ESD - Digital Systems Electronics

Exercises

Riccardo Berta

2025.12.11

1 Data Representation Exercises

1.1 Exercise 1

What is the largest 32-bit binary number that can be represented with:

1.1.1 (a) Unsigned numbers

1.1.2 (b) Two's complement numbers

1.1.3 (c) Sign/magnitude numbers

1.2 Exercise 2

What is the smallest (most negative) 16-bit binary number that can be represented with:

1.2.1 (a) Unsigned numbers

1.2.2 (b) Two's complement numbers

1.2.3 (c) Sign/magnitude numbers

1.3 Exercise 3

What is the smallest (most negative) 32-bit binary number that can be represented with:

1.3.1 (a) Unsigned numbers

1.3.2 (b) Two's complement numbers

1.3.3 (c) Sign/magnitude numbers

1.4 Exercise 4

Convert the following unsigned binary numbers to decimal and to hexadecimal:

1.4.1 (a) 1110_2

1.4.2 (b) 100100_2

1.4.3 (c) 11010111_2

1.4.4 (d) 011101010100100_2

1.4.5 (e) 0110_2

1.4.6 (f) 101101_2

1.4.7 (g) 10010101_2

1.4.8 (h) 110101001001_2

1.5 Exercise 5

Convert the following hexadecimal numbers to decimal and to unsigned binary:

1.5.1 (a) $4E_{16}$

1.5.2 (b) $7C_{16}$

1.5.3 (c) $ED3A_{16}$

1.5.4 (d) $403FB001_{16}$

1.5.5 (e) $2B_{16}$

1.5.6 (f) $9F_{16}$

1.5.7 (g) $42CE_{16}$

1.5.8 (h) $E34F_{16}$

1.6 Exercise 6

Convert the following two's complement binary numbers to decimal:

1.6.1 (a) 1110_2 (4-bit)

1.6.2 (b) 100011_2 (6-bit)

1.6.3 (c) 01001110_2 (8-bit)

1.6.4 (d) 10110101_2 (8-bit)

1.6.5 (e) 1001_2 (4-bit)

1.6.6 (f) 110101_2 (6-bit)

1.6.7 (g) 01100010_2 (8-bit)

1.6.8 (h) 10111000_2 (8-bit)

1.7 Exercise 7

Convert the following decimal numbers to unsigned binary and to hexadecimal

1.7.1 (a) 42_{10}

1.7.2 (b) 63_{10}

1.7.3 (c) 229_{10}

1.7.4 (d) 845_{10}

1.7.5 (e) 56_{10}

1.7.6 (f) 75_{10}

1.7.7 (g) 183_{10}

1.7.8 (h) 754_{10}

1.8 Exercise 8

Convert the following decimal numbers to 8-bit two's complement numbers or indicate overflow.

1.8.1 (a) 24

1.8.2 (b) -59

1.8.3 (c) 128

1.8.4 (d) -150

1.8.5 (e) 127

1.8.6 (f) 48

1.8.7 (g) -34

1.8.8 (h) 133

1.8.9 (i) -129

1.9 Exercise 9

How many bytes are in a 32-bit word? How many nibbles are in the 32-bit word? How many bytes are in a 64-bit word? How many nibbles are in the 64-bit word? How many bits are in 2 bytes? How many bits are in 6 bytes?

1.10 Exercise 10

Convert the following decimal numbers to IEEE 754 single-precision format:

1.10.1 (a) 45.375_{10}

1.10.2 (b) -13.25_{10}

1.10.3 (c) 0.1_{10}

1.10.4 (d) -0.125_{10}

1.11 Exercise 11

Convert the following IEEE 754 single-precision numbers into decimal values:

1.11.1 (a) 0 10000010 01100000000000000000000

1.11.2 (b) 1 10000001 01000000000000000000000

1.11.3 (c) 0 01111101 10000000000000000000000

1.11.4 (d) 1 01111100 00000000000000000000000

1.12 Exercise 12

A particular modem operates at 768 Kb/sec. How many bytes can it receive in 1 minute?

1.13 Exercise 13

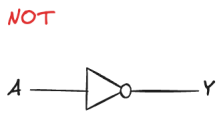
USB 3.0 can send data at 5 Gb/sec. How many bytes can it send in 1 minute?

2 Logic gates and circuits Exercises

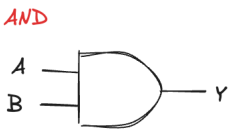
2.1 Exercise 1

Complete the truth tables of the following gates and use the DEEDS simulator to verify their correctness:

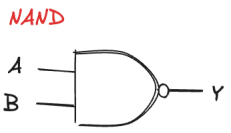
2.1.1 (a) NOT, AND, NAND



A	Y
0	
1	

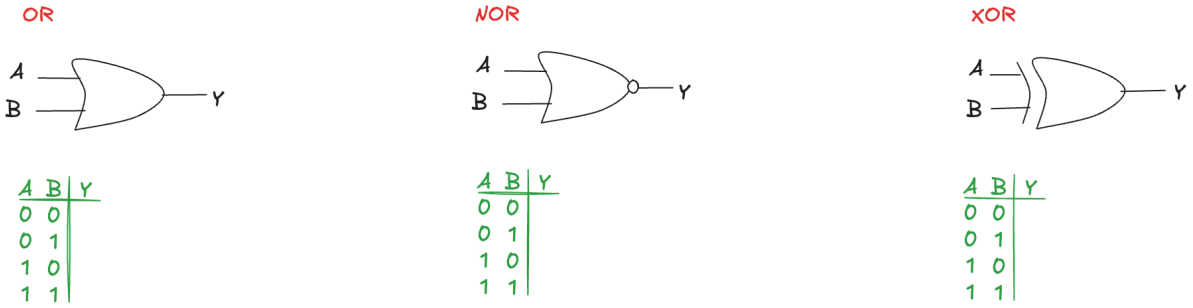


A	B	Y
0	0	
0	1	
1	0	
1	1	



A	B	Y
0	0	
0	1	
1	0	
1	1	

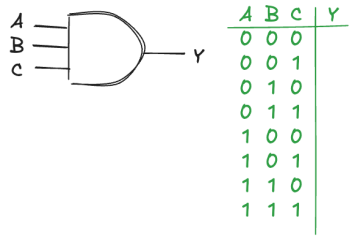
2.1.2 (b) OR, NOR, XOR



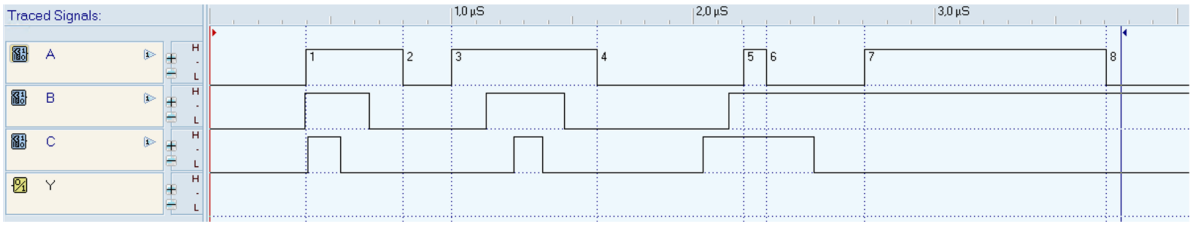
2.2 Exercise 2

Complete the truth table and the timing diagram of the following 3-inputs digital circuits, and use the DEEDS simulator to verify their correctness:

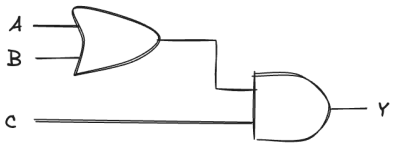
2.2.1 (a)



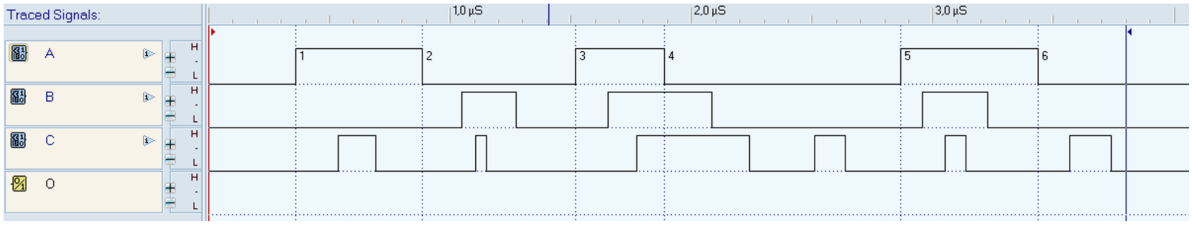
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



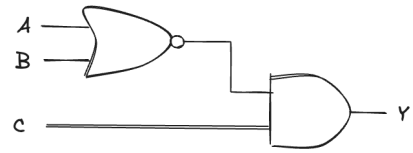
2.2.2 (b)



