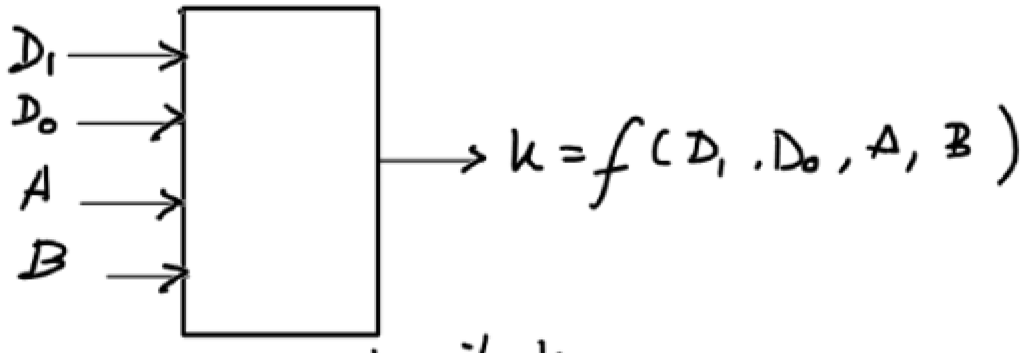
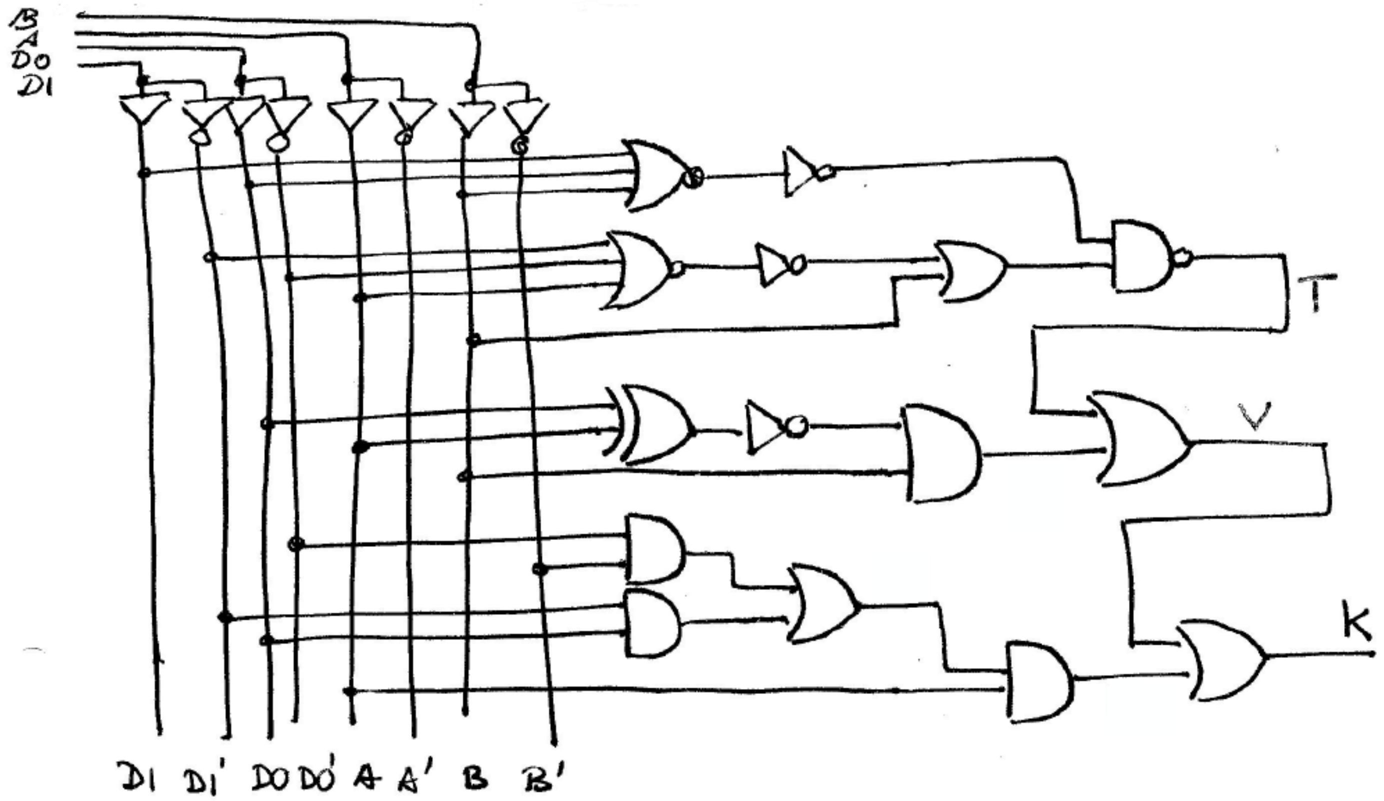
## 1.- Specifications

Find the truth table of the Circuit_K using the method based on EDA tools (method III) → VHDL



circuit_k

Circuit internal architecture



## 2.- Planning

This is the sequence of operations to solve the problem:

