ESD - Elettronica dei Sistemi Digitali

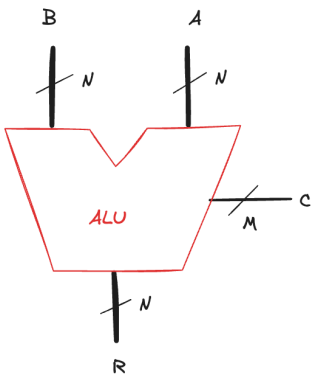
Exercises on Combinational Building Blocks

Riccardo Berta

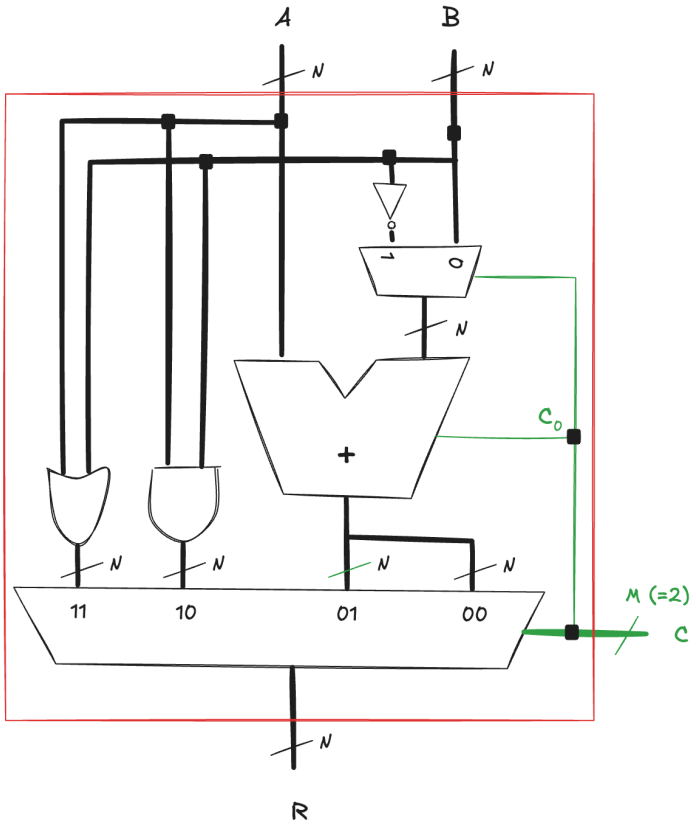
2025.11.13

1 Arithmetic/Logical Unit Exercises

Realize using DEEDS and VHDL a simple Arithmetic/Logical Unit (ALU) as described during the lectures. The ALU should be able to perform 8-bit addition, subtraction, bitwise AND, and bitwise OR operations:



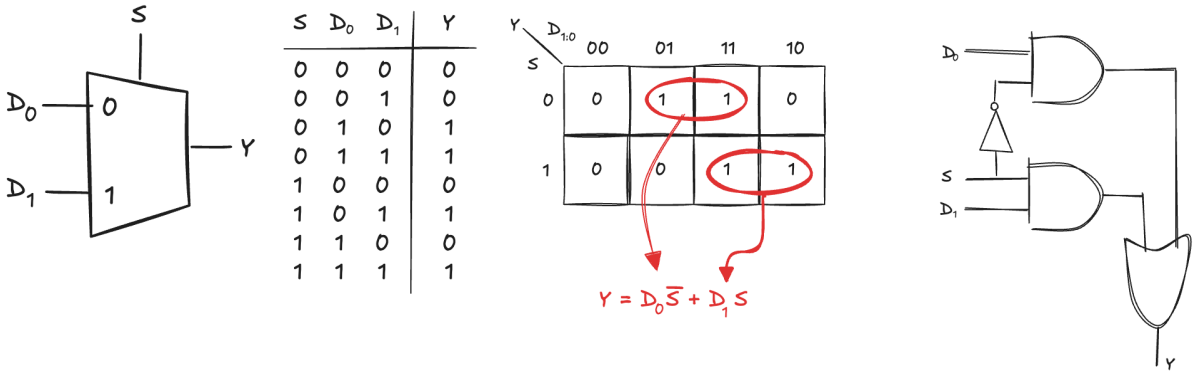
Control C_1	Control C_0	Funtion
0	0	Add
0	1	Subtract
1	0	AND
1	1	OR



To build the ALU, we combine several key components: a 2-bit control decoder that selects the desired operation, a 4-to-1 multiplexer that chooses the final output, an 8-bit adder for arithmetic, and 8-bit AND and OR logic blocks for bitwise operations.

1.1 2-bit Multiplexer

Create a 2-to-1 multiplexer (mux), a circuit that selects one of two input signals and forwards it to the output depending on the value of a single select line.



Design the circuit using DEEDS and simulate its truth table to verify its functionality.

[DEEDS]

DEEDS allows us to create custom block components that can be reused in larger designs. Create a new block component for the multiplexer.

