SECTION OF ELECTRICAL ENGINEERING

EPFL STI – SEL
ELG
Station nº 11
CH-1015 Lausanne

Téléphone : +4121 693 13 46
Fax :
E-mail : alexandre.levisse@epfl.ch
Site web : https://sti.epfl.ch/fr/sel/

Fundamentals of VLSI – project Full Custom, session 1
SEL October 2024

Full Custom Project
Session 1 : Schematics

1. Objectives of this project

At this point, you know the basic design techniques using the Virtuoso suite : Schematic edition, various kind of simulations, physical layout design and verification.

Thereby, we now propose you to evaluate and practice your newly acquired skills on a design project. This design project consists in designing a combinatorial 8-bit Arithmetic Logic Unit (ALU) from the idea to the layout and your final design must meet some speed and area specifications.

This project spans over 4 class (Week 1: Thursday 17/10, Friday 18/10, Week 2 : Friday 1/11 and Week 3: Friday 8/11), and is graded.

Each session will have a supporting document giving you sone guidelines and informative deadlines for you to make sure you are not getting late. This project requires a non-negligible amount of efforts, make sure you do not get late with regard to the proposed schedule. As an engineer, you must remember that time-management is one of your duties.

Here is a proposal for the project schedule. While you can take some freedom with it, try not to deviate too much. The support documents will consider this schedule :

- Week 1 – Thursday : Schematic edition of the 8bit ALU. Identification of the critical paths and preparation of the testbenches.
- Week 1 – Friday : Verification of the functionality through simulation. Evaluation of the critical paths, and comparison with the expected critical path. Optimization of the sizing to meet the specifications.
- Week 2 – Friday: Floor planning and definition of the layout strategy. Layout of the 1bit cells and integration in a 8bit ALU. Verification with DRC/LVS.
- Week 3 – Friday : Simulation of the post-layout netlist and critical path characterization. If you still have time, re-optimization of the circuit. Results presentation in a slide set.

2. Specifications and Grading

The grading takes into account the following metrics.

Mandatory Fail/Pass Criterions (if not met, the grade for this part will be below 4):

- Speed of the ALU post-PEX. The time from the rising edge of the input to the latest output transition should be below 1ns.
- Area of the design. The design must be smaller than 500um2.
- The design must have correct functionality.
- No DRC/LVS errors

Additional metrics :

- Efficient designs get a better grade
  - We apply a delay*area product to evaluate the designs
- Floorplan quality/justification
- Good practice in terms of routing, power management, substrate biasing

3. Schematics

The block diagram of the 8-bit ALU that you will design in this project is shown in Figure 1. It consists of three building blocks: a logic block, an adder and a multiplexer. This system has two 8-bit input signals *A* and *B*. These signals are the inputs of the logic block and the adder. Logic block performs a logic function determined by the control signal *CTR,* over the inputs *A* and *B*. The adder performs the addition of the two inputs. Finally, the multiplexer selects the output signal *Z* of the ALU, as the output of either the logic block (X) or the adder (S), depending on the select signal *SEL*. The two blocks (logic and adder) will be described in details in the following sections.

ℹ ***Important!*** *Make sure that the naming of all of your pins in the design is* ***exactly the same*** *as given in this material. In other words, pin* ***A<7:0>*** *should be named* ***A****, and not:* ***a, x, a_in*** *or anything similar.* ***Any differences from the defined naming will be penalized in the final grad****e.* ***Also please pay attention that in Cadence you should use '<' and '>' signs in the pin names not '[' or ']' signs.***

Naming conventions : in your EDATP library, you can call the cellviews as follows LOGIC_1BIT, ADDER_1BIT, ALU_1BIT, ALU_8BIT. For the testbenches, you could call them LOGIC_1BIT_TB and so on.

Keep the naming convention constant along the design, this will make your life easier.

