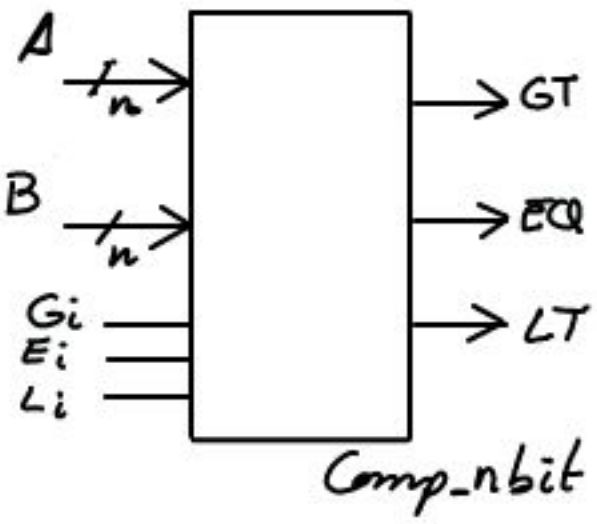
1. Specifications

Let's design a Comp_1bit using a flat design based on SoP

Symbol



Truth table

A	B	Gi	Ei	Li	GT	EQ	LT
A > B		x	x	x	1	0	0
A < B		x	x	x	0	0	1
A = B		1	0	0	1	0	0
		0	1	0	0	1	0
		0	0	1	0	0	1

In this way it is possible to chain blocks Comp1_nbit to build larger comparators

Thus, for a 1 bit comparator we have:

a	b	Gi	Ei	Li	GT	EQ	LT
1	0	x	x	x	1	0	0
0	1	x	x	x	0	0	1
0	0	1	0	0	1	0	0
0	0	0	1	0	0	1	0
0	0	0	0	1	0	0	1
1	1	1	0	0	1	0	0
1	1	0	1	0	0	1	0
1	1	0	0	1	0	0	1

If the bit a=b → the comparison will depend on the previous bits as represented by Gi, Ei, Li

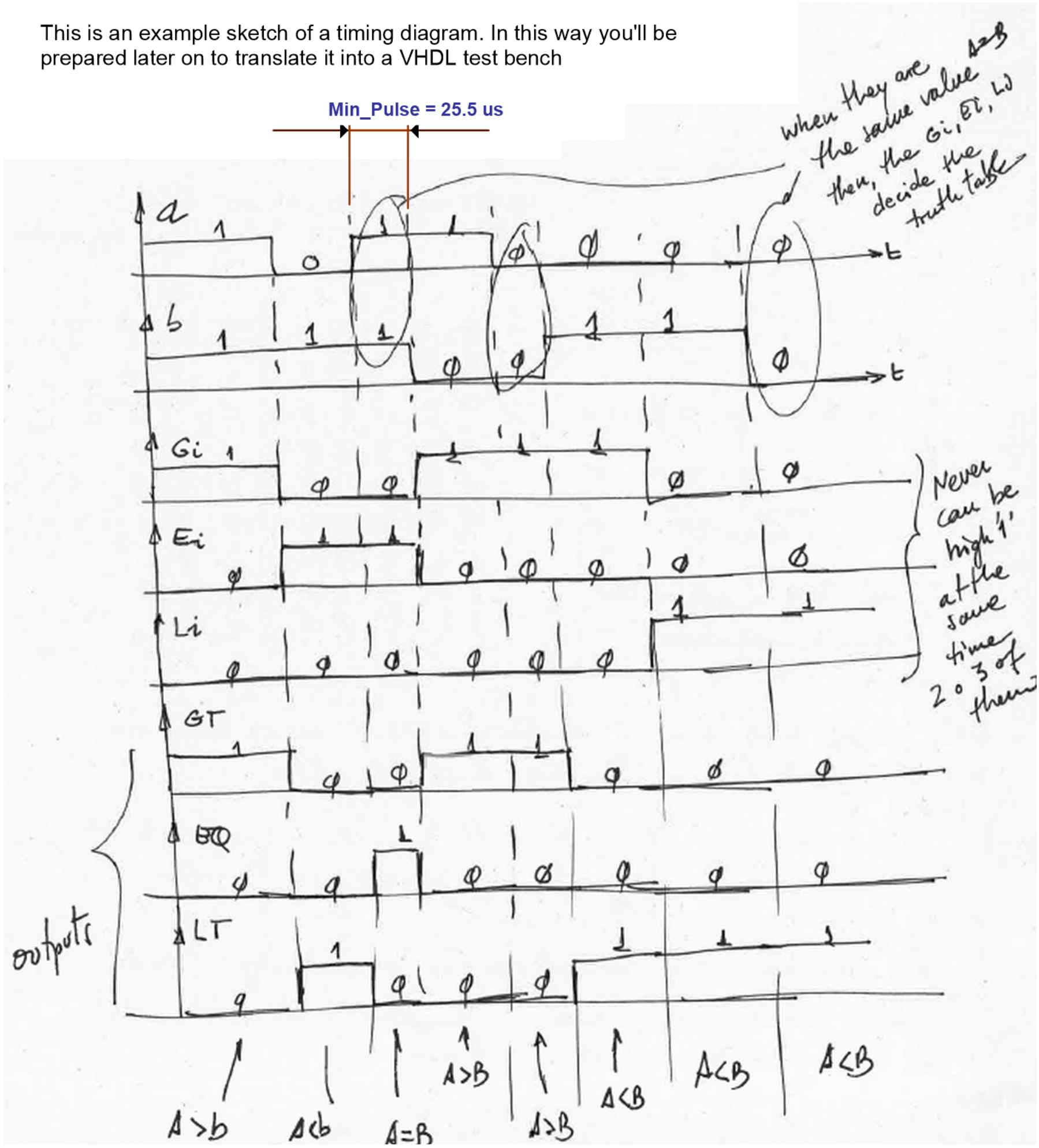
→ The thuth table has 2^5 combinations, but many terms are not important because they are impossible like: (don't care outputs)

