ENSL1 – Exercices VHDL

Exercice 1 (en TD) – Prise en main du logiciel Quartus II, synthèse en VHDL:
Réaliser le composant *xor3* décrit par le schéma suivant (en gardant les noms de signaux) en VHDL et effectuer toute la chaîne de développement.

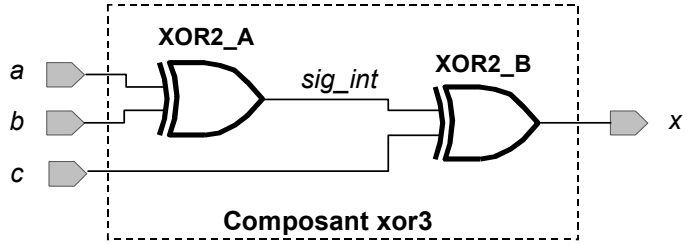


Figure 1. Schéma bloc du composant *xor3*

Structure du fichier

Déclaration de la bibliothèque

Déclaration de l'entité (brochage)

Architecture (structure interne)

```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY xor3 IS
  PORT (
        a : IN STD_LOGIC;
        b : IN STD_LOGIC;
        c : IN STD_LOGIC;
        x : OUT STD_LOGIC
       );
END xor3;

ARCHITECTURE arch_flot OF xor3 IS
  SIGNAL sig_int : STD_LOGIC;

BEGIN
  sig_int <= a XOR b ;
  x       <= c XOR sig_int;
END arch_flot;
```

Déclaration du signal interne

Corps de l'architecture

Travail à faire** (suivant le « Guide d'utilisation du logiciel Quartus II » - GUQII) **:

1. Créer un répertoire **dans votre répertoire de travail sur le réseau**, par exemple *z:\ensl_td*. Créer le répertoire *ex1* dans ce nouveau répertoire.
2. Lancer le logiciel Quartus II (voir § 3.1 du GUQII).
3. Vérifier que vous êtes dans l'environnement MaxPlus + II du logiciel Quartus II (voir § 3.2 du GUQII).
4. Créer le projet *xor3* pour la famille Cyclone II, avec la racine *xor3.vhd* (voir § 3.3 du GUQII) dans le répertoire *ex1*.
5. Créer un nouveau fichier texte (§ 3.6.1, partie 2) et éditer le texte figurant dans le bloc plus haut (module xor3).
6. Sauvegarder le fichier sous le nom *xor3.vhd* dans le répertoire *ex1* (§ 3.6.3, partie 2).
7. Sélectionner la famille Cyclone II de circuits logiques configurables à utiliser (voir § 3.8.1).
8. Appeler le compilateur et compiler le projet (§ 3.8.2).
9. Créer le fichier avec les chronogrammes (§ 3.10.1, partie 2).
10. Sélectionner les signaux d'entrée (*a*, *b*, *c*) et de sortie (*x*) pour la simulation (§ 3.10.2).

11. Modifier la longueur de la simulation (1 us) et la fréquence de la grille (100 ns) (§ 3.10.3).
12. Regrouper les signaux d'entrée (*a*, *b*, *c*) dans un groupe, par exemple *aa* (§ 3.10.5), et initialiser le groupe par une valeur incrémentée (§ 3.10.8).
13. Sauvegarder le fichier de chronogrammes sous le nom *xor3.vwf.*
14. Appeler, puis lancer le simulateur (§ 3.11.1).
15. Vérifier les résultats de la simulation (§ 3.11.2) – noter les résultats de la fonction logique pour différentes valeurs à l'entrée et les retards de la sortie par rapport aux entrées. Comparer les avec le tableau de vérité de la fonction réalisée.

Questions/problèmes :

1. De combien de signaux d'entrées/sorties et de signaux internes dispose le module sur la Figure 1 ?
2. Dessinez le symbole du module développé.

Exercice 2 - Réalisation d'un demi-additionneur d'un bit :
Il s'agit de créer en VHDL une entité représentant un additionneur de deux données d'un bit pour satisfaire le tableau de vérité suivant :

Entrées		Sorties	
a	b	s	rs
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

A noter : Il s'agit d'un demi-additionneur, parce qu'il n'additionne pas la retenue du bit précédent.

Questions/problèmes :

1. Complétez les chronogrammes de sorties *s* (somme) et *rs* (retenue sortante) dans la Figure 2 et confrontez les avec le tableau de vérité du demi-additionneur.
2. En utilisant le résultat de la simulation (temporelle), mesurez le retard du signal *s* à la sortie par rapport aux entrées.