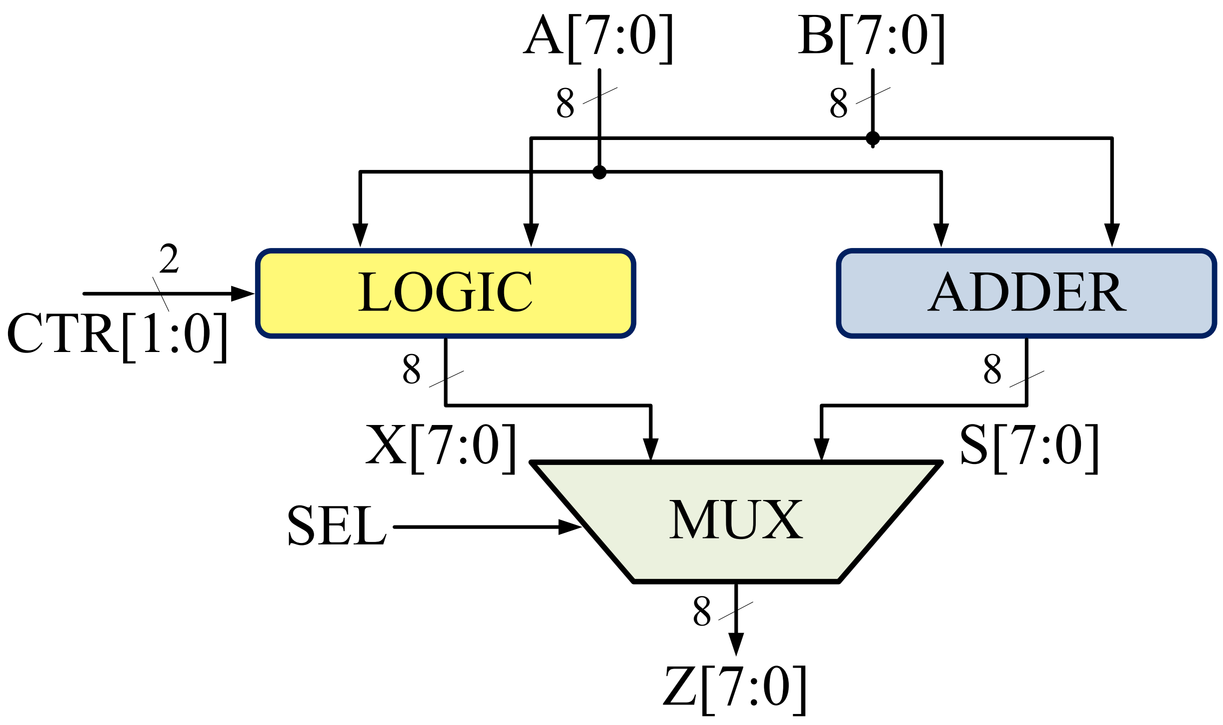


Figure 1 – Block Diagram of the 8-bit ALU

3.1. The logic block

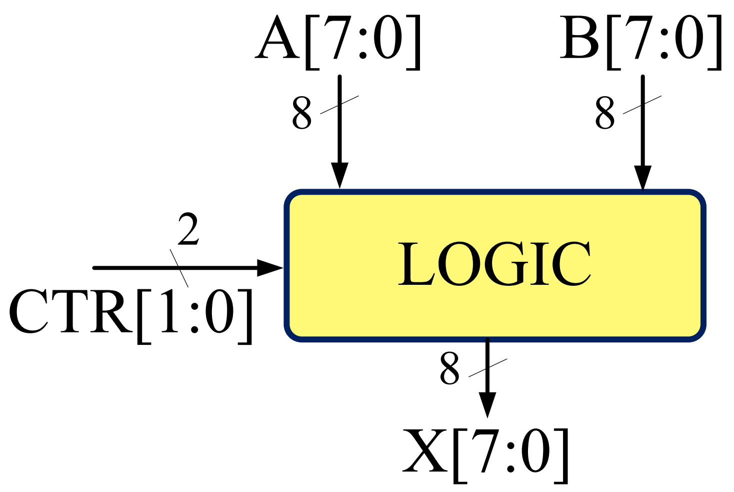


Figure 2 – Interface of the Logic Block

As mentioned, the logic block performs one logic function out of four, depending on the control signal *CTR*. These functions are described in the following truth table.

