TRAVAUX PRATIQUES

SYSTÈMES D'INFORMATION NUMÉRIQUES

Logique combinatoire

2020-2021

Ce document appartient à _____

TP 1

Fonctions fondamentales de la logique combinatoire

Objectifs

- Comprendre les objectifs et le fonctionnement des TP
- Prendre en main le logiciel Quartus
  - Appréhender la notion de projet
  - Savoir manipuler les fichiers dans un projet
- Mettre en œuvre les fonctions de base de l'électronique numérique
  - Multiplexage
  - Décodage
- Manipuler les différentes formes de tables de vérité
  - Simple
  - Avec priorité
  - À variables introduites

Fonctions logiques étudiées

- Multiplexage
- Décodage

Composants fournis

- *(Aucun)*

1. Introduction

1.1. Propos et objectif des TP

Au cours des séances de TP de SIN, vous mettrez en œuvre des systèmes numériques, d'abord combinatoires puis séquentiels. Ces séances vous permettront de développer vos connaissances dans ce domaine et de les mettre concrètement en application.

Nous utiliserons pour cela le langage VHDL, qui permet de décrire des circuits logiques sur la base d'un support textuel. Si la connaissance et la compréhension de ce langage sont bien entendu nécessaires, il ne s'agit ici que d'un outil parmi d'autres, au même titre qu'une représentation schématique ou une description par équations. L'élément commun fondamental dans toutes ces démarches est la méthode, car elle permet de traiter tout type de problème.

1.2. Sources d'information

Outre le présent cahier de TP, vous aurez besoin de vos TD et de votre cours. Par ailleurs, un guide présentant la démarche et les outils que l'on utilisera en électronique numérique vous a été distribué. Celui-ci constitue une ressource fondamentale pour la démarche, le fonctionnement du logiciel Quartus et la platine Cyclone GEII, **Vous devez disposer de tous ces documents à chaque séance et vous y référer dès que nécessaire.**

Enfin, vous avez accès à un espace Moodle « Systèmes d'information numériques ». Celui-ci rassemble toutes les informations nécessaires sur les TP de SIN. Vous y trouverez notamment :

- Les informations concernant les contrôles de TP, y compris des versions numériques du présent cahier de TP et du guide,
- Des fichiers source à télécharger,
- Toutes vos notes de TP et de contrôle de TP.

Il est nécessaire de se référer à cet espace en premier lieu lorsque vous avez des questions concernant le fonctionnement des TP.

1.3. Déroulement des TP

Les TP se déroulent en plusieurs phases récurrentes : une phase d'analyse, une phase de création d'un composant, une phase de simulation et une phase de test sur platine réelle.

La phase d'analyse consiste en l'étude du composant et sa description à l'aide de schémas et de tables de vérité. Celle-ci se base sur l'analyse préalable du composant que vous aurez généralement réalisée en cours et/ou en TD. Les questions de la phase d'analyse sont présentées en italiques et doivent être validées par un enseignant. Il est conseillé de préparer la phase d'analyse avant la séance afin d'optimiser votre temps en salle de TP.

Lors des phases de test, chaque étape doit être validée par un enseignant : simulation, test sur site, ou toute autre manipulation expressément définie comme telle.

Les validations sont repérées dans le texte à l'aide du pictogramme suivant : . Une note sanctionne chaque TP sur la base de vos validations. L'ensemble des notes de TP compte pour 50% de la note globale des TP, les 50% restants étant fournis par la séance de contrôle de TP.

1.4. Utilisation du matériel

Le matériel utilisé en TP consiste en un PC et une platine programmable, que l'on connectera parfois à une maquette.

Par ailleurs**, votre table doit être systématiquement rangée en fin de séance** : extinction de la platine et du PC, débranchement de la maquette si utilisée, aucun papier ou objet divers restant sur la table. Un malus sera appliqué à votre note de TP en l'absence d'application de cette consigne.

1.5. Environnement de travail

Le logiciel Quartus doit être installé sur votre PC personnel afin de pouvoir travailler depuis chez vous. Celui-ci est disponible à la fois sur le serveur « commun » (à copier sur une clé USB), ou depuis un lien de téléchargement sur le cours Moodle. Lorsque vous devez transférer du code entre différents projets, par exemple ramener des fichiers déjà complétés en dehors des séances, il est très important de **ne copier que les fichiers sources** (fichiers avec une extension .vhd ou .vwf pour les simulations), et non l'intégralité du projet.

La version de Quartus à utiliser pour les TP de SIN est la **version 9.0** du logiciel.

2. Fonction multiplexage

2.1. Cahier des charges

La fonction multiplexage est la fonction combinatoire la plus utilisée en électronique numérique. C'est une fonction d'aiguillage de plusieurs entrées vers une sortie.

L'interface de cette fonction comporte 3 parties :

- les entrées de données *E[m..0]*,
- les entrées de sélection (ou d'adresse) *A[n..0]*,
- la sortie *Y*.

À tout moment, la sortie doit afficher la valeur de l'entrée de donnée dont le numéro est égal à l'adresse indiquée sur les entrées de sélection : par exemple, lorsque l'adresse est définie sur « 10 » (2 en binaire), c'est l'entrée 2 qui sera recopiée sur la sortie.

Nous voulons réaliser un composant mettant en œuvre la fonction multiplexage avec 4 entrées de données. On aura donc 2 entrées d'adresse, car il faut 2 bits pour choisir parmi 4 valeurs ($2^2 = 4$). On nommera ce composant un multiplexeur 4 vers 1.

2.2. Analyse – Modélisation

Un multiplexeur étant un système combinatoire, son modèle est une table de vérité. Celle-ci a été présentée en cours et en TD. Si ce n'est déjà fait, sortez votre cours et votre TD pour vous y référer.

Comme notre composant dispose de 6 entrées au total (4 entrées de données + 2 entrées de sélection), la table de vérité associée devrait avoir $2^6 = 64$ lignes. Afin de réduire ce nombre, nous utiliserons une table de vérité à variables introduites.

Complétez les pointillés sur les deux représentations ci-dessous en ajoutant le nom des entrées/sorties.

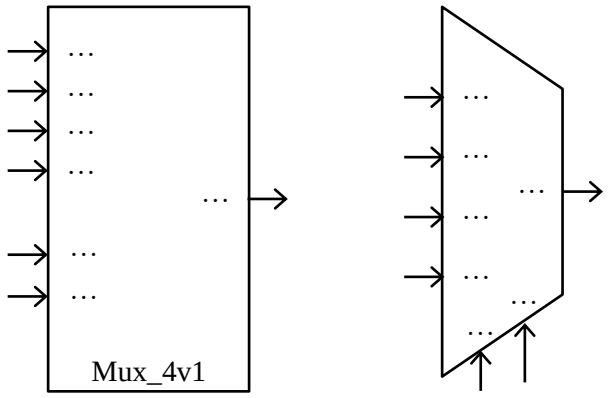
Vue composant et représentation usuelle d'un multiplexeur de 4 vers 1

Nom du composant :

Mux_4v1

Entrées/sorties :

E[3..0] : entrées de données
A[1..0] : entrées de sélection
Y : sortie du multiplexeur



Vue composant Représentation usuelle

Complétez la table de vérité à variables introduites du multiplexeur 4 vers 1.

Entrées		Sortie
A[1]	A[0]	Y

Quelle est l'instruction VHDL adaptée pour décrire cette table ? *Réponse :* ……

Faites valider les trois questions par un enseignant

2.3. Construction du composant

Pour cette première manipulation, nous allons décrire toutes les étapes de la construction de composant. Par la suite nous indiquerons uniquement « Construction du composant », et vous devrez vous référer à cette section et/ou au guide pour retrouver le détail de la démarche.

