https://github.com/xpospi0g/Digital-electronics-1

c	**b**	**a**	**f(c,b,a)**
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

```
library IEEE;
use IEEE.std_logic_1164.all;
-------------------------------------------------------------------------
-- Entity declaration for basic gates
-------------------------------------------------------------------------
entity gates is port(
        c_i     : in  std_logic;
        b_i     : in  std_logic;
        a_i     : in  std_logic;
        f_o     : out std_logic;
        fnand_o : out std_logic;
        fnor_o  : out std_logic
    );
end entity gates;
-------------------------------------------------------------------------
-- Architecture body for basic gates
-------------------------------------------------------------------------
-- Usage of De Morgan laws on function f using nands and nors
architecture dataflow of gates is begin
    f_o  <= ((not b_i) and a_i) or ((not c_i) and (not b_i));
    fnand_o <= not(not((not b_i) and a_i) and not((not c_i) and (not b_i)));
    fnor_o <= not(b_i or (not a_i)) or not(c_i or b_i);
end architecture dataflow;
```

https://www.edaplayground.com/x/NYbN

Distributive laws

z	**y**	**x**	**a(x,y,z)**	**b(x,y,z)**
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1

```
library ieee;
use ieee.std_logic_1164.all;


entity distributiveLaws is
    port(
        x_i    : in  std_logic;
        y_i    : in  std_logic;
        z_i    : in  std_logic;
        aL_o   : out std_logic;
        aP_o   : out std_logic;
        bL_o   : out std_logic;
        bP_o   : out std_logic
    );
end entity distributiveLaws;

architecture dataflow of distributiveLaws is
begin
    aL_o  <= (x_i and y_i) or (x_i and z_i);
    aP_o  <= (x_i and (y_i or z_i));
    bL_o  <= ((x_i or y_i) and (x_i or z_i));
    bP_o  <= (x_i or (y_i and z_i));

end architecture dataflow;
```