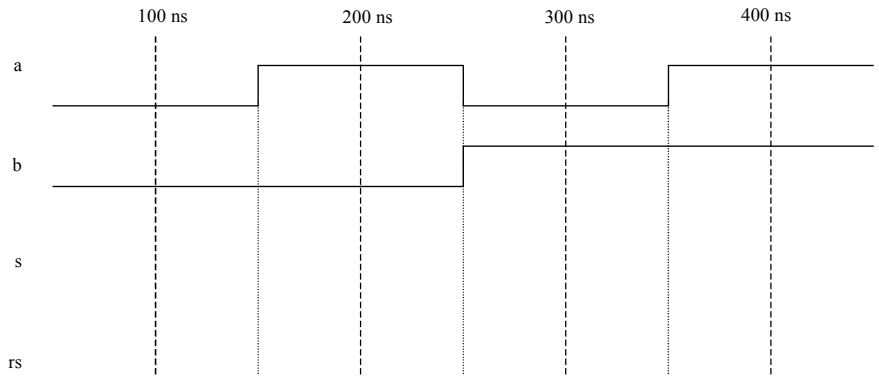


Figure 2. Chronogrammes d'entrées et de sorties (à compléter) du composant *demi_add*

Exercice 3 - Réalisation d'un demi-additionneur d'un bit en utilisant deux composants – une porte logique XOR à deux entrées et une porte logique AND à deux entrées :
Il s'agit de créer une entité représentant un additionneur de deux données d'un bit défini dans l'exercice 2. Cet additionneur doit être décrit en VHDL en utilisant deux composants (décrits dans deux autres fichiers VHDL).

Questions/problèmes :

1. Quel est la différence de comportement entre le module de l'Exercice 2 et 3 ?

Exercice 4 - Réalisation d'un additionneur complet *fa* (FA = Full Adder) sur un bit :
Il s'agit de créer en VHDL une entité représentant un additionneur complet sur un bit pour satisfaire le tableau de vérité suivant :

re	b	a	s	rs
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A noter : Il s'agit d'un additionneur complet, parce qu'il additionne deux données d'entrée *a* et *b* (chacune d'une largeur d'un bit) plus la retenue du bit précédent (*re*). Il faut deux demi-additionneurs pour réaliser un additionneur complet.

Questions/problèmes :

1. Simuler le projet pour les chronogrammes d'entrées présentés dans la Figure 3 et complétez dans cette figure les signaux de sorties obtenus dans la simulation.
2. Proposez le schéma d'un additionneur complet basé sur les demi-additionneurs.

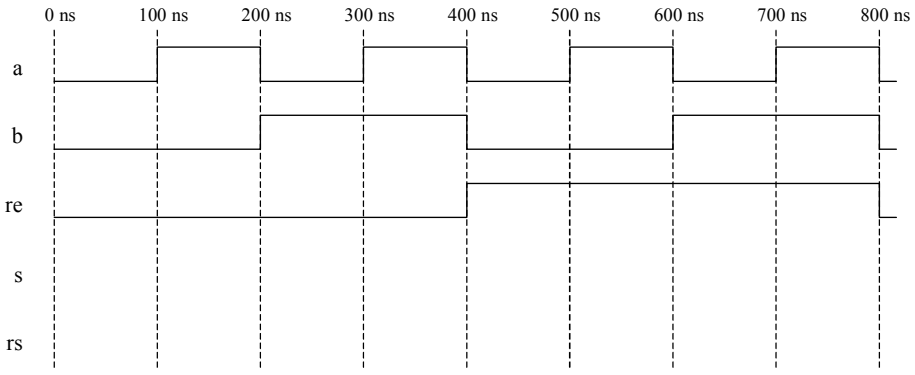


Figure 3. Chronogrammes d'entrées et de sorties (à compléter) du composant *fa*

Exercice 5 (en TD) - Réalisation d'un additionneur sur 4 bits :
Il s'agit de créer en VHDL une entité représentant un additionneur complet sur 4 bits donné par le schéma de la Figure 4 :

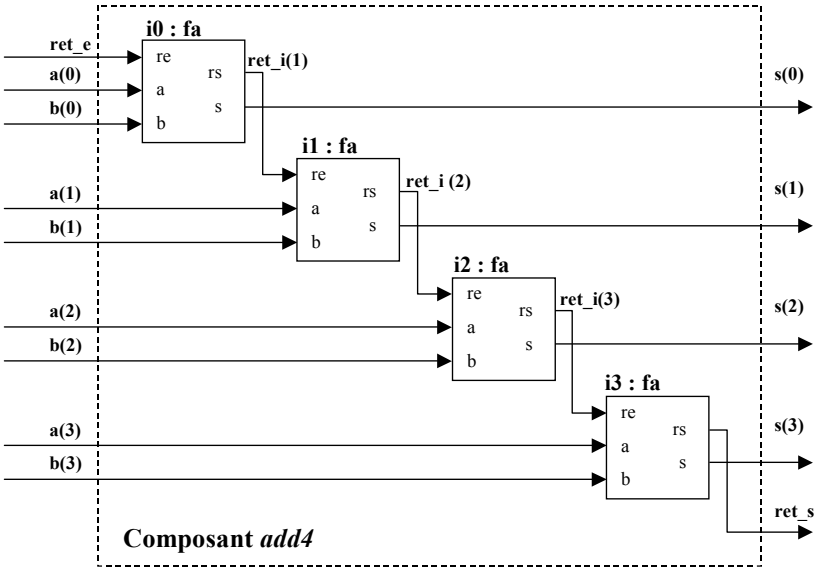


Figure 4. Schéma du module (composant) *add4*

Questions/problèmes :

1. Dessinez le symbole du module *add4*.
2. Combien de niveaux hiérarchiques et combien de types d'entités (types de modules) contient le projet ?
3. Quel est le nombre d'éléments logiques (Logic Elements – LE) utilisé dans le projet (suivant le rapport de compilation) ?
4. Simulez le projet pour les chronogrammes d'entrées suivant la Figure 5, complétez les résultats de la simulation dans cette même figure.

