Politecnico di Torino

III Facoltà di Ingegneria

Report Lab3
Integrated Systems Architecture

Master degree in Electrical Engineering

Authors: ISA01

Boeckelen Daniel, Piran Michael, Semino Emanuele

February 21, 2021

Contents

CHAPTER 1

Introduction

The goal of this lab is to design a RISCV-Lite processor.
Data and instructions are represented on 32 bits. The register file contain 32 registers of 32 bits, so we need 5 bits to address them. We have implemented a pipeline processor with 5 stages:
Fetch (IF), Decode (ID), Execute (EX), Memory (MEM) and Write Back (WB).
For simplicity, we have not implemented a Branch prediction unit and a Forward Unit, therefor our processor cannot handle data dependencies and will be less performant, however it will have a smaller area.

Here you can find the code of our design:

`https://github.com/ISAgroup01/Lab3`

CHAPTER 2

Datapath

We have implemented the VHDL architecture of the processor(Figure 2.1). Instructions and data of the entire processor are represented on 32 bits. As requested by the specifications, Instruction

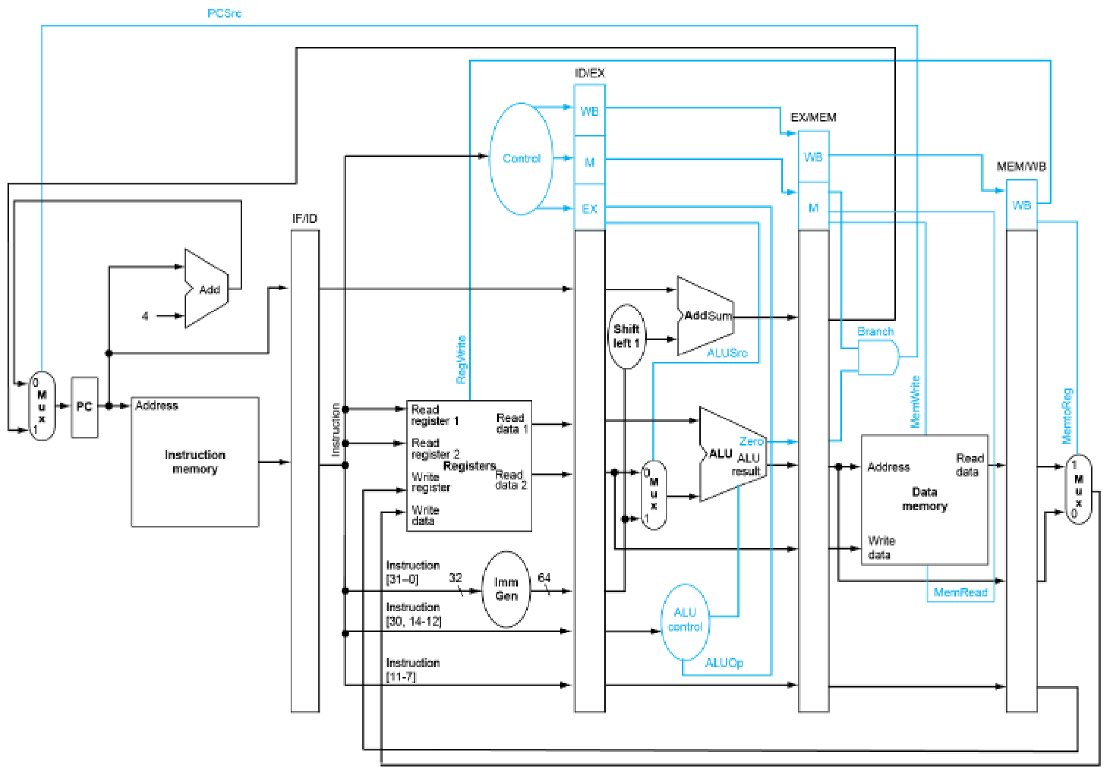


Figure 2.1: General structure

and Data memory are not included in the main architecture, however they are implemented it in the testbench.
Here we have tried to implement most of the single blocks in a Structural way.
Our approach was to describe, where possible, all blocks in a structural manner, so that some parts gain in performance (for instance the branch target adder is built using a CSEA structure instead of using the "+" operand inside a process). Only Register File, ALU and all Filp-flops are implemented as Behavioral.

On Figure 2.2 it is showed how we have organized the VHDL files. Now more details on the sin-

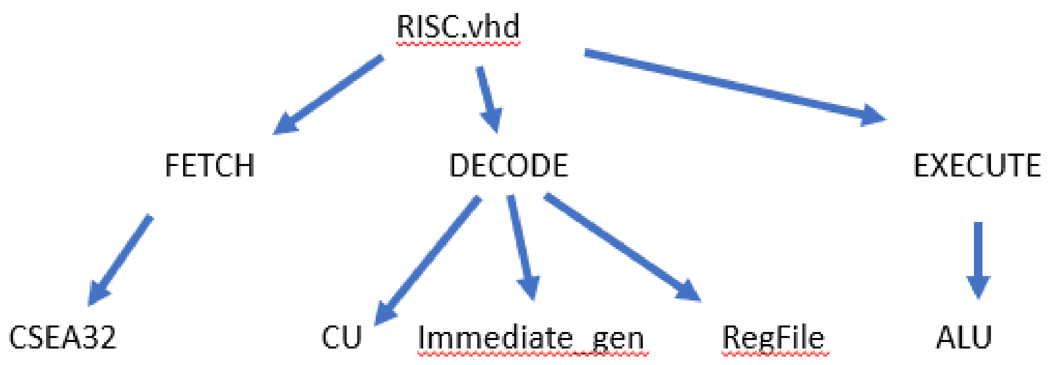


Figure 2.2: General VHDL structure

gular block implementation.

2.1 Fetch

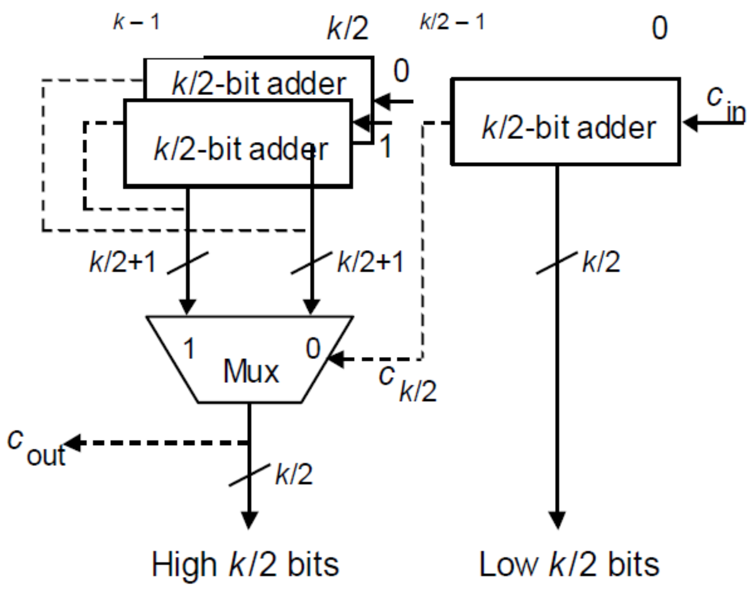


Figure 2.3: CSEA structure

To update the PC we have implemented a Carry Select Adder.
Each singular block is composed of a 8 bit Ripple carry adder, in this way the carry propagation delay is reduced.

2.2 Decode

The register file is described in a behavioral manner and it is composed of 32 registers with 32 bit each.
When the instruction requires to write data, it will be available only at the next clock cycle in the register. On the other hand, a read operation from a register, is immediately available at the corresponding output. Ports ReadRegister1 and ReadRegister2 are used to select (using the address at their inputs) a register for the read operation; same for the WriteRegister port that is used for write operations.
A write enable signal is used to tell when the RF has to write in the register location. The register file has the following ports:

- Register read1;
- Register read2;
- WriteRegister;
- WriteData;
- Read Data1;
- Read Data1;
- RegWrite;
- Reset;
- Clock;

The Immediate_generate block simply reads the instruction and generates a Immediate value (using bit extension). So based on which instruction we are decoding, it organizes a 32 bits Immediate value.