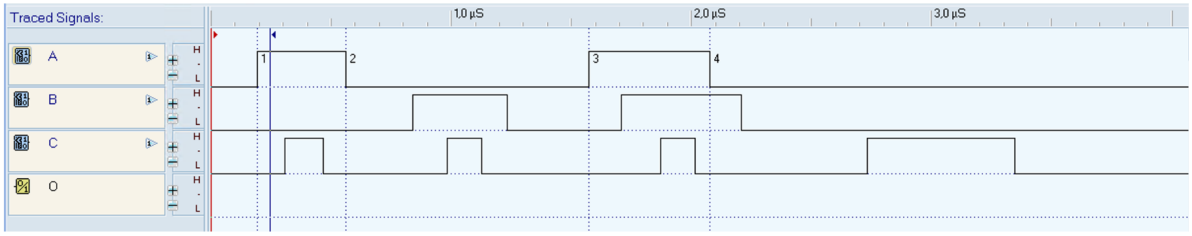
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



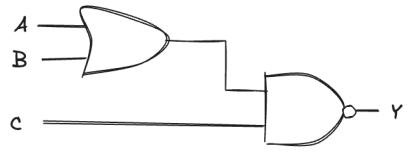
2.2.3 (c)



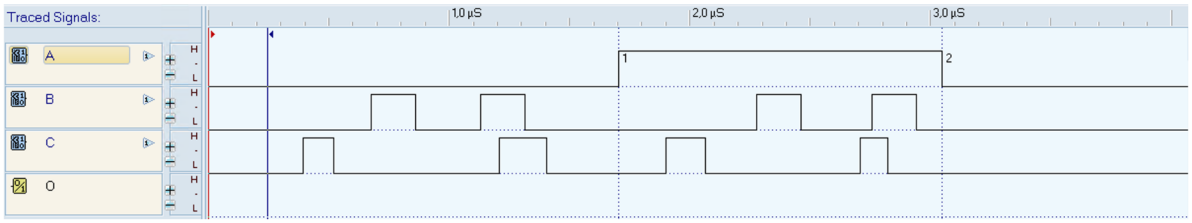
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



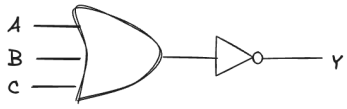
2.2.4 (d)



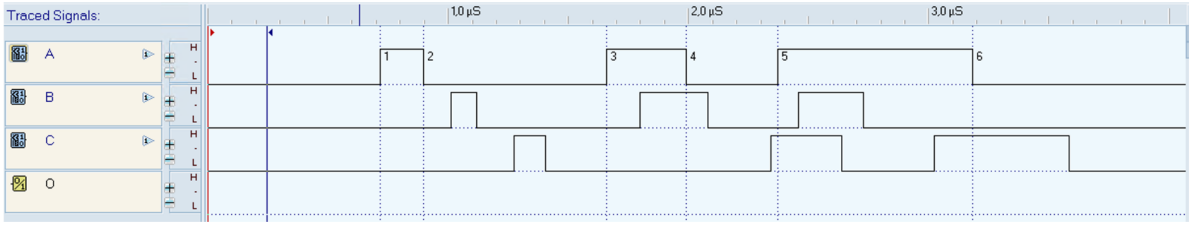
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



2.2.5 (e)



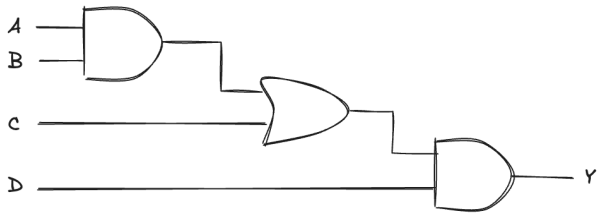
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



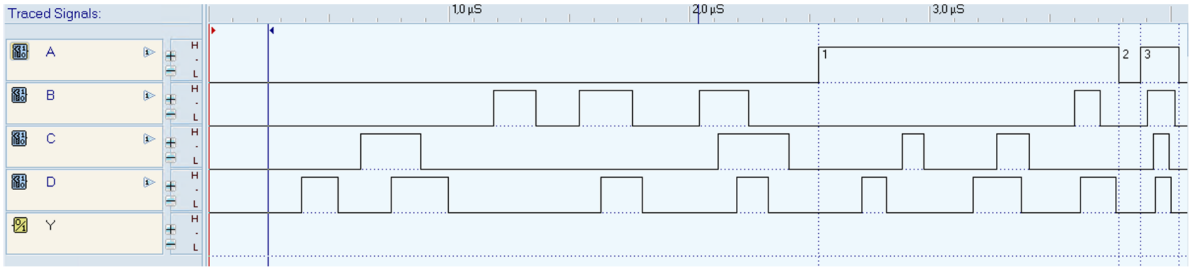
2.3 Exercise 3

Complete the truth table and the timing diagram of the following 4-inputs digital circuits, and use the DEEDS simulator to verify their correctness:

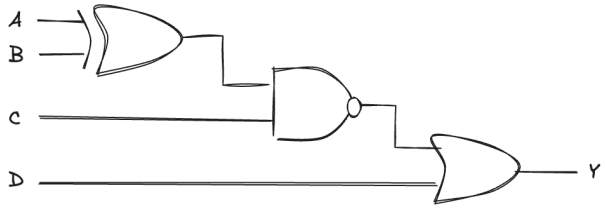
2.3.1 (a)



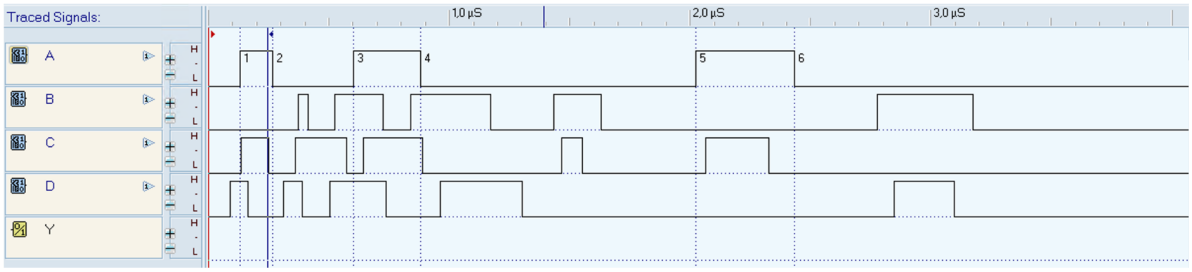
A	B	C	D	Y
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	



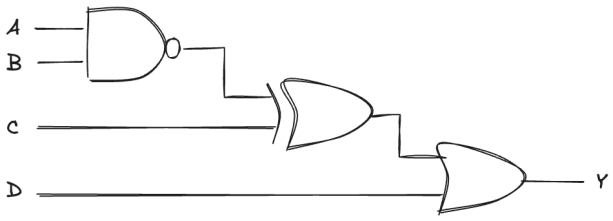
2.3.2 (b)



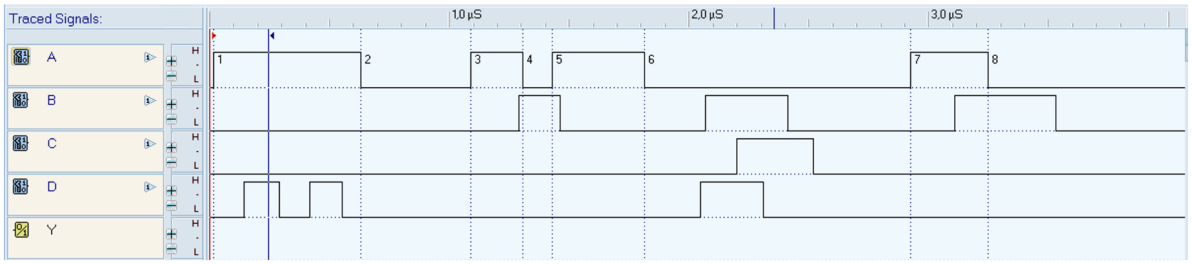
A	B	C	D	Y
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	



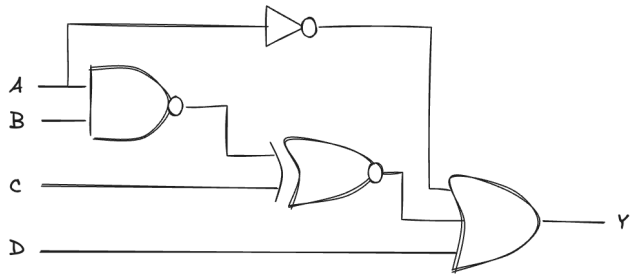
2.3.3 (c)