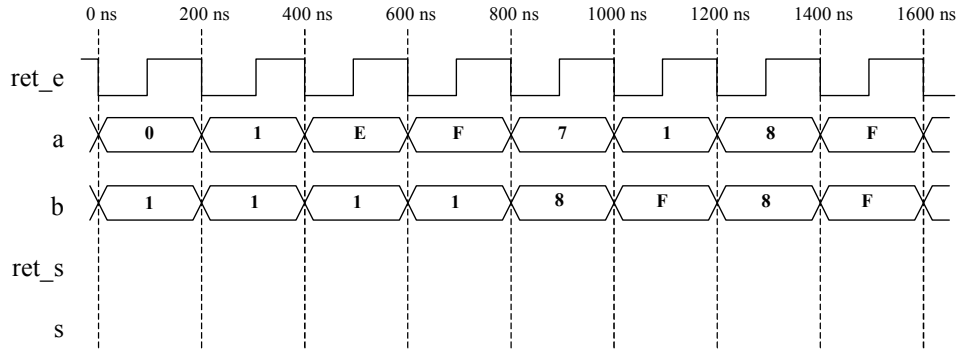


Figure 5. Chronogrammes d'entrées et de sorties (à compléter) du composant *add4*

Exercice 6 - Réalisation d'un additionneur sur 16 bits, basée sur les additionneurs génériques sur 4 bits, en utilisant la structure FOR … GENERATE :

Il s'agit de créer une entité représentant un additionneur sur 16 bits donné par le schéma sur la Figure 6 et donc basé sur l'utilisation de quatre composants *add4*.

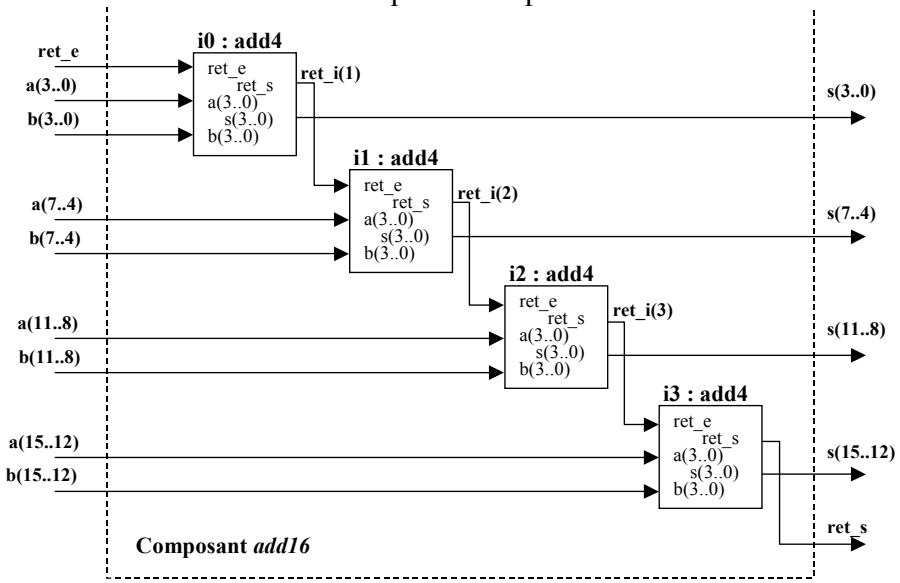


Figure 6. Schéma du composant *add16*

La différence par rapport à l'Exercice 5 (où vous avez dû instancier quatre fois le même composant) est que vous devez maintenant utiliser la structure FOR … GENERATE pour effectuer une instanciation multiple du même composant.

Questions/problèmes :

1. Dessinez le symbole du module *add16.*
2. Combien de niveaux hiérarchiques et combien de types d'entités (types de modules) contient le projet ?
3. Quel est le nombre d'éléments logiques (Logic Elements – LE) utilisé dans le projet (suivant le rapport de compilation) ?
4. Simulez le projet pour les chronogrammes d'entrées suivant la Figure 7, complétez les résultats de la simulation dans cette même figure.

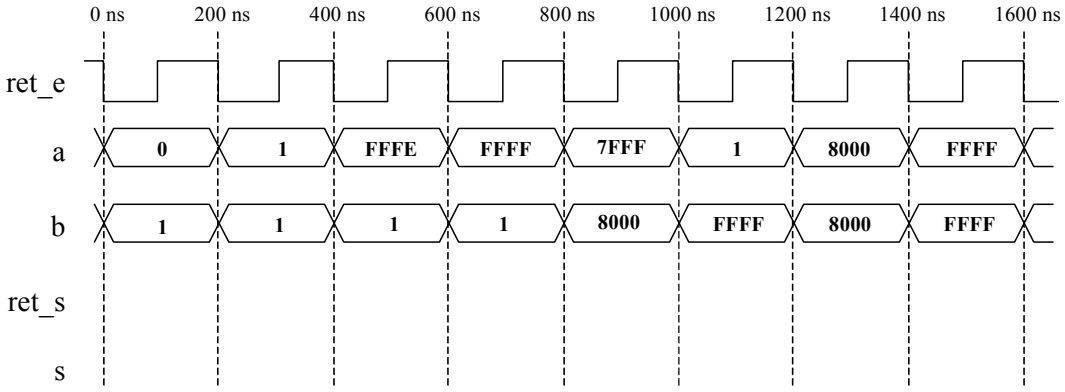


Figure 7. Chronogrammes d'entrées et de sorties (à compléter) du composant *add16*

Exercice 7 (en TD) – Réalisation d'un multiplexeur à deux entrées de 8 bits
Il s'agit de créer l'entité mux2x8 représentant un multiplexeur avec deux entrées de données 8 bits (*a* et *b*) et une entrée d'un bit permettant de sélectionner le canal a ou b (*sel*). Si *sel* = '0', c'est la donnée du canal *a* qui est présente à la sortie, sinon, c'est la donnée du canal *b*.