The same as in method I and method IV

1. - Obtain the circuit's equation and check it with other students
2. - Write the equation in this source file:

Translate

**L:\CSD\P1\Circuit_K\VHDL\Circuit_K.vhd**

location

development steps

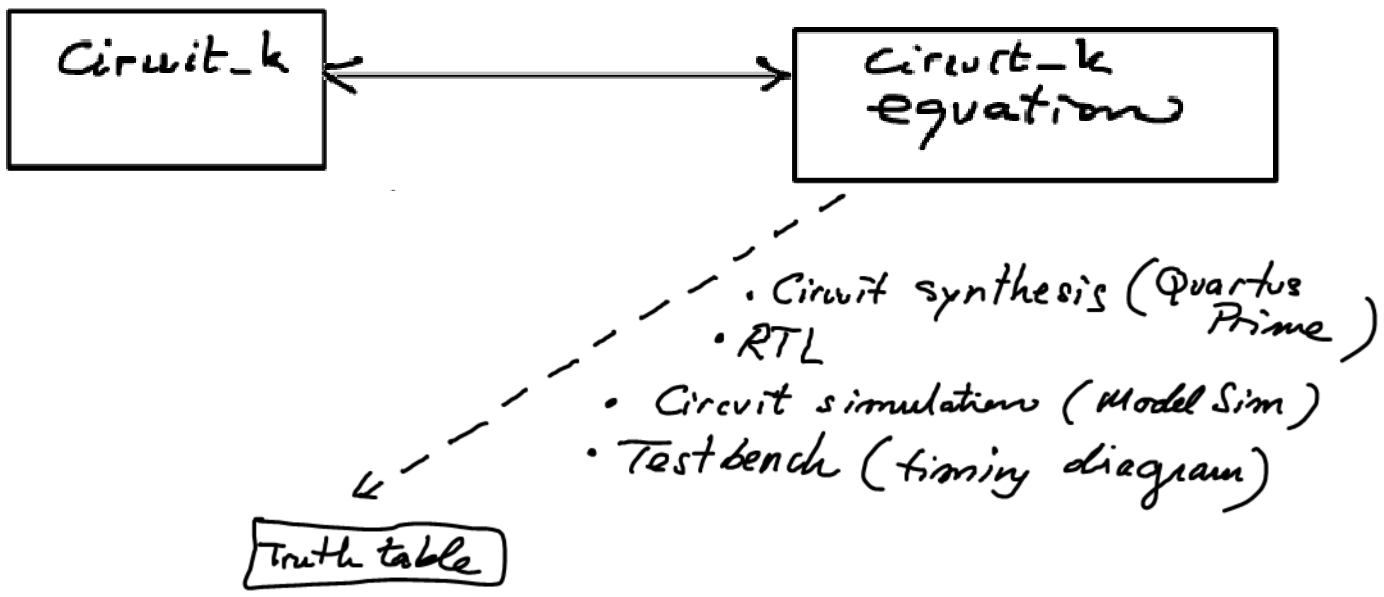
target

3. - Start and run a synthesis project using a EDA tool for a given chip
4. - Verify the RTL schematic
5. - Generate a VHDL test bench template and add some input vector activity:

**L:\CSD\P1\Circuit_K\VHDL\Circuit_K_tb.vhd**

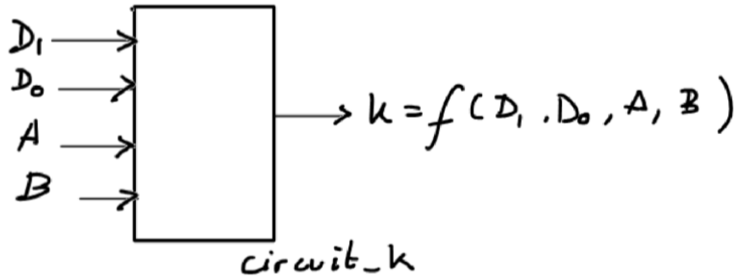
6. - Run the VHDL simulator and visualise the timing analyser
7. - Extract the truth table simulating all the inputs
8. - Test the truth table with other methods (Proteus, WolframAlpha, etc.)

Another analysis project


Cirwit_k
circuit_k equation
· Circuit synthesis (Quartus Prime)
· RTL
· Circuit simulation (Model Sim)
· Testbench (timing diagram)
Truth table