Ouverture du projet et du fichier

- En utilisant le menu **File ➔ Open Project**, ouvrez le projet **composants.qpf** situé dans le dossier **projet_composants**.
  - Le projet **composants** est configuré de manière à être simple à utiliser pour la construction de composants et la simulation. On utilisera également un autre projet pour un autre but plus tard dans le TP. Il est important de comprendre à quoi sert chacun des deux projets.
- Si vous avez déjà utilisé le projet auparavant, par exemple lors des TP d'introduction, celui-ci contient déjà des fichiers, visibles dans sur l'onglet **Files** de l'encart **Project Navigator** à gauche. Dans ce cas, commencez par nettoyer le projet en supprimant tous ces fichiers à l'exception de **sin_pack.vhd** que l'on utilisera à chaque fois. Vous pouvez supprimer ces fichiers sans risque, ceux-ci resteront disponibles dans le dossier **sources_composants**.
- Ajoutez le fichier **Mux_4v1.vhd** situé dans le dossier **sources_composants** au projet :
  - Ouvrez le menu **Project ➔ Add/Remove Files in Project**,
  - Cliquez sur le bouton représentant des points de suspension,
  - Ouvrez le dossier **sources_composants**, sélectionnez le fichier et cliquez sur **Ouvrir** puis sur **Add**.

Édition

- Le fichier apparait maintenant dans la liste des fichiers du projet, accessible en cliquant sur l'onglet **Files** dans l'encart **Project Navigator** à gauche. Double-cliquez dessus pour l'ouvrir.
- Ajoutez au fichier **Mux_4v1.vhd** le code correspondant à la table de vérité de la page précédente. Pensez à sauvegarder votre fichier après édition, et même régulièrement pendant l'édition.

Compilation

- Il faut indiquer au logiciel quel est le composant sur lequel on est en train de travailler. Mettez le composant **Mux_4v1** en top level en faisant un clic-doit sur le fichier **Mux_4v1.vhd** et en sélectionnant **Set as Top-Level Entity**. La notion de Top-Level est importante à comprendre : référez-vous à la section 4.5.4 du guide pour plus de détails.
- Lancez la compilation en accédant au menu **Processing ➔ Start Compilation**.
- La console au bas visible au bas de la fenêtre affiche de nombreux messages. Deux onglets sont à vérifier : **Warning** et **Error**. Dans chacun des deux onglets, vérifiez qu'il n'y ait aucun message, et s'il y en a, corrigez-les. Pour les erreurs, vous pouvez accéder à la ligne fautive en double-cliquant dessus. Accédez toujours à la première erreur en premier, car les erreurs suivantes sont parfois causées par celle-ci. Corrigez toutes les erreurs et tous les warnings, sauvegardez et recompilez.

Mise du composant dans un package

- Ajoutez le fichier package **sin_pack.vhd** au projet s'il n'est pas déjà présent. Ce fichier est disponible dans le dossier **sources_composants**.
- Copiez l'ensemble de l'entité du fichier **Mux4v1.vhd**, y compris les lignes `entity` et `end entity;`
- Collez dans le fichier package l'entité copiée, juste après la ligne `package sin_pack is`, puis :
  - remplacez `entity Mux_4v1` par `component Mux_4v1`
  - remplacez `end entity;` par `end component;`
- Sauvegardez le fichier package.

Le fichier package est compilé automatiquement avec les fichiers sources associés s'il est mis dans le projet. On ne doit pas le compiler individuellement.
Il ne faut jamais le mettre en top level, car il ne s'agit pas d'un composant.

2.4. Simulation du composant

Pour cette première manipulation, nous allons décrire toutes les étapes de la simulation d'un composant. Par la suite nous indiquerons uniquement « Simulation du composant », et vous devrez vous référer à cette section et/ou au guide pour retrouver le détail de la démarche.

Procédure de création d'un fichier de simulation

- Créez la feuille de simulation. On nommera toujours celle-ci du nom du composant simulé.
  - Création du fichier :
    - **File ➔ New**
    - Choisissez le type de fichier **Vector Waveform File**
  - Sauvegarde la feuille de simulation :
    - File ➔ Save
    - Placez-vous dans le dossier **simulations_composants** et saisissez pour nom de fichier **Mux_4v1.vwf**
- Définition des paramètres de la feuille de simulation.
  - Il est nécessaire de définir le pas de la grille de simulation :
    - **Edit ➔ Grid size**
    - Modifiez le temps. La valeur à utiliser en SIN sera toujours de 100 ns.
  - On peut définir la durée de la simulation :
    - **Edit ➔ End time**
    - Modifiez la durée : il faudra à chaque fois estimer la durée nécessaire pour bien montrer tous les modes de fonctionnement. Pour cet exercice, choisissez 5 µs.
- Insertion sur la feuille de simulation des entrées-sorties du système à simuler.
  - Cliquez avec le bouton droit dans la colonne **Name** de la feuille de simulation
  - Sélectionnez **Insert ➔ Insert Node or Bus**…
  - Une nouvelle fenêtre s'ouvre :
    - Choisissez le Radix **Hexadécimal**
    - Cliquez sur **Node Finder**…
  - Une nouvelle fenêtre apparaît avec les champs suivants :
    Named * **Filter:** Pins: assigned
    look in: |Mux_4v1|
  - Vérifiez que **Named** est bien suivi de * et **Look in** de |Mux_4v1|
  - Modifiez **Filter :** pour le mettre à la valeur **Pins: all**
  - Cliquez sur le bouton **List**. Ceci fait apparaître tous les signaux du composant.
  - Sélectionnez les signaux à faire passer dans la fenêtre de droite : pour cela, il faut faire passer les signaux dans la partie **Selected Nodes**. Pour cela, il suffit de cliquer sur la double flèche vers la droite qui est entre les deux champs (>>) pour choisir tous les signaux.
  - Cliquez sur **Ok** pour valider la liste et revenez sur la feuille de répondant **OK** à nouveau.

Attention, la vue de la fenêtre est parfois mal réglée au début. Avant de passer à la suite, pointez la souris sur le chronogramme, maintenez la touche « ctrl » du clavier appuyée, puis descendez la molette de la souris jusqu'à ce que plus rien ne bouge, afin de vous assurer de bien visualiser l'ensemble de la simulation.

Sur la simulation, on fera apparaitre tous les comportements combinatoires pertinents :
- Choisissez la valeur d'adresse « 00 » sur *A[1..0] sur plusieurs colonnes dans la simulation*
- Changez la valeur de l'entrée *E[0]* plusieurs fois sur dans la partie où *A[1..0]* vaut « 00 ».
- Changez ensuite la valeur d'une autre entrée dans une partie où *A[1..0]* vaut « 00 » et *E[0]* a une valeur fixe.
- Répétez l'opération avec toutes les valeurs possibles de l'entrée d'adresse.

Simulez et vérifiez que le comportement est correct. Si ce n'est pas le cas, modifiez le composant, compilez à nouveau et re-simulez.

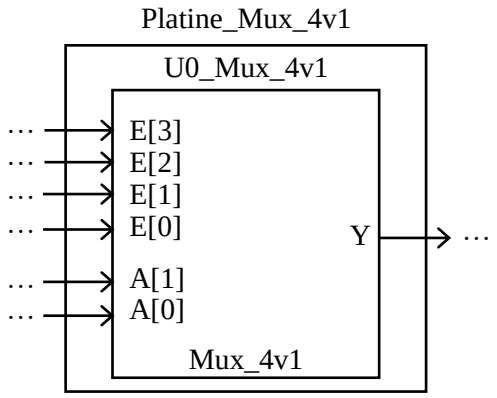
Lorsque le comportement est correct, faites noter par un enseignant auquel vous expliquerez la simulation.

2.5. Connexion sur la platine

2.5.1. Schéma

Pour tester le composant, nous devons le placer dans le FPGA. Pour cela, nous utilisons un composant spécial représentant le FPGA. On appelle ce composant **Platine**. Il permet d'utiliser les entrées/sorties physiques du FPGA.

