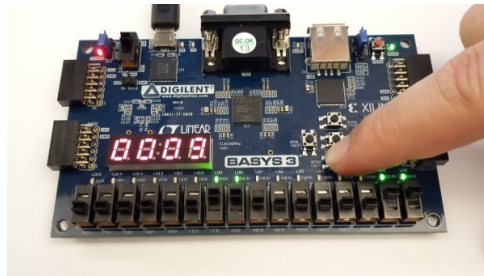


# Travaux Pratiques d'Électronique Numérique 1A-1B

Version 16 du 30/09/2019



(Séances 1 à 4)

---

## Chronomètre

---

Année 2019/2020  
Semestre 5

But : aborder les aspects combinatoires, séquentiels et technologiques de l'électronique numérique à travers la réalisation d'un chronomètre.

**Vous devez avoir lu les 19 premières pages de ce document avant de venir au premier TP et agi en conséquence.**

**Plusieurs préparations sont systématiquement demandées au début de chaque séance :**

- Séance 1 : préparations 1 et 2 (donc de Prep1A à Prep2C).
- Séance 2 : préparations 3, 4 et 5 (Prep3A à Prep5A).
- Séance 3 : préparations 6, 7 et 8 (Prep6A à Prep8A)
- Séance 4 : préparations 9, 10 et 11 (Prep9A à fin).

Il est important pour vous de connaître ce qui va être traité pendant les séances de travaux pratiques, avant de commencer le TP : cela va vous permettre d'exploiter au mieux chaque séance de TP. C'est la raison pour laquelle, on vous demande d'effectuer les préparations. D'ailleurs, nous pourrions demander aux élèves n'ayant pas abordés les préparations requises d'aller terminer ailleurs ces préparations avant d'être admis en TP.

## Un compte rendu devra être rendu à la fin de la dernière séance.

Vous devez fournir dans le compte rendu :

- Les schémas réalisés.
- Les descriptions réalisées en langage VHDL.
- Les Chronogrammes de test (à chaque fois que l'on vous demande un test) avec des commentaires sur ces chronogrammes afin de montrer clairement que votre réalisation fonctionne ou quels sont les dysfonctionnements.
- Les comptes rendus de tests sur maquette.
- Inutile de rendre un dossier paraphrasant ce qui est dans le document que vous avez entre les mains. Il ne s'agit pas de rendre un rapport complet sur le chronomètre mais une trace de votre travail accompagné de commentaires pertinents.
- Les analyses.

## 1 Précisions sur les objectifs visés et comment y parvenir


### **1.1 Les moyens**

Vous avez à votre disposition pour ce TP un PC et une carte Basys3. La carte Basys3 est programmable à l'aide de l'**outil de développement de circuits numériques « VIVADO »** de la société « Xilinx ». On vous recommande fortement de télécharger le logiciel VIVADO sur le site de Xilinx pour l'utiliser sur votre ordinateur personnel et ainsi continuer votre travail en dehors des séances. Attention car ce téléchargement suivi de l'installation est très long car le logiciel est « lourd » : plusieurs Giga. Dans les meilleures conditions, cela prendra environs 1 heure. Il est à noter que toutes les salles sont équipées de ce logiciel dans l'école.

Vous aurez aussi à votre disposition une carte contenant un composant programmable par une sonde de téléchargement reliée au PC. La **carte** utilisée s'appelle « **Basys 3** » et a été développée par la société Digilent. Cette carte possède tous les périphériques utiles à notre réalisation : boutons, LED et afficheurs. La carte « Basys3 » est empruntable au centre doc comme n'importe quel livre.

### **1.2 Une démarche systématique de développement**

La démarche que l'on vous demande de suivre pas à pas, nécessite :

- 
1. De concevoir en VHDL un composant logique.
  2. De simuler le composant logique réalisée pour la tester.
  3. D'implémenter, lorsque demandé, sur la carte Basys 3 le résultat obtenu.
  4. Parfois d'analyser le résultat d'implémentation.
  5. De transformer en un nouveau composant la fonction logique ainsi validée afin de la réutiliser dans la suite.

Le TP se décompose en 4 séances. Ces 4 séances vous conduiront dans la réalisation d'un chronomètre déclinable en plusieurs versions. La réalisation se déroule sous le logiciel VIVADO de Xilinx : vous créerez un unique projet qui vous servira tout au long de ces 4 séances. Nous vous invitons fortement à télécharger la dernière version de VIVADO sur votre ordinateur personnel (la version « Lab Edition » est autonome) afin de poursuivre le travail en dehors des séances.

### **1.3 Les objectifs**

Le but (pragmatique) est de faire fonctionner un chronomètre sur les maquettes mises à votre disposition en salle de travaux pratiques. Par manque de temps aucun câblage ne vous

sera demandé : il s'agit de donner vie à la maquette existante. Cependant une analyse des composants présents sur la carte de développement sera demandée. Le but visé vous permettra d'acquérir un certain nombre de connaissances et de savoir-faire :

- Dans le domaine de la logique **combinatoire** : par la synthèse élémentaire ou par assemblages de composants simples ou complexes.
- Dans le domaine de la logique **séquentielle** : synthèse de compteurs, association de compteurs, élaboration de circuits séquentiels.
- Dans le domaine **technologique** : connaissance d'un composant reconfigurable d'électronique numérique FPGA, utilisation d'un logiciel de développement pour composants numériques reconfigurables FPGA, langage de description matériel VHDL, architecture d'un composant reconfigurable...

## 1.4 La décomposition « fonctionnelle » en mode synchrone

En électronique numérique, nous décrivons des « composants ». Ce sont des modules, souvent symbolisés par des rectangles qui possèdent des entrées et des sorties (numériques). Ces composants sont aussi appelés « blocs », « modules ». Le projet final est un module principal dont les entrées et les sorties correspondent à des broches (pin) du composant réel sur lesquels sont connectés divers composants (LED, segments d'afficheurs, interrupteurs, horloge...). Chaque module peut être le résultat de l'association de plusieurs « modules » qui peuvent être eux-mêmes décomposés chacun sous forme de plusieurs modules, on parle de **hierarchie**... Dans notre cas, le chronomètre, qui est le module principal, va être décomposé sous forme de modules dont les fonctionnalités sont les suivantes:

- Comptage du temps (par des compteurs).
- Système séquentiel pour gérer les interactions Homme-Machine et « donner vie » au chronomètre.
- Visualisation des données (sur afficheurs 7 segments et éventuellement LED).

Vous remarquerez que les blocs sont synchronisés par la même horloge « clk ». C'est le principe d'une réalisation synchrone. Cette méthode est imposée et d'ailleurs les composants programmables sont conçus pour être développés en **mode synchrone** (tiens, quel hasard, on privilégie aussi cette méthode en cours) (ne serait-ce pas un complot ?).

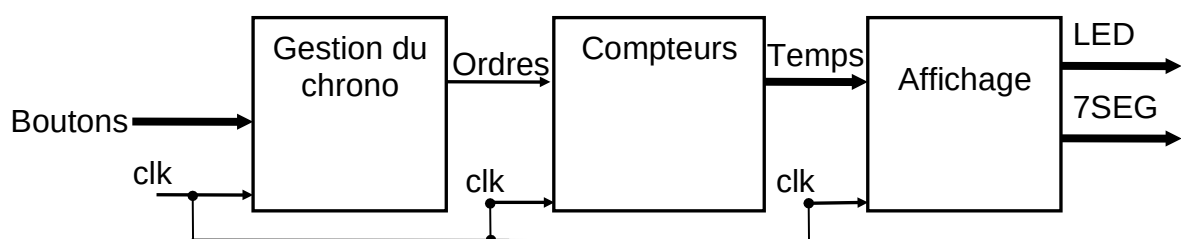


Figure 1 : schéma du principe global du chronomètre

## 1.5 Les séances et les préparations demandées

A titre indicatif, les objectifs des séances sont les suivantes :

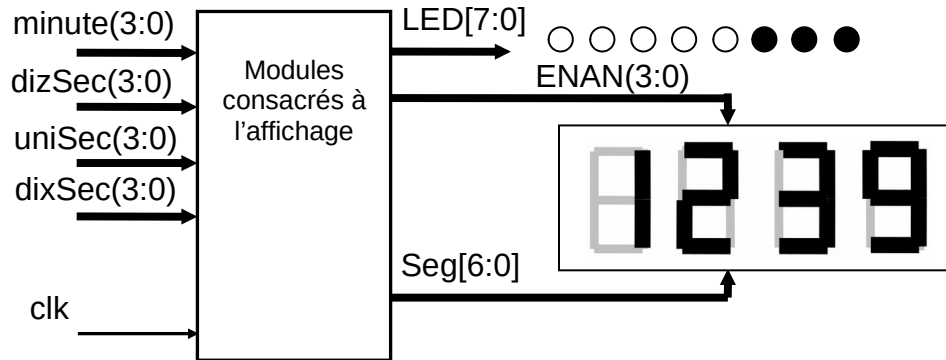
- Séance 1 : réalisation des décodeurs d'affichage.
- Séance 2 : réalisation des compteurs pour l'affichage dynamique.
- Séance 3 : fin de réalisation des compteurs du temps.
- Séance 4 : implémentation d'un système synchrone permettant le fonctionnement du chronomètre.

Vous devez arriver à chaque séance avec les préparations demandées. Ne confondez pas la numérotation des travaux : « travail 1 », « travail 2 » avec la numérotation des séances ! En

effet une séance se compose de plusieurs travaux ! Chaque travail est décomposé en plusieurs items : une explication, une préparation, une réalisation, un test ou une simulation et une analyse.

## 2 Commande de l'affichage

### 2.1 Schéma général



On veut construire le bloc complet qui va permettre d'effectuer la commande de l'affichage. Nous utilisons des cartes Basys 3 qui possèdent des interfaces de visualisation : afficheurs 7 segments et des LED.

### 2.2 Les entrées du bloc d'affichage

Les informations fournies à ce bloc pour effectuer l'affichage sont les suivantes :

- **clk** : signal d'horloge qui va permettre de synchroniser notre réalisation. Il provient d'un oscillateur 100 MHz connecté sur la broche W5 du FPGA Artix 7 de la carte Basys 3. Il n'est pas utile pour les blocs complètement combinatoires mais est nécessaire pour la mise en œuvre d'un séquençage des afficheurs.
- **Minute(3:0)** : ce bus code les minutes qui seront affichées sur l'afficheur le plus à gauche.
- **DizSec(3:0)** : ce bus contient le codage binaire des dizaines de secondes à afficher sur l'afficheur du milieu gauche.
- **UniSec(3:0)** : ce bus contient le codage binaire des unités de secondes à afficher sur l'afficheur du milieu droit.
- **DixSec(3:0)** : ce bus contient le codage binaire des dixièmes de secondes à afficher sur l'afficheur de droite.

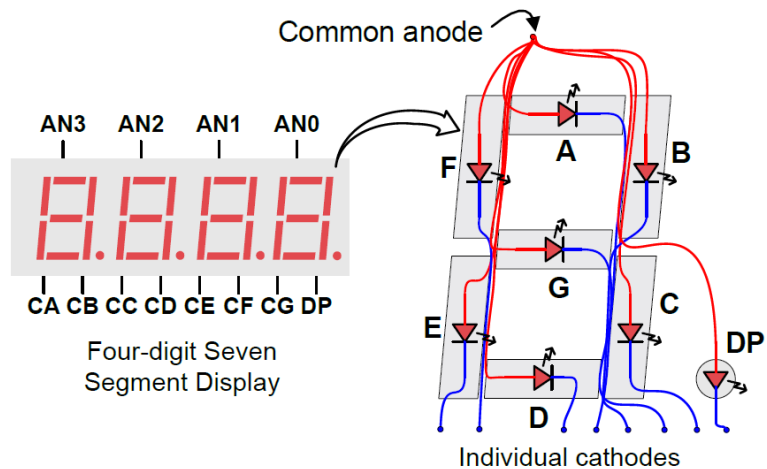
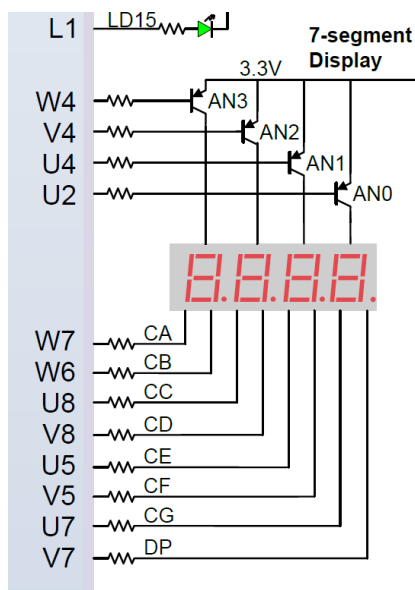
## 2.3 Le matériel

### 2.3.1 Les afficheurs 7 segments commandés

Ce qui nous intéresse le plus pour le chronomètre est l'affichage de valeurs. On va donc commander les quatre afficheurs 7 segments pour obtenir l'affichage des valeurs désirées.