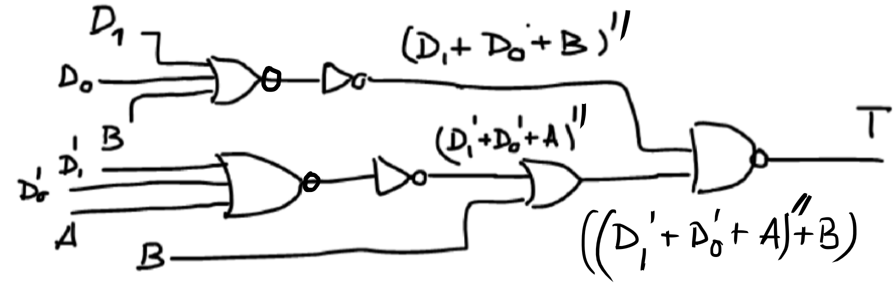
## 3.- Development

Obtain the circuit's equations

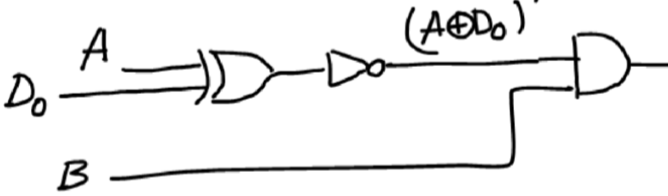


Expected solution

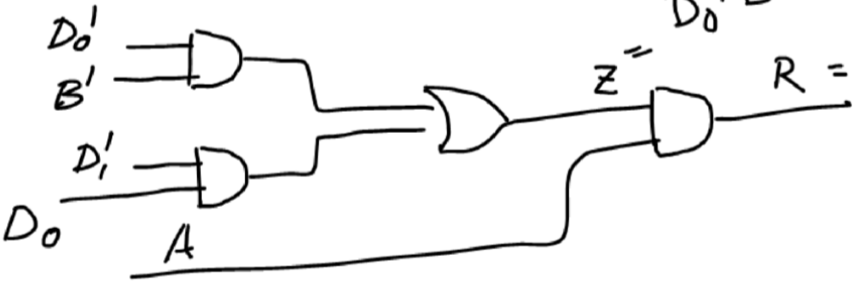
| $D_1$ | $D_0$ | $A$ | $B$ | K |
|---|---|---|---|---|
| ∅ | ∅ | ∅ | ∅ | · |
| ∅ | ∅ | ∅ | 1 | · |
| ⋮ | | | | |
| 1 | 1 | 1 | 1 | · |



$$T = \left[(D_1+D_0+B)'' \cdot \left[(D_1'+D_0'+A)''+B\right]\right]'$$

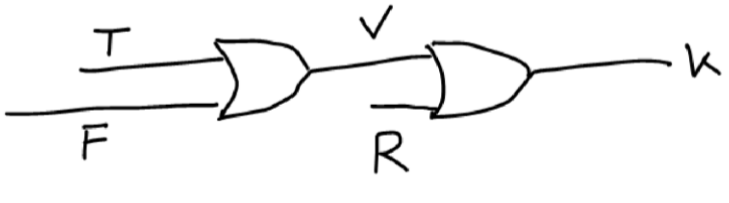


$$F = B \cdot (A \oplus D_0)'$$



$$Z = D_0' \cdot B' + D_1' \cdot D_0$$

$$R = A\left(D_0' B' + D_1' \cdot D_0\right)$$



Finally: $K = R + V = R + (T + F) =$

$$\left[(D_1+D_0+B)'' \cdot \left[(D_1'+D_0'+A)'' + B\right]\right]' + \left[B \cdot (A \oplus D_0)'\right] + A \cdot \left[D_0' B' + D_1' D_0\right]$$