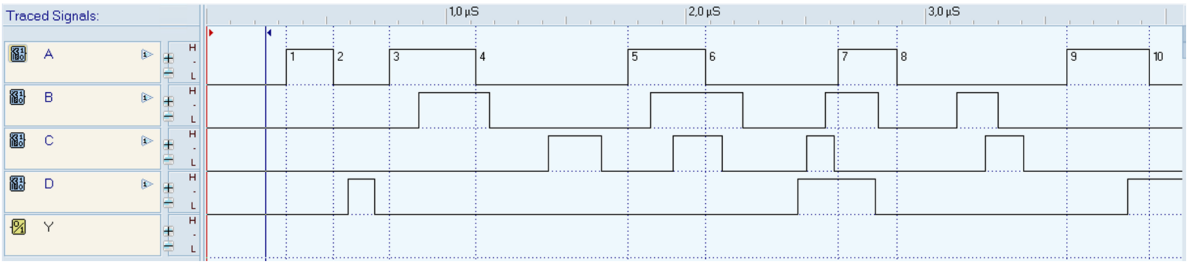
A	B	C	D	Y
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	



2.3.4 (d)



A	B	C	D	Y
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	



3 Boolean Algebra Exercises

3.1 Exercise 1

For each given truth table, write the Boolean equation in canonical **sum-of-products** form. Then simplify the expression to its minimal form using the theorems of Boolean algebra. Finally, implement the corresponding combinational circuit and verify its correctness by simulating it in DEEDS

3.1.1 (a)

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

3.1.2 (b)

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

3.1.3 (c)

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

3.1.4 (d)

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

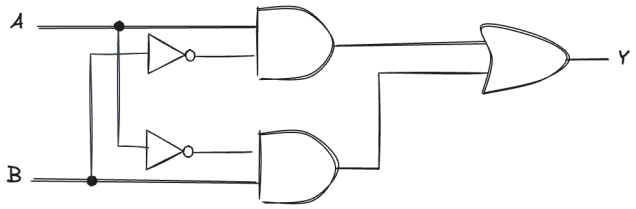
3.1.5 (e)

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

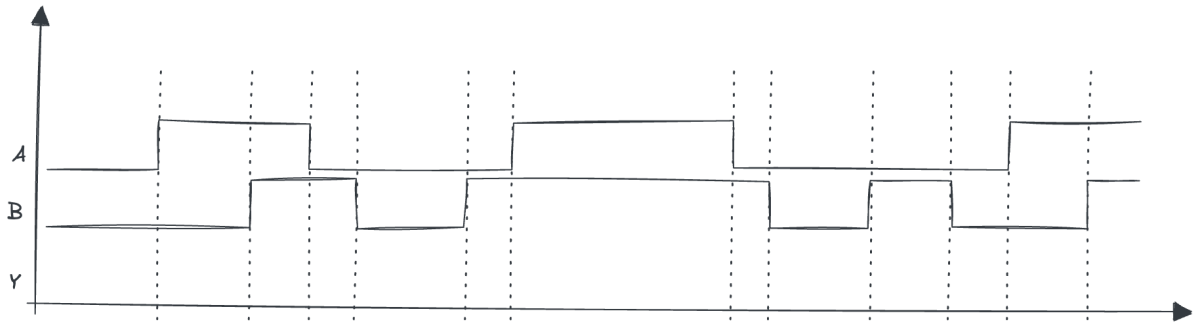
3.2 Exercise 2

Complete the truth table and the timing diagram of the following digital circuits. Then derive the Boolean equation in canonical **sum-of-products** form and simplify it to its minimal expression using Boolean algebra theorems. Finally, implement the corresponding circuit in DEEDS and verify the correctness of your solution.

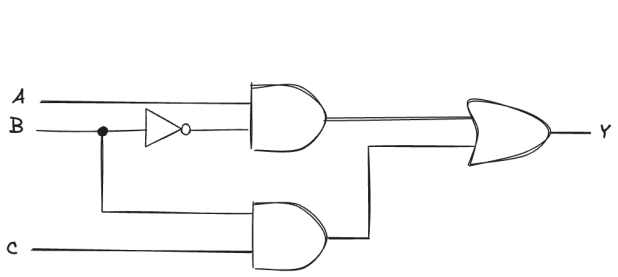
3.2.1 (a)



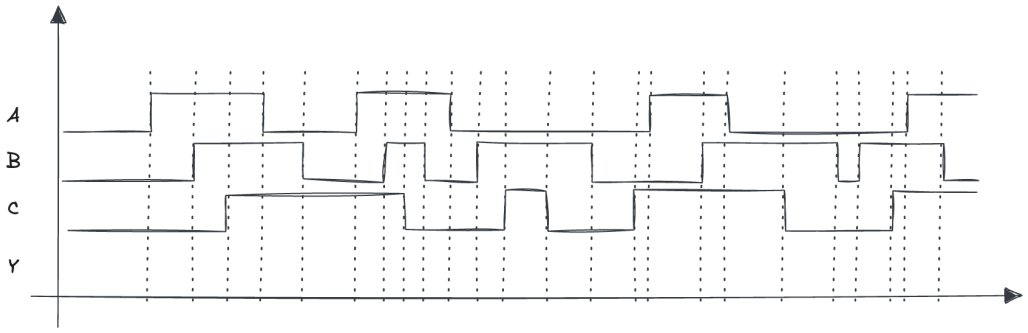
A	B	Y
0	0	
0	1	
1	0	
1	1	



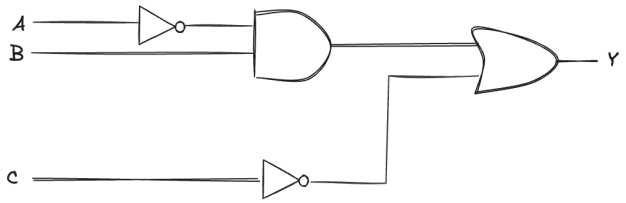
3.2.2 (b)



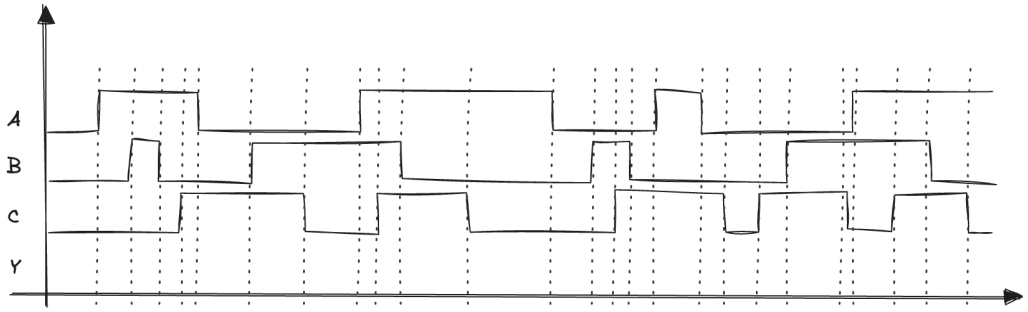
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



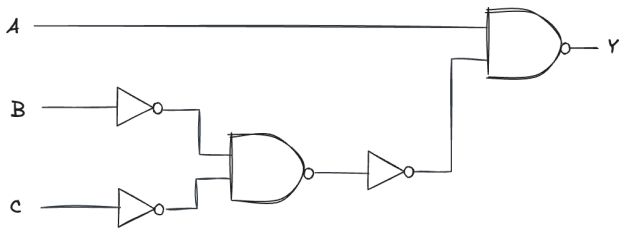
3.2.3 (c)



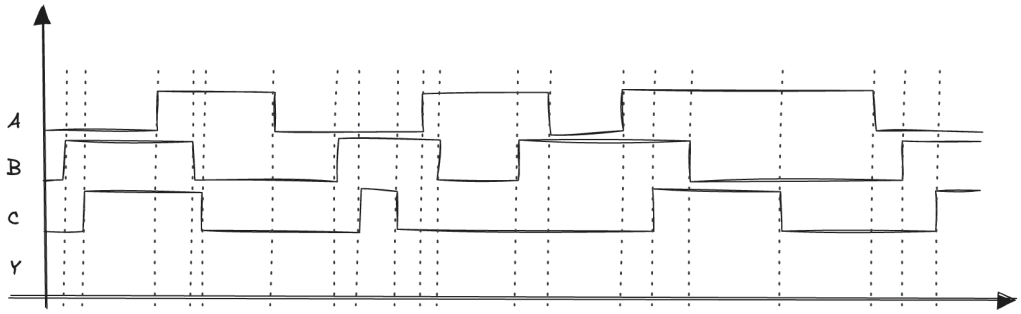
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



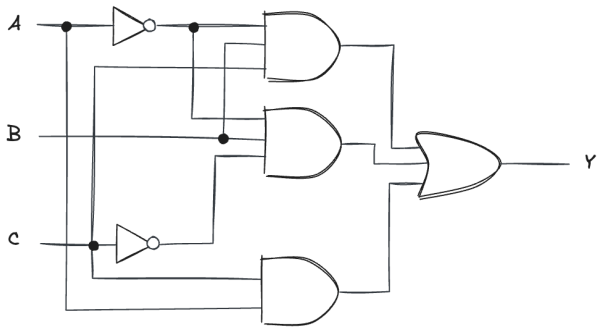
3.2.4 (d)



