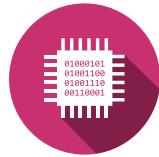


# Bus internes de microprocesseur

Laboratoire Conception Numérique

## Contenu

1 Objectif .....	2
2 Bus de données de l'ALU .....	3
2.1 Connexion des registres aux bus de données .....	4
2.2 Connexion au bus d'entrée/sortie .....	4
2.3 Données provenant de l'instruction .....	5
2.4 Réalisation .....	5
3 Réalisation logicielle d'un port série .....	6
3.1 Transmission sérielle .....	6
3.2 Algorithme .....	6
3.3 Implémentation .....	7
3.4 Simulation .....	7
4 Checkout .....	8
Glossaire .....	9



# 1 | Objectif

Ce laboratoire vise à pratiquer l'utilisation de circuits à trois états, en particulier dans le contexte des bus de données partagés sur un microprocesseur Xilinx **PicoBlaze**.

Il fournit des informations sur le fonctionnement interne d'un microprocesseur, en mettant l'accent sur l'interaction entre ses composants. Une attention particulière est accordée au fichier de registres (également connu sous le nom de registres de données), qui comprend les registres  $s_0$ ,  $s_1$ ,  $s_2$  et  $s_3$ .

En explorant les opérations de l'[Arithmetic and Logical Unit \(ALU\)](#), le fichier de registres et leur connexion à un bus partagé, ce laboratoire démontre comment la logique à trois états permet à plusieurs composants de communiquer sur un chemin de données commun sans interférence.

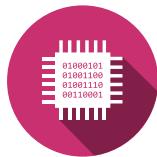
i

**Bibliothèque IO**

Dans ce laboratoire, vous utiliserez des composants à trois états. Ceux-ci se trouvent dans la bibliothèque **IO**.

S'ils ne sont pas visibles dans le navigateur de composants, vous pouvez les ajouter comme suit.

The screenshot shows the Xilinx ISE software interface. At the top, there is a toolbar with various icons. Below it, the main workspace is divided into two windows: 'Component Browser' on the left and 'Explore Library' on the right. In the 'Component Browser', there are categories like 'common', 'gates', and 'sequential'. In the 'Explore Library', there is a list of components including 'COR', 'COR\_test', 'IND', 'IND\_test', and 'IO'. The 'IO' component is checked, indicated by a blue checkmark. At the bottom of the 'Explore Library' window, there are 'OK' and 'Cancel' buttons.



## 2 | Bus de données de l'ALU

La Fig. 1 montre une partie du  $\mu$ Processeur Xilinx PicoBlaze, composé de:

- un **ALU**
- un registre de 4 registres ( $s_0, s_1, s_2, s_3$ )
- une interface à un bus d'entrée/sortie (**Input/Output (I/O)**)
- et une connexion à l'instruction du  $\mu$ Processeur.

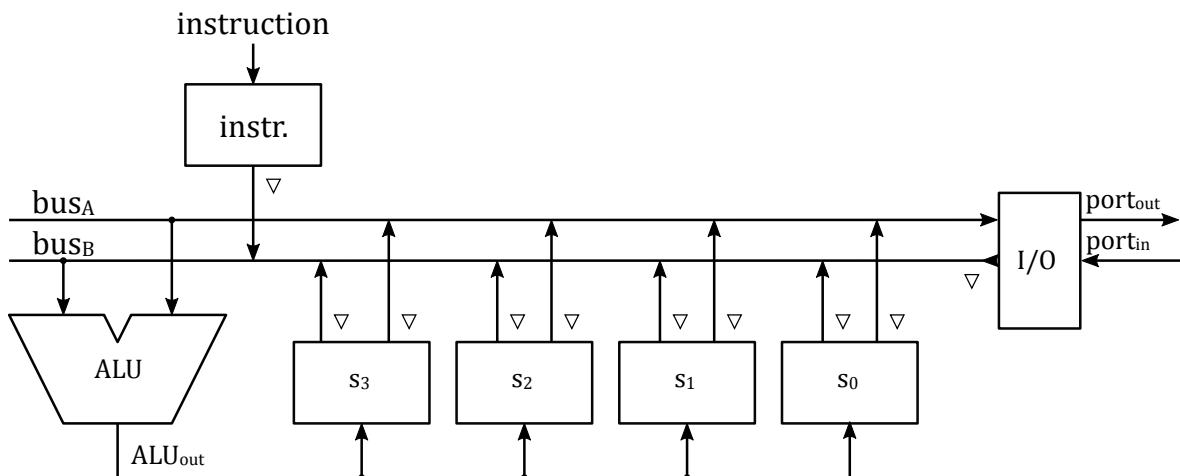


Fig. 1 - Composants du  $\mu$ Processeur reliés par les bus de données  $bus_A$  et  $bus_B$ .