CTR[1]	*CTR[0]*	*X[n]*
0	*0*	$X[n] = \overline{A[n] \cdot B[n]}$
0	*1*	$X[n] = \overline{B[n]}$
1	*0*	$X[n] = \overline{A[n]}$
1	*1*	$X[n] = \overline{A[n] + B[n]}$

Table 1 – Truth Table of the Logic Block

The simplest way to build the logic block is to use logic gates for each function and then choose the correct output using a multiplexer as shown in Figure 3.

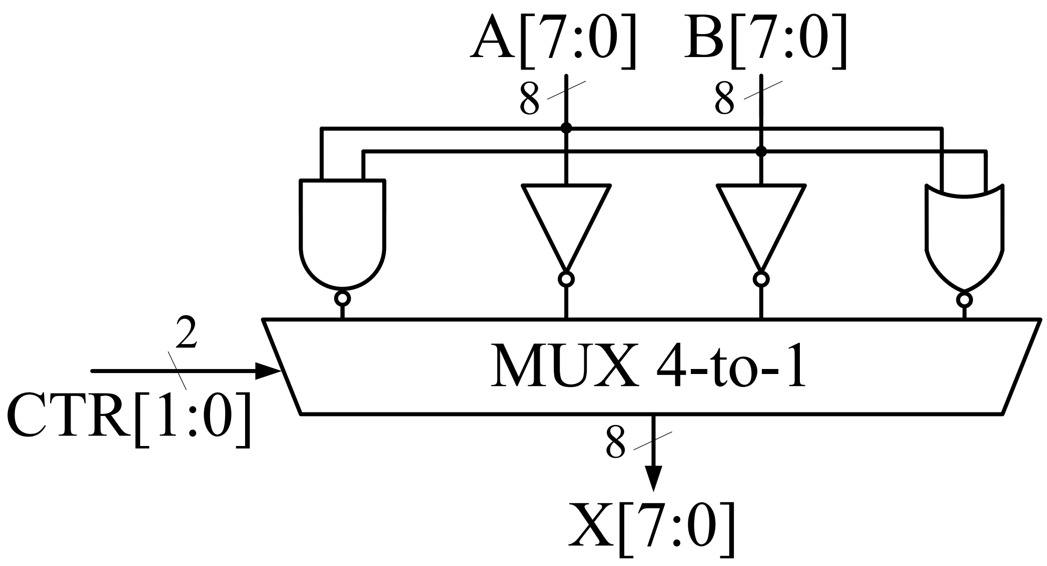


Figure 3 – Schematic of the Logic Block

This solution contains a large number of transistors. For example, if a 4-to-1 multiplexer as the one developed in laboratory session 1 is used, the complete circuit would contain 48 transistors. **Think about the critical path and decide if you want to go for speed or area for this block.**

In order to minimize the size of the block, a solution would be to create a Karnaugh map of the logic block, so that the Boolean expression of the logic function could be obtained, as you did in the exercises. Then, a circuit at the transistor level would be easily determined.

Reminder

- *The "product" term in the Boolean expression corresponds to the* ***series*** *connection in the NMOS network.*
- *The "sum" term in the Boolean expression corresponds to the* ***parallel*** *connection in the NMOS network.*
- *The PMOS network can be constructed as the exact* ***dual*** *of the NMOS network.*

P-channel side:	*AND → Parallel*
	OR → Series
N-channel side:	*AND → Series*
	OR → Parallel

If you decide to proceed with the area-optimized circuit, you muse fill-in the following Karnaugh map using the truth table from Table 1.