0 1 1 1 0
1 0 0 1 1
0 0 0 0 0
⋮

here it is impossible to have two '1', etc.

This property will simplify the output logic, and 'Minilog' will detect such incomplete truth table asking for what to do → don't care terms → '-'

This is an example sketch of a timing diagram. In this way you'll be prepared later on to translate it into a VHDL test bench



So that, running the simulator for about 300 us will be enough to test the more meaningful values of the truth table.

2. Planning

The idea will be to write the truth table in "minilog" text format , so that we can minimise immediately and obtain the SoP equation.

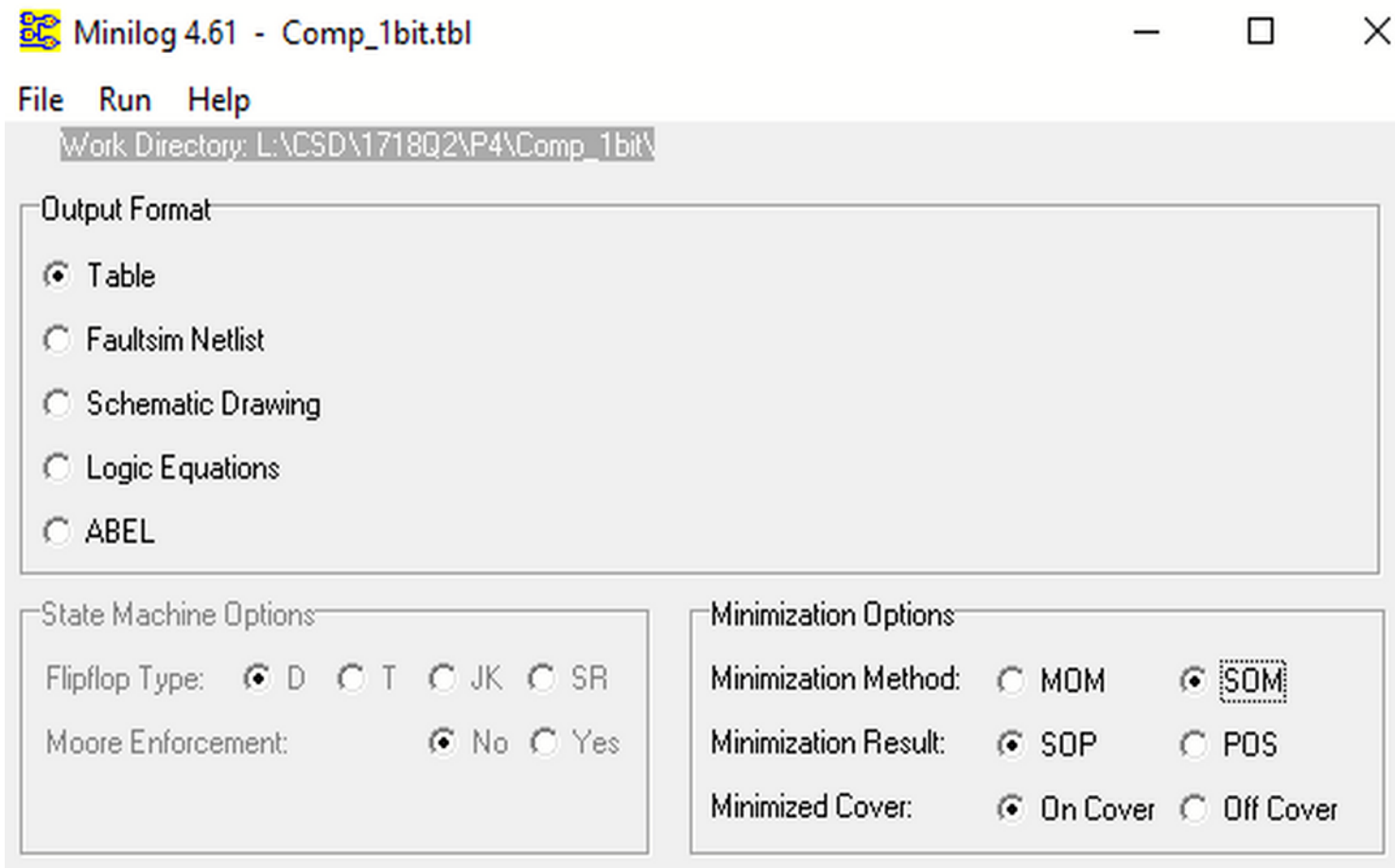
```
table Comp_1bit

  input A B Gi Ei Li

  output GT EQ LT

" A B  Gi Ei Li    GT EQ LT
" ------------    --------
  1 0  -  -  -     1  0  0  "This is A > B
  0 1  -  -  -     0  0  1  "This is A < B

  0 0  1  0  0     1  0  0  " Because A = B, the chaining inputs decides
  0 0  0  1  0     0  1  0  " Because A = B, the chaining inputs decides
  0 0  0  0  1     0  0  1  " Because A = B, the chaining inputs decides

  1 1  1  0  0     1  0  0  " Because A = B, the chaining inputs decides
  1 1  0  1  0     0  1  0  " Because A = B, the chaining inputs decides
  1 1  0  0  1     0  0  1  " Because A = B, the chaining inputs decides

" There are impossible situation like:
" - -  1  1  1
" - -  1  1  0  which will never occur, and so, they may be a don't care output
"-------------------------------------------------------------------------
end
```

* Find this text file in the P4 project on arithmetic circuits

Example procedure to implement this project

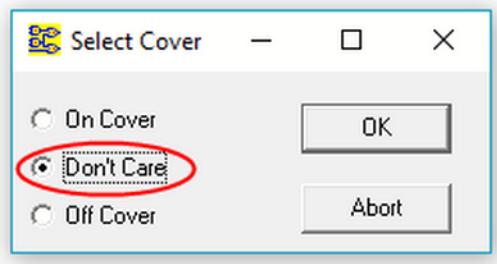
1. Study carefully the way the chip comparator works
2. Simplify in order to obtain the SoP
3. Draw the circuit and check with team members
4. Write the VHDL equivalent and start a Quartus II project
5. Synthesize and study the RTL and the technology schematics
6. Translate the timing diagram or some vector from the truth table to a testbench
7. Run the VHDL simulation to verify how the circuit works. ✓
8. Use this Comp_1bit in other circuits like a Comp_4bits

3. Development




```
FUNCTION TABLE

===========
  GEL   GEL
ABIII   TQT
===========
10--- | 100
01--- | 001
00100 | 100
00010 | 010
00001 | 001
11100 | 100
11010 | 010
11001 | 001

CHECKING COMPLETENESS OF INPUT TABLE
INCOMPLETELY SPECIFIED INPUT TABLE; MISSING COVER:
===========
  GEL   GEL
ABIII   TQT
===========
00000 | ???
11000 | ???
00-11 | ???
11-11 | ???
001-1 | ???
111-1 | ???
0011- | ???
1111- | ???

ADD ABOVE TERMS TO ON-/DC-/OFF-COVER OR ABORT PROGRAM
SELECTED COVER: ( 1 = ON; - = DC; 0 = OFF; Q = Abort ) >
```

