Travaux pratiques 1

Modélisation et simulation d'un additionneur

IESE5/MISTRE – Polytech Grenoble

Objectif

Ce TP a pour but d'illustrer les notions d'entité, architecture, et configuration à travers la description d'un circuit faisant l'addition de deux nombres codés en binaire naturel sur 1, 2 puis 8 bits.

Outils

— **nedit/gedit/vim** : éditeur de texte utilisé pour la création et édition des descriptions VHDL des circuits.
— **ModelSim** : environnement de simulation et débogage utilisé pour la vérification du comportement des circuits.

1 Environnement de travail

Chaque TP sera considéré comme un projet structuré en un ensemble de sous-répertoires. Chaque répertoire contient des fichiers d'une même catégorie.
Attention : cette organisation sera répétée pour chaque TP.

— **Bench** : répertoire des fichiers sources des testbench (bancs de test) :
  — Entités et architectures des testbench (`.vhd`)
  — Configurations (`.vhd`)
  — Script (`compile_bench.sh`) pour la compilation des testbench

— **Confmodelsim** : répertoire des fichiers de configuration de Modelsim :
  — `.bashrc_modelsim10_2c`
  — `config`
  — `modelsim.ini`

— **Libs** : répertoire des bibliothèques des fichiers VHDL compilés (au début du TP ce répertoire est vide). Après la compilation du TP1, ce répertoire doit contenir les bibliothèques :
  — `LIB_ADDER` : bibliothèque des fichiers issus du répertoire Sources
  — `LIB_ADDER_BENCH` : bibliothèque des fichiers issus du répertoire Bench

— **Sources** : répertoire des fichiers sources :
  — Entités et architectures des composants du projet (`.vhd`)
  — Script (`compile_sources.sh`) pour la compilation des composants

— **Synth** : répertoire des fichiers après synthèse (ce répertoire n'est pas utilisé dans ce TP)

Question 1.1. Avez-vous tous les sous-répertoires décrits précédement ?
Ouvrez un terminal, allez dans le répertoire crée pour le TP1 et vérifiez l'existance des répertoires et fichiers :

```
$ cd ~/tp_vhdl_msn/TP1
$ ls
```

Question 1.2. Pouvez-vous exécuter les fichiers de configuration de votre environnement ?
Attention : Exécutez les opérations suivantes dans chaque terminal que vous ouvrez afin de le configurer.

```
$ cd ~/tp_vhdl_msn/TP1/Confmodelsim
$ source .bashrc_modelsim10_2c
$ source config
```

Question 1.3. Pouvez-vous expliquer à quoi sert chacune des commandes exécutées précédemment ?

2 Prise en main des outils : l'additionneur complet (Full Adder - Dataflow)

Une première solution pour créer le `full_adder` est de l'écrire dans le style dataflow (Figure 1).

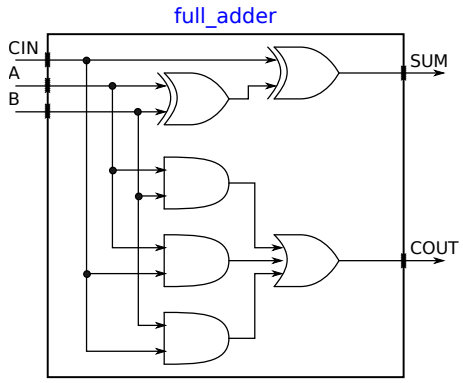

Figure 1 – L'additionneur complet (Full Adder - Dataflow)

⇒ **Consigne 1 :** Vérifiez que le répertoire `Sources` contient les fichiers présentés ci-dessous.

- **`full_adder.vhd`** : contenant la définition de l'entité `full_adder` et de chacun de ses ports.
- **`full_adder_dataflow.vhd`** : contenant la définition d'une architecture `dataflow` pour l'entité `full_adder`.
- **`compile_sources.sh`** : script qui réinitialise la *bibliothèque des sources* du projet et compile chaque fichier source `.vhd` situé dans le répertoire courant. Pour le TP1, la bibliothèque est appelée `LIB_ADDER`.

Attention : Tout nouveau fichier *source* qui sera rajouté au projet doit être compilé en utilisant ce script (c.-à-d. doit être rajouté dans ce script).