X[n]		***A[n] B[n]***			
		00	***01***	***11***	***10***
CTR[1] CTR[0]	***00***				
	01				
	11				
	10				

Table 4 – Karnaugh Map

Then you should establish the Boolean expression (or its inverted form) of the logic function.

$X[n]=$

And translate it into transistors.

At the end of this section, you should have defined a schematic diagram for a 1bit logic block. Do not do the 8 bit already.

3.2. The adder

1-bit addition

The 1-bit adder that will be used for the 8-bit addition is the full adder. As it is shown in Figure 7, the 1-bit full adder is a 3-input and 2-output block. The inputs are the two signals to be summed, *a* and *b*, and the carry c_i, which derives from the calculations of the previous digit (as shown in Figure 10). The outputs are the result of the sum operation *s* and the resulting value of the carry bit c_o. More specifically, the sum and carry output are given by the following equations and displayed in details in Table 6.

$$s = a \oplus b \oplus c_i$$

$$c_o = ab + ac_i + bc_i = ab + (a \oplus b)c_i$$

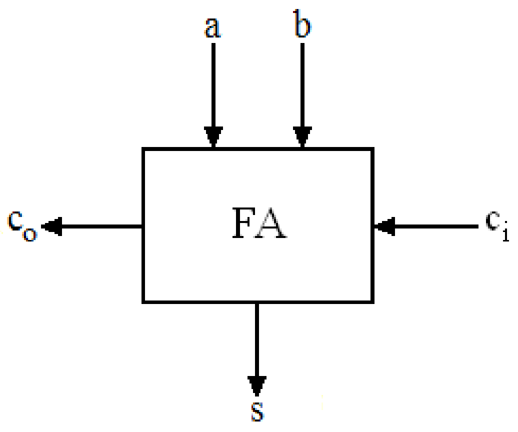


Figure 7 – The 1-bit Full Adder

a	b	c_i	s	c_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 6 – Truth Table of the Full Adder

Circuit of the 1-bit full adder

There are many ways to make a full adder. For the logic, you did make an optimized CMOS circuit using a Karnaugh map translation into transistors. Here, we propose that you make the full adder based on logic gates.

Generally, building a complex logic gate as you did for the logic will be denser, however, it will tend to be slower. While this approach used to be un-advised and un-used for old technology nodes, with the end of voltage scaling, sub-90nm technologies tend to be more efficient with less transistor stacks. Also, with technology scaling designing logic gates becomes more complex, reusing existing logic gates makes a lot of sense, advocating for standard cell based design flows.
Figure 10 presents a generic full adder composed of 2 XOR, 2 AND, and 1 OR gate. Think about potential optimizations for the Cout generation part of the adder, and prefer the use of NAND/NOR instead of AND/OR gates.

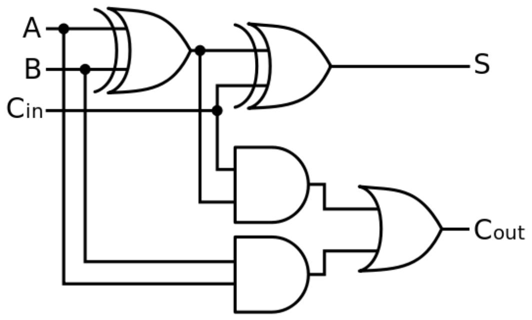


Figure 10 – example schematic of a gates-based full adder.

For the XOR gate, **do NOT use dynamic gates**. The densest and easy to design static gate you can achieve has 12 MOS transistors (google is your friend – be careful with the sizing – and don't take everything you find there for granted[1]). Standard XOR gates based on logic gates can be optimized down to 16 MOS transistors. You can take inspiration from the logic gates from the standard cell library. Do NOT use the gates from the standard cells (we will know you did …).