Il y a 4 signaux d'anodes: AN0 à AN3. Chaque signal d'anode correspond à un et un seul afficheur.

Si l'on considère l'afficheur  $i$  ( $i$  de 0 à 3), il possède 7 LED dont les anodes sont toutes connectées au même signal AN $i$ . Ce signal AN $i$  est commun aux 7 LED de l'afficheur  $i$ ; c'est pourquoi l'on parle d'anode commune (common anode).



Les afficheurs sont disposés l'un à côté de l'autre mais les signaux envoyés aux cathodes de chaque segment sont identiques pour les 4 afficheurs (Seg (6:0) ou CA à CG): ils affichent donc la même combinaison de 7 segments.

On peut sélectionner ou désélectionner chaque afficheur par les signaux AN0 à AN3. Vous noterez qu'il n'est pas possible à un instant fixé d'obtenir « 1239 » sur les afficheurs. En effet, si l'on sélectionne l'affichage de 4 chiffres par les anodes, les signaux de cathode sont identiques pour les 4 afficheurs, les chiffres affichés sont donc identiques. Comment faire alors pour tout de même percevoir « 1239 » ? Pour cela nous allons profiter d'une imperfection du système visuel humain qui s'appelle la persistance rétinienne: une image perçue par notre

système visuel, nous reste visible quelques dizaines de ms après sa disparition. Nous allons successivement, cycliquement et rapidement afficher : « 1\_\_ », « \_2\_ », « \_\_3\_ », puis « \_\_\_9 » ... Nous percevrons alors « 1239 ». Le tour est joué.

Vous remarquerez peut-être au passage que sur le schéma global le signal des anodes est nommé ENAN( $i$ ) alors qu'il s'appelle AN $i$  sur le schéma de la carte constructeur. Si les noms sont différents, c'est bien que les signaux sont différents. En effet ENAN( $i$ ) est un signal envoyé à la base d'un transistor de type PNP alors que AN $i$  est le signal récupéré sur son collecteur. D'ailleurs cela amène la première question.

#### Question 0

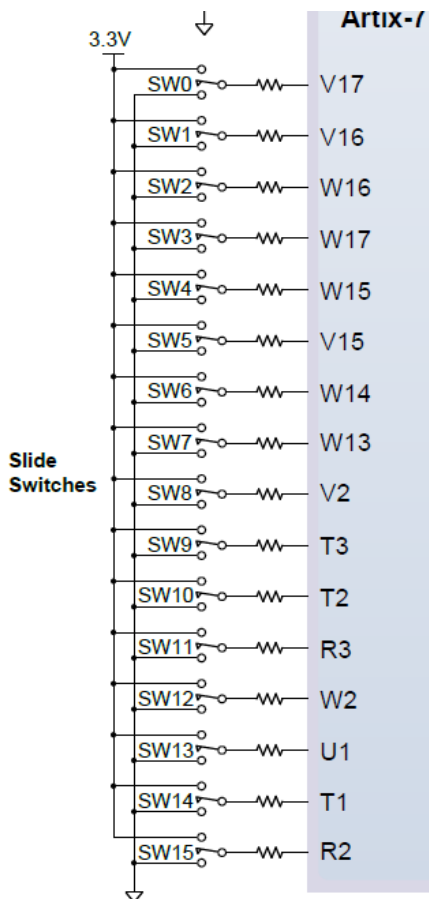
Quel niveau logique de ENAN( $i$ ) amène la sélection de l'afficheur  $i$  ?

- DizSec(3:0) : ce bus contient le codage binaire des dizaines de secondes à afficher sur l'afficheur du milieu gauche.

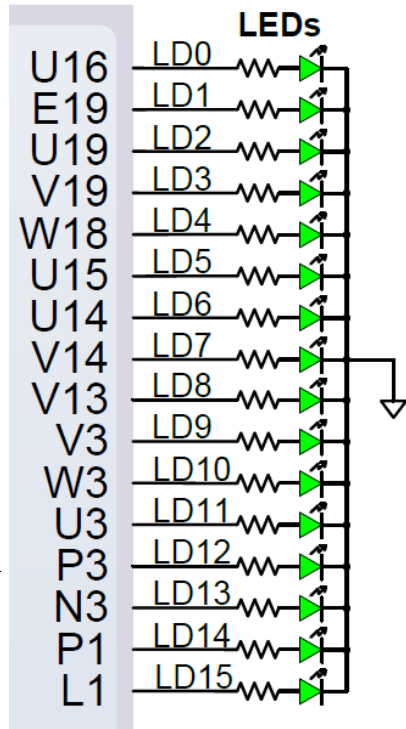
- UniSec(3:0) : ce bus contient le codage binaire des unités de secondes à afficher sur l'afficheur du milieu droit.
- DixSec(3:0) : ce bus contient le codage binaire des dixièmes de secondes à afficher sur l'afficheur de droite.

## 2.3.2 Il y a aussi des LED

Nous allons commencer par bâtir le décodage global pour un afficheur 7 segments mais on a aussi la possibilité de commander en affichage 16 LED. L'utilisation des LED est facultative.



Le schéma de leur connexion à l'Artix 7 sur la carte Basys 3 est donné ci-contre.



## 2.3.3 On peut même utiliser des interrupteurs glissants

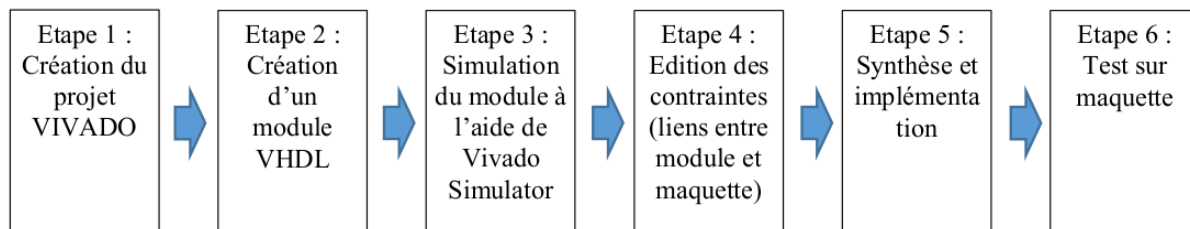
Il est possible d'utiliser les nombreux switches glissants (Slide Switches) pour effectuer des tests de l'affichage.

La documentation de la carte Basys 3 nous fournit les indications ci-contre.

## 2.4 La maîtrise du logiciel VIVADO

### 2.4.1 Le flot de développement d'un projet

Le logiciel VIVADO est l'outil indispensable pour développer les circuits numériques reconfigurables de la société Xilinx. Les étapes d'un développement sont les suivantes :



La première étape ne sera pas répétée dans le cadre de notre développement car nous conserverons le projet créé auquel nous viendrons ajouter de nouveaux modules. Les autres étapes, de 2 à 6 seront répétées pour chaque nouveau module créé. Rappelons qu'un module peut s'appuyer sur les modules précédemment insérés dans le projet.

## 2.4.2 Création de votre projet

Vous allez maintenant aborder la première étape et donc vous allez créer votre projet. Quelques conseils :

- Ne sauvegardez pas votre projet sur l'emplacement donné par défaut par « VIVADO ». En effet cet emplacement se situe sur le disque dur qui peut être à tout moment reformaté ou effacé. Vous êtes prévenus !
- Sauvez votre projet dans l'emplacement qui vous est réservé sous le lecteur réseau de votre compte LDAP (renseignez-vous car au moment où sont écrits ces lignes, nous ne connaissons pas sur quel espace de travail vous serez invités à travailler). Créez un répertoire au nom explicite et **sans espaces** (« TPNUM » par exemple) dans lequel vous allez sauvegarder votre projet afin que vos fichiers ne se trouvent pas tous dans la racine (vous réutiliserez ce même compte pour les TP de microprocesseur mais aussi les mini-projets). Il est à noter que l'arborescence conduisant à votre projet doit aussi ne pas contenir de noms de répertoire avec des espaces.
- Ne sauvegardez pas votre travail sur le bureau, car le nom contient des espaces et Vivado ne le supporte pas au moment de l'implémentation!
- Gardez le même projet pour les 4 séances.
- Remarque importante: le composant utilisé est le « **Artix-7 (XC7A35T) CPG236 -1 C** » dans la catégorie « General Purpose ».
- A ne surtout pas faire : nommer un module VHDL du même nom que celui du projet.
- A ne surtout pas faire : nommer avec un même nom un module VHDL et un fichier de simulation.

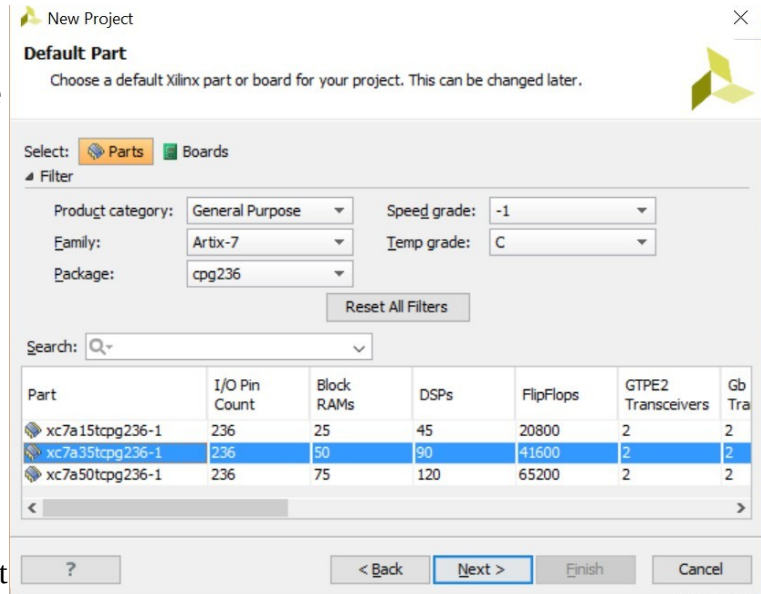
Les étapes à suivre :

1. Lancez « Vivado 2016.2 » ou la dernière version installée.
2. Un peu de patience. Vous verrez ! Il en faut à certaines étapes car le logiciel est gourmand en ressources.
3. Dans la fenêtre d'accueil du logiciel, cliquez sur « Create New Project ».
4. Une fenêtre « New project » s'ouvre alors. Cliquez sur « Next ».
5. Dans la fenêtre suivante « Project Name », vous devez spécifier le nom (« Project Name ») de votre projet ainsi que sa localisation (« Project Location ») (relire les conseils précédents). Laissez « Create project subdirectory » en position cochée. Passez à la fenêtre suivante en cliquant sur « Next ».
6. Remarquez au passage que vous pourrez revenir sur une page précédente en cliquant sur « Back ».
7. Il s'agit maintenant de la fenêtre « Project Type ». Sélectionnez « RTL Project » sans demander l'ajout de sources : cochez « Do not specify sources at this time » afin de vous épargner des fenêtres supplémentaires. Vous pouvez aussi passer les fenêtres successives avec « Next ».
8. Passez à la fenêtre suivante.