Les composants sont reliés par deux bus de données,  $bus_A$  et  $bus_B$ , qui sont utilisés pour transférer des données entre les composants. Sur le  $bus_A$ , les 4 registres peuvent transférer des données vers l'**ALU** et est utilisé pour le *premier opérande* d'une opération. Tandis que le  $bus_B$  est connecté aux 4 registres, au bloc **I/O** et au bloc d'instruction, et est utilisé pour le *second opérande* d'une opération.

```

1 ADD s0 s1 # Adds the contents of register s1 to register s0. s0 = s0 + s1
2 #   ^   ^
3 #   |   +-+ Second operand, register s1
4 #   |   +---+ First operand, register s0
5 #   +-----+ Operation, ADD

```

Liste 1 - Exemple d'une instruction assembleur **ADD**.

Le premier opérande est le registre  $s_0$ , le second est le registre  $s_1$ .

Le flux de données pour  $bus_A$  va:

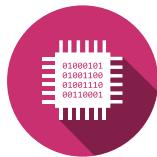
- d'un des 4 registres  $s_0-s_3$  vers l'**ALU** pour le premier opérande d'une opération.
- d'un des 4 registres  $s_0-s_3$  vers le bloc **I/O** pour écrire des données vers un périphérique externe.

Le flux de données pour  $bus_B$  va:

- d'un des 4 registres  $s_0-s_3$  vers l'**ALU** pour le second opérande d'une opération.
- du bloc **I/O** vers l'**ALU** pour le second opérande d'une opération.
- du bloc d'instruction vers l'**ALU** pour le second opérande d'une opération.

Le flux de données pour  $ALU_{out}$  va:

- de l'**ALU** vers un des 4 registres  $s_0-s_3$  pour écrire le résultat d'une opération.



## 2.1 Connexion des registres aux bus de données

Fig. 2 montre deux registres  $s_1$  et  $s_2$  avec leur bloc de contrôle pour écrire dans les registres ou lire à partir des registres vers le bus<sub>A</sub> ou le bus<sub>B</sub>.

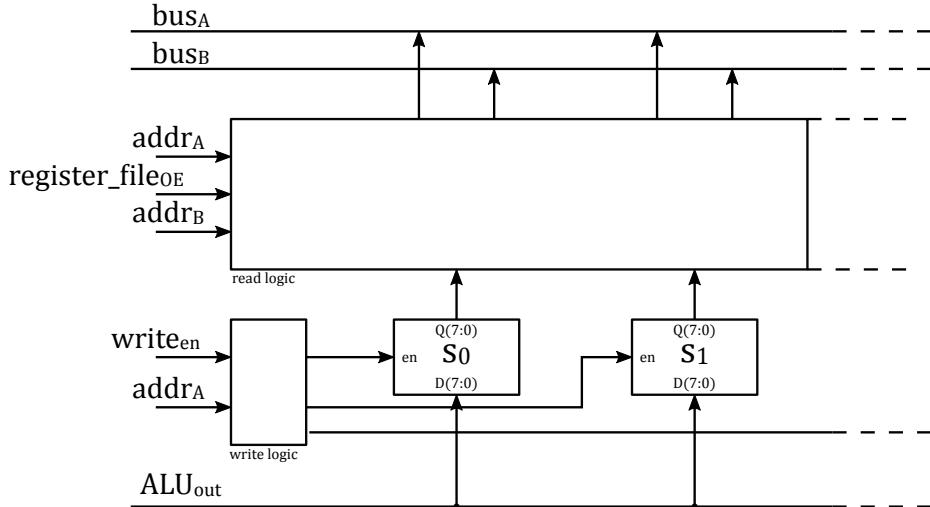


Fig. 2 - Registres de données  $s_X$

Le circuit de Fig. 2 permet de connecter les sorties des registres aux bus<sub>A</sub> et bus<sub>B</sub>. Le système créé doit permettre à un registre de placer ses données sur le bus<sub>A</sub> et un autre de le faire sur le bus<sub>B</sub>.

