# Microprocessor internal databus

Labor Digital Design

## Contents

## 1 Goal

This lab aims to practice the use of tri-state circuits, specifically within the context of shared data buses on a Xilinx PicoBlaze $\mu$Processor.

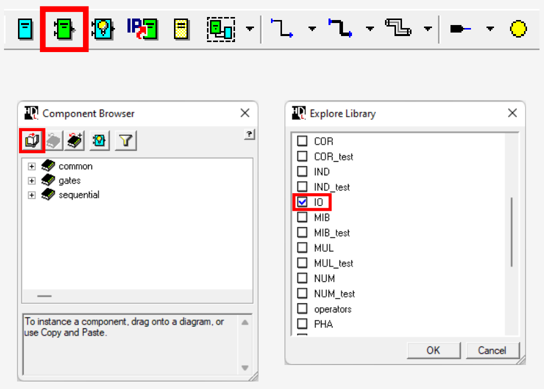
It provides insights into how a $\mu$Processor operates internally, focusing on the interaction between its components. A key emphasis is placed on the register file (also known as data registers), which includes registers $s_0$, $s_1$, $s_2$, and $s_3$.

By exploring the Arithmetic and Logical Unit (ALU) operations, register file and their connection to a shared bus, this lab demonstrates how tri-state logic enables multiple components to communicate over a common data path without interference.

**Library `IO`**

In this lab, you will be using tri-state components. These can be found in the **`IO`** library.

If they are not visible in the component browser, you can add them as follows.



To instance a component, drag onto a diagram, or use Copy and Paste.

# 2 Data buses of the ALU

Figure 1 shows a part of the Xilinx PicoBlaze $\mu$Processor, composed of:
- an ALU
- a registerfile of 4 registers ($s_0$, $s_1$, $s_2$, $s_3$)
- an interface to an input/output (Input/Output (I/O)) bus
- and a connection to the $\mu$Processor instruction.

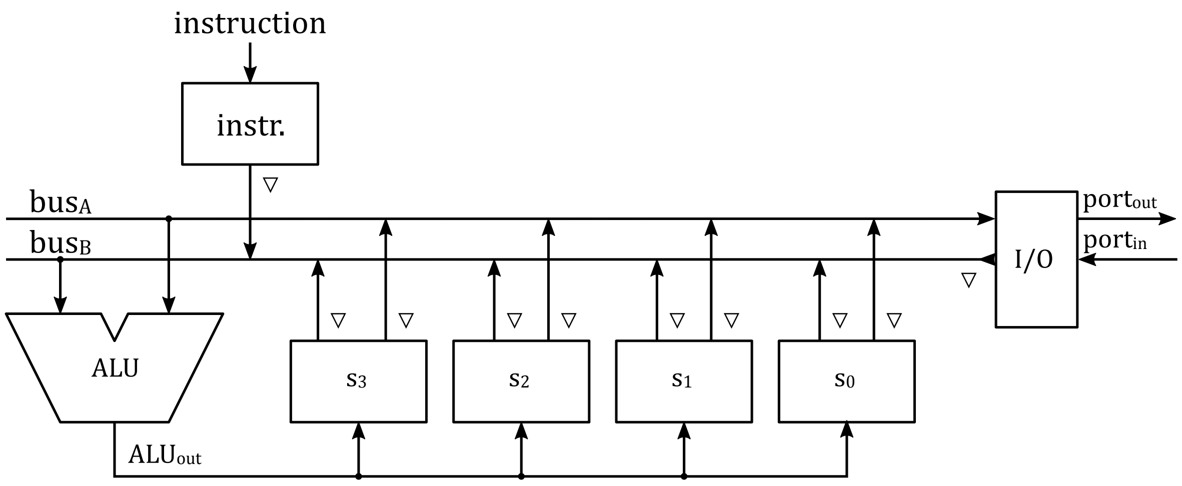


Figure 1 - $\mu$Processor components connected by data buses $\text{bus}_A$ and $\text{bus}_B$.

The components are connected by two data buses, $\text{bus}_A$ and $\text{bus}_B$, which are used to transfer data between the components. On the $\text{bus}_A$ the 4 registers can transfer data to the ALU and is used for the *first operand* of an operation. While the $\text{bus}_B$ is connected to the 4 registers, the I/O block, and the instruction block, and is used for the *second operand* of an operation.

```
  ADD s0 s1  # Adds the contents of register s1 to register s0. s0 = s0 + s1
#  ^   ^  ^
#  |   |  +-- Second operand, register s1
#  |   +----- First operand, register s0
#  +--------- Operation, ADD
```

Listing 1 - Example of a assember instruction **ADD**.
The first operand is register $s_0$, the second is register $s_1$.

