

Sujets de TP d'UE 308  
Electronique Numérique  
Université Paris-Saclay

## TP 1 : Introduction à Quartus et logique combinatoire

### EXERCICE 1 : LA PORTE ET

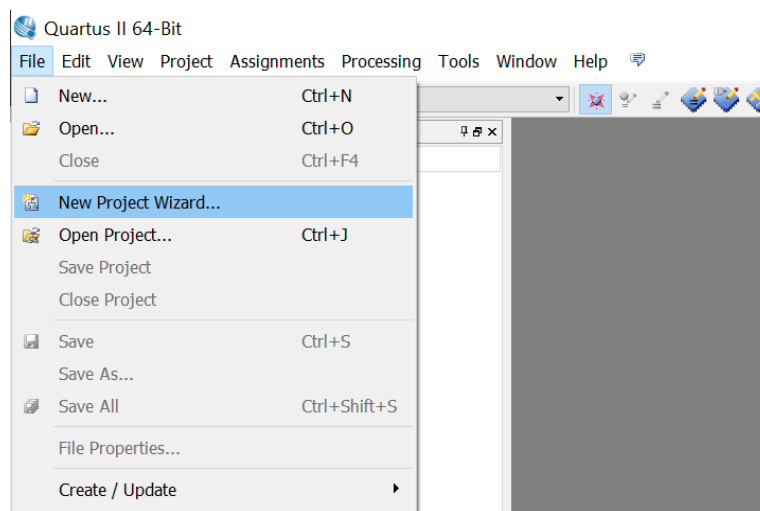
L'outil de conception QUARTUS est un logiciel qui fonctionne par projet. Il est nécessaire de créer un répertoire (dossier) pour chaque projet. Sur votre espace de travail, commencez par créer un dossier « TP308 », puis un sous-dossier « TP1 », puis un sous-dossier « TP1\_Exo1 ». Tous les fichiers créés pour cet exercice devront se trouver dans ce sous-dossier.

Attention : les noms du dossier et du projet créés ne doivent contenir ni "espace" ni "lettre avec accent".

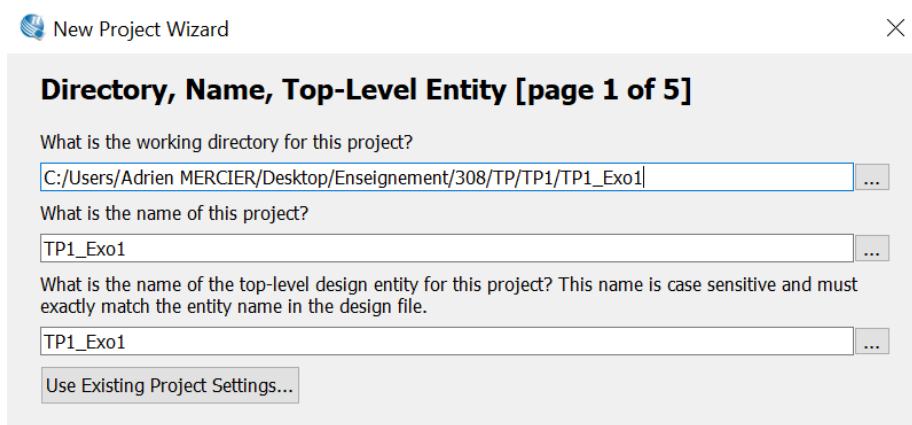
Lancer le logiciel Quartus II 13.0sp1. Si une fenêtre s'ouvre vous demandant un numéro de licence, renseignez-vous auprès du chargé de TP pour qu'il vous communique le numéro de licence. Laissez-vous guider par le tutoriel ci-dessous pour réaliser une porte ET de deux façons différentes.

#### **Création de projet**

Cliquez sur File → New Project Wizard



Renseignez le chemin vers le répertoire de travail que vous venez de créer, ainsi que le nom du projet. Pour une facilité de lecture, pensez à nommer votre projet du même nom que le répertoire.



La page suivante vous demande si vous souhaitez ajouter des fichiers existants. Pour ce premier TP, il n'y a pas de nécessité de le faire. Cliquez sur Next.

New Project Wizard

Add Files [page 2 of 5]

Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

Note: you can always add design files to the project later.

File name:

...

Add

File Name	Type	Library	Design Entry/Synthesis Tool	HDL Version	

Add All

Remove

Up


Down

Properties

Specify the path names of any non-default libraries. User Libraries...

< BackNext >FinishCancelHelp

Sélectionnez ensuite le FPGA correspondant à votre carte de développement. Si vous travaillez sur un cyclone III il faut sélectionner le modèle EP3C16F484C6 et si vous travaillez sur un cyclone V, il vous faut sélectionner le modèle 5CEBA4F23C7


New Project Wizard

## Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.  
You can install additional device support with the Install Devices command on the Tools menu.

Device family

Family: Cyclone III

Devices: All

Target device

☐ Auto device selected by the Fitter

☒ Specific device selected in 'Available devices' list

☐ Other: n/a


Show in 'Available devices' list

Package: Any

Pin count: Any


Speed grade: Any

Name filter:

☒ Show advanced devices
 ☐ HardCopy compatible only
 

Available devices:

Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded mul
EP3C16F484C6	1.2V	15408	347	516096	112
EP3C16F484C7	1.2V	15408	347	516096	112

Companion device 

HardCopy:

