner grâce aux chips select du décodeur d'adresse quelles données sont lues par le nanoProcesseur, à savoir soit la RAM, ou l'un des 2 ports d'entrée de 8 bits synchronisé.

Registres de sorties

Les 2 ports de sorties sont deux registres tampon (parallèles) de 8 bits qui mémorisent la valeur envoyée par le nanoProcesseur lors d'une instruction STORE.

Ports d'entrées

Il y a deux ports d'entrées de 8 bits.

nanoProcesseur

Le bloc nanoProcesseur est la CPU du nanoControleur. Il est également décomposé hiérarchiquement et il est décrit ci-dessous.

6. Blocs du nanoProcesseur

Le nanoProcesseur est décomposé hiérarchiquement afin d'obtenir des blocs simples, étudiés durant le cours de systèmes numériques. Vous trouverez ci-dessous une description des blocs qui figurent dans le nanoProcesseur:

Le séquenceur

Le séquenceur est le chef d'orchestre du nanoProcesseur. Il s'agit d'une machine à états synchrone de Mealy permettant de générer les différents signaux de contrôles des autres blocs en fonction de l'état actuel et des entrées du séquenceur.

Les entrées sont :
- L'opcode provenant du registre IR
- Les indicateurs (flags : Z, C, N et V) provenant du registre CCR

Les sorties sont (par ordre alphabétique) :
- Accu_load
  permet le chargement de la sortie de l'ALU dans le registre Accu

- CCR_load
  permet le chargement des indicateurs Z C N V dans le registre CCR

- data_wr
  permet l'écriture d'une donnée dans la RAM ou sur un port de sortie

- IR_load
  permet le chargement de l'instruction provenant de la ROM dans le registre d'instruction

- oper_load
  permet le chargement des opérateurs dans le registre du bloc operande_select

- oper_sel
  code de 3 bits permettant la sélection des opérandes utilisé par l'ALU en fonction de l'opcode

- PC_inc
  PC_inc vaut '1' quand le ProgramCounter (PC) doit être augmenté d'une unité.
  PC_inc vaut '0' quand le PC doit prendre une nouvelle valeur à cause d'un « branchement »

- PC_load
  permet le chargement du nouveau PC dans le registre PC

Le ProgramCounter

Le bloc ProgramCounter est un compteur qui contient l'adresse de la prochaine instruction à exécuter. La nouvelle adresse est soit l'adresse actuelle incrémentée de 1 ou soit une adresse provenant d'une instruction de branchement.

Le registre IR

Le registre d'instructions (IR) mémorise l'instruction à exécuter fourni par la ROM. Les instructions sont composées d'un opcode de 6 bits et d'un opérande de 8 bits.

Le registre Accu

Le registre Accu contient la valeur de l'accumulateur. Cette valeur provient de l'ALU. Elle est chargée à la fin du traitement l'ALU.

Le registre CCR

Le registre de contrôle CCR mémorise les indicateurs (flags : Z, C, N et V) qui sont calculés par l'ALU.

Le sélecteur d'opérandes « operande_select »

Il s'agit d'un multiplexeur qui permet de choisir les opérandes utilisés par l'ALU. La sélection se fait à l'aide d'un code de 3 bits fourni par le séquenceur.

Le registre Opérandes

Registre permettant de mémoriser les 2 opérandes de l'ALU

L'ALU

L'ALU effectue en parallèle (combinatoire) toutes les opérations arithmétiques et logiques sur les opérandes ainsi que les opérations de chargement. Un multiplexeur commandé par l'opcode permet de choisir le résultat de l'opération demandée.
L'ALU calcule également les indicateurs (flags : Z, C, N et V) (combinatoire). Un système de masque dépendant de l'opcode permet de ne pas modifier certains indicateurs selon l'opération effectuée.

L'Alu est donc séparé en deux parties distinctes :

1. Le bloc alu combinatoire qui calcule le résultat en fonction de l'opcode des opérandes et des flags, ce bloc gère également un signal local CCR_mask de 4 bit qui permet de savoir quel seront les flags mis à jours lors de l'instruction en cours. Créer un signal ALU_result sur 9 bits pour les opérations nécessitant la gestion du carry, ceci simplifiera grandement la gestion du flag C.

2. Un bloc combinatoire qui va calculer individuellement les flags Z, C, N et V en fonction de ALU_result, de CCR_mask, du registre CCR et selon le type de flags des opérandes et/ou de l'opcode.

7. Travail à réaliser

Rendu 1 :

Analyser le projet Vivado fourni et créer les documents suivants en fonction des fichiers *.vhdl :

- Schéma bloc du processeur (nanoProcesseur)
- Schéma bloc du contrôleur (nanoControleur)
- Graph des états du séquenceur
- Table de vérité des sorties du séquenceur
- Plan mémoire

Rendu 2 :

Modifier le programme qui se trouve dans la ROM en utilisant uniquement le jeu d'instructions fourni. Un banc de test (testbench) doit également être créé afin de valider et vérifier le bon fonctionnement de votre programme.

