Master’s degree in Computer Engineering

SIMPLE CPU

Giovanni Barbieri

Alessio Di Ricco

Academic Year 2022/2023

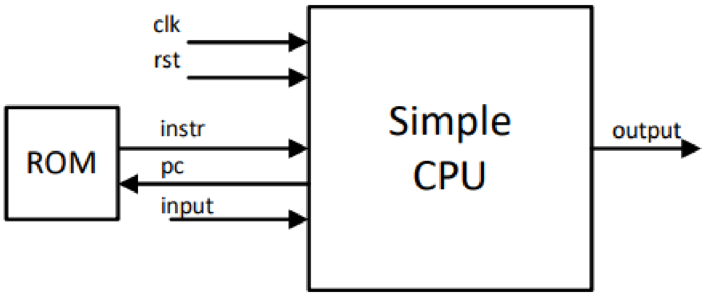
INDEX

INTRODUCTION

A CPU is a hardware device dedicated to the execution of instructions starting from an instruction set.

In this project we consider a simplified version of the CPU in which there are 4 8-bit registers (R0-R3) and 3 8-bit status registers: Status Register (SR), Instruction Register (IR), Program Counter (PC).

Instructions are taken from a 16x8-bit ROM.



The instruction set of the simplified CPU is:

RCSR Rx, SR	Write in Rx the content of the status register (SR)
IN Rx	Read the input into the register Rx
OUT Rx	Rx Write the content of register Rx to output.
MOV Ry, Rx	Move the content of register Rx into register Ry.
ADD Ry, Rx	Put into Ry the sum of Rx and Ry
MUL Ry, Rx	Put into Ry the multiplication of Rx and Ry
LSL Rx	Put into Rx, Rx shifted to the left of 1 position.
LSR Rx	Put into Rx, Rx shifted to the right of 1 position.

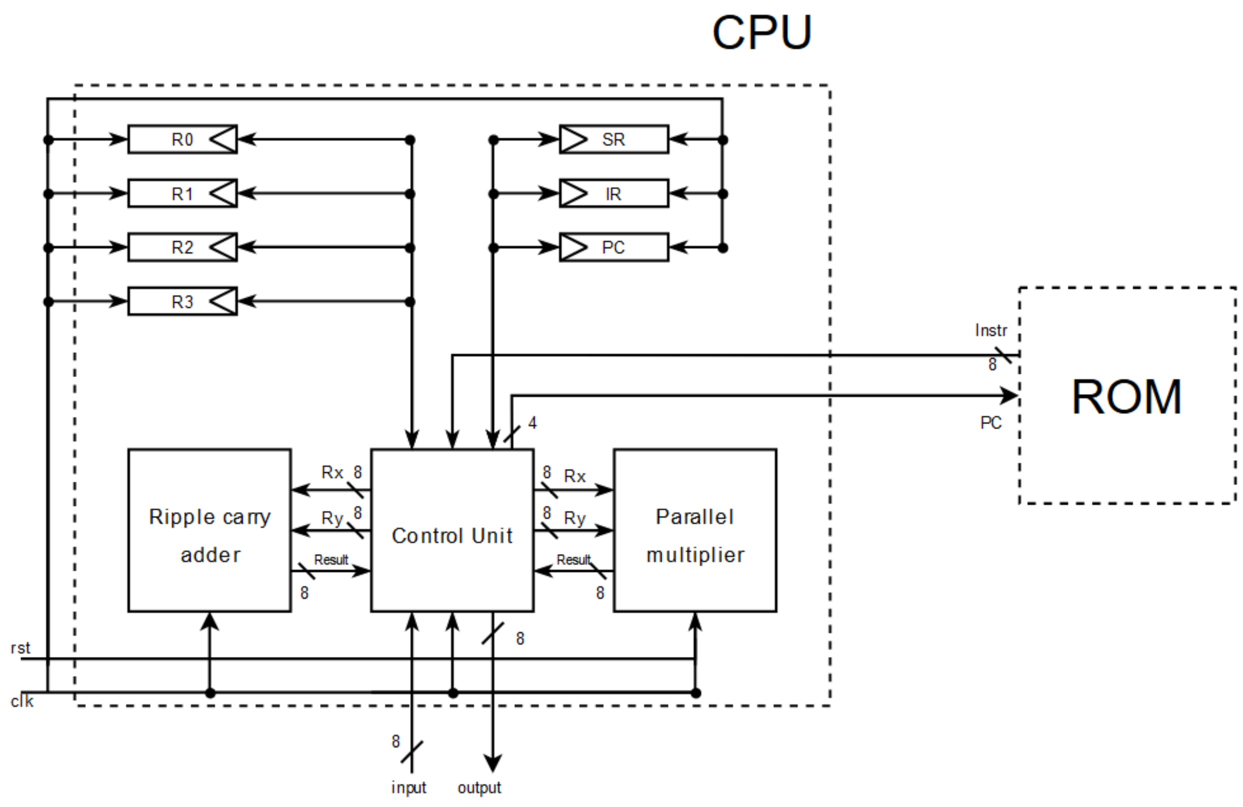
We have chosen the following instructions format:

Last 3 bits identify the instruction type, first 2 bits identify the register Rx while the second two the register Ry (if present)

RCSR:	xx000000
IN:	xx000001
OUT:	xx000010
MOV:	xxyy0011
ADD:	xxyy0100
MUL:	xxyy0101
LSL:	xx000110
LSR:	xx000111

ARCHITECTURE

The block diagram for the simple CPU is the following:



Basic Components of the Simple CPU

We are now moving to analyze the basic components used to make the CPU:

- Adder
- Multiplier
- Register

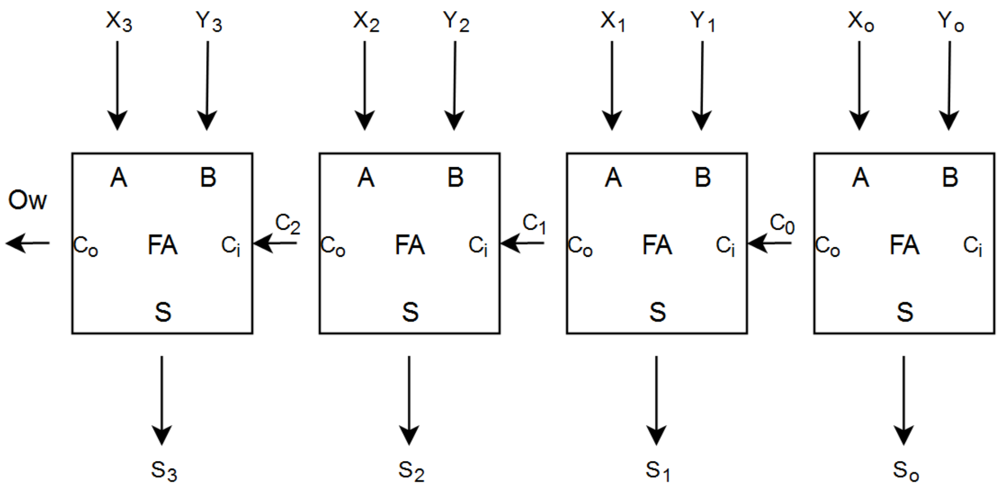
In our system we have also implemented a ROM which is external respect to the CPU module

Register implementation has been made using d-flip-flop.

