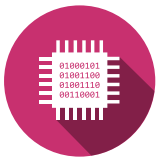


Compteur de programme

Laboratoire Conception Numérique

Contenu

1 Objectifs

Ce laboratoire présente le développement d'un code de programme à l'aide d'une mémoire morte (Read-Only Memory (ROM)) et exerce la réalisation d'un compteur de programme ROM.

Il exerce aussi le dessin de circuits hiérarchiques.

2 ROM de commande de l'ALU

2.1 Circuit

La Fig. 1 présente la vue simplifiée d'un processeur, avec une Arithmetic and Logical Unit (ALU), des registres et un compteur de programme.

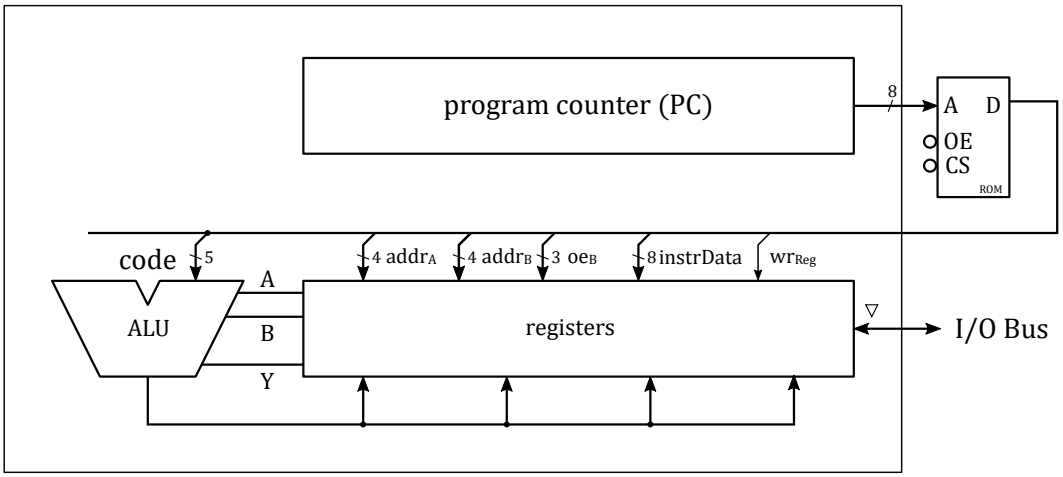


Fig. 1 - ROM contrôlant l'ALU et les registres

L'adresse de la ROM est générée par un compteur de programme, ce qui permet de lire le code stocké de manière séquentielle. Les données de la ROM contiennent des Machine Instructions (MIs) qui se composent de signaux de commande pour l'ALU et les registres.

L'entrée Output Enable (OE) contrôle la sortie à haute impédance de la ROM. L'entrée Chip Select (CS) est le signal de sélection de la ROM. Les deux doivent être actifs pour que le composant fournisse ses données en sortie.

2.2 Séquencement des opérations

Les instructions sont exécutées en 2 phases, comme illustré dans Fig. 2.

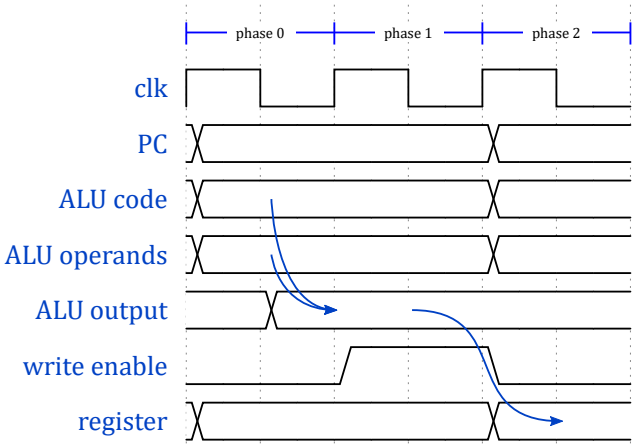
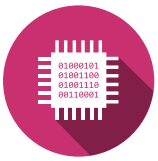


Fig. 2 - Phases des instructions

La sortie de l'ALU est stable en fin de phase 0. La nouvelle valeur du registre sélectionné est enregistrée au flanc montant de l'horloge en fin de phase 1.