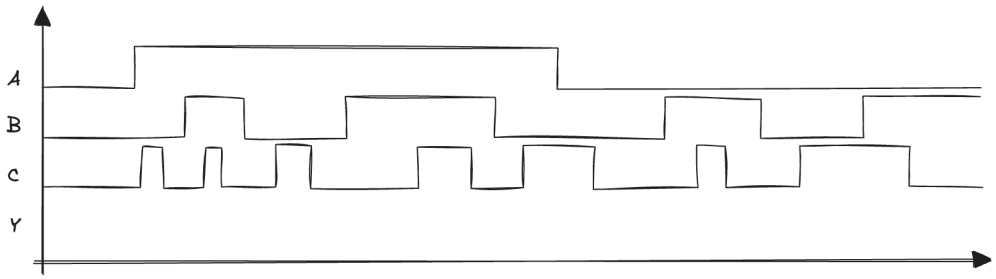
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



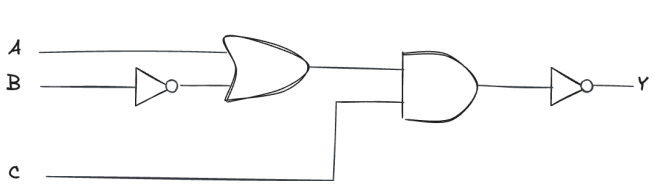
3.2.5 (e)



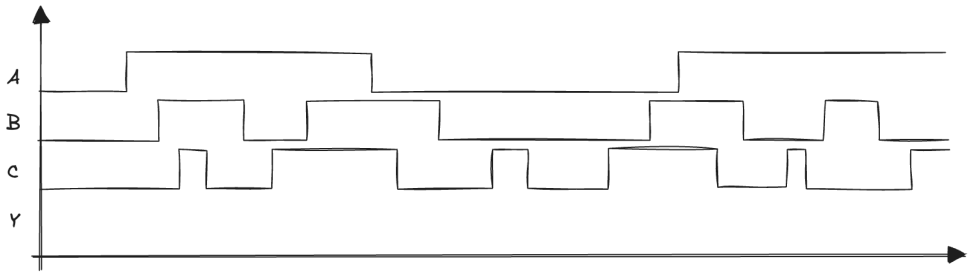
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



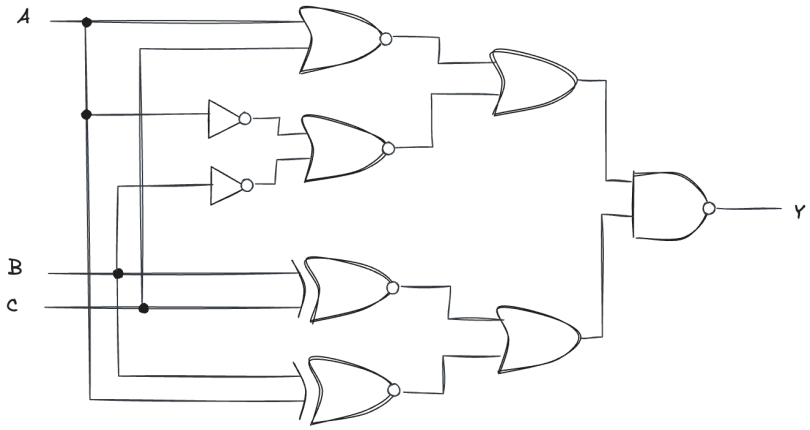
3.2.6 (f)



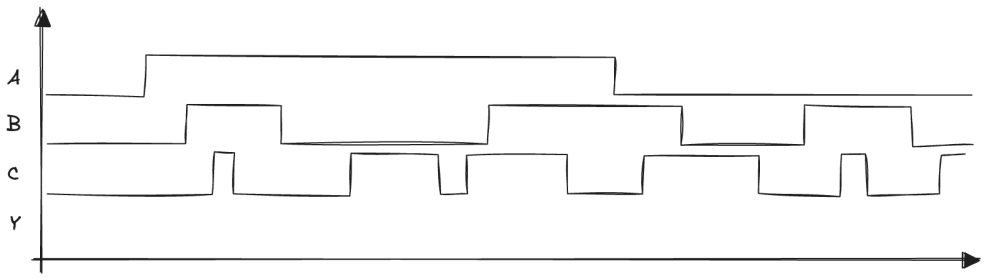
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



3.2.7 (g)



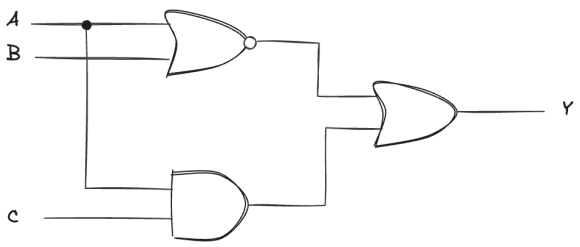
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



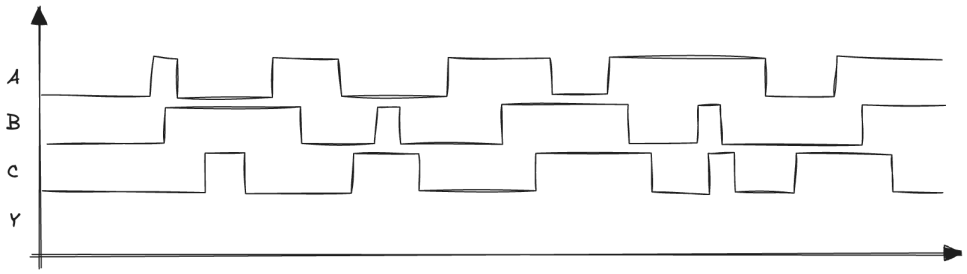
3.3 Exercise 3

Complete the truth table and the timing diagram of the following digital circuits. Then derive the Boolean equation in canonical **product-of-sums** form and simplify it to its minimal expression using Boolean algebra theorems. Finally, implement the corresponding circuit in DEEDS and verify the correctness of your solution.

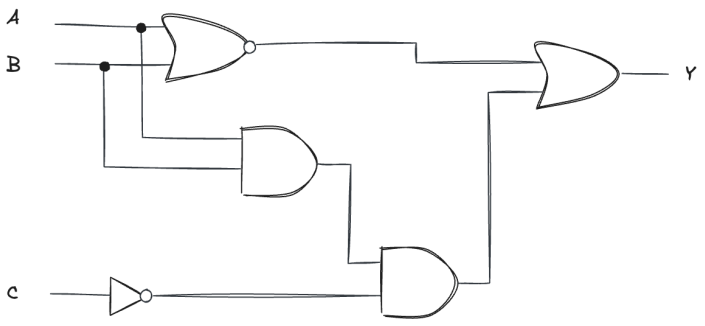
3.3.1 (a)



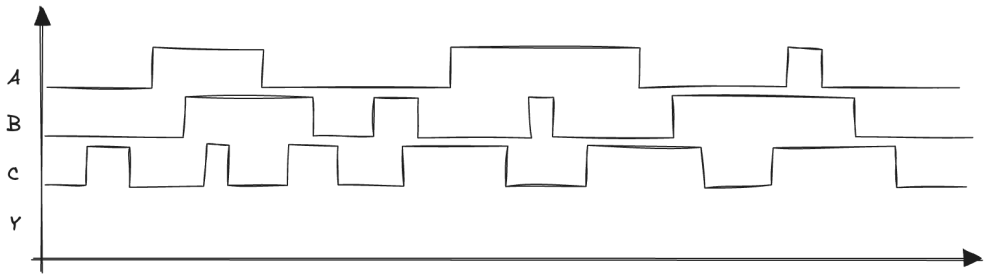
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



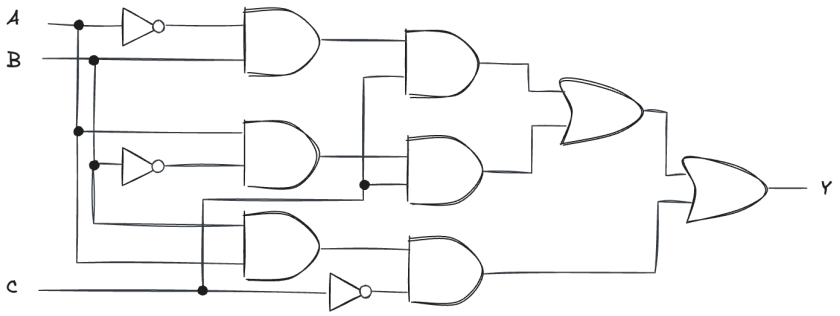
3.3.2 (b)



