Projet : VHDL

Réalisation d'un processeur

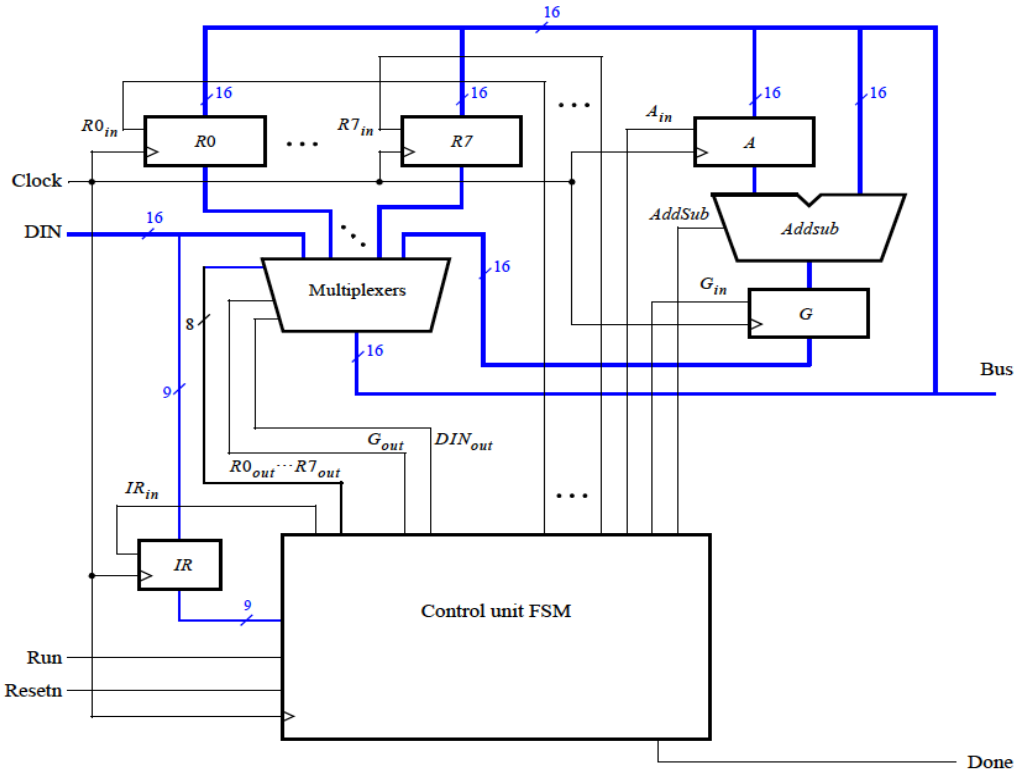
Normalement, vous ne devez créer qu'un seul projet QUARTUS dans un répertoire de votre espace, et **celui-ci a déjà été créé lors de la première séance**. Vous réutiliserez les fichiers VHDL développés au fur et à mesure des séances.

Rappel sur le lancement de quartus :
-Ouvrir le terminal, puis taper cd **/home/embed/intelFPGA_lite/18.1/quartus/bin** puis taper **./quartus** (ne pas oublier le « ./ »)

• Introduction

L'objectif de ce projet est d'implémenter un processeur en se basant sur les modules que vous avez conçu lors des séances précédentes de TP.

La figure suivante montre l'architecture du système numérique composé de registres 16 bits, un multiplexeur, un additionneur/soustracteur et une unité de contrôle (implémentée comme une FSM).



Fonctionnement :

Les données entrantes passent par le bus de 16 bits nommé DIN et peuvent être rangées dans un des registres (R0…R7 ou A) en sélectionnant le chemin approprié dans le multiplexeur. Ce multiplexeur permet aussi de déplacer les données d'un registre vers un autre via son bus de sortie.

Pour réaliser une opération dans l'unité arithmétique et logique (Addition ou soustraction dans un premier temps), le multiplexeur doit être commandé par l'unité de contrôle pour positionner le premier opérande sur le bus pour la placer dans le registre A . Puis, au cycle suivant, un seconde opérande doit être positionné sur le bus.

L'opération est alors réalisée entre le registre A et le contenu du bus. Le résultat est rangé dans le registre G et peut ensuite être déplacé si cela est nécessaire.

Contrôle de l'architecture :

Les signaux de commande sortant de l'unité de contrôle permettent de sélectionner les opérations réalisées par chaque composant. Par exemple, si les signaux R0out et Ain sont actifs, alors c'est le registre R0 qui écrira sur le bus pour ranger son contenu au cycle suivant dans le registre A .