Les nombres addr<sub>A</sub> et addr<sub>B</sub> indiquent quel registre transmet ses informations sur le bus<sub>A</sub>, respectivement le bus<sub>B</sub>. Le signal register\_file<sub>OE</sub> indique si les données du registre sélectionné doivent être amenées sur le bus<sub>B</sub> et empêche un conflit avec les données provenant du port I/O ou du bloc d'instruction.

De même, les signaux write<sub>en</sub> et addr<sub>A</sub> sont utilisés pour écrire dans les registres. Ainsi, une opération dont le premier opérande est le registre sélectionné par addr<sub>A</sub> écrira son résultat dans ce même registre. Comme vu dans le code assembleur de Liste 1, le premier opérande est le registre  $s_0$  et le résultat de l'opération sera écrit dans ce même registre.



Développez la logique de lecture et d'écriture pour les registres  $s_0$ ,  $s_1$ ,  $s_2$  et  $s_3$  dans le bloc **MIB/aluAndRegister**.

## 2.2 Connexion au bus d'entrée/sortie

Lors de la lecture de données depuis l'extérieur, il est nécessaire d'activer le signal de contrôle port<sub>in\_OE</sub>, puis les données du bus port<sub>in</sub> sont écrites sur le bus<sub>B</sub>. Lors de l'écriture de données vers l'extérieur, les données du bus<sub>A</sub> sont écrites sur le bus port<sub>out</sub>, et un signal externe vers l'**ALU**, write<sub>strobe</sub>, est activé via le banc d'essai. Cela permet d'enregistrer ces données dans un registre externe au  $\mu$ Processeur.

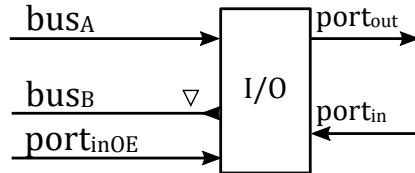
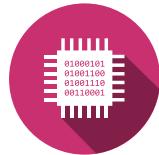


Fig. 3 - Bloc d'entrée/sortie



Développez le schéma interne du bloc I/O dans **MIB/aluAndRegisters** à partir de Fig. 3.

## 2.3 Données provenant de l'instruction

Pour le second opérande d'une opération, une valeur constante peut être codée dans l'instruction et amenée sur le bus<sub>B</sub> de l'**ALU**. Le bloc gérant ce transfert est représenté dans la Fig. 4.

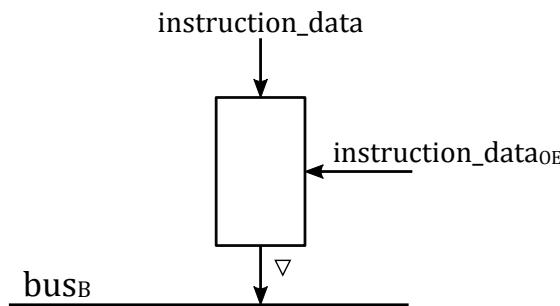


Fig. 4 - Données en provenance de l'instruction

```

1 LOAD s0 10 # Loads the constant 10 to register s0. s0 = 10
2 #   ^   ^   ^
3 #   |   |   +- immediate (constant) value 10
4 #   |   +--- First operand, register s0
5 #   +----- Operation, LOAD

```

Liste 2 - Exemple d'une instruction assembleur **LOAD**.

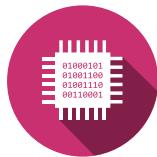
Le premier opérande est le registre  $s_0$ , le second est une valeur immédiate (constante).



Développez le schéma interne du bloc d'instruction à partir de Fig. 4 dans **MIB/aluAndRegisters**.

## 2.4 Réalisation

En se basant sur les points précédents, le circuit de bus interne du  $\mu$ Processeur est complété dans **MIB/aluAndRegisters**.



# 3 | Réalisation logicielle d'un port série

## 3.1 Transmission série

La Fig. 5 présente le déroulement temporel de l'envoi en série d'un mot de donnée.

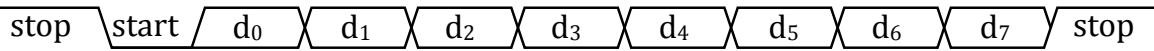


Fig. 5 - Transmission série

Lors d'une transmission série, le signal sur le bus est par défaut à un niveau haut (logique 1). La transmission débute par un bit de départ (logique 0), suivi des bits de données, et se termine à nouveau par un bit d'arrêt (logique 1). Les bits de données sont envoyés les uns après les autres, en commençant par le bit de poids faible (LSB) jusqu'au bit de poids fort (MSB).