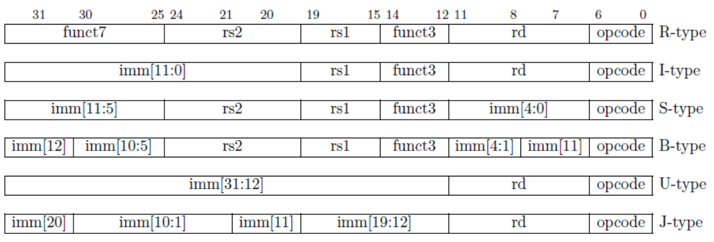

31 30 25 24 21 20 19 15 14 12 11 8 7 6 0
funct7 | rs2 | rs1 | funct3 | rd | opcode | R-type
imm[11:0] | rs1 | funct3 | rd | opcode | I-type
imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode | S-type
imm[12] | imm[10:5] | rs2 | rs1 | funct3 | imm[4:1] | imm[11] | opcode | B-type
imm[31:12] | rd | opcode | U-type
imm[20] | imm[10:1] | imm[11] | imm[19:12] | rd | opcode | J-type

Figure 2.4: Instruction organization

Here we have placed the Control Unit, see next chapter for further details.

2.3 Execute

The adder that creates the new program counter is again implemented with a 32 bits CSEA. Then we have the ALU that receives two inputs values and the Opcode specifies which instruction should be implemented.

In the second version of the processor (that implements the absolute value), we decided to to make the ALU perform the operation, given the proper instruction (that we called ABSV). More details about the instruction ABSV are described in the Control Unit section of this report.

2.4 Memory

This module is described in the RISC top entity. It contains only the AND port that generates the selection signal for the multiplexer on the fetch phase.

2.5 Write back

This module is implemented in the RISC top entity. It contains a four inputs multiplexer and selects between:

- Data memory read;
- Alu result;
- PC + 4 taken from fetch unit;
- PC where we need to jump;

The last two inputs are considered for the instruction AUIPC and JAL. The JAL stores the address of the instruction following the jump (pc+4) into register rd.
So we take the updated PC in the fetch phase and we bring it in the Write Back phase.
AUIPC forms a 32-bit offset from the 20-bit U-immediate and adds this offset to the address of the AUIPC instruction.
Then places the result in register rd. We take the result in the Execute phase, where with the CSEA we create the branch PC.

CHAPTER 3

Control Unit

The Control Unit is the component that directs the operations of the processor.
There are different possibilities for its implementation and we choose the hardwired one:
the control hardware is a finite state machine that switches from a state to another at every clock cycle, generating a sequence of individual bits that represent the control signals (i.e. control words).

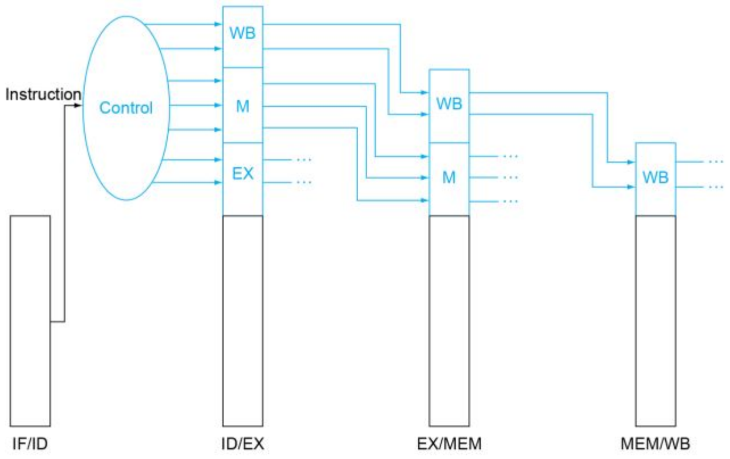
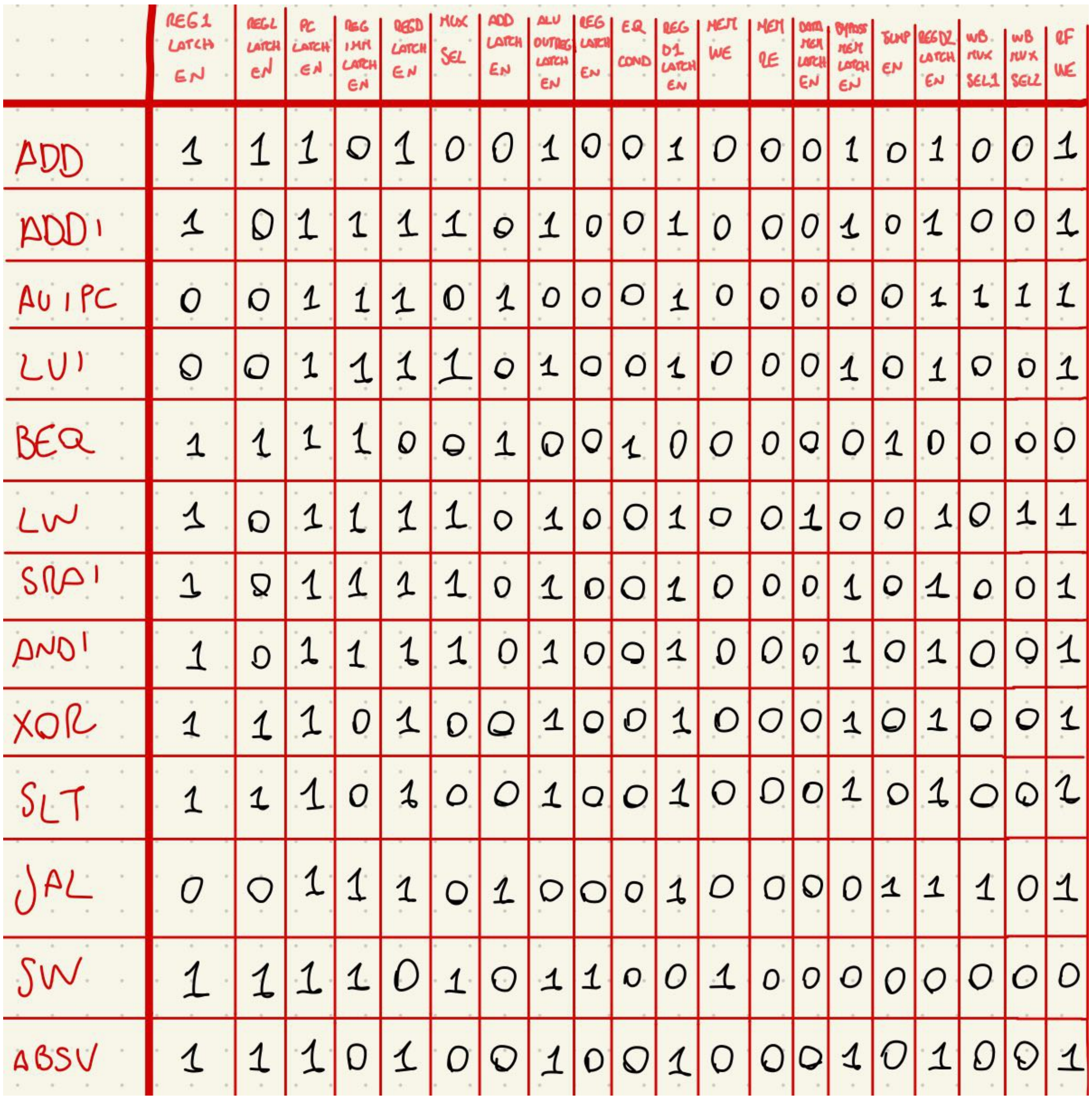