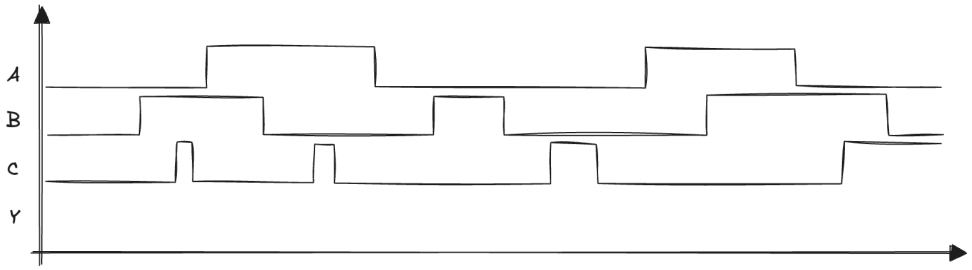
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



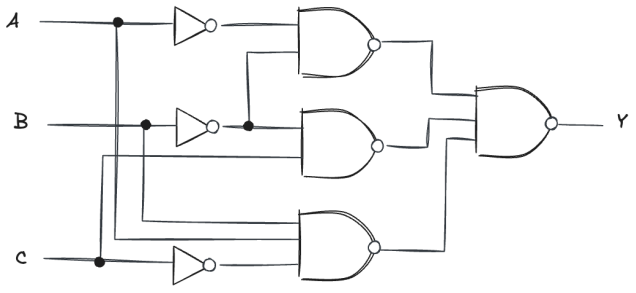
3.3.3 (c)



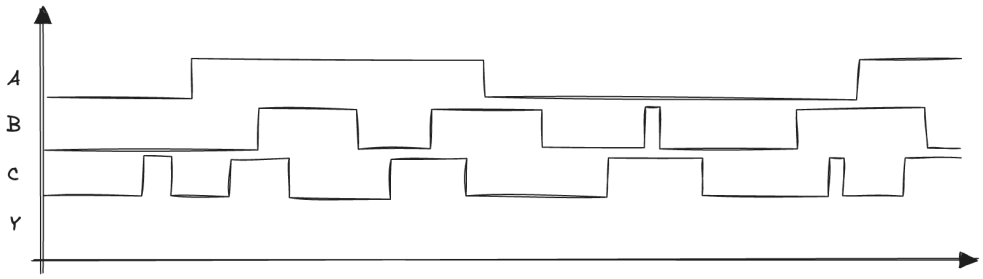
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



3.3.4 (d)



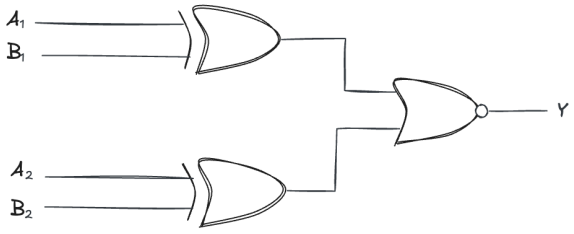
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



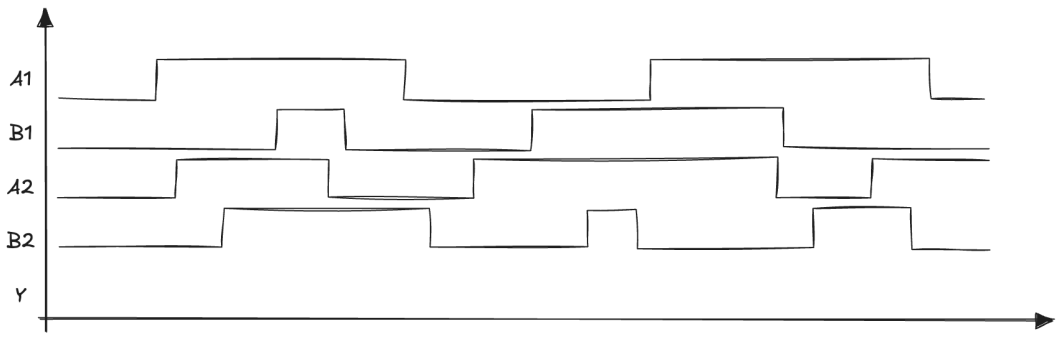
3.4 Exercise 4

Consider the following circuits. The first one is a **2-bit comparator**, which determines whether the value of A is equal to the value of B. The second one is a **1-bit adder**, which adds A, B, and the input carry, generating both the sum and the carry outputs. Complete the truth table and the timing diagram. Explain the operation of the two circuits in light of the truth table, and verify that they behave as expected. Then implement the corresponding circuits in DEEDS and verify the correctness of your solutions through simulation.

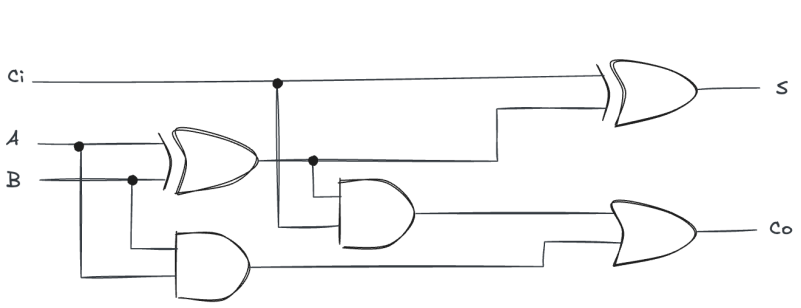
3.4.1 (a) 2-bit comparator



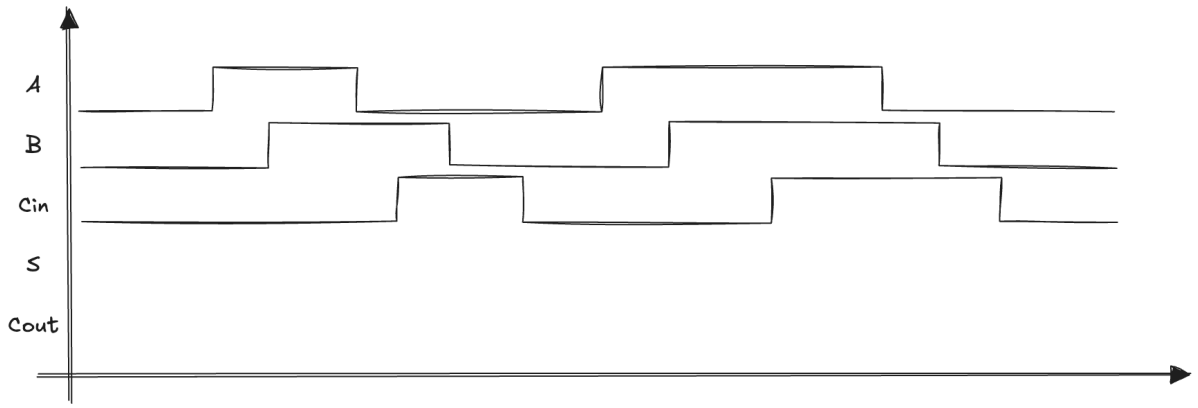
A1	B1	A2	B2	Y
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	



3.4.2 (b) 1-bit adder



A	B	Ci	S	Co
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		



4 Graphical minimization methods Exercises

4.1 Exercise 1

Design the following circuits by proceeding through these steps: define the truth table based on the textual description, construct the corresponding Karnaugh map, minimize the Boolean function, draw the circuit schematic, and finally verify the correctness of the truth table using Deeds.

4.1.1 (a) Majority Detector

Design a circuit with three inputs A, B, C and one output Y, which is 1 whenever at least two inputs are 1.

4.1.2 (b) Odd Parity Detector

Design a circuit with three inputs A, B, C and output Y = 1 if the number of 1s is odd

4.1.3 (c) Code Detector

Design a circuit with three inputs A, B, C. The output Y = 1 only when the input pattern is 101.

4.1.4 (d) Greater Than 9

Design a circuit that takes a 4-bit input A3 A2 A1 A0 and outputs Y=1 if the unsigned decimal number represented by the binary input is greater than 9 (decimal 10–15).

4.1.5 (e) 2-bit Comparator

Design a circuit with two 2-bit binary numbers A1 A0 and B1 B0. The output Y = 1 if the unsigned decimal number represented by A is greater than the unsigned decimal number represented by B.

4.1.6 (f) Binary-to-Gray Code Converter

Design a circuit that converts a 3-bit binary input (A2 A1 A0) to its Gray code (G2 G1 G0)

4.2 Exercise 2

Simplify the following Boolean expressions using two different approaches: using Boolean algebra theorems and Karnaugh maps.