La configuration de l'ensemble des signaux de commande à un instant donné correspond à l'exécution d'une instruction du processeur. La Table 1 liste les instructions qui seront supportées par cette première version du processeur. Dans la syntaxe assembleur, l'écriture RX <- [RY] signifie que le contenu du registre RY est chargé dans le registre RX (sans modifié le contenu de RY), ce qui correspond à une instruction move **mv**. Une instruction **mvi** (move immediate) permet de spécifier une valeur D de 16 bits à placer dans le registre RX soit RX <- D.

Opération	Fonction réalisée
mv Rx,Ry	$Rx \leftarrow [Ry]$
mvi Rx,#D	$Rx \leftarrow D$
add Rx, Ry	$Rx \leftarrow [Rx] + [Ry]$
sub Rx, Ry	$Rx \leftarrow [Rx] - [Ry]$

Table 1 : Instructions réalisées par le processeur

A chaque cycle, une nouvelle instruction est donc chargée depuis le bus DIN dans le registre IR de 9 bits. Ces instructions sont encodées suivant 2 formats.

Dans le format R (registre), les 9 bits de l'instruction sont organisés de la manière suivante : IIIXXXYYY

avec

III représentant le code opération (codop) de l'instruction à réaliser sur 3 bits,

XXX, le numéro du registre RX (3 bits)

YYY, le code le registre RY(3 bits).

A noter que l'on utilise 3 bits pour le codop pour convenir aux futures extensions du processeur.

Dans le format I (Immédiat), les 9 bits sont organisés de la manière suivante : IIIXXX000. En effet, avec ce formalisme, on charge uniquement une valeur de 16 bits (constante), lue sur le port DIN, directement le registre RX. Le registre RY n'est quant à lui pas utilisé.

Les instructions utilisant l'UAL sont codées selon le format R mais utilisent plusieurs cycles pour s'exécuter à cause des multiples transferts à opérer sur le bus. C'est alors la machine d'états de l'unité de contrôle qui dirige ces opérations de transfert à chaque cycle. Une opération n'est exécutée que si le signal d'entrée Run est actif, et le processeur indique qu'une instruction est terminée en activant le signal de sortie Done. La Table 2 indique quels sont les signaux de commande à activer à chaque cycle pour réaliser les instructions de la Table 1. Une étape préalable (non représentée) à toute instruction est l'étape T0 pendant laquelle le signal IRin est activé pour stocker la nouvelle instruction à décoder.

	T_1	T_2	T_3
(**mv**): I_0	$RY_{out}, RX_{in},$ *Done*		
(**mvi**): I_1	$DIN_{out}, RX_{in},$ *Done*		
(**add**): I_2	RX_{out}, A_{in}	RY_{out}, G_{in}	$G_{out}, RX_{in},$ *Done*
(**sub**): I_3	RX_{out}, A_{in}	$RY_{out}, G_{in},$ *AddSub*	$G_{out}, RX_{in},$ *Done*

Table 2 : Signaux de contrôle positionnés à chaque étape d'exécution des instructions

- # Réalisation

1. Première partie

Réaliser la description en VHDL du processeur.

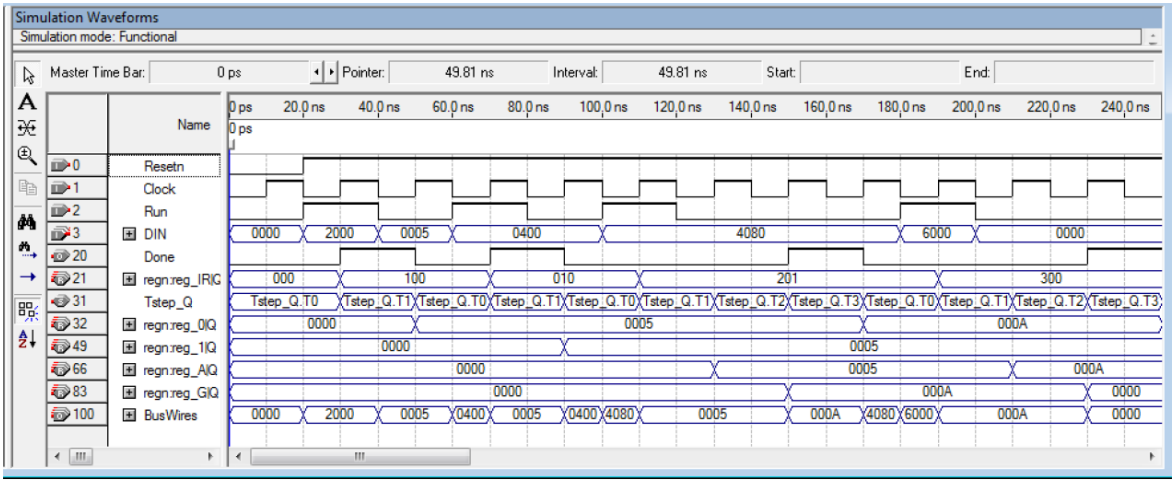
a. Ouvrer votre projet Quartus.