The list of impossible terms which we don't care




```
MINIMIZATION RESULT STATISTICS
==============================
FOUND 8 ESSENTIAL PRODUCT TERMS
MAXIMUM FANIN:                  6
TOTAL LITERAL COUNT:           26
MAXIMUM PRODUCT TERM SIZE:     3
MAXIMUM OUTPUT FUNCTION SIZE: 3

===========
  GEL   GEL
ABIII   TQT
===========
-01-- | 1..
1-1-- | 1..
10--- | 1..
00-1- | .1.
11-1- | .1.
0---1 | ..1
-1--1 | ..1
01--- | ..1
```

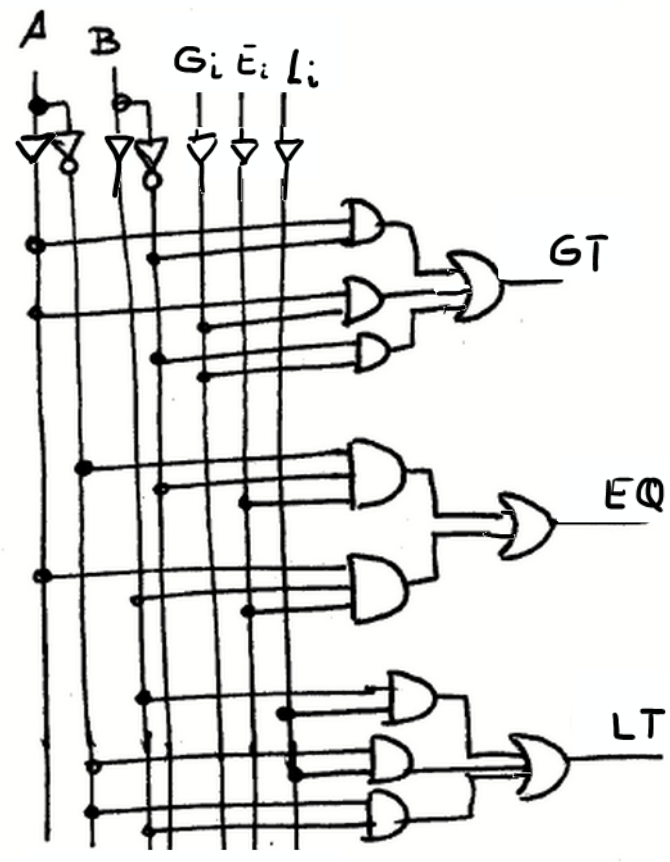
The interpretation of the output format:

Greater than $GT = A \cdot B' + A \cdot G_i + B' \cdot G_i$

Equal: $EQ = A' \cdot B' \cdot E_i + A \cdot B \cdot E_i$

Less than: $LT = B \cdot L_i + A' \cdot L_i + A' \cdot B$

And the equivalent circuit based on SoP



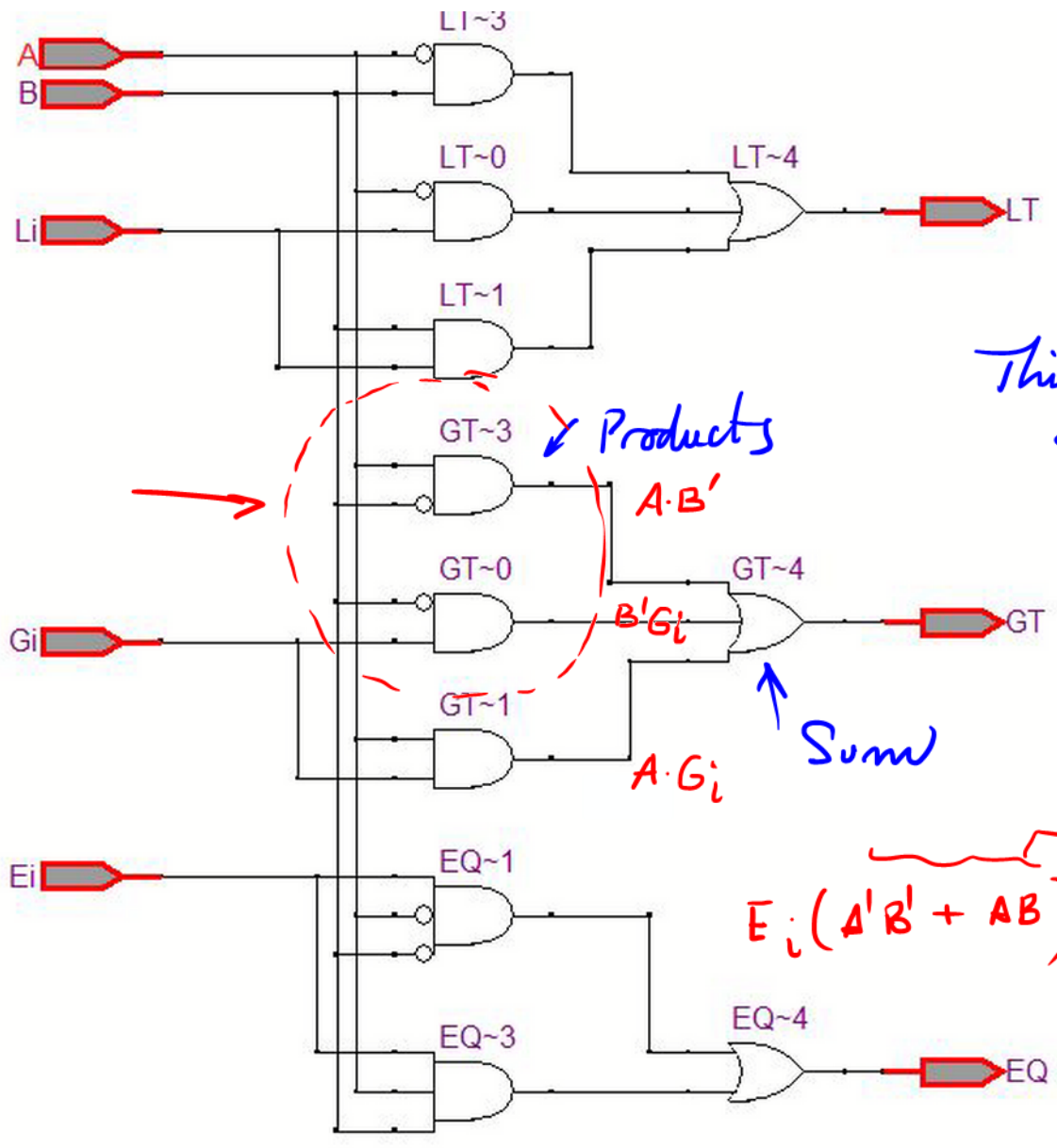
Development. Synthesis of the circuit using an EDA tool like Lattice Semiconductor ispLEVER Classic / Intel-Altera Quartus II or Xilinx ISE

This is the VHDL code. the structural equations in SoP from above:

```
-------------------------------------------------------------------------------
-- An example of a 1-bit comparator using described using simplified logic
-- equations (structural) Sum of Products (SoP)
-------------------------------------------------------
-- Project P4  - CSD :  Arithmetic circuits
-- http://digsys.upc.es
-------------------------------------------------------------------------------
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;

ENTITY Comp_1bit IS
  PORT (
   A,B, Gi, Ei, Li : IN STD_LOGIC;
   GT, EQ, LT  : OUT STD_LOGIC
    );
END Comp_1bit;

ARCHITECTURE logic_equations_SoP OF Comp_1bit IS
BEGIN

GT <= (not(B) and Gi) or (A and Gi) or (A and not(B));
LT <= (not(A) and Li) or (B and Li) or (B and not(A));
EQ <= (not(A) and not(B) and Ei) or (A and B and Ei);


END logic_equations_SoP;
```