Figure 3.1: General CU structure

3.1 Control Word Generation

The component receives as an input the instruction and based on its Opcode, the control unit decides which control word (CW) has to be sent to the output.
This selection is based on the bits in position 2, 4, 5, 6; and once concatenated together, they identify each instruction uniquely.
These bits are used to access to the matrix that generates the corresponding CW.
Each instruction has its own control signals, and they are the same for instructions belonging to the same class (however, there are some exceptions of signals that belong to the same class but have different control signals):



	REG1 LATCH EN	REG2 LATCH EN	PC LATCH EN	REG IMM LATCH EN	REGD LATCH EN	MUX SEL	ADD LATCH EN	ALU OUTREG LATCH EN	REG LATCH EN	EQ COND	REG D1 LATCH EN	MEM WE	MEM RE	DATA MEM LATCH EN	BYPASS MEM LATCH EN	JUMP EN	REGD2 LATCH EN	WB MUX SEL1	WB MUX SEL2	RF WE
ADD	1	1	1	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1
ADDI	1	0	1	1	1	1	0	1	0	0	1	0	0	0	1	0	1	0	0	1
AUIPC	0	0	1	1	1	0	1	0	0	0	1	0	0	0	0	0	1	1	1	1
LUI	0	0	1	1	1	1	0	1	0	0	1	0	0	0	1	0	1	0	0	1
BEQ	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0
LW	1	0	1	1	1	1	0	1	0	0	1	0	0	1	0	0	1	0	1	1
SRAI	1	0	1	1	1	1	0	1	0	0	1	0	0	0	1	0	1	0	0	1
ANDI	1	0	1	1	1	1	0	1	0	0	1	0	0	0	1	0	1	0	0	1
XOR	1	1	1	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1
SLT	1	1	1	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1
JAL	0	0	1	1	1	0	1	0	0	0	1	0	0	0	0	1	1	1	0	1
SW	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0
ABSV	1	1	1	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1

Figure 3.2: Control signal of each instruction

- I-type (ADDI, SRAI, ANDI);
- LW (I-type);
- R-type (ADD, XOR, SLT);
- B-type (BEQ);
- J-type (JAL);
- S-type (SW);
- AUIPC (U-type);
- LUI (U-type);

3.2 ALU Opcode generation

The second process present in the CU description is the one dedicated to the generation of the code destinated to the ALU, in order to determine its behavior. Its sensitivity list contains the instruction's OPCODE, the instruction's FUNCT and it follows the scheme below:

- I-type -> OPCODE = "0010011" :
  - ADDI -> FUNCT = "000";
  - SRAI -> FUNCT = "101";
  - ANDI -> FUNCT = "111";
- LW (I-type) -> OPCODE = "0000011";
- R-type -¿ OPCODE = "0110011":
  - ADD -> FUNCT = "000";
  - XOR -> FUNCT = "100";
  - SLT -> FUNCT = "010";
- B-type (BEQ) -> OPCODE = "1100011";
- J-type (JAL) -> OPCODE = "1101111";
- S-type (SW) -> OPCODE = "0100011";
- AUIPC (U-type) -> OPCODE = "0010111";
- LUI (U-type) -> OPCODE = "0110111";

3.3 Absolute value (ABSV))

For the absolute value we decided to use the same instruction bit of the SLTIU that is not used in our lite version of the RISC-V. In doing so, the instruction will be in the I-type family and in assembly can be easily written by giving the source register, the destination register and writing the immediate part as 0s.

For instance:
ABSV x2, x16, 0x0
The processor will take the value stored in register 16 of the register file, performs the absolute value operation and stores the new value in the register 2 of the register file.

- I-type -> OPCODE = "0010011" :
  - ADDI -> FUNCT = "000";
  - SRAI -> FUNCT = "101";
  - ANDI -> FUNCT = "111";
  - **ABSV -> FUNCT = "011";**

- LW (I-type) -> OPCODE = ”0000011”;
- R-type -¿ OPCODE = ”0110011”:
  - ADD -> FUNCT = ”000”;
  - XOR -> FUNCT = ”100”;
  - SLT -> FUNCT = ”010”;
- B-type (BEQ) -> OPCODE = ”1100011”;
- J-type (JAL) -> OPCODE = ”1101111”;
- S-type (SW) -> OPCODE = ”0100011”;
- AUIPC (U-type) -> OPCODE = ”0010111”;
- LUI (U-type) -> OPCODE = ”0110111”;

CHAPTER 4

Testbench

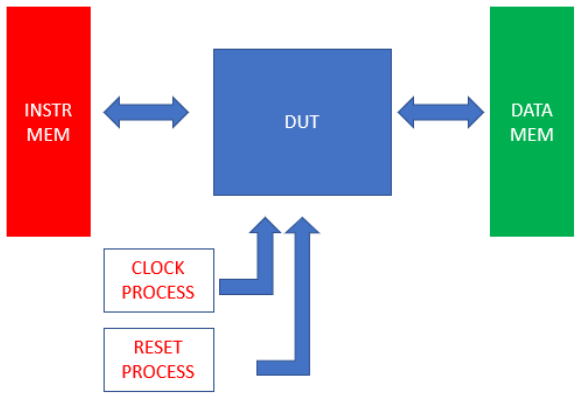


Figure 4.1: Testbench organization

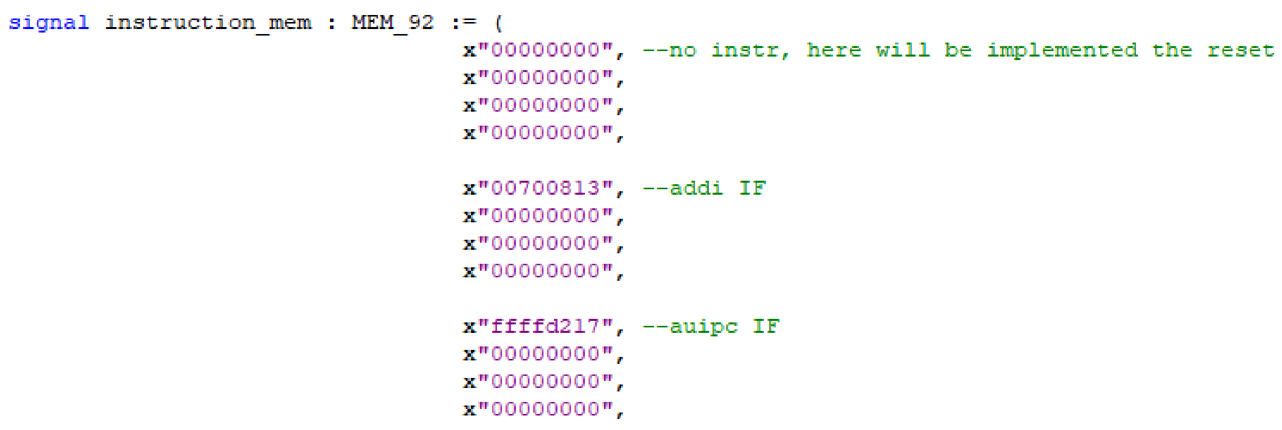
The Clock process generates the clock signal for the entire processor and has a period of 20ns (for the testbench); in this way the processor is tested at 50MHz.
Our goal was to verify if the processor works correctly, without considering the performance, for this reason we have tested it at a low frequency.
The reset process initialize all the component. This signal is active for 15ns then it is disabled.
A good way to start the simulation is to reserve a clock cycle to reset the entire processor (without fetching any instruction in this phase). The signal that gives the reset will be active low for 15ns and then it will go up again (off) until the end of the simulation. After the reset phase, the first instruction in the instruction memory will be fetched and the processor will start to perform its tasks.