ADDER:

To perform the add operation we add a ripple carry adder to our CPU.

It is composed of 8 1-bit Full Adder. This number of FA is due to the register size.
The overflow is obtained by looking at the last Full Adder Cout and it is used to set the Status Register.



RCA Block Diagram

MULTIPLIER

To perform the multiplier operation, we add a parallel multiplier to our CPU.

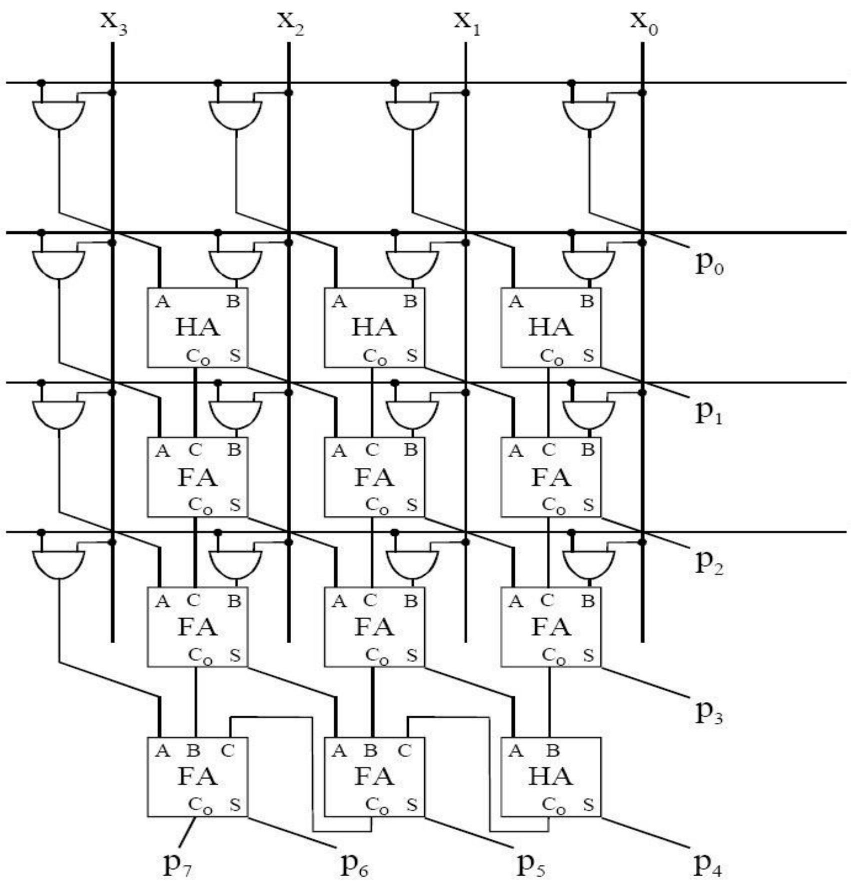
It is composed of:

- N^2 AND
- N*(N-2) Full Adder
- N Half Adder

As for the RCA we set N = 8

We select this kind of multiplier because it can perform multiplication in O(n) where n is the number of bits of our numbers.

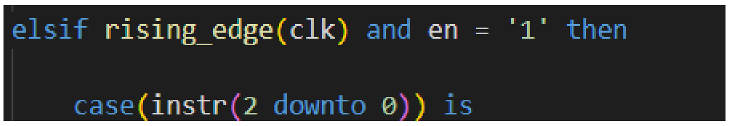
With this implementation we are able to have 8-bits*8-bits multiplications which cannot be stored in the CPU register, so we made a slight change in the scheme below: we still have implemented such logic so the CPU Is able to make 8-bit*1-bit multiplications but we added an overflow for results over 8-bit and it is used to set the Status Register.



Parallel Multiplier Block Diagram

VHDL AND TEST-PLAN

The main .vhd file is simple_cpu.vhd in which we have written the code for the sequential process.



```
elsif rising_edge(clk) and en = '1' then

    case(instr(2 downto 0)) is
```

At the rising edge of the clock, we check which operation the CPU has to perform, this is done with the **case (instr (2 downto 0))**. Last 3 bits identify the type of operation.

Operations can take more than one clock, so we introduced the signal **stato** for storing the phase of the operation, basing on this value operations are performed rather than others.

For the operations involving the usage of registers we check which of the four registers has been specified in the instruction.

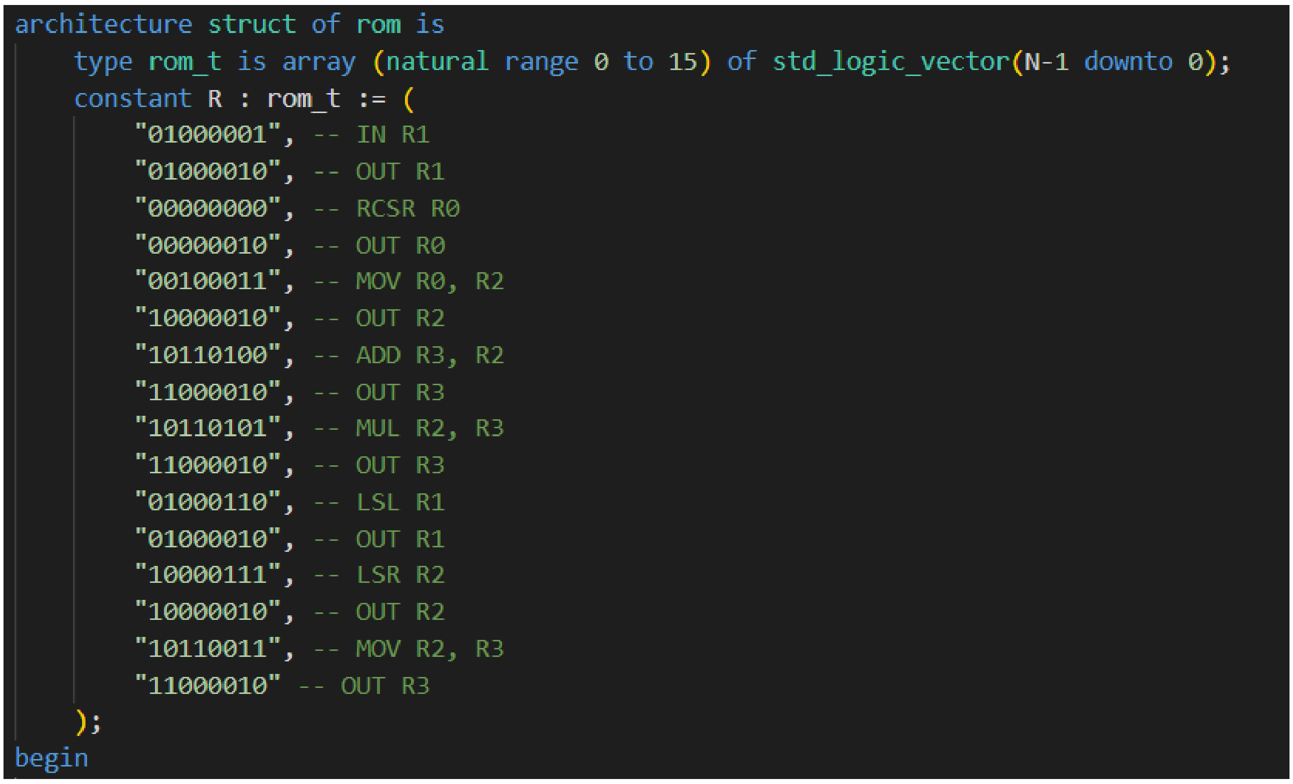
For the operations ADD and MUL we used two combinational networks, for the first the **ripple carry adder** while for the second we implemented a **parallel multiplier**.