Exemple : `vcom -work LIB_ADDER full_adder.vhd`

Question 2.1. Pouvez-vous compiler les sources à l'aide du script `compile_sources.sh` ?
Placez-vous dans le répertoire `Sources`, donnez les droits d'exécution au fichier `compile_sources.sh` et lancez la compilation.

```
$ cd ~/tp_vhdl_msn/TP1/Sources
$ chmod u+x compile_sources.sh
$ ./compile_sources.sh
```

⇒ **Consigne 2 :** Vérifiez que le répertoire `Bench` contient les fichiers présentés ci-dessous.

— **`bench_full_adder.vhd`** : contenant la définition de l'entité `bench_full_adder` et la définition de son architecture `test`.

— **`config_bench_full_adder_dataflow.vhd`** : contenant la définition d'une configuration (couple entité/architecture = `full_adder/dataflow`) pour l'entité `bench_full_adder`.

— **`compile_bench.sh`** : script qui réinitialise la *bibliothèque des testbenches* du projet et compile chaque fichier *bench* ou *config* `.vhd` situé dans le répertoire courant. Pour le TP1, la bibliothèque des testbenches est appelée `LIB_ADDER_BENCH`.

Attention : Tout nouveau fichier *bench* ou *config* qui sera rajouté au projet doit être compilé en utilisant ce script.

Exemple : `vcom -work LIB_ADDER_BENCH bench_full_adder.vhd`

Question 2.2. Pouvez-vous compiler les testbenches et les configurations à l'aide du script `compile_bench.sh` ? Placez-vous dans le répertoire `Bench`, donnez les droits d'exécution au fichier `compile_bench.sh` et lancez la compilation.

```
$ cd ~/tp_vhdl_msn/TP1/Bench
$ chmod u+x compile_bench.sh
$ ./compile_bench.sh
```

3 Prise en main des outils : Simulation avec ModelSim

Pour simuler votre modèle avec ModelSim, vous devez suivre les instructions présentées ci-dessous :

— Tapez dans un terminal la commande permettant de démarrer le simulateur :

```
$ vsim&
```

— Attendez l'ouverture de la fenêtre de la Figure 2.

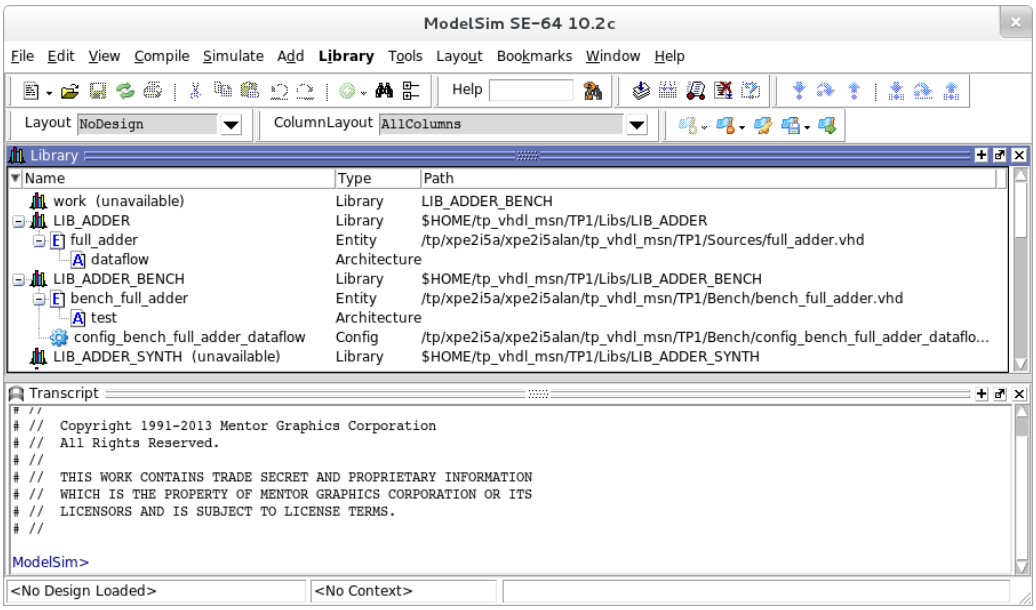

Figure 2 – ModelSim SE-64 10.2c

— Cliquez sur `Simulate` → `Start Simulation`

— Choisissez la configuration du testbench que l'on vient de compiler. Il s'agit de prendre : `config_bench_full_adder_dataflow` (Figure 3).