Before describing the content of the instruction memory in the vhd file, it was necessary to get some instructions in order to perform the test. To achieve this, an assembly program was loaded in the RARS simulator, returning the instructions (reported in hexadecimal) that have to be written in the instruction memory (*). After getting the list of all instructions, the instruction memory was filled, interleaving each instruction with three "empty lines" (x"00000000") in such a way that the program counter update(PC+4) was respected.

The results are written in the Register file or in the Data memory, depending on which instruction is executed.

For what concerns the Instruction and Data memory we have implemented both like an array of 32 bits.
This because both memory are not so big, we have decided to implement them without resorting to a specific structure.

```
*     in memory settings -> compact, Data at address 0 was selected
```



```
signal instruction_mem : MEM_92 := (
                                  x"00000000", --no instr, here will be implemented the reset
                                  x"00000000",
                                  x"00000000",
                                  x"00000000",

                                  x"00700813", --addi IF
                                  x"00000000",
                                  x"00000000",
                                  x"00000000",

                                  x"ffffd217", --auipc IF
                                  x"00000000",
                                  x"00000000",
                                  x"00000000",
```

Figure 4.2: Memory example

4.1 ABSV testbench

We performed a separate testbench to test the ABSV instruction. We wrote as a first instruction the addi that adds a negative number (like ”-7”) to the content of register zero (that in this case is 0) and then it writes back the result to register sixteen. In this way we simply wrote the value ”-7” in register sixteen. After that the second instruction (ABSV) is performed and take the value of register sixteen and the ALU performs the absolute value operation storing it in register two. From the testbench it results that the new value is 7, therefor the instruction worked as intended.

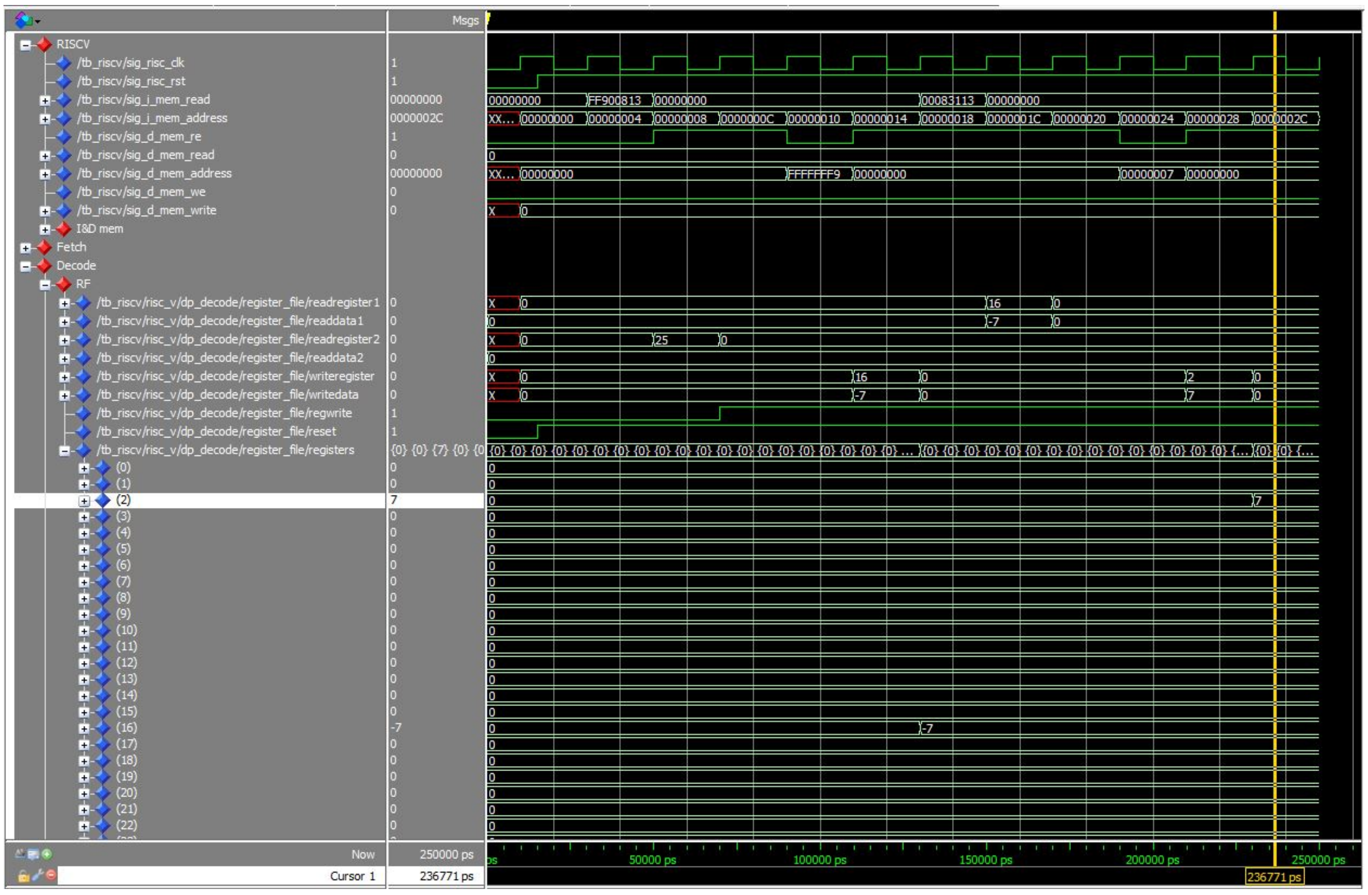

Msgs
RISCV
/tb_riscv/sig_risc_clk
/tb_riscv/sig_risc_rst
/tb_riscv/sig_i_mem_read
/tb_riscv/sig_i_mem_address
/tb_riscv/sig_d_mem_re
/tb_riscv/sig_d_mem_read
/tb_riscv/sig_d_mem_address
/tb_riscv/sig_d_mem_we
/tb_riscv/sig_d_mem_write
I&D mem
Fetch
Decode
RF
/tb_riscv/risc_v/dp_decode/register_file/readregister1
/tb_riscv/risc_v/dp_decode/register_file/readdata1
/tb_riscv/risc_v/dp_decode/register_file/readregister2
/tb_riscv/risc_v/dp_decode/register_file/readdata2
/tb_riscv/risc_v/dp_decode/register_file/writeregister
/tb_riscv/risc_v/dp_decode/register_file/writedata
/tb_riscv/risc_v/dp_decode/register_file/regwrite
/tb_riscv/risc_v/dp_decode/register_file/reset
/tb_riscv/risc_v/dp_decode/register_file/registers
00000000 FF900813 00000000 00083113 00000000
XX... 00000000 00000004 00000008 0000000C 00000010 00000014 00000018 0000001C 00000020 00000024 00000028 0000002C
XX... 00000000 FFFFFFF9 00000000 00000007 00000000
16 0
-7 0
25 0
16 0 2 0
-7 0 7 0
(16) -7
(2) 7
Now 250000 ps
Cursor 1 236771 ps
50000 ps
100000 ps
150000 ps
200000 ps
250000 ps