↗ Circuit_K equation

Print the VHDL file with the equation translation

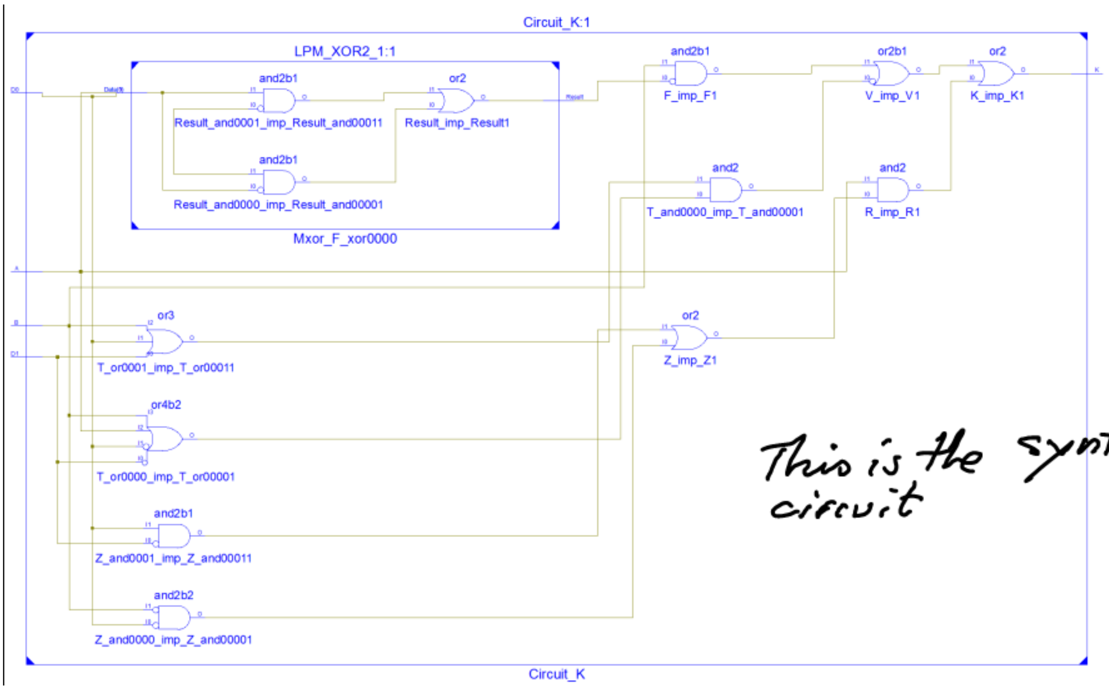
This is the VHDL source file containing the circuit's equation in the architecture section:

```
--------------------------------------------------------------------------------
-- An example of the P1 - Section A - method #4: Using VHDL to analyse a circuit
-- 1.- Specifications: Find the truth table of the Circuit_K
-- 2.- Plan:
--             - Obtain the circuit's equation and check it with other students
--             - Write the equation in this source file
--             - Start and run a synthesis project using a EDA tool for a given chip
--             - Verify the RTL schematic
--             - Generate a VHDL testbench template and add some input vector activity
--             - Run the VHDL simulator and visualise the timing analyser
--             - Extract the truth table simulating all the inputs
--             - Test the truth table with other methods (Proteus, WolframAlpha, etc.)
-- 3.- Develop de problem as stated in the plan
-- 4.- Test the truth table

-- This is also an exercise to introduce the VHDL language and the EDA tools
---------------------------------------------------
-- Exercises P1 - P2 - CSD : Analysis of a logic circuit
-- http://digsys.upc.edu
--------------------------------------------------------------------------------
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
ENTITY Circuit_K IS
       PORT (
             D1, D0 :      IN     STD_LOGIC;
             A, B          :      IN     STD_LOGIC;
             K             :      OUT    STD_LOGIC
         );
END Circuit_K;
ARCHITECTURE circuit_equation OF Circuit_K IS
SIGNAL T, F, V, R, Z : STD_LOGIC;
       BEGIN

-- This is the translation of the circuit's equation to VHDL. It is very similar
-- to the WolframAlpha expression:
-- truth table  not[[not(not(not(D1) or not(D0) or A))  or B ] and
-- not(not (D1 or D0 or B ) )  ]  or [not(A xor D0) and B]
-- or [ (not(D0) and not(B) or not(D1) and D0 ) and A ]
       K      <= R or V;
```

```
        V       <= T or F;
        R       <= Z and A;
        Z       <= (not(D0) and not(B) ) or ( not(D1) and D0 );
        T       <= not( (not(not(not(D1) or not(D0) or A))  or B ) and not(not (D1 or D0 or B )));
        F       <=     not(A xor D0) and B ;
END circuit_equation;
```

This is a picture of the RTL view of the circuit:

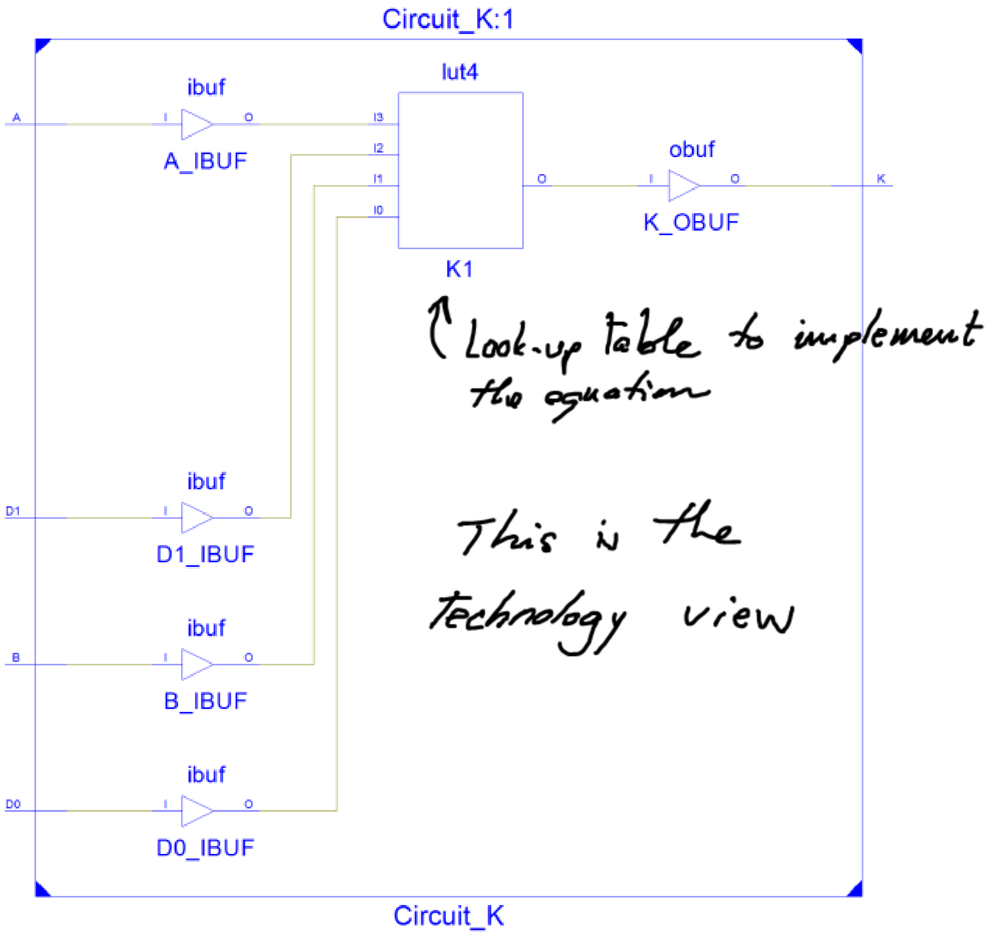
It can be seen how the XOR gate is solved using AND and OR gates.