Figure 3 – Sélection de la configuration full_adder/dataflow

— Sélectionnez les signaux désirés (`shift + click gauche`) et ajoutez-les dans le chronogramme (`click droite` → `add wave`). (Figure 4)

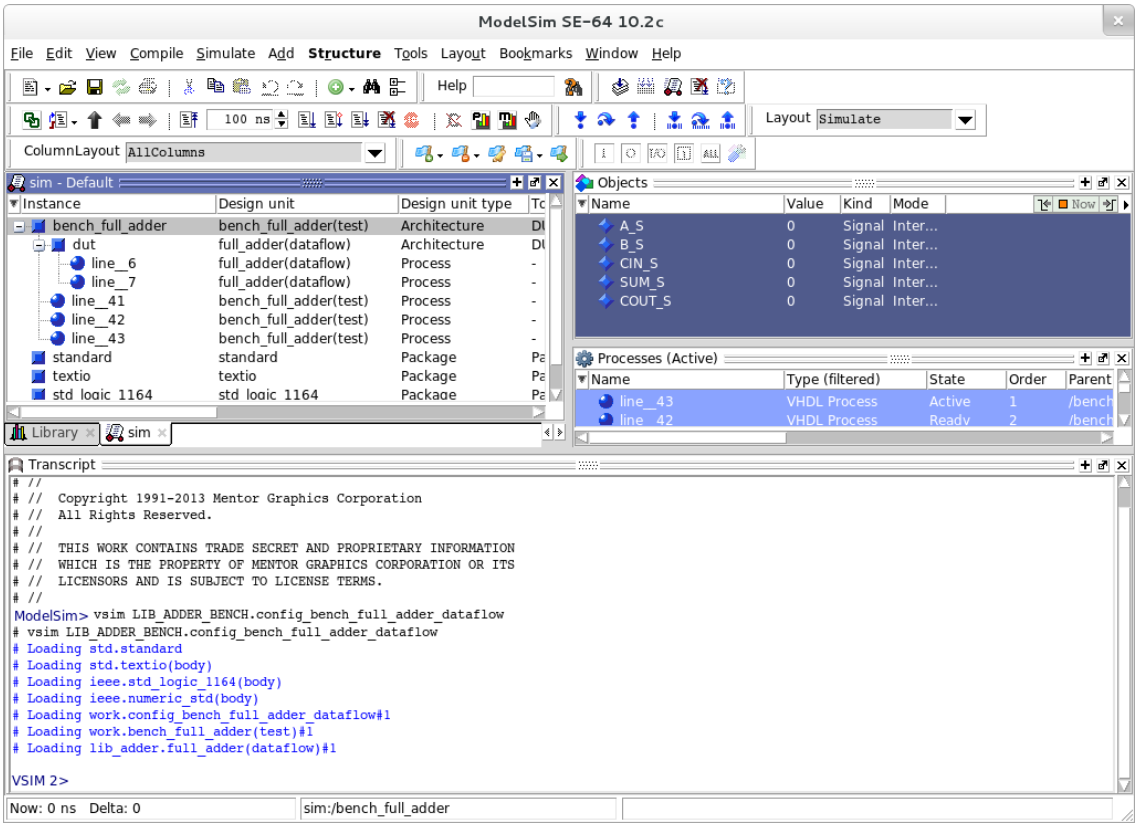

Figure 4 – Sélection des signaux

— Lancez la simulation en utilisant le bouton associé et le champ de choix d'un temps (Figure 5)

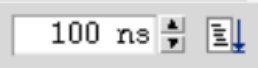

Figure 5 – Sélection du temps de simulation

Attention : Le temps de simulation doit absolument être saisi faute de quoi le simulateur bouclera à l'infini, ce qui peut provoquer un plantage du programme.

— Une fois la simulation exécutée, vous obtenez le résultat de la Figure 6.