L'application en assembleur est la suivante et doit tourner en boucle (vérifier le fichier de contrainte afin de savoir où sont connectés les différents périphériques):

- Lire l'état des 8 dilswitch 1 et les sauvegarder en RAM
- Lire l'état des 8 dilswitch 2
- Si l'état des 8 dilswitchs 2 sont 0 alors :
  - Afficher la valeur binaire lue sur les dilswitch 1 sur les 8 leds du bargraphe 1
  - Afficher sur l'affichage 7 segment la valeur hexadécimale correspondant à l'état des 4 bits de poids forts des dilswitch 1
- Sinon :
  - Faire un ou exclusif entre les 8 dilswitch 1 et les 8 dilswitch 2, afficher le résultat sur le bargraphe 1 et afficher un C (pour calcul) sur l'affichage 7 segments

Rendu 3 :

Modifier l'architecture du nanoControleur pour pouvoir faire des appels de **sous-routines** dans l'application assembleur, il faudra y apporter les modifications matérielles suivantes (liste non–exhaustive) :

- Ajouter un registre (Pile ou Stack) pour sauvegarder l'adresse de retour (Program_Counter) de la sous-routine. Ce registre LIFO (Last IN First OUT) se présente sous la forme d'un registre à décalage de 8 x 8 bits, ce qui permet de sauvegarder jusqu'à maximum 8 adresses de retour.

- Modifier le Program_Counter pour permettre de restaurer/charger l'adresse de retour en fin de sous-routine.

- Modifier la table de vérité des sorties du séquenceur et ensuite le séquenceur pour gérer les signaux de sauvegarde et de restauration.

- Ajouter les instructions appel et retour de sous-routine dans la liste d'instructions et modifier le paquetage nanoProcesseur_package en conséquence.

Modifier l'architecture du nanoControleur pour pouvoir effectuer une **interruption**, il faudra y apporter les modifications matérielles suivantes (liste non–exhaustive) :

- Ajouter une entrée Interrupt au top de la hiérarchie. Cette entrée doit arriver dans un bloc de synchronisation et détections de flanc montant (dans le nanoProcesseur). La sortie de ce bloc doit aller dans un registre qui permet de mémoriser/effacer l'interruption.

- Modifier la (Pile ou Stack) pour sauvegarder l'adresse de retour (Program_Counter) de la routine d'interruption, même pile qu'au point précédent.

- Modifier le Program_Counter pour permettre de restaurer/charger l'adresse de retour en fin de routine d'interruption. Le Program_Counter doit également être modifié pour sauter à l'adresse 0xFF en cas d'interruption.

- Ajouter deux registres qui permettent de sauvegarder l'accu (W_register) et les flags (Status_Register) avant d'effectuer la routine d'interruption. Ces registres permettront de restaurer la valeur de l'accu et des flags à la fin de la routine d'interruption.

- Modifier les Registres W_Register et Status_Register pour permettre de restaurer les valeurs de l'accu et des flags à la fin de la routine d'interruption.

- Modifier le séquenceur pour ajouter un état sInterrupt dans lequel on entre lorsque l'on a une interruption. Ensuite, modifier la table de vérité des sorties du séquenceur pour gérer les signaux de sauvegarde et de restauration.

- Ajouter l'instruction de retour d'interruption dans la liste d'instructions et modifier le paquetage nanoProcesseur_package en conséquence.

Modifier l'architecture du nanoControleur pour y ajouter un port de sortie de 8 bits, sur lequel seront connectées les 8 leds bicolores rouges (bit 3..0) et vertes (bit 7..4) du Kit Xilinx 7.

Modifier les schémas blocs réalisés lors du premier rendu pour correspondre à votre nouvelle architecture.

Modifier votre plan mémoire du nanoControleur pour y ajouter le port de sortie décrit ci-dessus.

Modifier le jeu d'instructions pour y ajouter les nouvelles instructions demandées ci-dessus.

Rendu 4 :

Ajouter et/ou modifier tous les fichiers VHDL structurels du nanoControleur en fonction des modifications réalisées pour le troisième rendu sur vos schémas bloc.

Ajouter et/ou modifier tous les fichiers VHDL comportementaux du nanoControleur en fonction des modifications réalisées pour le troisième rendu.

Modifier et adapter le fichier de contrainte du projet.

Ecrire une application en assembleur tournant en boucle selon la description ci-dessous :

- Allumer 4 leds bicolores rouges.
- Lire l'état des 8 dilswitch 1.
- Sauvegarder la valeur des dilswitch 1 en RAM.
- Ecrire l'état de ces 8 dilswitch sur 8 leds du barre graphe 1.
- Créer une temporisation en utilisant la valeur des 8 dilswitch sauvegardée en RAM comme variable (n). Pour ce faire, appeler une sous-routine qui décrémente n fois cette variable jusqu'à 0.
- Dans cette sous-routine appeler une deuxième sous-routine qui décrémente n fois cette variable jusqu'à 0.

- Dans cette deuxième sous-routine appeler une troisième sous-routine qui décrémente n fois cette variable jusqu'à 0.
- A la fin de la temporisation, inverser l'état des 4 leds bicolores rouges/vertes.
- Revenir au deuxième point (lecture des dilswitch).

Si une interruption survient, on doit sauter immédiatement dans la routine d'interruption qui doit :

- Lire l'état des 8 dilswitch 2.
- Ecrire la valeur de ces 8 dilswitch sur 8 leds du barre graphe 2.
- Quitter la routine d'interruption et continuer l'exécution du programme à l'endroit où il s'était interrompu.

Rendu 5 :

Créer dans le projet Vivado un banc de test (testbench) VHDL permettant de valider totalement l'application ci-dessus.

Analyser la simulation et être capable d'expliquer le déroulement d'instructions dans le nanoControleur.

Synthétiser, implémenter et programmer l'application sur le Kit Xilinx 7 et tester le système numérique réalisé sur le Kit Xilinx 7.