Figure 4.3: Memory example

CHAPTER 5

Synthesis AND Timing

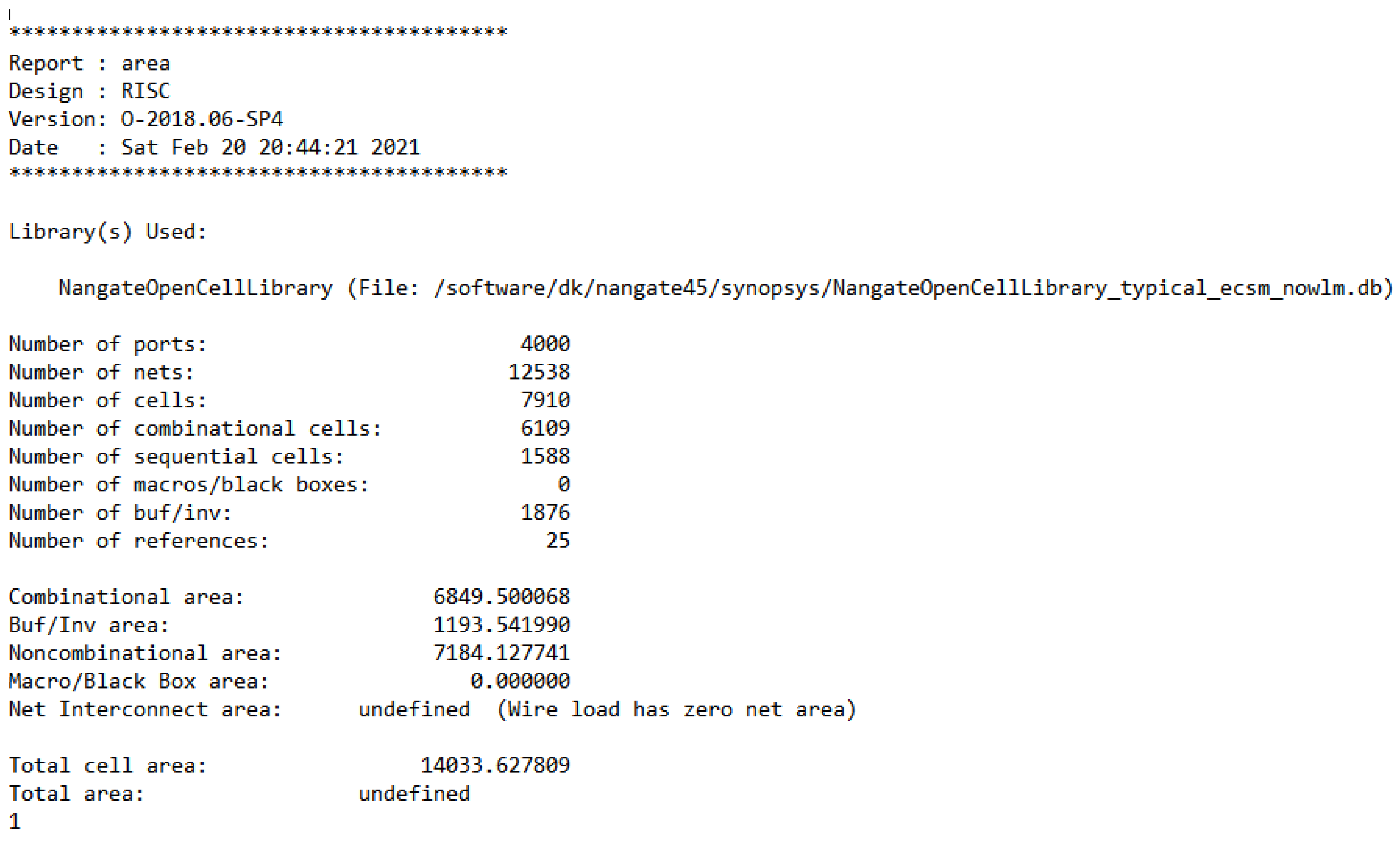
We have synthesized the architecture and we have created the report for the area and the timing. Moreover we made sure that there are no latches in the memory elements of our design, checking the ”elaborate.txt” file.

5.1 Area analysis

As a first step, after the synthesis, we have retrieved the area occupied by our processor.

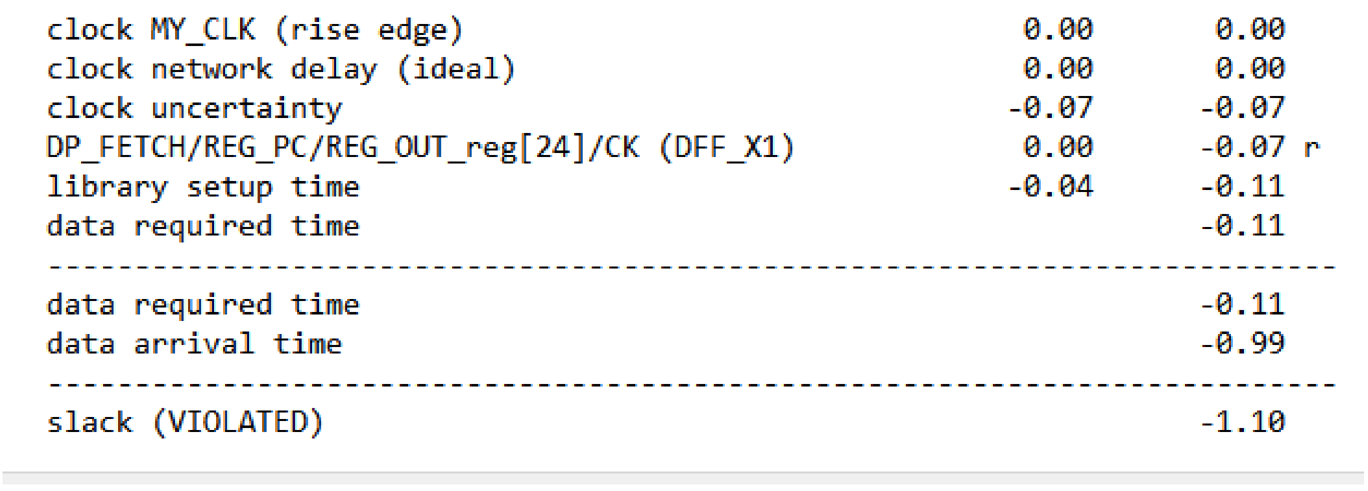
5.2 Timing analysis

We have run the synthesis with a clock period of 0 seconds (to have a really high frequency). So looking the slack,
we can retrieve the maximal clock frequency that our architecture can reach. Then we have applied that constraint and we have run again the synthesis.
Finally we have implemented the routing phase. The result is on Figure 5.7.



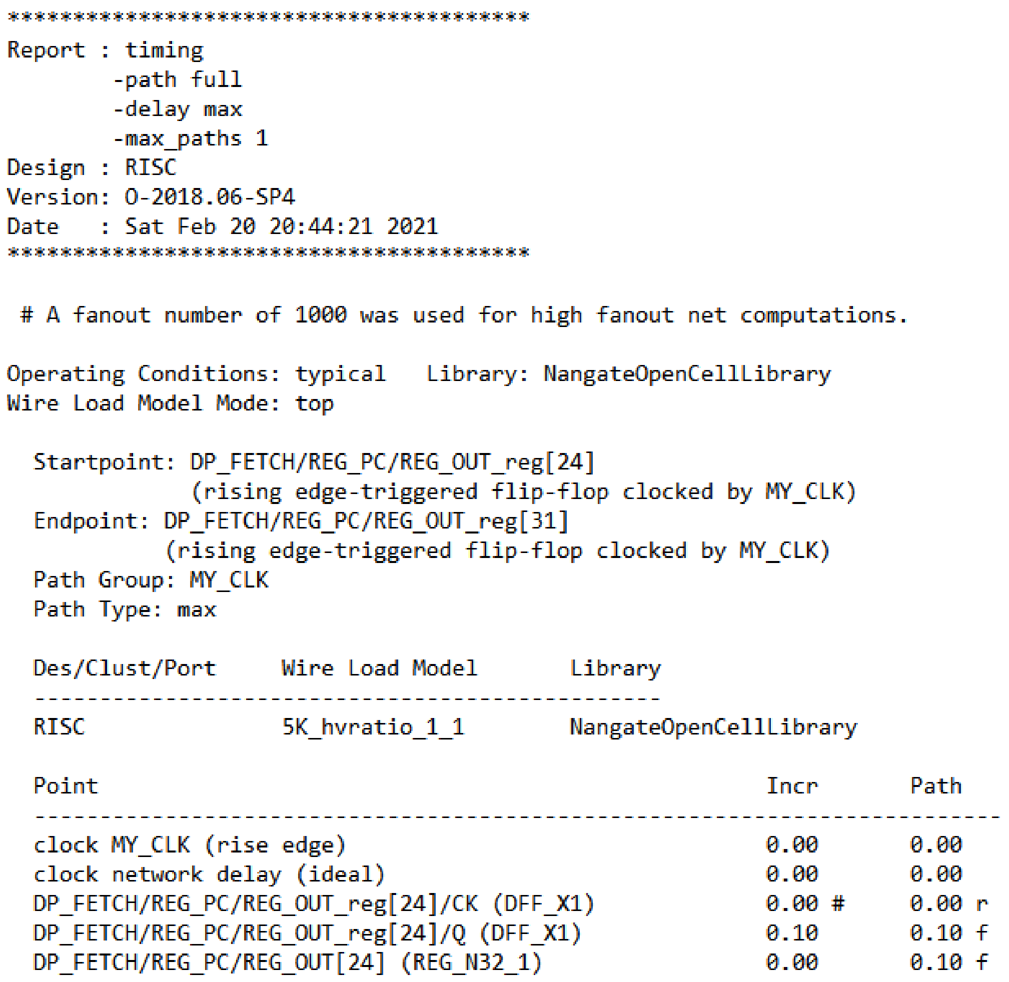
```
****************************************
Report : area
Design : RISC
Version: O-2018.06-SP4
Date   : Sat Feb 20 20:44:21 2021
****************************************

Library(s) Used:

    NangateOpenCellLibrary (File: /software/dk/nangate45/synopsys/NangateOpenCellLibrary_typical_ecsm_nowlm.db)

Number of ports:                         4000
Number of nets:                         12538
Number of cells:                         7910
Number of combinational cells:           6109
Number of sequential cells:              1588
Number of macros/black boxes:               0
Number of buf/inv:                       1876
Number of references:                      25

Combinational area:               6849.500068
Buf/Inv area:                     1193.541990
Noncombinational area:            7184.127741
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:                 14033.627809
Total area:                 undefined
1
```