The dataflow for $\text{bus}_A$ goes:
- from one of the 4 registers $s_0$-$s_3$ to the ALU for the first operand of an operation.
- from one of the 4 registers $s_0$-$s_3$ to the I/O block for writing data to an external device.

The dataflow for $\text{bus}_B$ goes:
- from one of the 4 registers $s_0$-$s_3$ to the ALU for the second operand of an operation.
- from the I/O block to the ALU for the second operand of an operation.
- from the instruction block to the ALU for the second operand of an operation.

The dataflow for $\text{ALU}_{\text{out}}$ goes:
- from the ALU to one of the 4 registers $s_0$-$s_3$ for writing the result of an operation.

## 2.1 Connection of registers to data buses

Figure 2 shows two registers $s_1$ and $s_2$ with their control block for writing to the registers or reading from the registers to either bus$_A$ or bus$_B$.

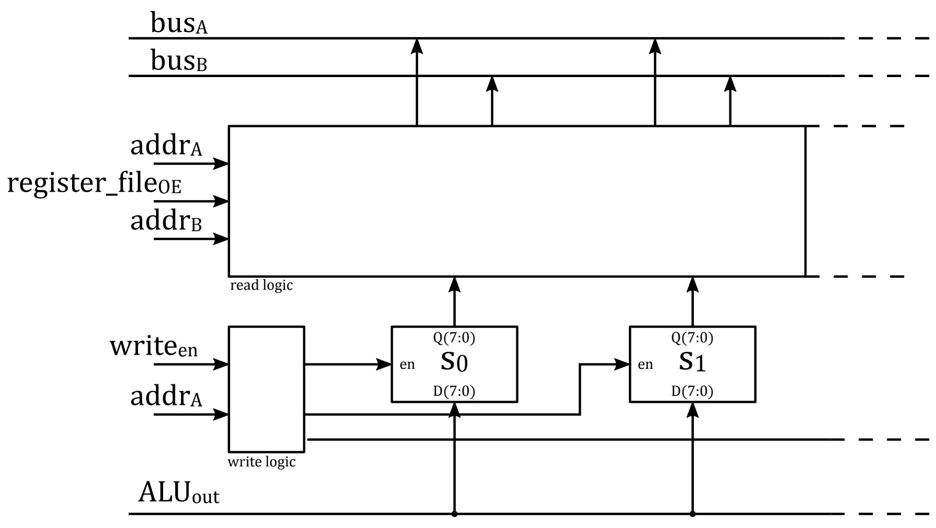


Figure 2 - Data registers $s_X$

The circuit of Figure 2 allows so that the outputs of the registers can be connected to both buses bus$_A$ and bus$_B$. The system created should allow one register to place its data on bus$_A$ and another to do so on bus$_B$.

The numbers addr$_A$ and addr$_B$ indicate which register transmits its information on bus$_A$, respectively bus$_B$. The signal register_file$_{\mathrm{OE}}$ indicates whether data from the selected register should be brought onto bus$_B$ and prevents a conflict with data coming from the I/O port or the instruction block.

Similarly, write$_{\mathrm{en}}$ and addr$_A$ signal is used to write to the registers. Thus, an operation whose first operand is the register selected by addr$_A$ will write its result into that same register. As seen on the Assembler code in Listing 1, the first operand is register $s_0$ and the result of the operation will be written into that same register.

Develop the read and write logic for the registers $s_0$, $s_1$, $s_2$, and $s_3$ in the bloc **`MIB/aluAndRegister`**.

## 2.2 Connection to the input/output bus

When reading data from outside, it is necessary to activate the control signal port$_{\mathrm{in\_OE}}$, and then the data from bus port$_{\mathrm{in}}$ is written onto bus$_B$. When writing data outside, the data from bus$_A$ is written onto bus port$_{\mathrm{out}}$, and an external signal to the ALU, write$_{\mathrm{strobe}}$, is activated via the testbench. This allows this data to be recorded in a register external to the $\mu$Processor.