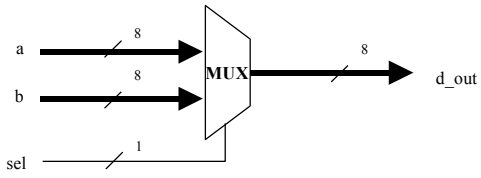


Figure 7. Symbole du module

Questions/problèmes :

1. Faites un tableau précisant le nombre de bits *m* du signal *sel* pour un nombre de canaux *n* allant de 2 à 32.
2. Quel est le nombre minimum de bits *m* du signal *sel* en général pour sélectionner un sur *n* canaux (donnez la formule m = f(n)) ?

Exercice 8 (en TD) – Réalisation d'un codeur « code un sur sept » vers code binaire avec priorité :
Il s'agit de créer l'entité *pri_enc* (Priority Encoder) représentant un codeur de sept bits d'entrées sur un code de trois bits. Le codeur est destiné à coder les demandes d'interruption arrivant de sept sources différentes vers un processeur : La demande *req(1)* étant la plus prioritaire et la demande *req(7)* la moins prioritaire. On ne peut pas exclure l'arrivée simultanée de deux demandes. On aura dans ce cas le numéro de la demande plus prioritaire à la sortie du codeur. Les demandes sont actives à un. S'il n'y a pas de demande d'interruption, le codeur donne 0 à la sortie. Le symbole du composant est présenté sur la figure suivante :

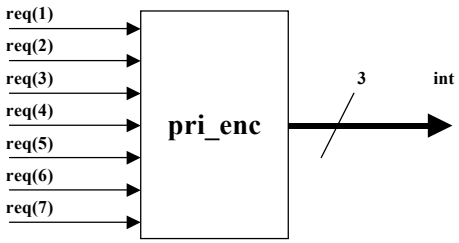


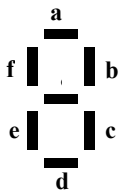
Figure 8. Symbole du module

Questions/problèmes :

1. Combien de combinaisons à l'entrée doit-on tester pour valider le comportement du module ?

Exercice 9 (en TD) – Réalisation d'un décodeur « code binaire sur 4 bits » -> code afficheur sept segments

Il s'agit de créer une entité en VHDL permettant de décoder une valeur binaire sur 4 bits à un code de sept bits pour afficher la valeur numérique présente à l'entrée du module sur un afficheur sept segments. On suppose tout d'abord d'utiliser un afficheur à anode commune. Un segment est donc allumé avec le niveau logique zéro à la sortie du décodeur. Les segments de l'afficheur sont codés de la manière suivante :



Représentation de 16 chiffres hexadécimaux :

Questions/problèmes :

1. Comment peut-on changer facilement la polarité de sorties (pour utiliser un afficheur à cathode commune) ?
2. Proposez une solution permettant d'éteindre toutes les diodes avec une entrée *ena (*si *ena* = '0', les segments sont éteints*)*.

Exercice 10 – Réalisation d'un additionneur/soustracteur 8 bits pour les nombres arithmétiques et logiques avec les drapeaux N, Z, C et V :

Il s'agit de créer une entité représentant un additionneur/soustracteur avec deux entrées de données 8 bits (*a* et *b*) et une entrée d'un bit permettant de sélectionner le type d'opération à effectuer (*op*). Le module donne le résultat *r* sur 8 bits et les drapeaux *N*, *Z*, *C*, et *V*.

Conseils : Utiliser les additionneurs complets de l'Exercice 4, puis réaliser la soustraction *a* – *b* comme addition de l'opérande *a* avec le complément à deux de l'opérande *b*.

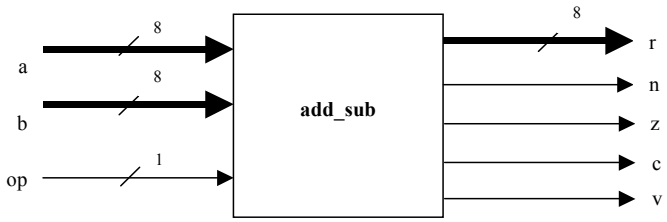


Figure 9. Symbole du module

Tableau de fonctionnement :

op	Fonction
0	Addition
1	soustraction

Exercice 11 (en TD) – Réalisation d'un multiplexeur à quatre entrées de 8 bits avec autorisation de sortie (sortie à trois états)

Il s'agit de réaliser un multiplexeur à quatre entrées de 8 bits en utilisant les modules de l'exercice numéro 7. La sortie de ce multiplexeur sera autorisée par un signal *oe* (Output Enable). Si ce signal est égal à un, la sortie du multiplexeur est autorisée (basse impédance), sinon, elle est en haute impédance ('Z').