7. Dans la fenêtre suivante « Default Part », il s'agit de spécifier le composant. Vous allez rentrer les caractéristiques du FPGA implanté sur la carte Basys 3 dans cette fenêtre « Default Part ». Dans l'onglet « Parts » choisissez le



« xc7a35tcpg236-

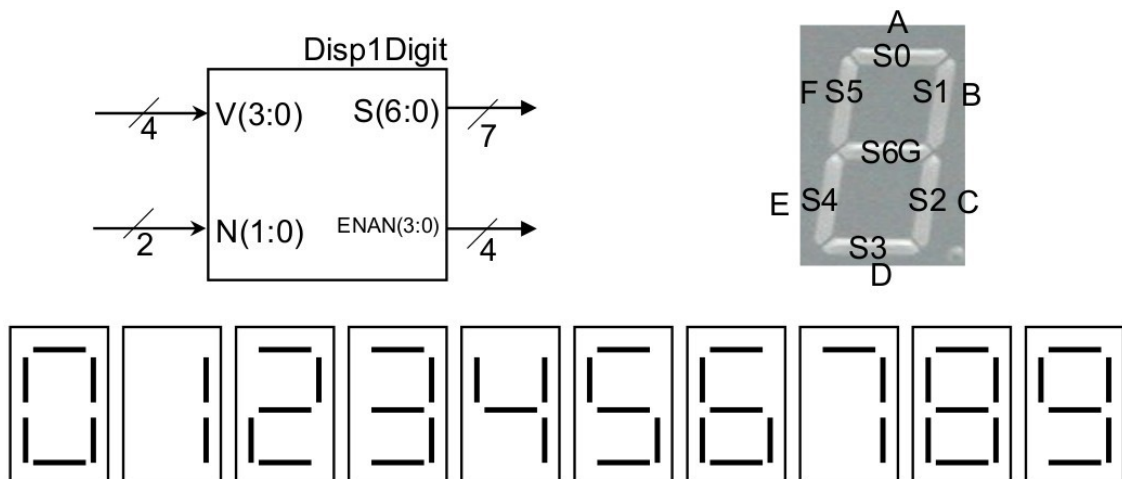
1 » en vous aidant du filtrage présenté

ci-contre. Passez à la fenêtre suivante par « Next> ».

8. Une dernière fenêtre récapitulative « New Project Summary » s'ouvre. Cliquez sur « Finish » pour ouvrir votre projet fraîchement créé.
9. Ce sera votre projet pour les 4 séances de TP; il n'y aura pas d'autre projet à créer. Vous viendrez y créer et ajouter vos propres modules VHDL.
10. Sur la gauche, vous devez avoir une fenêtre « Flow navigator ». C'est cette fenêtre que vous allez abondamment utiliser dans la suite. Elle présente un onglet « Project Manager ». Dans ce dernier cliquez sur « Project Settings ». Vous allez spécifier votre langage préféré: VHDL. Une fenêtre « Project Settings » s'ouvre. Sélectionnez « General » dans celle-ci et modifiez « Target language » en « VHDL » puis cliquez sur « OK ».

## 2.5 Affichage d'un chiffre décimal: Disp1Digit (travail #01)

Nous allons débiter par la création d'un module Disp1Digit permettant d'afficher une valeur en décimal sur un afficheur 7 segments. Ce module fera partie à terme du module global d'affichage. Nous avons besoin d'un décodeur 7 segments qui associe à une valeur V de 0 à 9 (les unités en décimal) l'affichage des segments d'un afficheur 7 segments. En plus du décodeur 7 segments, le bloc doit fournir un signal de sélection ENAN (de l'une des anodes communes) correspondant au chiffre N en entrée.



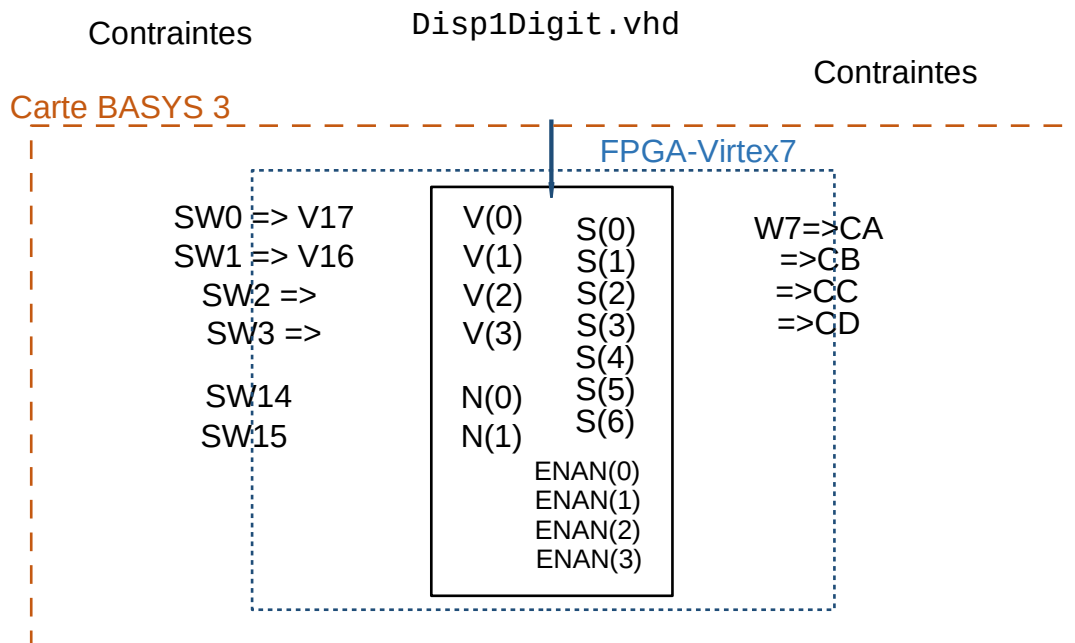


Les signaux de sélection ENAN(3:0) permettent de sélectionner lequel parmi les 4 afficheurs va être actif sur la carte Basys 3. C'est N(1:0) qui fournit la position. On choisit N=0 correspondant au chiffre le plus à droite.

Quelques détails sur les variables:

- V(3 downto 0) désigne la valeur V. Ce bus peut être décomposé sous la forme de 4 signaux de noms : V(3), V(2), V(1) et V(0). Il code en binaire naturel la valeur à afficher (avec V(3) étant le MSB).
- N(1downto 0) code le numéro de l'afficheur sélectionné (donc de 0 à 3).
- S(6 downto 0) est aussi un bus mais de sortie décomposable en S(6) à S(0) (ils correspondent aux S6 à S0 de l'afficheur).
- ENAN(3 downto 0) correspond aux 4 signaux envoyés chacun sur le transistor connecté à l'anode commune d'un afficheur. Un seul sera à 0 pour sélectionner le chiffre à afficher. Attention ENAN(0) est différent du signal AN0 vu précédemment. En effet le signal AN0 est le signal commun envoyé sur l'anode des LED de l'afficheur 0 alors que le signal ENAN(0) est le signal envoyé sur la base du transistor (connecté à l'anode commune). Il faut s'être posé la question concernant la valeur de ENAN(i) permettant de sélectionner l'afficheur i.
- Lorsque la valeur V représente une valeur supérieure ou égale à 10, on éteindra les segments!

L'objectif est maintenant de décrire les liens entre les broches réelles du composant que vous allez programmer et les entrées et sorties de votre description VHDL. En effet, votre description VHDL va être programmée à l'aide de blocs logiques sur le composant programmable. Vous allez donc avoir la réalisation suivante si l'on choisit de connecter le switch 0 (SW0) (donc sur la broche V17 du composant programmable) sur le LSB de la valeur c'est-à-dire V(0) :

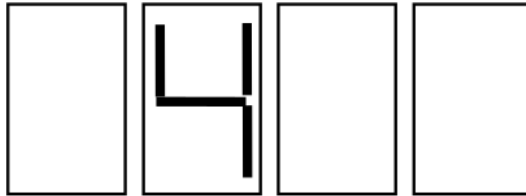


On appelle une « contrainte » le lien entre une entrée ou une sortie de votre description principale VHDL (Disp1Digit.vhd) et une broche réelle du composant programmable (FPGA-Artix-7). On vous propose dans la suite de décrire dans un tableau les contraintes de votre première réalisation.

**Préparation 1:**

**Prep1.A** Pour quel niveau logique sur S(i) le segment i d'un afficheur s'allume-t-il ?

Si l'on veut afficher le chiffre 4 sur l'afficheur 2. C'est-à-dire si l'on veut obtenir le résultat suivant :



**Prep1.B** Quel doivent être les entrées : N et V ? Quelles sorties S et ENAN doit alors générer le bloc « Disp1Digit » ?

**Prep1.C** Donnez la table de vérité et les équations simplifiées de S6 (G) à S0 (A) sous forme disjonctive (somme de produits) en fonction de V3 à V0. Respectez bien les noms car vous devrez retrouver ces équations dans la suite et mettez toujours les MSB à gauche et les LSB à droite.

**Prep1.D** Donnez l'architecture implémentée du bloc « Disp1Digit » : dessinez « Disp1Digit » uniquement à l'aide de blocs de type LUT (LUT1, LUT2, LUT3 ou LUT4) dont vous n'avez pas à préciser le contenu. Rappel : une LUT4 fournit une variable logique en fonction de 4 autres variables logiques. Combien de LUTs sont-elles nécessaires pour définir « Disp1Digit » ?

**Prep1.E** Précisez la table de vérité et les équations simplifiées de ENAN(0) à ENAN(3) sous forme disjonctive (somme de produits) en fonction de N0 et N1.

**Prep1.F** En vous appuyant sur les documentations de la carte Basys3 et sachant que l'on veut pouvoir tester en connectant les entrées V(3) à V(0) sur les « switch » glissants (ceux à droite, donc notés SW0 à SW3 sur la documentation Basys) de la carte, N(1) et N(0) sur les interrupteurs glissants de gauche, donnez l'association des variables S(6) à S(0) avec les numéros des broches du FPGA. Il faut pour cela consulter le « Digilent Basys3 FPGA Board Reference Manual ». Vous y trouverez, entre autre comme information que le switch glissant 0, noté SW0 est connecté à la broche V17 du FPGA. On veut le connecter ici à l'entrée V(0) de notre bloc « Disp1Digit ». Complétez le tableau des « contraintes » de placement des broches :

Nom dans la description VHDL (module hiérarchique supérieur)	Nom sur la carte (que vous pouvez lire sur la sérigraphie en TP)	Nom de la broche du FPGA (écrit sur le circuit entre parenthèse)	Type (entrée ou sortie vis-à-vis du FPGA)
V(3)	SW3		Entrée
V(2)	SW2		Entrée
V(1)	SW1	V16	Entrée
V(0)	SW0	V17	Entrée
N(1)			
N(0)			
S(6)			
S(5)			
S(4)			
S(3)			
S(2)			
S(1)			
S(0)			
ENAN(3)			
ENAN(2)			
ENAN(1)			
ENAN(0)			

Vous allez créer (quand on vous le demandera) le module VHDL « Disp1Digit » avec pour entrées les bus V(3:0) et N(1:0), pour sortie les bus S(6:0) et ENAN(3:0). Ensuite, pour ce premier exemple, nous passerons par toutes les étapes, (qui ne seront plus détaillés par la suite). Cela permet d'effectuer une prise en main complète du logiciel.

Vous allez exprimer S et aussi ENAN à l'aide d'une expression d'affectation conditionnelle de type « when/else » dont voici un exemple (et ce n'est qu'un exemple pour montrer la syntaxe) :

```

S <=      "1111110" when V="0000" else
          "0110000" when V="0001" else
          "1101101" when V="0010" else
...
          "1101101";

```

Remarque : ce type de code doit impérativement se terminer par une valeur comme ici ("1101101") et surtout, surtout pas un « when » (when V="0010") même si vous avez passé en revue toutes les combinaisons.

Ce code est à insérer dans le code VHDL entre « begin » et « end Behavioral ».

