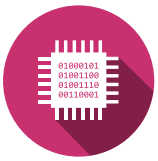


Unité arithmétique et logique

Laboratoire Conception Numérique

Contenu

1 Objectif

Ce laboratoire a pour but de s'exercer à la conception de circuits logiques à l'aide de multiplexeurs. Il permet de mettre en œuvre des unités arithmétiques et logiques (Logical Unit (LU) et Arithmetic Unit (AU)) pour les microprocesseurs.

Lors d'une première séance de laboratoire, les LU et AU sont réalisés. Lors d'une deuxième séance, une Arithmetic and Logical Unit (ALU) complète est construite, combinant les deux unités et sélectionnant le résultat souhaité à l'aide de signaux de contrôle appropriés.

2 Unité logique LU

La Fig. 1 montre le schéma d'une Unité Logique (LU) d'un microprocesseur. Les opérations logiques sont effectuées bit à bit. Huit de ces blocs forment une LU de 8 bits.

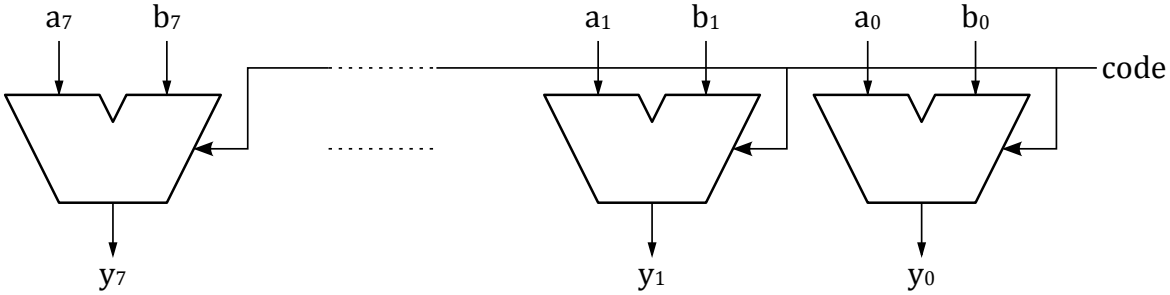


Fig. 1 - Unité logique LU

Le circuit logique de chaque bloc itératif est réalisé avec des multiplexeurs, qui représentent une table de vérité. Les entrées de contrôle $\text{sel}_0 = a_i$, $\text{sel}_1 = b_i$ et $(\text{sel}_3, \text{sel}_2) = \text{code}[1:0]$ servent à déterminer la fonction à générer.

Établissez la table de vérité de la fonction logique qui produit les opérations suivantes dans le bloc logique programmable :

- $y_i = b_i$ pour **`code = "00"`** Chargement de la valeur b
- $y_i = a_i * b_i$ pour **`code = "01"`** Fonction ET entre a et b
- $y_i = a_i + b_i$ pour **`code = "10"`** Fonction OU entre a et b
- $y_i = a_i \oplus b_i$ pour **`code = "11"`** Fonction OU exclusif entre a et b

2.1 Implémentation et simulation

Implémentez le circuit de la LU de Fig. 1. Certains éléments du circuit sont déjà présents dans le bloc **`ALU/LU1`**. Complétez les entrées manquantes du multiplexeur qui doit être connectées soit à un **`"0"`** logique, soit à un **`"1"`** logique. Ces valeurs peuvent être générées par les éléments **`gates/logic0`** respectivement **`gates/logic1`**.

Le banc d'essai **`ALU_test/LU8_tb`** est déjà fourni mais ne teste pas tous les cas. Le banc d'essai doit vérifier la fonctionnalité de l'ensemble de la LU.

Complétez le circuit du bloc itératif de l'**`ALU/LU1`** qui effectue les 4 opérations spécifiées.

Complétez les stimuli de test **`ALU_test/LU8_tester`** et vérifiez la fonction de l'ensemble du **`ALU_test/LU8_tb`** avec le fichier de simulation **`$SIMULATION_DIR/ALU1.do`**.

3 Unité arithmétique AU

La Fig. 2 montre le circuit itératif d'une Unité Arithmétique (AU). Ce circuit est composé de huit unités de 1 bit connectées pour former une AU de 8 bits. Le circuit logique sera réalisé avec des multiplexeurs, qui représentent une table de vérité.