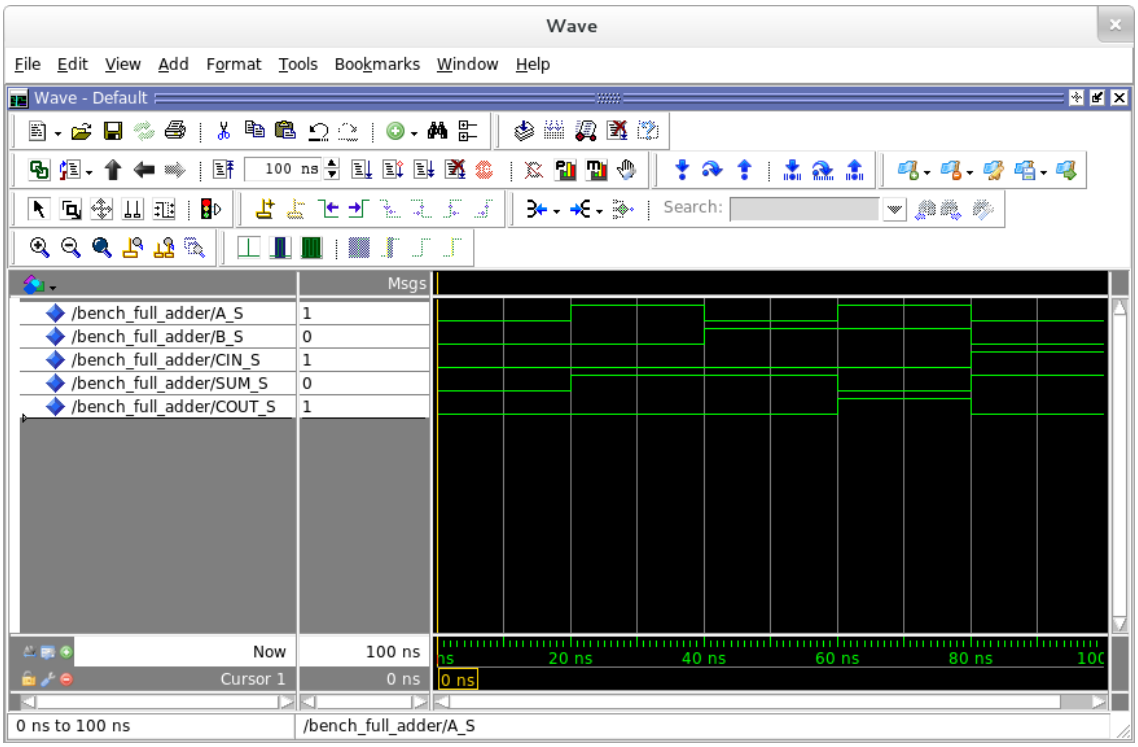

Figure 6 – Résultat de la simulation de l'additionneur complet (Full Adder – Dataflow)

Question 3.1. Confirmez-vous que le résultat sur les signaux de sortie est conforme à ce qui était attendu ? Pourquoi ?

4 Description structurelle : Additionneur complet (Full Adder - Structural)

Une seconde solution pour créer le `full_adder` est de le décomposer en deux `half_adder` (Figure 7). Ceux-ci mis en série (via la sortie `S`) permettent de générer la sortie `SUM` du `full_adder`. La conjonction des retenues des deux `half_adder` permet de générer la retenue `COUT` du `full_adder`.

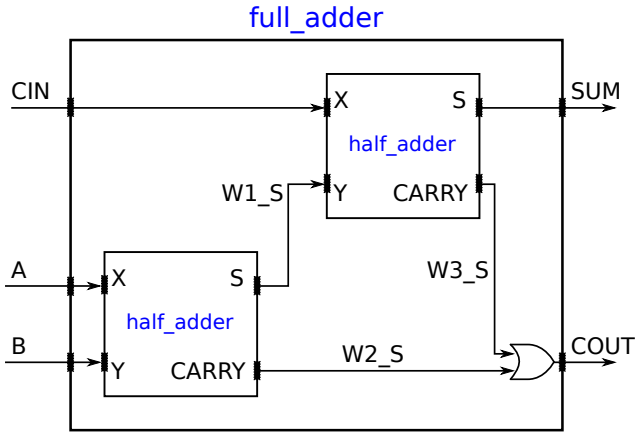

Figure 7 – Additionneur complet (Full Adder – Structural)

Question 4.1. Expliquez succinctement, en utilisant l'algèbre de Boole, comment est-il possible d'obtenir les équations du `full_adder` en assemblant deux `half_adder`.

Question 4.2. Modélisez le composant demi-additionneur (Half Adder – Dataflow) et vérifiez son comportement par simulation.