This is the synthesised circuit

And this is the schematic of the circuit as it is implemented in a Xilinx chip Spartan-3E XC3S500E-FG320 ← Target chip

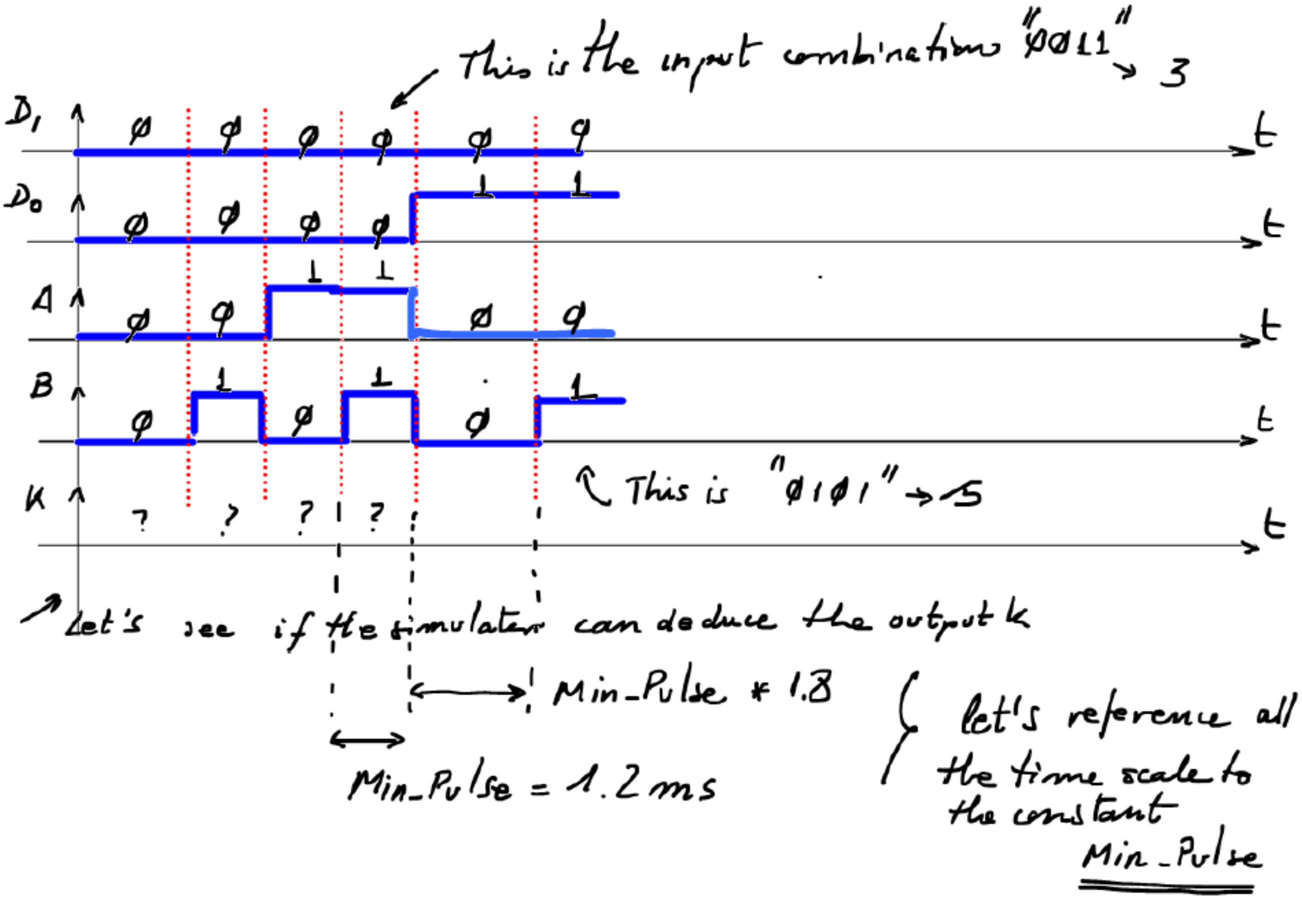
The FPGA realises the Circuit_K using a single Look-up table:



Look-up table to implement the equation

This is the technology view

Let's generate a testbench file to drive the circuit K output from some input activity. Here the complete truth table:



This is the input combination "ØØ11" → 3

This is "Ø1Ø1" → 5

Let's see if the simulator can deduce the output K

Min_Pulse * 1.8

Min_Pulse = 1.2 ms

Let's reference all the time scale to the constant Min_Pulse

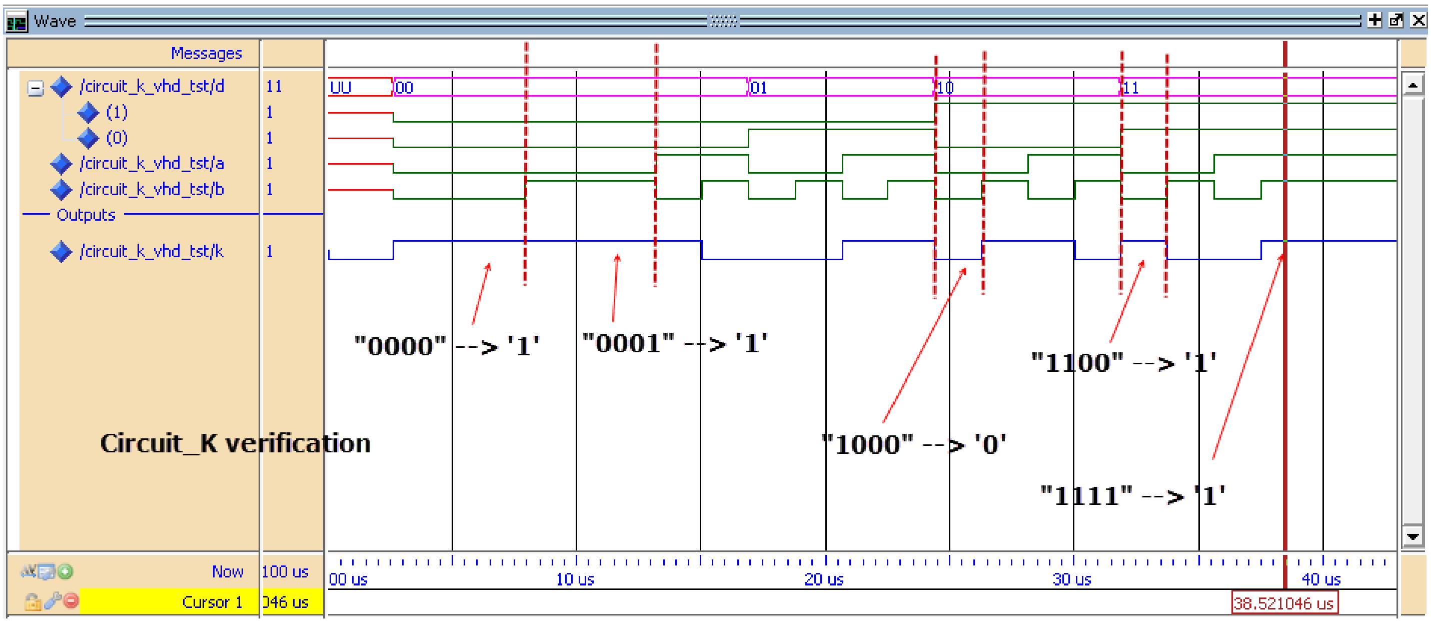
The translation of the Testbench into VHDL