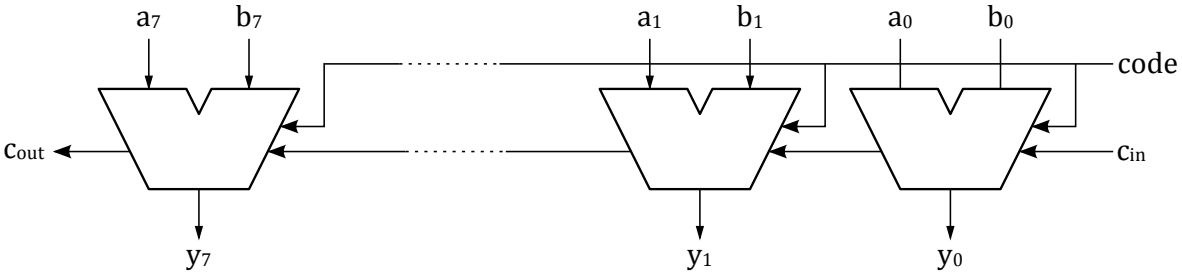


Fig. 2 - Unité arithmétique itérative

Le circuit itératif de chaque bloc itératif est réalisé avec des multiplexeurs, qui représentent une table de vérité. Les entrées de contrôle $\text{sel}_0 = a_i$, $\text{sel}_1 = b_i$ et $\text{sel}_2 = \text{c}_{\text{in}}$ et $\text{sel}_3 = \text{code}[0]$ servent à déterminer la fonction à générer.

Établissez la table de vérité du bloc logique itératif **`ALU/AU1`** de Fig. 2 pour les opérations suivantes sur les entiers :
- $y = a + b + c_{\text{in}}$ pour **`code[0] = '0'`** ⇨ addition
- $y = a - b - c_{\text{in}}$ pour **`code[0] = '1'`** ⇨ soustraction

En considérant qu'un décalage vers la gauche correspond à une multiplication par deux, proposez une extension du circuit itératif pour réaliser la fonction de décalage vers la gauche :

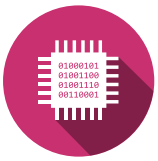
$$y = a \ll 1 = 2a = a + a \text{ pour } \texttt{code[1:0] = '10'}$$

Étendez le circuit **`ALU/AU1`** pour prendre en charge l'opération de décalage vers la gauche lorsque la valeur de **`code[1:0] = '10'`**.

3.1 Implémentation et simulation

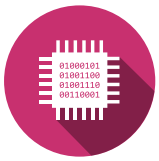
Implémentez le circuit de l'AU de Fig. 2 pour couvrir toutes les fonctionnalités mentionnées précédemment. Certains éléments du circuit sont déjà présents dans le bloc **`ALU/AU8`**.

Le banc d'essai **`ALU_test/AU8_tb`** est déjà fourni mais ne teste pas tous les cas. Le banc d'essai doit vérifier la fonctionnalité de l'AU dans son ensemble.



Complétez le circuit du bloc itératif de l'**`ALU/AU1`** qui effectue les 3 opérations spécifiées.

Complétez les stimuli de test **`ALU_test/AU8_tester`** et vérifiez la fonction de l'ensemble du **`ALU_test/AU8_tb`** avec le fichier de simulation **`$SIMULATION_DIR/ALU2.do`**.



4 Checkout Part 1

Ceci est la fin de la première partie du laboratoire, vous avez réussi à construire un Logical Unit et un Arithmetic Unit. Avant de quitter le laboratoire, assurez-vous d'avoir accompli les tâches suivantes :

- [ ] Conception du circuit
  - [ ] Vérifiez que les blocs **`ALU/LU8`** et **`ALU/AU8`** ont été conçus et testés avec les fonctionnalités mentionnées dans Chapitre 2 et Chapitre 3.
- [ ] Simulations
  - [ ] Les tests spécifiques des bancs d'essai respectifs (**`ALU_test/AU8_tb`** et **`ALU_test/LU8_tb`**) ont été adaptés au circuit et garantissent un test complet.
  - [ ] Les circuits ont été testés avec succès avec les bancs d'essai respectifs **`ALU_test/AU8_tb`** et **`ALU_test/LU8_tb`**.
- [ ] Documentation et fichiers de projet
  - [ ] Assurez-vous que toutes les étapes (conception, conversions, simulations) sont bien documentées dans votre rapport de laboratoire.
  - [ ] Enregistrez le projet sur une clé USB ou sur le lecteur réseau partagé (**`\\filer01.hevs.ch`**).
  - [ ] Partagez les fichiers avec votre partenaire de laboratoire pour assurer la continuité du travail.