Operations are taken from a **16x8-bit ROM**.

After the RTL description we move into the test phase.

Our test-plan provides a **standard case test** where we made the CPU do all the operations without creating any limit case situation.

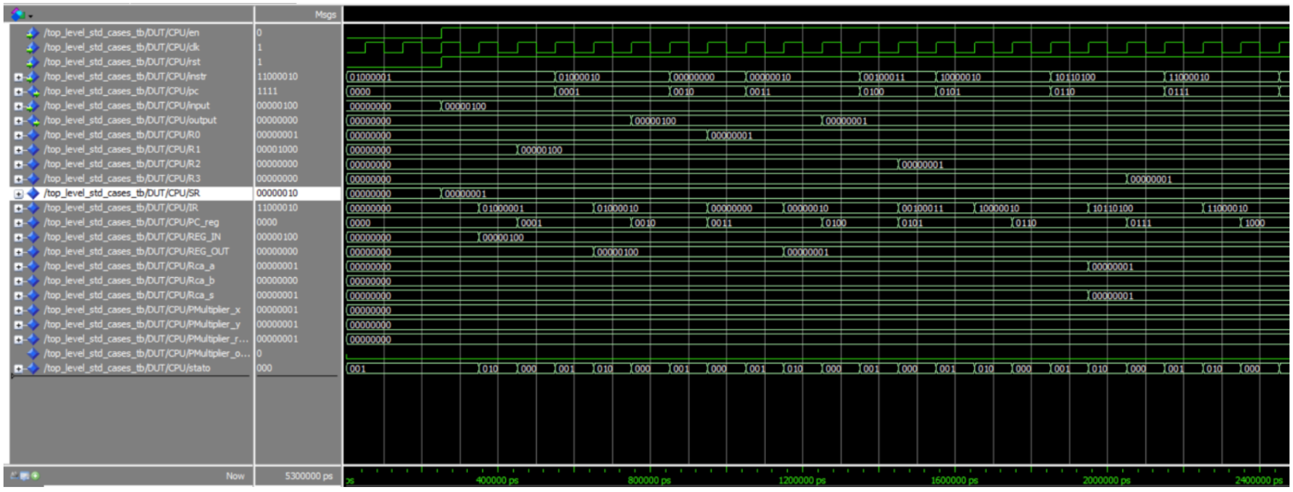
This test aims to verify that the cpu works in the basic cases.



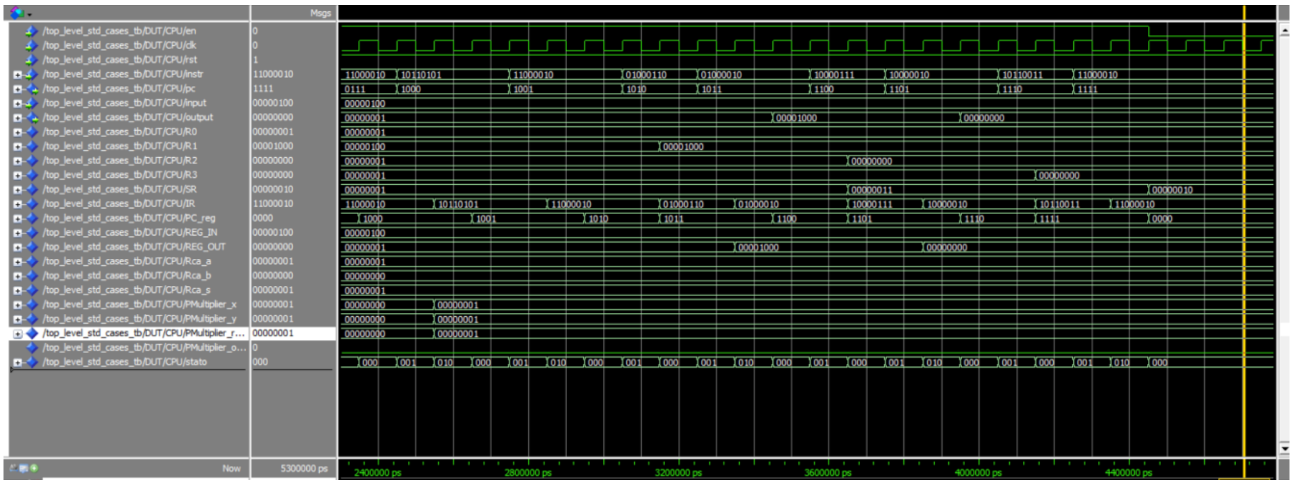
```
architecture struct of rom is
    type rom_t is array (natural range 0 to 15) of std_logic_vector(N-1 downto 0);
    constant R : rom_t := (
        "01000001", -- IN R1
        "01000010", -- OUT R1
        "00000000", -- RCSR R0
        "00000010", -- OUT R0
        "00100011", -- MOV R0, R2
        "10000010", -- OUT R2
        "10110100", -- ADD R3, R2
        "11000010", -- OUT R3
        "10110101", -- MUL R2, R3
        "11000010", -- OUT R3
        "01000110", -- LSL R1
        "01000010", -- OUT R1
        "10000111", -- LSR R2
        "10000010", -- OUT R2
        "10110011", -- MOV R2, R3
        "11000010" -- OUT R3
    );
begin
```

Operations performed in the standard cases test

We got this behavior in the simulation which is correct.