This is the main section of the **Circuit_K_tb.vhd** file

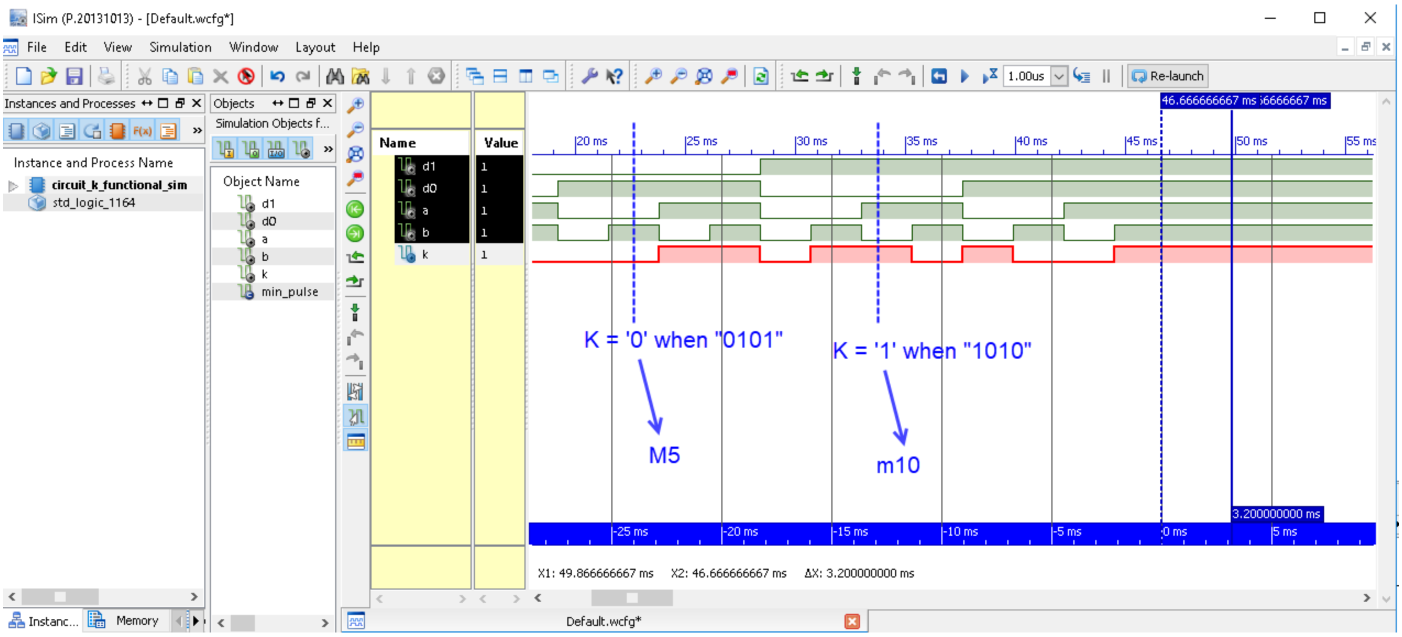
```
  -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;

      wait for Min_Pulse *10;

      -- insert stimulus here
            D1 <= '0';
            D0 <= '0';
            A <= '0';
            B <= '0';
      wait for Min_Pulse * 2.3;

            D1 <= '0';
            D0 <= '0';
            A <= '0';
            B <= '1';
      wait for Min_Pulse * 2.3;

            D1 <= '0';
            D0 <= '0';
            A <= '1';
            B <= '0';
      wait for Min_Pulse * 2.3;

            D1 <= '0';
            D0 <= '0';
            A <= '1';
            B <= '1';
      wait for Min_Pulse * 2.3;

            D1 <= '0';
            D0 <= '1';
            A <= '0';
            B <= '0';
      wait for Min_Pulse * 2.3;

            D1 <= '0';
            D0 <= '1';
            A <= '0';
            B <= '1';
      wait for Min_Pulse * 2.3;

            D1 <= '0';
            D0 <= '1';
            A <= '1';
            B <= '0';
      wait for Min_Pulse * 2.3;

            D1 <= '0';
            D0 <= '1';
            A <= '1';
            B <= '1';
      wait for Min_Pulse * 2.3;

            D1 <= '1';
            D0 <= '0';
            A <= '0';
            B <= '0';
      wait for Min_Pulse * 2.3;

            D1 <= '1';
            D0 <= '0';
            A <= '0';
            B <= '1';
      wait for Min_Pulse * 2.3;
```

```
        D1 <= '1';
        D0 <= '0';
        A <= '1';
        B <= '0';
    wait for Min_Pulse * 2.3;

        D1 <= '1';
        D0 <= '0';
        A <= '1';
        B <= '1';
    wait for Min_Pulse * 2.3;

        D1 <= '1';
        D0 <= '1';
        A <= '0';
        B <= '0';
    wait for Min_Pulse * 2.3;

        D1 <= '1';
        D0 <= '1';
        A <= '0';
        B <= '1';
    wait for Min_Pulse * 2.3;

        D1 <= '1';
        D0 <= '1';
        A <= '1';
        B <= '0';
    wait for Min_Pulse * 2.3;

        D1 <= '1';
        D0 <= '1';
        A <= '1';
        B <= '1';
    wait for Min_Pulse * 2.3;

    wait;
  end process;
```

This is an example of timing diagram from the VHDL functional simulator



This is another timing diagram:



This is the truth table deduced inspecting the timing diagram:

**K = f(D1, D0, A, B) =** $\sum_4 \boldsymbol{m}(\mathbf{0, 1, 2, 6, 7, 9, 10, 12, 15})$

Or, instead,

**K = f(D1, D0, A, B) =** $\prod_4 \boldsymbol{M}(\mathbf{3, 4, 5, 8, 11, 13, 14})$

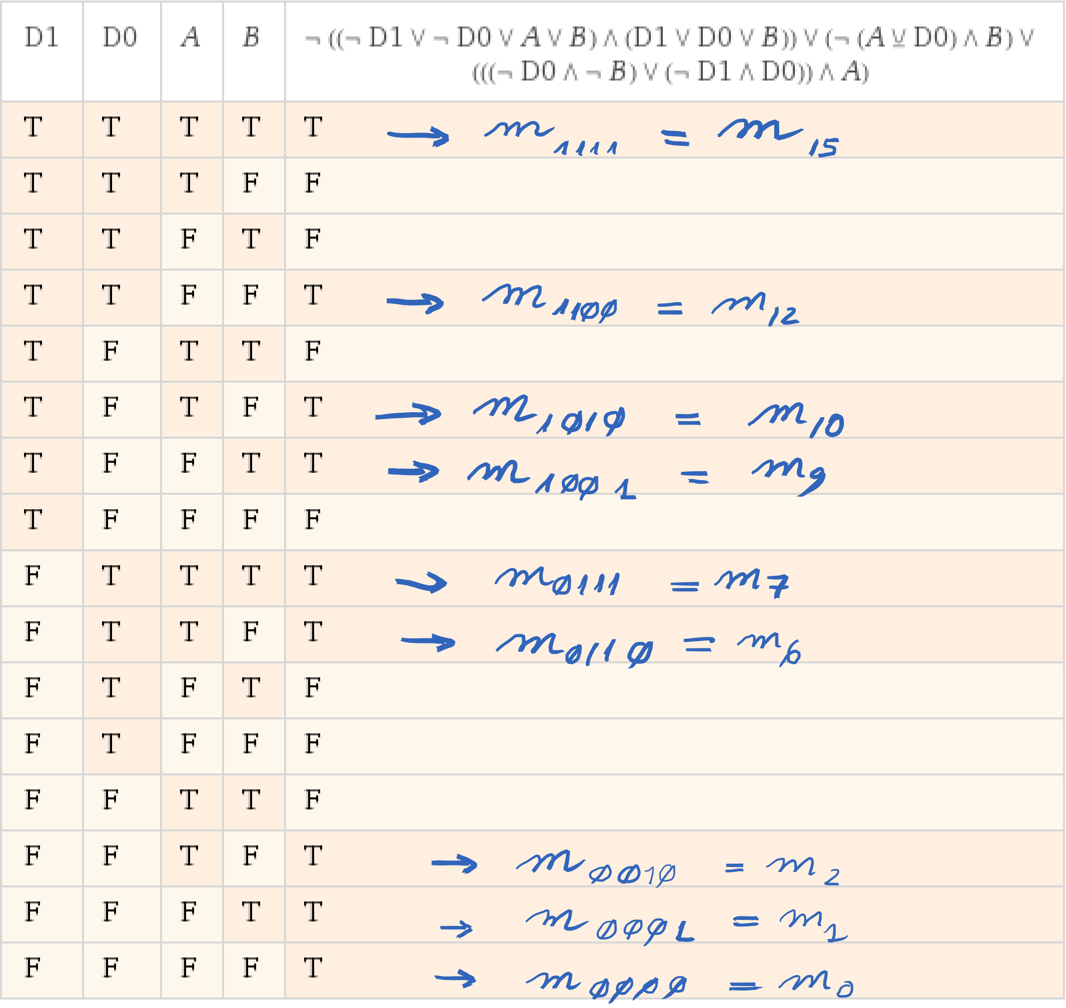
## 4. Test

We have solved the circuit using method I and also method IV and we obtain the same truth table with method III

## 4. TEST and VERIFICATION

This is the circuit equation in WolframAlpha, so that we can obtain the truth table using the Method #2 and compare:

**truth table not[[not(not(not(D1) or not(D0) or A)) or B ] and not(not (D1 or D0 or B ) ) ] or [not(A xor D0) and B] or [ (not(D0) and not(B) or not(D1) and D0 ) and A ]**



| D1 | D0 | $A$ | $B$ | $\neg((\neg D1 \lor \neg D0 \lor A \lor B) \land (D1 \lor D0 \lor B)) \lor (\neg(A \veebar D0) \land B) \lor (((\neg D0 \land \neg B) \lor (\neg D1 \land D0)) \land A)$ | |
|---|---|---|---|---|---|
| T | T | T | T | T | → $m_{1111} = m_{15}$ |
| T | T | T | F | F | |
| T | T | F | T | F | |
| T | T | F | F | T | → $m_{1100} = m_{12}$ |
| T | F | T | T | F | |
| T | F | T | F | T | → $m_{1010} = m_{10}$ |
| T | F | F | T | T | → $m_{1001} = m_{9}$ |
| T | F | F | F | F | |
| F | T | T | T | T | → $m_{0111} = m_{7}$ |
| F | T | T | F | T | → $m_{0110} = m_{6}$ |
| F | T | F | T | F | |
| F | T | F | F | F | |
| F | F | T | T | F | |
| F | F | T | F | T | → $m_{0010} = m_{2}$ |
| F | F | F | T | T | → $m_{0001} = m_{1}$ |
| F | F | F | F | T | → $m_{0000} = m_{0}$ |

↳ $K = f(D_1, D_0, A, B) = \sum_{4} m(0,1,2,6,7,9,10,12,15)$

And so, we've verified that the truth table deduced using VHDL tools is the correct one!

→ check it using Proteus (Method 1) or analytically (Method 3)