5 Unité Arithmétique et Logique ALU

L'Arithmetic and Logical Unit (ALU) est réalisée par la combinaison des Logical Unit et Arithmetic Unit développées jusqu'à présent dans Chapitre 2 et Chapitre 3, elle comprend également une opération de *décalage à droite* supplémentaire, voir Fig. 3. Elle supportera le code assembleur de l'instruction Xilinx PicoBlaze Fig. 3.

Afin que l'ALU puisse exécuter les différentes opérations, les signaux de contrôle doivent être définis en conséquence. Il s'agit de $\mathrm{LU}_{\mathrm{Code}}[1:0]$, $\mathrm{AU}_{\mathrm{Code}}[1:0]$, $\mathrm{select}_{\mathrm{AU}}$, $\mathrm{select}_{\mathrm{SR}}$, $c_{\mathrm{in\_AU}}$. Ces signaux sont listés dans Fig. 3 ainsi que dans Tableau 2.

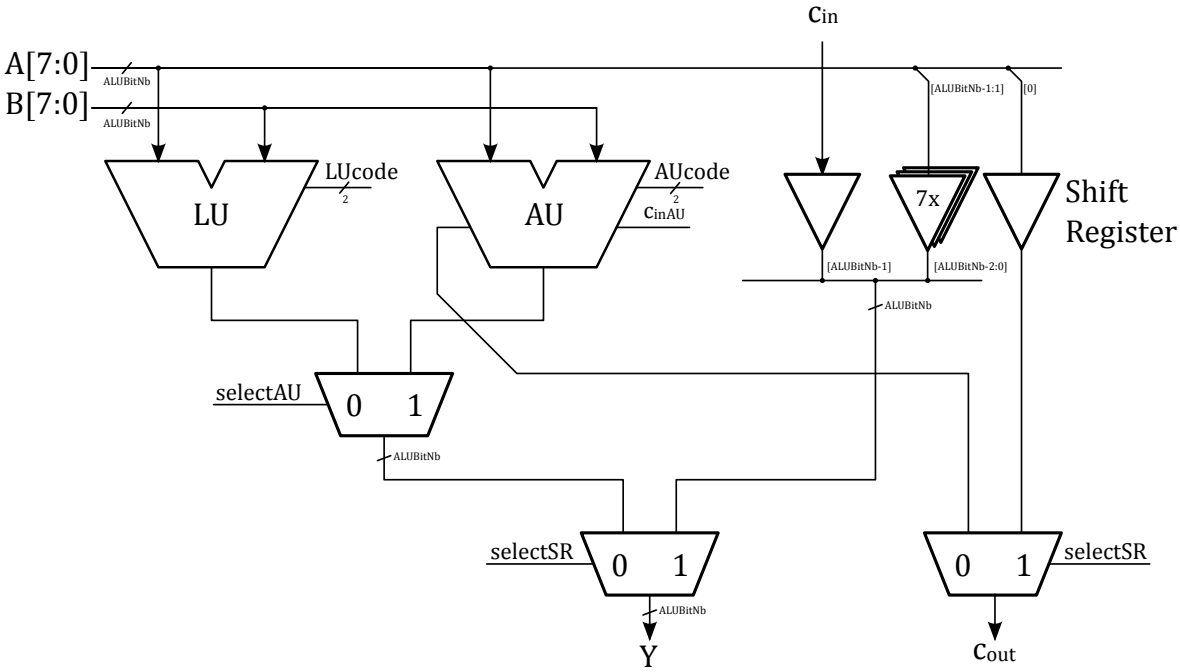
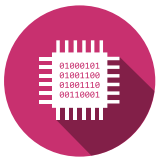


Fig. 3 - Structure interne ALU

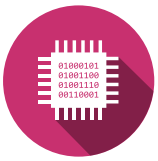
Étudier le fonctionnement du registre à décalage dans Fig. 3 et pour quelles instructions il est utilisé.



Les différentes opérations sont listées dans le Tableau 1. Il définit quel OpCode exécute quelle opération. L'OpCode est un code de 5 bits situé dans les bits $I_{17} : I_{13}$ du mot d'instruction.