Complétez le schéma ci-dessous représentant l'instanciation du multiplexeur et ses connexions avec la platine selon les relations fournies. Faites valider le schéma par un enseignant

Relations avec la platine	**Schéma de connexion du composant sur la platine**
« ➔ » signifie « est connecté à » E[3] ➔ inter[3] E[2] ➔ inter[2] E[1] ➔ inter[1] E[0] ➔ inter[0] A[1] ➔ inter[7] A[0] ➔ inter[6] Y ➔ led[0]	

2.5.2. Instanciation du composant sur la platine

Nous allons décrire ici toutes les étapes de la connexion du composant. Par la suite nous indiquerons uniquement « Instanciation du composant sur la platine », et vous devrez vous référer à cette section et/ou au guide pour retrouver le détail de la démarche.

Ouverture du projet et des fichiers

- Ouvrez le projet platine en utilisant le menu **File ➔ Open Project** et en sélectionnant le fichier **carte_cycloneII.qpf** dans le dossier **projet_platine**.
  - Le projet **Platine** est un projet qui contient toutes les informations sur les entrées/sorties physiques du FPGA. Il faudra toujours utiliser celui-ci pour réaliser le test sur site.
- Si le projet a déjà été utilisé, nettoyez-le en supprimant tous les fichiers du projet, à l'exception de **sin_pack.vhd** s'il est présent et du fichier **cyclone_pack.vhd** qui permet le bon fonctionnement du projet.
- Ajoutez le fichier **Platine_Mux_4v1.vhd**. Le fichier est disponible dans le dossier **sources_platines**.
- Ajoutez au projet le fichier comportant la description VHDL du composant **Mux_4v1** : **Mux_4v1.vhd**.
- Ajoutez au projet le fichier package **sin_pack.vhd** s'il n'est pas déjà présent.
- S'il n'est pas présent, ajoutez le fichier package **cyclone_pack.vhd** disponible dans **sources_platine**.

Édition

- Ouvrez le fichier **Platine_Mux_4v1.vhd**.
- Insérez l'invocation du package **sin_pack** au début du fichier **Platine_Mux4v1.vhd** en ajoutant les deux lignes suivantes avant la déclaration de l'entité :

```
library work;
use     work.sin_pack.all;
```

- Instanciez votre composant après le **begin** de l'architecture. Référez-vous au cours pour la syntaxe à utiliser, et au schéma d'instanciation ci-dessus pour connaître les connexions à réaliser.
- Mettez en commentaire la ligne déjà présente dans le fichier et affectant une valeur à **led[0]**.
  - En effet, une sortie ne devant jamais être laissée sans valeur, des valeurs par défaut sont déjà fournies pour toutes les sorties dans les fichiers platine. À chaque fois que vous utiliserez une sortie, il faudra donc mettre en commentaire ou supprimer la ligne existante dans le code fourni.
- Sauvegardez le fichier.

Compilation

- Mettez le fichier **Platine_Mux_4v1.vhd** en top level.
- Compilez.
- Dans les onglets **Warning** et **Error**, vérifiez qu'il n'y a aucun avertissement ni aucune erreur.
- Corrigez les éventuels problèmes et recompilez le cas échéant.

2.5.3. Test sur la platine

Nous allons décrire ici toutes les étapes de l'application du test sur la platine. Par la suite, nous indiquerons uniquement « Test sur la platine », et vous devrez vous référer à cette section et/ou au guide pour retrouver le détail de la démarche.

Préparation de la platine

- Connectez la platine à l'alimentation (prise de courant) et à l'ordinateur (câble USB).
- Allumez la platine à l'aide de l'interrupteur en bas à gauche.

Chargement du circuit sur la platine

- Ouvrez le menu **Tools ➔ Programmer**,
- Configurez le type de connexion à la platine :
  - Cliquez sur le bouton **Hardware Setup**,
  - Dépliez la liste déroulante **Currently selected hardware:**,
  - Sélectionnez dans la liste déroulante **USB_Blaster**,
  - Fermez la fenêtre avec le bouton **Close**.
- Sélectionnez le fichier à utiliser pour programmer le circuit dans le FPGA :
  - Dans la partie droite de l'outil **Programmer**, en blanc, sélectionnez le fichier déjà présent,
  - Supprimez le fichier déjà présent dans la liste en cliquant sur le bouton **Delete**,
  - Ajoutez le fichier **carte_cycloneII.pof** présent dans le dossier **projet_platine** à l'aide du bouton **Add File…** Attention à l'extension, ne pas prendre le fichier **.sof** !
    - Apprès avoir ajouté le fichier, vérifiez bien que le nom est exactement **carte_cycloneII.pof** sans aucun « / » dans le nom.
- Cochez les cases **Program/Configure** et **Verify**.
- Cliquez sur **Start**.
- Attendez que la barre de progression en haut à droite se remplisse. La programmation peut prendre un certain temps à démarrer, c'est tout à fait normal.
- Lorsque la programmation du composant est terminée, la diode située en bas à gauche de la platine doit clignoter pour indiquer que le composant est prêt à être utilisé.

Test du composant

Assurez-vous du bon fonctionnement du composant en vérifiant toutes les combinaisons d'entrée possibles. Par exemple, ici :

- Réglez les entrées d'adresse sur la valeur « 00 »,
- Modifiez la valeur de chacune des 4 entrées de donnée pour voir le résultat sur les diodes,
- Pour chaque combinaison d'entrée, assurez-vous que le résultat est correct : que fait un multiplexeur ? Quel est doit être le comportement de la sortie vis-à-vis des entrées de données lorsque les entrées de sélection sont réglées sur l'adresse « 00 » ?
- Recommencez pour les adresses « 01 », « 10 » et « 11 ».

Si le résultat obtenu n'est pas celui attendu, il faut corriger votre code VHDL. Modifiez votre code, compilez à nouveau et rechargez votre composant sur la platine.

Une fois que le comportement correspond à celui attendu, faites valider à un enseignant.

3. Fonction décodage

3.1. Cahier des charges

La fonction décodage consiste à activer une sortie sélectionnée parmi plusieurs. En entrée, on doit indiquer le numéro (adresse) de la sortie à activer. On parle de décodage 1 parmi N pour N sorties.

Cette fonction comporte *n* entrées de sélection (ou d'adresse) et $N \leq 2^n$ sorties dont une seule au plus est activée pour chacune des combinaisons des entrées.

La sortie activée est celle dont le numéro correspond à l'équivalent décimal de la combinaison binaire appliquée sur les entrées de sélection. Par exemple, pour une adresse *A* = « 010 », la sortie 2 est activée.

Nous voulons réaliser un décodeur 1 parmi 8 avec une entrée de validation *en* (enable). Lorsque l'entrée de validation sera égale à 0, toutes les sorties seront forcées à 0.

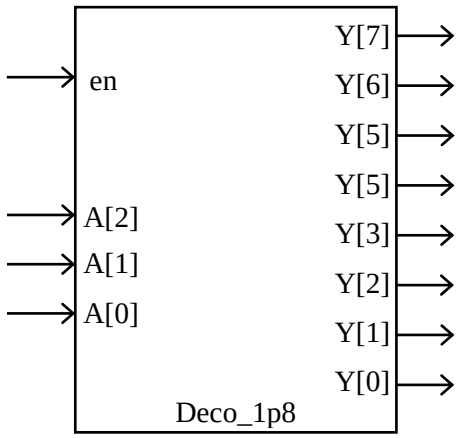
Vue composant du décodeur 1 parmi 8

Nom du composant :

Deco_1p8

Entrées/sorties :

A[2..0] : entrées de sélection
en : entrée de validation
Y[7..0] : sorties actives au niveau haut



3.2. Analyse – Modélisation

On peut décrire ce décodeur par l'une des 3 tables de vérité ci-dessous :

- table de vérité simple (16 lignes car 4 entrées : en, A[2], A[1], A[0])
- table de vérité avec priorité (9 lignes, l'entrée *en* est prioritaire sur A[2..0])
- table de vérité à variables introduites (l'entrée *en* est utilisée comme variable introduite)

Complétez la table de vérité simple et la table de vérité avec priorité du décodeur en page suivante.

Pour simplifier le remplissage, les 0 pourront être omis dans les colonnes de sortie. Les colonnes d'entrée devront en revanche être remplies intégralement.