4.2.1 (a)

$G = ABC + B\overline{C}$

4.2.2 (b)

$H = (A + \overline{B})(B + \overline{C})$

4.2.3 (c)

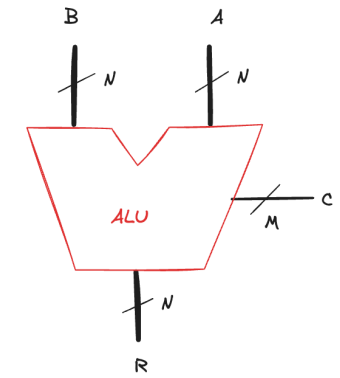
$Y = BC\overline{D} + CD + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}C$

4.2.4 (d)

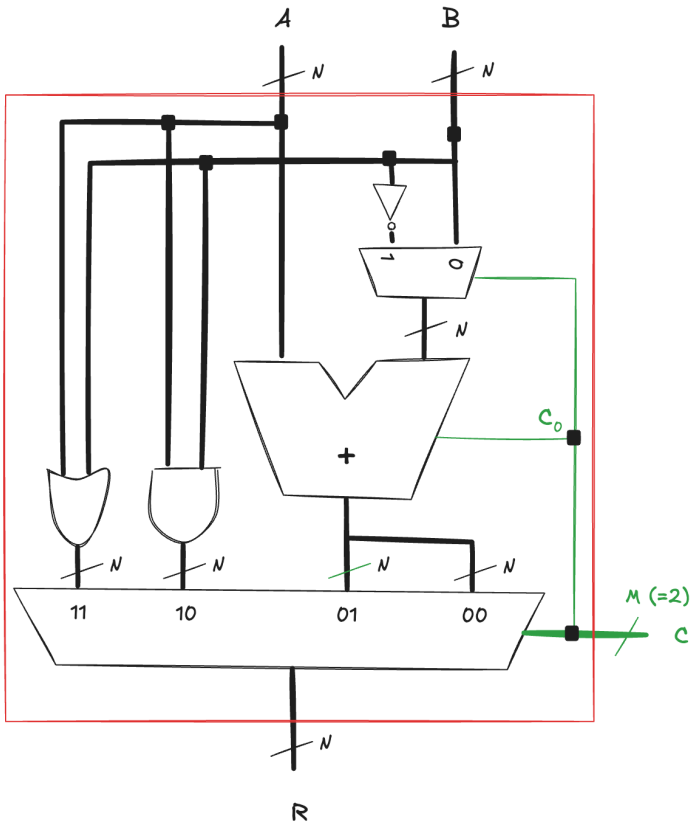
$Y = AB + A\overline{B}\,\overline{C}\,\overline{D}$

5 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:



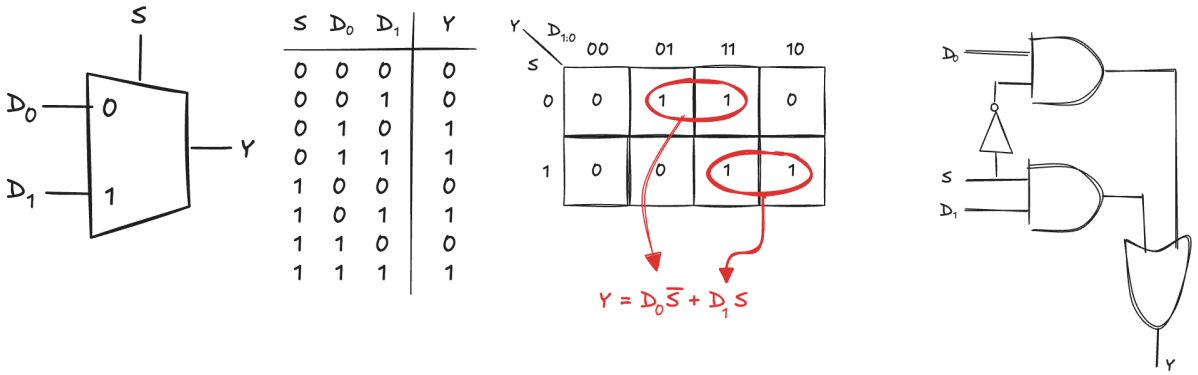
Control C_1	Control C_0	Funtion
0	0	Add
0	1	Subtract
1	0	AND
1	1	OR



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.

5.1 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.