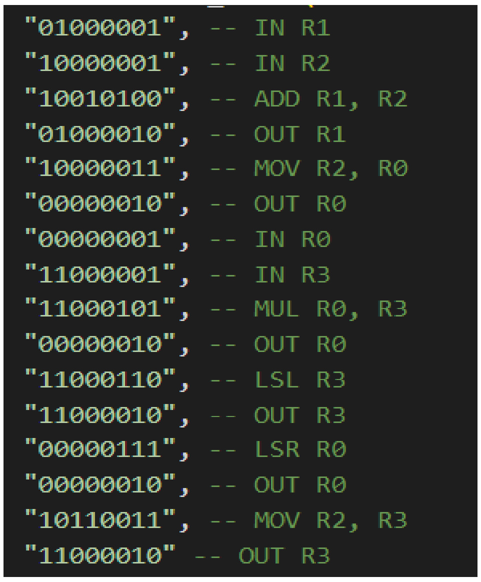


```
/top_level_std_cases_tb/DUT/CPU/en                 0
/top_level_std_cases_tb/DUT/CPU/clk                1
/top_level_std_cases_tb/DUT/CPU/rst                1
/top_level_std_cases_tb/DUT/CPU/instr              11000010
/top_level_std_cases_tb/DUT/CPU/pc                 1111
/top_level_std_cases_tb/DUT/CPU/input              00000100
/top_level_std_cases_tb/DUT/CPU/output             00000000
/top_level_std_cases_tb/DUT/CPU/R0                 00000001
/top_level_std_cases_tb/DUT/CPU/R1                 00001000
/top_level_std_cases_tb/DUT/CPU/R2                 00000000
/top_level_std_cases_tb/DUT/CPU/R3                 00000000
/top_level_std_cases_tb/DUT/CPU/SR                 00000010
/top_level_std_cases_tb/DUT/CPU/IR                 11000010
/top_level_std_cases_tb/DUT/CPU/PC_reg             0000
/top_level_std_cases_tb/DUT/CPU/REG_IN             00000100
/top_level_std_cases_tb/DUT/CPU/REG_OUT            00000000
/top_level_std_cases_tb/DUT/CPU/Rca_a              00000001
/top_level_std_cases_tb/DUT/CPU/Rca_b              00000000
/top_level_std_cases_tb/DUT/CPU/Rca_s              00000001
/top_level_std_cases_tb/DUT/CPU/PMultiplier_x      00000001
/top_level_std_cases_tb/DUT/CPU/PMultiplier_y      00000001
/top_level_std_cases_tb/DUT/CPU/PMultiplier_r...   00000001
/top_level_std_cases_tb/DUT/CPU/PMultiplier_o...   0
/top_level_std_cases_tb/DUT/CPU/stato              000
Now                                                5300000 ps
```



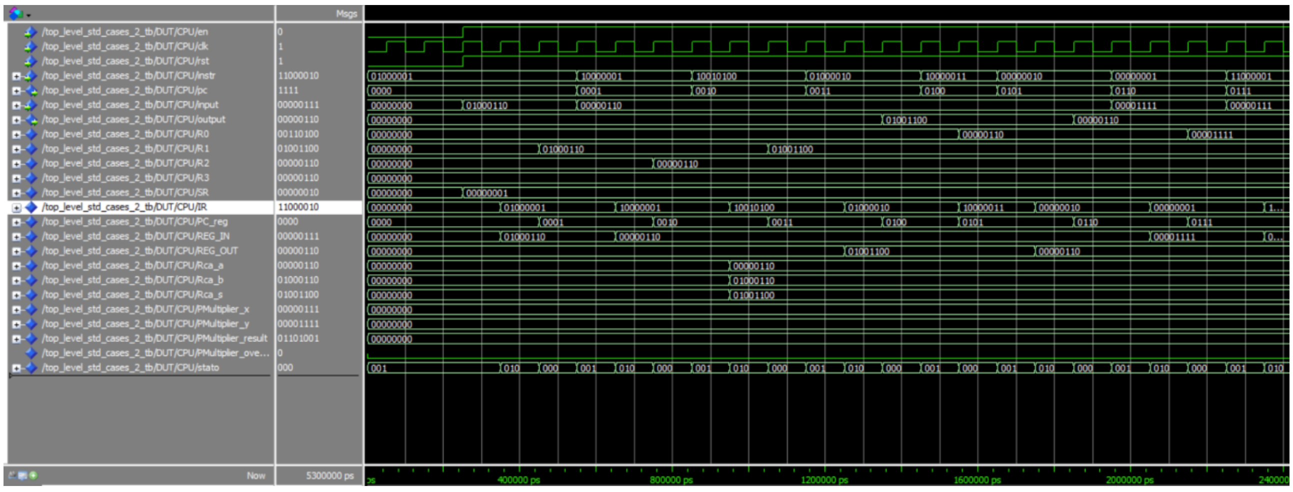
```
/top_level_std_cases_tb/DUT/CPU/en                 0
/top_level_std_cases_tb/DUT/CPU/clk                0
/top_level_std_cases_tb/DUT/CPU/rst                1
/top_level_std_cases_tb/DUT/CPU/instr              11000010
/top_level_std_cases_tb/DUT/CPU/pc                 1111
/top_level_std_cases_tb/DUT/CPU/input              00000100
/top_level_std_cases_tb/DUT/CPU/output             00000000
/top_level_std_cases_tb/DUT/CPU/R0                 00000001
/top_level_std_cases_tb/DUT/CPU/R1                 00001000
/top_level_std_cases_tb/DUT/CPU/R2                 00000000
/top_level_std_cases_tb/DUT/CPU/R3                 00000000
/top_level_std_cases_tb/DUT/CPU/SR                 00000010
/top_level_std_cases_tb/DUT/CPU/IR                 11000010
/top_level_std_cases_tb/DUT/CPU/PC_reg             0000
/top_level_std_cases_tb/DUT/CPU/REG_IN             00000100
/top_level_std_cases_tb/DUT/CPU/REG_OUT            00000000
/top_level_std_cases_tb/DUT/CPU/Rca_a              00000001
/top_level_std_cases_tb/DUT/CPU/Rca_b              00000000
/top_level_std_cases_tb/DUT/CPU/Rca_s              00000001
/top_level_std_cases_tb/DUT/CPU/PMultiplier_x      00000001
/top_level_std_cases_tb/DUT/CPU/PMultiplier_y      00000001
/top_level_std_cases_tb/DUT/CPU/PMultiplier_r...   00000001
/top_level_std_cases_tb/DUT/CPU/PMultiplier_o...   0
/top_level_std_cases_tb/DUT/CPU/stato              000
Now                                                5300000 ps
```