Table 1

Table de vérité simple

Entrées				Sorties							
en	A[2]	A[1]	A[0]	Y[7]	Y[6]	Y[5]	Y[4]	Y[3]	Y[2]	Y[1]	Y[0]

Table 2

Table de vérité avec priorité

L'entrée *en* peut être considérée comme une entrée prioritaire puisque si *en* vaut 0, toutes les sorties sont à 0 (inactives).

Entrées				Sorties							
en	A[2]	A[1]	A[0]	Y[7]	Y[6]	Y[5]	Y[4]	Y[3]	Y[2]	Y[1]	Y[0]

Table 3

Table de vérité à variables introduites

L'entrée *en* peut être considérée comme une variable introduite.

Entrées			Sorties							
A[2]	A[1]	A[0]	Y[7]	Y[6]	Y[5]	Y[4]	Y[3]	Y[2]	Y[1]	Y[0]
0	0	0	0	0	0	0	0	0	0	en
0	0	1	0	0	0	0	0	0	en	0
0	1	0	0	0	0	0	0	en	0	0
0	1	1	0	0	0	0	en	0	0	0
1	0	0	0	0	0	en	0	0	0	0
1	0	1	0	0	en	0	0	0	0	0
1	1	0	0	en	0	0	0	0	0	0
1	1	1	en	0	0	0	0	0	0	0

Quelle est l'instruction VHDL adaptée pour chaque table de vérité ?

Table 1 : ……

Table 2 : ……

Table 3 : ……

Faites valider les tables de vérité et les questions par un enseignant

3.3. Construction du composant

Ouvrez à nouveau le projet **composants**, et nettoyez le en supprimant le fichier **Mux_4v1.vhd**. Celui-ci restera bien entendu disponible dans le dossier **sources_composants**, mais il ne faut conserver dans un projet que les fichiers utiles au travail en cours. Ajoutez au projet le fichier **Deco_1p8.vhd**.

Choisissez une table de vérité, et appliquez la construction du composant dans le fichier **Deco_1p8.vhd** selon la procédure déjà suivie au 2.3. Si vous avez choisi la table de vérité à variables introduites, il est conseillé de traiter chaque bit de sortie indépendamment avec une instruction à part pour simplifier le code. Pour les autres tables de vérité, il est possible de traiter toutes les sorties en une seule instruction en utilisant un vecteur.

3.1. Simulation du composant

Vérifiez le fonctionnement du composant par simulation. Référez-vous au guide Cyclone ou à la section 2.4 précédente pour les détails de chaque étape.

Sur la simulation, on fera apparaitre tous les comportements combinatoires pertinents :

- Choisissez la valeur d'adresse « 000 » sur *A[2..0] sur plusieurs colonnes dans la simulation*
- Changez la valeur de l'entrée *en* plusieurs fois sur dans la partie où *A[2..0]* vaut « 000 ».
- Répétez l'opération avec toutes les valeurs possibles de l'entrée d'adresse.

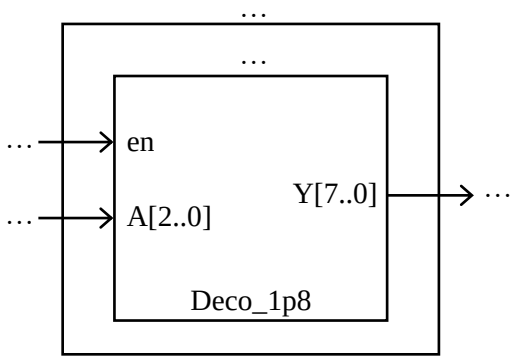
Simulez et vérifiez que le comportement est correct. Si ce n'est pas le cas, modifiez le composant, compilez à nouveau et re-simulez.

Lorsque le comportement est correct, faites noter par un enseignant auquel vous expliquerez la simulation.

3.2. Connexion sur la platine

3.2.1. Schéma

Complétez le schéma de connexion ci-dessous. Faites-le valider par un enseignant

Relations avec la platine	**Schéma de connexion du composant sur la platine**
« ➔ » signifie « est connecté à » en ➔ inter[7] A[2..0] ➔ inter[2..0] Y[7..0] ➔ led[7..0]	

3.2.2. Instanciation du composant sur la platine

Dans le projet platine, appliquez la procédure du 2.5.2. Vous utiliserez le fichier **Platine_Deco_1p8.vhd**.

3.2.3. Test sur la platine

Vérifiez le bon fonctionnement puis faites noter à un enseignant.

TP 2

Décodeur 7 segments
Commande d'un système

Objectifs

- Comprendre le principe d'un décodeur 7 segments
- Mettre en œuvre un système de commande à partir d'un cahier des charges

Fonctions logiques étudiées

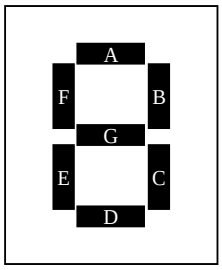
- Décodeur binaire vers 7 segments
- Commande d'un château d'eau

Composants fournis

- *(Aucun)*

1. Fonction décodage binaire vers 7 segments

1.1. Cahier des charges

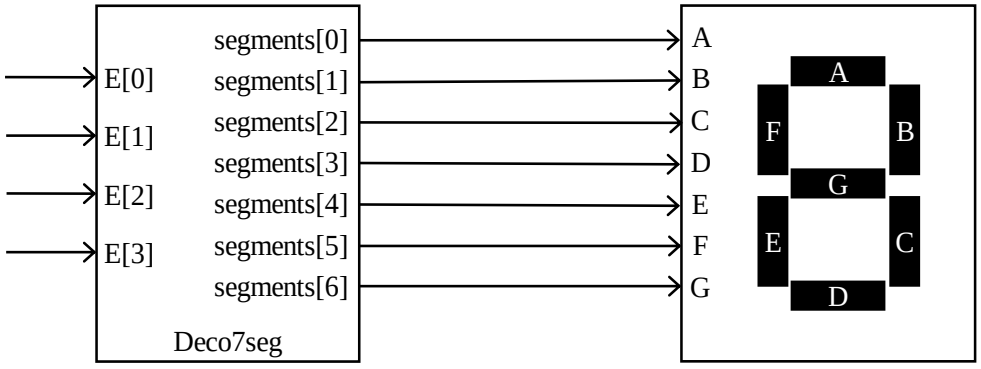


On désire réaliser un système permettant de visualiser sur un afficheur 7 segments le symbole hexadécimal correspondant à un nombre binaire de 4 bits.

Un afficheur 7 segments est constitué d'un ensemble de diodes électroluminescentes (LED) longilignes formant des segments disposés de manière à permettre de former des symboles reconnaissables. Le schéma ci-contre montre la disposition des segments. On peut afficher des formes représentant des chiffres hexadécimaux entre 0x0 et 0xF.

Chaque segment est piloté par une entrée du composant. Pour allumer un segment, il faut appliquer un '1' logique sur l'entrée correspondante, et pour l'éteindre un '0'. Comme on raisonne sur des nombres en binaire naturel (sur 4 bits pour une valeur entre 0x0 et 0xF), il est nécessaire d'effectuer une correspondance entre les différentes valeurs en binaire naturel et les segments devant être allumés ou éteints pour afficher le symbole correspondant. On souhaite réaliser le composant effectuant cette correspondance, que l'on nommera **Deco7seg**.

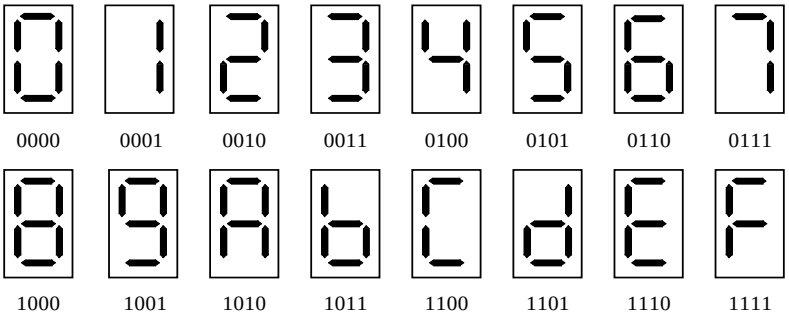
Le schéma du système est le suivant :



Décodeur binaire vers 7 segments

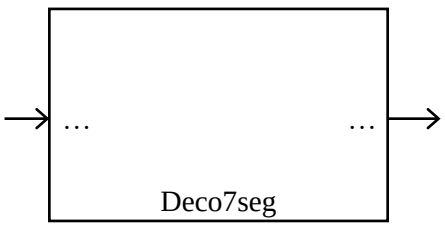
Afficheur 7 segments

La correspondance entre les symboles hexadécimaux à afficher et la valeur binaire présentée à l'entrée du décodeur est la suivante :



1.2. Analyse – Modélisation

Complétez le schéma ci-dessous à partir des informations fournies à côté.