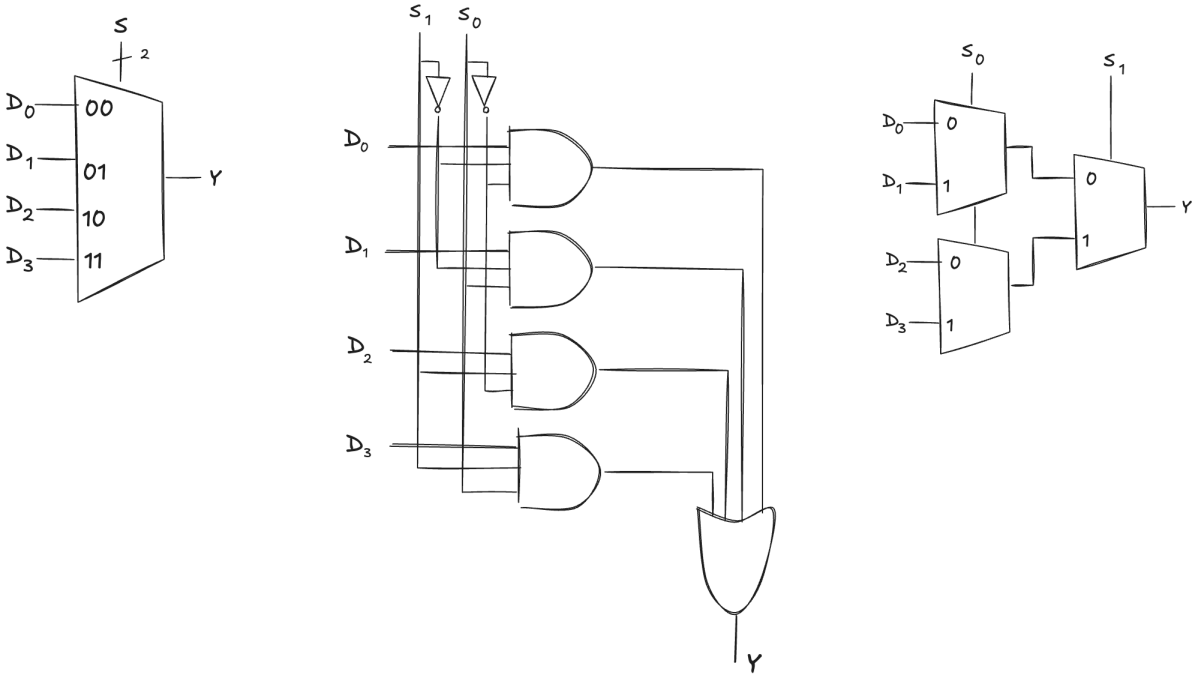
```
[VHDL]
```

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]
```

5.2 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.

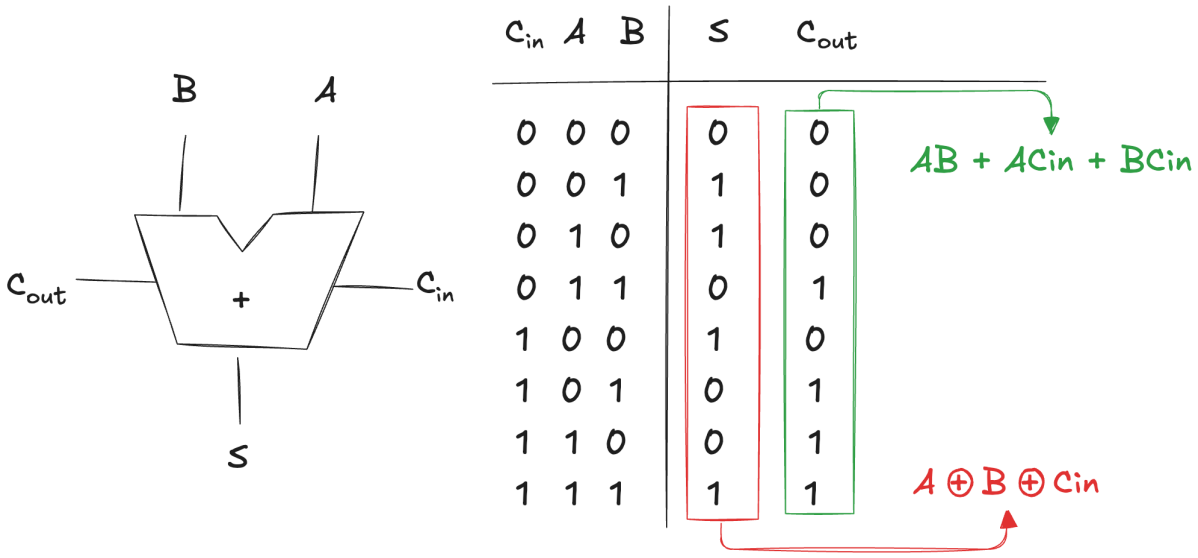
[VHDL]

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]

5.3 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]

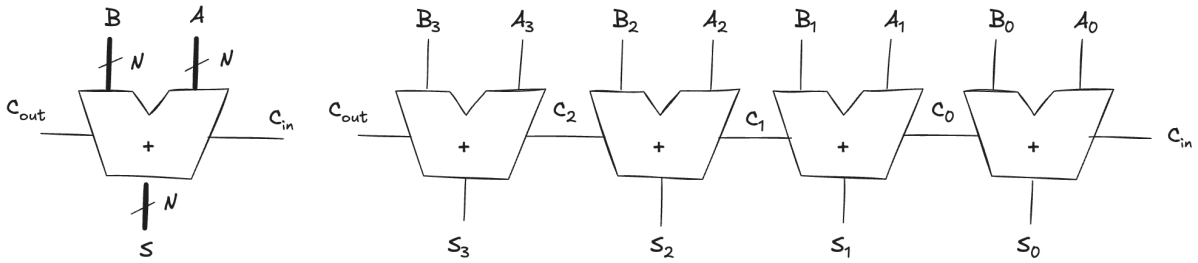
Write the VHDL description of the full adder and include it in the Vivado project.

[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]

5.4 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]

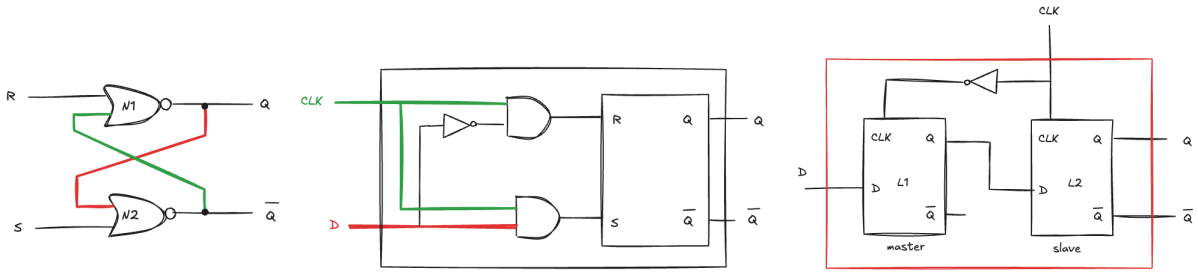
5.5 Next Steps

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.

6 Latch and Flip-Flop Exercises

6.1 Exercise 1

Implement a D flip-flop in DEEDS and simulate its behavior using a timing diagram that shows it is transparent only at the rising edge of the clock and that, at all other times, it retains its previously stored value. Then use this circuit to create a reusable block that can be incorporated into other designs:



6.2 Exercise 2

Consider the following behavioral descriptions of different types of flip-flops. For each type, design a possible implementation, build it in DEEDS, verify its operation using a timing diagram and write the VHDL code that describes its behavior.

6.2.1 (a) Toggle Flip-Flop

The toggle (T) flip-flop inverts its output on every active clock edge when T is high, producing a clean alternating sequence that effectively divides the clock frequency by two.

6.2.2 (b) JK Flip-Flop

A JK flip-flop receives a clock and two inputs, J and K. It updates its output on each clock edge by setting Q when J=1 and K=0, resetting it when J=0 and K=1, holding its value when both inputs are 0, and toggling when J and K are both 1, making it a versatile device that can act as a set/reset flip-flop, a memory cell, or a toggle depending on its inputs.

6.2.3 (c) Asynchronous Preset and Clear Flip-Flop