[DEEDS]

However, the multiplexer we need must handle two 8-bit inputs, so we must build an 8-bit version of the 2-to-1 mux. We can construct it by reusing the 1-bit mux as a building block and instantiating eight of them in parallel. DEEDS also provides bus-based components that simplify this task, allowing us to bundle the signals and connect the mux array more efficiently. Search the DEEDS component library and find a bus-based 2-to-1 multiplexer to use in our design.

[DEEDS]

Develop the VHDL implementation of the 8-bit 2-to-1 multiplexer and add the source file to a Vivado project configured for the Artix-7 (xc7a200tfbg484-1) FPGA device.

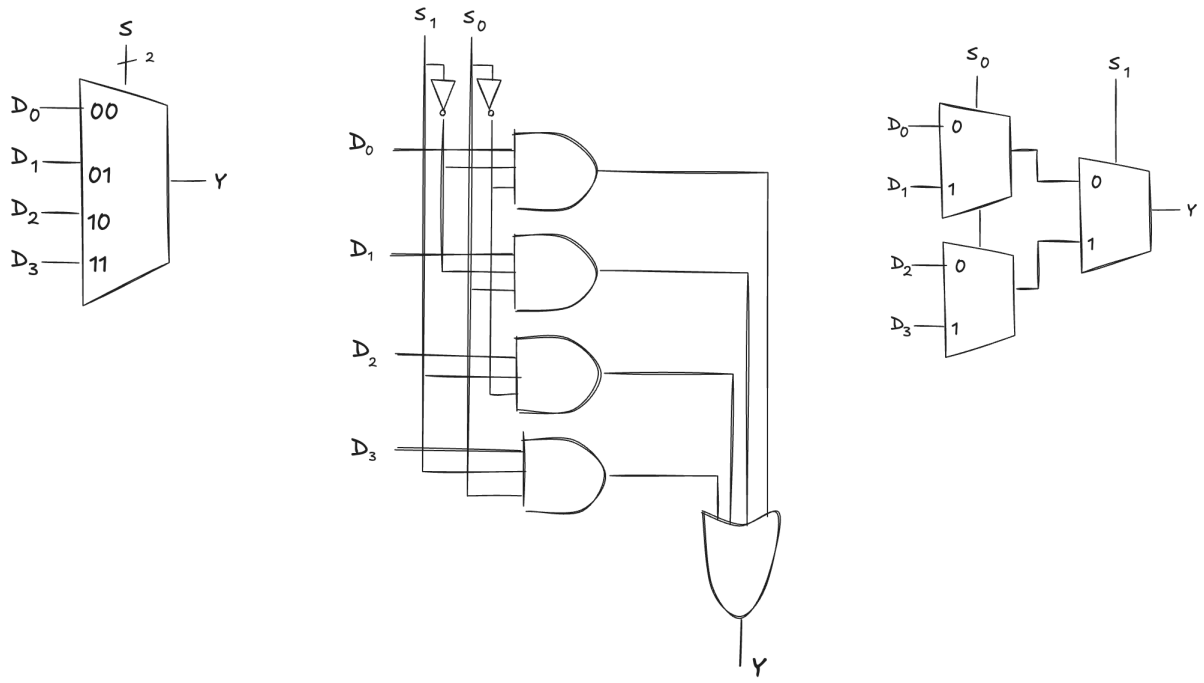
```
[VHDL]
```

Create a testbench to validate the behavior of the 2-bit multiplexer. Simulate the design and verify that it produces the correct output for every possible input combination.

```
[VHDL]
```

1.2 4-bit Multiplexer

We need a 4-to-1 multiplexer to select the correct value to drive to the output based on the input control signals:



Design, using DEEDS, a 1-bit multiplexer that chooses one of four inputs, with the two select signals determining which input is routed to the output.

[DEEDS]

Build the DEEDS block component that implements the 4-to-1 multiplexer, so it can be reused as a module in more complex designs.

[DEEDS]

As before, we need to route four 8-bit input signals, so we must build an 8-bit version of the 4-to-1 multiplexer. This can be done by reusing the 1-bit 4-to-1 mux as a building block and instantiating eight of them in parallel. DEEDS also provides bus-based components that simplify this process and allow us to bundle and connect the signals efficiently. Find the bus-based 4-to-1 multiplexer in the DEEDS component library to incorporate into our design.

[DEEDS]

Develop the VHDL implementation of the 8-bit 4-to-1 multiplexer and include it in the Vivado project.