3 Réalisation logicielle d'un port série

La Fig. 3 présente le déroulement temporel de l'envoi en série d'un mot de donnée.

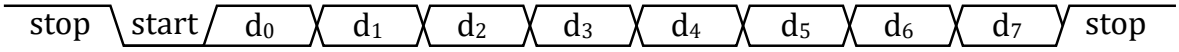


Fig. 3 - Transmission sérielle

Le signal sériel est transmis sur le bit de poids faible (Least Significant Bit (LSB)) du port de sortie.

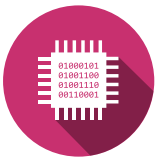
3.1 Algorithme linéaire

Le code Assembler stocké dans la ROM est présenté dans Liste 1.

```
LOAD        s3, FF              ; load stop bit
OUTPUT      s3                  ; output stop bit
LOAD        s3, s3              ; no operation
LOAD        s3, s3              ; no operation
LOAD        s3, s3              ; no operation
LOAD        s3, s3              ; no operation
LOAD        s0, 00              ; load start bit
OUTPUT      s0                  ; output start bit
INPUT       s1                  ; load word to send
OUTPUT      s1                  ; output word, LSB is considered
SR0         s1                  ; shift word, bit 1 -> LSB
OUTPUT      s1                  ; output bit 1
SR0         s1                  ; bit 2 -> LSB
OUTPUT      s1                  ; output bit 2
SR0         s1                  ; bit 3 -> LSB
OUTPUT      s1                  ; output bit 3
SR0         s1                  ; bit 4 -> LSB
OUTPUT      s1                  ; output bit 4
SR0         s1                  ; bit 5 -> LSB
OUTPUT      s1                  ; output bit 5
SR0         s1                  ; bit 6 -> LSB
OUTPUT      s1                  ; output bit 6
SR0         s1                  ; bit 7 -> LSB
OUTPUT      s1                  ; output bit 7
LOAD        s3, s3              ; no operation
OUTPUT      s3                  ; output stop bit
```

Liste 1 - Algorithme linéaire

Dans l'algorithme linéaire, une instruction est lue après l'autre. Il n'y a pas de boucles ou de sauts dans le programme. Notre premier compteur de programme doit seulement pouvoir incrémenter l'adresse de un.



3.1.1 Implémentation

Un compteur itératif doit être créé. Commencer par créer le composant de base du compteur itératif : **`CNT/cnt_1bit`**. Ce compteur doit s'incrémenter au flanc montant de l'horloge, lorsque **`incPC = '1'`**. À noter que dans notre système, **`incPC`** est activé chaque seconde période d'horloge. Ignorer les signaux **`loadInstrAddr`** et **`instrAddress`** pour charger une nouvelle valeur dans le compteur.

- Établir la liste des entrées/sorties du compteur 1-bit.
- Créer le bloc du compteur 1-bit **`CNT/cnt_1bit`** (Voir Chapitre 4.1).
- Implémenter le circuit d'un compteur 1-bit à l'intérieur du bloc créé.
- Convertir ce bloc en composant (Voir Chapitre 4.2).

Les bascules D, soit disponibles dans la bibliothèques « Sequential » sous le nom de **`DFF`** (D Flip-Flop)

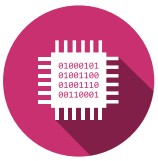
D'une fois le composant **`CNT/cnt_1bit`** créé, utiliser ce composant pour créer le compteur 8-bits.

- Copier-coller **8**-fois le composant **`CNT/cnt_1bit`**.
- Connecter les compteurs 1-bit entre eux pour créer le compteur 8-bits.

3.1.2 Simulation

Simuler le système et vérifier le bon fonctionnement du compteur et du processeur.

Vérifiez le bon fonctionnement du compteur en utilsant le banc de test **`CNT_test/nanoProcess_tb`** en utilsant le fichier de simulation **`$SIMULATION_DIR/CNT1.do`**.