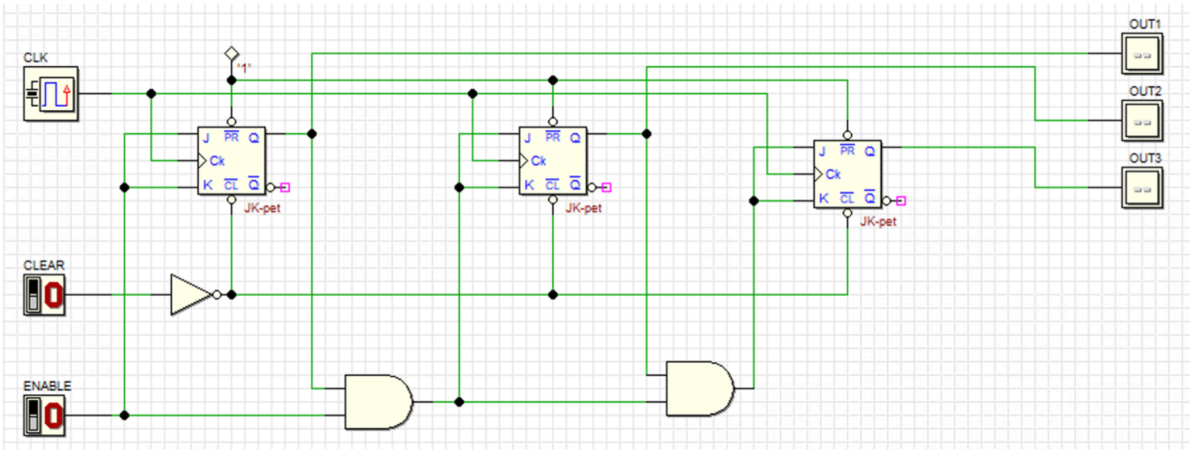
We already considered a reset input for flip-flops, which allows us to force the output Q to 0 regardless of the data inputs. However, we considered only a **synchronous reset**, which acts only on the active clock edge. In many applications, it is useful to have asynchronous inputs that can override the normal operation of the flip-flop at any time, without waiting for a clock event. Preset and Clear are **asynchronous inputs** that let us immediately force a flip-flop into a known state (Preset driving the output high and Clear driving it low), so the circuit can **start up correctly**, **recover from faults**, or be placed instantly in a controlled condition without waiting for a clock edge. Modify the D flip-flop block in order to add a Set inputs, and simulate its operation with a timing diagram that shows how this input override the normal D and clock

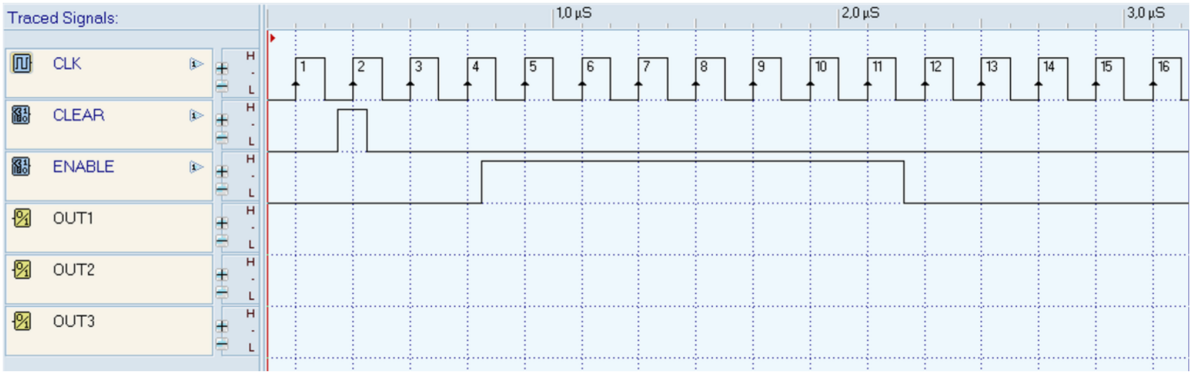
behavior. Then try the DEEDS library flip-flop with preset and clear inputs and compare its behavior with your design.

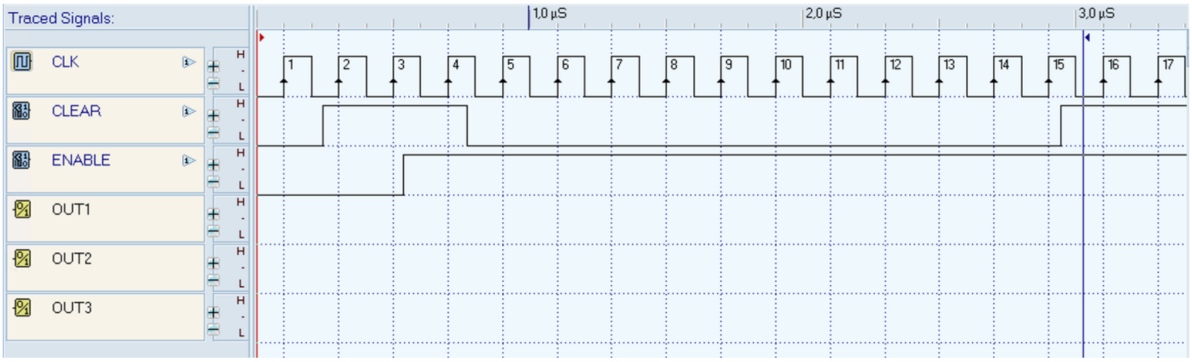
6.3 Exercise 3

Analyze the following synchronous sequential circuits by completing the timing diagrams. First draw the waveforms by hand and then use DEEDS only to verify your solutions.

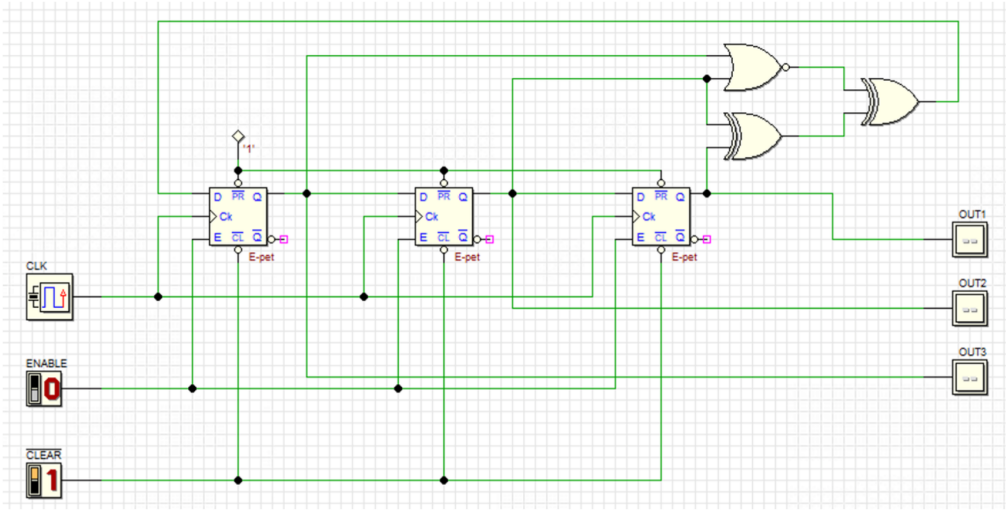
6.3.1 (a)

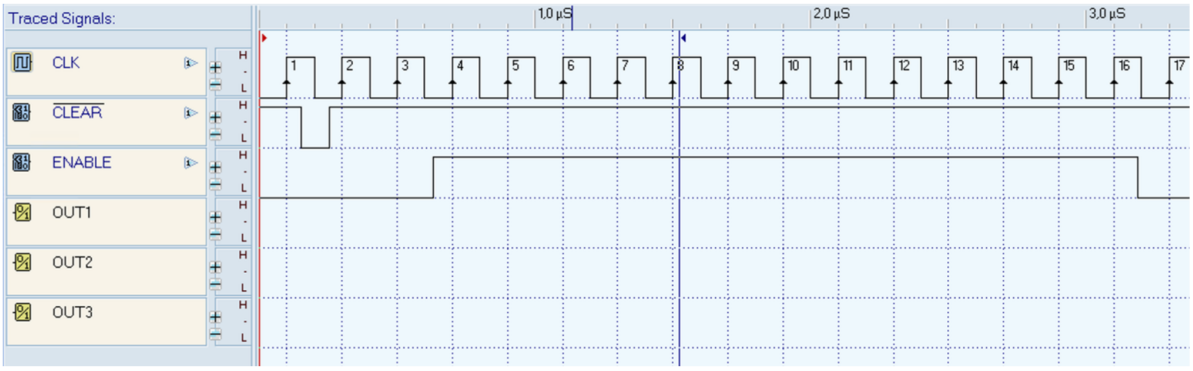


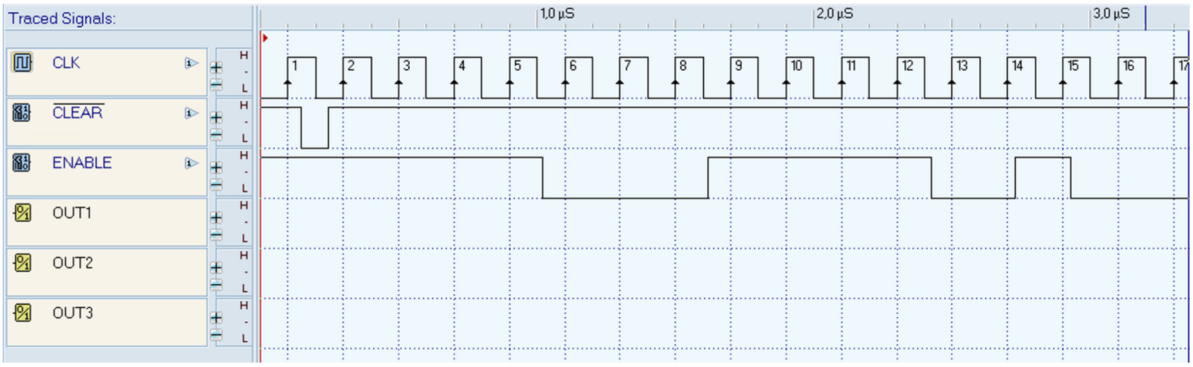



Traced Signals:
1,0 µS
2,0 µS
3,0 µS
CLK
CLEAR
ENABLE
OUT1
OUT2
OUT3

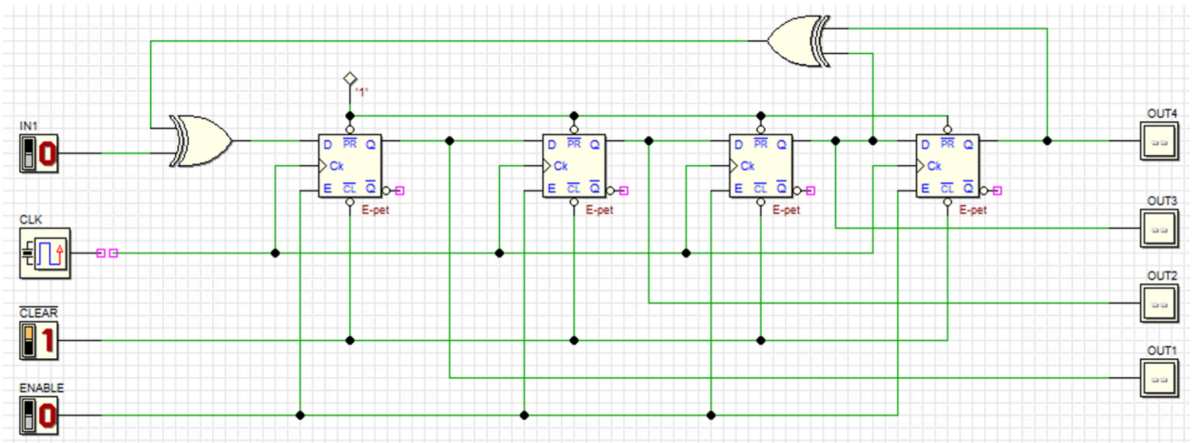
6.3.2 (b)

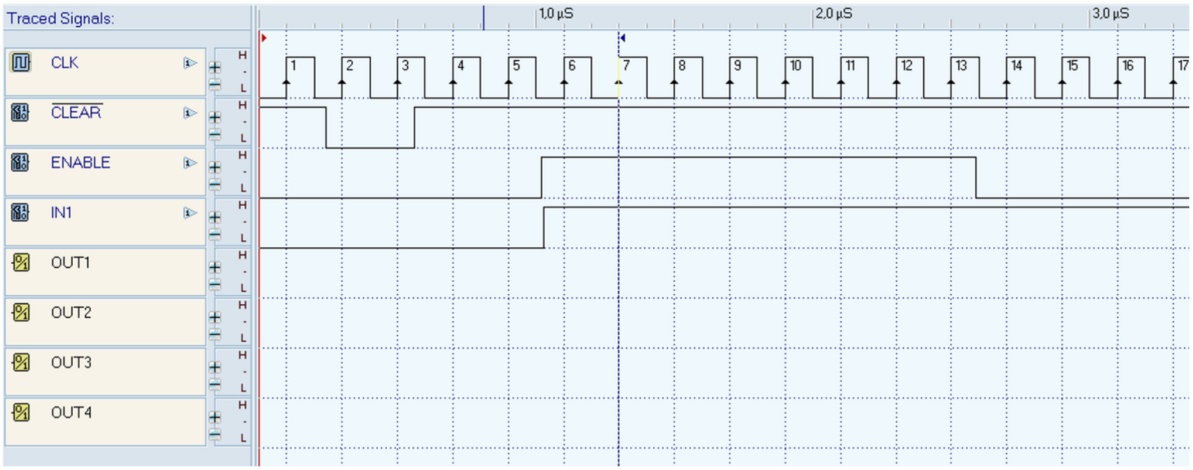

CLK
ENABLE
CLEAR
OUT1
OUT2
OUT3
E-pet
E-pet
E-pet


Traced Signals:
1,0 µS
2,0 µS
3,0 µS
CLK
CLEAR
ENABLE
OUT1
OUT2
OUT3


Traced Signals:
1,0 µS
2,0 µS
3,0 µS
CLK
CLEAR
ENABLE
OUT1
OUT2
OUT3

6.3.3 (c)


IN1
CLK
CLEAR
ENABLE
OUT4
OUT3
OUT2
OUT1
E-pet


Traced Signals:
1,0 µS
2,0 µS
3,0 µS
CLK
CLEAR
ENABLE
IN1
OUT1
OUT2
OUT3
OUT4