Figure 5.1: Area report



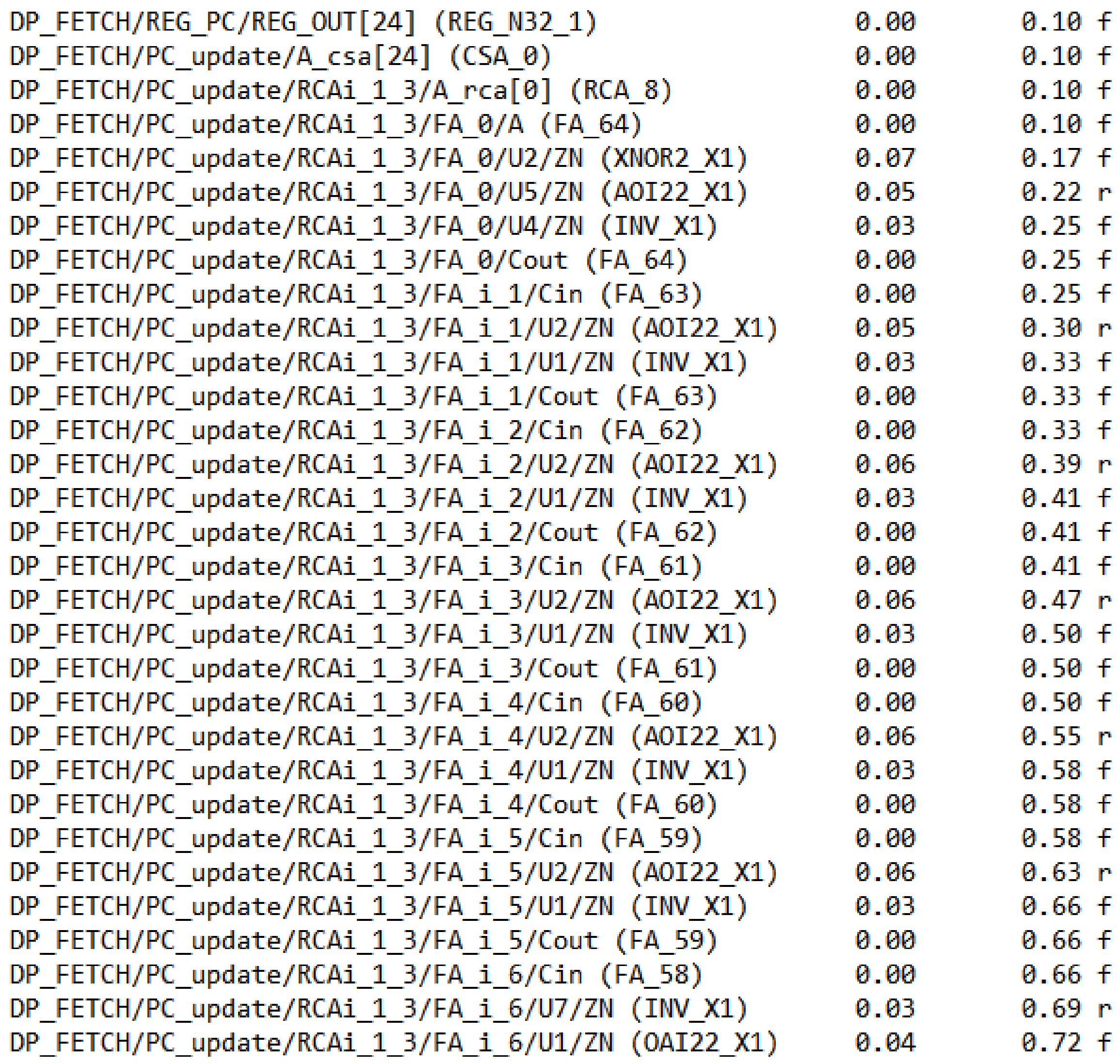
```
clock MY_CLK (rise edge)                         0.00       0.00
clock network delay (ideal)                      0.00       0.00
clock uncertainty                               -0.07      -0.07
DP_FETCH/REG_PC/REG_OUT_reg[24]/CK (DFF_X1)      0.00      -0.07 r
library setup time                              -0.04      -0.11
data required time                                         -0.11
-----------------------------------------------------------------
data required time                                         -0.11
data arrival time                                          -0.99
-----------------------------------------------------------------
slack (VIOLATED)                                           -1.10
```