☐ Limit DSP & RAM to HardCopy device resources

< Back

Next >

Finish

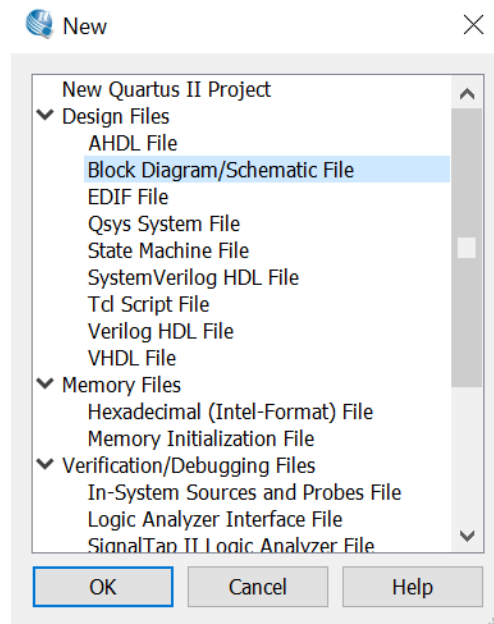
Cancel

Help

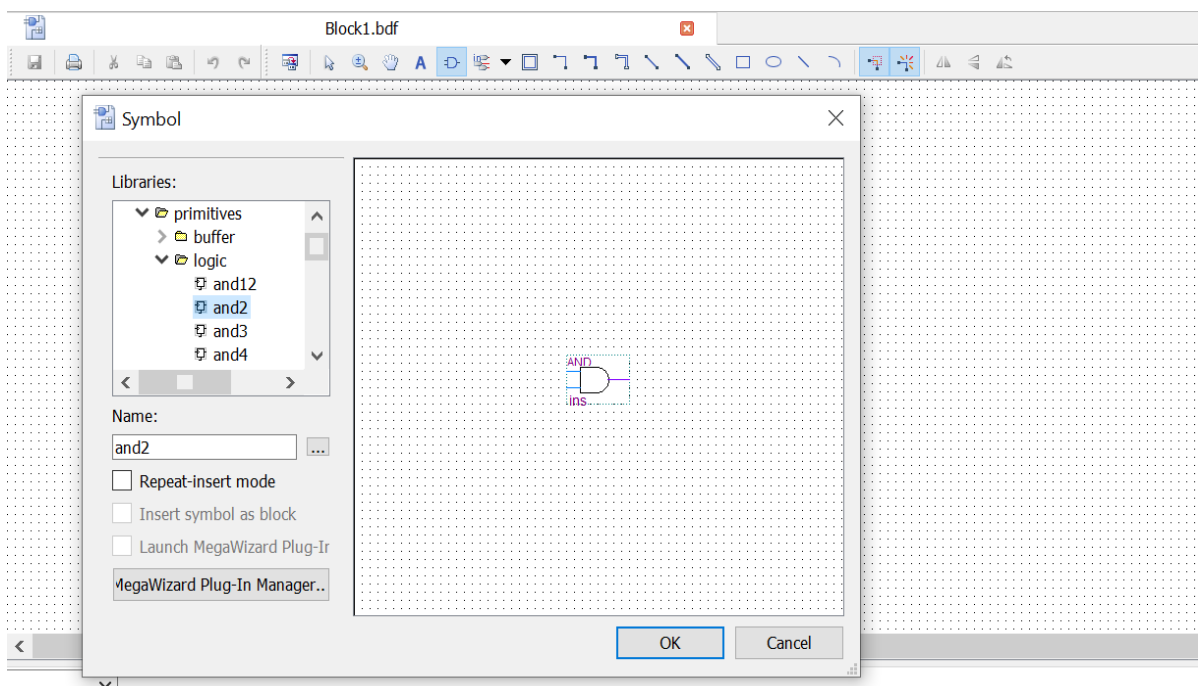
Cliquez sur Finish. On n'a pas besoin de définir des outils supplémentaires. Une page vide devrait vous apparaître, avec dans l'encadré de gauche le nom du FPGA ainsi que le nom du projet.

### Approche schématique du problème

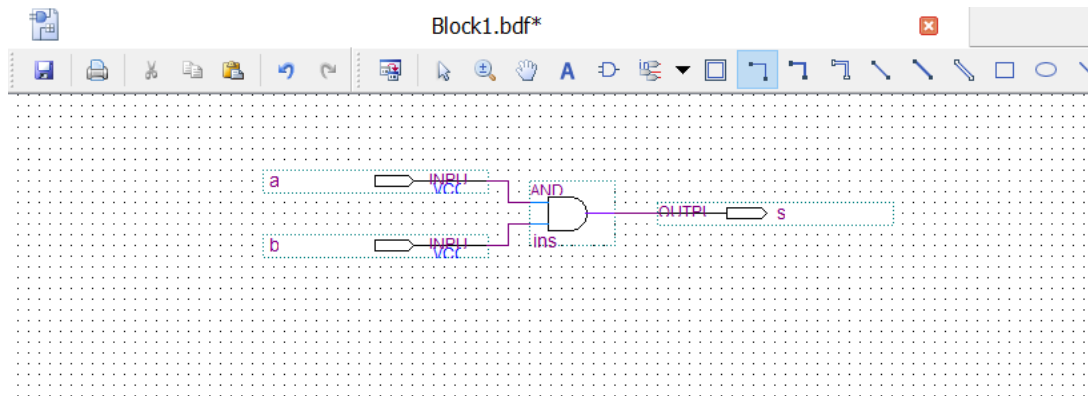
Pour ce premier exemple, il faut réaliser une porte ET à l'image des schémas utilisés en séances de cours ou bien en séances travaux dirigés. Pour cela, cliquez sur File → New, pour sélectionner Block Diagram/Schematic File.



Il apparaît une page blanche avec un quadrillage, ce qui constitue un espace de conception numérique. Pour placer des composants, il faut faire un double click gauche sur la page, ou bien cliquer sur l'icône Symbol Tool dans la barre d'outils. Sélectionnez une porte ET à deux entrées, et placez-là sur la page.

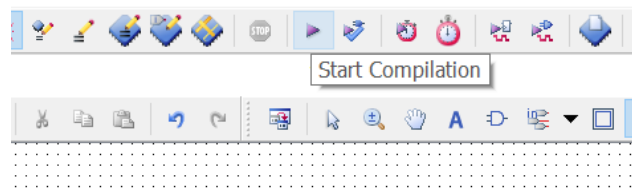


Pour utiliser cette porte, il faut compléter la conception par deux entrées (input) et une sortie (output). Cherchez ces composants dans la même bibliothèque. Pour relier les symboles entre eux, cliquez sur Orthogonal Node Tool pour utiliser des fils de bits. Attention à ne pas confondre avec Orthogonal Bus Tool qui sert à tracer des fils de bus (trait épais). Enfin, renommez les entrées/sorties en double cliquant sur les noms, ou bien à l'aide d'un clic droit sur les entrées/sorties puis dans l'onglet Properties.

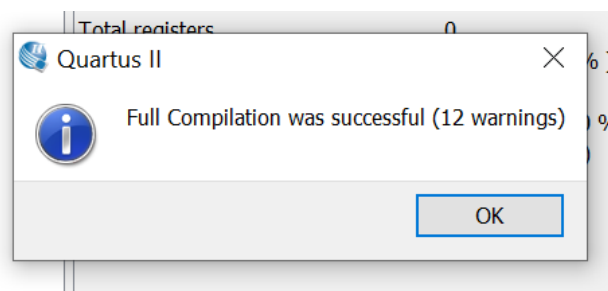


Une fois le schéma terminé, il faut sauvegarder le fichier sous « TP1\_Exo1.bdf » dans le dossier « TP1\_Exo1 ». Attention, le nom de fichier doit être le même que celui du projet. Dans votre répertoire, il se trouve donc au moins un fichier avec l'extension .qpf qui correspond au projet, et un fichier .bdf qui correspond au schématique, et d'autres fichiers et dossiers dont on ne s'occupe pas.

Il faut maintenant procéder à la compilation de cette description schématique avant de pouvoir l'implémenter sur la carte. Cliquez sur Start Compilation. Cette étape correspond à la phase de synthèse de l'architecture correspondante au circuit conçu. Elle peut durer plusieurs dizaines de secondes.



Si besoin, corrigez votre schéma à l'aide du débogueur jusqu'à obtenir une compilation Successful. Un fichier d'extension ".sof" devrait être généré dans le dossier output\_files. C'est ce fichier qui sera téléversé plus tard dans le FPGA.



Ensuite, il faut renseigner la correspondance entre les entrées/sorties de la description du circuit logique avec celles qui sont sur la carte de développement. Pour cela, cliquez sur Assignements → Pin Planner. Un plan devrait apparaître avec toutes les pins du FPGA. Si la compilation s'est déroulée correctement, un tableau doit contenir vos variables. Il faut remplir l'assignation des entrées/sorties aux bonnes coordonnées. En fonction du FPGA de votre carte (EP3C16F484C6 ou 5CEBA4F23C7), renseignez la colonne Location.

Variable	Nom sur la carte	EP3C16F484C6	5CEBA4F23C7
a	SW[0]	PIN_J6	PIN_U13
b	SW[1]	PIN_H5	PIN_V13
s	LEDG[0] ou LEDR[0]	PIN_J1	PIN_AA2

Une fois que l'assignement est fait, fermez la fenêtre.

Suite à l'adressage des entrées/sorties, **il faut à nouveau compiler en cliquant sur Start Compilation.**

Connecter la carte DE0 à l'ordinateur via le câble USB et éventuellement l'alimentation sur secteur. Allumer la carte en appuyant sur le bouton rouge, et vérifier que l'interrupteur situé à gauche des afficheurs 7 segments soit bien sur la position RUN et non sur la position PROG. Un programme de démonstration sur les afficheurs devrait se lancer automatiquement.

Pour configurer le circuit FPGA (c'est-à-dire **téléverser la description du circuit sur la cible**), cliquez sur l'icône Programmer.