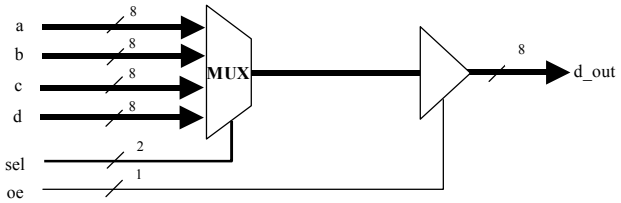


Figure 11. Schéma bloc du dispositif

Exercice 12 (en TD) – Réalisation d'un registre asynchrone de 8 bits (composé de 8 bascules asynchrones - latches)

Réaliser en langage VHDL un registre asynchrone de 8 bits (composé de 8 bascules asynchrones, appelées les D-Latches ou tout simplement les Latches). Le registre doit avoir le comportement suivant : pendant que le signal *ena* à l'entrée de ce registre est actif (égal à '1'), le registre est transparent, donc on obtient à la sortie *q* le signal présent à l'entrée *d* (retardé). Quand le signal *ena* est égal à '0', le registre mémorise la dernière valeur présente à l'entrée avant que le signal *ena* soit descendu à '0'. La figure suivante présente le symbole de ce registre asynchrone et les chronogrammes, qui illustrent son fonctionnement.

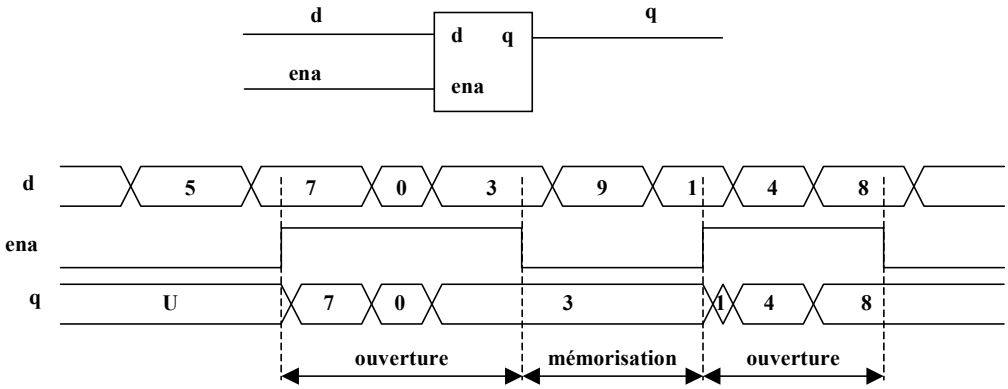


Figure 12. Symbole du module et chronogrammes

Exercice 13 – Réalisation d'un décodeur d'adresse de 8 bits commandé par un signal *ale*

Réaliser le décodeur d'adresse de 8 bits (voir la figure suivante) permettant de valider le signal *hit* à la sortie (le mettre à '1') si pendant l'intervalle où le signal *ale* (Address Latch Enable) était actif, la valeur présente sur le bus partagé par les adresses et les données *a_d* à l'entrée du décodeur (qui correspond à ce moment à une adresse) était supérieurs à $63_{(10)}$ ($0011\ 1111_{(2)}$). Le signal *hit* doit rester valide jusqu'à la prochaine activation du signal *ale*.

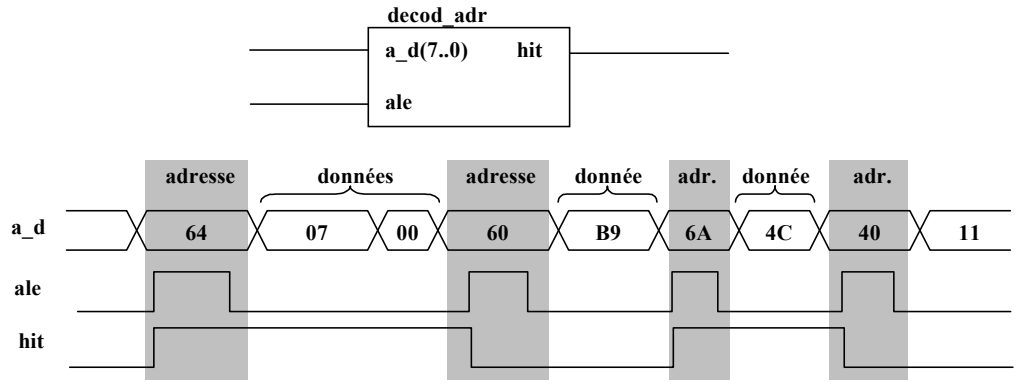


Figure 13. Symbole du module et chronogrammes

Exercice 14 (en TD) – Réalisation d'une bascule D synchrone avec RAZ asynchrone
Réaliser en langage VHDL une bascule D synchrone, sensible sur le front **montant** du signal d'horloge *clk*, avec remise à zéro asynchrone *raz* (active à zéro).