Dans notre application, le signal série est transmis sur le bit de poids faible du bus  $\text{port}_{\text{out}}$  resp.  $\text{port}_{\text{out}}[0]$ . Dans le banc d'essai, ce bus est connecté à un registre externe **MIB\_test/MIB\_tb/I2**. La commande  $\text{write}_{\text{strobe}}$  du banc d'essai contrôle l'écriture dans ce registre.

## 3.2 Algorithme

L'algorithme à programmer est le suivant :

```

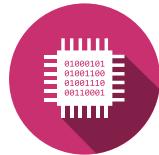
1 LOAD      s3, FF          # load stop bit
2 OUTPUT    s3              # output stop bit
3 LOAD      s3, s3          # no operation
4 LOAD      s3, s3          # no operation
5 LOAD      s3, s3          # no operation
6 LOAD      s3, s3          # no operation
7 LOAD      s0, 00          # load start bit
8 OUTPUT    s0              # output start bit
9 INPUT     s1              # load word to send
10 OUTPUT   s1              # output word, LSB is considered
11 SR0      s1              # shift word, bit 1 -> LSB
12 OUTPUT   s1              # output bit 1
13 SR0      s1              # bit 2 -> LSB
14 OUTPUT   s1              # output bit 2
15 SR0      s1              # bit 3 -> LSB
16 OUTPUT   s1              # output bit 3
17 SR0      s1              # bit 4 -> LSB
18 OUTPUT   s1              # output bit 4
19 SR0      s1              # bit 5 -> LSB
20 OUTPUT   s1              # output bit 5
21 SR0      s1              # bit 6 -> LSB
22 OUTPUT   s1              # output bit 6
23 SR0      s1              # bit 7 -> LSB
24 OUTPUT   s1              # output bit 7
25 LOAD      s3, s3          # no operation
26 OUTPUT   s3              # output stop bit

```

Liste 3 - Implémentation logicielle du protocole de transmission série



Étudier et comprendre l'algorithme du protocole de transmission série Liste 3.



### 3.3 Implémentation

Chaque ligne ou instruction doit être implémentée dans le bloc Tester du Testbench.



Complétez le testbench tester **MIB\_test/MIB\_tb** pour exécuter la séquence d'instructions de la transmission série Liste 3.



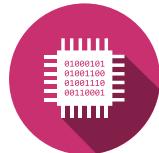
Il est important de ne laisser aucun bus en état de haute impédance. Programmez l'algorithme de manière à ce qu'il y ait toujours un signal sur bus<sub>A</sub> et bus<sub>B</sub>, même lorsque aucune information n'est recherchée.

### 3.4 Simulation



Simulez le Testbench **MIB\_test/MIB\_tb** avec le fichier de simulation **\$SIMULATION\_DIR/MIB.do**.

Combien de bits et quelle valeur de données sont transmis ?



## 4 | Checkout

Vous avez terminé le labo, vous avez construit la structure interne du  $\mu$ Processor Xilinx **PicoBlaze**. Avant de quitter le laboratoire, assurez-vous d'avoir complété les tâches suivantes :

- Conception du circuit
  - Vérifiez que le bloc **MIB/aluAndRegisters** a été conçu et testé avec les fonctionnalités mentionnées.
- Simulations
  - Assurez-vous d'avoir compris l'algorithme de transmission série Liste 3.
  - Les instructions spécifiques ont été implémentées dans **MIB\_test/MIB\_tester**.
  - La valeur et le nombre de bits transmis sont lus à partir de la simulation.
- Documentation et fichiers du projet
  - Assurez-vous que toutes les étapes (conception, conversions, simulations) sont bien documentées dans votre rapport de labo.
  - Enregistrez le projet sur une clé USB ou sur le lecteur réseau partagé (\\\filer01.hevs.ch).
  - Partagez les fichiers avec votre partenaire de labo pour assurer la continuité du travail.



# Glossaire

**ALU – Arithmetic and Logical Unit** [2](#), [3](#), [3](#), [3](#), [3](#), [3](#), [3](#), [3](#), [4](#), [5](#)

**I/O – Input/Output** [3](#), [3](#), [3](#), [3](#), [4](#), [5](#)

**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. [2](#), [3](#), [8](#)