} The block definition as an entity

} Equations in VHDL in SoP



Products

$A \cdot B'$

$B'G_i$

$A \cdot G_i$

Sum

$GT = AB' + B' \cdot G_i + AG_i$

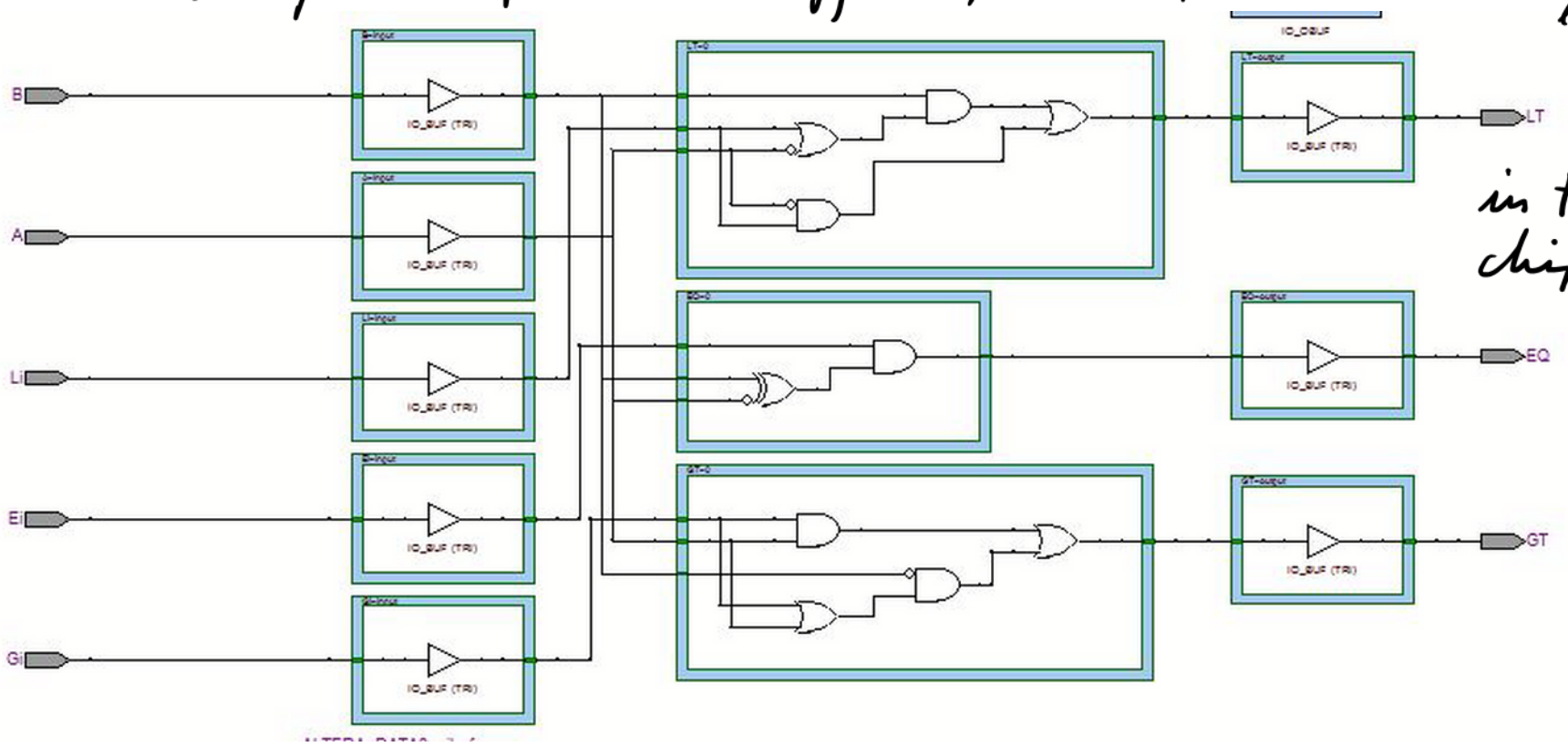
This is the RTL synthesised by Quartus II