Cliquer sur Add File, puis dans le dossier output\_files, et enfin sélectionnez le fichier d'extension ".sof". Il faut aussi vérifier que la carte est bien vue par le logiciel. Pour cela, assurez-vous en cliquant sur Hardware Setup que USB-Blaster est bien sélectionné. Cliquer ensuite sur Start. Si toutes les étapes ont été bien suivies, il devrait apparaître l'indication Successful dans la barre Progress.

Rechercher les deux interrupteurs SW[0] et SW[1] sur la carte, et testez ensuite le résultat en vérifiant les différents états possibles de la LED.

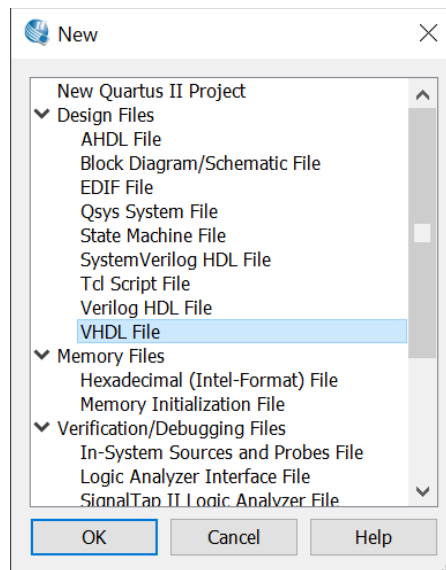
Avant de passer à l'exercice suivant, il faut fermer le projet en cliquant sur File → Close Project.

### **Approche mixte (schématique/VHDL) du problème : méthode 1**

On souhaite faire le même exercice, mais en créant la porte logique ET en VHDL. Créez un nouveau répertoire « TP1\_Exo1\_2 », puis créez un New Project Wizard « TP1\_Exo1\_2 » de la même manière que précédemment. Cliquer à nouveau sur File → New → Bloc Diagram/Schematic File.

Il faut immédiatement sauvegarder le fichier en cliquant sur File → Save As, et il faut que le fichier .bdf soit placé dans le répertoire « TP1\_Exo1\_2 » et avec le même nom que le projet « TP1\_Exo1\_2 ».

Par la suite, pour coder en VHDL, il faut créer un nouveau fichier en cliquant sur New → VHDL File



Une page blanche apparaît, prête à être remplie par du code en VHDL. Commencez à rédiger la description du circuit par les bibliothèques usuelles.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
```

Définissez l'entité et les ports d'entrées/sorties.

```
5  ENTITY PORTE IS          -- l'entity ne doit pas avoir le même nom que le projet
6  PORT (
7      a, b : IN STD_LOGIC;
8      s : OUT STD_LOGIC    -- /\ pas de ; sur la dernière ligne
9  );
10 END PORTE;
11
```

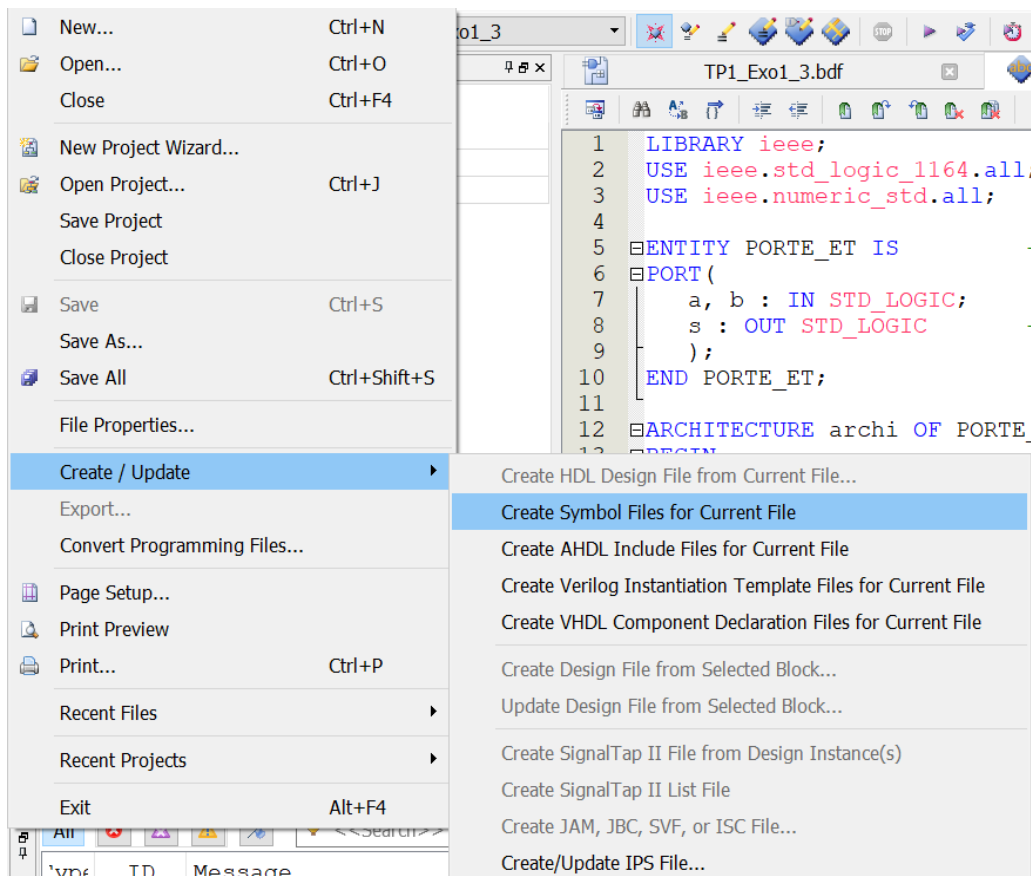
Décrivez ensuite l'architecture.

```
12 ARCHITECTURE archi OF PORTE IS
13 BEGIN
14
15     s <= a AND b;
16
17 END archi;
```

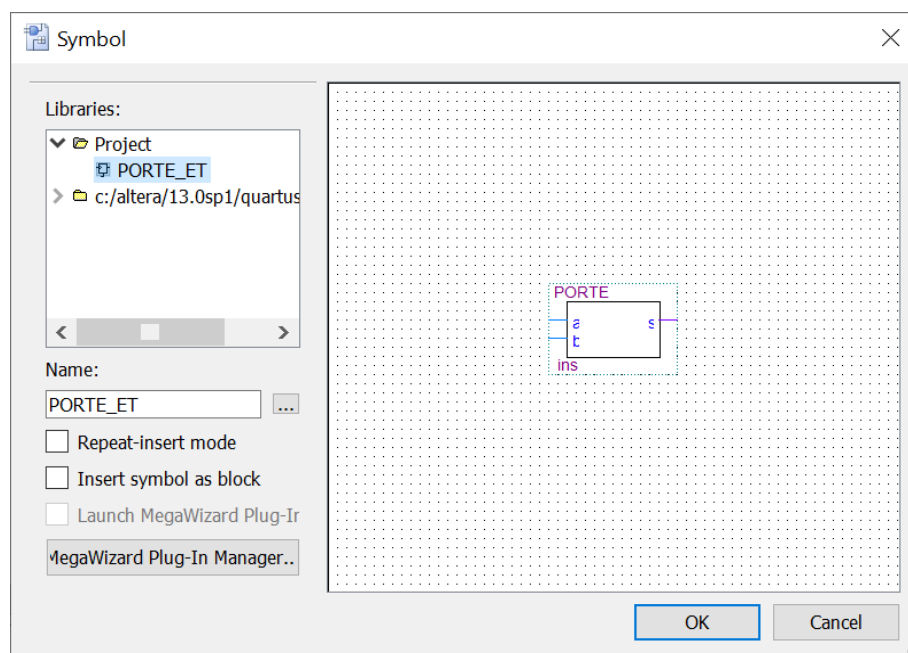
Pour le cas de cet exemple, il n'y a pas besoin d'utiliser de Process. Dans les exercices qui suivront, il faudra en utiliser un la plupart du temps. Prenez donc le réflexe de le coder dès maintenant.