Vue composant du décodeur 7 segments	
Nom du composant : **Deco7seg** Entrés/sorties : E[3..0] : entrées binaires segments[6..0] : sorties vers l'afficheur	

Complétez la table de vérité de la fonction décodage 7 segments.

E[3]	E[2]	E[1]	E[0]	Valeur hexa.	segments[6] G	segments[5] F	segments[4] E	segments[3] D	segments[2] C	segments[1] B	segments[0] A
0	0	0	0	0	0	1	1	1	1	1	1
0	0	0	1	1	0	0	0	0	1	1	0
0	0	1	0	2							

Quelle est l'instruction VHDL adaptée pour cette description ? *Réponse : ……*

Faites valider les 3 questions par un enseignant

1.3. Construction du composant

Ouvrez le projet **composants**. Nettoyez le projet en supprimant les références aux fichiers du TP précédent. Le fichier package reste le même et peut donc être conservé.

Ajoutez ensuite le fichier D**eco7seg.vhd** disponible dans **sources_composants** et appliquez la construction du composant D**eco7seg**. Mettez ce fichier en top level et compilez.

1.1. Simulation du composant

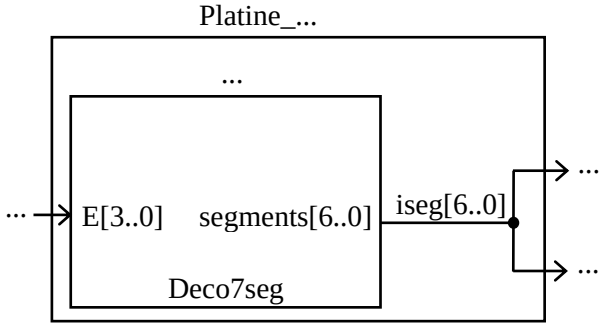
Vérifiez toutes les valeurs possibles de l'entrée *E[3..0]*. Pour chacune d'entre elles, vérifiez que la valeur de sortie correspond bien à la valeur de la table de vérité. Pour cela, il vaut mieux utiliser un affichage en binaire pour la sortie, en choisissant le bon « Radix » pour le signal *segments[6..0]*.

Faites valider votre simulation par un enseignant

1.2. Connexion sur la platine

1.2.1. Schéma

Complétez le schéma ci-dessous représentant la vue composant du décodeur et ses connexions avec la platine en respectant les noms fournis. On utilisera un signal intermédiaire iseg[6..0] *pour réutiliser la sortie. Faites valider le schéma par un enseignant*

Relations avec la platine	**Schéma de connexion du composant sur la platine**
« ➔ » signifie « est connecté à » E[3..0] ➔ inter[3..0] segments[6..0] ➔ iseg[6..0] iseg[6..0] ➔ unite[6..0] iseg[6..0] ➔ led[6..0]	

1.2.2. Instanciation du composant sur la platine

Ouvrez le projet **platine**, et nettoyez le projet : supprimez les fichiers du TP précédent, sauf **sin_pack.vhd** que l'on réutilisera à chaque fois et **cyclone_pack.vhd** qui sera toujours nécessaire. Ajoutez le fichier **Platine_Deco7seg.vhd** dans lequel vous ajouterez votre code, ainsi que le fichier **Deco7seg.vhd**.

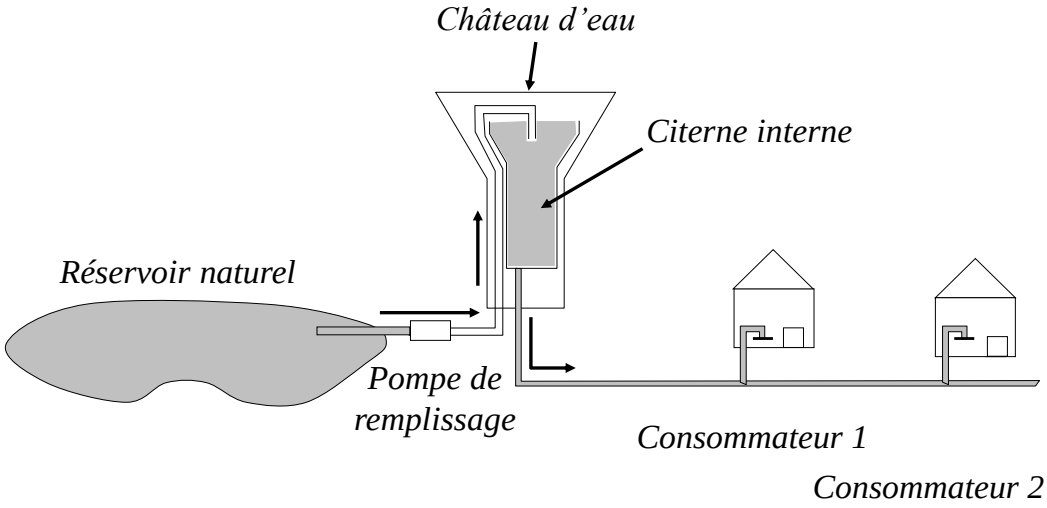
Pensez à déclarer le signal interne *iseg[6..0]* et à mettre en commentaire le code par défaut affectant une valeur aux sorties *unite[6..0]* et *led[6..0]* dans le fichier platine. Mettez le fichier platine en top level, compilez.

1.2.3. Test sur la platine

Vérifiez le bon fonctionnement du composant pour toutes les combinaisons d'entrée et profitez-en pour vérifier que vous savez compter en binaire. Faites ensuite noter à un enseignant .

2. Application de fonctions logiques : Gestion d'un château d'eau

2.1. Principe du château d'eau



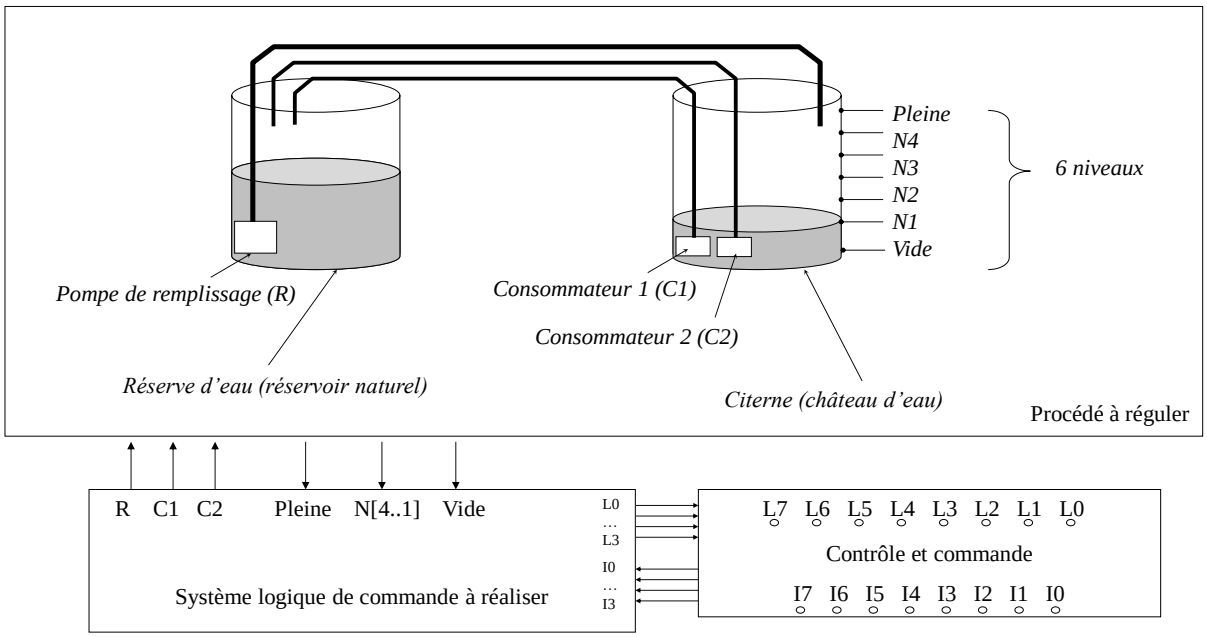
Le service des eaux garantit pour chaque utilisateur une pression d'eau minimale dans les canalisations. Pour cela, on utilise des citernes intermédiaires (châteaux d'eau) permettant de stocker un volume d'eau suffisant pour garantir cette pression minimale, ces châteaux d'eau étant situés en hauteur par rapport aux consommateurs. Une pompe de remplissage permet de prélever l'eau depuis une réserve et de remplir la citerne. Une régulation du niveau de la citerne doit donc être réalisée pour garantir un niveau d'eau minimal.