Questions/problèmes :

1. Expliquez, quel est le signal prioritaire : *clk* ou *raz*. Quelle est la structure VHDL à employer ?

Exercice 15 (en TD) – Réalisation d'une bascule T (synchrone) en VHDL
Réaliser en langage VHDL une bascule T (toggle) synchrone, sensible sur le front **montant** du signal d'horloge *clk*.

Questions/problèmes :

1. Quelle est l'utilisation pratique de la bascule T ?
2. Proposez le montage pour réaliser la bascule T en utilisant la bascule D.

Exercice 16 (en TD) – Réalisation d'une bascule JK (synchrone) en VHDL
Réaliser en langage VHDL une bascule JK synchrone, sensible sur le front **descendant** du signal d'horloge *clk*. Attention, aucune entrée (J ou K) ne doit être prioritaire.

Questions/problèmes :

1. Quelle est l'utilisation pratique de la bascule JK ?
2. Proposez le montage pour réaliser la bascule T en utilisant la bascule JK.

Exercice 17 (en TD) – Réalisation d'un circuit anti-rebonds
Les commutateurs et les interrupteurs mécaniques génèrent juste avant et après la commutation des petites perturbations dues aux rebonds mécaniques de l'interrupteur. Ces perturbations ont une fréquence d'environ un kilohertz. Si un tel signal est utilisé comme un signal d'horloge, il faut supprimer ces rebonds. Le principe consiste à échantillonner le signal à la sortie de l'interrupteur par une bascule D à une fréquence inférieure à un kilohertz (par

exemple 100 Hz) et comparer les deux valeurs : la valeur originale et la valeur échantillonnée (et donc retardée). Si les deux valeurs sont égales à un, il faut enregistrer dans la deuxième bascule à la sortie du circuit la valeur logique un, si elles sont égales à zéro, il faut enregistrer zéro et si les valeurs sont différentes, il faut garder la valeur précédente du registre (pour éviter de copier des rebonds à la sortie du dispositif). Nous disposons d'un signal « *bouton* » (avec des rebonds) et le signal d'horloge clk100 de 100 Hz pour générer le signal *bouton_sans_rebonds*.

Questions/problèmes :

1. Quel sera le retard du signal généré (*bouton_sans_rebonds*) par rapport au signal original présent à la sortie de l'interrupteur (*bouton*) ?

Exercice 18 (en TD) – Réalisation d'un registre synchrone de 8 bits avec la remise à zéro
Réaliser en langage VHDL un registre synchrone de 8 bits, sensible sur le front **montant** du signal d'horloge *clk*. L'écriture dans le registre est autorisée si le signal *ena* est égal à un. Le signal *raz* doit être actif à un.

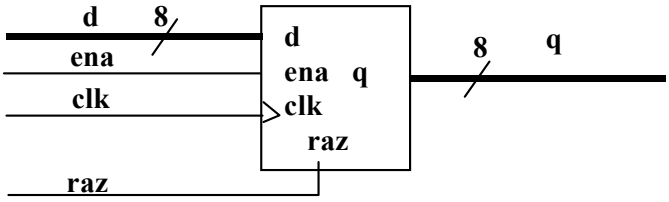


Figure 14. Symbole du module

Questions/problèmes :

1. Quelle est l'utilisation pratique des registres ?

Exercice 19 – Réalisation d'une mémoire – un jeu de 4 registres synchrones de 8 bits
Réaliser en langage VHDL une mémoire composée de 4 registres synchrones de 8 bits, sensible sur le front **montant** du signal d'horloge *clk*. Ecriture dans la mémoire est autorisée avec le niveau zéro du signal *n_wr*.

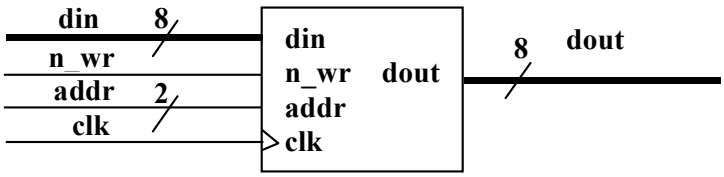


Figure 15. Symbole du module

Questions/problèmes :

1. Quelle est l'utilisation pratique de la mémoire ?

Exercice 20 (en TD) – Réalisation d'un diviseur de fréquence par deux
Il s'agit de créer une entité permettant de diviser la fréquence d'horloge *clk* par deux. Le schéma de cette unité basée sur une bascule D est présenté sur la Figure 16.

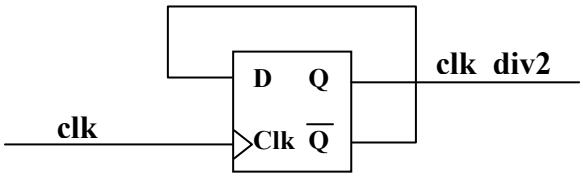


Figure 16. Schéma à réaliser

Exercice 21 – Réalisation d'un compteur asynchrone basé sur les bascules D
Réaliser le compteur asynchrone de 4 bits basé sur les bascules D. Le compteur est présenté sur la Figure 17.