Vous pouvez aussi choisir cette façon équivalente sous forme d'un tableau de constantes pour décrire S et ENAN :

```
architecture Behavioral of encodeur is
    type tt is array (0 to 15) of std_logic_vector(6 downto 0);
    constant table:tt:=( "1111110",
                        "0110000",
                        ...,
                        "0000000");
    begin
        S <= table(to_integer(unsigned(V)));
    end Behavioral;
```

Il faut ajouter la bibliothèque « `numeric_std` » au début de votre code à la suite de la bibliothèque « `STD_LOGIC_1164` » en dé-commentant la ligne correspondante :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
```

### **Réalisation 1**

Dans la zone « Project Manager », cliquez sur « Add Sources ».

Dans le premier formulaire choisissez « Add or create design sources ». En effet nous voulons créer un module source VHDL « Disp1Digit ». « Next ».

Dans le deuxième formulaire cliquez sur « Create File ».

Une fenêtre s'ouvre. Choisissez « VHDL » pour « File Type », « Disp1Digit » pour « File Name » et « Local to Project » pour « File Location ». « OK ». « Finish ».

Ce n'est pas fini : une nouvelle fenêtre « Define Module » s'ouvre. Elle va vous permettre de définir les entrées et les sorties de votre module « Disp1Digit ». Allez-y ! Ca y est ? Vous devez avoir 4 bus dont 2 en entrées et 2 en sorties. Vérifiez la taille des bus. S'il y a une erreur, vous pourrez toujours corriger le code généré mais c'est mieux, surtout au début d'avoir le moins d'erreur possible à ce niveau-là. Donc « OK ».

Rien de spectaculaire n'arrive. Cependant vous devez voir apparaître dans l'arborescence de votre projet (fenêtre « Sources » de « Project Manager ») au niveau « Design Sources » le fichier « Disp1Digit ». Cliquez sur le fichier « Disp1Digit.vhd » dans l'arborescence pour l'éditer. Vérifiez que le bloc « entity » est correctement défini avec ses 4 bus : 2 en entrées et 2 en sorties.

Insérez au bon endroit le code de S et ENAN dans le module VHDL « Disp1Digit ». Définissez l'un d'eux par une structure de type « ...when...else » et l'autre par une structure de type tableau.

### **Analyse par blocs logiques (RTL) avant implémentation 1**

Une fois votre code rentré sans erreur de syntaxe, vous allez visualiser la vision qu'en a le logiciel à l'aide de l'outil RTL. Pour cela, commencez par sélectionner dans la fenêtre « Sources », dans « Design Sources », votre fichier « Disp1Digit » puis dans « Flow Navigator », « RTL Analysis », « Open Elaborated Design » cliquez sur « Schematic ». Visualisez le résultat obtenu. Redessinez (au crayon ou stylo) la structure proposée pour obtenir EN\_AN à l'aide de multiplexeurs, éventuellement de comparateurs et de N (à faire impérativement !). Soyez précis sur la description de vos multiplexeurs. Montrez en quoi ce résultat correspond bien à la structure que vous avez décrite. Vous remarquerez que la description peut utiliser des MUXs et qu'une entrée notée « V=B 0010, S=2'b01 » signifie que pour la sélection 01 on

trouve en sortie 0010 alors que V=default signifie que les autres sélections prennent la valeur présenté sur le bus en question. Vous pouvez accéder pour chaque multiplexeur à sa description détaillée : sélectionnez le multiplexeur que vous voulez étudier et faites apparaître le menu contextuel (clic droit) et sélectionnez « Cell Properties... » ; une fenêtre devrait s'ouvrir, donnant les détails des connections de ce multiplexeur.

Vous allez tester ce premier circuit en l'implémentant, c'est-à-dire en programmant le composant reconfigurable de la carte Basys 3. En effet le résultat est facilement accessible de manière visuelle sur la carte Basys3 : on sélectionne la valeur de 0 à 15 par 4 interrupteurs glissants et on sélectionne l'afficheur par 2 interrupteurs glissants. D'autres fois un test en simulation sera préférable pour valider le bon fonctionnement du module ou détecter facilement l'origine de dysfonctionnements.

Vous avez déjà décrit dans un tableau les liens entre les entrées et sorties de « Disp1Digit » qui est votre module hiérarchiquement le plus élevé (c'est le seul !) et les broches du composant. Vous allez devoir le faire pour le logiciel à travers un langage dont il suffit de vous donner un exemple pour le lien entre la broche V17 du composant à programmer et le signal V(0) de votre description VHDL:

```
set_property PACKAGE_PIN V17 [get_ports {V[0]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {V[0]}]
```

Attention : il faut respecter la casse pour les noms de signaux.

La première ligne établit le lien entre la broche « V17 » et l'entrée V(0).

La deuxième ligne semble superflue et pourtant elle ne l'est pas et elle doit bien être définie pour chaque signal d'entrée/sortie : elle spécifie la technologie et donc la tension correspondant au niveau logique haut. Ici 3,3 V est la tension maximale possible et elle est nécessaire car on doit commander des LED. On vous invite à recopier ces 2 lignes autant de fois que nécessaire dans le fichier de contraintes dans la suite.

### **Implémentation et test sur maquette 1**

Nous allons implémenter votre réalisation « Disp1Digit » directement sur le composant programmable de la Basys 3. Pour effectuer une implémentation de votre circuit, il est nécessaire que celui-ci apparaisse en tant que module hiérarchique supérieur : c'est toujours avec le module au sommet de votre description hiérarchique que s'effectuent les liens avec les broches du composant ; on parle de « top level ». Par défaut, lorsque vous travaillez avec un seul fichier, celui-ci est le module hiérarchique supérieur. Revenez dans l'onglet « Sources » pour bien sélectionner dans « Design Sources » votre unique (pour le moment) fichier VHDL (Disp1Digit).

Avant d'activer la création du fichier d'implémentation, vous devez spécifier vers quelles broches (réelles) du composant programmable vous allez rattacher les entrées et sorties définies dans votre module VHDL.

Pour cela vous allez dans « Flow Navigator », « Project Manager », « Add Sources » sélectionner « Add or Create Constraints » puis « Next » ». Dans la fenêtre « Add or Create Design Sources », utilisez « Add File » pour ajouter le fichier nommé « Disp1Digit\_Basys3.xdc » (à récupérer sur Moodle). Cochez « Copy constraints files into project » pour copier le fichier dans votre projet. « Finish ».

Vous devez disposer dans « Elaborated Design », « Sources » d'un fichier d'extension « xdc » dans le répertoire « Constraints ». Ouvrez-le, dé-commentez où

cela est nécessaire et complétez-le pour chaque signal d'entrée/sortie de « Disp1Digit ».

Cliquez sur « Synthesis », « Run Synthesis ». Patientez. Remarquez en haut à droite la barre de défilement.

Connectez la carte Basys3 en vous assurant que le Power est sur ON.

« Implementation » et « Run Implementation ».

Patience.

Choisissez «Program and Debug», « Generate Bitstream ».

Re-patience.

« Open Hardware Manager ».

« Open Target » « Auto Connect » si rien ne se passe.

Pour télécharger le code : « Program Device ».

Pour le test, commencez par mettre tous les interrupteurs au niveau bas.

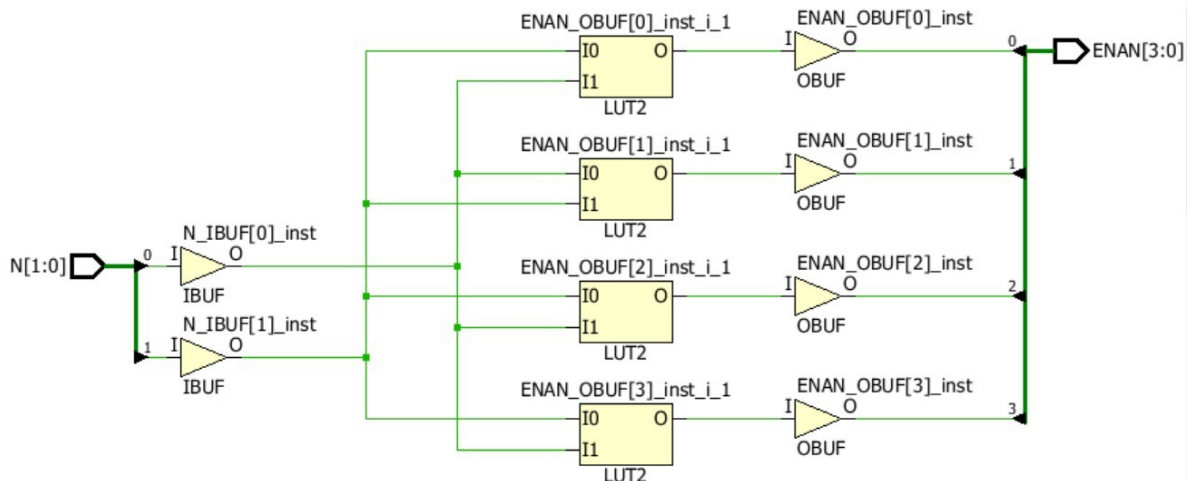
Le programme téléchargé, testez un grand nombre de combinaisons à l'aide des interrupteurs. Corrigez éventuellement les erreurs et re-testez l'ensemble jusqu'à obtenir le résultat escompté.

## **Analyse par ressource FPGA 1**

Le lancement de l'implémentation doit avoir généré des descriptions bas niveaux relatives à l'implémentation de votre projet.

Visualisez la représentation de l'implémentation par « Flow Navigator », « Synthesis », « Synthesized Design », « Schematic ».

Vous devriez obtenir une description de ce type (incomplète ici), à l'aide de LUTs et de buffers d'entrée ou de sortie:



Justifiez les blocs utilisés et leur nombre. Pour une LUT en particulier que vous choisirez, justifiez les valeurs trouvées à l'intérieur de la table de vérité (Truth Table). Faites le lien avec les ressources du FPGA qui ont été mobilisées pour décrire votre fonction. Pour cela dans la fenêtre horizontale du bas, cliquez sur l'onglet « Reports ». Ouvrez le fichier « Utilization Report » sous « Synth Design ». Cherchez « Primitives » et faites le lien avec les blocs visualisés.

Quel pourcentage représentent les cellules LUTs mobilisées par rapport aux cellules LUTs restant disponibles dans le FPGA que vous venez de programmer ? Indice : fichier « Utilization Report » sous « Place Design ».

Choisissez l'une des LUTs permettant d'obtenir un signal de ENAN (par exemple) et montrez que sa table de vérité correspond bien à ce qui est attendu.



Remarque : les questions précédentes d'analyse sont importantes, ne les sous-estimez pas. C'est ici que commence votre « valeur ajoutée » en tant qu'ingénieur.  
Imprimer ce dont vous avez besoin.  
On a fait un premier tour de ce qu'il y a « sous le moteur » mais on y reviendra.

## 2.6 Comment corriger les erreurs

Vous avez dû être confronté à diverses erreurs provenant de votre fichier VHDL. Voici quelques conseils pour corriger les erreurs de syntaxe provenant de vos descriptions VHDL :

- Prenez l'habitude **de tabuler systématiquement** vos descriptions de manière cohérente. La tabulation n'a pas d'effet direct sur les erreurs (contrairement au langage Python) : ici une mauvaise tabulation ne génère pas d'erreurs. Cependant il est plus facile de retrouver une erreur dans une description correctement tabulée que dans une description brouillonne. D'ailleurs, avant de vous venir en aide afin de rechercher une erreur, nous vous demanderons comme tout préalable de tabuler correctement votre description VHDL.
- Commencez par chercher à corriger la première erreur qui se trouve dans la liste. En effet les autres erreurs peuvent tout simplement être une conséquence de cette première erreur.
- Utilisez la facilité qui vous est offerte dans le logiciel consistant à repérer où se trouve l'erreur en cliquant dessus.
- Lisez calmement le texte. It is in English: what a surprise! KEEP CALM. N'hésitez pas à chercher la traduction de certains mots pour comprendre ce qu'a voulu vous dire le logiciel. Passez du temps sur la compréhension d'un message : vous aurez souvent les mêmes types d'erreur. Par exemple « width mismatch » signifie erreur sur la taille ou incompatibilité sur la taille.
- Demandez quand vous avez épuisé votre sagacité.

## 2.7 MUX4x4v1x4 (travail #02)

Nous aurons besoin d'effectuer un choix parmi 4 valeurs exprimées sur 4 bits. C'est la raison pour laquelle nous vous invitons maintenant à créer un multiplexeur de 4 signaux de 4 bits vers 1 signal de 4 bits. Vous avez à décrire un opérateur de multiplexage de 4 signaux: A(3:0), B(3:0), C(3:0) et D(3:0) vers une sortie O(3:0) dépendant d'une entrée de sélection Sel(1:0). Votre multiplexeur, sélectionne sur sa sortie O :

- A si Sel=0
- B si Sel=1
- C si Sel=2
- D si Sel=3

Décrivez ce multiplexeur en VHDL à l'aide d'une structure « When ... else ».

Vous aurez les entrées et les sorties suivantes pour votre fonction logique « MUX4x4v1x4 » :

- A(3 downto 0), B(3 downto 0), C(3 downto 0), D(3 downto 0) les 4 entrées sur 4 bits.
- Sel(1 downto 0) désigne l'entrée qui se retrouve en sortie.
- O(3 downto 0) est la sortie (O comme Output).

Votre code VHDL devrait ressembler à ceci :