$E_i(A'B' + AB)$ → Identity function

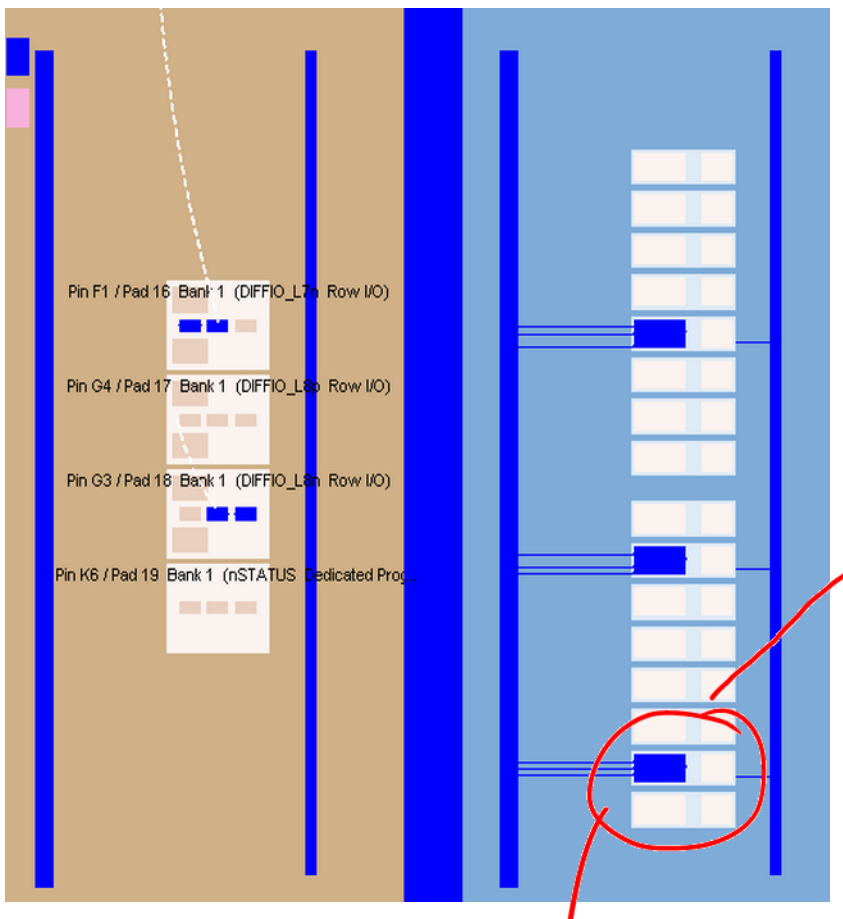
A	B	y
0	0	1
0	1	0
1	0	0
1	1	1

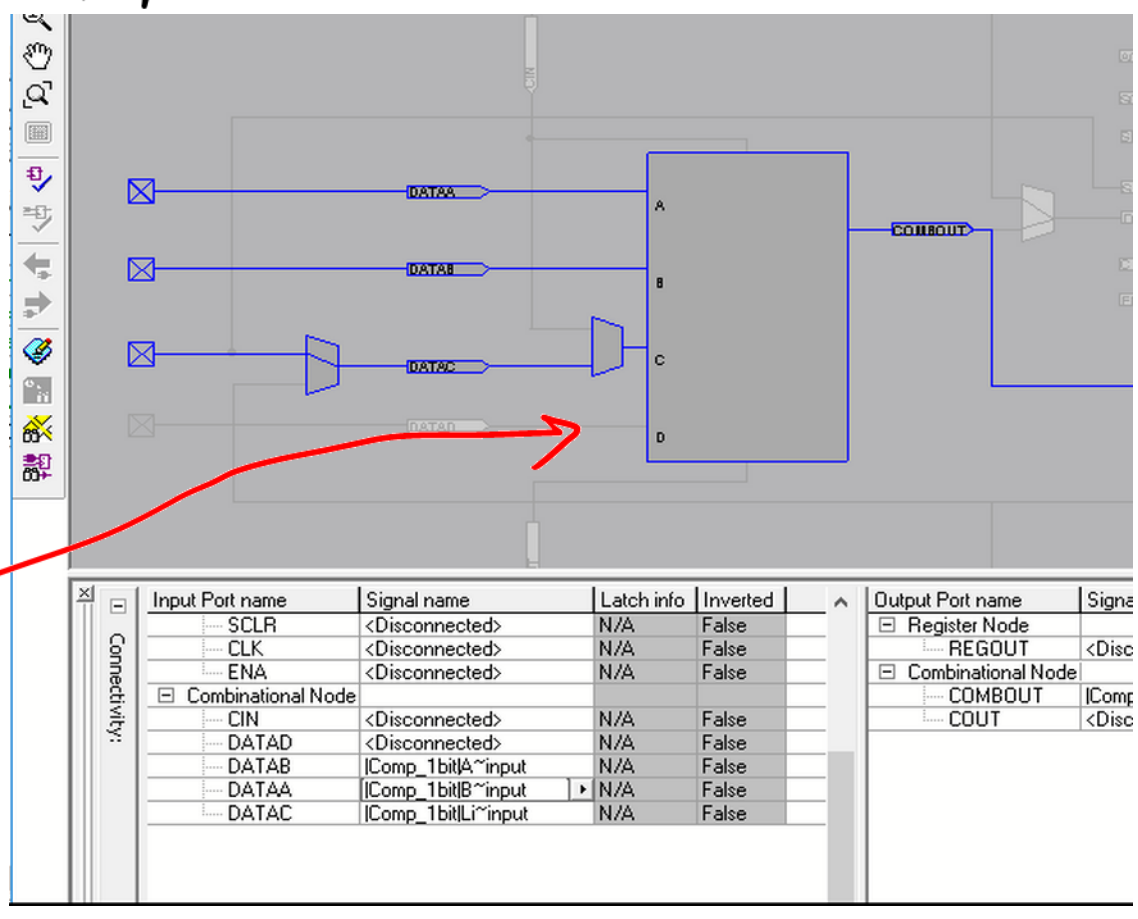
$(A \oplus B)'$

This is a picture of the technology view, where you can see the exact logic implemented in the target chip Cyclone III