OpCode $I_{17} : I_{13}$	Assembler Instruction	ALU code Operation	Operation
00000	LOAD	LOAD B	$y = b$
00001	*ununsed*	-	-
00010	INPUT	LOAD B	$y = b$
00011	FETCH	LOAD B	$y = b$
00100	*ununsed*	-	-
00101	AND	AND	$y = a$ AND b
00110	OR	OR	$y = a$ OR b
00111	XOR	XOR	$y = a$ XOR b
01000	*ununsed*	-	-
01001	TEST	AND	$y = a$ AND b
01010	COMPARE	SUB	$y = a - b$
01011	*ununsed*	-	-
01100	ADD	ADD	$y = a + b$
01101	ADDCY	ADDCY	$y = a + b + c_{\text{in}}$
01110	SUB	SUB	$y = a - b$
01111	SUBCY	SUBCY	$y = a - b - c_{\text{in}}$
10000	SH / ROT	SHR	$a \gg 1$
10001	SH / ROT	SHL	$a \ll 1$
10010	*ununsed*	-	-
10011	*ununsed*	-	-
10100	*non-ALU*	-	-
...	...	...	...
11111	*non -LU*	-	-

Tableau 1 - Décodage de l'ALU en relation avec les opérations ALU



5.1 Implémentation et Simulation

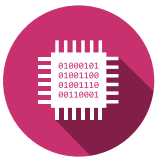
Pour chaque instruction, tous les signaux de contrôle dans le circuit **`ALU/ALU8`** tels que les multiplexeurs et les commandes des LU et AU sont donnés dans le Tableau 2. Le tableau de vérité est basé sur l'OpCode de Tableau 1.

code[4 : 0]	$\mathrm{LU}_{\mathrm{code}}[1:0]$	$\mathrm{AU}_{\mathrm{code}}[1:0]$	$\mathrm{select}_{\mathrm{AU}}$	$\mathrm{select}_{\mathrm{SR}}$	$c_{\mathrm{in\_AU}}$
00000					
00001					
00010					
00011					
00100					
00101					
00110					
00111					
01000					
01001					
01010					
01011					
01100					
01101					
01110					
01111					
10000					
10001					
10010					
10011					
10100					
...					
11111					

Tableau 2 - Signaux de commande de l'ALU

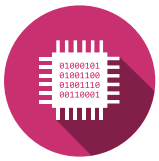
Compléter le Tableau 2 qui donne les valeurs de tous les signaux de contrôle en fonction de l'ALU Opération indiquée par le signal code[4 : 0].

Pour chaque signal de contrôle, déduire l'équation booléenne et l'implémenter dans le circuit **`ALU/ALU8`**.



Vérifier et si nécessaire compléter les stimuli de test **`ALU_test/ALU8_tester`**. Exécuter le banc de tests **`ALU_test/ALU8_tb`** avec le fichier de simulation **`$SIMULATION_DIR/ALU3.do`** et vérifier la fonctionnalité de l'ALU.

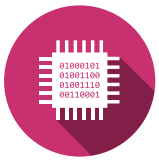
Les opérations individuelles telles que **`ADD`**, **`OR`**, **`AND`**, ... ont déjà été vérifiées dans le labo précédent. Et ne doivent pas être vérifiées à nouveau dans leur intégralité.



6 Checkout Partie 2

Ceci est la fin de la deuxième partie du laboratoire, vous avez réussi à construire une Arithmetic and Logical Unit du Xilinx PicoBlaze. Avant de quitter le laboratoire, assurez-vous d'avoir complété les tâches suivantes :

- [ ] Compréhension
  - [ ] Vous comprenez comment fonctionnent les différentes composantes de la ALU, en particulier le nouveau circuit du registre à décalage.
- [ ] Conception du circuit
  - [ ] Vérifiez que le bloc **`ALU/ALU8`** a été conçu et testé avec les fonctionnalités mentionnées dans Tableau 1.
- [ ] Simulations
  - [ ] Les tests spécifiques du banc de tests **`ALU_test/ALU8_tb`** ont été adaptés au circuit et garantissent un test complet.
- [ ] Documentation et fichiers du projet
  - [ ] Assurez-vous que toutes les étapes (conception, conversions, simulations) sont bien documentées dans votre rapport de laboratoire.
  - [ ] Enregistrez le projet sur une clé USB ou le lecteur 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** 1, 6, 7, 7, 8, 8, 9, 10

***AU* – Arithmetic Unit** 1, 1, 3, 3, 3, 3, 8

***LU* – Logical Unit** 1, 1, 2, 2, 2, 2, 8

PicoBlaze: PicoBlaze is a small, 8-bit microcontroller designed by Xilinx for use in FPGAs. It is often used in educational settings to teach basic microcontroller concepts. 6, 10