```
-----  
-- ...  
-- Module Name:      MUX4x4v1x4 - Behavioral  
-----
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

...
entity MUX4x4v1x4 is
    Port (
        A : in  STD_LOGIC_VECTOR (3 downto 0);
        B : in  STD_LOGIC_VECTOR (3 downto 0);
        C : in  STD_LOGIC_VECTOR (3 downto 0);
        D : in  STD_LOGIC_VECTOR (3 downto 0);
        Sel : in  STD_LOGIC_VECTOR (1 downto 0);
        O : out STD_LOGIC_VECTOR (3 downto 0));
end MUX4x4v1x4;

architecture Behavioral of MUX4x4v1x4 is
    ...
begin
    O <=  A when Sel="00" else
         B when ...
         ;
end Behavioral;
```

## **Préparation 2:**

**Prep2.A** Donnez la réalisation d'un multiplexeur 4 voies vers 1 (MUX4x1v1x1 ou MUX4v1) à partir de multiplexeurs 2 vers 1 (MUX2v1).

**Prep2.B** Donnez la réalisation d'un multiplexeur 4 voies de 4 bits vers 1 voie de 4 bits (MUX4x4v1x4) à partir de multiplexeurs 4 vers 1. Combien de multiplexeurs MUX2v1 se cachent dans un multiplexeur MUX4x4v1x4 ?

**Prep2.C** Dessinez des chronogrammes de tests qui vont permettre de tester votre module. Remarque : le test peut être exhaustif mais le nombre de combinaisons de tests commence à être important. D'ailleurs combien de combinaisons de test différentes existe-t-il pour tester de manière exhaustive votre module ?

## **Réalisation 2**

Attention, nous ne répétons plus les étapes déjà détaillées.

Réalisez dans votre projet le module VHDL « MUX4x4v1x4 ».

Vous devez avoir dans votre projet 2 modules VHDL.

## **Simulation 2**

Nous allons faire un test du module « MUX4x4v1x4 » en simulation.

Le test de ce module demande la création d'un « test bench » en langage VHDL.

Cliquez sur « Add Sources » et cochez maintenant « Add or create simulation sources ». Cliquez « Next> ».

Créez un nouveau fichier source nommé « MUX4x4v1x4\_tb » par exemple avec le type VHDL sélectionné. Cliquez « OK ».

Cliquez « Finish » dans la fenêtre « Add Sources ».

La fenêtre « Define Module » s'ouvre pour le module de test « MUX4x4v1x4\_tb ». Ce module n'a pas d'entrée et sortie (il ne s'agit pas d'un composant), vous pouvez donc cliquer sur OK. Une fenêtre vient vous rappeler que vous n'avez pas ajouté d'entrée ou de sortie : en effet ! Cliquez sur « Yes ».

Ouvrez le fichier de simulation « MUX4x4v1x4\_tb.vhd » qui va vous servir à décrire la simulation de votre composant MUX4x4v1x4 décrit dans « MUX4x4v1x4.vhd ». Vous devez toujours avoir à l'esprit, pour ce type de fichier de simulation, que vous allez maintenant décrire la simulation de votre composant (et non plus un composant). Une simulation nécessite un composant (que vous avez décrit) mais aussi des signaux à imposer sur votre composant ou à observer. Vous devrez donc déclarer et instancier votre composant à simuler mais aussi décrire les signaux de simulation que vous allez envoyer en entrée de votre composant à tester et que vous allez observer.

Il y a déjà du code dans ce fichier (en fait pas grand-chose !). Vous allez devoir le compléter en 4 temps :

1. Déclaration du composant à tester.
2. Création des signaux de simulation.
3. Instanciation de votre composant à tester et connections aux signaux de simulation.
4. Description temporelle des signaux d'entrée de simulation (ou stimuli).

Vous devrez donc compléter le fichier « MUX4x4v1x4\_tb.vhd » qui comporte les lignes suivantes en insérant le code nécessaire dans les zones (1) à (4) (attention au « begin » entre les zones 2 et 3):

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX4x4v1x4_tb is
-- (0) : surtout ne rien mettre!
end MUX4x4v1x4_tb;

architecture Behavioral of MUX4x4v1x4_tb is
    (1): déclaration du composant à tester (ports)
    (2): déclaration des signaux de simulation
begin
    (3): instanciation de votre composant et connexions aux signaux
    (4): description temporelle des stimuli dans un process.
end Behavioral;
```

Dans la zone (0) vous devez ne rien mettre : ceci est bien un commentaire. Déclarer des ports ou autre chose dans cette zone amène des erreurs difficilement détectables. En effet le fichier que vous allez écrire décrit une simulation et non pas un composant : il n'y a pas de ports.

Maintenant, voyons voir à quoi correspondent les 4 zones principales et comment les écrire.

La grammaire générale pour la déclaration (1) du composant est la suivante :

```
component <component_name>
generic (
    <generic_name> : <type> := <value>;
```

```
<other generics>...
);
port (
  <port_name> : <mode> <type>;
  <other ports>...
);
end component;
```

Bon ça n'est pas très clair pour vous à ce stade.

Cela devient dans notre cas pour la déclaration (1) de votre composant multiplexeur:

```
component MUX4x4v1x4
port ( A : in STD_LOGIC_VECTOR (3 downto 0);
       B : in STD_LOGIC_VECTOR (3 downto 0);
       C : in STD_LOGIC_VECTOR (3 downto 0);
       D : in STD_LOGIC_VECTOR (3 downto 0);
       Sel : in STD_LOGIC_VECTOR (1 downto 0);
       O : out STD_LOGIC_VECTOR (3 downto 0));
end component;
```

Ce qu'il faut comprendre c'est que l'on déclare dans cette partie 1 le composant à simuler avec ses entrées et ses sorties telles que vous les avez définies précédemment.

Pour la déclaration des signaux de simulation (2) (qui ne sont considérés ni comme des entrées ni comme des sorties), nous vous conseillons de prendre les mêmes signaux :

```
signal A,B,C,D,O : STD_LOGIC_VECTOR (3 downto 0);
signal Sel : STD_LOGIC_VECTOR (1 downto 0);
```

Vous devez reprendre les signaux d'entrées et de sortie de votre composant. Ces signaux sont ici les signaux de simulation qui vont être connectés à votre composant.

Procédez maintenant à l'instanciation de votre composant (3) et les connections aux signaux de simulation :

```
UUT : MUX4x4v1x4 port map (A=>A,B=>B,C=>C,D=>D,Sel=>Sel,
O=>O) ;
```

La ligne ci-dessus signifie que vous connectez le signal de simulation A sur l'entrée A du composant MUX4x4v1x4, le signal de simulation B sur l'entrée B...

Remarque : vous pouvez donner un nom différent aux signaux de simulation que ceux de votre composant, comme par exemple :

```
signal sA,sB,sC,sD,sO : STD_LOGIC_VECTOR (3 downto 0);
signal sSel : STD_LOGIC_VECTOR (1 downto 0);
```

Dans ce cas là vous devrez écrire ainsi l'instanciation:

```
UUT : MUX4x4v1x4 port map (A=>sA,B=>sB,C=>sC,D=>sD,Sel=>sSel,
O=>sO) ;
```

Dans ce cas votre simulation fera apparaître ces signaux.

Enfin voici une description temporelle possible des signaux de tests (stimulis) (4) dans un **process**:

```
process
begin
```

```
A <= "0010";
B <= "0110";
C <= "0111";
D <= "1001";
Sel <= "00";
wait for 100 ns;
Sel <= "01";
wait for 100 ns;
...
wait;
end process;
```

Ici, on positionne les bus A à D et Sel. On attend 100 ns. Puis on modifie le bus Sel (les autres restent identiques)...

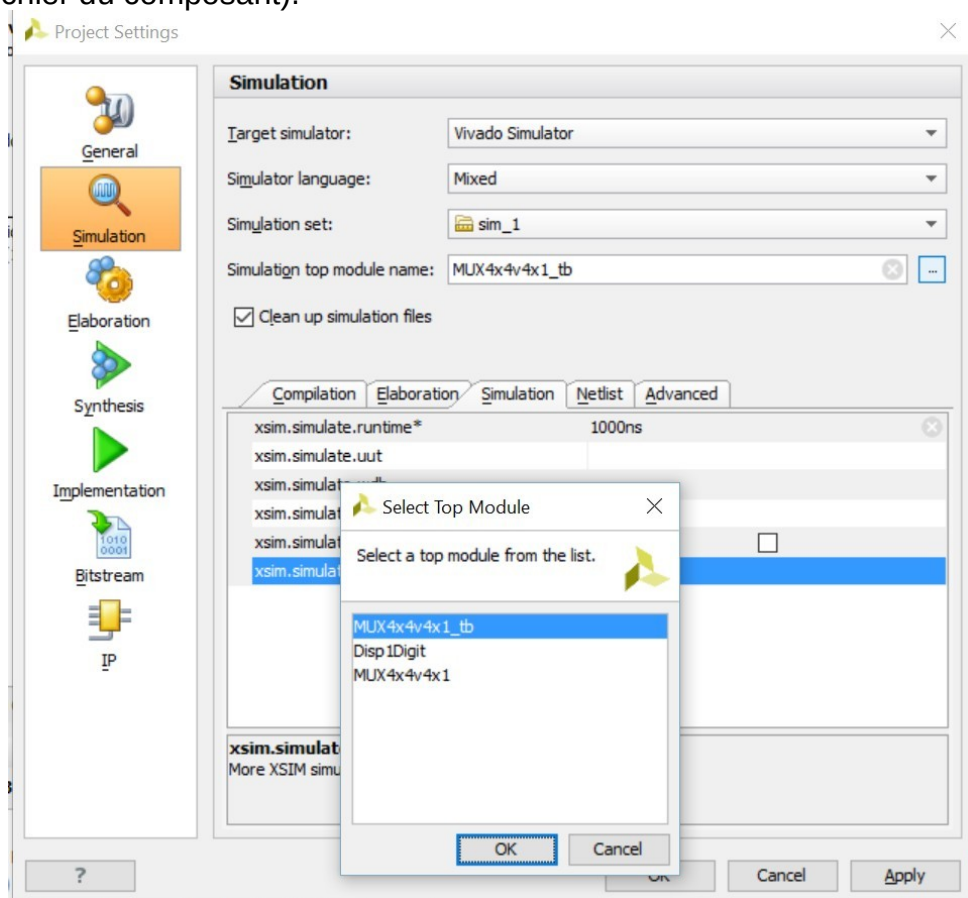
Le module « MUX4x4v1x4.vhdl » doit être positionné au sommet de la hiérarchie : « Set as Top ».

Assurez-vous que le module « MUX4x4v1x4\_tb.vhdl » est aussi au sommet de la hiérarchie de simulation.

Concernant le fichier de contraintes « \*.xdc » rendez le inactif pour le temps de la simulation par « Disable ».

Pour lancer une simulation le fichier « MUX4x4v1x4\_tb » doit être exempt d'erreurs de syntaxe et être au sommet de la hiérarchie de simulation. Pour cela ouvrez la fenêtre « Project settings » et sélectionnez « Simulation ».

Choisissez alors comme « top Module » pour la simulation le fichier de test (et non le fichier du composant).



Remarquez que vous pouvez modifier la durée totale de simulation.

Vous pouvez lancer la simulation par « Simulation », « Run Simulation ». Choisissez « Run Behavioral Simulation ».

Le résultat de la simulation ne semble pas convaincant au premier abord. En effet le simulateur (aussi appelé le farceur) ne visualise que la dernière pico seconde de simulation (très utile !). Il faut commencer par activer « Zoom to full view » avec



Vérifiez à l'aide du simulateur, le bon fonctionnement de votre module VHDL.