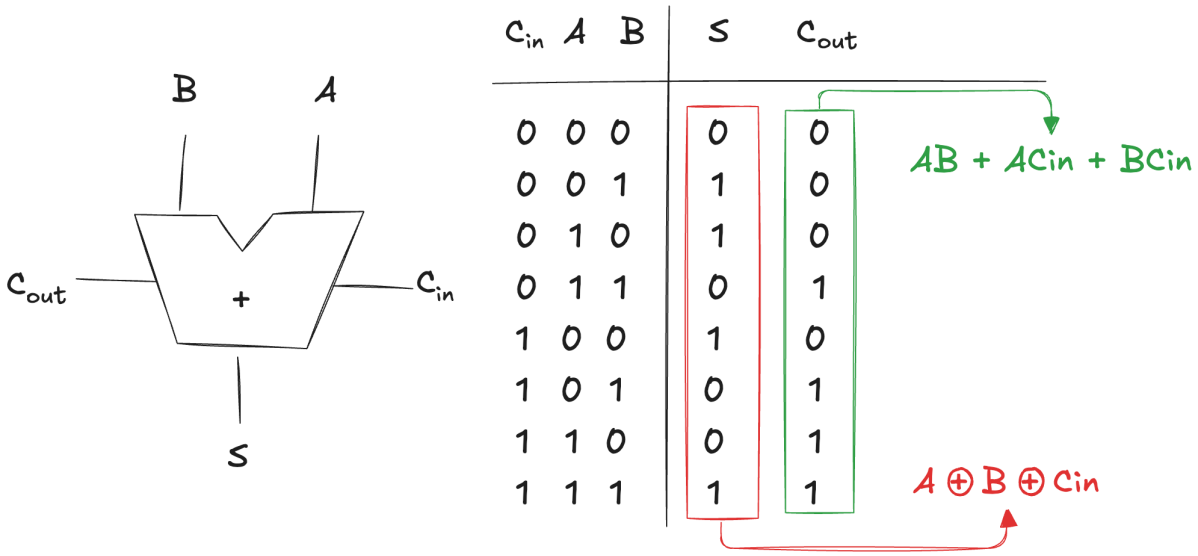
```
[VHDL]
```

Write a testbench to verify the functionality of the multiplexer. Run a simulation and check that the circuit produces the correct output for every possible combination of inputs.

[VHDL]

1.3 8-bit Adder

We need an 8-bit adder to support both addition and subtraction operations. First, we need to design a full adder— a circuit that adds two input bits and a carry-in bit, producing a sum bit and a carry-out bit.



Design the full adder using DEEDS, simulate it, save it as a block diagram.

[DEEDS]

[DEEDS]

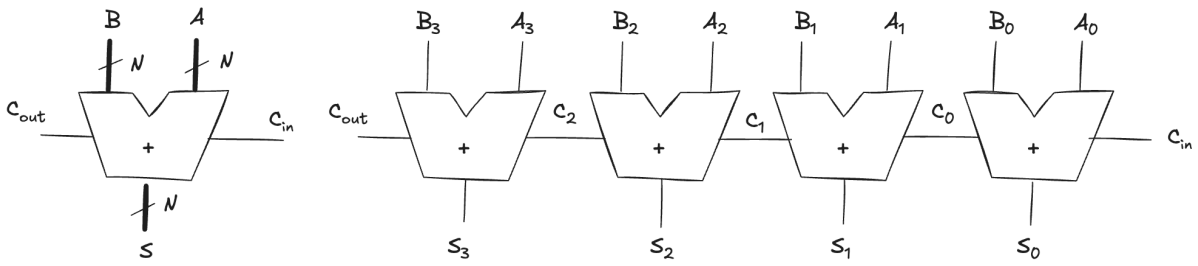
Write the VHDL description of the full adder and include it in the Vivado project.

[VHDL]

Write a testbench to verify the functionality of the full adder. Simulate the design and ensure that it behaves as expected for all possible input combinations.

[VHDL]

To support 8-bit addition, we must build an 8-bit adder. A straightforward approach is to use a ripple-carry structure, connecting eight full adders in series so that each carry-out feeds the next stage.



Using the full adder as a building block, design the 8-bit ripple-carry adder in DEEDS and simulate its behavior. As an intermediate step, begin by implementing a 4-bit version.

[DEEDS]

As the number of bits increases, using individual wires makes the schematic difficult to read. To keep the design clean and organized, redesign the 4-bit adder in DEEDS using bus connections.

[DEEDS]

Unfortunately, DEEDS does not allow creating new components that contain custom-made blocks. However, it already provides an adder component that we can use in our design. Search the DEEDS component library for an 8-bit adder and incorporate it into our design.

[DEEDS]

Write the VHDL description of the ripple-carry adder using the full adder you previously created, and add it to our Vivado project.

```
[VHDL]
```

Write a testbench to verify the functionality of the ripple-carry adder. Simulate the design and ensure that it behaves as expected for all possible input combinations.

```
[VHDL]
```

1.4 Putting everything together

Using the DEEDS adder, multiplexers, and the AND/OR gates, build the complete ALU design. Simulate the ALU in DEEDS to verify that each operation behaves correctly based on the control signals.

[DEEDS]

Using the VHDL enetities created in the previous steps, write the VHDL description of the complete ALU and add it to our Vivado project.

```
[VHDL]
```

Write a testbench to verify the funtionality of the complete ALU. Simulate the design and ensure that it behaves as expected for all possible input combinations and control signal configurations.

```
[VHDL]
```

1.5 Next Steps

Explore additional operations to enhance the ALU's capabilities, such as adding flag outputs (zero, carry, overflow) or other bitwise operations like XOR and NAND. Another improvement could be extend the ALU to more than 8 bits, such as 16 or 32 bits, to handle larger data sizes. In that case, consider using more advanced adder architectures, like carry-lookahead adders, to improve performance.