Figure 5.2: Timing clock equal to 0



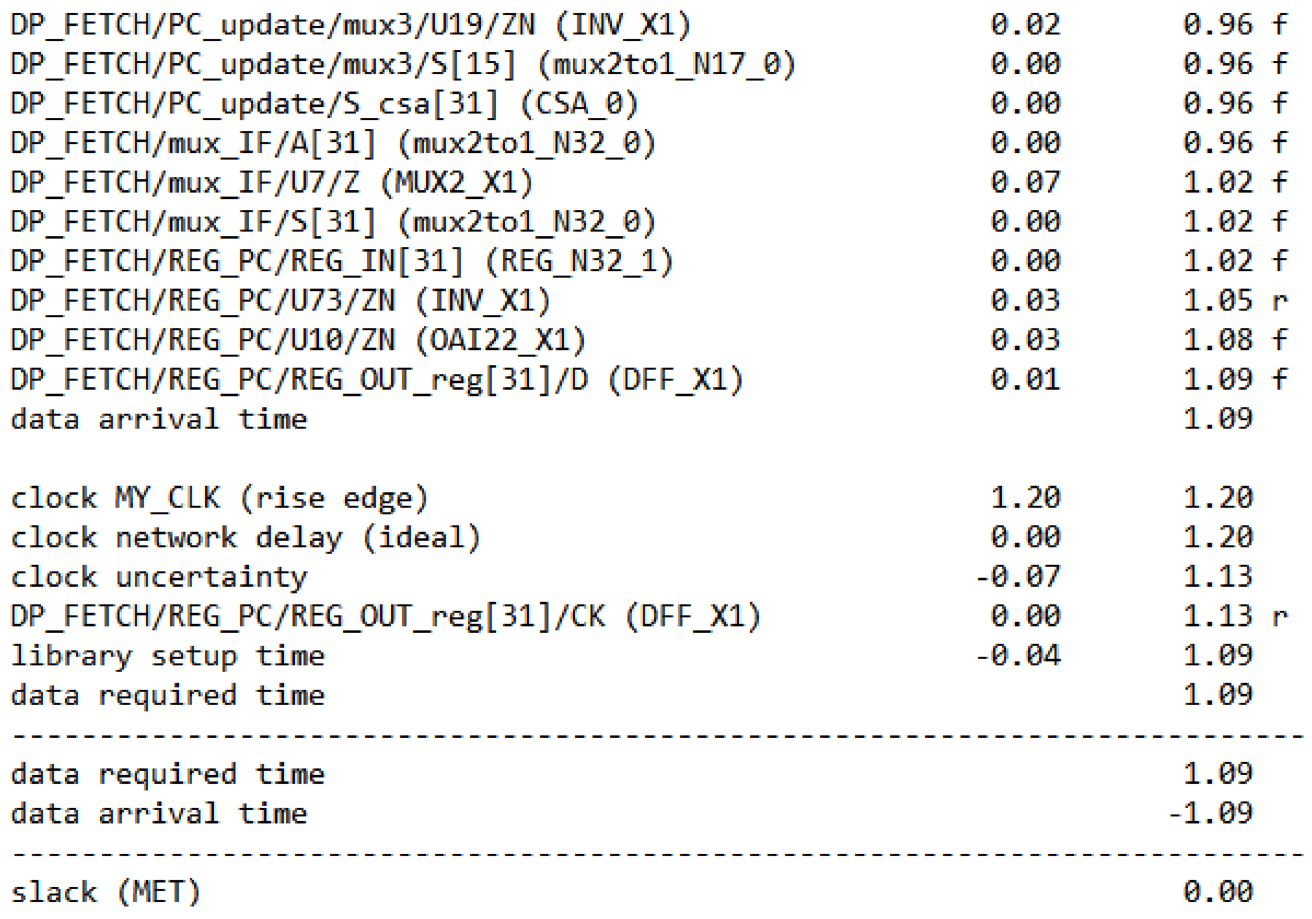
```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : RISC
Version: O-2018.06-SP4
Date   : Sat Feb 20 20:44:21 2021
****************************************

 # A fanout number of 1000 was used for high fanout net computations.

Operating Conditions: typical   Library: NangateOpenCellLibrary
Wire Load Model Mode: top

  Startpoint: DP_FETCH/REG_PC/REG_OUT_reg[24]
              (rising edge-triggered flip-flop clocked by MY_CLK)
  Endpoint: DP_FETCH/REG_PC/REG_OUT_reg[31]
            (rising edge-triggered flip-flop clocked by MY_CLK)
  Path Group: MY_CLK
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  RISC               5K_hvratio_1_1        NangateOpenCellLibrary

  Point                                                   Incr       Path
  --------------------------------------------------------------------------
  clock MY_CLK (rise edge)                                0.00       0.00
  clock network delay (ideal)                             0.00       0.00
  DP_FETCH/REG_PC/REG_OUT_reg[24]/CK (DFF_X1)             0.00 #     0.00 r
  DP_FETCH/REG_PC/REG_OUT_reg[24]/Q (DFF_X1)              0.10       0.10 f
  DP_FETCH/REG_PC/REG_OUT[24] (REG_N32_1)                 0.00       0.10 f
```

Figure 5.3: Timing1



```
DP_FETCH/REG_PC/REG_OUT[24] (REG_N32_1)                       0.00       0.10 f
DP_FETCH/PC_update/A_csa[24] (CSA_0)                          0.00       0.10 f
DP_FETCH/PC_update/RCAi_1_3/A_rca[0] (RCA_8)                  0.00       0.10 f
DP_FETCH/PC_update/RCAi_1_3/FA_0/A (FA_64)                    0.00       0.10 f
DP_FETCH/PC_update/RCAi_1_3/FA_0/U2/ZN (XNOR2_X1)             0.07       0.17 f
DP_FETCH/PC_update/RCAi_1_3/FA_0/U5/ZN (AOI22_X1)             0.05       0.22 r
DP_FETCH/PC_update/RCAi_1_3/FA_0/U4/ZN (INV_X1)               0.03       0.25 f
DP_FETCH/PC_update/RCAi_1_3/FA_0/Cout (FA_64)                 0.00       0.25 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_1/Cin (FA_63)                0.00       0.25 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_1/U2/ZN (AOI22_X1)           0.05       0.30 r
DP_FETCH/PC_update/RCAi_1_3/FA_i_1/U1/ZN (INV_X1)             0.03       0.33 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_1/Cout (FA_63)               0.00       0.33 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_2/Cin (FA_62)                0.00       0.33 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_2/U2/ZN (AOI22_X1)           0.06       0.39 r
DP_FETCH/PC_update/RCAi_1_3/FA_i_2/U1/ZN (INV_X1)             0.03       0.41 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_2/Cout (FA_62)               0.00       0.41 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_3/Cin (FA_61)                0.00       0.41 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_3/U2/ZN (AOI22_X1)           0.06       0.47 r
DP_FETCH/PC_update/RCAi_1_3/FA_i_3/U1/ZN (INV_X1)             0.03       0.50 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_3/Cout (FA_61)               0.00       0.50 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_4/Cin (FA_60)                0.00       0.50 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_4/U2/ZN (AOI22_X1)           0.06       0.55 r
DP_FETCH/PC_update/RCAi_1_3/FA_i_4/U1/ZN (INV_X1)             0.03       0.58 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_4/Cout (FA_60)               0.00       0.58 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_5/Cin (FA_59)                0.00       0.58 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_5/U2/ZN (AOI22_X1)           0.06       0.63 r
DP_FETCH/PC_update/RCAi_1_3/FA_i_5/U1/ZN (INV_X1)             0.03       0.66 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_5/Cout (FA_59)               0.00       0.66 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_6/Cin (FA_58)                0.00       0.66 f
DP_FETCH/PC_update/RCAi_1_3/FA_i_6/U7/ZN (INV_X1)             0.03       0.69 r
DP_FETCH/PC_update/RCAi_1_3/FA_i_6/U1/ZN (OAI22_X1)           0.04       0.72 f
```

Figure 5.4: Timing2



```
DP_FETCH/PC_update/mux3/U19/ZN (INV_X1)                    0.02       0.96 f
DP_FETCH/PC_update/mux3/S[15] (mux2to1_N17_0)              0.00       0.96 f
DP_FETCH/PC_update/S_csa[31] (CSA_0)                       0.00       0.96 f
DP_FETCH/mux_IF/A[31] (mux2to1_N32_0)                      0.00       0.96 f
DP_FETCH/mux_IF/U7/Z (MUX2_X1)                             0.07       1.02 f
DP_FETCH/mux_IF/S[31] (mux2to1_N32_0)                      0.00       1.02 f
DP_FETCH/REG_PC/REG_IN[31] (REG_N32_1)                     0.00       1.02 f
DP_FETCH/REG_PC/U73/ZN (INV_X1)                            0.03       1.05 r
DP_FETCH/REG_PC/U10/ZN (OAI22_X1)                          0.03       1.08 f
DP_FETCH/REG_PC/REG_OUT_reg[31]/D (DFF_X1)                 0.01       1.09 f
data arrival time                                                     1.09

clock MY_CLK (rise edge)                                   1.20       1.20
clock network delay (ideal)                                0.00       1.20
clock uncertainty                                         -0.07       1.13
DP_FETCH/REG_PC/REG_OUT_reg[31]/CK (DFF_X1)                0.00       1.13 r
library setup time                                        -0.04       1.09
data required time                                                    1.09
---------------------------------------------------------------------------
data required time                                                    1.09
data arrival time                                                    -1.09
---------------------------------------------------------------------------
slack (MET)                                                           0.00
```