3.2 Algorithme avec boucles

L'algorithme suivant permet une écriture plus compacte du programme, mais il utilise des boucles ou respectivement des sauts dans le programme:

```
LOAD    s3, FF              ; load stop bit
OUTPUT  s3                  ; output stop bit
LOAD    s2, 04              ; initialize loop counter 3
SUB     s2, 01              ; decrement loop counter 4
JUMP    NZ  03              ; loop back if not end of count 5
LOAD    s0, 00              ; load start bit 6
OUTPUT  s0                  ; output start bit 7
LOAD    s2, 08              ; initialize loop counter 8
INPUT   s1                  ; load word to send 9
LOAD    s3, s3              ; no operation
OUTPUT  s1                  ; output word, LSB is considered
SR0     s1                  ; next bit -> LSB
SUB     s2, 01              ; decrement loop counter
JUMP    NZ  0A              ; loop back if not end of count
OUTPUT  s3                  ; output bit 1
```

Liste 2 - Algorithme avec boucles

3.2.1 Réalisation

Pour réaliser l'algorithme avec boucles, le Program Counter (PC) doit permettre le chargement d'une nouvelle valeur. Les deux signaux **`instrAddr`** et **`loadInstrAddr`** sont utilisés pour charger une nouvelle valeur.

Le composant **`CNT/cnt_1bit`** doit être modifié, afin de permettre le chargement d'une nouvelle valeur dans le compteur.

- Modifier le composant **`CNT/cnt_1bit`** pour permettre le chargement d'une nouvelle valeur.
- Les entrées du composant ayant été changées, il est nécessaire de mettre à jour l'interface du composant (Voir Chapitre 4.3).

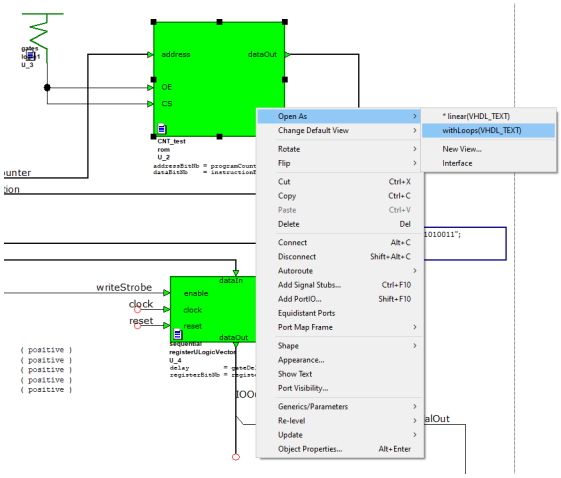
Dans la première partie du laboratoire, le compteur de programme a été réalisé en copiant-collant le composant **`CNT/cnt_1bit`** 8 fois. Afin de simplifier la réalisation, utilisez une boucle **`FOR Generate`** pour créer le compteur 8-bits.

- Utiliser une boucle **`FOR Generate`** autour du composant **`CNT/cnt_1bit`**, afin d'automatiquement le répéter 8 fois (Voir Chapitre 4.4).
- Connecter les différentes itérations entre elles pour créer le compteur 8-bits à l'aide de l'index **`i`** de la boucle.
- Utiliser le paramètre générique **`programCounterBitNb`** pour définir le nombre de bits du compteur et ainsi créer un compteur générique.

3.2.2 Simulation

Simulez le système et vérifiez le bon fonctionnement du compteur et du processeur. Modifiez la vue par défaut de la ROM pour sélectionner la version avec code incluant des boucles nommée **`withLoops`**.

1. **`Cliquez-droit`** sur le composant,
2. Sélectionnez **`Open As`**,
3. Cliquez sur **`withLoops`**

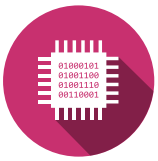


Vérifiez le bon fonctionnement du compteur en utilsant le banc de test à disposition **`CNT_test/nanoProcess_tb`** en utilsant le fichier de simulation **`$SIMULATION_DIR/CNT2.do`**.