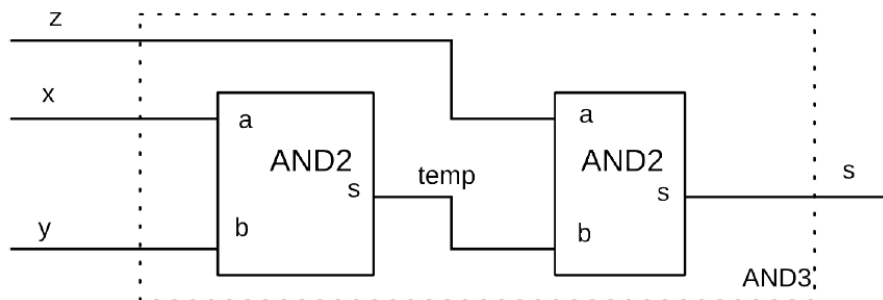
## 2.8 Attendez ! Remarque sur la simulation

La remarque qui est faite ici porte sur l'utilisation de l'instruction « wait » ou « wait for ... ». Cette instruction ne peut être utilisée que dans le cadre d'une simulation, elle permet de passer une commande d'attente au simulateur. En aucun cas elle ne permet de décrire du matériel (Hardware Description) et donc elle génère une erreur si vous l'utilisez en dehors d'un process de simulation. On dit que cette instruction n'est pas synthétisable : elle ne décrit pas de matériel. C'est dommage car cela aurait été utile dans la suite ! Il faut vous ôter de l'esprit cette possibilité. La commande « wait » correspond à une action et non à une description de matériel. Pour effectuer une attente sur un matériel numérique réel, nous allons devoir décrire des composants de type « compteur » et les mettre en œuvre.

## 2.9 Réutilisation d'un bloc précédemment décrit

On appelle « structurelles » dans un langage de description matériel les descriptions qui font appel à des blocs hiérarchiques déjà construits.

Supposons que vous venez de définir une entité « AND2 », dont les entrées sont « a » et « b » et la sortie « s ». Si l'on souhaite créer une fonction AND à 3 entrées, on peut tout à fait proposer la réalisation suivante (d'autres solutions sont possibles) :



Comment décrire en langage VHDL, une nouvelle entité AND3 définie hiérarchiquement à partir de l'entité AND2 ? Voici le code VHDL du fichier « AND3.vhd » correspondant au schéma ci-dessus :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity AND3 is
    Port ( x,y,z : in STD_LOGIC;
          s : out STD_LOGIC);
end AND3;

architecture Behavioral of AND3 is
    signal temp :STD_LOGIC ;
```



```
begin
  u1 : entity work.AND2 port map (a => x, b=> y, s=> temp);
  u2 : entity work.AND2 port map (a=> z, b=> temp, s=>s);
end Behavioral;
```

Remarque 1 : la définition d'un signal « temp » entre « is » et « begin » qui permet de relier la sortie s de u1 à l'entrée b de u2.

Le nom u1 désigne le nom du « boîtier » et *entity* rappelle la description de l'intérieur du « boîtier », l'instruction *port map* permet lier les signaux internes de AND2 (a, b et s) aux signaux de l'entité AND3.

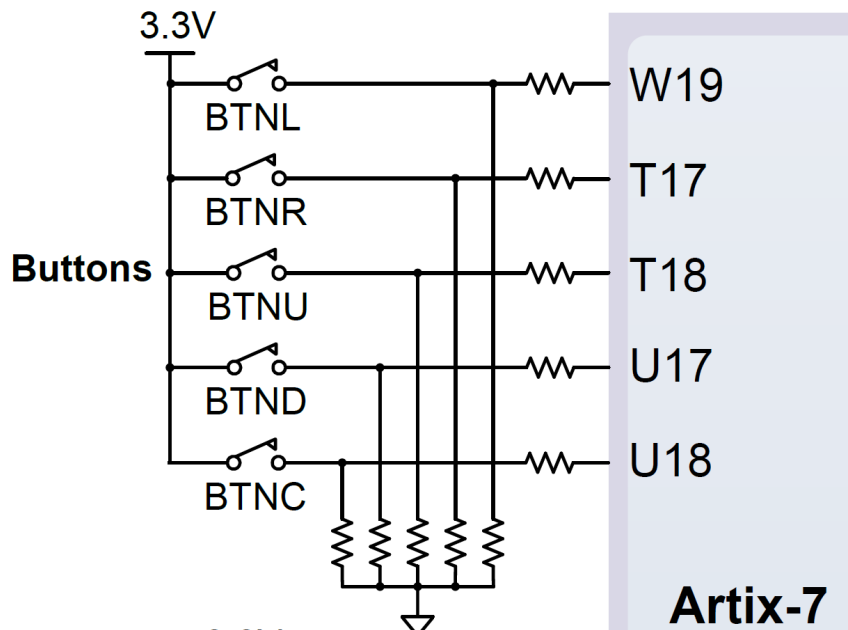
De la même manière que lors d'un appel de fonction en C une variable n'a pas le même nom au sein de la fonction appelante et de la fonction appelée, il n'est pas nécessaire de donner le même nom aux signaux qui interconnectent les boîtes noires. Attention les similarités avec le C se limitent à cela.

Vous remarquerez que si l'on sait tracer un schéma à l'aide de blocs déjà définis, l'écriture du code VHDL devient évidente. **Il faut donc impérativement tracer votre schéma avant de passer à l'écriture du code VHDL.**

## TP2

### 2.10 Des boutons pour la sélection

La carte possède des boutons que vous pourrez utiliser pour votre test : deux des boutons permettront la sélection de l'afficheur 7 segments parmi les 4 disponibles.



### 2.11 Disp1of4Digits (travail #03)

Le but est maintenant de décrire un composant qui permette d'afficher un seul chiffre décimal de 0 à 9 choisi parmi 4 chiffres (si c'est supérieur strictement à 9 on n'affiche rien). Les 4 chiffres décimaux seront représentés chacun par 4 bits (codage en binaire naturel) : A(3:0), B(3:0), C(3:0) et D(3:0). L'entrée de sélection N(1:0) va fournir à la fois laquelle des valeurs est à afficher sur les afficheurs 7 segments et à quelle position. Ainsi, on réalise :

- Si N=0 □ affichage en décimal de A sur l'afficheur le plus à droite
- Si N=1 □ affichage en décimal de B sur l'afficheur du milieu droit

- Si N=2 □ affichage en décimal de C sur l'afficheur du milieu gauche
- Si N=3 □ affichage en décimal de D sur l'afficheur le plus à gauche

On a donc la correspondance suivante avec les afficheurs: DCBA.

On vous conseille pour écrire ce VHDL de ré-exploiter la démarche utilisée pour « Disp1Digit ».

Pour être précis, vous aurez les entrées et les sorties suivantes pour votre fonction logique Disp1of4Digits :

- A(3 downto 0) désigne la valeur à afficher (en décimal) sur l'afficheur le plus à droite.
- B(3 downto 0)...
- C(3 downto 0)...
- D(3 downto 0)...
- N(1 downto 0) désigne à la fois la position et la valeur à afficher comme détaillé ci-dessus.
- S(6 downto 0) correspond aux segments des afficheurs.
- ENAN(3:0) correspond à 4 signaux pour commander les anodes communes dont un seul sera à 0 pour sélectionner le chiffre à afficher.

Si par exemple D=5, C=6, B=7, A=8, N=3 alors S représente les segments pour afficher 5 et ENAN sélectionne l'afficheur le plus à gauche.

Vous remarquerez qu'en plus des entrées et des sorties, on a ajouté un vecteur logique V(3:0) de type « signal » et qu'il a été ajouté entre les mots clés « architecture ...is » et « begin ». Ici c'est un signal interne dont on a besoin comme intermédiaire.

Votre code VHDL devrait ressembler à ceci :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
...
entity Disp1of4Digits is
    Port (
        A : in  STD_LOGIC_VECTOR (3 downto 0);
        B : in  STD_LOGIC_VECTOR (3 downto 0);
        C : in  STD_LOGIC_VECTOR (3 downto 0);
        D : in  STD_LOGIC_VECTOR (3 downto 0);
        N : in  STD_LOGIC_VECTOR (1 downto 0);
        ENAN : out STD_LOGIC_VECTOR (3 downto 0);
        S : out  STD_LOGIC_VECTOR (6 downto 0));
end Disp1of4Digits;

architecture Behavioral of Disp1of4Digits is
    signal V: STD_LOGIC_VECTOR(3 downto 0);
begin
    u1 : entity work.MUX4x4v1x4
        port map (A=> A, B=> B, C=>C, D=>D, N => ... ) ;

    u2 : entity work.Disp1Digit
        port map ( ... ) ;

end Behavioral;
```

## **Préparation 3:**

**Prep3.A** Donnez la réalisation de « Disp1of4Digits » sous forme d'un logigramme (dessin avec des circuits logiques), en réutilisant les modules précédemment définis (Disp1Digit et MUX4x4v1x4). Soyez précis dans votre dessin : spécifiez la taille des bus.

**Prep3.B** Créez un fichier de contraintes permettant de tester les fonctionnalités de votre réalisation sur la carte Basys. Les entrées A, B, C et D seront connectées aux interrupteurs glissants, N à 2 boutons, ENAN et S aux afficheurs 7 segments.

## TP EN 1AB CHRONOMETRE 2018-2019

Nom dans la description VHDL (module hiérarchique supérieur)	Nom sur la carte (que vous pouvez lire sur la sérigraphie en TP)	Nom de la broche du FPGA (écrit sur le circuit entre parenthèse)	Type (entrée ou sortie vis-à-vis du FPGA)
D(3)	SW15		Entrée
D(2)	SW14		Entrée
D(1)	SW13		Entrée
D(0)	SW12		Entrée
C(3)	SW11		Entrée
C(2)	SW10		Entrée
C(1)	SW9		Entrée
C(0)	SW8		Entrée
B(3)	SW7		Entrée
B(2)	SW6		Entrée
B(1)	SW5		Entrée
B(0)	SW4		Entrée
A(3)	SW3		Entrée
A(2)	SW2		Entrée
A(1)	SW1	V16	Entrée
A(0)	SW0	V17	Entrée
N(1)			
N(0)			
S(6)			
S(5)			
S(4)			
S(3)			
S(2)			
S(1)			
S(0)			
ENAN(3)			
ENAN(2)			
ENAN(1)			
ENAN(0)			

### **Réalisation 3**

Réalisez dans votre projet le module VHDL « Disp1of4Digits » répondant au cahier des charges. Vérifiez que votre fichier « Disp1of4Digits.vhd » est au sommet (top) de votre hiérarchie (doit apparaître en gras dans « Sources », « Design

sources »). Pour cela dans la fenêtre « Sources » vérifiez que « Disp1of4Digits » apparaît en gras. Si ce n'était pas le cas, sélectionnez ce fichier, faites apparaître le menu contextuel et sélectionnez « Set as Top » (non disponible si déjà au sommet de la hiérarchie). Visualisez-le à l'aide de « RTL Analysis », « Elaborated Design », « Schematic ». Vérifiez ainsi que le schéma généré à partir de votre description VHDL correspond bien à ce que vous vouliez décrire à l'origine.

N'oubliez pas de conserver une trace de ce résultat. Pour cela, vous pouvez faire une copie d'écran mais aussi une sauvegarde au format PDF.

Pour effectuer une sauvegarde au format PDF, allez sur le schéma à l'aide de la souris et faites apparaître le menu contextuel par un clic droit. Sélectionnez « save as PDF file ».

### **Implémentation 3**

Le module « Disp1of4Digits » doit apparaître au sommet (top) de la hiérarchie du projet. Vous devrez toujours vous assurer que le fichier au sommet de la hiérarchie est celui que vous voulez implémenter. Créez un nouveau fichier de contrainte et complétez-le. Désactiver le précédent fichier (disable). Relancez une synthèse. Implémentez et testez les fonctionnalités de votre réalisation : faites afficher des valeurs différentes sur les afficheurs 7 segments en fonction des boutons appuyés. Appelez l'enseignant pour lui montrer.