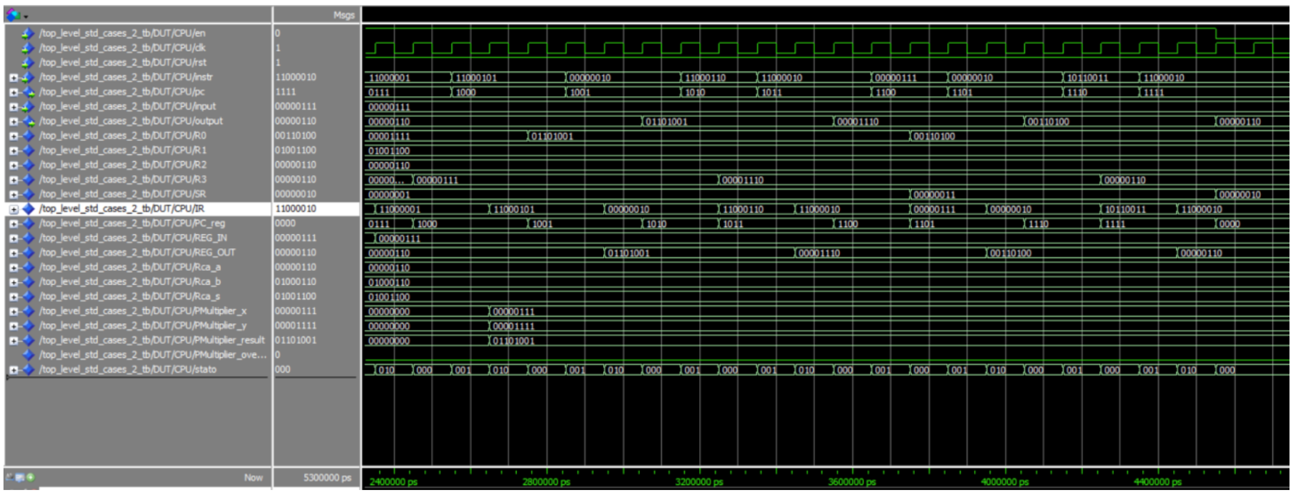
We did another test in standard cases to make our test-plan more accurate.



```
"01000001", -- IN R1
"10000001", -- IN R2
"10010100", -- ADD R1, R2
"01000010", -- OUT R1
"10000011", -- MOV R2, R0
"00000010", -- OUT R0
"00000001", -- IN R0
"11000001", -- IN R3
"11000101", -- MUL R0, R3
"00000010", -- OUT R0
"11000110", -- LSL R3
"11000010", -- OUT R3
"00000111", -- LSR R0
"00000010", -- OUT R0
"10110011", -- MOV R2, R3
"11000010"  -- OUT R3
```

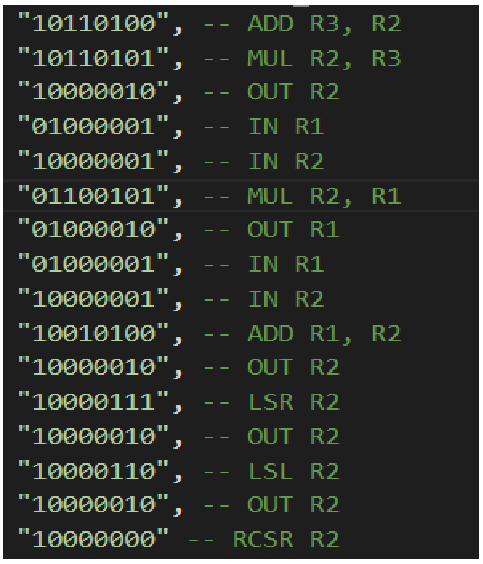
Instructions performed in the second standard cases test.





Also this second test went well.

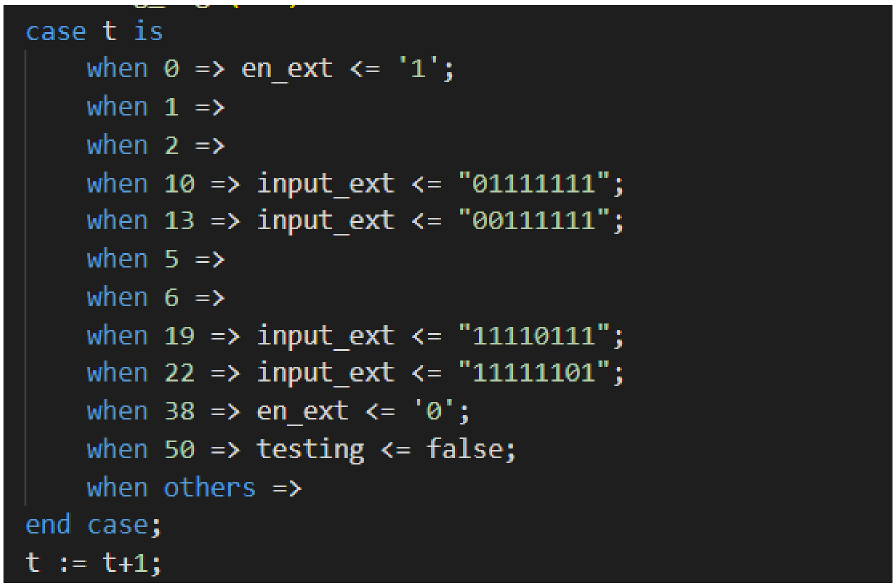
Then we did the **limit cases test** in which we forced all the possible situations that could create errors, such as overflows or inputs at zero.



```
"10110100", -- ADD R3, R2
"10110101", -- MUL R2, R3
"10000010", -- OUT R2
"01000001", -- IN R1
"10000001", -- IN R2
"01100101", -- MUL R2, R1
"01000010", -- OUT R1
"01000001", -- IN R1
"10000001", -- IN R2
"10010100", -- ADD R1, R2
"10000010", -- OUT R2
"10000111", -- LSR R2
"10000010", -- OUT R2
"10000110", -- LSL R2
"10000010", -- OUT R2
"10000000" -- RCSR R2
```

Operations performed in the limit cases test

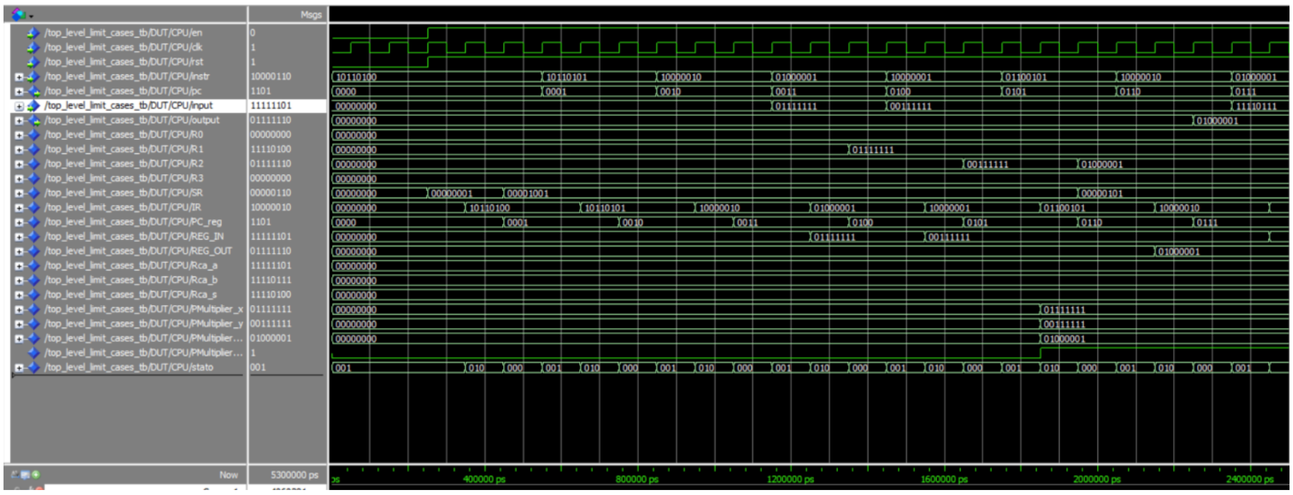
The first ADD and MUL have inputs at zero (from the reset) so we tested that in this case everything works correctly in particular that the zero flag was set.