b. On donne un squelette du fichier VHDL principal que l'on nommera Proc

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;
ENTITY proc IS
PORT (
        DIN : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        Resetn, Clock, Run : IN STD_LOGIC;
        Done : BUFFER STD_LOGIC;
        BusWires : BUFFER STD_LOGIC_VECTOR(15 DOWNTO 0));
END proc;

ARCHITECTURE Behavior OF proc IS
… declare components
… declare signals
TYPE State_type IS (T0, T1, T2, T3);
SIGNAL Tstep_Q, Tstep_D: State_type;
…
 BEGIN
I <= IR(1 TO 3);
decX : dec3to8 PORT MAP (IR(4 TO 6), ...);
decY : dec3to8 PORT MAP (IR(7 TO 9), ...);
…
```

c. Utilisez la simulation fonctionnelle pour vérifier le comportement du circuit. Un exemple de simulation est donné dans la figure ci-dessous. Il montre que la valeur $(2000)_{16}$ est chargée dans le registre IR depuis le port DIN à la date 30 ns. Les 9 bits de poids fort (gauche) correspondent à une instruction mvi R0,#D, où D = 5 est une constante chargée dans R0 sur front montant à 50 ns. Puis la simulation montre la lecture de l'instruction mv R1, R0 à 90 ns, add R0, R1 à 110 ns, et sub R0,R0 à 190 ns.



d. Créer une nouvelle entité **Top** dans laquelle vous instancierez votre processeur et dont vous connecterez les ports aux périphériques de la carte : les switches SW15 à SW0 (si non disponible alors on utilisera les données sur un nombre de bits limité) pour écrire sur le bus DIN et le switch SW17 (ou un autre)pour la commande Run. Le bouton KEY0 sera utilisé pour configurer le Resetn et KEY1 pour l'horloge CLK . Connectez le bus de sortie du processeur bus aux LEDR15 à 0 (idem selon le nombre de leds disponible) et le flag Done à LED17 (ou une autre).
e. Compiler le design en incluant le fichier d'assignation des broches si ce n'est pas déjà fait à l'étape d.
f. Tester et valider le fonctionnement

2. Seconde partie : Ajout de mémoire

Dans cette partie, vous allez ajouter une mémoire au processeur pour éviter de positionner manuellement les instructions. Le circuit correspondant est représenté sur la figure suivante. Vous pouvez observer qu'un compteur est utilisé pour incrémenter l'adresse. Dans les processeurs, il est appelé Compteur Ordinal (ou Program Counter).
Pour simplifier le test du circuit, deux horloges sont utilisées : PClock et MClock , la première pour le Processeur, la deuxième pour la Mémoire.

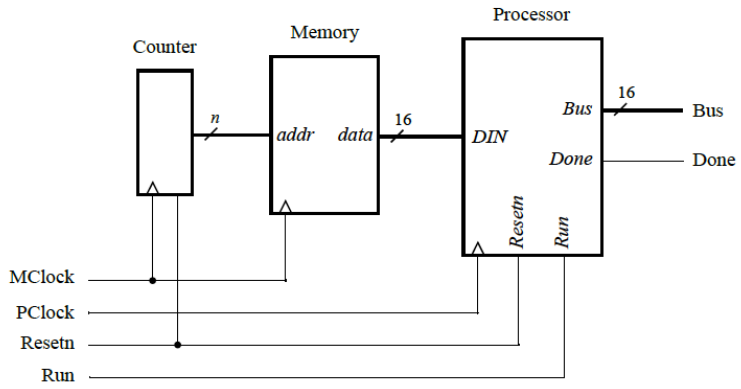
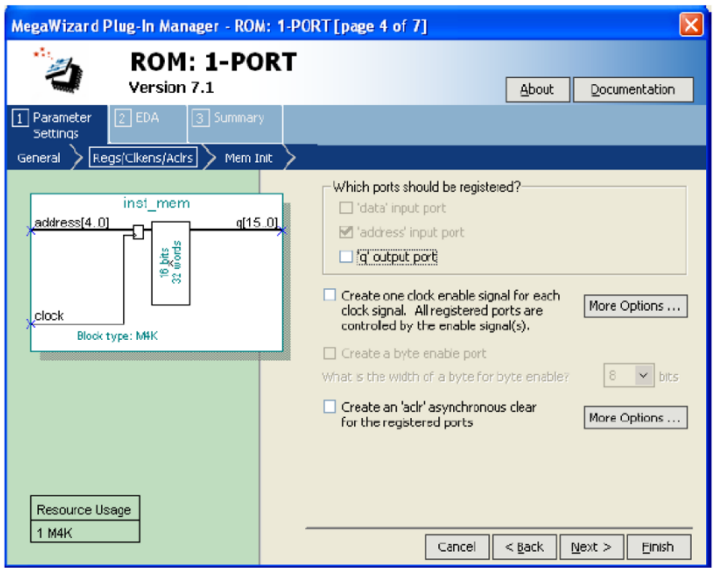


Schéma du processeur avec la mémoire

A faire :

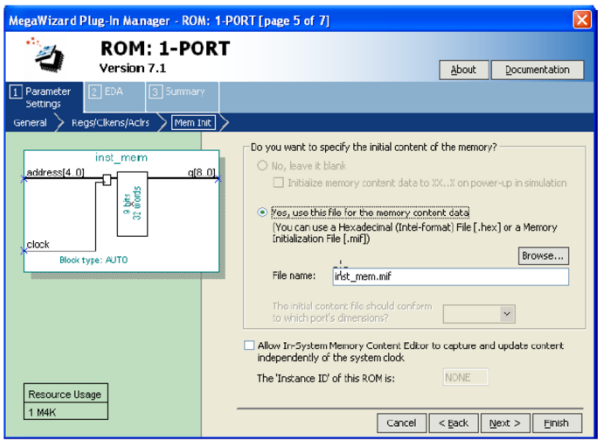