2.2. Maquette utilisée

Le système que nous allons étudier est basé sur le principe de ce château d'eau. Pour simplifier le procédé, nous avons réalisé la maquette ci-dessous qui fonctionne en circuit fermé. Notre système comporte :

- une pompe de remplissage (*R*), qui alimente la citerne,
- deux pompes (*C1* et *C2*) qui représentent les consommateurs en prélevant de l'eau dans la citerne,
- les informations sur le niveau d'eau de la citerne (*N1* à *N4*, ainsi que *pleine* et *vide*). Ces valeurs sont fournies par un capteur ultrason et traitées par un microcontrôleur. Une information de niveau est à l'état actif en cas de *présence* de liquide à la hauteur correspondante, sauf pour *vide* dont l'état actif indique l'*absence* de liquide.

Tous les capteurs et toutes les commandes de pompes sont actifs à 1



2.3. Cahier des charges

On veut commander la pompe de remplissage de telle sorte que le niveau d'eau reste supérieur au niveau N2 quelle que soit la consommation d'eau. Le système disposera d'une entrée de validation du fonctionnement *en* et d'une entrée de demande de remplissage *dr*, prioritaire sur la régulation de niveau.

Si l'entrée *en* est inactive, le système sera à l'arrêt. Lorsque l'entrée *en* sera active, on distinguera deux cas : le fonctionnement normal (*dr* = '0'), dans lequel on remplira la citerne lorsque le niveau passe en dessous de N2 et on arrêtera la pompe lorsque le niveau repasse au-dessus de N2. L'entrée *N[2]* est active lorsque le niveau est au-dessus du capteur N2. Le fonctionnement forcé (*dr* = '1') activera la pompe quel que soit le niveau, sauf si la citerne est pleine.

Par ailleurs, le système devra également exprimer le niveau de remplissage de la citerne sur la sortie *niveau*, en binaire naturel pour affichage : 0 si le niveau est en-dessous de N1, 1 si le niveau est au-dessus de N1, 2 s'il est au-dessus de N2, 3 au-dessus de N3 et 4 au-dessus de N4. On utilisera un signal sur 4 bits pour faciliter la connexion ultérieure du signal.

2.4. Analyse – Modélisation

À partir du cahier des charges décrit ci-dessus, on veut établir les tables de vérité des fonctions à réaliser. Pour cela, on cherche à déterminer la commande de *R* et *niveau* en fonction des entrées *en*, *dr* et des capteurs.

Dessinez la vue composant du bloc à réaliser. Faites valider le schéma par un enseignant

Vue composant du système de commande	
Nom du composant : **Cmd_chateau** Entrées : en : entrée de validation dr : demande de remplissage N[4..1] : capteurs de niveau pleine : capteur de saturation Sorties : R : Commande pompe niveau[3..0] : Affichage du niveau	… … … … … … … … …

On constate que toutes les combinaisons d'entrée ne sont pas pertinentes pour le cahier des charges. Il n'est donc pas nécessaire de les décrire toutes.

Pour réduire le nombre de combinaisons et donc de lignes de la table de vérité, nous allons utiliser deux critères :

1) Les fonctions peuvent être indépendantes :
Cette notion permet de diviser le problème en plusieurs tables de vérité indépendantes. Chaque table nécessite alors moins de variables.
Du fait de l'indépendance des fonctions, nous pouvons diviser le problème en deux tables :
- la première concerne la commande de la pompe de remplissage *R.*
- l'autre concerne la commande de la sortie *niveau*.

2) La priorité entre les variables :
La prise en compte de certaines variables ou combinaisons des variables permet de ne pas considérer les autres variables ou leurs combinaisons.

2.4.1. Commande de la pompe de remplissage *R*

Nous allons identifier tous les cas d'utilisation significatifs et nous allons aussi classer les variables pour tenir compte de la priorité.
La réalisation doit tenir compte de l'entrée *en*, qui autorise le fonctionnement normal lorsqu'elle est à 1, et éteint la pompe à 0. L'activation de la demande de remplissage prioritaire active la pompe quel que soit l'état des capteurs de niveau, sauf si la citerne est pleine. Enfin, en fonctionnement normal, la pompe s'active dès que le niveau de liquide passe *en dessous* du niveau N2.

Complétez la table de vérité de régulation du niveau d'eau.

Description des cas d'utilisation	en	dr	pleine	N[2]	R
Arrêt du système	0	X	X	X	
Demande de remplissage prioritaire	1	1	0	X	
	1	1	1	X	
Régulation de niveau automatique	1	0	X	0	
	1	0	X	1	

Quelle est l'instruction VHDL adaptée pour cette description ? *Réponse : ……..*

Faites valider la table de vérité et la question par un enseignant

2.4.2. Commande de la sortie *niveau*

La commande de la sortie *niveau* peut être représentée par une table de vérité incomplète : par exemple, la situation dans laquelle *N[1]* serait inactif alors que *N[2]* serait actif est normalement impossible : si le niveau d'eau est au-dessus du capteur N2, elle est forcément au-dessus du capteur N1.

Complétez la table de vérité de la sortie niveau.

N[4]	N[3]	N[2]	N[1]	niveau[3..0]
0	0	0	0	
0	0	0	1	
0	0	1	1	
0	1	1	1	
1	1	1	1	

Quelle est l'instruction VHDL adaptée pour cette description ? *Réponse : ……..*

Faites valider la table de vérité et la question par un enseignant

2.5. Construction du composant

Appliquez la construction du composant en utilisant le fichier **Cmd_chateau.vhd**.

2.1. Simulation du composant

Simulez le composant en faisant apparaître toutes les valeurs des tables de vérité. Procédez l'une après l'autre sur la simulation, en séparant clairement celles-ci temporellement, et en ajoutant des commentaires textuels sur la simulation au moyen de l'outil adéquat (voir le guide cyclone pour plus de détails sur les outils).