```
12 ARCHITECTURE archi OF PORTE IS
13 BEGIN
14     PROCESS(a,b)          -- liste des variables d'entrées
15     BEGIN
16
17         s <= a AND b;
18
19     END PROCESS;
20 END archi;
```

Une fois la description complète, enregistrer le fichier .vhd qui doit avoir le même nom que l'entité, et bien sûr un nom différent de celui du projet. Par exemple, « porte.vhd ». Il faut ensuite créer un composant pour l'utiliser dans le fichier schématique. Pour cela, cliquez sur File → Create/Update → Create Symbol File for Current File.

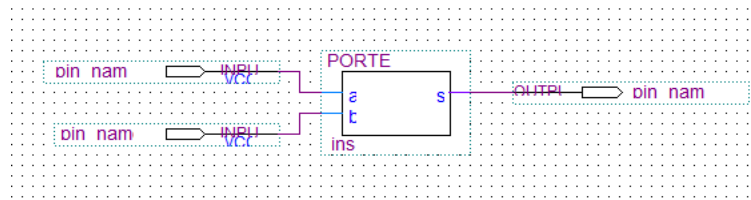


Si cette opération réussit, retournez alors sur la planche à dessin. Cliquez sur l'icône Symbol, et cherchez le nouveau composant dans les bibliothèques.

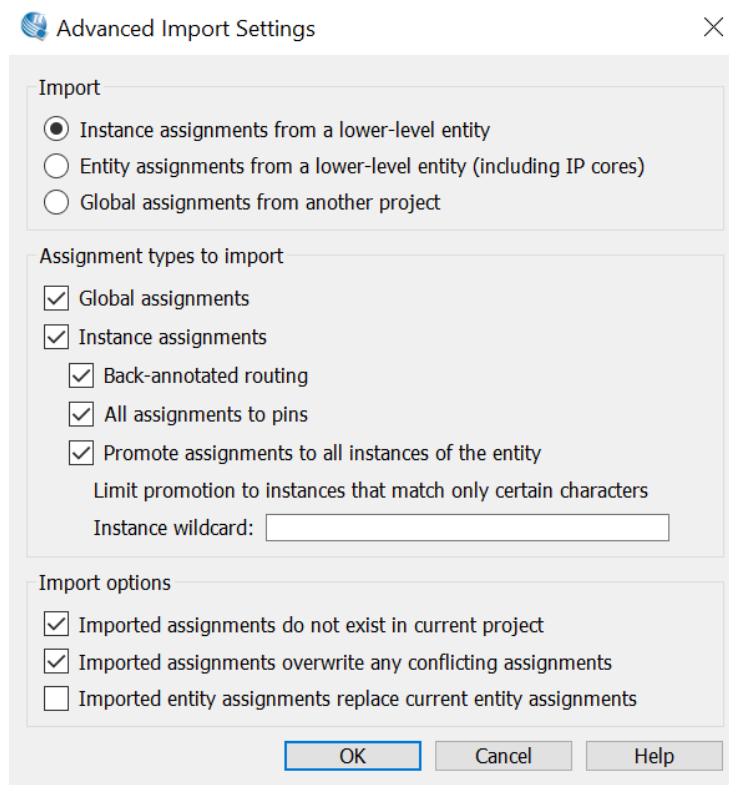




Vous pouvez alors le placer, et le relier à des ports d'entrées/sorties.



A ce stade, il faut à nouveau assigner les ports d'entrées/sorties aux bonnes pins. **Contrairement à l'exemple précédent, il est possible de faire l'assignation automatiquement (on utilisera toujours cette méthode par la suite).** Pour cela, il faut cliquer sur Assignments → Import Assignments, et aller chercher le fichier dans R:\Depart\_Phys\L3-E3A\UE308. Le fichier est DE0\_POPS.qsf pour les cartes avec un Cyclone III et DE0\_CV.qsf pour les cartes avec un cyclone V. Avant de valider définitivement le fichier, il faut également cocher la case Global assignments accessible dans l'option Advanced.



Il faut alors remplacer les noms des entrées/sorties du dessin par les noms disponibles dans le Pin Planner, **puis ne pas oublier de compiler**. Par exemple, il faut changer « pin\_name1 » par « SW[0] ».

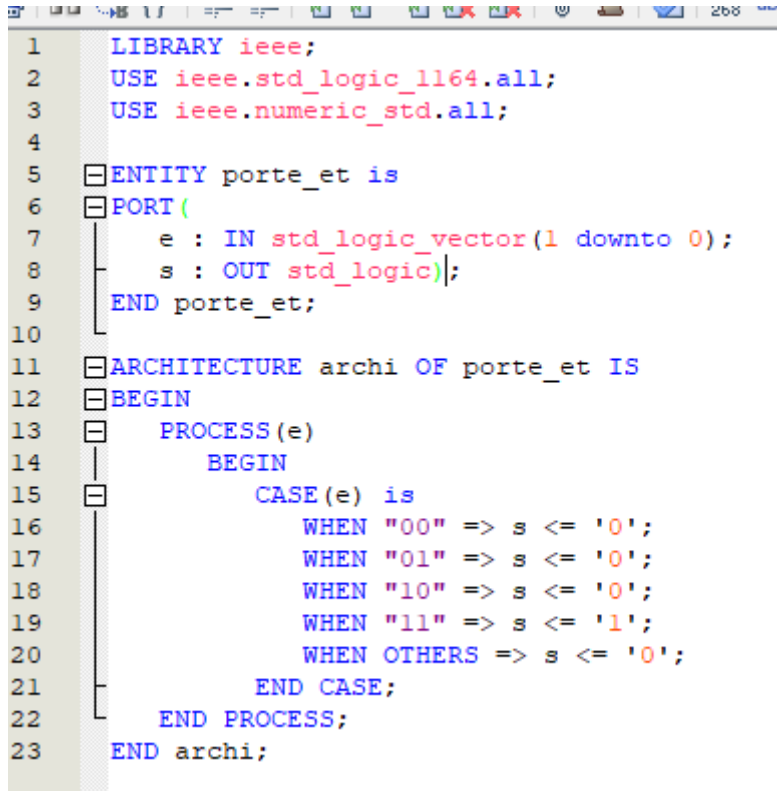
**Téléverser la description** sur le FPGA en utilisant à nouveau le Programmer comme ci-avant.

### **Approche mixte (schématique/VHDL) du problème : méthode 2**

Il est possible de reprendre exactement la méthode 1 mais en écrivant directement la table de vérité dans la description VHDL plutôt que la description liée l'équation logique. Cette

approche est pertinente lorsque le nombre de sortie est important afin d'éviter de longues études des tableaux de Karnaugh.

Pour procéder, il faut à nouveau créer un projet « TP1\_Exo1\_3 » dans un dossier du même nom. Il faut également créer un premier fichier Block Diagram/Schematic File à enregistrer avec le même nom que le projet, puis un second fichier VHDL File à enregistrer avec le même nom que l'entity. Dans la partie architecture de la description VHDL, il est possible d'écrire la table de vérité en utilisant les fonctions CASE et WHEN.