a. Pour générer la mémoire, nous allons utilisé comme lors du lab précédent, l'utilitaire MegaWizard Plug-In Manager. **Tools < MegaWizard Plug-In Manager**

b. Dans le MegaWizard, le composant à choisir se trouve dans la catégorie Memory Compiler et est appelé LPM_ROM : 1-PORT. Suivez les écrans de configuration pour générer une mémoire simple port de lecture dont les données sont sur 16-bits et pouvant mémoriser 32 mots. La Page 4 du Megawizard est indiquée ci-dessous. Puisque cette mémoire n'a pas de port d'écriture elle est appelée synchronous read-only memory (synchronous ROM).



c. Pour placer les instructions dans la mémoire, vous devez spécifier un contenu initial de la mémoire. Celui-ci peut être fourni par un fichier dédié appelé *memory_initialization file (.mif)*.

d. L'écran de configuration est représenté sur la figure suivante :



Utilisez l'aide de Quartus-II pour créer un fichier inst_mem.mif qui dispose de suffisamment d'instructions pour tester le circuit.

e. Effectuer une simulation fonctionnelle pour vérifier que le processeur lit correctement les données issues de la mémoire.
f. Utilisez ensuite le switch SW0 pour commande le port Run, le bouton KEY0 pour le Resetn , KEY1 pour MClock , et KEY2 pour PClock. Comme précédemment le bus de sortie sera relié aux LED. Il faudra trouver une astuce pour pouvoir visualiser la valeur du bus sur le nombre de leds limité dont on dispose.
g. Compilez et testez le fonctionnement sur la carte. Les 2 horloges sont séparées pour éviter de lire plusieurs instructions lors de l'exécution d'instructions multi-cycles (ALU).

3. Extensions

De manière à apporter vos propres modifications au processeur, il vous sera demander de réfléchir et de réaliser l'une des extensions suivantes :

- Ajout d'instruction de type I pour le chargement (load) et le rangement (store) en mémoire qui permettent respectivement de charger et de ranger une donnée depuis ou vers la mémoire. Les opérandes de ces instructions sont un registre et une adresse mémoire codée comme un immédiat de 16 bits.
- Ajout d'instruction de type R pour les opérations logiques (AND, OR, NOT).
- Ajout d'instruction de type R pour la multiplication de deux nombres stockés dans des registres. Vous porterez attention à expliquer comment est traité le résultat dans les registres.
- Ajout d'un nouveau type d'instruction de type J pour réaliser des sauts dans l'exécution des instructions. Ce type d'instructions n'a qu'un seul opérande : une adresse de saut sur 16 bits.