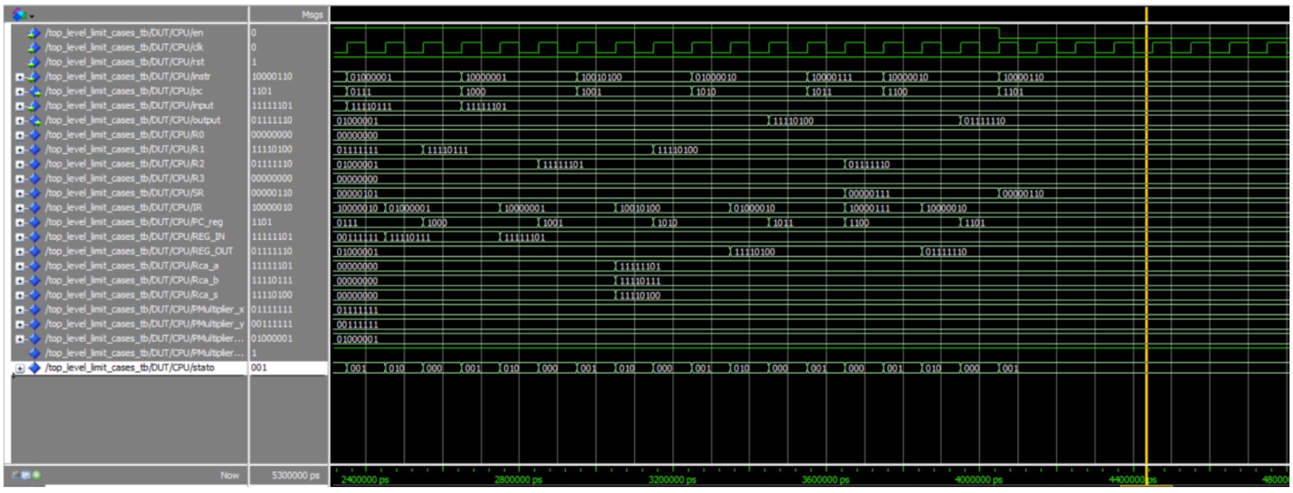
```
case t is
    when 0 => en_ext <= '1';
    when 1 =>
    when 2 =>
    when 10 => input_ext <= "01111111";
    when 13 => input_ext <= "00111111";
    when 5 =>
    when 6 =>
    when 19 => input_ext <= "11110111";
    when 22 => input_ext <= "11111101";
    when 38 => en_ext <= '0';
    when 50 => testing <= false;
    when others =>
end case;
t := t+1;
```

Limit cases testbench

We gave input so that overflow was generated, also we test that switching off the CPU (setting en = 0) before than the operations were finished it effectively stop to work.



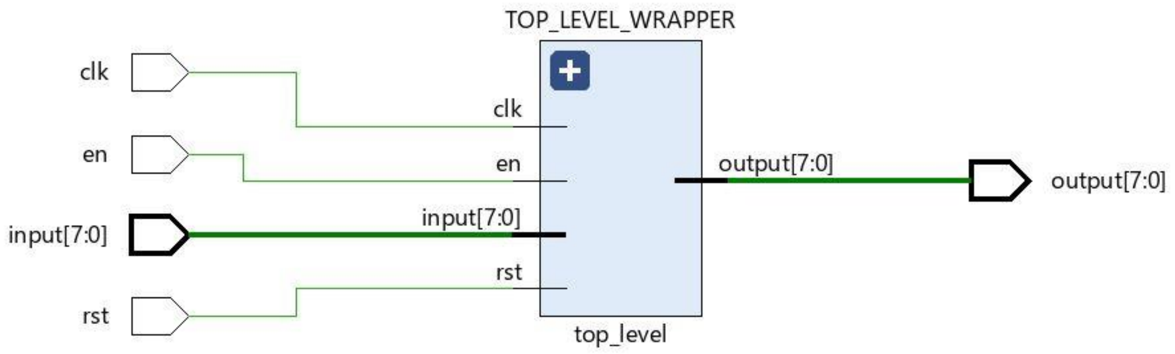
```
                                                  Msgs
/top_level_limit_cases_tb/DUT/CPU/en              0
/top_level_limit_cases_tb/DUT/CPU/clk             1
/top_level_limit_cases_tb/DUT/CPU/rst             1
/top_level_limit_cases_tb/DUT/CPU/instr           10000110
/top_level_limit_cases_tb/DUT/CPU/pc              1101
/top_level_limit_cases_tb/DUT/CPU/input           11111101
/top_level_limit_cases_tb/DUT/CPU/output          01111110
/top_level_limit_cases_tb/DUT/CPU/R0              00000000
/top_level_limit_cases_tb/DUT/CPU/R1              11110100
/top_level_limit_cases_tb/DUT/CPU/R2              01111110
/top_level_limit_cases_tb/DUT/CPU/R3              00000000
/top_level_limit_cases_tb/DUT/CPU/SR              00000110
/top_level_limit_cases_tb/DUT/CPU/IR              10000010
/top_level_limit_cases_tb/DUT/CPU/PC_reg          1101
/top_level_limit_cases_tb/DUT/CPU/REG_IN          11111101
/top_level_limit_cases_tb/DUT/CPU/REG_OUT         01111110
/top_level_limit_cases_tb/DUT/CPU/Rca_a           11111101
/top_level_limit_cases_tb/DUT/CPU/Rca_b           11110111
/top_level_limit_cases_tb/DUT/CPU/Rca_s           11110100
/top_level_limit_cases_tb/DUT/CPU/PMultiplier_x   01111111
/top_level_limit_cases_tb/DUT/CPU/PMultiplier_y   00111111
/top_level_limit_cases_tb/DUT/CPU/PMultiplier...  01000001
/top_level_limit_cases_tb/DUT/CPU/PMultiplier...  1
/top_level_limit_cases_tb/DUT/CPU/stato           001
                                          Now     5300000 ps
```