Create a schematic for the 1bit Full Adder cell on virtuoso and save it. You may need to create schematic cells for the XOR, NOR and NAND gates as well. For now, do the sizing of the cells with a basic balancing approach as you did learn it in the class.

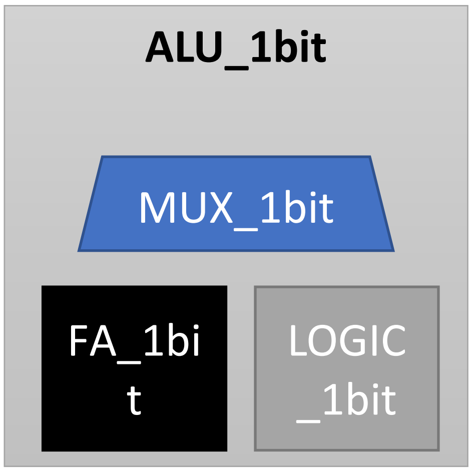
3.3. THE MULTIPLEXER

For each bit "slice", a 2-input 1-output multiplexer is used. The same multiplexer as the one in LAB1 can be used.

[1] "Don't believe everything you read on the internet" – Abraham Lincoln

4. Assembly of the blocks

One good way to proceed is to organize the 1-bit ALU as presented below. And then assemble 8 of them by connecting the Cin of the n-1th block to the Cout of the nth block.



- ✓ Create a new schematic in your **EDATP** library. Name it **ALU_1bit**.
- ✓ Draw the schematic of the 1-bit ALU block using the **LOGIC_1bit, FA_1bit and MUX_1bit** that you have already designed. Once you are finished drawing the schematic and creating the pins, create the symbol for the cell.
- ✓ Create a new schematic in your **EDATP** library. Name it **ALU_8bit**.
- ✓ Draw the schematic of the 8-bit ALU as shown in Figure 11. Use the **ALU_1bit** that is already designed. When you are finished drawing the schematic and creating the pins, create the symbol for the cell.
- ✓ ***Important!*** It has to be noticed that if *SEL = 0 → Z[7:0] = X[7:0]* and that if SEL = *1 → Z[7:0] = S[7:0]* (see Figure 1).
- ✓ ***Important!*** The name of the top-level cell has to be exactly **ALU_8bit.** Furthermore, all the pins in the top-level cell have to be named exactly as given in Figure 1. The functionality of all the pins has to be as discussed in the instructions. Please **do not** use logically inverted pins, or you may use them but the logical function (looking from the outside) has to remain the same.

5. IDENTIFICATION OF THE CRITICAL PATH

In this project, it seems obvious that the critical path will be mainly carried by the 8-bit adder.

8-bit addition

The 8-bit adder is then simply designed by connecting eight 1-bit full adders as shown in Figure 11 (Ripple carry adder). As it can be seen, the carry out of one digit is connected to the carry in of the next digit.

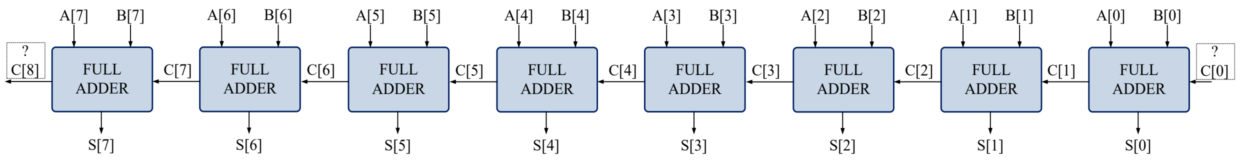


Figure 11 – The 8-bit Full Adder

The LSB and MSB full adders are slightly different from the other adders. You can either optimize them by removing C[0] and C[8] or leave the existing input C[0] grounded and the output C[8] floating.