### **2.12 Cmp0to3 (travail #04)**

Vous disposez d'un afficheur pouvant afficher un seul chiffre décimal. On va exploiter cet afficheur comme on l'a vu précédemment pour donner l'impression que l'on affiche 4 chiffres à la fois. Pour cela, il faut envoyer sur les entrées A, B, C et D les 4 valeurs à afficher « simultanément » et faire varier rapidement l'entrée N de manière à sélectionner successivement et assez rapidement les 4 chiffres. Pour cela nous avons besoin d'un compteur qui va générer N : 0, 1, 2, 3, 0, 1... C'est-à-dire un compteur synchrone cyclique de 0 à 3. Nous entrons dans le monde de la logique séquentielle synchrone.

Les signaux de « Cmp0to3 » sont:

- CLK désigne le signal de synchronisation. Les fronts montants vont marquer l'évolution du compteur.
- EN permet ou non l'évolution du compteur. Si EN='1' alors le compteur évolue. Si EN='0' le compteur reste dans le même état.
- N(1:0) est la sortie qui va évoluer cycliquement : 0, 1, 2, 3, 0, 1...

### **Préparation 4:**

**Prep4.A** Effectuez la synthèse d'un compteur évoluant de 0 à 3 mais sans « Enable » (c'est-à-dire sans l'entrée EN : en effet on peut connecter l'entrée EN sur des bascules D qui en sont munies).

**Prep4.B** Donnez son schéma à l'aide de bascules D (avec ENABLE) et les parties combinatoires sous la forme de LUTs dont vous préciserez la table de vérité.

## **Réalisation 4**

Réalisez le module VHDL « Cmpt0to3.vhd ».

En plus de la bibliothèque standard, vous insérerez la bibliothèque suivante :

```
use ieee.numeric_std.all ;
```

Cela permet de d'avoir accès au type « unsigned » pour lequel l'opérateur « + » est défini.

Le bloc « architecture » a la structure suivante :

```
architecture Behavioral of cmpt0to3en is
  signal M: unsigned(1 downto 0):=(OTHERS => '0');
begin
  process(clk)begin
    if rising_edge(clk)then
      --insérez ici l'évolution de M
      if(EN='1') then
        M<=... ;
      else
        M<=... ;
      end if ;
    end if;
  end process;
  --recopie de M sur la sortie N
  N<=M;
end Behavioral;
```

Une petite remarque qui peut vous arranger ici : l'addition effectuée sur 2 bits correspond à une addition modulo  $2^2=4$ . Ainsi  $3+1=0$ .

L'introduction du signal M n'est pas qu'un simple artifice. En effet on a l'impression que l'on pourrait s'en passer puisque la sortie N n'est que la recopie de M. Cependant il est bien nécessaire de déclarer un signal interne M. Car une sortie (par exemple N) ne peut pas être à la fois une sortie et une entrée ce qui serait le cas en écrivant quelque chose comme: «  $N \leq N+1$  ». Ce point est abordé en cours.

Vous remarquerez aussi que l'on a initialisé le signal M. La valeur d'initialisation est « (OTHERS=>'0') ». Ceci signifie que tous les bits du bus sont à 0. On aurait pu écrire « "00" ».

## **Test en simulation 4**

Créer un fichier de simulation « Cmpt0to3\_tb.vhd » pour tester maintenant votre composant Cmpt0to3 décrit dans le fichier « Cmpt0to3.vhd ». Complétez-le le fichier de test comme cela a été fait lors de la simulation précédente. En plus des 4 éléments précédemment vus (déclaration du composant, création des signaux, instanciation du composant et description des stimuli) vous devez ajouter une constante pour l'horloge CLK et la génération de l'horloge.

La constante doit être ajoutée entre « architecture... is » et « begin » :

```
-- Clock period definitions
constant CLK_period : time := 10 ns;
```

Ensuite ajoutez la génération de l'horloge entre « begin » et « end Behavioral » :

```
-- Clock process definitions
CLK_process :process
```



```
begin
    CLK <= '0';
    wait for CLK_period/2;
    CLK <= '1';
    wait for CLK_period/2;
end process;
```

Vous devez aussi ajouter à la suite de la génération de l'horloge la description des autres signaux de simulations, ici EN :

```
process
begin
    EN <= '1';
    ...
    wait;
end process;
```

Votre fichier « Cmt0to3\_tb.vhd » doit comporter ces différents blocs.

N'oubliez pas de mettre votre module « Cmt0to3\_tb » au sommet de la hiérarchie de simulation. Testez votre réalisation par une simulation (Behavioral).

### **Implémentation et analyse 4**

Utilisez les outils d'analyse (vus pour le 1) afin de fournir le schéma logique (RTL) et le schéma d'analyse (Analysis) d'implémentation interne de votre module « Cmt0to3 » dans le FPGA. Faites le lien avec votre description trouvée en préparation. Analysez les équations des tables de vérité des LUTs.

Remarque 1 : assurez-vous de bien positionner au sommet de la hiérarchie Cmt0to3 dans « Design Sources ».

Remarque 2 : on ne vous le répétera pas.

Remarque 3 : lorsque vous ouvrez un schéma celui-ci peut ne pas être actualisé ; sélectionnez « reload » dans le menu apparaissant au-dessus des sources.

### **2.13 Disp4D (travail #05)**

Grâce au module « Disp1of4 » nous pouvons afficher 4 valeurs différentes sur 4 afficheurs. On a besoin de créer un module qui va afficher successivement et automatiquement chacune des 4 valeurs en entrée en utilisant le module « Disp1of4 » associé au module « Cmt0to3 ».

Nous avons les signaux suivants :

- A(3:0) désigne la valeur à afficher (en décimal) sur l'afficheur le plus à droite.
- B(3:0)...
- C(3:0)...
- D(3:0)...
- CLK est l'horloge de synchronisation (à 100 MHz).
- S(6:0) correspond aux segments des afficheurs.
- ENAN(3:0) correspond à 4 signaux de sélection dont un seul sera à 0 pour sélectionner le chiffre à afficher.

### **Préparation 5:**

**Prep5.A** Donnez la réalisation de ce module Disp4D (sous la forme d'un dessin) en réutilisant les blocs précédents.

## Réalisation 5

Réalisez le module VHDL « Disp4D ».

Remarque : vous pouvez connecter une entrée à une valeur constante :

```
compteur: entity work.Cmpt0to3  
port map(CLK=>CLK, EN=>'1', N=>Sel);
```

Vous avez dans cet exemple (qui n'est qu'un exemple) un composant de type « Cmpt0to3 » (que vous avez créé), que vous nommez « compteur » dont l'entrée EN est reliée à '1' (je rappelle que ce n'est qu'un exemple).

## Test en implémentation et mesures 5

Dans le fichier de contrainte, mettez en commentaire les connexions avec les boutons. Conservez les connexions avec les interrupteurs glissants pour rentrer les valeurs à afficher et les connexions avec les afficheurs 7 segments. Ajoutez une connexion de deux des signaux ENAN(0) et ENAN(1) vers deux des broches des 4 connecteurs PMOD : A14 et A15 sur le connecteur JB par exemple. N'oubliez pas d'ajouter aux contraintes l'horloge CLK qui est disponible sur la carte et connectée à la broche W5 du FPGA.

Implémentez cette réalisation sur la carte. Qu'observe-t-on si l'on positionne 4 valeurs différentes sur les 4 groupes de switches? Pourquoi ça ne marche pas ? Qu'observe-t-on si l'on positionne maintenant 4 valeurs identiques sur les 4 groupes de switches? Pourquoi cela marche ?

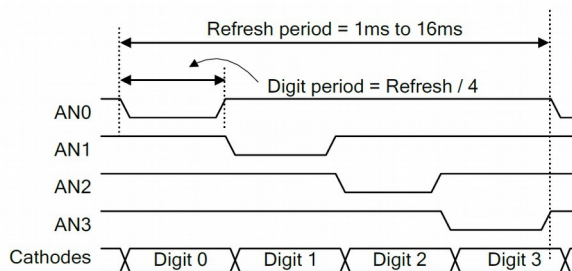


Figure 19. Four digit scanning display controller timing diagram.

Mais est-ce compatible avec ce qui est demandé dans la documentation ?

## TP3

### 2.14 TICK\_1ms (travail #06)

L'horloge dont nous disposons sur la carte est issue d'un oscillateur à 100 MHz connecté à la broche W5 du FPGA. Nous la nommons CLK dans notre code VHDL. Vous venez de l'utiliser pour synchroniser votre affichage et vous avez pu constater que ça allait trop vite. Le composant a été conçu pour être synchrone et l'on ne dispose que de cette horloge à 100 MHz. C'est la raison pour laquelle, nous vous proposons de créer un module « TICK\_1ms ». Ce module va nous permettre de commander au bon rythme l'afficheur 7 segments mais aussi dans la suite de compter le temps.

L'objectif de « TICK\_1ms » est de générer un signal qui passe à 1 pendant une période de CLK et cela une fois toute les millisecondes, tout cela à partir de l'horloge à 100 MHz.

Vous nommerez ainsi les signaux de « TICK\_1ms »:

- CLK désigne le signal de synchronisation (ici à 100 MHz).
- Tick est le signal généré.

## **Préparation 6:**

**Prep6.A** Quel est le rapport de division entre l'horloge d'entrée et celle du signal de sortie ?

**Prep6.B** La réalisation est basée sur un comptage. Combien de bascules D sont-elles nécessaires pour compter le rapport de division ?

**Prep6.C** Ecrivez le module VHDL selon le modèle suggéré à compléter :

```
architecture Behavioral of TICK_1ms is
  signal Q: integer range 0 to n_div := 0;
begin
  process(CLK)begin
    if rising_edge(CLK)then
      if(Q = ...)then
        Q <= ...
      else
        Q <= ...;
      end if;
    end if;
  end process;
  Tick<= ... when Q ... else ...;
end Behavioral;
```

**Prep6.D** Quel test proposez-vous pour valider TICK\_1ms ?  
**Simulation ou implémentation ? Décrivez votre mode opératoire dans le cas choisi.**

Vous remarquerez que dans la structure suggérée, on traite d'une part de l'évolution synchrone du signal Q dans un « process » sensible à l'horloge d'évolution qui est ici « CLK » et d'autre part du calcul combinatoire de « Tick » en fonction de la valeur de « Q » mais en dehors du process de CLK.

Remarque : vous noterez d'ailleurs que dans un « process » on peut utiliser le mot clé « if » mais pas la structure « ...<=...when ... else... ». De même, vous pouvez utiliser la structure « ...<=...when ... else... » hors d'un « process » mais pas à l'intérieur. De plus mettre à jour un signal dans le « process » peut s'avérer plein de pièges. Nous vous conseillons donc fortement cette méthode :

- Evolution d'un signal sur front montant d'un signal de synchronisation dans un « process » qui ne réagit que sur le signal de synchronisation avec un « rising\_edge ».
- Calculs combinatoires hors du process en fonction de la variable d'évolution.

En résumé, le process n'est là que pour décrire une bascule D.

## **Réalisation 6**

Réalisez le module VHDL « TICK\_1ms ».

Il est intéressant d'utiliser des entiers qui vont servir de paramètres. Si c'est le cas vous pouvez les déclarer comme constante au même endroit que les signaux:

```
architecture Behavioral of TICK_1ms is
    constant Ndiv: integer := 5;
    signal ...;
begin
```

### **Simulation 6**

Mettez au point le test pour « TICK\_1ms ».

La simulation risque d'être très longue et peu exploitable si vous gardez le rapport de division exact. Il est préférable de lancer une simulation avec un rapport de division faible. Il est alors intéressant d'avoir seulement une constante comme ci-dessus qu'il suffit de modifier pour modifier la génération du signal de sortie.

Testez votre réalisation en simulation.

Vous pouvez effectuer un test en implémentation en prévoyant de sortir l'un des signaux et en l'observant à l'oscilloscope.

### **2.15 Disp4D again (travail #07)**

Nous allons compléter le module VHDL « Disp4D » afin que notre réalisation fonctionne correctement.

### **Réalisation 7**

Complétez le module VHDL « Disp4D » en y insérant le module « TICK\_1ms ».

Vérifier que votre description correspond bien à ce que vous voulez (RTL schematic).

### **Test 7 sur carte Basys**

Effectuez le test sur carte Basys.

Si le test ne fonctionnait pas et que l'erreur ne vous apparaît pas immédiatement, il faut revenir à une simulation avec un rapport de division réduit.