```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  ENTITY porte_et IS
6  PORT (
7      e : IN std_logic_vector(1 downto 0);
8      s : OUT std_logic);
9  END porte_et;
10
11 ARCHITECTURE archi OF porte_et IS
12 BEGIN
13     PROCESS(e)
14     BEGIN
15         CASE(e) IS
16             WHEN "00" => s <= '0';
17             WHEN "01" => s <= '0';
18             WHEN "10" => s <= '0';
19             WHEN "11" => s <= '1';
20             WHEN OTHERS => s <= '0';
21         END CASE;
22     END PROCESS;
23 END archi;
```

Tout le reste de la procédure est identique à celui de l'approche mixte, méthode 1.

## EXERCICE 2 : LE LOUP, LA CHÈVRE ET LES CHOUX

Jean-Michel Lefermier doit passer la rivière dans une barque juste assez grande pour lui et son loup, ou lui et sa chèvre, ou lui et ses choux. Les choux seront mangés s'il les laisse seuls, sur une des deux berges, avec la chèvre, et la chèvre sera mangée s'il la laisse seule, sur une des deux berges, avec le loup. Comment faire passer tout ce monde sans dégâts ?

Pour traiter le problème, on notera les variables d'entrées F, L, V, X correspondant respectivement au fermier, au loup, à la chèvre, et aux choux. Une valeur à '0' correspond à une présence sur la berge du bas, et une valeur à '1' correspond à une présence sur la berge du haut. On choisira une variable de sortie A correspondant à une alarme si la chèvre ou les choux se font manger.

**Q1 Préparation** : Commencez par réaliser la table de vérité de ce problème. Simplifiez l'équation logique à l'aide de l'algèbre de Boole ou bien d'un tableau de Karnaugh.

**Q2** : Implémentez ce système sur la carte en utilisant une approche schématique.

**Q3** : Implémentez ce système sur la carte en utilisant une approche mixte.

## EXERCICE 3 : AFFICHEUR 7 SEGMENTS

On souhaite réaliser un décodeur pour afficheur 7 segments qui décode un nombre hexadécimal N codé sur 4 bits,  $N=(n_3n_2n_1n_0)_2$  ; N varie de  $(0)_{16}$  à  $(F)_{16}$  en hexadécimal et peut donc être affiché sur un seul afficheur. **On tiendra compte du contexte de la carte ALTERA : les segments sont allumés avec une commande à '0'**. Les lettres b et d seront codées en minuscules (les autres en majuscules). Le chiffre 1 sera codé avec les segments de droite de l'afficheur. Les sorties du premier afficheur 7 segments sur la carte DE0 sont noté/codé avec les affectations HEX0[ a] avec  $a=0,1,2,3,...,6$

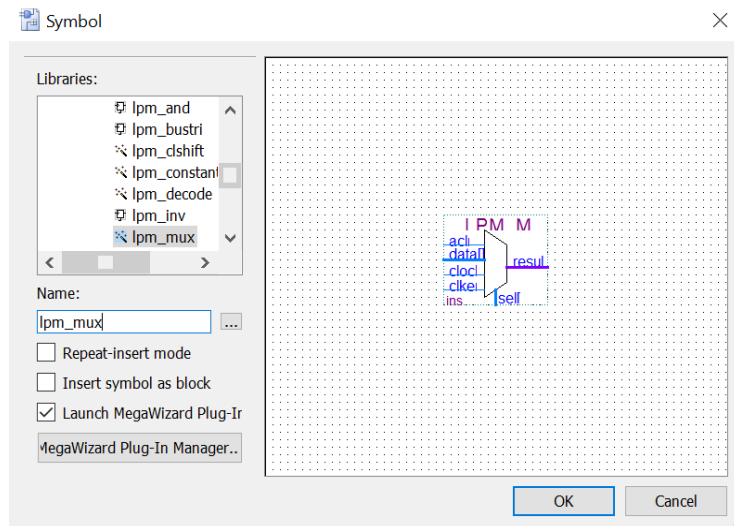
**Q1 Préparation** : Réalisez la table de vérité de ce décodeur. Les lettres seront écrites en majuscule sauf pour les lettres b et d (en minuscule) et le chiffre 1 sera écrit en utilisant les deux segments les plus à droite.

**Q2** : Implémentez la description du circuit par une l'approche qui vous paraît la plus pertinente.

## EXERCICE 4 : DÉCALAGE PAR MULTIPLEXEUR

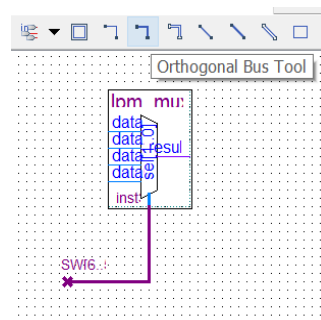
On souhaite réaliser un décalage circulaire de bits dans un mot de 4 bits. Les interrupteurs 1 à 4 servent à coder le mot d'entrée. Les interrupteurs 5 et 6 servent à indiquer de combien de cases faut-il faire un décalage vers la gauche : 0, 1, 2, ou 3 cases. Les LEDs 1 à 4 servent à visualiser le résultat. Pour cela, en schématique, on se sert de quatre « lpm\_mux ». Par exemple, le bit sur les interrupteurs 1 à 4 est  $(n_3n_2n_1n_0)_2$  et que l'on sélectionne sur les interrupteurs  $(00)_2$  on obtient sur les LED  $(n_3n_2n_1n_0)_2$ . Si l'on sélectionne sur les interrupteurs  $(01)_2$  on obtient sur les LED  $(n_2n_1n_0n_3)_2$ . Si l'on sélectionne sur les interrupteurs  $(10)_2$  on obtient sur les LED  $(n_1n_0n_3n_2)_2$ . Etc.

**Q1 Préparation** : A l'aide exclusivement de 4 multiplexeurs 4 vers 1, réaliser le schéma de ce circuit en reliant astucieusement les entrées, les sélections, et les sorties pour décaler les bits correctement.



**Q2 :** Sélectionnez le composant « lpm\_mux ». **Vous devez alors le configurer**, en indiquant qu'il doit posséder quatre entrées. Placez quatre fois ce composant sur une planche à dessin.

**Q3 :** Reliez astucieusement les entrées, les sélections, et les sorties pour décaler les bits correctement. Il est nécessaire d'utiliser des fils de bus pour la sélection.



Vous devez nommer ces fils correctement (click droit → Properties) pour les relier aux entrées. Par exemple, « SW[6..5] » permet de faire correspondre ce fil de bus aux interrupteurs 5 et 6, à condition qu'ils apparaissent tout de même sur la planche à dessin. Il n'est pas possible de connecter par un trait une entrée simple sur 1 bit et un bus. **De plus, le multiplexeur n'a pas les mêmes assignations d'entrées vis-à-vis des commandes que ceux vu en cours/TD. Ici tout est inversé. Par exemple, la commande «  $(00)_2$  » relie l'entrée la plus basse (Data 0) à la sortie.**

## TP 2 : Arithmétique

### EXERCICE 1 : COMPAREUR DE NOMBRES

**Q1 Préparation** : Ecrire la table de vérité avec, en entrée, un nombre binaire sur 4 bits et qui donne une sortie haute ( $=1$ ) quand le nombre binaire est supérieur ou égal à  $(0010)_2$  et inférieur ou égal à  $(1010)_2$ .

**Q2** : Concevez une description en VHDL de ce circuit en utilisant la fonction CASE.

**Q3** : Concevez une description en VHDL de ce circuit en utilisant la fonction IF ainsi que les conversions de types. Quelle approche vous paraît la plus pertinente si on augmente le nombre de bit du mot binaire d'entrée ?

### EXERCICE 2 : COMPAREUR DE NOMBRES INCONNUS

Soient deux nombres binaires (non signés) codés sur 5 bits. On cherche à les comparer pour savoir si le premier est supérieur au second. On se placera ici dans le cadre d'une conception itérative (comme pour l'additionneur binaire vu en cours).