3.3 Comparaison

Comparer les 2 algorithmes en termes de vitesse de transmission (baudrate) et la taille du code.

- Comparer la vitesse de transmission (baudrate) des deux algorithmes.
- Comparer la taille du code des deux algorithmes de Liste 1 et Liste 2.



4 Blocs, Composants et Boucles `FOR`

Les sections suivants vous guideront dans la création de blocs, de composants, et à l'utilisation de boucle **`FOR Generate`**.

4.1 Création d'un bloc

Pour créer un nouveau bloc, il faut ajouter un bloc en cliquant sur le bouton **`ajouter un bloc`** (voir Fig. 4).

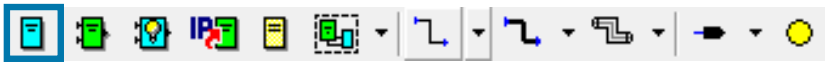


Fig. 4 - Ajouter un bloc

Après avoir cliqué sur le bouton, un nouveau bloc peut être ajouté en cliquant sur le schéma. Une fois le bloc ajouté, les Input/Outputs (I/Os) nécessaires peuvent être connectées :

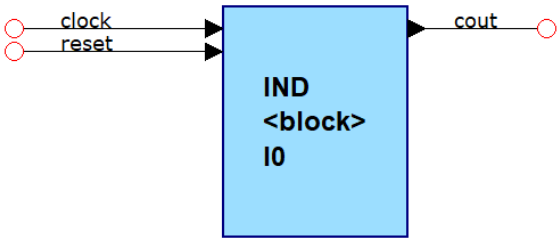
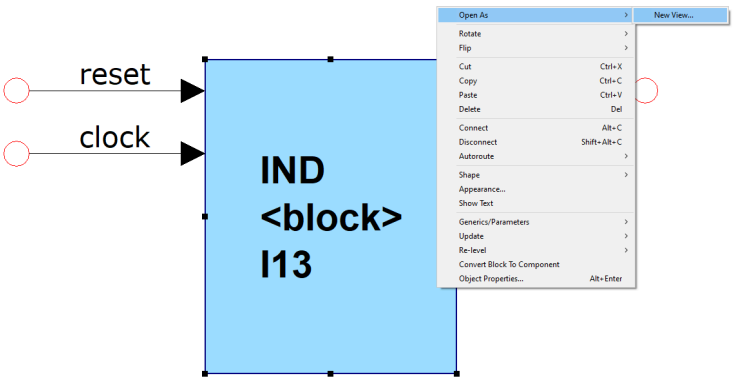


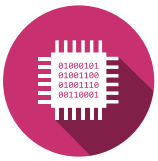
Fig. 5 - Nouveau bloc câblé

> ⚠ Un **`bloc bleu`** ne peut pas être copié et collé car il n'existe qu'une seule fois. Seul le **`bloc vert`** (composants) peut être copié et collé.

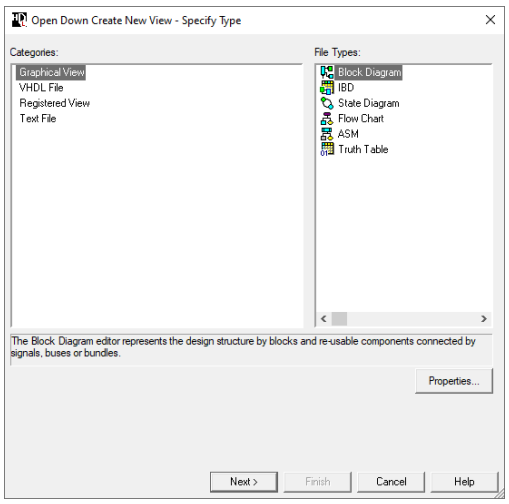
Une fois les I/Os câblées, le type de bloc doit être sélectionné (**`Block Diagram`** /**`State Diagram`** /**`VHDL file`**).