Faites valider la simulation par un enseignant

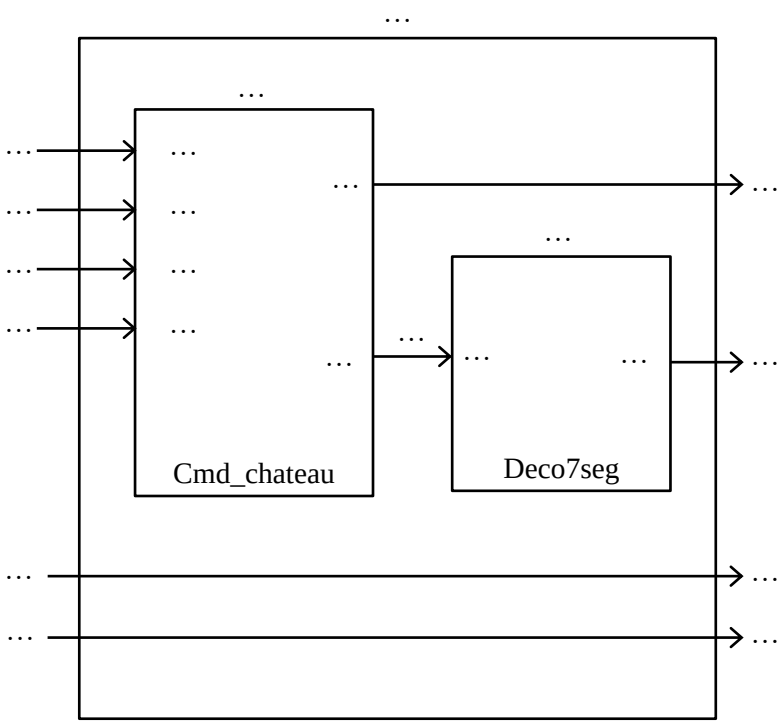
2.2. Connexion sur la platine

Pour connecter le composant sur la platine, nous devons réaliser plusieurs choses :

- La connexion de la fonction *R* à la pompe de remplissage,
- L'affichage de la sortie *niveau* sur un afficheur 7 segments via un décodeur,
- La commande des pompes consommateurs *C1* et *C2* à partir des interrupteurs.

2.2.1. Schéma

Complétez la représentation de l'instanciation du système sur la platine ci-dessous. On utilisera un signal intermédiaire iaff *pour connecter le composant de commande au décodeur. Les pompes consommateurs seront reliées directement aux interrupteurs. Faites valider le schéma par un enseignant*

Relations avec la platine	Schéma de connexion du composant sur la platine
Composant **Cmd_chateau** : en ➔ inter[0] dr ➔ inter[1] N[4..1] ➔ capt[4..1] pleine ➔ capt[5] R ➔ sortie[2] niveau[3..0] ➔ iaff[3..0] Composant **Deco7seg** : E[3..0] ➔ iaff[3..0] segments[6..0] ➔ unite[6..0] Pompes consommateurs : inter[7] ➔ sortie[1] (C2) inter[6] ➔ sortie[0] (C1)	

2.2.2. Instanciation du composant sur la platine

Utilisez le fichier **Platine_Cmd_chateau.vhd**.

2.2.3. Test sur la platine

Branchez la maquette du château d'eau à l'alimentation et au connecteur de sortie de la platine, puis programmez la platine.

Vérifiez le bon fonctionnement du composant puis faites noter à un enseignant.

TP 3

Additionneurs

Objectifs

- Décomposer une opération mathématique et l'exprimer sous la forme d'une table de vérité
- Comprendre la composition d'un système par assemblage de composants simples

Fonctions logiques étudiées

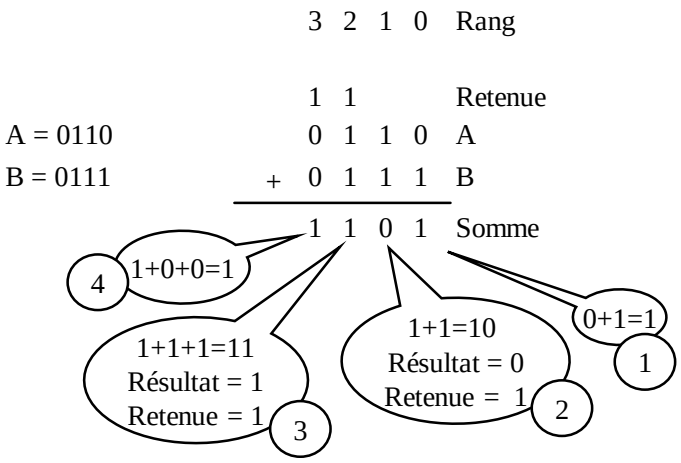
- Additionneur 1 bit
- Additionneur 4 bits

Composants fournis

- *(Aucun)*

1. Principe de l'additionneur

Une addition binaire fonctionne de la même manière qu'une addition décimale. Si l'on souhaite additionner A et B, on procèdera par l'addition les chiffres de même rang de chacun des nombres. Chacune de ces additions peut produire une retenue, qui sera alors additionnée avec le rang supérieur.

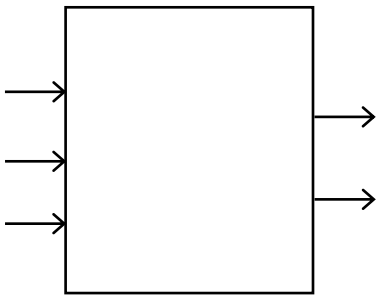


À chaque rang, le principe est donc le même : ajouter les deux chiffres de ce rang ainsi que la retenue si elle existe, et produire le résultat du rang et une éventuelle retenue. Il est donc possible de réaliser un composant faisant l'addition d'un seul bit, puis de mettre en cascade plusieurs de ces composants pour réaliser une addition sur plusieurs bits. Il suffit pour cela de prendre en compte les retenues, qui feront le lien entre deux rangs. On appelle un tel additionneur prenant en compte les retenues un additionneur complet.

2. Additionneur complet 1 bit

2.1. Cahier des charges

Complétez la vue composant de l'additionneur 1 bit. Faites valider le schéma par un enseignant

Vue composant de l'additionneur complet 1 bit	
Nom du composant : **Add_1bit** Entrées/sorties : A et B : opérandes Rin : retenue entrante S : résultat Rout : retenue produite	

2.2. Analyse – Modélisation

Complétez la table de vérité de ce composant. Faites-la valider par un enseignant

Rin	A	B	Rout	S	Interprétation de l'opération
0	0	0	0	0	0 + 0 + 0 = 00
0	0	1	0	1	0 + 0 + 1 = 01
0	1	0			0 + 1 + 0 = …
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

2.3. Construction du composant

Construire le composant dans le fichier **Add_1bit.vhd** en mettant en œuvre la table de vérité de la page précédente.

Il est possible d'utiliser une instruction ***with…select*** pour réaliser cette table de vérité simple. Néanmoins, l'instruction ***with…select*** s'applique sur un vecteur, or nous avons ici trois entrées indépendantes. Il est donc nécessaire de créer un signal intermédiaire concaténant les trois bits en un seul vecteur. Pour cela, déclarer un signal interne de type vecteur avant le ***begin*** du composant, puis lui affecter la concaténation des trois entrées après le ***begin***. Attention, l'ordre de la concaténation doit respecter l'ordre des entrées dans la table de vérité !

Les deux sorties étant indépendantes, on réalisera une instruction ***with…select*** pour chacune d'entre elles.

Pour les plus avancés d'entre vous : il est possible d'utiliser une seule instruction ***with…select***, voyez-vous comment faire ?

2.1. Simulation du composant

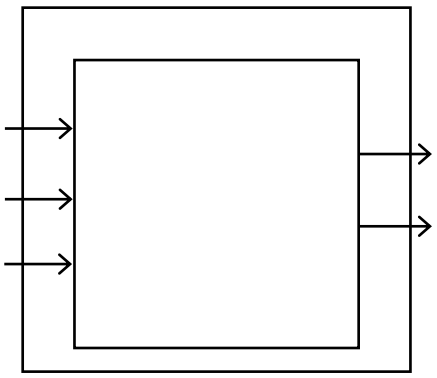
Simulez le composant en faisant apparaître toutes les combinaisons possibles des entrées et en vérifiant que la somme est correcte à chaque fois. Faites valider la simulation par un enseignant

2.2. Connexion sur la platine

2.2.1. Schéma

Les valeurs des opérandes *A* et *B* et de la retenue entrante *Rin* seront commandées par les interrupteurs. On visualisera les sorties *S* et *Rout* sur les LEDs.

Complétez le schéma de connexion ci-dessous. Faites-le valider par un enseignant

Relations avec la platine	**Schéma de connexion du composant sur la platine**
B ➔ inter[0] A ➔ inter[1] Rin ➔ inter[2] S ➔ led[0] Rout ➔ led[1]	

2.2.2. Instanciation du composant sur la platine

Utilisez le fichier **Platine_Add_1bit.vhd**.