**Q1 Préparation** : refaire l'exercice 8 du TD2 en fixant  $N=5$  bits.

**Q2** : Par utilisation mixte entre schématique et VHDL, rédigez une description en VHDL qui permet de comparer deux bits d'entrées. Une première sortie doit être à '1' si le premier bit est strictement supérieur au second. Une seconde sortie doit être à '1' si les deux bits sont égaux. Créez un composant à partir de cette description pour l'inclure dans une planche à dessin.

**Q3** : Répétez cette étape de comparaison bit à bit pour comparer les deux nombres. La sortie finale doit être à '1' si le premier nombre est strictement supérieur au second.

### EXERCICE 3 : SOUSTRACTEUR DE NOMBRES POSITIFS

Pour fabriquer un soustracteur entre deux nombres binaires de même taille, il existe plusieurs méthodes. On peut considérer, par exemple, un additionneur classique mais avec un des deux nombres codé en complément à 2. Ici on va suivre la méthode (plus directe) qui a été proposée pour l'additionneur classique, à savoir, l'étude d'un demi-soustracteur (soustraction sur un bit) puis, par un raisonnement itératif, l'extension au cas de deux nombres de  $N$  bits chacun.

**Q1 Préparation** : Reprendre l'exercice 5 du TD3. Coder en complément à 2 (sur 2 bits) les chiffres décimaux signés suivants :  $(-1)_{10s}$ ,  $(0)_{10s}$  et  $(1)_{10s}$ . En déduire la table de vérité du soustracteur de deux mots de un bit chacun (attention à la retenue). Simplifier les équations des sorties et donner le circuit logique du demi-soustracteur. **Par un raisonnement itératif similaire à la conception d'un circuit additionneur**, proposer un schéma détaillé pour le soustracteur complet sur  $N$  bits.

**Q2** : On cherche à soustraire deux nombres entiers positifs, chacun étant codé sur quatre bits **non signés**. Le résultat de la soustraction est donc compris entre  $(-F)_{16s}$  et  $(F)_{16s}$ , et nécessite donc 5 bits. En schématique, réalisez cette soustraction **par conception itérative** et affichez le résultat sur 5 LEDs.

**Q2 bis** (facultatif): Décrire la même fonction de soustraction en utilisant cette fois-ci l'opérateur « - » ainsi que les conversions de type (SIGNED, INTEGER, etc.)

**Q3 :** En utilisant l'un des circuits précédents, créer une nouvelle description VHDL (vous aurez donc deux descriptions VHDL pour cette question) pour afficher le résultat de cette soustraction en nombre décimal sur autant d'afficheurs 7 segments que nécessaire.

#### EXERCICE 4 : DÉCODEUR BINAIRE VERS BCD (BINARY CODED DECIMAL)

Le BCD est une autre façon de représenter un nombre entier. En binaire naturel, un nombre décimal trouve son équivalent par l'intermédiaire des puissances de 2, de telle sorte que  $(18)_{10} = (10010)_2$ . En BCD, chaque chiffre d'un nombre décimal est codé en binaire sur quatre chiffres, de telle sorte que  $(18)_{10} = (0001\ 0100)_{\text{BCD}}$ . En conséquence, certains nombres comme  $(1101\ 0101)_{\text{BCD}}$  n'ont pas d'équivalent en décimal. Le BCD est très répandu dès lors que des valeurs numériques doivent être affichées.

**Q :** Soit un nombre compris entre  $(00)_{10}$  et  $(99)_{10}$ . Codé en binaire, ce nombre est compris entre  $(0)_2$  et  $(1100011)_2$ . Réalisez un circuit en VHDL pour passer du binaire naturel au décimal codé binaire, en passant par le décimal (**on utilisera obligatoirement une conversion avec des INTEGER**). Afin de « séparer » les unités des décimales en base 10, on effectuera une division par 10 (et un reste). Afficher ce nombre sur deux afficheurs 7 segments. Le nombre à afficher est représenté par la position des interrupteurs 0 à 6.

## TP3 : Logique séquentielle

### EXERCICE 1 : REGISTRE À DÉCALAGE PRECHARGEABLE

Nous allons maintenant utiliser la brique de base des systèmes séquentiels : la bascule D (DFF sous Quartus). Nous allons construire tout d'abord un registre, circuit qui permet de mémoriser un mot de plusieurs bits, puis nous le modifierons pour que le mot mémorisé puisse être décalé vers la gauche ou vers la droite d'une colonne par coup d'horloge.

Le registre sera codé sur 4 bits et fera donc appel à 4 bascules D (1 bascule par bit). Toutes ces bascules sont utilisées de manière synchrone : leur horloge est la même.

Sur la carte de développement, nous n'avons accès qu'à un quartz qui permet de fournir un signal d'horloge à 50MHz. Cette fréquence est très élevée pour notre application et il nous faut donc la diviser pour créer le signal d'horloge de notre registre.

Pour cela, nous utiliserons le signal de la carte CLOCK\_50 comme horloge d'un compteur de 26 bits dont le bit de poids fort (Q25) constituera le signal d'horloge du registre. Il varie en effet à une fréquence  $2^{26}$  fois plus faible que CLOCK\_50. Ce compteur sera créé en configurant le composant lpm\_counter en mettant juste son nombre de bits à 26.

**Préparation** : réfléchir aux questions Q1 et Q2 ci-dessous pour proposer un schéma que vous implémenterez lors de la séance de TP. **On s'inspire de l'exercice 6 du TD3.**

**Q1** : En schématique, utilisez 4 bascules et 4 multiplexeurs 2 vers 1 (comme ceux de l'exercice 4 du TP1) pour créer un registre préchargeable commandé par l'interrupteur SW[8] : quand il vaut 0, le registre mémorise sa valeur actuelle ; quand il vaut 1, le registre charge les valeurs des interrupteurs SW[3], SW[2], SW[1] et SW[0]. On s'inspirera de la bascule E étudiée lors de l'exercice 1 du TD3.

**Q2** : Remplacez les multiplexeurs 2 vers 1 par des multiplexeurs 4 vers 1 (toujours avec le composant lpm\_mux) et les connecter de manière à obtenir un registre commandé maintenant par SW[9..8] de la manière suivante : Pour « 00 », mode mémoire ; pour « 01 », chargement de la valeur de SW[3..0] ; pour « 10 », décalage d'une colonne vers la droite à chaque coup d'horloge de la valeur du registre (le bit de poids fort est remplacé par un 0) ; pour « 11 », décalage d'une colonne vers la gauche de la valeur du registre (le bit de poids faible est remplacé par un 0).

**Q3** : Réalisez le circuit précédent en VHDL mais pour une taille de N bits. Rajouter 2 entrées de 1 bit pour pouvoir contrôler la valeur injectée sur le bit de poids fort (respectivement faible) dans le cas d'un décalage à droite (respectivement à gauche), et deux sorties de 1 bit pour pouvoir récupérer les bits qui sortent du registre lors de chaque décalage. Prévoir aussi une entrée reset qui permet de mettre à 0 le registre quand reset =1.

## EXERCICE 2 : CHRONOMÈTRE

On souhaite réaliser un chronomètre comptant les secondes de 0 à 99. Son horloge doit donc être de 1Hz.