1. **`Cliquez-droit`** sur le bloc
2. Sélectionnez **`Open As`**
3. Cliquez sur **`New View...`**





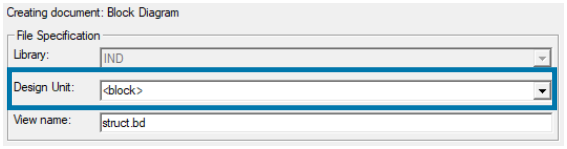
La fenêtre suivante s'ouvre, permettant de choisir le type de bloc à créer. Sélectionnez le type de bloc souhaité et cliquez sur **`Finish`**.



Link	Block type	Icon
`Graphical View/Block Diagram`	block diagram	
`Graphical View/State Diagram`	state machine	
`VHDL File/Architecture`	VHDL code	

4.1.1 Block diagram

- Sélectionnez **`Graphical View/Block Diagram`**
- Appuyez sur **`Next`**
- Entrez le nom du bloc sous **`Design unit`**
- Remplissez le tableau des I/Os
  - Assurez-vous que les types soient corrects
  - Définissez les limites pour les types multi-bits
- Appuyez sur « Finish »

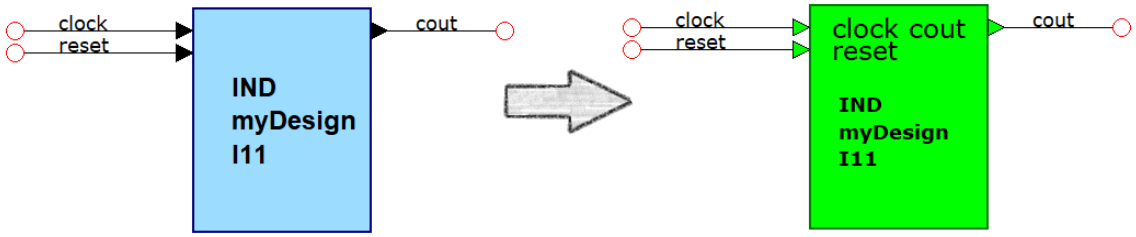


Les I/Os peuvent toujours être ajoutées, supprimées et modifiées lors de l'édition du schéma.

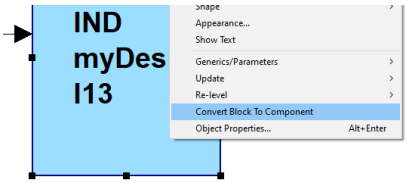
4.2 Convertir un bloc en composant (bleu vers vert)

Convertir un bloc en composant (de bleu à vert) permet de copier-coller l'élément et le rend accessible dans la bibliothèque du projet.

Les blocs bleus sont idéaux pour créer rapidement une interface de bloc, tandis que les composants verts sont indispensables pour réutiliser le bloc ailleurs dans le projet.



1. **`Cliquez-droit`** sur le composant (bloc bleu)
2. Cliquez sur **`Convert Block to component`**

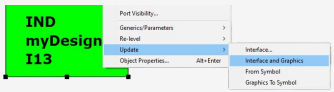


4.3 Mettre à jour l'interface du composant

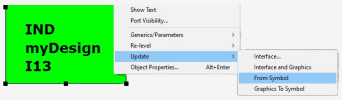
L'interface d'un composant se compose de plusieurs parties : **`Entrées`** et **`Sorties`**, **`Génériques`** et **`Symbole`**. Ces différents paramètres peuvent être ajoutés, supprimés et modifiés.