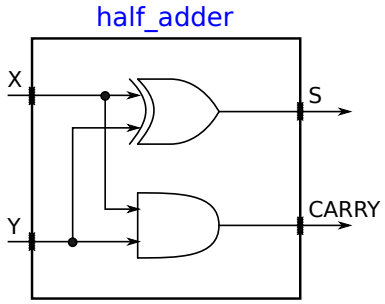

Figure 8 – Demi-additionneur (Half Adder)

— **Créez l'entité et l'architecture** du demi-additionneur (Figure 8) :
  — Dans le répertoire `~/tp_vhdl_msn/TP1/Sources`, créez un fichier `half_adder.vhd` et définissez l'entité et l'architecture du composant `half_adder`.
  — Modifiez le script `compile_sources.sh` afin de pouvoir compiler le fichier `half_adder.vhd`
  — Exécutez le script (`./compile_sources.sh`) et vérifiez l'absence d'erreur.

— **Créez l'entité test** du demi-additionneur :
  — Dans le répertoire `~/tp_vhdl_msn/TP1/Bench`, créez un fichier `bench_half_adder.vhd` et définissez l'entité et l'architecture du testbench `bench_half_adder`.
  — Modifiez le script `compile_bench.sh` afin de pouvoir compiler le fichier `bench_half_adder.vhd`
  — Exécutez le script (`./compile_bench.sh`) et vérifiez l'absence d'erreur.
  — Lancez ModelSim, simulez et vérifiez le bon fonctionnement du composant `half_adder`.

Question 4.3. Modélisez l'additionneur complet dans le style structurel (Full Adder – Structural) et vérifiez son comportement par simulation.

Attention : Vous disposez déjà de l'entité `full_adder` et du test `bench_full_adder`

— **Créez l'architecture structurel** de l'additionneur complet (Figure 7) :
  — Dans le répertoire `~/tp_vhdl_msn/TP1/Sources`, créez un fichier `full_adder_structural.vhd` et définissez une nouvelle architecture structural pour le composant `full_adder` (déjà défini dans le fichier `full_adder.vhd`). Vous pouvez vous inspirer de l'architecture vue en cours, attention les noms des ports sont différents !
  — Modifiez le script `compile_sources.sh` afin de pouvoir compiler le fichier `full_adder_structural.vhd`
  — Exécutez le script (`./compile_sources.sh`) et vérifiez l'absence d'erreur.

— **Créez une nouvelle configuration** (`full_adder/structural`) pour l'additionneur complet :
  — Dans le répertoire `~/tp_vhdl_msn/TP1/Bench`, créez un fichier `config_bench_full_adder_structural.vhd` et définissez une nouvelle configuration, c'est-à-dire, un nouveau couple `full_adder/structural` pour l'entité `bench_full_adder`.
  — Modifiez le script `compile_bench.sh` afin de pouvoir compiler le fichier `config_bench_full_adder_structural.vhd`
  — Exécuter le script (`./compile_bench.sh`) et vérifier l'absence d'erreur.
  — Lancez ModelSim, simulez et vérifiez le bon fonctionnement du composant `full_adder` pour les différentes configurations.

Question 4.4. Validez l'équivalence de deux architectures écrites pour l'additionneur complet (`dataflow` vs. `structural`)

— Dans le répertoire `~/tp_vhdl_msn/TP1/Bench`, créez un fichier `bench_test_equiv.vhd` et définissez l'entité et l'architecture du testbench. Celui-ci doit utiliser le principe de la configuration incorporée décrit au chapitre II.3 du cours, c'est-à-dire qu'il doit instancier deux composants `P1` et `P2` du `full_adder` :
  — `P1` est associé à l'architecture `dataflow`.
  — `P2` est associé à l'architecture `structural`.

— Modifiez le script `compile_bench.sh` afin de pouvoir compiler le fichier `bench_test_equiv.vhd`

— Exécutez le script (`./compile_bench.sh`) et vérifiez l'absence d'erreur.

— Lancez ModelSim, simulez et vérifiez le bon fonctionnement : comparez les sorties de `P1` et `P2` et observez qu'elles sont identiques.