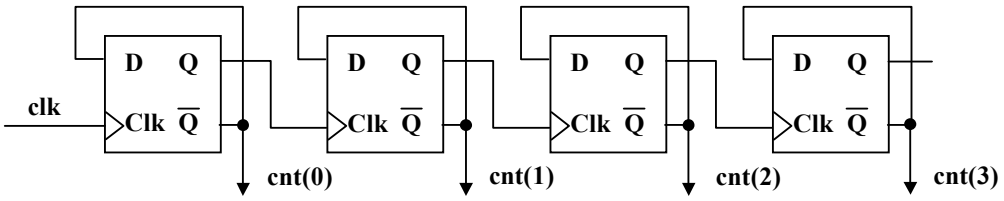


Figure 17. Schéma à réaliser

Questions/problèmes :

1. Notez bien le temps de basculement de différentes bascules. Quelle est la conséquence de ce comportement ?

Exercice 22 (en TD) – Réalisation d'un diviseur de fréquence par dix
Il s'agit de créer une entité permettant de diviser la fréquence d'horloge *clk* par dix. Le diviseur doit être basé sur un compteur synchrone : chaque fois que le compteur passera à cinq, il sera remis à zéro et il fera basculer la bascule D commandée par ce compteur. La sortie de cette bascule *clk_div10* sera utilisée comme la sortie du module.

Questions/problèmes :

1. Quel est le rapport cyclique du signal d'horloge généré ?

Exercice 23 (en TD) – Réalisation d'un compteur bi-directionnel synchrone et préchargeable sur 8 bits
Réaliser le compteur synchrone de 8 bits qui peut être préchargé (initialisé) par une valeur qui se trouve à l'entrée *init* du compteur au moment où l'entrée *load* est égale à '1'. L'entrée *dir* permet de changer la direction du comptage (*dir* = '1' pour l'incrémentation et dir = '0' pour la décrémentation). Le comptage est autorisé par le signal *ena* (si *ena* = '1', le compteur compte, sinon il s'arrête et garde la valeur actuelle à sa sortie). Les entrées *load*, *dir* et *ena* sont testées seulement pendant le front montant du signal d'horloge *clk* (entrées synchrones).

De plus, le compteur peut être remis à zéro d'une manière asynchrone par un signal *rst* actif à zéro. Le symbole du compteur est présenté sur la Figure 18.

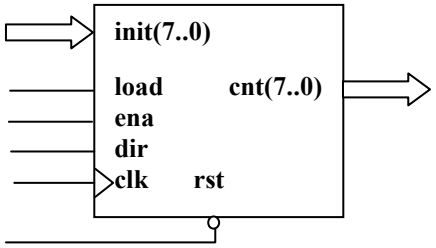


Figure 18. Symbole du module

Questions/problèmes :

1. Avant proposer le code VHDL, définissez la priorité de tous les signaux de contrôle (*load*, *ena*, *dir*, *clk* et *rst*).

Exercice 24 (en TD) – Réalisation d'un séquenceur pour animer 4 diodes LED

Il s'agit de créer une entité permettant d'animer quatre diodes LED D1 à D4. A chaque instant donné, au plus une diode peut être allumée et elle restera allumée pendant une période d'horloge. Puis c'est la diode suivante qui doit s'allumer. Il y a deux modes de fonctionnement (deux suites possibles) sélectionnés par l'intermédiaire du signal d'entrée *sel* :

a) sens horaire (si *sel* = 0),
b) contre sens (si *sel* = 1).

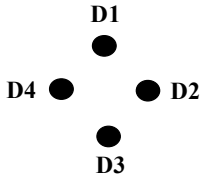


Figure 19. Disposition de quatre diodes à faire animer

Exemple de la suite en mode a) : **D1 → D2 → D3 → D4 → D1 → D2 → …**

Exemple de la suite en mode b) : **D1 → D4 → D3 → D2 → D1 → D4 → …**

Le module a une entrée d'horloge (*clk*), une entrée de sélection du mode (*sel*) et un signal de remise à zéro asynchrone actif à zéro (*reset*). Après la remise à zéro l'afficheur se trouve dans un état initial (aucune diode allumée), puis une des deux suites commence suivant la valeur d'entrée *sel*, en partant de la diode D1. Si on changeait la direction pendant le fonctionnement de l'afficheur, le contrôleur doit passer directement de la suite a) à la suite b) (sans passer par l'état initial comme c'était le cas après la remise à zéro du contrôleur). Le module a quatre sorties connectées avec les cathodes des diodes (les diodes s'allument donc avec le niveau logique zéro).

Questions/problèmes :

1. Quel est le type de machine d'état que vous avez employé ?

Exercice 25 (en TD) – Commande d'un ascenseur dans un immeuble à deux étages (plus RDC)

Il s'agit de créer l'entité *commande* permettant de commander (d'une manière simplifiée) le moteur d'un ascenseur avec trois boutons d'appel *call* actif à l'état bas (un bouton par étage). L'entité *commande* doit pouvoir mettre en marche le moteur d'ascenseur (par la sortie *cmnd_go*) et changer sa direction (par la sortie *cmnd_dir*). Par exemple, *call*(0) = 0 signifie un appel au RDC, *call*(1) = 0 un appel au premier étage. L'appel d'un étage plus haut est prioritaire (au cas où il a deux appels simultanés). On peut remettre à zéro la commande de l'ascenseur avec un bouton *reset* qui est présent dans l'armoire de commande (inaccessible aux utilisateurs normaux). Deux diodes LED *led_up* (pour la montée) et *led_down* (pour la descente) à chaque étage permettent de visualiser, si l'ascenseur est en train de monter (*led_up* = 0), de descendre (*led_down* = 0) ou s'il est arrêté (*led_up* = *led_down* = 1) ou bloqué dans l'état « arrêt de secours » (*led_up* = *led_down* = 0). Les diodes *led_up* pour chaque étage sont connectées en parallèle. De même pour les diodes *led_down* (le module a donc seulement une sortie *led_up* et une sortie *led_down*). Le signal *cmnd_go* = 1 met l'ascenseur en marche (et allume une des diodes *led_up* ou *led_down*). Le signal *cmnd_dir* définie la direction du mouvement (*cmnd_dir* = 1 pour la montée, *cmnd_dir* = 0 pour la descente).