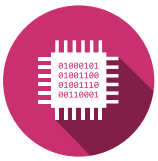
`Une fois l'interface du composant modifiée`

1. **`Cliquez-droit`** sur le composant
2. Sélectionnez **`Update`**
3. Cliquez sur **`Interface and Graphics`**



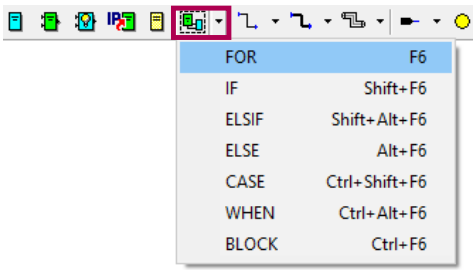
1. **`Cliquez-droit`** sur le composant
2. Sélectionnez **`Update`**
3. Cliquez sur **`From Symbol`**



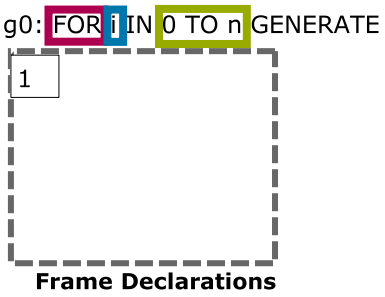


4.4 For Generate

Le cadre **FOR Generate** permet de créer plusieurs itérations de la même structure. Pour ajouter un nouveau cadre, **cliquez** sur le bouton **add frame** et **cliquez** sur **FOR**.



Quand vous ajoutez un cadre **FOR**, l'élément suivant est dessiné sur le schéma.



Part	Description
FOR	Type du cadre (**FOR** loop)
i	Index de la boucle
0 TO n	Plage de la boucle **FOR**

Tous les signaux et les blocs à l'intérieur du cadre seront copiés-collés **n+1** fois. L'index **i** va de **0** à **n**. *Modifier n par une constante ou un générique.*

Toutes les sorties de composants doivent utiliser l'index **i** dans leur **Slice/Index**. Sinon, des courts-circuits se produiront entre les sorties.

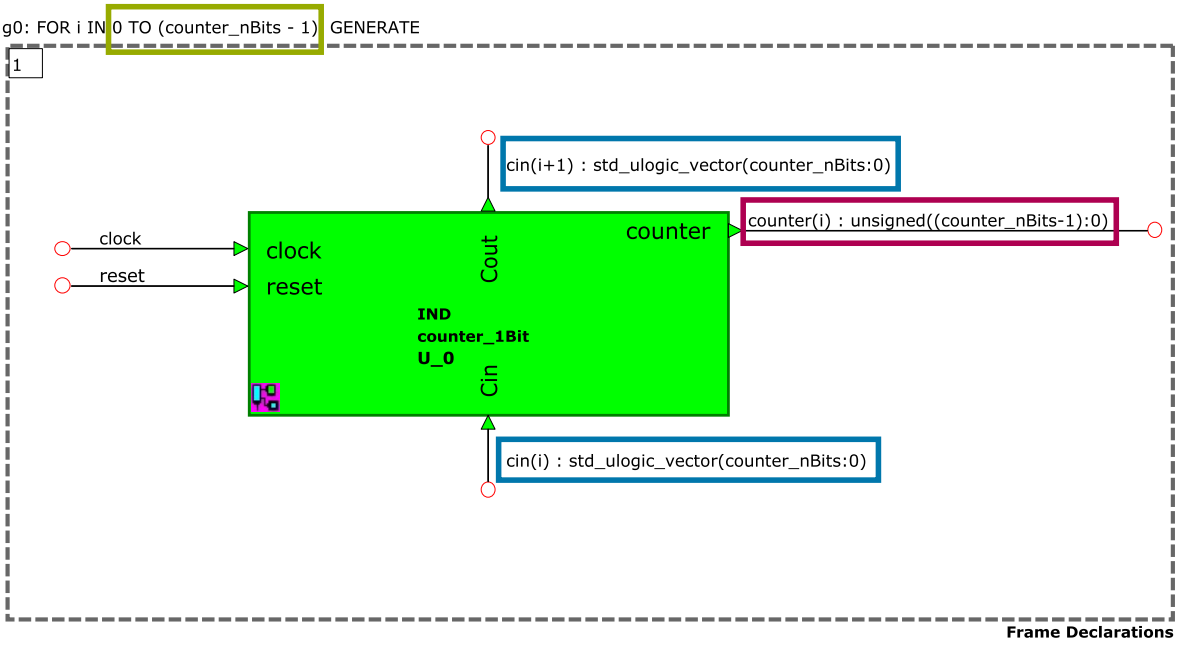
Les entrées de composants qui sont communes à toutes les instances (comme une horloge ou un reset) ne doivent pas utiliser l'index **i**. Par contre des entrées qui varient entre les instances (comme un signal de données) doivent utiliser l'index **i**.