Figure 5.5: Timing3

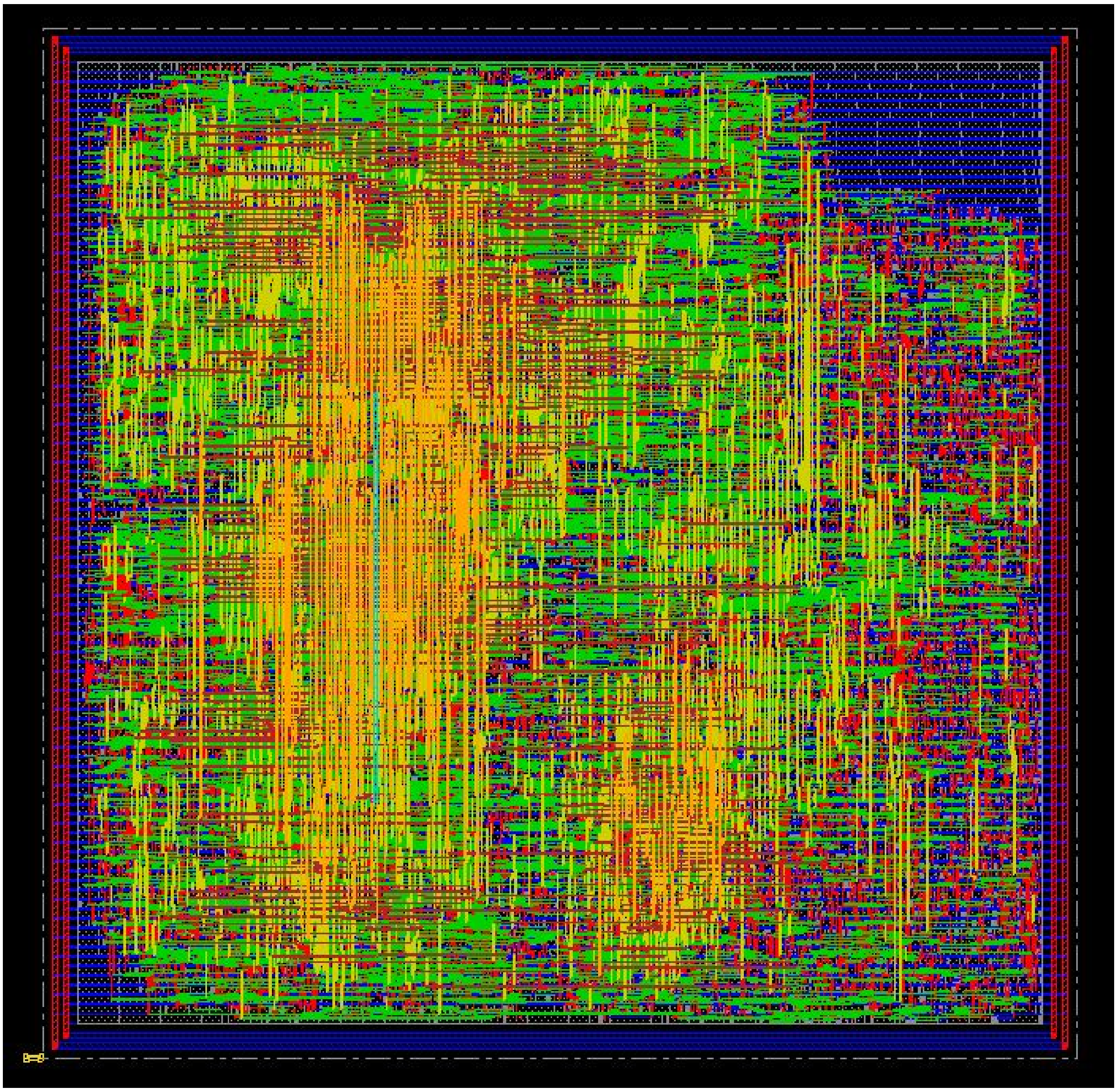


Figure 5.6: Routing of the original design

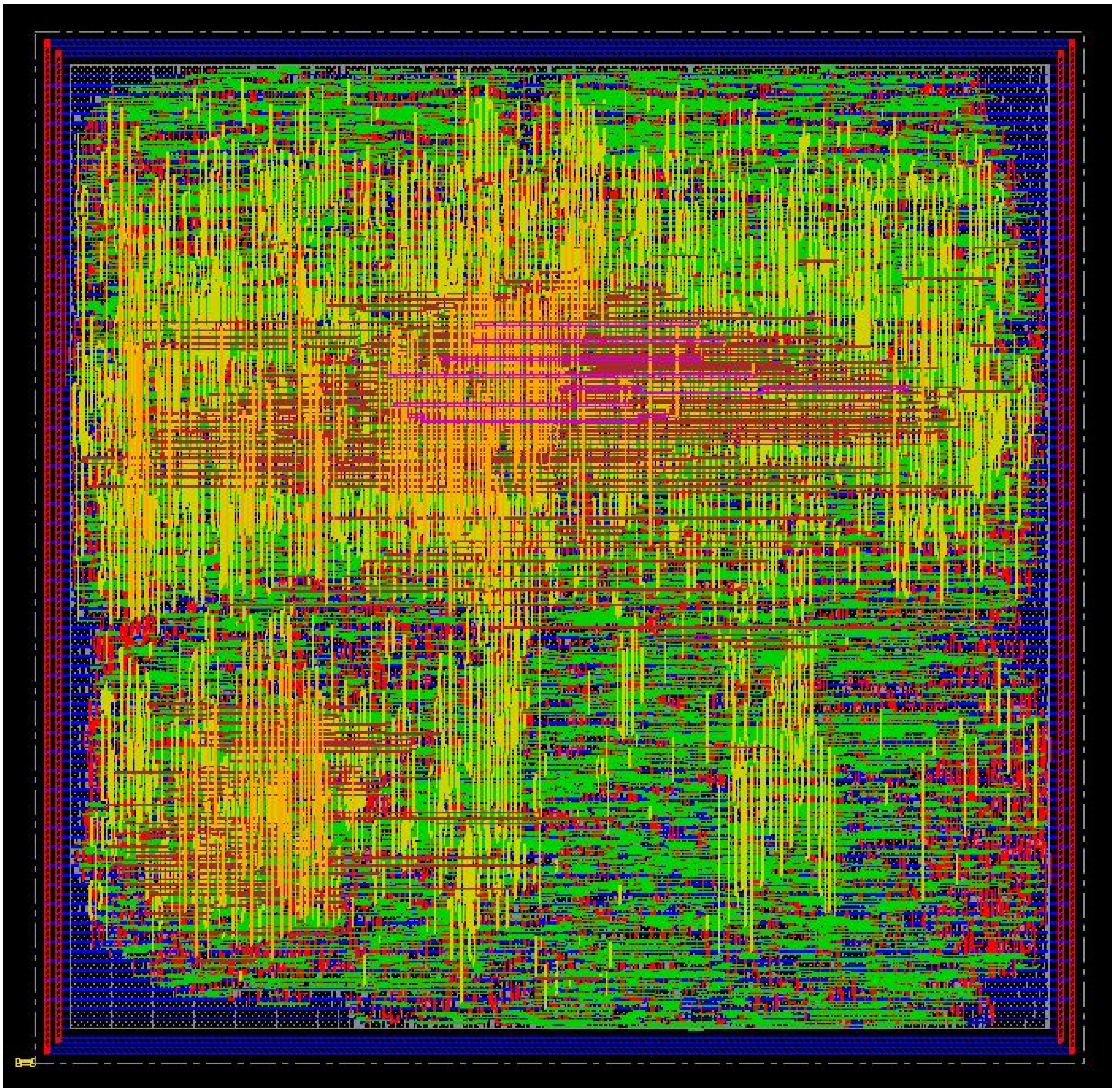


Figure 5.7: Routing of the design with ABSV functionality