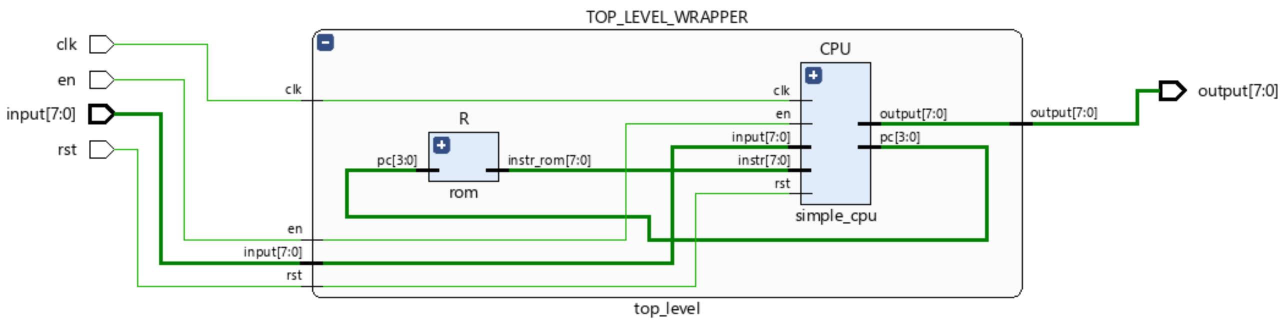


```
                                                  Msgs
/top_level_limit_cases_tb/DUT/CPU/en              0
/top_level_limit_cases_tb/DUT/CPU/clk             0
/top_level_limit_cases_tb/DUT/CPU/rst             1
/top_level_limit_cases_tb/DUT/CPU/instr           10000110
/top_level_limit_cases_tb/DUT/CPU/pc              1101
/top_level_limit_cases_tb/DUT/CPU/input           11111101
/top_level_limit_cases_tb/DUT/CPU/output          01111110
/top_level_limit_cases_tb/DUT/CPU/R0              00000000
/top_level_limit_cases_tb/DUT/CPU/R1              11110100
/top_level_limit_cases_tb/DUT/CPU/R2              01111110
/top_level_limit_cases_tb/DUT/CPU/R3              00000000
/top_level_limit_cases_tb/DUT/CPU/SR              00000110
/top_level_limit_cases_tb/DUT/CPU/IR              10000010
/top_level_limit_cases_tb/DUT/CPU/PC_reg          1101
/top_level_limit_cases_tb/DUT/CPU/REG_IN          11111101
/top_level_limit_cases_tb/DUT/CPU/REG_OUT         01111110
/top_level_limit_cases_tb/DUT/CPU/Rca_a           11111101
/top_level_limit_cases_tb/DUT/CPU/Rca_b           11110111
/top_level_limit_cases_tb/DUT/CPU/Rca_s           11110100
/top_level_limit_cases_tb/DUT/CPU/PMultiplier_x   01111111
/top_level_limit_cases_tb/DUT/CPU/PMultiplier_y   00111111
/top_level_limit_cases_tb/DUT/CPU/PMultiplier...  01000001
/top_level_limit_cases_tb/DUT/CPU/PMultiplier...  1
/top_level_limit_cases_tb/DUT/CPU/stato           001
                                          Now     5300000 ps
```

SYNTHESIS

We used **VIVADO** as a tool for doing the synthesis.





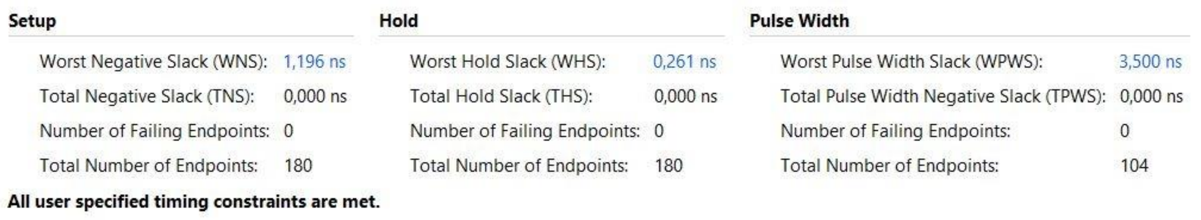
For avoiding the mapping of the I/O of our circuit to the FPGA physical pins we did the synthesis by specifying the out_of_context mode in the synthesis options, this of course led to some warnings.

Before running the synthesis, we added the clock constraint, we set the clock period at 8ns.

We ran the synthesis and in the timing summary we obtained a slack above 1,196ns so the constraints we imposed were met.

Maximum clock frequency can be computed as:

8ns – slack = 6,804ns (146,97 MHz)

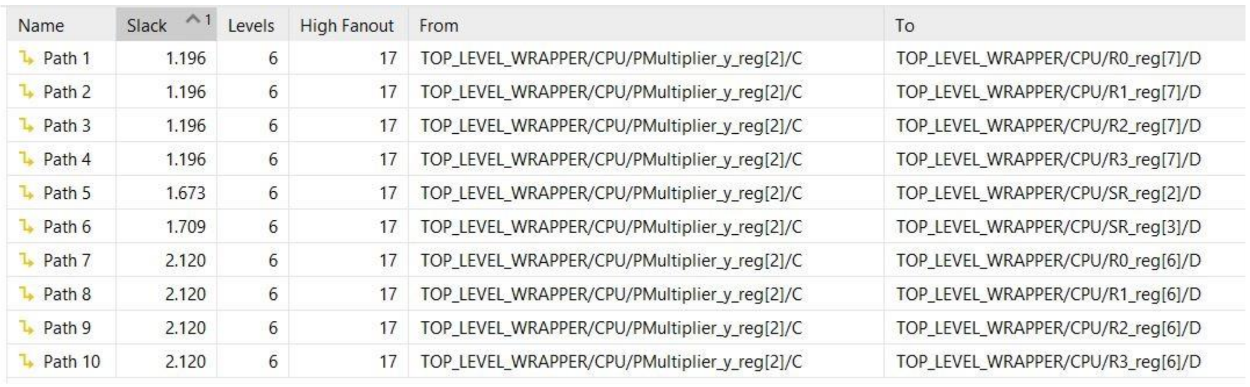


Setup		**Hold**		**Pulse Width**	
Worst Negative Slack (WNS):	1,196 ns	Worst Hold Slack (WHS):	0,261 ns	Worst Pulse Width Slack (WPWS):	3,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	180	Total Number of Endpoints:	180	Total Number of Endpoints:	104

All user specified timing constraints are met.

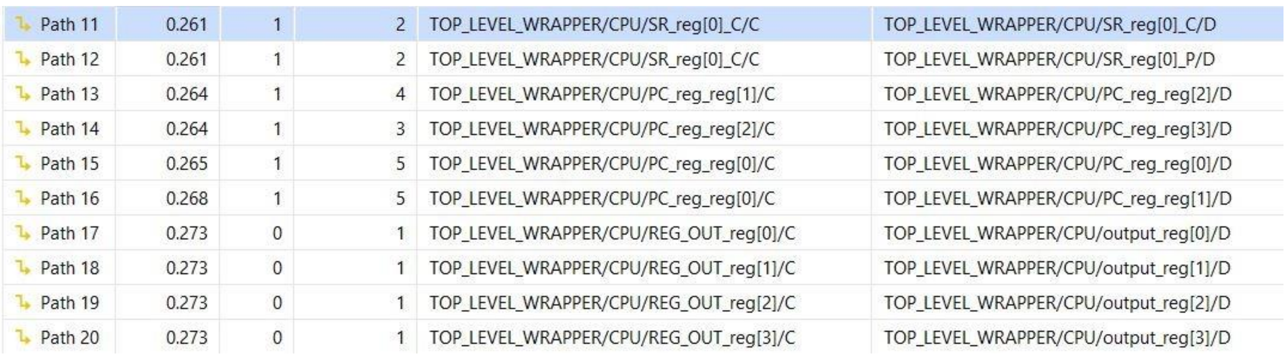
Timing summary

Then we passed to identify the critical path, we checked the set-up violation rule and the hold time violation rule.