**Q1 :** En VHDL, rédigez une description à l'aide d'un compteur pour obtenir une fréquence de 1 Hz (on s'inspirera pour cela de la méthode vue lors de l'exercice précédent). Validez en allumant une LED. Le signal d'alimentation de cette LED doit présenter un rapport cyclique de 0,2, c'est-à-dire la LED est allumée 20% du temps, et qu'elle est éteinte 80% du temps.

**Q2 :** Utilisez un deuxième compteur permettant d'afficher les secondes de 0 à 9 sur un afficheur 7 segments.

**Q3 :** Complétez votre circuit en rajoutant un troisième compteur pour afficher en plus les dizaines sur un second afficheur 7 segments. Les compteurs des unités et des dizaines doivent être synchrones, c'est-à-dire qu'ils doivent avoir la même horloge.

**Q4 :** Perfectionnez votre modèle VHDL pour inclure une remise à zéro à l'aide d'un bouton poussoir.

## EXERCICE 3 : CHENILLARD

Un chenillard est un mouvement lumineux obtenu en allumant et éteignant successivement des LED. Cet effet de mouvement est appelé effet phi, et est très bien illustré par KITT la voiture vedette de la série K2000. On souhaite réaliser un chenillard de la manière suivante :

- Le chenillard est composé de 8 LEDs : on a toujours 3 LED qui sont allumées et 5 LED qui sont éteintes. Lors de l'initialisation, ce sont les 3 LED de droite qui sont allumées.
- Chaque LED allumée se déplace de droite à gauche d'une position à chaque front montant d'horloge.
- Lorsqu'une LED allumée atteint l'extrémité gauche, elle recommence le même trajet en repartant de l'extrémité droite.

**Q :** Réalisez ce chenillard en VHDL avec une horloge de 5 Hz.



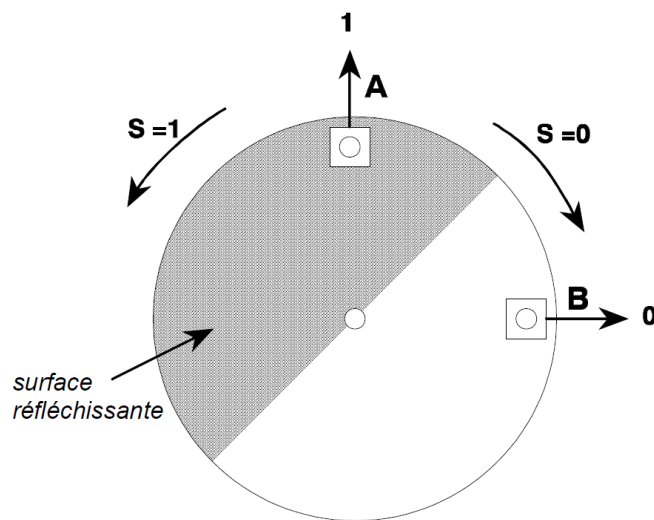
## TP 4 : Automate

### EXERCICE 1 : DÉTERMINATION DU SENS DE ROTATION D'UN ARBRE DE MOTEUR

Deux capteurs optiques A et B sont positionnés en face d'un disque relié à l'axe du moteur. La moitié du disque comporte une surface réfléchissante. Lorsque la surface réfléchissante se trouve en face du capteur, un niveau logique '1' est généré par celui-ci, la moitié du disque non réfléchissante génèrera un niveau logique '0'. Selon le sens de rotation, la séquence du couple de variables AB est différente. Un changement du sens de rotation peut intervenir à tout moment.

**Q1 :** Dessinez le graphe de transition de l'automate délivrant un signal de sens de rotation S (sens horaire :  $S = 0$ , sens trigonométrique :  $S = 1$ ).

**Q2 :** Transcrivez ce graphe en une description VHDL et simuler son fonctionnement ; les capteurs A et B seront simulés par 2 interrupteurs.



### EXERCICE 2 : DÉS ÉLECTRONIQUES POUR JEUX DE RÔLES

Dans de nombreux jeux de rôles, il est nécessaire de disposer de dés de 7 différentes tailles : d4 (valeurs de 1 à 4), d6 (de 1 à 6), d8 (de 1 à 8), d10, d12, d20 et d100 (0 à 99 – 0 correspond à 100).

Nous allons concevoir un circuit électronique qui pourra remplacer tous ces dés.

**Q1 :** Dans sa première version, le circuit ne pourra remplacer que le d4 et le d6. Ce sera un automate de type Moore dont l'horloge fonctionnera à 50 MHz. Le choix du type de dé se fera grâce à une entrée D reliée à l'interrupteur : si  $D=0$ , ce sera un d4 ; si  $D=1$ , ce sera un d6. Une autre entrée L, reliée à l'interrupteur SW[1], sert à lancer le dé : lorsque L vaut 1, le dé tourne (comme un compteur mais très rapidement vu la fréquence de l'horloge) et lorsque L vaut 0, il s'arrête et sa valeur s'affiche sur un afficheur 7 segments. Proposez un diagramme d'états décrivant le comportement de ce circuit et concevez-le ensuite en schématique en utilisant des bascules D (DFF). Le téléverser sur la carte pour le tester.

**Q2 :** Dans sa deuxième version, le circuit pourra remplacer tous les types de dés. L'entrée D passera donc sur 3 bits :

- 000 → le dé ne se lance pas même si L vaut 1,
- 001 → d4,
- 010 → d6,
- 011 → d8,
- 100 → d10,
- 101 → d12,
- 110 → d20,
- 111 → d100.

Concevez ce circuit en VHDL en gérant un compteur pour les unités et un autre pour les dizaines. La valeur maximale de l'ensemble est fixée par l'entrée D. Les unités et les dizaines seront affichées sur deux afficheurs 7 segments.

### EXERCICE 3 : CHENILLARD, LE RETOUR

Nous allons concevoir une nouvelle version du chenillard pour nous rapprocher de celui de KITT, la voiture de la série K2000. Il y a toujours huit LED avec trois LED consécutives allumées et les autres éteintes, mais cette fois, lorsqu'une LED allumée arrive à une extrémité elle repart dans sens inverse. Pour concevoir ce système, il va falloir concevoir une Machine à état finis de type Moore.

**Q1 :** Trouvez le diagramme d'états décrivant le fonctionnement de l'automate commandant les huit LEDs.

**Q2 :** Proposez un modèle VHDL permettant de réaliser cet automate. Utilisez trois processus pour décrire la machine de Moore correspondante :

- Un processus doit calculer, en fonction de l'état courant et des entrées, le numéro de l'état futur qui remplacera l'état courant au prochain coup d'horloge. Il doit avoir un comportement combinatoire.
- Un processus qui met à jour l'état courant en détectant un front montant de l'horloge ; ce processus doit aussi servir à positionner l'état courant dans celui de départ lorsqu'un signal reset a lieu. Ce processus a un comportement séquentiel (la valeur de l'état courant ne change qu'après un front d'horloge et est mémorisé entre deux fronts successifs).
- Un processus qui donne la valeur des sorties en fonction de la valeur de l'état courant ; ce processus a un comportement combinatoire.

**Q3 :** Faire la synthèse de l'architecture de ce circuit et téléversez-la sur la carte, en lui ayant associée au préalable une horloge de fréquence 5 Hz. Vérifiez son bon fonctionnement.

### EXERCICE 4 : MULTIPLIEUR SÉQUENTIEL

Le but de cet exercice est de créer un multiplieur de deux nombres de 4 bits mais en utilisant seulement un additionneur pour faire les calculs. Le calcul se fera donc en plusieurs étapes successives, correspondant à autant de coups d'horloge. Un signal start indique, quand il passe à 1 que les nombres à multiplier sont positionnés sur les entrées du circuit et que le calcul peut commencer. Il reste à 1 tant que le calcul n'est pas terminé et que le résultat n'a