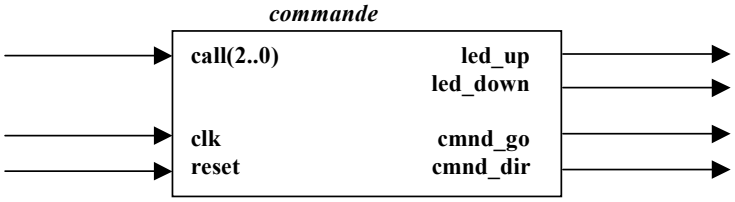


Figure 20. Symbole du contrôleur d'ascenseur

Questions/problèmes :

1. Qu'est qu'il faut ajouter pour sécuriser le fonctionnement du système ?

Exercice 26 – Gestion d'initialisation d'une mémoire asynchrone

Il s'agit de créer l'entité *init_mem* permettant d'initialiser une mémoire de 256 octets par des valeurs croissantes (0 à 255). L'entité *init_mem* doit générer les adresses *addr* (8 bits), les données *data* envoyées sur un bus de données bidirectionnel (8 bits), le signal de sélection mémoire nCS (actif à zéro) et le signal d'écriture dans la mémoire (nWR). Suite au déclenchement de la procédure par un signal d'entrée *start* (actif à zéro), le contrôleur doit utiliser un signal d'horloge *clk* pour générer 256-fois 6 phases différentes du protocole :

1. attente
2. préparation d'adresse,
3. préparation de données,
4. transmission de données,
5. fin transmission de données,
6. fin d'accès mémoire.

Signification de ces différentes phases est illustrée sur la Figure 21.

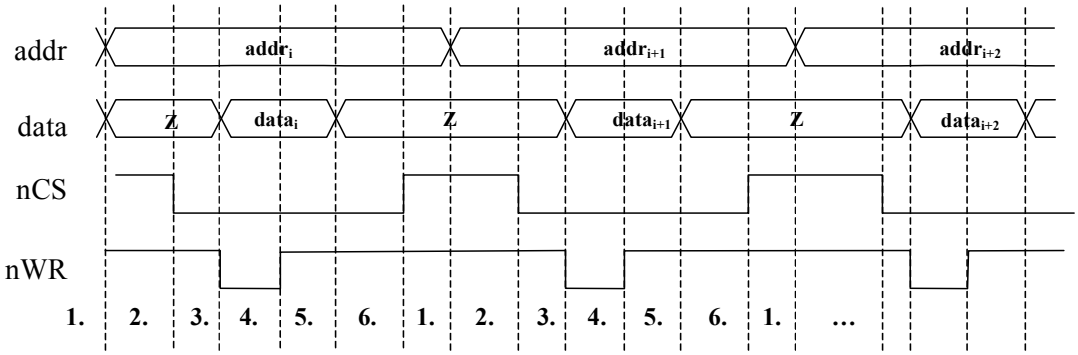


Figure 21. Protocole gestion d'initialisation de la mémoire

A noter : Les sorties data doivent être en haute impédance pendant les phase 1., 2, 3 et 6. L'écriture de données s'effectue à la fin de la phase 4. Le compteur d'adresses est incrémenté au début de la phase 2. Les données peuvent avoir la même valeur (incrémentée) que l'adresse ($data_i = addr_i$).

Questions/problèmes :

1. Dessinez le schéma bloc de l'application (compteur données/adresse + machine d'états).
2. Dessinez le diagramme états/transitions pour la machine d'états qui s'appellera *sm_ctrl*.
3. Pourquoi la phase 3. doit s'achever AVANT que l'on mette les sorties *data* en haute impédance ?

Exercice 27 (en TD) – Gestion d'un compteur animé

Il s'agit de créer l'entité *anim_cnt* permettant de gérer un compteur bi-directionnel de quatre bits. Le compteur doit se trouver dans un des quatre états successifs :

1. comptage (incrémentation) de 0 à 9,
2. décomptage (décrémentation) de 9 à 5,
3. incrémentation de 5 à 7,
4. décrémentation de 7 à 0.

L'état 4 sera ensuite suivi par l'état 1 etc.

Le module *anim_cnt* aura deux entrées (clk et reset) et quatre sorties (cnt(0) à cnt(3)) – les sorties du compteur.

Questions :

1. Dessinez le schéma bloc de l'application (compteur + machine d'états).
2. Dessinez le diagramme états/transitions pour la machine d'états qui s'appellera *sm_ctrl*.
3. Qu'est qu'il faut faire pour réaliser le compteur et la machine d'état dans deux entités différentes ?