### **2.16 Cnt0toN\_EN (generic) (travail #08)**

Nous allons maintenant compter le temps synchronisé par le signal d'horloge à 100 MHz ceci à l'aide d'une association de compteurs. Nous aurons besoin de compteurs par 10 et par 6 qui puissent s'associer. Afin d'éviter d'avoir à construire un compteur par 10 puis un autre par 6, nous allons simplement décrire un compteur général paramétrable qui compte par N (ici N peut aller de 2 à 16) (N est la longueur du cycle). Nous allons utiliser la spécificité du langage VHDL de définir des termes de type « generic ».

On écrit juste avant l'instruction port le paramètre:

```
entity NAME_OF_ENTITY is
    generic generic_declarations;
    port (    signal_names: mode type;
            ...
            signal_names: mode type);
end NAME_OF_ENTITY ;
```

Dans notre cas, la déclaration peut être:

```
generic (N : integer :=10);
```

La valeur « 10 » signifie ici c'est la valeur par défaut si le paramètre n'est pas défini au moment de l'instanciation.

Modifiez le fichier pour que la valeur du cycle de comptage (N ici) soit un paramètre (instruction « generic ») que l'on donne au moment de l'instanciation.

Notre compteur doit posséder les caractéristiques suivantes :

- Comptage de 0 à N-1 (codé sur le bus de sortie Q).

- Entrée de validation du comptage.
  - Sortie de retenue pour permettre l'association des compteurs.
- Notre module a les entrées/sorties suivantes :
- CLK est l'horloge de synchronisation à 100 MHz.
  - EN permet de valider l'évolution du comptage.
  - CLR permet de mettre à 0 le comptage de manière prioritaire mais synchrone.
  - Cout est la retenue permettant d'associer les compteurs.
  - Q(3:0) représente l'état du comptage.

## **Préparation 8:**

**Prep8.A** Ecrivez le module VHDL « Cnt0toN\_EN ».

## **Réalisation 8**

Réalisez le module VHDL « Cnt0toN\_EN ».

## **Test en simulation 8**

Testez le module VHDL « Cnt0toN\_EN ». Soyez le plus exhaustif possible.

## **TP4**

### **2.17 Chrono en mode continu (travail #09)**

L'objectif est maintenant de venir afficher les minutes sur les 2 afficheurs gauches et les secondes sur les 2 à droite. Vous devez aussi compter les milli secondes, les centièmes de secondes, les dixièmes de secondes, les unités de secondes, les dizaines de secondes, les unités de minute, les dizaines de minutes.

Pour cela, vous allez créer un module « Chrono » dont les entrées sont : CLK, EN et CLR. Le signal CLR est prioritaire et met à 0 le chrono. Le chrono compte le temps lorsque EN=1. La sortie représente le temps écoulé par un vecteur logique T de 28 bits (27 downto 0) où T(3:0) représente les unités de millièmes de seconde, T(7:4) représente les dizaines de millièmes de seconde,... T(27:24) le chiffre des dizaines de minute. Mais c'est du BCD ! diront, tout de suite, les plus rapides d'entre vous. Pour réaliser ce bloc vous avez à votre disposition le bloc TICK\_1ms ainsi que les Cnt0toN.

Vous allez maintenant utiliser la possibilité de définir des compteurs comptant des cycles de 6 par exemple (ou par 10 pourquoi pas). Pour cela la déclaration de votre composant dans « chrono » doit être de la forme :

```
comptdizs: entity work.Cnt0toN_EN
  generic map(N=>6)
  port map(CLK=>CLK, EN=>..., ...);
```

Remarque : vous aurez besoin de ce module « Chrono » ainsi que du module précédent « Disp4D » pour le TP 5 d'électronique numérique planifié pour la fin Février. Ne détruisez donc pas votre travail : vous allez encore l'améliorer.

## **Préparation 9:**

**Prep9.A** Réalisez un schéma « chrono » utilisant « TICK\_1ms » et des compteurs « Cnt0toN\_EN » pour compter le temps des millisecondes aux dizaines de minutes.

## **Réalisation 9**

Créez un nouveau module VHDL « ChronoEnContinu » pour obtenir le comptage et l'affichage des minutes et des secondes en continu sur la carte Basys 3.

## **Test en simulation ou sur carte 9**

Testez, montrez les résultats à l'enseignant.

## **Analyse 9**

Testez, montrez les résultats à l'enseignant.

## **2.18 Chronomètre à 2 interrupteurs glissants (Chrono\_SW)**

Vous allez réaliser un chronomètre assez simple à l'aide de 3 interrupteurs glissants:

- Un premier interrupteur glissant doit permettre de mettre à 0 le chronomètre de manière prioritaire.
- Un autre interrupteur doit permettre, selon sa position d'arrêter ou d'activer le comptage.

## **Préparation 10:**

**Prep10.A** Réalisez un schéma « chrono\_SW » utilisant « TICK\_1ms » et des compteurs « Cnt0toN\_EN » pour compter le temps des millisecondes aux dizaines de minutes commandé par 2 interrupteurs glissants.

## **Réalisation 10**

Créez le module VHDL « Chrono\_SW » pour obtenir le comptage et l'affichage des minutes et des secondes en fonction de 2 interrupteurs glissants.

## **Test en simulation ou sur carte 10**

Testez, montrez les résultats à l'enseignant.

## **2.19 Changement de niveau sur détection de fronts montants : DFM (travail #11)**

Un chronomètre s'utilise plus communément par des appuis sur des boutons c'est pour cela que nous allons avoir besoin de détecter un appui sur un bouton donc de détecter un front montant.

Il s'agit de construire le système synchrone suivant.

ENTREES :

CLK : Horloge de synchronisation.

Btn : signal issu d'un bouton.

SORTIES :

Detect : passe à 1 pendant une seule période de CLK lorsqu'un front montant a été détecté sur « Btn ».

Niveau : bascule (ou « toggle » c'est-à-dire passe alternativement de 1 à 0 ou de 0 à 1) à chaque nouvelle détection d'un front montant sur Btn.

On propose de réaliser de la manière suivante DFM.

entity DFM is

```
Port ( CLK : in STD_LOGIC;  
      Btn : in STD_LOGIC;  
      Niveau : out STD_LOGIC;  
      Detect : out STD_LOGIC);
```

end DFM;



```
architecture Behavioral of DFM is
    signal BtnCurr,BtnPrev : STD_LOGIC:= '0';
    signal Q,D : STD_LOGIC_VECTOR(1 downto 0):= "00";
    signal EF,EP : STD_LOGIC:= '0';
begin
    --PARTIE 1
    process(CLK) begin
        if RISING_EDGE (CLK)then
            BtnCurr <= Btn;
            BtnPrev <= BtnCurr;
        end if;
    end process;

    --PARTIE 2 début
    D(1) <= (NOT(Q(1)) and Q(0)) or ((Q(1) and NOT(Q(0)))
and (BtnCurr or BtnPrev));
    D(0) <= (NOT(Q(1)) and NOT(Q(0))) and(BtnCurr and
NOT(BtnPrev));

    process(CLK) begin
        if RISING_EDGE (CLK)then
            Q(1) <= D(1);
            Q(0) <= D(0);
        end if;
    end process;
    -- PARTIE 2 fin
    -- PARTIE 3
    EF <= EP XOR Q(0);
    process(CLK) begin
        if RISING_EDGE (CLK)then
            EP <= EF;
        end if;
    end process;

    Detect <= Q(0);
    Niveau <= EP;

end Behavioral;
```

## **Préparation 11:**

**Prep11.A** Dessinez la réalisation de la partie 1 de DFM à l'aide de bascules D.

**Prep11.B** Dessinez la réalisation de la partie 2 de DFM à l'aide de bascules D et de portes logiques.

### **Prep11.C**

Ré-écrivez proprement les équations de D1 et D0 (pas de simplification demandée).

D1=

D0=

On peut interpréter ce bloc comme un compteur à 3 états (00 01 et 10) dont le codage de l'état présent est fourni par Q1 et Q0. Les entrées BtnCurr et BtnPrev représentent les entrées de ce compteur et orientent son comptage (comme EN et CLR sur un compteur). D1 et D0 fournissent le codage de l'état futur. A partir des équations de D1 et D0, complétez le graphe des transitions de la partie 2 de DFM.

Q1	Q0	BtnCurr	BtnPrev	D1	D0
0	0	0	X		
0	0	1	1		
0	0	1	0		
0	1	X	X		
1	0	1	X		
1	0	X	1		
1	0	0	0		

Comment peut-on se retrouver dans l'état Q1Q0=00 (attention on ne demande pas pour D1D0=00 mais bien pour Q1Q0)? Quel état Q1Q0 correspond à la détection d'un front montant ?

**Prep11.D** Expliquez le principe de cette détection à partir des blocs mis en œuvre.

**Prep11.E** Dessinez la réalisation de la partie 3 de DFM à l'aide de bascules D et de portes logiques. Que permet d'obtenir cette partie

### Réalisation 11

Créez DFM et insérez-le à votre projet.

### Test en réel 11

Testez DFM seule : Btn doit être issu d'un bouton et Niveau doit être envoyé sur une LED. Il y a un fichier de contrainte à créer. Procédez à une implémentation pour générer le fichier de test à télécharger sur le composant.

### Analyses 11

Faites afficher les schémas d'analyse RTL et analyse pre-implémentation. Commentez.

## **2.20 Chronomètre à boutons : chrono\_BTN (travail #12)**

Il s'agit maintenant de construire un chronomètre évoluant sur l'appui de 2 boutons.

ENTREES :

CLK : Horloge de synchronisation (à 100 MHz),

BTN\_GO\_STOP : arrêt et lancement du chronométrage.

BTN\_RESET : remise à 0 du chronomètre.

SORTIES :

Pour permettre l'affichage

## **Préparation 12:**

**Prep12.A** Dessinez chrono\_BT.

**Prep12.B** Combien de bascules D sont nécessaires à la réalisation?

## **Réalisation 12**

Réalisez chrono\_BTN en VHDL en vous inspirant du modèle ci-dessus.

## **Implémentation et test sur maquette 12**

Implémentez et testez votre chronomètre à bouton sur maquette.

## **Analyse 12**

Justifiez le nombre de bascules D et de LUTs utilisées dans cette implémentation. Réalisez un tableau pour donner le nombre de bascules et de LUTs de chacun des blocs.

## **3 VHDL mon ami**

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use ieee.numeric_std.all;
```

permettent l'utilisation des opérateurs arithmétiques.

La fonction «to\_integer(arg)» convertit «arg» en une valeur entière. Ici «arg» doit être de type «unsigned» ou «signed». Pour convertir un «std\_logic\_vector» en valeur entière, il faut d'abord le convertir en une valeur signée ou non signée, à l'aide des fonctions de conversion «signed(arg)» ou «unsigned(arg)».

Exemple :

```
signal slv : std_logic_vector (3 downto 0);  
signal i1, i2 : integer;  
...  
slv <= "1001";  
i1 <= to_integer(unsigned(slv)); -- 9  
i2 <= to_integer(signed(slv)); -- -7
```

Les fonctions «to\_unsigned(arg, size)» et «to\_signed(arg, size)» convertissent «arg» respectivement en un unsigned» ou un signed». «size» correspond au nombre de bits sur lequel effectuer la conversion. On peut utiliser l'attribut «length» du signal à convertir. On peut alors simplement convertir ces signaux en «std\_logic\_vector» à l'aide de la fonction «std\_logic\_vector(arg)».

Exemple :

```
signal i1, i2 : integer;  
signal slv1, slv2 : std_logic_vector(3 downto 0);  
...  
i1 <= 9;  
i2 <= -7;  
wait for 10 ns;  
slv1 <= std_logic_vector(to_unsigned(i1,slv'length));  
slv2 <= std_logic_vector(to_signed(i2,slv'length ));
```

On peut utiliser la concaténation: a(5 downto 2) & b(3 downto 1).

```
-- No clocks detected in port list. Replace <clock> below with  
-- appropriate port name  
    constant <clock>_period : time := 10 ns;  
...
```

### **4 VIVADO mon meilleur ennemi**

Remarque : il peut être intéressant de mettre en « disable » les sources lors d'un pb.