pas été récupéré. Il est nécessaire d'associer à l'additionneur des registres pour mémoriser les résultats intermédiaires.

En plus de l'additionneur, nous aurons besoin de 4 registres qui peuvent faire des décalages à droite de leur valeur : RA, RB, RP et RC. Chaque étape du calcul consistera à calculer un produit partiel en analysant successivement un des bits de A et de l'additionner à la somme des produits partiels déjà pris en compte. Si le bit analysé vaut 0, le produit partiel correspondant vaudra 0 et la somme des produits partiels ne sera donc pas modifiée à cette étape. Si le bit analysé vaut 1, le produit partiel vaut alors B décalé en fonction de la position du bit considéré. Il devra être ajouté à la somme des produits partiels. Lorsque les 4 bits de A ont été pris en compte, la somme des produits partiels représente le résultat de la multiplication.

Voici ci-dessous un exemple de calcul.

<b>Exemple :</b>	$  \begin{array}{r}  0101 \\  \times 1101 \\  \hline  00001 \\  00000 \\  11001 \\  00000 \\  \hline  100001  \end{array}  $	<table style="border-collapse: collapse;"> <tr> <td style="text-align: right; padding-right: 5px;"><b>A</b></td> <td></td> </tr> <tr> <td style="text-align: right; padding-right: 5px;"><b>B</b></td> <td></td> </tr> <tr> <td style="text-align: right; padding-right: 5px;"><math>a_0 =</math></td> <td style="text-align: left;">1</td> </tr> <tr> <td style="text-align: right; padding-right: 5px;"><math>a_1 =</math></td> <td style="text-align: left;">0</td> </tr> <tr> <td style="text-align: right; padding-right: 5px;"><math>a_2 =</math></td> <td style="text-align: left;">1</td> </tr> <tr> <td style="text-align: right; padding-right: 5px;"><math>a_3 =</math></td> <td style="text-align: left;">0</td> </tr> </table>	<b>A</b>		<b>B</b>		$a_0 =$	1	$a_1 =$	0	$a_2 =$	1	$a_3 =$	0
<b>A</b>														
<b>B</b>														
$a_0 =$	1													
$a_1 =$	0													
$a_2 =$	1													
$a_3 =$	0													
<b>5 x 13 = 65</b>														

RP sera initialisé à 0 en début de calcul et recevra la somme des produits partiels en cours de calcul. Sa taille sera de 5 bits pour pouvoir stocker la retenue sortante de l'additionneur. Après la prise en compte de chaque bit de A, ce registre sera décalé d'une colonne vers la droite pour sauvegarder les bits qui n'interviendront plus dans le calcul dans le registre RA. Cela évitera d'avoir à décaler les produits partiels pour les prendre en compte dans le calcul.

RB recevra un des nombres à multiplier au lancement du calcul et ne sera plus modifié jusqu'à la fin.

RA recevra le deuxième nombre à multiplier lors du lancement du calcul. Son bit de poids faible sera analysé à chaque étape de calcul pour déterminer quelle opération devra être réalisée : s'il vaut 0, aucune modification ne sera faite sur RP, s'il vaut 1, RP recevra  $RP + RB$ . Une fois que le bit en cours a été analysé, RA sera décalé d'une colonne vers la droite en même temps que RP.

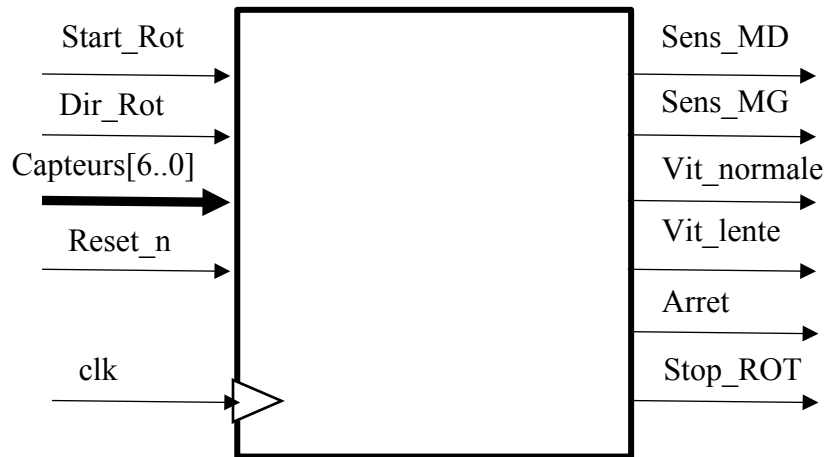
RC sera chargé avec la valeur 8 en début de calcul et sera décalé vers la droite en même temps que RA et RP. Lorsque son bit de poids faible vaudra 1, cela voudra dire que le calcul est terminé.

Le résultat sera contenu partiellement dans RP et dans RA.

Une unité de contrôle (UC), qui aura pour entrées, en plus de l'horloge et du signal reset, les signaux start et RC0 (bit de poids faible de RC), permettra de commander les différents registres en fonction de l'avancement dans le calcul : remise à zéro, chargement des nombres à calculer, mise à jour de la somme des produits partiels, décalages vers la droite.

La figure 1 vous donne le schéma de l'architecture de calcul envisagée.





**Figure 1 : Description des entrées/sorties de l'automate gérant la rotation du robot**

Le robot comporte 7 capteurs à infra-rouge capables de détecter la présence d'une ligne noire en dessous : lorsque la ligne est présente le signal du capteur vaut 1, il vaut 0 sinon. Le nombre de capteurs à 1 en même temps dépend de la largeur de la ligne (dans notre cas 1 ou 2 pourront être allumés en même temps quand le robot se situe au-dessus de la ligne). Lorsque le robot atteint une intersection et pour se diriger, il dispose de deux moteurs commandables indépendamment qui contrôlent les deux roues motrices : moteur gauche (MG) et moteur droit (MD). Chaque moteur peut tourner vers l'avant (**sens=0**) ou vers l'arrière (**sens=1**). Pour la rotation, seules deux vitesses de rotation sont possibles (elles sont prédéfinies dans le bloc de commande): une vitesse normale (**vit\_normale=1**), et une vitesse lente (**vit\_lente=1**). Quand le robot doit s'arrêter, on doit avoir les deux signaux de vitesse à 0 et le signal **arrêt** à 1. Les deux moteurs ont toujours la même vitesse durant la rotation (ou l'arrêt).

Lorsqu'il parcourt le labyrinthe, le robot va rencontrer des intersections de différents types. En fonction de l'intersection, il devra tourner soit vers la droite, soit vers la gauche. Le robot traverse complètement l'intersection avant de tourner. Le signal **Start\_ROT**, quand il passe à 1 indique à l'automate que le robot doit tourner. Le signal **Dir\_Rot** indique s'il doit tourner vers la droite, s'il vaut 0, et vers la gauche, s'il vaut 1.

Le robot doit tourner jusqu'à se retrouver positionné avec une ligne au milieu des capteurs. Au début de la rotation, il tournera à vitesse normale jusqu'à commencer à retrouver la ligne (les capteurs d'une des extrémités de **Capteurs[6..0]** passent à 1). Ensuite, le robot tournera à vitesse lente jusqu'à ce que le robot soit au milieu de la ligne (les capteurs du milieu de **Capteurs[6..0]** sont à 1). Le robot doit alors s'arrêter et signaler au bloc de commande qu'il a fini de tourner en faisant passer **stop\_ROT** à 1.

**Q1** : Dessinez le diagramme d'état décrivant le fonctionnement de l'automate de rotation pour que le robot se comporte comme décrit ci-dessus.

**Q2** : Modéliser cet automate en VHDL puis créer le symbole correspondant et l'insérer dans le schéma principal du projet de parcours de labyrinthe fourni. Téléversez le circuit sur le robot et le tester.