Intuitively, it is easy to understand that the critical path for the delay occurs when the carry signal propagates from the LSB to the MSB. This situation is displayed in the figure below. The initial condition is:

A[7:0] = 11111111
B[7:0] = 00000000
(or the opposite)

Let's say the LSBs of B (B[0]) changes from 0 to 1. C[1] will switch to 1, inducing C[2] to switch to 1 and so on. This scenario will induce a propagation from the LSB to the MSB. Here C[8] is not checked. Thereby, the expected critical path is from A[0] (or B[0]) to S[7].

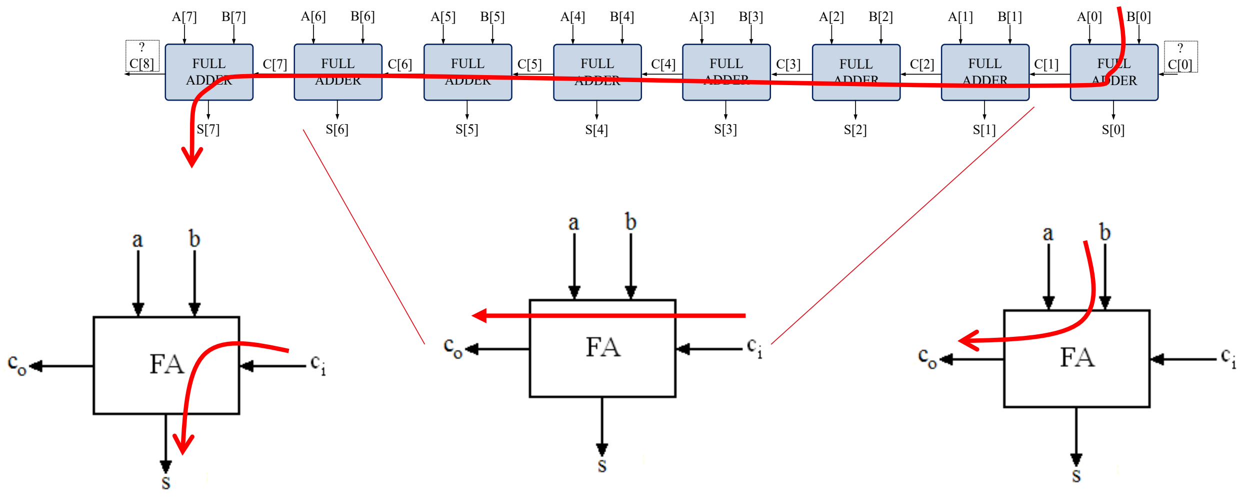


Figure 12 – Critical Path of the 8-bit Adder

Starting from the critical path of the 8-bit adder you can identify the critical path inside the 1bit adder. Note how , depending on their position inside the 8bit adder, they seemingly have different critical paths. identify these paths inside your 1 bit adder.

When optimizing the adder sizing later-on, it seems clear that the Ci → Co critical path will be the most important one to optimize in order to make the adder go faster. But keep in mind that A/B → Co and Ci → S could also be critical in some situations.

Remark *: could there be a situation where S[7] is not anymore the slowest output ?*

Define the list of testcases you need to simulate and characterize in order to catch the slowest possible timing of the 8bit adder.

6. TEST BENCH CREATION

Create testbench views for :

- The 1bit adder
- The 1bit logic block
- The 8bit ALU

For realistic simulations, put a 10fF capacitive load on the outputs of the 8Bit ALU.

For the testbench of the 1bit Adder, think about a realistic load you could put, to make your simulation realistic.

Hint 1 *: for your testbench input. From the analoglib, you could either use the vpulse or the vbit cell to generate your signals.*

Hint 2*: don't forget that the final circuit is not *just* the 8bit adder. There is also a 4to1 multiplexer. Do not neglect it when doing the sizing.*

Hint 3*: do not forget to test the functionality of the Logic block. While for the 8bit adder it makes sense to check only a subset of *carefully* selected cases, for the logic block, make sure that you test all the cases (it would be sad to fail if the functionality is not correct…).*