Par exemple : l'horloge doit être la même partout ; mais un compteur doit sortir les differents bits. **Bit[0]**, puis **bit[1]**, puis **bit[2]** ... **bit[i]**.

- Seuls les signaux et les composants (verts) peuvent être placés dans une boucle **FOR**.
- Aucun **In Port** ou **Out Port** ne doit se trouver à l'intérieur du cadre (flèches bleues).
- Aucun bloc (bleu) ne peut être placé à l'intérieur.

4.4.1 Exemple

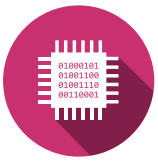
L'exemple montre le circuit de compteur itératif. Le générique **counter_nBits** permet de sélectionner le nombre de bits du compteur.



Part	Description
`0 TO (counter_nBits-1)`	Répéter la boucle **counter_nBits** fois
`counter(i): unsigned...`	Écriture sur la sortie du compteur. Le **1er bloc** écrit le **bit0** du signal compteur. Le **2ème bloc** écrit le **bit1** du signal compteur, etc..
`cin(i) : std_ulogic_vector ...` `cin(i+1) : std_ulogic_vector...`	Transmission l'information au bloc suivant. Le **cout** du **1er bloc** est connecté au **cin** du **2ème bloc**, etc..

Comme nous avons **cin(i+1)**, lorsque **i = (counter_nBits - 1)**⇨ **cin(counter_nBits)**, donc la taille du bus **cin** doit être **(counter_nBits DOWNTO 0)**.

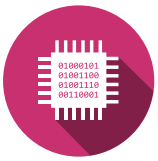
N'oubliez pas de connecter **cin(0)**, EN DEHORS du cadre.



5 Checkout

C'est la fin du laboratoire, vous avez implémenté avec succès un compteur de programme qui supporte toutes les instructions du μProcesseur. Avant de quitter le laboratoire, assurez-vous d'avoir complété les tâches suivantes :

- ☐ Conception du circuit
  - ☐ Le bloc compteur 1-bit **`CNT/cnt_1bit`** a été créé.
  - ☐ Le cadre **`FOR`** a été utilisé pour dupliquer le bloc compteur 1-bit.
  - ☐ Le compteur **`n`**-bit a été créé.
  - ☐ Le compteur a été ajusté pour permettre le chargement d'une nouvelle valeur.
- ☐ Simulations
  - ☐ Les deux compteurs ont été testés avec succès avec le banc d'essai **`CNT_test/nanoProcess_tb`** et les fichiers de simulation **`$SIMULATION_DIR/CNT1.do`** et **`$SIMULATION_DIR/CNT2.do`**.
- ☐ Comparaison
  - ☐ Le débit en bauds des deux algorithmes a été comparé.
  - ☐ La taille du code des deux algorithmes a été comparée.
- ☐ Documentation et fichiers du projet
  - ☐ Assurez-vous que toutes les étapes (conception, simulations, comparaison) sont bien documentées dans votre rapport de laboratoire.
  - ☐ Enregistrez le projet sur une clé USB ou dans le dossier réseau partagé (**`\\filer01.hevs.ch`**).
  - ☐ Partagez les fichiers avec votre partenaire de laboratoire pour assurer la continuité du travail.



Glossaire

ALU – **Arithmetic and Logical Unit** 2, 2, 2, 2

CS – **Chip Select** 2

I/O – **Input/Output** 7, 7, 8, 8

LSB – **Least Significant Bit** 3

MI – **Machine Instruction**: A machine instruction is a binary-coded operation that a processor can execute directly. It typically consists of an opcode, operands and immediates. 2

OE – **Output Enable** 2

PC – **Program Counter** 5

ROM – **Read-Only Memory** 1, 1, 2, 2, 2, 2, 2, 3, 6