5 L'additionneur 2 bits (Two Bit Adder)

L'additionneur 2 bits est réalisé par la mise en série de deux `full_adder` (Figure 9)

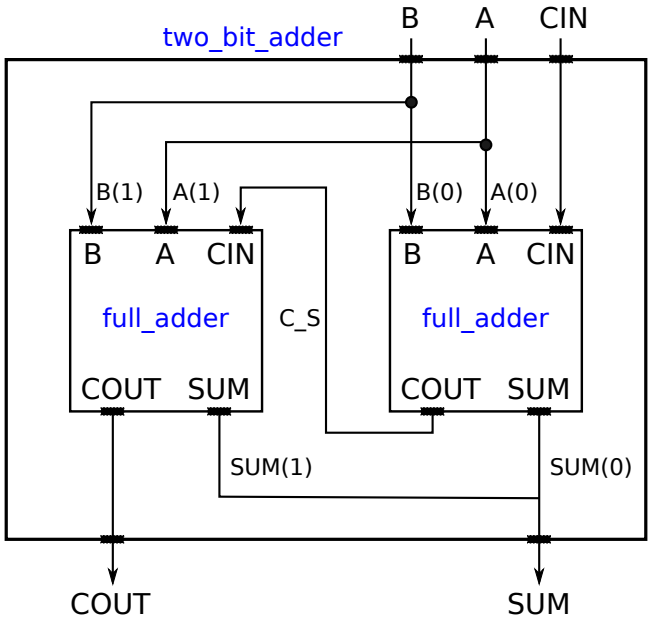

Figure 9 – Additionneur 2 bits (Two Bit Adder)

Question 5.1. Modélisez l'additionneur 2 bits et vérifiez son comportement par simulation.

— **Créez l'entité et l'architecture** de l'additionneur 2 bits (Figure 9) :

  — Dans le répertoire `~/tp_vhdl_msn/TP1/Sources`, créez un fichier `two_bit_adder.vhd` et définissez l'entité `two_bit_adder` et son architecture `structural`. Les entrées `A` et `B`, et la sortie `SUM` sont des vecteurs de 2 bits.

  — Modifiez le script `compile_sources.sh` afin de pouvoir compiler le fichier `two_bit_adder.vhd`

  — Exécutez le script (`./compile_sources.sh`) et vérifiez l'absence d'erreur.

— **Créez l'entité test et deux configurations** pour l'additionneur 2 bits :

  — Dans le répertoire `~/tp_vhdl_msn/TP1/Bench`, créez un fichier `bench_two_bit_adder.vhd` et définissez l'entité `bench_two_bit_adder` et son architecture test (banc des tests).

  — Dans le répertoire `~/tp_vhdl_msn/TP1/Bench`, créez un fichier `config_bench_two_bit_adder_dataflow.vhd` et définissez une configuration pour `bench_two_bit_adder`, qui associe les couples entité/architecture suivants :

    — Entité `two_bit_adder` avec l'architecture `structural` :
    `use entity LIB_ADDER.two_bit_adder(structural)`

    — Entité `full_adder` (instantiée par `two_bit_adder`) avec l'architecture `dataflow` :
    `use entity LIB_ADDER.full_adder(dataflow)`

  — Dans le répertoire `~/tp_vhdl_msn/TP1/Bench`, créez un fichier `config_bench_two_bit_adder_structural.vhd` et définissez une nouvelle configuration pour `bench_two_bit_adder`, qui associe les couples entité/architecture suivants :

  - Entité `two_bit_adder` avec l'architecture `structural` :
    `use entity LIB_ADDER.two_bit_adder(structural)`
  - Entité `full_adder` (instantiée par `two_bit_adder`) avec l'architecture `structural` :
    `use entity LIB_ADDER.full_adder(structural)`

- Modifiez le script `compile_bench.sh` afin de pouvoir compiler les fichiers :
  - `bench_two_bit_adder.vhd`
  - `config_bench_two_bit_adder_dataflow.vhd`
  - `config_bench_two_bit_adder_structural.vhd`
- Exécutez le script (`./compile_bench.sh`) et vérifiez l'absence d'erreur.
- Lancez ModelSim, simulez et vérifiez le bon fonctionnement du composant `two_bit_adder` pour les différentes configurations.

6 L'additionneur N bits

Question 6.1. Concevez et testez un additionneur N bits réalisé par la mise en série de N additionneurs 1 bit en utilisant l'opérateur **`for ... generate`** (chapitre III.2.4 du cours)

7 L'additionneur 8 bits (Eight Bit Adder)

Question 7.1. En utilisant l'additionneur générique de N bits, concevez un nouveau banc de test dans lequel vous devez instancier un additionneur 8 bits. Testez et validez votre composant.