2.2.3. Test sur la platine

Vérifiez le bon fonctionnement du composant puis faites noter à un enseignant.

3. Additionneur complet 4 bits

3.1. Cahier des charges

On désire réaliser maintenant un additionneur de 2 nombres de 4 bits chacun. On conserve la logique de l'additionneur 1 bit, à savoir que l'on souhaite pouvoir mettre en cascade plusieurs additionneurs de 4 bits. Ceux-ci doivent donc comporter une entrée et une sortie dédiées aux retenues.

Les entrées sont donc :

- *A [3..0]* et *B[3..0]* les opérandes
- *Rin* la retenue entrante

Et les sorties :

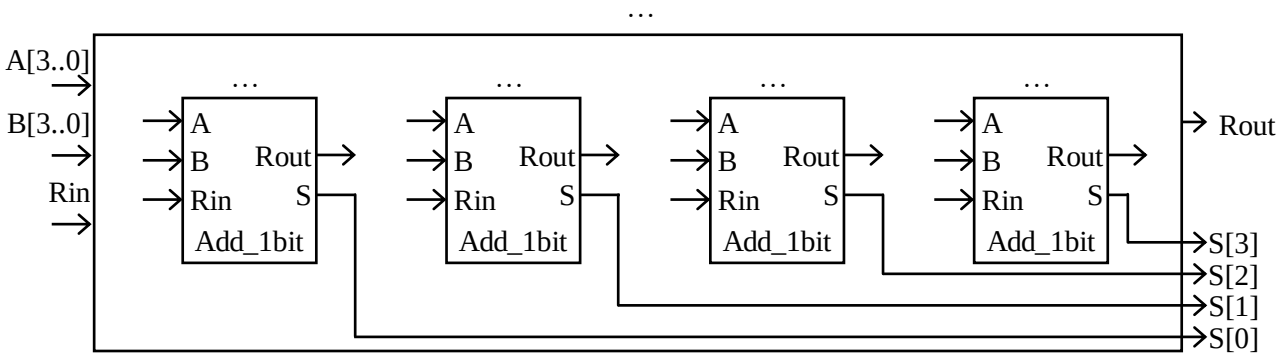
- *S[3..0]* le résultat
- *Rout* la retenue sortante

3.2. Analyse – Modélisation

Comme on l'a vu précédemment, on réalisera cet additionneur en mettant en cascade 4 additionneurs complets de 1 bit chacun.

Le modèle est toujours combinatoire car nous allons le construire par connexion de composants combinatoires sans rebouclage.

Sachant que l'addition se fait en partant des bits de poids faible vers les bits de poids fort comme détaillé au début du TP, complétez le schéma ci-dessous avec toutes les informations nécessaires à la traduction directe de celui-ci en VHDL. Vous pouvez utiliser la connexion par noms si vous le souhaitez. On rappelle que, comme pour les entrées et les sorties, tous les signaux internes doivent être nommés. On nommera le signal intermédiaire iret[2..0]. Sur le schéma, on placera le nom directement sur le fil.



Faites valider le schéma par un enseignant

3.3. Construction du composant

Complétez le fichier **Add_4bits.vhd** pour correspondre au schéma ci-dessus.

Il se résume en une connexion de composants en VHDL. Vous aurez néanmoins besoin de déclarer des signaux internes pour les connexions directes entre les composants.

Nous utilisons cette fois-ci un composant : il est donc nécessaire d'inclure la bibliothèque **sin_pack** dans le fichier. Pour cela, procédez comme vous le faites habituellement dans le fichier platine en ajoutant les deux lignes correspondantes en haut du fichier.

3.4. Simulation du composant

Testez plusieurs valeurs sur votre simulation. Pour chaque valeur, vérifier que le résultat est correct. On pourra pour cela afficher les valeurs en décimal sur la simulation pour simplifier la vérification.

Faites valider la simulation par un enseignant

3.5. Connexion sur la platine

3.5.1. Schéma

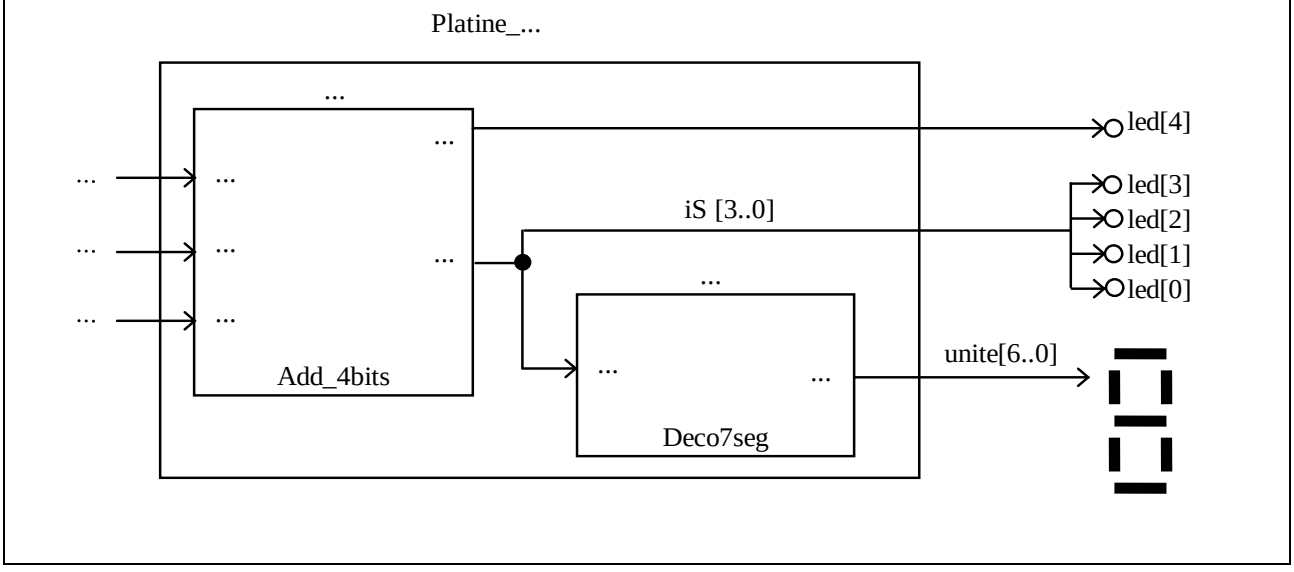
On souhaite tester l'additionneur en visualisant le résultat sur des LEDs et en même temps sur un afficheur 7 segments. Pour cela, on utilisera deux composants :

- l'additionneur, **Add_4bits**
- le décodeur 7 segments, **Deco7seg**

On aura également besoin d'un signal intermédiaire, *S_i* qui permettra de brancher la sortie *S* du composant à deux signaux différents, pour visualiser le résultat à la fois sur les LEDs et l'afficheur 7 segments.

Complétez le schéma suivant en respectant les connexions indiquées ci-dessous. On distingue trois groupes de connexions : celles du composant additionneur, celles du décodeur 7 segments et les signaux intermédiaires utilisés pour faire des liens supplémentaires. Faites valider le schéma par un enseignant

Relations avec la platine		
Composant : **Add_4bits** Connexions : A[3..0] ➔ inter[7..4] B[3..0] ➔ inter[3..0] Rin ➔ int[0] Rout ➔ led[4] S[3..0] ➔ iS[3..0]	Composant : **Deco7seg** Connexions: E[3..0] ➔ iS[3..0] segments[6..0] ➔ unite[6..0]	Connexions de signaux internes : iS[3..0] ➔ led[3..0]
Schéma de connexion du composant sur la platine		



3.5.2. Instanciation du composant sur la platine

Appliquez la connexion sur la platine en utilisant le fichier **Platine_Add_4bits.vhd**. Le fichier **Deco7seg.vhd** est disponible sur Moodle.

3.5.3. Test sur la platine

Faites noter à un enseignant.