Name	Slack ^1	Levels	High Fanout	From	To
Path 1	1.196	6	17	TOP_LEVEL_WRAPPER/CPU/PMultiplier_y_reg[2]/C	TOP_LEVEL_WRAPPER/CPU/R0_reg[7]/D
Path 2	1.196	6	17	TOP_LEVEL_WRAPPER/CPU/PMultiplier_y_reg[2]/C	TOP_LEVEL_WRAPPER/CPU/R1_reg[7]/D
Path 3	1.196	6	17	TOP_LEVEL_WRAPPER/CPU/PMultiplier_y_reg[2]/C	TOP_LEVEL_WRAPPER/CPU/R2_reg[7]/D
Path 4	1.196	6	17	TOP_LEVEL_WRAPPER/CPU/PMultiplier_y_reg[2]/C	TOP_LEVEL_WRAPPER/CPU/R3_reg[7]/D
Path 5	1.673	6	17	TOP_LEVEL_WRAPPER/CPU/PMultiplier_y_reg[2]/C	TOP_LEVEL_WRAPPER/CPU/SR_reg[2]/D
Path 6	1.709	6	17	TOP_LEVEL_WRAPPER/CPU/PMultiplier_y_reg[2]/C	TOP_LEVEL_WRAPPER/CPU/SR_reg[3]/D
Path 7	2.120	6	17	TOP_LEVEL_WRAPPER/CPU/PMultiplier_y_reg[2]/C	TOP_LEVEL_WRAPPER/CPU/R0_reg[6]/D
Path 8	2.120	6	17	TOP_LEVEL_WRAPPER/CPU/PMultiplier_y_reg[2]/C	TOP_LEVEL_WRAPPER/CPU/R1_reg[6]/D
Path 9	2.120	6	17	TOP_LEVEL_WRAPPER/CPU/PMultiplier_y_reg[2]/C	TOP_LEVEL_WRAPPER/CPU/R2_reg[6]/D
Path 10	2.120	6	17	TOP_LEVEL_WRAPPER/CPU/PMultiplier_y_reg[2]/C	TOP_LEVEL_WRAPPER/CPU/R3_reg[6]/D

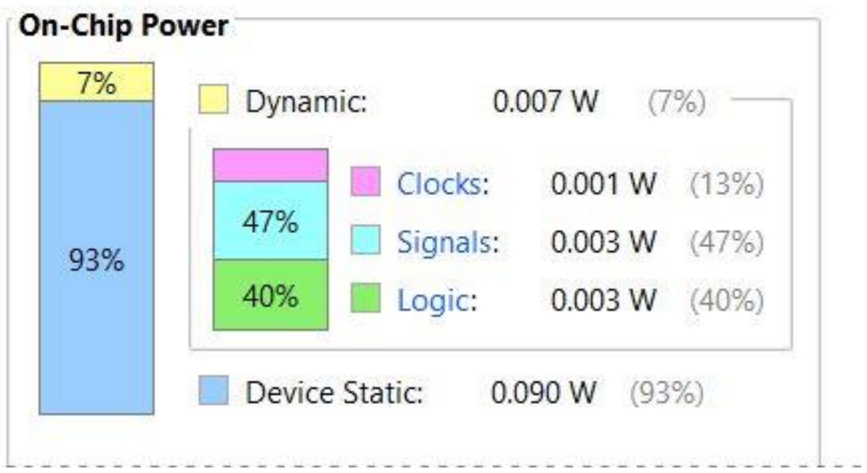
Set-Up violation rule



Path 11	0.261	1	2	TOP_LEVEL_WRAPPER/CPU/SR_reg[0]_C/C	TOP_LEVEL_WRAPPER/CPU/SR_reg[0]_C/D
Path 12	0.261	1	2	TOP_LEVEL_WRAPPER/CPU/SR_reg[0]_C/C	TOP_LEVEL_WRAPPER/CPU/SR_reg[0]_P/D
Path 13	0.264	1	4	TOP_LEVEL_WRAPPER/CPU/PC_reg_reg[1]/C	TOP_LEVEL_WRAPPER/CPU/PC_reg_reg[2]/D
Path 14	0.264	1	3	TOP_LEVEL_WRAPPER/CPU/PC_reg_reg[2]/C	TOP_LEVEL_WRAPPER/CPU/PC_reg_reg[3]/D
Path 15	0.265	1	5	TOP_LEVEL_WRAPPER/CPU/PC_reg_reg[0]/C	TOP_LEVEL_WRAPPER/CPU/PC_reg_reg[0]/D
Path 16	0.268	1	5	TOP_LEVEL_WRAPPER/CPU/PC_reg_reg[0]/C	TOP_LEVEL_WRAPPER/CPU/PC_reg_reg[1]/D
Path 17	0.273	0	1	TOP_LEVEL_WRAPPER/CPU/REG_OUT_reg[0]/C	TOP_LEVEL_WRAPPER/CPU/output_reg[0]/D
Path 18	0.273	0	1	TOP_LEVEL_WRAPPER/CPU/REG_OUT_reg[1]/C	TOP_LEVEL_WRAPPER/CPU/output_reg[1]/D
Path 19	0.273	0	1	TOP_LEVEL_WRAPPER/CPU/REG_OUT_reg[2]/C	TOP_LEVEL_WRAPPER/CPU/output_reg[2]/D
Path 20	0.273	0	1	TOP_LEVEL_WRAPPER/CPU/REG_OUT_reg[3]/C	TOP_LEVEL_WRAPPER/CPU/output_reg[3]/D

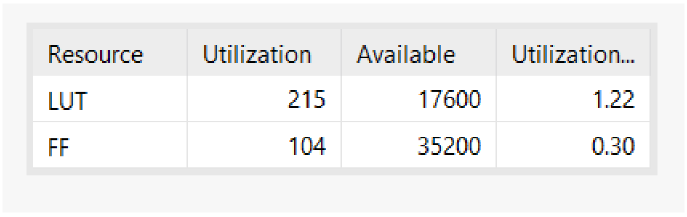
Hold time violation rule.

Finally, we did the power analysis, VIVADO includes a tool for doing this estimation, more than 90% of the power consumption is due to the static power, that's due to the out of context mode, in fact the power consumption related to the I/O is zero.



🟨	Dynamic:	0.007 W	(7%)
🟪	Clocks:	0.001 W	(13%)
🟦	Signals:	0.003 W	(47%)
🟩	Logic:	0.003 W	(40%)
🟦	Device Static:	0.090 W	(93%)

The elements used for the synthesis of our circuit are the followings:



Resource	Utilization	Available	Utilization...
LUT	215	17600	1.22
FF	104	35200	0.30

CONCLUSIONS

We realized the design and the synthesis of a simplified version of a CPU, modern devices are more complex rather than compared to our CPU.

Some main simplifications of our project are:

- Instruction set of only eight instructions.
- Little number of registers
- ROM with only 16 instructions

We notice that clock speed is around 150 MHz which is distant from those of modern CPUs (around 2-4 GHz).

One reason can be that our CPU has been synthesized using FPGA while big companies use full-custom design which led to better performance.