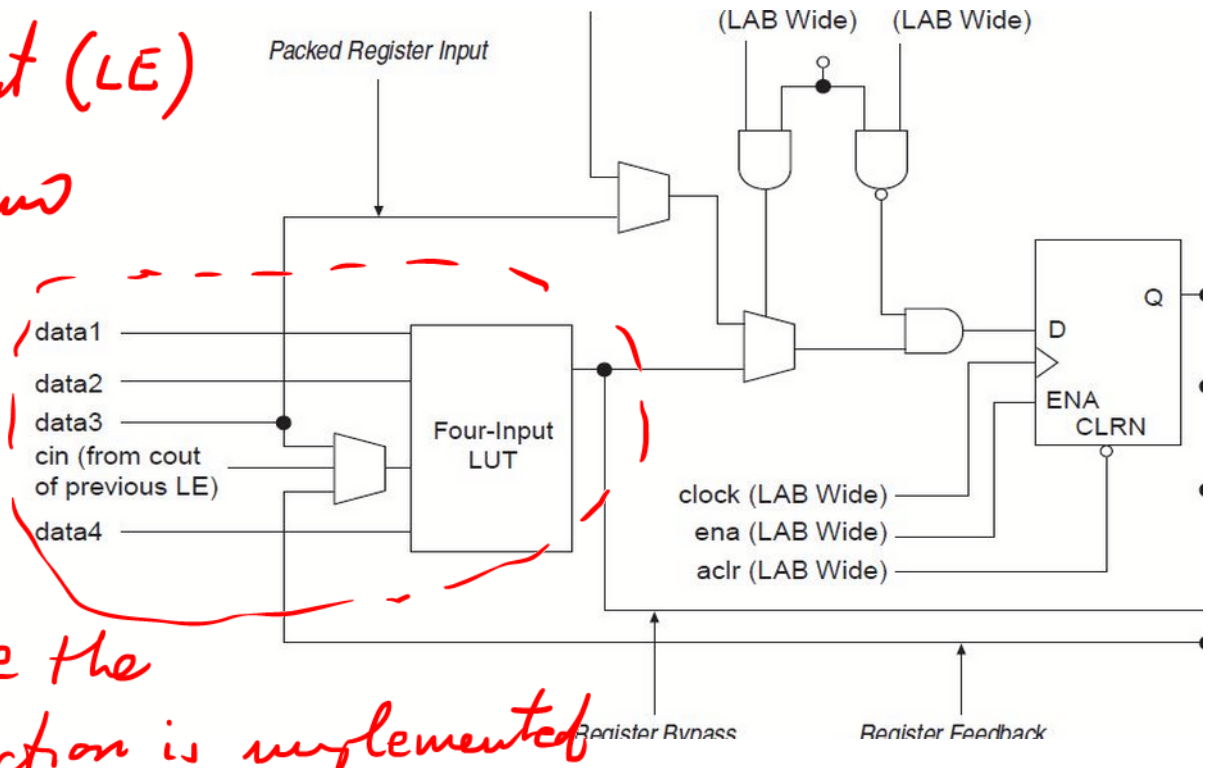
These are some pictures from the chip planner tool





logic element (LE)

There are thousands of them in the Cyclone III FPGA



This is the combinational LUT of a LE where the logic function is implemented

4. Testing using a VHDL test bench fixture, so that we can apply input stimulus and check the outputs, thus, verifying the circuit's truth table and validating the design of the component.

```
-- *** Test Bench - User Defined Section ***
   tb : PROCESS
   BEGIN
 -- Circuit initialisation period (do nothing).
  wait for 100 ns;
       wait for Min_Pulse*1.5;

 -- Let's start the stimulus activity:
  A  <=  '0';      -- This is EQ = 1
  B  <=  '0';

  Gi  <= '0';
  Ei  <= '1';
  Li  <= '0';

       wait for Min_Pulse*1.76;

  A  <=  '1';      -- This is EQ = 1
  B  <=  '1';

  Gi  <= '0';
  Ei  <= '1';
  Li  <= '0';
```

stimulus of a given vector

25.5 µs

```
-- constants:
 constant Min_Pulse : time := 25.5 us;
-- In this way, for testing several vectors,
-- running the simulator 1000 us will be enough
```

run 1000 us in ModelSim

```
     wait for Min_Pulse*1.76;

A  <=  '1';      -- This is GT = 1
B  <=  '0';

Gi  <= '0';
Ei  <= '1';
Li  <= '0';
```

This situation is wrong, so, the output 4 not important