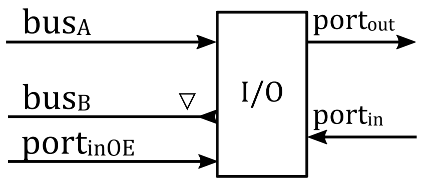


Figure 3 - Input/output block

Develop the internal schematic of the I/O bloc in **`MIB/aluAndRegisters`** from Figure 3.

## 2.3 Data coming from the instruction

For the second operand of an operation, a constant value can be encoded in the instruction and brought onto bus$_B$ of the ALU. The block managing this transfer is represented in Figure 4.

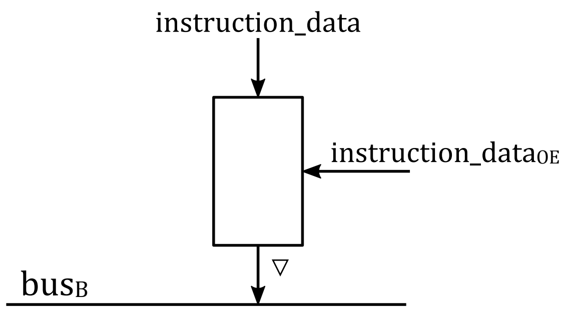


Figure 4 - Data coming from the instruction

```
  LOAD s0 10  # Loads the constant 10 to register s0. s0 = 10
#  ^   ^  ^
#  |   |  +-- immediate (constant) value 10
#  |   +----- First operand, register s0
#  +--------- Operation, LOAD
```

Listing 2 - Example of a assember instruction **`LOAD`**.
The first operand is register $s_0$, the second value is an immediate (constant).

Develop the internal schematic of the instruction bloc from Figure 4 in **`MIB/aluAndRegisters`**.

## 2.4 Implementation

Based on the previous points, the internal bus circuit of the $\mu$Processor is completed in **`MIB/aluAndRegisters`**.

# 3 Software implementation of a serial port

## 3.1 Serial transmission

Figure 5 shows the timing of the serial transmission of a data word.

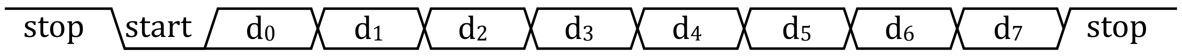


Figure 5 - Serial transmission

In a serial transmission, by default the signal on the bus hag a high level (logic 1). The transmission starts with a start bit (logic 0), followed by the data bits, and ends again with a stop bit (logic 1). The data bits are sent one after the other, starting from the least significant bit (LSB) to the most significant bit (MSB).

In our application, the serial signal is transmitted on the least significant bit of the $\text{port}_{\text{out}}$ bus resp. $\text{port}_{\text{out}}[0]$. In the test bench, this bus is connected to an external register **MIB_test/MIB_tb/I2**. The $\text{write}_{\text{strobe}}$ command from the test bench controls writing to this register.

## 3.2 Algorithm

The algorithm to be programmed is as follows:

```
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
```

Listing 3 - Software implementation of the serial tranmission protocol

Study and understand the algorithm of the serial transmission protocol Listing 3.

## 3.3 Implementation

Each line or instruction needs to be implemented in the testbench Tester bloc.

Complete the test bench tester **`MIB_test/MIB_tester`** to perform the instruction sequence for serial transmission Listing 3.

It is important not to leave any busses in a high impedance state. Program the algorithm so that there is always a signal on bus$_A$ and bus$_B$, even when no information is being sought from them.

## 3.4 Simulation

Simulate the Testbench **`MIB_test/MIB_tb`** with the simulation file **`$SIMULATION_DIR/MIB.do`**.

How many bits and what datavalue is being transmitted?

# 4 Checkout

This is end of the labo, you have successfully built the internal structure of the minimalistic $\mu$ Processor Xilinx PicoBlaze. Before leaving the laboratory, ensure you have completed the following tasks:

- ☐ Circuit Design
  - ☐ Verify that the block **`MIB/aluAndRegisters`** have been designed and tested with features mentioned.
- ☐ Simulations
  - ☐ Ensure that you have understood the serial transmission algorithm Listing 3.
  - ☐ The specific instructions were implemented in **`MIB_test/MIB_tester`**.
  - ☐ The value and number of bits transmitted is read from the simulation.
- ☐ Documentation and Projectfiles
  - ☐ Ensure all steps (design, convertions, simulations) are well-documented in your lab report.
  - ☐ Save the project to a USB stick or the shared network drive (**`\\filer01.hevs.ch`**).
  - ☐ Share files with your lab partner to ensure work continuity.

# Glossary

***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